Unique Bedtime Stories

Using Recurrent Neural Networks to generate unique stories in the style of the late great Hans Christian Andersen

By Michael Weber

Motivation

Bedtime stories are important. A stimulating bedtime story can have a lasting effect on a child. Many studies have shown a significant link between children who either read or are read to at bedtime and higher literacy rates and brain development, as well as child parent relationships.

There is also a proven market for children's books and stories. For example "Go the F--k to Sleep" by Adam Mansbach was the New York Times #1 selling book not just in the children's category but for all books. Lastly, for the parents out there tired of reading the same story to their children over and over again night after night...

Wouldn't it be nice if there was an app to instantly generate a new unique story on demand?

Data and Tools

For data I used the **digital archive Project Gutenberg**, specifically three popular fairy tales from the late Danish author Hans Christian Andersen. The Little Mermaid, The Snow Queen and The Emperor's New Clothes, each has inspired and spawned numerous other story spin offs and feature films. I ended up with about 120k individual characters, 21k individual words and about 1k unique words.

I used some new powerful and extremely useful tools for this project starting with **Google Colab GPU, a FREE cloud computing platform** that I used to run my models remote to avoid over stressing my local machine. The GPU was set up to run **Keras and TensorFlow** which I used to run my **Long Short Term Memory (LSTM) model.** The GPU was a time saver to the max. A model that took my local machine 45 mins to run only took 7 mins on the GPU, a massive asset for this time sensitive project. Another tool I utilized for this project was **Flask**. This was used to deploy my Unique Bedtime Story Generator program to a web app that can be used by parents with access to the web.

Approach

After I had settled on text generation as my project I read a good deal of available information on the topic and found that there are two typical methods of generating text from a corpus trained model, word-based and character-based. There are pros and cons to each, but for this project I used **character-based text generation** because it required **significantly less computational cost** and seemed to be **better with smaller data sets**, like my 3 stories.

Preprocessing

Another added plus of using character-based text generation is it **requires significantly less preprocessing** of the text before modeling. I proceeded to **remove many of the redundant characters using regular expressions.** Things like, extra line breaks, "\n", and some unique characters like underscores _, brackets [], and others. This removed about 6k characters and as easy as that, I was ready for modeling.

Long Short Term Memory Model was the model of choice because to generate text, character by character from a sequence of text, I needed a model with "memory". The LSTM model can retain the effects of previous inputs and predictions to more accurately improve output.

LSTM models take input in the form of vectors of numbers, not text of characters and words, so I needed to transform my text corpus data to numbers in the form of sequence vectors. I created a set of all the unique characters in my corpus and then mapped a character to number dictionary and also a second number to character dictionary to be used to translate and then output the finished "text" output. I then used the character to number dictionary and chunked the total corpus into sequences of 100 characters and put them in vector form. Lastly, I reshaped vectors to fit the LSTM model input parameters and then it was time to run the model.

Modeling with LSTM

As a **baseline** and my first time working with a neural network, I chose to run just one epoch and one layer with default parameters to check that things were able to run smooth. With a successfull epoch I tried the model for a prediction and received nothing but blank spaces. Dissapointing but nonetheless I had a baseline and a working pipeline.

Refining the parameters became the focus. Over the next few days, I added a **second layer** to the sequence, **increased the units to 400** and then **700**, set the **dropout rate** to 0.2, chose the **activation of softmax** (best for multi class classification), used a **categorical crossentropy for the loss function** and used **adam for the optimizer.** This process was difficult and worked partly through extensive reading of articles on Medium and other sites and also partly through trial and error of running the model on the GPU.

One huge discovery was that I could **save the weights of each epoch of the model** and use those to predict and generate output text instead of waiting for the full model to run each time and then using the final product. This showed improvements with the above parameters and once the loss function got below about 2 I was able to see "words" (usually mispelled or non english) start to form with periods in seemingly correct places.

Results in Flask App

To actually generate text on the back end I loaded the model, weights and generated a random number to use as an index to take a section of the actual corpus to use as a seed for the model to begin predicting characters in a given range after an end point.

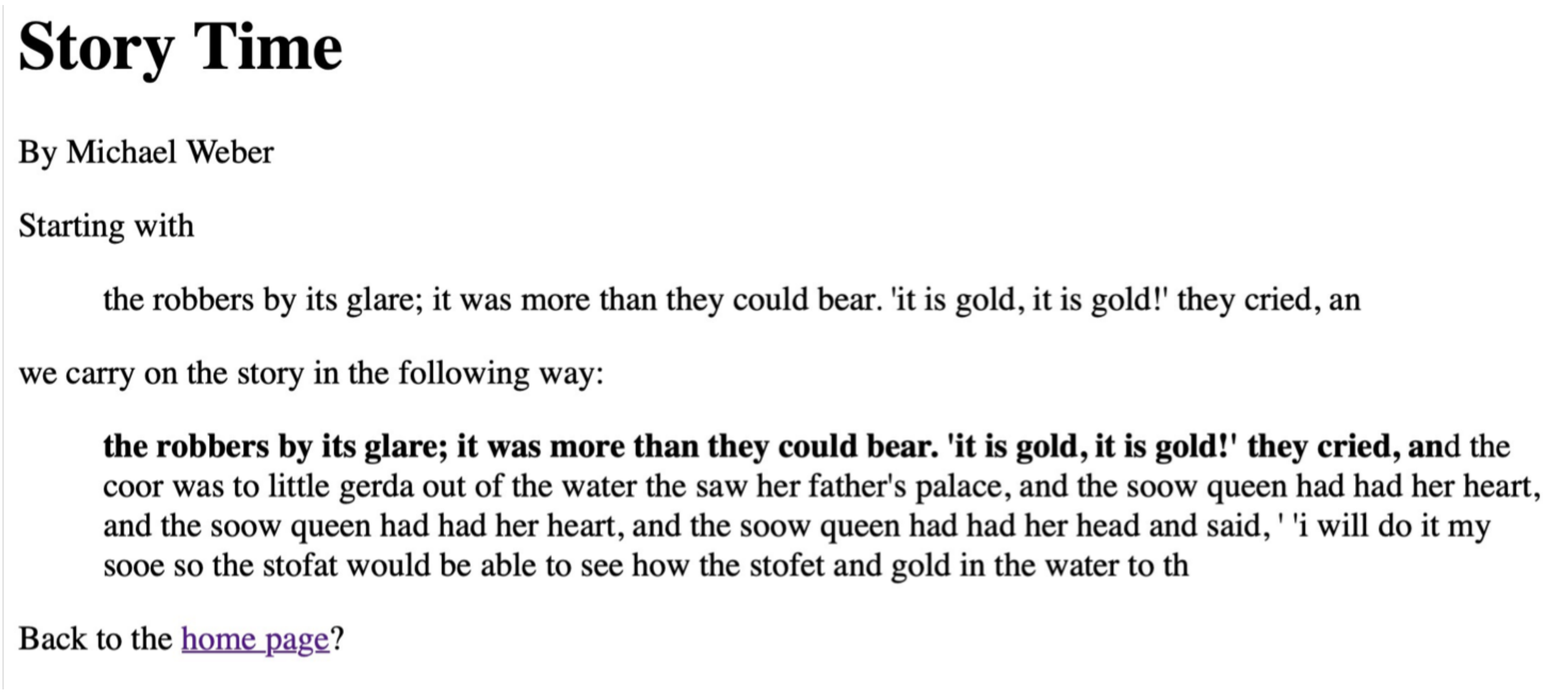
I was running out of time so I chose a weight function of 1.319 that had proven to produce some dicipherable output and chose to **build out a Flask App** to have a functioning end product for my project. I was able to pickle the necessary objects and import my model and built a home page with a button that returned a continuous string of text as the finished "Unique Bedtime Story".

Early Example:

```
Seed = ", and make my old excuse,'proving his beauty by succession thine!
```

Output = this were to be new made whe the the the thee and the the the the the the the the the the thee and the the the the th"

Final Output with Flask App web page



Story Time

By Michael Weber

Starting with

> the robbers by its glare; it was more than they could bear. 'it is gold, it is gold!' they cried, an

we carry on the story in the following way:

> **the robbers by its glare; it was more than they could bear. 'it is gold, it is gold!' they cried, an**d the coor was to little gerda out of the water the saw her father's palace, and the soow queen had had her heart, and the soow queen had had her heart, and the soow queen had had her head and said, ' 'i will do it my sooe so the stofat would be able to see how the stofet and gold in the water to th

Back to the home page?

Future Work

1) The model is far from perfect but it shows promise. I'd like to continue to tune the parameters, specifically add another layer and try the 'stateful' method to keep more information in the model's memory as it runs.

2) The ouptut context was fairly narrow and I saw the snow queen and mermaid show up quite often, so I would like to add more text to the corpus to diversify the output capabilities.

3) Finally, to improve the Flask App. Visually to pretty it up for the user and also to allow for custom input to use as the seed to allow users to direct where they would like a given story to go.

Thank you for reading

Michael Weber