dl-lab-assignment-2

December 12, 2023

Q1.Implement DL with numpy random data.

[1]:
```
#Importing libraries
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Activation,Dense
```

[2]:
```
#Generating a dummy data set for showing the working of DLL-NN training␣
 ↪data-8000 rows 15 columns for x and 3000 rows 1 column for y testing␣
 ↪data-3000 rows 15 columns for x and 3000 rows 1 column for y.
x_train=np.random.random((8000,15))
#print(x_train)
y_train=np.random.randint(2,size=(8000,1))
#print(y_train)
x_test=np.random.random((3000,15))
y_test=np.random.randint(2,size=(3000,1))
```

[13]:
```
#Defining the model architecture with NN layers No of neurons=64,32,16,8,4,2,1.
model=Sequential()
#Layer 1
model.add(Dense(64,input_dim=15,activation='relu'))
#Layer 2
model.add(Dense(32,activation='relu'))
#Layer3
model.add(Dense(16,activation='relu'))
#Layer4
model.add(Dense(8,activation='relu'))
#Layer5
model.add(Dense(4,activation='relu'))
#Layer6
model.add(Dense(2,activation='relu'))
#Layer7
model.add(Dense(1,activation='sigmoid'))
```

[14]:
```
!pip install visualkeras
```

```
Requirement already satisfied: visualkeras in /usr/local/lib/python3.10/dist-
```

```
packages (0.0.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from visualkeras) (9.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-
packages (from visualkeras) (1.23.5)
Requirement already satisfied: aggdraw>=1.3.11 in
/usr/local/lib/python3.10/dist-packages (from visualkeras) (1.3.18)
```

[15]:
```
import visualkeras
visualkeras.layered_view(model)
```

[15]:

[16]:
```
#Configure the model
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])
```

[17]:
```
#Define Validation data
x_val=np.random.random((3000,15))
y_val=np.random.randint(2,size=(3000,1))
```

[18]:
```
#Training the data with batch-size=64 and epoch=3.
model.fit(x_train,y_train,batch_size=64,epochs=3,validation_data=(x_val,y_val))
```

```
Epoch 1/3
125/125 [==============================] - 2s 7ms/step - loss: 0.6931 -
accuracy: 0.4988 - val_loss: 0.6932 - val_accuracy: 0.4933
Epoch 2/3
125/125 [==============================] - 0s 4ms/step - loss: 0.6930 -
accuracy: 0.5056 - val_loss: 0.6933 - val_accuracy: 0.4920
Epoch 3/3
125/125 [==============================] - 0s 4ms/step - loss: 0.6932 -
accuracy: 0.5035 - val_loss: 0.6933 - val_accuracy: 0.4917
```

[18]:
```
<keras.src.callbacks.History at 0x78e62dd29ff0>
```

[19]:
```
#Model Evaluation with test data
print(model.evaluate(x_test,y_test))
```

```
94/94 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy:
0.4937
[0.6932805776596069, 0.4936666786670685]
```

[20]:
```
#Model Prediction with test data
print(model.predict(x_test))
```

```
94/94 [==============================] - 0s 2ms/step
[[0.5039798]
 [0.5039798]
 [0.5039798]
 …
 [0.5039798]
 [0.5039798]
 [0.5039798]]
```

Q2.Implement image classification using DL with MNIST data.

[21]:
```
from keras.datasets import mnist
```

[22]:
```
(train_images,train_labels),(test_images,test_labels)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

[23]:
```
train_images.shape
```

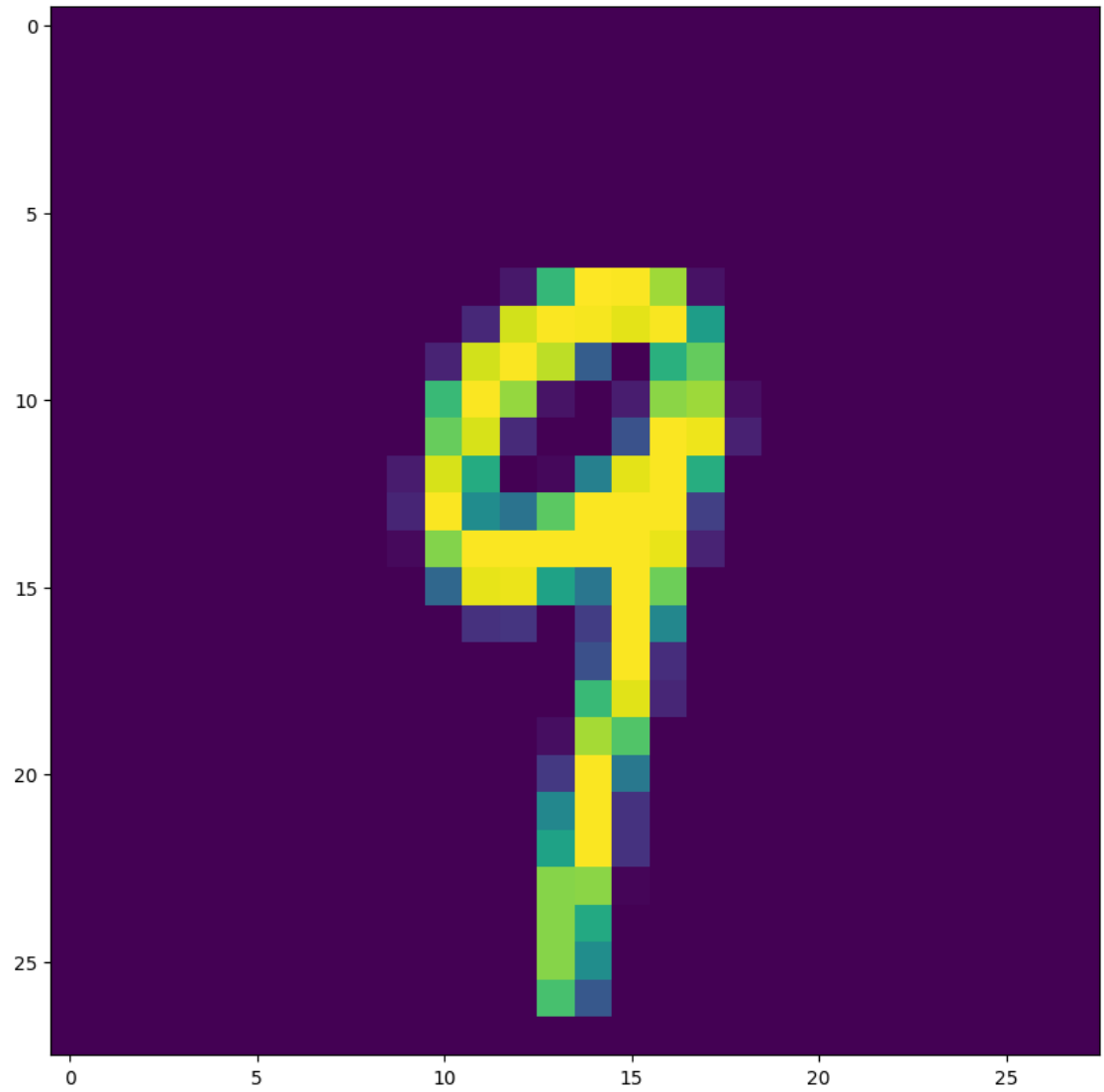
[23]: (60000, 28, 28)

[24]:
```
test_images.shape
```

[24]: (10000, 28, 28)

[25]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,10))
plt.imshow(train_images[600])
print(train_labels[600])
```

```
9
```



[26]:
```
#Make the data in the keras accepting format.
print(train_images[1])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  51 159 253
  159  50   0   0   0   0   0   0   0   0]
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0  48 238 252 252
 252 237   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0  54 227 253 252 239
 233 252  57   6   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0  10  60 224 252 253 252 202
  84 252 253 122   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0 163 252 252 252 253 252 252
  96 189 253 167   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0  51 238 253 253 190 114 253 228
  47  79 255 168   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0  48 238 252 252 179  12  75 121  21
   0   0 253 243  50   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  38 165 253 233 208  84   0   0   0   0
   0   0 253 252 165   0   0   0   0   0]
[  0   0   0   0   0   0   0   7 178 252 240  71  19  28   0   0   0   0
   0   0 253 252 195   0   0   0   0   0]
[  0   0   0   0   0   0   0  57 252 252  63   0   0   0   0   0   0   0
   0   0 253 252 195   0   0   0   0   0]
[  0   0   0   0   0   0   0 198 253 190   0   0   0   0   0   0   0   0
   0   0 255 253 196   0   0   0   0   0]
[  0   0   0   0   0   0  76 246 252 112   0   0   0   0   0   0   0   0
   0   0 253 252 148   0   0   0   0   0]
[  0   0   0   0   0   0  85 252 230  25   0   0   0   0   0   0   0   0
   7 135 253 186  12   0   0   0   0   0]
[  0   0   0   0   0   0  85 252 223   0   0   0   0   0   0   0   0   7
 131 252 225  71   0   0   0   0   0   0]
[  0   0   0   0   0   0  85 252 145   0   0   0   0   0   0   0  48 165
 252 173   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  86 253 225   0   0   0   0   0   0 114 238 253
 162   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  85 252 249 146  48  29  85 178 225 253 223 167
  56   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  85 252 252 252 229 215 252 252 252 196 130   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  28 199 252 252 253 252 252 233 145   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0  25 128 252 253 252 141  37   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

[27]:
```
train_images=train_images.reshape((60000,28*28))
train_images=train_images.astype('float32')/255
```

[28]:
```
test_images=test_images.reshape((10000,28*28))
test_images=test_images.astype('float32')/255
```

[29]:
```
from keras.models import Sequential
from keras.layers import Dense,Activation
from keras import regularizers
```

[30]:
```
model1=Sequential()
model1.add(Dense(512,activation='relu',input_shape=(28*28,)))
model1.add(Dense(10,activation='softmax'))
```

[31]:
```
#Confifure the model
model1.
 ↪compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[32]:
```
#Preprocessing the labels.
from keras.utils import to_categorical
train_labels=to_categorical(train_labels)
test_labels=to_categorical(test_labels)
```

[37]:
```
#Training the model with data
model1.fit(train_images,train_labels,epochs=10,batch_size=256)
```

```
Epoch 1/10
235/235 [==============================] - 4s 15ms/step - loss: 0.0211 -
accuracy: 0.9953
Epoch 2/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0163 -
accuracy: 0.9969
Epoch 3/10
235/235 [==============================] - 5s 20ms/step - loss: 0.0140 -
accuracy: 0.9975
Epoch 4/10
235/235 [==============================] - 4s 18ms/step - loss: 0.0112 -
accuracy: 0.9982
Epoch 5/10
235/235 [==============================] - 4s 15ms/step - loss: 0.0088 -
accuracy: 0.9991
Epoch 6/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0072 -
accuracy: 0.9992
Epoch 7/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0064 -
accuracy: 0.9994
```

```
Epoch 8/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0047 -
accuracy: 0.9997
Epoch 9/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0040 -
accuracy: 0.9998
Epoch 10/10
235/235 [==============================] - 5s 20ms/step - loss: 0.0034 -
accuracy: 0.9998
```

[37]:
```
<keras.src.callbacks.History at 0x78e62dce2860>
```

[38]:
```
loss1,acc1=model1.evaluate(test_images,test_labels)
print("Accuracy:",acc1*100)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0572 -
accuracy: 0.9831
Accuracy: 98.30999970436096
```

[39]:
```
model1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 512)               401920

 dense_15 (Dense)            (None, 10)                5130

=================================================================
Total params: 407050 (1.55 MB)
Trainable params: 407050 (1.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[40]:
```
test1_labels=model1.predict(test_images)
```

```
313/313 [==============================] - 1s 3ms/step
```