Missing the Forest for the Trees: Experimentation with Classifier-Based Neural Networks

CPSC 66, Fall 2017

Robert Hwang RHWANG1@SWARTHMORE.EDU
Swarthmore College, 500 College Ave, Swarthmore, PA 19081 USA

Tai Warner TWARNER2@SWARTHMORE.EDU
Swarthmore College, 500 College Ave, Swarthmore, PA 19081 USA

Colin Pillsbury CPILLSB1@SWARTHMORE.EDU
Swarthmore College, 500 College Ave, Swarthmore, PA 19081 USA

Abstract

Machine learning has transformed from a nebulous, clouded branch of computer science to an indispensable tool utilized across several industries and disciplines. Alongside its massive growth, the implementation and usage of classifiers have become exponentially more complex and often require expert domain knowledge alongside a seemingly infinite pool of resources to create meaningful results. In this paper, we present an implementation of a neural forest classifier, a novel technique that fuses the statistical, computational, and representational strengths of ensemble learners with the powerful learning capability of neural networks. The model will be able to combat common obstacles facing most deep neural network models today, while also introducing a modular implementation strategy for maximum code reusability. We hypothesized that our neural forest implementation would achieve comparable, if not better accuracy relative to standard classifiers. Our goal for this study was to provide an implementation of a neural forest that would be significantly less resource intensive to train, while also providing a basis upon which to further tune key hyperparameters that have already been identified in prior literature.

CPSC 68 Bioinformatics Final Report Proceedings, Swarthmore College, Spring 2017.

1. Introduction

Recent advances in the machine learning field have advanced our comprehension and capability to design high throughput algorithms for complex, multifaceted classification tasks. Since their inception in the 1950s, neural networks have fundamentally altered the way we approach large scale, complex computational tasks. Although the original implementations of neural networks were far from ideal, in recent years, neural networks have been utilized successfully for a broad range of tasks, particularly in computer vision and pattern recognition(Rowley & Kanade, 1998). More recently, multilayer neural networks have been examined as a viable alternative to principal component analysis for dimensionality reduction(Hinton & Salakhutdinov, 2006). Inspired by modeling a highly complex and effective biological system, the neural network is a directed acyclic graph, consisting of weighted connections between three layers of nodes which are termed the input layer, the hidden layer, and the output layer. The input layer contains nodes that accept messy real world data through feature vectors. The data is passed through one or several hidden layers before finally being passed to the output layer. Figure 1 shows a basic neural network, complete with directed edges and implied weights.[1]

In essence, the neural network model is attempting to draw a nonlinear decision boundary through the calculations done in the hidden layers, while also adjusting the weights to reduce error through backpropagation. While there are several important neural network architectures, the ones we will focus on are single layer perceptrons and multilayer perceptrons.

We hypothesized that a neural network comprised of nodes

[1] Obtained from SimplyML

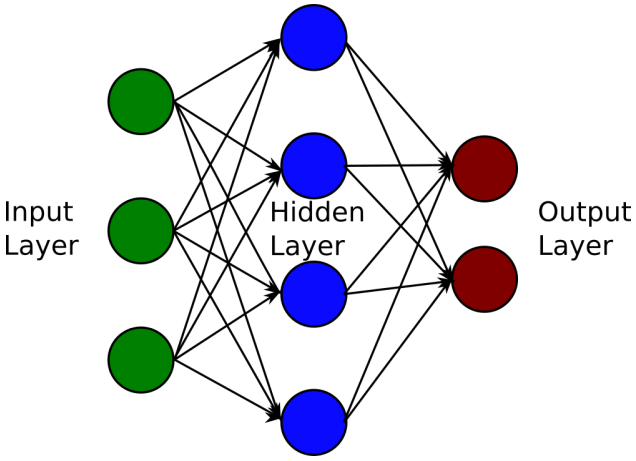

Figure 1. A classical neural network schema, with an input layer, a hidden layer, and an output layer.

of ensemble learners would be able to capture a better representation of the distribution of data during the training phase. To test our results, we ran our novel approach on two binary classification tasks, and compared the performance of our neural forest with the performance of a random forest, a neural network, and a decision tree. Performance was evaluated based on the accuracy, precision, and recall, and confusion matrices were constructed for each test parameter.

1.1. Ensemble Learning and Error Reduction

Ensemble learning is a powerful concept in supervised machine learning, which allows for a model to learn a problem by averaging the results of several learners. Diversity and independence are the key conditions that must be met for the averaging of these learners to be effective (Kuncheva & Whitaker, 2003).

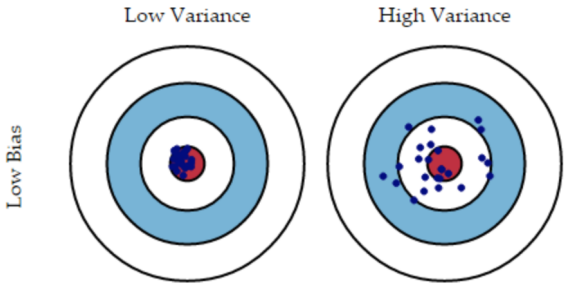

Figure 2. Bias and Variance as drawn on targets, or an example distribution of an ideal ensemble learning distribution.

Ensemble algorithms have been developed to handle different cases in terms of the bias/variance tradeoff. The error in a model can be decomposed into three parts — bias error, variance error, and unavoidable error. Figure 2 may seem to be a simple diagram of bias and variance in models, but it demonstrates the effectiveness of ensembles quite well.[2] We have implied that an ensemble works best with low bias, high variance base learners. Imagine that the base learner distribution is the one to the right, demonstrated in Figure 2. If we averaged all of these learners, we would get a single point that is extremely close to the ideal solution of low bias and low variance, or the one to the left. Bagging is a common resampling technique in ensemble learning, that subsamples with replacement to train independent learners on independently and identically distributed (i.i.d.) data. The random forest algorithm is a prime example of an ensemble learner that takes full advantage of the properties of bagging. A forest trains several decision trees through bagging, before taking the average of each learner's prediction, which results in a large reduction in variance error with no increase in bias error. Figure 3 expresses the bias variance tradeoff, and shows the optimal model complexity, as the minimum of both variance and bias error, and also pictures the powerful allure such a technique has on model design.[3]

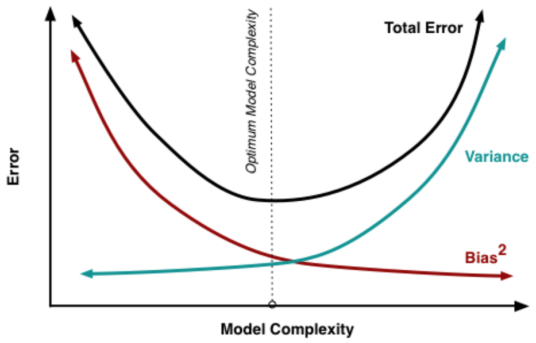

Figure 3. The bias-variance tradeoff, expressed graphically.

A neural network provides the ideal structure to test a network of ensemble learners. Since nodes are trained independently of other nodes with replacement in drawing training sets from the data, we are able to capture a larger representation of our data. Neural trees use a similar approach, where each node in the tree is a small single output neural network (Stromberg & Isaksson, 1991). Secondly, the choice of activation function is extremely flexible — in our case, we chose to use a classifier instead of a function. This resulted in several other issues, which we discuss in our later section on methodology.

[2]Obtained from AnalyticsVidhya
[3]Obtained from AnalyticsVidhya

1.2. Problems with Neural Networks

While neural networks have been widely successful, the model still has its faults, and often requires significant tuning and data curation to be effective. In terms of the classical bias/variance tradeoff, neural network models exhibit extremely low bias, at the cost of high variance which introduces a high possibility to overfit to the data during training and testing. Secondly, the interpretability of the trained model is very low. Since the model relies heavily on calculations and function outputs from the hidden layers, it is difficult to discern what feature values and/or features are important towards the final output.

A much more recent variation of the classical neural network is the deep neural network. While deep neural networks have achieved incredible results in recent published literature, there are two key limitations. Firstly, deep neural networks demand significant amounts of labeled data for training. Labeled data is expensive to create, and even in the Big Data era, when data appears to be plentiful, the ratio of labeled data to unlabeled data is extremely low (Zhu & Ghahramani, 2002). Secondly, the amount of parameter tuning required to fully take advantage of the power of deep neural networks requires a high level of expertise to do precisely. These are large scale, complicated models, and the amount of resources and specialized knowledge required to run them effectively at scale is not trivial.

1.3. Related Work on Neural Networks in conjunction with other classifiers

The flexibility and scalability of neural networks has made it an attractive option for optimization of well-studied algorithms and complex classification tasks in modern machine learning research. Fusing successful algorithms in machine learning with the structure of neural networks has had significant success. Zhou 2017 proposed a deep neural forest with a "cascade forest structure", which was developed to compete with deep neural networks over a large range of applications (Zhou & Feng, 2017). The gcForest algorithm described in Zhou was advantageous to its deep counterpart due to the significantly reduced amount of training data required to create the model (Zhou & Feng, 2017). Our model takes inspiration from this study, in how we enforce diversity and independence in the nodes and informed feature selection through the construction of our base classifiers.

Neural networks have been applied with many other algorithms successfully. One of the most successful examples is the research behind support vector networks, which were developed for binary classification tasks, and aimed to map the input vector into a much higher dimensional space before drawing the decision boundary (Cortes & Vapnik, 1995). Back-propagation through the weights of a neural network has been used as a technique to find the optimal representations for k-nearest neighbor models (Zoran & Blundell, 2017). In both cases, classifiers have been shown to perform well in the neural network structure, and prior work on classifer-based neural networks has provided the background necessary to proceed with this study.

2. Approach

2.1. Architecture

Out of all the popular machine learning techniques, neural networks may take the cake for the amount of hyperparameters and general design choices available for their implementation. We knew that finding the absolute best architecture for our neural forest would be extremely difficult, and more importantly would likely not be the best architecture for all learning problems. Not only are there lots of these settings to tune, but the choice of how many layers there are and how many nodes are in each layer yields infinitely many possible decisions. To this end, we decided to start with an input layer and an output layer, fully connected. Figure 4 below shows the general sketch.

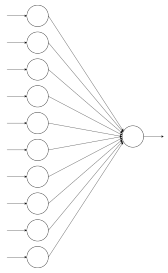

Figure 4. Basic Neural Forest Architecture

Each circle represents a node in the network and each arrow represents an edge connecting nodes, or an input or output. Our network differs from classical neural networks in that while classical neural networks represent a node as an activation function triggered by the sum of its inputs, our nodes represent an already existing classifier! In particular, a node could have a decision tree or random forest in it which looks at the input vector and outputs a decision about it to be fed onward along the network.

2.2. Baseline

Before implementing our proposed neural networks, we thought it would be a good idea to implement simpler versions of the classifiers we were going to modify and blend together. We used Scikit-Learn and Keras libraries for our basic classifiers. Random forests were implemented with Scikit-Learn's module. We tried a number of different types of decision trees: DecisionTreeClassifier, which

splits on the feature with the highest information gain; and ExtraTreeClassifier, which chooses a random split on each feature, and then chooses the best information gain out of these prefabricated splits for more stochasticity. The regular neural network was created using the Keras implementation, with a few pretty standard settings for the optimizer and loss function. We tested a few different values for some of the standard parameters for these classifiers, then trained each of them on both of our datasets, the details of which will be discussed in more detail in the later section on experimental methodology.

2.3. Algorithms

2.3.1. High Level Overview

An early concern was that, given that each node is an algorithmic classifier rather than a mathematical function, backpropagating information would be extremely difficult. Backpropagation is an important algorithm for neural networks because it allows the network to adjust what it cares about by getting an idea of how much each node contributes to the error in the prediction. For this, it relies on the functions internal to the nodes to be mostly differentiable (some activation functions such as ReLU are not differentiable everywhere, but in practice can be treated as being differentiable). It was unclear how to take the derivative of what is going on inside a decision tree — much less a random forest — so to defer this obstacle to further research, we put a logistic regression classifier in the output node. Since logistic regression optimizes weights, this acted towards the same ends as backpropagation, but also forced us to only have one layer of decision trees. Our network runs as follows. First, the first layer is trained on the data X and the labels y. Then, all of X is passed through the first layer, and kept as another dataset. The logistic regression node is trained on this and the labels y. The weights of the logistic regression node are dependent upon the output of X, with a step size of 0.0001. Our implementation of the network relies on the algorithms below.

2.3.2. Generation

First, the network is generated by calling Algorithm 2 below for as many layers as the network will have.

Algorithm 1 Generate Layer

Input: data X, labels y, classifier clf, int $numNodes$
Initialize $layer = []$
for $i = 1$ **to** $numNodes$ **do**
  Initialize $c_i = clf.train(X, y)$
  Add c_i to $layer$
end for

For our implementation, each new classifier which would make up the layer was created using an ExtraTreeClassifier and trained on a set of examples bagged from the original input data. This was done in order to diversify the overall classifier through what each node cares about.

2.3.3. Prediction

For training of the network, the classifier inside each node predicts the label of the incoming data. Algorithm 3 oversees the repeated application of Algorithm 2 on the data.

Algorithm 2 Layer Predict

Input: data X
Given: list of nodes $layer$
Initialize $predictions = []$
for $datum$ **in** X **do**
  Initialize $prediction = []$
  for $node$ **in** $layer$ **do**
    Initialize $y = node.predict(datum)$
    Add y to $prediction$
  end for
  Add $prediction$ to $predictions$
end for
return $predictions$

Algorithm 3 Network Predict

Input: data X
Given: list of $layers$
for $layer$ **in** $layers$ **do**
  $data \leftarrow layer.predict(data)$
end for
return $predictions$

Since each layer in the network is trained on the new data generated by passing the original data through all the preceding layers, this network can take a long time to train for arbitrary architectures, especially if each node represents a random forest with many estimators. This was a fairly trivial part of our process, since the data only needed to be passed through one layer before being used to train the final logistic regression layer.

3. Experimental Methodology

3.1. Datasets

We used two datasets to train and evaluate our models: the Wisconsin Breast Cancer dataset from the UCI Machine Learning Repository and the Higgs Boson dataset from the CERN Kaggle competition. Our model is only capable of binary classification, so the two datasets we selected are binary classification problems. The breast cancer dataset contains 569 instances with 30 real-valued features, while

the Higgs Boson dataset contains 250,000 instances with 30 real-valued features.

3.2. Preprocessing

Our experiments did not require much preprocessing. The breast cancer dataset has already been cleaned of missing values, so the only preprocessing required was to separate the test and training sets. We set aside 100 instances to use as the test set, leaving 469 instances for training. The Higgs boson dataset had already been preprocessed for the Kaggle challenge, where instances with negative weights were removed and the normalization of the signal and backgrounds were slightly altered (Adam-Bourdarios, 2014). We set aside 50,000 instances for the test set and used the remaining 200,000 for the training set.

3.3. Experiment Design

To evaluate our algorithm, we compared its performance against other typical machine learning classifiers. We chose to use decision trees, random forests, and neural networks as our baseline evaluators. For each of these baseline evaluators, we compared the performance over a set of parameters and selected the best parameters to generate a model to be used for testing.

3.3.1. Decision Tree

We used Scikit-Learn's DecisionTreeClassifier with default values for all of the parameters except *max_depth*. We tested trees with *max_depth* values of [5, 10, 30, 50].

3.3.2. Random Forest

We used Scikit-Learn's default Random Forest with a *max_depth* of 1 to improve training time. We tested $n\_estimators$ values of [5, 10, 30, 50].

3.3.3. Neural Network

We used Keras to generate a simple, sequential neural network for binary classification. Our goal was not to find the most optimal network, but rather to generate networks with comparable architectures to that of our model. The neural networks were set up as follows:

- 1. Dense Layer with sigmoid activation and Truncated Normal kernel intializer.
- 2. Dropout Layer with dropout rate of 0.01.
- 3. Dense Layer with two nodes and a softmax activation for prediction.
- To compile we used RMSProp as the optimizer and binary cross entropy as the loss function.

The Truncated Normal kernel initializer draws random initial weights for the layer from a normal distribution with mean=0 and standard deviation=0.05, and resamples weights that are more than two standard deviations from the mean. Truncated Normal is generally the preferred alternative to initializing all the weights as zeroes. RMSProp is an optimizer that improves on regular Stochastic Gradient Descent by scaling the learning rate of the by the magnitude of the gradient (Hinton & Swersky, 2012). This change helps compensate for the vanishing gradient problem of Stochastic Gradient Descent by effectively normalizing the step size during backpropagation. We tested [5, 10, 30, 50] as the number of nodes for our input layer, and trained each network for 100 epochs.

3.3.4. Decision Tree Neural Network

For our implementation, we tested neural forests that use Scikit-Learn's ExtraTreeClassifiers as nodes. For these networks, we used tested layers with [5, 10, 30, 50] nodes.

3.3.5. Random Forest Neural Network

We also tested neural forests that use random forests as the internal node classifier. For these networks we used Scikit-Learn's default parameters and *max_depth* of 1. We generated models that use random forests with [5, 10, 30, 50] *n_estimators*.

3.4. Evaluation

For evaluating our models, we decided to employ a variety of different metrics. We first trained each model using cross validation with 75-25 train-test split. For the baseline evaluators, we selected the models the best accuracy on the tuning set. We generated confusion matrices for each of the models to visualize their performance on the two datasets. To measure the performance of our models we used three standard metrics for binary classification problems: precision, recall, and accuracy.

4. Results

Tables 1 and 2 exhibit a cross-comparison of classifiers' performance on the Higgs boson and breast cancer datasets, respectively. The naming scheme was as follows for the classifiers we implemented: <core classifier>NN<parameter value>. For DTreeNN, the parameter was the number of nodes; for RFNN, the number of nodes was fixed at 30 and the parameter was the number of estimators in each node.

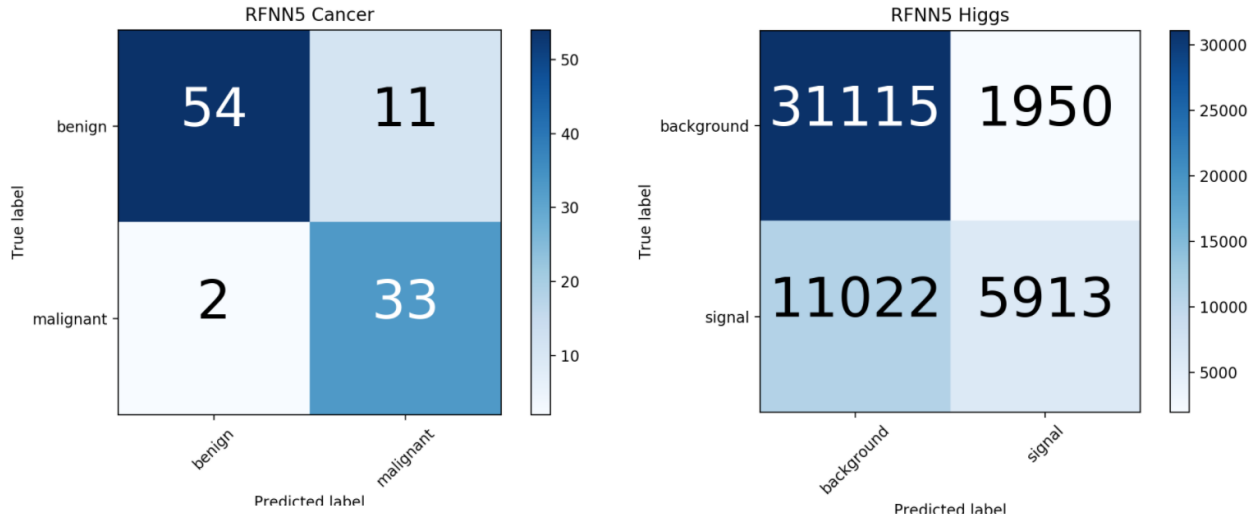

Figure 5. The confusion matrices of the random forest neural network on the Cancer dataset and the Higgs boson dataset. Both of these models had 5 estimators inside each node of the first layer.

Table 1. Accuracy, Precision, and Recall for the various models on the Higgs boson dataset.

CLASSIFIER	ACCURACY	PRECISION	RECALL
DTREENN5	0.66	0.50	0.61
DTREENN10	0.67	0.93	0.01
DTREENN50	0.67	0.52	0.54
RFNN5	0.74	0.75	0.35
RFNN10	0.71	0.84	0.17
RFNN50	0.69	0.88	0.09
DECISION TREE	0.77	0.68	0.65
RANDOM FOREST	0.74	0.72	0.37
NEURAL NET	0.76	0.69	0.55

Table 2. Accuracy, Precision, and Recall for the various models on the breast cancer dataset.

CLASSIFIER	ACCURACY	PRECISION	RECALL
DTREENN5	0.84	0.69	0.97
DTREENN10	0.78	0.61	1.00
DTREENN30	0.90	0.79	0.97
RFNN5	0.87	0.75	0.94
RFNN10	0.86	0.71	1.00
RFNN50	0.83	0.67	1.00
DECISION TREE	0.88	0.79	0.89
RANDOM FOREST	0.80	0.64	1.00
NEURAL NET	0.85	0.71	0.97

4.1. Metrics for the Best Classifier

Many of the models yielded relatively low recall scores on the Higgs boson dataset, which can likely be attributed to the imbalanced size of the two classes. Because there are almost twice as many negative instances as positive instances (164333 vs 85667) and the fact that we used accuracy as our metric for selecting the best model, we ended up learning models that frequently yield many negative predictions. This pattern is especially evident with the DtreeNN50 classifier, which had a recall score of just 0.01 (predicting 49471 negative and only 259 positive). However, this particular aggregate could just be an outlier caused by the random nature of the ExtraTreeClassifiers used in the network.

However, the breast cancer dataset is also somewhat imbalanced, with 357 negative and 212 positive instances, but all of the model yielded high recall scores. One possible explanation could be that the Higgs boson classification problem is simply a more difficult classification task. If the features in the Higgs boson challenge are not very indicative of the true label, then the learned models will tend to just predict the more frequent class.

4.2. Summary

Figures 5 and 6 show the performance of our best networks for each dataset. On both datasets, our Random Forest Neural Net performed best with five estimators. For the Decision Tree Neural Network, we found that on the breast can-

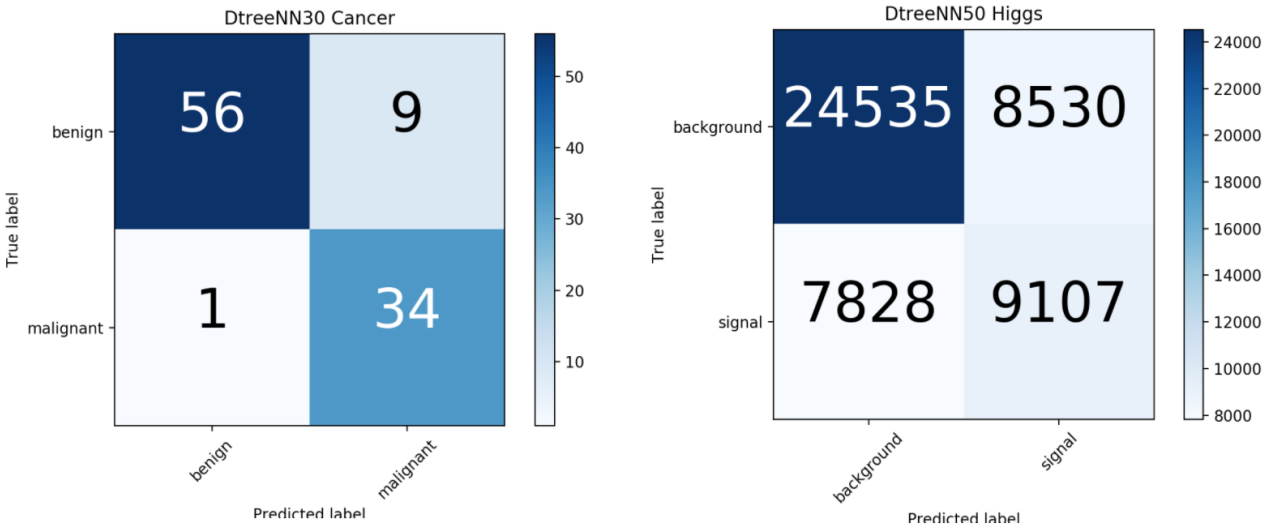

Figure 6. The confusion matrices of the decision tree neural network on the Cancer dataset and the Higgs boson dataset. The decision tree model run on the cancer dataset had 30 nodes and the decision tree model run on the Higgs boson dataset had 50 nodes.

cer dataset the model performed best with 30 nodes. On the Higgs boson dataset, the 50 node model yielded the highest accuracy.

5. Discussion

5.1. Interpretability

Neural networks as a general paradigm have been recently notorious for how hard it is to interpret their decision criteria, and ours is no different. Early in the course of this project, we observed that since the data fed into even the second layer is the aggregate of the decisions of the first layer, the input to the second layer looks wildly different from the original input data. In particular, reconstructing the original data given the intermediate set would probably be impossible in all but the simplest cases. Whether this high degree of obfuscation would lead to a more plastic decision space or just confuse the network was a driving question for this research.

5.2. Efficacy of our Network

With even a brief perusal of the data in tables 1 and 2, it is obvious that our network performs much better than randomly guessing. The accuracy of each of our implemented networks is comparable to their constituents. Our decision tree neural network is about 10% less accurate than Scikit-Learn's decision tree on the Higgs boson dataset, and 5% less on the breast cancer dataset. Between our decision tree neural network and Keras' vanilla neural network, our accuracy is again about 10% less on the Higgs boson dataset, and only about 3% lower for the breast cancer dataset. Moving on to a comparison between our random forest neural network's accuracy and that of the Keras' neural network, we see the regular neural network outperforming ours by 5% for the Higgs boson dataset, and about equal performance on the breast cancer dataset. Finally, our network is outperformed by Scikit-Learn's random forest by about 2% for the Higgs boson dataset, but does about 5% better on the breast cancer dataset. One drawback of our network is the long time it takes to generate, but in terms of accuracy, our classifier aggregates are competitive with the classical techniques which make them up.

5.3. Further Research

As mentioned above, we left many aspects of this project open to further elaboration and research. The particulars of this project entailed that we didn't have time to test the vast majority of parameters for the implementation of our neural network. We recommend the following areas as a starting place for any research team looking to further our inquiry.

5.3.1. Choice of Classifier

We chose decision trees as the atomic classifier of this network both because their decision space seemed most conducive to propagating information through the network and because the hierarchy of attention given to each feature seemed in line with the paradigm of weights along the

edges of a neural network. However, we are not sure how crucial the choice of node-internal classifier is to the overall performance of the network. It would be interesting to test whether the network undergoes significant changes in performance depending on the classifier chosen. The ensemble nature of this aggregate implies that a stable classifier such as Naive Bayes would not be a good candidate for inside nodes, since it probably wouldn't experience much of an improvement on its hypothesis space. On the other hand, the emergent complex system due to the neural aspect of the network could create more flexibility in learning regardless of classifier.

5.3.2. Architecture

Although our model only used two layers, a natural extension of our network would be to create a multilayered implementation. One of the main hurdles for a multilayered version of our model would be figuring out a scheme for backpropagation between hidden layers. This would likely depend heavily on the choice of classifier in each node. Potentially, multiple classifiers could be used, distributed among nodes or layers per the user's choice. This would imply different "derivatives" for each classifier.

5.3.3. Multi-class Prediction

Because our network uses a logistic regression classifier for prediction, it is currently only capable of binary classification. An obvious extension would be to modify the network to be capable of multi-class prediction. One way this could be achieved would be to have the prediction layer contain one-vs-all logistic regression nodes for each of the possible class and then have a final layer that predicts the highest probability class.

6. Conclusion

Our implementations of neural forests are much better than random classification. In terms of accuracy, our implementation is comparable with the regular models. However, overall, the decision tree neural network and the random forest neural network models performed relatively poorly compared to their constituent classifiers for the amount of extra computation that goes into training. Our network is able to outperform a few algorithms (namely random forests) in some places, but the overall performance increase is not huge. However, the novelty of the approach and the extra flexibility the neural forest architecture provides to base classifiers is still intriguing, and the results of our networks were definitely comparable to the base algorithms. We can definitively say that further research to improve these neural network models is worthwhile.

Acknowledgments

We extend our deepest gratitude to Dr. Ameet Soni, who was an invaluable resource for helping us brainstorm ways to overcome some of the obstacles we faced and foresee problems with our implementation.

References

Adam-Bourdarios, et al. Learning to discover: the higgs boson machine learning challenge. Technical report, CERN, 2014.

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

Hinton and Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

Hinton, Srivastava and Swersky. Lecture 6a: Overview of mini-batch gradient descent. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2012.

Kuncheva, Ludmila and Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, May 2003.

Rowley, Baluja and Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

Stromberg, Zrida and Isaksson. Neural trees-using neural nets in a tree classifier structure. In *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pp. 137–140, 1991.

Zhou and Feng. Deep forest: Towards an alternative to deep neural networks. *arXiv*, 2017.

Zhu and Ghahramani. Learning from labeled and unlabeled data with label propagation, June 2002.

Zoran, Lakshminarayanan and Blundell. Learning deep nearest neighbors representations using differentiable boundary trees. *arXiv*, 2017.