# Letter Recognition using Deep Feed Forward Neural Networks

Anish Lakhwara
The Australian National University, Canberra
u5838848@anu.edu.au

**Abstract.** Shallow/deep feed forward neural networks were trained using the back propagation algorithm on a large categorization data set of 20,000 unique letters to determine how well this task could be executed by a machine learning algorithm. The deep neural network employed a scheduler, mini-batches as well as drop out algorithms. The parent data set was generated by distorting 26 uppercase letters from 20 different fonts and were then summarized into 16 numerical attributes. Different sizes, learning rates and network topologies were tested to find optimal results, which are evaluated using an accuracy measure and some cases a confusion matrix. The results of using a deep neural network, with all these additions were quite abysmal, with only a 3.64% testing accuracy. The shallow neural net performed significantly better without optimizations on the data set with 13.5% accuracy.

**Keywords:** Machine learning, neural networks, feed forward, back propagation, categorization,

## 1. Introduction

Since the dawn of computing human beings have been trying to get computers to perform complex tasks at a competency level that is indistinguishable from actual humans (turing test[3]). While computers have easily been able to outperform humans at certain tasks (memory, computation), only in the last two decades have computers become masters of complex games such as Chess (Deep Blue[1]) and Go (AlphaGo[2]), that humans were previously undefeated in. These class of problems can be split into two different components – Understanding the current state of the situation (categorization), and applying a string of actions which leads to beneficial outcomes (decision-making). The broad field of computing commonly referenced to as "Machine Learning" (ML) has lead the charge against humanity in many of these endeavors. However utilizing these algorithms required either vast amounts of computing power or huge swaths of data or both. In recent years the advancement in computing power and the proliferation of data has sparked a renewed interest in

machine learning algorithms, making them common, everyday options available on over a billion devices. The application of neural networks to the first component of these problems, that is the 'categorization aspect', is not new, however their high success rates across various different categorization problems (**LINKS**) in recent history has made them a popular option for solving such types of problems.

This research paper focuses on using a supervised learning algorithm or neural network, to categorize a large number of examples presented to the network appropriately. I wanted to test how accurately a relatively small neural network (less than 5 layers) and a deeper neural network (with 7 fully connected layers) would perform on a complex data set with many examples, even though the data set may not be large enough for such a deep neural network. The data set I chose to use in this paper was acquired from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/Letter+Recognition). I chose this data set because of the (20,000) size of the data set as well as it's complexity. The original paper[4] that utilized this data set dates back to 1991, and is using a very different method to solve the classification problem. Instead of relying on neural networks Frey and Slate decided on using Holland-Style adaptive classifiers. They utilize fuzzy logic and rule-based systems to classify the data set to high degrees of success (at best they achieved 80% classification accuracy) and so I wished to compare small neural networks against their previous attempts. Other papers that use this data-set as a classification problem also exist and use a variety of methods, both supervised and unsupervised with somewhat successful results.

## 2. Data Set Attributes

This data set contains 20 different fonts which represent five different types of stroke styles (simplex, duplex, triplex, complex and Gothic) and six different letter styles (block, script, italic, English, Italian and German). Covering a large amount of different stroke and letter styles adds diversity to our data set which helps to represent "real world data" accurately. Furthermore each character image was distorted along one of various axis. Characters could be linearly magnified, have their aspect ratio changed (horizontal magnification) and be horizontally or vertically "warped". This distortion considerably increases the complexity of the data set which is part the reason I ended up choosing it as a complex data set raises many challenges for a shallow neural network.

To produce the resulting data set that was used as input, each character was condensed down to 16 integer attributes scaled to an integer range of 0-15. These attributes are available in the original paper and will not be copied here for brevity sake. However I will include that from these attributes the original character cannot be uniquely determined, as these attributes are mostly different means that can be

calculated from the letters “on” pixels. This creates further challenges for the neural network to correctly classify the data as it does not have the original image of the character to go off of, but instead attributes of the character.

Finally there were some transformations I applied to this data set to make output easily classifiable for the neural network. Of the 20,000 samples in the data set I split them into two different sets for testing and training. The training data set contains 15000 of the original samples and the testing data set contained the remaining 5000. Given the large size of the data set I thought this was a fair split for our neural network, though it is worth noting that the original paper had a training set of 16,000 and testing set of 4000.

Also transformed was the character output. Originally each of the character outputs was referenced by letter, and I transformed this to an integer range of 0-25 to include all 26 characters of the alphabet for ease of classification for the neural network.

For the deep neural networks the range of all the attributes was scaled down to be between 0 – 1. This scaling was applied so that larger weights do not carry more significance in the network. It should have also been applied to the shallow neural networks.

## 3. Description of the Neural Networks

A variety of different network sizes, structures and training times and parameters were tested to see which would produce the best output for the shallow network. However for all these networks were trained with Stochastic Gradient Descent (SGD) as an optimizer, and performance was evaluated using cross-entropy, and since this is a classification problem, these are the standard choice of algorithm. The same optimizer and loss algorithms were used for evaulation of the deep neural network, however there were additional algorithms used to further optimise training time, performance and improve resiliency of the network. The deep neural network utilized mini-batches, drop out layers, as well as a scheduler to vary learning rate.

Deciding upon the number of input neurons and output neurons was fairly straightforward for both shallow and deep neural networks. Each attribute had one neuron assigned to it, resulting in 16 input neurons. There were 26 output neurons each relating to one letter of the alphabet. Choosing the number of hidden neurons was the subject of much experimentation during the shallow neural network phase of this paper. The results of the experimentation are captured in the table below.

| Hidden Neurons | Learning Rate | Epochs | Testing accuracy |
|---|---|---|---|
| 12 | 0.01 | 500 | 4.32% |
| 24 | 0.01 | 500 | 5.46% |
| 48 | 0.01 | 500 | 6.4% |
| 96 | 0.01 | 500 | 2.95% |
| 50 | 0.01 | 500 | 5.4% |
| 44 | 0.01 | 500 | 5.1% |
| 48 | 0.01 | 1000 | 6.78% |
| 48 | 0.01 | 2000 | 5.5% |
| 48 | 0.01 | 4000 | 7.12% |
| 48 | 0.01 | 10000 | 13.55% |
| 48 | 0.02 | 3000 | 13.19% |
| 48 | 0.03 | 3000 | 7.9% |

Results of experimentation with a shallow single hidden layer neural networks

Having then exhausted the possibilities of tinkering with parameters of a single hidden layer neural network I moved on to exploring the outcomes of a neural network with two hidden layers with an epoch size of 3000. I decided to split the 48 single layer into two layers each consisting of 24 neurons each. This performed far below expectation with a testing accuracy of 1.24%. Assuming I had far too many neurons I dropped the number of neurons in the first layer to 18 and the second layer to 12, producing marginally better results at 4.56% testing accuracy. Moving on to a neural network with 10 hidden neurons in the first layer and 8 in the second layer resulted in further marginal improvements with an accuracy rate of 5.89% in training and 5.3% during testing. Going below this with 8 neurons in the first hidden layer and 6 in the second led to a much reduced accuracy of only 2.4%.

For implementation of the deep neural network, I first scaled the values in each of the attributes to be between 0 – 1 and I decided on first using 5 layers each with between 60 – 200 neurons. In order to overcome the vanishing gradient problem I used ReLUs as the activation function instead of the standard sigmoid activation function. During training I noticed that many of the ReLUs were “dying” (ie, they remained 0 forever) in layer 1. Because of the way back propagation works, this implied that there were dead ReLUs further down the neural network. In order to

remedy this, I tried a variety of different activation functions, including leaky ReLUs, RreLUs, and Hardtanh. However even after solving the dead ReLU problem, the accuracy of the neural network was very low. I then decided to lower the number of neurons in each layer, but kept the first layer larger than the rest. The intuition behind this is that the first layer will have various simple properties to understand and the later layers will use these simple properties, and won't need to come up with as many complex ones. Still the accuracy rate was abysmal. In order to improve this, I then implemented some other functions, including utilizing mini-batches during testing, implementing a scheduler to change the learning rate as the epochs progress, and implementing some drop out layers. Despite all these additional improvements, the accuracy remained pityfully low, lower than even the shallow network, at a maximum of just 4.67%. Having reached my wits end, I concluded that the data set was simply too small and not varied enough for a deep neural network to performally effectively.

## 4. Results and Future Work

Comparing results to various other studies that utilized the same data set with very different methods, the outcome of uing neural networks on this data set leaves a lot to be desired. With other papers getting accuracy of over 80% while utilizing "older" methods such as clustering and decision trees, neural networks should be able to perform better on this data-set.

What the results above definitively prove is that there is much more work to be done with this data set when it comes to neural networks. Achieving only a measly 13.5% accuracy at best is disappointing. This was achieved through an epoch size of 3000, 48 hidden neurons and a learning rate of 0.02, using the back propagation algorithm with SGD as the optimizer and cross-entropy as the evaluation. While this does fit my description of a shallow neural network and had a short training time, the performance leaves a lot to be desired.

There are a few reasons that performance was sub-par. For one the high complexity of the data set makes it inherently difficult for a shallow neural network to work with. The large number of differences between examples of the same class, as well as having to deal only with abstract attributes of the characters instead of images increases the number of input neurons and adds additional complexity. Instead of simplifying the data set, a transformation on the range of the integer values of the attributes from the current 0-15 scale to a 0-1 scale could drastically improve performance even on a shallow neural network. Because of the way back propagation works, higher values have a larger impact while training neurons, which heavily skews results in favor of higher values, thus scaling the values down to a range of 0-1 would solve this skew and possibly improve accuracy of the neural network.

What's even more disappointing is the results from the deep neural network. Achieving approximately the same results as random guessing would, despite various methods for improving accuracy, I concluded that this data set is simply too small for a deep neural network to be appropriate to be used here. Given the improvements that could be made to the results of a shallow neural network, it seems likely that a deep neural network is not fit for the task of accurately classifying this data.

Further research could be conducted into different optimizer algorithms, activation functions as well as evaluation metrics. The current combination of SGD, Linear activation and cross-entropy as a loss metric though common may not be the best fit for this data. Through the deep neural experiments we have shown that deeper neural networks do not corrospond with higher accuracy rates, and that for some problems, a better solution is to use shallow neural networks.

## 5. References


1. Feng-Hsiung Hsu (1995) https://ieeexplore.ieee.org/abstract/document/755469/ IBM's Deep Blue Chess grand master chips. IEEE Micro ( Volume: 19, Issue: 2) ISSN: 0272-1732
2. How Google's AlphaGo beat a Go World Champion (2016) https://www.theatlantic.com/technology/archive/2016/03/the-invisible-opponent/475611/
3. Alan Turing (1950) https://link.springer.com/chapter/10.1007/978-1-4020-6710-5_3 Computing Machinery and Intelligence
4. P. W. Frey and D. J. Slate (1991). https://link.springer.com/article/10.1007/BF00114162 "Letter Recognition Using Holland-style Adaptive Classifiers". (Machine Learning Vol 6 #2 March 91)
5. Nele Verbiest, Sarah Vluymans, Chris Cornelis, Nicolás García-Pedrajas and Yvan Saeys (2016) https://www-sciencedirect-com.virtual.anu.edu.au/science/article/pii/S1568494616301247 Improving nearest neighbor classification using Ensembles of Evolutionary Generated Prototype Subsets
6. Pytorch Weight Pruning https://github.com/wanglouis49/pytorch-weights_pruning
7. Pytorch tutorial https://github.com/MorvanZhou/PyTorch-Tutorial
8. Pytorch docs https://pytorch.org/docs/stable/nn.html
9. Xioli Z. Fern and Carla E. Brodley Clustered Ensembles for high dimensional Clustering: An Emperical Study

10. Unsupervised Clustering Of Temporal Patterns in High-dimensional Neuronal Ensembles Using a Novel Dissimilarity Measure Lukas Grossberger-Francesco Battaglia-Martin Vinck – 2018