Machine Learning Part II

Nov 5:	**Machine Learning: Perceptron**	slides
Nov 10:	**Machine Learning: Neural Network 1** HW7 DUE (TENTATIVE) HW8 RELEASED (REGRESSION)	slides
Nov 12:	**Machine Learning: Neural Network 2**	slides
Nov 17:	**Machine Learning: Deep Learning 1** (TENTATIVE) HW8 DUE (TENTATIVE) HW9 RELEASED (NEURAL NETWORK)	slides,
Nov 19:	**Machine Learning: Deep Learning 2**	slides
Nov 24:	**Machine Learning: Deep Learning 3** (TENTATIVE) HW9 DUE	slides
Nov 26:	**Happy Thanksgiving!**	
Dec 1:	**Machine Learning: Reinforcement Learning** (TENTATIVE) HW10 RELEASED (DEEP LEARNING)	slides
Dec 3:	**Machine Learning: Part II Summary**	slides

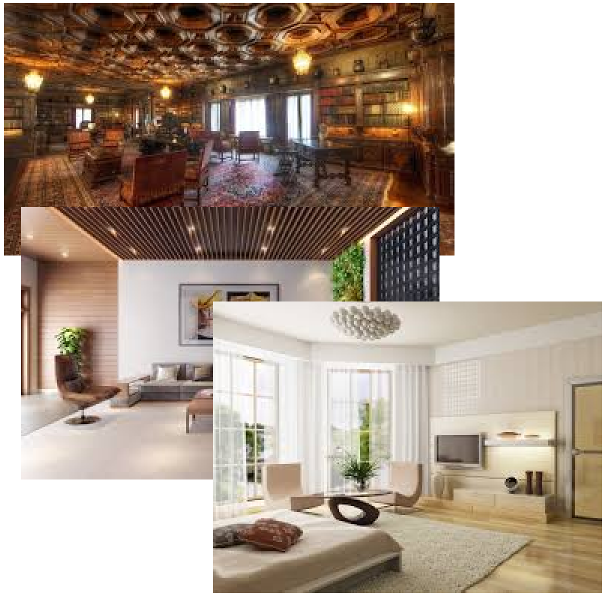
Neural Networks
Part 0: Perceptron

Yin Li
yin.li@wisc.edu
University of Wisconsin, Madison

[Based on slides from Yingyu Liang, Jerry Zhu]

Motivation I: learning features

- Example task



Experience/Data:
images with labels

Indoor

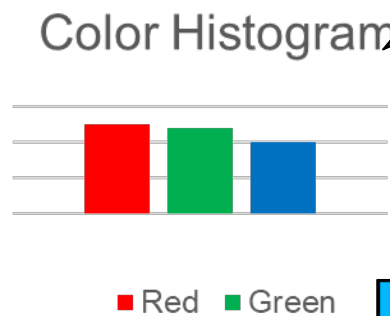
outdoor

Motivation I: learning features

- Featured designed for the example task

Feature x_i

Extract features



Label y_i

Indoor

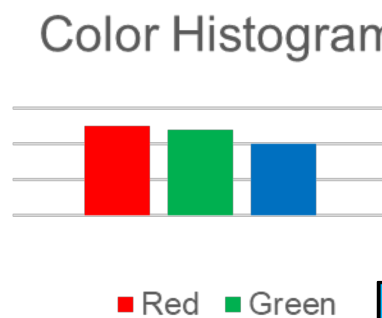
0

Motivation I: learning features

- Featured designed for the example task
- Supervised learning for a decision function $y = f(x)$

Feature x_i

Extract features

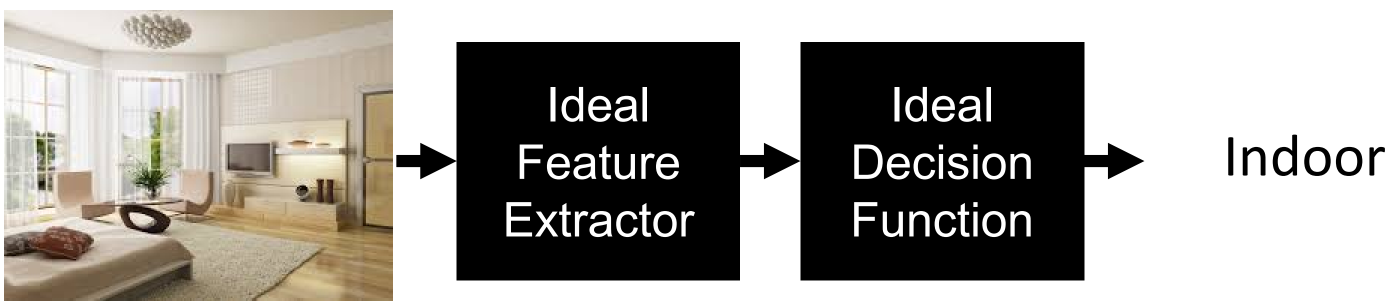


Label y_i

Indoor → 0

Motivation I: learning features

- What types of features shall we consider?
  - Color histogram, bag of words, …
- What type of decision functions shall we consider?
  - Naïve Bayes, linear models, KNN, …

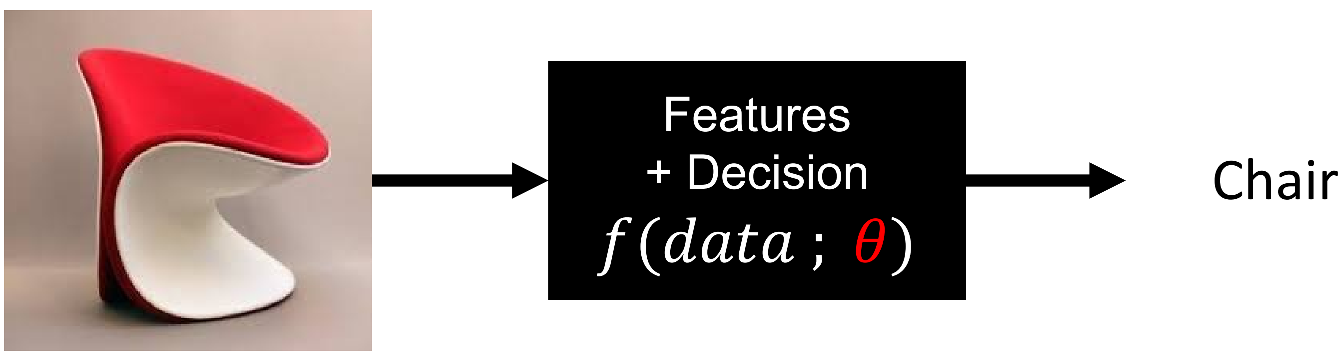


Motivation I: learning features

- More complicated tasks: hard to design
- Would like to learn features

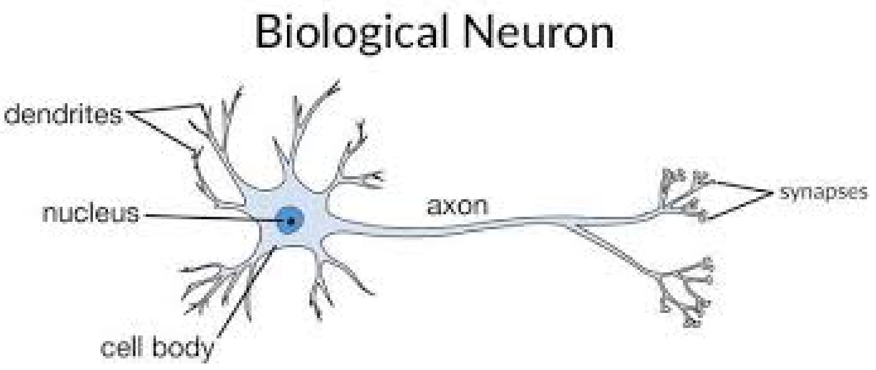
Motivation I: learning features

- What if we only learn a single function?
  - Directly mapping from the data to the label
  - Combining features and the decision function
  - Defined by its parameters θ
- What type of functions shall we consider for *f*?



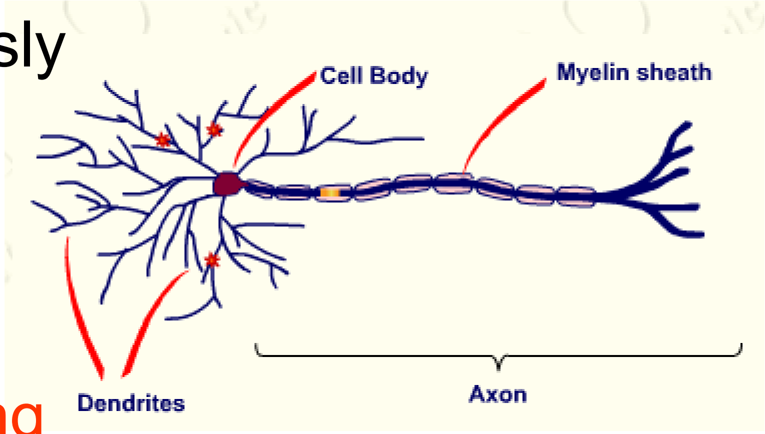
Motivation II: neuroscience

- Inspirations from human brains
- Networks of simple and homogenous units



Motivation II: neuroscience

- Human brain: 100, 000, 000, 000 neurons
- Each neuron receives input from 1,000 others
- Impulses arrive simultaneously
- Added together*
  - an impulse can either increase or decrease the possibility of nerve pulse firing

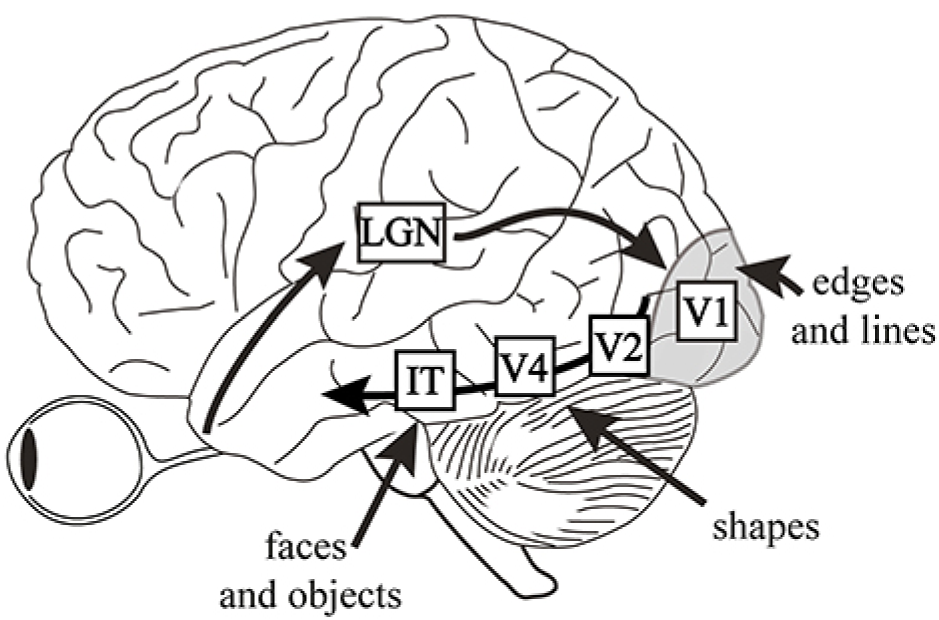


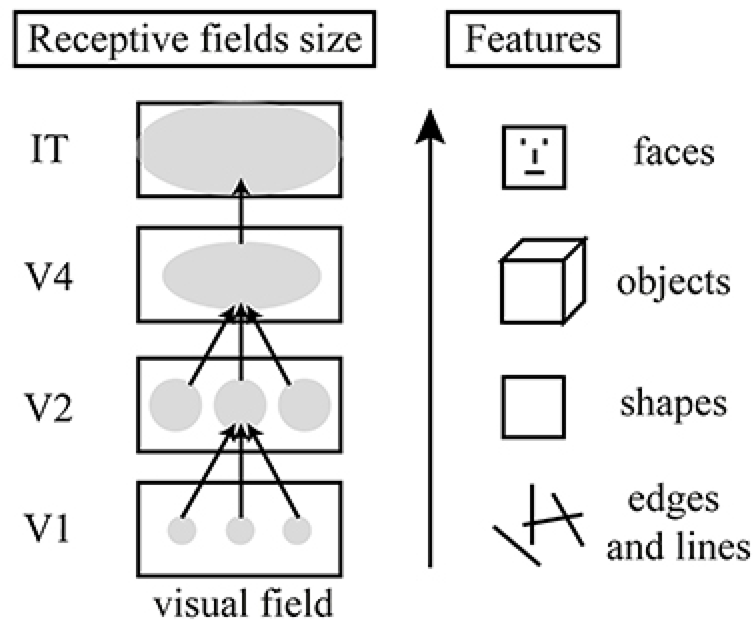
- If sufficiently strong, a nerve pulse is generated
- The pulse forms the input to other neurons.
- The interface of two neurons is called a synapse

http://www.bris.ac.uk/synaptic/public/brainbasic.html

Motivation II: neuroscience

- Hierarchical information processing
- Visual cortex has many areas that forms a hierarchy
- Outputs of one area as inputs of another one
- Gradually build up more complex concepts





https://www.frontiersin.org/articles/10.3389/fncom.2014.00135/full

Neural Networks / Deep Learning

- What type of functions shall we consider for *f?*

Features
+ Decision
$f(data\ ;\ \theta)$

Chair

Proposal: Composing a set of (nonlinear) functions *g*

$$f(data;\theta) = g_1(\ldots g_{n-1}(g_n(data;\theta_n),\theta_{n-1})\ldots,\theta_1)$$

Successful applications

- Computer vision: Image Generation

Slides from Kaimin He, MSRA

Successful applications

- NLP: Question & Answer

I:	Jane went to the hallway.
I:	Mary walked to the bathroom.
I:	Sandra went to the garden.
I:	Daniel went back to the garden.
I:	Sandra took the milk there.
Q:	Where is the milk?
A:	garden

Figures from the paper "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing ",
by Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Richard Socher

Successful applications

- Game: AlphaGo

Outline

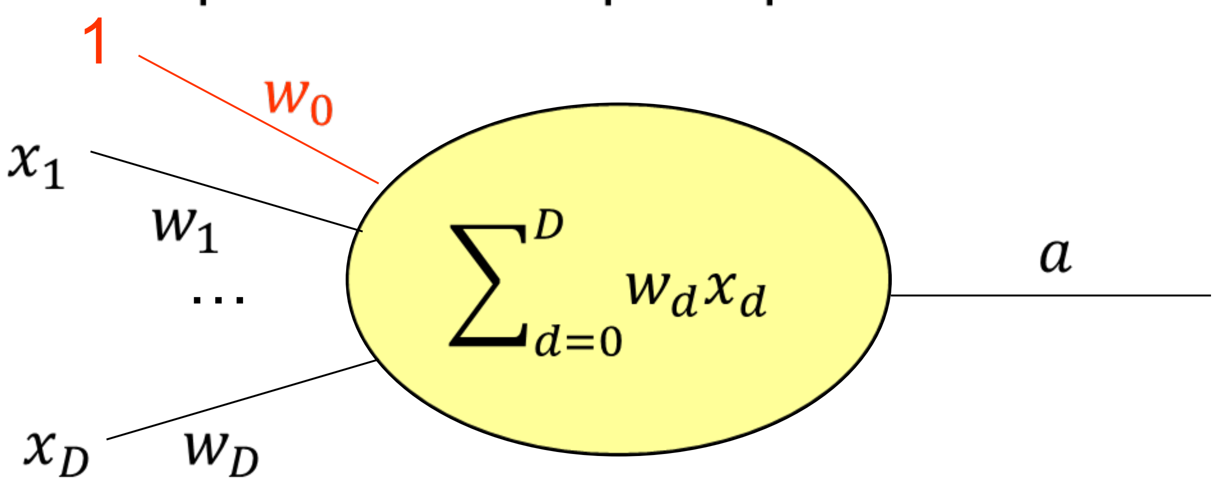
- A single neuron
  - Linear perceptron
  - Non-linear perceptron
  - Learning of a single perceptron
  - The power of a single perceptron
- Neural network: a network of neurons
  - Layers, hidden units
  - Learning of neural network: backpropagation
  - The power of neural network
  - Issues
- Deep learning: deep neural networks
- Everything revolves around gradient descent

Linear perceptron

- Perceptron = a math model for a single neuron
- Input: $x_1, \ldots, x_D$ (signal from other neurons)
- Weights: $w_1, \ldots, w_D$ (dendrites, can be negative)
- We sneak in a constant (bias term) $x_0 = 1$, with some weight w_0
- Activation function: linear (for the time being)

$$a = w_0 + w_1 * x_1 + \ldots + w_D * x_D$$

- This is the output of a linear perceptron



Learning in linear perceptron

- Training data $\{(X_1, y_1), \ldots, (X_N, y_N)\}$
- X_1 is a vector: $(x_{11}, \ldots, x_{1D})$, so are $X_2 \ldots X_N$
- y_1 is a real-valued output, so are $y_2 \ldots y_N$

- Goal: learn the weights $w_0, \ldots, w_D$, so that given input X_i, the output of the perceptron a_i is close to y_i
- Define "close":

$$E = ½ \sum_i (a_i - y_i)^2$$

- E is the "error". Given the training set, it is a function of $w_0, \ldots, w_D$.
- Minimize E: unconstrained optimization with variables $w_0, \ldots, w_D$. Exactly linear regression.

Learning in linear perceptron

- Gradient descent: $W \leftarrow W - \alpha \nabla E(W)$
- α is a small constant, "learning rate" = step size
- The gradient descent rule:

$$E(W) = ½ \sum_i (a_i - y_i)^2$$

$$\partial E / \partial w_d = \sum_i (a_i - y_i) \partial a_i / \partial w_d$$

Learning in linear perceptron

- Gradient descent: $W \leftarrow W - \alpha \nabla E(W)$
- α is a small constant, "learning rate" = step size
- The gradient descent rule:

$$E(W) = ½ \sum_i (a_i - y_i)^2$$

$$\partial E/\partial w_d = \sum_i (a_i - y_i)\partial a_i/\partial w_d$$

with $a_i = w_0 + w_1 * x_{i1} + w_2 * x_{i2} + \cdots w_D x_{iD}$

$$\Rightarrow \frac{\partial E}{\partial w_d} = \sum_i (a_i - y_i)\, x_{id}$$

Learning in linear perceptron

- Gradient descent: $W \leftarrow W - \alpha \nabla E(W)$
- α is a small constant, "learning rate" = step size
- The gradient descent rule:

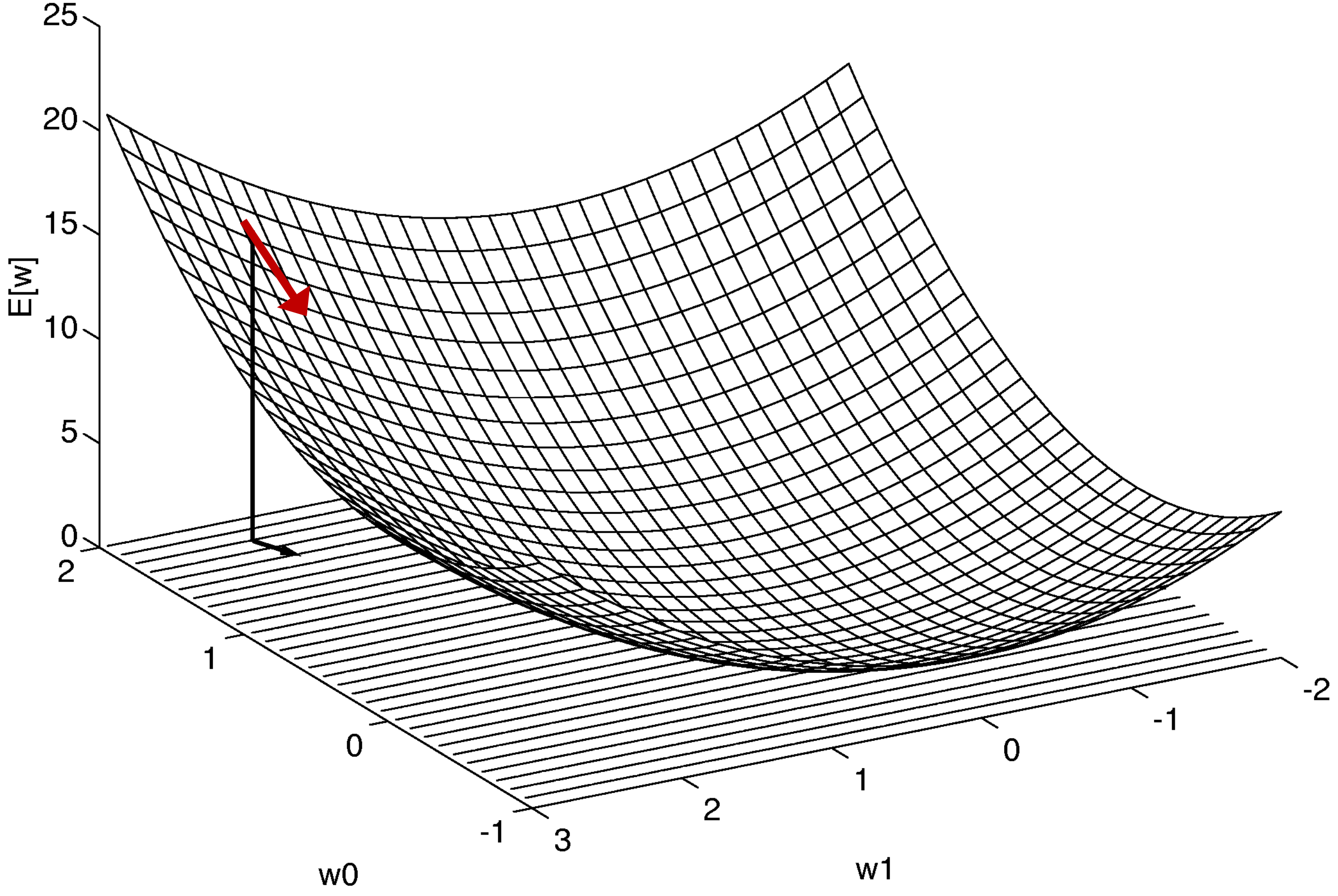
$$E(W) = ½ \sum_i (a_i - y_i)^2$$

$$\partial E / \partial w_d = \sum_i (a_i - y_i) x_{id}$$

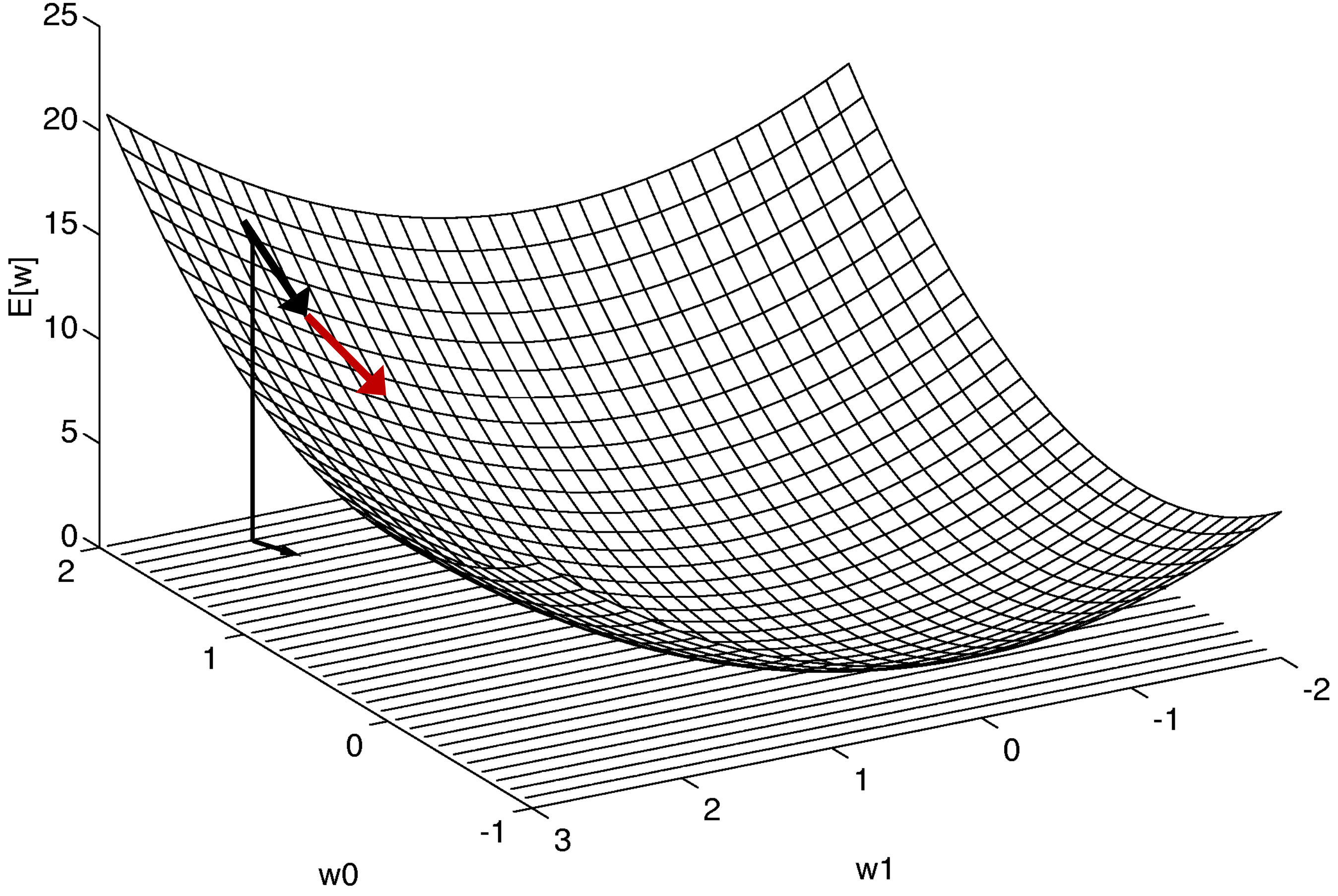
$$w_d \leftarrow w_d - \alpha \sum_i (a_i - y_i) x_{id}$$

- Repeat until E converges.
- E is convex in W: there is a unique global minimum

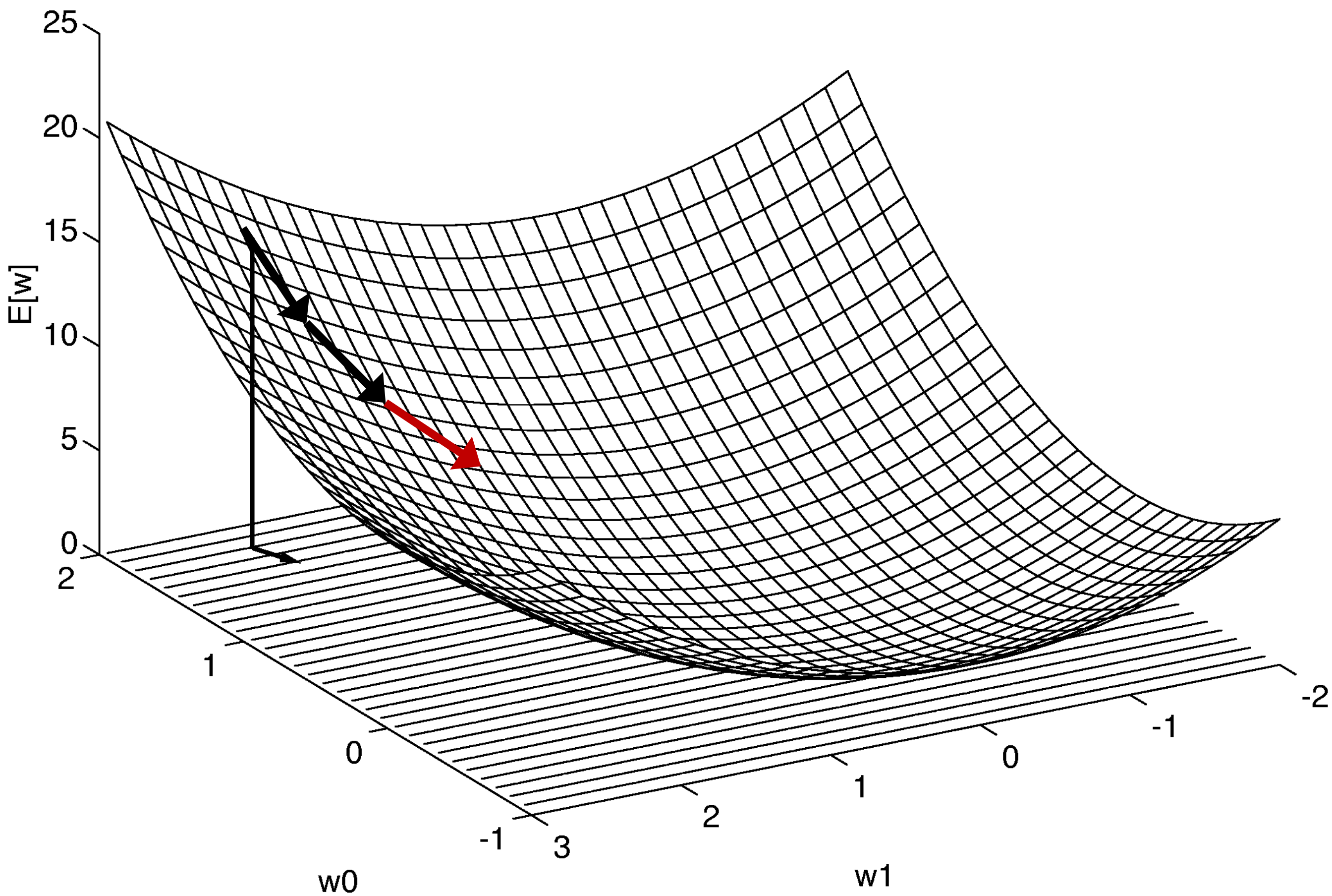
Visualization of gradient descent



Visualization of gradient descent

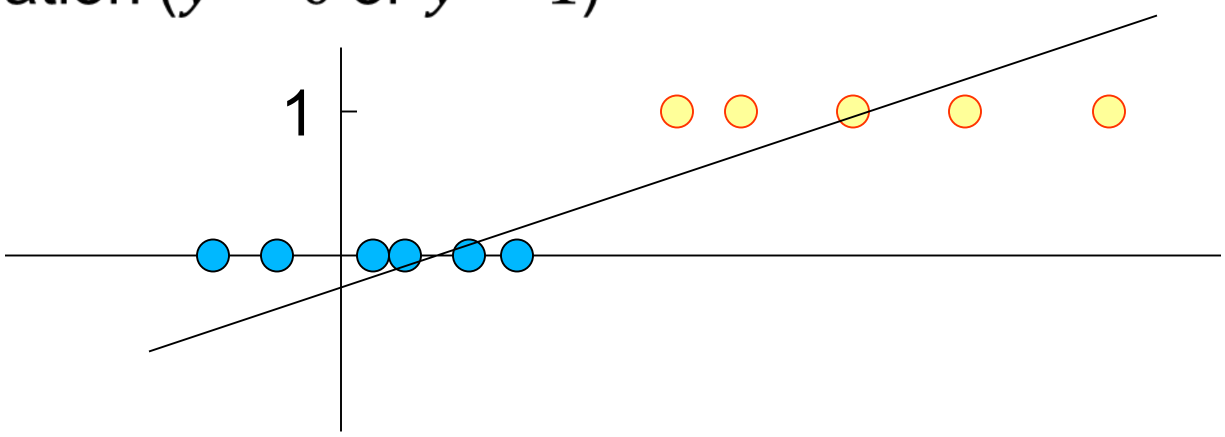


Visualization of gradient descent



The (limited) power of linear perceptron

- Linear perceptron is just
$$a = WX$$
- where X is the input vector, augmented by $x_0 = 1$
- It can represent any linear function in $D + 1$ dimensional space… but that's it
- In particular, it won't be a nice fit to binary classification ($y = 0$ or $y = 1$)

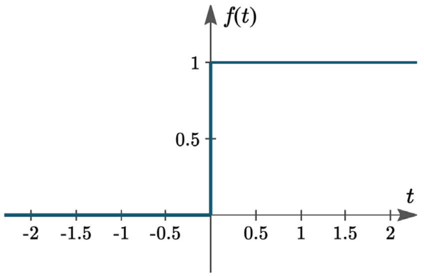


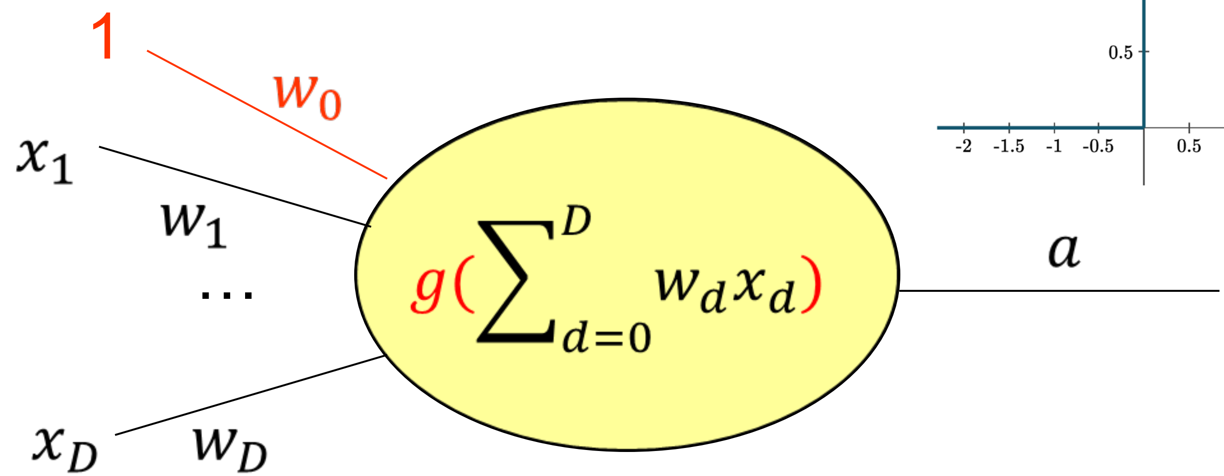
Non-linear perceptron

- Change the activation function: use a step function

$$a = g(w_0 + w_1 * x_1 + \dots + w_D * x_D)$$

- $g(h) = 0$, if $h < 0$; $g(h) = 1$ if $h \geq 0$





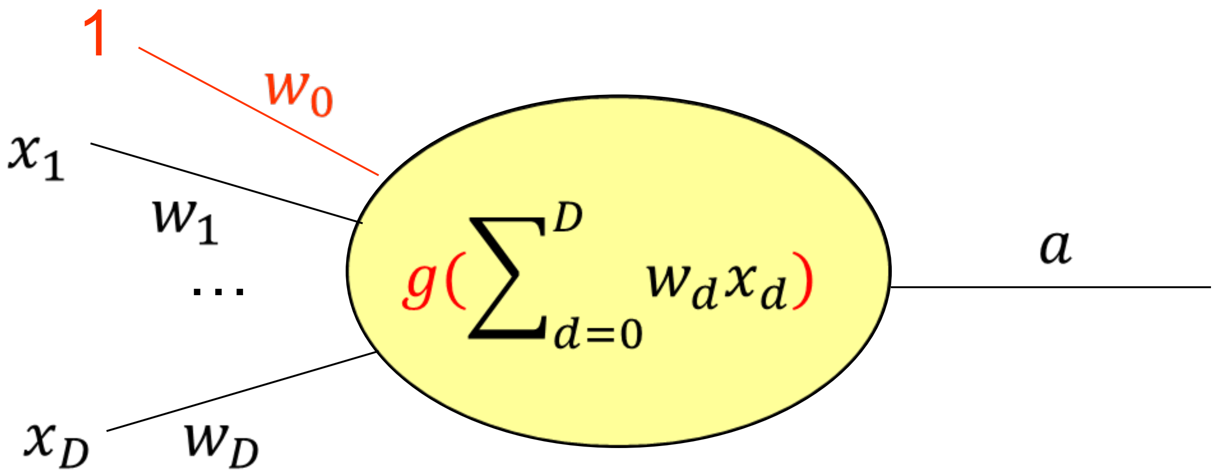
- Can you see how to make logic AND, OR, NOT with such a perceptron?

Non-linear perceptron

- Change the activation function: use a step function

$$a = g(w_0 + w_1 * x_1 + \ldots + w_D * x_D)$$

- $g(h) = 0$, if $h < 0$; $g(h) = 1$ if $h \geq 0$

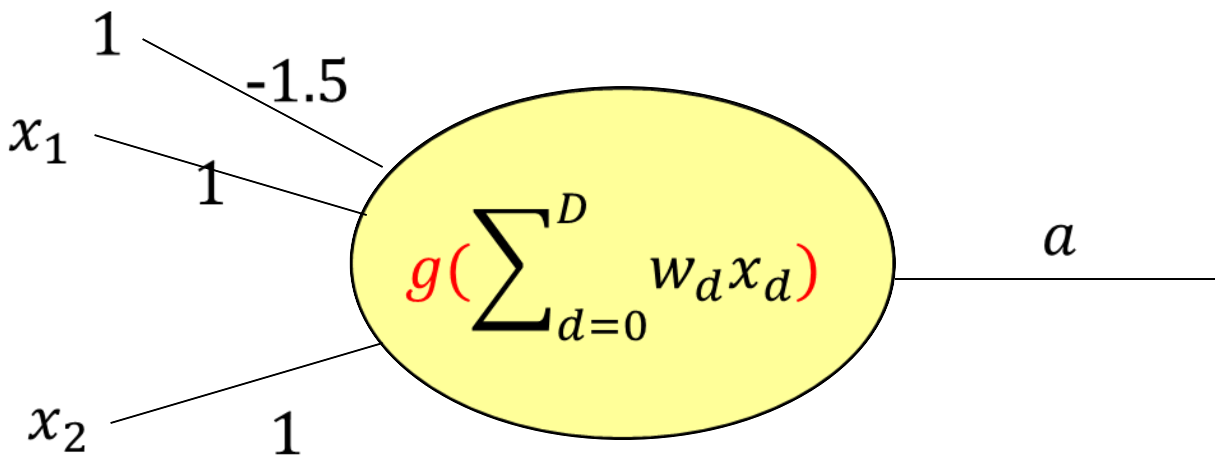


- AND: $w_1 = w_2 = 1, w_0 = -1.5$
- OR: $w_1 = w_2 = 1, w_0 = -0.5$
- NOT: $w_1 = -1, w_0 = 0.5$

Now we see the reason for bias terms

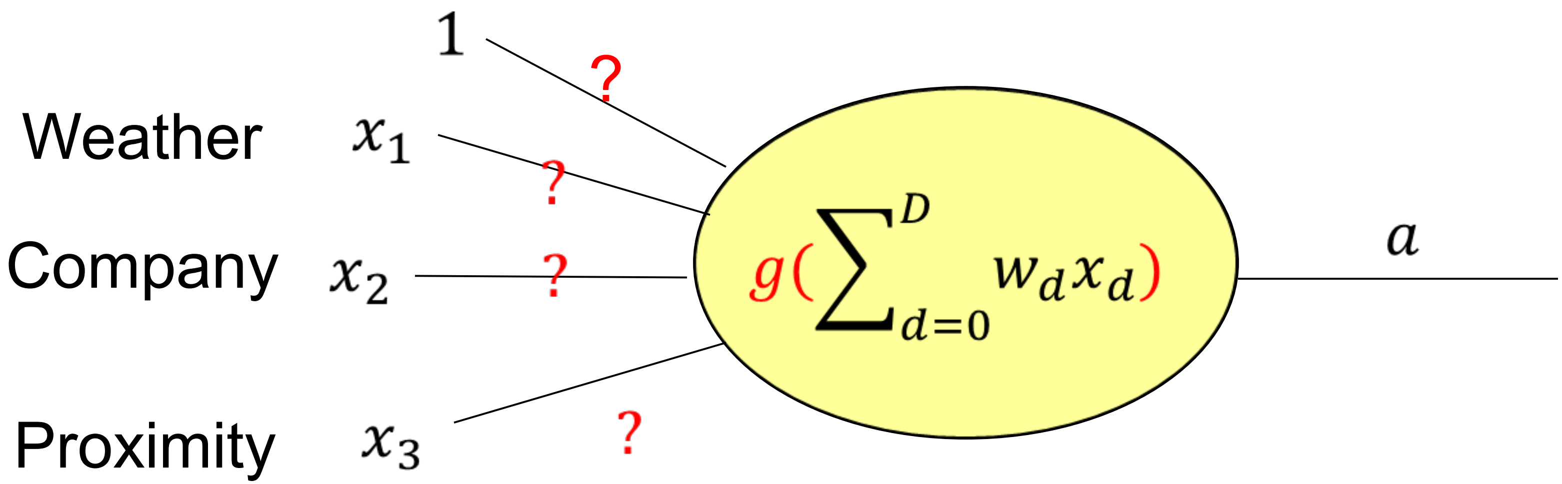
Non-linear perceptron for AND

- Change the activation function: use a step function
$$a = g(w_0 + w_1 * x_1 + \ldots + w_D * x_D)$$
- $g(h) = 0, \text{if } h < 0; \ g(h) = 1 \text{ if } h \geq 0$



Example Question

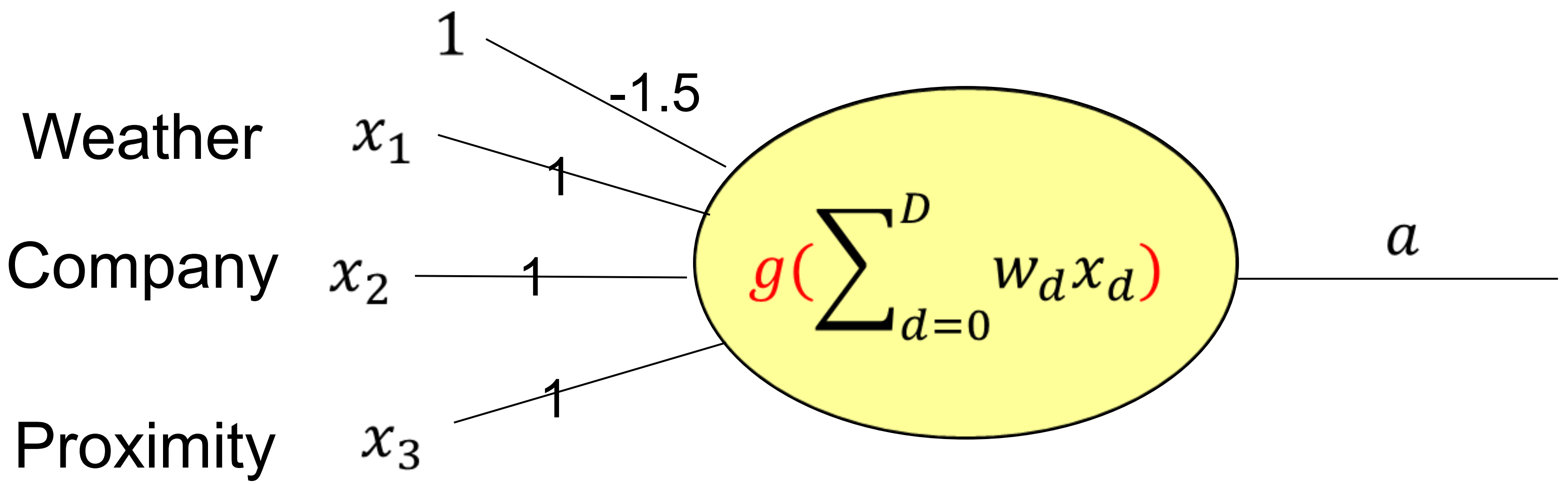
- Will you go to the festival?
- Go only if at least two conditions are favorable



All inputs are binary; 1 is favorable

Example Question

- Will you go to the festival?
- Go only if at least two conditions are favorable



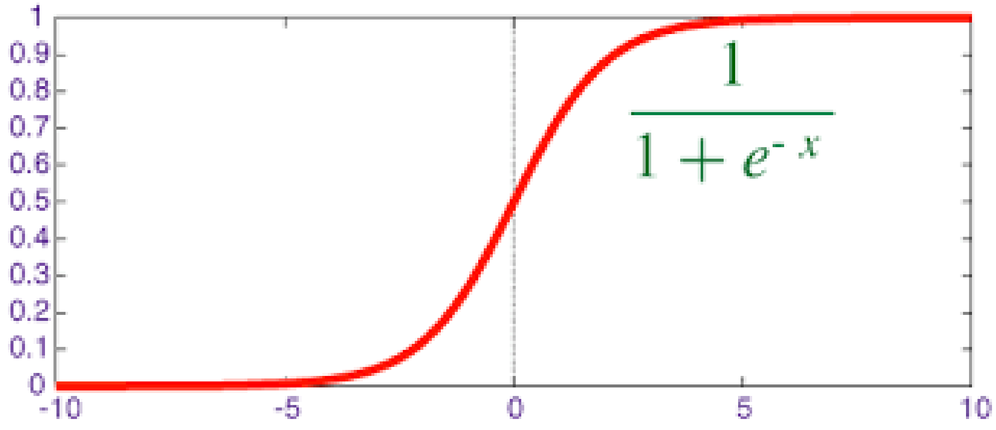
All inputs are binary; 1 is favorable

Sigmod activation function: Our second non-linear perceptron

- The problem with LTU: step function is discontinuous, cannot use gradient descent
- Change the activation function (again): use a sigmoid function

$$g(x) = 1 / (1 + \exp(-x))$$

also called a logistic function



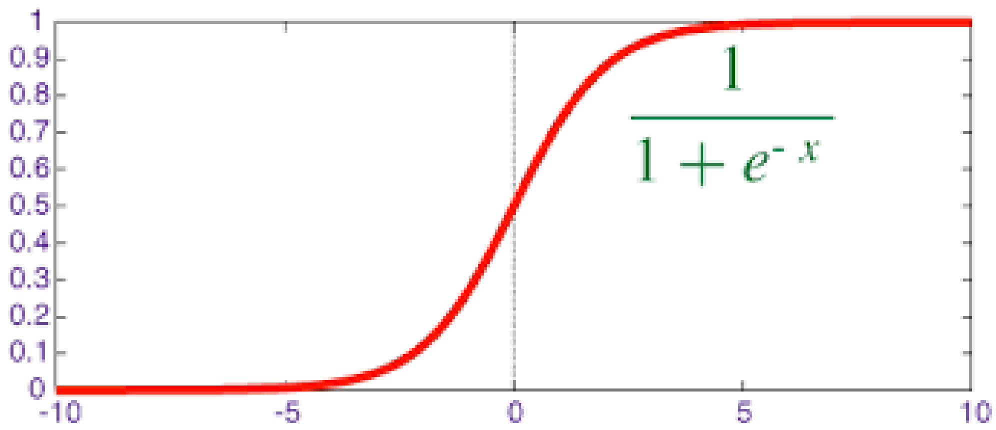
Sigmod activation function: Our second non-linear perceptron

- The problem with LTU: step function is discontinuous, cannot use gradient descent

- Change the activation function (again): use a sigmoid function

$$g(x) = 1 / (1 + \exp(-x))$$

- Exercise: $g'(x) = ?$



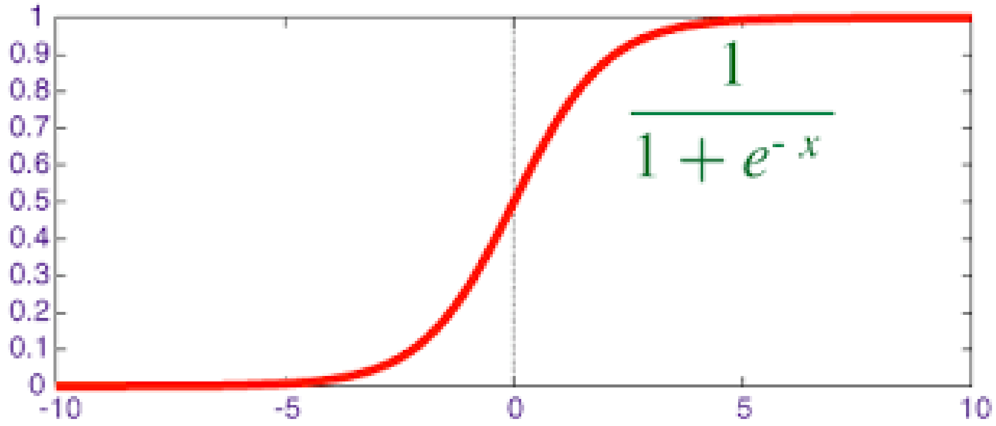
Sigmod activation function: Our second non-linear perceptron

- The problem with LTU: step function is discontinuous, cannot use gradient descent

- Change the activation function (again): use a sigmoid function

$$g(x) = 1 / (1 + \exp(-x))$$

- Exercise: $g'(x) = g(x)(1 - g(x))$



Learning in non-linear perceptron

- Again we will minimize the error:

$$E(W) = ½ \sum_i (a_i - y_i)^2$$

- Now $a_i = g(\Sigma_d w_d * x_{id})$

$$\partial E / \partial w_d = \sum_i (a_i - y_i) a_i (1 - a_i) x_{id}$$

- The sigmoid perceptron update rule

$$w_d \leftarrow w_d - \alpha \sum_i (a_i - y_i) a_i (1 - a_i) x_{id}$$

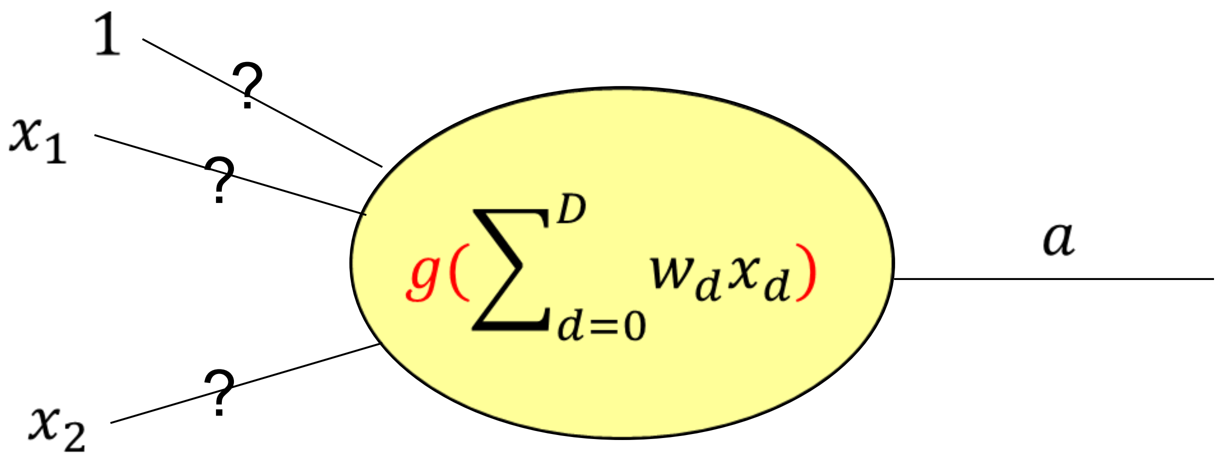
- α is a small constant, “learning rate” = step size
- Repeat until E converges

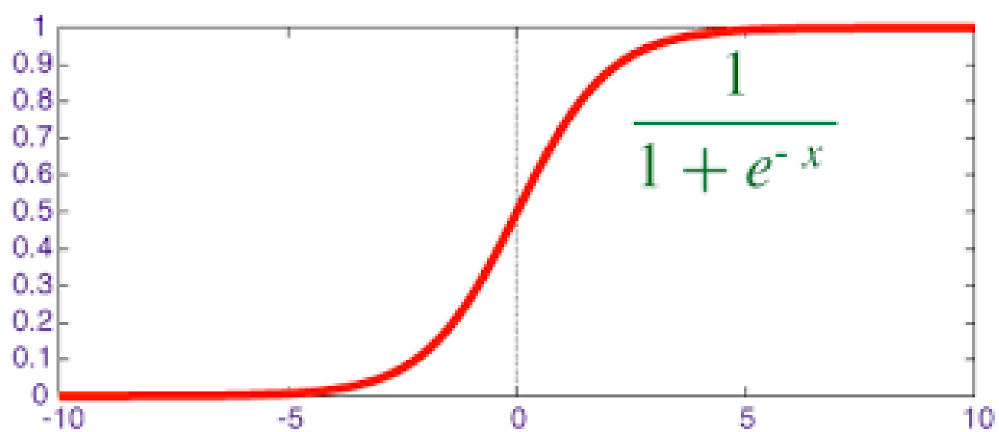
Non-linear perceptron for AND

- Change the activation function use a sigmoid function

$$A = g(w_0 + w_1 * x_1 + \cdots + w_D * x_D)$$

- $g(x) = 1/(1 + \exp(-x))$





The (limited) power of non-linear perceptron

- Even with a non-linear sigmoid function, the decision boundary a perceptron can produce is still linear
  - Think about logistic regression

- AND, OR, NOT revisited

- How about XOR?

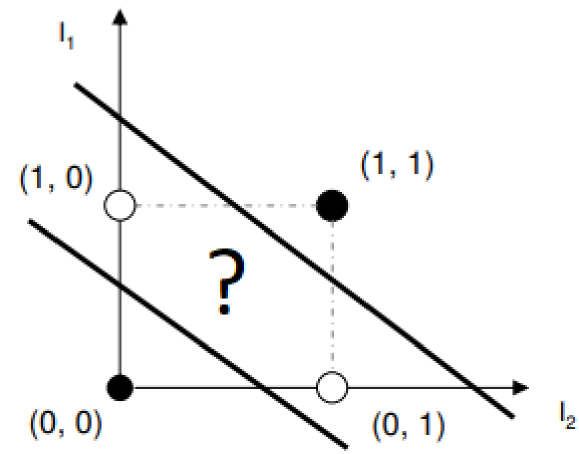
The (limited) power of non-linear perceptron

- Even with a non-linear sigmoid function, the decision boundary a perceptron can produce is still linear

- AND, OR, NOT revisited

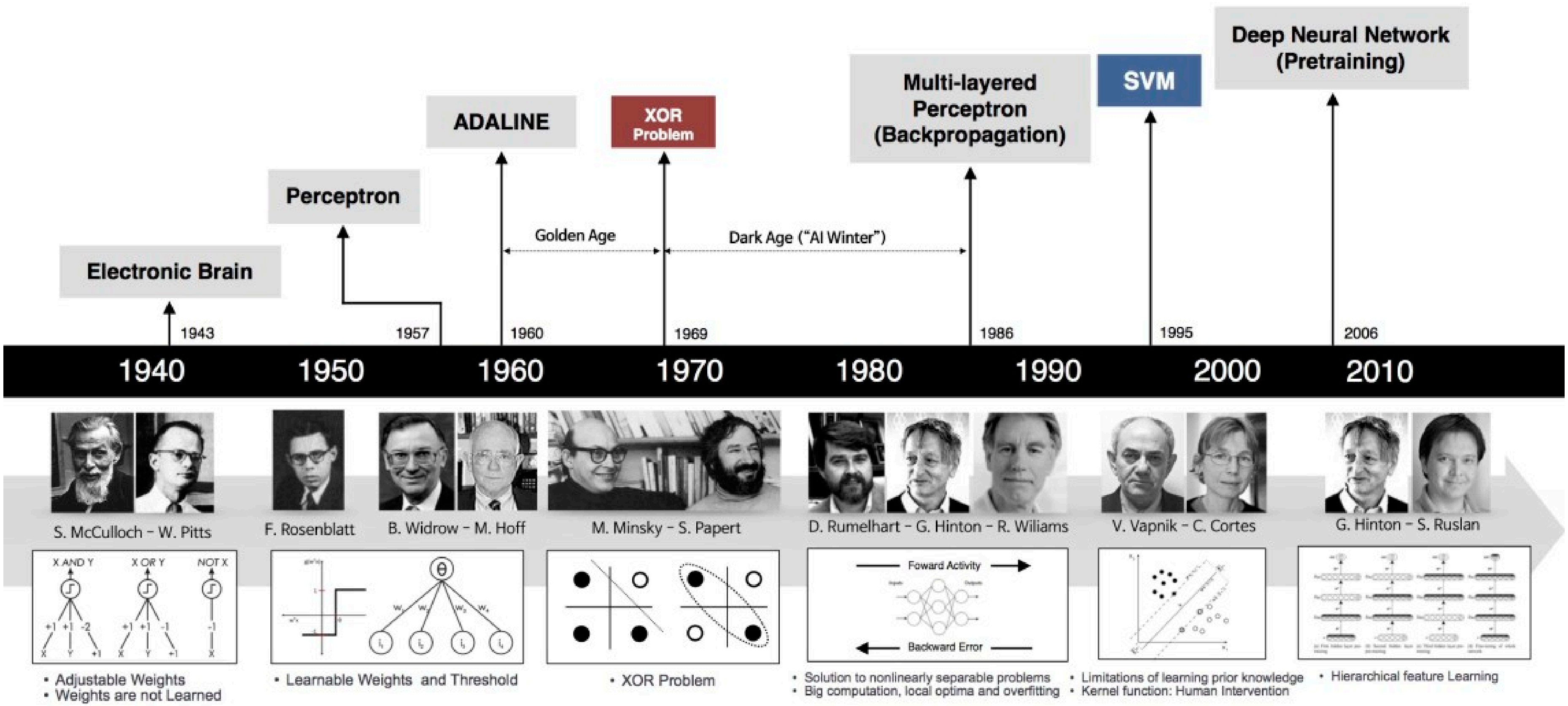
- How about XOR?

XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



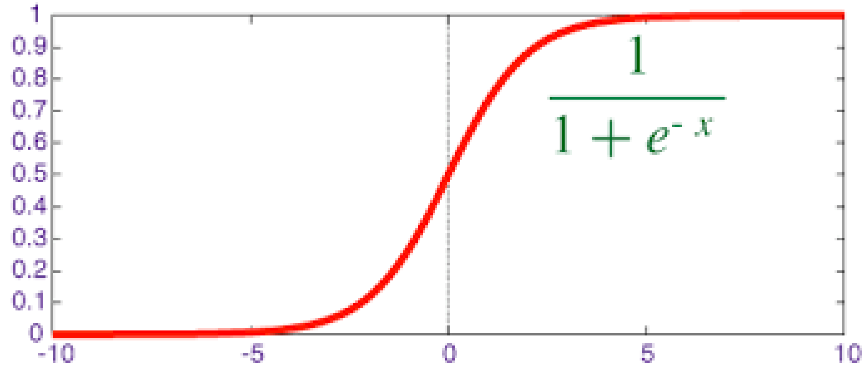
- This contributed to the first AI winter

Brief history of neural networks

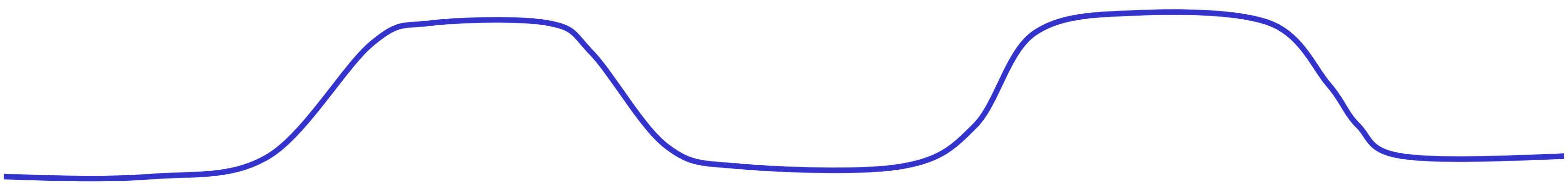


(Multi-layer) neural network

- Given sigmoid perceptrons



- Can you produce output like



- which had non-linear decision boundarys