2017-10-03

Simulating Recurrent Neural Networks on Graphic Processing Units

└─Introduction

└─Introduction

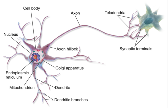


Figure 1: Anatomy of a neuron[1]

[1] By BruceBlaus - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=28761830

Let me start by motivating it by this picture of a neuron. The features that define a neuron are electrical excitability, where a neuron spikes and discharges electrical signals through the synapses, which are the complex membrance junctions that transmit signals to other neurons. In the human brain, there are approximately 10^{14} neurons and the artificial neural networks that we have in deep learning are inspired by these biological neurons.

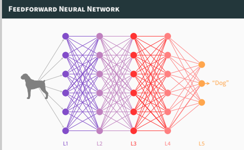
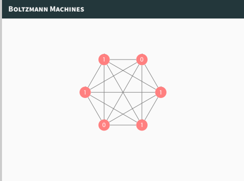


Figure 2: Feedforward Neural Network

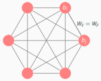
For example, we have feedforward networks where the connections between the units do not form a cycle. Feedforward networks are good for supervised learning like regression and classification in non-temporal data (data not related to time instances). Recurrent neural networks are more suited for temporal data like natural language processing and reinforcement learning and many DL researchers are now focused in understanding this kind of neural networks. Eventually we want to train large and deep RNNs and we like to parallelize the training over multiple GPUs, due to the large magnitude of linear operations involved. There are two kinds of parallelism, (1) data parallelism where data is partitioned into batches to be trained on each GPU and (2) model parallelism where the network is partitioned into subnets to be trained on each GPU. To train large networks, data parallelism will not be enough, we need model parallelism, and sampling from the RNN over several GPUs will be an important step in training. We will not be talking about the training step today, but will focus on the sampling.

RNNs are networks where the connections between units form a directed cycle. One of the popular RNNs is long short term memory (LSTM), which are able to connect previous information to the present task. However, the neurons in LSTM communicate with real values, which is different from the way neurons communicate in our brain. Thus I'm going to talk about RNNs whose architecture is closer to the human brain and by building such neural network works with the number of artificial neurons coming close to the number of neurons in the human brain, we hope to possibly arrive at some learning theories that is close to how learning is done in the brain, if not as good as the brain. Today the two RNNs I'm going to talk about is Boltzmann machines and McCulloch-Pitts machines. They both have the spiking characteristic when we simulate them, and what differs is that the sampling for BM is done in discrete time and continuous time for MPM.



In the BM, it is composed of primitive computing elements called units (neurons). Each unit has two possibly states, on or off, represented by $\{0, 1\}$. The units are connected to each other by bi-directional edges and can take on any real value. Bi-directional also means that the edge weights are symmetric, having the same strength in both directions.

Boltzmann Machines



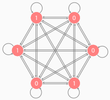
Energy configuration, $E = -\sum_{i<j} W_{ij} x_i x_j - \sum_i b_i x_i$

Energy gap, $\Delta E_i = E(x_i = 0) - E(x_i = 1) = \sum_j W_{ij} x_j + b_i$

$$p_i := \mathbb{P}(x_i = 1) = \frac{1}{1 + e^{-\Delta E_i/\tau}}$$

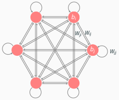
The units are binary stochastic units and depending on the energy gap, $\Delta E_i > 0 (< 0), p_i > 0.5 (< 0.5)$. Here, the temperature variable controls the amount of noise; higher temperature means more noise and also gives us a higher probability of transiting to a higher energy state and hence avoids local minimum when we do training. When $\tau \rightarrow 0$ we get Hopfield network. So in Hopfield networks, we have binary threshold units instead, meaning when the energy gap is positive, we update the state to 1 surely.

McCulloch-Pitts Machines



Now let's look at MPM. In MPM, the neurons also have binary states $\{0, 1\}$. However, the states do not mean the same thing as in BM. State 1 is the refractory state, where the neuron just fired and is unable to fire till it recovers. State 0 is the armed state, the neuron just recovered and is waiting to fire. Here we model the units with the Nossenson-Messer neuron model, which explains biological firing rates in response to external stimuli.

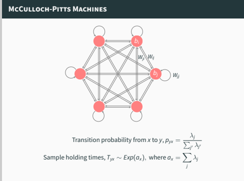
McCulloch-Pitts Machines



$$\text{Transition Energy, } E(y, x|\theta) = -\sum_{ji \in E} W_{ji} y_j x_i - \sum_{j \in V} b_j x_j - \sum_{i \in V} b_i x_i$$

$$\Gamma_{yx} = \exp\left(-\frac{1}{2\tau} E(y, x|\theta) + \frac{1}{2\tau} E(x, x|\theta)\right)$$

In the MPM, we do not require the edges to be bi-directional, thus the matrix W denoting the edge weights need not be symmetric and there are also edges from a unit back to itself. We define a transition as a state that is one hop away from the current state, i.e. differs by one bit. We shall think of x as the current state and y to be any state that is one hop away. In the computation of the transition energy, we require both the current and the future state that it is transiting to. Thus for each possible future state y, start a Poisson process with rate Γ_{yx}, hence for d neurons, we start d Poisson Processes. The neuron chosen to transit is the neuron with the smallest interarrival time, which uniquely determines the new state. This smallest interarrival time is then stored as the holding time for state x; the length of time that the machine stays in state x. Thus for each simulation of MPM, we get the new state and the time spent in the previous state.



Practically, to identify the transiting neuron, we update the linear responses and sign, following which we apply the softmax function to the λ_j's to get a probability distribution of the transitions. It seems counter-intuitive to think of 0 as armed and 1 as refractory, but it is actually a very natural idea. When a neuron transit from $0 \to 1$, it changes the value of the linear response; for a transiting neuron i, if $W_{ji} > 0$, then such a transition increases the linear response of neuron j and if $W_{ji} < 0$ it decreases the linear response of neuron j. The sign s depends on the state of the neuron, it preserves the sign of the linear response if it is armed and flips the sign of the linear response if it is refractory.

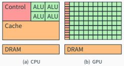


Figure 3: Comparison between the amount of transistors devoted to different functions inside a CPU and a GPU.

To simplify quite a bit, think of a GPU as a factory and a CPU as Steven Hawking. Factory workers, each represented by a core, can complete lots of easy, similar tasks with incredible efficiency?tasks like geometry and shading. On the other hand Mr. Hawking, while incredibly smart and only occasionally baffled, is just one man. His skill set is better used on singular, complex problems like artificial intelligence. (1)GPUs have an explicit programming model, the programs are written in such a way that we utilise as much of the parallel processing as much as possible. (2)GPUs are good at efficiently launching lots of threads and running them in parallel (3)GPUs optimize for throughput (maximum rate of production), not latency; they are willing to accept increase latency of any single individual computation in exchange for more computation being performed per second, the computation performed per second is measured by floating point operations per second (FLOPS)

Simulating on GPUs

Importance to Simulating on GPUs

- Faster matrix multiplication
- Larger neural networks
- Larger function space
- Energy efficiency

- train larger neural networks
- learning from a larger function space
- GPUs are more energy efficient than CPUs; they are optimized for throughput and performance per watt and not absolute performance