EE6132: Advanced Topics in Signal Processing

Assignment # 1
MNIST Classification using Multilayer Neural Networks

Sourav Sahoo, EE17B040

September 1, 2019

Contents

1 Introduction

Deep learning methods have radically improved the state-of-the-art models in numerous tasks across various domains like image classification, segmentation, machine translation, speech recognition etc. Deep learning identifies the intricate patterns in very large datasets by backpropagation algorithm due to which it has become a very successful idea in recent times [1]. In this assignment, we apply a multilayered perceptron (MLP) to classify the MNIST handwritten digit dataset [2].

2 Dataset

The MNIST handwritten digit dataset contains 60K data points as training set and 10K data points in the test set. The data is present in `idx3-ubyte` format which is converted into `numpy` arrays. The images are normalized and flattened (i.e. the shape changes from 28×28 to 784×1) and the labels are converted into one-hot vectors.

3 Baseline Model

We implement a baseline model without using any existing deep learning frameworks. The baseline model consists of three fully connected hidden layers containing 500, 250 and 100 neurons. The input to the neural network are flattened 28×28 images in mini batches of 64. We use sigmoid activation function in all layers except the final layer where we use linear activation. The outputs of the final layer are passed through a softmax layer that finally predicts the class of each input. The weight matrices is initialized using Glorot initialization [3] and biases are initialized to zero. The learning rate is found by doing a grid search over a logarithmic scale from 1e-6 to 10. For training, we use mini batch gradient descent in mini batch sizes of 64 with learning rate 0.05 for 15 epochs.

3.1 Baseline Training

The baseline model using sigmoid activation is trained five times and the average training and test loss is shown in Figure 1. It is to be noted that the noise in training loss is quite high as compared to the test loss which converges very smoothly.

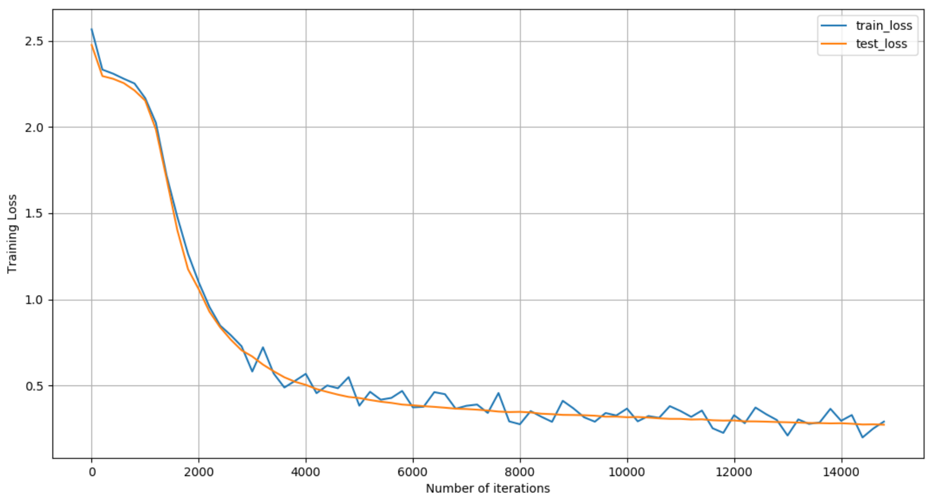

Figure 1: Variation of training loss and testing loss with number of iterations

3.2 Varying Learning Rates

We try to train the model by varying the learning rates on a logarithmic scale from 0.0001 to 10 and observe the variation of the training loss. In Figure 2, we observe that for very small and very high learning rates the model fails to converge, as anticipated. For learning rates 0.1 and 1.0, the training loss converges sufficiently.

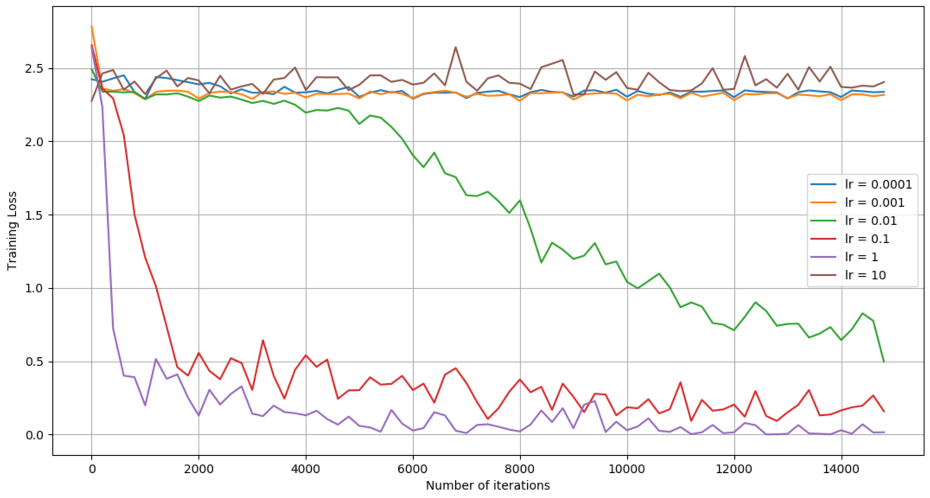

Figure 2: Variation of training loss as a function of learning rate

3.3 Results and Discussion I

In this section, we report the various accuracy metrics calculated and analyze the results. The baseline model is trained for five times and the average accuracy and standard deviation is reported. The confusion matrix is presented in Table 1 for the test dataset. We also calculate the precision, recall and f1-scores in Table 2,3 and 4.

$$\text{Average Accuracy} = 0.9212$$
$$\text{Standard Deviation} = 0.0014$$

Table 1: Confusion Matrix of the baseline model

	0	1	2	3	4	5	6	7	8	9
0	**956**	0	5	1	1	7	6	2	2	0
1	0	**1108**	2	2	1	3	4	1	12	0
2	13	8	**933**	6	9	3	17	9	30	3
3	2	1	25	**900**	0	40	1	12	20	6
4	3	2	6	1	**907**	0	12	1	4	42
5	10	3	4	33	6	**778**	13	5	31	8
6	12	3	6	0	9	16	**905**	0	6	0
7	4	14	24	3	6	1	0	**940**	2	33
8	4	10	7	15	10	30	14	3	**867**	13
9	12	10	1	8	30	8	0	11	4	**923**

The vertical axis is the true class and the horizontal axis is the predicted class. The values in bold represent the true positives for each class.

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{1}$$

Table 2: Precision of the baseline model with sigmoid activation function

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Prec.	0.9409	0.9559	0.9210	0.9288	0.9265	0.8781	0.9311	0.9553	0.8865	0.8979

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{2}$$

Table 3: Recall of the baseline model with sigmoid activation function

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Recall	0.9755	0.9779	0.9049	0.8937	0.9274	0.8732	0.9456	0.9153	0.8911	0.9166

$$\text{f1-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

Table 4: F1-Score of the baseline model with sigmoid activation function

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
F1	0.9579	0.9668	0.9129	0.9109	0.9269	0.8756	0.9383	0.9349	0.8888	0.9071

From the confusion matrix in Table 1 and F1 scores in Table 4, we observe that the model often **confuses among classes 3,5 and 8**. The F1-scores of the above mentioned classes are quite low as compared to the other classes. This can be explained by the fact that the digits 3, 5 and 8 have very similar appearance. Digits that have very distinct style of writing such as **1** have quite high F1 score (0.9668) as compared to other classes. This shows that the model actually learns to recognize the digits instead of memorization.

4 Activation Functions

In this section, we experiment with two activation function different from sigmoid: 1) tanh and 2) ReLU.

4.1 Tanh

Mathematically, the expression for tanh function is given as,

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

The baseline model with tanh activation function is trained five times and average training and test losses are reported in Figure 4. It can be seen that the loss (training as well as testing) converges to a smaller value as compared to sigmoid activation in Figure 1. The accuracy metrics are discussed in Section 4.3.

Average Accuracy = 0.9784
Standard Deviation = 0.0005

4.2 ReLU

Rectified linear units (ReLU) [4] makes up for the vanishing gradients problem that is present in sigmoid and tanh activation functions. It is also computationally less expensive to calculate as compared sigmoid or tanh. Mathematically, ReLU can be described as,

$$ReLU(x) = max(0, x) \tag{5}$$

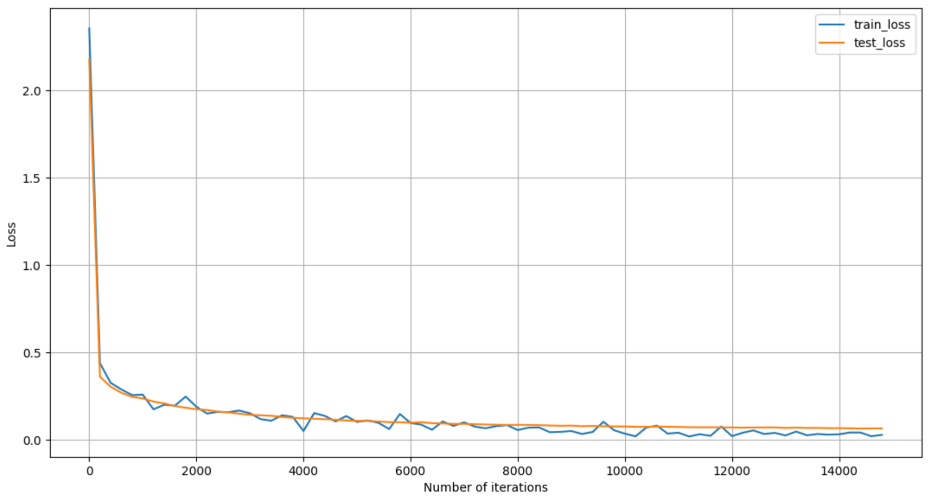

Figure 3: Variation of training loss and testing loss with number of iterations for tanh activation

Table 5: Confusion Matrix of the model with tanh activation function

	0	1	2	3	4	5	6	7	8	9
0	**971**	0	2	1	0	1	1	1	1	0
1	0	**1121**	2	3	0	1	2	1	2	0
2	2	0	**1016**	4	1	0	2	3	2	0
3	0	0	3	**992**	0	0	0	3	3	6
4	0	0	4	1	**965**	1	2	1	1	6
5	3	1	0	16	2	**856**	5	1	4	2
6	6	3	4	1	3	5	**934**	0	1	0
7	2	2	11	4	0	0	0	**992**	2	14
8	3	0	5	11	4	5	2	3	**939**	2
9	4	2	2	11	11	2	0	3	0	**973**

The vertical axis is the true class and the horizontal axis is the predicted class. The values in bold represent the true positives for each class.

The baseline model with ReLU activation function is trained five times and average training and test losses are reported in Figure 4. It can be seen that the loss (training as well as testing) converges to a smaller value compared to both sigmoid and tanh activation. The accuracy metrics are discussed in Section 4.3.

$$\text{Average Accuracy} = 0.9805$$
$$\text{Standard Deviation} = 0.0008$$

4.3 Results and Discussion II

In this section, we compare the baseline model and the models with different activation functions on different parameters like train and test loss and accuracy metrics.

ReLU is immune to the vanishing gradient problem which is evident in both tanh and sigmoid activation function. This becomes more prominent as the neural network becomes deeper. So, as we can see in Table 7, the training as well as testing loss in case of ReLU is much smaller than sigmoid or tanh. The reduction is loss gives rise to the improved test accuracy.

The precision, recall and F1-scores of the three models are compared. As seen in Table 8, 9 and 10, the models using tanh and ReLU activation function outperform the baseline model in each

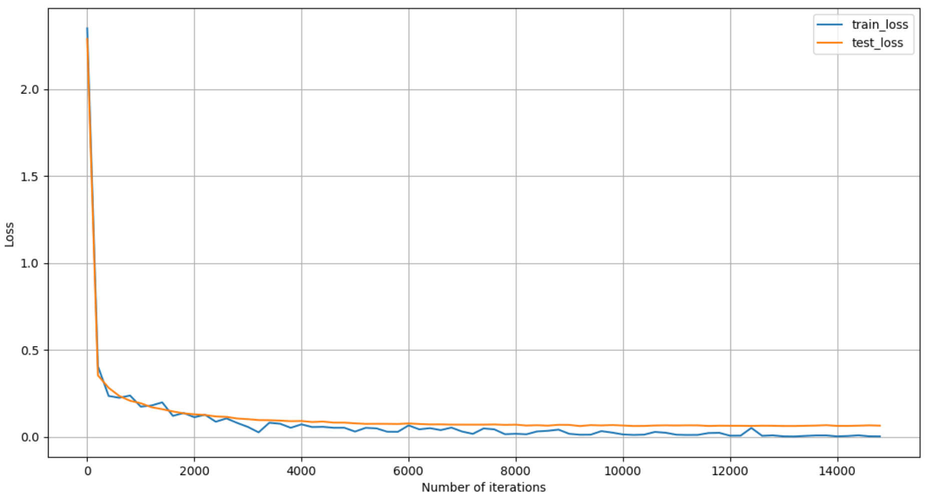

Figure 4: Variation of training loss and testing loss with number of iterations for ReLU activation

Table 6: Confusion Matrix of the model with ReLU activation function

	0	1	2	3	4	5	6	7	8	9
0	**968**	0	2	1	0	1	2	1	3	1
1	0	**1124**	3	1	0	0	0	1	5	0
2	2	2	**1004**	6	3	0	3	4	4	0
3	1	0	4	**995**	0	2	0	3	3	1
4	0	0	4	0	**964**	0	1	3	1	8
5	2	0	0	11	1	**867**	3	0	3	3
6	3	3	3	1	8	4	**933**	0	3	0
7	1	5	4	3	1	0	0	**1008**	2	3
8	2	1	5	6	3	5	2	2	**940**	5
9	2	2	0	8	7	1	0	4	3	**980**

The vertical axis is the true class and the horizontal axis is the predicted class. The values in bold represent the true positives for each class.

Table 7: Comparison of the baseline model with models with different activation functions

	Baseline	**Tanh**	**ReLU**
Training Loss	0.3089	0.0292	**0.0037**
Testing Loss	0.2740	0.0680	**0.0653**
Test Accuracy	0.9212	0.9784	**0.9805**
Standard deviation	0.0014	**0.0005**	0.0008

The values are calculated after training and testing each model for five times. The values in bold represent the model with best performance for that particular parametric.

class. The models with the different activation functions gain significant increase in F1-scores for classes 3 (**0.9750** in ReLU model compared to 0.9109), 5 (**0.9797** in ReLU model compared to 0.8756) and 8 (**0.9736** in tanh model compared to 0.8888) which performed rather poorly in the baseline model.

Table 8: Precision of the three models

	0	1	2	3	4	5	6	7	8	9
M1	0.9409	0.9559	0.9210	0.9288	0.9265	0.8781	0.9311	0.9553	0.8865	0.8979
M2	0.9798	0.9929	0.9685	0.9502	0.9787	0.9828	0.9852	0.9841	0.9832	0.9701
M3	0.9867	0.9886	0.9757	0.9641	0.9767	0.9852	0.9883	0.9824	0.9721	0.9790

where M1 = Baseline Model, M2 = Model with tanh activation and M3 = Model with ReLU activation. The shaded cells shows the function that show best performance in that class.

Table 9: Recall of the three models

	0	1	2	3	4	5	6	7	8	9
M1	0.9755	0.9779	0.9049	0.8937	0.9274	0.8732	0.9456	0.9153	0.8911	0.9166
M2	0.9928	0.9903	0.9864	0.9851	0.9837	0.9618	0.9760	0.9659	0.9641	0.9653
M3	0.9888	0.9912	0.9767	0.9861	0.9827	0.9742	0.9739	0.9815	0.9681	0.9732

where M1 = Baseline Model, M2 = Model with tanh activation and M3 = Model with ReLU activation. The shaded cells shows the function that show best performance in that class.

Table 10: F1-scores of the three models

	0	1	2	3	4	5	6	7	8	9
M1	0.9579	0.9668	0.9129	0.9109	0.9269	0.8756	0.9383	0.9349	0.8888	0.9071
M2	0.9863	0.9916	0.9774	0.9673	0.9812	0.9722	0.9806	0.9749	0.9736	0.9677
M3	0.9878	0.9899	0.9762	0.9750	0.9797	0.9797	0.9811	0.9820	0.9701	0.9761

where M1 = Baseline Model, M2 = Model with tanh activation and M3 = Model with ReLU activation. The shaded cells shows the function that show best performance in that class.

4.4 Inactive Neurons

The neurons for which the absolute value of the gradients is less than 10^{-5} are called inactive neurons. In this section, we do a comparative study on the percentage of the inactive neurons when the model is trained using different activation functions. The plot of the percentage of inactive neurons is shown in Figure 5.

Although ReLUs show very promising results, there are certain pitfalls of RELUs too. ReLUs can "die" in the training process [5]. It means that once a ReLU becomes inactive due to negative value of a parameter, the neuron stops working forever. This phenomena is clearly visible in Figure 5. The percentage of inactive neurons in case of ReLU is approximately half in the beginning and at the end of the training process, the percentage is higher than 90% of the total neurons. On the other hand, in case of sigmoid and tanh the percentage is significantly lower. This issue can be tackled by using modified ReLU functions like leaky ReLU, parametric ReLU (PReLU) [6] etc.

5 Regularization

Deep neural networks are often prone to overfitting if the training dataset is not sufficient. Regularization is a method to handle overfitting. We try to regularize our model by various means of regularization which are discussed in the subsequent sections.

5.1 Adding Noise

In this section we add gaussian noise to the h matrices during forward and backward pass. The noise added in the experiment $x \sim \mathcal{N}(0, 0.01)$. The training and test loss is shown in Figure 6 and 7.

The σ of the added noise is varied and the corresponding effects are observed. We observe that the *backward pass is highly sensitive* to addition of external noise and the loss explodes as $\sigma > 0.05$. On

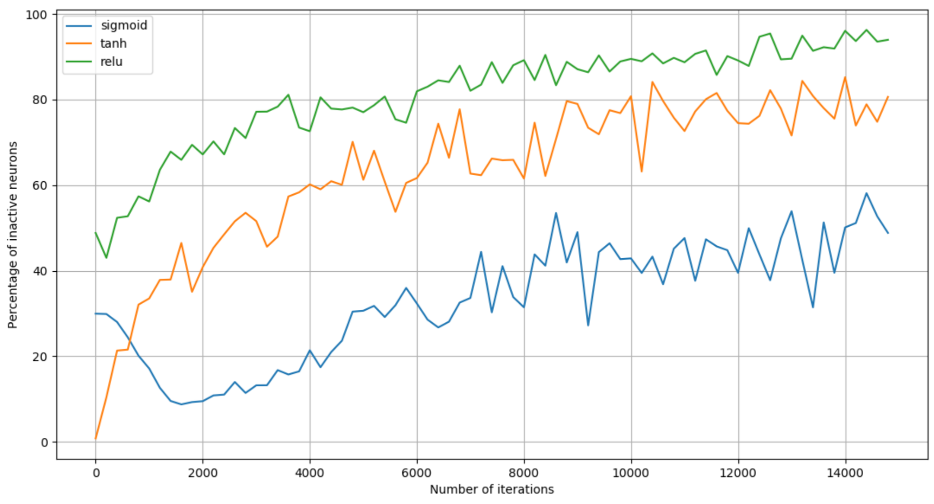

Figure 5: Percentage of inactive neurons as a function of the number of iterations

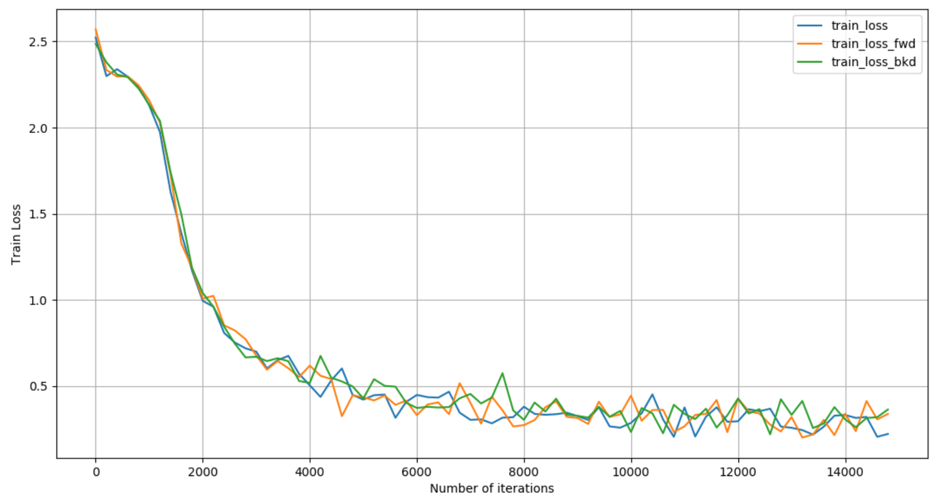

Figure 6: Variation of training loss when gaussian noise is added in training process

the other hand, the forward pass is much more robust to the addition of noise which is evident from the fact that the model manages to converge even when $\sigma = 1$, though the test accuracy decreases to ~ 0.86. A plausible explanation for the observation could be that the *gradients calculated in backpropagation is more crucial* than the weights for convergence of the loss function.

5.2 Data Augmentation

The size of the dataset is quite small as compared to most of the modern day datasets which may contain millions of data points. So, we try to "create" more data to train our models robustly. We add gaussian noise with $\sigma = 0.01$ to the *training set only* and create a new training dataset containing both noisy and noise-free data (i.e. 120K data points). The model is trained and tested three times on the augmented dataset and the mean accuracy and standard deviation is reported.

$$\text{Average Accuracy} = 0.9463$$
$$\text{Standard Deviation} = 0.0002$$

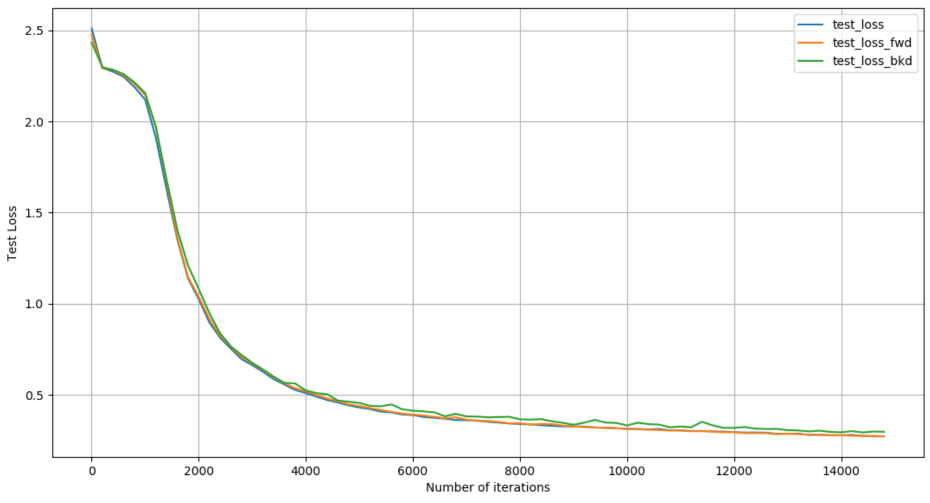

Figure 7: Variation of test loss when gaussian noise is added in training process

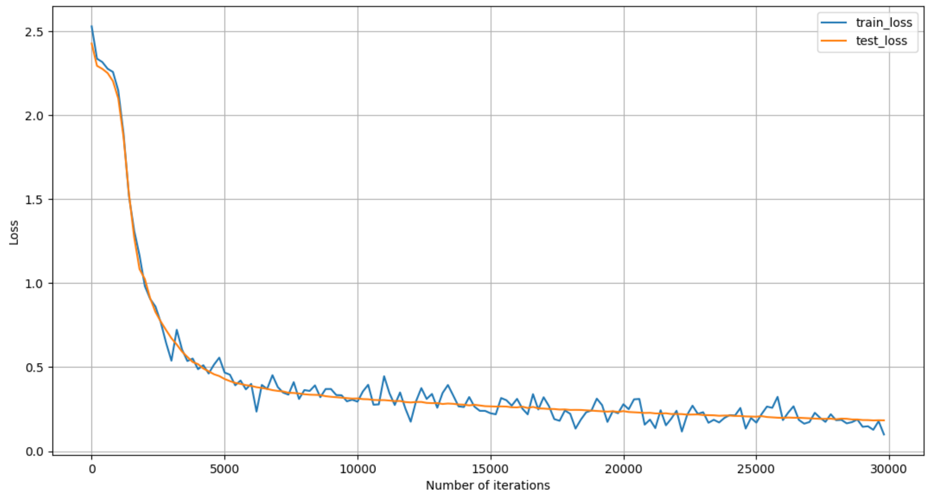

Figure 8: Variation of training and test loss when trained on augmented dataset

5.3 L2-Regularization

L2-Regularization is a widely acclaimed method of reduce overfitting. The L2-norm of the weight matrices are also taken into consideration while calculating the loss. A regularization parameter λ maintains the balance between bias and variance. Mathematically,

$$J_{new} = J_{old} + \lambda \|w\|_2^2 \tag{6}$$

For our model, we do a grid search over logarithmic scale from 1e-5 to 1 to find the best value of λ. The best performance is achieved with $\lambda = 0.0001$. However, the model fails to achieve any significant gain. Infact, the model performance reduces slightly after introducing regularization. The model is trained three times and the training loss of the model with and without regularization is shown in Figure 9.

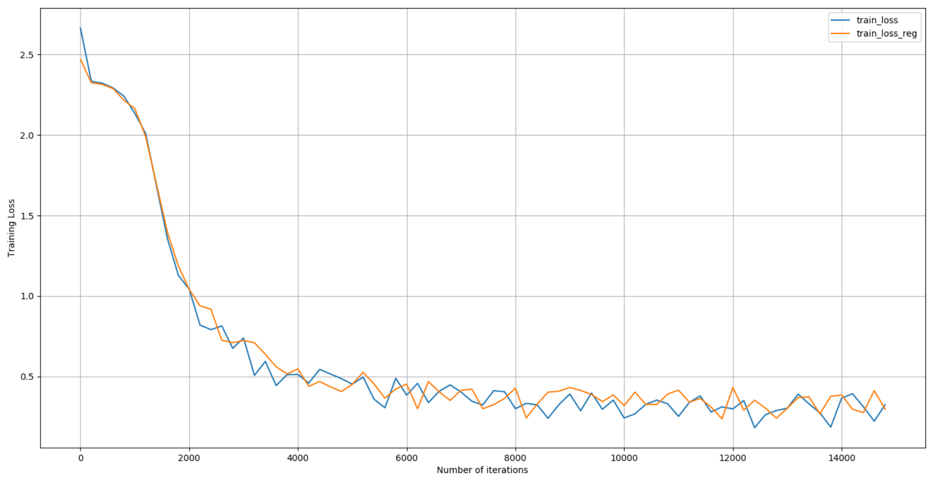

Figure 9: Variation of training loss with and without regularization

5.4 Results and Discussion III

The models with regularization are compared with the baseline model in this section. The model

Table 11: Comparison of the baseline model with models with regularization

	Baseline	**Data Aug.**	**L2 Reg.**
Training Loss	0.3089	**0.0995**	0.2976
Testing Loss	0.2740	**0.1836**	0.2879
Test Accuracy	0.9212	**0.9463**	0.9160
Standard deviation	0.0014	**0.0002**	0.0008

where Data Aug. = Data Augmentation, L2 Reg. = L2 Regularization. The values in bold represent the model with best performance in that particular parametric.

trained on the augmented dataset performs much better than the other two models as observed in Table 11. This emphasizes on the fact that most neural networks are "data hungry" i.e. the model performance improves in most of the cases when trained on a larger dataset. The L2-regularization actually causes a slight dip in accuracy but the difference between training and test loss is much lower as compared to the other two models which is an indication of ideal model training.

6 Hand-crafted Features

In this section, we try to train a neural network as well as implement two traditional machine learning techniques, namely 1) K-Nearest Neighbour and 2) Support Vector Machines on handcrafted features extracted from the images. We use Histogram of Gradients (HOG) [7] method to extract the features. In HOG, locally normalized histogram of gradient orientation similar to SIFT [8] descriptors is calculated. For extracting HOG features, `scikit-image` package in Python is used. We use 7×7 pixels per cell and number of cells in each block is 2. So, a total of **324** features is extracted.

6.1 Neural Network

The architecture and the initialization is similar to the baseline model except:

- The model consists of three fully connected hidden layers with 300, 150 and 70 neurons.

- The input to the neural network are feature vectors of size 324
- ReLU activation is used in all the hidden layers

The training and test loss is presented in Figure 10. The model is trained three times and the mean average and standard deviation is reported.

$$\text{Average Accuracy} = 0.9761$$
$$\text{Standard Deviation} = 0.0006$$

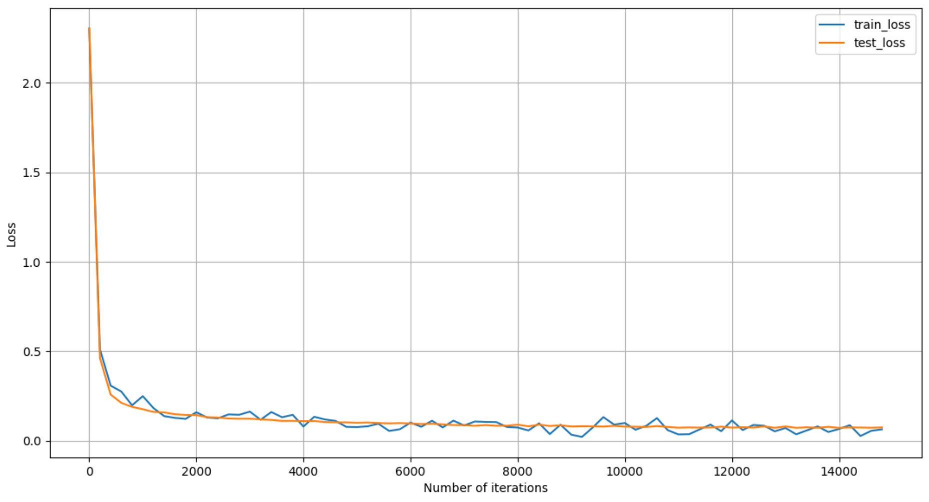

Figure 10: Variation of training and test loss when HOG features are used

6.2 K-Nearest Neighbours and Support Vector Machines

In this subsection, we utilize two traditional machine learning techniques to classify the features extracted from the dataset. We use K-Nearest Neighbour [9] based on Euclidean distance to classify the data points. The value of k is fixed to be 3 for the K-NN algorithm. For support vector machine, squared hinge loss function is used giving equal weightage to each class. We use `scikit-learn` package in Python to implement K-NN and SVM. The performance of the neural network, K-NN and SVM is compared in Table 12. The neural network outperforms the both K-NN as well as SVM. However, the standard deviation in case of K-NN and SVM is zero whereas performance of the neural network varies slightly with the initialization.

Table 12: Comparison of the models when trained on HOG features

	Neural Net	**K-NN**	**SVM**
Training Loss	0.0410	-	-
Testing Loss	0.0733	-	-
Test Accuracy	**0.9761**	0.9631	0.9691
Standard deviation	0.0006	-	-

The values in bold represent the model with best performance in that particular parametric.

7 Conclusion

In this assignment, we implement a neural network from scratch without using any deep learning framework. We trained models using different activation functions and found that ReLU activation outperforms the sigmoid activation by a significant margin (5.93%). The case of inactive neurons especially in case of ReLU activation is studied and methods are discussed to reduce it. We try various regularization techniques and compare their performance with the baseline model. We found that training the model with combined dataset of noise-free and noisy data gives the best results. In the final section, HOG features are extracted from the images and observed that a neural network trained on the features outperforms traditional machine learning techniques like K-Nearest Neighbours and Support Vector Machines.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[2] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 2010.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[4] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[5] Neural networks - part 1. `http://cs231n.github.io/neural-networks-1/`. Accessed on 30-Aug-2019.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893, June 2005.

[8] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[9] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.