COSC 528 Project 4: Backpropagation

Elliot Greenlee

November 22, 2017

Abstract

This report covers the design and implementation of a general artificial neural network. Neural networks of a variety of structures are optimized for spam detection in email data. Additionally, dimensionality reduction by principal component analysis is applied to measure the speedup and accuracy effects. From this analysis, it is observed that the best accuracy results come from a learning rate around 0.1, a variance for the Gaussian weight initialization of 0.2, a logistic sigmoid output layer activation function, and a large number of hidden nodes with fewer hidden layers. This structure produced the best result of 95.3% accuracy on 20% of the data.

1 Introduction

In every brain, neurons connect together, working in concert to produce behaviours, thoughts, and actions. In the the human brain these neurons develop over time, with more often used paths becoming more influential. This leads to learning, and can be replicated using artificial neural networks. Each artificial neural network is made up of perceptrons that mimic neuron behavior; a collection of inputs feed in to a perceptron, which performs a sum and then undergoes an activation function to determine output. In a neuron, this is the decision to fire or not fire an electrical pulse based on chemical inputs. By combining these perceptrons into multiple layers, new functions can be approximated.

The motivation for neural networks is to learn higher level functions using combinations of linear perceptrons. Given a set of inputs to a perceptron $(x_1, x_2, ..., x_d)$ along with a bias node of value $b = 1$ and a set of weights $(w_1, w_2, ..., w_d)$ connecting each input to the perceptron, the final value is calculated as

$$(\Sigma_{i=1}^{d} x_i * w_i) - w_b * 1$$

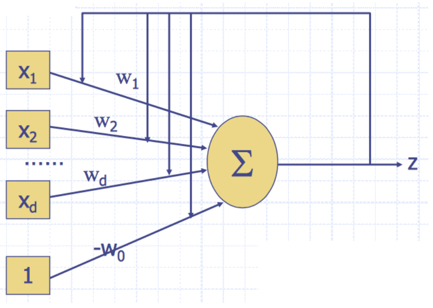

Figure 1: An example perceptron

Unfortunately, these perceptrons are limited to linear combinations of the inputs, which prevents approximation of functions like XOR, which is not linearly separable. To solve this problem, multiple perceptrons can be connected together, with the output of one perceptron being fed into an activation function before leading to the input of another. These activations functions mimic the behavior of activation in real neurons, and are represented by functions like sigmoid. Artificial neural networks vary widely in structure and function.

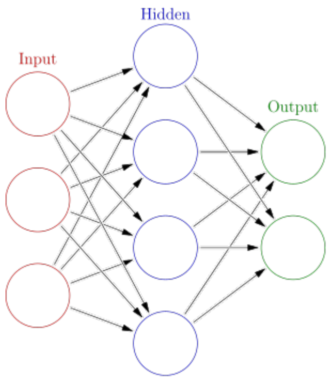

Figure 2: An example neural network

The objective of this project is to achieve the highest possible accuracy on a spam detection problem using an artificial neural network trained using backpropagation. Hyperparameter optimization of the learning rate, weight initialization, structure, and output layer is used to increase performance, and dimensionality reduction is tested. Sections 2 and 3 explain the methods used. Details of the experiments performed can be found in section 4. The work is summarized in section 5.

2 Dimensionality Reduction

Although in theory a greater number of statistically independent features should reduce further and further the error, in practice additional features often lead to worse performance. This occurs either when the wrong model is used, or because the training sample is not infinite, so the distributions are not accurately estimated. This leads to problems with overfitting, where the model works well on the training data but poorly on the testing data, and complexity, where the model takes longer to run at $O(d^2n)$ where d is dimension and n is number of samples. The solution to these problems is dimensionality reduction. Linear methods are used because of their simplicity, projecting high dimensional data onto a lower dimensional space.

One common method is principal component analysis, with the objective of minimizing information loss in a least-squares sense. Principal component analysis attempts to reduce the data while minimizing information loss. One might imagine the major and minor axis of a set of two dimensional elliptical data as in figure 3. Principal component analysis would choose the major axis as the projection direction to preserve information. Because there is no consideration of class information, it is an unsupervised method [2].

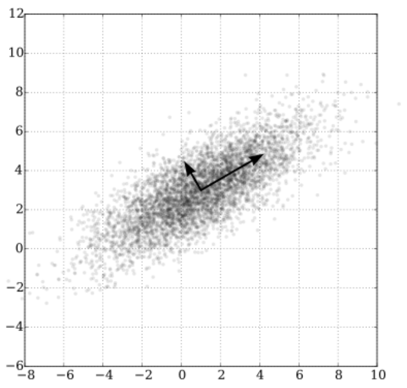

Figure 3: An example of the first two principal components on a graph.

One option for principal component analysis is singular value decomposition, which can decompose any Nxd matrix X into components VAW^T where V is the NxN matrix of the eigenvectors of XX^T in its columns, W is the dxd matrix of the eigenvectors of X^TX in its columns, and A is the Nxd matrix of the q singular values on its diagonal. For A, $q = \min(N, d)$, where the values are square roots of the nonzero eigenvalues of both XX^T and X^TX. The principal component P matrix of shape Nxd can be constructed as $P = (W^TX^T)^T$, where each row of P corresponds to one of the N instances. To reduce, take the first k columns of P [1].

3 Neural Networks

This section starts by defining equations and notation, and along the way describes forward propagation in general and in the specific case of this project. Keep figure 2 in mind. A single bias node of 1 has also been added to the inputs each layer. The number of layers in the network, not including the outputs as a layer, is labeled as L. Each layer includes the starting point inputs $X[l]$, the weights $W[l]$, an activation function $X[l+1] = f[l](Z[l])$, where $Z[l] = W[l]X[l]$, the dot product of the inputs and weights. This proceeds from layer 0, starting with the original inputs, up to the output $y = X[L]$. In the case of this project, the activation function is the logistic sigmoid up until the last layer, which could have a linear, softmax, or logistic sigmoid activation.

$$linear = f(x) = x$$

$$softmax = f_j(x) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; for \; j = 1, ..., K$$

$$logistic \; sigmoid = f(x) = \frac{1}{1 + e^{-x}}$$

With neural networks, the challenge is to find weights that minimize the error between the expected value $y\_$ and true output y. To accomplish this, backpropagation is used, based on gradient descent. A set of initial weights are chosen, and then weights are updated by removing the error contribution from the weight value. The error function for this project is cross entropy, where i in the number of classes.

$$cross \; entropy = f(y\_, y) = \sum_i y\__i \log(y_i)$$

Consider the error space to be determined by the weights. As the weights are updated, we travel around the error space, either increasing or decreasing in value. Eventually, we hope to find the global minimum error. The learning rate is a limiter on step size taken by each update of backpropagation. Take too small of steps, and it will take a long time to get to the correct value. Also, it might be impossible to get out of local minima along the way. Take too large of steps, and the risk of divergence, where the global minimum is stepped over repeatedly and never converges to the correct value, increases. The optimal learning rate and update direction would cause the algorithm to take one step and arrive perfectly, but this is impractical. Another consideration is the initial position in this error space. Different initializations of the weight contribute to different trajectories towards the global minimum. One initialization might place the starting point right inside of a local minimum which can't be escaped.

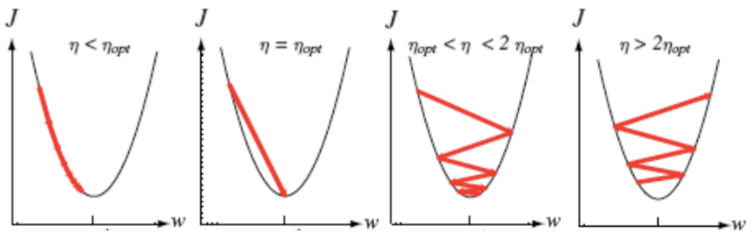

Figure 4: Learning rate possibilities

In backpropagation, the objective is to calculate the contribution in error change caused by the weights of each layer. Since each weight value in each layer in calculated in the same way, we can consider values as vectors and matrices, and program the same way. The chain rule is used to go back along the chain of equations, which at this point are unknown. First the special case of the output layer is considered.

$$\frac{\delta E}{\delta W[L-1]} = \frac{\delta E}{\delta y}\frac{\delta y}{\delta Z[L-1]}\frac{\delta Z[L-1]}{\delta W[L-1]}$$

The change in error is also considered with respect to the previous layer inputs.

$$\frac{\delta E}{\delta X[L-1]} = \frac{\delta E}{\delta y}\frac{\delta y}{\delta Z[L-1]}\frac{\delta Z[L-1]}{\delta X[L-1]}$$

Next, the rest of the layers are considered iteratively from the end of the network to the beginning.

$$\frac{\delta E}{\delta W[l]} = \frac{\delta E}{\delta X[l+1]}\frac{\delta X[l+1]}{\delta Z[l]}\frac{\delta Z[l]}{\delta W[l]}$$

$$\frac{\delta E}{\delta X[l]} = \frac{\delta E}{\delta X[l+1]}\frac{\delta X[l+1]}{\delta Z[l]}\frac{\delta Z[l]}{\delta X[l]}$$

These equations can be applied to any error function, activation function, or output function. In the case of this project, cross entropy error is used with three separate output functions. Luckily, cross entropy has the nice property of returning the same value for all three output functions in the backpropagation.

$$\frac{\delta E}{\delta Z[L-1]} = y\_ - y$$

Here are the rest of the specific derivative results that can be multiplied as above.

$$\frac{\delta X[l+1]}{\delta Z[l]} = logistic\ sigmoid' = f'(Z[l]) = f(Z[l]) * (1.0 - f(Z[l]))$$

$$\frac{\delta Z[l]}{\delta X[l]} = W[l]$$

$$\frac{\delta Z[l]}{\delta W[l]} = X[l]$$

4 Experiments and Results

These experiments are implemented in Python using the numpy, pandas, scipy, and scikit-learn libraries. For each accuracy and speed measure, five-fold cross validation on 80% of the data is used to test the hyperparameters. For convergence, 80% of the data is used to train. The best methods in terms of accuracy are applied to the testing data.

4.1 Data

The data for this report is character and word level features of emails measured in terms of lengths and frequencies. There are 48 real word frequency features, 6 character frequency features, 1 real average length of capital letters, 1 integer longest length of capital letters, 1 integer sum of capital letters, and a class identification of either spam (1) or not spam (0). The original data comes from a csv file and label file with data information. There were fortunately no missing values. This all totals to 4601 examples with 57 features each and a class value. The data is shown in figure 5 reduced to two dimensions by PCA. The spam examples are colored red and the non spam examples are colored green. This information was provided as a csv file without source as part of COSC 528, Fall 2017 at the University of Tennessee, Knoxville [3], but originated from the UCI Machine Learning Repository `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

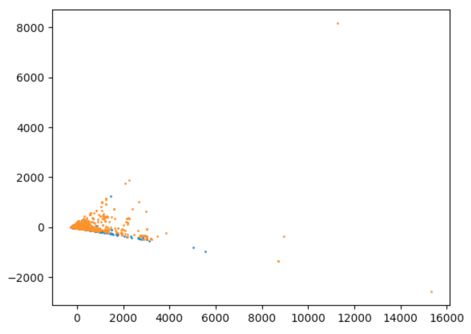

Figure 5: Dataset reduced to two principal components. Spam examples are shown in blue and nonspam examples are shown in orange.

4.2 Performance Metrics

For this project, two performance metrics are reported: the confusion matrix and accuracy. The confusion matrix reports true negatives tn, false negatives fn, false positives fp, and true positives tp. Accuracy is calculated as $(tn+tp)/(tn+tp+fn+fp)$. Overall, the methods with the maximum accuracy are applied to the testing data, and each of the results below are reported for the best method with optimal hyperparameters.

4.3 Hyperparameter Effects on Accuracy

4.3.1 Parameters

The learning rate, variance of the weight initialization Gaussian, and multiple output layer functions were evaluated for their effects on accuracy after 500 epochs, in order to find the best parameters for further structure investigation. For the learning rate, a weight variance of 0.1 was used, with a sigmoid network and 50 nodes in a single hidden layer. For the weight variance, a learning rate of 0.01 was used, with a sigmoid network and 50 nodes in a single hidden layer. For the output layer trials, a learning rate of 0.05 was used with a weight variance of 0.1 and a structure of 40 then 30 hidden nodes in two layers. The results can be found in tables 1, 2, and 3. From these results, it can be observed that the output layer does not have an effect on the accuracy, a learning rate in the range of 0.01 is appropriate, and a weight variance of 0.2 works best.

Learning Rate	1.0	0.5	0.4	0.3	0.2	0.1	0.03	0.01	0.003	0.001
Accuracy	87.4	90.3	90.8	90.7	91.3	91.3	90.4	91.1	90.4	89.9

Table 1: Accuracy data for various learning rates.

Weight Variance	0.5	0.4	0.3	0.2	0.1	0.03	0.01
Accuracy	89.7	90.0	90.2	91.2	91.1	89.6	84.2

Table 2: Accuracy data for various weight variance initializations.

Output Layer	Linear	Logistic Sigmoid	Softmax
Accuracy	91.7	91.8	91.6

Table 3: Accuracy data for various output layer activation functions.

4.3.2 Structure

Using these basic parameters, various structure patters were investigated. Table 4 shows the accuracy while varying the number of nodes in a single layers. From this it can be observed that a higher number of nodes leads to a higher accuracy. Table 5 shows the accuracy while keeping the total number of nodes consistent but varying the number of layers. From this it can be observed that the number of nodes in a single layer is more important than the total number of nodes, and a higher number of nodes per layer leads to more accuracy. Table 6 shows various structures across number of nodes and number of layers. Form this is can be observed that an even number of layers with a consistent number of nodes, and an increasing number of nodes leads to higher accuracy.

Nodes	5	10	15	20	25	30	35	40	45	50	55	60	65	70
Accuracy	71.4	79.3	80.1	85.5	86.2	88.4	89.1	89.8	90.7	90.9	91.5	91.8	91.8	92.1

Table 4: Accuracy data for various nodes per single hidden layer.

Nodes	60	30	20	15	12	10
Accuracy	92.3	91.7	90.1	88.3	86.3	61.3

Table 5: Accuracy data for the same number of nodes total but different numbers of layers. The number of nodes in each layer is reported.

Nodes (Layers)	5 (1)	5 (2)	5 (3)	5 (4)	20 (1)	20 (2)	20 (3)	20 (4)
Accuracy	66.7	77.1	76.4	78.6	86.4	90.2	89.9	90.7
Nodes (Layers)	57 (1)	57 (2)	57 (3)	57 (4)	80 (1)	80 (2)	80 (3)	80 (4)
Accuracy	92.4	93.3	92.2	93.2	92.7	93.3	92.9	92.7

Table 6: Accuracy data for the number of nodes and (layers).

4.4 Hyperparameter Effects on Convergence

While the learning rate and weight initialization hyperparameter choices had little effect on the overall accuracy after 500 epochs, it can be observed from tables 7 and 8 that a larger learning rate contributes to a faster convergence in the short term but too large causes instability, and a weight initialization around a variance of 0.5 works the fastest. The learning rate trials ran until an error of 0.35, using a weight variance of 0.2 and a single hidden layer of 50 nodes. The weight variance trials ran until an error of 0.35, using a learning rate of 0.01 and a single hidden layer of 50 nodes. Trials were capped at 150 epochs.

Learning Rate	10.0	1.0	0.1	0.01	0.001	0.0001	0.00001
Epochs	150	150	3.0	4.4	27.4	132.8	150

Table 7: Number of epochs to converge to an error of 0.35 for various learning rates. Capped at 150 epochs.

Weight Variance	5.0	1.0	0.5	0.1	0.01	0.001
Epochs	50.4	4.4	3.2	10.0	150	150

Table 8: Number of epochs to converge to an error of 0.35 for various weight variances. Capped at 150 epochs.

4.5 Hyperparamter Effects on Runtime

Learning rate, weight variance, and output layer activation function have a negligible effect on the runtime of the program. Table 9 shows various structures across number of nodes and number of layers. From this data it can be observed there is a small increase in runtime based on the number of nodes, and a large increase based on the number of layers. This is consistent with the vectorized implementation used, and iterative backpropagation.

Nodes (Layers)	5 (1)	5 (2)	5 (3)	5 (4)	20 (1)	20 (2)	20 (3)	20 (4)
Runtime	102	143	182	220	106	158	200	244
Nodes (Layers)	57 (1)	57 (2)	57 (3)	57 (4)	80 (1)	80 (2)	80 (3)	80 (4)
Runtime	115	187	260	317	129	235	310	401

Table 9: Runtime data in seconds for the number of nodes and (layers).

4.6 Dimensionality Reduction

In order to test a reduction in dimensionality, a constant hidden layer of 50 nodes with a learning rate of 0.01 and a weight variance of 0.2 was used over 500 epochs. The accuracy and runtime results are in table 10. From this data it can be observed that the reduction in data contributes minimally to runtime reduction, and reduces the accuracy slightly. Less principal components leads to less accuracy.

PCs	2	4	8	16	32
Accuracy	87.2	87.3	89.7	90.1	91.8
Runtime	104	108	99	99	116

Table 10: Accuracy and runtime in seconds data for various numbers of principal components after PCA reduction.

4.7 Best Results

The lessons learned from the hyperparameter investigation were combined to create a two layer, 200 node per layer network with a weight initialization of 0.2, a learning rate of 0.01, and a logistic sigmoid output layer. The training was stopped at epoch 120 in order to prevent overfitting to the training data as seen figure 6, where the training accuracy is shown in orange and testing accuracy is shown in blue.

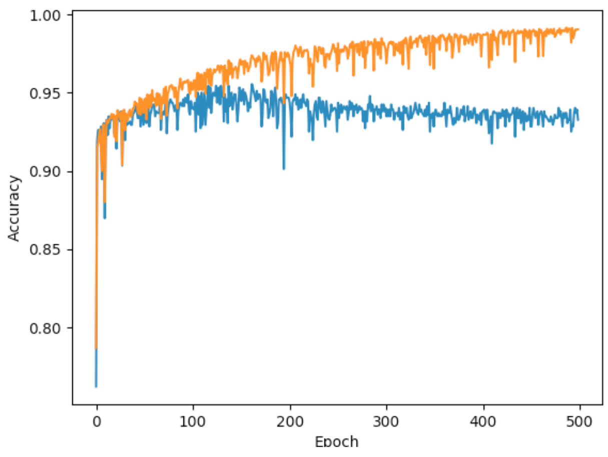

Figure 6: Convergence using the best structure. From epoch 120 on overtraining begins to occur.

The training accuracy achieved was 97.4%, the testing accuracy was 95.3%, and the final cross entropy error on the training data was 0.08. Table 11 is a confusion matrix of the best results, and figure shows the training and testing accuracy while converging.

	Predicted Class	
True Class	Not Spam	Spam
Not Spam	509	16
Spam	27	369

Table 11: Confusion matrix for the best neural network.

5 Summary

In this project, artificial neural networks were applied to email spam detection, and hyperparameters were optimized for best performance. Overall, an accuracy of 95.3% was achieved.

The objectives of this project were to implement artificial neural networks from scratch, and to optimize the hyperparameters for a network to discriminate between spam and non spam emails. From this, I learned interesting natural language processing features, became confident in my theoretical understanding of backpropagation, and practiced multiple error stopping methods for training. In future work, better training methods like momentum and regularization could be implemented.

References

[1] Ethem Alpaydin. *Introduction to Machine Learning Third Edition.* The MIT Press, 2014.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification Second Edition.* Wiley, 2001.

[3] Bruce MacLennan. Under 5 mortality rates 1800 to 2015, 2015. http://web.eecs.utk.edu/~mclennan/Classes/425-528/.

6 Appendix

The code for this project can be found on GitHub at https://github.com/LambentLight/https://github.com/LambentLight/bp_spam. Please contact me with any questions or considerations.