Machine Learning Engineer Capstone Project

Song Lyrics Generator

If I could describe my project in a line, it would be the following – A program that generates lyrics.

Problem Overview

Before we get into all the technicalities of the project, let us establish the domain and familiarity with the background of the problem. Primarily, the problem deals with lyrics, which is Natural Language. Since we are generating new lyrics, the done work falls under Generative Natural Language processing.

Natural Language processing has been evolved exponentially over the past few years, especially in the last year with the invention of Transformer models that accelerated modern NLP algorithms performance massively, even matching human-level accuracies in tasks like Question Answering, where the algorithm is supposed to find an answer to a question from a paragraph. (see SQuAD 2.0)

Technically, what we try to do in this project is to make a language model. Roughly, a language model learns the language (text) it is trained on. If you train it one news articles, it learns the vocabulary and the language structure of the text in those news articles. Recently, OpenAI, an AI company lead by Elon Musk, released a language model called GPT-2 that could astonishingly generate real appearing text. For example, the text below is generated by an algorithm.

```
The scientist named the population, after their distinctive horn,
Ovid's Unicorn. These four-horned, silver-white unicorns were
previously unknown to science.
Now, after almost two centuries, the mystery of what sparked this odd
phenomenon is finally solved.
Dr. Jorge Pérez, an evolutionary biologist from the University of La
Paz, and several companions were exploring the Andes Mountains when
they found a small valley, with no other animals or humans. Pérez
noticed that the valley had what appeared to be a natural fountain,
surrounded by two peaks of rock and silver snow.
```

Dataset

The data is obtained from Kaggle (link: https://www.kaggle.com/mousehead/songlyrics)
It contains information of more than 55000 songs like title, lyrics, artist name, etc.
For our work, we use song lyrics. An example of lyrics is given below

Look at her face,
it's a wonderful face

And it means something special to me
Look at the way that she smiles when she sees me
How lucky can one fellow be?
She's just my kind of girl, she ma...

We use the lyrics data and train our algorithm to learn the language of music.

Problem Solving Strategy

Data Preparation Steps

Below is the brief outline of the steps we follow for the work. A detailed analysis specific to our case is provided ahead in the next sections.

1) Cleaning the data: We select top 200 song lyrics. After we get them from a pandas DataFrame column to a list, we perform normalizing operations like lowercasing, punctuation removal, etc.

2) Understanding the vocabulary: To understand what kind of words are present in our data, we compute the frequency of the tokens (or words) to check the variety and how dominant are the most frequent words. If a few words comprise 40-50% of our total words, we might face a problem while training the algorithm. It is always good to know about such cases.

3) Deciding the training strategy: This includes deciding whether to use the words as individual inputs to our network or characters (subword models). If our model has the majority of unique words with few counts (1-3), using a word-level model can be difficult. Similarly, using character-level model requires more training time and resources than word model.
   Besides, we also need to think about how we want to train our LSTM? In our case, we input 50 characters, and the output of the final time step is used to predict the 51st character. Accordingly, the loss is computed, and back-propagation occurs.

4) Building model friendly inputs: After we have decided our strategy, let's say character-level model, we build a mapping from unique characters to a unique index, since models take numbers as inputs. Simultaneously we also build an exact inverse mapping from unique index to characters, since model outputs numbers which we need to convert to characters.

5) Training the network: We now build the LSTM used for training, we have experimented with several parameters like a number of hidden neurons, a number of LSTM layers, dropout to prevent overfitting, etc. Results are reported accordingly. We also keep track of loss (Cross Entropy) and accuracy to monitor the model performance.

   The training is done using SageMaker APIs and custom build model and train scripts.
6) Testing the network: The model is tested by giving a trigger string as inputs (any random 50 characters, for example – '*I want to say, I liked this from my neighborhood*'. Moreover, then the model will predict the next N characters.

Data Exploration

A summary of the Data exploration is tabulated below.

Content	Number
Number of lyrics	200
Total number of lines	9416
The average number of lines in a song	47

Table 1: Line level statistics

Content	Number
The average number of words in a lyric	242
Total number of words	48388
Total number of unique words	5238
The average frequency of a unique word	9.2
Top 5 most frequent words and their counts	`('the', 1764), ('I', 1734), ('you', 1189), ('a', 1177), ('to', 880)`

Table 2: Word-level statistics

Fraction of unique words vs counts

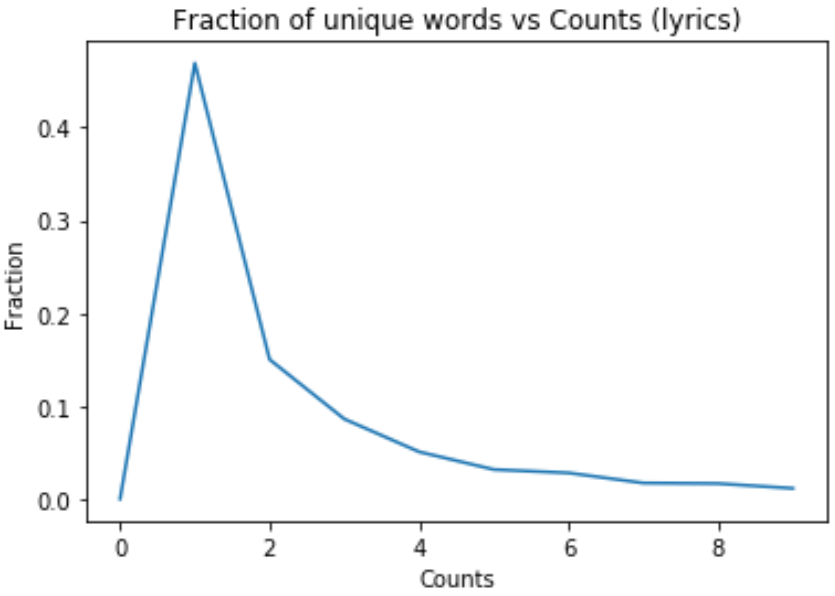

Fig 1: Fraction of unique words vs counts in song lyrics

As we can see, around 50% of the unique words have counts equal to 1. This is not desirable as the model will have difficulty learning so many rare words. Hence, it is favorable to use a **character-level** language model.

Distribution for the number of characters in lyrics

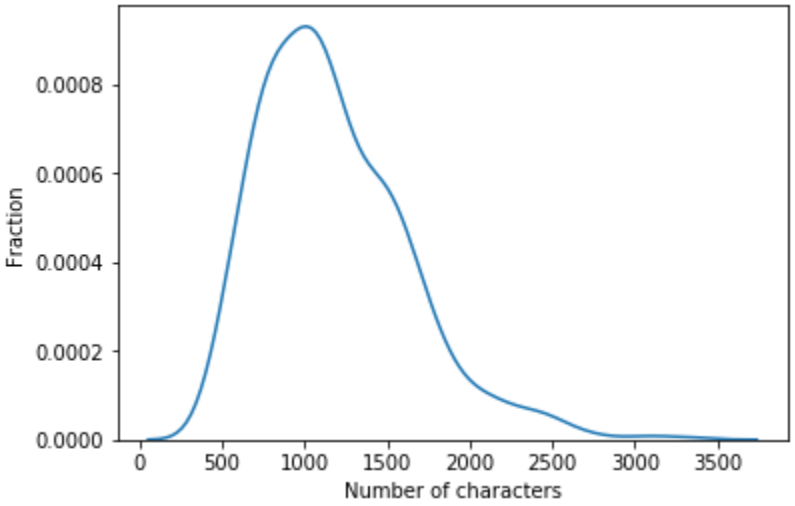

Fig 2: Distribution of the number of characters in lyrics

Content	Number
Median number of characters in lyrics	1222
Number of characters at 90th percentile	1891

Table 2: Character level statistics

Preprocessing

Normalization
The preprocessing operations we perform as very standard to any NLP task. We perform the following steps.

- Lowercasing all the alphabets
- Removing punctuations
- Replacing '\n' with '!' (which is then indexed)
- Creating unique characters to index mapping and vice versa

Example:

Original text:

```
I'll never know why I had to go
Why I had to put up such a lousy rotten show
Boy, I was tough, packing all my stuff
Saying I don't need you anymore, I've had enough
And now, look at me standing here again 'cause I found out that
```

Ma ma ma ma ma ma ma ma ma ma ma ma ma ma ma my life is here
Gotta have you near

As good as new, my love for you
And keeping it that way is my intention
As good as new and growing too
Yes, I think it's taking on a new dimension
It's as good as new, my love for you
Just like it used to be and even better
As good as new, thank God it's true
Darling, we were always meant to stay together

Preprocessed text:

ill never know why i had to go ! why i had to put up such a lousy rotten show ! boy i was tough packing all my stuff ! saying i dont need you anymore ive had enough ! and now look at me standing here again cause i found out that ! ma ma ma ma ma ma ma ma ma ma ma ma ma ma ma my life is here ! gotta have you near ! ! as good as new my love for you ! and keeping it that way is my intention ! as good as new and growing too ! yes i think its taking on a new dimension ! its as good as new my love for you ! just like it used to be and even better ! as good as new thank god its true ! darling we were always meant to stay together ! ! feel like a creep never felt so cheap ! never had a notion that my love could be so deep ! how could i make such a dumb mistake ! now i know im not entitled to another break ! but please baby i beg you to forgive cause i found out that ! ma ma ma ma ma ma ma ma ma ma ma ma ma ma ma my life is here ! gotta get you near ! ! i thought that our love was at an end but here i am again ! ! as good as new my love for you ! and keeping it that way is my intention ! as good as new and growing too ! yes i think its taking on a new dimension ! its as good as new my love for you ! just like it used to be and even better ! as good as new thank god its true ! darling we were always meant to stay together ! ! yes the love i have for you feels as good as new ! darling we were always meant to stay together! !

Creating input and target data

We take a sequence of 50 characters as input and use the model output to predict the 51st character. For instance,

Input	**Label**
[i,l,l, ,n,e,v,e,r, ,k,n,o,w, ,w,h,y, ,i, ,h,a,d, ,t,o, ,g,o, , ,!, **w**]	[**h**]
[l,l, ,n,e,v,e,r, ,k,n,o,w, ,w,h,y, ,i, ,h,a,d, ,t,o, ,g,o, , ,!, **w,h**]	[**y**]

Table 3: Example of Input Sequence and labels

In terms of indexes

Input	Label
[1, 22, 34, 2, 8,6, 1, 40, 22, 5, 7]	[39]
[22, 34, 2, 8,6, 1, 40, 22, 5, 7, **39**]	[7]

Table 4: Example of indexed input and sequence

Preparing the data according to the above mentioned scheme, we prepare our dataset.

Final Dataset size: 260279
90% of the dataset was used as the train set and the rest 10% as the test set.

Algorithms used – LSTM

LSTM has been used quite a few times throughout the Nano Degree. For the sake of it, we'll just brush up on what it is capable of and why is it preferred over traditional RNNs

LSTM or Long Short Term Memory is a variation or RNNs (Recurrent Neural Networks). RNNs are neural networks used to model sequential data (Natural Language, Time Series data).

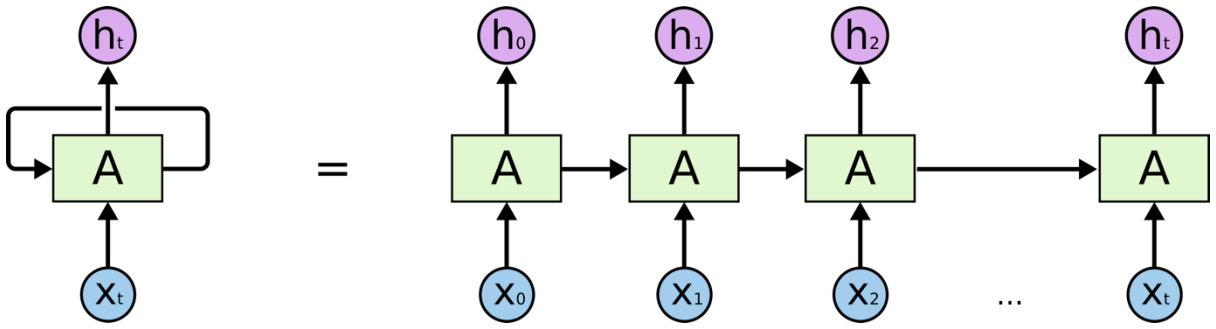

Image credits:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

If a sequence of inputs $[X_0, X_1, X_{2, ...}]$ is given as an input to the RNN, stepwise, the hidden state output for a time step is used as a secondary input to the next time step. For example, hidden state of X_0, called h_0 is an input to the next time step along with X_1 .

When the output of the algorithms is used to predict the labels $[y_0, y_1, y_{2, ...}]$, the loss computed is used to perform optimization using gradient descent. The gradient for a time step **N**, flows through the previous time steps **N-1, N-2, ... 3, 2, 1**.

It so happens that as the gradients flow deeper through the previous time steps, either the gradients start minimizing as they flow deeper. Thus, the update step using gradient descent isn't that effective. This is called the problem of vanishing gradients. In another case, these gradients accumulate and take large abrupt values. This is called the problem of exploding gradients.

LSTMs help avoid this problem by accompanying memory in their structure.

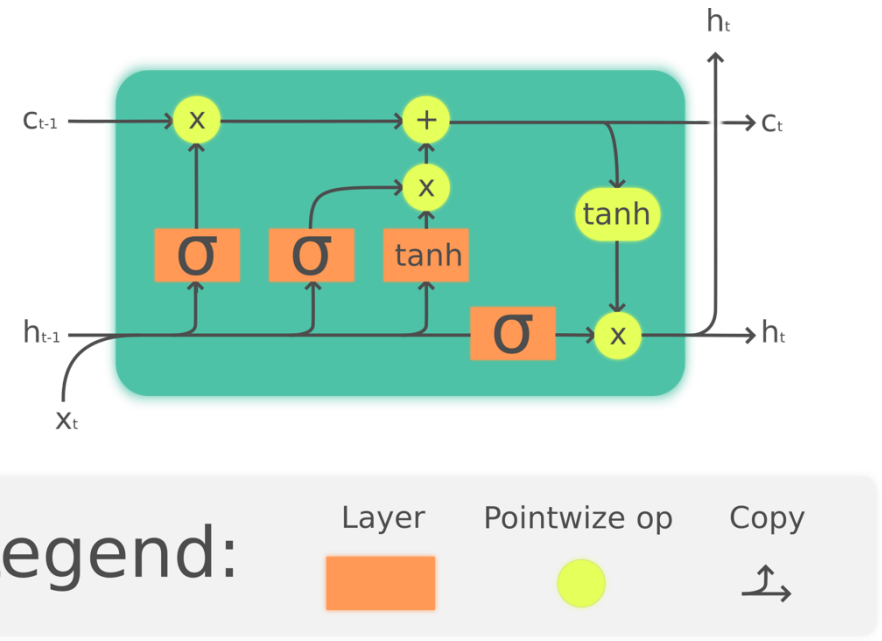

Image credits: https://upload.wikimedia.org/wikipedia/commons/3/3b/The_LSTM_cell.png

LSTMs, along with hidden states, have gates that allow a fraction of information to be transferred to the next time step and a fraction of it to be forgotten. This helps LSTMs have longer memory and also avoid abrupt gradient values.

Evaluation Metrics

Since we are modeling the language using an input sequence to predict the output single next character, the problem can be thought of as a classification problem with a number of classes equal to the vocabulary size. Hence we use **accuracy** as our model performance metric.

$$accuracy = \#correct\ predictions/\#total\ number\ of\ predictions$$

Why might it be a good candidate? Because a high accuracy score means that the model is learning the correct spelling structure and the syntax of the language. For example, is an input sequence is something like - ***'Hey girl! How are you this eveni'***. A highly accurate model knows that the word ***eveni*** is actually evening and will predict ***n*** as the next character. This becomes important in subword models where spelling structures are to be captured along with language syntax

Results

Benchmark: We keep the benchmark to be a model with no features. If a model were to predict the outcome for every input sequence, without learning any parameter, it would have maximum accuracy if it predicted the most frequent character every time. In our case, for the test set, it happens to be the <SPACE> (' ') character, which comprises of 26% of our test data

Therefore, benchmark accuracy = 0.26

NOTE: While examining the result, keep 2 things in mind

1) The results were computed on 1 epoch only. The aim is to examine how key hyperparameters play a role in the model performance
2) There are total of 38 possible class labels (unique characters). The more the number of classes, the harder it is to achieve higher accuracies

LSTM performance variation with hidden states

Parameters	**Values**
Number of Layers	2
Embedding Dimensions	256
Batch Size	200
Epochs	1

Table 5: Model Parameters

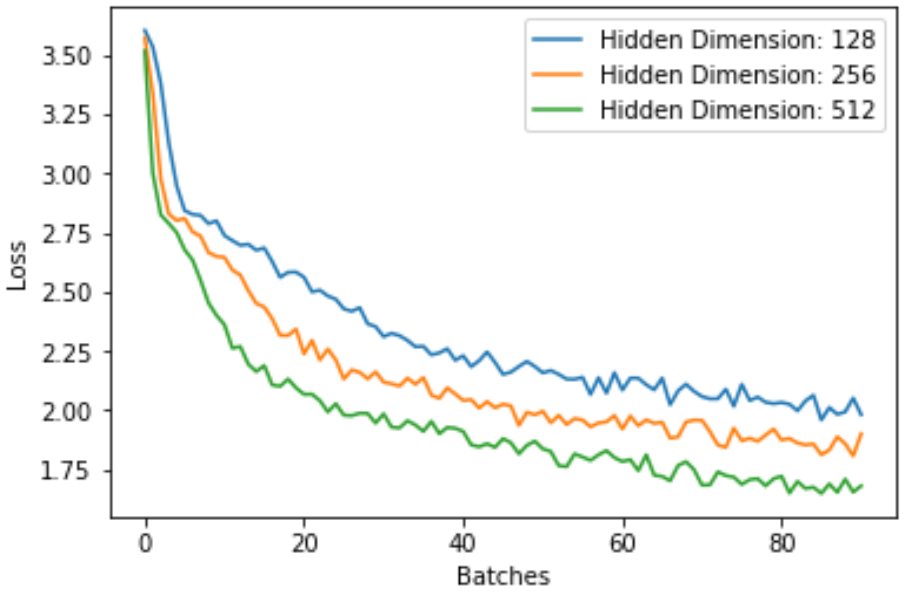

Fig 3: Loss vs batches

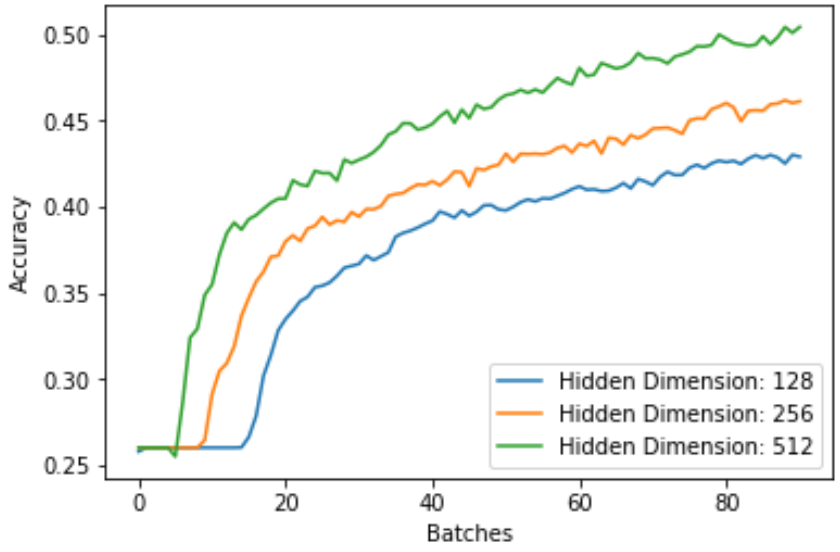

Fig 4: Test accuracy vs batches

We calculate percent improvement over baseline as

$$\%\ improvement = \frac{baseline\ error - model\ error}{baseline\ error}$$

Hidden Dimension	**Percentage Improvement in Error Over Baseline (After 1 Epoch)**
128	20.5%
256	23%
512	32%

Table 6: Model Improvement over Baseline

I was also interested in how the embedding dimension played a role in model training. The following results indicate that they do play a little role.

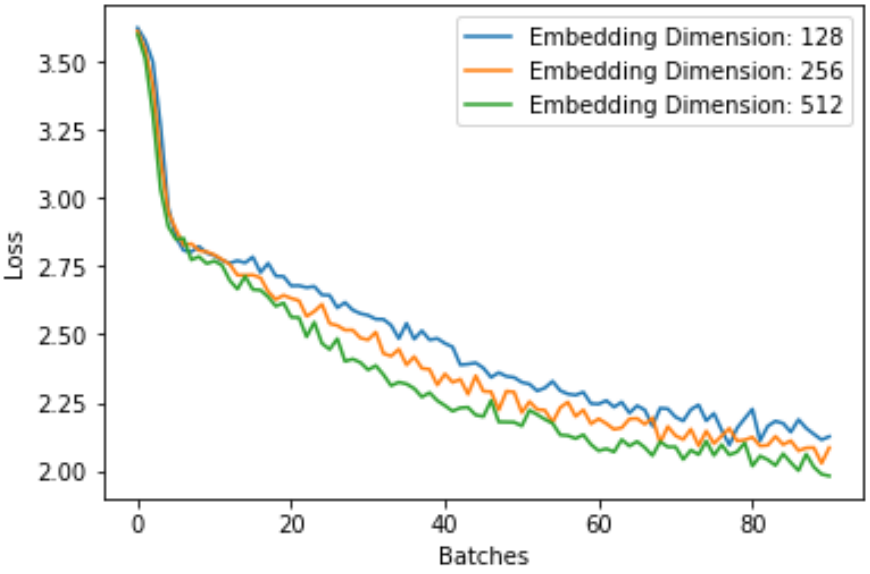

Fig 5: Embedding dimension variation with loss

After tuning the hyperparameters, the best score the model attained was 59% This is 43% improvement on error over the baseline. The model parameters were as follows.

Hyper Parameter	**Value**
Hidden Dimension	512
Embedding Dimension	256
Number of Layers	2
Epochs	10

Table 7: Final Model Hyperparameters

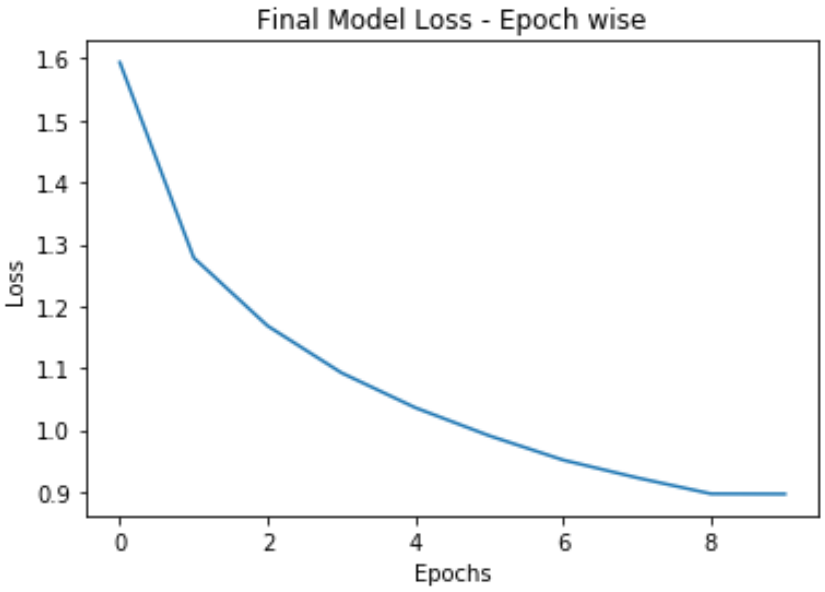

Fig 6: Loss for the final model

Language Generation Example

Now we experiment how the results turned out for some trigger input string. For example, for a string like

Example 1:

Input string
hey girl how are you! its me! that dude ! please listen

The output
i was so happy new year
and i wonder it sweet

i was so lonely
if it wasnt for the nights i think that i could be
sometimes i close my love

im always have always will

you know that you can learn to blue
i was so lonely
i wonder its frightening
when youre so bad youre so bad youre so bad you and me
so i wanna stay
why did you do i do i do i do

Example 2:

Input string
'it was summer ! i was having the best time of my life'

Output
when you can hear them will didnt he
you can do magic
i wont let you go
what is right for me
what about livingstone

we wish i would lose her
were gonna live a little love affairs

nina pretty ballerina now thats all the fun
suppper trooppper
smiling having for these visitors
my friends do

im a bird dog
they drum the way that you try and the lights

Problems occurred in the process

I was familiar with the concepts of LSTMs and RNNs. So those weren't a problem. But I faced two major issues while trying out different models

1) Memory Issues: As I tried out mulitiple models with increasing complexity and hence the number of parameters, I faced memory issues while training on GPU instances. But since my main focus for this Nano Degree was to learn how to deploy models on Cloud, I did not focus too much on getting accuracy.

2) Model endpoint predictions: When an endpoint is created, to test it on a test set on the local notebook, we call the predict method on the endpoint. I observed and verified on Github issues that the predict method fails if we try to predict a test set of large size like 10000. I had to break down the test set and then compute the scores separately.

Conclusion

Overall, the model's performance over a baseline is pretty well. Of course, more improvements can be made with increasing complexity and computational resources. But for the scope of the Nano Degree, our model performs conveniently well. It is also evident from the sample outputs the model generates. Although few sentences do not make sense, the output, at the character level seems good.

Vote of Thanks

I'd like to thanks Udacity, the AWS and the Kaggle team for making such a wonderful course and the support platform around it. The skills I learned here are directly applicable to my job. I am surprised as to how much practical knowledge course imparts instead of just dumping chunks of less useful information.