Lecture 13: Learning 3

http://cs.nju.edu.cn/yuy/course_ai18.ashx

Previously...

Learning

- Decision tree learning
- Nearest Neighbors
- Naive Bayes

- Why we can learn

Linear model

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$$

$$\boldsymbol{w} = \ w_1, w_2, \ldots, w_n \quad b$$

$$\Downarrow$$

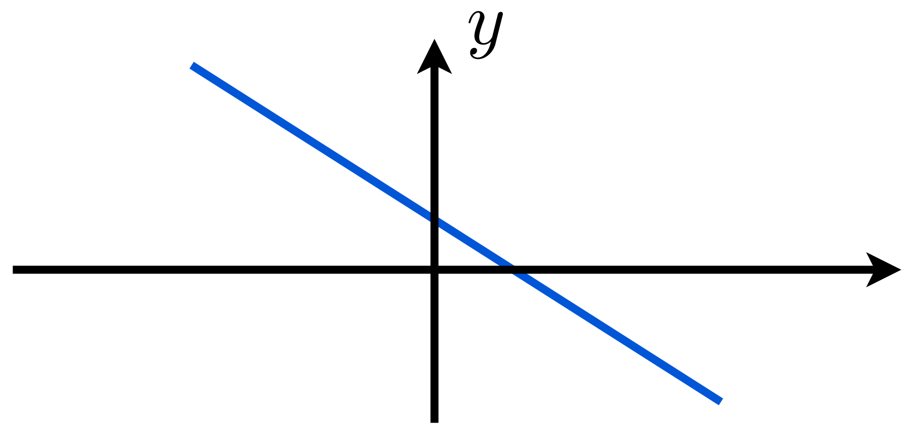
$$w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n + b$$

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$$

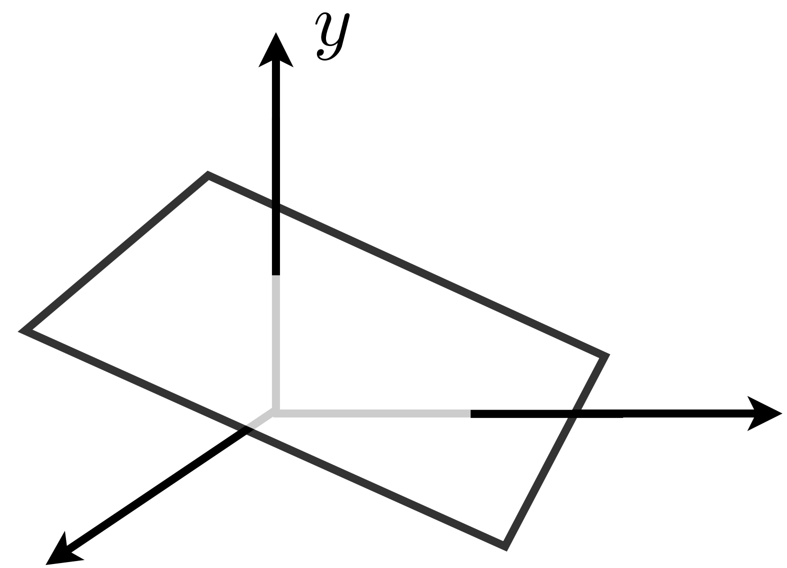
Vladimir Vapnik

Linear model

$$y = ax + b$$



$$y = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$



is the following a linear model?

$$y = w_1 \cdot x + w_2 \cdot x^2 + b$$

yes, the parameters are linear

Least square regression

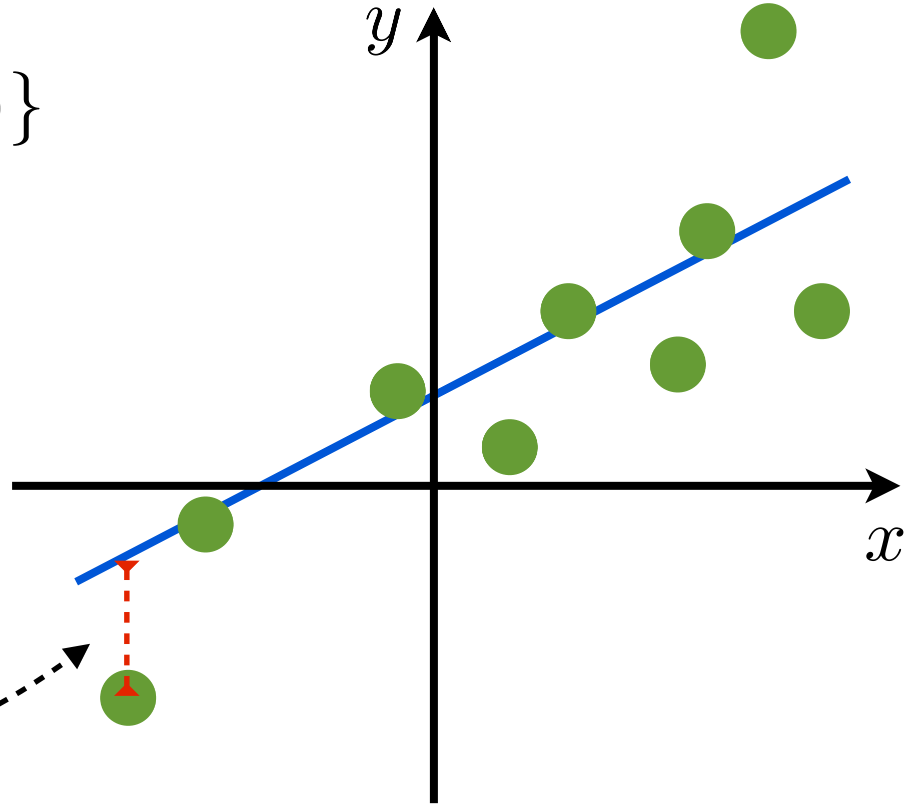
Regression: $y \in \mathbb{R}$

Training data:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_m, y_m)\}$$

Least square loss:

$$\frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2$$



Least square regression

$$L(\boldsymbol{w}, b) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2$$

$$\frac{\partial L(\boldsymbol{w}, b)}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} 2(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i) = 0$$

$$\frac{\partial L(\boldsymbol{w}, b)}{\partial \boldsymbol{w}} = \frac{1}{m}\sum_{i=1}^{m} 2(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)\boldsymbol{x}_i^\top = 0$$

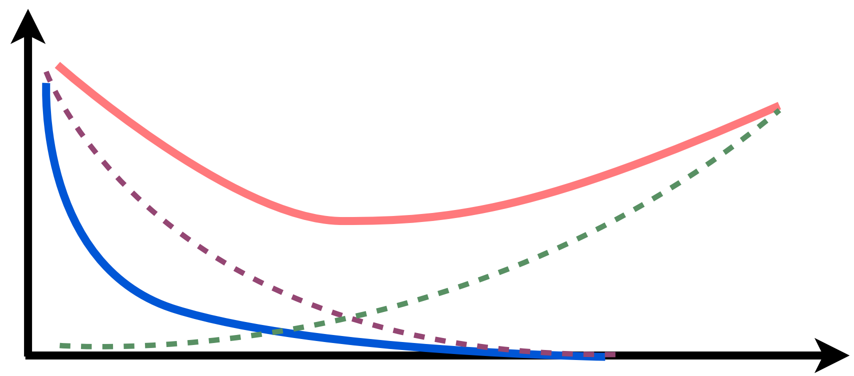
$$b = \frac{1}{m}\sum_{i=1}^{m}(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i) = \bar{y} - \boldsymbol{w}^\top \bar{\boldsymbol{x}}$$

$$\boldsymbol{w} = \left(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^\top - \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top\right)^{-1}\left(\frac{1}{m}\sum_{i=1}^{m}(y_i\boldsymbol{x}_i) - \bar{y}\bar{\boldsymbol{x}}\right)$$

$$= var(\boldsymbol{x})^{-1}cov(\boldsymbol{x}, y) = (X^\top X)^{-1}X^\top Y$$

closed form solution

Complexity of linear models



$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$$

possibility of $\boldsymbol{w}$

Regularization

make hypothesis space small
$\rightarrow$ better generalization ability
make numerical analysis stable

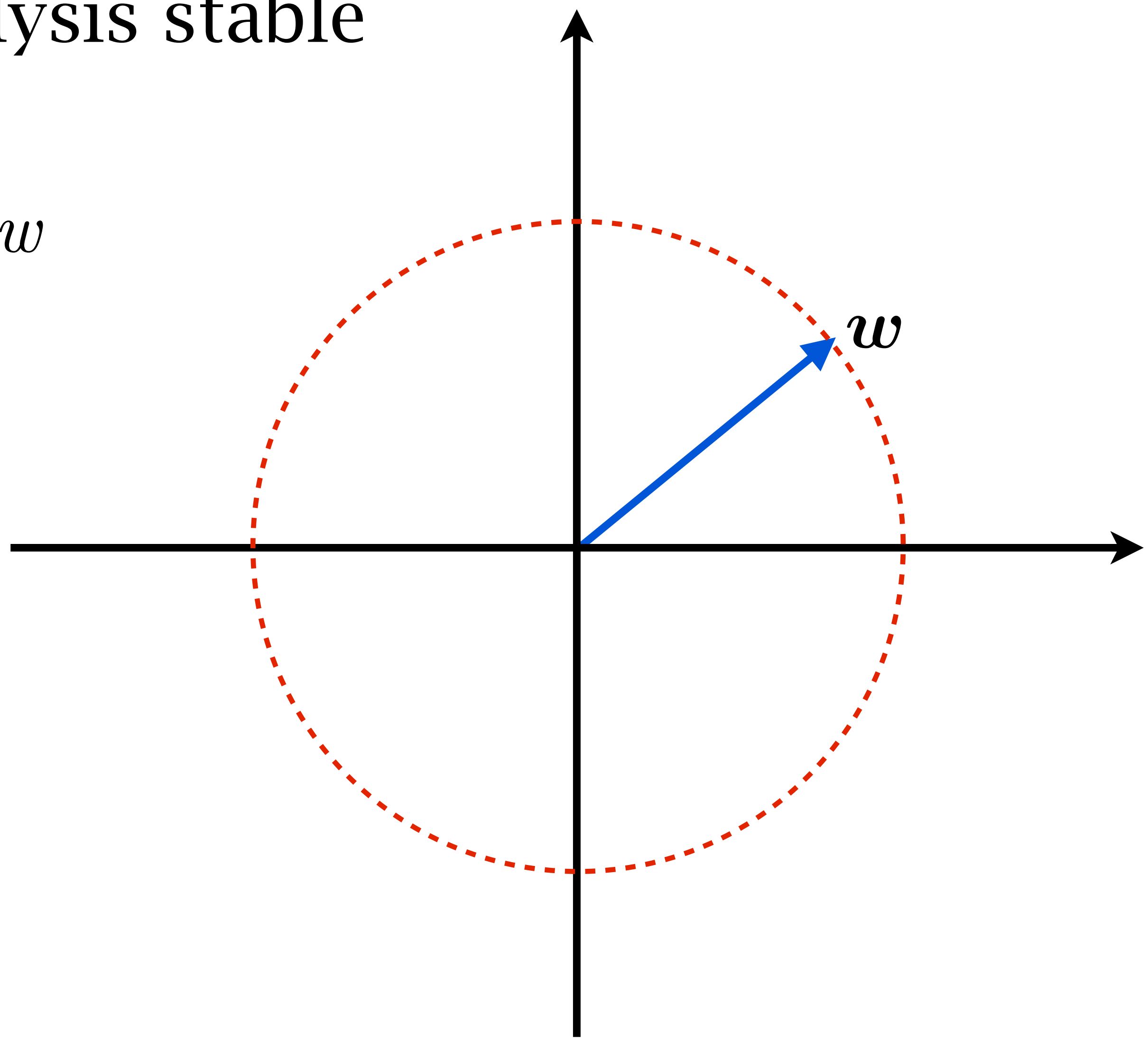
restrict the norm of w

$$\|\boldsymbol{w}\|_p = \left(\sum_{i=1}^{n} |w_i|^p\right)^{1/p}$$

$$\|\boldsymbol{w}\|_2 = \sqrt{\sum_{i=1}^{n} w_i^2}$$

$$\|\boldsymbol{w}\|_1 = \sum_{i=1}^{n} |w_i|$$

$$\|\boldsymbol{w}\|_\infty = \max_{i=1,\ldots,n} |w_i|$$



Ridge regression

Regression: $y \in \mathbb{R}$

Training data:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_m, y_m)\}$$

objective:

$$\arg\min_{\boldsymbol{w}, b} \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2$$

$$s.t. \qquad \|\boldsymbol{w}\|_2 \leq \theta$$

or:

$$\arg\min_{\boldsymbol{w}, b} \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 + \lambda \|\boldsymbol{w}\|_2$$

Ridge regression

centered data, no bias:

$$\arg\min_{\boldsymbol{w}} \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2 + \lambda\|\boldsymbol{w}\|_2$$

closed form solution:

0

$$\boldsymbol{w} = \Big(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^\top - \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top + \lambda\boldsymbol{I}\Big)^{-1}\Big(\frac{1}{m}\sum_{i=1}^{m}(y_i\boldsymbol{x}_i) - \bar{y}\bar{\boldsymbol{x}}\Big)$$

$$= (var(\boldsymbol{x}) + \lambda\boldsymbol{I})^{-1}cov(\boldsymbol{x}, y)$$

$$= (X^\top X + \lambda I)^{-1}X^\top Y$$

$\boldsymbol{I}$ is the identity matrix

Least square v.s. ridge regression

$$\boldsymbol{w} = \left(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^\top - \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top\right)^{-1}\left(\frac{1}{m}\sum_{i=1}^{m}(y_i\boldsymbol{x}_i) - \bar{y}\bar{\boldsymbol{x}}\right)$$
$$= var(\boldsymbol{x})^{-1}cov(\boldsymbol{x}, y) = (X^\top X)^{-1}X^\top Y$$

$$\boldsymbol{w} = \left(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i\boldsymbol{x}_i^\top - \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^\top + \lambda\boldsymbol{I}\right)^{-1}\left(\frac{1}{m}\sum_{i=1}^{m}(y_i\boldsymbol{x}_i) - \bar{y}\bar{\boldsymbol{x}}\right)$$
$$= (var(\boldsymbol{x}) + \lambda\boldsymbol{I})^{-1}cov(\boldsymbol{x}, y)$$
$$= (X^\top X + \lambda I)^{-1}X^\top Y$$

stable solution

Least absolute shrinkage and selection operator (LASSO)

Regression: $y \in \mathbb{R}$

Training data:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), (\boldsymbol{x}_m, y_m)\}$$

objective:

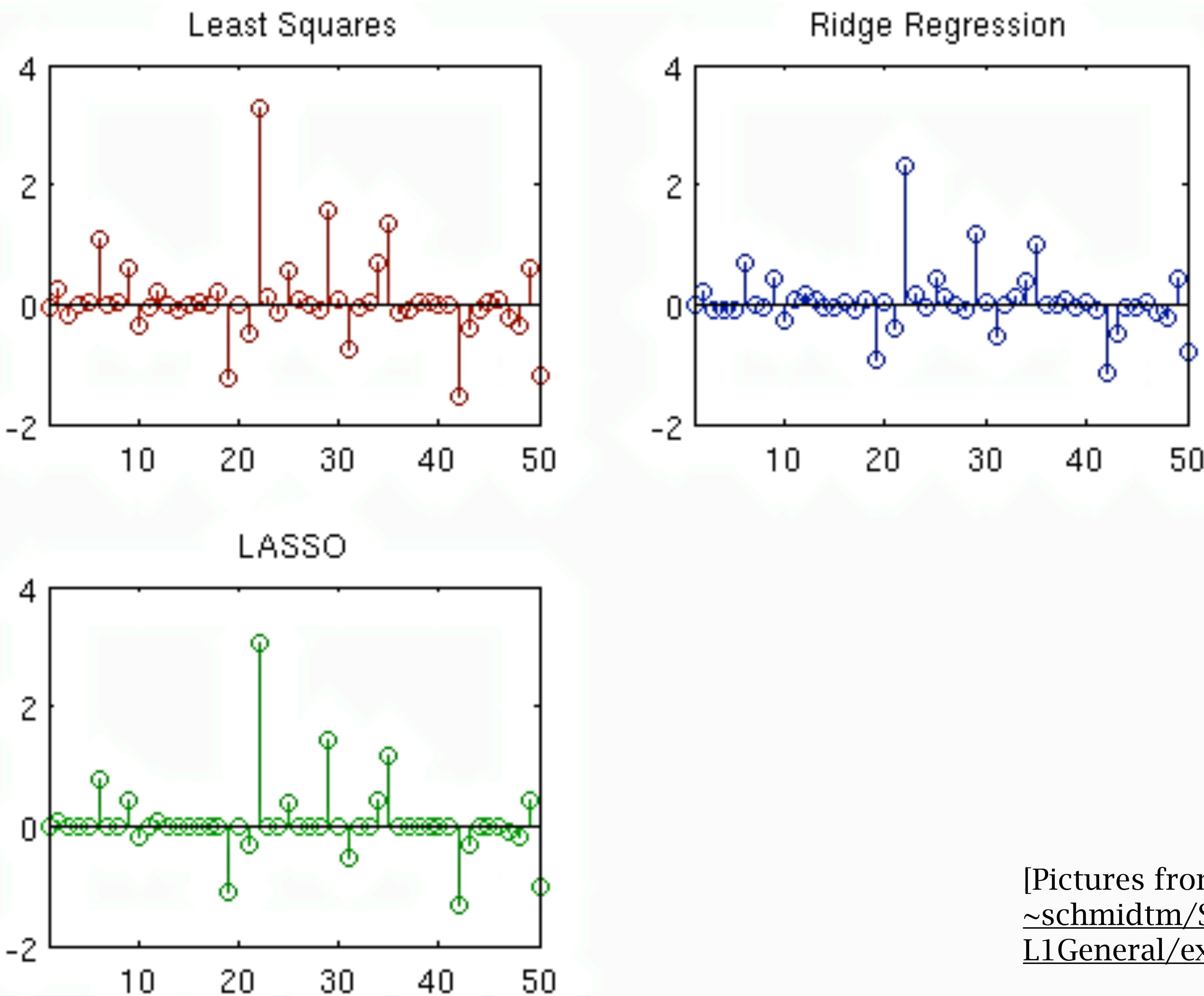
$$\arg\min_{\boldsymbol{w},b} \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2$$

$$s.t. \qquad \|\boldsymbol{w}\|_1 \leq \theta$$

or:

$$\arg\min_{\boldsymbol{w},b} \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 + \lambda \|\boldsymbol{w}\|_1$$

Comparing different regressions



[Pictures from www.cs.ubc.ca/~schmidtm/Software/L1General/examples.html]

A general framework

objective function:

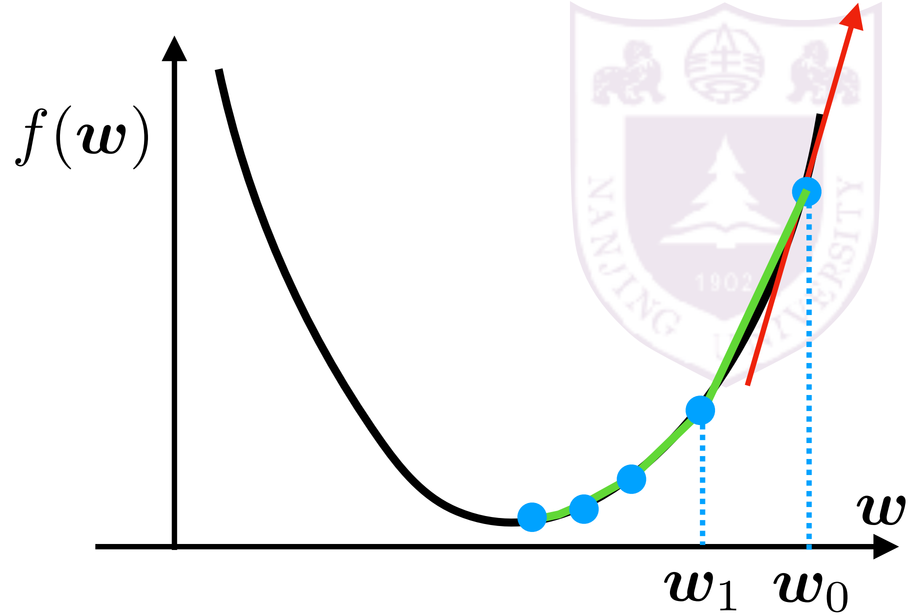
$$\arg\min_{\boldsymbol{w},b} L(\boldsymbol{w}, b) + \|\boldsymbol{w}\|_p$$

how to solve the parameters?

a generally applied technique: gradient-descent

Gradient descent

(steepest descent)

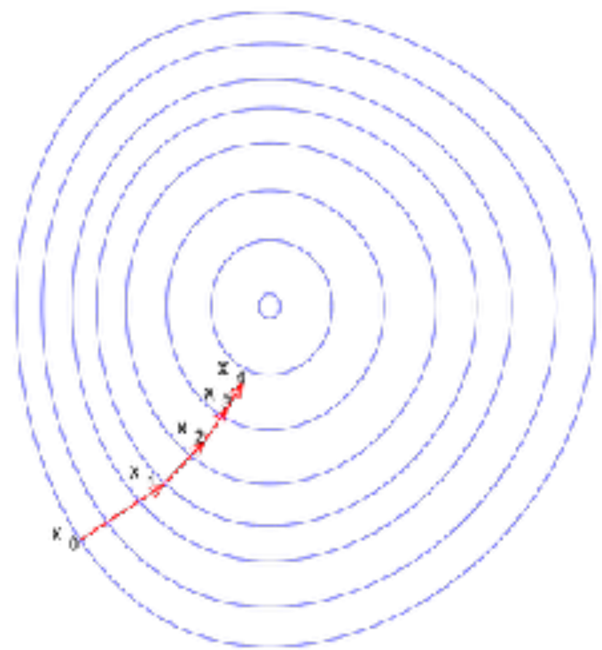


for a differentiable function f

$$\arg\min_{\boldsymbol{w}} f(\boldsymbol{w})$$

can be solved by

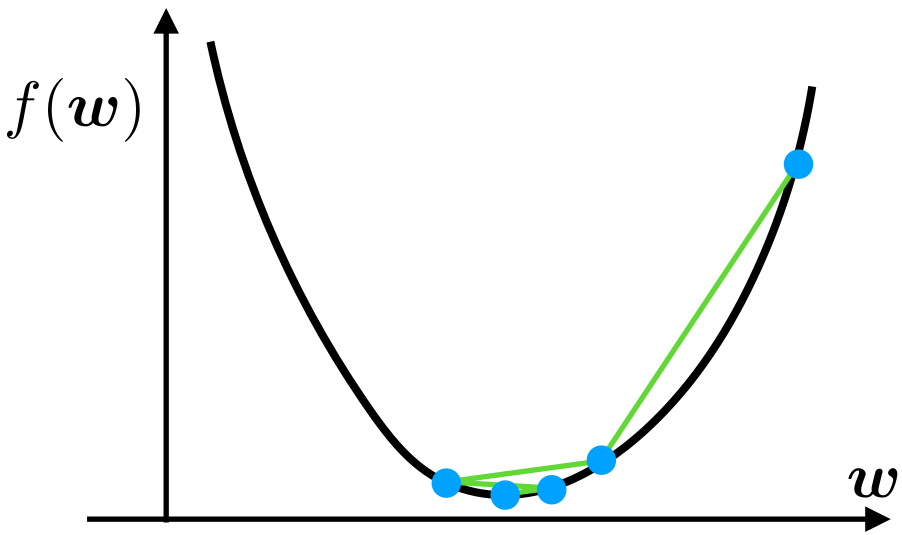
1. start from an arbitrary initial point $\boldsymbol{w}_0$
2. loop from t=0
3. $$\boldsymbol{w}_{t+1} = \boldsymbol{w} - \eta \frac{\partial f(\boldsymbol{w})}{\partial \boldsymbol{w}}$$
   or $\boldsymbol{w}_{t+1} = \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{w})$
4. until convergence $\|\nabla_{\boldsymbol{w}} f(\boldsymbol{w})\| < \epsilon$

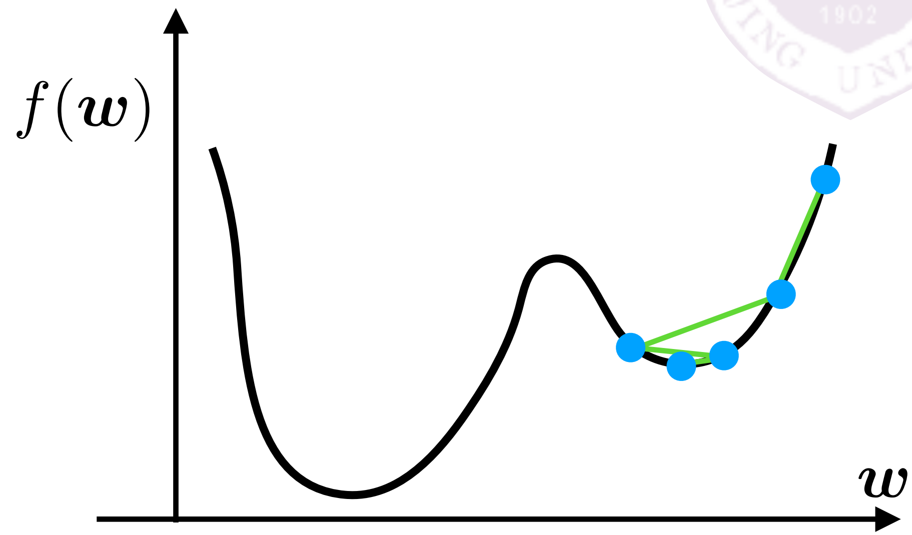


[image from wikipedia]

Gradient descent

$$\boldsymbol{w}_{t+1} = \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} f(\boldsymbol{w})$$

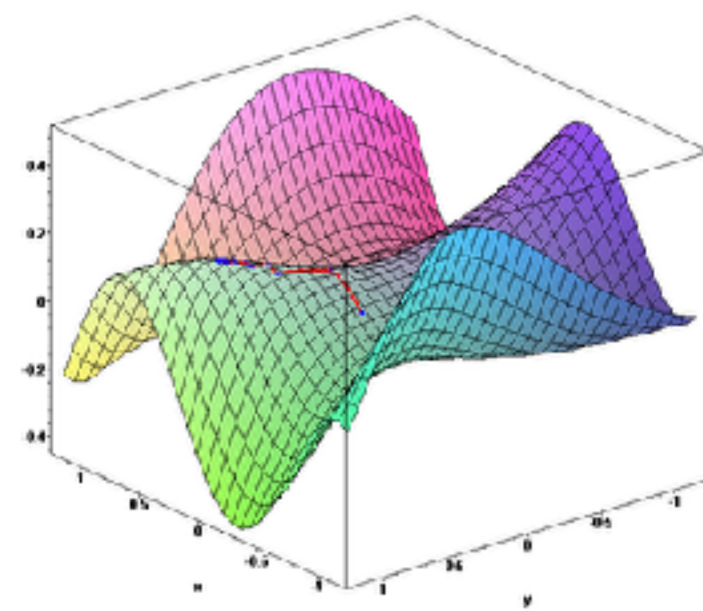




for convex functions:
converge to global optima

$$f(\alpha \boldsymbol{w}_1 + (1-\alpha)\boldsymbol{w}_2)) \geq \alpha f(\boldsymbol{w}_1) + (1-\alpha) f(\boldsymbol{w}_2)$$

for other functions:
converge to stationary points



[image from wikipedia]

A general framework

objective function:

$$\arg\min_{\boldsymbol{w},b} L(\boldsymbol{w}, b) + \|\boldsymbol{w}\|_p$$

how to solve the parameters?

general optimization: gradient descent

$$(\boldsymbol{w}, b) - = \eta \frac{\partial (L(\boldsymbol{w}, b) + \|\boldsymbol{w}\|_p)}{\partial (\boldsymbol{w}, b)}$$

Linear classifier

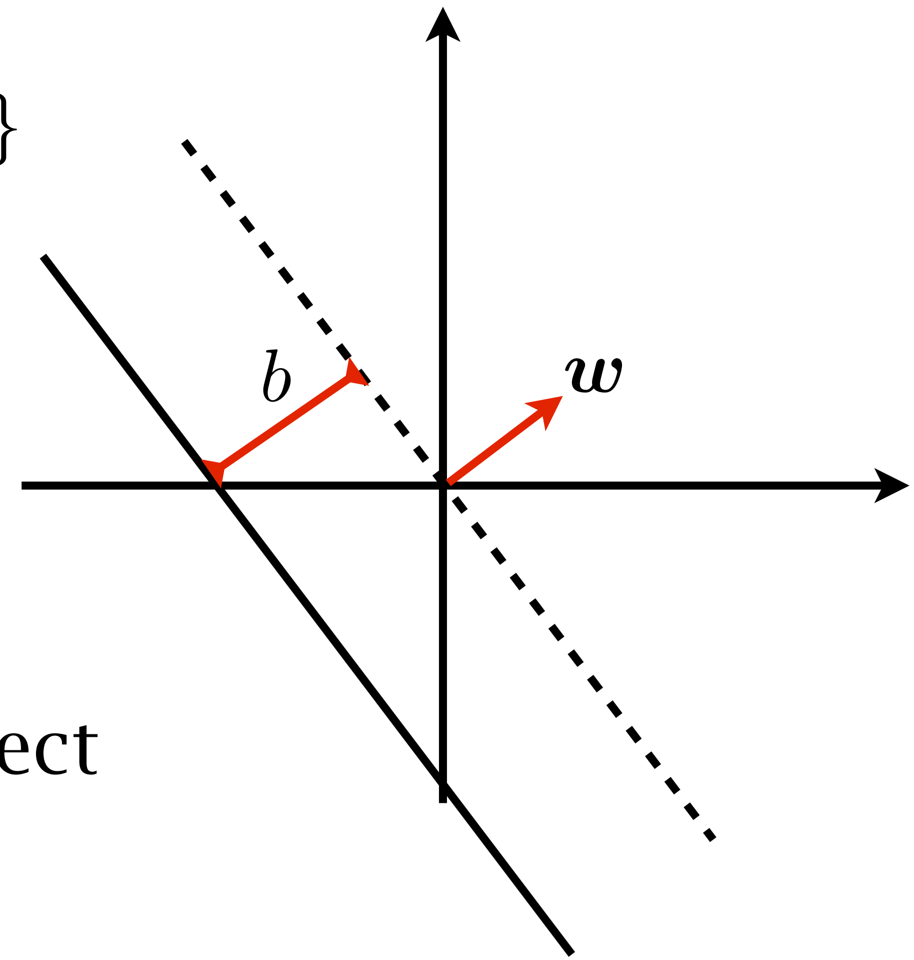
model space: $\mathbb{R}^{n+1}$

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$$

for classification $y \in \{-1, +1\}$

we predict an instance by

$$\text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b) = \begin{cases} +1, & \boldsymbol{w}^\top \boldsymbol{x} + b > 0 \\ -1, & \boldsymbol{w}^\top \boldsymbol{x} + b < 0 \\ \text{random}, & otherwise \end{cases}$$



for an example $(\boldsymbol{x}, y)$, a correct prediction means

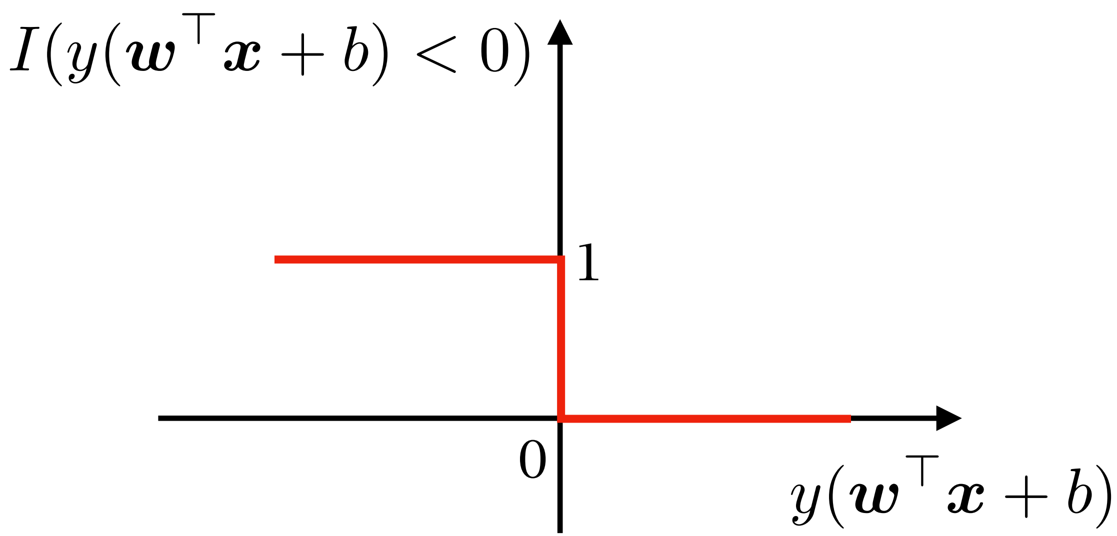
$$y(\boldsymbol{w}^\top \boldsymbol{x} + b) > 0$$

Ideal classifier

$$\arg\min_{\boldsymbol{w},b} \sum_i I(y(\boldsymbol{w}^\top \boldsymbol{x} + b) \leq 0)$$

non-differentiable

hard to solve by gradient descent



Prototype

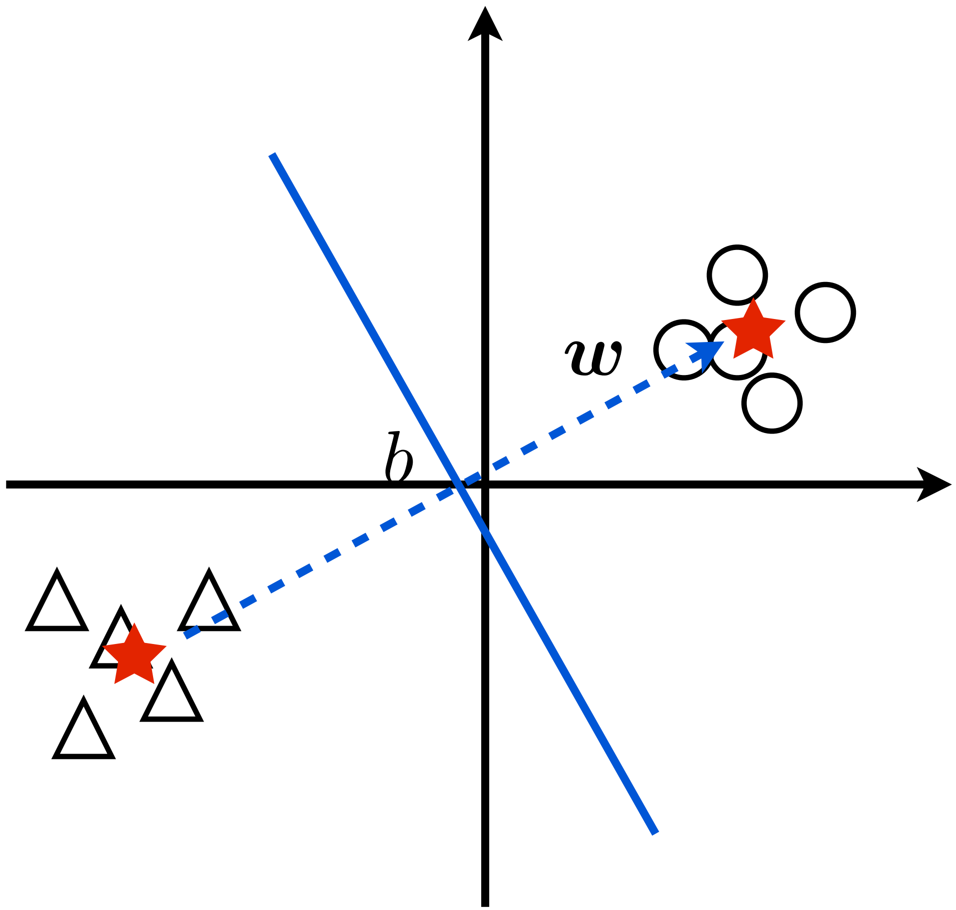
simple, but too restricted

$$\bar{\boldsymbol{x}}^+ = \frac{1}{\sum_{i:y_i=+1} 1} \sum_{i:y_i=+1} \boldsymbol{x}_i$$

$$\bar{\boldsymbol{x}}^- = \frac{1}{\sum_{i:y_i=-1} 1} \sum_{i:y_i=-1} \boldsymbol{x}_i$$

$$\boldsymbol{w} = \bar{\boldsymbol{x}}^+ - \bar{\boldsymbol{x}}^-$$

$$b = -\boldsymbol{w}^\top \cdot \frac{\bar{\boldsymbol{x}}^+ + \bar{\boldsymbol{x}}^-}{2}$$



Perceptron

perception loss

$$\arg\min_{\boldsymbol{w},b} \sum_i \max\{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b), 0\}$$

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$$

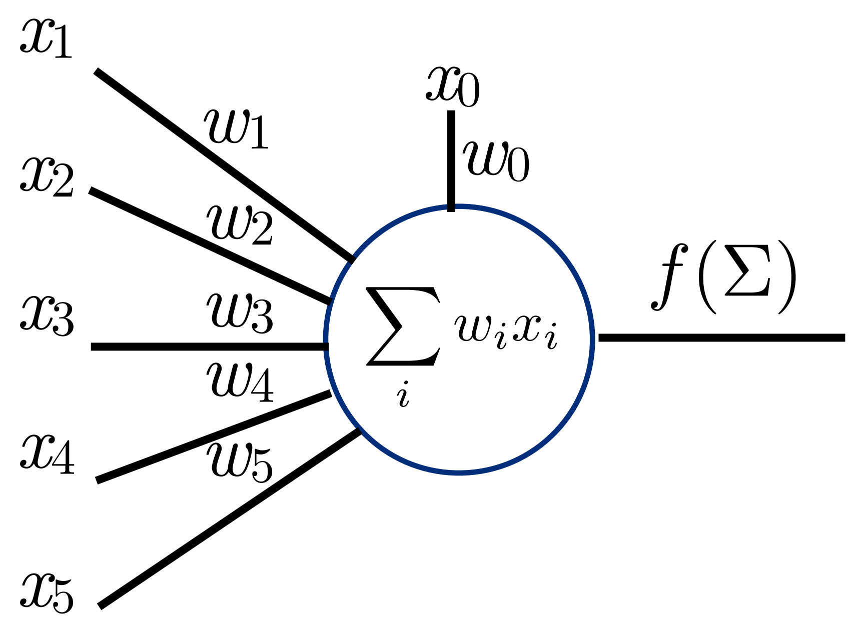
gradient ascent

$$\frac{\partial y \boldsymbol{w}^\top \boldsymbol{x}}{\partial \boldsymbol{w}} = y\boldsymbol{x}$$

feed training examples one by one

1. $\boldsymbol{w} = 0$
2. for each example $(\boldsymbol{x}, y)$
   if $\text{sign}(y\boldsymbol{w}^\top \boldsymbol{x}) < 0$

$$\boldsymbol{w} = \boldsymbol{w} + y\boldsymbol{x}$$

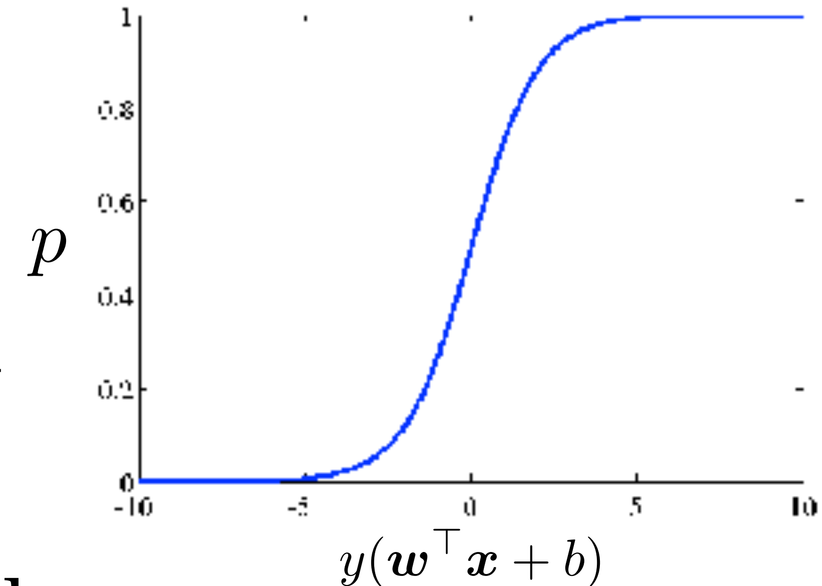


Logistic regression

assume logit model: for a positive example

$$\boldsymbol{w}^\top \boldsymbol{x} = \log \frac{p(+1 \mid \boldsymbol{x})}{1 - p(+1 \mid \boldsymbol{x})}$$

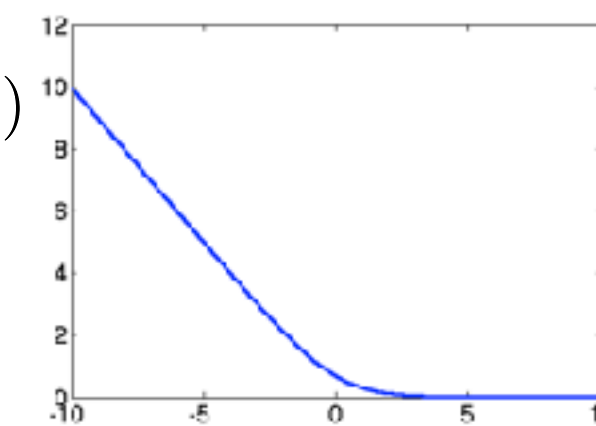
so that $p(y \mid \boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + e^{-y(\boldsymbol{w}^\top \boldsymbol{x})}}$



minimize negative log-likelihood:

$$\arg\min_{\boldsymbol{w},b} -\log \prod_{i=1}^{m} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) = -\sum_i \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w})$$

$$= \sum_i \log \left(1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i)}\right)$$



convex

Linear classifier revisit

model space: $\mathbb{R}^{n+1}$

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$$

for classification $y \in \{-1, +1\}$

Original objective:

$$\arg\min_{\boldsymbol{w}, b} \sum_i I(y(\boldsymbol{w}^\top \boldsymbol{x} + b) \leq 0)$$

0-1 loss
hard to optimize

Surrogate objective:

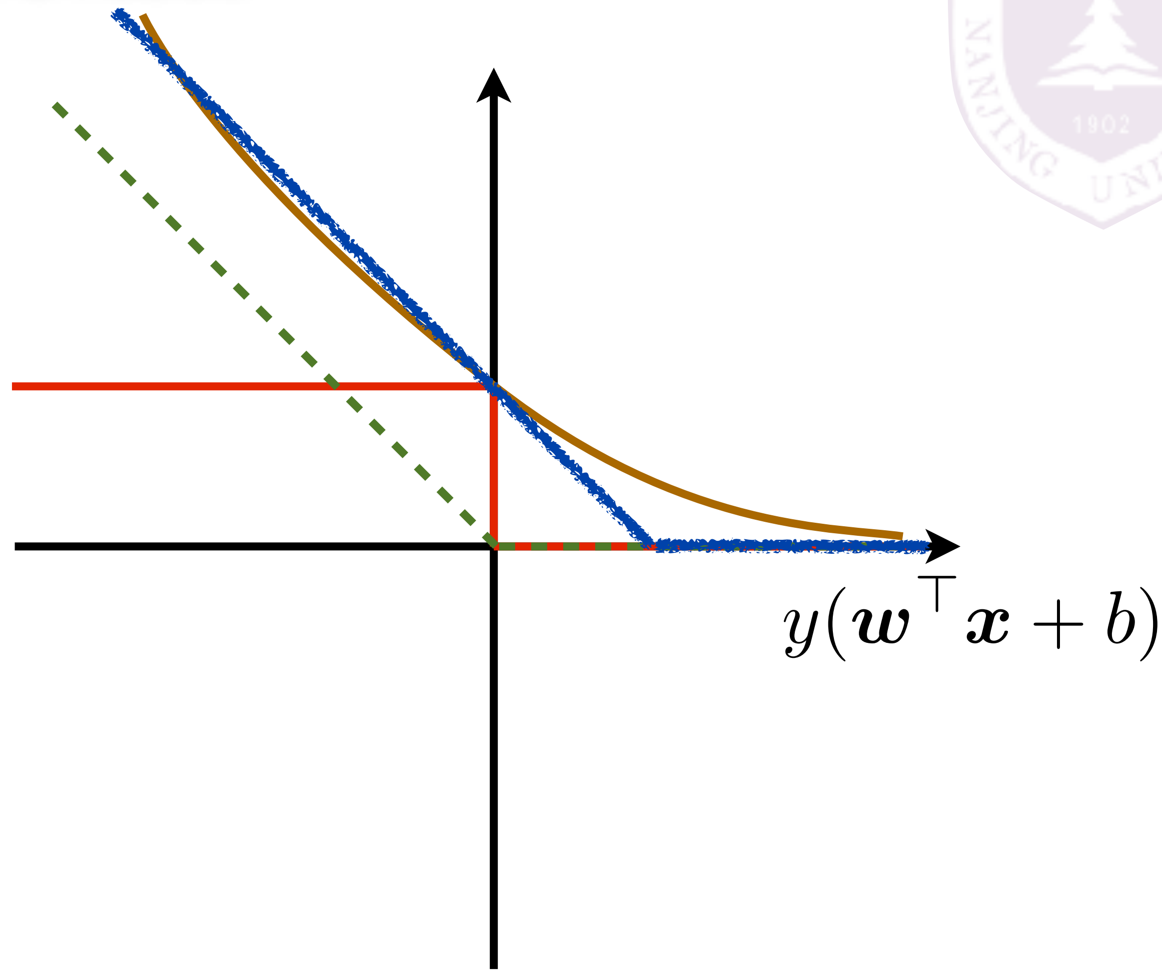
$$\arg\min_{\boldsymbol{w}, b} \sum_i \log\left(1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}\right)$$

logistic regression

$$\arg\min_{\boldsymbol{w}, b} \sum_i \max\{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b), 0\}$$

perceptron

Linear classifier revisit



0-1 loss

$I(y(\boldsymbol{w}^\top \boldsymbol{x} + b) \leq 0)$

logistic regression

$\log_2(1 + e^{-y(\boldsymbol{w}^\top \boldsymbol{x} + b)})$

perceptron

$\max\{-y(\boldsymbol{w}^\top \boldsymbol{x} + b), 0\}$

hinge loss

$\max\{1 - y(\boldsymbol{w}^\top \boldsymbol{x} + b), 0\}$

Support vector machines (SVM)

hinge loss + L2-norm

$$\arg\min_{\boldsymbol{w},b} \sum_i \max(1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b), 0) + \lambda \|\boldsymbol{w}\|_2$$

$\max(1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b), 0) = \xi_i$

$\xi_i \geq 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)$

$\xi_i \geq 0$

$$\arg\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2 + C\sum_i \xi_i$$

$$s.t. \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i$$

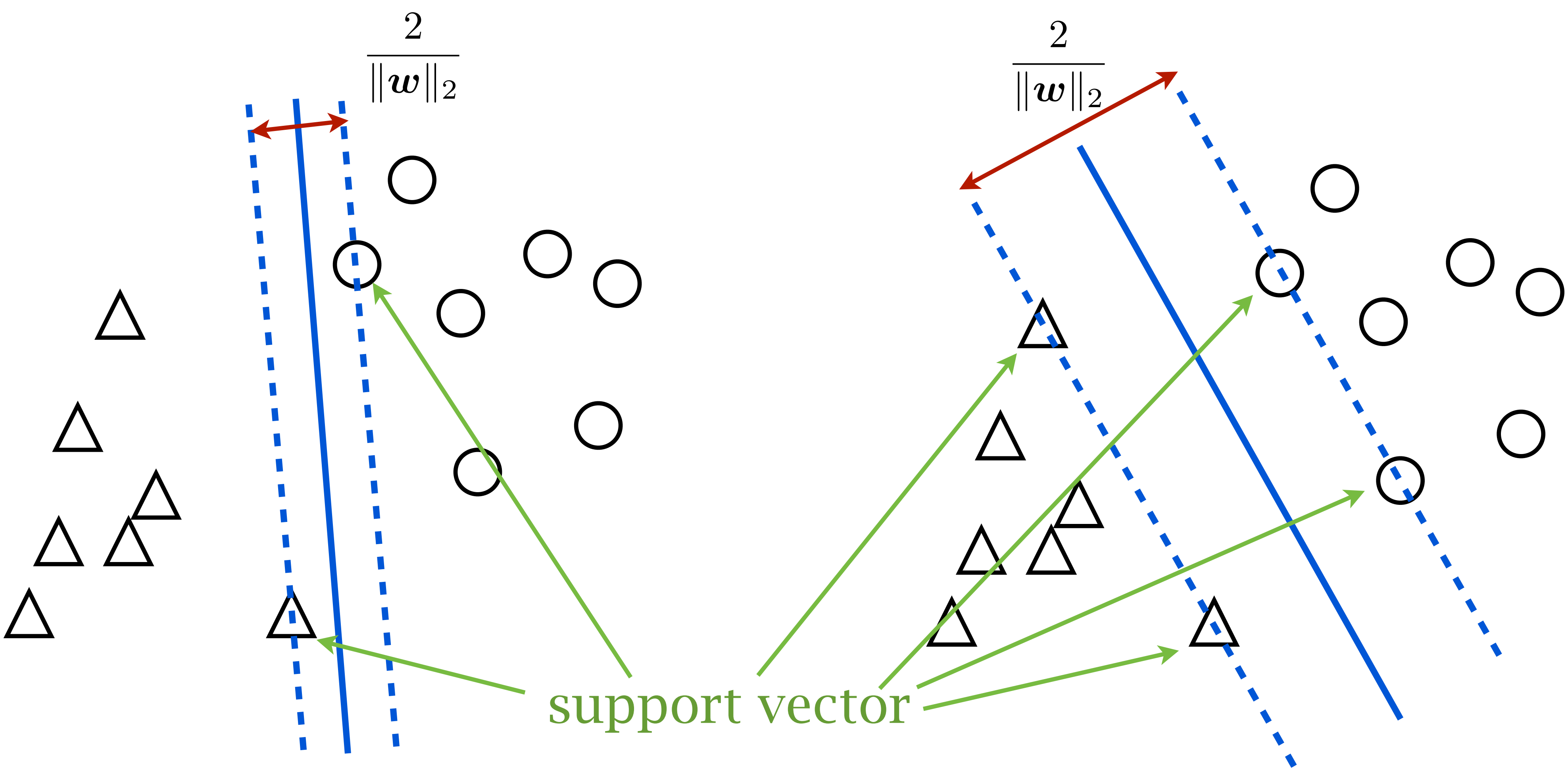
$$\xi_i \geq 0$$

quadratic

Support vector machines (SVM)

$$\arg\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2$$

$$s.t. \qquad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1$$



Scoring functions

$$\frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2$$ least square regression

$$\frac{1}{m}\sum_{i=1}^{m}|\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i|$$ LAD regression

$$\frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 + \lambda\|\boldsymbol{w}\|_2$$ ridge regression

$$\frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 + \lambda\|\boldsymbol{w}\|_1$$ LASSO

Scoring functions

$$\sum_i I(y(\boldsymbol{w}^\top \boldsymbol{x} + b) > 0)$$ 0-1 loss

$$\sum_i \max\{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b), 0\}$$ perceptron

$$\sum_i \log\left(1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}\right)$$ logistic regression

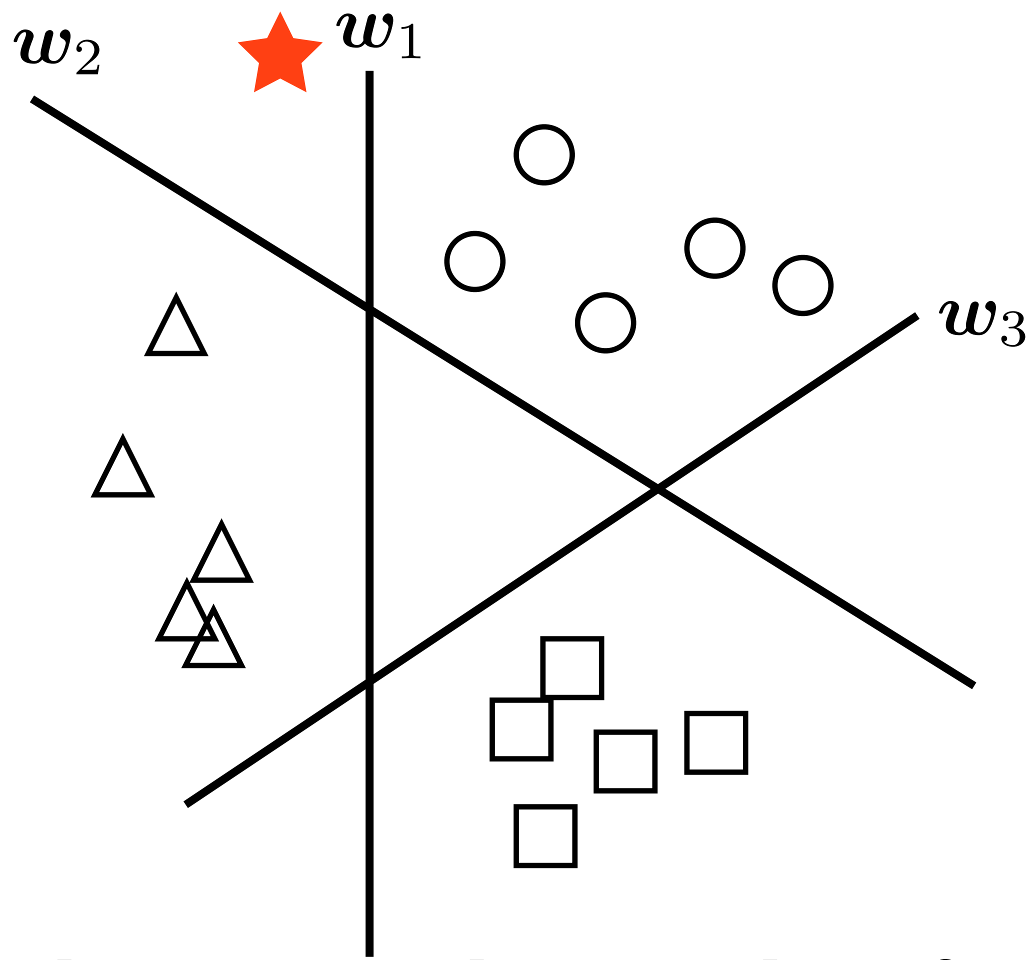
$$\sum_i \log\left(1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}\right) + \lambda \|\boldsymbol{w}\|_2$$ regularized LR

$$\sum_i \max(1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b), 0) + \lambda \|\boldsymbol{w}\|_2$$ SVM

minimize loss + regularization

Multi-class classification

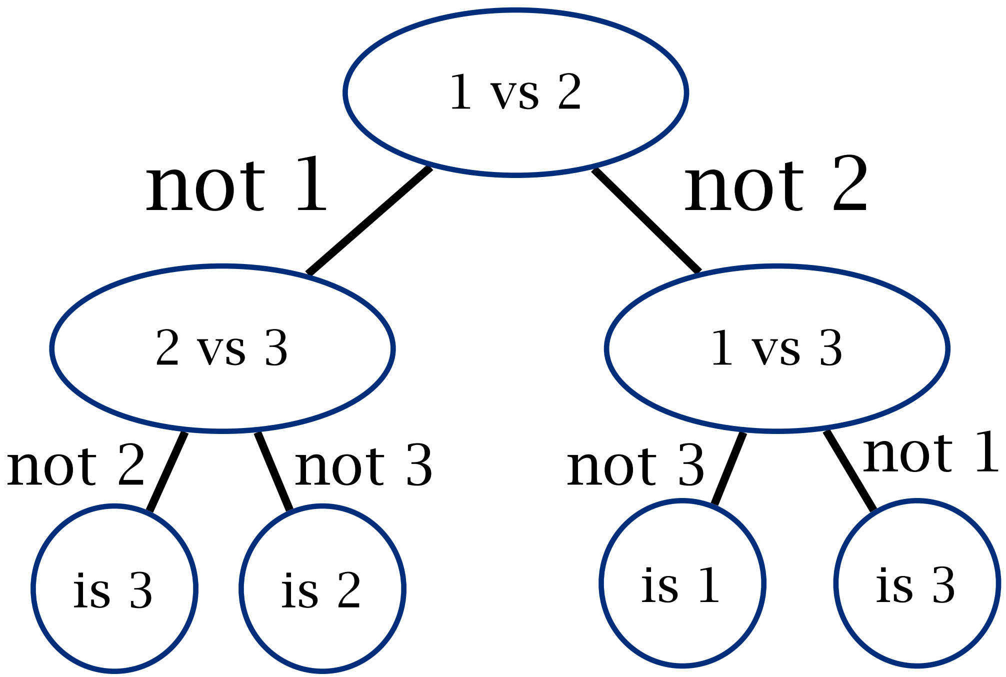
one-vs-rest

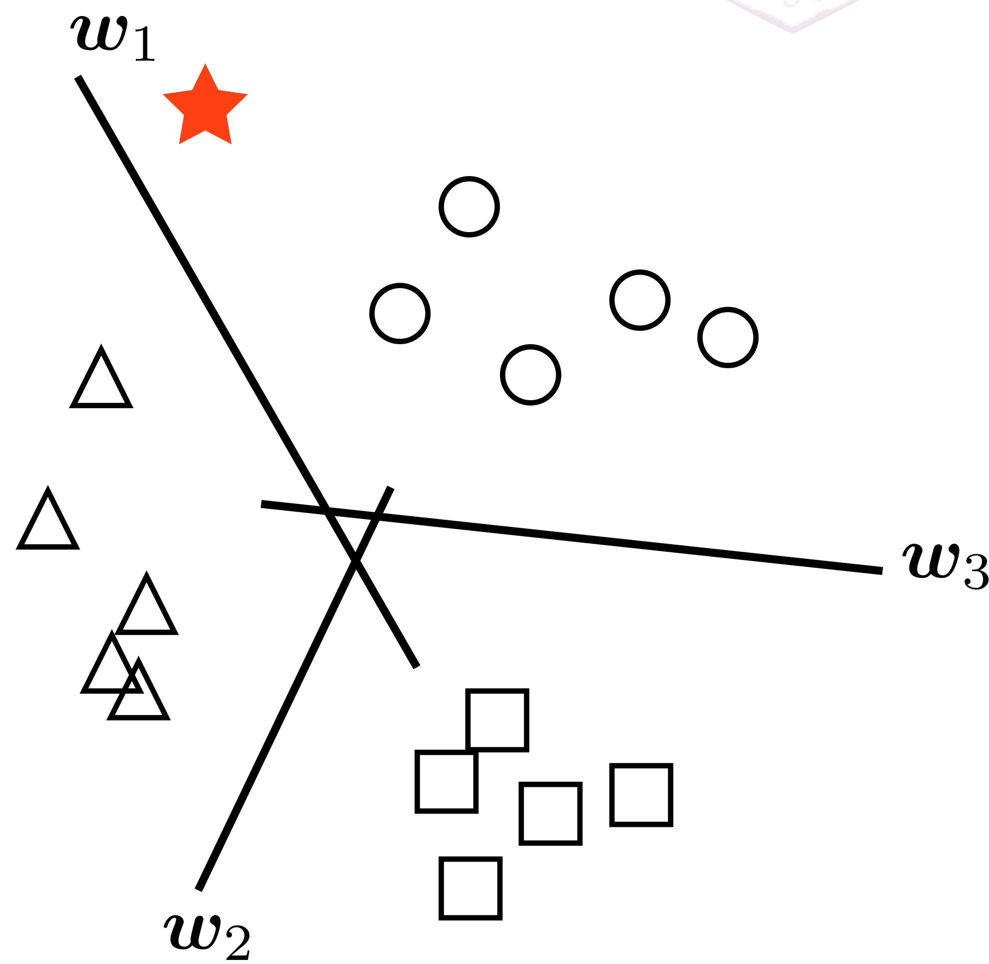


for C classes, need to train C binary classifiers

Multi-class classification

one-vs-one

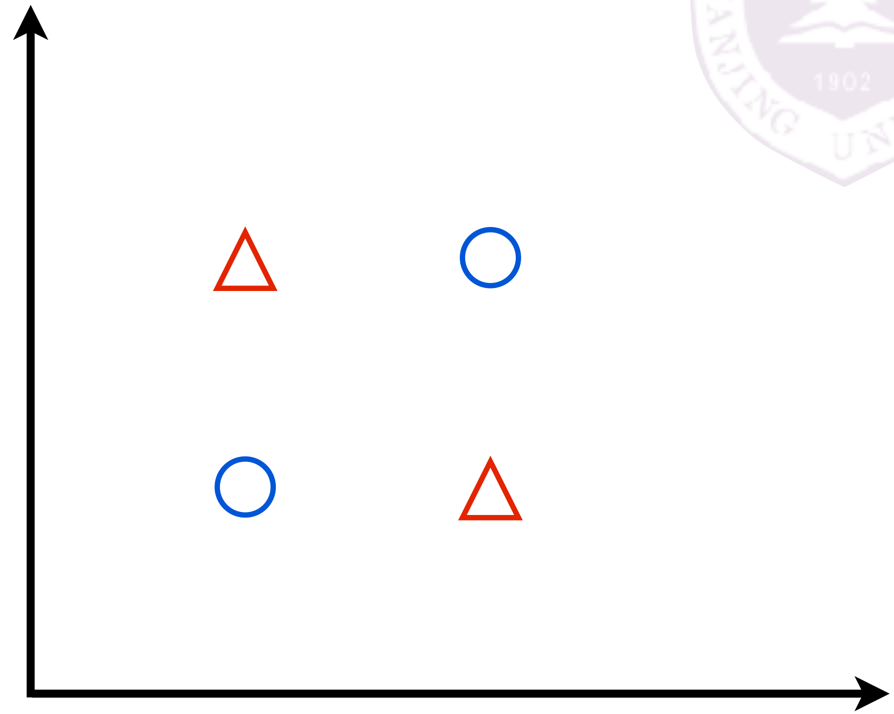




for C classes, need to train $C(C\text{-}1)/2$ binary classifiers

Limitation of linear classifier

may not complex enough



XOR in 2D

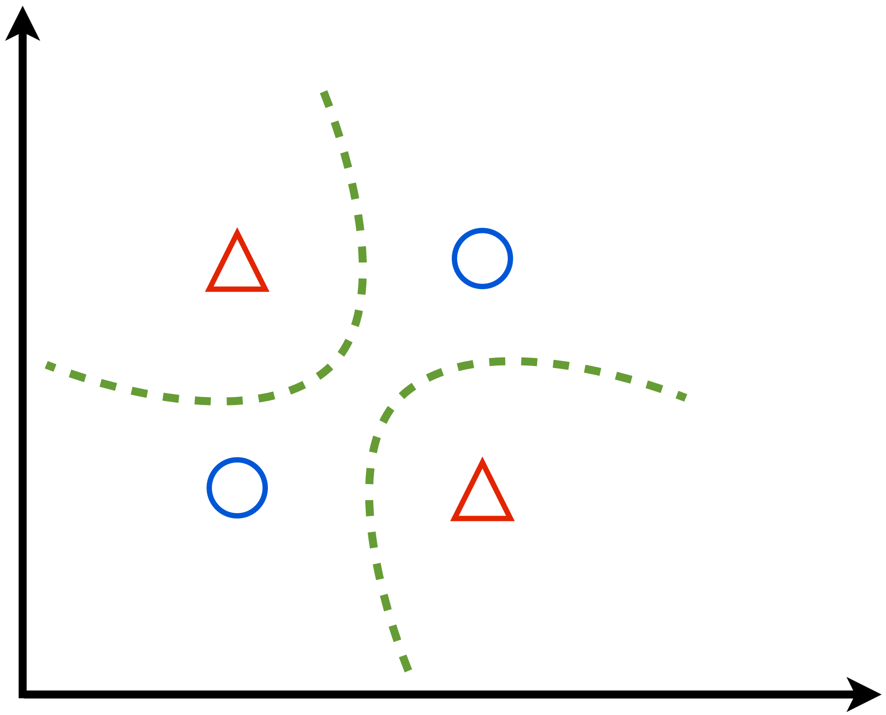
is the following a linear model?

$$y = w_1 \cdot x + w_2 \cdot x^2 + b$$

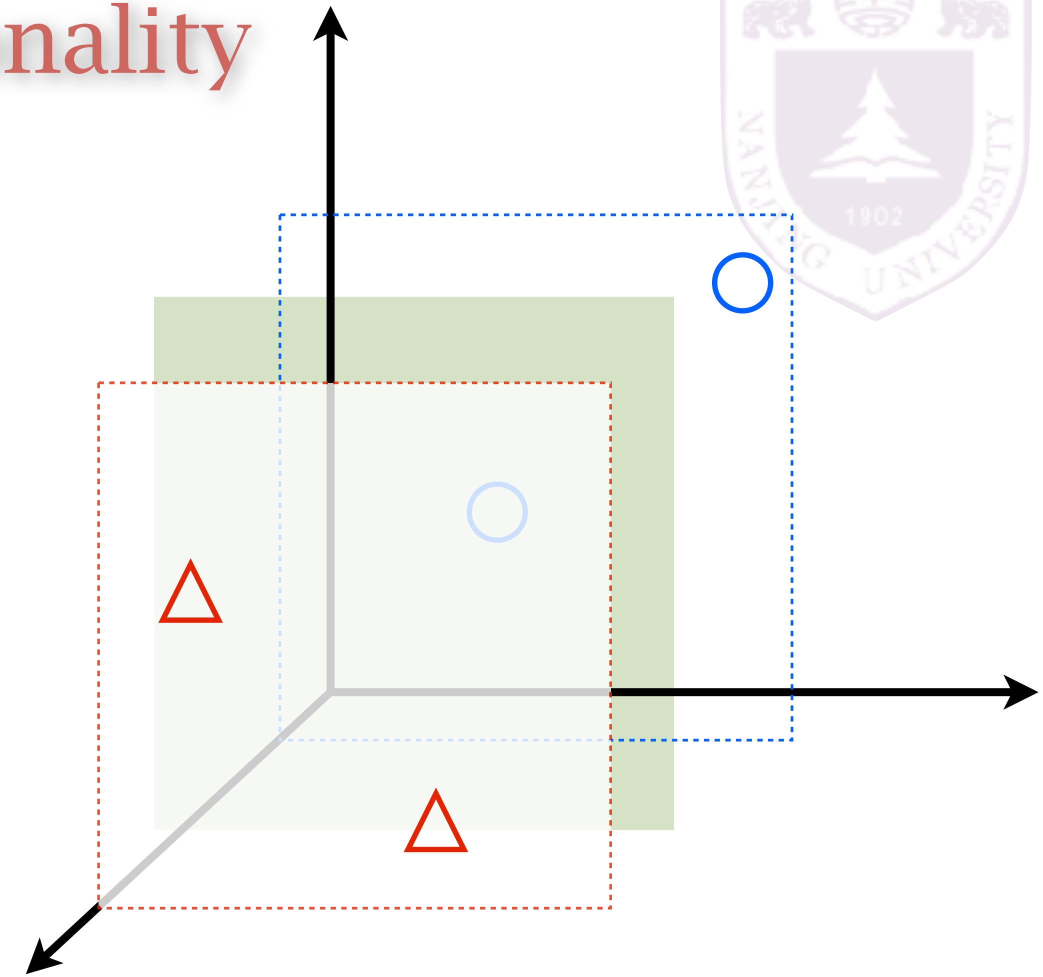
yes, the parameters are linear

better **basis?**

Linearity v.s. dimensionality



XOR in 2D



x_1	x_2	y
0	0	+1
0	1	-1
1	0	-1
1	1	+1

x_1	x_2	x_1x_2	y
0	0	0	+1
0	1	0	-1
1	0	0	-1
1	1	1	+1

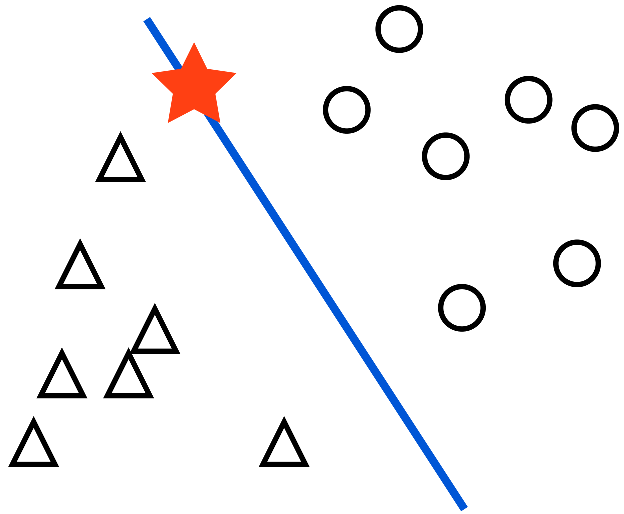
$$\boldsymbol{w} = \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix}, b = -0.5$$

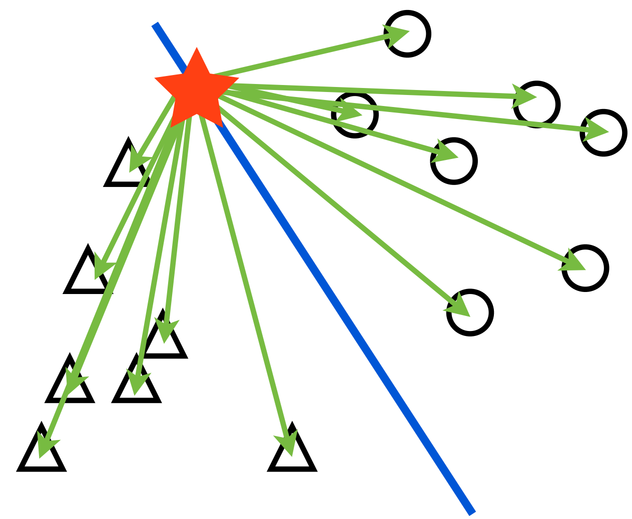
Nonlinearity from linearity

$$f(\boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{x}$$

↓ linear model in sample space

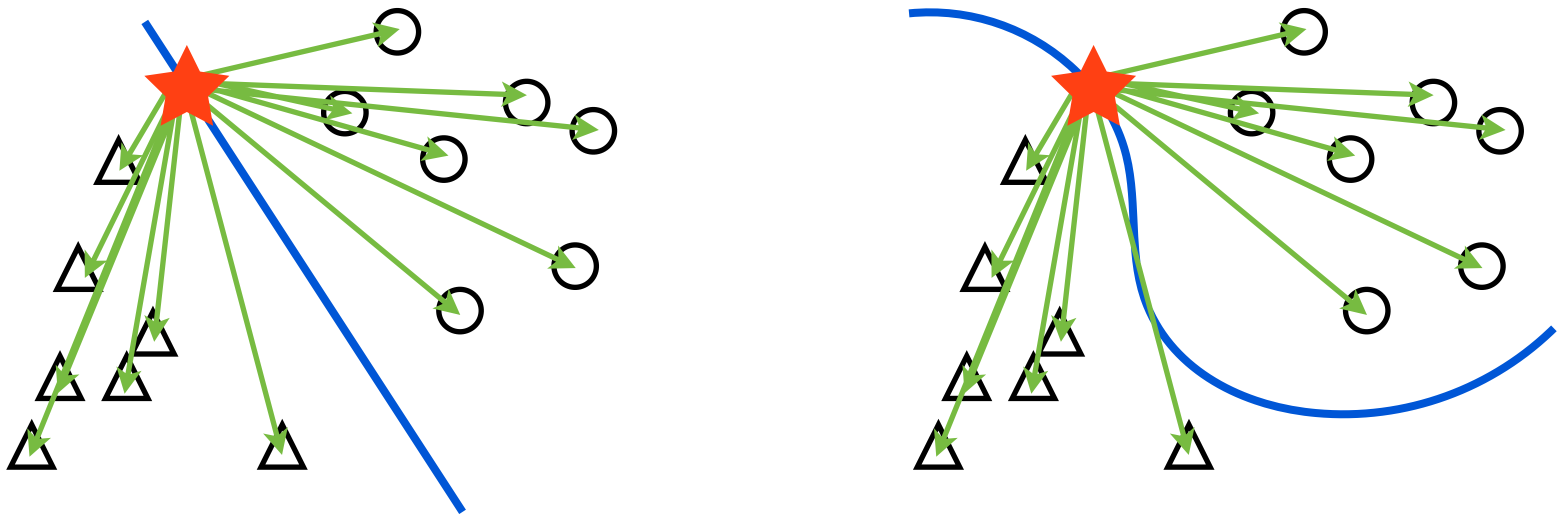
$$f(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i (\boldsymbol{x}_i^\top \boldsymbol{x}) = \boldsymbol{\alpha}^\top X \boldsymbol{x} \xrightarrow[\text{distance}]{\text{change}} \sum_{i=1}^{m} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$$





Nonlinearity from linearity

$$f(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i (\boldsymbol{x}_i^\top \boldsymbol{x}) = \boldsymbol{\alpha}^\top X \boldsymbol{x} \xrightarrow[\text{distance}]{\text{change}} \sum_{i=1}^{m} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x})$$



example:

$$K(x,y) = \left(\sum_{i=1}^{n} x_i y_i + c\right)^2 = \sum_{i=1}^{n} (x_i^2)(y_i^2) + \sum_{i=2}^{n}\sum_{j=1}^{i-1} (\sqrt{2}x_i x_j)(\sqrt{2}y_i y_j) + \sum_{i=1}^{n} (\sqrt{2c}x_i)(\sqrt{2c}y_i) + c^2$$

$$\varphi(x) = \langle x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_n x_1, \sqrt{2}x_{n-1}x_{n-2}, \ldots, \sqrt{2}x_{n-1}x_1, \ldots, \sqrt{2}x_2 x_1, \sqrt{2c}x_n, \ldots, \sqrt{2c}x_1, c\rangle$$

Nonlinearity from composition

linear transformation

$W_1 \boldsymbol{x}$

nonlinear transformation function

$f_1(W_1 \boldsymbol{x})$ new basis

composition

1-layer

$$f(\boldsymbol{W}) = \boldsymbol{w} f_1(W_1 \boldsymbol{x})$$

k-layer

$$f(\boldsymbol{W}) = f_k(W_k \cdots f_2(W_1 f_1(W_1 \boldsymbol{x})))$$

optimization ?

Nonlinearity from composition

1-layer

$$f(\boldsymbol{W}) = \boldsymbol{w} f_1(W_1 \boldsymbol{x}) \qquad \text{loss} \qquad (\boldsymbol{w} f_1(W_1 \boldsymbol{x}) - y)^2$$

$$\frac{\partial(\boldsymbol{w} f_1(W_1 \boldsymbol{x}) - y)^2}{\partial \boldsymbol{w}} = 2(\boldsymbol{w} f_1(W_1 \boldsymbol{x}) - y) \cdot f_1(W_1 \boldsymbol{x})$$

$$2(\boldsymbol{w} f_1(W_1 \boldsymbol{x}) - y) \cdot \boldsymbol{w} = \Delta_1$$

$$\frac{\partial(\boldsymbol{w} f_1(W_1 \boldsymbol{x}) - y)^2}{\partial W_1} = 2(\boldsymbol{w} f_1(W_1 \boldsymbol{x}) - y) \cdot \boldsymbol{w} \frac{\partial f_1(W_1 \boldsymbol{x})}{\partial W_1} \cdot \boldsymbol{x}$$

$$= \Delta_1 \frac{\partial f_1(W_1 \boldsymbol{x})}{\partial W_1} \cdot \text{input}$$

Nonlinearity from composition

k-layer

$$f(\boldsymbol{W}) = f_k(W_k \cdots f_2(W_1 f_1(W_1 \boldsymbol{x})))$$

loss $(\boldsymbol{w} f(\boldsymbol{W}) - y)^2$

$$\frac{\partial (f(\boldsymbol{W}) - y)^2}{\partial \boldsymbol{w}} = \text{err} \cdot \text{input}$$

$$\cdot \boldsymbol{w} = \Delta_k$$

$$\frac{\partial (f(\boldsymbol{W}) - y)^2}{\partial W_k} = \Delta_k \frac{\partial f_1(W_k \cdot \text{input})}{\partial W_k} \cdot \text{input}$$

$$\Delta_k \cdot \frac{\partial f_k(W_k f_{k-1})}{\partial f_{k-1}} = \Delta_{k-1}$$

$$\frac{\partial (f(\boldsymbol{W}) - y)^2}{\partial W_i} = \Delta_i \frac{\partial f_1(W_i \cdot \text{input})}{\partial W_i} \cdot \text{input}$$

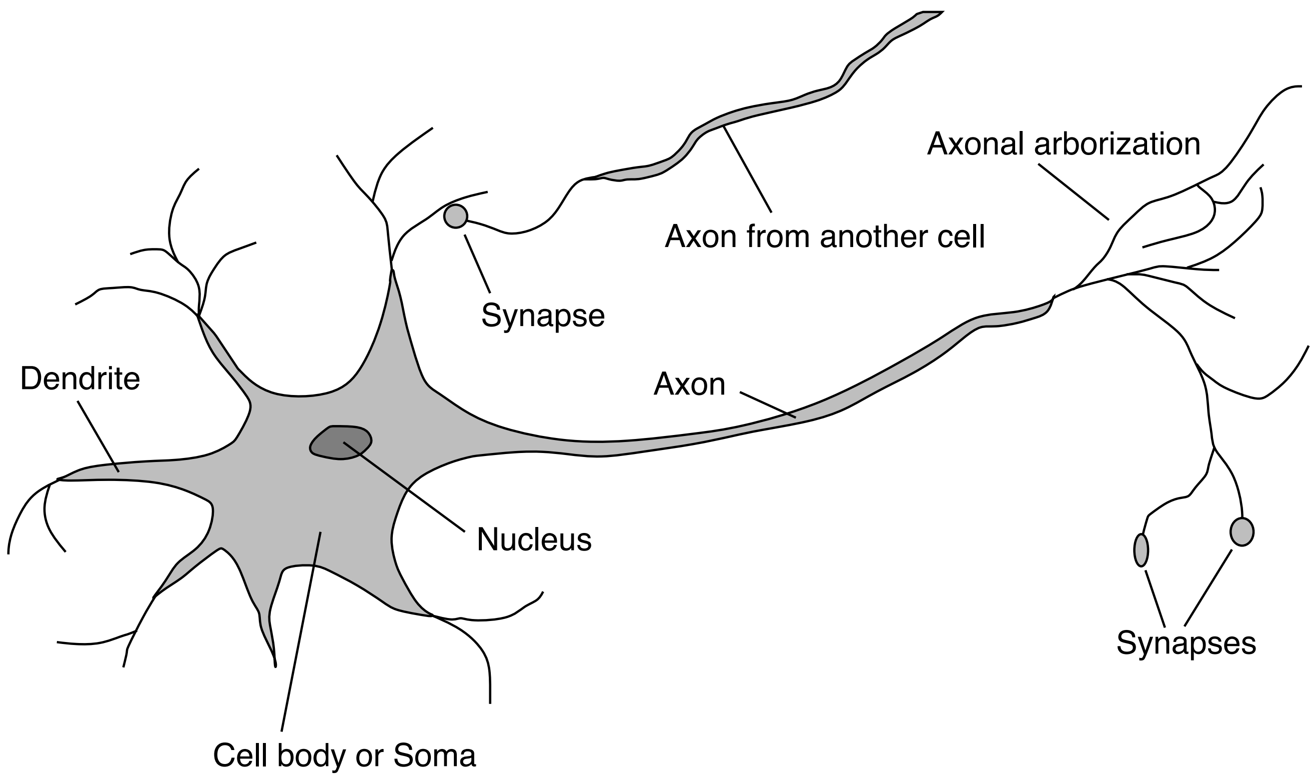
$$\Delta_i \cdot \frac{\partial f_i(W_i f_{i-1})}{\partial f_{i-1}} = \Delta_{i-1}$$

back-propagation

Biological neurons

10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time

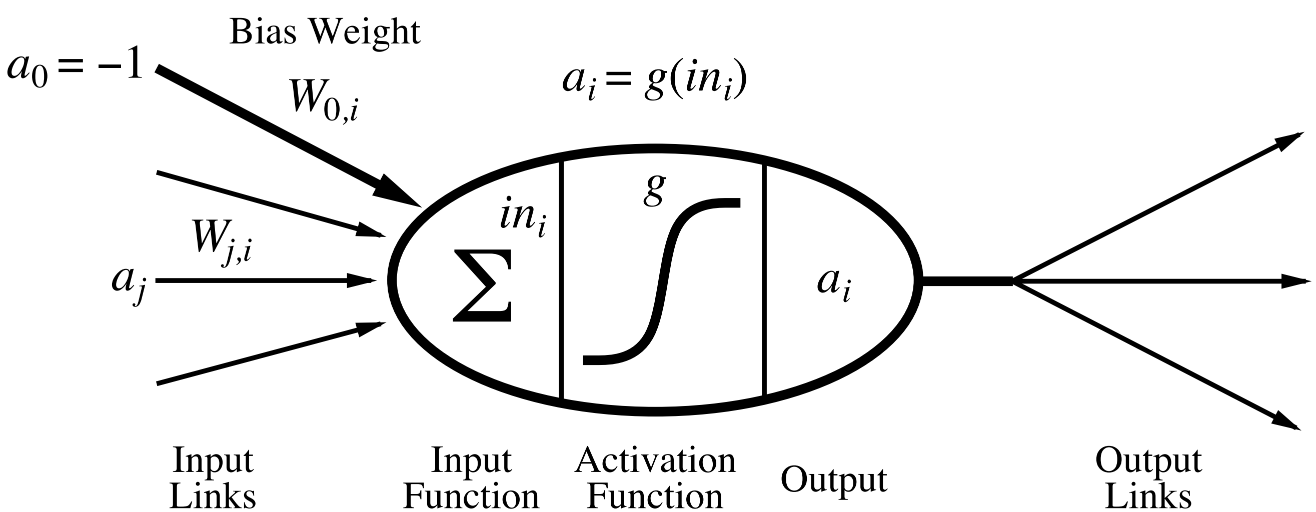
Signals are noisy "spike trains" of electrical potential



McCulloch–Pitts "unit"

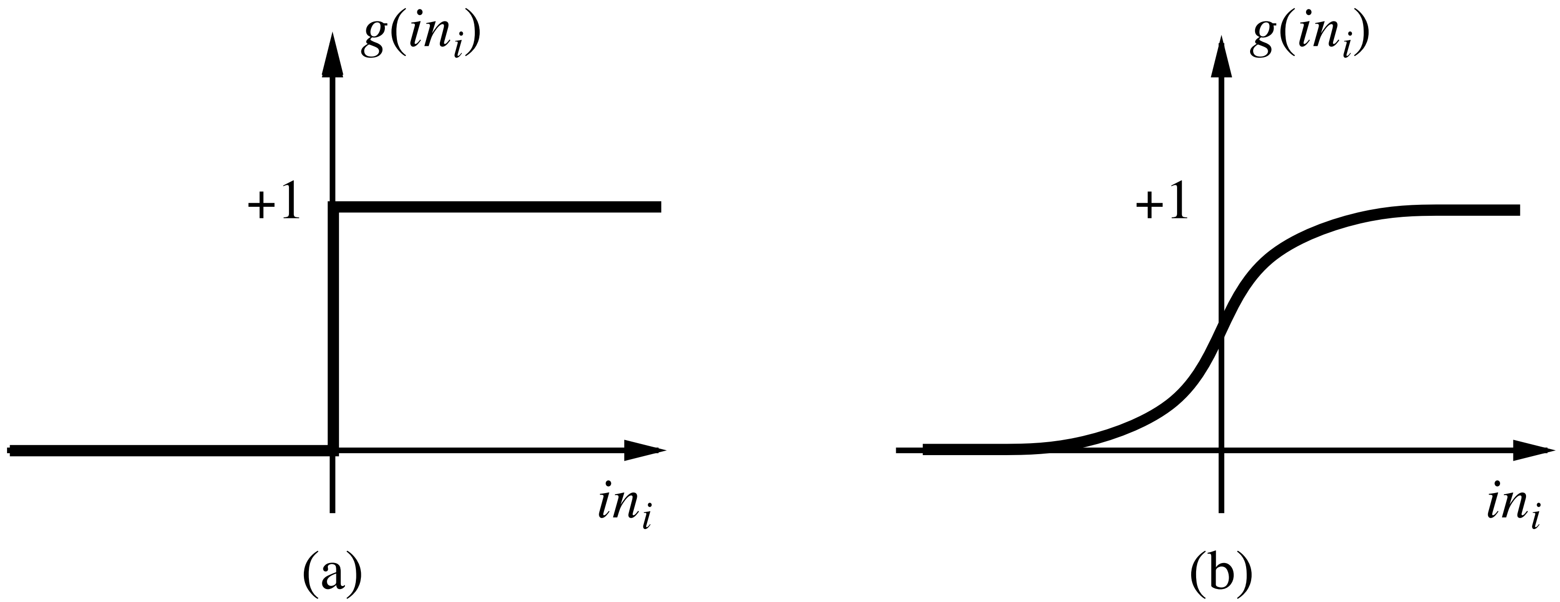
Output is a "squashed" linear function of the inputs:

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions

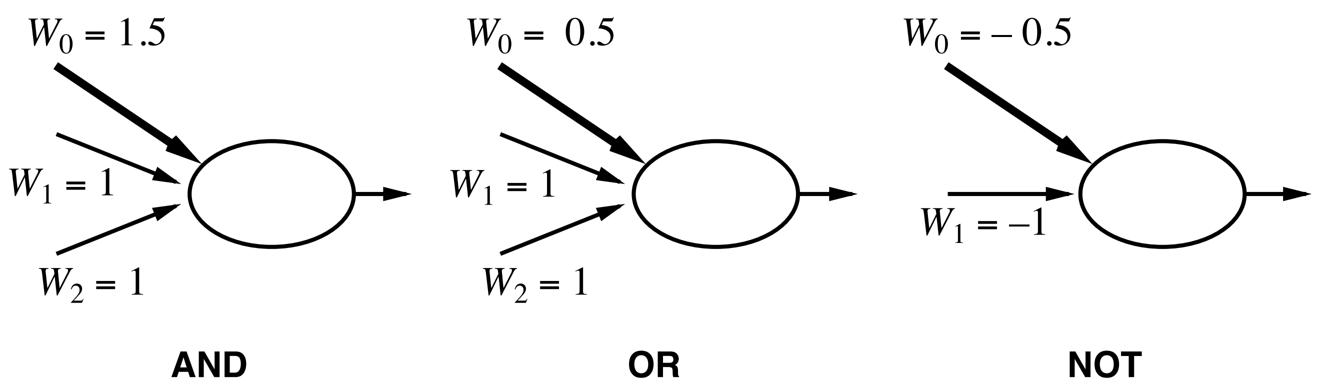


(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

Network structures

Feed-forward networks:

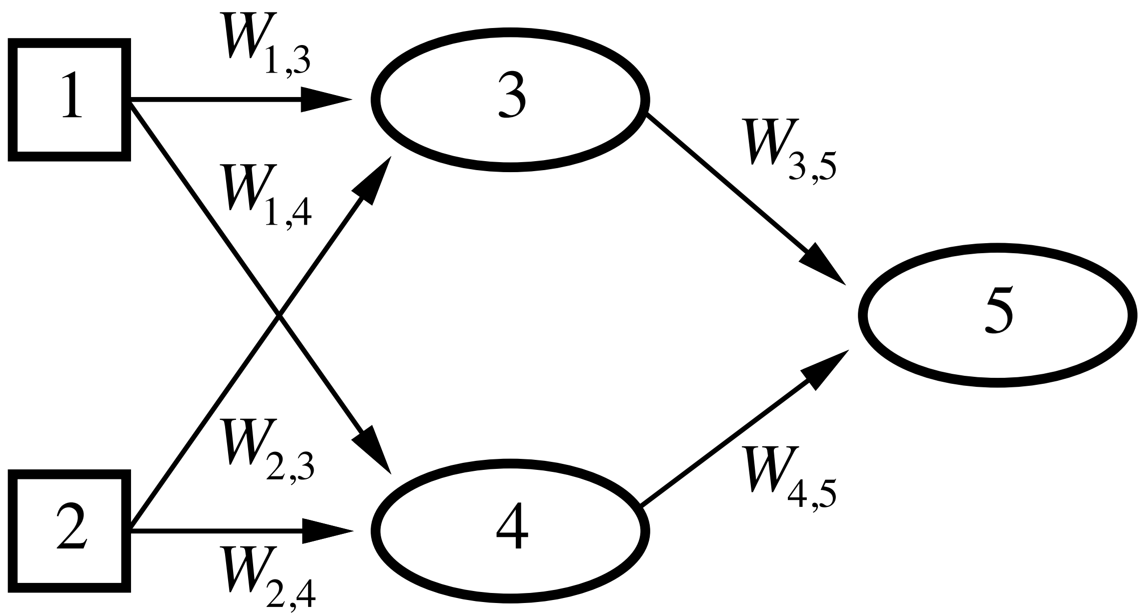
- single-layer perceptrons
- multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Recurrent networks:

- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
  $g(x) = \mathsf{sign}(x)$, $a_i = \pm 1$; **holographic associative memory**
- Boltzmann machines use stochastic activation functions,
  $\approx$ MCMC in Bayes nets
- recurrent neural nets have directed cycles with delays
  $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

Feed-forward example

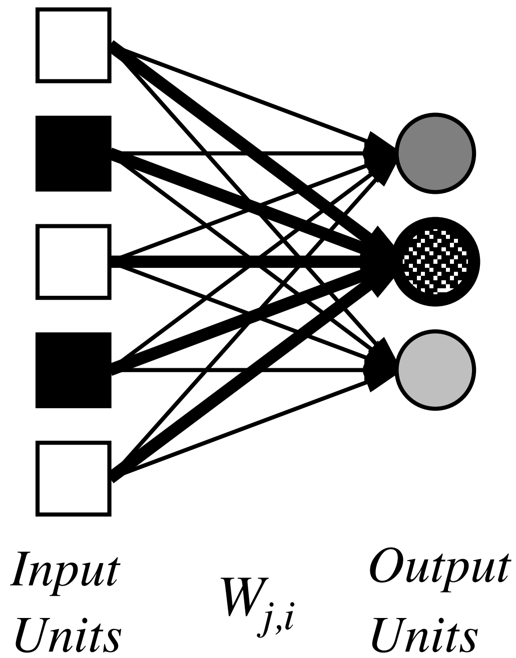


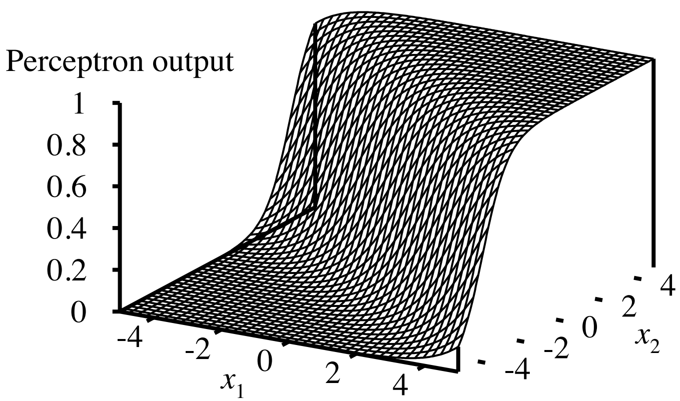
Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

Adjusting weights changes the function: do learning this way!

Single-layer perceptrons





Output units all operate separately—no shared weights

Adjusting weights moves the location, orientation, and steepness of cliff

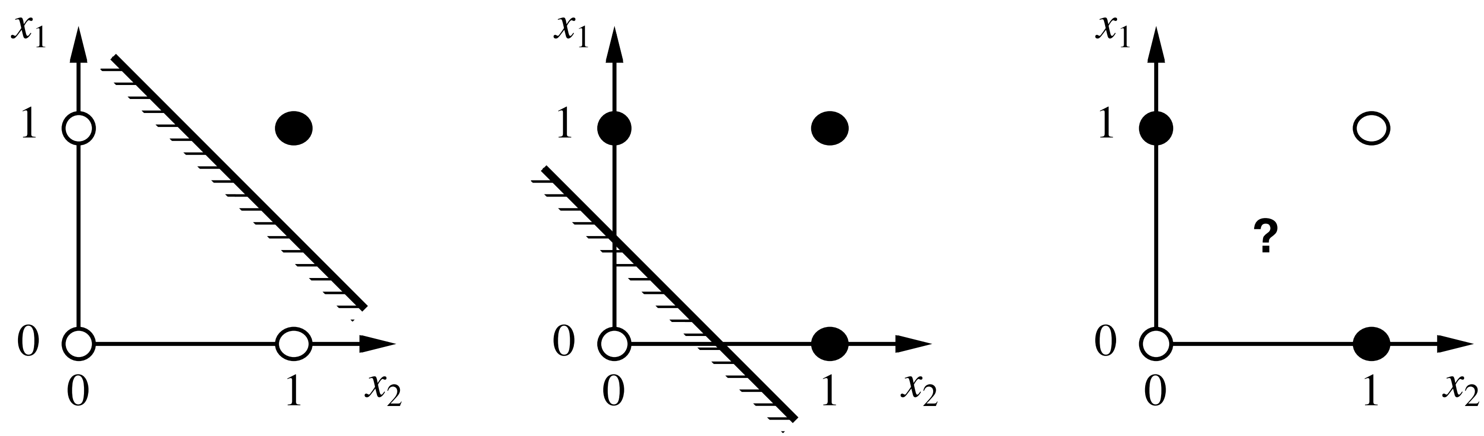
Expressiveness of perceptrons

Consider a perceptron with $g =$ step function (Rosenblatt, 1957, 1960)

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space:

$\Sigma_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$

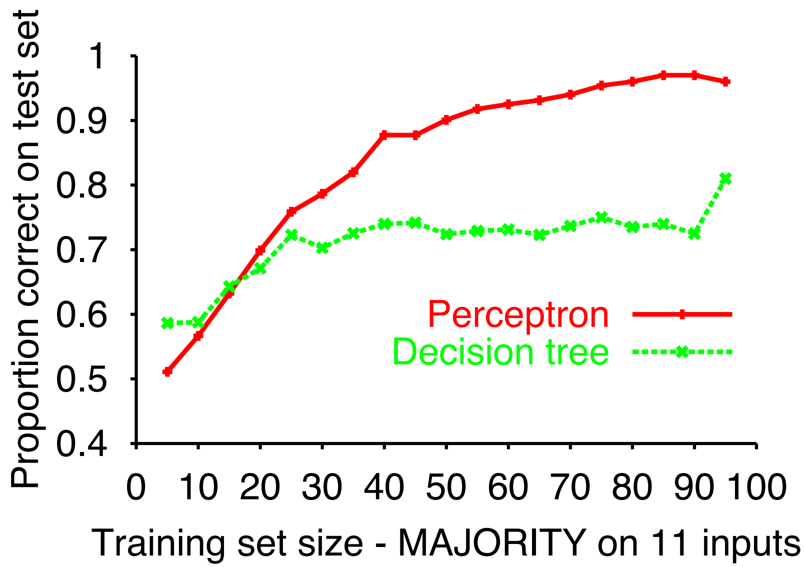


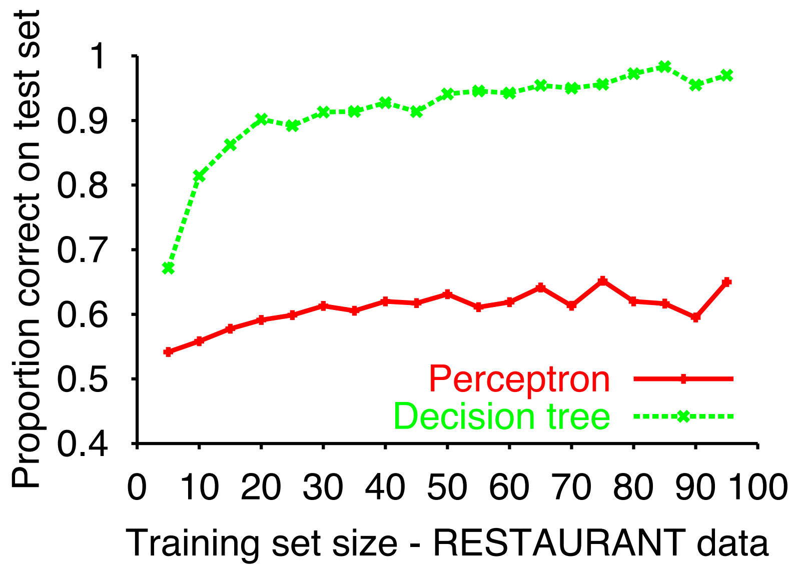
(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Minsky & Papert (1969) pricked the neural network balloon

Perceptron learning contd.

Perceptron learning rule converges to a consistent function
for any linearly separable data set



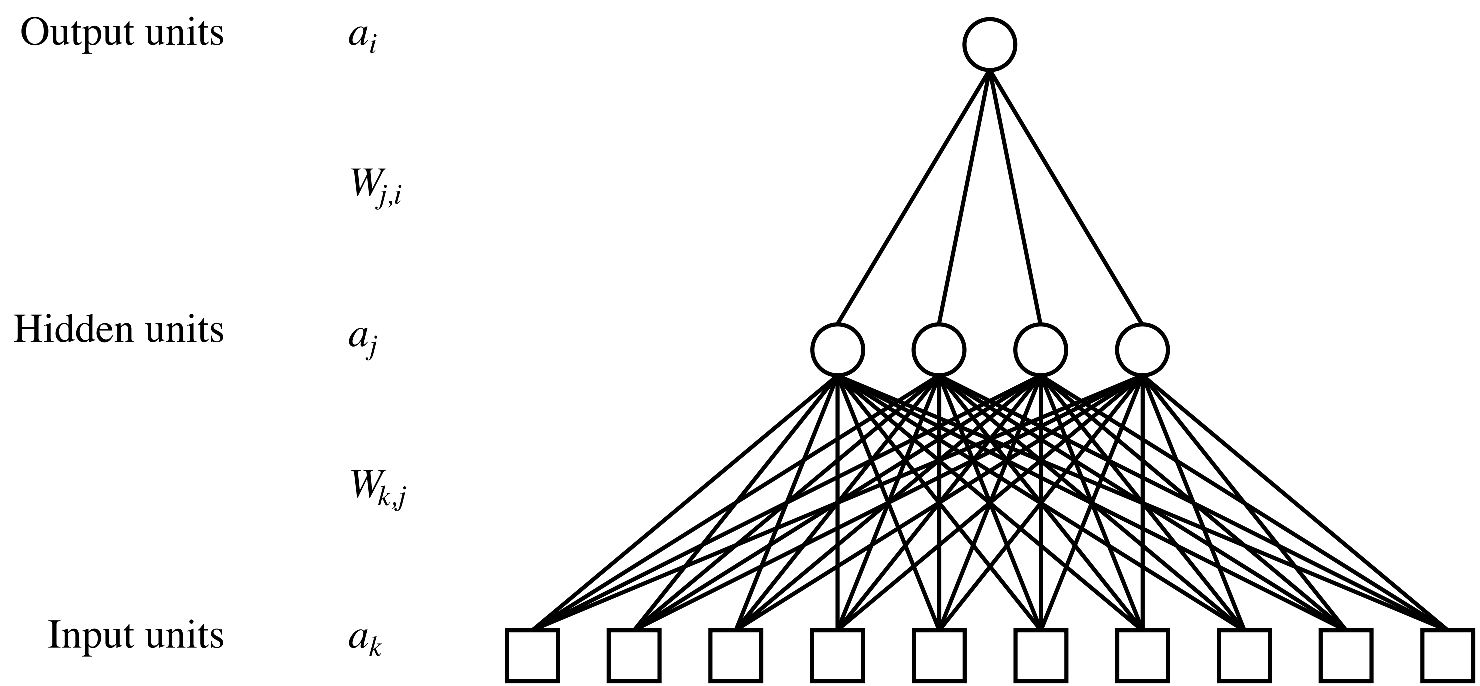


Perceptron learns majority function easily, DTL is hopeless

DTL learns restaurant function easily, perceptron cannot represent it

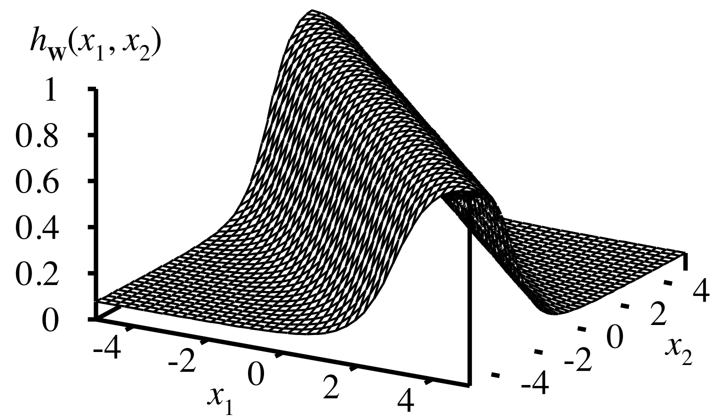
Multilayer perceptrons

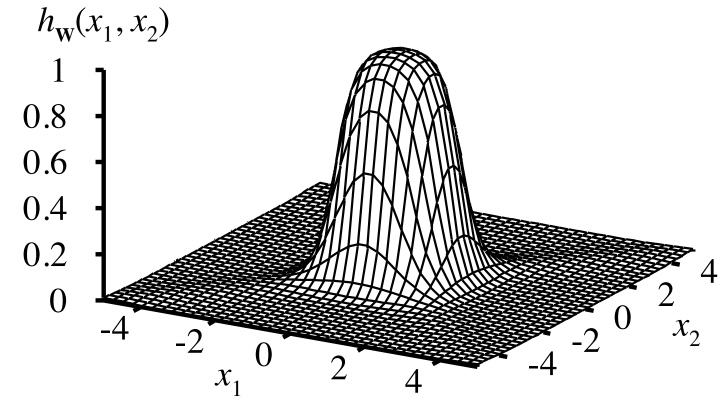
Layers are usually fully connected;
numbers of hidden units typically chosen by hand



Expressiveness of MLPs

All continuous functions w/ 2 layers, all functions w/ 3 layers





Combine two opposite-facing threshold functions to make a ridge

Combine two perpendicular ridges to make a bump

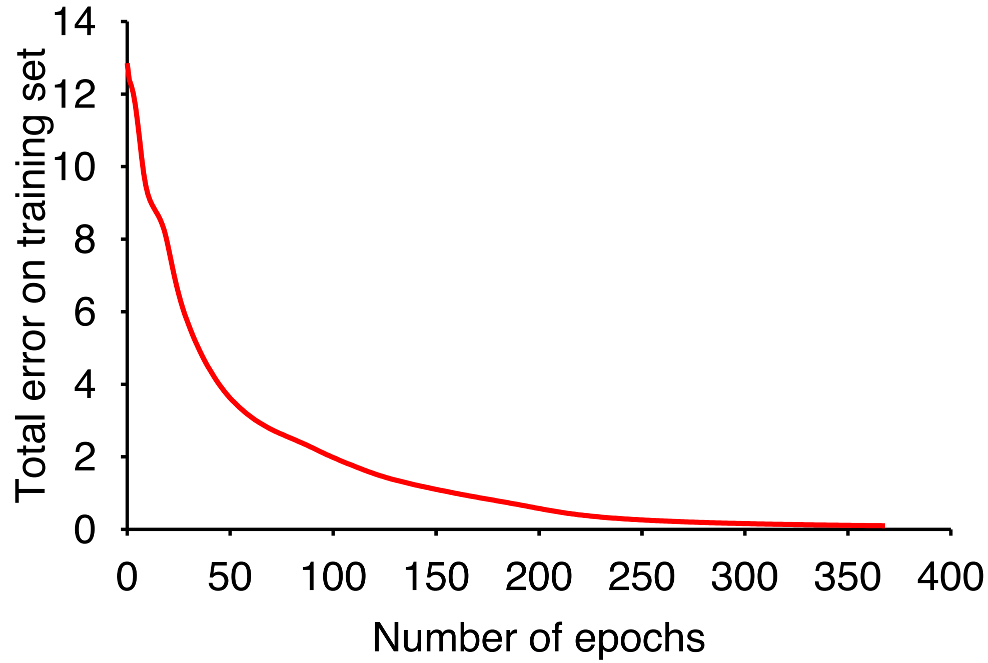
Add bumps of various sizes and locations to fit any surface

Proof requires exponentially many hidden units (cf DTL proof)

Back-propagation learning

At each epoch, sum gradient updates for all examples and apply

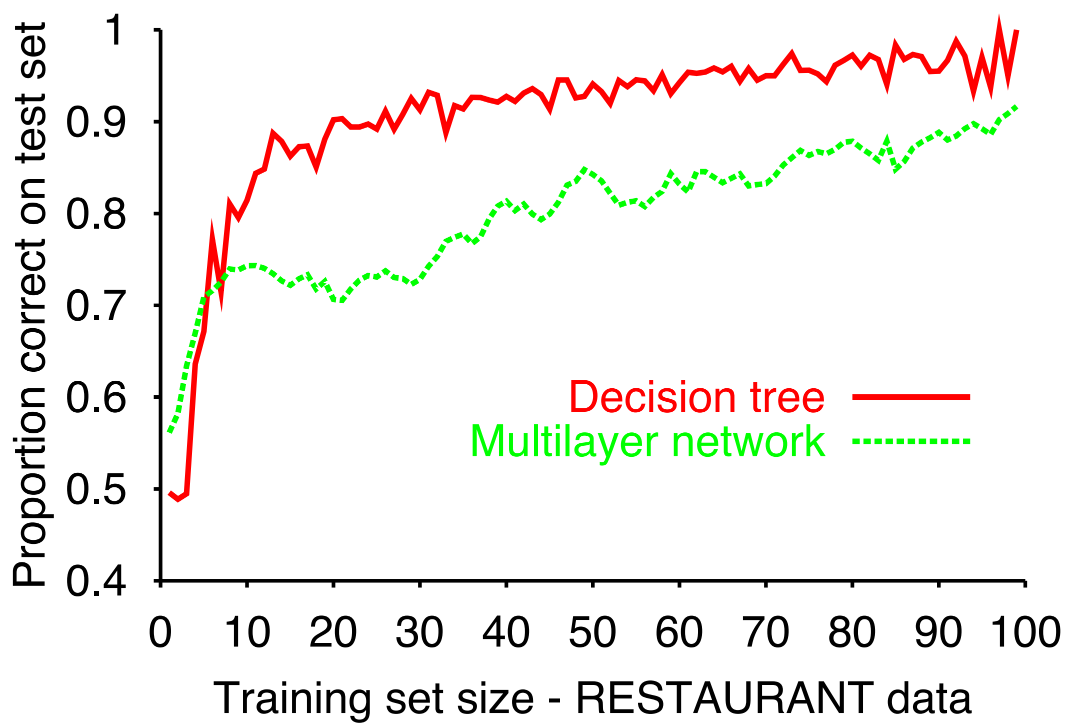
Training curve for 100 restaurant examples: finds exact fit



Typical problems: slow convergence, local minima

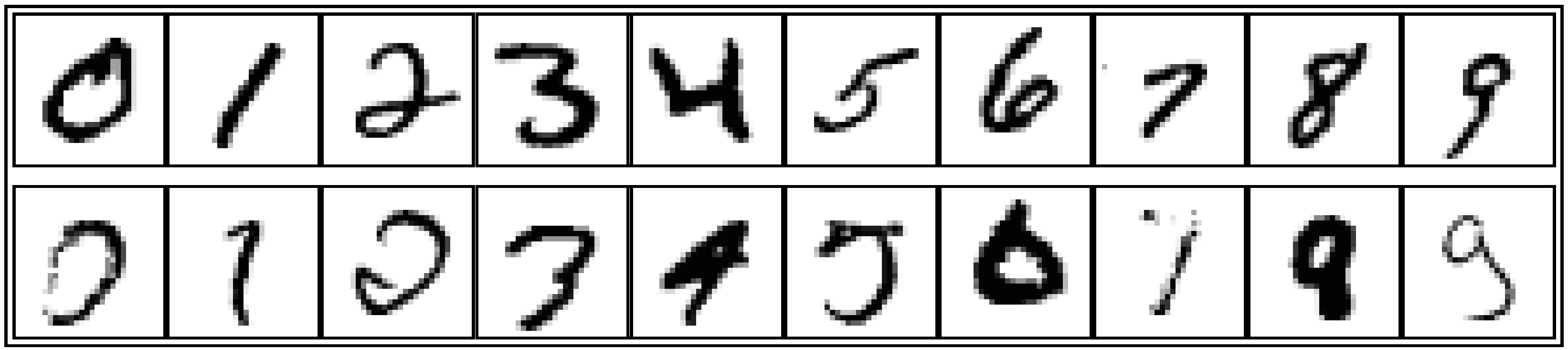
Back-propagation learning

Learning curve for MLP with 4 hidden units:



MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily

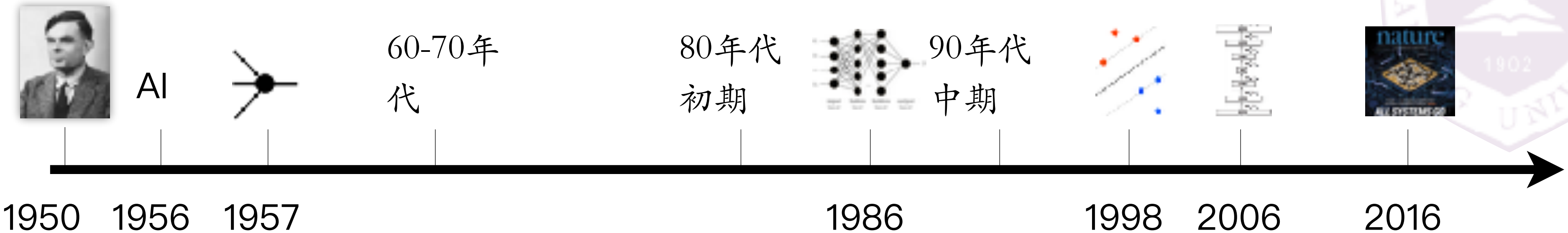
Handwritten digit recognition



3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error

Current best (kernel machines, vision algorithms) ≈ 0.6% error

A little history



Geoff Hinton