Regulating Orthogonality of Feature Functions for Highly Compressed Deep Neural Networks

by

Wei-Chen Wang

B.S., B.A., University of Illinois at Urbana-Champaign (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
May 13, 2022

Certified by .....
Lizhong Zheng
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Regulating Orthogonality of Feature Functions for Highly Compressed Deep Neural Networks

by

Wei-Chen Wang

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 2022, in partial fulfillment of the
requirements for the degree of
Masters of Science

Abstract

When designing deep neural networks (DNN), the number of nodes in hidden layers can have a profound impact on the performance of the model. The information carried by the nodes in each layer creates a subspace, whose dimensionality is determined by the number of nodes and their linear dependency. This paper focuses on highly-compressed DNN – network with significantly less nodes in the last hidden layer than in the output layer. Each node in the last hidden layer is considered a feature function, and we study how the orthogonality of feature functions changes throughout the training process. We first develop how information is learned, stored and updated in the DNN throughout training, and propose an algorithm which regulates the orthogonality before and during training. Our experiment on high-dimensional mixture Gaussian dataset reveals that the algorithm achieves higher orthogonality in feature functions, and accelerates network convergence. Orthogonalizing feature functions enable us to approximate Newton's method via the gradient descent algorithm. We can take advantage of the superior convergence properties of the second-order optimization, without directly computing the Hessian matrix.

Thesis Supervisor: Lizhong Zheng
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

In the past two years, I have grown to become a better student, researcher, person, and none of which would have been possible without the support, guidance and love I received from everyone around me.

First and foremost, I would like to express my sincere gratitude to my thesis advisor, Prof. Lizhong Zheng, for his unique advising style. When I first joined the group, I had little knowledge in information theory, and my lack of mathematical maturity was detrimental to the research progress. In the first few months, Lizhong patiently explained fundamental concepts in detail, and emphasized on the connections between topics. As I started taking graduate classes in information theory, these connections became powerful tools for me to have a deeper understanding of the course materials. Outside of research, he cares deeply about the future success of his students. His unique perspective motivates me to reflect on the purpose and meaning of my graduate career. Looking back, he is one of the few people who encourages me to take risks, and I am grateful for the opportunity to explore alternative career paths.

Next, I would like to acknowledge my labmates Erixhen Sula and Xiangxiang Xu for their contributions to my thesis. Several ideas in this paper were inspired by our discussions, and your advice helped me solve many challenging problems. I would also like to thank my other labmates, Mohamed Ibrahim AlHajri, Melihcan Hasan Sabri Erol and Jiejun Jin. You have not only been my closest friends at MIT, but also my mentors who gave me advice and guided me on the right path. In addition, I want to thank Melihcan for being my pset partner. I am always amazed by your ability to quickly and elegantly solve every challenging problem that I have no idea how to start after staring at it for hours.

Being a student at MIT, I am privileged to be taught by the world renowned faculties, including Prof. Gregory Wornell, Prof. Polina Golland, Prof. David Gamarnik, Prof. Patrick Jaillet, and Prof. Michael Sipser. You have been great sources of knowledge, and I thank you for your amazing classes. I am also grateful for Prof. Nidhi Seethapathi for the opportunity to collaborate with her on the inverse reinforcement learning project. I am excited to apply my knowledge in information theory and computer science to solve real world problems in biomechanics.

During my time at the University of Illinois at Urbana-Champaign, a number of professors have inspired me to pursue my studies in computer science, including Prof. Neal Davis and Prof. Hope Michelson. I also want to thank many people who have left a profound impact during my four years of undergraduate.

Lastly, I want to thank my parents for their love and encouragement, and I dedicate this thesis to them. To my girlfriend, Ainsley, thank you for your infinite support, and I am grateful to have you as someone I can always rely on. The past two year has been challenging, and I could not have made it through without you.

Contents

List of Figures

Chapter 1

Introduction

1.1 Neural Network

Artificial neural networks, or simply neural networks (NN) in this context, are machine learning algorithms which model how human brains operate. Inspired by neurological systems, NN consists of neurons and edges, resembling how signals are transmitted through the nervous system. A NN has one input layer and one output layer, and can have hidden layers in between. Activation functions are added after layers to enable a non-linear transformation. NN with hidden layers are considered deep neural networks (DNN), and excels at handling tasks with high-dimensional input [1], such as computer vision [2], speech recognition [3, 4] and natural language processing [5].

A typical NN training process alternates between two phases. In forward propagation, training data are passed from the input layer, and go through each layer from left to right. The results at the output layer are compared to the true label with a specified loss function. During backward propagation, the gradient of the loss with respect to each weight is computed. To minimize the loss, the weights are adjusted accordingly layer by layer in reverse direction.

1.2 Feature Functions

In this paper, we focus on DNN with softmax being the last activation function, as illustrated in Figure 1. We shift our focus to the last hidden layer. Let x and y denote the input and output, respectively, and k denote the number of nodes in the last hidden layer. The nodes in the last layer are *feature functions* of the input, namely $f_1(x), \ldots, f_k(x)$, which span a linear subspace and provide information for inference.

The number of feature functions in the last hidden layer plays an important role in the accuracy and the complexity of the model. If k is greater than $\min\{|x|, |y|\}$, the linear subspace spanned by the feature functions are more capable of explaining the dataset. In such cases, the model tends to have high testing accuracy, yet the k nodes usually carry redundant information, and can potentially lead to overfitting if k is too large. On the other hand, when k is less than $\min\{|x|, |y|\}$, the last hidden layer has limited expressive power as it can only span a low-dimensional subspace. Since it cannot fully represent the complexity of the dataset, testing accuracy tends to be low. However, under the constraint that the last hidden layer is highly compressed, the k feature functions are encouraged to extract the most relevant information from the dataset. This enables the features to be more unique and thus orthogonal, which has many desirable properties in practice.

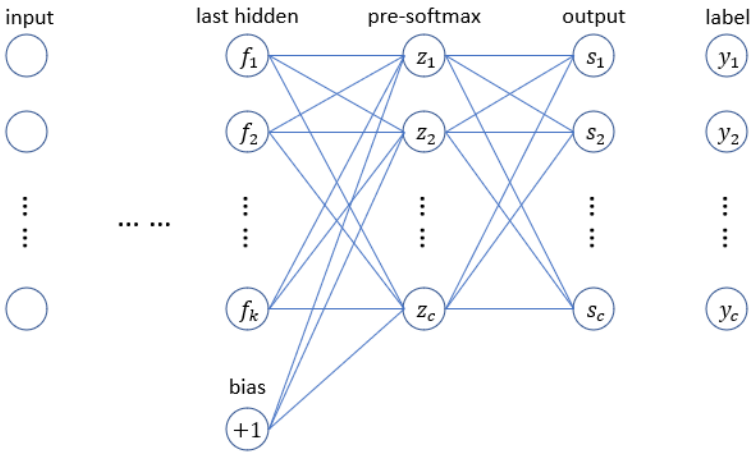


Figure 1: Deep neural network for multi-class classification problem

1.3 Outline

First, a theoretical analysis on the convergence of NN is presented, with its accuracy and orthogonality of features with varying number of nodes compared. Next, using a high-dimensional mixture Gaussian dataset, we design a NN under the constraint of a low number of features in the last hidden layer. We propose an algorithm which adds a regulator to the loss function to incentivize the feature functions to be more orthogonal. Its performance is compared to the regular training method, and we consider an extension, in which the feature functions are orthogonalized prior to training. Finally, we show that the proposed algorithm is an approximation to the second-order optimization. Our implementation is available on GitHub[1].

[1] `https://github.com/ericwang1997/Feature-Orthogonalization`

Chapter 2

Related Work

In the past few decades, NN has shown promising results in solving complicated problems. For almost all datasets, we can design a similar NN structure and achieve high accuracy with limited parameter tuning [6]. A theoretical analysis on the convergence of NN shows that low-dimensional features extract the most relevant information from a high-dimensional data [7]. These inferences are *universal* in the sense that the properties of the feature functions are locally equivalent, and reveal the relationship between singular value decomposition, maximal correlation, canonical correlation and principle component analysis in neural networks [8].

Despite their best efforts, researchers have yet discovered a systematic way to determine the hyper-parameters of neural networks. Trial-and-error remains the most common approach, in which every combination of hyper-parameters is experimented and evaluated [6, 9]. Several approaches have been proposed, including searching for an optimal learning rate, batch size, momentum and weight decay without going through all combinations [10]. Other parameters, such as the number of nodes in a layer, still rely on rule-of-thumb [11], such as the number of neurons in hidden layers being 70-90% of the input size, and between input and output sizes [12, 13]. Other methods include starting from a low number of neurons in the hidden layers, and subsequently adding more if the model underfits (or starting from a high number of neurons and removing some of them if the model overfits) [14].

Most neural network libraries today rely on first-order optimization for back

propagation, with gradient descent being the most common method [15]. For low-dimensional non-convex data, first-order optimization is susceptible to obtaining local minima. However, this phenomenon does not translate to high-dimensional data, whose critical points with error terms much larger than that of the global minima are exponentially more likely to be saddle points instead of local minima [16, 17]. Despite its popularity and simplicity, first-order optimization still suffers from slow convergence and stagnation near flat regions or saddle points [18].

We list several methods which have been proposed to improve gradient descent. First, stochastic gradient descent aims to reduce computational cost by randomly choosing one data point (or more commonly one *mini-batch*, which is a small subset of the entire dataset) to approximate the gradient in each iteration [19]. Second, batch normalization increases the speed and stability of NN training by normalizing each mini-batch [20]. The reason why the method is effective remains debatable, as the original authors claim that it reduces internal covariate shift, while others argue that it smoothens the objective function [20, 21]. Third, initializing NN weights with an orthogonal matrix alleviates the issue of vanishing or exploding gradients, which becomes common as the depth of the DNN increases [22, 23]. It also speeds up convergence compared to the standard Gaussian initialization [24].

On the other hand, Newton's method in optimization, or second-order optimization, is an alternative which takes the Hessian matrix into consideration. By doing so, the information from the curvature enables the optimization to take a more direct step and can dramatically accelerate convergence [25]. Despite having superior convergence properties, second-order optimization are not commonly used in practice due to its computational infeasibility [26], both for computing and storing the Hessian matrix. Furthermore, Newton's method moves towards the eigen-direction with positive eigenvalue, making saddle points an attractor [27].

Chapter 3

Expressive Power of Neural Networks

3.1 Maximal Correlation

Consider discrete random variables X and Y with joint distribution $P_{X,Y}$, over finite alphabets $\mathcal{X} = \{1, 2, \ldots, |\mathcal{X}|\}$ and $\mathcal{Y} = \{1, 2, \ldots, |\mathcal{Y}|\}$, respectively. Here, we introduce the Hirschfeld-Gebelein-Rényi (HGR) maximal correlation $\rho(X;Y)$ as a normalized measure to quantify the dependence between X and Y:

$$\rho(X;Y) \triangleq \max_{\substack{f:\, \mathcal{X}\to\mathbb{R},\, g:\, \mathcal{Y}\to\mathbb{R} \\ \mathbb{E}[f(X)]=\mathbb{E}[g(Y)]=0 \\ \mathbb{E}[f^2(X)]=\mathbb{E}[g^2(Y)]=1}} \mathbb{E}[f(X)g(Y)] \tag{1}$$

Observe that the maximum is taken over all functions f and g with zero mean and unit variance. Intuitively, HGR maximal correlation measures the correlation between the most correlated function mappings of X and Y [28]. Some useful properties include $0 \leq \rho(X;Y) \leq 1$, where equality on the LHS is achieved iff X and Y are independent; equality on the RHS is achieved iff there exists some mapping such that $f(X) = g(Y)$ [29]. HGR maximal correlation can be extended to higher dimensional spaces, where we consider the correlation in k-dimensional function mappings:

$$\rho_k(X;Y) \triangleq \max_{\substack{f:\, \mathcal{X}\to\mathbb{R}^k,\, g:\, \mathcal{Y}\to\mathbb{R}^k \\ \mathbb{E}[f(X)]=\mathbb{E}[g(Y)]=\mathbf{0} \\ \mathbb{E}[f(X)f^T(X)]=\mathbb{E}[g(Y)g^T(Y)]=\mathbf{I}}} \mathbb{E}[f^T(X)g(Y)] \tag{2}$$

where the special case $k = 1$ corresponds to (1). The HGR maximal correlation is closely related to the field of machine learning, in which X and Y are viewed as inputs and labels, respectively. Under this setup, f^* can be considered the optimal feature functions to predict Y, and g^* the corresponding weights [30]. Next, we introduce matrix $\mathbf{B} \in \mathbb{R}^{|\mathcal{Y}|\times|\mathcal{X}|}$, also known as the divergence transition matrix (DTM), whose entries are:

$$\mathbf{B}(y,x) = \frac{P_{XY}(x,y)}{\sqrt{P_X(x)}\sqrt{P_Y(y)}} \tag{3}$$

or in matrix form:

$$\mathbf{B} = \left[\sqrt{P_Y}\right]^{-1} P_{Y,X} \left[\sqrt{P_X}\right]^{-1} \tag{4}$$

where P_X and P_Y are the marginal distributions. It is known that $\rho(X;Y)$ is the second largest singular value of $\mathbf{B}$ [6, 8], and f^* and g^* are chosen such that

$$\left[f^*(1)\sqrt{P_X(1)}, \ldots, f^*(|\mathcal{X}|)\sqrt{P_X(|\mathcal{X}|)}\right]^T \text{ and } \left[g^*(1)\sqrt{P_Y(1)}, \ldots, g^*(|\mathcal{Y}|)\sqrt{P_Y(|\mathcal{Y}|)}\right]^T \tag{5}$$

are the right and left singular vectors of $\mathbf{B}$ corresponding to the second largest singular value [28]. Given that the largest singular value is always 1, it is convenient to subtract the largest singular value from $\mathbf{B}$. We construct $\tilde{\mathbf{B}}$, also referred as the canonical dependence matrix (CDM) as follows [8]:

$$\tilde{\mathbf{B}}(y,x) = \mathbf{B}(y,x) - \sqrt{P_X(x)}\sqrt{P_Y(y)} = \frac{P_{XY}(x,y) - P_X(x)P_Y(y)}{\sqrt{P_X(x)}\sqrt{P_Y(y)}} \tag{6}$$

In this case, the HGR maximal correlation becomes the largest singular value of $\tilde{\mathbf{B}}$. The generalized version of HGR maximal correlation retains a similar property, such that ρ_k is the sum of the k largest singular values of $\tilde{\mathbf{B}}$, and f^*, g^* are the k top right and left singular vector, respectively [28]. Consider the basic properties of the singular value decomposition. The number of non-zero singular values equals the rank of $\mathbf{B}$, which is upper bounded by $\min\{|\mathcal{X}|, |\mathcal{Y}|\}$. It implies that

$$\rho(X;Y) \leq \rho_2(X;Y) \leq \rho_3(X;Y) \leq \cdots \leq \rho_t(X;Y) = \rho_{t+1}(X;Y) = \ldots \tag{7}$$

where $t = \min\{|\mathcal{X}|, |\mathcal{Y}|\} - 1$. In the context of neural networks, the NN reaches the theoretical upper bound of expressive power when its last hidden layer has t nodes (assuming all other hidden layers have at least t nodes), and additional nodes do not enable the NN to infer further information.

3.2 Feature Projection

Through the lens of maximal correlation, we explore how the NN operates during training. The NN first initializes its weights[1] to form random feature functions $f(x)$ and random output layer weights $g(y)$. During back-propagation, we first fix $f(x)$ and update $g(y) = \mathbb{E}[f(X)|Y = y]$, which are the optimal weights that best align with $f(x)$. Next, we fix $g(y)$ and update $f(x) = \mathbb{E}[g(Y)|X = x]$, which are the optimal feature functions that best align with $g(y)$. After some iterations, we reach a stable point where $f(x)$ and $g(y)$ align with each other. This procedure is referred as feature projection [6], where the NN searches for two functional mappings which achieve the highest maximal correlation.

Let T denote the function which maps the input to the desired output, and $U \triangleq span(f_1, \ldots, f_k)$ denote the span of feature functions. In each iteration, the NN seeks to minimize the loss function by projecting T onto U. In practice, computing such a projection is expensive. Hence, for stochastic gradient descent, it is common to approximate the projection by projecting T onto each feature independently, and sum up the projections. Note that $proj_U T = \sum_{i=1}^{k} proj_{u_i} T$ if the features are completely orthogonal. Given that this rarely happens, the approximation in each iteration deviates from the true projection but eventually converges. To compensate with the difference, a small learning rate is used to prevent overshoot. When the features are highly correlated, the NN tends to spend more iterations to converge.

[1]For a fully-connected linear layer with n inputs, the weights are drawn randomly from a uniform distribution between $-1/\sqrt{n}$ and $1/\sqrt{n}$ by PyTorch's default [15]. Other methods such as Kaiming and Xavier initialization are also widely popular.

3.3 Orthogonality of Feature Functions

Suppose we have inputs $x_1, \ldots, x_n$ which are passed through the neural network from the input layer. Consider the values at the last hidden layer with k nodes, namely $f_1(x_i), \ldots, f_k(x_i)$ for $i = 1, \ldots, n$. We construct a matrix M of shape $k \times n$ such that its (i, j)'th entry is the value at the i'th node of the last hidden layer for the j'th input. Define the covariance matrix[2]:

$$K_{MM} \triangleq \frac{1}{n-1} \sum_{i=1}^{n} (f - \bar{f})(f - \bar{f})^T \tag{8}$$

where $\bar{f} = \frac{1}{n} \sum_{i=1}^{n} f$. Observe that the (i, j)'th entry of K_{MM} is simply the average dot product of the i'th and j'th pair of feature functions. We propose a measure s to measure the orthogonality of the covariance matrix, which is simply the sum of squared diagonal terms of K_{MM} divided by the sum of square of all terms of K_{MM}, or more formally:

$$s = \frac{\sum_i (K_{MM}(i,i))^2}{\sum_{i,j} (K_{MM}(i,j))^2} = \frac{tr(diag(K_{MM})^2)}{tr(K_{MM}^T K_{MM})} \tag{9}$$

where $A(i, j)$ denotes the (i, j)'th entry of A, $tr(A)$ denotes the trace of A, and $diag(A)$ denotes a square diagonal matrix such that its (i, i)'th entry is $A(i, i)$. Clearly, $0 \leq s \leq 1$, where the right equality is achieved iff K_{MM} is a diagonal matrix, which implies that the features are orthogonal to each other. Using the definitions and metric above, we study the orthogonality of features of an untrained NN. Since feature functions are initially formed by randomly chosen weights, they can also be seen as k randomly chosen functions in a m-dimensional space, where k and m are the number of nodes in the last hidden layer and the dimensionality of the inputs, respectively. As shown in Figure 2, a lower m and a higher k correspond to a lower s. This is an expected result – when more feature functions are randomly chosen in a lower-dimensional space, it is more likely that the feature functions are more redundant, hence less orthogonal.

[2]We compute the unbiased estimator of $K_{MM} \triangleq \mathbb{E}[(M - \mathbb{E}[M])(M - \mathbb{E}[M])^T]$

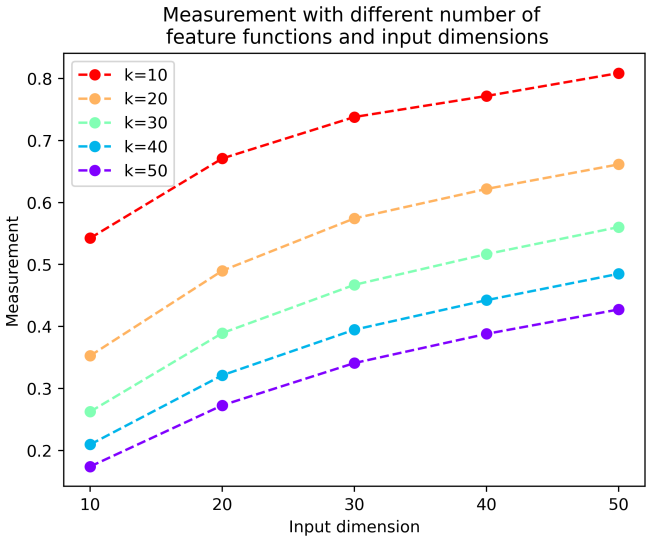


Figure 2: Input dimension vs. measurement with varying k. As shown above, when less feature functions were chosen from a higher-dimensional space, they tend to be more orthogonal. The above result is the average of 100 runs with different random seeds, using the dataset described in section 4.3.

Chapter 4

Experiment

4.1 Neural Network Model

In this paper, we study the multi-class classification problem. Consider the dataset with n samples $\{(x_i, y_i)\}_{i=1}^n$ such that $x_i \in \mathbb{R}^m$ is the input and y_i is its label. We represent each label as an one-hot encoded vector[1] of size c, where c is the number of classes. We focus on NN with at least one hidden layer, as shown in Figure 1. The nodes in the last hidden layer are feature functions, connected to the pre-softmax layer with a fully connected linear layer such that $z = wf + b$, where w is the linear layer weights and b is the bias. The outputs are fed to a softmax function to exponentially normalize the probabilities, such that

$$s_i = \frac{e^{z_i}}{\sum_{l=1}^{n} e^{z_l}} \tag{10}$$

and the softmax outputs are compared to the true labels using cross-entropy loss:

$$L(y, s) = -\sum_{i=1}^{c} y_i \log s_i \tag{11}$$

[1]Without loss of generality, assume that each sample has label between 0 and $c-1$. The one-hot encoding representation of label y_i is a vector whose $(i+1)$'th entry is 1, and 0 elsewhere.

4.2 Motivation

To better understand the updates during back-propagation, we analyze the gradient of the loss function with respect to each layer in Appendix A. Intuitively, the process is equivalent to projecting the *correct* function that maps the input to the output onto each feature function and summing up the projections, instead of projecting onto the linear span of feature functions. In practice, the NN does not have knowledge of the correct function, but instead projects the empirical distribution of a mini-batch of samples in the case of stochastic gradient descent. As shown in section 3.2, the two approaches are equivalent when the features are orthogonal. When a large number of feature functions are drawn, they are expected to over-represent the complexity of the problem, thus containing more redundant information. Therefore, this paper focuses on highly compressed DNNs, which indicate that they have significantly less nodes in the last hidden layer compared to the input and output dimensions. For these networks, we investigate how the orthogonality of features change throughout the training of the model, and whether we can accelerate the convergence by encouraging them to be more orthogonal.

4.3 Data Generation and Neural Network Design

We first generate a dataset from a mixture Gaussian distribution using Algorithm 1, which enables us to adjust the complexity and dimensions of the dataset. The function takes five parameters, which refer to the dimension of input, number of output classes, cluster per class, samples per cluster, and standard deviation for the multivariate normal distribution covariance. Note that the code implementation differs slightly from Algorithm 1 for computational optimization, and a random seed is fixed to ensure the consistency of the dataset. The centers were uniformly chosen in $[0, 1]^{\text{xdim}}$, and $\mathbf{\Sigma}$ is a diagonal matrix such that each diagonal entry is σ^2. A visual illustration of the dataset in 3D is shown in Figure 3.

Algorithm 1 Generate samples from a mixture Gaussian distribution

function GETSAMPLES(xdim, ycard, cpy, spc, sigma)
  centers $\leftarrow$ ycard $\times$ cpy coordinates uniformly chosen in xdim
  nSample $\leftarrow$ ycard $\times$ cpy $\times$ spc
  data, labels $\leftarrow$ initialize arrays of shapes [nSample, xdim] and [nSample]
  for $i = 0, \ldots,$ nSample $- 1$ **do**
    labels$[i] = i$ mod ycard
    data$[i]$ = centers[labels$[i]] + \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ // Multivariate Normal
  end for
  shuffle data and labels accordingly
  return data, labels
end function

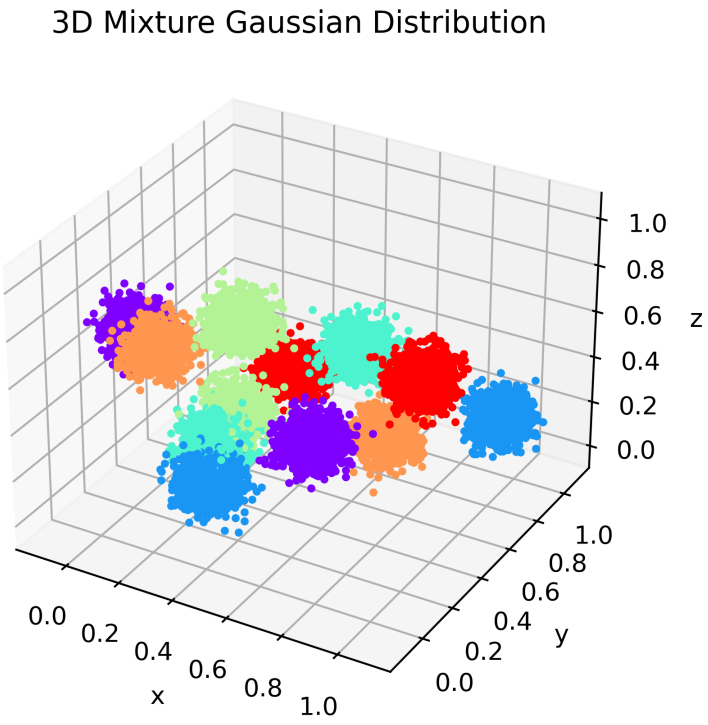


Figure 3: A visual representation of the 3D mixture Gaussian distribution, generated from Algorithm 1, `GETSAMPLES(3, 6, 2, 1000, 0.05)`. Each class is represented by a different color. A specific random seed is chosen for a better visualization.

We generate $100,000$ samples using `GETSAMPLES(50, 50, 2, 1000, 0.1)`, and partition the samples to the training and testing dataset with a $3:1$ ratio. The parameters are chosen to ensure a reasonable complexity and size of the dataset. We design a DNN with the following layers:

layer	nodes	activation function
input	xdim$=50$	N/A
hidden	k	tanh
output	ycard$=50$	softmax

We use stochastic gradient descent as our optimization method, with a batch size of $1,000$ and a learning rate of 0.05. Cross-entropy loss is chosen as the loss function to solve the multi-class classification problem. To verify the claim in section 3.3, we experiment[2] different number of feature functions on the same dataset. As shown in Figure 4, the accuracy increases as k increases, while the features become less orthogonal.

[2]All experiments in this work are conducted for 100 times, and a different random seeds is fixed to initialize the NN for each trial. The results are shown either in a box plot or a line graph (where average is shown).

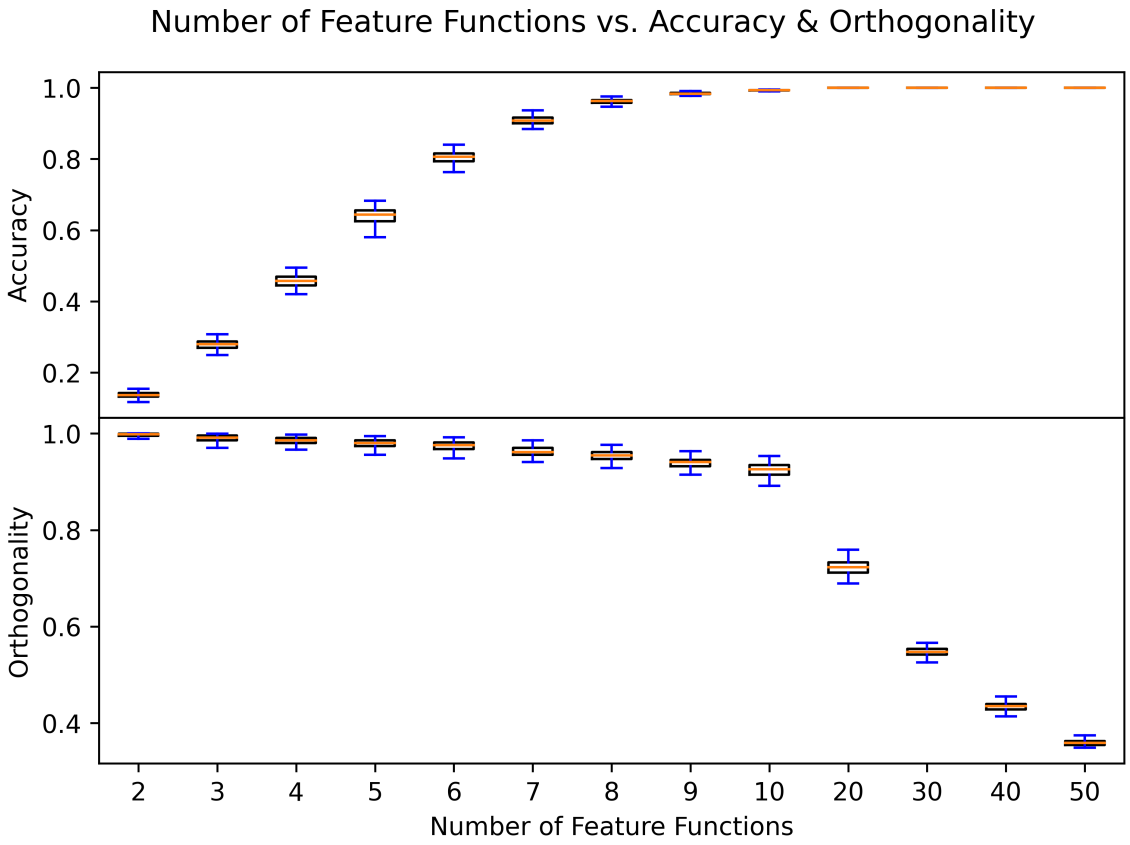


Figure 4: We train neural networks with same structures except for the number of nodes (feature functions) in the hidden layer. The above shows the accuracy and orthogonality of feature functions after 100 epochs of training. Outliers are removed due to small IQRs.

Chapter 5

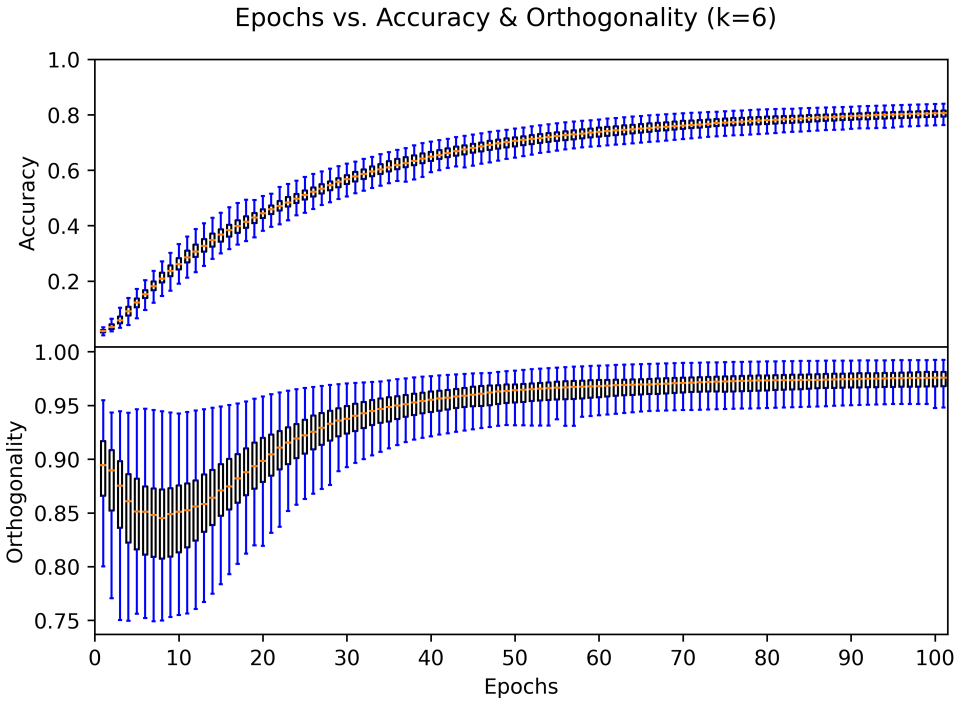
Increasing Orthogonality of Feature Functions

5.1 Orthogonality during Training

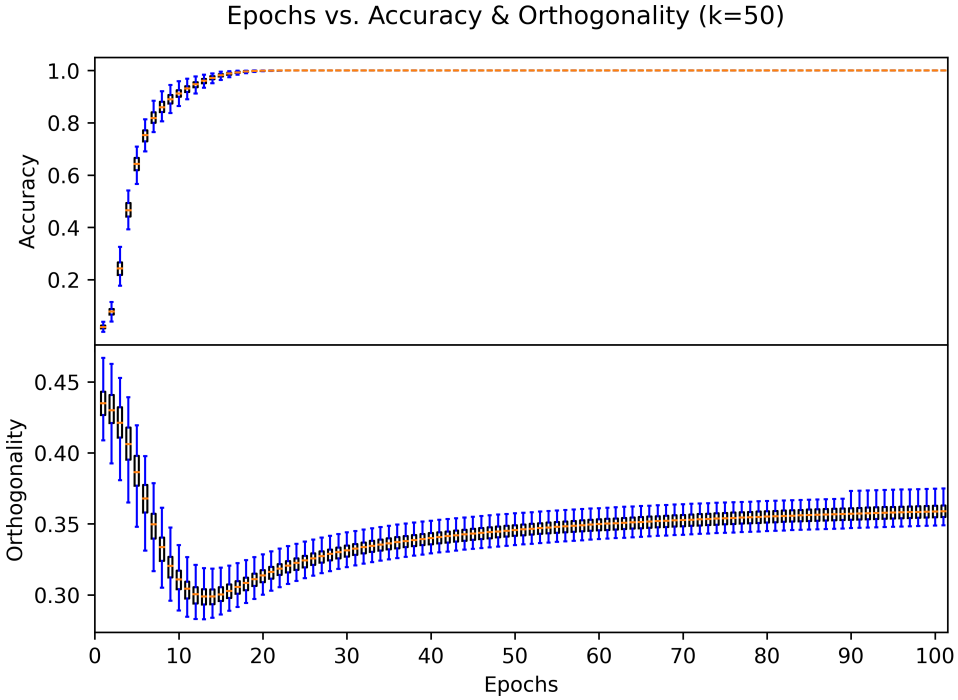
In the remaining development, the dataset described in section 4.3 is used to train the NN. Using the results in Figure 4, we fix $k = 6$ such that the model can achieve high accuracy, while the feature functions remain fairly orthogonal. As shown in Figure 5a, the accuracy increases steadily from 1.98%[1] to 80.5%. On the other hand, orthogonality starts from about 89%, reaches a minimum average of 83.9% after 6 epochs, and increases steadily afterwards, exceeding the initial orthogonality (around 97.4% after 100 epochs). Since few feature functions are randomly chosen out of a high-dimensional space, the high initial orthogonality of an untrained NN is expected[2]. In the beginning, the NN emphasizes on extracting meaningful features at the expense of the orthogonality to rapidly increase accuracy. After a few epochs, in order to reduce the loss further under the tight constraint, the NN encourages the features to contain more unique information to capture broader features of the dataset. As the NN converges, the features are highly orthogonal ($s = 96.1\%$ after 100 epochs).

[1] Since there are 50 output classes, an untrained network randomly guesses the output, yielding a theoretical accuracy of $1/50 = 2\%$

[2] See section 3.3 and Figure 2.



(a) Number of Feature Functions $k = 6$



(b) Number of Feature Functions $k = 50$

Figure 5: We train the model with different number of feature functions for 100 epochs with different random weight initialization. The accuracy and orthogonality of features are measured after every epoch.

Suppose we loosen the constrain and perform the same experiment on $k = 50$. As shown in Figure 5b, the NN converges significantly faster and achieves a higher accuracy, but the orthogonality remains low and does not even exceed its initial value.

5.2 Orthogonality Regulator & Pre-Orthogonalization

Recall that in section 3.2, we show that during the training process, the NN alternates between updating the features and the weights of the last hidden layer while fixing one another, until they align with each other after several iterations. If the features are more orthogonal, the projections are more accurate and we can allow a more aggressive learning rate. Therefore, we encourage the features to be more orthogonal by adding a regulator to the loss function:

$$L'(y, s) = L(y, s) + \lambda \cdot (1 - s) \tag{12}$$

where $L(y, s)$ is the original loss function from (11), λ is the regulator weight, and s is the orthogonality measurement from (9). In other words, we add a penalty on the redundancy of features to the loss function. Here, a high λ encourages the gradient descent to emphasize on orthogonalizing the features; a low λ encourages the gradient descent to emphasize on reducing the cross-entropy loss. Therefore, we experiment on the same dataset with a regulator with varying weights.

As shown in Figure 6, adding the regulator enables the NN to converge faster, where the improvement is maximized at $\lambda = 0.5$. For $\lambda \geq 0.1$, the algorithm with a regulator added converges slower in the first few epochs. This corresponds to Figure 5a, where the NN focuses on extracting information instead of orthogonalizing features in the beginning. Therefore, the regulator does not become helpful until the features represent some useful information, and encouraging them to be more orthogonal becomes more beneficial afterwards under the tight constraint. At $\lambda = 0.5$, the regulator outperforms the approach without the regulator, yielding nearly a 1.5% improvement after 12 epochs. In addition, a heavier weight on the regulator

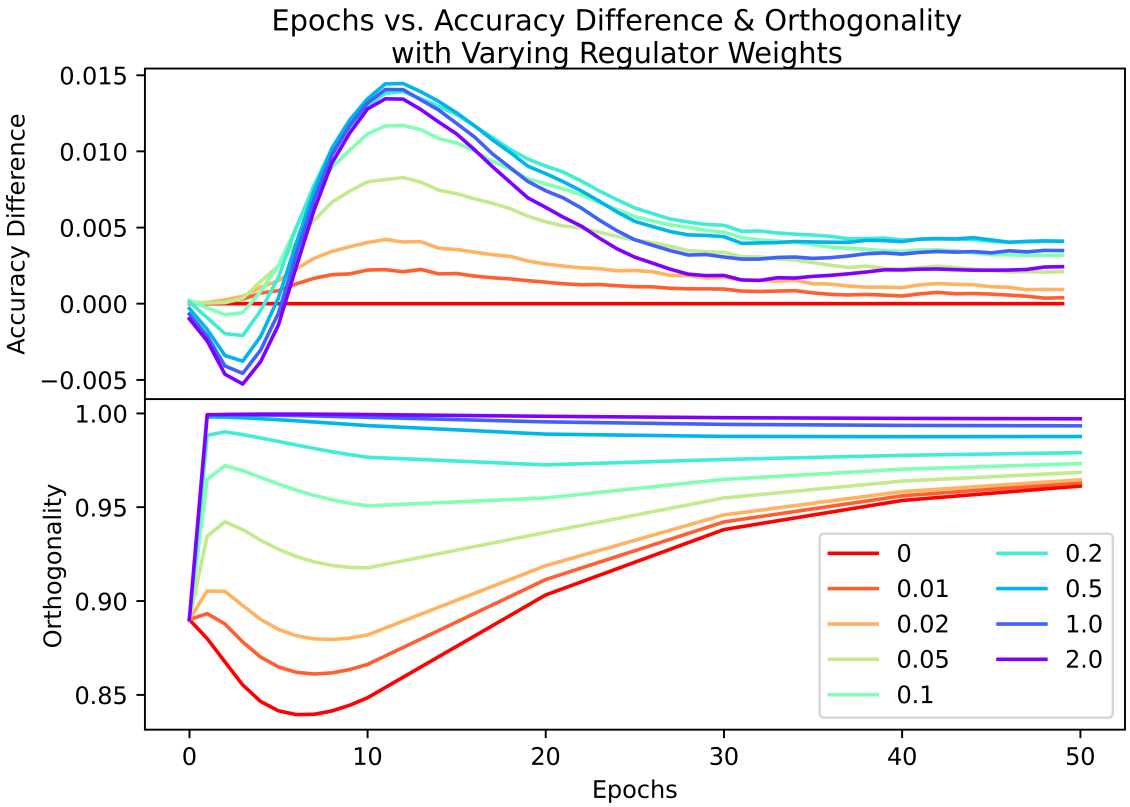


Figure 6: Orthogonalizing regulators were added to the loss function with varying weights. The networks are trained for 50 epochs, and the average results are compared to the one without a regulator ($\lambda = 0$, shown in red). As expected, a heavier weight on the regulator leads to a higher orthogonality. Accuracy is evaluated after every epoch, while the orthogonality is measured every after every epoch only for the first 10 epochs, and every 10 epochs afterwards (same for figures 7-7[TODO: Change this]).

encourages the features to be more orthogonal, and a high orthogonality is maintained throughout training.

Recall that throughout the NN training process, the orthogonality first decreases then increases after a few epochs, eventually surpassing initial orthogonality, as shown in Figure 5a. We explore whether starting from an orthogonal set of features can accelerate the convergence. We start from a randomly initialized NN and back-propagate *only* $(1 - s)$ for multiple iterations. During this phase, the learning rate is reduced to 0.01 to enable a careful selection of orthogonal features. Since pre-orthogonalizing does not require labels, it can be done *offline* without additional overhead cost. As shown in Figure 7, we orthogonalize the features prior to training

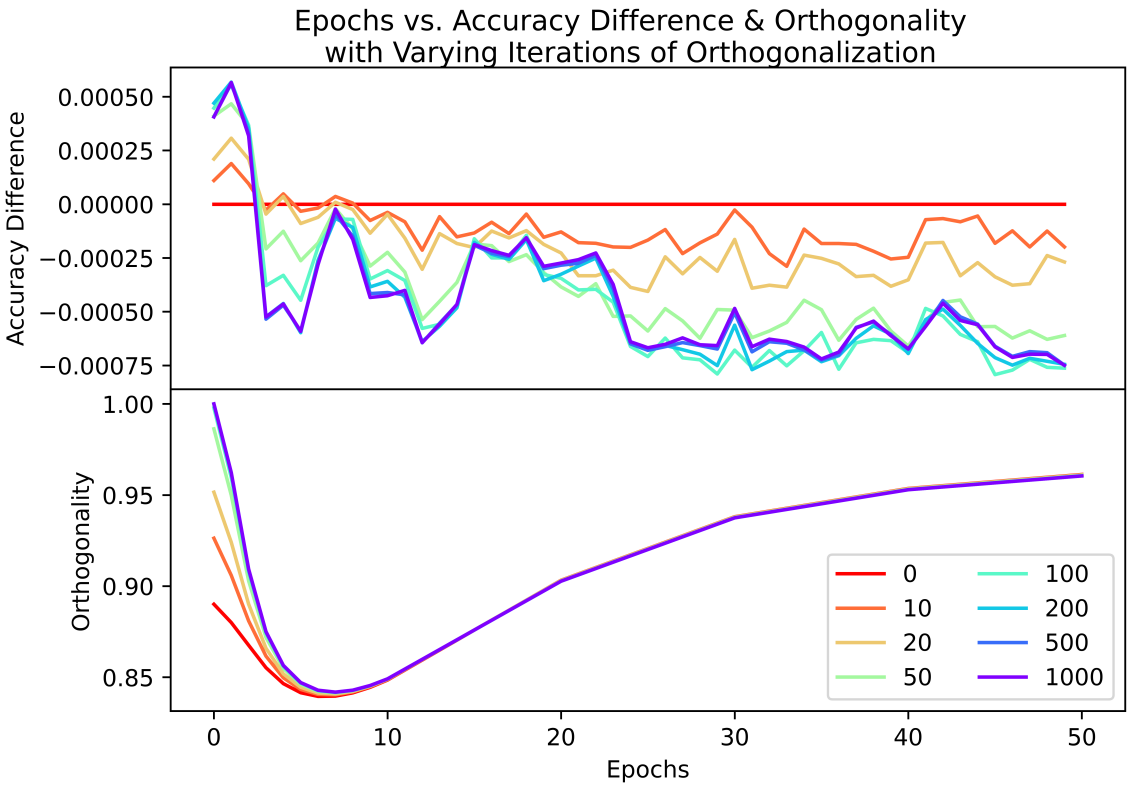


Figure 7: We back-propagate $(1-s)$ for varying number of iterations to orthogonalize the features. More iterations correspond to higher initial orthogonality, but does not make a significant difference during training.

with varying numbers of iterations. As expected, more iterations lead to higher initial orthogonality ($s = 99.78\%$ at 100 iterations). Nevertheless, the orthogonality drops to nearly the same minimum in the first few epochs and increases afterwards, and does not lead to a significant change in accuracy.

5.3 Two-Phase Training

Lastly, we propose a two-phase training by combining both ideas: we first orthogonalize the features, and use a regulator to maintain its high orthogonality throughout training. We consider the combination of parameters from the two previous experiments, and similarly compute its orthogonality and accuracy. In Figure 8, the average accuracy of each combination is compared to that of the original approach (without pre-orthogonalizing or regulator). Observe that the accuracy difference is dominated

by the regulator, and the impact of pre-orthogonalizing the features is minimum, if not slightly worse in some cases. In addition, as shown in Figure 9, pre-orthogonalization increases the initial orthogonality but has minimum impact afterwards. In combination with the regulators, if the network starts training from an orthogonal set of features, a higher weight encourages the orthogonality to remain high.

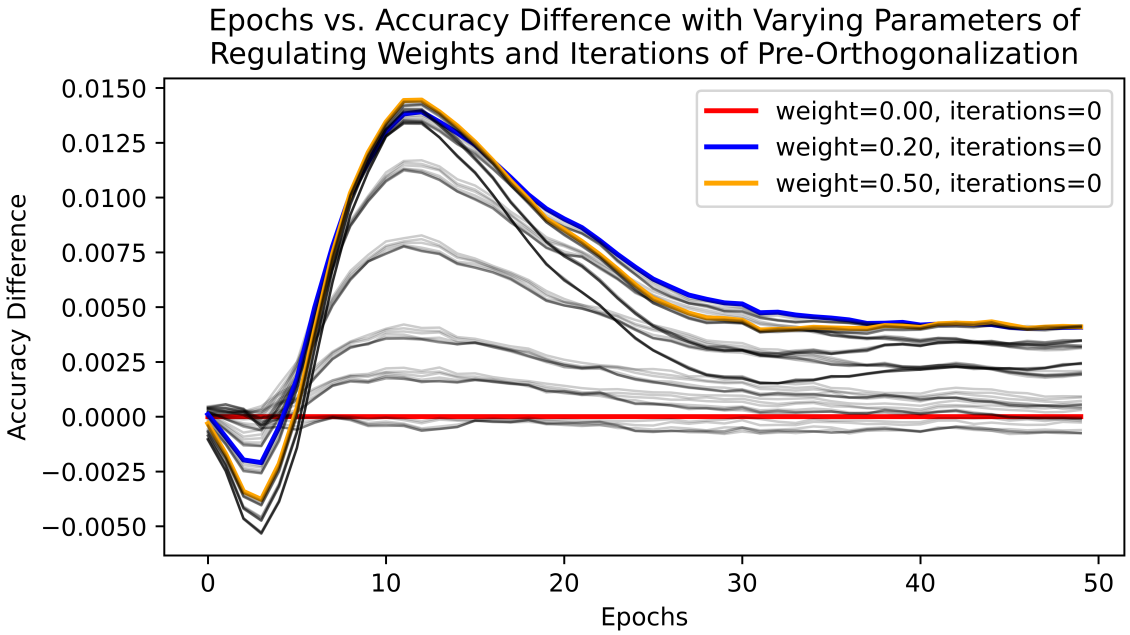


Figure 8: We experiment different combinations of parameters, and compare the results with the standard gradient descent approach. Since the accuracy difference is heavily impacted by the regulator weight, we only highlight the baseline and the two set of parameters which yield the highest accuracy after the first few epochs.

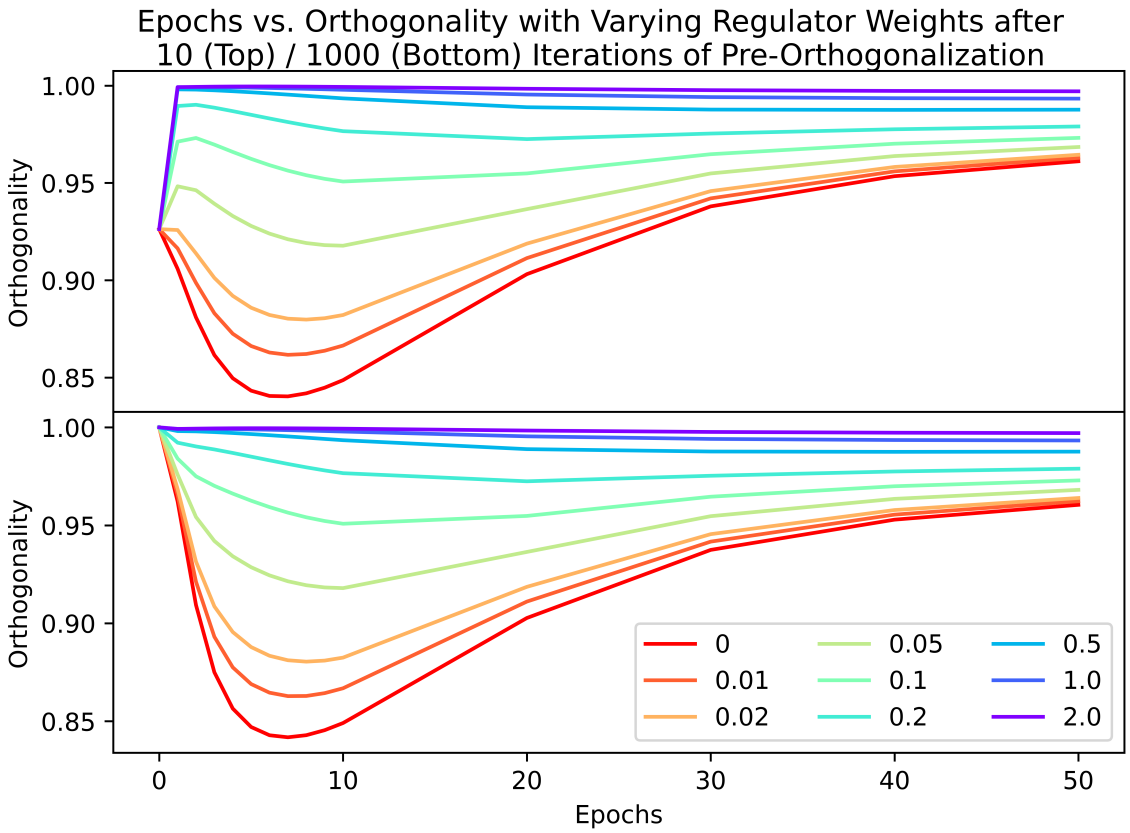


Figure 9: We compute the orthogonality of all combination of parameters, but we only show the results after 10 (Top) and 1000 (Bottom) iterations of pre-orthogonalization for brevity. More iterations enable the network to start with more orthogonal feature functions, but some weight on the regulator is required to maintain high orthogonality.

Chapter 6

Discussion

6.1 Effects on Orthogonalizing Feature Functions

This paper develops an approach to accelerate the convergence of neural networks by orthogonalizing feature functions. **Chapter 2** gives an overview of the universal feature functions extracted in high-dimensional space, and methods to make the neural networks more robust. Some main challenges include stagnation near saddle points, and the infeasibility to replace gradient descent with the more efficient second-order optimization. **Chapter 3** introduces the HGR maximal correlation to quantify the dependence between inputs and outputs, and explains how information is computed and stored during the training process. The orthogonality of features reflects their uniqueness, and plays a significant role in feature projection. **Chapter 4** details the experiment setup, including the mixture Gaussian dataset and the neural network structure. It verifies the claim that more feature functions correspond to higher accuracy but lower orthogonality. **Chapter 5** proposes three methods to increase orthogonality, including adding a regulator to penalize redundancy of features, pre-orthogonalizing features prior to training, and combining both ideas. Results indicate that the first method outperforms the standard gradient descent, while the second method has minimum impact on the accuracy. When both methods are combined, the accuracy difference is dominated by the regulator.

Orthogonal features have two desirable properties, which are particularly useful

for searching for an optimal point in the neural network. First, as shown in section 3.2, if they are more orthogonal, projecting onto the linear subspace of the set of feature functions is closer to summing up the projections onto each feature. This enables a more aggressive learning rate, and can drastically reduce computation cost. Second, recall that the main challenge of taking advantage of the superior properties of the second-order optimization is the cost of computing and storing the Hessian matrix. As the feature functions become more orthogonal, the Hessian matrix converges to an identity matrix, making gradient descent an approximation to Newton's method in optimization[1]. This enables us to take advantage of the superior properties of the second-order optimization without directly computing the Hessian matrix.

6.2 Future Work

While the regulator successfully makes the feature functions more orthogonal, it can have adversarial effects on the model accuracy in some cases. As shown in Figure 5, the gradient descent method to increase accuracy indeed reduces orthogonality. Therefore, adding a heavy weight on the orthogonality regulator in our experiments can negatively impact the accuracy in the beginning, which motivates us to adjust the weight on the regulator accordingly. Nevertheless, we are unable to locate where the orthogonality reaches the minimum, which implies that the adjustments have to be made on the fly. In addition, this paper considers the mixture Gaussian distribution due to its flexibility to adjust the complexity and parameters. It will be helpful to consider more complicated high-dimensional data, such as MNIST and CIFAR, to examine if they share a similar pattern. While we do not expect improvements on all datasets, we aim to generalize and explain on what type of datasets our algorithm exhibits superior convergence. Deeper NN should also be considered, although their properties are significantly harder to analyze due to an increasing number of non-linear transformations.

[1]Gradient descent updates the solution by $x_{k+1} = x_k - \alpha \cdot \nabla f(x)$, while Newton's method in optimization updates the solution by $x_{k+1} = x_k - \alpha \cdot \frac{\nabla f(x)}{\nabla^2 f(x)}$, where α is the learning rate. As we minimize the loss function, if features are highly orthogonal, then $\nabla^2 f(x) \approx \mathbf{I}$.

Appendix A

Gradient of the deep neural network

Suppose the NN described in Figure 1 is trained on one sample (x, y) for one iteration. We compute its gradient with respect to each layer:

1. w.r.t. the softmax output:
$$\frac{\partial L}{\partial s_i} = -\frac{y_i}{s_i} \tag{13}$$

2. w.r.t. the output layer:
$$\frac{\partial L}{\partial z_j} = \sum_{i=1}^{c} \frac{\partial L}{\partial s_i}\frac{\partial s_i}{\partial z_j} = -\sum_{i=1}^{c} \frac{y_i}{s_i} \cdot s_i(\mathbb{1}_{i=j} - s_j) = \sum_{i=1}^{c} y_i s_j - \sum_{i=1}^{c} y_i \cdot \mathbb{1}_{i=j} = s_j - y_j \tag{14}$$
where we get the last equality because each sample belongs to exactly one class. We can express the above in matrix form: $\frac{\partial L}{\partial z} = s - y$.

3. w.r.t. the last hidden layer:
$$\frac{\partial L}{\partial w_{ij}} = \sum_{l=1}^{c} \frac{\partial L}{\partial z_l} \cdot \frac{\partial z_l}{\partial w_{ij}} = \frac{\partial L}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_{ij}} = (s_i - y_i) \cdot f_j \tag{15}$$
where we get the second to last equality because $\frac{\partial z_l}{\partial w_{ij}} = 0$ if $l \neq i$, and the last equality because $z_i = \sum_{j=1}^{k} w_{ij} f_j + b_i$. We can express the above in matrix form $\frac{\partial L}{\partial w_j} = (s - y) \cdot f_j$, where w_j is the j'th column vector of w.

Equation 15 reveals how gradient descent increases the model accuracy. During each iteration, w_{ij} is adjusted by the gradient of the loss with respect to itself, or more formally,

$$w'_{ij} = w_{ij} - \alpha \cdot \frac{\partial L}{\partial w_{ij}} \tag{16}$$

where α is the learning rate. Consider a sample with label t, such that $y_i = \mathbb{1}_{i=t}$ in one-hot representation. During the forward propagation, the model receives the sample and passes it through the last hidden layer and the output layer, forming f and s, respectively[1]. Since $s_t - y_t < 0$, we get that $\frac{\partial L}{\partial w_{tj}} < 0$ if $f_j > 0$, and $\frac{\partial L}{\partial w_{tj}} > 0$ if $f_j < 0$. Based on equation 16, the adjustments to w_{tj} is in the same direction as f_j, and its magnitude is proportional to how much t is under-predicted. Similarly, for $i \neq t$, the adjustments to w_{ij} is in the opposite direction, and its magnitude is proportional to how much class i is over-predicted.

[1]We assume that $f_j \neq 0$ and $0 < s_i < 1$, otherwise it will simply yield a zero gradient.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[3] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[5] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[6] L. Zheng. Understanding the power of neural networks, 2022.

[7] S.-L. Huang, X. Xu, L. Zheng, and G. W. Wornell. An information theoretic interpretation to deep neural networks. In *2019 IEEE international symposium on information theory (ISIT)*, pages 1984–1988. IEEE, 2019.

[8] S.-L. Huang, A. Makur, G. W. Wornell, and L. Zheng. On universal features for high-dimensional learning and inference. *arXiv preprint arXiv:1911.09105*, 2019.

[9] N. Bacanin, T. Bezdan, E. Tuba, I. Strumberger, and M. Tuba. Optimizing convolutional neural network hyperparameters by enhanced swarm intelligence metaheuristics. *Algorithms*, 13(3):67, 2020.

[10] L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[11] S. Karsoliya. Approximating number of hidden layer neurons in multiple hidden layer bpnn architecture. *International Journal of Engineering Trends and Technology*, 3(6):714–717, 2012.

[12] Z. Boger and H. Guterman. Knowledge extraction from artificial neural network models. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 4, pages 3030–3035 vol.4, 1997.

[13] S. Kazuhiro. A two phase method for determining the number of neurons in the hidden layer of a 3-layer neural network. In *Proceedings of SICE Annual Conference 2010*, pages 238–242, 2010.

[14] G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[16] A. J. Bray and D. S. Dean. Statistics of critical points of gaussian fields on large-dimensional spaces. *Physical review letters*, 98(15):150201, 2007.

[17] G. Swirszcz, W. M. Czarnecki, and R. Pascanu. Local minima in training of neural networks. *arXiv preprint arXiv:1611.06310*, 2016.

[18] P. Xu, F. Roosta, and M. W. Mahoney. Second-order optimization for non-convex machine learning: An empirical study. In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pages 199–207. SIAM, 2020.

[19] L. Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[20] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[21] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[22] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[23] D. Xie, J. Xiong, and S. Pu. All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6176–6185, 2017.

[24] W. Hu, L. Xiao, and J. Pennington. Provable benefit of orthogonal initialization in optimizing deep linear networks. *arXiv preprint arXiv:2001.05992*, 2020.

[25] Tibshirani. R. Newton's method. Carnegie Mellon University, 10-725, 2019.

[26] R. Anil, V. Gupta, T. Koren, K. Regan, and Y. Singer. Second order optimization made practical. *arXiv preprint arXiv:2002.09018*, 2020.

[27] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.

[28] S.-L. Huang and X. Xu. On the sample complexity of hgr maximal correlation functions for large datasets. *IEEE Transactions on Information Theory*, 67(3):1951–1980, 2020.

[29] J. Lee. *Maximal Correlation Feature Selection and Suppression With Applications*. PhD thesis, Massachusetts Institute of Technology, 2021.

[30] X. Xu and S.-L. Huang. Maximal correlation regression. *IEEE Access*, 8:26591–26601, 2020.