PROJECT 5
FEEDFORWARD NEURAL NETWORKS

DAVID ATLAS

Abstract. This paper will introduce feed-forward neural networks, a technique for learning non-linear functions. The backpropogation algorithm will be used in conjunction with gradient descent. Networks of 0, 1 and 2 hidden layers will be applied to several different real-world classification problems. The size of the hidden layer will be tuned using a validation set. A sigmoid activation function will be used to introduce non-linearities into the network. A softmax classifier will be used to train a multi-net classifier. It is found that the networks outperform a naive baseline in most problems, although require substantial tuning effort to converge to a good solution (and in some cases do not converge as desired).

1. Problem Statement & Hypothesis

Feed-forward neural networks are a popular technique for estimating non-linear relationships between a set of inputs, and the densities of a corresponding class label or regression target. In a feed-forward neural network, an affine transformation is conducted on an input, at which point a non-linear function (which will be called an activation function) is applied. This is the basic unit of a network. Many of these units can be combined, with the effect being a flexible model that can learn non-linearities and feature interactions.

Therefore, the hypothesis is that the networks should outperform naive baselines across all the real-world experiments run in this paper, as it is known from previous experiments that a simple linear classifier can outperform baseline on these problems. As such, the feed-forward network has *at least* as much capacity as those models, and should therefore perform equally or better (as measured by classification accuracy).

However, the additional cost of the expanded model capacity is that the networks can be difficult to train. They are sensitive to the learning rate, number of iterations and size of each of the units. Therefore, it would not be wholly unexpected if it were to be too difficult to find a network that outperforms a simpler linear model on problems with classes that are largely linearly seperable, especially where there is not a lot of data.

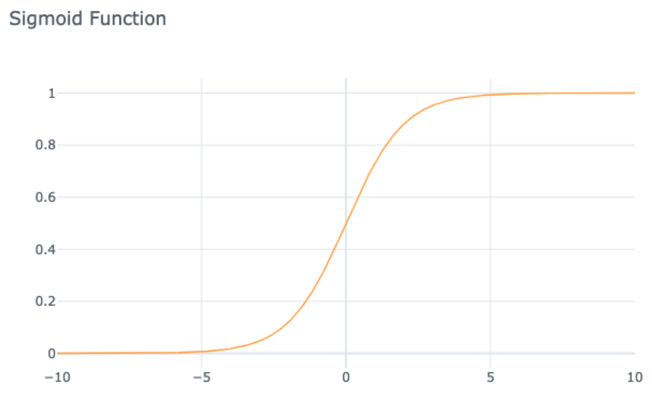
In the experiments below, 3 network configurations are used. A network with 0, 1 and 2 hidden layers will be applied to each problem. It is expected that the network with no hidden layers will perform in-line or better that the networks with hidden layers on problems with linear seperability. If an experiment presents a non-linear boundary between classes, it is expected that the hidden layer networks will outperform. It is also expected that the network with no hidden layer will be easier to tune, and therefore will have fewer instances of non-convergence in real-world experiments.

2. Description of Algorithms

Single Perceptron Unit. The networks discussed in this paper are all composed of simple, small units. Each unit takes an input X and generates an output z via the following function:

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$
$$z = \sigma(XW)$$

where W is a matrix with the same number of rows as X has columns. $\sigma(x)$ is the sigmoid activation function, shown below in Figure 2. The sigmoid function plays two roles in the network. First, it introduces non-linearity. This is important, because several sequential linear transformations create a linear transformation, meaning that the network would only be able to create linear functions. Adding in the sigmoidal activation changes that. Second, the function maps from an unbounded space to the $[0, 1]$ interval. When training a classifier, the output must be interpreted as binary. The output of the sigmoid function can be interpreted as the probability of a positive prediction. This is better than a strictly binary label because the interpretation is naturally probablistic. It is also easier to train relative to a binary step function because the gradient is



continuous and differentiable across the real number line. It is steepest around the decision boundary (where $x = 0$). If $x > 0$, then $\sigma(x)$ can be interpretted as indicating a positive class label.

Network Topology. Now that the basic unit of the network has been established, there are two parameters that must be chosen within the network[3].

The number of additional units to include in the network (which will be called the hidden layers) is crucial in defining the capacity of the network. As the number of units grows, the capacity of the network increases, all else equal, which reflects a change in the representation bias. However, the corresponding search space becomes bigger, and the selection bias plays a large role in whether the network converges to a performant solution.

Beyond the number of hidden layers, the size of each hidden layer must be chosen. Above, W was defined as a $m \times k$ (where X is $n \times m$). This leaves k as a parameter. In the experiments conducted here, k will be tuned over a validation set. Note that the effect of a larger value of k is similar to adding more units in the network - larger values will increase the representation capacity, but will create more parameters (and thus a larger search space). It is sensible for the applications here to restrict k to be between the number of input features and the number of output features, although not unreasonable to move beyond that range. Choosing $k > m$ allows for a higher dimensional representation (similar to the representation used in SVMs to achieve non-linear decision boundaries), but it not strictly necessary as the decision boundary here is non-linear. Choosing k less than the number of output classes creates a pinching effect (similar to that of an autoencoder), but for this problem space, indicates an unnecessary reduction of the representation space that is not desirable.

There is also a relationship between the two parameters. A network might have fewer large hidden layers, or many small hidden layers. Fewer layers may be easier to train, but many layers can give more capacity with fewer parameters. Both should be chosen based on the problem.

Backpropogation. Given the network defined above, the challenge is in tuning the weight matrices W_h for each of the h units in the network. As in past papers, gradient descent will be used, but the challenge remains in calculating the gradient[5] across all the layers. Given the loss function

$$E(W, X) = (O - Y)^T (O - Y),$$

where $o_i \in O$ is the output of the network given input row X_i, and $y_i \in Y$ is a vector of targets corresponding rowwise to X, the derivative of the loss relative to the output of the network is

$$\frac{dE}{dO} = 2(O - Y).$$

At this point, the chain rule must be applied to find the derivative of the loss relative to the weights in the final layer. The derivative of the sigmoid function is

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)),$$

and so

$$\frac{dE}{dW_H} = \frac{dE}{dO}\frac{\partial O}{\partial W_H} = (O - Y)\sigma(X)(1 - \sigma(X))X.$$

Next, the derivative for each of the preceding units must be found. First, δ_i is defined as

$$\delta_i = -\frac{\partial E}{\partial O_i},$$

where O_i is the output of the i^{th} layer. Using the chain rule, this can be written as

$$\begin{aligned}\delta_i &= -\frac{\partial E}{\partial O_{i+1}}\frac{\partial O_{i+1}}{\partial O_i}\\ &= \delta_{i+1}\frac{\partial O_{i+1}}{\partial O_i}\\ &= \delta_{i+1}W_{i+1}\frac{\partial O_i}{\partial X_i}\\ &= \delta_{i+1}W_{i+1}O_i(1 - O_i).\end{aligned}$$

Practically, the delta rule will be leveraged. Rather than recalculating the derivative for each unit, the derivative can be expressed as the cumulative effect of all the units downstream from a given unit. This cumulative effect will be denoted δ_j for the j^{th} unit in the network:

$$\delta_j = \delta_{j+1}W_{j+1} \circ O_j(1 - O_j),$$

where $A \circ B$ denotes the Hadamard (element-wise) product of A and B, and O_j indicates the output of the j^{th} unit. This is where the term *backpropogation* comes from - the derivative of the error is propogated backwards through each of the preceding units, all without having to recalculate the individual quantities, by accumulated δ_i throughout the network.

Gradient Descent. Having presented the delta-rule, calculating the gradient updates is easy.

$$\Delta W_h = \eta\delta_h X_h,$$

where η is the learning rate, δ_h is as defined above (the sum over all of the downstream δ_i), and X_h is the input to the h^{th} unit (the output of the $h - 1^{th}$ unit or the original set of features if $h = 1$).

This process will repeat iteratively until either some convergence criteria is met, or the a specified number of iterations is reached.

3. Experimental Approach

For each of the 5 real-world experiments conducted, networks with 0, 1 and 2 hidden layers were trained. The size of the hidden layers and the learning rate were both tuned using a validation set, and chosen based on which set minimized the out-of-sample classification error. A stratified 5-fold cross-validation scheme was used to estimate the out-of-sample error. All problems were trained using a multi-net approach, where each class label has a corresponding output, and a softmax function was used to calculate the error relative to a one-hot encoding of the class labels.

4. Experimental Results

Breast Cancer Data. The first experiment conducted involved the classification of a breast tumor as malignant or benign[7]. Rows containing null values were dropped, as they are a small portion of the data. The features were also mean-centered at zero and scaled by the standard deviation. As mentioned above, a cross-validation approach was used to pick the size of the hidden layers and the learning rate. Batches of size 48 were, providing some stability in the learning process. The results of the experiments can be found in Table 4.

Model	Size of H1	Size of H2	Mean Accuracy
No Hidden Layer	NA	NA	97%
One Hidden Layer	4	NA	97%
Two Hidden Layers	4	8	97%
Naive Baseline	NA	NA	65%

Table 1. Breast Cancer Classification Results

The model converges to a solution in line with previous linear classifiers. This solution vastly outperforms the baseline model.

Glass Data. The Glass data[2] experiment involved classifying the source of broken glass based on characteristics of the broken glass. The features were also mean-centered at zero and scaled by the standard deviation. As mentioned above, a cross-validation approach was used to pick the size of the hidden layers and the learning rate. Batches of size 48 were, providing some stability in the learning process. The results of the experiments can be found in Table 4.

Model	Size of H1	Size of H2	Mean Accuracy
No Hidden Layer	NA	NA	46%
One Hidden Layer	7	NA	34%
Two Hidden Layers	3	5	35%
Naive Baseline	NA	NA	36%

Table 2. Glass Classification Results

The network did not converge to a performant solution with hidden layers added. There is not a lot of data for each class, which may prevent convergence. The model with no hidden layers does converge to a solution in line with previous work.

Soybean Experiment. The next experiment run was to classify the type of rot exhibited by a soybean given characteristics about the bean[4]. As above, cross-validation was used to determine the best size of both of the hidden layers and the learning rate. A batch size of 48 was used again. The network with no hidden layer performed quite well, while the networks with hidden layers perform in line with a naive baseline. The results can be found in Table 4.

Model	Size of H1	Size of H2	Mean Accuracy
No Hidden Layer	NA	NA	98%
One Hidden Layer	5	NA	36%
Two Hidden Layers	7	9	36%
Naive Baseline	NA	NA	36%

Iris Experiment. The Iris dataset[1] involves classifying the type of flower based on measurements of the flower. As above, cross-validation was used to find the size of the hidden layers and the learning rate. A batch size of 48 was used. The networks had some trouble converging to extremely performant results, but all were able to outperform the baseline naive model on average. The experiment results can be found in Table 4.

Model	Size of H1	Size of H2	Mean Accuracy
No Hidden Layer	NA	NA	74%
One Hidden Layer	3	NA	67%
Two Hidden Layers	3	3	66%
Naive Baseline	NA	NA	33%

House Votes Experiment. The House Votes dataset[6] involves predicting the party affiliation of congresspeople based on their votes on several key issues. As above, cross-validation was used to tune the size of the hidden layer and the learning rate. The votes were one-hot encoded, with null values being encoded as a boolean value indicator. The experiment results can be found in Table 4.

Model	Size of H1	Size of H2	Mean Accuracy
No Hidden Layer	NA	NA	90%
One Hidden Layer	4	NA	94%
Two Hidden Layers	3	4	96%
Naive Baseline	NA	NA	61%

5. Algorithm Behavior

The performance of the neural networks are largely in line with the enumerated hypotheses above. The networks are able to outperform baseline in 3 of the 5 real-world problems viewed. As the class discriminants are mostly linear, this is as expected. The network did not materially outperform the logistic regression classifier of previous work, which is likely more a statement about the form of the class discriminants for these datasets than of the effectiveness of the network.

The biggest difficulty in all this is finding a performant solution within the hypothesis space of the network. Initially, the networks were tuned by hand, manually finding learning rates, layer sizes, and number of training iterations to use. This was untenable, as it took too long, and wasn't stable. This spurred a move to an automated grid search approach, which succeeded in finding performant networks for most of the problems. However, it is quite time consuming to run, as each proposed network configuration takes 1-3 minutes to train.

One behavior that stands out as useful in training the networks is the relationship between loss and number of gradient updates applied. This can give insight into the appropriate number of iterations, as well as the learning rate. If the loss bounces around without showing improvement, the learning rate is too large. If it shows little movement, the learning rate is too small. Training can be stopped when the loss stops materially decreasing.

6. Summary

This paper introduced feed-forward neural networks, as well as the backpropogation algorithm that helps train them. The networks outperform naive baseline models in situations where they are able to converge to a good solution. It is difficult and time consuming to tune the networks to such a good solution. They would be easier to train and more outperformant relative to linear methods if the boundary between classes was more non-linear than in these problems. However, given the largely linear boundaries in the problems examined, the performance is not beyond that of a simplier logistic regression.

References

[1] R.A Fisher. Iris. URL: https://archive.ics.uci.edu/ml/datasets/Iris.
[2] B. German. Glass identification data set. URL: https://archive.ics.uci.edu/ml/datasets/Glass+Identification.
[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
[4] R.S.n Michalski. Soybean (small) data set. URL: https://archive.ics.uci.edu/ml/datasets/Soybean+%28Small%29.
[5] Hinton G.E. Rumelhart, D.E and R.J. Williams. Learning representations by back-propogating errors. *Nature*, pages 533–536, 1986.
[6] Jeff Schlimmer. Congressional voting records data set. URL: https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records.
[7] Dr. WIlliam H. Wolberg. Breast cancer wisconsin (original) data set. URL: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29.