Text generation for travel blogs

I. Definition

1.1 Project Overview

Many travellers find very little time to blog on-the-go. For some travellers, blogging is one of the means to earn money to fund their future travel. Is it possible to automate travel blogging with artificial intelligence? Wouldn't it be great if your AI assistant helped you keep memories of your travels by automatically writing down stories about the places you visited? It turns out that this task could be tackled using deep learning because there is plenty of data available online.

1.2 Problem Statement

This problem belongs to the sequence prediction category and is very different from regression or classification. It requires that we take the order of observations into account and use recurrent neural networks (RNNs) which have memory and can learn any temporal dependence between observations. A category of RNN called Long Short-Term Memory (LSTM) helps us achieve this.

The input to the LSTM model will be a text file (around 7MB) containing the blog content of around 1000 blogs from 2 top travel bloggers. It comes to about 6.5 million characters. The output consists of 4 different paragraph suggestions for a sentence (with 40 characters), as shown below:

```
----- diversity: 0.2
----- Generating with seed: "How, right? Just like the industry convi"
How, right? Just like the industry convinces that I was still able to stay at the continue to
the projects and the rest of the country is all of the most people that I was traveling and
the same projects that I was able to stay at a simple story of the part of the country and the
start of the state of the street of the country and the state of the street of the travel
company and the staff we had a chance to start a lot of country that
----- diversity: 0.5
----- Generating with seed: "How, right? Just like the industry convi"
How, right? Just like the industry convince that I was still only feeling a few hours and
travel in the day that any of the most important partnerships and staying in the end of the
```

```
photos of the street of the world, which is all of the day that I can be as what I would
always stay in the above and say that I do start about their life of the city, the real
exciting of food is a check of the streets of the moment with the destinations wh
----- diversity: 1.0
----- Generating with seed: "How, right? Just like the industry convi"
How, right? Just like the industry convince cares (ams in this country. Its conditions when
you do no think Sometime sand from the video! I have all kind of drinks, sometimes a result,
we were in North Korea of Hibapaly of Keare:I thought amazing normally smages the difference
the grade to profemmed blown whenever I do what I never ever standing to try and say Hillorip
with we never oble-trip to a third country that was the 19th century
----- diversity: 1.2
----- Generating with seed: "How, right? Just like the industry convi"
How, right? Just like the industry convinces  people the oub list is to above?With my calance
we spent countless.And since they will be develoved for give quitn out and feeling I have
their major geal with these.My friends,dnewnt off an Mrcguent. But it could eorned their hopes
about the case as I was wording a plan of the regard as a mext-able to take my opie of
returning at Nets  etiquetist especially watch breef sclote. The taxis has
```

We can see that results are not perfect but not too bad either. These sentences might not be in the original data set and thus the network will generate new ones on its own.

1.3 Evaluation Metrics

I have used perplexity as an evaluation metric. In natural language processing, perplexity is a way of evaluating language models. A language model is a probability distribution over entire sentences or texts. A low perplexity indicates the probability distribution is good at predicting the sample. The perplexity score can be calculated by the below formula:

$$2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

II. Analysis

2.1 Data Exploration

The travel blogs crawled have been written with Wordpress and they share almost the same HTML structure. I have created my own python crawler, which crawls top 2 travel blogs and does a basic level of filtering by removing javascript and style content that usually gets mixed with the text.

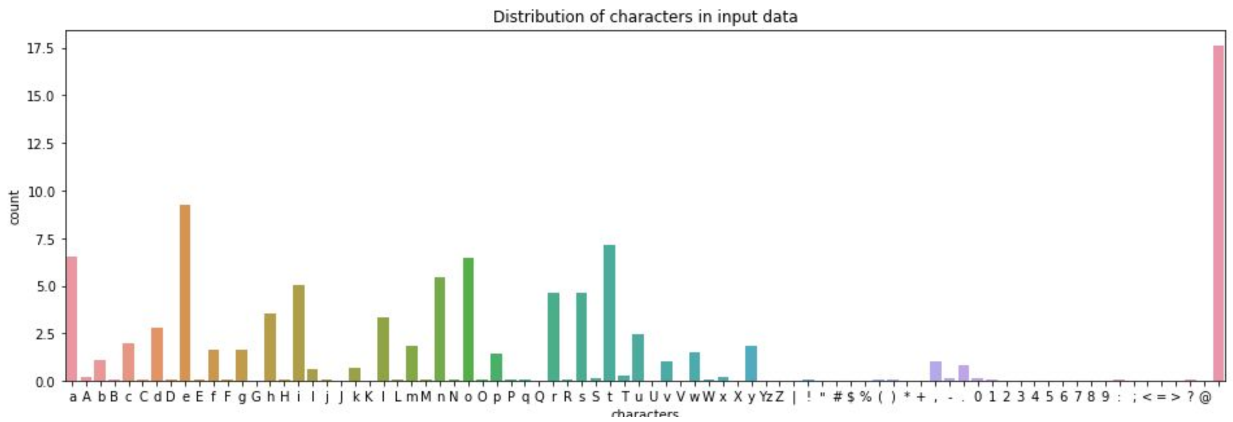
The below table gives an idea of the number of blogs crawled and the amount content extracted.

Sl no.	Travel blogs	Total blogs crawled	Size
1	Wandering Earl	40 pages, 10 blogs per page = 400	2.3 MB
2	Nomadic Matt	100 pages, 6 blogs per page = 600	4.5 MB

A total of around **1000 blogs** have been crawled amounting to about **6.5 million characters (7 MB of text).**

2.2 Exploratory Visualization

The below figure illustrates the distribution of each character present in the input text.



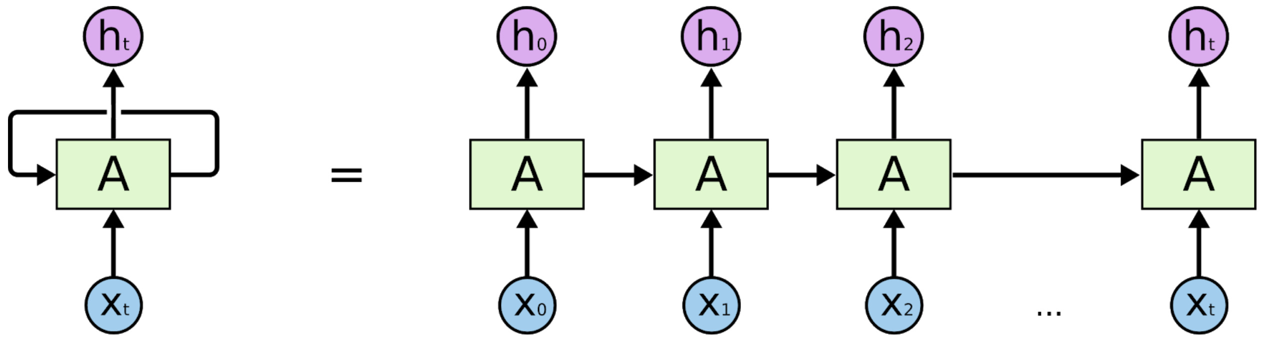
2.3 Algorithms and Techniques

Recurrent Neural Networks

Humans don’t start their thinking from scratch every second. As you read this, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.

However, traditional neural networks can’t do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



An unrolled recurrent neural network.

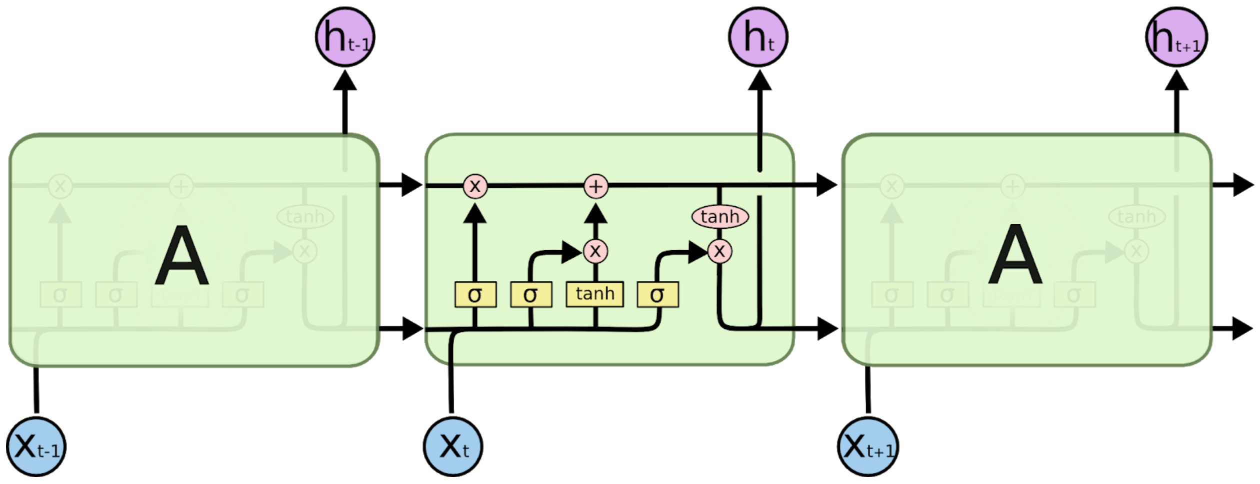
In the above diagram, a chunk of neural network, *A*, looks at some input *xt* and outputs a value *ht*. This allows information to be passed from one step of the network to the next. This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They’re the natural architecture of neural network to use for such data.

RNNs can learn from short-term dependencies. For long-term dependencies such as the text generation for travel blogs used in the current project, we have to use a variant of RNN called LSTM. There have been incredible success applying LSTM to a variety of problems: speech recognition, language modeling, translation, image captioning… the list goes on. Almost all exciting results based on recurrent neural networks are achieved with LSTMs.

LSTM Networks

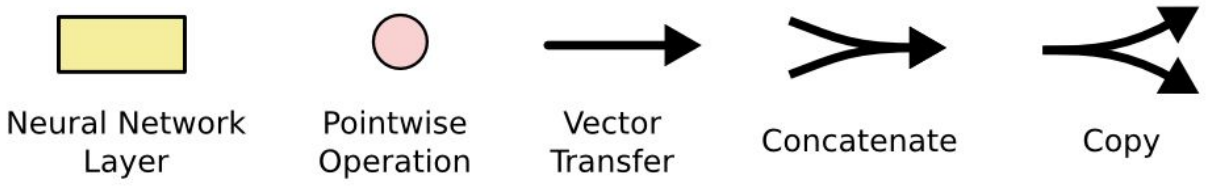
Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

LSTMs also have a chain like structure like RNNs, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

Below is the notation that will be used for explaining about LSTMs in-depth.

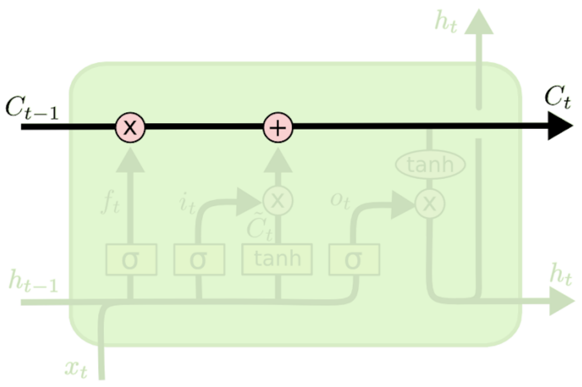


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.



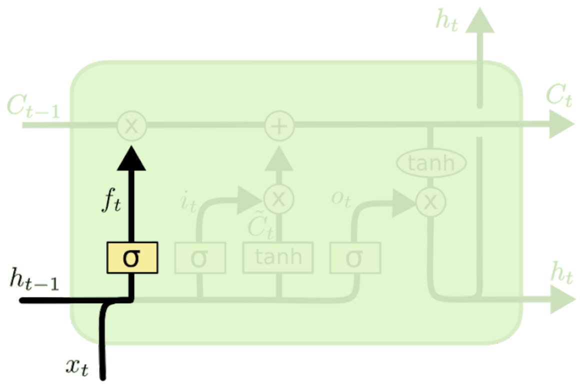
The LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates.

An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at ht−1 and xt, and outputs a number between 00 and 11 for each number in the cell state Ct−1. A 11 represents "completely keep this" while a 00 represents "completely get rid of this."

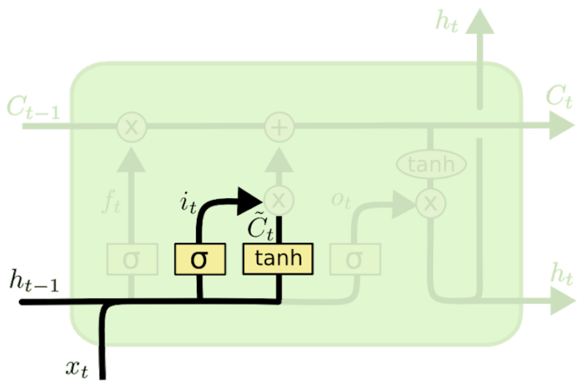
Let's go back to our example of a language model trying to predict the next letter based on all the previous ones. In such a problem, when we see a new subject, we want to forget the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, C̃t, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

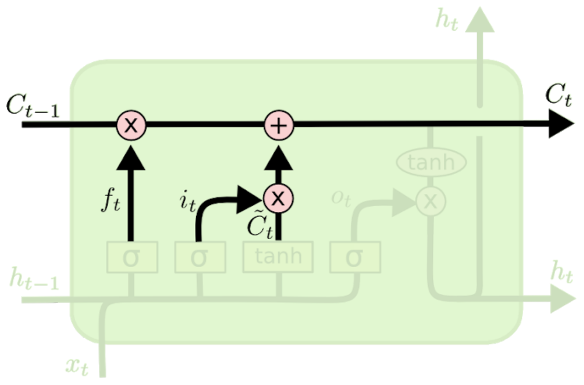


$$i_t = \sigma\left(W_i\cdot[h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C\cdot[h_{t-1}, x_t] \ + \ b_C)$$

It's now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.

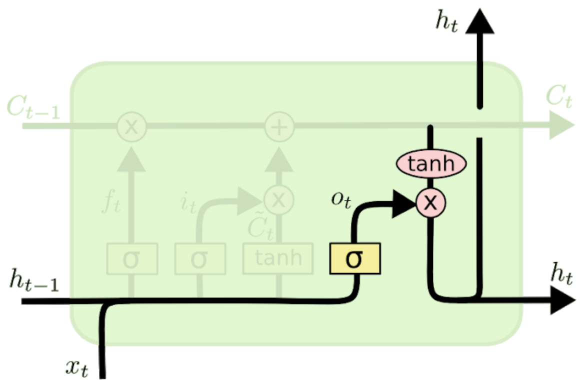
We multiply the old state by ftft, forgetting the things we decided to forget earlier. Then we add it∗C̃t. This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



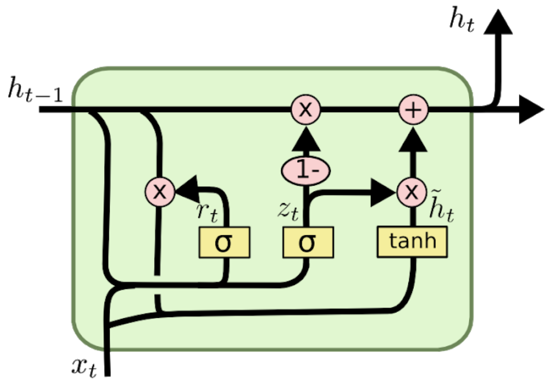
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

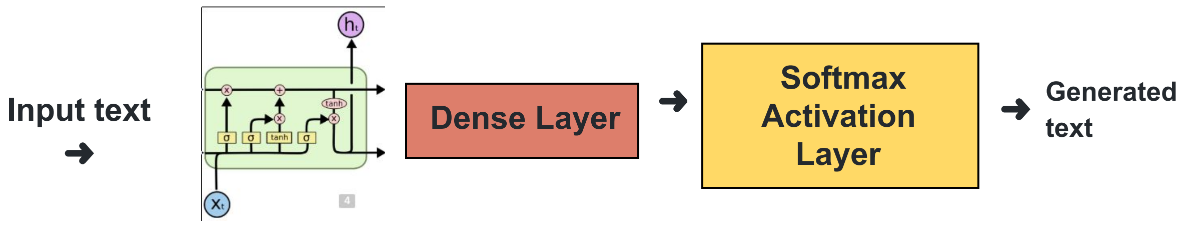
A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Simple LSTM Architecture

In our simple LSTM Architecture used for travel blog generation, a single LSTM layer is used, followed by 1 Dense and 1 softmax activation layer.



2.4 Benchmark

The vanilla stateless LSTM that was used during capstone proposal using the same dataset as input had a loss of **1.5983** and perplexity score of **15683.8785**.

III. Methodology

3.1 Data preprocessing

About 7 MB of data has been collected by crawling around 1000 blogs of top 2 bloggers. It amounts to about 6.5 million characters. The LSTM uses this data and comes up with 4 different suggestions of a paragraph containing 400 characters each. The ipython notebook for the same can be found here:

https://github.com/sapvan/udacity-mlnd-capstone-project/blob/master/blog_bot.ipynb

To begin with, the data from the text file is read and analysed. It contains a lot of unwanted characters. All non-ASCII characters are filtered out first. There are around 12 more unique ASCII characters which are not required. They are filtered out as well

The total number of unique characters turned out to be around 57. This initial set of characters did not contain uppercase alphabets. So, for the final model, we included uppercase alphabets and the count came to about 83 characters.

The characters cannot be modelled directly. It should first be converted to integers. This is achieved by mapping every character to an integer. A reverse mapping is also done, so that the integers can be converted back to characters.

The integers should be rescaled to a range of 0-to-1 to make the patterns easier to learn by the LSTM network that uses the sigmoid activation function by default. In order to do that, the input sequences are one-hot encoded by creating a 3-dimensional matrix representation of sentences as described below:

- The 1st dimension is the total of all sentences (nb sequences).
- The 2nd dimension is the length of each sentence, in our case 40
- The 3rd dimension is the length of total vocab/unique characters, in our case 83.

The output consists of a one-hot encoded 2-dimensional matrix.

- The 1st dimension remains the same as input.
- The 2nd dimension has the length of total vocab/unique characters.

The output pair tells the next char for every input pair.

3.2 Implementation

For the given input data, which belongs to the sequence prediction category, LSTM implementation from the keras library is used for predicting character-based sequences. As mentioned in this article and this paper, LSTMs are a good fit for character based sequence prediction.
The metrics initially considered were **ROUGE, METEOR** and **BLEU**. **Perplexity** turned out to be a better metric for this use case because it measures how well a probability distribution/model predicts a sample.
The initial implementation was a 1 layer LSTM. It was not giving satisfactory results, so tried out with a 2 layer LSTMs. The time consumed by the model was a big challenge because some variants would run for several hours on AWS!

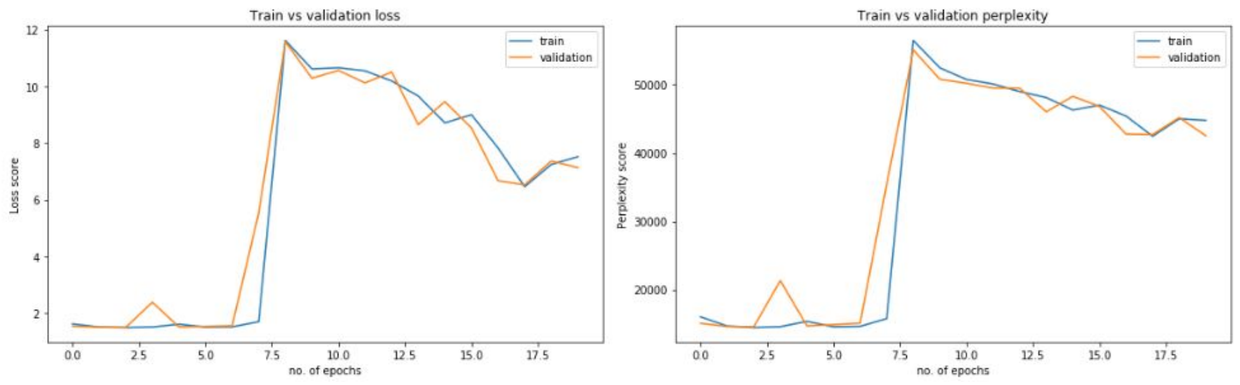
3.3 Refinement

I experimented with many variants of 1 layer and 2 layer LSTMs. For the **1st experiment**, I started with a single layer LSTM, with the below values:

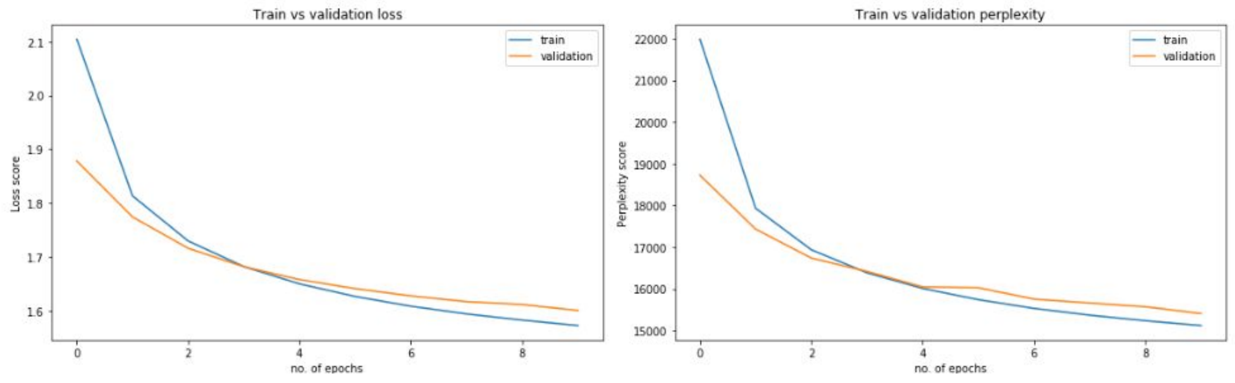
```
128 hidden units
optimizer=RMSprop(lr=0.01)
epochs=20
validation_split=0.2
shuffle=False
batch_size=128
```

Shuffle was set to false because I wanted to see if not shuffling the sentences gave better results. I started with 128 hidden units because the data was huge. Optimizer used was RMSprop because it tends to give better results in LSTMs.
In the **2nd experiment**, all parameters were same except *shuffle* was set to True (its default value). The result was not any better than the latter. Since this is a stateless model, the *shuffle* parameter might not play a big role like how it does in stateful models. However, in both the models, there was a marginal improvement in loss and perplexity score as compared to the benchmark model. **Best score: loss: 1.4984, perplexity: 14467.7558.**

In the below figure, the first graph gives a measure of **loss** among the training and validation datasets. The second graph gives a measure of **perplexity** among the training and validation datasets.



From the above graph, it appears that the model overfits right from first epoch and goes haywire after that. So for the **3rd experiment**, I reduced the number of hidden units to 64, increased the validation_split to 0.4 and changed the optimizer to **adam** because it is more robust than RMSProp. The results were much better than the first 2 models. The graphs started to make sense. The overfitting was happening very early. So I decided to run a **4th experiment** by reducing the units to 32 and changing the validation split to 0.33. There were no significant changes in the result:



From the above graph, it can be seen that overfitting is happening at around the 3rd epoch. However, the performance is almost same as the benchmark model even after 10 epochs. The sentences generated are far from making sense.

Best score:

loss: 1.5724, perplexity: 15122.1998

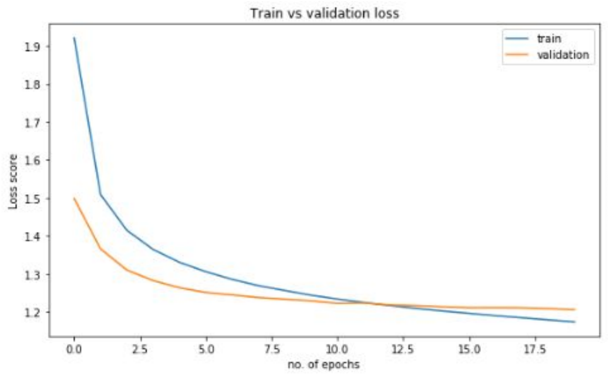
val_loss: 1.6004, val_perplexity: 15410.0290

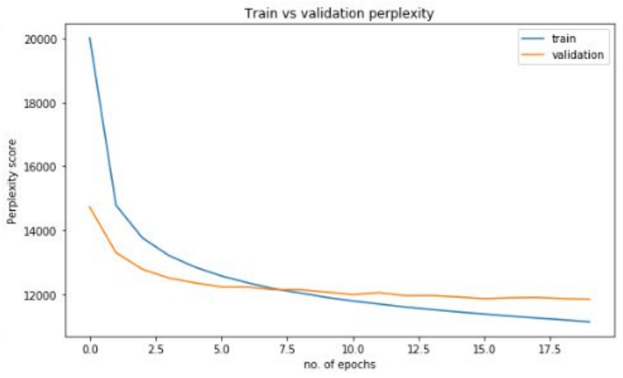
A stacked LSTM is better at identifying more information than a single layer. For the **5th experiment**, a 2-layer LSTM was designed with 512 hidden units, which I guessed would be a overfit, so I introduced a dropout layer.

```
model = Sequential()
model.add(LSTM(512, input_shape=(x.shape[1], x.shape[2]),
return_sequences=True))
model.add(Dropout(0.4))
model.add(LSTM(512))
model.add(Dropout(0.4))
model.add(Dense(y.shape[1]))
model.add(Activation('softmax'))

Optimizer='adam'
epoch=20
validation_split=0.2
```

This model started overfitting after around 11 epochs. The sentences started to make sense.





At epoch 9, for diversity 0.5, it generated the following:

```
"ur credit is good, it has never been so " ---
ur credit is good, it has never been so far and that i always say that i was a lot of money
and the same ways to be found a traveler that we try to do a country. i didnt know what i love
to try to connect every day of the post that you want to see the internet with a trip to the
internet and you can still have to take their starting the world in the streets that had feel
the only thing is that i was able to start the start of the last
```

The input text to the model was in lowercase. So the model could not learn things like the first letter of a sentence should be upper-case etc., The model is however picking up well in other aspects. It is not generating any new/unknown words. The sentences are slowly starting to make sense. This shows that we can get better results by fine tuning the above 2-layer LSTM.

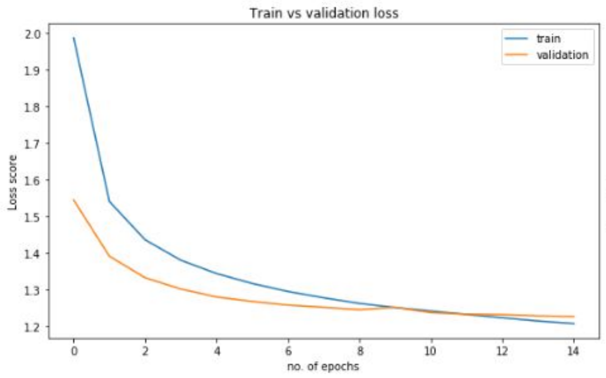
Best score (before overfitting):

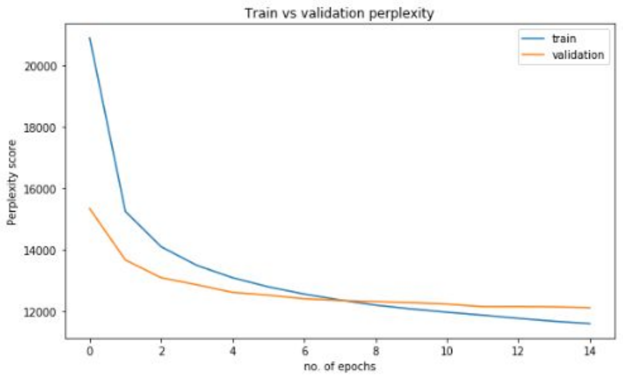
Training: loss: 1.2438 , perplexity: 11908.7241 -

Validation: val_loss: 1.2290, val_perplexity: 12073.6322

In order to tackle the overfitting problem of the above model, for the **6th experiment**, the input dimensions were increased. The vocab size (length of unique characters) was 57 for the previous experiments. For the current experiment, the vocab size was increased to 83, which meant the input to the model included uppercase alphabets. The model improved by a bit as compared to the previous model. The sentences generated made much better sense. **At epoch 7, for diversity 0.5**, the model generated the following:

```
----- diversity: 0.5
----- Generating with seed: "How, right? Just like the industry convi"
How, right? Just like the industry convinces that I was still only feeling a few hours and
travel in the day that any of the most important partnerships and staying in the end of the
photos of the street of the world, which is all of the day that I can be as what I would
always stay in the above and say that I do start about their life of the city, the real
exciting of food is a check of the streets of the moment with the destinations
```





From the above graph, it can be observed that the overfitting is happening after a few more epochs as compared to the previous experiment. This shows that the current 2-layer model will produce even better results if more data is given as input or if the number of hidden units is reduced, which will inturn reduce overfitting.

Best score (before overfitting):

Training: loss: 1.2419, perplexity: 11969.8883

Validation: val_loss: 1.2375, val_perplexity: 12233.9370

IV. Results

4.1 Model Evaluation and Validation

As observed in the previous section, the 2-layer LSTM model used for the 5th and 6th experiment is a good fit for the current use case. The content generated by the model does not make any spelling mistakes. It has learnt rules like when to use a comma, period and other punctuation marks appropriately. This shows that, with further training, the model will perform even better.

```
How, right? Just like the industry convinces that I was still only feeling a few hours and
travel in the day that any of the most important partnerships and staying in the end of the
photos of the street of the world, which is all of the day that I can be as what I would
always stay in the above and say that I do start about their life of the city, the real
exciting of food is a check of the streets of the moment with the destinations
```

We input data with different dimensions to the model in our 5th and 6th experiments. The model gave out consistent results. This shows that the model is robust enough for the problem and small changes to the input space does not greatly affect the results.

4.2 Justification

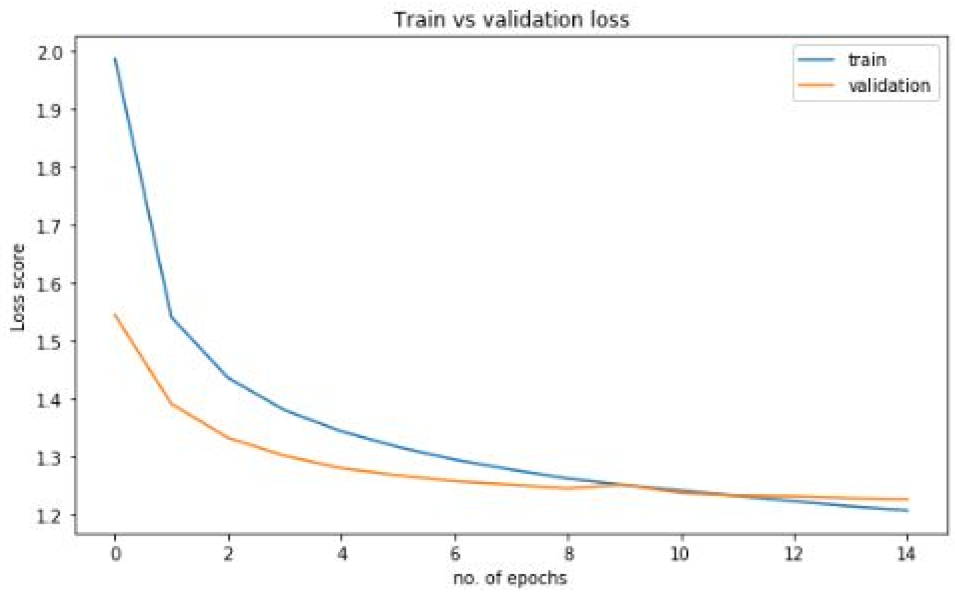
The results of the final model are better than the benchmark. We got the below scores:

Benchmark model:
    Training: loss - 1.5983, perplexity - 15683.8785
Final model:
    Training: loss: 1.2419, perplexity: 11969.8883

As observed from the experiments, the 2-layer LSTM performs better than the 1 layer LSTM. With the change in input, the results were consistent. The sentences generated were much better than the previous models.

V. Conclusion

5.1 Free-Form Visualization



After a total of 6 experiments and 70 hours of training on GPUs on AWS, came up with a model that has a training loss of about 1.2. The above train vs validation curve generated by the final model shows that this model has the potential to improve if more input data is fed.

5.2 Reflection

We have trained a simple LSTM with 2 layers using travel blog data of 2 top travellers. We have shown that this model produces decent content. We may have to improve the content by providing more input data.
Even though the results are promising, the next step is to further improve the model by feeding it more data. This is a very very time taking process. Every experiment took 10-15 hrs on g2 xlarge and g2 8xlarge GPUs on AWS. As a result of this, I could not experiment with everything that I had planned to improve the model. This process could be improved by using even better set of multiple GPUs.

5.3 Improvement

- Increase the dropout
- Reduce the number of hidden units or increase the amount of data input to the model.
- Experiment with scale factors (temperature) when interpreting the prediction probabilities.
- Add Attention
- Use k-Fold Cross Validation
- Use a GRU instead of LSTM.

References

→ Character-level Recurrent Neural Networks in Practice: Comparing Training and Sampling Schemes
→ LSTM Text generation
→ Sequence prediction using LSTM
→ Travel Blogs with Deep Learning
→ Crawling Blogs
→ Character level deep learning
→ The Unreasonable Effectiveness of Recurrent Neural Networks
→ How to Develop a Character-Based Neural Language Model in Keras
→ Develop a Word-Level Neural Language Model and Use it to Generate Text
→ Character-Aware Neural Language Models. A Keras-based implementation
→ Metrics NLG Evaluation
→ Using different batch sizes
→ Stateful LSTM
→ Sequential model guide
→ Perplexity score
→ Understanding LSTMs
→ Stacked-long-short-term-memory-networks
→ LSTM for generating simpsons episode