Analysis of hyper-parameters and Novel Encoding Methods on Character-Level LSTM Performance

1. Abstract

This project aims to optimize the content generated by character-level text-based LSTM. Particularly, since word completeness is quickly attainable with LSTM, this experiment focuses especially on the coherence of generated results and whether parts could potentially pass as human-written text. In order to explore this, various hyper-parameters were tested, such as hidden size, layer amount, learning rate, dropout, and training sequence length. Additionally, and probably more significantly, the experimentation also explores novel ways of encoding characters that would be fed to the LSTM model. These ways include encoding every *unique* pair of characters within the text as its own "character" within the model, as well as encoding every *possible* pair of characters that could be formed using those within the text dataset. In essence, these encoding schemes treat "ai" or "L:" the same as "A" or "!". And the idea between the two is that, while both offer potential to provide more meaningful contextual information, the first might save greatly on efficiency while the second could potentially allow the network to learn some flexibility around the tricky restraint that is generating its results in pairs of two characters.

Overall, it was found that while there is potential promise in these unique encoding schemes, the generated character sequences are still hard to draw semantic meaning from, and without any objective way to ascertain interpretability of results, no significant results were observed. However, outside of the encoding methods, the testing found many hyperparameters to be significantly impactful on model accuracy, result readability, and grammatical accuracy. And with the right hyperparameters, the character-pair encoding schemes were showing noteworthy production ability despite high training losses. It would be useful to look more into some of these encoding schemes, especially in the context of multiple encoding schemes simultaneously or in tandem with a more powerful model+method, such as transformers.

2. Introduction

AI text production has a plethora of real world applications. To see this first hand, one need not look further than mainstream media outlets, which seem to have a new article every week about ChatGPT and other large language models. Transformers are taking over. And while transformers and various other powerful computing models garner more and more attention, it becomes increasingly important to gain understanding by looking into the fundamental concepts and models which more advanced techniques evolved from, including RNNs and LSTM.

During initial research, I was drawn to LSTM because it is relatively straightforward while also being significantly more powerful than basic RNN's. In line with this, choosing to implement an LSTM model allowed me to focus more on other contributing factors, such as hyper-parameters and encoding schemes (though admittedly, LSTM was still a lot harder to implement than I had initially thought it would be). As I read more, I came across an article that compared the results of a word-level LSTM to those of character-level LSTMs. In it, the author Ruthu Sanketh deduced that

“word based LSTM model in general has a very poor prediction of next words,” and it seemed to me that character-level LSTM models had more potential in terms of producing coherent sentences. But it is surprising that the word-level modeling would not naturally capture context more effectively, if for no other reason than it can cast a larger (albeit less detailed) net over the sampled text. I figure that, if for no other reason than to be thorough, it is worth looking into the in-between, and thus encoding based on short character combinations seems to be the reasonable direction to explore. Plus, most words are made up of shorter, meaningful phrases (prefixes, suffixes, etc), so looking at text from this perspective might uncover lots of useful information about the formation of words themselves.

In an ideal case, you get the best of both worlds, where you can pay attention to the smaller details of punctuation while still getting word-level meaning from prefixes and suffixes. All without having to code any specific implementation and wrapped up into a neat little 1-network package. Thus, it was decided to investigate such a contextual apporach through the encoding of characters, by looking at character pairs within the text instead of individual, isolated characters.

3. Methods

The first order of business was to get an LSTM model up and running, then do some initial tests. To do this, I pulled text from Andrej Karpathy’s *tinyshakespeare* dataset. Next, it was read from and encoded according to a list of all printable characters in Python, which was then fed into a basic network with 1 LSTM module and 1 linear module with an output size of 100 (the number of printable characters). There were no dropout layers, Adam was used as the optimization method with a constant learning rate of 0.005, and cross entropy loss was used for the loss function. This network was then trained with various hyper-parameters, with 10,000 iterations of 64 (batch size) sequences of length 128 characters per iteration. The initial hyper-parameters tested include number (1, 2, and 3) and size of hidden layers (100, 200, 300, 500, and 700) in the LSTM module. Learning rate was not found to make a significant difference at this step, so it was not tested formally.

After initial info gathering, I moved to testing these methods in the context of the novel encoding techniques. 3 different techniques were used here. The first was essentially the same as the initial methods, except with a reduction to 65 characters total, because the tinyshakespeare dataset only has 65 unique characters within its entirety. This was decided to be reasonable because the model would have no reason to produce characters outside of those provided to it, and it saves on processing demands. The second encoding method involved a step-by-step search through the entire input space of the text, encoding every unique 2-character sequence found (1403 total) to a Python dictionary with a reference number. The idea behind this is that, by encoding more information (2 per number vs 1), it might allow the model to capture more context per prediction while not adding too much computational complexity. The final encoding method was similar to the second, but instead formed a dictionary including all possible unique pairs from the 65 characters within the dataset (4225 total, or 65^2). The idea behind this was that, during character generation, the second model might have been limited to pairs that were in the training set. And that it might have more expressive ability if it could generate character pairs beyond that. For example,

assume ‘m’ is only ever used after a space character within the input text. Under this context, the model would then only ever be able to express ‘m’ within words and not as a starting character to a word, significantly limiting its expressive ability. It would have been useful to incorporate a specific method to encourage the network to express the pairs that aren’t present within the training data but are still grammatically accurate, but I did not have the time nor inspiration to implement such a method.

With these new encoding methods, different hyper-parameters were tested, but with less variation because certain ones had been ruled out during initial testing. Specifically, only 2 and 3 hidden layers were tested and with sizes of 300-500. Sequence length was ultimately varied as well, in hopes of capturing sentence-level context, but it was not found to garner any significant improvement in accuracy or interpretability.

4. Experiments and Results

Between all of the encoding techniques and hyper-parameters tested, there was a lot of variation in results. Though, unfortunately, none of the differences were stark enough to garner conclusive results. It is possible that as models got larger in terms of contextual scope (more layers, more hidden nodes, double-encoded), that more semantic meaning and relevance was present, but not in any clear or directly measurable way. It was mostly just that certain parts of sentences and groups of sentences began to make more sense to me, but this is hardly a reliable metric. It is possible that with more testing and a more concrete way to assess semantic meaning, differences might have been found, but in this case, the finding would be that there appears no significant improvement with the use of the double or max encoding method.

To test the various parameters and encoding methods, character sequences of length 200-1000 were generated randomly from various priming words and characters. Some of the primers included were ‘W’, ‘ ROMEO’, ‘when’, and ‘ha’. It could have been useful to randomly generate priming sequences, but in the context of finding significant semantic differences, I don’t think this would have been any more or less revealing. Just more controlled. And considering that no measurable improvements were found, this difference would be negligible.

5. Discussion and Conclusion

The results from the study were twofold. Firstly, hyperparameters mahaveke a extremely powerful impact on the quality of LSTM results. Secondly, while it is possible that encoding characters as pairs might allow for more context to be captured by LSTM models, it is not immediately or visible impactful in a significant way. And if it were to make a difference, it would require careful percision and lots of optimizing. It would be worth exploring mores objective measurements for semantic meaning, so that results such as those from this study can be more interpretable. Too, it would have helped to do tests on text containing more up-to-date language, as it is hard for me and most individuals to interpret Shakespearean English to begin with.

If I had more time, I would have loved to dive into word-based encoding, but I was struggling to implement and understand the aspects of this project for so long that I did not end up having enough time left. From what I had gathered through preemptive

research, it is difficult to obtain meaningful results with word-level LSTM implementations without lots of training time and processing, but this is not enough reason to rule them out completely.

Additionally, this was not the most graceful implementation of a multi-character level encoded LSTM. For starters, I only tried encoding in pairs. On top of that, there were certain bits and pieces that could have been implemented more rigorously if I had the time, such as how to encourage the use of character pairs that don't exist within text. Or some sort of added functionality that allowed for odd sequence-length generation.

Overall, it seems that where varying encoding methods may offer the most promise is in the framework of considering different levels while generating text. For example, a model that considers text at the character level, sequence level, and word level simultaneously.

6. References

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

https://towardsdatascience.com/word-and-character-based-lstms-12eb65f779c2

Chat GPT also helped to facilitate+debug different code segments and introduced me to many new techniques, such as the 'yield' keyword, which were very enriching to try and implement