Project 3: Classification

Name	UBIT name	Person No
Anant Gupta	anantram	50249127
Parikshit Deshmukh	pdeshmuk	50247649
Harshdeep Sokhey	hsokhey	50247213

20th November, 2017

-

Highlight work:

- Implemented Back-Propagation from scratch
- Implementing single hidden layer nn using Keras library
- Tested model on both USPS 'Test' and 'Numeral' data
- Tried to implement Bayesian neural network

Problem Statement:

We have to implement the Machine Learning classification concept by using the MNIST digit images and the USPS images. MNIST is a dataset which has 60000 training images in it and 10000 test images. We will train our model using the training images and then test it on the test images.
We are also given USPS images under Test and the Numerals directory. We will perform the testing of our model on all of these images.

Below are the task which we'll be doing ahead:

1. Implement logistic regression, train it on the MNIST digit images and tune hyperparameters

2. Implement single hidden layer neural network, train it on the MNIST digit images and tune hyperparameters such as the number of units in the hidden layer.

3. Use a publicly available convolutional neural network package, train it on the MNIST digit images and tune hyperparameters.

4. Test our MNIST trained models on USPS test data and compare the performance with that of the MNIST data and then check if our finding support the "No Free Lunch" theorem?

Task 1: Logistic Regression:

Here we initially calculate the activation function. Once the activation function is calculated we calculate the output y. This y value helps is knowing the error function for Stochastic Gradient Descent when compared with the expected Target value "t"

We do the performance tuning on learning rate, and start with weights = a vector of 0. after every iteration we update the weights.

These weights are then used in validating the data. we get an accuracy of 92.40

The same weights are used on USPS data and we obtain an accuracy of 0.36

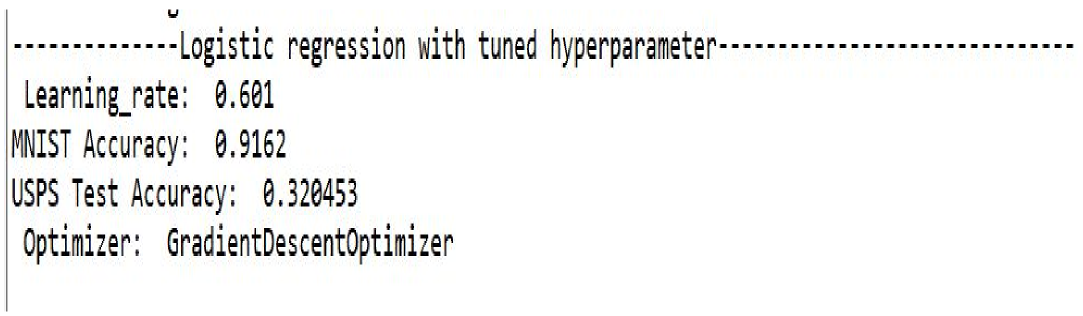
Since the accuracy received here for every model is very low, we can prove the **"No Free Lunch Theorem".** (https://en.wikipedia.org/wiki/No_free_lunch_theorem)

Note: We have also implemented the logistic regression from scratch. We performed the Hypertuning of the parameters. With this we found that Learning rate of 0.601 gives the best accuracy.

Below table shows the values of our analysis;

Learning Rate	**MNIST Accuracy**	**USPS Accuracy (Test)**	**USPS Accuracy (Numerals)**
0.601	91.6%	32.1%	35.3%

Below is the output:



```
-------------Logistic regression with tuned hyperparameter----------------------------
 Learning_rate:  0.601
MNIST Accuracy:  0.9162
USPS Test Accuracy:  0.320453
 Optimizer:  GradientDescentOptimizer
```

Task 2: Single Layer Neural Network:

A) Using TensorFlow library:

- Here we have used tensorflow open source library to implement the neural network with single hidden layer.
- The training dataset is the MNIST data taken from tensorflow library. It contains the training and testing data in separate chunks. First we have trained our model on train data and then used the model to test the testing accuracy. Further we have used our trained model to test on USPS data.
- The single hidden layer nn has the input layer which is 784 (28*28), output layer which is 10 (number of possible classes; digits) and Hidden layer
- Hidden layer in our case has been chosen starting from 500 to 1000. We have tuned it and found that the maximum accuracy is given for 1000 nodes for hidden layer.
- We have also tuned the model for below params:
    - Hidden Layer nodes, Learning Rate, Optimizer
- We have explicitly created grid pattern for tuning using loop for our hyper-parameters.
- We have also tried three different activation functions- Relu, Sigmoid and Tanh for the hidden layer and obtained that the accuracy for Sigmoid function is maximum. So we have finally trained our model using Sigmoid as the hidden layer activation fuction.
- Since it is a multi class classification problem, we have used Softmax at the output layer to get eh probability distribution for the input over 10 classes.
- For the Backward propagation, we first find loss using the cross entropy formula. Then the output of the cross entropy is reduced using the tf.reduce.mean function.
- This output is then back propagated to minimise the error using different optimizers. We have tried GradientDescent and Adam Optimizer and found that Adam optimizer gives good results.
- Then the accuracy is calculated for MNIST test data and USPS dataset.
- Below are the brief results for this part:

HyperParams	**Range**	**Final tuned for max accuracy**
Hidden layer nodes	500-1000	1000
Learning rate	0.001 - 1	0.001

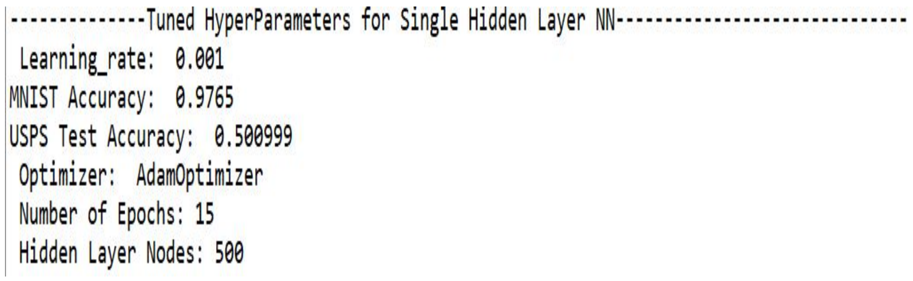
Optimizers	[GradientDescent, Adam]	AdamOptimizer
Batch size	-	20
Epoch	-	15
	MNIST	**USPS**
Test Accuracy	97.7%	50.1%

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}$$

$$y_k = \frac{\exp\left(a_k\right)}{\sum_j \exp\left(a_j\right)}$$

Below is the output:



```
-------------Tuned HyperParameters for Single Hidden Layer NN----------------------------
 Learning_rate:  0.001
MNIST Accuracy:  0.9765
USPS Test Accuracy:  0.500999
 Optimizer:  AdamOptimizer
 Number of Epochs: 15
 Hidden Layer Nodes: 500
```

b) Using BackPropagation from scratch: (Using: Sklearn, Numpy)

- In order to implement Back propagation from scratch we have used Sklearn for loading MNIST data and Numpy for mathematical computations
- We have split the data into train and testing set in ratio 70:30
- Due to processor limitations we have kept hidden layer nodes as 500 and we haven't tested it for usps data due to time limit and technical limitations, but we have implemented it successfully
- The input is multiplied with randomly initialized weights W1 and a bias b1 is added in the out of first layer. a1.
- For the hidden layer we have used Tanh as the activation function which takes the input as the output of the first layer a1. This gives z1.
- This is then multiplied with the W2 and a bias b2 is added which is then sent to the final output layer
- At the output layer we have wrote a Softmax function that that gives us the probability distribution over number of classes.
- This output is then used to calculate the loss using Cross Entropy algorithm written as a independent function that also takes One hot vector of the actual output Y.
- For back-propagation we have implemented below chain of algorithm and then we have adjusted ur initially defined weights so that we can have minimum loss. It is typical gradient descent algorithm

$$\delta_k = y_k - t_k$$

$$\delta_j = h'(z_j) \sum_{k=1}^{K} w_{kj} \delta_k$$

The gradient of the error function would be

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Having the gradients, we will be able to use stochastic gradient descent to train the neural network.

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \nabla_{\mathbf{w}} E(\mathbf{x})$$

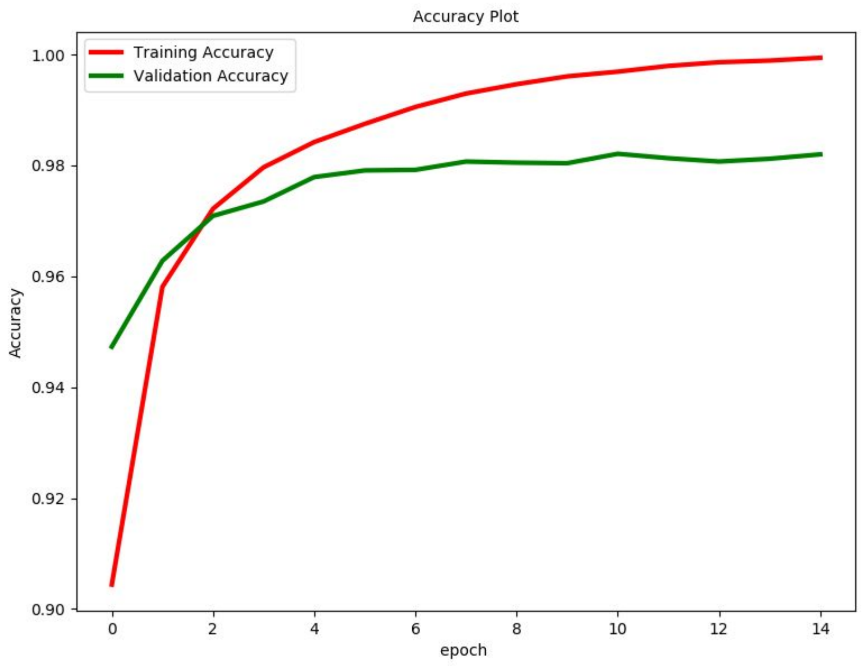
- Here we have also used the regularizer lambda to avoid the Over-fitting.

- Using the SGD with batch size of 30 and epoch of 15 we get the final weights which we then used to test the accuracy of the model on the MNIST test data.

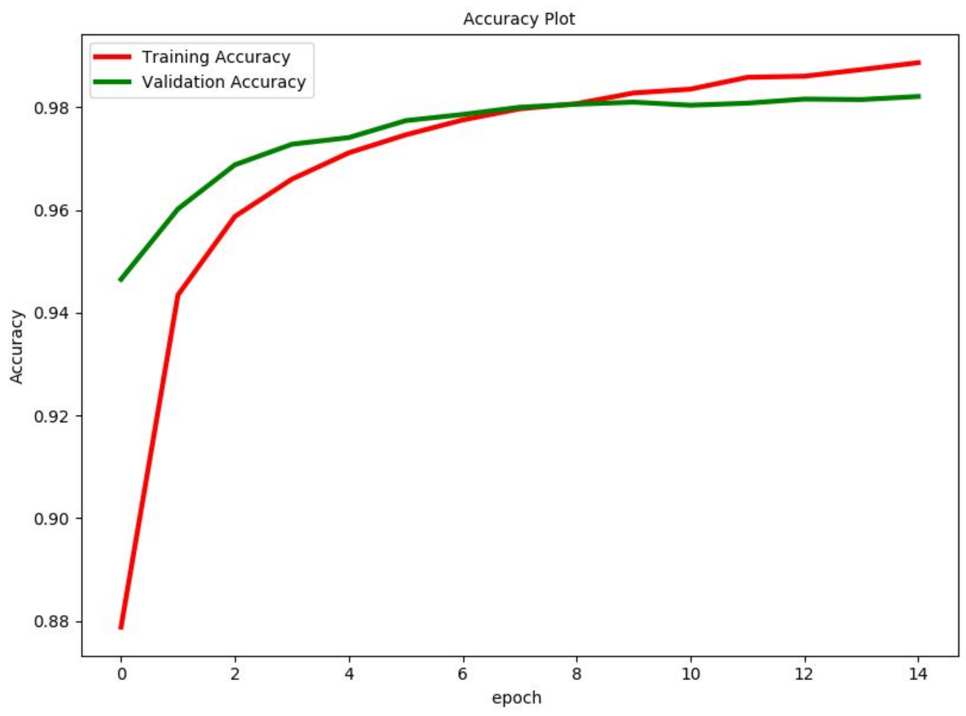
c) Using Keras library:

- Here we are going to use one of the famous open source library Keras to implement the Single hidden layer NN.
- Keras has it's advantage as it intuitively selects the library to use from Tensoflow and Theanos.
- We first learnt this library and then came up with the code to implement single layer nn.
- The code is not very lengthy which saves the time and effort and increases the readability for users.
- One more advantage of this lib is that it becomes easy to add extra layers while designing neural network with multiple layer using Keras' sequential layering feature.
- We have submitted this code separately along with the Back propagation from scratch.

Without adding regularization:



After adding Dropout regularization:



So after adding the regularization we can see that the difference between trainings and Test accuracy is reduced substantially.
Test Accuracy Achieved: 98%

Task 3: Convolutional Neural Network:

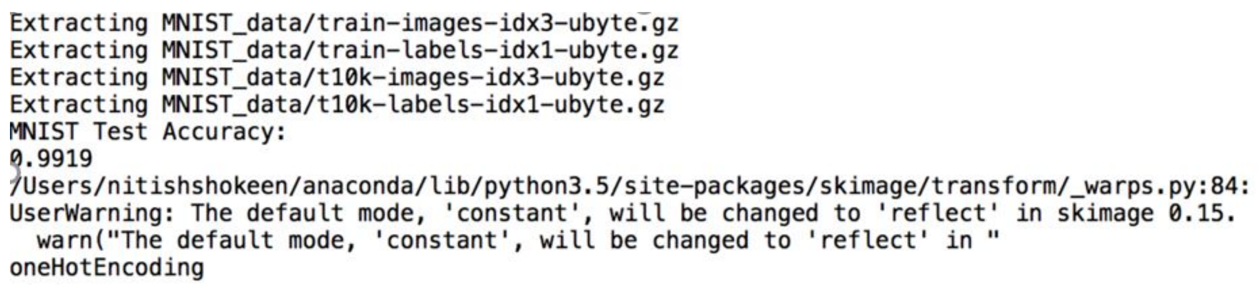
- In this assignment we have used TensorFlow library to utilize the CNN functions and formulas
- We use CNN because of the undeperformance of normal neural network.
- The basic pattern followed i a CNN is as below:

- Convolution Layer-> Activation (relu) -> Pooling ->Regulazier -> Fully selected layer

- In here we have used Max pooling, relu as activation function, Dropout regularization technique, cross entropy for finding the loss and Adam optimizer for minimizing it.
- In our code we have implmented CNN using two convolutional layer incuding activation(relu) and max pooling in both of them, one densly connected layer, Dropout function for regularizing to rpevent over-fitting and a final fully connected layer.
- The output of the final layer is used to calculate the loss by comparing with the actual outout.
- We are using softmaxEntropy function of tensorflow to get this. And then we use AdamOptimizer to reduce the minimize the loss.
- We get the accuracy of around 99% using this method.
- We then use the model to predict the USPS data and the accuracy of its prediction.

	MNIST	USPS
Test Accuracy	99.2%	64.9%

Below is the output of our script:



```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
MNIST Test Accuracy:
0.9919
/Users/nitishshokeen/anaconda/lib/python3.5/site-packages/skimage/transform/_warps.py:84:
UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
oneHotEncoding
```

Task 4 USPS Data:

Here we use the same weights as that received from training on the MNIST data. We perform the below steps:

1) Read the image using the imread() method from imageio package

2) Converted the image to gray-scale
3) Resize the image and normalize it such that the final image should be of size 28*28
4) Perform flattening on the image obtained above and then vectorize this.
5) Pass the final array to logistic regression and neural network model we made earlier.

Since the accuracy received here for every model is very low, we can prove the "No Free Lunch Theorem".