# Advanced Introduction to Deep Learning
# Lecture 5: Feedforward Neural Networks

Han Zhao
han.zhao@cs.cmu.edu
Machine Learning Department, Carnegie Mellon University

July. 24th, 2019

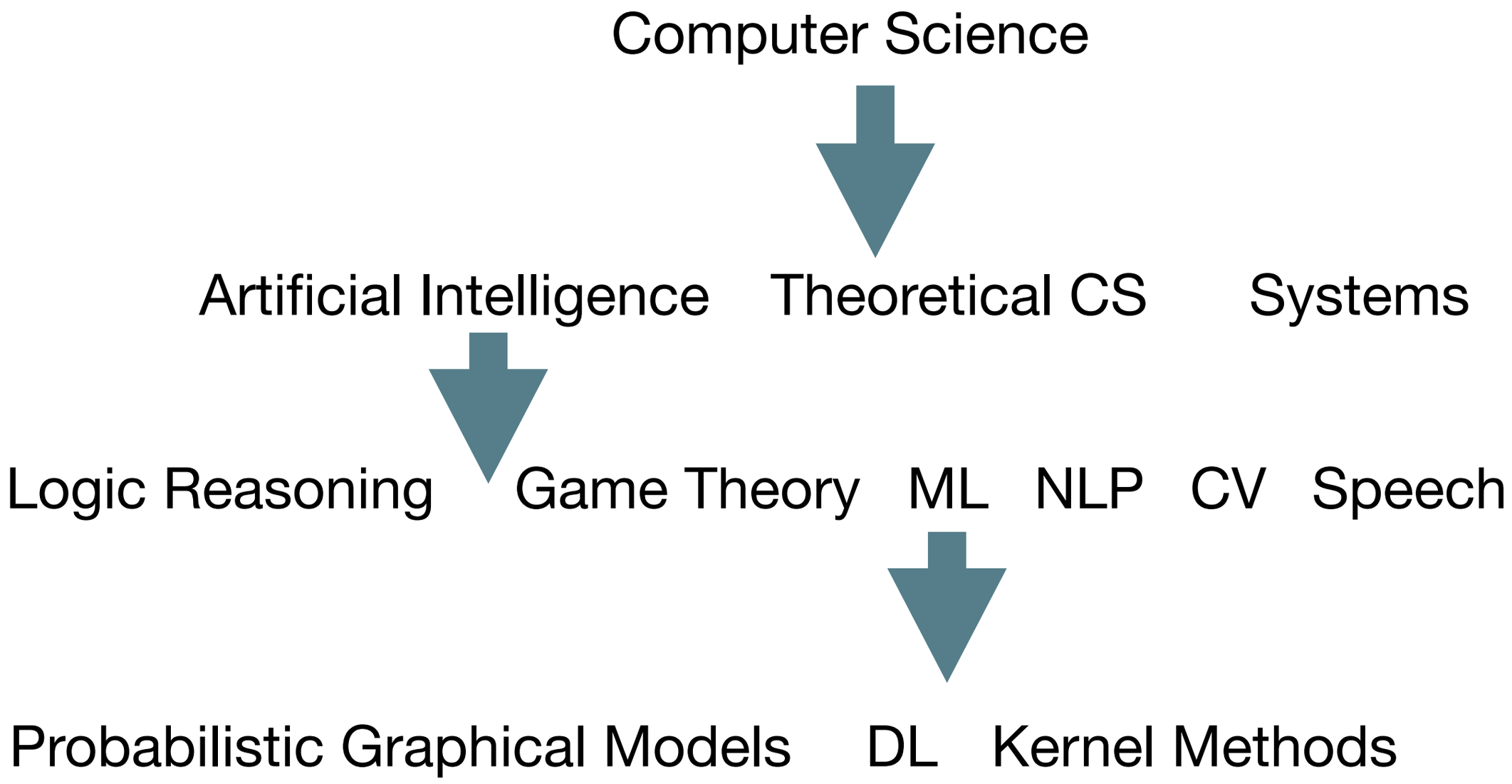
# Lecture 5: Feedforward Neural Networks

Overview:

- The history of AI and Neural Networks
- The 1960s: Perceptron
- The 1980s: Multilayer Perceptron
- The 2010s: Deep Learning

# Feedforward Neural Networks: Overview

Recall the following picture we see in the first lecture:



# Feedforward Neural Networks: Overview

If we further dive into the field of Machine Learning, then we see:

Machine Learning

Probabilistic Methods

Optimization

Kernel Methods

Deep Learning

Learning Theory

DNN

Boltzmann Machine

CNN

RNN

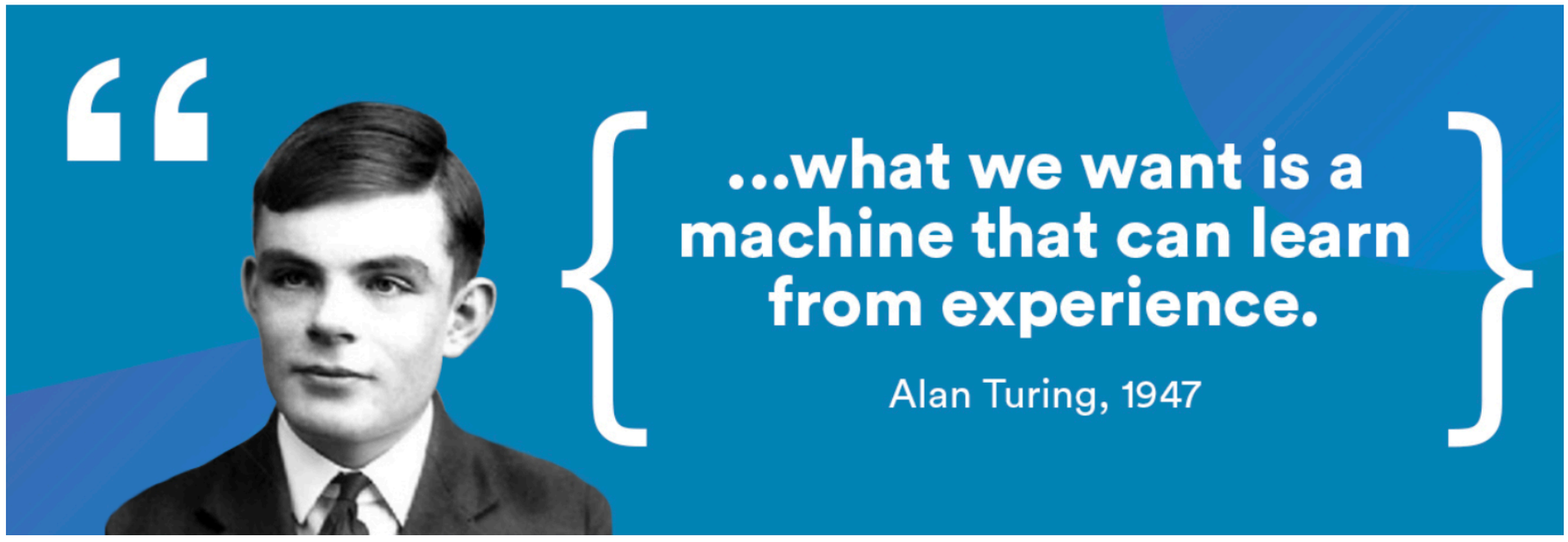
Auto-encoder

Helmholtz Machine

Hopfield Net

# Feedforward Neural Networks: Overview

Indeed, AI has a long history:




Alan M. Turing (1950) Computing Machinery and Intelligence. Mind 49: 433-460.

Turing-test: Black-box test of AI

# Feedforward Neural Networks: Overview

## The 1956 summer Dartmouth workshop

The founding event of AI as a field:

- Organized by John McCarthy, the father of AI, then an AP of Maths @ Dartmouth College
- The original proposal of Dartmouth workshop:

> We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

# Feedforward Neural Networks: Overview

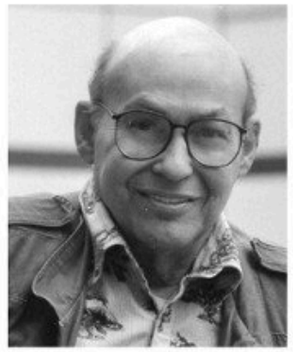
## The 1956 summer Dartmouth workshop

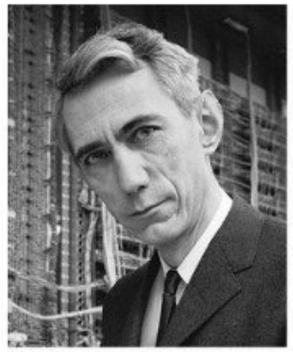
The founding event of AI as a field:

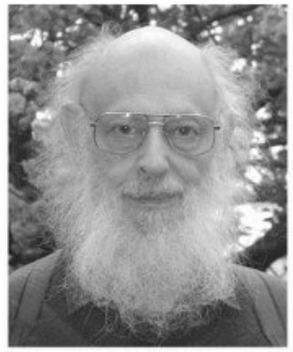
- Organized by John McCarthy, the father of AI, then an AP of Maths @ Dartmouth College

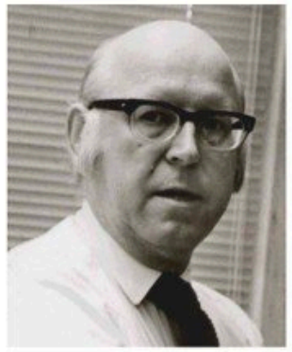
**1956 Dartmouth Conference:**
**The Founding Fathers of AI**


John MacCarthy


Marvin Minsky


Claude Shannon


Ray Solomonoff

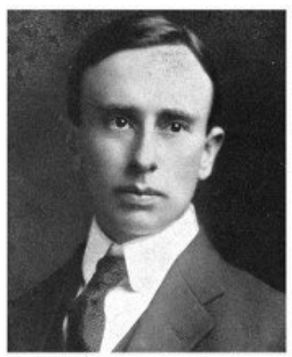

Alan Newell


Herbert Simon


Arthur Samuel


Oliver Selfridge


Nathaniel Rochester

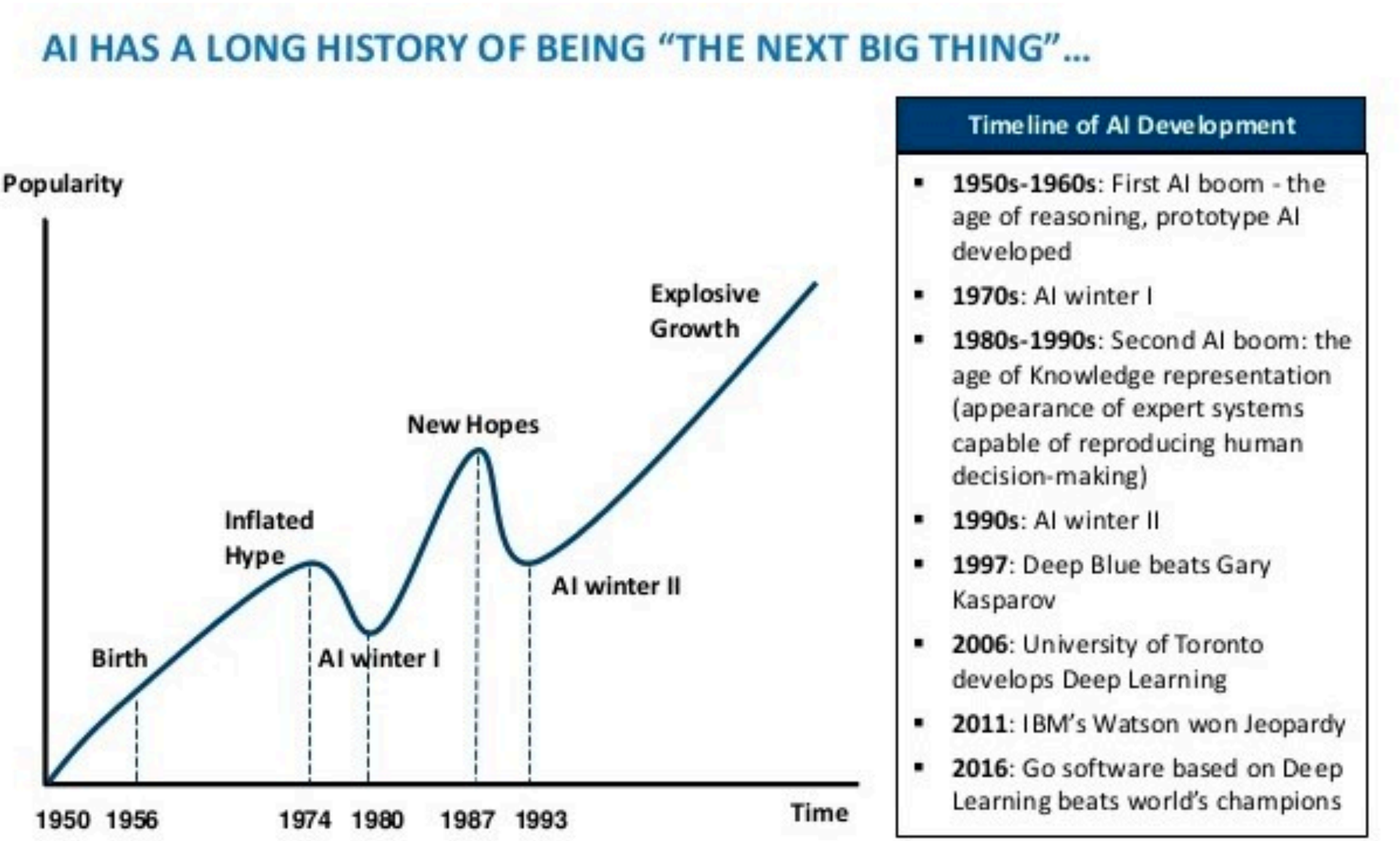

Trenchard More

# Feedforward Neural Networks: Overview

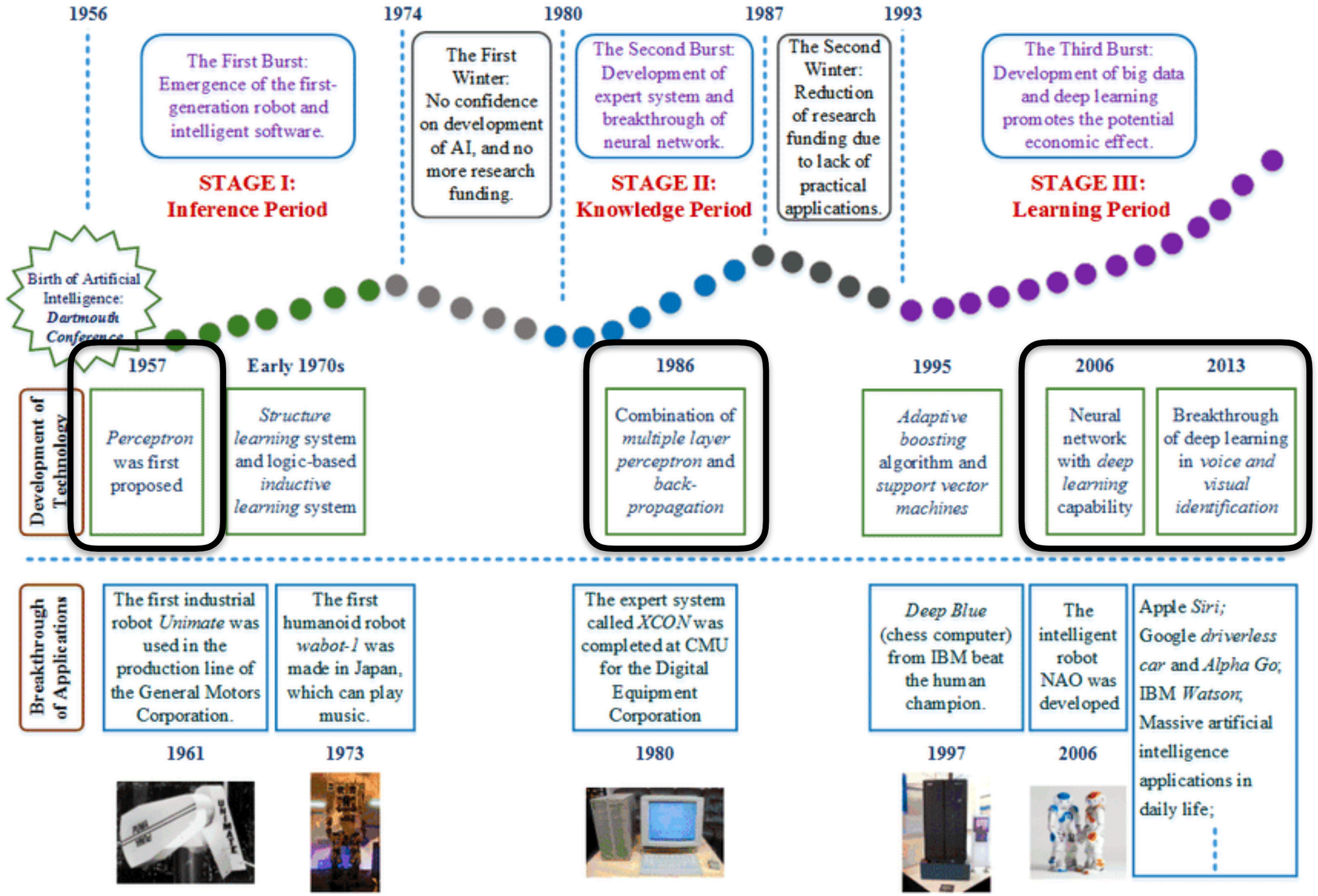
# Feedforward Neural Networks: Overview

# Feedforward Neural Networks: Overview

Indeed, AI has a long history:



# Feedforward Neural Networks: Overview



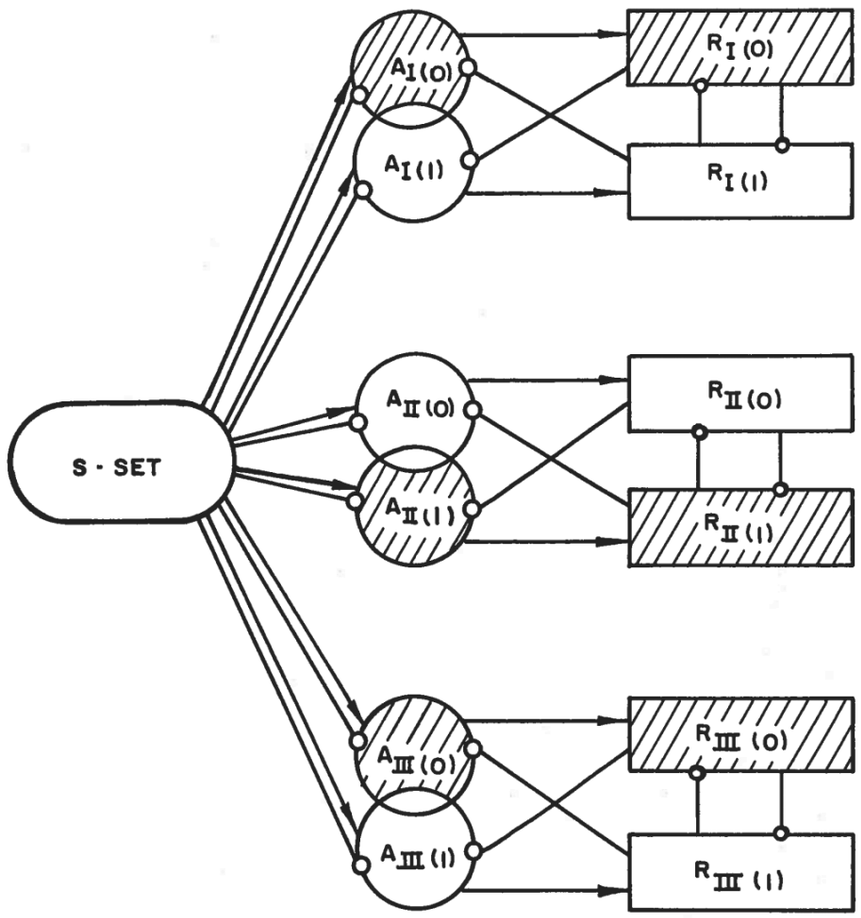
# Lecture 5: Feedforward Neural Networks

Overview:

- The history of AI and Neural Networks
- The 1960s: Perceptron
- The 1980s: Multilayer Perceptron
- The 2010s: Deep Learning

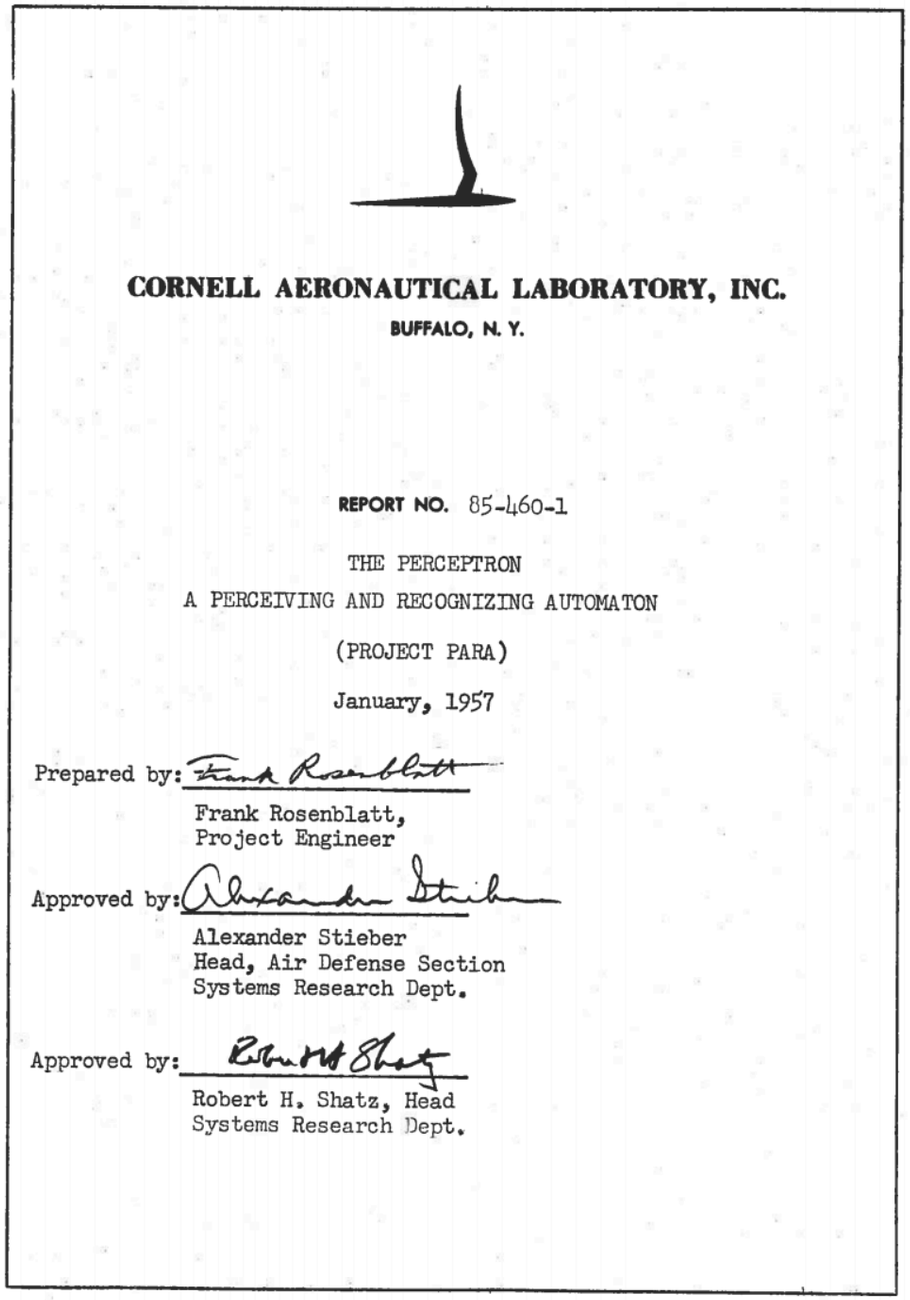
# The 1960s: Perceptron

Frank Rosenblatt (1957): The Perceptron: A Perceiving and Recognizing Automaton




NOTE:
THE SHADING SHOWS THE ASSOCIATION SETS AND R-UNITS WHICH WOULD BE INHIBITED WHEN THE RESPONSE 101 IS ACTIVE.

FIGURE 3
ORGANIZATION OF A PERCEPTRON WITH THREE BINARY RESPONSE SETS



CORNELL AERONAUTICAL LABORATORY, INC.
BUFFALO, N. Y.

REPORT NO. 85-460-1

THE PERCEPTRON
A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

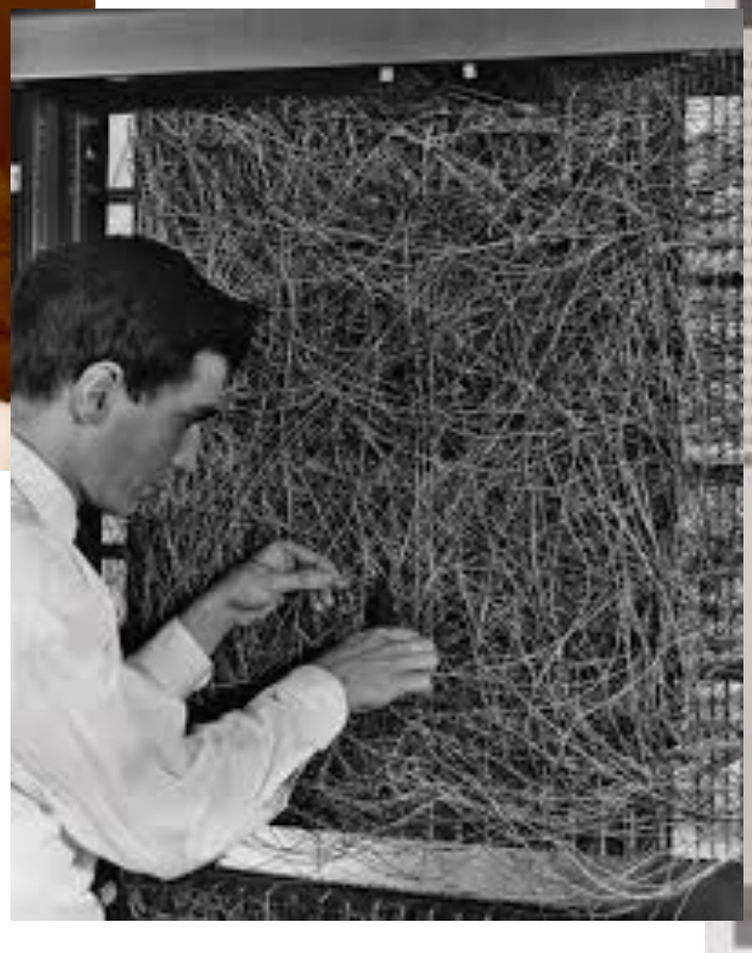
Prepared by: Frank Rosenblatt
Frank Rosenblatt,
Project Engineer

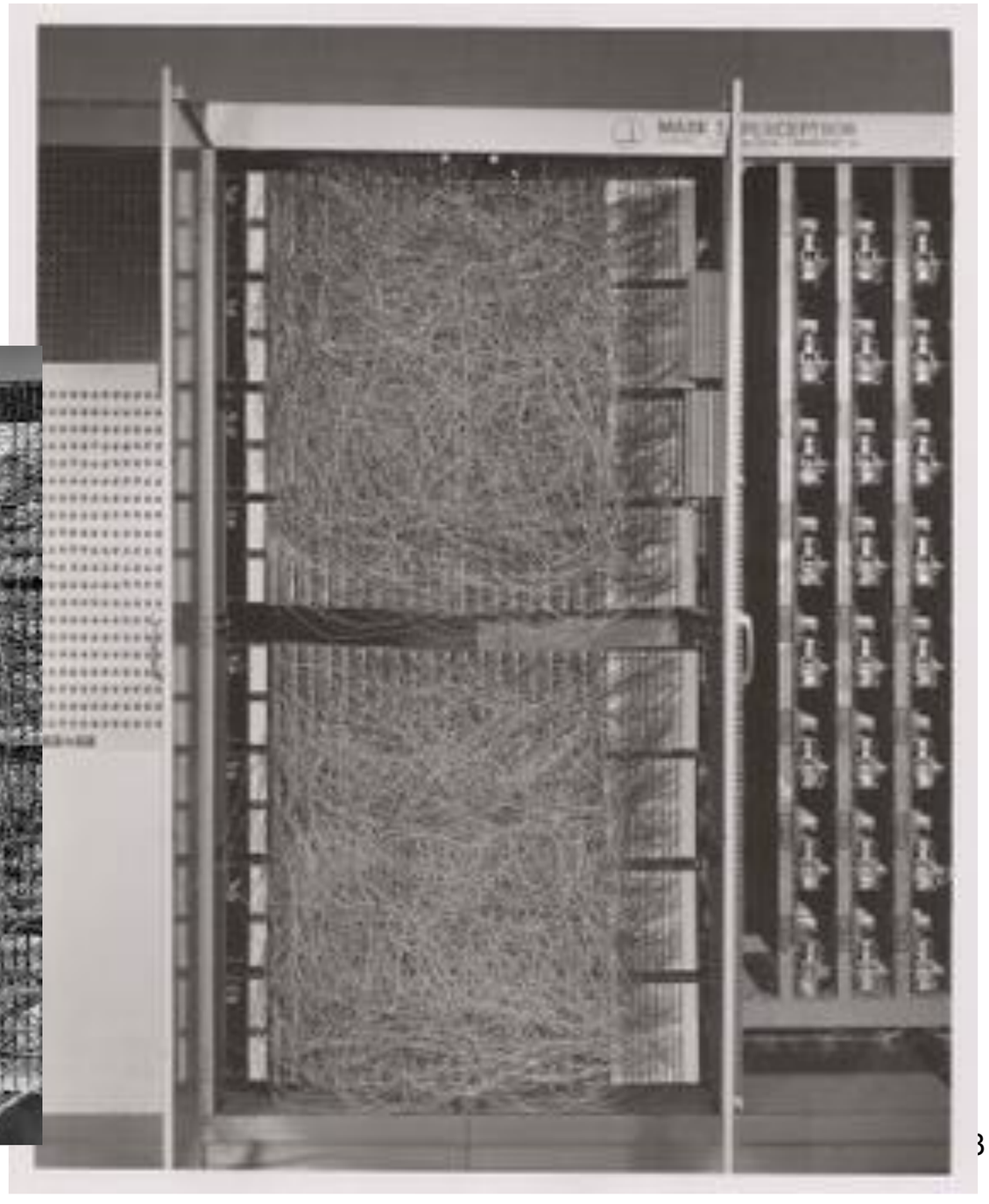
Approved by: Alexander Stieber
Alexander Stieber
Head, Air Defense Section
Systems Research Dept.

Approved by: Robert H. Shatz
Robert H. Shatz, Head
Systems Research Dept.

# The 1960s: Perceptron

Frank Rosenblatt (1957): The Perceptron: A Perceiving and Recognizing Automaton



# The 1960s: Perceptron

## The Perceptron Algorithm: Biological analogy

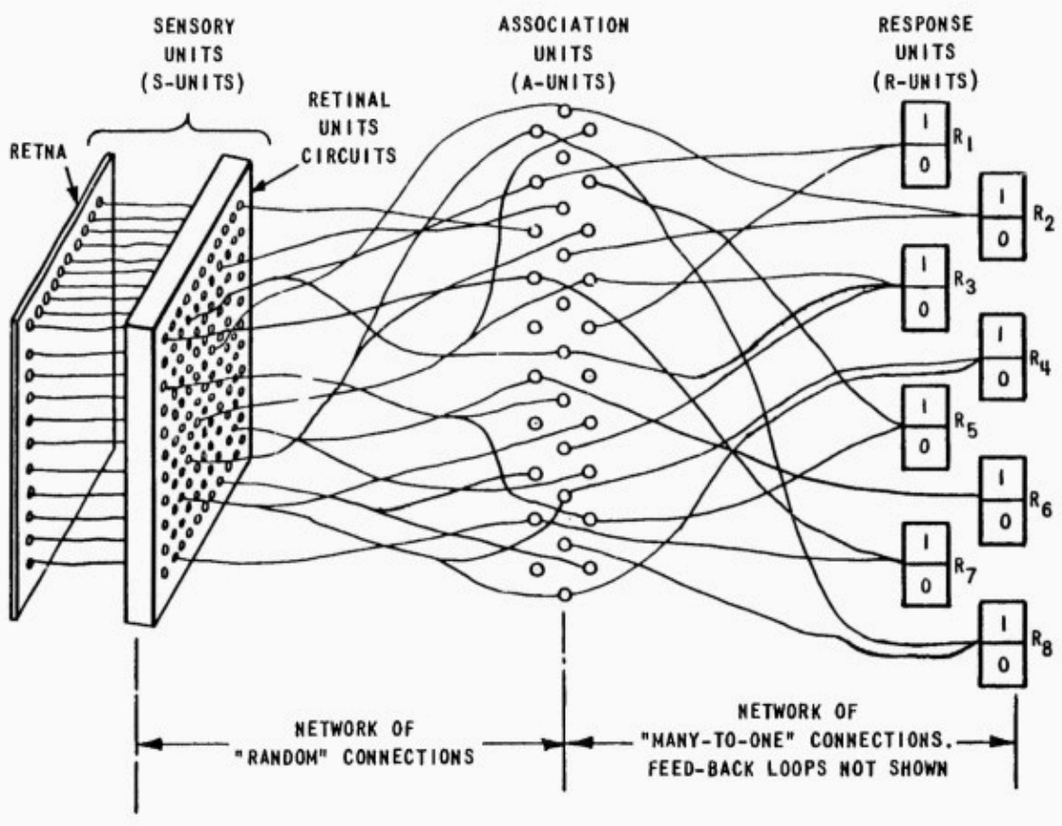
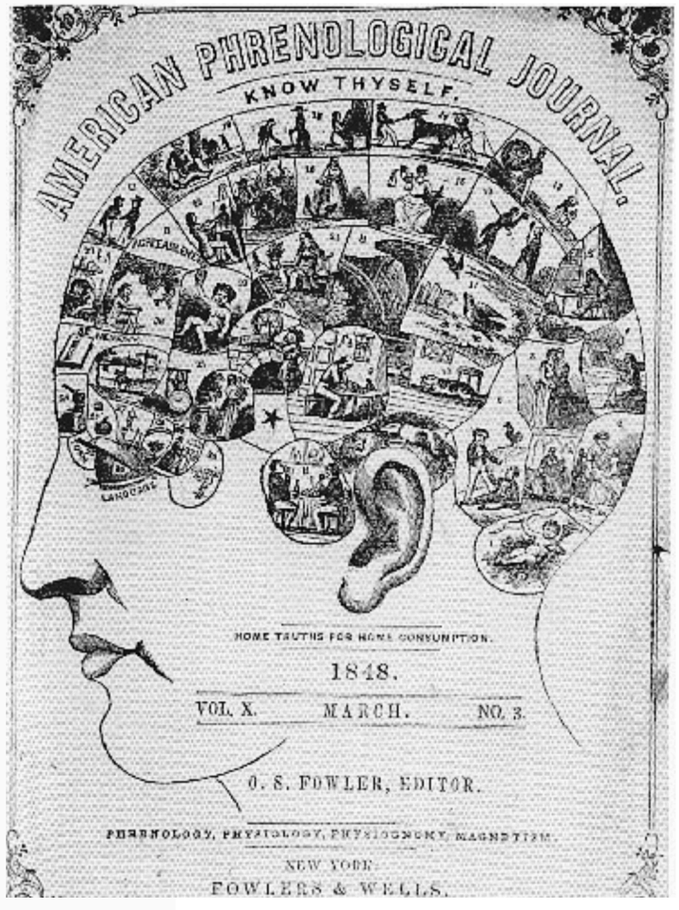



Figure 1 ORGANIZATION OF THE MARK 1 PERCEPTRON

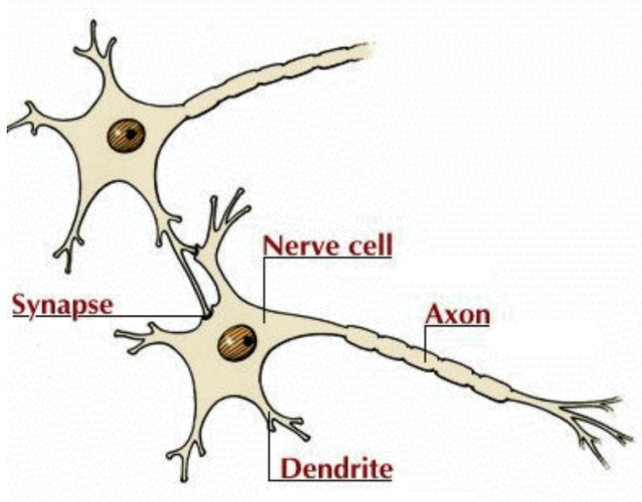



Neurons and Learning

# The 1960s: Perceptron

## The Perceptron Algorithm: Biological analogy

- Basic ideas:
  - Good behaviors should be rewarded, bad behaviors punished (or not rewarded). This improves system fitness
  - Correlated events should be combined
- Training mechanisms:
  - Behavioral modification of individuals (learning)
  - Successful behavior is rewarded
  - The wrongly  coded animal does not reproduce

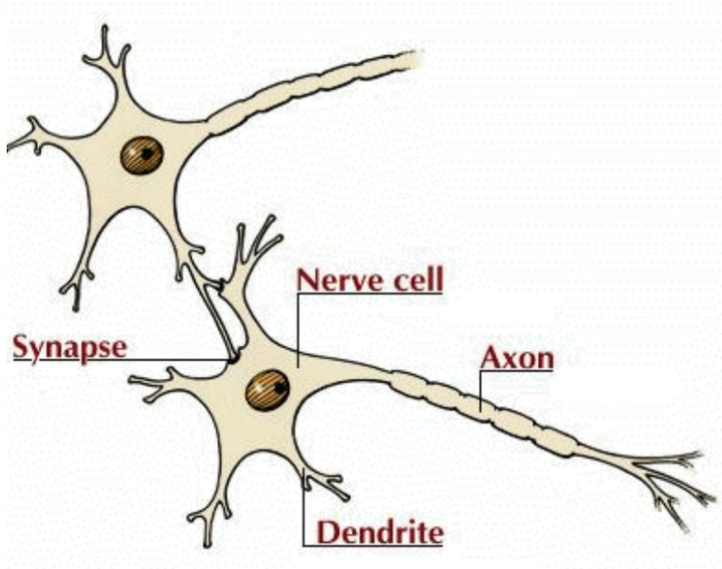


# The 1960s: Perceptron

The Perceptron Algorithm: Biological analogy

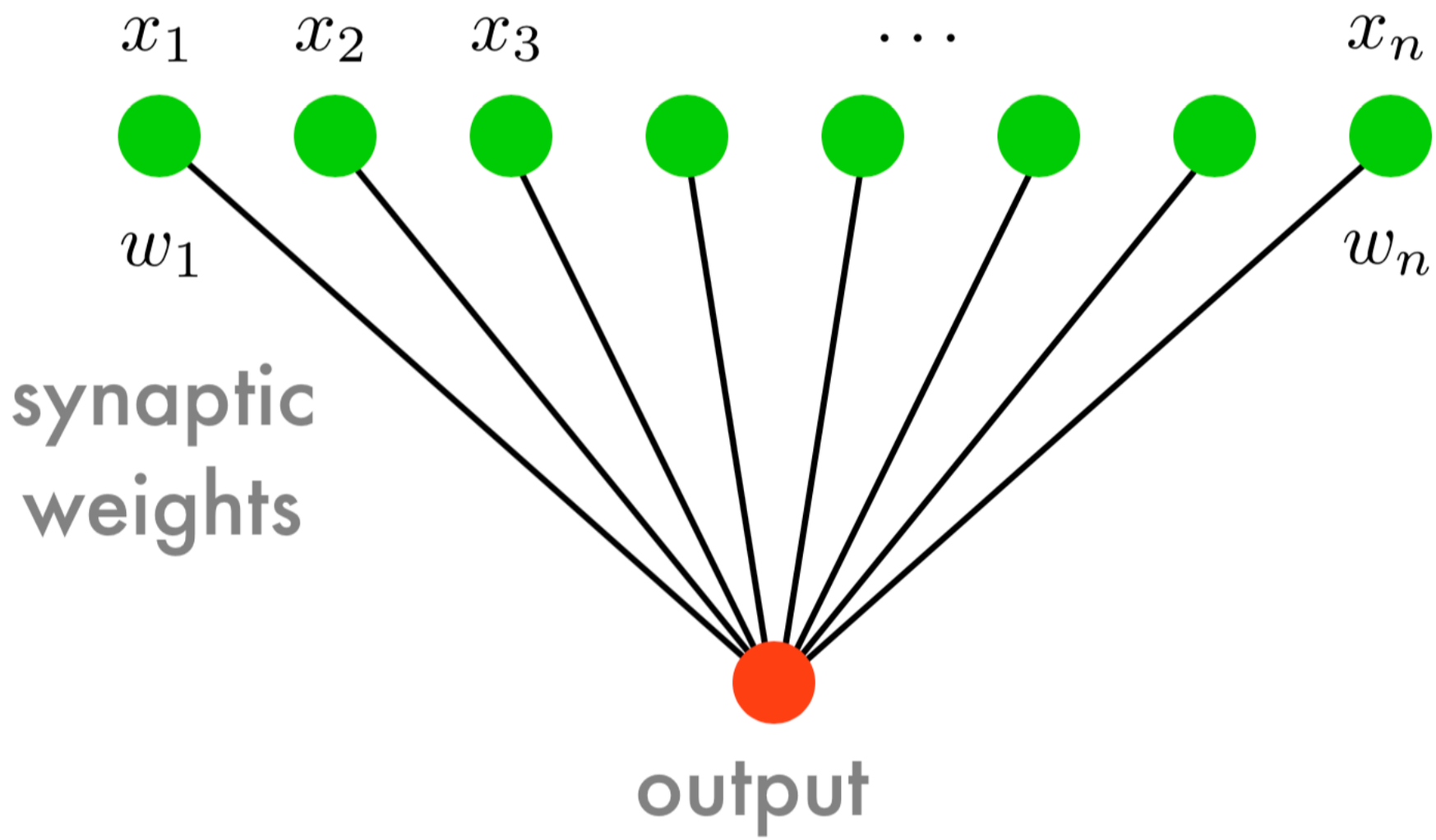
- Neuron:
  - Soma: cell body, combines signals (CPU)
  - Dendrite: combines the inputs from several other nerve cells (input bus)
  - Synapse: Interface and parameter store between neurons (interface)
  - Axon: Transport the activation signal to neurons at different locations (cable)

Quite accurate analogy between human brains and our computers!



# The 1960s: Perceptron

The Perceptron Algorithm: Linear classifier with hard-thresholding

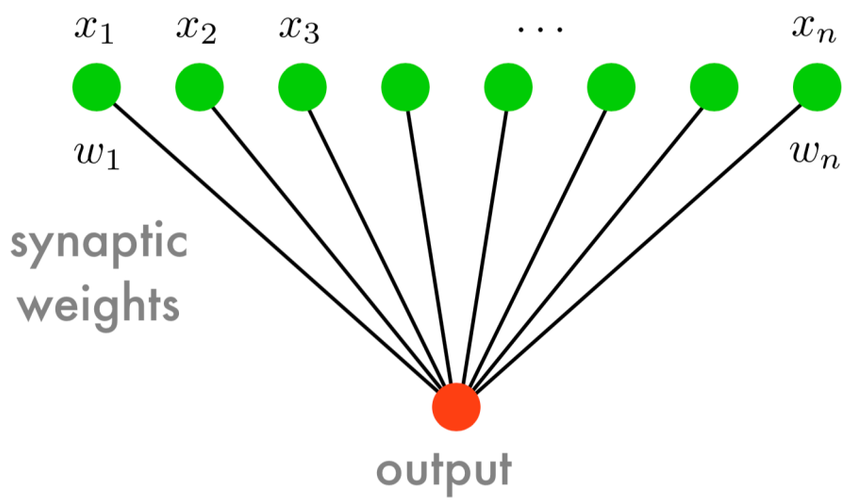



Recall: what's this? $f(\mathbf{x}) = \mathbb{I}(\mathbf{w}^T\mathbf{x} + b \geq 0)$

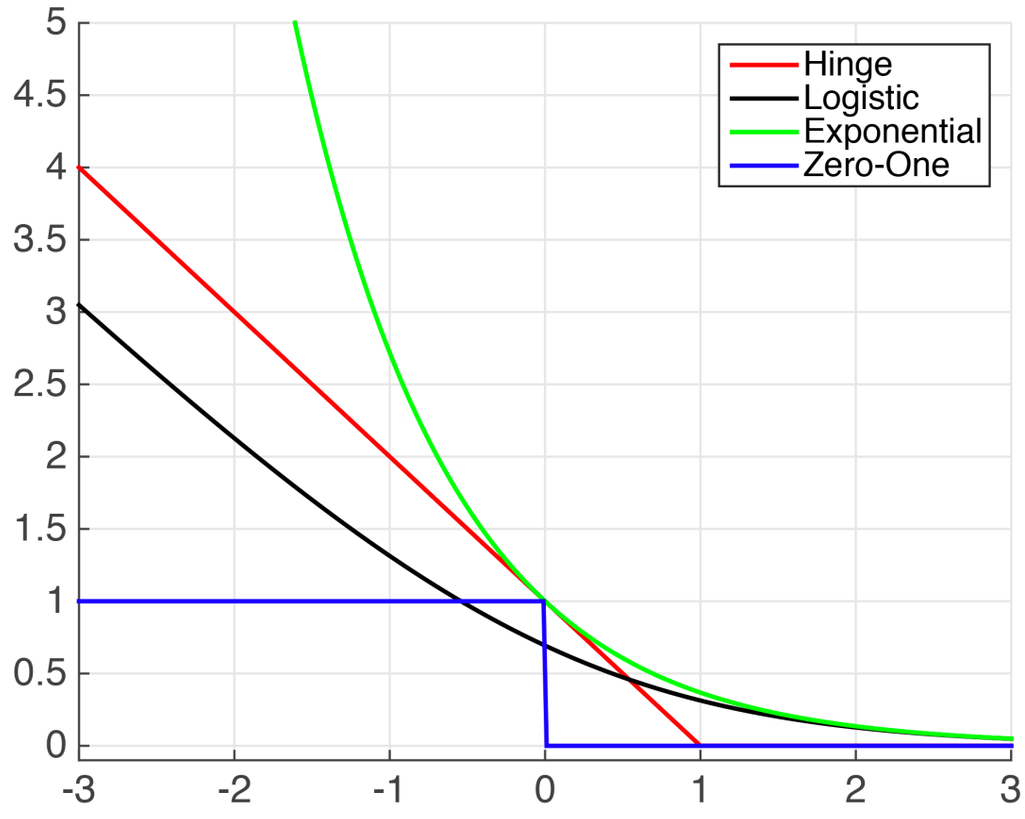
# The 1960s: Perceptron

The Perceptron Algorithm:

**initialize** $w = 0$ and $b = 0$
**repeat**
  **if** $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  **end if**
**until** all classified correctly




Hinge: SVM
Logistic: Logistic Regression
Exponential: AdaBoost
Zero-One: No (NP-hard)

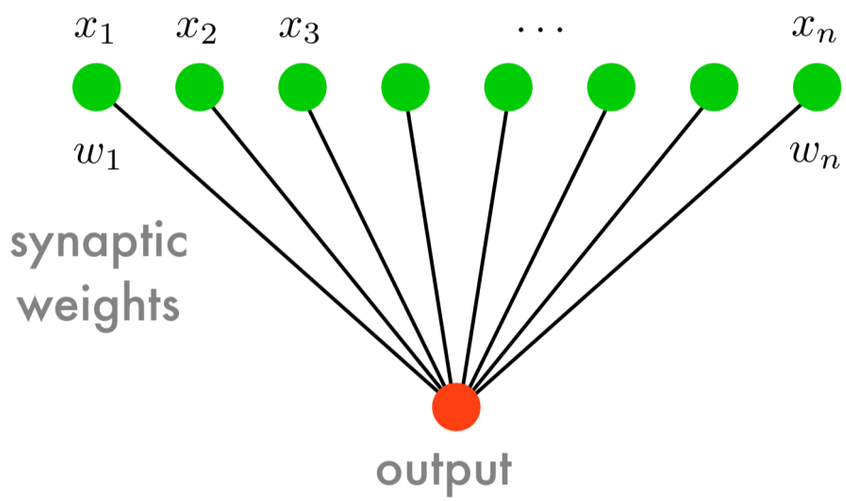



Think: what's the loss function of Perceptron?

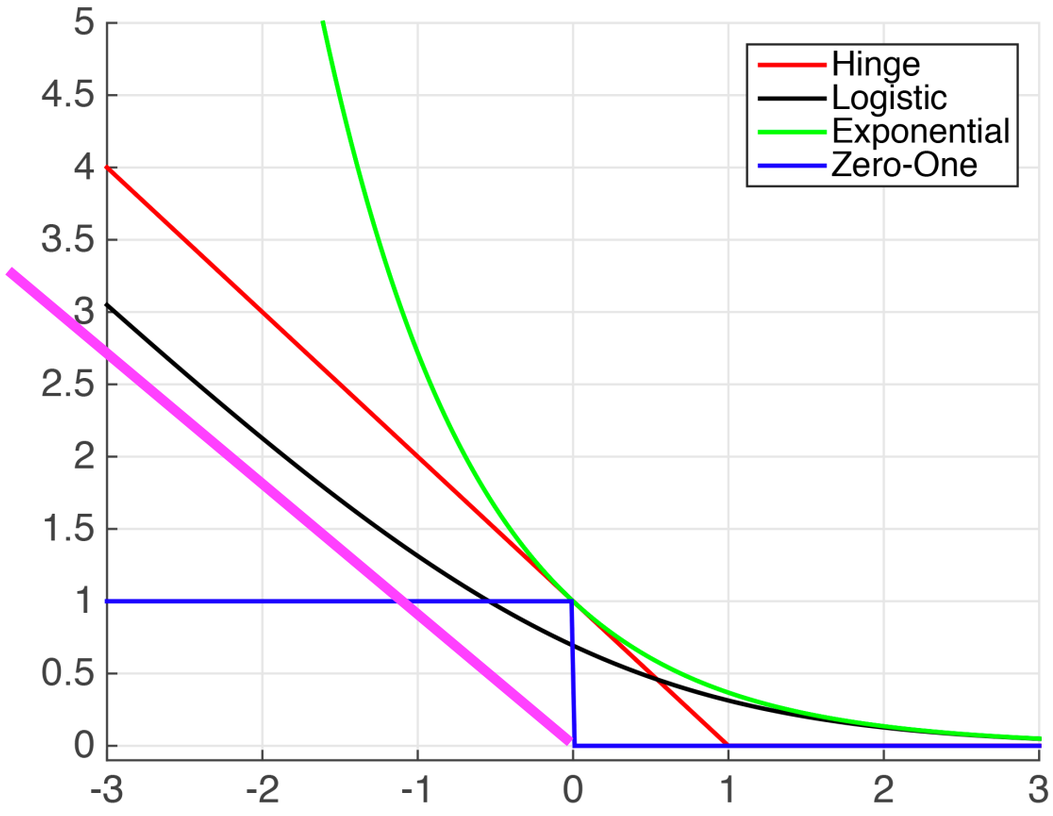
# The 1960s: Perceptron

The Perceptron Algorithm:

**initialize** $w = 0$ and $b = 0$
**repeat**
  **if** $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  **end if**
**until** all classified correctly




Hinge: SVM
Logistic: Logistic Regression
Exponential: AdaBoost
Zero-One: No (NP-hard)
Perceptron




Think: what's the loss function of Perceptron?

# The 1960s: Perceptron

## The Perceptron Algorithm: Convergence Analysis

Will the algorithm converge on linearly separable dataset? If yes, how long does it take the algorithm to converge?

Several key observations:

- Nothing happened if classified correctly
- Weight vector is a linear combination of signed inputs $\mathbf{w} = \sum_{i \in I} y_i \mathbf{x}_i$
- Classifier is a linear combination of inner products

$$f(\mathbf{x}) = \sum_{i \in I} y_i \mathbf{x}_i^T \mathbf{x} + b$$

**initialize** $w = 0$ and $b = 0$
**repeat**
  **if** $y_i \left[ \langle w, x_i \rangle + b \right] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  **end if**
**until** all classified correctly

# The 1960s: Perceptron

## The Perceptron Algorithm: Convergence Analysis

Theorem: If there exists $(\mathbf{w}^*, b^*)$ with unit length and $y_i(\mathbf{x}_i^T \mathbf{w}^* + b^*) \geq \rho, \ \forall i \in [n]$, $||\mathbf{x}_i||_2 \leq r, \ \forall \in [n]$, then the perceptron algorithm converges to a linear classifier after a number of steps bounded by:

$$\frac{r^2}{\rho^2}$$

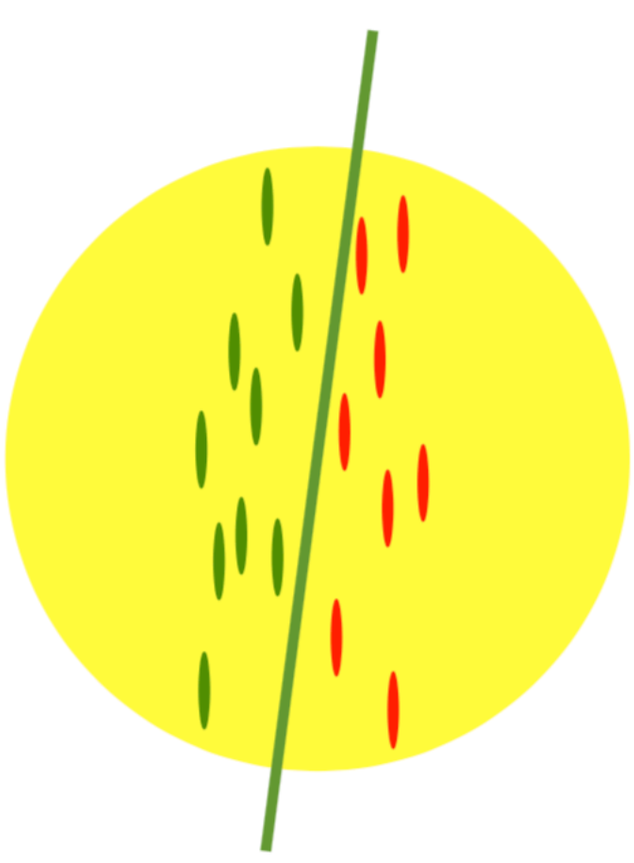
- Dimensionality independent
- Large margin implies faster convergence

Proof on board

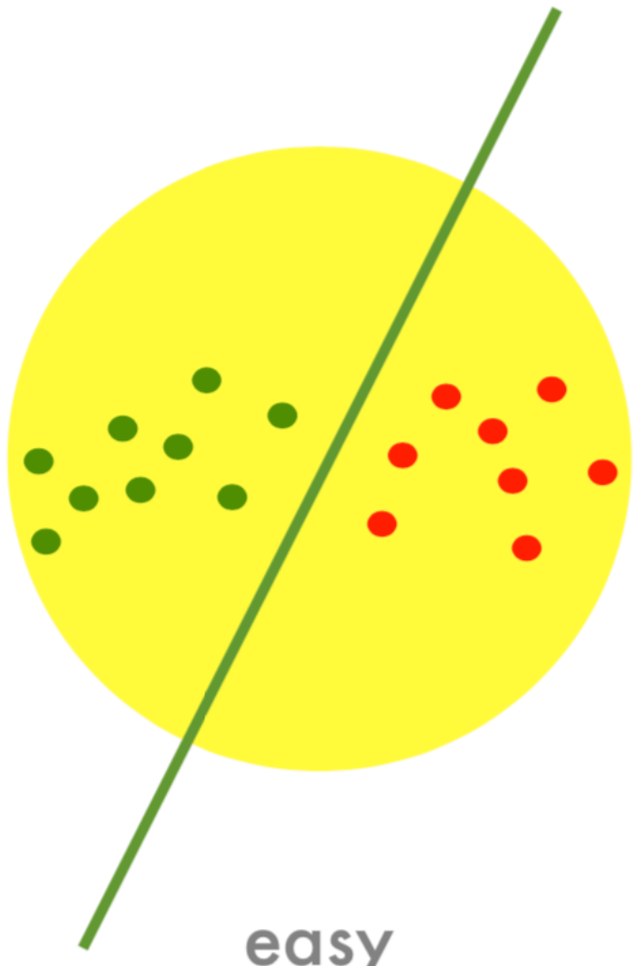
**initialize** $w = 0$ and $b = 0$
**repeat**
  **if** $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  **end if**
**until** all classified correctly

# The 1960s: Perceptron

## The Perceptron Algorithm: visualization



hard

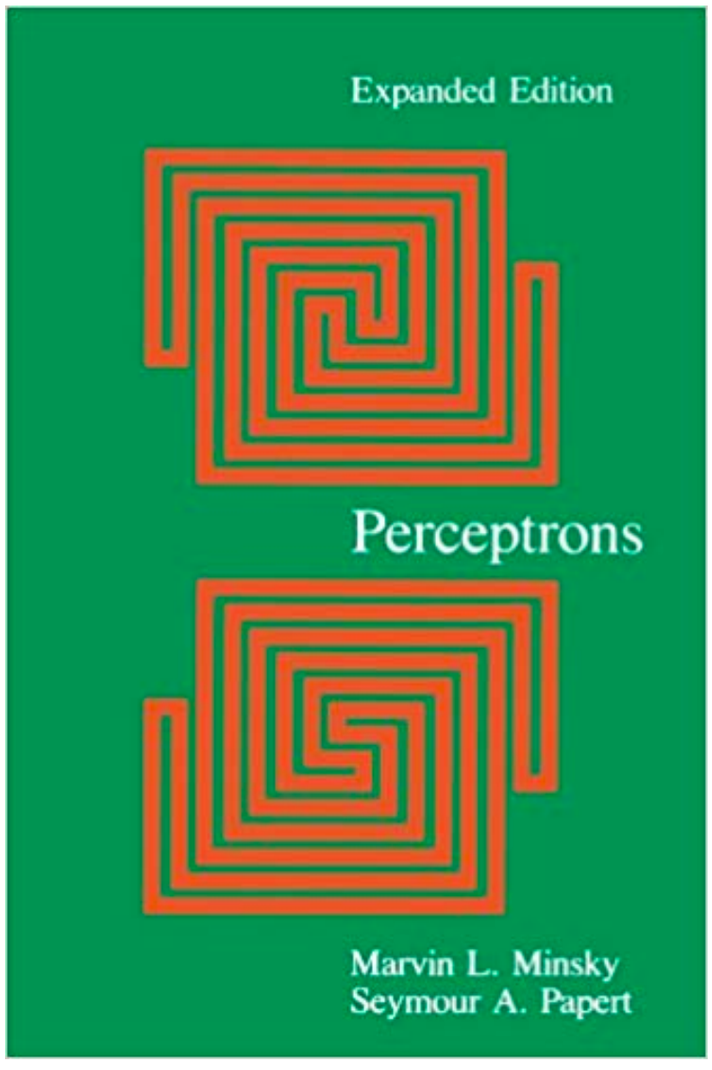


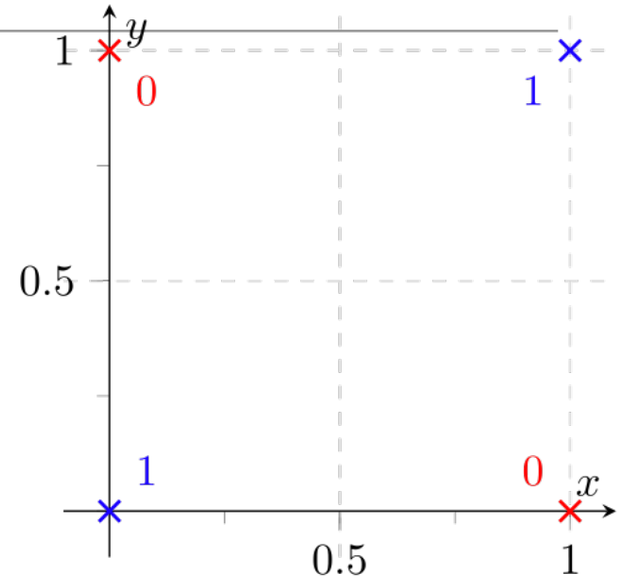
easy

# The 1960s: Perceptron

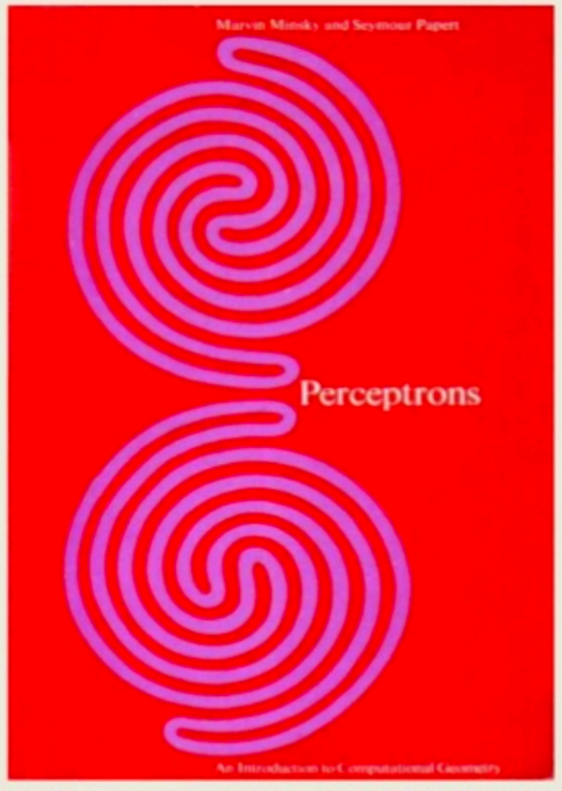
## The “Perceptrons” Book

The first AI winter

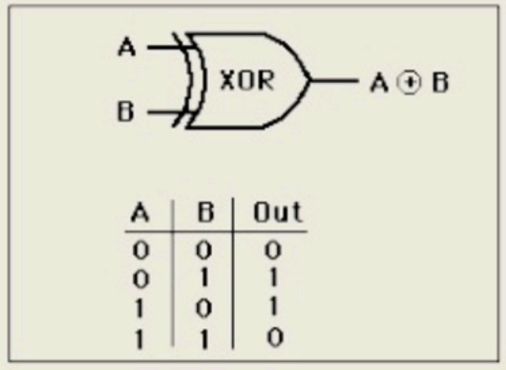



## 1969: Perceptrons can’t do XOR!




http://www.i-programmer.info/images/stories/BabBag/AI/book.jpg




http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/ietron/xor.gif

Minsky & Papert

https://constructingkids.files.wordpress.com/2013/05/minsky-papert-71-csolomon-x640.jpg

# Lecture 5: Feedforward Neural Networks

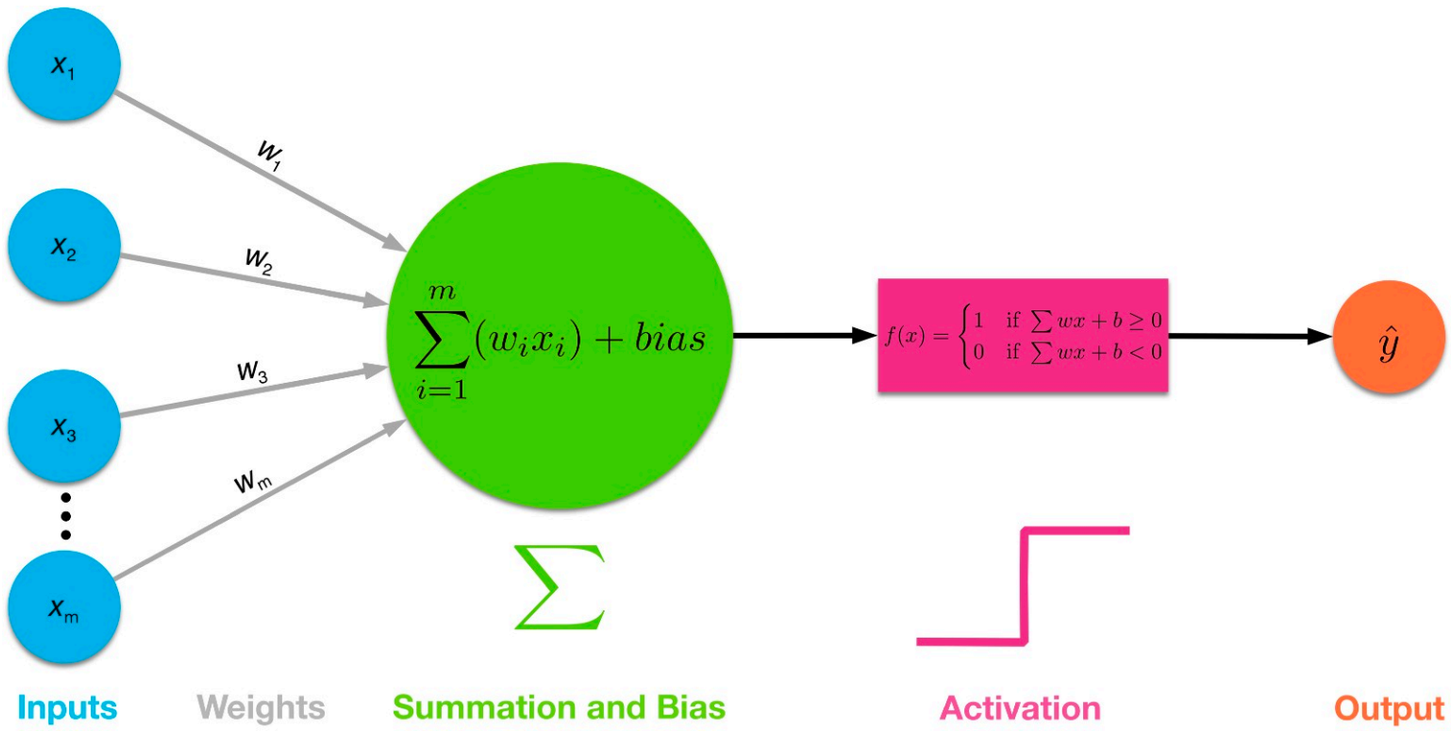
Overview:

- The history of AI and Neural Networks
- The 1960s: Perceptron
- The 1980s: Multilayer Perceptron
- The 2010s: Deep Learning

# The 1980s: Multilayer Perceptron

## The second rise of AI

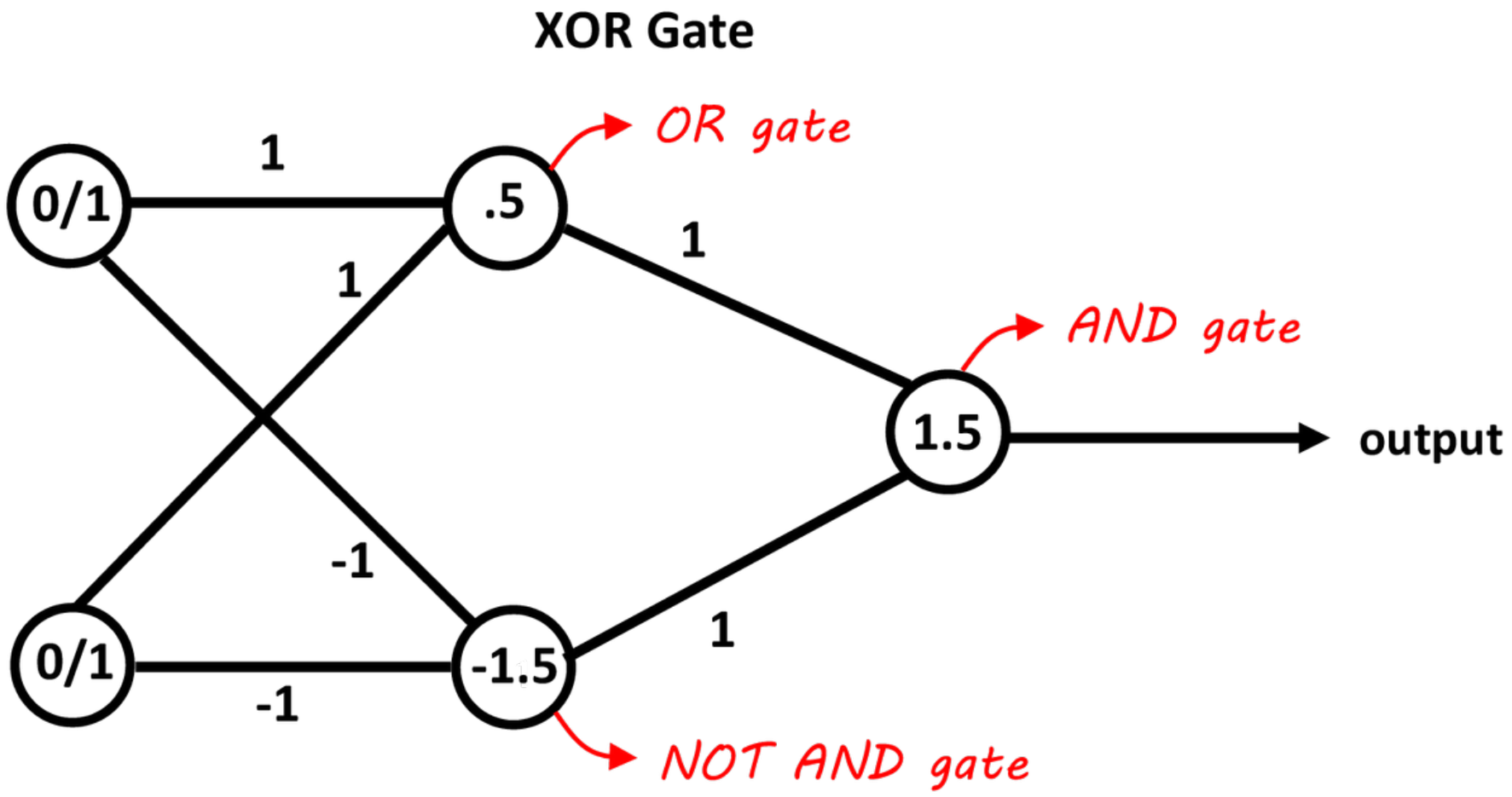
The perceptron algorithm could be equivalently represented as the following diagram:




It cannot solve XOR, but how about multiple such artificial neurons?

# The 1980s: Multilayer Perceptron

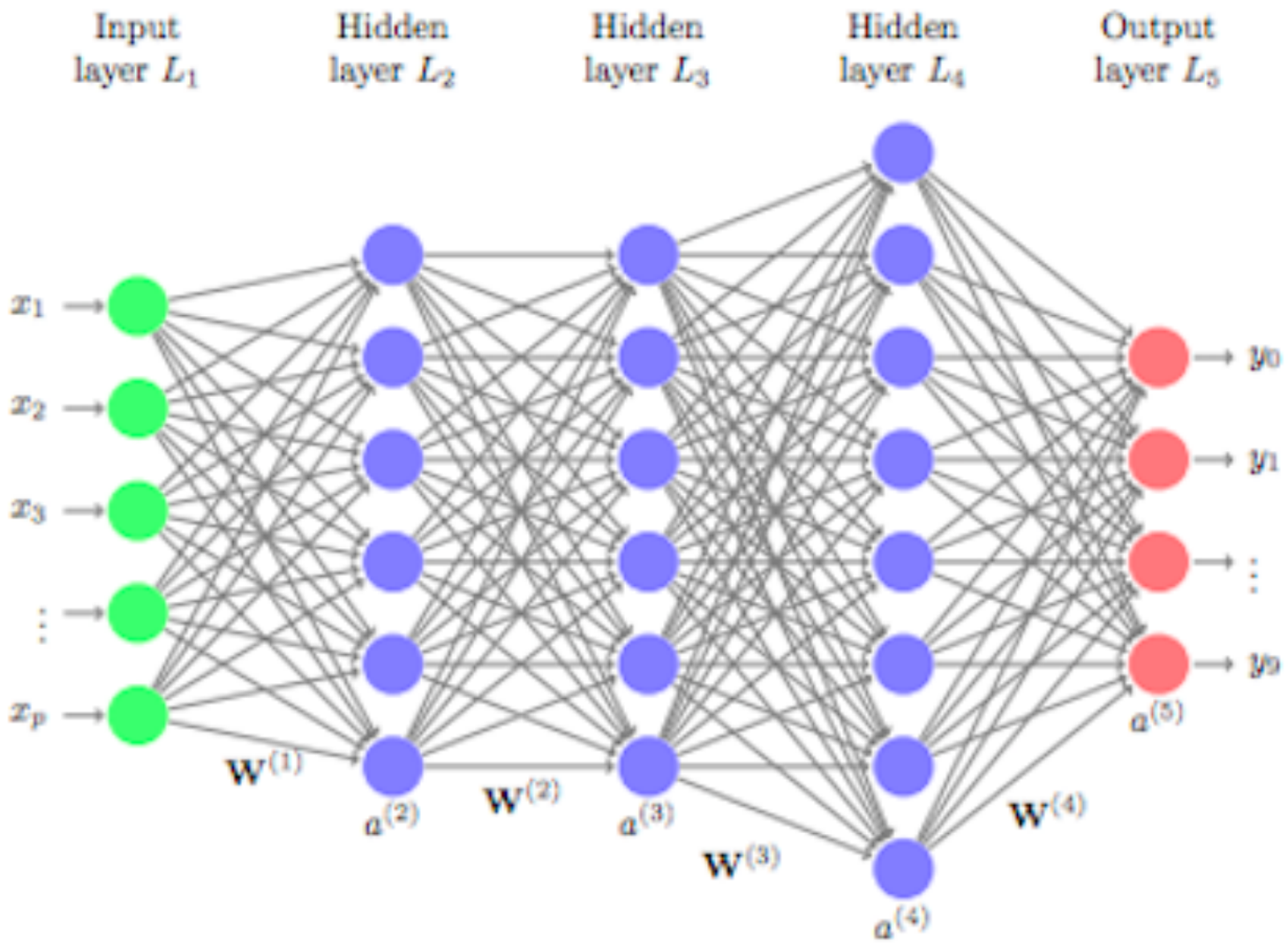
XOR can be solved by two-layer networks:




Key idea: stacking multiple neurons into several layers

# The 1980s: Multilayer Perceptron

Multilayer Perceptron:

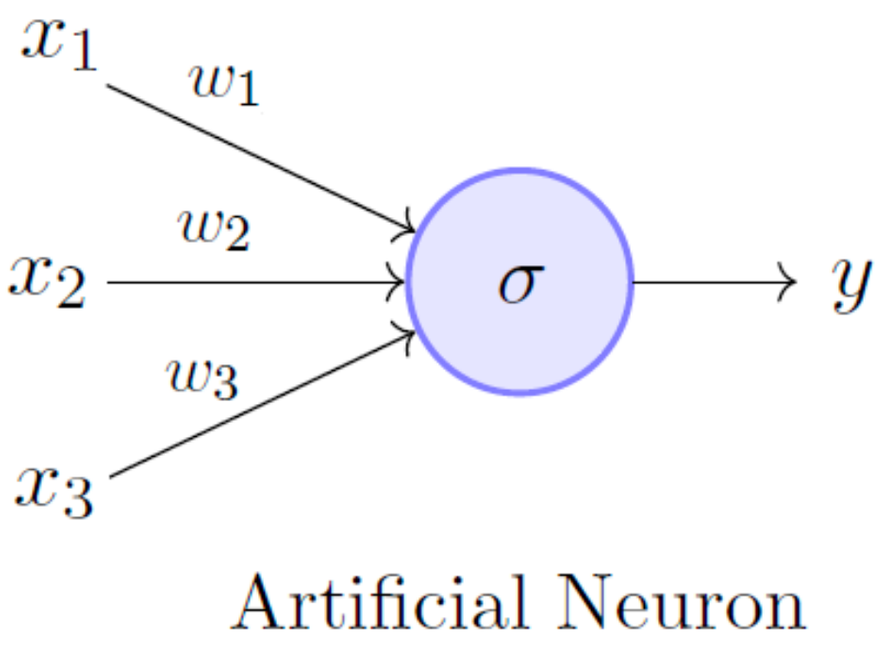



# The 1980s: Multilayer Perceptron

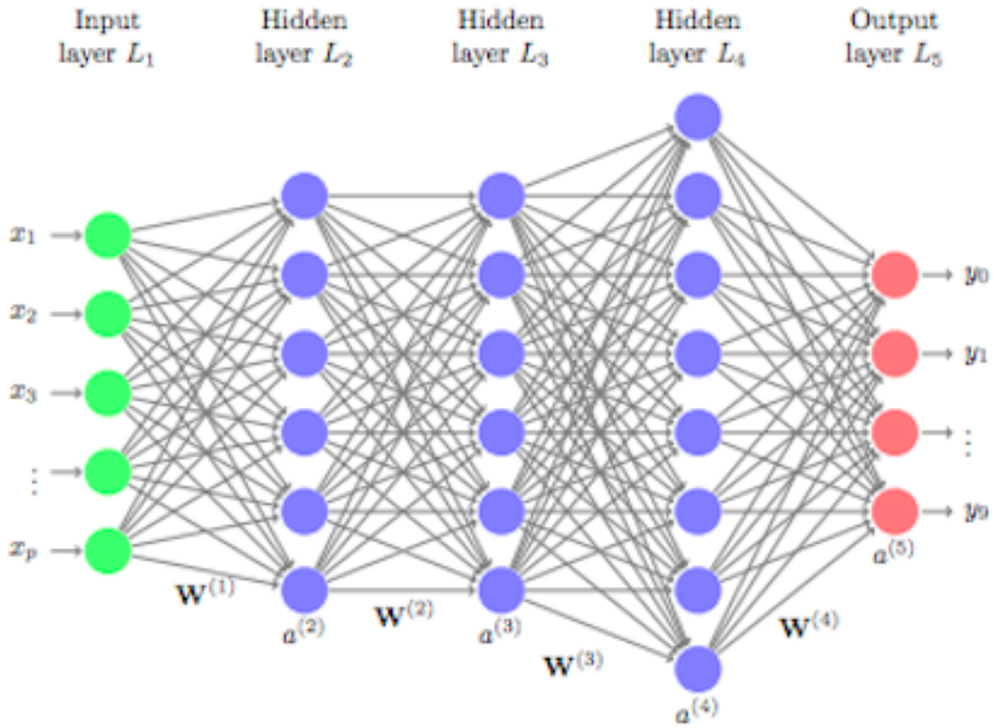
Multilayer Perceptron:

$$\mathbf{a}^{(l+1)} = \sigma(W^{(l)}\mathbf{a}^{(l)})$$

where $\sigma(\cdot)$ is a component-wise nonlinear function, also known as the activation function



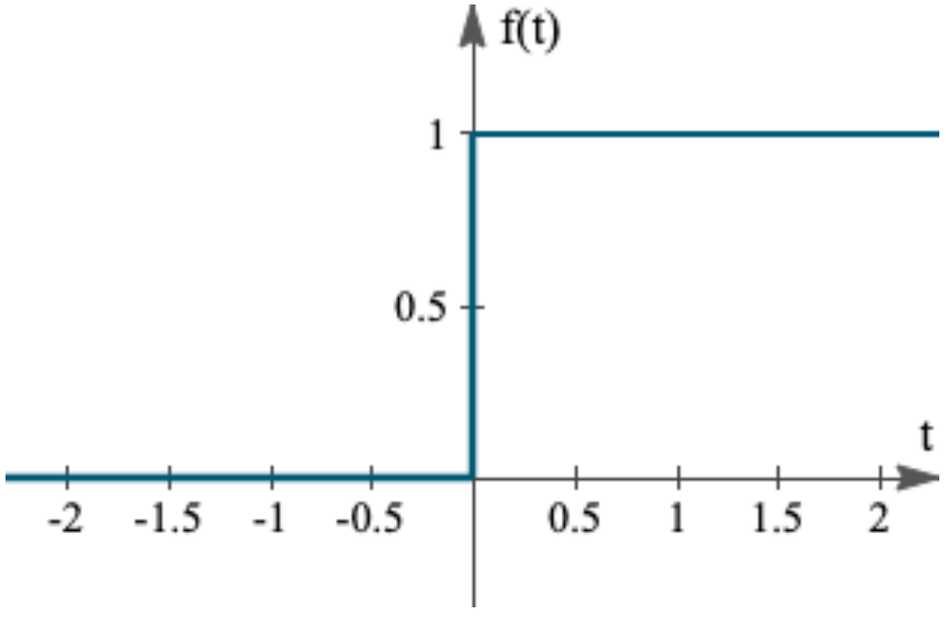

Artificial Neuron



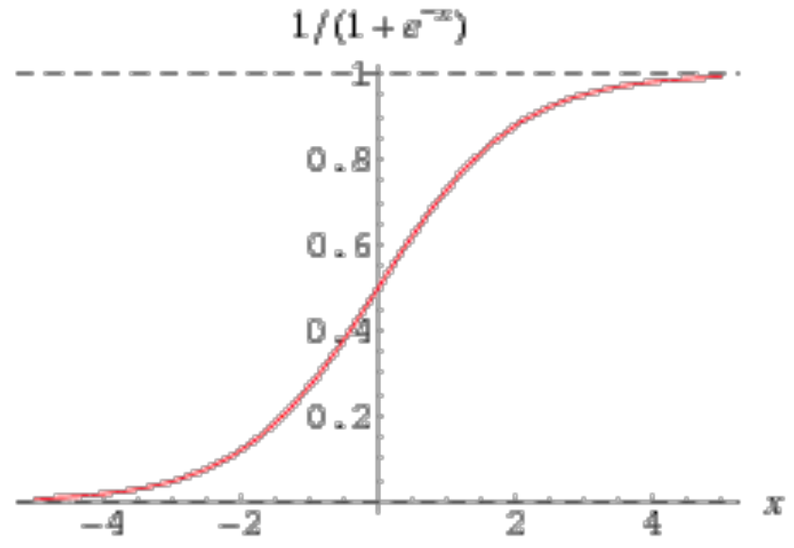
# The 1980s: Multilayer Perceptron

Multilayer Perceptron: activation functions

$$\mathbf{a}^{(l+1)} = \sigma(W^{(l)}\mathbf{a}^{(l)})$$




Step function
(hard thresholding)

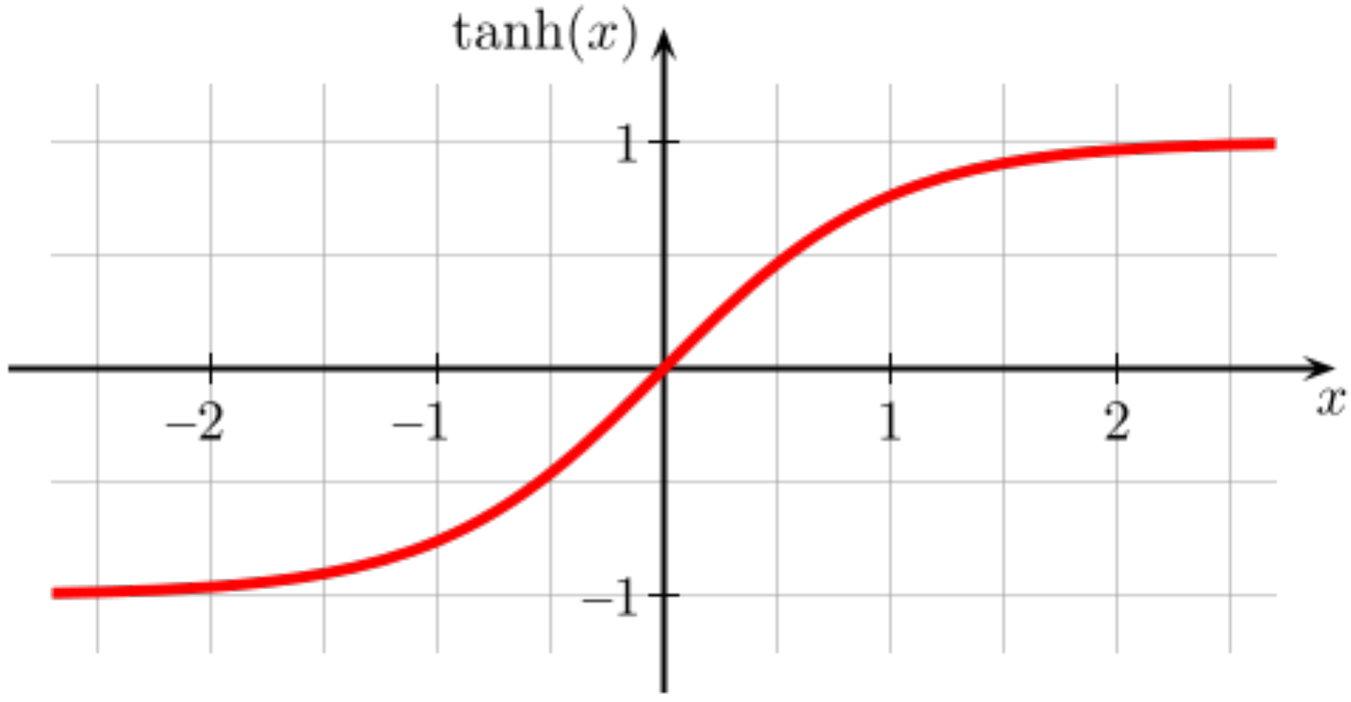



Sigmoid function
(soft thresholding)

# The 1980s: Multilayer Perceptron

## Multilayer Perceptron: activation functions

$$\mathbf{a}^{(l+1)} = \sigma(W^{(l)}\mathbf{a}^{(l)})$$

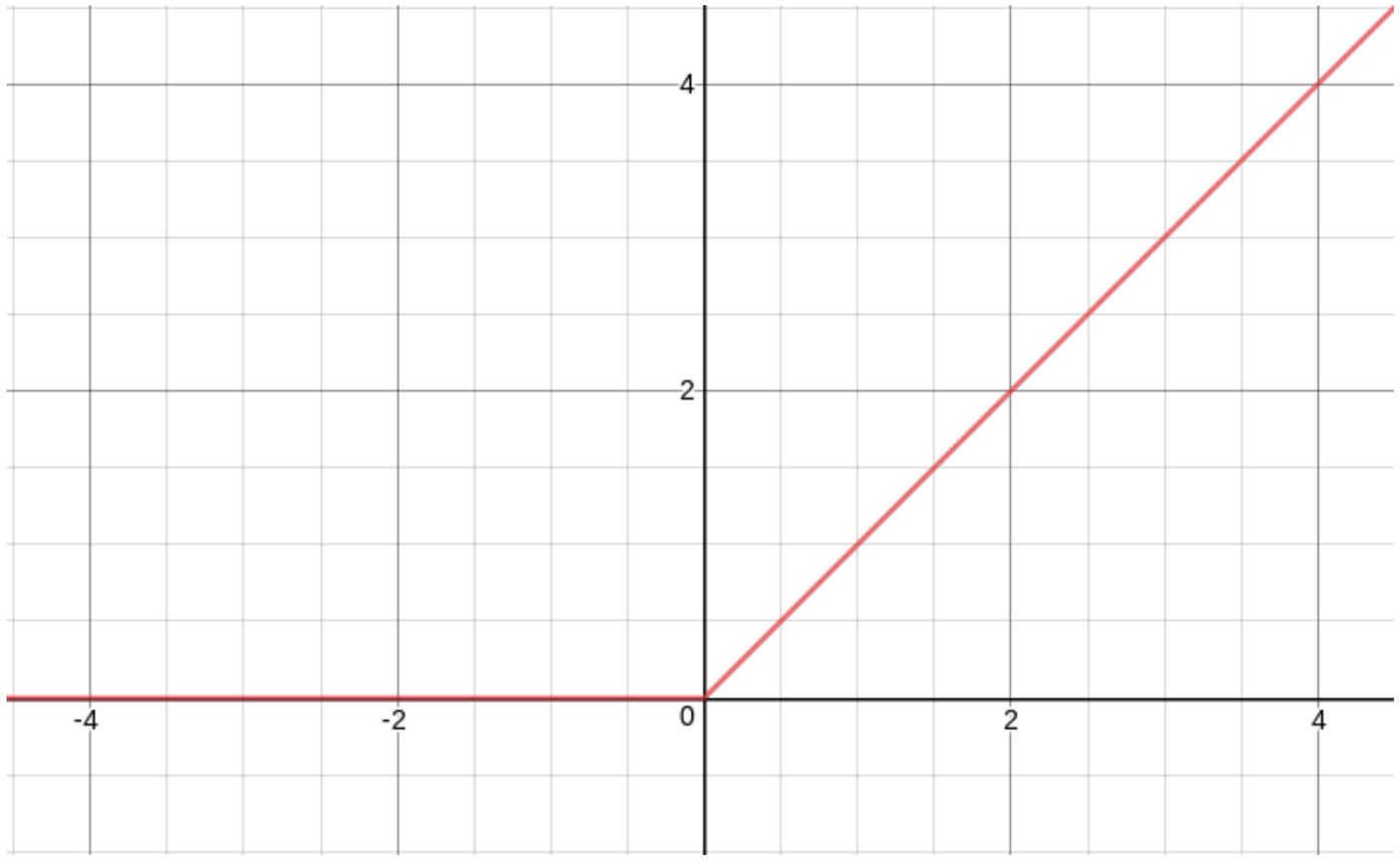



$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

# The 1980s: Multilayer Perceptron

## Multilayer Perceptron: activation functions

$$\mathbf{a}^{(l+1)} = \sigma(W^{(l)}\mathbf{a}^{(l)})$$




$$\mathrm{ReLU}(x) = \max\{0, x\} = x_+$$

# The 1980s: Multilayer Perceptron

Multilayer Perceptron: activation functions

$$\mathbf{a}^{(l+1)} = \sigma(W^{(l)}\mathbf{a}^{(l)})$$

Softmax activation functions:

$$\text{softmax}(\mathbf{x}) = \left( \frac{\exp(x_1)}{\sum_{j=1}^{d} \exp(x_j)}, \ldots, \frac{\exp(x_d)}{\sum_{j=1}^{d} \exp(x_j)} \right)^T$$

- A generalization of sigmoid from binary classification to multiple class classification
- Smooth approximation of the argmax function

# The 1980s: Multilayer Perceptron

Backpropagation: an algorithm to compute the gradient in MLP

Stochastic Gradient Descent:

- Perform weight updates after observing each instance: $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$

    1. Initialization of model parameter
    2. For j = 1 to T:
        1. For each training example $(\mathbf{x}_i, y_i)$

            Compute stochastic gradient $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i; \theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

            Update model parameter $\theta \leftarrow \theta - \gamma \Delta_i$

    Stop until convergence

- Key components:
    - Loss function $\ell(f(\mathbf{x}; \theta), y)$
    - A procedure to compute gradient $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i; \theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

# The 1980s: Multilayer Perceptron

Backpropagation: an algorithm to compute the gradient in MLP

Loss function for multi-class classification

Softmax output layer of neural networks for classification problems

$$f_c(\mathbf{x}; \theta) = \Pr(y = c \mid \mathbf{x})$$

Maximize the log-probability of the correct class given an input: $\log \Pr(y = c \mid \mathbf{x})$

Equivalently, this is the same as minimizing the negative log-likelihood of the data under our model:

$$\ell(f(\mathbf{x}), y) = -\sum_c \mathbb{I}(y = c) \log \Pr(y = c \mid \mathbf{x}) = -\sum_c \mathbb{I}(y = c) \log f_c(\mathbf{x})$$

This loss function is usually called the cross-entropy loss function for multi-class classification problem

# The 1980s: Multilayer Perceptron

## Backpropagation: an algorithm to compute the gradient in MLP

- Consider a network with L hidden layers:
  - Pre-activation for k > 0 layer:

$$\mathbf{a}^{(k+1)} = W^{(k)}\mathbf{h}^{(k)} + \mathbf{b}^{(k)}$$

  - Hidden layer activation for k > 0 layer:

$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{a}^{(k)}(\mathbf{x}))$$

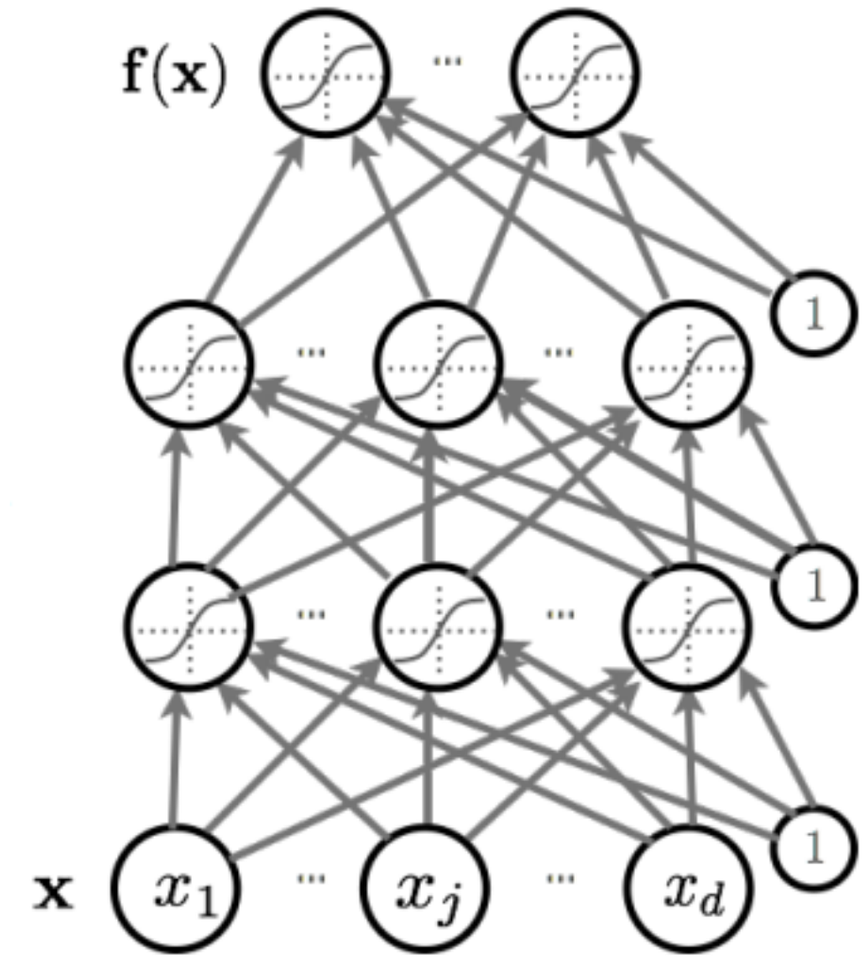
  - First layer k = 0:

$$\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$$

  - Last layer k = L:

$$\mathbf{h}^{(L)}(\mathbf{x}) = f(\mathbf{x}) = \sigma(\mathbf{a}^{(L)}(\mathbf{x}))$$

$\sigma(\cdot)$ is the softmax activation function.



# The 1980s: Multilayer Perceptron

Backpropagation: an algorithm to compute the gradient in MLP

Key idea: Chain rule

$$\ell(f(\mathbf{x}), y) = -\sum_c \mathbb{I}(y = c) \log \Pr(y = c \mid \mathbf{x}) = -\sum_c \mathbb{I}(y = c) \log f_c(\mathbf{x})$$

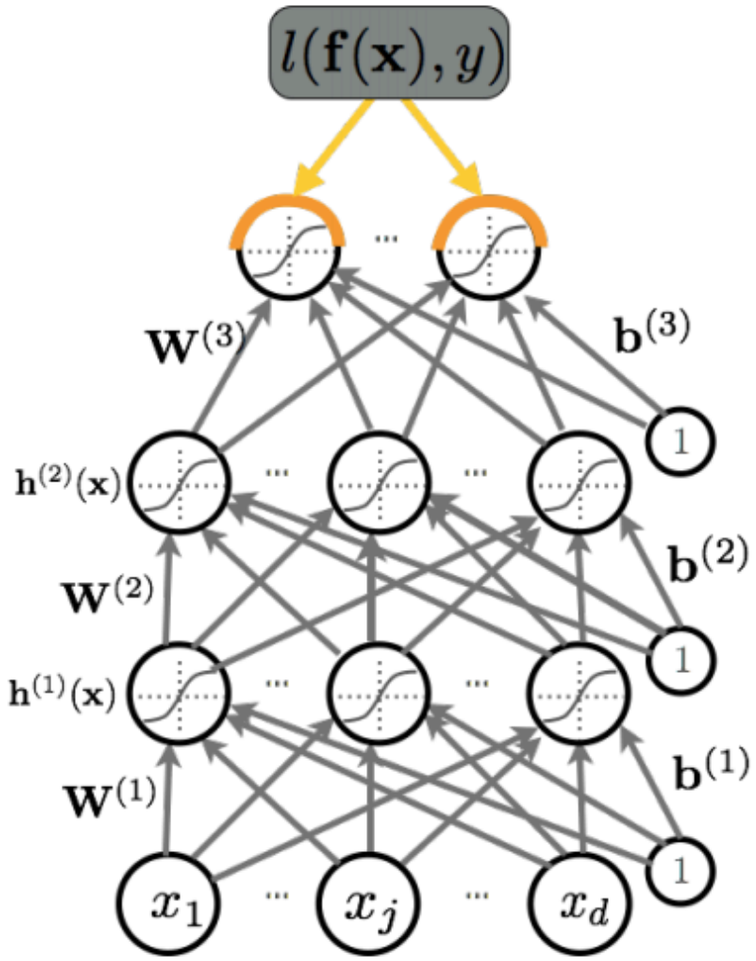
Partial derivative w.r.t. output neuron

$$\frac{\partial \ell(f(\mathbf{x}), y)}{\partial f_c(\mathbf{x})} = -\frac{\mathbb{I}(y=c)}{f_c(\mathbf{x})}$$

Put everything into matrix form:

$$\nabla_{f(\mathbf{x})} \ell = -\begin{pmatrix} \mathbb{I}(y = 1)/f_1(\mathbf{x}) \\ \vdots \\ \mathbb{I}(y = C)/f_C(\mathbf{x}) \end{pmatrix}$$

$$= -\frac{1}{f_y(\mathbf{x})} \begin{pmatrix} \mathbb{I}(y = 1) \\ \vdots \\ \mathbb{I}(y = C) \end{pmatrix} = -\frac{\mathbf{e}(y)}{f_y(\mathbf{x})}$$



# The 1980s: Multilayer Perceptron

Backpropagation: an algorithm to compute the gradient in MLP

Key idea: Chain rule

$$\ell(f(\mathbf{x}), y) = -\sum_c \mathbb{I}(y = c) \log \Pr(y = c \mid \mathbf{x}) = -\sum_c \mathbb{I}(y = c) \log f_c(\mathbf{x})$$

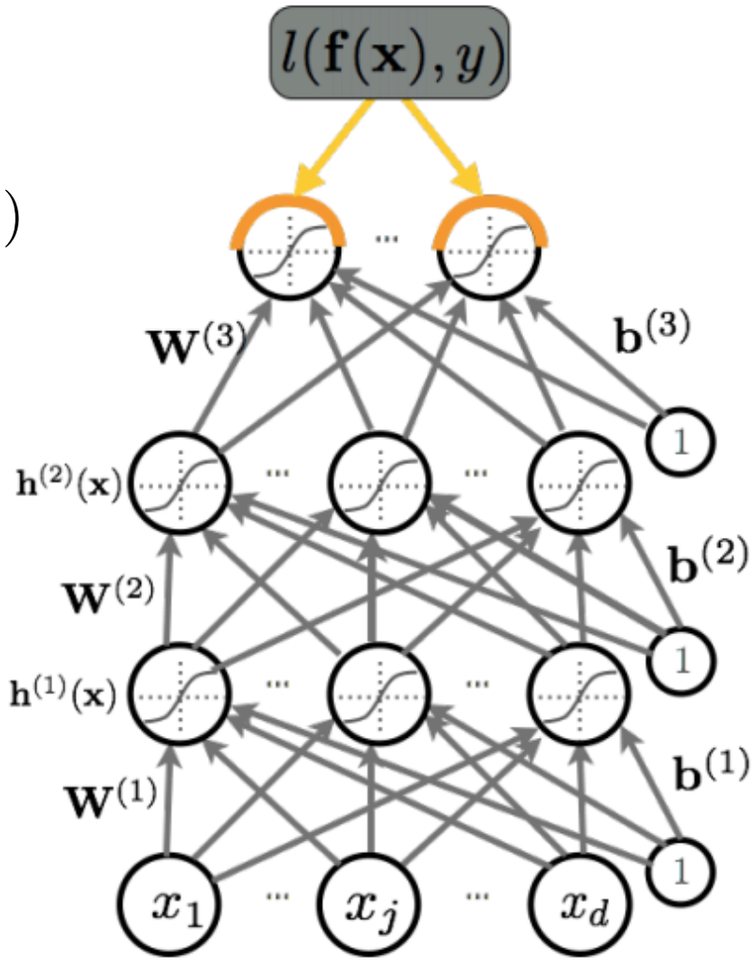
Partial derivative w.r.t. pre-activation of output neuron

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_c^{(L)}(\mathbf{x})} = -\frac{\partial \log f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} = -\frac{1}{f_y(\mathbf{x})} \frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} = -\left(\mathbb{I}(y = c) - f_c(\mathbf{x})\right)$$

Put everything into matrix form:

$$\nabla_{a^{(L)}(\mathbf{x})} \ell = -(\mathbf{e}(y) - f(\mathbf{x}))$$

Think for 2 mins, why?



# The 1980s: Multilayer Perceptron

Backpropagation: an algorithm to compute the gradient in MLP

Proof:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_c^{(L)}(\mathbf{x})} = \frac{\partial \ell(f(\mathbf{x}),y)}{\partial f_y(\mathbf{x})} \frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} = -\frac{1}{f_y(x)} \frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})}$$

$$\begin{aligned}
\frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} &= \frac{\partial}{\partial a_c^{(L)}} \left( \frac{\exp(a_y^{(L)}(\mathbf{x}))}{\sum_{c'} \exp(a_{c'}^{(L)}(\mathbf{x}))} \right) \\
&= \frac{\mathbb{I}(y=c)\exp(a_y^{(L)}(\mathbf{x}))\left(\sum_{c'} \exp(a_{c'}^{(L)}(\mathbf{x}))\right) - \exp(a_c^{(L)}(\mathbf{x}))\exp(a_y^{(L)}(\mathbf{x}))}{\left(\sum_{c'} \exp(a_{c'}^{(L)}(\mathbf{x}))\right)^2} \\
&= \mathbb{I}(y=c) f_y(\mathbf{x}) - f_c(\mathbf{x}) f_y(\mathbf{x})
\end{aligned}$$

$$\Rightarrow \quad \frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_c^{(L)}(\mathbf{x})} = -(\mathbb{I}(y=c) - f_c(\mathbf{x}))$$

# The 1980s: Multilayer Perceptron

Backpropagation: an algorithm to compute the gradient in MLP

Key idea: Chain rule

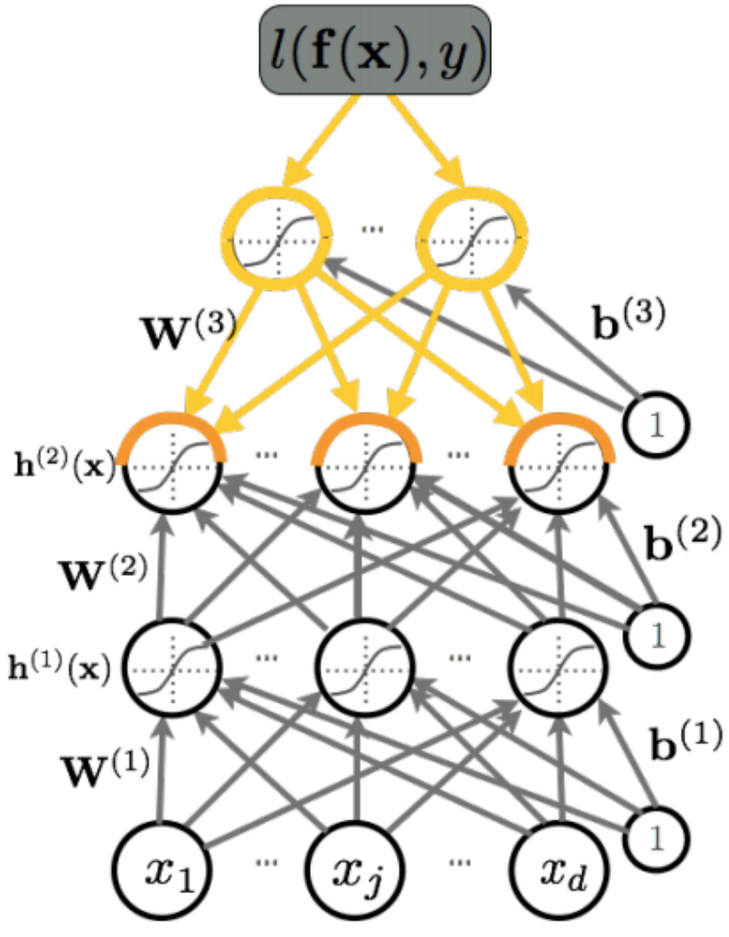
$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{a}^{(k)}(\mathbf{x})) \qquad \mathbf{a}^{(k+1)} = W^{(k)}\mathbf{h}^{(k)} + \mathbf{b}^{(k)}$$

Partial derivative w.r.t. intermediate neuron at layer k:

$$\begin{aligned}\frac{\partial \ell(f(\mathbf{x}, y))}{\partial h_i^{(k)}(\mathbf{x})} &= \sum_j \frac{\partial \ell(f(\mathbf{x}), y)}{\partial a_j^{(k+1)}(\mathbf{x})} \frac{\partial a_j^{(k+1)}(\mathbf{x})}{\partial h_i^{(k)}(\mathbf{x})} \\ &= \sum_j \nabla_{a_j^{(k+1)}} \ell \cdot W_{ji} \\ &= \left(W^T \nabla_{a^{(k+1)}} \ell\right)_i\end{aligned}$$

Put everything into matrix form:

$$\nabla_{h^{(k)}(\mathbf{x})} \ell = W^T \nabla_{a^{(k+1)}} \ell$$



# The 1980s: Multilayer Perceptron

Backpropagation: an algorithm to compute the gradient in MLP

Key idea: Chain rule

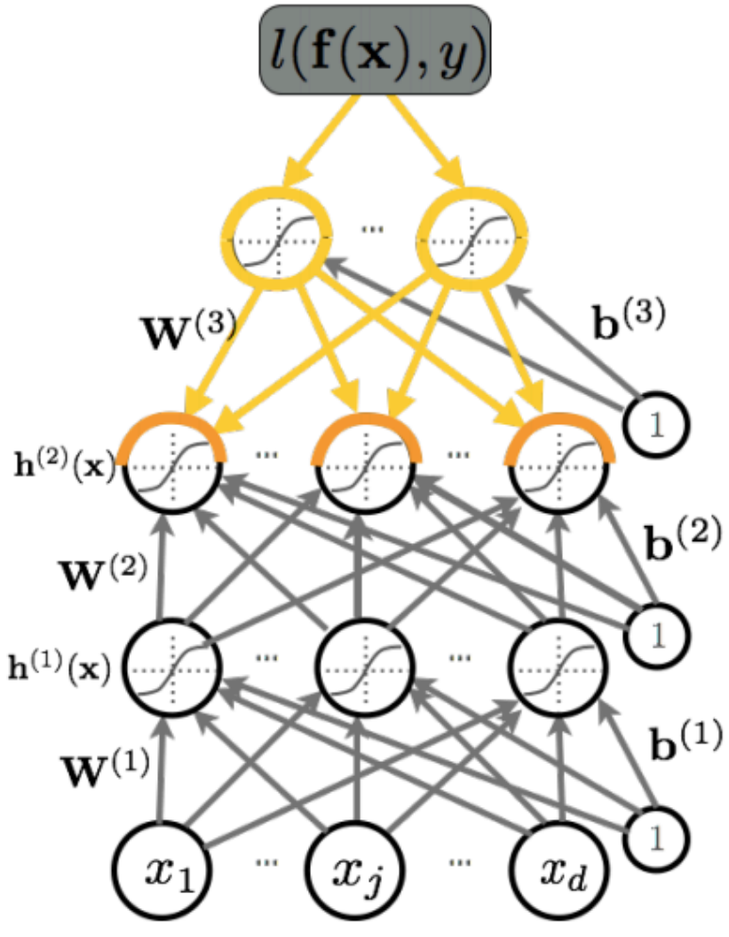
$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{a}^{(k)}(\mathbf{x})) \qquad \mathbf{a}^{(k+1)} = W^{(k)}\mathbf{h}^{(k)} + \mathbf{b}^{(k)}$$

Partial derivative w.r.t. pre-activation of intermediate neuron at layer k:

$$\frac{\partial \ell(f(\mathbf{x},y))}{\partial a_i^{(k)}(\mathbf{x})} = \frac{\partial \ell(f(\mathbf{x},y))}{\partial h_i^{(k)}(\mathbf{x})} \frac{\partial h_i^{(k)}(\mathbf{x})}{\partial a_i^{(k)}(\mathbf{x})} = \nabla_{h_i^{(k)}(\mathbf{x})} \ell \cdot g'(a_i^{(k)}(\mathbf{x}))$$

Put everything into matrix form:

$$\nabla_{a^{(k)}(\mathbf{x})} \ell = \nabla_{h^{(k)}(\mathbf{x})} \ell \odot g'(a^{(k)}(\mathbf{x}))$$



# The 1980s: Multilayer Perceptron

Given all these, how to compute the gradient?

Key idea: Chain rule

$$\mathbf{a}^{(k+1)} = W^{(k)}\mathbf{h}^{(k)} + \mathbf{b}^{(k)}$$

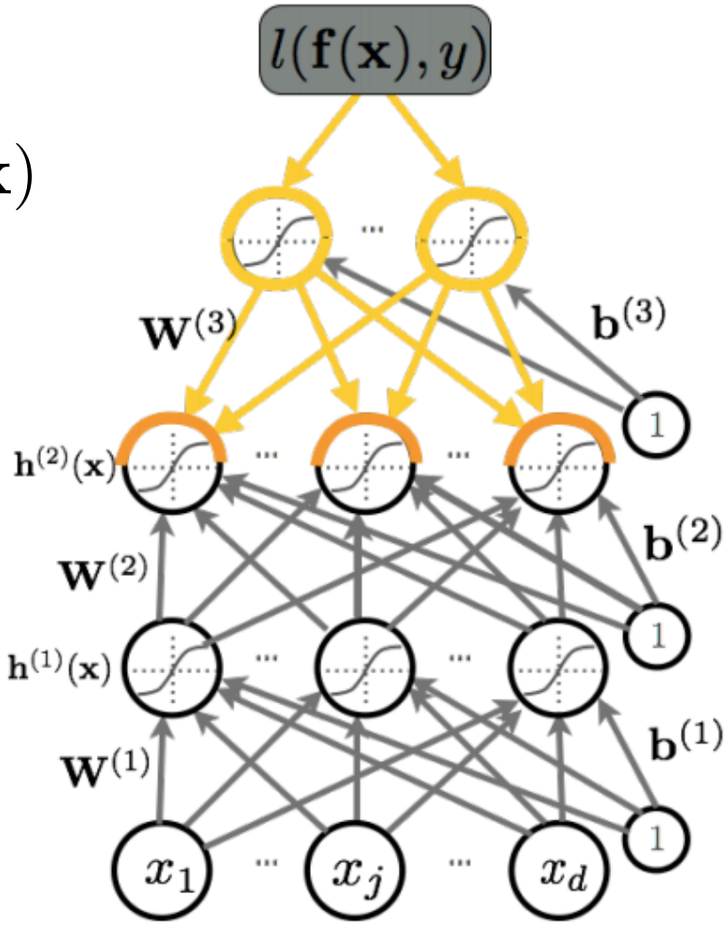
Partial derivative w.r.t. model weight at layer k:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial W_{ij}^{(k)}} = \frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_i^{(k+1)}(\mathbf{x})} \frac{\partial a_i^{(k+1)}(\mathbf{x})}{\partial W_{ij}^{(k)}} = \nabla_{a_i^{(k+1)}(\mathbf{x})} \ell \cdot h_j^{(k)}(\mathbf{x})$$

Put everything into matrix form:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial W^{(k)}} = \nabla_{a^{(k+1)}(\mathbf{x})} \ell \cdot h^{(k)^T}(\mathbf{x})$$

Basically, outer product of forward evaluation signal and backward differentiation signal



# The 1980s: Multilayer Perceptron

Given all these, how to compute the gradient?

Key idea: Chain rule

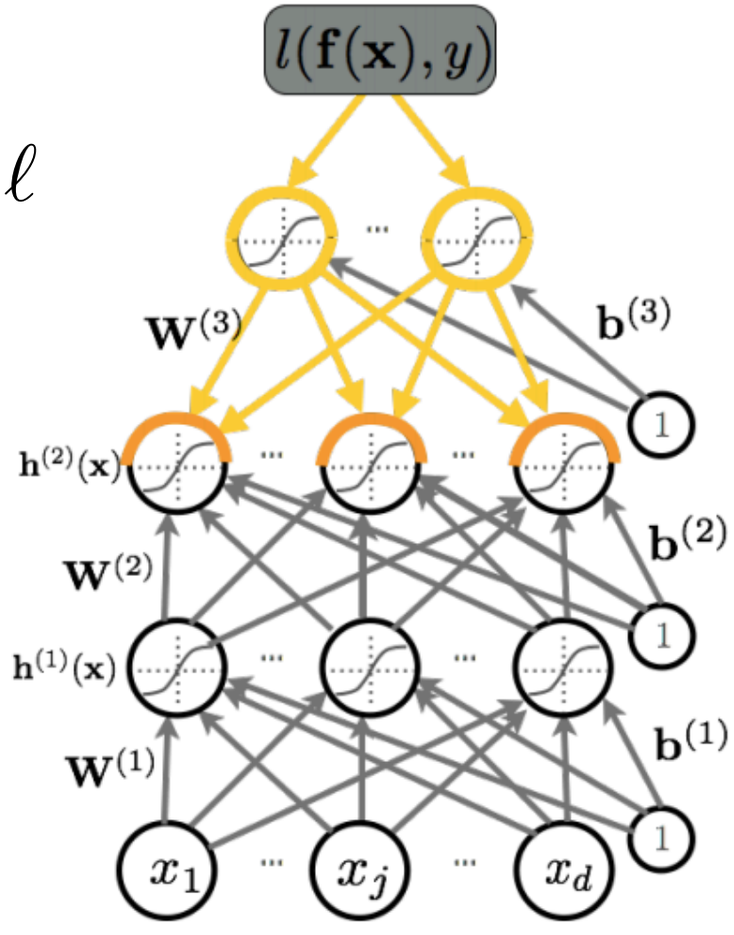
$$\mathbf{a}^{(k+1)} = W^{(k)}\mathbf{h}^{(k)} + \mathbf{b}^{(k)}$$

Partial derivative w.r.t. intercept weight at layer k:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial b_i^{(k)}} = \frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_i^{(k+1)}(\mathbf{x})} \frac{\partial a_i^{(k+1)}(\mathbf{x})}{\partial b_i^{(k)}} = \nabla_{a_i^{(k+1)}(\mathbf{x})} \ell$$

Put everything into matrix form:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial b^{(k)}} = \nabla_{a^{(k+1)}(\mathbf{x})} \ell$$



# The 1980s: Multilayer Perceptron

## SGD algorithm for training MLP:

- Perform forward propagation
- Compute gradient at the output softmax layer (pre-activation):

$$\nabla_{a^{(L)}(\mathbf{x})}\ell = -(\mathbf{e}(y) - f(\mathbf{x}))$$

- For k = L to 1
- Compute gradient w.r.t. the model parameters at layer k:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial W^{(k)}} = \nabla_{a^{(k+1)}(\mathbf{x})}\ell \cdot {h^{(k)}}^T(\mathbf{x}) \qquad \frac{\partial \ell(f(\mathbf{x}),y)}{\partial b^{(k)}} = \nabla_{a^{(k+1)}(\mathbf{x})}\ell$$

- Compute the gradient w.r.t. the intermediate neuron at layer k:

$$\nabla_{h^{(k)}(\mathbf{x})}\ell = W^T \nabla_{a^{(k+1)}}\ell$$

- Compute the gradient w.r.t. the pre-activation of the intermediate neuron at layer k:

$$\nabla_{a^{(k)}(\mathbf{x})}\ell = \nabla_{h^{(k)}(\mathbf{x})}\ell \odot g'(a^{(k)}(\mathbf{x}))$$

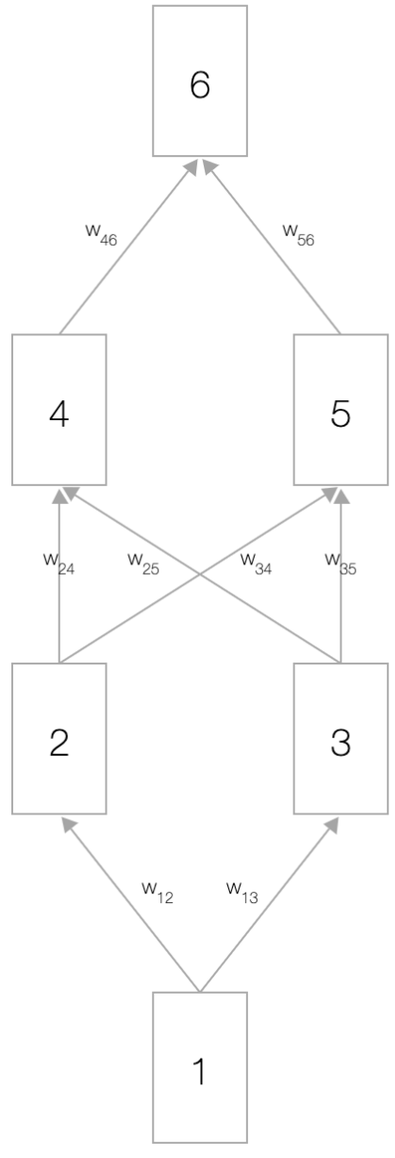
# The 1980s: Multilayer Perceptron

Let's consider a simple example to illustrate the BP algorithm:

## Simple neural network

On the right, you see a neural network with one input, one output node and two hidden layers of two nodes each.

Nodes in neighboring layers are connected with weights $w_{ij}$, which are the network parameters.



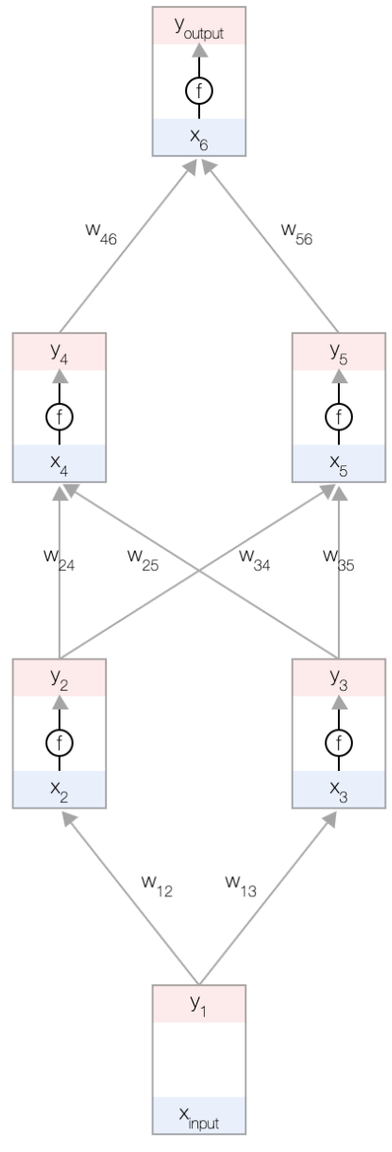
# The 1980s: Multilayer Perceptron

Let's consider a simple example to illustrate the BP algorithm:

## Activation function

Each node has a total input $x$, an activation function $f(x)$ and an output $y = f(x)$. $f(x)$ has to be a non-linear function, otherwise the neural network will only be able to learn linear models.

A commonly used activation function is the Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$.



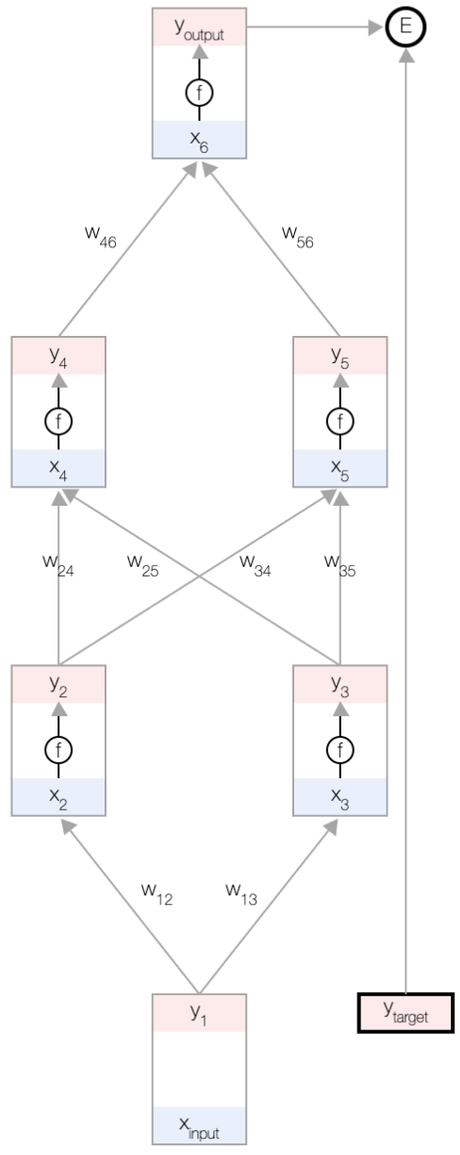
# The 1980s: Multilayer Perceptron

Let's consider a simple example to illustrate the BP algorithm:

## Error function

The goal is to learn the weights of the network automatically from data such that the predicted output $y_{output}$ is close to the target $y_{target}$ for all inputs $x_{input}$.

To measure how far we are from the goal, we use an error function $E$. A commonly used error functon is $E(y_{output}, y_{target}) = \frac{1}{2}(y_{output} - y_{target})^2$.



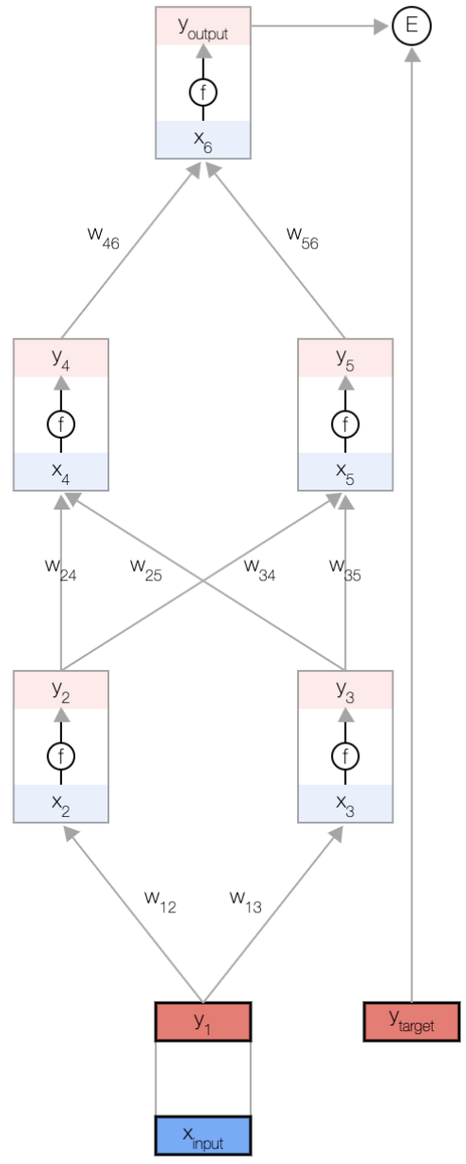
# The 1980s: Multilayer Perceptron

Let's consider a simple example to illustrate the BP algorithm:

## Forward propagation

We begin by taking an input example $(x_{input}, y_{target})$ and updating the input layer of the network.

For consistency, we consider the input to be like any other node but without an activation function so its output is equal to its input, i.e. $y_1 = x_{input}$.



# The 1980s: Multilayer Perceptron

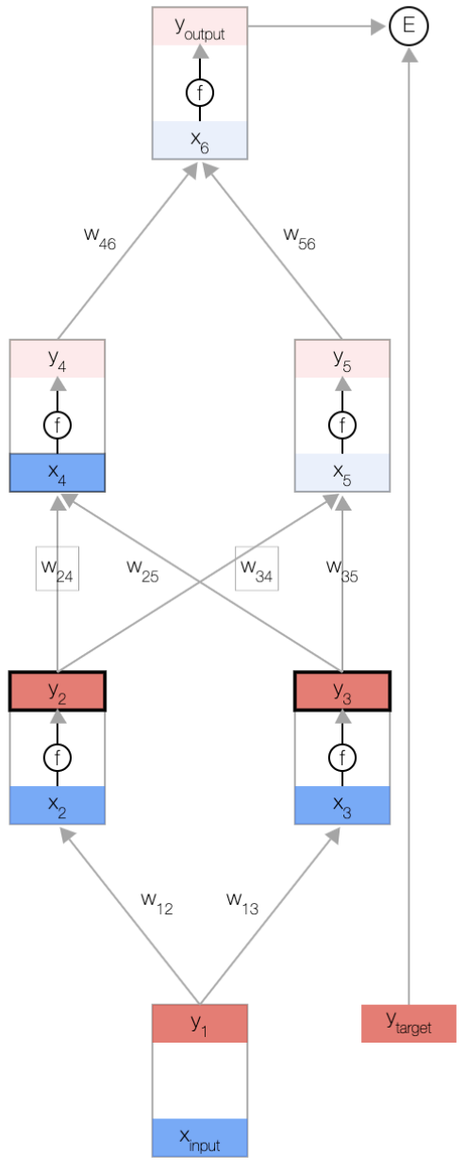
Let's consider a simple example to illustrate the BP algorithm:

## Forward propagation

Using these 2 formulas we propagate for the rest of the network and get the final output of the network.

$$y = f(x)$$

$$x_j = \sum_{i \in in(j)} w_{ij} y_i + b_j$$



# The 1980s: Multilayer Perceptron

Let's consider a simple example to illustrate the BP algorithm:

## Error derivative

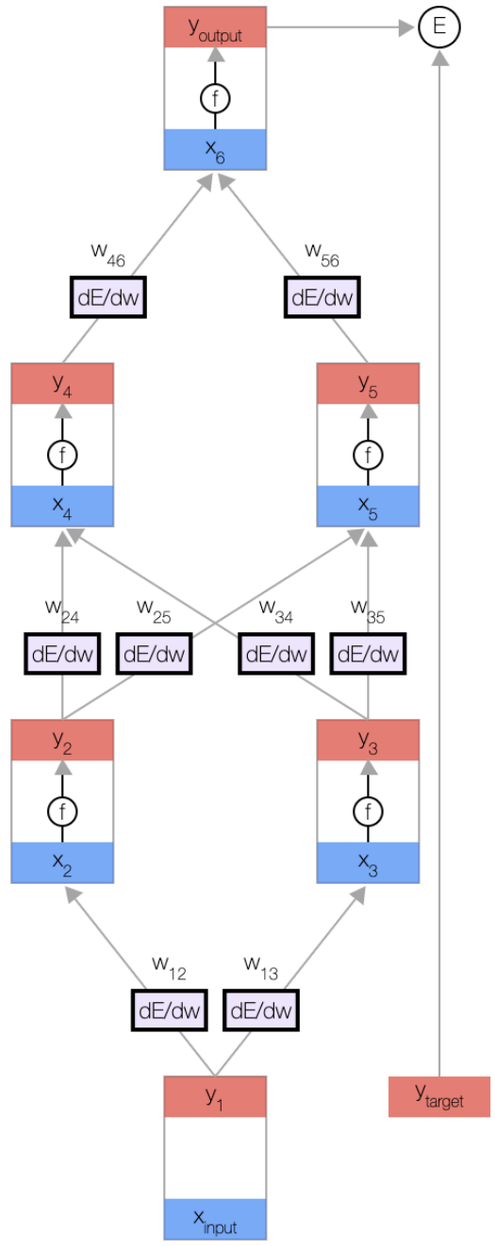
The backpropagation algorithm decides how much to update each weight of the network after comparing the predicted output with the desired output for a particular example. For this, we need to compute how the error changes with respect to each weight $\frac{dE}{dw_{ij}}$.

Once we have the error derivatives, we can update the weights using a simple update rule:

$$w_{ij} = w_{ij} - \alpha \frac{dE}{dw_{ij}}$$

where $\alpha$ is a positive constant, referred to as the learning rate, which we need to fine-tune empirically.

[Note] The update rule is very simple: if the error goes down when the weight increases ($\frac{dE}{dw_{ij}} < 0$), then increase the weight, otherwise if the error goes up when the weight increases ($\frac{dE}{dw_{ij}} > 0$), then decrease the weight.



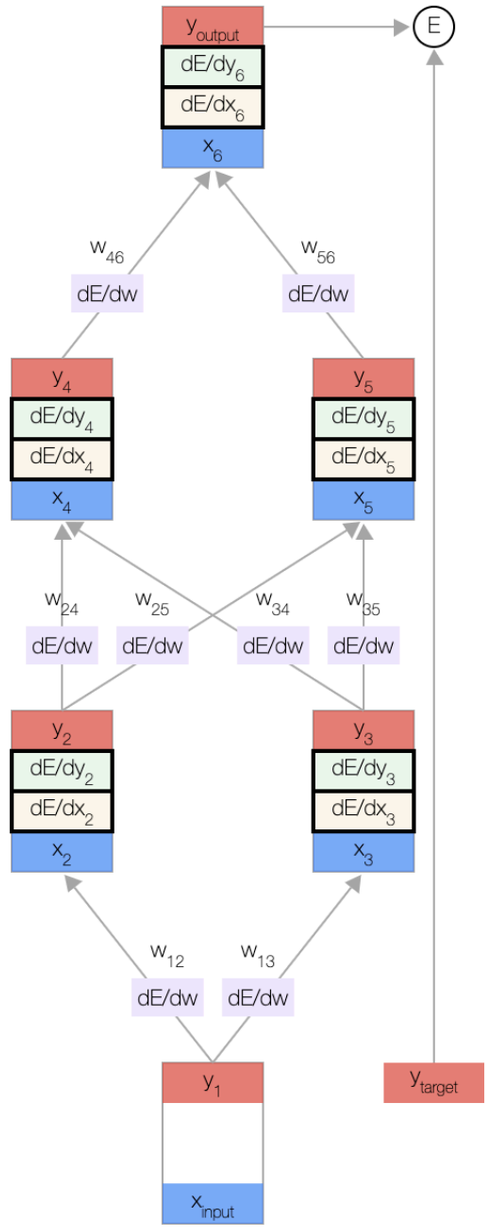
# The 1980s: Multilayer Perceptron

Let's consider a simple example to illustrate the BP algorithm:

## Additional derivatives

To help compute $\frac{dE}{dw_{ij}}$, we additionally store for each node two more derivatives: how the error changes with:

- the total input of the node $\frac{dE}{dx}$ and
- the output of the node $\frac{dE}{dy}$.



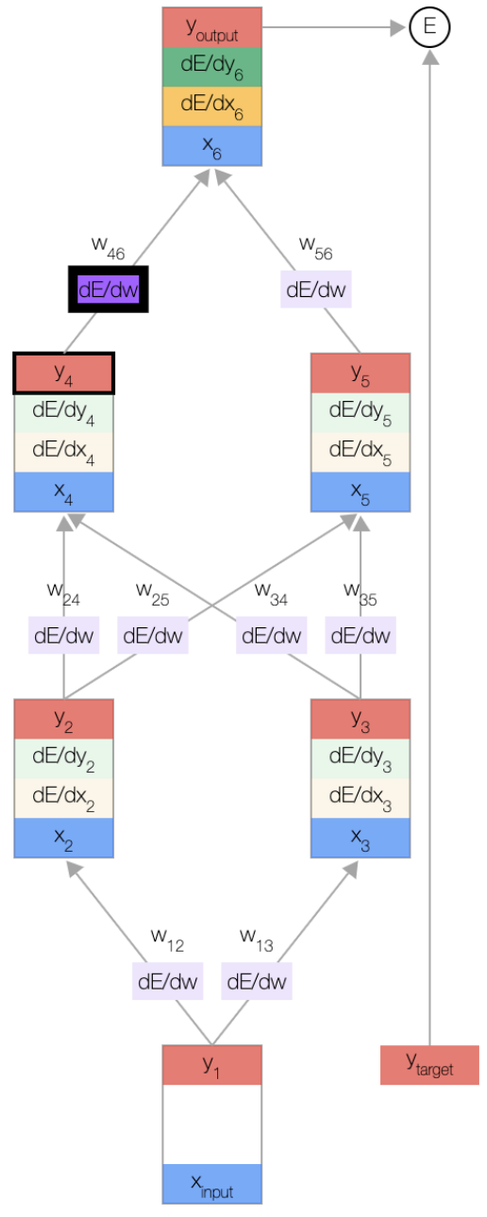
# The 1980s: Multilayer Perceptron

Let’s consider a simple example to illustrate the BP algorithm:

## Back propagation

As soon as we have the error derivative with respect to the total input of a node, we can get the error derivative with respect to the weights coming into that node.

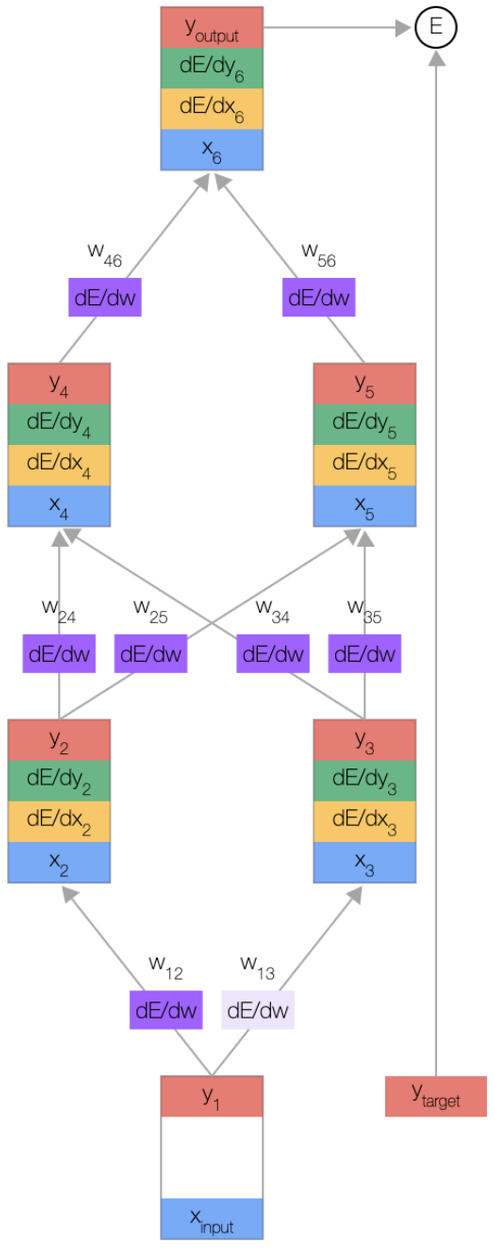
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}} \frac{\partial E}{\partial x_j} = y_i \frac{\partial E}{\partial x_j}$$



# The 1980s: Multilayer Perceptron

Let’s consider a simple example to illustrate the BP algorithm:

The end.



# Lecture 5: Feedforward Neural Networks

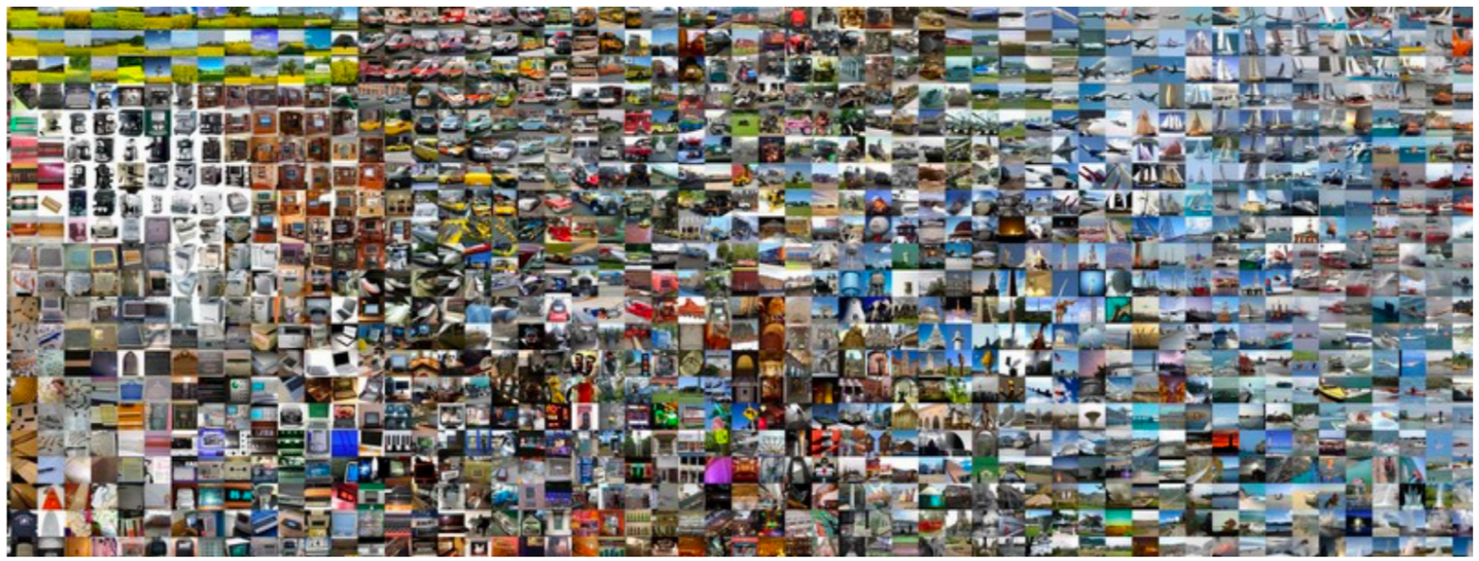
Overview:

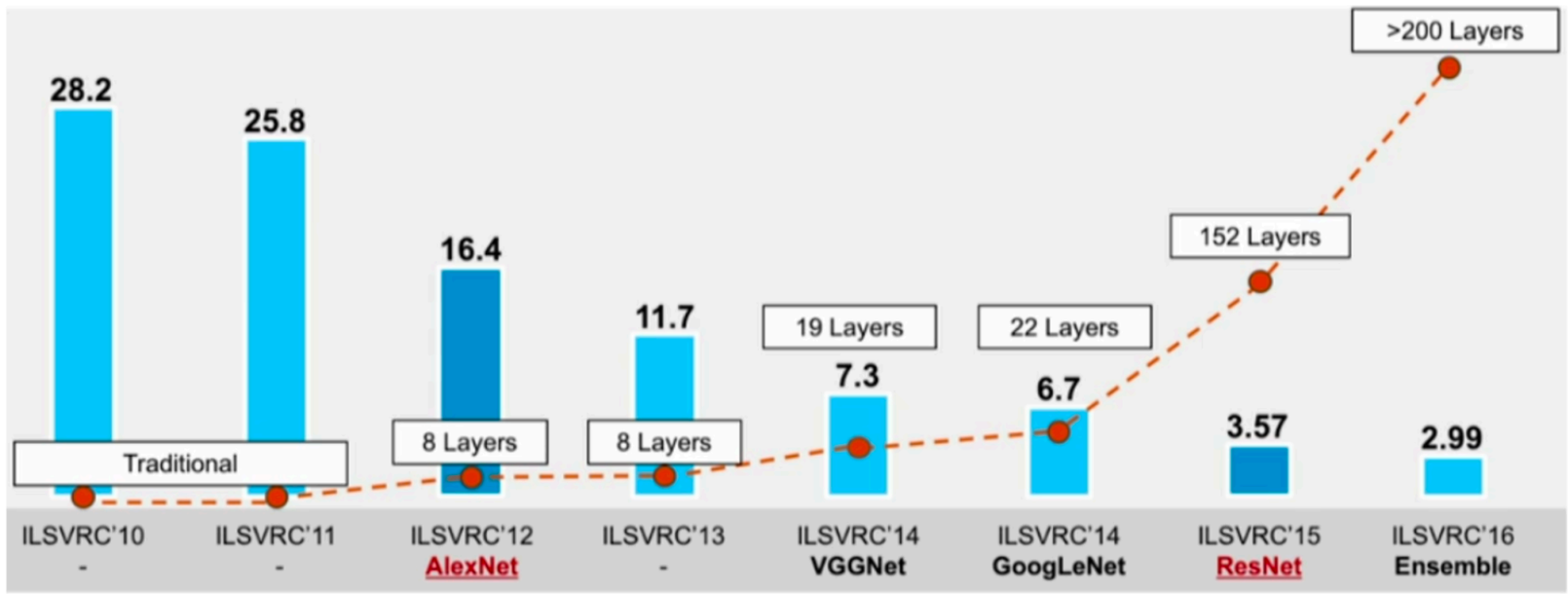
- The history of AI and Neural Networks
- The 1960s: Perceptron
- The 1980s: Multilayer Perceptron
- The 2010s: Deep Learning

# The 2010s: Deep Learning

So, what's the difference nowadays?

Large-scale datasets + Powerful computation

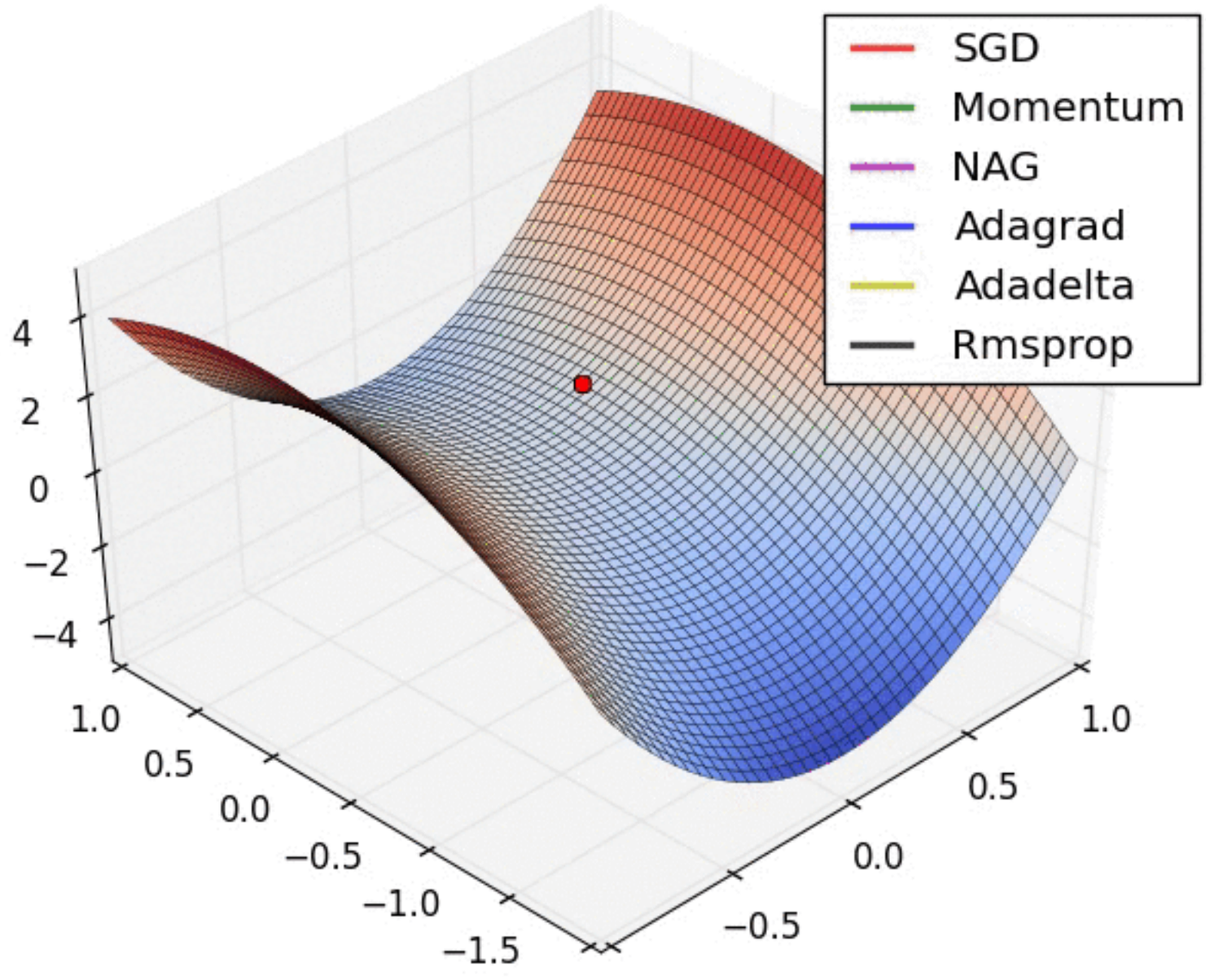




# The 2010s: Deep Learning

## Many practical tricks:

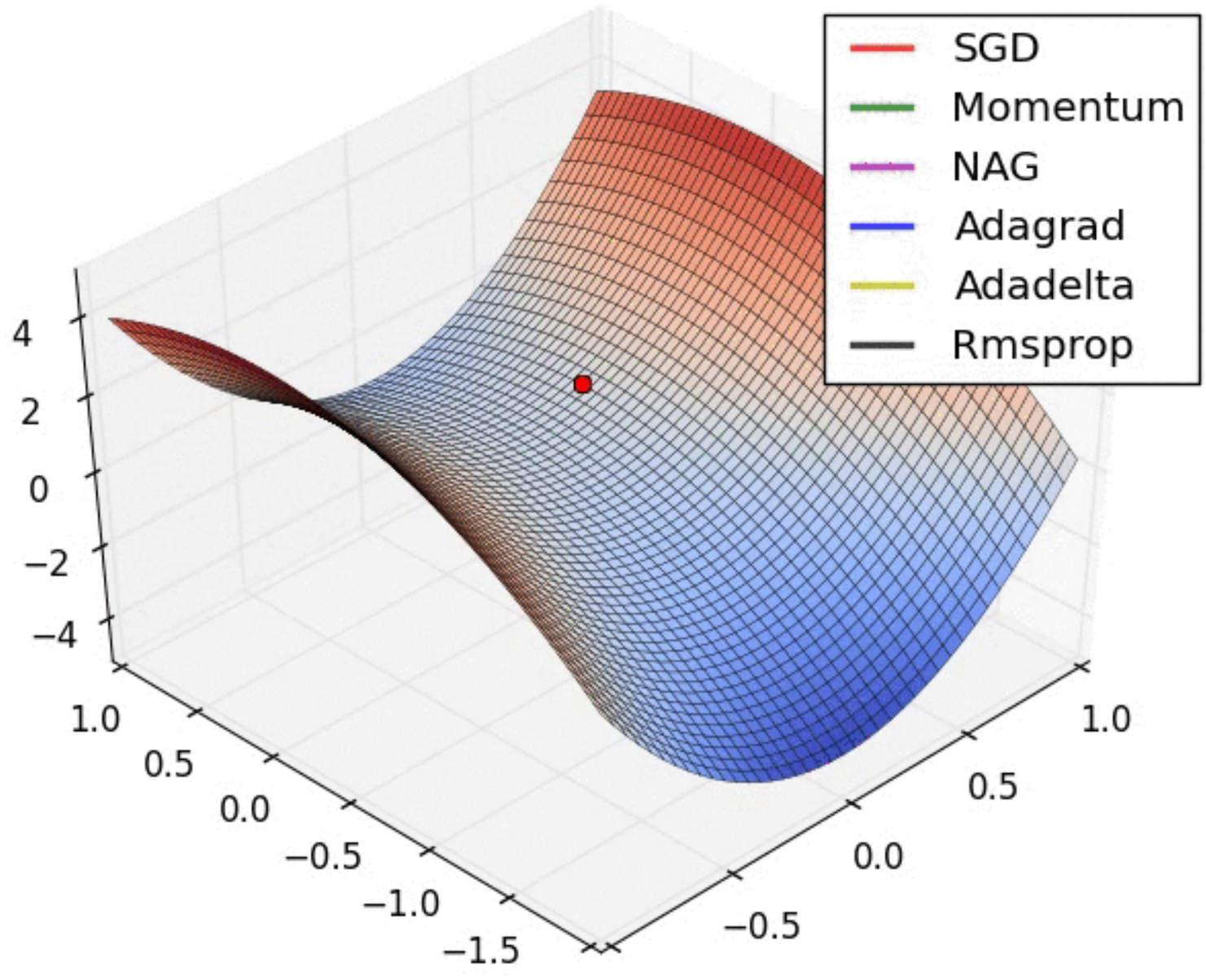
### Adaptive optimization algorithms



# The 2010s: Deep Learning

## Many practical tricks:

### Adaptive optimization algorithms



# The 2010s: Deep Learning

Many practical tricks:

Initialization

- Initialize all the intercepts (b) = 0
- Initialize model weights uniformly from $[-U, U]$

$$U = \frac{\sqrt{6}}{\sqrt{\text{fan\_in}+\text{fan\_out}}}$$

- fan_in = # neurons in the last layer
- fan_out = # neurons in the current layer

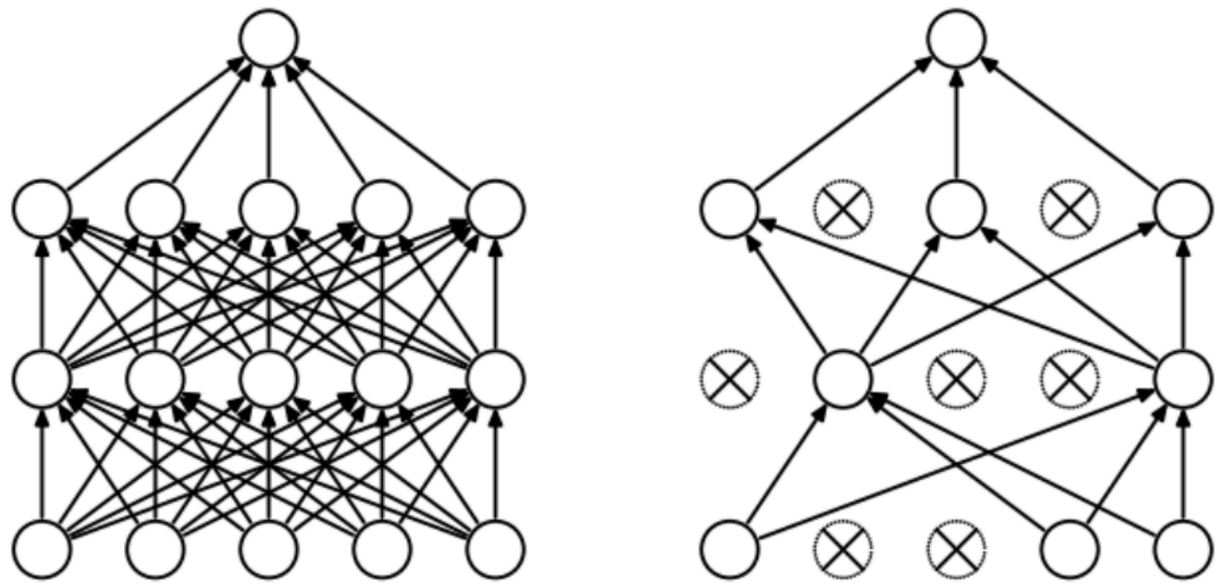
Initialization is very important for the success of training neural networks

# The 2010s: Deep Learning

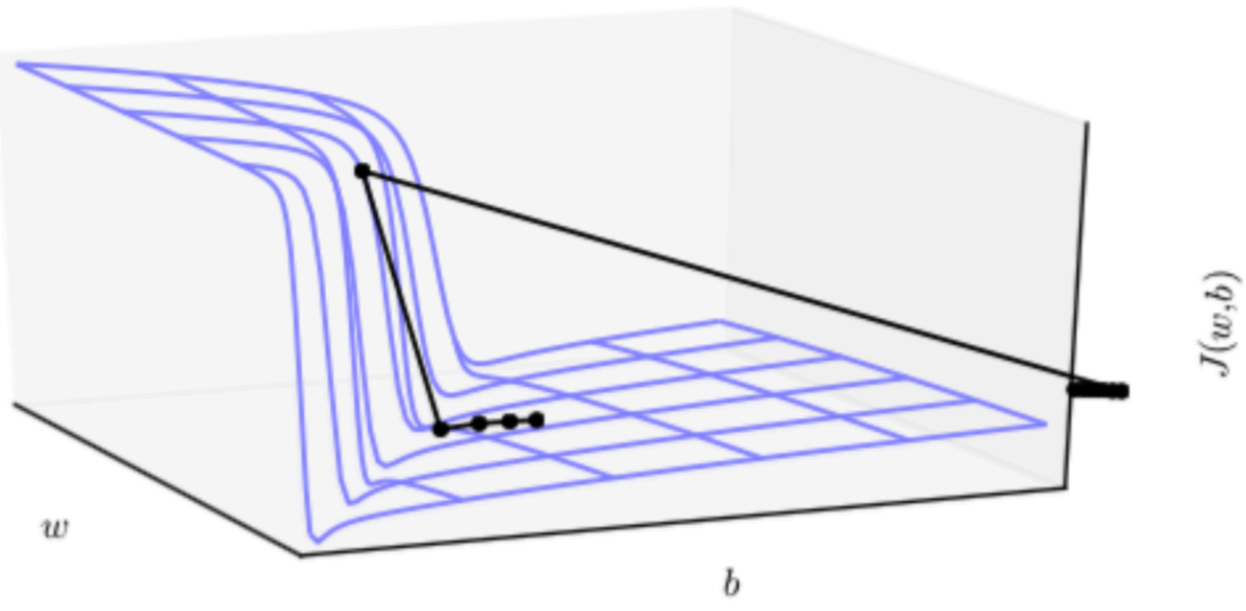
Many practical tricks:

Regularization

- L2 regularization
- Dropout
- Batch Normalization
- Gradient truncation for RNN
- ……



(a) Standard Neural Net (b) After applying dropout.



# The 2010s: Deep Learning

Geoffrey E. Hinton: One of the three 2018 A. M. Turing award laureates for his contributions in:

- Backpropagation (1986)
- Boltzmann machines (1983)
- Improvements to CNNs (2012)

# Lecture 5: Homework

Again, no maths HW. But,

- You have two days to implement a MLP for MNIST classification using numpy
- Three layer model: 784-200-10
- Hidden activation function: ReLU
- Output layer: softmax
- You should tune other parameters based on your implementation:
  - batch size
  - learning rate
  - regularization
  - etc