Neural Networks

Shikui Tu

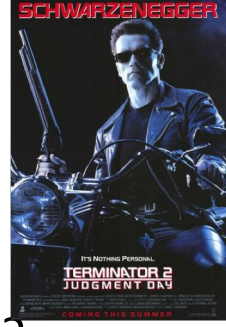
Department of Computer Science and Engineering, Shanghai Jiao Tong University

2018-05-10

Outline

- **Overview of neural networks**
- Perceptron model
- Neural network
- Training the neural networks
  - Backpropagation (BP) algorithm

Terminator 2 (1991)



- John Connor: Can you learn stuff you haven't been programmed with so you could be... you know, more human? And not such a dork all the time?
- The Terminator: My CPU is a neural net processor; a learning computer. But Skynet pre-sets the switch to read-only when we're sent out alone.
- Sarah Connor: Doesn't want you doing too much thinking, huh?
- The Terminator: No.

Switch: weights

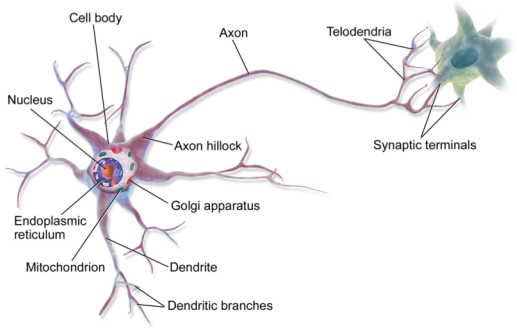
……

- The Terminator: The Skynet Funding Bill is passed. The system goes on-line August 4th, 1997. Human decisions are removed from strategic defense. Skynet begins to learn at a geometric rate. It becomes self-aware at 2:14 a.m. Eastern time, August 29th. In a panic, they try to pull the plug.
- Sarah Connor: Skynet fights back.
- The Terminator: Yes. It launches its missiles against the targets in Russia.
- John Connor: Why attack Russia? Aren't they our friends now?
- The Terminator: Because Skynet knows the Russian counter-attack will eliminate its enemies over here.

http://www.imdb.com/title/tt0103064/quotes

Neurons in Biology

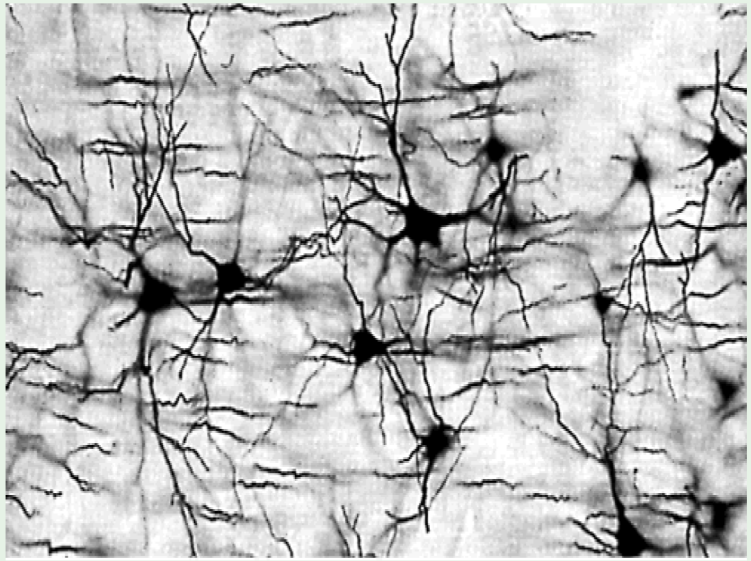
- The human brain has 100,000,000,000 neurons
- On average, each neuron is connected to 1000 other neurons
- Impulses arriving simultaneously are added together
- Sufficiently strong impulses lead to the generation of an electrical discharge (an action potential, a 'nerve impulse').
- The action potential then forms the input to the next neuron in the network.

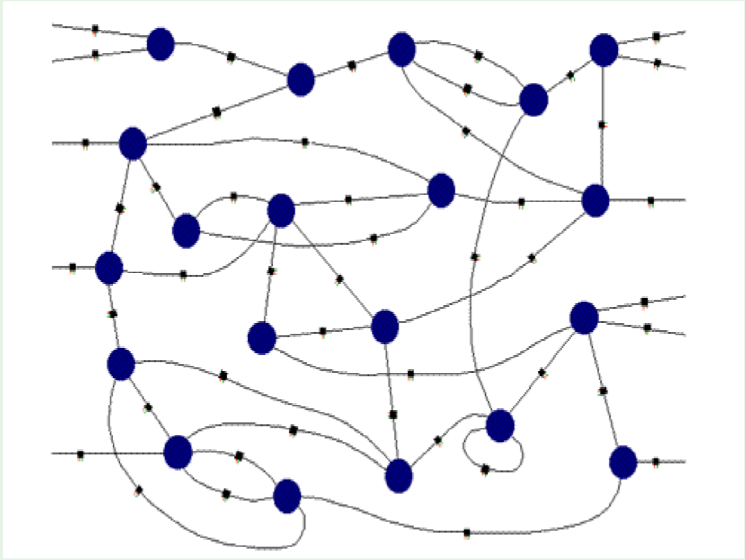


http://www.bris.ac.uk/synaptic/basics/basics-1.html

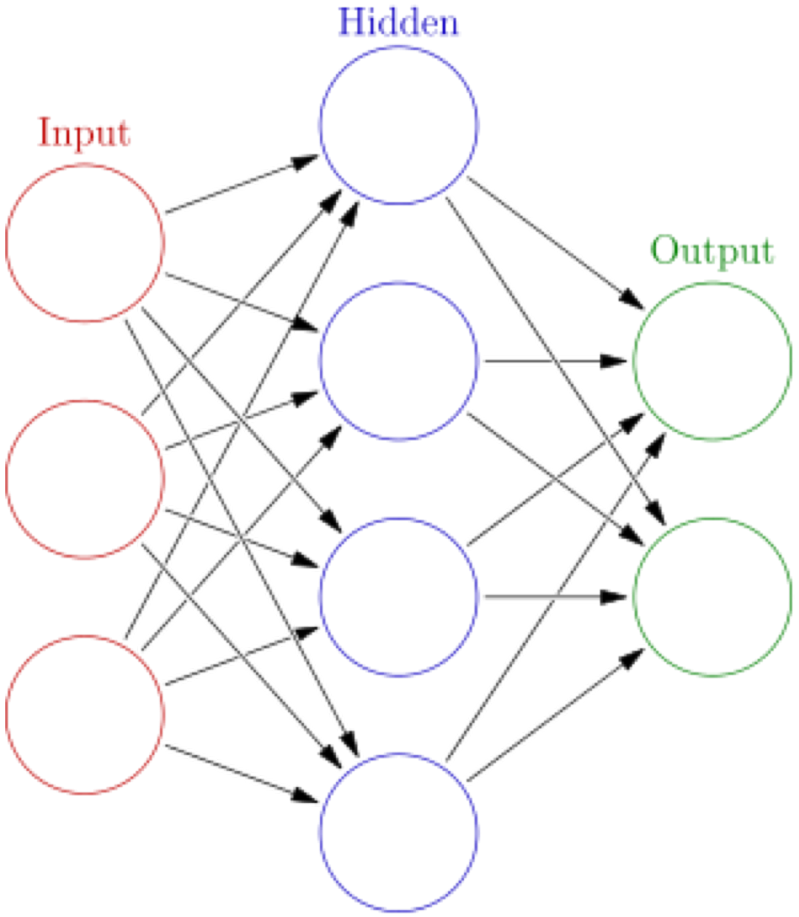
Biological inspiration

biological function $\longrightarrow$ biological structure



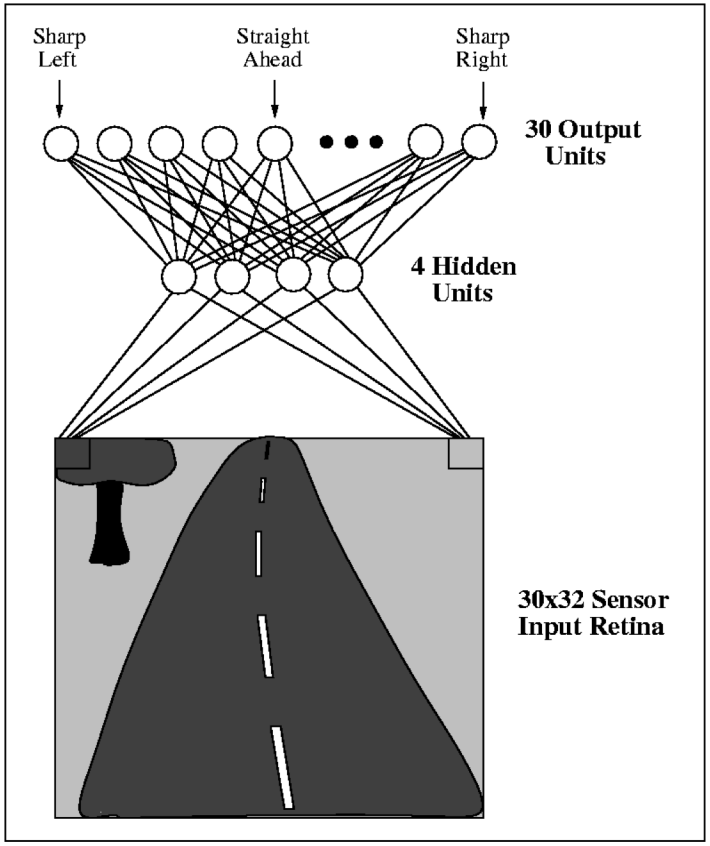


Neural Networks in Machine Learning



Applications

- ALVINN (Autonomous Land Vehicle In a Neural Network)



1990s

http://ftp.utcluj.ro/pub/docs/imaging/Autonomous_driving/Articole%20sortate/CThorpe/ALVINN%20Project%20Home%20Page.htm

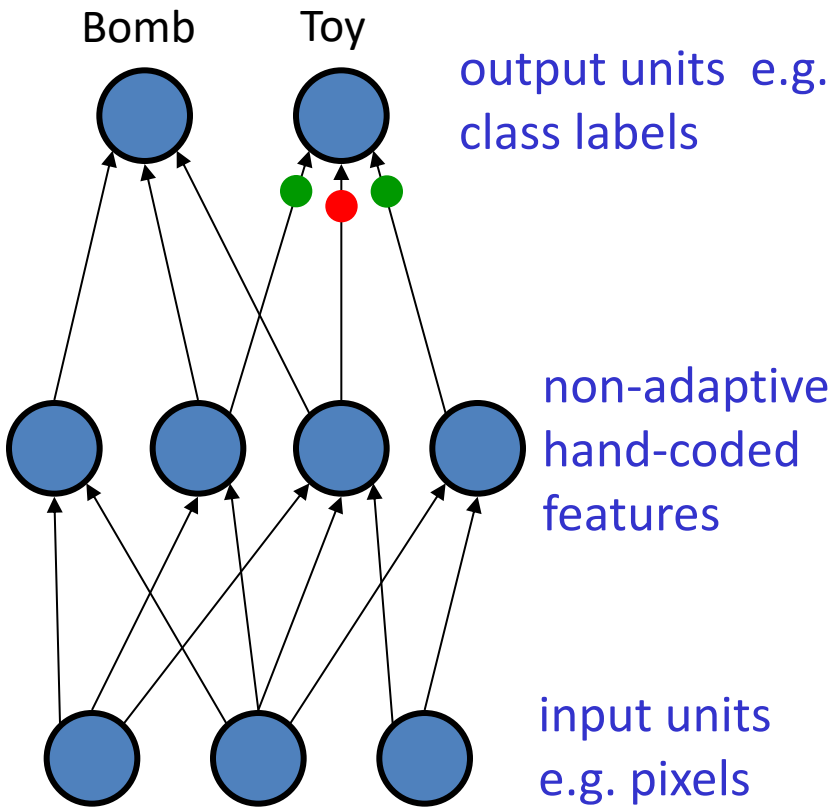
The main aim of neural networks

- People are much better than computers at recognizing patterns. How do they do it?
  - Neurons in the perceptual system represent features of the sensory input.
  - The brain learns to extract many layers of features. Features in one layer represent combinations of simpler features in the layer below.
- Can we train computers to extract many layers of features by mimicking the way the brain does it?
  - Nobody knows how the brain does it, so this requires both engineering insights and scientific discoveries.

From G. E. Hinton

First generation neural networks

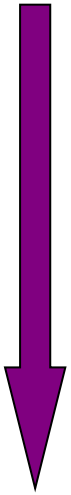
- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
  - There was a neat learning algorithm for adjusting the weights.
  - But perceptrons are fundamentally limited in what they can learn to do.



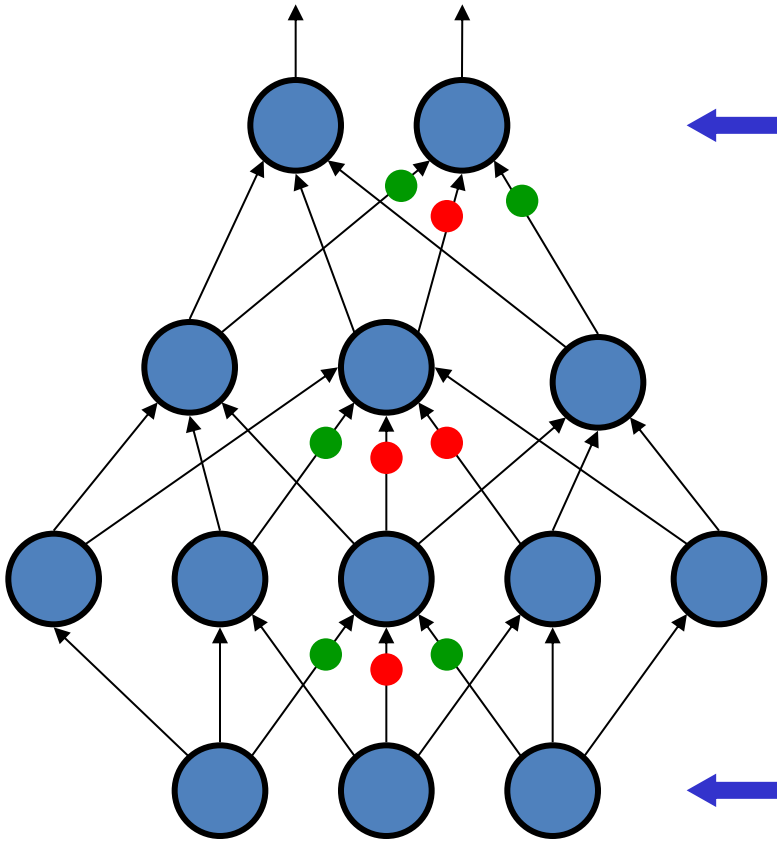
Sketch of a typical perceptron from the 1960's

Second generation neural networks (~1985)

Back-propagate error signal to get derivatives for learning



Compare outputs with correct answer to get error signal



outputs

hidden layers

input vector

A temporary digression

- Vapnik and his co-workers developed a very clever type of perceptron called a Support Vector Machine.
  - Instead of hand-coding the layer of non-adaptive features, each training example is used to create a new feature using a fixed recipe.
    - The feature computes how similar a test example is to that training example.
  - Then a clever optimization technique is used to select the best subset of the features and to decide how to weight each feature when classifying a test case.
    - But its just a perceptron and has all the same limitations.
- In the 1990's, many researchers abandoned neural networks with multiple adaptive hidden layers because Support Vector Machines worked better.

Outline

- Overview of neural networks

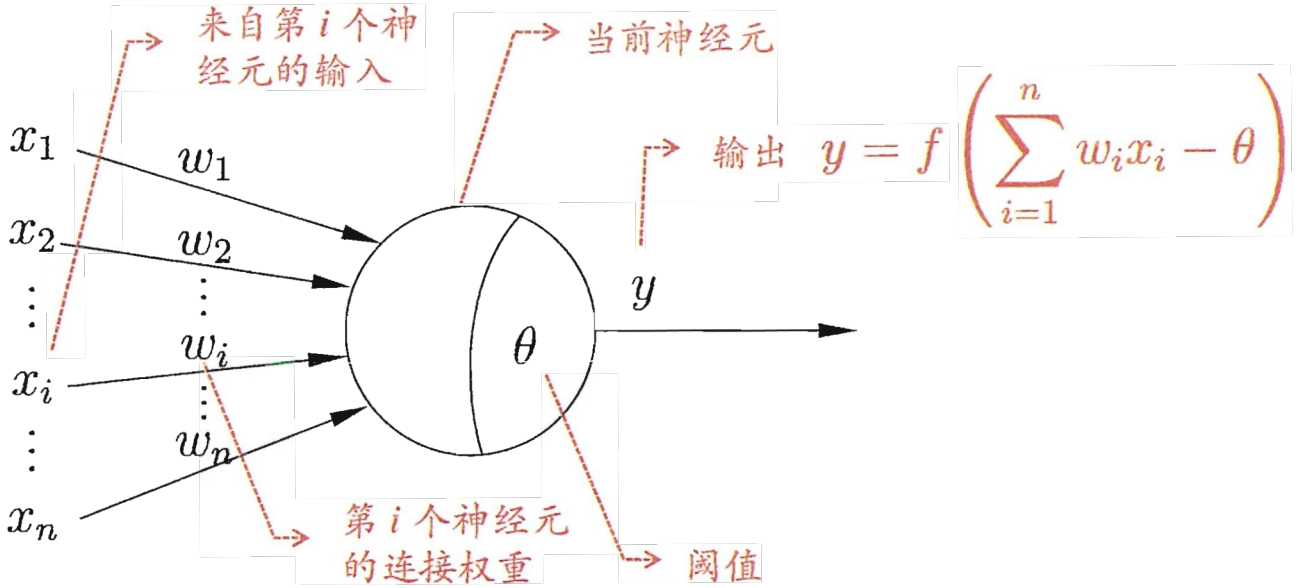
- **Perceptron model**

- Neural network

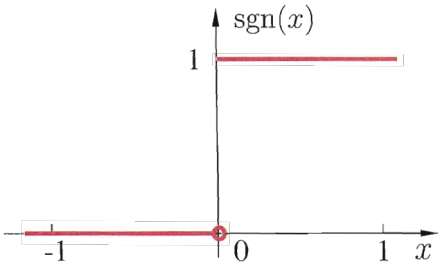
- Training the neural networks
  - Backpropagation (BP) algorithm

M-P neuron model

[McCulloch and Pitts, 1943]

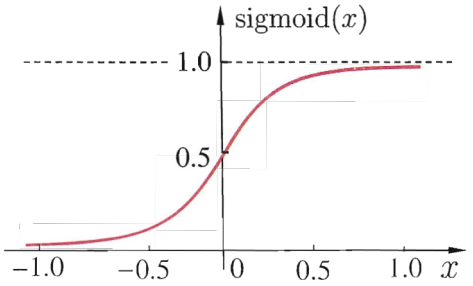


Step function



$$\mathrm{sgn}(x) = \begin{cases} 1, & x \geqslant 0; \\ 0, & x < 0. \end{cases}$$

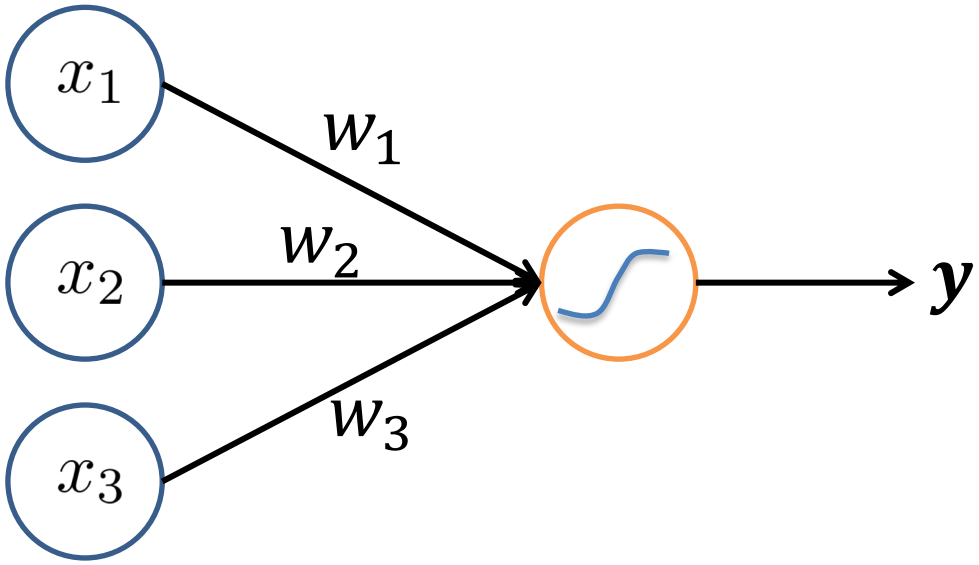
Sigmoid function



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

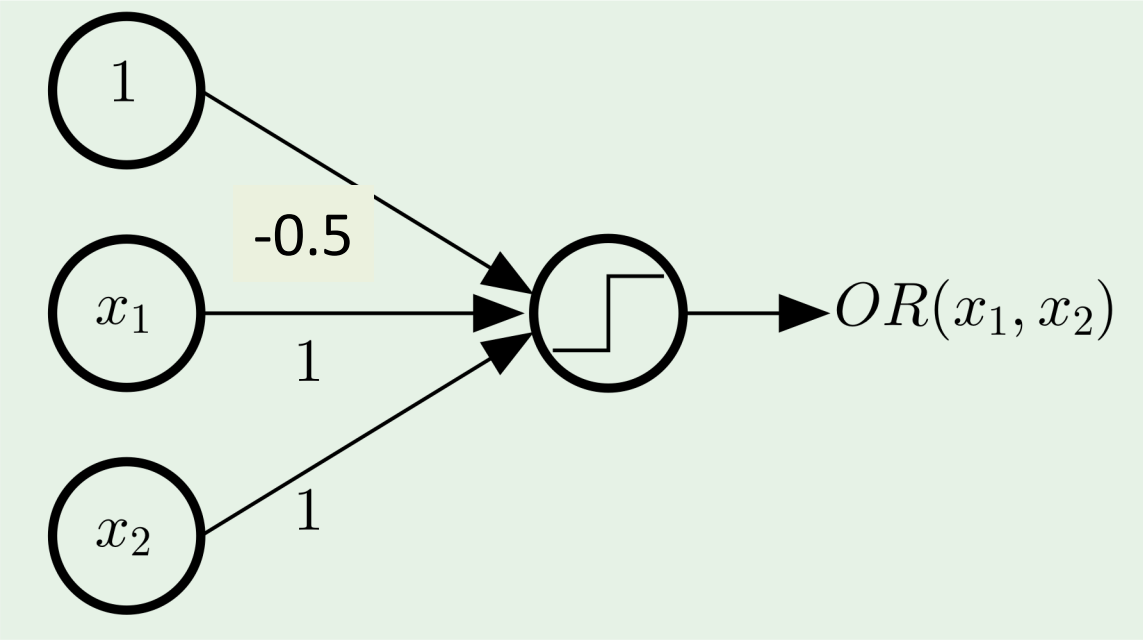
Perceptron

- Input layer + M-P neuron (output)

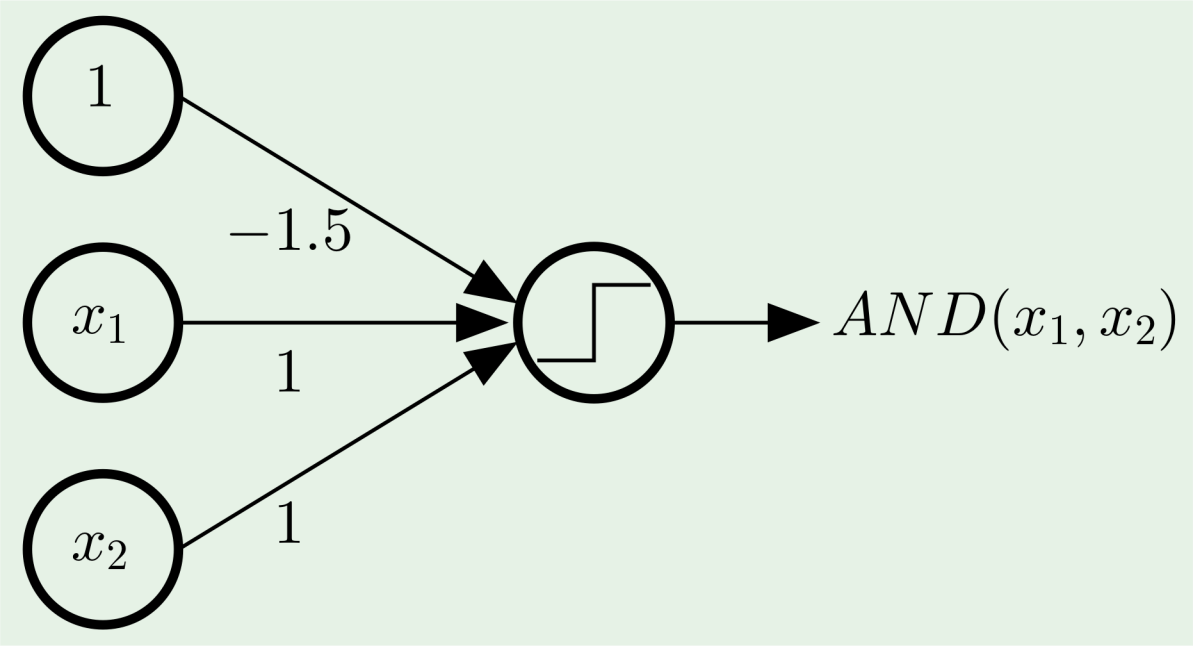


What can it do?

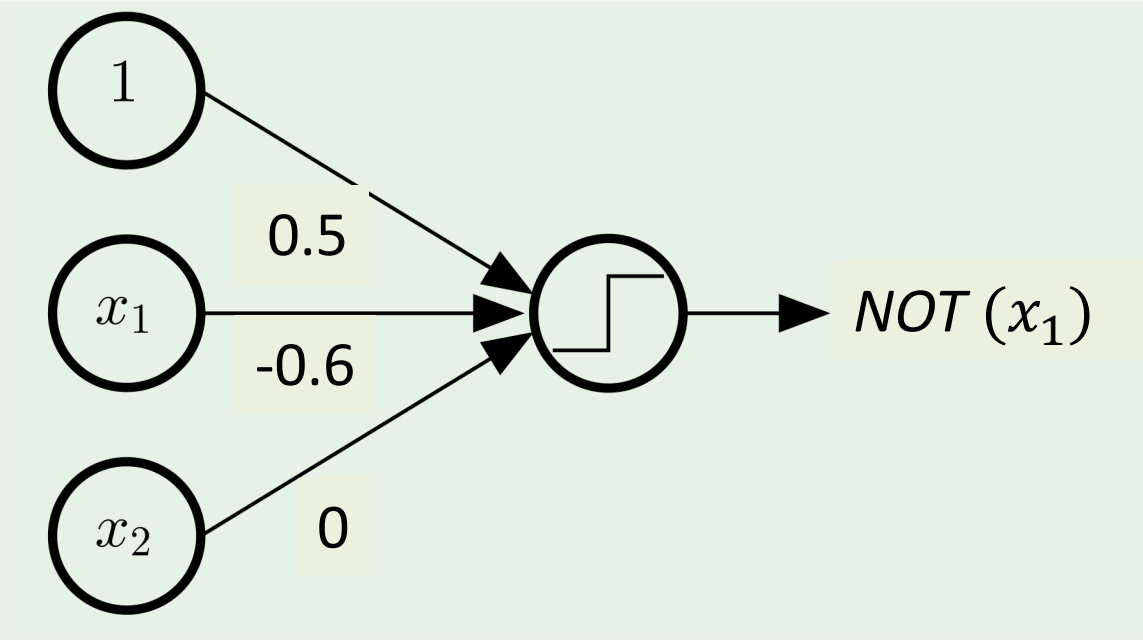
OR by perceptron



AND by perceptron

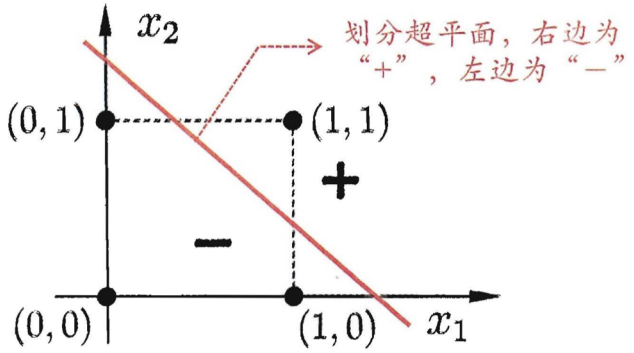


NOT by perceptron

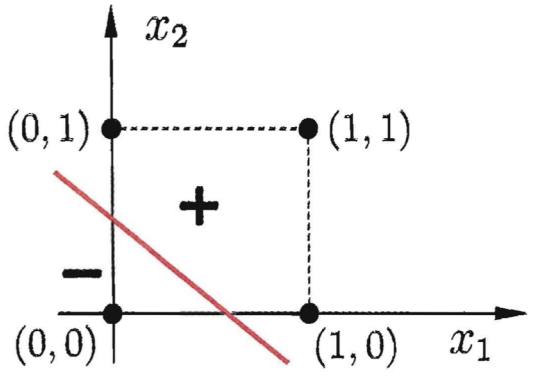


A single perceptron could not implement XOR

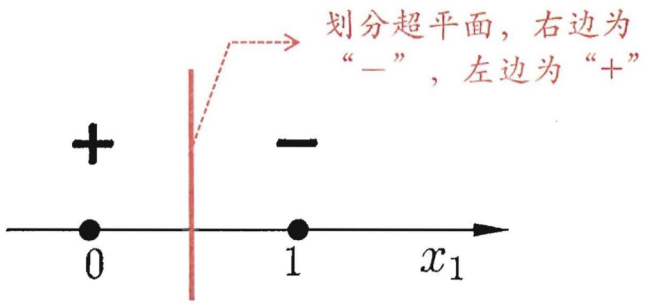
$$y(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + w_0)$$ Linear classifier



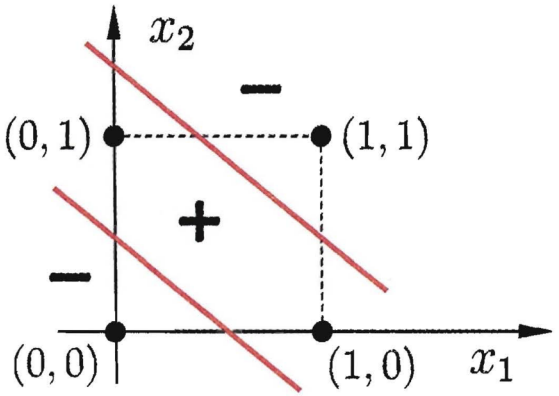
(a) “与”问题 $(x_1 \wedge x_2)$



(b) “或”问题 $(x_1 \vee x_2)$

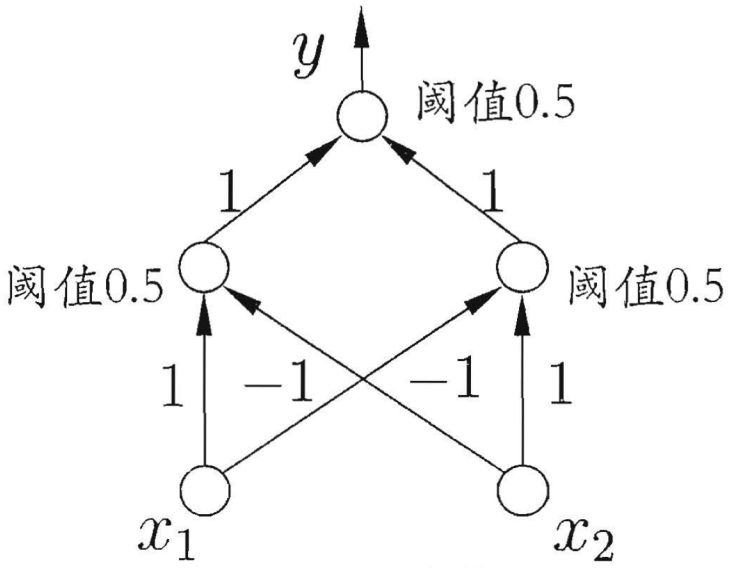


(c) “非”问题 $(\neg x_1)$

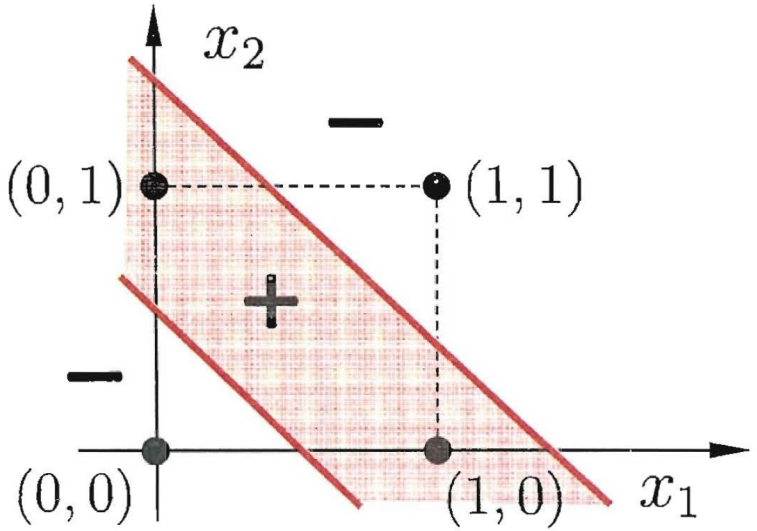


(d) “异或”问题 $(x_1 \oplus x_2)$

A two-layer perceptron for XOR



(a) 网络结构



(b) 分类区域

Perceptron learning

$$w_i \leftarrow w_i + \Delta w_i \ ,$$

$$\Delta w_i = \eta(y - \hat{y})x_i \ ,$$

Perceptron convergence theorem: if there exists an exact solution (which means the training data is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

In practice, the number of steps required to achieve convergence could be substantial. Until convergence is achieved, we are not able to distinguish between a nonseparable problem and one that is simply slow to converge.

The perceptron algorithm does not converge when the data are not linearly separable.

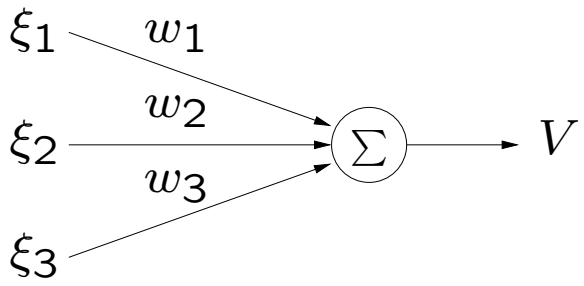
Recall:

The Hebbian Neuron

A computational system which implements Hebbian learning.

Let's assume a linear unit; experiment shows this is largely sufficient:

$$V = \sum_j w_j \xi_j = \bar{w}^T \bar{\xi} \tag{2}$$



Plain Hebbian learning:

$$\Delta \bar{w} = \eta \, V \, \bar{\xi} \tag{3}$$

A adaptive learning rule

The Perceptron algorithm

More general form of a linear classifier

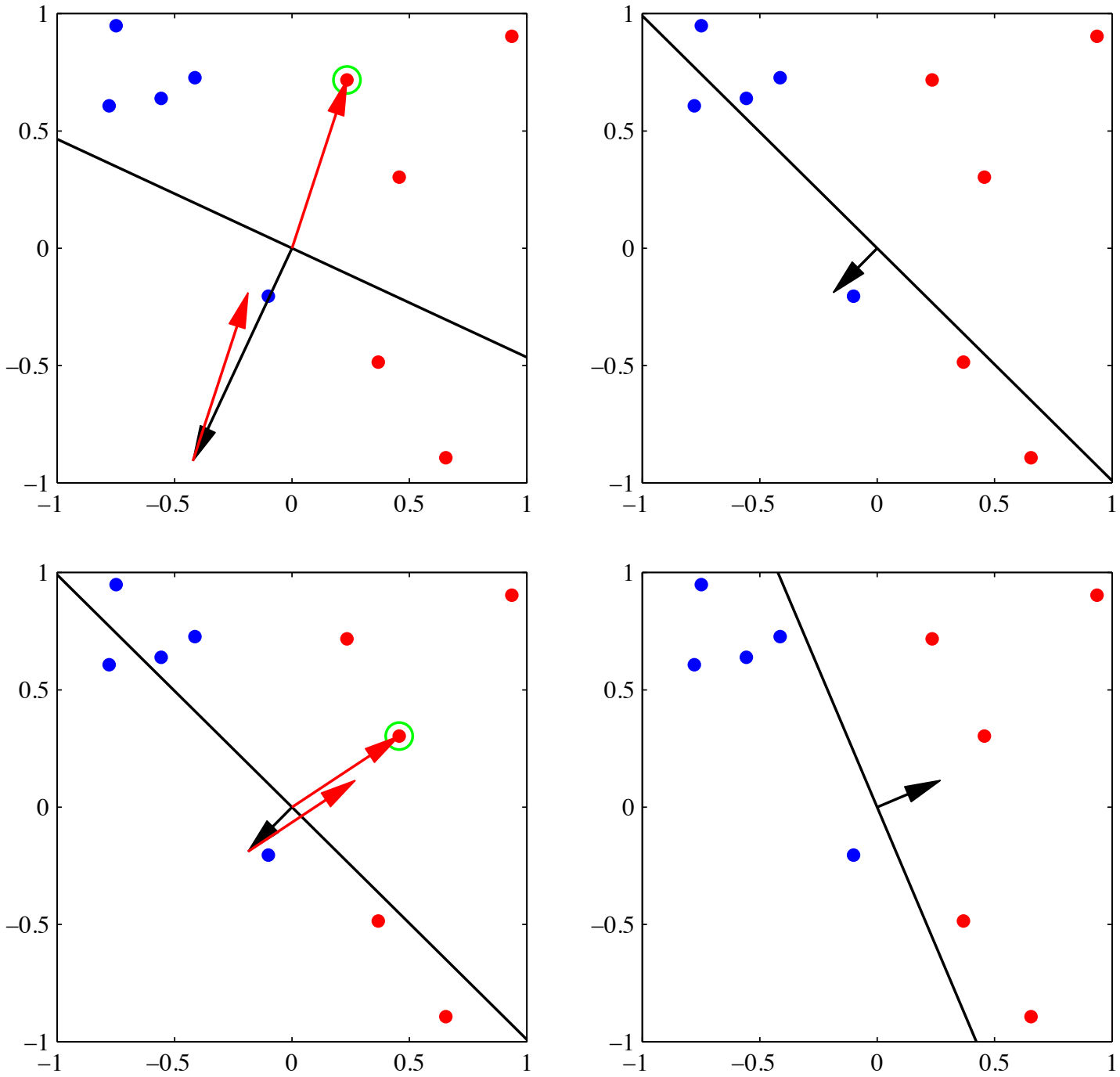
$$y(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})\right)$$

where $\boldsymbol{\phi}(\mathbf{x})$ is the feature vector of $\mathbf{x}$ (recall the basis functions in linear regression), and typically includes a bias component $\phi_0(\mathbf{x}) = 1$.

Given a set of samples $\{\mathbf{x}_n, t_n\}_{n=1}^N,\ \mathbf{x}_n \in \mathbb{R}^D,\ t_n \in \{-1, 1\}$, our goal is to find the optimal parameter $\mathbf{w}$ to minimize the training error

$$E_p(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^T \boldsymbol{\phi}_n t_n$$

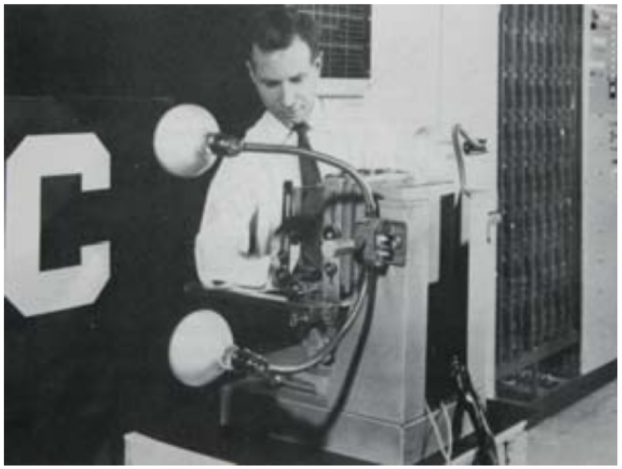
where $\boldsymbol{\phi}_n = \boldsymbol{\phi}(\mathbf{x}_n)$ and $\mathcal{M}$ denotes the set of all misclassified patterns.



PRML

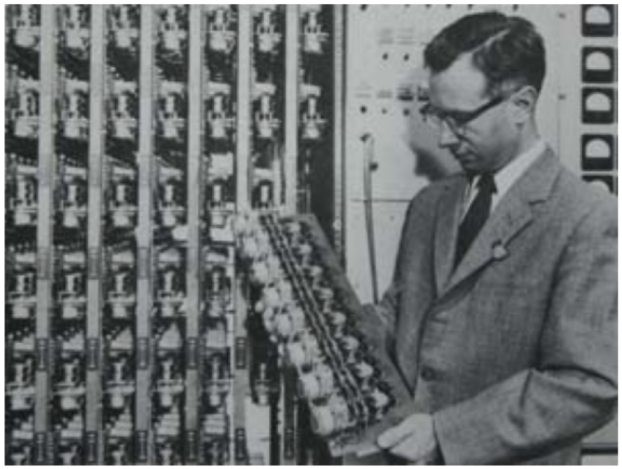
Perceptron in Action

Frank Rosenblatt



Input: a simple camera system (400 pixel image)

Patch board: different configurations of input features

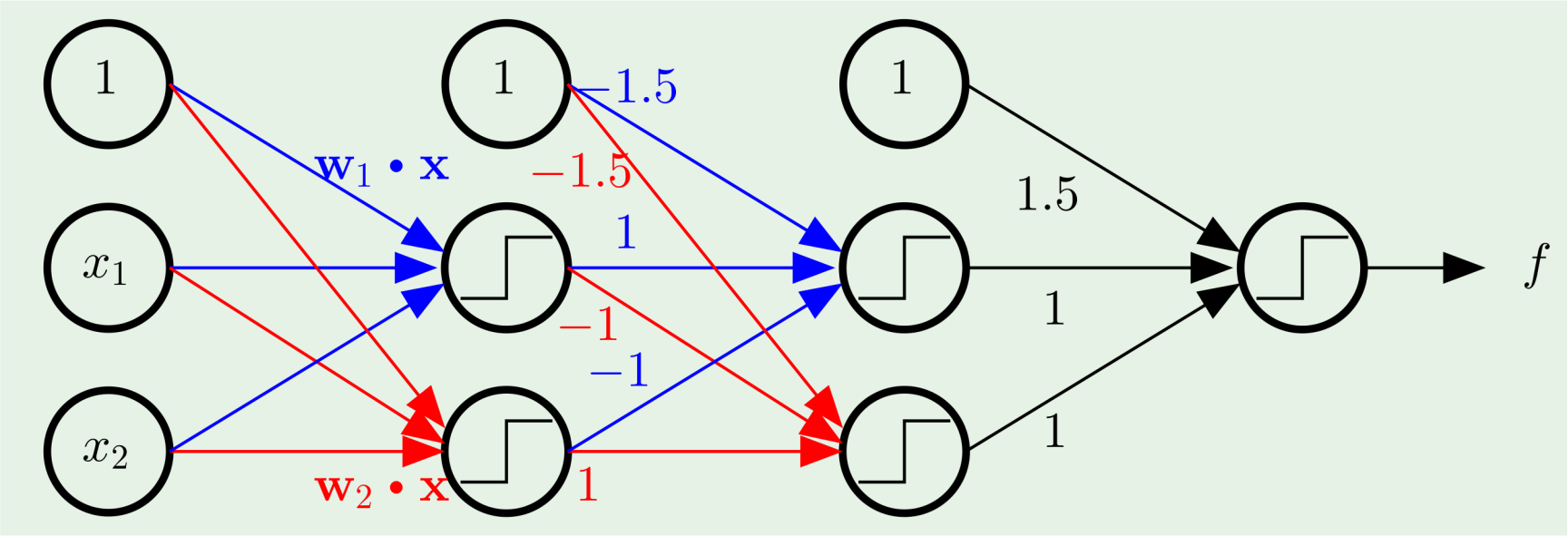


Racks of adaptive weights: Each weight was implemented using a rotary variable resistor, driven by an electric motor thereby allowing the value of the weight to be adjusted automatically by the learning algorithm.

Outline

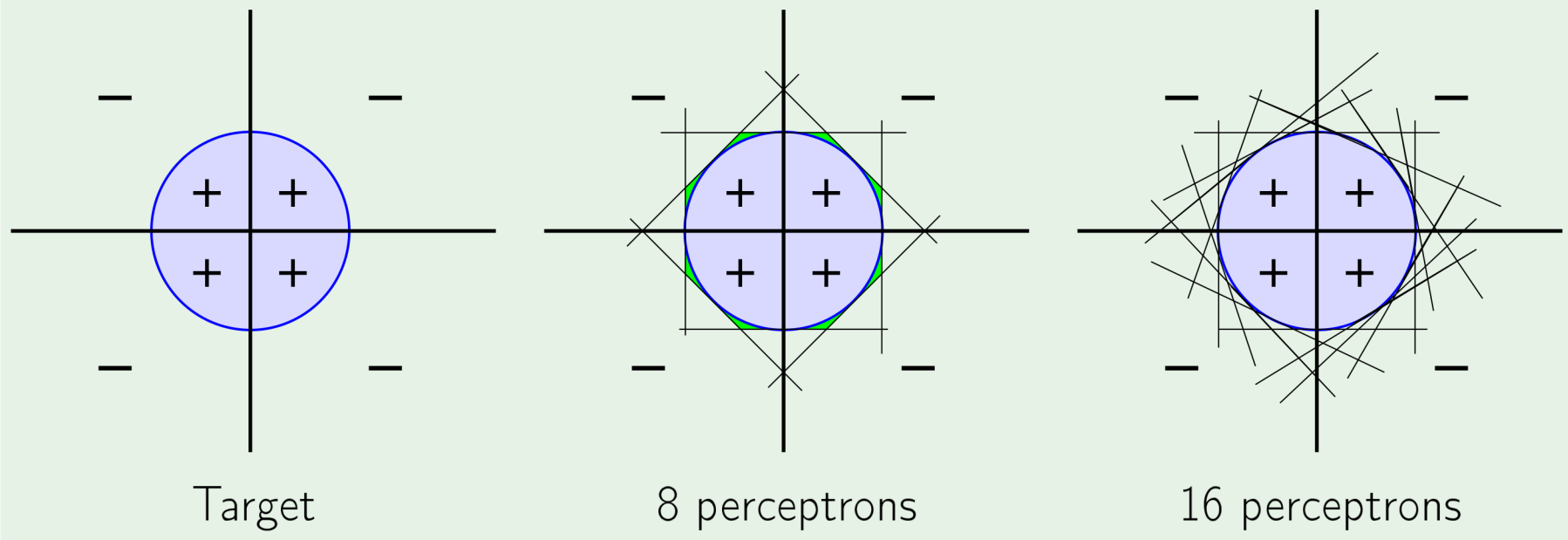
- Overview of neural networks
- Perceptron model
- **Neural network**
- Training the neural networks
  - Backpropagation (BP) algorithm

The multilayer perceptron



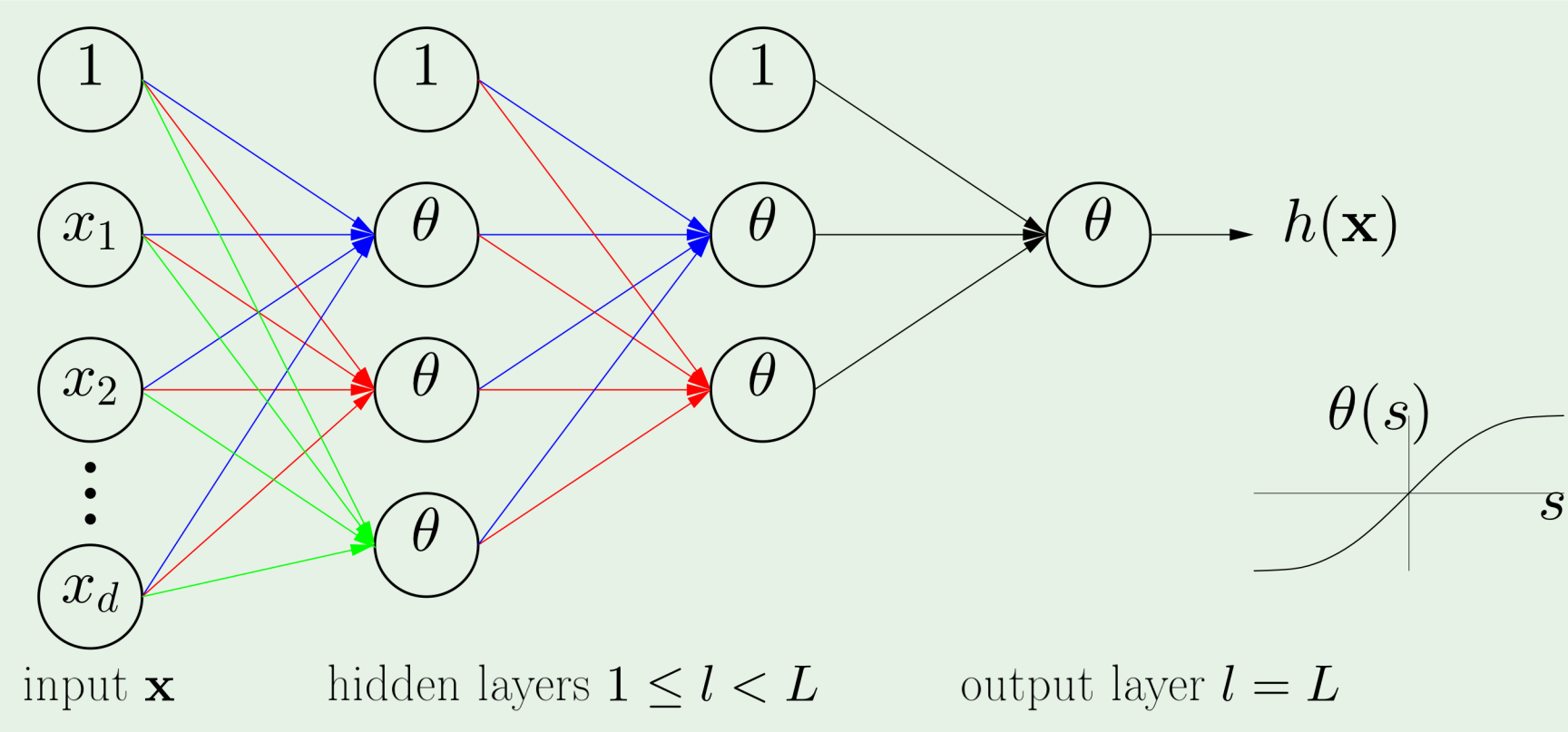
3 layers, feedforward

The multilayer perceptron is powerful



But not good for generalization and optimization

Neural networks



Functions by neural networks

Neural network is defined by recursively constructing a nonlinear function over linear combination of inputs.

For input $\mathbf{x} = (x_1, \cdots, x_D) \in \mathbb{R}^D$, we construct M linear combinations

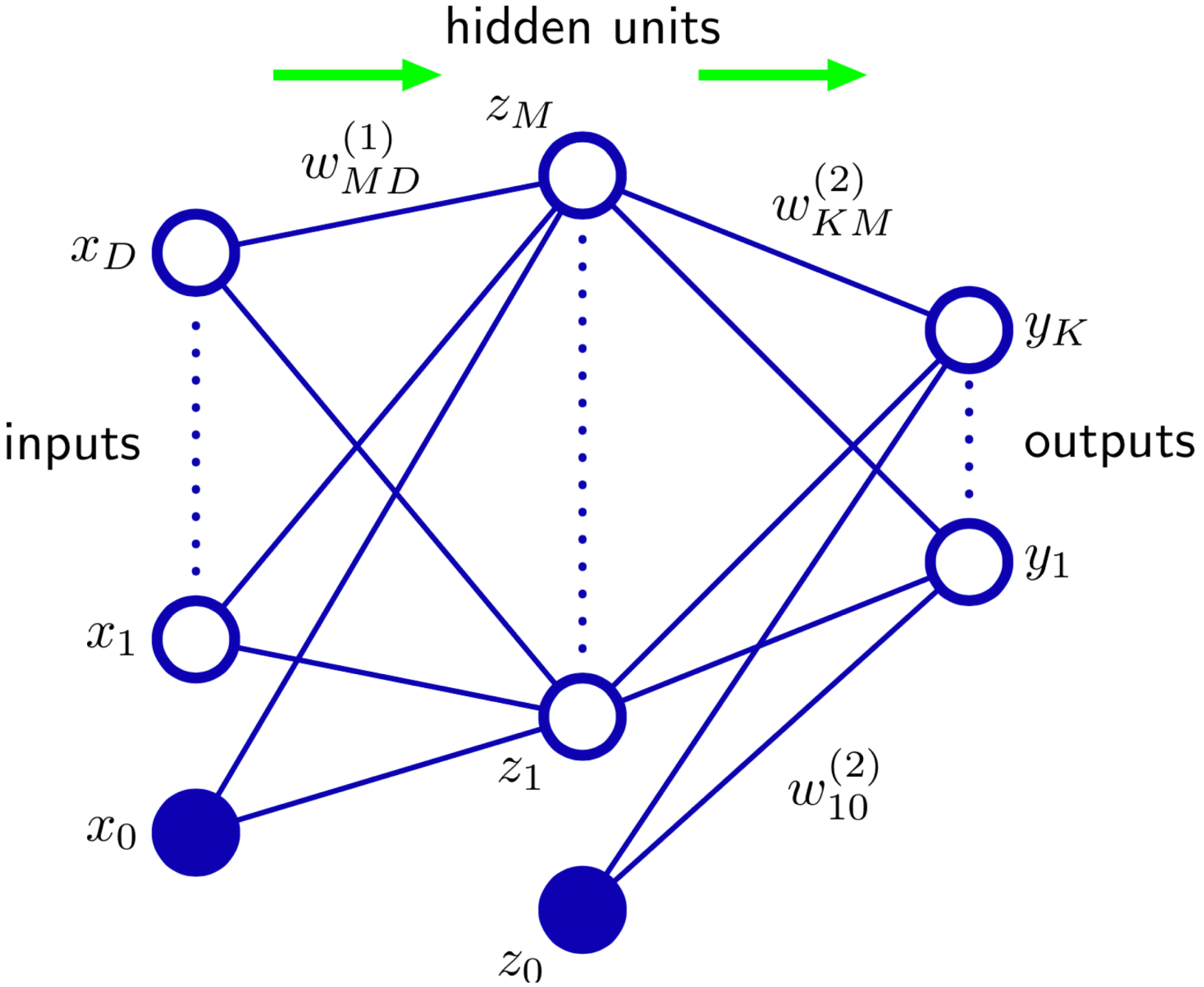
$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \quad j = 1, \cdots, M$$

where the superscript (1) indicates the corresponding parameters are in the first layer of the network. $w_{ji}^{(1)}$ are *weights* and $w_{j0}^{(0)}$ are biases. a_j are known as *activations*.

Each activation is then transformed using a differentiable, nonlinear *activation function* $h(\cdot)$ to give

$$z_j = h(a_j), \quad j = 1, \cdots, M$$

which are called *hidden units*. The nonlinear functions $h(\cdot)$ are often sigmoid functions such as logistic and tanh.


hidden units
z_M
$w^{(1)}_{MD}$
$w^{(2)}_{KM}$
x_D
y_K
inputs
outputs
y_1
x_1
z_1
$w^{(2)}_{10}$
x_0
z_0

PRML

Recursive Nature

- Activations of hidden units:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \qquad j = 1, ..., M$$

- Each activation is transformed using a differentiable, nonlinear activation function

$$z_j = h(a_j) \qquad j = 1, ..., M$$

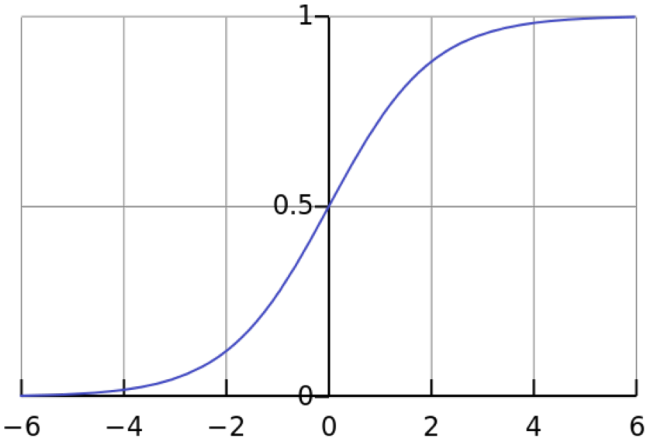
- The hidden units can be linearly combined to give output unit activation

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \qquad k = 1, ..., K$$

- Output layer

- Sigmoid function

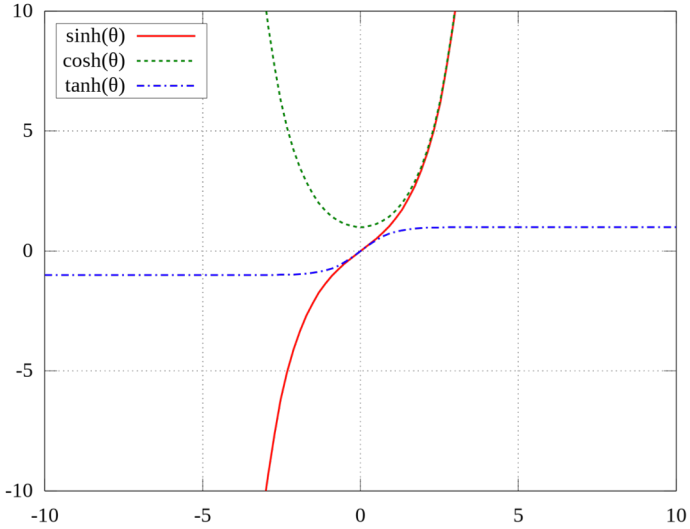
$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

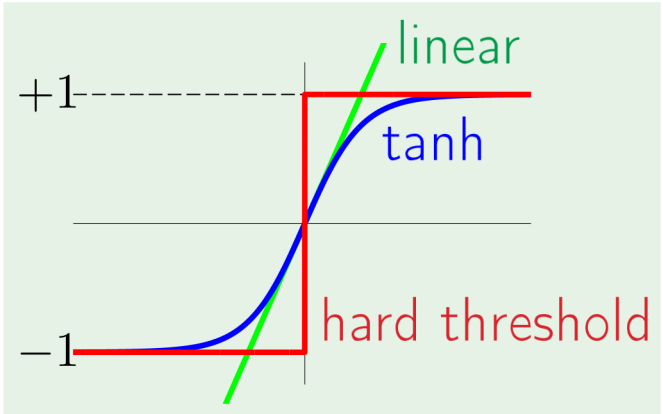


- tanh function

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} =$$

$$= \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$





Output layer for classification

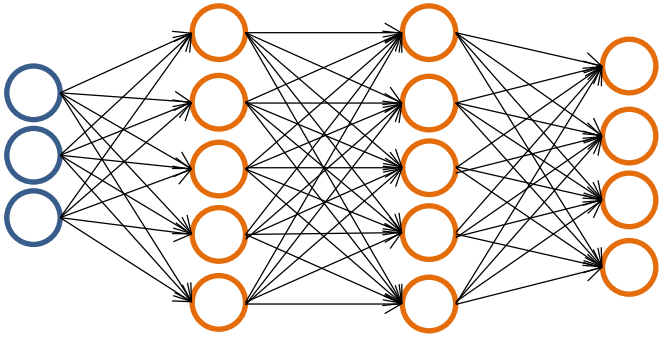
Multi-class classification

Pedestrian

Car

Motorcycle

Truck



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Forms of network output

- The output unit activations are transformed to the network output y_k
  - Regression problems: $y_k = a_k$
  - Multiple binary classification problems, each output unit activation can be logistic sigmoid function: $y_k = \sigma(a_k)$
  - Multi-class problems, soft max function:

$$\sigma(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}}$$

Overall neural network function

- Overall network function (using sigmoidal activation function)

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{k=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

  where weight parameters are grouped into **w**

- Neural network is a function that maps **x** to **y**, controlled by adjustable **w**.

Function with a dummy variable

- We can introduce a dummy variable x_0=1 to simply the notation
- The first layer activation:

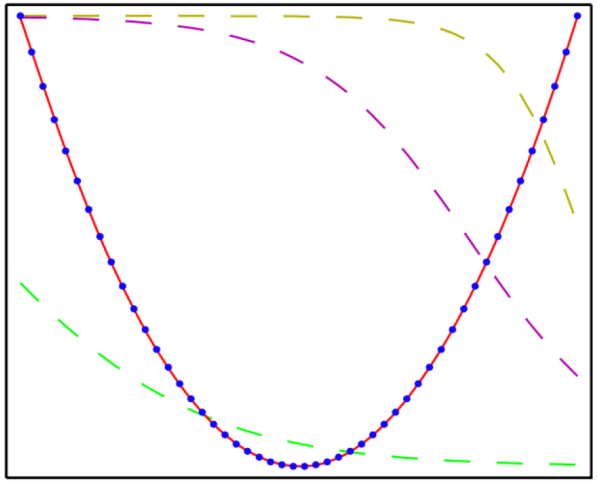
$$a_j = \sum_{i=0}^{M} w_{ji}^{(1)} x_j$$

- Neural network function

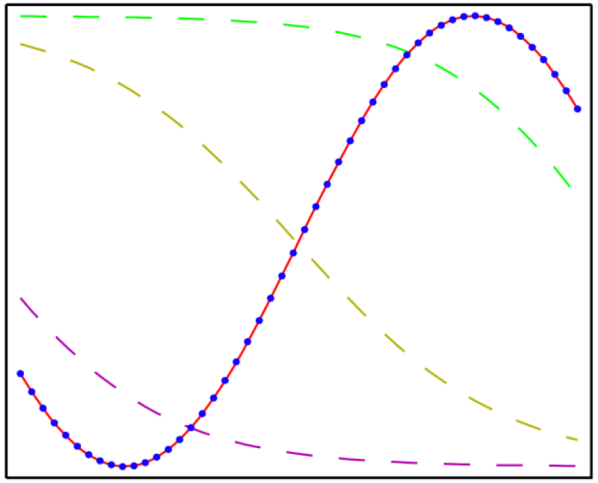
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{k=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right)\right)$$

A Reassuring Theorem

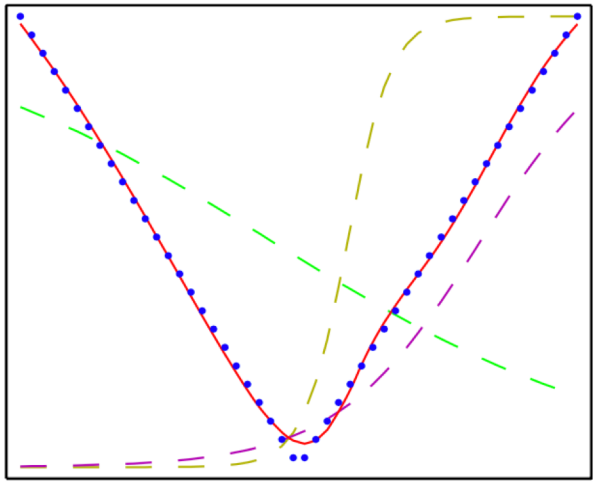
- *A two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided that the network has a sufficiently large number of hidden units*



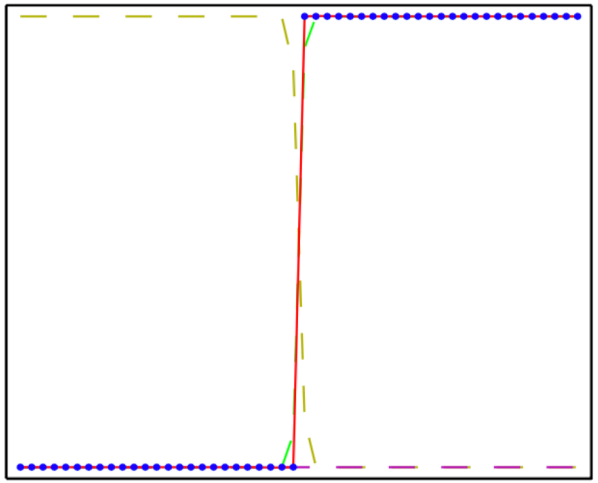
(a) $f(x) = x^2$



(b) $f(x) = \sin(x)$



(c) $f(x) = |x|$



(d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function

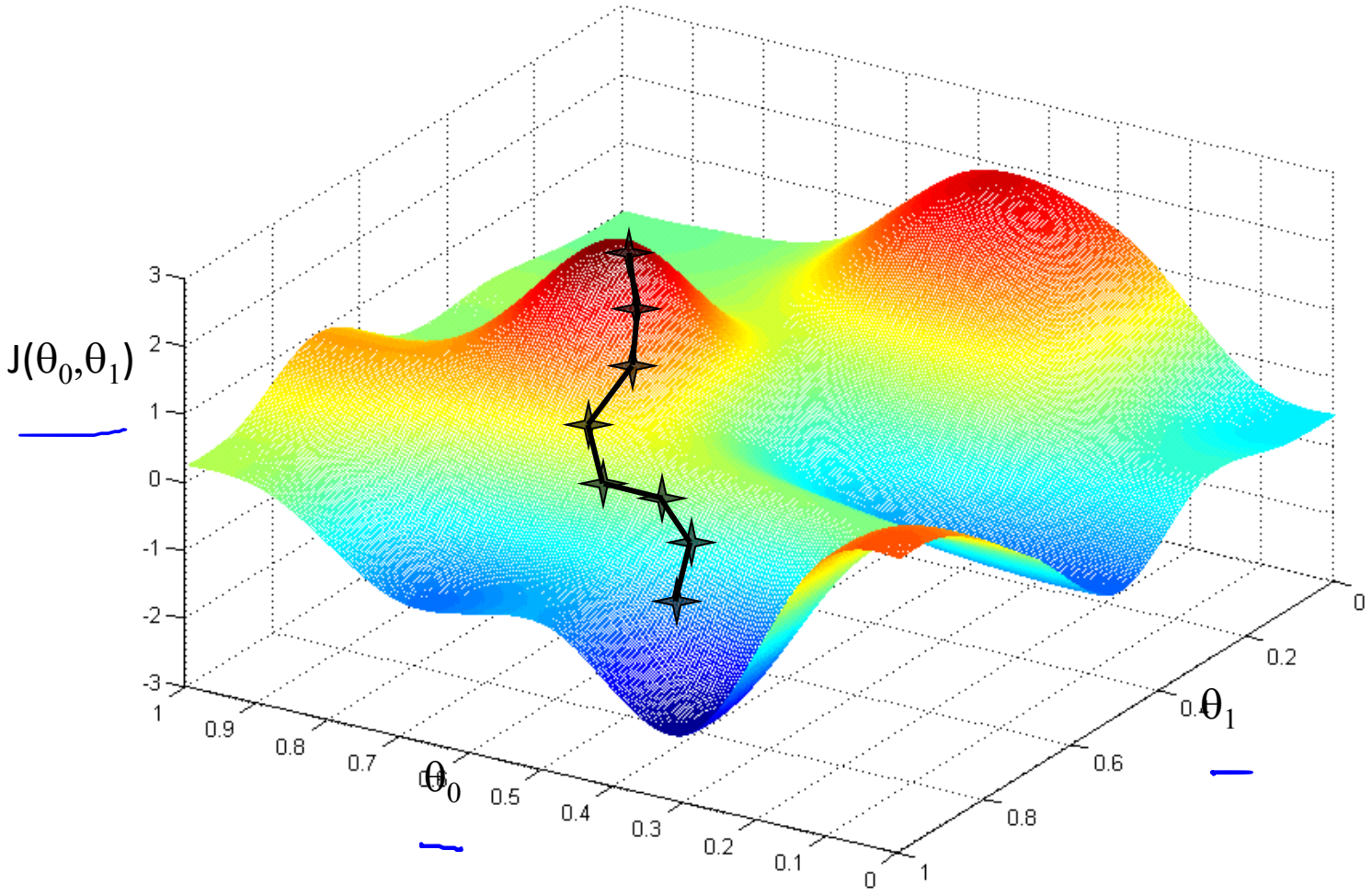
50 training points

These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

Outline

- Overview of neural networks
- Perceptron model
- Neural network
- Training the neural networks
  - Backpropagation (BP) algorithm

Gradient Descent



Ng, A. Linear Regression with One Variable [PowerPoint slides]

Error Function

- For a set of N training samples $\{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^{N}$

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

- How to evaluate the derivative for one sample pair in the error function?

$$\nabla E_n(\mathbf{w})$$

Error function for linear function

For a linear function

$$y_j = \sum_i w_{ji} x_i$$

together with a squared error function

$$E_n = \frac{1}{2} \sum_j (y_{nj} - t_{nj})^2$$

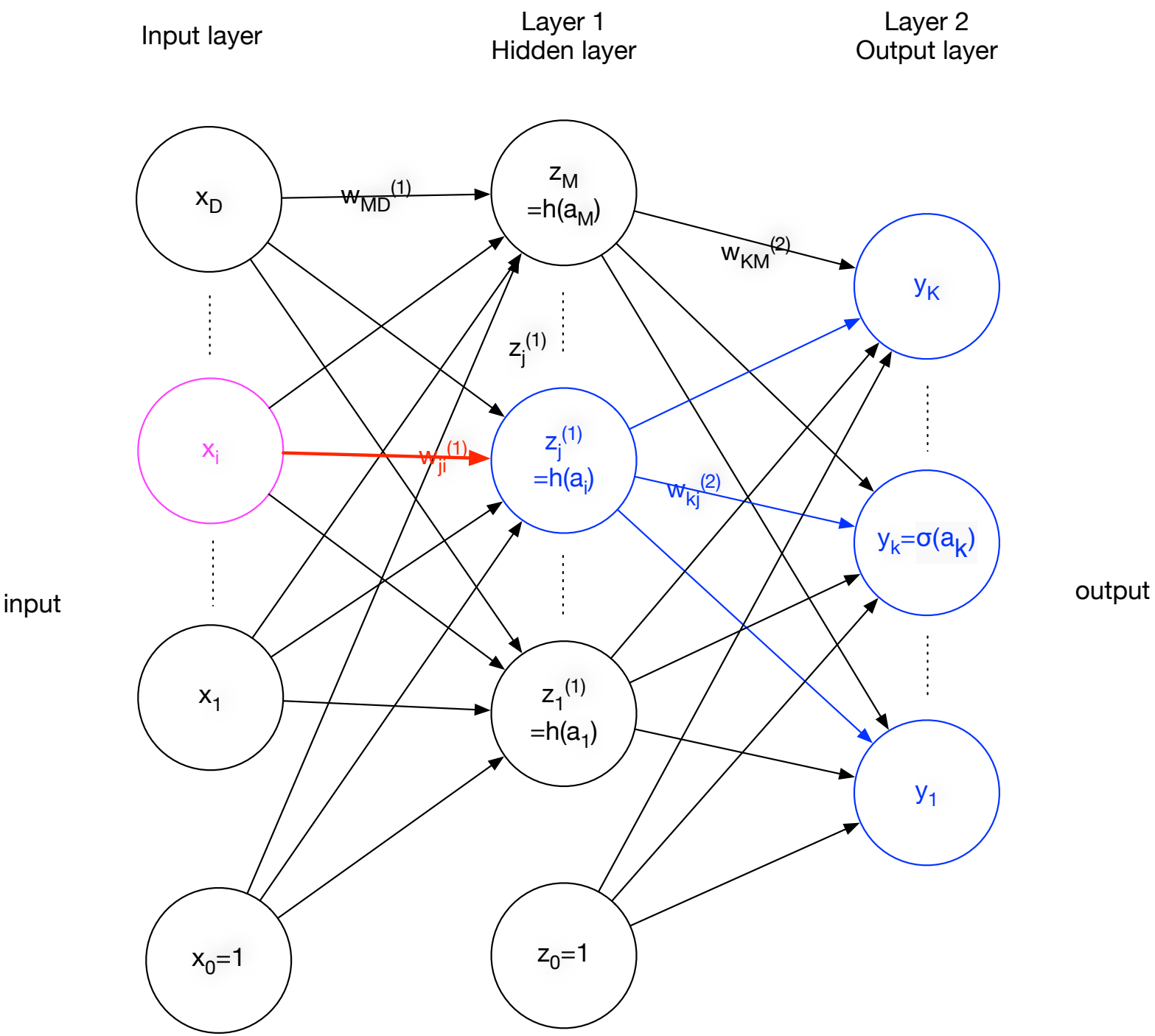
where $y_{nj} = y_j(\mathbf{x}_n, \mathbf{w})$. The gradient w.r.t. w_{ji} is

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

which can be interpreted as a *local* computation involved with the product of

- An 'error' signal $y_{nj} - t_{nj}$ associated with the *output* end of the link w_{ji}
- The variable x_{ni} associated with the *input* end of the link

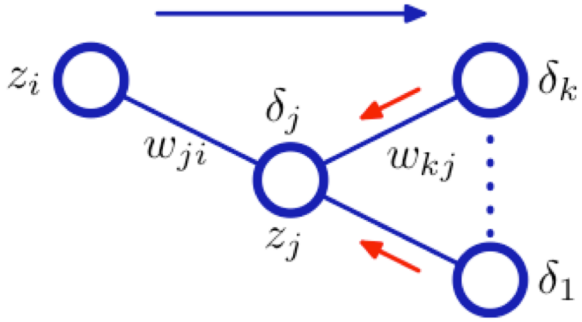
We will show that this property also holds for more complicated FFNN.


Input layer
Layer 1
Hidden layer
Layer 2
Output layer
input
output
x_D
x_i
x_1
$x_0=1$
z_M
$=h(a_M)$
$z_j^{(1)}$
$z_1^{(1)}$
$=h(a_1)$
$z_0=1$
y_K
$y_k=\sigma(a_k)$
y_1
$w_{MD}^{(1)}$
$w_{ji}^{(1)}$
$w_{KM}^{(2)}$
$w_{kj}^{(2)}$

Error backpropagation (1/3)

Note that E_n depends on the weight w_{ji} only via the summed input a_j to unit j. We can apply the chain rule for partial derivative to give

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$



Introduce a useful notation

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

where δ's are referred to as *errors*. Because $a_j = \sum_i w_{ji} z_i$, we have

$$\frac{\partial a_j}{\partial w_{ji}} = z_i$$

Therefore

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

Error backpropagation (2/3)

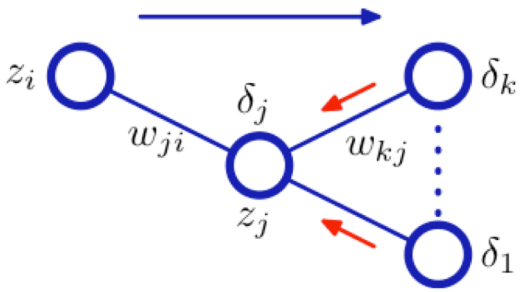
To evaluate the δ's for the hidden units at level (l), we again use the chain rule

$$\delta_j^{(l)} \equiv \frac{\partial E_n}{\partial a_j^{(l)}} = \sum_k \frac{\partial E_n}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}}$$

where the sum runs over all units k to which unit j sends connection.

Since we have defined $\delta_k^{l+1} \equiv \dfrac{\partial E_n}{\partial a_k^{(l+1)}}$, and $a_k^{(l+1)} = \sum_j w_{kj}^{(l)} h(a_j^{(l)})$, we obtain the *backpropagation* formula

$$\delta_j^{(l)} = h'(a_j^{(l)}) \sum_k w_{kj}^{(l)} \delta_k^{(l+1)}$$



Error propagation (3/3)

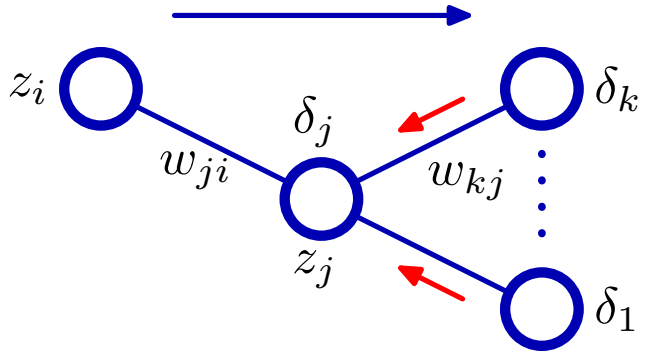
For the squared error function and for the output units, we have

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

The simplified error backpropagation formula

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

is a weighted sum of the errors.



Error propagation algorithm

- Forward propagation: use equation

$$a_k = \sum_j w_{kj} z_j = \sum_j w_{kj} h(a_j)$$

  to find the activations of all the hidden and output units

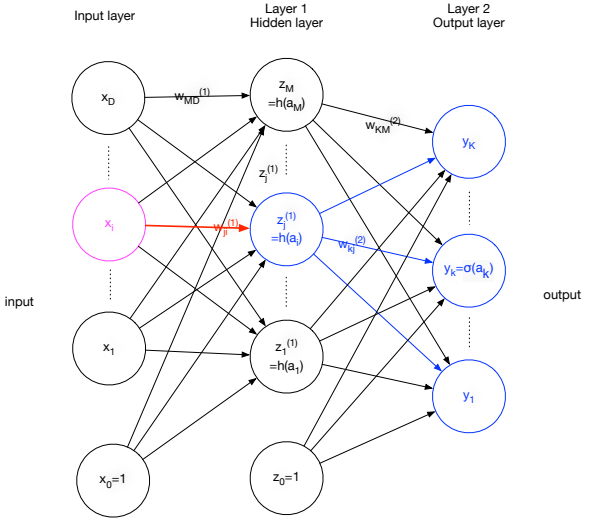
- Evaluate δ_k for all the output units using $\delta_k = y_k - t_k$
- Backpropagate δ's using

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

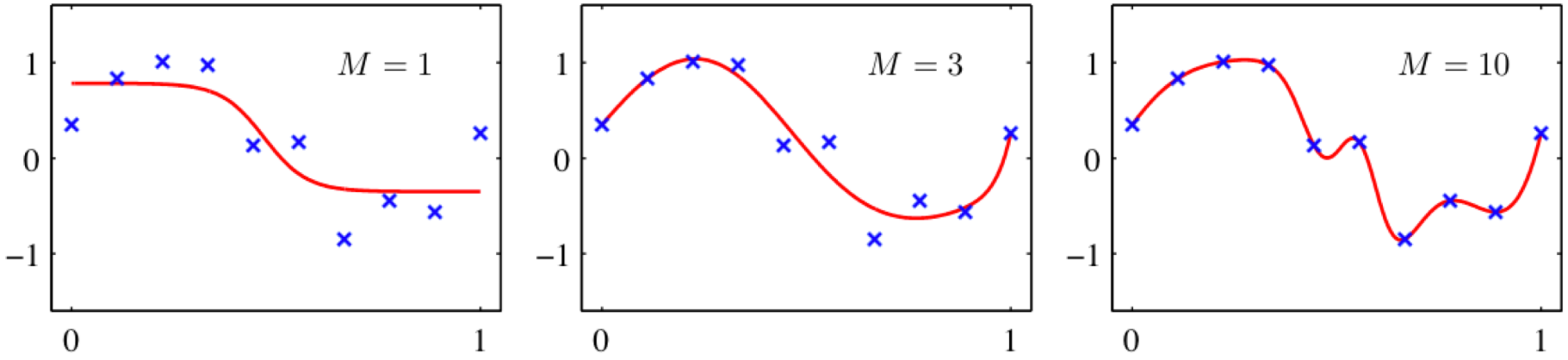
  to obtain δ_j for each hidden unit in the network

- Evaluate derivatives

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$



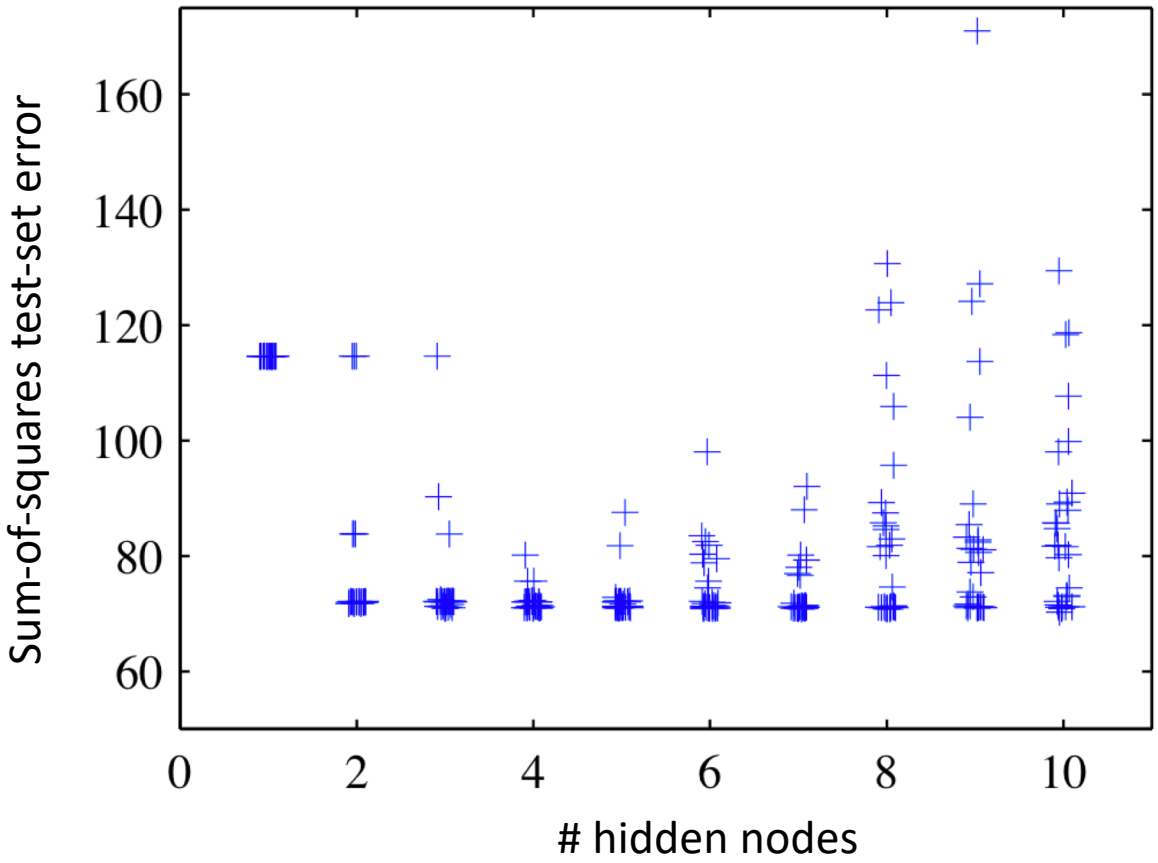
Regularization in Neural Nets

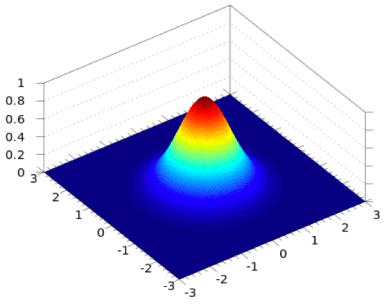


PRML

Local Minima

- Test error as a function of the number of hidden units. 30 random starts for each network size.





PRML

Regularization

- The simplest regularizer is a Gaussian prior, with a new objective function

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

What is wrong with back-propagation?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The brain needs to fit about 10^14 connection weights in only about 10^9 seconds.
  - Unless the weights are highly redundant, labels cannot possibly provide enough information.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- The neurons need to send two different types of signal
  - Forward pass: signal = activity = y
  - Backward pass: signal = dE/dy

Overcoming the limitations of back-propagation

- We need to keep the efficiency of using a gradient method for adjusting the weights, but use it for modeling the structure of the sensory input.
  - Adjust the weights to maximize the probability that a generative model would have produced the sensory input. This is the only place to get 10^5 bits per second.
  - Learn p(image) not p(label | image)

- What kind of generative model could the brain be using?

Thank you!