Abstract

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. RNN works for machine translation for hundred thousands of words and long sentences and this can be used for the correction for sentences as well. In this project we have tried to train a RNN for finding the spelling mistakes in sentences. English spelling can be a source of great frustration for people who are constantly engaged in their devices. Chatting and informal language use has reduced proper spelling usage. These habits have played a role in shaping up difficulties when it comes to precise, clear and proper spelling usage. This improper writing habit also affects reading abilities. Taking examples from about more than fifteen textbooks, we have tried to train our RNN to reduce text errors as much as possible.

Introduction

Hand and brain coordination is very important to avoid typos. It is better to write lesser words at a slower pace than type fast but end up with a lot of typos. However, it is something that is happening to the majority of the population and needs to be addressed.
Some examples and areas where the project will bring corrections are:
Writng is an art, which is **wyh** we need to write.
The correct form is:
Writing is an art, which is why we need to write.

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.

Related Works

a. Deep Learning for Spell Checking

Tal Weiss first implemented his spelling corrector years ago based on Peter Norvig's excellent tutorial — a spelling corrector in 21 lines of Python code which was not so good.

He then piled on double metaphone phonetic similarity, unicode support, multi-word expressions, weighted Damerau-Levenshtein edit-distance, efficient Trie and smart caching. Finally, he tried a different approach. Deep Learning , a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers. The steps implemented were:

Build an artificial network of sufficient "computational power".

Feed it a lot of data until it figures out what to do with it.

b. Creating a Spell Checker with TensorFlow

The objective of this project was to build a model that can take a sentence with spelling mistakes as input, and output the same sentence, but with the mistakes corrected. The model was designed using grid search to find the optimal architecture, and hyperparameter values. The best results, as measured by sequence loss with 15% of our data. This project has been made using Python 3 and TensorFlow 1.1.

c. A Simple Spell Checker Built From Word Vectors

Ed Rushton proposed a system in which you can use the same word vector algebra to fix spelling mistakes. He has done this by building a lookup table containing the most common spelling mistakes and their corrections using the pre-trained GloVe vectors from Stanford.

Computers only deal with numbers, and so to get a computer to analyse text data. A 'word vector' is simply a set of numbers which represent a word: the computer's internal representation of that word. If we train a computer to predict the missing word from a sentence, giving it millions of examples to learn from, and we allow the computer to improve its predictions by changing the numbers allocated to each word, we find that synonyms end up being allocated numbers that are close to one another.

Methods

Software Specifications

Front End Tools:

1. Programming language - Python 3.6.8
2. Operating System - Ubuntu linux,Windows

Back End Tools :

1. Jupyter Notebook
2. Anaconda

Libraries Used

1. Numpy: It is the fundamental package for scientific computing with Python and can be used as an efficient multi-dimensional container of generic data.
2. Pandas : It takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data frame that looks very similar to table in a statistica software.
3. Random : It generates a random float uniformly in the semi-open range [0.0, 1.0)
4. Tensorflow: TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. It comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.
5. Skilearn: Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
6. Listdir- The method listdir() returns a list containing the names of the entries in the directory given by path.
7. Namedtuples- is factory function for creating tuple classes with named fields.

8. Dense- Dense implements the operation: output = activation(dot(input, kernel) + bias) where kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).
9. Time- it allows time access and conversions.
10. Re- is used to import regular expressions into the file.

Methodology

Dataset

The data is composed of popular books from Project Gutenberg. We have incorporated about 10 books for this project and we have also added a 5-10 new books which we have selected ourselves for project. All these books have been stored in data folder and are in Rich Text Format. We have considered books of different types. Few of the books are:

1. David_Copperfield_by_Charles_Dickens
2. The_Adventures_of_Sherlock_Holmes_by_Arthur_Conan_Doyle
3. Catcher-in-the-Rye
4. Think_and_grow_rich_by_Napolean_Hill
5. The_Picture_of_Dorian_Gray_by_Oscar_Wilde

Of the many words and sentences in this dataset. We have only considered sentences form range of 10 to 95 characters avoiding very long and too short sentences which will not be very effective to train our data and choosing of this sentences have been made so that we can select only the correct amount of data as training on large sentences will prove to be very challenging.

The data set has been split into 85% and 15% in which, 85% of the data has been used to train our network and 15% of the data has been used in testing.

Rnn and LSTM

The term "recurrent neural network" is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored state, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units.

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.

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). You can see an illustration of a RNN with its three gates below:

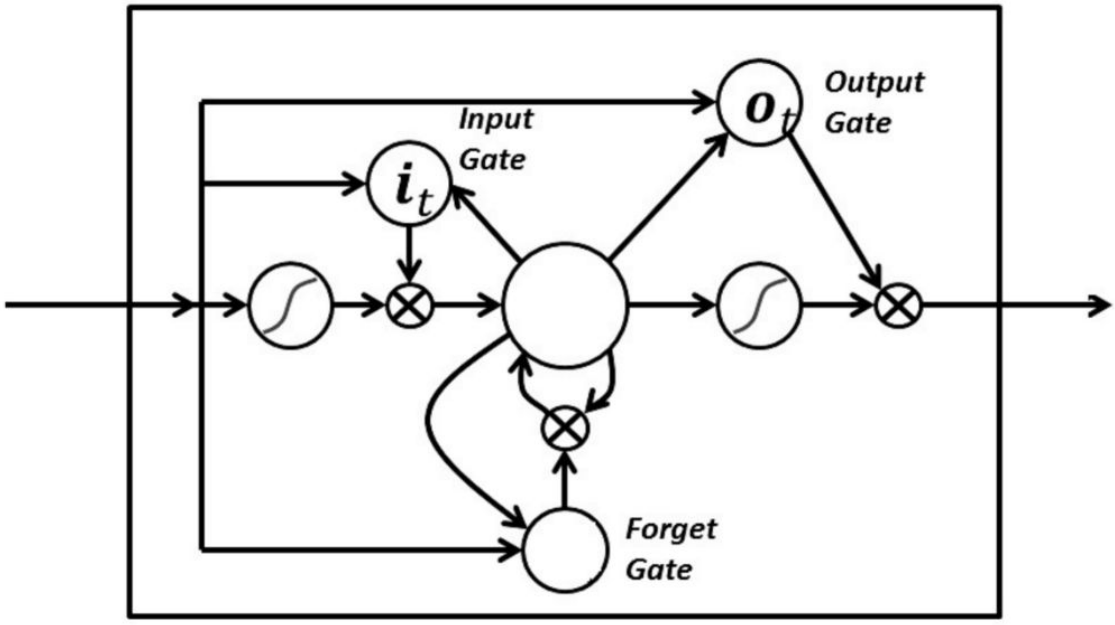

Fig. LSTM Network.

Preprocessing of Dataset

First of all we replace contractions with their longer forms and remove any unwanted characters. (this step needs to be done after replacing the contractions because apostrophes will be removed. Notice the backward slash before the hyphen. Without this slash, all characters that are 'between' the characters before and after the hyphen would be removed. The stop words will only be removed from the descriptions. They are not very relevant in training the model, so by removing them we are able to train the model faster because there is less data. To make things a little more organized, we have put all of the books that we will use in their own folder, called "data". To clean the text of these books is rather simple. Since we will be using characters instead of words as the input to our model, we do not need to worry about removing stop words, or shorten words down to their stems. We only need to remove the characters that we do not want to include and extra spaces.

We are going to limit our vocabulary to words that are either in CN or occur more than 20 times in our dataset. This will allow us to have very good embeddings for every word because the model can better understand how words are related when they see them more times.

To track the performance of this model, we have split the data into a training and testing set. The testing set composed of 15% of the data.We have sorted the data by length. This results in sentences of a batch being of similar length, thus less padding is used, and the model will train faster.

Making Noise

Perhaps the most interesting/important part of this project is the function that will convert the sentences to sentences with mistakes, which will be used as the input data. Mistakes are created within this function in one of three ways:

the order of two characters will be swapped (hlelo ~hello)

an extra letter will be added (heljlo ~ hello)

a character will not be typed (helo ~hello)

The likelihood of either of the three errors occurring is equal, and the likelihood of any error occurring is 5%. Therefore, one in every 20 characters, on average, will include a mistake.

Function Used:

Numpy.random.uniform

Draw samples from a uniform distribution.Samples are uniformly distributed over the half-open interval [low, high) (includes low, but excludes high). In other words, any value within the given interval is equally likely to be drawn by uniform.

Building the model

Placeholders are used in creating our model. So far we have used Variables to manage our data, but there is a more basic structure, the placeholder. A placeholder is simply a variable that we will assign data to at a later date. It allows us to create our operations and build our computation graph, without needing the data. In TensorFlow terminology, we then feed data into the graph through these placeholders.

The Encoder-Decoder LSTM is a recurrent neural network this has been designed to address sequence-to-sequence problems, sometimes called seq2seq.
Sequence-to-sequence prediction problems are challenging because the number of items in the input and output sequences can vary. For example, text translation and learning to execute programs are examples of seq2seq problems.

Why do we need sequence models when we already have feedforward networks and CNN? The problem with these models is that they perform poorly when given a sequence of data. An example would be a sentence in English which contains a sequence of words. Feedforward networks and CNN take a fixed length as input, but, when you look at sentences, not all are of the same length. You could overcome this issue by padding all the inputs to a fixed size. However, they would still perform worse than an RNN because those conventional models do not understand the context of the given input. This is where the major difference between sequence models and feedforward models lies. Given a sentence, when looking at a word, sequence models try to derive relations from the previous words in the same sentence. This is similar to how humans think as well. When we are reading a sentence, we don't start from scratch every time we encounter a new word. We process each word based on the understanding of the previous words we have read.

a. Recurrent Neural Network

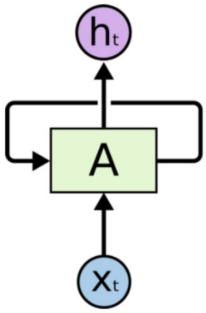

Fig. A Recurrent Neural Network

The recurrent neural network is represented as shown in the above figure. Each node at a time step takes an input from the previous node and this can be represented using a feedback loop. We can unfurl this feedback loop and represent it as shown in the figure below. At each time step, we take an input x_i and a_i-1(output of the previous node) and perform computation on it and produce an output h_i. This output is taken and given to the next node. This process continues until all the time steps are evaluated.

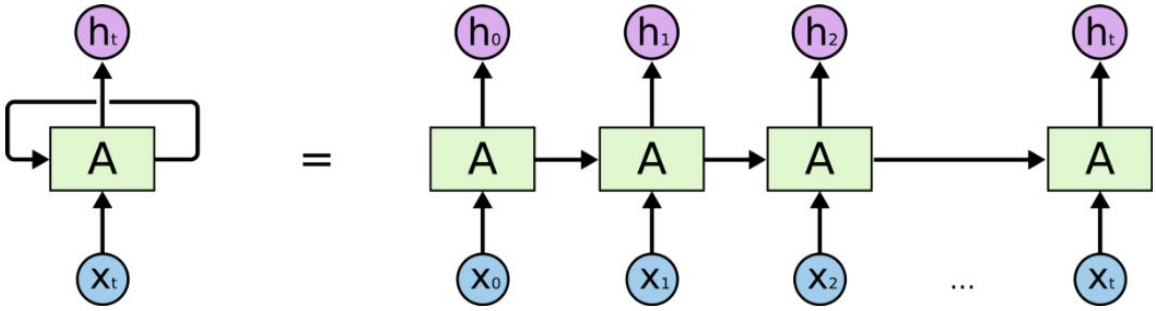

Fig. Recurrent Neural Network

Let a_t represent the ouput from the previous node

$$a_t = f(h_{t-1}, x_t)$$

$$g(x) = \tanh x$$

$$a_t = g(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

$$a_t = \tanh W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t$$

$$h_t = W_{hy} \cdot a_t$$

b. LSTM

The disadvantage with RNN is that as the time steps increase, it fails to derive context from time steps which are much far behind.

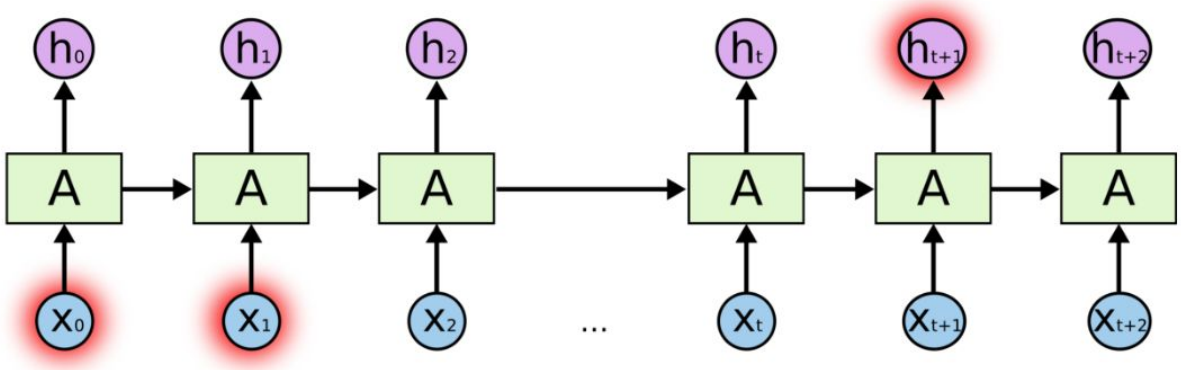

Fig. Problem with RNN

To understand the context at time step t+1, we might need to know the representations from time steps 0 and 1. But, since they are so far behind, their learned representations cannot travel far ahead to influence at time step t+1. Ex: “I grew up in France ….. I speak fluent French”, to understand that you speak French, the network has to look far behind. But, it is not able to do so and this problem can be attributed to the cause of vanishing gradients. Therefore, RNN is able to remember only short-term memory sequences.

To solve this problem, a new kind of network was introduced by Hochreiter & Schmidhuber called Long Short-Term Memory.Enhancing the repeating module enables the LSTM network to remember long-term dependencies.

c. Bidirectional Recurrent Neural Network

A major issue with all of the above networks is that they learn representations from previous time steps. Sometimes, you might have to learn representations from future time steps to better understand the context and eliminate ambiguity.

```
tf.nn.bidirectional_dynamic_rnn(
    cell_fw,
    cell_bw,
    inputs,
    sequence_length=None,
    initial_state_fw=None,
```

```
    initial_state_bw=None,
    dtype=None,
    parallel_iterations=None,
    swap_memory=False,
    time_major=False,
    scope=None
)
```

Creates a dynamic version of bidirectional recurrent neural network.

Takes input and builds independent forward and backward RNNs. The input_size of forward and backward cell must match. The initial state for both directions is zero by default (but can be set optionally) and no intermediate states are ever returned -- the network is fully unrolled for the given (passed in) length(s) of the sequence(s) or completely unrolled if length(s) is not given.

d. Sequence2Sequence model

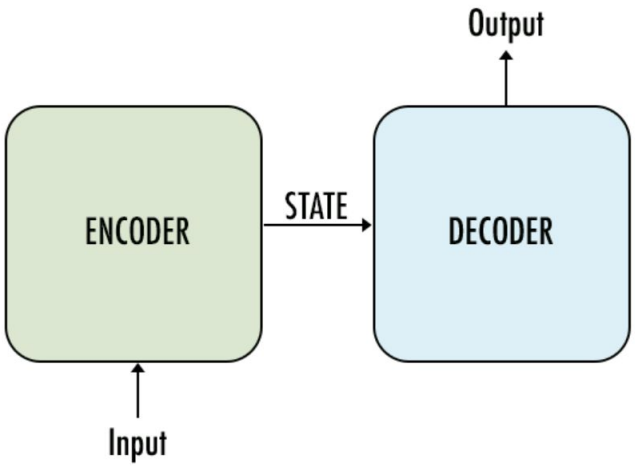

Fig. Sequence2Sequence model

It mainly has two components i.e encoder and decoder, and hence sometimes it is called the Encoder-Decoder Network.

Encoder: It uses deep neural network layers and converts the input words to corresponding hidden vectors. Each vector represents the current word and the context of the word.

Decoder: It is similar to the encoder. It takes as input the hidden vector generated by encoder, its own hidden states and current word to produce the next hidden vector and finally predict the next word.

Attention: The input to the decoder is a single vector which has to store all the information about the context. This becomes a problem with large sequences. Hence the attention mechanism is applied which allows the decoder to look at the input sequence selectively.

Training Process

The choice of optimization algorithm for the deep learning model can mean difference between good results in minutes,hours and days.

Optimizer-Adam Optimizer

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. Stochastic gradient descent maintains a single learning rate for all weight updates and the learning radt does not change during training. A learning rate is maintained for each network weight(parameter) and separately adapted a learning unfolds. Adam in a combination of advantages of two other extensions of stochastic gradient descent(Adaptive Gradient Algorithm and Root Mean Square Propagation) and provides an optimization algorithm that can handle sparse gradients on noisy problems. Adam is relatively easy to configure where the default configuration parameters do well on most problems. Tensorflow library has default learning rate of 0.001. In our case learning rate is 0.0005 because our dataset is limited.

Batch size and Epoch

Epochs

One Epoch is when an entire data set is passed forward and backward through the neural network only once. Since, we are using a limited dataset and to optimise the learning and the graph we are using Gradient Descent which is an iterative process. So, updating the weights with single pass or one epoch is not enough. As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from under fitting to optimal to overfitting curve.

Batch

Since, one epoch is too big to feed to the computer at once we divide it in several smaller batches. Batch Size is the total number of training examples present in a single batch.

Results

Our model is able to correct the sentences but it cannot correct all the sentences. The mistake must be in accordance with the noise we have included in the model. This is because of low computational power and lack of data.

Some of the outputs generate from our model are listed below:

```
INFO:tensorflow:Restoring parameters from ./kp=0.75,nl=2,th=0.95.ckpt

Text
  Word Ids:    [58, 4, 9, 9, 7, 5, 23, 6, 6, 19, 7, 5, 5, 19, 21, 4, 0, 0, 7, 4, 10, 23, 19, 7, 6, 19, 4, 19, 21, 4, 1, 1, 2
3, 0, 66, 19, 13, 2, 19, 8, 24, 4, 31, 5, 8, 23, 43, 19]
  Input Words: Happiness inn marriage is a matterq of chaxnce.

Summary
  Word Ids:       [58, 4, 9, 9, 7, 5, 23, 6, 6, 19, 7, 5, 19, 21, 4, 0, 0, 7, 4, 10, 23, 19, 7, 6, 19, 4, 19, 21, 4, 1, 1, 2
3, 0, 19, 13, 2, 19, 8, 24, 4, 5, 8, 23, 43, 80]
  Response Words: Happiness in marriage is a matter of chance.<EOS>
```

Fig. Correct Result

```
INFO:tensorflow:Restoring parameters from ./kp=0.75,nl=2,th=0.95.ckpt

Text
  Word Ids:    [54, 5, 23, 19, 8, 4, 5, 5, 13, 1, 19, 30, 5, 28, 13, 19, 28, 24, 1, 4, 19, 4, 19, 21, 4, 5, 19, 0, 23, 4, 2
0, 20, 39, 19, 7, 6, 19, 17, 39, 19, 1, 24, 23, 19, 23, 5, 22, 19, 13, 2, 19, 4, 19, 2, 13, 0, 1, 5, 7, 10, 24, 1, 43, 19]
  Input Words: One cannot knwo whta a man really is by the end of a fortnight.

Summary
  Word Ids:       [54, 5, 23, 19, 8, 4, 5, 5, 13, 1, 19, 28, 4, 19, 4, 19, 21, 23, 4, 20, 20, 39, 19, 7, 6, 19, 1, 24, 23, 1
9, 23, 5, 22, 19, 13, 2, 19, 4, 19, 2, 13, 0, 1, 7, 5, 10, 24, 1, 43, 80]
  Response Words: One cannot wa a meally is the end of a fortinght.<EOS>
```

Fig. Incorrect Result

Shortcomings and Difficulties

1. Due to limitation in computational power of our laptop we are unable to train all the available dataset.
2. All the spelling mistake are not corrected by this spell-checker. In fact it can only correct those mistakes which are in accordance with noise_maker.

Conclusion

Spelling mistakes are a common practice when it comes to typing. Frequent re-checks are almost inevitable while we are typing anything. Hence, spelling correction is one of the most popular and successful applications of AI. In the mini project, we have demonstrated the spelling correction application using Tensorflow. It has its shortcomings and is a simple project but through the development of this application, we got to be familiar with the development of projects in ML. Moreover, this project was a fruitful one.

References:

1. https://www.tensorflow.org/api_docs/python(Tensorflow).Last accessed: 2019-01-05
2. https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5(Rnn and lstm).Last accessed:2019-01-05
3. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/(Recurrent neural networks).Last accessed2019-01-09
4. https://github.com/barrust/pyspellchecker(Spell Checker).Last accessed:2019-01-21
5. http://aclweb.org/anthology/J15-1011(Spelling Error Pattern).Last accessed2019-01-10
6. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/(Adam Optimizer).Last accessed:2019-01-22