FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

A STUDY OF FEATURE INTERACTIONS AND PRUNING ON NEURAL NETWORKS

By

YANGZI GUO

A Dissertation submitted to the
Department of Mathematics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2020

Yangzi Guo defended this dissertation on July 15, 2020.
The members of the supervisory committee were:

Adrian Barbu

Professor Directing Dissertation

Yiyuan She

University Representative

Kyle Gallivan

Committee Co-Chair

Lingjiong Zhu

Committee Member

Monica Hurdal

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

Dedicated to every person gives me help in my life

ACKNOWLEDGMENTS

I would like to express my sincere thanks to Dr. Adrian Barbu. Without his endless encouragement and help, I will never imagine I can reach this step. He is the kindest professor I have ever seen in my life. I am really lucky to be a student under his supervision.

I also appreciate Dr. Kyle Gallivan, and all the committee members, for their gracious support in any aspect.

CONTENTS

List of Tables vii

List of Figures viii

List of Abbreviations x

Abstract xi

1 Introduction **1**
- 1.1 Brief History of Machine Learning 1
- 1.2 Categories of Machine Learning Tasks 1
- 1.3 Overview of Neural Networks 3
- 1.4 Topic Concern and Thesis Structure 4

2 A Study of the Feature Interactions on Neural Networks **6**
- 2.1 Introduction 6
- 2.2 Related Work 7
- 2.3 Problem Statement 8
- 2.4 A Mathematical Formulation of Feed-Forward Neural Networks 8
- 2.5 The Loss Landscape of the NNs on the XOR Data 10
- 2.6 The Trainability of Neural Networks 12
- 2.7 Existence of Junk Nodes 14
- 2.8 Node Selection 18
  - 2.8.1 Overview of the FSA algorithm 18
  - 2.8.2 Node Selection with Annealing Schedule 20
  - 2.8.3 Neural Network Generalization Consistency with NSA 23
  - 2.8.4 Neural Node Normalization 25
- 2.9 Idea on Selecting Features 31
- 2.10 Experiments 32
  - 2.10.1 XOR Simulated Datasets 32
  - 2.10.2 Parity Data 34
  - 2.10.3 Madelon Data 36
  - 2.10.4 Real Datasets 39
- 2.11 Conclusion 41

3 A Network Pruning Framework for Deep Convolutional Neural Networks via Annealing and Direct Sparsity Control **45**
- 3.1 Introduction 45
- 3.2 Related Work 47
- 3.3 Network Pruning via Annealing and Direct Sparsity Control 47
  - 3.3.1 Basic Algorithm Description 48
  - 3.3.2 Implementation Details 50
- 3.4 Experiments 52

3.4.1 Non-structured Pruning on MNIST 52
3.4.2 Structured Channel Pruning on the CIFAR and SVHN Datasets 55
3.5 Conclusions 61

4 Conclusion and Future Work **63**
4.1 Conclusion 63
4.2 Future Work 64

Bibliography 66

Biographical Sketch 71

LIST OF TABLES

2.1 Comparison between the standard NN, pruned sub-networks on the hidden layer and all layers, and retrained pruned networks with random weights and the initial weights from the beginning. 33

2.2 Comparison of the test error of Xgboost, NN, RF, NRE and FSBT+FSA+NSNA on the Madelon dataset. 39

2.3 Datasets used for evaluating the performance of fully connected NN and sparse NN with FSA+NSNA. 40

2.4 Performance results of NN(best), NN(equivalent) and NN+FSA+NSNA for each dataset. 41

3.1 Non-structured pruning comparison on LeNet-300-100 and LeNet-5. 53

3.2 Layer by layer compression comparisons on LeNet-300-100 and LeNet-5. 54

3.3 Channel pruning performance results comparison on CIFAR-10. 59

3.4 Channel pruning performance results comparison on CIFAR-100. 59

3.5 Channel pruning performance results comparison on SVHN. 59

LIST OF FIGURES

1.1 Causal structure representation of supervised and unsupervised learning 2

2.1 Visualization of 2D and 3D XOR data without noisy features. 9

2.2 Test AUC for non-noisy XOR data. n is the size of training and testing datasets. h is the number of hidden neurons. 9

2.3 Example of a neural network for binary classification with one hidden layer and one output node. 11

2.4 Loss landscape for 4D XOR ($k = 4$). 12

2.5 Values of sorted local minima (top) and train and test AUC (bottom) for 3D XOR. . 13

2.6 Values of sorted local minima (top) and train and test AUC (bottom) for 4D XOR. . 14

2.7 Values of sorted local minima (top) and train and test AUC (bottom) for 5D XOR. . 15

2.8 Test AUC of best energy minimum out of 10 random initializations vs. data dimension p for a NN with 512 hidden nodes. 15

2.9 Hit time vs dimension p for different XOR problems with $n = 3000$ observations. . . . 16

2.10 Training and testing AUC with (top) and without (bottom) Dropout vs. number of hidden nodes. 17

2.11 Decision tree generated by the label formulation of 2D XOR data. 18

2.12 The number of kept features M_e vs iteration e for different schedules with $p = 1000$, $k = 10$, $N^{epoch} = 500$. Figure source comes from Barbu et al. [2017] 19

2.13 Average test AUC vs number of hidden nodes for NN+NSA and NN+Dropout. 22

2.14 Test AUC of 100-run averaged and average of 10 best trials vs number of hidden nodes for NN+NSA. 23

2.15 100-run averaged test AUC vs number of hidden nodes for NN+NSA. The solid lines are the means, dashed lines are mean ± std. 24

2.16 Average test AUC vs number of hidden nodes for NNs with NSNA or NSA. 29

2.17 Average test AUC vs number of hidden nodes for NNs with NSNA or Dropout. 29

2.18 Test AUC of 100-run averaged and average of 10 best trials vs number of hidden nodes for NN+NSNA. 30

2.19 100-run averaged test AUC vs number of hidden nodes for NN+NSNA. The solid lines are the means, dashed lines are mean ± std. 30

2.20 Feature weights r_i from (2.11) for the $p = 20$ features for 10 random 4D XOR datasets of size $n = 3000$. 31

2.21 Testing AUC vs number of trials. 34

2.22 Loss and AUC evolution for training a pruned sub-network using NN+FSA+NSNA for $k = 4, p = 40$ XOR data. 35

2.23 Test error vs number of hidden nodes. Comparison between single hidden layer neural networks trained by NN + Adam + NSNA starting with 256 hidden nodes, NN + Adam, NN + SGD and BoostNet. 36

2.24 Train and test AUC of boosted trees with 1-100 boosted iterations, sorted by decreasing train AUC. 37

2.25 Variable detection rate. Percent of the runs all k features were found together in one of the 3000 subsets generated by FSBT with 30 boosting iterations. 38

2.26 Sorted loss values for 200 initializations obtained on the five real datasets. 43

2.27 Sorted test accuracy values for 200 initializations obtained on the five real datasets. . 44

3.1 Annealing schedule with $N_1 = 10, N^{iter} = 20, N^c = 1$, $p_0 = 0.8, p = 0.9, \nu = 0.02$. . . . 51

3.2 Examples of handwritten digits in the MNIST dataset. 53

3.3 Example images in the CIFAR10 dataset [Krizhevsky et al., 2009b]. 56

3.4 Example images in the CIFAR100 dataset [Krizhevsky et al., 2009b]. 57

3.5 Example images in the SVHN dataset [Netzer et al., 2011]. 58

3.6 FLOPs ratio between the original DCNNs and pruned sub-network for VGG-16 and DenseNet-40 on CIFAR and SVHN dataset. 60

3.7 The remaining channel distribution for each CONV layer for pruned networks on CIFAR 10. The first CONV layer is not displayed as our channel pruning algorithm DSC-2 will not act on this layer, thus contain the full percentage of channels. 61

3.8 The test error for various channel pruned percentage using one single global pruning rate for CIFAR-10. 62

LIST OF ABBREVIATIONS

Following is a short list of abbreviations used throughout this document.

- **NN**: Neural Network
- **ANN**: Artificial Neural Network
- **CNN**: Convolutional Neural Network
- **DCNN**: Deep Convolutional Neural Network
- **SVM**: Support Vector Machine
- **GB**: Gradient Boosted Tree
- **XGB**: Extreme Gradient Boosting
- **FSA**: Feature Selection with Annealing
- **XOR**: Exclusive Or
- **XOR**: Area Under Curve
- **NSA**: Node Selection with Annealing
- **NSNA**: Node Selection with Normalization and Annealing
- **FSBT**: Feature Selection using Boosted Trees
- **AUC**: Aras Under Curve
- **ML**: Machine Learning
- **DSC**: Direct Sparsity Control
- **SGD**: Stochastic Gradient Descent
- **BN**: Batch Normalization
- **CONV**: Convolutional
- **ACC**: Accuracy
- **FLOP**: Floating Point Operation

ABSTRACT

Artificial neural networks (ANNs) are very popular nowadays and offer reliable solutions to many classification problems. Recent research indicates that these neural networks might be overparameterized and different solutions have been proposed to reduce the complexity both in the number of parameters and in the training time of the neural networks. Furthermore, some researchers argue that after reducing the neural network complexity via weight selection or pruning, the remaining weights are irrelevant and retraining the sub-network would obtain a comparable accuracy with the original one. This may hold true in vision problems where we always enjoy a large number of training samples and research indicates that most local optima of the convolutional neural networks may be equivalent. However, in non-vision sparse datasets, especially with many irrelevant features where a standard neural network would overfit, this might not be the case and there might be many non-equivalent local optima. In this work, we present empirical evidence for these statements and an empirical study of the learnability of neural networks (NNs) on some challenging non-linear real and simulated data with irrelevant variables. Our simulation experiments indicate that the cross-entropy loss function on XOR-like data has many local optima, and the number of local optima grows exponentially with the number of irrelevant variables. We also introduce a novel efficient weight or neuron node selection method to improve the capability of NNs to find a deep local minimum even when there are irrelevant variables. Furthermore, we extend our approach to a network pruning framework that is scalable in dealing with various kinds of deep convolutional neural networks (DCNNs), both on structured and non-structured pruning without sacrificing inference accuracy.

CHAPTER 1

INTRODUCTION

1.1 Brief History of Machine Learning

Machine learning is a research field which aims at imitating using machines based and training data the intelligent ability of humans to accomplish different tasks. The research topics of machine learning span from the study of pattern recognition to computational learning theory. Machine learning arises from the long-time study of artificial intelligence. In the early days when the study of artificial intelligence began, some researchers were interested in utilizing machines to learn from data. They tried to invent several particular machines that have the ability to learn from data without the human's direct help. That might be the earliest study of machine learning [Langley, 2011]. In 1959, when machine learning was still considered a branch of artificial intelligence, Arthul Samuel presented a classical informal definition of it as "a field of study that gives computers the ability to learn without being explicitly programmed" [McCarthy and Feigenbaum, 1990]. This means that the main goal of machine learning was perceived to be to devise algorithms for machines or computers that understand data, learn from data or predict with data automatically, without human intervention and assistance. After great development over decades, machine learning was reorganized as a separate research field from artificial intelligence, and started to flourish in 1990s. Although machine learning is still the core part of the study of artificial intelligence, its focus has been towards methods and models derived from mathematical optimization and probability theory. Today, machine learning is a highly interdisciplinary research area, and intersects broadly with other fields such as Mathematics, Statistics, Computer Science, Physics and more.

1.2 Categories of Machine Learning Tasks

Based on the casual structure of machine learning algorithms, we can typically divide the learning tasks into two categories [Langley, 2011]. One is supervised learning, which involves learning a statistical model that predicts an output based on input examples and desired outputs. On the other side is unsupervised learning, in which there are inputs but no desired outputs.

However, even so, one can still study structures or relationships in the data. That is, all the inputs are assumed to be caused by latent variables of a casual chain. Figure.1.1 gives an illustration about the difference between supervised and unsupervised learning.

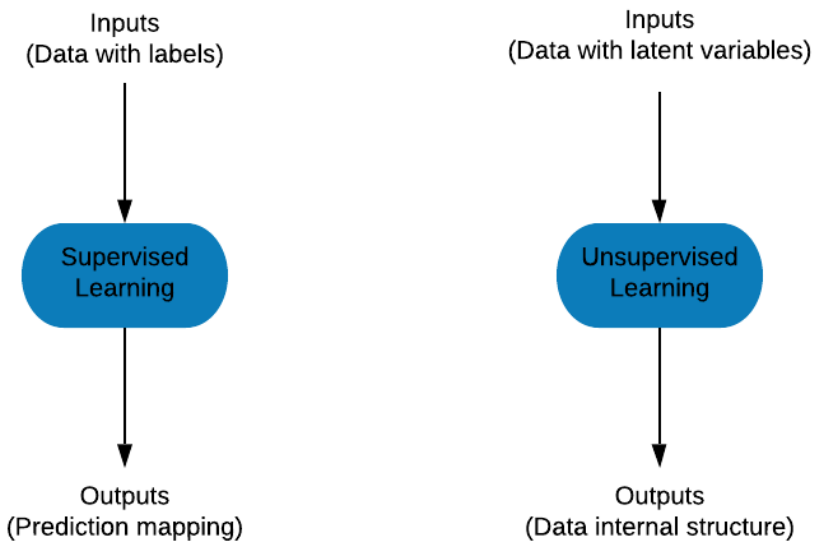


Figure 1.1: Causal structure representation of supervised and unsupervised learning .

When considering the desired output of a machine learning system, one can categorize the machine learning tasks [Bishop, 2006] as follows: 1) Classification, which it is the problem that assigns unseen new data to a specified class, on the basis of inputs treated as a training set of data points whose class memberships are known before hand; 2) Regression, which is similar to classification but the output consists of continuous variables rather than discrete class labels; 3) Clustering, which is the task that divides the unknown inputs into several groups, and with inputs in the same group being more similar to each other than to those in other groups; 4) Density estimation, which is to construct an estimate of the unknown distribution modes given the inputs; 5) Feature analysis, which aims at study the feature importance, interior structure of the data feature, where some sub-domains of this area include feature selection, feature extraction and dimensionality reduction.

1.3 Overview of Neural Networks

In this section, we give an overview of neural networks, especially the Feedforward Neural Networks we are going to study in our work.

Artificial neural networks are a very popular technique in the machine learning field this decade. However, in history, it is not a young research sub-area of artificial intelligence. One of the earliest works of artificial neural networks could trace back to 1943 when Warren McCulloch and Walter Pitts developed a computational model called threshold logic [McCulloch and Pitts, 1943]. This classical paper showed two ways to study neural networks. One lies in the biology part which studies the processes in the brain. The other is focused on applying the neural networks in artificial intelligence. During a similar time period, an algorithm for pattern recognition called the perceptron has been created by Rosenblatt [1958]. In 1965, Ivakhnenko and Lapa [1965] proposed the world's first neural network with many layers. These early papers form the initial basis of the research in artificial neural networks.

One of the most outstanding early works of learning neural networks occurred in 1975 when Werbos [1974] published a paper that introduces the backpropagation algorithm, which significantly improves the efficiency of training multi-layer neural networks. This learning technique stimulated the research interest of neural networks for many years and is still the main technique for training deep neural networks today.

The improvement progress of neural networks stagnated in the 1990s due to the gradient vanishing problem in backpropagation [Bengio et al., 1994], which impedes training deep neural networks with more than 4-5 layers. At the same time, other much easier to train machine learning models such as support vector machines (SVM) [Cortes and Vapnik, 1995] took over the neural network's place in machine learning popularity. In 1998, LeCun et al. [1998] proposed a pioneering 7-level neural network with several convolutional layers named as LeNet-5.

The true prosperity of neural networks came in the 2000s with the advance of computing hardware. In 2012, AlexNet [Krizhevsky et al., 2012], a similar CNN architecture compared to LeNet-5 but with deeper layers and new activation function, offered far better results than Support Vector Machines (SVMs) in the ImageNet [Deng et al., 2009] competition. This is the starting point of the advancements of deep convolutional neural networks. After that, the GoogLeNet [Szegedy et al., 2015] is proposed, to be the winner of the ILSVRC2014 [Russakovsky et al., 2015] competition.

The performance achieved by GoogLeNet was very close to the human-level performance. In the same year, another deep learning model named VGGNet [Simonyan and Zisserman, 2014] was developed, consisting of sixteen convolutional layers with a uniform architecture, and becoming one of the most preferred choices for extracting features from images. In the same year, Goodfellow et al. [2014] proposed a zero-sum game framework named generative adversarial network (GAN) in which two neural networks train against each other, which can generate very realistic synthetic images that are hard to tell by human eyes. In 2015, another breakthrough neural network architecture named ResNet [He et al., 2016] leads the convolutional neural network models going deeper, with 152 layers but lower complexity than VGGNet, and achieving a classification accuracy that even outperforms human beings for vision datasets. One year later, DenseNet[Huang et al., 2017] was proposed based on the inspiration of ResNet and quickly became one of the most frequently used deep convolutional neural networks at that time. Nowadays, many various kinds of deep convolution neural networks are proposed.

The research in the application of neural networks has been widely studied these days. In the meantime, the theoretical analysis of neural networks and even deep convolutional networks also attracted more and more researchers. Livni et al. [2014] gave a theoretical study of the computational complexity of training neural networks from a modern perspective. Janzamin et al. [2015] proposed a novel algorithm to train a two-layer neural network based on tensor decomposition, in which generalization bounds are proved to be guaranteed. Zhang et al. [2017] showed the L_1-regularized neural networks are properly learnable in polynomial time.

1.4 Topic Concern and Thesis Structure

Among many attractive research topics in the machine learning area, what we are particularly interested in is the supervised learning field — especially the study of the feature interactions and pruning technique on artificial neural networks (ANNs). More particularly we are interested in how to improve the neural network generalization power via neuron node selection and network pruning methods on either highly nonlinear noisy sparse datasets or rich vision datasets. The dissertation is organized as follows.

In chapter 2, we will study an extremely non-linear XOR-based data with irrelevant variables. Through numerical experiments on the XOR-based classification problems, we will show that the

logistic energy landscape has many shallow local minima as well as some deep minima that are very hard to find. We will also observe that the number of shallow local minima grows prohibitively large as the number of irrelevant variables increase, and the chances of finding a deep minimum decrease quickly with the number of irrelevant variables. This decrease is much faster for harder problems (e.g. 5D XOR) than for easier problems (3D XOR). Based on the insight obtained from our experiments, we propose a node and feature selection way to improve the capability of the ANNs to find a good deep local minimum when ANNs almost lost its generalization capability. We also discover the usefulness of boosted trees to propose multiple reduced feature sets to reduce the number of irrelevant features to a certain extent that NNs can handle when there are hundreds of irrelevant variables.

In chapter 3, we will extend our feature and neuron node selection approach derived in Chapter 2 to a network pruning framework that is scalable in dealing with various kinds of deep convolutional neural networks (DCNNs) both on structured and non-structured pruning without sacrificing prediction accuracy. We combine regularization techniques with sequential algorithm design and direct sparsity level control to bring forward a novel network pruning scheme that could be suitable for either non-structured pruning or structured pruning (particular for filter channel-wise pruning of DCNNs). We investigate an estimation optimization problem with a L_0-norm constraint in the parameter space, together with the use of annealing to lessen the greediness of the pruning process and a general metric to rank the importance of the weights or filter channels. Experiments on extensive real vision datasets including the MNIST, CIFAR and SVHN provide empirical evidence that the proposed network pruning framework can be comparable to or better than other state of art pruning methods.

In Chapter 4, we will draw the final conclusions about what we have achieved so far for the dissertation and explore the future works we can research.

CHAPTER 2

A STUDY OF THE FEATURE INTERACTIONS ON NEURAL NETWORKS

In many fields such as bioinformatics, high energy physics, power distribution, etc., it is desirable to learn non-linear models where a small number of variables are selected and the interaction between them is explicitly modeled to predict the response. In principle, artificial neural networks (ANNs) could accomplish this task since they can model non-linear feature interactions very well. However, ANNs require large amounts of training data to have a good generalization. In this section we study the data-starved regime where an ANN is trained on a relatively small amount of training data. For that purpose we study neuron node and feature selection for ANNs, which is known to improve generalization for linear models. As an extreme case of data with feature interactions we study the XOR-like data with irrelevant variables. We experimentally observed that the logistic loss function on XOR-like data has many non-equivalent local optima, and the number of local optima grows exponentially with the number of irrelevant variables. To deal with the local minima and for feature selection we propose a novel node and feature selection approach that improves the capability of ANNs to find better local minima even when there are irrelevant variables. Finally, we show that the performance of an ANN on real datasets can be improved using our method, obtaining compact networks on a small number of neurons and features, with good prediction and interpretability.

2.1 Introduction

Many fields of science such as bioinformatics, high energy physics, power distribution, etc., deal with tabular data with the rows representing the observations and the columns representing the features (measurements) for each observation. In some cases, we are interested in predictive models to best predict another variable of interest (e.g. catastrophic power failures of the energy grid). In other cases, we are interested in finding what features are involved in predicting the response (e.g. what genes are relevant in predicting a certain type of cancer) and the predictive power is

secondary to the simplicity of explanation. Furthermore, in most of these cases, a linear model is not sufficient since the variables have high degrees of interaction in obtaining the response.

Neural networks (NN) have been used in most of these cases because they can model complex interactions between variables, however they require large amounts of training data. We are interested in cases when the available data is limited and the NNs are prone to overfitting.

To get insight on how to train NNs to deal with such data, we will study the XOR data, which has feature interactions and irrelevant variables. The feature interactions are hard to detect in this data because they are not visible in any marginal statistics.

We will see that the loss function has many local minima that are not equivalent and that irrelevant features make the optimization harder when data is limited. To address these issues we propose a node and feature selection algorithm that can obtain a compact NN on a small number of features, thus helping deal with the case of limited data and irrelevant features.

2.2 Related Work

Recent studies [Draxler et al., 2018, Garipov et al., 2018] have shown that the local minima of some convolutional neural networks are equivalent in the sense that they have the same loss (energy) value and a path can be found between the local minima along which the energy stays the same. For this reason, we will focus our attention to fully connected neural networks and find examples where the local minima have different loss values. Moreover, Soudry and Carmon [2016] proves that all differentiable local minima are global minima for the one hidden layer ANNs with piecewise linear activation and square loss. However, nothing is proved for nondifferentiable local minima.

There has been quite a lot of work recently studying of the intrinsic sub-network about neural networks. Han et al. [2015a] proposed the "Deep Compression", a three stage technique, which significantly reduces the storage requirement for training deep neural networks without affecting their accuracy. Liu et al. [2018] showed that for structured pruning methods, directly training the small target sub-network or pruned model with random initialization can achieve a comparable or even better performance than retraining using the remaining parameters after pruning. They also obtained similar results towards to a unstructured pruning method [Han et al., 2015b] after fine-tuning the pruned sub-network on small-scale datasets. Frankle and Carbin [2019] introduced the

Lottery Tickets Hypothesis which claims that a random initialized dense neural network contains a sub-network that can be trained in isolation with the corresponding original initialized parameters to obtain the same test accuracy of the original network after training for the same number of iterations.

2.3 Problem Statement

To study the feature interactions on neural networks, we will look at an extreme case, the highly nonlinear noisy exclusive-OR (XOR) data classification problem. The k-dimensional XOR is a binary classification problem that can be formulated as

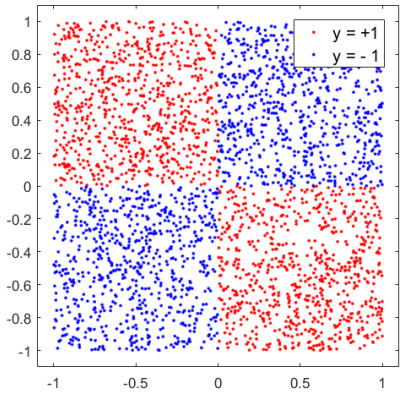
$$y(\mathbf{x}) = \begin{cases} +1 & \text{if } \prod_{i=1}^{k} x_i < 0 \\ -1 & \text{else} \end{cases} \tag{2.1}$$

Where data vector $\mathbf{x} \in \mathbb{R}^p$ is assumed to be sampled uniformly from $[-1, +1]^p$. Observe that in this formulation the XOR data is p dimensional but the degree of interaction is k-dimensional, with $k \leq p$. We call this data the k-D XOR in p dimensions. In this chapter we will work with $k \in \{3, 4, 5\}$, since $k = 2$ is a very simple case.

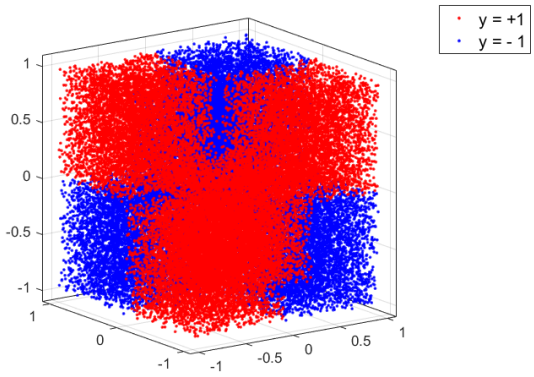
The XOR problem is an example of data that can only be modeled by using higher order feature interactions, and for which lower order marginal models have no discrimination power. This makes it very difficult to detect what features are relevant for predicting the response y. Figure 2.1 gives a visualization of the XOR data for the case $p = k \in \{2, 3\}$.

2.4 A Mathematical Formulation of Feed-Forward Neural Networks

Our experimental results show that a fully connected neural network with one hidden layer, ReLU activation for the hidden node and logistic for the loss can handle the non-noisy XOR data ($p = k \in \{3, 4, 5\}$) very well given sufficiently many training samples and hidden nodes. Figure 2.2 displays the test AUC (Area Under Curve) on the non-noisy XOR data with trained neural networks under the above setting. We can clearly see that, with sufficiently many training examples and hidden nodes, the curves of the test AUC will closely reach 1.0 within 200 training epochs in all cases. Thus, we are going to study and analyze these NNs on the same k cases but with sparse data and many irrelevant (noise) features. And we will see that in these sparse and noisy cases they



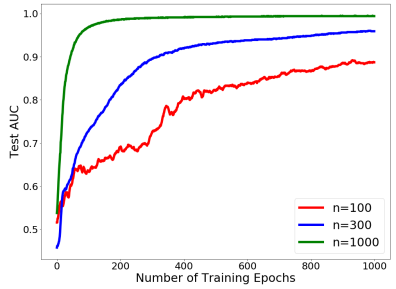
2D XOR with $p = 2$



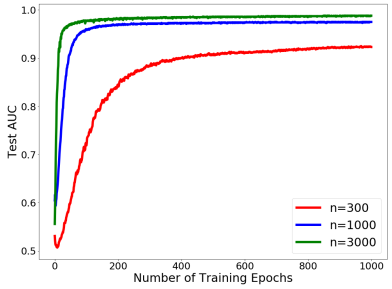
3D XOR with $p = 3$

Figure 2.1: Visualization of 2D and 3D XOR data without noisy features.

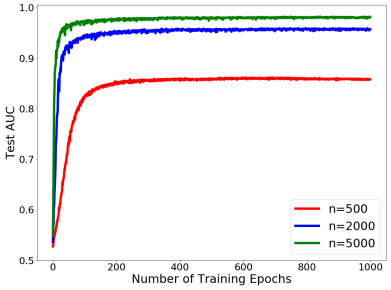
have difficulties finding the global optimum that obtains a good fit of the data. This is especially true when $p \gg k$.



$p = k = 3, h = 20$



$p = k = 4, h = 100$



$p = k = 5, h = 500$

Figure 2.2: Test AUC for non-noisy XOR data. n is the size of training and testing datasets. h is the number of hidden neurons.

Before we move on, we first want to formulate the NNs we want to study in a formal mathematical way. The neural networks we use for the nonlinear noisy XOR data are fully connected with one hidden layer and ReLU activation for the hidden node. Since we are studying binary classification, the output layer consists of only one node. If the hidden layer has h neurons and the input $\mathbf{x} \in \mathbb{R}^p$, we can represent the weights of the hidden nodes as vectors $\mathbf{w}_j = (w_{j1}, ..., w_{jp}, w_{jp+1})^T \in \mathbb{R}^{p+1}, j = 1, ..., h$, incorporating the bias term. The weights and bias

of the output neuron are a vector $\boldsymbol{\beta} = (\beta_1, ..., \beta_h)^T \in \mathbb{R}^h$, and $\beta_0 \in \mathbb{R}$, respectively. Denoting the ReLU activation as $\sigma(z) = \max(0, z)$ we can formulate the neural network classifier function as:

$$f(\mathbf{x}) = \sum_{j=1}^{h} \beta_j \sigma(\mathbf{w}_j^T \mathbf{x}) + \beta_0 = \begin{cases} > 0 & \text{predict} + 1 \\ < 0 & \text{predict} - 1 \end{cases} \tag{2.2}$$

Training neural networks of the form (2.2) based on the logistic loss $l(z) = \log(1 + \exp(-z))$ can be formulated as a non-convex unconstrained optimization problem as:

$$\min_{\mathbf{w}, \boldsymbol{\beta}, \beta_0} L(\mathbf{w}, \boldsymbol{\beta}, \beta_0) = \min_{\mathbf{w}, \boldsymbol{\beta}, \beta_0} \left\{ \sum_{i=1}^{n} \log \left(1 + \exp \left\{ -y_i \sum_{j=1}^{h} \beta_j \sigma(\mathbf{w}_j^T \mathbf{x}_i) + \beta_0 \right\} \right) \right\}, \tag{2.3}$$

where $(\mathbf{x}_i, y_i), i = 1, ..., n$ are the training examples.

There is no closed form analytic solution for the minimization problem (2.3), so we will commonly use a gradient descent based optimizer via backpropagation Werbos [1974] to minimize the loss in an iterative way.

Figure 2.3 shows a visualization example of a fully connected neural network for binary classification with one hidden layer and one output layer. The input layer is not a real layer with parameters for the neural network, it consists of the data sample features which feed into the NN for training or testing purpose.

2.5 The Loss Landscape of the NNs on the XOR Data

We follow Li et al. [2017] to visualize the loss landscape locally. First we find a deep optimum then use two directions in the parameter space as a system of reference based on which to compute the loss values on a grid. If we didn't find any local optima then we used two random directions. If we found exactly one other local optimum we used the direction towards the local optimum as the first direction and a random direction as the second direction. If we found at least two local optima different from the deep optimum, we used the directions towards the first two local optima we found as the reference directions.

Based on this strategy we obtained the loss maps as shown in Figure 2.4 for the 4D XOR data. We can clearly observe from the resulted figure that locally the loss landscape is smooth for non-

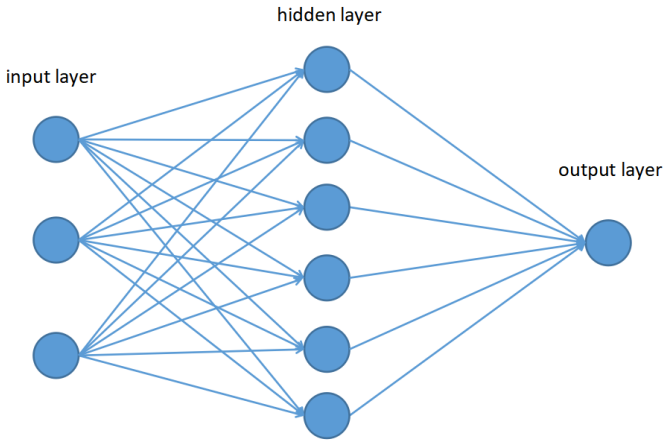
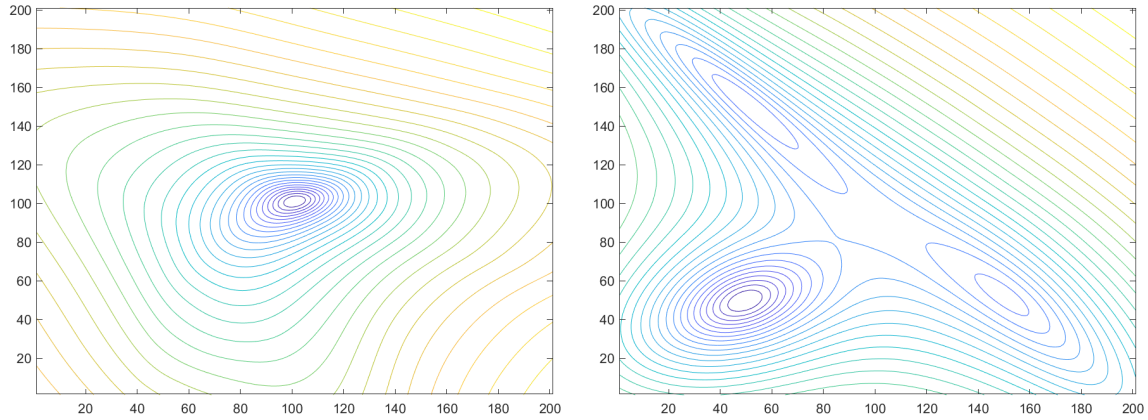


Figure 2.3: Example of a neural network for binary classification with one hidden layer and one output node.

noisy XOR data (i.e. the case $p = k = 4$), but there are several local minima on the noisy XOR data, especially for $p \gg k$.

To see the difference between the values of the various local minima, we ran the NNs for 1000 training epochs with 100 random initializations on a XOR dataset contains 3000 training and 3000 testing samples for $k = 3$, $k = 4$ and $k = 5$. We plotted in Figures 2.5, 2.6 and 2.7 the values of the local minima sorted in increasing order and their corresponding training and testing AUCs.

From Figure 2.5, 2.6 and 2.7 we can observe that, under the same nonlinearity level of k, the inequivalence degree of the local minima increases as the number of noisy features increases. For example, in $k = 4$ case, when $p = 4$ we see that all the sorted local minima have almost equivalent generalization power for NNs with one hidden layer complexities, to obtain a similar and very good training and testing AUCs (above to 0.98). But when $p = 27$, we can see for the NNs with 64 hidden nodes that the local minima are very different, and render the testing AUC lie in a wide range from 0.5 to 0.8. Increasing p to 100, we observe that the NNs can memorize the training data very well but lost their efficiency to deal the testing data. This is because in the $p = 100$ case, the XOR data becomes very sparse in the high dimensional space, and in such a sparse feature space, a neural network will have no difficulties finding a good pattern from the numerous local minima to fit the training data.



$p = 4$ with 100 hidden units $p = 14$ with 100 hidden units

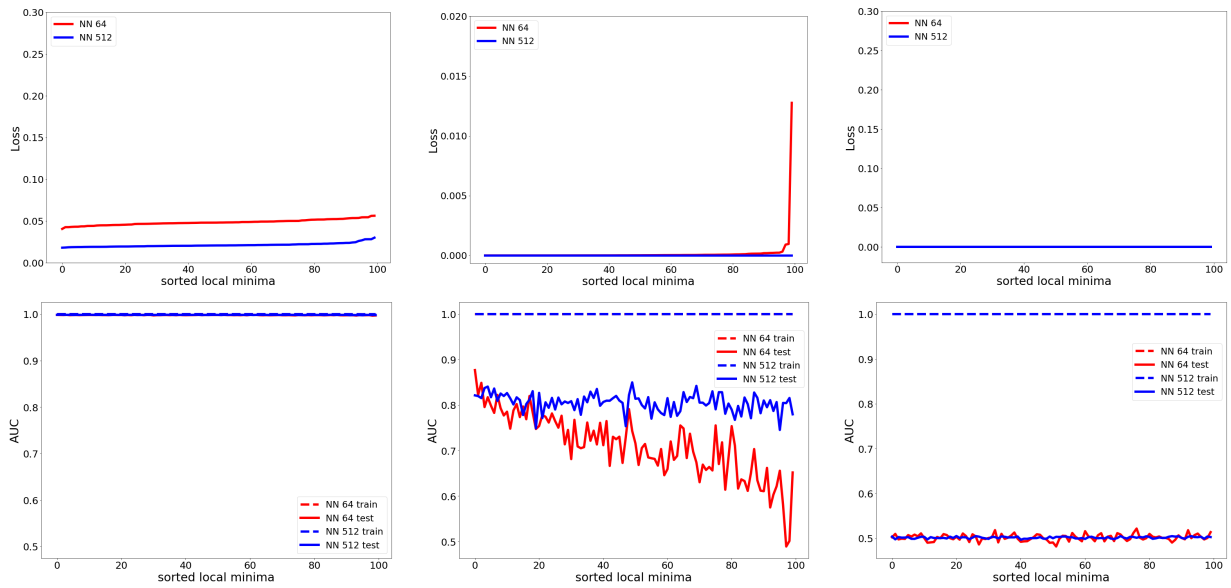
Figure 2.4: Loss landscape for 4D XOR ($k = 4$).

Besides observing in Figures 2.5, 2.6 and 2.7 that the difference between the local minima increases as p increases, we also observe that the increasing of k will lead to an increasing of local minima differences. Firstly we look at $\{k = 3, p = 3\}$ case, the perfect training and testing AUCs (exactly or close to 1.0) are obtained by almost all the local minima. Then we check case $\{k = 5, p = 5\}$, we see the value of local minima fluctuates in a significant range and will not always have a similar value, and this fluctuation is also reflected in the distribution of their testing AUC. Not all the testing AUCs of those local minima will get perfect value of 1.0, but many of them will still achieve very good values like 0.9 but with a clearly visible distance from the perfect case.

Finally, we observe that increasing the number of hidden nodes of NNs could improve the NN's capability to reach a deeper local optimum on XOR data from Figure 2.5, 2.6 and 2.7. In all three different k cases, the NNs with 512 hidden nodes could always outperform or at least be in the same level compared to the NNs with 64 hidden nodes in reaching a smaller values of of the loss function.

2.6 The Trainability of Neural Networks

To see the change from an easily trainable NN to a poorly trained NN for each dimension p, we train a NN with 512 hidden nodes starting from 10 random initializations and keeping the solution with smallest loss value. Then we compute the test AUC of the obtained NN (the training dataset



$n = 3000, p = 3$ $n = 3000, p = 35$ $n = 3000, p = 100$

Figure 2.5: Values of sorted local minima (top) and train and test AUC (bottom) for 3D XOR.

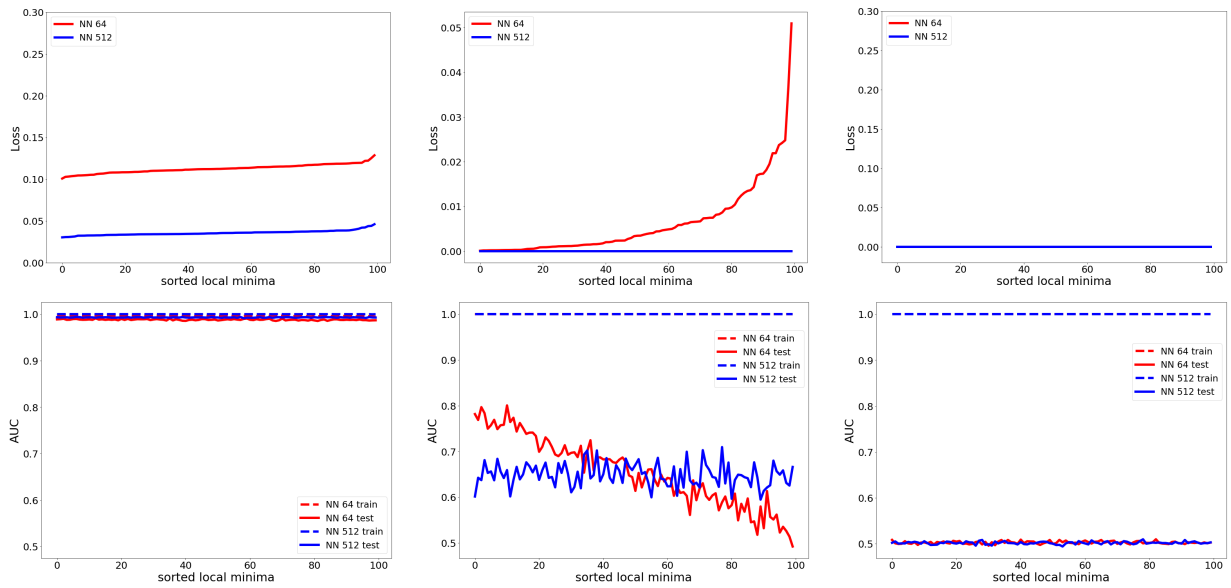
size $n \in \{1000, 3000, 10000\}$, testing dataset size is set to be n in each case). For each p we repeat this process 10 times and display in Figure 2.8 the average test AUC vs p.

From Figure 2.8 we observe that the test AUC quickly drops from close to 1 to 0.5. The number of variables p where the test AUC gets below a threshold (e.g. 0.8) depends on the number n of training examples. This drop does resemble a phase transition, from an "easy to train" regime where the local minima are easy to find, to the "hard to train" regime.

Finally, to see how hard to find are the local minima, we compute the hit time, which we define as the average number of random initializations required to find a local minimum with a train AUC of at least 0.95. The hit time is displayed in Figure 2.9 for NNs with 20 hidden nodes and $n = 3000$ training and test observations. Observe that the hit time quickly blows up as p increases. It is impractical to learn NN models on 3D, 4D or 5D XOR data when there are hundreds of irrelevant variables.

In summary, our observations from all above study are the following:

- If the training data is difficult (such as the XOR data), not all local minima are equivalent, since in Figure 2.5, 2.6 and 2.7 there was a large difference between the loss value and the test AUC of the best local minimum and the worst one.



$n = 3000, p = 4$ $n = 3000, p = 27$ $n = 3000, p = 100$

Figure 2.6: Values of sorted local minima (top) and train and test AUC (bottom) for 4D XOR.

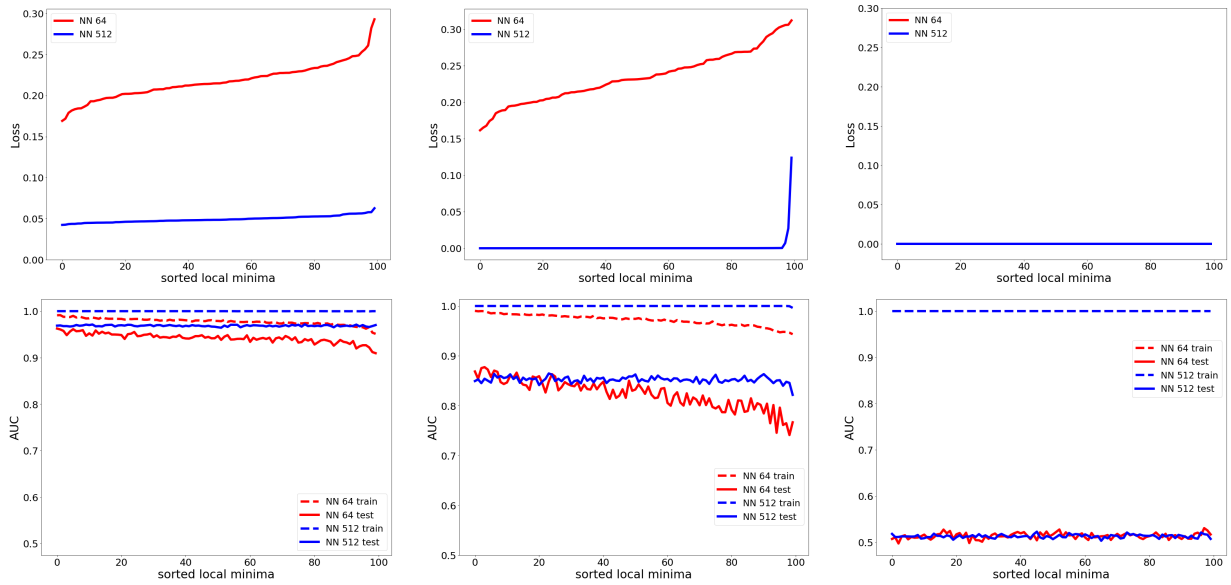
- For a fixed training size n, the number of shallow local minima quickly blows up as the number of irrelevant variables increases and finding the deep local minima becomes extremely hard.
- If the number of irrelevant variables is not too large, an NN with a sufficiently large number of hidden nodes will find a deep optimum more often than one with a small number of hidden nodes.

These observations form the basis for the proposed node and feature selection methodology presented in Section 2.8.

2.7 Existence of Junk Nodes

The study from the previous sections showed how difficult it is to obtain a trained neural network that can find a good deep local optimum on highly non-linear data with many noisy features. In the following sections, we will introduce our methodology that tries to address this issue.

As we see from Figure 2.8 for $k \in \{3, 4, 5\}$, the NNs can handle the XOR data if p is small. For example, in the training size $n = 3000$ case, the NNs still can work for $p \leq 50$ with $k = 3$, $p \leq 35$ with $k = 4$ and $p \leq 20$ with $k = 5$. However even when p is in the range where the NN can work,

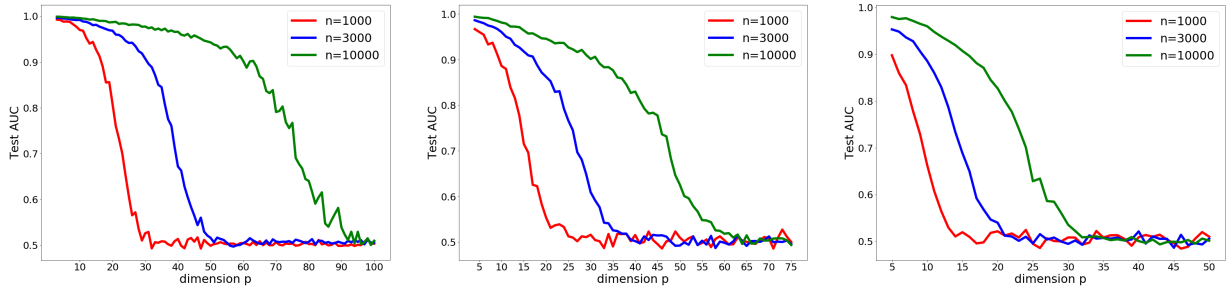


$n = 3000, p = 5$ $n = 3000, p = 10$ $n = 3000, p = 100$

Figure 2.7: Values of sorted local minima (top) and train and test AUC (bottom) for 5D XOR.

the generalization power severely decreases as p increases, with a phase transition from "easy to train" to "hard to train".

Dropout [Hinton et al., 2012] is one of the most commonly used techniques to increase the model generalization capability and reduce overfitting in neural network training nowadays. In each step of the training process, it will randomly kill a pre-specified proportion of hidden nodes and make the current neural network smaller than the original one. This technique can de-correlate the unnecessary "strong" connections between neurons and give a higher chance for all nodes to be



$k = 3$ $k = 4$ $k = 5$

Figure 2.8: Test AUC of best energy minimum out of 10 random initializations vs. data dimension p for a NN with 512 hidden nodes.

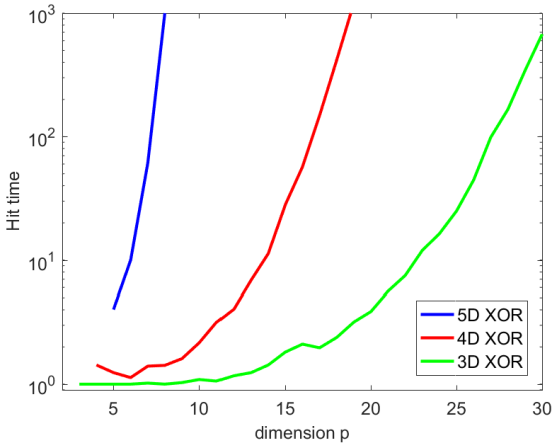
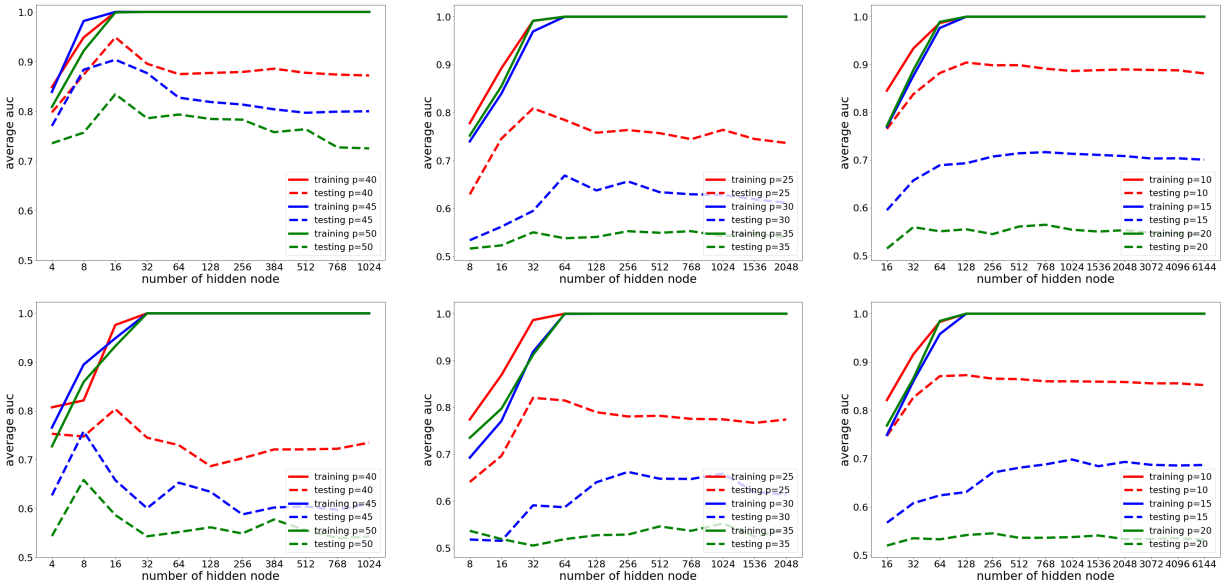


Figure 2.9: Hit time vs dimension p for different XOR problems with $n = 3000$ observations.

trained sufficiently. At the end of training, the original neural network is expected to obtain better performance in prediction on unseen data. Hereby we also want to apply the Dropout method and see whether it can help improve the generalization ability of NNs in the noisy XOR classification problem beyond the phase transition regime.

We select 3 experiment cases in the "easy to train" to "hard to train" phase for each $k \in \{3, 4, 5\}$ with 3000 training samples respectively, and train a NN with 10 random initializations with Dropout of the default dropping probability 0.5. Then we keep the best test AUC and its associated training AUC among the 10 trials. We repeat this process 10 times and display Figure 2.10 the average test and train AUC vs the number of hidden node. Comparing results between with and without Dropout in Figure 2.10, we observe that the Dropout technique indeed can help to reduce the overfitting and increase the NNs generalization power, especially in the $k = 3$ case. However as the level of XOR data nonlinearity increases, the efficiency of the Dropout becomes weaker and weaker. In the $k = 5$ case, we can hardly see the performance improvement of Dropout compared to the naive NNs in Figure 2.7. We also observe that as the number of hidden nodes increases, the training AUC becomes better and better to finally reach 1.0. But the test AUCs reach a peak when the number of hidden nodes is relatively small, and then no further improvement happens as the number of hidden nodes increases. This tells us that increasing the number of hidden nodes will make too many irrelevant hidden nodes exist in NNs, and overfitting occurs so severely that even Dropout can not handle it, making it very hard to find a deep local minimum.



$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.10: Training and testing AUC with (top) and without (bottom) Dropout vs. number of hidden nodes.

We propose a tentative explanation of the failure of Dropout on noisy XOR data for large numbers of hidden nodes. If we carefully check the formulation of the labels of the XOR data, we could find that for any value of k, there exists a decision tree stacked by a set of rules to induce the target label. Figure 2.11 is a visualization of this decision tree formed by such rules for the 2D XOR data. This kind of decision tree gives the solution exactly, i.e. t can obtain the global minimum of the loss function used in neural networks training. Unfortunately, it is not realistic to obtain this global minimum during the NN training process, since the weight initialization is random and so many local optima exist to trap us from getting the true global minimum. But this tree still shows us some hints about approximately how many hidden neurons are needed to make the NN capable to capture the true mode. That is, if we account for the internal nodes of the induced decision tree as our neural nodes, roughly we could estimate how many hidden nodes we need in NNs — and with a good weight initialization, we only need a small number of hidden nodes to power up the NNs.

This observation is further demonstrated by checking the result we get in Fig 2.10. We can see that all of the experimental cases can achieve roughly very good test AUC results with a small

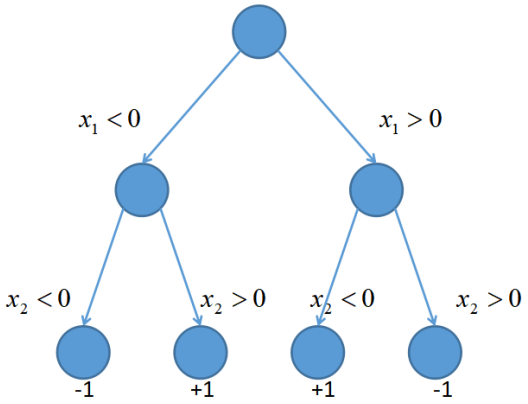


Figure 2.11: Decision tree generated by the label formulation of 2D XOR data.

number of hidden nodes ($k = 3$ with 16 hidden nodes, $k = 4$ with 32 hidden nodes and $k = 5$ with 128 hidden nodes). Increasing the number of hidden nodes on one hand indeed brings the NNs more opportunities to find a deep local optimum, but it also introduces a lot of noisy nodes that have no contribution to the NN in capturing the true mode. Dropout's random node dropping ensures that all hidden nodes in NNs are well trained, de-correlates some unnecessary connections among the hidden nodes, which is highly useful for those NNs with a small number of hidden nodes. But for the case of NNs with many hidden nodes, a large proportion of hidden nodes might be "junk" nodes, and training them well will not lead to any improvement in the NN's capability, even it can be harmful to the NN model.

2.8 Node Selection

In this section we introduce a node selection technique that gradually removes hidden nodes, inspired by the Feature Selection with annealing (FSA) [Barbu et al., 2017] method, which will be described next.

2.8.1 Overview of the FSA algorithm

Feature Selection with Annealing (FSA) [Barbu et al., 2017] is a recently proposed feature selection method dealing mainly with the linear noisy data. It gradually removes features based on the magnitude of their associated parameter weights in the linear classifier according to a

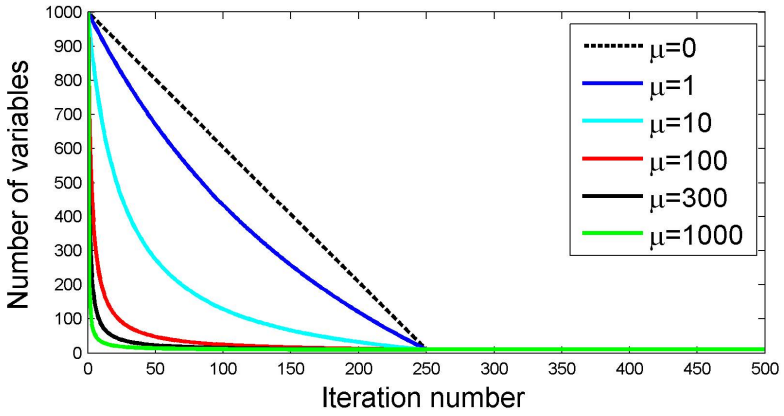


Figure 2.12: The number of kept features M_e vs iteration e for different schedules with $p = 1000$, $k = 10$, $N^{epoch} = 500$. Figure source comes from Barbu et al. [2017]

pre-specified annealing drop schedule M_e during the training process. At the end of training, only a certain number of relevant features with larger parameter weights will be kept, and the features with smaller weights will be all dropped. This method can perform the feature selection simultaneously with the training process, and has been shown to work quite well in linear regression and classification problems.

Suppose we are given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \{-1, +1\}$, and want to train a binary linear classifier $f(x) = \boldsymbol{\beta}^T \mathbf{x} + \beta_0$. If we further define a differentiable loss function $L(\boldsymbol{\beta}, \beta_0)$ based on this training dataset, we can formulate the feature selection problem on such classifier as a constrained optimization as following:

$$\begin{aligned} \min_{\boldsymbol{\beta}, \beta_0} \quad & L(\boldsymbol{\beta}, \beta_0) \\ \text{s.t.} \quad & ||\boldsymbol{\beta}||_0 \leq k \end{aligned} \tag{2.4}$$

where k is a preset hyperparameter that defines the upper limit of the total number of non-zero feature weights.

The idea of FSA lies in: the feature selection can be performed at the same time with the loss minimization that trains a classifier. The number of features kept at each epoch e is controlled by a deterministic annealing plan M_e. In every training step, this annealing schedule will eliminate a certain number of irrelevant features based on the associated weight magnitudes and keep the total number of features to be exactly M_e. The feature dropping or killing schedule will continuously be enforced unless the remaining total number of features at most k. The exact mathematical

formulation of the annealing function M_e is shown below:

$$M_e = k + (p - k) \max\left(0, \frac{N^{epoch} - 2e}{2e\mu + N^{epoch}}\right) \tag{2.5}$$

where p is the dimension of the feature vectors, and the hyperparameter μ is prespecified by the user to provide a balance between efficiency and accuracy. Figure 2.12 gives a visualization for M_e for six different choices of μ with $p = 1000$, $k = 10$ and $N^{epoch} = 500$.

The FSA also enjoys theoretical guarantees of consistency and convergence. If both the learning rate and feature dropping schedule are sufficiently slow, the FSA algorithm will find all the k true feature variables with high probability. The detailed description of the FSA algorithm is displayed in Algorithm 1.

Algorithm 1 Feature Selection with Annealing (FSA)

Input: Normalized training set $\{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}\}_{i=1}^N$

Output: Trained model $f_{\boldsymbol{\beta}}(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x}$ with parameter vector $\boldsymbol{\beta}$.

1: Initialize $\boldsymbol{\beta} = 0$.
2: **for** e = 1 to N^{iter} **do**
3: Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta\frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$
4: Keep only the M_e variables corresponding to the highest $|\beta_j|$.
5: **end for**

2.8.2 Node Selection with Annealing Schedule

A straightforward way to try to improve the NN's model performance is to increase the number of hidden nodes and to apply Dropout while training the NN. The experiments and analysis from Section 2.7 indeed show that Dropout can improve to some extent the NNs' generalization capability for the nonlinear noisy XOR data. However this kind of improvement will significantly decrease as the nonlinearity level of the XOR data increases. Moreover, our tentative explanation indicates that as the number of hidden nodes increases, there will exist a lot of "junk" nodes in the NN model, leading to a degree of overfitting that even Dropout cannot handle. In Figures 2.5, 2.6 and 2.7 we also show that a NN with many hidden nodes can more easily find a deep local optimum for which the training loss value is significantly smaller than a NN with a small number of hidden nodes (although this deep local optimum might not have a very good generalization performance to the test set). Thus if we can find a method to start training these NNs with many hidden nodes,

and remove the so-called junk nodes, and only keep the important hidden neurons at the end of training, we may obtain a neural network with better generalization performance in the "easy to train" to "hard to train" phase for nonlinear noisy XOR data.

The noisy XOR data classification is a totally nonlinear problem and can not be handle by any kind of linear classifier. However we should notice that a fully connected neural network can be thought as a ensemble of many linear classifiers, one neuron could account for one linear classifier in some sense. Mathematically, for these NNs with one hidden layer and one output neuron, if we treat the activation $\sigma(\cdot)$ of hidden neurons as the input feature vector $\boldsymbol{a}$ of a linear classifier, we can transform a binary NN classifier (2.2) into a binary linear classifier such as:

$$f(\mathbf{x}) = \sum_{j=1}^{h} \beta_j \sigma(\mathbf{w}_j^T \mathbf{x}) + \beta_0 = \sum_{j=1}^{h} \beta_j a_j + \beta_0 = \boldsymbol{\beta}^T \boldsymbol{a} + \beta_0, \tag{2.6}$$

where a_j is the activation of the j-th hidden node, and we treat it similarly as the input x_j in a linear classifier.

After this transformation, we now can train a NNs initialized with many hidden nodes, then gradually drop the hidden nodes based on the magnitude of the associated weights $|\beta_j|$ of the output neuron during the training, and finally keep a few relevant hidden nodes at the end. Since training a neural network is non-convex optimization, it is better to pre-train the NNs to reach a local optimum and then apply the node selection with annealing schedule to escape and go deeper. Thus we can modify the annealing schedule M_e as follows:

$$M_e = \begin{cases} p & 1 \leq e \leq N^{pretrain} \\ k + (p-k)\max\left(0, \frac{(N^{epoch} - N^{pretrain}) - 2e}{2e\mu + (N^{epoch} - N^{pretrain})}\right) & N^{pretrain} < e \leq N^{epoch} \end{cases} \tag{2.7}$$

and the proposed method for selecting the nodes is presented in Algorithm 2.

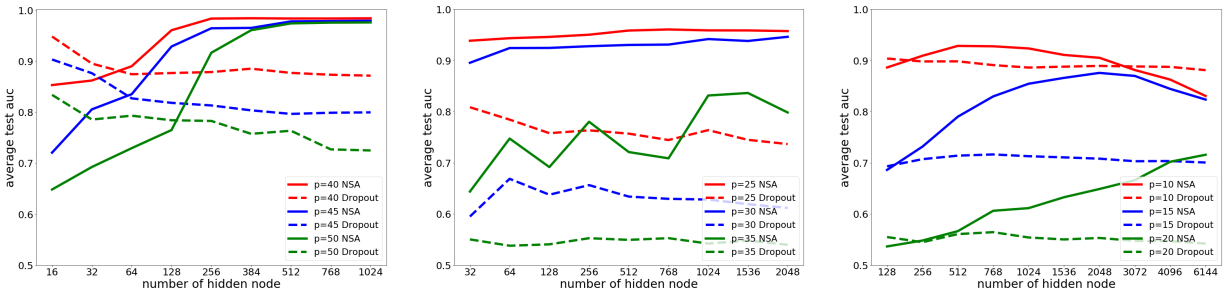
In order to have a fair comparison with the Dropout method on neural networks, we follow the same experiment pipeline as we did in Section 2.7. We train NNs with the same initial number of hidden nodes as we tried in Dropout, and apply NSA during the training, to finally keep 8 hidden nodes for $k = 3$, 16 hidden nodes for $k = 4$ and 64 hidden nodes for $k = 5$ at the end of training. We display the comparison average test AUC results between NN+NSA and NN+Dropout in Figure 2.13.

Algorithm 2 Node Selection with Annealing (NSA)

Input: Training set $T = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}\}_{i=1}^n$, desired number h of hidden neurons, starting number H of hidden neurons, annealing schedule $M_e, e = 1, .., N^{iter}$.

Output: Trained NN with h hidden neurons.

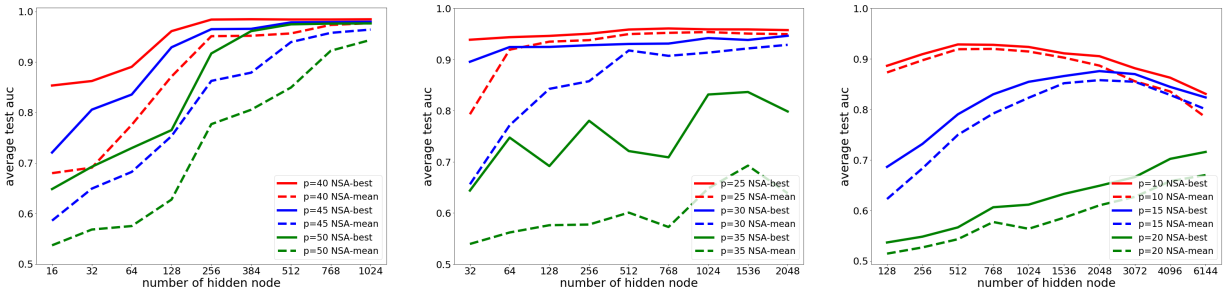
1: Initialize a NN with H hidden neurons with random initialization
2: **for** $e = 1$ to N^{iter} **do**
3: Update $\boldsymbol{w}$, $\boldsymbol{\beta}$ and β_0 via backpropagation with a gradient descent based optimizer
4: Remove hidden nodes to keep the M_e nodes with largest $|\beta_j|$
5: **end for**



$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.13: Average test AUC vs number of hidden nodes for NN+NSA and NN+Dropout.

From Figure 2.13, we can see that the NN+NSA can obtain a higher best test AUC than the NN+Dropout in every case. With the procedure gradually removing the junk hidden nodes, the NNs avoid to suffer the severe overfitting issue and prevent to be trapped in a shallow local optimum or a bad deep optimum, and enjoy a far higher chance than Dropout to reach a optimum that has good generalization. These results achieved by NN+NSA also prove our tentative explanation of the failure of Dropout drawn in Section 2.7, that we only need to have a small number of hidden nodes to capture the true mode for nonlinear XOR noisy data classification. However, we also see some weird things happened with NN+NSA. For example in the $\{k = 4, p = 35\}$ case, although we see that the test AUC increases as the number of hidden nodes increases, the curve is not a continuously increasing curve but has a zig-zag path, which indicates that the NSA could not consistently power up these NNs in finding a good deep local optimum. Also in the $\{k = 5, p = 10\}$ and $\{k = 5, p = 15\}$ cases, we do not see that the test AUCs increasing continuously or at least remaining the same as the number of hidden nodes increases — we see that the AUCs go up for a



$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.14: Test AUC of 100-run averaged and average of 10 best trials vs number of hidden nodes for NN+NSA.

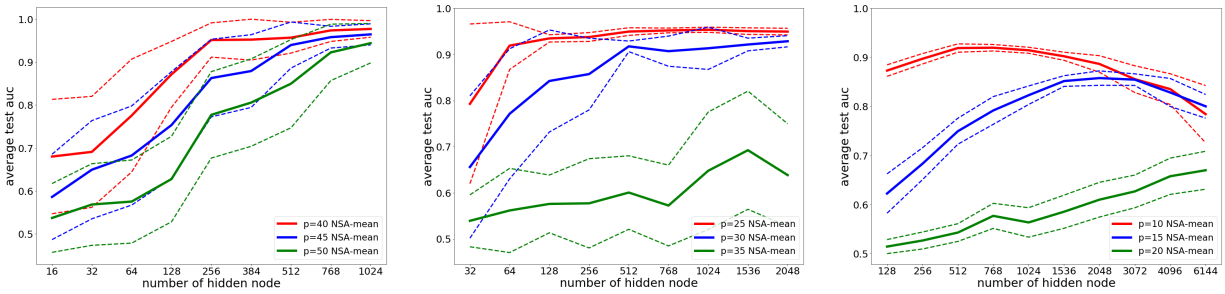
while as the number of hidden node increases, but then starts to go down with the further increase in the hidden node number. We will discuss and analyze these kind of issues in the next section.

2.8.3 Neural Network Generalization Consistency with NSA

As training neural networks is a non-convex optimization problem, there are two aspects regarding the measurement of how good a technique is in improving the NN generalization capability: one aspect is what is the best NN performance this technique can achieve; the other aspect is how consistent or how often we can get such best performance using this method.

From Section 2.8.2, we can see that our node selection method NSA can always achieve a better best test AUC than the Dropout in all $k \in \{3, 4, 5\}$ cases. But we also see that NSA seems not very stable in some cases such as the zig-zag test AUC curve for case $\{k = 4, p = 35\}$ and the up-down test AUC curve for cases $\{k = 5, p = 10\}$ and $\{k = 5, p = 20\}$. These uncommon curves shown in Figure 2.13 may indicate that NSA would not consistently obtain the best test AUC level results among all the 10 random initializations in one trial of some "hard to train" cases.

In order to see the consistency or frequency of NSA to get the best test AUC level results reported in Figure 2.13, we perform another experiment. We compute the overall average test AUC among all 100 trials for each tried number of hidden nodes, and make a comparison of the NSA results from Figure 2.13, which are the average of best test AUC from 10 runs (each run records the best test AUC from 10 random initializations). We want to see how close or how far these two different average test AUCs is. The comparison results for all $k \in \{3, 4, 5\}$ cases are shown in Figure 2.14.



$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.15: 100-run averaged test AUC vs number of hidden nodes for NN+NSA. The solid lines are the means, dashed lines are mean $\pm$ std.

What we can see in Figure 2.14 are: for $k = 3$, the averaged test AUCs for different p cases all increase as the number of hidden nodes increases. But the increase in p leads to a weaker increase in the average AUC. We can see the difference between the 100-trial averaged test AUC and averaged best test AUC of 10 runs for case $p = 50$ is much larger than the case $p = 40$. For $k = 4$, we also see a similar phenomenon, the difference between the two averaged curves grows larger and larger as the XOR data becomes more and more noisy. Especially for the $p = 35$ case, we can see the 100-trial averaged test AUC is significantly worse than the averaged best test AUC of 10 runs for any tried initial number of hidden nodes. For $k = 5$, in all p cases, the difference between the two averaged curves does not get closer and closer as we should expect when the number of initial hidden nodes increases — the value of difference first gets smaller and then gets larger.

To better check the consistency of the test AUC for different random initializations of NN with NSA, we display in Figure 2.15 the average test AUC result from 100 random initializations plus and minus the standard deviation of these 100 trials. It is more clear to see in Figure 2.15 that for the "hard to train" cases, such as $\{k = 3, p = 40\}$, $\{k = 4, p = 35\}$ and $\{k = 5, p = 20\}$, the NNs with NSA do not consistently have a good generalization capability as the number of initial hidden nodes increases. We do not see the standard deviation continuously decreasing along with the increase in the hidden node number.

From the experiments displayed in Figures 2.13, 2.14 and 2.15, we demonstrate that the NSA method can significantly improve the NNs generalization power for nonlinear noisy XOR data in the transition phase from the "easy to train" to the "hard to train" regime for $k \in \{3, 4, 5\}$. But in the meantime, we also demonstrate that the naive NSA, which simply drops the hidden nodes with

smaller associated weights connecting to the output neuron cannot consistently help the NNs find a good deep optimum, especially for the "hard to train" cases. We need to find a way to improve the generalization consistency on the naive NSA.

2.8.4 Neural Node Normalization

The naive NSA comes out from the assumption that for a two layer neural network displayed in Figure 2.3, the weights that connect the hidden neurons to the output node could represent the importance levels of those hidden nodes in NN. By keeping the hidden neurons with larger weight magnitudes and dropping those with smaller ones, we expect to preserve the most relevant nodes that make a positive contribution and remove those junk nodes that may introduce noise information to the neural network. After doing this, we hope to finally reduce overfitting and improve the NN's generalization capability.

This kind of assumption will hold true for the case of training a linear classifier, as the input features are directly connected to the output neuron, and the training dataset would always hold the same or similar statistical distribution (depending on how we do data preprocessing) for each input feature. But in neural network training, it will not be the case since the activation of hidden neurons will change as its internal weight vector will be updated in the backpropagation procedure for every training step. Thus, it should be unreasonable to consider those weights connecting the hidden neurons to the output node as the unique measure of the importance of hidden nodes in such neural networks.

If we carefully look at the mathematical form of the neural network's classifier function in Eq. (2.2), we can notice that a hidden neuron's effectiveness to the neural network should lie in two parts:

$$f(\mathbf{x}) = \sum_{j=1}^{h} \left(\underbrace{\beta_j}_{I} \cdot \underbrace{\sigma(\mathbf{w}_j^T \mathbf{x})}_{II} \right) + \beta_0$$

The first part β is the weight we consider to be the hidden node importance measurement as in naive NSA. The second part is the activation level of the hidden node, which was simply overlooked before. In Section 2.8.2, we unintentionally treated the second part to be a fixed value for each training sample during the node selection process. But in fact both parts will change their values in the backpropagation update.

That implies that the second part will also carry importance information about the hidden node for keeping or dropping consideration. The value of the hidden node activation is mainly determined by the inner product between the internal weight vector $\mathbf{w}$ and data vector $\mathbf{x}$. As the input vector $\mathbf{x}$ is fixed during every training step, the change of each hidden node activation roots in the change of the internal weight $\mathbf{w}$. We can see that when a test data sample goes through the above two layer neural network shown in Figure 2.3, if we fix the value of β for all hidden nodes, a hidden node with a larger magnitude value of the inner product should have a larger contribution to the prediction than a smaller one. Thus, if we do not extract the importance information stored in the activation of each hidden node and incorporate it into the weight β, we cannot truly keep those hidden neurons carrying larger hidden node importance information via NSA.

Another thing we have to consider arises from the fact that the change in each hidden node activation will not be at the same pace during the backpropagation update in the training process. Some hidden nodes' internal weights may change quickly, some may change slowly. This unbalanced internal weight update pace of hidden nodes will lead the statistical distribution of the activation based on the training dataset for each hidden node to be different. To achieve a better node selection, we need to find a way to convert the activation of all the hidden nodes to the same scale.

At first glance, to extract the importance information from the hidden node activation and to convert the activation of all hidden node to the same scale should be addressed totally separately. However, if we carefully select the extraction criterion and scaling method, actually we could handle this two issues simultaneously.

We see that the degree of importance information stored in the hidden node activation is mainly determined by the inner product of the internal weight vector $\mathbf{w}$ and data vector $\mathbf{x}$. Since the data vector $\mathbf{x}$ is usually fixed during training, the magnitude of internal weight vector $\mathbf{w}$ should be the main source account for how much importance information the activation carries. The $L2$-norm is one of the best and most commonly used ways to measure the magnitude of a vector. So here we will utilize the $L2$-norm to compute the node importance information carried by the hidden node activation. Hereby, for the j-th hidden node, the node importance information number stored in its activation is defined as following:

$$I(j) := \left\|\mathbf{w}_j\right\|_2 = \sqrt{\sum_{l=1}^{p+1} w_{jl}^2} \tag{2.8}$$

Next we deal with the hidden node activation rescaling problem. In order to determine the distribution of a hidden node activation based on the training dataset, we first need to derive the value range of the inner product between internal weight vector $\mathbf{w}$ and data vector $\mathbf{x}$ as this is the key source to cause the unbalanced update pace of different hidden nodes. As we assumed that the XOR data is uniformly distributed in range $[-1, +1]^p$, we calculate the norm of such inner product for j-th hidden node as:

$$\begin{aligned} ||\mathbf{w}_j^T \mathbf{x}|| &= ||\mathbf{w}_j|| \cdot ||\mathbf{x}|| \cdot \underbrace{\cos\theta}_{\leq 1} \\ &\leq ||\mathbf{w}_j|| \cdot \underbrace{||\mathbf{x}||}_{\leq 1} \\ &\leq ||\mathbf{w}_j|| \end{aligned}$$

where θ is the angle between two vectors $\mathbf{w}_j$ and $\mathbf{x}$. One could clearly see from the derived inequality that the value range of hidden node activation is determined by the magnitude of the internal weight vector $\mathbf{w}$. Thus, to rescale the activation of different hidden nodes for the training dataset to the same scale is roughly equivalent to rescaling each hidden node's internal weight vector $\mathbf{w}$ to the same scale. For internal weight vectors with different magnitude, a commonly way to convert them into the same scale is to divide the weight vector by its own $L2$-norm respectively. Therefore, the rescaling factor for the j-th hidden node can be defined as:

$$S(j) := \frac{1}{\left\|\mathbf{w}_j\right\|_2} = \frac{1}{\sqrt{\sum_{l=1}^{p+1} w_{jl}^2}} \tag{2.9}$$

After we determined the extraction criterion and the rescaling factor for hidden node $j \in \{1, 2.., h\}$, we now are able to multiply β_j by $I(j)$ to incorporate the hidden node importance information in one place, and multiply $\mathbf{w}_j$ by $S(j)$ to rescale all the hidden node to the same scale. Furthermore, this two kinds of multiplication even will not change our neural network classifier function but just transform it to another form due to the usage of the ReLU as our hidden node

activation. The following shows the mathematical calculation in detail:

$$
\begin{aligned}
f(\mathbf{x}) &= \sum_{j=1}^{h} \beta_j \sigma(\mathbf{w}_j^T \mathbf{x}) + \beta_0 \\
&= \sum_{j=1}^{h} \beta_j \cdot \left\|\mathbf{w}_j\right\|_2 \cdot \frac{1}{\left\|\mathbf{w}_j\right\|_2} \cdot \sigma(\mathbf{w}_j^T \mathbf{x}) + \beta_0 \\
&= \sum_{j=1}^{h} \left(\beta_j \cdot \left\|\mathbf{w}_j\right\|_2\right) \cdot \frac{1}{\left\|\mathbf{w}_j\right\|_2} \cdot \max(0, \mathbf{w}_j^T \mathbf{x}) + \beta_0 \\
&= \sum_{j=1}^{h} \left(\beta_j \cdot \left\|\mathbf{w}_j\right\|_2\right) \cdot \left(\max\left(0, \frac{\mathbf{w}_j^T \mathbf{x}}{\left\|\mathbf{w}_j\right\|_2}\right)\right) + \beta_0 \\
&= \sum_{j=1}^{h} \underbrace{\left(\beta_j \cdot I(j)\right)}_{incorporation} \cdot \underbrace{\left(\max\left(0, (\mathbf{w}_j^T \mathbf{x}) \cdot S(j)\right)\right)}_{normalization} + \beta_0 \\
&= \sum_{j=1}^{h} \tilde{\beta}_j \sigma(\tilde{\mathbf{w}}_j^T \mathbf{x}) + \beta_0
\end{aligned}
$$

Now we present the node selection with neural node normalization via annealing schedule in Algorithm 3.

Algorithm 3 Node Selection with Normalization and Annealing (NSNA)

Input: Training set $T = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}\}_{i=1}^n$, desired number h of hidden neurons, starting number H of hidden neurons, annealing schedule $M_e, e = 1, .., N^{iter}$.

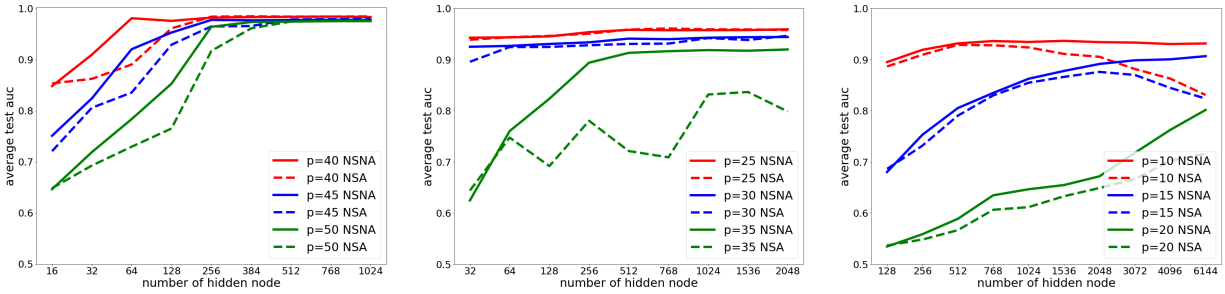
Output: Trained NN with h hidden neurons.

1: Initialize a NN with H hidden neurons with random initialization
2: **for** $e = 1$ to N^{iter} **do**
3: Update $\boldsymbol{w}$, $\boldsymbol{\beta}$ and β_0 via backpropagation with a gradient descent based optimizer
4: **for** $j = 1$ to h **do**
5: Normalize hidden node j and incorporate the normalizer to β_j:

$$\tilde{\beta}_j \leftarrow \|\mathbf{w}_j\| \beta_j, \tilde{\mathbf{w}}_j \leftarrow \frac{\mathbf{w}_j}{\|\mathbf{w}_j\|} \tag{2.10}$$

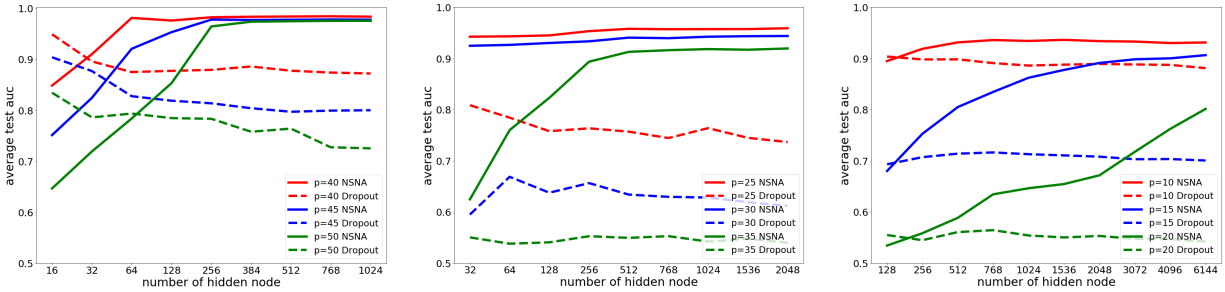
6: **end for**
7: Remove hidden nodes to keep the M_e nodes with largest $|\tilde{\beta}_j|$
8: **end for**

Figure 2.16 and 2.17 show the comparison result between NSNA and NSA, NSNA and Dropout. The experiment pipeline to apply NSNA on NNs for nonlinear noisy XOR data is the same as we did



$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.16: Average test AUC vs number of hidden nodes for NNs with NSNA or NSA.

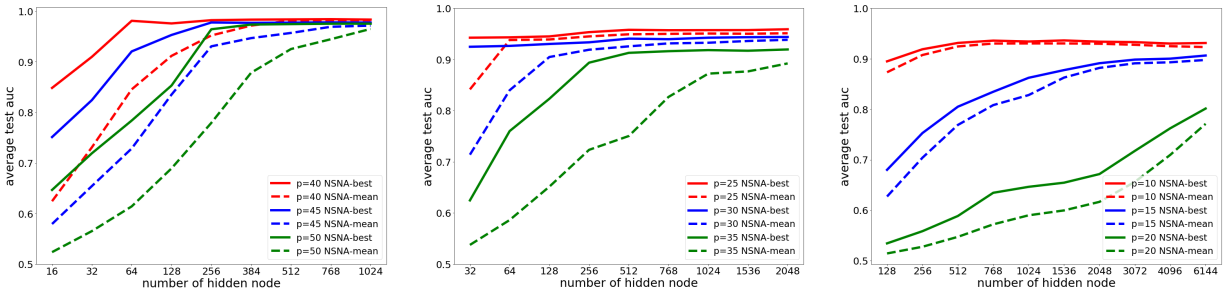


$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.17: Average test AUC vs number of hidden nodes for NNs with NSNA or Dropout.

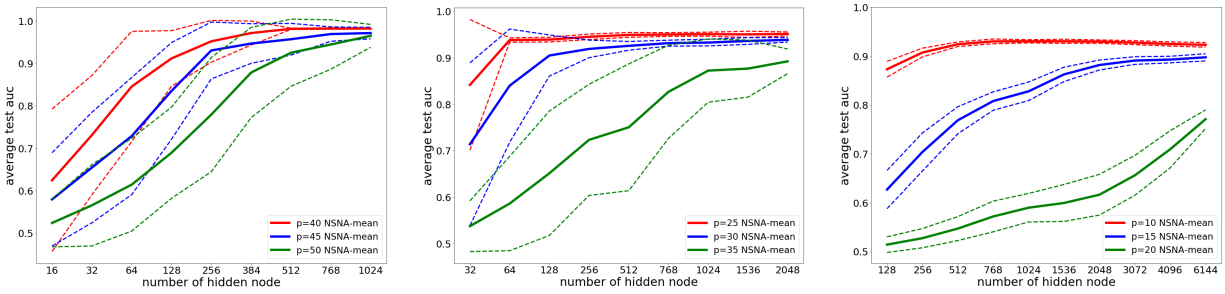
for NSA. Clearly to see for all cases, those NNs with NSNA can always obtain the best generalization level capability among the three. Comparing NSNA to NSA, we can see that NSNA is more efficient than NSNA in helping neural networks achieve good testing performance. For example in $k = 3$, the NNs with NSNA could reach the best generalization level with fewer hidden nodes than NNs with NSA. Moreover, unlike NSA which had a weird zig-zag AUC curve in case $\{k = 4, p = 35\}$, and up-down AUC curve in case $\{k = 5, p = 10\}$ and $\{k = 5, p = 15\}$, the NNs' generalization level in NSNA continuously improves or at least stays the same as the initial number of hidden nodes increases. This even holds true in the "hard to train" case for each k.

In Figure 2.16 and 2.17, we have demonstrated that NSNA outperforms the NSA and the Dropout in improving neural network generalization, when dealing with the nonlinear noisy XOR data for cases $k \in \{3, 4, 5\}$ and the number of features in the range where the NNs' generalization capability quickly decreases from an "easy to train" to a "hard to train" regime. Besides that, we are also interested in how consistent is the NSNA method in assisting NNs reach a deep local



$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.18: Test AUC of 100-run averaged and average of 10 best trials vs number of hidden nodes for NN+NSNA.



$k = 3, p = \{40, 45, 50\}$ $k = 4, p = \{25, 30, 35\}$ $k = 5, p = \{10, 15, 20\}$

Figure 2.19: 100-run averaged test AUC vs number of hidden nodes for NN+NSNA. The solid lines are the means, dashed lines are mean $\pm$ std.

optimum with good generalization. So we display the comparison result between 100-run averaged test AUC and averaged test AUC of 10 best trials for each tried initial number of hidden nodes, and the comparison result of 100-run averaged test AUC with the same curve but plus and minus the standard deviation of such 100 runs in Figures 2.18 and 2.19 respectively.

From above figures, we see that the result is much better than the naive NSA from the previous section. In all cases, as the number of initial hidden nodes increases, the difference between the two averaged curves becomes smaller and smaller and the variances, which measure the fluctuation of those 100 runs, also get closer and closer to zero. Having more hidden nodes at the beginning helps the NNs with more chances to reach a deep local optimum, and the NSNA procedure truly keeps those most important ones and drops the irrelevant hidden nodes, consistently leading to an improvement in neural network generalization at the end of training.

2.9 Idea on Selecting Features

In the previous section we saw that we can use the node selection with normalization and annealing to train better NNs than by random initialization when there are irrelevant variables. However, the irrelevant variables will still have a negative influence in the obtained model, and an even better model can be obtained by removing the irrelevant features. We can use two different kinds of measurements to determine whether a feature should be kept or dropped during.

The first choice we call **group criterion**, which compute the group weight (relevance) of each feature using the L_2-norm of the corresponding variables in the h weight vectors $\mathbf{w}_j$:

$$r_l = ||\boldsymbol{w}_{.,l}||_2 = \sqrt{\sum_{j=1}^{h} w_{jl}^2} \tag{2.11}$$

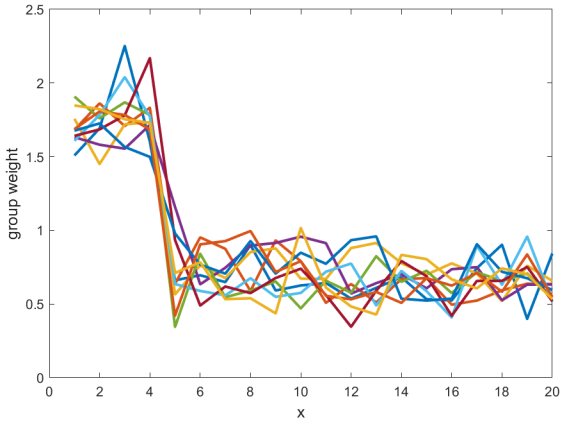


Figure 2.20: Feature weights r_i from (2.11) for the $p = 20$ features for 10 random 4D XOR datasets of size $n = 3000$.

An example for these group weights r_j for NNs trained on 10 different datasets with $k = 4, p = 20$ and $n = 3000$ is shown in Figure 2.20. We can clearly see in Figure 2.20 that the relevant variables 1 to 4 have larger group weights than the rest. Using this group criterion we can use Feature Selection with Annealing [Barbu et al., 2017] to select the relevant features for a NN.

The second choice we call **separate criterion**, which uses a similar procedure we do in determining the importance of a hidden node. In this measurement, we compute the magnitude of the weight for all connections between features and hidden nodes.

$$r_{j,l} = |w_{j,l}| \tag{2.12}$$

We will remove a feature for the network during the training procedure once all of the connections to the hidden nodes are dropped for that feature.

The selecting feature procedure is described in Algorithm 4.

Algorithm 4 Feature Selection with Annealing and NSNA for Neural Networks (FSA+NSNA)

Input: Training set $T = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}\}_{i=1}^n$, desired number k of features, annealing schedule $M_e, e = 1, .., N^{iter}$.

Output: Trained NN depending on exactly k features.

1: Train a NN using Algorithm 3.
2: **for** $e = 1$ to N^{iter} **do**
3: Train the NN for 1 epoch
4: Normalize the hidden nodes using $\tilde{\beta}_j \leftarrow \|\mathbf{w}_j\|\beta_j, \tilde{\mathbf{w}}_j \leftarrow \frac{\mathbf{w}_j}{\|\mathbf{w}_j\|}$
5: Compute the feature weights either using r_l or $r_{j,l}$
6: Keep the M_e features with largest r_l or $r_{j,l}$
7: **end for**

2.10 Experiments

In this section, we present experiments on XOR-related artificial datasets and real datasets to demonstrate the effectiveness of our proposed NN generalization capability improvement technique.

2.10.1 XOR Simulated Datasets

In this section, we present experiments on 3D, 4D, 5D XOR datasets with a 3000 observations training set and 3000 different testing samples. The total number p of features for each dataset will be selected such that the neural network almost lost all its generalization capability. For the 3D XOR dataset, we selected $p = 60$, for 4D XOR is $p = 40$ and 5D XOR is $p = 20$. Figure 2.8 shows that a one hidden layer NN without regularization can only obtain a test AUC close to 0.5 for these cases, which means it has nearly no generalization power.

We also train a one hidden layer neural network starting from a large number of hidden nodes, then gradually reduce the nodes to some target number, with the purpose of finding a deeper local optimum that can make the NN generalize better. We perform node selection in the hidden layer (NN-NSNA-hidden), and leave the input connections which connect the features to the hidden neurons untouched. After that we prune connections to the input layer using FSA. After searching

through a number combination of hyper-parameters, we obtained the best initial number of hidden nodes as 1024 for 3D XOR, 2048 for 4D XOR, and 4096 for 5D XOR, and the final number of hidden nodes as 8, 16 and 64 respectively. The obtained test AUC (Area Under the ROC curve) results using a default Adam optimizer and L2 penalty 0.001 are displayed in Table 2.1.

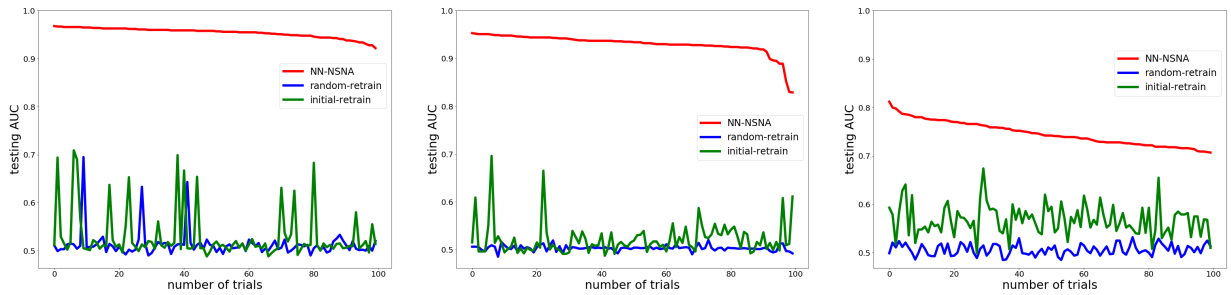
Table 2.1: Comparison between the standard NN, pruned sub-networks on the hidden layer and all layers, and retrained pruned networks with random weights and the initial weights from the beginning.

Test AUC(%)					
Dataset	NN best	NN+NSNA hidden	NN+FSA+NSNA all	NN+NSNA retrained random	NN+NSNA retrained initial wts
3D-XOR $p = 60$	82.53	**96.58**	**99.58**	70.47	71.68
4D-XOR $p = 40$	72.17	**95.31**	**98.31**	52.83	69.32
5D-XOR $p = 20$	66.24	**80.03**	**96.83**	53.21	66.29

In Table 2.1 are also displayed the best test AUC of a standard one hidden layer NN, the best result of a pruned sub-network that was retrained after the node selection stage using a random initialization, and the best result of a pruned sub-network that was retrained after the node feature selection stage using the initial weights used before pruning. Clearly, we see that our NSNA that preserved the trained weights helps the NN have a far better generalization than the others networks, and the sub-network does not seem to find a good local optimum when retrained.

We further examine whether or not we can recover the performance of the sub-network after the pruned sub-network was obtained, either via retraining from scratch or from the corresponding initial original weights. In each case, we repeat 100 times the process of pretraining a fully connected neural network, applying NSNA to prune the hidden nodes, and retraining the sub-network using random initialization and initial weights, using the best combination of hyper parameters described above. We sorted the runs by the test AUC and displayed the obtained resulting curves in Figure 2.21.

From Figure 2.21 we see that for these three cases on the XOR data, neither retraining from scratch nor using the original initial weights can achieve the performance of the corresponding sub-network that was obtained by pruning the fully connected dense neural network. The retraining using initial weights seems works a little better than retraining from a random initialization, as sometimes the sub-network retrained with initial weights can obtain a test AUC beyond 0.6, but still way behind the sub-network obtained form large network pruning.



$k = 3, p = 60$ $k = 4, p = 40$ $k = 5, p = 20$

Figure 2.21: Testing AUC vs number of trials.

Besides selecting the hidden nodes, we also select features to the input layer to remove irrelevant features, shown as NN+FSA+NSNA(all) in Table 2.1. This may further improve the performance for the XOR data. The obtained results are also displayed in Table 2.1, and almost perfectly match the best result we could obtain from training the same XOR data without irrelevant features.

Figure 2.22 displays a complete training and test loss obtained during training a NN with FSA+NSNA on the 4D XOR data with $p = 40$. Also shown is the AUC evolution, which indicates that a good sub-network was obtained in the end, performing well on both training and testing data. The fully connected neural network was first trained for 1200 iterations to reach a local optimum with a training AUC of 1.0 but a bad test AUC. After selecting the relevant hidden nodes, the training AUC went back to 1 and the test AUC improved considerably. Finally, after selecting the features to the input layer to remove irrelevant features, the test AUC obtains a comparable performance to the training AUC, which means this is a good deep local optimum. This figure is a good illustration of how the loss and AUC evolve during the training procedure. In most cases the number of training iterations can be reduced to have same final testing performance.

2.10.2 Parity Data

The parity data with noise is a classical problem in computational learning theory [Zhang et al., 2017]. It is a simpler case of the XOR data: it has the same labels as the XOR data but each variable in the feature vector $\mathbf{x}$ is uniformly drawn from two values $\{-1, +1\}$. Like in Zhang et al. [2017], we made the classification problem harder by randomly selecting 10% of the data points and switching their labels to the opposite values. Thus the best classifier would have a prediction

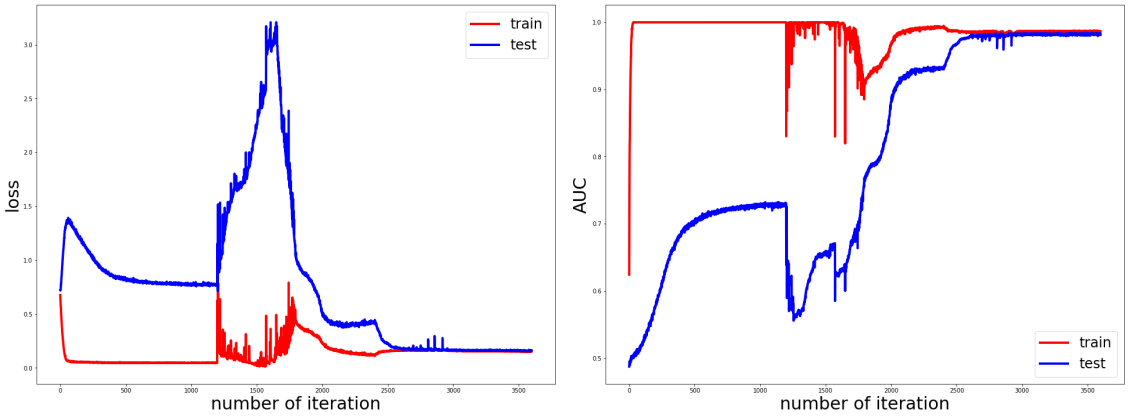


Figure 2.22: Loss and AUC evolution for training a pruned sub-network using NN+FSA+NSNA for $k = 4, p = 40$ XOR data.

error of 0.1.

$$y = \begin{cases} x_{i_1} x_{i_2} ... x_{i_k} & \text{with probablity 0.9} \\ -x_{i_1} x_{i_2} ... x_{i_k} & \text{with probablity 0.1} \end{cases}$$

This kind of dataset is frequently used to test different optimizers and regularization techniques on the NN model. We perform the experiment in $p = 50$ dimensional data with parities $k = 5$. The training set, valid set, and testing set contain respectively 15,000, 5,000 and 5,000 data points. We train a one hidden layer neural network with default stochastic gradient descent (SGD) optimizer, Adam [Kingma and Ba, 2014] optimizer and Adam + NSNA. For NN with Adam + NSNA, we start with 256 hidden nodes, and down to a hidden node number B in the range $B \in [1, 16]$ using annealing schedule M_e. We report the best result out of 10 independent random initializations. Recently, a neural network based boosting method named BoostNet [Zhang et al., 2017] significantly outperformed a normal NN on this data. As Zhang et al. [2017]'s experiment setting is very similar to us but with more training data, so we directly extract their results and report together with our experimental outcomes.. The comparison of the test errors is shown in Figure 2.23.

We can see that the ANN with the SGD optimizer cannot learn any good model with less than 100 hidden nodes on this data, while an ANN with the Adam optimizer can learn some pattern when the number of hidden nodes is greater than 25, but still mostly cases are trapped in shallow local optima. The BoostNet can learn well if the hidden node number is greater than about 45 hidden nodes. The best performance is achieved by ANN with Adam + NSNA, with 256 starting hidden nodes. After applying NSNA during the ANN training, we only needed to keep as few as 6 hidden

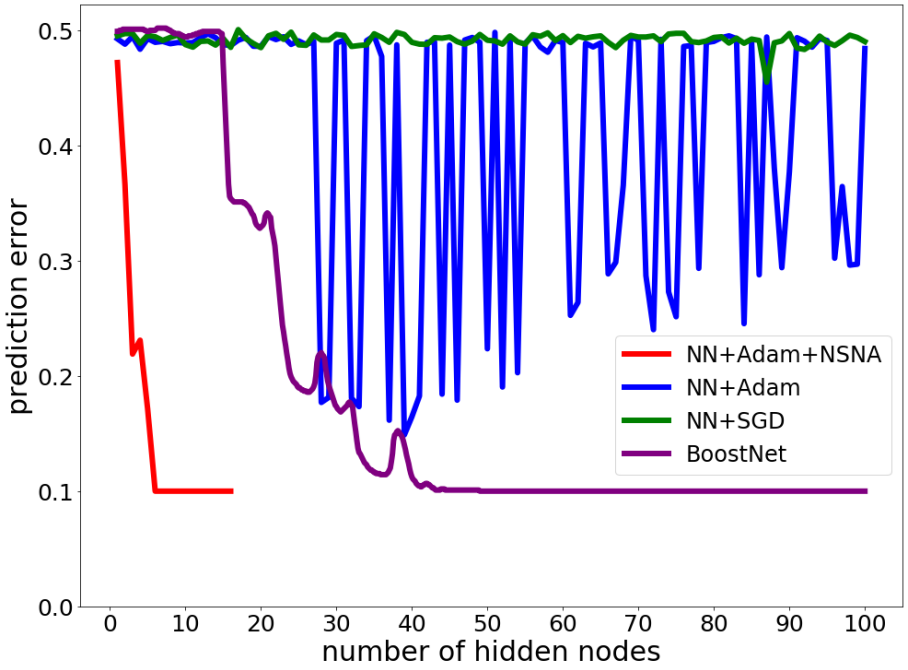


Figure 2.23: Test error vs number of hidden nodes. Comparison between single hidden layer neural networks trained by NN + Adam + NSNA starting with 256 hidden nodes, NN + Adam, NN + SGD and BoostNet.

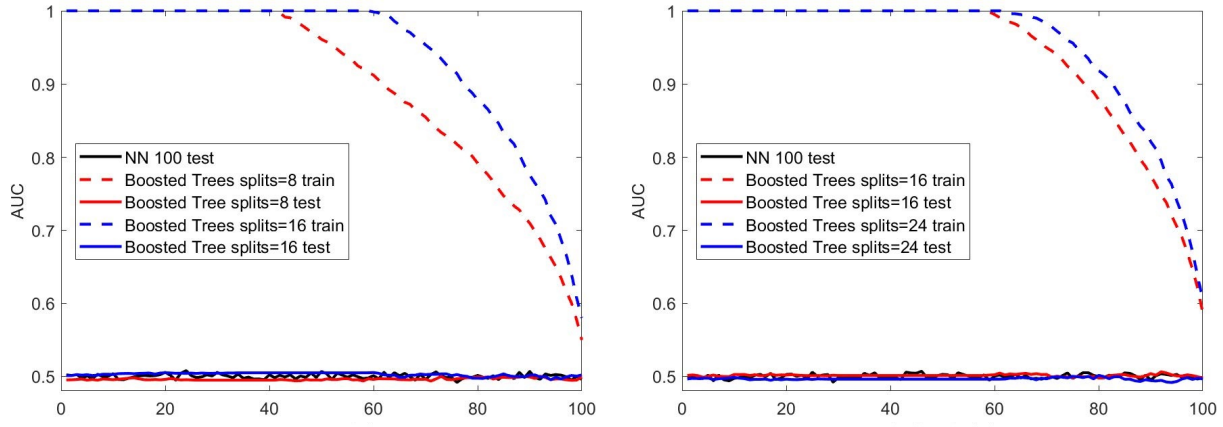
nodes to get the best possible prediction error. This observation implies The NSNA algorithm has a good capability to find a global or deep enough local optimum by gradually removing unimportant nodes.

2.10.3 Madelon Data

The Madelon dataset, featured in the NIPS 2003 feature selection challenge [Guyon et al., 2004], is a generalization of classical XOR dataset to five dimensions. Each vertex of a five dimensional hypercube contains a cluster of data points randomly labeled as +1 or -1. The five dimensions constitute 5 informative features and 15 linear combinations of those features were added to form a set of 20 redundant but informative features. Additionally, 480 distractor features with no predictive power were added at random locations. So the total number of variables of the Madelon data is 500.

Screening using Boosted Trees. When the number of irrelevant features is very large, even the FSA Algorithm 4 described above has a hard time finding the true features. In this case we

can employ boosted trees to generate a number of candidate feature sets. The boosted trees have a different behavior on the XOR data. To see that, we trained 100 boosted tree models with between 1 and 100 boosting iterations and sorted them in decreasing order of their training AUC.



$n = 3000, p = 100, k = 3$ $n = 3000, p = 100, k = 4$

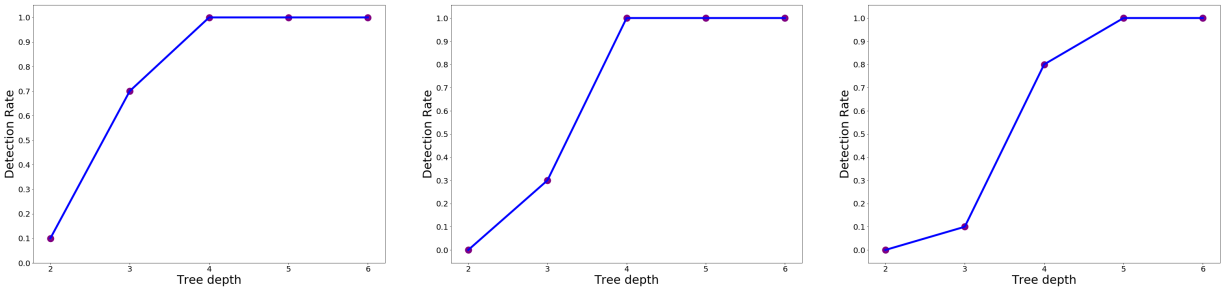
Figure 2.24: Train and test AUC of boosted trees with 1-100 boosted iterations, sorted by decreasing train AUC.

In Figure 2.24 are shown the training and test AUC of the 100 models for 3D and 4D XOR with $n = 3000, p = 100$, averaged over 10 independent datasets. We see that the boosted trees severely overfit the training data, and cannot learn any model that would generalize to the test data.

However, we observed that they are still capable of sometimes finding the correct features. We again trained boosted trees with maximum d depth on 80% of the training data sampled randomly. We then looked at each tree to see whether all k true features were among the total features used by the tree. If they were, we consider the features were found by the boosted tree.

In Figure 2.25 are shown the percent times all features were found vs tree depth out of 10 independently generated datasets. For each dataset we used 100 trials with 30 boosting iterations each, with maximum depth $d = \{2, 3, 4, 5, 6\}$. We see that if the maximum tree depth is deep enough (4 for $k = 4$ and 5 for $k = 5$) at least one of the 3000 feature sets contains all k true features.

Using this screening procedure we introduce our proposed algorithm for training NNs with feature selection. The procedure is described in Algorithm 5.



$k = 3, p = 100$ $k = 4, p = 100$ $k = 5, p = 100$

Figure 2.25: Variable detection rate. Percent of the runs all k features were found together in one of the 3000 subsets generated by FSBT with 30 boosting iterations.

Algorithm 5 Feature Selection using Boosted Trees (FSBT)

Input: Training set $T = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}\}_{i=1}^n$, number N^{trial} of trials, number B of boosting iterations, max tree depth d, number k of selected features

Output: Trained NN on k selected features

1: **for** j = 1 to N^{trial} **do**
2: Subsample 80% of the training examples without replacement.
3: Train a boosted tree with B boosted iterations and maximum d depth for each tree
4: **for** i = 1 to B **do**
5: Set S_j^i as the set of features used in the i-th boosted tree.
6: Train a NN model m_{ij} on feature set S_j^i, obtaining training AUC a_{ij}.
7: **end for**
8: **end for**
9: Output the model m_{ij} with largest training AUC a_{ij}

Experiment Results. This is a high-dimensional, non-linear, sparse dataset with only 2000 training points. The test set predictions cannot be evaluated online anymore so we used the validation set (containing 600 data points) as the test set for our experiment. We compared our method with the boosted trees, random forest, a regular single hidden layer NN and a tree-rule induced NN called neural rule ensembles (NRE) [Dawer et al., 2020]. For boosted trees, we used the Xgboost package [Chen and Guestrin, 2016], searched for the best test error for the number of boosting iterations $i \in \{10, 20, 30\}$, maximum tree depth $d \in \{2, 4, 6, 8, 10\}$. For the random forest, we searched for the best test error for number of trees $n \in \{100, 200, 400\}$, maximum tree depth $d \in \{2, 4, 6, 8, 10\}$, maximum splitting features $m \in \{\sqrt{p}, 0.2p, 0.4p, 0.6p, 0.8p\}$. For the one

single hidden layer NN, we executed 10 independent random initializations and searched for the best result in number of hidden nodes $h \in \{32, 64, 128, 512, 1048, 2048\}$. For the NRE, we directly extract the best result from the original paper [Dawer et al., 2020]. For our method, we first used FSBT with $N^{trial} = 10$ trials, $B = 20$ boosting iterations and maximum tree depth $d \in \{4, 5\}$ to obtain a list of 400 reduced feature sets. Then for each feature set we applied 10 independent random initializations of FSA+NSNA starting with $H = 2048$ hidden nodes, and pruned down to $h \in \{64, 128\}$ hidden nodes and selected $k \in \{5, 10, 15, 20\}$ features. Like the other methods, we reported the best result out of these $2 \times 4 = 8$ parameter combinations.

Table 2.2: Comparison of the test error of Xgboost, NN, RF, NRE and FSBT+FSA+NSNA on the Madelon dataset.

Method	Xgboost	NN	RF	NRE	FSBT+FSA+NSNA
Test error %	15.50	40.13	12.19	10.30	**7.83**

The results are shown in Table 2.2. Due to the high-dimensional character of the Madelon data, the learnability of a normal NN is poor, with the worst test performance among the four. Xgboost and RF can handle this data to some extent, achieving a test error of 15.50% and 12.19% respectively. The NRE, which uses a pre-trained decision tree to construct a one hidden layer neural network, can get a test error of 10.30%. Our FSBT+FSA+NSNA algorithm works best, outperforming the other four by a clear margin, learns a model with test error of 7.83%.

2.10.4 Real Datasets

In this section, we perform an evaluation on a number of real multi-class datasets to compare the performance of a fully connected NN and the compact NN obtained by FSA+NSNA. The real datasets were carefully selected from the UCI ML repository [Dua and Graff, 2017] to ensure that the dataset is not too large (the number of data points less than 10000) and that a standard fully connected neural network (with one hidden layer) can have a reasonable generalization power on this data. If a dataset is large, then the loss landscape is simple and the neural network can be trained easily, so there is no need for pruning to escape bad optima. If a dataset is such that a neural network can rarely be trained on it successfully, it means that the loss might not have any good local optima, then again pruning might not make sense. The details about the datasets are displayed in Table 2.3.

Table 2.3: Datasets used for evaluating the performance of fully connected NN and sparse NN with FSA+NSNA.

Dataset	Number of classes	Number of features	Number of observations
Car Evaluation	4	21	1728
Image Segmentation	7	19	2310
Optical Recognition of Handwritten Digits	10	64	5620
Multiple Features	10	216	2000
ISOLET	26	617	7797

Our real dataset experiments are not aimed at comparing the performance with other classification techniques, but to test the effectiveness of FSA+NSNA in guiding neural networks to find better local optima. For this reason we will combine all the samples including training, validation and testing data to form a single dataset for each data type first, and then divide them into a training and testing set with a ratio $4:1$. The obtained training dataset will be used in a 10-run averaged 5-fold cross-validation grid search training process to find the best hyper-parameter settings of a one hidden layer fully connected neural network. After getting the best hyper-parameter setting from the cross-validation, we use them to retrain the fully connected NNs with the entire training dataset 10 different times, and each time we record the best test accuracy. This procedure is used for the fully connected NN, and the NN with FSA+NSNA with different sparsity levels and record the best sparsity level and testing accuracy. Finally, we will also train a so-called "equivalent" fully connected neural network with roughly the same number of connections as the best sparse neural network we get from FSA+NSNA.

The number of hidden nodes was searched in $\{16, 32, 64, 128, 256, 512\}$, the L2 regularization coefficient was searched in $\{0.0001, 0.001, 0.01, 0.1\}$, the batch size was searched in $\{16, 32, 64\}$. Other NN training techniques like Dropout [Srivastava et al., 2014] and Batch Normalization [Ioffe and Szegedy, 2015] were not used in our experiments due to the simplicity of the architecture of experimented NNs. The comparison results are listed in Table 2.4. The sorted loss values and test accuracys of the models with 200 random initializations are shown in Figure 2.26 and 2.27.

We see from Table 2.4 that using NN+FSA+NSNA to guide the search for a local optimum leads to NNs with good generalization on all these datasets, easily outperforming a NN of an equivalent size (with a similar number of weights) and in most cases even the standard NN with the best generalization to unseen data. We see from Fig.2.26 and Fig.2.27 that the NN+FSA+NSNA can obtain lower loss values and higher test accuracy than the other networks in nearly all cases.

Table 2.4: Performance results of NN(best), NN(equivalent) and NN+FSA+NSNA for each dataset.

	NN(best)	NN(equivalent)	NN+FSA+NSNA
Car Evaluation, $p = 21, n = 1728$, 4 classes.			
Number of weights (nodes)	1600 (64)	150 (6)	120+32 = 152
Test Accuracy	**100.0±0.00**	98.23±0.06	**100.0±0.00**
Image Segmentation, $p = 19, n = 2310$, 7 classes.			
Number of weights (nodes)	6656 (256)	364 (14)	266+98 = 364
Test Accuracy	96.87±0.72	96.27±0.58	**98.40±0.32**
Optical Recognition of Handwritten Digits, $p = 64, n = 5620$, 10 classes.			
Number of weights (nodes)	37888 (512)	1998 (27)	1792+160 = 1952
Test Accuracy	98.80±0.29	98.25±0.19	**99.01±0.20**
Multiple Features, $p = 216, n = 2000$, 10 classes.			
Number of weights (nodes)	14464 (64)	904 (4)	583+320 = 903
Test Accuracy	97.85±0.80	95.45±0.98	**98.15±0.82**
ISOLET, $p = 617, n = 7797$, 26 classes.			
Number of weights (nodes)	41152 (64)	5787 (9)	4683+1118 = 5801
Test Accuracy	96.73±0.50	94.31±0.61	**96.91±0.54**

The experiments show that the XOR data is indeed an extreme example where deep local optima are be hard to find, but even the real datasets exhibit some non-equivalent local optima and the things we learned from the XOR data carry over to these datasets to help us train NNs with better generalization.

2.11 Conclusion

In this chapter, we presented an empirical study of the trainability of neural networks and the connection between the amount of training data and the loss landscape. We observed that when the training data is large (where "large" depends on the problem), the loss landscape is simple and easy to train. When the training data is limited, the number of local optima can become very large, making the optimization problem very difficult. For these cases we introduce a method for training a neural network that avoids many local optima by starting with a large model with many hidden neurons and gradually removing neurons to obtain a compact network trained in a deep minimum. Moreover, the performance of the obtained pruned sub-network is hard to achieve by retraining using random initialization, due to the existence of many shallow local optima around

the deep minimum. Experiments also show that our node and feature selection method is useful in improving generalization on the XOR data and on a number of real datasets.

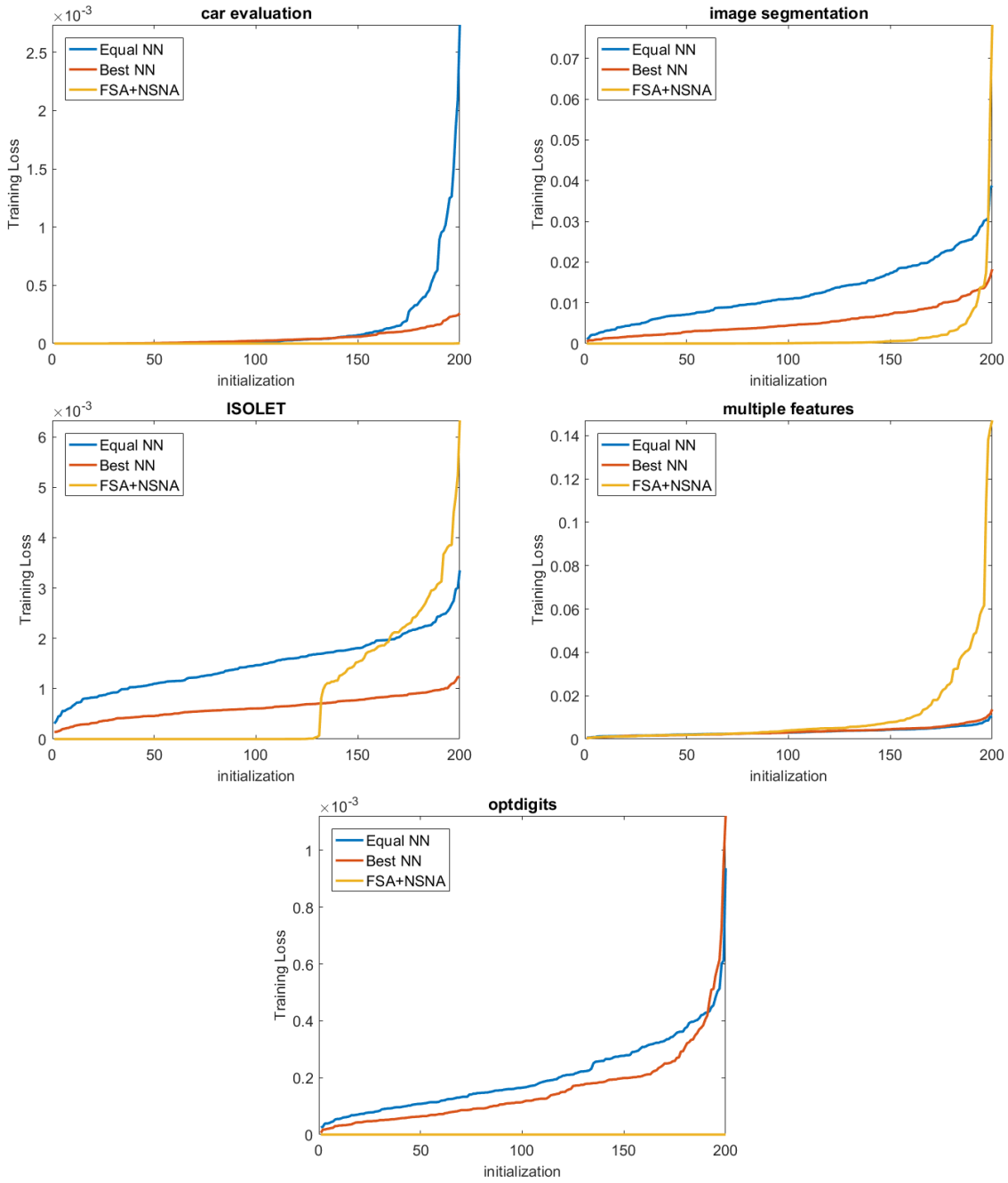


Figure 2.26: Sorted loss values for 200 initializations obtained on the five real datasets.

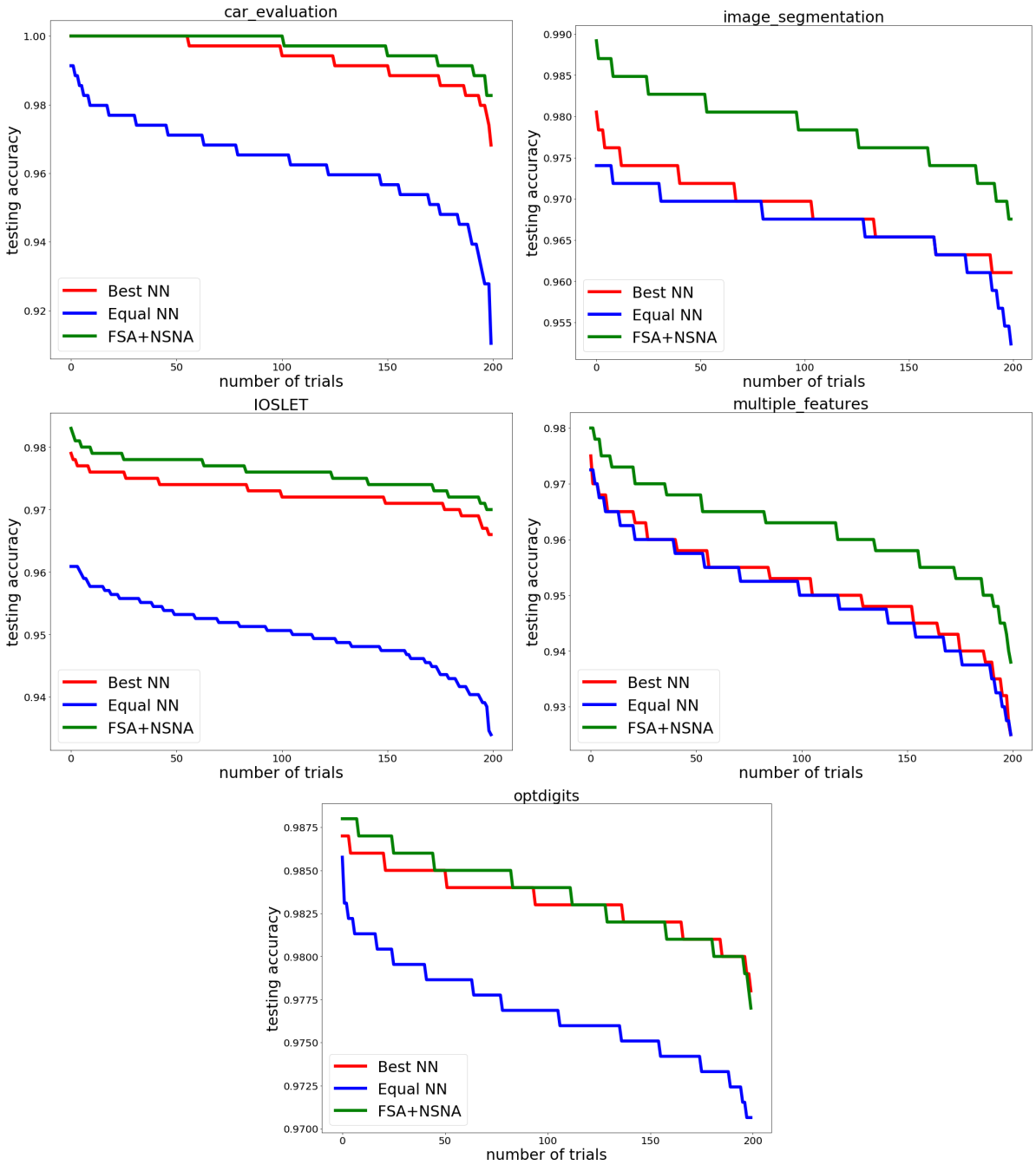


Figure 2.27: Sorted test accuracy values for 200 initializations obtained on the five real datasets.

CHAPTER 3

A NETWORK PRUNING FRAMEWORK FOR DEEP CONVOLUTIONAL NEURAL NETWORKS VIA ANNEALING AND DIRECT SPARSITY CONTROL

Deep convolutional neural networks (DCNNs) have proved to successfully offer reliable solutions to many vision problems. However, the use of DCNNs is widely impeded by their intensive computational and memory cost. In this chapter, we propose a novel efficient network pruning framework that is suitable for both non-structured and structured channel-level pruning. Our proposed method tightens a sparsity constraint by gradually removing network parameters or filter channels based on a criterion and a schedule. The attractive fact that the network size keeps dropping throughout the iterations makes it suitable for the pruning of any untrained or pre-trained network. Because our method uses a L_0 constraint instead of the L_1 penalty, it does not introduce any bias in the training parameters or filter channels. Furthermore, the L_0 constraint makes it easy to directly specify the desired sparsity level during the network pruning process. Finally, experimental validation on real datasets show that the proposed framework with a certain pruning criterion and schedule obtains better or competitive performance compared to other states of art network pruning methods.

3.1 Introduction

In recent years, artificial neural networks (ANNs) especially deep convolutional neural networks (DCNNs) are widely applied and have become the dominant approach in many computer vision tasks. These tasks include image classification [Krizhevsky et al., 2012, Simonyan and Zisserman, 2014, He et al., 2016, Huang et al., 2017], object detection [Girshick et al., 2014, Ren et al., 2015], semantic segmentation Long et al. [2015], 3D reconstruction [Dou et al., 2017], etc. The quick development in the deep learning field leads to network architectures that can go nowadays as deep as 100 layers and contain millions or even billions of parameters. Along with that, more and more computation resources must be utilized to successfully train such a deep modern neural network.

The deployment of DCNNs in real applications is largely impeded by their intensive computational and memory cost. With this observation, the study of network pruning methods that learn a smaller sub-network from a large original network without losing much accuracy has attracted a lot of attention. Network pruning algorithms can be divided into two groups: non-structured pruning and structured pruning. The earliest work for non-structured pruning is conducted by LeCun et al. [1990], the most recent work is done by Han et al. [2015a,b]. The non-structured pruning aims at directly pruning parameters regardless of the consistent structure for each network layer. This renders modern GPU acceleration technique unable to obtain computational benefits from the irregular sparse distribution of parameters in the network, only specialized software or hardware accelerators can gain memory and time savings. The advantage of non-structured pruning is that it can obtain high network sparsity and at the same time preserve the network performance as much as possible. On the other side, structured pruning aims at directly removing entire convolutional filers or filter channels. Li et al. [2016] determines the importance of a convolutional filter by measuring the sum of its absolute weights. Liu et al. [2017] introduces a L_1-norm constraint in the Natch Normalization layer to remove filter channels associated with smaller γ. Although structured pruning cannot obtain the same level of sparsity as non-structured pruning, it is more friendly to modern GPU acceleration techniques and independent of any specialized software or hardware accelerators.

Unfortunately, many of the existing non-structured and structured pruning techniques are conducted in a layer-wise way, requiring a sophisticated procedure for determining the hyperparameters of each layer in order to obtain a desired number of weights or filters/channels in the end. This kind of pruning manner is not effective nor efficient.

We combine regularization techniques with sequential algorithm design and direct sparsity level control to bring forward a novel network pruning scheme that could be suitable for either non-structured pruning or structured pruning (particular for filter channel-wise pruning of DCNNs with Batch Normalization layers). We investigate a parameter estimation optimization problem with a L_0-norm constraint in the parameter space, together with the use of annealing to lessen the greediness of the pruning process and a general metric to rank the importance of the weights or filter channels. An attractive property is that parameters or filter channels are removed while the model is updated at each iteration, which makes the problem size decrease during the iteration process.

Experiments on extensive real vision data, including the MNIST, CIFAR, and SVHN provide empirical evidence that the proposed network pruning scheme obtains a performance comparable to or better than other state of art pruning methods.

3.2 Related Work

Network pruning is a very active research area nowadays, it provides a powerful tool to accelerate the network inference by having a much smaller sub-network without too much loss in accuracy. The earliest work about network pruning can be dated back to 1990s, when LeCun et al. [1990] and Hassibi and Stork [1993] proposed a weight pruning method that uses the Hessian matrix of the loss function to determine the unimportant weights. Recently, Han et al. [2015b] used a quality parameter multiplied by the standard deviation of a layer's weights to determine the pruning threshold. A weight in a layer will be pruned if its absolute value is below that threshold. Guo et al. [2016] proposed a pruning method that can properly incorporate connection slicing into the pruning process to avoid incorrect pruning. These pruning schemes mentioned above are all non-structured pruning, needing specialized hardware or software to gain computation and time savings.

For structured pruning, there are also quite a few works in the literature. Li et al. [2016] determined the importance of a convolutional filter by measuring the sum of its absolute weights. Hu et al. [2016] computed the average percentage of zero activations after the ReLu function and determine to prune the corresponding filter if its this percentage score is high. He et al. [2017] proposed an iterative two-step channel pruning method by a LASSO regression based channel selection and least square reconstruction. Liu et al. [2017] introduced a L_1-norm constraint in the Batch Normalization layer to remove filter channels associated with smaller $|\gamma|$. Zhou et al. [2018] imposed an extra cluster loss term in the loss function that forces filters in each cluster to be similar and only keep one filter in each cluster after training. Yu et al. [2018] utilized a greedy algorithm to perform channel selection in a layer-wise way by constructing a specific optimization problem.

3.3 Network Pruning via Annealing and Direct Sparsity Control

Given a set of training examples $\mathcal{D} = \{(\mathbf{x_i}, y_i), i = 1, ..., N\}$ where $\mathbf{x}$ is an input and y is a corresponding target output, with a differentiable loss function $L(\cdot)$ we can formulate the pruning

problem for a neural network with parameters $\mathcal{W} = \{(\mathbf{W_j}, \mathbf{b_j}), j = 1, ..., L\}$ as following **constrained** problem

$$\min_{\mathcal{W}} \quad L(\mathcal{W}) \qquad \text{s.t.} \quad ||\mathcal{W}||_0 \leq K \tag{3.1}$$

where the L_0 norm bounds the number of non-zero parameters in $\mathcal{W}$ to be less than or equal to a specific positive integer K.

For non-structured pruning, we directly address the pruning problem in the whole $\mathcal{W}$ space. The final $\mathcal{W}$ will have an irregular distribution pattern of the zero-value parameters across all layers.

For structured pruning, suppose the DCNN is with convolutional filters or channels $\mathcal{C} = \{C_j, j = 1, ..., M\}$, we can replace the constrained problem (3.1) by

$$\min_{\mathcal{W}} \quad L(\mathcal{W}) \qquad \text{s.t.} \quad ||\mathcal{C}||_0 \leq K \tag{3.2}$$

By solving the problem (3.2), we will obtain the $\mathcal{W}$ on the convolutional layers having more uniform zero-value parameter distribution, specialized in some filters or filter channels.

These constrained optimization problems (3.1) and (3.2) facilitate parameter tuning because our sparsity parameter K is much more intuitive and easier to specify in comparison to penalty parameters such as λ in $\lambda||\mathcal{W}||_1$ and $\lambda||\mathcal{C}||_1$.

In this work, we will focus on the study of the weight-level pruning (non-structured pruning) for all neural networks and channel-level pruning (structured pruning) particularly for neural networks with Batch Normalization layers.

3.3.1 Basic Algorithm Description

Some key ideas in our algorithm design are: a) We conduct our pruning procedures in the specified parameter spaces; b) We use an annealing plan to directly control the sparsity level in each parameter space; c) We gradually remove the most "unimportant" parameters or channels to facilitate computation. The prototype algorithms, summarized in Algorithm 6 and 7, show our ideas. It starts with either an untrained or pre-trained model and alternates two basic steps: one step of parameter updates towards minimizing the loss $L(\cdot)$ by gradient descent and one step that removes some parameters or channels according to a ranking metric $\mathcal{R}$.

The intuition behind our DSC algorithms is that during the pruning process, each time we remove a certain number of the most unimportant parameters/channels in each parameter/channel

Algorithm 6 Network Pruning via Direct Sparsity Control - Weight-Level (DSC-1)

Input: Training set $T = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$, desired parameter space $\{\mathcal{W}_j | \cup \mathcal{W}_j = \mathcal{W}\ \&\ \cap \mathcal{W}_j = \varnothing\}_{j=1}^{B}$, desired number $\{K_j\}_{j=1}^{B}$ of parameters, desired annealing schedule $\{M_j^e, e = 1, .., N^{iter}\}_{j=1}^{B}$, an ANN model.

Output: Pruned ANN depending on exactly $\{K_j\}_{j=1}^{B}$ parameters in each parameter space $\{\mathcal{W}_j\}_{j=1}^{B}$.

1: If the ANN is not pre-trained, train it to a satisfying level.
2: **for** $e = 1$ to N^{iter} **do**
3: Sequentially update $\mathcal{W} \leftarrow \mathcal{W} - \eta \frac{\partial L(\mathcal{W})}{\partial \mathcal{W}}$ via backpropagation.
4: **for** $j = 1$ to B **do**
5: Keep the M_j^e most important parameters in $\mathcal{W}_j$ based on ranking metric $\mathcal{R}$.
6: **end for**
7: **end for**
8: Fine-tune the pruned ANN with exactly $\{K_j\}_{j=1}^{B}$ parameters in each parameter space $\{\mathcal{W}_j\}_{j=1}^{B}$.

space based on an annealing schedule. This ensures that we do not inject too much noise in the parameter/channel dropping step so that the pruning procedure can be conducted smoothly. Our method directly controls the sparsity level obtained at each parameter/channel space, unlike many layer-wise pruning methods where a sophisticated procedure has to be used to control how many parameters are kept, because pruning the weights or channels in all layers simultaneously can be very time-consuming.

Algorithm 7 Network Pruning via Direct Sparsity Control - Channel-Level (DSC-2)

Input: Training set $T = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$, desired channel space $\{\mathcal{C}_j | \cup \mathcal{C}_j = \mathcal{C}\ \&\ \cap \mathcal{C}_j = \varnothing\}_{j=1}^{B}$, desired number $\{K_j\}_{j=1}^{B}$ of channels, desired annealing schedule $\{M_j^e, e = 1, .., N^{iter}\}_{j=1}^{B}$, a DCNN model.

Output: Pruned DCNN depending on exactly $\{K_j\}_{j=1}^{B}$ channels in each channel space $\{\mathcal{C}_j\}_{j=1}^{B}$.

1: If the DCNN is not pre-trained, train it to a satisfying level.
2: **for** $e = 1$ to N^{iter} **do**
3: Sequentially update $\mathcal{W} \leftarrow \mathcal{W} - \eta \frac{\partial L(\mathcal{W})}{\partial \mathcal{W}}$ via backpropagation
4: **for** $j = 1$ to B **do**
5: Keep the M_j^e most important channels in $\mathcal{C}_j$ based on ranking metric $\mathcal{R}$.
6: **end for**
7: **end for**
8: Fine-tune the pruned DCNN with exactly $\{K_j\}_{j=1}^{B}$ channels in each parameter space $\{\mathcal{C}_j\}_{j=1}^{B}$.

Through the annealing schedule, the support set of the network parameters or channels is gradually shrunken until we reach $||\mathcal{W}||_0 \leq K$ or $||\mathcal{C}||_0 \leq K$. The keep-or-kill rule is based on the ranking metric $\mathcal{R}$ and does not involve any information of the objective function L. This is in contrast to many ad-hoc networking pruning approaches that have to modify the loss function and can not easily be scaled up to many existing pre-trained models.

3.3.2 Implementation Details

In this part, we provide implementation details of our proposed **DSC** algorithms.

First, the annealing schedule M_e is determined empirically. Our experimental experience shows that the following annealing plans can perform well to balance the efficiency and accuracy:

$$M_e = \begin{cases} (1-p_0) + p_0(\frac{N_1 - e}{\mu e + N_1}))M, & 1 \leq e < N_1 \\ (1 - \min(p, p_0 + \left\lfloor \frac{e - N_1}{N^c} \right\rfloor \nu))M, & N_1 < e \leq N^{iter} \end{cases}$$

Here M is the total number of parameters or channels in the neural network. Our M_e consists two parts. The first part can be used to quickly prune the unimportant parameters with a reasonable value of μ down to a percentage p_0 of the parameters. The second part can further refine our pruned sub-network to a more compact model. μ is the pruning rate and we will set it to $\mu = 10$ for all experiments. $p_0 \in [0, 1]$ denotes the percentage of parameters or channels to be pruned in the first part. $p \in [0, 1]$ denotes the final pruning percentage goal at the end of the pruning procedure, thus the number of remaining parameters is $K = M(1 - p)$. The parameter N^c specifies how many epochs to train before performing another pruning. We will select $N^c \in \{1, 2\}$. ν denotes the incremental pruning percentage as the annealing continues and will be set to $\nu \in \{0.005, 0.01, 0.02\}$. An example of an annealing schedule M_e (with $M = 1$ for clarity) with $N_1 = 10, N^{iter} = 20, N^c = 1$, $p_0 = 0.8, p = 0.9$, and $\nu = 0.02$ is shown in Figure 3.1.

Second, as the convolutional layers and fully connected layers have very different behavior in a DCNN, we will prune them separately during the structured and non-structured pruning process, i.e. we will fix the convolutional layer parameters while pruning the fully connected layer, and vice versa.

Third, the ranking metric $\mathcal{R}$ we select for structured and non-structured pruning is different. For non-structured pruning, the parameter dropping procedure based on the magnitude of the

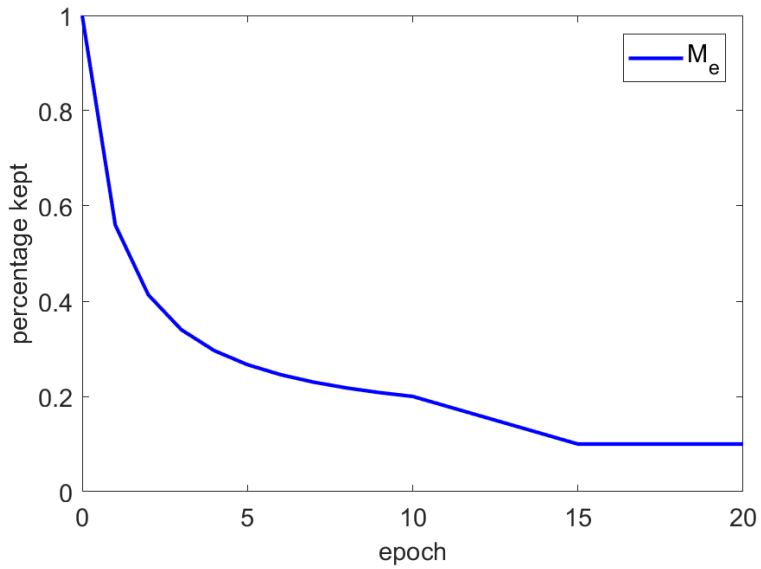


Figure 3.1: Annealing schedule with $N_1 = 10, N^{iter} = 20, N^c = 1$, $p_0 = 0.8, p = 0.9, \nu = 0.02$.

parameter yields quite good pruning results in our experiments. Therefore we will select it as our metric to rank the importance of parameters for all our non-structured pruning experiments:

$$\mathcal{R}(w) = |w|, w \in \mathcal{W} \tag{3.3}$$

For structured channel pruning, various dropping criteria are proposed. One family of channel pruning metrics are based on the value of the channel weights. Li et al. [2019] uses the L_1-norm by summing up the magnitude of all channel weights to rank the importance of the metric in a channel space; Wen et al. [2016] suggests the use of the L_2-norm. Another family of channel pruning metrics [Liu et al., 2017] lies in the absolute value of the Batch Normalization scales, as Batch Normalization [Ioffe and Szegedy, 2015] has been widely adopted by most modern DCNNs to accelerate the training speed and convergence. Assume z_{in} and z_{out} to be the input and output of a Batch Normalization layer, we can formulate the transformation of that BN layer performs as:

$$BN(z_{in}) = \frac{z_{in} - \mu_{\mathbf{B}}}{\sqrt{\sigma^2_{\mathbf{B}} + \epsilon}}; z_{out} = \gamma \cdot BN(z_{in}) + \beta$$

where $\mathbf{B}$ denotes the mini-batch statistic of input activations, $\mu_{\mathbf{B}}$ and $\sigma_{\mathbf{B}}$ are the mean and standard deviation over $\mathbf{B}$, γ and β are trainable scale and shift parameters of the affine transformation. Liu et al. [2017] directly leverages the parameters γ in the Batch Normalization layers as the scaling

factors they need for channel pruning. They impose a L_1 norm on each Batch Normalization layer for the γ to reformulate the training loss function. Here we combine the metrics of these two families to enjoy wider flexibility on the DCNNs and define our ranking metrics as follows:

$$\begin{aligned} \mathcal{R}_{\mathbf{B}}(\mathcal{C}) &= |\gamma_{\mathcal{C}}| \\ \mathcal{R}_{\mathbf{L}}(\mathcal{C}) &= (||\mathcal{C}||_{L_1} + ||\mathcal{C}||_{L_2})/2 \\ \mathcal{R}(\mathcal{C}) &= \alpha \cdot \frac{\mathcal{R}_{\mathbf{B}}(\mathcal{C})}{\mathcal{R}_{\mathbf{B}}^{max}(\mathcal{C})} + (1-\alpha) \cdot \frac{\mathcal{R}_{\mathbf{L}}(\mathcal{C})}{\mathcal{R}_{\mathbf{L}}^{max}(\mathcal{C})} \end{aligned} \tag{3.4}$$

where $\gamma_{\mathcal{C}}$ is the scale parameter of the BN for channel $\mathcal{C}$, $\alpha \in [0, 1]$ is a hyper-parameter that needs to be specified to balance the two ranking terms $\mathcal{R}_{\mathbf{B}}$ and $\mathcal{R}_{\mathbf{L}}$. The main differences from the other pruning methods are that we do not make any modifications to the loss function, but utilize a L_0 norm constraint and we use an annealing schedule to gradually eliminate channels and lessen the greediness.

Fourth, after the pruning process, we will conduct a fine-tuning procedure to gain back the performance lost during the pruning period. Before we start the fine-tuning, we can remove for non-structured pruning the neurons that have zero incoming or outgoing degree and mask the convolution filter channels with all zero parameters by adding a channel selection layer [Liu et al., 2017] to form a more compact network for later inference use.

3.4 Experiments

In this section, we conduct non-structured pruning with Lenet-300-100 and LeNet-5 [LeCun et al., 1998] on MNIST [LeCun and Cortes, 2010] dataset and conduct our structured channel pruning experiments with VGG-16 [Simonyan and Zisserman, 2014] and DenseNet-40 [Huang et al., 2017] on the CIFAR [Krizhevsky et al., 2009a] and SVHN [Netzer et al., 2011] datasets.

3.4.1 Non-structured Pruning on MNIST

The MINIST dataset provided by LeCun and Cortes [2010] is a handwritten digits dataset that is widely used in evaluating machine learning algorithms. It contains 50K training observations, 10K validation and 10K testing observations respectively. Some digit visualization examples are displayed in Figure 3.2. In this section, we will test our non-structured pruning method **DSC-1** on two network models: LeNet-300-100 and LeNet-5.

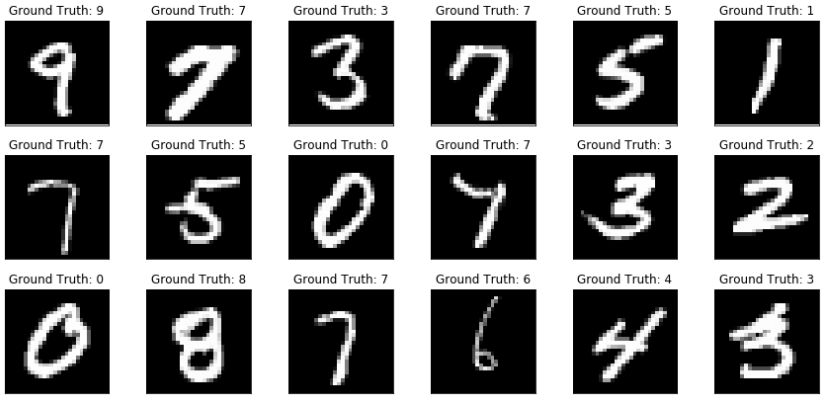


Figure 3.2: Examples of handwritten digits in the MNIST dataset.

LeNet-300-100 [LeCun et al., 1998] is a classical fully connected neural network with two hidden layers. The first hidden layer has 300 neurons and the second has 100. The LeNet-5 is a conventional convolutional neural network that has two convolution layers and two fully connected layers. LeNet-300-100 consists of 267K learnable parameters and LeNet-5 consists of 431K.

To have a fair comparison with Han et al. [2015b], we follow the same experimental setting by using the default SGD method, training batch size and initial learning rate to train the two models from scratch. After a model with similar performance was obtained, we stop the training and directly apply our **DSC-1** pruning algorithm to compress the model. During the pruning and retraining procedure, a learning rate with 1/10 of the original network's learning rate is adopted. A momentum with value of 0.9 is used to speed up the model retraining.

Table 3.1: Non-structured pruning comparison on LeNet-300-100 and LeNet-5.

Model	Error	Params	Prune Rate
Lenet-300-100 (Baseline)	1.64%	267k	-
Lenet-300-100 ([Han et al., 2015b])	1.59%	22K	91.8%
Lenet-300-100 (Ours)	**1.57%**	**17.4K**	**93.5%**
Lenet-5 (Baseline)	0.8%	431K	-
Lenet-5 ([Han et al., 2015b])	0.77%	36k	91.6%
Lenet-5 (Ours)	0.77%	**15.8k**	**96.4%**

In LeNet-300-100, a total of 20 epochs were used for both pruning and fine-tuning. For the annealing schedule, p_0 is directly set to 0.85 without using any annealing schedule. Then we

Table 3.2: Layer by layer compression comparisons on LeNet-300-100 and LeNet-5.

Model	Layer	Params.	Han %	Ours %
Lenet-300-100	fc1	236K	8%	4.6%
	fc2	30K	9%	20.1%
	fc3	1K	26%	68.5%
	Total	267K	8.2%	6.5%
Lenet-5	conv1	0.5K	66%	75%
	conv2	25K	12%	29.1%
	fc1	400K	8%	1.8%
	fc2	5K	19%	17.2%
	Total	431K	8.4%	3.6%

follow the fine-grain pruning annealing schedule which $N^c = 1$ and $\nu = 0.05$ to reach at the final percentage goal $p = 0.935$. The remaining epochs are used for fine-tuning purposes.

In LeNet-5, the pruning for fully connected layers and convolutional layers are treated separately. For pruning on fully connected layers, we directly set at $p_0 = 0.9$ and then reach $p = 0.98$ with $N^c = 1, \nu = 0.05$. For the convolutional layers we start with $p_0 = 0$, $N^c = 1$ and $\nu = 0.05$ to reach at $p = 0.7$. The total number of pruning and retraining epochs for LeNet-5 is 40 epochs. After several experimental trials, we output our best result in Table 3.1 .

From the result table shown above, one can observe that our proposed non-structured pruning algorithm can learn a more compact sub-network for both LeNet300-100 and LeNet-5 with comparable performance with Han et al. [2015b].

By using a hyperparameter we can directly control the sparsity level to get close to the most compact model achievable. It is not hard to conjecture that using a quality factor times the variance as a pruning threshold in each layer as proposed by Han et al. [2015b] cannot exactly determine how many parameters should be kept. Our method can directly control the sparsity level and therefore enjoy a higher possibility to reach the position of the most compact sub-network.

Table 3.2 shows the layer-by-layer compression comparisons between ours and Han et al. [2015b]. It is interesting to see that although two different pruning algorithms yield a similar performance result, the network architecture is quite different. Our **DSC-1** algorithm controls the directly specified sparsity level in the parameter space with an annealing schedule, this ensures the target sub-network can learn its pattern in an automatic way. For LeNet300-100, the most parameter killing comes from the first layer, which is quite reasonable as the images in the MNIST dataset are

grayscale containing a large portion of pure black pixels. This large portion of black pixels almost has nothing to contribute to the neural network learning of useful information. The least parameter percentage dropping comes from the output layer, preserved as high as 68.5%. We can conjecture the reason for this behavior could be that as the most unrelated features are removed from the first fully connected layer, the output layer should remain a considerable number of parameters to bear the weight of those kept and useful features. For LeNet-5, the most parameter preservation occurs in the first convolutional layer. This is again really very reasonable, as indeed the first layer should be the most important layer that directly extracts relevant features from the raw input image pattern. Our direct sparsity control strategy lets the network itself decide which part is more important, and which part contains most irrelevant or junk connections that could be removed safely. The parameter percentage distribution of the two fully connected layers in LeNet-5 has a similar behavior as in LeNet-300-100.

3.4.2 Structured Channel Pruning on the CIFAR and SVHN Datasets

The CIFAR datasets (CIFAR10 and CIFAR100) provided by Krizhevsky et al. [2009a] are well established computer vision datasets used for image classification and object recognition. Both CIFAR datasets consist of a total of 60K natural color images and are divided into a training dataset with 50K images and a testing dataset with 10K images. The CIFAR-10 dataset is drawn from 10 classes with 6000 images per class. The CIFAR-100 dataset is drawn from 100 classes with 600 images per class. The color images in the CIFAR datasets have resolution 32×32. Some visualization examples of CIFAR dataset are displayed in Figures 3.3 and 3.4.

The SVHN dataset [Netzer et al., 2011] is a real-world image dataset for developing machine learning classification and object recognition algorithms. Similar to MNIST it consists of cropped digit images, but has as many as 600K training samples and 26K testing images in total. Each digit image is 32×32 and extracted from natural scenes. Some visualization examples of the SVHN dataset are displayed in Figure 3.5.

In this section, we will test our structured channel pruning method **DSC-2** on two network models: VGG-16 [Simonyan and Zisserman, 2014] and DenseNet-40 [Huang et al., 2017]. The VGG-16 [Simonyan and Zisserman, 2014] is a deep convolutional neural network containing 16 layers which was mainly designed for the ImageNet dataset. Initially, we planned to use the same variation of the original VGG-16 designed for CIFAR datasets studied in Liu et al. [2017] to have

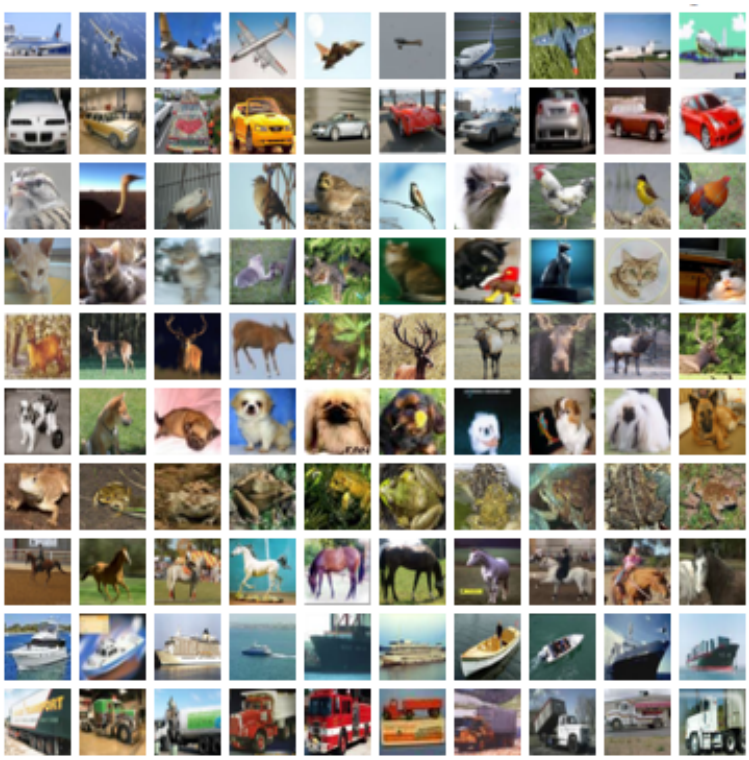


Figure 3.3: Example images in the CIFAR10 dataset [Krizhevsky et al., 2009b].

an identical comparison of our channel pruning method DSC-2 with theirs. However, we had a hard time training it from scratch to obtain a similar baseline performance. So here we adopt another variation of VGG-16 also designed for CIFAR datasets, which was used in Li et al. [2016] and has a smaller number of total parameters, to conduct our experiments and compare with other state of art pruning algorithms. For DenseNet [Huang et al., 2017] we adopted the DenseNet-40 with a total of 40 layers and a growth rate of 12.

We first train all the networks from scratch to obtain similar baseline results compared to Liu et al. [2017]. The total epochs for training was set to 250 epochs for CIFAR, 20 epochs for SVHN, for all networks. The batch size used was 128. A Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of 0.1, weight decay of 5×10^{-4} and momentum of 0.9 was adopted. A division of the learning rate by 5 occurs at every $25\%, 50\%, 75\%$ training epochs. For these datasets, standard data augmentation techniques like normalization, random flipping, and cropping may be applied.

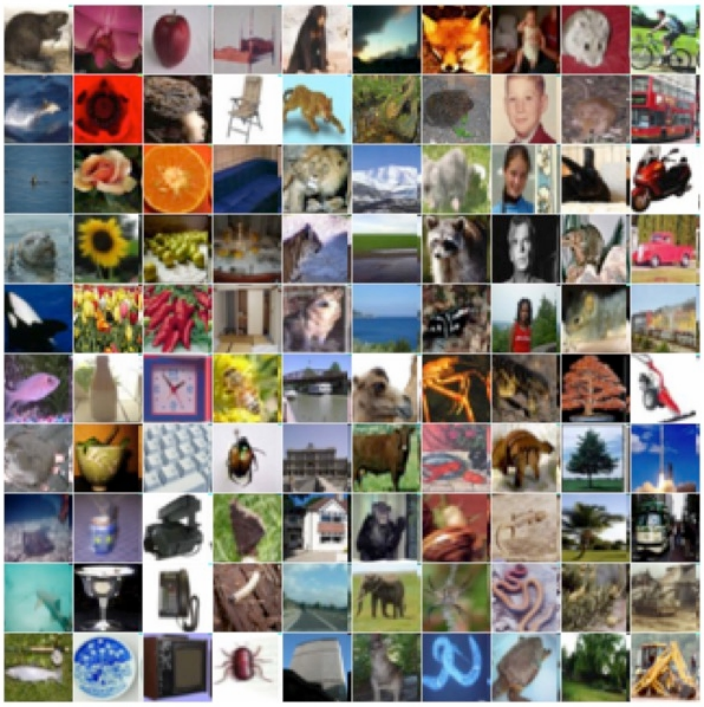


Figure 3.4: Example images in the CIFAR100 dataset [Krizhevsky et al., 2009b].

During the pruning and fine-tuning procedure, the same number of training epochs is adopted in total. We use an SGD optimizer with an initial learning rate of 0.005 and no weight decay or very small weight decay for pruning and fine-tuning purposes. Similarly, a division of the learning rate by 2 occurs at every $25\%, 50\%, 75\%$ training epochs. For the annealing schedule, a grid search is utilized here to determine the best p_0, N^c and α for different p. After the first part of the pruning schedule when we reach the pruning target p_0, we conduct the fine-grain pruning for each final pruning rate p. We output our best results in Table 3.3 for CIFAR 10, Table 3.4 for CIFAR 100 and Table 3.5 for SVHN.

The experimental results displayed in Tables 3.3, 3.4 and 3.5 demonstrate the effectiveness of our proposed channel **DSC-2** pruning algorithm . It can be observed that our **DSC-2** method can obtain results competitive with or even better than Liu et al. [2017]. What's even better, our **DSC-2** pruning method does not introduce any extra term in the training loss function. By using the annealing schedule to gradually remove the "unimportant" channels based on a specified channel importance ranking metric $\mathcal{R}$, we could successfully find a compact sub-network without losing

Figure 3.5: Example images in the SVHN dataset [Netzer et al., 2011].

any model performance. Our **DSC-2** is easy to use and can be easily scaled up to any untrained or existing pre-trained model. The results of the FLOPs ratio between the original DCNNs and pruned sub-networks are shown in Figure 3.6.

Figure 3.7 displays two 70% channel-pruned network models for the CIFAR-10 dataset. Due to the significant differences in network architecture between the VGG-16 and DenseNet-40, the resulting distribution of the percentage of remaining channels is quite different. For VGG-16, only a very small number of channels are kept in the last five CONV layers. This is reasonable as the last five CONV layers are those layers that initially have 512 input channels. Evidently, we do not need so many channels in each of the last five layers. The high pruning percentage may suggest that the VGG-16 network is over-parameterized in a layer-wise way for the CIFAR 10 dataset. For DenseNet-40 with a growth rate of 12, the kept channel percentage is relatively evenly distributed in each CONV layer except the two transitional layers. This is again very reasonable based on the special architecture of DenseNet. With a growth rate of 12, every 12 consecutive layers are correlated with each other, and outputs of those previous CONV layers will be concatenated to be

Table 3.3: Channel pruning performance results comparison on CIFAR-10.

DCNN	Model	Error (%)	Channels	Pruned	Params	Pruned
VGG-16	Base-unpruned [Liu et al., 2017]	6.34	5504	-	20.04M	-
	Pruned [Liu et al., 2017]	6.20	1651	70%	2.30M	88.5%
	Base-unpruned (Ours)	6.34	4224	-	14.98M	-
	Pruned (Ours)	**6.14**	1689	60%	4.40M	70.6%
	Pruned (Ours)	**6.20**	1267	70%	2.88M	80.7%
DenseNet-40	Base-unpruned [Liu et al., 2017]	6.11	9360	-	1.02M	-
	Pruned [Liu et al., 2017]	5.65	2808	70%	0.35M	65.2%
	Pruned (Ours)	**5.48**	3744	60%	0.45M	55.9%
	Pruned (Ours)	**5.57**	2808	70%	0.34M	66.7%

Table 3.4: Channel pruning performance results comparison on CIFAR-100.

DCNN	Model	Error (%)	Channels	Pruned	Params	Pruned
VGG-16	Base-unpruned [Liu et al., 2017]	26.74	5504	-	20.08M	-
	Pruned [Liu et al., 2017]	26.52	2752	50%	5.00M	75.1%
	Base-unpruned (Ours)	26.81	4224	-	15.02M	-
	Pruned (Ours)	26.55	2112	50%	6.01M	60.0%
DenseNet-40	Base-unpruned [Liu et al., 2017]	25.36	9360	-	1.06M	-
	Pruned [Liu et al., 2017]	25.72	3744	60%	0.46M	54.6%
	Pruned (Ours)	**25.66**	3744	60%	0.47M	55.6%

the inputs of the following CONV layer inside the growth rate period. Only the transitional layers do not hold that property. Overall, our channel-level pruning algorithm **DSC-2** can automatically detect the reasonable sub-network without performance loss for VGG-16 and DenseNet-40 on the CIFAR and SVHN datasets.

Figure 3.8 displays the best results we obtained for just using one single global pruning rate $p \in \{0.1, 0.3, 0.5, 0.7\}$ to perform the channel pruning in the whole channel space. We can observe

Table 3.5: Channel pruning performance results comparison on SVHN.

DCNN	Model	Error (%)	Channels	Pruned	Params	Pruned
VGG-16	Base-unpruned [Liu et al., 2017]	2.17	5504	-	20.04M	-
	Pruned [Liu et al., 2017]	**2.06**	2201	60%	3.04M	84.8%
	Base-unpruned (Ours)	2.18	4224	-	14.98M	-
	Pruned (Ours)	**2.06**	1689	60%	4.31M	71.2%
DenseNet-40	Base-unpruned [Liu et al., 2017]	1.89	9360	-	1.02M	-
	Pruned [Liu et al., 2017]	1.81	3744	60%	0.44M	56.6%
	Pruned (Ours)	**1.80**	3744	60%	0.46M	54.9%

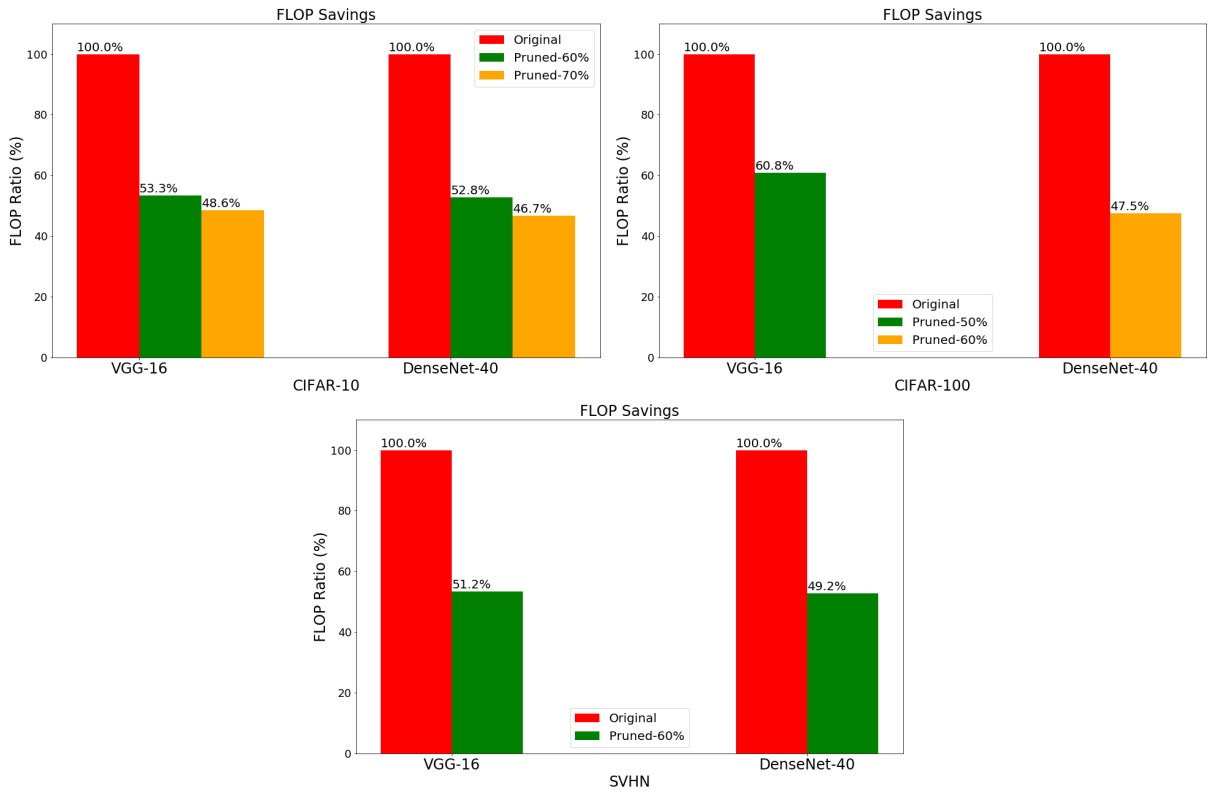
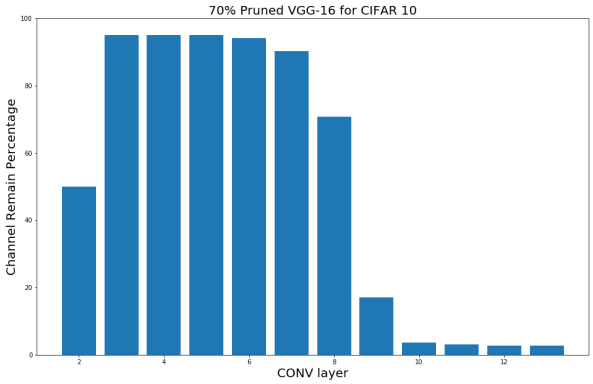
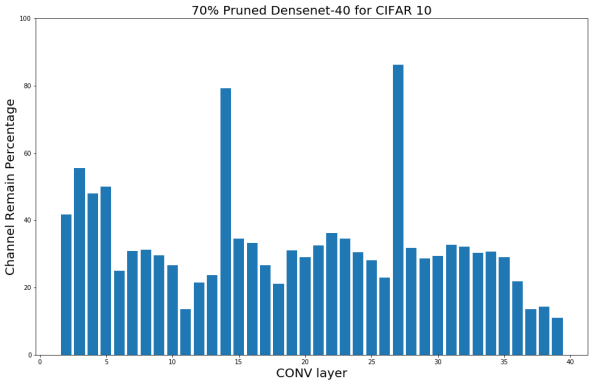


Figure 3.6: FLOPs ratio between the original DCNNs and pruned sub-network for VGG-16 and DenseNet-40 on CIFAR and SVHN dataset.

that even using a single target pruning rate parameter, we can still obtain very good sub-networks that generalize to CIFAR-10 test data well. A large portion of our best results displayed in Tables 3.3, 3.4 and 3.5 are obtained just using a single global pruning rate to guide the network pruning procedure. This observation makes our annealing pruning algorithm very easy to use, without worrying about channel subspace partitions. We also tested three different values for the parameter α of Eq. (3.4). For DenseNet-40, all of the choices of α can yield satisfactory performance for the pruned sub-network. For VGG-16, when α is set to 0, that is when the magnitude of γ is used as the ranking metric for channel pruning, gives the best results. This implies that different α values may be suitable for different network architectures.



70% channel pruned VGG-16

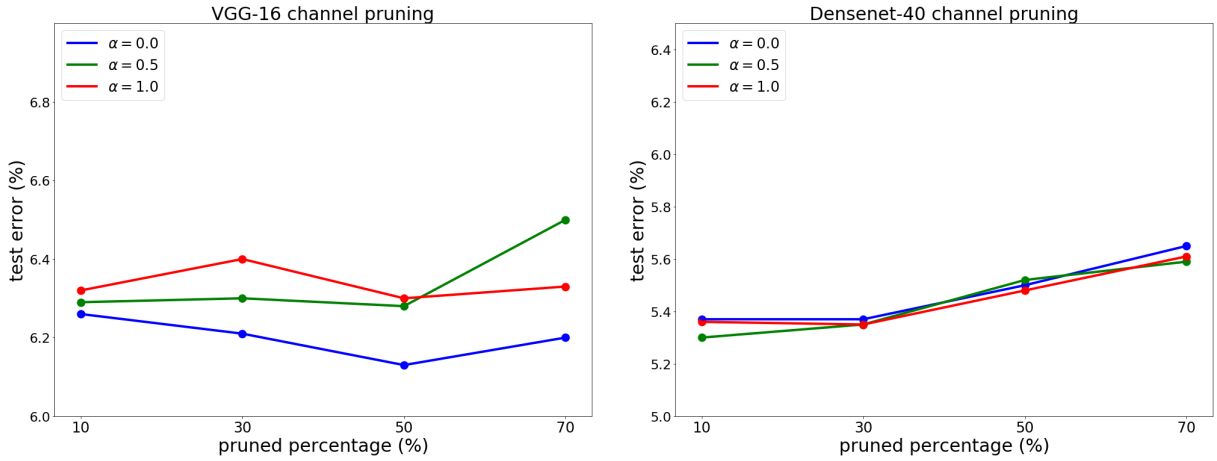


70% channel pruned DenseNet-40

Figure 3.7: The remaining channel distribution for each CONV layer for pruned networks on CIFAR 10. The first CONV layer is not displayed as our channel pruning algorithm DSC-2 will not act on this layer, thus contain the full percentage of channels.

3.5 Conclusions

This chapter presented a neural network pruning framework that is suitable for both structured and non-structured pruning. The method directly imposes a L_0 sparsity constraint on the network parameters, which is gradually tightened to the desired sparsity level. This direct control allows us to obtain the precise sparsity level desired, as opposed to other methods that obtain the sparsity level indirectly through either a quality factor times the variance or the use of penalty parameters. Experiments on extensive image classification real data, including the MNIST, CIFAR, and SVHN provide empirical evidence that the proposed network pruning scheme obtains a performance comparable to or better than other state of art pruning methods.



VGG-16 DenseNet-40

Figure 3.8: The test error for various channel pruned percentage using one single global pruning rate for CIFAR-10.

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

In this work, we studied the feature interactions and pruning problems on neural networks. The major contributions of this dissertation are:

1. We presented an empirical study of the learnability of neural networks on some non-linear, particularly XOR-based data with extra irrelevant variables. Our experiments show that the logistic loss function on this data has many local optima, and the number of local optima grows exponentially with the number of irrelevant variables. We also observe that there exists a phase transition from an "easy to train" regime where the local minima are easy to find, to the "hard to train" regime with the increasing of the total noisy features.

2. We presented a node selection method for training a neural network that avoids many local optima by starting with a model with many hidden neurons and gradually removing the weaker ones to obtain a compact model trained in a deep minimum of the loss function. We used a neural rule normalization technique to further improve the correctness and keep the important hidden neurons during the dropping procedure. We also proposed a feature selection procedure inspired by our node selection technique by either using the "group criterion" or "separate criterion" to select the most important features.

3. The performance of the obtained pruned sub-network is hard to achieve by retraining the pruned nework from a random initialization or from the original network initial values, due to the existence of many shallow local optima around the deep minimum. Experiments also show that the node selection method is useful in improving generalization of fully connected neural networks on the parity data with irrelevant noisy features compared to those neural networks trained by using the traditional SGD, Adam or a newly proposed BoostNet method, and on a number of real datasets compared to normal fully connected neural networks and so-called "equivalent" neural networks.

4. We extended our method to a network pruning framework that is suitable for dealing with various kinds of deep convolutional neural networks (DCNNs). We combined regularization techniques with sequential algorithm design and direct sparsity level control to bring forward a novel network pruning framework that could be applicable to either non-structured pruning

or structured pruning (particular for filter channel-wise pruning of DCNNs). This direct control allows us to obtain the precise sparsity level desired, as opposed to other methods that obtain the sparsity level indirectly through either a quality factor times the variance or the use of penalty parameters.

5. An attractive property of our network pruning framework is that the parameters or filter channels are removed while the model is updated at each iteration, which makes the problem size decrease during the iteration process. Extensive experiments on real vision datasets, including MNIST, CIFAR and SVHN, provide empirical evidence that the proposed network pruning framework has performance comparable to or better than other state of art pruning methods.

Neural network techniques are more and more popular nowadays, and studies of how to reduce the complexity of the neural network architectures have drawn much attention recently. Our proposed pruning algorithm based on an annealing schedule can be easily applied to compress many neural network architectures. The pruned neural networks, which do not sacrifice much accuracy, can be deployed to various computing resource-limited devices such as cell-phones, embedded systems, etc.

Other possible applications of this work are in learning complex non-linear models for non-vision data, e.g. bio-informatics, power systems, physics, etc, where irrelevant variables are present and thus feature selection is essential.

4.2 Future Work

The future research will be explored in several directions:

- We would like to extend our work to deal with the recurrent neural networks like LSTM [Hochreiter and Schmidhuber, 1997] and GRU [Cho et al., 2014]. We have demonstrated that our annealing pruning algorithms work well for feedforward fully connected neural networks and deep convolutional neural networks. It is natural to extend our work to deal with the neural network with recurrent loops, but there are challenges related to how to actually do it the right way.

- We would like to study the application of our work in the Dictionary Learning [Tosic and Frossard, 2011] field. Dictionary learning aims at obtaining a sparse linear representation for high dimensional data in the image processing area. Our annealing algorithm is proved to be useful to train a sparse neural network at the end, so there exists the possibility that we

can apply our algorithms to obtain a sparse representation for high dimensional image data effectively in an unsupervised setting.

- Another future direction for exploration is Kernel Density Estimation, where probability density functions are constructed using many Gaussian kernels around the training observations. Our annealing procedure could be applied to keep a small number of most relevant kernels while minimizing the KL divergence to the original p.d.f.

- Another direction is to study how to add parameters or convolutional channels back from heavily pruned sub-neural networks. Our node selection and network pruning algorithms are top-down approaches: We first train large over-parameterized neural networks to enjoy a higher probability to learn the desired pattern from the training data, then we gradually remove those redundant parameters or channels to obtain a more compact sub-network without performance sacrifices. It is quite interesting to think about in opposite directions: given a sparse neural network with just a few parameters or channels, how feasible is to gain performance or generalization power by adding more parameters or channels to the model but still keep it compact. This bottom-up way to study a sparse neural network may be more challenging than the top-down approaches.

BIBLIOGRAPHY

Adrian Barbu, Yiyuan She, Liangjing Ding, and Gary Gramajo. Feature selection with annealing for computer vision and big data learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(2):272–286, 2017.

Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Gitesh Dawer, Yangzi Guo, Sida Liu, and Adrian Barbu. Neural rule ensembles: Encoding sparse feature interactions into neural networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, 2020.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Pengfei Dou, Shishir K Shah, and Ioannis A Kakadiaris. End-to-end 3d face reconstruction with deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5908–5917, 2017.

Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A Hamprecht. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*, 2018.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2019.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8803–8812, 2018.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *NIPS*, pages 545–552, 2004.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015b.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Alekseĭ Grigorevich Ivakhnenko and Valentin Grigorévich Lapa. *Cybernetic predicting devices*. CCM Information Corporation, 1965.

Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Generalization bounds for neural networks through tensor factorization. *CoRR, abs/1506.08473*, 1, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009a.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6, 2009b.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Pat Langley. The changing science of machine learning. *Machine Learning*, 82(3):275–279, 2011.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6391–6401, 2017.

Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, David Doermann, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Exploiting kernel sparsity and entropy for interpretable cnn compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2800–2809, 2019.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, pages 855–863, 2014.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

John McCarthy and Edward A Feigenbaum. In memoriam: Arthur samuel: Pioneer in machine learning. *AI Magazine*, 11(3):10–10, 1990.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Ivana Tosic and Pascal Frossard. Dictionary learning. *IEEE Signal Processing Magazine*, 28(2): 27–38, 2011.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

Paul Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

Yuchen Zhang, Jason Lee, Martin Wainwright, and Michael I Jordan. On the learnability of fully-connected neural networks. In *Artificial Intelligence and Statistics*, pages 83–91, 2017.

Zhengguang Zhou, Wengang Zhou, Richang Hong, and Houqiang Li. Online filter weakening and pruning for efficient convnets. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2018.

BIOGRAPHICAL SKETCH

Yangzi Guo is currently a PhD candidate majoring in Applied and Computational Mathematics at the Mathematics Department of Florida State University in Tallahassee, Florida, USA. His research interests include machine learning, supervised learning and neural networks. After obtaining his bachelor's degree major in Mathematics and Applied Mathematics from China, he flew to the United States and attended Florida State University to pursue his graduate career. In 2017, he obtained the Master of Science in Mathematics.