**PROJECT REPORT**

**CLASSIFICATION OF CIFAR-10 DATASET**

**Introduction:**

A Multilayer Perceptron (MLP) is a class of feed forward Artificial Neural Networks (ANN) and is often used for classification, with each output corresponding to a different binary class (e.g., spam/ham, urgent/not-urgent). It is composed of one input layer, one or more hidden layers and one final layer called the output layer.

In this project we are implementing a multi-layer perceptron (MLP) algorithm that trains using [Backpropagation](http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm) on various different feed forward fully connected neural networks to classify the 10 classes in the CIFAR-10 dataset (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).

**Dataset:**

The CIFAR-10 dataset is a subset of 80 Million small images which consists of 60000 32x32 color images and 10 balanced classes with 6000 thousand images per class. The data set is divided into 50000 sample images considered as training set and 10000 sample images considered as the test set.

The distribution of the training set and test set according to the respective classes can be seen in the figures shown below. The figure depicts the number of images that belong to each class in the respective training and test sets.

Training set:{0: 5000, 1: 5000, 2: 5000, 3: 5000, 4: 5000, 5: 5000, 6: 5000, 7: 5000, 8: 5000, 9: 5000}

Test set: {0: 1014, 1: 1014, 2: 952, 3: 1016, 4: 997, 5: 1025, 6: 980, 7: 977, 8: 1003, 9: 1022}

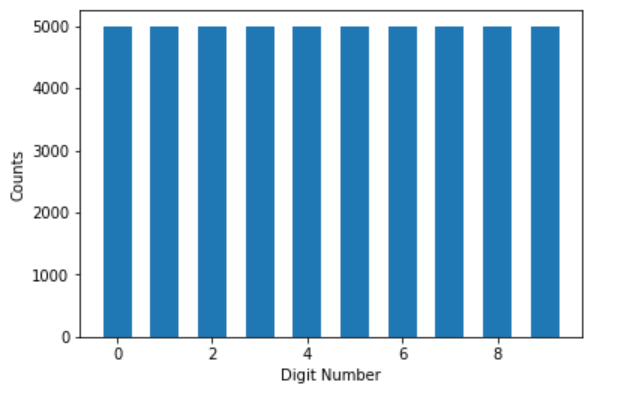
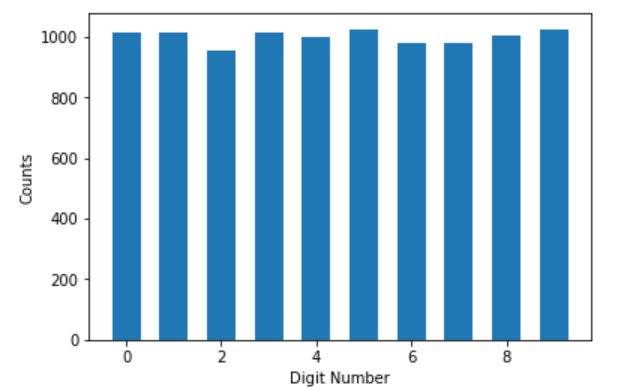
 

Figure.1 Training set distribution Figure.2 Test set distribution

There are 60000 images and each image consists of 3072 features. This is because each image is 32x32x3 pixels, further which are transformed to 3072-dimensional vectors. To reduce the storage size the data is converted from RGB to gray, normalized (to [0-1] range) and rounded to two places.

The shape of training data before and after the image transformation and normalization are as follows:

Before: After:

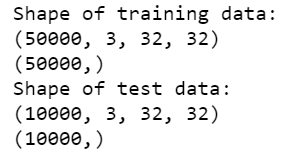
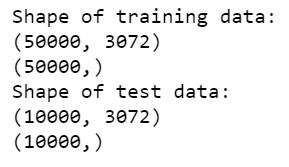
 

Figure3. Training data before normalization Figure4. Training data after normalization

The shape of training data (train\_X) and test data (test\_X) is a 2D tensor (i.e., a matrix) with instances along the first dimension and features along the second dimension, and it is known that the number of features is going to be 32 x 32 x 3 (one feature per pixel). Similarly, the output (train\_Y) is a 1D tensor with one entry per instance.

**Classifier:**

In this dataset to perform the classification task we are using the MLP model as the classifier. Here the MLP model is trained using the backpropogation algorithm, in this for each training instance the algorithm first makes a prediction (forward pass), measures the error, then goes through each layer in reverse to measure the error contribution from each connection (reverse pass), and finally slightly tweaks the connection weights to reduce the error (Gradient Descent step).

To train the MLP here we are taking the help of TensorFlow which is a machine learning library developed at Google. TensorFlow works on the concept of computation graphs where in each mathematical operation is realized as a node in the graph and value from it is then used as input to another node. The value is exported from one node to another in the form of a data type called Tensor. One easy way to train the model is to use either of the high-level and low-level API’s of TensorFlow.

The working of the model is explained with the help of following steps:

1. Retrieving the dataset from the given [link](https://www.cs.toronto.edu/~kriz/cifar.html) and getting it in the desired format ([instances], [features = 32 x 32 x 3]). Since the data was already separated into batches of training data and test data, we directly loaded it to the notebook and did not split into train and test data. We use placeholder nodes to represent the input layer and output layer.
2. The name scope is created using the name of the layer, it contains all the computation nodes for this neuron layer. Variables for the weights (w) and biases (b) are created
3. A deep neural network is created using the neuron\_layer function. The number of hidden layers and the type of activation functions are defined using this function
4. The cost function is defined which is used to train the model. Here in this case we are using cross entropy. It will penalize model if it estimates a low probability for output class.
5. In this step we define optimizer, in this case we are using GradientDescentOptimizer that will modify the model parameters to minimize cost function.
6. To evaluate the model we will simply use accuracy as our performance measure.
7. Once the model is constructed, we are now left with the execution step. We define the number epochs that we want to run, as well as the batch size. The code runs the main training loop: at each epoch, the code iterates through a number of mini batches that corresponds to the training set size. At the end of each epoch, the code evaluates the model on the last mini batch and on the full training set, and prints out the result.

Many variations were incorporated in the model such as change in- number of epochs, activation functions, training set size etc. so as to come up with a good baseline model. In all, five kinds of variations were performed as detailed below with the idea behind them.

1. Varying the number of hidden layers.

If the data is not linearly separable then increasing the number of hidden layers can help in training a network better. Linearly separable data implies that there exists a boundary that divides the sample space into different groups each representing a class. Learning such a decision boundary helps to classify effectively. A neural network with multiple hidden layers can model complex functions using exponentially fewer neurons, making them faster to train.

1. Different kinds of activation functions.

The main reason behind using different activation functions is to analyze what is its effect on prediction and performance metrics. In this project three activation functions have been explored: sigmoid, hyperbolic tangent and ReLU. Each function has different characteristics which make them interesting to explore. Sigmoid function has a well-defined non-zero derivative everywhere, allowing gradient descent to make some progress at every step. Tanh function varies smoothly between [-1, 1] whereas ReLU remains dormant till [0, 0] and then varies linearly.

1. Varying the number of epoch.

One epoch is when an entire dataset is passed forward and backward through the neural network only once. To train the model efficiently one epoch is not enough, we need to pass the full dataset multiple times to the same neural network. Gradient descent is an iterative process, therefore updating weights with single pass or one epoch is not enough. As the number of epochs increases, more number of times the weights are changed in the neural network and the curve goes from **underfitting**to **optimal**to **overfitting**curve.

1. Varying the training set size.

The variation in the training set size has significant effect on the performance of the system. Ideally having more training examples leads to lower train and test error but theoretically more data does not always tend to more accurate model.

1. Different learning rates for the Optimizer.

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect to loss gradient. The lower the value, the slower we travel along the downward slope. If learning rate is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge. The learning rate affects how quickly our model can converge to local minima.

The results and discussions based on the above variations are explained in the next section with the help of different graphs and plots. Note that the error rate and accuracy are represented with respect to either the training set size or the number of neurons in the hidden layer.

**Results and Discussions:**

* Initial Results:

The MLP model was initially trained with the raw unscaled data. The model consists of an input layer, output layer and one hidden layer with 3072 neurons in it. The training batch size is initialized to 450 and the number of iterations (steps) are 1000. The initial accuracy is depicted in the figure below. Later to improve the performance of the model the input data was scaled using feature scaling to obtain a slightly better accuracy as shown in figure below.

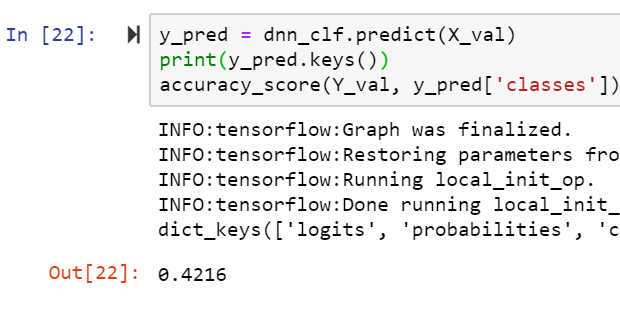
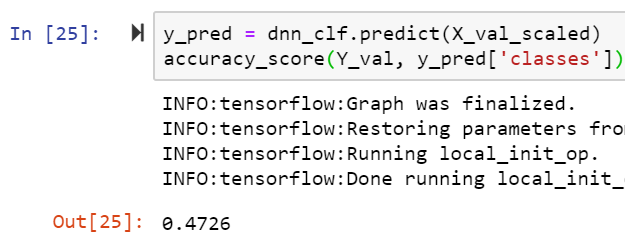
 

Figure5. Accuracy before scaling data Figure6. Accuracy after scaling data

* Changing the number of hidden layers:

In this project we have implemented different variations in the number of hidden layers and hidden neurons. We have considered three different cases, in the first case (h1) there are two hidden layers with 300 neurons in first hidden layer and 100 neurons in second hidden layer, in second case (h2) there are three hidden layer with 300,200 and 100 neurons in hidden layer and finally in the last case (h3) there are of four hidden layers with 400,300,200 and 100 neurons in hidden layers. The comparison of the training error and the validation error between each case is depicted in the figure below. Note that the error rate is plotted against the training set size.

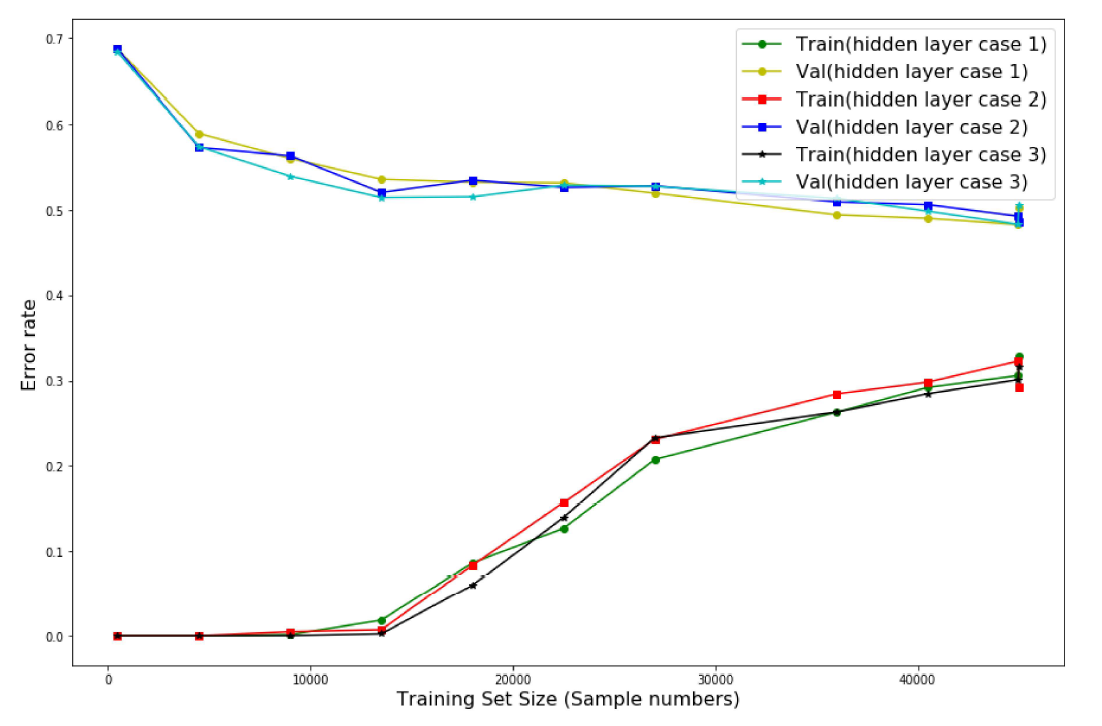


Figure7. Variation in number of hidden layers

It is seen from the figure that the validation error in all three cases is high when the training set size is low and reduces significantly as the training set size increases, whereas in the case of training error it is totally opposite the training error is less when the training set size is low and increases with the increase in the size of training set.

* Different Activation Functions:

In this project we have used three different non-linear functions to train the model. The three functions were ReLU function, sigmoid function and hyperbolic tangent function. Each function has its own characteristics which when applied to the dataset gives unique and improved performance metrics in each case. However, in practice ReLU function works very well and has advantage of being fast to compute.

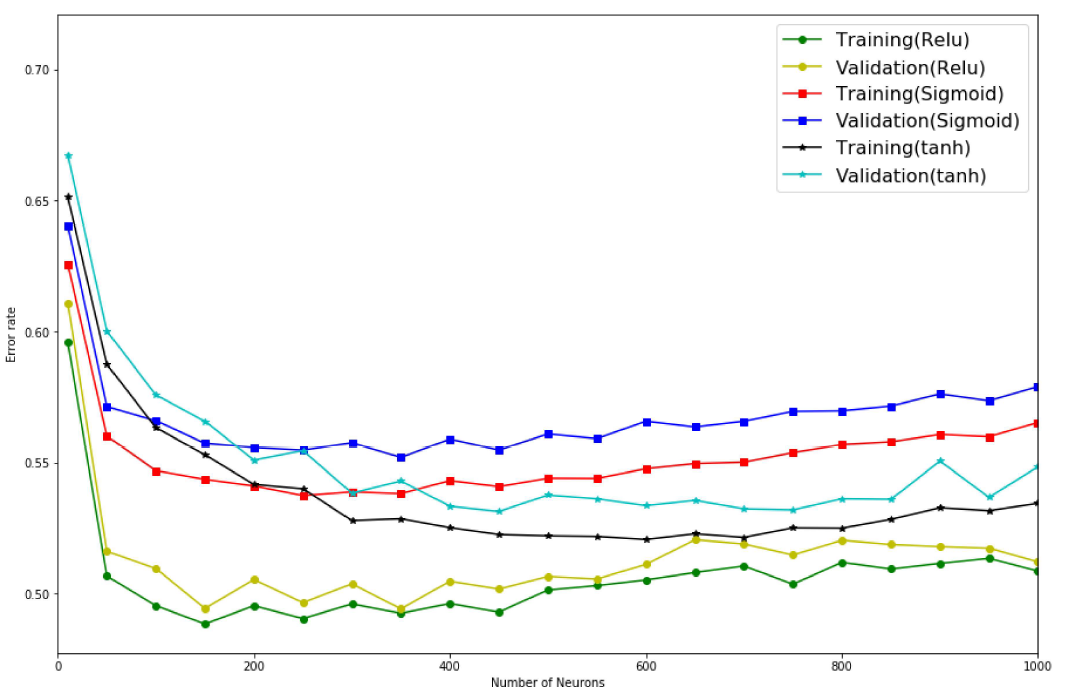


Figure8. Error rate comparison of different activation functions

It is seen in the above shown figure that for every activation function used the error rate is high when the number of neurons in the hidden layer are less and the error rate decreases as the number of neurons are increased in the hidden layer. Considering the plot, we can conclude that the least error rate is achieved while using the ReLU function amongst all the activation functions.

* Change in number of Epochs:

The change in number of Epochs can have a significant change on the performance metrics of the model. Here in this project we are implementing three cases, in the first case we using one regular epoch, in second case the epoch is five times the epoch in first case and in last case the epochs are ten times the epoch in first case. Having more number of epochs while training the data can improve the performance of the model.

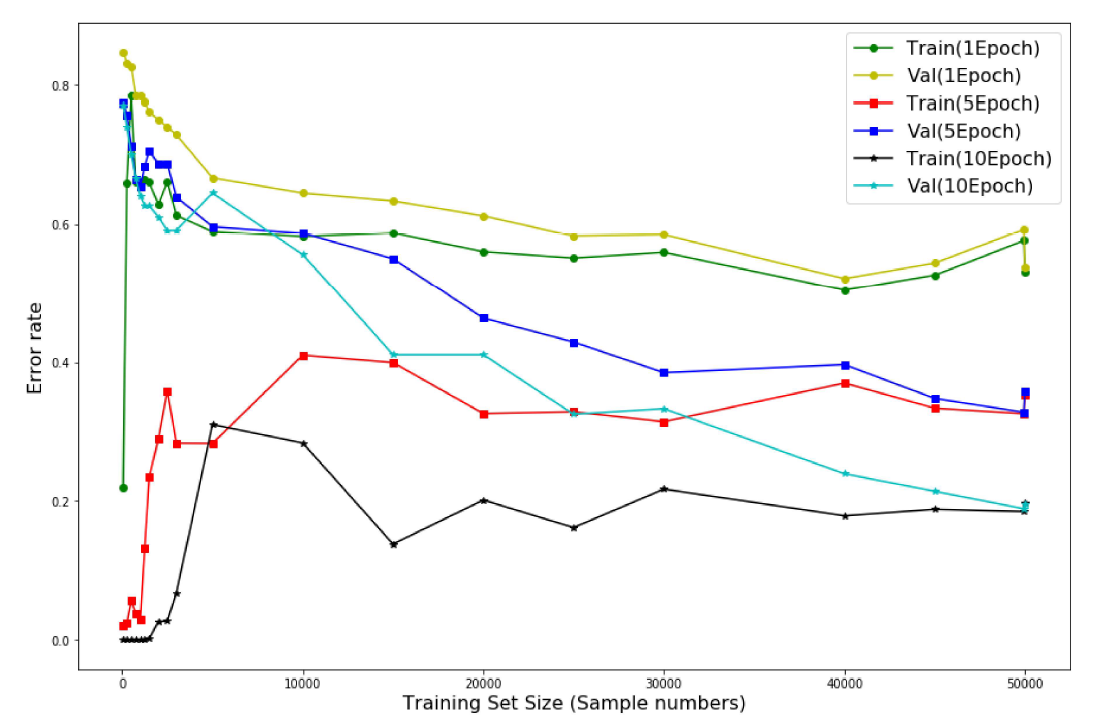


Figure9. Variations in number of epochs

The error rate decreases with increase in the training set size for both training error and validation error. It is observed in the plot shown above that when the number of epochs are less the error rate is high and as the number of epochs are increased to 10 then the error rate reduces. In practical aspects the model needs to be trained multiple times so as to achieve better results.

* Cross-Validation:

Cross-validation is a technique for evaluating ML models by training the respective model on subsets of the available input data and evaluating them on the complementary subset of the data. In this project we are using K-fold cross-validation, with value of K equal to 5 which means that the input data is split into 5 subsets of data (folds).

* Confusion Matrix:

A confusion matrix is a table that is often used to describe the performance of a classification model. The basic idea behind confusion matrix is to count the number of times instances of one class that are classified as other class. In this project the shape of confusion is 10 x 10 since the total number of classes are ten.

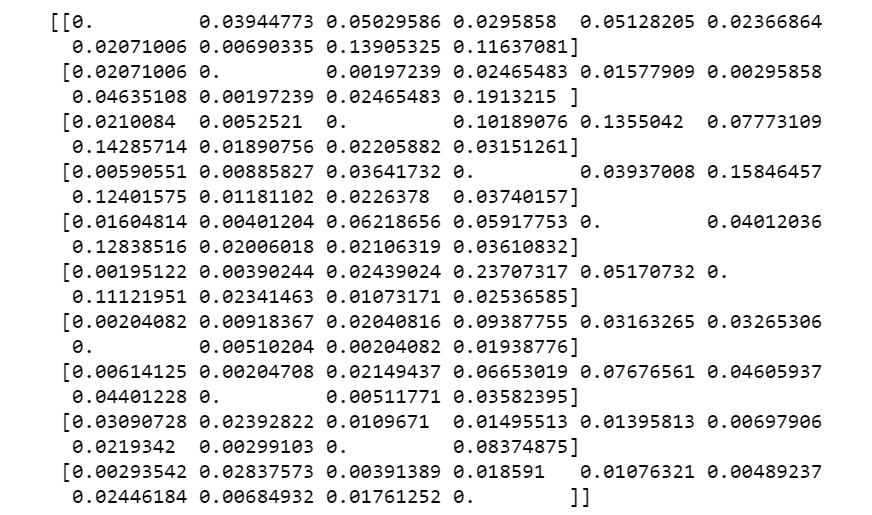


Figure10. Confusion Matrix

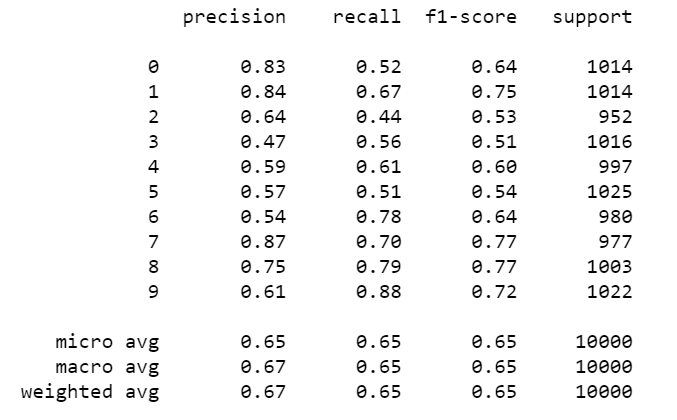


Figure11. Classification Report

The best accuracy for the model came out be 64.7%.