Hierarchical Representation Learning with Connectionist Models

by

DE WANG

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

MAY 2018

Copyright © by De Wang 2018

All Rights Reserved

I dedicate this work to my dear parents and brother.

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my advisor Dr. Heng Huang for all the supports and helps all the way along past 6 years, both in academic and in life. The dissertation would not be possible without the invaluable guidance and support by Dr. Heng Huang. I joined the Ph.D. program right after I got my Bachelor in China, it is Dr. Heng Huang who guide me into the field of machine learning. I enjoyed discuss research with him a lot, it is my fortune to have the opportunity working with him during the last few years.

I would also like to thank my committee members. I am grateful to Dr. Chris Ding for discussing problems with me. His rigorousness towards academic research influenced me a lot. I would also like to thank Dr. Junzhou Huang for serving as my committee member. I have learned a lot from Dr. Junzhou Huang's classes. I also want to express my thankfulness to Dr. Jeff Lei. Thank you for all the time devoted to serve on my committee, and introducing me to the work of using machine learning for software testing.

I would also like to thank all my friends during my PhD study. I really enjoyed discussing problems with those talented minds, Dr. Xiantong Zhen, Dr. Xiaojun Chang, Dr. Hongyi Xin, Dr. Feng Zheng, Dr. Hong Chen, Dr. Jingbo Xia, Dr. Lei Luo, Dr. Feiping Nie, Dr. Xiao Cai, Dr. Miao Zhang, Xiaoqian Wang, Zhouyuan Huo, Hongchang Gao, Guodong Liu, Xin Miao, Xianghua Wu, Chaochao Yan, Jie Xu, Kamran Ghasedi, Amirhossein Herandi. You guys make my life more colorful, those time will be in my memories for my whole lifetime.

My gratefulness also goes to my hosts and friends during my internship and work. Thanks Dr. Jack Stokes at Microsoft Research offer me the position as a research intern in the security group, where he introduced me to the fantastic world of malware detection, attacks and defenses. Thanks Dr. Carlos Guestrin and Sethu Raman for offering me the intern position at Turi, and Haijie Gu for being my mentor during the wonderful three months at Seattle. Thanks Dr. Matthew Zeiler for offering me a internship position at Clarifai, and discuss with me on cutting edge industry research problems. Thanks Dr. Po-sen Huang and Dr. David Eigen at Clarifai for being my mentor, and Jake Zhao, Robert Wen, Michal Wolski for all the wonderful memories in New York City and MIT Hackthon event. Thanks my manager Etienne Jacques and Guillaume Binet at Argo AI for all the guidance on my work, and the talented scientists/engineers in the detection team at Pittsburgh, Jingdan Zhang, Kyoungmin Lee, Nicholas Meyer, Lawrence Jackson, Josh Manela.

Last but not least, I would like to express my deepest gratitude to the most important persons in my life: my parents Hanyuan Wang, Zhenfang Tian, and my elder brother Jinliang Wang and my sister in law Yan Liu. You take all the heavy lifting work, raised me up and support me through my 20 years of study, even though the economic hardship. Without the support from them, I can never get this far in my life.

February 28, 2018

ABSTRACT

Hierarchical Representation Learning with Connectionist Models

De Wang, Ph.D.

The University of Texas at Arlington, 2018

Supervising Professor: Dr. Heng Huang

To unleash the power of big data, efficient algorithms which are scalable to millions of data are desired. Deep learning is one area that benefits from big data enormously. Deep learning uses neural networks to mimic human brains, this approach is termed connectionist in AI community. In this dissertation, we propose several novel learning strategies to improve the performance of connectionist models.

Evaluation of a large neural network during inference phase requires a lot of GPU memory and computation, which will degrade user experience due to response latency. Model distillation is one way to distill the knowledge contained in one cumbersome model to a smaller one, which imitates the way that human learning is guided by teachers. We propose darker knowledge: a new method of knowledge distillation via rich targets regression. The proposed method outperforms current state-of-the-art model distillation methods proposed by Hinton et. al. A lot of high level machine learning tasks depend on model distillation, such as knowledge transfer between different neural network architectures, black box attack and defense in computer security, policy distillation in reinforcement learning, etc. Those tasks would benefit a lot from the improved model distillation method.

In another work, we design a new deep neural network architecture, which enables model ensemble in a single network. The network is composed of many columns, where each column is a small computational graph that performs a series of non-linear transformation. We train multi-column branching neural networks by stochastically dropping off many columns to prevent co-adaption of columns from overfitting, and promote each column to learn different features which will enhance the aggregated representation. The new architecture exhibits ensemble property in one single model and improves the classification performance of a single neural network upon current state-of-the-art architecture.

On the other hand, we studied the vulnerability of modern deep learning systems, both at the training stage and evaluation stage. At the training stage, it is possible that the training data can be contaminated by attackers with noise. This is deteriorate the recognition performance of deep learning models. We propose a new loss function that is more robust to noise input, and outperforms standard practice of neural network training. At the evaluation stage, we show that even though neural networks can achieve unprecedented high recognition accuracy on image recognition tasks, but the models are vulnerable to access attacks where attackers can generate fake identity proof easily by exploiting deployed neural networks. We show that what neural networks learn is very different from humans vision system. Given a trained model, we can easily generate an image that will be classified into a target class with almost 100% confidence, while the image might even look like white noise to human eyes.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv
ABSTRACT vi
LIST OF ILLUSTRATIONS x
LIST OF TABLES xii
Chapter Page
1. Introduction 1
2. Darker Knowledge 8
2.1 Introduction 8
2.2 Related Works 11
2.3 Darker Knowledge 15
2.4 Experimental Results 18
2.4.1 Preliminary Experiments on MNIST Dataset 19
2.4.2 CIFAR10 Dataset 20
2.4.3 Knowledge Transfer across Datasets 24
2.5 Future Works 25
2.6 Conclusion 25
3. Stochastic Columnar Network 29
3.1 Introduction 29
3.2 Stochastic Columnar Network 31
3.2.1 DropColumn 35
3.2.2 Relationship with Dropout 37
3.3 Experimental Results 38

3.3.1 Effect of Number of Columns 40
3.3.2 Effect of DropColumn Probability 41
3.3.3 Classification Performance 43
3.3.4 Ensemble Property 44
3.4 Conclusion 45
4. Attacks and Defenses of Deep Learning Models 50
4.1 Introduction 50
4.2 Preliminary 53
4.3 Training Attack 55
4.3.1 Method: Deep Modal Regression 56
4.3.2 Experimental Results 58
4.4 Evaluation Attack: Malware 61
4.4.1 Related Work 61
4.4.2 System Overview and Threat Model 64
4.4.3 Baseline DNN Malware Classifier 67
4.4.4 Crafting Adversarial Samples 68
4.4.5 Defensive Methods 71
4.4.6 Experimental Results 73
4.5 Evaluation Attack: Images 78
4.5.1 Method: Deep Generator Attack 79
4.5.2 Experimental Results 80
4.6 Conclusion 82
5. Conclusion 92
REFERENCES 94

LIST OF ILLUSTRATIONS

Figure | Page

2.1 A common deep convolutional neural network architecture. Key: conv: convolutional layer, mp: max pooling layer. 11

2.2 Sample images from MNIST and CIFAR10 datasets. 19

2.3 Loss on validation set of different student model architecture *w.r.t.* number of epochs. Since the loss are of different scale, we do not plot the curves in one figure. Key: c: number of convolutional layers, p: number of max pooling layers. 26

3.1 Stochastic columnar network architecture. 31

3.2 Learning rate schedule and loss curves with different DropColumn probability during training. 40

3.3 Effects of different number of columns and DropColumn probability. . 40

3.4 Training time in one epoch with different DropColumn probability and different number of columns. 41

3.5 Error rate of different dropping probability during the inference stage of a trained SCN network. The legend refers to the keeping probability of the DropColumn operation during the training stage. 43

4.1 Noise distribution. 56

4.2 Overview of the adversarial attack and defense of a dynamic analysis-based malware classification system. 65

4.3 Model of the baseline deep neural network malware classifier. 69

4.4 ROC curves of the baseline malware classifier for different numbers of hidden layers. 75
4.5 ROC curves of the malware classifiers with the distillation defense with different temperature. 76
4.6 Success rates of adversarial samples against the baseline classifer and the using defensive distillation with temperatures, $T \in 2, 10$. Each subfigure shows the results of a DNN with different number of hidden layers, H. 84
4.7 Percentage of successfully crafted adversarial samples for different sample crafting strategies with different temperature for distillation defense. 85
4.8 ROC curves of the malware classifiers for different regularization strength with different numbers of hidden layers. 86
4.9 Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with different weight decay strength. 87
4.10 Percentage of successfully crafted adversarial samples for the first 20 iterations with different sample crafting strategies, $D = 0.0005$ weight decay and $L = 3$ hidden layers. 88
4.11 ROC curves of the ensemble malware classifier with $E = 5$ classifiers for different numbers of hidden layers. 89
4.12 Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with different number of models for the ensemble defense. 90
4.13 Illustration of the deep generator attack architecture. 90
4.14 Generated images on different datasets that are recognized 100% confident as being the target class that attackers desired. 91

LIST OF TABLES

Table Page

2.1 Number of errors made by student model on MNIST testing set. MLP represents the result using a two hidden layer MLP as student, conv1 represents using 1 convolutional layer network as student. "rich", "logits" and "soft" represents distillation using rich targets, logit targets, and soft targets respectively. 20

2.2 Error rates on the test set with different percentage of training data available. "hard" refers to direct training on original hard label. . . . 21

2.3 Error rates on the CIFAR10 test set of different model distillation methods. Key: c: number of convolutional layers, p: number of max pooling layers, followed by number of parameters in the student network. M: millions of parameters in the student network. Multiplier: multipliers for the number of convolutional feature maps. 27

2.4 Error rates of knowledge transfer from CIFAR100 to CIFAR10 dataset. Key: c: number of convolutional layers, p: number of max pooling layers, followed by number of parameters in the student network. M: millions of parameters in the student network. 28

3.1 Architecture of stochastic columnar network. 47

3.2 Classification error rates on the CIFAR10 data set with different instantialization of stochastic columnar network. "Cxx" denotes number of columns in the network, and "Bxx" denotes number of feature maps used in each column. During training different probabilities are used for the DropColumn operation. 48
3.3 Classification error rates on the CIFAR100 data set with different instantialization of stochastic columnar network. 48
3.4 State-of-the-art performance achieved by other models: error rates on the CIFAR10 and CIFAR100 datasets 49
4.1 Classification accuracy of different level of noise rates on the MNIST dataset. 58
4.2 Classification accuracy of different level of noise rates on the CIFAR10 dataset. 59
4.3 Test error rates of the baseline malware classifier for different numbers of hidden layers. 75

CHAPTER 1

Introduction

Deep learning [1] gets tremendous traction in both academic and industry, thanks to its superior performance in perception tasks like image recognition [2], speech recognition [3, 4, 5] and natural language understanding [6, 7, 8]. Although many research has been done all over last few decades, the whole deep learning field really took off after 2012, in which year the AlexNet [9] came out and won the ImageNet competition [10] in a landslide. The success of deep learning can not be achieved without the emergence of big dataset, computing infrastructure (especially GPU), and the prosperity of experimental frameworks like Caffe[11], Tensorflow[12] (mainly backed by Google), MXNet[13], Theano[14] (mainly backed by MILA lab led by Yoshua Bengio, but ceased from active development now), PyTorch/Torch (mainly backed by NYU and Facebook), CNTK (computational network toolkit, mainly backed by Microsoft) Keras[15], Chainer[16], etc. Now many tech giants/start ups enter into the race for self-driving cars/medical images where perception plays one of the most crucial rule, deep learning is a vital part of the technology stack.

Before the resurgence of neural networks, data driven learning are usually comprised of two separated steps:

1) Feature engineering: extract features from raw data. Some representative approaches include Histogram of Oriented Gradients (HOG) [17], SIFT [18] for images, Bag of Words (BoW) representation for text mining, etc.

2) Learn models on extracted features. Among them, most popular models are Support Vector Machine (SVM) [19], Logistic Regression (LR).

There are advantages and disadvantages for this learning paradigm. The advantage is that the features are designed by human expert, so every feature has explicit meaning to model builder. In addition, those shallow models have good interpretability due to the simplicity in model where only a single weight matrix is learned. For example, we can easily know which features contribute more toward the classification. However, this paradigm depends heavily on human expert to design meaningful features for the target task. Neural networks, on the other hand, do not require the features to be designed by domain expert, but accept raw inputs (images, texts) and learn the feature representation automatically, thanks to the large amount of data available for training. The performance of deep learning models outperform hand crafted feature engineering by a large margin. Therefore, the academic community shift towards the new paradigm of representation learning [20], which learn useful features automatically from raw features. Under this paradigm, everything can be embedded into a vector space in which the similarity can be characterized by the distance of the vectors in the embedding space. For example, words, sentences or even paragraphs can be embedded into the vector space [21, 22, 23, 24, 25] and arithmetic operation can be applied onto the embedding to get meaningful words. The drawback of neural networks is that it relies on large amounts of labeled training data to properly learn the large amount of parameters, although this bothers academic/industry community a lot, it is not an obstacle insurmountable in the big data era.

Neural networks are hierarchical models stacked by layers of non-linear transformations. Hierarchical structure allows for efficient compositionality of input features to more complex feature representations. Primate visual cortex (V1) is composed of layers of neurons. Compositionality is one of the most fundamental property of how the world is formed. Simple features can compose into more complex features in the hierarchical architecture. For example, the lower layers in neural network learn the

edge, where those features are composed into junctions and more complex contours in the upper layers of neural networks [26]. Neural networks can be categorized in the connectionist model approaches, which is a term introduced by Donald Hebb to model the process of mental/behavior development of interconnected networks of simple units.

Due to higher capacity of neural network models, the performance improvement will not saturate as quickly as low capacity models. Therefore, with the massively available amount of data, deep learning models usually get better performance. Before the resurgence of deep learning, neural networks has been hot research topics in machine learning community in the 90s, and most of the critical technique advancements have been published. The network architecture are mostly comprised of convolutional operators and max pooling/average pooling operators, which is already pretty standard in convolutional neural networks (CNN) that time [27]. Due to the weight sharing in convolution and pooling operation, convolutional neural network is able to capture the spatial structure in inputs, and it is invariant to translation. One may also regard the sequence structure in text as one dimensional structure, and there are works that use convolutional networks for text processing like sentiment analysis task [28]. Recurrent neural networks (RNN) like long short term memory RNN [29] has also come out in 90s. The recurrent structure can be used to capture the temporal structure in inputs. The most important algorithm in leraning deep models, *i.e.* the back-propagation [30, 31] algorithm [30, 31] has been there since 80s as well. Although there are many variants of optimization algorithms (for example AdaDelta [32], Adam [33]) for neural networks, the core idea of learning remains unchanged, *i.e.* to update the weights based on error gradients.

During the 90s, most of the neural networks are shallow, and have a much smaller amount of parameters, due to the limitation of computing capability. There-

fore, the representation power of the models are limited, and MNIST [34] is considered a pretty big data set to run machine learning models on. However, due to the fast development of high performance computing technology like GPU (Graphics Processing Unit) computing technology (and some ASICs like TPU (Tensor Processing Unit)), nowadays ImageNet kind of become the new MNIST. The higher end computing devices offers hundreds of TFLOPS (tera floating point operations per second) compute capability, which vastly reduce experimental turnaround and expedite researchers to make more discovery in a short time. For example, in one of Google's experiment, 64 TPUs are used to train a ImageNet ResNet-50 model to 75% accuracy in just 22 minutes. This is a huge speedup compared with the computing infrastructure in 2012, where it takes two weeks for researchers to train an AlexNet [9] on GPUs. Researchers have exploited different ways to parallelize the training of deep learning models to reduce the experimental turnaround. The distributed training can be basically categorized into two paradigms, data parallel approach and model parallel approach. For the data parallel approach, the training data is splitted into many shards and distributed to multiple nodes. Each node compute the gradient and send back to a server to aggregate the results [35]. The model parallel approach partition the computational graph into different parts, and each node compute only part of the computational graph [36]. Data parallel approach is more popular due to the simplicity in implementation. WIth the distributed training framework, Facebook managed to train ImageNet in one hour [37] with a minibatch size of 8192 on 256 GPUs.

The development of modern deep neural networks can be split-ted into two stages. The first stage starts from 2006, where researchers start to explore unsupervised learning of network weights. Since the highly non-convex nature of the optimization for neural networks, finding a better initialization can help the model to

find a better optimum solution. The training is consisted of two steps: first, pre-train [38] with unsupervised model such as restrictive Boltzmann machine (RBM) [39] or stacked auto-encoder [40]/denoising auto-encoder [41]/contractive auto-encoder [42, 42]/sparse coding [43, 44]. Those pre-training algorithms train the weights layer by layer via minimizing the reconstruction error. After the pre-training, the model is fine-tuned on labeled data set with normal gradient descent.

However, researchers abandons the step of pre-training after 2012 due to that the size of labeled training data increases, and some technique advancements that alleviate the notorious gradient vanishing problem. The difficulty of preventing people from training deep neural network is the gradient vanishing problem [45, 46] during the back-propagation the gradients become a smaller value layer by layer such that the gradient approaches zero at the lower layers of the network. Therefore, the weights in the lower layers can not be properly updated. Basically, most of the advancements in recent years are related to this problem in different aspects. The various type of activation functions like (ReLU [9, 47], LeakyReLU, PReLU [48], ELU [49]) do not suffer from the gradient vanishing problem as the standard sigmoid activation function. For further reference, Xu et. al. [50] has a good benchmark on the performance of different type of activation functions. Xavier et. al [6] proposed to initilize the weights with respect to the number of parameters in each layer, which will help to keep a stable flow of gradients during the trainging. Batch normalization [51] helps to prevent the covariate shift problem (the input distribution of each layer changes during training), so that the we can be less careful about weight initialization, and more aggressive learning rate can be used during training. Another approach to alleviate the gradient is to add shortcut to the network architecture, most representative works are Highway Network [52], ResNet [53], DenseNet [54].

With the rapid advancement of deep learning in recent years, there are still many challenges remains to be tackled before it can reach to ubiquitous deployment. It sounds pretty exciting when AlphaGo [55] beats Lee Sedol in the ancient game of GO, apply those technology in real world applications is still far from ready due to the complex nature of real world environments. Although reinforcement learning algorithms have made rapid progress during the last few years [56, 57], the environment which is viable to deploy the AI agent is still mostly in toy experimental setup or very limited tasks. The other obstacle is the dependency on large amount of training data. Therefore, researchers have been actively researched into unsupervised deep learning, which is regarded as one of the most important problem for achieving AI.

With the wide adoption of deep learning models in industry, improve the accuracy of the models can have a broad impact on the success of applications where deep learning is in the critical path. In addition, making the models to run more efficiently while preserving the accuracy can save a lot of computing power/energy and reduce the inference latency. Last but not least, the security of the deployed models is of paramount importance for organization to protect data integrity and security of properties.

In this dissertation, we proposed some approaches to address those problems. We organize the dissertation as follows. In chapter 2, we introduce a new method for model distillation, which can distill the knowledge learned in a larger model into smaller ones while preserving the accuracy. The proposed approach outperforms standard practice for knowledge distillation. In chapter 3, we introduce a new neural network architecture which is termed as stochastic columnar network (SCN). The new architecture improves the model recognition performance upon state-of-the-art models with a lower amount of computational overhead. In chapter 4, we study the potential attacks during the training and evaluation stages of deep learning. We show

that deep learning models are vulnerable to attacks which can generate fake identity proofs easily. In addition, we proposed a new loss function which is more robust to injected noise during the training stage. We conclude the dissertation in chapter 5.

CHAPTER 2

Darker Knowledge

2.1 Introduction

Deep learning is popular nowadays for its superior performance in perception tasks in computer vision, speech recognition and natural language processing. In the meantime, it is also notorious for the difficulty in training deep networks.

Universal approximation theorem pointed out that: giving large enough number of neurons, a single hidden layer neural network can be a universal function approximator [58]. It is not that the expressive capability, but the optimization difficulty that limits the performance of shallow fat networks. Training of neural networks mostly rely on back-propagation of gradients using stochastic gradient descent (SGD) or its variants like AdaGrad, AdaDelta, RMSProp, and momentum based SGD, Nesterov Accerlerated Gradient (NAG), Adam. All these methods rely on the gradients as guidance to update model parameters. As the first wave of the resurgence of neural networks, layer-wise pre-training using Restricted Boltzman Machine (RBM) [39] or Stacked (denoising) Auto-Encoder (SA/SdA) [41] provides a good initialization for the highly non-convex optimization of deep netowrks. There are also works [59, 60, 61] that use Evolutionary Computation (EC) algorithms like Genetic Algorithm (GA) for neural network optimization. But GA requires to evaluate the fitness function many times to guide the evolution of agents. Intuitively, validation accuracy can be used as the fitness score, but it is very computation expensive, and seems like only viable for organizations that have a lot of computational power like Sentient AI, a company that use millions of distributed cores to evolve neural networks.

On the other hand, Human beings attend school to learn knowledge from teachers who have experienced much more than average students. In order to implement intelligence in learning machines, it is essential for the system to have the capability of distilling knowledge from teacher models (which are usually cumbersome models with large number of parameters or neurons in the case of neural network). The knowledge learned by teacher models can guide the parameter search of student models which are usually harder to be optimized without the guidance. This process is called knowledge/model distillation, and there are some seminal works introduced by Geoffrey Hinton et al. [62] and Jimmy Ba et al [63] and Rich Caruana et al. [63].

In the context of deep learning, knowledge distillation is a cornerstone used in many active research areas. Hinton et al. [62] mentioned that: knowledge learned by a model are stored in the weight parameters, which are bound to a specific network architecture. Conventional knowledge transfer like fine-tuning top layers of the network [11] still requires the bottom layers to remain the same in order to use the learned weights. Net2Net [64] enables knowledge transfer to a deeper/wider network by warmstart the model with identity connection, but still requires extended models share modules with the same architecture. When the architecture changes, the weights can not be used anymore. Knowledge/model distillation has no architecture constraints, provides a way to transfer the knowledge learned by one model to another one with any different architecture. This enables us to transfer knowledge from larger/deeper neural networks to smaller/shallower ones, and even transfer knowledge between recurrent neural networks(RNN), convolutional neural networks (CNN) and deep neural networks/multi-layer perceptrons (DNN/MLP). For example, William Chan et al. [65] used soft alignment to distill the knowledge learned from a RNN to a DNN. The distilled DNN model is easier for deployment, and outperforms state-of-the-art DNN models trained directly. Krzysztof J. Geras et al. [66] compressed the

knowledge learned by a long short term memory (LSTM) RNN into a CNN. The distilled model achieved state-of-the-art performance on the standard 309h Switchboard automatic speech recognition task, and consumes less memory and computational time in deployment.

Light memory consumption makes it possible to deploy distilled models onto embedded devices like mobile phones. Shorter response time reduces latency and improves user experience when deployed at massive scale. Shallow networks take less cycles in the inference stage, allows for easier parallelization. Note that knowledge distillation is orthogonal to other model compression techniques like low rank decomposition of weights [67], vector quantization [68], DeepCompression [69], SqueezeNet [70], so the distilled model can be further compressed using the those methods. In this work, we focus on the performance comparison with current standard practice of model distillation but not the model compression rate.

In reinforcement learning, Google DeepMind used knowledge distillation to distill action policy in Deep Q Networks (DQN), in which multi-task policies are distilled into a single policy [71]. With a drastically smaller network, the distilled model demonstrates expert level performance. In gradual learning, knowledge distillation is used to prevent the model from forgetting old tasks (using soft target alignment) when learning new tasks [72].

Knowledge distillation is also widely used in security domain to implement black box attack on neural network based security systems [73]. To implement an adversarial attack, gradients information of the output *w.r.t.* the input is required. With model distillation, attackers could mimic the deployed model (which is usually a black box to users) with the student model, and use gradient information from the student model as the guide to implement adversarial attacks. Knowledge distillation

can also be used in a semi-supervised way and ensure differential privacy while training deep models [74].

With such a broad range of research topics built on top of model distillation, a lot of higher level tasks would benefit from it if we could improve the performance of model distillation.

In this paper, we propose a simple yet effective way to improve model distillation. We organize the paper as follows: In section 2.2, current standard practice of model distillation is introduced. Following that, we present the new way of performing model distillation. Extensive experiments on are presented in the next section. We conclude the papers with considerations for future works.

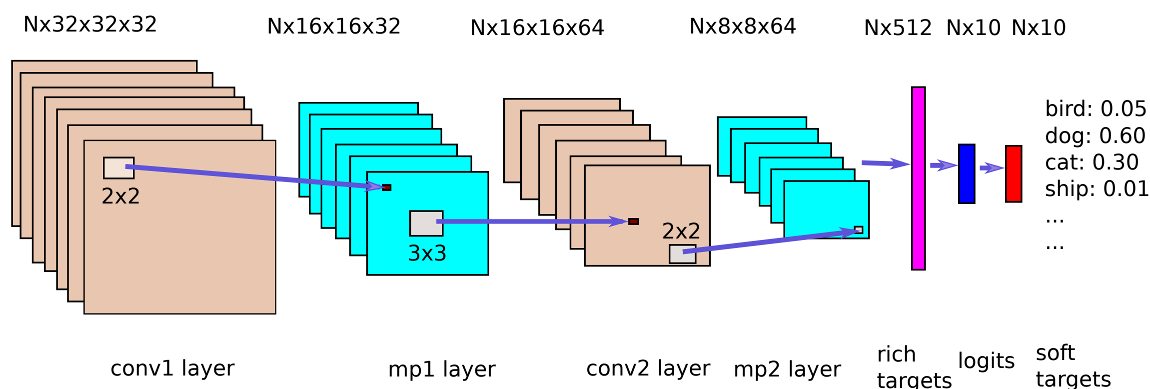


Figure 2.1: A common deep convolutional neural network architecture. Key: conv: convolutional layer, mp: max pooling layer.

2.2 Related Works

In this section, we will introduce related works on the topic of knowledge/model distillation. Those familiar with the literature might skip to the next section directly.

Vanilla Model Compression

Caruana et al. [75] proposed model compression to distill the knowledge of a trained large teacher model to a much smaller one. The teacher model is used to obtain labels on a large set of data, then the data with the predicted labels are fed into training the small student model. This approach is similar to self-taught learning, except that the goal is to train a student model, rather than the model itself. It also proposed methods for modeling real data distribution so that they can generate more training data to further improve the training of student model.

The distilled model is as accurate as the teacher model, although with much fewer parameters. However, the same accuracy often can not be reached by directly train the student network on original training data.

Dark Knowledge/Soft Targets Alignment

Dark Knowledge [62] is proposed by Geoffrey Hinton et al to improve model distillation performance. Instead of using the predicted hard labels as training targets, dark knowledge uses the predicted probability $p = [p_1, p_2, ..., p_c]$ ($\forall i, 0 < p_i < 1$, and p_i is the probability of a sample being classified into class i) of the teacher model as the training target. The probability scores are also known as soft targets, in contrast to the hard targets of original {0, 1} labels. Figure 4.13 is a schematic illustration of a standard convolutional network. The last red column is the predicted probability of each class. Dark knowledge argues that the probability scores capture the relationship between classes. Taking MNIST dataset for example, digit 1 and 7 looks more similar than 1 and 8, so the prediction of a hand written digit with a ground truth 1 is usually classified with a higher probability of 7 than 8. Similarly, for natural image classification, cats looks more similar to dogs than desks. Thus for a dog image, the output might be $[0.6, 0.3, 0.01]$ for dog, cat, and ship class respectively. By utilizing the relationship between classes, the teacher model communicates more information

to the student model, which helps training the student model to better mimic the complex non-linear function learned by the teacher model.

But usually with vanilla softmax activation (without using a temperature to normalize the logits feeding into softmax function), many of the probability scores would concentrate on one class too much, making the correlation between classes not that obvious. So Hinton et al. proposed to use a higher temperature to normalize the logits so that the probability scores distribute more evenly to better reflect the correlation between classes.

Specifically,

$$p = softmax(z/T) \tag{2.1}$$

$$softmax(x) = exp(x)/\sum_{i=1}^{c}(exp(x)) \tag{2.2}$$

where $p \in \mathbb{R}^{c\times 1}$ is the output probability scores *w.r.t.* each class, softmax is a function which is usually applied to a vector to get the probability scores, $z \in \mathbb{R}^{c\times 1}$ is the logits (the layer before feeding into the softmax function) output by a deep neural network $f(x)$ applied on an input x. T is the scalar temperature. Intuitively, when T is large, the difference between normalized logits is small, so the output probability will be pushed towards uniform distribution.

If the the distilled student model could match these soft targets on a large transfer set (note that the transfer set do not need to be constrained to the original data used for training, but could be any data), then we can say that the student model distill most of the knowledge stored in the larger teacher model. So we can formulate the model distillation problem as soft targets alignment via cross-entropy loss between probability scores of student model and teacher model.

In the distillation process, the student model should use the same temperature as the teacher model. When deploy the student model, vanilla softmax function

should be used to get the probability scores back to normal. The temperature need to be tuned for best performance. The teacher model can be an ensemble of teachers to form a super teacher, where the targets could be the geometric or arithmetic mean of all teachers. They found that a single student model, while guided by the super teacher, could perform similarly on speech recognition tasks.

Logit Regression

Jimmy Ba and Rich Caruana [63] also studied whether shallower networks could perform as well as deep nets. Instead of using the probability scores as the training targets, logits are used as regression targets. In Figure 4.13, the blue column represents the logits, which are real values before feeding into the final softmax layer.

Specifically, [63] formulate it as logit regression with L2 loss as follows:

$$L(W_S; X) = \frac{1}{2N} \left\| Z_T - Z_S \right\|_2^2 \tag{2.3}$$

$$Z_T = TNN(W_T; X) \tag{2.4}$$

$$Z_S = SNN(W_S; X) \tag{2.5}$$

where TNN and SNN are the non-linear function of teacher network and student network that mapping the input data $X \in \mathbb{R}^{N \times d}$ into logit space, $Z_T \in \mathbb{R}^{N \times c}$ and $Z_S \in \mathbb{R}^{N \times c}$ are the logits of the teacher network and student network, W_T and W_S are parameters of teacher and student networks, N is the number of data. Note that here logits refer to the summarized logit scores Z, and should not be confused with logistic regression.

Ba et al. found that by using the proposed model distillation strategy, single hidden layer feed-forward network can perform similarly to well engineered complex deep convolutional networks. They argue that it is not the expressive capability of shallow models that limits the performance, but because current optimization algorithms work better for deep architectures. If we can come up with better learning

algorithms, shallow models might perform as good as deeper ones. Ba et al. also claims that logit regression works better than using L2 loss or KL divergence in probability space.

Deep and Convolutional?

In [76], researchers studied that whether the the depth and the convolutional structure are needed in the student model. Extensive experiments are carried out using Bayes Optimization for hyper-parameters. They conclude that it is hard for feed-forward neural networks to learn the function learned by convolutional neural networks in image recognition tasks. To reach the accuracy of deep convolutional teacher models, at least serveral layers of convolution are needed. In this work, we do not study the effect of having convolutional layers or not, but focus on the performance comparison between different knowledge distillation methods. It's also possible to achieve better accuracy with better teacher model formed by ensemble or other advanced architectures/methods, which is beyond the study of this work.

2.3 Darker Knowledge

To the best of our knowledge , most of the works based on model distillation are using either soft targets alignment, or logit regression. If the model distillation method is improved, many other tasks depend on that can be benefited as well. In this section, we propose darker knowledge via rich latent targets regression, and validate empirically that the new proposed approach improves upon current standard practice of model distillation.

The idea is very intuitive. From information theoretic point view, there are a lot more information contained in the activations below the logit layer. We call these activations rich targets since they contain semantically rich information. In a common convolutional architecture shown in figure 4.13, the magenta column represents the

rich targets (note that rich targets is not restricted in convolutional networks, but could be in any network architectures). Logits are summarized from rich targets, and softmax function is applied to logits to get the final output in the probability space. Each hard target contains only one bit of information. Soft targets/probability scores give better proximity information between different classes than hard targets. Logits carry more information, but it is still much less than rich targets. Rich targets can potentially communicate much more about the underlying latent factors that actually lead to the final classification decision. Although we do not know specifically what each target represents, we know that those targets are trained to be discriminate in order to properly classify the training data. Once a deep teacher model is trained, rich targets can be easily obtained.

It is known that deep neural networks are highly nonconvex, and hard to optimize. Deeper networks usually have more local minimum, and it tends out the many of the local minimum are of similar energy, so it is easier to get a better solution than shallower ones [77]. By using rich targets from a deep model as guidance, shallower networks have more hints to search over the huge parameter space and find a better local optimum than directly optimize the shallower network. Using logits or soft targets could also guide the optimization of student network, but the constraint is less tight as rich targets. It is possible that top layer weights co-adapt to minimize the divergence of logits or soft targets, thus features do not need to be very strong to achieve the divergence minimization goal, weaker features could be learned. By minimizing the divergence of rich targets, the student model is forced to learn stronger features already learned by the teacher model.

We have validated this assumption using a simple experiment. We train a student model of 6 convolutional layer and 3 interleaved max pooling layers using soft target alignment. Then we chop off the top layers and using the rich target layer

as features to train a linear classifier, this approach achieves 14.41% error rate on the CIFAR 10 test set. If the student model is trained with logit regression, 13.15% erorr rate is achieved using the layer corresponds to rich targets. While using rich target regression, we achieve 11.95% error rate.

Based on the above intuition, we propose to perform model distillation via rich latent targets regression. Following dark knowledge, we term this method Darker Knowledge since the knowledge are hidden deeper in the lower layers of the network, and is not very straight-forward to be discovered. Note that not exactly the last layer activations should be used as training targets, but layers below that could also be used if appropriate. For models without a fully-connected layer on the very top of the teacher network, the flattened pooling layer may be used as the rich targets. In general, lower layers contain more specific low-level information, higher layers contain more general high-level information. We may even use multiple layers to guide the distillation process better.

We formulate the method mathematically as follows:

$$L(W_S; X) = \frac{1}{2N} \left\| R_T - R_S \right\|_2^2 \tag{2.6}$$

$$R_T = TNN(W_T; X) \tag{2.7}$$

$$R_S = SNN(W_S; X) \tag{2.8}$$

where $R \in \mathbb{R}^{N \times h}$ is the rich targets (h is the dimension of the hidden layer that rich targets correspond to).

After the distilling phase, the simplest way to go would be just concatenate the last layer of the teacher model on top of the student model if the discrepancy of the rich latent targets is small between student and teacher models. Actually in our experiments, we found that decent accuracy can be achieved in this way, but train a linear layer on top of that could be more accurate than directly concatenate the

last layer weights. In our experiments, we will learn a linear classifier on top of the student model. Off-the-shelf SGDClassifier from scikit-learn is used for simplicity.

We have also tried using L1 loss instead of L2 loss. Empirical results show that L2 loss works better, so we stick to L2 loss in our experiments.

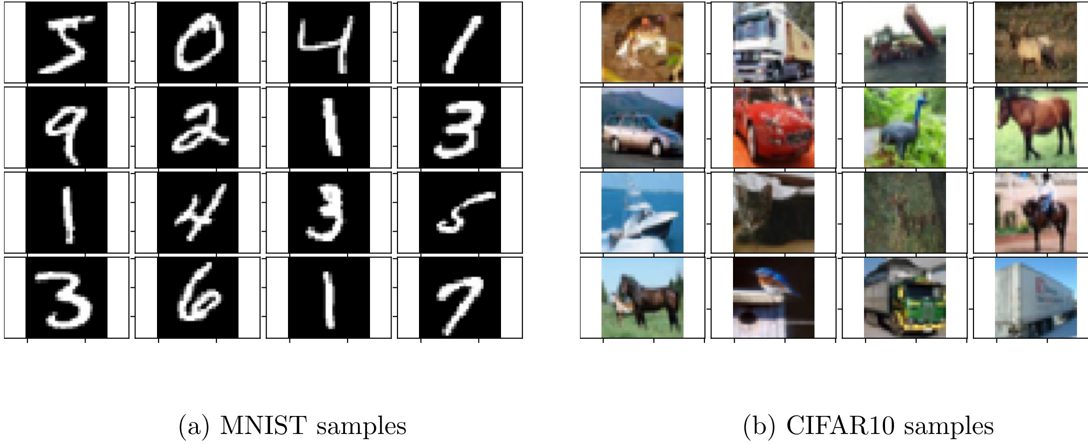
2.4 Experimental Results

Experiments Setup

The experiments are carried out in Keras with tensorflow as the backend. All codes will be open sourced shortly for reproducibility.

For all the experiments, Adam [33] optimizer is used with default parameters $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 1e - 8$, with a batch size of 32, for a total of 200 epochs. Weights are initialized by the default glorot uniform intializer [45], learning rate is scheduled as [0.0001, 0.001, 0.0001, 0.00001] with turning points at epochs [5, 160, 180, 200]. The first 5 epochs is for warm start the training, then the more aggressive learning rate is used for the most of the training epochs. In the final stage, the learning rate is reduced to further optimize the objective function. Learning rate will be reduced if a plateau is reached for a consecutive of 5 epochs, by a factor of $\sqrt{0.1}$. The learning rate schedule is visualized in figure 3.2b (d). The temperature for soft targets is tuned from 5 to 15.

We organize the experiment section as follows: in the first subsection, preliminary results on MNIST dataset is presented to distill a convolutional network into a MLP and a shallower convolutional net. After that, extensive experiments with different training data setting and student network architectures on CIFAR10 dataset are presented. We demonstrate the effectiveness of knowledge transfer across datasets in the last subsection.



(a) MNIST samples

(b) CIFAR10 samples

Figure 2.2: Sample images from MNIST and CIFAR10 datasets.

2.4.1 Preliminary Experiments on MNIST Dataset

We first do experiments on MINST data set to validate our idea. MNIST is a widely used digit recognition dataset, where each image is 28 by 28, with gray scale value. Figure 3.3a (a) shows some sample images from MNIST dataset. We use the default training/testing split in our experiments. This is a relatively easy task for convolutional network nowadays, and vanilla MLP can achieve a pretty decent accuracy on this task.

Firstly, we show that we can transfer the knowledge learned by a convolutional network to a MLP. Within tensorflow, we trained a convolutional network with two convolutional layers interleaved with 2x2 max pooling layers and ReLU (rectified linear units) activation function. Filter size of the convolutional layers are (5, 5, 1, 32) and (5, 5, 32, 64), respectively. A dropout rate of 0.5 is used during the training phase. This teacher network achieved 87 errors among the total 10, 000 testing data. We have also trained an MLP, with two hidden layers, each hidden layer has 1024 neurons. While trained directly on the original hard label, this network achieves 304 errors.

	rich	logits	soft
MLP (784-1024-1024-10)	114	131	144
conv1 ((5, 5, 1, 32)-1024-10)	91	123	130

Table 2.1: Number of errors made by student model on MNIST testing set. MLP represents the result using a two hidden layer MLP as student, conv1 represents using 1 convolutional layer network as student. ”rich”, ”logits” and ”soft” represents distillation using rich targets, logit targets, and soft targets respectively.

Table 4.1 shows the results of using different model distillation methods. We can see that rich target regression makes less errors on the test set than commonly used soft targets and logit regression. We have also trained a shallower CNN to mimic the teacher model, with only one convolutional layer (5, 5, 1, 32) of stride 2 and a max pooling layer with stride 2. As expected, convolutional network makes less errors than MLP student network on the test set. Again, our method outperforms standard practice for model distillation.

Since MNIST dataset is simple for convolutional network nowadays, it is used to serve as an initial validation of idea. It would be more convincing to show the performance comparison on larger datasets. In the next section, we will resort to CIFAR10 dataset to show the performance of each approach under different settings. The experiments require a lot of model training. It takes a lot of computation power and weeks of training time to train a model on the ImageNet dataset, we will add the performance comparison on ImageNet dataset in the future.

2.4.2 CIFAR10 Dataset

Dataset Description

	10%	30%	50%	70%	90%	100%
rich	21.52	16.86	15.97	15.07	14.85	14.26
logits	30.24	23.23	21.71	20.88	20.77	18.77
soft	30.53	24.72	23.86	22.01	21.55	18.77
hard	40.15	27.71	25.05	22.69	21.73	21.05

Table 2.2: Error rates on the test set with different percentage of training data available. "hard" refers to direct training on original hard label.

CIFAR10 dataset consists of 60, 000 natural images selected from 80 million tiny images [78]. It is divided into 50, 000 training set and 10, 000 testing set. There are 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. Each image is 32 by 32 pixels with 3 RGB color channels. In our experiment, we use the 50, 000 training set as the transfer set to train the student model. Figure 3.3a (b) shows some sample images from CIFAR10 data set. Note that we can also use outside images from the 80 million images as the transfer set, but it would take very long for the training, so we do not resort to images beyond the training set of CIFAR10 dataset.

Convolutional networks help with modeling invariance in images, and is more efficient than feed-forward networks with same parameter budgets. It is observed in previous works [63, 76] that student models without convolutional architecture could not learn very good classifiers for natural images with many sources of invariance such as light, view angle, rotation, thus we do not study the effect of having convolutional layers or not, and only use convolutional networks as student models.

Train Student Model with Partial Data

We have trained a teacher model using the CIFAR10 dataset, with 4 convolutional layers interleaved by 2 max pooling layers. The model is trained using augmented data with random horizontal flip, and vertical and horizontal shifts, and achieve a 86.38% accuracy on the testing set.

We first perform experiments using different fraction of training data to train the student model. The student model is similar to the architecture of teacher model, with 4 convolutional layers and 2 max pooling layers. To make fair comparsion, the same random seed is used to generate index of training data being selected. No data augmentation is used to make the comparison more strict. Table 2.2 shows the performance of different distillation methods. We can see that our method outperforms standard practice of using logit regression or soft target alignment by a large margin. This suggests that our method can communicate more useful information to the student.

In addition, training directly on the original hard label overfit seriously since the training data is scarce, but the performance using model distillation is much better. When the training data is scarce, student model is vulnerable to overfitting, but the teacher model is trained on a larger dataset without overfitting. Model distillation serves as an implicit regularization to the student model. Using the same teacher model, darker knowledge better regularizes the student model to learn stronger features than standard practice because it communicates more information in the latent feature space of the teacher model.

Student Models with Different Architectures

In this subsection, we make thorough comparison of different knowledge distillation methods with different student model architectures. To boost the performance, we have trained a stronger teacher using wide residual network (WRN) [79], which is a variant of the residual network family that uses more feature maps. The WRN

model consists of 16 layers, with a width factor of 8. The teacher model achieves 93.68 accuracy on the CIFAR10 test set.

We have trained student models with different number of convolutional layers, max pooling layers, and different number of convolutional feature maps. The number of feature map in each convolutional layer is set with a base number and then multiplied by a multiplier. Figure 3.2b (a) - (c) shows the convergence of validation loss of different student model architectures using the WRN teacher model. Figure 3.2b (d) shows the learning rate schedule used during training. We do not show the training loss in case that the figures look too cluttered. We can see that when the number of convolutional layers is increased from 2 layers to 4 layers, the loss drops significantly. This suggests that in order for the student model to perform well, at least several layers of convolution are needed to model the invariance learned by the teacher model.

Table 2.3 shows the performance comparison of different knowledge distillation methods. Note that when the number of max pooling layer is increased, the total number of parameters drops significantly since the input neurons for the last fully connected layer will be reduced significantly. We can see that our method consistently outperforms the current commonly used model distillation methods. The performance of logit regression and soft target alignment is somewhat mixed, because the knowledge contained in these two layers are similar. Not surprisingly, when the number of convolutional layer is small, adding more convolutional layers improves the accuracy; when the number of parameter is small, adding more parameters also reduces the errors rates noticeably.

2.4.3 Knowledge Transfer across Datasets

In this subsection, we demonstrate the capability of knowledge transfer between datasets with the proposed model distillation method. For the benefits of knowledge transfer, we use a dataset homogeneous to CIFAR10 to train the teacher model. CIFAR100 [80] is also a subset collected from 80 million tiny image dataset by Alex Krizhevsky. In contrast to CIFAR10, CIFAR100 contains one hundred classes with a coarse label and several fine labels.

A teacher model of the architecture (with 4 convolutional layers, 2 max pooling layers) similar to the vanilla CNN model in CIFAR10 experiments is trained on the CIFAR100 dataset, which achieves 62.46% accuracy on the test set for the 100-way multi-class classification problem. The model is trained using augmented data with horizontal flip, and random horizontal and vertical shifts. The student models are similar as the ones used in the CIFAR10 dataset, except that the logit layer has 100 neurons for the logit regression.

Different from previous experiments, the teacher model now contains 100 logit neurons and probability scores. Therefore, for logit regression, a linear classifier is sticked on top of the learned logits after model distillation. Since the probability soft targets is for the 100 classes of the CIFAR100 dataset, it is not principled to use the soft targets approach in this situation. Therefore, we do not compare with soft targets distillation in this experiment.

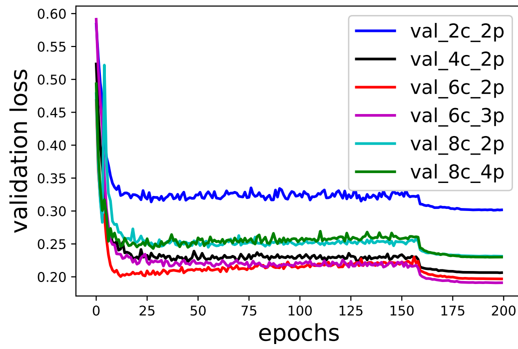
Table 2.4 shows the error rates of knowledge transfer from CIFAR100 to CIFAR10 dataset. We can see that the performance via rich target regression is significantly better than logit regression. We argue that the reason lies in two folds: on the one hand, rich targets contain more information than logits; on the other hand, the knowledge contained in logits is too specific and restricted to the original datasets, thus limits its usability for knowledge transfer across datasets.

2.5 Future Works

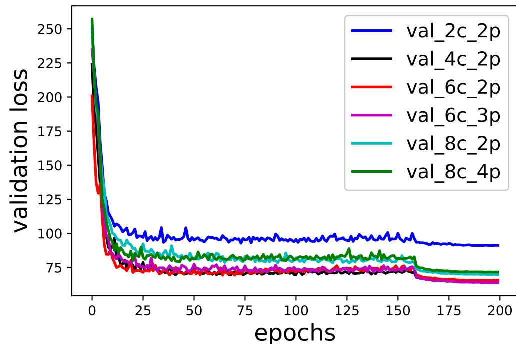
As the experiments suggest that darker knowledge works better than current standard practice of knowledge distillation, it would be interesting to see whether in other domains like speech recognition can be improved using the proposed method. We leave this study for the future work.

2.6 Conclusion

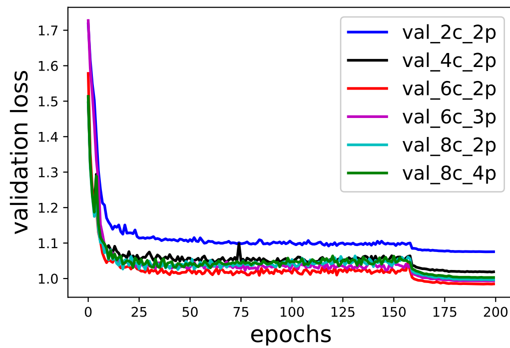
In this work, we propose to perform knowledge distillation via rich latent targets regression. Rich targets communicate more useful information to guide the optimization of student models, help student models learn stronger features for classification. The performance of the proposed approach works better than standard practice of using logit regression or soft target alignment on a wide variety of datasets and model architectures. On the task of knowledge transfer across datasets, the proposed method also works significantly than previous methods. We conjecture that a broad range of machine learning tasks (as mentioned in the introduction) relying on model distillation should be benefiting from the improved model distillation method.



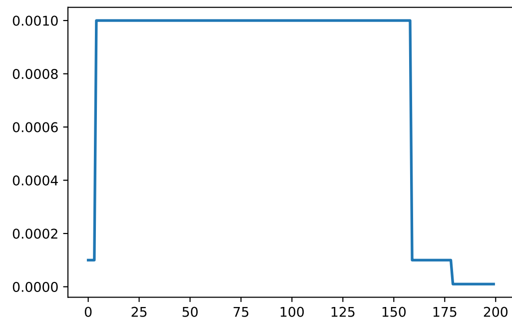
(a) Validation loss of rich target regression.



(b) Validation loss of logit target regression.



(c) Validation loss of soft target alignment.



(d) Learning rate schedule during training.

Figure 2.3: Loss on validation set of different student model architecture *w.r.t.* number of epochs. Since the loss are of different scale, we do not plot the curves in one figure. Key: c: number of convolutional layers, p: number of max pooling layers.

multiplier 1	2c-2p, 2.12M	4c-2p, 2.16M	6c-2p, 4.6M	6c-3p, 1.45M	8c-2p, 2.26M	8c-4p, 0.85M
rich	20.27	14.69	11.89	12.33	11.78	13.88
logits	20.97	15.69	14.09	14.78	14.26	15.83
soft	20.83	16.39	14.02	14.20	14.22	16.59
multiplier 2	2c-2p, 4.27M	4c-2p, 4.46M	6c-2p, 9.98M	6c-3p, 3.70M	8c-2p, 10.72M	8c-4p, 2.85M
rich	19.23	12.75	11.48	11.95	11.94	14.49
logits	20.06	14.93	13.77	13.38	14.38	15.52
soft	19.92	15.34	13.05	13.93	13.27	15.72
multiplier 3	2c-2p, 6.50M	4c-2p, 6.88M	6c-2p, 16.10M	6c-3p, 6.72M	8c-2p, 17.8M	8c-4p, 6.02M
rich	18.64	12.49	12.11	12.28	11.54	14.4
logits	19.65	14.35	13.97	13.70	13.54	15.48
soft	18.77	15.38	13.24	13.37	13.96	15.16

Table 2.3: Error rates on the CIFAR10 test set of different model distillation methods. Key: c: number of convolutional layers, p: number of max pooling layers, followed by number of parameters in the student network. M: millions of parameters in the student network. Multiplier: multipliers for the number of convolutional feature maps.

methods / arch	2c-2p	4c-2p	6c-2p	6c-3p	8c-2p	8c-4p
rich	25.17	23.69	24.04	23.78	24.14	24.55
logits	32.01	29.78	30.69	31.11	31.19	33.48

Table 2.4: Error rates of knowledge transfer from CIFAR100 to CIFAR10 dataset. Key: c: number of convolutional layers, p: number of max pooling layers, followed by number of parameters in the student network. M: millions of parameters in the student network.

CHAPTER 3

Stochastic Columnar Network

3.1 Introduction

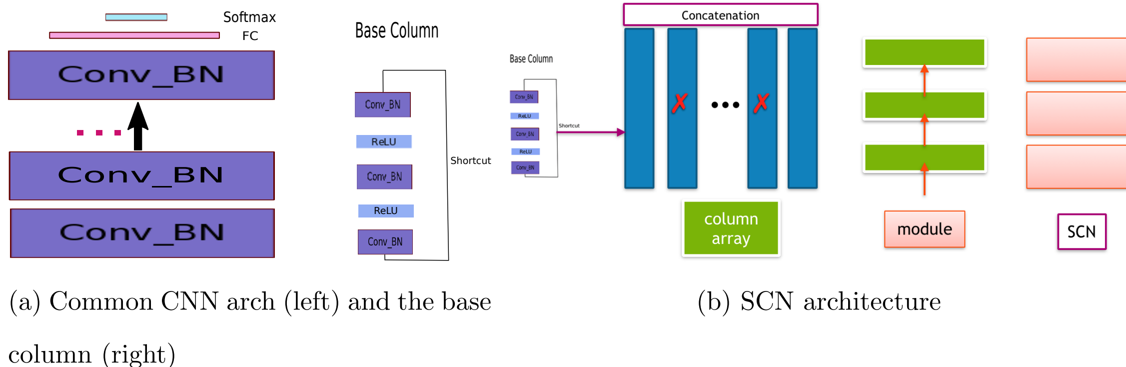
Neural network architecture is the cornerstone for deep learning research. After Krizhevsky et al. kicked off the ball of deep learning for image recognition, the architecture of neural networks evolves a lot all along these years, and the error rates on the ImageNet competition drops all the below 5 percent for the top 5 predictions. To be more specific, AlexNet uses interleaved layers of convolutional layers and pooling layers, and summarize the information with two fully connected layers which is then fed into a softmax classifier. These shared weights convolutional filters and pooling layers can model many type of invariance property of images. In 2013, Matthew Zeiler and Rob Fergus used an architecture similar (which is termed ZFNet) to AlexNet, and spotted the top 5 position on the ImageNet leaderboard. In the coming year, VGG net [81] from Oxford and GoogLeNet [82, 83] further improved image recognition accuracy by train a network that is much deeper. In addition, GoogLeNet introduced the branching into the network architecture in the first time. Different branches use different filter size (i.e. 1x1, 3x3, 5x5 convolutional filter) to capture different scale of features, and then summarized into the top feature map by depth concatenation.

The 2015 ImageNet competition is won by ResNet [53] from MSRA, which achieved an error rate of 3.57%. This is the first time which reduced the error rate drops below 5%, and trained a network more than 150 layers (later with stochastic depth network [84], researchers have trained network more than 1000 layers.) ResNet conjectures that deeper network should perform better given enough training data,

but in practice just stacking more layers does not give superior performance due to the notorious problem of vanishing gradient during back propagation training. With a very deep network, the very bottom layer cannot receive significant gradient information upon which the back propagation stage hinges. ResNet add identity mapping to facilitate the flow of gradients from top layers to bottom layers. With those identity mappings wired between layers, we can train much deeper neural networks. Inspired by ResNet, there are a lot of variants to further improve the architecture. DenseNet [54] adds identity mapping between a certain layer to each upper layer to form a densely connected computational graph. WideResNet [79] adds more filters to the feature maps which leads to a wider network. Inception V4 network [85] adds identity mapping to the inception architecture and improved the performance a little bit.

It seems that researchers are always focusing on making the network deeper and deeper along these years. We conjecture that going deep is not the only way out to push the boundary of neural network performance. In addition, given the inherent sequential computational paradigm of neural networks, deeper networks also need more stages of computation, which might lead to slow training and inference of the model.

Currently neurons in neural networks are usually laid out in a flat manner, which is easy for implementation. But human visual cortex exhibits more delicate structures that is very helpful for signal processing. One important type of structure in visual cortex is called cortex column. Each column serves as a basic functional unit in human visual cortex. In each column, many neurons are encapsulated to collaboratively evolve so that the capsule fires it gets stimulated by certain configurations or setups. In this work, we are trying to mimic this important structure in visual cortex, and design neural network architecture for image recognition tasks. We will validate the idea with popular image recognition benchmark datasets.



(a) Common CNN arch (left) and the base column (right)

(b) SCN architecture

Figure 3.1: Stochastic columnar network architecture.

3.2 Stochastic Columnar Network

We first categorize the common approaches for model ensembling, which usually takes a majority vote from a committee of models. This committee of models can be trained in various ways. It is better to have different/heterogeneous models in the committee. If all the model are making similar predictions, the prediction of the ensemble may reduces to that of a single model. If ranked from more heterogeneous models to homogeneous ones, we have the following strategies to form the committee for model ensemble.

(1) Each model is of different architecture. In this case, the models are heterogeneous, the ensemble of models should be performing better since the trained models are less correlated. However, the drawback of the above approach is that each model need to be trained separately, which introduces significant computational overhead.

(2) Each model is trained with the same architecture, but different initialization. Since deep neural network models are highly non-convex, different initialization usually leads to different converged model, although the performance is usually similar due to the fact that many of the local minimums are of similar energy. Similar to the

approach (1), this approach also introduces a lot of computational overhead during training. The trained model would be more correlated, since the same architecture is used.

(3) The models are trained with the same architecture, same initialization, but snapshots of the trained weights are saved at different epochs. Those different snapshots form a committee for the ensemble. This approach incurs no extra overhead during training, but would still require to save all the weights for the models. The models are very correlated since they are trained with the same architecture and the same initialization.

The above approaches only takes the final classification output from each large network, therefore each network provides very limited information about what it has learned. In all the above approaches, all the networks/models in the ensemble need to be evaluated during inference stage, which is also quite computational expensive, and also memory expensive for storing the weights of all models. This is a big challenge faced by many real world applications, especially in the circumstance of running the inference on embedded devices like cellphones, IoT (internet of things) perception sensors, etc.

It is widely known that Dropout [86] can be viewed as an ensemble of an exponential number of models. With a dropout probability of 0.5, dropout demonstrates strongest regularization. However, since it is just randomly dropping neurons during training, all the models are sharing the same base architecture, therefore the models being ensembled are still very similar, making the ensemble not very strong.

In this section, we introduce a new neural network architecture that implements model ensemble in one single network such that best parameter efficiency is achieved. We call this type of ensemble as *IME(Intra-Model Ensemble).* Since we are dealing with vision perception tasks, the network is instantiated as a convolutional network

which can model various types of invariance that is helpful in image recognition tasks. Note that it can also be instantiated as a fully connected network if the data is more suited for a vanilla DNN.

We think the key elements for implementing a IME learning system lie in the following aspects:

(1) Modular design with small components such that less parameters are needed in total. If each component is too large, the total number of parameters would also be large. In this work, we propose to use a small computational graph (a small stacked layers of neurons) as the base component.

(2) Hierarchical architecture with early fusion of learned features to allow for complex synergy between components. Instead of fusing the final prediction for each large model, we fuse the information of each component in early stage to allow for more complex synergy between components.

(3) Independence between components. With more independent components, the redundancy is much smaller, so that given a fixed parameter budget the model can capture more useful features for recognition. We propose DropColumn operation to promote the independence between components, which will be introduced in the following subsection.

Through modular design, we can enable feature reuse by hierarchically composing a number of small computational graphs into a single bigger network. By stacking multiple of such small computational graphs horizontally and vertically, the whole network allows for more complex compositionality compared with just using the classification output of each network. In this way, we only need to store the weights for one single network, and evaluate the one single network once for prediction. Note that the traditional neural network ensemble methods can be viewed as an implementation of IME if we view each gigantic network as a basic component.

It is also an active research area for gradual/continual learning, where the goal is to encompass more knowledge into one single model instead of training one model for one task. Due to it's modular design, our model could potentially encompass many tasks into one single network for continual learning by adapting different modules to different tasks, which we will left for the future research. In this work we focus on the single task (classification) problem rather than continual learning.

Taking the analogy from visual cortex, we call each small computational graph a column (a set of encapsulated neurons), where each small computational graph accepts inputs and go through a series of nonlinear mappings to get an intermediate output layer. In figure 3.1a, the left shows a common neural network architecture (pooling operation is omitted for simplicity), and the right shows a base column setup. Each column/small computational graph is composed of a series of convolution and non-linearities. Without bells and whistles, we use 3x3 kernel size for all the convolution and ReLU as activation function. It's possible that using different convolutional filters or alternative activation functions can further improve the performance of the whole model, but those most standard settings in modern neural networks are used since we want to focus on the effect of the newly introduced network architecture. Neurons only pass information to the neurons in the higher layer *within* the same column. A shortcut connection which connects the input and output of a column is added to facilitate the training of deeper models (if the number of input and output feature map is different, a convolution mapping is used instead). The base column has a width parameter, *i.e.* the base number of feature maps, which will be represented as B in the following. Usually a column is slim because B is set to a value much smaller than the number of feature maps in a common CNN.

Figure 4.13 is a sketching illustration of the proposed neural network architecture. Many columns form a column array, then the output of those columns are

concatenated to form the input of the next column array to allow for complex interaction between columns. Note that there is no information flowing between columns within a certain column array, however, between column arrays the columns could interact with each other since the output of each column in the lower column array is concatenated and then fed into the next module as input. To fix the design space of the architecture, we stack three layers of column array as a module, and stack three modules to form the final network. The original input images are first transformed into a 4-D tensor before feeding into the first module, and the final output goes through a softmax function which outputs number of class probability values. Without bells and whistles, cross-entropy is used as a loss function for the classification problems. Table 3.1 summarizes the architecture's basic setup for repeatability, where B is the number of feature maps in a base column, C is the number of columns, and repetition represents the number of times a certain module is stacked.

$$p_c = \exp(z_i) / \sum_{c=1}^{C} (\exp(z_i)) \tag{3.1}$$

where z is the logits score in the final hidden layer, p_i is the probability score for each class, C is the total number of classes.

$$L = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log(p_{ic}) \tag{3.2}$$

where y is the ground truth label, and takes a one-hot encoding for multi-class classification problems.

3.2.1 DropColumn

In deep neural networks, neurons tend to co-adapt to overfit the classification tasks. Many neurons are learning correlated/similar functions. In addition, deep

learning models are usually over-parameterized, with millions of trainable parameters but much less training examples. To avoid the overfitting, Hinton et al. [86] proposed dropout, which randomly drops many neurons during the training. With such a simple operation, the dependence between neurons are reduced significantly.

To promote the independence between columns, we propose to stochastically drop many of the columns during training. Specifically, we specify a drop or keep probability of columns. Multiply with the number of total columns, we get the number of columns to be dropped. If the multiplication results in a non-integer value, it is rounded to the closest integer. A random index generator is utilized to get the index of columns to be dropped. During the training stage, those columns do not need to be computed, and the output of those columns is set as zero, which will be concatenated with other columns and fed into the upper module. Since many of columns are dropped, the magnitude of activations is changed, which would lead to the change of weight magnitude. Therefore, similar to Dropout operation, we also need to compensate the change of weight magnitude incurred by DropColumn training. There are two approaches to compensate the weights, either during the training stage or postponed to test/inference stage.

(1) scaling the activations during the inference stage: *i.e.* multiply the activations by p_k when we perform prediction, where p_k is the probability of keeping the neurons (instead of the probability of dropping the columns, *i.e.* $p_k = 1 - p_d$).

Mathematically,

$$a_0 = max(0, h_0) \tag{3.3}$$

$$a = a_0 * p_k \tag{3.4}$$

suppose ReLU activation function is used, the network output a after the scaling as the input for the next layer.

(2) inverting the activations during the training stage: *i.e.* divide the activations by p_k during the training stage. Then at inference stage, predication is performed just as usual without any modification. Mathematically,

$$a_0 = max(0, h_0) \tag{3.5}$$

$$a = a_0/p_k \tag{3.6}$$

Usually the second approach is more widely used, one potential reason is that: one does not need to concern about the magnitude changes during the inference stage, therefore, the same testing function could be used for both models whether trained with or without DropColumn operation. Therefore, we use the second approach to invert the activations during the training stage in the experiments. We call this type of network as Stochastic Columnar Network (SCN), since the network stochastically drops many column during training.

Note that during the inference stage, all columns are kept for either one of the two strategies, *i.e.* the DropColumn operation is only performed during the training stage.

3.2.2 Relationship with Dropout

With the operation of stochastically dropping columns in stochastic columnar network, one might relate it to dropout naturally. In convolutional neural networks, there are two way of performing dropout.

(1) Drop or keep the activations in each feature map independently.

(2) Drop or keep the whole feature map.

Because adjacent activations/pixels are related to each other in the feature maps, the vanilla *i.i.d.* dropout will just result in learning rate decrease. The second approach, which is also known as spatial dropout, however, can help promote

independence between feature maps. Therefore, it helps to regularize the model to prevent co-adaptation of feature maps from overfitting.

Compared with dropout, DropColumn performs dropping at a much higher level, or a more coarse granularity: instead of dropping single neurons in dropout, DropColumn drops many columns/computational sub-graphs. It is compatible to perform dropout within each column, and perform DropColumn simultaneously. However, note that DropColumn operation can only be performed in those networks that do actually have the columnar structure.

In this work, we have also studied the effect of using spatial dropout during training. We found that adding spatial dropout do not help to improve the performance (sometimes even deteriorate the performance a little bit). For example, we have set the dropout ratio as 0.2 (i.e. 80% of neurons are kept) after each convolutional layer, the accuracy dropped from 81.54% to 80.71%. The reason might be that BatchNormalization [51] is used throughout the network, and the convolutional layer does not contain many parameters, which makes the usage of dropout less necessary.

Due to the DropColumn operation, the model needs to run on different computational graph in each iteration. Therefore, we adopt PyTorch to implement the model. With the imperative programming style of PyTorch, we can implement the dynamic computational graph more easily. The DropColumn operation is implemented as a separate module inherited from the nn.module, and will be invoked when the training flag is set as True.

3.3 Experimental Results

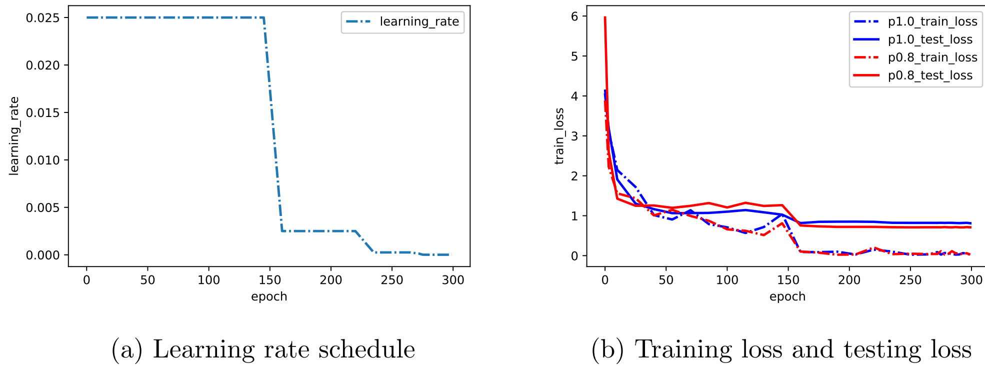
In this section, we show the experimental results compared with state-of-the-art deep learning models. For the ease of comparison, we use two of the most widely used image datasets for empirical validation: the CIFAR10 and CIFAR100 datasets.

CIFAR10 dataset consists of 60, 000 natural images selected from 80 million tiny images by Alex Krizhevsky. It is divided into 50, 000 training set and 10, 000 testing set. There are 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. Each image is 32 by 32 pixels with 3 RGB color channels. Similar to the CIFAR10 dataset, CIFAR100 is also a subset collected from 80 million tiny image database, but contains one hundred classes with a coarse label and several fine labels. The dataset is also split into 50, 000 training images and 10, 000 testing images. Figure 3.3a (b) shows some sample images from CIFAR data set.

For the training setup, we use momentum based SGD algorithm as the optimizer with the default moment parameter of 0.9, the model is trained 300 epochs with a batch size of 64, with an initial learning rate of 0.025, and scheduled learning rate decay at the 150 and 225 epochs, the learning rate will decay further if a plateau appears between multiple epochs. Figure 3.2a shows a typical learning rate schedule used in our experiments.

The models to be trained are composed of three blocks, where each block is composed of 3 modules, each module contains a certain number of columns. If we denote the number of columns as C, and the number of filters in each column as B, a SCN can be instantiated given the two parameters. Note that the total number of parameters is proportional to the product of B and C, therefore, we keep the product fixed in order to keep the number of parameters fixed. In this way, we can see how the number of columns affect the classification performance.

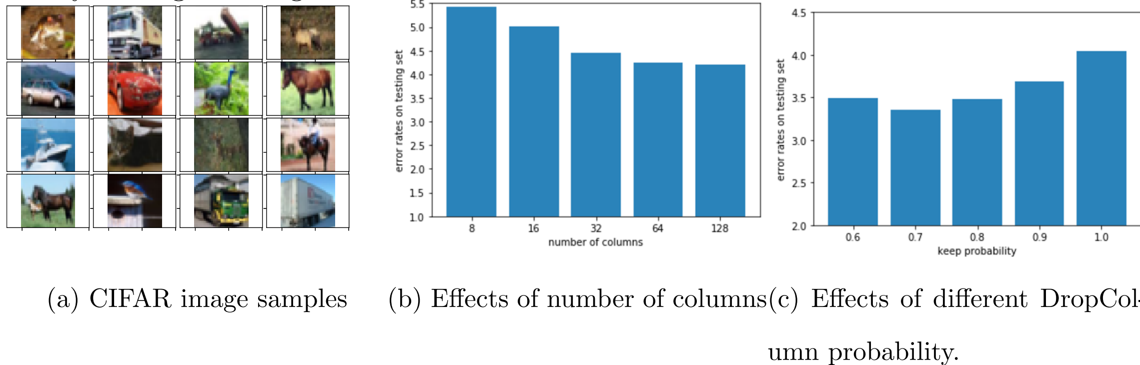
We first show how the number of columns and the DropColumn probability (in the experiment, we use the probability of keeping the columns instead of dropping) affect the classification performance. Then we empirically evaluate the models with different columns and DropColumn probability on the CIFAR datasets.



(a) Learning rate schedule

(b) Training loss and testing loss

Figure 3.2: Learning rate schedule and loss curves with different DropColumn probability during training.



(a) CIFAR image samples

(b) Effects of number of columns

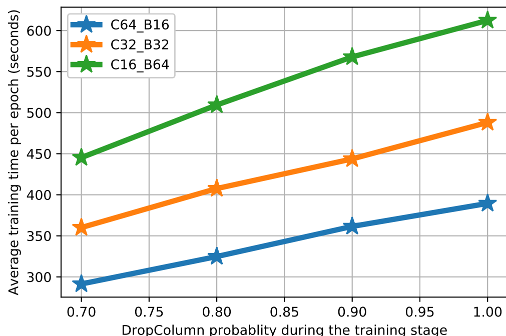
(c) Effects of different DropColumn probability.

Figure 3.3: Effects of different number of columns and DropColumn probability.

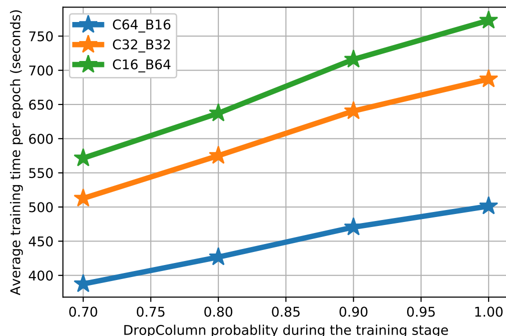
3.3.1 Effect of Number of Columns

We roughly keep the number parameters the same by increasing C (the number of columns and decreasing the base number of filters B. We evaluate the trained model with different number of columns on the test set of CIFAR10 dataset. Figure 3.3b shows the classification performance of models with different columns, where the horizontal axis is the number of columns, and the vertical axis is the testing error rate. We can see that: by keeping the number of parameters fixed, the testing error rate decreases when the number of columns increases. In addition, the benefits of

adding columns diminishes when we keep adding more dividing convolutional filters into more columns: in the beginning, the error rate drops more significantly when dividing large convolutional blocks into few columns. The benefits then become less significant when we further divide those columns into more slim columns (*i.e.* with smaller number of base filters in each column). In practice, programs can exploit the multi-column structure to parallelize the computation of each column into different streams to accelerate the training.



(a) CIFAR10 training time in one epoch



(b) CIFAR100 training time in one epoch

Figure 3.4: Training time in one epoch with different DropColumn probability and different number of columns.

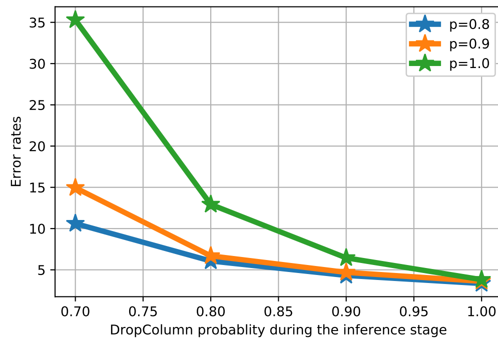
3.3.2 Effect of DropColumn Probability

As we have explained in the above section, performing DropColumn operation during training can potentially improve the classification performance of the trained model. We evaluate the classification error rate on CIFAR10 dataset using different DropColumn probability to train the model while keeping all the other factors fixed. In figure 3.3c, the horizontal axis is the probability of keeping a certain column, and

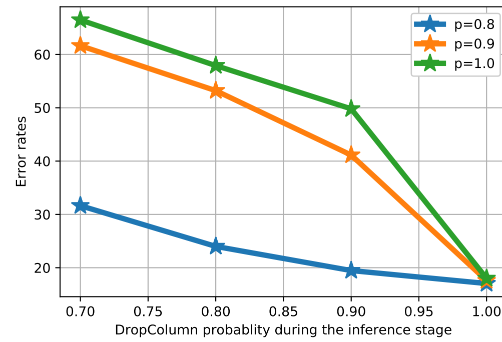
the vertical axis is the classification error rate. We can see that with appropriate rate of dropping the columns, the testing error rate can be reduced. Similar to the effects of adding columns, the error rate drops more significantly when we first start dropping some columns from not dropping any columns. Then the effect become less significant when we keep increasing the DropColumn probability. However, if excessively dropping too many columns, the error rate might go up a little bit. In contrast to dropout where the optimal dropout rate usually lies around 0.5, in the DropColumn operation, the optimal drop rate is usually higher. The potential reason might be that there are much less columns compared to number of neurons, therefore dropping too much columns leads to severe loss of information.

Figure 3.2b demonstrates how the training loss and testing loss decrease during the 300 epochs of training with different DropColumn probability. From the figure we can see that the training loss converges to a similar value, but the testing loss of $p = 0.8$ (*i.e.*, drop 20% of columns during the training stage) is lower than that of $p = 1.0$ (*i.e.*, without dropping columns during the training stage). This shows that DropColumn operation during the training stage helps to reduce overfitting.

On the other hand, properly increasing the DropColumn probability not only improve the classification performance, but also reduces the training time a lot. Figure 3.4 shows the training time in one epoch with different DropColumn probability and different number of columns. From the figure we can see that: dropping some columns reduces the training time a lot , due to the fact that many columns do not need to be computed during training. In addition, using more columns also reduce the training time, because many columns can be computed in parallel. Due to the decoupled computation, the training time can be further reduced with more delicate engineering tricks.



(a) CIFAR10 error rates



(b) CIFAR100 error rates

Figure 3.5: Error rate of different dropping probability during the inference stage of a trained SCN network. The legend refers to the keeping probability of the DropColumn operation during the training stage.

3.3.3 Classification Performance

In this subsection, we empirically validate the effectiveness of the proposed neural network architecture. Table 3.2 and 3.3 show the classfication error rates on the testing set of the two data sets, respectively. We only tested till to dropping 30% percent of columns since we found that excessive dropping of columns would lose too much information for the classification task. Note that in the table we are using keeping probabilities, *e.g.* p=0.7, keeping 70% percent, and p=1.0 means without dropping any column. When we increase the number of columns, the number of feature maps in each column is reduced in order to keep the total number of parameters the roughly the same.

From the two tables, we can conclude that: under the same parameter budget, better classification performance can be achieved when adding more columns . In addition, the performance improves significantly when we go from without dropping any column to dropping columns. As we have shown before, dropping columns can

lead to speed up for training. Therefore, the model can be trained both faster and better.

SCN demonstrates state-of-the-art classification performance on the widely used image recognition benchmarks. Table 3.4 shows some state-of-the-art performance on CIFAR10 and CIFAR100 datasets. We can see that SCN achieves better accuracy than most of the methods on the standard training/testing split. NSANet gets the best performance, but the network is searched in a huge design space by a lot of cloud TPUs, which is computationally prohibitive for most AI researchers.

3.3.4 Ensemble Property

We show the classification performance of SCN using different dropping probability during *inference* stage in figure 3.5. From the figure we can see that the testing error rate increases with higher dropping probability (*i.e.*, lower keeping probability). This is expected since dropping more columns during inference loses information for the classification. This is in contrast with dropping columns during training, which could promote the independence between columns and helps improve the classification performance. The testing error increases more significantly if we drop more and more columns by increasing the dropping probability.

Note that the error rate increases *smoothly* when we start dropping columns during inference stage (smaller gap of DropColumn probability can be used to smooth the curve). This is in stark contrast with traditional architectures like AlexNet, ZFNet or VGG network where if we drop some intermediate blocks, the whole classifier amounts to random guess. This is because those traditional architectures adopt sequential blocks of convolutional operations, therefore, the upper layers depends heavily on the previous layer's activation distribution. Note that we are not recommending dropping columns during the inference stage, but to show that the learned

columns are independent in a certain degree, thanks to the DropColumn operation which promotes the independence between columns. Since each column is a small computational graph, the whole stochastic columnar network can be viewed as an implementation of intra-model ensemble which enables efficient feature reuse and composition. Residual networks can also be viewed as an ensemble of networks, but in each residual block there are only two paths for the information to pass through. In a SCN, there are C (number of columns) paths in each module. This helps to boost the ensemble property compared with residual networks.

We can also see that when the same number of columns are dropped during the inference stage, the error rates of the SCN model trained with $p = 0.8$ is much smaller than that of the model trained with $p = 1.0$ (*i.e.*, without dropping any column during training). With the same number of columns being kept during the inference stage, models trained with a lower keep probability (dropping more columns) perform much better. This justifies that the columns are pushed to learn more independent features when dropping more columns during the training stage. Therefore, this approach boosts the parameter efficiency since the learned features are much less correlated and less redundant. This accounts for the better classification performance with the same amount of parameters compared with state-of-the-art models.

3.4 Conclusion

We propose a new class of neural network architecture that exhibits ensemble property in a single network. The network adopts a basic information processing unit of a small computational graph which we call column. Through hierarchically composing those columns into a network, and fusing the information output from columns, we achieve a stochastic columnar network that can efficiently enable feature reuse and models complex compositionality. To prevent overfitting, DropColumn operation is

performed during training stage. We show that the classification performance improves upon state-of-the-art neural network models, and in the mean time could save the training time with appropriate DropColumn probability. Interestingly, we can see that the error rates increases smoothly when DropColumn operation is performed at the inference stage, which justifies the ensemble property of the proposed model.

SCN introduces a new level of indirection into and hopefully could inspire the design of neural network architectures. Further research can be done along the direction of more sophisticated design of base column, information fusion between columns, more sophisticated regularization techniques for promoting independence between columns, etc.

stage	layer	filter, stride	repetition
stage_0	conv_0	3x3, 64, stride 1	
stage_1			3
	conv_10	3x3, B x C, stride 1	
	conv_11	3x3, B x C, stride 1	
	conv_12	3x3, 256, stride 1	
stage_2			3
	conv_20	3x3, B x C, stride 2	
	conv_21	3x3, B x C, stride 1	
	conv_22	3x3, 512, stride 1	
stage_3			3
	conv_30	3x3, B x C, stride 2	
	conv_31	3x3, B x C, stride 1	
	conv_32	3x3, 1024, stride 1	
stage_4			
	FC	1024 x nLabels	

Table 3.1: Architecture of stochastic columnar network.

Keep prob / Arch config	C64_B16	C32_B32	C16_B64
p=0.7	3.52	3.61	3.69
p=0.8	3.41	3.48	3.49
p=0.9	3.53	3.52	3.58
p=1.0	3.76	3.81	3.85

Table 3.2: Classification error rates on the CIFAR10 data set with different instantialization of stochastic columnar network. "Cxx" denotes number of columns in the network, and "Bxx" denotes number of feature maps used in each column. During training different probabilities are used for the DropColumn operation.

Keep prob / Arch config	C64_B16	C32_B32	C16_B64
p=0.7	16.69	16.97	17.84
p=0.8	16.61	17.09	17.44
p=0.9	16.65	17.17	17.71
p=1.0	17.75	18.46	18.76

Table 3.3: Classification error rates on the CIFAR100 data set with different instantialization of stochastic columnar network.

	CIFAR10	CIFAR100
FitResNet [87]	5.84	27.66
ResNet [53]	4.69	22.68
Stochastic Depth Net [84]	5.23	24.58
ResNet in ResNet [88]	5.01	22.9
WideResNet [79]	4.17	20.5
DenseNet-BC [54]	3.62	17.5
FractalNet [87]	4.6	23.73
ResNeXt [89]	3.58	17.31
Dual Path Network [90]	3.65	-
NASNet (best) [91]	2.4	-

Table 3.4: State-of-the-art performance achieved by other models: error rates on the CIFAR10 and CIFAR100 datasets

CHAPTER 4

Attacks and Defenses of Deep Learning Models

4.1 Introduction

In the era of big data, many industries rely more and more on data to achieve automate tasks. Deep learning is transforming the transportation and health care industry. Companies building self-driving cars rely heavily on deep learning models for image recognition/object detection. More and more companies are using deep learning for cancer diagnostics in order to alleviate the insufficient supply of experienced doctors. With the wide adoption of deep learning models in each aspects/verticals of our daily lives, it is of paramount importance to study the security of deep learning models, and the potential defense mechanisms to make the model more robust to various types of attacks.

Attacks can happen either in the training phase or testing phase. In the training phase, attackers can try to contaminate the training data to make the model behave as they want. In the testing phase, attackers can alter the testing data, or generate some data to deceive the model.

To begin with, we first introduce on the potential of attacks during the training phase. One underlying assumption is that the input data and their corresponding labels are faithful and correct. However, this is not always the case in real world scenarios. In fact, it is common that data/labels may contain noise. The source of noise can be roughly categorized into two aspects: unintentional and intentional. Unintentional noise comes from the process of data labeling. For example, nowadays computer vision API providers distribute labeling tasks to human labellers on

MTurk (Amazon Mechanic Turk)/CrowdFlower or other crowd sourcing platforms. The labeling quality depends heavily on the carefulness of labellers and the quality assurance of crowd sourcing platforms. In this process, the human labellers can be careless or tired or lack of expertise to correctly label some data. In addition, the goal of the labeller is usually to finish as many tasks as possible to maximize their rewards, therefore noisy labels is pretty hard to be removed due to the large amounts of labeling task assigned to human labellers. On the other hand, intentional noise may exist in the training data. In the future, more and more services will rely on machine learning models, therefore there is big economic incentive for people to inject noise labels to training data, trying to compromise the training of the model. With data-driven machine learning becomes more and more popular, data is deemed as one of the most valuable property of a company/person. Therefore, people do not want to share data with outside institutes. In the future, it will be more popular for federated learning/multi-party learning [92], where training data are distributed among many parties due to that people want to protect the privacy/security of their training data. it is possible that some shards of training data are compromised by people for economic returns, or even vicious people pretend to be part of the training cohort. Due to the reasons mentioned above, we believe it is important to make deep learning models robust to noisy labels. Since the training of neural network totally depends on the error gradient back propagated from the output loss layer, in this work, we explore a robust loss function to replace the commonly used loss functions in neural network for classification problems. This approach is very intuitive and easy to be implemented since only the last layer of the neural network need to be changed.

On the other hand, we study potential attacks during the testing phase. Researchers have studied adversarial attacks where the input can be perturbed with minimum magnitude that human can not even notice, but the deep learning model

will recognize it as the other class. Usually this process is guided by the gradient of the output with respect to the input data. There have been many works elaborate on this type of adversarial attacks[73, 93, 94, 95]. Those type of techniques usually start from a certain input example and gradually evolve the input to the direction where the output of the neural network will be the target class the attackers want to achieve.

Apart from the above mentioned adversarial attack, we think it is important to study whether deep learning models are vulnerable to access attacks when deep learning based models are deployed for access control in security critical applications (for example, in bank vault, iPhone's i-Touch fingerprint recognition system, or center control room of an organization). Access attack is a type of attack where people have not been granted access to a certain software/system/places compromise the security guard (the deployed model) to gain access to privileged information/rights illegally. Once the deployed model is being compromised by vicious people, enormous loss can be incurred since attackers can do whatever they want in the system. Especially for those biometrics based system, users are much harder to alter them compared with password based access control systems. Therefore once biometrics based systems are compromised by attackers, it might be harder to take approaches for loss stopping. Note that the goal of this type of attack is to generate the object (for example image in vision based system) that serve as the credential to enter into privileged section/area. Note that we do not care about whether the generated image has any visual regularity to human. In this paper, we will demonstrate a technique that can generate images which will be recognized by deep learning models with 100% certainty to be any class that the attackers want.

We summarize the takeaways from this paper as follows:

1) We tested the most commonly used classification loss functions for deep learning with the presence of different ratio of noisy labeling, *i.e.* cross entropy loss and hinge loss. According to the experiment, hinge loss is not robust to the training attack of noisy data, while cross entropy is much more robust to this type of training attack than hinge loss.

2) We proposed to use a new type of loss function for deep learning: the correntropy loss. Empirical results suggest that correntropy is more robust to noisy input. Therefore, it serves as a good surrogate loss function for defending against noisy data training attack.

3) We proposed a new type of algorithm to generate adversarial samples for access attack during the evaluation/inference stage. Experiments show that neural networks are vulnerable to this type of evaluation attack. We urge that the academic community should pay more attention to the security of deep learning models, given all the buzz from industry and wide adoption of deep learning.

We organize the paper as follows: in the next section, we introduce some preliminary knowledge on commonly used loss functions for classification. After that, we introduce our new robust loss function. Then we demonstrate that deep learning models are vulnerable to access attacks on commonly used computer vision benchmark data sets.

4.2 Preliminary

In this section, we introduce preliminary knowledge before diving into the newly proposed approach.

A deep learning model is usually formed by stacking many layers of non-linear mapping. Within the non-linear transformation, it incorporates prior knowledge about the inherent property of data modality, *e.g.* modeling spatial correlation with

convolutional mapping, and modeling the context of sequential structure with recurrence. After many layers of non-linear transformation, the output hidden activations are used as features to input to a linear classifier. The output of the last layer is summarized into probability scores by softmax function in Eq. (4.1). Usually cross-entropy in Eq. (4.2) between the probability scores and ground truth is used as the loss function for classification problems. The error of the cross-entropy loss back propagates through out the whole network to update the trainable parameters in the model.

$$p_c = \exp(z_i) / \sum_{c=1}^{C} (\exp(z_i)) \tag{4.1}$$

where z is the logits score in the final hidden layer, p_i is the probability score for each class, C is the total number of classes.

$$L = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log(p_{ic}) \tag{4.2}$$

Due to the popularity of SVM (support vector machine) before the deep learning era, researchers have tried to use the loss function of SVM (hinge loss) as a loss function [96]. The motivation of hinge loss is to learn a large margin classifier. Therefore, hinge loss does not penalize the loss function when the margin (the product of the prediction and the ground truth) is larger than a certain threshold, and the loss increase linearly when margin falls below the specified threshold. Mathematically, Hinge loss for binary classification problem can be written as follows:

$$L = -\sum_{i=1}^{N} \max(0, 1 - y_i * s_i) \tag{4.3}$$

$$s = f(W, x_i) \tag{4.4}$$

where x_i is the i-th data point, y_i is the ground truth label, s_i is the prediction of the model, W is the model parameter we want to learn, and N is the total number of training data.

For the multi-class hinge loss, we can formularize it as follows:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \tag{4.5}$$

where s_j represents the model prediction score for the j-th class, and s_{y_i} represents the score for the ground truth class.

4.3 Training Attack

In this section, we study one type of training attack. As mentioned in the introduction, this type of attack is implemented by injecting noise to training data. Therefore, the model training procedure should be robust to noise in the training data. The naive approach might be examine the data one by one, and correct the labeling of noisy cases, or even just remove all of those annotations. However this approach is not scalable, and in the real world it might be not even realistic (*e.g.* in the federated learning setting, we do not have a centralized copy of the whole training data nor do we have access to all shards of training data.) Therefore, it is not viable to solve the problem from the input of neural network.

The goal of machine learning models is to extract regularities from training data, however, noisy data would introduce many irregularities to the training data, therefore making the learning harder to disentangle key properties/attributes that decides the category of objects. When the prediction deviates from the ground truth, many commonly used loss functions (e.g. least square loss) incur a much larger loss value. Therefore, with the presence of noisy data, the loss will be dominated by noisy points, making the model fail to learn. Least square estimator models the conditional

mean of the output, when the data does not follow gaussian distribution, it can be easily skewed with few noisy data if the noise is large enough.

4.3.1 Method: Deep Modal Regression

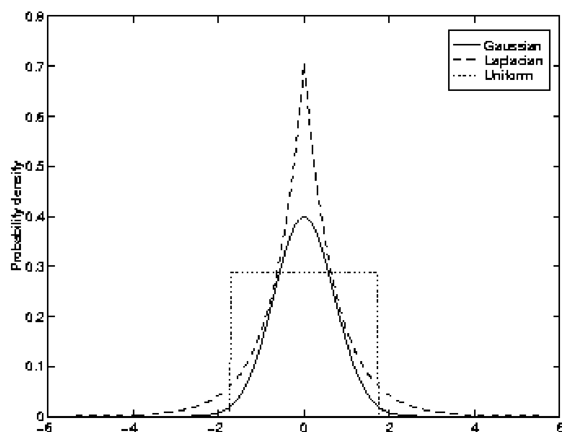


Figure 4.1: Noise distribution.

In this work, we propose a principled approach to alleviate the noisy training data problem. Without the hustles and bustles to remove/correct the data labeling in the input space, our proposed method can automatically adapt to the heavy noise in the data.

When we model the data distribution, usually we assume a conditional data generation model as follows:

$$Y = f(X) + \epsilon \tag{4.6}$$

The learning process is to learn the function f based on certain assumptions on the distribution of noise ϵ. The most common assumption is Gaussian distribution on the noise, where the conditional distribution is modeled by conditional mean, and it works reasonably well when data are sampled from a Gaussian distribution. However

with the presence of human injected spiky noise, the Gaussian noise assumption fails to hold, conditional mean is not valid to characterize the data generation process.

Some other usually used way to characterize the conditional distribution are conditional median (which assumes Laplacian distribution of data, and can be used to model long-tail distribution), and conditional mode. Conditional mode [97] offers a very appealing property in which it does not assuming any specific underlying distribution. When human injected noise exists, the conditional mode is affected least. Therefore, we propose to characterize the conditional distribution via modeling conditional mode.

The mode of a distribution lies on where the probability density peaks. Mathematically, the modal regression function can be written as follows:

$$f^*(x) = arg \max_{y \in R} P_{Y|X}(y|X = x) \tag{4.7}$$

As proved in Thereom 3 of [97], modal regression is the maximizer of the likelihood criterion:

$$R(f) = \int_x p_{Y|X}(f(x)|X = x)dp(x) \tag{4.8}$$

The empirical estimation of $R(f)$ can be obtained by kernel density estimation,

$$R^\sigma(f) = \frac{1}{l\sigma} K_\sigma(y_i - f_i, 0) = \frac{1}{l\sigma} \sum_{i=1}^{l} \phi(\frac{y_i - f_i}{\sigma}) \tag{4.9}$$

where ϕ is the representing function.

With a Gaussian kernel, it can be written as:

$$R^\sigma(f) = \frac{1}{l\sigma^2} \sum_{i=1}^{l} exp^{-\frac{(y_i - f_i)^2}{\sigma^2}} \tag{4.10}$$

Or equivalently we can minimize the following loss function:

$$L = \sigma^2 \sum_{i=1}^{l} (1 - exp^{-\frac{(y_i - f_i)^2}{\sigma^2}}) \tag{4.11}$$

This loss function is usually known as the correntropy loss, and it has gain popularity regression tasks to deal with non-gaussian noise. To model the complex non-linearity between inputs and outputs in perception problems like image recognition, we constrain the hypothesis space f on the function space spanned by the convolutional neural network (CNN) function family. Therefore, we dub the method deep modal regression. CNN is well suited for dealing with translational invariance in image recognition. The CNN learns a function that maps the input from image space to prediction probability scores, then the error is evaluated with correntropy loss. With this differentiable loss function, the optimization can follow standard stochastic gradient descent approach or any of its variants.

MNIST	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.6	0.7
CrossEntropy	99.23	99.11	98.99	98.75	98.66	97.93	97.24	96.91	95.74	93.86	92.77	86.67	66.42
HingeLoss	99.25	99.09	99.07	98.49	98.07	97.29	96.89	92.15	89.05	82.45	63.09	33.34	12.28
CorrEntropy	99.22	99.16	99.12	98.90	98.82	98.05	97.61	97.03	96.72	94.48	93.56	88.63	70.61

Table 4.1: Classification accuracy of different level of noise rates on the MNIST dataset.

4.3.2 Experimental Results

To evaluate the effectiveness of the proposed loss function for deep learning, we run experiments on the two widely used benchmark datasets, MNIST and CIFAR10 datasets. Standard networks for each dataset are used, and the networks are trained

CIFAR10	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.6	0.7
CrossEntropy	91.67	91.32	90.89	90.46	90.43	89.85	89.65	89.63	88.35	87.56	86.64	83.24	79.65
HingeLoss	91.28	90.85	72.85	56.7	56.43	46.52	37.63	29.18	27.18	23.46	20.68	12.42	10.00
CorrEntropy	91.63	91.30	91.03	90.98	90.87	90.38	90.27	90.15	89.93	89.71	88.57	86.15	83.85

Table 4.2: Classification accuracy of different level of noise rates on the CIFAR10 dataset.

with cross entropy (CE), hinge loss, and correntropy, respectively. Error rates on the standard test test are reported.

The architecture for MNIST experiment is a standard LeNet architecture: the original input image of size 28x28 go through two interleaved series of operations: 5x5 convolution, ReLU activation, 2x2 max-pooling operation. The first convolutional operator has 32 filters, and the second convolutional operator has 64 filters. The feature map after the second max-pooling operation is then summarized into 1024 neurons, with another fully connected layer transforming the hidden features into 10 output classes. Adam optimizer [33] is used with a learning rate of 10^{-4}. We train the network with 20000 iterations, with a batch size of 50 samples.

For the CIFAR10 experiment, we use an architecture that has three interleaved blocks, where each block consists of two consecutive padded convolutional layers followed by a 3x3 max-pooling operator with a stride 2. The feature maps are summarized by two fully connected layers of neurons 1024 and 512 before feeding into the classification layer with a output of 10 classes. Random cropping and horizontal flipping are applied as data augmentation to alleviate the overfitting problem. A weigh decay with regularization strength of 0.0004 is applied on all parameters. Adam op-

timizer is used for training, with an initial learning rate of 10^{-2} and the learning rate will decay when the objective do not decrease for a few epochs with a decay rate of 0.31. The batch size is 128, and the network is trained with 200 epochs.

We train the networks with different level of noise rate, from 0.05 to 0.7, with a step size of 0.05. The noise is added by randomly perturb certain percentage of the ground truth labels. Table 4.1 and 4.2 show the classification accuracy on the MNIST and CIFAR10 dataset, respectively. Note that all the other configurations are the same except for the loss being used on the top of the neural networks. From the tables, we can conclude that:

1) With the increase rate of noise level, the error rates on the testing set on both datasets increase. This is expected since the randomly perturbed label make the neural network harder to learn the correct mappings between input image and the output label.

2) Cross entropy is pretty robust to noise in the training data. Even with half of the training data are random noise, we can still achieve above 90% accuracy. The reason might be that input are surpressed with the range [0, 1] by the softmax function. Correntropy is more robust to noise in the training data, due to its ability to handle non Gaussian noise.

3) While the hinge loss can achieve similar accuracy with no noise, the accuracy drops quickly with the increase of noise rate, in contrast to cross entropy and correntropy loss functions. The loss is dominated by noisy data, making it hard to extract regularities from data. When the noise exceeds certain level, the model can not be well trained and basically reduces to random guess.

In a nutshell, when the training data is contaminated by attackers with noise, correntropy can be used as a surrogate loss function to defend against this type of attacks.

4.4 Evaluation Attack: Malware

4.4.1 Related Work

In this section, we will review previous works related to adversarial samples. First we describe previous papers related to adversarial learning in general. We then discuss papers related to adversarial learning to either create or detect adversarial malware samples.

Adversarial Attacks: Adversarial attacks and defense for deep learning models have been a popular research topic recently due to the wide range applications of deep learning models. Goodfellow, *et al.*, [98] demonstrated that deep learning models can be fooled by crafting adversarial samples from the original input data by adding a perturbation on the direction of the sign of the model's cost function gradient. This method is known as the *fast gradient sign* method. For images which are considered in their paper, the algorithm computes the gradient information once and perturbs all of the pixels to a certain amplitude. Since the fast gradient sign method requires continuous features, it is not applicable to our malware classification data which is composed of sparse binary features.

Papernot, *et al.*, [99] proposed another algorithm for crafting adversarial samples, which iteratively perturbs the input along the dimension with largest gradient saliency. The algorithm perturbs one input feature in each iteration until the altered sample is misclassified into the desired target class. The goal of this method is to use the minimum perturbation to the original sample such that the perturbation is not perceivable by humans, but is misclassified by a machine learning model. This algorithm has a larger computational complexity compared to the fast gradient sign method in [98], because in each iteration, the algorithm needs to compute the derivative of the model's output probability with respect to the perturbed sample.

In most cases, users do not have the knowledge about the architecture and parameters of the trained model that are deployed into a service. Deployed models are known as black box models due to the fact that the attacker does not have any information beyond the outputs of the model on input queries. Papernot, *et al.*, [73] proposed a method based on model distillation to craft adversarial attack samples on black box models. The authors in [73] found that adversarial samples are transferable among models, *i.e.*, the adversarial samples crafted for one model can also mislead the classification of other models. They use model distillation techniques to compromise an oracle hosted by MetaMind. In this case, the oracle is a defensive system where the users only know the input and output, but they do not know anything about the architecture of the model.

A defensive strategy using model distillation is proposed in [100]. Model distillation is performed by using the soft labels (prediction probability on a trained neural network) as the label of training samples to train a new deep neural network. They found that distillation captures class correlation, and the model trained on soft labels is more robust than one trained using hard labels. In this case, a hard label is specified as the discrete class label. The authors also found that using a high temperature in distillation training enforces smoothness of the model, which could make the model more robust to adversarial samples. Using the high temperature distilled model, the changes in adversarial samples have much less impact on the classification of the model.

Several authors [101, 102, 103, 104] have proposed using an ensemble of models to avoid different type of malicious attacks. For example, the authors in [102] proposed using an ensemble of models to improve the privacy of deployed models since attackers will only be able to obtain an approximation of the target prediction function. Kantchelian, *et al.*, [101] proposed two algorithms for evasion attacks on

tree ensemble classifiers, like gradient boosted trees and random forests. However, each tree classifier is very weak compared with a full-fledged neural network.

Malware Classification: Several deep learning malware classifiers are proposed in [105, 106, 107, 108, 109, 110]. The first study of deep learning for a DNN malware classifier was presented in [105]. Similar to our results, the authors found that a shallow neural network slightly outperformed a DNN on dynamic analysis-based malware classification. Saxe, *et al.*, studied DNNs in the context of static malware classification in [106]. Huang and Stokes proposed a deep, multi-task approach for dynamic analysis which simultaneously tries to optimize predicting a) if a file is malicious or benign and b) the file's family if it is malware or returning a benign label in the case it is clean. In [107], the authors propose a two-stage approach where the first stage employs a language-model, using a recurrent neural network (RNN) or an echo state network (ESN), to first learn an embedding of the behavior of the file based on its system call events. This embedding then serves as the features for a DNN in the second stage. Athiwaratkun, *et al.*, [109] explored similar architectures for deep malware classification using long short-term memory (LSTM) or a gated recurrent units (GRU) for the language model, as well as a separate architecture using a character-level convoluation neural network (CNN). In [110], Kolosnjaji, *et al.*, propose an alternative model also employing a CNN and an LSTM.

Several authors have proposed methods for creating adversarial malware samples. In [111], Xu, *et al.*, propose a system which uses a genetic algorithm to generate adversarial samples which can be mispredicted by a classifier. The system assumes access to the classifier's output score. The authors demonstrate that their system can automatically create 500 malicious PDF files that are classified as benign by the PDFrate [112] and Hidost [113] systems.

Hu and Tan [114] propose a generative adversarial network (GAN) to create adversarial malware samples. In their work, the authors assume that the attackers know the features which are employed by the malware classifier, but they do not know the classification model or its parameters. They use static analysis where the features are API calls and a sparse binary feature is constructed to indicate which APIs were called by the program. Furthermore, the authors assume that the prediction score from the model is reported from the malware classification model.

Grosse, *et al.*, [115] study the distillation defense for *static* analysis-based malware classification. Similar to this paper, the authors assume that the attacker has access to all of the deep learning malware classifier's model parameter. In our work, we also consider the distillation defense for *dynamic* analysis-based malware classification. In addition, we evaluate the ensemble defense and introduce the regularization defense for a dynamic malware classifier. In another recent paper, Grosse, *et al.*, [116] add a separate class for adversarial samples and propose a statistical hypothesis test to identify adversarial samples.

4.4.2 System Overview and Threat Model

In this section, we provide a high-level overview of the defender's training and evaluation systems as well as the threat model which includes the assumptions about the attacker and the detection strategies.

System Overview: The system overview is depicted in Figure 4.2. The original data for this study was collected by scanning a large collection of Windows portable executable (PE) files with a production version of a commercial anti-malware engine which had been modified to generate two sets of logs for each file including unpacked file strings and system API (application protocol interface) calls including their parameters. Before an unknown file is executed on the actual operating

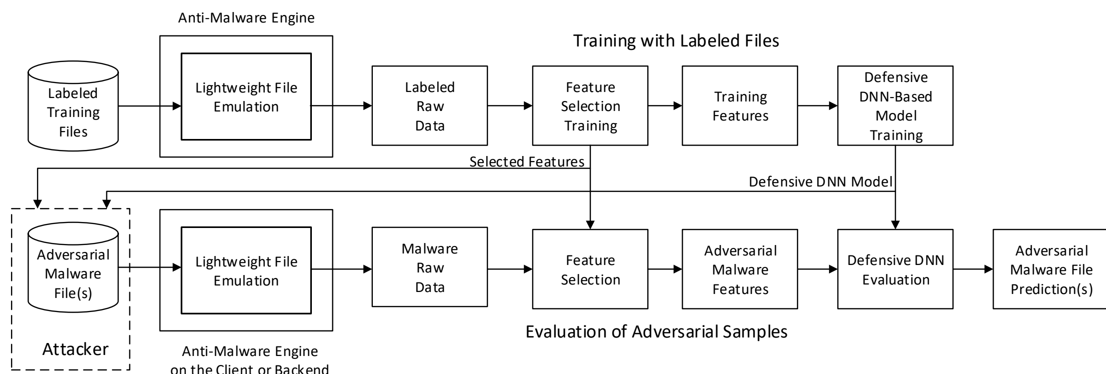


Figure 4.2: Overview of the adversarial attack and defense of a dynamic analysis-based malware classification system.

system, the anti-malware engine fist analyzes the file with its lightweight emulator which induces the dynamic behavior of the file. The first log file that is generated during emulation is a set of unpacked file strings. Typically, a malware file is packed, or encrypted, to make it difficult to reverse engineer by malware analysts. During emulation, text strings, which are included in the PE files, are unpacked and written to the system memory. The emulator's system memory is next scanned to recover null terminated objects which include the original text strings. In addition, the engine also logs the sequence of API calls and their parameters which are generated during execution. This sequence provides an indication of the dynamic behavior of the unknown file.

From these two log files, we generate three sets of sparse binary features for our deep learning models. We consider each distinct, unpacked file string as a potential feature. Two sets of features are derived from the system call data. First, we generate a potential feature for each distinct value of an API call and input parameter value

for a specific input position. Second, we generate all possible combinations of API trigrams (*i.e.*, (k) API call, (k+1) API call, (k+2) API call) as a feature which represents the local behavior of the file.

There are tens of millions of potential features which are generated from the three sets of raw features. Since the neural network cannot process this extremely large set of data, we utilize feature selection using mutual information [117] in order to reduce the final feature set to 50,000 features. If any of these final features are generated during emulation, the corresponding feature will be set to 1 in the sparse, binary input feature for that file. This set of feature vectors is then used to train the deep learning model which has been enhanced to defend against adversarial attacks.

We assume the attacker has knowledge of the selected features and the trained DNN model. With this information, they are able to craft adversarial malware samples which are processed by the anti-malware engine and the identical inference engine. The goal of the attacker is for their malware sample to have a benign prediction.

Threat Model: We follow earlier work [100] and assume that the attacker has access to all of the model parameters and operating thresholds. For an ensemble classification system, we assume that the attacker has obtained all parameters and threshold values for each classifier in the ensemble. This is the most challenging scenario to protect. Once the attacker has successfully obtained of the parameters for the model or ensemble of models, we assume they implement the Jacobian-based strategies proposed in [100, 99] to determine the ranking of important malicious and benign features.

Modern anti-malware systems consist of two main components: an anti-malware client on the user's computer and a backend web service which processes queries from all of the individual anti-malware clients. It would be difficult and most likely require a successful spearphishing campaign to obtain any classification models running in a

backend web service. However it would be much easier to reverse engineer a malware classifier's parameters and threshold values running on a user's client computer.

All of the data is generated by the anti-malware engine's emulator running in a virtual machine without external network access. We assume that malware does not detect that it is being emulated and halt all malicious activity. We further assume that the malware does not alter its behavior due to the lack of external internet access.

Finally, in several of the attack strategies proposed in the next section, we assume that the malware author can remove key features related to malicious activity (*i.e.*, malware features) while maintaining its ability to achieve the desired malicious objective. Since most malware is either packed or encrypted, our analysis is based on the *behavior* of the malicious code, and we use a dataset of over 2.3 million malware and benign files in this study, it is impossible for us to actually modify the malware to remove malicious features. Removing important malicious content may actually transforms the malware into a benign file. However, attackers often employ metamorphic strategies to use alternate code paths to reach the desired malicious objective [118]. In order to continue to perform its desired malicious behavior, we assume the attacker has the ability to engineer an alternative attack strategy. For example, instead of writing a value to the registry, the attacker may choose to instead write important data to a local file or memory. In other cases, the attacker may re-implement key functions of the operating system. We, therefore, assume the attacker has the ability to effectively remove malicious features by re-implementing the key pieces of the malware's code related to the most important malicious features.

4.4.3 Baseline DNN Malware Classifier

Before discussing the strategies for crafting and defending against adversarial samples, we first review the baseline deep neural network malware classifier which is

illustrated in Figure 4.3. We follow earlier work in [105] and use a sparse random projection matrix [119] to reduce the input feature dimension from 50,000 to 4,000 for the DNN's input layer. The sparse random projection matrix R is initialized with 1 and -1 as

$$Pr(R_{i,j} = 1) = Pr(R_{i,j} = -1) = \frac{1}{2\sqrt{d}} \tag{4.12}$$

where d is the size of the original input feature vector. All hidden layers have a dimension of 2000. Following [108], we use the rectified linear unit (ReLU) as the activation function, and dropout [120] is utilized with the dropout rate set to 25%. All inputs to the DNN are normalized to have zero mean and unit variance. The output layer employs the softmax function to generate probabilities for the output predictions:

$$softmax(x) = exp(x) / \sum_{i=1}^{c} (exp(x_i)). \tag{4.13}$$

4.4.4 Crafting Adversarial Samples

In this section, we describe six iterative strategies for crafting adversarial samples. Essentially, the attacker's strategy is to first discover features that have the most influence on the classification output, and then alter their malware to control these features. The Jacobian, which is the forward derivative of the output with respect to the original input, has been proposed in [100, 99] as a good criterion to help determine these features. For a malware classifier, the prediction output indicates that an unknown file is either malicious or benign. Thus, the attacker's goal is to alter (*i.e.*, perturb) the important features such that the malware classification model incorrectly predicts that a malicious file is benign. To compromise the malware classifier, the attacker can modify their malware to decrease the number of features that

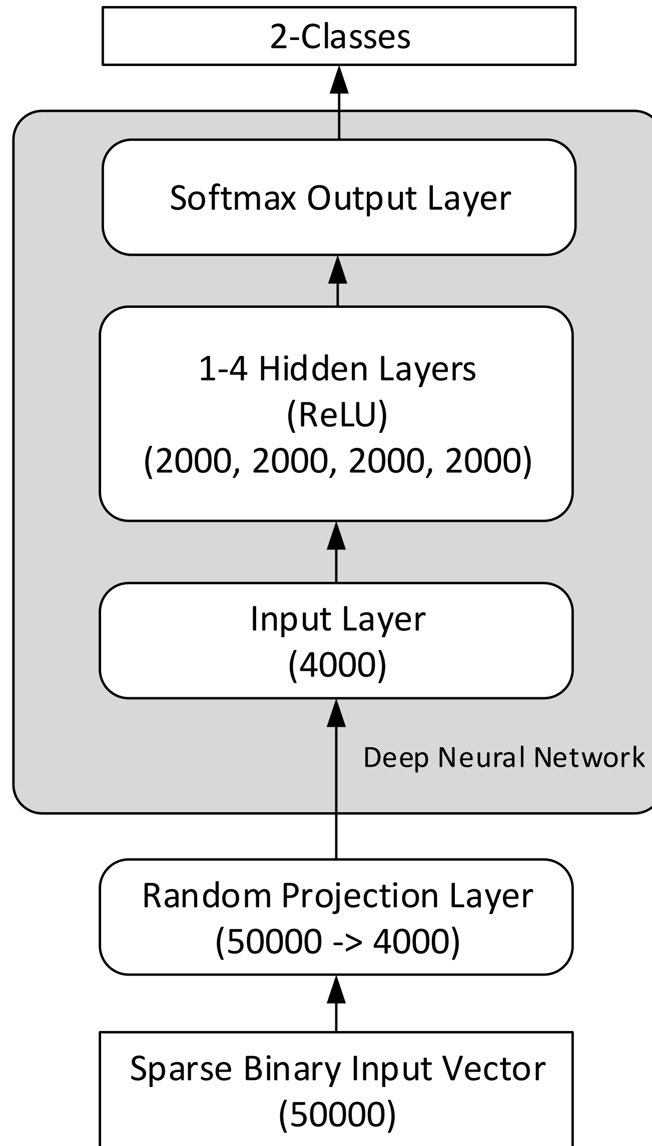


Figure 4.3: Model of the baseline deep neural network malware classifier.

are important for a malware prediction, increase the number of features that lead to a benign prediction, or both.

For each iterative attack strategy that simulates an attacker modifying their malware, we alter one feature during each iteration and then re-evaluate the Jacobian with respect to the perturbed sample. We analyze six strategies to craft adversarial samples. The first three methods use the Jacobian information [100, 99] to identify which features to alter:

(1) dec_pos, *i.e.*, disabling the features that would lead the classifier to predict that an unknown file is malware based on the Jacobian of the classification output with respect to the original input features. We define a feature to be a positive feature if the Jacobian with respect to the feature is positive. We call these features *positive features* since they are the key indicators of malware behavior.

(2) inc_neg, *i.e.*, enabling the features that would lead a classifier to predict that an unknown file is benign. These features are called *negative features* with respect to the malware class. A negative feature has a positive Jacobian with respect to the benign class.

(3) dec_pos + inc_neg, *i.e.*, alternatively disabling one positive feature for one iteration and then enabling one negative feature in the next iteration. This strategy investigates whether there is any synergy between removing malicious content and adding benign features in a round robin fashion.

In contrast to the above methods that use the Jacobian information, we also include three, similar "randomized" strategies that do not use the Jacobian for comparison. For these additional algorithms, we randomly select positive features to disable or negative features to enable instead of selecting them using the rank of the Jacobian's forward derivatives. Thus, the additional strategies include: (4) randomized dec_pos, (5) randomized inc_neg_random, and (6) randomized dec_pos + inc_neg.

4.4.5 Defensive Methods

In this section, we review three methods for defending against adversarial attacks including the distillation, ensemble, and weight decay defenses. Although the distillation and ensemble defenses have been previously proposed, the weight decay defense is new. Only the distillation defense has been previously explored to defend against adversarial attacks in malware detection applications, and this work was done in the context of static malware classification [115].

Distillation Defense: The first defense we study is the distillation defense [100, 115] where the model model is trained using knowledge distillation. As discussed previously, knowledge distillation is typically used to distall the knowledge learned from a large model into a smaller network making the smaller model more efficient in terms of its memory, energy, or processing time in deployment. However, in adversarial learning, the goal is to make the distilled model more robust to adversarial perturbations, instead of focusing on compressing the network size.

The motivation of using model distillation as a defense mechanism is that with a higher temperature during the distillation process, the error surface of the learned model can be smoothed. We denote the function learned by the neural network model as F. During the inference stage, the feature vector is input into the trained network and transformed into logit scores $z \in \mathbb{R}^{c \times 1}$. Then a softmax function is used to convert those scores into probabilities with respect to each class. Mathematically, the Jacobian's forward derivative of the output with respect to the input can be calculated as follows [100, 99]. For notational clarity, we denote the denominator of

the softmax function as $h(x) = \sum_{k=1}^{c}(exp(z_k)/T)$, where T is the temperature used during distillation. Thus, we have:

$$\begin{aligned}\frac{\partial F_i}{\partial x_j} &= \frac{\partial}{\partial x_j}(\frac{e^{z_i/T}}{h(x)}) \\ &= \frac{1}{h^2(x)}(\frac{\partial e^{z_i(x)/T}}{\partial x_j}h(x) - e^{z_i/T}\frac{\partial h(x)}{\partial x_j}) \\ &= \frac{1}{T}\frac{e^{z_i/T}}{h^2(x)}(\sum_{k=1}^{c}(\frac{\partial z_i}{\partial x_j} - \frac{\partial z_k}{\partial x_j})e^{z_k/T}). \end{aligned} \tag{4.14}$$

From (4.14), we see that as the derivative becomes smaller with higher temperature, the model is less sensitive to adversarial perturbations.

Ensemble Defense: The ensemble defense for extraction attacks and evasion attacks has been recently proposed by several authors [101, 102, 104] for tree ensemble classifiers. In this work, we study the ensemble defense with neural networks. The idea behind the ensemble defense is intuitive. It may be easy for an attacker to craft adversarial samples to compromise an individual detection model, but it is much more difficult for them to create samples which fool a set of models in an ensemble with different properties. We employ a "majority vote" ensemble defense in this work. We first train an ensemble with E classifiers where E is an odd number. During prediction, an unknown file is predicted to be malware if the majority (*i.e.*, $> E/2$) of the classifiers predict that the file is malicious.

Weight Decay Defense: The third defense we propose and study is the weight decay defense. Weight decay is typically used to prevent overfitting of machine learning models. The ℓ_2 norm of a weight matrix is defined as the square sum of all the elements. By adding an ℓ_2 penalty of the model weights in the objective function during optimization, the model is encouraged to prefer smaller magnitude weights since large values are penalized by the objective function.

With a smaller magnitude of weights, the function parameterized by the neural network is smoother, and therefore, changes in the input space lead to smaller changes in the output of a deep learning model. We conjecture that weight decay could help alleviate the vulnerability of a deep learning system against adversarial attacks.

4.4.6 Experimental Results

In this section, we evaluate the adversarial defenses against the different attack strategies described in the previous sections. We first describe some details related to data preparation and experimental setup. We then present the performance of the baseline classification system which does not employ any defenses. Finally, we evaluate the results for the distillation, weight decay, and the ensemble defenses.

Data Preparation and Setup: In some cases, multiple files can share the same input vector. Therefore, we only include the first instance of a unique input vector and discard any remaining duplicates. After de-duplication, we have input data and labels from 2,373,671 files. A file is assigned the label of 1 if it is malware and 0 if it is benign. We then randomly split the original dataset into a training set, validation set, and test set including 1,523,978, 268,937, and 580,756 files, respectively.

In our training, we implement all models using the Microsoft Cognitive Toolkit (CNTK) [121]. All models are derived from the baseline model described Section 4.4.3. We use the adam optimizer for training where the initial step size is set to 0.1. Training proceeds for each step size until no further improvement is observed in the validation error. At that point, CNTK halves the step size for subsequent epochs. We train for a maximum of 200 epochs, but CNTK implements early stopping when no additional improvement in the validation error is observed for a minimum step size of 1e-4.

Baseline Classifier: Before investigating the various defenses, we first analyze the performance of the baseline malware classifier useing the receiver operating characteristic (ROC) curves depicted in Figure 4.4 for a range of DNN hidden layers, H, varying from 1 to 4. Malware classifiers need to operate at very low false positive rates to avoid false positive detections which may result in the removal of critical operating system and legitimate application files. Thus, our desired operating point is a false positive rate (FPR) of 0.01%. While the DNNs with multiple hidden layers offer equivalent performance at higher false positive rates compared to a shallow neural network with one hidden layer, the figures indicates that the DNNs offer improved performance at very low false positive rates. In particular, the false positive rate of the shallow neural model immediately jumps to over 0.015% which is above our desired operating point.

For reference, we next analyze the test error rates of the baseline malware classification system in Table 4.3. As observed in [105] for a different dataset created for dynamic analysis malware classification, a shallow neural network with a single hidden layer provides the best overall accuracy. The test error rates in Table 4.3 are computed with the probability that the file is malicious $p_M \geq 0.5$. This threshold corresponds to operating points with higher false positive rates on the ROC curves for $H \in 1, 2, 3, 4$. These higher thresholds explain why the shallow network has a better test error, but the ROC curves indicate better performance for multiple hidden layers at the same FPR.

Distillation Defense: We next analyze the performance of the distillation defense system for all malware and benign files. The ROC curves of the DNN systems employing the distillation defense are presented in Figure 4.5a for temperature setting $T = 2$ and Figure 4.5b for $T = 10$. We make several observations from these figures. Both systems provide multiple operating points below FPR = 0.01% which allows

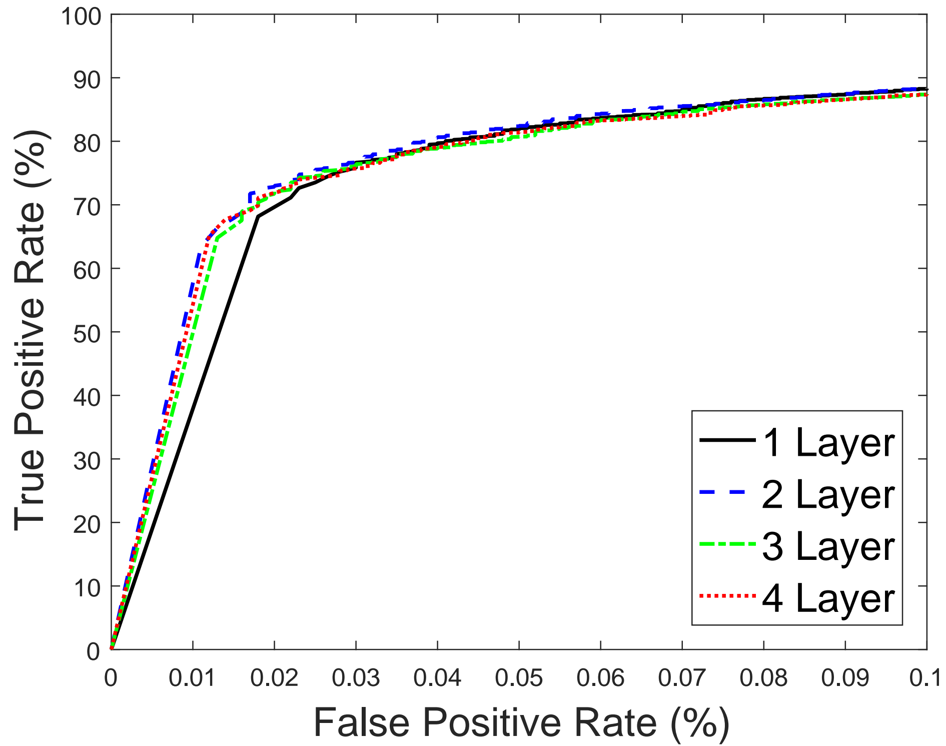
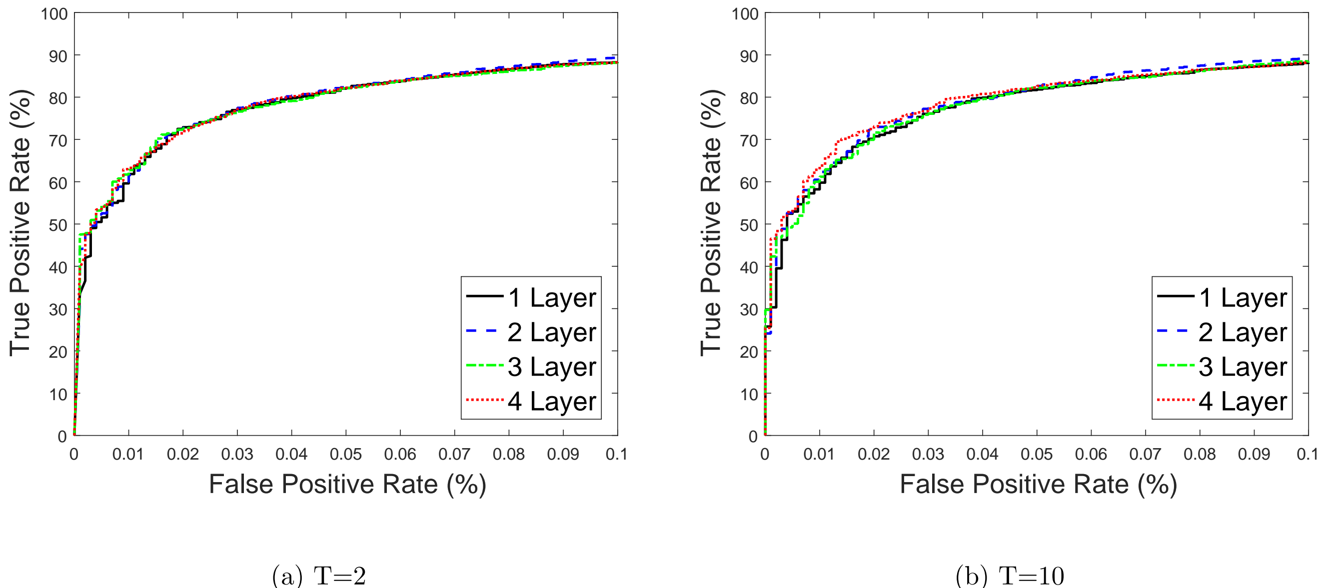


Figure 4.4: ROC curves of the baseline malware classifier for different numbers of hidden layers.

Layers	Test Error Rate (%)
1	1.1378272
2	1.2053255
3	1.1762255
4	1.1619338

Table 4.3: Test error rates of the baseline malware classifier for different numbers of hidden layers.



(a) T=2

(b) T=10

Figure 4.5: ROC curves of the malware classifiers with the distillation defense with different temperature.

better fine-tuning of the models. For the model with $T = 2$, we do not observe any benefit from adding multiple hidden layers. However, we do get a small lift in the performance for the DNN with 4 hidden layers for $T = 10$. Both systems offer similar performance above an FPR $= 0.02\%$ compared to the baseline classifiers in Figure 4.4.

In Figure 4.12, we next investigate the effectiveness of the six adversarial sample crafting strategies for the baseline classifiier and distillation defense, with temperatures $T =\in \{2, 10\}$, for model depths $H \in \{1, 2, 3, 4\}$. In each iteration, a single feature is modified, and the generated sample is evaluated by the trained model to test whether the sample is misclassified. From Figure 4.12, we make several observations. Generally, the distilled models follow a similar trend with regard to the six strategies for crafting adversarial samples, where dec_pos and dec_pos+inc_neg are the

two most effective strategies for the attacker. With a higher distillation temperature, it becomes much harder to craft adversarial samples for the distilled model. If the same number of features is perturbed, the success rate for crafting adversarial samples is reduced significantly for models distilled with a higher temperature. This result is because the error surface of the distilled model is smoothed for higher temperatures, such that the output is less sensitive with respect to the input.

We summarize the success of the different iterative strategies for crafting adversarial samples after iteration 20 in Figures 4.7a for the baseline classifier, Figure 4.7b for $T = 2$, and Figure 4.7c for $T = 10$. The figures indicate that shallow networks with $H = 1$ hidden layers are the most susceptible to successfully crafted adversarial samples. We see that using the Jacobian information can help to craft more adversarial samples with the same number of perturbed features than its randomized counterparts. From the attacker's perspective, the dec_pos strategy (switching off positive malware features) is the most effective approach for crafting adversarial samples for the full defense with $T = 10$. Likewise, dec_pos + inc_neg (alternatively switching off positive feature and switching on negative feature) is more effective than inc_neg (switching on negative features). This is fortunate from the defender's perspective because it requires the attacker to potentially spend more effort implementing alternative strategies for removing malicious features.

Weight Decay Defense: We next present an analysis of the proposed weight decay defense. We train the malware classification model using different strengths of weight decay regularization, $D \in \{0.0001, 0.0005, 0.001, 0.01\}$, and plot the ROC curves for these values of D in Figures 4.8a-4.8d, respectively. In general, the true positive rates drop with increasing values of D.

We analyze all combinations of weight decay strength and hidden layer depth in terms of defense to adversarial attacks. The best overall resilience of this model

defense to the six adversarial sample crafting strategies for iteration 20 also employs $D = 0.0001$ and is summarized in Figure 4.9a. For comparison, we also summarize the defensive capabilities for $D = 0.0005$ in Figure 4.9b. Figure 4.9a shows that the resilience to adversarial sample crafting strategies also increases as the hidden layer depth increases. The weight decay defense is not as effective as the distillation defense in Figure 4.7c or even the basline model in Figure 4.7a.

Ensemble Defense: Finally, we present the results for the ensemble defense on our dataset. In Figure 4.11, we present the ROC curves for an ensemble with $E = 5$ classifiers. Ensembles with other numbers of classifiers offer similar results.

The summary results after 20 iterations for $E = 3$ and $E = 5$ classifiers are shown in Figure 4.12a and Figure 4.12b, respectively. The figures indicate that increasing the number of classifiers in the ensemble make increases the difficulty of successfully crafting adversarial examples. Furthermore, the ensemble defense greatly reduces the percentage of successfully crafted samples compared to the results for the baseline classifier in Figure 4.7a and the distillation defense with $T = 10$ in Figure 4.7c.

4.5 Evaluation Attack: Images

In this section, we study the a type of attack during the evaluation stage of a trained model. Nowadays there are more and more systems employee biometrics as a proof-of-identity for accessing privileged information. Among them fingerprint, iris and face are the most widely used identity proof. The backend of those security systems/payment system is usually powered by machine learning models. Many of them are taken convolutional neural network based models due to the superior performance.

The underlying assumption for these security systems are that those identity proofs are unique in the world and hard to get by attackers. However, in this work we show that these systems are vulnerable to attacks by the computer generated images which exploit the deployed deep learning models. Attackers can generate fake identity proofs easily, without any physic/digital access to the concerned subject. This would be detrimental since the users has no fault at all on their end, while their confidential/privileged information is at stake. The situation might be more serious consider that biometrics are, unlike password based access control, hard to change.

4.5.1 Method: Deep Generator Attack

In this subsection, we introduce an attack during the evaluation stage. The attack can automatically generate images that would be very confidently (100%) recognized as being any desired target class that the attackers want it to be.

We exploit an architecture as illustrated in Figure 4.13. At the first glance, it looks similar to the Generative Adversarial Network (GAN) [122, 123], where images are sampled from a latent subspace, as indicated in Figure 4.13 at the bottom, represented by green and blue dots. The green dot represents a class indicator that can only accept integer value in the range from one to the number of classes that the discriminator outputs. The remaining blue dots represent the random noise. The generator map the input from the latent space to the image space, upsampled by layers of deconvolutional operators. Then the generated image is fed into a discriminator, which is a trained CNN that deployed for services.

Note that although the architecture looks similar to GAN, it does not have much thing to do with GAN beyond that. A GAN is a game play between the generator and discriminator, where the generator plays the thief trying to generate image as realistic as possible that can not be distinguished from real images, while

the discriminator plays the police tying to discern the fake generated images from the real ones. In our network, however the weights in discriminator are fixed. To prevent the confusion with GAN, we emphasize the differences as follows:

1) The discriminator is taken from a trained network, and the weights in the discriminator are fixed. This is because we try to mimic the situation where the trained network is deployed at service.

2) The output of the discriminator is the real classes that the discriminator has been trained to predict. In contrast, GAN's discriminator predicts whether the input is real image or generated fake image.

3) Our goal is not to generate realistic images, but to generate images that will break into the system's prediction system. During the training process, not a single real image is used.

The training procedure optimizes the weights in the generator network via stochastic gradient descent. The training examples are generated by sampling the latent space $z = z_0, z_1, ...z_n$, $n = 50$, z_0 is the class indicator element. Each pair (z, z_0) forms a training case, where z is the feature, and z_0 is the ground truth label, emphi.e. the generator will a generate an image recognized the trained model as being class z_0. In this way, after the generator is trained by many such samples, we can simply generate images that will be classified into the desired class by specifying the class indicator z_0 in the input space.

4.5.2 Experimental Results

We run the experiment on both MNIST, CIFAR10 and ImageNet dataset to show the effectiveness of the attack. For the MNIST experiment, the discriminator is a standard LeNet as described in section 4.3.2, which achieves classification accuracy of 99.23% on the held out testing set. The generator starts from the 50 dimensional

vector in the latent space, and go through two fully connected layers with 1024 and 7*7*128 neurons respectively. The hidden layer is reshaped into 128 7x7 feature maps and upsampled with two deconvolutional operators. In the output of the generator we get 28x28 images.

For the CIFAR10 experiment, we use off-the-shelf trained model with the same architecture as described in section 4.3.2 as the discriminator, which achieves 91.82% classification accuracy. The generator is similar to the one used in the MNIST dataset, except that the 1024 neurons in the first layer is mapped into 8*8*128 neurons, because the output image should be of the shape 32x32 for the CIFAR10 classifier.

For the ImageNet experiment, we use a trained ResNet-50 [53] as the discriminator, which achieves a reported 92.2% top5 classification accuracy on the ImageNet evaluation set . The generator is of the similar architecture of the one used in the MNIST experiment except that 3 more upsampling layers are used to map the input to the 224x224 image space. For all the experiments, we use the Adam operator with a learning rate of 10^-4 and train the generator 20000 iterations while keep the discriminator's weights fixed. During the training process, we observe that the training loss goes down quickly and the discriminator will soon recognize the generated images as the class specified by in z_0 in the latent space. This suggests that the attacker can exploit the deployed model to gain access to the system illegally pretty easily with this type of technique.

Figure 4.14 shows the generated images on each dataset. Although the most of the generated images are not discernible to human eyes, the neural network models (which all achieve more than 90% accuracy on the held out testing sets) recognize those images with 100% confidence of being a certain class. This suggests that although those neural networks can achieve incredible image recognition accuracy, what the neural networks have learned is fundamentally different from human eyes. Neural

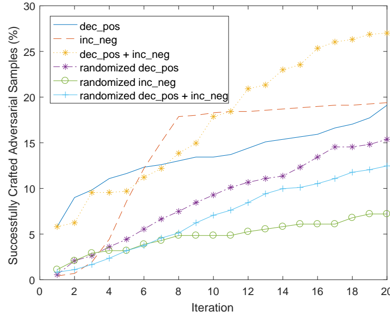
networks deployed might be vulnerable to access attacks, where the attackers can generate fake identity proof easily, without ever approaching the real identity holder. To massively adopt neural network methods in safety critical applications like access control, self-driving cars, and biometric payment services, we need to further investigate effective approaches to improve the robustness to such type of attacks.

It seems for smaller input space (*e.g.* 28x28 for the MNIST dataset), we might we able to reveal what the training data looks alike by adding some regularization. However, for the larger input space like CIFAR or ImageNet images, it is very hard to reveal how the training data looks alike even though we reach 100% confidence on the generated images. Due to the curse of dimensionality, the number of samples needed to properly train the model increases exponentially. Although trained on large amounts of training data, it is still hard cover the whole space of variation in the image. Modern neural network models which do not considering much geometric invariance beyond translation are still incapable of disentangle features that is essential for the classification, and overfit to the training data in a certain degree.

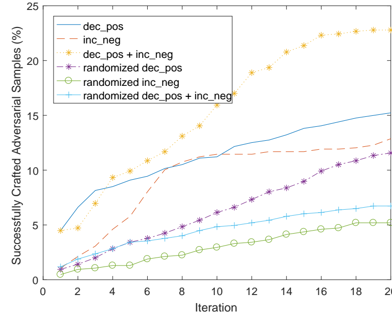
4.6 Conclusion

With the wide adoption of deep learning in each industry, we believe it is of paramount importance to investigate potential attacks of deep learning system on every stage: from data collection, model training, model testing/evaluation stage. This paper investigates two potential attacks, one in training stage via injecting noise in the training data, where we proposed a new loss function to alleviate the potential loss of model performance. The other attack is in the evaluation stage via generating pseudo-images which will be predicted to be any target class with 100% confidence by the deployed model. Those attacks are never meant to be exhaustive, attackers can come up with different attack techniques in any stage. Given the success of deep

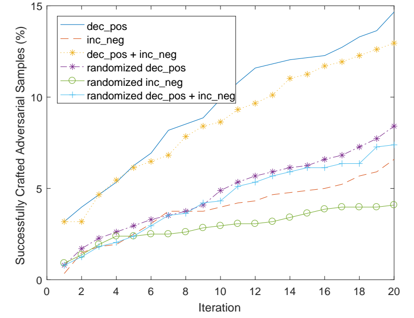
learning in industry applications. We believe the security of deep learning models need much more attention/vigilance from academic community and industry adopters.



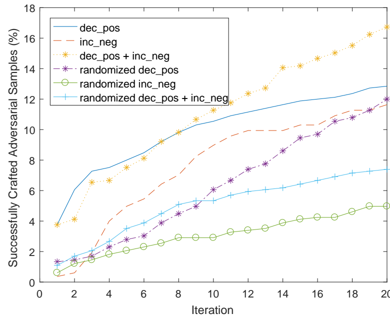
(a) $H = 1$, Baseline



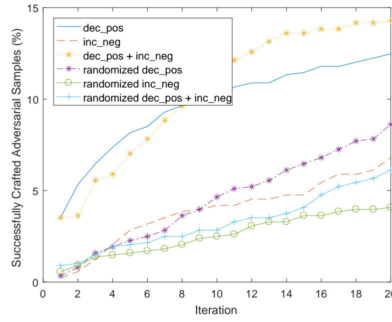
(b) $H = 1$, $T = 2$



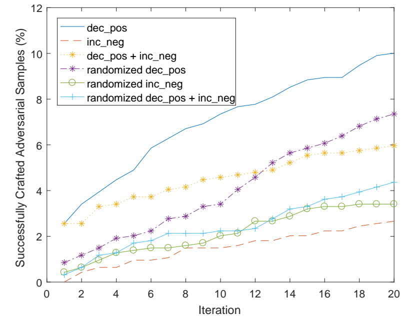
(c) $H = 1$, $T = 10$



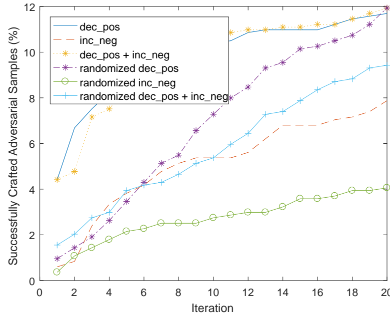
(d) $H = 2$, Baseline



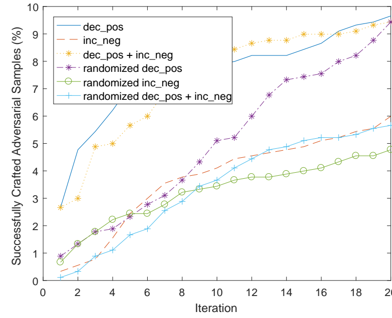
(e) $H = 2$, $T = 2$



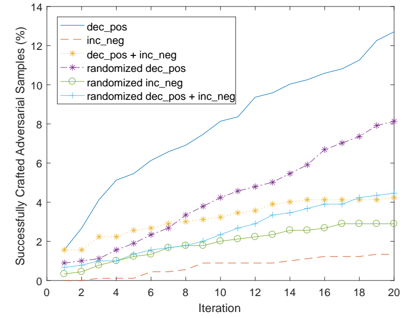
(f) $H = 2$, $T = 10$



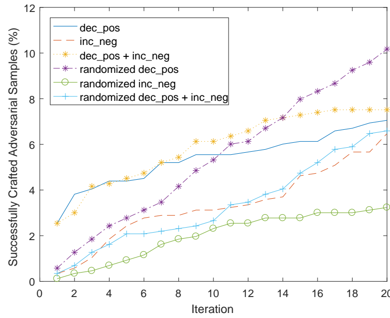
(g) $H = 3$, Baseline



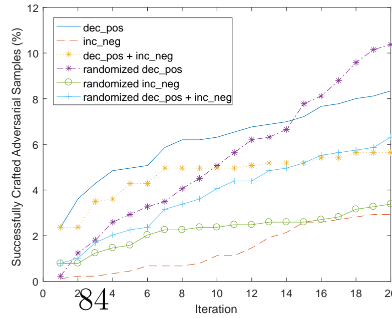
(h) $H = 3$, $T = 2$



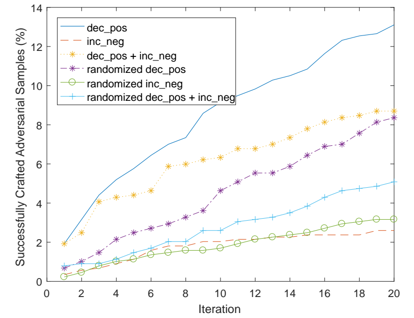
(i) $H = 3$, $T = 10$



(j) $H = 4$, Baseline

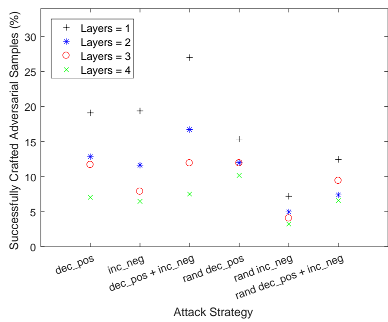


(k) $H = 4$, $T = 2$

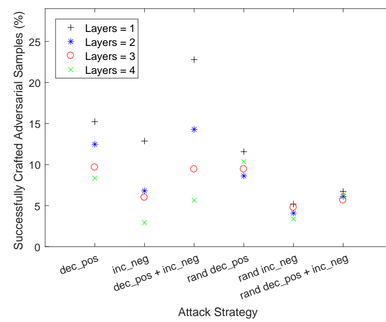


(l) $H = 4$, $T = 10$

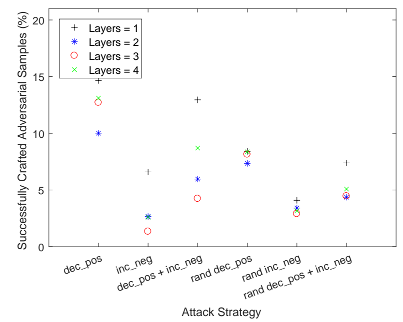
Figure 4.6: Success rates of adversarial samples against the baseline classifer and the



(a) baseline model, no defense

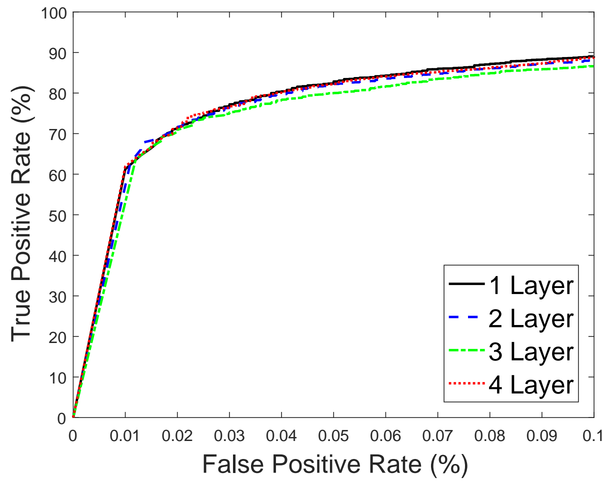


(b) T=2

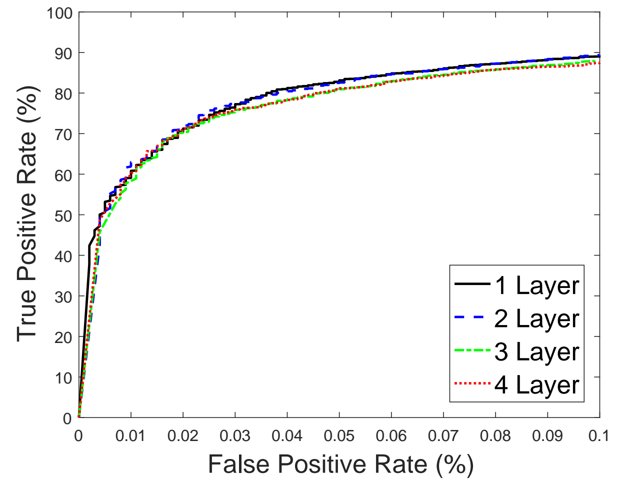


(c) T=10

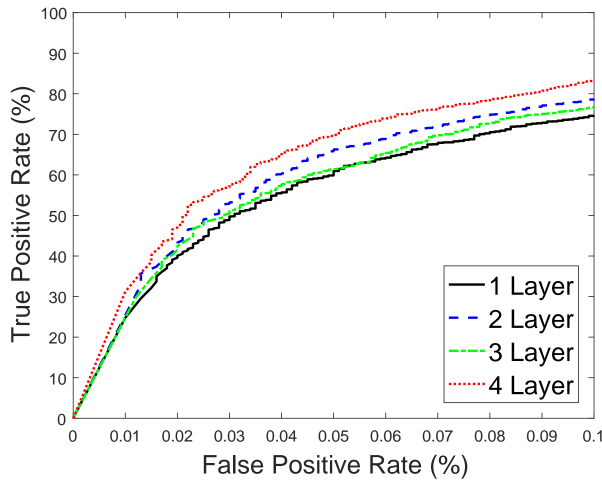
Figure 4.7: Percentage of successfully crafted adversarial samples for different sample crafting strategies with different temperature for distillation defense.



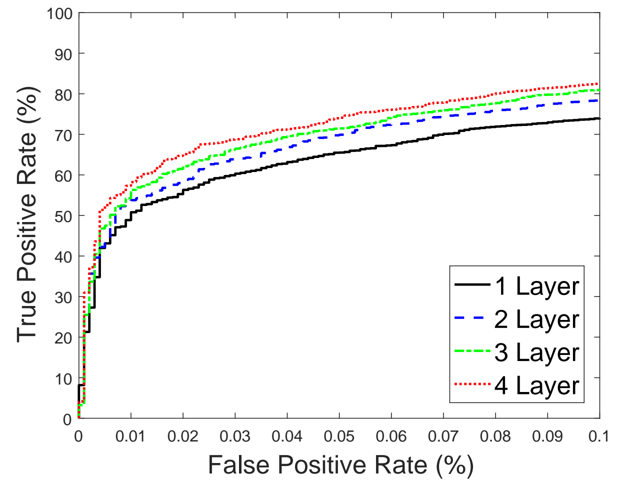
(a) D=0.0001



(b) D=0.0005

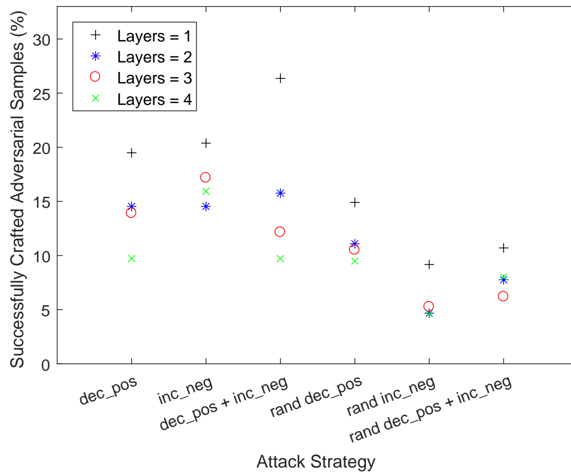


(c) D=0.001

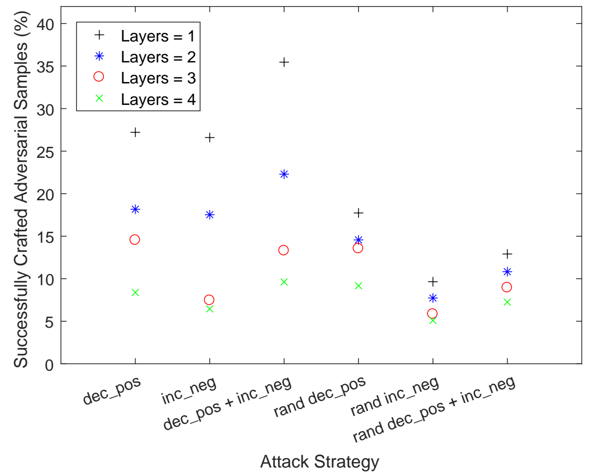


(d) D=0.01

Figure 4.8: ROC curves of the malware classifiers for different regularization strength with different numbers of hidden layers.



(a) D=0.0001



(b) D=0.0005

Figure 4.9: Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with different weight decay strength.

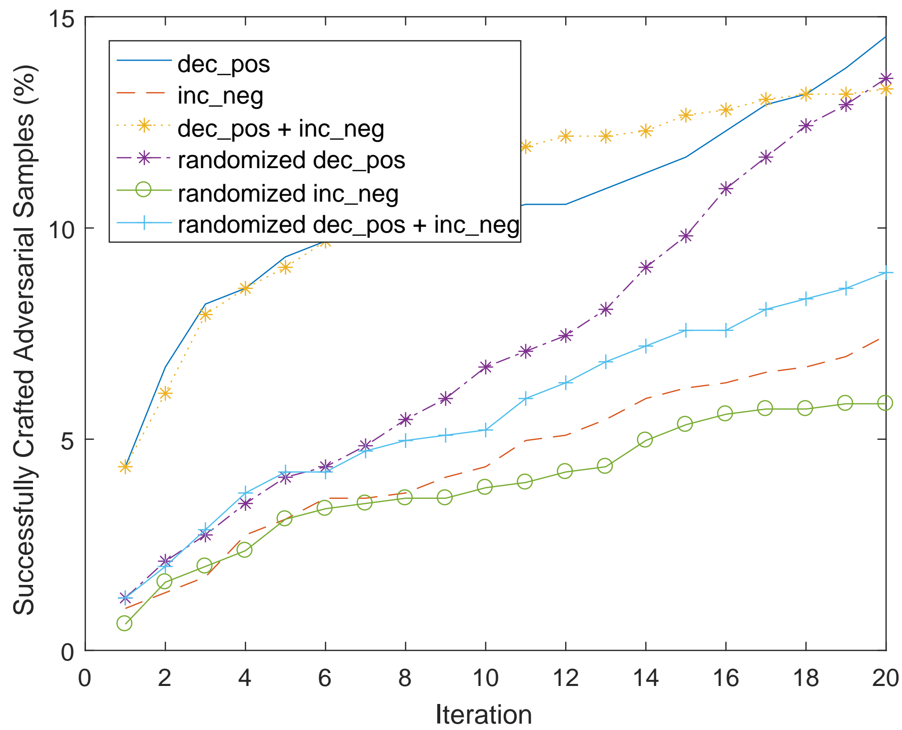


Figure 4.10: Percentage of successfully crafted adversarial samples for the first 20 iterations with different sample crafting strategies, $D = 0.0005$ weight decay and $L = 3$ hidden layers.

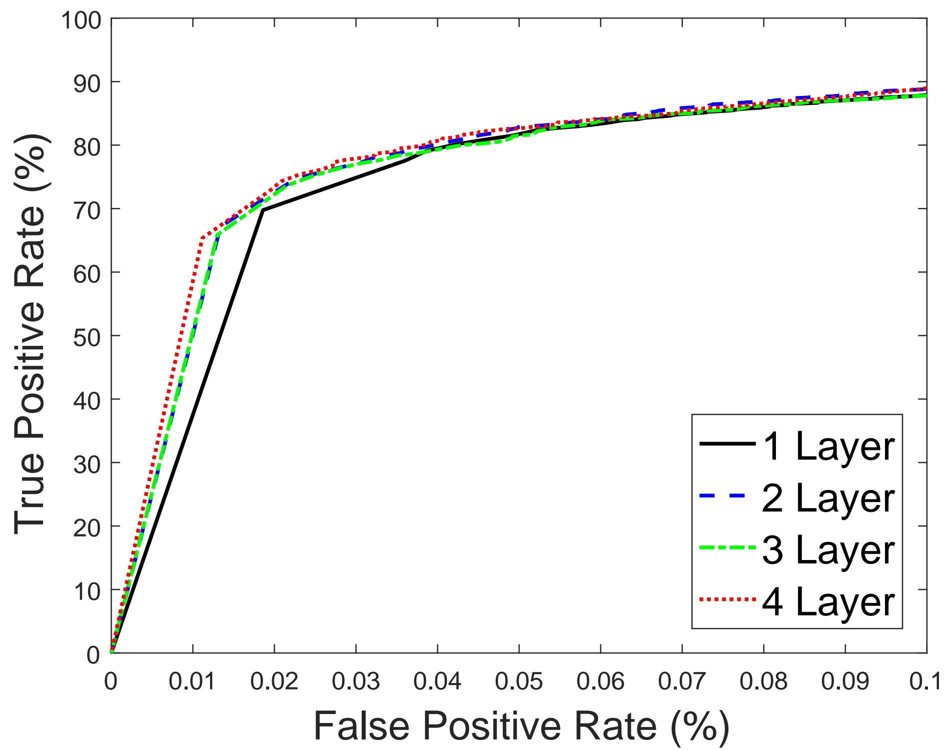
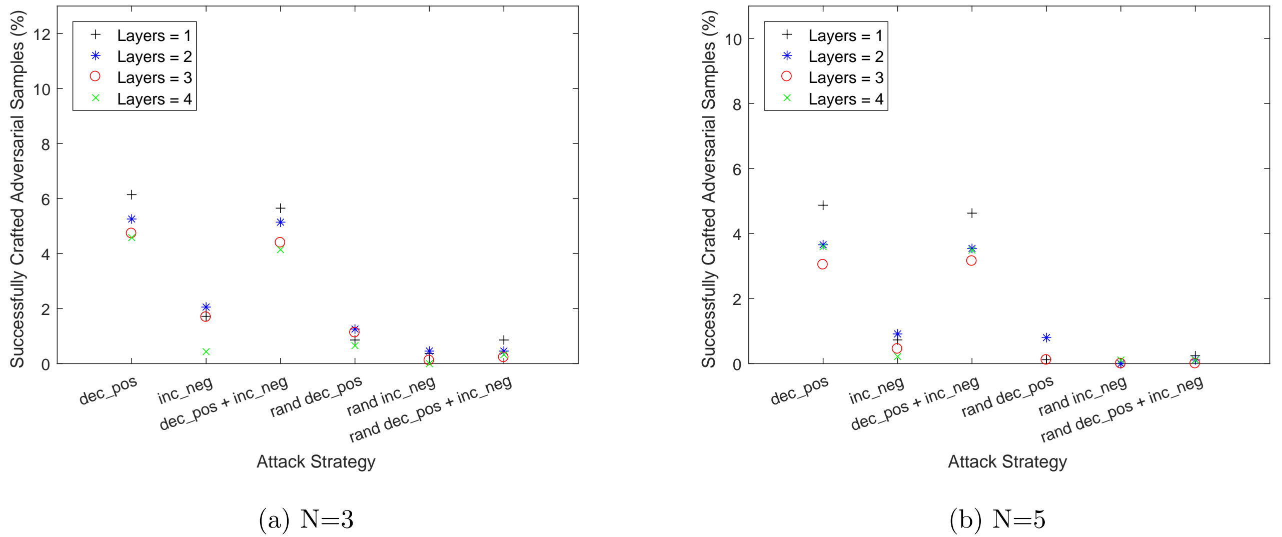


Figure 4.11: ROC curves of the ensemble malware classifier with $E = 5$ classifiers for different numbers of hidden layers.



(a) N=3 (b) N=5

Figure 4.12: Percentage of successfully crafted adversarial samples after iteration 20 for different sample crafting strategies with different number of models for the ensemble defense.

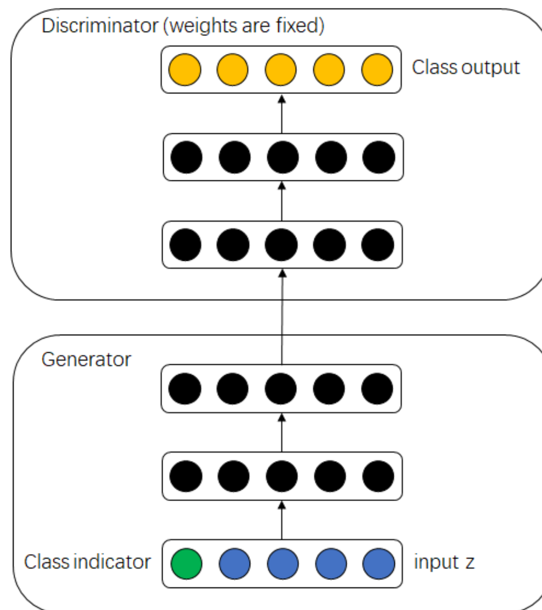
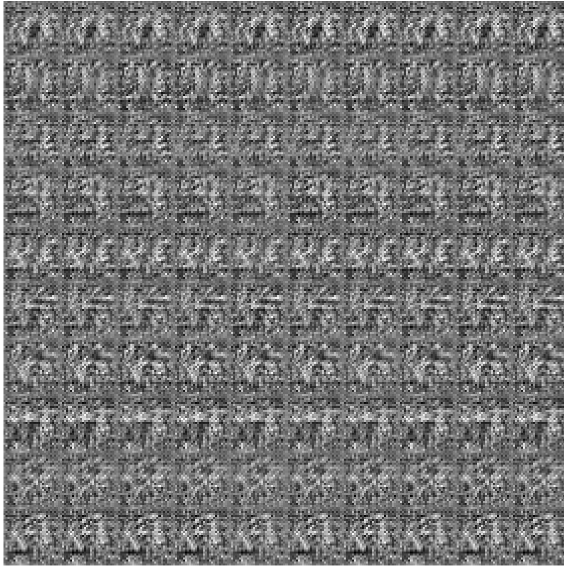
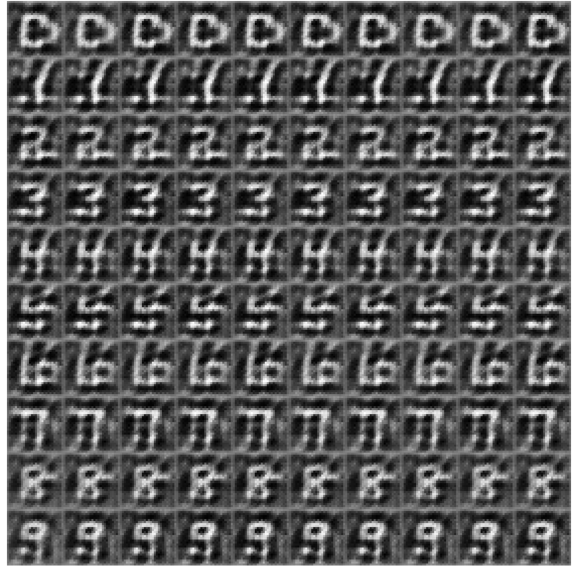


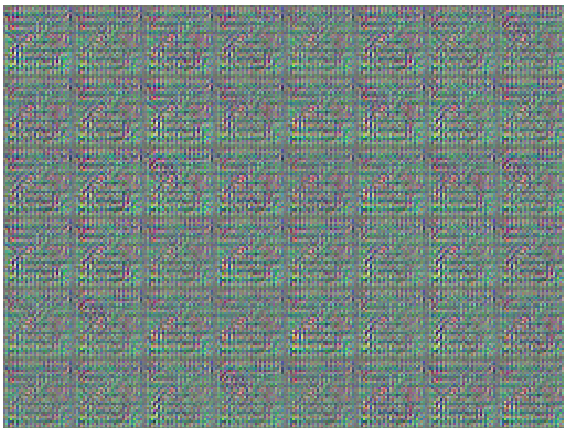
Figure 4.13: Illustration of the deep generator attack architecture.



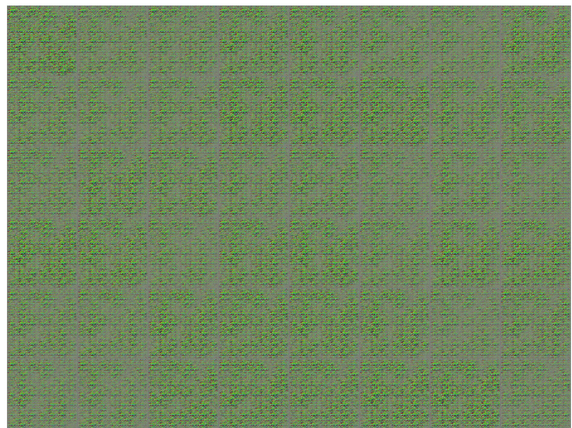
(a) MNIST without regularization



(b) MNIST with regularization (10^{-3})



(c) CIFAR10: 6x8 images randomly sampled



(d) ImageNet: 6x8 images randomly sampled

Figure 4.14: Generated images on different datasets that are recognized 100% confident as being the target class that attackers desired.

CHAPTER 5

Conclusion

As the industry is adopting deep learning models in more and more applications, such as self-driving cars, medical image diagnostics, video surveillance, etc., it is of paramount importance to develop efficient and accurate deep learning models such that the application can achieve satisfactory performance in limited computational resources within a certain latency budget.

Better designed architecture reduces the computational overhead and improves the perception accuracy. In this dissertation, we proposed a new architecture, which is termed stochastic columnar network. The network is composed of many columns where each column is a small computational graph. During the training stage, those columns will be randomly dropped with a certain probability. The DropColumn operation promotes the independence of the columns. We show that the network exhibits ensemble property, by removing many columns during the inference stage, the accuracy drops smoothly. Empirical results show that the proposed methods improve the performance compared with state-of-the-art neural network architectures.

In addition, efficient knowledge distillation methods can be applied afterwards to further reduce model memory footprint while preserving the model accuracy. We proposed a new knowledge distillation method. Instead of distilling knowledge from the probability score or logit layer (the penultimate layer of a neural network which is of the same dimension as the number of classes), we proposed to distill knowledge from the lower layer of the neural network. The intuition is that the lower layer of the teacher model contain more useful information that can be used to distill

knowledge from. After distill the useful information from the teacher model, the student model can learn a linear function on the distilled feature space to predict the final class of testing examples. Empirical results show that the proposed approach works better than standard practice for knowledge distillation like dark knowledge or logit regression.

On the other hand, the security of deep learning models should also be take into serious consideration. In this dissertation, we show that deep learning models are vulnerable to attacks at both training stage and evaluation stage. For the training attacks, we proposed to replace the standard loss function with a new loss function called correntropy loss, such that the model can be more robust to injected noise in training data. Empirical results show that the performance of the proposed loss function works better than standard loss function like cross entropy or hinge loss with the presence of injected noise. We also studied one type of evaluation attacks, and showed that deep learning models are subject to access attack where the attackers can exploit the deployed models and generate fake identity proof easily. Therefore, we should be wary about the security while deploying deep learning models.

REFERENCES

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

[2] 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.

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

[4] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, *et al.*, "Recent advances in deep learning for speech research at microsoft," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 8604–8608.

[5] 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," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[6] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 513–520.

[7] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 129–136.

[8] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[9] 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.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[13] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[14] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *http://deeplearning.net/software/theano/*, 2012.

[15] F. Chollet, "Keras," *KerasCN Documentation*, p. 55, 2017.

[16] S. Tokui, K. Oono, and S. Hido, "Chainer: a next-generation open source framework for deep learning," in *https://chainer.org/*, 2015.

[17] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[18] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[19] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[20] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[21] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: a simple and general method for semi-supervised learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 2010, pp. 384–394.

[22] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[23] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[24] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[26] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[27] Y. Lecun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The handbook of brain theory and neural networks*. MIT Press, 1995.

[28] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[31] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[32] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[33] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[35] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)14)*. USENIX Association, pp. 583–598.

[36] 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.

[37] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[38] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

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

[40] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153–160.

[41] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[42] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 2011, pp. 833–840.

[43] Y. He, K. Kavukcuoglu, Y. Wang, A. Szlam, and Y. Qi, "Unsupervised feature learning by deep sparse coding," in *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 2014, pp. 902–910.

[44] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area v2," in *Advances in neural information processing systems*, 2008, pp. 873–880.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feed-forward neural networks." in *Journal of Machine Learning Research*, 2010.

[46] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[47] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, *et al.*, "On rectified linear units for speech processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3517–3521.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[49] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[50] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[51] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[52] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[53] 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.

[54] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely connected convolutional networks,” *arXiv preprint arXiv:1608.06993*, 2016.

[55] 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.

[56] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[57] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[58] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[59] Jürgen Schmidhuber and Daan Wierstra and Faustino J. Gomez, “Evolino: Hybrid neuroevolution/optimal linear search for sequence learning,” in *IJCAI*, 2005.

[60] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, “Training recurrent networks by evolino,” *Neural computation*, vol. 19, no. 3, pp. 757–779, 2007.

[61] J. Schmidhuber, M. Gagliolo, D. Wierstra, and F. Gomez, “Evolino for recurrent support vector machines,” *arXiv preprint cs/0512062*, 2005.

[62] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

[63] J. Ba and R. Caruana, “Do deep nets really need to be deep?” in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[64] T. Chen, I. Goodfellow, and J. Shlens, “Net2net: Accelerating learning via knowledge transfer,” *arXiv preprint arXiv:1511.05641*, 2015.

[65] W. Chan, N. R. Ke, and I. Lane, “Transferring knowledge from a rnn to a dnn,” in *INTERSPEECH*, 2015.

[66] K. J. Geras, A.-r. Mohamed, R. Caruana, G. Urban, S. Wang, Ö. Aslan, M. Philipose, M. Richardson, and C. A. Sutton, “Compressing lstms into cnns,” *International Conference on Learning Representations workshop track*, 2016.

[67] S. Lin, R. Ji, X. Guo, X. Li, *et al.*, “Towards convolutional neural networks compression via global error reconstruction,” in *International Joint Conferences on Artificial Intelligence*, 2016.

[68] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” *arXiv preprint arXiv:1412.6115*, 2014.

[69] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” in *International Conference on Learning Representations (ICLR’16 best paper award)*, 2015.

[70] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[71] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, “Policy distillation,” *arXiv preprint arXiv:1511.06295*, 2015.

[72] Z. Li and D. Hoiem, “Learning without forgetting,” in *European Conference on Computer Vision*. Springer, 2016, pp. 614–629.

[73] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against deep learning systems using adversarial examples,” *arXiv preprint arXiv:1602.02697*, 2016.

[74] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and K. Talwar, “Semi-supervised knowledge transfer for deep learning from private training data,” *arXiv preprint arXiv:1610.05755*, 2016.

[75] C. Bucilu, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[76] G. Urban, K. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. rahman Mohamed, M. Philipose, and M. Richardson, “Do deep convolutional nets really need to be deep or even convolutional?” *CoRR*, vol. abs/1603.05691, 2016.

[77] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” in *Advances in neural information processing systems*, 2014, pp. 2933–2941.

[78] A. Torralba, R. Fergus, and W. T. Freeman, “80 million tiny images: A large data set for nonparametric object and scene recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[79] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.

[80] A. Krizhevsky, “Cifar10 and cifar100 dataset,” https://www.cs.toronto.edu/~kriz/cifar.html, 2010.

[81] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[82] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[83] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[84] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision.* Springer, 2016, pp. 646–661.

[85] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[86] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[87] g. larsson, m. maire, and g. shakhnarovich, "fractalnet: ultra-deep neural networks without residuals," *arxiv preprint arxiv:1605.07648*, 2016.

[88] s. targ, d. almeida, and k. lyman, "resnet in resnet: generalizing residual architectures," *arxiv preprint arxiv:1603.08029*, 2016.

[89] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *arXiv preprint arXiv:1611.05431*, 2016.

[90] y. chen, j. li, h. xiao, x. jin, s. yan, and j. feng, "dual path networks," in *advances in neural information processing systems*, 2017, pp. 4470–4478.

[91] b. zoph, v. vasudevan, j. shlens, and q. v. le, "learning transferable architectures for scalable image recognition," *arxiv preprint arxiv:1707.07012*, 2017.

[92] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Research Blog*, 2017.

[93] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure estimation and control for cyber-physical systems under adversarial attacks," *IEEE Transactions on Automatic Control*, vol. 59, no. 6, pp. 1454–1467, 2014.

[94] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Information Sciences*, vol. 239, pp. 201–225, 2013.

[95] 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.

[96] Y. Tang, "Deep learning using linear support vector machines," *arXiv preprint arXiv:1306.0239*, 2013.

[97] Y. Feng, J. Fan, and J. A. Suykens, "A statistical learning approach to modal regression," *arXiv preprint arXiv:1702.05960*, 2017.

[98] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *Proceedings of the International Conference on Learning Representations (ICML)*, 2015.

[99] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2015.

[100] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," *Proceedings of the 37th IEEE Symposium on Security and Privacy*, 2015.

[101] A. Kantchelian, J.D.Tygar, and A. D.Joseph, "Evasion and hardening of tree ensemble classifiers," *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[102] F. Tramer, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," *Proceedings of the USENIX Security Symposium*, 2016.

[103] J. Feng, T. Zahavy, B. Kang, H. Xu, and S. Mannor, "Ensemble robustness of deep learning algorithms," *arXiv preprint arXiv:1602.02389v3*, 2016.

[104] F. Tramer, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defense," *arXiv preprint arXiv:1705.07204v2*, 2017.

[105] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 3422–3426.

[106] J. Saxe and K. Berlin, "Deep neural network based malware detection using two-dimensional binary program features," *Malware Conference (MALCON)*, 2015.

[107] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2015, pp. 1916–1920.

[108] W. Huang and J. W. Stokes, "Mtnet: A multi-task neural network for dynamic malware classfication," in *Proceedings of Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2016, pp. 399–418.

[109] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 2482–2486.

[110] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence.* Springer International Publishing, 2016, pp. 137–149.

[111] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[112] C. Smutz and A. Stavrou, "Malicious pdf detection using metadata and structural features," *Technical report*, 2012.

[113] N. Srndic and P. Laskov, "Detection of malicious pdf files based on hierarchical document structure," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2013.

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

[115] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2017.

[116] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," in *arXiv preprint arXiv:1702.06280v2*, 2017.

[117] C. D. Manning, P. Raghavan, and H. Schutze, *An Introduction to Information Retrieval.* Cambridge University Press, 2009.

[118] J. Crandall, Z. Su, F. Chong, and S. Wu, "On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS'05)*, 2005, pp. 235–248.

[119] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ICDM)*, 2006, pp. 287–296.

[120] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313

[121] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 2135–2135.

[122] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[123] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.