LEARNING AND DESIGN METHODOLOGIES FOR EFFICIENT, ROBUST NEURAL NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Priyadarshini Panda

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Prof. Kaushik Roy, Chair

School of Electrical and Computer Engineering

Prof. Anand Raghunathan

School of Electrical and Computer Engineering

Prof. Byunghoo Jung

School of Electrical and Computer Engineering

Prof. Vijay Raghunathan

School of Electrical and Computer Engineering

Approved by:

Dimitrios Peroulis

Head of the School Graduate Program

Dedicated to the two guiding forces in my life

for Papa, even though you passed away like a glittering star falling from the sky, you never let the sky go dark in my life

for Yeshi, you brought me the happiness of an eternity in this lifetime

ACKNOWLEDGMENTS

It is a bitter-sweet feeling writing this last piece of the thesis. As I look back, I see so many euphoric moments intermixed with countless struggles and incredible kindness from so many people who pushed me, made me grow into the person I am today. I know I cannot do full justice to all the goodwill and warmth from all those who have touched my life all these years. Nevertheless, I am going to give it my best shot.

First and foremost, I would like to thank the supreme almighty Lord *Jagannath* for embracing me at my loneliest times and allowing me to view this day. This *odia* girl would not have made it this far had it not been for his blessings.

In Hindu philosophy, a *guru* or teacher is held equivalent to god. Thus, it is only befitting that the next person I would like to thank is my advisor, Prof. Kaushik Roy. From a confused Ph.D. student to an individual who is going to become a faculty, Prof. Roy has been instrumental in this transformation. His research vision, inspiring academic insights and unfailing guidance has resulted in this dissertation. While I am grateful for his invaluable advice, I am indebted to the fact that he let me decide the fate of my Ph.D. He gave me complete freedom to explore diverse ideas and concepts. While he steered me towards many successes during this exploration, he also let me have my share of failures and taught me to own them and learn from them. This gave me the resilience and the confidence to face the turbulence of academic life and allowed me to discover my strengths. His enthusiasm to try something *cool and exciting* has constantly motivated me to set the bar high for my research goals. I believe his enthusiasm will be my greatest inheritance. I also thank him for his countless pieces of wisdom and a guidance checklist (if i may say) on 'how to lead a better & peaceful life' that he gave me during our lunch and coffee discussions. He has been a true

friend, philosopher and guide, and I feel honored and privileged to have worked with him.

Next, I would like to express my sincerest regards for Prof. Anand Raghunathan for infusing new enthusiasm and vigor into my research by giving me an opportunity to work with him in the very beginning of my Ph.D. career. I consider myself really fortunate to have interacted with him. His passion for teaching and mentorship is absolutely amazing and inspiring. I am also very thankful to Prof. Vijay Raghunathan for his positive feedback and mentoring during my Ph.D. His irreplaceable advice to be bold and ambitious in research and lead a healthy work-life balance will continue to guide me throughout my life. I also would like to thank Prof. Byunghoo Jung for his expert advice and encouragement over these years. I feel extremely blessed to have been surrounded by such great mentors and I thank everyone for kindly serving on my doctoral committee.

I would like to express my deepest gratitude to Prof. Sumeet Gupta who contributed significantly towards my career in the last few months. I feel fortunate to have worked with him and have gained so much technical and non-technical advice on academia from him. I am also thankful to Prof. Shriram Ramanathan, a great mentor, who helped me understand a completely new topic on quantum devices. It provided me with new insights and possibilities to explore in my research and contributed significantly to the shaping of this dissertation. I also thank Prof. Shreyas Sen for his help and guidance during these years. I would like to express my deepest regards for all my seniors: Dr. Swagath Venkataramani, Dr. Kon-Woo Kwon, Dr. Woo-Suhl Cho and Prof. Deliang Fan who helped me get started as a Ph.D. student and gave me directions that shaped my fundamentals.

I also thank all current members and alumni from Nanoelectronics Research Lab for being there for me, be it for hot research discussions or sharing lighter moments. I especially thank, Dr. Akhilesh Jaiswal, Dr. Parami Wijesinghe, Gopalakrishnan Srinivasan, Chamika Liyanagedara, Aayush Ankit, Indranil Chakraborthy, Deboleena Roy, and others who provided a synergistic and fun lab environment to work in.

I would also like to convey my sincerest gratitude to Neuromorphic Computing Lab (Intel) and Dr. Narayan Srinivasa for offering me an internship opportunity during my Ph.D. Narayan was a great mentor from whom I learnt several concepts and insights on designing neuromorphic algorithms. I also thank my undergraduate advisors: Prof. V. K. Chaubey, Prof. R. R. Mishra and late Prof. Champak Baran Das for inspiring and assisting me during my undergraduate research.

On a personal front, I am extremely fortunate to have made great friends with whom I shared most of my joys, happiness, sorrows and disappointments of my life in United States. I am greatly indebted to my dearest friends, Satyabrata Parida, Siddharth Gupta, Shruthi Suresh, Kavya Cherukuri, Ayush Parolia, Shanmugam Palaniappan, Soubhagya Sutar, Devyn Maugel and Adam Dachowicz who became a family to me here at Purdue. I would like to especially acknowledge Murtuza Shergadwala in whom I found my best friend, brother, and my life-guide. Their care and undiminishing support grounded me and provided me perpetual encouragement. I also thank my friends: Ruchir Jain, Sonal Nayak, Vineet Mohapatra, Abinash Mallick and Malavika Pradhan back home for the fondest of memories.

My parents and family. I am forever indebted to them for their love, support and guidance. I thank my late grandmother Sunamani, my late father Jayadev, my mother Babita, my brother Debabrata, my sister-in-law Annie and my niece Niyara for their limitless love. I would also like to thank all my well-wishers who nurtured me into the person I am today.

Finally, I would like to take this moment to thank all my predecessors, each and every woman whose sacrifices and determination broke the *glass ceiling*. I owe everything to those unnamed heroes who enabled me *to dream*. This thesis is a token of my gratitude and appreciation to all of them.

Funding acknowledgements: I would like to acknowedge the funding supprt from Center for Spintronic Materials, Interfaces, and Novel Architectures (C-SPIN), a MARCO and DARPA sponsored StarNet center, the Semiconductor Research Corporation, the National Science Foundation, Intel Corporation, the US DoD Vannevar

Bush Faculty Fellowship, US-UK Distributed Analytics and Information Science International Technology Alliance (DAIS-ITA) and the Center for Brain-inspired Computing Enabling Autonomous Intelligence (C-BRIC) - a Joint University Microelectronics Program (JUMP) center.

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

ABSTRACT

Panda, Priyadarshini Ph.D., Purdue University, August 2019. Learning and Design Methodologies for Efficient, Robust Neural Networks. Major Professor: Kaushik Roy.

"Can machines think?", the question brought up by Alan Turing, has led to the development of the field of brain-inspired computing, wherein researchers have put substantial effort in building smarter devices and technology that have the potential of human-like understanding. However, there still remains a large (several orders-of-magnitude) power efficiency gap between the human brain and computers that attempt to emulate some facets of its functionality. In this thesis, we present design techniques that exploit the inherent variability in the difficulty of input data and the correlation of characteristic semantic information among inputs to scale down the computational requirements of a neural network with minimal impact on output quality. While large-scale artificial neural networks have achieved considerable success in a range of applications, there is growing interest in more biologically realistic models, such as, Spiking Neural Networks (SNNs), due to their energy-efficient spike based processing capability. We investigate neuroscientific principles to develop novel learning algorithms that can enable SNNs to conduct on-line learning. We developed an auto-encoder based unsupervised learning rule for training deep spiking convolutional networks that yields state-of-the-art results with computationally efficient learning. Further, we propose a novel "learning to forget" rule that addresses the catastrophic forgetting issue predominant with traditional neural computing paradigm and offers a promising solution for real-time lifelong learning without the expensive re-training procedure. Finally, while artificial intelligence grows in this digital age bringing large-scale social disruption, there is a growing security concern in the research community

about the vulnerabilities of neural networks towards adversarial attacks. To that end, we describe discretization-based solutions, that are traditionally used for reducing the resource utilization of deep neural networks, for adversarial robustness. We also propose a novel noise-learning training strategy as an adversarial defense method. We show that implicit generative modeling of random noise with the same loss function used during posterior maximization, improves a model's understanding of the data manifold, furthering adversarial robustness. We evaluated and analyzed the behavior of the noise modeling technique using principal component analysis that yields metrics which can be generalized to all adversarial defenses.

1. INTRODUCTION

The desire to build more intelligent computers is an enduring vision that has inspired advances in the fields of neuroscience, artificial intelligence (AI) and nano-electronics, and increasingly, drives their convergence. With the proliferation of intelligent devices and the Internet of Things, and the resultant explosion of digital data that they generate, computing platforms across the spectrum will increasingly need to extract structure, patterns, and meaning from raw and unstructured datasets in real-time, and in unsupervised environments. Advances in AI, notably deep learning networks (DLNs), have led to computers matching or surpassing human performance in several sensory tasks including vision, speech and natural language processing. However, there still remains a large (several orders-of-magnitude) power efficiency gap between the human brain and computers that attempt to emulate some facets of its functionality. For instance, AlphaGo (a DLN-based system) defeated Go-Champion Lee Sedol in 2016 in the complex game Go that requires intuitive and strategic thinking [1]. While this is a truly impressive feat on the part of AI, the computational cost (energy or power) expended by the DLN to that of a human performing the same task gives a conflicting viewpoint. Human brain operates on a power budget of only ∼20Watts, whereas DLNs running on powerful servers can expend ∼KWatts of power [2].

This massive power-gap has initiated research efforts to explore more biologically realistic information-processing systems that are both competent as well as efficient like the brain. Spiking neural networks (SNNs), a widely used model for computational neuroscience and neuromorphic computing, holds potential of bridging this gap. Similar to biological neurons, neurons in SNN use discrete spikes to compute and transmit information in a sparse event-driven manner. The binary all-or-nothing spike-based communication combined with sparse temporal processing precisely make

SNNs a low-power alternative to conventional DLNs. In SNNs, information processing occurs in the temporal domain. This is a significant variation from the real-valued computations used in DLNs. From a signal theory perspective, this means that the analog amplitude or voltage value of the signal carries information in DLNs. In contrast, SNNs use timing of signals (or spikes) to encode information. The presence of the additional temporal dimension enables information to be encoded in a sparse manner yielding efficient computations. In fact, SNNs gain efficiency when integrated with event-based vision and audio sensors [3], which reduce redundant information in real-time data processing. This further makes spiking systems an attractive alternative for efficient hardware implementation of real-world applications such as, IoT, intelligent robotics, autonomous driving among others. With all its appeal for efficiency, training SNNs still remains a challenge. Practically, SNNs still lag behind DLNs, in terms of performance, in traditional learning tasks. While DLNs have found applicability across different domains from language processing to video analysis, SNNs still struggle to achieve competitive accuracy on standard image classification tasks. Note, SNNs can also be implemented as a deep convolutional hierarchy similar to that of DLNs. For notational convenience, we refer to spike-based temporal computing paradigm as SNNs and the real-valued analog computing paradigm as DLNs.

While SNNs can potentially resolve the huge energy consumption expended by DLNs, both DLNs and SNNs suffer from training or learning inefficiency. So far, supervised learning mechanisms in both SNNs and DLNs have proven to be very effective in providing competetive accuracy in several image classification tasks [4]. However, supervised learning is inherently static. That is, the learning methods use data points from past or old experience to build a predictor (classifier, regression model, recurrent time series models) for processing future behavior. Static learning mechanisms suffer from catastrophic forgetting wherein the knowledge of previously learnt tasks is abruptly lost as information relevant to a current task is incorporated. Thus, continual learning requires retraining of the predictor with both previous and

current information, leading to large training overhead, that becomes a bottleneck for efficiency.

So far, efficiency and accuracy remains a key focus for researchers in the hardware/software community to deploy lower complexity, yet, accurate AI, either through DLNs or SNNs. The innovations and proposals in this direction has been traditionally developed with an assumption that the environment is benign during both training and evaluation of the AI model. However, a crucial aspect that has been implicitly ruled out is the robustness of such systems. Specifically, an attacker can alter the statistical properties of the input data that can adversely affect the operation of the AI model. DLNs have been shown to be vulnerable to *adversarial examples*: slightly perturbed inputs that are specifically designed to fool a model during test time [5–8]. Recent works have demonstrated the security danger that adversarial attacks pose across several platforms with DLN backend such as computer vision [6, 7, 9–11], malware detectors [12–15] and gaming environments [16, 17]. Our preliminary analyses [18] have also shown SNN's vulnerability to adversarial attacks. Thus, there is a need to understand the vulnerabilities inherent in AI systems and explore techniques to defend against them.

1.1 Contributions

In this thesis, we target three specific challenges: energy-efficiency, accuracy and robustness of neural networks, and propose principled techniques to overcome the challenges in spiking and deep learning domain. Our contributions are:

- *Energy-efficient implementation of conventional deep learning frameworks (or DLNs) that reduce the computational requirements for a given network without compromising its accuracy*: Our proposals are based on the observation that current training processes often result in networks that are more complex than necessary (e.g., they spend the same computational effort on each input regard-

less of the fact that in practice the inherent difficulty of classification varies greatly across inputs).

- *Learning algorithms for SNNs that enables deeper convolutional implementations with competitive accuracy and strategies that enable continual learning without catastrophic forgetting*: We explore both supervised and unsupervised spike-based training to develop the SNN frameworks for image classification tasks. Further, we design learning algorithms for a novel SNN computing model, Liquid State Machines (LSMs), that offer an efficient, low complexity architecture for sequential spatio-temporal data recognition.

- *Adversarial robustness of DLNs based on discretization themes and prior learning*: We explore the efficacy of discretization techniques like, input and weight quantization (or binarization) that are primarily used to deploy energy-efficient DLNs, for defending against adversarial attacks. We also introduce a novel noise-based prior learning technique for training neural networks that are intrinsically robust to adversarial attacks.

1.2 Thesis Outline

Fig. 8.1 illustrates the overall outline of this thesis. Chapter 2, 3 deal with the energy-efficiency aspect of DLNs, Chapter 4, 5, 6 investigate the new spike-based learning rules for SNNs and Chapter 7, 8 investigate adversarial robustness of DLNs.

Chapter 2 discusses **conditional deep learning** [19, 20]. Although traditionally the entire network is utilized for the recognition of all inputs, we observe that the classification difficulty varies widely across inputs in real-world datasets; only a small fraction of inputs require the full computational effort of a network, while a large majority can be classified correctly with very low effort. In this chapter, we describe Conditional Deep Learning (CDL) where the convolutional layer features are used to identify the variability in the difficulty of input instances and conditionally activate the deeper layers of the network. We achieve this by cascading a linear

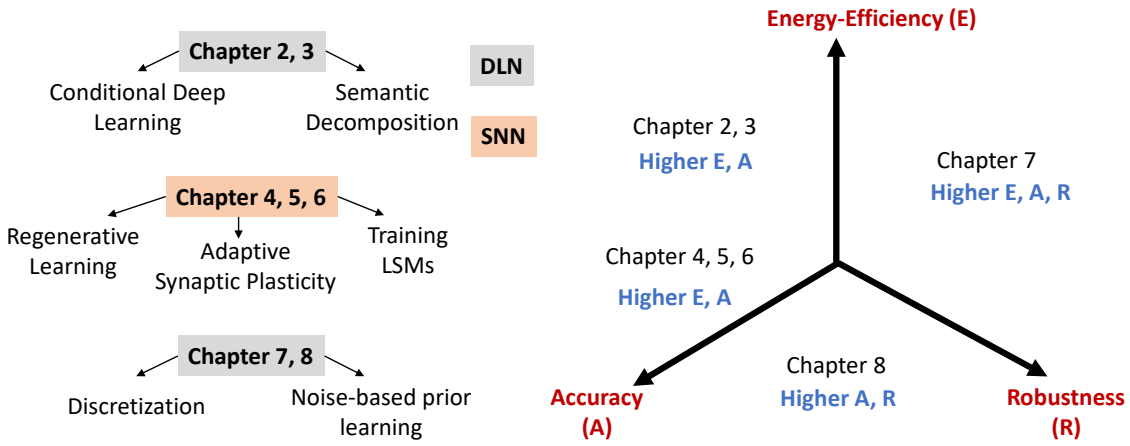


Fig. 1.1. Thesis Contributions addressing the three design challenges for neural networks: higher energy-efficiency, improved robustness and high accuracy for a given task. Our proposals span across both SNNs and conventional DLNs for various design aspects across different recognition or classification tasks.

network of output neurons for each convolutional layer and monitoring the output of the linear network to decide whether classification can be terminated at the current stage or not. The proposed methodology thus enables the network to dynamically adjust the computational effort depending upon the difficulty of the input data while maintaining competitive classification accuracy. We further employ the conditional approach to train deep learning networks from scratch with integrated supervision from the additional output neurons appended at the intermediate convolutional layers. Our proposed integrated CDL training leads to an improvement in the gradient convergence behavior giving substantial error rate reduction.

Chapter 3 discusses **semantic decomposition** for efficient classification [21]. In this chapter, we present a new approach to optimize energy efficiency of object detection tasks using semantic decomposition to build a hierarchical classification framework. We observe that certain semantic information like color/texture are common across various images in real-world datasets for object detection applications. We

exploit these common semantic features to distinguish the objects of interest from the remaining inputs (non-objects of interest) in a dataset at a lower computational effort. We propose a 2-stage hierarchical classification framework, with increasing levels of complexity, wherein the first stage is trained to recognize the broad representative semantic features relevant to the object of interest. The first stage rejects the input instances that do not have the representative features and passes only the relevant instances to the second stage. Our methodology thus allows us to reject certain information at lower complexity and utilize the full computational effort of a network only on a smaller fraction of inputs resulting in energy-efficient detection.

Chapter 4 describes a spike-based unsupervised **regenerative learning** [22] scheme to train Spiking Deep Networks for object recognition problems using biologically realistic leaky integrate-and-fire neurons. The training methodology is based on the Auto-Encoder learning model wherein the hierarchical network is trained layer wise using the encoder-decoder principle. Regenerative learning uses spike-timing information and inherent latency to update the weights and learn representative levels for each convolutional layer in an unsupervised manner. The features learned from the final layer in the hierarchy are then fed to an output layer. The output layer is trained with supervision by showing a fraction of the labeled training dataset and performs the overall classification of the input. The proposed methodology also introduces sparsity in the hierarchical feature representations on account of event-based coding resulting in computationally efficient learning.

Chapter 5 presents **Adaptive Synaptic Plasticity** (ASP) [23, 24] that enables us to build a stable-plastic adaptive SNN with limited memory. A fundamental feature of learning in animals is the *"ability to forget"* that allows an organism to perceive, model and make decisions from disparate streams of information and adapt to changing environments. Against this backdrop, we present a novel unsupervised learning mechanism for improved recognition with SNNs for real time on-line learning in a dynamic environment. We incorporate an adaptive weight decay mechanism with the traditional STDP learning to model adaptivity in SNNs. The leak rate of the

synaptic weights is modulated based on the temporal correlation between the spiking patterns of the pre- and post-synaptic neurons. This mechanism helps in gradual forgetting of insignificant data while retaining significant, yet old, information. ASP, thus, maintains a balance between forgetting and immediate learning to construct a stable-plastic self-adaptive SNN for continuously changing inputs. We demonstrate that the proposed learning methodology addresses catastrophic forgetting while yielding significantly improved accuracy over the conventional STDP learning method for digit recognition applications. Additionally, we observe that the proposed learning model automatically encodes selective attention towards relevant features in the input data while eliminating the influence of background noise (or denoising) further improving the robustness of the ASP learning.

Chapter 6 describes a combined Hebbian and non-Hebbian plasticity rule for **training LSMs** or recurrent SNNs [25]. Synaptic Plasticity, the foundation for learning and memory formation in the human brain, manifests in various forms. We combine the standard spike timing correlation based Hebbian plasticity with a non-Hebbian synaptic decay mechanism for training a recurrent spiking neural model to generate sequences. We show that inclusion of the adaptive decay of synaptic weights with standard STDP helps learn stable contextual dependencies between temporal sequences, while reducing the strong attractor states that emerge in recurrent models due to feedback loops. Furthermore, we show that the combined learning scheme suppresses the chaotic activity in the recurrent model substantially, thereby enhancing its ability to generate sequences consistently even in the presence of perturbations.

Chapter 7 investigates the effect of **discretization** techniques on adversarial robustness of DLNs [26]. Adversarial examples are perturbed inputs that are designed (from a DLN's parameter gradients) to mislead the DLN during test time. Intuitively, constraining the dimensionality of inputs or parameters of a network reduces the 'space' in which adversarial examples exist. Guided by this intuition, we demonstrate that discretization greatly improves the robustness of DLNs against adversarial attacks. Specifically, discretizing the input space (or allowed pixel levels

from 256 values or 8bit to 4 values or 2bit) extensively improves the adversarial robustness of DLNs for a substantial range of perturbations for minimal loss in test accuracy. Furthermore, we find that Binary Neural Networks (BNNs) and related variants are intrinsically more robust than their full precision counterparts in adversarial scenarios. Combining input discretization with BNNs furthers the robustness, even waiving the need for adversarial training for certain magnitude of perturbation values. We also show standalone discretization remains vulnerable to stronger multi-step attack scenarios necessitating the use of adversarial training with discretization as an improved defense strategy.

Chapter 8 describes **Noise-based prior Learning** (NoL) [27]. We find that the implicit generative modeling of random noise with the same loss function used during posterior maximization, improves a model's understanding of the data manifold furthering adversarial robustness. We evaluate our approach's efficacy and provide a simplistic visualization tool for understanding adversarial data, using Principal Component Analysis. Our analysis reveals that adversarial robustness, in general, manifests in models with higher variance along the high-ranked principal components. We show that models learnt with our approach perform remarkably well against a wide-range of attacks. Furthermore, combining NoL with state-of-the-art adversarial training extends the robustness of a model, even beyond what it is adversarially trained for, in both white-box and black-box attack scenarios.

Finally, **Chapter 9** summarizes the thesis and discusses future work.

2. CONDITIONAL DEEP LEARNING FOR ENERGY-EFFICIENT AND IMPROVED IMAGE RECOGNITION

2.1 Introduction

Deep Learning Networks (DLNs) have emerged as one of the most prominent classification tools across the computing spectrum for search, recognition and other cognitive applications [28, 29]. They have been successfully deployed in several real-world products such as Google Image search [30], Google Now speech recognition [31, 32] among others. However, being large scale and densely connected makes them highly computationally intensive. For instance, SuperVision [33], a DLN that won the ImageNet visual recognition challenge, demands compute performance in the order of 2-4 Giga-OPS (OPS: total number of Multiply and Accumulate or MAC operations) per classification [34]. With compute efficiency becoming critical across modern computing platforms, energy-efficient realization of DLNs is of great interest.

Conventional deep learning algorithms require an input instance to be processed through every layer of the DLN to obtain the final classification result or output label. However, we observe that real-world datasets exhibit an inherent variability in the difficulty of inputs; consider the simple example of recognizing a person from two images: one where the person is standing against a plain blue backdrop and other where he is in the midst of a crowd. Clearly, the latter one takes more time and effort. Ideally, to obtain both speed and energy efficiency, computational time and energy used by algorithms should be proportionate to the difficulty of the input instances [35]. In this chapter,we propose Conditional Deep Learning (CDL) to construct a cascaded architecture for conditional activation of the latter layers in a DLN depending upon the difficulty of the input data, for faster and more energy-efficient implementations.

Interestingly, we note that the convolutional layers (CNN layers) of a DLN, interpreted as visual layers, learn a hierarchy of features which transition from general (similar to Gabor filters and color blobs [36]) to specific as we go deeper into the network [37]. In fact, DLN models that are trained for classification have been used as feature extractors by removal of the final output layer [38–40]. In particular, features extracted from a pre-trained DLN, OverFeat [41], have been successfully used in computer vision tasks such as scene recognition or object detection. Here, we utilize the generic-to-specific transition in the learnt features of the CNN layers to identify the inherent variability in the difficulty of the inputs in a dataset. The outputs of the first layers of a DLN are used to classify the easy instances of a given dataset without activating the latter layers of the network. Only for the hard instances that in general, constitute a small fraction of the dataset, the deeper layers are enabled to make more accurate classifications.

DLNs alike other supervised learning approaches have two modes of operation: training and testing. In the training phase, decision boundaries are constructed with training labels provided with the dataset. In the testing phase, the trained model is used to classify new instances. The basic methodology of Conditional Deep Learning (CDL) is as follows: During training, we construct a series of decision models (i.e. cascade of linear networks at every convolutional layer). This is completely different from the traditional approach where a single complex model (i.e. baseline DLN) is only used. In the test phase, the difficulty of the input instance determines the number of models or linear networks to be applied for accurate classification. Fig. 8.1 illustrates our methodology with a 2-hidden layer artificial neural network classifier. Fig. 8.1(a) shows the traditional approach where input training instances are classified into two categories by the complex model X. It is evident that non-linear boundaries would require more number of hidden layers and would thus be more computationally intensive than the linear boundary models. In the example of Fig. 8.1(a), the model X requires activation of both the hidden layers to classify the instances with high accuracy. However, this causes redundant activation of the second hidden layer for

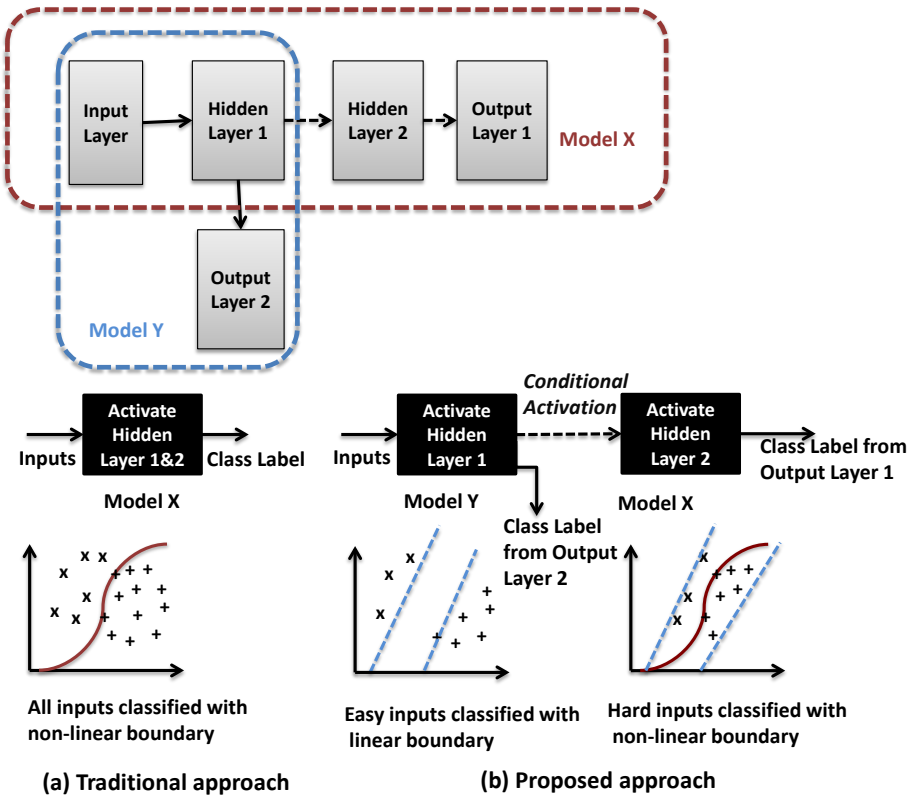


Fig. 2.1. (a) Traditional approach where both layers are activated and all inputs are classified with a non-linear boundary (b) Proposed approach where easy instances are classified at hidden layer 1 with linear boundary and hard instances at 2nd layer with non-linear boundary [35].

the easy instances of the dataset which increases the computational effort. We address this inadequacy with our proposed approach shown in Fig. 8.1(b). It consists of two decision models (Y and X) created by adding an output layer 2 after the 1st hidden layer. The simpler model Y (only 1st layer activated) selectively classifies only the easy training inputs that lie further from the original non-linear boundary by creating a hyperplane or region around it. If the confidence level of the output layer 2 for a given instance is below a certain threshold δ, the complex model X is employed by activating the 2nd hidden layer. Thus, the 2nd layer is only activated for the hard instances in the dataset. This approach thus leads to substantial energy savings,

since the complex decision boundary (non-linear model X) need not process all data instances. In addition to energy-efficiency, our experiments demonstrate that the CDL methodology shows substantial accuracy improvements over the baseline DLN. Please note that in CDL, we take a trained baseline model and append output layers on top of the trained model in order to lower the compute effort while testing.

Inspired from the accuracy enhancement results, we explored a completely new direction; Integrated training with CDL, where we investigated the use of the additional output layers to train a DLN from scratch. DLN's have been known to face training difficulty because of 'vanishing' gradients [42]. CDL exploits the usefulness of the CNN features and the additional output layers to reduce the testing complexity of a DLN. In contrast, in Integrated CDL (ICDL) training, we use the supervision provided by the additional output layers to train DLN to get improved accuracy and reduce the training difficulty. Recent works have proposed various training techniques like data augmentation [43], dropout [44], maxout [45] and layer-wise pre-training [46] to bring enhanced performance with DLNs on challenging problems. In [39], the authors have used two additional output layers in a 22-layer DLN (GoogleLeNet) to provide additional regularization and increase the gradient signal that gets propagated back. We build upon this observation and introduce Integrated CDL training to further optimize the learning of a DLN. Our experiments demonstrate that Integrated CDL gives us substantial improvement in the gradient convergence behaviour further minimizing the error. Please note that though our training method draws inspiration from [39], the two methods have different focus, design and evaluation strategies.

We studied the CDL architecture using well-known networks (LeNet, AlexNet, ResNet) and datasets (MNIST, CIFAR10, Tiny ImageNet) on the widely-used Torch platform [47]. We show that even for larger DLN models, our proposed scheme can both improve accuracy and significantly reduce the runtime performance and computational cost during testing. Also, we propose an improved training scheme based on the accuracy improvements observed with CDL. The training scheme, Integrated

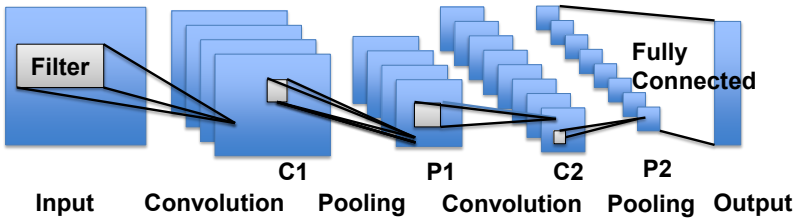


Fig. 2.2. A standard architecture of a Deep Learning Convolutional Network.

CDL training, further improves the accuracy of a DLN using additional supervision from intermediate output layers during training. We achieve this by backpropagating the error gradients during training from the intermediate output layers in addition to the final output layer of the DLN. This results in improved gradient convergence behaviour.

2.2 Conditional Deep Learning Classification

In this section, we present our structured approach to design the proposed Conditional Deep Learning Network (CDLN). As introduced earlier, CNNs form the basis of a deep learning network. DLN consists of one or more pairs of convolution and max pooling layers [48]. Fig. 8.2 shows a basic DLN structure with convolutional layers (C1, C2) followed by pooling layers (P1, P2). A convolution layer convolves a set of weigt! kernels with the previous layer to obtain an array of output maps. These kernels are repeated across the entire input space. A max-pooling layer lowers the dimensionality of the activations in the convolution layer by taking the maximum activation in a portion of the previous layer map. This incorporates translational invariance in the DLN to small variations in pixel positions of input images. Deeper layers require larger number of kernels that work on lower dimensional inputs to process complex components of the image. The final fully connected layers com-

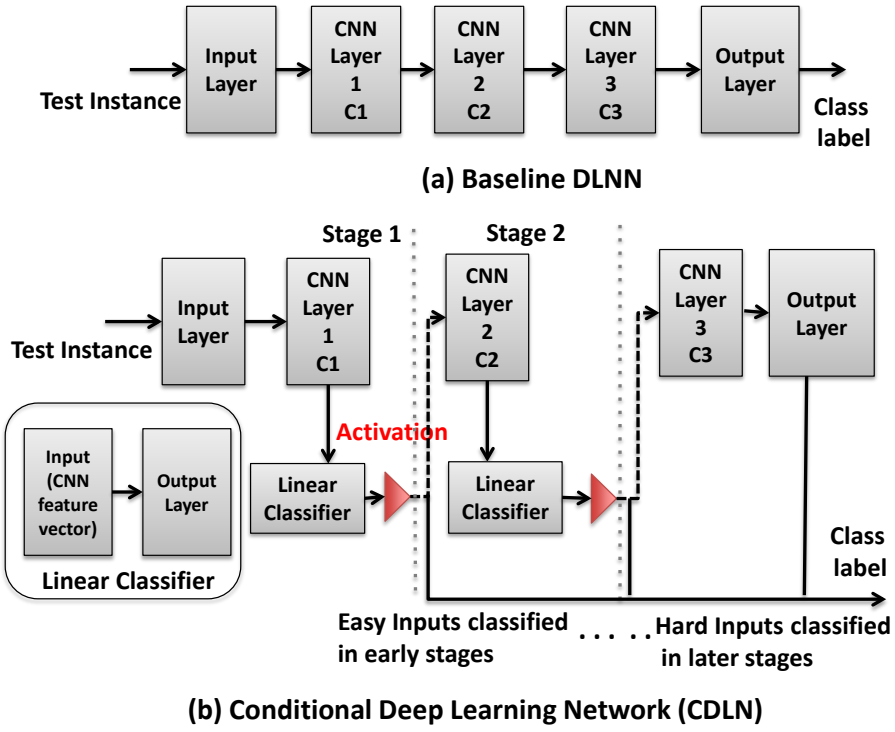


Fig. 2.3. (a) Baseline deep learning network (b) CDLN with linear classifiers added at the convolutional layers whose output is monitored to decide if classification can be terminated at current stage or not.

bine inputs from all maps in the preceding layer and perform overall classification of the input data. This hierarchical structure gives good results for image recognition tasks [49]. As mentioned earlier, CNN layers of DLN models that are trained for classification, have been used as feature extractors by removal of the output layer. We exploit the efficacy of the convolutional layer features to develop an architecture in which easy instances can be classified earlier without activating the latter layers of the DLN network.

Fig. 6.2 shows the conceptual view of CDLN. Fig. 6.2(a) consists of the baseline deep learning network with 3 convolutional layers (CNN layers: C1, C2, C3) which are learnt using the standard backpropagation algorithm. We have not shown the

pooling layers or the filters for the sake of convenience in representation. Fig. 6.2(b) illustrates our cascaded approach wherein the output features from each convolutional layer are fed to a linear classifier. The linear classifiers consist of same number of output neurons as that of the output layer of the baseline DLN (Fig. 6.2(a)). Thus, our proposed methodology consists of several stages, equivalent to the number of CNN layers, connected in a sequence. Each stage contains a linear classifier trained on the convolutional layer features corresponding to that stage. Depending upon the output of the linear classifiers in a given stage, the following stage of the CDLN is enabled. As mentioned earlier, as we go deeper into the network, the decision boundary model for each stage in the CDLN becomes progressively non-linear. Thus, class labels are produced at stage 1 for easy inputs and latter stages for hard ones.

Besides the linear classifiers, the stage consists of an activation module (triangles in Fig. 6.2(b)). During test time, input instance is passed through each stage to produce a class label. The stage also generates a confidence value (probability value at the output neuron) along with the class label. The activation module utilizes this confidence to decide if the input instance should get classified in the current stage or passed to the next stage. This decision is based on the following two criteria:

- If the linear classifiers do not produce sufficient confidence associated with any of the class labels or produce a sufficient confidence for more than one label, the input is deemed to be difficult to classify by the current stage and is passed along to the next stage.

- If the linear classifiers produce sufficient confidence associated with only one label, then the classification process is terminated at that stage and the corresponding label is produced as output of the framework.

In addition to energy-efficiency, we also observe that the performance of the CDL network is better than the baseline DLN in terms of classification accuracy. This can be attributed to the fact that the linear networks being small scale with few neurons and synapses can be trained rapidly and easily. Also, the linear networks are added

on top of a trained baseline DLN to construct the CDL. The linear networks are trained on learnt features from the convolutional layers of the trained baseline DLN. So, these additional networks achieve better least mean square error as compared to the baseline DLN. Hence, a DLN, which is less than optimal, i.e. not fully trained or over-fitting, can also extract features for the linear networks. The CDL with the linear networks thus yields competitive classification accuracy.

2.2.1 Efficiency and Accuracy Optimization

As the CDLN is composed of many individual stages, comprising of a series of linear classifiers, the following two factors determine their overall efficiency and accuracy: (a) the number of linear classifiers added and (b) the fraction of inputs processed at each stage. These factors present a fundamental tradeoff in the CDLN design methodology. The tradeoffs are discussed in the following subsections.

Adding linear classifiers at the convolutional layers

First, we examine whether it is desirable to add a linear classifier for every convolutional layer of the DLN. Please note that we need to take into account the additional cost of adding an output layer of neurons for each convolutional layer while calculating energy costs [35]. Let the computational cost of the CDLN at a particular stage (including the additional cost of linear network at that stage) i be γ_i per instance. Let the fraction of instances that reach stage i be I_i. Similarly, the fraction of instances that reach stage $i+1$ is I_{i+1}. Thus, the stage i classifies only a smaller subset $(I_i - I_{i+1})$ of the inputs. Stage i should satisfy Eq.(2.1) shown below in order to improve the overall efficiency of the framework.

$$(\gamma_{i+1} - \gamma_i).(I_i - I_{i+1}) > \gamma_i.I_{i+1} \tag{2.1}$$

The left hand side of Eq.(2.1) signifies the efficiency improvement due to the addition of the linear classifier at the convolutional layer i, which is the product

of the fraction of inputs classified at the stage$(I_i - I_{i+1})$ and the reduction in cost compared with the activation of the convolutional layer in the next stage. The left side product should be larger than the rigt! side of Eq.(2.1), which represents the penalty that addition of the linear classifier inflicts on instances that are misclassified i.e. if the linear classifier was not present then the instances (I_{i+1}) can be classified directly by the linear classifier corresponding to the convolutional layer in the next stage, *i+1*.

Modulating activation of layers using confidence value

The activation module described earlier uses the confidence value of the output at each stage to selectively classify the input or pass it to the next stage. To better understand how the confidence influences the CDLN, consider the example shown in Fig. 6.3. Each stage in the Fig. 6.3 is defined as shown in Fig. 6.2(b) i.e. it consists of a convolutional layer (from the baseline DLN) which is fed as input to a linear classifier. The CDLN classifies a given input instance into one of the four class labels. During test time, data is processed through each stage of the CDL network to produce a class label. The linear classifiers in the CDLN, in addition to the class label, provide a measure of confidence (e.g. class probabilities or distance from the decision boundary).

Referring to Fig. 8.1 (b), the confidence value, thus, defines a region around the initial or original non-linear decision boundary that separates the easy and hard instances. Hence, a low confidence value output at any stage implies that the given test instance is a hard input and needs to be processed by the latter layers of the network for accurate classification. Fig. 6.3(a) shows that stage 1 gives a confidence level of 0.9 and 0.8 associated with a fraction of easy instances while 0.3 and 0.4 for the hard instances. Choosing the activation value of 0.8 would terminate the classification process at stage 1 for the easy instances and activate the latter stage only for the hard inputs. If the value is chosen to be 0.9, then the second stage will be

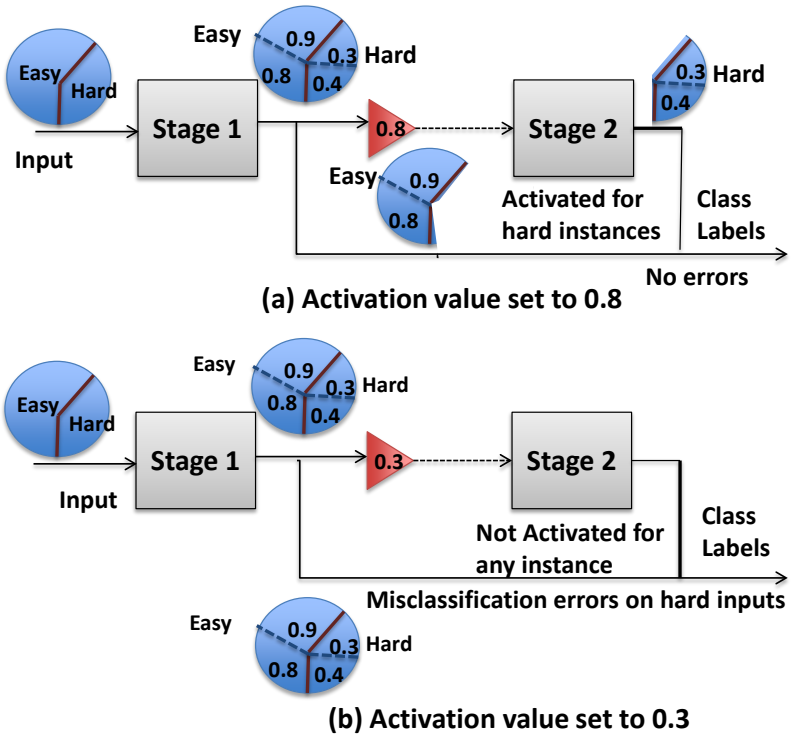


Fig. 2.4. (a) Activation value set to 0.8 terminates the classification for easy instances at stage 1 and enables stage 2 for hard instances (b) Activation set to 0.3 terminates the classification for easy and hard instances at stage 1.

redundantly activated for a fraction of easy instances (those with confidence value of 0.8) resulting in a decline in efficiency. On the other hand, if the activation value is set to 0.3 as in Fig 4(b), then, the second stage is not enabled at all. However, due to the low confidence value, majority of the hard instances will be misclassified. This would result in a significant decline in accuracy. Thus, modulating the activation value allows us to control the efficiency and accuracy of the framework. For computational efficiency, we choose higher confidence values (around 0.5 -0.7) during training to avoid misclassification errors.

Algorithm 1 Methodology to train the CDLN

Input: Original DLN N_{orig}, training dataset D_{tr} with the target labels

Output: CDLN N_{cdl} including the optimum number of stages

1: Train N_{orig} using D_{tr} and obtain the classifier cost γ_{orig}
2: **initialize** count= # of Convolutional layers in N_{orig}, $G_i = +\infty$
3: **while** $(G_i > \epsilon)$
4: **for** i=1:count **do**
5: Obtain the CNN features for the given D_{tr} corresponding to all maps for each convolutional layer of the DLN.
6: Concatenate the CNN features into a 1-D vector and feed it as input to a linear classifier (LC_i) with the same number of output neurons as N_{orig}.
7: Train LC_i with the target labels from D_{tr} using the least mean square rule.
//decide if a linear classifier needs to be added from second CNN layer or stage onwards
8: **initialize** I_i= # of input instances that reach stage i, CL_i= # of instances classified at stage i, CDLN cost till stage i = γ_i
9: G_i=(γ_{orig}- γ_i). CL_i- γ_i.$(I_i - CL_i)$
10: **if** $G_i > \epsilon$ **then** admit LC_i into N_{cdl}
11: **end for**
12: **end while**

2.3 Design Methodology

In this section, we describe the procedure for training and testing the CDLN.

2.3.1 Training CDLN

Algorithm 3.5.5 shows the pseudo code for training the CDLN. The process takes the original DLN N_{orig}, training data D_{tr} with the corresponding labels as input and produces a conditional deep learning network N_{cdl} with the optimized number of stages.

The baseline DLN (N_{orig}) is first learnt for a given training set (step 1). The learnt CNN features corresponding to the training data are concatenated into a 1-D vector and given as input to the linear classifier at every stage. The linear classifiers

(LC_i) are then trained on the same training data using the least mean square rule (steps 4-7). For each layer/stage from the second layer onwards, we compute the gain G_i which is the difference between the increase in computational efficiency for the instances classified at stage i and the additional cost it inflicts on instances that are passed to the next stage (step 9). We add the linear classifier LC_i to the CDLN N_{cdl} if G_i exceeds a certain user-defined threshold ϵ (step 10). The algorithm terminates if addition of an output layer at a particular CNN layer (i.e. a linear classifier) does not improve the overall gain G_i beyond ϵ (step 3). Please note that the linear classifiers being small scale and also being trained on learnt convolutional features from the trained DLN converge to the global minima (least error attainable by the linear classifier) in short time as compared to the baseline DLN. Also, the linear classifier at every stage is trained only on those instances passed from the previous stage. Since the fraction of input instances passed to the next stage decreases as we go deeper into the network, the training time for the linear classifiers progressively decreases.

2.3.2 Testing CDLN

Algorithm 5 describes the overall testing methodology for the CDLN. Given a test instance I_{test}, the methodology produces the class label L_{test} for it using N_{cdl}. The output from the linear classifier at every stage is monitored to decide if the input can be classified at the current stage or not. For the worst case (very hard instance), all the CNN layers and the corresponding linear classifiers at every stage will be activated and L_{test} will be the class label produced by the final output layer.

In summary, the design methodology implicitly modulates the number of stages or layers used for classification based on the input and produces an optimal CDLN. The user defined threshold, δ, for the confidence level can be adjusted during runtime to achieve the best tradeoff between accuracy and efficiency improvements of the

Algorithm 2 Methodology to test the CDLN

Input: Test instance I_{test}, CDLN N_{cdl} with the no. of linear classifiers or stages in N_{cdl}

Output: Class label L_{test}

1: Obtain the CNN layer feature vectors for I_{test} (CNN_i) corresponding to a stage/layer i.

2: If a linear classifier (LC_i) is present at stage i, obtain the output of LC_i corresponding to CNN_i.

3: If the confidence value of the output is beyond a certain threshold δ (user defined), then TERMINATE testing at stage i and Output L_{test} = Class label given by LC_i. The layers or stages of N_{cdl} from $i+1$ onwards are not activated if testing is terminated at stage i

4: If the confidence value of the output is below the threshold δ or output has high confidence value for more than one class label, activate the next stage $i+1$.

5: Goto step 1 and repeat this until you reach the final layer of the CDLN.

CDLN. Thus, the proposed approach is systematic and hence can be applied to all image recognition applications.

2.4 Experimental Methodology

In this section, we describe the experimental setup used to evaluate the performance of the Conditional Deep Learning Network against well-known benchmark datasets: MNIST [48], CIFAR10 [50] and Tiny Imagenet [51]. The Tiny ImageNet is derived from the standard ImageNet dataset with 200 categories (instead of 1000). Each of the 200 categories consists of 500 training, 50 validation and 50 test images down sampled to a resolution of 64x64 pixels.

We adapted three different widely-studied DLNs to our proposed CDL architecture: LeNet-5 [48] for MNIST, AlexNet-8 [33] for both CIFAR10 and Tiny ImageNet and ResNet-50 [52] for Tiny ImageNet. The baseline DLNs were trained using the standard convolutional back-propagation algorithm [53]. We employed the training methodology discussed in Section 3.3 to construct the CDLN with optimum number of stages. For LeNet-5 that consists of 3 convolutional layers and 2 fully connected layers, we observe that the CDL methodology, in addition to the final output layer (FC), adds a linear layer of output neurons (O1, O2) after the first and second convo-

LeNet ARCHITECTURE For MNST

BASELINE DLN	SIZE	MNIST_CDL (CDLN)
Input (I)	32x32	I→C1→P1→C2→P2→C3→P3→FC O1 O2
Convolution C1	28x28, 6 maps	
Pooling P1	14x14, 20 maps	
Convolution C2	10x10, 16 maps	
Pooling P2	5x5, 16 maps	
Convolution C3	120	
Fully Connected 1	84	
Output (FC)	10	

Fig. 2.5. Baseline and CDLN architecture of the networks for MNIST

lutional (or pooling) layers of the network as shown in Fig. 8.7 (LeNet_CDL). Please note that the learnt feature vectors from the pooling layers are used as training inputs to the linear classifiers. Addition of linear classifiers enables conditional activation of the layers: (C2, P2, C3, H1, FC) in LeNet_CDL depending upon the difficulty of the input instance. For AlexNet-8 that consists of 5 convolutional and 3 fully connected layers, the CDLN has four/two additional output layers (or linear classifiers) added after each pooling layer of the baseline AlexNet-8 DLN for Tiny ImageNet/CIFAR-10 respectively. With ResNet-50 that has 49 convolutional layers and 1 fully connected layer, the CDLN constructed for the Tiny ImageNet dataset has 8 additional output layers (in addition to the final output layer of the baseline ResNet-50 DLN) that implement conditional activation. Please note that the CDLNs constructed in each case uses a pre-trained baseline DLN model to append additional linear classifiers to the learnt intermediate convolutional layers.

Here, we used software simulations to obtain classification accuracy and hardware simulations to obtain energy values. We implemented each of the CDLNs for various datasets in the widely used Torch platform [47]. For CIFAR-10 and Tiny ImageNet, we use the standard AlexNet-8, ResNet-50 architectures available in Torch and customize the input layer size of the network to fit the requirements of our downsampled images. For hardware execution, we specified each classifier as an accelerator at the

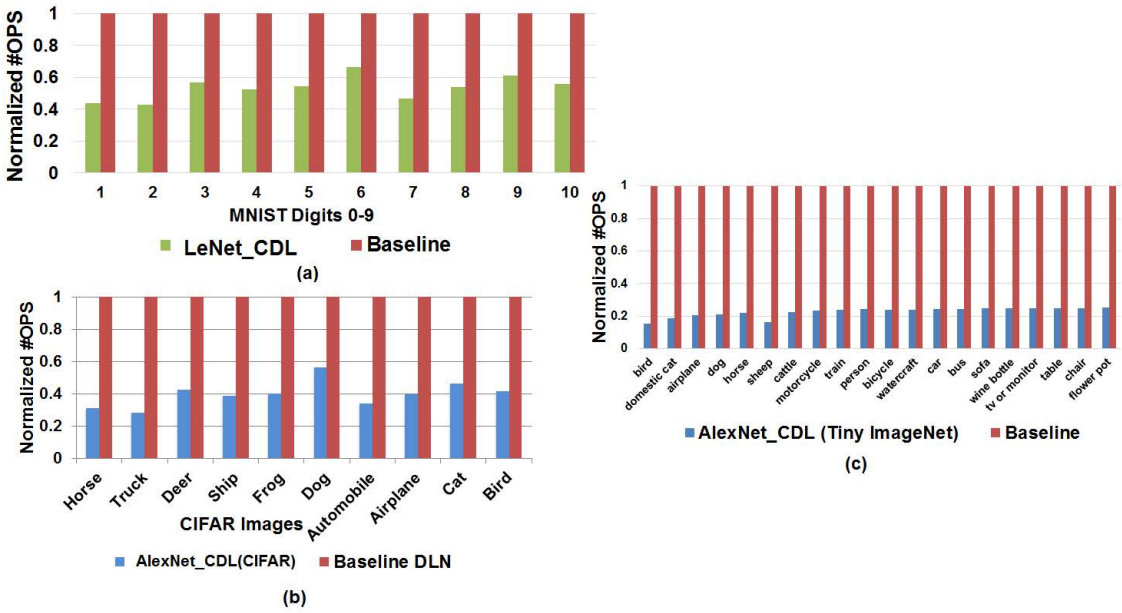


Fig. 2.6. (a) Normalized OPS for the CDLN (LeNet_CDL) (b) Normalized OPS of AlexNet_CDL(CIFAR) (c) Normalized OPS of AlexNet_CDL(Tiny ImageNet) for top-20 easy classes with respect to baseline

register transfer logic (RTL) level. Synopsys design compiler was used to synthesize the integrated design to a 45nm SOI process from IBM. Finally, Synopsys Power compiler was used to estimate energy consumption of the synthesized netlists.

2.5 Benefits with CDL

In this section, we present the experimental results that establish the potential of CDL. We use MNIST (with LeNet as baseline DLN) as our primary benchmark to evaluate the benefits with CDL with respect to energy and accuracy.

2.5.1 Energy Improvement

Fig. 6.5 shows the normalized improvement in efficiency with respect to the standard DLN models (which forms the baseline) for CDLN across all classes of different datasets. In case of Tiny ImageNet implemented with AlexNet-8 (Fig. 6.5 (c)), we only show the efficiency for 20 classes (out of the total 200) for convenience in representation. We quantify efficiency as the average number of operations (or computations) per input (OPS). From Fig. 6.5(a), we observe that LeNet_CDL gives 1.50x-2.32x (average: 1.91x) improvement. Note that the benefits observed vary for different digits. Fig. 6.5(a) clearly illustrates that maximum benefit in both the frameworks is observed for digit 1 and minimum for digit 5. We can thus infer that digit 5 has more hard instances in the testing set that are closer to the non-linear decision boundary and hence need activation of deeper layers for accurate classification. Digit 1, on the other hand, has easier instances away from the non-linear boundary and thus can be classified by the early layers with an approximate linear boundary decision model. Fig. 6.5(b) shows the normalized OPS for AlexNet_CDL(CIFAR) which provides an average of 2.85x improvement with respect to the corresponding baseline (AlexNet-8). Similar to MNIST, we can infer that dog/truck are the most difficult/easy inputs in the CIFAR10 dataset. Fig. 6.5(c) shows the OPS for Tiny ImageNet CDL implementation (AlexNet-8) that yields an average of 4.4x improvement with respect to the corresponding baseline. Of the 20 categories in the Tiny ImageNet shown in Fig. 6.5(c), bird/flower pot are the most easy/difficult input instances. The average OPS improvement ($\tilde{6}$.8x) in the ResNet-50 CDL implementation is significantly higher than the AlexNet-8 CDL. The higher benefits observed can be attributed to the fact that the DLN structure for ResNet-50 is more complex (higher number of neurons and synapses) than AlexNet-8. Additionally, the former DLN has more linear classifiers that gives the advantage of turning off more layers for a given input instance.

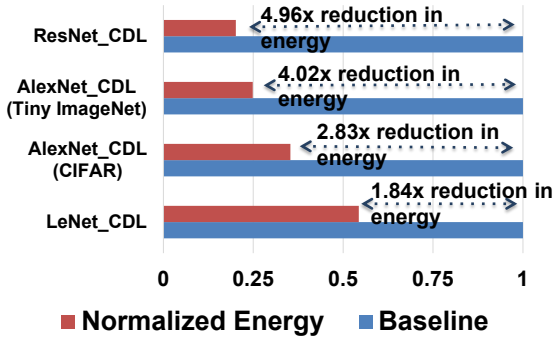


Fig. 2.7. Normalised energy benefits of CDLN with respect to baseline

In case of hardware implementation, the reduction in OPS translates on an average to 1.84x/2.83x/4.02x/4.96x reduction in energy for LeNet_CDL/ AlexNet_CDL (CIFAR)/ AlexNet_CDL (Tiny ImageNet)/ ResNet_CDL, respectively as shown in Fig. 6.6. For runtime performance, we report the overall inference time averaged across all test samples for each CDL implementation as shown in Table 3.4. It is clearly seen that CDL outperforms the original baseline network for all datasets in terms of runtime. LeNet_CDL has the largest performance gain as the baseline. A noteworthy observation here is that the fraction of instances classified at the appended output layers (4th column in Table 3.4) vary across datasets. For MNIST (LeNet-5), we observe that the $\tilde{9}0\%$ instances are classified at the early layers passing only 4.8% of the entire testing dataset to the final output layer (FC from Fig. 8.7(a)). Since CIFAR/Tiny ImageNet are more challenging datasets than MNIST, we observe that the fraction of instances passed to the latter layers increases implying that the images in such datasets are more difficult. However, even in both the challenging cases, we observe that less than 20% of the entire testing set requires the full computational effort of the DLN. As majority of instances ($\tilde{8}0\%$) are inferred at the early output layers (or linear classifiers), we observe that conditional activation continues

Table 2.1.
Performance results for different CDLN structures

Network	**Time (ms)**	**Gain**	**Fraction (%) of instances classified at the additional layers (O1, O2 ...) and final output layer (FC)**
LeNet-5 (MNIST)	3.31	-	-
LeNet_CDL	0.78	4.24x	82%, 9.2%, 4.8%
AlexNet-8(CIFAR10)	9.42	-	-
AlexNet_CDL(CIFAR)	5.13	1.83x	69.6%, 18.8%, 11.6%
AlexNet-8(Tiny ImageNet)	23.87	-	-
AlexNet_CDL(Tiny ImageNet)	17.31	1.37x	48.4%, 14.4%, 8.3%, 6.1%, 12.8%
ResNet-50 (Tiny ImageNet)	285.20	-	-
ResNet_CDL	166.8	1.71x	31.4%, 10.1%, 9.2%, 7.1%, 6.5%, 5.9%, 5.2%, 6.3%, 18.3%

to yield performance benefits (in terms of runtime/energy/OPS) even with increasing complexity of the original DLN (LeNet-5 →AlexNet-8→ResNet-50).

2.5.2 Improvement in Accuracy

Table 3.5.1 shows the overall accuracy for the baseline DLN architectures shown in Fig. 8.7 and the corresponding CDLNs LeNet_CDL, AlexNet_CDL(CIFAR), AlexNet_CDL (Tiny ImageNet), ResNet_CDL over the testing set. We observe that there is a consistent enhancement in accuracy for all CDL structures compared to the baseline. The accuracy shown for Tiny ImageNet is the top-1 accuracy. The difference in accuracy improvement in the networks is due to the fact that the baseline DLN in each case has a different training and test accuracy. In the beginning of the experiment, our motivation behind adding linear classifiers was to get an improvement in efficiency. However, the accuracy enhancement with the CDL methodology implies that the linear classifiers trained by the convolutional layer features perform better than the

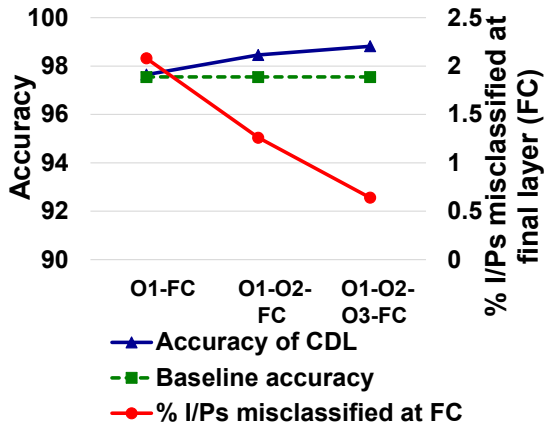


Fig. 2.8. Accuracy improvement in CDLN with the increase in no. of output layers

Table 2.2.
Accuracy of CDLN compared to baseline

Network	Baseline	CDLN
LeNet (MNIST)	97.55%	98.92%
AlexNet-8 (CIFAR)	78.38%	79.19%
AlexNet-8 (Tiny ImageNet)	65.24%	66.14%
ResNet-50 (Tiny ImageNet)	55.4%	56.52%

baseline DLN. We note that the main difference between the baseline DLN and the CDLN is that the baseline DLN uses the last convolutional or pooling layer features as inputs to the fully connected output layer of neurons to predict the classification result. As opposed to using the final fully connected layer for prediction, CDL methodology uses the trained linear classifiers added at every stage or layer which perform their own learning on the input CNN features.

As mentioned earlier, the linear classifiers being small can be trained to converge to the global minima (least error attainable by the linear classifier) in a short time

as compared to the baseline DLN. It is known that CNN learnt features become more specific and the feature vector also becomes smaller as we go deeper into the network. Thus, it is intuitive that the linear classifiers trained on the latter layers will reach smaller error minima in lesser time than the former ones. Adding more linear classifiers at every layer of the network would progressively minimize the overall error thereby improving the accuracy.

To quantify our theory, we designed an experiment where we added the linear classifiers one at a time with the baseline DLN (LeNet-5 for MNIST) in Fig. 8.7(a). During test time, we monitor the prediction results from the each of the added output layers (O1, O2, O3) in addition to the final output layer (FC) to measure the overall accuracy. Fig. 5.8 shows the normalized accuracy of the CDLN as we add the output layers one by one at every convolutional layer. We can observe an improvement in accuracy for each of the cases as compared to the baseline (97.55 %). While addition of just one linear classifier (O1-FC) enhances the accuracy by 0.1% (97.65 %), the benefits observed are higher up to 1.4% (98.92 %) with three linear classifiers for every CNN layer of the network. We also observe that the fraction of inputs misclassified by the final layer progressively decreases further corroborating our theory.

2.5.3 Optimizing the Number of Stages in the CDLN

Choosing the right number of stages or linear classifiers added at the CNN layers is critical to the efficiency of the CDL methodology. In order to pass fewer instances to FC, addition of linear classifiers at every CNN layer of the DLN is desirable. Fig. 5.10 shows the normalized OPS of the CDLN as the output layers are added one at a time for every CNN layer of the DLN architecture from Fig. 8.7 (a). It is clearly seen that fraction of inputs passed to FC decreases with the increasing number of output stages. We observe a significant drop in the fraction from 42% to 5% with the addition of two output stages (O1-O2-FC). Thus, initially we observe a decrease in #OPS. However, increasing the number of output stages adds an additional overhead

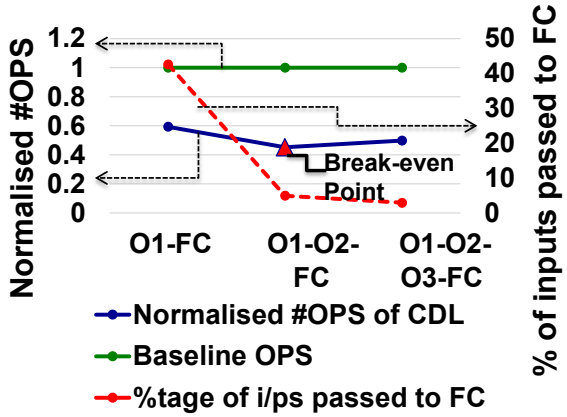


Fig. 2.9. Normalized #OPS as the no. of stages are increased in CDLN

to the cost of computation. Note, addition of a third stage (O1-O2-O3-FC) results in a marginal drop in the fraction of inputs passed from 5% to 3%, which is not significant enough to overcome the cost penalty that the addition of the stage imposes. So, we see an increase in #OPS from this point onwards. This break-even point (0.45 #OPS) corresponds to the maximum benefits or lowest #OPS that we can achieve using the CDL methodology for a given baseline DLN. This behavior is taken into account in our design methodology described in Section 3.3.

2.5.4 Efficiency-Accuracy Tradeoff using Confidence Level δ

The linear classifiers in the CDLN, in addition to the class label, provide a class probability. The activation module discussed in Section 3.2 compares this probability to the confidence level δ set by the user to selectively classify the input or pass it to the next stage. Thus, we can regulate δ to modulate the number of inputs being passed to the latter layers. Fig. 5.10 shows the variation in the normalized OPS (with respect to baseline DLN which quantifies efficiency) and accuracy for the CDLN (LeNet_CDL) with different δ. Setting δ to a low value implies more input

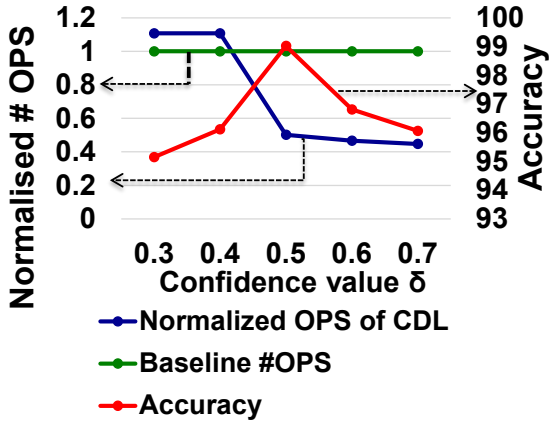


Fig. 2.10. Efficiency vs. accuracy tradeoff using confidence value δ

instances will be qualified as hard inputs and passed to the final output layer (FC) for classification. Then, FC will be redundantly activated for the easy inputs as well. As explained earlier, the accuracy of the CDLN improves as we increase the number of stages. In other words, more inputs should be classified by the linear classifiers at the CNN layers, instead of being passed to FC, for an improvement in accuracy. So, increasing δ would qualify more inputs as easy instances and result in more inputs being classified at the linear classifiers leading to lesser activation of latter layers. Thus, we see a decrease in #OPS and an increase in accuracy initially. However, beyond a particular δ, a fraction of hard inputs that should be ideally passed to the final layer for classification will now be misclassified at the early stages. This value of δ (0.5 in Fig. 5.10) corresponds to the maximum overall accuracy that can be achieved for the given CDLN. Beyond this point, the accuracy would decrease. The # OPS would still continue to decrease with increasing δ. In Fig. 5.10, we observe that the accuracy increases from 96.12% (δ=0.4) to 99.02% (δ=0.5) while the normalized #OPS reduce from 1.1 to 0.51. Further increase in δ degrades the accuracy and does not produce a significant reduction in #OPS. Thus, δ serves as a powerful knob to

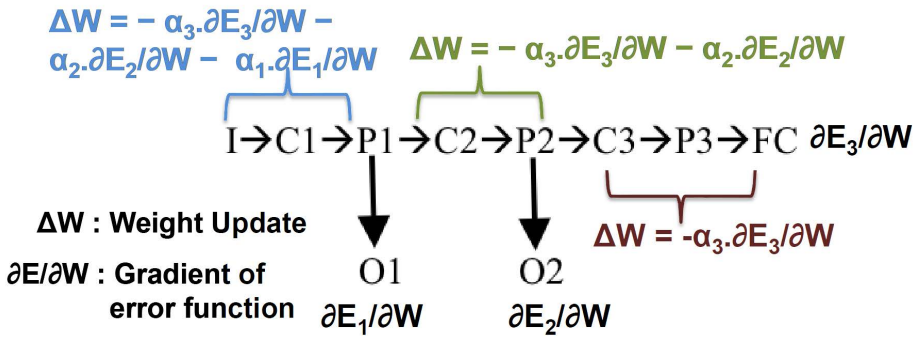


Fig. 2.11. Error propagation with Integrated CDL training where the gradient of cost/error function from each output layer is propagated back to calculate the weigt! update. The learning rate denoted as α varies across layers such that the gradient does not vanish as it is propagated deeper towards the initial input layer. The notations here are similar to that of Fig. 8.7. The weigt! update for a given layer is calculated by summing the gradients calculated from all the outputs that follow that layer.

trade accuracy for efficiency that can be easily adjusted during runtime to get the most optimum results.

We conducted a similar tradeoff analysis for each of our experiments to attain the most optimum δ that yields the highest accuracy. For our OPS evaluations in Fig. 6.5, we use δ = 0.5, 0.45, 0.5, 0.6 for LeNet_CDL, AlexNet_CDL(CIFAR), AlexNet_CDL(Tiny ImageNet), ResNet_CDL resepectively as obtained from the trade-off analysis.

2.6 Integrated CDL Training of DLN

In the training methodology described in Section 3.3, a trained baseline DLN is employed to construct a CDLN to get energy improvements. In addition to efficiency, we observe significant performance (or accuracy) gains for the different experiments conducted in Section 3.5. This indicates that the additional supervised learning for

each convolutional layer that the conditional approach imposes with the addition of an output layer enables further network optimization. Inspired by this observation, we utilize the conditional approach to train deep learning neural networks to get enhanced performance or better classification accuracy. Integrated CDL training enables the earlier layers to be learnt with additional direct supervised training with the error propagated from output layers appended at each convolutional layer.

Integrated CDL backpropagates the error not only from the final fully connected output layer but also from the intermediate linear classifiers. We build the DLN from scratch in this case. We would like to note that the CDLN that emerges from integrated training would not result in minimum #OPS. For instance, referring to Fig. 5.9, integrated CDL adds an output layer for every convolutional layer of the network. Thus, the CDLN from integrated training comprises of four output stages (O1-O2-O3-FC), which does not correspond to the break-even point that yields maximum energy benefits. However, the accuracy with the given configuration considerably increases. Our experiments on the MNIST and CIFAR10 dataset in the next section show consistent improvement in classification accuracy with integrated CDL training with respect to baseline DLN. We believe the performance boost associated with integrated CDL training can be attributed to the following reasons: a) the linear classifiers trained on the CNN features optimize the learning process preventing overfitting [54], b) the addition of the linear networks results in improved convergence behavior diminishing/reducing the vanishing gradient issue [42, 55].

Integrated CDL training is straigt!forward and follows the standard convolutional backpropagation algorithm discussed in [53] using stochastic gradient descent. However, we add the gradients from the linear classifiers associated with every convolutional layer to the gradient of the error function from the final fully connected layer (FC) to calculate the weigt! update across the entire CDLN for every epoch as shown in Fig. 5.11. The weigt! update for each layer has additional gradient terms depending upon the number of linear classifiers present in the subsequent layers.

Algorithm 3 Integrated CDL training (ICDL)

Input: Training dataset D_{tr} with the target labels

Output: Trained CDLN N_{cdl}

Require: L= # of layers of the DLN, K= # of linear classifiers or output layers added to the DLN, DLN with parameters W and the additional weigt!s, w, of linear classifiers, Input data x and corresponding label y, learning rates α_k

1: **procedure** ICDL(DLN, x, y, α_k)

Forward Propagation:

2: **for** i=1:L **do**

3: Compute activation a_i according to previous layer output a_{i-1}, W_{i-1}.

4: **for** j=1:K **do**

5: Compute activation of linear classifiers l_i as per the activation of corresponding convolutional layer a_j and w_j.

6: **end for**

7: **end for**

Backward Propagation:

8: Calculate final fully connected output layers error signal $\psi = \frac{\partial C}{\partial a_L}$

9: **for** i=L:1 **do** *//Error propagation from final output layer*

10: Compute weigt! update $\Delta W_1^i = -\alpha_k . \frac{\partial C}{\partial a_i} . a_{i-1}$

//C is the error/cost function

11: **for** j=K:1 **do***//Error propagation from additional linear classifiers*

12: Compute weigt! update $\Delta W_2^j = -\alpha_k . \frac{\partial C}{\partial a_j} . a_{j-1}$

13: $W = W - \Delta W_1 - \Delta W_2$ (in accordance with Fig. 5.11)

14: Continue to Step 1 with next input.

15: **end for**

16: **end for**

Algorithm 3 shows the pseudo-code for training the DLN with integrated CDL. The methodology appends output layers for each convolutional layer in the DLN without accouting for any optimization criteria for the number of stages (Eq.(**??**)). It requires training data D_{tr} with the corresponding labels as input and produces a trained conditional deep learning network, N_{cdl}.

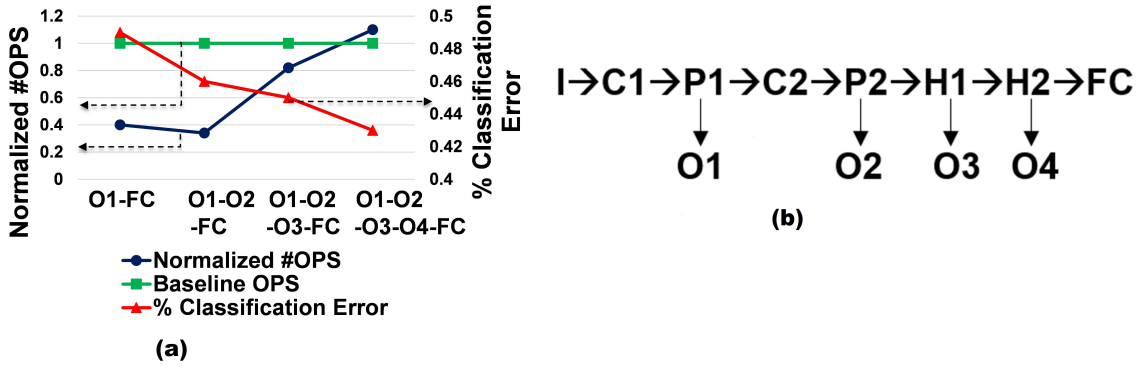


Fig. 2.12. (a) Normalised #OPS and % error for a DLN trained with ICDL on MNIST (b) CDL architecture for baseline DLN corresponding to [56] for MNIST. The notations here are similar to Fig. 8.7. H1 and H2 denote the additional fully-connected hidden layers prior to the final output layer in the baseline DLN.

2.7 Benefits of Integrated CDL Training

We assess our training method with MNIST and CIFAR10 datasets. Algorithm 3 is used to train a deep model with Integrated CDL (ICDL). For fair comparison of performance and to show the benefits with integrated training, the CDL network for MNIST and CIFAR10 has the same complexity as that of the DLN in [45, 56]. Also, in this case we append softmax classifiers instead of linear classifiers at the convolutional layers. For testing, the methodology described in Section 3.3 (Algorithm 5) is employed. Thus, the outputs from each additional output layer is monitored to decide if the classification can be terminated at an earlier stage. Hence, easy vs. hard input discrimination can be observed in this case as well. We set the confidence value $\delta = 0.5$ for all analysis in the following subsections.

2.7.1 Cost and Accuracy Analysis

Fig. 5.13 (a) shows the normalised #OPS and test classification error for a DLN trained with ICDL for MNIST. The CDL architecture with extra output layers is shown in Fig. 5.13 (b). During testing, each output layer is added one at a time for every CNN/hidden layer of the baseline DLN architecture in [56]. The #OPS in Fig. 5.13(a) follow a similar trend as that of Fig. 5.9 with increasing number of stages. Here, the break-even point corresponds to the stage with two additional output layers (O1-O2-FC) that results in 3.14x reduction in #OPS. We observe that the #OPS increases beyond the break-even point while the error decreases with the addition of output stages that is coherent with our previous analysis in Section 3.5. A noteworthy observation here is that when the output is monitored from *O1-FC* (adding a single output stage while testing) the error observed is 0.49% implying a 7.5% reduction in error rate with respect to baseline. There is a 2.57x reduction in resultant #OPS with *O1-FC*. Since the additional output layers are now trained with backpropagated gradients from latter layers, the features learnt at *O1* are more discriminative. In other words, the fraction of inputs misclassified at *O1* decreases considerably with ICDL improving the overall performance. The fact that we observe energy benefits with substantial accuracy improvement from ICDL training followed by conditional testing shows the efficacy of our proposed training methodology.

To quantify the improvement in the learnt features with our proposed training, we compare the performance of the network in Fig. 5.13 (b) constructed from scratch with ICDL training against a standard CDL constructed from a trained baseline DLN (Algorithm 3.5.5). Fig. 5.14 illustrates the % of inputs misclassifed at each output layer for the instances that are deemed to be suitable for classfication at that stage as per Algorithm 5. It is clearly evident ICDL gives lower error than the standard CDL at each output stage. Previously, in standard CDL, the hard instances for which classification was terminated at earlier stages were misclassified resulting in overall decline in accuracy. However, owing to the improved feature learning with

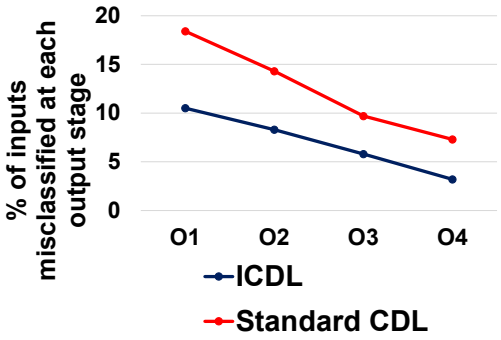


Fig. 2.13. Comparison of % of inputs misclassified at each output layer for ICDL and standard CDL training

ICDL, those hard inputs are now correctly classified by earlier layers. Thus, we infer that ICDL further improves the hierarchical feature learning of a DLN making the intermediate layers more robust and discriminative.

2.7.2 Improved Gradient Convergence with ICDL

The main motivation of the integrated training is improvement in classification accuracy. So, in order to observe lowest error rate, we have to append output stages at every layer of the DLN while testing. Thus, we use the configuration O1-O2-O3-O4-FC (from Fig. 5.13 (b)) to measure the overall test classification error and evaluate the performance of ICDL training. The results in Table are corresponding to the configuration with output layers at all stages. From Fig. 5.13 (a), it is clearly seen that this configuration lies beyond the break-even point where the overall gain fails to compensate for the penalty of the additional output layer. Hence, we observe more #OPS for CDL (with output stages at every convolutional layer) than the baseline DLN. However, it is important to note that for all other configurations we obtain computational savings as well as an improved error rate with respect to the baseline.

Table 2.3.
Classification Error (%) for MNIST and CIFAR10

Dataset	**DLN**	**Integrated CDL (ICDL)**
MNIST	0.53% [56]	0.44%
CIFAR10	11.68% [45]	10.76%

In order to find the maximum testing accuracy attainable with ICDL training and to compare our results with other state-of-the art DLN methods, we evaluate the configuration with output stages at every convolutional layer. Table compares the overall classification test performance of the DLNs in [45,56] against the corresponding models trained with integrated CDL for MNIST and CIFAR10. It is clearly seen that the error rate decreases with our proposed training by 16.98% for MNIST and 7.87% for CIFAR10 with respect to the standard DLN architectures.

With the training methodology in Section 3.3, we observed that the enhancement in accuracy was 1.4-1.5% for both the datasets. The integrated training improves the gradient convergence behavior of the DLN considerably in contrast to the approach where a CDL is built on a trained DLN (Algorithm 3.5.5). Thus, we observe significantly larger improvement with integrated CDL than that of the previous approach.

Please note that our goal here is not to get the best results in terms of classification accuracy but rather show the benefits that our proposed training would provide over standard convolutional training. Fig. 5.15 (a) shows a comparison of testing error between integrated CDL and the standard DLN in [38] with increasing epochs of training. We can clearly observe that integrated training gives better results over standard DLN even at lower epochs. This establishes the usefulness of our training methodology.

To observe the convergence behaviour with integrated training, we plotted the average gradient of the absolute value of all weigt!s (ψ in Algorithm 3) at every training epoch for both the the MNIST architecture (DLN, ICDL) in Table 3.5.1 as shown in

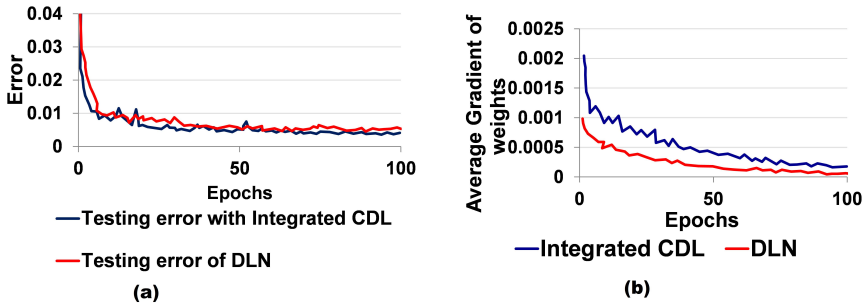


Fig. 2.14. (a) Testing error of DLN vs. Integrated CDL with increasing epochs of training (b) Averaged absolute value of gradient of weigt!s of the entire CDL (Table) for MNIST learnt with integrated training and DLN with standard training with respect to the number of training epochs

Fig. 5.15 (b). We observe that the gradient that is to be backpropagated converges to a miniscule value of 5.2e-5 at 100 epochs for the DLN. In contrast, the gradient with our proposed training is an order of magnitude greater (1.6e-4) at the same epoch. This shows that integrated CDL results in better convergence reducing the vanishing gradient issue. Finally, we note that this approach does not incorporate optimization techniques like dropout [57] or averaging [43]. We believe that combining our approach with these techniques would result in further enhancement in classification accuracy.

2.8 Conclusion

Deep learning convolutional networks are vital for many computer vision applications and demand significant computational effort in modern computing platforms. Here, we explore a novel approach to optimize conventional deep learning networks by employing the convolutional layer features to discriminate between easy and hard input data. We propose the concept of Conditional Deep Learning in which easy instances can be classified earlier without activating the latter layers of the network. We achieve this by cascading a linear network of output neurons for each convolutional

layer and monitoring the output of the linear network to decide if the classification can be terminated at a current layer or not. The design methodology implicitly varies the number of stages or layers used for classification based on the difficulty of the input and produces a CDL with optimal efficiency. To quantify the potential of CDL, we designed the CDLN with state-of-the-art deep learning architectures for the MNIST/CIFAR10/Tiny ImageNet dataset. Our experiments show considerable improvements in energy-efficiency. In addition to energy benefits, our results show that the CDLN yields better classification accuracy as compared to the corresponding baseline DLN. Inspired by this observation, we used the main idea of CDL (i.e. to append output layers for each convolutional layer) to introduce Integrated CDL training wherein the errors at the additional linear classifiers are used in the training process to construct a CDLN from scratch. The classification performance results on MNIST and CIFAR10 datasets with integrated training show significant accuracy improvements.

3. ENERGY-EFFICIENT OBJECT DETECTION USING SEMANTIC DECOMPOSITION

3.1 Introduction

Object detection is one of the core areas of research in computer vision [58]. A detection task is basically a classification problem of distinguishing an object of interest from a host of input data. Traditionally, a single complex classifier model (shown in Fig. 8.1 (a)) is used to perform detection. Here, all the inputs are processed through the single model to detect the object of interest. However, in order to scale to more challenging object detection problems, the classifier models must become larger, which implies an increase in computational resources. With computational efficiency becoming a primary concern across the computing spectrum, energy-efficient object detection is of great importance.Interestingly, we note that in a real world dataset, a major portion of input images have some characteristic broad semantic features like color, texture etc. that are common to the object of interest. Consider the simple example of detecting a red Ferrari from a sample set of vehicle images consisting of motorbikes and cars. The first intuitive step is to recognize all red vehicles in the sample and then look for a Ferrari-shaped object from the sub-sample of red vehicles.Thus, we can reduce and simplify the original sample set by utilizing the semantic information as we progress towards the primary object detection task. Based on this idea, we introduce semantic decomposition of inputs into characteristic broad features, like color (red) or shape (car) in the above example, and using the representative semantics to build a hierarchical classification framework, with increasing levels of complexity, for faster and more energy-efficient object detection.

Fig. 8.1 illustrates our methodology. In the traditional approach shown in Fig. 8.1(a), a single classifier (Classifier X) clearly needs to be highly complex (more hidden

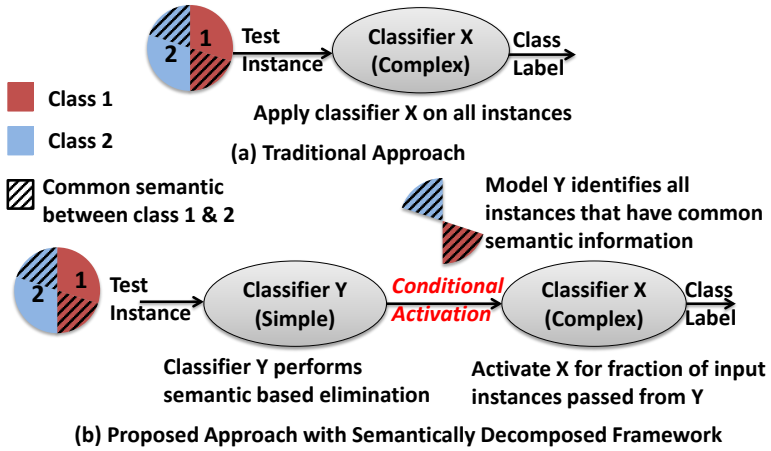


Fig. 3.1. (a) Traditional approach: learn one complex classifier and apply to all instances (b) Proposed approach: learn multiple simple classifiers on semantically decomposed features and activate complex model conditionally for instances that have common features with the object of interest. The simple classifiers in the first stage perform semantic based elimination.

neurons and hence, more synapses) in order to separate the classes with high accuracy. However, this leads to high computational effort for not only the test instances that have common semantic between the two classes but also the ones that do not share common features across the class labels. In contrast, Fig. 8.1(b) shows our approach where we create a semantically decomposed framework with multiple classifiers (Y and X) with varying levels of complexity.

Classifier Y is trained to identify all those instances that share the particular semantic with our object of interest (Class 2). It receives important yet simple semantically decomposed characteristics like color, edges, etc. from the input sensor data. The decomposed input features are simpler and easy to process than the original input image. Thus, the classifier in the first stage (Y) of the proposed framework is less complex with few neurons and synapses. The classifier X is then conditionally activated only for those instances that have the semantic information that model Y is trained to detect. Hence, a significant portion of clutter (Class 1) are filtered out

or eliminated at the first stage leading to energy savings. Please note that since the proposed methodology adds an extra classifier (first stage) into the overall classification framework, the additional cost overhead for the instances that are processed by both stages has to be taken into account in the computational cost.

In order to observe maximum benefits and overcome the cost penalty (that the addition of first stage imposes), it is evident that the input dataset should have significantly larger clutter fraction than the objects of interest. Fortunately, in many useful detection applications, only a small fraction of the input dataset has relevant objects of interest. In [59], the authors have quantitatively established that in a wide range of video-based object detection datasets, only 5% of the input data contains the relevant objects of interest. Our approach exploits this disproportionate distribution of input data to obtain compute-efficiency. It is worth mentioning that our hierarchical classification methodology is complementary to the concept of cascading classifiers proposed in [35, 60]. However, the novelty of our work arises from the fact that we leverage the semantic information to develop a systematic algorithm that automatically detects the characteristic features underlying the input data to perform semantic-based elimination in a multi-stage framework. On the algorithmic front, using multiple classifiers has been an active area of research [61, 62], however, with the primary aim of improvement in accuracy. The use of multiple classifiers in our methodology is entirely driven by energy-efficiency and reduced computational complexity.

3.2 Semantically Decomposed Object Detection

3.2.1 Semantic Decomposition of Input Data

Here, we use color and texture information individually in a set of experiments described in Section 3.4 as the first step of eliminating objects that do not share common semantic information. We use Hue-Saturation-Value (HSV) transformation [63] and Gabor filtering [64] to extract color and texture components, respectively.

Note that, after applying an HSV/Gabor transformation, an image in the HSV/Gabor space is much smaller as compared to the RGB space. The extracted feature vectors are then used as training instances to train the simpler (or less complex) classifiers in the first stage. It is worth mentioning that the additional cost of HSV or Gabor processing also has to be taken into account for energy computations. While we use color and texture as characteristic semantics, please note that other semantics like edges (with cany or sobel detectors), corners and blobs (with Laplacian of Gaussian) can also be used with the proposed methodology.

3.2.2 Semantic based Elimination: Concept

Fig. 8.2 (a) shows the conceptual view of the framework. In the 2-stage framework, each of the ANNs in the first stage are computationally efficient as they are trained on the optimal simple semantic feature vectors extracted from the original RGB image. The ANN in the second stage has a higher complexity on account of being fed the original RGB image for classification. Depending upon the output of the ANNs in the first stage, the second stage is enabled. The final classifier, same as the NN in the traditional structure, makes sure that any clutter that are passed onto the stage by the former ANNs due to misclassification are properly discarded or, classified as clutter in this stage, thereby maintaining the same classification accuracy as the traditional single classifier.

Besides the ANNs in the hierarchy, the setup also contains an activation threshold module (Fig. 8.2(a)). This module decides if the second stage should get enabled or not to determine the final output of the hierarchy for a given input image. Note that if the input is the desired object we are trying to detect, it will always be passed to the second stage. Then, the output of the hierarchy is based on the classification result of the second stage. Only when any clutter image is presented, the module then decides based on the confidence level of the output produced in the first stage whether to activate the second stage.

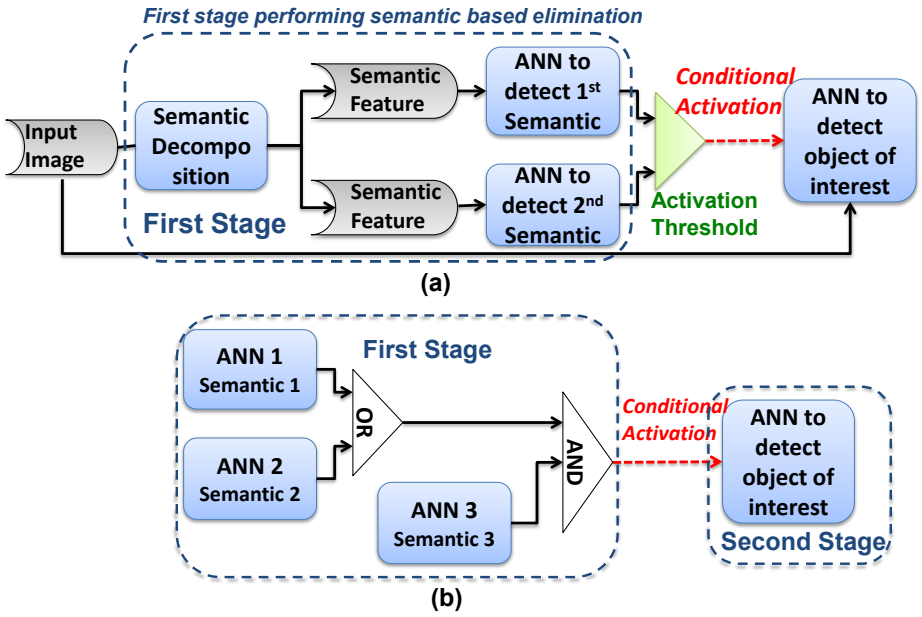


Fig. 3.2. (a) Proposed hierarchical structure with semantic decomposition where multiple classifiers in the first stage detect the appropriate semantic followed by single complex classifier in the second stage enabled by the activation threshold (b) 2-level OR-AND first stage configuration with 3 optimal semantic features

To make our proposed approach more systematic, we devise an algorithm that recognizes the most optimum features and constructs a 2-level OR-AND configuration of the first stage for the most favorable classification results. Fig. 8.2(b) shows an example of the 2-level OR-AND first stage configuration for a given input dataset with 3 optimal semantic features. The given configuration implies that the second stage is only enabled when the first stage detects either Semantic 1 or 2 (OR) in combination with Semantic 3 (AND). In other words, the final ANN is enabled if ANN 3 and either of ANN 1 or 2 produce sufficient confidence level for a given input.

To better understand the need for 2-level OR-AND configuration, consider the example shown in Fig. 6.2. Fig. 6.2(a) shows that certain instances in the objects of interest have one semantic in common while the rest have the second semantic. So, we can choose the OR operation where the second stage is enabled when we get a

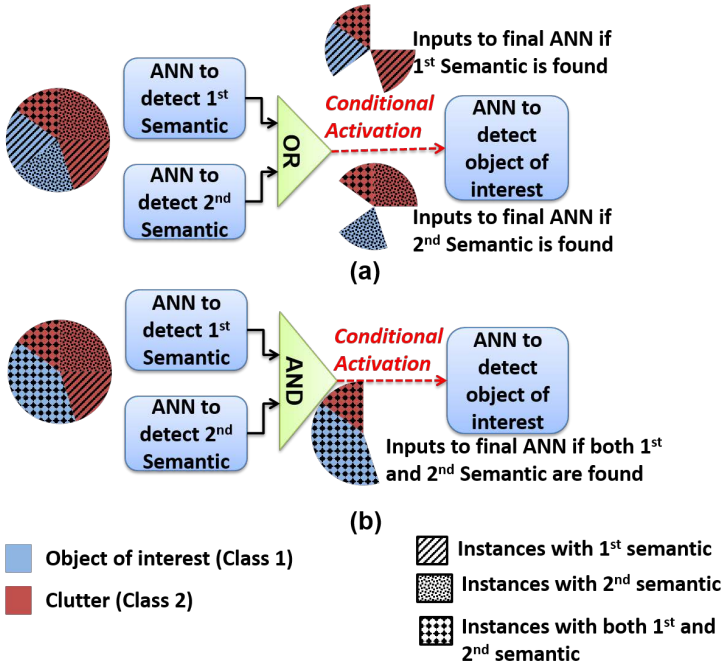


Fig. 3.3. (a) Two different semantic features not shared among all objects of Class 1 leading to OR operation for maximum accuracy (b) Two different semantic features common across all objects of Class 1 leading to AND operation for maximum accuracy as well as optimum efficiency

desired output from any one of the NNs in the first stage. If the operation is set to be AND here, certain objects of interest will be rejected or misclassified in the first stage that will result in a significant decline in accuracy. On the other hand in Fig. 6.2(b), both semantics are present in all the instances of objects of interest. While an OR would give a good result i.e. all objects of interest will be classified by the first stage and passed to the final classifier, however, the first stage would also pass a lot of unnecessary clutter resulting in a decline in efficiency. Thus, we need to set the activation as an AND operation where the final NN is enabled for inputs having both semantics i.e. we get a good confidence level for both the NNs in the initial stage. Thus, it is evident that while AND improves the efficiency, OR increases the accuracy of the semantically decomposed framework.

Algorithm 4 Pseudo code for 2-level OR-AND construction of first stage

Input: Baseline classifier N_{orig} with accuracy Q, training dataset D_{tr}, # features in the search space N

Output: First stage NN configuration: $N_{initial}$

Gain calculation: $G = N_{orig} - [N_{initial} + (1 - f).N_{orig}]$, where f is the fraction of instances correctly filtered in the first stage

1: **initialize** $N_{initial}$ = NULL, pairwise = FALSE, $G_0 = \epsilon_1$
2: **for** $i = 1 : N$ // *for each feature vector in the search space*
3: Train a NN (N_i) on the feature vector i using D_{tr} and obtain accuracy, Q' for the hierarchical framework with N_i as first stage and N_{orig} as second stage.
4: **if** ($Q - Q' < \epsilon$ // *if quality constraint is met*
5: $N'_{initial} = N_i$ AND $N_{initial}$, Calculate gain G_i with $N'_{initial}$ in the first stage and N_{orig} as second stage.
6: **if** $G_i > G_{i-1}$ then $N_{initial} = N'_{initial}$ // *if gain improves then admit the new semantic N_i ANDed with the existing first stage*
7: **elseif** ($G_i < G_{i-1}$ && pairwise= TRUE) **TERMINATE** algorithm and return the existing first stage configuration $N_{initial}$ as output. //
8: **end if end if end for**
9: **remove** the semantic vectors already admitted into first stage $N_{initial}$ from the search space. # features in the remaining search space= N' , pairwise = TRUE, Select the top $k(k < N')$ features quality-wise from the N' search space.
10: **for** $i = 1 : \binom{k}{2}$
11: $N'_{initial} = N_1$ OR N_2 //*where N_1 and N_2 are the two NNs corresponding to the semantic pair for the i^{th} combination*
12: $N^{temp}_{initial} = N_{initial}$ AND $N'_{initial}$
13: Obtain accuracy Q' for the hierarchical framework
14: Repeat Steps 5-10 with $N_i \equiv N^{temp}_{initial}$
if quality constraint is met and gain improves, then, $N_{initial} = N^{temp}_{initial}$
15: **if** $N_{initial}$ (current iteration) == $N_{initial}$ (previous iteration) then **continue**
16: **else** GOTO Step 12 and Repeat Steps 12-20
17: **end for**

3.3 Design Methodology

3.3.1 Constructing the 2-stage Hierarchical Framework

Algorithm 3.5.5 shows the pseudo code for selecting the optimal semantics and constructing the 2-level OR-AND configuration for the first stage of the semantically decomposed framework. The process takes the baseline/traditional single ANN N_{orig}, training dataset D_{tr} and the semantic feature search space as input and produces the optimal first stage $N_{initial}$ with appropriate OR-AND configuration. First, we train N_{orig} on D_{tr} and obtain the accuracy Q. Next, we iteratively traverse through the semantic feature space selecting the feature (or combination of features) that improves the gain G while maintaining the quality constraint (lines 2-17). The procedure terminates if adding a particular feature to the first stage doesn't improve the gain of the existing first stage configutation (line 7).

Algorithm 5 Methodology to test the hierarchical framework

Input: Test instance I_{test}, hierarchical framework N_{hier} with appropriate OR-AND configuration for first stage

Output: I_{test} classified as clutter or object of interest

1: Obtain semantic feature vectors of I_{test} that are inputs to the NN(s) comprising the initial stage in N_{hier}

2: If output of the NN(s) in the initial stage of N_{hier} is such that second stage is not enabled, then **TERMINATE** testing and Output =I_{test} classified as clutter.

3: If the initial stage NN(s) produce a sufficient confidence level on the output meeting the OR-AND conditions, then, second stage is activated and Output = Output of final classifier. Please note that the input to the final classifier is the original test instance, I_{test} and not the semantic feature vector.

Initially, we search through the vector space and check the quality and gain constraints for each semantic vector (lines 3-8). The semantics that improve the overall gain are ANDed together and set as the initial stage ($N_{initial}$) (line 6). Next, we eliminate the semantic vectors already admitted into the initial stage and search through the remaining search space for pairwise ORed combinations of semantics from the

top k features that would improve the accuracy and the overall gain (lines 9-11). For datasets where inputs can be characterized by two different semantic features not shared among all the objects of interest (Fig. 6.2 (a)), OR combination is essential for improving the accuracy of the hierarchical framework. It ensures that all objects of interest are passed to the second stage without being eliminated by the first stage. The pairwise OR combination of NNs are then ANDed with the existing first stage (lines 12-14). If accuracy loss of the hierarchical framework with new first stage configuration with respect to the baseline is lesser than certain threshold ϵ (line 4), we check for the gain constraint. If the gain of the new configuration improves over the previous one, we select the corresponding semantic vectors and set the new OR-AND configuration as the first stage of the hierarchical framework (line 14). After updating the first stage, $N_{initial}$, with a pairwise combination (line 14-15), the search space is pruned. We explore through the remaining space for the top k features and continue looking for other combinations that will improve the overall gain of the framework (line 16).

In the HSV space, we have 8 ranges of H that correspond to the 8 major colors [63]. So, we set N=8 and execute Algorithm to select the most optimal individual colors as well as pairwise combinations ($\binom{k}{2}$) for the first stage. For texture, we use the filter-bank approach as discussed in [65, 66]. The initial Gabor space consists of 20 filters [64, 67] corresponding to 5 scales/frequencies and 4 orientations. So, in case of texture selection, we set N=20 and execute Algorithm exploring the individual as well as pairwise combinations of textures to construct the first stage. We set $k = 4$ in Algorithm during color selection from HSV space and $k = 5$ during texture selection from Gabor space.

After obtaining the first stage of the framework, $N_{initial}$, using Algorithm , the baseline classifier N_{orig} is appended to obtain the overall 2-stage hierarchical framework N_{hier}.

Table 3.1.
First Stage Configuration of N_{hier} for CALTECH101

Image	Configuration of first stage	Representations
Soccer Ball	W.B	R:Red;W:White
Bonsai	(Y+R).G	B:Black;Y:Yellow
Lotus	R+Y	G:Green
Sunflower	Y	
Stop sign	R	**COLOR**
Brain	(G1+G2).G3	G1: $(32\sqrt{2},0°)$; G2: $(64\sqrt{2},0°)$
Menorah	G4+G5	G3: $(32\sqrt{2},90°)$; G4: $(32\sqrt{2},90°)$
Revolver	G6.G7	G5: $(64\sqrt{2},45°)$; G6: $(32\sqrt{2},0°)$
Guitar	G8+G9	G7: $(64\sqrt{2},45°)$; G8: $(16\sqrt{2},0°)$
Starfish	G10+G11	G9: $(64\sqrt{2},90°)$; G10: $(16\sqrt{2},0°)$
		G11:$(64\sqrt{2},90°)$ **TEXTURE**

3.3.2 Testing the 2-stage Hierarchical Framework

Algorithm 5 describes the overall testing methodology for the hierarchical framework. Given a test instance, I_{test}, the process classifies it as clutter or the object of interest.

3.4 Experimental Methodology

We have implemented an ANN based image recognition platform for the Caltech101 dataset [56] which is a large colored image dataset containing over 30,000 labeled examples of 101 different natural images. Each classifier used is a feedforward ANN with 3 layers (Input, Hidden and Output). Each of the ANNs are trained using the standard backpropagation algorithm. For up to 50 different images of the dataset, we implemented the hierarchical framework (N_{hier}) trained to recognize the particular object of interest from a host of other images (clutter) exploiting both color and texture based semantic information.

Of the 50 images, the initial stage configurations for 10 different images are shown in Table 3.4. We can see that the first stage is set to different configurations of OR-AND (OR denoted as +, AND denoted as .) by the training methodology described in the previous section for both color and texture. Each of the Gabor filters selected are represented in the table by their corresponding (scale, orientation). The methodology for constructing the hierarchical framework confirms accuracy or quality check with that of the traditional classifier using OR operation and then it optimizes the efficiency using AND. The Gabor filters/colors selected in the process are also the most optimum semantics for the given set of images. In addition to Caltech101, we evaluated our approach on another dataset CIFAR10 [68] which consists of 60,000 colored images belonging to 10 classes. The initial stage configuration for 4 images are shown in Table 3.4.

Here, we used software simulations to obtain classification accuracy and hardware simulations to obtain energy values. For energy evaluation, we specified each classifier on a standard Neuromorphic Engine that serves as an optimised hardware framework for ANNs [69]. The hardware framework was implemented at the register-transfer logic (RTL) level and mapped to 45 nm technology using Synopsys Design compiler. Finally, we used Synopsys Power compiler to estimate energy consumption of the synthesized netlists. For software simulations, we implemented the 2-stage semantically decomposed classification framework for each object detection application in Matlab. We measured runtime for the applications using performance counters on Intel Core i7 3.60 GHz processor with 16 GB RAM. Please note that the software baseline classifier was aggressively optimized for performance.

3.5 Results

3.5.1 Energy Improvement

Fig. 6.3 (a, b) shows the normalized improvement in efficiency with respect to the traditional single ANN classifier (which forms the baseline) for the images of Table

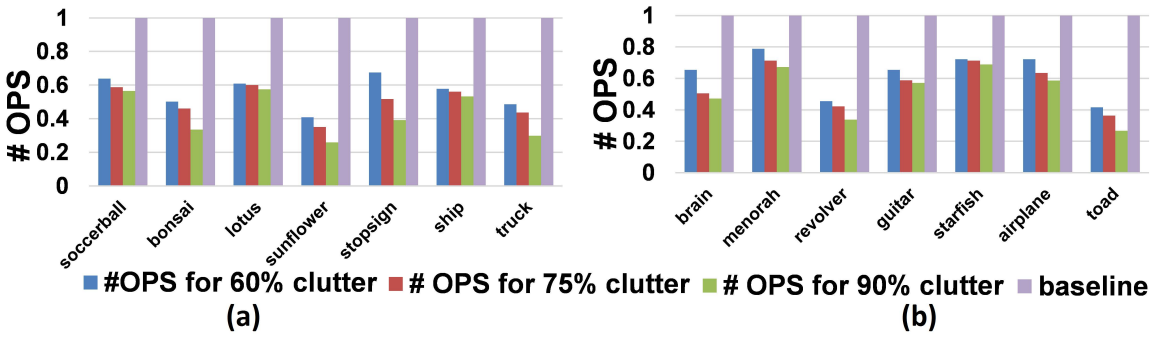


Fig. 3.4. Normalized OPS for images with (a) color as semantic (shaded portion shows the contribution of first stage to the total # OPS) (b) texture as semantic

Table 3.2.
First Stage Configuration of N_{hier} for CIFAR10

Image	Configuration of first stage	Representations
Ship	W+B	R:Red;W:White
Truck	(W+R).B	B:Black **COLOR**
Airplane	(G12+G13)	G12: ($42\sqrt{2}$,22.5°); G13: ($36\sqrt{2}$,67.5°)
Toad	G14.G15	G14: ($10\sqrt{2}$,90°); G15: ($28\sqrt{2}$,45°)
		TEXTURE

3.4 and 3.5.1. We quantify efficiency as the average number of operations (or total number of MAC/ Multiply and Accumulate computations) per input (OPS). As mentioned earlier, there is a significant disproportion in the distribution of input data [9]. Thus, in our experiments we evaluated our approach by varying the fraction of clutter (60%, 75% and 90% non-objects of interest) in the input data for an object detection task. We observe that the hierarchical framework provides between 1.97x-2.64x (average: 2.31x) improvement in average OPS/input compared to baseline across the 10 different images for Caltech. For CIFAR, the average reduction in OPS corresponds to 1.88x across the 4 different images. Note that the benefits vary depending on the fraction of clutter in the dataset.

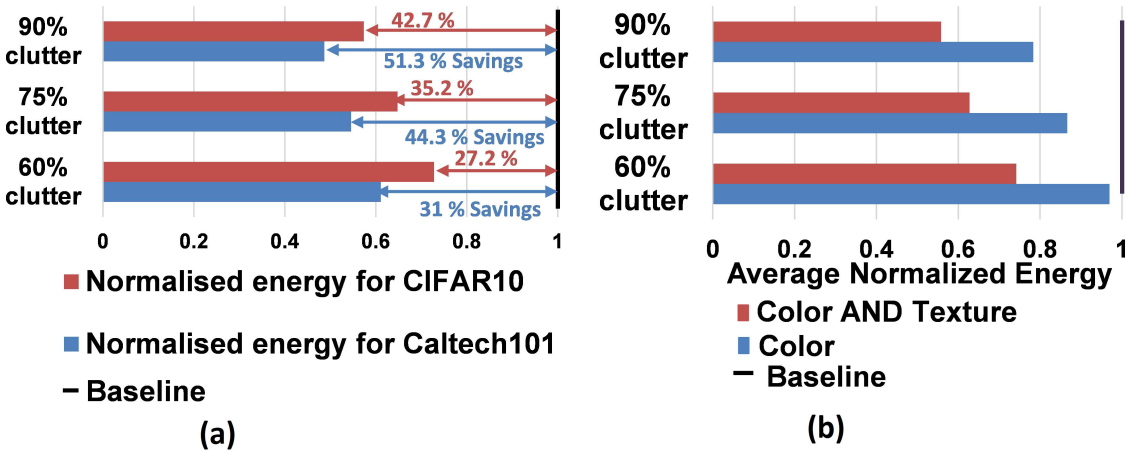


Fig. 3.5. (a) Average hardware energy for different fraction of clutter in the dataset (b) Average normalized energy for both configurations: color only and combined color/texture

Fig. 6.3 clearly illustrates that maximum benefit for each image is observed when the fraction of clutter is 90%. This can be corroborated to the fact that the initial stage filters out a lot of the object of interest. In case of hardware implementation, the reduction in OPS for Caltech translates on an average to 1.64x-2.05x (average: 1.93x) improvement in energy with variation of clutter as illustrated in Fig. 8.7 (a). For CIFAR10, the average improvement in energy with variation of clutter is 1.46x. For software implementation, we observed that the reduction in OPS converts to an average of 2.23x/1.95x improvement in runtime for Caltech/CIFAR10 respectively. Note, the hierarchical framework across all experiments for both datasets maintains an iso-accuracy with that of the baseline classifier (i.e. 97.8% for Caltech101, 78.2% for CIFAR10).

3.5.2 Combining Color and Texture in the Initial Stage

For a given dataset, Algorithm first constructs the individual color/texture configurations. Then, the individual color/texture stages are ANDed together. In case, the overall gain of the hierarchical framework improves with the ANDed configuration, the color(AND)texture combination is selected as the first stage of the hierarchy. For

Table 3.3.
First Stage Configuration

Image	Configuration with color & texture	Configuration with color only
Motor bike	(G1+G2).B	B
Buddha	(G3+G4).Y	Y
Flamingo	(G5+G6).R	R

certain images in the Caltech101 dataset shown in Table 3.5.2, such configuration was chosen.

In order to observe the additional benefits with the color/texture ANDed configuration and for comparison purpose, we implemented a separate semantically decomposed framework using only color configuration obtained from Algorithm . Fig. 8.7 (b) shows the average energy in both cases (color, color AND texture) as the clutter fraction is varied. It is clearly seen that color AND texture configuration gives more savings than the latter. This is due to the fact that the benefits of reduced final stage activation in the combined case overcomes the penalty due to the addition of texture configuration in the first stage. Thus, our proposed design methodology ensures maximum cost savings by selecting the most optimum semantic configuration.

3.5.3 Optimizing the complexity of the first stage

The hierarchical design methodology first meets the output quality or accuracy constraint and then optimizes the framework to get maximum efficiency. In order to get the most benefits, we need to filter out more clutter in the initial stage. We can achieve this by increasing the complexity of the first stage by adding more neurons to the hidden layer. Fig. 6.5 (a) shows the normalized energy of the entire hierarchical framework as the complexity of the first stage (R+Y configuration from Table 3.4) is varied for detecting lotus from the Caltech101 dataset. It can be clearly seen that the amount of clutter filtered increases with the increasing complexity of the first

stage. So, as the initial stage becomes more complex, the final stage is enabled for fewer clutter data from the total fraction of clutter. Thus, in the beginning, we observe a decreasing trend in energy. However, the increasing complexity of the first stage would also add an additional overhead to the cost computation that would at some point overcome the total cost savings. This break-even point corresponds to the maximum benefits or the lowest energy that we can achieve using the hierarchical framework for this particular example. Beyond this point, the cost increases. In Fig. 6.5 (a), we see that the break-even point corresponds to 0.508 (Normalized energy) which translates to 1.97x improvement in computational cost. This behavior is taken into account in our design methodology described in the previous section.

3.5.4 Efficiency-Accuracy Tradeoff using Confidence level (δ)

In Section 3.2, we discussed that the confidence level or activation threshold (δ) can be regulated to modulate the amount of clutter being passed to the final classifier and further optimize the efficiency while maintaining comparable accuracy with that of the baseline. Fig. 6.5 (b) shows the normalized energy of the hierarchical framework as the accuracy of the first stage is varied by changing the δ value for the Caltech 101 dataset. The tradeoff analysis helps us attain the most optimum δ of a semantically decomposed framework for an object detection task. Setting δ to a low value implies that more clutter will now be misclassified by the first stage, and forwarded to the final classifier. Increasing δ would result in lesser clutter being misclassified thus improving the overall accuracy of the first stage as can be seen from Fig. 6.5 (b).

Beyond a particular δ, the objects of interest will be misclassified and filtered out. This δ value corresponds to the maximum overall accuracy of the first stage in the hierarchy. In Fig. 6.5 (b), we observe that as the normalized accuracy value increases from 0.86 (δ=0.1) to 0.95 (δ=0.4), there is a 2.25x improvement in energy-efficiency. In this case, beyond δ=0.4 the accuracy declines and hence those δ values are not considered. Please note that, the energy benefits will continue to increase beyond δ =

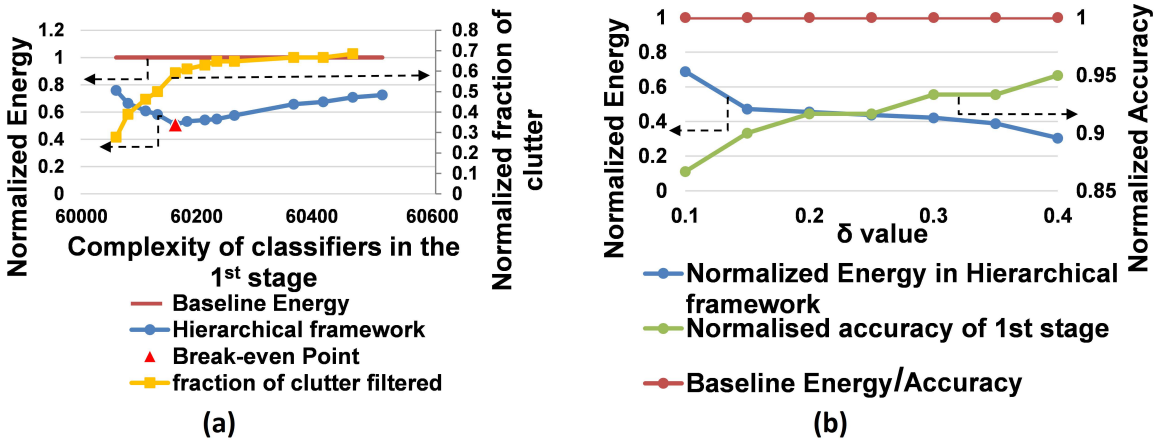


Fig. 3.6. (a) Normalized reduction in energy by varying complexity of 1st stage in N_{hier} for detecting lotus from Caltech101 dataset (b) Normalized energy vs. accuracy tradeoff by modulating confidence level (δ)

0.4 as the second stage of the hierarchy is enabled for less instances with increasing δ. Thus, for applications that permit tolerable accuracy loss, we can use higher values of δ to get higher energy benefits. Please note that the accuracies shown in Fig. 6.5 (b) are normalized with respect to the baseline accuracy (∼97.8% in this case). For our OPS evaluations with Caltech 101 and CIFAR10 (Fig. 4), we use δ =0.4 and 0.55 respectively as obtained from the tradeoff analysis. Thus, the δ value serves as a powerful knob to modulate the efficiency-accuracy tradeoff which can be easily adjusted during runtime to get the most optimum results.

3.5.5 Impact of addition of first stage to the overall training time

Until now, the energy benefits observed correspond to the reduction in testing complexity for a given task using our proposed hierarchical framework. However, the first stage also adds an additional overhead on the total training time to construct the 2-stage framework. Fig. 6.6 shows the overall normalized training time of the hierarchical framework for detecting the 10 images of Table 3.4 (Caltech 101). The first stage training time shown includes the additional time expended during the iterative optimal semantic selection process (*Algorithm* in Section 3.3). It is clearly seen that

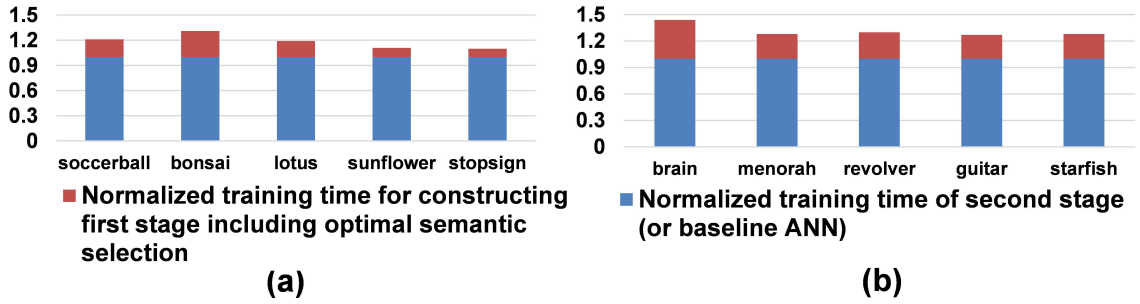


Fig. 3.7. Normalized training time for images with (a) color as semantic (b) texture as semantic

the overall training time of the hierarchical framework is greater than the baseline single stage classifier for each image. We also observe that the first stage construction with color features (Fig. 6.6 (a)) takes lesser time than that of texture (Fig. 6.6 (b)). This can be attributed to two factors: a) Gabor filtering is computationally more expensive involving complex operations than HSV. b) In *Algorithm* , for color features, we only explore the 8 feature search space to select the optimal color in comparison to 20 features for texture. The increased search space further adds to the training time. On an average, we observe that the training time increases by 18.4% in Fig. 6.6 (a) and 31.4% in Fig. 6.6 (b) compared to the baseline traditional classifier. While there is a training overhead with our proposed framework, in typical object detection applications, training is performed only once or very infrequently. Testing, on the other hand, is done more frequently over longer periods of time. Since our proposed framework yields significant reduction in testing energy, a small increase in training cost is a favorable tradeoff.

3.6 Conclusion

In this chapter, we presented a systematic approach to optimize energy-efficiency of machine learning classifiers in object detection applications.We use the common semantic features observed across images in real-world datasets to distinguish the objects of interest from the remaining inputs at lower complexity with 2-stage classifica-

tion. Based on the above insight, we proposed the concept of hiererchical classification based on semantic decomposition. We developed a systematic methodology to implement a 2-stage semantically decomposed classification framework using color/texture as semantic information. We achieve this by arranging the classifiers (ANNs) in increasing order of complexity as per the characteristic semantic features they are trained to recognise. The design methodology is equipped to implicitly gather the most appropriate semantic information for optimum efficiency. To quantify the potential of semantic decomposition, we used color and texture as a basis for segmentation and designed the hierarchical framework for object detection for various images of the Caltech101/CIFAR10 dataset. Our experiments demonstrate significant improvement in energy over hardware implementation with respect to traditional approach. We use color and texture as distinctive traits to carry out several experiments for object detection. Our experiments on the Caltech101/CIFAR10 dataset show that the proposed method yields 1.93x/1.46x improvement in average energy over the traditional single classifier model for a minimal increase in training time. Finally, we would like to note that while our 2-stage framework is composed of simple ANNs, the methodology and the underlying feature-based elimination strategy can be extended to deep learning models for more complex recognition/detection tasks.

4. UNSUPERVISED REGENERATIVE LEARNING OF HIERARCHICAL FEATURES IN SPIKING DEEP NETWORKS FOR OBJECT RECOGNITION

4.1 Introduction

Deep Learning Networks, inspired by the cortical visual processing systems, have seen increasing success in recent years due to the availability of more powerful computing hardware (GPU accelerators) and massive datasets for training. Regardless of their success, the substantial computational cost of training and testing such large-scale networks has limited their implementation to clouds and servers. In order to build devices with cognitive abilities, there is a need for specialized hardware with new computational theories. Spiking Neural Networks (SNNs) are a prime candidate for enabling such on-chip intelligence.

Driven by brain-like asynchronous event based computations, SNNs focus their computational effort on currently active parts of the network, effectively saving power on the remaining part, thereby achieving orders of magnitude lesser power consumption in comparison to their Artificial Neural Network (ANN) counterparts [70, 71]. In 2014, IBM research demonstrated a large-scale (>1Million neurons & 256 Million synapses) digital CMOS neurosynaptic chip, TrueNorth [72], which implements a network of integrate-and-fire spiking neurons. However, TrueNorth does not incorporate any information pertaining to the learning mechanisms, which is at present a major constraint for realizing SNNs for real-world practical applications like visual and speech recognition among others. Thus, there is a need to develop efficient learning algorithms that might take the advantage of the specific features of SNNs (event-driven, low power, on-chip learning) while keeping the properties (general-purpose, scalable to larger problems with higher accuracy) of conventional ANN models.

Recent efforts on training of deep spiking networks do not use spike-based learning rules, but instead start from a conventional ANN fully trained using labeled training data, followed by a conversion of the same into a model consisting of simple spiking neurons [73–75]. However, in order to extend the applicability of learning methods, the use of unlabeled data for machine learning is imperative. In the non-spiking domain, unsupervised learning of hierarchical regenerative models such as Auto-Encoders [76] have been successfully used to learn high-level features. The learnt features are then used as inputs to a supervised classification task [77] or to initialize a CNN [78] to avoid local minima. We develop upon Auto-Encoders where we build a spiking deep CNN by training each layer in the hierarchy in a purely unsupervised manner using the regenerative model and the temporal spike information to update the weights.

SNNs are equipped with unsupervised weight-modification rules like Spike Timing Dependent Plasticity (STDP) that learn the structure of input examples without using labels [79]. However, the network structure that has been successful in achieving competitive classification accuracy for pattern recognition problems is a single-layer SNN, which does not scale well to realistic sized high dimensional images in terms of computational complexity [80]. Moreover, STDP does not support learning hierarchical models that simultaneously represent multiple levels like edges or object parts in a visual context that is fundamental to a deep learning model. We propose regenerative learning using spike-timing information to implement layer-wise weight modification that learn representative levels for each convolutional layer. The features from the final layer of the deep convolutional spiking network (SpikeCNN) are then used for classification tasks.

4.2 Preliminaries

4.2.1 Convolutional Neural Networks

CNNs have proven to be very successful frameworks for image classification tasks [29, 40, 41]. Fig. 8.1 shows the basic CNN structure. They are mainly composed of three main blocks: convolutional layer, spatial sampling/pooling layer and a fully connected layer. The weights of a CNN are convolution kernels. A convolution layer convolves a portion of the previous layer with a set of weight kernels to obtain an array of output maps. The output maps are given by

$$x^k = f(\Sigma_l W^k * x^l) \tag{4.1}$$

where f is the neurons activation function, x^k denotes the activation value of the neurons in the output maps $k(k = 1, 2n)$, x^l denotes the activation of the neurons in a previous layers map l, W^k are the set of weight kernels and * denotes a 2D-valid convolution operation. The weight kernels are replicated and moved portion-wise over the whole input map. This sharing of weights significantly reduces the number of parameters to be learnt during the training process and thus enables the CNN to be scalable to high-dimensional images. The CNNs are trained in a supervised fashion (showing the training labels) by using standard backpropagation to train the convolutional weight kernels along with the fully connected weights for the final output layer as described in [48].

4.2.2 Unsupervised learning with Auto-Encoders

Unsupervised learning methods are generally used to extract useful features from unlabeled data, to detect oversimplified input representations and remove input redundancies and finally to obtain only robust and discriminative representations of the data. In fact, deep neural network architectures have been built by stacking layers

trained in an unsupervised way. This is done to avoid local minima and to increase the network performance [28, 81, 82].

Auto-Encoder (AE) methods are one of the most widely used unsupervised feature extractor models for ANNs (non-spiking). AE models are based on the encoder-decoder principle [78]. The input is first transformed into a lower dimensional space (encoder), and then expanded to reproduce the original input (decoder) as shown in Fig. 8.2. This captures the non-linear dependencies in the input. Each layer is trained on the above principle by feeding the activation from one layer to the next. The basic mathematical formalisms for an AE based training of an ANN are as follows [83]:

$$h = f_\theta = \sigma(Wx) \tag{4.2}$$

$$h = f'_\theta = \sigma(W'x) \tag{4.3}$$

$$W' = W^T \tag{4.4}$$

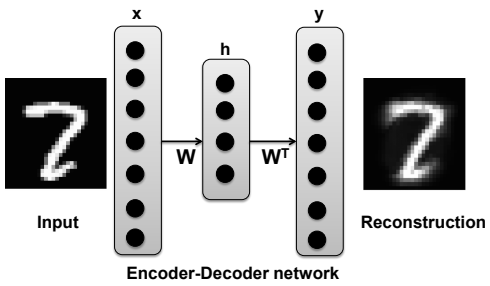


Fig. 4.1. Auto-Encoder network with input and reconstructed pattern

For a given input x, the AE first obtains the hidden representation, h using the neuron activation function denoted as σ. This activation value, h, is then reverse mapped to reconstruct the input. The weights used in the reverse mapping is generally the transposed form of the weights connecting the input and the hidden layer. Thus, the method uses the same weights for encoding the input and decoding the hidden values, thereby reducing the number of parameters to be learned in the training process. The parameters are then optimized to minimize the error for each training

pattern x_i and its reconstruction y_i. Note, the training labels are nowhere used in the weight update process. Hence, the method is completely unsupervised. Inspired by the reconstructive model of training, we use the auto-encoder model to update the weights for each convolutional layer by tracking the spike information at the input and modifying the weights such that the original input spike pattern is reproduced. This enables the network to learn hierarchical representative features of the input data.

4.3 Deep Spiking Convolutional Network: Learning and Implementation

4.3.1 Spiking neuron model

Unlike conventional ANNs where a vector is given at the input layer once and the corresponding output is produced after processing through several layers of the network, SNNs require the input to be encoded as a stream of events. At a particular instant, each event is propagated through the layers of the network while the neurons accumulate the events over time causing the output neuron to fire or spike. Thus, the spike information is used to communicate between the layers of the network. The spiking neuron model used in this work is the Leaky Integrate-and-Fire (LIF) model [84]. The membrane potential $v_{mem}(t)$ of a post synaptic neuron is given by

$$\tau_{RC}\frac{dv_{mem}(t)}{dt} = -v_{mem}(t) + J(t) \tag{4.5}$$

where $J(t)$ is the input current and τ_{RC} is the membrane time constant. The neuron fires when the membrane potential v_{mem} crosses a certain user-defined threshold v_{th}. Once a spike is generated, the membrane potential of the neuron is set to the reset potential, v_{res}, for a refractory period of τ_{ref}. Once the refractory period is complete, the neuron follows the response shown in equation (5.5). In our simulations, we discretize the above continuous time equation into 1 ms time steps.

The total synaptic input current received by a neuron is

$$J(t) = \Sigma_i(\Sigma_{s\in S_i} w_i \delta(t-s)) \tag{4.6}$$

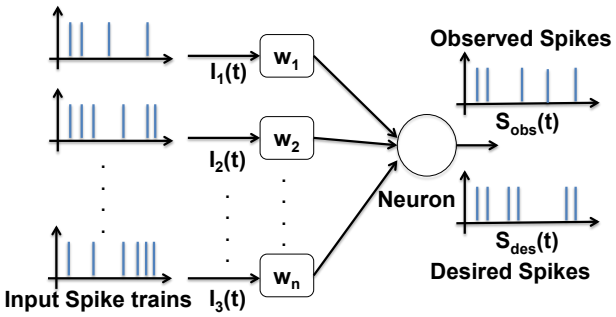


Fig. 4.2. The learning problem identifies the optimal weight vector so as to achieve the desired spike train from the given input spike pattern [85]

where w_i is the synaptic efficacy of the i^{th} synapse, δ is the delta function that contains the time of arrival of spikes at the i^{th} synapse denoted by $S_i = \{t_i^0, t_i^1 ...\}$.

4.3.2 Error Backpropagation in SNNs

Weight update in Convolutional Deep Learning Networks follow the convolutional backpropagation algorithm which is an extension of the standard stochastic gradient descent rule for feedforward ANNs in the convolutional context [53]. As discussed earlier, in this work, we use the regenerative learning method inspired from AEs to train the hierarchical convolutional layer features of a deep spiking convolutional network (SpikeCNN). Similar to ANNs, in the regenerative learning for deep SNNs, the backpropagation algorithm with gradient descent is employed to update the weights to enable unsupervised layer wise training.

Learning theory

The learning problem for spiking neurons is illustrated in Fig. 8.2. There is a spiking neuron that receives input spike trains through n synaptic connections. The objective is to evaluate the n-dimensional weight vector $\mathbf{w} = [w_1 w_2 ... w_n]^T$ for the

neuron to produce the desired spike train $S_{des}(t)$. Let the desired spike train be given by

$$S_{des}(t) = \Sigma_i \delta(t - t^i_{des}) \tag{4.7}$$

where δ is the Dirac delta function and $t^1_{des}, t^2_{des} ... t^k_{des}$ are the desired spike arrival times. We now require a learning rule to identify the changes in synaptic weights of the neuron to achieve the desired output spike pattern using the spike information for weight updates.

Cost Function

The weight update in ANNs follows the backpropagation algorithm aimed at minimizing a cost function. Similarly, in SNNs we need to define a cost function that would drive the learning process. Assume $V(t)$ is the membrane potential of the neuron, with synaptic weight vector $\mathbf{w}$, when input spike trains are fed to it. Correspondingly, let the neuron issue spikes at time instants $t^1_{obs}, t^2_{obs} ... t^{k'}_{obs}$. Hence, in accordance with (5.7) the observed spike train can be denoted as

$$S_{obs}(t) = \Sigma_i \delta(t - t^i_{obs}) \tag{4.8}$$

Now the error function can be defined as

$$e(t) = S_{des}(t) - S_{obs}(t) \tag{4.9}$$

However, the desired spike train is not known in an unsupervised learning process. In non-spiking AE models, encoder-decoder method [78] is used to calculate the error that is back propagated through the layers without showing any training labels. Hence, the AE based training is completely unsupervised. Similarly, in the regenerative layer-wise training of SpikeCNN, we add another output layer in addition to the input and the convolutional layer which will be interpreted as a *pseudo-visible layer* that should ideally imitate the input layer patterns as shown in Fig. 6.2. This will enable us to update weights for the intermediate layers without showing any training

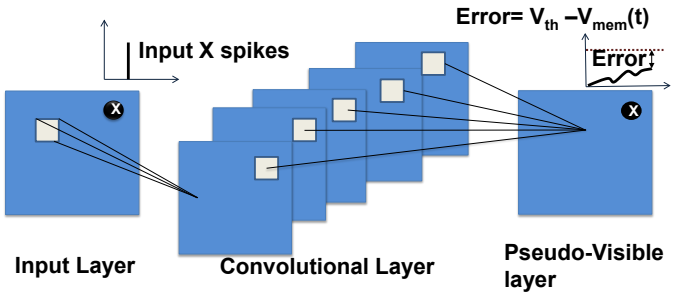


Fig. 4.3. Layer-wise training of a convolutional layer using Regenerative Learning. If a neuron X in the input layer spikes at a given time instant, the regenerative learning model updates the weights in such a way that the neuron X in the pseudo-visible layer also spikes. This is achieved by propagating the error calculated at the pseudo-visible layer using gradient descent

labels. In a non-spiking model, the weight update rule is driven by the activation values of the output neurons. In the spiking context, the activation values of an output neuron can be interpreted as its membrane potential. If an input neuron spikes at a given time instant, the corresponding neuron in the output layer should also spike as per the regenerative approach. This can only be achieved if the membrane potential of the output neuron crosses the threshold.

Depending upon the input spike pattern, the error is defined as follows:

- If an input neuron spikes at a given time instant, the error is calculated as the difference in the threshold value (v_{th}) and the membrane potential (v_{mem}) of the corresponding output neuron in the pseudo-visible layer.
- If the input neuron does not spike, the error is evaluated as the difference in the reset value (v_{res}) and the membrane potential (v_{mem}) of the corresponding output neuron in the pseudo-visible layer.

Thus, the error function can now be indicated as

$$e(t) = V_{des}(t) - V_{obs}(t) \tag{4.10}$$

where V_{des} is v_{th} or v_{res} depending upon the spike event at the input neuron. The cost function corresponding to the synaptic weight vector $\mathbf{w}$ can now be defined as

$$C(w) = \frac{1}{2}\int_0^T (V_{des}(t) - V_{obs}(t))^2 dt \tag{4.11}$$

where T denotes the duration of the training epoch. The desired weight vector w_{des} is

$$w_{des} = argmin_w C(w) \tag{4.12}$$

The learning process thus takes into account the spike information and the inherent latencies. The learning rule follows the gradient descent optimization where the weights are adjusted depending upon the gradient of cost w.r.t the weights. Note that the sign of the corresponding synaptic weight as determined by the algorithm can be either positive (excitatory) or negative (inhibitory) leading to a corresponding increase or decrease in membrane potential of the neuron.

Approximate gradient descent

The cost function $C(w)$ as given in (4.11) depends on $V(t)$ which has several discontinuities in the weight space. Hence, for simplification as proposed in [85], we minimize the contribution to the cost function at each time instant independently rather than minimizing the total cost over an entire epoch.

The cost-function at time t is obtained by restricting the limits of integral in (4.11) to an infinitesimally small interval about time t. Thus,

$$C(w,t) = \frac{1}{2}(V_{des}(t) - V_{obs}(t))^2 \tag{4.13}$$

Hence, its gradient with respect to w is

$$\nabla_w C(w,t) = -(V_{des}(t) - V(t))\nabla_w V(t) \tag{4.14}$$

Now, the synaptic weight update corresponding to the activity observed at time t is given as

$$\Delta w(t) = -\eta \nabla_w C(w,t) = -\eta(V_{des}(t) - V(t))\nabla_w V(t) \tag{4.15}$$

where η is the user-defined learning rate. Now, the discontinuities in $V(t)$ due to the LIF neuron model will render $\nabla_w V(t)$ undefined for some particular values of w. In [85], the authors have implemented a weight update rule and have used certain approximations to overcome these non-linear dependencies. In this work, we invoke similar approximations that would allow us to replace $\nabla_w V(t)$ with appropriate quantities.

In non-spiking context, the weight update for a synapse connecting neuron i and j [86] (with exponential activation function) is computed as

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}; where \frac{\partial E}{\partial w_{ij}} = \delta_j o_i \tag{4.16}$$

$$\begin{aligned} \delta_j &= E o_j \ if\ j\ is\ an\ output\ neuron \\ &= (\Sigma_l \delta_l w_{jl}) o_j \ if\ j\ is\ a\ hidden\ neuron \end{aligned} \tag{4.17}$$

Here, o denotes the activation value of neuron. Comparing (4.15) and (4.16), $\nabla_w C(t)$ is equivalent to $\frac{\partial E}{\partial w_{ij}}$. Interpreting the activation value of the neuron as the membrane potential in the spiking context, we can now replace o_j in the above equations with the membrane potential of a neuron ($V(t)$) at a given time instant. Thus, the weight update equations for SNN with approximate gradient descent can be written as

$$\Delta w_{ij} = -\eta \delta_j(t) V_i(t) \tag{4.18}$$

$$\begin{aligned} \delta_j(t) &= (V_{des_j(t)} - V_j(t)) V_j(t) \ if\ j\ is\ an\ output\ neuron \\ &= (\Sigma_l \delta_l w_{jl}) V_j(t) \ if\ j\ is\ a\ hidden\ neuron \end{aligned} \tag{4.19}$$

In summary, the approximate gradient descent assumes the LIF neuron model to be an equivalent standard activation model in non-spiking context. The gradient calculation is then carried out by using the membrane potential of a neuron at a given time instant as the activation value. The inherent error-resiliency of these neural networks allows us to use such approximate models.

Regenerative Learning for Spike-based Convolutional Auto-Encoder

Convolutional Auto-Encoders (CAEs) differ from conventional AEs as their weights are shared among all locations in the input preserving spatial locality. The CAE architecture with weight sharing is shown in Fig. 6.2. The reconstruction represented by the pseudo-visible layer is due to a linear combination of basic image patches based on the activations in the convolutional layer.

For a mono-channel input x, the convolutional layer representation of the k^{th} feature map is given by

$$h^k = \sigma(x * w^k) \tag{4.20}$$

Here, h^k denotes the membrane potential of the neuron ($v_{mem}(t)$)in the convolutional layer, σ denotes LIF model discussed earlier to calculate $v_{mem}(t)$ at a given time instant t, x contains the spike information from the input layer neurons and * denotes the convolution operation. When h^k crosses v_{th}, the spikes generated at the neurons of the convolutional layers serve as input for the pseudo-visible layer. The reconstruction is obtained using

$$y = \sigma(\Sigma_k \epsilon(h^k) * \tilde{w}^k) \tag{4.21}$$

where $\tilde{w}^k$ denotes the transpose (or flip in both dimensions) operation, $\epsilon(h^k)$ denotes the spike information corresponding to the convolutional layer and y gives $v_{mem}(t)$ of the neurons in the pseudo-visible layer. The convolution of a mxm matrix with a nxn matrix may result in an $(m-n+1)x(m-n+1)$ matrix (valid convolution) or $(m+n-1)x(m+n-1)$ (full convolution). In our simulations, we perform a valid convolution from input to the convolutional layer and a full convolution from convolutional to pseudo-visible layer in order to get a map of the same size as the input layer. All spike-based models and calculations for implementing convolutional network remain the same as described earlier. In (5.6), the input synaptic current was given by the weighed summation of spike inputs. In this case, the only difference is that the synaptic current is obtained by convolving the spike information of an image patch with the weight kernel.

Please note that the h^k and y are calculated at every time instant of a given epoch and the error is calculated from the difference in the $v_{mem}(t)$ of the neurons in the input and pseudo-visible layer. It is clear that the regenerative learning would activate the neurons in the pseudo-visible layer such that they imitate the input layer spike pattern. Thus, the cost function to minimize is the mean squared error (MSE) given by

$$C(w) = \frac{1}{2n}\Sigma_{i=1}^{n}(v_{des}(t) - v_{mem}(t))^2 \tag{4.22}$$

where n is the total number of neurons in the input /pseudo-visible layer. As mentioned earlier, we minimize the error at each time instant rather than over the entire epoch duration.

Now the approximate gradient descent model discussed previously is used to calculate the weight updates. The gradient of the cost function now involves convolution operations and is given by

$$\frac{\partial C(w)}{\partial W} = x * \delta h^k + \epsilon(\tilde{h}^k) * \delta y \tag{4.23}$$

δh^k and δy are the gradients for the convolutional and pseudo-visible layer neurons which are evaluated using the approximate model. δh^k is calculated using (4.19) for hidden neuron and δy using (4.19) for output neuron. Several such convolutional AEs can be assembled together to construct a deep CNN hierarchy. Each convolutional layer in the hierarchy is trained separately with the regenerative method described above.

Average Pooling

For CNNs, a pooling layer [44,87] is often introduced after the convolutional layers to obtain translational invariance. The average pooling or subsampling layers combine the responses from multiple neurons in the convolutional layer into one. The representation of the averaging layer is identical to (8.1), except that the kernels consist of uniform weights fixed to $1/size(w^k)$, where $size(w^k)$ is the size of the sampling window. After training the convolutional layer with the regenerative method,

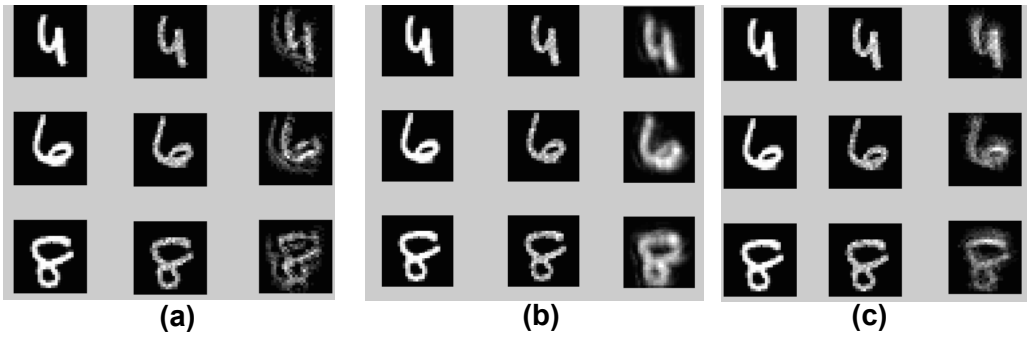


Fig. 4.4. Reconstructed patterns observed after training the first convolutional layer of MNIST_2C with regenerative learning for different v_{th} and I_{rate} values Original pixel image (Column 1), Spike input image (Column 2), Reconstructed image (Column 3) (a) MNIST_2C initialized with v_{th}= 1.0, I_{rate} =100 Hz (b) MNIST_2C with parameters ($P2$) v_{th}= 1.2, I_{rate} =100 Hz (c) MNIST_2C with parameters ($P1$) v_{th}= 0.8, I_{rate} =75 Hz

the membrane potentials (h^k) of the neurons across all feature maps are obtained corresponding to a training input for a given instant of an epoch. The membrane potentials (h^k) within the sampling window are averaged to obtain the output membrane potential of the neuron (p^k) in the pooling layer for the given time instant. When p^k crosses v_{th}, the spikes generated at the neurons of the pooling layer serve as input for training the next convolutional layer. It is clear that the averaging operation down-samples the convolutional layer representation while conserving the spike information and inherent latencies from the previous layer.

The regenerative auto-encoder based learning, thus, trains several convolutional layers in layer-wise fashion which can be assembled together to form a deep hierarchy [88]. Each layer receives its input from the previous layer as described above. The assembled convolutional layers can be used to initialize a CNN with the same topology prior to a fully connected stage.

Supervised training with labels for classification

The Fully Connected layer (FC) at the end of the CNN combines the inputs from the feature maps in the previous layer to perform classification of the overall inputs at

the output layer. The training of the fully connected layer cannot follow the encoder-decoder principle (regenerative learning) as the aim here is to classify rather than to obtain abstract representations of the input. At this stage, the spike information from all end layer maps are concatenated into a vector which serves as input from the FC layer to the output layer.

The training labels are used to fix the spike pattern of the output layer neurons that will drive the error backpropagation to calculate the weight updates. The weight update rules follow the same equations as described in the approximate gradient descent method. For a given label, a Poisson spike pattern of a particular frequency is generated that serves as the desired spike train for the corresponding output neuron for all training inputs with that label. The remaining output neurons have no spike events. The errors are calculated as per the spike events at the output layer neurons. This method ensures that the weights connected to the desired output neuron are potentiated while inhibiting the activity of the remaining connections. In our simulations, we fix the frequency of the desired spike train for a training label at 30 Hz. After training is done, an input is presented as a stream of events at the input layer of the SpikeCNN. At the end of the time duration, the spike responses from the output layer neurons are monitored. The output neuron with the highest response is the class predicted by the network for the given input.

4.4 Experimental Results

4.4.1 Network Architecture and Parameters

We evaluate our proposed learning on two datasets: MNIST [48] and CIFAR10 [50]. MNIST is a standard dataset of handwritten digits that contains 60,000 training and 10,000 test patterns of 28 x 28 pixel sized greyscale images of the digits 0-9. CIFAR10 is a more challenging dataset that consists of 60,000 colored images belonging to 10 classes. Each image has 32x32 pixels. We used the first 50,000 images for training and last 10,000 images for testing. Regenerative learning is used to train

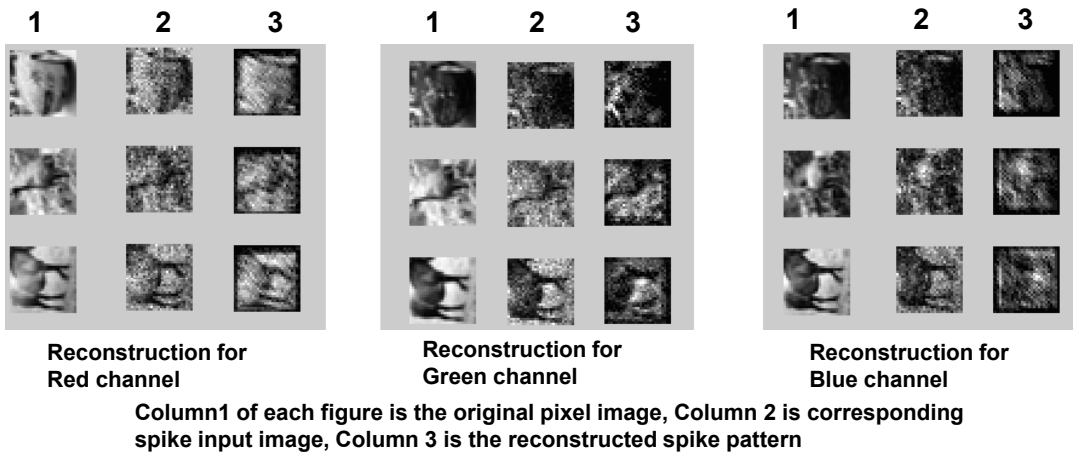


Fig. 4.5. Reconstructed pattern observed after training the first layer of CIFAR_3C initialized with $P2$. CIFAR_3C has 3 maps at the input layer corresponding to the 3 color channels (Red, Green, Blue). The figures show the reconstruction of the input pattern at the pseudo-visible layer corresponding to the 3 separate channels.

spike-based convolutional AEs which are then used to initialize a SpikeCNN with the same topology. The final fully connected layer of the SpikeCNN is then trained with a fraction of training labels from the entire dataset to perform classification. The input is presented as Poisson distributed spike train with firing rates proportional to the intensity of pixels [73]. In the experiments, the epoch duration or the time for which an input image is shown to the network is 250ms. During layer-wise training, each input is presented multiple times (depending upon the depth of SpikeCNN) and the weights are updated each time to minimize the reconstruction error. Since each layer receives its input from the previous trained layer, this process helps in maintaining an adequate firing rate for each layer. This ensures spike propagation as we go deeper into the network. The membrane potentials of all neurons in the convolutional and pseudo-visible layers are all set to v_{res} before presenting a new training pattern.

We implemented a SpikeCNN for MNIST (MNIST_2C) with 2 convolutional layers: 28x28-12c5-2a-64c5-2a-10o. The input layer is 28x28. Both convolutional layers use 5x5 kernel size with 12 and 64 maps, respectively. A 2x2 average pooling window is used after each convolutional layer. The final features from the second averaging

layer are then fully connected to a 10-neuron output layer. SpikeCNN for CIFAR10 (CIFAR_3C) consists of 3 convolutional layers: 32x32x3-32c5-2a-32c5-2a-64c4-10o. In this case, the input layer has 3 maps corresponding to the 3 color channels RGB. The first and second convolutional layer have 5x5 sized kernels with 32 maps while the third layer has 4x4 kernel with 64 maps. The features from the third layer are directly fed to the output layer without any average pooling. Please note that we do not use any data augmentation or normalization techniques like dropout [89] in the SpikeCNN implementation.

The network parameters like input rates (I_{rate}) and threshold values (v_{th}) for the networks were set by trial and error by cross-validating a few times to get the lowest reconstruction error from the input image layer. Fig. 6.3 shows the reconstructed image patterns formed at the pseudo-visible layer after training the first convolution layer of MNIST_2C with regenerative learning for different values of v_{th} and I_{rate}. Visually, we can inspect that Fig. 6.3(b) with parameters v_{th}= 1.2, I_{rate} =100 Hz ($P2$) and Fig. 6.3(c) with v_{th}= 0.8, I_{rate} =75 Hz ($P1$) give more convincing reconstruction than Fig. 6.3 (a) v_{th}= 1.0, I_{rate} =100 Hz. We use the parameters $P2, P1$ corresponding to Fig. 6.3 (b), (c) to initialize MNIST_2C and evaluate the classification accuracy in both cases. Similarly, Fig. 6.5 shows the reconstructed image patterns from the first convolutional layer of CIFAR_3C for the three color channels separately. The parameters used are v_{th}= 1.2, I_{rate} =100Hz to initialize CIFAR_3C and obtain the classification accuracy. The figures show the accumulated spike count over 250ms of simulated time. Please note that though the deeper layers in MNIST_2C/CIFAR_3C have significantly larger number of maps than that of initial layers in both the configurations, the training time still remains same due to the down-sampling of input size with average pooling.

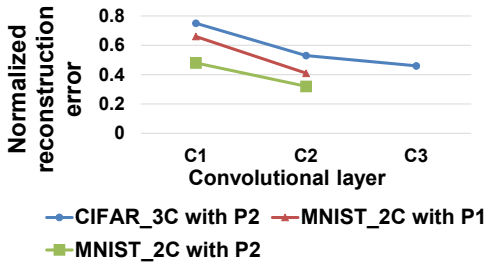


Fig. 4.6. Reconstruction errors at different layers of the SpikeCNN architecture

4.4.2 Reconstruction error across network layers

For quantitative evaluation, we plot the reconstruction errors obtained at each layer of the network (MNIST_2C, CIFAR_3C) as shown in Fig. 6.5 for the network parameters discussed above. In order to ensure propagation of spike information across layers, we present the input data 3/5 times while training every convolutional layer of MNIST_2C/CIFAR_3C respectively. The reconstruction error plotted in Fig. 6.5 is the aggregate loss over the multiple presentation of the training data. We use the squared Euclidean distance of the difference in the spike events as a measure of loss. It is clearly seen that error observed for MNIST_2C with parameters $P1$ is higher than that of $P2$ which supports the visual reconstruction patterns shown in Fig 6.3(b), (c). A noteworthy observation here is that the reconstruction error in both networks decreases as we move towards deeper layers. In [90], the authors have shown that in the context of deep ANNs, blurring an image enables better reconstruction as the network then learns more general representations. Learning the finer details may lead to overfitting increasing the reconstruction error. As we apply convolution and average pooling in the initial stage, certain information from the original image is lost in this process. Thus, we can interpret that the deeper layers work on slightly blurred details of the original image causing lower reconstruction error.

4.4.3 Classification Accuracy

After training the convolutional layers by optimizing the reconstruction error for the entire training dataset, the features from the final layer are then fed to the output layer. The weights at the output are trained in a supervised manner by showing training labels. However, in the supervised case, we use only a subset of the training data to train the final layer. During testing, an input test pattern is presented two times to the trained SpikeCNN. The spikes at the output neurons are aggregated over two simulations and the neuron with the highest response is the predicted class for the given test input. Since the pixel intensity values are converted to Poisson spike trains, the accuracy can differ for different spike timings. Thus, the accuracies are averaged over five iterations of presentation of the entire testing dataset. Fig. 6.6 shows the classification error for MNIST_2C initialized with parameters $P1, P2$ and CIFAR_3C with $P2$ as the size of the labelled training subset is varied. The results for CIFAR_3C initialized with $P2$ have only been shown as it yields a lower reconstruction and classification error in comparison to $P1$. For MNIST_2C with $P2$, the lowest error achieved by showing all the 60000 training labels at the final layer is 0.92% (99.08% classification accuracy). It is clearly seen from Fig. 6.6 (a) that the error decreases significantly as the size of the training set is increased from 500 to 20000. However, for subsets $\geq$ 20000, the error almost remains the same. Similarly in Fig. 6.6 (b), for CIFAR, the error doesn't change significantly for subsets $\geq$ 30000. For MNIST_2C/ CIFAR_3C with $P2$, the minimum error obtained showing 20000/30000 labels at the final layer is 0.95%/24.58%. In [91]/ [92], the authors have implemented a spiking deep network by converting a deep static CNN to SNN and have achieved 22.57%/0.86% error on CIFAR-10/MNIST dataset. The fact that our network performs favorably incorporating the inherent latencies of a spiking neuron model in the learning process suggests that the regenerative learning scheme can be used to train deep spiking networks to obtain state-of-the-art results.

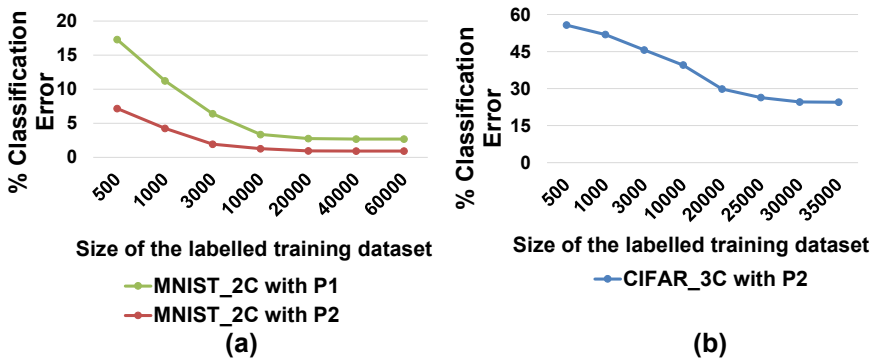


Fig. 4.7. Classification error as the size of the labelled dataset for the supervised training of output layer is varied

4.4.4 Sparsity with Regenerative Learning

The spike-based regenerative learning scheme, on account of event-based coding, introduces sparsity over the convolutional layer feature representations. Since the learning is based on spiking activity of the neurons, the output at the pseudo-visible layer is reconstructed using only the maximally active neurons in the feature maps of the convolutional layers. As a result, the reconstruction error at each time instant of the training epoch is back-propagated through these active neurons. Basically, the sparse event based computation acts as a regularizer that prevents learning of over-complete representations of the input. In other words, the sparsity in features decreases the number of filters or weight kernels required to reconstruct the input thereby forcing the filters to be more general. Fig. 5.8 shows the feature maps learnt for MNIST_2C (with $P2$) with accumulated spike counts over 250 ms for a particular input pattern. It is evident that the feature maps in both convolutional layers have sparse active units. Only a smaller section of the SpikeCNN is active in a training epoch. Thus, we can effectively save power on the remaining idle or inactive portions of the network. Sparsity in learning has a key role in reducing the overall power consumption by decreasing the spike rate in SNN architectures which is one of the main reasons to use SNN over ANN. Fig. 5.8 also shows the weight kernels learnt at

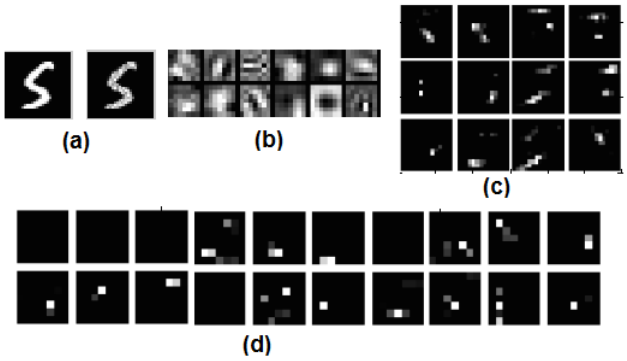


Fig. 4.8. (a) MNIST training input to MNIST_2C initialized with $P2$: Original pixel image (left), Spike input image (right) (b) The weight kernels learnt at the input layer (c) 12 feature maps showing the sparse representations of maximally active spiking neurons in the first convolutional layer of MNIST_2C (d) 20 of 64 feature maps in the second convolutional layer of MNIST_2C with sparsely active neurons

the first layer from the input. Visually, we can interpret that the weights are more diverse and global.

4.5 Conclusion

We introduced a spike-based learning scheme to train Spiking Deep Networks (SpikeCNN) for object recognition problems using leaky integrate-and-fire (LIF) neurons. The regenerative model learns the hierarchical feature maps layer-by-layer in a deep convolutional network in an unsupervised manner. Once the convolutional layers are learnt, the features are then fed to an output layer trained in a supervised manner by showing a fraction of the labeled training dataset. The output layer performs the overall classification of the input. While previous work on deep spiking networks have examined the conversion of ANNs to SNNs, we build a spiking deep CNN from scratch with the proposed learning using spike-timing information and inherent latencies to implement layer-wise weight modification. Also, the sparsity in representations introduced with regenerative learning suggests overall power savings in the learning process which generally takes up a significant part of the time the

network is used. Finally, the SpikeCNN system developed with our proposed learning with the current initialized parameters is in its first generation, and we expect its accuracy on CIFAR-10 to improve as we gain experience with the method and tune the network for better parameters.

5. ASP: LEARNING TO FORGET WITH ADAPTIVE SYNAPTIC PLASTICITY IN SPIKING NEURAL NETWORKS

5.1 Introduction

With the proliferation of intelligent devices, including smartphones and the Internet of Things, and the resultant explosion of digital data that they generate, computing platforms across the spectrum will increasingly need to acquire, process and analyze new data. This requires such resource-constrained devices to continually extract structures, patterns, and meaning from raw and unstructured datasets in real-time, unsupervised dynamically changing environments.

While advances in deep learning and other machine learning techniques have led to computers matching or surpassing human performance in several tasks (recognition, analytics, inference) [1, 4, 40], the inherent learning mechanisms for such tasks are static. That is, the learning methods use data points from past or old experience to build a predictor (classifier, regression model, recurrent time series model) for processing future behavior. The predictor does not adjust itself (self-correct or adapt) as new events happen. However, accurate prediction based on the new incoming data characteristics with limited memory bandwidth and computational resources needs to be ensured to enable on-chip intelligence. This requires continuous learning over a long period of time with the ability to evolve structural components on demand and forget data becoming obsolete. Based on this, we introduce a novel adaptive self-learning scheme: ASP (Adaptive Synaptic Plasticity) that forgets (or weakens) already learnt connections to make room for new information to adapt to the continuously changing inputs.

Forgetting is essential for learning new things. Recent work in [93] suggests that the brain is actively erasing memories while learning to continuously process the new input stimuli. At a preliminary level, learning in the brain involves making synaptic connections and re-enforcing them with repeated exposure to a given input stimuli. However, due to limited space available, the brain forgets already learnt connections, gradually, to associate them with new data. In contrast, all computing systems learnt using static mechanisms suffer from "*catastrophic forgetting*" where the exposure of an already trained system to new information results in severe loss of previously learned information. It has been shown that this extreme loss of information is not due to the constrained memory capacity (amount of resources available for learning) of the learning system, but is caused by the overlap of representations of information within the system [94–96]. To address this, our learning scheme maintains a balance between continuous learning and forgetting that is necessary to deal with dynamic environments.

For many years, Artificial Neural Networks (ANNs, including deep learning networks) have dominated across all machine learning models due to their unprecedented performance (in terms of classification/recognition accuracy) in a wide range of computer vision and related applications. However, for applications that require real-time interaction with the environment, the repeated and redundant update of large number of units (neural and weight connections) becomes a bottleneck for efficiency. An alternative has been proposed in the form of Spiking Neural Networks (SNNs), an emergent class of computing paradigm in theoretical neuroscience and neuromorphic engineering [72, 97]. Driven by brain-like asynchronous event based computations, SNNs focus their computational effort on currently active parts of the network, effectively saving power on remaining idle parts, thereby achieving orders of magnitude lesser power consumption in comparison to their ANN counterparts [70, 71]. In addition, SNNs are equipped with self-learning capabilities like Spike Timing Dependent Plasticity (STDP) [79] that further make them desirable for adaptive on-chip implementations.

STDP is an associative form of synaptic plasticity that modulates the synaptic strength of a connection based on the temporal correlation between the spiking patterns of pre and post neuron pairs. Several neuroscience theories have demonstrated the role of STDP for Long Term Potentiation (LTP) of synapses that underlie long-lasting memory formation and learning in the brain [98]. However, an SNN learnt with STDP alone does experience catastrophic forgetting while trying to learn quickly in response to a changing environment. Memory persistence is a particularly prominent problem with STDP, as in its naive form STDP implies that any pre/post spike pair can modify the synapse, potentially erasing past memories abruptly. Grossberg et. al [99, 100] have called the problem where the brain learns quickly and stably without catastrophic forgetting its past knowledge while gradually adapting to the flood of new data, the *stability-plasticity dilemma*.

The ASP learning discussed here addresses the catastrophic forgetting and stability-plasticity dilemma by incorporating a gradual "decay mechanism" with the temporal STDP based weight update procedure. In fact, several works on hippocampal learning have demonstrated memory trace decaying over a period of time [93, 101–103]. Specifically, LTP that is involved in maintaining memory has been shown to be not permanent and such potentiated responses eventually decay to certain baseline levels [102, 103]. While the underlying mechanisms for the LTP decay are not known, certain neuroscientists hypothesize that the decay is correlated with "forgetting" [101–103]. Inspired from this biological evidence, we integrate a weight leaking mechanism with the standard STDP learning rule to model adaptivity in SNNs for pattern recognition applications. ASP facilitates the gradual degradation or forgetting of already learnt weights to realize new and recent information while preserving some memory or knowledge about the old data that are significant. To the best of our knowledge, ASP is the first of its kind biologically plausible learning paradigm that effectively combines "forgetting with learning" and tremendously boosts the capability of traditional SNNs to deal with dynamic environments.

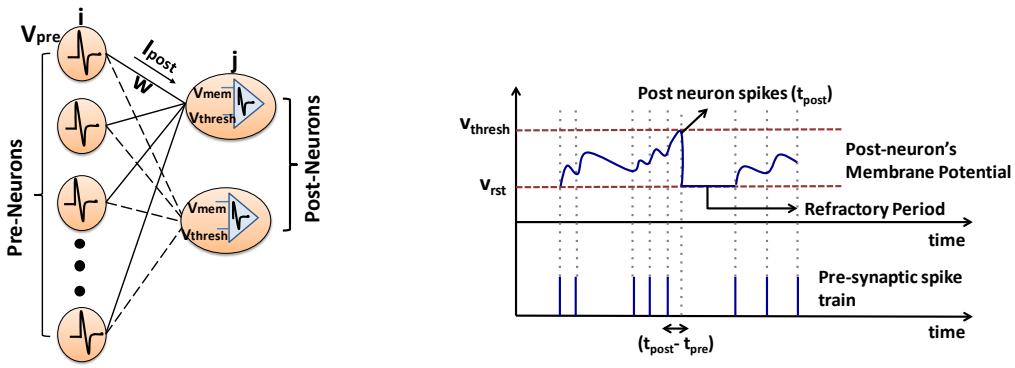
Essentially, in an SNN that is learnt using STDP for pattern recognition, the weight states are static or are not altered until a post/pre neuron spike is observed. This static learning of weights results in overlap of representations leading to catastrophic forgetting when presented with frequently changing inputs. The key idea of ASP is to leak the weights at every time instant (towards a baseline value) irrespective of post/pre-neuron spikes. The leak rate for each synaptic connection (or leak time constant) is modulated based on the degree of potentiation or depression of a synapse obtained from STDP. While STDP helps in learning the input patterns, the retention of old data/gradual forgetting is attained with the weight decay. Thus, ASP maintains a balance between forgetting and immediate learning to construct a *stable-plastic* self-adaptive SNN for dynamic environments.

In addition to dynamic learning, the "learning to forget" mechanism achieved with ASP further enables an SNN to perform unsupervised selective attention and denoising by focusing on the task relevant information in the data. In fact, our results (in section 5.4) for handwritten digit recognition from noisy inputs show that the ASP trained SNNs extract distinct digit representations while filtering out the irrelevant background noise. As we will see later, the adaptive weight decay mechanism helps in eliminating the noisy elements in an image as the weights corresponding to those pixels are gradually forgotten. This result emphasizes the significance of the adaptive weight decay for an improved and robust unsupervised learning procedure.

5.2 Spiking Neural Network: Fundamentals

5.2.1 Spiking neuron and synapse model

The SNN topology used for pattern recognition consists of a couple of layers of spiking neurons interconnected by synapses as standard fully connected ANNs shown in Fig. 8.1 (a). However, unlike conventional ANNs where a vector is given at the input layer once and the corresponding output is produced after processing through several layers of the network, SNNs require the input to be encoded as a stream of



(a) (b)

Fig. 5.1. (a) A typical SNN architecture consisting of pre-neurons and post-neurons interconnected by synapses. The pre-synaptic voltage spike V_{pre} is modulated by the synaptic weight, w, to get the resultant post-synaptic current, I_{post}. The post-neuron integrates the current from each interconnected pre-neuron that causes its membrane potential, V_{mem}, to increase and spikes when the potential crosses a certain threshold, V_{thresh}. (b) The Leaky-Integrate-and-Fire dynamics of the membrane potential of a post-neuron that increases upon the arrival of pre-synaptic spike and decays subsequently. The post-neuron fires when the potential exceeds the threshold V_{thresh}. the potential is then reset to V_{rst} and a refractory period ensues during which the neuron is prohibited from firing. The relative timing of the post-neuron and pre-neuron spikes $(t_{post} - t_{pre})$ determines the synaptic potentiation.

spike events. The neurons transmitting the spikes are referred to as pre-neurons. Each pre-neuronal spike is modulated by the synaptic conductance to produce a resultant post-synaptic current that is received by the post-neurons.

$$\tau_{post}\frac{dI_{post_{j,i}}}{dt} = -I_{post_{j,i}} + w_{j,i}E_{pre_i} \tag{5.1}$$

where $I_{post_{j,i}}$ is the current received by the jth post-neuron due to a spike event E_{pre_i} at the i^{th} pre-neuron, that are interconnected by a synapse of strength $w_{j,i}$. The post-synaptic current increases by the synaptic weight $w_{j,i}$ instantaneously when a pre-neuronal spike occurs at t_{pre}, and subsequently decays with time constant τ_{post}. The spiking neuron model used is the Leaky-Integrate-and-Fire (LIF) model as illustrated in Fig. 8.1 (b). A simplistic LIF model can be described as follows:

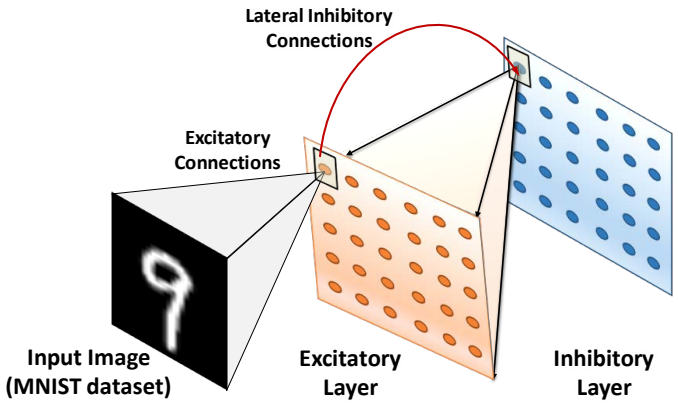


Fig. 5.2. SNN topology for pattern recognition consisting of input , excitatory and inhibitory layers arranged in a hierarchical fashion. The input layer is fully connected to the excitatory neurons, that are connected to the corresponding inhibitory neurons in a one-to-one manner. Each of these neurons inhibits the excitatory layer neurons except the one from which it receives the one-to-one connection.

$$\tau_{mem}\frac{dV_{mem_j}}{dt} = -V_{mem_j} + \sum_i I_{post_{j,i}} \tag{5.2}$$

where V_{mem_j} is the membrane potential of the j^{th} post-neuron. The post-neuron sums the total post-synaptic current leading to an increase in its membrane potential that eventually leaks exponentially with time constant τ_{mem}. It fires or emits a spike when its membrane potential reaches a specific threshold, similar to biological neurons, thus adding an asynchronous component of computation to SNNs. The time instant of occurrence of post-neuron spike is denoted by t_{post}. After each post firing event, the neuron's membrane potential is reset and a refractory period ensues during which the neuron is incapable of firing even if additional input spikes are received.

5.2.2 SNN topology for Pattern Recognition

The experimental tests are digit recognition tasks conducted with the MNIST dataset [48]. A hierarchical SNN structure as shown in Fig. 8.2 is used. The architecture consists of an input layer followed by excitatory and inhibitory layers.

The input layer (28x28 dimension) constitutes the pixel image data for the various patterns in the dataset. Each pixel value is converted to Poisson-distributed spike trains with rates proportional to corresponding pixel intensities. The input layer is fully connected to the neurons in the excitatory layer. Thus, each excitatory neuron has 784 weighted synaptic connections from the input, that are trained (or learnt) to classify a specific input pattern or digit. The excitatory layer is connected in a one-to-one fashion to the inhibitory layer neurons i.e. a spike event in an excitatory neuron would cause the corresponding inhibitory neuron to fire. Every inhibitory neuron is connected back to all excitatory neurons, except the one from which it receives a forward connection. This connectivity structure provides lateral inhibition that limits the simultaneous firing of various excitatory neurons in an unsupervised learning environment and promotes competitive learning.

Besides lateral inhibition, we employ an adaptive membrane threshold mechanism called homeostasis [104] that regulates the firing threshold to prevent a neuron from being hyperactive. Specifically, each excitatory neurons membrane threshold is not only determined by v_{thresh} but by $v_{thresh}+\theta$, where θ is increased each time the neuron fires and then decays exponentially [105]. If the neuron is too active in a short time window, the threshold grows gradually and in turn, the neuron requires more input to spike in the near future. Homeostasis, thus, equalizes the firing rate of all neurons preventing single neurons from dominating the response. Note, the SNN topology shown in Fig. 8.2 is used to conduct all recognition experiments detailed in the later sections of the chapter.

Next, we discuss the standard STDP model for unsupervised on-line learning in SNNs and then, describe our decay based Adaptive Synaptic Plasticity learning.

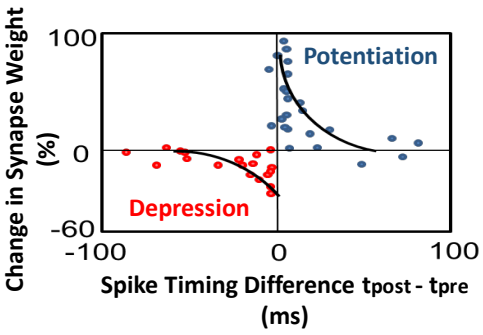


Fig. 5.3. Spike timing dependent plasticity curve showing the original measurement from biology in rat's hippocampus [106] and the traditional model. The relative change in synaptic weight is exponentially related to the difference in spike times of a post-neuron (t_{post}) and pre-neuron (t_{pre}).

5.3 Existing Learning Model for SNN

5.3.1 Spike Timing dependent Plasticity (STDP) and its limitations

In the SNN topology described above, the synapses connecting the input pre-neurons to the post neurons in the excitatory layer need to be trained to learn a generic representation of a class of input patterns. STDP is a set of Hebbian learning rules with mathematical formulation based on the biological evidence observed in the rat hippocampus [106] as follows:

$$\begin{aligned} \Delta w &\propto exp(\Delta t/\tau^{-}) \quad if \Delta t < 0 \\ &\propto exp(-\Delta t/\tau^{+}) \quad if \Delta t \geq 0 \end{aligned} \tag{5.3}$$

where $\Delta t = (t_{post} - t_{pre})$ is the time delay between the post and presynaptic spike, $\tau^{+/-}$ is the time constant for potentiation/depression. There are several variations of STDP learning that have been demonstrated to be suitable for learning spatio-temporal patterns. Fig. 6.2 shows the traditional exponential STDP model with the potentiation and depression window for weight learning [79, 107, 108] and the original measurement from biology [106]. Synaptic plasticity (or weight change) depends on the interval of time elapsed between pre- and post-synaptic spikes as indicated in

Fig. 8.1(b). The weight is increased/potentiated if a post-neuron fires subsequently after a pre-neuron, with shorter time intervals resulting in an exponentially larger change. On the other hand, it is decreased/depressed if a post-neuron fires before a pre-neuron indicating the absence of a causal relationship.

When multiple neurons are organized in a simple competitive SNN topology, the STDP learning will enable the network to learn multiple distinct patterns. However, in case of on-line learning in a non-stationary environment, a fixed-size network (fixed in terms of number of neurons in excitatory layer and corresponding synaptic connections) must continually alter its response to process new data. Here, competition would not preclude already learnt neurons from responding to a new pattern and adjusting synaptic weights accordingly. This will often result in overlap of representations unless the network size is increased. But circuit/hardware and power limitations pose resource limitations on network sizes where storage cannot be increased. So, constant exposure to new information would result in cumulative synaptic changes leading to a dramatic loss of stored information (or Catastrophic Forgetting) as the previous changes in synaptic weight will be effectively lost. This effect does not occur in the brain since the information loss is gradual. To retain previously learnt information and avoid overlap or overwriting, a common solution proposed is to provide data reinforcement both during initial training and any subsequent training [109] as discussed next.

5.3.2 Mitigating Catastrophic forgetting in STDP learnt models with data reinforcement

Data reinforcement is presenting old information to the network along with the new or current information to ensure that previously learnt representation is stably retained while new information is learnt. This helps in resolving the stability-plasticity dilemma of SNNs in a dynamic input environment. Let us discuss two scenarios for digit recognition: a) In Scenario ‘A’, the digits ‘0’ through ‘9’ are presented one by

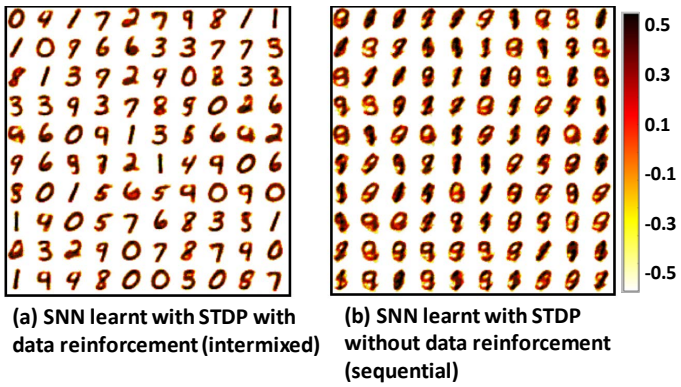


Fig. 5.4. Digit representations learnt with STDP in an SNN with 100 excitatory neurons connected to input layer (28x28 pixels) when (a) the digits '0' through '9' are presented in an intermixed manner (i.e. data reinforcement) (b) the digits '0' through '9' are presented sequentially without re-presenting any previous category inputs (i.e. no data reinforcement in dynamic environment). Catastrophic forgetting is observed in SNNs learnt with STDP in a dynamic environment (b) due to significant overlap of representations.

one sequentially i.e. first all the images for digit '0' followed by all images for digit '1', and so on till digit '9' and thus can be treated as a dynamic learning environment. Here, periodically after a given number of iterations, the SNN is presented with a new class or digit. b) In Scenario 'B', the digits are all presented randomly in an intermixed fashion and the intermixed selection of classes (or digits) are repeated iteratively. Presenting all the digits in an intermixed manner iteratively is basically performing data reinforcement wherein old information is re-presented with new data so that later or new input data do not replace previous patterns. Effectively, from a traditional neural network perspective, in Scenario 'B', we are retraining the network with both the new and the old information when the network has to learn a new class. In contrast, Scenario 'A' is an example of learning without data reinforcement as all classes are presented disjointedly without any relevance to other classes.

Fig. 6.3 compares an SNN learnt with traditional exponential STDP in the above two scenarios with/without data reinforcement. The SNN topology consists of 100 neurons in the excitatory layer with fully connected synaptic weights from the input

(28x28 pixels for MNIST). During learning, the excitatory synaptic weights from the input layer to each excitatory neuron are modulated to learn a particular input digit using the learning rule. Towards the end of the training phase, the weights (or excitatory connections) that are randomly initialized eventually learn to encode a generic representation of the digit patterns. Specifically, the excitatory connection weights (refer to Fig. 8.2) fanning out of the higher-intesity (or white) pixel regions will get potentiated while the weights from the low-intensity regions on the input image will be depressed during the learning phase. Correspondingly, the color-map figures shown in Fig. 6.3 represent the weight values learnt corresponding to each excitatory neuron when learning stops.

Fig. 6.3 (a) clearly shows how the SNN presented with a random mixture of input classes is able to learn a balanced representation of all input patterns due to data reinforcement. Fig. 6.3 (b) shows the SNN learnt in Scenario A where old information or data categories were not re-presented to the network, causing an overlap of input representations. The SNN always tried to learn the new digit representation while trying to retain a portion of the old data. However, fixed network size and absence of data reinforcement resulted in accumulation causing new weight updates to coalesce with already learnt patterns. Hence, an SNN will be rendered useless for categorizing the digits in a dynamic environment without data reinforcement. However, in on-line real time learning, it is often impractical and even expensive to store all old data samples for retraining or data reinforced learning each time a new input pattern or class is encountered. ASP, wherein we perform STDP updates augmented with dynamic weight leaking, would enable an SNN (with fixed resources) to gracefully forget already learnt information while retaining the significant patterns and gradually adapt to new data in an online learning environment.

Next, we discuss the principles of weight decay based learning and how it can be applied for pattern recognition with SNNs.

5.4 Adaptive Synaptic Plasticity for Learning to Forget

5.4.1 Adaptivity with Weight Decay in SNNs

Before we discuss the ASP learning rule, let's see how the weight decay alone helps in adapting a network to continuously changing inputs. Here, we do not incorporate any timing-based associative rules such as STDP with the weight decay process. We only perform an exponential decay and recovery of synaptic weights based on the incoming spikes from the pre-neurons as

$$\tau_{leak}\frac{dw}{dt} = -\alpha w + P(t) \tag{5.4}$$

where τ_{leak} (*200 ms*) is the time constant for the exponential decay of weights, α (*0.01*) determines the overall learning rate. Here, $P(t)$ denotes the presynaptic trace that models the recent presynaptic history. To improve simulation speed, the weight dynamics are computed using synaptic traces [80, 110]. This means that, besides the synaptic weight, each synapse keeps track of the presynaptic trace as well. Each time a pre-neuron fires, the presynaptic trace is increased by 1, otherwise $P(t)$ decays exponentially as

$$P(t) = exp(-t/\tau_{trace}) + t_{pre} \tag{5.5}$$

where τ_{trace} (*20 ms*) is the time constant for the decay of pre-synaptic trace and t_{pre} indicates the presence/absence of pre-neuronal spike. The t_{pre} value is 0 if pre-neuronal spike is absent, while 1 when pre-neuronal spike occurs at any time instant t. As per Eqn. 5.4, during a given epoch or time period when an input pattern is presented, the synaptic weights are potentiated proportional to the presynaptic trace value calculated at the particular time instant when a t_{pre} occurs (or pre-neuron fires). At all other time instants, the weights are leaking or decaying. Please note that the above learning is unaccompanied by any sort of correlation between post and pre-neuron spike timings. It is solely driven by pre-neuronal spikes.

Fig. 8.7 shows the digit representations learnt in an SNN's excitatory layer using the isolated decay based weight modulation (Eqn. 5.4) when different digits (Digit: 5,

Fig. 5.5. Digit representations learnt with isolated weight decay learning in an SNN with 9 excitatory neurons connected to input layer (28x28 pixels). The input digits are presented sequentially (i.e. dynamic environment with no data-reinforcement). The network is overly plastic and adaptive to the continually changing inputs.

0, 4, 1; order chosen randomly) are presented to the network sequentially. The SNN comprises of 9 neurons in the excitatory layer. The color intensity of the patterns are representative of the value of synaptic weights. As different digits are presented, the weights adapt to the new input pattern while forgetting the old information. Thus, we see that the current input representation is learnt on top of the older and most recent pattern. While the weights are highly potentiated for the current input pattern (almost black-to-red intensity), we can observe from the color intensity (orange-to-yellow) that the old information is being forgotten (or corresponding synaptic connections leak due to absence of spike inputs from the pre-neurons in that region as per Eqn. 5.4).

A key observation here is that while learning a new pattern, the overlap occurs with the most recent input. For instance, while learning a digit '4', the SNN still retains some information about the previous input digit '0'. However, it has completely forgotten all information regarding the older digit '5'. Thus, weight decay enables the SNN to gracefully forget old information while adapting the network to learn new data. This is what helps the SNN to avoid catastrophic forgetting that was observed with standard STDP learning without data reinforcement (Fig. 6.3 (b)), wherein the weight updates got accumulated resulting in significant overlap of representations. Thus, it can be inferred that forgetting or weight decay is integral to self-learning and adapting in a dynamic environment. However, in the above learning rule, relative

timing of pre- and post-synaptic spikes has not been considered for modifying the weights. The weight updates occur based only on pre-neuronal spike that is in contrast to the commandments of Hebbian learning [111]. Hence, there is no possibility of initiating a competition between neurons to learn different patterns. As a result of this, all the excitatory neurons in Fig. 8.7 learn uniformly and pick up the same pattern. The SNN in this case is excessively plastic and inadequately stable as it immediately adapts to the new information. Thus, it is essential to incorporate spike timing based correlation with the weight decay to learn in an unsupervised dynamic environment in a *stable-plastic* way.

Another way of invoking competition is by forcing certain neurons selectively to learn different patterns. However, in that case, we move into the supervised learning domain that is generally undesirable for on-chip implementations as motivated earlier.

5.4.2 Adaptive Synaptic Plasticity: Combining STDP with Weight Decay

In the weight decay learning discussed above, the role of post synaptic neuron's spiking activity was not accounted for modulating the weights. Adaptive Synaptic Plasticity (ASP) blends in associative learning with the adaptation behavior that helps in retention and gradual adaptation to new inputs as well as evokes competition across neurons to learn distinct patterns. Essentially, we modulate the leak time constant (or forgetting behavior) and the exponential recovery of synaptic weights, observed earlier with isolated decay-based learning, using the temporal dynamics of pre- and post-synaptic neurons. In STDP (refer to Fig. 6.2), the spike timing correlation (or difference of the spike events) between the post/pre neurons determines the window (positive/negative) across which the synaptic modification (potentiation/depression) takes place. Likewise, ASP also incorporates different windows for potentiation and depression based on the firing events of the post/pre neurons as shown in Fig. 6.5.

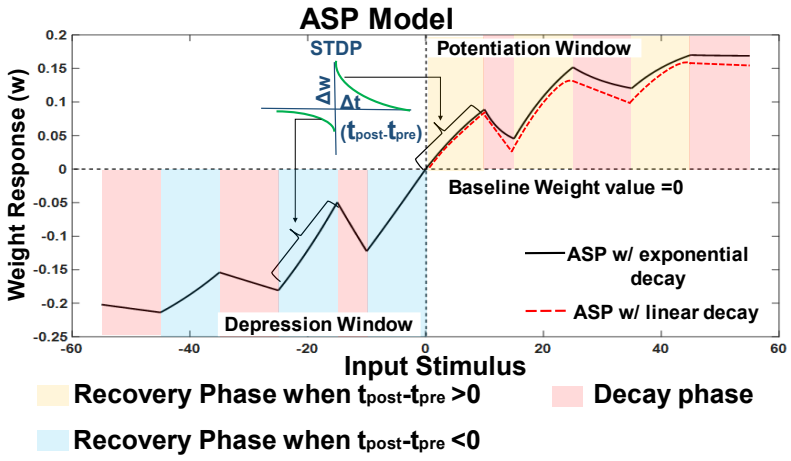


Fig. 5.6. ASP model for weight modulation. During the recovery phase, i.e. when a spiking event occurs at the post/pre-neuron, the weights are potentiated (or depressed) based on the t_{pre} and t_{post} realtive timing difference following the STDP dynamics (Eqn. 5.7). The weights leak towards baseline value (=0) during the decay phase and the leak time constant is varied based on the post-synaptic neuron's spiking activity (Eqn. 5.8). Two different ASP models with exponential (black curve) and linear (dotted red curve) decay are shown.

ASP interprets any spiking activity at the post or pre-neuron as the presence of input stimulus to initiate, what we term as, recovery phase for the synaptic weight interconnecting them. The recovery can be an exponential increase (potentiation) or decrease (depression) based on the temporal difference between the spiking activities of the neurons. The potentiation or depression of the weights in the recovery phase is obtained from the exponential STDP formulation (Refer to Eqn. 5.3) as shown in Fig. 6.5. Thus, if a pre-neuronal spike forces a post-neuron to fire (i.e. $t_{pre} < t_{post}$), the corresponding synapse should potentiate based on Eqn. 5.3 since its positively correlated with the input pattern. On the contrary, if a pre-neuron fires after a post-neuronal spike (i.e. $t_{pre} > t_{post}$), the specific synapse ought to depress.

Following the recovery (STDP based weight modulation), the decay or forgetting phase in a synaptic weight ensues, as illustrated in Fig. 6. During the decay phase, the weights leak in both potentiation/depression window towards a baseline value with varying leak time constants. Please note that in ASP, the decay phase (or weight

leaking) continues even when there is no spiking activity (or input stimulus) observed at both post/pre neuron. For instance, during the refractory period, when a post neuron does not spike, then in the absence of a pre-neuronal spike the weights will continue to leak. This is in stark contrast to the traditional STDP learning wherein the weight states are only altered in the presence of a pre- or post-neuronal spiking activity. Thus, it can be inferred that the fundamental weight response in ASP is to conduct STDP based weight modifications with intermediary weight decay (or leak) at all time instants during the training period.

To emphasize the efficacy of ASP for adaptive learning, we validate our experiments with two different kinds of decay: exponential and linear as illustrated by the black and dotted red model, respectively, in Fig. 6.5. The formulations for each of these decay are explained later. In Fig. 6.5, the baseline value of weights toward which the weights decay is 0. This implies that the weights are dynamic or continuously changing at every time instant during the training phase when input patterns are presented. Please note, for the sake of convenience in representation and for clarity, the characteristics of ASP with linear decay has not been shown for the depression window in Fig. 6.5. However, it will follow similar behavior as the other ASP model with weights decaying linearly toward the baseline value.

Fig. 6.6 shows the synaptic weight modulation using ASP for a given synaptic weight based on the spiking patterns of the interconnecting neurons. As in Fig. 6.5, the baseline value toward which the synapses leak is assumed to be 0 here. Please note that while depression of weights in recovery phase is a result of negative correlation between post and pre-neuron, the leak dynamics in decay phase (only shown for exponential leak) encode the adaptive behavior of the network. The leak dynamics determine which synaptic weights connected to a post neuron (that has learnt old or insignificant data) should be forgotten to learn the new data. On the other hand, the synaptic weight updates performed during the recovery phase potentiation/depression enable an SNN to learn a generic representation of a class of input patterns competetively. For instance, the weights of an excitatory layer post-neuron

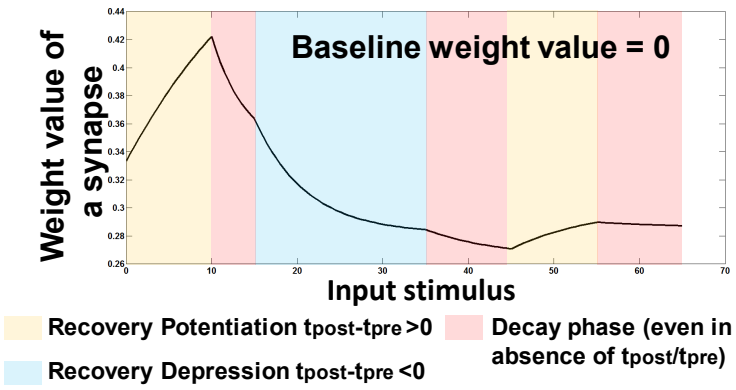


Fig. 5.7. Weight response behavior of a particular synapse during intermediate recovery (potentiation or depression) and decay phases.

learning a digit '2' should spike for different instances of '2' so that it learns a more generic representation rather than just mimicking a specific instance. Thus, synaptic depression (based on STDP) and leak (based on weight decay) have different roles in ASP learning.

Note, Fig. 6.5, 6.6 are animations that show the weight modification with ASP based on presence/absence of spiking activity. For the sake of clarity, we demarcated the ASP rule into two phases since the weight update rule in the presence of a pre/post spike is different than when spikes are absent. Input stimulus on the X-axis basically indicates the presence (or absence) of pre/post synaptic spike that initiates a recovery (or decay) phase. It quantifies the timing correlation ($t_{post} - t_{pre}$) in a post/pre neuronal pair.

5.4.3 ASP: Learning Rules

Now, we will discuss the mathematical formulations for ASP and understand the temporal dynamics related insights that eventually enables the SNN to learn to forget irrelevant old information while retaining significant data and adapting to new patterns.

Recovery Phase

As described earlier, the weight dynamics are calculated using synaptic traces. In ASP learning, we maintain three different kinds of traces, for each synaptic weight, to track the spiking activities of the corresponding pre- and post-synaptic neurons:

- Recent presynaptic trace (Pre_{rec}) that does not accumulate over time (only accounts for the most recent spike)
- Accumulative presynaptic (Pre_{acc}) trace that adds over time (accounts for the entire spike history of the presynaptic neuron for a given time period or epoch during which a particular pattern is presented to the SNN)
- Postsynaptic trace ($Post$) that accumulates over time based on the postsynaptic neurons spiking activity.

The above trace values are computed as shown in Eqn. 5.6. When any spiking activity is observed the trace value is increased, otherwise it decays exponentially.

$$\begin{aligned} Pre_{rec}(t) &= exp(\frac{-Pre_{rec}}{\tau_{rec}});\ Pre_{rec} = 1\ at\ t_{pre} \\ Pre_{acc}(t) &= exp(\frac{-Pre_{acc}}{\tau_{acc}});\ Pre_{acc} += 1\ at\ t_{pre} \\ Post(t) &= exp(\frac{-Post}{\tau_{post}});\ Post += 1\ at\ t_{post} \end{aligned} \tag{5.6}$$

Now, it is evident that in order to account for appropriate accumulative spike history, the time constant for decay of the accumulative pre-trace (Pre_{acc}) has to be larger than that of the recent pre-trace (Pre_{rec}). In our simulations, $\tau_{acc} = 10\tau_{rec}$, $\tau_{post} = 2\tau_{acc}$. Now, the weight update during the recovery phase follows the exponential STDP behavior discussed earlier. We use a modified version of the STDP model used in [80] to obtain the weight changes in presence of input stimulus in ASP. When a post-synaptic neuron fires a spike, the weight change Δw is calculated based on the presynaptic traces (Pre_{rec}, Pre_{acc})

$$\Delta w = \eta(t)[(Pre_{rec} - offset) - \frac{k_{const}}{2^{Pre_{acc}}}]$$
$$\eta(t) = \frac{k_{1,const}}{(Post(t) + 1)} \tag{5.7}$$

where $\eta(t)$ is a time dependent learning rate inversely proportional to the post-synaptic trace value ($Post(t)$ from Eqn. 5.6) at a given time instant. The learning rate decreases as the spiking activity of the post-synaptic neuron increases for a given input. This ensures stable and retentive learning of an input pattern by a particular neuron. It also prevents the neuron from quickly adapting to a new pattern (or catastrophic forgetting). The $offset$ ensures that the weights interconnecting those pre-synaptic neurons that rarely lead to firing of a post-synaptic neuron will depress. For instance, in case of digit inputs, the lower intensity pixel regions for a particular digit will become more and more disconnected resulting in lowering of synaptic weight values corresponding to those pre-neurons. In Eqn. 5.7, the first part represents the potentiation or depression of weights based on the most recent pre-synaptic spike (as the simple STDP model in [80]). However, as seen earlier, such simplistic weight modification leads to erasure of memory traces. Besides precise spike timings that identify the temporal correlation between input patterns, learning rule should incorporate the significance of the inputs to modulate the weights. Input based significance driven learning would enable the SNN to learn in a stable-plastic manner in a dynamic environment.

The second part of Eqn. 5.7 ($k_{const}/2^{Pre_{acc}}$) quantifies the dependence of the weight change on input significance. We define input significance as directly proportional to the number of training patterns (of a particular input class) shown at the input layer of an SNN. Consequently, for larger number of similar patterns, the corresponding input neurons will have more frequent spikes. In that case, Pre_{acc} value will be high that would eventually make the second term in Eqn. 5.7 less dominant for determining the final weight update. Thus, for more frequent input spikes at the pre-neuron, the

weight update will be more prominent. Hence, the learning rule in the recovery phase encompasses significance of the inputs with synaptic plasticity.

Besides significance driven learning, the second part of Eqn. 5.7 also enables an SNN to learn more generalized input representations. Fig. 5.8 shows two scenarios wherein the weight updates vary based on the frequency of input spikes. It is seen that the final weight value towards the end of the recovery phase is greater for the frequent input (I_1). The prominent weights will essentially encode the features that are common across different classes of old and new inputs as the pre-neurons across those common feature regions in the input image will have frequent firing activity. This eventually helps the SNN to learn more common features with generic representations across different input patterns. We can visualize this as sort of regularization wherein the network tries to generalize over the input rather than overfitting such that the overall accuracy of the network improves.

Note that during the recovery phase in ASP, the weight updates are triggered only when a spike is fired by a postsynaptic excitatory neuron. Since the firing rate of the post-synaptic neurons is generally low, the weight update does not require many computational resources. Using a learning rule that updates the weights even at pre-synaptic spikes will initiate a weight change for every post neuron in the excitatory layer fully-connected to the pre-neuron. This will be computationally more expensive to simulate in software simulations.

Decay Phase

The decay phase in ASP learning involves weight leak to emulate forgetting of insignificant information to adapt to new data without catastrophic overlap of repre-

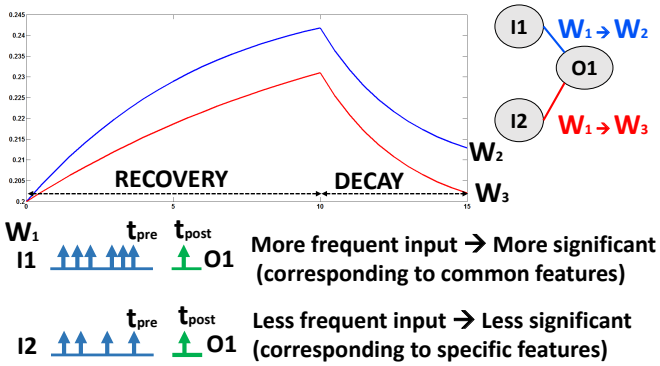


Fig. 5.8. Significance driven weight update observed with ASP. More frequent input spikes corresponding to the common features across old and new input patterns will have a greater weight update than the less frequent ones that correspond to specific features for a given input.

sentations. As discussed earlier, the weights can undergo an exponential/linear decay (refer to Fig. 6.5) towards a baseline value as

$$\begin{aligned} \tau_{leak}\frac{dw}{dt} &= -\alpha w \quad Exponential\ decay \\ \tau_{leak}\frac{dw}{dt} &= -\alpha_{lin} \quad Linear\ decay \\ \tau_{leak} &= k_{2,const}[(Post(t)+1) * 2^{(v_{thresh}+\theta)}] \end{aligned} \tag{5.8}$$

where α, α_{lin} are constant decay rates and τ_{leak} is the time constant of decay. τ_{leak} is a time dependent quantity that is proportional to the post-synaptic trace value ($Post(t)$ from Eqn. 5.6), the homeostatic membrane threshold value ($v_{thresh}+\theta$) at a given time instant and the current value of the synaptic weight w.

Earlier, in isolated weight decay learning, we observed that the network was overly plastic modifying its weights for any new pattern. However, in order to retain the learnt information, it is desirable that the corresponding weights should leak less. It is evident that a neuron that has learnt a particular pattern will have a higher synaptic weight. Also, the neuron will exhibit more spiking activity reckoned by the higher post trace value $Post(t)$ that will, subsequently, increase the time constant of decay, τ_{leak}. Higher τ_{leak} causes the weight to forget less. The overall leak rate(α/τ_{leak}) decreases with increasing τ_{leak}. Here, the value of $Post(t)$ is indicative of how recent

and latest the input pattern is (i.e. higher $Post(t)$ observed for most recent input). It does not account for the significance of the input pattern defined in terms of number of times a particular pattern has been presented to an SNN.

The homeostatic membrane threshold of a post-neuron is representative of the significance of the input pattern. A neuron's membrane threshold ($v_{thresh} + \theta$) will be high only when it is firing more. A neuron spikes more learning a given pattern well, when that pattern is presented several times to the network. Higher membrane threshold qualifies that the post-neuron (in excitatory layer) has learnt a significant input and its corresponponding synaptic connections should leak less. Whilst, the connections to an excitatory neuron that has learnt a pattern that the network has seen fewer number of times are forgotten. The weights corresponding to those neurons should eventually be modified to learn more recent patterns. Hence, the SNN learns to forget insignificant information while trying to learn more recent and retain significant, yet old, data using ASP.

A key observation here is that the weight leak in the decay phase (irrespective of linear or exponential decay) is dominated by the post-neuron's spiking activity (and membrane threshold). All weights connected to a post-neuron in the excitatory layer will have similar (*not exactly same due to dependence on current weight value*) decay time constant and hence show uniform leak dynamics during the decay phase. On the contrary, during recovery phase, the weight dynamics of each synapse will be different as it is determined by both the post and pre-neuronal spiking activity. In Fig. 5.8, we see that both the weights connected to the post-neuron O_1 exhibit almost same leak. As the weight update in recovery phase was more prominent for $I_1 - O_1$ (2nd term in Eqn. 5.7), it finally settles at a higher value than that of $I_2 - O_1$. Thus, the combined recovery/decay phase weight modulation in ASP ensures that more prominent weights (corresponding to common features across different input patterns) are forgotten less. This in turn helps to retain the common features (or weight connections) from old data (forgetting connections that are specific to the old

data) while learning a new input pattern making ASP learning more generic. Note, the values for all the hyperparameters used in Eqn. 5.6- 5.8 are shown in Table 5.1.

Table 5.1.
Parameter Table

Hyperparameters	**Value**
τ_{rec}	4 ms
τ_{acc}	40 ms
τ_{post}	80 ms
$offset$	0.2
$k_{const}/k_{1,const}$	0.01
$k_{2,const}$	1e2
α, α_{lin}	0.01

5.5 Experiments

5.5.1 Simulation Methodology

The ASP learning algorithm was implemented in BRIAN [112] that is an open source large-scale SNN simulator with parameterized functional models (LIF) for spiking neurons. We used the hierarchical SNN framework to perform digit recognition with the MNIST dataset [48]. The network topology and the associated synaptic connectivity configuration were programmed in the simulator. The spiking activity (or time instants of spikes) of pre- and post-neurons were monitored to track the corresponding pre/post synaptic traces that were used to estimate the weight updates in the recovery/decay learning phase of ASP. As mentioned earlier (in Section 4.3.1), we use the post-synaptic spike triggered weight update to speed up the simulation.

The effectiveness of the ASP algorithm was found to be dependent on the following system parameters that had to be tuned in a holistic manner for *stable-plastic* learning.

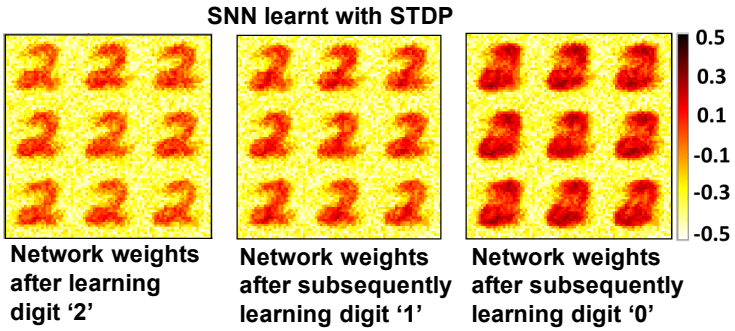
- ASP recovery phase parameters, namely, the decay time constants of the accumulative/recent presynaptic trace and postsynaptic trace.

- ASP decay phase parameters, namely, the decay rate of the weights (α, α_{lin}). The decay rate cannot be too high as it would result in faster weight leaking prohibiting the neurons from learning any representation. Very low decay rate will cause the connections to forget slowly that would result in subsequent overlapping of representations while learning new patterns. We used a nominal α value = 0.0001 for exponential decay and α_{lin} value = 0.01 in our simulations.

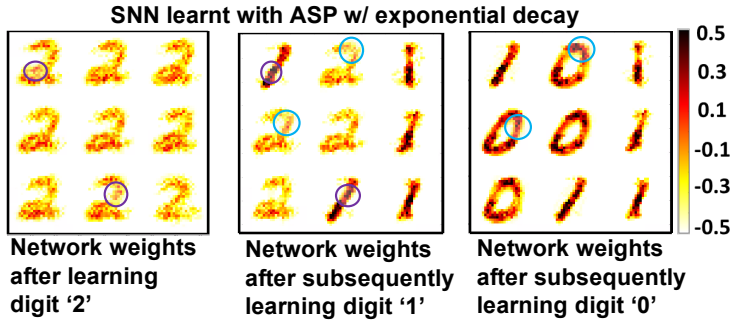
Please note, we use identical parameters for neuron and synapse models, homeostasis, input encoding and input image presentation time as Diehl & Cook [80] for fair comparison of our ASP learning with standard STDP learning. The standard STDP learning model is implemented using the power law weight dependent rule [80, 105]. Furthermore, across all comparative experiments between standard STDP and ASP described below, the SNNs in both cases were trained using the same number of training images with equivalent dynamic environment setup. After the training is complete, each excitatory neuron of the SNN encodes generalized representation of the individual patterns as shown in Fig. 6.3 (a). During testing, the neurons that have learnt a particular digit fire steadily for a corresponding test input based on which the predicted class is inferred. This inference approach is similar to that of Diehl & Cook.

5.5.2 Learning to forget with ASP in a dynamic digit recognition environment

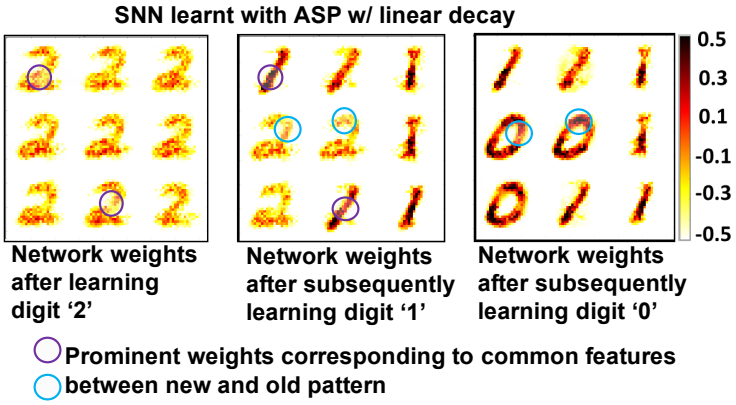
The main motivation for ASP is to come up with a biologically inspired learning paradigm that will facilitate on-line and adaptive learning in non-stationary environments. In order to create a dynamic environment with digit recognition framework, we presented the training instances of digits '0' through '9' sequentially with no reinforcement i.e. no training image is re-shown to the network. The digit categories are



(a)



(b)



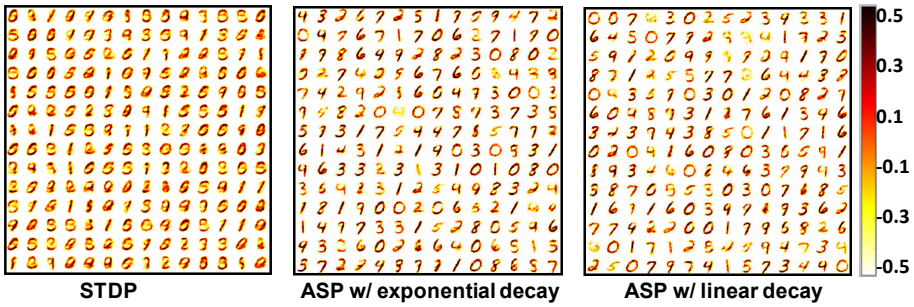
(c)

Fig. 5.9. Digit representations learnt in a dynamic environment with digits '2' through '0' shown sequentially to an SNN (with 9 excitatory neurons) (a) learnt with STDP (b) learnt with ASP with exponential decay (c) learnt with ASP with linear decay. Prominent weights corresponding to common features across different categories that are accentuated during the learning process with ASP have been selectively marked.

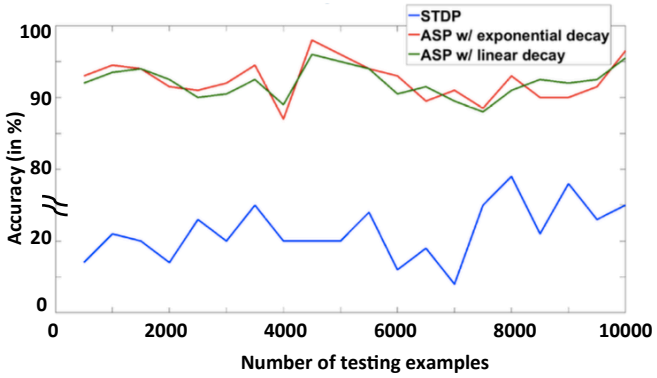
shown one-by-one with no intermixing of categories at any time during the training phase. Thus, the network must learn to forget old digit categories and retain the more recent ones while trying to learn a new category.

Fig. 5.9 shows the representations learnt in a fixed-size SNN (with 9 excitatory neurons) with traditional STDP learning against our ASP learning (both exponential and linear decay) in a dynamic environment (wherein digits '2' through '0' are presented sequentially in the order as '2→1→0'). We see that as the network is shown digit '1', ASP learnt SNN (Fig. 5.9 (b), (c)) forgets the already learnt connections for '2' and learns the new input. Also, ASP enables the SNN to learn more stably as some neurons corresponding to the older pattern '2' are retained while learning '1'. When the last digit '0' is presented to the ASP learnt SNN, the connections to the excitatory neurons that have learnt digit '2' are forgotten to learn 0 while the connections (or neurons) corresponding to recently learnt digit '1' remain intact. This is in coherence with our significance and latest data driven forgetting mechanism (incorporated in the decay phase in ASP) wherein older digits are forgotten to learn new digits. However, with STDP learnt SNN, the representations overlap thereby rendering the network useless towards the end of training. It is worth mentioning that the network representation after learning digit '1' in Fig. 5.9 (a) is not very different from the prior representation after learning digit '2'. This indistinguishability arises as the network weights corresponding to '1' overlap with the previously learnt weights of '2'. Specifically, the overlap occurs in the inclined region of '2' thereby making both the SNN configurations visibly similar.

A notable observation in Fig. 5.9 (b), (c) (ASP learnt SNN) is that a new digit ('1'/ '0') is learnt by forgetting specific features that have no similarity with the new pattern and retaining (and accentuating) common features from the old pattern '2'). As discussed earlier, the common features are encoded in the prominent weights (learnt during recovery phase of ASP). In Fig. 5.9, we can see from the color coding that the prominent weights (or common features between digit '2' and '1'; digit '2' and '0') have a higher intensity (almost black). This shows that ASP learns more generic



(a) Dynamic environment (no data reinforcement)

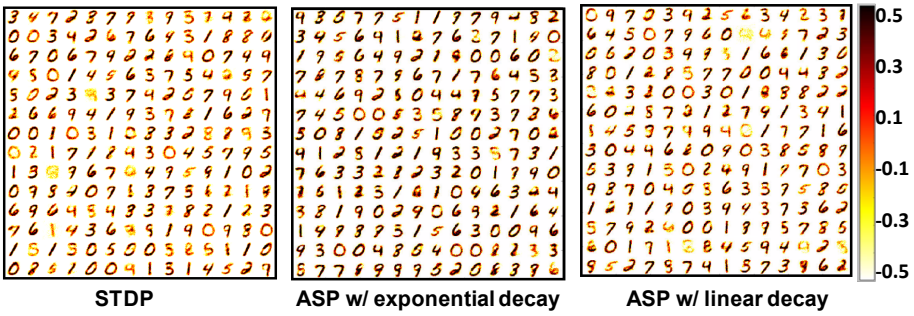


(b)

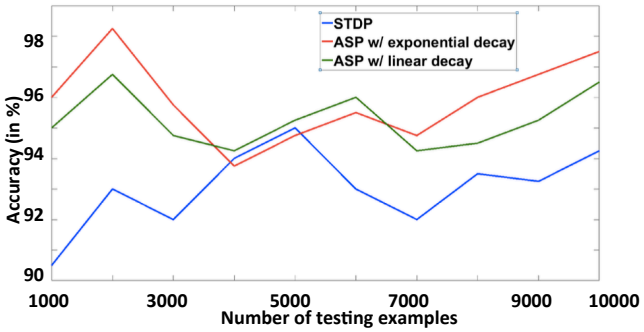
Fig. 5.10. (a) Digit representations (shown for a sample of 196 neurons) learnt in a dynamic environment with digits '0' through '9' shown sequentially to an SNN (with 6400 excitatory neurons) with ASP and traditional STDP learning. (b) Classification accuracy obtained from the networks trained on the different learning models in dynamic environment as the number of test instances shown to the network are varied.

representations associating relevant information (or connections in this context) from already learnt data to new data. Please note that similar behavior is observed in both exponential and linear decay based ASP.

Fig. 5.10 (a) shows SNNs (with 6400 excitatory neurons) learnt with ASP (both exponential and linear decay) and standard STDP in a dynamic environment when all digits '0' through '9' are presented sequentially. To ensure that the earlier digits are not completely forgotten, the number of training instances of each digit category were arranged in a decreasing order i.e. digit '0' had more training instances than digit '1' and so on. It has been stated earlier that the number of times a particular



(a) Data reinforcement



(b)

Fig. 5.11. (a) Digit representations (shown for a sample of 196 neurons) learnt in a data reinforced environment with digits '0' through '9' presented in an intermixed manner to an SNN (with 6400 excitatory neurons) with ASP and traditional STDP learning (b) Classification accuracy obtained from the networks as the number of test instances are varied in data reinforced environment.

category is shown to the SNN also quantifies the significance of that digit. So, the network will try to retain more significant data while learning recent patterns. It is clearly seen that the SNN learnt with our ASP encodes a better representation of the input patterns in comparison to the standard STDP trained network. In fact, the network is able to represent all digits suitably. In the latter case, we observe that most of the representations are illegible due to substantial overlap. Fig. 5.10 (b) shows the accuracy of the networks on the testing data from MNIST. While STDP yields a very low average accuracy of 23.30%, ASP with exponential (or linear) decay has an accuracy of 94.85% (or 94.2%). This reckons the effectiveness of ASP (predominantly

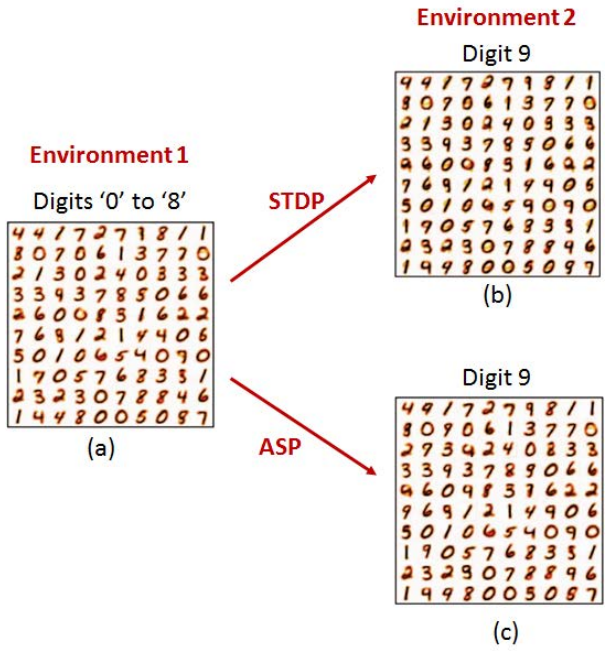


Fig. 5.12. (a) SNN after learning digits 0 through 8 with data reinforcement (b) The same SNN from (a) after being presented with digit 9 learnt with traditional STDP(c) The same SNN from (a) after being presented with digit 9 learnt with ASP.

forgetting mechanism) for a dynamic on-line learning scenario. Furthermore, the comparable accuracy with exponential and linear decay ASP indicates that both types of decay can be effectively used to emulate the forgetting behavior and implement significance driven learning.

5.5.3 Accuracy improvement with ASP over standard STDP

Till now, we have discussed how ASP enables an SNN to learn in a dynamic environment when the input digits are presented sequentially. As we saw earlier, SNN trained with standard STDP has significant overlap of representation when the inputs are presented without data reinforcement. For fair comparison for classification performance, we compare the accuracy for the two training methods when digits are presented to the SNN in an intermixed manner (i.e. with data reinforcement wherein

the different digit categories are iteratively repeated over the training process). Fig. 5.11 (a) shows the network weights at the end of training for a SNN with 6400 excitatory neurons learnt using ASP (both exponential and linear decay) and standard STDP. Fig. 5.11 (b) illustrates the accuracy of the SNNs on the MNIST testing data. ASP learning yields an average test accuracy of 96.8%/95.6% for exponential/linear decay, respectively, that is ∼2.5% higher than that of 94.3% obtained with STDP learning. This proves that the forgetting behavior realized with ASP does not dominate the overall plasticity of synapses making the overall learning balanced. The improvement in accuracy is also indicative of ASPs ability to generalize the network that causes it to learn more generic representations of the training data thereby avoiding overfitting. The slight difference in accuracy observed in the ASP case for linear and exponential decay can be attributed to the randomness in the Poisson based encoding of the input images.

To further elucidate the problem of sequential learning with STDP against ASP, lets discuss two scenarios: In Fig. 5.12, an SNN (with 100 excitatory neurons) is initially trained using STDP with data reinforcement (i.e. all image categories were intermixed) for the digits 0 through 8. In Scenario 1 continuing the STDP training, after this initial presentation of digits '0' to '8' (i.e. Environment 1), we subsequently presented the input image examples for the digit '9' without reinforcing or re-presenting any of the previous digits that resulted in data being overwritten. Fig. 5.12 (b) illustrates how weights associated with each neuron were affected in Environment 2 (i.e. only digit '9') with STDP learning, with many learned digits slowly transforming into the digit '9'. In Scenario 2, we learnt the same SNN from Fig. 5.12 (a) using ASP (instead of STDP) while showing only digit '9'. It is evident that changing the presentation of digits from Environment 1 to 2 is a dynamic or sequential learning scenario. Fig. 5.12 (c) shows the weights learnt in the SNN with ASP in Scenario 2. It is clearly seen that the network learns the digits '9' without any overlap while forgetting some old instances that are eventually transformed into digit 9. Evidently, the SNN learnt with STDP (62.8%) yields lower accuracy than

(a) AWGN (b) AWGN and reduced contrast

Fig. 5.13. Noisy-MNIST images with (a) Additive White Gaussian Noise (AWGN) (b) Reduced Contrast with AWGN.

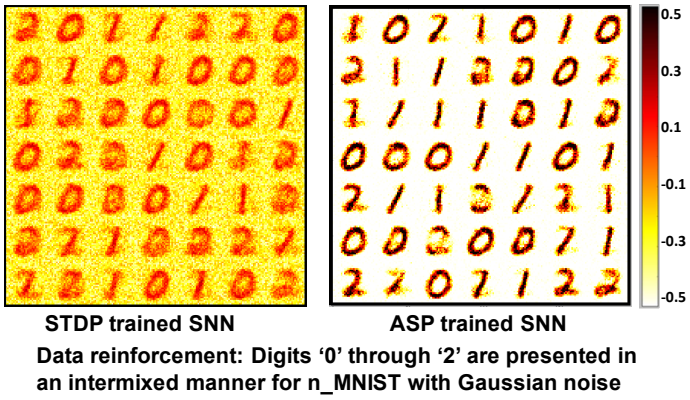


Fig. 5.14. Digit representations learnt with digits '0' through '2' presented in an intermixed manner to an SNN (with 49 excitatory neurons) with STDP and ASP for Noisy-MNIST images with AWGN as Fig. 5.13 (a).

that of ASP (73.4%). The lower accuracy with ASP in this case as compared to Fig. 5.11/5.10 is due to the lesser number of excitatory neurons used in the SNN. Thus, in the absence of data reinforcement or retraining, training with STDP alone does result in catastrophic forgetting of old data in a sequential learning environment.

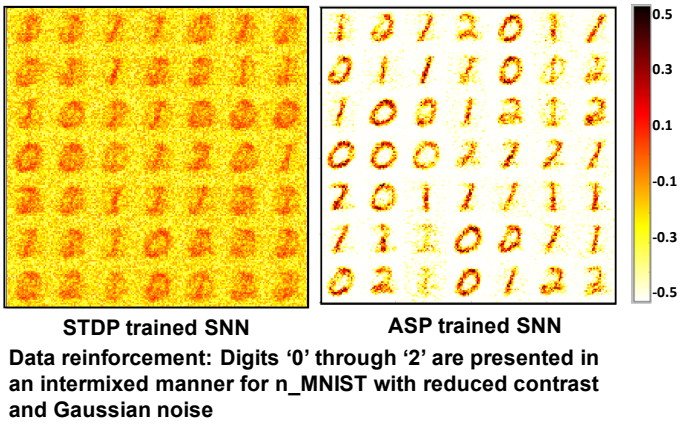


Fig. 5.15. Digit representations learnt with digits '0' through '2' presented in an intermixed manner to an SNN (with 49 excitatory neurons) with STDP and ASP for Noisy-MNIST images with AWGN and reduced contrast as Fig. 5.13 (b).

5.5.4 Denoising with ASP

The decay incorporated in ASP besides rendering an SNN to self-adapt in a dynamic environment also offers a key advantage of denoising by extracting relevant features from noisy input images. To demonstrate this, we trained an SNN with Noisy-MNIST (n-MNIST) [113] images as shown in Fig. 5.13. Fig. 5.13 illustrates the sample images from the n-MNIST dataset with (a) Additive White Gaussian Noise (AWGN) (b) a combination of AWGN and reduced contrast. Fig. 5.14 shows SNNs (with 49 excitatory neurons) learnt with ASP and standard STDP when digits '0' through '2' are presented in an intermixed manner (with data reinforcement) for the n-MNIST images from Fig. 5.13 (a). It is clearly seen that both STDP and ASP are able to encode the corresponding digit representations. However, ASP owing to the dynamic decay mechanism is able to complete eliminate the background noise while selectively attending (or learning) the task relevant information in the data. It is worth mentioning that, in the denoising experiments below, we compare the STDP trained SNN against ASP with exponential decay and show the results for the same. However, similar results can also be obtained for ASP with linear decay.

To further establish the adeptness of our ASP rule in reducing the influence of noisy inputs, we train an SNN on the n-MNIST images from Fig. 5.13 (b) that have AWGN with reduced contrast. The intensity of the digit pixels are now equalized with that of the background noise pixels on account of the reduced contrast (as seen in Fig. 5.13 (b)). Fig. 5.15 shows the representations learnt with ASP and standard STDP for the same data-reinforced recognition scenario as that of Fig. 5.14. Now, we can clearly see that that, in spite of the reduced contrast, the ASP learning is able to distinguish the relevant information (or digit patterns) from the noisy background and learn robust representations. The weight decay phenomenon with varying leak rate in ASP (refer to Section 4.3.2) retains significant information while forgetting (or leaking) the weights corresponding to irrelevant information thereby enabling the selective exclusion of the background noise. In contrast, standard STDP owing to the absence of dynamic weight decay is unable to filter out the relevant digit information from the background noise of similar intensity. Thus, the representation learnt in the latter case are mostly cluttered and obscure. We would like to note that, in both cases of standard STDP and ASP, the SNNs were trained using the same number of training images.

To quantify the classification performance on the n-MNIST images, we trained a larger SNN (with 6400 excitatory neurons) with standard STDP and ASP in a data-reinforced environment for digits '0' through '9'. STDP in both cases yields a lower accuracy (on the testing data from n-MNIST) than ASP. While the average accuracy with STDP is 87.4% on just AWGN based noisy images (Fig. 5.13 (a)), the accuracy degrades to 52.1% for the AWGN with reduced contrast images (Fig. 5.13 (b)). ASP, on the other hand, consistently performs better with 93.8% (85.6%) classification accuracy for AWGN (reduced contrast with AWGN) n-MNIST images. This illustrates how the adaptive decay process incorporated in ASP automatically encodes selective filtering and attention towards task relevant features in the input data further improving the robustness of the unsupervised learning paradigm.

5.6 Conclusion

In this chapter, we discussed a novel bio-inspired unsupervised learning rule Adaptive Synaptic Plasticity (ASP) for real time on-line learning with Spiking Neural Networks (SNNs). We integrate weight decay with traditional STDP to devise our dynamic weight update ASP mechanism that addresses *"catastrophic forgetting"*, a key issue across conventional learning models. The weight decay employed in ASP emulates the "forgetting" behavior of the mammalian brain. While STDP helps in learning new input patterns, the retention of significant, yet old, data or gradual forgetting of insignificant information is attained with weight decay. In ASP, we modulate the leak rate of the synaptic weight decay process using the temporal dynamics of pre- and post-synaptic neurons that maintains a balance between continuous learning and forgetting to construct a *stable-plastic* self-adaptive SNN for evolving environments. ASP owing to its significance driven "forgetting while learning" formulation, enables an SNN to generalize over the training data with more generic learning of representations (thus avoiding overfitting) yielding significantly improved accuracy over traditional STDP methods.

Further, we would like to point out that in order to prevent "catastrophic forgetting" in STDP-learnt SNNs, the network is generally re-trained with both the new and the old information (already learnt) when the network has to learn a new class. However, in on-line real time learning, it is often impractical and even expensive to store all old data samples for retraining. ASP offers a promising solution for real time dynamic learning without the expensive re-training procedure. In addition to the dynamic learning, the adaptive weight decay mechanism in ASP also enables unsupervised denoising or selective attention towards relevant features in the input data further improving the robustness of ASP learning.

Finally, the recent resurgence of neural networks after the *artificial intelligence winter era* can be attributed to the learning of good, flexible representations from the visual world. In the current digital era, incorporating active sensing and diverse task

modelling (classification, decision making, analytics) within the same learning model will no doubt shape the nature of learning representations in future cognitive systems. To that effect, the *"learning to forget"* behavior realized with ASP (inspired deeply from biological principles) provides a potentially exciting and promising direction towards improved representation learning with the emerging computing paradigm of SNNs.

6. LEARNING TO GENERATE SEQUENCES WITH COMBINATION OF HEBBIAN AND NON-HEBBIAN PLASTICITY IN RECURRENT SPIKING NEURAL NETWORKS

6.1 Introduction

Learning to recognize, predict and generate spatio-temporal sequence of spikes is a hallmark of the nervous system. It is critical to the brains ability for anticipating the next ring of a telephone or next action/movement of an athlete. Several neuroscience works have shown that such abilities can emerge from dynamically evolving patterns of neural activity generated in a recurrently connected neocortex [114–116]. Implementing a *'stable and plastic* recurrent spiking neural network for learning and generating sequences with long-range structure or context in practical applications (such as text prediction, video description etc.), however, remains an open problem.

Here, we develop a reservoir spiking neural model with a biologically relevant architecture and unsupervised plasticity mechanisms that learns to generate sequences with complex spatio-temporal relations. Generally, reservoir networks, owing to their high level of recurrence, operate in a dynamic regime wherein stable input driven periodic activity and chaotic activity tend to coincide [117, 118]. We show that the chaotic activity can be reduced by effective tuning of connections within a reservoir with a multi-time scale learning rule and diverse plasticity mechanisms. This helps in the robust learning of stable correlated activity that is even resistant to noise. This further enables the reliable generation of learnt sequences over multiple trials lending our reservoir model a key feature characteristic of biological systems: *the ability to remember the previous state and return to the sequence being generated* in presence of perturbations.

Recent efforts in building functional spiking neural systems with Spike Timing Dependent Plasticity (STDP) for self-learning on practical recognition tasks have been mostly focused on feed-forward and hierarchical deep/shallow architectures [22,80,119–121] with minimal recurrent connections (for instance, recurrent inhibitory connections incorporated with such architectures for inducing competition and stabilizing neural activity in an unsupervised learning environment). Another body of work in the spiking domain encompasses 'Reservoir or Liquid Computing frameworks that attempt to capture the anatomy of the neocortex with substantial recurrent connectivity between groups of excitatory and inhibitory spiking neurons [122–124]. [125] proposed a similar randomly connected recurrent model, termed as echo state network, that uses analog sigmoidal neurons (instead of spiking) as fundamental computational units. In both liquid or echo state frameworks, the burden of training the recurrent connections is relaxed by fixing the connectivity within the reservoir and that from input to reservoir. Instead, an output layer of readout neurons is trained (generally in a supervised manner i.e. with class labels) to extract the information from the reservoir [126, 127]. However, the fixed connectivity severely limits the ability of such frameworks to do general learning over varied applications. Such networks perform poorly as the number of possible patterns (or classes) as well as the temporal correlations (or context) between patterns increases.

It was shown recently that incorporation of synaptic plasticity within the reservoir can result in the emergence of long-term memory [116] that can help in learning and inferring relationships between inputs [128]. However, the main aim of those works were to build neural computing models that suitably elucidate the activity of the neocortex to better understand the mammalian brain. On the other hand, in this work, we take a more practical approach to engineer a recurrent spiking model capable of character-level recognition and word prediction. Essentially, given a dictionary of visual words, our spiking reservoir model learns to recognize each visual character (or alphabet) as well as the context/relation between subsequent characters of different words such that it can consistently generate the entire word. Note, in this work, we

use the term 'reservoir' in the context of a recurrent spiking neural network similar to that of a Liquid State Machine (LSM) [122] with spike-driven dynamics compatible with STDP.

Similar to [116,128], we modify the synaptic weights of the connections from input to reservoir as well as the recurrent connections within the reservoir to develop the character-level language model. The learning is performed over different time scales. While the fast input to reservoir learning, operating in millisecond range, facilitates in recognition of individual characters, the slower learning of the recurrent connections, operating in the range of 100 milliseconds, within the reservoir enables the network to learn the context between the characters such that the learnt model can generate words by predicting one character at a time.

In addition, to reduce the attractor states that emerge from the feedback loops in the reservoir, we introduce a non-Hebbian adaptive weight decay mechanism in the learning rule. The decay in addition to STDP enables synaptic depression for hyperactive neurons in the reservoir that result from strong feedback dynamics. We show that the combined effects of Hebbian and non-Hebbian plasticity mechanisms results in a *stable-plastic* recurrent SNN capable of generating sequences reliably. We also justify the effectiveness of the combined plasticity scheme by analyzing the eigen value spectra of the synaptic connections of the reservoir before and after learning. As shown in later sections, this theoretical analysis provides a key insight about the inclusion of non-Hebbian decay to reduce chaotic dynamics within a reservoir.

6.2 Materials and Methods

6.2.1 Reservoir Model: Framework and Implementation

Network Architecture

The general architecture we consider is a 2-layered network as shown in Fig.8.1(a). The topology consists of an input layer connected to a reservoir of N Leaky-Integrate-

and-Fire (LIF) neurons [80], with a connection probability P_{IN} of 30%. The input layer contains the pixel image data (with one neuron per pixel), corresponding to the visual words or characters in the dictionary. Of the total N neurons in the reservoir, 80% are excitatory and 20% are inhibitory, in accordance with the ratio observed in the mammalian cortex [129]. The reservoir is composed of all possible combinations of recurrent connections, depending upon the pre- and post-neuron type at each synapse: E→E, E→I, I→E, I→I, where E (I) denote excitatory (inhibitory) neurons. These recurrent connections are set randomly with a relatively sparse connection probability of $P_{EE,EI,IE,II}$. Here, all connections going to excitatory neurons (i.e. input to reservoir excitatory neurons and E→E connections within the reservoir) are plastic, while all other connections maintain their initial random values.

Note that the synaptic weight modification of the plastic connections shown in Fig. 8.1(a) helps in learning the rich temporal information present in the input data and also enables the understanding of contextual dependencies (for learning a word from individual characters) that span over multiple time steps. Another interesting property of our model is that it does not contain a readout layer that is generally present across all conventional reservoir spiking models [122, 123, 130] for sequence recognition. The synaptic plasticity from input to reservoir, specifically, helps in learning the generalized representations of the input patterns (i.e. images of characters in this case). This in turn enables us to perform unsupervised inference without a readout layer of output neurons.

In Fig. 8.1 (a), the connection probabilities and the Excitatory(E)/Inhibitory(I) neuron ratio in the reservoir are chosen such that we have a balanced contribution of E/I synaptic currents that contribute to the spontaneous activity of the reservoir [129]. Generally, the number of E/I neurons in the reservoir maintain a 4:1 ratio as observed in the neocortex. Since E>I, inhibitory connections must be larger than excitatory connections (i.e. $P_{EE} < P_{EI} \times P_{IE}$). The connection probabilities are usually set randomly while ensuring that E/I balance is maintained. However, a simple excitation of the reservoir with noisy Poisson inputs and observing the trajectory (or

neuronal firing activity from the reservoir) generally helps to choose a probability range. For such random noisy inputs, the reservoir will be chaotic and should ideally show irregular trajectory with varying firing activity for every stimulation. However, if excitation dominates ($P_{EE} > P_{EI} \times P_{IE}$), then, there will be a mean positive drift of the neuronal membrane potential towards the threshold, resulting in a fixed trajectory (or regular firing activity). In contrast, if excitation and inhibition balance each other, the membrane potential will follow a random trajectory resulting in a trajectory with Poisson statistics as expected. This happens when $P_{EE} < P_{EI} \times P_{IE}$. While it might seem that a lot of synaptic fine-tuning is required, on the network level, such an E/I balance can arise dynamically if two conditions are met. First, connections must be sparse and random as shown in [131]. Second, connections for inhibitory must be greater than excitatory. We follow both these rules in addition to observing the reservoir trajectory for random noise triggering while initializing our model. P_{IN} (30%) is set such that a minimal firing rate of 15-20 Hz is observed from the reservoir for a ∼45 Hz input that will allow significant synaptic learning within the STDP timing window of simulation.

Synaptic Plasticity & Homeostasis

The synapses connecting the input to excitatory reservoir neurons (In→Exc) and the E→E connections within the reservoir are trained using a combination of Hebbian STDP and non-Hebbian Heterosynaptic plasticity that prevents strong feedback loops in the recurrent connections. STDP is a widely used weight update rule to accomplish unsupervised learning in SNNs. The weights of the synaptic connections are strengthened or weakened based on the time interval elapsed between pre- and post-synaptic spikes. We adopt different forms of weight dependent STDP rule to compute the weight updates. To implement the non-Hebbian plasticity, we introduce an adaptive decay mechanism [23, 132] (that only depends on the state of the post-synaptic neuron) in the weight update rule.

In addition to synaptic plasticity, we employ a homeostatic membrane threshold mechanism [104], for the excitatory neurons in the reservoir that regulates the firing threshold to prevent a neuron from being hyperactive and dominating the overall response. Specifically, each excitatory neurons membrane threshold is not only determined by v_{thresh} but by $v_{thresh} + \theta$, where θ is increased each time the neuron fires and then decays exponentially [105]. It is worth mentioning here that the interplay between homeostatic threshold and combined Hebbian/non-Hebbian plasticity results in a stable and plastic network with a balance between excitatory and inhibitory currents at each neuron in the reservoir.

In [116, 123, 128], the authors have used inhibitory STDP to induce homeostasis or balance the activity of the excitatory neurons by modifying the synaptic weights of I→E recurrent connections. This is in stark contrast to our model where only the E→E connections within the reservoir are plastic. We note that among all the recurrent connections, E→E help learn the context between subsequent patterns while reinforcing patterns with similar statistics. The remaining connections mainly contribute to fostering competition (E→I) among different excitatory neurons to learn different patterns while maintaining a balanced and asynchronous firing activity (I→I, I→E) in the reservoir. Since the main aim of our model is to learn the underlying representations of visual inputs and understand the correlation between subsequent patterns, we can achieve this simply with E→E plasticity, while maintaining an optimum fixed and sparse connectivity across remaining connections as specified in Fig.8.1(a) .

6.2.2 Sequence Learning with the Proposed Reservoir Model

Given a visual word CAT composed of individual characters as shown in Fig.8.1 (b), our model processes each character individually to learn the representations of each character with Input to Excitatory reservoir (In→Exc) plasticity. Simultaneously, the plasticity among the E→E connections within the reservoir should be such

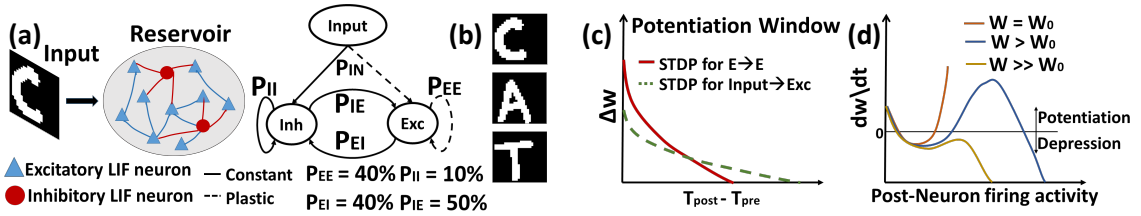


Fig. 6.1. (a) General topology of Recurrent SNN used for sequence learning and prediction (b) Sample image of dictionary of visual words (c) STDP Potentiation window for Hebbian Phase learning of In→Exc & E→E reservoir connections over diverse time scales (d) Synaptic changes with the combined Hebbian/non-Hebbian Plasticity as a function of rate of post-synaptic neuron that prevents strong attractor dynamics by regulating the over-potentiation of synapses. Note, (c) & (d) are cartoons (that do not depict empirical data) to show the behavior of STDP based weight change for slow/steep learning in (c) and effect of inclusion of non-hebbian decay on the hyperactive neuronal weights in (d).

that the network learns that CAT is one entity with a sequential correlation (i.e. C is followed by A followed by T). During testing, when the model is presented with a test image C, the network should recognize the character and output the next most probable character from the entity it has learnt previously. In order to perform such sequence generation, we conduct the learning in two phases as described below.

Hebbian Phase

In this phase, we modify the synaptic weights of both In→Exc and E→E connections with different forms of STDP. For In→Exc plasticity that helps in learning the underlying representations of the individual images/characters, we perform the weight updates using the power law weight dependent STDP rule [80,105], illustrated in Fig.8.1 (c). To improve simulation speed, the weight dynamics are computed using synaptic traces as proposed in [110]. When a post-synaptic neuron fires a spike, the weight change Δw is calculated as

$$\Delta w_{In \rightarrow Exc} = \eta[(x_{pre} - offset)(w_{max} - w)^{\mu})]; x_{pre} += 1 \tag{6.1}$$

where η (0.05) is the learning rate, w_{max} (1.0) is the maximum constraint imposed on the synaptic weight, x_{pre} is the pre-synaptic trace value that exponentially decays with $\tau_{pre} = 30ms$ and w is the current weight value. The synaptic strength is increased by $w = w + \Delta w_{In \rightarrow Exc}$ if a pre-neuron subsequently causes the connected post-neuron to fire which signifies a strong causal relationship. On the other hand, the synaptic strength is decreased for larger spike time differences as determined by the $offset$. This training enables the excitatory neurons in the reservoir (connected to the input) to encode a generic representation of an image pattern in their corresponding In→Exc weights.

Concurrent to the above training, we simultaneously modify the E→E connections within the reservoir with steeper STDP learning as shown in Fig. 8.1 (c). For the E→E weight updates, we use a modified version of the exponential weight-dependent STDP rule [80, 133]. The synaptic weight updates based upon the arrival of pre- and post-synaptic spikes are again calculated using synaptic traces as follows:

$$\begin{aligned} w &= w - \eta_1 [x_{post} w^{\mu}] (\textit{when pre-neuron fires}); x_{pre'} = 1 \\ w &= w + \eta_2 [x_{post} x_{pre} (w_{max} - w)^{\mu})]; (\textit{when post-neuron fires}); x_{post} = 1 \end{aligned} \tag{6.2}$$

where $\eta_{1,2}$ is the learning rate (0.002, 0.01) and w_{max} value is 0.5. Similar to above learning, the weights are potentiated or depressed based on the spike timing correlation. However, with slower learning rate and smaller time constants of decay for the traces ($\tau_{pre',post} = 10, 20ms$), significant synaptic weight updates are carried out for really small spiking differences between pre- and post- neurons in this case. This slow learning is desirable as the E→E connections must encode the correlation between the individual images. Hence, the E→E connections should get updated only when both the pre and post excitatory neuron in the reservoir have spiked very closely and over longer periods of time. The synaptic traces will increase more and cause meaningful weight updates for such stronger causal relationship. Please note that all the pre- and post-synaptic traces in the above learning rules are disjoint and calculated separately. Furthermore, the weight dependence terms in the above equations prevents abrupt or fast change in weight values.

Non-Hebbian Phase

In addition to the exponential STDP learning, the E→E connections further undergo a decay towards a baseline value ($w_0 = 0.2$) to prevent the reservoir plasticity to cause strong feedback loops in the network thereby curtailing the emergence of strong attractor dynamics. Such attractor states cause the network dynamics to converge to the same state for different input sequences, strongly degrading the inference capability of the network. For instance, if the network learns the words CAT, COT wherein the recurrent connections are strongly correlated for excitatory neurons representing C, O, T, then, the network will only output O when presented with C during testing. Ideally, we would like our Reservoir Model to give all possible sequences or words for a given input.

The decay of E→E synaptic weights is performed at every simulation time step as

$$\frac{dw}{dt} = -\gamma(t)(w - w_0) \tag{6.3}$$

where $\gamma(t)$ is the decay rate that is a time dependent quantity proportional to the squared post-synaptic trace value ($(x_{post})^2$ from Eqn. 2) and the homeostatic membrane threshold value ($v_{thresh} + \theta$) at a given time instant. The direction of change depends on the present value w of the synaptic weight in relation to the baseline value w_0. If the post-synaptic neuron fires more due to the strong feedback loops, the decay rate proportionately increases weakening the E→E connections thereby reducing the influence of the attractor state on the network dynamics.

In some situations, the attractor states may also arise due to disproportionate input sequences. For instance, if the network learns the words CAT/CRAFT, CRAFT on account of being a longer sequence will induce stronger correlation between the neurons representing the underlying characters thereby creating an imbalanced recurrent model. The homeostatic membrane threshold of an excitatory neuron is representative of the length of the input pattern. For a longer sequence, E→E connections are reinforced more with the exponential rule (Eqn. 2) such that the neuron

learns the correlation well. The reinforcement in turns causes the neurons to spike more thereby increasing their membrane threshold. Higher membrane threshold correspondingly increases the decay rate, thus, preventing the length of input sequences from causing strong attractor states. Please note that this decay mechanism is non-Hebbian (does not involve pre, post spike timing correlation) and bears resemblance with the heterosynaptic plasticity observed in the mammalian brain that prevents runaway synaptic dynamics and stabilizes the distribution of synaptic weights [132].

The rate of change of weight with our combined plasticity scheme is illustrated in Fig.8.1 (d). The adaptive decay mechanism helps in un-learning the weights of neurons that become hyperactive due to strong feedback loops. For synaptic weights that are larger than the baseline value w_0, it allows synaptic depression of E→E connections even for high post-neuronal firing activity that is not possible with lone exponential STDP learning. In addition to reducing the attractor states, the synaptic decay acts a memory consolidation mechanism wherein certain connections that are relevant for the reliable generation of the sequence are selectively potentiated while depressing recurrent connections that lead to undesirable correlation. As mentioned earlier, both In→Exc and E→E connections are updated simultaneously with the above mentioned rules (Eqn. 1-3). The In→Exc learning is done at each time step based upon pre/post firing (order of ms). In contrast, the Exc→Exc weight updates occur slowly over multiple time steps due to the slower STDP learning. This concerted interplay of the multiple plasticity mechanisms (both Hebbian and non-Hebbian) leads to a stable reservoir that avoids attractor states and reliably generates all possible answers for different sequence of words learnt.

In our simulations, each individual character of the word sequence (that is a 28x28 pixel image of A to Z) is presented to the reservoir as Poisson spike trains (with firing rates proportional to the intensity of the pixels) for 350 ms with a simulation time step of 0.5 ms. For instance, while learning CAT, each individual character (C, A, T) is shown individually and sequentially for 350 ms each. Then, before presenting the next sequence/word (for instance, COT or a different representation of CAT),

there is a 300 ms phase without any inputs to allow all variables of all neurons to decay to their resting values (except the adaptive membrane threshold). This sequential presentation of the characters (without resetting the membrane potential of the neurons until the entire sequence or word is shown) helps the E→E connections to learn the correlation between them.

Note, the appendix (Section 6.5 at the end of the chapter) contains additional details regarding the neuron/synapse model, input encoding method and the training/assignment/inference methodology. Please refer it to get further insights about the intrinsic parameter values.

6.3 Results

The proposed reservoir model and learning was implemented in BRIAN [112]. We created a dictionary of visual words (samples shown in Fig.6.3 (a)) from the handwritten characters in Char74K dataset [134] that were used to train and test our reservoir model for sequence learning.

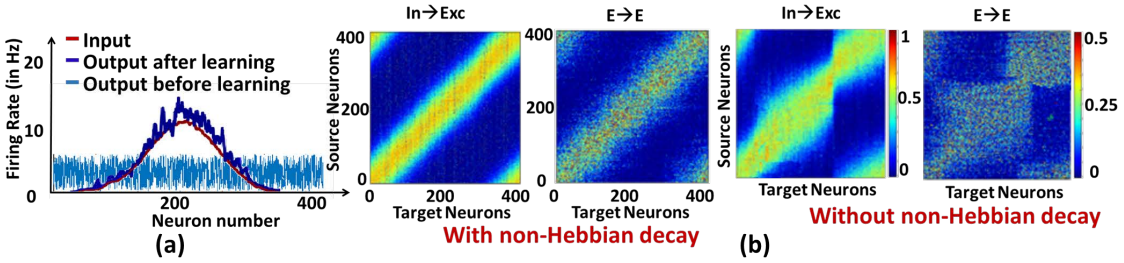


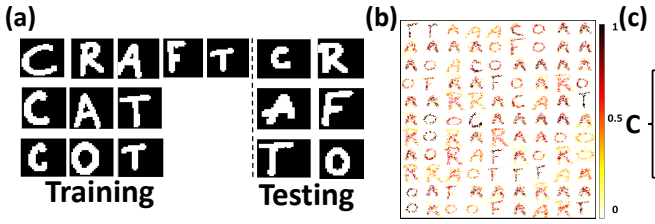
Fig. 6.2. (a) Response pattern of reservoir neurons for Gaussian Input profile (average: 5 Hz) before and after learning (b) Visualization of the weight matrices between Input→Exc and E→E reservoir connections learnt with and without non-Hebbian decay

We first show the effectiveness of the combined plasticity mechanism in reducing the strong attractor dynamics that emerge in a recurrent spiking neural network. We simulated a reservoir model of 400 excitatory, 100 inhibitory LIF neurons with an in-

put layer composed of 400 Poisson Spike Generators that gives the input a Gaussian shaped firing rate profile. In this case, learning the In→Exc and E→E connections as described earlier (along with non-Hebbian decay) causes the reservoir neurons spiking response (that fired randomly before training) to match the firing rate profile of the input as shown in Fig.6.2 (a). Correspondingly, the weight matrices for the In→Exc connections and E→E connections within the reservoir form a diagonal structure representative of the bell-shaped Gaussian distribution of the input patterns over the reservoir neurons, as illustrated in Fig.6.2 (b). In contrast, the weight values accumulate in certain regions causing attractor dynamics in the reservoir because of the strong feedback loops when the non-Hebbian decay based plasticity is not incorporated in the E→E synaptic learning as shown in Fig.6.2 (b). Such weight crowding is caused by the Hebbian nature of lone STDP learning. Since STDP causes reinforcement of correlated activity, the feedback loops between sub-groups of neurons that are strongly interconnected due to the recurrent dynamics of the reservoir will over-potentiate the E→E connections, further causing them to be overly active. As a result, the weights get crowded instead of having a homogeneous distribution. Inclusion of non-Hebbian decay in the learning mechanism helps in decreasing the activity of such sub-groups of neurons by enabling synaptic depression even at high post-synaptic firing rates as seen earlier (refer to Fig.8.1 (d)).

Now, we discuss the recognition and generation of words from visual characters. We simulated a reservoir model of 200 neurons (160 excitatory, 40 inhibitory) to recognize a dictionary of words: CAT, CRAFT, COT. Please note, the words selected are such that some sequences have more common underlying characters (such as CAT, COT) than others. To validate that our model is effective and can generate all possible learnt sequences consistently without any bias towards a particular sequence, we use the above selection. For training, we use 200 different representations for each word composed of dissimilar characters (i.e. total 600 words for training). For testing, we used 100 distinct representations of each character. Fig. 6.3 (a) shows the sample training and testing images used. After training, the In→Exc connections that are

essentially responsible for recognizing the individual characters of the word encode the generalized representations of the individual characters as shown in Fig.6.3 (b). Individual characters get associated with different excitatory neurons in the reservoir that fire steadily when presented with an input similar to the pattern it has learnt.



Sequence	% Correct Prediction	Sequence	% Correct Prediction
CAT	30	PEN	92
COT	26	MAN	90
CRAFT	32	BIRD	95
DOG	93	TOW	83

Fig. 6.3. (a) Sample training and testing images of visual words (b) Representations encoded by In→Exc connections of 100 excitatory neurons in a 200-neuron reservoir. The color intensity of the patterns are representative of the value of synaptic weights (after training) with lowest intensity (white) corresponding to a weight value of '0' and highest intensity (black) corresponding to '1'. (c) Percentage of correct predictions made by the 400-neuron reservoir for different sequences during testing. The prediction accuracy is averaged across 100 trials of different presentations of the test input characters: 'C', 'D', 'P', 'M', 'B', 'T'.

During testing, when an input pattern is shown, several excitatory neurons assigned to different characters may potentially fire. For example, for a test image of C, a group of neurons associated to D may also fire. The spiking/firing rates for different groups will however differ (say spiking rate of C > D). However, since we learn the E→E connections in the reservoir while presenting the individual characters sequentially (without resetting the membrane potentials) during training, the average firing rates of the neurons in the reservoir interestingly follow a particular sequence. For a test image of C, the top-2 average spiking activity are observed for neurons associated with C, A. Further, the difference between the top-2 spiking activities is quite low (an average of $\sim$ 3-4). Similarly, based on the second highest spiking activity, we input the next character (i.e. A in this case) to which the top-2 spiking activity recorded are for neurons associated with A, T. Next, when we input T, while

the highest spiking activity is observed for T, the second highest activity is quite random (associated with different neurons for varying test presentations) with an average spiking difference that is greater than 10. The large spiking difference in the top-2 activity indicates that the last character of the sequence has been recognized by the reservoir for a given test trial and no further inputs are then provided to the reservoir. Please note, similar to training, the membrane potentials of the neurons in the reservoir are not reset in a test trial until the entire sequence/word is generated or a large spiking difference in top-2 activity is observed. We observe such consistent sequential generation of top-2 spiking activity across all test trials. As the dictionary contains 3 words starting with C, the second highest spiking activity alters between R, A, O for different trials. This difference arises due to the randomness in the Poisson distribution of inputs. In fact, for 100 trials, when the reservoir model was presented with test input C, the network yielded a correct word from the dictionary 85 times with CRAFT, CAT, COT generated 34, 23, 28 times respectively. In the remaining 15 trials, some garbage words such as CAFT, CRT etc. were generated.

Next, we simulated a reservoir model of 400 neurons (320 excitatory, 80 inhibitory) to learn a larger dictionary with: CAT, COT, CRAFT, DOG, PET, MAN, BIRD, TOW. Fig.6.3 (c) shows the number of times a correct sequence is generated by the reservoir across 100 different presentations of the first character of every word: C, D, M, B, T. The average accuracy of the reservoir is 91.2% wherein the sequence generation was more accurate for PET, DOG, BIRD that share less characters with the remaining words of the dictionary. Minimum accuracy is observed for TOW since presentation of T in most cases yielded a large difference in the top-2 spiking activity of the neurons. This can be attributed to the fact that T being the last character for most words limits the neuron learning a T to develop contextual dependencies towards other neurons. Few garbage words generated by the reservoir include COW, CRAT, DOT, BRD, PT, MAT, BRAT among others. Of the garbage words, it is quite remarkable to see that some are actual English words that the reservoir generates not having seen them earlier. Another noteworthy observation is that most garbage

words end with W, T, D, that are essentially the last characters of the words in the dictionary. Please note that in the words present in the dictionary, repetition among characters is not present (for instance, SEEN or RALLY). In such cases, our reservoir model generates SEN, RALY instead of the correct words. Since the generation of sequences in our reservoir model is based on top-2 spiking activity, when E is presented to the reservoir, the top-2 highest spiking activity with minor difference is observed for neurons associated with E, N. We cannot identify such repetition among characters with this scheme.

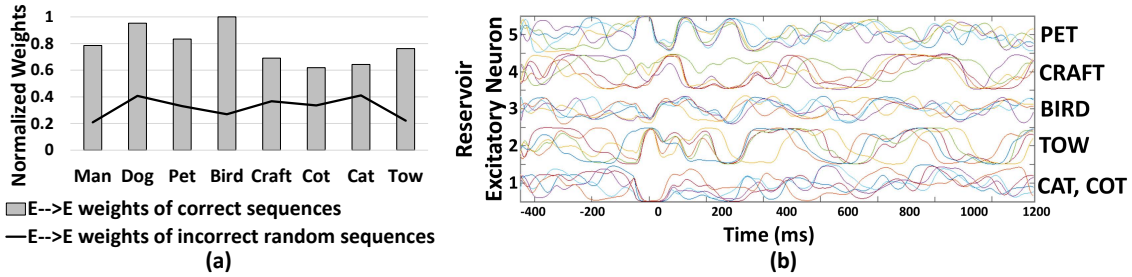


Fig. 6.4. (a) Normalized average value of trained weights of E→E reservoir connections corresponding to different correct/incorrect sequences predicted by the 400-neuron reservoir model. All weight values are normalized with respect to the highest average value (0.42) recorded for the sequence 'BIRD' in this case. (b) Firing rates/Trajectories of 5 excitatory neurons in the 400-neuron reservoir encoding different characters. Different color coding of trajectories specify different trials.

In order to demonstrate the effectiveness of our combined learning scheme for generating sequences, we recorded the average synaptic weights of the E→E connections (that encode the correlation) learnt among the excitatory neurons associated with different characters. Fig.8.7 (a) shows that the average value of the recurrent connections learnt by the 400-neuron reservoir model described above, differs across different words with minimal variation. In fact, for words with similarities (such as CAT, COT, CRAFT) the weights have almost similar values. The slight variation across the weight values is indicative of the fact that the reservoir model (after

learning) has reduced chaotic activity. Generally, all recurrent networks exhibit some chaotic activity owing to the feedback connections that, if not controlled, can cause abrupt changes in neuronal activity leading to a severe degradation in their inference ability. The reduction in chaotic states enables our model to produce stable recurrent activity while generating a particular sequence. This further establishes the suitability of our combined learning scheme for reservoir plasticity. Additionally, the strength of the connections between neurons associated with random sequences (that are not present in the dictionary) such as DR, POM, TMO, CG etc. are very low (2.46x lesser than the mean of all connections generating relevant sequences). This result illustrates that the proposed reservoir model learns stably to form and retrieve relevant sequences.

To supplement the above result of reduction in chaotic states, we plot the trajectories (i.e. firing rates of neurons as time evolves) of 5 excitatory neurons in the 400-neuron reservoir that encode the characters (C, B, P, T) across 5 test trails as shown in Fig.8.7 (b). It is clearly seen that the activity of each neuron prediciting a particular word follows a stable trajectory with slight variation across different trials for different sequences. When the network is presented with any input (say P corresponding to Unit 5), initially, the trajectories vary across different trials until they converge around time t= 0 ms, where the highest spiking activity is observed for the given character in that particular neuron. Based on the second highest activity, when the next character is presented to the network, the neuronal activity of the neuron across different trials varies as expected (due to the randomness in the input distribution as well as the chaotic activity). However, the trajectories tend to converge toward one another with time implying more correlated activity. In fact, sequences that have less commonality with other words ('PET', 'BIRD') have more synchronized trajectories. Thus, we can infer that the contextual dependencies developed with the modification of E→E connections has a stabilizing effect on the chaotic states of the reservoir. While the existence of chaotic states help a neuron differentiate between sequences that have more similarities (such as CAT, COT), it does not

have an overwhelming effect disrupting its inference capability thereby enabling our model to operate at the *edge of chaos*.

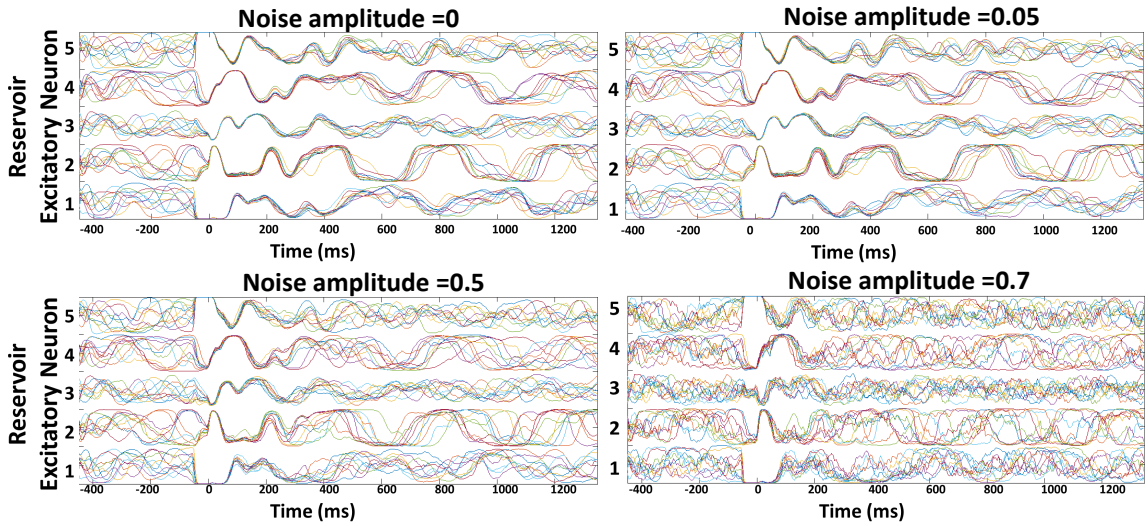


Fig. 6.5. Robustness of the reservoir spiking model learnt with combined Hebbian/non-Hebbian plasticity against noise. Trajectories shown for 5 different excitatory neuron of the 400-neuron reservoir model for varying noise activity across 10 different test trials.

Suitably learnt reservoir models generally operate with co-existing chaotic states and stable trajectories as shown above. However, external noise can induce more chaos in the reservoir that will overwhelm the stable patterns of activity due to the inevitable feedback loops among recurrent connections. Here, we discuss the susceptibility of our reservoir model (and hence our learning scheme) to external noise. To conduct the noise analysis, we introduced random Gaussian noise along with the standard Poisson inputs to the 400-neuron reservoir model during testing. The injection of noise alters the net post-synaptic current received by the reservoir neurons ($I_{post} = \Sigma_i[W_i Input_i + N_0 randn(i)]$, where N_0 is defined as the noise amplitude) thereby affecting the overall firing rate (or trajectory). In order to prevent the variation in trajectory caused due to the randomness in the Poisson inputs, we fixed the distribution for a given input/chararcter across different trials. Fig.6.5 shows the trajectories of 5 different

recurrent units for varying levels of noise (with different N_0) for 10 different test trials. Since the input distribution is constant, the neuronal trajectory during the presentation of the first character in each trial (from t= 0 to t=350 ms) remains almost equivalent in all cases. As time evolves, we input the next character predicted by the reservoir based on the top-2 spiking activity that leads to a different trajectory for each neuron, representative of the chaotic states. As N_0 increases, we observe a steady increase in the variation of the trajectories which in turn degrades the capability of each neuron to predict correctly. However, only for noise levels with $N_0 \geq 0.7$, the reservoir yields diverse trajectories. For moderate noise (with $N_0 \leq 0.5$), our model exhibits high robustness with negligible degradation in prediction capability.

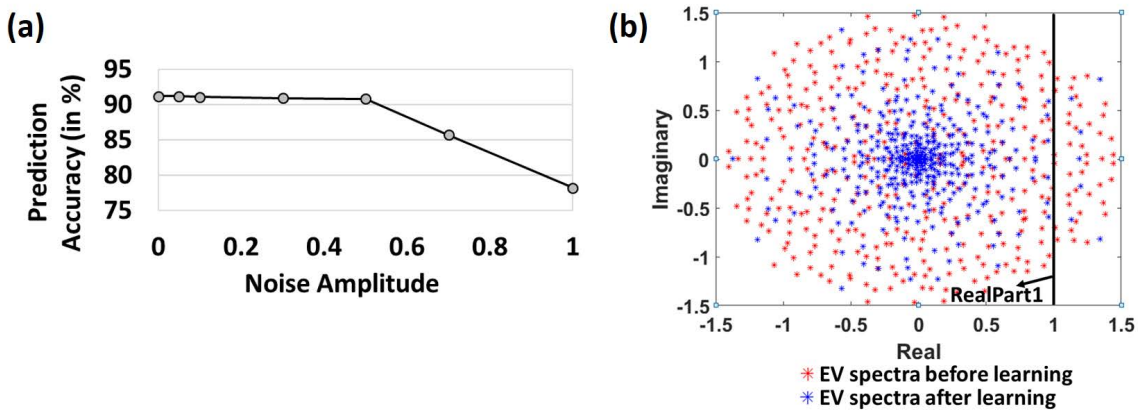


Fig. 6.6. (a) Variation of Prediction accuracy with noise amplitude (b) Evolution of spectrum of eigen values of the reservoir synaptic connections (includes $E \rightarrow E, E \rightarrow I, I \rightarrow E, I \rightarrow I$) in the complex plane before and after learning

Fig. 6.6(a) also shows the prediction accuracy of the 400-neuron reservoir model for varying noise levels. For noise amplitude of 0.5, the accuracy observed is 90.8% (0.4% lower than the model without noise). Our models insensitivity to adequate levels of noise further validates the efficacy of the combined Hebbian/non-Hebbian plasticity learning in reducing the chaotic states within the reservoir model. The reduction in chaotic dynamics during the training phase allows the network to look

back in history to formulate the predictions correctly even in presence of external noise. The accuracy levels, however, degrade steeply with increasing levels of noise (N_0 beyond 0.5) as the chaotic activity (due to the recurrent feedback loops) starts overwhelming the locally stable reservoir activity. For a noise amplitude of 1.0, the accuracy observed is 78.2%.

To further elucidate the effectiveness of Hebbian/non-Hebbian plasticity in reducing chaos, we probed into random matrix theory that gives a powerful understanding of the complex emergent behavior of large networks of neurons, such as the reservoir, with random recurrent connections. Diagonalization of the synaptic weight matrix of the reservoir connections yields equal number of modes as the number of neurons in the reservoir [135]. Each mode is a complex numbered eigenvalue, whose real part corresponds to the decay rate of an associated mode and imaginary part is proportional to the frequency of the pattern. Activation of any one of these complex modes results in a network that exhibits spontaneous oscillations at the corresponding frequency [117]. The activation of multiple such modes results in complex dynamics due to a superposition of individual frequencies in a highly nonlinear manner resulting in chaotic dynamics/persistent reservoir activity as observed earlier in Fig. 8.7(b), 6.5. We analyzed the EigenValue (EV) spectra of the synaptic weights of the reservoir before and after learning. The EV of all the connections (includes $E \rightarrow E, E \rightarrow I, I \rightarrow E, I \rightarrow I$) in the reservoir initially (drawn from a random distribution) are distributed uniformly in a circle in the complex plane in accordance with Girko's circle law [136]. If the real part of a complex eigenvalue exceeds 1, the activated mode leads to oscillatory behavior contributing to the overall chaotic dynamics of the reservoir. From Fig. 6.6 (b), we observe that before learning, the eigenvalues of the 400-neuron reservoir have a larger radius with more values > RealPart1 implying more activated modes characteristic of chaos [135]. However, as learning progresses, the EV spectral circle shrinks to a non-uniform distribution with a high density of values towards the center and few modes > RealPart1. The dense center EVs are fixed points that are non-chaotic and persistent. In fact, the changing shape of EV spectrum with training of $E \rightarrow E$,

computationally corresponds to learning. The gradual movement of EV spectra from chaotic to fixed points establishes the stabilizing effect of learning the E→E reservoir connections with our combined plasticity scheme.

6.4 Discussion

We presented an unsupervised recurrent spiking neural model learnt with different forms of plasticity for reliable generation of sequences even in presence of perturbations. We incorporated a non-Hebbian decay mechanism, inspired from the Heterosynaptic plasticity observed in mammalian brain, with standard STDP learning while evolving the recurrent connections within the reservoir to suppress the chaotic activity emerging from the attractor states. Our results indicate that the mutual action of the Hebbian/non-Hebbian plasticity enables the formation of locally stable trajectories in the reservoir that works in symphony with the reduced chaotic states to produce different sequences. While we demonstrate the efficacy of our model for fairly simple character-level prediction of visual words, we believe that the general functional properties of our combined plasticity learning can be extended to larger recurrent models for more complex spatio-temporal pattern recognition (such as action recognition, video analysis etc.). Investigation of the learning rule on other recurrent architectures is a promising direction of future research. However, large-scale networks, with larger number of recurrent connections, are more vulnerable to chaotic dynamics, that would require more number of training examples for convergence. This will affect the training complexity of the reservoir. In such cases, the decay rate as well as the learning rate (in Hebbian phase learning) has to be varied suitably to avoid the formation of strong feedback dynamics while maintaining reasonable training time. Also, the inference ability of the reservoir is a strong function of the synaptic connecticity (or size of the reservoir). To avoid escalating the reservoir size for more complex problems, the learning rules can further be optimized. For instance, varying the learning rate as training progresses or a mechanism that adapts the decay rate

(in the non-Hebbian phase) depending upon the extent of attractor states formed, might further enhance the inference ability of a reservoir.

It is worth mentioning that there has been previous effort on combining different plasticity schemes with reservoir models. In [137], the authors combine STDP, synaptic scaling of $E \rightarrow E$ connections and homeostasis to engineer a plastic reservoir model that performs better than static networks on simple tasks. While our work complements [137], the inclusion of non-Hebbian decay gives an entirely new perspective toward learning in recurrent networks with reduced chaos and regular trajectories even in presence of noise. Furthermore, this work deals with more difficult visual character prediction while entailing a detailed analysis of the contribution of the Hebbian/non-Hebbian Plasticity scheme for stable memory states. Finally, we would like to mention the recent work [138] that deals with reducing the attractor states developed in a recurrent SNN (different from our model) by un-learning the strong feedback connections after training. In contrast, the combined plasticity scheme proposed in this chapter can serve as a general learning methodology that *cohesively reduces the synaptic weights of strong correlations during training* for more robust and noise-resilient recurrent SNN implementations.

6.5 Appendix

6.5.1 Neuron and Synapse Model

We use the standard Spiking Neural Network (SNN) dynamics to model the neuronal and synaptic changes in our reservoir spiking model. As shown in Fig. 6.7, a post-neuron receives the pre-synaptic spikes modulated by the synaptic strengths that results in a change in its membrane potential. Synapses are modeled by exponentially decaying conductance changes wherein the conductance increases by a synaptic weight, w, on arrival of a pre-synaptic spike. Otherwise, the conductance continues

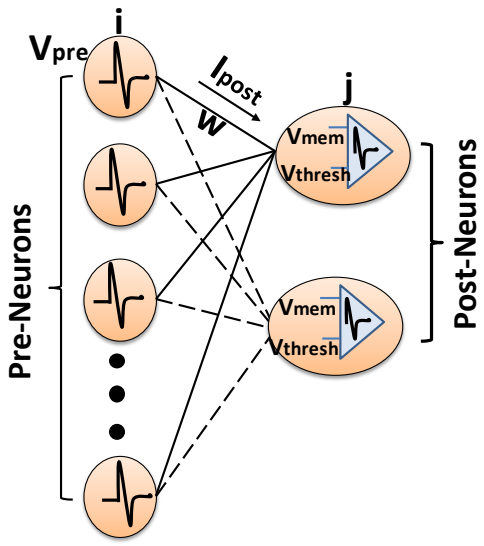


Fig. 6.7. A typical SNN architecture consisting of pre-neurons and post-neurons interconnected by synapses. The pre-synaptic voltage spike V_{pre} is modulated by the synaptic weight, w, to get the resultant post-synaptic current, I_{post}. The post-neuron integrates the current from each interconnected pre-neuron that causes its membrane potential, V_{mem}, to increase and spikes when the potential crosses a certain threshold, V_{thresh}.

to decay. The dynamics of both Inhibitory (Inh) and Excitatory (Exc) conductance for corresponding pre-synaptic neurons are

$$\tau_e \frac{dg_e}{dt} = -g_e \;\; \tau_i \frac{dg_i}{dt} = -g_i \tag{6.4}$$

where $\tau_e(2ms), \tau_i(4ms)$ are the time constants for Excitatory and Inhibitory post-synaptic potentials.

We use the Leaky-Integrate-and-Fire (LIF) model illustrated in Fig. 6.8 to simulate the membrane potential V_{mem} of a neuron as

$$\tau_{mem} \frac{dV_{mem}}{dt} = (V_{rest} - V_{mem}) + g_e(V_{exc} - V) + g_i(V_{inh} - V) \tag{6.5}$$

where V_{rest} is the resting membrane potential (-65 mV/-60 mV for Exc/Inh neurons, respectively), V_{exc} (0 mV) and V_{inh} (-100 mV) the equilibrium potentials of Exc and Inh synapses, τ is the leak time constant (60 ms/ 10 ms for Exc/Inh neurons). The LIF model causes V_{mem} to increase or decrease when pre-synaptic spikes are received and to otherwise decay exponentially. The post-neuron fires when V_{mem} crosses

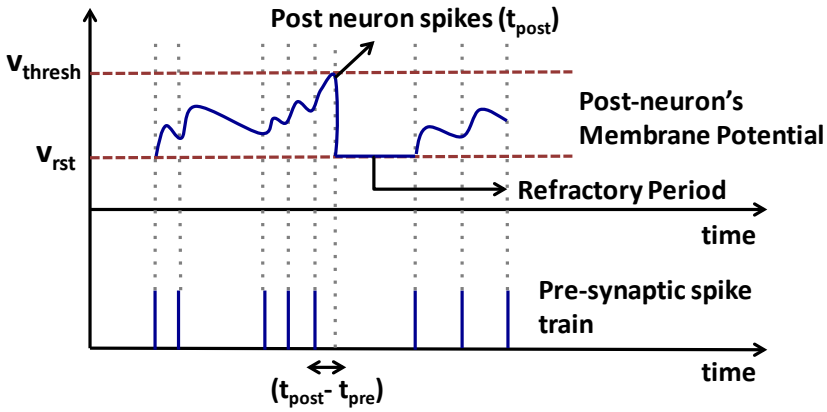


Fig. 6.8. The Leaky-Integrate-and-Fire dynamics of the membrane potential of a post-neuron that increases upon the arrival of pre-synaptic spike and decays subsequently. The post-neuron fires when the potential exceeds the threshold V_{thresh}. The potential is then reset to V_{rst} and a refractory period ensues during which the neuron is prohibited from firing. The relative timing of the post-neuron and pre-neuron spikes ($t_{post} - t_{pre}$) determines the synaptic potentiation.

the membrane threshold v_{thresh} (-52 mV/ -40 mV for Exc/Inh neurons, respectively) and then its membrane potential is reset to v_{rst} (-65 mV/ -45 mV for Exc/Inh neurons). After each firing event, a refractory period (5 ms/ 2 ms for Exc/Inh neurons) ensues during which the post-neuron is inhibited from firing even if additional input spikes arrive. Please note that both inhibitory and excitatory neurons in our proposed reservoir architecture follow the above LIF dynamics with different parameters.

6.5.2 Input Encoding

The input images of characters from the Char74K dataset (that comprise the visual words of the dictionary) are presented to the reservoir model as Poisson based spike trains, with firing rates proportional to the pixel intensity value. Specifically, each pixel intensity is divided by 6 resulting in firing rates between 0 and 42.5 Hz for standard grayscale valued pixels in the 0-255 range. As mentioned in the main manuscript, each input is presented to the reservoir for 350 ms. If the excitatory neurons in the reservoir, for a given input image, fire less than 5 spikes within the

350ms presentation time, the input firing rate is then increased by 20 Hz and the image is presented to the network again for 350 ms. This procedure is repeated until the reservoir outputs at least 5 spikes for the given input.

6.5.3 Training, Assignment and Inference

The plastic synapses connecting the input to the reservoir excitatory (In→Exc) neurons and the E→E connections within the reservoir are trained using the combined Hebbian/non-Hebbian Plasticity mechanism as mentioned in the manuscript. By the end of the training phase, the In→Exc connections learn to encode the generalized representations of each of the characters in the visual word/sequence while the E→E connections learn the correlation between each character. After the training, we set the learning rate for all plastic connections to zero and fix the adaptive membrane threshold of each excitatory neuron in the reservoir (as obtained from homeostasis). Then, the training set is presented once again to the reservoir model. Now, each excitatory neuron is assigned a particular class (or character) for which it spiked the most during this presentation. The assignment phase is the only time when training labels are used. Otherwise, the training of synaptic connections is completely unsupervised (i.e. no use of class labels).

During testing, when the reservoir is presented with a test image of an individual character from the visual word sequence, the test input is predicted to belong to the class represented by the group of assigned neurons with the highest average spiking rate. In addition, as described in the manuscript, we also monitor the second highest spiking activity among the reservoir neurons to obtain the next character of the sequence. The next character predicted is then presented to the reservoir and this process is repeated until the entire sequence has been generated. We use the value of the difference between the top-2 spiking activity to gauge whether or not sequence generation is done, as described in Section 6.3 above.

7. DISCRETIZATION BASED SOLUTIONS FOR SECURE MACHINE LEARNING AGAINST ADVERSARIAL ATTACKS

7.1 Introduction

Deep Learning Networks (DLNs) have exhibited better than human performance in several vision-related tasks [4]. However, they have been recently shown to be vulnerable toward adversarial attacks [10,27,139]: slight changes of input pixel intensities can fool a DLN to misclassify an input with high confidence (Fig. 7.1). What is more worrying is that such small changes (that craft adversaries) are visually imperceptible to humans, yet, mislead a DLN. This vulnerability severely limits the potential safe-use and deployment of DLNs in real-world scenarios. For instance, an attacker may fool a DLN deployed on a self-driving car to mispredict a STOP sign as a GO signal, and cause fatal accidents.

Subsequently, there have been several theories pertaining to the adversarial susceptibility of DLNs [139]. The most common one suggests that the presence of adversary is an outcome of the excessive linearity of a DLN (a property of high-dimensional dot-products). While one can argue that rectified linear unit (ReLU) activation imposes non-linearity in a model, the linear operations such as Convolution, Pooling exceed the number of non-linear ReLU operations. Further, ReLU is typically a linear functionality in the > 0 regime, and hence, plagues a DLN to be sufficiently linear. Now, this linearity causes a model to extrapolate its behavior for points in the hyper-space (of data and model parameters) that lie outside the training/test data manifold. Adversarial inputs, essentially, are images that are synthesized such that they lie far from the typical data manifold and hence get misclassified.

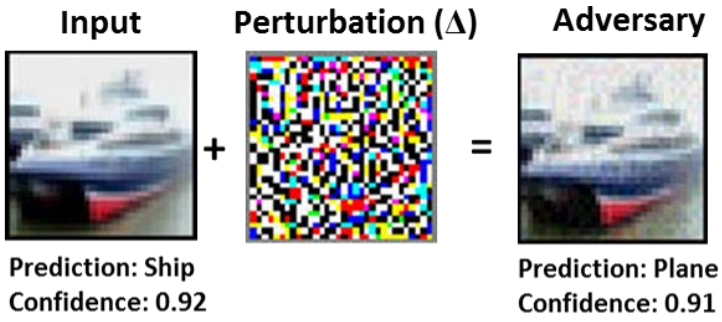


Fig. 7.1. An image of a **ship** perturbed with adversarial noise yields an adversarial image that fools the classifier. The classifier predicts the original image correctly with a confidence of 92%, while gets fooled by the adversarial image mispredicting it as **plane** with a high confidence of 91%.

Fig. 7.2 (a) demonstrates this data manifold intuition and adversarial input creation with a cartoon. Since DLNs are discriminative models, they partition a very high-dimensional input space into different classes by learning appropriate decision boundaries. The class-specific decision boundaries simply divide the space into hyper-volumes. Interestingly, these hyper-volumes encompass the training data examples as well as large areas of unpopulated space that is arbitrary and untrained. This extrapolation of decision boundary beyond the training data space is a result of *linearity*, that in turn, gives rise to generalization ability. The fact that a model trained only on training data is able to predict well on unseen test data (termed as, generalization) is a favorable outcome of this extrapolation. Unfortunately, generalization also exposes a model to adversarial attacks. Adversarial data are created by simply adding small perturbations to an input data point, that shifts it from its manifold (or hyper-volume) to a different hyper-volume (that the model has not been trained upon and shows extrapolated behavior), causing misclassification.

From the above intuition, one can deduce that adding regularization features to a DLNs training will improve adversarial robustness. In fact, the most effective form of adversarial defense so far is training a model with adversarial data augmentation (called adversarial training) [6]. It is evident that explicit training on adversarial data will increase the models capability to generalize and hence, predict correctly on unseen

adversarial data. However, the above discussion on excessive linearity and hyper-space dimensionality points to an alternate and unexplored regularization possibility, that is discretizing or constraining the data manifold for achieving adversarial robustness. For instance, discretizing the input data (say from 256-pixel value levels (or 8-bit) to 4 levels (or 2-bit)) reduces the regions into which data can be perturbed. In other words, the minimum perturbation required to shift a particular data point from one hyper-volume to another will increase in a discretized space (Fig. 7.2 (b)). This in turn will intrinsically improve the resistance of a DLN. Similarly, discretizing the parameter space (as in binarized neural networks (BNNs) [140]) will introduce discontinuities and quantization in the manifold (that is non-linear by nature). This will further decrease the extent of hyper-volume space that is arbitrary/untrained and thus reduce adversarial susceptibility (Fig. 7.2(b)). It is evident that such discretization methods have an added advantage of computational efficiency. In fact, low-precision neural networks (BNNs and related XNOR-Nets [141]) were introduced with a key motif of obtaining reduced memory and power-consumption for hardware deployment of DLNs.

We demonstrate that discretization, besides offering obvious efficiency improvements, has far-reaching implication on a models adversarial resistance. We particularly emphasize on three different discretization themes and illustrate their suitability toward improving a DLNs adversarial robustness, as follows:

- *Discretization of input space:* We reduce the input dimensionality by quantizing the RGB pixel intensities into a variable range: $2^8 = 256$ to $2^2 = 4$. We show that for minimal loss in accuracy, the adversarial robustness of a model substantially improves ($< 1\%$ accuracy difference between clean test and adversarial test data), even, without any adversarial training. Furthermore, we show that combining adversarial training with 2-bit input discretization makes a model substantially more robust (than adversarial training with full 8-bit input precision) for large perturbation ranges.

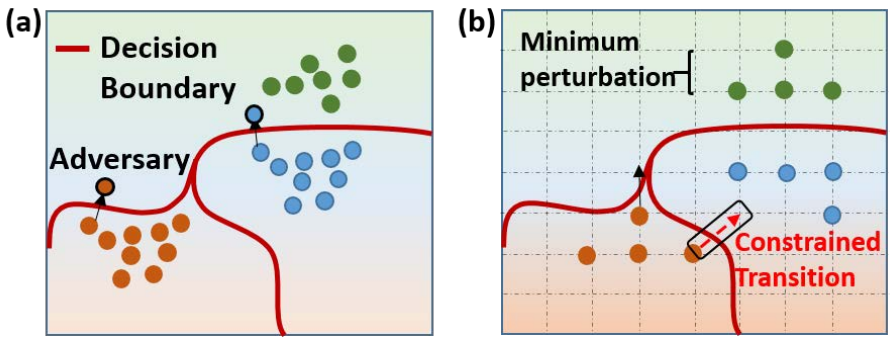


Fig. 7.2. Cartoon of the intuition behind adversary creation and discretization. (a) The data points (shown as 'dots') encompass the data manifold in the high-dimensional subspace. The classifier is trained to separate the data into different categories or hyper-volumes based on which the decision boundary is formed. Note, the decision boundary is a characteristic of the trained parameters (weights) of the model. The decision boundary is, however, extrapolated to vast regions of the high-dimensional subspace that are unpopulated and untrained because of **linear** model behavior. Adversaries are created by perturbing the data points into these empty regions or hyper-volumes and are thus misclassified (**orange** mispredicted as **green** in this case). (b) Discretization quantizes the data manifold thereby introducing a minimum perturbation required to shift a data point. As quantization will increase, so will the minimum allowed distortion. Further, discretization constrains the creation of adversaries since not all transitions can cause a data point to shift between hyper-volumes.

- *Discretization of parameter space:* We show that models trained with low-precision weights and activations, such as BNNs, are intrinsically more robust to adversarial perturbations than full precision networks. Furthermore, we find that training BNNs with adversarial data augmentation is difficult. However, increasing the capacity of the BNN (with more neurons and weights) minimizes the adversarial training difficulty. For sufficient model capacity, adversarially trained BNNs yield higher adversarial robustness than their full-precision counterpart.

- *Discretization of both input & parameter space:* We demonstrate that combining input discretization with binarized weight /activation training greatly improves

a model's robustness. In fact, training a BNN with input discretization (say, 2-bit input) yields similar or better adversarial accuracy as that of an adversarially trained full-precision model. Thus, the combined discretization scheme can be seen as an efficient alternative to achieving adversarial robustness without the expensive data augmentation procedure (note, only for single-step attacks).

7.2 Related Work

Based on the intuition demonstrated in Fig. 7.2, robustness of DLNs can be attributed to two factors: property of the input and model property. Consequently, there have been many recent proposals [142–146] that exploit the input-dependent factor and try to remove adversarial perturbations by applying input preprocessing or transformations. Due to the simplicity of this approach, these methods are attractive for practical implementations as they do not incur large computational overhead (as with adversarial training) and do not interfere with the learning process. Our work complements the results of the prior works while presenting a novel result on the effectiveness of combined parameter and input discretization for adversarial robustness.

One of the first works on input discretization by Xu et al. [145] proposes a *depth-color-squeezing* technique wherein they reduce the degrees of freedom available to an adversary by 'removing' unnecessary features. Our pixel discretization scheme is based on their color depth reduction technique. However, the key idea in [145] is to compare the model's prediction on the original input with its prediction on the squeezed input during testing. Xu et al. train the model with regular inputs and during inference use pixel discretization to detect adversarial inputs. That is, if the original and squeezed inputs produce predictions with large difference (greater than an user-defined threshold), the model deems the input to be adversarial and rejects it. Ultimately, the model outputs prediction for only legitimate or non-adversarial inputs. In contrast, the key novel aspect of our work is to train a model with discretized pixel data such that the model looks at a limited range of input values during training

that decreases or constrains its' ability to overly generalize in the high-dimensional subspace. Similarly, the thermometer encoding technique and input transformation technique proposed in [143], [142] are guided by the same intuition of limiting the range of adversarial perturbations by constraining the input. Guo et al. [142] trained the network with images transformed in various ways and observed improved adversarial resistance. However, they measured robustness for controlled gray-box attack settings (where, the model parameters are known to the attacker but the input transformations are unknown). Our results on white-box attack is a stronger notion of robustness as we assume all parameters as well as input discretization known to the attacker. Thus, our results while supporting the claims of [142] are more substantial and generalizable. In Buckman et al. [143], the authors propose a thermometer encoding technique to map input pixels to a binary vector in order to make more meaningful change during pixel discretization without losing any information from the original image. While, the authors show good adversarial robustness results on small tasks, such as, MNIST [147], they are shown to achieve poor performance on more complex datasets (like, CIFAR10 [148]) [144].

In order to address the limitation of the prior works solely based on input transformation, we investigate the effect of combining model discretization with input discretization thereby leveraging both criteria that contribute to adversarial dimensions. A recent work [149] demonstrated the effectiveness of BNNs against adversarial attacks and observed a similar difficulty in adversarial training with BNNs. However, they did not consider input space discretization and its impact on robustness. While complementing their results, we show that quantizing the input pixels of a BNN during training greatly improves its robustness, even waiving the need for the expensive and time-consuming adversarial training, for certain perturbation ranges. To summarize, the key contributions and novelty that discerns this work from above mentioned works are:

- In contrast to previous works that use input transformation during test phase only, our analysis shows that DLN models can resist single-step attacks across

different black-box, white-box and gray-box scenarios if the quantization measure is introduced during training.

- We show that combined input and parameter discretization during training is a major enabler towards imparting adversarial robustness. To the best of our knowledge, this work is the first to formally evaluate and analyze the impact of input and parameter space discretization for DLNs (across simple and complex datasets including CIFAR10, CIFAR100, ImageNet2012 dataset) on robustness.

- We provide an extensive comparison of our approach (combining input and parameter discretization) against prior works using only input discretization [145] or only parameter discretization [149] for multi-step iterative attack scenarios. The results (discussed in Section IV.F) establish combined scheme as a stronger defense in imparting intrinsic robustness to DLNs. However, such combined schemes while doing better than prior works [145, 149] still remain vulnerable in iterative attack scenarios. This is a significant aspect of our work as it exposes the vulnerability/limitation of discretization techniques.

7.3 Background on Adversarial Attacks

Generating Adversaries: Adversarial examples are created using a trained DLNs parameters and gradients. As shown in Fig. 7.1, the adversarial perturbation, Δ, is not just some random noise, but carefully designed to bias the networks prediction on a given input towards a wrong class. Goodfellow et. al [139] proposed a simple method called Fast Gradient Sign Method (FGSM) to craft adversarial examples by linearizing a trained models loss function ($\mathcal{L}$, say cross-entropy) with respect to the input (X):

$$X_{adv} = X + \epsilon \times sign(\nabla_X \mathcal{L}(\theta, X, y_{true})) \tag{7.1}$$

Here, y_{true} is the true class label for the input X, θ denotes the model parameters (weights, biases etc.) and ϵ quantifies the magnitude of distortion. The net pertur-

bation added to the input ($\Delta = \epsilon \times sign(\nabla_X \mathcal{L}(\theta, X, y_{true}))$) is, thus, regulated by ϵ. Distorting the input image in the direction of steepest gradient has the maximal effect on the loss function during prediction. Intuitively, referring to Fig. 7.2, this distortion shifts the data point from the trained region or hyper-volume to an arbitrary region thereby fooling the model.

Types of Attacks: There are two kinds of attacks: Black-Box (BB), White-Box (WB) that are used to study adversarial robustness [150]. WB adversaries are created using the *target* models parameters, that is, the attacker has full knowledge of a target models training information. BB attacks refer to the case when the attacker has no knowledge about the target models parameters. In this case, adversaries are created using a different *source* models parameters trained on the same classification task as the target model. Since BB attacks are transferred onto the target model, they are weaker than WB attacks. Security against WB attacks is a stronger notion and robustness against WB attacks guarantees robustness against BB for similar perturbation (ϵ) range.

Adversarial Training: Adversarial training simply injects adversarial examples into the training dataset of a model [6]. For each training sample in the dataset, an adversary is created using FGSM [139]. There are several forms of adversarial training. For instance, instead of using the same ϵ for all training examples, [151] and [150] propose to sample a unique ϵ (from a random normal distribution) for each training example. This increases the variation in the adversaries created, thereby, increasing the robustness of a network to larger range of ϵ values. The authors in [152] use WB adversaries created, using a multi-step variant of FGSM to guarantee a strong defense against both BB and WB attacks. Note, the common theme across all adversarial training methods is data augmentation.

In this work, in most experiments (Section IV.A - IV.D), we focus on adversarial attacks created using FGSM and evaluate the robustness of models against WB adversaries. We evaluate a model's robustness and report adversarial accuracy on the adversarial dataset created using the test data for a given task. We show additional

attack results considering BB/gray-box attack with adversaries created using FGSM (in Section IV.E). We also compare our analysis on input and parameter discretized network's robustness with prior works [144, 145, 149] utilizing either input or parameter discretization on multi-step projected gradient descent [152] based iterative WB attack scenarios in Section IV.F.

7.4 Experiments

We conduct a series of experiments for each discretization theme, primarily, using MNIST [147] (Fully Connected Network, FCN) and CIFAR10 [148] (AlexNet [153] architecture), detailing the advantages and limitation of each approach. We compare the adversarial robustness of each discretization approach with its full-precision counterpart (with and without adversarial training), using ϵ values reported in recent works [150, 152]. For adversarial training, we employ Random-step FGSM (R-FGSM) proposed in [150] to create a variety of training set adversaries. R-FGSM perturbs the input X with a small random step (sampled from a normal distribution $\mathcal{N}$) before adding the loss gradient to the input: $X_{adv} = X + \Delta$, where $X = X + \alpha sign(\mathcal{N}(0, I), \alpha = \epsilon/2$. We use WB adversarial training to confer strong robustness toward all forms of attacks. Note, for MNIST (CIFAR), we use $\epsilon = 0.3$ (8/255) during adversarial training. For evaluating the robustness of parameter space discretization, we use BNNs [140, 153] to evaluate CIFAR10 and MNIST datasets. We also evaluate the robustness of discretization methods on large-scale datasets, CIFAR100 (ResNet20 architecture [154]) and Imagenet [155] (AlexNet architecture) using XNOR networks [141, 156]. Please note, for MNIST we use two different FCN architectures: FCN1-4 hidden layer network with 6144 neurons each (784-6144($\times$4)-10), FCN2-4 hidden layer network with 600 neurons each (784-600($\times$4)-10). We imported github models [153, 156] for implementing our experiments. We used the same hyperparameters (such as weight decay value, learning rate etc.) as used in [153, 156] to train our models.

In all experiments, we impose the discretization constraints (input discretization or parameter discretization or both) during training as well as testing. To elucidate this, we describe the training and test methodology for each of the discretization themes below:

- *Input Discretization Training/Testing Method*: Here, we quantize the input both during training and testing. Say, for 2-bit input discretization (pixel intensities vary in [0, 4] range), we train a model with 2-bit discretized inputs. During testing, both adversarial and clean test inputs are quantized into 2-bit discretized forms and fed into the model to monitor the prediction accuracy. Note, for input discretization experiments in Section III.A, we use full-precision models (that is, parameters and activations of the DLN model have 32-bit floating point precision).

- *Parameter Discretization Training/Testing Method*: Here, we quantize the parameter/activation values of a DLN both during training and testing. Say, for BNN, we train a model with extremely low precision weights and activation values clamped to {+1, -1}. During testing, both adversarial and clean test inputs are fed into the binarized model to monitor the prediction accuracy. Note, for parameter discretization experiments in Section III.B, we use full-precision 8-bit inputs (that is, input pixel intensities range from [0, 255]).

- *Input and Parameter Discretization Training/Testing Method*: Here, we quantize the input as well as parameter/activation values of a DLN both during training and testing. Say, for 2-bit input discretization BNN, we train a model with {+1, -1} weight/activation values while showing 2-bit quantized inputs. During testing, both adversarial and clean test inputs are again quantized to 2-bit range and fed into the BNN to monitor the prediction accuracy. Note, for input and parameter discretization experiments in Section IV.C, we use BNN or XNOR models with 2-bit/4-bit/8-bit input precision.

It is worth mentioning that this is the only other work besides [149] demonstrating the effectiveness of discretized/binarized parameter space on adversarial attacks. While [149] conducted experiments with various forms of attacks (primarily, on MNIST), we restrict ourselves to the WB attack scenario and extrapolate our analysis on larger datasets (CIFAR10, CIFAR100, Imagenet).

7.4.1 Discretization of Input Space

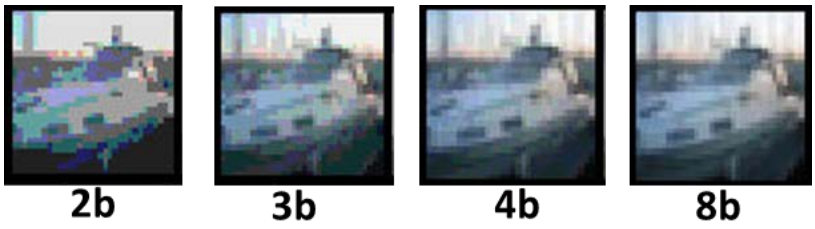


Fig. 7.3. Sample images from CIFAR10 dataset for varying levels of input pixel discretization: $2-bit, 3-bit, 4-bit, 8-bit$

With input space discretization, we convert raw integer pixel intensities ($0 \leq I \leq 255$) that are typically 8-bit (or $8b$) values to a low precision range of δ bits ($0 \leq I_\delta \leq 2^\delta$) as:

$$I_\delta = \left\lfloor \frac{I}{(256/2^\delta)} \right\rfloor (\frac{256}{2^\delta}) + \frac{1}{2}(\frac{256}{2^\delta}) \tag{7.2}$$

where $\lfloor\rfloor$ denotes integer division. Such quantization reduces the number of data points (given a grayscale input image of size $N \times N$) in the manifold from $2^{8^{N^2}}$ to $2^{\delta^{N^2}}$. This can be broadly interpreted as constraining the data space (that can also be viewed as limiting the range of values across different input dimensions). In accordance with our intuition in Fig. 7.2, constrained data space can possibly limit the creation of adversaries as well. Fig. 7.3 illustrates sample CIFAR10 images discretized to varying δ values. The corresponding accuracy (trained on AlexNet for 20 epochs) is shown in Table 7.1. There is a natural tradeoff between input discretization and overall accuracy of a network. Yet, the test accuracy loss from the full precision $8b$

to $3b$ is $\sim 2.2\%$. This verifies the presence of unnecessary input features that do not substantially contribute to the classification task or accuracy. This result has also been noted by Xu et al. in [145]. $2b$ discretization decreases the accuracy by a larger margin ($\sim 6\%$). Note, this accuracy loss can be minimized by training the $2b$ inputs for more epochs. However, for iso-comparison, we fix the number of epochs across all experiments for a given dataset.

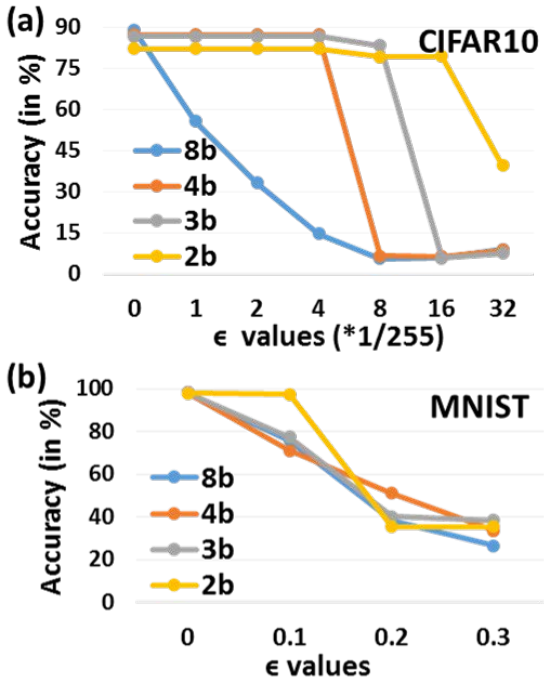


Fig. 7.4. Adversarial accuracy on test data for varying perturbation values on (a) CIFAR10 (b) MNIST for different input discretization

A remarkable outcome of this discretization method is the substantial improvement in a models adversarial accuracy. Fig. 7.4 illustrates the evolution of adversarial accuracy of the CIFAR10 models (from Table 7.1) with increasing level of perturba-

Table 7.1.
CIFAR10 Accuracy

Input-bit	Accuracy
2b	82
3b	86.64
4b	87.1
8b	88.9

tion, ϵ. Here, the discretized adversarial inputs are created using FGSM on discretized clean inputs as follows:

$$X_{adv} = X_{\delta} + \epsilon \times sign(\nabla_{X_{\delta}} \mathcal{L}(\theta, X_{\delta}, y_{true})) \tag{7.3}$$

$$X_{adv} \rightarrow X_{adv_{\delta}} \tag{7.4}$$

For a given discretization constraint δ (say, $2b$), $2b$ input (X_{δ}) is passed to a model. Then, the corresponding loss $\mathcal{L}$ is linearized and multiplied with perturbation value ϵ which is then added to the $2b$ input to create the adversary X_{adv}. We quantize X_{adv} to create the $2b$ discretized adversary $X_{adv_{\delta}}$. It is evident that quantization after creation of X_{adv} reduces the effect of perturbation ϵ. Note, however, the same approach is also used in prior works [142, 145] to create adversaries with quantized inputs. The claim is that quantization can be perceived as an input preprocessing step which, serves as a deterrent to the attacker by reducing the intensity of perturbation and, hence, improve adversarial robustness.

In Fig. 7.4, $\epsilon = 0$ corresponds to clean test accuracy. It is clear that clamping the input precision to lower values increases the resistance of the model to larger magnitude of distortion. We speculate that constraining the input precision or allowed range of input values reduces the overall hyper-volume space thereby leaving less space for shifting or adversarially perturbing a data point (referring to Fig. 7.2 (b) intuition). $8b$ input model shows a decline in accuracy even for a small value of $\epsilon = 1/255$. Thus, we can deduce that higher input precision (which implies larger range of input values for a data point) allows even small perturbations to shift the data

point (as seen earlier in Fig. 7.2 (a)). In contrast, increasing discretization increases the minimum ϵ that affects a model's accuracy catastrophically. What is surprising is that for $2b$ input, a models adversarial accuracy ($\sim 79\%$) for large $\epsilon = (8, 16)/255$ is almost similar to that of clean accuracy ($\sim 82\%$). For larger $\epsilon = 32/255$, the accuracy of all models declines to $< 10\%$, except $2b$. This is a very interesting result since we have not employed any adversarial training, and still achieve substantial adversarial resistance for a large ϵ range.

Fig. 7.4 (b) shows the adversarial accuracy results for MNIST (trained on FCN2 for 10 epochs). We observe a similar trend of increasing adversarial resistance with increasing discretization for larger ϵ. Since MNIST is a simple dataset with predominantly black-background, input discretization $(8b, 4b, 3b)$ does not contribute much to adversarial resistance until we go to extremely low $2b$ precision. In fact, $2b$ discretization yields adversarial accuracy similar to the clean test accuracy (for $\epsilon = 0.1$) exhibiting the effectiveness of this technique even for simple datasets. Note, in all experiments, we imposed input discretization constraints both during training and testing as mentioned earlier.

Next, we trained the $2b$ input discretized CIFAR10 and MNIST models with adversarial training to observe the improvement in adversarial accuracy compared to $8b$ input adversarial training (Table 7.2). Note, the adversaries were discretized to $2b$ precision (using Eqn. 3, 4) to adversarially train the $2b$-input CIFAR10, MNIST models. Compared to the results in Fig. 7.4 (a, b), adversarial training substantially improves the robustness of a model with full $8b$ input for larger ϵ values. Input discretization greatly furthers this robustness with $> 10\%$ accuracy gain across different perturbation ranges in both MNIST and CIFAR10. It is worth mentioning that the CIFAR10 accuracy (79%) without adversarial training for $\epsilon = (8, 16)/255$ for $2b$ input is as good as the accuracy (83%) with adversarial training. This shows that input discretization is a good regularization scheme that improves the generalization capability of a network on adversarial data. Note, for $\epsilon = 32/255$ in case of CIFAR10, the accuracy is similar for $8b, 2b$ since the adversarial training was conducted with

adversaries created using $\epsilon = 8/255$. Including larger perturbation adversaries during adversarial training will yield improved accuracy gain.

Table 7.2.
Accuracy with adversarial training for varying ϵ. Text in red are the ϵ values for MNIST and corresponding accuracy.

Data	Model	Clean	ϵ:8/255 0.1	ϵ:16/255 0.2	ϵ:32/255 0.3
CIFAR10	2b	83.1	82.7	82.7	43.9
	8b	84.3	62.2	53.6	45.5
MNIST	2b	98.5	98.5	84.7	85.4
	8b	98	84.8	74.5	65.9

7.4.2 Discretization of Parameter Space

Since input discretization gave us such promising results, we were naturally inclined toward analyzing a binarized neural networks (BNN) behavior against adversarial attacks. Here, the weights and activations (or parameters) are discretized to extremely low precision values $\{+1, -1\}$ [140]. The discretization constraints are imposed on a BNN during training, wherein, the parameters are clamped to $\{+1, -1\}$ after every backpropagation step. One can view this discretization as an implicit form of regularization. In fact, it is this extreme form of regularization that makes a BNN difficult to train (clean test accuracy observed with BNNs is, typically, lower than full-precision networks). As suggested in [149], the difficulty in training a BNN translates to difficulty in attacking the BNN as well. Referring to the data manifold intuition (Fig. 7.2), we can deduce that constraining the parameter space during a models training will introduce discontinuities and non-smoothness in its decision boundary. Since adversaries are created using gradients of a model (that is a property

of the models decision boundary), generating gradients (and hence adversaries) for non-smooth functions will be difficult. This in turn will make a BNN less susceptible to adversaries. Note, the input image to a BNN is full $8b$ precision.

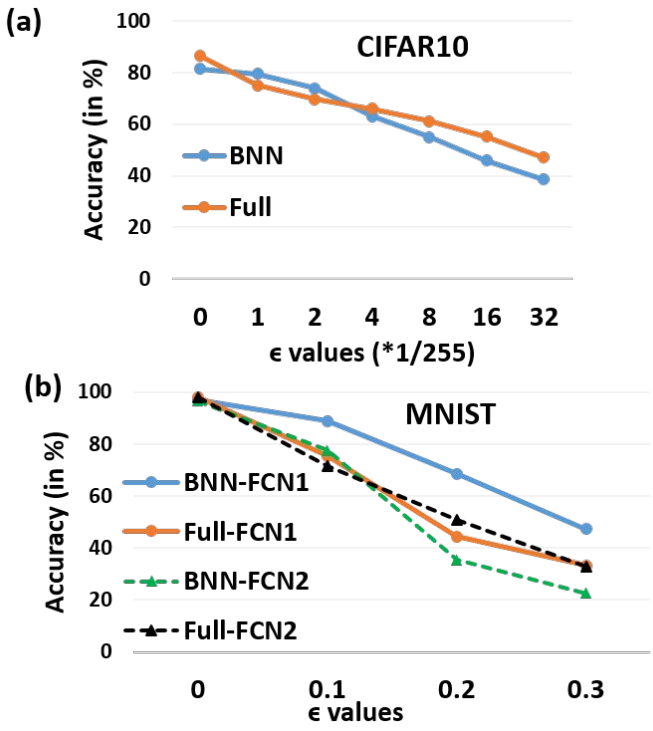


Fig. 7.5. Adversarial accuracy on test data for varying perturbation values on (a) CIFAR10 (b) MNIST for binarized and full-precision ($32b$ weights) models

Fig. 7.5 (a) compares the adversarial accuracy obtained for varying ϵ values for CIFAR10 BNN (AlexNet architecture) against a similar architecture full-precision network (with $32b$ precision for weights and activations). We trained the networks for 40 epochs since BNNs require more training iterations to attain comparable accuracy as that of a full-precision network. Here, we do not incorporate input discretization in our analyses. All networks are fed full-precision $8b$ inputs (both during training and testing). In Fig. 7.5(a), for $\epsilon \leq 2/255$, BNN shows better adversarial resistance (i.e. adversarial accuracy is closer to clean accuracy). However, the BNNs accuracy

declines steeply as we move toward larger perturbation ranges. We note a similar trend for MNIST (trained for 10 epochs on FCN2 architecture), wherein the full-precision network yields improved robustness than the BNN for $\epsilon \geq 0.2$. These results contradict our intuition that increased discretization of BNNs should result in lesser adversarial susceptibility.

To understand this, we calculated the L1 norm of the first hidden layer activation of the FCN2 network in response to clean input images. We found that BNNs generally have a larger variance and range of values than full-precision network. Since BNN uses weight values (+/-1) which are typically of greater magnitude than the small weight values of a full precision network, we observe a larger range in the former case. Interestingly, we find that the L1 norm of the BNN activations (in response to adversarial images perturbed with lower ϵ values) approximately lie within the same range as that of the clean input case. In contrast, L1 norm for higher ϵ adversaries have a much higher range. For a full-precision network, the L1 norm range of the different ϵ adversaries and clean data typically intersect with each other owing to the lower weight values (Fig. 7.6). We believe that the extreme quantization of weight values in BNNs to higher magnitudes causes adversarial susceptibility for larger range perturbations. While the L1 norm analysis is not very substantial from a mathematical standpoint, it hinted us to increase the capacity (more neurons and weights) of the network. The motif here is that increasing the capacity would increase the overall range of activation values that might incorporate larger range perturbations. Exploding the network capacity for MNIST (FCN1 architecture) yielded a sizable improvement in adversarial resistance with BNN as compared to its corresponding full-precision counterpart (Fig. 7.5(b)). This is a crucial detail of our analysis that: *while BNNs are intrinsically robust to adversaries (for small ϵ), only models with sufficient capacity can withstand against large ϵ values.*

Even with adversarial training, we observed the same trend that binarized networks of insufficient capacity do not yield as good adversarial robustness as that of a full-precision network (Table 7.3). For CIFAR10, full-precision network is the clear

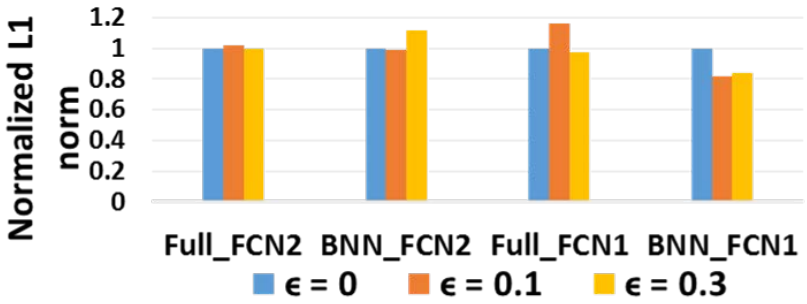


Fig. 7.6. Normalized L1 norm of first hidden layer activations in response to clean ($\epsilon = 0$) and adversarial inputs ($\epsilon = 0.1, 0.3$) for binarized and full-precision MNIST model of different architecture: FCN1, FCN2

Table 7.3.
Accuracy with adversarial training for varying ϵ. Text in red are the ϵ values for MNIST and corresponding accuracy.

Data	Model	Clean	ϵ:8/255 0.1	ϵ:16/255 0.2	ϵ:32/255 0.3
CIFAR10	BNN	79.7	53.1	43.6	35.3
	Full	82.7	72.2	63.6	55.5
MNIST	BNN	96.9	89.1	74.5	65.8
(FCN1)	Full	98	84.8	71	61.7

winner. While for MNIST (with excessive capacity FCN1 architecture), BNN yields improved robustness. A noteworthy observation here is that adversarial training substantially improves the robustness of a full-precision network (see CIFAR10 results in Fig. 7.5(a) , Table 7.3), while BNNs do not benefit much from them. In fact, we find that BNNs are difficult to train with adversarial training. The learning rate/other hyperparameters need to be tuned carefully to ensure that the BNN model converges to lower error values during adversarial training. [149] also observed a similar trend and explained that binarized weights are not as *malleable* as full-precision weights and hence cannot easily adjust to all possible variations of adversarial data augmented to

the training dataset. We think that increasing the capacity of the network compensates for the *'non-malleability'* of the constrained parameters to certain extent. As a result, we see improved accuracy for MNIST in Table 7.3 with FCN1 architecture.

7.4.3 Discretization of Input and Parameter Space

Next, we combined both discretization strategies and analyzed the adversarial robustness of BNNs with varying image-level discretization. We compare the adversarial accuracy of BNNs to that of a full-precision network for iso-input discretization scenarios, as shown in Table 7.4 for CIFAR10 (AlexNet architecture trained for 40 epochs). In Table 7.4, *BNN-2b* (*Full-2b*) refers to a binarized (full-precision) model with $2b$ input precision. Note, we use Eqn. 3, 4 to create discretized adversarial inputs for a given input precision correspnding to each experiments. That is, the adversarial accuracy for $BNN-2b$ or $Full-2b$ in Table 7.4 corresponds to testing with $2b$ discretized adversarial data. Full precision models have $32b$ precision weights and activations. While input discretization for a full-precision network suffers a sizeable accuracy loss, BNNs accuracy fluctuation is marginal with a maximum of 1% change. This is expected since BNNs (owing to +/-1 binarized parameters) do not have as many dimensions (as a full-precision network with $32b$ weights and activations) to fit the extra information in the $8b$ input data. Thus, BNNs fit $2b$, $8b$ data likewise yielding similar generalization error. As opposed to the results seen earlier with $8b$ inputs, BNNs with lower input precision $(2b, 4b)$ have significantly higher adversarial resistance than their full-precision counterparts even for large ϵ values. *Model capacity does not restrict the adversarial resistance in this case. This is an artefact of the two-step quantization that increases the minimum allowable perturbation to shift a data point.* We can also draw an alternate insight from this result: The constrained parameter space of BNNs restricts their overall exploration of the data manifold during training. Referring to Fig. 7.2 (b), this increases the probability of untrained or arbitrary hyper-volumes (for BNNs) thereby increasing their adversarial suscepti-

bility. Increasing the capacity enables a BNN to explore the manifold better during training. By discretizing the input, we are restricting the overall data manifold space. This allows a model, even, with lower capacity to explore the manifold well thereby decreasing the extent of arbitrary hyper-volumes. Table 7.5 illustrates the accuracy results for MNIST (FCN1 architecture trained for 20 epochs).

Table 7.4.
Adversarial accuracy with CIFAR10 for varying ϵ with different combinations of input and parameter discretization.

Model	Clean	ϵ:8/255	ϵ:16/255	ϵ:32/255
BNN-2b	81	80	80	36.7
Full-2b	82	79.1	79.3	39.6
BNN-4b	81.9	58.3	52.3	36.8
Full-4b	81.1	53.8	45.1	37.3
BNN-8b	81.5	55.1	45.9	38.6
Full-8b	86.5	61.1	55.2	47.1

Table 7.5.
Adversarial accuracy with MNIST for varying ϵ with different combinations of input and parameter discretization.

Model	Clean	ϵ:0.1	ϵ:0.2	ϵ:0.3
BNN-2b	96.4	96.4	60.7	62.3
Full-2b	97.8	97.4	35.4	35.3
BNN-4b	96.4	88.9	76.7	58.7
Full-4b	98.1	71.1	50.9	33.7
BNN-8b	97.1	89.4	56.1	33.6
Full-8b	98.2	75.9	38.5	26.4

We conducted adversarial training with $2b$ input discretized BNNs to find out if it helps build adversarial robustness. Note, adversarial training was performed with $2b$ precision adversaries (created using Eqn. 3, 4). The results are shown in Table 7.6. Comparing to the $8b$ input BNN adversarial training results in Table 7.3, we observe a substantial gain in adversarial accuracy. However, contrasting the BNN results against Table 7.2 ($2b$ input full-precision networks), we observe similar performance gains. In fact, the accuracy gains for $2b$ input CIFAR10 BNN with and without adversarial training (Table7.4/Table 7.6) are nearly the same. Earlier, we saw that the accuracy (for low ϵ values) of a full-precision network working on $2b$ input data without adversarial training is similar to that of an adversarially trained network on $8b$ inputs (Table 7.2, Fig. 7.4). Combining the adversarial training results till now, we can deduce the following: *1) For low input-precision (2b) regime, adversarial training does not compound the adversarial resistance of a network (irrespective of binarized or full-precision parameters), for lower ϵ values. Adversarial training helps when the input has higher (8b) precision. 2) Input discretization, in general, offers very strong adversarial defense for lower ϵ values. Discretizing the input as well as the parameter space furthers adversarial robustness. Adversarial training in a discretized input and parameter space does not benefit much and hence can be waived only for single-step FGSM attacks.* However, in case of stronger multi-step attack scenarios and to gain robustness against larger perturbations (such as $\epsilon = 32/255, 0.3$ in CIFAR10, MNIST), the network needs to be adversarial trained with corresponding large ϵ values.

7.4.4 Analysis on CIFAR100 and Imagenet

Scaling up the discretization analysis to larger datasets yielded similar results as observed with CIFAR10, MNIST. Fig. 7.7 demonstrates the adversarial accuracy evolution for CIFAR100 (trained on ResNet20 architecture for 164 epochs) for binarized XNOR ($1b$ weights and activations) networks and corresponding full-precision

Table 7.6.
Accuracy with adversarial training for varying ϵ. Text in red are the ϵ values for MNIST and corresponding accuracy.

Data	Model	Clean	ϵ:8/255 0.1	ϵ:16/255 0.2	ϵ:32/255 0.3
CIFAR10	BNN-2b	78.4	78.1	78.1	30.5
MNIST (FCN1)	BNN-2b	95.7	95.7	89.3	88.6

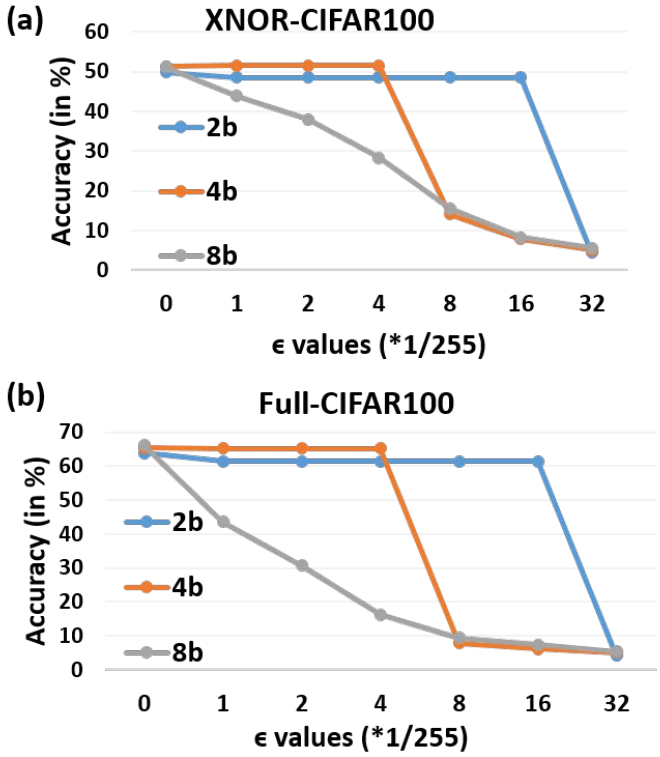


Fig. 7.7. Adversarial accuracy on test data for increasing ϵ values on binarized and full-precision ($32b$ weights) models trained on CIFAR100 with different input discretization:$8b, 4b, 2b$

($32b$ weights and activations) models. Note, XNOR networks are similar to BNNs (1-bit weights/activations) with additional scaling factors to achieve higher accuracy

on complex datasets. It is evident that input discretization is the most beneficial to obtain adversarial robustness. $2b$ input discretized models in both cases yield adversarial accuracy close to the clean accuracy ($\epsilon = 0$) for a large range of perturbations. The accuracy loss between clean and $\epsilon = 16/255$ adversary for $2b$-input XNOR (1.6%) is slightly better than the $2b$-input full-precision model (2.4%). This can be attributed to the intrinsic robustness offered by discretizing the parameter space of XNOR networks. Furthermore, the fact that $8b$-input XNOR yields higher adversarial accuracy for iso-perturbation values than $8b$-input full-precision model further demonstrates the ability of binarized networks to counter adversarial attacks. A noteworthy observation here is that the loss in clean accuracy between $2b$ and $8b$ input discretized full-precision network is small ($\sim$ 2%) as compared to the large 6% loss observed earlier with CIFAR10 (Table 7.1). As we scale up the complexity of the dataset (and the complexity of the DLN architecture), the features in the input/parameter space increase (that can be viewed as increase in the range of input/parameter values). Discretizing the input/parameters for a complex dataset on a larger DLN architecture (such as ResNet20) possibly eliminates certain range of input/parameter values that do not necessarily contribute to the accuracy. In contrast, smaller datasets or smaller DLN architectures have lesser range of input/parameter values and are thus at a risk of suffering a large accuracy drop with discretization.

Fig. 7.8 shows the accuracy results for Imagenet (trained on AlexNet). We only show the top-5 adversarial accuracy. We see similar trends as CIFAR100. Note, the XNOR models are trained for 50 epochs, while full-precision models are trained for 90 epochs. As a result, we see lower baseline accuracy ($\epsilon = 0$) in the former case. Like CIFAR100, the loss in clean test accuracy between $2b$ and $8b$ input discretization is minimal for each model. Also, the accuracy difference between clean and $\epsilon = 16/255$ adversarial data for $2b-XNOR$ (0.9%) is much lower than $2b-full$ precision models (2.8%). This highlights the intrinsic robustness capability of binarized networks even for large-scale datasets.

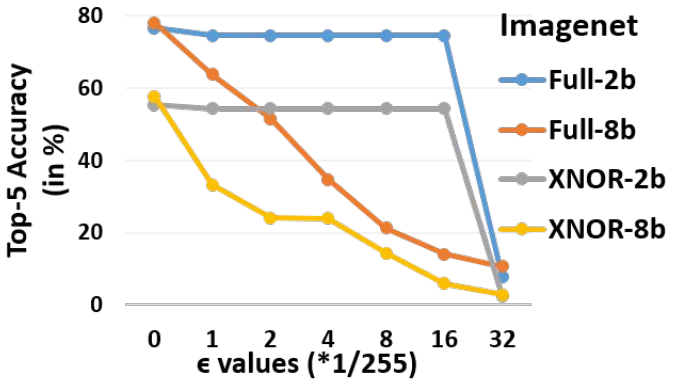


Fig. 7.8. Top-5 Adversarial accuracy on test data for increasing ϵ values on binarized and full-precision ($32b$ weights) models trained on Imagenet with different input discretization:$8b, 2b$

7.4.5 Other Attack Scenarios

Till now, we have focused on WB attack scenarios. For the sake of completeness, we analyzed the adversarial accuracy in gray-box attack scenarios for CIFAR100 dataset with XNOR/ full-precision models with $2b, 8b$ input discretization (Table 7.7). Here, the CIFAR100 attack model for an XNOR (or full-precision parameter) target is a separately trained XNOR (or full-precision parameter) model, respectively. Note, the attack model is a ResNet20 architecture with $2b, 8b$ input precision depending upon the input precision of the target. Essentially, we attack an XNOR-2b (or Full-2b) model with another XNOR-2b (Full-2b) model trained separately. The adversaries are created using FGSM. In this regime, even though the attack model is different from the target, the nature of input discretization is known to the attacker (hence, gray-box attack scenario). In Table 7.7, higher accuracy ($\sim>$ 25%) observed for large range of ϵ values in the $8b$ input regime establishes that gray-box attacks are weaker than WB. A noteworthy observation here is that gray-box attacks on $2b$ input discretized models (both full-precision and XNOR) seem to be stronger for $\epsilon = 4, 8/255$ than WB attacks (refer to Fig. 7.7). However, WB attacks (Fig. 7.7) are extensively more devastating in the higher $\epsilon > 8/255$ regime than gray-box. We conjecture that transferred attacks have more variability in the perturbations generated with lower ϵ

FGSM. As a result, a model (XNOR or full-precision) may not be able to counter such variability. While this variability remains in the higher ϵ range, the gray-box nature of the attack decreases the overall strength. Further investigation is required to analyze the nature of transferred attacks in a low ϵ regime on input-/parameter-discretized networks. This observation is a novel result as prior works on input transformation schemes (note, no parameter discretization has been explored in earlier works) have mostly focused on white-box or gray-box attacks for $\epsilon > 8/255$ regime and have thus never observed this phenomenon of higher vulnerability in gray-box than WB regime for low ϵ perturbations.

Table 7.7.
Gray-Box Adversarial accuracy with CIFAR100 for varying ϵ with different combinations of input and parameter discretization.

Model	Clean	ϵ:4/255	ϵ:8/255	ϵ:16/255	ϵ:32/255
XNOR-2b	49.9	44.6	39.6	30.7	18.7
Full-2b	63.9	59.4	54.2	43.8	24.5
XNOR-8b	51.1	42.9	35.6	25.4	16.3
Full-8b	66.2	56.9	48.3	34.1	19.63

To show the effectiveness of discretization against black-box attacks created with FGSM, we analysed the CIFAR10 dataset with BNN/ full-precision models with $2b, 4b$ input discretization. Here, the CIFAR10 attack model is a separately trained full-precision parameter model with $8b$ input discretization. It is evident from this attack/target model experimental setting that both input and parameter discretization regimes of the target model are unknown to the attacker (hence, BB attack scenario). Table 7.8 illustrates the results. Here, we observe higher adversarial accuracy (nearly equal to the clean accuracy) for a larger range of perturbations $\epsilon = 4, 8, 16/255$. For $\epsilon = 32/255$, the adversarial accuracy drops significantly. The higher range of accuracy in this case proves that BB attacks are the weakest among gray-box and WB attacks.

Table 7.8.
Black-Box Adversarial accuracy with CIFAR10 for varying ϵ with different combinations of input and parameter discretization.

Model	Clean	ϵ:4/255	ϵ:8/255	ϵ:16/255	ϵ:32/255
BNN-2b	81	79.8	79.8	79.8	56.1
Full-2b	82	81.8	81.8	81.7	54.1
BNN-4b	81.9	80.74	72.5	66.9	59.73
Full-4b	81.1	80.75	71.7	65.4	57.4

7.4.6 Comparison with Prior Works

A significant contribution of our analysis is that we include input or parameter discretization during training and highlight the relevance of discretized training for adversarial robustnees. To further elucidate the effectiveness of discretization during training phase, we compare our results with that of prior works of Xu et al. [145] and Galloway et al. [149]. Xu et al. propose to use color depth bit reduction along with 2×2 median filter smoothing (referred to as *feature squeezing*) to detect adversarial images. However, Xu et al. train a model with full-precision inputs and use feature squeezing during the test phase to identify clean vs. adversarial input. We replicated their experiments by utilizing the open source framework and code made available by the authors [157]. Galloway et al. simply analysed the effect of attacking binarized neural networks (without input discretization). We utilized the open source framework [158], to investigate how input-and-parameter discretized models perform in attack scenarios (crafted with the widely used CleverHans library to benchmark adversarial vulnerability of neural networks [159]) as compared to only parameter discretized models.

Table 7.9 - 7.12 summarizes our comparative analysis for MNIST, CIFAR10 and ImageNet. Here, we conducted multi-step white box iterative attacks with iso-attack, perturbation parameters and model architectures to have a fair comparison with prior

works. The various cases are as follows: a) Carlini-Wagner L2 attack (CWL2 [160]) over 10, 40, 100 attack iterations [149] on FCN1 architecture for MNIST (Table 7.9), b) Projected Gradient Descent (PGD [152]) attack over 100 attack iterations with $\epsilon = 0.3$ for MNIST on a 4-layer convolutional architecture as [144, 145] (Table 7.10), c) PGD attack over 40 attack iterations with $\epsilon = 8/255$ for CIFAR10 on a ResNet architecture as [144, 145] (Table 7.11), d) PGD attack over 40 attack iterations with $\epsilon = 4/255$ for Imagenet on a InceptionResNetV2 architecture as [144, 145] (Table 7.12).

Table 7.9.
WB adversarial accuracy subject to Carlini-Wagner L2 attacks of increasing strength on MNIST dataset with different combinations of input and parameter discretization. The adversarial accuracy shown is for different iterations of CWL2 attacks: 10, 40, 100.

Network	Model	Clean	CWL2 Attack Iterations: 10/ 40/ 100
FCN1	BNN-2b (Galloway [149])	99	98/38/0.1
	BNN-2b (Ours)	96.4	96/57/10

Table 7.10.
WB adversarial accuracy subject to PGD attacks of $\epsilon = 0.3$ over 100 steps on MNIST dataset with different combinations of input and parameter discretization.

Network	Model	Clean	PGD Attack $\epsilon = 0.3$
4-layer CNN	Full-2b (Xu [145])	98.8	75.3
	BNN-2b (Ours)	96.4	82.7

In each case, we compare our combined input/parameter discretized network proposal (either XNOR or BNN with $2b, 3b$ input precision: $BNN - 2b, BNN -$

Table 7.11.
WB adversarial accuracy subject to PGD attacks of $\epsilon = 8/255$ over 40 steps on CIFAR10 dataset with different combinations of input and parameter discretization.

Network	Model	Clean	PGD Attack $\epsilon = 8/255$
ResNet	Full-3b (Xu [145])	76.1	32.5
	BNN-3b (Ours)	78.4	42

Table 7.12.
WB adversarial accuracy subject to PGD attacks of $\epsilon = 4/255$ over 40 steps on Imagenet dataset with different combinations of input and parameter discretization.

Network	Model	Clean	PGD Attack $\epsilon = 4/255$
InceptionResNetV2	Full-3b (Xu [145])	65.6	10.1
	XNOR-3b (Ours)	59	18.8

$3b, XNOR-3b$) that are trained with quantized input and weight values to prior works : a) Xu et al. where only $2b, 3b$ input discretization is performed during the testing phase on a full-precision network ($Full-3b, Full-2b$), b) Galloway et al. [149] where only parameter discretization is performed and only MNIST results for iterative WB attacks are available. As mentioned earlier, the inclusion of input/parameter discretization during training serves as a major benefactor in advancing adversarial robustness of models. Consequently, we observe a substantial improvement over Xu et al. in Table 7.10,7.11, 7.12. It is worth mentioning that iterative attacks cause drastic drop in accuracy in CIFAR10 and Imagenet datasets than simpler MNIST data. This can be addressed by using adversarial training defense. Thus, we can deduce that while discretization extends the inherent robustness of a model, adversarial training is pertinent for defense against stronger iterative attacks. The results

in Table 7.9 illustrate the effectiveness of input discretization (during training) as an extra layer of intrinsic defense in addition to the binarized parameter or activation model.

7.5 Discussion

Low-precision models or quantization techniques, so far, have been explored to reduce the resource utilization of DLNs for energy-efficient deployment on edge devices. We have demonstrated that *discretization* also warrants *security* against adversarial attacks, thereby, offering a key benefit of *robustness* in hardware implementation for a certain range of attacks. Our work also exposes the vulnerability of quantization or discretization techniques for multi-step iterative attacks necessitating the use of adversarial training with discretization as an improved defense strategy. In summary, the main findings/recommendations from this work are:

- Input discretization is major benefactor for adversarial robustness (with both binarized and full-precision) models. $2b$ input discretized models (without adversarial training) yield similar adversarial accuracy as adversarially trained $8b$ input models for lower ϵ values. Robustness against higher ϵ and multi-step attack requires adversarial training.

- Binarized (low-precision weights/activations) models are intrinsically more (although, slight) robust than full-precision ($32b$ weights/activations) models. Adversarial training needs to be carefully done on sufficient capacity binarized networks to attain similar adversarial robustness as the full-precision models.

- Combining input and parameter discretization is an efficient way of obtaining adversarial robustness to a moderate range of perturbation values without conducting the iterative adversarial training.

- Performing adversarial training on models with combined input and parameter discretization improves the adversarial accuracy ($\sim>$ 10%) in comparison to adversarially trained full-precision models with $8b$ inputs.

Our work unravels a simple idea that: *hardware optimization related techniques can potentially resolve or resist software vulnerabilities (specifically, adversarial attacks).* While we focus on discretization, there is a lot of future scope to explore other efficiency-driven techniques (such as, stochasticity, model pruning etc.) to gauge their implication on adversarial robustness. As seen earlier, standard discretization remains vulnerable to stronger multi-step attack scenarios and hence needs to be defended with adversarial training or data augmentation. We believe that stochasticity or pruning techniques tied with discretization can possibly resist better, even against stronger attacks. This requires further investigation. Finally, through this work, our goal is to project the security advantages of discretization techniques that are primarily used to deploy efficient DLN models in general-purpose (GPU) or accelerator (TPU or FPGA) computing platforms, thereby, paving possible research directions of exploring efficiency-robustness tradeoff in artificial intelligence applications.

8. IMPLICIT GENERATIVE MODELING OF RANDOM NOISE DURING TRAINING FOR ADVERSARIAL ROBUSTNESS

8.1 Introduction

Despite surpassing human performance on several perception tasks, Machine Learning (ML) models remain vulnerable to *adversarial examples*. In fact, adversarial inputs *transfer* across models: same inputs are misclassified by different models trained for the same task, thus enabling simple *Black-Box* (BB) [1]attacks against deployed ML systems [161].

Several works [162–164] demonstrating improved adversarial robustness have been shown to fail against stronger attacks [165]. The state-of-the-art approach for BB defense is ensemble adversarial training that augments the training dataset of the target model with adversarial examples transferred from other pre-trained models [166]. [167] showed that models can even be made robust to *White-Box* (WB)[1] attacks by closely maximizing the model's loss with Projected Gradient Descent (PGD) based adversarial training. Despite this progress, errors still appear for perturbations beyond what the model is adversarially trained for [168].

There have been several hypotheses explaining the susceptibility of ML models to such attacks. The most common one suggests that the overly linear behavior of deep neural models in a high dimensional input space causes adversarial examples [7, 169]. Another hypothesis suggests that adversarial examples are off the data manifold [170–172]. Combining the two, we infer that excessive linearity causes models to extrapolate their behavior beyond the data manifold yielding pathological results for

[1]BB (WB): attacker has no (full) knowledge of the target model parameters

slightly perturbed inputs. A question worth asking here is: *Can we improve the viability of the model to generalize better on such out-of-sample data?*

In this chapter, we propose *Noise-based Prior Learning (NoL)*, wherein we introduce multiplicative noise into the training inputs and optimize it with Stochastic Gradient Descent (SGD) while minimizing the overall cost function over the training data. Essentially, the input noise (randomly initialized at the beginning) is gradually learnt during the training procedure. As a result, the noise approximately models the input distribution to effectively maximize the likelihood of the class labels given the inputs. Fig. 8.1 (a) shows the input noise learnt during different stages of training by a simple convolutional network ($ConvNet2$ architecture discussed in Section 8.3.2 below), learning handwritten digits from MNIST dataset [48]. We observe that the noise gradually transforms and finally assumes a shape that highlights the most dominant features in the MNIST training data. For instance, the MNIST images are centered digits on a black background. Noise, in fact, learnt this centered characteristic. Fig. 8.1 suggests that noise discovers some knowledge about the input/output distribution during training. Fig. 8.1 (b) shows the noise learnt with NoL on colored CIFAR10 images [50] (on ResNet18 architecture [154]), which reveals that noise template (also RGB) learns prominent color blobs on a greyish-black background, that de-emphasizes background pixels.

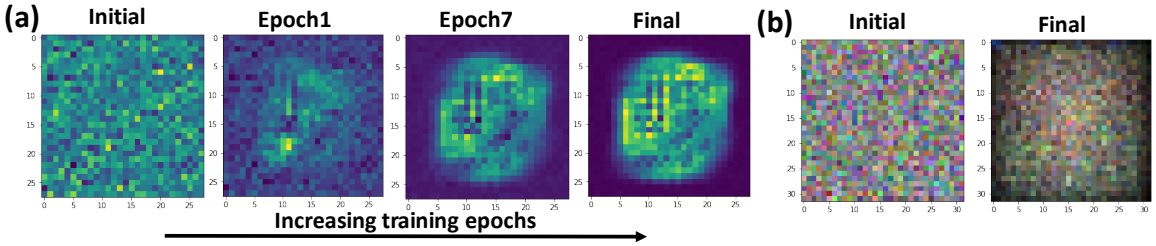


Fig. 8.1. (a) Noise learnt with NoL on MNIST data- (b) Noise learnt with NoL on CIFAR10 data- with mini-batch size =64.The template shown is the mean across all 64 noise templates.

A recent theory [173] suggests that adversarial examples (off manifold misclassified points) occur in close proximity to randomly chosen inputs on the data manifold that are, in fact, correctly classified. With NoL, we hypothesize that the model learns to look in the vicinity of the on-manifold data points and thereby incorporate more out-of-sample data (without using any direct data augmentation) that, in turn, improves its generalization capability in the off-manifold input space. We empirically evaluate this hypothesis by visualizing and studying the relationship between the adversarial and the clean inputs using Principal Component Analysis (PCA). Examining the intermediate layer's output, we discover that models exhibiting adversarial robustness yield significantly lower distance between adversarial and clean inputs in the Principal Component (PC) subspace.We further harness this result to establish that NoL noise modeling, indeed, acquires an improved realization of the input/output distribution characteristics that enables it to generalize better. To further substantiate our hypothesis, we also show that NoL globally reduces the dimensionality of the space of adversarial examples [174]. We evaluate our approach on classification tasks such as MNIST, CIFAR10 and CIFAR100 and show that models trained with NoL are extensively more adversarially robust. We also show that combining NoL with ensemble/PGD adversarial training significantly extends the robustness of a model, even beyond what it is adversarially trained for, in both BB/WB attack scenarios.

8.2 Noise-based Prior Learning

8.2.1 Approach

The basic idea of NoL is to inject random noise with the training data, continually minimizing the overall loss function by learning the parameters, as well as the noise at every step of training. The noise, N, dimensionality is same as the input, X, that is, for a $32 \times 32 \times 3$ sized image, the noise is $32 \times 32 \times 3$. In all our experiments, we use mini-batch SGD optimization. Let's assume the size of the training minibatch is m and the number of images in the minibatch is k, then, total training images

are $m \times k$. Now, the total number of noisy templates are equal to the total number of inputs in each minibatch, k. Since, we want to learn the noise, we use the same k noise templates across all mini-batches $1, 2, ..., m$. This ensures that the noise templates inherit characteristics from the entire training dataset. Algorithm 1 shows the training procedure. It is evident from Algorithm 1 that noise learning at every training step follows the overall loss ($\mathcal{L}$, say cross-entropy) minimization that in turn enforces the maximum likelihood of the posterior.

Algorithm 6 Noise-based Prior Learning of a model f with parameters θ, Loss Function $\mathcal{L}$.

Input: Input image X, Target label Y, Noise N, Learning rates η, η_{noise}

Output: Learnt noise N and parameters θ.

1: Randomly initialize the parameters θ and Noise $N : \{N^1, ...N^k\}$.
2: **repeat**
3: **for** each minibatch $\{X^{[1]}, ..., X^{[m]}\}$
4: Input $X = \{X^1, ..., X^k\}$
5: New input $X = \{X^1 \times N^1, ..., X^k \times N^k\}$
6: **Forward Propagation:** $\hat{Y} = f(X; \theta)$
7: **Compute loss function:** $\mathcal{L}(\hat{Y}, Y)$
8: **Backward Propagation:** $\theta = \theta$ - $\eta \nabla_\theta \mathcal{L}$; $N = N$ - $\eta_{noise} \nabla_N \mathcal{L}$
9: **end**
10: **until** training converges

Since adversarial attacks are created by adding perturbation to the clean input images, we were initially inclined toward using additive noise ($X + N$) instead of multiplicative noise ($X \times N$) to perform NoL. However, we found that NoL training with $X \times N$ tends to learn improved noise characteristics by the end of training. Fig. 8.2 (a) shows the performance results for different NoL training scenarios. While NoL with $X + N$ suffers a drastic $\sim 10\%$ accuracy loss with respect to standard SGD on clean data, $X \times N$ yields comparable accuracy. Furthermore, we observe that using

only negative gradients for training the noise (i.e. $\nabla_N \mathcal{L} \leq 0$) during backpropagation with NoL yields best accuracy (and closer to that of standard SGD trained model). Visualizing a sample image with learnt noise after training, in Fig. 8.2 (b), shows $X + N$ disturbs the original image severely, while $X \times N$ has a faint effect, corroborating the accuracy results. Since noise is modeled while conducting discriminative training, the multiplicative/additive nature of noise influences the overall optimization. Thus, we observe that noise templates learnt with $X \times N$ and $X + N$ are very different. We also analyzed the adversarial robustness of the models when subjected to WB attacks created using the Fast Gradient Sign Method (FGSM) for different perturbation levels (ϵ) (Fig. 8.2 (a)). NoL, for both $X \times N / X + N$ scenarios, yields improved accuracy than standard SGD. This establishes the effectiveness of the noise modeling technique during discriminative training towards improving a model's intrinsic adversarial resistance. Still, $X \times N$ yields slightly better resistance than $X + N$. Based upon these empirical studies, we chose to conform to multiplicative noise training. [2] Note, WB attacks, in case of NoL, are crafted using the model's parameters as well as the learnt noise N.

In all our experiments, we initialize the noise N from a random uniform distribution in the range $[0.8, 1]$. We select a high range in the beginning of training to limit the corruption induced on the training data due to the additional noise. During evaluation/testing, we take the mean of the learnt noise across all the templates $((\sum_{i=1}^{k} N_i)/k)$, multiply the averaged noise with each test image and feed it to the network to obtain the final prediction. Next, we present a general optimization perspective considering the maximum likelihood criterion for a classification task to explain adversarial robustness. It is worth mentioning that while Algorithm 1 describes the backpropagation step simply by using gradient updates, we can use other techniques like regularization, momentum etc. for improved optimization.

[2] Additional studies on other datasets comparing $X + N$ vs. $X \times N$ with different gradient update conditions can be found in Appendix (Section 8.5.1). See, experimental details and model description for Fig. 8.2 in Appendix (Section 8.5.3).

It is noteworthy to reiterate that in our NoL approach, we introduce a random set of noise templates (say, K noise templates) to the training data (say, M number of total training instances in the dataset), where, $K << M$. Here, as we learn the noise templates ($N : \{N^1, ...N^K\}$) over the duration of training, the noise templates (randomly initialized at the beginning of training) begin to model the input distribution (as seen from Fig. 8.1). The fact that the same set of noise templates are introduced batch-wise (when training with mini-batch gradient descent) with the training data, enables the prior modeling property. Thus, in Fig. 8.1, we see the evolution of the noise templates from random distribution in the initial epoch to some shape (characteristic of input data) in the final epoch (as they are learnt with backpropagation $N := N - \nabla_N \mathcal{L}$). NoL is in contrast to previous works [175–177], wherein, random noise is injected to the training dataset, which can be viewed as having same number of noise templates as the number of training instances, that is, $K = M$. Here, the noise templates are simply used to perturb the training data to impose some regularization effect during training. These noise templates are 'not learnt' unlike NoL. In such cases, each batch (when training with mini-batch gradient descent) of training data sees a different set of random noise templates drawn from a given distribution. Referring to Fig 8.1, with random noise injection as [175–177], we will see that the noise template remains random as shown in the initial epoch throughout the training process till the final epoch.

8.2.2 Adversarial Robustness from Likelihood Perspective

Given a data distribution D with inputs $X \in R^d$ and corresponding labels Y, a classification/discriminative algorithm models the conditional distribution $p(Y|X;\theta)$ by learning the parameters θ. Since X inherits only the on-manifold data points, a standard model thereby becomes susceptible to adversarial attacks. For adversarial robustness, inclusion of the off-manifold data points while modeling the conditional probability is imperative. An adversarially robust model should, thus, model

$p(Y|X, A; \theta)$, where A represents the adversarial inputs. Using Bayes rule, we can derive the prediction obtained from posterior modeling from a generative standpoint as:

$$\underset{Y}{argmax}\, p(Y|X, A) = \underset{Y}{argmax}\, \frac{p(A|X, Y)p(X, Y)}{p(X, A)} = \underset{Y}{argmax}\, p(A|X, Y)p(X|Y)p(Y) \tag{8.1}$$

(a)

Scenario	Gradient $\nabla_N \mathcal{L}$	Clean	WhiteBox
X+N	Negative	78.1	59.1/49.6
	All	77.1	57.48/49.03
X×N	Negative	87.08	62.5/51.6
	All	85.1	62.4/51.9
SGD (X)	-	88.85	50.6/44.9

CIFAR10 Accuracy (in %) of ResNet18 target model for WB-FGSM attacks for ε =(8/16)/255 for different training scenarios with ExL.

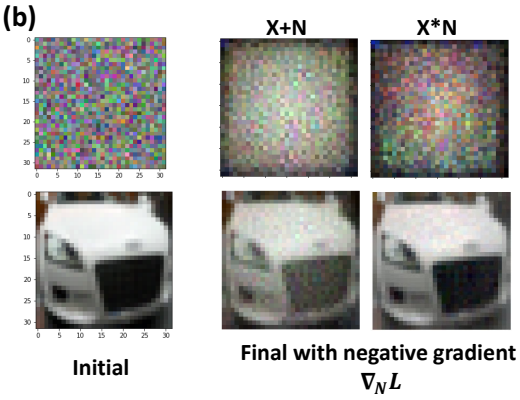


Fig. 8.2. For multiplicative and additive noise training scenarios- (a) - accuracy comparison of NoL with SGD (b) -RGB noise template learnt with NoL on CIFAR10 data. In (b), a sample training image of a 'car' before and after training with noise is shown. Note, we used the same hyperparameters (batch-size =64, η, η_{noise} etc.) and same inital noise template across all scenarios during training. Noise shown is the mean across 64 templates.[2]

The methods employing adversarial training [10, 166, 167] directly follow the left-hand side of Eqn. 8.1 wherein the training data is augmented with adversarial samples ($A \in A$). Such methods showcase adversarial robustness against a particular form of adversary (e.g. ℓ_∞-norm bounded) and hence remain vulnerable to stronger attack scenarios. In an ideal case, A must encompass all set of adversarial examples (or the entire space of off-manifold data) for a concrete guarantee of robustness. However, it is infeasible to anticipate all forms of adversarial attacks during training. From a generative viewpoint (right-hand side of Eqn. 8.1), adversarial robustness requires

modeling of the adversarial distribution while realizing the joint input/output distribution characteristics ($p(X|Y), p(Y)$). Yet, it remains a difficult engineering challenge to create rich generative models that can capture these distributions accurately. Some recent works leveraging a generative model for robustness use a PixelCNN model [171] to detect adversarial examples, or use Generative Adversarial Networks (GANs) to generate adversarial examples [178]. But, one might come across practical difficulties while implementing such methods due to the inherent training difficulty.

With Noise-based Prior Learning, we partially address the above difficulty by modeling the noise based on the prediction loss of the posterior distribution. First, let us assume that the noise (N) introduced with NoL spans a subspace of potential adversarial examples ($N \subseteq A$). Based on Eqn. 8.1 the posterior optimization criterion with noise (N) becomes $argmax_Y\, p(Y|X,N) = argmax_Y\, p(N|X,Y)p(X|Y)p(Y)$. The noise learning in NoL (Algorithm 1) indicates an implicit generative modeling behavior, that is constrained towards maximizing $p(N|X,Y)$ while increasing the likelihood of the posterior $p(Y|X,N)$. We believe that this partial and implicit generative modeling perspective with posterior maximization, during training, imparts an NoL model more knowledge about the data manifold, rendering it less susceptible toward adversarial attacks.

Intuitively, we can justify this robustness in two ways: First, by integrating noise during training, we allow a model to explore multiple directions within the vicinity of the data point (thereby incorporating more off-manifold data) and hence inculcate that knowledge in its underlying behavior. Second, we note that noise learnt with NoL inherits the input data characteristics (i.e. $N \subset X$) and that the noise-modeling direction ($\nabla_N \mathcal{L}$) is aligned with the loss gradient, $\nabla_X \mathcal{L}$ (that is also used to calculate the adversarial inputs, $X_{adv} = X + \epsilon sign(\nabla_X \mathcal{L})$). This ensures that the exploration direction coincides with certain adversarial directions improving the model's generalization capability in such spaces. Next, we empirically demonstrate using PCA that, noise modeling indeed embraces some off-manifold data points. Note, for fully guaranteed adversarial robustness as per Eqn. 8.1, the joint input/output distribution

$(p(X|Y), p(Y))$ has to be realized in addition to the noise modeling and N should span the entire space of adversarial/off-manifold data. We would like to clarify that the above likelihood perspective is our intuition that noise might be inculcating some sort of generaive behavior in the discriminative classification. However, more rigorous theoretical analysis is required to understand the underlying behavior of noise.

8.2.3 PC Subspace Analysis for Variance & Visualization

PCA serves as a method to reduce a complex dataset to lower dimensions to reveal sometimes hidden, simplified structure that often underlie it. Since the learned representations of a deep learning model lie in a high dimensional geometry of the data manifold, we opted to reduce the dimensionality of the feature space and visualize the relationship between the adversarial and clean inputs in this reduced PC subspace. Essentially, we find the principal components (or eigen-vectors) of the activations of an intermediate layer of a trained model and project the learnt features onto the PC space. To do this, we center the learned features about zero ($\mathcal{F}$), factorize $\mathcal{F}$ using Singular Value Decomposition (SVD), i.e. $\mathcal{F} = USV^T$ and then transform the feature samples $\mathcal{F}$ onto the new subspace by computing $\mathcal{F}V = US \equiv \mathcal{F}^{PC}$. In Fig. 8.3 (b), we visualize the learnt representations of the *Conv1 layer* of a ResNet18 model trained on CIFAR-10 (with standard SGD) along different 2D-projections of the PC subspace in response to adversarial/clean input images. Interestingly, we see that the model's perception of both the adversarial and clean inputs along high-rank PCs (say, PC1- PC10 that account for maximum variance in the data) is alike. As we move toward lower-rank dimensions, the adversarial and clean image representations dissociate. This implies that adversarial images place strong emphasis on PCs that account for little variance in the data. While we note a similar trend with NoL (Fig. 8.3 (a)), the dissociation occurs at latter PC dimensions compared to Fig. 8.3 (b). A noteworthy observation here is that, adversarial examples lie in close vicinity of the clean inputs for both NoL/SGD scenarios ascertaining former theories of [173].

To quantify the dissociation of the adversarial and clean projections in the PC subspace, we calculate the cosine distance ($\mathcal{D}^{PC} = \frac{1}{N}\sum_{i=1}^{N} 1 - \frac{\mathcal{F}^{PC}_{clean\,i} \cdot \mathcal{F}^{PC}_{adv\,i}}{\mathcal{F}^{PC}_{clean\,i2}\mathcal{F}^{PC}_{adv\,i2}}$) between them along different PC dimensions. Here, N represents the total number of sample images used to perform PCA and $F^{PC}_{clean}(F^{PC}_{adv})$ denote the transformed learnt representations corresponding to clean (adversarial) input, respectively. The distance between the learnt representations (for the *Conv1 layer* of ResNet18 model from the above scenario) consistently increases for latter PCs as shown in Fig. 8.4 (a). Interestingly, the cosine distance between adversarial and clean features measured for a model trained with NoL noise is significantly lesser than a standard SGD trained model. This indicates that noise enables the model to look in the vicinity of the original data point and inculcate more adversarial data into its underlying representation. Note, we consider projection across all former dimensions (say, PC0, PC1,...PC100) to calculate the distance at a later dimension (say, PC100) i.e., ${\mathcal{D}^{PC}}_{100}$ is calculated by taking the dot product between two 100-dimensional vectors: $\mathcal{F}^{PC}_{clean}, \mathcal{F}^{PC}_{adv}$. Please note, in the remainder of the chapter, *NoL noise* denotes the prior or noise learnt during the training procedure by minimizing the loss of a neural network. As per Algorithm 1, we use the same set of templates over each training minibatch that ensures that the noise templates randomly initialized at the beginning of training learn and evolve to model the input characteristics (as illustrated in Fig. 8.1). Specifically, *NoL noise* refers to the learnt noise templates $N : \{N^1, ...N^k\}$ as shown in Algorithm 1.

To further understand the role of NoL noise in a model's behavior, we analyzed the variance captured in the *Conv1* layer's activations of the ResNet18 model (in response to clean inputs) by different PCs, as illustrated Fig. 8.4 (b). If $s_i = \{1, ..., M\}$ are the singular values of the matrix S, the variance along a particular dimension PC_k is defined as: $Var_k = 100 \times (\sum_{i=0}^{k} {s_i}^2 / \sum_{i=0}^{M} {s_i}^2)$. Var_k along different PCs provides a good measure of how much a particular dimension explains about the data. We observe that NoL noise increases the variance along the high rank PCs, for instance, the net variance obtained from PC0-PC100 with NoL Noise (90%) is more than that of

standard SGD (76%). In fact, we observe a similar increase in variance in the leading PC dimensions for other intermediate blocks learnt activations of the ResNet18 model [See *Appendix (Section 8.5.2)*]. We can infer that the increase in variance along the high-rank PCs is a consequence of inclusion of more data points during the overall learning process. Conversely, we can also interpret this as NoL noise embracing more off-manifold adversarial points into the overall data manifold that eventually determines the model's behavior. It is worth mentioning that the variance analysis of the model's behavior in response to adversarial inputs yields nearly identical results as Fig. 8.4 (b) [*Appendix (Section 8.5.2)*].

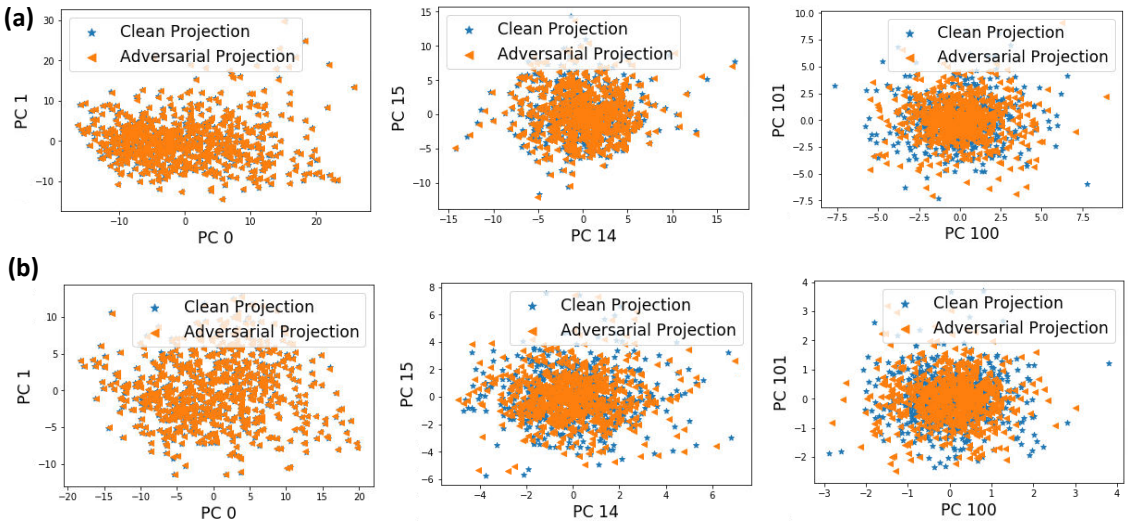


Fig. 8.3. Relationship between the model's understanding of adversarial and clean inputs in PC subspace when trained with (a) NoL (b) SGD.

Interestingly, the authors in [179] conducted PCA whitening of the raw image data for clean and adversarial inputs and demonstrated that adversarial image coefficients for later PCs have greater variance. Our results from PC subspace analysis corroborates their experiments and further enables us to peek into the model's behavior for adversarial attacks. Note, for all the PCA experiments above, we used 700 random images sampled from the CIFAR-10 test data, i.e. $N = 700$. In addition, we used

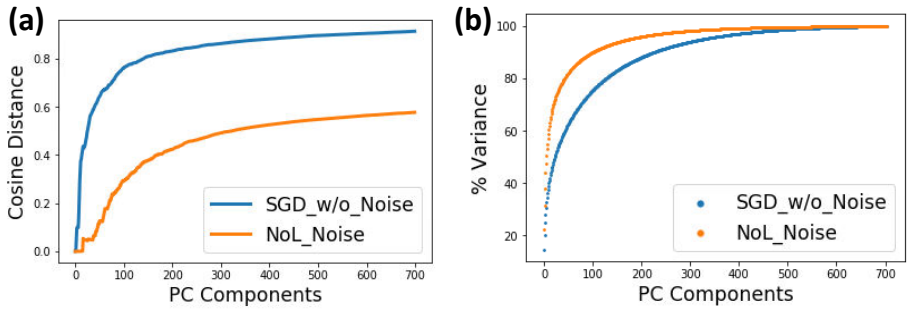


Fig. 8.4. (a) Cosine Distance between the model's response to clean and adversarial inputs in the PC subspace. (b) Variance of the $Conv1$ layer of ResNet18 model. (a), (b) compare the SGD/ NoL training scenarios.

the Fast Gradient Sign Method (FGSM) method to create BB adversaries with a step size of 8/255, from a different source model (ResNet18 trained with SGD).

8.3 Results

8.3.1 Attack Methods

Given a test image X, an attack model perturbs the image to yield an adversarial image, $X_{adv} = X + \Delta$, such that a classifier f misclassifies X_{adv}. In this work, we consider ℓ_∞ bounded adversaries studied in earlier works [7, 166, 167], wherein the perturbation ($\Delta_\infty \leq \epsilon$) is regulated by some parameter ϵ. Also, we study robustness against both BB/WB attacks to gauge the effectiveness of our approach. For an exhaustive assessment, we consider the same attack methods deployed in [166, 167]:

Fast Gradient Sign Method (FGSM): This single-step attack is a simple way to generate malicious perturbations in the direction of the loss gradient $\nabla_X \mathcal{L}(X, Y_{true})$ as: $X_{adv} = X + \epsilon sign(\nabla_X \mathcal{L}(X, Y_{true}))$.

Random Step FGSM (R-FGSM): [166] suggested to prepend single-step attacks with a small random step to escape the non-smooth vicinity of a data point that might degrade attacks based on single-step gradient computation. For parameters ϵ, α ($\alpha = \epsilon/2$), the attack is defined as: $X_{adv} = X + \epsilon sign(\nabla_X \mathcal{L}(X, Y_{true}))$, $where\ X =$

$X + \alpha sign(\mathcal{N}(0^d, I^d))$.

Iterative FGSM (I-FGSM): This method iteratively applies FGSM k times with a step size of $\beta \geq \epsilon/k$ and projects each step perturbation to be bounded by ϵ. Following [166], we use two-step iterative FGSM attacks.

Projected Gradient Descent (PGD): Similar to I-FGSM, this is a multi-step variant of FGSM: ${X_{adv}}^{t+1} = \prod({X_{adv}}^t + \alpha sign(\nabla_X \mathcal{L}(X, Y_{true})))$. [167] show that this is a universal first-order adversary created by initializing the search for an adversary at a random point followed by several iterations of FGSM. PGD attacks, till date, are one of the strongest BB/ WB adversaries.

8.3.2 Experiments

We evaluated NoL on three datasets: MNIST, CIFAR10 and CIFAR100. For each dataset, we report the accuracy of the models against BB/WB attacks (crafted from the test data) for 6 training scenarios: a) Standard SGD (without noise), b) NoL, c) Ensemble Adversarial (EnsAdv) Training (SGD_{ens}), d) NoL with EnsAdv Training (NoL_{ens}), e) PGD Adversarial (PGDAdv) Training (SGD_{PGD}), f) NoL with PGDAdv Training (NoL_{PGD}). Note, SGD_{ens} and SGD_{PGD} refer to the standard adversarial training employed in [166] and [167], respectively. Our results compare how the additional noise modeling improves over standard SGD in adversarial susceptibility. Also, we integrate NoL with state-of-the-art PGD/Ensemble adversarial training techniques to analyze how noise modeling benefits them. In case of EnsAdv training, we augmented the training dataset of the target model with adversarial examples (generated using FGSM), from an independently trained model, with same architecture as the target model. In case of PGDAdv training, we augmented the training dataset of the target model with adversarial examples (generated using PGD) from the same target model. Thus, as we see later, EnsAdv imparts robustness against BB attacks only, while, PGD makes a model robust to both BB/WB attacks. In all experiments below, we report the WB/BB accuracy against strong adversaries created

with PGD attack. In additon, for BB, we also report the worst-case error over all small-step attacks FGSM, I-FGSM, R-FGSM, denoted as *Min BB* in Table 8.1, 8.2.

Note, NoL scenario refers to our proposed approach (or Algorithm 1) wherein we introduce multiplicative noise at the beginning of training and eventually learn the noise while minimizing the loss for model parameters. NoL_{ens} refers to the scenario wherein we combine EnsAdv training with NoL (or Algorithm 1 with EnsAdv training). That is, a model is trained with adversarial data augmentation and also has multiplicative noise that is eventually learnt during the training procedure. In this case, since we show both clean and FGSM-based BB adversarial input data during training, we can expect that the learnt noise will model both clean/BB-adversarial input distribution. NoL_{PGD} refers to the scenario where PGDAdv training is combined with NoL (or Algorithm 1 with PGDAdv training). Here, we perform the prior noise modeling (with multiplicative noise injected at the beginning of training) while training a network with both clean and PGD-based WB adversarial data. We can expect the learnt noise to model both clean/WB-adversarial input distribution in this case. As we will see later, NoL_{ens}, NoL_{PGD} will serve as good indicators of how integrating the prior noise learning approach with the adversarial defense methods, SGD_{ens}, SGD_{PGD}, improves their ability to defend against a larger range of perturbations/attacks.

All networks were trained with mini-batch SGD using a batch size of 64 and momentum of 0.9 (0.5) for CIFAR (MNIST), respectively. For CIFAR10, CIFAR100 we used additional weight decay regularization, $\lambda = 5e-4$. Note, for noise modeling, we simply used the negative loss gradients ($\nabla_N \mathcal{L} \leq 0$) without additional optimization terms. In general, NoL requires slightly more epochs of training to converge to similar accuracy as standard SGD, a result of the additional input noise modeling. Also, NoL models, if not tuned with proper learning rate, have a tendency to overfit. Hence, the learning rate for noise (η_{noise}) was kept 1-2 orders of magnitude lesser than the overall network learning rate (η) throughout the training process. All networks were

implemented in PyTorch.[3]

MNIST: For MNIST, we consider a simple network with 2 Convolutional (C) layers with 32, 64 filters, each followed by 2×2 Max-pooling (M), and finally a Fully-Connected (FC) layer of size 1024, as the target model (ConvNet1: 32C-M-64C-M-1024FC). We trained 6 ConvNet1 models independently corresponding to the different scenarios. The EnsAdv (NoL_{ens}, SGD_{ens}) models were trained with BB adversaries created from a separate SGD-trained ConvNet1 model using FGSM with $\epsilon = 0.1$. PGDAdv (NoL_{PGD}, SGD_{PGD}) models were trained with WB adversaries created from the same target model using PGD with $\epsilon = 0.3$, step-size = 0.01 over 40 steps.

Table 8.1 (Columns 3 - 5) illustrates our results for BB attacks under different perturbations (ϵ)[4]. NoL noise considerably improves the robustness of a model toward BB attacks compared to standard SGD. Without adversarial training, standard SGD and NoL suffer a drastic accuracy decline for $\epsilon > 0.1$. With EnsAdv training, the robustness improves across all ϵ for both SGD_{ens}, NoL_{ens}. However, the robustness is significantly accentuated in NoL_{ens} owing to the additional noise modeling performed during EnsAdv training. An interesting observation here is that for $\epsilon = 0.1$ (that was the perturbation size for EnsAdv training), both NoL_{ens}/SGD_{ens} yield nearly similar accuracy, $\sim$ 98%. However, for larger perturbation size $\epsilon = 0.2, 0.3$, the network adversarially trained with NoL noise shows higher prediction capability ($\sim>$ 5%) across the PGD attack methods. Columns 6-7 in Table 8.1 show the WB attack results. All techniques except for the ones with PGDAdv training fail miserably against the strong WB PGD attacks. Models trained with NoL noise, although yielding low accuracy, still perform better than SGD. NoL_{PGD} yields better accuracy

[3] *Appendix (Section 8.5.3)* provides a detailed table of different hyperparameters used to train the source and target models in each scenario corresponding to all experiments of Table 8.1, 8.2 . Appendix (Section 8.5.3) shows different visualization of noise learnt (N) in each scenario of Table 8.1, 8.2.

[4] For fair comparison, BB attacks on $SGD, NoL, SGD_{ens}, NoL_{ens}$ were crafted from another model trained with standard SGD on natural examples as in [166]. While, BB attacks on SGD_{PGD}, NoL_{PGD} were crafted from a model trained with PGDAdv training (without noise modeling) on adversarial examples as in [167] to cast stronger attacks.

than SGD_{PGD}even beyond what the network is adversarially trained for ($\epsilon > 0.3$). Note, for PGD attack in Table 8.1, we used a step-size of 0.01 over 40/100 steps to create adversaries bounded by $\epsilon = 0.1/0.2/0.3$. We also evaluated the worst-case accuracy over all the BB attack methods when the source model is trained with NoL noise (not shown). We found higher accuracies in this case, implying NoL models *transfer* attacks at lower rates. As a result, in the remainder of the chapter, we conduct BB attacks from models trained without noise modeling to evaluate the adversarial robustness.

Table 8.1.
MNIST Accuracy (in %) of ConvNet1 target model for different scenarios. $\epsilon = 0.1/0.2/0.3$ for $SGD, NoL, SGD_{ens}, NoL_{ens}$; $\epsilon = 0.3/0.4$ for SGD_{PGD}, NoL_{PGD}. For PGD attack, we report accuracy for 40-/100-step attacks. Accuracy $< 5\%$, in most places, have been omitted and marked as '-'.

Scenario	Clean	Min BB	PGD-40	PGD-100	PGD-40	PGD-100
		(——————**BlackBox**——————)			(—**WhiteBox**—)	
SGD	99.1	77.9/20.6/4.3	75/9.9/-	74.5/8/-	22.3/-/-	-
NoL	99.2	83.6/30.5/9.6	80.5/20.6/-	80/18/-	29.4/-/-	-
SGD_{ens}	99	98.5/92.6/73.2	98/89.3/71	98.1/88/57	2.1/-/-	-
NoL_{ens}	99.1	99/94.7/76	98.8/93.4/79	98.7/91.9/66	3.3/-/-	-
SGD_{PGD}	97.9	91.8/29	93.6/48.7	92.3/20	90/27	86.5/4.5
NoL_{PGD}	98	93/42.2	94/60.4	92.6/28.7	90.7/55.7	88/20.1

CIFAR: For CIFAR10, we examined our approach on the ResNet18 architecture. We used the ResNext29(2×64d) architecture [180] with bottleneck width 64, cardinality 2 for CIFAR100. Similar to MNIST, we trained the target models separately corresponding to each scenario and crafted BB/WB attacks. For EnsAdv training, we used BB adversaries created using FGSM ($\epsilon = 8/255$) from a separate SGD-trained network different from the BB source/target model. For PGDAdv training, the target models were trained with WB adversaries created with PGD with $\epsilon = 8/255$,

step-size=2/255 over 7 steps. Here, for PGD attacks, we use 7/20 steps of size 2/255 bounded by ϵ. The results appear in Table 8.2.

For BB, we observe that NoL (81%/63.2% for CIFAR10/100) significantly boosts the robustness of a model as compared to SGD (50.3%/44.2% for CIFAR10/100). Note, the improvement here is quite large in comparison to MNIST (that shows only 5% increase from SGD to NoL). In fact, the accuracy obtained with NoL alone with BB attack, is almost comparable to that of an EnsAdv/PGDAdv trained model without noise (SGD_{ens}, SGD_{PGD}). The richness of the data manifold and feature representation space for larger models and complex datasets allows NoL to model better characteristics in the noise causing increased robustness. As seen earlier, NoL noise (NoL_{ens}, NoL_{PGD}) considerably improves the accuracy even for perturbations ($\epsilon = (16, 32)/255$) greater than what the network is adversarially trained for. The increased susceptibility of SGD_{ens}, SGD_{PGD} for larger ϵ establishes that its capability is limited by the diversity of adversarial examples shown during training. For WB attacks as well, NoL_{PGD} show higher resistance. Interestingly, while SGD, SGD_{ens} yield infinitesimal performance ($< 5\%$), NoL, NoL_{ens} yield reasonably higher accuracy ($> 25\%$) against WB attacks. This further establishes the potential of noise modeling in enabling adversarial security. It is worth mentioning that BB accuracy of SGD_{PGD}, NoL_{PGD} models in Table 8.1, 8.2 are lower than SGD_{ens}, NoL_{ens}, since the former is attacked with stronger attacks crafted from models trained with PGDAdv[4]. Attacking the former with similar adversaries as latter yields higher accuracy.

PC Distance & Variance Analysis : Next, we measured the variance and cosine distance captured by the $Conv1$ layer of the ResNet18 model corresponding to different scenarios (Table 8.2). Fig. 8.5 shows that variance across the leading PCs decreases as $NoL_{PGD} > SGD_{PGD} > NoL_{ens} > NoL > SGD_{ens} > SGD$. Inclusion of adversarial data points with adversarial training or noise modeling informs a model more, leading to improved variance. We note that NoL_{ens} and SGD_{PGD} yield nearly similar variance ratio, although SGD_{PGD} gives better accuracy than NoL_{ens} for similar BB and WB attacks. Since we are analyzing only the $Conv1$

Table 8.2.
CIFAR10/ CIFAR100 Accuracy (in %) of ResNet18/ ResNext-29 target model for different scenarios. $\epsilon = \frac{8}{255}/\frac{16}{255}/\frac{32}{255}$ for NoL, SGD, NoL_{ens}, $SGD_{ens}, NoL_{PGD}, SGD_{PGD}$. For PGD attack, we report accuracy for 7-/20-step attacks. Accuracy $< 5\%$, in most places, have been omitted and marked as '-'.

Scenario	Clean	Min BB	PGD-7	PGD-20	PGD-7	PGD-20
		(——————**BlackBox**——————)			(————**WhiteBox**————)	
	ResNet18 (CIFAR10)					
SGD	88.8	50.3/32/16.2	34/23/17.1	28/11.2/6.2	2.1/-/-	-
NoL	87.1	81/76/67	80.1/75.7/66.4	80/74.8/61	39.1/29.2/23	-
SGD_{ens}	86.3	81.3/76.6/68.3	80.9/75.1/67.1	80.2/74.4/63	0.8/-/-	-
NoL_{ens}	86.4	84.4/81.4/72.6	83/80/71.3	82.7/79/71	29/21/16.5	-
SGD_{PGD}	83.2	71.3/58/50	69.9/62/50.1	54.2/50.3/46	58.4/48/42	57.3/42.8/28
NoL_{PGD}	83	73/62/56.8	71/65/53	57.6/53.7/49.8	63/59/57	59.2/45/30.1
	ResNext29 (CIFAR100)					
SGD	71	44.2/38.4/26.7	42.7/35/25.4	40.5/27/17	-	-
NoL	69.4	63.2/58.5/50.1	62.9/54.3/48.4	62.3/53.1/42.5	19/14/10.3	-
SGD_{ens}	69.8	64.8/60.9/50	63.6/57.5/45.4	63/56/42	2.5/-/-	-
NoL_{ens}	67.3	65.1/62.8/57	64.8/61.4/52.2	64.4/58/49	18/14/11	-
SGD_{PGD}	71.6	57.5/48/38.4	56/45/41.3	48/40/38.4	51.5/49.8/46	50.4/43/33
NoL_{PGD}	69	66.3/62/59.9	63/58.7/54.1	52.3/50/40.8	58.1/56/53	53/48/37.9

layer, we get this discrepancy. In Fig. 8.5, we also plot the cosine distance between the adversarial (created from FGSM with specified ϵ) and clean inputs in the PC subspace. The distance across different scenarios along latter PCs increases as: $NoL_{PGD} < SGD_{PGD} < NoL_{ens} < NoL < SGD_{ens} < SGD$. A noteworthy observation here is, PC distance follows the same order as decreasing variance and justifies the accuracy results in Table 8.2. The decreasing distance with NoL compared to SGD further signifies improved realization of the on-/off-manifold data. Also, the fact that NoL_{PGD}, NoL_{ens} have lower distance for varying ϵ establishes that integrating noise modeling with adversarial training compounds adversarial robustness. Interestingly,

for both variance and PC distance, NoL has a better characteristic than SGD_{ens}. This proves that noise modeling enables implicit inclusion of adversarial data without direct data augmentation, as opposed to EnsAdv training (or SGD_{ens}) where the dataset is explicitly augmented. This also explains the comparable BB accuracy between NoL, SGD_{ens} in Table 8.2.

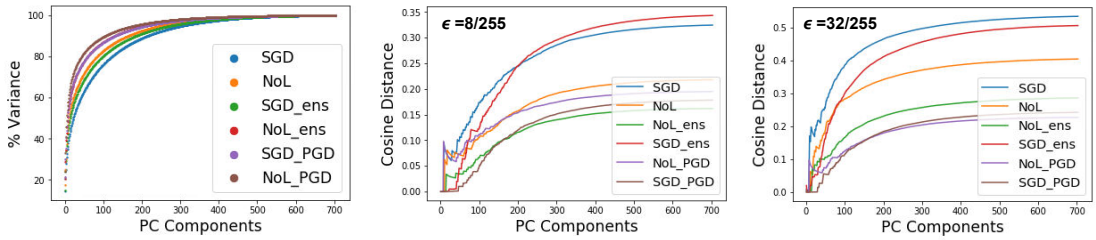


Fig. 8.5. [Left] Variance (in response to clean inputs) across different scenarios for the first 700 PC dimensions. [Middle, Right] Cosine distance across 700 PCs between clean and adversarial representations for varying ϵ. SGD_{ens}, SGD_{PGD} **exhibit improved variance (and lower distance) than** SGD**, suggesting PC variance/ distance as a good indicator of adversarial robustness.** PCA was conducted with sample of 700 test images.

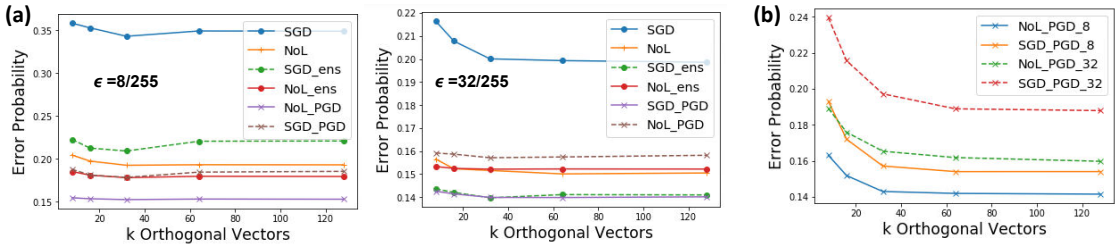


Fig. 8.6. Adversarial subspace dimensionality for varying ϵ for- (a) -BB adversaries crafted from a model trained with natural examples (b) -WB adversaries crafted for models trained with PGDAdv training.

Adversarial Subspace Dimensionality : To further corroborate that NoL noise implicitly embraces adversarial points, we evaluated the adversarial subspace

dimension using the Gradient-Aligned Adversarial Subspace (GAAS) method of [174]. We construct k orthogonal vectors $r_1, .., r_k \in \{-1, 1\}$ from a regular Hadamard matrix of order $k \in \{2^2, 2^3, .., 2^7\}$. We then multiply each r_i component-wise with the gradient, $sign(\nabla_X \mathcal{L}(X, Y_{true}))$. Hence, estimating the dimensionality reduces to finding a set of orthogonal perturbations, r_i with $r_{i\infty} = \epsilon$ in the vicinity of a data point that causes misclassification. For each scenario of Table 8.2 (CIFAR10), we select 350 random test points, x, and plot the probability that we find at least k orthogonal vectors r_i such that $x + r_i$ is misclassified. Fig. 8.6 (a), (b) shows the results with varying ϵ for BB, WB instances. We find that the size of the space of adversarial samples is much lower for a model trained with NoL noise than that of standard SGD. For $\epsilon = 8/255$, we find over 128/64 directions for $\sim 25\%/15\%$ of the points in case of SGD/NoL. With EnsAdv training, the number of adversarial directions for SGD_{ens}/NoL_{ens} reduces to 64 that misclassifies $\sim 17/15\%$ of the points. With PGDAdv training, the adversarial dimension significantly reduces in case of NoL_{PGD} for both BB/WB. As we increase the perturbation size ($\epsilon = 32/255$), we observe increasingly reduced number of misclassified points as well as adversarial dimensions for models trained with noise modeling. The WB adversarial plot, in Fig. 8.6 (b), clearly shows the reduced space obtained with noise modeling with PGDAdv training (NoL_{PGD}) against plain PGDAdv (SGD_{PGD}) for $\epsilon = (8, 32)/255$.

Loss Surface Smoothening: By now, it is clear that while NoL alone can defend against BB attacks (as compared to SGD) reasonably well, it still remains vulnerable to WB attacks. For WB defense and to further improve BB defense, we need to combine NoL noise modeling with adversarial training. To further investigate this, we plotted the loss surface of MNIST models on examples $x = x + \epsilon_1 \cdot g_{BB} + \epsilon_2 \cdot g_{WB}$ in Fig. 8.7, where g_{BB} is the signed gradient, $sign(\nabla_X \mathcal{L}(X, Y_{true})_{source})$, obtained from the source model (crafting the BB attacks) and g_{WB} is the gradient obtained from the target model itself (crafting WB attacks), $sign(\nabla_X \mathcal{L}(X, Y_{true})_{target})$. We see that the loss surface in case of SGD is highly curved with steep slopes in the vicinity

of the data point in both BB and WB direction. The EnsAdv training, SGD_{ens}, smoothens out the slope in the BB direction substantially, justifying their robustness against BB attacks. Models trained with noise modeling, NoL (even without any data augmentation), yield a softer loss surface. This is why NoL models *transfer* BB attacks at lower rates. The surface in the WB direction along ϵ_2 with NoL, NoL_{ens} still exhibits a sharper curvature (although slightly softer than SGD_{ens}) validating the lower accuracies against WB attacks (compared to BB attacks). PGDAdv, on the other hand, smoothens out the loss surface substantially in both directions owing to the explicit inclusion of WB adversaries during training. Note, NoL_{PGD} yields a slightly softer surface than SGD_{PGD} (not shown). The smoothening effect of noise modeling further justifies the boosted robustness of NoL models for larger perturbations (outside ϵ-ball used during adversarial training). It is worth mentioning that we get similar PCA/ Adversarial dimensionality/ loss surface results across all datasets.

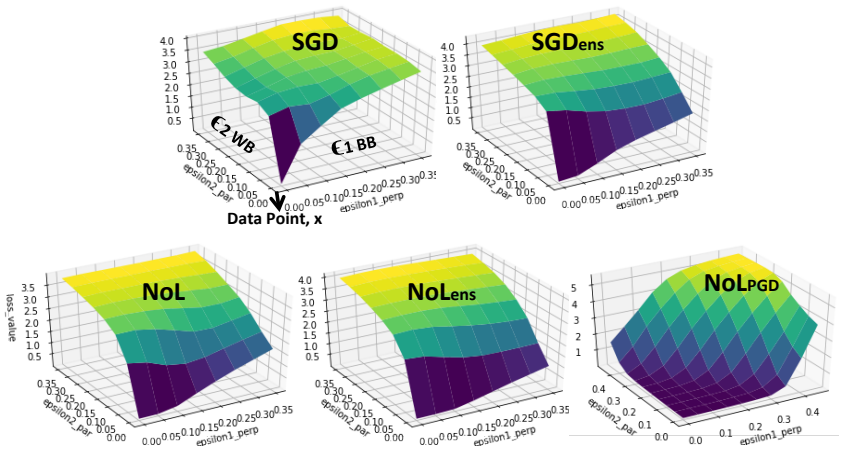


Fig. 8.7. Loss surface of models corresponding to MNIST (Table8.1).

8.4 Discussion

We proposed *Noise-based Prior Learning, NoL,* as a reliable method for improving adversarial robustness. Specifically, our key findings are:

1) We show that noise modeling at the input during discriminative training improves a model's ability to generalize better for out-of-sample adversarial data (without explicit data augmentation).

2) Our PCA variance and cosine distance analysis provides a significant perspective to visualize and quantify a model's response to adversarial/clean data.

A crucial question one can ask is, ***How to break NoL defense?*** The recent work [165] shows that many defense methods cause 'gradient masking' that eventually fail. We reiterate that, NoL alone does not give a strong BB/WB defense. However, the smoothening effect of noise modeling on the loss (Fig. 8.7) suggests that noise modeling decreases the magnitude of the gradient masking effect. NoL does not change the classification model that makes it easy to be scaled to larger datasets while integrating with other adversarial defense techniques. Coupled with other defense, NoL performs remarkably (even for larger ϵ values). We combine NoL with EnsAdv & PGDAdv, which do not cause obfuscated gradients and hence can withstand strong attacks, however, upto a certain point. For WB perturbations much greater than the training ϵ value, NoL+PGDAdv also breaks. In fact, for adaptive BB adversaries [166] or adversaries that query the model to yield full prediction confidence (not just the label), NoL+EnsAdv will be vulnerable. Note, advantage with NoL is, being independent of the attack/defense method, NoL can be potentially combined with stronger attacks developed in future, to create stronger defenses.

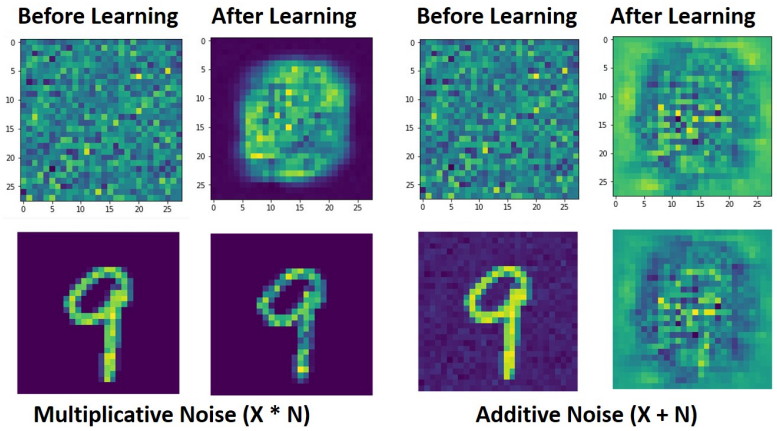
While variance and principal subspace analysis help us understand a model's behavior, we cannot fully describe the structure of the manifold learnt by the linear subspace view. However, PCA does provide a basic intuition about the generalization capability of complex image models. In fact, our PC results establish the superiority of PGDAdv training ([167] best defense so far), in general, as a strong

defense method and can be used as a valid metric to gauge adversarial susceptibility in future proposals. Finally, as our likelihood theory indicates, better noise modeling techniques with improved gradient penalties can further improve robustness and requires further investigation. Also, performing noise modeling at intermediate layers to improve variance, and hence robustness, are other future work directions.

A final note on novelty of NoL. As noted earlier, there have been past works [175–177] where the training dataset is perturbed with random noise, and the perturbed images are then used to train a deep neural network. The role of such noise is to impose better regularization effect (or reduce overfitting) during training which, in turn, improves a models generalization capability or inference accuracy [177]. It is worth mentioning that such methods do not result in adversarial robustness (also, verified in [7]). In the proposed NoL approach, the input noise (randomly initialized at the beginning of training) is gradually learnt during the training procedure. The fact that we train the noise (or prior) is the major distinguishing factor between our approach and previous works [175–177] that simply perturb the training data with random noise. The implicit prior modeling property of NoL goes beyond simple regularization and in fact, results in adversarial robustness (wherein, a model's ability to generalize on adversarial data improves). Our PC distance and variance analysis (in Section 8.2.3, 8.3.2) further attest the capability of the learnt noise (or prior) to embrace adversarial points into the data manifold thereby improving a model's adversarial accuracy. In fact, our results in Table 8.2 show that standalone NoL achieves better white-box accuracy than the ensemble adversarial training defense approach SGD_{ens}. This implies that even some white-box perturbations are inherited in this implicit noise modeling procedure. Such adversarial robustness cannot be attained with a setup of training over randomly perturbed examples as [175, 176]. Finally, we would also like to point out that our noise-based prior learning can possibly render some regularization effects on a model during training. The smoothened loss curves in Fig. 8.7 observed with NoL in comparison to SGD verifies the regularized model behavior.

8.5 Appendix

8.5.1 Justification of $X+N$ vs $X\times N$ and use of $\nabla\mathcal{L}_N \leq 0$ for noise modeling

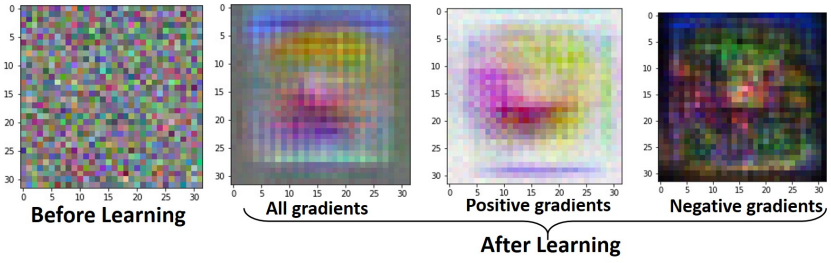


Scenario	Accuracy
SGD without noise	98.13%
NoL (X * N)	98.62%
NoL (X + N)	95.42%

Fig. 8.8. Noise learnt for multiplicative vs. additive noise inclusion during NoL training with MNIST data

In Fig. 8.8, for MNIST dataset, we show the noise template learnt when we use multiplicative/additive noise (N) for Noise-based Prior Learning. The final noise-integrated image (for a sample digit '9') that is fed to the network before and after training is also shown. Additive noise disrupts the image drastically. Multiplicative noise, on the other hand, enhances the relevant pixels while eliminating the background. Accuracy corrsponding to each scenario is also shown and compared against standard SGD training scenario (without any noise). Here, we train a simple convolutional architecture (ConvNet: 10C-M-20C-M-320FC) of 2 Convolutional (C) layers with 10, 20 filters, each followed by 2×2 Max-pooling (M) and a Fully-Connected (FC) layer of size 320. We use mini-batch SGD with momentum of 0.5, learning rate (η=0.1) decayed by 0.1 every 15 epochs and batch-size 64 to learn the network parameters. We trained 3 ConvNet models independently corresponding to each scenario for 30 epochs. For the NoL scenarios, we conduct noise modelling with only negative loss

gradients ($\nabla\mathcal{L}_N \leq 0$) with noise learning rate, $\eta_{noise} = 0.001$, throughout the training process. Note, the noise image shown is the average across all 64 noise templates.



Scenario	Accuracy
SGD without noise	64.43%
NoL (all gradient)	59.94%
NoL (Positive gradient)	60.06%
NoL (Negative gradient)	63.45%

Fig. 8.9. Noise learnt for different training conditions with NoL on CIFAR10 dataset

In Fig. 8.9, we showcase the noise learnt by a simple convolutional network (ConvNet: 10C-M-20C-M-320FC), learning the CIFAR10 data with NoL (multiplicative noise) under different gradient update conditions. As with MNIST (Fig. 8.8), we observe that the noise learnt enhances the region of interest while deemphasizing the background pixels. Note, the noise in this case has RGB components as a result of which we see some prominent color blobs in the noise template after training. The performance table shows that using only negative gradients (i.e. $\nabla\mathcal{L}_N \leq 0$) during backpropagation for noise modelling yields minimal loss in accuracy as compared to a standard SGD trained model. We use mini-batch SGD with momentum of 0.9, weight decay 5e-4, learning rate (η=0.01) decayed by 0.2 every 10 epochs and batch-size 64 to learn the network parameters. We trained 4 ConvNet models independently corresponding to each scenario for 30 epochs. For the NoL scenarios, we conduct noise modelling by backpropagating the corresponding gradient with noise learning rate ($\eta_{noise} = 0.001$) throughout the training process. Note, the noise image shown is the average across all 64 noise templates.

8.5.2 PC variance for SGD and NoL scenarios in response to adversarial and clean inputs across different layers of ResNet18

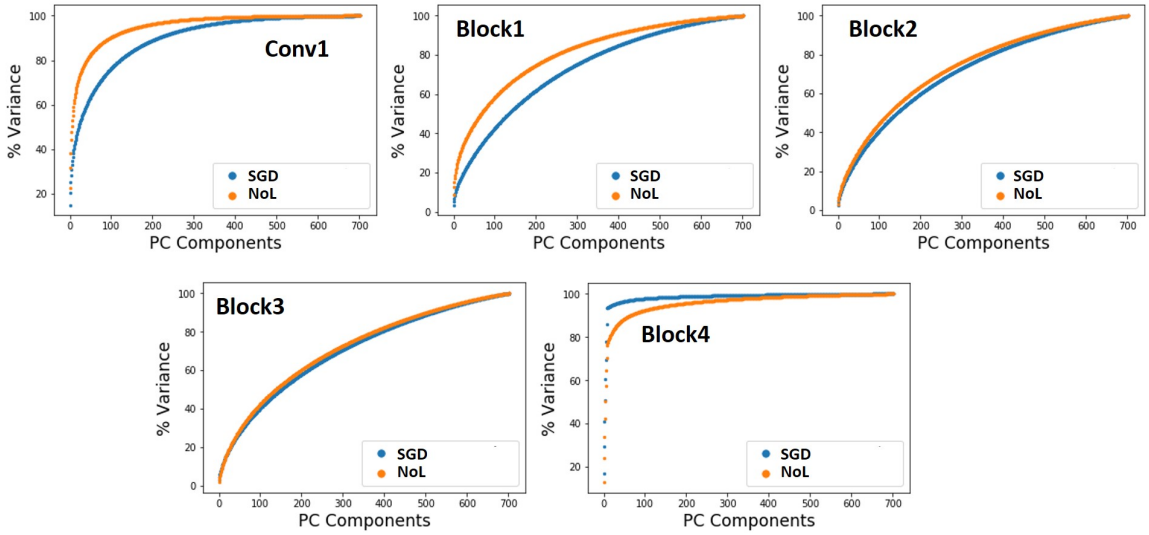


Fig. 8.10. Variance captured in PC dimensions of intermediate layers' activations (in response to clean data) of a ResNet18 model trained with NoL and SGD on CIFAR10 data

In Fig. 8.10, we show the variance captured in the leading Principal Component (PC) dimensions for the inital convolutional layer's ($Conv1$) and intermediate blocks learnt activations of a ResNet-18 model trained on CIFAR10 data. We compare the variance of the learnt representations (in response to clean inputs) for each block across two scenarios: SGD (without noise) and NoL (with noise). Note, we capture the variance of the final block's activations before average pooling. That is, the activations of $Block4$ have dimension $512\times4\times4$. We observe that NoL noise increases the variance along the high rank PCs. Also, as we go deeper into the network, the absolute difference of the variance values between SGD/NoL decreases. This is expected as the contribution of input noise on the overall representations decreases as we go deeper into the network. Moreover, there is a generic-to-specific transition in the hierarchy of learnt features of a deep neural network. Thus, the linear PC subspace analysis to

quantify a model's knowledge of the data manifold is more applicable in the earlier layers, since they learn more general input-related characteristics. Nonetheless, we see that NoL model yields widened variance than SGD for each intermediate layer except the final $Block4$ that feeds into the output layer. We use mini-batch SGD with momentum of 0.9, weight decay 5e-4, learning rate (η=0.1) decayed by 0.1 every 30 epochs and batch-size 64 to learn the network parameters. We trained 2 ResNet-18 models independently corresponding to each scenario for 60 epochs. For noise modelling, we use η_{noise} = 0.001 decayed by 0.1 every 30 epochs. Note, we used a sample set of 700 test images to conduct the PCA.

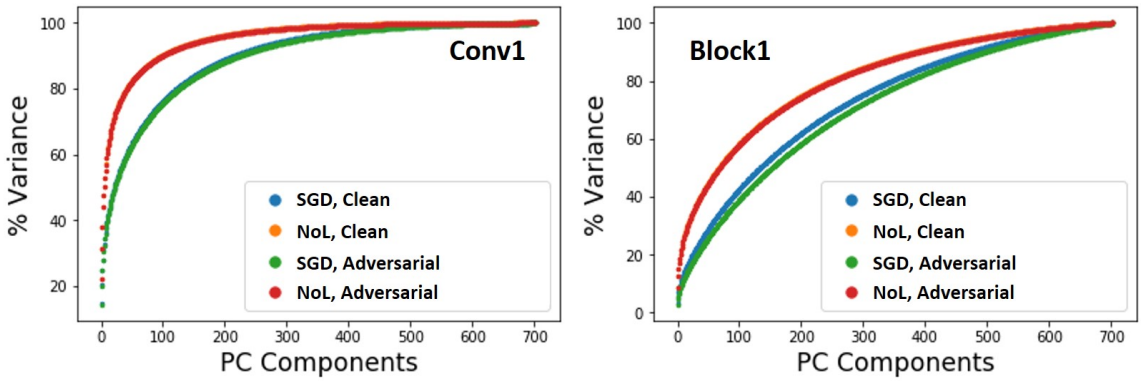


Fig. 8.11. Variance captured in the leading Principal Component (PC) dimensions for the $Conv1$ and $Block1$ learnt activations in response to both clean and adversarial inputs for ResNet-18 models corresponding to the scenarios discussed in Fig. 8.10

In Fig. 8.11, the model's variance for both clean and adversarial inputs are exactly same in case of NoL/SGD for $Conv1$ layers. For $Block1$, the adversarial input variance is slighlty lower in case of SGD than that of clean input. With NoL, the variance is still the same for $Block1$. This indicates that PC variance statistics cannot differentiate between a model's knowledge of on-/off- manifold data. It only tells us whether a model's underlying representation has acquired more knowledge about the data manifold. To analyze a model's understanding of adversarial data, we need to

look into the relationship between the clean and adversarial projection onto the PC subspace and measure the cosine distance. Note, we used the Fast Gradient Sign Method (FGSM) method [7] to create BB adversaries with a step size of 8/255, from another independently trained ResNet-18 model (*source*) with standard SGD. The *source* attack model has the same hyperparameters as the *SGD* model in Fig. 8.10 and is trained for 40 epochs.

8.5.3 Experimental Details and Model Description

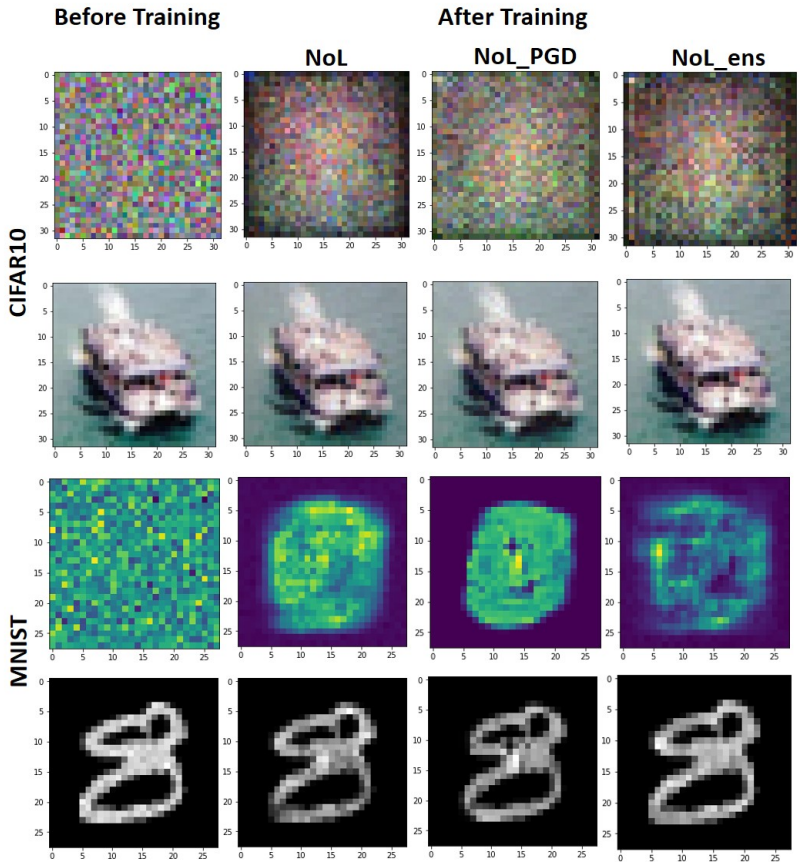


Fig. 8.12. Noise templates shown for different training scenarios with noise-enabled prior learning

In Fig. 8.12, we show the noise templates learnt with noise modeling corresponding to different training scenarios of Table 8.1, 8.2: NoL (only noise modeling), NoL_PGD (noise modeling with PGDAdv training NoL_{PGD}), NoL_ens (noise modeling with EnsAdv training NoL_{ens}) for MNIST and CIFAR10 data. A sample image ($X \times N$) before and after training with different scenarios is shown. The fact that every training technique yields different noise template shows that noise influences the overall optimization. Column 1 shows the noise template and correponding image ($X \times N$) before training, Coulmns 2-4 show the templates after training. Note, noise shown is the mean across 64 templates.

Table 8.3.
Hyperparameter Table for training ResNet18 models on CIFAR10 data

Model Type	Training Method	Epochs	η/η_{adv}	η, η_{adv} decay/step-size	$\eta_{noise}/\eta_{noise_{adv}}$	$\eta_{noise}, \eta_{noise_{adv}}$ decay/step-size	Test Accuracy in (%)
Target	SGD	120	0.1/–	0.1/30	–	–	88.8
	NoL	120	0.1/–	0.1/30	0.001/–	0.1/30	87.1
	SGD_{ens}	80	0.1/0.05	0.1/30	–	–	86.3
	NoL_{ens}	120	0.1/0.05	0.1/30	0.001/0.0005	0.1/30	86.4
	SGD_{PGD}	122	0.1/0.1	0.1/20	–	–	83.2
	NoL_{PGD}	122	0.1/0.1	0.1/20	0.001/0.0005	0.1/20	83
Source	SGD	300	0.1/–	0.1/100	–	–	89
	$PGDAdv$	122	0.1/0.1	0.1/20	–	–	83
EnsAdv	SGD	31	0.1/–	0.1/30	–	–	81

The Pytorch implementation of ResNet-18 architecture for CIFAR10 and ResNext-29 architecture for CIFAR100 were taken from [181]. For CIFAR10/CIFAR100, we use mini-batch SGD with momentum of 0.9, weight decay 5e-4 and batch size 64 for training the weight parameters of the models. A detailed description of the learning

rate and epochs for ResNet18 model (corresponding to Table 8.2) is shown in Table 8.3. Similarly, Table 8.4 shows the parameters for ResNext-29 model. The hyperparmeters corresponding to each scenario (of Table 8.3, 8.4) are shown in Rows1-6 under *Target* type. The hyperparameters for the source model used to attack the target models for BB scenarios is shown in Row 7/8 under *Source* type. We use BB attacks from the SGD trained source model to attack $SGD, NoL, NoL_{ens}, PGD_{ens}$. We use BB attacks from a model trained with PGD adversarial training ($\epsilon = 8/255$, *step-size=2/255 over 7 steps*) to craft strong BB attacks on SGD_{PGD}, NoL_{PGD}. The model used to generate black box adversaries to augment the training dataset of the SGD_{ens}, NoL_{ens} target models is shown in Row 9 under *EnsAdv* type.

Table 8.4.
Hyperparameter Table for training ResNext29 models on CIFAR100 data

Model Type	Training Method	Epochs	η/η_{adv}	η, η_{adv} decay/step-size	$\eta_{noise}/\eta_{noise_{adv}}$	$\eta_{noise}, \eta_{noise_{adv}}$ decay/step-size	Test Accuracy in (%)
Target	SGD	100	0.1/–	0.1/40	–	–	71
	NoL	58	0.1/–	0.1/20	0.001/–	0.1/20	69.4
	SGD_{ens}	42	0.1/0.05	0.1/20	–	–	69.8
	NoL_{ens}	48	0.1/0.05	0.1/20	0.001/0.0005	0.1/20	67.3
	SGD_{PGD}	52	0.1/0.05	0.1/20	–	–	71.6
	NoL_{PGD}	52	0.1/0.05	0.1/20	0.001/0.0005	0.1/20	69
Source	SGD	34	0.1/–	0.1/10	–	–	67.2
	$PGDAdv$	48	0.1/0.05	0.1/20	–	–	68.4
EnsAdv	SGD	45	0.1/–	0.1/20	–	–	71.3

How to conduct Ensemble Adversarial Training? Furthermore, in all our experiments, for EnsAdv training (SGD_{ens}), we use a slightly different approach than [10]. Instead of using a weighted loss function that controls the relative weight of adversarial/clean examples in the overall loss computation, we use a different learn-

ing rate η_{adv}/η ($\eta_{adv} < \eta$) when training with adversarial/clean inputs, respectively, to learn the network parameters. Accordingly, while performing adversarial training with noise-based learning (NoL_{ens}), the noise modeling learning rate in addition to overall learning rate, η_{adv}/η, for adversarial/clean inputs is also different, $\eta_{noise_{adv}}/\eta_{noise}$ ($\eta_{noise_{adv}} < \eta_{noise}$).

Table 8.5.
Hyperparameter Table for training ConvNet1/ConvNet2 models on MNIST data

Model Type	Training Method	Epochs	η/η_{adv}	η, η_{adv} decay/step-size	$\eta_{noise}/\eta_{noise_{adv}}$	$\eta_{noise}, \eta_{noise_{adv}}$ decay/step-size	Test Accuracy in (%)
Target ConvNet1	SGD	100	0.01/–	0.1/50	–	–	99.1
	NoL	150	0.01/–	0.1/50	0.001/–	0.1/50	99.2
	SGD_{ens}	64	0.01/0.005	0.1/30	–	–	99
	NoL_{ens}	32	0.01/0.005	0.1/30	0.001/3.3e-5	0.1/30	99.1
	SGD_{PGD}	142	0.01/0.01	0.1/30	–	–	97.9
	NoL_{PGD}	162	0.01/0.01	0.1/30	1e-4/1e-5	0.1/30	98
Source (ConvNet2)	SGD	15	0.01/–	–/–	–	–	98.6
	$PGDAdv$	128	0.01/0.01	0.1/30	–	–	97
EnsAdv ConvNet1	SGD	15	0.01/–	–/–	–	–	98.8

How to conduct PGD Adversarial Training? For PGD adversarial training (SGD_{PGD}), we used the techniques suggested in [182]. [182] propose that training on a mixture of clean and adversarial examples (generated using PGD attack), instead of literally solving the min-max problem described by [167] yields better performance. In fact, this helps maintain good accuracy on both clean and adversarial examples. Like EnsAdv training, here as well, we use a different learning rate η_{adv}/η ($\eta_{adv} < \eta$) when training with adversarial/clean inputs, respectively, to learn the network

parameters. Accordingly, while performing PGD adversarial training with noise-based learning (NoL_{PGD}), the noise modeling learning rate in addition to overall learning rate, η_{adv}/η, for adversarial/clean inputs is also different, $\eta_{noise_{adv}}/\eta_{noise}$ ($\eta_{noise_{adv}} < \eta_{noise}$)

Note, the adversarial inputs for EnsAdv training of a target model are created using BB adversaries generated by standard FGSM from a source (shown in Row 9 of Table 8.3, 8.4), while PGDAdv training uses WB adversaries created with PGD attack from the same target model. We also show the test accuracy (on clean data) for each model in Table 8.3,8.4 for reference. Note, the learning rate in each case decays by a factor of 0.1 every 20/30 epochs (Column 5 in Table 8.3, 8.4).

Table 8.6.
Hyperparameter Table for training ResNet18 models on CIFAR10 data for different types of noise modeling ($X + N, X \times N$) with all/ only negative gradient $\nabla\mathcal{L}_N$

Noise Modeling Type	Gradient $\nabla\mathcal{L}_N$	Epochs	η	η decay/step-size	η_{noise}	η_{noise} decay/step-size	Test Accuracy in (%)
$X + N$	Negative	120	0.1	0.1/30	0.001	0.1/30	78.1
$X + N$	All	120	0.1	0.1/30	0.001	0.1/30	77.1
$X \times N$	Negative	120	0.1	0.1/30	0.001	0.1/30	87.1
$X \times N$	All	120	0.1	0.1/30	0.001	0.1/30	85.1
SGD	-	120	0.1	0.1/30	-	-	88.9

For MNIST, we use 2 different architectures as source/ target models. ConvNet1: 32C-M-64C-M-1024FC is the model used as target. ConvNet2: 10C-M-20C-M-320FC is the model used as source. Here, we use mini-batch SGD with momentum of 0.5, batch size 64, for training the weight parameters. Table 8.5 shows the hyperparameters used to train the models in Table 8.1. The notations here are similar to that of

Table 8.3. Note, the source model trained with PGDAdv training to craft BB attacks on NoL_{PGD}, SGD_{PGD} was trained with $\epsilon = 0.3$, *step-size=0.01 over 40 steps.*

Model Description for Fig. 8.2

We use mini-batch SGD with momentum of 0.9, weight decay 5e-4 and batch size 64 for training the weight parameters of the models in Table 8.6.

9. SUMMARY & FUTURE WORK

Neural networks are being increasingly deployed across the entire computing spectrum from data centers to mobile devices to drive artificial intelligence (AI) applications. AI is undoubtedly disrupting society as it is becoming ubiquitous- sometimes to the extent that AI models replace humans and take decision autonomously- in domains like healthcare, transportation or education among others. However, the vulnerability of AI models particularly to adversarial attacks poses a threat on the integration of AI for critical applications (that can cause fatal failures). Further, the advent of IoT (Internet-of-Things) has created the need to embed on-chip intelligence in mobiles, wearables and related devices. The limited resources and power budget of IoT devices, however, prohibits the deployment of large neural networks (that are both power hungry and memory intensive). Thus, democratization of AI in the future presses the need for accurate, energy-efficient and robust realization of neural networks.

In this thesis, we presented some design techniques spanning both conventional deep learning networks (DLNs) and biologically-inspired spiking neural networks (SNNs). SNNs naturally offer more energy-efficient computations due to the inherent sparse, event-driven nature of neuronal processing and communication. However, SNNs lack appropriate training techniques that can push their applicability towards a large range of AI systems. To that effect, we described unsupervised/supervised learning rules for training deep SNNs and recurrent SNNs that yields state-of-the-art results with computationally efficient learning (chapter 4, 5, 6). Additionally, we proposed several complementary approaches focusing on ensemble implementations of SNNs using the principle of 'divide-and-conquer' [183,184]. Here, the main idea is to divide the task into simpler problems and learn simple networks on them individually. Finally, during evaluation, we combine the decision of each network. This reduces

the training complexity of an SNN and also allows the individual networks to learn better, discriminative features about the input causing improved accuracy [185].

On the deep learning end, DLNs have enabled a wide-range of AI applications, but incur high compute and memory costs. To reduce the inference cost for DLNs, we proposed techniques exploiting the inherent variability in input data to come up with an early-classification strategy. Here, easy inputs are classified at early layers and hard inputs at latter layers, without any impact of output accuracy (chapter 2). In fact, we used the consensus in the characteristic features (color/texture) across images in a dataset, as discussed in chapter 3, to decompose the original classification problem and construct a tree of classifiers (nodes) with a generic-to-specific transition in the classification hierarchy [69]. The initial nodes of the tree separate the instances based on feature information and selectively enable the latter nodes to perform object specific classification. The proposed FALCON methodology [69] allows selective activation of only those branches and nodes of the classification tree that are relevant to the input while keeping the remaining nodes idle, resulting in energy-efficiency. FALCON was later adapted to more generic tree-based DLN implementations [186].

Finally, we explored different quantization themes to analyse the robustness of low-precision DLNs against adversarial attacks (chapter 7). We also proposed a new noise-based learning technique that imparts intrinsic adversarial resiliency to DLNs (chapter 7). A significant contribution of chapter 7 is the PCA variance and cosine distance analysis that provides a novel perspective to visualize and quantify a model's response to adversarial/clean data. We also conducted a comprehensive analysis on the adversarial robustness of SNNs [18]. Our analysis reveals that SNN robustness is largely dependent on the corresponding training mechanism. We observe that SNNs trained by spike-based backpropagation are more adversarially robust than the ones obtained by ANN-to-SNN conversion rules in several whitebox and blackbox scenarios. In [18], we also proposed a simple, yet, effective framework for crafting adversarial attacks from SNNs. This method of crafting adversaries can potentially

be used in the future to craft stronger attacks and measure the adversarial resiliency of SNNs, in comparison to DLNs.

Future Work Conditional Deep Learning (chapter 2) offers useful insights into the computationally efficient characteristics of a DLN's inference. In the future, it is worth investigating the applicability of CDL or the early termination strategy towards implementing distributed edge-cloud intelligence. One way of doing this can be a hybrid precision DLN with low-precision weights/activations in the early layers predicting easy or simple inputs and higher-precision weights/activations in the latter layers for the harder inputs. Scaling the precision in a principled manner will result in a computationally efficient DLN architecture without accuracy loss. Thus, CDL strategy with hybrid precision scaling can lead to edge-cloud partitioned implementation of a DLN with partitioned inference. Similarly, CDL can be combined with reinforcement learning systems or deep Q-networks (DQNs) to implement efficient drone-navigation systems. Intelligence of the drone, concerning training and inference of the DQN, are generally deployed on cloud servers. However, the continuous on-/off-loading of data between the cloud and the drone leads to increased latency in processing time per frame. CDL can enable efficient distributed autonomous intelligence (with early layers and exit branches implemented on the device and latter layers on cloud). With properly trained exit branches, we can expect fast and accurate decision making on the drone itself.

Similarly, the insights developed for Adaptive Synaptic Plasticity (chapter 5) can be extended towards implementing a deep SNN with better generalization and transfer learning capability. The spike-based learning strategies for reservoirs and auto-encoder based learning in deep SNNs (chapter 6, 4) can be refined further to improve accuracy on image recognition tasks as well as, extend to beyond-vision applications [187, 188].

Finally, as we move towards an AI-driven world, there will be a growing need for a formal definition of the notion of robustness and interpretability of learning systems. Our recent work's analysis on the stability of training of reservoirs for a sequential task

is an initial work in this direction [189]. We formalized the training convergence not just as a measure of accuracy, but in terms of eigenvalue spectra (which is, shrinking of spectral circle as training progresses). Similarly, for robustness, a strong and well-motivated definition for adversarial attacks (similar to principal component analysis shown in chapter 8) can directly facilitate research in the formal verification of neural networks.

To conclude, let me re-iterate the question (by Alan Turing) from the abstract of the thesis, 'Can machines think?'. This question presents several opportunities for research across the computing stack from algorithm-to-hardware. Today, machine intelligence is broadly pursued by the deep learning and the neuromorphic computing community in the design space of energy-accuracy trade-off. Despite the spectacular advances in both algorithm and hardware to enable ubiquitous AI, the community's understanding of adversarial robustness still is at its infancy. Going forward, there is a need to explore the energy-accuracy-robustness trade-off cohesively with algorithm-hardware co-design to create truly functional cognitive systems.

REFERENCES

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[3] T. Delbrück, B. Linares-Barranco, E. Culurciello, and C. Posch, "Activity-driven, event-based vision sensors," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 2426–2429.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[8] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[9] S. M. Moosavi Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, no. EPFL-CONF-218057, 2016.

[10] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[11] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[12] P. Laskov *et al.*, "Practical evasion of a learning-based classifier: A case study," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 197–211.

[13] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in *Proceedings of the 2016 Network and Distributed Systems Symposium*, 2016.

[14] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[15] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017.

[16] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *arXiv preprint arXiv:1702.02284*, 2017.

[17] V. Behzadan and A. Munir, "Vulnerability of deep reinforcement learning to policy induction attacks," in *International Conference on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2017, pp. 262–275.

[18] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran, and K. Roy, "A comprehensive analysis on adversarial robustness of spiking neural networks," *Accepted in International Joint Conference on Neural Networks (IJCNN), arXiv preprint arXiv:1905.02704*, 2019.

[19] P. Panda, A. Sengupta, and K. Roy, "Energy-efficient and improved image recognition with conditional deep learning," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 33, 2017.

[20] ——, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 2016, pp. 475–480.

[21] P. Panda, S. Venkataramani, A. Sengupta, A. Raghunathan, and K. Roy, "Energy-efficient object detection using semantic decomposition," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2673–2677, 2017.

[22] P. Panda and K. Roy, "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 299–306.

[23] P. Panda, J. M. Allred, S. Ramanathan, and K. Roy, "Asp: Learning to forget with adaptive synaptic plasticity in spiking neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[24] F. Zuo*, P. Panda*, M. Kotiuga, J. Li, M. Kang, C. Mazzoli, H. Zhou, A. Barbour, S. Wilkins, B. Narayanan *et al.*, "Habituation based synaptic plasticity and organismic learning in a quantum perovskite," *Nature Communications*, vol. 8, 2017, *Equal Author Contribution.

[25] P. Panda and K. Roy, "Learning to generate sequences with combination of hebbian and non-hebbian plasticity in recurrent spiking neural networks," *Frontiers in neuroscience*, vol. 11, p. 693, 2017.

[26] P. Panda, I. Chakraborty, and K. Roy, "Discretization based solutions for secure machine learning against adversarial attacks," *arXiv preprint arXiv:1902.03151*, 2019.

[27] P. Panda and K. Roy, "Implicit generative modeling of random noise during training for adversarial robustness," *arXiv preprint arXiv:1807.02188*, 2018.

[28] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[29] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2. Granada, Spain, 2011, p. 5.

[31] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

[32] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[34] S. G. Ramasubramanian, R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "Spindle: Spintronic deep learning engine for large-scale neuromorphic computing," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 15–20.

[35] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 67.

[36] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2528–2535.

[37] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, 2014, pp. 3320–3328.

[38] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE, 2014, pp. 512–519.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv preprint arXiv:1502.01852*, 2015.

[41] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[42] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[43] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[44] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[45] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[46] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[47] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[48] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[49] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–97.

[50] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[51] L. Hansen, "Tiny imagenet challenge submission," *CS 231N*, 2015.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[53] R. B. Palm, "Prediction as a candidate for learning deep hierarchical models of data," *Technical University of Denmark*, 2012.

[54] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[55] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *arXiv preprint arXiv:1211.5063*, 2012.

[56] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2146–2153.

[57] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[58] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," 2005.

[59] S. Venkataramani, V. Bahl, X.-S. Hua, J. Liu, J. Li, M. Phillipose, B. Priyantha, and M. Shoaib, "Sapphire: An always-on context-aware computer vision system for portable devices," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 2015, pp. 1491–1496.

[60] J. Hosang, R. Benenson, P. Dollár, and B. Schiele, "What makes for effective detection proposals?" 2015.

[61] L. Deng and J. C. Platt, "Ensemble deep learning for speech recognition." in *INTERSPEECH*, 2014, pp. 1915–1919.

[62] Y. Sun, X. Wang, and X. Tang, "Deep convolutional network cascade for facial point detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3476–3483.

[63] H. Levkowitz and G. T. Herman, "Glhs: A generalized lightness, hue, and saturation color model," *CVGIP: Graphical Models and Image Processing*, vol. 55, no. 4, pp. 271–285, 1993.

[64] A. K. Jain, N. K. Ratha, and S. Lakshmanan, "Object detection using gabor filters," *Pattern Recognition*, vol. 30, pp. 295–309, 1997.

[65] S. Arivazhagan and L. Ganesan, "Texture classification using wavelet transform," *Pattern recognition letters*, vol. 24, no. 9, pp. 1513–1521, 2003.

[66] M. Haghighat, S. Zonouz, and M. Abdel-Mottaleb, "Identification using encrypted biometrics," in *Computer Analysis of Images and Patterns*. Springer, 2013, pp. 440–448.

[67] G.-H. Hu, "Optimal ring gabor filter design for texture defect detection using a simulated annealing algorithm," in *Information Science, Electronics and Electrical Engineering (ISEEE), 2014 International Conference on*, vol. 2. IEEE, 2014, pp. 860–864.

[68] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, 2010.

[69] P. Panda, A. Ankit, P. Wijesinghe, and K. Roy, "Falcon: Feature driven selective classification for energy-efficient image recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[70] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. Merolla, K. Boahen *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[71] J. Park, S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs, "A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver," in *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*. IEEE, 2014, pp. 675–678.

[72] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[73] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in neuroscience*, vol. 7, 2013.

[74] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of computer and system sciences*, vol. 69, no. 4, pp. 593–616, 2004.

[75] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 11, pp. 2706–2719, 2013.

[76] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[77] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks," *Communications of the ACM*, vol. 54, no. 10, pp. 95–103, 2011.

[78] M. A. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[79] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[80] P. U. Diehl, M. Cook, M. Tatsuno, and S. Song, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity." *Frontiers in Computational Neuroscience*, 2015.

[81] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *The Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[82] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[83] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[84] A. van Schaik, "Building blocks for electronic spiking neural networks," *Neural networks*, vol. 14, no. 6, pp. 617–628, 2001.

[85] N. Anwani and B. Rajendran, "Normad-normalized approximate descent based supervised learning rule for spiking neurons," in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–8.

[86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, p. 3, 1988.

[87] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010*. Springer, 2010, pp. 92–101.

[88] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[89] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[90] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," *arXiv preprint arXiv:1412.0035*, 2014.

[91] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[92] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–8.

[93] N. Madroñal, J. M. Delgado-García, A. Fernández-Guizán, J. Chatterjee, M. Köhn, C. Mattucci, A. Jain, T. Tsetsenis, A. Illarionova, V. Grinevich *et al.*, "Rapid erasure of hippocampal memory following inhibition of dentate gyrus granule cells," *Nature communications*, vol. 7, 2016.

[94] R. M. French, "Semi-distributed representations and catastrophic forgetting in connectionist networks," *Connection Science*, vol. 4, no. 3-4, pp. 365–377, 1992.

[95] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[96] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[97] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[98] S. Martin, P. Grimwood, and R. Morris, "Synaptic plasticity and memory: an evaluation of the hypothesis," *Annual review of neuroscience*, vol. 23, no. 1, pp. 649–711, 2000.

[99] S. Grossberg, "How does a brain build a cognitive code?" in *Studies of mind and brain*. Springer, 1982, pp. 1–52.

[100] G. A. Carpenter and S. Grossberg, "The art of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol. 21, no. 3, pp. 77–88, 1988.

[101] O. Hardt, K. Nader, and Y.-T. Wang, "Glua2-dependent ampa receptor endocytosis and the decay of early and late long-term potentiation: possible mechanisms for forgetting of short-and long-term memories," *Phil. Trans. R. Soc. B*, vol. 369, no. 1633, p. 20130141, 2014.

[102] D. M. Villarreal, V. Do, E. Haddad, and B. E. Derrick, "Nmda receptor antagonists sustain ltp and spatial memory: active processes mediate ltp decay," *Nature neuroscience*, vol. 5, no. 1, pp. 48–52, 2002.

[103] P. W. Frankland, S. Köhler, and S. A. Josselyn, "Hippocampal neurogenesis and forgetting," *Trends in neurosciences*, vol. 36, no. 9, pp. 497–503, 2013.

[104] W. Zhang and D. J. Linden, "The other side of the engram: experience-driven changes in neuronal intrinsic excitability," *Nature Reviews Neuroscience*, vol. 4, no. 11, pp. 885–900, 2003.

[105] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, 2013.

[106] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[107] T. Masquelier, R. Guyonneau, and S. J. Thorpe, "Competitive stdp-based spike pattern learning," *Neural computation*, vol. 21, no. 5, pp. 1259–1276, 2009.

[108] L. Abbott and S. Song, "Temporally asymmetric hebbian learning, spike timing and neural response variability," *Advances in neural information processing systems*, pp. 69–75, 1999.

[109] M. Muhlbaier, A. Topalis, and R. Polikar, "Incremental learning from unbalanced data," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, vol. 2. IEEE, 2004, pp. 1057–1062.

[110] A. Morrison, A. Aertsen, and M. Diesmann, "Spike-timing-dependent plasticity in balanced random networks," *Neural computation*, vol. 19, no. 6, pp. 1437–1467, 2007.

[111] R. Morris, "Do hebb: The organization of behavior, wiley: New york; 1949," *Brain research bulletin*, vol. 50, no. 5, p. 437, 1999.

[112] D. Goodman and R. Brette, "Brian: a simulator for spiking neural networks in python," 2008.

[113] S. Basu, M. Karki, S. Ganguly, R. DiBiano, S. Mukhopadhyay, S. Gayaka, R. Kannan, and R. Nemani, "Learning sparse feature representations using probabilistic quadtrees and deep belief nets," *Neural Processing Letters*, pp. 1–13, 2015.

[114] R. Laje and D. V. Buonomano, "Robust timing and motor patterns by taming chaos in recurrent neural networks," *Nature neuroscience*, vol. 16, no. 7, pp. 925–933, 2013.

[115] F. Zenke, E. J. Agnes, and W. Gerstner, "Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks," *Nature communications*, vol. 6, 2015.

[116] S. Klampfl and W. Maass, "Emergence of dynamic memory traces in cortical microcircuit models through stdp," *Journal of Neuroscience*, vol. 33, no. 28, pp. 11 515–11 529, 2013.

[117] K. Rajan, *Spontaneous and stimulus-driven network dynamics*. Columbia University, 2009.

[118] K. Rajan, L. Abbott, and H. Sompolinsky, "Stimulus-dependent suppression of chaos in recurrent neural networks," *Physical Review E*, vol. 82, no. 1, p. 011903, 2010.

[119] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, p. 201604850, 2016.

[120] S. R. Kheradpisheh, M. Ganjtabesh, and T. Masquelier, "Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition," *Neurocomputing*, vol. 205, pp. 382–392, 2016.

[121] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput Biol*, vol. 3, no. 2, p. e31, 2007.

[122] W. Maass, "Liquid state machines: motivation, theory, and applications," *Computability in context: computation and logic in the real world*, pp. 275–296, 2010.

[123] N. Srinivasa and Y. Cho, "Unsupervised discrimination of patterns in spiking neural networks with excitatory and inhibitory synaptic plasticity," *Frontiers in computational neuroscience*, vol. 8, p. 159, 2014.

[124] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[125] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[126] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[127] D. Sussillo and L. F. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, no. 4, pp. 544–557, 2009.

[128] P. U. Diehl and M. Cook, "Learning and inferring relations in cortical networks," *arXiv preprint arXiv:1608.08267*, 2016.

[129] M. Wehr and A. M. Zador, "Balanced inhibition underlies tuning and sharpens spike timing in auditory cortex," *Nature*, vol. 426, no. 6965, pp. 442–446, 2003.

[130] R. Legenstein and W. Maass, "Edge of chaos and prediction of computational performance for neural circuit models," *Neural Networks*, vol. 20, no. 3, pp. 323–334, 2007.

[131] C. Van Vreeswijk, H. Sompolinsky *et al.*, "Chaos in neuronal networks with balanced excitatory and inhibitory activity," *Science*, vol. 274, no. 5293, pp. 1724–1726, 1996.

[132] J.-Y. Chen, P. Lonjers, C. Lee, M. Chistiakova, M. Volgushev, and M. Bazhenov, "Heterosynaptic plasticity prevents runaway synaptic dynamics," *Journal of Neuroscience*, vol. 33, no. 40, pp. 15 915–15 929, 2013.

[133] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity," *Journal of Neuroscience*, vol. 26, no. 38, pp. 9673–9682, 2006.

[134] T. E. de Campos, B. R. Babu, and M. Varma, "Character recognition in natural images." in *VISAPP (2)*, 2009, pp. 273–280.

[135] K. Rajan and L. Abbott, "Eigenvalue spectra of random matrices for neural networks," *Physical review letters*, vol. 97, no. 18, p. 188104, 2006.

[136] V. L. Girko, "Circular law," *Theory of Probability & Its Applications*, vol. 29, no. 4, pp. 694–706, 1985.

[137] A. Lazar, G. Pipa, and J. Triesch, "Sorn: a self-organizing recurrent neural network," *Frontiers in computational neuroscience*, vol. 3, 2009.

[138] J. Thiele, P. Diehl, and M. Cook, "A wake-sleep algorithm for recurrent, spiking neural networks," *arXiv preprint arXiv:1703.06290*, 2017.

[139] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[140] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[141] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[142] C. Guo, M. Rana, M. Cisse, and L. van der Maaten, "Countering adversarial images using input transformations," in *International Conference on Learning Representations*, 2018.

[143] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," in *International Conference on Learning Representations*, 2018.

[144] J. Chen, X. Wu, Y. Liang, and S. Jha, "Improving adversarial robustness by data-specific discretization," *arXiv preprint arXiv:1805.07816*, 2018.

[145] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[146] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," in *International Conference on Learning Representations*, 2019.

[147] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[148] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 and cifar-100 datasets," *URl: https://www. cs. toronto. edu/kriz/cifar. html (vi sited on Mar. 1, 2016)*, 2009.

[149] A. Galloway, G. W. Taylor, and M. Moussa, "Attacking binarized neural networks," in *International Conference on Learning Representations*, 2018.

[150] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *International Conference on Learning Representations*, 2018.

[151] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *International Conference on Learning Representations*, 2016.

[152] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[153] https://github.com/itayhubara/BinaryNet.pytorch.

[154] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[155] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* Ieee, 2009, pp. 248–255.

[156] https://github.com/jiecaoyu/XNOR-Net-PyTorch.

[157] https://evadeML.org/zoo.

[158] https://github.com/AngusG/cleverhans-attacking-bnns.

[159] N. Papernot, I. Goodfellow, R. Sheatsley, R. Feinman, and P. McDaniel, "cleverhans v1.0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2016.

[160] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on.* IEEE, 2017, pp. 39–57.

[161] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 506–519.

[162] D. Krotov and J. J. Hopfield, "Dense associative memory is robust to adversarial inputs," *arXiv preprint arXiv:1701.00939*, 2017.

[163] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 582–597.

[164] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, "Parseval networks: Improving robustness to adversarial examples," in *International Conference on Machine Learning*, 2017, pp. 854–863.

[165] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[166] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[167] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[168] Y. Sharma and P.-Y. Chen, "Breaking the madry defense model with l_1-based adversarial examples," *arXiv preprint arXiv:1710.10733*, 2017.

[169] Y. Lou, X. Boix, G. Roig, T. Poggio, and Q. Zhao, "Foveation-based mechanisms alleviate adversarial examples," Center for Brains, Minds and Machines (CBMM), arXiv, Tech. Rep., 2016.

[170] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning.(2016)," *Book in preparation for MIT Press. URL: http://www. deeplearningbook. org*, 2016.

[171] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "Pixeldefend: Leveraging generative models to understand and defend against adversarial examples," *arXiv preprint arXiv:1710.10766*, 2017.

[172] H. Lee, S. Han, and J. Lee, "Generative adversarial trainer: Defense to adversarial perturbations with gan," *arXiv preprint arXiv:1705.03387*, 2017.

[173] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, "Adversarial spheres," *arXiv preprint arXiv:1801.02774*, 2018.

[174] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "The space of transferable adversarial examples," *arXiv preprint arXiv:1704.03453*, 2017.

[175] H. Noh, T. You, J. Mun, and B. Han, "Regularizing deep neural networks by noise: Its interpretation and optimization," in *Advances in Neural Information Processing Systems*, 2017, pp. 5109–5118.

[176] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "Robustness of classifiers: from adversarial to random noise," in *Advances in Neural Information Processing Systems*, 2016, pp. 1632–1640.

[177] G. Hinton, "https://www.youtube.com/watch?v=LN0xtUuJsEI," *Neural Networks for Machine Learning, Coursera, (Lecture 9.3)*.

[178] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," in *International Conference on Learning Representations*, vol. 9, 2018.

[179] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," 2017.

[180] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 2017, pp. 5987–5995.

[181] Github, "https://github.com/kuangliu/pytorch-cifar/tree/master/models."

[182] H. Kannan, A. Kurakin, and I. Goodfellow, "Adversarial logit pairing," *arXiv preprint arXiv:1803.06373*, 2018.

[183] P. Panda, G. Srinivasan, and K. Roy, "Ensemblesnn: Distributed assistive stdp learning for energy-efficient recognition in spiking neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2629–2635.

[184] G. Srinivasan, P. Panda, and K. Roy, "Spilinc: Spiking liquid-ensemble computing for unsupervised speech and image recognition," *Frontiers in neuroscience*, vol. 12, 2018.

[185] P. Wijesinghe, G. Srinivasan, P. Panda, and K. Roy, "Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines," *Frontiers in Neuroscience*, vol. 13, p. 504, 2019.

[186] D. Roy, P. Panda, and K. Roy, "Tree-cnn: a hierarchical deep convolutional neural network for incremental learning," *arXiv preprint arXiv:1802.05800*, 2018.

[187] P. Panda and N. Srinivasa, "Learning to recognize actions from limited training examples using a recurrent spiking neural model," *Frontiers in neuroscience*, vol. 12, p. 126, 2018.

[188] D. Roy, P. Panda, and K. Roy, "Learning spatio-temporal representations using spike-based backpropagation," *openreview.net*, 2018.

[189] P. Panda, E. Soufleri, and K. Roy, "A comprehensive analysis on adversarial robustness of spiking neural networks," *Accepted in International Joint Conference on Neural Networks (IJCNN), arXiv preprint arXiv:1905.03219*, 2019.