CS 4824/ECE 4424: Perceptron

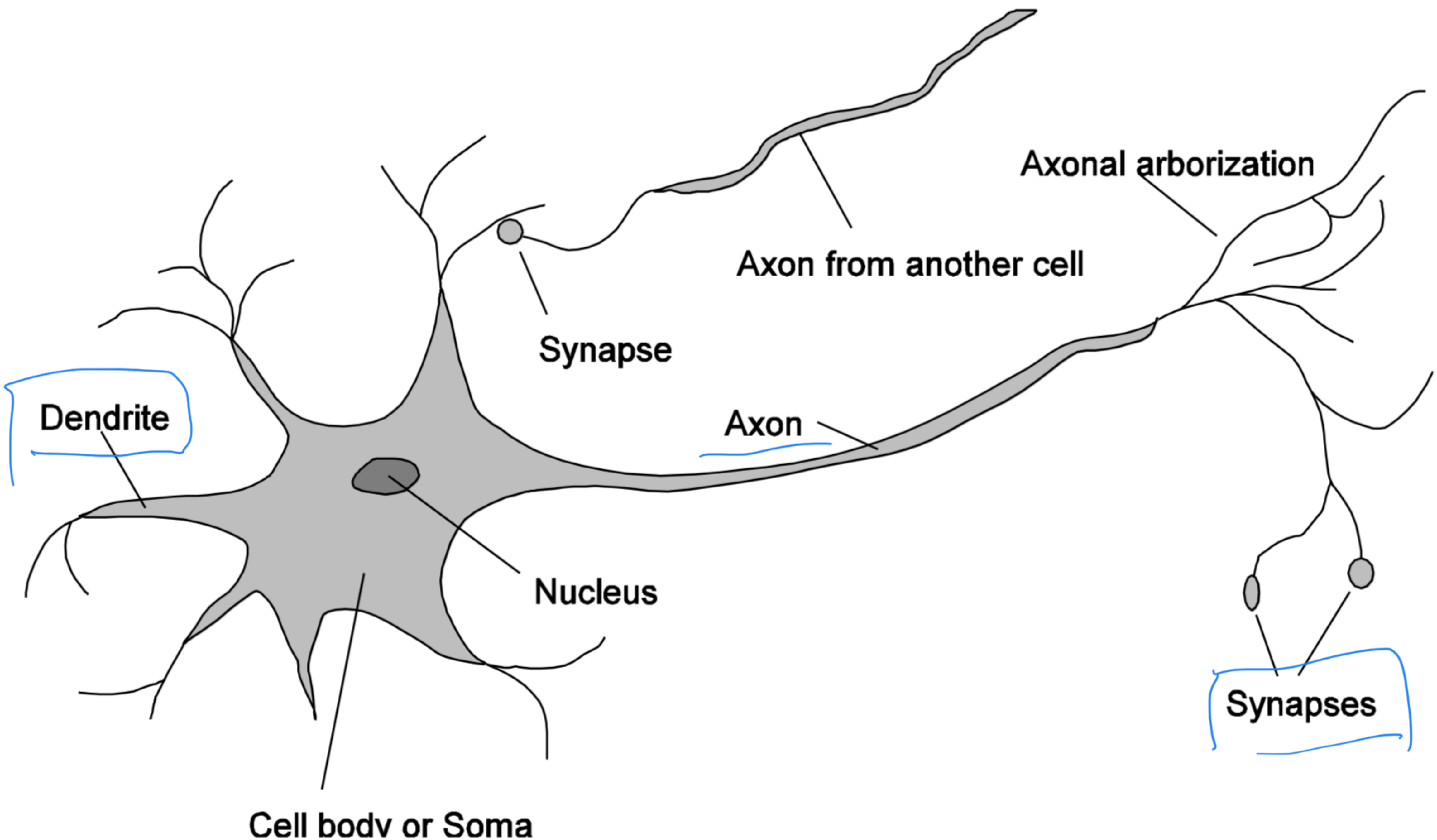
Acknowledgement:
Many of these slides are derived from Tom Mitchell, Pascal Poupart, Pieter Abbeel, Eric Eaton, Carlos Guestrin, William Cohen, and Andrew Moore.

Human Intelligence

- Brian is responsible for human intelligence by performing
  - Learning
  - Memorization
  - Cognition and recognition
  - Decision making

- Brian consists of nerve cells called neurons
  - Neurons can propagate nervous signal
  - Neurons form giant network of signal propagation

Neuron


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Comparison

- Brain
  - Network of neurons
  - Nerve signals propagate via neural network
  - Parallel computation
  - Robust (neurons die everyday without any impact)

- Computer
  - Bunch of gates
  - Electrical signals directed by gates
  - Sequential and parallel computation
  - Fragile (if a gate stops working, computer crashes)

Artificial Neural Networks

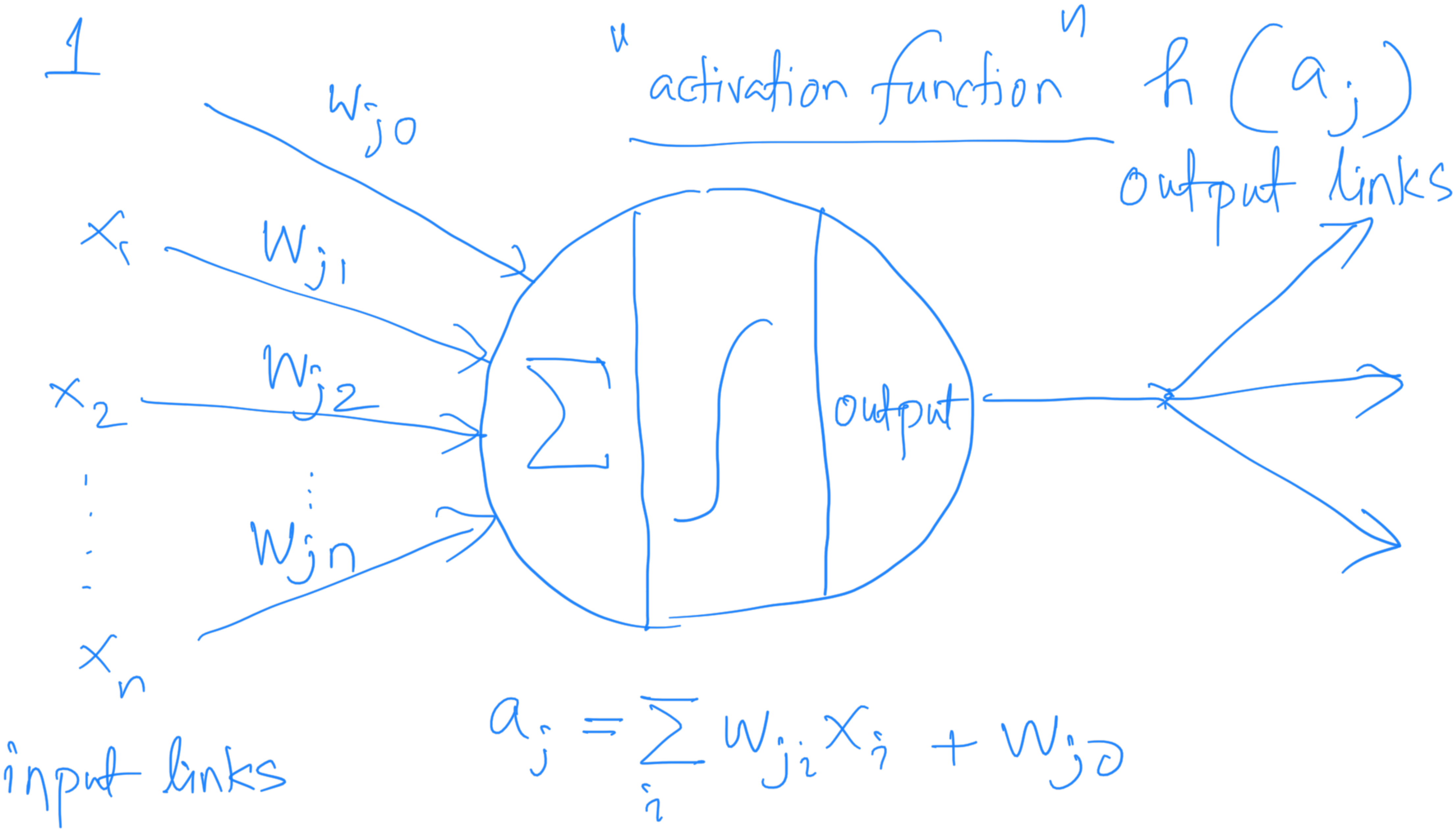
- **Key idea: emulate biological neurons for computation**

- Artificial neural network (ANN)
  - Units are called "**nodes**" and correspond to neurons
  - Connections between nodes correspond to synapses

- Correspondence between ANN and biological neural network
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal correspond to neurons firing rate

ANN: Node

- Schematic



$$a_j = \sum_i w_{ji} x_i + w_{j0}$$

ANN

- **Node:** $\boldsymbol{i}$
- **Weights:** $\boldsymbol{W}$
  - Strength of the connection from node i to node j
  - Input signals x_i weighted by W_{ji} and linearly combined:
    - $\boxed{a_j = \sum_i W_{ji} x_i + w_0 = \boldsymbol{W_{ji} x}}$ W_j
- **Activation function:** $\boldsymbol{h}$
  - Numerical signal produced: $y_j = h(a_j)$
- **Nodes are interconnected to form a network**

ANN: Activation Function

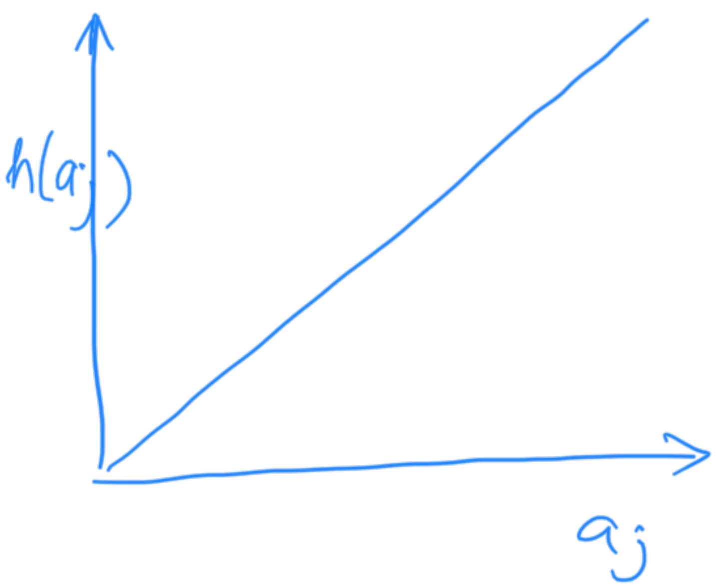
- Genrerally non-linear
  - Else, the network is just a linear function

- Mimics the firing in neurons
  - Nodes should be “active” (output close to 1) when fed with the “right” inputs
  - Nodes should be “inactive” (output close to 0) when fed with the “wrong” inputs

Common Activation Functions

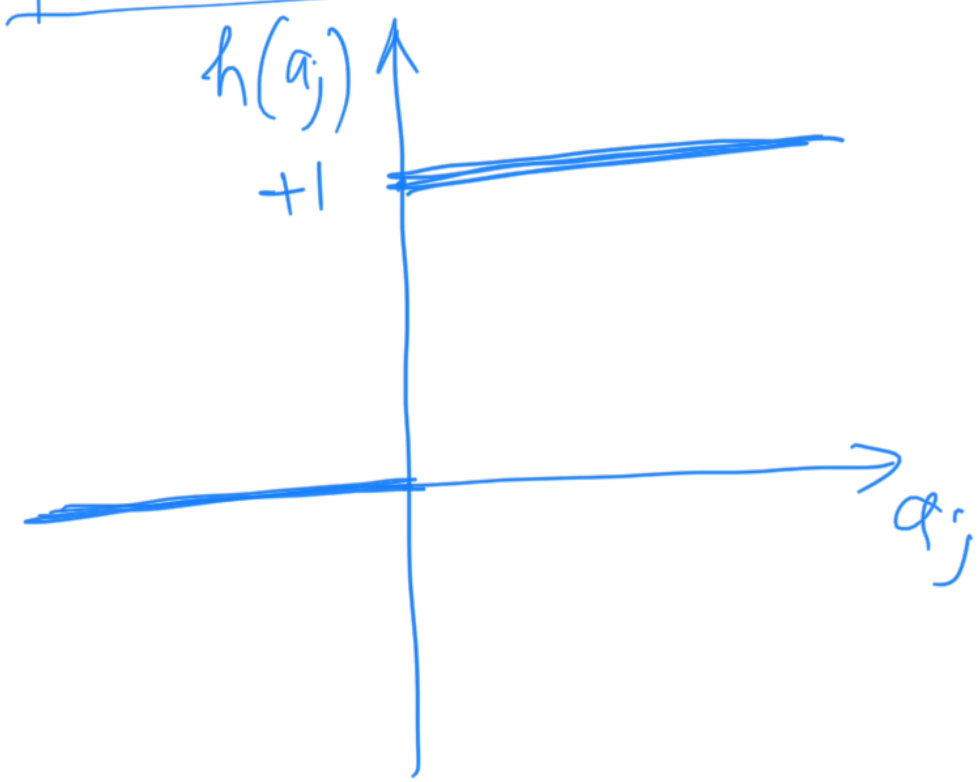
Identity

$$h(a) = a$$



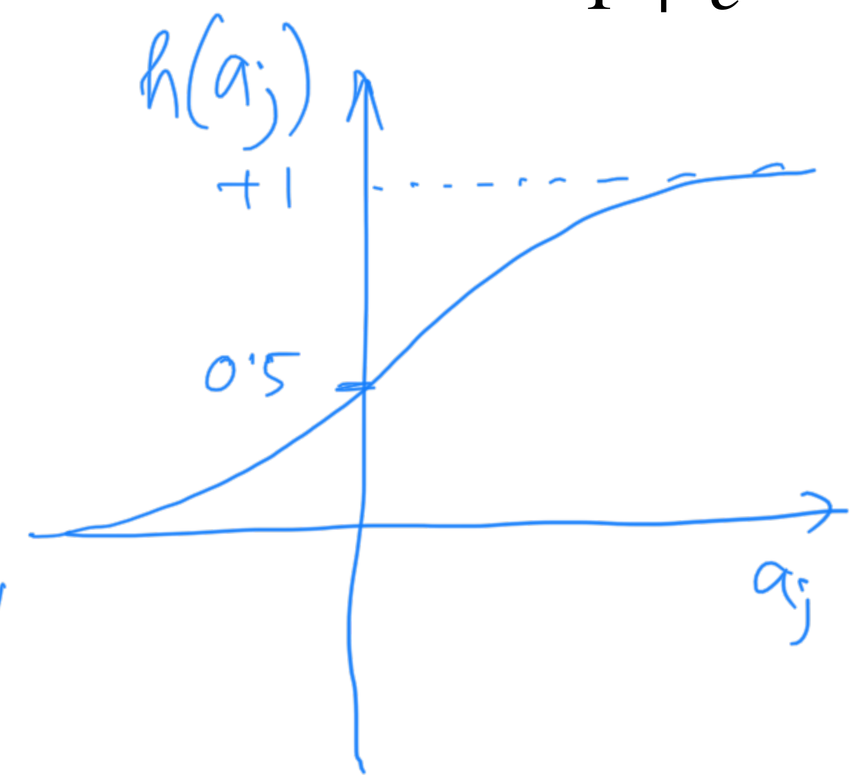
Threshold

$$h(a) = \begin{cases} 1 & \textit{if } a \geq 0 \\ 0 & \textit{if } a < 0 \end{cases}$$



Sigmoid

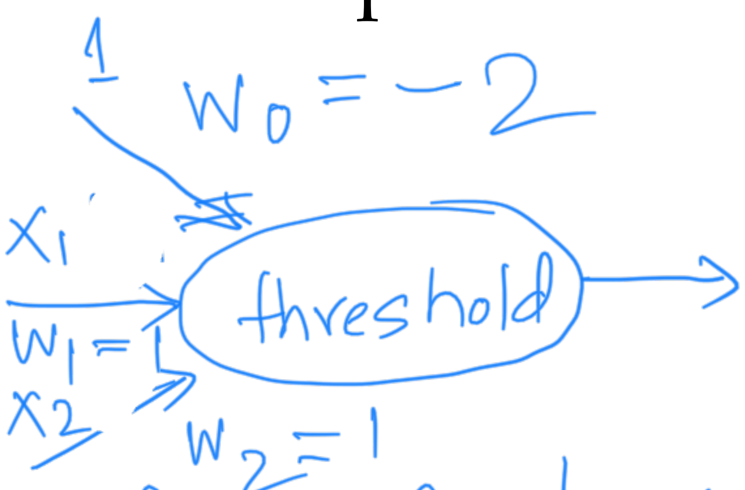
$$h(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$



Representing Boolean Functions

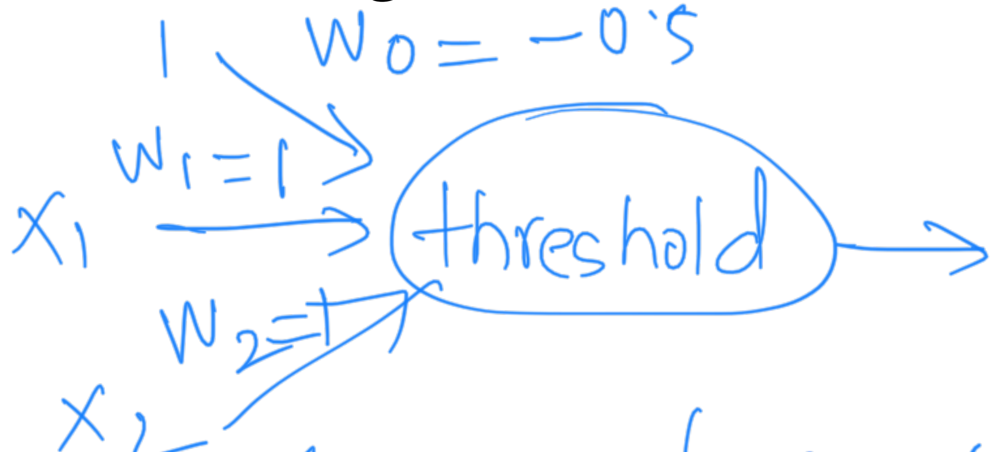
- Design ANN for logic gates

- What should be the weights of the following unites to represent AND, OR, NOT gates?



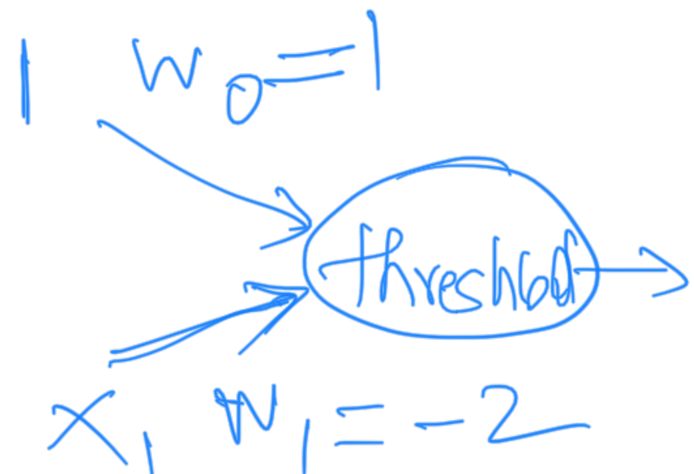
x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

AND



x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

OR



x_1	$\neg x_1$
$\boxed{0}$	1
1	0

NOT

Representing Boolean Functions

- ANN can be used to design various logic gates
- So ANN can be used to approximate any boolean functions

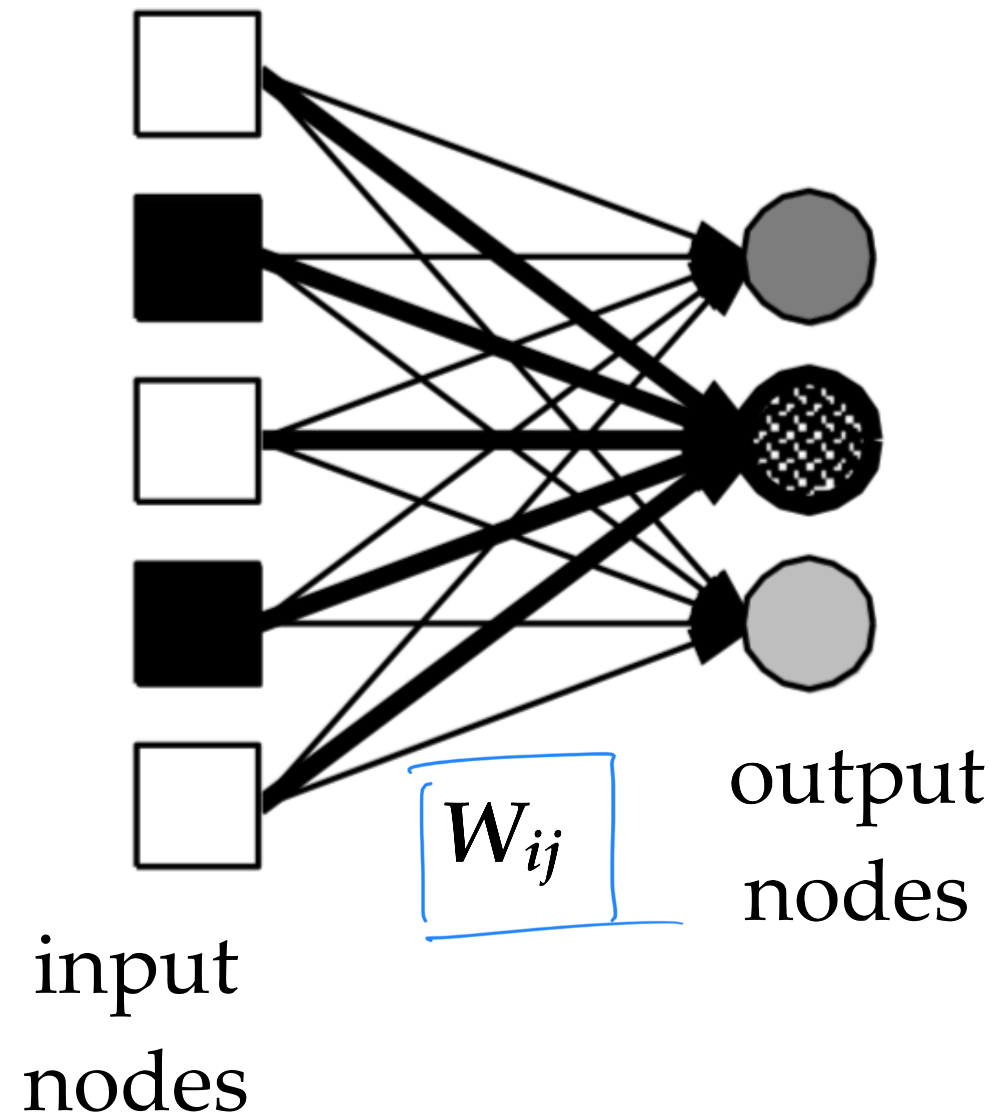
Network Architecture

- **Feed-forward Network**
  - Directed **acyclic** graph
  - No internal state

- **Recurrent Network**
  - Directed **cyclic** graph
  - Dynamical system with internal states
  - Can memorize information

Perceptron

- Single layer feed-forward network



Threshold Perceptron

- Given list if $(\boldsymbol{x}, \boldsymbol{y})$ pairs
- Train feed-forward ANN
  - Compute correct outputs $\boldsymbol{y}$ when fed with inputs $\boldsymbol{x}$
  - Accordingly adjust wights $\boldsymbol{W_{ji}}$
- **Leads to a simple algorithm for threshold perceptrons**

Threshold Perceptron Learning

- Learning is done separately for each output node j
  - Since nodes do not share weights

- Perceptron learning for node j
  - For each $(\boldsymbol{x}, \boldsymbol{y})$ pairs do:
    - Case 1: correct output produced

$$\forall_i \boxed{W_{ji} \leftarrow W_{ji}}$$

    - Case 2: output produced 0 instead of 1

$$\forall_i W_{ji} \leftarrow W_{ji} + x_i$$

    - Case 3: output produced 1 instead of 0

$$\forall_i W_{ji} \leftarrow W_{ji} - x_i$$

- Until correct output for all training instances

Threshold Perceptron Learning

- Dot products $\boldsymbol{x}^T\boldsymbol{x} \geq 0$ and $-\boldsymbol{x}^T\boldsymbol{x} \leq 0$
- Perceptron computes
  - 1 when $\boldsymbol{w}^T\boldsymbol{x} = \sum_i x_i w_i + w_0 > 0$
  - 0 when $\boldsymbol{w}^T\boldsymbol{x} = \sum_i x_i w_i + w_0 < 0$
- If output should be 1 instead of 0
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}$ since $(\boldsymbol{w} + \boldsymbol{x})^T\boldsymbol{x} \geq \boldsymbol{w}^T\boldsymbol{x}$
- If output should be 1 instead of 0
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{x}$ since $(\boldsymbol{w} - \boldsymbol{x})^T\boldsymbol{x} \leq \boldsymbol{w}^T\boldsymbol{x}$

Alternative Approach

- Let $\boxed{y \in \{-1, 1\}}\ \forall y$
- Let $\boxed{M = \{\{x_n\ ,\ y_n\} \forall n\}}$ be set of misclassified examples
  - i.e. $\boxed{y_n\ \boldsymbol{w}^T\boldsymbol{x}} < 0$
- Find $\boldsymbol{w}$ that minimizes misclassification error:
  - $\mathrm{E}(\boldsymbol{w}) = -\sum_{(x_n,\ y_n) \in M} \boxed{y_n\ \boldsymbol{w}^T\boldsymbol{x}}$
- Apply gradient descent algorithm
  - $\boxed{\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{\eta} \nabla \boldsymbol{E}}$

learning rate or step size

Sequential Gradient Descent

- Gradient $\nabla \boldsymbol{E} = -\sum_{(x_n,\, y_n) \in M} y_n \boldsymbol{x_n}$

- Sequencial gradient descent
  - Adjust w based on one example (x, y) at a time
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\eta}\, y\, \boldsymbol{x}$

−1 +1

- When $\boldsymbol{\eta} = 1$, we recover the threshold perceptron algorithm

Threshold Perceptron Algorithm

- Let $y \in \{-1, 1\}\ \forall y$
- Start with randomly initialized weights: w
- For t = 1..T (T passes over data)
  - For l =1..n: (each training example)
    - Classify with current weights
      - $\hat{y} = sign\,(\boldsymbol{w}^T\boldsymbol{x})$ where $sign(x) = +1$ if $x > 0$ else -1
- If correct (i.e., $\hat{y} = y^l$), no change!)
- If wrong: update:
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + y^l\,\boldsymbol{x}^l$

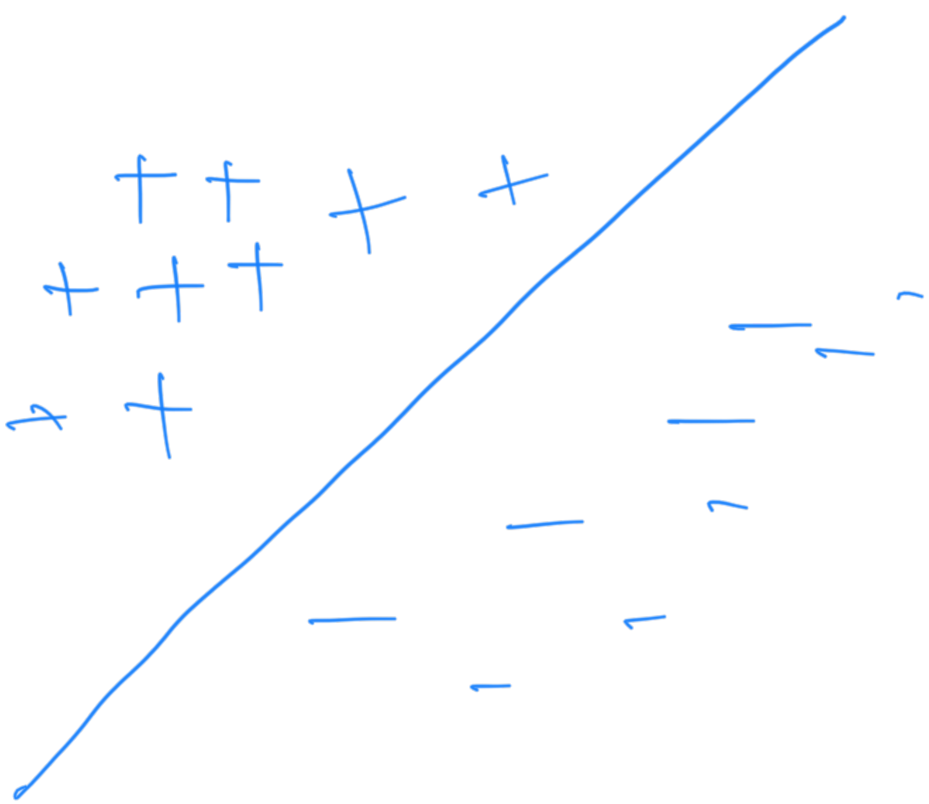
Properties of Threshold Perceptron

- Hypothesis space $h_{\boldsymbol{w}}$
- Binary classifications with parameters $\boldsymbol{w}$
- Since $\boldsymbol{w}^T\boldsymbol{x}$ is linear in $\boldsymbol{w}$, perceptron is a **linear separator**
- Converges *iff* the data is linearly separable

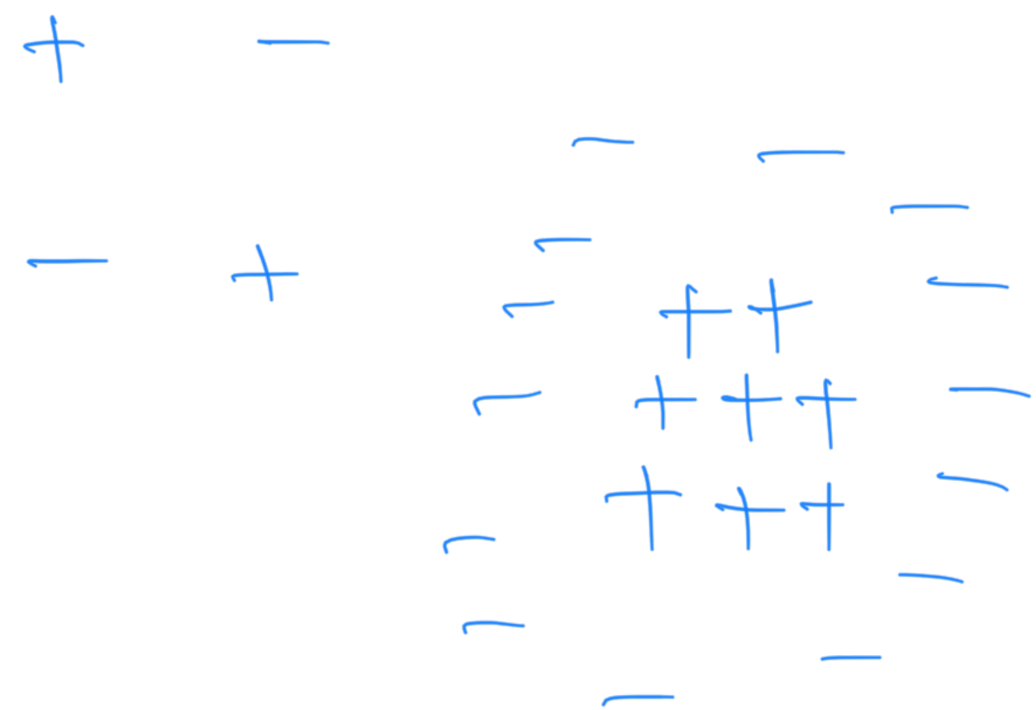
Perceptron Linear Separability

- Examples

Linearly Separable

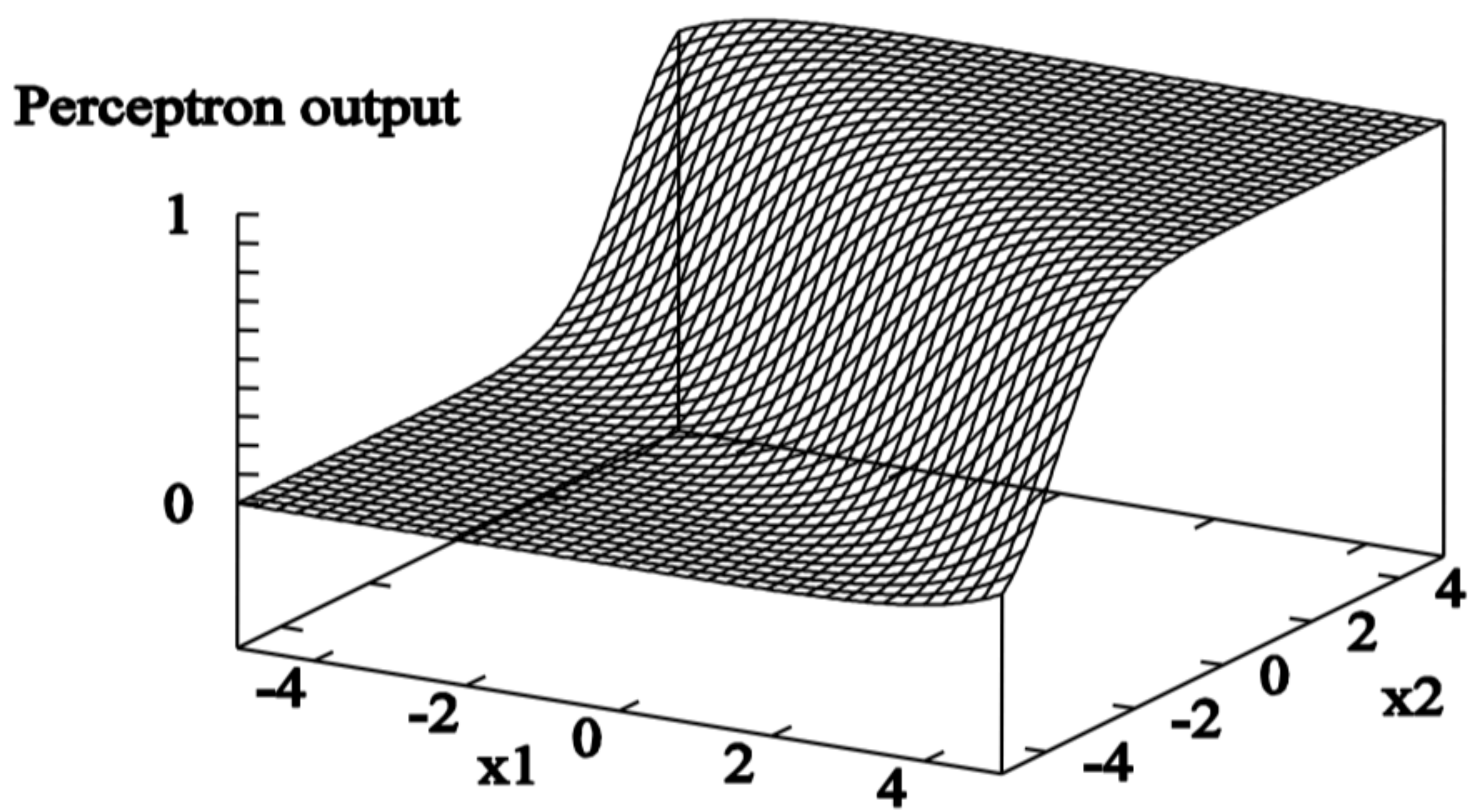


Linearly Nonseparable



Sigmoid Perceptron

- "Soft" linear separator



Can we use sigmoid perceptron for linearly nonseparable data points?

Sigmoid Perceptron Learning

- **Maximum likelihood estimation**
  - Equivalent to logistic regression
- Objective function can be:
  - **Mimimim squared error**

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n} E_n(\mathbf{w})^2 = \frac{1}{2}\sum_{n}(y_n - \sigma(\mathbf{w}^{\mathbf{T}}\mathbf{x_n}))^2$$

Gradient

- Derivation

$$\frac{\partial E}{\partial w_i} = \sum_n E_n \frac{\partial E_n}{\partial w_i}$$

$$= \sum_n E_n(w)\sigma' \, (\mathbf{w^T x_n})\mathbf{x}_i$$

$$= \sum_n E_n(w)\sigma(\mathbf{w^T x_n})(1 - \sigma(\mathbf{w^T x_n}))\mathbf{x}_i$$

Recall,

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

$$\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

No closed form solution!

Gradient Descent

- Perceptron-Learning(examples, network)
  - Repeat
  - For each ($\boldsymbol{x_n}$, y_n) in examples, do:
    - $E_n \leftarrow y_n - \sigma\,(\boldsymbol{w}^T\boldsymbol{x})$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta E_n\, \sigma\,(\boldsymbol{w}^T\boldsymbol{x})\,(1 - \sigma\,(\boldsymbol{w}^T\boldsymbol{x}))\, \boldsymbol{x}_n$
  - Until some stopping criteria satisfied
  - Return learnt network

Demo Time 🙂

https://playground.tensorflow.org/