EE6132 Advanced Topics in Signal Processing - Assignment No 1

Name: Atishay Ganesh
Roll Number: EE17B155

September 2, 2019

1 Abstract

The goal of this assignment is the following.

- To experiment with Multilayer Perceptrons with Backpropogation learning
- To experiment with various non-linear activation functions.
- To explore regularization and the effects of adding noise.
- To study the performance of MLP using off the shelf features.

2 Assignment

2.1 Part 0

Implementation Details: We use Python 3.5 for implementing the MLP. We use numpy for vectors and matrices, pyplot for plotting the graphs and sklearn for the accuracy metrics and We create a seperate module called activations for the non-linear functions as well as loading the data. The data is sourced from an external source [1] since that source has it in CSV form.

Importing the standard libraries

```
import numpy as np
from sklearn.metrics import log_loss, accuracy_score, confusion_matrix,f1_score, pre
import matplotlib.pyplot as plt
from activations import *
import cv2
from sklearn.svm import SVC
import sklearn.neighbors
```

2.2 Training an MLP

For reusability of code, we combine parts 1, 2, and 3 into one major block.
For brevity I will only touch upon the major points of the code.
Forward Pass function:

```
def forward_pass(self,input):
    if not self.gaussian_fws:
        return self.activation_fn.fn(input@self.weights + self.biases)
    else:
        noise = np.random.normal(0,0.01,self.biases.shape)
        return self.activation_fn.fn(input@self.weights + self.biases)+ noise
```

Backward Pass Function:

```
def backward_pass(self,activations,pred_outputs,targets):
    error_final = pred_outputs - np.squeeze(targets)
    if self.gaussian_reg:
        activations = activations+ np.random.normal(0,0.01,activations.shape)
    db = np.expand_dims(np.sum(error_final,axis=0),0)/error_final.shape[0]
    dw = np.transpose(activations)@error_final/error_final.shape[0]
    inact_w = (np.sum(np.where(np.absolute(dw)<10**-5,1,0))+np.sum(np.where(np.abso
    sum_w = np.size(dw)+np.size(db)
    self.weights = self.weights*(1-self.l2*self.lr)
    self.biases = self.biases*(1-self.l2*self.lr)
    self.weights = self.weights-self.lr*dw
    self.biases = self.biases -self.lr*db
    return (error_final@np.transpose(self.weights)),inact_w,sum_w
```

The code for the backward pass of the a non - output layer is very similar, except for the line where we calculate the error which is as follows.

```
errors =  errors*self.activation_fn.der(post_activations)
```

2.2.1 Sigmoid Activation

We train it with Sigmoid Non-linearity for 15 epochs. The results of the same are shown. We also show the percentage of inactive neurons (as required in Part 2 Question 2).

Precision: 0.9566, Recall: 0.9567, F1 score: 0.9566

Confusion Matrix:
$[966, 0, 1, 0, 0, 4, 4, 3, 2, 0]$
$[0, 1118, 3, 2, 0, 2, 2, 1, 7, 0]$
$[7, 2, 983, 10, 5, 1, 5, 7, 9, 3]$
$[0, 1, 4, 969, 0, 14, 0, 12, 6, 4]$
$[1, 1, 5, 0, 934, 2, 7, 2, 4, 26]$
$[5, 1, 0, 7, 2, 861, 7, 1, 2, 6]$
$9, 3, 1, 0, 7, 20, 913, 0, 4, 1]$
$[1, 13, 16, 6, 3, 1, 0, 959, 1, 28]$
$[5, 1, 1, 15, 7, 15, 8, 5, 912, 5]$
$[7, 7, 1, 7, 15, 6, 1, 6, 4, 955]$

Common Errors are between 5 and 6, 7 and 9 and 4 and 9, possibly due to similarities in the way they are written (especially 5,6 and 4,9). 1 is probably the most visually distinct digit and hence it is identified correctly the most. Variation with learning rate was tested, and it was observed that the network converged faster with higher learning rates.

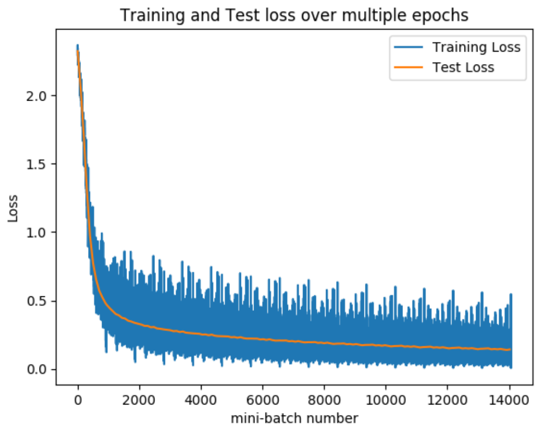

Figure 1: Sigmoid Activation

2.3 Activation Function

We now try out different activation functions including Rectified Linear Unit (ReLU) and the hyperbolic Tangent (Tanh) For brevity we do not show the confusion matrix in the report, but it can be found the stats text file which is created by the code after training finishes.

2.3.1 ReLU Activation

Precision: 0.9568, Recall: 0.9570, F1 score: 0.9568

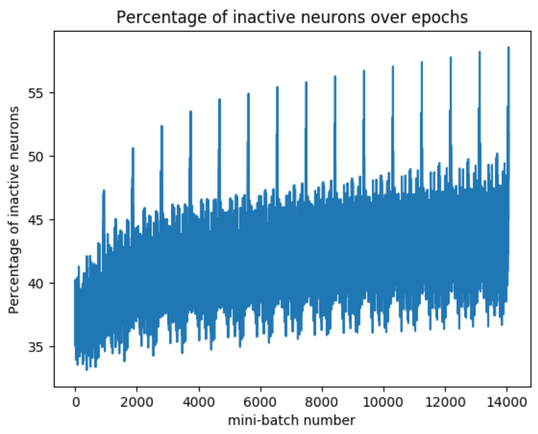

Figure 2: Sigmoid Activation

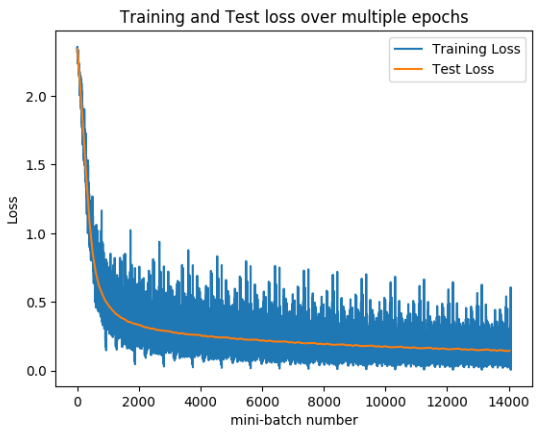

Figure 3: ReLU Activation

2.3.2 Tanh Activation

Precision: 0.9565, Recall: 0.9567, F1 score: 0.9564

2.3.3 Analysis

It appears that ReLU outperforms the other two activations, but the difference is so small it is probably not significant. The number of inactive neurons also hovers around 40-50% for all of the activations.

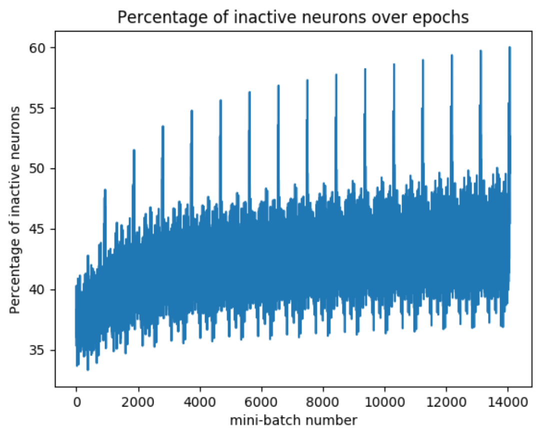

Figure 4: ReLU Activation

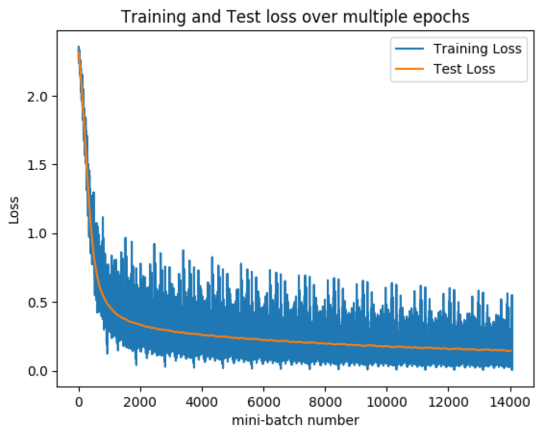

Figure 5: Tanh Activation

2.4 Part 3, Regularization

We now experiment by adding gaussian noise to the forward and backward passes. We also try L_2-Regularization. We also try adding Gaussian Noise as as a method of data augmentation. For brevity's sake we do not mention the confusion matrices here, but those are output in the stats file under descriptive names.

2.4.1 Gaussian Noise in Forward pass

Precision: 0.9453, Recall: 0.9456, F1 score: 0.9452

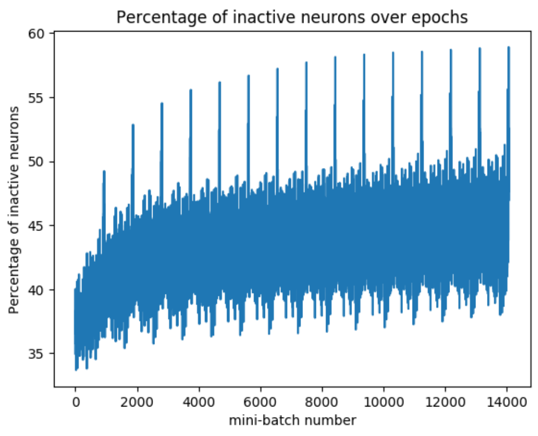

Figure 6: Tanh Activation

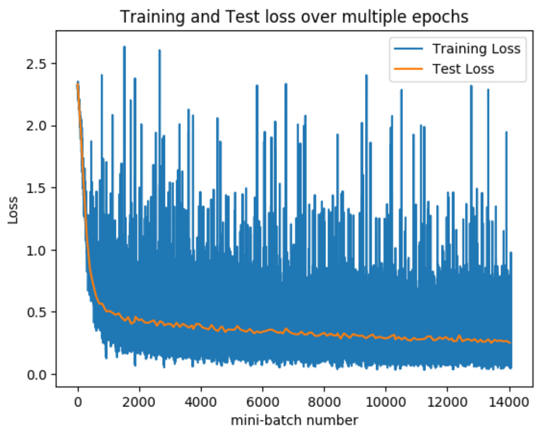

Figure 7: Forward Noise

2.4.2 Gaussian Noise in Backward pass

Precision: 0.9553, Recall: 0.9554, F1 score: 0.9553

2.4.3 Input Noise

Precision: 0.9552, Recall: 0.9556, F1 score: 0.9552

2.4.4 Weight Decay

Precision: 0.9560, Recall: 0.9560, F1 score: 0.9559

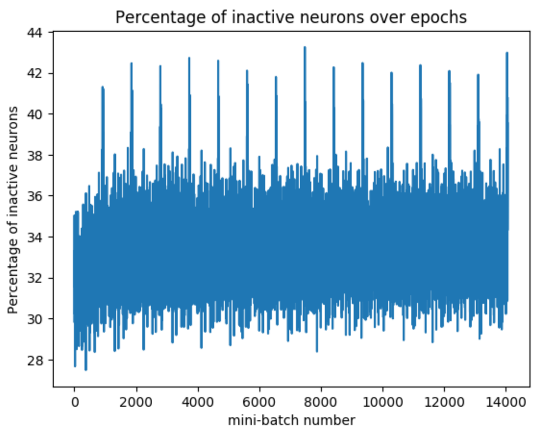

Figure 8: Forward Noise

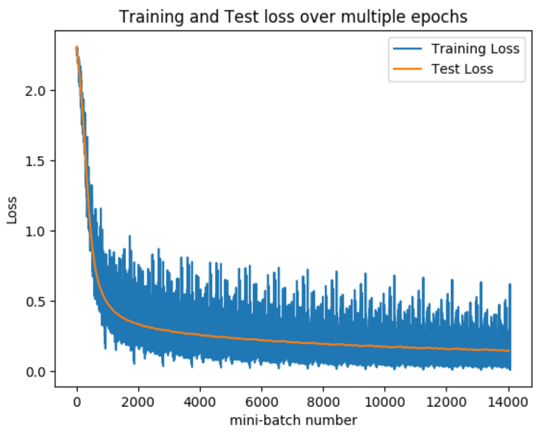

Figure 9: BP Noise

2.4.5 Analysis

Adding Forward Pass Noise causes the training to fluctuate quite a lot. While overall it learns well, there are a lot of cases in which the loss suddenly spikes up a lot. However this does also mean that the number of inactive neurons is lesser, as it is always learning something.

Adding Noise in BP does not cause any such issues, and consequentially, it performs better, with better metrics over the board.

Adding input noise also causes the network to have to learn more number of features and better, and hence it causes the percentage of inactive neurons to fall a lot.

Weight decay is useful to reduce the weights of the network, and also performs pretty well.

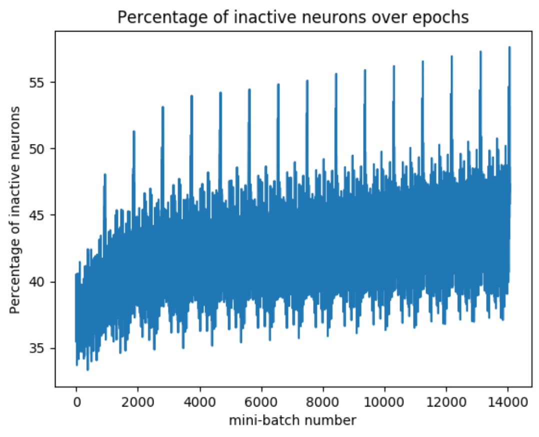

Figure 10: BP Noise

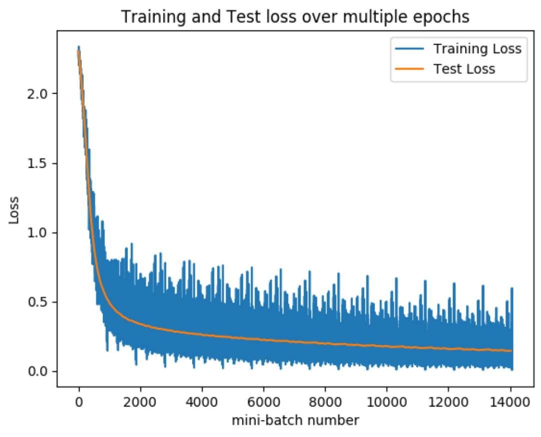

Figure 11: Input Noise

2.5 Hand Crafted Features

We now use the hand crafted features known as the histogram of gradients (HOG). We obtain a set of 324 features from the image and use various methods including MLP, KNN, and SVM to classify these images.

```
winSize = (28,28)
blockSize = (8,8)
cellSize = blockSize
nbins = 9
blockStride = (4,4)
derivAperture = 1
winSigma = -1.
histogramNormType = 0
```

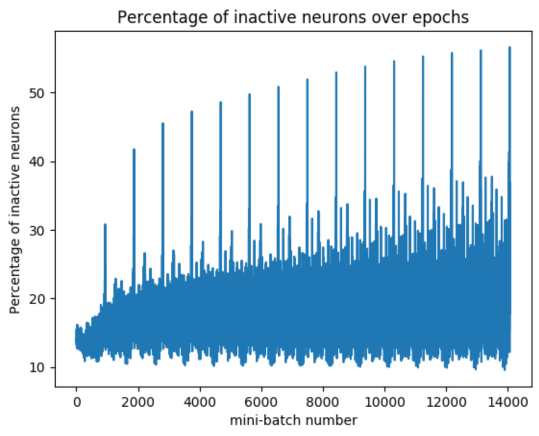

Figure 12: Input Noise

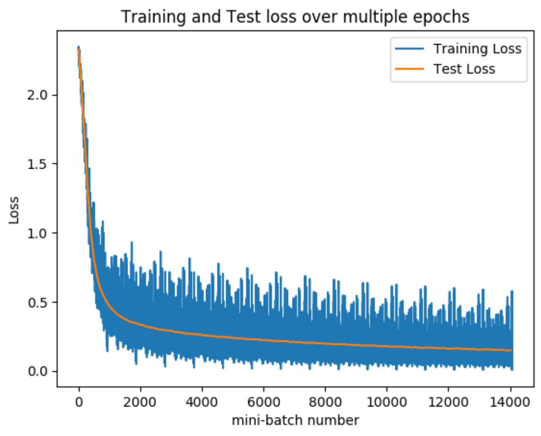

Figure 13: Weight Decay

```
    L2HysThreshold = 2.0000000000000001e-01
    gammaCorrection = 1
    nlevels = 64
    hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivApertu
                            histogramNormType,L2HysThreshold,gammaCorrection,nlevels
    return hog.compute(img)
```

2.5.1 K-Nearest Neighbor

We use a K - Nearest Neighbor based classifier to classify these HOG features. We set the number of neighbors as 5.
Accuracy Score: 0.975

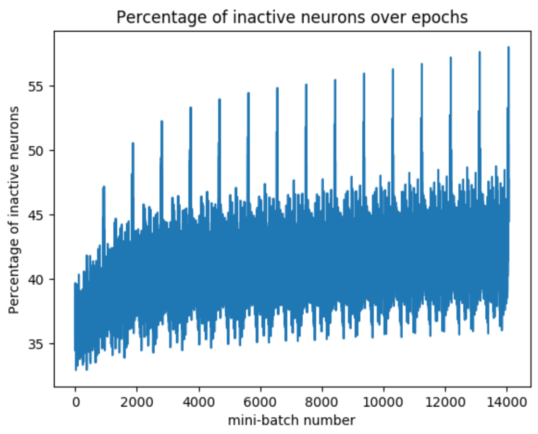

Figure 14: Weight Decay

2.5.2 Support Vector Machine

We use a SVM based classifier to classify these HOG features. We use the Radial Basis Function Kernel.
Accuracy Score: 0.9812

2.5.3 Multilayer Perceptron

We Use a Neural Network with an Input Dimension of 324, A hidden layer of size 32, another hidden layer of size 16 and an output layer of size 10. We use ReLU with Learning Rate $5e-3$.
Precision: 0.9266, Recall: 0.9265, F1 score: 0.9264

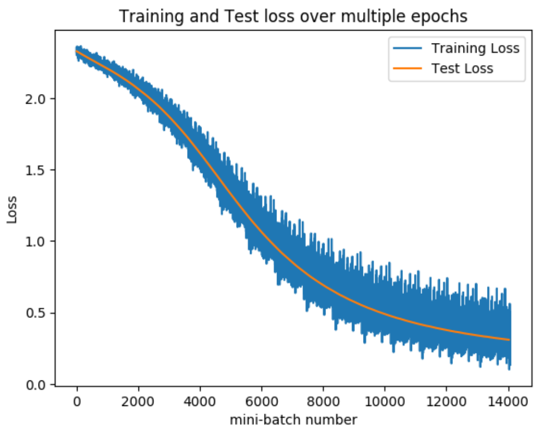

Figure 15: HOG Gradients + MLP

2.5.4 Analysis

Clearly we can see that the HOG features do better at the task of MNIST classification than a vanilla Neural Network. The reason for this is probably that MNIST images are very small, and that Deep Learning is generally suited for higher dimensional tasks. An SVM or KNN using carefully crafted features can do better than an MLP on this dataset but will really struggle on ImageNET or MS-COCO dataset.

3 Conclusions

- We experimented with various MLP models with BP.
- We used regularisation
- To explore regularization and the effects of adding noise.
- To study the performance of MLP using off the shelf features.

References

[1] D. Dato-on, “Mnist in csv,” May 2018.