**ID5030 :: MNIST Neural Net**

Varun V (AE14B050)

**(a) Code:**

The parameter part of code is editable for desired layers. We assign them as given in the question.

Batch size variable should be changed depending on the system ram and processing power. Smaller batch size has lower memory and computation requirements while increasing the time required linearly. Larger batch sizes allow for faster convergence but need more ram and computational speeds.

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Sat Mar 17 18:44:23 2018
5. @author: Varun
6. """
8. **import** numpy as np
9. **import** pandas as pd
10. **import** matplotlib.pyplot as plt
11. **import** tensorflow as tf
12. **import** os
14. **from** sklearn.preprocessing **import** MinMaxScaler
15. **from** tensorflow.examples.tutorials.mnist **import** input\_data
16. **from** tensorflow.contrib.layers **import** flatten
17. """
18. Setting working device to CPU as my GPU got fried
19. """
20. os.environ['CUDA\_VISIBLE\_DEVICES'] = '-1'
22. """
23. Importing Data with flattened images
24. """
25. data = input\_data.read\_data\_sets("MNIST\_data/")
26. X\_train, y\_train           = data.train.images, data.train.labels
27. X\_test, y\_test             = data.test.images, data.test.labels
28. X\_valid, y\_valid             = data.validation.images, data.validation.labels
30. """
31. Normalized scaling to (0,1)
32. """
33. scaler = MinMaxScaler()
34. X\_train = scaler.fit\_transform(X\_train)
35. X\_test = scaler.transform(X\_test)
36. X\_valid = scaler.transform(X\_valid)
38. """
39. Reshaping to 28x28 images
40. """
41. X\_train = X\_train.reshape(X\_train.shape[0],28,28,1)
42. X\_test = X\_test.reshape(X\_test.shape[0],28,28,1)
43. X\_valid = X\_valid.reshape(X\_valid.shape[0],28,28,1)
45. """
46. Padding with zeros, images are now 32x32
47. """
48. X\_train      = np.pad(X\_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
49. X\_valid = np.pad(X\_valid, ((0,0),(2,2),(2,2),(0,0)), 'constant')
50. X\_test       = np.pad(X\_test, ((0,0),(2,2),(2,2),(0,0)), 'constant')
52. """
53. placeholders and one hot encoding for cross entropy calculation
54. """
55. x = tf.placeholder(tf.float32, (None, 32, 32, 1))
56. y = tf.placeholder(tf.int32, (None))
57. onehoty = tf.one\_hot(y, 10)
59. """
60. %%PARAMETERS
61. """
62. filters1 = 6           #number of filters in 1st convolution
63. filterframe = [5,5]    #frame of the convolution filters
64. filters2 = 16          #number of filters in 2nd convolution
65. maxpoolframe = [2,2]   #maxpooling frame size
66. maxpoolstrides = [2,2] #stride of maxpooling frame
67. fconshape1 = 120       #neurons in 1st fully connected layer
68. fconshape2 = 84        #neurons in 2nd fully connected layer
69. classes = 10           #final neurons for logits
70. a = 0.1                #learning rate
71. maxitr = 3             #total number of iterations over the data set
72. batchsize = 128        #batchsize for each gradient descent run

75. """
76. Network Architecture
77. """
78. conv1 = tf.layers.conv2d(x, filters1, filterframe, activation = tf.nn.relu)
79. pool1 = tf.layers.max\_pooling2d(conv1,maxpoolframe,maxpoolstrides)
80. conv2 = tf.layers.conv2d(pool1, filters2, filterframe, activation = tf.nn.relu)
81. pool2 = tf.layers.max\_pooling2d(conv2,maxpoolframe,maxpoolstrides)
82. fcon0 = flatten(pool2)
83. fcon1 = tf.layers.dense(fcon0,fconshape1,tf.nn.relu)
84. fcon2 = tf.layers.dense(fcon1,fconshape2,tf.nn.relu)
85. logits = tf.layers.dense(fcon2, classes)
87. """
88. Loss function and optimizer algorithm definition
89. """
90. entropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits = logits, labels = onehoty)
91. loss = tf.reduce\_mean(entropy)
92. algo = tf.train.GradientDescentOptimizer(a).minimize(loss)
94. """
95. Predictions and accuracy helper function
96. """
98. checkprediction = tf.equal(tf.argmax(tf.nn.softmax(logits), 1), tf.argmax(onehoty, 1))
99. **def** calc\_accuracy(X,Y,batchsize):
100. total\_accurate = 0
101. **for** start **in** range(0,X.shape[0],batchsize):
102. end = start+batchsize
103. accurate = np.sum(sess.run(checkprediction, feed\_dict = {x:X[start:end], y:Y[start:end]}))
104. total\_accurate += accurate
105. **return**  100\*total\_accurate/X.shape[0]
107. """
108. Intitiate session
109. """
110. sess = tf.Session()
111. sess.run(tf.global\_variables\_initializer())
113. """
114. Batch gradient descent
115. """
116. **for** i **in** range(maxitr):
117. **for** start **in** range(0,X\_train.shape[0],batchsize):
118. end = start+batchsize
119. sess.run(algo,feed\_dict = {x:X\_train[start:end],y:y\_train[start:end]})
121. **print**(i,calc\_accuracy(X\_train,y\_train,batchsize))
123. """
124. Calculating final prediction accuracies and printing the same
125. """
126. accuracy = calc\_accuracy(X\_train,y\_train,batchsize)
127. accuracy\_v =calc\_accuracy(X\_valid,y\_valid,batchsize)
128. accuracy\_t =calc\_accuracy(X\_test,y\_test,batchsize)
129. **print**(accuracy,accuracy\_v,accuracy\_t)

**(b)Results:**

* We calculate the final prediction accuracies on each data set given to us.
* The results are quite satisfactory for the given 3 iterations over the data set with a batchsize of 128.
* Higher number of iterations can be used but the improvement is very low.
* Increasing the batch size require more number of iterations over the dataset but the overall number of times the gradient descent algorithm is called reduces with increased batch size.

|  |  |
| --- | --- |
| **Prediction On** | **Accuracy(%)** |
| Training | 99.01 |
| Validation | 99.52 |
| Testing | 99.13 |