An analysis of logistic regression and single-layer neural networks as applied to optical character recognition

Federico Pizarro Bejarano

IB candidate number: 001096-0155

St. Francis Xavier Secondary School

Abstract word count: 294

Essay word count: 3891

**Abstract**

This investigation determines the extent that characters can be identified in images using the logistic regression and single-layer neural network algorithms. Optical character recognition (OCR) is a computer vision, supervised learning problem. The dependent variables were the optimal value of the regularization parameter lambda, the accuracy on the training, cross validation, and test sets, and the time needed to train each classifier.

A dataset of 74,000 images composed of fonts, handwritten characters, and real images of letters and numbers was used. For the purposes of this investigation only a subset of the font dataset was used. Each image was resized to be 20x20 pixels and then converted to a 1x400 vector of pixel values.

The logistic regression algorithm attempts to fit parameters to the 400 pixel values to form a hypothesis function. To optimize the parameters, the algorithm defines a cost function and then performs gradient descent on the parameters. The tunable parameters were additional features added in an attempt to create more complex, representative functions.

A single-layer neural network passes the input data to a hidden layer where the data is partially processed. The partially processed data is then passed to the output layer where the final predictions are made. The tunable parameter was the number of hidden units in the hidden layer.

The logistic regression algorithm achieved an accuracy of 85.14% with no added features and a lambda value of 1. The neural network achieved a significantly higher accuracy of 90.19% using 200 hidden units and no regularization. Logistic regression had a time complexity of O(n) while the neural network had a significantly better time complexity of O(√h). This paper investigates the properties of both algorithms as well as establishes the inability of both algorithms to identify characters to sufficiently high accuracies.

**Table of Contents**

**1. Introduction 4**

**2. OCR Dataset 5**

2.1 Acquiring Dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.2 Data Processing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**3. OCR Using Logistic Regression 6**

3.1 Implementing Logistic Regression . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

` 3.2 Analyzing Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

3.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**4. OCR Using Neural Networks 12**

4.1 Implementing the Neural Network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

4.2 Analyzing Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

4.3 Visualizing the Hidden Layer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

4.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

**5. Comparison and Evaluation 21**

5.1 Accuracy and Optimization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

5.2 Time Complexity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

**6. Conclusion 23**

**7. References 24**

**8. Appendix A: Raw Data 25**

**9. Appendix B: Program Code 27**

9.1 Logistic Regression Cost Function and Calculation of Gradients . . . . . . . . . . 27

` 9.2 Training Logistic Regression using the One vs. All Method . . . . . . . . . . . . . . 27

9.3 Single-Layer Neural Network Forward and Back Propagation . . . . . . . . . . . . 28

**1 Introduction**

With the growth of digital photography and the importance of computerization, there has been an interest to use computers to understand and digitalize the information in images. Optical character recognition (OCR) is the branch of computer vision that focuses on recognizing characters. Considering the increase in amounts of data and the importance of digitalization, optimization of OCR algorithms is increasingly important.

The variety of sizes, fonts, backgrounds, and colors demand that the OCR algorithm be robust and flexible. The ability to recognize complex patterns such as characters while minimizing “noise” such as backgrounds is necessary for real-world applications [1].   
 Accuracy varies widely, especially depending on how “clean” the data is. One study found that commercial OCR software varies between 81% and 99% accuracy [3], which leaves many errors per page. This large error rate makes the technology only useful in specific applications where errors are either acceptable or easily corrected. Thus, improving OCR algorithms remains an important challenge for many applications.

This paper evaluates two algorithms applied to optical character recognition: a logistic regression model using batch gradient descent and a single-layer neural network that uses logistic units and the backpropagation algorithm to train the units.

This task is considered a supervised learning, computer vision problem as it takes labeled images and attempts to categorize them using machine learning. The data set was divided into a training set, a cross validation set, and a test set. The parameters for both logistic regression and the neural network are generated on the training set, the optimal value for lambda, the regularization parameter, will be determined based on the accuracy on the cross validation set, and the final accuracy is measured on the test set using the optimal value of lambda found on the cross validation set. This will ensure that the algorithm and parameters can generalize to other sets of data and not just the training set. The cross validation set eliminates artificial inflation of the accuracy due to assigning the highest accuracy as the final accuracy of the classifier, when the accuracy may be high due to random differences of the set.

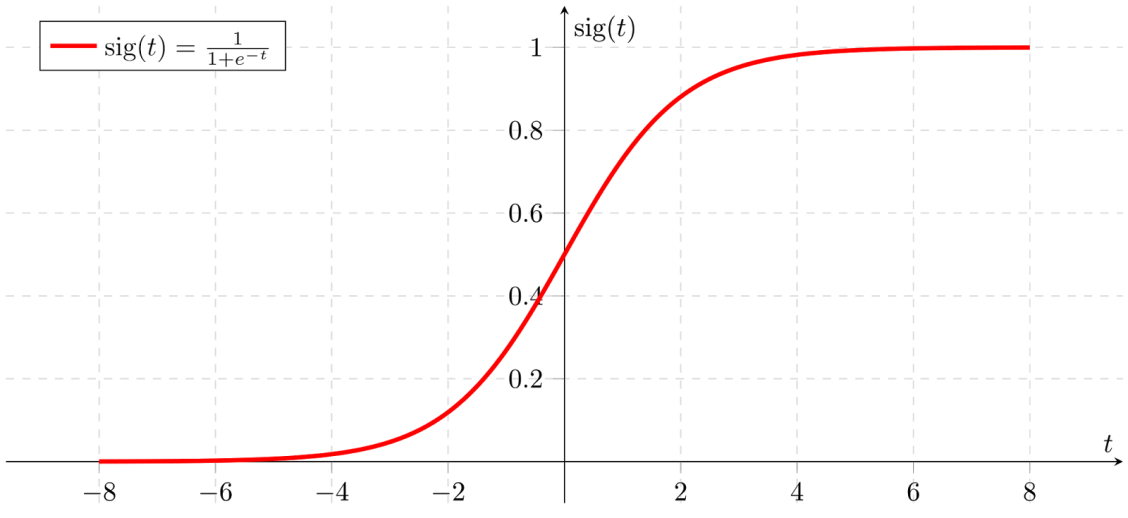
**2     OCR Dataset**

**2.1     Acquiring Dataset**

For the purposes of this investigation, a dataset that was both complex enough to allow for optimization as well as clean enough to use without extensive preprocessing was necessary. Centered images were necessary as translations of the input are not tolerated by either logistic regression or simple neural networks; to tolerate translation of inputs one would need to use more complex systems such as convolutional neural networks [5]. The Chars74k dataset[[1]](#footnote-1) was chosen as it has 62 different types of characters (0-9, A-Z, a-z) and three kinds of images (fonts, handwritten characters, and real images), all of which are centered. However, the dataset had very small numbers of images for both the real images and the handwritten datasets; thus, only the font dataset was used. Also, due to the memory limitations of Octave (the program used for this investigation), a smaller dataset than the total amount of images in the font dataset was necessary (the dataset has 62669 images, 1016 per class). Thus, to have enough images per character to train an algorithm yet also remain within the memory capacities, characters a-z were discarded.

**2.2     Data Processing**

The size of the images had to be reduced to be able to remain within the memory limitations. The font images were all 128x128 pixels and grayscale, and were reduced in size until the matrix of training examples fit within the memory capabilities but retained sufficient detail to be identified by the algorithms; the optimal size found was found to be 20x20 pixels. Also, the full number of images per character in the font dataset (1016) created matrices too large for Octave. Thus, 600 random images per character were chosen, processed, and then 400 were assigned to the training set, 100 to the cross-validation set, and the final 100 to the test set.

****

*Figure 1[[2]](#footnote-2): A sigmoidal or logistic curve*

**3        OCR Using Logistic Regression**

Recognition of symbols such as characters relies on the relationship between different portions of the image which come together to be form a cohesive whole. The simplest way to divide a digital image is by each pixel, using every pixel as a feature. The relationship between each pixel to form the image is not simple however, as every character can be represented in many different ways; for example, human handwriting varies largely from person to person. However, logistic models can still be built even though the relationship isn't simple as they can model more complex relationships.

Data can be either continuous or discrete. Annual income is an example of continuous data while citizenship to a certain country is a discrete quantity, either yes or no (usually represented by 1 and 0). Logistic regression takes as inputs either discrete or continuous features and attempts to map them to discrete outputs. The hypothesis for a logistic regression algorithm can be modeled as:

 if , predict 1

 else predict 0

Where: is the hypothesis function

sig() is the sigmoid function (see Figure 1). It ensures the values are between 1 and 0 (this allows the hypothesis to be easily mapped to a discrete value [7]).

n is the number of features. An additional feature, x0, is always added that always has a value of 1. This is known as the bias feature or bias unit and has a corresponding parameter (θ0).

**x** is a row-vector of a single training or test example that has dimensions 1x(n+1)

**θ** is the column-vector of parameters that has dimensions (n+1) x 1

A cost function must be defined to find the error in every parameter, and although many may be used the one that will be used for the purposes of this paper is:

**J(θ)**  = \*

This equation has the property that it is convex, with one global minimum and no local minima [4]. The convex property is vital as to optimize each parameter the partial derivative of the cost function is found. Local optima could cause the algorithm to never reach the global optimum.

When all the gradients with respect to every parameter are calculated over the entire training set (hence the term *batch* gradient descent), every parameter is subtracted by the gradient times the change parameter alpha. This lowers the cost one step towards the global optima, and repeated iterations of these small steps will lead closer and closer towards optimized parameters. A regularization term is also added to the cost function and the gradient function that taxes the cost function for large values for each parameter (except the bias parameter), forcing it to choose the smaller parameters while still optimizing the performance on the training set [5]. Smaller features limit overfitting of data (overly complex decision boundaries, also known as variance), while overly small features may cause underfitting (excessively simplified decision boundaries, also known as bias) [9]. This balance between complexity and accuracy is controlled by the regularization parameter lambda.

**3.1 Implementing Logistic Regression**

Logistic regression can only predict binary values, either 1 or 0, so a “One vs. all” method is used to decide between the 36 classes. This method creates a different logistic regression classifier for every class (every character). Each classifier takes one character as the positive case, which it will predict as 1, and classify all others as negative cases, 0. All classifiers are trained against the entire dataset and then every classifier is applied to the cross validation dataset to get 36 different predictions for every example[[3]](#footnote-3). The classifier with the highest probability, the highest value of the hypothesis function for that example, is the final prediction. This system is inefficient in that 36 classifiers need to be independently trained, thus 100 iterations over the training set in logistic regression means 100 iterations per classifier.

The program has several tunable parameters. The first is iterations per classifier, which will remain constant through all the trials (100 iterations). The independent variable that will be investigated in this paper with regards to logistic regression will be applying feature engineering to the training set. This will be done by adding 400 more features which are the original 400 pixel values raised to successive powers up to the power of 4, thus possibly allowing the logistic regression classifier to create more complex polynomial decision boundaries. The sets will thus have 401, 801, 1201, and 1601 features respectively (counting the bias unit). Every set will be run seven times with lambda values of 0, 0.3, 1, 3, 10, 30, and 100. The optimal value of lambda for each change of parameters will be decided based on the highest accuracy on the cross-validation set, and the accuracy of the classifier of that set will be determined as the accuracy of the classifier with the optimal value of lambda on the test set. The optimal value of lambda, the accuracy on the training set, the cross-validation set, and the test set, as well as the time taken to train all the classifiers will be the dependent variables.

**3.2 Analyzing Results**

Overfitting causes the parameters to fit too well to the training set, making them generalize badly. As the value of lambda increases, it taxes how large the parameters are more. This creates simpler decision boundaries which thus combats overfitting. However, larger values of lambda than are optimal cause underfitting. Thus, as lambda increases, the training set accuracy should fall, and the cross validation set accuracy should rise then fall after the optimal value is reached. The optimal value of lambda, the point of the zenith on the cross validation curve, should increase as more parameters are passed to the classifier. Also, the time needed to run each set should grow as the number of features passed to the algorithm grows.

*Graph 1: Training set and cross validation set accuracies for logarithms of lambda. For visualization purposes, the accuracies were plotted against the logarithms of lambda and a lambda of 0 was plotted as -2, as there is no logarithm of 0. Training set accuracies are plotted as dotted lines while cross validation set accuracies are solid lines*

The training set accuracies can be observed to consistently fall as lambda increases, and the cross validation set accuracies rise and then fall as expected, demonstrating that the algorithm was overfitting and then began underfitting as lambda grew too large. The drop in accuracy is less pronounced as the powers increase, as they are more prone to overfitting and less to underfitting. The optimal values of lambda for each power, and thus the point of the maximum on each curve, increase from a value of lambda of 1 to 10 from a power of 1 to a power of 2. However, the optimal value remains unchanged at 10 the rest of the powers, in contradiction to the expected result.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Power** | **Opt. Lambda** | **% Training** | **% CV** | **% Test** | **Avg. Time (s)** |
| 1 | 1 | 90.15 | 85.39 | 85.14 | 848 |
| 2 | 10 | 88.85 | 85.44 | 84.56 | 1614 |
| 3 | 10 | 89.51 | 85.39 | 84.72 | 2361 |
| 4 | 10 | 89.99 | 85.39 | 84.92 | 3308 |

*Table 1: Optimal values of lambda with corresponding training set, cross validation set, and test set accuracies (%) for each power. Average time for each power is also included.*

The optimal value of lambda did not change from a power of two to a power of 10, and the optimal value of the cross validation set was 85.39% for every power except power 2, which was slightly higher. The actual accuracies of each power, the test set accuracies, changed slightly. The power of one, which corresponded to training over solely the raw data with no added features, performed the best, again in contradiction to the hypothesis.

*Graph 2: Average time (s) used to train per power.*

The average time per power grew linearly, as expected, increasing approximately 800s per 400 features added.

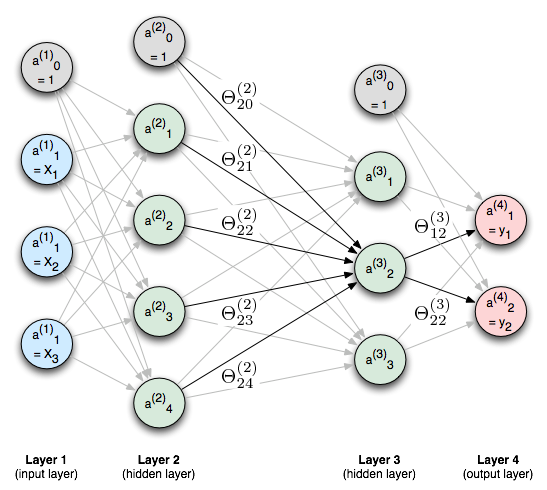
**3.3 Summary**

The results clearly showed the effects of regularization on the algorithm, as it caused accuracy to increase compared to using no regularization (lambda value of 0), but needed to be tuned to prevent underfitting. However, using feature engineering, adding more features corresponding to the previous features raised to successive powers, had little effect on the final accuracy and in fact decreased it. The first set with no feature engineering performed the best with a final accuracy of 85.14%. Clearly, feature engineering applied in this way is useless for logistic regression with this dataset. This is probably due to the fact that the added features do not add significantly new information as OCR does not depend on individual features but on relationships as a whole. Also, due to the one vs. all approach of multinomial logistic regression the algorithm was quite slow. Adding features caused the algorithm to grow roughly linearly, although this may be implementation specific. An algorithm that can formulate more complex relationships should perform better.

**4 OCR using Neural Networks**

Neural networks work on the same principle as logistic regression; they fit parameters to features and attempts to optimize these parameters to minimize cost. However, instead of fitting parameters to the features to immediately get a prediction, a neural network computes partial answers which are then passed to the output layer to formulate predictions. The logistic units (logistic because they each formulate hypotheses using the same sigmoid function as logistic regression) which compute the partial  answers are organized in ‘hidden layers’ which are sandwiched by the input layer and the final output layer. There can be any number of hidden layers, each with any number of units; more units and layers allows a greater ability to recognize complex relationships [6]. Rather than computing the derivative of the cost function to train the parameters, neural networks typically use the backpropagation algorithm which calculates the error in each unit starting from the output units and through the hidden layers [6].

Each unit’s output (represented by on Figure 2) can be calculated using the same equation of logistic regression. As the input is passed through the neural network, it calculates the value of each unit and finally creates an output; this is called forward propagation. To calculate the error in each unit and optimize each parameter a rather complicated algorithm called backpropagation is used. Much like logistic regression, backpropagation produces gradients to each parameter and then subtracts them, moving them one step towards the global optima. Regularization with lambda can also be applied to neural networks.



*Figure 2[[4]](#footnote-4): A neural network with 2 hidden layers. The input (blue) is connected to a hidden layer (green) which is itself connected to a second hidden layer (green). This is finally connected to an output layer (red). Each layer other than the output layer has a bias unit added to it which always has a value of 1 (purple). Each layer is connected to another by parameters Theta (a matrix represented by ).*

**4.1 Implementing the Neural Network**

For the purposes of this paper only a single hidden layer neural network will be analyzed. The independent variable will be the number of hidden units in the network, starting from 25 and doubling progressively to 400. The total number of iterations of forward propagation followed by backpropagation will be kept constant at 500 iterations. Every change of number of hidden units will be run seven times using lambda values of 0, 0.3, 1, 3, 10, 30, and 100 to determine an optimal value of lambda on the cross validation set. The optimal value of lambda will be used on the test set for each number of hidden units and the accuracy on the test set will be considered the accuracy for that set. Time elapsed to train the network, the accuracy on each set, and the optimal value of lambda will be recorded.

**4.2 Analyzing Results**

Just as in logistic regression, regularization using the lambda parameter limits the complexity of the decision boundary to prevent overfitting, but will cause underfitting when too high. I hypothesize that as the number of hidden units grows, the complexity of the decision boundary will grow, increasing the accuracy on the training set as well as the cross validation set. However, the propensity of the classifier to overfit will also grow [6] and thus the optimal value of lambda should also increase. The time complexity should increase as the number of hidden units increases due to increased computations.

*Graph 3: Training set and cross validation set accuracies for logarithms of lambda for 25, 50, 100, 200, and 400 hidden units (HU). For visualization purposes, the accuracies were plotted against the logarithms of lambda and a lambda of 0 was plotted as -2, as there is no logarithm of 0. Training set accuracies are plotted as dotted lines while cross validation set accuracies are solid lines.*

All the accuracies dropped greatly when the value of lambda approached 100, as expected, and most lines seem to drop as lambda increases in a curve. This implies that the optimal value of lambda for most numbers of hidden units is 0 to 1, and the accuracy drops after that. However, the curves corresponding to small numbers of hidden units, 25 and 50, are far more sporadic and experience a sudden spike in accuracy at a value of lambda of 10. The results do demonstrate that accuracy increases and number of hidden units increases, but they directly contradict the hypothesis in regards to the optimal value of lambda, as it seems to actually decrease as the number of hidden units increases.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hidden Units** | **Opt. Lambda** | **%Training Set** | **%CV set** | **%Test set** | **Avg. Time (s)** |
| 25 | 10 | 85.17 | 83.67 | 84.00 | 526 |
| 50 | 10 | 87.90 | 85.69 | 85.58 | 759 |
| 100 | 1 | 92.43 | 87.56 | 87.39 | 1066 |
| 200 | 0 | 96.92 | 89.89 | 90.19 | 1599 |
| 400 | 0 | 96.85 | 90.03 | 90.11 | 2404 |

*Table 2: Optimal values of lambda with corresponding training set, cross validation set, and test set accuracies (%) for each number of hidden units. Average time for each number of hidden units is also included.*

*Graph 4: Training set and cross validation set accuracies graphed against the number of hidden units.*

The accuracies of the training set and the cross validation set corresponding to the optimal values of lambda were graphed against the number of hidden units. The trendline that represented the data best was determined to be the logarithmic function. However, although it seems to represent the values corresponding to 25, 50, and 100 hidden units it does not represent the last two values well on either line. To better visualize the relationship, the accuracies were graphed against the logarithm of the number of hidden units.

*Graph 5: Training set and cross validation set accuracies graphed against the logarithm of the number of hidden units.*

The data corresponding to the accuracies of the training set and the cross validation set fall almost perfectly onto a linear trendline, with the exception of the last point. The point corresponding to 400 hidden units falls almost exactly horizontal to the point corresponding to 200 hidden units, as accuracy did not change much in either the training or the cross validation step. This suggests that in a single-layer neural network accuracy increases with the logarithm of the number of hidden units until it suddenly reaches a point where it no longer increases at all, the maximal accuracy for the network. This may be specific to OCR and to single-layer networks. Adding multiple hidden layers may increase the accuracy beyond this point.

*Graph 6: Average time (s) used for training per number of hidden units*

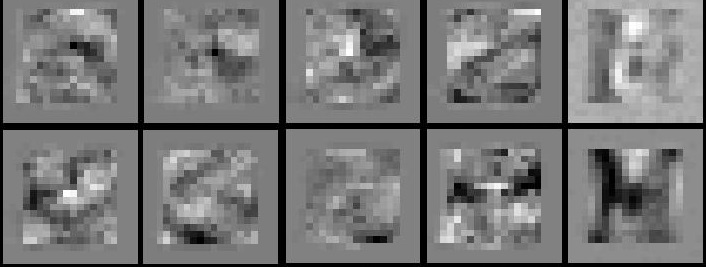
The average time needed to train the network per number of hidden units was graphed. The trendline that best represented the data was determined to be the power curve, which resulted in a curve very close to a square root. To determine if time was truly proportional to the square root of the number of hidden units, it was graphed against the square root of the number of hidden units.

*Graph 7: Average time (s) used for training graphed against the square root of the number of hidden units*

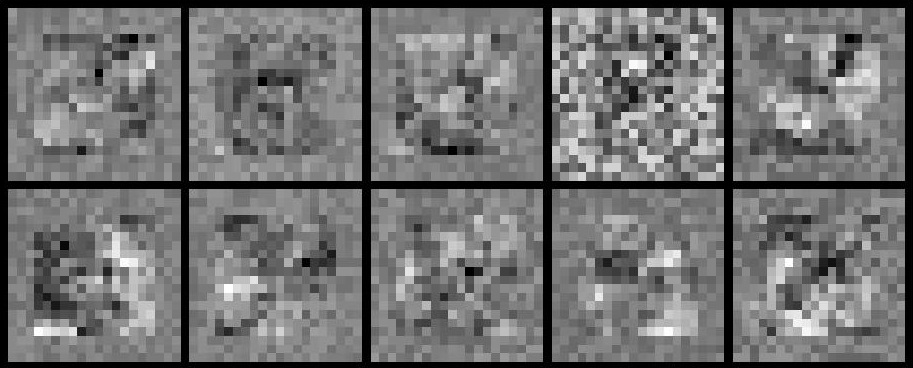
A linear trendline does in fact fit average time versus the square root of the number of hidden units very well. Therefore it can be concluded that, as opposed to logistic regression which increased linearly with added parameters, the amount of time needed to train a single-layer neural network increases by the square root of the number of hidden units. This may be implementation specific.

**4.3 Visualizing the Hidden Layer**

To better understand the functioning of the neural network as well as investigate the effect of the number of hidden units, it is useful to visualize the hidden layer. The hidden layer partially processes the data, creating partial answers which are then analyzed by the output layer. These partial answers may correspond to strokes, pixel density, even nearly complete characters [2]. To visualize the layer one informal way is to reshape the vector of 400 parameters (excluding the bias unit) corresponding to a single hidden unit into a 20x20 image [2]. This has been done with 10 random units from the neural network trained with 25 hidden units and 10 from the one trained with 400 hidden units.



*Figure 3: A visualization of 10 hidden units in the hidden layer of a 25 hidden unit network*



*Figure 4: A visualization of 10 hidden units in the hidden layer of a 400 hidden unit network.*

The outer region of every image in the 25 hidden unit visualization is an almost uniform shade of gray as the images from the dataset were centered, thus the outer region of every image held little valuable information. Also, an “M” (second row, fifth column) is seemingly visible, which is unfavorable. The network is producing complete answers before it should and not doing as much preprocessing on strokes and patterns as it should, reducing accuracy. However, the 400 hidden unit visualizations are far more vague and complex, with no discernable characters or shapes, with even the outer rim is being analyzed. This allows for more complex decision boundaries, and thus higher accuracy.

**4.4 Summary**

The neural network algorithm performed as expected in that the accuracy of its predictions and the time necessary to train the classifier increased as the number of hidden units increased. However, the optimal value of the regularization parameter decreased as the number of hidden units increased, which is in complete opposition to the hypothesis. More hidden units may, as the visualizations of the hidden layer suggest, allow for vaguer partial-answers which don't necessitate high values of parameters but are simply more complex. Thus, regularization would be simply detrimental to the success of the algorithm, at least for a single-layer neural network applied to OCR. Also, the accuracy of the algorithm increased directly with the logarithm of the number of hidden units, until it reached a maximum of 90.19% accuracy using 200 hidden layers and suddenly plateaued. Possibly single layer neural networks have such a property, or possibly it is only when applied to optical character recognition. More research is needed to determine the validity of this property. Finally, the time necessary to train the network increased with the square root of the number of hidden units.

**5      Comparison and Evaluations**

**5.1     Accuracy and Optimization**

The maximal accuracy achieved by the logistic regression algorithm was 85.14% accuracy on the test set with no feature engineering and a lambda value of 1. Feature engineering was incapable of increasing the accuracy of the results; however, it did not significantly detriment the accuracy either. The optimal value of lambda for logistic regression did increase when feature engineering was introduced, but immediately plateaued at a value of 10.

The highest accuracy achieved with a single-layer neural network was 90.19% using 200 hidden units and no regularization. Accuracy increased directly with the logarithm of the number of hidden units, until it suddenly stopped increasing at 400 hidden units. This suggests that single layer neural networks, at least when applied to OCR, produces a directly logarithmic relationship between accuracy and number of hidden units until a maximum is reached. Also, regularization seemed to become unnecessary as the number of hidden units increased. This may be because complex relationships and patterns that aid in the classification do not require high values for the parameters; thus regularization is useless for higher complexity neural network classifiers.

**5.2     Time Complexity**

The time required to train the logistic regression algorithm on average for the optimal classifier that reached 85.14% accuracywas 848s. In comparison, the neural network achieved the slightly higher accuracy of 85.58% in 759s, and continued to reach its maximal accuracy of 90.19% in 1599s. From the data it has been determined that the time complexity of the logistic regression algorithm is O(n), where n is the number of features, a linear relationship. In comparison, the single layer neural network has a complexity of O(√h) where h is the number of hidden units. These complexities may be implementation specific and are calculated with a constant data set of 144000x400 values.

**6      Conclusion**

The neural network achieved approximately 5% better accuracy than the logistic regression algorithm, and was more responsive to optimization. The direct logarithmic relationship between accuracy and number of hidden units until a maximum is reached for a single-layer neural network evidenced by this investigation allows for the determination of the optimal number of hidden units fairly simply. Also, little or no regularization is necessary for larger numbers of hidden units, making optimization easier. The neural network’s superior accuracy and response to optimization, along with its more efficient time complexity of O(√h) makes it more suitable to any application. Logistic regression only benefits from the relative simplicity of its implementation, as well as the fact that it performed best with no feature engineering, although the neural network performed better.

The inaccuracy in both simplified algorithms however would cause multiple errors per page, thus making them unsuitable for applications where high accuracy is important. However, neural networks with multiple layers may increase accuracy significantly. Also, larger datasets with higher resolution images, which could not be used in this investigation due to the memory limitations of Octave, may significantly increase the accuracy of the results [8]. To implement solutions which use more data, as well as increase the speed for serious testing, a faster programming language such as C would have to be used [8]. Further research is needed to confirm the properties of neural network found in this investigation as well as testing the effects of other tunable parameters on accuracy and time complexity.

**7      References**

[1] Eikvil, Line. "Optical character recognition." citeseer. ist. psu. edu/142042. html (1993).

[2] Erhan, Dumitru, Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Visualizing

Higher-Layer Features of a Deep Network." Statistical Learning of Complex Data with Complex Distributions (2009). Web. 13 Jan. 2016. <http://www.iro.umontreal.ca/~bengioy/mprime/Presentations.htm>.

[3] Holley, Rose. "How Good Can It Get? Analysing and Improving OCR Accuracy in Large

Scale Historic Newspaper Digitisation Programs." D-lib Magazine. Web. 3 Jan. 2016.

<http://www.dlib.org/dlib/march09/holley/03holley.html>.

[4] Hosmer, David W., and Stanley Lemeshow. Applied Logistic Regression. Enskede: TPB,

2007. Print.

[5] Karpathy, Andrej. "Neural Networks." Convolutional Neural Networks for Visual

Recognition. Stanford University. Web. 13 Feb. 2016. <http://cs231n.github.io/neural-networks-1/#summary>.

[6] K. Gnana Sheela and S. N. Deepa, “Review on Methods to Fix Number of Hidden

Neurons in Neural Networks,” Mathematical Problems in Engineering, vol. 2013, Article ID 425740, 11 pages, 2013.

[7] Ng, Andrew. "Stanford University Machine Learning (Course Handouts)." CS229 Machine

Learning Autumn 2015. Stanford University. Web. 5 Oct. 2015.

<http://cs229.stanford.edu/materials.html>.

[8] Sarkar, Deepayan. "Optical Character Recognition using Neural Networks." Department of

Statistics University of Wisconsin, Madison (2003).

[9] Tesauro, Gerald, David S. Touretzky, and Todd Leen. Advances in Neural Information

Processing Systems 7. Cambridge, MA: MIT, 1995. Print.

**8      Appendix A: Raw Data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Logistic Regression (Iterations = 100)** | | | | | |
| **Power** | **Lambda** | **% Training set** | **% CV set** | **% Test set** | **Time (s)** |
| 1 | 0 | 90.909722 | 85.083333 | 84.888889 | 925.882454 |
| 0.3 | 90.819444 | 84.888889 | 84.805556 | 818.754829 |
| 1 | 90.152778 | 85.388889 | 85.138889 | 854.409370 |
| 3 | 89.277778 | 85.194444 | 84.388889 | 814.504585 |
| 10 | 87.597222 | 84.638889 | 84.166667 | 806.672134 |
| 30 | 85.527778 | 83.472222 | 83.111111 | 794.272430 |
| 100 | 82.701389 | 81.166667 | 81.055556 | 921.052673 |
| 2 | 0 | 91.208333 | 84.833333 | 85.250000 | 1512.843475 |
| 0.3 | 91.111111 | 84.805556 | 85.361111 | 1629.144180 |
| 1 | 90.756944 | 84.944444 | 85.222222 | 1657.154785 |
| 3 | 90.159722 | 85.388889 | 85.277778 | 1609.975082 |
| 10 | 88.854167 | 85.444444 | 84.555556 | 1650.837921 |
| 30 | 87.111111 | 84.361111 | 83.916667 | 1650.571411 |
| 100 | 84.513889 | 82.750000 | 82.250000 | 1590.516472 |
| 3 | 0 | 91.354167 | 85.055556 | 85.250000 | 2482.912018 |
| 0.3 | 91.201389 | 84.888889 | 85.277778 | 2383.255318 |
| 1 | 91.291667 | 84.583333 | 85.277778 | 2282.487549 |
| 3 | 90.694444 | 85.000000 | 85.055556 | 2346.591209 |
| 10 | 89.513889 | 85.388889 | 84.722222 | 2390.274719 |
| 30 | 87.888889 | 84.638889 | 84.166667 | 2242.522263 |
| 100 | 85.409722 | 83.222222 | 82.805556 | 2399.849258 |
| 4 | 0 | 91.381944 | 84.638889 | 84.972222 | 3356.988007 |
| 0.3 | 91.555556 | 84.444444 | 85.000000 | 3381.260391 |
| 1 | 91.354167 | 84.555556 | 85.194444 | 3349.599083 |
| 3 | 91.083333 | 84.916667 | 85.027778 | 3244.657578 |
| 10 | 89.993056 | 85.388889 | 84.916667 | 3274.399284 |
| 30 | 88.465278 | 85.083333 | 84.166667 | 3233.695953 |
| 100 | 86.041667 | 83.611111 | 83.277778 | 3313.442513 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Single-Layer Neural Network (Iterations = 500)** | | | | | |
| **Hidden Units** | **Lambda** | **% Training set** | **% CV set** | **% Test set** | **Time (s)** |
| 25 | 0 | 82.854167 | 79.861111 | 79.888889 | 662.260000 |
| 0.3 | 81.763889 | 79.055556 | 79.027778 | 522.310000 |
| 1 | 80.680556 | 78.055556 | 77.194444 | 500.590000 |
| 3 | 81.937500 | 80.333333 | 79.055556 | 495.230000 |
| 10 | 85.166667 | 83.666667 | 84.000000 | 526.160000 |
| 30 | 82.555556 | 81.444444 | 81.305556 | 443.110000 |
| 100 | 76.965278 | 76.277778 | 75.833333 | 481.650000 |
| 50 | 0 | 86.083333 | 82.472222 | 82.277778 | 871.650000 |
| 0.3 | 87.673611 | 85.222222 | 84.444444 | 798.090000 |
| 1 | 86.826389 | 83.805556 | 83.944444 | 759.680000 |
| 3 | 87.576389 | 84.666667 | 84.583333 | 789.740000 |
| 10 | 87.902778 | 85.694444 | 85.583333 | 758.570000 |
| 30 | 85.027778 | 83.666667 | 84.000000 | 759.350000 |
| 100 | 78.562500 | 78.111111 | 78.250000 | 788.790000 |
| 100 | 0 | 92.402778 | 87.333333 | 87.138889 | 1009.500000 |
| 0.3 | 91.576389 | 86.972222 | 87.361111 | 1054.700000 |
| 1 | 92.430556 | 87.555556 | 87.388889 | 1066.000000 |
| 3 | 91.180556 | 87.250000 | 86.722222 | 1033.800000 |
| 10 | 87.527778 | 85.277778 | 84.888889 | 1017.500000 |
| 30 | 85.458333 | 84.250000 | 84.027778 | 1045.000000 |
| 100 | 80.777778 | 80.250000 | 80.000000 | 976.410000 |
| 200 | 0 | 96.916667 | 89.888889 | 90.194444 | 1599.400000 |
| 0.3 | 95.194444 | 89.611111 | 89.500000 | 1509.600000 |
| 1 | 93.750000 | 89.500000 | 88.777778 | 1587.700000 |
| 3 | 93.534722 | 89.166667 | 88.277778 | 1620.400000 |
| 10 | 89.854167 | 87.333333 | 86.833333 | 1700.700000 |
| 30 | 86.090278 | 84.833333 | 84.222222 | 1680.600000 |
| 100 | 81.250000 | 80.472222 | 79.972222 | 1818.500000 |
| 400 | 0 | 96.854167 | 90.027778 | 90.111111 | 2404.400000 |
| 0.3 | 95.875000 | 89.611111 | 89.777778 | 2680.800000 |
| 1 | 96.326389 | 90.000000 | 89.666667 | 3229.900000 |
| 3 | 93.236111 | 89.333333 | 88.388889 | 3268.200000 |
| 10 | 89.965278 | 87.361111 | 86.972222 | 3261.200000 |
| 30 | 85.597222 | 84.111111 | 84.138889 | 3419.300000 |
| 100 | 81.208333 | 80.305556 | 79.750000 | 2884.400000 |

**9      Appendix B: Program Code**

**9.1 Logistic Regression Cost Function and Calculation of Gradients**

function [J, grad] = LogisticRegressionCostFunction(theta, X, y, lambda)

**% Number of training examples**

m = length(y);

**% The cost function**

J = (1/m)\*sum(-y.\*log(sigmoid(X\*theta)) - (1-y).\*log(1-sigmoid(X\*theta))) + ...

(lambda/(2\*m))\*sum(theta([2:1:length(theta)], :).^2);

**% Finding the gradients for bias parameter, theta0, and then the gradients for the rest % of the parameters.**

grad0 = (1/m)\*(sum(sigmoid(X\*theta) - y))';

gradj = (1/m)\*(sum((sigmoid(X\*theta) - y).\*X(:, [2:1:size(X, 2)])))' + ...

(lambda/m)\*theta([2:1:length(theta)], :);

grad = [grad0; gradj];

grad = grad(:);

end;

**9.2 Training Logistic Regression using the One vs. All Method**

function [all\_theta] = OneVsAll(X, y, num\_labels, lambda)

**% Number of training examples**

m = size(X, 1);

**% Number of features**

n = size(X, 2);

**% Empty matrix in which to add all parameters for all 36 classifiers**

all\_theta = zeros(num\_labels, n + 1);

**% Add bias feature to the input data**

X = [ones(m, 1) X];

**% Setting options for fmincg**

options = optimset('GradObj', 'on', 'MaxIter', 200);

**% Training 36 classifiers for each class using fmincg**

for i=1:(num\_labels),

theta = zeros(n+1, 1);

[temp] = fmincg (@(t)(LogisticRegressionCostFunction(t, X, (y == i), ...

lambda)),theta, options);

all\_theta(i, :) = temp;

end;

end;

**9.3 Single-Layer Neural Network Forward and Back Propagation**

function [J grad] = NeuralNetworkCostFunction(nn\_params, input\_layer\_size, ...

hidden\_layer\_size, num\_labels, X, y, lambda)

**% Reshaping the matrices of parameters Theta1 and Theta2**

Theta1 = reshape(nn\_params(1:hidden\_layer\_size \* (input\_layer\_size + 1)), ...

              hidden\_layer\_size, (input\_layer\_size + 1));

Theta2 = reshape(nn\_params((1 + (hidden\_layer\_size \* (input\_layer\_size + 1))):end), ...

               num\_labels, (hidden\_layer\_size + 1));

**% Number of training examples**

m = size(X, 1);

**% Performing forward propagation**

X = [ones(m, 1) X];

z2 = X\*Theta1';

a2 = sigmoid(z2);

a2 = [ones(m, 1) a2];

z3 = a2\*Theta2';

a3 = sigmoid(z3);

**% Creating a 144000x36 matrix of binary labels**

ny = zeros(m, size(a3, 2));

for i = 1:m,

ny(i, y(i)) = 1;

end;

**% Cost function**

J = (1/m)\*sum(sum(-ny.\*log(a3) - (1-ny).\*log(1-a3)));

**% Adding regularization to the cost function**

J = J + (lambda/(2\*m))\*(sum(sum(Theta1(:, [2: size(Theta1, 2)]).^2)) + ...

sum(sum(Theta2(:, [2: size(Theta2, 2)]).^2)));

**% Calculating error in every logistic unit**

d3 = a3-ny;

d2 = d3\*Theta2(:, [2:end]) .\* sigmoidGradient(z2);

**% Creating copies of Theta1 and Theta2 without the bias parameter for regularization as**

**% the bias parameter is not regularized**

regTheta1 = Theta1;

regTheta1(:, 1) = zeros(size(Theta1, 1), 1);

regTheta2 = Theta2;

regTheta2(:, 1) = zeros(size(Theta2, 1), 1);

**% Final accumulators of error. These are the gradients for Theta1 and Theta2**

D2 = (1/m)\*(d3'\*a2) + (lambda/m)\*(regTheta2);

D1 = (1/m)\*(d2'\*X) + (lambda/m)\*(regTheta1);

grad = [D1(:) ; D2(:)];

end;

1. Special thanks is given to T de Campos from Microsoft Research India for creating this dataset and making it available. It can be found at http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/ [↑](#footnote-ref-1)
2. Thoma, Martin. Sigmoid Function. Digital image. Wikimedia. N.p., 25 May 2014. Web. 7 Jan. 2016. <https://commons.wikimedia.org/wiki/File:Sigmoid-function-2.svg>. [↑](#footnote-ref-2)
3. Special thanks are given to Carl Edward Rasmussen for creating the “fmincg” minimization function (copyright 2002) used for training in this investigation for both logistic regression and the neural network. It is freely available for education and research purposes. It was obtained on the Coursera online course “Machine Learning” by Stanford University at <https://www.coursera.org/learn/machine-learning/> [↑](#footnote-ref-3)
4. Ng, Andrew. Neural Network Architecture. Digital image. Feature Space. N.p., 9 Nov. 2011. Web. 7 Jan. 2016. <http://feature-space.com/en/post45.html>. [↑](#footnote-ref-4)