Topics in Deep Learning
Home Assignment 2
Jaykumar Nariya(1116571)

Task1: GAN

In [ ]:

```
#pip install tensorflow==1.14
#install for colab
```

In [187]:

```
#import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf

%matplotlib inline
#import dataset from tensorflow
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data", one_hot=True)
```

```
Extracting MNIST_data\train-images-idx3-ubyte.gz
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
```

In [188]:

```
# split data into train and test
x_train, y_train = mnist.train.images, mnist.train.labels
x_test, y_test = mnist.test.images, mnist.test.labels

#Reshape dataset for model
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

In [189]:

```
x_train.shape
```

Out[189]:

```
(55000, 28, 28, 1)
```

In [190]:

```
#trainable finction for model layer freez and unfreez
def modifiy_trainable(net, val):
    net.trainable = val
    for l in net.layers:
        l.trainable = val
```

In [191]:

```
#import libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, BatchNormaliz
ation, Activation, Reshape, Input, Dropout,UpSampling2D,LeakyReLU
from keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from keras.utils import np_utils
#from keras.layers.convolutional import UpSampling2D
#from keras.layers.advanced_activations import LeakyReLU
from tensorflow.keras.models import Model
from keras.backend import image_data_format
```

In [193]:

```
#Define generator learning rate
Gen_learningrate = 1e-4
#generator first input layer
gen_input = Input(shape= [100])
# Define generatore sequential model
generator = Sequential()
generator.add(Dense(200*14*14, kernel_initializer='glorot_normal'))
generator.add(BatchNormalization())
generator.add(Activation('relu'))
generator.add(Reshape([14,14,200]))
generator.add(UpSampling2D(size=(2,2)))
generator.add(Conv2D(100,(3,3), padding='same', kernel_initializer='glorot_uniform'))
generator.add(BatchNormalization())
generator.add(Activation('relu'))
generator.add(Conv2D(50,(3,3), padding='same', kernel_initializer='glorot_uniform'))
generator.add(BatchNormalization())
generator.add(Activation('relu'))
generator.add(Conv2D(1,(1,1), padding='same', kernel_initializer='glorot_uniform'))
generator.add(Activation('sigmoid'))
```

In [194]:

```
#combine input layer and other layer in one model
gen_model = Model(gen_input,generator(gen_input))
```

In [195]:

```
gen_model.summary()
```

```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        [(None, 100)]             0
_________________________________________________________________
sequential_9 (Sequential)    (None, 28, 28, 1)         4341801
=================================================================
Total params: 4,341,801
Trainable params: 4,263,101
Non-trainable params: 78,700
_________________________________________________________________
```

In [196]:

```
#compile generator
gen_model.compile(loss='binary_crossentropy', optimizer=Adam(lr=Gen_learningrate))
```

In [197]:

```
#define discremenato learning rate
declearningrate = 1e-3
#discremenator input layer
dec_input=Input(shape=(28,28,1))

# define discremenatore sequential model
discriminator = Sequential()
discriminator.add(Conv2D(256,(5,5),strides=(2,2), padding = 'same'))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.25))
discriminator.add(Conv2D(512,(5,5),strides=(2,2), padding = 'same'))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.25))
discriminator.add(Flatten())
discriminator.add(Dense(256))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.25))
discriminator.add(Dense(2, activation='softmax'))

#combine input layer and other layers

des_model = Model(dec_input,discriminator(dec_input))
```

In [198]:

```
des_model.summary()
```

```
Model: "model_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        [(None, 28, 28, 1)]       0
_________________________________________________________________
sequential_10 (Sequential)   (None, 2)                 9707266
=================================================================
Total params: 9,707,266
Trainable params: 9,707,266
Non-trainable params: 0
_________________________________________________________________
```

In [199]:

```
# compile discremenator model
des_model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=declearningrate))
```

In [200]:

```
#combine generator and discremenator in GAN
Generatorinput = Input(shape=[100])

Gan_op = gen_model(Generatorinput)
des_model.trainable = False
Discreminatoroutput = des_model(Gan_op)
GAN = Model(Generatorinput, Discreminatoroutput)
GAN.compile(loss='categorical_crossentropy', optimizer=Adam(lr=Gen_learningrate))
GAN.summary()
```

```
Model: "model_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_15 (InputLayer)        [(None, 100)]             0
_________________________________________________________________
model_11 (Model)             (None, 28, 28, 1)         4341801
_________________________________________________________________
model_12 (Model)             (None, 2)                 9707266
=================================================================
Total params: 14,049,067
Trainable params: 4,263,101
Non-trainable params: 9,785,966
_________________________________________________________________
```

In [201]:

```
from IPython import display
import os
```

In [213]:

```
def plot_loss(losses, step):# loss plotting function
    display.clear_output(wait=True)
    display.display(plt.gcf())
    plt.figure(figsize=(10,8))
    plt.plot(losses["d"], label="discriminative loss")
    plt.plot(losses["g"], label="generative loss")
    plt.legend()
    plt.savefig(os.path.join('generatedimages', 'loss_{}.png'.format(step)))
    plt.show()


def plot_gan(step, n_ex=16, dim=(4,4), figsize=(4,4)):# Generated image showing and sav
ing function
    noise = np.random.uniform(0,1,size=[n_ex,100])
    generated_images = generator.predict(noise)

    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0], dim[1], i+1)
        img = generated_images[i, :, :, 0]
        plt.imshow(img)
        plt.axis('off')
    plt.tight_layout()
    plt.savefig(os.path.join('generatedimages', 'gann_{}.png'.format(step)))
    plt.show()
```

In [214]:

```
import random
```

In [215]:

```
# taking random samples
n_train = 1000
#train_idx = np.random.sample()
train_idx = random.sample(range(0, x_train.shape[0]), n_train)
print(type(train_idx))
print(len(train_idx))
print(train_idx[:10])

X_T = x_train[train_idx, :, :, :]
X_T.shape
```

```
<class 'list'>
1000
[20318, 33560, 53312, 1622, 25705, 3783, 51823, 32497, 15534, 12229]
```

Out[215]:

```
(1000, 28, 28, 1)
```

In [216]:

```
# noise for generator
noise_gen = np.random.uniform(0,1,size=[X_T.shape[0],100])
generated_images = generator.predict(noise_gen)
X = np.concatenate((X_T, generated_images))
X.shape
```

Out[216]:

```
(2000, 28, 28, 1)
```

In [217]:

```
n = X_T.shape[0]
y = np.zeros([2*n,2])
y[:n, 1] = 1
y[n:, 0] = 1
print(y.shape)
print(y)
```

```
(2000, 2)
[[0. 1.]
 [0. 1.]
 [0. 1.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

In [218]:

```
#cheking loss for noise
modifiy_trainable(des_model, True)
print(X.shape)
print(y.shape)
des_model.fit(X,y, epochs=1, batch_size=128)
y_pred = des_model.predict(X)
```

```
(2000, 28, 28, 1)
(2000, 2)
2000/2000 [==============================] - ETA: 9s - loss: 0.115 - ETA:
9s - loss: 0.111 - ETA: 8s - loss: 0.157 - ETA: 7s - loss: 0.138 - ETA: 7s
- loss: 0.146 - ETA: 6s - loss: 0.134 - ETA: 5s - loss: 0.125 - ETA: 5s -
loss: 0.113 - ETA: 4s - loss: 0.105 - ETA: 3s - loss: 0.099 - ETA: 3s - lo
ss: 0.095 - ETA: 2s - loss: 0.089 - ETA: 1s - loss: 0.087 - ETA: 1s - los
s: 0.086 - ETA: 0s - loss: 0.083 - 11s 5ms/sample - loss: 0.0813
```

In [219]:

```
print(y_pred.shape)
y_pred_idx = np.argmax(y_pred, axis=1)
print(y_pred_idx.shape)
print(y.shape)
y_idx = np.argmax(y, axis =1)
print("y_idx[10:] :", y_idx[10:])
print("y_idx[1500:] :", y_idx[1500:])
diff = y_idx - y_pred_idx
print(diff.shape)
print(diff[:10])
n_total = y.shape[0]
n_rig = (diff==0).sum()
acc = float(n_rig*100.0/n_total)

print("Accuracy : {} of {} right".format(acc, n_rig, n_total))
```

```
(2000, 2)
(2000,)
(2000, 2)
y_idx[10:] : [1 1 1 ... 0 0 0]
y_idx[1500:] : [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 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 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 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]
(2000,)
[0 0 0 0 0 0 0 0 0 1]
Accuracy : 98.4 of 1968 right
```

In [220]:

```
# intialize noise
losses = {"d":[], "g":[]}
```

In [221]:

```
from tqdm import tqdm
```

In [222]:

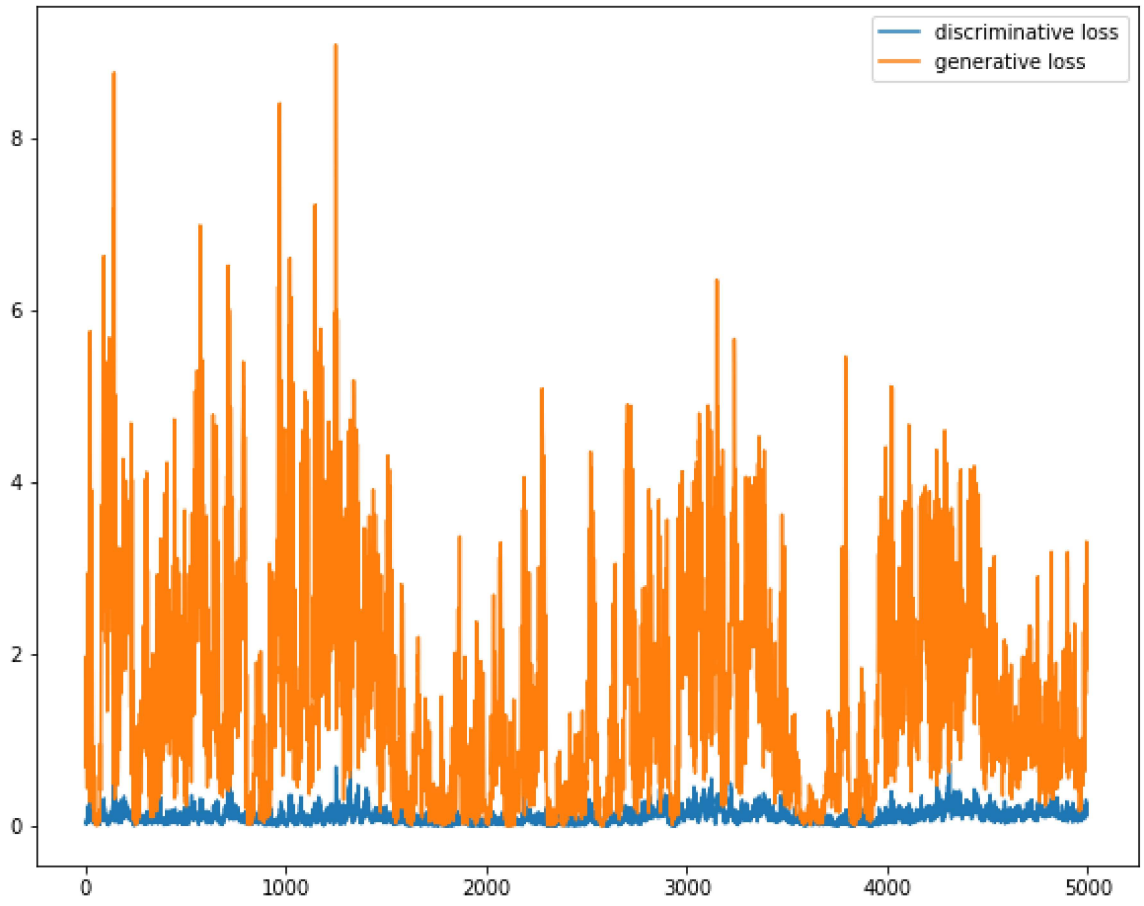
```
#deine train function for model
def train_for_n(epoch=5000, plt_frq=25, batch_size=32):
    for e in tqdm(range(epoch)):

        image_batch = x_train[np.random.randint(0, x_train.shape[0], size=batch_size),
:, :, :]
        noise_gen = np.random.uniform(0,1,size=[batch_size, 100])
        generated_images = gen_model.predict(noise_gen)

        #concat image batch and generated images
        X = np.concatenate((image_batch, generated_images))
        y = np.zeros([2*batch_size,2])
        y[:batch_size,1] = 1
        y[batch_size:, 0] = 1
        #plot loss
        modifiy_trainable(des_model, True)
        d_loss = des_model.train_on_batch(X,y)
        losses["d"].append(d_loss)
        #training noise
        noise_tr = np.random.uniform(0,1,size=[batch_size,100])
        y2 = np.zeros([batch_size,2])
        y2[:,1] =1

        modifiy_trainable(des_model,False)
        g_loss = GAN.train_on_batch(noise_tr,y2)
        losses["g"].append(g_loss)

        if e%plt_frq == plt_frq -1:
            plot_loss(losses,e)
            plot_gan(e)
```

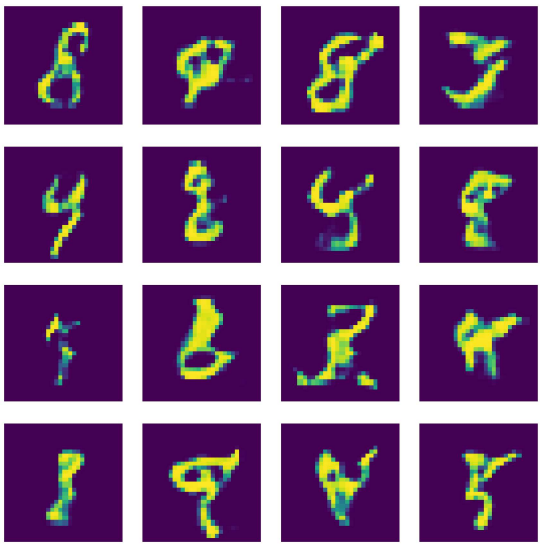
In [223]:

```
#call training function for 5000 epochs with 64 batch size
train_for_n(epoch=5000, plt_frq=25, batch_size=64)
```

```
<Figure size 432x288 with 0 Axes>
```

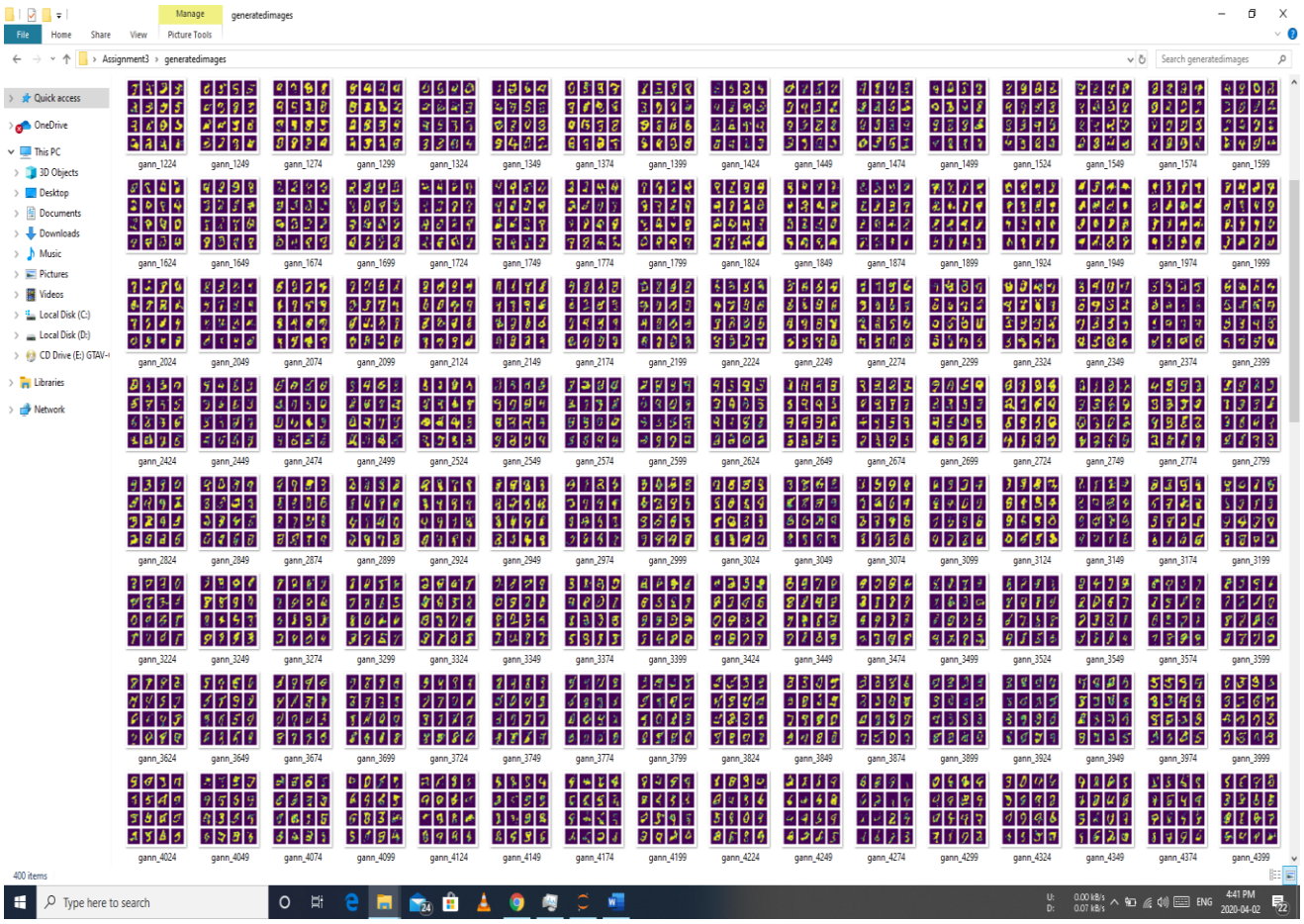
```
<Figure size 432x288 with 0 Axes>
```





```
100%|
      | 5000/5000 [1:52:48<00:00,  2.53s/it]
```

OP:


generatedimages
Assignment3 › generatedimages
Search generatedimages
Quick access
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Local Disk (D:)
CD Drive (E:) GTAV-I
Libraries
Network
400 items
Type here to search
ENG
4:41 PM
2020-04-02

Task2: Prediction of Time series dataset using Lstm

In [81]:

```
#pip install tensorflow==1.
#install if want to run in colab
```

```
Requirement already satisfied: tensorflow==1.14 in /usr/local/lib/python3.
6/dist-packages (1.14.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.
6/dist-packages (from tensorflow==1.14) (1.1.0)
Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dis
t-packages (from tensorflow==1.14) (0.3.3)
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/pytho
n3.6/dist-packages (from tensorflow==1.14) (0.2.0)
Requirement already satisfied: keras-applications>=1.0.6 in /usr/local/li
b/python3.6/dist-packages (from tensorflow==1.14) (1.0.8)
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.
6/dist-packages (from tensorflow==1.14) (3.10.0)
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/di
st-packages (from tensorflow==1.14) (0.8.1)
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/
dist-packages (from tensorflow==1.14) (0.9.0)
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/d
ist-packages (from tensorflow==1.14) (1.27.2)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dis
t-packages (from tensorflow==1.14) (0.34.2)
Requirement already satisfied: tensorflow-estimator<1.15.0rc0,>=1.14.0rc0
in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.14.0)
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/d
ist-packages (from tensorflow==1.14) (1.12.1)
Requirement already satisfied: numpy<2.0,>=1.14.5 in /usr/local/lib/python
3.6/dist-packages (from tensorflow==1.14) (1.18.2)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dis
t-packages (from tensorflow==1.14) (1.12.0)
Requirement already satisfied: tensorboard<1.15.0,>=1.14.0 in /usr/local/l
ib/python3.6/dist-packages (from tensorflow==1.14) (1.14.0)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/li
b/python3.6/dist-packages (from tensorflow==1.14) (1.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packa
ges (from keras-applications>=1.0.6->tensorflow==1.14) (2.10.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist
-packages (from protobuf>=3.6.1->tensorflow==1.14) (46.0.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python
3.6/dist-packages (from tensorboard<1.15.0,>=1.14.0->tensorflow==1.14) (1.
0.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.
6/dist-packages (from tensorboard<1.15.0,>=1.14.0->tensorflow==1.14) (3.2.
1)
```

In [0]:

```
#import libraries
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense,LSTM,Dropout,Activation
```

In [0]:

```
# read dataset
df = pd.read_csv("AirPassengers.csv")
```

In [84]:

```
#show first five line of dataset
df.head()
```

Out[84]:

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

In [0]:

```
# save Month column as index of time sequence
df.Month = pd.to_datetime(df.Month)
df= df.set_index('Month')
```

In [86]:

```
#show dtaframe
df.head()
```

Out[86]:

	#Passengers
Month	
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

In [0]:

```
#scaling data using MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
scal = MinMaxScaler()
scal.fit(df)
scaled = scal.transform(df)
```

In [0]:

```
#from sklearn.model_selection import train_test_split

#train, test =train_test_split(scaled,test_size=0.4, random_state=0)
```

In [0]:

```
# split data set into training and testing
train_size = 0.60
train_size = int(len(scaled) * train_size)
test_size = len(scaled) - train_size
train, test = scaled[0:train_size, :], scaled[train_size:len(scaled), :]
```

In [0]:

```
window_size = 1
#devide training data into feature and label
data_X, data_Y = [], []
for i in range(len(train) - window_size - 1):
    a = train[i:(i + window_size), 0]
    data_X.append(a)
    data_Y.append(train[i + window_size, 0])
```

In [0]:

```
import numpy as np
#convert training data into numpy array
x_train = np.array(data_X)
y_train = np.array(data_Y)
```

In [0]:

```
#devide testing data into feature and label
test_X, test_Y = [], []
for i in range(len(test) - window_size - 1):
    a = test[i:(i + window_size), 0]
    test_X.append(a)
    test_Y.append(test[i + window_size, 0])
```

In [0]:

```
#convert testing data into numpy array
x_test = np.array(test_X)
y_test = np.array(test_Y)
```

In [0]:

```
#reshape data
x_train = np.reshape(x_train, (x_train.shape[0], 1, x_train.shape[1]))
x_test = np.reshape(x_test, (x_test.shape[0], 1, x_test.shape[1]))
```

In [95]:

```
#define sequential regression model
model = Sequential()
model.add(LSTM(input_shape = (1,window_size),
               units = 4,
               return_sequences = True))
model.add(Dropout(0.5))
model.add(LSTM(256))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation("linear"))
model.compile(loss = "mse",
              optimizer = "adam")
model.summary()
#fir model
model.fit(x_train, y_train, epochs = 100, batch_size = 1, verbose = 2)
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_12 (LSTM)               (None, 1, 4)              96        
_________________________________________________________________
dropout_9 (Dropout)          (None, 1, 4)              0         
_________________________________________________________________
lstm_13 (LSTM)               (None, 256)               267264    
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 257       
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0         
=================================================================
Total params: 267,617
Trainable params: 267,617
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
 - 2s - loss: 0.0185
Epoch 2/100
 - 1s - loss: 0.0113
Epoch 3/100
 - 1s - loss: 0.0095
Epoch 4/100
 - 1s - loss: 0.0099
Epoch 5/100
 - 1s - loss: 0.0087
Epoch 6/100
 - 1s - loss: 0.0079
Epoch 7/100
 - 1s - loss: 0.0080
Epoch 8/100
 - 1s - loss: 0.0052
Epoch 9/100
 - 1s - loss: 0.0061
Epoch 10/100
 - 1s - loss: 0.0049
Epoch 11/100
 - 1s - loss: 0.0038
Epoch 12/100
 - 1s - loss: 0.0048
Epoch 13/100
 - 1s - loss: 0.0029
Epoch 14/100
 - 1s - loss: 0.0045
Epoch 15/100
 - 1s - loss: 0.0051
Epoch 16/100
 - 1s - loss: 0.0046
Epoch 17/100
 - 1s - loss: 0.0048
Epoch 18/100
 - 1s - loss: 0.0051
Epoch 19/100
 - 1s - loss: 0.0043
Epoch 20/100
 - 1s - loss: 0.0045
Epoch 21/100
```

```
 - 1s - loss: 0.0050
Epoch 22/100
 - 1s - loss: 0.0034
Epoch 23/100
 - 1s - loss: 0.0047
Epoch 24/100
 - 1s - loss: 0.0046
Epoch 25/100
 - 1s - loss: 0.0065
Epoch 26/100
 - 1s - loss: 0.0049
Epoch 27/100
 - 1s - loss: 0.0050
Epoch 28/100
 - 1s - loss: 0.0043
Epoch 29/100
 - 1s - loss: 0.0044
Epoch 30/100
 - 1s - loss: 0.0035
Epoch 31/100
 - 1s - loss: 0.0032
Epoch 32/100
 - 1s - loss: 0.0045
Epoch 33/100
 - 1s - loss: 0.0063
Epoch 34/100
 - 1s - loss: 0.0048
Epoch 35/100
 - 1s - loss: 0.0032
Epoch 36/100
 - 1s - loss: 0.0045
Epoch 37/100
 - 1s - loss: 0.0040
Epoch 38/100
 - 1s - loss: 0.0067
Epoch 39/100
 - 1s - loss: 0.0033
Epoch 40/100
 - 1s - loss: 0.0048
Epoch 41/100
 - 1s - loss: 0.0044
Epoch 42/100
 - 1s - loss: 0.0048
Epoch 43/100
 - 1s - loss: 0.0062
Epoch 44/100
 - 1s - loss: 0.0050
Epoch 45/100
 - 1s - loss: 0.0055
Epoch 46/100
 - 1s - loss: 0.0047
Epoch 47/100
 - 1s - loss: 0.0047
Epoch 48/100
 - 1s - loss: 0.0052
Epoch 49/100
 - 1s - loss: 0.0059
Epoch 50/100
 - 1s - loss: 0.0045
Epoch 51/100
 - 1s - loss: 0.0040
```

```
Epoch 52/100
 - 1s - loss: 0.0037
Epoch 53/100
 - 1s - loss: 0.0057
Epoch 54/100
 - 1s - loss: 0.0049
Epoch 55/100
 - 1s - loss: 0.0066
Epoch 56/100
 - 1s - loss: 0.0043
Epoch 57/100
 - 1s - loss: 0.0043
Epoch 58/100
 - 1s - loss: 0.0054
Epoch 59/100
 - 1s - loss: 0.0045
Epoch 60/100
 - 1s - loss: 0.0032
Epoch 61/100
 - 1s - loss: 0.0044
Epoch 62/100
 - 1s - loss: 0.0053
Epoch 63/100
 - 1s - loss: 0.0038
Epoch 64/100
 - 1s - loss: 0.0039
Epoch 65/100
 - 1s - loss: 0.0050
Epoch 66/100
 - 1s - loss: 0.0048
Epoch 67/100
 - 1s - loss: 0.0047
Epoch 68/100
 - 1s - loss: 0.0040
Epoch 69/100
 - 1s - loss: 0.0038
Epoch 70/100
 - 1s - loss: 0.0045
Epoch 71/100
 - 1s - loss: 0.0041
Epoch 72/100
 - 1s - loss: 0.0037
Epoch 73/100
 - 1s - loss: 0.0046
Epoch 74/100
 - 1s - loss: 0.0033
Epoch 75/100
 - 1s - loss: 0.0045
Epoch 76/100
 - 1s - loss: 0.0044
Epoch 77/100
 - 1s - loss: 0.0045
Epoch 78/100
 - 1s - loss: 0.0038
Epoch 79/100
 - 1s - loss: 0.0054
Epoch 80/100
 - 1s - loss: 0.0047
Epoch 81/100
 - 1s - loss: 0.0034
Epoch 82/100
```

```
 - 1s - loss: 0.0036
Epoch 83/100
 - 1s - loss: 0.0035
Epoch 84/100
 - 1s - loss: 0.0040
Epoch 85/100
 - 1s - loss: 0.0045
Epoch 86/100
 - 1s - loss: 0.0054
Epoch 87/100
 - 1s - loss: 0.0049
Epoch 88/100
 - 1s - loss: 0.0044
Epoch 89/100
 - 1s - loss: 0.0037
Epoch 90/100
 - 1s - loss: 0.0039
Epoch 91/100
 - 1s - loss: 0.0057
Epoch 92/100
 - 1s - loss: 0.0053
Epoch 93/100
 - 1s - loss: 0.0044
Epoch 94/100
 - 1s - loss: 0.0059
Epoch 95/100
 - 1s - loss: 0.0042
Epoch 96/100
 - 1s - loss: 0.0051
Epoch 97/100
 - 1s - loss: 0.0046
Epoch 98/100
 - 1s - loss: 0.0043
Epoch 99/100
 - 1s - loss: 0.0044
Epoch 100/100
 - 1s - loss: 0.0040
```

Out[95]:

```
<keras.callbacks.History at 0x7fb45c7dc588>
```

In [96]:

```
import math
from sklearn.metrics import mean_squared_error
train_predict = scal.inverse_transform(model2.predict(x_train))
# Prepare y_train data to also be on the original scale for interpretability.
d = scal.inverse_transform([y_train])
#calculat rmse training
rmse_train = math.sqrt(mean_squared_error(d[0], train_predict[:, 0]))
print("Training data score: %.2f RMSE" % rmse_train)
```

```
Training data score: 24.94 RMSE
```

In [97]:

```
    test_predict = scal.inverse_transform(model2.predict(x_test))
    # Prepare y_test data to also be on the original scale for interpretability.
    d = scal.inverse_transform([y_test])
    #calculate #rmse testing
    rmse_test = math.sqrt(mean_squared_error(d[0], test_predict[:, 0]))
    print("Training data score: %.2f RMSE" % rmse_test)
```

```
Training data score: 70.77 RMSE
```

In [98]:

```
import matplotlib.pyplot as plt

# Start with training predictions.
train_predict_plot = np.empty_like(scaled)
train_predict_plot[:, :] = np.nan
train_predict_plot[window_size:len(train_predict) + window_size, :] = train_predict

# Add test predictions.
test_predict_plot = np.empty_like(scaled)
test_predict_plot[:, :] = np.nan
test_predict_plot[len(train_predict) + (window_size * 2) + 1:len(scaled) - 1, :] = test
_predict

# Create the plot.
plt.figure(figsize = (15, 5))
plt.plot(scal.inverse_transform(scaled), label = "True data")
plt.plot(train_predict_plot, label = "Training set prediction")
plt.plot(test_predict_plot, label = "Test set prediction")
plt.xlabel("Months")
plt.ylabel("1000 International Airline Passengers")
plt.title("Comparison true vs. predicted training / test")
plt.legend()
plt.show()
```