A Method for Finding Similarity between Multi-Layer Perceptrons by Forward Bipartite Alignment

Stephen Ashmore and Michael Gashler
Department of Computer Science and Computer Engineering
University of Arkansas
Fayetteville, Arkansas
Email: {scashmor, mgashler}@uark.edu

***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 have a 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.**

I. Introduction

Artificial neural networks are powerful tools. 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, decision trees produce models that are relatively easy to understand. Decision trees may make their decisions based on entropy [20] [21] or random choices [12] [4], or other methods. 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 do not work with simple rules, but rather with a complex model surface determined by their weights that combine attributes non-linearly. These surfaces may be in high-dimensional spaces, and the weights may take on any continuous values. Although neural networks are quite powerful, they are seen as a black box [24].

As a black box, 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. Even though two neural networks may make similar predictions with the training data, they may still generalize differently. Even though 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.

We present a method that aligns the topological structures of two neural networks called *Forward Bipartite Alignment* (FBA). 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, transforming neural networks into a canonical form, facilitating cross-over between unrelated neural networks in evolutionary optimization, and producing meaningful vizualizations of sets of neural networks.

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. However, when two neural networks are aligned by the proposed method, weights with similar functional meaning are moved to the same locations.

Given two multi-layer neural networks with the same number of nodes in each layer, FBA adjusts the weights 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 other network. FBA uses bipartite matching to find the operations that optimally align the two neural networks.

A method named Hogwild has been shown to be effective for parallelizing multilayer perceptrons [22]. It averages 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

978-1-4799-1959-8/15/$31.00

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.

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 [23]. 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 [25]. 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 [5]. 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.

Section 2 introduces the detailed algorithm of aligning two networks. Section 3 covers the other applications of aligned neural networks including visualization, ensembles, and genetic algorithms. Section 4 explores the validation and evaluation results of the algorithm. Finally, section 5 concludes the paper.

II. Algorithm

When two models are represented in a canonical form, 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 (For example, the logistic function is antisymmetric about the point (0, 0.5), so if the logistic activation function is used then 2a(0)=1.). 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

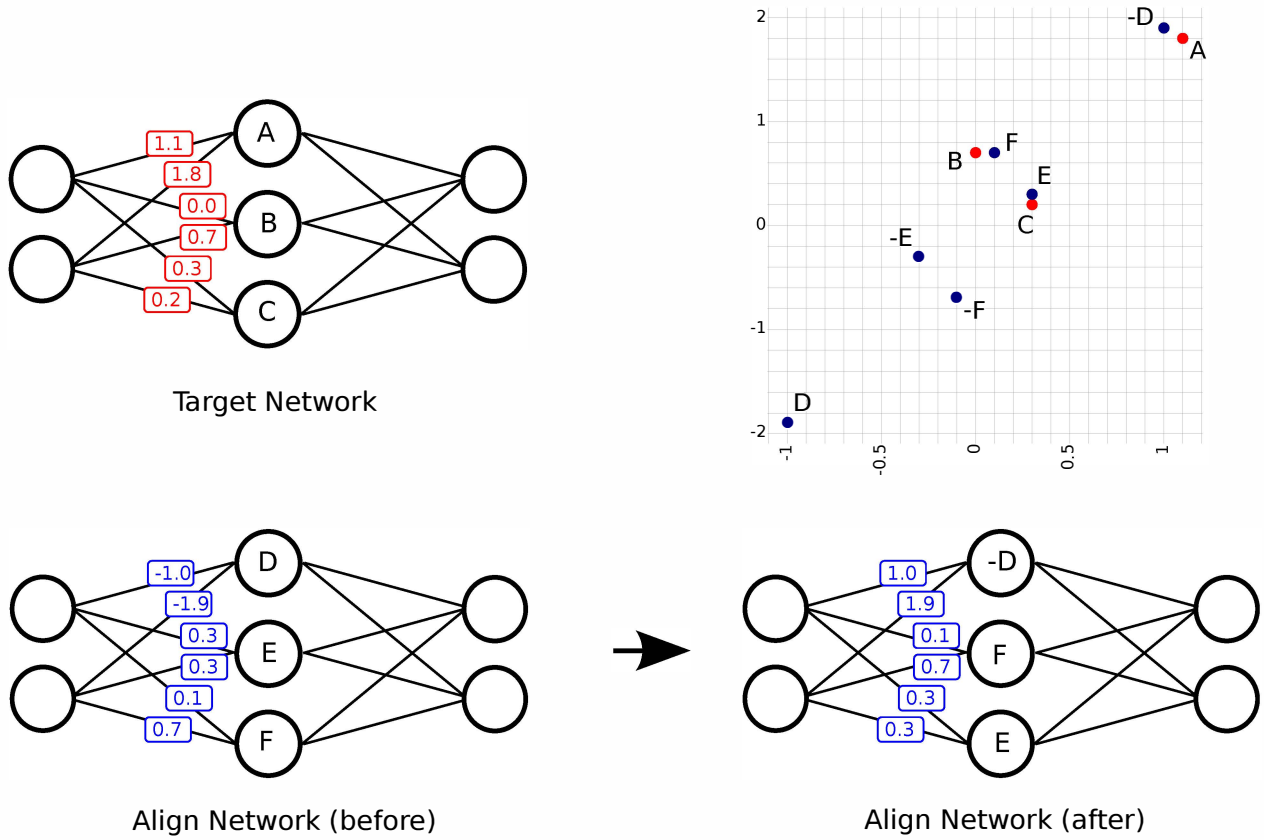

Fig. 1. This figure shows how FBA analyzes weights. The align network will be aligned to the target network. Neurons A and D are similar, however neuron D should be negated. Neuron E is similar to neuron C, because of how close the weights are; likewise neuron F is similar to neuron B. FBA finds the 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.

two feed-forward neural networks. In Figure 1, we show two neural networks that are to be aligned.

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 1 the weight-vectors are plotted in a graph, 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 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

Fig. 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 ∈ 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 ∈ 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 ∈ 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
```

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.

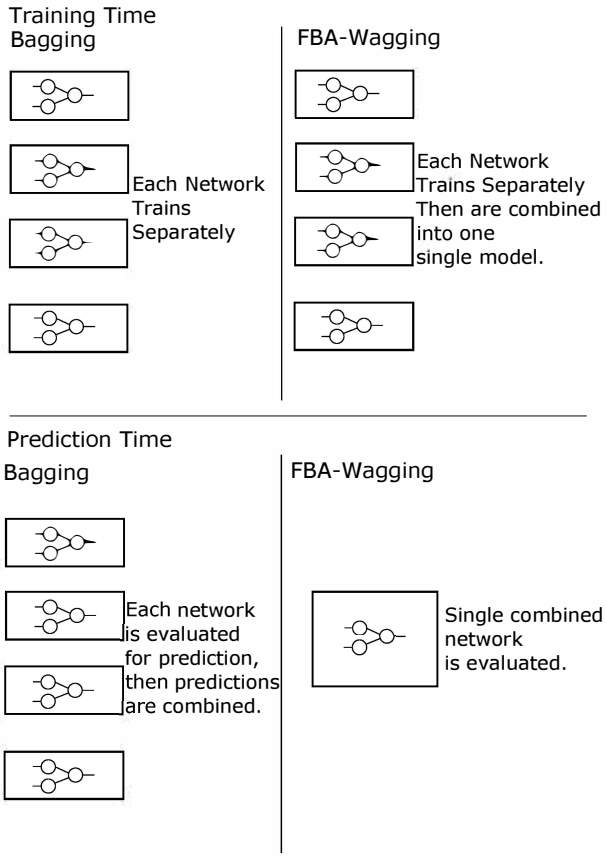

Fig. 3. This figures shows the difference between bagging and FBA-wagging. Bagging pays the ensemble cost of evaluating all neural networks at both training and prediction time. FBA-wagging speeds up prediction time by allowing the ensemble to be encapsulated in a single model. This single model then performs as well as the bag of networks.

III. Applications

A. Averaging Weights Ensemble

Forward Bipartite Alignment can also 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 [5]. These different neural networks would produce a model different from the others because of their differing training and initialization.

For an ensemble set 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 3 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.

B. 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.

To visualize networks then 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. Once two networks are aligned, their weights can be compared in a meaningful way. 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.

This distance metric can be used in many different ways. One method could be to use multi-dimensional scaling to visualize the relationships between neural 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 [26] 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.

C. Genetic Algorithms

Genetic algorithms use simulated evolution with a population of "genomes" to seek a genome that is well-fit for a particular purpose. In the case where a genetic algorithm is used to train a neural network, each genome 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 genomes, and generates a new child genome by drawing some elements from one of the parent genomes, and some elements from the other parent genome. Another operation that is commonly used when the genome 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 genome by combining elements from two (or more) selected parent genomes. 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 [25]. 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 genomes 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 genomes 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.

IV. Evaluation

A. 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.

B. 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 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, except for those datasets where FBA-wagging improved

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

Fig. 4. 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.

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

Fig. 5. 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.

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 4 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 4 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 5, 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 5 shows that when combining a multilayer perceptron with another using FBA results in little loss to accuracy.

V. Conclusion

We have presented Foward Bipartite Alignment (FBA), a method to align two arbitrary artificial neural networks of any size. 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] Andersen, Tim, and Tony Martinez. "Wagging: A learning approach which allows single layer perceptrons to outperform more complex learning algorithms." Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'99. 1999.

[2] Bengio, Y. "Learning Deep Architectures for AI", Foundations and Trends in Machine Learning. 2009.

[3] Bengio, Y. "Deep Learning of Representations for Unsupervised and Transfer Learning". ICML Unsupervised and Transfer Learning, 2012, pg 17-36.

[4] Breiman, Leo. "Random Forests". Machine Learning Vol. 45, pg 5-32.

[5] Breiman, L. "Bagging predictors". Machine Learning. 1996, 26(2), 123-140.

[6] Castillo, P. A., J. J. Merelo, Alberto Prieto, V. Rivas, and Gustavo Romero. "G-Prop: Global optimization of multilayer perceptrons using GAs." Neurocomputing 35, no. 1 (2000): 149-163.

[7] Ciresan, Dan Claudiu and Meier, Ueli and Schmidhuber, Jürgen. "Transfer learning for Latin and Chinese characters with deep neural networks". Neural Networks (IJCNN), The 2012 International Joint Conference on. 2012, IEEE, pg 1-6.

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

[9] Gallagher, Marcus, and Downs, Thomas. "Visualization of Learning in Multilayer Perceptron Networks Using Principal Component Analysis". Systems, Man, and Cybernetics Part B, IEEE Transactions on. 2003 Vol 33 (1), 28-34.

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

[11] Hinton, Geoffrey, et al. "A fast learning algorithm for deep belief nets." Neural Computation 18 no. 7. 2006, 1527-1554.

[12] Ho, Tin. "The Random Subspace Method for Constructing Decision Forests". IEEE Transactions on Pattern Analysis and Machine Intelligence. 1998 Vol. 20, pg 832-844.

[13] Hornik, Kurt. "Approximation Capabilities of Multilayer Feedforward Networks". Neural Networks. 1991, 4(2), 251-257.

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

[15] Krizhevsky, Alex. Sutskever, Ilya. and Hinton, Geoffrey. "ImageNet Classification with Deep Convolutional Neural Networks" Neural Information and Processing Systems, Proceedings of, 2012.

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

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

[18] Mesnil, Grégoire et. a. "Unsupervised and Transfer Learning Challenge: a Deep Learning Approach". ICML Unsupervised and Transfer Learning, 2012, pg 97-110.

[19] Montana, David J., and Lawrence Davis. "Training Feedforward Neural Networks Using Genetic Algorithms." In IJCAI, vol. 89, pp. 762-767. 1989.

[20] Quinlan, J. R. 1986. "Induction of Decision Trees". Mach. Learn. 1, 1 (Mar. 1986), 81-106

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

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

[23] Shiffmann, W. Joost, M. and Werner, R. "Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons". Artificial Neural Nets and Genetic Algorithms, 1993. pg. 675-682

[24] Sjoberg, Jonas, et al. "Nonlinear Black-Box Modeling in System Identification: a Unified Overview." Automatica, 1995. Vol. 31, pg 1691–1724.

[25] Stanley, Kenneth O., and Risto Miikkulainen. "Efficient evolution of neural network topologies." Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on. Vol. 2. IEEE, 2002.

[26] Tenenbaum, Joshua B., Vin De Silva, and John C. Langford. "A global geometric framework for nonlinear dimensionality reduction." Science 290.5500 (2000): 2319-2323.

[27] Werbos, Paul J. "The Roots of Backpropagation. From Ordered Derivatives to Neural Networks and Political Forecasting". New York, NY: John Wiley & Sons, Inc.

[28] West, Douglas Brent. "Introduction to graph theory". Vol. 2. Upper Saddle River: Prentice hall, 2001.

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