Nisa Ahsen Öztürk

22001858

EEE 443 Mini Project 3:
Human Activity Recognition (HAR)

1. Introduction

The aim of this project is to build a human activity recognition (HAR) algorithm by using two hidden layers and one output layer. Dataset contains six different motion type. Moreover, training dataset has 7352 patterns and test dataset has 2943 patterns. First hidden layer has 300 neurons (N1). However, number of neurons for second layer can be changed as 100 or 200 (N2). Output layer has 6 neurons since there are 6 different motion type. For hidden layers activation function is selected as RELU function. Softmax function is used as activation function of output layer. The structure of the neural network is given below.

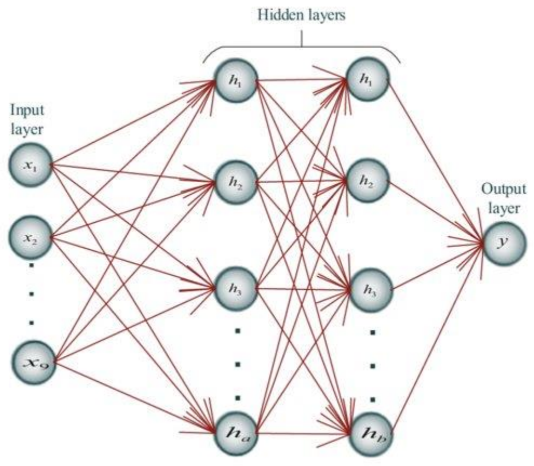


Figure 1: Structure of designed neural network [1]

2. Theory and Algorithm

2.1 Dataset Implementations Matrix Initialization

The dataset is loaded by using "np.loadtxt" function. After this step training and test labels are reshaped as (7352, 1) and (2947, 1) respectively. All of the parameters are initialized as desired.

For the sake of integrity of the project:

Size of input weight matrix is chosen as N1x561.

Size of first hidden layer's weight matrix is chosen as N2xN1.

Size of second hidden layer's weight matrix is chosen as 6xN2.

2.2 Forward Propagation

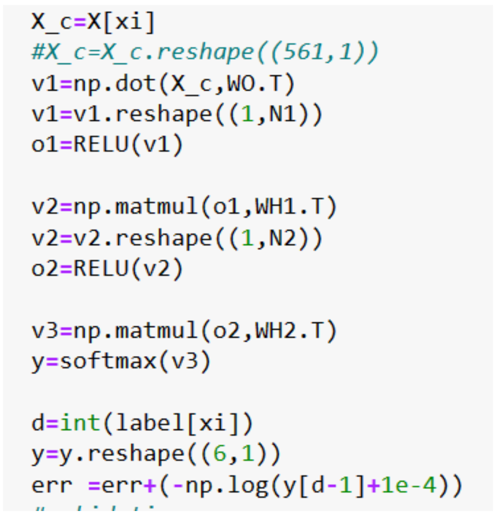
The forward propagation algorithm is similar to first mini project. The difference is that there are two hidden layers.

$$v1_{1xN1} = [x]_{1x561} * {WO_{N1x561}}^T$$
$$o1_{1xN1} = RELU(v1)$$

$$v2_{1xN2} = o1_{1xN1} * {WH1_{N2xN1}}^T$$
$$o2_{1xN2} = RELU(v2)$$

$$v3_{1x6} = o2_{1xN2} * {WH2_{6xN2}}^T$$
$$o_{1x6} = softmax(v3)$$

y matrix represents different motion types, a label is assigned to the output y for training and testing the algorithm. For the error function cross entropy is used.



```
X_c=X[xi]
#X_c=X_c.reshape((561,1))
v1=np.dot(X_c,WO.T)
v1=v1.reshape((1,N1))
o1=RELU(v1)

v2=np.matmul(o1,WH1.T)
v2=v2.reshape((1,N2))
o2=RELU(v2)

v3=np.matmul(o2,WH2.T)
y=softmax(v3)

d=int(label[xi])
y=y.reshape((6,1))
err =err+(-np.log(y[d-1]+1e-4))
```

Figure 2: Code for forward propagation

2.3 Back Propagation

In order to minimize the error, weights should be updated. Backpropagation is useful to update weights while training. Cost function is given below. Error for each x pattern and total error are given below.

$$E_x = -\sum_{k=1}^{6} d_k \log o_k = -\log y_k$$

$$E = \frac{1}{N}\sum_x E_x$$

After finding the cost error, the gradients are calculated from formulas given below.

$$\left(\frac{\partial E_x}{\partial o}\right) i = -\frac{d_i}{o_i}$$

$$\left(\frac{\partial o}{\partial v3}\right) ij = \begin{cases} o_i o_j & if\ i \neq j \\ o_i(1-o_i) & if\ \ i = j \end{cases}$$

Therefore,

$$\left(\frac{\partial E_x}{\partial v3}\right) = (d - o)^T$$

Gradient of WH2 can be defined as follows.

$$\frac{\partial E_x}{\partial WN2} = (d - o)x^T$$

Similarly gradient of WN1 and WO can be found.

$$\frac{\partial E_x}{\partial WN1} = \frac{\partial E_x}{\partial o}\frac{\partial o}{\partial v3}\frac{\partial v3}{\partial o2}\frac{\partial o2}{\partial v2}\frac{\partial v2}{\partial WN1}$$

$$\frac{\partial v3}{\partial o2} = WH2$$

$$\frac{\partial o2}{\partial v2}\frac{\partial v2}{\partial WN1} = diag\big(RELU'(v2)\big)x$$

Therefore,

$$\frac{\partial E_x}{\partial WN1} = (d - o)^T\, WH2\, diag\big(RELU'(v2)\big)x$$

Similarly, $\frac{\partial E_x}{\partial WO}$ can be found as follows.

$$\frac{\partial E_x}{\partial WO} = \frac{\partial E_x}{\partial o}\frac{\partial o}{\partial v3}\frac{\partial v3}{\partial o2}\frac{\partial o2}{\partial v2}\frac{\partial v2}{\partial v1}\frac{\partial v1}{\partial WO}$$

$$\frac{\partial o2}{\partial v2} = WH1$$

$$\frac{\partial v2}{\partial v1}\frac{\partial v1}{\partial WO} = \; diag\big(RELU'(v2)\big)x$$

Therefore,

$$\frac{\partial E_x}{\partial WO} = (d - o)^T\, WH2\, diag\big(RELU'(v2)\big)\, WH1\, diag\big(RELU'(v2)\big)x$$

After finding gradients, one can apply gradient descent method. However, it is asked to use momentum method to update weights. Therefore, one can update weights as follows.

$$WO(n+1) \Leftarrow WO(n) + \Delta \mathrm{WO}(n)$$

$$W\mathrm{H1}(n+1) \Leftarrow WH1(n) + \Delta \mathrm{WH1}\ (n)$$

$$W\mathrm{H2}(n+1) \Leftarrow WH2(n) + \Delta \mathrm{WH2}\ (n)$$

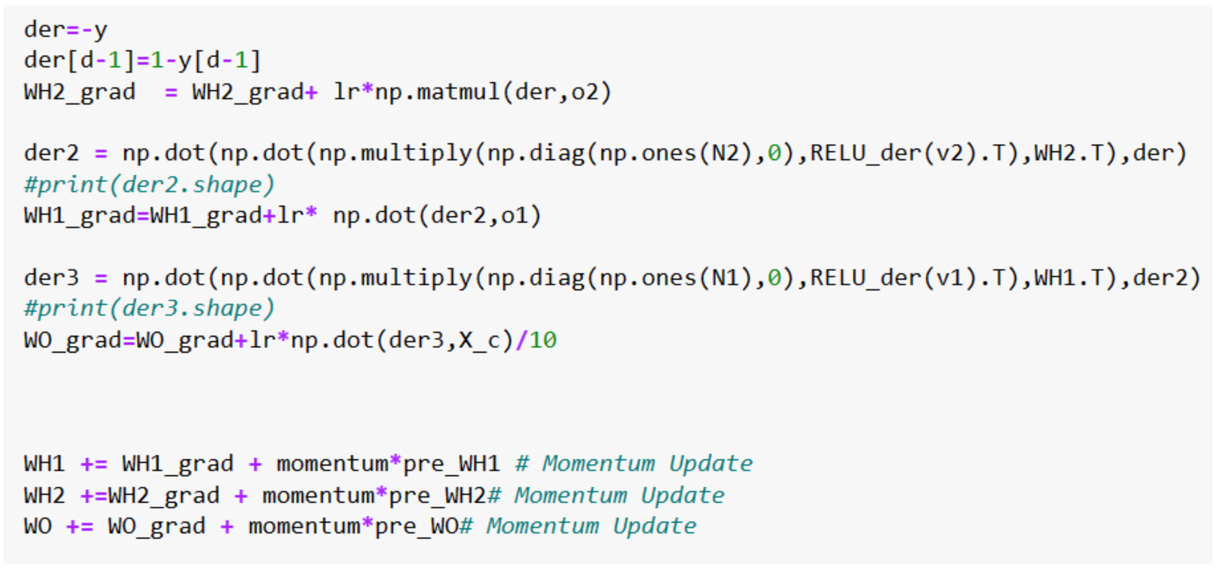
$$\Delta \mathrm{WO(n)} = -\,\eta\frac{\partial E}{\partial WO} + \;\alpha\,\Delta \mathrm{WO(n-1)}$$

$$\Delta \mathrm{WH1(n)} = -\,\eta\frac{\partial E}{\partial WH1} + \;\alpha\,\Delta \mathrm{WH1(n-1)}$$

$$\Delta \mathrm{WH2(n)} = -\,\eta\frac{\partial E}{\partial WH2} + \;\alpha\,\Delta \mathrm{WH2(n-1)}$$

Where α is momentum parameter.

Code for back propagation is given in Figure 3



```
der=-y
der[d-1]=1-y[d-1]
WH2_grad  = WH2_grad+ lr*np.matmul(der,o2)

der2 = np.dot(np.dot(np.multiply(np.diag(np.ones(N2),0),RELU_der(v2).T),WH2.T),der)
#print(der2.shape)
WH1_grad=WH1_grad+lr* np.dot(der2,o1)

der3 = np.dot(np.dot(np.multiply(np.diag(np.ones(N1),0),RELU_der(v1).T),WH1.T),der2)
#print(der3.shape)
WO_grad=WO_grad+lr*np.dot(der3,X_c)/10



WH1 += WH1_grad + momentum*pre_WH1 # Momentum Update
WH2 +=WH2_grad + momentum*pre_WH2# Momentum Update
WO += WO_grad + momentum*pre_WO# Momentum Update
```

Figure 3: Code for backpropagation

3. Results

Results are observed for different values of parameters. Number of neurons at hidden layer 2, learning rate (lr), momentum (m), batch size are variables. Each parameter has two different values. Therefore, there are 16 distinct cases. A 1x6 matrix is given as an output to misclassified patterns. First element of the matrix represents the number of misclassification for type 1 motion. Similarly, each index of this matrix is used for displaying the number of misclassified elements during testing.

No Batch Training and Testing:

	Test Accuracy (%)	Misclassified Patterns (Testing) (from 1 to 6)	Misclassified Patterns (Training) (from 1 to 6)

N1=100
lr=0.001
m=0
	93.993	[19,39,18,38,63,0]	[1226,1073,986,1286,1369,0]

N1=100
lr=0.01
m=0
	92.67	[68,13,35,56,44,0]	[1226,1073,986,1285,1367,0]

N1=100
lr=0.001
m=0.9
	93.1455	[18,26,18,81,59,0]	[1226,1073,986,1287,1368,0]

N1=100
lr=0.01
m=0.9
	91.55	[19,31,75,10,114,0]	[1226,1073,985,1288,1353,0]

N1=200
lr=0.001
m=0
	94.33	[33,23,11,47,53,0]	[1226,1073,986,1287,1367,0]

N1=200
lr=0.01
m=0
	80.522	[53,4,23,417,76,0]	[1226,1073,986,1287,1364,0]

N1=200
lr=0.001
m=0.9
	82.219	[48,19,6,401,49,0]	[1226,1073,986,1291,1368,0]

N1=200
lr=0.01
m=0.9
	87.44	[262,11,5,32,60,0]	[1225,1072,986,1294,1353,0]

Table 1: No Batch results

Batch Size = 50 Training and Testing

	Test Accuracy (%)	Misclassified Patterns (Testing) (from 1 to 6)	Misclassified Patterns (Training) (from 1 to 6)

N1=100
lr=0.001
m=0
	92.738	[35,56,8,97,17,0]	[1226,1072,987,1283,1353,0]

N1=100
lr=0.01
m=0
	83.135	[20,43,22,371,41,0]	[1225,1073,986,1294,1351,0]

N1=100
lr=0.001
m=0.9
	76.4845	[193,3,119,365,13,0]	[1226,1073,986,1285,1364,0]

N1=100
lr=0.01
m=0.9
	85.816	[74,40,19,281,4,0]	[1226,1073,985,1988,1367,1]

N1=200
lr=0.001
m=0
	94.876	[24,28,6,41,52,0]	[1226,1073,986,1289,1354,0]

N1=200
lr=0.01
m=0
	79.233	[71,19,18,482,22,0]	[1226,1072,986,1296,1352,0]

N1=200
lr=0.001
m=0.9
	88.29	[43,24,10,213,56,0]	[1226,1073,986,1294,1353,0]

N1=200
lr=0.01
m=0.9
	93.04	[56,44,11,34,60,0]	[1226,1072,986,1291,1349,0]

Table 2: Results when batch size = 50

No Batch Error Graphs

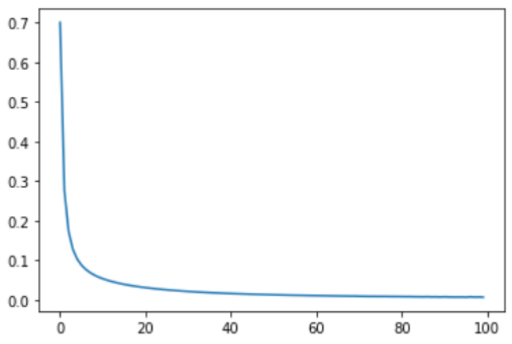


Figure 4: Error when N2=100, no batch, ,lr=0.001, m=0

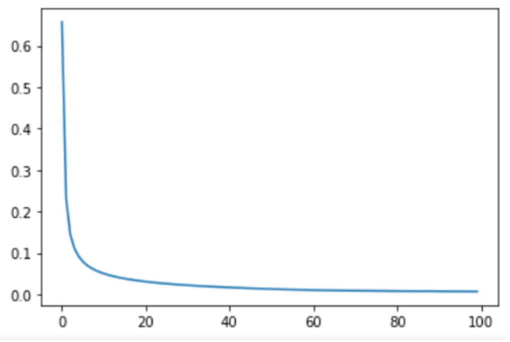


Figure 5: Error when N2=200, no batch, ,lr=0.001, m=0

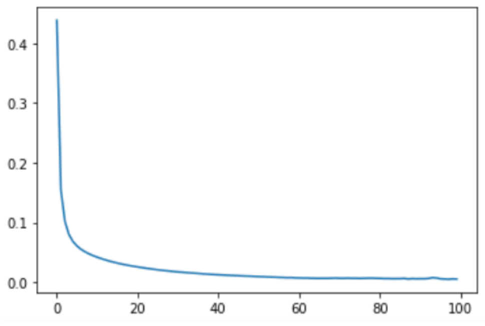


Figure 6: Error when N2=200, no batch, ,lr=0.001, m=0.9

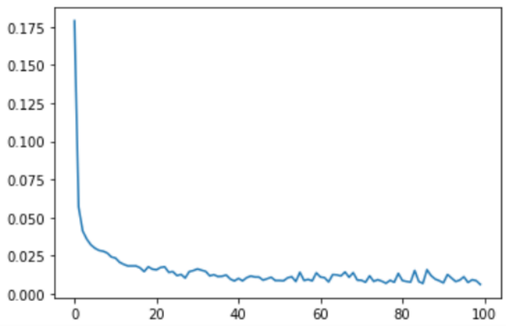


Figure 7: Error when N2=100, no batch, ,lr=0.01, m=0

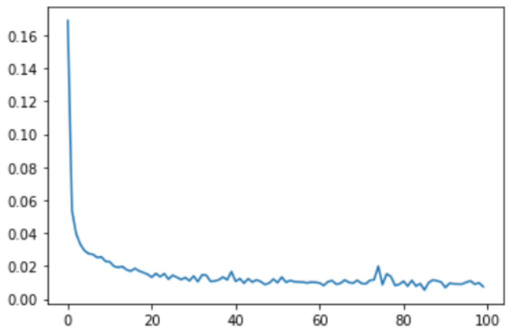


Figure 8: Error when N2=200, no batch, ,lr=0.01, m=0

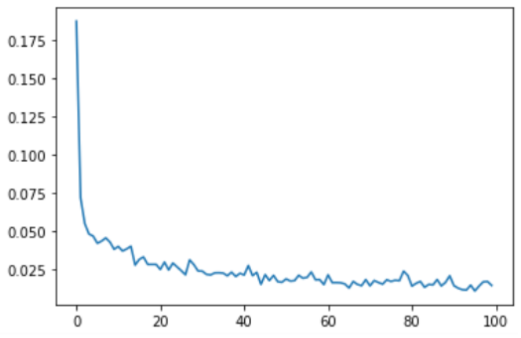


Figure 9: Error when N2=200, no batch, ,lr=0.01, m=0.9

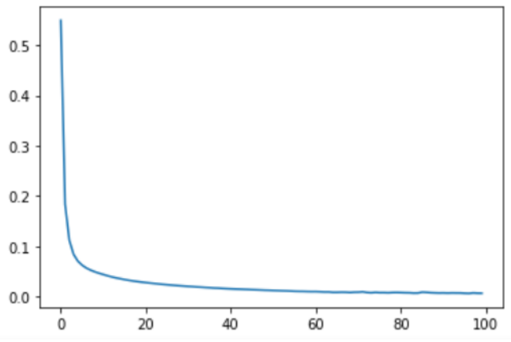


Figure 10: Error when N2=100, no batch, ,lr=0.001, m=0.9

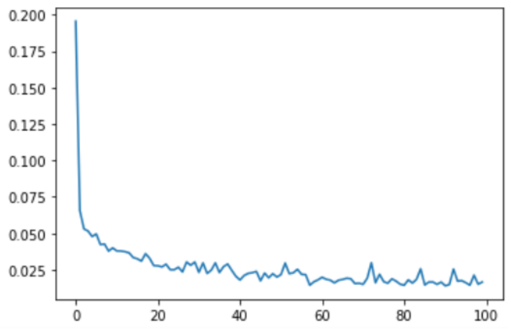


Figure 12: Error when N2=100, no batch, ,lr=0.01, m=0.9

Batch Size=50 Error Graphs

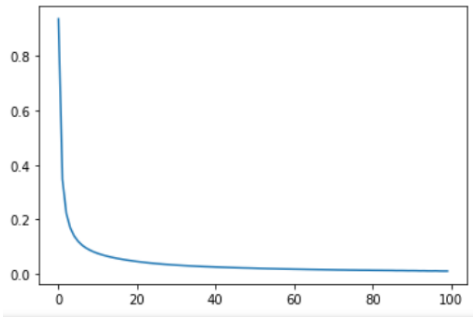


Figure 12: Error when N2=200, bs=50, lr=0.001, m=0

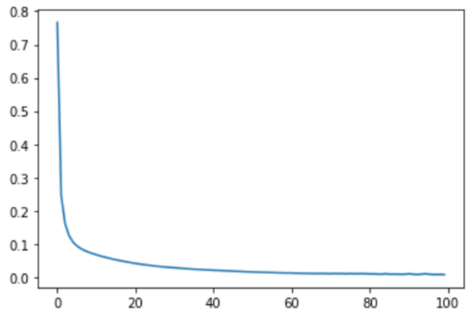


Figure 13: Error when N2=100, bs=50, lr=0.001, m=0.9

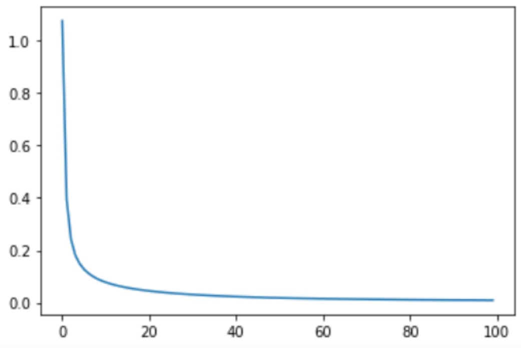


Figure 14: Error when N2=100, bs=50, lr=0.001, m=0

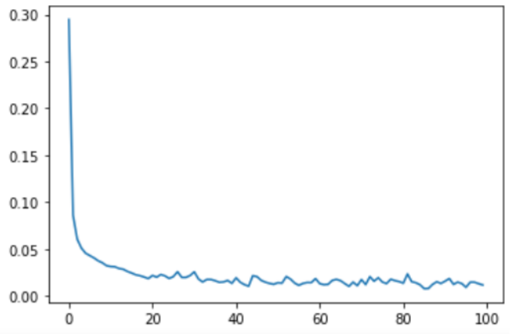


Figure 15: Error when N2=100, bs=50, lr=0.01, m=0

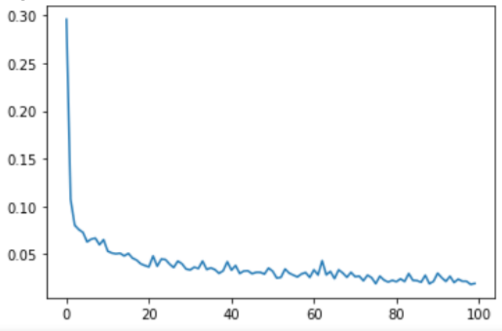


Figure 16: Error when N2=100, bs=50, lr=0.01, m=0.9

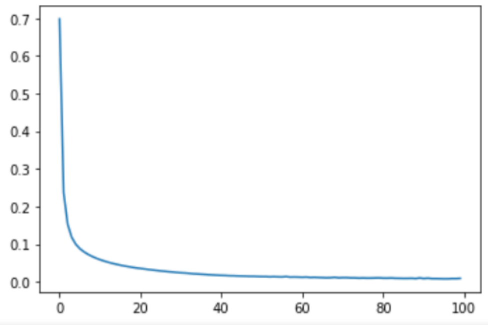


Figure 17: Error when N2=200, bs=50, lr=0.001, m=0.9

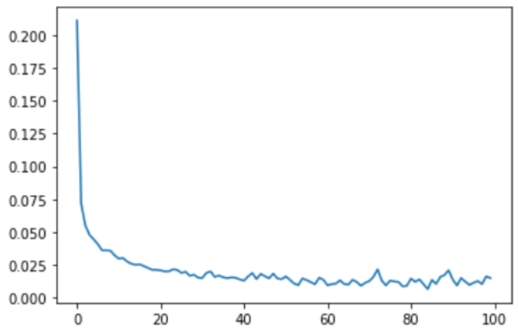


Figure 18: Error when N2=200, bs=50, lr=0.01, m=0

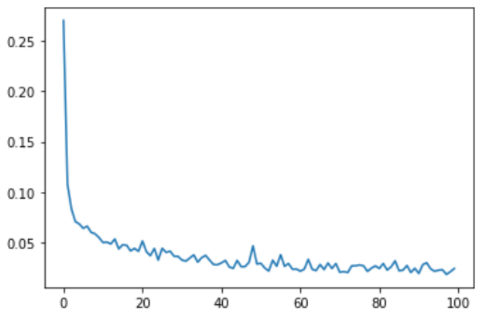


Figure 19: Error when N2=200, bs=50, lr=0.01, m=0.9

From Table 1 and 2 one can understand that the best case occurs when N2=200, learning rate= 0.001 and momentum=0. Best case configuration is expected since the learning rate is lower and number of neurons for second hidden layer is larger. From Figure 4 to 19, it can be seen that when there is momentum in the learning algorithm, error decreases rapidly compared to the cases which momentum is zero. This means that in the beginning of training, momentum speeds up the learning process. However, momentum did not create crucial change in terms of accuracy. There is not significant accuracy difference between online learning and learning with mini-batch. Nevertheless, mini-batch cases are slightly better than online learning. This result is expected since updating gradients in mini-batch learning is more precise.

After that, misclassified classes are observed. Table 1 and 2 display the number of misclassifications for each motion type. It can be seen that motion types 1,2 and 4 are the most misclassified motions. One of the surprising results for misclassification is that the algorithm was nearly 100% successful to predict motion type 6 correctly both in training and testing. On the other hand, the reason behind this success rate could not be understood.

Finally, a dropout algorithm with p=0.5 is written for second hidden layer. The algorithm is run for only the best case that is determined before. p=0.5 represents that half of the neurons in second hidden layer is omitted. Therefore, there is no contribution to training or testing from those neurons which are dropped. The outcomes of this implementation are given below.

N2, lr, m, batch, p	Accuracy (%)	Misclassified Patterns (Testing) (from 1 to 6)
200, 0.001, 0, 50, 0.5	94.72	[16,14,72,58,25,0]

Table 3: Results of dropout algorithm

The accuracy when there is no dropout is observed as 94.876%, when dropout is applied accuracy is 94.72. There is not any significant change in accuracy. The difference can be derived from the weight initialization at the beginning of each training. Therefore, for this project, dropout was not successful to increase the accuracy.

4. Conclusion

The aim of this project was to design a feed-forward neural network for HAR. Two hidden layers are used in the network. While designing the network, gradients are calculated by considering the effect of momentum. The calculations are transferred into code by using Python.

Secondly, outcomes are observed and the effect of each parameter is examined. It is observed that the momentum constant has an effect on the speed of learning in the first epochs. Therefore, it can affect the steepness of the slope of error graphs. However, a significant effect cannot be observed in terms of accuracy. Moreover, best case is observed when momentum was equal to zero. Furthermore, using batches increase the overall accuracy of the network and the smaller learning rate gives more correct predictions. After, training a dropout algorithm is written to detect whether it will increase the accuracy or not. However, there was not any change in accuracy percentage. Therefore, dropout method was not efficient. It may change accuracy if different values are given to p parameter. The results of misclassified motion types were unexpected since one of the motions is predicted correctly at nearly 100%. Theoretically, it is expected that the misclassification percentage of motions should not vary too much especially in training. However, this was not the case in experiment. In conclusion, the project was successful since accuracy rates are high and the algorithm is capable of learning.

REFERENCES

[1] Aqib M. et al. “Disaster Management in Smart Cities by Forecasting Traffic Plan Using Deep Learning and GPUs” *ResearchGate.* [Online]. (Accessed: 15 Dec. 2023) https://www.researchgate.net/publication/326538421_Disaster_Management_in_Smart_Cities_by_Forecasting_Traffic_Plan_Using_Deep_Learning_and_GPUs