Neural Network for Digit Classification

Shaishavkumar Jogani (1212392985) Siva Kongara (1212345483)

Abstract—This project implements a Neural Network for the purpose of digit classification. The network contains two hidden layers and is implemented in python without using any advanced machine learning libraries. The network parameters are learned though the backpropagation algorithm. MNIST dataset is used to train and test the network. We have also implemented Dropout -- a regularization scheme to prevent overfitting and analysed the effect of dropout on training.

Index Terms—Back Propagation, Digit Classification, Dropout, Neural Network, Stochastic Gradient Descent

1 Introduction

Digit classification is the task of classifying a given image into one of the digits from 0 to 9. This is of key importance in modern day machine intelligence and has many practical applications such as Natural Language Processing (to autograde), Self Driving Cars (to detect speed limits) etc. This is a typical classification problem. A classification problem consists of a set of classes and the machine has to classify a given object into one of the classes. Other examples of a classification problem include facial recognition, image classification (identifying objects as car, bus, dog) etc. This classification problem can be addressed using many techniques such as Linear Regression, K-nearest-neighbours, SVM, Neural Network, Convolutional Nets and many other such machine learning techniques. In this project we have done the task of digit classification using a Neural Network. The motivation for this project comes up from the fact that digit classification has been an important research area for exploring several learning techniques ranging from automatically learning feature representations, learning classifiers invariant to distortions, matching and alignment based distances, and learning multilayered representations of data[1].

2 BACKGROUNDS

2.1 Neural Network

The term 'Neural Network' is inspired from the way biological nervous systems, such as the brain, process information. Human nervous system contains millions of neurons and each neuron propagates the information forward to another neuron. Based on the input signal that a neuron receives, it is either activated or not. The key idea of this paradigm is the novel structure of the artificial networks[2]. A NN consists of an input layer, any number of hidden layers (including 0), and an output layer. Each layer has a collection of neurons and all the layers are connected in an acyclic manner. In other words, the outputs of some neurons in a layer will be the inputs to other neurons in the above layer. Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs including but not limited to sales forecasting, business marketing, customer research, data validation, risk management, medicine, robot learning etc.

The key difference between linear regression and a neural network is that the output can be modelled as a nonlinear function of the input. This nonlinearity arises because of the hidden layers. It is equivalent to mapping the input to higher dimensions and thus classifying the samples. The mdel is stored in the weights and biases. Weights are the parameters that are used to weigh the incoming inputs to a neuron, thus suppressing some of the inputs while enhancing the others. Then the bias is added, and it is passed through an activation function which decides whether the neuron should be activated or not.

So, fundamentally training a network boils down to the problem of learning these weights such that the predicted result at the output layer is as close to the actual result. There are many types of error functions that we can define when comparing the predicted result to the actual result. Some of them are the Mean Squared Error, Maximum Likelihood, Cross-Entropy loss, etc. For classification, cross-entropy loss function performs better than the other loss functions. This is because the problem of classification is a discrete one. The predicted value in this case is not a continuous numeric value, but a single class. So, if we have some kind of outputs for each class which represents the probabilities of the object belonging to that class, then we can use the Cross-Entropy loss to learn the parameters.

2.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a stochastic approximation of the gradient descent optimization and an iterative method for minimizing the loss function. Loss is defined as some kind of difference between the predicted output and the desired output. It is an iterative method in which the network is modified after each training sample is fed through the network and thus updating the network parameter in the direction in which the loss is minimised. SGD computes the gradient using a single sample and is much faster than the traditional batch descent algorithm. Large datasets often can't be held in RAM, which makes vectorization much less efficient. Rather, each sample or batch of samples must be loaded, worked with and the results are stored. This is computationally less expensive and also SGD is most useful when the objective function is non-convex. An objective function is termed non-convex if it has more than one local minima. In batch

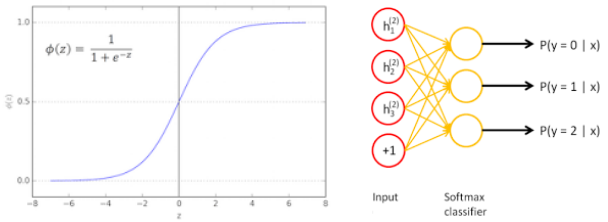


Fig. 2.3.1. Sigmoid on the Left. SoftMax on the Right.

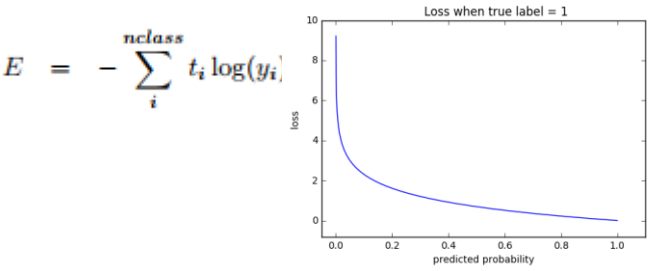


Fig. 2.4.1 Cross Entropy loss.

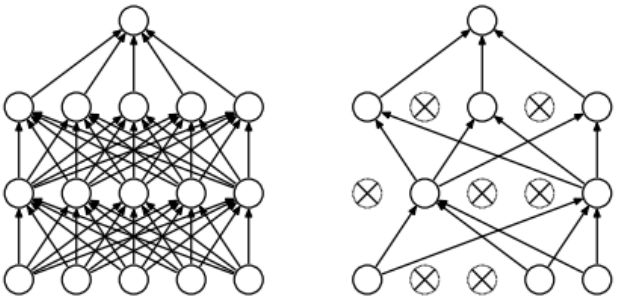
gradient descent we step down the true gradient and thus may eventually converge to local minima. SGD is better in finding the global minima. SGD involves two main steps: Forward Propagation and Backpropagation.

2.3 Forward Propagation

As discussed before, an artificial neural network (ANN) consists of an input layer, an output layer, and any number of hidden layers situated between the input and output layers. The feed-forward computations performed by the ANN are as follows: The signals from the input layer are multiplied by a set of fully-connected weights connecting the input layer to the hidden layer. These weighted signals are then summed and combined with a bias. This calculation forms the pre-activation signal for the hidden layer. The pre-activation signal is then transformed by the hidden layer activation function to form the feed-forward activation signals leaving leaving the hidden layer. Similarly, the activations of the other hidden layers are calculated from the previous hidden layer. In a similar fashion, the hidden layer activation signals are multiplied by the weights connecting the hidden layer to the output layer, a bias is added, and the resulting signal is transformed by the output activation function to form the network output. The output is then compared to a desired target and the error between the two is calculated. This entire process is called feed forward propagation.

The activation functions should be such that they should be differentiable, and this is important while propagating the error back through the network. Sigmoid activation [5] is defined as follows: $f(x) = 1/(1+e^{-x})$ and the derivative of the sigmoid activation is $f'(x) = f(x)(1-f(x))$. Also, a sigmoid function maps any arbitrarily large value to a range between 0 and 1. Thus, it can be thought as a function that decides whether a neuron is activated or not.

The softmax activation [6] is also known as the normalized exponential function. This activation is generally



(a) Standard Neural Net (b) After applying dropout.

Fig. 2.5.1. Dropout Architecture

applied to the output layer in a classification problem. It is defined as follows: $f(z_k) = e^{Z_k} / \Sigma\, e^{Z_k}$ for k = 1 to C, where is the total number of classes to be classified into. This function reduces the value of each neuron in the output layer to be in the range of [0,1] and all the values of the output layer add up to 1. This behaviour is similar to probability and thus the outputs can be interpreted as the probability with which the given object belongs to each class. This can be represented as shown in the Fig. 2.3.1. Ideal output should be that these output values should have 1 for the class to which the object belongs and 0 for all the other classes.

2.4 Back Propagation

Backpropagation is the process of propagating the error that has been calculated at the end of forward propagation back through the network and update the network parameters so that the error is minimised. To do so, we make use of the chain rule to calculate the derivative of the loss with respect to parameters. Once we have the gradients of loss with respect to each of the network parameters, we update the parameter using a learning rate. This means that we change the values of the parameters in the direction in which the loss reduces.

We need to calculate the error between the predicted output and the actual output at the end of the forward propagation step. One type of such loss function is the Categorical Cross Entropy loss [7]. The loss is defined as shown in Fig. 2.4.1 and the loss behaviour is shown also in the same figure.

In the above definition, y_i is the predicted output (probability obtained after applying softmax activation) and t_i is the actual output for the corresponding classes. Since this loss assumes that the predicted output is some kind of probability, it is often used along with the softmax activation on the output layer for a multi class classification problem. The main advantage of using cross entropy error over mean squared error is that the training doesn't stall. This is because the predicted output should match either 0 or 1 for each output node and in mean squared error, the gradient that is propagated back contains a term (output)(1-output) which diminishes as the training progress. This leads to a very minimal change in the network parameters.

2.5 Dropout

Neural network with large number of parameters are very powerful but they are prone to the issue of overfitting. As the machine trains on the training samples, the machine tries to change its model (network parameters), to best fit the training data. This is not necessarily a desired thing as the main objective of the network is to classify/predict on a new test sample. Such a network which has 100% accuracy on the training data may not perform in general on a new test data. This is called the problem of overfitting as the network fits itself to the training data in a tight manner. This problem can be addressed by combining the predictions from many different networks at test time. But large networks are slow to train and use. So, Dropout is an effective way to prevent overfitting in the network. The key idea of Dropout is to randomly drop hidden units to prevent them from co-adapting too much. So, every time a sample is trained, some of the hidden units are dropped at random as shown in the Fig. 2.5.1. Such a network is called a thinned network and, so we train on many such thinned networks. These thinned networks are exponential in number and thus effectively giving us exponentially large number of models to predict the output from. At test time, we use all the units to predict the output and to have the output at test time to be equal to the expected output during training, we multiply the weights with the probability with which the unit is retained during the training [8].

The effect of dropout on training is that as the iterations increase, the accuracy of the network remains the same or increases but doesn't get worse. Without dropout, after some iterations, the network overfits the training data and thus starts performing badly on the validation and test set.

3 IMPLEMENTATIONS

3.1 Architecture

In this project, we have implemented a neural network for the purpose of digit classification. Our implementation is in python and does not use any advanced libraries. There are two hidden layers with 256 neurons in each hidden layer and an output layer with 10 neurons corresponding to the ten digits from 0 to 9. The dataset that we have used in this project is the MNIST dataset in which each image is represented by 28 x 28 pixels. Each pixel's value ranging from 0 to 255. We use these 784-pixel values to represent the input and to train the network. So, we have 784 neurons in the input layer. This constitutes the structure of our neural network as shown in Fig. 3.1.1.

3.2 Preparing the training data:

MNIST database [3] of handwritten digits is used to train the network and to test the accuracy. The data is available in a byte format and we have used the python-mnist library [4] to read the dataset which will return a list of all

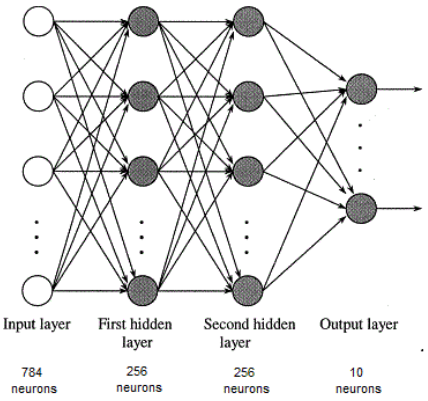


Fig. 3.1.1. Structure of our Neural Network

the images and the labels corresponding to each image. Each image returned by this library will be in the form of a list of 784 features corresponding to the 784 pixels with a value from 0 to 255. The label corresponding to that image is a number from 0 to 9. As the loss that we are about to calculate is categorical cross entropy, we have encoded the labels as one-hot vectors [9]. So, that means a label of '2' has been encoded as a list of 10 in which the second index is 1 and the rest are zeros i.e. [0,1,0,0,0,0,0,0,0,0].

There are 60000 images in the data set and we have sampled randomly 5000 out of them for validation and 5000 for testing. From the rest 50000, we have performed experiments changing the training size from 10000 to 50000. Since we have randomly sampled all the images, our assumption is that there are equal number of samples from each class in all the three splits. The image data has pixel values ranging from 0 to 255, so we have normalized each pixel value to be in the range [0,1] by dividing each value by 255. It makes the weak input weaker and the strong input stronger. This can be thought as each pixel either being on or off and it helps the algorithm to converge faster.

3.3 Network Parameters

We have defined six network parameters as follows:

W1: From input layer to the first hidden layer. Since input layer has 784 neurons and the first hidden layer has 256 neurons., this will be a 784 x 256 vector.

B1: This is the bias applied to each of the neuron in the first hidden layer, a 256 x 1 vector.

W2: From the first hidden layer to the second hidden layer. Since each of the hidden layers has 256 neurons, this will be a 256 x 256 vector.

B2: This is the bias applied to each of the neuron in the second hidden layer, a 256 x 1 vector.

W3: From the second hidden layer to the output layer. Since the second hidden layer has 256 neurons and the output layer has 10 neurons, this will be a 256 x 10 vector.

B3: This is the bias applied to each of the neuron in the output layer, a 10 x 1 vector.

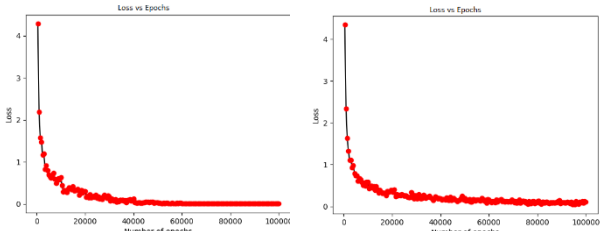


Fig. 4.2.1. Loss vs Epoch no dropout (left) and with dropout

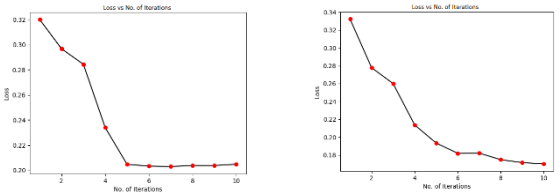


Fig 4.2.2 Loss vs Iteration; No dropout(left) vs with dropout(right).

Initially all the parameters are randomly initialised in the range of [0,1] with a mean of 0.5. However, we have observed that this initialisation lead to a very slow convergence and we have researched that initialising weights with a mean 0 would give us better results. So, we have initialised all the weights and biases in the range of [-1,1] with a mean of 0 [10].

3.4 Stochastic Gradient Descent

We have implemented Stochastic Gradient Descent on each of the training sample in the training dataset. Each training sample is fed to the input and the output value is calculated through forward propagation. Then, using Back Propagation we have propagated the loss back to each layer and adjusted the weights and bias at each layer for each of the training sample. Thus, in our implementation each training sample is an epoch. In each iteration of training, the network is trained on all the training samples and then the network is evaluated on the validation dataset at the end of each iteration.

3.5 Forward Propagation

During forward propagation, at each of the hidden layers a sigmoid activation function is applied to each neuron in the hidden layers. For each of the output neuron, softmax activation is applied.

3.6 Dropout

To implement dropout in this network, we have defined a parameter called dropout rate and an element wise multiplication of each of the neuron in the hidden layer with a 256 x 1 mask generated out of a binomial distribution with dropout rate as its parameter. Our dropout rate parameter indicates the proportion of the number of hidden units that are dropped.

3.7 Back Propagation

The error between the predicted output and the actual label which is encoded as a one hot vector is back propagated and the partial derivatives of the loss with respect to each of the network parameters are calculated [11]. A learning rate of 0.1 is used initially, and it is decayed after every 5000 epochs at a rate of 0.005. Finally, the learning rate is fixed at 0.001.

After the partial derivatives of the loss with respect to the network parameters are determined, the parameters are updated. The partial derivatives at each layer are derived using the result from the above layer and the chain rule.

Update step: H is the cross-entropy loss function and the update step is shown below for W1 and similar step is done for all the other parameters (W2, W3, B1, B2, B2).

$\Delta W1 = \delta H/\delta W1$
$W1 = W1 - \eta * \Delta W1$

3.8 Validation and Testing

In each iteration, the training dataset is shuffled around so that each sample has no dependency on its previous or the next sample and is trained as if it is chosen independently. This leads to a faster convergence and is more efficient [12]. This also prevents any cycles in training.

Once the training is completed, we have run the neural network on the test dataset to calculate the accuracy and thus measuring the performance of our digit classifier.

4 EXPERIMENTS AND RESULTS

4.1 Nuerical Gradient Check

For the first training sample, we have numerically calculated the gradient of loss with respect to the the first weight in W3 and it is found to be $2.0094e^{-7}$. We compared it to the analytical gradient obtained through the back-propagation formulas which is $2.0099e^{-7}$. The values matched to a great extent and thus confirming that our analytical derivatives are indeed correct, and we can procced with training [13].

4.2 Loss behavior

We have trained the network on 10000 samples and for 10 iterations

4.2.1 Training

We have plotted the loss behavior against the number of epochs in Fig. 4.2.1. Since our implementation is a true SGD, there would be a loss at each epoch, but to show the overall loss behavior, we have averaged out the loss over every 500 epochs.

4.2.2 Validation

At the end of each iteration, the network is evaluated on the validation dataset and the loss is plotted against the iteration in Fig. 4.2.2. As expected the loss reduces as the iterations increase. After the first iteration itself, the loss reduces significantly.

4.3 Accuracy

As the training progressed, we have plotted accuracy of the network on the validation dataset in Fig. 4.3.1. Like the loss behavior, we have a very high accuracy after 1

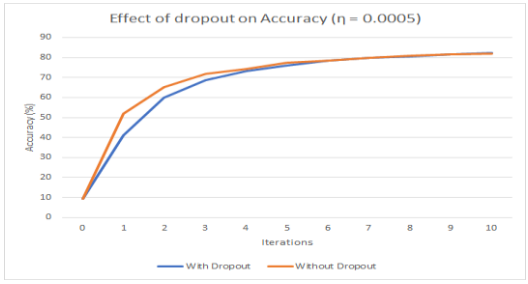


Fig 4.3.1. Accuracy vs Iterations.

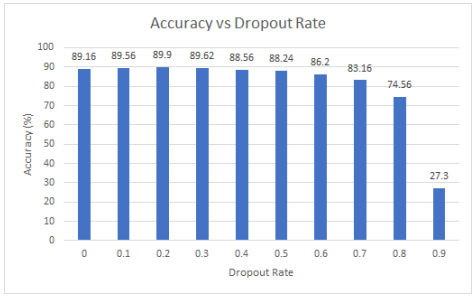


Fig. 4.4.1 Effect of Dropout Rate on Accuracy

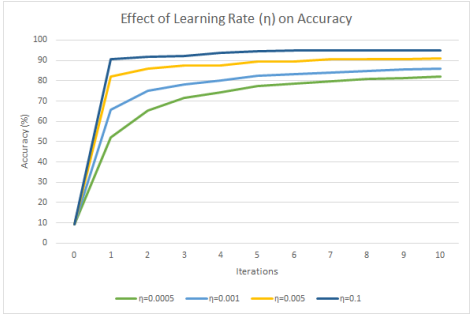


Fig. 4.5.1 Effect of learning rate on accuracy.

iteration and it continues to improve.
Once the training is complete, the network is evaluated on the testset and the accuracy is obtained as **95.12%**

4.4 Effect of Dropout Rate

We have changed the dropout rate (the proportion of units dropped at each hidden layer) and have plotted in Fig. 4.4.1. the accuracy on the testset after training for one iteration of 10000 samples. We observed that a dropout rate of 0.2 i. e 20% of hidden units dropped, yields the highest accuracy. So, we have chosen 0.2 as the dropout rate for the rest of the experiments.

4.5 Effect of Learning Rate

We have analysed various learning rates by plotting the accuracy on the validation dataset vs the number of iterations. As shown in the Fig. 4.5.1, for a learning rate of 0.1, the learning is very fast, and a high accuracy is reached after one iteration, whereas for a low learning rate, it took little more iterations to reach a good accuracy. So, we have chosen to use a learning rate of 0.1.

4.6 Effect of Dropout

4.6.1 Training

To illustrate the behavior of overfitting, we have trained the 10000 samples for 50 iterations and have plotted the loss on the validation dataset with and without using Dropout. As we can see in Fig. 4.6.1, when there is no dropout, after a certain number of iterations, the network instead of improving on the validation dataset, it started to deteriorate as the loss started to increase. However, when we use dropout, even at the end of all the iterations,

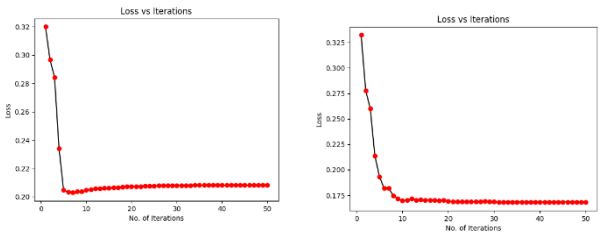


Fig. 4.6.1 Effect of dropout on loss for 50 iterations.

the loss either decreased or remained constant, but never increased. This clearly illustrates the fact that dopout has avoided overfitting the data.

4.6.2 Testing

We have trained the network on 10000 samples for 10 iterations for a dropout rate of 0.2 and have evaluated the performance on the testset. We have achieved an accuracy of **95.44%.** This accuracy is better when compared to the

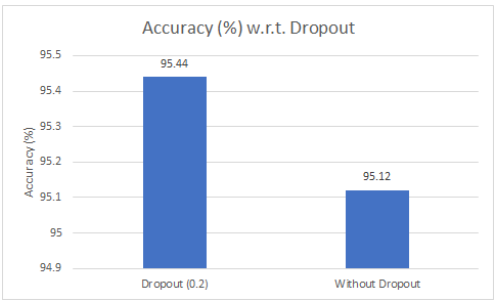


Fig. 4.6.2 Accuracy with dropout and without dropout.

network with no dropout and can be illustrated in the Fig. 4.6.2.

4.7 Effect of Training Samples

We can see as shown in Fig. 4.7.1 that accuracy increases

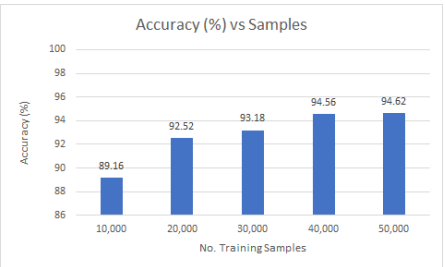


Fig. 4.7.1 Accuracy vs Training Samples (1 iteration)

with the number of training samples. We were able to achieve **98.26%** accuracy while training the network on 50,000 samples for 5 iterations.

4.8 Confusion Matrix

From the confusion matrix shown in Fig. 4.8.1, we can interpret the digits with which a given digit is confused.

Digits	0	1	2	3	4	5	6	7	8	9
0	96.20	0.20	0.20	0.20	0.00	1.30	0.80	0.40	0.80	0.00
1	0.00	98.30	0.40	0.60	0.40	0.00	0.20	0.00	0.00	0.20
2	1.40	0.20	93.50	0.40	0.80	0.20	1.00	1.00	1.00	0.40
3	0.40	0.00	1.80	92.20	0.20	1.20	0.80	1.00	1.20	1.00
4	0.20	0.00	0.40	0.00	96.60	0.00	0.60	0.00	0.40	1.70
5	0.60	0.00	0.00	2.60	0.00	94.50	0.80	0.00	1.00	0.40
6	1.00	0.00	0.20	0.00	0.00	0.60	97.80	0.20	0.20	0.00
7	0.00	0.40	0.60	0.00	1.40	0.20	0.00	94.50	0.60	2.20
8	0.60	0.40	0.20	2.00	0.20	1.00	0.40	0.00	93.50	1.60
9	0.40	0.60	0.20	0.40	3.20	0.40	0.00	0.80	0.40	93.60

Fig. 4.8.1 Confusion Matrix. (10,000 samples x 10 Iterations)

5 OBSERVATIONS

The loss function of a neural network with one or more hidden layers with nonlinear activations is neither convex nor concave, but instead it is non-convex. The reason for this, is the nonlinear activation functions. The nonlinear activation function corresponds to a non-convex optimization. So, a multilayered network with all linear activation function can still be convex but once a nonlinear activation function is used in any one of the layers, the optimization problem becomes non-convex because of the second order derivatives of the nonlinear activation functions are neither positive semidefinite, nor negative semidefinite. Also by swapping the parameters of nodes in a layer and doing the corresponding swaps in the layers above until the output layer, we can achieve the same loss. So, we have multiple solutions for the same loss value which are all the local minimas.

The purpose of a nonlinear activation function is to model a behavior which is not linear in nature. If all the layers have a linear activation function, then no matter how many layers we use, it would still behave as a single layer perceptron and is equivalent to a linear regression model which is not very powerful. Also, while back propagating the error, the gradient of a linear function is a constant and is not dependent on the input. So, we cannot determine the change of error with respect to a change in the input which is bad for learning the weights.

The way we update the weights is by back propagating the error back through the network. The objective is to calculate how the loss changes with respect to changes in the weights at each level in the network. We can calculate the partial derivative of the loss with respect to any weight by traversing through the nodes from that weight to the output layer. In other words, this can be done in reverse order by first calculating the partial derivatives at the output layer and using the chain rule, we can calculate the derivatives at the layers below. So, by storing the results of these partial derivatives at the top layers, we can reuse them to calculate the derivatives at the layers below them. This is the core idea of dynamic programming: to reuse the results of a smaller problem to find the solution of a larger problem. This leads to an increase in the computational efficiency. So effectively backpropagation is a combination of using the chain rule and dynamic programming [14].

6 CONCLUSION AND FUTURE SCOPE

We have seen that Neural network is a powerful tool to encode nonlinear behavior between entitities. We have also seen that choosing the activation functions and the loss functions are key to the purpose of the network. We need enough training samples to learn the network parameters through the back-propagation algorithm which is a combination of the chain rule and dynamic programming. However powerful the neural network is, there is always the problem of overfitting when the network is overtrained. We have seen one regularization scheme called Dropout that prevents the network from overfitting and thus providing a better performance. Dropout effectively prevents the nodes of the hidden layers from coadapting too much and thus making sure that each node has enough information to decide.

When an image is shifted to ledt or right by one pixel, the entire input to the network is now different from the previous image. This is because we have the pixel values as the input features. The network should still be able to predict the output with minimal training. This is possible only if we have shift invariant features. Convolutoin neural networks deal with this shift invariance and are much more powerful in predicting a new sample with minimal training samples. So, as continuation to this project, a convolution neural network can be implemented, and the performance can be analysed.

REFERENCES

[1] Subhransu Maji and Jitendra Malik. "Fast and Accurate Digit Classification."

[2] Christos Stergiou and Dimitrios Siganos. "NEURAL NETWORKS."

[3] http://yann.lecun.com/exdb/mnist/

[4] https://pypi.python.org/pypi/python-mnist

[5] https://en.wikipedia.org/wiki/Sigmoid_function

[6] https://en.wikipedia.org/wiki/Softmax_function

[7] https://www.ics.uci.edu/~pjsadows/notes.pdf

[8] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting. Journal of machine learning research, 15(1):1929–1958, 2014"

[9] http://www.cs.toronto.edu/~guerzhoy/321/lec/W04/onehot.pdf

[10] https://arxiv.org/pdf/1707.09725.pdf#page=95

[11] https://cookedsashimi.wordpress.com/2017/05/06/an-example-of-backpropagation-in-a-four-layer-neural-network-using-cross-entropy-loss/?frame-nonce=f08dbcf84b

[12] https://arxiv.org/abs/1206.5533

[13] http://ufldl.stanford.edu/wiki/index.php/Gradient_checking_and_advanced_optimization

[14] http://blog.ezyang.com/2011/05/neural-networks/