Simulating Recurrent Neural Networks on Graphic Processing Units

Summer Project

Zhangsheng Lai

September 28, 2017

Introduction

Introduction

Definition
A feedforward neural network is an artificial neural network where connections between the units do not form a cycle.

Definition
A recurrent neural network is an artificial neural network where connections between units form a directed cycle.

Feedforward Neural Network

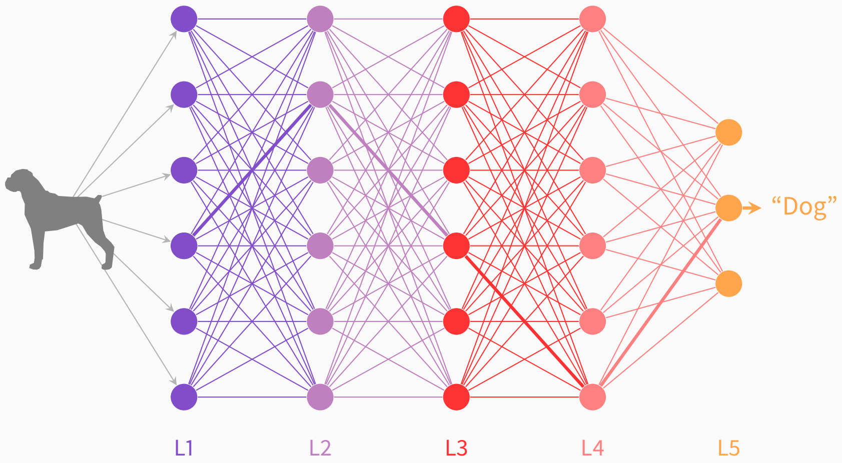
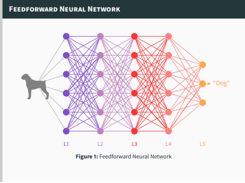


Figure 1: Feedforward Neural Network


Feedforward Neural Network

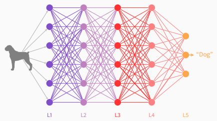


Figure 1: Feedforward Neural Network

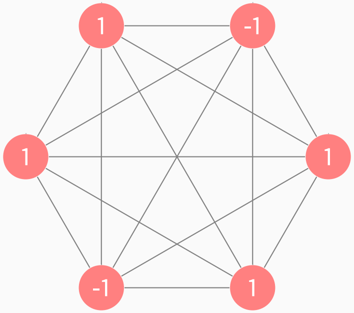
- for feedforward neural networks, we have can have supervised or unsupervised
- supervised: MNIST digits classification, unsupervised: autoencoders
- these neural networks are the more popular and mainstream ones, but today we are going to look at RNNs and how to simulate them.
- I will begin with a brief introduction of some RNNs and then introduce a RNN that does not

Recurrent Neural Network

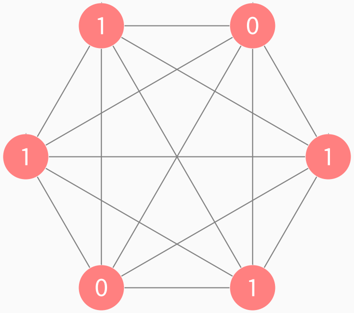
Recurrent Neural Network

- Long short term memory (LSTM) is a RNN architecture that remembers values over arbitrary intervals
- LSTMs able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame

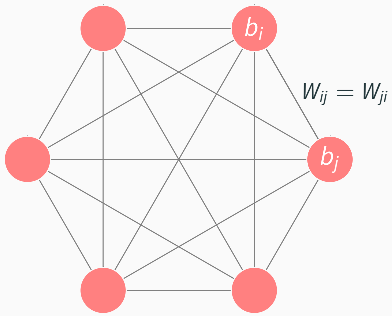
Hopfield Networks



Hopfield Networks



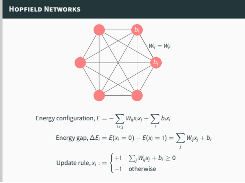
Hopfield Networks



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

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

Update rule, $x_i := \begin{cases} +1 & \sum_j W_{ij} x_j + b_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$



Hopfield Networks

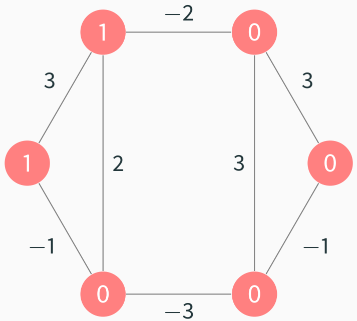
Energy configuration, $E = -\sum_{i<j} W_{ij} x_i x_j - \sum_i b_i x_i$

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

Update rule, $x_i := \begin{cases} +1 & \sum_j W_{ij} x_j + b_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$

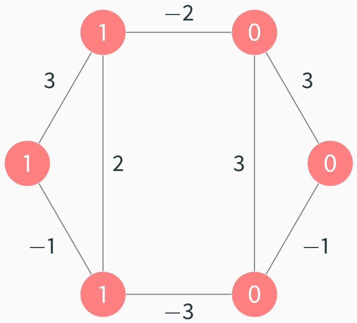
- composed of primitive computing elements called units
- units has two states, on or off, represented by $\{1, -1\}$ or $\{1, 0\}$
- connected to each other by bi-directional links
- adopts these states as a function of the states of its neighbouring units and weights of its links to them, it is a probabilistic function for a Boltzmann machine.
- weights can take on any real value
- a unit being on or off is taken to mean that the system currently accepts or rejects some elemental hypothesis of the domain
- weight on a link represents a weak pairwise constrain between two hypothesis
- positive (negative) weights indicate that two hypothesis support (contradict) one another with other things being equal
- link weights are symmetric, having the same strength in both directions

Hopfield Networks



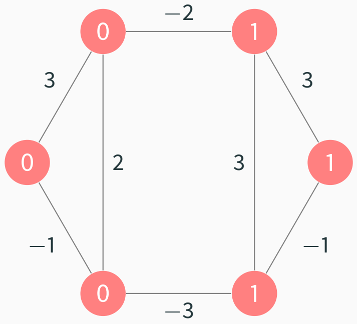
$$(1, 0, 0, 0, 0, 1)$$

Hopfield Networks



$$(1, 0, 0, 0, 0, 1)$$
$$(1, 1, 0, 0, 0, 1)$$

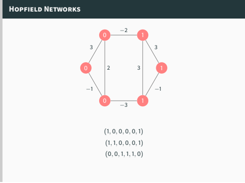
Hopfield Networks



$$(1, 0, 0, 0, 0, 1)$$

$$(1, 1, 0, 0, 0, 1)$$

$$(0, 0, 1, 1, 1, 0)$$

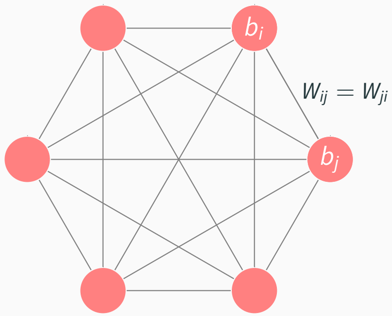


Hopfield Networks

(1, 0, 0, 0, 0, 1)
(1, 1, 0, 0, 0, 1)
(0, 0, 1, 1, 1, 0)

- Updating of Hopfield networks is done sequentially usually in a randomized order. Parallel updating might increase the energy instead.
- Hopfield networks always make decisions to reduce the energy and makes it impossible to escape from local minima.
- random noise can help us escape from poor minima, by starting with lots of noise so its easy to cross energy barriers and gradually decrease the noise so the system ends in a deep minimum. This is called simulated annealing.

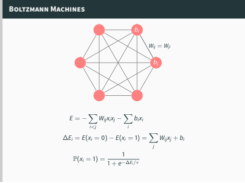
Boltzmann Machines



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

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

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



Boltzmann Machines

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

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

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

- replace the binary threshold units by binary stochastic units that make biased random decisions
- temperature variable controls the amount of noise
- when $\tau \to 0$ we get back the Hopfield network
- for $\tau_1 > \tau_2$, we are less likely to go to a lower energy state compared to in τ_1 compared to τ_2, i.e. more likely to go to a higher energy state when the temperature is higher. This allows us to escape from local minimum and arrive at the global minimum
- we simulate the BM in the similar way to how it was in the Hopfield networks, but replace the binary threshold units with binary stochastic units

Boltzmann Machines

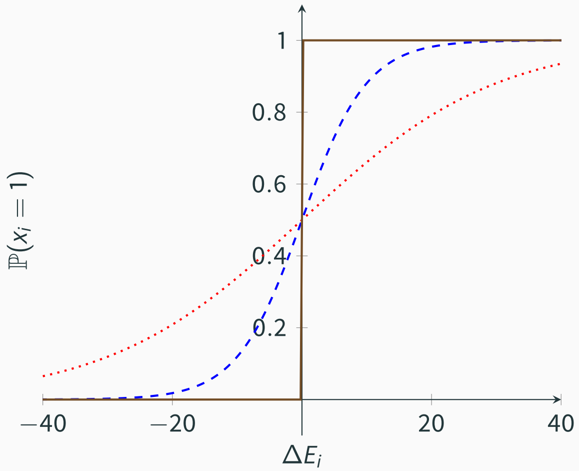
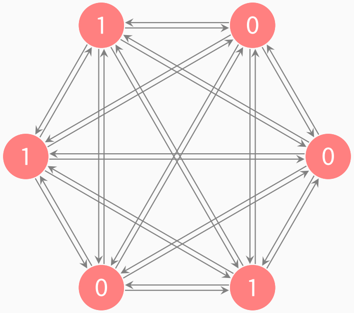
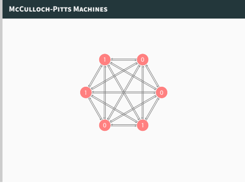


Figure 2: $\tau = 0$ (solid), $\tau = 5$ (dashed), $\tau = 15$ (dotted)

McCulloch-Pitts Machines

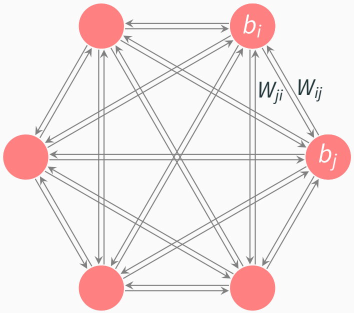
McCulloch-Pitts Machines





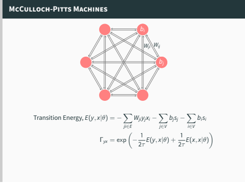
- state 1 is the refractory state, the neuron just fired and is unable to fire till it recovers
- state 0 is the armed state, the neuron just recovered and is waiting to fire
- here we model the units with the Nossenson-Messer neuron model, which explains biological firing rates in response to external stimuli

McCulloch-Pitts Machines



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

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



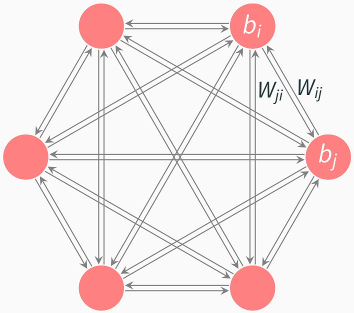
McCulloch-Pitts Machines

$$\text{Transition Energy, } E(y,x|\theta) = -\sum_{ji\in E} W_{ji}y_jx_i - \sum_{j\in V} b_js_j - \sum_{i\in V} b_is_i$$

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

- digraph, $G = (V, E)$, weights $W : V \to \mathbb{R}$, biases $b : E \to \mathbb{R}$, binary states $\mathbb{B} = \{0, 1\}$ with an initial distribution $\mathbb{B}^{|V|} \to \delta$ and a temperature τ
- allow transitions where y and x differ by only one bit
- here the W matrix need not be symmetrical with zero diagonals like what we had in the Hopfield network and Boltzmann machine models
- for each $y \neq x$, start a Poisson process with rate Γ_{yx}
- as such, we can talk about the interarrival timings of the Poisson process and our simulation of the McCulloch-Pitts machine not only gives us a binary tuple, but also the time taken from it to transit from its earlier state

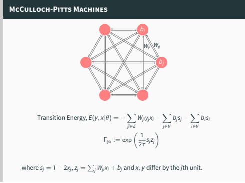
McCulloch-Pitts Machines



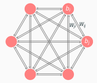
$$\text{Transition Energy, } E(y, x|\theta) = -\sum_{ji \in E} W_{ji} y_j x_i - \sum_{j \in V} b_j s_j - \sum_{i \in V} b_i s_i$$

$$\Gamma_{yx} := \exp\left(\frac{1}{2\tau} s_j z_j\right)$$

where $s_j = 1 - 2x_j$, $z_j = \sum_j W_{ji} x_i + b_j$ and x, y differ by the jth unit.



McCulloch-Pitts Machines



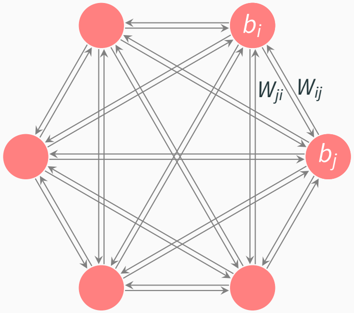
Transition Energy, $E(y, x|\theta) = -\sum_{ji \in E} W_{ji} y_j x_i - \sum_{j \in V} b_j s_j - \sum_{i \in V} b_i s_i$

$$\Gamma_{yx} := \exp\left(\frac{1}{2\tau} s_j z_j\right)$$

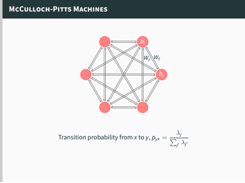
where $s_j = 1 - 2x_j$, $z_j = \sum_j W_{ji} x_i + b_j$ and x, y differ by the jth unit.

- when doing the updates we can just update the linear responses z_j and apply softmax on the λ_j's to get the probability distribution of the transitions.
- it seems counter-intuitive to think of 0 as armed and 1 as refractory, but it is in fact the most natural thinking
- a transition from $0 \longrightarrow 1$ is a act of firing and a transition from $1 \longrightarrow 0$ is the act of recovery
- when a neuron transit from $0 \longrightarrow 1$, 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

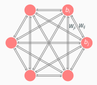
McCulloch-Pitts Machines



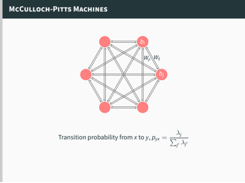
Transition probability from x to y, $p_{yx} = \frac{\lambda_j}{\sum_{j'} \lambda_{j'}}$


McCulloch-Pitts Machines

Transition probability from x to y, $p_{yx} = \frac{\lambda_j}{\sum_{j'} \lambda_{j'}}$



- when doing the updates we can just update the linear responses z_j and apply softmax on the λ_j's to get the probability distribution of the transitions.



McCulloch-Pitts Machines

Transition probability from x to y, $p_{yx} = \frac{\lambda_j}{\sum_{j'} \lambda_{j'}}$

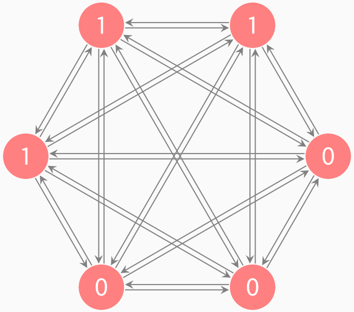
- McCulloch-Pitts machine is continuous time; we have a

McCulloch-Pitts Machines

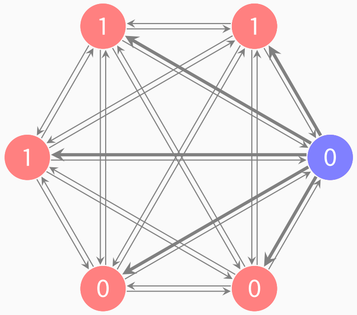
Algorithm 1 CTMC Simulation.

```
1: Initialize $\boldsymbol{w}, \boldsymbol{b}$
2: Initialize $\boldsymbol{x}^{(0)}$
3: for $i$ from 1 to $N$ do
4:     Compute $\Gamma_{yx}, P_{yx}$ for each $\boldsymbol{y}$
5:     Compute $a_x = \sum \Gamma_{yx}$
6:     $\boldsymbol{x}^{(i)} \leftarrow \text{flip}(\boldsymbol{x}^{(i-1)})$
7:     Sample holding time $t_{(i-1)} \sim \text{Exp}(a_x)$
8: end for
```

McCulloch-Pitts Machines

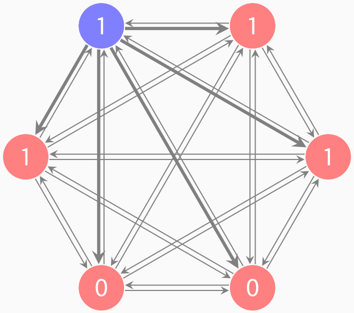


McCulloch-Pitts Machines



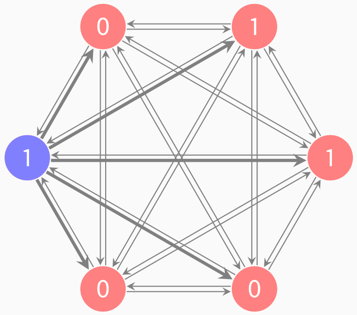
$$(T_0, (1, 0, 0, 0, 1, 1))$$

McCulloch-Pitts Machines



$$(T_0, (1, 0, 0, 0, 1, 1))$$
$$(T_1, (1, 0, 0, 1, 1, 1))$$

McCulloch-Pitts Machines



$$(T_0, (1, 0, 0, 0, 1, 1))$$
$$(T_1, (1, 0, 0, 1, 1, 1))$$
$$(T_2, (1, 0, 0, 1, 1, 0))$$

McCulloch-Pitts Machines

Simulating on GPUs

Simulating on GPUs

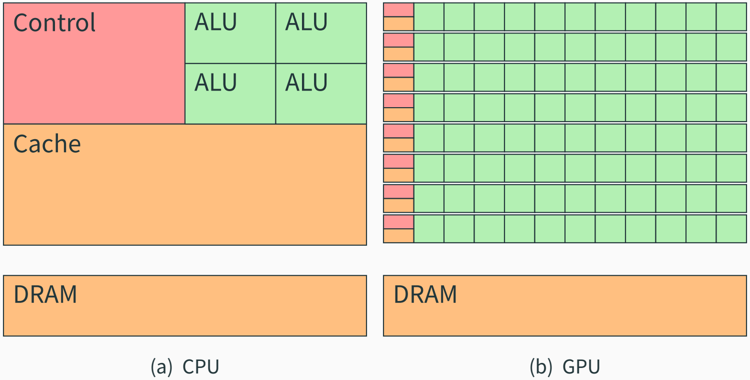


Figure 3: Comparison between the amount of transistors devoted to different functions inside a CPU and a GPU.

└─Simulating on GPUs

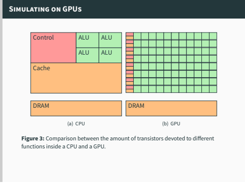
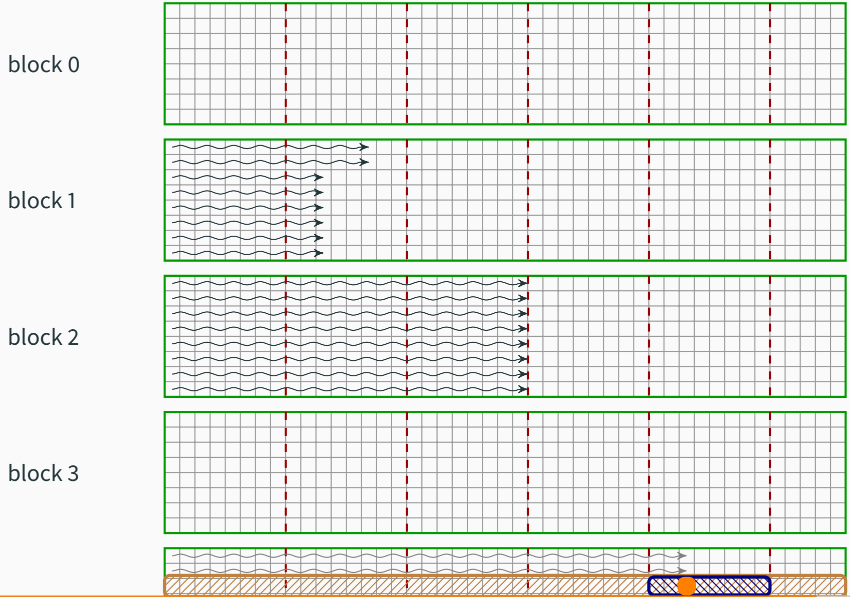


Figure 3: Comparison between the amount of transistors devoted to different functions inside a CPU and a GPU.

- 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.
- DRAM: dynamic random access memory, ALU: arithmetic logic unit,Cache, Control
-

Simulating on GPUs

Simulating on GPUs



Simulating on GPUs

References

J. Macor.
A Brief Introduction to Type Theory and the Univalence Axiom
http://math.uchicago.edu/ may/REU2015/REUPapers/Macor.pdf

The Univalent Foundations Program
Homotopy Type Theory: Univalent Foundations of Mathematics.
https://homotopytypetheory.org/book

The n-Category Café
From Set Theory to Type Theory
https://golem.ph.utexas.edu/category/2013/01/from_set_theory_to_type_theory.html

The *n*Lab
Function Type
https://ncatlab.org/nlab/show/function+type