GROUP 7
Kshitij Sankesara
Nishant Priyam
Rishabh Agarwal
Rucha Kadam

Sentence Boundary Disambiguation Using CBHG Model

PROF. LU XIAO

IST 664 – Project Report

Table of Contents

PROJECT OVERVIEW

A Sentence Boundary Disambiguation (SBD), also known as sentence tokenization, is a problem in Natural Language Processing (NLP) of deciding the begin and end points of a sentence. However, sentence boundary identification is challenging because punctuation marks can often become ambiguous while identifying. For example, a period may denote an abbreviation, decimal point, an ellipsis, or an email address but not the end of sentence every time.

We are trying to solve this problem with the help of deep learning algorithms, machine learning architecture to detect the whole sentence without messing the meaning of period in different sentences. We saw the use of CBHG model in the research paper called Tacotron: Towards end-to-end speech synthesis. We tried to implement the CBHG model for our problem statement in the best possible manner and get the desired output in the form of detection of perfect sentence from the data set that we will provide. We are using TensorFlow platform to implement the algorithm. We are using TensorBoard to get the visualization in the form of graphs and models for our model's learning rate, time loss and many more functions.

We are also using hyper parameters tuning using trial and error method. We are defining

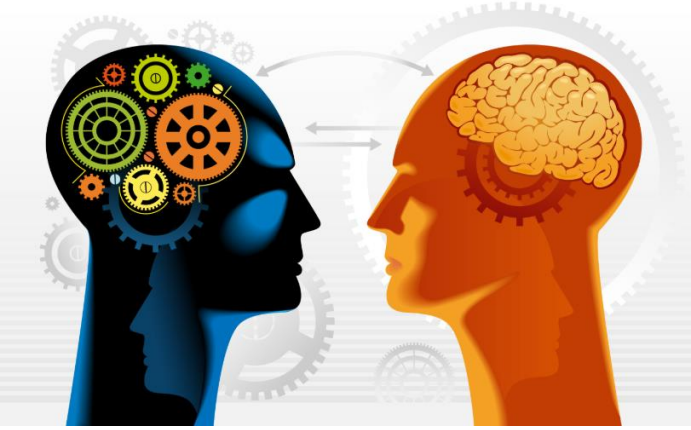
1. Learning Rate
2. Maximum Number of Characters in a Sentence
3. Minimum Number of Characters in a Sentence
4. Hidden Unit
5. Number of Encoders and Decoders blocks
6. Number of Head
7. Dropout Rate
8. Encoder Number of Bank
9. Number of Highway Network Blocks
10. Number of Epochs and Size of Batches.

DATASET

In this project, we needed a clean data set to find the tokenized sentence. Therefore, we decided to move forward with a prebuild data set rather than building our own data set. There are many corpora present in NLTK like Gutenberg Corpus, Reuters Corpus, Brown Corpus etc.

The Brown University Standard Corpus of Present-Day American English (or just Brown Corpus) was compiled in the 1960s by Henry Kučera and W. Nelson Francis at Brown University, Providence, Rhode Island as a general corpus (text collection) in the field of corpus linguistics. It contains 500 samples of English-language text, totalling roughly one million words, compiled from works published in the United States in 1961.

The texts for the corpus were sampled from 15 different text categories to make the corpus a good standard reference. Much of the corpus usefulness lies in the fact that the Brown corpus lay-out has been copied by other corpus compilers. The LOB corpus (British English) and the Kolhapur Corpus (Indian English) are two examples of corpora made to match the Brown corpus. The availability of corpora which are so similar in structure is a valuable resource for researchers interested in comparing different language varieties, for example For a long time, the Brown and LOB corpora were almost the only easily available computer readable corpora. Much research within the field of corpus linguistics has therefore been made using these data. There are more than 500 sources from which the data set was created. All these sources are categorized in genres. These genres contain news, editorial, hobbies, religion, humor, fiction etc.



IMPORTANT DEFINITIONS

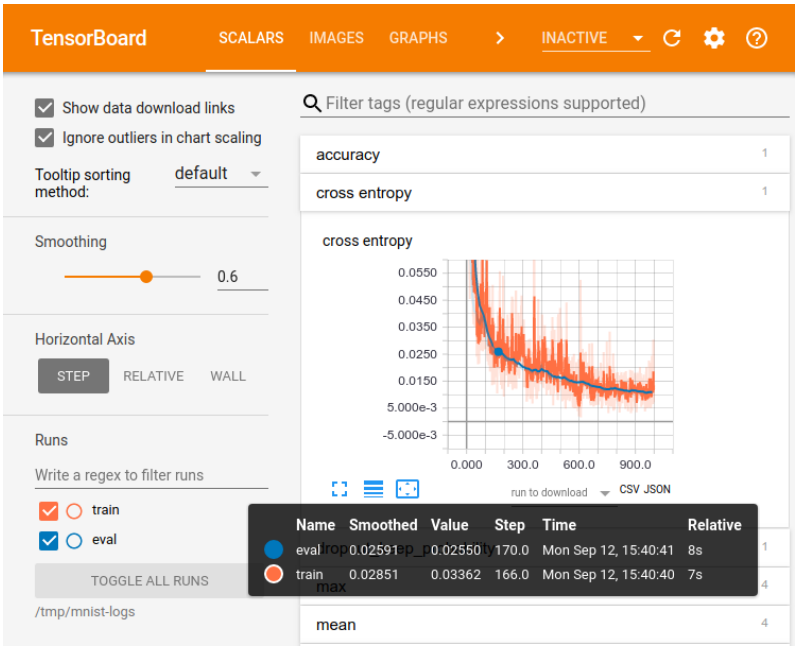
NLTK

Natural Language Toolkit, also known as NLTK is free, community-driven, open source platform to work with human language data to build our desired python programs. There are many prebuild text processing libraries like tagging, tokenization, parsing, semantic reasoning, wrappers etc. It is available for all PC (Personal Computer) OS platform like Mac OS, Windows, Linux.

TensorFlow

In the field of Machine Learning and Data Science, TensorFlow is the one of the best platform to work with. TensorFlow is an open source end-to-end machine learning platform. Different algorithms and Machine Learning Models like CNN, RNN etc. can be directly implemented in our project using TensorFlow. Companies like Intel, Google, AirBNB, CocaCola etc. uses TensorFlow as their regular part of companies' recommendation architecture etc.

TensorBoard

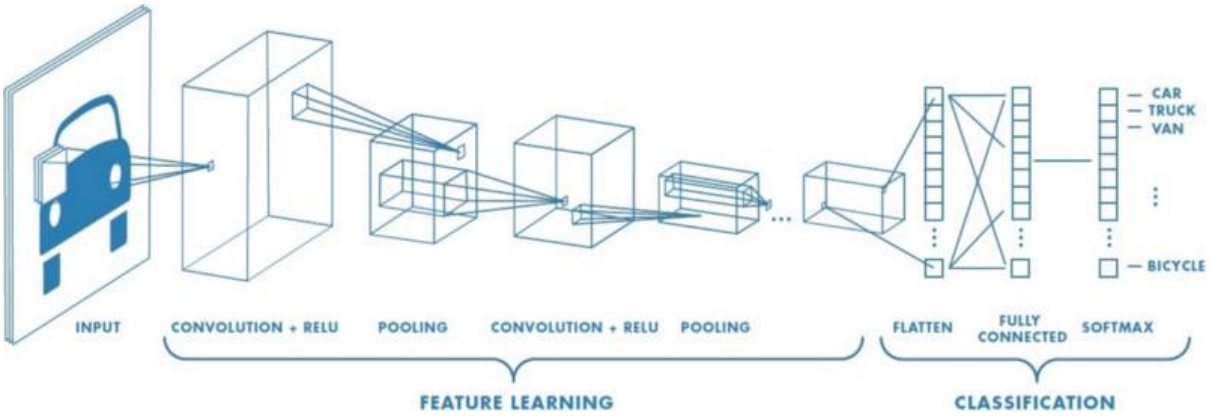


Computation of algorithms and deep neural network for which we will use TensorFlow, can be complex and confusing. To make it easy for debug and optimizing, a visualization board was added by TensorFlow, known as TensorBoard. The accuracy rate graph and time loss graph that we will get at the end of this report, we have used TensorBoard. After fully configuring, TensorBoard looks like this.

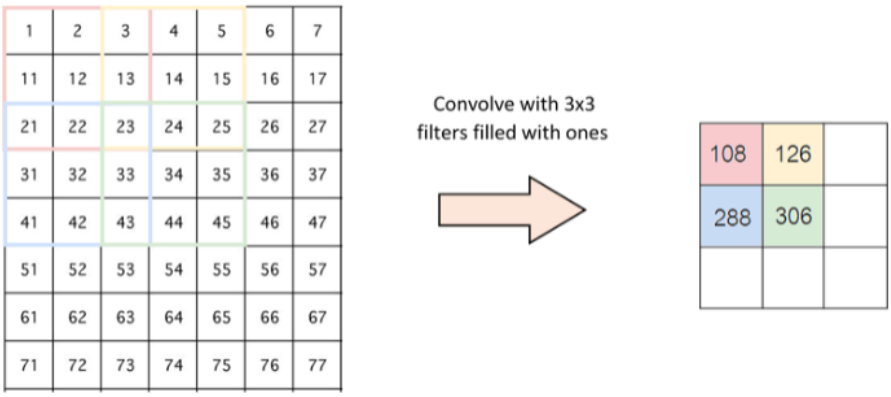
CNN

Convolution Neural Network, also known as CNN, is a deep learning algorithm which is famously used for Image Classification problems like image reorganization, object detection etc. It is generally used for supervised learning. Convolution is the very first layer of a CNN which is used for feature extraction of image using image matrix operation.

The below diagram will give you the overview of how many convolution layers works with neural network.



- **Convolution Layers:** Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).
- **Strides:** Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

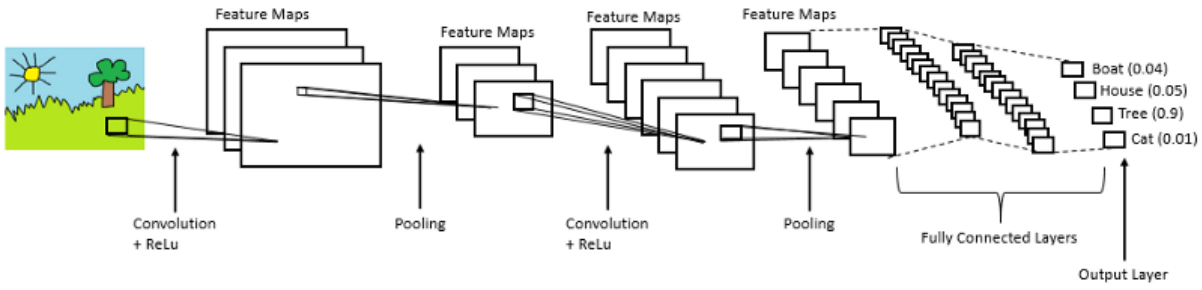


Pic: Strides with 2 pixels

- **Polling Layers:** Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types:
  - Max Pooling
  - Average Pooling
  - Sum Pooling

Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

Below is the complete CNN architecture:



In short, CNN works as follows:

- Provide input image into convolution layer

- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size
- Add as many convolutional layers until satisfied
- Flatten the output and feed into a fully connected layer (FC Layer)
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

Convolution Bank

When we are using several layers of convolution filters, it is called convolution bank. The general concept of a convolution bank is to extract as much accurate features of our data as possible.

Highway Network

Highway Network is another concept of Machine Learning. It is an approach to optimize the network and increase the depth.

Parameters

The variable that is defined internally and whose values can be predicted and estimated using data is called model parameters. Generally, model parameters are learned from historical training data.

Hyper-parameters

The variable that is defined externally and whose values cannot be predicted and estimated using the training data is called model hyper-parameter. We tune hyper-parameter using trial and error and define them after seeing better or worse result.

RNN

Recurrent Neural Networks is a powerful set of artificial neural network algorithm especially useful for processing sequential data such as sound, time series (sensor) data or written natural language. They are also very powerful and unique till date because they are the only one that uses internal memory.

RNN's are relatively old, like many other deep learning algorithms. They were initially created in the 1980's, but can only show their real potential since a few years, because of the increase in available computational power, the massive amounts of data that we have nowadays and the invention of LSTM in the 1990's.

Because of their internal memory, RNN's are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next.

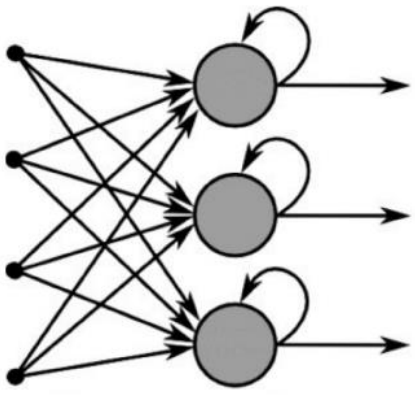
This is the reason why they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more because they can form a much deeper understanding of a sequence and its context, compared to other algorithms.

Recurrent Neural Networks produce predictive results in sequential data that other algorithms can't.

But when do you need to use a Recurrent Neural Network ?

"Whenever there is a sequence of data and that temporal dynamics that connects the data is more important than the spatial content of each individual frame." – Lex Fridman (MIT).

- **Working of RNN:** In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously.



Recurrent Neural Network

- **Obstacle with RNN:** There are two major obstacles with RNN. But first we have to know what a gradient is. A gradient is a partial derivative with respect to its inputs. A gradient measures how much output of a function changes, if we change the input a little bit. We can also think gradient as slope of function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops to learning. A gradient simply measures the change in all weights with regard to the change in error.

The errors are **Vanishing Gradient** and **Exploding Gradients.**

- **Exploding Gradients:** We speak of "Exploding Gradients"when the algorithm assigns a stupidly high importance to the weights, without much reason. But fortunately, this problem can be easily solved if you truncate or squash the gradients.
- **Vanishing Gradients:** We speak of "Vanishing Gradients"when the values of a gradient are too small and the model stops learning or takes way too long because of that. This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber, which we will discuss now.

Why to compare Model with LSTM?

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore, it is well suited to learn from important experiences that have very long-time lags in between.

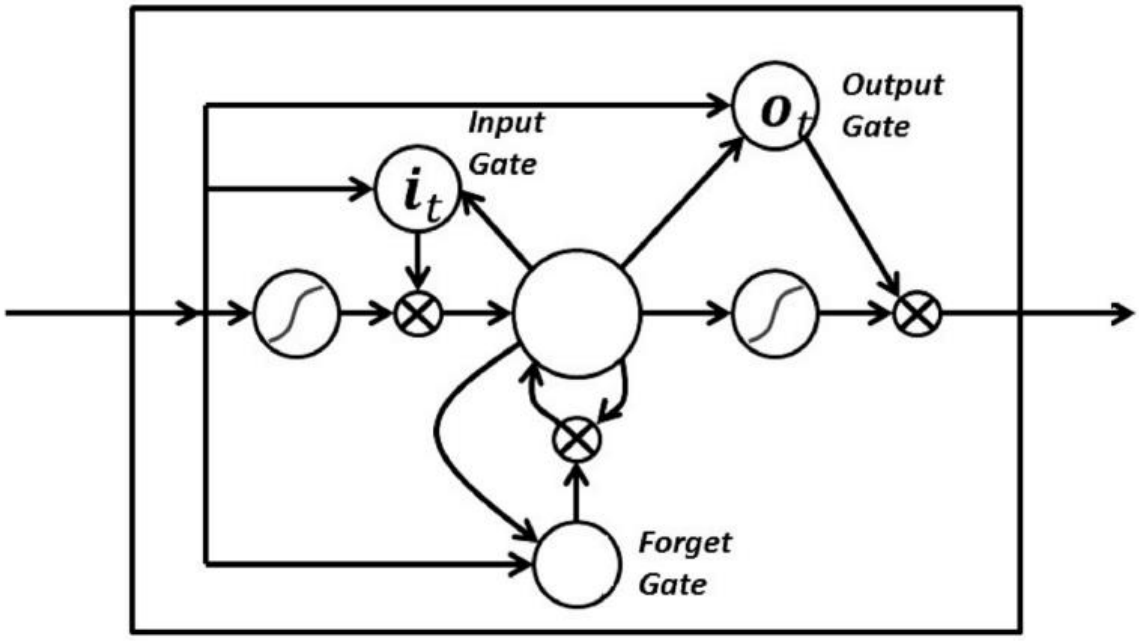
The units of an LSTM are used as building units for the layers of a RNN, which is then often called an LSTM network.

LSTM's enable RNN's to remember their inputs over a long period of time. This is because LSTM's contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory.

This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (e.g if it opens the gates or not), based on the importance it assigns to the

information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not.

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate). Below is the diagram of RNN with three gates:



Pic: RNN with three gates, basic of LSTM (12)

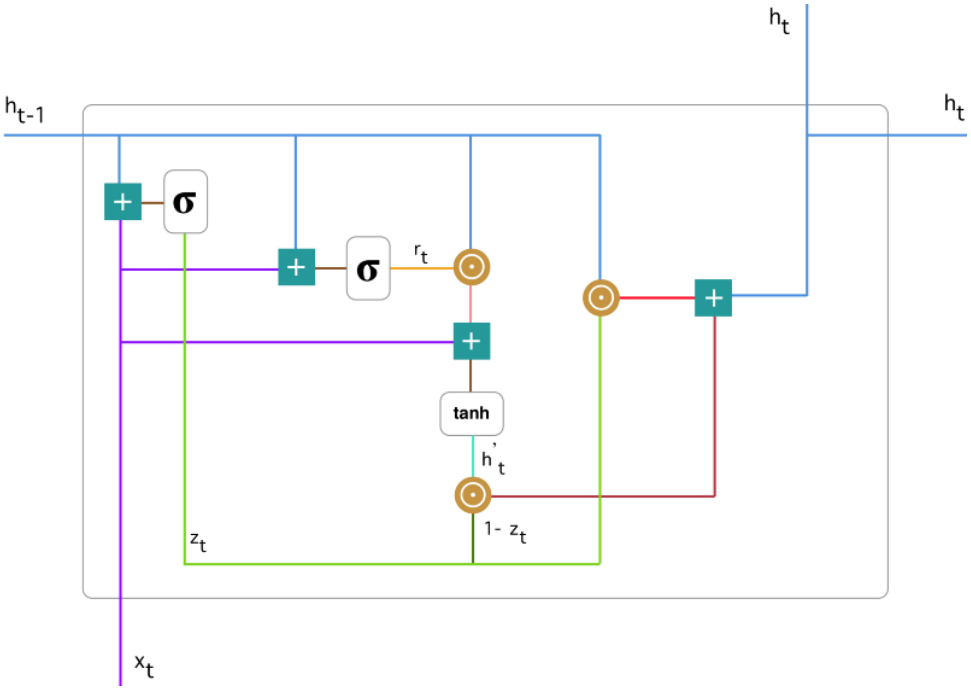
The gates in a LSTM are analog, in the form of sigmoids, meaning that they range from 0 to 1. The fact that they are analog, enables them to do backpropagation with it.

The problematic issues of vanishing gradients is solved through LSTM because it keeps the gradients steep enough and therefore the training relatively short and the accuracy high.

Bidirectional GRU

GRU was first introduced in the research paper "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". The aim was to solve the **vanishing gradient** problem which comes with standard RNN. We can also consider GRU as a variation of LSTM as both of their design and design flow are similar and produce somewhat equally good results. The difference between GRU and LSTM (Long Short-term Memory) is LSTM has three gates (input, output and forget gates) and GRU has two gates (reset and update gates) which is used to solve the above-mentioned problem of RNN. Generally, GRU has better performance of small data sets in comparison with LSTM. But when we need more accurate result on a big data set, LSTM is always better. The only problem with LSTM is, it takes more resources and time than GRU.

- **Working of GRU:** There are two gates used by GRU called **update gate** and **reset gate**. These are two vectors that decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.



Pic: Gated Recurrent Unit (14)

We have to understand the notations which are used in the above diagram:

Pic: Notations(14)

Notation Definition:

- **Tanh function** is mainly used classification between two classes.
- **Sigmoid function** is especially used for models where we have to **predict the probability** as an output. Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.
- **Hadamard Product** is elementwise multiplication and it outputs a vector. Values that correspond positionally are multiplied to produce a new matrix.

Let us know about the gates of GRU.

1. **Update Gate:** The update gate helps the model to determine how much of the past information (from previous time steps) needs to be passed along to the future. That is rearlly powerful because the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient problem.
2. **Reset Gate:** This gate is used from the model to decide how much of the past information to forget. This formula is the same as the one for the update gate. The difference comes in the weights and the gate’s usage.

Let’s see how exactly the gates will affect the final output. First, we start with the usage of the reset gate. We introduce a new memory content which will use the reset gate to store the relevant information from the past. It is calculated as follows:

$$h'_t = tanh(Wx_t + r_t \ (hadamard\ operation)\ Uh_{t-1})$$

1. Multiply the input *x_t* with a weight *W* and *h_(t-1)* with a weight *U.*
2. Calculate the Hadamard (element-wise) product between the reset gate *r_t* and *Uh_(t-1).* That will determine what to remove from the previous time steps. Let's say we have a sentiment analysis problem for determining one's opinion about a book from a review he wrote. The text starts with "This is a fantasy book which illustrates…" and after a couple paragraphs ends with "I didn't quite enjoy the book because I think it captures too many details." To determine the overall level of satisfaction from the book we only need the last part of the review. In that case as the neural network approaches to the end of the text it will learn to assign *r_t* vector close to 0, washing out the past and focusing only on the last sentences.
3. Sum up the results of step 1 and 2.
4. Apply the nonlinear activation function *tanh*.

There is one more step. The network needs to calculate *h_t* — vector which holds information for the current unit and passes it down to the network. In order to do that the update gate is needed. It determines what to collect from the current memory content — *h'_t* and what from the previous steps — *h_(t-1)*.

Now, we can see how GRUs are able to store and filter information using their update and reset gates. That eliminates the vanishing gradient problem since the model is not washing out the new input every single time but keeps the relevant information and passes it down to the next time steps of the network. If carefully trained, they can perform extremely well even in complex scenarios.

Residual Connection

Residual Connection is similar to 'skip connection'. As the name suggests, they help to flow the gradient without passing through some activation functions and some layers.

Stacking

Stacking is also a machine learning technique which helps in combining multiple classification or regression models using a meta-classifier.

HYPOTHESIS

While reading multiple papers and open articles regarding text-to-speech conversion or vice-versa, we learned that LSTM architecture is best suitable with large data set to get the desired result. Meanwhile GRU is suitable for small data set to get desired output. Through this project, we are trying to get the better sentence tokenization result with GRU in comparison to LSTM architecture on small data set.

We have considered that the case of under-fitting and over-fitting of training data set is not there and proceeded with the project.

We have also tuned the hyper-parameters and not checked with multiple learning rate because of time and resource constraints.

IDEA TOWARDS CBHG

While we were reading about different neural networks and machine learning algorithms that can help us for sentence tokenization, we read ample amount of research papers and open source codes and articles as well. Some of them are GSTs (Global Style Tokens), Tacotron: Towards end-to-end speech synthesis, End-to-end Neural Speech Synthesis and many more. The most common thing that we observed was the use of GRU (Gated Recurrent Unit) in combination with different convolution neural networks, layers, strides, RNN and ANN, encoder, decoder etc. So we decided to use CBHG model architecture for sentence tokenization.

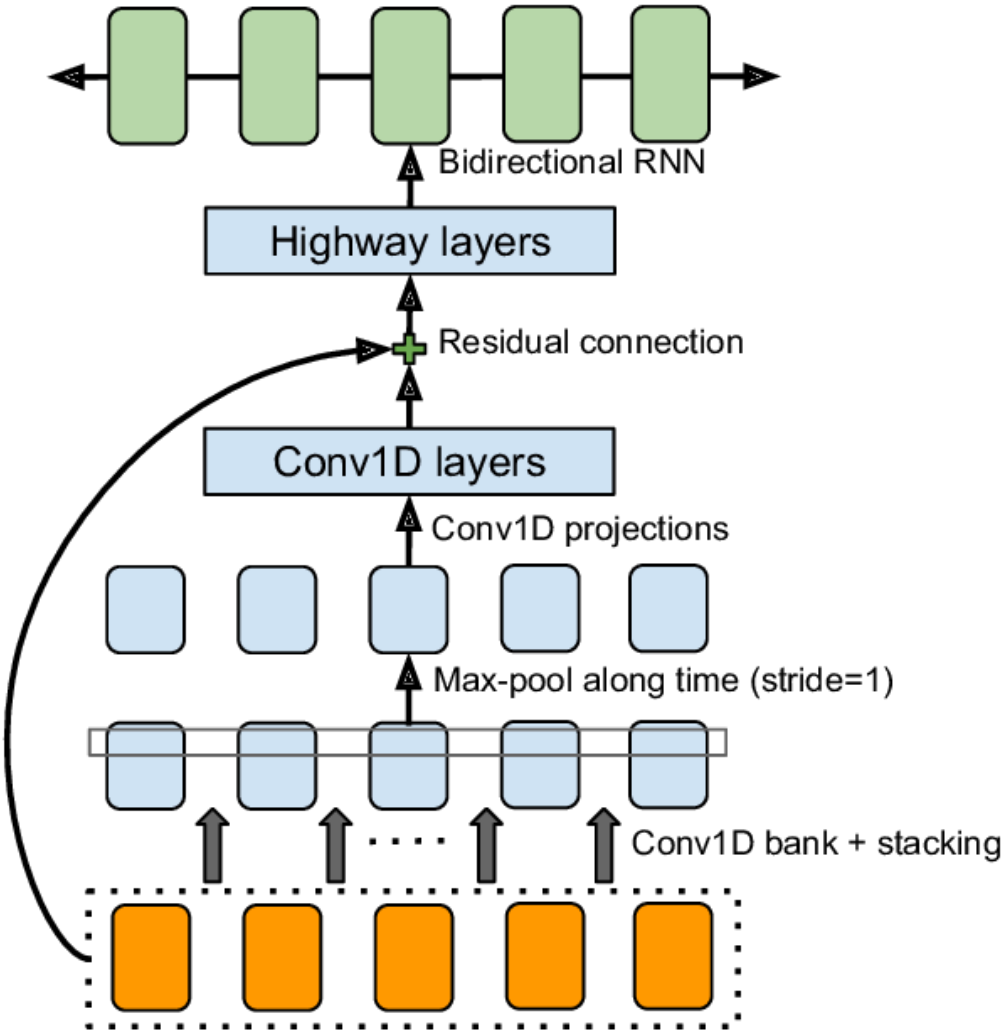
What is CBHG?

The CBHG (1-D Convolution Bank + Highway Network + Bidirectional Gated Recurrent Unit, GRU) was first introduced in Tacotron.

CBHG is a powerful module for extracting representations from sequences. The input sequence is first convolved with K sets of 1-dimensional convolutional filters, where the k-th set contains C_k filters of width k where k can be 1, 2 ..k. The convolution bank consists of K convolutions of kernel size 1..K respectively each with C filters and followed by a ReLU activation. “Same” padding is used to preserve the original number of time steps across all convolutions. These outputs are then concatenated along the channels dimension, batch normalized and max-pooled with stride 1, kernel 2. This is followed by two 3x3 convolutions to project the output back to the same size as the input, so that a residual connection can be formed by adding the original inputs to our convolution bank outputs.

This output is then passed through 4 highway network layers and the final representation is extracted with a bi-directional GRU. This allows the network to construct features using both the forward and backward contexts.

CBHG Architecture:



Initially, a layer of one dimensional convolution bank is used with stacking. The convolution ID layers and highway layers is connected using residual connection. Then we use bidirectional GRU at the end. For optimize transmission or passing of the output from the Conv1D layers to Highway Layers, we use residual connection. Then it is passed to bidirectional GRU which is type of RNN and similar to LSTM. The above is the architecture of the CBHG model.

REQUIRED CODE SNIPPET

Convolutional Bank:

Convolutional Bank

In [5]:
```
def conv1d(inputs, filters=None,size=1,rate=1,padding="SAME",use_bias=False,activation_fn=None,scope="conv1d",reuse=None):
    with tf.variable_scope(scope):
        if padding.lower() == "causal":
            pad_len = (size - 1) * rate
            inputs = tf.pad(inputs, [[0, 0], [pad_len, 0], [0, 0]])
            padding = "valid"

        if filters is None:
            filters = inputs.get_shape().as_list[-1]

        params = {"inputs": inputs, "filters": filters, "kernel_size": size,
                  "dilation_rate": rate, "padding": padding, "activation": activation_fn,
                  "use_bias": use_bias, "reuse": reuse}

        outputs = tf.layers.conv1d(**params)
    return outputs
```

Gated Recurrent Unit

```
def gru(inputs, num_units=None, bidirection=False, scope="gru", reuse=None):
    if num_units is None:
        num_units = inputs.get_shape()[-1]

    with tf.variable_scope(scope, reuse=reuse):
        if num_units is None:
            num_units = inputs.get_shape().as_list[-1]

        cell = tf.contrib.rnn.GRUCell(num_units)
        if bidirection:
            cell_bw = tf.contrib.rnn.GRUCell(num_units)
            outputs, _ = tf.nn.bidirectional_dynamic_rnn(cell, cell_bw, inputs, dtype=tf.float32)
            return tf.concat(outputs, 2)
        else:
            outputs, _ = tf.nn.dynamic_rnn(cell, inputs, dtype=tf.float32)
            return outputs
```

Prenet and Highwaynet:

In [8]:
```
def prenet(inputs, num_units=None, dropout_rate=0, is_training=True, scope="prenet", reuse=None):
    if num_units is None:
        num_units = [inputs.get_shape()[-1], inputs.get_shape()[-1]]

    with tf.variable_scope(scope, reuse=reuse):
        outputs = tf.layers.dense(inputs, units=num_units[0], activation=tf.nn.relu, name="dense1")
        outputs = tf.layers.dropout(outputs, rate=dropout_rate, training=is_training, name="dropout1")
        outputs = tf.layers.dense(outputs, units=num_units[1], activation=tf.nn.relu, name="dense2")
        outputs = tf.layers.dropout(outputs, rate=dropout_rate, training=is_training, name="dropout2")

    return outputs
```

In [9]:
```
def highwaynet(inputs, num_units=None, scope="highwaynet", reuse=None):
    if num_units is None:
        num_units = inputs.get_shape()[-1]

    with tf.variable_scope(scope, reuse=reuse):
        H = tf.layers.dense(inputs, units=num_units, activation=tf.nn.relu, name="H")
        T = tf.layers.dense(inputs, units=num_units, activation=tf.nn.sigmoid, name="T")
        C = 1. - T
        outputs = H * T + inputs * C

    return outputs
```

Training Graph Implementation:

```
In [14]:
g = Graph()
print("Graph loaded")

char2idx, idx2char = load_vocab()
with g.graph.as_default():
    X_val, Y_val = load_data(mode="val")
    num_batch = len(X_val) // batch_size
    sv = tf.train.Supervisor(graph=g.graph,
                             logdir=logdir,
                             save_model_secs=0)
    with sv.managed_session() as sess:
        for epoch in range(1, num_epochs + 1):
            if sv.should_stop(): break
            for step in tqdm(range(g.num_batch), total=g.num_batch, ncols=70, leave=False, unit='b'):
                sess.run(g.train_op)
                if step % 100 == 0:
                    gs, mean_loss = sess.run([g.global_step, g.mean_loss])
                    print("\nAfter global steps %d, the training loss is %.2f" % (gs, mean_loss))
            gs = sess.run(g.global_step)
            sv.saver.save(sess, logdir + '/model_epoch_%02d_gs_%d' % (epoch, gs))
            total_hits, total_targets = 0, 0
            for step in tqdm(range(num_batch), total=num_batch, ncols=70, leave=False, unit='b'):
                x = X_val[step*batch_size:(step+1)*batch_size]
                y = Y_val[step*batch_size:(step+1)*batch_size]
                num_hits, num_targets = sess.run([g.num_hits, g.num_targets], {g.x: x, g.y: y})
                total_hits += num_hits
                total_targets += num_targets
```

Run Code

```
            y = Y_val[step*batch_size:(step+1)*batch_size]
            num_hits, num_targets = sess.run([g.num_hits, g.num_targets], {g.x: x, g.y: y})
            total_hits += num_hits
            total_targets += num_targets
        print("\nAfter epoch %d, the validation accuracy is %d/%d=%.2f" % (epoch, total_hits, total_targets, total_hits/t

print("Done")
```

```
WARNING:tensorflow:From <ipython-input-13-837ac878fb28>:53: arg_max (from tensorflow.python.ops.gen_math_ops) is deprecated
and will be removed in a future version.
Instructions for updating:
Use `argmax` instead
Graph loaded
WARNING:tensorflow:From <ipython-input-14-8af56cd4d630>:13: Supervisor.__init__ (from tensorflow.python.training.supervisor)
is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
WARNING:tensorflow:Error encountered when serializing global_step.
Type is unsupported, or the types of the items don't match field type in CollectionDef.
'Tensor' object has no attribute 'to_proto'
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting standard services.
INFO:tensorflow:Starting queue runners.
  0%|                                          | 0/273 [00:00<?, ?b/s]
```

```
INFO:tensorflow:Starting queue runners.
  0%|                                          | 0/273 [00:00<?, ?b/s]
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:global_step/sec: 0

After global steps 1, the training loss is 0.75
  3%|▉                                 | 9/273 [01:50<52:50, 12.01s/b]
INFO:tensorflow:Recording summary at step 9.
INFO:tensorflow:global_step/sec: 0.0764846
  7%|██                                | 20/273 [03:53<47:23, 11.24s/b]
INFO:tensorflow:Recording summary at step 20.
INFO:tensorflow:global_step/sec: 0.0919877
 11%|███▏                              | 30/273 [05:51<48:46, 12.04s/b]
INFO:tensorflow:Recording summary at step 30.
INFO:tensorflow:global_step/sec: 0.083334
 15%|████▏                             | 40/273 [07:56<44:19, 11.41s/b]
INFO:tensorflow:Recording summary at step 40.
INFO:tensorflow:global_step/sec: 0.0833326
```

```
import os
import numpy as np


def eval():
    # Load graph
    g = Graph(is_training=False)
    print("Graph loaded")

    # Load data
    X, Y = load_data(mode="test")  # texts
    char2idx, idx2char = load_vocab()

    with g.graph.as_default():
        sv = tf.train.Supervisor()
        with sv.managed_session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
            # Restore parameters
            sv.saver.restore(sess, tf.train.latest_checkpoint(logdir))
            print("Restored!")

            # Get model
            mname = open(logdir + '/checkpoint', 'r').read().split('"')[1]  # model name

            # Inference
            if not os.path.exists(savedir): os.mkdir(savedir)
            with open("{}/{}".format(savedir, mname), 'w') as fout:
                results = []
```

Result Set:

```
            # Inference
            if not os.path.exists(savedir): os.mkdir(savedir)
            with open("{}/{}".format(savedir, mname), 'w') as fout:
                results = []
                baseline_results = []
                for step in range(len(X) // batch_size):
                    x = X[step * batch_size: (step + 1) * batch_size]
                    y = Y[step * batch_size: (step + 1) * batch_size]

                    # predict characters
                    preds = sess.run(g.preds, {g.x: x})

                    for xx, yy, pp in zip(x, y, preds):  # sentence-wise
                        expected = ''
                        got = ''
                        for xxx, yyy, ppp in zip(xx, yy, pp):  # character-wise
                            if xxx == 0:
                                break
                            else:
                                got += idx2char.get(xxx, "*")
                                expected += idx2char.get(xxx, "*")
                            if ppp == 1: got += " "
                            if yyy == 1: expected += " "

                            # prediction results
                            if ppp == yyy:
                                results.append(1)
```

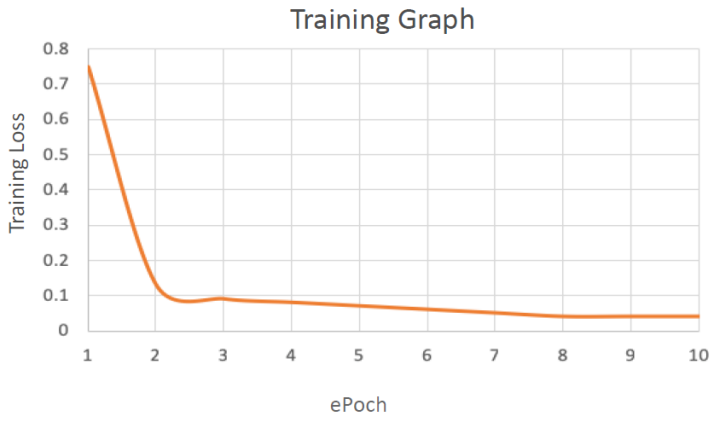
```
                            # prediction results
                            if ppp == yyy:
                                results.append(1)
                            else:
                                results.append(0)

                            # baseline results
                            if yyy == 0: # no space
                                baseline_results.append(1)
                            else:
                                baseline_results.append(0)

                        fout.write("Expected: " + expected + "\n")
                        fout.write("Got: " + got + "\n\n")
                fout.write(
                    "Final Accuracy = %d/%d=%.4f\n" % (sum(results), len(results), float(sum(results)) / len(results)))
                fout.write(
                    "Baseline Accuracy = %d/%d=%.4f" % (sum(baseline_results), len(baseline_results), float(sum(baseline_resu


if __name__ == '__main__':
    eval()
    print("Done")
```

```
                            if yyy == 0: # no space
                                baseline_results.append(1)
                            else:
                                baseline_results.append(0)

                        fout.write("Expected: " + expected + "\n")
                        fout.write("Got: " + got + "\n\n")
                fout.write(
                    "Final Accuracy = %d/%d=%.4f\n" % (sum(results), len(results), float(sum(results)) / len(results)))
                fout.write(
                    "Baseline Accuracy = %d/%d=%.4f" % (sum(baseline_results), len(baseline_results), float(sum(baseline_resu


if __name__ == '__main__':
    eval()
    print("Done")
```

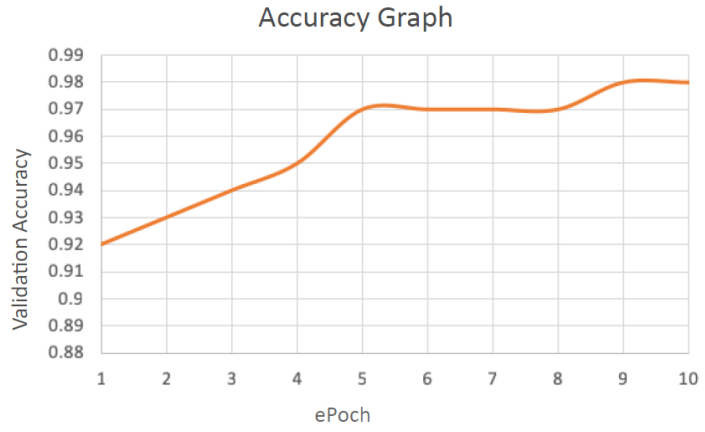
```
Graph loaded
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting standard services.
WARNING:tensorflow:Standard services need a 'logdir' passed to the SessionManager
INFO:tensorflow:Starting queue runners.
INFO:tensorflow:Restoring parameters from logdir\model_epoch_10_gs_2730
Restored!
Done
```

REQUIRED GRAPHS

1) Training Graph



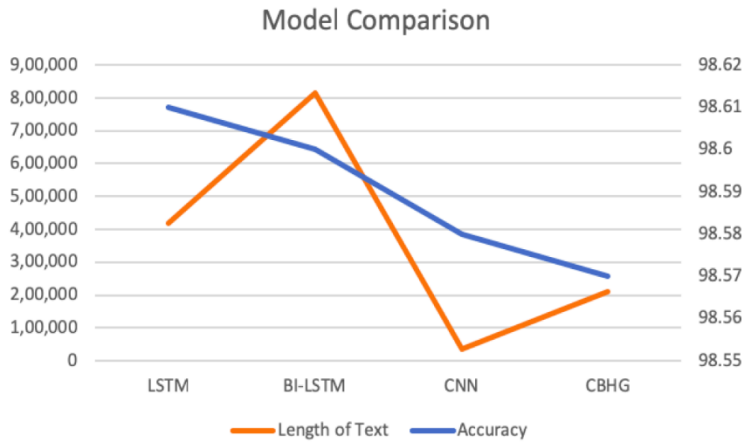
The above Training graph depicts the ePoch value against the training loss time. An Epoch is a complete pass through all the training data. A model is trained until the error rate is acceptable, and this will often take multiple passes through the complete data set. In the graph, we can see that as ePoch increases, the training loss time decreases. Here, when the ePoch value is 1 the training loss time is close to 0.75. And when we keep on increasing the ePoch value, the training loss time keeps on decreasing. We had thought about ePoch till 20 but we didn't had sufficient time for the same. Finally, at ePoch 10 the training loss time was about 0.06.



2) Accuracy Graph

The above Accuracy graph depicts the ePoch value against the validation accuracy. Again, the model is trained over here until we get an acceptable accuracy. Here we started with an accuracy of 0.92. As we increased the ePoch, the accuracy started increasing significantly. At ePoch 10, we achieved 0.986 accuracy. This is acceptable when we compare it with the other models as they give us approximately the same accuracy.

3) Results



In the above graph, we have compared four models on the basis of their text length and their model accuracy. The four models are LSTM, BI-LSTM, CNN and CBHG. The primary axis of the Model Comparison graph shows us the number of texts of the dataset used. It ranges from 100k words to 900k. The secondary axis gives us the model accuracy starting from 98.55% till 98.62%. Here the blue line gives us the accuracy for each models and line orange shows us the Length of text.

CBHG Model gives us approximately the same accuracy when compared to other models like LSTM, BI-LSTM and CNN. Although, CBHG uses GRU with convolution bank which is expected to give better accuracy than LSTM when trained on larger dataset, LSTM gave marginally better accuracy.

RECEIVED OUTPUTS

```
Expected: At least he had the decency to blush she thought
Obtained: At least he had the decency to blush she thought

Expected: We caught the early train to New York
Obtained: We caught the early train to New York

Expected: Ran away on a black night with a lawful wedded man
Obtained: Ran a way on a black night with a lawful wedded man

Expected: He gave a short hard laugh and looked at her knowingly
Obtained: He gave a short hard laugh and looked at her knowingly

Expected: His energy was prodigious
Obtained: His energy was prodigious

Expected: Still there it is
Obtained: Still there it is

Expected: Have you ever heard of Thuggee
Obtained: Have you ever heard of Thuggee

Expected: In short and to borrow an arboreal phrase slash timber
Obtained: In short and to borrow an arboreal phrases lash timber

Expected: Quite candidly fellows I wouldn't be in your shoes for all
the rice in China
Obtained: Quite candidly fellows I wouldn't be in your shoes for all
the rice in China

Expected: Quasimodo defines his own art as the search for what is
not there
Obtained: Quasimodo defines his own art as the search for what is
not there

Final Accuracy = 208674/211699=0.9857
Baseline Accuracy = 166107/211699=0.7846
```

CONCLUSION

Our hypothesis states that GRU is better than LSTM for sentence detection. If we run 20 ePoch then CBHG model is definitely better than LSTM. But as we had limited resources, we were not able to run for 20 ePoch. The time required for running it for 20 ePoch was more. The number of resources required were more for 20 ePoch than for 10, because of that we were not able to run. The time taken for 10 ePoch was less and we received approximately the same accuracy as LSTM.

FUTURE SCOPE

Currently as compared to LSTM which is the best model, we have got approximately the same accuracy while running CBHG model. If we increase our ePoch to 20-25 rather than 10, then the accuracy of CBHG model will exceed the accuracy obtained by LSTM model. If we run our model for more time, then we can exceed accuracy obtained by LSTM. The accuracy obtained on a smaller dataset is much better than the accuracy obtained by LSTM on a bigger dataset.

REFERENCES

- Wang, Y., Skerry-Ryan, R., & Saurous, R. A. (2017, April 6). TACOTRON: TOWARDS END-TO-END SPEECH SYN- THESIS. Retrieved from https://arxiv.org/pdf/1703.10135.pdf

- Barron, A. (n.d.). End-to-End Neural Speech Synthesis. Retrieved from http://web.stanford.edu/class/cs224s/reports/Alex_Barron.pdf

- Stanton, D., Wang, Y., & Skerry-Ryan, R. (2018, August 04). Predicting Expressive Speaking Style From Text In End-To-End Speech Synthesis. Retrieved from https://www.groundai.com/project/predicting-expressive-speaking-style-from-text-in-end-to-end-speech-synthesis/

- Nbronbro 8111023, & Hugh PerkinsHugh Perkins 2. (n.d.). What are "residual connections" in RNNs? Retrieved from https://stats.stackexchange.com/questions/321054/what-are-residual-connections-in-rnns

- What is the Difference Between a Parameter and a Hyperparameter? (2017, June 15). Retrieved from https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

- Lee, C., & Lee, C. (2017, November 13). Understanding Bidirectional RNN in PyTorch. Retrieved from https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66

- Li, Bo. “A Question Answering System Using Encoder-Decoder, Sequence-To-Sequence, Recurrent Neural Networks.” doi:10.31979/etd.np7w-q8y3.

- Prabhu, and Prabhu. “Understanding of Convolutional Neural Network (CNN) - Deep Learning.” Medium, Medium, 4 Mar. 2018, medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

- “TensorBoard: Visualizing Learning | TensorFlow Core | TensorFlow.” TensorFlow, www.tensorflow.org/guide/summaries_and_tensorboard.

- “Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation.” Towards Data Science, Towards Data Science, 24 Sept. 2018, towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

- “Activation Functions in Neural Networks.” Towards Data Science, Towards Data Science, 6 Sept. 2017, towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

- 1143621442379174. “Understanding GRU Networks.” Towards Data Science, Towards Data Science, 16 Dec. 2017, towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

- Donges, Niklas, and Niklas Donges. “Recurrent Neural Networks and LSTM.” Towards Data Science, Towards Data Science, 25 Feb. 2018, www.towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5

- “Convolution Filter.” Convolution Filter - an Overview | ScienceDirect Topics, www.sciencedirect.com/topics/computer-science/convolution-filter

- Accessing Text Corpora and Lexical Resources, www.nltk.org/book/ch02.html

- “TensorFlow.” TensorFlow, www.tensorflow.org/