**FINAL REPORT – TEMPLATE**

Your final report is also the milestone-2. It should contain all the 6 Sections mentioned in the project document. The report structure and brief requirements of each section is listed below.

1. Summary of problem statement, data and findings

Every good abstract describes briefly what was intended at the outset, and summarizes findings and implications.

2. Overview of the final process

Briefly describe your problem methodology. Include information about the salient features of your data, data pre-processing steps, the algorithms you used and how you combined techniques.

3. Step-by-step walk through the solution

Describe the steps you took to solve the problem. What did you find at each stage, and how did it inform the next steps? Build up to the final solution.

4. Model evaluation

Describe the final model in detail. What was the objective, what parameters were prominent, and how did you evaluate the success of your models?

We have chosen 4 models:

1. Encoder decoder with Attention model

Attention networks are also more efficient and require less computational resources. This is an important improvement, as it often requires significant computing power in the form of a GPU (which is not always accessible) to train RNN’s.

Attention is a mechanism that addresses a limitation of the encoder-decoder architecture on long sequences, and that in general speeds up the learning and lifts the skill of the model no sequence to sequence prediction problems.

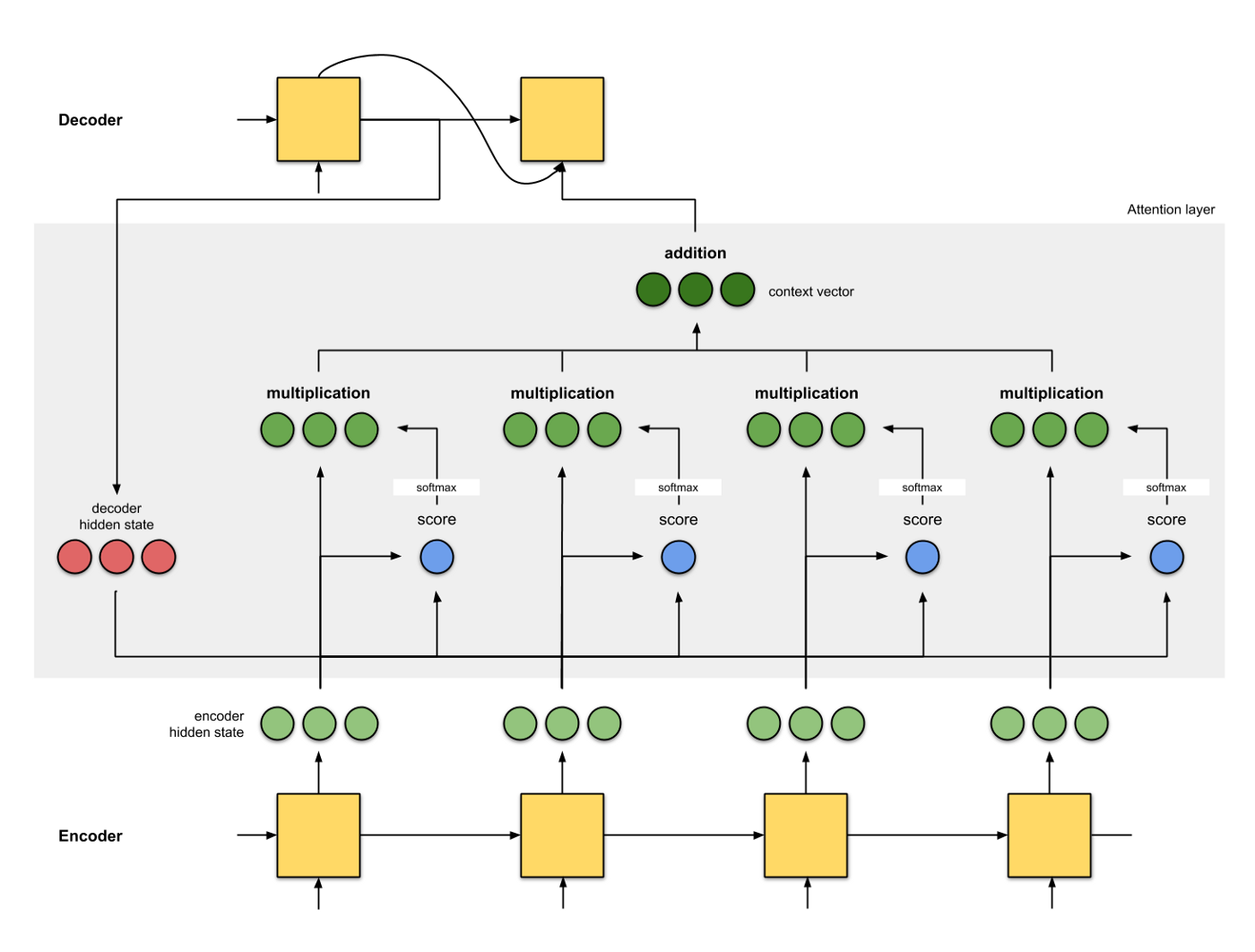
The encoder-decoder architecture for recurrent neural networks is proving to be powerful on a host of sequence-to-sequence prediction problems in the field of natural language processing such as machine translation and caption generation.

The encoder-decoder model for recurrent neural networks is an architecture for sequence-to-sequence prediction problems.

It is comprised of two sub-models, as its name suggests:

Encoder: The encoder is responsible for stepping through the input time steps and encoding the entire sequence into a fixed length vector called a context vector.

Decoder: The decoder is responsible for stepping through the output time steps while reading from the context vector.



Code for Encoder decoder with Attention Model

from random import randint

from numpy import array

from numpy import argmax

from numpy import array\_equal

from keras.models import Sequential

from keras.layers import LSTM

from attention\_decoder import AttentionDecoder

# generate a sequence of random integers

def generate\_sequence(length, n\_unique):

return [randint(0, n\_unique-1) for \_ in range(length)]

# one hot encode sequence

def one\_hot\_encode(sequence, n\_unique):

encoding = list()

for value in sequence:

vector = [0 for \_ in range(n\_unique)]

vector[value] = 1

encoding.append(vector)

return array(encoding)

# decode a one hot encoded string

def one\_hot\_decode(encoded\_seq):

return [argmax(vector) for vector in encoded\_seq]

# prepare data for the LSTM

def get\_pair(n\_in, n\_out, cardinality):

# generate random sequence

sequence\_in = generate\_sequence(n\_in, cardinality)

sequence\_out = sequence\_in[:n\_out] + [0 for \_ in range(n\_in-n\_out)]

# one hot encode

X = one\_hot\_encode(sequence\_in, cardinality)

y = one\_hot\_encode(sequence\_out, cardinality)

# reshape as 3D

X = X.reshape((1, X.shape[0], X.shape[1]))

y = y.reshape((1, y.shape[0], y.shape[1]))

return X,y

# configure problem

n\_features = 50

n\_timesteps\_in = 5

n\_timesteps\_out = 2

# define model

model = Sequential()

model.add(LSTM(150, input\_shape=(n\_timesteps\_in, n\_features), return\_sequences=True))

model.add(AttentionDecoder(150, n\_features))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# train LSTM

for epoch in range(5000):

# generate new random sequence

X,y = get\_pair(n\_timesteps\_in, n\_timesteps\_out, n\_features)

# fit model for one epoch on this sequence

model.fit(X, y, epochs=1, verbose=2)

# evaluate LSTM

total, correct = 100, 0

for \_ in range(total):

X,y = get\_pair(n\_timesteps\_in, n\_timesteps\_out, n\_features)

yhat = model.predict(X, verbose=0)

if array\_equal(one\_hot\_decode(y[0]), one\_hot\_decode(yhat[0])):

correct += 1

print('Accuracy: %.2f%%' % (float(correct)/float(total)\*100.0))

# spot check some examples

for \_ in range(10):

X,y = get\_pair(n\_timesteps\_in, n\_timesteps\_out, n\_features)

yhat = model.predict(X, verbose=0)

print('Expected:', one\_hot\_decode(y[0]), 'Predicted', one\_hot\_decode(yhat[0]))

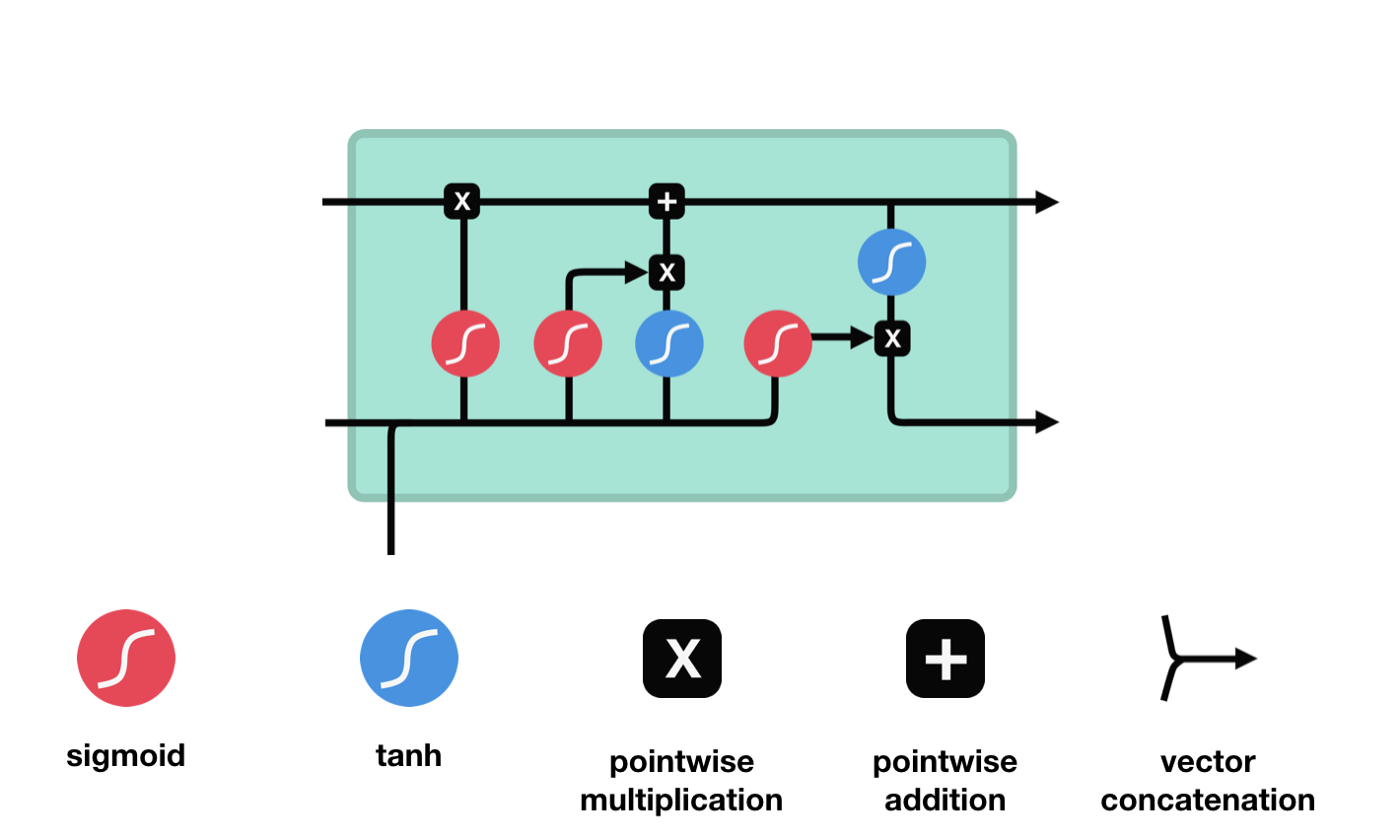
links:

<https://machinelearningmastery.com/encoder-decoder-attention-sequence-to-sequence-prediction-keras/>

<https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3>

1. LSTM

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM’s cells.



These operations are used to allow the LSTM to keep or forget information. Now looking at these operations can get a little overwhelming so we’ll go over this step by step.

# create and fit the LSTM network

model = Sequential()

model.add(LSTM(4, input\_shape=(1, look\_back)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(trainX, trainY, epochs=100, batch\_size=1, verbose=2)

# make predictions

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# invert predictions

trainPredict = scaler.inverse\_transform(trainPredict)

trainY = scaler.inverse\_transform([trainY])

testPredict = scaler.inverse\_transform(testPredict)

testY = scaler.inverse\_transform([testY])

# calculate root mean squared error

trainScore = math.sqrt(mean\_squared\_error(trainY[0], trainPredict[:,0]))

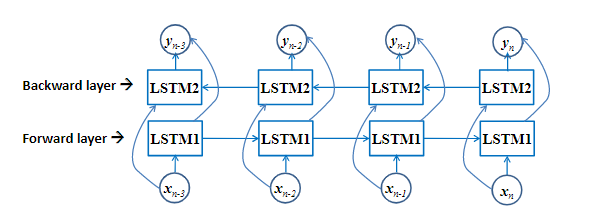
print('Train Score: %.2f RMSE' % (trainScore))

testScore = math.sqrt(mean\_squared\_error(testY[0], testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

link: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

1. Bi-Directional LSTM



When a problem requires a reverse flow of information then it’s better to use this model.

Example: POS tagging (Part of speech) .

Example:

model = Sequential()  
model.add(Bidirectional(LSTM(10, return\_sequences=True), input\_shape=(5, 10)))  
model.add(Bidirectional(LSTM(10)))  
model.add(Dense(5))  
model.add(Activation('softmax'))  
model.compile(loss='categorical\_crossentropy', optimizer='rmsprop')  
  
 # With custom backward layer  
 model = Sequential()  
 forward\_layer = LSTM(10, return\_sequences=True)  
 backward\_layer = LSTM(10, activation='relu', return\_sequences=True,  
                       go\_backwards=True)  
 model.add(Bidirectional(forward\_layer, backward\_layer=backward\_layer,  
                         input\_shape=(5, 10)))  
 model.add(Dense(5))  
 model.add(Activation('softmax'))  
 model.compile(loss='categorical\_crossentropy', optimizer='rmsprop')

Example 2:

from keras.models import Sequential

from keras.layers import Activation, LSTM, Merge, TimeDistributedDense

from keras.optimizers import SGD

def fork (model, n=2):

forks = []

for i in range(n):

f = Sequential()

f.add (model)

forks.append(f)

return forks

# First bidirectional LSTM layer

forward = Sequential()

forward.add(LSTM(output\_dim=512, input\_shape=(50, 43), return\_sequences=True))

backward = Sequential()

backward.add(LSTM(output\_dim=512, input\_shape=(50, 43), return\_sequences=True, go\_backwards=True))

model = Sequential()

model.add(Merge([forward, backward], mode='concat'))

# Second bidirectionl LSTM layer

forward\_2, backward\_2 = fork(model)

forward\_2.add(LSTM(output\_dim=512, input\_shape=(50, 512), return\_sequences=True))

backward\_2.add(LSTM(output\_dim=512, input\_shape=(50, 512), return\_sequences=True, go\_backwards=True))

model = Sequential()

model.add(Merge([forward\_2, backward\_2], mode='concat'))

# Softmax decision layer

model.add(TimeDistributedDense(output\_dim=5))

model.add(Activation('softmax'))

# Optimizer function

sgd = SGD(lr=0.1, decay=1e-5, momentum=0.9, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer=sgd)

print("Train...")

model.fit([X\_train, X\_train], Y\_train, batch\_size=1, nb\_epoch=nb\_epoches, validation\_data=([X\_test, X\_test], Y\_test), verbose=1, show\_accuracy=True)

Reference Link:

1: <https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional>

2: <https://github.com/keras-team/keras/issues/1629>

1. GRU

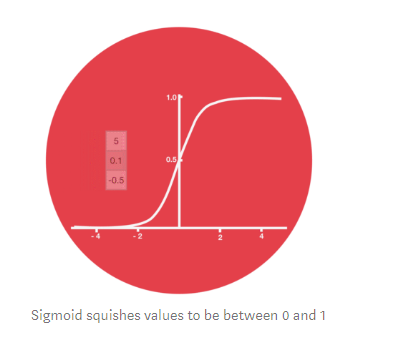
1: Forget or reset gate

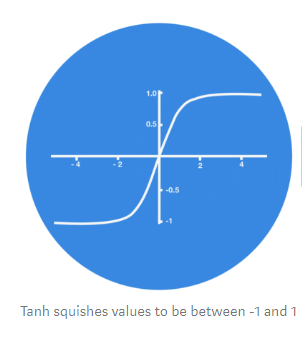
2: update gate.

Gates uses sigmoid functions with point wise multiplication.

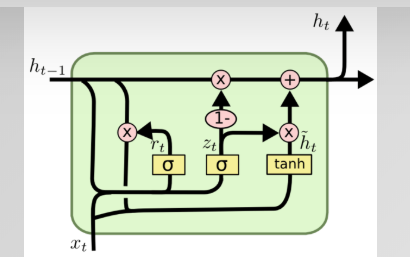
GRU has fewer parameters (U and W are smaller) hence it trains faster and needs less data to generalize.

Sigmoid functions are used so that the output can be compressed between 0 and 1.

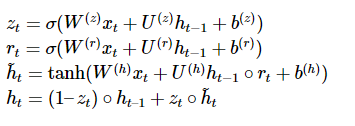




GRU’s are used to overcome the vanishing gradient problem faced by RNN (Recurrent Neural Network)

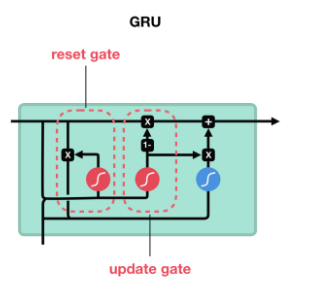


Mathematical GRU Model.



Zt filter for previous state, Zt lower then lot of previous state is reused, So the input Xt of current state does not affect the output , if Zt is high then output of the current step is influenced a lot by the current input step Xt but it is not influenced a lot by the previous state Ht-1

Rt is the forget gate or reset gate, it allows the cell to forget certain parts of the state.



Example:

from keras.models import Sequential

from keras.layers.recurrent import GRU

|  |
| --- |
| model = Sequential() |
|  | model.add(GRU(X\_train.shape[-1], y\_train.shape[-1])) |
|  | model.add(Activation('softmax')) |
|  | model.compile(loss='categorical\_crossentropy', optimizer='adadelta') |
|  | history = model.fit(X\_train, y\_train, nb\_epoch=12, batch\_size=16, validation\_data=(X\_test, y\_test), show\_accuracy=True, verbose=2) |

Reference Link:

1: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

2: <https://www.data-blogger.com/2017/08/27/gru-implementation-tensorflow/>

5. Comparison to benchmark

How does your final solution compare to the benchmark you laid out at the outset? Did you improve on the benchmark? Why or why not?

6. Visualizations

In addition to quantifying your model and the solution, please include all relevant visualizations that support the ideas/insights that you gleaned from the data.

7. Implications

How does your solution affect the problem in the domain or business? What recommendations would you make, and with what level of confidence?

8. Limitations

What are the limitations of your solution? Where does your model fall short in the real world? What can you do to enhance the solution?

9. Closing Reflections

What have you learned from the process? What you do differently next time?