Modeling Motor Neuron Timing Using a Recurrent Neural Network

Emily Whyms

July 25, 2025

Abstract

The whole paper but not quite: A Recurrent Neural Network model is trained on a dataset[1] of monkey's neuron signal times so it can eventually output the intervals when given an initial cue. This model produces graphs on the model's loss and the change in it's weights before and after training. The three kinds of Recurrent Neural Networks (RNNs) tested was a Gated Recurrent Unit(GRU), Long-Short-Term Memory(LSTM), and basic Recurrent Neural Network(RNN). some findings....

Introduction

Like circuits, your brain passes information through electrical signal. The neurons in your brain act as wires to carry those signals to the needed parts of the body in chains, or what I will refer to as neural networks. In the instance of motor neural networks, they will send signals from the brain to muscles and vice versa. When a signal reaches the muscle, the muscle will preform the task that the brain sent the signal for. This model was trained on the timing of the signals and was done to understand the learning and weight changes that the model makes to accurately obtain the timing of the network. [1] Recurrent Neural Network(RNN)s is a form of training for a model on a dataset you desire it to make future outputs for. In the dataset[1] that is being used, the model is training to produce an output like that of the readings from the monkeys. The model then predicts the outcome of the monkey's network timing given a starting input.

Methods

The Recurrent Neural Network(RNN) is trained on a motor timing dataset which is then given a set pulse to produce a timed interval once the output reaches the target. The model was run on PyTorch using Matplotlib, pandas, Neurogym, and Numpy. It was run with the basic RNN, LSTM, and GRU as parameters; however, the GRU is known to be a simpler version of an LSTM. The dimension was set too 15 and the hidden layers were set to 81. These values should be set within a range of 0-20 and 50-100, respectively, in order to get enough eigenvalues to graph. For the amount of hidden layers, it must be equal to a positive integer when square-rooted since you must pull the eigen values from a tensor later. The only way to do that is to have the matrices inside the tensor be squares such as 8x8 or 9x9.

A RNN is much like a feedforward network in the sense that it takes an input, runs it through a few hidden layers with weights, and spits out an output influenced by those layers. Where it differs is how the hidden layers are used. RNNs will sum up their data in every layer since they like sharing their information and have a common weight. This weight can change as the RNN is fed more data based on previous input data. The RNNs back propagation is what allows it to be a better learner since it starts with the end data and will change weights according to what it needs to get the answers going backward.

The Basic RNN uses what we call short-term memory. For example, if it is given a sentences and is tasked to find the next word in a new sentence, it will use the words within the sentence it is trying to find the word for to find that word. Issues can arise from this, since the information of that next word could be in the sentences prior to the sentence it is on. That's where long-short-term memory(LSTM)s and gated recurrent unit(GRU)s come in hand. These RNNs solve the problem that arose with the basic RNN. To add to the example, they use the previous sentences in their long-term memory to help influence the decision when outputting the next word. To do this, they use states to store the previous sentences. Then explain states because why not? T-T

GRUs are known to be a simpler version of an LSTM, having 2 gates instead of 3 and requiring fewer parameters to run. Gates are a way of RNNs to manage data. The gates the LSTM uses are input, forget, and output gates. The two gates that the GRU uses are the reset and update gates. These are implemented as sigmoid functions

$$S(x) = \frac{1}{1 + e^{-x}}$$

which are functions that maps real values to a number between 0 and 1. Explanation of parameters. LSTMs are goofy and silly little guys that is the main RNN that would be used on this dataset.

Results

The loss graph of the basic RNN starts with a steep drop in loss but has a few large loss spikes in the middle. This can tell us that the RNNs short term memory isn't a good fit for this dataset and that it needs data from past inputs further than the basic RNN can store. On the other hand, the long-term memory models(LSTM and GRU) were able to stay flat after the initial drop in loss. For figs, reference at the end of a sentence mentioning them using (figure num)

Graphs in order : RNN, LSTM, GRU

weights and how it changes? because rnn weight is different from linear weight. Quick explanation for future me: rnn weight is the input, hidden layers, and hidden layer dimensions. Linear weight is going to be the output with the hidden layers and dimensions. yes they are very different, the dictionary made the linear weight 2 matrices and the rnn weight a stupidly giant stand alone matrix.

Discussion

I don't know what to put here yet... (this should tell the reader why this is important... I think) Implications of what was found

The RNN is modeling the time it takes for a neuron to respond since the physical reaction was too slow to (gather brain fast data) *(stupid way of saying, so find a better option) The original research was testing monkey's movement and reaching behaviors? *(fact check that)

An issue that arose was the initial weights. I believe each dimension has its own initial weight but this model only used the very first initial weight of the first dimension. Following that, the learned weights got condensed over the training time, leaving only a few eigenvalues to plot. It still gave a good understanding of the model's learning, but could be improved upon.

The next steps toward improving the model is to fix the compounding of eigenvalues and knowing which parameters are changing the values of the initial and output eigenvalues. Following up on the full project[1] and training a model on the non-motor sides of the data as well as the rest of the motor related datasets. Working with real datasets like ones from DANDI would show open up real data on neurons to look at and calculate.

References

References

[1] Jing Wang et al. "Flexible timing by temporal scaling of cortical responses". en. In: *Nat. Neurosci.* 21.1 (Jan. 2018), pp. 102–110.