# Introduction to Neural Networks

Yao Zhang

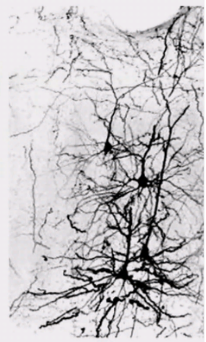
The guy is a populace

Mostly based on Thomas Hofmann's lecture in ETH

https://zhims.github.io/

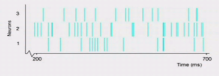
Dec 5, 2019

# Biological Neural Networks

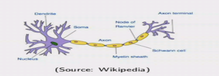


- Neurons: basic functional & structural units of nerous system
- Cells connected by nervous fibers
- Signaling via electrical impulses
- Human brain:
  - $\sim$ 100 billion neurons
  - $\sim$ 100 trillion connections
  - very (!) large scale system

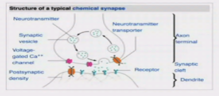
# Biological Neural Networks




(a) Neurons



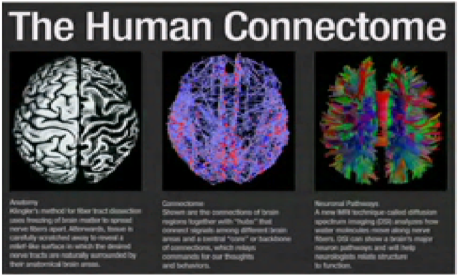

(b) Anaomy




(c) Synapses

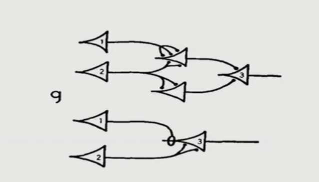
- Neurons:
  all-or-none principle = action potential (spike)
- Anatomy: Soma, dendrite, axon
- Functional:
  many inputs ($\sim 10^3 - 10^5$), one output
- Synapses: plasticity (strengthening & weakening)

# Connectome




- Scientific challenge: decipher brain connectivity
- Small scale connectivity vs. overall "wiring" (white matter pathways)
- Network analysis: Rich club(2001) $\downdownarrows$

# Boolean Abstraction



- Boolean logic view of neurons

$$f : \{0,1\}^n \rightarrow \{0,1\} \tag{1}$$

- Neural network = logical circuit

The response of any neuron is factually equivalent to a proposition which proposed its adequate stimulus

# Mathematical Abstraction

- Abstract neuron: implements real-valued function

$$f : \mathbb{R}^n \to R \subseteq \mathbb{R} \tag{2}$$

  - interpret real-valued output as firing rate or probability (ignoring temporal dynamics)
  - neuron = computational unit
- Each unit is (implicitly) parametrized by some $\theta \in \mathbb{R}^d$

$$f : \mathbb{R}^n \left( \times \mathbb{R}^d \right) \to \mathbb{R}. \tag{3}$$

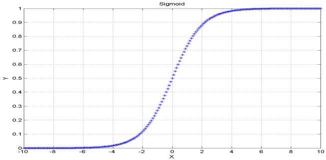
# Parameterization

Typical choice: weighted average + non-linearity

$$f(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \quad (4)$$

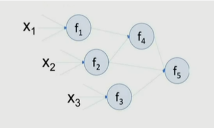
- parameterization $\theta = (b, w_1, ..., w_n)$
- weights $\{w_i\}$ = synaptic strengths, bias $b$ = threshold
- e.g. sigmoid activation function $\sigma : \mathbb{R} \to \mathbb{R}$ (soft threshold)

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (5)$$



# Mathematical Abstraction

- Simplify connectivity structure: loop-free Directed Acyclic Network (DAG)
- Activity propagation = feedforward network
- Nested functions = compositionality



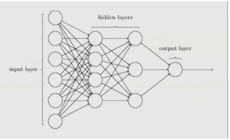

$$g(x_1, x_2, x_3) = f_5(f_4(f_1(x_1), f_2(x_2)), f_2(x_2), f_3(x_3)) \tag{6}$$

# Compositionality

- Basic idea: define complex functions in terms of compositions of simple(r) functions
- Powerful as a biological principle: common biological substrate
- Powerful as an engineering principle: universal model toolbox
- Simple & intuitive weighted-based parameterization ($\Rightarrow$ learning)
- Traditionally (ML, approximation theory): shallow networks
  Deep learning: higher degrees of nesting = depth

# Multi-Layer Perception

- DAGs model space (too) large $\Rightarrow$ simplification
- Arrange neurons in densely inter-connected layers
- Inputs = input layer
  Outputs = output layer
  Intermediate = hidden layers
- Also called: MLP (Multi-Layer Perceptron)



# Map/Matrix Notation

- Layers $l = 0, ..., L$ of dimensionality $m^l$
  - $l = 0, m^0 = n$ : input
  - $l = L, m^L = m$ : output
- Transfer map $F^l$ between layer $l - 1$ and $l$

$$F^l = \sigma^l \circ \overline{F}^l, \;\; \overline{F}^l(x) = W^l x + b^l \in \mathbb{R}^{m^l} \tag{7}$$

  - $\sigma^l$ : element-wise non-linearity of layer $l$
  - $\overline{F}^l$ : linear function in layer $l$ (pre-activations)
  - $W^L \in \mathbb{R}^{m^l \times m^{l-1}}$ : weight matrix, $b^l \in \mathbb{R}^{m_l}$ : biases
- Overall function by composition of maps

$$F = F^L \circ \cdots \circ F^1 \tag{8}$$

# Partial Derivatives

- Given parameterized map $F: \mathbb{R}^n \left(\times \mathbb{R}^d\right) \to \mathbb{R}^m$, (e.g. realized by a neural network)
- Partial derivatives w.r.t. parameter $\theta \in \left\{w_{ij}^l, b_i^l\right\}$,

$$\delta_\theta = \frac{\partial F}{\partial \theta}, \quad \delta_\theta : \mathbb{R}^n \to \mathbb{R}^m \tag{9}$$

  - same signature as $F$
  - inputs $x \in \mathbb{R}^n$ are usually "clamped" (implicitly given)
  - $\delta_\theta \in \mathbb{R}^m$ then is a vector in output space
  - how to compute these? backpropagation

# Gradient-based Learning

- Given an input-output example$(x, y)$
- Loss function: $\ell_y : \mathbb{R}^m \to \mathbb{R}$
  - e.g. $\ell_y(\nu) = \frac{1}{2}\|y - \nu\|^2, \nu = F(x)$: model prediction
- Derivatives w.r.t. parameter: provide update directions

$$\frac{\partial \ell}{\partial \theta} = \langle \nabla \ell_y, \delta_\theta \rangle \tag{10}$$

  - follows from chain rule
  - e.g. $\nabla \ell_y = \nu - y$
- Incremental adaptation step: $\theta \leftarrow \theta - \eta \ell_y (F(x))$
  - $\eta$: step size or learning rate

# Last But Not Least

Thank you all of you! –Yao