Wondering if Were RNNs All We Needed?

1st Semester of 2023-2024

Besel Adrian
marin-adrian.besel@s.unibuc.ro

Stroescu Dragos
dragos-ionut.stroescu@s.unibuc.ro

Popescu George
andrei-geroge.popescu@s.unibuc.ro

Abstract

Transformers changed the landscape of deep learning when they were first introduced in 2017. Still, because of their scalability limitations, people felt the need to revisit the Recurrent Neural Networks (RNNs) which dominated the field for two decades before the Transformers. The RNNs that we will refer to are LSTMs (1997) and GRUs (2014). In this paper, we are trying to examine the models proposed by the paper "Were RNNs all we needed?" replicate them, and come to a conclusion about their capabilities.

1 Introduction

Why this? We chose this project because we thought that the RNNs are an interesting subject, and we wanted to learn more about them. RNNs are sequence models that maintain a hidden state across time steps, capturing temporal dependencies. However, because of this, they are vulnerable to vanishing and exploding gradients which limit their ability to learn long-term dependencies. To address these issues S.Hochreiter and J.Schmidhuber (1997) introduced Long Short-Term Memory (LSTM) Networks that are specifically designed to mitigate the exploding and vanishing gradient problem.

Even though the LSTM networks and the Gated Recurrent Units (GRUs) (Kyunghyun Cho (2014)) have been among the best methods for sequence modeling tasks like machine translation and text generation, their sequential nature always caused a great limitation in parallelization which made them computationally inefficient and too slow to train on long sequences, they require backpropagation through time (BTT) during training which limits their ability to scale long contexts.

The original paper includes implementations of minimal LSTM and minimal GRU, lightweight models capable of remarkable results that require fewer weights and train faster, we wanted to replicate their architecture, training, and testing environments to get a sense for ourselves just how good they are relative to their classic counterpart and a classic transformer.

2 Approach

Were RNNs all that we needed?(Leo Feng (2024)) Our approach was pretty straightforward and consisted of three big steps. The first step was finding a task and a decent dataset for it. We decided to try and replicate the language modeling task presented in the paper. Since there were no concrete results in the paper, this was the perfect experiment to see what a minLSTM can do against a transformer and a classic LSTM, and answer the question of whether RNNs were all we needed.

The experiment consists of generating text character by character. The dataset that the models were trained on, is the Shakespeare dataset, consisting of 1075394 characters, out of which 65 were unique.

As for the architectures, we used minLSTM as the authors presented in their study, a classical LSTM model, while for the transformer, we used a nanoGPT presented in Karpathy (2022). It's worth mentioning that for the recurrent models, we didn't rely on the optimized PyTorch models and instead created them from scratch using indications from the paper and the Lightning Deep Learning framework. Our idea was to keep them as simple as possible so we would have a fair environment to compare the performance.

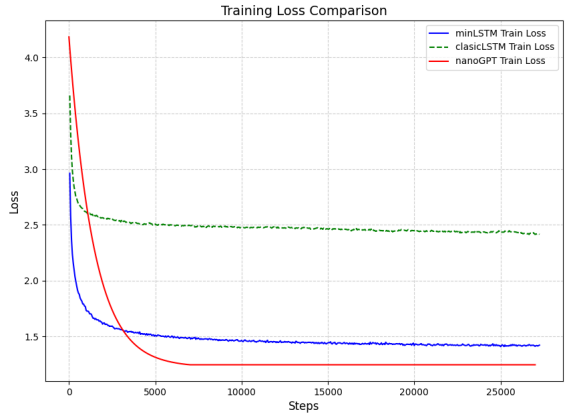
To make the minLSTM and classic LSTM models work with our dataset we had to add a few extra machine-learning components: an embedding layer before the actual LSTM (minLSTM) cell for

a better understanding of the character so that the input could be in numeric form, embedding of size 128; and a fully connected final layer that returns the most probable next character.

The 2-layer minLSTM model was trained on a NVIDIA GTX 1050, using CUDA technology. The training process took approximately 10 hours when the minLSTM was trained on the Shakespeare dataset with 27000 iterations. The loss on the train was 1.4116.

The nanoGPT transformer was replicated from Karpathy 2022, and it was trained on a CPU with 27000 iterations while having the same number of layers as the minLSTM. The loss obtained by the nanoGPT on the train was 1.2236.

The training for the classic LSTM was made on an NVIDIA GTX 1650TI, using the same amount of layers and taking the same amount of iterations as the previous models. The LSTM loss was 2.4181.



As we can see from the loss plot, minLSTM is performing way better than the classic LSTM on the training data, but he, in turn, is performing worse than the nanoGPT. the results when put up to the task of generating text, were a bit surprising. While all the models succeeded in understanding the structure of the text, only the transformer is relatively close to replicating Shakespeare's writing, even though, the recurrent models are really weak and can't put words together with meaning, ending up just repeating the same sequence over and over again.

3 Limitations

GPUs, dataset and time. One of our main limitations was the computational power required to train the models. We wanted to try to replicate as much as possible at least one of the tasks found in the paper, so we went with the Shakespeare dataset which when taking into consideration the big sequence length (100), the hidden state size (128), and the number of layers (2) we needed to train 116k parameters for the minLSTM and 270k for the classic LSTM (these numbers include the embedding no of parameters 8.3k and the final fully connected layer 8.4k). Because of this, we could only compare minLSTM to Classical LSTM and nanoGPT, we would have liked to try and test just how good minGRU is but the computational resources were scarce. So in our time, we concluded that minLSTM could give us a glimpse of just how good minimal LSTMs could be.

4 Conclusions and Future Work

In the end. Based on the generated text obtained from each of the three models, the only one at least close to rivaling Shakespeare's mastery was the transformer. With the results we've got, seems like we can safely say that the Transformers were a game-changer technology in this industry.

By all means our experiment is not a perfect project and there is room for improvement. You can try different datasets, simpler or larger, you can also try to train the LSTM models on 3 or even 4 layers that will most likely get better results. With better equipment, more time can be spent on training the models. Maybe in the future, it would be a nice experience to try and implement minGRU, train it more, and compare it to a transformer, more time and finding a task more suitable to RNNs could spotlight the strengths of minimal RNNs.

There was a lot to learn while working on this project. We learned to work better with PyTorch, Lightning framework, and LSTM. Also, it's nice to know that LSTMs are still used today and are not obsolete. The project was really interesting and was pretty fun to experiment with different architectures and let them train for hours and hours. The downside though was the pretty underwhelming results, we expected at least a competition between minLSTM and nanoGPT but it wasn't even close. For sure things could have been more enjoyable if the recurrent models were better, and their results were at least close to the

results of the transformer, but overall, the project was pretty engaging and we liked working on it.

There were some things that, to be honest, we did not understand from the original paper, regarding minLSTM because that is the one we got to play more with, why just by getting rid of the hidden state and removing some gates should the model perform better than the classical variant, we understand the necessity of parallel training but the transformers may still be better than the minimal RNNs.

Source Code

Who did what?

- Besel Adrian - Found the Shakespeare dataset and trained the classic LSTM.
- Popescu George - Wrote the minLSTM code from scratch and trained the minLSTM model.
- Stroescu Dragos - Reproduced Karpathy (2022) experiment to train the transformer.
- Together we wrote the paper in LaTeX.

References

Andrej Karpathy. 2022. nanogpt. Github repository.

Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares Holger Schwenk Yoshua Bengio Kyunghyun Cho, Bart van Merrienboer. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*.

Mohamed Ahmed Yoshua Bengio Hossein Hajimirsadeghi Leo Feng, Federick Tung. 2024. Were rnns all we needed?

S.Hochreiter and J.Schmidhuber. 1997. Long short-term memory. *Neural Computation*.