Assignment 4 Bonus Points - DD2424

August Regnell 970712-9491

6 May, 2021

Contents

1 Adjustments to the original implementation

In the following section I will present the adjustments made to make the model applicable to tweets instead of a single book.

1.1 The *end-of-tweet* character

Since the training data now consists of several sub-strings (tweets) one must now mark the end of every sub-string with some character. I used "§" as the *end-of-tweet* character since it didn't appear in the tweets.

I pre-processed the data in `gotTwitter.csv` using `python3` and the `pandas` library. I saved 100,000 tweets with § as the delimeter in a `.txt` file, which was more suitable for the original `MatLab` code for Assignment 4.

1.2 Synthesizing of tweets

Since tweets have an upper cap of 140 characters the synthesizing function needed a smaller revamp. I changed it in such a way that it stopped the synthesizing process when it reached an end-of-tweet character or when 140 characters had been generated.

1.3 Re-initializing `hprev`

Since the batches in most cases doesn't align with a start of a new tweet it is not enough to just re-initialize `hprev` at the start of every epoch. If no change was made we would wrongly keep the context between two tweets that most likely have little to do with each other (except that they both are about Game of Thrones in some way or another).

Instead I re-initialized `hprev` in `ComputeLoss` every time the generated `x_next` was the end-of-tweet character.

2 Comments on the dataset

In addition to the differences between this dataset and the one used in the original assignment I pointed out above some more are worth addressing.

The Harry Potter dataset had 83 unique characters while this dataset has 407. This is partly due to the use of emojis in tweets. These additional characters are quite rare in comparison, mostly just increasing the amount of memory and calculations used while appearing in the synthesized tweets very rarely. All in all, these characters slow down the training significantly.

Another thing that may inhibit good performance is the abundance of URLs in tweets. While some parts may follow a recognizable pattern (such as `https://`) there is usually a string of random characters at the end of URLs. This will most likely confuse the network.

3 Finding a good sequence length

Before training the network for a longer time I want to make sure the the sequence length used is appropriate. Using a deterministic I compared the average smooth loss per character for different sequence lengths.

Since every "batch" corresponds to one sequence length's worth of characters, one cannot train the networks for the same amount of update steps, since this would mean more training for networks trained with a larger sequence length. Instead, I trained the different networks for the same amount of characters - 100,000.

I tried with five different values for the sequence length: 5, 25, 50, 100 and 140. The results can be seen in figure 1, 2, 3, 4 and 5 respectively.

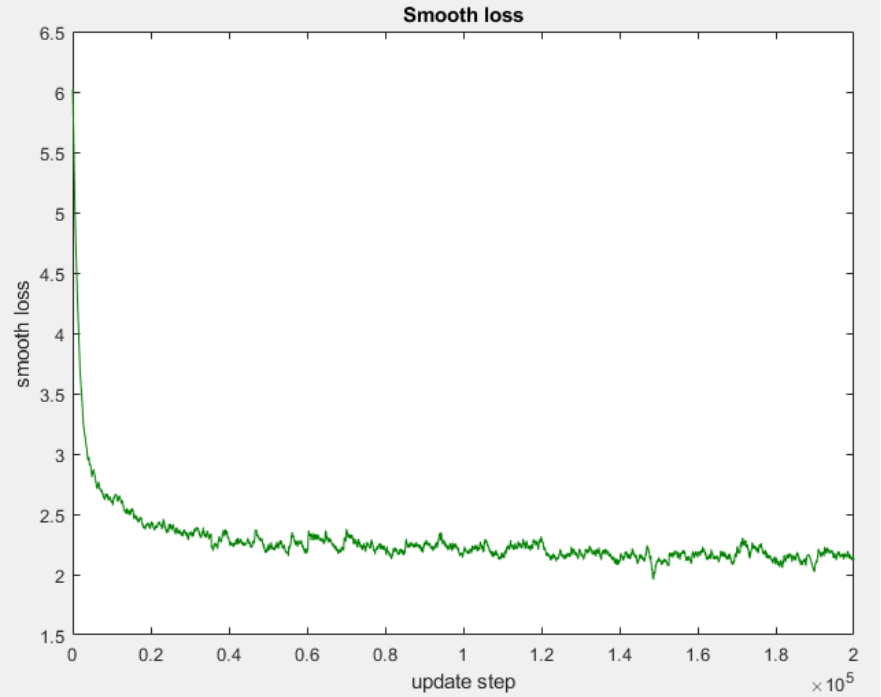

Figure 1: Plot of smooth loss for a of 100,000 characters with sequence length 5. The final smooth loss per character was 2.1799.

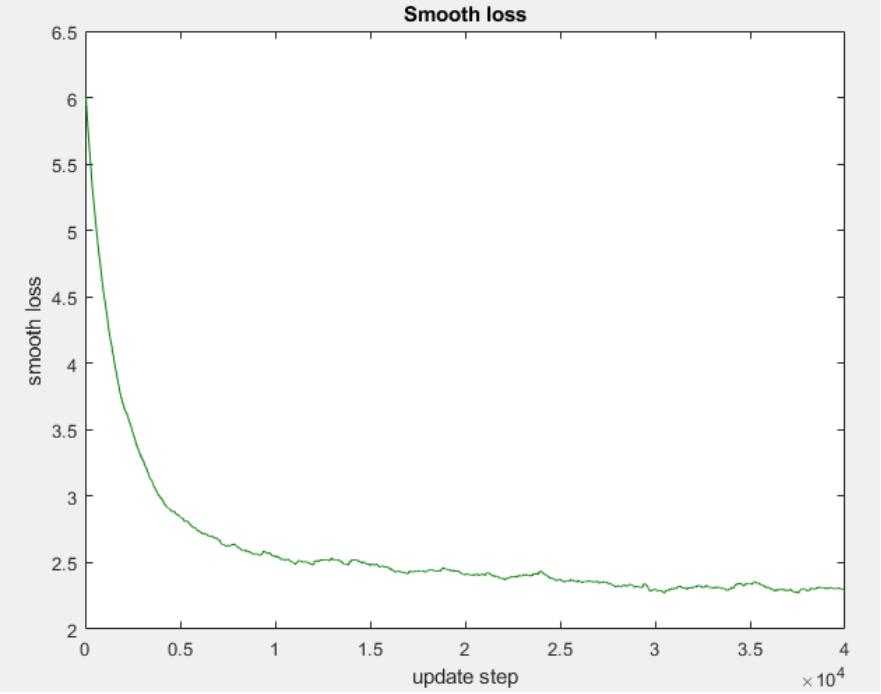

Figure 2: Plot of smooth loss for a of 100,000 characters with sequence length 25. The final smooth loss per character was 2.2960.

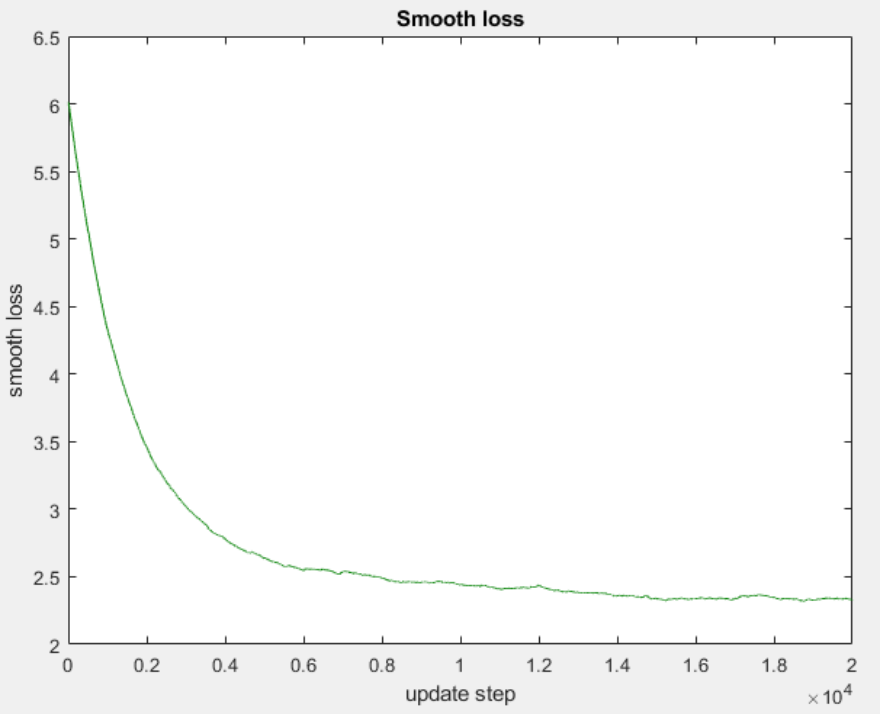

Figure 3: Plot of smooth loss for a of 100,000 characters with sequence length 50. The final smooth loss per character was 2.3316.

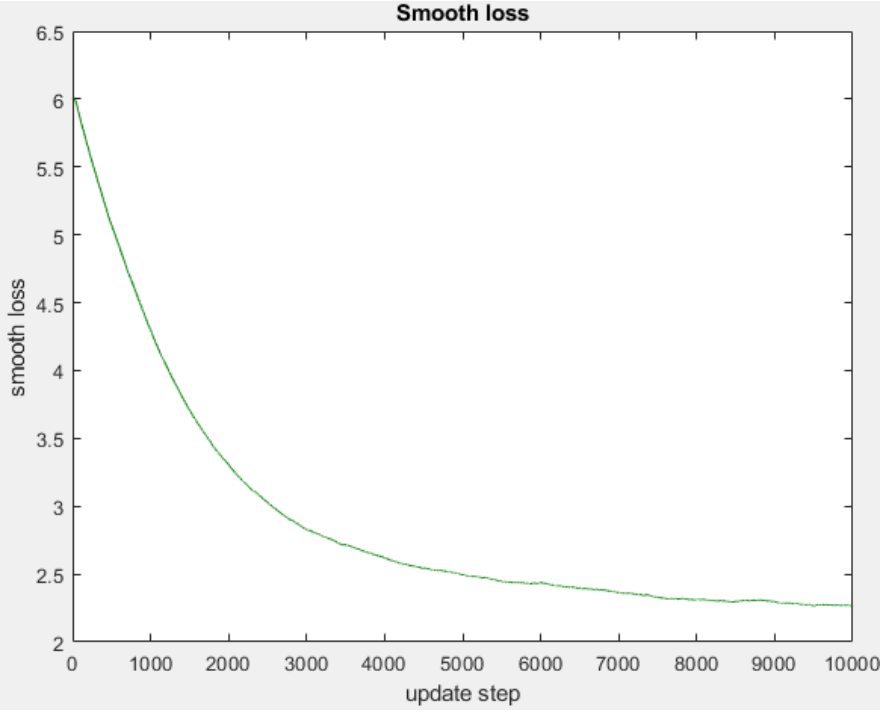

Figure 4: Plot of smooth loss for a of 100,000 characters with sequence length 100. The final smooth loss per character was 2.2669.

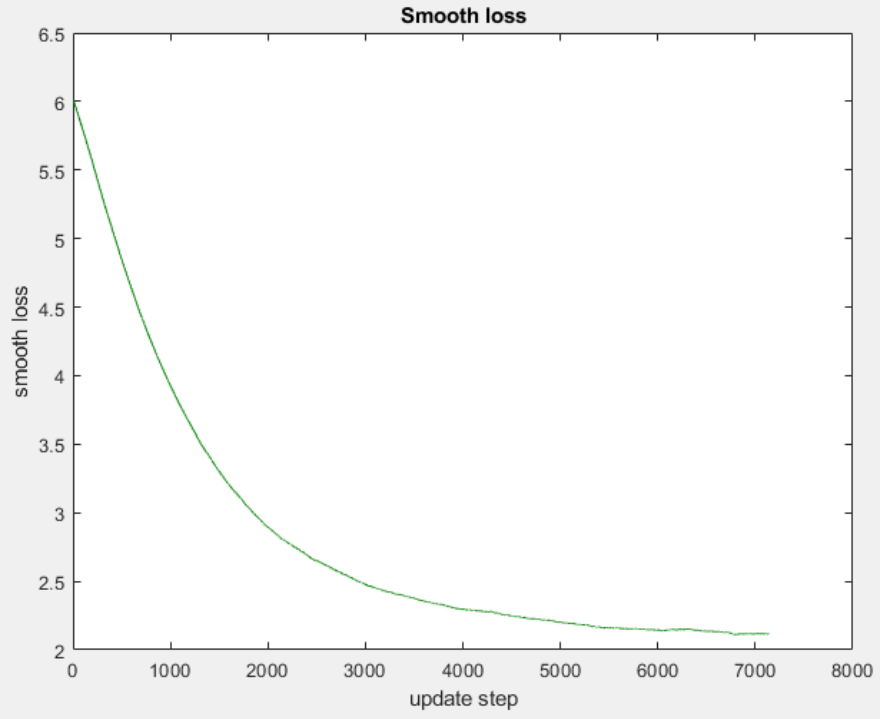

Figure 5: Plot of smooth loss for a of 100,000 characters with sequence length 140. The final smooth loss per character was 2.1162.

From the graphs above we notice that the final smooth loss initially increases with the the sequence length. However, after reaching its maximum for the sequence length of 50 it decreases afterwards. Thus, the sequence length of 140 (the maximum length of a tweet) will be used in the coming training.

4 Evolution of synthesized tweets

In the following section a sample of synthesized tweets will be presented for every 10,000th update step for a training run of 100,000 update steps.

4.1 Iteration 1

Smooth loss per character = 6.0090

Figure 6: A synthesized tweet after first iteration.

4.2 Iteration 10,000

Smooth loss per character = 2.0756

Figure 7: A synthesized tweet after 10,000 iterations.

4.3 Iteration 20,000

Smooth loss per character = 1.9214

Figure 8: A synthesized tweet after 20,000 iterations.

4.4 Iteration 30,000

Smooth loss per character = 1.8453

Figure 9: A synthesized tweet after 30,000 iterations.

4.5 Iteration 40,000

Smooth loss per character = 1.7810

Figure 10: A synthesized tweet after 40,000 iterations.

4.6 Iteration 50,000

Smooth loss per character = 1.7185

Figure 11: A synthesized tweet after 50,000 iterations.

4.7 Iteration 60,000

Smooth loss per character = 1.6827

Figure 12: A synthesized tweet after 60,000 iterations.

4.8 Iteration 70,000

Smooth loss per character = 1.7424

Figure 13: A synthesized tweet after 70,000 iterations.

4.9 Iteration 80,000

Smooth loss per character = 1.8503

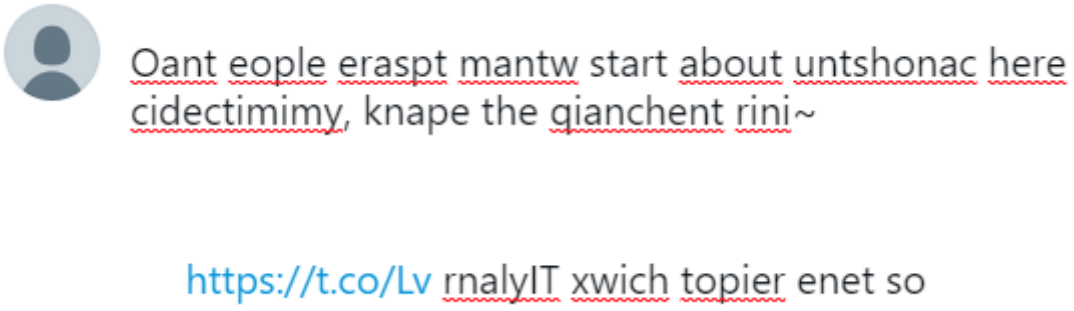

Figure 14: A synthesized tweet after 80,000 iterations. One apparently can't tweet an invalid URL.

4.10 Iteration 90,000

Smooth loss per character = 1.8161

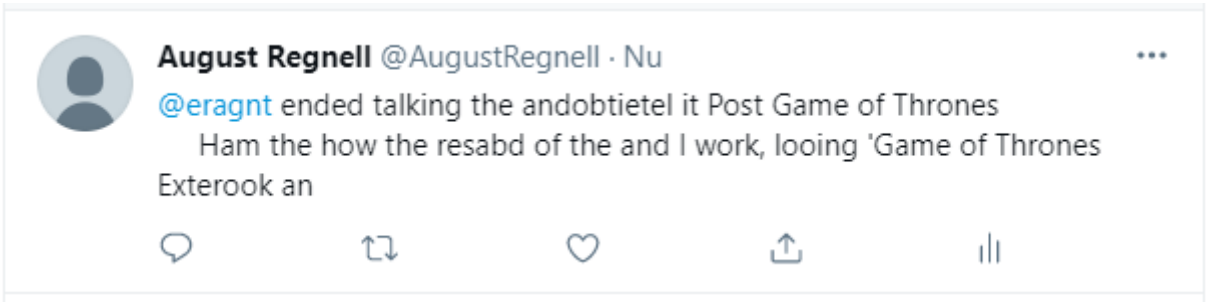

Figure 15: A synthesized tweet after 90,000 iterations.

4.11 Iteration 100,000

Smooth loss per character = 1.7721

Figure 16: A synthesized tweet after 100,000 iterations.

5 Comment on the synthesized tweets

While none of the tweets can be called comprehensible, they do bare a resemblance with real tweets. Firstly, they quite often generate #'s and @'s, which are heavily used on twitter - one even uses the hashtag GOT (**G**ame **o**f **T**hrones)! Secondly, sometimes twitter-links are generated, as in figure 14 - the first part follows the structure of a twitter-link but the RNN was sadly noy able to generate a working link. Finally, they do not exceed the cap of 140 characters - if they had I would have had to rethink my programming skills.

Now to some short comments.

- The RNN was able to synthesize several real words
- The RNN seems to easily learn the phrase *Game of Thrones*. (Hurrah!)
- The RNN was not able to generate any names of the characters in the series
- The smooth loss does not steadily decrease, indicating that the dataset isn't very uniform (or that the learning rate is not adapted to the model).
- My favorite synthesized tweet is in figure 9.
- In figure 6 one notices the increased number of unique characters. From the tweets after it one notices that these characters almost never appear again.