12-2015

Evaluating the Intrinsic Similarity between Neural Networks

Stephen Charles Ashmore
University of Arkansas, Fayetteville

Evaluating the Intrinsic Similarity between Neural Networks.

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Stephen C Ashmore
Northeastern State University
Bachelor of Science in Computer Science, 2012

December 2015
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____

Dr. Michael S. Gashler
Thesis Director

_____

Dr. John Gauch
Committee Member

_____

Dr. Wing Ning Li
Committee Member

Abstract

We present Forward Bipartite Alignment (FBA), a method that aligns the topological structures of two neural networks. Neural networks are considered to be a black box, because neural networks contain complex model surface determined by their weights that combine attributes non-linearly. Two networks that make similar predictions on training data may still generalize differently. FBA enables a diversity of applications, including visualization and canonicalization of neural networks, ensembles, and cross-over between unrelated neural networks in evolutionary optimization. We describe the FBA algorithm, and describe implementations for three applications: genetic algorithms, visualization, and ensembles. We demonstrate FBA's usefulness by comparing a bag of neural networks to a bag of FBA-aligned neural networks. We also show that aligning, and then combining two neural networks has no appreciable loss in accuracy which means that Forward Bipartite Alignment aligns neural networks in a meaningful way.

Acknowledgments

I would like to thank my wife, Jaleesa Ashmore for keeping me going over the past few years. I would like to thank my advisor Dr. Michael Gashler for being extremely supportive, hands-on, and thoughtful. Special thanks to the Computer Science and Computer Engineering department for placement of assistantships and general assistance. I would like to thank Zac and Ezekiel Kindle for their friendship, and help.

Contents

List of Figures

List of Tables

Terms and Definitions

Accuracy The rate of correct predictions made by a model over a data set. Typically this is determined by using an independent test set.

Attribute An attribute describes some part of an instance, this is analogous to a column in the data set. Example: a single pixel value of an image, or a single response to a survey question by one responder.

Classifier A classifier maps from unlabeled instances to classes, like classifying a picture of a hand-written digit to the actual number the picture represents.

Data set Contains a number of instances that follow the same scheme that defines the data set, as in a set of readings from sensors.

Decision Tree A decision tree is a type of machine learning algorithm which uses simple rules based on the attributes of the data set to classify or predict.

Error Rate The inverse of accuracy, see accuracy.

Example Used the same as an instance.

FBA Forward Bipartie Alignment; the algorithm described in this thesis.

Feature See attribute.

Field See attribute

i.i.d. Independent and Identically distributed. This describes the nature of instances in a data set, sometimes this is not the case.

Instance An instance is a single object from which a model will learn. This could be an image, a survey response, etc.

Layer A layer is part of a MLP, there can be many layers. Each layer consists of some number of nodes. The first layer feeds into the second layer, and so forth. The first layer takes as input the instances of the dataset. The last layer gives the output of the network.

MLP Multi Layer Perceptron. The standard type of artificial neural network using layers of neurons or nodes in a feed forward manner.

MLP Weights Each node in an MLP has weights attached to the inputs given to that node. These weights are used to determine what the node outputs on any given input.

Model The learning algorithm produces a model which can be used to predict, classify, etc. analogous to the weights on a MLP.

Neuron See node.

Node A node in a neural network is part of a layer, and accepts input from the previous layer. Each input is multiplied by a weight, and then summed that determines the output of the node.

NN Neural Network. A machine learning algorithm that abstractly mimics the human brain.

RMSE Root Mean Squared Error. This is a different metric to measure the error rate.

Chapter 1

Introduction

Artificial neural networks are function approximating models with signficant utility. Theoretically, they can approximate arbitrary functions [13] [8] and practically, they have been shown to outperform other methods at many real world problems [15]. Because neural networks combine attributes in nonlinear combinations, it is difficult to understand exactly how they work internally.

By contrast, several other popular models including rule-based learners and decision trees produce models that are relatively easy to understand. Decision trees may make their decisions based on entropy [21] [22] or random choices [12] [5]. But all of these metrics base the decision on the value of a single attribute, and are therefore generally easy to examine and understand; as well as compare against other decision trees. Unlike decision trees, neural networks use complex multivariate model surfaces that combine attributes non-linearly. These surfaces may be in in high-dimensional spaces, and the weights may take on any continuous values. A neural network is a powerful tool, but seen as a black box[25].

As black boxes, it is also difficult to compare different neural networks. A naïve comparison might evaluate their accuracy with a particular dataset, but this is no guarantee of similarity between the two networks[20]. Even though two neural network may make similar predictions with some arbitrary training data, they may still generalize differently; their accuracies may be similar, the models may be very different. While methods for training artificial neural networks are well established and varied [11] [2], comparing neural networks is not something that can yet be easily done.

To evaluate the similarity between two neural networks, we must first consider what makes them similar. The models of two different neural networks can be very dissimilar, even though they were trained on the same data set. Two neural networks are not objectively similar based only on the accuracy. A new metric is needed to determine if the networks are similar in what they have

learned. Because the weights in neural networks are typically initialized randomly, each trained network may arrive at an entirely different model. The subset of weights that represent some concept in one neural network may be used by another neural network to represent a completely different part of the problem.

This issue means that it is not easy to compare two neural network model instances. Before they can be compared, the models must first be changed in some way such that weights with similar functional meaning are in the same place across both models. One solution would be to align one of the two neural networks to the other. When two neural networks are aligned, the weights with similar functional meaning should be moved to the same locations.

Meaningful distances between MLPs can be computed by aligning the weights with function-invariant topological transformations using bipartite matching. This enables a variety of applications, as well as the ability to compare networks based on their intrinsic models. Our algorithm, FBA, aligns neural networks according to their intrinsic models. The problem that we are solving is two-fold. First, how can neural networks be compared? Second, aligning can be used to compare neural networks, how can we align the networks?

To validate this statement we present a novel method called *Forward Bipartite Alignment* (FBA), that aligns the topological structures of two neural networks. FBA enables a diversity of applications, including evaluating the similarity of two neural networks, reducing large ensembles of neural networks down to a single model, facilitating cross-over between unrelated neural networks in evolutionary optimization, and producing meaningful vizualizations of sets of neural networks.

Once neural networks are aligned then we can evaluate how similar or dissimilar they are based on their models, rather than measurements taken over a finite set of sample points. Then, similarity can be evaluated based on the intrinsic nature of the neural networks.

Given two multi-layer neural networks with the same number of nodes in each layer, FBA adjusts the weights of one of the networks such that it will be aligned with the other network, using transformations that have no impact on the overall network output. It can negate weights

as long as affected downstream weights are also negated, and it can swap hidden units as long as correspondingly affected weights are also swapped. These transformations are applied to one of the neural networks such that its organization of weights aligns with those in the other network. FBA uses bipartite matching to find the operations that optimally align the two neural networks.

1.1 Thesis Contributions and Organization

To support this statement, this thesis provides the following contributions:

- A method to align multi-layer perceptrons
- A metric for measuring similarity
- Reasoning to show why alignment can find similarity
- Applications to demonstrate the usefulness of aligning
- Related work showing that this method did not exist previously
- Evaluation of the alignment algorithm

This thesis is organized to demonstrate each of the preceeding contributions. The next chapter will discuss background and related works. Chapter 3 examines similarity and the correlation with alignment. Then, implementation details of the alignment algorithm(FBA) will be discussed. Chapter 5 discusses three large application areas of the algorithm. Chapter 6 presents evaluation of the alignment algorithm. Chapter 7 concludes this thesis.

Chapter 2

Background

There are several different algorithms and methods that have similarities with our work. First, there are those algorithms which focus on combining neural networks in order to parallelize training. A method named Hogwild has been shown to be effective for parallelizing multilayer perceptrons [23]. It average the weights of multiple neural networks together at frequent intervals. This works because the frequent averaging forces every neural network to utilize the same weights for the same purpose, so there is no need to worry about which weights correspond with each other. Unfortunately, Hogwild is not useful in cases where communication is limited. For example, it could not be used to distribution computation across a cluster of separate machines without a very high-speed link between them. It certainly would not suffice for allowing arbitrary machines connected to the Internet to participate in a distributed training effort. However, FBA enables neural networks that have learned in unrelated directions to be averaged together with little-to-no loss. In some cases, averaging two neural networks together even leads to improved accuracy.

Other related work include methods that are not specifically for combining networks, but deal with some of the problems of non-aligned networks. Genetic algorithms have been used with multi-layer perceptrons in the past. Some work has focused on optimizing the training of a neural network, such as G-Prop [6]. Other work has been to optimize and find ideal topologies of neural networks such as Schiffmann's work [24]. Genetic algorithms can be used to optimize the weights, topology or parameters of a neural network, for example Montana and Davis' work with genetic algorithms replaced backpropagation and attempted to find globally optimal weights for a neural network [19]. In some cases, these genetic algorithms use multiple neural networks created over time as part of an evolution of networks. These networks are improved over generations to create a network capable of solving the desired problem. Forward Bipartite Alignment can be used to align these networks before crossover, or at anytime as part of a genetic algorithm.

Stanley used historical markings to track which weights correspond with each other in evolving generations of neural networks [26]. He showed that this resulted in better crossover of weights. Because of the historical nature of the weights, crossover could be done with weights that represent similar structures or functions. For example, those weights should be performing the same functionality for each neural network, even though the exact weights may differ. However, his approach does not detect when two separate evolutionary lines have serendipitously converged to compatible regions; because his method only tracks historical markings, it cannot anticipate two different evolutionary lines becoming similar. Merging two such genomes could have significant potential for making discoveries in previously unexplored regions of the gene space. FBA could be used to detect when these lines of neural networks are similar, and merge them even when learning has been distributed across different parts of the neural networks. Our approach can also be used with other genetic algorithms, after the weights have been trained. Crossover of weights may not be entirely meaningful if the weights are not aligned. If the networks were aligned before crossover occurred, more meaningful changes to the networks could be made, such as selectively choosing certain weights.

Because there are multiple ways to represent any problem with an MLP, no mechanism currently exists for measuring the distance between two MLPs. Such a distance metric could be useful for detecting when two MLPs have fallen into the same local optimum, or it could enable analysis with multidimensional scaling to visualize the relationships between different MLPs. Other work to visualize MLPs has focused on the training process and using principal component analysis to gain some insight [9]. Our alignment approach can be used to compute a distance between MLPs, opening up another avenue of visualization. Using this alignment method, neural networks can now be compared in more meaningful ways, showing how their internal models are different.

Forward Bipartite Alignment can be used for other applications, because it is only a technique to align two networks, and find similarity. While it can be used to visualize networks, or canonicalize them; our method can also be leveraged as a component in an ensemble. Ensembles enable a learning model to achieve greater predictive accuracy with the cost of additional computation,

which must be paid at both training-time and prediction time [4]. However, if the power of the ensemble can be encapsulated into a single model by averaging the weights together, then no additional cost must be paid at prediction-time. This principle was demonstrated by Anderson and Martinez in [1]. Even though their work was published over 15 years ago, no effective method has yet been found for averaging the weights of multilayer perceptrons. FBA can be used to align two neural networks, then averaging the weights becomes meaningful.

Chapter 3

Similarity between Neural Networks

3.1 Alignment and Similarity

Neural networks can be very complex systems, sometimes containing many thousands of nodes, each with potentially thousands of weights. How do we evaluate similarity between them? Two networks could be similar based on the generalization accuracy, training method, what problem they are intended to solve, etc. No suitable method exists to measure how different their models are. A naive approach could be to simply take the difference in their weights and use this to distance to say how dissimilar they are. However, this does not take into account any of the intricacies involved with the network's weights. As mentioned previously, neural networks combine attributes in nonlinear ways which leads to complex training patterns and widely varying utilization of the weights. Some methods attempt to regularize the weights but in these cases the weights are still widely varying in utilization.

The key problem addressed in this thesis is that weights which have been trained to represent some part of a problem could be in different places in the networks. As an example, consider two neural networks that have been trained to predict the maximum loan size for a person. The data set for this would be a set of survey answers from the person in question. One attribute could be the person's annual income. Some subset of weights in the neural network would learn how to handle the income attribute and combine it with other attributes to predict the maximum loan amount. That subset of weights would most likely be located in different places in another network even if they are trained on the same data. This is due to the randomness of initializing weights, the semi-random nature of training, but more importantly due to the nonlinear combinations that are made and the large number of iterations that it takes to train.

To solve this important problem, transforming the neural network such that it is aligned to the

other network is one possible solution. This solution is the basis for Forward Bipartite Alignment. Aligning a neural network has its own difficulties which are outlined in the subsequent chapters. Once two networks are aligned, the intrinsic nature of the networks can be evaluated and compared. Alignment is key in order to be able to compare two networks' models.

3.2 Distance Metric

Once two networks are aligned, a simple distance metric can be applied to generate a measurement which shows how similar the networks are. A large distance means they are not very similar, and a small distances means that they are very similar. A simple distance metric could be a summed, squared Euclidean distance. This would entail taking the distance between each pair of nodes in the two networks, squaring each distance, and then summing up the total. This single distance would represent how different the networks are as a whole. We will examine this very simple distance metric in more detail in the results chapter.

These MLPs are very complex as we have already stated. Due to the complexity, it could be argued that a more complex distance metric is required. There are many similarity functions that could be applied to neural networks, but we will outline a metric specifically for MLPs. Because the networks are made up of layers, we could calculate a distance for each pair of layers instead of just a single distance for the entire network. This would give us a distance tuple of the size of the number of layers in the networks. This tuple can be more difficult for a human to analyze but algorithms could analyze the distance to cluster similar networks together or examine the relationships. This tuple is helpful because of the nature of forward bipartite alignment. FBA aligns layer by layer, and analyzing the distance layer by layer can be helpful.

Chapter 4

Implementation

When two models are aligned, the elements of that model may be compared in a pair-wise manner to evaluate the similarity (or dissimilarity) between the two models. Unfortunately, multilayer perceptrons lack a canonical form, so even two multilayer perceptrons that represent precisely the same function for all possible inputs may yet have significantly different internal weights. In order to address this problem, we define two transformations that may be applied to a multilayer perceptron without affecting the functions they represent:

First, assuming the activation functions are antisymmetric, the output of any hidden unit may be negated if the weights into which it feeds are also negated. If the hidden unit has an activation function, a, which is antisymmetric about the input 0, then the output of this unit may be negated by adding $\sum_i 2a(0)w_i$ to its bias, and negating all of the other incoming weights. In cases where $a(0) = 0$, such as tanh, the biases will not be changed.

Second, any two hidden units, u_a, and u_b may be swapped if the corresponding weights and activation functions are also swapped. That is, all of the weights that previously fed into u_a should now feed into u_b, all of the weights that previously fed from u_a should now feed from u_b, and the corresponding activation functions must also be exchanged.

The target network remains constant, and the align network is changed. Neuron D is updated using the first transformation, negation. Neurons F and E are swapped using the second transformation. Certain function-invariant transformations also exist in degenerate cases. For example, when any two hidden units represent functions that differ only by a scalar factor, then it is possible to continuously adjust the weights that feed from these two units without affecting the function represented by the overall network. However, such cases are extremely rare since they require exact weight conditions, and network weights are typically initialized with small random values. Therefore, we can safely ignore such degenerate cases in the vast majority of real-world situations,

and assume that managing only the first and second cases is sufficient to align two feed-forward neural networks. In Figure 4.1, we show two neural networks that are to be aligned.

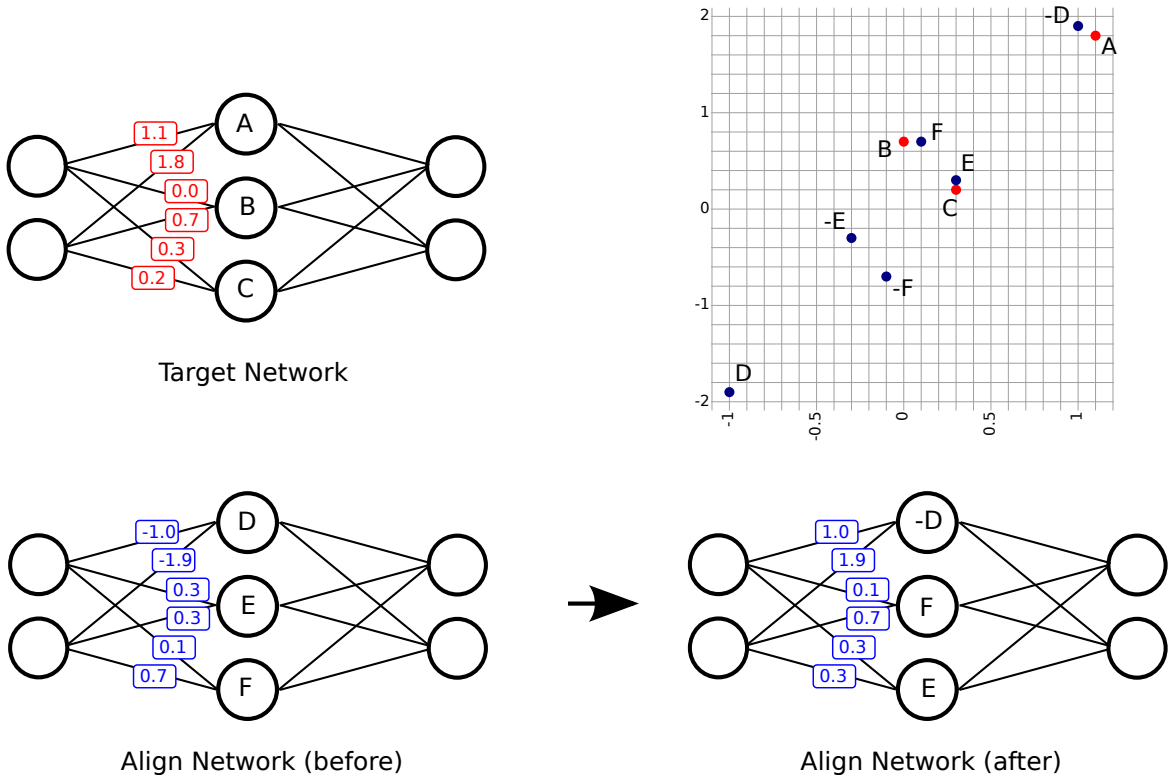

Figure 4.1: This figure shows how FBA analyzes weights. The align network will be aligned to the target network. Neurons A and D are similar, however neuron D should be negated. Neuron E is similar to neuron C, because of how close the weights are; likewise neuron F is similar to neuron B. FBA finds the optimal matching that minimizes the difference between the weights, but does not require matching weights to be identical. A plot of the weights feeding into hidden units is given, there it can be see that the similar neurons are grouped together. The final aligned network is shown in the bottom right.

A naïve approach for aligning neural networks might be to select an arbitrary canonical form, and convert both neural networks into this form. For example, one might swap network units such that they occur in each layer in sorted order according to the magnitude of the multi-dimensional vector of weights that feed into each unit, and one might invert the weights of any hidden layer in which the incoming weight with the largest magnitude is negative. The problem with such canonical forms is that they create arbitrary asymmetries in the space of possible neural networks.

In other words, very small changes in weights could result in a dramatically different canonical representation, depending on how close the networks happened to fall to the conditions that were selected to represent the canonical form.

An unbiased approach for aligning neural networks requires finding the optimal bipartite matches between the nodes in the corresponding layers of a multilayer perceptron. Fortunately, bipartite matching can reduce to a graph cutting problem with efficient known solutions [28]. If swapping network units were the only function-invariant operation with neural networks, then bipartite matching algorithms would provide a straight-forward solution to identifying the best way to swap the nodes. Unfortunately, negation is also a function-invariant operation. FBA addresses this complication by including both positive and negated representations of the weights of the units in one of the neural networks, such that n units are matched against $2n$ units in the other network. When one of the negated points is found to be optimal for the bipartite matching, this indicates that the weights of that unit need to be negated. In figure 4.1 the weight-vectors are plotted in a graph,

Figure 4.2: Foward Bipartite Alignment Psuedocode

let X be the target neural network
let Y be the network to be aligned
let L be the number of layers.
for all $l \in L$ **do**
  n:=number of units in l
  S:=set of weight-vectors that feed into l, for network X
  R:=set of weight-vectors that feed into l, for network Y, plus the n negations of each weight in R
  K:=maximum bipartite matching between S and R
  for all $i \in K$ **do**
    if i is a matching negated weight in K **then**
      negate the output of that unit in Y
    end if
  end for
  for all $i \in n$ **do**
    swap i in Y such that i now matches the node in X that it matched in K
  end for
  layer l is now aligned.
end for

including the negation of each weight. The similar neurons are closer together, and these would be

the matching weight vectors that bipartite matching would choose. The final aligned network can be seen in the bottom right of the figure.

Figure 4.2 gives pseudocode for the FBA algorithm. Let X be the "target" neural network, and Y be the network that we wish to align with X. X and Y must have the same number of units in each of their corresponding layers. For each layer in X, called l, let S be the set of n weight-vectors that feed into each unit in the lth layer. For each corresponding layer in Y, let R be the set of n weight-vectors that feed into each unit, plus the n negated weights. To find similarity between these layers we use bipartite matching. This finds a pairing between the point-vectors in R and the n closest points in S. If the matched pairings includes one of the negated vectors, as in the weight vector from R is the negation of the weight vector from S, then we negate the output of that unit in R. Next, we swap the units in R such that they align with the matching units in S. This process is repeated for each layer in the network until all hidden layers have been aligned. It is not necessary to align the output layer because both unit location and sign are constrained by the output labels that are used to train the neural network.

Chapter 5

Applications

5.1 Averaging Weights Ensemble

Forward Bipartite Alignment can be leveraged as a component in an ensemble learning process. Given a set of neural networks, these networks can be trained separately on the same problem set. Each neural network would be initialized differently, and perhaps trained on a different subset of the problem [4]. These different neural networks would produce a model different from the others because of their differing training and initialization.

For an ensemble of neural networks, prediction time can be very slow due to the need to propagate input through each network to receive a prediction. As shown in Anderson and Martinez's work on combining Single-Layer Perceptrons, an ensemble of perceptrons can be much faster at prediction time if there is only one model. They averaged weights between SLPs and generated a new combined model that performed as well or better than the ensemble of SLPs at prediction time. Similarly, our approach of alignment can be used to combine multi-layer perceptrons to produce a single model for prediction. See Figure 5.1 for a demonstration of the difference between bagging and wagging.

We detail a simple implementation of an ensemble using Forward Bipartite Alignment. This technique will work with a large number of networks, or a small number. Each neural network should be initialized randomly, and trained on the problem set. Bootstrapping can be used to further separate the networks' models such that they are presented with a different subset of the total pattern set. Training of these models should proceed normally with stochastic gradient descent, or some other training method. Once training has completed, the neural networks can be aligned, and combined into a single model. Align each network using the above shown algorithm to a "target" network. We suggest using the network with the highest accuracy as the "target," but

any of the networks can be used. Once alignment has been completed, comparing weights is now meaningful. A method to combine the networks could be to simply average their weights together. This averaging only works because the networks were first aligned. Other methods could be used, such as weighted averaging based on a confidence level in each network.

For this ensemble technique to be useful, no accuracy loss should occur. We implement the simple version of this algorithm, with only simple averaging of weights. We do believe that other averaging techniques can perform better than simple averaging, but we choose this method to show that the most simple combining method still suffers no loss. See the following section for results on the accuracy after combining the networks.

5.2 Visualization

Visualizing a neural network can be very difficult. Being black boxes, it is difficult to know if a comparison between two neural networks is correct. As two neural networks could achieve similar accuracy; their internal models, or how they reach that accuracy; could be very different. If we grouped networks based on accuracy, it does not reveal much information about the models themselves. Forward Bipartite Alignment can offer significant insight into the black box. Aligning two neural networks ensures that they are now similar, previously comparing un-aligned networks would be similar to comparing an apple to an orange. If accuracy of a fruit is determined by size, the apple and orange would be very similar. However, they are not the same thing. Un-aligned networks are similar to the fruit, they may be similar in one dimension, but very dissimilar in another.

Visualizing networks requires the networks to be comparable. Once two networks have been aligned, they are then comparable. We introduce a simple metric to quantify the difference in two networks. Taking the sum-squared weights of both networks and calculating the difference now yields a meaningful value that represents how similar or dissimilar the two networks are. This metric can then be used as the distance between two neural networks, and in many different ways. One method could be to use multi-dimensional scaling to visualize the relationships between neural

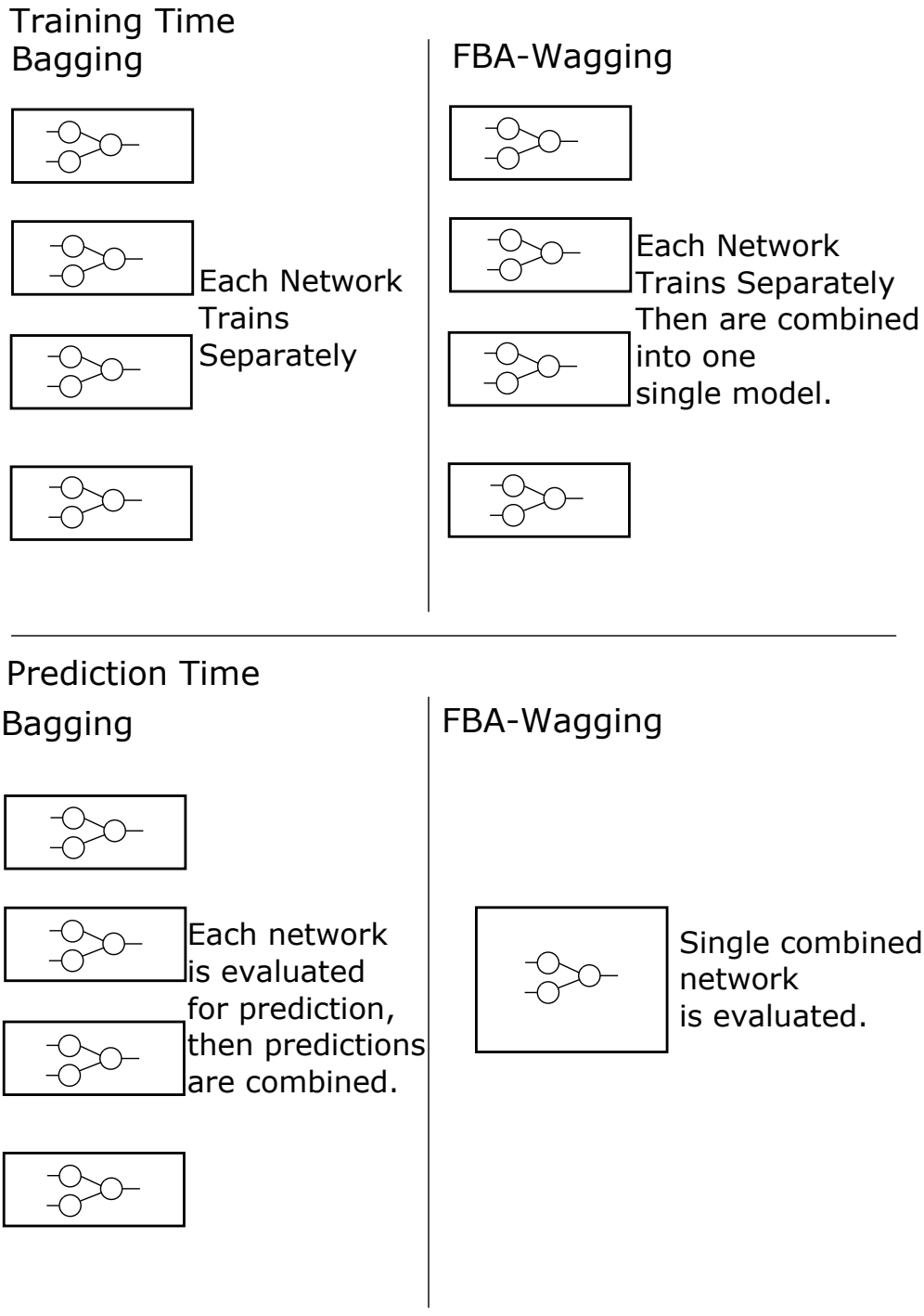

Figure 5.1: This figure shows the difference between bagging and FBA-wagging. Bagging pays the ensemble cost of evaluating all neural networks at both training and prediction time. FBA-wagging speeds up prediction time by allowing the ensemble to be encapsulated in a single model. This single model then performs as well as the bag of networks.

networks. Another could be to use simple clustering algorithms to cluster neural networks together. Given a set of neural networks with the same topology, each trained on a different dataset; FBA could allow for the comparison of datasets from within the network model space. Clustering these networks could show that some problems are solved in similar ways, or some similar problems are solved by neural networks in very different ways.

Multidimensional scaling (MDS) is a class of well-established dimensionality reduction methods that are useful for visualizing a set of things that have complex high dimensional representations [16][17]. MDS accepts as input a matrix of the pair-wise distances between every pair of high dimensional representations, and produces a low dimensional representation that exhibits approximately the same pair-wise distances. When the data is projected into 2 or 3 dimensions, humans can visualize the data that would otherwise be inaccessible due to its high dimensionality. A closely related method called Isomap [27] improves on MDS by only requiring the pair-wise distances to be measured in local neighborhoods, and estimating the other distances with the Floyd-Warshall algorithm. Isomap has demonstrated an ability to extract very high-level concepts from image-based data.

Previously, these methods could not be used for visualizing a set of neural networks, because no meaningful metric for evaluating the distances between neural networks was known. FBA solves this problem by enabling meaningful distance metrics that operate on a pair of neural networks. The ability to visualize sets of neural networks is significant because it makes ensembles of neural networks accessible to the powerful human intuition. For example, a visualization might enable humans to quickly determine which trained models in an ensemble of neural networks are outliers, or how many clusters of local optima are found with a particular problem. It also has potential applications in transfer-learning. Such problems typically involve training a neural network on a problem where abundant data is available, then retraining it on a problem with limited training data [29, 3, 7, 10, 18, 14]. The ability to visualize sets of neural networks would enable humans to cluster and categorize large sets of problems, and identify those that are the best candidates for transfer-learning.

5.3 Genetic Algorithms

Genetic algorithms use simulated evolution with a population of "chromosomes" to seek a chromosome that is well-fit for a particular purpose. In the case where a genetic algorithm is used to train a neural network, each chromosome in the population represents a set of candidate weights for the neural network. One of the most common operations used in genetic algorithms is cross-over, which selects two parent chromosomes, and generates a new child chromosome by drawing some elements from one of the parent chromosomes, and some elements from the other parent chromosome. Another operation that is commonly used when the chromosome consists of continuous values, as is the case with the weights in a neural network, is interpolation. Like cross-over, interpolation generates a new child chromosome by combining elements from two (or more) selected parent chromosomes. Unfortunately, both of these operations are meaningless if the neural networks are unaligned. Stanley mitigated this problem by using special markers to track the ancestral lineage of each weight [26]. However, this approach severely limits which parents may be combined to generate offspring to those that are closely related. Since the combination of close relatives in biological populations is known to be problematic, it is reasonable to suppose that combining chromosomes that are not closely related may be important for effective evolution.

Forward Bipartite Alignment is well-suited for aligning the selected parent neural networks, such that their weights can be combined in a meaningful manner to generate a child network. In both crossover and interpolation, FBA is first applied to align one of the selected parent networks with the other one, then the operation is performed on the chromosomes that represent the aligned networks. (It does not matter which parent network is selected to be aligned with the other one, because the resulting child network may also be aligned with other networks in future generations.)

For crossover, some of the weights for the child network are then drawn from the first parent, and the rest are drawn from the other parent. Some implementations may draw all the weights for a particular layer from the same parent, whereas other implementations may randomly choose a different parent for each weight. The advantages and disadvantages of these implementation

details are outside the scope of this paper, but it is relevant to note that FBA enables the networks to be aligned, which is necessary for meaningful crossover between neural networks that are not closely related.

For interpolation (or extrapolation), each weight in the child network, c is computed as a linear combination of the corresponding weights in the two parent networks a and b, such that $w_i^c = \gamma w_i^a + (1-\gamma) w_i^b$, where γ is a scalar factor for interpolation, and i iterates over all the weights in the child network. When γ is a value between 0 and 1, the child weight is an interpolation of the parent weights. When γ is less than 0 or greater than 1, the child weight is an extrapolation that further extends the difference between the two parent networks. Both cases are meaningful in a genetic algorithm, so both interpolation and extrapolation are likely to be used in the same genetic algorithm. Whether a constant value for γ is used at each weight, or whether a random value for γ is used for each weight is an implementation-specific detail.

Chapter 6

Results

6.1 Asymptotic Complexity

We test the efficiency of our alignment method by comparing the size of the network and alignment time. The cost of aligning a network should be small compared to the training of a network. To be an effective tool, the alignment algorithm should take a relatively small time. We compare a set of neural network sizes against the asymptotic complexity of the alignment process. The asymptotic complexity of FBA is essentially overwhelmed by the complexity of the bipartite matching. The asymptotic complexity of FBA as a whole is equal to $n * (ml^2 + lk)$. Where, n is equal to the number of layers, m is the nodes in the previous layer, l is the number of nodes in the current layer, and k is the number of nodes in the next layer. For each layer, bipartite matching is performed, then swapping and negating nodes is very small in comparison. In practice, we have found that the align method is very fast, and does not seem to slow an ensemble of many 3 layer neural networks.

6.2 Ensemble

We also examined the effect alignment has on an ensemble of neural networks. Our goal is to show that our alignment method has no appreciable effect on the accuracy of a neural network, even when it has been aligned and then combined with another network. This test would validate our alignment method has little loss when combining networks. We use simple averaging to combine the weights during this step, other methods could be used here instead. We chose simple averaging to show that even when combining weights in the simplest manner we still do not see loss. We also test our ensemble method against bagging, to show that in cases where bagging performs well, FBA aligned weight averaging (FBA-wagging) can speed up prediction time.

We compare against two methods of training a neural network, stochastic gradient descent by

Dataset	Neural Network	Bagging	FBA-Wagging
Breast-Cancer	0.3776	0.3440	0.3265
Bupa	0.4689	0.4231	0.4220
Dermatology	0.7147	0.6759	0.6459
Diabetes	0.373	0.349	0.348
Ionosphere	0.171	0.194	0.165
Iris	0.048	0.046	0.037
Lenses	0.325	0.383	0.283

Table 6.1: This table shows a subset of the UCI datasets with the error rate of neural network, bagging, and FBA-Wagging, where bagging improved the error rate over a neural network. Each column is the Neural Network, Bagging, or FBA-Wagging error rate, respectively. A lower number is better. This table shows the correlation between bagging and FBA-Wagging, when bagging does better FBA-Wagging should allow the bag to be combined into a single model to improve prediction time.

backpropagation, and a bagging ensemble of neural networks. For some problems and datasets, bagging improves accuracy over other methods. Similarly, we would expect a FBA-aligned bag of networks to perform as well as the simple bag ensemble. In cases where bagging improves accuracy, FBA could improve training time and will improve prediction time. Because FBA-wagging has only one model that is used in prediction, the prediction step is a simple evaluation of the neural network. Comparing against a single multi-layer perceptron shows that FBA and simple averaging does not cause a loss in accuracy.

We train a bag of neural network with 12 individual models; each with two hidden layers with sizes of 60 and 40 nodes respectively. We also train a separate set of neural networks with the same topology and number of models that will be aligned and averaged, known as FBA weight averaging (FBA-wagging). FBA-wagging is trained in the same way as the bagging ensemble, except at the end of training the neural networks are aligned and then their weights averaged. Finally, we train a single multi-layer perceptron with an identical topology (60 nodes in the first hidden layer, then 40 nodes in the second hidden layer) to the other networks. We repeated this training for a subset of the UCI dataset.

For datasets where bagging does not increase accuracy, FBA-wagging is not expected to have an impact on accuracy. We remove any datasets where bagging does not improve accuracy, ex-

Dataset	Neural Net	FBA-Wagging	Difference
Adult-Census	1653.97	1643.53	10.44
Anneal	1.63937	1.94778	−0.30841
Audiology	7.34623	6.72402	0.62221
Autos	75.9002	75.9811	−0.0809
Badges2	0.0057312	0.00586879	−0.00013759
Balance-Scale	50.295	47.3272	2.9678
Balloons	0.0107167	0.0138307	−0.003114
Breast-Cancer	25.4237	25.8919	−0.4682
Breast-W	4.56037	4.6047	−0.04433
Bupa	34.1569	33.3015	0.8554
Cars	386.546	385.52	1.026
Chess	4383.35	1444	2939.35
Colic	22.7566	22.4344	0.3222
Colon	3.90537	4.25473	−0.34936
Credit-a	24.9422	24.8268	0.1154
Credit-g	89.0866	90.092	−1.0054
Dermatology	15.0379	18.8277	−3.7898
Diabetes	53.593	54.065	−0.472
Glass	58.622	58.2534	0.3686
Heart-c	38.157	36.4324	1.7246
Heart-h	212.002	212.003	−0.001
Heart-statlog	15.4266	15.5202	−0.0936

Table 6.2: This table compares the Root Mean Squared Error (RMSE) of a single neural network, and a combined network which was combined using FBA-Wagging. The single neural network was first evaluated, then combined with 3 other neural networks using FBA-Wagging. The FBA-Wagging column shows the RMSE of this combined model. FBA-Wagging should not have any appreciable effect on accuracy. This table shows that even though 4 neural networks were combined with simple averaging after alignment, the accuracy does not deteriorate due to the aligning and combination process. In some cases, it improves the model.

cept for those datasets where FBA-wagging improved over the neural network where bagging did not. For datasets where bagging does improve accuracy, we show error rates with bagging, FBA-wagging, and a single neural network. See Figure 6.2 for the error rates. A lower score is better, as it means the model missed fewer testing samples. We expect to see the accuracy of FBA-wagging to be correlated with bagging, when it does better so should FBA-wagging. FBA-wagging would be used to combine the bagging models to improve prediction time, and in some cases it can improve generalization accuracy. Figure 6.2 shows that there is a correlation between bagging and FBA-Wagging.

To show that we have little loss when averaging networks together, we trained a set of neural networks on a subset of the UCI datasets. The network with the highest accuracy is shown for each dataset. We then combined the networks, using FBA-wagging, and display the combined model's accuracy. In the case that FBA aligns networks in a meaningful way, we would expect to see similar error rates on the majority of datasets. In Fig 6.2, we show the RMSE for a neural net and FBA-wagging, as well as the difference. The difference should be small, or positive as positive means that FBA-wagging beats the single neural network. Figure 6.2 shows that when combining a multi-layer perceptron with another using FBA results in little loss to accuracy.

We also want to show the benefit of aligning when averaging weights by showing what happens when you do not align the networks. For this experiment, we created a single baseline neural network for comparison, created with the same random number generator seed. Then, we created a set of neural networks for averaging without aligning. Finally, we created a set of neural networks for aligning and then averaging (FBA-Wagging). The three experiments share the same seed, so the weights of the networks and the order of training will be the same, the only thing different between them is whether they were aligned before their weights were averaged. The baseline neural network also had the same seed, so its weights will be similar to the weights of the first neural network in each of the other groups. The single neural network is there to show how well the topology of the networks will do on that particular dataset. For some datasets, using an arbitrary topology will result in poor results. For those datasets a better topology or combination of other metaparameters

(regularization, learning rate, etc.) would improve the accuracy. As these are a small number of the overall dataset results we received and tuning metaparameters is time consuming in both training cost and size of the parameter space, we did not perform tuning of metaparameters for those datasets.

The variable that changes between the two experiment groups is whether the networks were aligned before their weights are averaged. If our algorithm does find similarity, and does properly align the similar weights with each other; then we should see that the FBA error rate is lower than the WAG (or simple averaging) error rate. Our hypothesis is that averaging neural networks together is not only not good, it will typically make the error rate increase. This is because averaging unaligned networks results in combining weights that are not being used for the same concept, which leads to changing weights which may have been well-trained to something new that does not reinforce the training.

We now present a series of graphs, and plots of experiments that followed the above description. We varied a number of different parameters across these different tests: number of models, topology, and random seed. The number of models refers to how many neural networks there are in the ensemble of networks that were trained and averaged together in both WAG and FBA-Wagging. This was varied to show that even with an increase of models, WAG does not improve. The topology is the number of layers and the number of neurons in each layer. We varied the topology for a variety of reasons: First, the topology is a key factor in error rates for individual datasets. Second, the topology is important for demonstrating how our algorithm works. FBA aligns networks layer-by-layer, adding more layers and more neurons shows that our algorithm still finds similarity even though the complexity has increased. For these experiments we did not put too much time into optimizing extraneous metaparameters like momentum, learning rate, and so on. These metaparameters can help get a lower error rate for a single neural network, however optimizing these metaparameters can be very time consuming (as in grid search or some other algorithm) and only improve a single neural network. This does not demonstrate that we can take an arbitrary set of neural networks and align then average them together.

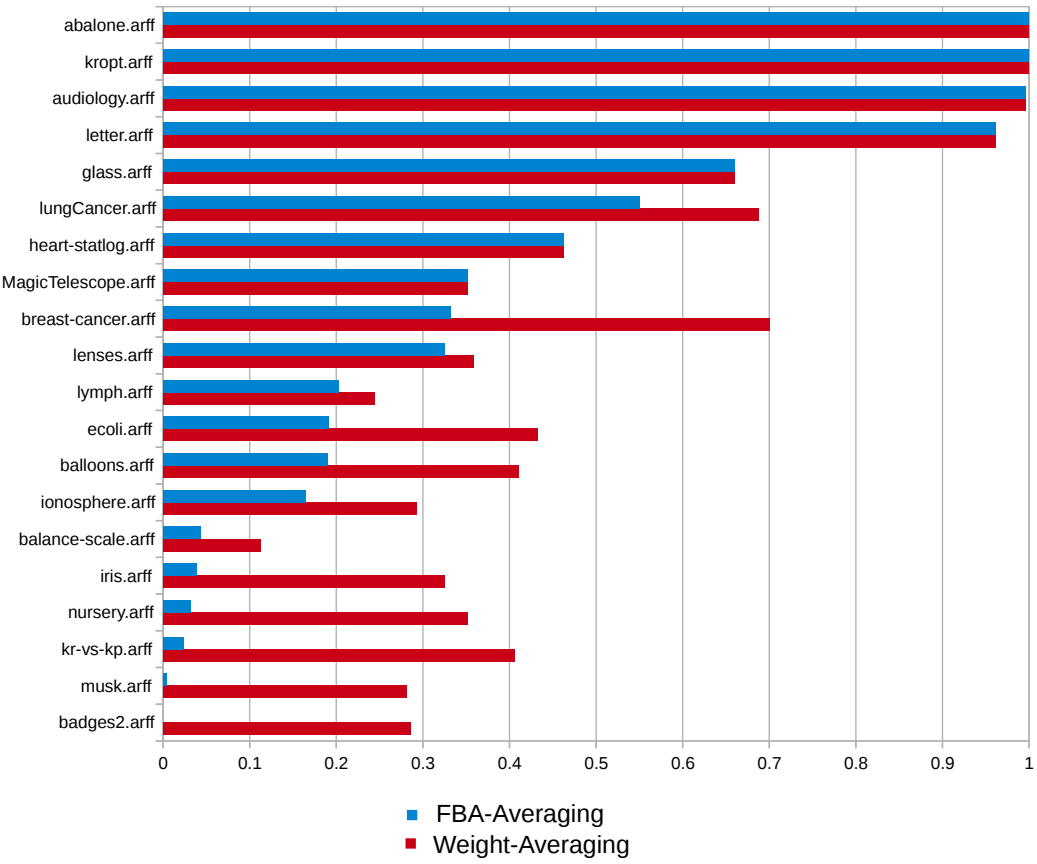

Figure 6.1: Parameters: Topology is 60, 40 hidden units (2 hidden layers). Number of models is 4. X Axis is error rate, lower is better. Y Axis lists the data sets. FBA-Averaging is in blue and simple Weight Averaging is in Red.

These graphs show when our align and then average method (FBA-Wagging) does better, and when it does worse than simple averaging. We have found that in nearly all cases when simple averaging ties or beats FBA-Wagging it is due to a handful of reasons. The first common reason is that the neural networks were not trained. This can be seen in the graphs for datasets like abalone and kropt. The error rate for both simple averaging and FBA-Wagging is extremely high. Indeed, for a single neural network trained with the same topology it is that high as well. In these cases the neural networks themselves were not trained well enough. This can be because of a poor topology that could not facilitate learning on that dataset, poor other metaparameters that could have been optimized better, or the complexity of the dataset. For those cases, we can safely eliminate their comparison. We show them below for completeness. Other cases are when the error rate is around 50 percent or more, but not as untrained as the extremely high error rates. Here, it seems that we did not train for long enough, metaparameters were nut tuned well enough, or the models were stuck in a local optimum. When this happens the error rates are typically identical across the three models. An example of this is heart-statlog or MagicTelescope in Figure 6.2.

When the neural networks have been sufficiently trained we can see the real impact of aligning and then averaging. Consider these datasets from Figure 6.2: iris, nursery, kr-vs-kp, musk, and badges2. Here, our error rates are less than 5 percent, and for a few nearly 0 percent. Compared to simple averaging which reduces the accuracy by 20 or 30 percent. Consider Figure 6.2 where FBA-Wagging does well and reaches 0 percent error rate, simply averaging can cause such a negative effect that the error rate of that model reached almost 85 percent. If that model had been aligned before averaging, the error rate would have been nearly 0 percent. This shows that averaging is detrimental to a set of neural networks if the networks have not been aligned. Once the networks have been aligned, we can properly share information between the networks. For Figure 6.2 we postulate the small difference in some previously large error rates to be a result of the random seed that was used. The seed determines the order of training, and in some cases we can get lucky with how the ensemble of neural networks was trained. In the case of these, we believe that the neural networks were using some similar weights for similar functionality. Those networks were more

aligned to begin with, so simple averaging did not completely ruin the accuracy. This can not be assumed for all neural networks or ensembles however. In nearly all cases, simple averaging will result in a worse error rate. This is because we are not sharing information across functionally similar neurons, but squashing arbitrary neurons together hoping that they improve.

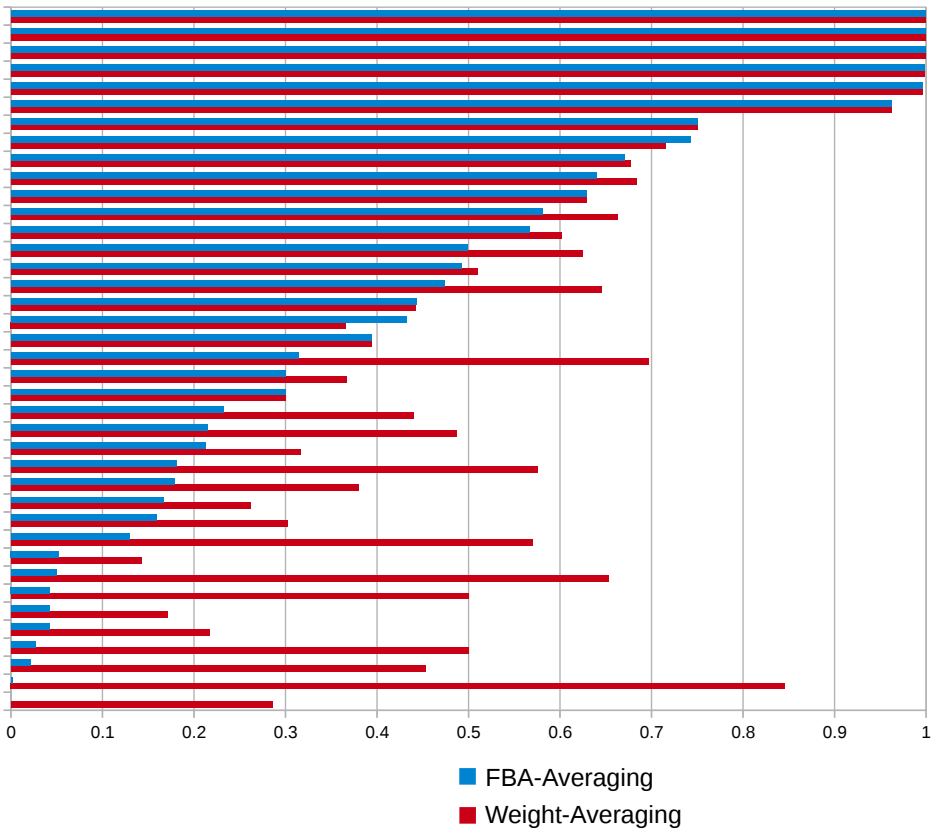

Figure 6.2: Parameters: Topology is 120, 60, 40 hidden units (3 hidden layers). Number of models is 4. X Axis is error rate, lower is better. Y Axis lists the data sets. FBA-Averaging is in blue and simple Weight Averaging is in Red.

Finally, we show some raw data that we used to generate the above graphs. The table 6.2 contains the raw error rates that were used to generate the charts. We do not include all raw data in table form for aestethic reasons. The table contains four columns, one of the dataset, one for the single neural network, one for weight averaging and finally one for FBA-Wagging. This table is specifically the data for 6.2. The dataset names are listed if crosschecking is needed.

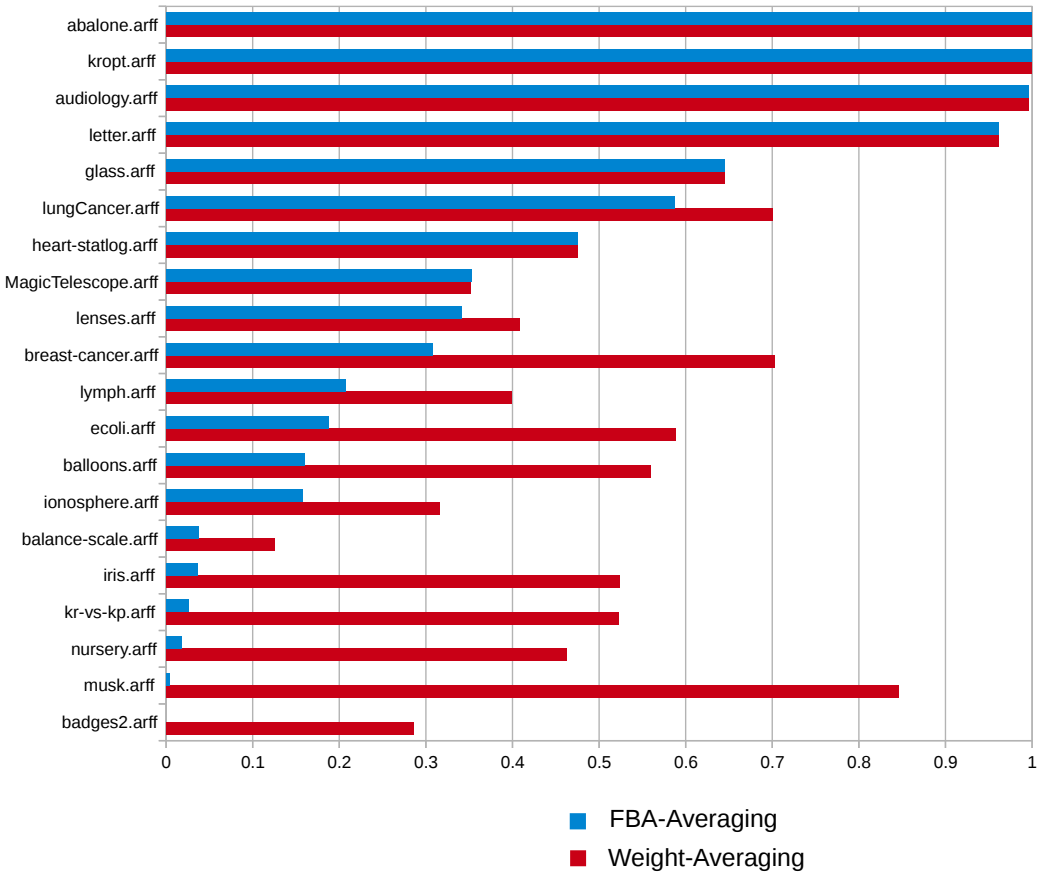

Figure 6.3: Parameters: Topology is 60, 40 hidden units (2 hidden layers). Number of models is 8. X Axis is error rate, lower is better. Y Axis lists the data sets. FBA-Averaging is in blue and simple Weight Averaging is in Red.

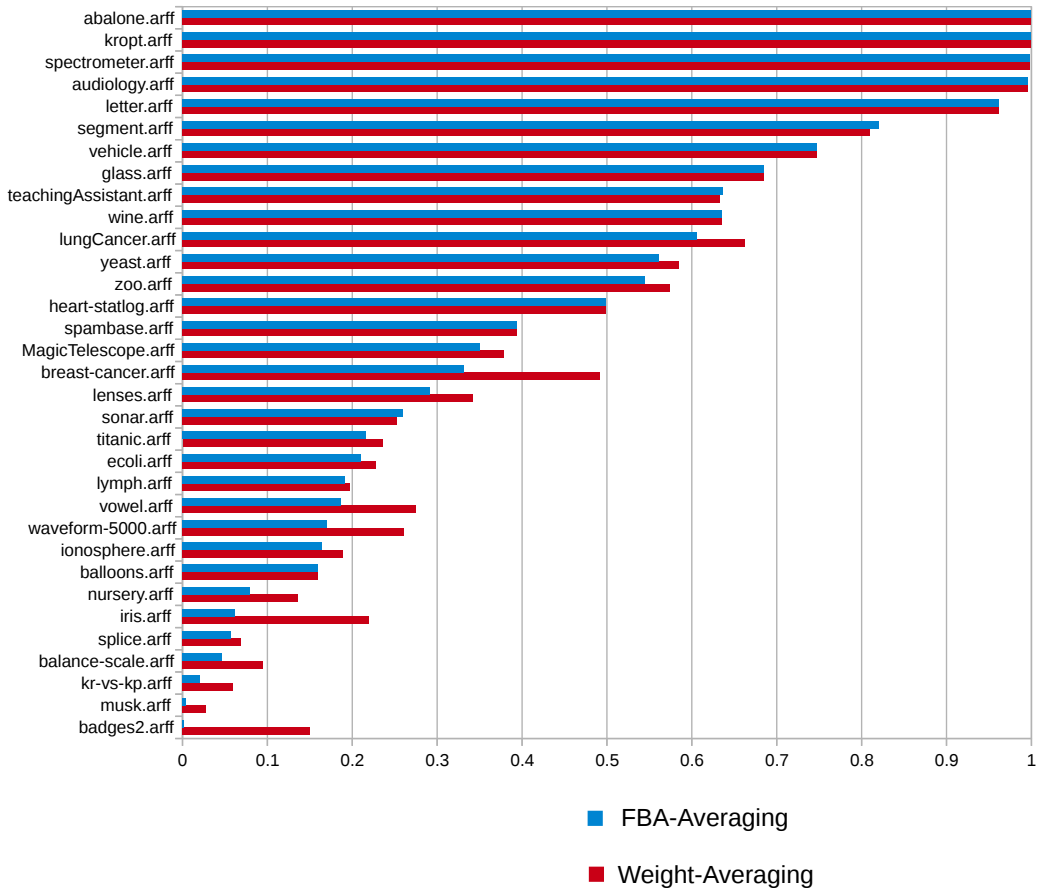

Figure 6.4: Parameters: Topology is 60, 40 hidden units (2 hidden layers). Number of models is 2. X Axis is error rate, lower is better. Y Axis lists the data sets. FBA-Averaging is in blue and simple Weight Averaging is in Red.

Dataset	Single Neural Net	Weight Averaging (WAG)	FBA (Align then Average)
abalone	0.9997	0.9997	0.9997
audiology	0.9955	0.9955	0.9955
badges2	0.0000	0.2857	0.0000
balance-scale	0.0384	0.1705	0.0425
balloons	0.1400	0.5700	0.1300
breast-cancer	0.3580	0.6965	0.3146
breast-w	0.0480	0.6535	0.0500
bupa	0.4411	0.4423	0.4428
cars	0.0356	0.2165	0.0424
chess	0.9990	0.9990	0.9990
chess-KRVKPawn	0.0219	0.4768	0.0267
colon	0.3322	0.6451	0.4741
credit-g	0.3420	0.3000	0.3000
ecoli	0.1952	0.4869	0.2142
glass	0.4074	0.6775	0.6700
heart-statlog	0.5133	0.5096	0.4918
ionosphere	0.1703	0.3025	0.1589
iris	0.0533	0.5000	0.0426
kropt	0.9990	0.9990	0.9990
kr-vs-kp	0.0264	0.4994	0.0270
lenses	0.4166	0.3666	0.300
letter	0.9617	0.9617	0.9617
lungCancer	0.6000	0.6625	0.5812
lymph	0.1972	0.3797	0.1783
MagicTelescope	0.3194	0.3661	0.4317
musk	0.0017	0.8458	0.0020
nursery	0.0006	0.4532	0.0213
segment	0.0483	0.7152	0.7430
sonar	0.2346	0.4394	0.2326
spambase	0.4482	0.3940	0.3941
spectrometer	0.9981	0.9981	0.9981
splice	0.0707	0.1424	0.0525
teachingAssistant	0.6423	0.6834	0.6397
titanic	0.2211	0.3159	0.2129
vehicle	0.2326	0.7498	0.7498
vowel	0.1062	0.5749	0.1804
waveform-5000	0.1744	0.2614	0.1671
wine	0.0258	0.6292	0.6292
yeast	0.4766	0.6247	0.4990
zoo	0.6158	0.6019	0.5663

Table 6.3: This table shows the data for 6.2 which is the experiment with the topology of 120, 60, 40 with 4 models. For cases where there are ties or losses, these are when the neural networks were not trained well enough to the dataset, either because of a poor topology for that dataset or the complexity of the dataset.

Chapter 7

Conclusion

To support the thesis statement, we provide the following set of contributions:

- A method to align multi-layer perceptrons
- A metric for measuring similarity
- Reasoning to show why alignment can find similarity
- Applications to demonstrate the usefulness of aligning
- Related work showing that this method did not exist previously
- Evaluation of the alignment algorithm

We have presented Foward Bipartite Alignment (FBA), a method to align two arbitrary artificial neural networks that have the same topology. FBA is general, and works for any number of networks, layers, or nodes. We introduced a simple metric for measuring the similarity between networks that have been aligned. We gave reasoning for why alignment finds similarity, and demonstrated the usefulness of aligning beyond just comparing similarity. We thoroughly covered related work and background material.

FBA has many uses in the field of neural networks, including ensembles, visualization, and genetic algorithms. We described specific implementation details on those three categories to show that FBA has real use. We anticipate other uses for aligning two networks. To validate our method, we described two experiments one designed to show the correlation between FBA-Wagging and bagging; and another to show that FBA aligns neural networks in a meaningful way.

References

[1] Tim Andersen and Tony Martinez. Wagging: A learning approach which allows single layer perceptrons to outperform more complex learning algorithms. In *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'99*, 1999.

[2] Y. Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2009.

[3] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. *ICML Unsupervised and Transfer Learning*, pages 17–36, 2012.

[4] L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

[5] Leo Breiman. Random forests. *Machine Learning*, 45:5–32.

[6] P. A. Castillo, J. J. Merelo, Alberto Prieto, V. Rivas, and Gustavo Romero. G-prop: Global optimization of multilayer perceptrons using gas. *Neurocomputing*, 35(1):149–163, 2000.

[7] Dan Claudiu Ciresan, Ueli Meier, and Jürgen Schmidhuber. Transfer learning for latin and chinese characters with deep neural networks. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2012.

[8] George Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[9] Marcus Gallagher and Thomas Downs. Visualization of learning in multilayer perceptron networks using principal component analysis. *IEEE Transactions on Systems, Man, and Cybernetics Part B,*, 33(1):28–34, 2003.

[10] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (icassp)*, pages 6645–6649, 2013.

[11] Geoffrey et al. Hinton. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[12] Tin Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:832–844, 1998.

[13] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[14] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *IEEE International Conference on Acoustics, Speech and Signal Processing (icassp)*, pages 7304–7308, 2013.

[15] Sutskever Ilya. Krizhevsky, Alex and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Neural Information and Processing Systems*, 2012.

[16] Joseph B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29.1:1–27, 1964.

[17] Joseph B. Kruskal and Myron Wish. Multidimensional scaling. 11, 1978.

[18] Grégoire et. a. Mesnil. Unsupervised and transfer learning challenge: a deep learning approach. *ICML Unsupervised and Transfer Learning*, pages 97–110, 2012.

[19] David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. *IJCAI*, 89:762–767, 1989.

[20] Adam H. Peterson. COD: Measuring the similarity of classifiers. Master's thesis, Brigham Young University, January 2005.

[21] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):1986, Mar 1986.

[22] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[23] Benjamin Recht. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, 2011.

[24] W. Joost M. Shiffmann and R. Werner. Application of genetic algorithms to the construction of topologies for multilayer perceptron. *Artificial Neural Nets and Genetic Algorithms*, pages 675–682, 1993.

[25] Jonas et al. Sjoberg. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31:1691–1724, 1995.

[26] Kenneth O. Stanley and Risto Miikkulainen. Efficient evolution of neural network topologies. In *CEC'02. Proceedings of the 2002 Congress on Evolutionary Computation*. IEEE, 2002.

[27] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290.5500:2319–2323, 2000.

[28] Douglas Brent West. *Introduction to graph theory*, volume 2. Prentice hall, Upper Saddle River, 2001.

[29] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

Papers Published In Conferences

Stephen Ashmore and Michael Gashler, "A Method for Finding Similarity between Multi-Layer Perceptrons by Forward Bipartite Alignment ", In Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2015, Killarney, Ireland, 2015.