Grammatical Error Correction System with Deep Learning

A Project

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Computer Science

By

Jose Silva

2018

SIGNATURE PAGE

PROJECT:	GRAMMATICAL ERROR CORRECTION SYSTEM WITH DEEP LEARNING
AUTHOR:	Jose Silva
DATE SUBMITTED:	Fall 2018 Department of Computer Science

Dr. Yu Sun
Project Committee Chair
Computer Science

_____

Dr. Abdelfattah Amamra
Project Committee
Computer Science

_____

ABSTRACT

Using Natural Language Processing via Deep Learning, we will attempt to implement a context-sensitive spelling error correction system focused on casual text messaging. For example, usual autocorrect systems will not correct a sentence like “An apple is better then a banana” as this is a syntactically correct sentence. However, a correction should be made here to the word “then” and the sentence should be, “An apple is better than a banana.” These types of mistakes are common in people texting and can be found for example in people learning English as a second language.

Natural Language Processing can be summarized as combining the utilization of computation to understand the concepts of natural language and speech. In other words, we can use several techniques found in fields like Computer Science to leverage computers to help understand and manipulate human language and human speech patterns. There are several techniques that can be used, the majority which can be categorized under the field of Machine Learning.

The subfield of Machine Learning that will be utilized is Deep Learning, which is currently leading the innovations in Natural Language Processing. To create these types of models, we need a significant amount of data, and we need a significant amount of computational power that recent advances in computer hardware have provided. When combining current state-of-the-art hardware and the ginormous amount of text data available online, we can leverage this by

potentially creating a correct neural network based on the online text which will help in the processing of a natural language.

The project is to be implemented as follows. A large collection of what is supposed to be grammatically correct sentences of people having a normal conversational will be obtained. This will act as a dataset for the deep learning approach is taken. To artificially increase the set of data we have when we train the sequence model, we will take the approach similar to TensorFlow's sequence to sequence tutorial, which uses Long Short-Term Memory networks. After the model has been created and successfully tested, an API service will be implemented that will be available to use for any third party client application that can communicate via a Representational State Transfer (REST) interface.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Natural Language Processing or NLP has been a field of research for decades that attempts to make computers understand and speak human languages as well as humans do. It has spawned multiple platforms of human to computer interaction such as Amazon’s Alexa, Apple's Siri, spell checkers, sentence summarizers, and many more. For years, researchers attempted to create a language modeling technique that attempted get computers to communicate with humans at a level that humans feel they are communicating with another human. The work done by scientists in the early 1950s is considered the era of machine translation, where being able to treat text and language in general as information allowed the possibility that language might be manipulated on the new digital computers that were then being constructed [1]. This goal eluded researches for decades, and unfortunately still does, however, there have been huge leaps in this research area in the last decade or so that have closed the gap significantly on this elusive goal.

Although the Mathematical/Computer Science techniques have been there since the mid-1900s, there was two limiting factors which made it almost practically impossible to formalize them. The first, a limitation in computer processing power, and the second, a lack of data. Nowadays these limitations have been relaxed as the internet has provided an avenue for people all over the world to share and produce data. According to Jones, it is now possible, with

present computing resources, to ‘run up’ surprisingly powerful systems and to conduct impressively large experiments in a matter of months or even weeks [2]. Companies like Facebook, Google, and Microsoft all have their own data storage warehouses where they store and maintain all kinds of data from people all over the world who use their services. Early on, these companies saw the potential in obtaining all different kinds of data which users voluntarily give up in order to obtain access to these services for free. What they needed was more processing power, which was doubling every two years or so thanks to computer engineering. There was a period since the mid-1900s when according to Moore, the industry was more than doubling the total number of transistors ever made every year, although he also states, The pace has slowed recently but is still on a good growth curve [3]. Once it got to the point where there were enough resources for a large amount of computation, and large amounts of data was available, machine learning began to explode in popularity as several breakthroughs were made in fields like Image Recognition, Medical Diagnosis, Statistical Abridge, and most relevant to this project, Natural Language Processing.

Natural Language Processing can be divided into two main areas, Speech Recognition and Text Recognition. Speech Recognition is considered a different domain of Natural Language Processing as it primarily deals with auditory data and finding a conceptual “meaning” (e.g., what meaning the Speaker intended to convey) of the detected words by analyzing their grammatical relationship and relative context [4]. Data that consists of vocal recordings are used to train

models to recognize what words are being spoken. The techniques to understand and process this audio data are understandably different from what it takes to process text data, and it has its own dedicated research community and conferences just for Speech Processing. Text data is represented differently, however the meaning people derive from it can be the same. In text data, people convey the message by typing and is usually stored in some text file, which is different from the audio file like an mp3.

One task that has become an everyday one for most people around the world is communicating through our phones. Our phones have been replacing our use of computers, so much so that 70% of the world's population own at least one mobile phone [5]. Which means, people are using their phones as a medium to enter their messages they would like to communicate with. The primary way of doing this is through a virtual keyboard found as a feature of their phones. Many of these messages lack grammatical correctness because most of the time the messages in a casual conversation are typed out fast. The current way of correcting issues in grammar is to have the user either trace back their error, highlight it and replace it manually, or the phone can suggest a corrected word replacement. Smartphones are helpful enough to correct the syntax part of the sentence, however they tend to leave the grammar alone as it can be hard to correctly assume whether the user meant what he or she exactly typed, and the nature of some languages tolerate the ambiguity. In the English language, for example, phonetically same sounding words occur that have multiple meanings. Many phones will take a correct spelling word that sounds the same, but is the

incorrect word. This grammatical mistake is extremely common in everyday texting. There is also not much importance given to the matter from a user perspective. This is because the messages usually don't have much importance as the person is usually a friend or family member that won't bat an eye if they see an error as opposed to an important business partner that a user would want to make certain to send a grammatically correct message. For important messages like those, email is usually preferred which has much more capability as email clients usually have more capabilities. This grammar issue is primarily focused on those people who don't give much care to the small common grammatical errors everyday texting is exposed to. Having an automatic solution for this that focuses on small text messages sent can impact numerous users.

Proposed solution: **An intelligent Grammar Error Correction system applicable for everyday messaging trained by a neural network using Deep Learning.**

To address the problem, everyday usage of text messaging is considered and the common ways these messages are structured. As an example, the text message "An apple is better then a banana" would be considered correct syntactically. However, in this message, the person behind the message intended to state "An apple is better than a banana." This would replace the incorrect "then" with "than." The idea behind the solution is using some Python programming, huge sets of data are created for correcting grammar by starting with what is assumed to be grammatically correct samples of data and having the text go through some mutations to create training data. This idea of training data

would consist of input-output matching of data where the input is mutated text and the output be the original intact text. Once this is achieved, the goal is using this dataset, and a sequence-to-sequence modeling to generate a prediction function that can correct these errors. Once these models are generated, it's feasible to act as the backbone of a RESTful service for other technologies to leverage.

CHAPTER 2

Literature Survey

2.1 Natural Language Processing Components

The project displays Deep Learning applied to Natural Language Processing (NLP). Natural Language Processing can be summarized as combining the utilization of computation to understand the concepts of natural language and speech. In other words, several techniques found in fields like computer science can be leveraged for computers to help understand and manipulate human language and human speech patterns. There are several techniques that can be used, the majority which can be categorized under the field of Machine Learning. The subfield of Machine Learning that will be discussed is Deep Learning, which is currently leading the innovations in NLP. Deep Learning can be summarized as a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts [6]. To create these types of models, significant amount of data is needed and computational power that recent advances in computer hardware have provided. When combining current state-of-the-art hardware and the ginormous amount of text data available online, this can be leveraged by potentially creating the correct neural network based on the online text which will help in the processing of a natural language. Natural Language Processing has the potential to be used in virtually every human interaction subfield, such as speech recognition, text processing, UI, machine translation, artificial intelligence,

and much more. Our primary concern will be focused on text data as the data available is substantially larger.

Just like other Machine Learning areas, a neural network can be designed similarly that can be adapted to the processing of raw text data. One important characteristic that text data will encounter when creating a solution however, is that the text data is high dimensional. What is meant by high dimensionality is one word, for example, can mean completely different things. Which means there needs to be an understanding that is much higher than if there were just mapping a single meaning to each word. Manipulating high dimensional objects is hard when it comes to using neural networks.

2.2 Preprocessing

Usual preprocessing steps are used to take some text data in its raw form and transform it into text data that will be more useful for neural network processing. According to researchers, the way data are represented can make a huge difference in the success of a learning algorithm, thus making the correct decisions in organizing this data is critical for the success of the application [7]. So, what is raw text data? Well it could be considered as a long string or a list of token strings. For example, a sentence is considered a long string. Or a paragraph in a book is considered a list of strings. A series of steps must be taken to preprocess the raw data in a format that will be easier to manipulate when feeding these tokens into a neural network. This type of machine learning can be classified as Supervised Machine learning, which can be summarized as

tasks that can be abstracted in terms of (X, Y) pairs, where X is an input random variable and Y is a label that we wish to predict given X [7]. This differs from Unsupervised Machine learning, which mainly differs in that there is that there is no training set for the former and hence, no obvious role for cross-validation [8].

First tokenizing the text is necessary, which means separating the sentence into different valuable tokens. A valuable token can be a word, but it is not limited to just a word. So, for a sentence “I like dogs”, the sentence can be split up with a space delimiter to get “I” “like” “dog” tokens. There are numerous datasets for the English language that are available, so many times, this step has already been done automatically. In many languages it can just be considered the space or some punctuation marks as a good enough separator whenever the preprocessing step has not been done already. However, there are some special cases for example in the location “San Francisco”, that should be considered as a single token so there needs to be some specialized rules that will help in doing this. For example, there can be a separate list of locations if the dataset will deal with several topics involving locations. Or in capitalization, if there is two separate tokens with capitalization like “San Francisco” or first name last name, these two tokens should be considered as one.

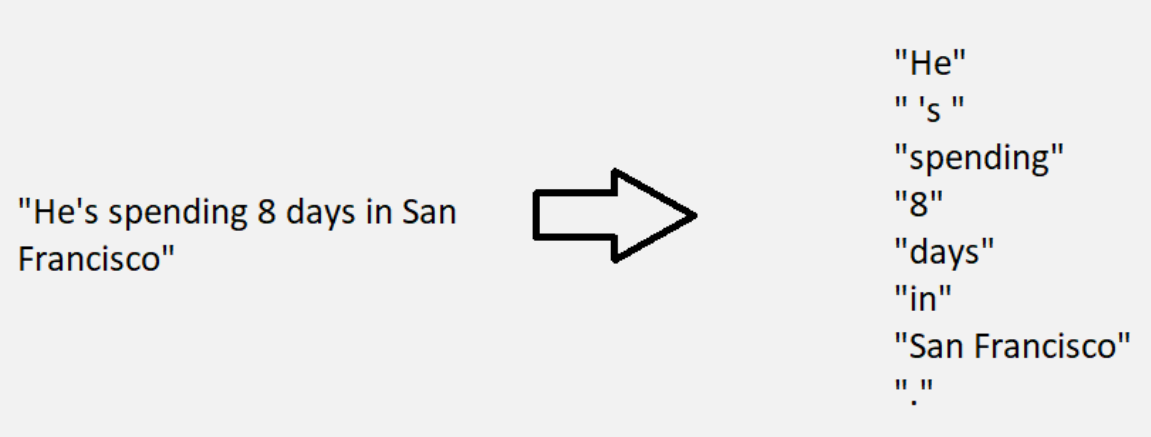

Figure 1: Tokenization of Text

Next step is to lemmatize the tokens. This means the removal of unnecessary information. All the words are standardized and made similar to each other. Words are broken apart so the expression can all be the same, even when the ASCII representation is different like capital and lower-case letters. Numbers are just represented as a number string since they do not add much value to the grammar if they are different numbers. Removal of plurals is necessary and also as they do not change much of the value of the word. The specific lemmatization primarily done depends on the problem and can completely remove variations of words that do not apply to the problem.

The next step is to create a vocabulary convert all the tokens into unique IDs. The way this is done is to form a vocabulary that maps to a specific number. For example, by adding to a list each new word and have the number associated just be an index. Using different criteria to select which words are part of the vocabulary. For example, picking the most frequent words from our dataset, or having a minimum recurring word vocabulary with a threshold, or ignoring some

uninformative words that can define in a short list that add no real value like “the” or “a”. For all the ignored words, a map of the words to an 'out of vocabulary' ID that is ultimately ignored is added to. This “out of vocabulary” or OOV word is placed at the last index in the vocabulary. Typical vocabularies are large, they can vary from between 10000 to 250000 on average.

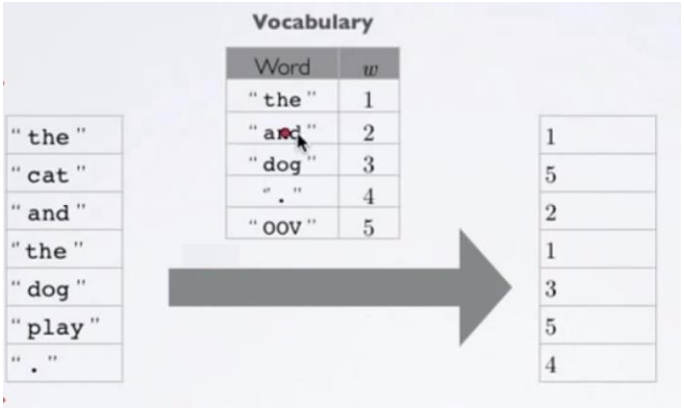

Figure 2: Creating the Vocabulary

2.3 One Hot Encoding

Vocabularies as our representation for words presents multiple problems. The biggest one is taking the index as the input to our vectors, they would have values because they are numbers, and the values would essentially be the order in which they are placed inside. A vocabulary token with index 2 will be automatically closer to a word in index 3 just because of their proximity, even if the words meaning is completely opposite. One hot encoding is a simple way of representing a word as a vector which helps eliminate that bias. They can be sent into a ML algorithm with the confidence that each word will not have any

relation to each other. Out of efficiency, many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric [9]. One hot encoding takes a vector, fills up with zeroes and putting a 1 only at the position that corresponds to the id of a word in the vocabulary. It makes no assumption about the similarity of the word. If the squared Euclidean distance of these two words w and w' are the same, then it will be 0, and the output will be 2 if they are not for every two words that are different. The important thing to notice is that all the words in our vocabulary then are equally different from each other. This is a natural representation to start with, although a poor one.

$$e(w) = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$$

Figure 3: One Hot Encoding Representation

Major problems with one hot encoding revolve around the fact that it is high dimensional. The dimensionality of the vector is the size of the vocabulary. As previously mentioned; the vocabulary size is on average about 100000. So just 10 words using one hot encoding, the decision to concatenate them as a sentence, would give 100000 units. There are 2 main consequences to this. One, is the vulnerability to overfitting, which means millions amount of inputs would need millions of parameters to train a regular neural network. Second, it means powerful computers are needed to compute this because of sparse vectors.

There are ways computers can efficiently compute space vectors, however not all operations can which would eventually cost much computation.
The idea of Continuous word representations is ultimately used as the best way to represent words. Creating a vector representation of a word that is mapped by several attributes. The word will be passed as a parameter to the C(w) function and spit out these types of values. Like ‘Cat’ and ‘Dog’ would have a similar representation based on size or number of legs. With recent progress in machine learning, researchers from google stated that it has become possible to train more complex models on much larger data set, and they typically outperform the simple models, like N-Grams which is discussed below and that the continuous representations are probably the most successful concept is to use distributed representations of words [10].

In theory, these representations can be stored in a matrix for easy access of the values quickly. This can be done by using one hot encoding representation as a way of just stripping out all the other values stored in that matrix of words by attributes. In practice, this would be implemented as a lookup table, so no need for multiplication of the matrix with a vector of a bunch of zeros.

2.4 Language Modeling

The next important problem to cover in NLP is language modeling. A language model is a probabilistic model that assigns probabilities to any sequence of words. It is considered the task of learning a language model that assigns high probabilities to well-formed sentences. Plays a crucial role in

speech and translation systems. If a language model has seen “people like dogs” many times, whenever it runs into a “People like” which is n-2 words, then there is a higher probability that the next word is dog or dogs or anything in our word representation model that is close to dogs. N-gram models is very frequently used. An N-gram is a sequence of n words that are created into several grams with the length of n. Unigrams are n=1 for example, so only one words, Bigrams are pairs of words. Probability will be calculated by n-grams of training data, so a document will be taken apart word by word and paired in bi grams or trigrams. The formula below is how the probability is calculated. It counts how many times it sees the n-gram with a particular word and then divides it with the sum of the same n-gram with any other word to normalize it.

$$P(Wt \mid Wt-(n-1), \ldots, Wt-1) = \frac{COUNT(Wt-(n-1), \ldots, Wt-1, Wt)}{COUNT(Wt-(n-1), \ldots, Wt-1, *)}$$

Figure 4: Probability using N-Grams

N-gram models main issue is the data sparsity problem. It is better for n in n-gram to be large as possible. If for example it is only one, it is assumed that every word is independent since n-1 when n= 1 is zero so there is no previous words to predict. If for example n=2, it is assumed every word is only influenced by the one previous word, with no correlation to any of the n-3 and more words. Note the downside of it being too big is 20 N-gram is likely to not be seen again, it is more likely to see “People like dogs” than “Tall Blue Wide Smart Bold Crazy

Adventurous People like dogs” in another document if this was found and classified before. The next time “People like” is seen the model will have a hard time predicting dogs. The way to help alleviate is a technique known as smoothing, which essentially takes a factorial type of solution in adding the count of all the words together in a gram, then ignoring the first gram and counting all of the times that occurs, then ignoring the first gram of that and so forth.

2.5 Word Tagging

One of the other preprocessing tasks which is helpful to do is something known as word tagging. Raw text is taken and augmented with higher level data like syntactic or semantic data. First thing is part-of-speech tagging, and identify which part of the sentence is which, like if it is a noun or verb. Labeling each of the tokens if possible, and they can use Penn Treebank POS tags, which is a set already defined that many NLP users use. The next thing is known as chunking, which is segmenting phrases into syntactic phrases, like noun phrase or verb phrase. Segments follow a special encoding that gives the neural network some insight on which part of the sentence it is currently dealing with. A Noun phrase can be considered for example the word “He” or a series of words like “the current dog owner.” Verb phrases can be a verb like “fights” or “fights angrily.” Lastly, adding Named Entity Recognition to the words is done, which just takes nouns and labels them as person, location, organization, etc.

2.6 Recursive Neural Network

Using these techniques, it's been possible to classify single words by themselves so far, however the interesting and more useful problem that is faced is classifying phrases of arbitrary length. In other words, how can it compare the phrase "consider" with the phrase "take into account" if they are different lengths and each individual word means totally different things. It is only when combining the three tokens "take, into, account" can it get what will be a vector representation that maps very closely to the word "consider." To tackle this problem a recursive neural network will be used, which essentially builds up a sentence and its meaning from the ground up. It then continues to output a new vector representation of the previous inputs and uses that output as another input for the next iteration, paired with another part of the sentence. To leverage this neural network, two things are necessary. One, a model that merges pairs of representation and two, a model that determines the tree structure.

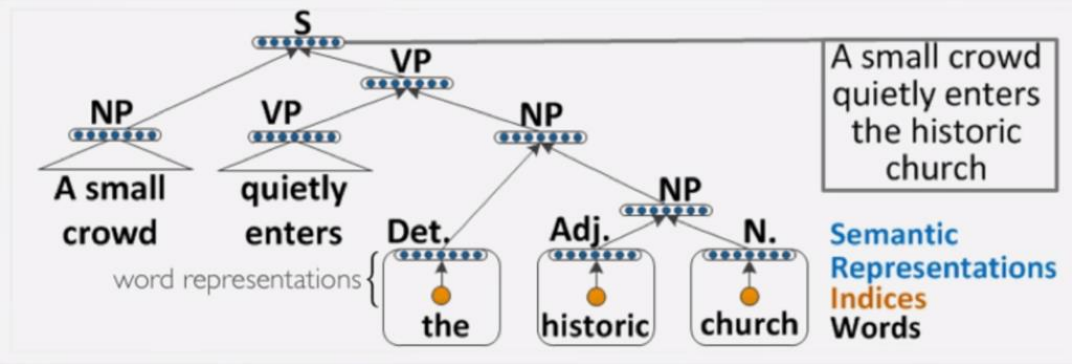

Figure 5: Sentence building with Recursive Neural Network

The neural network used is quite simple, it essentially corresponds to a network where a hidden layer called p and which is connected to the concatenation of the two input vector representations c1 and c2, which are vector representation of words or some other phrase that has already been concatenated before and is used as input in this recursive net. The p vector is going to correspond to the new representation to the node that has c1 and c2 as children in the syntactic tree. The vector will be created by taking the concatenation of children, multiply by matrix w, adding a bias b, and passing it through a nonlinearity f. Another thing that the neural network will compute is a score of the quality of the merge which will be used to eventually decide which syntactic tree will be used.

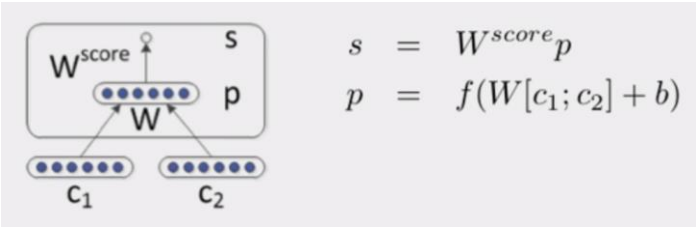

Figure 6: Score function

The last portion of the recursive neural network for phrase representation involves using the score which is computed and calculated an overall score from each possible tree in order to select the best tree which will give the best possible vector representation of the phrase. The score is calculated by adding every output score as the tree is build. This means that every merge will

contribute to the overall score instead of going with the final score. After merging different parts of the sentence in numerous variations, the totals of the scores will be the deciding factor as to what tree will represent the correct recursive network that will help with future phrase representation. According some research by Mikolov, Recurrent neural networks outperformed significantly state of the art backoff models in several experimentations performed [11].

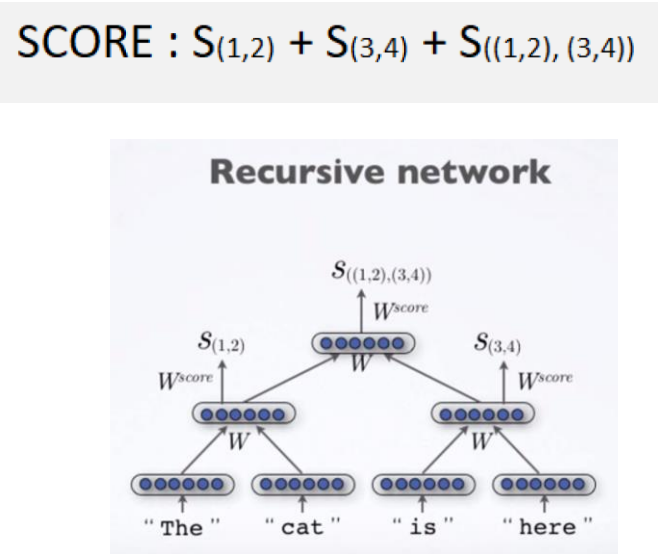

Figure 7: Adding up scores for prediction

2.7 Long short-term memory (LSTM)

Applications of ML have gotten much traction in the last few years in what is known as sequence to sequence space. Long Short-Term Memory units were introduced as an enhancement to Recurrent Neural Networks alone as a shortcoming of the Recurrent Neural networks is that it is only capable of dealing with short-term dependencies. They address this problem by introducing a long-

term memory into the network [13]. This is where future predictions are based on the previous output, and the prediction to the previous output was calculated using the one before that one. The missing entry in the following sentence, “People who live in Brazil love it there, they enjoy the food and they primarily speak the ____ language,” all depends on the sequence that has occurred before it. Much of the English language is formulated such that the context is first stated and the following words describe some characteristic of the context. Humans can go back several sequences and observe the context in this example, and conclude the missing entry is most likely “Portuguese.” Computers a difficulty without a separate algorithm called Long Short-Term Memory Units which formulate a state machine which helps the previous outputs have some impact on future predictions. This technique will help in the grammatical correction since the nature of sentences completely rely on context to be formulated correctly.

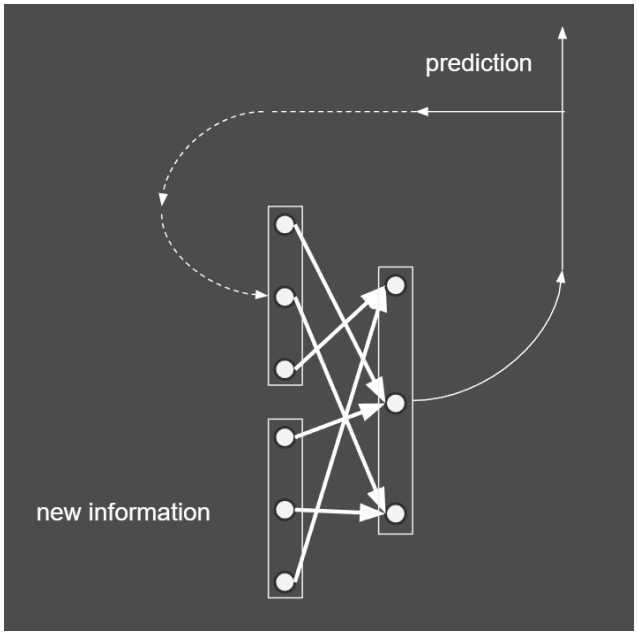

Figure 8: Using previous prediction as new input

In Figure 8, it is shown how the prediction output can be reused for some level of influence in the next prediction. The next prediction is a combination of the previous prediction and the new information that has been passed in. Combining these two, will give us the ability to have some influence for all the new information we have obtained since the beginning.

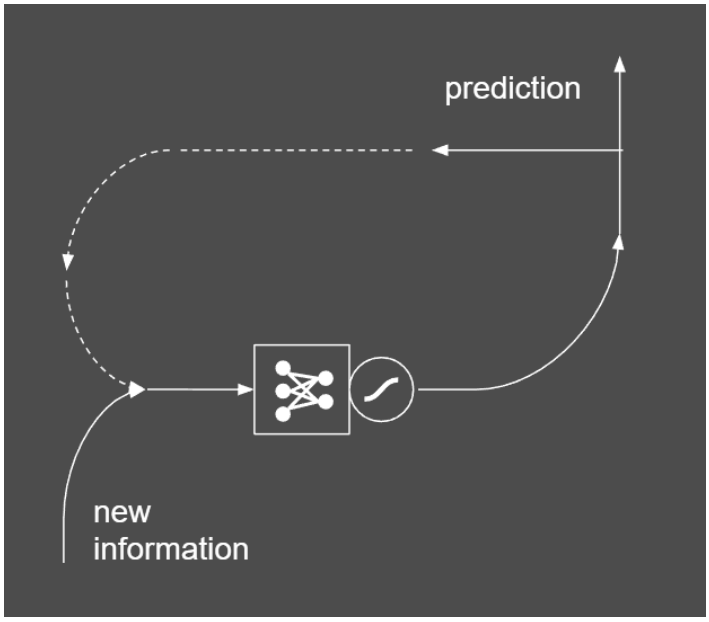

Figure 9: Recurrent Neural Network for LSTM

The next important thing to note in the system is the symbol added after the neural network. This is known as squashing function, and it just helps the network to behave. How it works is we take all of our outputs from the neural networks and normalize their outputted value. For small numbers, the output after the squashing function will end up being very close the original value, however for larger numbers, there will be a hard limit. For example, the squashing function tanh will normalize the output to be between the range of -1 and 1. This is helpful when there is a loop like the system shown above as the same predictions from previous iterations will have too much influence if just one output is greater than one as there is no check on them.

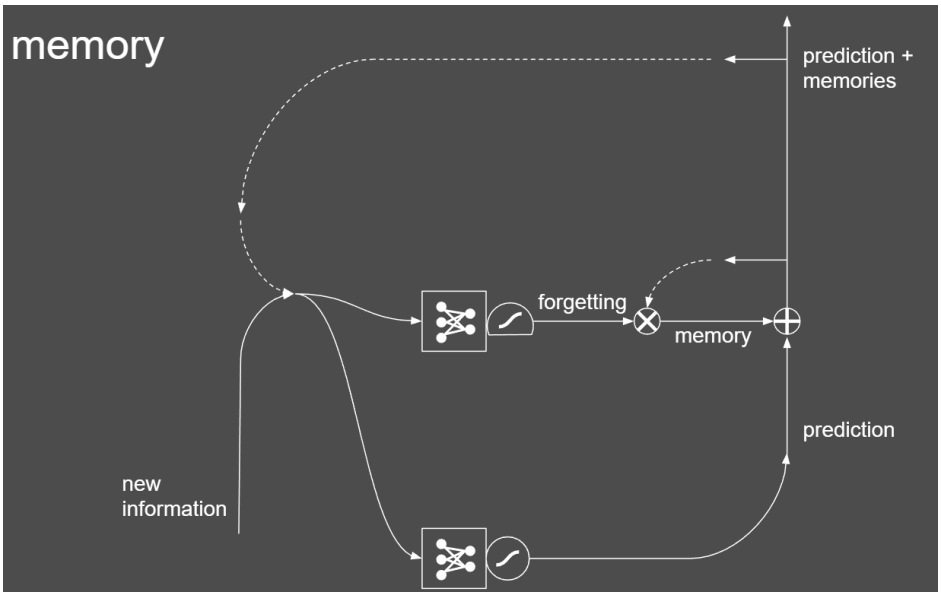

Figure 10: Memory in Recurrent Neural Networks

To not have the same output consistently be generated, there needs to be some level of filter. That is where the X comes in our diagram above. It essentially gives us the ability to forget certain outputs by multiplying them with a vector where the node that needs to be filtered out is equal to 0. Depending on how much the node needs to be filtered out, there can be a decimal value that corresponds to the previous prediction and reduce the same node as input in the new iteration.

Long short-term memory units have special characteristics as shown in previously, which help in dealing with problems that involve sequences. They treat each sentence as a sequence of words and recursively compose each word with its previous memory, until the meaning of the whole sentence has been derived [12]. If there is a problem that deals with consecutive tokens, or deals

with the passage of time, LSTMs help in several applications. They are part of TensorFlow’s feature, which will be taken full advantage of in the solution.

CHAPTER 3

Solution Implementation

3.1 Preprocessing Text Data

Using python as our data scripting language, the model was trained by using Google's TensorFlow. The dataset was increased by getting some snippets of text and created the incorrect/correct pairs by

- Obtaining a randomized sample from our data.
- Randomly mutate some of the characteristics of the sample of data
- Having the untouched original and presumably correct syntax sample be the target.

The random mutations introduced were limited to targeting articles of the sentence, removing the part following a verb contraction, and substituting words that have multiple meanings but the same phonetic sound or pronunciation. This was done 3 times to increase the data and could potentially be done more times, but considering time constraints, three was chosen. The model trained can be classified as a sequence-to-sequence model.

3.2 Decoding

The decoding technique, besides the one the sequence to sequence model suggest, used is say that all previous tokens in the sequence to be decoded is there in the input or is part of the correction set. This set is created when trained and has all tokens in the target, but none in the source, for a single

sample in our data. The idea is there is usually the errors deal with small vocab words that are very common and the predictive function should be focused on correcting these as a priority.

In NLP, dealing with OOV tokens is a must, or Out of Vocabulary tokens. The issue of dealing with these isn't straight forward, but here it is solved by assuming the OOV tokens going into our model is the same as the OOV out of it, then it can be said the correct token to the unknown encountered in the decoding. This seems to be a correct assumption, as the model should never include bad mistakes that make it include a rarely seen token.

Once the predictive function has been generated using the previous characteristics, there was a series of extra tests to assess performance. Once it was deemed to perform well, the next step was a simpler one which was the need to expose this for others to access. The library used to create a simple web API is called Flask. This lightweight framework needed some declarations in the special file called app.py. Using the framework, the project was uploaded to an AWS server to be hosted as their free tier comes with enough free perks to host this project for some time. Once uploaded, the Flask framework in combination with python creates a simple web API endpoint where a client application can send a string and the model internally will make appropriate corrections if deemed necessary then return the string in the response.

3.3 Developmental Challenges

There were numerous challenges while developing this project. There was the challenge of learning a different side of python that is primarily used in data science and machine leraning. There was the option of using the R programming language, but there was less familiarity there. The Machine Learning Framework that was decided on was Google's TensorFlow. TensorFlow is an open source library that numerous machine learning applications used as their training framework. It uses numerous researched algorithms as tools for developers to pick and choose which is the best algorithm to use for a specific problem. As previously mentioned before, this project was based on TensorFlow's sequece-to-sequence model. Besides, the preprocessing steps of the data mentioned earlier, there was the challenge of choosing a web service which helped train and test the model. As oppose to purchasing the hardware, the FloydHub was decided on because of their free intro pricing model, which satisfied the requirements for this simple project.

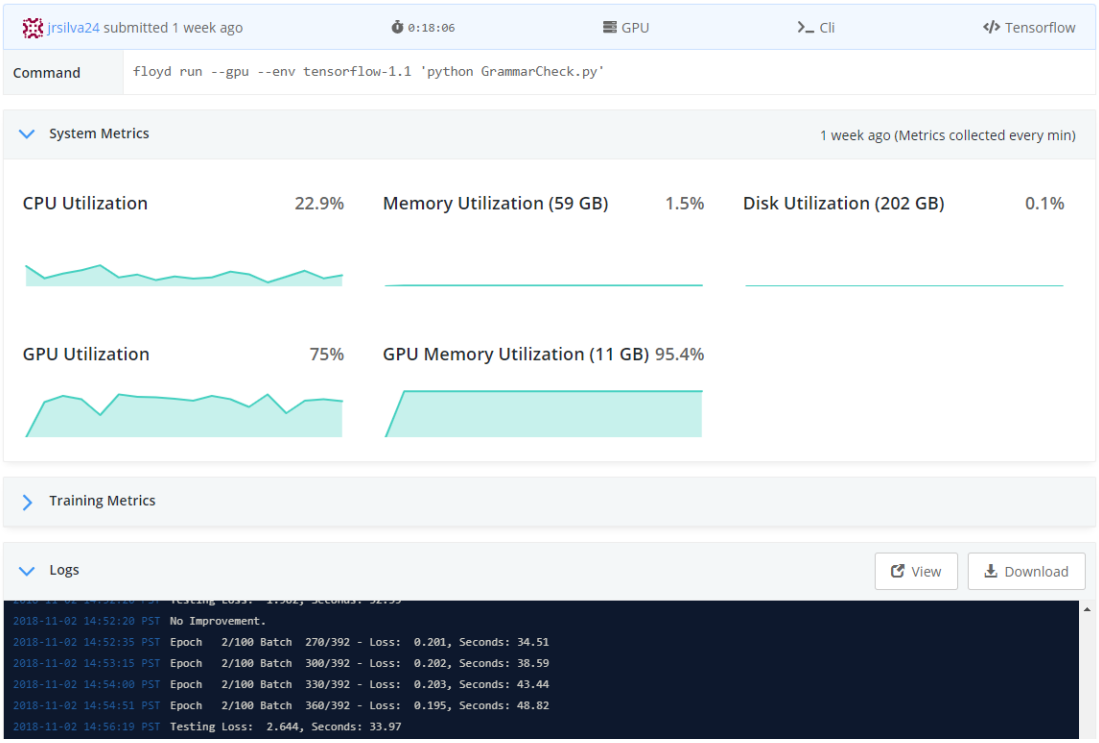

Figure 11: FloydHub Intuitive Interface

FloydHub is a mask behind AWS machines that provide a more intuitive interface when entering the Machine Learning space. After several tweaks and attempts, the model was trained after an hour or so on one of their higher tear GPU clusters. There was the attempt to train this same model on a MacBook air, but the first batch only completed overnight, which meant a total of 30 or so days would've been the total time to train the same model. A smoke test was performed, and unfortunately did not fare as well as it should've, but still performed well enough to what it set out to do. The next challenge was making this model available via REST. The framework Flask for python made this simple and the endpoint was now available for clients to use.

Out of previous experience, ASP.Net was chosen as the framework to implement a client side web application that just calls the REST endpoint. Using C# as the server side language, the application is able to make a http request without worrying about cross site scripting. It retrieves the json from the endpoint and generates an CSHTML page, which is only an html page that asp.net uses to embed C# code for generating these web pages dynamically. After the basic output was exposed, the project was finished with some CSS/JavaScript visual aesthetics and was deployed to Azure web services as it has easy integration.

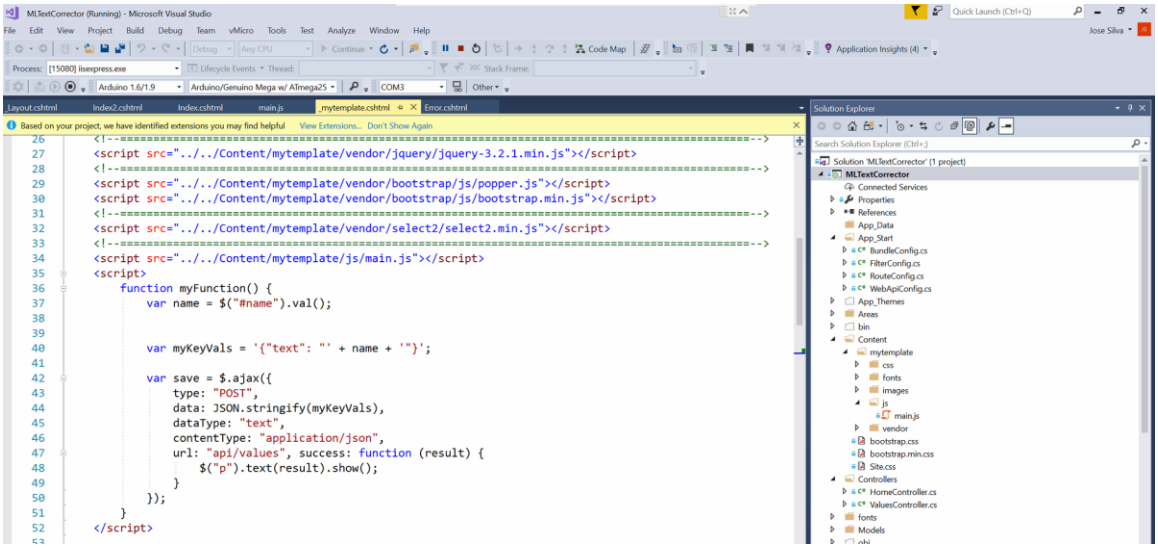

Figure 12: ASP.NET with Visual Studio

3.4 Application Overview

The project can be found at https://grammarcorrector.azurewebsites.net. Here there will be a form which asks for a sentence to send to the server. Using Ajax, the form is submitted using the Enter button and it is retrieved in the server side. After sending the request to the Rest API endpoint where the model is

hosted, it returns and updates the web page with a paragraph tag that shows what the model believed is correct, which may or may not be an alteration of the sentence. It shows this with an appended “Corrected Text ::” for clarity

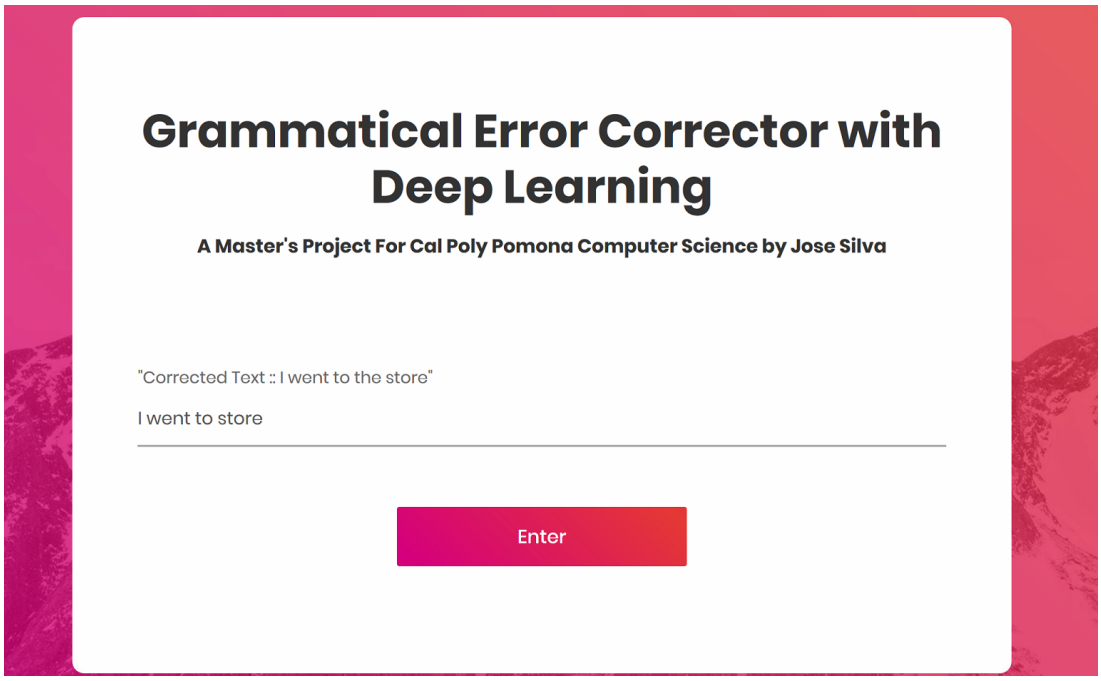

Figure 13: Grammatical Error Corrector Interface

CHAPTER 4

Conclusion and Future Work

The implementation of this model showed to be effective in catching several of the mistakes in the small text samples. However, there are many it did not do well on, which implies there is clear room for improvement. Since English is a complex language filled with several dimensions depending on context, there might not ever be a computer system which can fully be catch 100 percent of the errors all the time. Humans themselves struggle to keep up with all the new lingo and different ways words gain or lose meanings every day. However, there can be a more accurate model than the one generated and several approaches can be taken.

With all the current research in Natural Language Processing, there can be a revolutionary new way of doing text NLP which does not revolve around Deep Learning. If and when it comes to be, the algorithm used to train the dataset should reflect it. Although a more practical approach is to find a greater amount of data than the one used in this model and have many more iterations of training. The key is to find data which is known to be grammatically correct, which can be difficult to find. Many published bodies of text are a good resource for this type of data, but it is important to stay away from non-reliable data which includes things like blogs, or twitter feeds, or anything where there is no verification of correctness. With the additional data, and additional time training the new data, a model should be produced with higher accuracy.

The application itself for additional data will soon ask for user feedback. This can be used to add additional data to the dataset with examples that users have corrected themselves. Of course the additional challenge will be identifying incorrect user corrections, but this additional dataset is a good tradeoff. As more data is generated every day by users all around the world, more datasets will become available, and if more datasets are available, machine learning algorithms will continue to thrive and create incredible systems as they have been in recent times.

REFERENCES

[1] Wendy, Lehnert, and Martin Ringle. *Strategies For Natural Language Processing*. Psychology Press, 2014.

[2] Jones, K. (2018). *Natural language processing: a historical review*. [ebook] Cambrdige. Available at: https://www.cl.cam.ac.uk/archive/ksj21/histdw4.pdf [Accessed 13 Nov. 2018].

[3] Moore, Gordon (2006). "Chapter 7: Moore's law at 40". In Brock, David. Understanding Moore's Law: Four Decades of Innovation (PDF). Chemical Heritage Foundation. pp. 67–84. ISBN 0-941901-41-6. RetrievedMarch 15, 2015.

[4] Weber, D. (2002). *Object Interactive User Interface Using Speech Recognition and Natural Language Processing*. San Diego. Available at: https://patentimages.storage.googleapis.com/96/09/99/8c8bcf49a60059/US6434524.pdf [Accessed 14 Nov. 2018].

[5] Osman, M., Talib, A., Sanusi, Z., Shiang-Yen, T. and Alwi, A. (2012). *A study of the trend of smartphone and its usage behavior in Malaysia*. [Accessed 14 Nov. 2018].

[6] Goodfellow,Bengio, Courville et al. *Deep Learning*. The MIT Press, 2017.

[7] Bengio, Yoshua. “Deep Learning of Representations for Unsupervised and Transfer Learning.” *Proceedings of Machine Learning Research*, 2012, proceedings.mlr.press/v27/bengio12a/bengio12a.pdf.

[8] Gentleman R., Carey V.J. (2008) Unsupervised Machine Learning. In: Bioconductor Case Studies. Use R!. Springer, New York, NY

[9] Brownlee, Jason. “Why One-Hot Encode Data in Machine Learning?” *Machine Learning Mastery*, 19 May 2018, machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/.

[10] Mikolov, Tomas, et al. “Efficient Estimation of Word Representations in Vector Space.” *Cornell University Library*, Cornell University, 7 Sept. 2013, arxiv.org/pdf/1301.3781.pdf.

[11] Mikolov, Tomas. “Recurrent Neural Network Based Language Model.” *International Speech Communication Association*, 2010, www.isca-speech.org/archive/archive_papers/interspeech_2010/i10_1045.pdf.

[12] Cheng, Jianpeng. “Long Short-Term Memory-Networks for Machine Reading.” *Cornell University Library*, Cornell University, 20 Sept. 2016, arxiv.org/pdf/1601.06733.pdf.

[13] Miedema, Fenna. “Sentiment Analysis with Long Short-Term Memory Networks.” *VRIJE UNIVERSITEIT AMSTERDAM*, 1 Aug. 2018, beta.vu.nl/nl/Images/werkstuk-miedema_tcm235-895557.pdf.