TU WIEN Informatics

Capturing Long-Term Dependencies in Neural Regulatory Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Hannes Brantner, BSc
Registration Number 01614466

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu, BSc
Assistance: Dipl.-Ing. Mathias Lechner, BSc

Vienna, 31st January, 2021

Hannes Brantner	Radu Grosu

Declaration of Authorship

Hannes Brantner, BSc

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 31st January, 2021

Hannes Brantner

Acknowledgements

Abstract

Contents

CHAPTER 1

Introduction

CHAPTER 2

Architecture

CHAPTER 3

Experiments

All models and benchmarks can be found under:
`https://github.com/Oidlichtnwoada/LongTermDependenciesLearning`

3.1 Models

3.1.1 Differentiable Neural Computer

This model defines a memory-augmented neural network architecture, that consists of a controller, read and write heads and obviously an external memory that is not parameterized by the neural network parameter. Furthermore, the external memory was structured in rows, where each memory row has a specific length. The architecture was taken from [GWR+16] and is an enhancement to the Neural Turing Machine firstly introduced in [GWD14]. The Neural Turing Machine introduced differentiable read and write functions that allow to access the memory by context or by location in both read and write mode. The access by context was implemented by comparing the cosine distance of an emitted key vector to all memory row contents and by applying the softmax function to that distance vector, which yields a weight vector. The access by location was implemented by adding a possibility to interleave the previous step weights with the current step content-based weights and adding a "blurry" shift operation on top of it. These weight vectors are then normalized using a softmax function that takes each argument to the power of an emitted number to sharpen the weights. Some improvements of the Differentiable Neural Computer include a memory management system that is able to allocate and free memory in the external memory to avoid overwriting of important information and a memory use link matrix that allows the model to track its memory operations through time.

3.1.2 Memory Cell

The Memory Cell is a simple RNN consisting of two LTC neurons as described in [HLA+20] and used in [LHA+20]. However the leakage term was removed from the ordinary differential equation describing the state dynamics, as a fading potential would mean losing information stored in the memory cell over time. It has a 2-dimensional input, the input voltage for each of the two neurons delivered by a synapse, and a 1-dimensional output, which is just the potential of the first neuron. This model was implemented to tune state dynamics and parameters for the suggested memory cell architecture. Each cell has three incoming synapses:

- an excitatory synapse that delivers the input voltage over a synapse to the neuron
- a recurrent excitatory synapse that connects each neuron with itself, which is useful to maintain an excitation in a single neuron
- an inhibitory synapse from the other neuron, to create mutual exclusive excitation

This simple memory cell should now be able to store information over a long time horizon, the input vectors are provided in the right way. Recent results have shown that this architecture is not capable of repeatedly storing information and I am not sure in what range the input values shall lie. As described in [HLA+20], the synaptic current is computed by multiplying a non-linear conductance with a potential difference. The potential difference is just a parameter E_{ij} minus the potential of the postsynaptic neuron V_j. However, if the potentials are not bounded, this would mean even an excitatory synapse can deliver a negative current or analogously an inhibitory synapse can deliver a positive current. The bounded dynamics of LTC networks is no longer valid, as the leakage term was removed from the state dynamics equation.

3.1.3 Memory Cell NCP

This is a wiring that was written for the keras-ncp Python package derived from [LHA+20] that implements the memory cell architecture from the previous section in a parallel manner, however with a leakage current term in the state dynamics. However results have shown that it is not really able to store information over a long time horizon.

3.1.4 Memory Layer

This model implements the architecture of the memory cell, in a parallelized manner. It is therefore also an RNN and includes an input and output control, which are simply multi-layer perceptrons. The input control gets the current step RNN inputs and the current memory state as input and transforms this information to an input current to each memory cell without passing it through a synaptic non-linearity. One neuron in each memory cell gets the unmodified input from the input control, the other neuron gets the negated input current as input. The output of the whole architecture is then

computed by passing the potentials of every first neuron in each memory cell through the output control, that generated the final RNN output. This model also performs not as well as expected.

3.1.5 Memory Layer Attention

To possibly improve the Transformer architecture described in [VSP+17], a new mechanism instead of multi-head attention must be implemented. Therefore, memory layer attention was created which simply concatenates every query vector with each value vector and feeds all concatenated vectors per query as a sequence to an RNN architecture. As an RNN architecture, the predescribed memory layer RNN is used. The output of the RNN is then the new representation of the query vector at the input of the memory layer attention. This architecture does not use separate key and value vectors and combines them in a single representation. A similar approach was taken in the Reformer architecture [KuKL20], which is a computationally more efficient transformer. This approach was not thoroughly tested as the basic building block, the memory layer RNN does not work as expected.

3.1.6 Transformer

This model implements the transformer architecture firstly introduced in [VSP+17]. The input and output embedding were replaced with a fully connected layer, as this model should also work with vector sized input. The linear layer at the output of the transformer was also replaced with a fully connected layer, which emits an output token of the specified token size. Instead of stopping the transformer architecture when the stop token was emitted by the decoder layer, this architecture stops after a specified amount of tokens. Moreover, the positional input encoding was extended to also support non-equidistant positions at the input, occurring when sampling irregularly. Therefore, the input to the transformer is a tuple, which must include position intervals, between two successive inputs. These are then used to build a cumulative sum to generate the absolute positions. If you just want to use the regularly sampled mode, just provide only 1s for the intervals. This architecture performs very well if a reasonably sized model with enough parameters is chosen.

3.1.7 Recurrent Transformer

This is a slightly modified version of the transformer model 3.1.6. It does not simply add the weighted value vectors up in the multi-head attention mechanism, but instead passes the weighted value vectors through an RNN, whose final output vector represents the added weighted values vectors in the ordinary multi-head attention. This architecture also performs reasonably well in the benchmarks.

3.1.8 Neural Circuit Policies

This model is a specifically connected LTC network [HLA+20], firstly described in [LHA+20]. It is very demanding in computational resources and does not perform very well on the benchmarks.

3.1.9 Unitary RNN

Unitary RNNs were firstly introduced in [ASB16] and made learning long-term dependencies with RNNs possible by keeping gradient sizes stable, even if backpropagation occurs to a distant past state. It is achieved by parameterizing the weight matrix of the hidden state in the next state equation in a way, that it stays a unitary matrix. This matrix can be constructed out of real numbers, which was done in [ASB16], but it can also be extended to the complex domain, which was done in [JSD+17]. With this approach, it can be shown that distant past inputs have the same influence on the current state as current inputs in terms of their gradient size, which only differs by a constant.

3.2 Benchmarks

3.2.1 Walker

This benchmark evaluates how well a model can predict a dynamical, physical system's behavior and was taken from [LH20]. The training data is acquired simulation data of the `Walker2d-v2` OpenAI gym [BCP+16] controlled by a pre-trained policy. The objective was to learn the kinematic simulation of the physics engine in an auto-regressive fashion using imitation learning. To increase the task difficulty, the simulation data was acquired from different training stages of the pre-trained policy (between 500 and 1200 Proximal Policy Optimization iterations) and 1% of actions were overwritten by random actions. Furthermore, frame-skips were introduced by removing 10% of all time-steps to get irregularly sampled training data. Moreover, the training data was divided into equally long sequences. For non-recurrent models, the input per time-step was the whole training sequence, but future values were zero-padded to ensure the model is only able to derive future predictions from past values. The model needs to predict the simulation state in the next time-step. Mean squared error was used as loss function when training the models. Further results of recurrent neural network models doing this benchmark can be found in [RCD19]. The results are presented in the following table:

Model	Parameter Count	Mean Squared Error
Transformer	338257	0.793
Recurrent Transformer	439633	0.599
Recurrent Transformer	29617	1.102
Memory Layer Transformer	174529	7.624
Neural Circuit Policies	21889	2.297

3.2.2 Memory

This benchmark evaluates how well a model can be trained to incorporate long-term dependencies in its predictions. The structure of this benchmark was taken from [ASB16], who also tested their recurrent neural network with this long-term dependency benchmark. The training data only contains integers from 0 to $N-1$. The first ten input symbols of each input sequence are randomly sampled integers from 0 to $N-3$ (i.i.d. uniform). Then the following symbols are copies of the integer $N-2$ in the amount of the tested memory length T minus 1. After that follows a marker, which is a single integer $N-1$. Then the remaining ten symbols are again copies of the integer $N-2$. The expected output sequence starts with $T+10$ copies of $N-2$ followed by the ten randomly sampled symbols from the start of the input sequence. Each tested model must produce an output vector of size N per time-step to a apply a sparse categorical cross-entropy loss directly from the model output logits. The loss was weighted such that the loss given by each of the last ten output vectors has the same weight as all previous output vectors combined, as it is easily to optimize to predict only the same class for a long interval. Clearly, a model can only solve this task if it is able to store information over a time period in the same size of the memory length. The baseline for this benchmark's mean categorical entropy loss per batch can be set to $\frac{10\log(N-2)}{T+20}$, as a memory-less strategy can perfectly predict the first $T+10$ symbols of the required output sequence and then it predicts the remaining 10 symbols randomly from 0 to $N-3$ (i.i.d. uniform). The results are presented in the following table, where N was 10, T was 100 and the baseline therefore was 0.173:

Model	Parameter Count	Cross-Entropy Error	Correctly Memorized Symbols
LSTM	7130	0.157	1.1
Memory Layer	49720	0.400	0.0
Differentiable Neural Computer	81685	0.145	2.0
Efficient Unitary Neural Network	1674	0.009	10.0

3.2.3 Cell

This is a very small benchmark that tries to find the right parameter ranges and input ranges for the memory cell 3.1.2. It is directly implemented in the same file as the memory cell itself.

CHAPTER 4

Summary

List of Figures

List of Tables

Bibliography

[ASB16] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks, 2016.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.

[GWR+16] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[HLA+20] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks, 2020.

[JSD+17] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns, 2017.

[KuKL20] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[LH20] Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series, 2020.

[LHA+20] Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2:642–652, 10 2020.

[RCD19] Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.