Project 4: Back-Propagation

CS 425

Samuel Steinberg

November 11th, 2019

Description:

This paper analyzes the application of a multilayer artificial neural network (ANN) to classify emails as spam or not spam. ANN's are computing systems that learn to by considering examples without task-specific rules. They are made up of neuron nodes that communicate with one another, similar to biological brains. In this project, the ANN is implemented with a back-propagation algorithm, which computes the gradient of the loss function with respect to input/output weights. This allows for training a multiplayer network and is ideal for feedforward neural networks and updating weights with minimal loss. Online learning was performed for this project. The database being tested on was called *spambase.data* and consisted of 4601 entries with 58 attributes each. Also given for the project was *spambase.names*, which includes the source of the dataset along with the characteristics of the attributes. Users could specify the form of the output layer (softmax, logistic sigmoid, or linear), along with the learning rate, hidden layers, number of hidden neurons, and various other conditions discussed in this paper.

Data Pre-Processing:

Data was given in the following form:

- 48 continuous real attributes of types of word frequencies in the form of a percentage. A "word" is defined in *spambase.names* as "any string or alphanumeric characters bounded by non-alphanumeric characters or end-of-string". These values are all given in a range of 0 to 100.
- 6 continuous real attributes representing character frequency as a percentage. These values are all given in a range of 0 to 100.
- 1 continuous real attribute characterizing the average length of uninterrupted capital letter sequences.
- 1 continuous integer symbolizing the longest length of uninterrupted capital letter sequences.
- 1 continuous integer which is the total number of capital letters in the email.
- 1 nominal class attribute denoting whether an email is spam (1) or not spam (0).

The data was delivered clean, so there were no alterations necessary such as mean imputation or needing to drop any entries. The data was separated into test, training, and validation sets and *k*-fold cross validation was performed. This separates the dataset into *k* equal-sized subsamples, for which one was given to validation, two for training, and the remaining for testing. In this project, the user was able to specify the number of *k*-folds. These folds separated the 4601 entries from *spambase.data* to create the sets.

Solution Description:

After data pre-processing was completed, it was possible to begin computation. See Figure 1 showing the online weight update formula below:

$$\Delta w_j^t = \eta(r^t - y^t)x_j^t$$

Figure 1: Online Weight Update Formula

Here, Δw_j^t represents the new updated weight for each instance pair with an index *t* where *j* ranges from 0, ..., *d* dimensions. η represents the learning factor while (r^t, x^t) is the instance pair in the network and y^t is the output. Weights will represent the strength of connection between nodes and lessens the importance of the input value. When the learning factor is gradually decreased in time for convergence this is stochastic gradient descent. One place the weights are used is in the error function. See Figure 2 below:

$$E^t(\mathbf{w}|\mathbf{x}^t, r^t) = \frac{1}{2}(r^t - y^t)^2 = \frac{1}{2}[r^t - (\mathbf{w}^T\mathbf{x}^t)]^2$$

Figure 2: Error Calculation

Here, *w* represents the weight at an index *t*, while (r^t, x^t) are again the instance pair in the network and y^t is the output. The error begins with random initial weights, and at each iteration the parameters are adjusted to minimize error while remembering previously learned data.

$$Activation = \Sigma_i\ (w_i * input_i)$$

Figure 3: Activation Function

The activation function shown above in Figure 3 defines the output of a node given an input set or set of inputs. It is calculated by summing the product of the weights (w_i) and inputs ($input_i$). The activation is important to the transfer function, which uses it to calculate the output (See Figure 5 below).

$$Transfer = \frac{1}{1 + e^{-activate}}$$

Figure 4: Sigmoid Transfer Function

The transfer function models the output for each input. For this project, the sigmoid function was used. It uses the activation of a neuron and hence also depends on the calculation of the weights. In Figure 5 above, *activate* represents the output of the activation function. This is a necessary piece to feedforward networks.

$$A = cx$$

Figure 5: Linear Function

The linear transfer function multiplies an activation *x* by a scalar *c* to give an output value *A*. It will compile into a linear predictor combining a set of weights with the feature vector.

$$y_i = \hat{P}(C_i|\boldsymbol{x}) = \frac{\exp[\boldsymbol{w}_i^T\boldsymbol{x} + w_{i0}]}{\sum_{j=1}^{K}\exp[\boldsymbol{w}_j^T\boldsymbol{x} + w_{j0}]},\ i = 1, \ldots, K$$

Figure 6: Softmax Function

The softmax function will treat all classes uniformly and is displayed in Figure 6. If the weighted sum of one class is significantly larger after exponentiation and normalization, y_i (output) will be close to one and the others close to zero. This works like a differentiable maximum. In this equation *w* represents the weights at index *j* and *x* a pair. The variable *i* will iterate up to *K* classes.

In addition to the transfer functions, a transfer derivative further aids in minimizing error. This is done by calculating its slope. Since this differs with each function, the example included below in Figure 7 is for the sigmoid transfer derivative, as each will differ.

$$Transfer\ Derivative = output * (1 - output)$$

Figure 7: Sigmoid Transfer Derivative Function

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}}$$

Figure 8: Back-Propagation Chain Rule Gradient

In the back-propagation algorithm, the chain rule can be used to calculate the necessary gradients used for multiple iterations of exposing the training set to the network. Shown in Figure 8 above, derivatives are being taken for the error (*E*) with respect to the first layer weights w_{hj}. Also, in the equation outputs (y_i) and inputs (z_h) are taken into account.

The performance of the algorithm depended on the predicted class vs. true class, as shown in the confusion matrix in Figure 9 below:

True Class	**Predicted Class**	
	spam	**not spam**
spam	TN	FP
not spam	FN	TP

Figure 9: Confusion matrix

In this matrix, TN represents a true negative, FP a false positive, FN a false negative, and TP a true positive. See Figures 10-14 for the performance metrics extracted from the matrix.

$$\text{Accuracy} = (\text{TN} + \text{TP}) \,/\, (\text{TN} + \text{TP} + \text{FN} + \text{FP})$$

Figure 10: Accuracy Formula

The Accuracy metric represents the overall accuracy of the algorithm in the form of a percentage of how many correct classifications there were out of the total. Hence, the calculation is the number of correct (true) classifications over the total number of classifications.

$$\text{TPR (true positive rate, recall, or sensitivity)} = \text{TP} \,/\, (\text{TP} + \text{FN})$$

Figure 11: TPR Formula

As shown in Figure 11 above, TPR is the percentage of the positive classifications determined by the algorithm over the total number of real positives. Since a FN (false negative) actually represents a truly positive case of cancer, the addition of it and the number of TP (true positives) make up the denominator for the total number of positive cases.

$$\text{PPV (positive predictive value or precision)} = \text{TP} \,/\, (\text{TP} + \text{FP})$$

Figure 12: PPV Formula

The PPV is the probability that emails with a positive classification actually have a negative value. It takes the percentage of the number of correct positive classifications over the total number of actual positive classifications. See Figure 12 above.

$$\text{TNR (true negative rate or specificity)} = \text{TN} \,/\, (\text{TN} + \text{FP})$$

Figure 13: TNR Formula

The TNR metric is a very similar measure of sensitivity to TPR in Figure 11, but for negative classifications. It takes the number of correctly classified not spam over the total number of actual spam.

$$F_1 \text{ Score} = 2 \times \text{PPV} \times \text{TPR} \,/\, (\text{PPV} + \text{TPR})$$

Figure 14: F_1 Score Formula

The F_1 Score is a measure of a test’s accuracy using the harmonic mean of precision and recall. This helps to balance precision and recall, which at times can offer a better idea of model performance than accuracy. The balance can be seen in Figure 14 above, with PPV (precision) and TPR (recall) being balances against each other.

Analysis and Discussion:

In this project, the best determination of the success of the algorithm was with experimentation with different learning rates and viewing their performance metrics when limited by a maximum number of epochs. There were also experiments with different *k* *k*-folds and the number of hidden neurons in the network. These turned out to be ideal at five folds and five hidden neurons, so these were used to calculate final performance values. Values below five hidden neurons tended to yield slightly lower accuracies, while any value higher resulted in similar accuracy but longer runtime. Error values for the algorithm were printed in the program. Overall, the algorithm performed quite similar with different numbers of epochs when compared to learning rate. With more epochs, the algorithm has more time to converge and increase accuracy, though after a learning rate of around 0.15 this begins to steady out. See Figures 15 and 16 below:

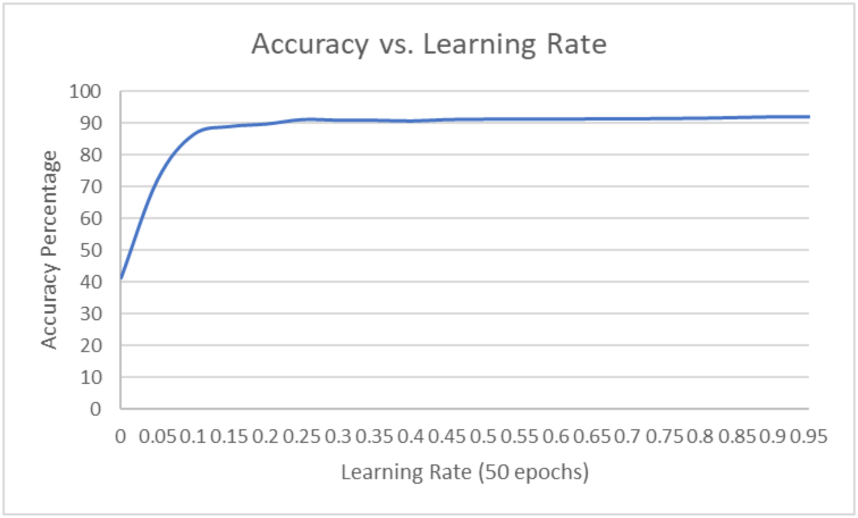

Figure 15: Comparison with 50 epochs

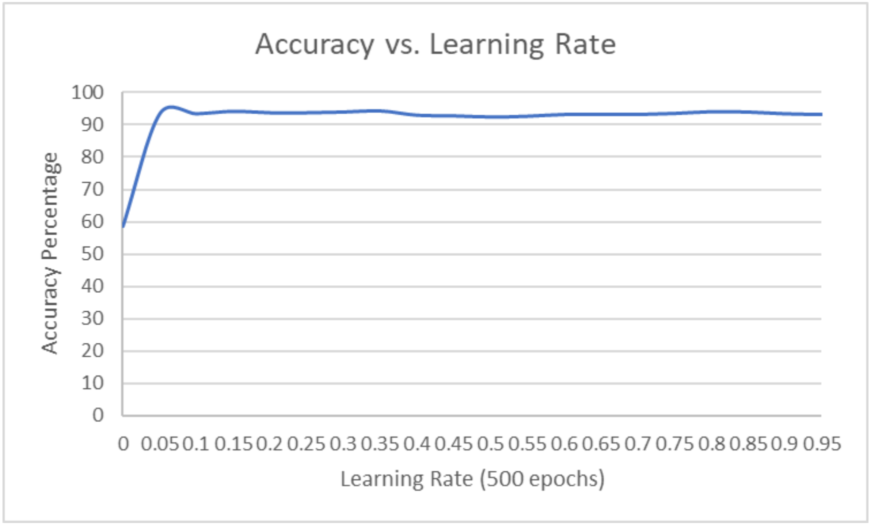

Figure 16: Comparison with 500 epochs

With a higher number of epochs, the algorithm was able to substantially better accuracy at a learning rate of 0. However, once the learning rate increased both algorithms performed significantly better and accuracy stayed steady. The experiments with higher epochs resulted in averages of around 93% accuracy, while a lower epoch resulted in around 90%. Additionally, the higher epoch network was able to learn significantly quicker. With more time to learn, the algorithm jumped from 58% accuracy with no learning rate to 93% with only a .05 learning rate. Conversely, the lower epoch network had a steadier climb from 41% to 72% to 86% and so on until averaging out around 90%. A similar performance increase is also seen when analyzing the F1 scores of each, where the effect on learning rate is also seen. See Figures 17 and 18 below:

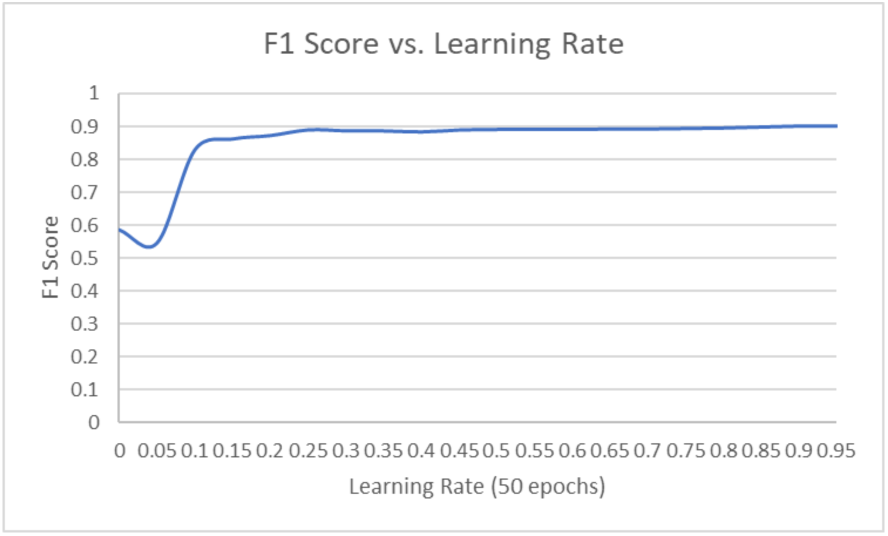

Figure 17: F1 with 50 epochs

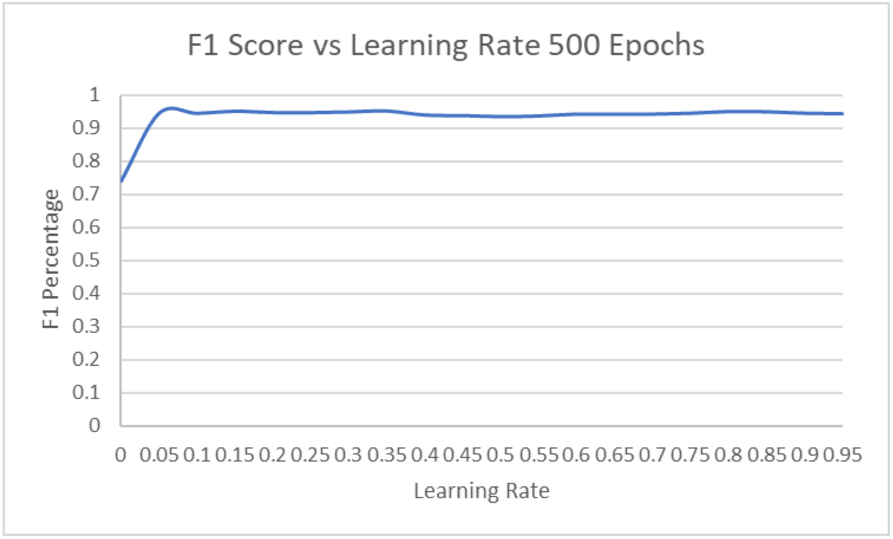

Figure 18: F1 with 500 epochs

Since the F1 score is an important metric to balance precision and recall, it is important to consider. Figures 17 and 18 revealed the same trends as the accuracy charts. The larger number of epochs began higher with no learning rate, and when it plateaued it was at a higher percentage than the lower number of epochs (around 95% vs 90%). The Tables 1 and 2 below represent the complete metrics from the algorithm. Their values were extracted from a confusion matrix following the format from Figure 9.

Table 1: 50 Epoch Performance Metrics

Learning Rate	**Accuracy (%)**	**F1 Score (%)**	**TPR (%)**	**TNR (%)**	**PPV (%)**
0	41.30435	58.46154	41.30435	0	100
0.05	72.06522	54.51327	83.24324	69.2517	40.52632
0.1	86.41304	82.75862	86.95652	86.08696	78.94737
0.15	88.91304	86.06557	89.48864	88.55634	82.89474
0.2	89.67391	87.10991	89.91597	89.52043	84.47368
0.25	91.08696	88.82834	92.0904	90.45936	85.78947
0.3	90.86957	88.55586	91.80791	90.28269	85.52632
0.35	90.86957	88.52459	92.04545	90.14085	85.26316
0.4	90.65217	88.21918	92	89.82456	84.73684
0.45	91.08696	88.79781	92.32955	90.3169	85.52632
0.5	91.19565	88.94952	92.35127	90.47619	85.78947
0.55	91.19565	88.94952	92.35127	90.47619	85.78947
0.6	91.19565	88.94952	92.35127	90.47619	85.78947
0.65	91.30435	89.10082	92.37288	90.63604	86.05263
0.7	91.30435	89.10082	92.37288	90.63604	86.05263
0.75	91.41304	89.2517	92.39437	90.79646	86.31579
0.8	91.52174	89.40217	92.41573	90.95745	86.57895
0.85	91.73913	89.7019	92.4581	91.28114	87.10526
0.9	91.95652	89.9729	92.73743	91.45907	87.36842
0.95	91.95652	89.9729	92.73743	91.45907	87.36842

With a lower stopping epoch, metrics tended to still be on an upward trend. All of these metrics were recorded with five hidden neurons, and the algorithm improved as the neural network learned more and more from previous iterations. This formed the beginnings of the hypothesis that epochs and learning rate are the most important factors in network performance. This was confirmed with a much higher volume of epochs, allowing the network more time to learn. See Table 2 on the following page for metrics for 500 epochs.

Table 2: 500 Epoch Performance Metrics

Learning Rate	Accuracy (%)	F1 Score (%)	TPR (%)	TNR (%)	PPV (%)
0	58.69565	73.9726	58.69565	0	100
0.05	93.58696	94.55217	94.29098	92.57294	94.81481
0.1	93.58696	94.53197	94.61967	92.12598	94.44444
0.15	94.34783	95.15829	95.69288	92.48705	94.62963
0.2	93.91304	94.75655	95.83333	91.32653	93.7037
0.25	93.91304	94.73684	96.18321	90.90909	93.33333
0.3	94.13043	94.94382	96.02273	91.58163	93.88889
0.35	94.45652	95.2381	96.0452	92.28792	94.44444
0.4	93.15217	94.01709	96.49123	88.94349	91.66667
0.45	92.93478	93.83886	96.1165	88.88889	91.66667
0.5	92.6087	93.53612	96.09375	88.23529	91.11111
0.55	92.82609	93.73814	96.10895	88.66995	91.48148
0.6	93.36957	94.29373	95.2741	90.79284	93.33333
0.65	93.36957	94.29373	95.2741	90.79284	93.33333
0.7	93.36957	94.29373	95.2741	90.79284	93.33333
0.75	93.69565	94.5591	95.81749	90.86294	93.33333
0.8	94.23913	95.06977	95.51402	92.46753	94.62963
0.85	94.13043	95.00924	94.83395	93.12169	95.18519
0.9	93.58696	94.58219	93.80692	93.26146	95.37037
0.95	93.36957	94.39853	93.62477	92.99191	95.18519

As was previously stated, more epochs allowed the network more time to learn. Unlearned accuracy began significantly higher than low epoch accuracy and rose to its peak much faster. The other metrics generally followed suit. Notably, the network at first only classified all emails as spam while at the next iteration at a learning rate it learned to separate.

In conclusion, the number of hidden neurons and number of *k k*-folds did not significantly alter the performance of the network. The most impactful determination of success was the iteration cap (number of epochs) and the learning rate. Higher learning rates allowed the network to learn and “remember” more, while a higher number of epochs allowed a longer time for learning. Unsurprisingly, with no learning rate all classifications were classified as all spam or all not spam. Once the learning rate increased, this rapidly changed and there were better results. Additionally, once the epoch cap was raised performance metrics improved further because of the said time increase. The main two metrics held in the highest regard for this project were accuracy and F1 score, since accuracy represents the overall accuracy of the algorithm in the form of a percentage of how many correct classifications there were out of the total, and the F1 score tests accuracy using the harmonic mean of precision and recall. These would give the best indications of whether classifications are successful and how well the network and algorithm are working.