

```
1 import numpy as np
2 import h5py
3 import random
4 import matplotlib as plt
5 from random import seed
6 from random import randrange
7 from random import random
8 from csv import reader
9 from math import exp
10 import random
11 import time
12 import sys
13 import matplotlib.pyplot as plt
14 from tempfile import TemporaryFile
15
16 def formonehot(data,i):
17     data = np.array(data)
18     onehot = np.array(data/i)
19     onehot[onehot != 1] = 0
20     return onehot
21
22 def crossentropyloss(predicted,label):
23     sum = 0
24     for i in range(len(predicted)):
25         labels = onehotlabel(label)
26         sum += -1*np.sum(labels*np.log(np.amax(predicted[i])))
27     return sum/372500
28
29 def onehotlabel(label):
30     onehot = np.zeros(250)
31     onehot[label] = 1
32     return onehot
33
34 def softmax(predicteds):
35     softmaxs = np.exp(predicteds - np.max(predicteds))/np.sum(np.exp(predicteds - np.ma
36     return softmaxs
37
38 def calculatemse(error,o,flag =False):
39     mse = 0
40     err= 0
41     error = error.T
42     for errorofiteration in error:
43         for column in errorofiteration:
44             err+=column
45         mse = mse + err*err
46     mse = mse/np.array(o).shape[0]
47     return mse
48
49 def classError(o,lbls):
50     acc = accuracy(o,lbls)
51     return acc
52
53 def accuracy(o,lbls):
54     count = 0
```

```

55     predict = np.argmax(o,axis=1)
56     for i in range(lbls.shape[0]):
57         if predict[i]==lbls[i]:
58             count = count+ 1
59     return count/lbls.shape[0]*100
60
61 def organizeInput(image):
62     x = np.ones(32*32)
63     index = 0
64     for row in image:
65         for column in row:
66             x[index]=((column))
67             index = index + 1
68     return x
69
70 def activation(inn,deriv =False):
71     inn = np.tanh(inn)
72     if deriv == True:
73         return 1-inn*inn
74     return inn
75
76 # Rescale dataset columns to the range 0-1
77 def normalize_dataset(arr):
78     arr = arr - np.amin(arr,axis=0)
79     arr = arr / np.amax(arr,axis = 0)
80     return arr
81
82 # Find the min and max values for each column
83 def organizeData(dataPath):
84     data = h5py.File(dataPath,'r+')
85     trainims =(data['trainims'])
86     trainlbls =(data['trainlbls'])
87     testims=(data['testims'])
88     testlbls =(data['testlbls'])
89     return trainims,trainlbls,testims,testlbls
90
91 def organizeData2(dataPath):
92     data = h5py.File(dataPath,'r+')
93     trainx =np.array((data['trainx']))
94     traind =np.array((data['traind']))
95     valx=(data['valx'])
96     vald =(data['vald'])
97     testx=(data['testx'])
98     testd =(data['testd'])
99     words = data['words']
100     return trainx,traind,valx,vald,testx,testd,words
101
102 def addBiasToDataset(dataset):
103     size = dataset.shape[0]
104     biasInDataset = -1*np.ones(size)
105     return np.hstack((dataset,(biasInDataset.reshape(-1,1))))
106
107 def addbiasToInputs(layerInput):
108     return np.hstack((layerInput,[-1]))
109

```

```

110 def forwardpropagate(W1,W2,testdata):
111     outputs = list()
112     for img in testdata:
113         output = activation(np.dot(W2,activation(addbiasToInputs(np.dot(W1,img)))))
114         outputs.append(output)
115     return np.array(outputs)
116
117 def normalize(v):
118     norm = np.linalg.norm(v)
119     if norm == 0:
120         return v
121     return v / norm
122
123 def shuffle(data,lbls):
124     deck = ((np.hstack((data,lbls.reshape(1,-1).T))))
125     deck = np.array(sorted(deck, key=lambda k: random.random()))
126     data = deck.T[:-1].T
127     lbls = deck.T[-1]
128     return data,lbls
129
130 def shuffle3(data,lbls):
131     deck = ((np.hstack((data,lbls.reshape(1,-1).T))))
132     deck = np.array(sorted(deck, key=lambda k: random.random()))
133     data = deck.T[:-1].T
134     lbls = deck.T[-1]
135     return data,lbls
136
137 def formdata():
138     Q2data =("/content/assign2_data1.h5")
139     trainims,trainlbls,testims,testlbls = organizeData(Q2data)
140     traindata = list()
141     testdata = list()
142
143     for image in trainims:
144         inputs = organizeInput(image)
145         traindata.append(inputs)
146
147     for image in testims:
148         inputs = organizeInput(image)
149         testdata.append(inputs)
150
151     traindata = np.array(traindata,dtype = float)
152     trainlbls = np.array(trainlbls,dtype = float)
153     testdata = np.array(testdata,dtype = float)
154     testlbls = np.array(testlbls,dtype = float)
155
156     traindata = (normalize_dataset(traindata))
157     testdata = (normalize_dataset(testdata))
158     traindata = addBiasToDataset(traindata)
159     testdata = addBiasToDataset(testdata)
160     return traindata,trainlbls,testdata,testlbls

```

```

1 def runNetwork(hiddenNo,epochno,miniBatchno,learningrate,alpha=0):
2     print("Hidden layer number:"+str(hiddenNo)+" Learning rate:"+str(learningrate)+" Ep

```

```

3  # handling array constructions
4  # trainx,traind,valx,vald,testx,testd = formdata2()
5  traindata,trainlbls,testdata,testlbls = formdata()
6
7  What= np.random.normal(0, 1/1025,(hiddenNo,1024))
8  bhat = np.random.normal(0, 1/1025,(hiddenNo)).reshape(-1,1)
9  What =np.hstack((What, bhat))
10
11  W= np.random.normal(0, 1/hiddenNo,(2,hiddenNo))
12  b = np.random.normal(0, 1/hiddenNo,(2)).reshape(1,-1)
13  W =np.hstack((W, b.T))
14
15  errorsFormse = list()
16  errorsForClassification = list()
17  errorsForTestClassifications = list()
18  errorsForTestmse = list()
19  # Start training process
20  for epoch in range(epochno):
21      errorsForEpochs = list()
22      # display epoch index and current time
23      if epoch==0:
24          print("Training started.   Time: "+time.ctime())
25          title = ("\r"+"Epoch no: "+str(epoch)+" in "+str(epochno))
26          sys.stdout.write(title +" Time: " +time.ctime())
27          sys.stdout.flush()
28          time.sleep(1)
29          trainouts = list()#-----
30          lblss = list()
31          for iterateno in range(1900):
32              # forward propagate
33              # handling array constructions
34              i = iterateno
35              img = traindata[i]
36              d = np.zeros(2)
37              lbl = trainlbls[i]
38              lblss.append(lbl)
39              d[int(lbl)] = 1
40              # activate hidden layer
41              vhat = list(np.dot(What,img))
42              vhat.append(-1)
43              vhat = np.array(vhat,dtype = float)
44              y = activation(vhat).T
45              # activate output layer
46              v = np.dot(W,y)
47              o = activation(v)
48              e = (d-o)/1900
49              trainouts.append(o)#-----
50              # store the error
51              errorsForEpochs.append(e)
52              # back propagate
53              # back propagate the output layer
54              Tprime = activation(v,deriv = True)*np.identity(2,dtype=float)
55              gradient = (np.dot(Tprime,e))
56              # back propagate the hidden layer
57              Tprimehat = activation(vhat,deriv = True)*np.identity(hiddenNo+1,dtype = fl

```

```

58     gradienthat = (np.dot((np.dot(lprimehat,W.I)),gradient))
59     # mini batch updates
60     if iterateno%miniBatchno== 0:
61         if not(iterateno==0 and epoch==0):#The first iteration of the first epo
62             W +=learningrate*normalize(Wupdate)/miniBatchno
63             What+=learningrate*normalize(Whatupdate)/miniBatchno
64             Wupdate = (np.dot(gradient.reshape(-1,1),y.reshape(1,-1)))
65             Whatupdate = (np.dot(gradienthat.reshape(-1,1),img.reshape(-1,1).T)[: -1
66         else:
67             Wupdate += (np.dot(gradient.reshape(-1,1),y.reshape(1,-1)))
68             Whatupdate += (np.dot(gradienthat.reshape(-1,1),img.reshape(-1,1).T)[: -
69     lblss = np.array(lblss)
70     trainouts = np.array(trainouts)
71     classificationError = classError(trainouts,lblss)
72     errorsForClassification.append(classificationError)
73
74     errorsForEpochs = np.array(errorsForEpochs)
75     mse=calculatemse(errorsForEpochs,o).reshape(-1,1)
76     errorsFormse.append(float(mse))
77
78     WtT = W.T[: -1].T
79     WthatT = (What.T[: -1]).T
80     errorsForEpochsTest = list()
81     testouts = list()
82     for i in range(testdata.shape[0]):
83         # forward propagate
84         imgT = testdata[i][: -1]
85         lblT = testlbls[i]
86         #
87         dT = np.zeros(2)
88         lblT = trainlbls[i]
89         dT[int(lblT)] = 1
90
91         vhatT =np.dot(WthatT,imgT).reshape(1,-1)
92         yT = activation(vhatT).T
93
94         vT = np.dot(WtT,yT)
95         oT = activation(vT)
96
97         eT = dT - oT
98         testouts.append(oT)
99         errorsForEpochsTest.append(eT)
100     # lbls = np.array(lbls)
101     testouts = np.array(testouts)
102     testclassifications = classError(testouts,testlbls)
103     errorsForTestClassifications.append(testclassifications)
104
105     errorsForEpochsTest = np.array(errorsForEpochsTest)
106     msetest = calculatemse(errorsForEpochsTest,o).reshape(-1,1)
107     errorsForTestmse.append(float(mse))
108     if testclassifications>72:
109         break
110     # Adjusting weight matrices by de-concetanating the bias term
111     print("Train mean squared errors for epochs")
112     errorsFormse= np.array(errorsFormse)

```

```
113 plt.plot(errorsFormse)
114 plt.show()
115 print("Train percentage classification errors for epochs")
116 errorsForClassification = np.array(errorsForClassification)
117 plt.plot(errorsForClassification)
118 plt.show()
119 print("Test mean squared errors for epochs")
120 errorsForTestmse = np.array(errorsForTestmse)
121 plt.plot(errorsForTestmse)
122 plt.show()
123 print("Test percentage classification errors for epochs")
124 errorsForTestClassifications = np.array(errorsForTestClassifications)
125 plt.plot(errorsForTestClassifications)
126 plt.show()
127 print("Maximum accuracy is obtained at epoch:"+str(np.argmax(errorsForTestClassific
```

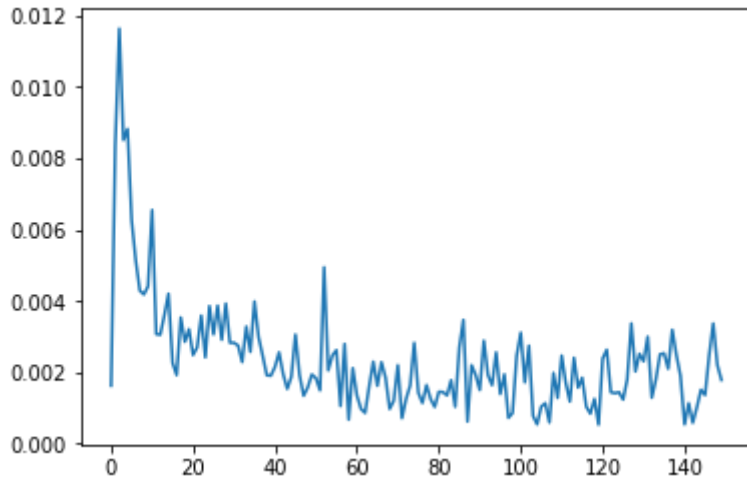
```
1 # hiddenNo,epochno,minibatchno,learningrate,alpha
2
3 runNetwork(30,100,30,0.3,0)
4 runNetwork(150,200,40,0.5,0)
5 runNetwork(30,300,60,0.3,0)
```



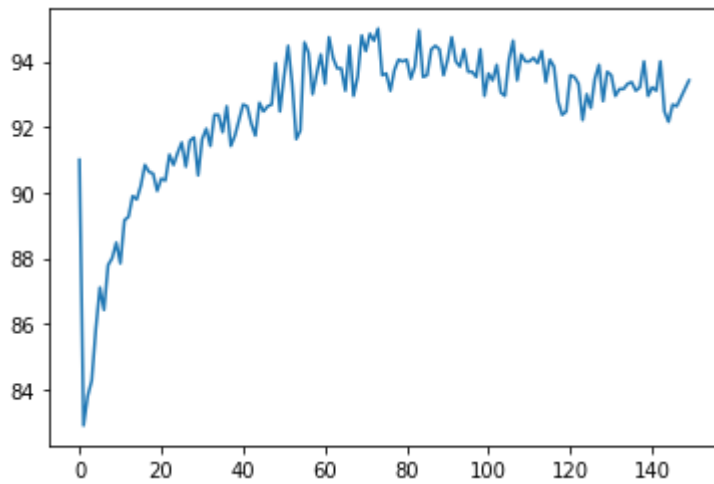
Hidden layer number:50 Learning rate:0.2 Epoch number:150 Batch size:10

Training started. Time: Sun Nov 17 19:24:33 2019

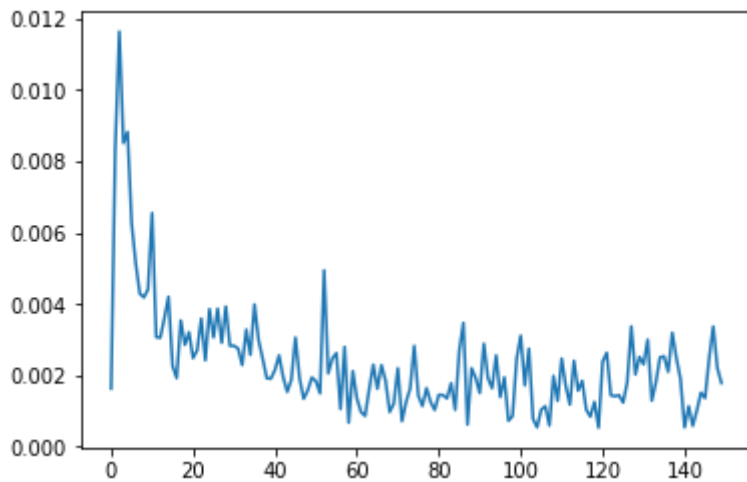
Epoch no: 149 in 150 Time: Sun Nov 17 19:29:00 2019 Train mean squared errors for epoch



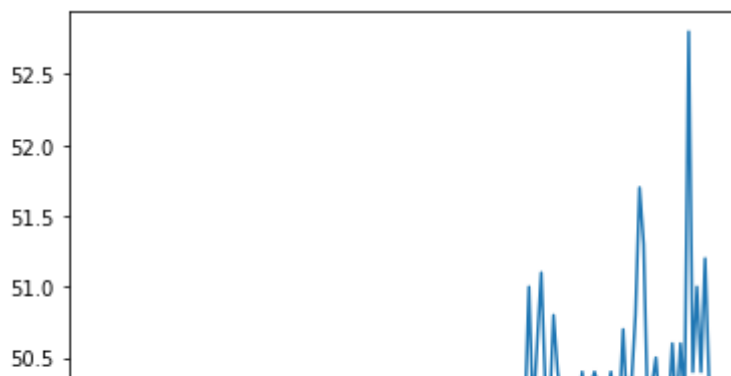
Train percentage classification errors for epochs

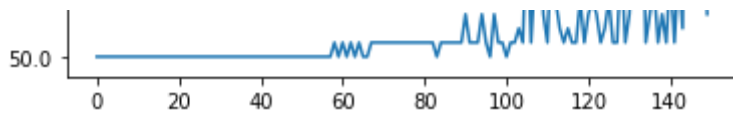


Test mean squared errors for epochs



Test percentage classification errors for epochs



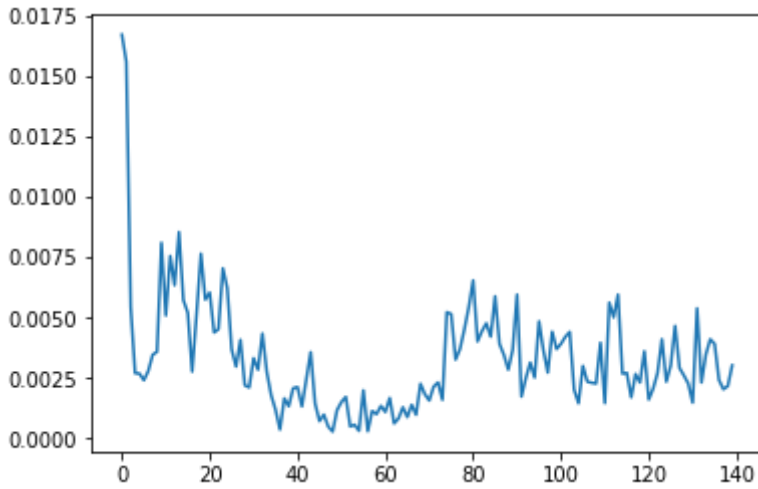


Maximum accuracy is obtained at epoch:144 as 52.800000000000004%

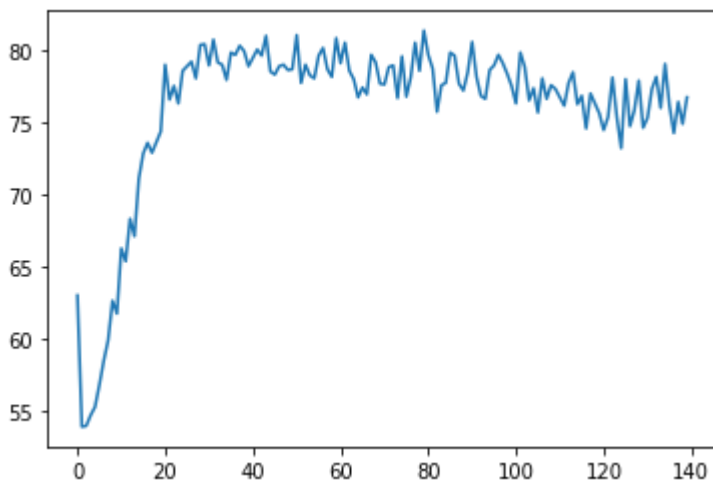
Hidden layer number:30 Learning rate:0.3 Epoch number:140 Batch size:30

Training started. Time: Sun Nov 17 19:29:04 2019

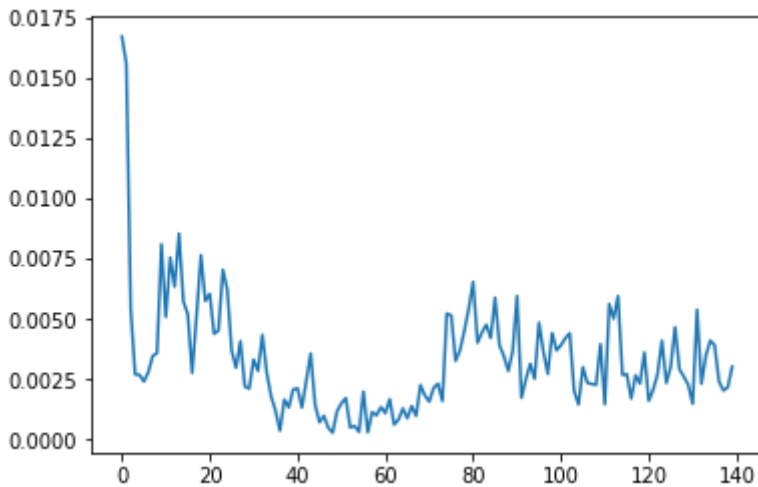
Epoch no: 139 in 140 Time: Sun Nov 17 19:32:17 2019 Train mean squared errors for epoc



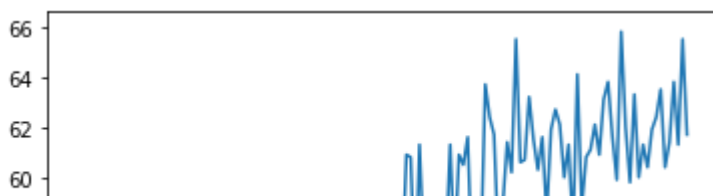
Train percentage classification errors for epochs

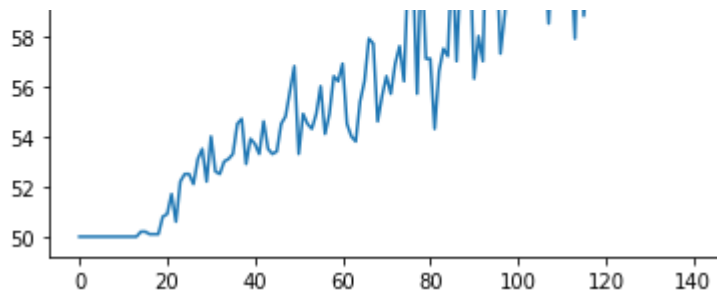


Test mean squared errors for epochs



Test percentage classification errors for epochs



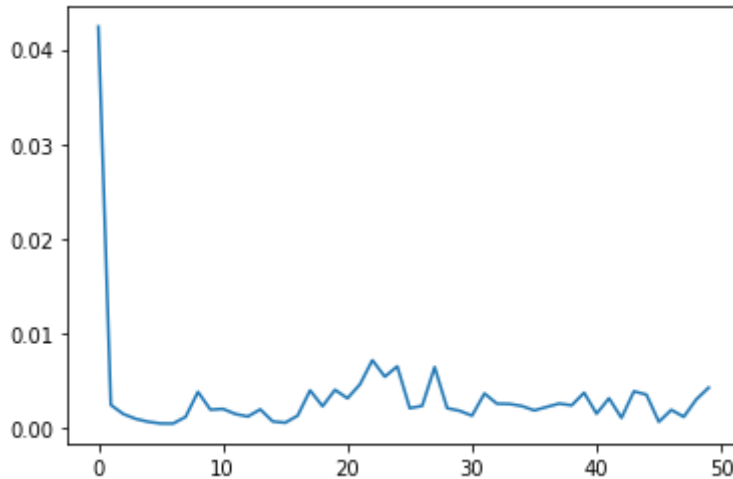


Maximum accuracy is obtained at epoch:124 as 65.8%

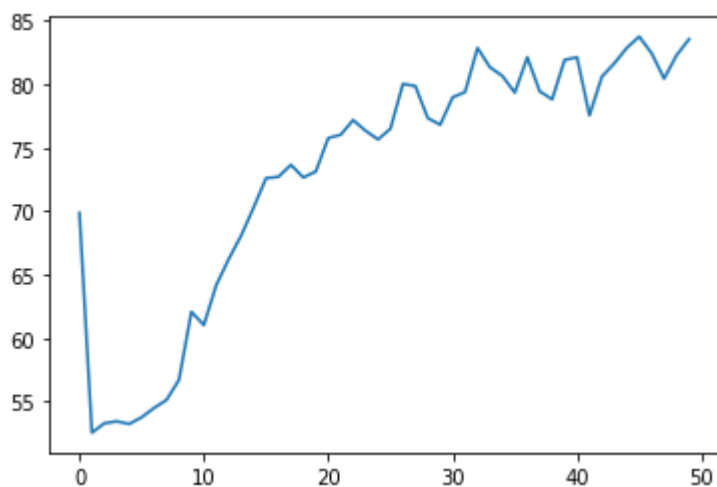
Hidden layer number:150 Learning rate:0.5 Epoch number:50 Batch size:40

Training started. Time: Sun Nov 17 19:32:20 2019

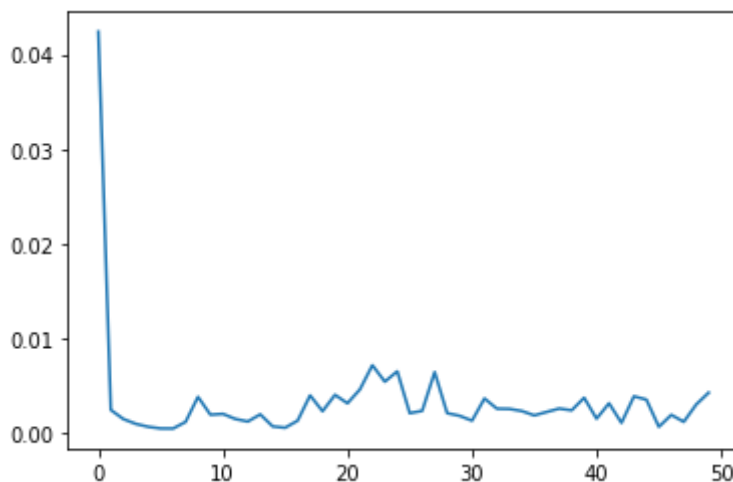
Epoch no: 49 in 50 Time: Sun Nov 17 19:34:41 2019 Train mean squared errors for epochs



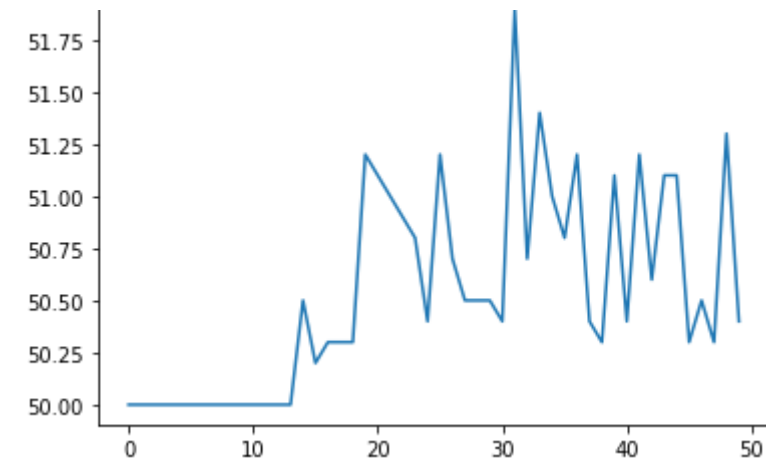
Train percentage classification errors for epochs



Test mean squared errors for epochs



Test percentage classification errors for epochs



Maximum accuracy is obtained at epoch:31 as 51.9%

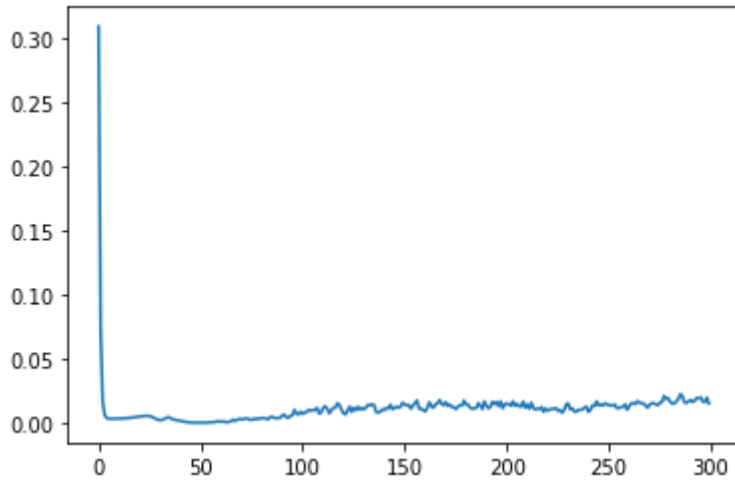
```
1 runNetwork(30,300,60,0.3,0)
```



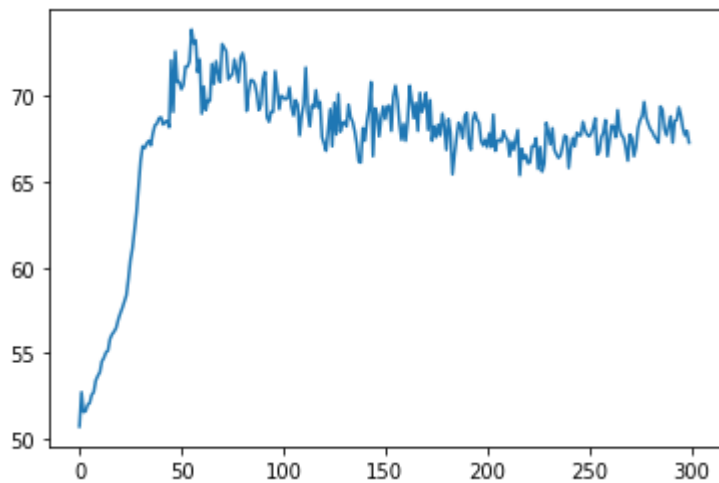
Hidden layer number:30 Learning rate:0.3 Epoch number:300 Batch size:60

Training started. Time: Sun Nov 17 19:58:30 2019

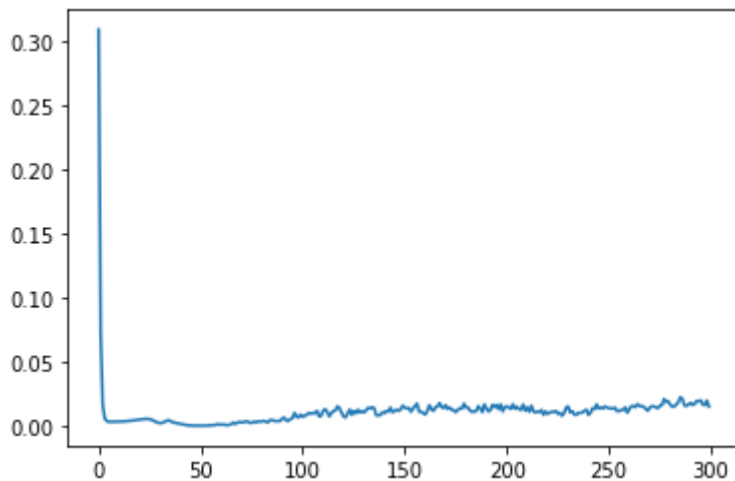
Epoch no: 299 in 300 Time: Sun Nov 17 20:05:25 2019 Train mean squared errors for epoc



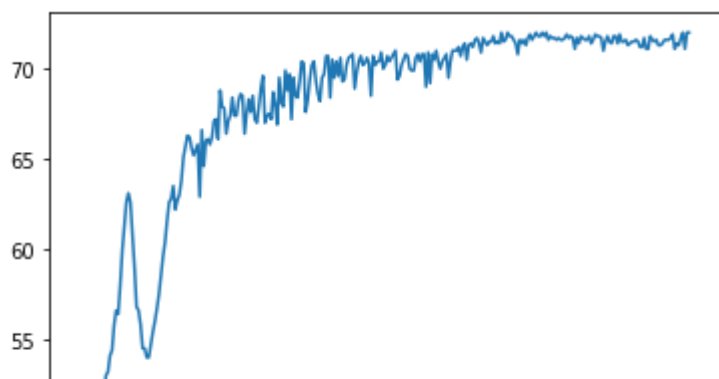
Train percentage classification errors for epochs

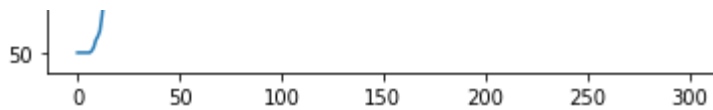


Test mean squared errors for epochs



Test percentage classification errors for epochs





Maximum accuracy is obtained at epoch:207 as 72.0%

```

1 def runNetwork2(hidden1No,hidden2No,epochno,miniBatchno,learningrate,alpha):
2     print("Hidden1 layer number:"+str(hidden1No)+" Hidden2 layer number:"+str(hidden2No)
3     # handling array constructions
4     traindata,trainlbls,testdata,testlbls = formdata()
5
6     What1= np.random.normal(0, 1/1025,(hidden1No,1024))
7     bhat1 = np.random.normal(0, 1/1025,(hidden1No)).reshape(-1,1)
8     What1 =np.hstack((What1, bhat1))
9
10    What2= np.random.normal(0, 1/hidden1No,(hidden2No,hidden1No))
11    bhat2 = np.random.normal(0, 1/1/hidden1No,(hidden2No)).reshape(-1,1)
12    What2 =np.hstack((What2, bhat2))
13
14    W= np.random.normal(0, 1/hidden2No,(2,hidden2No))
15    b = np.random.normal(0, 1/hidden2No,(2)).reshape(1,-1)
16    W =np.hstack((W, b.T))
17
18    errorsFormse = list()
19    errorsForClassification = list()
20    errorsForTestClassifications = list()
21    errorsForTestmse = list()
22    # Start training process
23    for epoch in range(epochno):
24        errorsForEpochs = list()
25        # display epoch index and current time
26        if epoch==0:
27            print("Training started.   Time: "+time.ctime())
28            title = ("\r"+"Epoch no: "+str(epoch)+" in "+str(epochno))
29            sys.stdout.write(title +" Time: " +time.ctime())
30            sys.stdout.flush()
31            time.sleep(1)
32            lbls = list()
33            trainouts = list()#-----
34            for iterateno in range(1900):
35                # forward propagate
36                # handling array constructions
37                i = random.randrange(traindata.shape[0])
38                img = traindata[i]
39                d = np.zeros(2)
40                lbl = trainlbls[i]
41                lbls.append(lbl)
42                d[int(lbl)] = 1
43                # activate hidden1 layer
44                vhat1 = list(np.dot(What1,img))
45                vhat1.append(-1)
46                vhat1 = np.array(vhat1,dtype = float)
47                y1 = activation(vhat1).T
48                # activate hidden2 layer
49                vhat2 = list(np.dot(What2,y1))
50                vhat2.append(-1)

```

```

51     vhat2 = np.array(vhat2,dtype = float)
52     y2 = activation(vhat2).T
53     # activate output layer
54     v = np.dot(W,y2)
55     o = activation(v)
56     trainouts.append(o)#-----
57     e = (d-o)/1900
58     errorsForEpochs.append(e)
59     # back propagate
60     # back propagate the output layer
61     Tprime = activation(v,deriv = True)*np.identity(2,dtype=float)
62     gradient = (np.dot(Tprime,e))
63     # back propagate the hidden2 layer
64     Tprimehat2 = activation(vhat2,deriv = True)*np.identity(hidden2No+1,dtype =
65     gradienthat2 = (np.dot((np.dot(Tprimehat2,W.T)),gradient))
66     # back propagate the hidden1 layer
67     Tprimehat1 = activation(vhat1,deriv = True)*np.identity(hidden1No+1,dtype =
68     gradienthat1 = (np.dot((np.dot(Tprimehat1,What2.T)),gradienthat2[:-1]))
69     # mini batch updates
70
71     if iterateno%miniBatchno== 0:
72         Wupdate = (np.dot(gradient.reshape(-1,1),y2.reshape(1,-1)))
73         What2update = (np.dot(gradienthat2.reshape(-1,1),y1.reshape(-1,1).T)[: -
74         What1update = (np.dot(gradienthat1.reshape(-1,1),img.reshape(-1,1).T)[:
75
76         if not(iterateno==0 and epoch==0):#The first iteration of the first epo
77         # batch updates
78             Wold = W
79             W +=normalize(learningrate*(Wupdate))/miniBatchno
80             W = normalize(W)
81             W +=alpha*((W-Wold))
82             W = normalize(W)
83
84             What2old = What2
85             What2+=normalize(learningrate*(What2update))/miniBatchno
86             What2 = normalize(What2)
87             What2+=alpha*((What2-What2old))
88             What2 = normalize(What2)
89
90             What1old = What1
91             What1+= normalize(learningrate*(What1update))/miniBatchno
92             What1 = normalize(What1)
93             What1+= alpha*((What1-What1old))
94             What1 = normalize(What1)
95     else:
96         Wupdate += (np.dot(gradient.reshape(-1,1),y2.reshape(1,-1)))
97         What2update += (np.dot(gradienthat2.reshape(-1,1),y1.reshape(1,-1))[:-1
98         What1update += (np.dot(gradienthat1.reshape(-1,1),img.reshape(-1,1).T)[
99     lbls = np.array(lbls)
100    trainouts = np.array(trainouts)
101    classificationError = classError(trainouts,lbls)
102    errorsForClassification.append(classificationError)
103
104    errorsForEpochs = np.array(errorsForEpochs)
105    mse=calculatemse(errorsForEpochs,o).reshape(-1,1)

```

```

106     errorsFormse.append(float(mse))
107
108     WtT = W.T[:-1].T
109     Wthat2T = (What2.T[:-1]).T
110     Wthat1T = (What1.T[:-1]).T
111     errorsForEpochsTest = list()
112     testouts = list()
113     for i in range(testdata.shape[0]):
114         # forward propagate
115         imgT = testdata[i][:-1]
116         lblT = testlbls[i]
117         #
118         dT = np.zeros(2)
119         lblT = trainlbls[i]
120         dT[int(lblT)] = 1
121
122         vhat1T = np.dot(Wthat1T, imgT).reshape(1, -1)
123         y1T = activation(vhat1T).T
124
125         vhat2T = np.dot(Wthat2T, y1T).reshape(1, -1)
126         y2T = activation(vhat2T).T
127
128         vT = np.dot(WtT, y2T)
129         oT = activation(vT)
130
131         eT = (dT - oT) / 1900
132
133         testouts.append(oT)
134         errorsForEpochsTest.append(eT)
135     testouts = np.array(testouts)
136     testclassifications = classError(testouts, testlbls)
137     errorsForTestClassifications.append(testclassifications)
138     # print(" Test classificaiton error: "+str(testclassifications))
139     if testclassifications > 75:
140         break
141     errorsForEpochsTest = np.array(errorsForEpochsTest)
142     msetest = calculatemse(errorsForEpochsTest, o).reshape(-1, 1)
143     errorsForTestmse.append(float(mse))
144 # Adjusting weight matrices by de-concetanating the bias term
145
146 print("Train mean squared errors for epochs")
147 errorsFormse = np.array(errorsFormse)
148 plt.plot(errorsFormse)
149 plt.show()
150 print("Train percentage classification errors for epochs")
151 errorsForClassification = np.array(errorsForClassification)
152 plt.plot(errorsForClassification)
153 plt.show()
154 print("Test mean squared errors for epochs")
155 errorsForTestmse = np.array(errorsForTestmse)
156 plt.plot(errorsForTestmse)
157 plt.show()
158 print("Test percentage classification errors for epochs")
159 errorsForTestClassifications = np.array(errorsForTestClassifications)
160 plt.plot(errorsForTestClassifications)

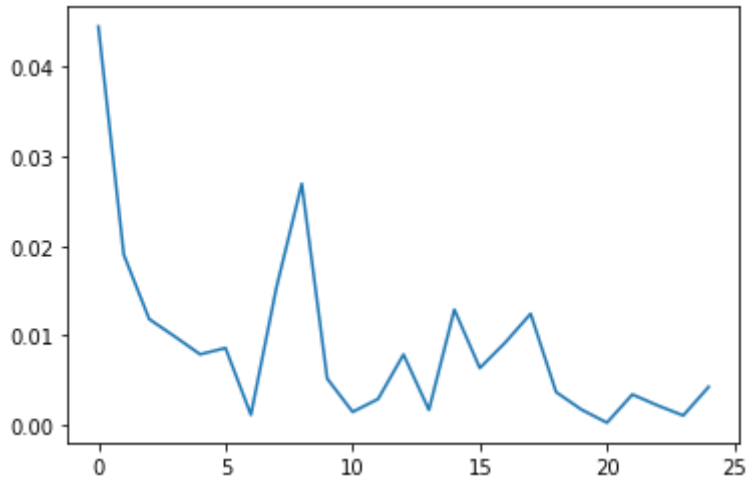
```

```
161     plt.show()
162     print("Maximum accuracy is obtained at epoch:"+str(np.argmax(errorsForTestClassific

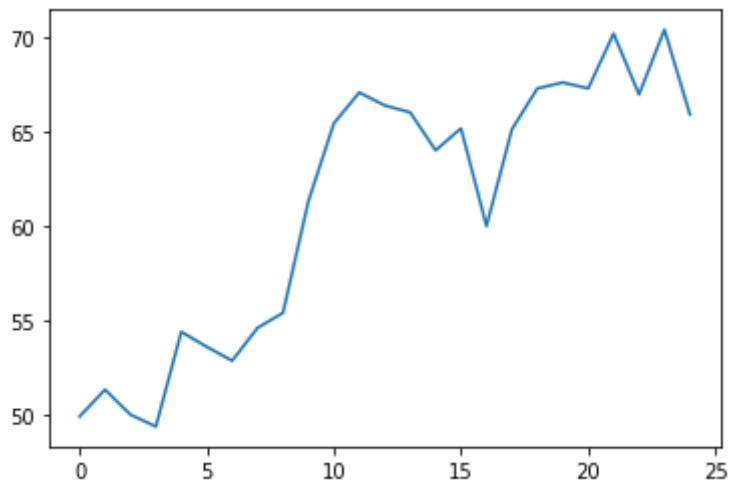
1 # hiddenNo1,hiddenNo2,epochno,minibatchno,learningrate
2 runNetwork2(300,100,600,40,0.2,0)
3
4 runNetwork2(100,75,100,60,0.15,0.1)
5
6 runNetwork2(150,50,300,100,0.13,0.15)
```



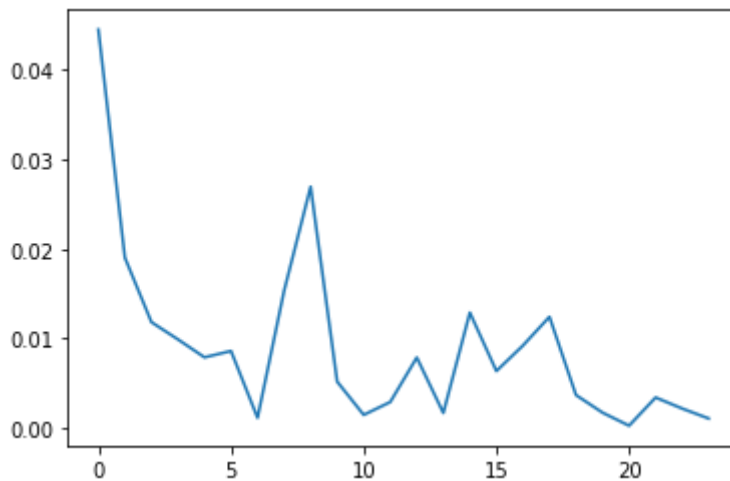
Hidden1 layer number:300 Hidden2 layer number:100 Learning rate:0.2 Epoch number:600
Training started. Time: Sun Nov 17 18:58:31 2019
Epoch no: 24 in 600 Time: Sun Nov 17 19:01:06 2019 Train mean squared errors for epoch



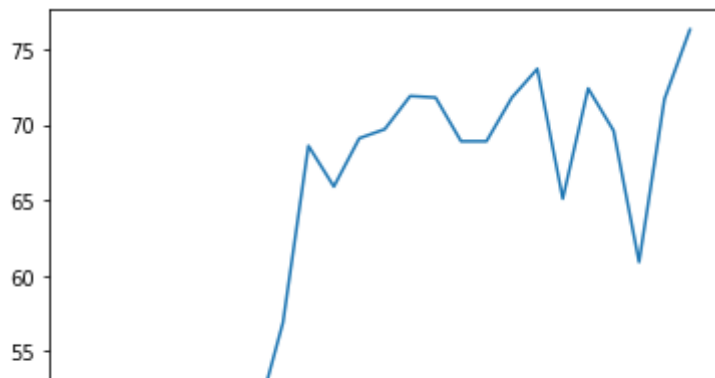
Train percentage classification errors for epochs

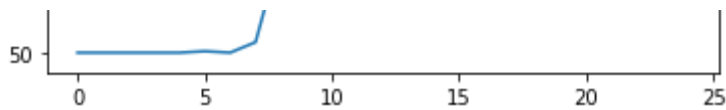


Test mean squared errors for epochs



Test percentage classification errors for epochs



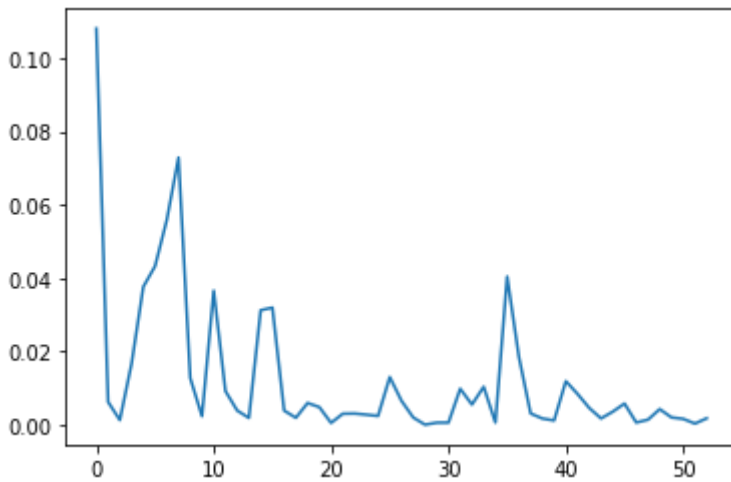


Maximum accuracy is obtained at epoch:24 as 76.3%

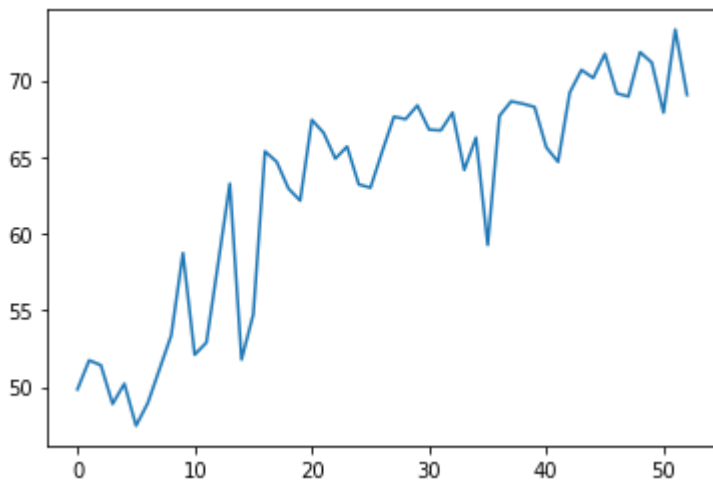
Hidden1 layer number:100 Hidden2 layer number:75 Learning rate:0.15 Epoch number:100

Training started. Time: Sun Nov 17 19:01:14 2019

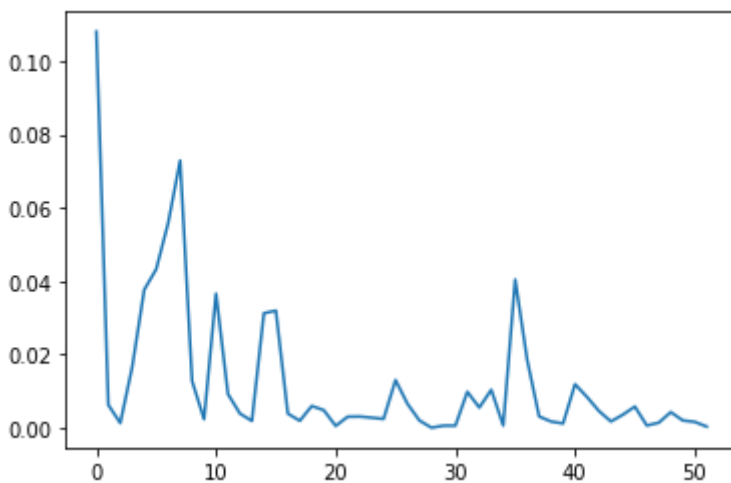
Epoch no: 52 in 100 Time: Sun Nov 17 19:03:36 2019 Train mean squared errors for epoch



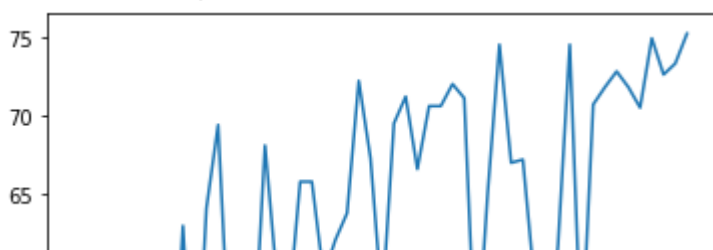
Train percentage classification errors for epochs

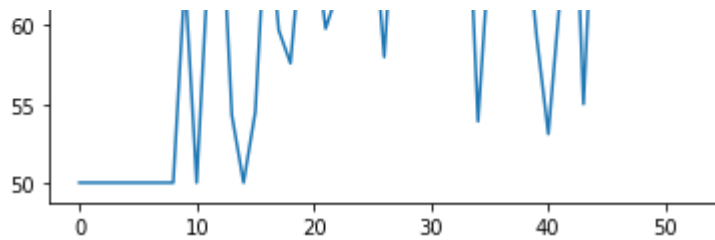


Test mean squared errors for epochs



Test percentage classification errors for epochs



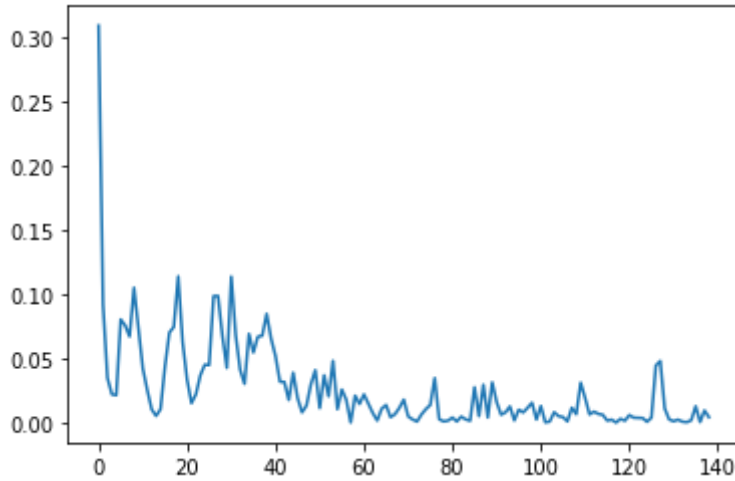


Maximum accuracy is obtained at epoch:52 as 75.2%

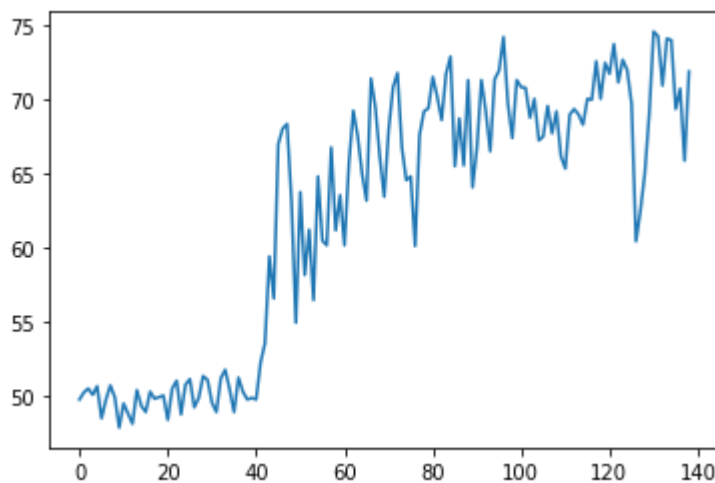
Hidden1 layer number:150 Hidden2 layer number:50 Learning rate:0.13 Epoch number:300

Training started. Time: Sun Nov 17 19:03:41 2019

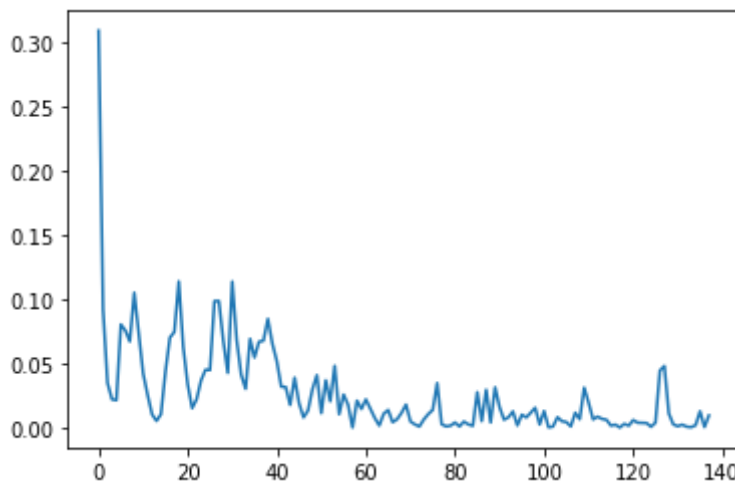
Epoch no: 138 in 300 Time: Sun Nov 17 19:11:14 2019 Train mean squared errors for epoch



Train percentage classification errors for epochs

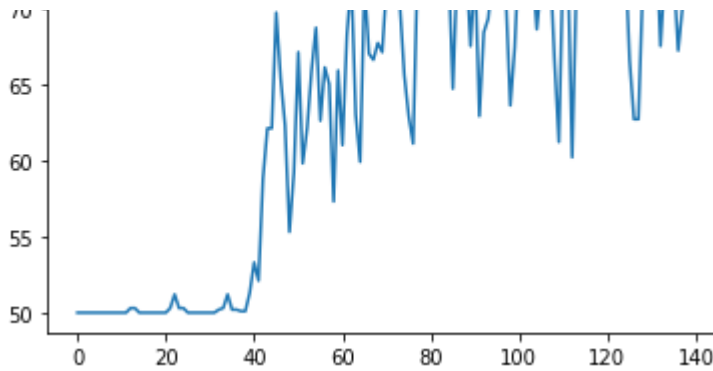


Test mean squared errors for epochs



Test percentage classification errors for epochs





Maximum accuracy is obtained at epoch:138 as 76.0%

```

1 def formdata2():
2     Q2data2 = ("/content/assign2_data2.h5")
3     trainx,traind,valx,valid,testx,testd,words = organizeData2(Q2data2)
4     trainx = np.array(trainx)
5     trainx,traind = shuffle3(trainx,traind)
6     print("Data for train")
7     onehot = []
8     for k in range(250):
9         i = k+1
10        index = i+1
11        onehot = formonehot(trainx,i)
12        if i == 1:
13            onehots = onehot
14        else:
15            onehots = np.concatenate((onehots, onehot),axis =1)
16        title = ("\r"+"Handling dataset, creating embedding matrix: "+str(i/250*100)+
17        sys.stdout.write(title +" Time: " +time.ctime())
18        sys.stdout.flush()
19        time.sleep(1)
20
21    print("\nData for validation")
22    onehotv = []
23    for k in range(250):
24        i = k+1
25        index = i+1
26        onehotv = formonehot(valx,i)
27        if i == 1:
28            onehotsv = onehotv
29        else:
30            onehotsv = np.concatenate((onehotsv, onehotv),axis =1)
31        title = ("\r"+"Handling dataset, creating embedding matrix: "+str(i/250*100)+
32        sys.stdout.write(title +" Time: " +time.ctime())
33        sys.stdout.flush()
34        time.sleep(1)
35
36    print("\nData for test")
37    onehott =[]
38    for k in range(250):
39        i = k+1
40        indext= i+1
41        onehott = formonehot(testx,i)
42
43        if i == 1:

```

```

44         onehotst = onehott
45     else:
46         onehotst = np.concatenate((onehotst, onehott),axis =1)
47     title = ("\r"+"Handling dataset, creating embedding matrix: "+str(i/250*100)+
48     sys.stdout.write(title +" Time: " +time.ctime())
49     sys.stdout.flush()
50     time.sleep(1)
51
52     onehots = onehots.T
53     onehotstv = onehotstv.T
54     onehotst = onehotst.T
55
56
57     return onehots,onehotstv,onehotst,traind,valid,testd,words
58 def formdata3():
59     print("Constructing onehots")
60     onehots,onehotstv,onehotst,traind,valid,testd,words = formdata2()
61     firstWord = list()
62     secondWord=list()
63     thirdWord = list()
64     print("Constructing word arrays")
65     print("Train data")
66     for i in range(int(len(onehots)/3)):
67         firstWordIndex = 3*i
68         secondWordIndex = 3*i+1
69         thirdWordIndex = 3*i+2
70
71         firstWord.append(onehots[firstWordIndex])
72         secondWord.append(onehots[secondWordIndex])
73         thirdWord.append(onehots[thirdWordIndex])
74     firstWord = np.array(firstWord).T
75     secondWord = np.array(secondWord).T
76     thirdWord = np.array(thirdWord).T
77     print("Validation data")
78     firstWordv = list()
79     secondWordv=list()
80     thirdWordv = list()
81     for i in range(int(len(onehotstv)/3)):
82         firstWordIndex = 3*i
83         secondWordIndex = 3*i+1
84         thirdWordIndex = 3*i+2
85
86         firstWordv.append(onehotstv[firstWordIndex])
87         secondWordv.append(onehotstv[secondWordIndex])
88         thirdWordv.append(onehotstv[thirdWordIndex])
89     firstWordv = np.array(firstWordv).T
90     secondWordv = np.array(secondWordv).T
91     thirdWordv = np.array(thirdWordv).T
92     print("Test data")
93     firstWordt = list()
94     secondWordt=list()
95     thirdWordt = list()
96     for i in range(int(len(onehotst)/3)):
97         firstWordIndex = 3*i
98         secondWordIndex = 3*i+1

```

```

99         thirdWordIndex = 3*i+2
100
101         firstWordt.append(onehotst[firstWordIndex])
102         secondWordt.append(onehotst[secondWordIndex])
103         thirdWordt.append(onehotst[thirdWordIndex])
104     firstWordt = np.array(firstWordt).T
105     secondWordt = np.array(secondWordt).T
106     thirdWordt = np.array(thirdWordt).T
107     return firstWord,secondWord,thirdWord,firstWordv,secondWordv,thirdWordv,firstWordt,
108
109     # firstWord,secondWord,thirdWord,firstWordv,secondWordv,thirdWordv,firstWordt,secon

1 firstWord,secondWord,thirdWord,firstWordv,secondWordv,thirdWordv,firstWordt,secondWordt

```

↳ Constructing onehots

Data for train

Handling dataset, creating embedding matrix: 100.0% Time: Sun Nov 17 14:42:16 2019

Data for validation

Handling dataset, creating embedding matrix: 100.0% Time: Sun Nov 17 14:46:42 2019

Data for test

Handling dataset, creating embedding matrix: 100.0% Time: Sun Nov 17 14:51:08 2019C

Train data

Validation data

Test data

1

```

1 print(words[26])
2
3 # print(firstWord.shape)
4 # print(firstWordv.shape)
5 # print(firstWordt.shape)
6
7 # print(firstWord[0])
8 # print(firstWordt[0])

```

↳ b'only'

```

1 # ff = list()
2 # ss = list()
3 # tt = list()
4 # dd = list()
5 # oo = list()
6 def runNetwork3(D,hidden2No,epochno,miniBatchno,learningrate,alpha):
7     print("Hidden1 layer number:"+str(D)+" Hidden2 layer number:"+str(hidden2No)+" Lear
8     # handling array constructions
9
10
11     What1= np.random.normal(0, 1/10,(int(D),250))
12
13     hidden1No = D
14     What2= np.random.normal(0, 1/hidden1No,(hidden2No,3*D))
15     bhat2 = np.random.normal(0, 1/1/hidden1No,(hidden2No)).reshape(-1,1)
16     What2 =np.hstack((What2, bhat2))
17

```

```

1 /
18 W= np.random.normal(0, 1/hidden2No,(250,hidden2No))
19 b = np.random.normal(0, 1/hidden2No,(250)).reshape(1,-1)
20 W =np.hstack((W, b.T))
21
22
23 errorsFormse = list()
24 errorsForClassification = list()
25 errorsForTestClassifications = list()
26 errorsForTestmse = list()
27 errorsForTestClassificationst = list()
28 errorsForTestmset = list()
29
30
31 # Start training process
32 for epoch in range(epochno):
33     errorsForEpochs = list()
34     # display epoch index and current time
35     if epoch==0:
36         print("Training started.   Time: "+time.ctime())
37         title = ("\r\r"+"Epoch no: "+str(epoch)+" in "+str(epochno))
38         sys.stdout.write(title +" Time: " +time.ctime())
39         sys.stdout.flush()
40         time.sleep(1)
41         trainouts = list()#-----
42         lbls = list()
43         lblst = list()
44         for iterateno in range(372500):
45             # forward propagate
46             # handling array constructions
47             i = iterateno
48
49             firstw = firstWord[i]
50             secondw = secondWord[i]
51             thirdw = thirdWord[i]
52
53             first = (np.dot(What1,firstw))
54             second = (np.dot(What1,secondw))
55             third = (np.dot(What1,thirdw))
56
57
58             d = np.zeros(250)
59             lbl = traind[i]-1
60             lbls.append(lbl)
61             d[int(lbl)] = 1
62             # activate hidden layer
63             vhat1 = np.append(np.concatenate((first,second,third)),([[ -1]]))
64             vhat1 = np.array(vhat1)
65
66             y1 = activation(vhat1).T
67             # activate hidden2 layer
68             vhat2 = np.append(np.dot(What2,y1),([[ -1]]))
69             y2 = activation(vhat2).T
70             # activate output layer
71             v = np.dot(W,y2)
72             a = softmax(v)

```