ITS group works.

In your Jupyter notebook, organize the program. Write the line's heading and description. Execute each cell and examine the output. At our next class meeting, we will discuss the meaning of a few terms, as we are currently focused solely on the application.

The data will automatically download from this line *(X_train, y_train), (X_test, y_test) = mnist.load_data()*

1. CLASSIFICATION OF NUMBERS 0-9

```
from keras.datasets import mnist

import matplotlib.pyplot as plt

from keras.models import Sequential

from keras.layers import Dense, Conv2D, Flatten

from keras.utils import to_categorical


#download mnist data and split into train and test sets

(X_train, y_train), (X_test, y_test) = mnist.load_data()


f1 = plt.figure(1)

plt.imshow(X_train[0])

f2 = plt.figure(2)

plt.imshow(X_train[1])

plt.show()


#check image shape and data count

print(X_train[0].shape, len(X_train))

print(X_train[0].shape, len(X_test))


#reshape data to fit model

X_train = X_train.reshape(len(X_train),28,28,1)
```

```
X_test = X_test.reshape(len(X_test),28,28,1)

#One-hot encode target column
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]

#Create model
model = Sequential()

#Add Input CNN Layer
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))

#Add second CNN Layer
model.add(Conv2D(32, kernel_size=3, activation='relu'))

#Add the fully connected layer
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

#Compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)

#predict first 6 images in the test set
model.predict(X_test[:6])
```

#actual results for first 6 images in the test set

y_test[:6]

ITCB and ITS group works

2. TIME SERIES DATA

In your Jupyter notebook, organize the program. Write the line's heading and description. Execute each cell and examine the output. At our next class meeting, we will discuss the meaning of a few terms, as we are currently focused solely on the application.

International Airline Passengers prediction problem. This is a problem where, given a year and a month, the task is to predict the number of international airline passengers in units of 1,000. The data ranges from January 1949 to December 1960, or 12 years, with 144 observations.

Check on folder I have attached a file named *AirPassengers.csv'*

```
import numpy as np
import matplotlib.pyplot as plt
from pandas import read_csv
import math

from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# load the dataset
dataframe = read_csv('Downloads/AirPassengers.csv', usecols=[1])
plt.plot(dataframe)

#Convert pandas dataframe to numpy array
dataset = dataframe.values
```

```
dataset = dataset.astype('float32') #COnvert values to float

# Normalization is optional but recommended for neural network as certain
# activation functions are sensitive to magnitude of numbers.
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1)) #Also try QuantileTransformer
dataset = scaler.fit_transform(dataset)

#We cannot use random way of splitting dataset into train and test as
#the sequence of events is important for time series.
#So let us take first 60% values for train and the remaining 1/3 for testing
# split into train and test sets
train_size = int(len(dataset) * 0.66)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]


#Use TimeseriesGenerator to organize training data into the right format
#We can use a generator instead.....
from keras.preprocessing.sequence import TimeseriesGenerator # Generates batches for sequence data
seq_size = length =  10
batch_size = 1
train_generator = TimeseriesGenerator(train,train,length=length,batch_size=batch_size)
print("Total number of samples in the original training data = ", len(train)) # 95
print("Total number of samples in the generated data = ", len(train_generator)) # 55
#With length 40 it generated 55 samples, each of length 40 (by using data of length 95)

# print a couple of samples...
x, y = train_generator[0]
```

```
#Also generate validation data
validation_generator = TimeseriesGenerator(test, test, length=length ,batch_size=batch_size)



#Input dimensions are... (N x seq_size)
num_features = 1 #Univariate example

################################################
#Check SimpleRNN before moving on to LSTM
# print('Build SimpleRNN model...')
# # create and fit pure, simple RNN
# model = Sequential()
# model.add(SimpleRNN(64, input_shape=(length, num_features), activation='relu')) #12
# model.add(Dense(1))
# model.compile(loss='mean_squared_error', optimizer='adam', metrics = ['acc'])
# print(model.summary())

#####################################################
#################################################################
#LSTM single layer with 50 units
# model = Sequential()

# model.add(LSTM(50, input_shape=(length, num_features)))
# model.add(Dense(1))
# model.compile(optimizer = 'adam', loss='mse')
# ################################################
# ########################################################
```

```
#Stacked LSTM with 1 hidden dense layer
# reshape input to be [samples, time steps, features]
#trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
#testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
#
model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(length, num_features)))
model.add(LSTM(50, activation='relu'))
#model.add(Dense(32))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

model.summary()
print('Train...')
################################################

#Bidirectional LSTM
# reshape input to be [samples, time steps, features]
#trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
#testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
#
##For some sequence forecasting problems we may need LSTM to learn
## sequence in both forward and backward directions
#from keras.layers import Bidirectional
#model = Sequential()
#model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(None, seq_size)))
#model.add(Dense(1))
#model.compile(optimizer='adam', loss='mean_squared_error')
#model.summary()
```

```
#print('Train...')


#############################################################

#ConvLSTM

#The layer expects input as a sequence of two-dimensional images,

#therefore the shape of input data must be: [samples, timesteps, rows, columns, features]


# trainX = trainX.reshape((trainX.shape[0], 1, 1, 1, seq_size))

# testX = testX.reshape((testX.shape[0], 1, 1, 1, seq_size))


# model = Sequential()

# model.add(ConvLSTM2D(filters=64, kernel_size=(1,1), activation='relu', input_shape=(1, 1, 1, seq_size)))

# model.add(Flatten())

# model.add(Dense(32))

# model.add(Dense(1))

# model.compile(optimizer='adam', loss='mean_squared_error')

# model.summary()

#print('Train...')




##########################################

model.fit_generator(generator=train_generator, verbose=2, epochs=100, validation_data=validation_generator)


################################################

trainPredict = model.predict(train_generator)

testPredict = model.predict(validation_generator)
```

```
trainPredict = scaler.inverse_transform(trainPredict)
trainY_inverse = scaler.inverse_transform(train)
testPredict = scaler.inverse_transform(testPredict)
testY_inverse = scaler.inverse_transform(test)


# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY_inverse[length:], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mean_squared_error(testY_inverse[length:], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting
#we must shift the predictions so that they align on the x-axis with the original dataset.
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[length:len(trainPredict)+length, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
#testPredictPlot[len(trainPredict)+(seq_size*2)-1:len(dataset)-1, :] = testPredict
testPredictPlot[len(train)+(length)-1:len(dataset)-1, :] = testPredict


# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
```

```
plt.plot(trainPredictPlot)

plt.plot(testPredictPlot)

plt.show()
```

3. EXTRACTION OF LETTERS FROM A BOOK

 # Download text file from: http://www.gutenberg.org/ebooks/236
Check on the folder I have attached the file named the_jungle_book.txt

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from keras.optimizers import RMSprop
import numpy as np
import random
import sys

#LOAD TEXT
#Save notepad as UTF-8 (select from dropdown during saving)
filename = "Downloads/the_jungle_book.txt"
raw_text = open(filename, 'r', encoding='utf-8').read()
raw_text = raw_text.lower()
print(raw_text[0:1000])

#CLEAN TEXT
#Remove numbers
raw_text = ''.join(c for c in raw_text if not c.isdigit())

#How many total characters do we have in our training text?
chars = sorted(list(set(raw_text))) #List of every character

#Character sequences must be encoded as integers.
#Each unique character will be assigned an integer value.
#Create a dictionary of characters mapped to integer values
char_to_int = dict((c, i) for i, c in enumerate(chars))

#Do the reverse so we can print our predictions in characters and not integers
int_to_char = dict((i, c) for i, c in enumerate(chars))

# summarize the data
n_chars = len(raw_text)
n_vocab = len(chars)
print("Total Characters in the text; corpus length: ", n_chars)
```

```
print("Total Vocab: ", n_vocab)


#########################
#Now that we have characters we can create input/output sequences for training
#Remember that for LSTM input and output can be sequences... hence the term seq2seq


seq_length = 60  #Length of each input sequence
step = 10   #Instead of moving 1 letter at a time, try skipping a few.
sentences = []    # X values (Sentences)
next_chars = []   # Y values. The character that follows the sentence defined as X
for i in range(0, n_chars - seq_length, step):  #step=1 means each sentence is offset just by a single letter
   sentences.append(raw_text[i: i + seq_length])  #Sequence in
   next_chars.append(raw_text[i + seq_length])  #Sequence out
n_patterns = len(sentences)
print('Number of sequences:', n_patterns)

#Have a look at sentences and next_chars to see the continuity...
##############################

#Just like time series, X is the sequence / sentence and y is the next value
#that comes after the sentence...

# reshape input to be [samples, time steps, features]

#time steps = sequence length
#features = numbers of characters in our vocab (n_vocab)
#Vectorize all sentences: there are n_patterns sentences.
#For each sentence we have n_vocab characters available for seq_length
#Vectorization returns a vector for all sentences indicating the presence or absence
#of a character.

x = np.zeros((len(sentences), seq_length, n_vocab), dtype=np.bool)
y = np.zeros((len(sentences), n_vocab), dtype=np.bool)
for i, sentence in enumerate(sentences):
   for t, char in enumerate(sentence):
      x[i, t, char_to_int[char]] = 1
   y[i, char_to_int[next_chars[i]]] = 1

print(x.shape)
print(y.shape)
```

```
print(y[0:10])

##################################################
#Basic model with one LSTM
# build the model: a single LSTM

model = Sequential()
model.add(LSTM(128, input_shape=(seq_length, n_vocab)))
model.add(Dense(n_vocab, activation='softmax'))

optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
model.summary()

########################################
# Deeper model woth 2 LSTM
#To stack LSTM layers, we need to change the configuration of the prior
#LSTM layer to output a 3D array as input for the subsequent layer.
#We can do this by setting the return_sequences argument on the layer to True
#(defaults to False). This will return one output for each input time step and provide a 3D array.
#Below is the same example as above with return_sequences=True.

#model = Sequential()
#model.add(LSTM(128, input_shape=(seq_length, n_vocab), return_sequences=True))
#model.add(Dropout(0.2))
#model.add(LSTM(128))
#model.add(Dropout(0.2))
#model.add(Dense(n_vocab, activation='softmax'))

#optimizer = RMSprop(lr=0.01)
#model.compile(loss='categorical_crossentropy', optimizer=optimizer)
#model.summary()


################


# define the checkpoint
# from keras.callbacks import ModelCheckpoint

# filepath="saved_weights/saved_weights-{epoch:02d}-{loss:.4f}.hdf5"
# checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True,
mode='min')
```

```
# callbacks_list = [checkpoint]


# Fit the model

history = model.fit(x, y,
        batch_size=128,
        epochs=10)

model.save('my_saved_weights_jungle_book_50epochs.h5')
################################################################################

from matplotlib import pyplot as plt
#plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.title('Training loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


################################################################################
####################################
#Generate characters
#We must provide a sequence of seq_lenth as input to start the generation process

#The prediction results is probabilities for each of the 48 characters at a specific
#point in sequence. Let us pick the one with max probability and print it out.
#Writing our own softmax function....

def sample(preds):
   preds = np.asarray(preds).astype('float64')
   preds = np.log(preds)
   exp_preds = np.exp(preds) #exp of log (x), isn't this same as x??
   preds = exp_preds / np.sum(exp_preds)
   probas = np.random.multinomial(1, preds, 1)
   return np.argmax(probas)



#Prediction
```

```
# load the network weights
filename = "my_saved_weights_jungle_book_50epochs.h5"
model.load_weights(filename)

#Pick a random sentence from the text as seed.
start_index = random.randint(0, n_chars - seq_length - 1)

#Initiate generated text and keep adding new predictions and print them out
generated = ''
sentence = raw_text[start_index: start_index + seq_length]
generated += sentence

print('----- Seed for our text prediction: "' + sentence + '"')
#sys.stdout.write(generated)


for i in range(400):   # Number of characters including spaces
    x_pred = np.zeros((1, seq_length, n_vocab))
    for t, char in enumerate(sentence):
        x_pred[0, t, char_to_int[char]] = 1.

    preds = model.predict(x_pred, verbose=0)[0]
    next_index = sample(preds)
    next_char = int_to_char[next_index]

    generated += next_char
    sentence = sentence[1:] + next_char

    sys.stdout.write(next_char)
    sys.stdout.flush()
print()
```