Table of Contents

[Recurrent Neural Networks (RNNs) 1](#_Toc118809079)

[Sequential Data 1](#_Toc118809080)

[Learning Sequences 2](#_Toc118809081)

[Hidden State 2](#_Toc118809082)

[Long Short-Term Memory Networks 2](#_Toc118809083)

[Multilayer LSTM’s 5](#_Toc118809084)

[Long Short-Term Memory 7](#_Toc118809085)

[Tokenizing and Sequencing Sentences 8](#_Toc118809086)

[Long Short-Term Memory Terminology 9](#_Toc118809087)

[Embedding Layer 9](#_Toc118809088)

[LSTM Layers 10](#_Toc118809089)

[Further Improvements 13](#_Toc118809090)

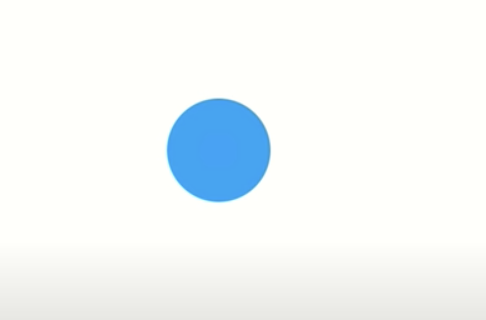
## Recurrent Neural Networks (RNNs)

This initial discussion about RNNs is based on the video at <https://www.youtube.com/watch?v=LHXXI4-IEns>

### Sequential Data

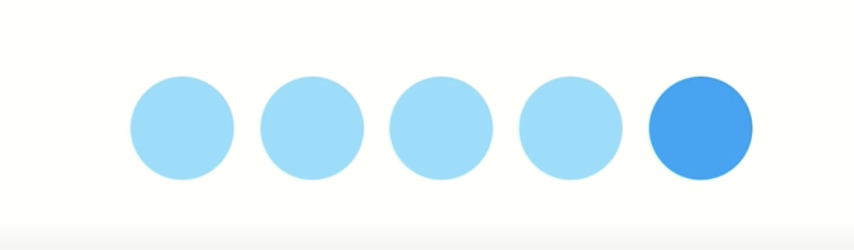
Recurrent neural networks help to explain sequential data. For example, when trying to predict the direction of a moving ball, it is difficult to know with only the current position. Figure 1 shows a snapshot of the ball at its current position.

Figure : Current Snapshot of a Ball

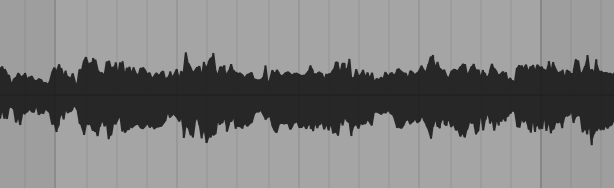


To improve your accuracy, you need where the ball has been. Figure 2 shows several snapshots of the ball’s historical positions in succession.

Figure : Sequence of Ball Positions



Sequence data can be obtained and stored in many different ways. For example, audio files contain wave data which is sequential.



Words also follow a sequence. If you only read the words “the beach” you have no context about why “the beach” is mentioned. A more complete sentence with leading words provides more detail.

Mocha and Cocoa went to the beach.

In all sequential cases if we understand what occurred earlier we can better understand the current context.

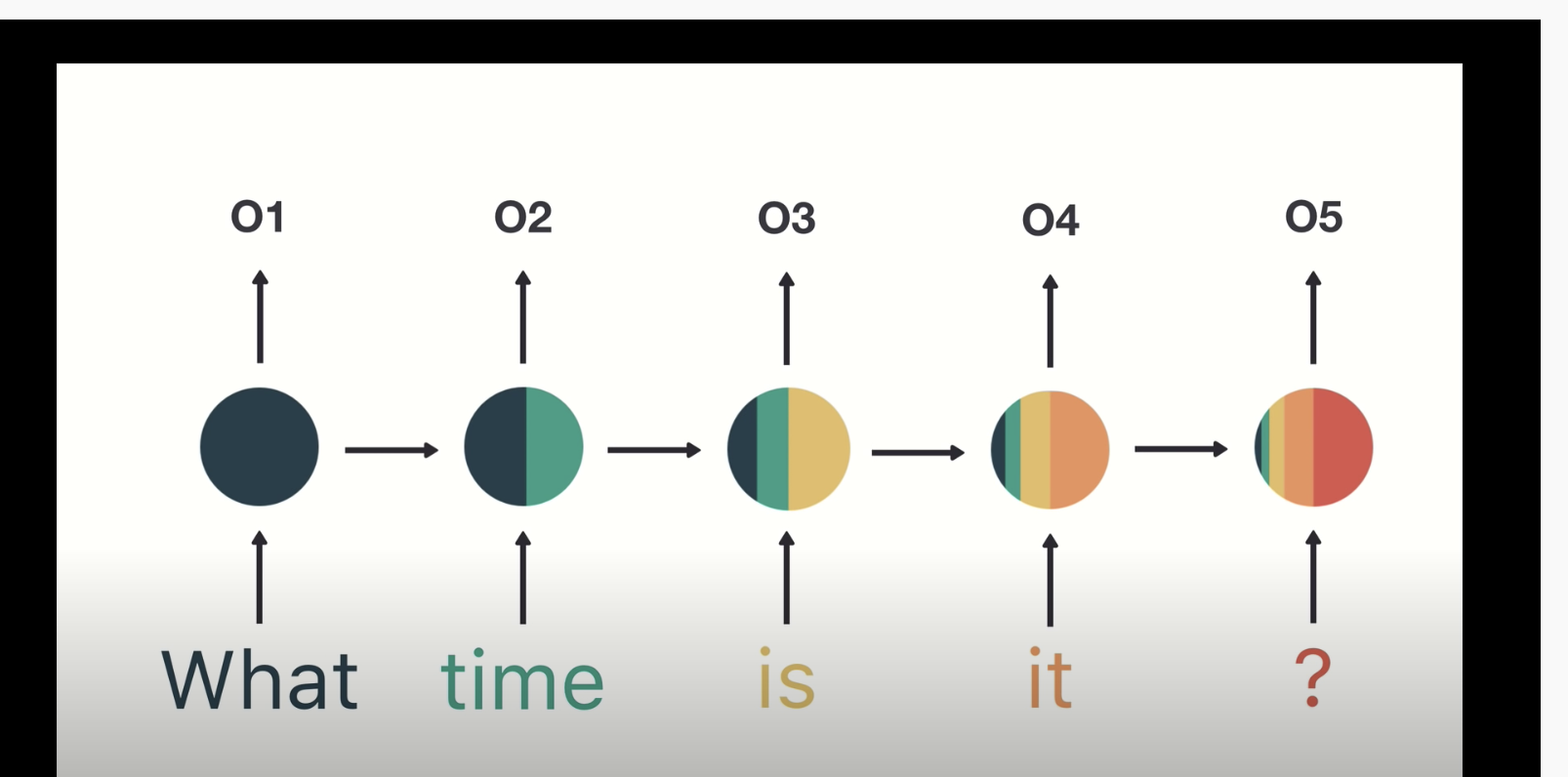
### Learning Sequences

Learning sequences can help with understanding. For example, if you were asked to list the letters of the alphabet you could very quickly recall the letters in a typical order. If you were asked to list the letters of the alphabet in reverse order you would probably pause and walk through those steps more slowly. However, an alternate sequence can be learned to make the task easier.

### Hidden State

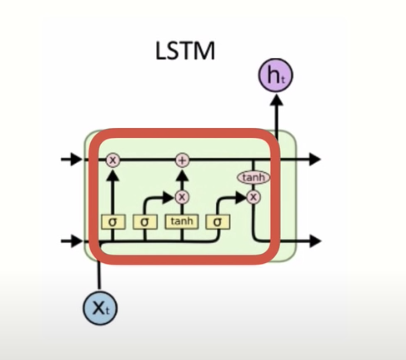
RNN’s include a hidden state to reference previous inputs. Figure 3 shows 5 steps to understand the meaning behind “What time is it?”. At the first step, the word “what” is read. At the second step, the word “time” is read and “what” is contained in a hidden state. In the third step, “is” is read and the previous two words are stored in the hidden state. By step 5, the representation of the initial states is significantly weighted less. As the RNN processes more steps it has difficulty retaining information from previous steps. Short term memory and vanishing gradients can often make it challenging for RNN’s to model with sequential data.

Figure : Tracking Hidden States with RNN’s



## Long Short-Term Memory Networks

LSTM’s were developed to overcome problems with RNN’s. LSTM’s use gates which perform operations to learn what to store in memory.



LSTM’s are computationally more expensive than RNN’s but LSTM’s can offer significant improvement to manage context about sequences. The result though is LSTM’s can be very slow to train.

Example : Single Layer LSTM

(This example is based on chapter 6 from Long Short Term Memory Networks by Dr. Jason Brownlee)

As mentioned earlier, LSTM’s use gates to determine what to store in memory. This example trains on samples of five numbers that range between 0 and 9. The numbers are one-hot-encoded so:

[5, 8, 3, 0, 9]

Becomes

# [[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

# [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

# [0, 0, 0, 1, 0, 0, 0, 0, 0, 0] # Third element where index = 2

# [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]

The target variable during training is always represented by the third element. In other words, y = 3 for this case. When the model is trained it will almost always pick the second element:

The code that creates the model and updates the weights is highlighted in green.

Here is the full program:

|  |
| --- |
| from random import randint  from numpy import array  from numpy import argmax  from keras.models import Sequential  from keras.layers import LSTM  from keras.layers import Dense  NUM\_FEATURES = 10  NUM\_SAMPLES = 5  TARGET\_INDEX = 2  NUM\_WEIGHT\_UPDATES = 100  # generate array of 5 numbers like [5, 8, 3, 0, 9].  # each number is >=0 and <10  def generate\_sequence():  return [randint(0, NUM\_FEATURES-1) for \_ in range(NUM\_SAMPLES)]  # one hot encode sequence  def oneHotEncode(sequence):  encoding = list()  # Convert [5, 8, 3, 0, 9]  # to  # [[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]  # [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]  # [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]  # [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]  # [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]  for value in sequence:  # Create vector of zeros.  vector = [0 for \_ in range(NUM\_FEATURES)]  vector[value] = 1 # Add 1 to vector.  encoding.append(vector)  return array(encoding)  # decode a one hot encoded string  def oneHotDecode(encoded\_seq):  # gets index of element with the maximum value.  return [argmax(vector) for vector in encoded\_seq]  # generate one example for an lstm  def generateSample(targetIndex):  # generate sequence such as [5, 8, 3, 0, 9]  sequence = generate\_sequence()  # one hot encode sequence so [5, 8, 3, 0, 9] becomes  # [[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]  # [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]  # [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]  # [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]  # [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]  encoded = oneHotEncode(sequence)  # reshape sequence to be 3D  X = encoded.reshape((1, NUM\_SAMPLES, NUM\_FEATURES))  # y becomes second element.  # [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]  y = encoded[targetIndex].reshape(1, NUM\_FEATURES)  return X, y  # define model  model = Sequential()  model.add(LSTM(25, input\_shape=(NUM\_SAMPLES, NUM\_FEATURES)))  model.add(Dense(NUM\_FEATURES, activation='softmax'))  # Our output is a one-hot encoded vector so use categorical  # crossentropy.  model.compile(loss='categorical\_crossentropy',  optimizer='adam', metrics=['accuracy'])  model.summary()  for i in range(10000):  trainX, trainy = generateSample(TARGET\_INDEX)  # Update model - weights are updated and are not reset.  model.fit(trainX, trainy, epochs=1, verbose=2)  # evaluate model  correct = 0  NUM\_EVALUATIONS = 100  for i in range(NUM\_EVALUATIONS):  X, y = generateSample(TARGET\_INDEX)  yhat = model.predict(X)  if oneHotDecode(yhat) == oneHotDecode(y):  correct += 1  print('Accuracy: %f' % ((correct / NUM\_EVALUATIONS))) |

Exercise (1 mark)

When model.fit() is called, indicate if the weights are updated or reset.

1. **Updated** b) Reset

## Multilayer LSTM’s

Multilayer LSTM’s have additional memory cells and each layer helps to boost performance by solving different parts of the problem.

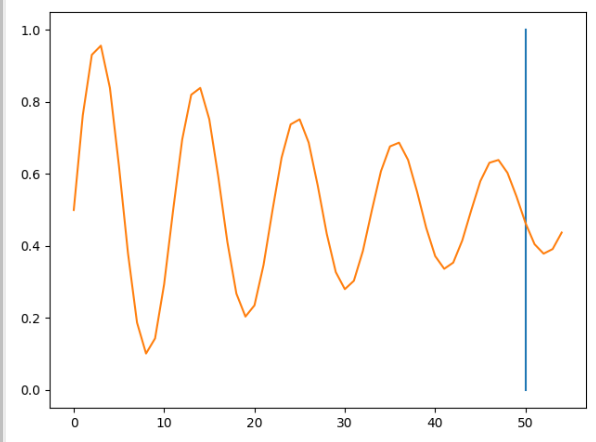
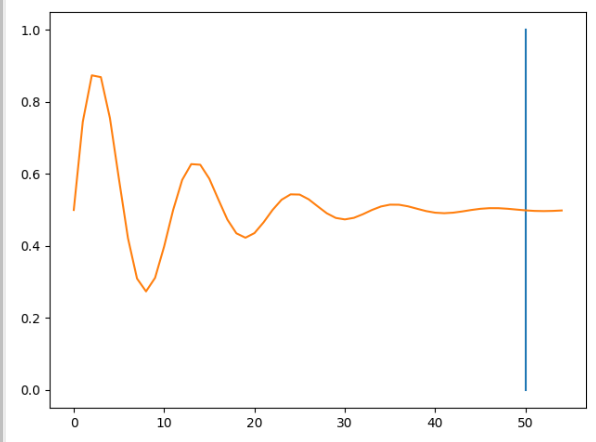
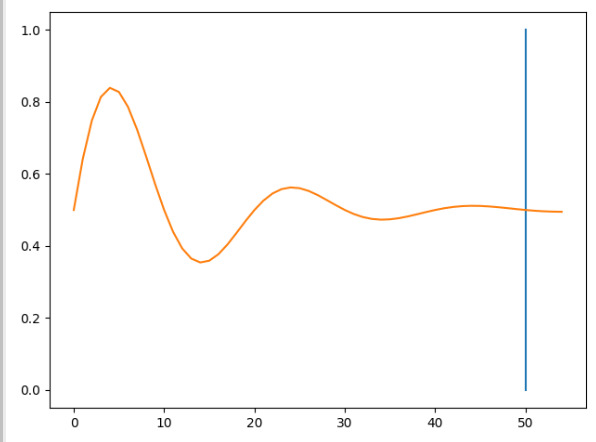
Example : Multilayer LSTM

(This example is based on chapter 7 from Long Short Term Memory Networks by Dr. Jason Brownlee)

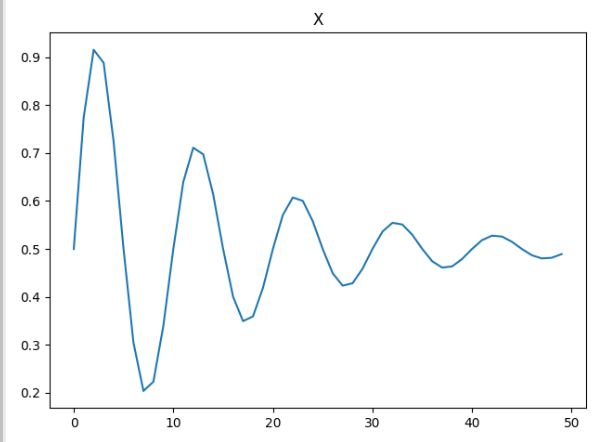
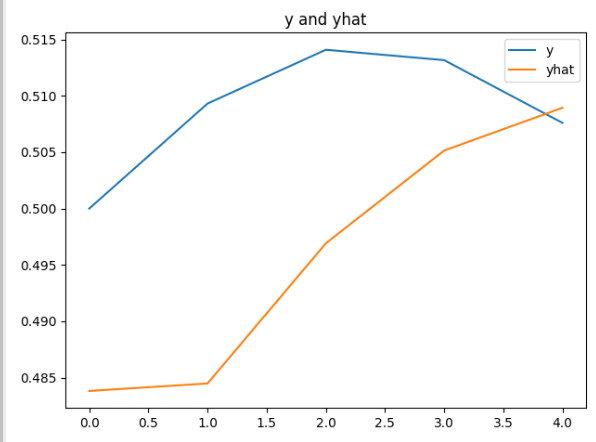
This example shows how a multi-layer LSTM can help to predict the next step in a sine wave sequence.

Samples used for this example include sets of sine waves that are defined by 55 points each. The first 50 points in the sequence define X. The last five points in the sequence are held back for y.

Figure : Three sequential sine wave samples. The first 50 points are X. The last 5 points are y.



The fitted model tries to predict the next five steps in the sequence.

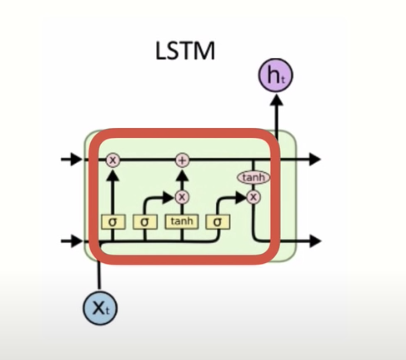
 

Here is the code:

|  |
| --- |
| from math import sin  from math import pi  from math import exp  from random import randint  from random import uniform  from numpy import array  from matplotlib import pyplot  from keras.models import Sequential  from keras.layers import LSTM  from keras.layers import Dense  # generate damped sine wave in [0,1]  def generate\_sequence(length, period, decay):  return [0.5 + 0.5 \* sin(2 \* pi \* i / period) \* exp(-decay \* i) \  for i in range(length)]  # generate input and output pairs of damped sine waves  def generate\_examples(input\_len, n\_patterns, output\_len):  X, y = list(), list()  for \_ in range(n\_patterns):  p = randint(10, 20)  d = uniform(0.01, 0.1)  sequence = generate\_sequence(input\_len + output\_len, p, d)  X.append(sequence[:-output\_len])  y.append(sequence[-output\_len:]) # Assigns next 5 values in sequence.  X = array(X).reshape(n\_patterns, input\_len, 1)  y = array(y).reshape(n\_patterns, output\_len)  return X, y  # configure problem  INPUT\_LEN = 50  OUTPUT\_LEN = 5  # define model  model = Sequential()  model.add(LSTM(20, input\_shape=(INPUT\_LEN, 1)))  model.add(Dense(OUTPUT\_LEN))  model.compile(loss='mae', optimizer='adam')  model.summary()  # fit model  X, y = generate\_examples(INPUT\_LEN, 10000, OUTPUT\_LEN)  history = model.fit(X, y, batch\_size=10, epochs=1)  # evaluate model  X, y = generate\_examples(INPUT\_LEN, 1000, OUTPUT\_LEN)  loss = model.evaluate(X, y, verbose=0)  print('Mean squared error: %f' % loss)  print("\n\*\*\* Make predictions")  for i in range(0, 5):  # prediction on new data  X, y = generate\_examples(INPUT\_LEN, 1, OUTPUT\_LEN)  yhat = model.predict(X, verbose=0)  pyplot.title("Y and Yhat")  pyplot.plot(y[0], label='y')  pyplot.plot(yhat[0], label='yhat')  pyplot.legend()  pyplot.show() |

### Long Short-Term Memory

LSTM’s were developed to overcome problems with RNN’s. LSTM’s use gates which perform operations to learn what to store in memory.



LSTM’s are computationally more expensive than RNN’s but LSTM’s can offer significant improvement to manage context about sequences. The result though is LSTM’s can be very slow to train.

## Tokenizing and Sequencing Sentences

We are going to look at LSTM’s to predict sentiment so we will take a moment first to discuss tokenization.

Example : Tokenizing and Sequencing Sentences

We are going to use LSTM’s to predict sentiment. Before we do though we are going to take a moment to understand how the words are converted to numeric vectors for modelling.

Starting with the following sentences:

"This is not yellow."

"This is a blue moon."

"Hello all."

texts\_to\_sequences()converts the sentences sets of integers:

[[1, 2, 3, 4],

[1, 2, 5, 6, 7],

[8, 9]]

The vectors are then padded with zeros so all sentence vectors are the same size. The first sentence vector represents four words. The second sentence vector represents five words. The third sentence vector represents two words.

|  |
| --- |
| array([[0, 1, 2, 3, 4],  [1, 2, 5, 6, 7],  [0, 0, 0, 8, 9]], dtype=int32) |

Here is all of the code which performs the transformation for the sentences. This code converts the code into a vertical array of features. We end up with is a vertical array of numerical features which is ideal input for a neural network.

|  |
| --- |
| from keras.preprocessing.text import Tokenizer  from keras\_preprocessing.sequence import pad\_sequences  sentence1 = "This is not yellow."  sentence2 = "This is a blue moon."  sentence3 = "Hello all."  sentences = [sentence1, sentence2, sentence3]  # Restrict tokenizer to use top 2500 words.  tokenizer = Tokenizer(num\_words=2500, lower=True,split=' ')  tokenizer.fit\_on\_texts(sentences)  # Convert to sequence of integers.  X = tokenizer.texts\_to\_sequences(sentences)  print(X)  # Showing padded sentences:  paddedX = pad\_sequences(X)  print(paddedX) |

Exercise (2 marks)

Just for fun, tokenize and pad the following sentences with your first and last name in it. Use three or more sentences such as:

YourFirstName Yourlastname is prepared to succeed.

YourFirstName Yourlastname sees opportunity in every challenge.

To get the mark, you must use your first and last name but if you want you can change the sentences. Show your code here:

|  |
| --- |
| *from* keras.preprocessing.text *import* Tokenizer  *from* keras\_preprocessing.sequence *import* pad\_sequences  sentence1 = "Sean Yue is prepared to succeed."  sentence2 = "Sean Yue sees opportunity in every challenge."  sentence3 = "Sean Yue is almost done with school."  sentences = [sentence1, sentence2, sentence3]  *# Restrict tokenizer to use top 2500 words.*  tokenizer = Tokenizer(*num\_words*=2500, *lower*=True,*split*=' ')  tokenizer.fit\_on\_texts(sentences)  *# Convert to sequence of integers.*  X = tokenizer.texts\_to\_sequences(sentences)  print(X)  *# Showing padded sentences:*  paddedX = pad\_sequences(X)  print(paddedX) |

Show your final output after tokenizing and padding here. Highlight the numbers which represent your first and last name.

|  |
| --- |
| [[1, 2, 3, 4, 5, 6], [1, 2, 7, 8, 9, 10, 11], [1, 2, 3, 12, 13, 14, 15]]  [[ 0 **1 2** 3 4 5 6]  [ **1 2** 7 8 9 10 11]  [ **1 2** 3 12 13 14 15]] |

## Long Short-Term Memory Terminology

We are not going to dive into the math or logic that enables an LSTM network but I will discuss some high-level definitions here before we build an LSTM model in code.

### Embedding Layer

Embedding layers are used in sentiment problems to store weights for words. Related words are geometrically closer such as dinner and kitchen. Words such as glacier and coconut are likely geometrically far apart. During a forward pass words are referenced in this layer to get a particular embedding. During a backward pass gradients are included in this layer.

### LSTM Layers

LSTM layers define cells for managing hidden states while iterating through different time steps. You have flexibility to adjust these values. Usually LSTM layers are accompanied with drop out layers.

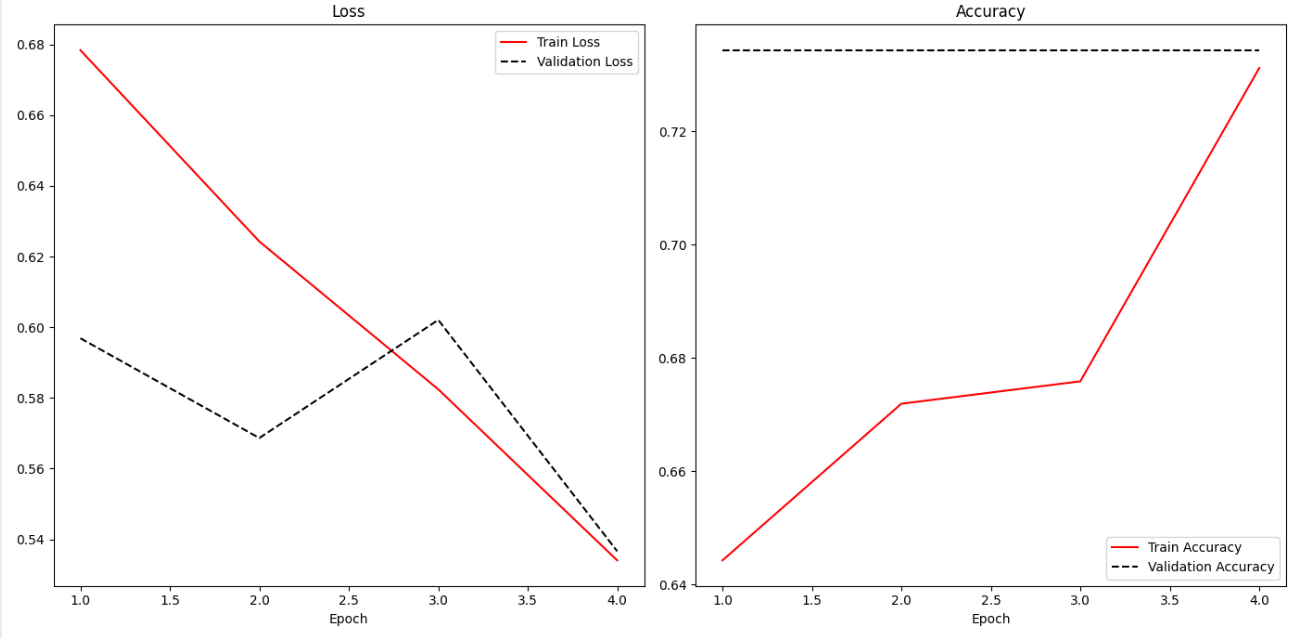
Example : LSTM for Sentiment Analysis

This example predicts sentiment for yelp reviews. It predicts based on this example.

<https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47>

I used a smaller file called yelp\_mini.csv to save time while tuning the hyperparameters. A larger yelp\_review.csv file is in the data sets folder for more rigorous training.

The loss and accuracy functions indicate a favourable situation where training and validation estimates coincide. Based on the graphs though, it may be possible that the model is underfit and that more epochs could improve results.



Here is the code:

|  |
| --- |
| import pandas as pd  import re  from keras\_preprocessing.sequence import pad\_sequences  from sklearn.model\_selection import train\_test\_split  from tensorflow.python.keras import Sequential  from tensorflow.python.keras.layers import Embedding, LSTM, Dense  PATH = "/Users/pm/Desktop/DayDocs/data/"  FILE = "yelp\_mini.csv"  data = pd.read\_csv(PATH + FILE)  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  # Create a sentiment column.  # Ratings above 3 are positive, otherwise they are negative.  data['sentiment'] = ['pos' if (x>3) else 'neg' for x in data['stars']]  data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))  from keras.preprocessing.text import Tokenizer  VOCABULARY\_SIZE = 2500  tokenizer = Tokenizer(num\_words=VOCABULARY\_SIZE, lower=True,split=' ')  tokenizer.fit\_on\_texts(data['text'].values)  X = tokenizer.texts\_to\_sequences(data['text'].values)  X = pad\_sequences(X)  WORDS\_PER\_SENTENCE = X.shape[0]  NUM\_REVIEWS = X.shape[1]  import numpy as np  VOCABULARY\_SIZE = np.amax(X) + 1  word\_info\_sz = 128 # Size of output vector for each word.  # This can be changed.    # Stores info about word sequence -  # "Eat to live" vs. "Live to eat" are very different.  sentence\_info\_sz = 200 # Vector size for storing info about  # entire sequence.  # This can be changed.  batch\_size = 32  model = Sequential()  model.add(Embedding(VOCABULARY\_SIZE, word\_info\_sz))  model.add(LSTM(sentence\_info\_sz, dropout=0.2))  model.add(Dense(2, activation='softmax')) # Two column one-hot encoded output.  # Target data is one-hot encoded so we must use ‘categorical\_crossentropy’ for loss.  # Here we are using one-hot encoding so we must use categorical\_crossentropy.  # One-hot encoding is a fancy way to say multi-column binary encoding.  # Y\_train  # [[0 1]  # [1 0]  # [0 1]  model.compile(loss = 'categorical\_crossentropy',  optimizer='adam',metrics = ['accuracy'])  print(model.summary())  Y = pd.get\_dummies(data['sentiment']).values  X\_train, X\_test, Y\_train, y\_test = train\_test\_split(X,Y, test\_size = 0.20)  history = model.fit(X\_train, Y\_train, batch\_size =batch\_size, epochs =4,  verbose = 1, validation\_data=(X\_test, y\_test))  score, acc = model.evaluate(X\_test, y\_test, verbose=2, batch\_size=batch\_size)  print("Score: %.2f" % (score))  print("Validation Accuracy: %.2f" % (acc))  import matplotlib.pyplot as plt  def showLoss(history):  # Get training and test loss histories  training\_loss = history.history['loss']  validation\_loss = history.history['val\_loss']  # Create count of the number of epochs  epoch\_count = range(1, len(training\_loss) + 1)  plt.subplot(1, 2, 1)  # Visualize loss history for training data.  plt.plot(epoch\_count, training\_loss, label='Train Loss', color='red')  # View loss on unseen data.  plt.plot(epoch\_count, validation\_loss, 'r--', label='Validation Loss',  color='black')  plt.xlabel('Epoch')  plt.legend(loc="best")  plt.title("Loss")  def showAccuracy(history):  # Get training and test loss histories  training\_loss = history.history['accuracy']  validation\_loss = history.history['val\_accuracy']  # Create count of the number of epochs  epoch\_count = range(1, len(training\_loss) + 1)  plt.subplot(1, 2, 2)  # Visualize loss history for training data.  plt.plot(epoch\_count, training\_loss, label='Train Accuracy', color='red')  # View loss on unseen data.  plt.plot(epoch\_count, validation\_loss, 'r--',  label='Validation Accuracy', color='black')  plt.xlabel('Epoch')  plt.legend(loc="best")  plt.title('Accuracy')  plt.subplots(nrows=1, ncols=2, figsize=(14,7))  showLoss(history)  showAccuracy(history)  plt.show() |

## Further Improvements

* We can use much larger dataset with more epochs to increase the accuracy. See yelp\_review.csv.
* We might use more hidden dense layers can be used to improve the accuracy (or not). We can tune other hyper parameters as well.

Exercise (1 mark)

What kind of loss function option is suitable when the target variable values are stored in a single column and values range between 0 and 1. You may have to refer to notes from earlier in the course to confirm.

|  |
| --- |
| Binary Cross Entropy Loss |

Exercise (1 mark)

What kind of loss function option is appropriate when the target variable values are stored in multiple columns and values for each column range between 0 and 1.

|  |
| --- |
| Categorical Cross Entropy Loss |

Exercise (1 mark)

What kind of loss function option is appropriate when the target variable values are stored in a single column and the possible values range between 0 and 5?

|  |
| --- |
|  |

Exercise (1 mark)

What kind of loss function option is appropriate when the target variable is one hot encoded?

|  |
| --- |
| Softmax Activation + Cross Entropy Loss |

Exercise (5 marks)

Grid search the word vector and sequence vector sizes ideally to improve accuracy. Or, if accuracy cannot be improved, then determine if the word vector and sequence vector sizes be reduced without deteriorating the accuracy.

NOTE: LSTM’s take a notoriously long amount of time to run. Please keep your grid search reasonable.

Show your accuracy score, word vector and sequence vector sizes here:

|  |
| --- |
|  |

Show the code used to perform the grid search here:

|  |
| --- |
| *import* pandas *as* pd  *import* re  *from* keras\_preprocessing.sequence   *import* pad\_sequences  *from* sklearn.model\_selection        *import* train\_test\_split  *from* tensorflow.python.keras        *import* Sequential  *from* tensorflow.python.keras.layers *import* Embedding, LSTM, Dense  PATH = "/Users/pm/Desktop/DayDocs/data/"  FILE = "yelp\_mini.csv"  data = pd.read\_csv(FILE)  *# Show all columns.*  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  *# Create a sentiment column.*  *# Ratings above 3 are positive, otherwise they are negative.*  data['sentiment'] = ['pos' *if* (x>3) *else* 'neg' *for* x *in* data['stars']]  data['text']      = data['text'].apply((lambda *x*: re.sub('[^a-zA-z0-9\s]','',x)))  *from* keras.preprocessing.text *import* Tokenizer  VOCABULARY\_SIZE = 2500  tokenizer = Tokenizer(*num\_words*=VOCABULARY\_SIZE, *lower*=True,*split*=' ')  tokenizer.fit\_on\_texts(data['text'].values)  X = tokenizer.texts\_to\_sequences(data['text'].values)  X = pad\_sequences(X)  WORDS\_PER\_SENTENCE = X.shape[0]  NUM\_REVIEWS        = X.shape[1]  *import* numpy *as* np  VOCABULARY\_SIZE = np.amax(X) + 1  word\_info\_sz      = 256 *# Size of output vector for each word.*  *# This can be changed.*    *# Stores info about word sequence -*  *# "Eat to live" vs. "Live to eat" are very different.*  sentence\_info\_sz  = 300 *# Vector size for storing info about*  *# entire sequence.*  *# This can be changed.*  batch\_size        = 32  model = Sequential()  model.add(Embedding(VOCABULARY\_SIZE, word\_info\_sz))  model.add(LSTM(sentence\_info\_sz, *dropout*=0.2))  model.add(Dense(2, *activation*='softmax')) *# Two column one-hot encoded output.*  *# Target data is one-hot encoded so we must use ‘categorical\_crossentropy’ for loss.*  *# Here we are using one-hot encoding so we must use categorical\_crossentropy.*  *# One-hot encoding is a fancy way to say multi-column binary encoding.*  *#  Y\_train*  *# [[0 1]*  *#  [1 0]*  *#  [0 1]*  model.compile(*loss* = 'categorical\_crossentropy',  *optimizer*='adam',*metrics* = ['accuracy'])  print(model.summary())  Y = pd.get\_dummies(data['sentiment']).values  X\_train, X\_test, Y\_train, y\_test = train\_test\_split(X,Y, *test\_size* = 0.20)  history = model.fit(X\_train, Y\_train, *batch\_size* =batch\_size, *epochs* =12,  *verbose* = 1, *validation\_data*=(X\_test, y\_test))  score, acc = model.evaluate(X\_test,  y\_test, *verbose*=2, *batch\_size*=batch\_size)  print("Score: %.2f" % (score))  print("Validation Accuracy: %.2f" % (acc))  *import* matplotlib.pyplot  *as* plt  def showLoss(*history*):  *# Get training and test loss histories*      training\_loss       = history.history['loss']      validation\_loss     = history.history['val\_loss']  *# Create count of the number of epochs*      epoch\_count = range(1, len(training\_loss) + 1)      plt.subplot(1, 2, 1)  *# Visualize loss history for training data.*      plt.plot(epoch\_count, training\_loss, *label*='Train Loss', *color*='red')  *# View loss on unseen data.*      plt.plot(epoch\_count, validation\_loss, 'r--', *label*='Validation Loss',  *color*='black')      plt.xlabel('Epoch')      plt.legend(*loc*="best")      plt.title("Loss")  def showAccuracy(*history*):  *# Get training and test loss histories*      training\_loss       = history.history['accuracy']      validation\_loss     = history.history['val\_accuracy']  *# Create count of the number of epochs*      epoch\_count = range(1, len(training\_loss) + 1)      plt.subplot(1, 2, 2)  *# Visualize loss history for training data.*      plt.plot(epoch\_count, training\_loss, *label*='Train Accuracy', *color*='red')  *# View loss on unseen data.*      plt.plot(epoch\_count, validation\_loss, 'r--',  *label*='Validation Accuracy', *color*='black')      plt.xlabel('Epoch')      plt.legend(*loc*="best")      plt.title('Accuracy')  plt.subplots(*nrows*=1, *ncols*=2,  *figsize*=(14,7))  showLoss(history)  showAccuracy(history)  plt.show() |

Exercise (5 marks)

Starting with the solution from Exercise 7, increase the number of epochs. Add a callback to save the best model with early stopping. (Please refer to the week 4 lesson for notes on how to do this). Then load the binary model and use it to make a prediction with the test set. Show the accuracy score that is achieved with the binary model.

Show your accuracy score here:

|  |
| --- |
|  |

Show your code here:

|  |
| --- |
|  |