The Influence of the Orthogonality of Feature Functions on Artificial Neural Networks

Lukas Florian Münzel

Under the direction of

Prof. Lizhong Zheng
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Dr. Erixhen Sula
Postdoctoral researcher
Massachusetts Institute of Technology

Research Science Institute
August 2, 2021

Abstract

Conventional methods for adjusting the weights of artificial neural networks only consider the output layer when calculating the loss. Our method also takes into account information from the functions generating the outputs of the neurons in the second to last layer, which are called *feature functions*. Specifically, we measure how *orthogonal* the feature functions are using a *covariance measure*. We incorporate a term in the loss to explicitly reward orthogonal feature functions. We were then able to generate nearly orthogonal feature functions without compromising on accuracy. Due to the convenient mathematical properties of orthogonal feature functions, we will in a future next step be able to find the optimal choice of weights between the second to last and the output layer with low computational cost.

Summary

Artificial neural networks try to infer features from inputs, analogous to how neurons in the human brain conclude that a dog is being seen from the light reflected by that dog hitting the eyes. Conventional methods for improving the performance of these artificial neural networks only consider the final output of the network and then adjust the entire network to improve that final output. In order to improve, our method also analyses an intermediate layer of the neural network. Specifically, we measure the *orthogonality* of the functions of the neurons in the second-to-last last layer, which gives us information about the independence of these functions. We were then able to increase the orthogonality significantly without impacting the performance of the neural network. Due to some convenient mathematical properties, this will make parts of the adjustment of the neural network to maximize its performance fast and remarkably accurate.

1 Introduction

The concepts of artificial intelligence and machine learning have fascinated the scientific community and authors alike since as early as the 1950s [1]. With exponentially growing computational resources, these early dreams started to become feasible in the 1980s and especially in the 21st century. Now, state of the art methods can be used from recommending YouTube videos to detecting breast cancer [2, 3].

Deep learning has proven itself to be one of the most fruitful techniques in the field of machine learning. Inspired by the way the neurons in our brain interact, deep learning models are comprised of multiple *layers* of simulated artificial neurons. The neurons in the first layer directly take some input, say a picture, weigh it and pass it on to the next layer. The neurons in that layer can, for example, learn how to detect edges. The neurons in the following layer can then recognize more complex shapes like circles based on a combination of the outputs of the neurons recognizing edges. This process is repeated until the last layer is reached, where a neuron could predict the probability of the given picture being a dog.

To evaluate the performance of the neural network, the *loss* is calculated. The loss measures how far the output of that last layer is from the intended output. The network is then adjusted to minimize that loss, and the process is repeated.

Our approach, however, can also take information from the second-to-last layer into account. We call the functions generating the output of these neurons in the second-to-last layer *feature functions*. Specifically, we add a measure of the redundancy of information between the feature functions to the loss. This rewards the neural network for making the feature functions independent, for making them *orthogonal*. If the feature functions were orthogonal, the optimal choice of weights could be computed easily due to some convenient mathematical properties discussed in subsection 2 of section 2. How close to being orthogonal two feature functions are orthogonal is measured using the *covariance*. Calculating this

covariance for training neural networks is not an entirely new idea, but it was dismissed by Yann LeCun et al. due to its high computational cost.

Understanding and influencing the effect the covariance between the feature functions has on the training process and using its mathematical properties could help make that training process faster and its results more accurate. Thus we have decided to investigate the influence of this until now largely dismissed method.

Content

In section 2, we discuss the basic function of artificial neural networks and the mathematical background required to understand the relevance of the orthogonality of the feature functions. Section 3 discusses the used data and methods to obtain orthogonality. The drawbacks of orthogonalizing the feature functions and the causes of changes in covariance in regular training processes are discussed in section 4. How future work can build upon our results can be read in section 5. To conclude, section 6 summarizes our results and our gratitude goes to those mentioned in section 7, who made this research possible.

2 Preliminaries

How neural networks work

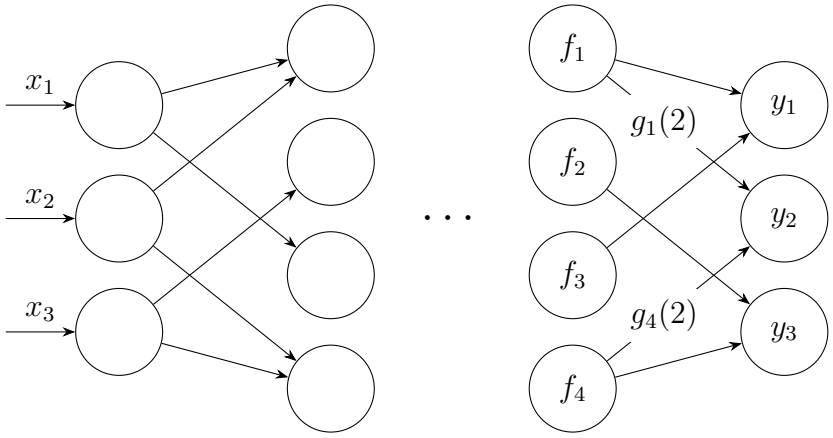

Figure 1: Artificial neural network with feature functions and weights

Figure 1 demonstrates the basic structure of an artificial neural network. The vertical stacks of neurons are called *layers*, $f_1, f_2, f_3 \cdots f_k$ are called *feature functions* and $g_1(y), g_2(y) \cdots g_k(y)$ are called *weights*. The weight $g_i(j)$ connects the ith feature function with the jth output neuron. Using these weights and the ones in the layers before, the neural network calculates outputs $y_1, y_2 \cdots y_m$ based on the inputs $x_1, x_2, \cdots x_n$.

The value of say y_2 is given by the sum of all feature functions scaled by the respective weights, meaning that $y_2 = \sum_{i=1}^{k} g_i(2) f_i(x_1, x_2, \cdots)$. To convert it into a probability, y_2 is then normalized. Other layers use different normalizations for different purposes. But not only are all other outputs y_i calculated similarly using different weights, but the feature functions themselves are also calculated in the same way as y_2, namely using a weighted sum of the outputs of neurons in the layer before. This process of taking a weighted sum of the outputs of the layer before is repeated until the input layer is reached.

The neural network then calculates its predictions for thousands of samples, and the *loss* is calculated, which measures how different the predictions are from their intended values.

Stochastic gradient descent or more sophisticated methods based on similar ideas iteratively calculate how to adjust the weights to minimize the loss [4].

Interpreting functions as vectors

Given some function $f(x)$, there exists a corresponding vector with the function's values at different points as its components.

$$\vec{f} = \begin{bmatrix} f(x=1) \\ f(x=2) \\ f(x=3) \\ \vdots \end{bmatrix}$$

Instead of only considering natural numbers, this vector could also have the function values at every real point as its components. For these kinds of vectors, there exists a mathematically rigorous way of defining vector operations analogue to these of Euclidean space, such as vector addition or the dot product.

For example, consider functions $f(x)$ and $g(x)$, defined for $0 \leq x \leq 1$.

We can define $f(x) \cdot g(x)$ as

$$\int_0^1 f(x)g(x)dx = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f\left(\frac{i}{n}\right) g\left(\frac{i}{n}\right)$$

For the corresponding vectors in Euclidean space with a finite number of components and sampling uniformly from the functions, this corresponds to

$$\frac{1}{n}\vec{f} \cdot \vec{g}$$

This means that the dot product of two functions is in the limit equal to the dot product of two Euclidean vectors with the values of these functions as components, normalized by $\frac{1}{n}$. Applying the insights gained from Euclidean space to this kind of vector space proves itself useful, as discussed in the following subsection.

Projection onto the target function

Let the target function $T(x_1, x_2 \cdots x_n)$ be defined as the function mapping an input $x_1, x_2 \cdots x_n$ to the respective desired value of y_i for some specific i. Our goal thus is to make neural network's output y_i as a function of inputs $x_1, x_2 \cdots x_n$ match the target function by adjusting the weights $g_1(i), g_2(i) \cdots g_k(i)$. The feature functions and the target function can now be interpreted as vectors, as described in subsection 2. Therefore, minimizing the loss corresponds to the minimizing the squared Euclidean distance between the target function and the sum of the feature functions scaled by the weights. The loss therefore is

$$||\vec{T} - \sum_i g_i \vec{f_i}||^2$$

The set of all vectors which can be expressed as $\sum_i g_i \vec{f_i}$ is called the *span* of the feature functions. Thus, the problem is to find the vector in the span of the feature functions with the minimal squared Euclidean distance to the target function. This can be interpreted geometrically as *projecting* the target function onto the span of the feature functions.

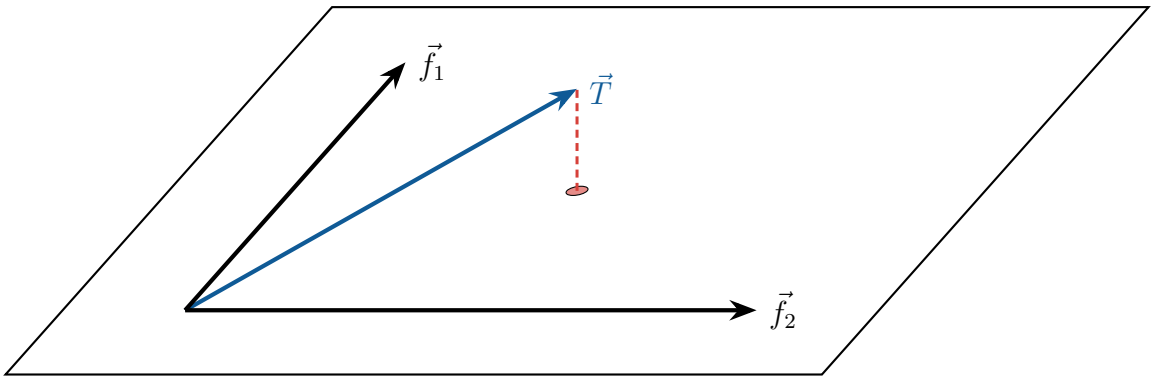

Figure 2: Projection of $\vec{T}$ on the span of $\vec{f_1}$ and $\vec{f_2}$

In Figure 2, the red point represents the projection of $\vec{T}$ onto the span of the two feature functions $\vec{f_1}$ and $\vec{f_2}$. The plane represents the span of $\vec{f_1}$ and $\vec{f_2}$.

Calculating the projection of $\vec{T}$ onto the span of $\vec{f_1}$ and $\vec{f_2}$ would be easy if that projection could be found by simply summing the projections of $\vec{T}$ onto the individual feature functions.

If all the feature functions are orthogonal to each other, the projection can indeed be found in this manner. Two feature functions are orthogonal if their inner product $\langle \vec{f_1}, \vec{f_2} \rangle$ is zero. In $\mathbb{R}^2$, this would mean that they have a right angle between them and that their dot product is zero. The additivity of the projection for orthogonal feature functions is shown in Figure 3.

The projection can, however, not be reduced to projecting onto the individual feature functions if the feature functions are not orthogonal, as visualized in Figure 4. Conventional methods for minimizing the loss avoid this problem by computing the solution iteratively. The size of the steps in these iterations is called *learning rate*. Having easily computed the projection onto the orthogonal feature functions, however, we find the choice of weights $g_1(i), g_2(i) \cdots g_k(i)$ that results in the smallest possible loss without any iterative processes.

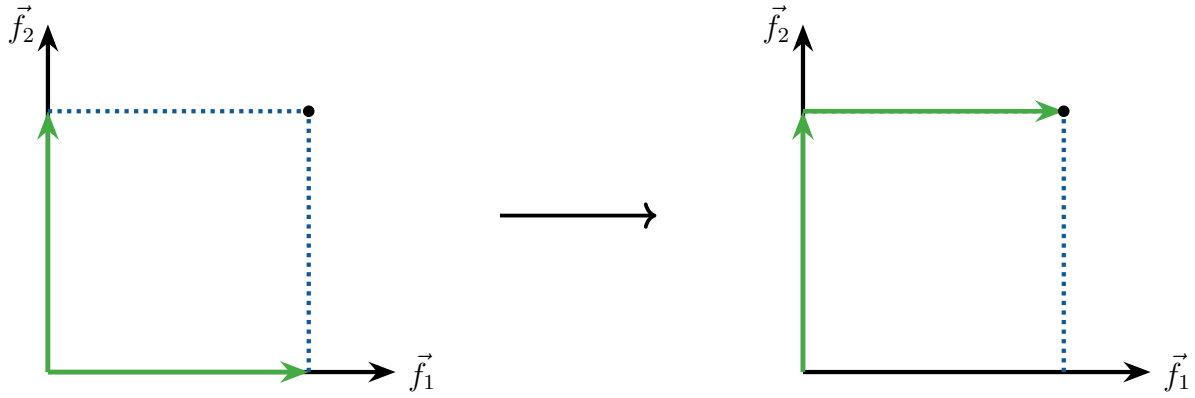

Figure 3: Additivity of projection onto orthogonal feature functions

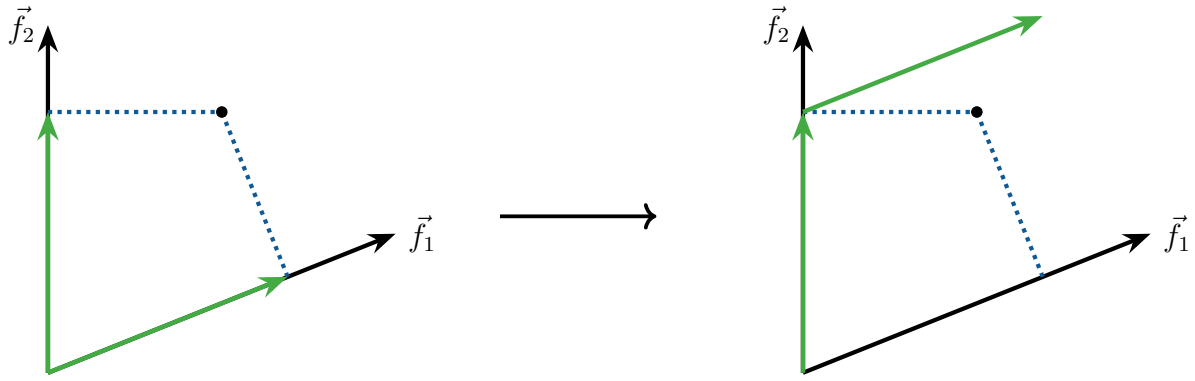

Figure 4: Non-additivity of projection onto non-orthogonal feature functions

3 Methods

Data generation

The data set used consists of points in clusters of multivariate Gaussian distributions. Each cluster has some class associated with it, which the neural network is supposed to predict. Using such a data set has the advantage of reducing training times while still being representative of many more complex data sets. Furthermore, it enabled us to modify the complexity of the input the neural network has to predict the classes of. In Figure 5, the class corresponds to the color of a point, and the background color represents the class the neural network predicted.

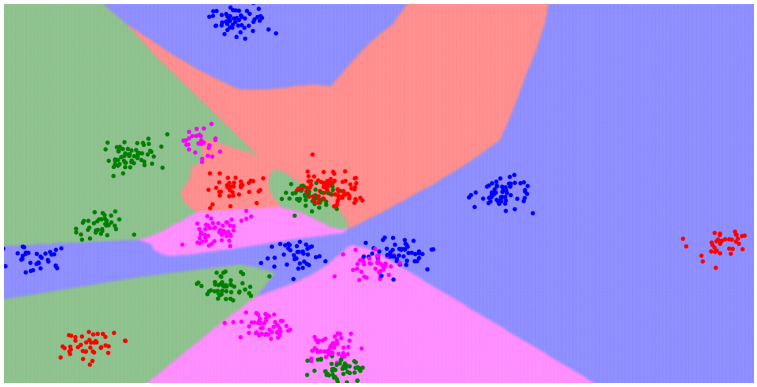

Figure 5: Example of the generated input data with four classes, two dimensions and five clusters per class. The color of the background represents the class the neural network predicted.

Measuring orthogonality

Making the feature functions orthogonal would greatly simplify finding the optimal choice of weights. Since making the feature functions orthogonal is rather hard, a measure of how close to orthogonal the feature functions are is needed. Say the dot product for f_i and f_j is computed as described in subsection 2 of section 2, given by the following, where x_s is the sth randomly sampled input

$$\frac{1}{n_{samples}} \sum_{s=1}^{n} f_i(x_s) \cdot f_j(x_s)$$

The calculation of the value that will be used for all further analysis of orthogonality is called *covariance* and it is calculated by normalizing the above dot product by subtracting the respective expected values from the feature functions before multiplying them. Thus, the *covariance* between feature functions f_i and f_j is given by

$$k_{i,j} = \frac{1}{n_{samples}} \sum_{s=1}^{n} [f_i(x_s) - \mathbb{E}(f_i)] \cdot [f_j(x_s) - \mathbb{E}(f_j)]$$

For every pair of two different vectors to be orthogonal means that $k_{i,j}$ is zero for $i \neq j$. To measure the degree to which this is the case, the following is computed

$$\frac{\sum_i k_{i,i}^2}{\sum_{i,j} k_{i,j}^2}$$

Thus, this measure of covariance is the ratio between the total covariance of pairs of the same feature functions and the total covariance of all pairs of feature functions. The covariance measure ranges from zero to one, where a value of one implies that all pairs of two different feature functions are orthogonal.

The main drawback of this method is that it uses $\mathcal{O}(k^2)$ operations per sample, where k is the number of feature functions. That is the case since there are k^2 pairs of feature functions, and for every one of these pairs, the sum over all samples is calculated.

Measuring the orthogonality in regular training processes

To investigate what causes the orthogonality measure to increase or decrease and why the covariance was measured at the end of regular training processes. To achieve this, hundreds of neural networks were trained with different parameters, such as the number of feature functions. Since the influence of the change of a single parameter on the covariance measure was to be investigated, the neural networks were constructed to only differ from some base case in a single parameter. Because the data was artificially generated, parameters in the input data, namely the number of dimensions, the number of classes and the number of clusters per class, could be adjusted as well.

Subtracting the covariance measure from the loss

Since how close the feature functions are to being orthogonal can now be measured, and since gradient descent minimizes the loss function, the covariance measure can simply be subtracted from the loss. This rewards the neural network for making the feature functions

orthogonal without further effort.

How well the neural networks performs in terms of the covariance measure and the accuracy was then measured for different learning rates and for different levels of the importance of minimizing the regular loss over maximizing the covariance measure. To quantify these levels of importance, we introduce the covariance measure weight w, $0 \leq w \leq 1$.

The loss is then given by

$$\mathcal{L} = (1 - w)\mathcal{L}_{regular} - w\frac{\sum_i k_{i,i}^2}{\sum_{i,j} k_{i,j}^2}$$

4 Results and discussion

Covariance measure and accuracy in regular training

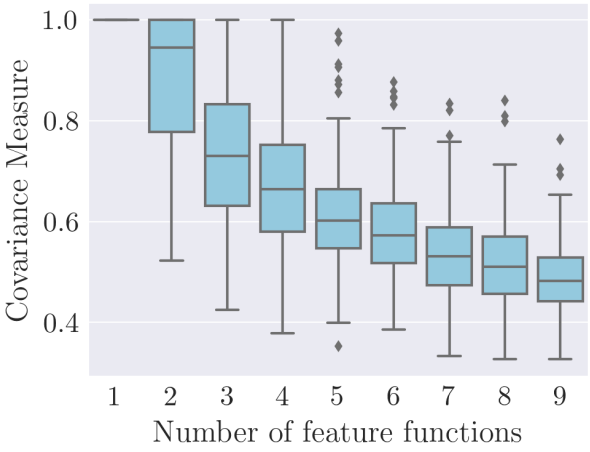

Figure 6: Orthogonality measure for different numbers of feature functions

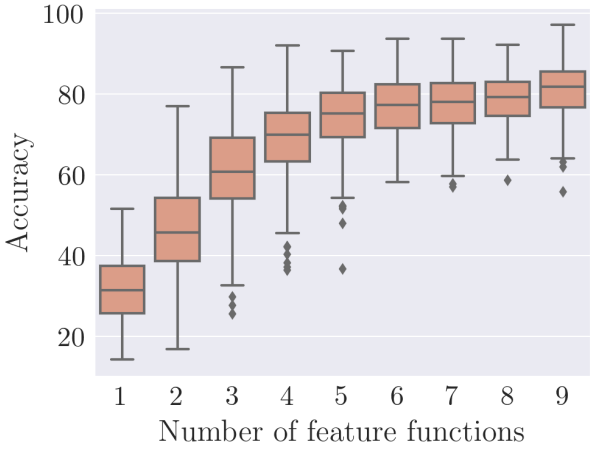

Figure 7: Accuracy for different numbers of feature functions

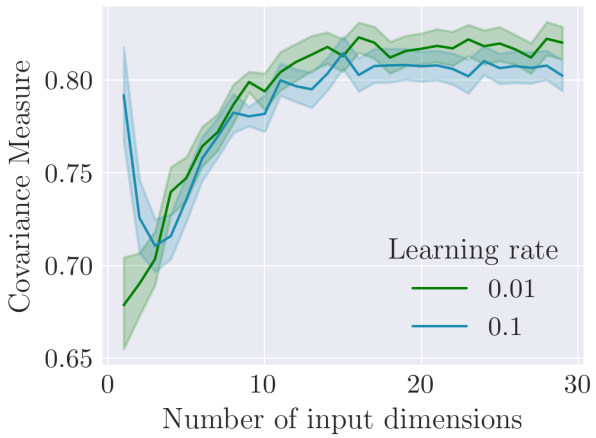

Figure 8: Orthogonality measure for different numbers of dimensions

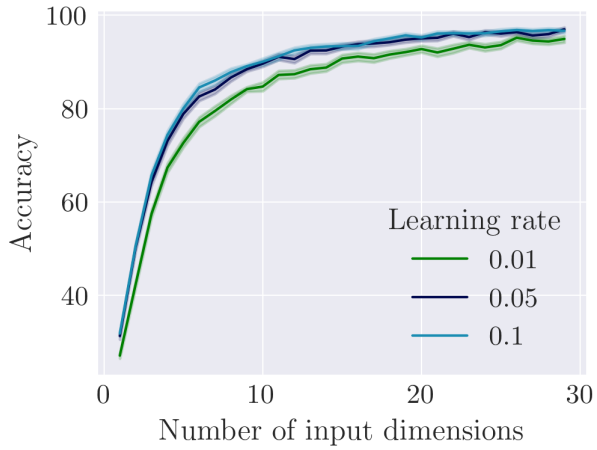

Figure 9: Accuracy for different numbers of dimensions

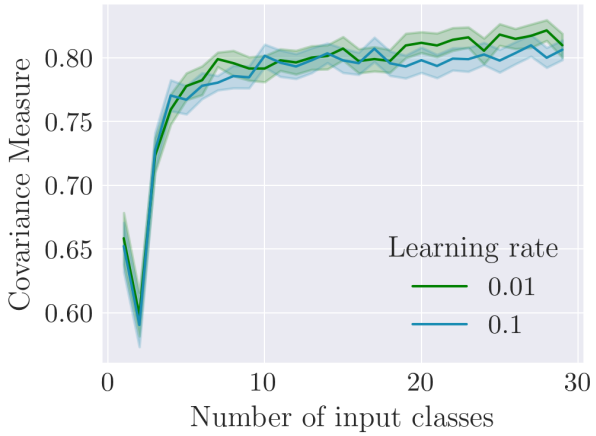

Figure 10: Orthogonality measure for different numbers of classes

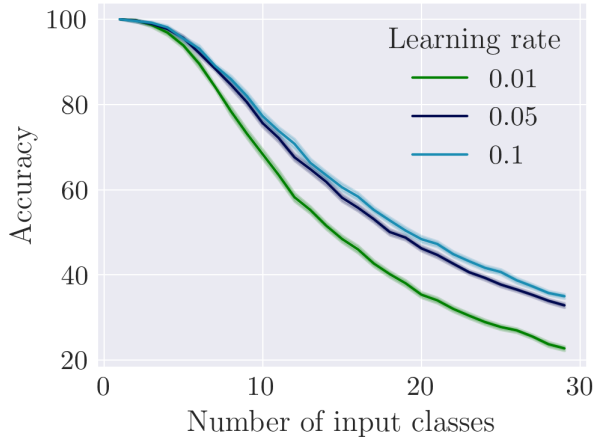

Figure 11: Accuracy for different numbers of classes

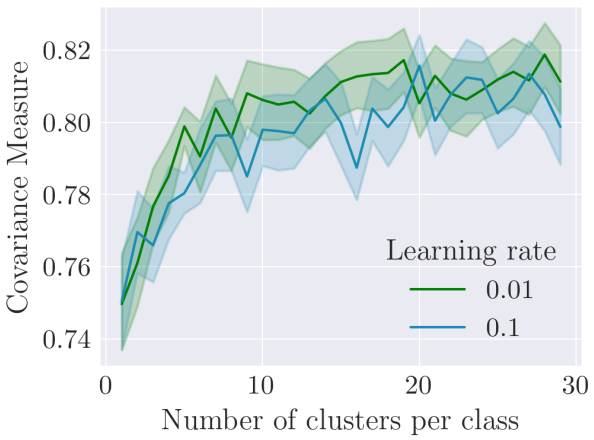

Figure 12: Orthogonality measure for different numbers of clusters

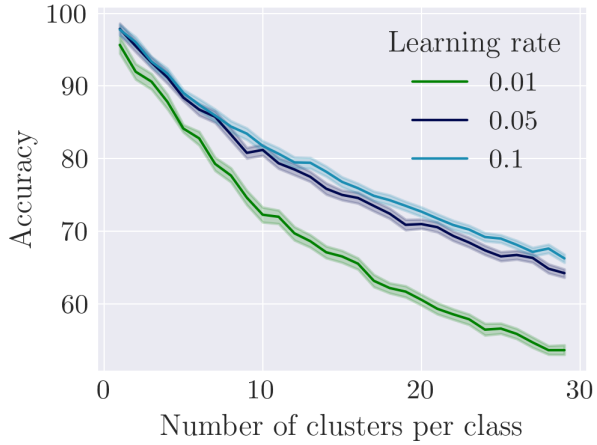

Figure 13: Accuracy for different numbers of clusters

The figures 6, 8, 10 and 12 show a clear trend of the covariance measure correlating with the complexity of the problem given to the neural network. The claim that these effects are indeed the results of a higher problem complexity is supported by figures 7, 11 and 13, because the accuracy is generally expected to correlate with problem difficulty, and the accuracy seems to be inversely correlated with the covariance measure. Our current hypothesis for this behaviour is that as the pressure imposed in the neural network increases, stochastic gradient descent is more and more forced to avoid redundant information, making the feature functions closer to being orthogonal.

Subtracting the covariance measure from the loss

When subtracting the covariance measure from the loss, the covariance measure is expected to increase since stochastic gradient descent is trying to modify the weights such that the loss is minimized. Figure 15 shows that this is indeed the case. For the figure, a learning rate of 0.05 was used together with a covariance measure weight of 10%. This choice of weight resulted in a covariance measure of more than 99%.

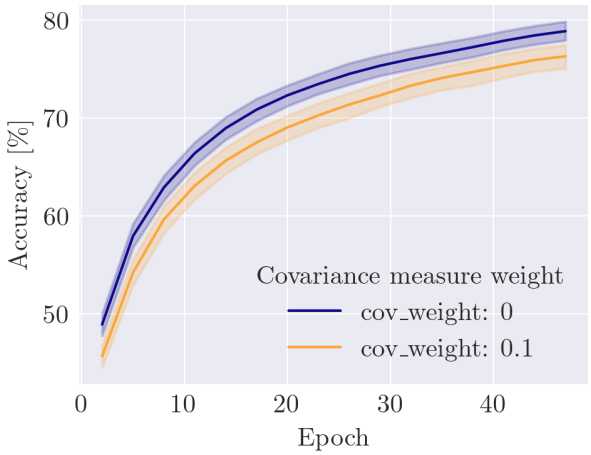

Figure 14: Accuracy over the training process with a learning rate of 0.05

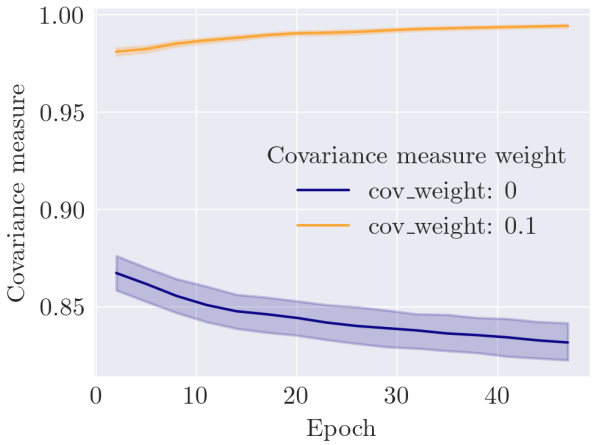

Figure 15: Covariance measure over the training process with a learning rate of 0.05

However, increasing the covariance measure did come at a cost, since the accuracy decreased when including the measure in the calculation of the loss, as shown in Figure 14. The shown intervals in all figures of this section are 95% confidence intervals, and the result are thus statistically significant. Our explanation for the observed behaviour is that stochastic gradient descent has to compromise between choosing the feature functions to be orthogonal and choosing feature functions with which it can minimize the regular loss. However, using the mathematical tools discussed in section 2, we could easily find the optimal choice of weights between the last and the second-to-last layer. This would potentially enable us to close the gap between the accuracies with low computational cost.

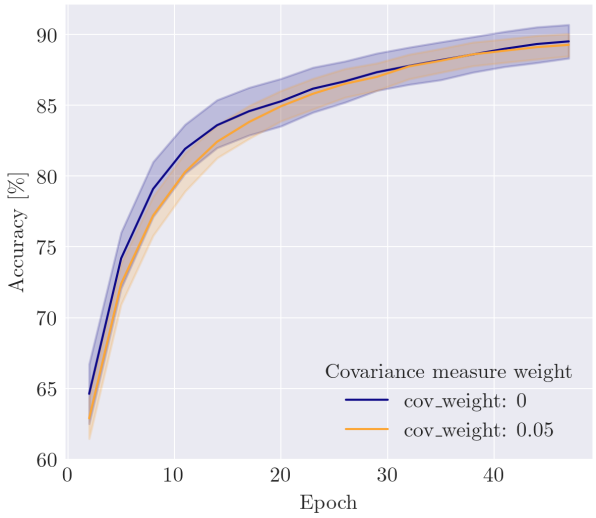

Figure 16: Accuracy over the training process with a learning rate of 0.1

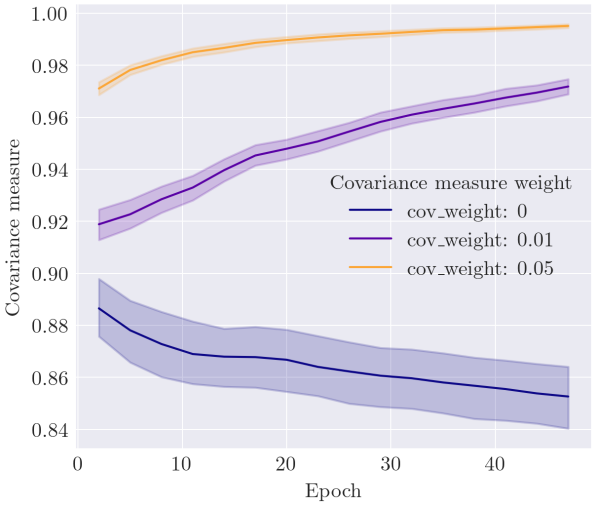

Figure 17: Covariance measure over the training process with a learning rate of 0.1

As shown in Figure 16, this gap can also be closed by an appropriate choice of the covariance measure weight. A value of 5% yields a covariance measure of more than 99% while maintaining close to the same accuracy as the base case with a covariance measure weight of zero. Note that since we could again easily find the optimal solution for the weights between the orthogonal feature functions and the last layer, it would be very well possible that we would be able to reach higher accuracies than with conventional methods.

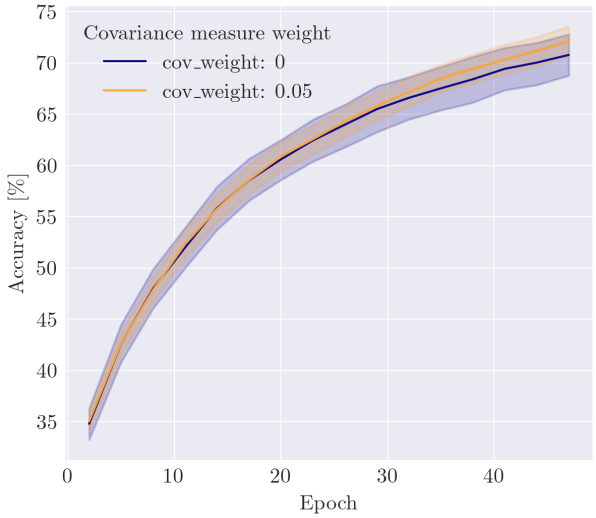

Figure 18: Accuracy over the training process with a learning rate of 0.01

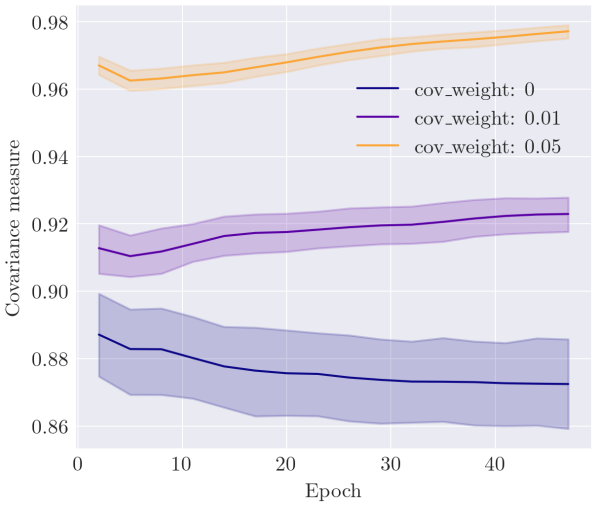

Figure 19: Covariance measure over the training process with a learning rate of 0.01

5 Future Work

A first step in verifying the importance and the validity of our analysis would be to repeat the same methodology for more complex input data sets like, for example MNIST [5], which is used to train handwritten digit recognition.

Then the projection of the target function onto the span of the feature functions can be computed. Investigating the performance of this for the given feature functions optimal method in terms of accuracy and training speed compared to conventional methods could yield highly promising results. An observation requiring further analysis is that even though the covariance measure increases with the complexity of the given input, it is decreasing throughout the entire training process if the neural network is not rewarded for increasing it. Explaining this behaviour could lead to insights useful for the future development of this method.

6 Conclusion

In addition to considering the output layer when calculating the loss, our method also takes into account information from feature functions. Specifically, we measure the *orthogonality* of the feature functions using the covariance. We first showed that these feature functions are closer to being orthogonal for more demanding tasks, indicating that conventional methods for maximizing the accuracy already naturally orthogonalize the feature functions to some degree. Using the loss to explicitly reward the neural network for orthogonalizing the feature functions, we were then able to generate nearly orthogonal feature functions without significantly decreasing accuracy. Furthermore, due to the convenient mathematical properties of orthogonal feature functions, we would in a next step be able to find the optimal choice of weights between the second-to-last and the output layer with low computational cost. Iterations of our work could thus cut down on computational cost and on the other hand potentially increase accuracy, which could potentially save resources and lives.

7 Acknowledgments

First and foremost, I would like to thank my mentors, Professor Lizhong Zheng and Dr. Erixhen Sula. My gratitude also goes to my tutor, Peter Gaydarov and my teaching assistants Lucy Cai, Ishan Khare, Kenneth Choi and Dimitar Chakrov. The Massachusetts Institute of Technology and the Center of Excellence in Education have made this program possible, so I would like to express my gratefulness towards them as well. Furthermore, I thank my fellow Rickoids of the year 2021. I would also like to thank the FBK Bern, the René & Susanne Braginsky Stiftung and the Fritz-Gerber-Stiftung for organizing and financing my participation at the Research Science Institute 2021. Finally, I am grateful as well for the support both my parents have given me throughout this program.

References

[1] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(October), 1950.

[2] I. Sechopoulos, J. Teuwen, and R. Mann. Artificial intelligence for breast cancer detection in mammography and digital breast tomosynthesis: State of the art. *Seminars in Cancer Biology*, 72:214–225, 2021. Precision Medicine in Breast Cancer.

[3] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[4] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[5] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[6] I. Csiszár and P. C. Shields. Information theory and statistics: A tutorial. *Commun. Inf. Theory*, 1(4):417–528, Dec. 2004.