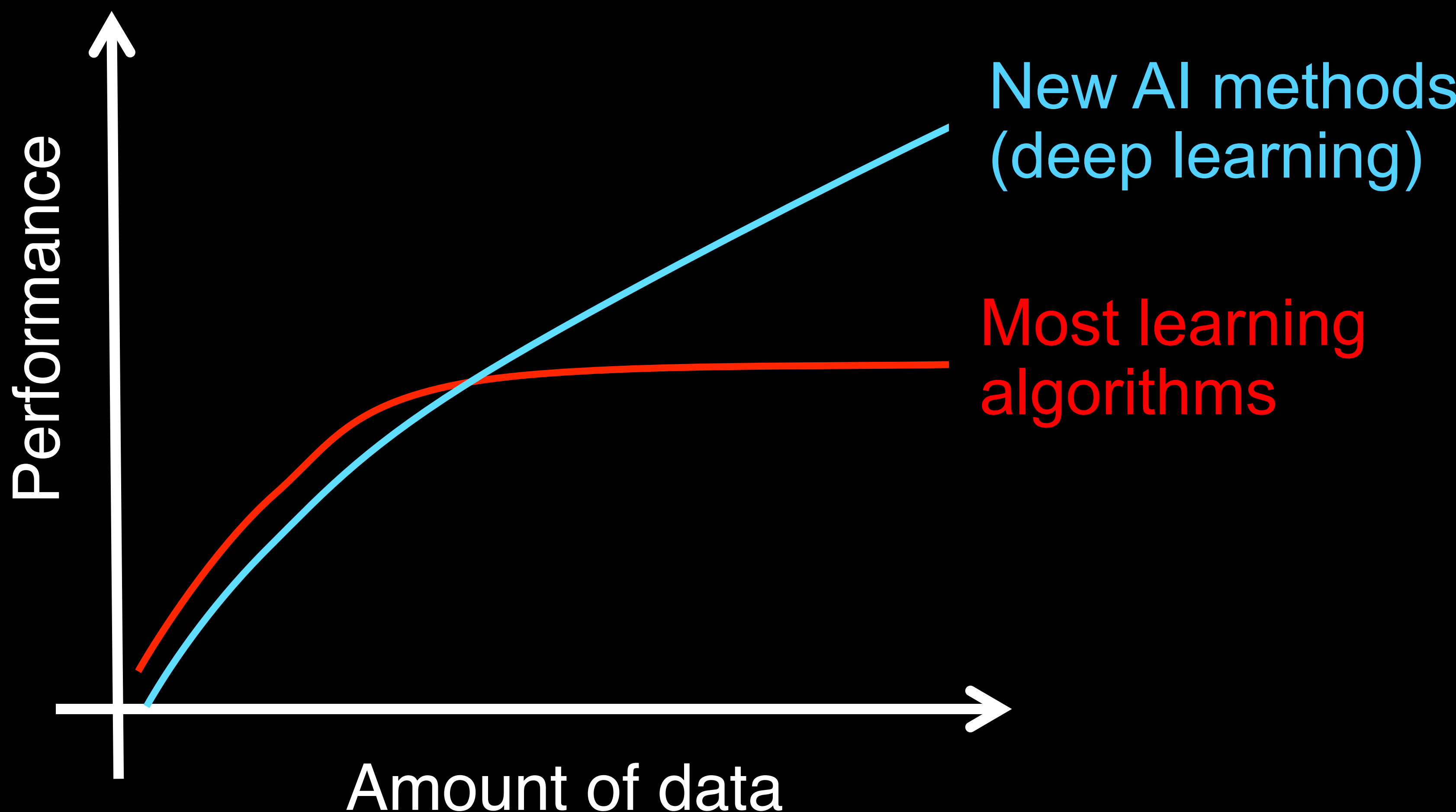
CS290D – Advanced Data Mining

Instructor: Xifeng Yan
Computer Science
University of California at Santa Barbara

Data and machine learning



Andrew Ng

The idea:

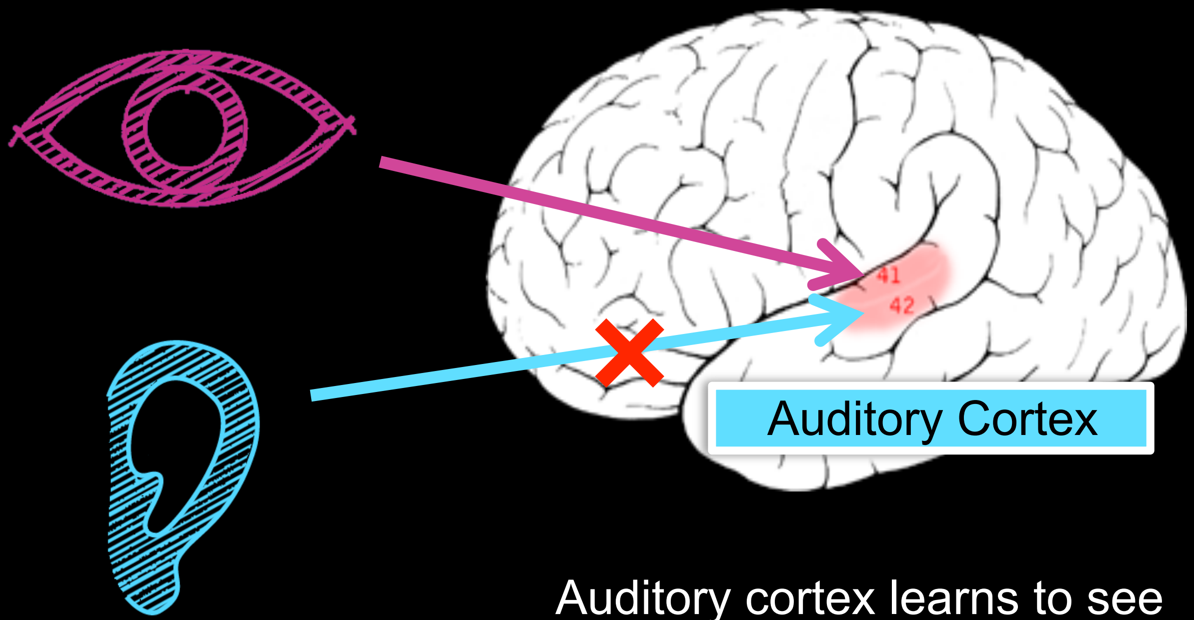
Most perception (input processing) in the brain may be due to one learning algorithm.

The idea:

Build learning algorithms
that mimic the brain.

Most of human intelligence may
be due to one learning algorithm.

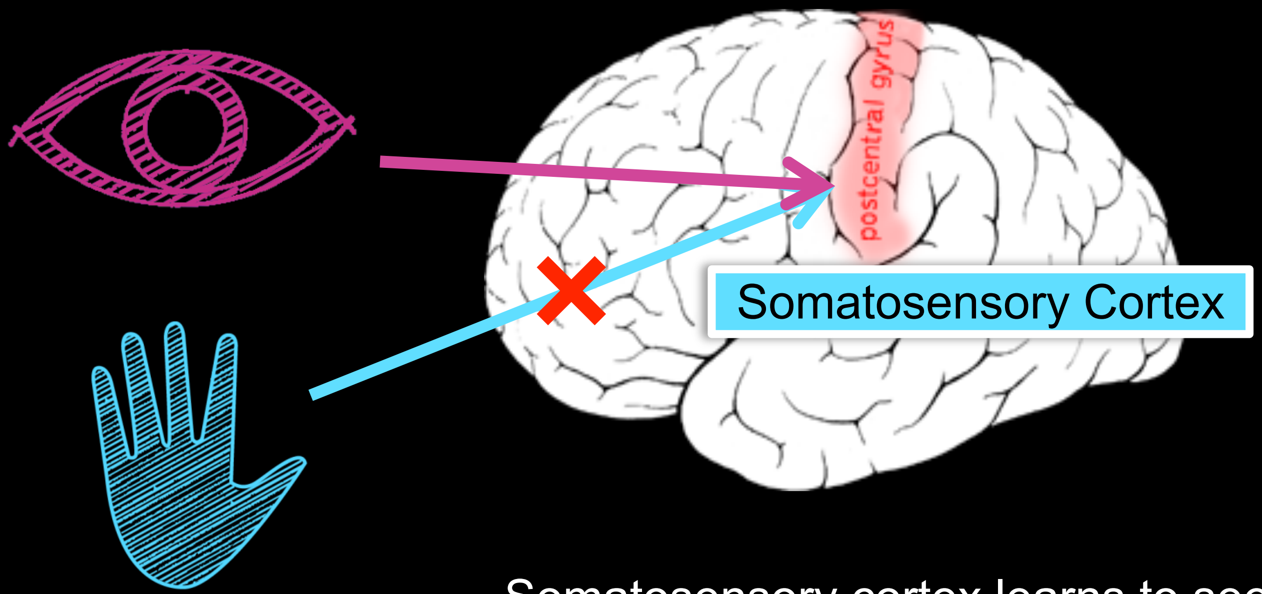
The “one learning algorithm” hypothesis



Auditory cortex learns to see

[Roe et al., 1992]

The "one learning algorithm" hypothesis



Somatosensory cortex learns to see

[Metin & Frost, 1989]

Success stories

Record performance

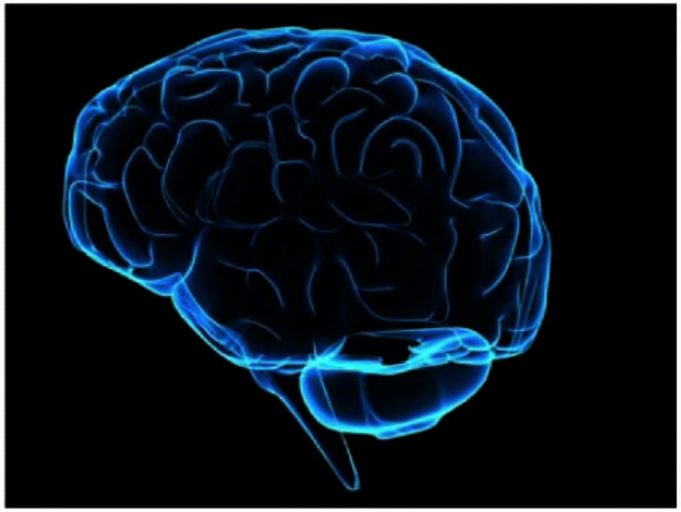
- MNIST (1988, 2003, 2012)
- ImageNet (since 2012) and Object Recognition
- …

Real applications

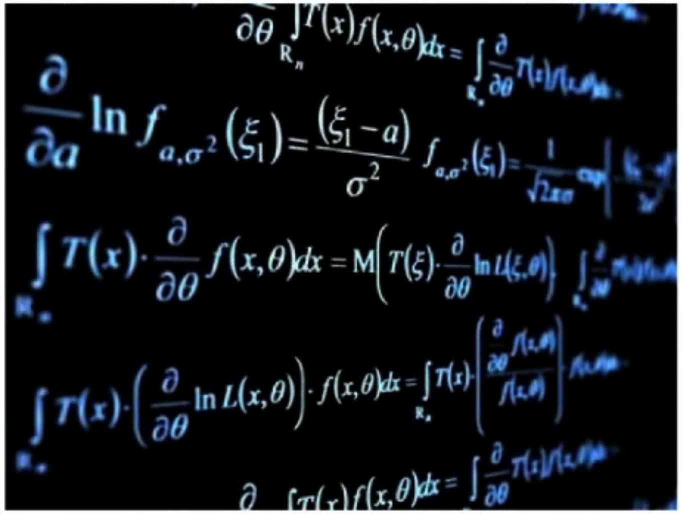
- Check reading (AT&T Bell Labs, 1995 – 2005)
- Optical character recognition (Microsoft OCR, 2000)
- Cancer detection from medical images (NEC, 2010)
- Object recognition (Google and Baidu’s photo taggers, 2013)
- Speech recognition (Microsoft, Google, IBM switched in 2012)
- Natural Language Processing (NEC 2010)
- …

How to design computers?

Biological computer



Mathematical computer

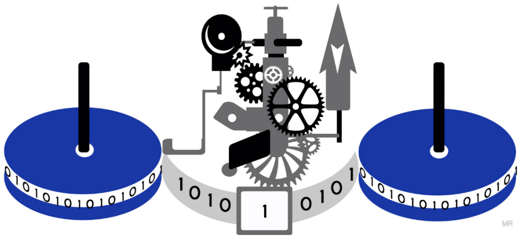


- Which model to emulate : brain or mathematical logic ?
- Mathematical logic won.

Computing with symbols

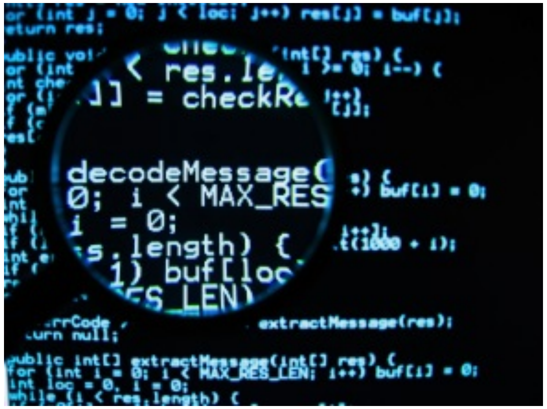
General computing machines

- Turing machine
- von Neumann machine



Engineering

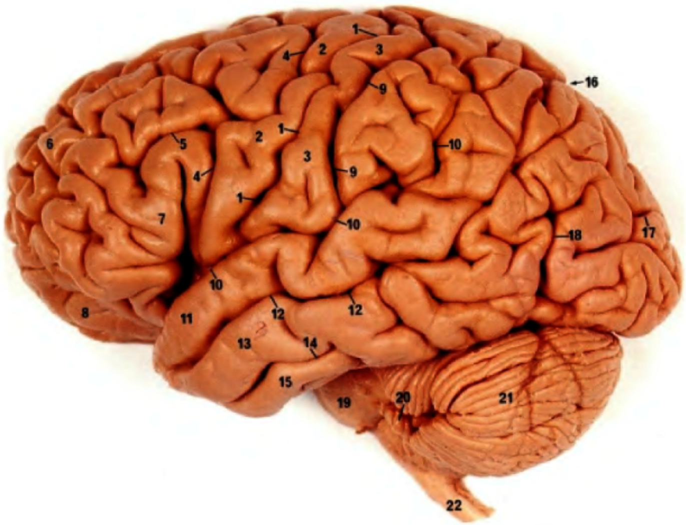
- Programming
  (reducing a complex task into
  a collection of simple tasks.)
- Computer language
- Debugging
- Operating systems
- Libraries



Computing with the brain

An engineering perspective

- Compact
- Energy efficient (20 watts)
- 10^{12} Glial cells (power, cooling, support)
- 10^{11} Neurons (soma + wires)
- 10^{14} Connections (synapses)
- Volume = mostly wires.



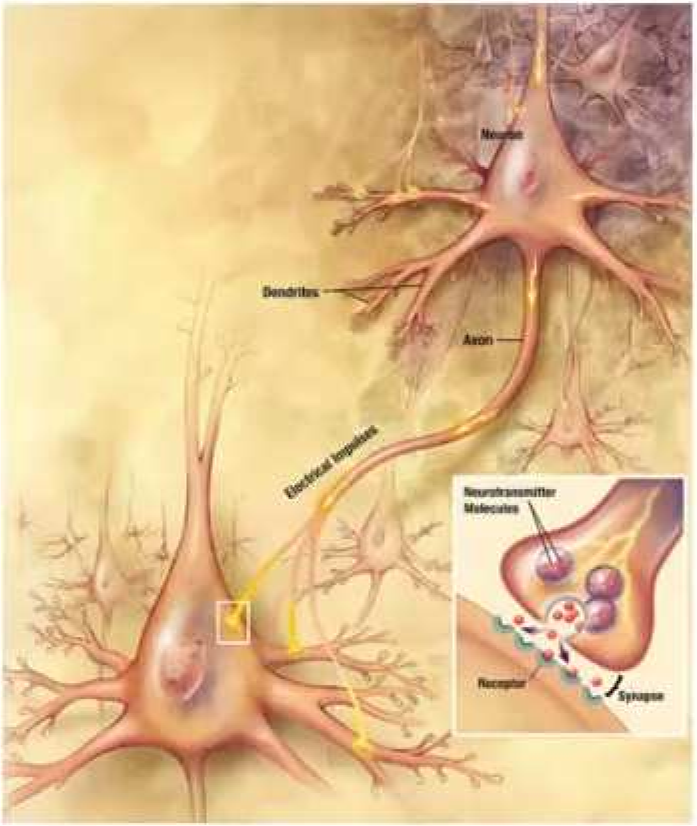
General computing machine?

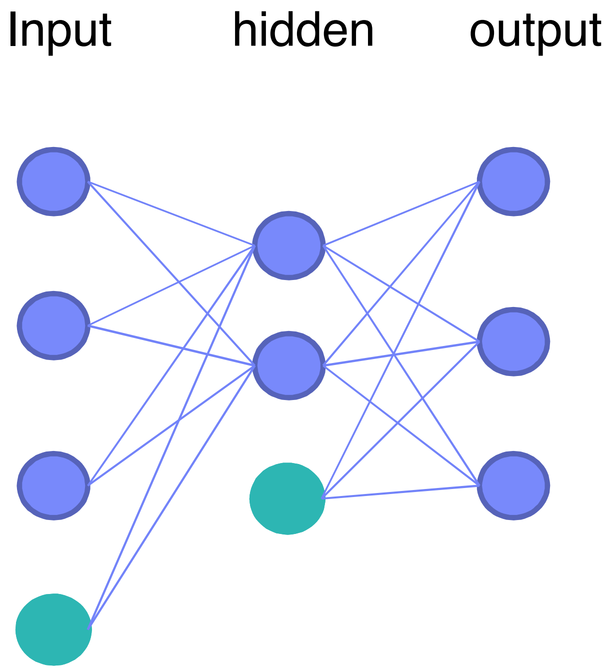
- Slow for mathematical logic, arithmetic, etc.
- Very fast for vision, speech, language, social interactions, etc.
- Evolution: vision –> language –> logic.

Neural Networks

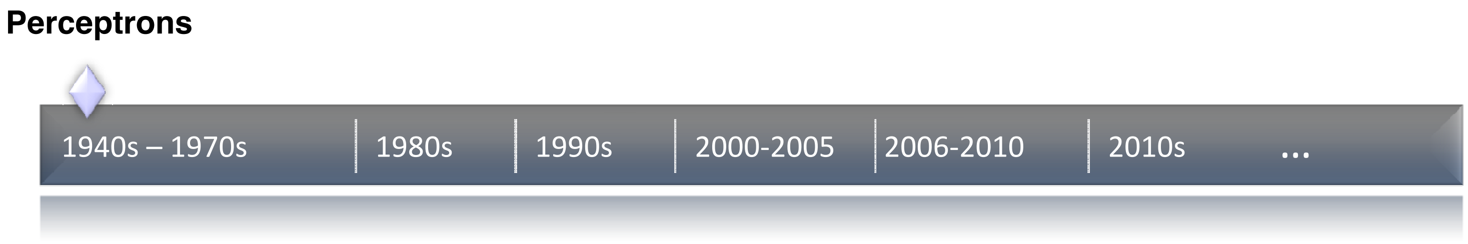
Lecturer : Fangqiu Han
Computer Science
University of California at Santa Barbara

What is neural networks?





Neural network timeline



Perceptrons

- The first perceptron was called Binary Threshold Models, and was first introduced by McCulloch and Pitts in 1943.
- Later it was popularized by Frank Rosenblatt in the early 1957.
- A famous book entitled Perceptrons by Marvin Minsky and Seymour Papert showed that it was impossible for these classes of network to learn an XOR function.

Neural network timeline

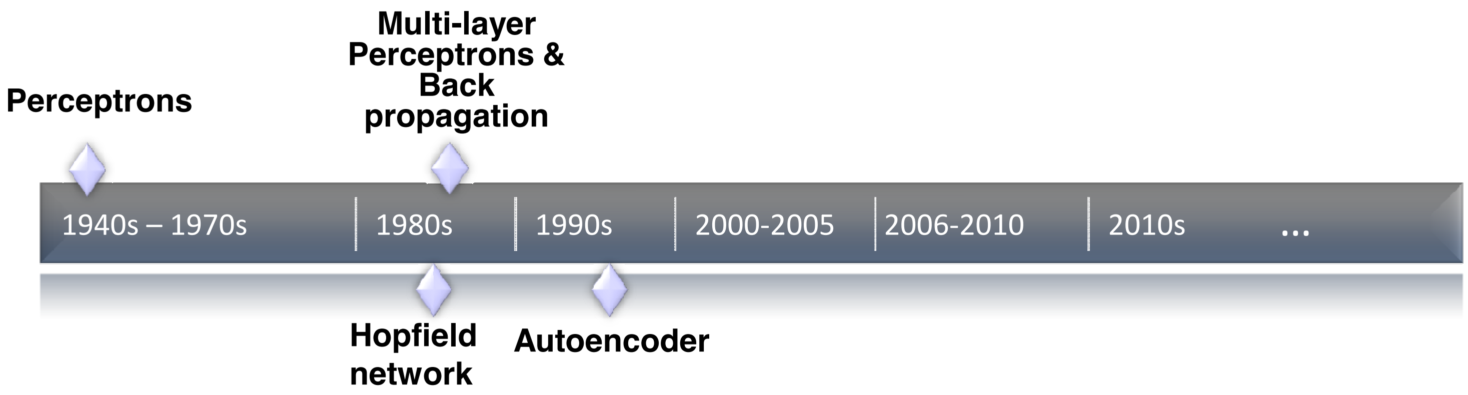
Multi-layer Perceptrons

- Also called feed forward networks.
- Introduced by Rumelhart, Hinton, and Williams in 1986.

Backpropagation

- First developed by Werbos in his doctoral dissertation in 1974.
- Remained almost unknown in the scientific community until rediscovered by Parker In 1982, and Rumelhart, Hinton, and Williams in 1986.

Neural network timeline

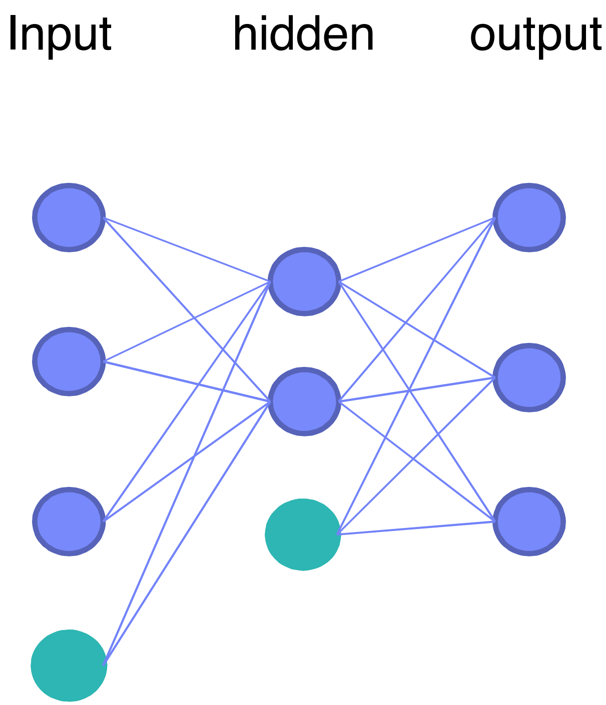


Hopfield network

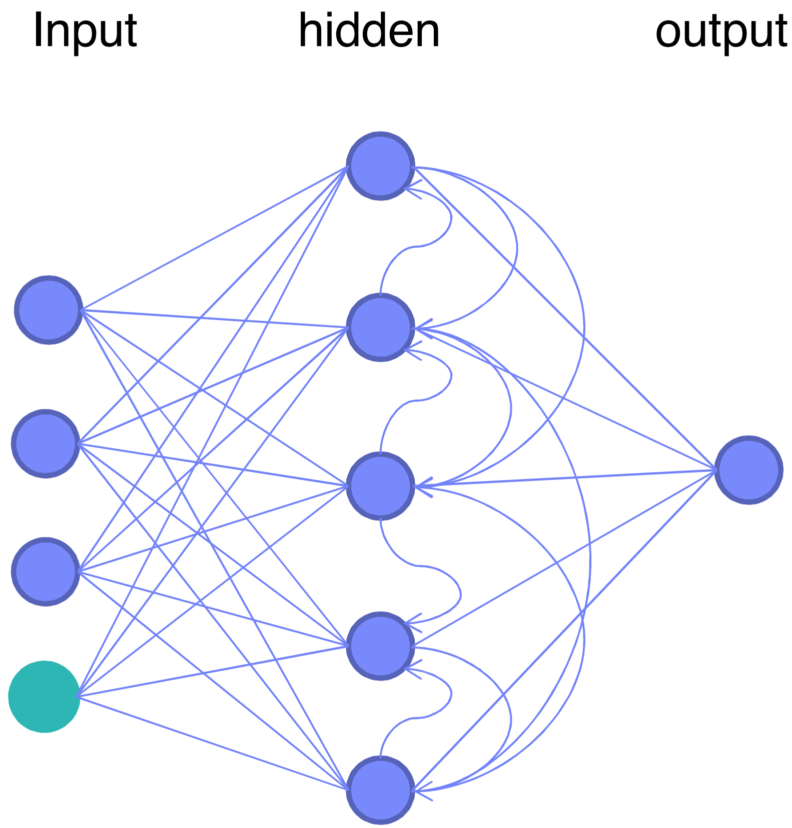
- First famous recurrent neural network invented by John Hopfield in 1982.
- A energy based model, inspired by Ising model in physics.
- Inspire the idea of Restricted Boltzmann Machine.

Autoencoder

- Learn a distributed representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.
- Idea first introduced by Olshausen in the name of Sparse Coding in 1996.

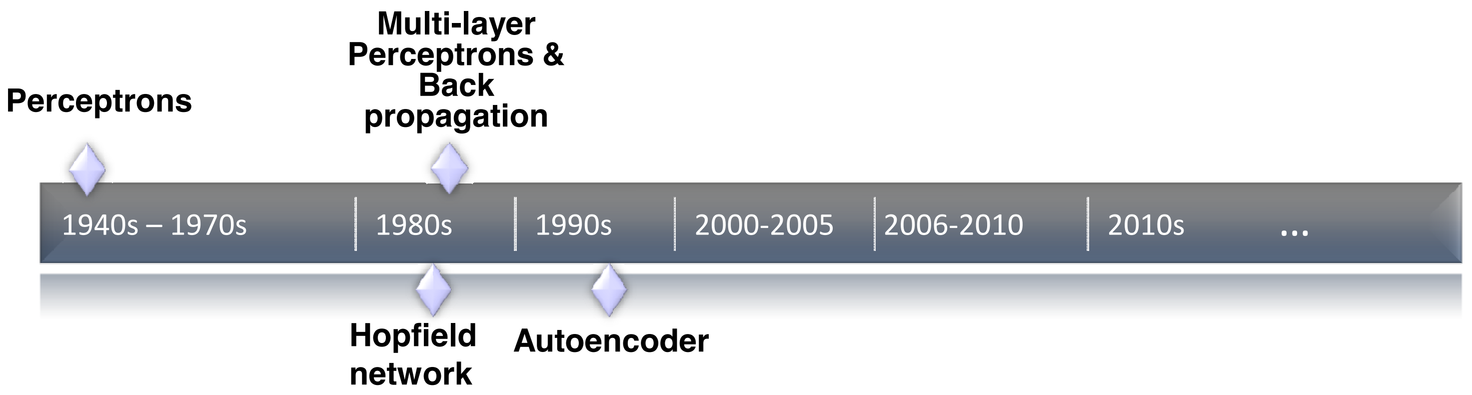


Multi-layer Perceptrons



Recurrent neural networks

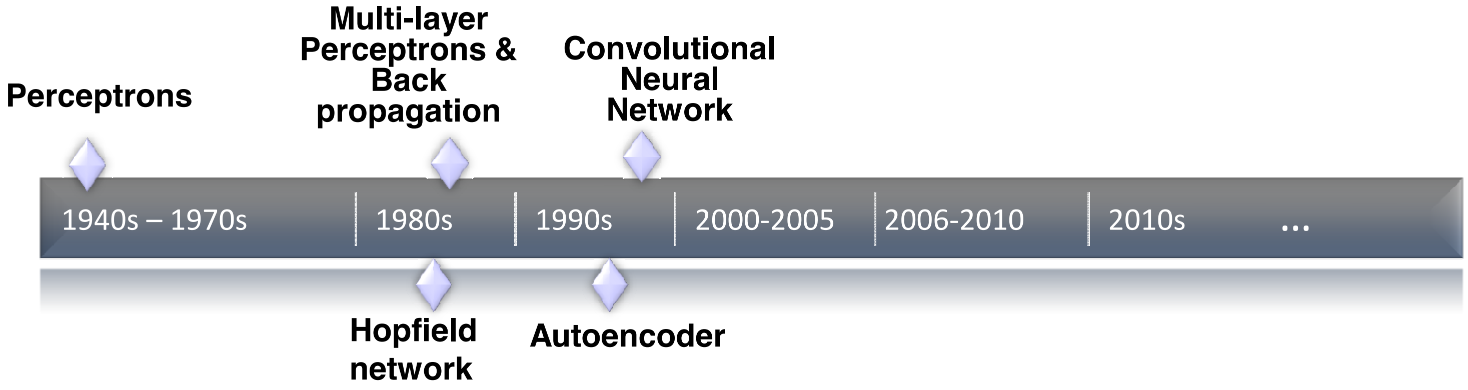
Neural network timeline



Hopfield network
- First famous recurrent neural network invented by John Hopfield in 1982.
- A energy based model, inspired by Ising model in physics.
- Inspire the idea of Restricted Boltzmann Machine.

Autoencoder
- Learn a distributed representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.
- Idea first introduced by Olshausen in the name of Sparse Coding in 1996.

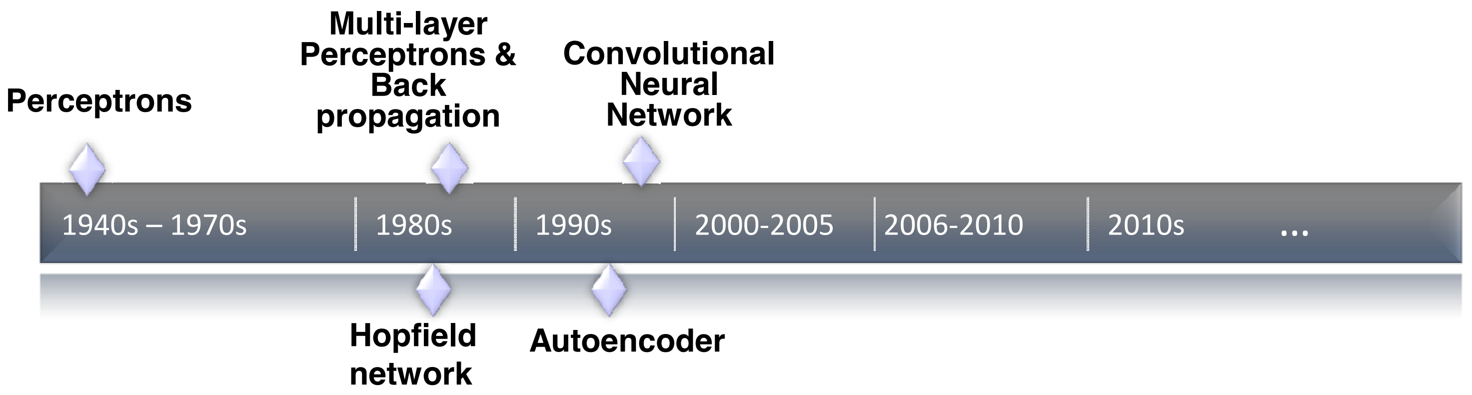
Neural network timeline



Convolutional Neural Network

- First successful deep Neural Network.
- First introduced by Kunihiko Fukushima in 1980.
- The design was later improved in 1998 by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.
- Still the state-of-art neural nets in computer vision.

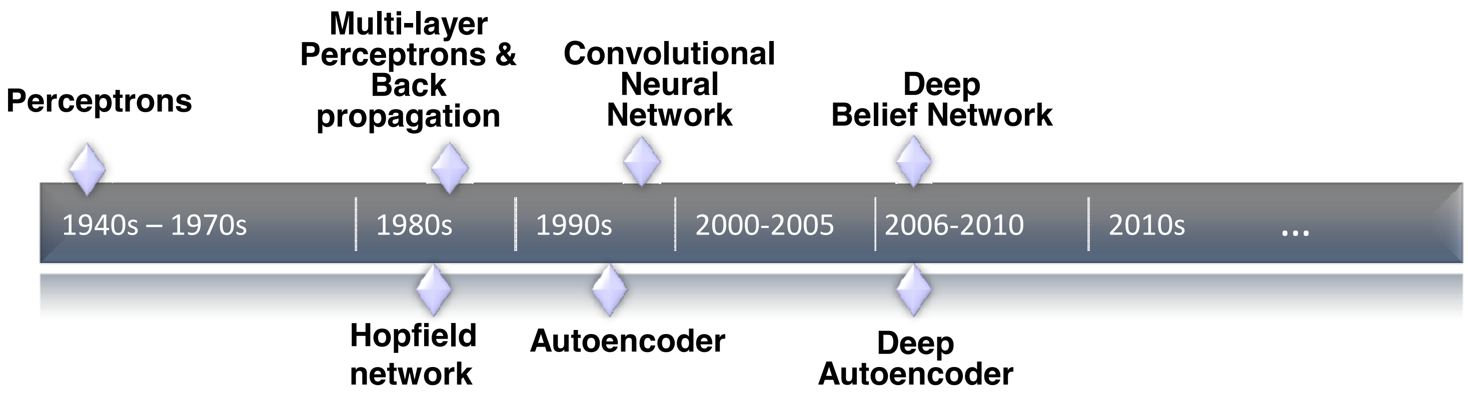
Neural network timeline



Popularity diminished in late 1990s

- Multi layer Perceptrons are not easy to train.
- The training of the only ‘trainable’ Convolutional neural nets is not efficient.
- Kernel method, e.g. SVM, are showed to be both efficient and effective.

Neural network timeline

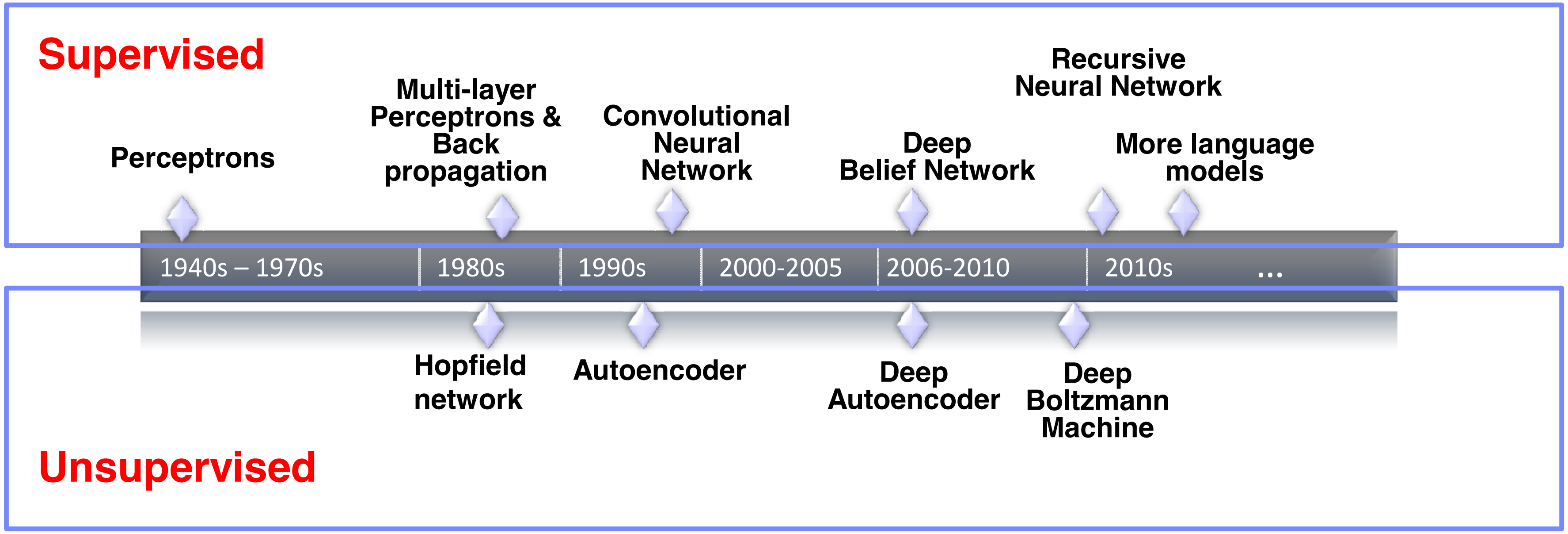


Deep Belief Network / Deep autoencoder

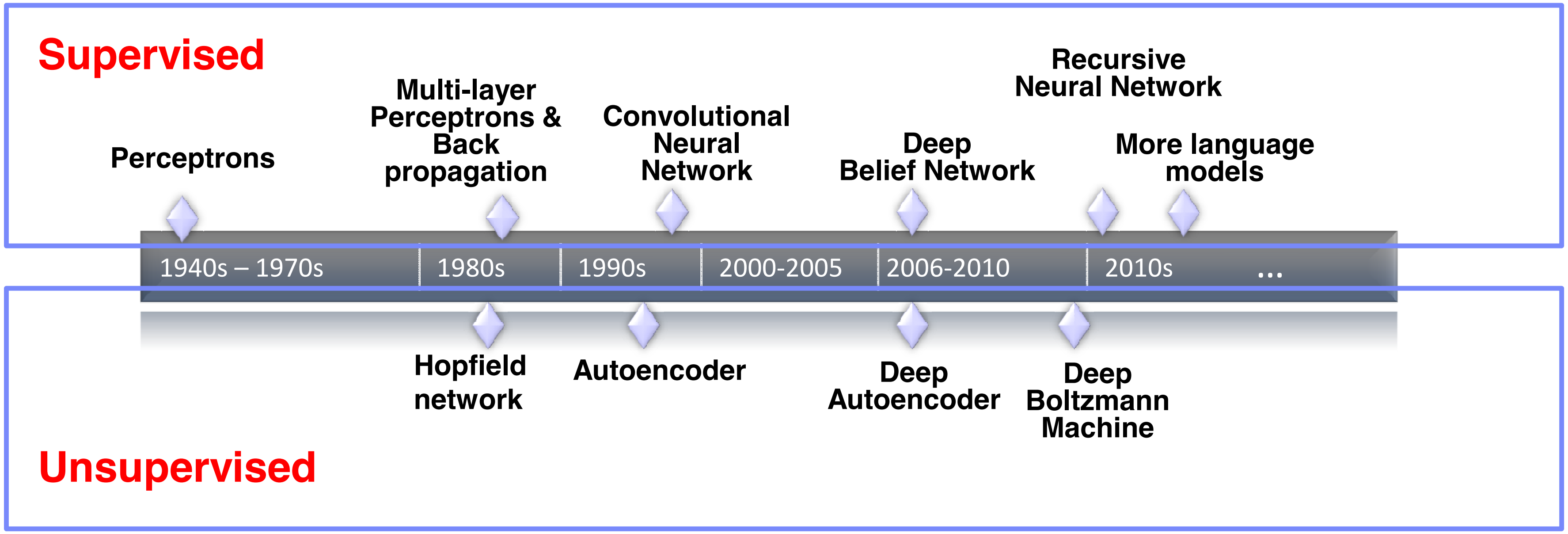
➢ A multi layer Perceptrons / autoencoder pre-trained by Restricted Boltzmann Machine, then fine-tuning using back-propagation.

➢ Restricted Boltzmann Machines, special cases of Hopfield Networks, is first invented by Paul Smolensky in 1986, but only rose to prominence after Hinton etc. invented fast learning algorithms in 2006.

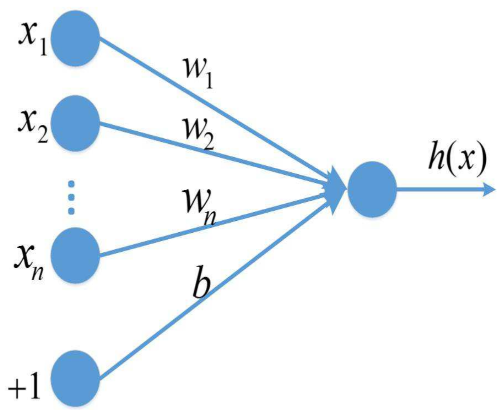
Neural network timeline



Neural network timeline



Perceptron: the simplest neural network



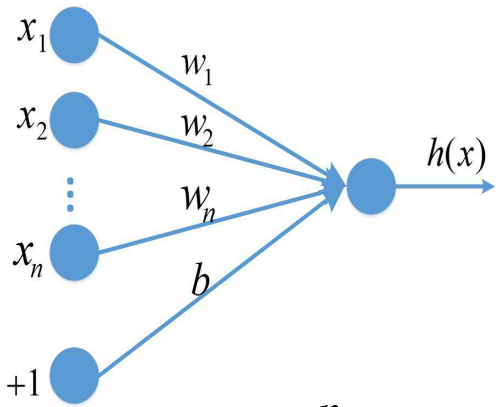
x: n-dimension input

w: parameters (weights)

b: bias

$$h(x) = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(w^T x + b)$$

Perceptron: the simplest neural network



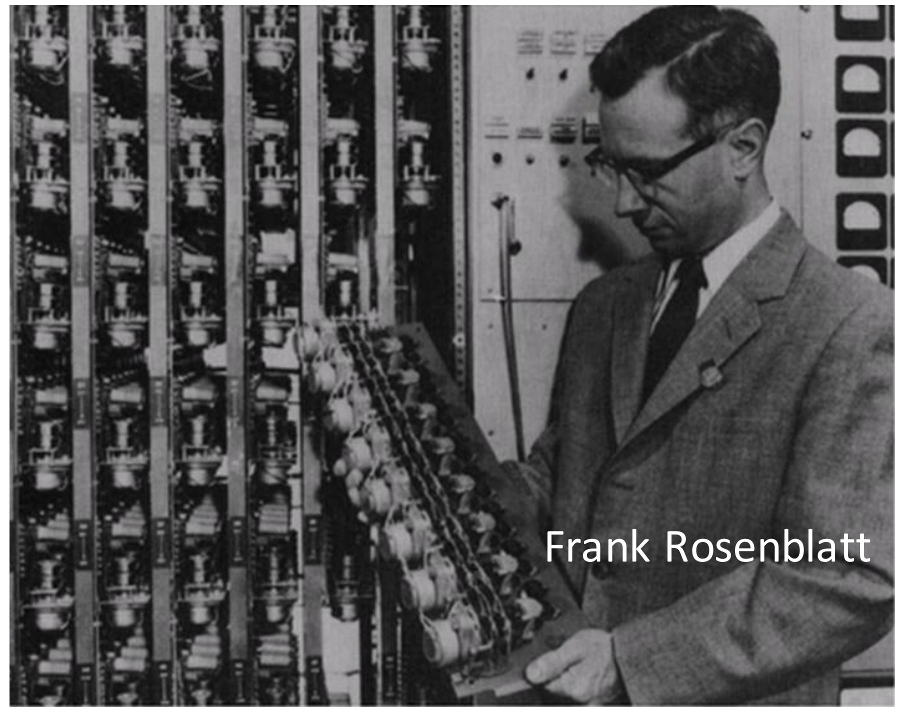
x: n-dimension input
w: combination weights
b: *bias*

$$h(x) = f(\sum_{i=1}^{n} w_i x_i + b) = f(w^T x + b)$$

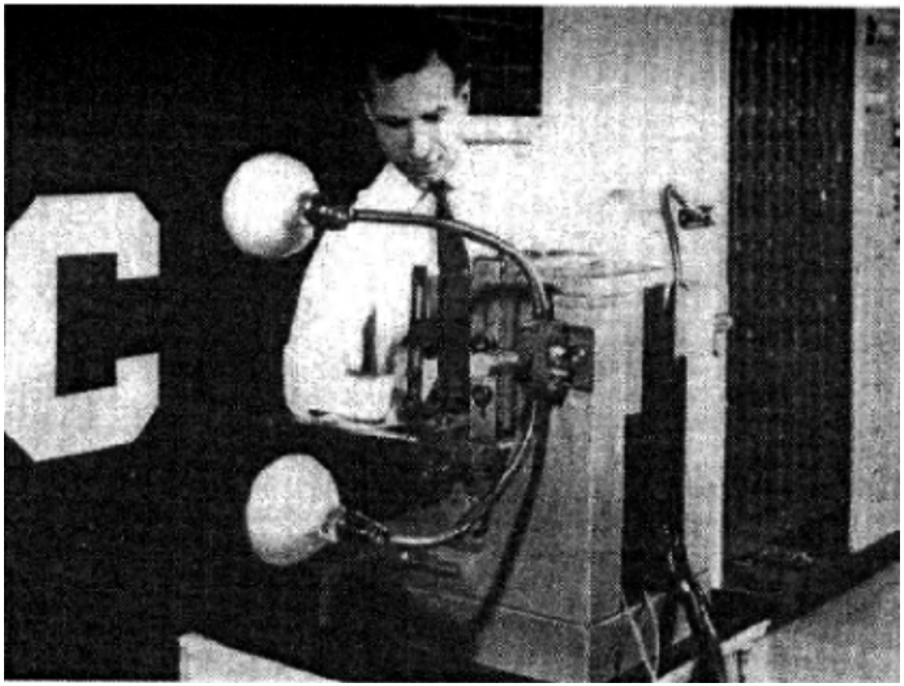
$f(\cdot)$ is called Activation function, e.g.,

Step function: $$f(z) = \begin{cases} +1 & if \ z > 0 \\ -1 & otherwise \end{cases}$$

The perceptron is a machine

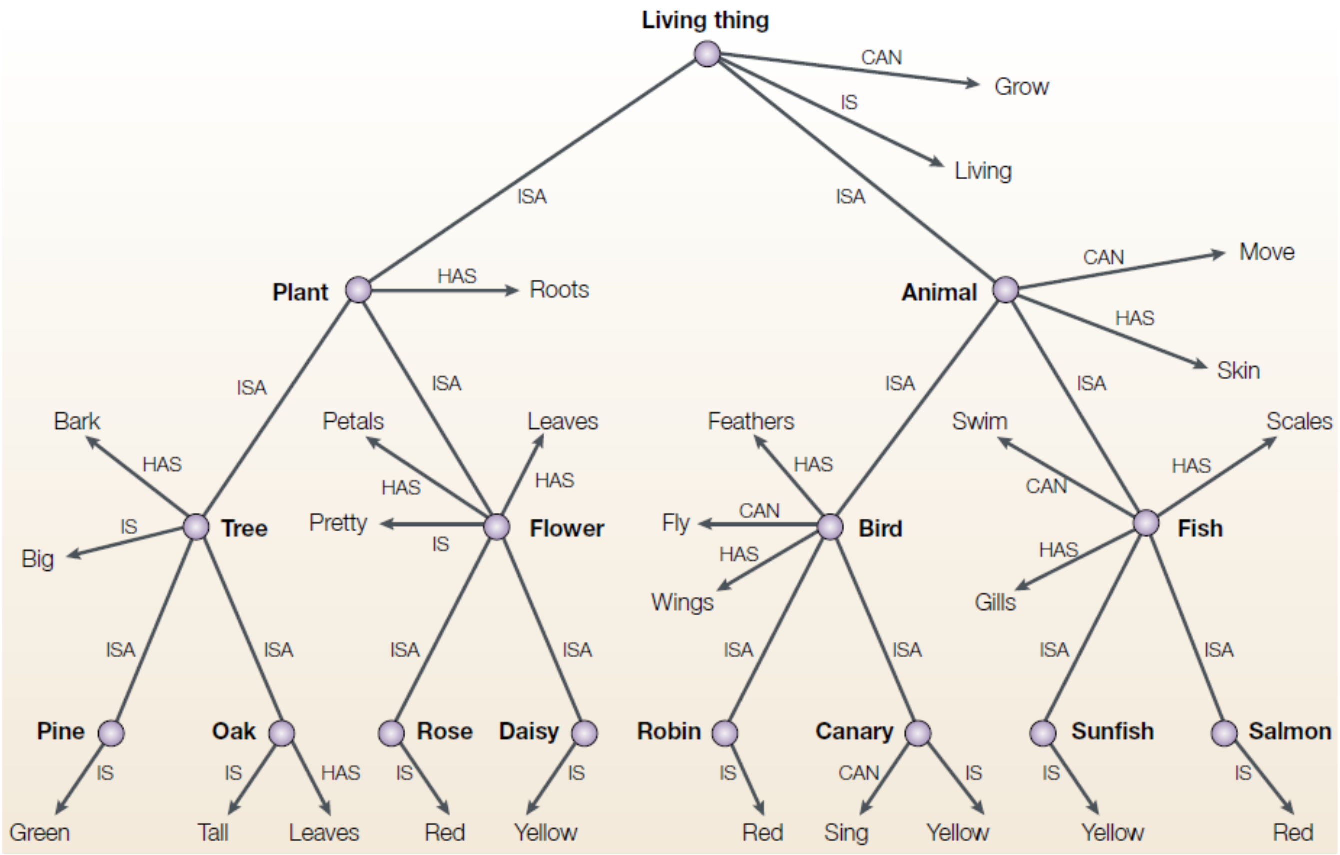


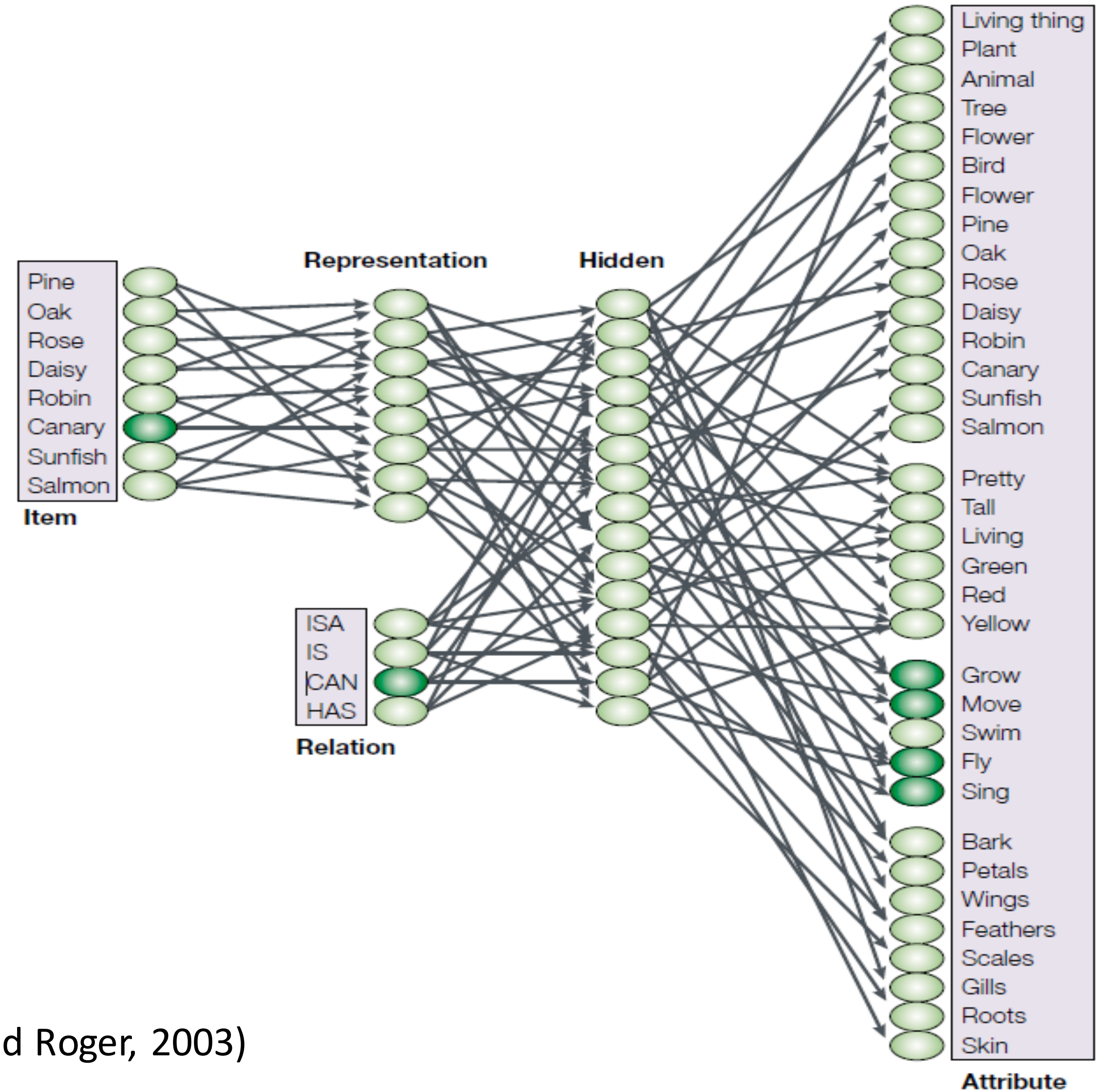
The perceptron



- The perceptron does things that vintage computers could not match.
- Alternative computer architecture? Analog computer?

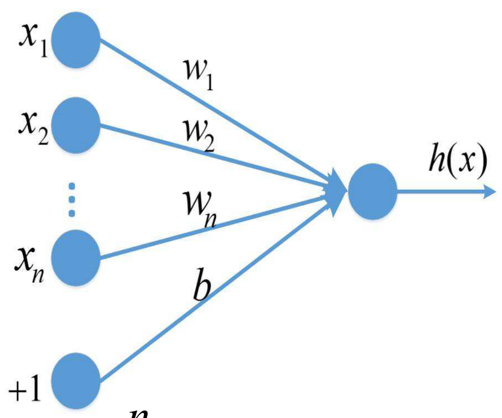
Quillian's hierarchical propositional model (1968)





(see McClelland and Roger, 2003)

Perceptron with sigmoid activation function



x: n-dimension input
w: combination weights
b: *bias*

$$h(x) = f(\sum_{i=1}^{n} w_i x_i + b) = f(w^T x + b)$$

- Activation function $f(\cdot)$, e.g.,

Sigmoid function $$f(z) = \frac{1}{1 + e^{-z}}$$

Construct cost function to learn parameters $\{w, b\}$: $E = [\mathrm{t} - h(\mathrm{x})]^2$

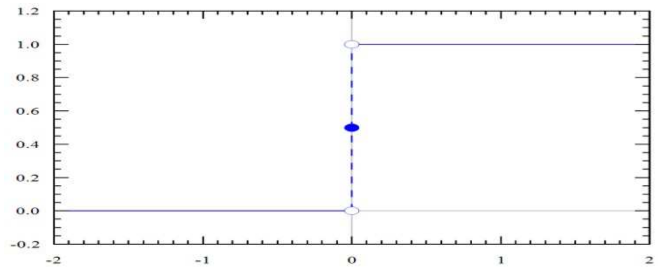
Where t is {1, 0} to denote two classes.

Logistic regression

Activation functions

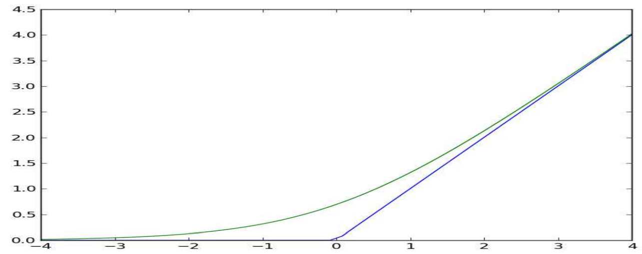
- Step function:

$$f(z) = \begin{cases} +1, z > 0 \\ 0, z \leq 0 \end{cases}$$



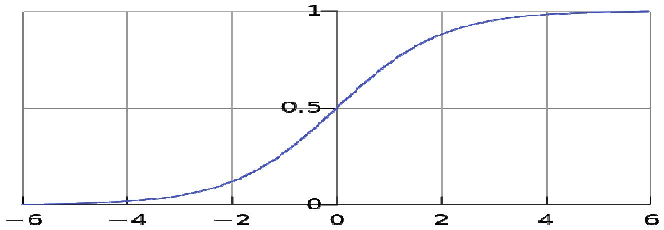
- Rectifier function:

$$f(z) = \max \{0, z\}$$



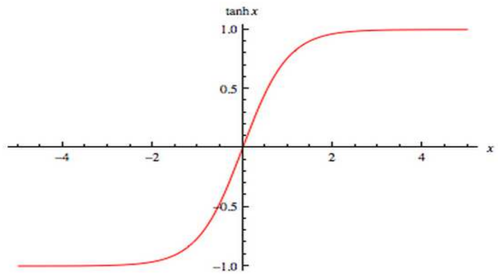
- Sigmoid function

$$f(z) = \frac{1}{1+e^{-z}}$$



- Hyperbolic tan function

$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



- Stochastic binary neural

$$\mathrm{P}(f(z) = 1) = \frac{1}{1 + e^{-z}}$$

Perceptron: the simplest neural network

- Algorithm

  1. Initialize: w, b

  2. For each data point x and label t

     Predict the label of x: $y = f(w^T x + b)$

     If y≠t, update the parameters by gradient descent

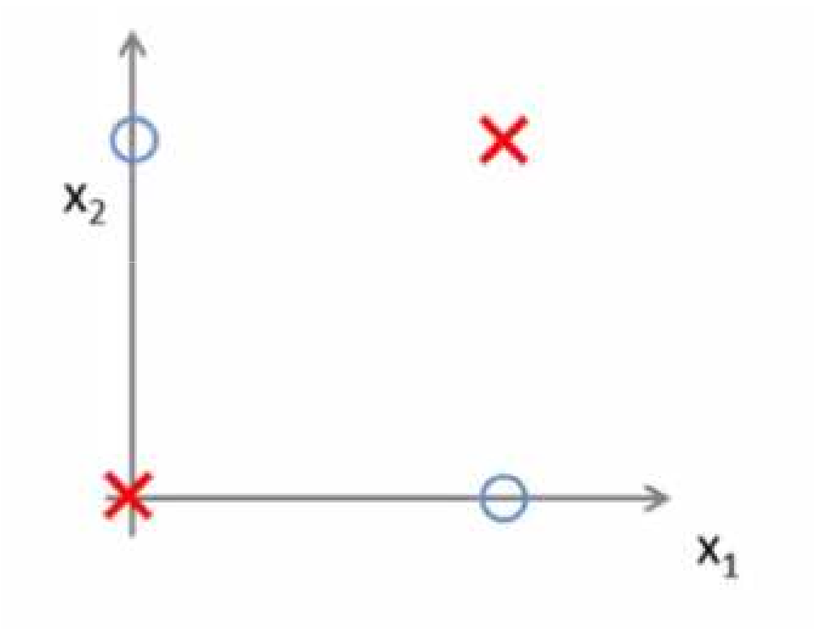
     $$w \leftarrow w - \eta\,(\nabla_w E) \quad \text{and} \quad b \leftarrow b - \eta\,(\nabla_b E)$$

     where $E = [\mathrm{t} - h(\mathrm{x})]^2$

     Else w and b does not change

  3. Repeat until convergence

Motivating example: Non-linear classification

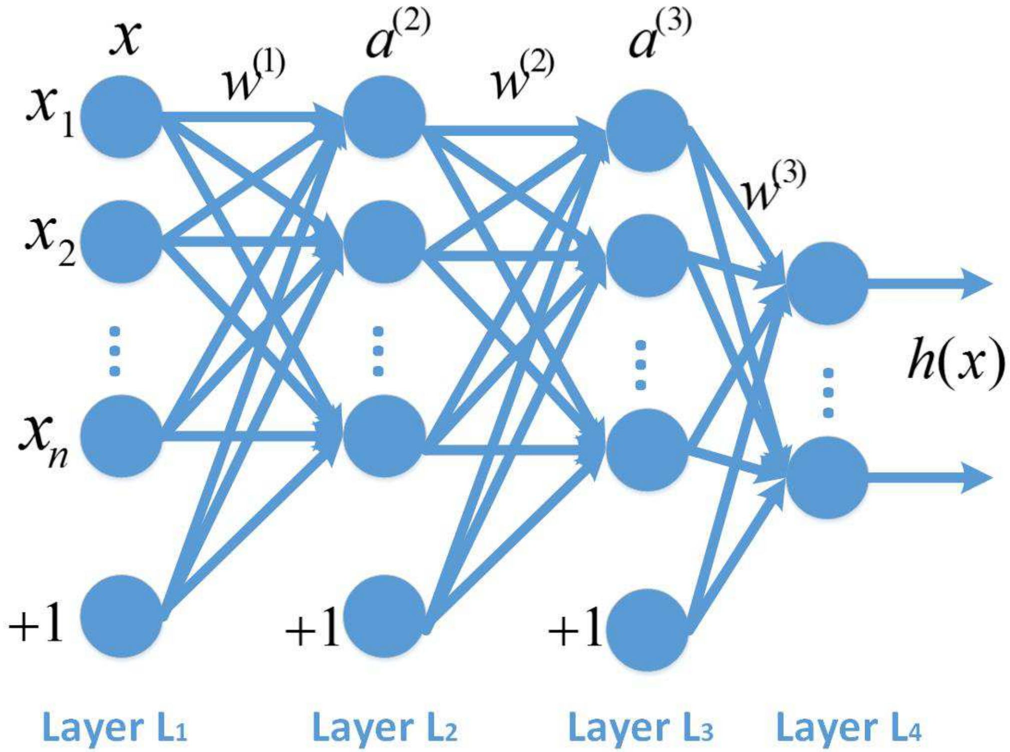


- x_1 and x_2 are binary (0 or 1)
- Learn y= x_1 xor x_2
- Perceptron does not work as the problem is not linear separable.
- One solution: Multi-layer Perceptron.

Multi-layer Perceptrons

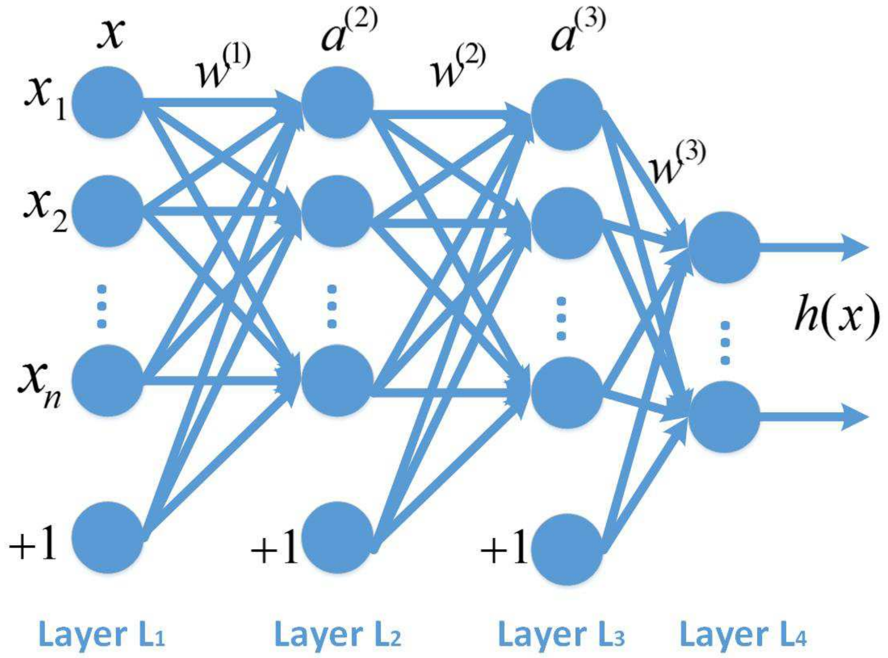
- Second generation (1980s)
  - Feed-forward neural networks

Stack of "perceptrons"



Multi-layer Perceptrons

- Second generation (1980s)



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

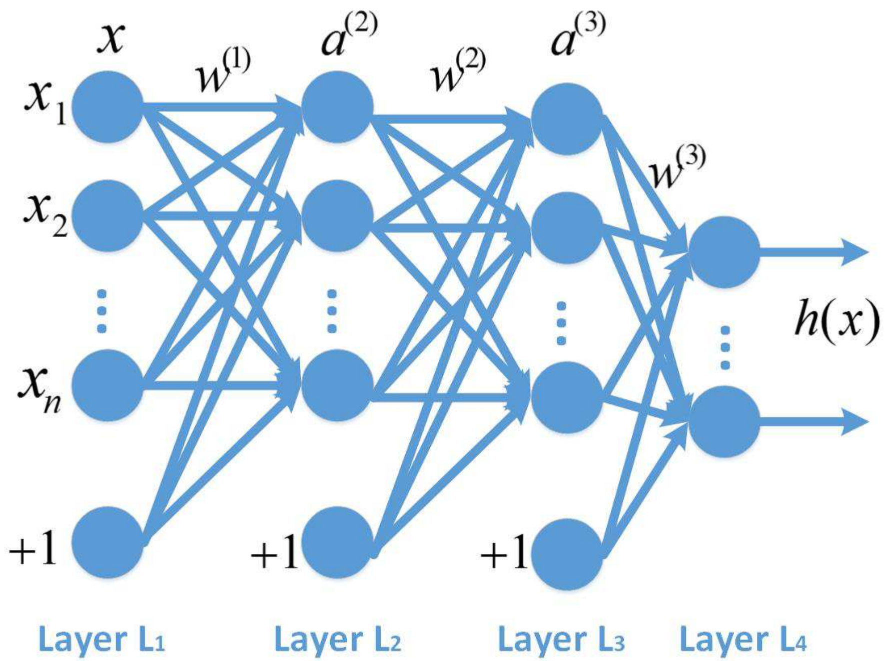
$$a^{(3)} = f(z^{(3)})$$

Output layer:

$$h(x) = f(w^{(3)}a^{(3)} + \mathrm{b}^{(3)})$$

Multi-layer Perceptrons

- Second generation (1980s)



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Output layer:

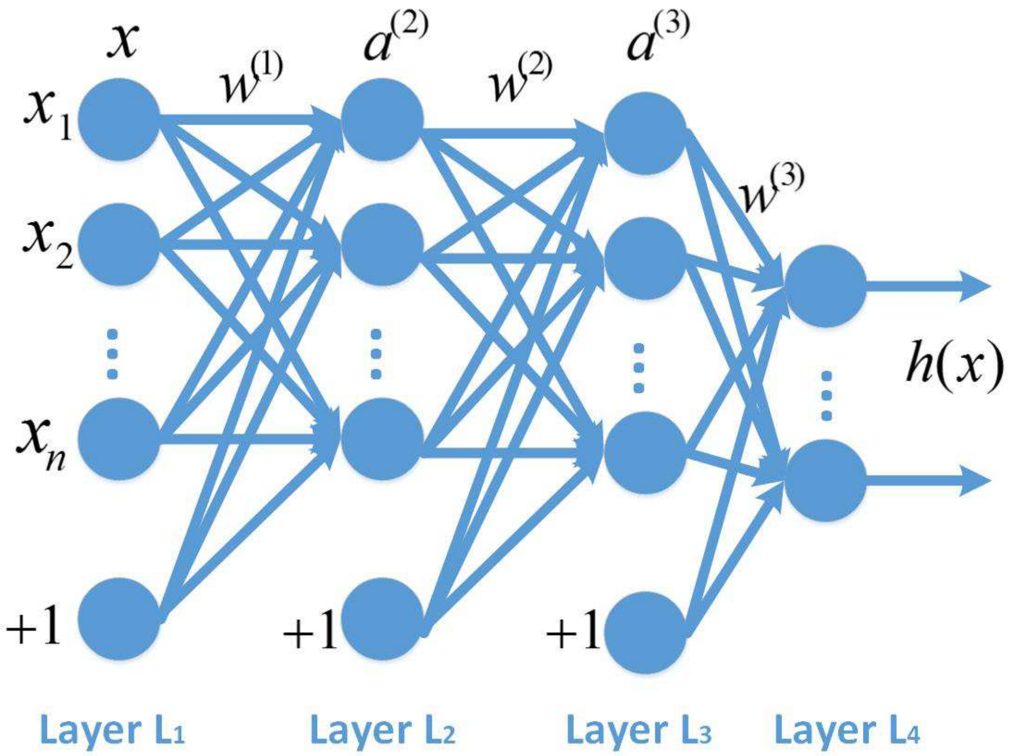
$$h(x) = f(w^{(3)}a^{(3)} + \mathrm{b}^{(3)})$$

Activation function f : continuous nonlinear function

$$f(\mathrm{z}) = \frac{1}{1+e^{-z}} \text{ (sigmoid)}, \quad or, \quad f(\mathrm{z}) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \text{ (tanh)}$$

Multi-layer Perceptrons

- Second generation (1980s)



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

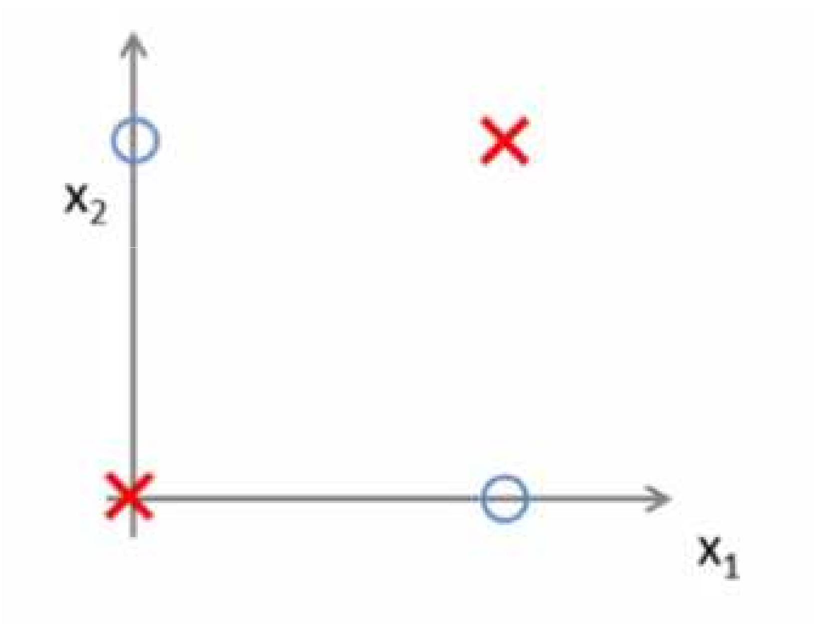
$$a^{(3)} = f(z^{(3)})$$

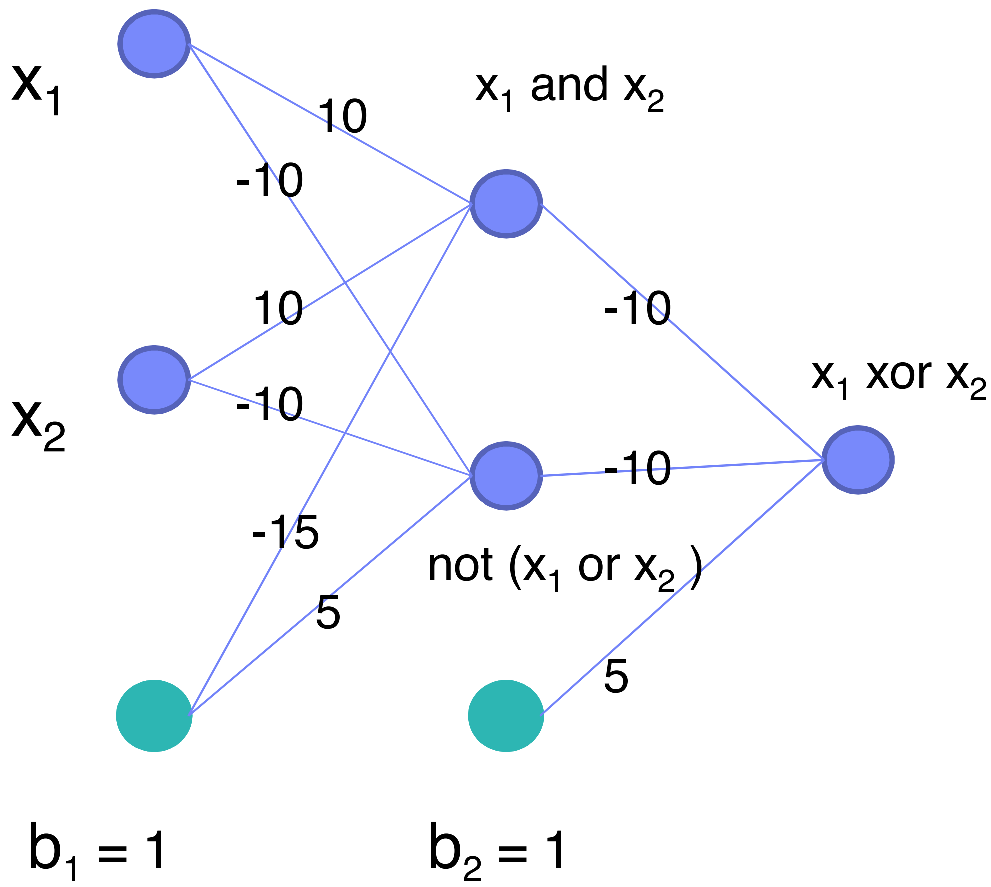
Output layer:

$$h(x) = f(w^{(3)}a^{(3)} + \mathrm{b}^{(3)})$$

Parameters { $w^{(1)}$, $w^{(2)}$, $w^{(3)}$, $b^{(1)}$, $b^{(2)}$, $b^{(3)}$ } to be learnt.

Motivating example: a solution





Universal Approximation Theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function.

- Here 'mild' means any non-constant, bounded, and monotonically-increasing continuous function.
- Example activation functions
  - Sigmoid function
  - Hyperbolic Tan function
  - Rectifier function

Activation functions

- Step function:

$$f(z) = \begin{cases} +1, z > 0 \\ 0, z \leq 0 \end{cases}$$

- Rectifier function:

$$f(z) = \max\ \{0, z\}$$

- Sigmoid function

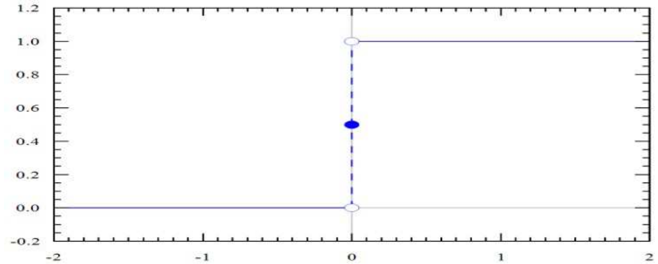
$$f(z) = \frac{1}{1+e^{-z}}$$

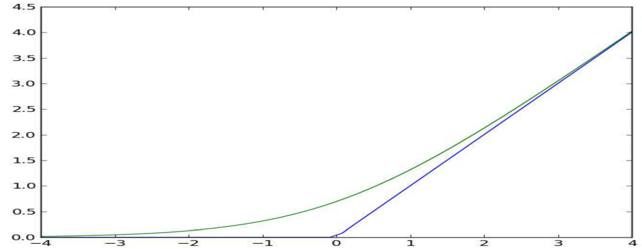
- Hyperbolic tan function

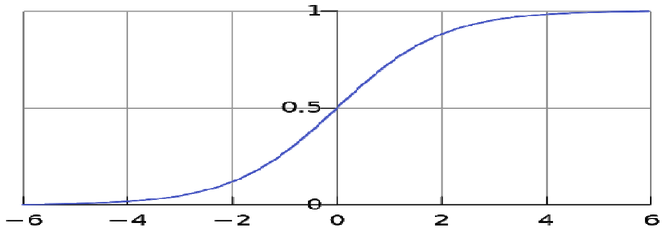
$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

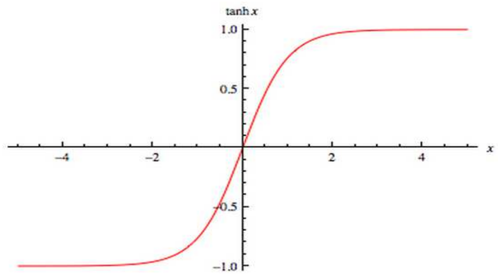
- Stochastic binary neural

$$\mathrm{P}(f(z) = 1) = \frac{1}{1 + e^{-z}}$$









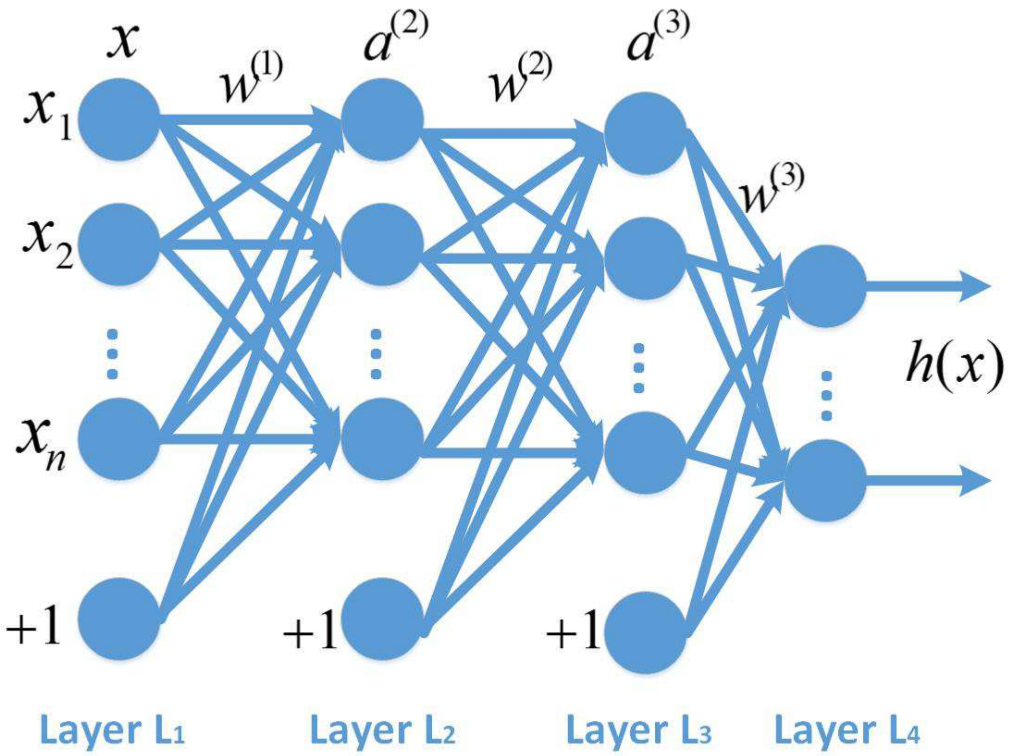
Universal Approximation Theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function.

- Here 'mild' means any non-constant, bounded, and monotonically-increasing continuous function.
- Example activation functions
  - Sigmoid function
  - Hyperbolic tan function
  - Rectifier function

Multi-layer Perceptrons

- Second generation (1980s)



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Output layer:

$$h(x) = f(w^{(3)}a^{(3)} + \mathrm{b}^{(3)})$$

Parameters { $w^{(1)}$, $w^{(2)}$, $w^{(3)}$, $b^{(1)}$, $b^{(2)}$, $b^{(3)}$ } to be learnt.

Parameter Estimation

- A training set of m data points, $\{(x^{(1)}, y^{(1)}), ..., (x^{(\mathrm{m})}, y^{(\mathrm{m})})\}$

- Objective function

$$\min \ \mathrm{H} = \frac{1}{2m}\sum_{i=1}^{m}\left\| h(x^{(i)}) - y^{(i)} \right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{L}\left\| w^{(l)} \right\|_F^2$$

where,

$\frac{1}{2m}\sum_{i=1}^{m}\left\| h(x^{(i)}) - y^{(i)} \right\|^2$: average sum-of-squares error term

$\frac{\lambda}{2}\sum_{l=1}^{L}\left\| w^{(l)} \right\|_F^2$: weight decay term; L : the number of layers

Optimization algorithm

- Gradient descent

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \alpha \frac{\partial H}{\partial w_{ij}^{(l)}}$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial H}{\partial b_i^{(l)}}$$

Optimization algorithm

- Gradient descent

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \alpha \frac{\partial H}{\partial w_{ij}^{(l)}}$$

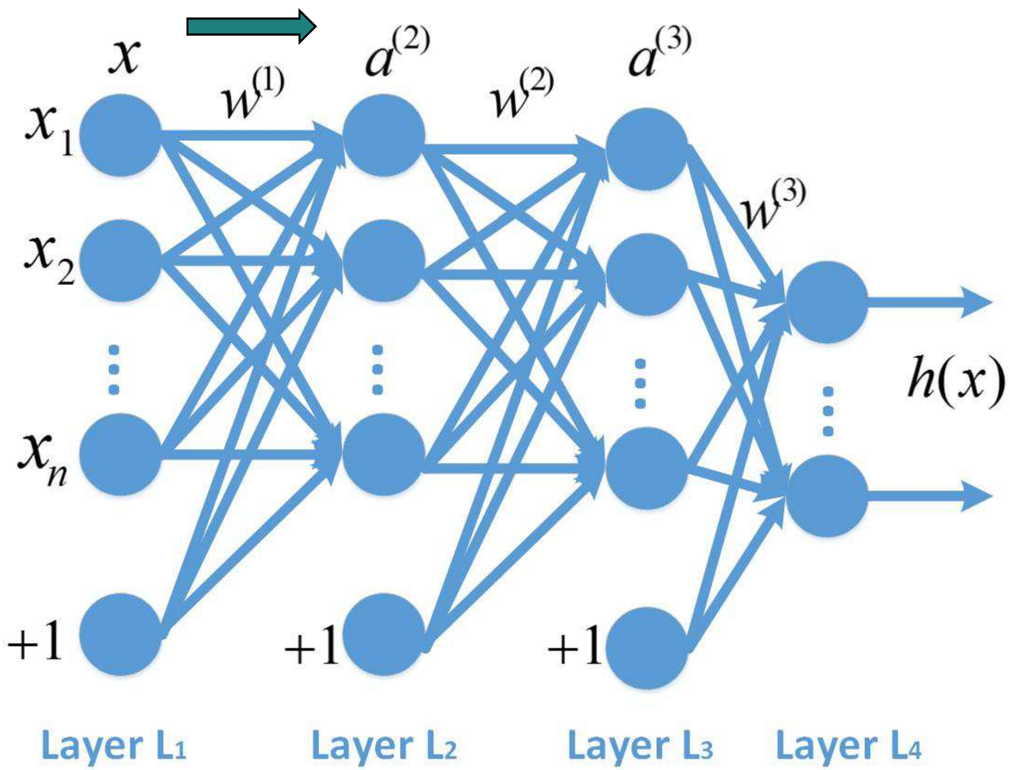
$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial H}{\partial b_i^{(l)}}$$

- Backpropagation algorithm: a systematic way

to compute $\frac{\partial H}{\partial w_{ij}^{(l)}}$ and $\frac{\partial H}{\partial b_i^{(l)}}$

Backpropagation

- Perform a **feedforward pass**, computing the activations for layers *L2*, *L3*, and so on up to the output layer *h(x)*.



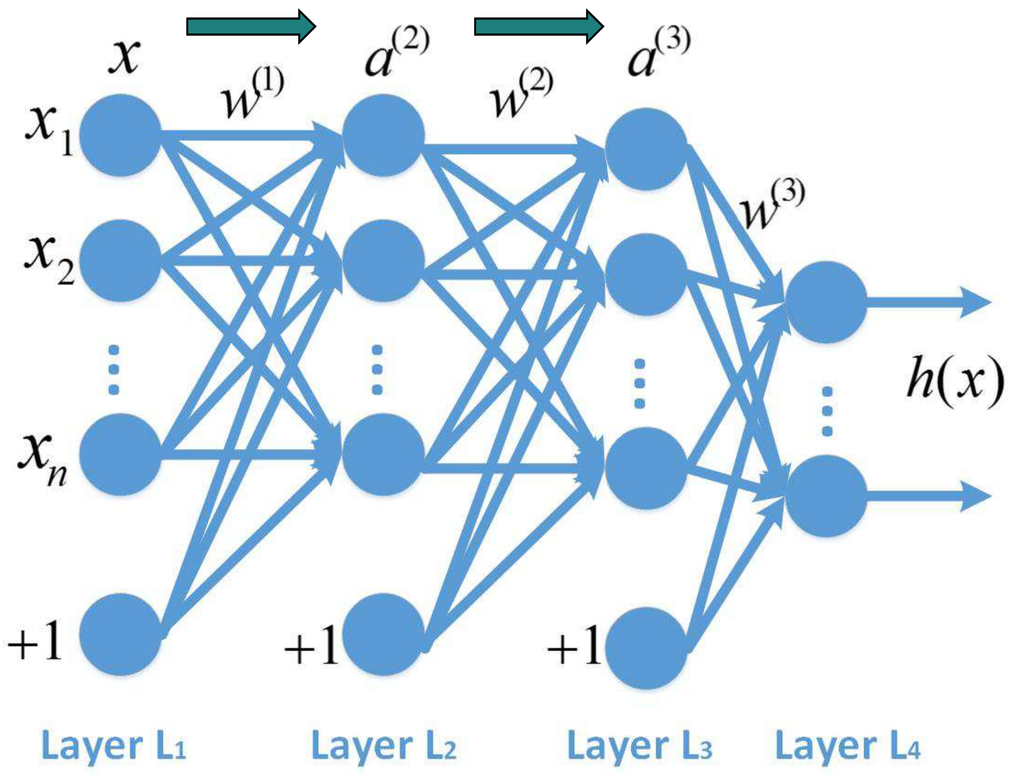
Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Backpropagation

- Perform a **feedforward pass**, computing the activations for layers L_2, L_3, and so on up to the output layer $h(x)$.



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

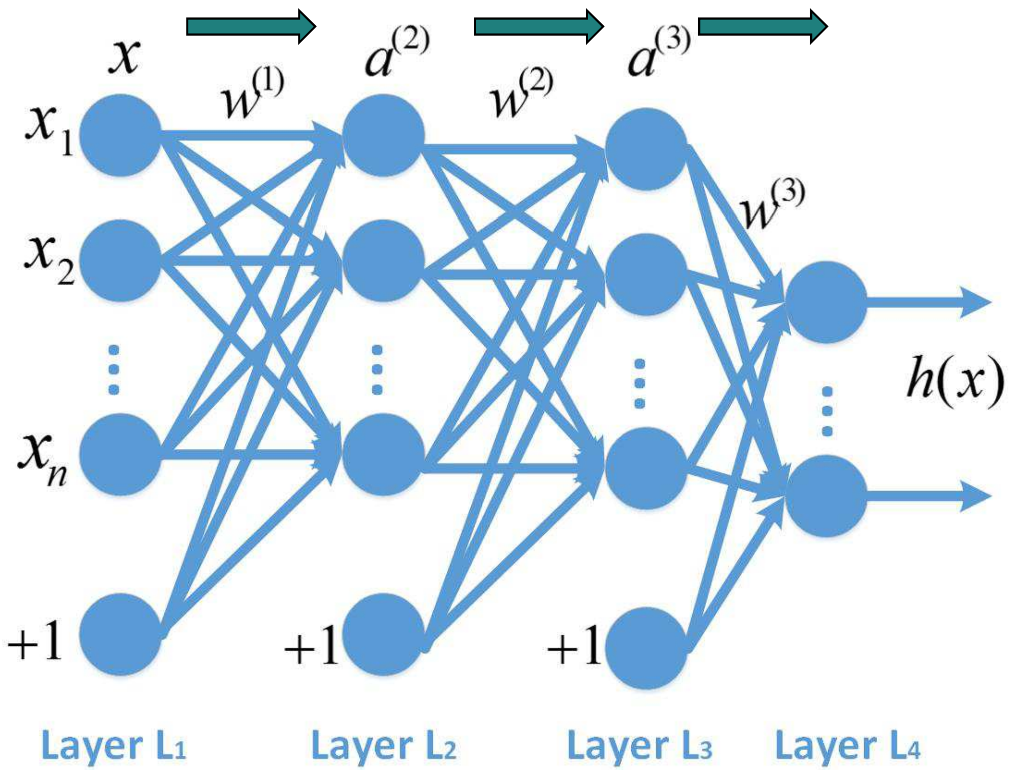
Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Backpropagation

- Perform a **feedforward pass**, computing the activations for layers *L2*, *L3*, and so on up to the output layer *h(x)*.



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Output layer:

$$h(x) = f(w^{(3)}a^{(3)} + \mathrm{b}^{(3)})$$

Loss bricks

	Propagation	Back-propagation
Square	$y = \frac{1}{2}(x-d)^2$	$\frac{\partial E}{\partial x} = (x-d)^T \frac{\partial E}{\partial y}$
Log $\quad c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1+e^{cx}} \frac{\partial E}{\partial y}$
Hinge $\quad c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \;\; \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax $\quad c = 1 \ldots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s}/\sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin $\quad c = 1 \ldots k$	$y = \left[\max_{k \neq c}\{x_k + m\} - x_c\right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \, \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

Gradient Checking (important!)

- Definition of derivative

  For function $J(\theta)$ with parameter θ

$$\frac{d}{d\theta} J(\theta) = \lim_{\varepsilon \to 0} \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

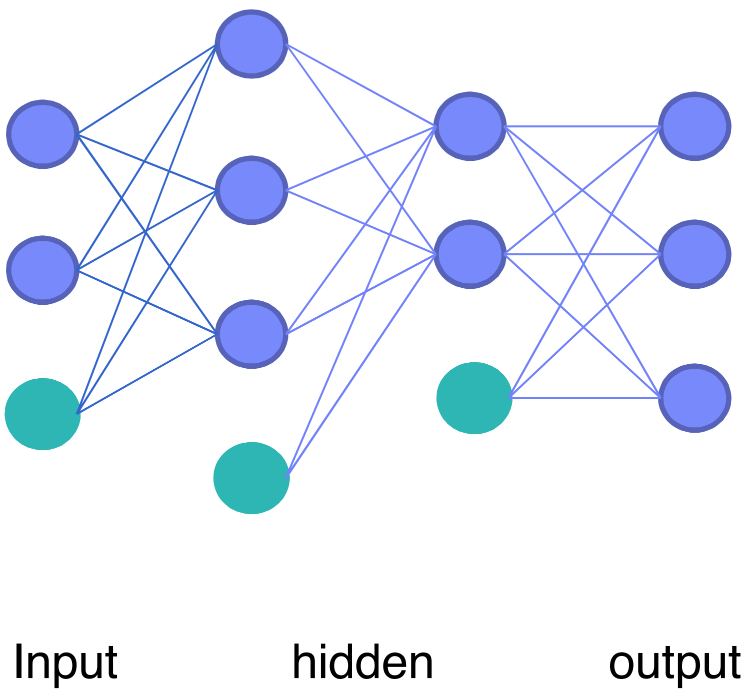
- Comparison

$$\frac{\|A - B\|_F}{\|A + B\|_F} \leq \delta$$

Where, A are the derivatives obtained by backpropagation; B are those obtained by definition;

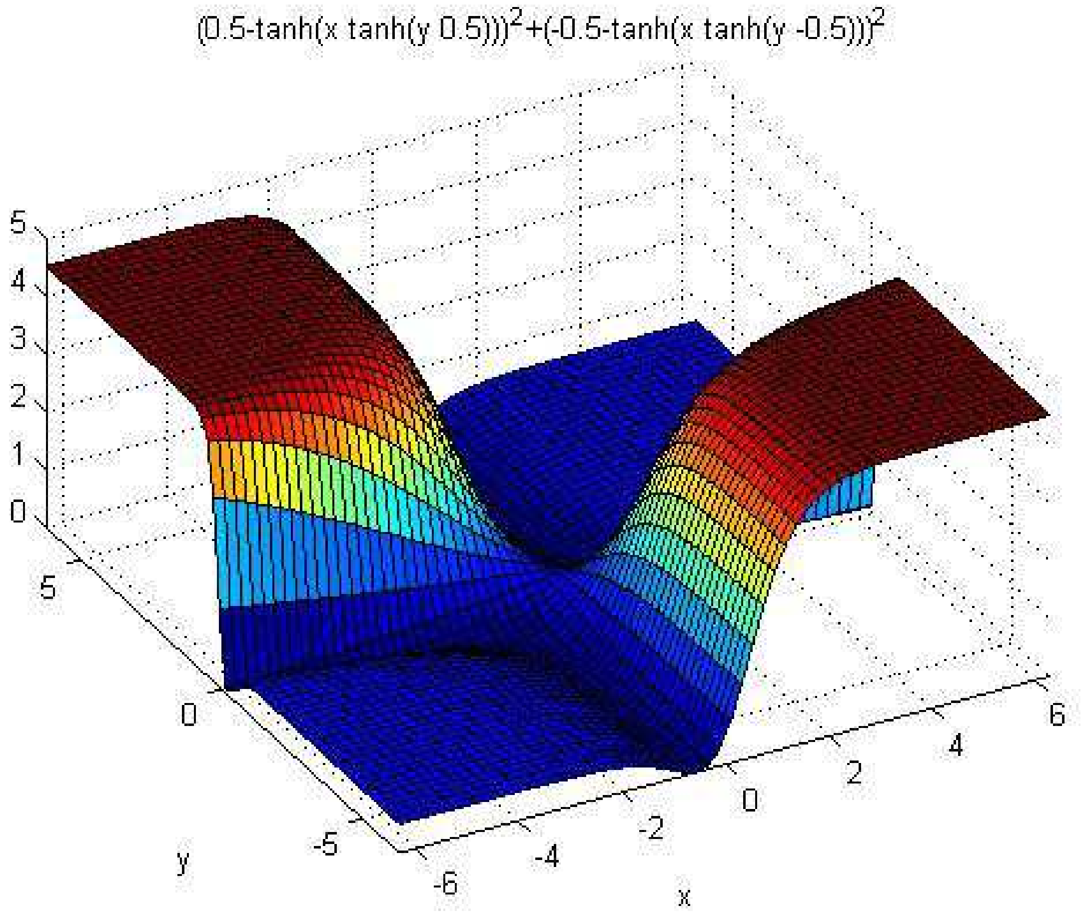
δ, usually, $\leq 10^{-9}$

Problems with back-propagation

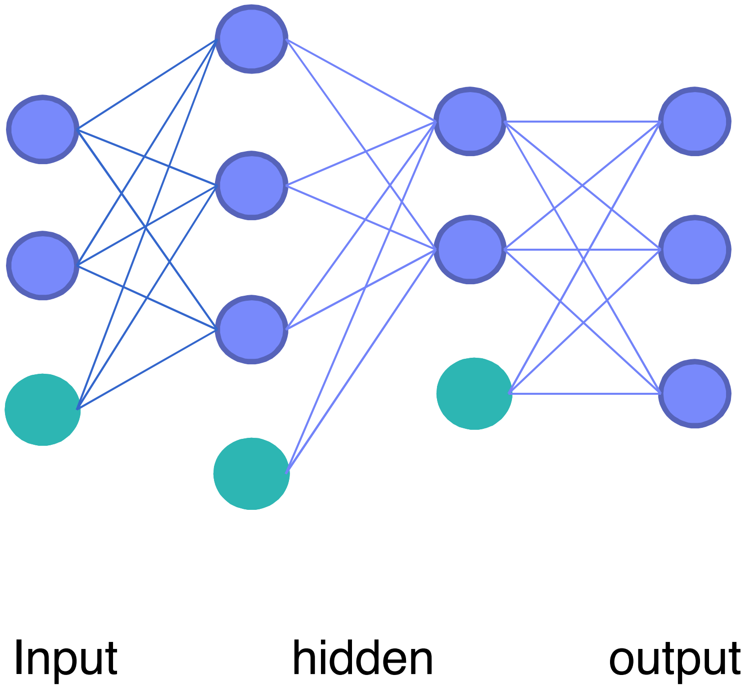


- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.

Deep Supervised Learning is Non-Convex

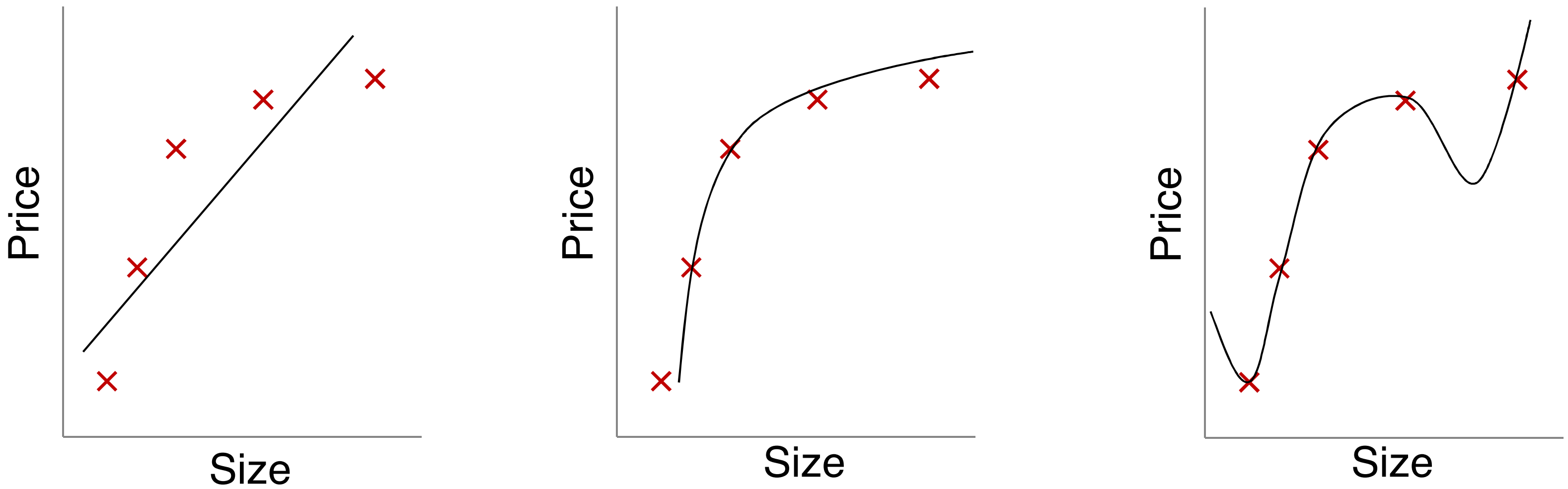


Why not multi-layer model with back-propagation



- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.
- Overfitting

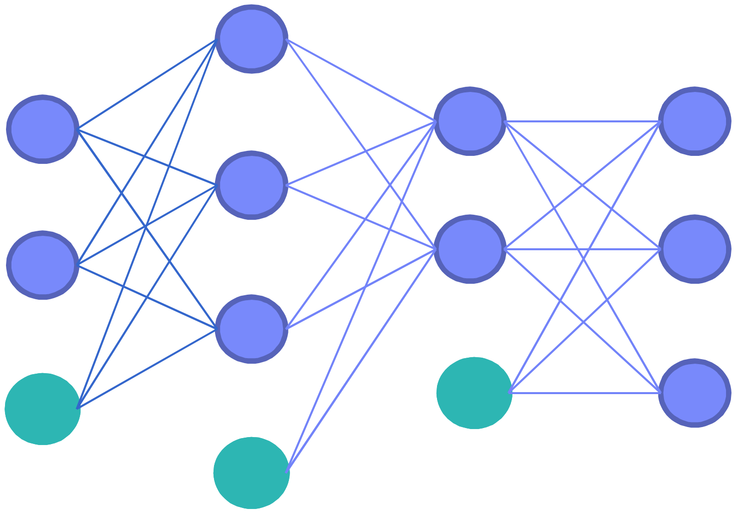
Overfitting: an example



$$\theta_0 + \theta_1 x \qquad \theta_0 + \theta_1 x + \theta_2 x^2 \qquad \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Overfitting: If we have too many parameters, the learned hypothesis may fit the training set very well, but fail to generalize to new examples (testing data).

Why not multi-layer model with back-propagation



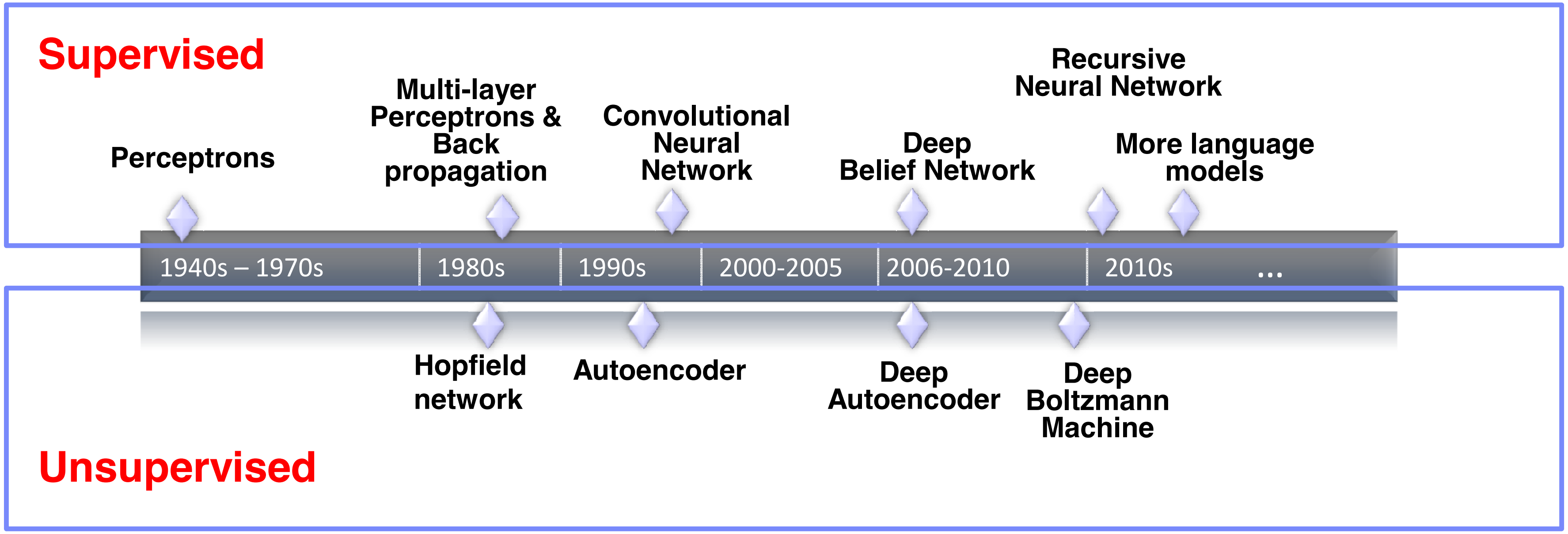
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.
- Overfitting

Solutions

- Solutions for local optima:
  - Use better initialization (Restricted Boltzmann Machine)
  - Find other method for optimization
  - Find better structures

- Solutions for overfitting:
  - More data
  - Weight decay (sparse autoencoder)
  - Reduce the number of parameters
    - Invariances (Convolutional NN)

Neural network timeline



Unsupervised neural network: Autoencoder

- Learn a distributed representation (encoding) for a set of data.

- One of the simplest unsupervised learning neural network.

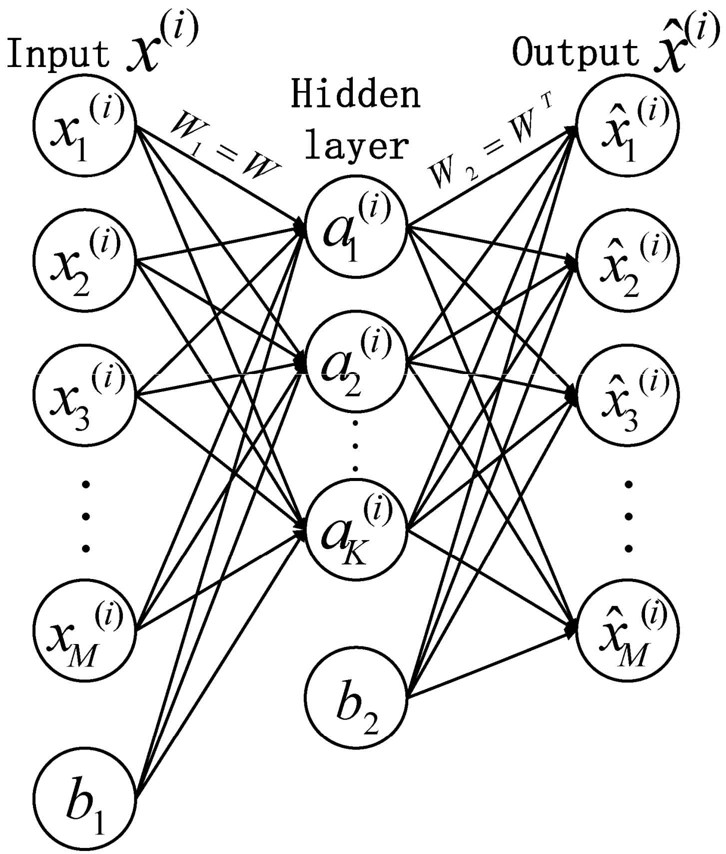
- Why unsupervised learning?

Why unsupervised learning?

- It is likely to be much more common in the brain than supervised learning. Most data are unlabeled.

- Most data are unlabeled. We need unsupervised learning to help on supervised tasks.

Autoencoder

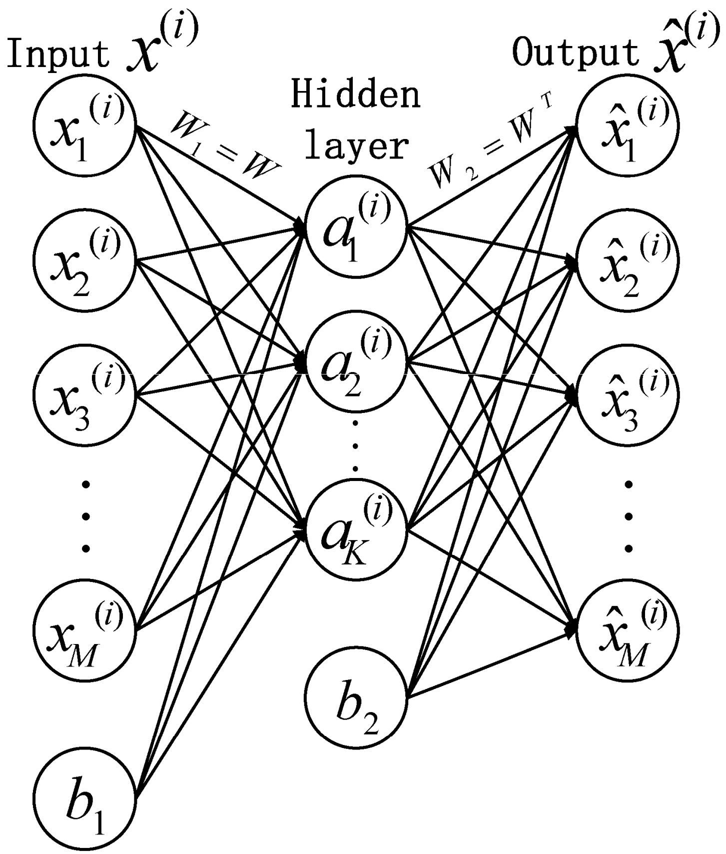


- An autoencoder is composed with an input layer, an output layer and one hidden layers connecting them.

- The difference with the MLP is that an autoencoder is trained to *reconstruct* its own inputs *x,* most time with fewer neurons in the hidden layer.

- The weights between hidden and output layer W_2 is the transpose of the weights W_1 between the input layer and the hidden layer.

Autoencoder



Activation function:

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

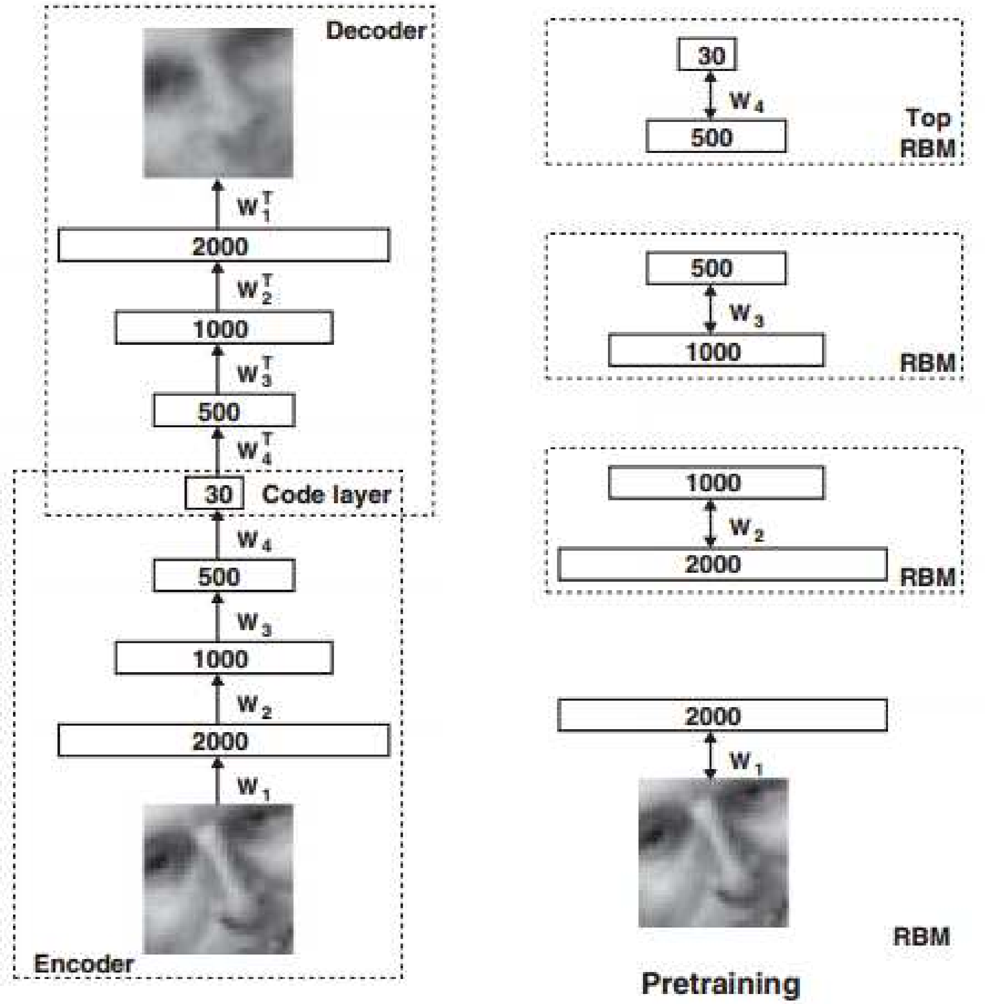
Forward pass:

$$\hat{x} = \underbrace{\boxed{f(W^T}}_{\text{decoder}}\underbrace{\boxed{f(W\, x)}}_{\text{encoder}})$$

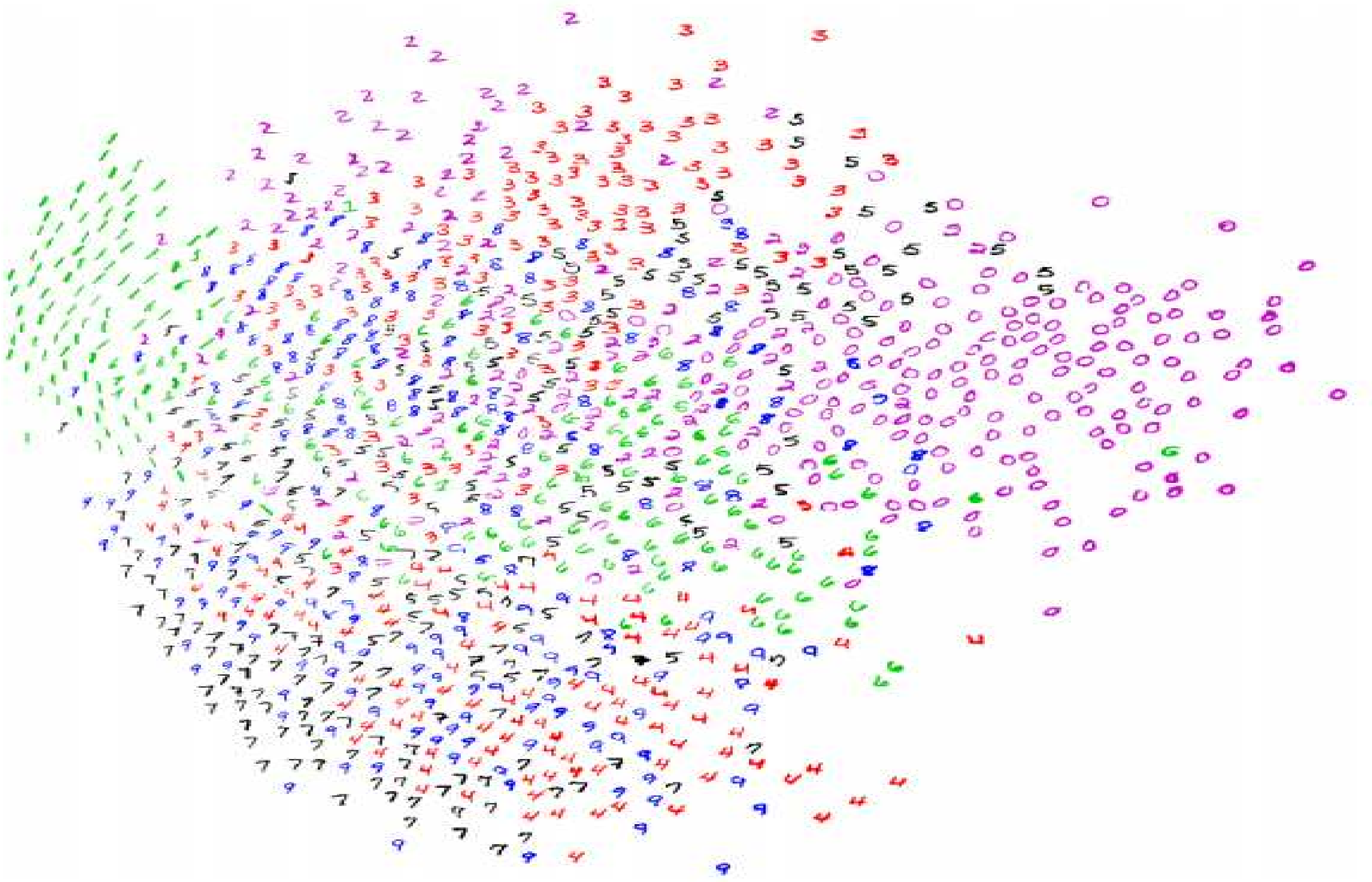
Objective function:

$$\underset{W,b_1,b_2}{argmin} \quad H = \frac{1}{2N} * \sum_{n=1}^{N}\sum_{m=1}^{M}(\hat{x}_m^{(n)} - x_m^{(n)})^2 \quad (i)$$
$$+ \frac{\lambda}{2} * \|W\|_F^2 \quad (ii)$$

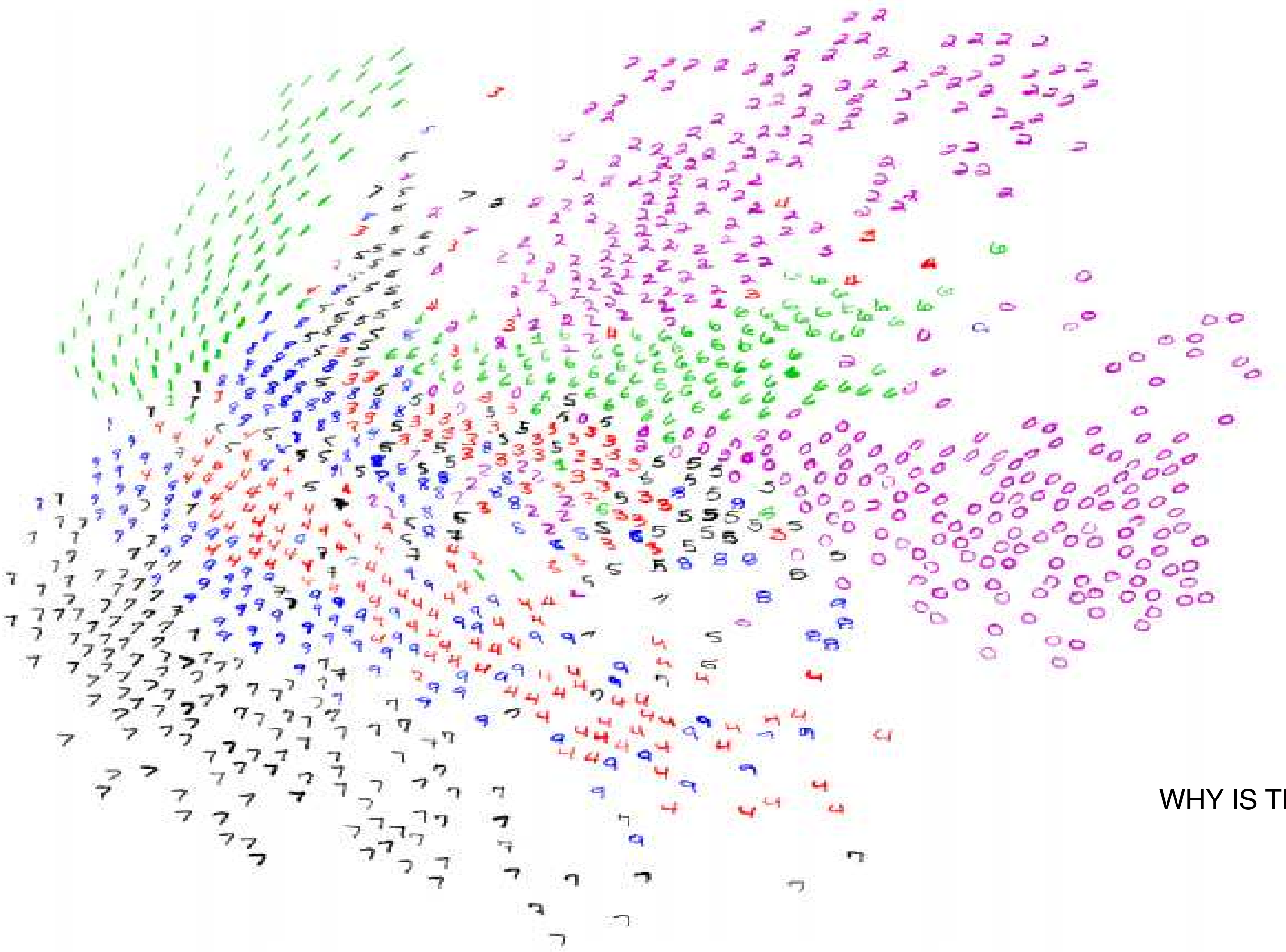
Deep Autoencoder



- Autoencoders can be stacked to form a deep network by feeding the latent representation (hidden layer) of one auto-encoder as the input layer of another autoencoder



Visualization of the 2-D codes produced 2-D PCA



WHY IS THE 1000 THERE??????

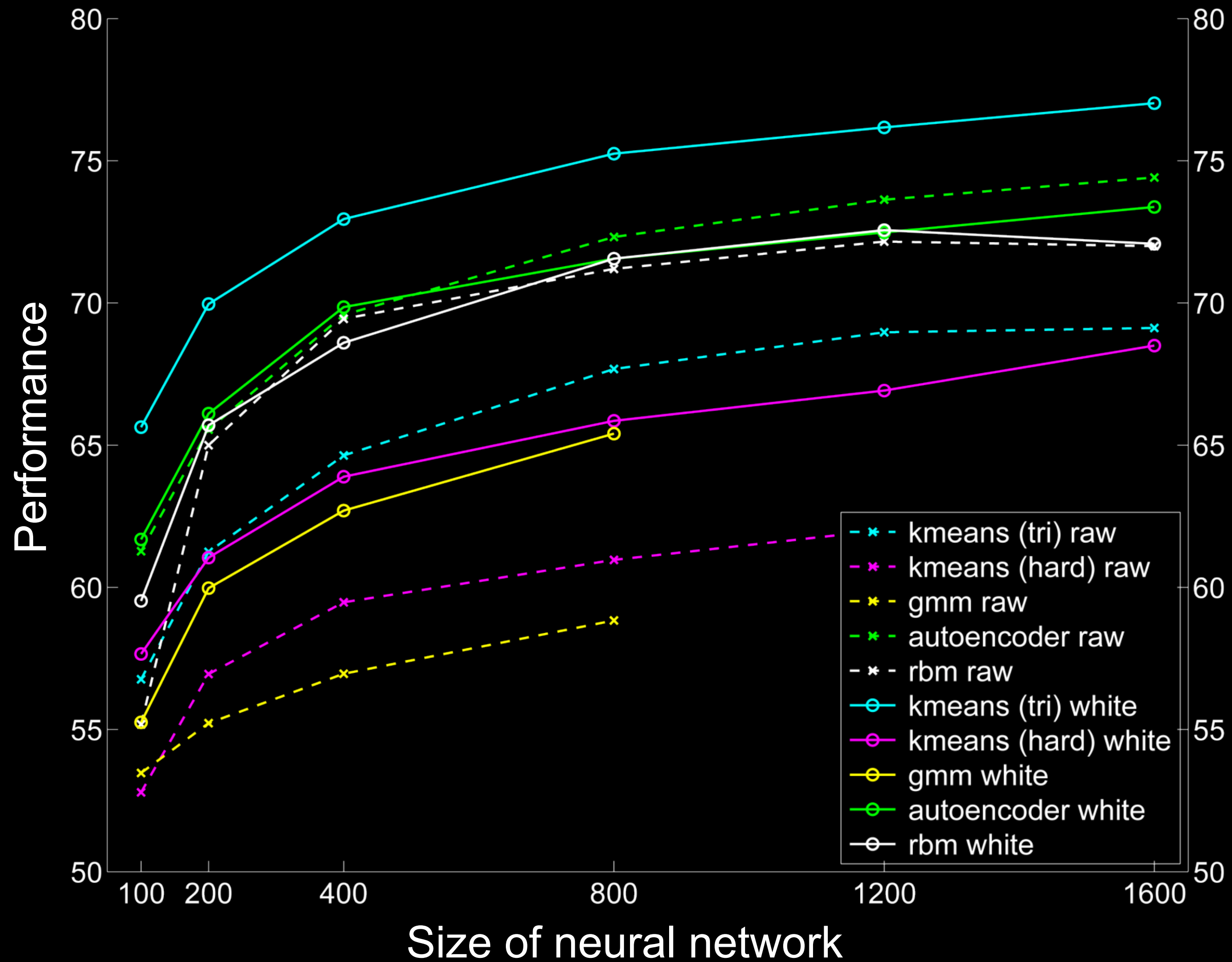
Visualization of the 2-D codes produced by a 784-1000-500-250-2 AutoEncoder

Applications

- Handwritten digit recognition
  - http://www.cs.toronto.edu/~hinton/adi/index.htm
- Face detection
  - https://www.youtube.com/watch?t=19&v=bKPf_6J0Qpk
- Off-Road robot navigation
  - https://www.youtube.com/watch?v=GLgX8ku5TOQ

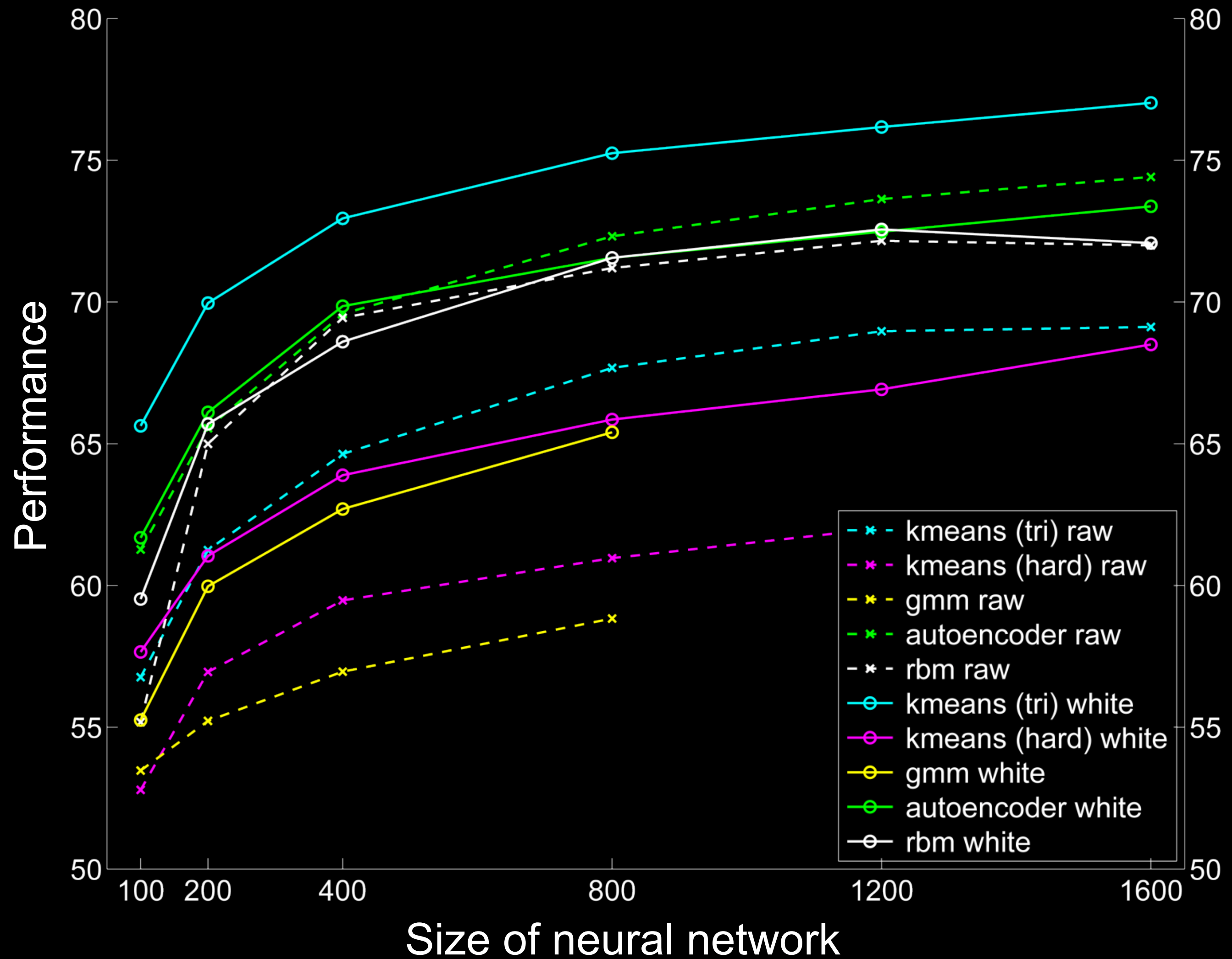
Questions?

Bigger is better



[Adam Coates]

Bigger is better



[Adam Coates]