

2018 EE448, Big Data Mining, Lecture 5

# Supervised Learning (Part II)

Weinan Zhang

Shanghai Jiao Tong University

<http://wnzhang.net>

<http://wnzhang.net/teaching/ee448/index.html>

# Content of Supervised Learning

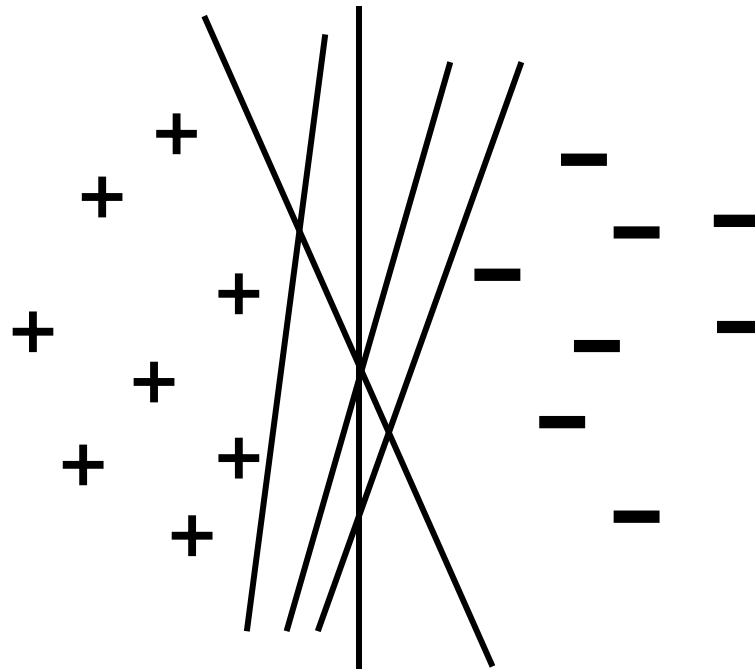
- Introduction to Machine Learning
- Linear Models
- Support Vector Machines
- Neural Networks
- Tree Models
- Ensemble Methods

# Content of This Lecture

- Support Vector Machines
- Neural Networks

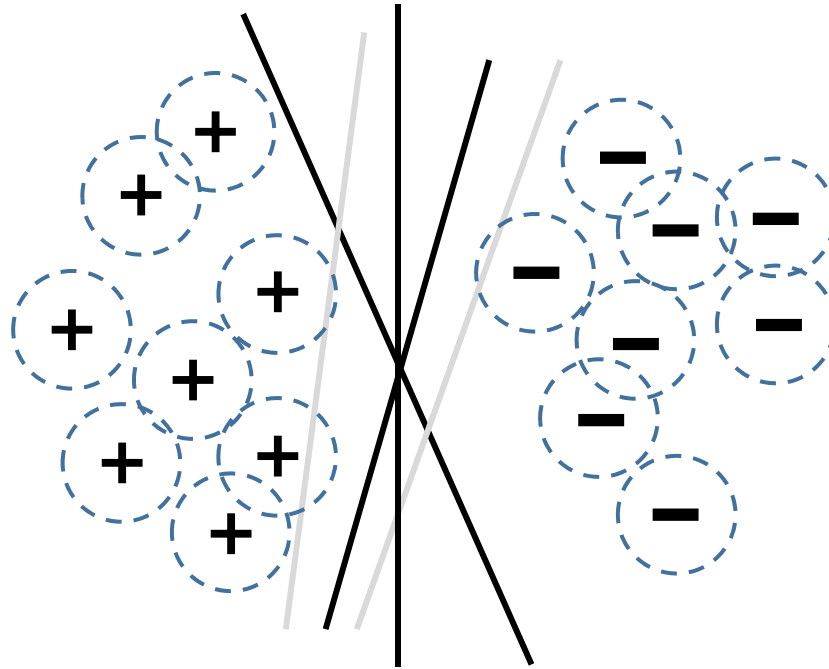
# Linear Classification

- For linear separable cases, we have multiple decision boundaries



# Linear Classification

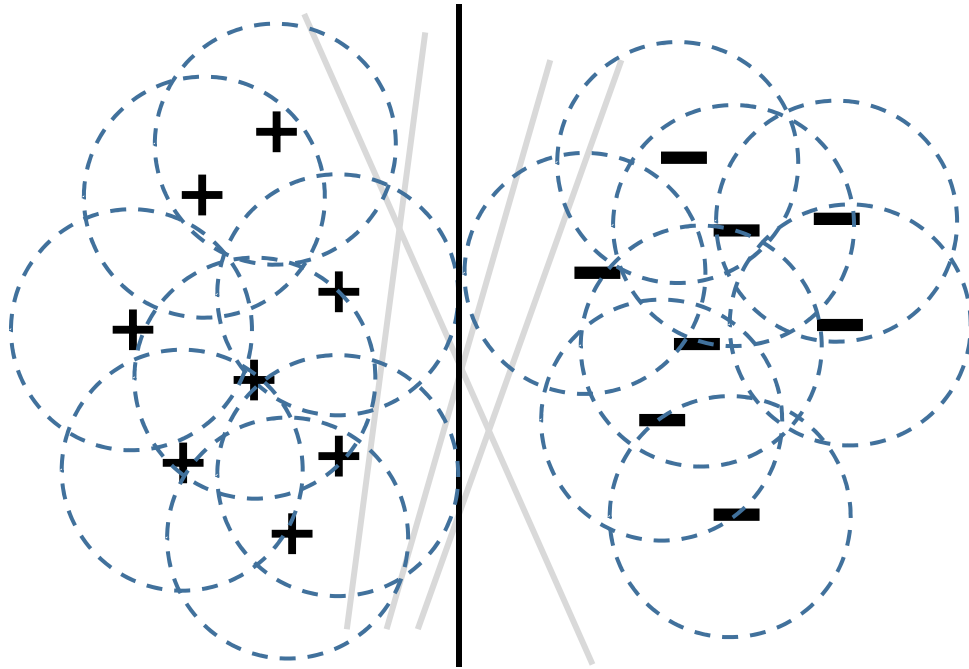
- For linear separable cases, we have multiple decision boundaries



- Ruling out some separators by considering data noise

# Linear Classification

- For linear separable cases, we have multiple decision boundaries

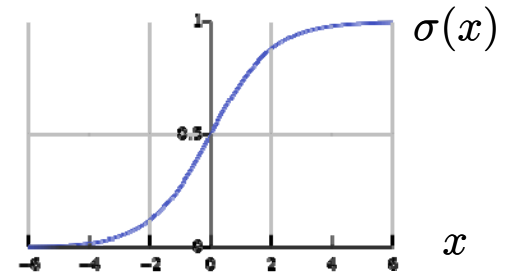


- The intuitive optimal decision boundary: the largest margin

# Review: Logistic Regression

- Logistic regression is a binary classification model

$$p_{\theta}(y = 1|x) = \sigma(\theta^{\top} x) = \frac{1}{1 + e^{-\theta^{\top} x}}$$
$$p_{\theta}(y = 0|x) = \frac{e^{-\theta^{\top} x}}{1 + e^{-\theta^{\top} x}}$$



- Cross entropy loss function

$$\mathcal{L}(y, x, p_{\theta}) = -y \log \sigma(\theta^{\top} x) - (1 - y) \log(1 - \sigma(\theta^{\top} x))$$

- Gradient

$$\frac{\partial \mathcal{L}(y, x, p_{\theta})}{\partial \theta} = -y \frac{1}{\sigma(\theta^{\top} x)} \sigma(z)(1 - \sigma(z))x - (1 - y) \frac{-1}{1 - \sigma(\theta^{\top} x)} \sigma(z)(1 - \sigma(z))x$$
$$= (\sigma(\theta^{\top} x) - y)x$$
$$\theta \leftarrow \theta + \eta(y - \sigma(\theta^{\top} x))x$$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$

# Label Decision

- Logistic regression provides the probability

$$p_{\theta}(y = 1|x) = \sigma(\theta^{\top} x) = \frac{1}{1 + e^{-\theta^{\top} x}}$$

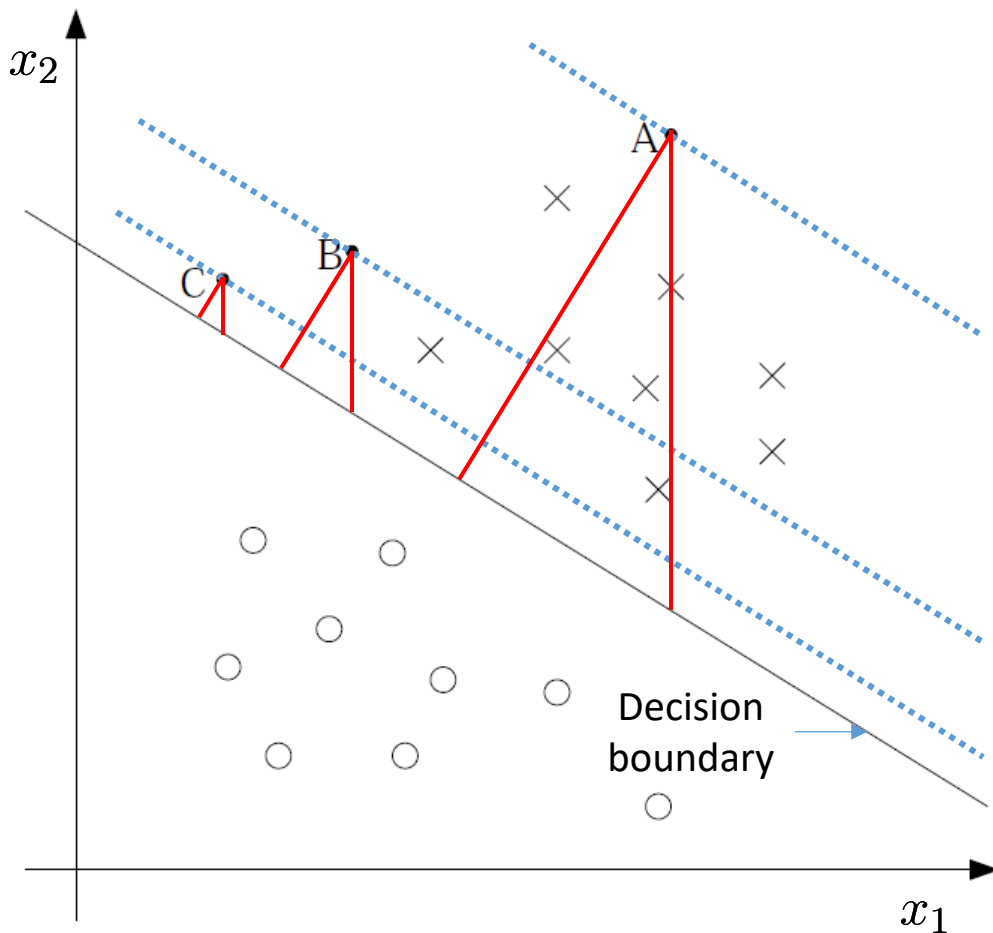
$$p_{\theta}(y = 0|x) = \frac{e^{-\theta^{\top} x}}{1 + e^{-\theta^{\top} x}}$$

- The final label of an instance is decided by setting a threshold  $h$

$$\hat{y} = \begin{cases} 1, & p_{\theta}(y = 1|x) > h \\ 0, & \text{otherwise} \end{cases}$$



# Logistic Regression Scores



$$s(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$p_{\theta}(y = 1|x) = \frac{1}{1 + e^{-s(x)}}$$

$$s(x) = \theta_0 + \theta_1 x_1^{(A)} + \theta_2 x_2^{(A)}$$

$$s(x) = \theta_0 + \theta_1 x_1^{(B)} + \theta_2 x_2^{(B)}$$

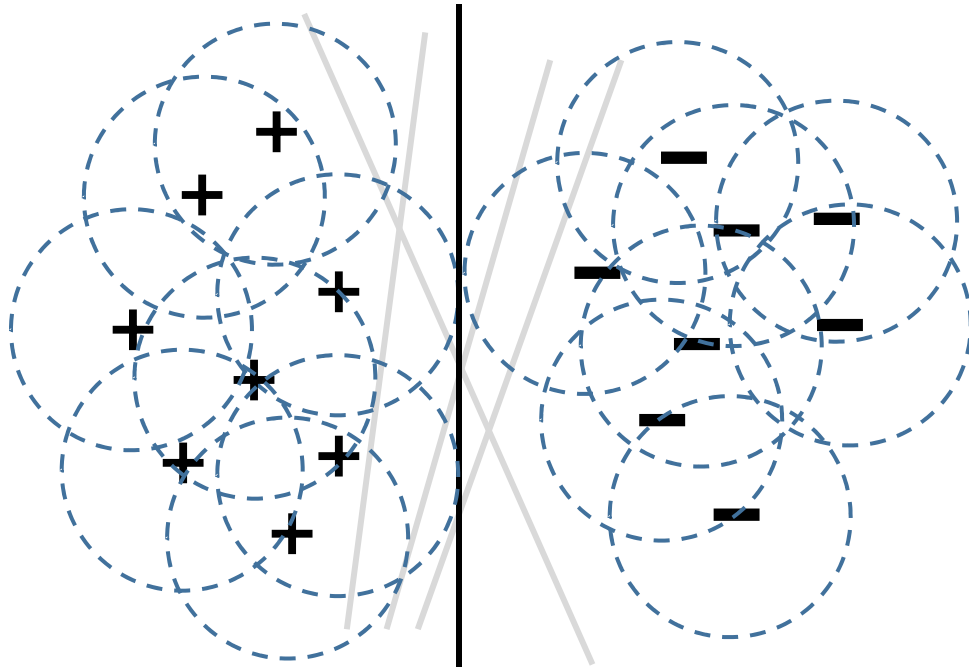
$$s(x) = \theta_0 + \theta_1 x_1^{(C)} + \theta_2 x_2^{(C)}$$

$$s(x) = 0$$

The higher score, the larger distance to the decision boundary, the higher confidence

# Linear Classification

- The intuitive optimal decision boundary: the highest confidence



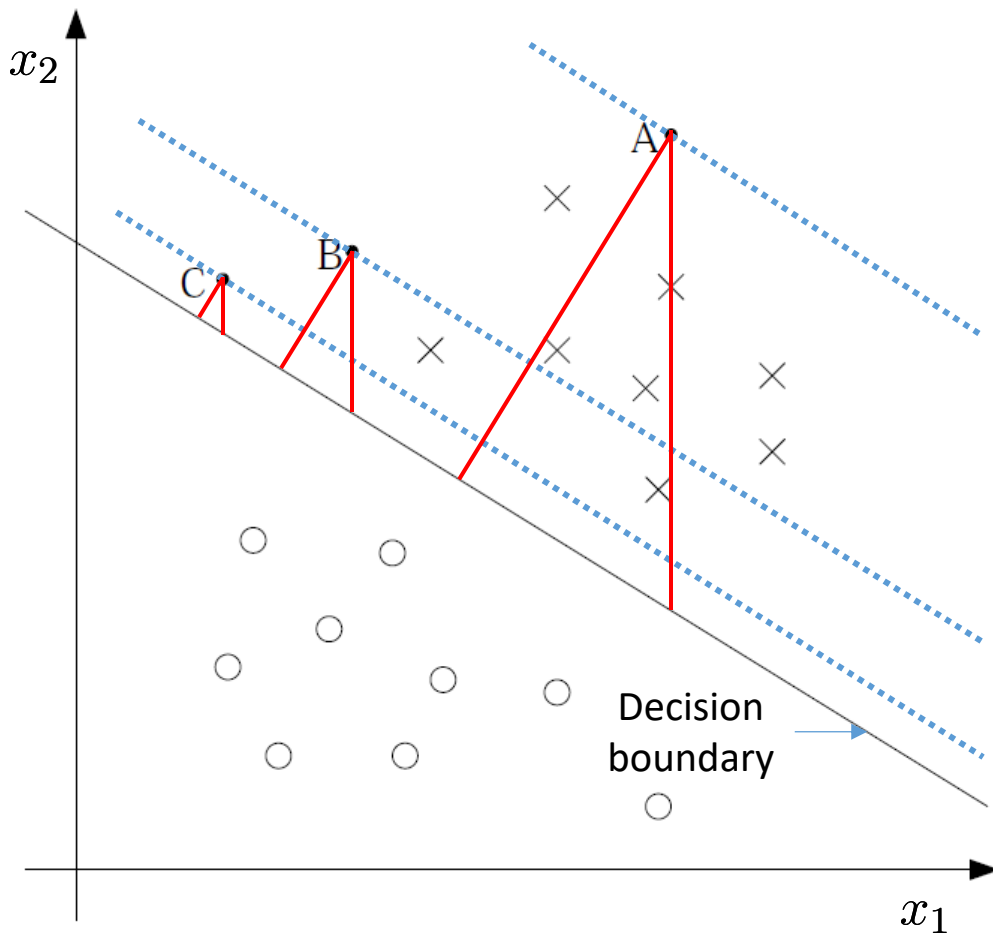
# Notations for SVMs

- Feature vector  $x$
- Class label  $y \in \{-1, 1\}$
- Parameters
  - Intercept  $b$
  - Feature weight vector  $w$
- Label prediction

$$h_{w,b}(x) = g(w^\top x + b)$$

$$g(z) = \begin{cases} +1 & z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

# Logistic Regression Scores



$$s(x) = b + w_1x_1 + w_2x_2$$

$$p_{\theta}(y = 1|x) = \frac{1}{1 + e^{-s(x)}}$$

$$s(x) = b + w_1x_1^{(A)} + w_2x_2^{(A)}$$

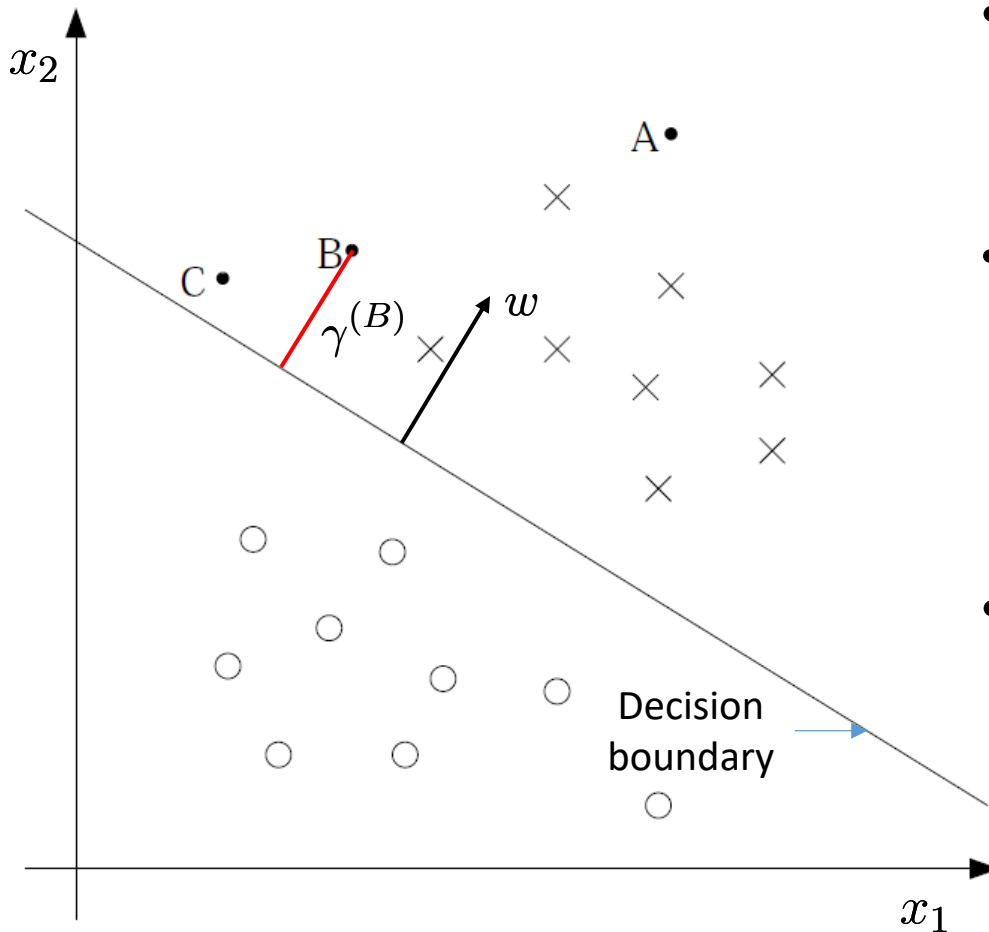
$$s(x) = b + w_1x_1^{(B)} + w_2x_2^{(B)}$$

$$s(x) = b + w_1x_1^{(C)} + w_2x_2^{(C)}$$

$$s(x) = 0$$

The higher score, the larger distance to the separating hyperplane, the higher confidence

# Margins



- Functional margin

$$\hat{\gamma}^{(i)} = y^{(i)}(w^\top x^{(i)} + b)$$

- Note that the separating hyperplane won't change with the magnitude of  $(w, b)$

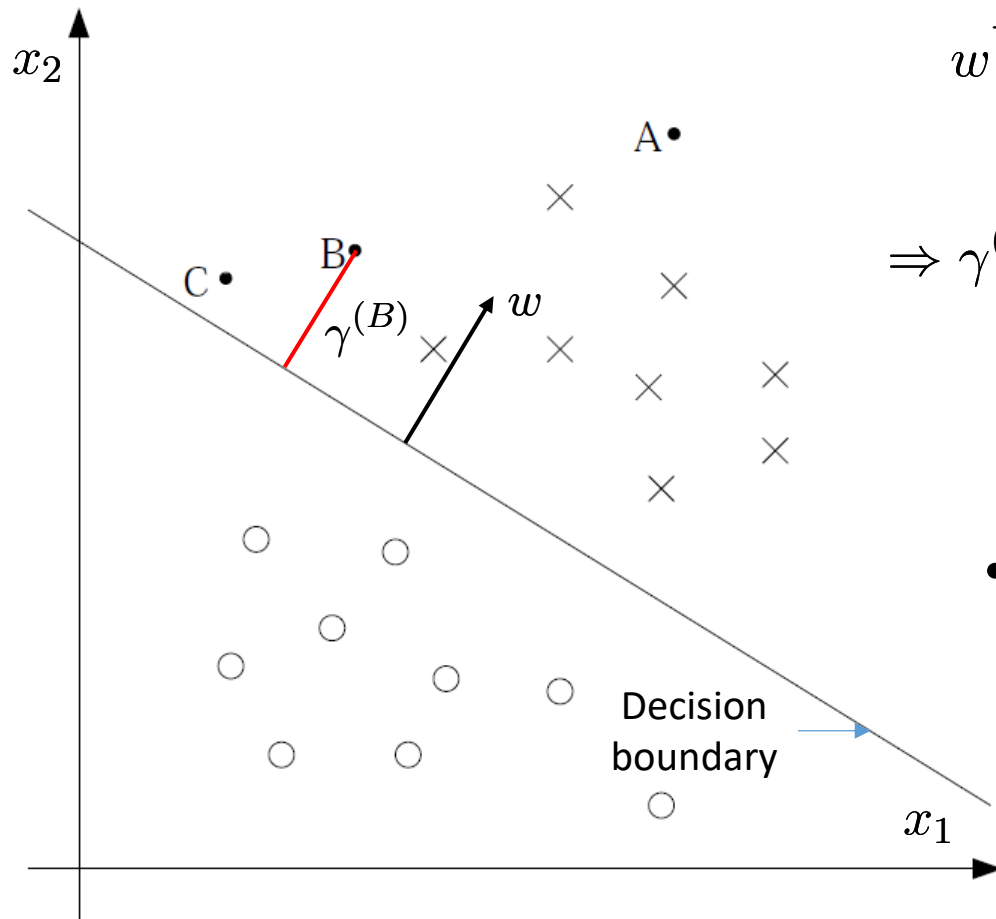
$$g(w^\top x + b) = g(2w^\top x + 2b)$$

- Geometric margin

$$\gamma^{(i)} = y^{(i)}(w^\top x^{(i)} + b)$$

$$\text{where } \|w\|^2 = 1$$

# Margins



- Decision boundary

$$w^T \left( x^{(i)} - \gamma^{(i)} y^{(i)} \frac{w}{\|w\|} \right) + b = 0$$

$$\begin{aligned} \Rightarrow \gamma^{(i)} &= y^{(i)} \frac{w^T x^{(i)} + b}{\|w\|} \\ &= y^{(i)} \left( \left( \frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|} \right) \end{aligned}$$

- Given a training set

$$S = \{(x_i, y_i)\}_{i=1 \dots m}$$

the smallest geometric margin

$$\gamma = \min_{i=1 \dots m} \gamma^{(i)}$$

# Objective of an SVM

- Find a separable hyperplane that maximizes the minimum geometric margin

$$\begin{aligned} \max_{\gamma, w, b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^\top x^{(i)} + b) \geq \gamma, \quad i = 1, \dots, m \\ & \|w\| = 1 \quad (\text{non-convex constraint}) \end{aligned}$$

- Equivalent to normalized functional margin

$$\begin{aligned} \max_{\hat{\gamma}, w, b} \quad & \frac{\hat{\gamma}}{\|w\|} \quad (\text{non-convex objective}) \\ \text{s.t.} \quad & y^{(i)}(w^\top x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \dots, m \end{aligned}$$

# Objective of an SVM

- Functional margin scales w.r.t.  $(w, b)$  without changing the decision boundary.
  - Let's fix the functional margin at 1.

$$\hat{\gamma} = 1$$

- Objective is written as

$$\begin{aligned} \max_{w, b} \quad & \frac{1}{\|w\|} \\ \text{s.t.} \quad & y^{(i)}(w^\top x^{(i)} + b) \geq 1, \quad i = 1, \dots, m \end{aligned}$$

- Equivalent with

$$\begin{aligned} \min_{w, b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^\top x^{(i)} + b) \geq 1, \quad i = 1, \dots, m \end{aligned}$$

This optimization problem can be efficiently solved by quadratic programming



# A Digression of Lagrange Duality in Convex Optimization

Boyd, Stephen, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

# Lagrangian for Convex Optimization

- A convex optimization problem

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & h_i(w) = 0, \quad i = 1, \dots, l \end{aligned}$$

- The Lagrangian of this problem is defined as

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^l \beta_i h_i(w)$$

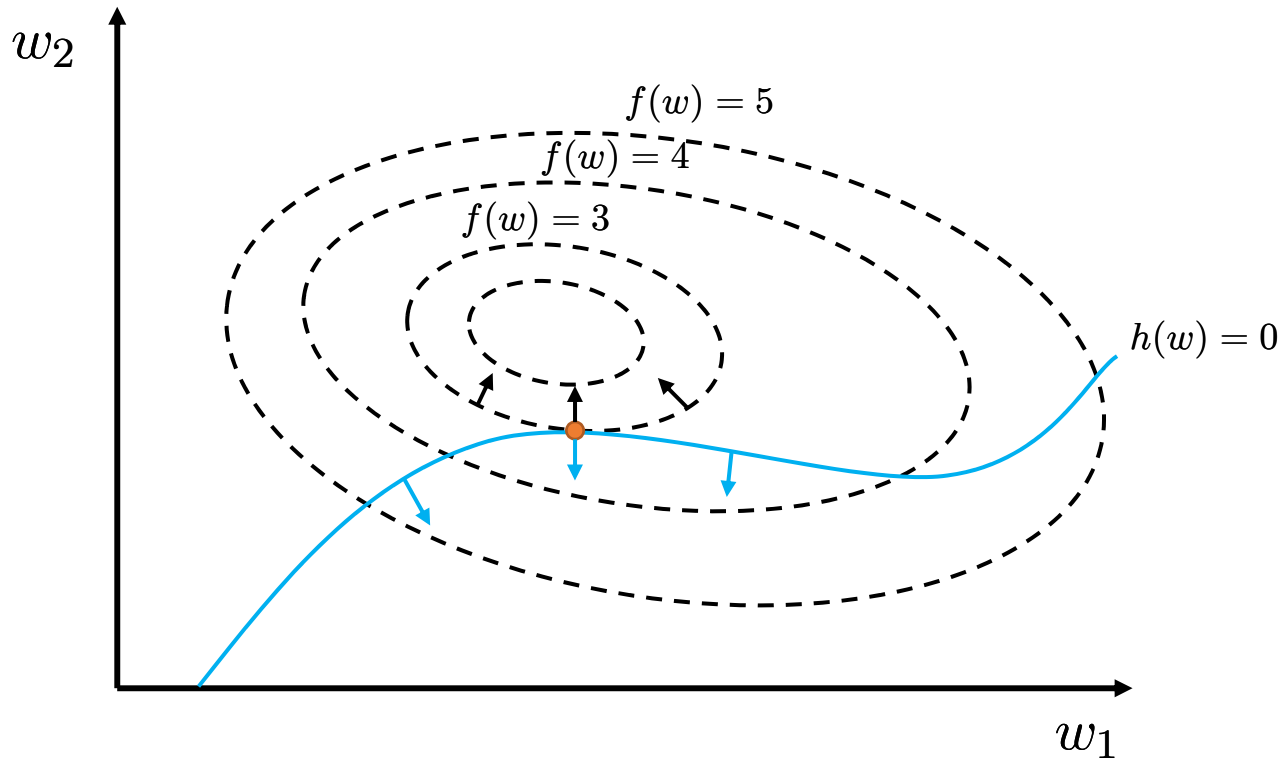
↑ Lagrangian multipliers

- Solving

$$\frac{\partial \mathcal{L}(w, \beta)}{\partial w} = 0 \quad \frac{\partial \mathcal{L}(w, \beta)}{\partial \beta} = 0$$

yields the solution of the original optimization problem.

# Lagrangian for Convex Optimization



$$\mathcal{L}(w, \beta) = f(w) + \beta h(w)$$

$$\frac{\partial \mathcal{L}(w, \beta)}{\partial w} = \frac{\partial f(w)}{\partial w} + \beta \frac{\partial h(w)}{\partial w} = 0$$

i.e., two gradients on the same direction



# Primal Problem

- A convex optimization

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \dots, k \\ & h_i(w) = 0, \quad i = 1, \dots, l \end{aligned}$$

- The Lagrangian

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^k \alpha_i g_i(w) + \sum_{i=1}^l \beta_i h_i(w)$$

- The primal problem

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

- If a given  $w$  violates any constraints, i.e.,

$$g_i(w) > 0 \quad \text{or} \quad h_i(w) \neq 0$$

- Then  $\theta_{\mathcal{P}}(w) = +\infty$

# Primal Problem

- A convex optimization

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \dots, k \\ & h_i(w) = 0, \quad i = 1, \dots, l \end{aligned}$$

- The Lagrangian

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^k \alpha_i g_i(w) + \sum_{i=1}^l \beta_i h_i(w)$$

- The primal problem

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

- Conversely, if all constraints are satisfied for  $w$
- Then  $\theta_{\mathcal{P}}(w) = f(w)$

$$\theta_{\mathcal{P}}(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ +\infty & \text{otherwise} \end{cases}$$

# Primal Problem

$$\theta_{\mathcal{P}}(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ +\infty & \text{otherwise} \end{cases}$$

- The minimization problem

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

is the same as the original problem

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \dots, k \\ & h_i(w) = 0, \quad i = 1, \dots, l \end{aligned}$$

- Define the value of the primal problem  $p^* = \min_w \theta_{\mathcal{P}}(w)$

# Dual Problem

- A slightly different problem

$$\theta_{\mathcal{D}}(\alpha, \beta) = \min_w \mathcal{L}(w, \alpha, \beta)$$

- Define the dual optimization problem

$$\max_{\alpha, \beta: \alpha_i \geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha, \beta: \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta)$$

- Min & Max exchanged compared to the primal problem

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

- Define the value of the dual problem

$$d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta)$$



# Primal Problem vs. Dual Problem

$$d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) \leq \min_w \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = p^*$$

- Proof

$$\min_w \mathcal{L}(w, \alpha, \beta) \leq \mathcal{L}(w, \alpha, \beta), \forall w, \alpha \geq 0, \beta$$

$$\Rightarrow \max_{\alpha, \beta: \alpha \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) \leq \max_{\alpha, \beta: \alpha \geq 0} \mathcal{L}(w, \alpha, \beta), \forall w$$

$$\Rightarrow \max_{\alpha, \beta: \alpha \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) \leq \min_w \max_{\alpha, \beta: \alpha \geq 0} \mathcal{L}(w, \alpha, \beta)$$

□

- But under certain condition  $d^* = p^*$

# Karush-Kuhn-Tucker (KKT) Conditions

- If  $f$  and  $g_i$ 's are convex and  $h_i$ 's are affine, and suppose  $g_i$ 's are all strictly feasible
- then there must exist  $w^*, \alpha^*, \beta^*$ 
  - $w^*$  is the solution of the primal problem
  - $\alpha^*, \beta^*$  are the solutions of the dual problem
  - and the values of the two problems are equal  $p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$
- And  $w^*, \alpha^*, \beta^*$  satisfy the KKT conditions

$$\frac{\partial}{\partial w_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \dots, n$$

$$\frac{\partial}{\partial \beta_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \dots, l$$

KKT dual  
complementarity  
condition

$$\longrightarrow \alpha_i^* g_i(w^*) = 0, \quad i = 1, \dots, k$$

$$g_i(w^*) \leq 0, \quad i = 1, \dots, k$$

$$\alpha_i^* \geq 0, \quad i = 1, \dots, k$$

- Moreover, if some  $w^*, \alpha^*, \beta^*$  satisfy the KKT conditions, then it is also a solution to the primal and dual problems.
- More details please refer to Boyd "Convex optimization" 2004.

Now Back to SVM Problem

# Objective of an SVM

- SVM objective: finding the optimal margin classifier

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^\top x^{(i)} + b) \geq 1, \quad i = 1, \dots, m \end{aligned}$$

- Re-wright the constraints as

$$g_i(w) = -y^{(i)}(w^\top x^{(i)} + b) + 1 \leq 0$$

so as to match the standard optimization form

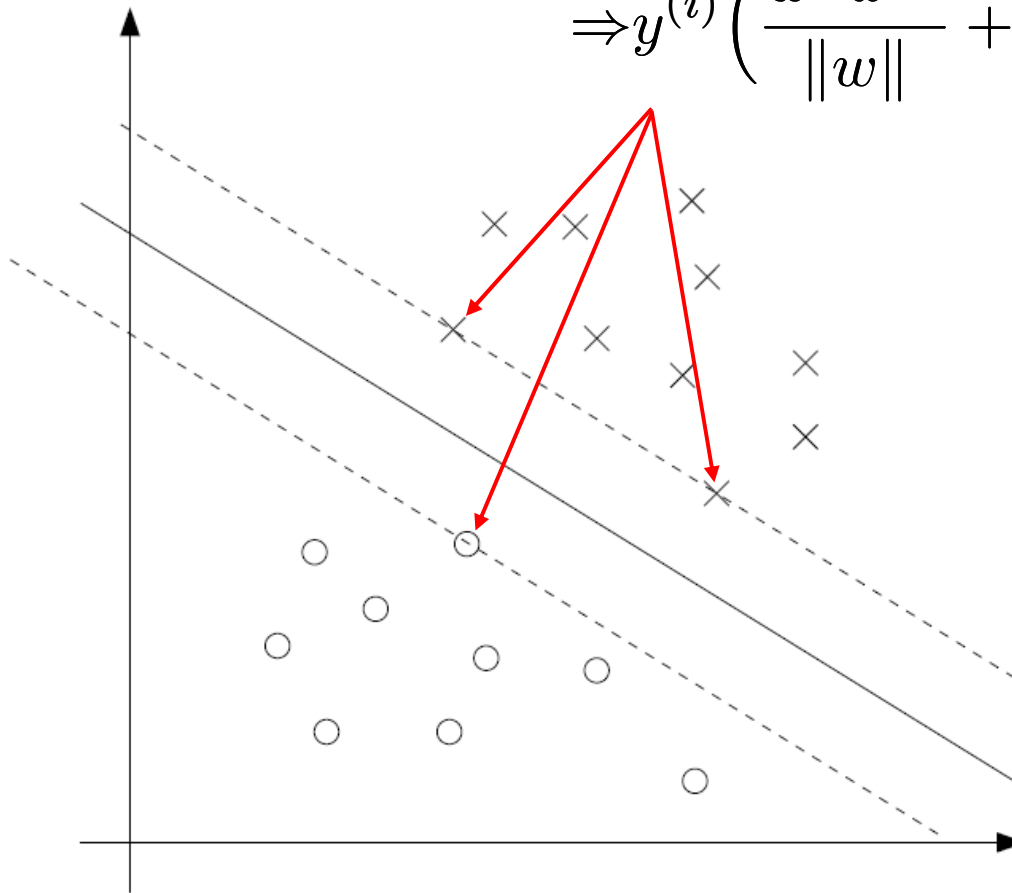
$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \dots, k \\ & h_i(w) = 0, \quad i = 1, \dots, l \end{aligned}$$

# Equality Cases

$$g_i(w) = -y^{(i)}(w^\top x^{(i)} + b) + 1 = 0$$

$$\Rightarrow y^{(i)} \left( \frac{w^\top x^{(i)}}{\|w\|} + \frac{b}{\|w\|} \right) = \frac{1}{\|w\|}$$

↑  
Geometric margin



The  $g_i$ 's = 0 cases correspond to the training examples that have functional margin exactly equal to 1.

# Objective of an SVM

- SVM objective: finding the optimal margin classifier

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & -y^{(i)}(w^\top x^{(i)} + b) + 1 \leq 0, \quad i = 1, \dots, m \end{aligned}$$

- Lagrangian

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i [y^{(i)}(w^\top x^{(i)} + b) - 1]$$

- No  $\beta$  or equality constraints in SVM problem

# Solving

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i [y^{(i)} (w^\top x^{(i)} + b) - 1]$$

- Derivatives

$$\frac{\partial}{\partial w} \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)}$$

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^m \alpha_i y^{(i)} = 0$$

- Then Lagrangian is re-written as

$$\begin{aligned} \mathcal{L}(w, b, \alpha) &= \frac{1}{2} \left\| \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \right\|^2 - \sum_{i=1}^m \alpha_i [y^{(i)} (w^\top x^{(i)} + b) - 1] \\ &= \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)\top} x^{(j)} \left[ -b \sum_{i=1}^m \alpha_i y^{(i)} \right] = 0 \end{aligned}$$

# Solving $\alpha^*$

- Dual problem

$$\max_{\alpha \geq 0} \theta_{\mathcal{D}}(\alpha) = \max_{\alpha \geq 0} \min_{w, b} \mathcal{L}(w, b, \alpha)$$

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i, j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)\top} x^{(j)}$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \dots, m$$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

- To solve  $\alpha^*$  with some methods e.g. SMO
  - We will get back to this solution later



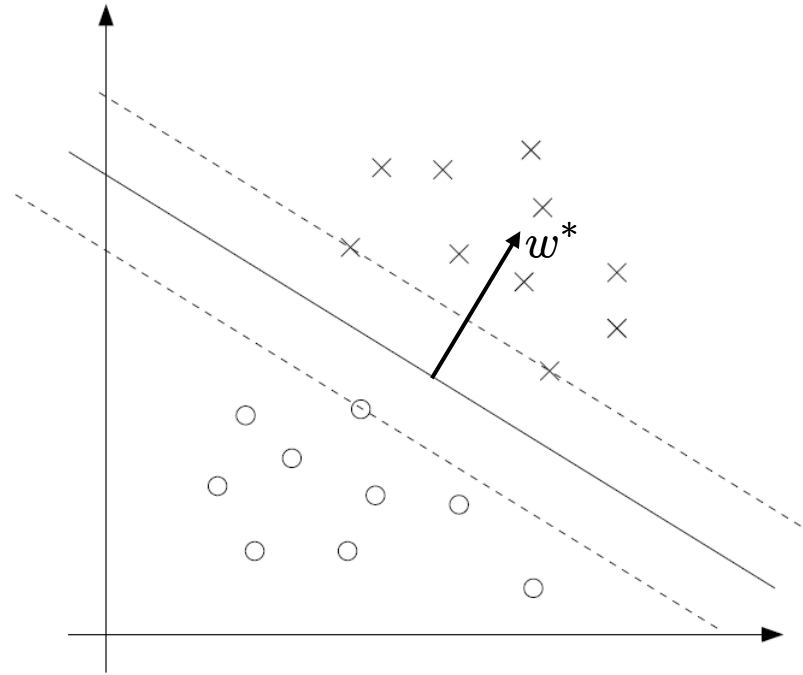
# Solving $w^*$ and $b^*$

- With  $\alpha^*$  solved,  $w^*$  is obtained by

$$w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)}$$

- Only supporting vectors with  $\alpha > 0$
- With  $w^*$  solved,  $b^*$  is obtained by

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*\top} x^{(i)} + \min_{i:y^{(i)}=1} w^{*\top} x^{(i)}}{2}$$

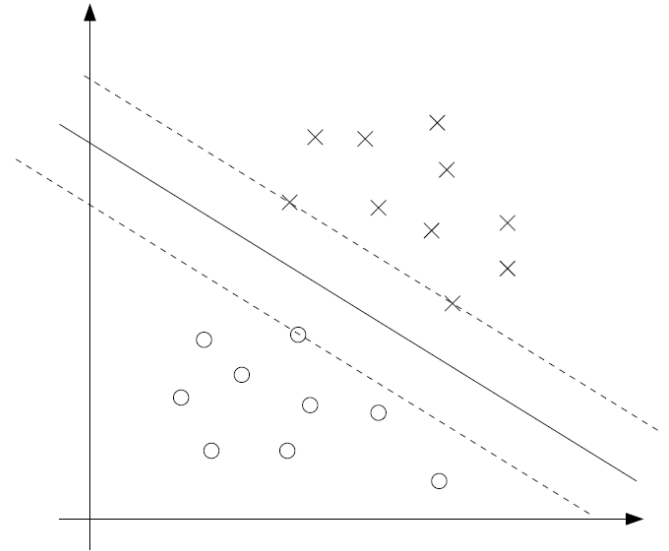


# Predicting Values

- With the solutions of  $w^*$  and  $b^*$ , the predicting value (i.e. functional margin) of each instance is

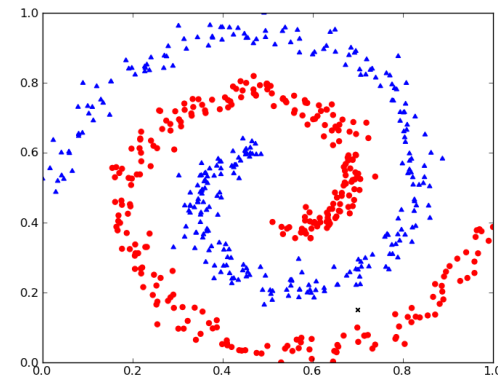
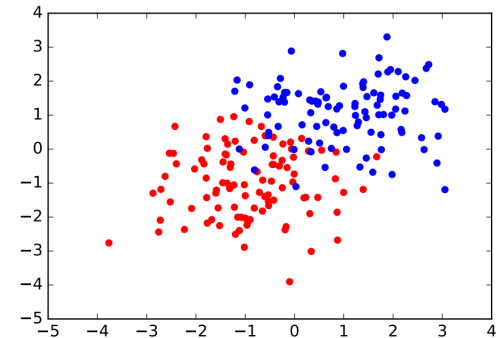
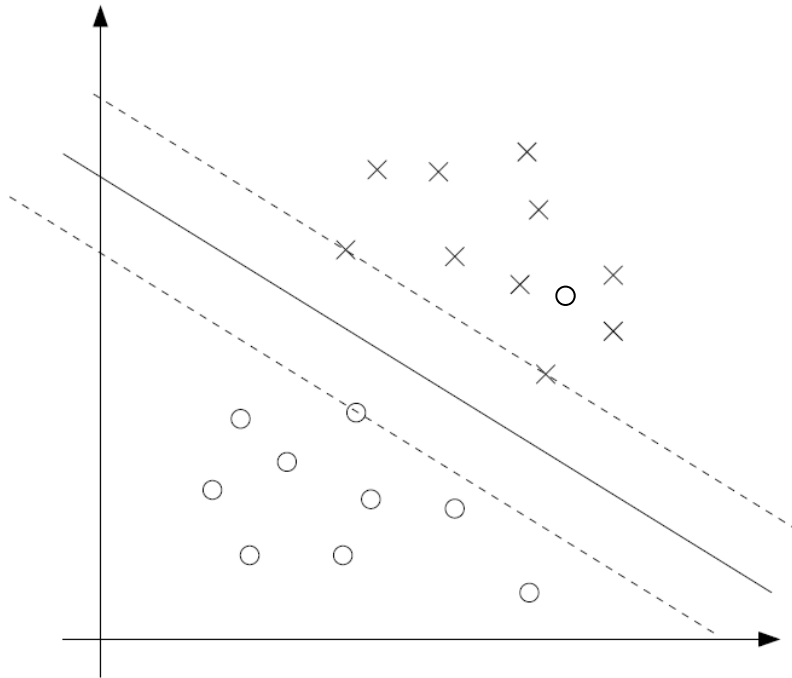
$$\begin{aligned}w^{*\top}x + b^* &= \left( \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \right)^\top x + b^* \\ &= \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b^*\end{aligned}$$

- We only need to calculate the inner product of  $x$  with the supporting vectors



# Non-Separable Cases

- The derivation of the SVM as presented so far assumes that the data is linearly separable.
- More practical cases are linearly non-separable.



# Dealing with Non-Separable Cases

- Add slack variables  $\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \leftarrow \text{L1 regularization}$   
s.t.  $y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i, i = 1, \dots, m$   
 $\xi_i \geq 0, i = 1, \dots, m$

- Lagrangian

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^\top w + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i [y^{(i)}(x^\top w + b) - 1 + \xi_i] - \sum_{i=1}^m r_i \xi_i$$

- Dual problem

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)\top} x^{(j)}$$

s.t.  $0 \leq \alpha_i \leq C, i = 1, \dots, m$  Surprisingly, this is the only change

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

Efficiently solved by SMO algorithm

# SVM Hinge Loss vs. LR Loss

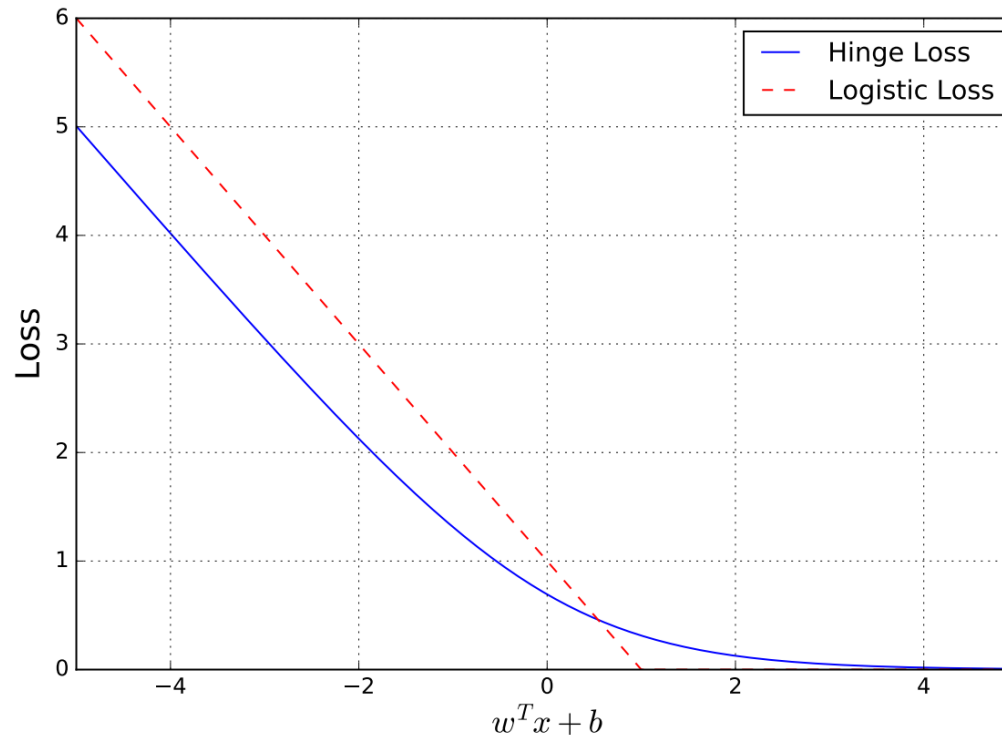
- SVM Hinge loss

$$\frac{1}{2}\|w\|^2 + C \sum_{i=1}^m \max(0, 1 - y_i(w^\top x_i + b))$$

- LR log loss

$$-y_i \log \sigma(w^\top x_i + b) - (1 - y_i) \log(1 - \sigma(w^\top x_i + b))$$

- If  $y = 1$



# Coordinate Ascent (Descent)

- For the optimization problem

$$\max_{\alpha} W(\alpha_1, \alpha_2, \dots, \alpha_m)$$

- Coordinate ascent algorithm

Loop until convergence: {

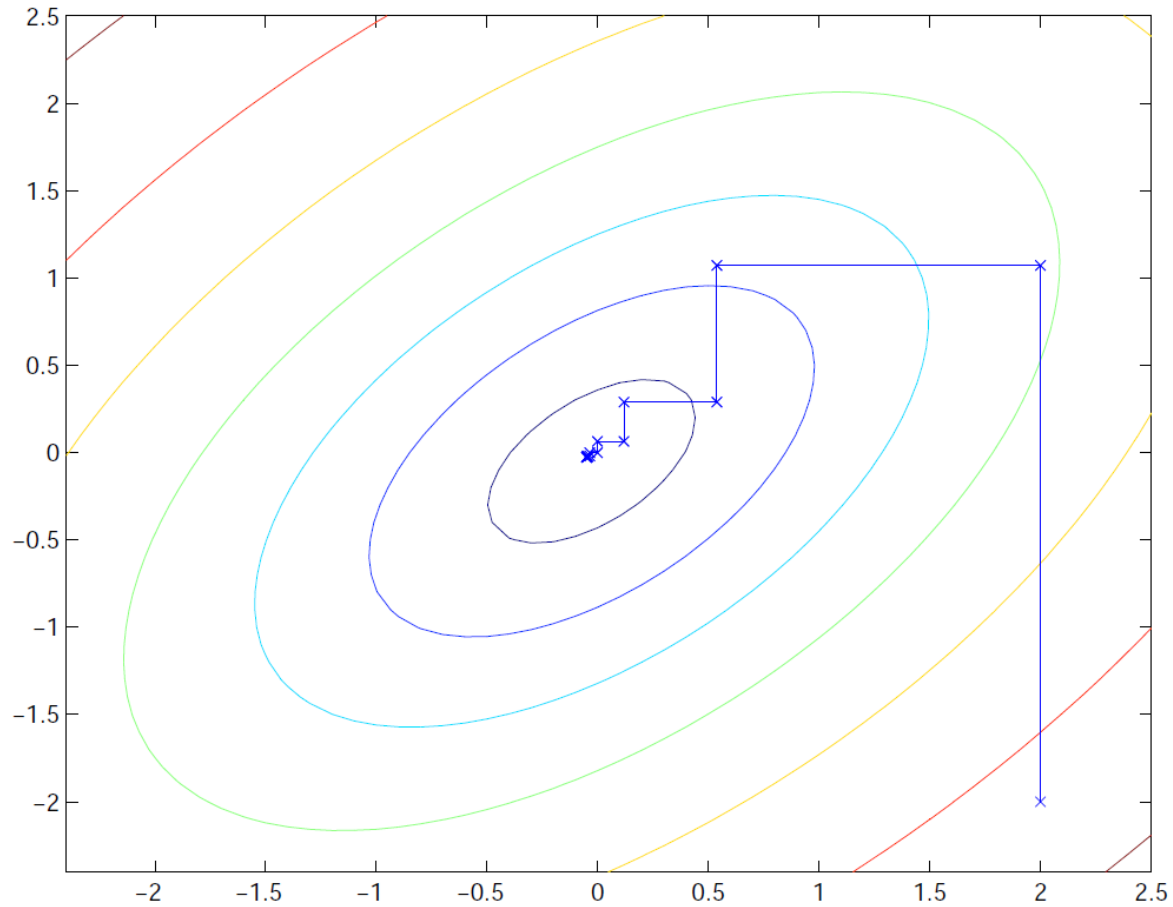
For  $i = 1, \dots, m$  {

$$\alpha_i := \arg \max_{\hat{\alpha}_i} W(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_m)$$

}

}

# Coordinate Ascent (Descent)



A two-dimensional coordinate ascent example

# SMO Algorithm

- SMO: sequential minimal optimization
- SVM optimization problem

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)\top} x^{(j)} \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0 \end{aligned}$$

- Cannot directly apply coordinate ascent algorithm because

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0 \Rightarrow \alpha_i y^{(i)} = - \sum_{j \neq i} \alpha_j y^{(j)}$$



# SMO Algorithm

- Update two variable each time

Loop until convergence {

1. Select some pair  $\alpha_i$  and  $\alpha_j$  to update next
2. Re-optimize  $W(\alpha)$  w.r.t.  $\alpha_i$  and  $\alpha_j$

}

- Convergence test: whether the change of  $W(\alpha)$  is smaller than a predefined value (e.g. 0.01)
- Key advantage of SMO algorithm is the update of  $\alpha_i$  and  $\alpha_j$  (step 2) is efficient

# SMO Algorithm

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)\top} x^{(j)}$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m$$

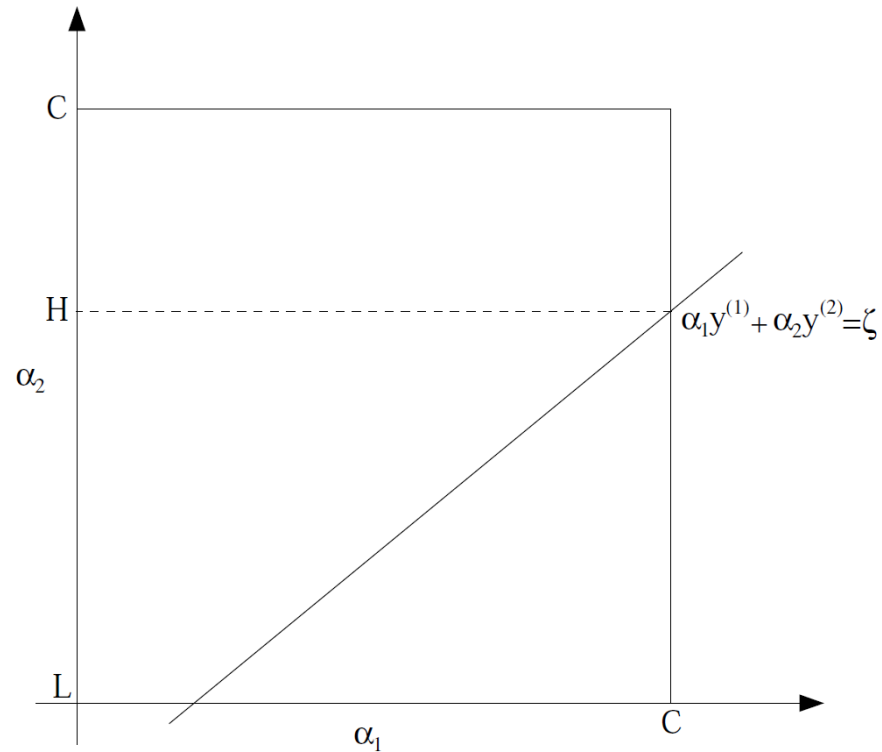
$$\sum_{i=1}^m \alpha_i y^{(i)} = 0$$

- Without loss of generality, hold  $\alpha_3 \dots \alpha_m$  and optimize  $W(\alpha)$  w.r.t.  $\alpha_1$  and  $\alpha_2$

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = - \sum_{i=3}^m \alpha_i y^{(i)} = \zeta$$

$$\Rightarrow \alpha_2 = -\frac{y^{(1)}}{y^{(2)}} \alpha_1 + \frac{\zeta}{y^{(2)}}$$

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}$$



# SMO Algorithm

- With  $\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}$ , the objective is written as

$$W(\alpha_1, \alpha_2, \dots, \alpha_m) = W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \dots, \alpha_m)$$

- Thus the original optimization problem

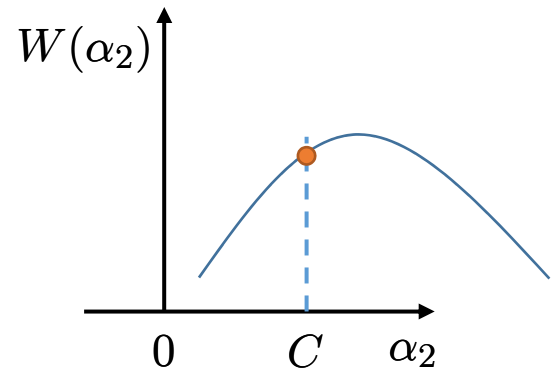
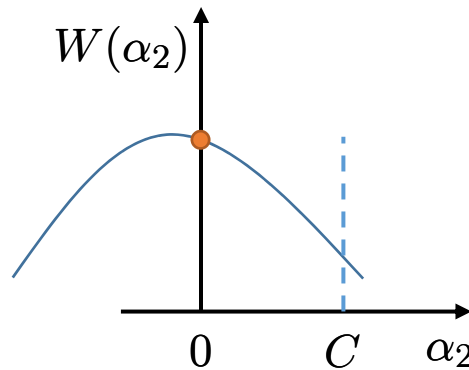
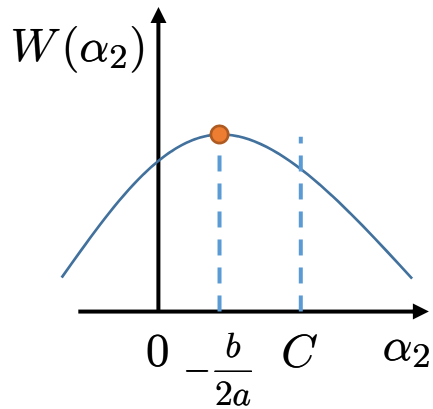
$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)\top} x^{(j)} \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0 \end{aligned}$$

is transformed into a quadratic optimization problem w.r.t.  $\alpha_2$

$$\begin{aligned} \max_{\alpha_2} \quad & W(\alpha_2) = a\alpha_2^2 + b\alpha_2 + c \\ \text{s.t.} \quad & 0 \leq \alpha_2 \leq C \end{aligned}$$

# SMO Algorithm

- Optimizing a quadratic function is much efficient



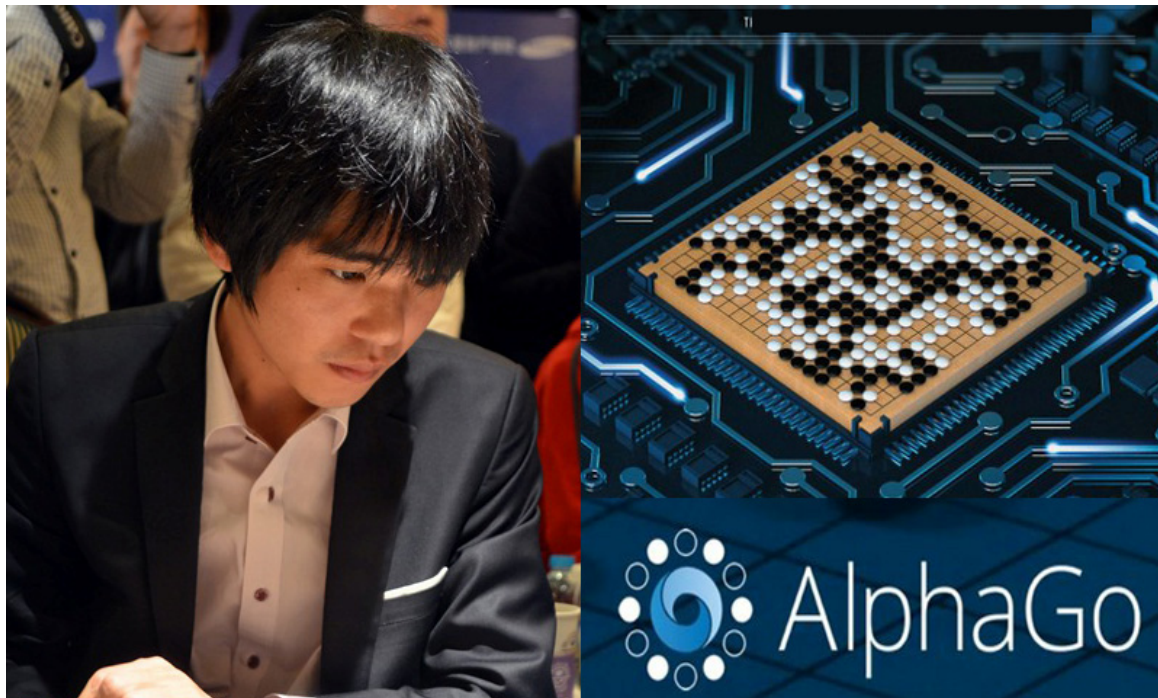
$$\begin{aligned} \max_{\alpha_2} \quad & W(\alpha_2) = a\alpha_2^2 + b\alpha_2 + c \\ \text{s.t.} \quad & 0 \leq \alpha_2 \leq C \end{aligned}$$

# Content of This Lecture

- Support Vector Machines
- Neural Networks

# Breaking News of AI in 2016

- AlphaGo wins Lee Sedol (4-1)



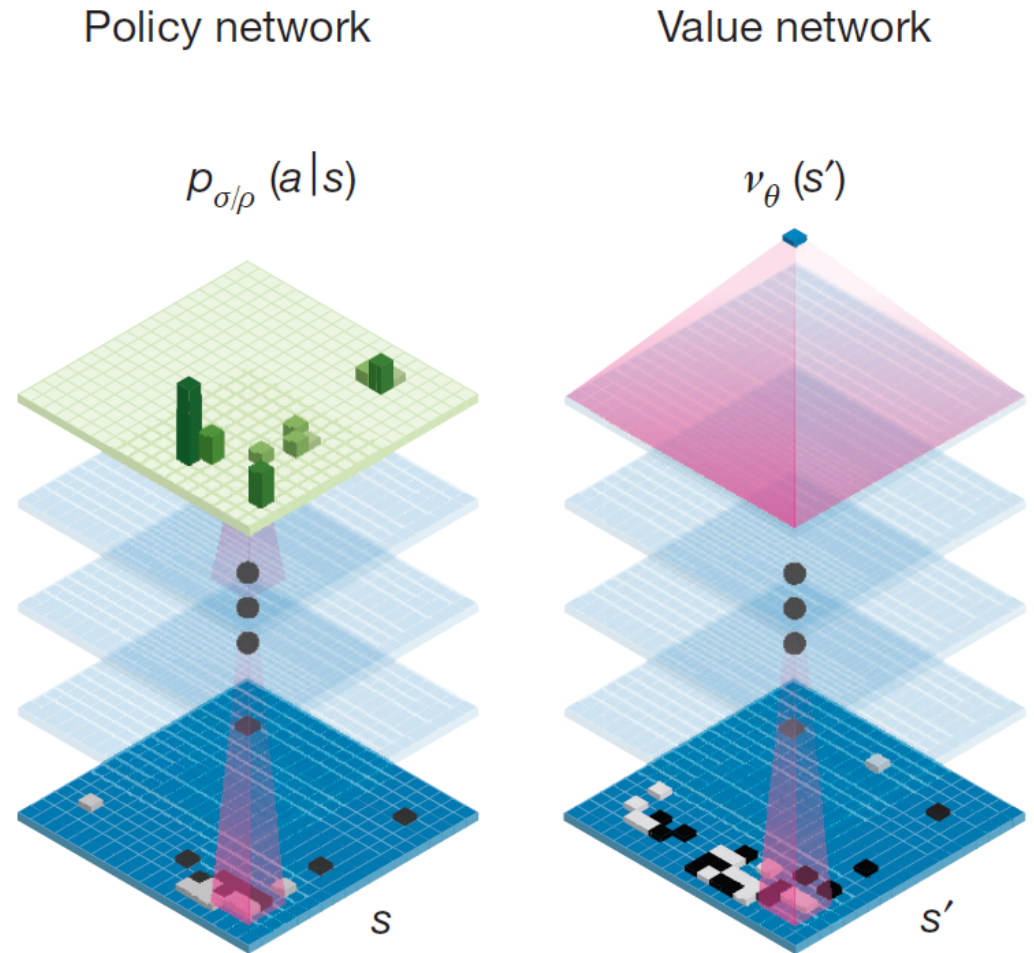
<https://deepmind.com/research/alphago/>

Rank	Name	↑ ↓	Flag	Elo
1	<a href="#">Ke Jie</a>	↑		3628
2	<a href="#">AlphaGo</a>			3598
3	<a href="#">Park Junghwan</a>	↑		3585
4	<a href="#">Tuo Jiaxi</a>	↑		3535
5	<a href="#">Mi Yuting</a>	↑		3534
6	<a href="#">Iyama Yuta</a>	↑		3525
7	<a href="#">Shi Yue</a>	↑		3522
8	<a href="#">Lee Sedol</a>	↑		3521
9	<a href="#">Zhou Ruiyang</a>	↑		3517
10	<a href="#">Shin Jinseok</a>	↑		3503
11	<a href="#">Chen Yaoye</a>	↑		3495
12	<a href="#">Lian Xiao</a>	↑		3493
13	<a href="#">Tan Xiao</a>	↑		3489
14	<a href="#">Kim Iiseok</a>	↑		3489
15	<a href="#">Choi Cheolhan</a>	↑		3482
16	<a href="#">Park Yeonghun</a>	↑		3482
17	<a href="#">Gu Zihao</a>	↑		3468
18	<a href="#">Fan Yunruo</a>	↑		3468
19	<a href="#">Huang Yunsong</a>	↑		3467
20	<a href="#">Li Qincheng</a>	↑		3465
21	<a href="#">Tang Weixing</a>	↑		3461
22	<a href="#">Lee Donghoon</a>	↑		3460
23	<a href="#">Lee Yeongkyu</a>	↑		3459
24	<a href="#">Fan Tingyu</a>	↑		3459
25	<a href="#">Tong Mengcheng</a>	↑		3447
26	<a href="#">Kang Dongyun</a>	↑		3442
27	<a href="#">Wang Xi</a>	↑		3439
28	<a href="#">Weon Seongiin</a>	↑		3439
29	<a href="#">Yang Dingxin</a>	↑		3439
30	<a href="#">Gu Li</a>	↑		3436

<https://www.goratings.org/>

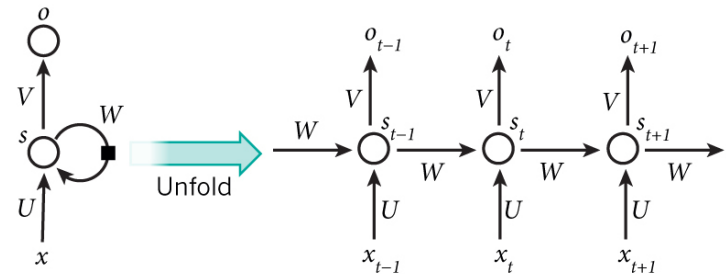
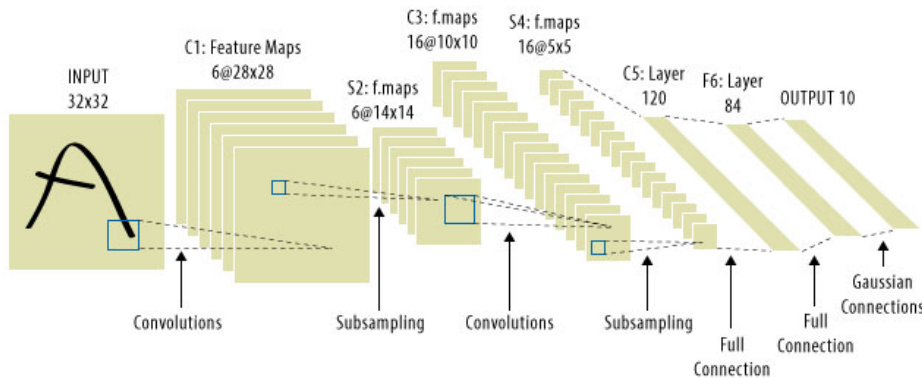
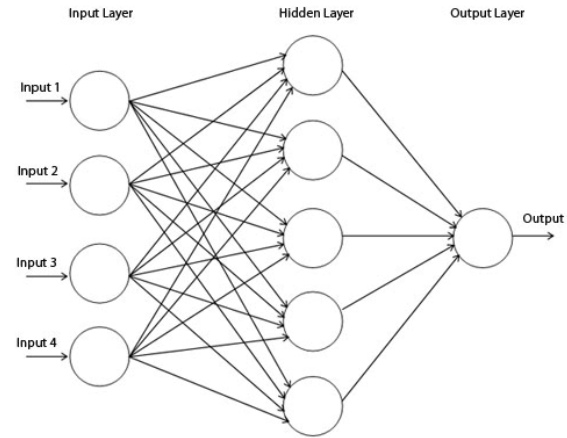
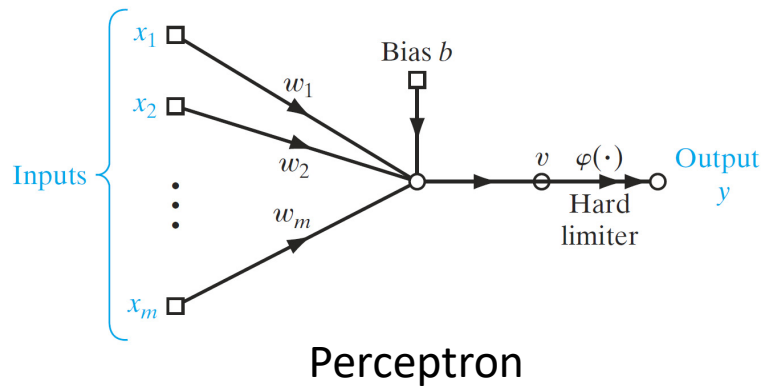
# Machine Learning in AlphaGo

- Policy Network
  - Supervised Learning
    - Predict what is the best next human move
  - Reinforcement Learning
    - Learning to select the next move to maximize the winning rate
- Value Network
  - Expectation of winning given the board state
- Implemented by (deep) neural networks



# Neural Networks

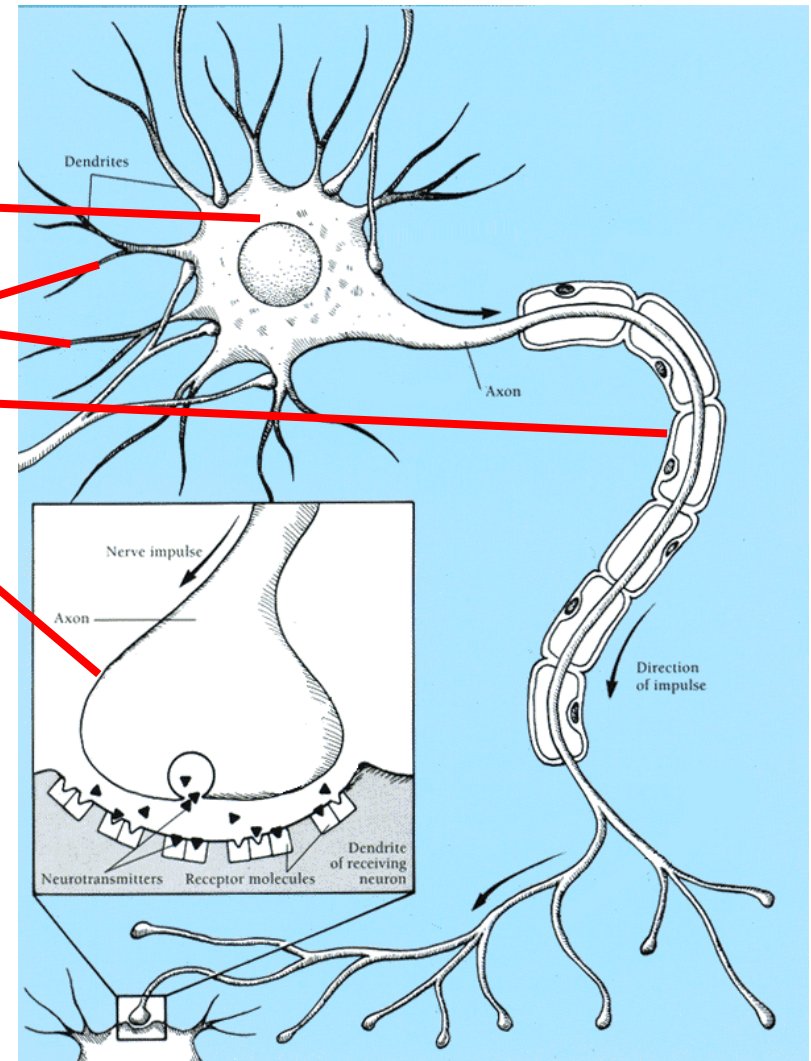
- Neural networks are the basis of deep learning





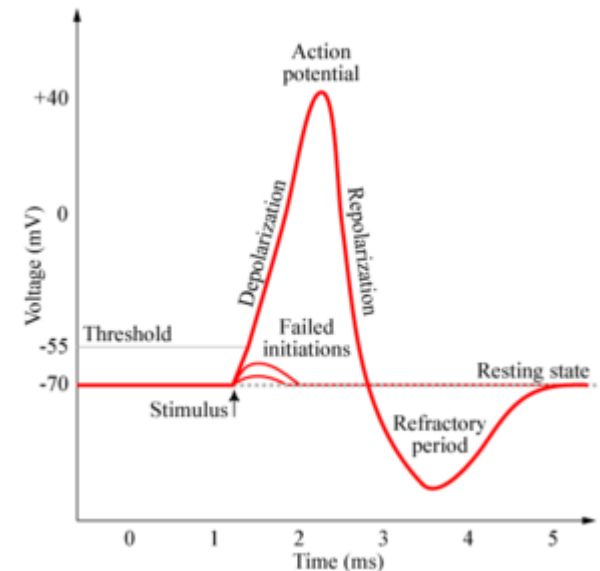
# Real Neurons

- Cell structures
  - Cell body
  - Dendrites
  - Axon
  - Synaptic terminals



# Neural Communication

- Electrical potential across cell membrane exhibits spikes called action potentials.
- Spike originates in cell body, travels down axon, and causes synaptic terminals to release neurotransmitters.
- Chemical diffuses across synapse to dendrites of other neurons.
- Neurotransmitters can be excitatory or inhibitory.
- If net input of neurotransmitters to a neuron from other neurons is excitatory and exceeds some threshold, it fires an action potential.



# Real Neural Learning

- Synapses change size and strength with experience.
- **Hebbian learning**: When two connected neurons are firing at the same time, the strength of the synapse between them increases.
- “Neurons that fire together, wire together.”
- These motivate the research of artificial neural nets

# Brief History of Artificial Neural Nets

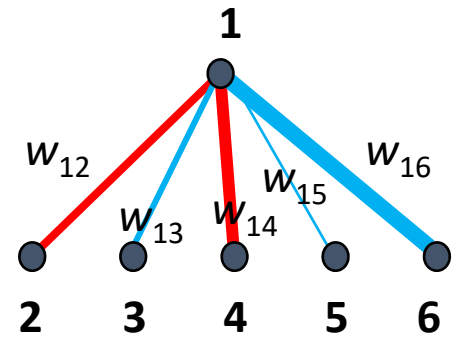
- The First wave
  - 1943 McCulloch and Pitts proposed the [McCulloch-Pitts neuron model](#)
  - 1958 Rosenblatt introduced the simple single layer networks now called [Perceptrons](#).
  - 1969 Minsky and Papert's book Perceptrons demonstrated the limitation of single layer perceptrons, and almost the whole field went into hibernation.
- The Second wave
  - 1986 The [Back-Propagation learning algorithm](#) for Multi-Layer Perceptrons was rediscovered and the whole field took off again.
- The Third wave
  - 2006 [Deep \(neural networks\) Learning](#) gains popularity and
  - 2012 made significant break-through in many applications.

# Artificial Neuron Model

- Model network as a graph with cells as nodes and synaptic connections as weighted edges from node  $i$  to node  $j$ ,  $w_{ji}$

- Model net input to cell as

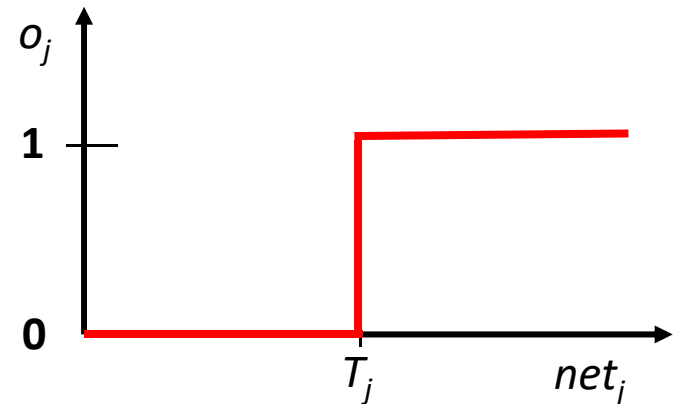
$$\text{net}_j = \sum_i w_{ji} o_i$$



- Cell output is

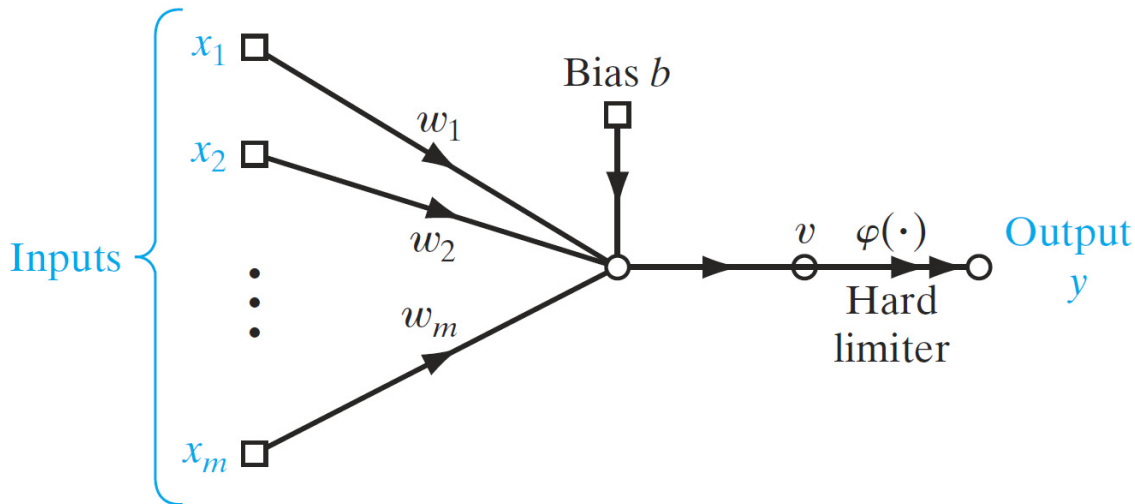
$$o_j = \begin{cases} 0 & \text{if } \text{net}_j < T_j \\ 1 & \text{if } \text{net}_j \geq T_j \end{cases}$$

( $T_j$  is threshold for unit  $j$ )



# Perceptron Model

- Rosenblatt's single layer perceptron [1958]



- Rosenblatt [1958] further proposed the *perceptron* as the first model for learning with a teacher (i.e., supervised learning)
- Focused on how to find appropriate weights  $w_m$  for two-class classification task

- Prediction

$$\hat{y} = \varphi\left(\sum_{i=1}^m w_i x_i + b\right)$$

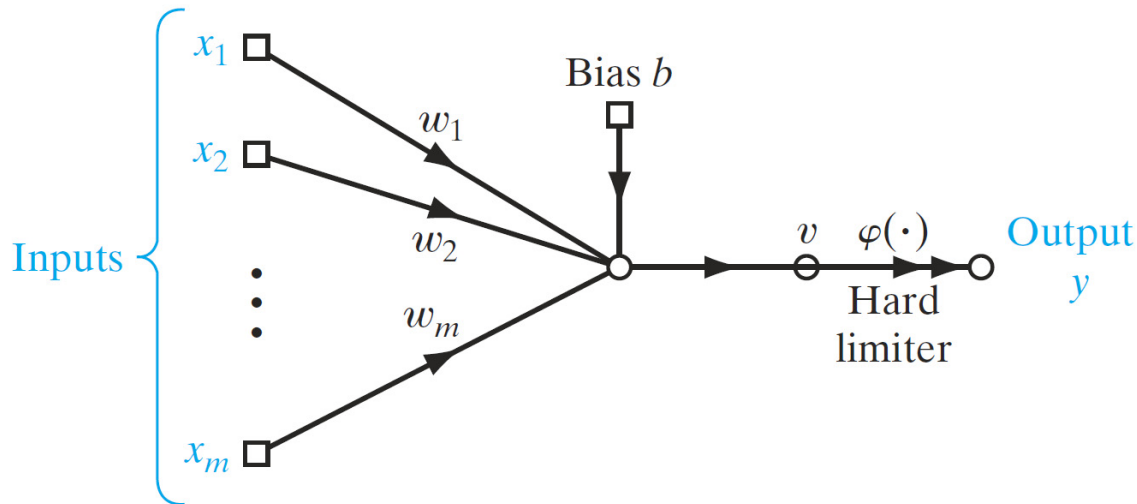
- Activation function

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- $y = 1$ : class one
- $y = -1$ : class two

# Training Perceptron

- Rosenblatt's single layer perceptron [1958]



- Training

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

$$b = b + \eta(y - \hat{y})$$

- Equivalent to rules:

- If output is correct, do nothing
- If output is high, lower weights on positive inputs
- If output is low, increase weights on active inputs

- Prediction

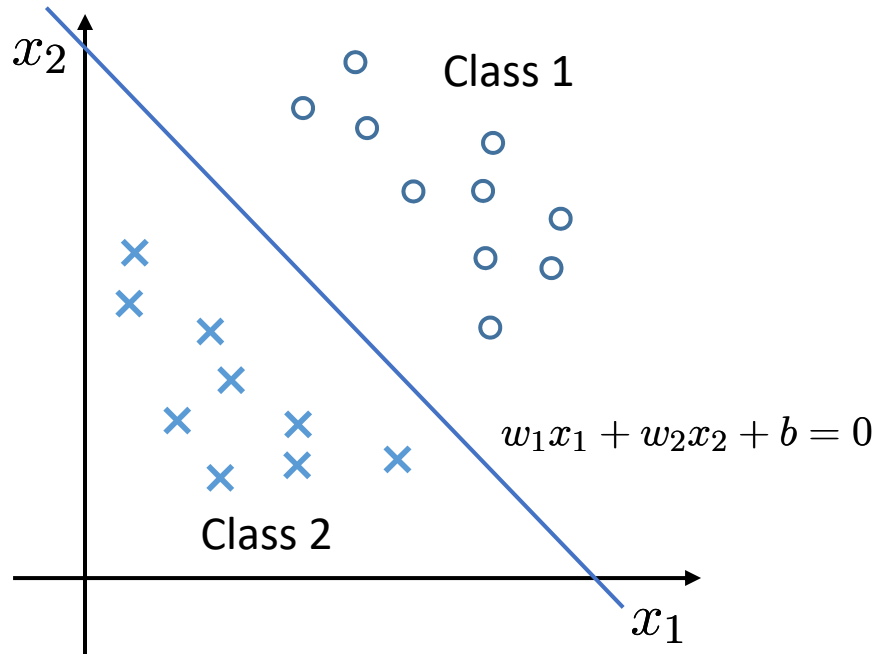
$$\hat{y} = \varphi\left(\sum_{i=1}^m w_i x_i + b\right)$$

- Activation function

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

# Properties of Perceptron

- Rosenblatt's single layer perceptron [1958]



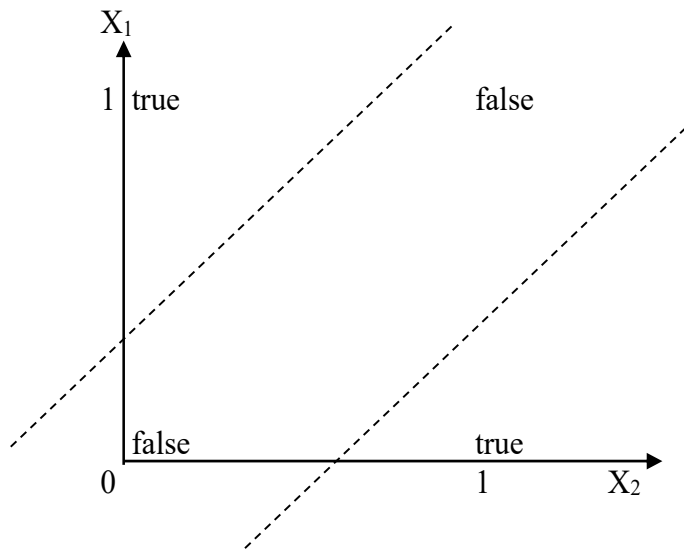
- Rosenblatt proved the convergence of a learning algorithm if two classes said to be linearly separable (i.e., patterns that lie on opposite sides of a hyperplane)
- Many people hoped that such a machine could be the basis for artificial intelligence



# Properties of Perceptron

- The XOR problem

Input $x$		Output $y$
$X_1$	$X_2$	$X_1 \text{ XOR } X_2$
0	0	0
0	1	1
1	0	1
1	1	0

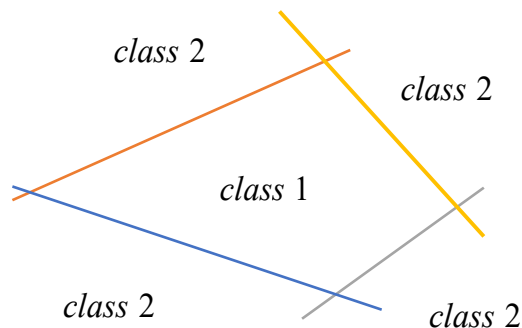
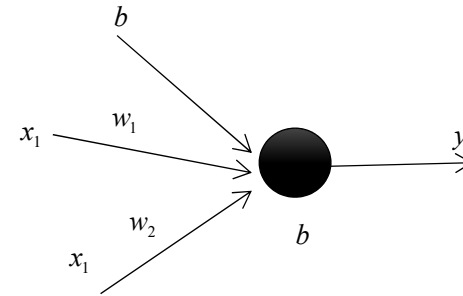
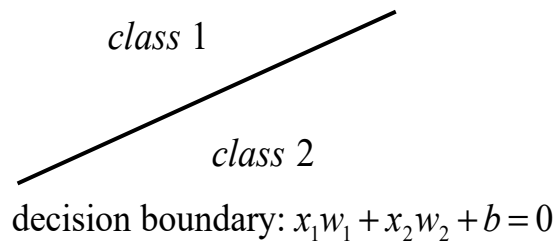


XOR is non linearly separable: These two classes (true and false) cannot be separated using a line.

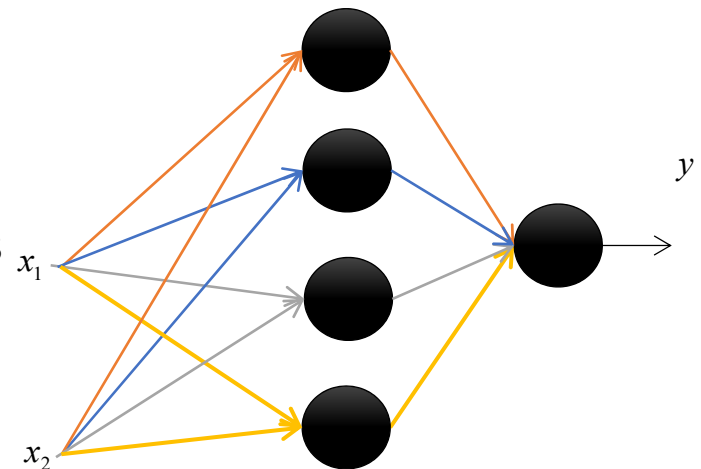
- However, Minsky and Papert [1969] showed that some rather elementary computations, such as *XOR* problem, could not be done by Rosenblatt's one-layer perceptron
- However Rosenblatt believed the limitations could be overcome if more layers of units to be added, but no learning algorithm known to obtain the weights yet
- Due to the lack of learning algorithms people left the neural network paradigm for almost 20 years

# Hidden Layers and Backpropagation (1986~)

- Adding hidden layer(s) (internal presentation) allows to learn a mapping that is not constrained by **linearly separable**



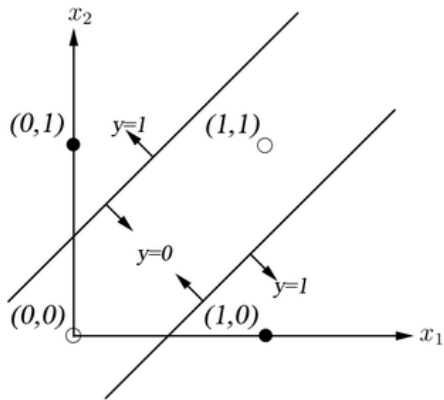
Each hidden node realizes one of the lines bounding the convex region



# Hidden Layers and Backpropagation (1986~)

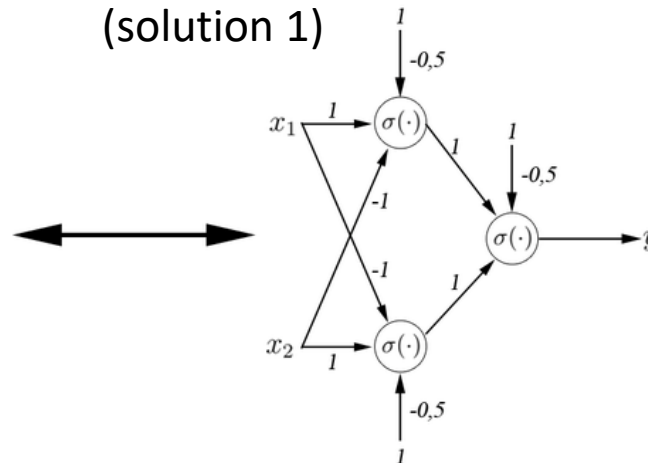
- But the solution is quite often not unique

Input $x$		Output $y$
$X_1$	$X_2$	$X_1 \text{ XOR } X_2$
0	0	0
0	1	1
1	0	1
1	1	0



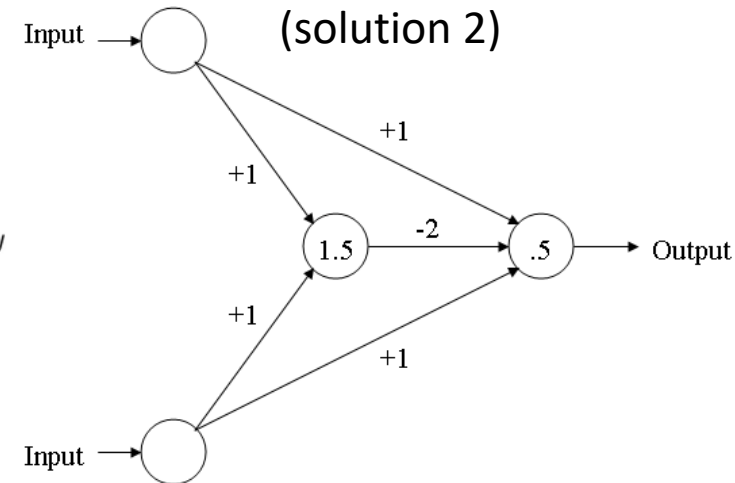
Two lines are necessary to divide the sample space accordingly

(solution 1)



Sign activation function

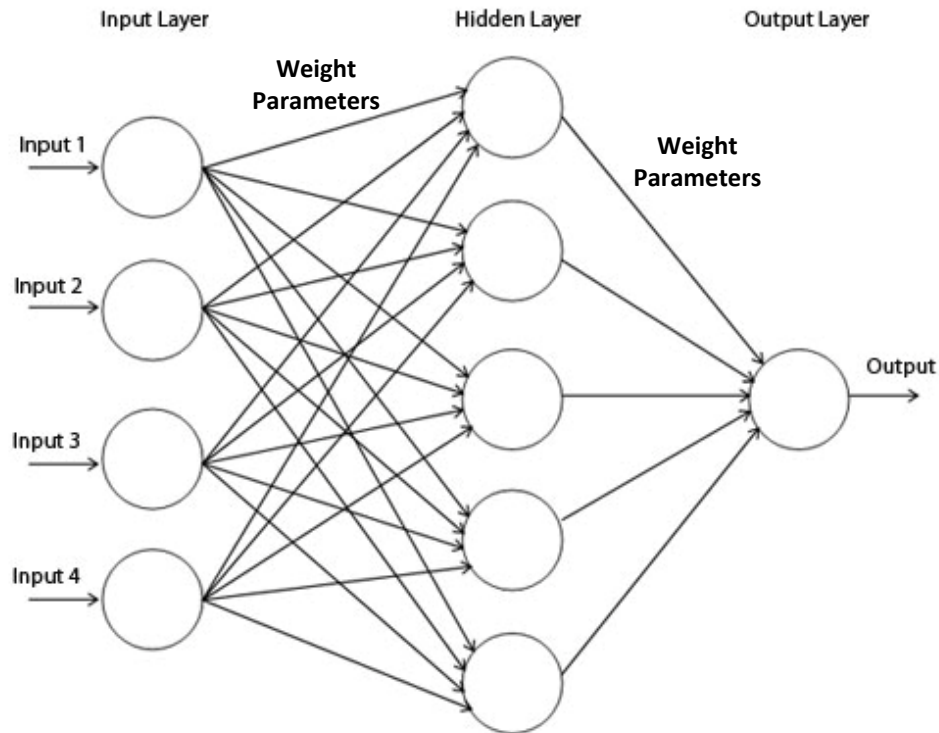
(solution 2)



The number in the circle is a threshold

# Hidden Layers and Backpropagation (1986~)

- **Feedforward**: messages move forward from the input nodes, through the hidden nodes (if any), and to the output nodes. There are no cycles or loops in the network

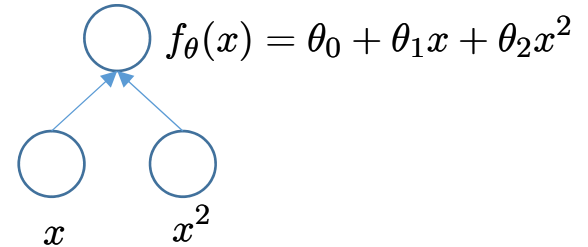


Two-layer feedforward neural network

# Single / Multiple Layers of Calculation

- Single layer function

$$f_{\theta}(x) = \sigma(\theta_0 + \theta_1 x + \theta_2 x^2)$$

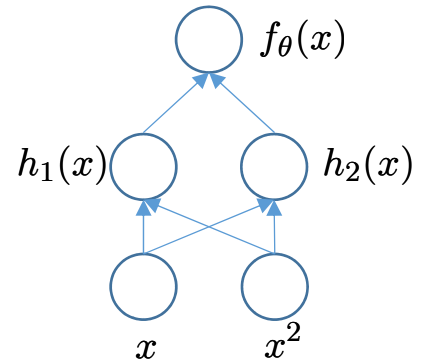


- Multiple layer function

$$h_1(x) = \tanh(\theta_0 + \theta_1 x + \theta_2 x^2)$$

$$h_2(x) = \tanh(\theta_3 + \theta_4 x + \theta_5 x^2)$$

$$f_{\theta}(x) = f_{\theta}(h_1(x), h_2(x)) = \sigma(\theta_6 + \theta_7 h_1 + \theta_8 h_2)$$



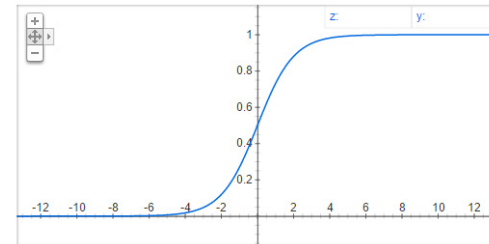
- With non-linear activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

# Non-linear Activation Functions

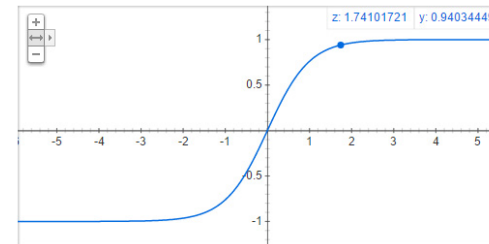
- Sigmoid

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



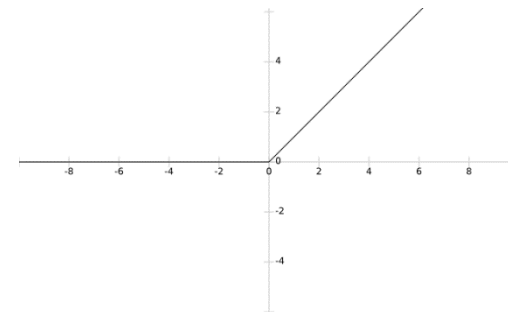
- Tanh

$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$



- Rectified Linear Unit (ReLU)

$$\text{ReLU}(z) = \max(0, z)$$

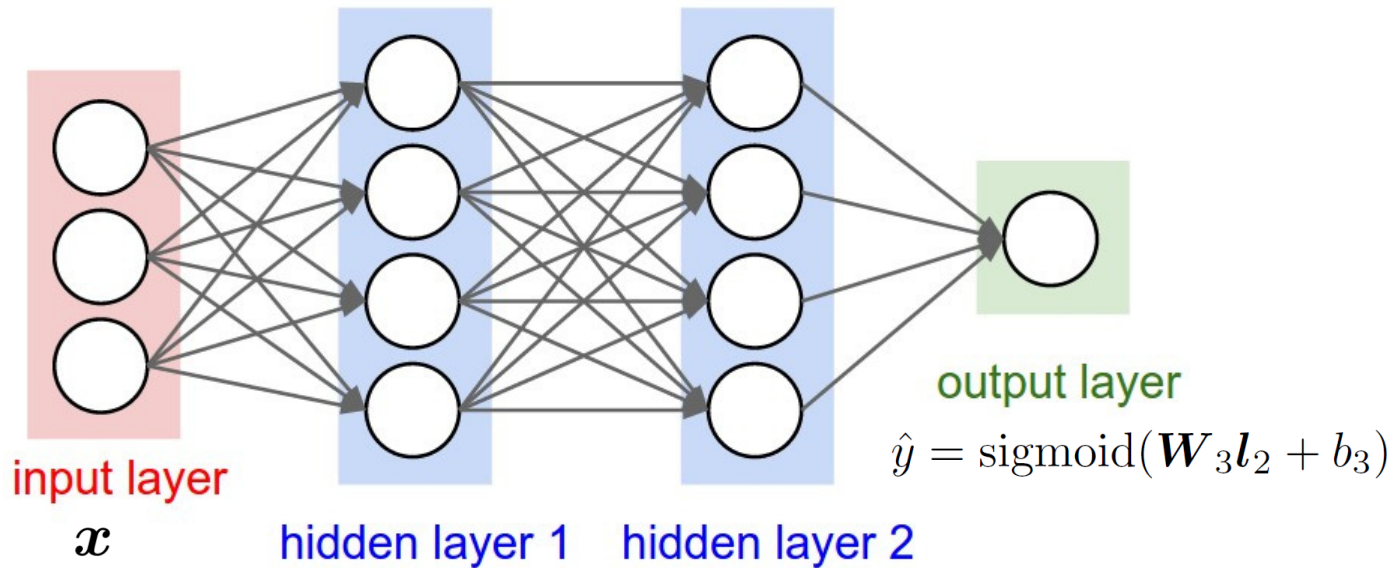


# Universal Approximation Theorem

- A feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions
  - on compact subsets of  $\mathbb{R}^n$
  - under mild assumptions on the activation function
    - Such as Sigmoid, Tanh and ReLU

# Universal Approximation

- Multi-layer perceptron approximate any continuous functions on compact subset of  $\mathbb{R}^n$



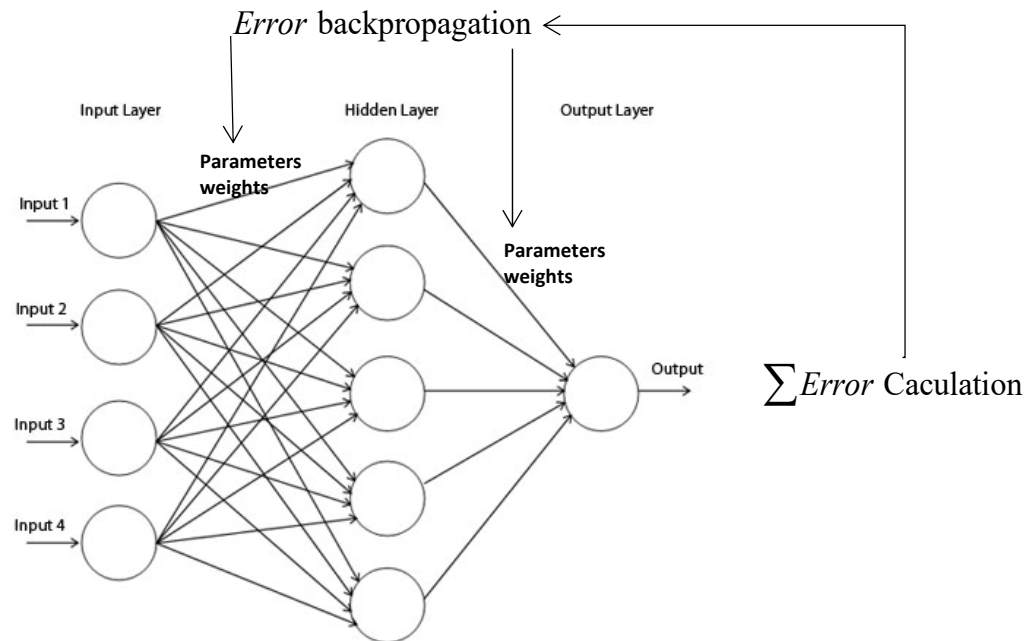
$$\mathbf{l}_1 = \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \quad \mathbf{l}_2 = \tanh(\mathbf{W}_2 \mathbf{l}_1 + \mathbf{b}_2)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



# Hidden Layers and Backpropagation (1986~)

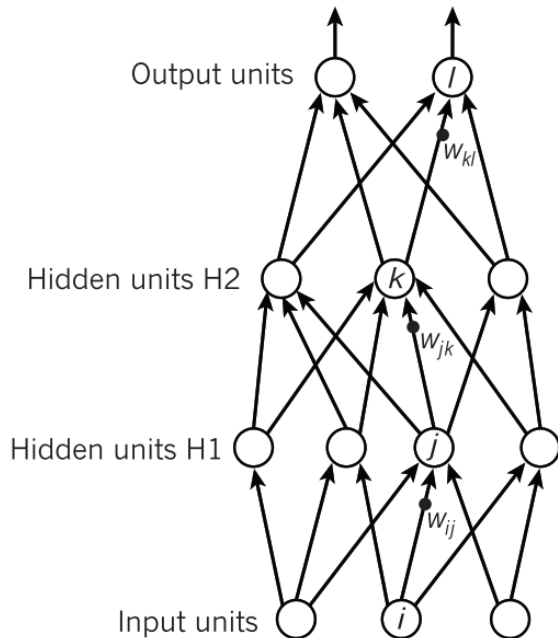
- One of the efficient algorithms for multi-layer neural networks is *the Backpropagation algorithm*
- It was re-introduced in 1986 and Neural Networks regained the popularity



Note: *backpropagation* appears to be found by Werbos [1974]; and then independently rediscovered around 1985 by Rumelhart, Hinton, and Williams [1986] and by Parker [1985]

# Learning NN by Back-Propagation

Compare outputs with correct answer to get error



$$y_l = f(z_l)$$

$$z_l = \sum_{k \in H2} w_{kl} y_k$$

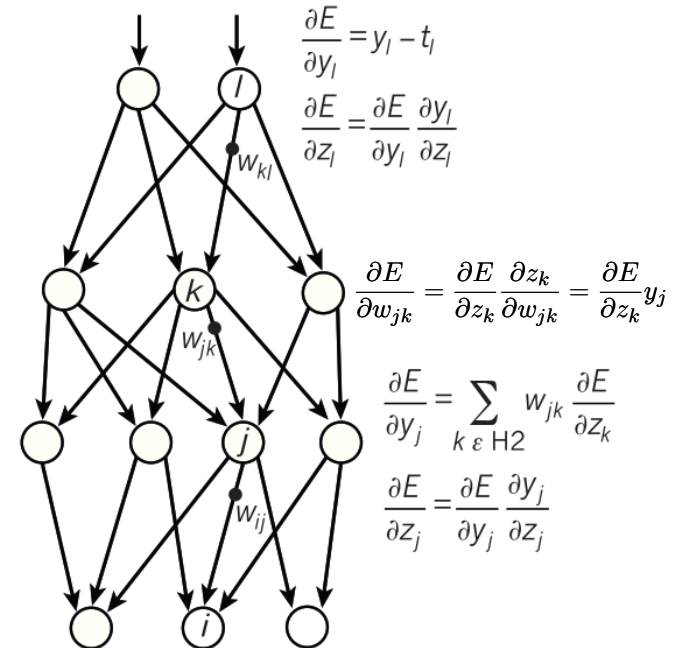
$$y_k = f(z_k)$$

$$z_k = \sum_{j \in H1} w_{jk} y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \in \text{Input}} w_{ij} x_i$$

Compare outputs with correct answer to get error derivatives



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$$

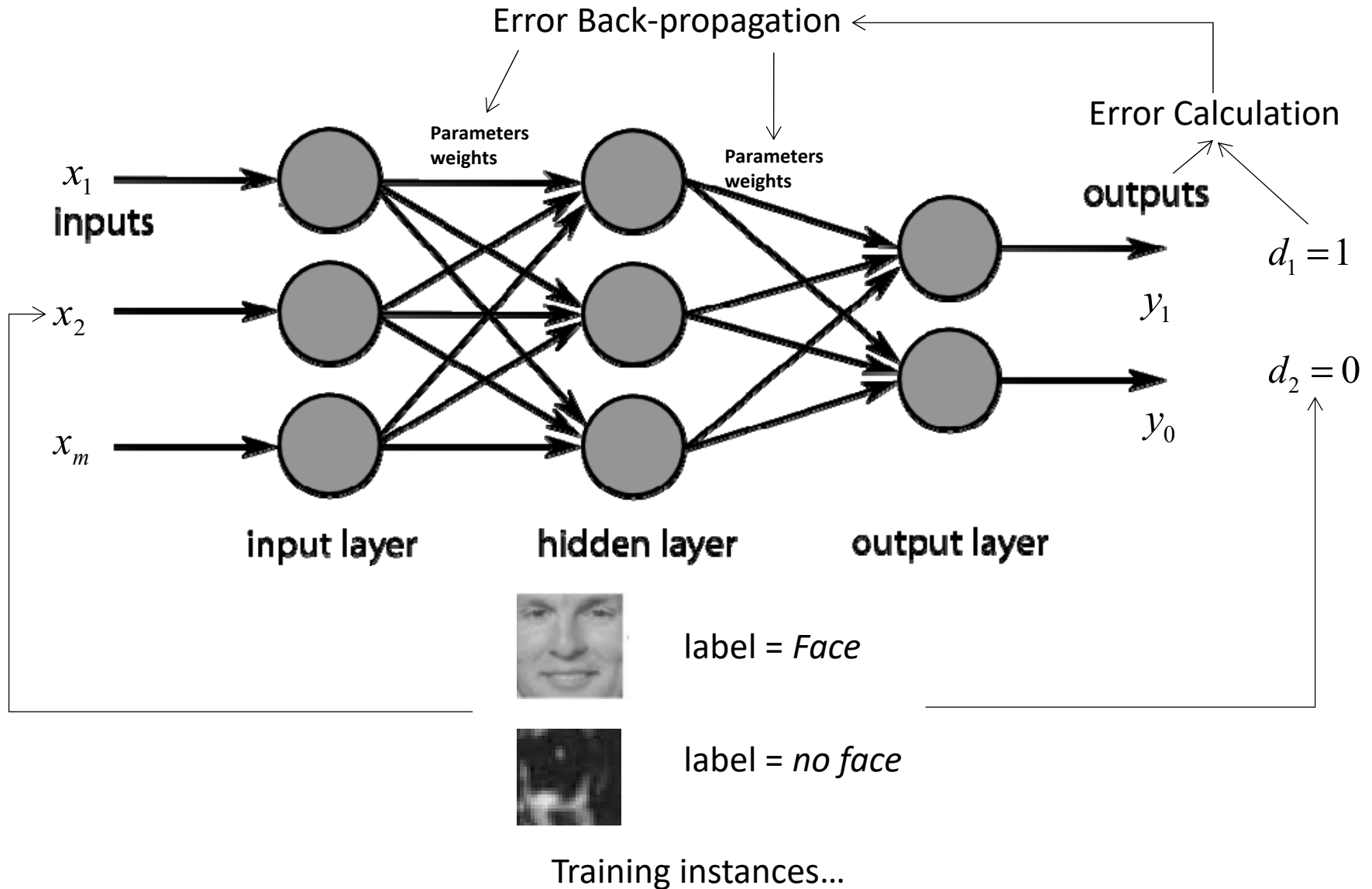
$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} y_j$$

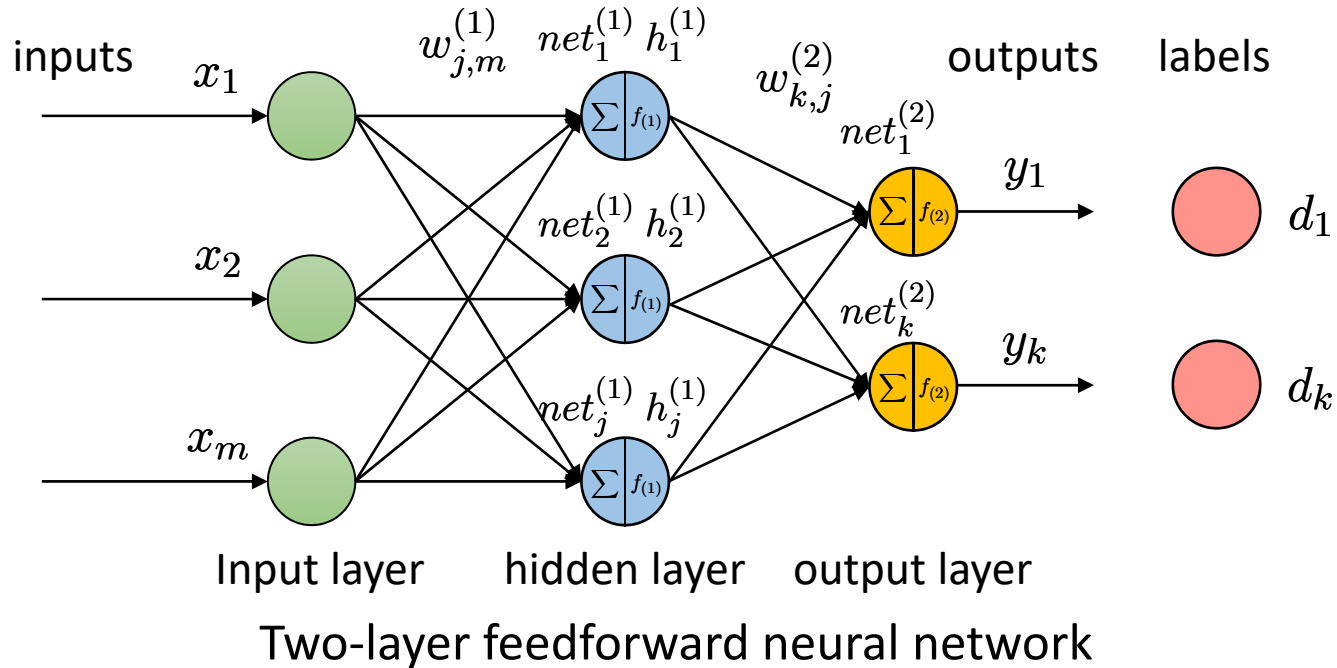
$$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk} \frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$

# Learning NN by Back-Propagation



# Make a Prediction



Feed-forward prediction:

$$h_j^{(1)} = f^{(1)}(net_j^{(1)}) = f^{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f^{(2)}(net_k^{(2)}) = f^{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

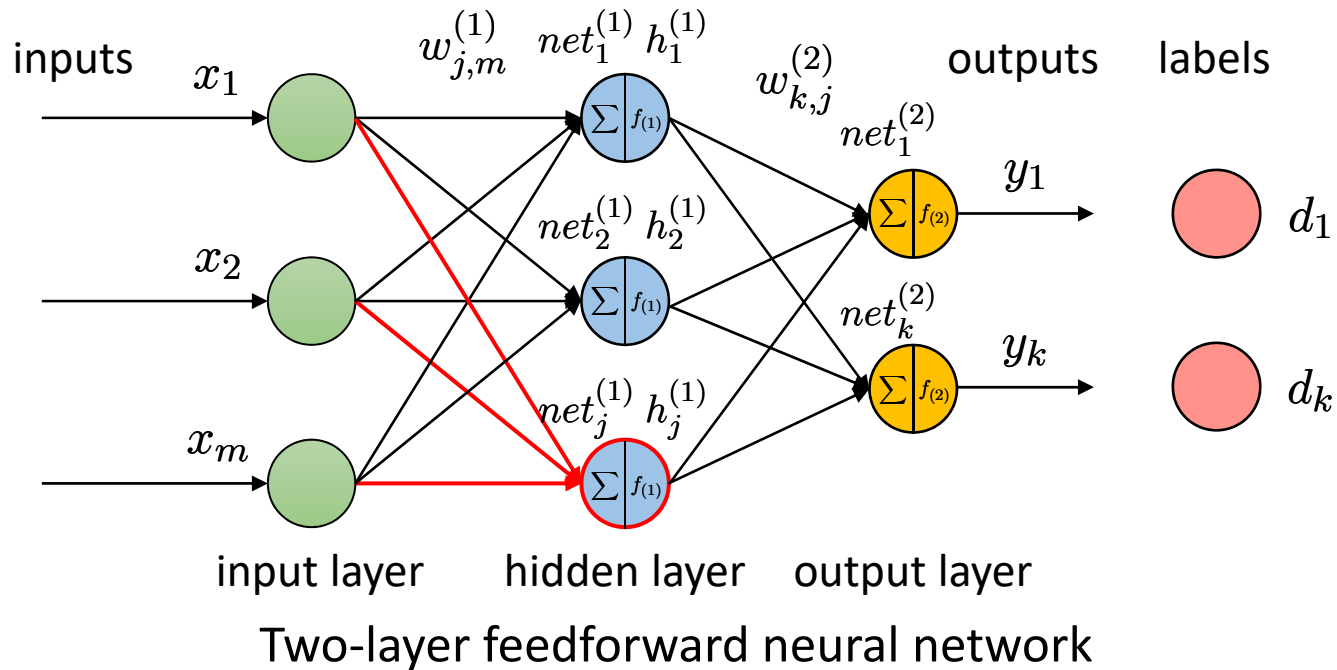
$x = (x_1, \dots, x_m)$   $\longrightarrow$   $h_j^{(1)}$   $\longrightarrow$   $y_k$

where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

# Make a Prediction



Feed-forward prediction:

$$h_j^{(1)} = f^{(1)}(net_j^{(1)}) = f^{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f^{(2)}(net_k^{(2)}) = f^{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

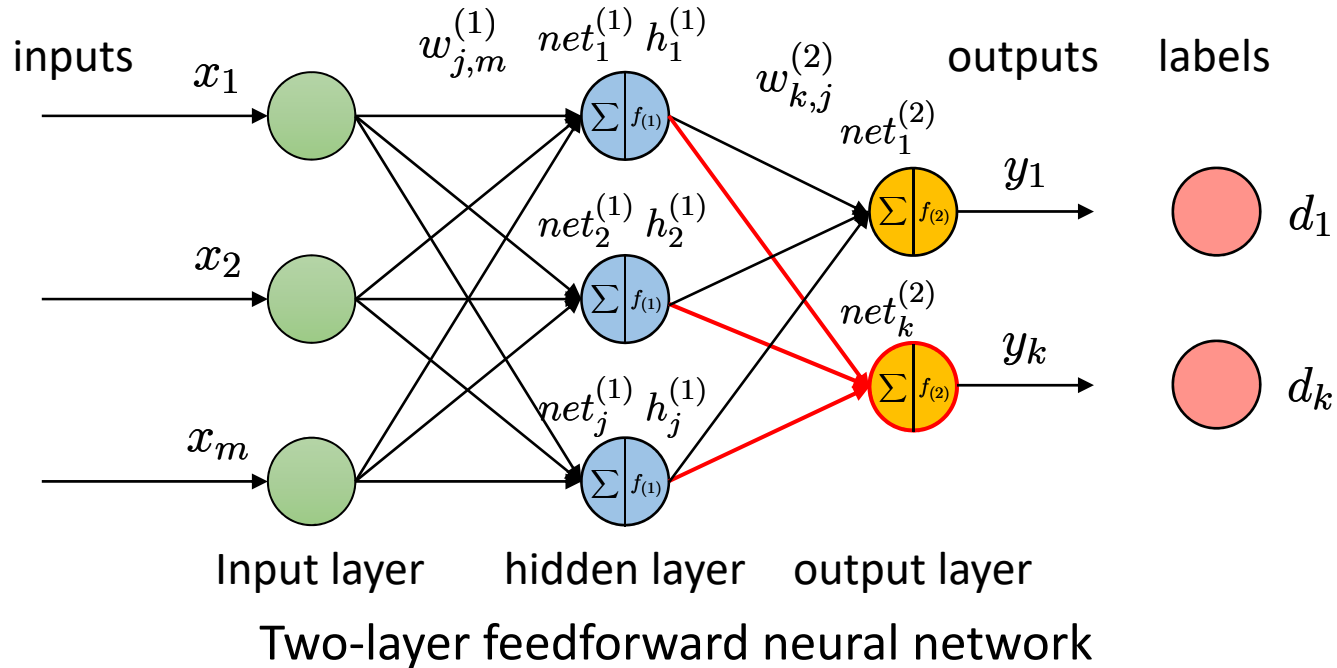
$x = (x_1, \dots, x_m)$   $\longrightarrow$   $h_j^{(1)}$   $\longrightarrow$   $y_k$

where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

# Make a Prediction



Feed-forward prediction:

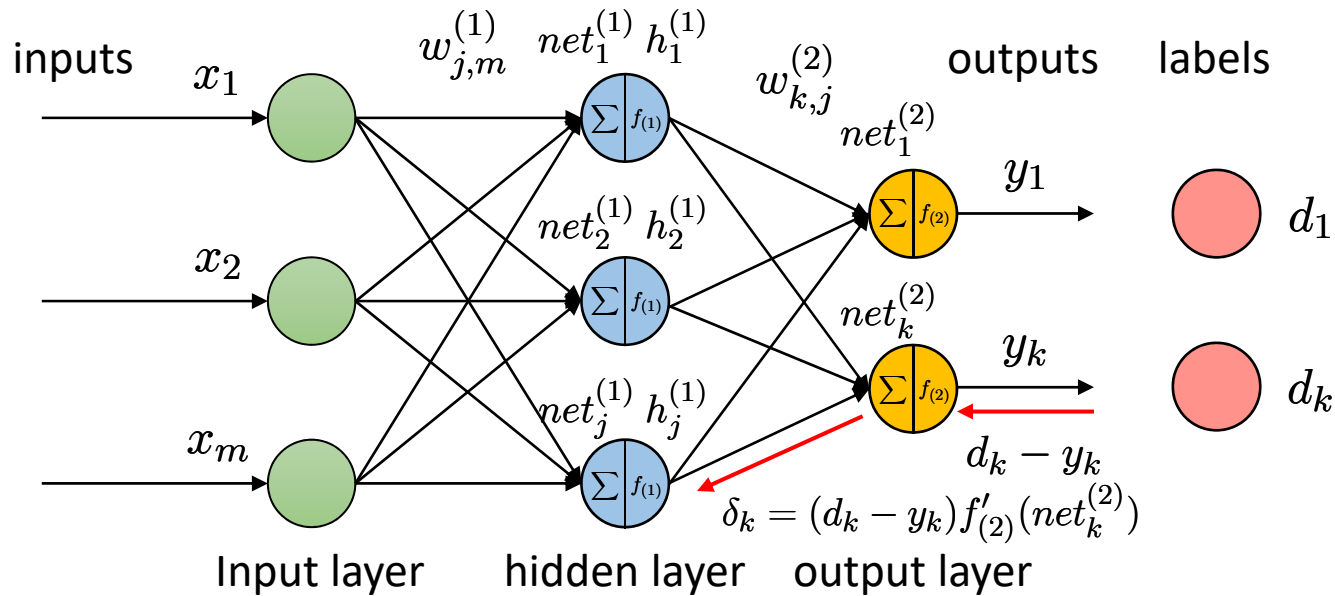
$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \quad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

$x = (x_1, \dots, x_m)$   $\longrightarrow$   $h_j^{(1)}$   $\longrightarrow$   $y_k$

where

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \quad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

# When Backprop/Learn Parameters



Two-layer feedforward neural network

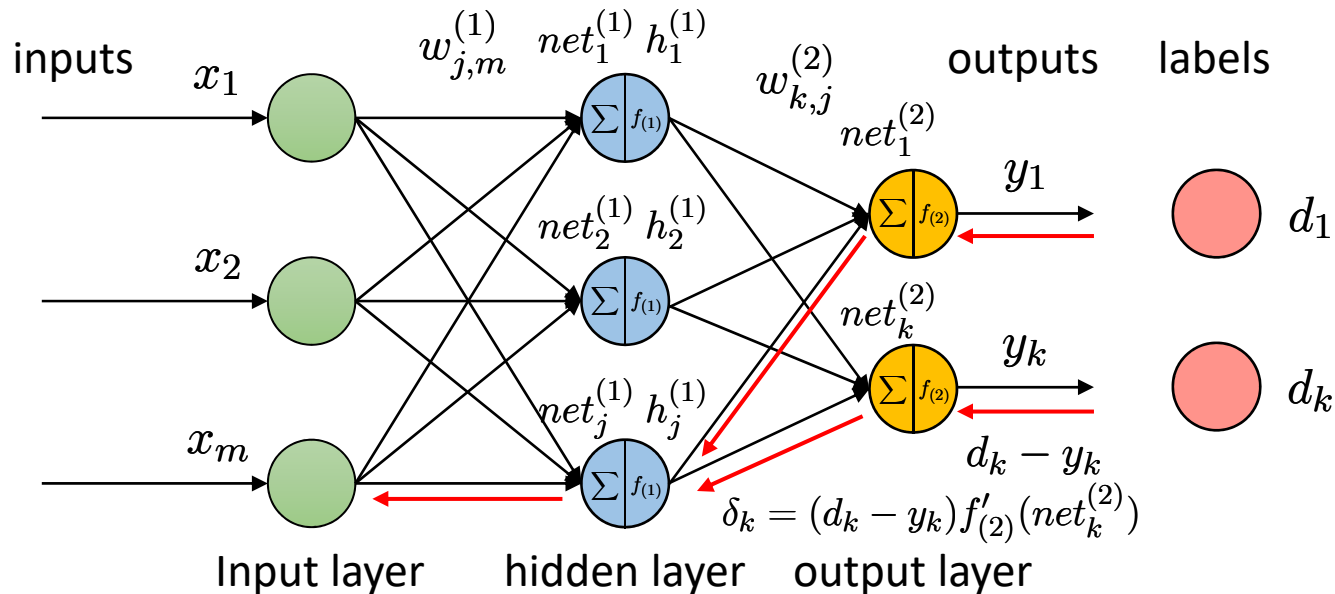
Notations:  $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$        $net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j$

Backprop to learn the parameters

$$\boxed{w_{k,j}^{(2)} = w_{k,j}^{(2)} + \Delta w_{k,j}^{(2)}} \quad \leftarrow \Delta w_{k,j}^{(2)} = \eta \text{Error}_k \text{Output}_j = \eta \delta_k h_j^{(1)} \quad E(W) = \frac{1}{2} \sum_k (y_k - d_k)^2$$

$$\Delta w_{k,j}^{(2)} = -\eta \frac{\partial E(W)}{\partial w_{k,j}^{(2)}} = -\eta (y_k - d_k) \frac{\partial y_k}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{k,j}^{(2)}} = \eta (d_k - y_k) f'_{(2)}(net_k^{(2)}) h_j^{(1)} = \eta \delta_k h_j^{(1)}$$

# When Backprop/Learn Parameters



Notations:  $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$   $net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j$

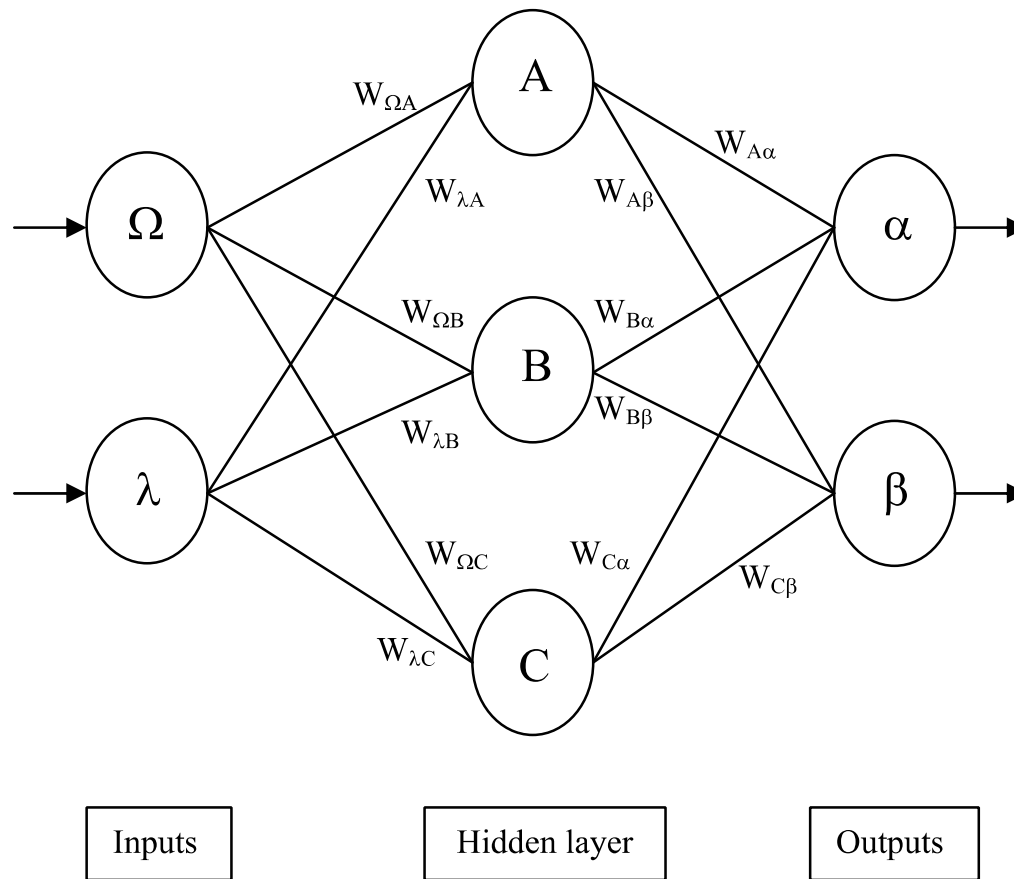
Backprop to learn the parameters

$$\boxed{w_{j,m}^{(1)} = w_{j,m}^{(1)} + \Delta w_{j,m}^{(1)}} \quad \Delta w_{k,j}^{(2)} = \eta \text{Error}_j \text{Output}_m = \eta \delta_j x_m \quad E(W) = \frac{1}{2} \sum_k (y_k - d_k)^2$$

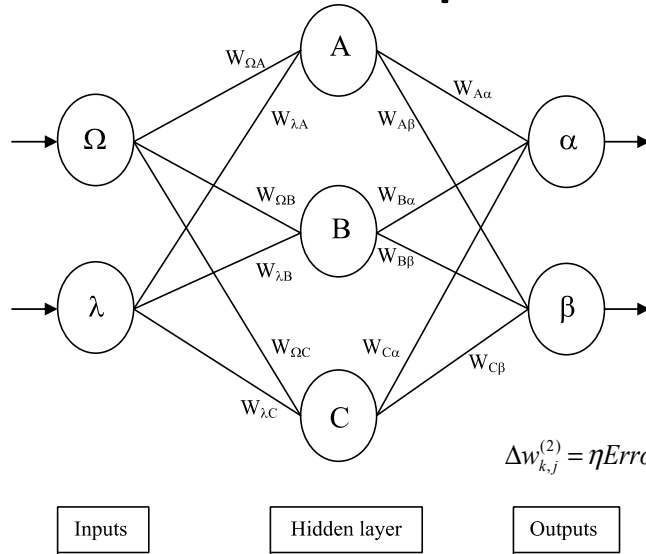
$$\Delta w_{j,m}^{(1)} = -\eta \frac{\partial E(W)}{\partial w_{j,m}^{(1)}} = -\eta \frac{\partial E(W)}{\partial h_j^{(1)}} \frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = \eta \sum_k (d_k - y_k) f'_{(2)}(net_k^{(2)}) w_{k,j}^{(2)} x_m f'_{(1)}(net_j^{(1)}) = \eta \delta_j x_m$$



# An example for Backprop



# An example for Backprop



1. Calculate errors of output neurons

$$\delta_\alpha = \text{out}_\alpha (1 - \text{out}_\alpha) (\text{Target}_\alpha - \text{out}_\alpha)$$

$$\delta_\beta = \text{out}_\beta (1 - \text{out}_\beta) (\text{Target}_\beta - \text{out}_\beta)$$

$$\delta_k = (d_k - y_k) f'_{(2)}(\text{net}_k^{(2)})$$

2. Change output layer weights

$$\begin{aligned} W_{A\alpha}^+ &= W_{A\alpha} + \eta \delta_\alpha \text{out}_A & W_{A\beta}^+ &= W_{A\beta} + \eta \delta_\beta \text{out}_A \\ W_{B\alpha}^+ &= W_{B\alpha} + \eta \delta_\alpha \text{out}_B & W_{B\beta}^+ &= W_{B\beta} + \eta \delta_\beta \text{out}_B \\ W_{C\alpha}^+ &= W_{C\alpha} + \eta \delta_\alpha \text{out}_C & W_{C\beta}^+ &= W_{C\beta} + \eta \delta_\beta \text{out}_C \end{aligned}$$

$$\Delta w_{k,j}^{(2)} = \eta \text{Error}_k \text{Output}_j = \eta \delta_k h_j^{(1)}$$

3. Calculate (back-propagate) hidden layer errors

$$\begin{aligned} \delta_A &= \text{out}_A (1 - \text{out}_A) (\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta}) \\ \delta_B &= \text{out}_B (1 - \text{out}_B) (\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta}) \\ \delta_C &= \text{out}_C (1 - \text{out}_C) (\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta}) \end{aligned}$$

$$\delta_j = f'_{(1)}(\text{net}_j^{(1)}) \sum_k \delta_k w_{k,j}^{(2)}$$

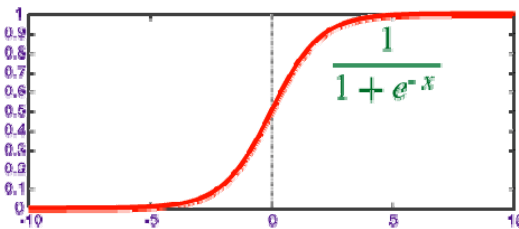
4. Change hidden layer weights

$$\begin{aligned} W_{\lambda A}^+ &= W_{\lambda A} + \eta \delta_A \text{in}_\lambda & W_{\Omega A}^+ &= W_{\Omega A} + \eta \delta_A \text{in}_\Omega \\ W_{\lambda B}^+ &= W_{\lambda B} + \eta \delta_B \text{in}_\lambda & W_{\Omega B}^+ &= W_{\Omega B} + \eta \delta_B \text{in}_\Omega \\ W_{\lambda C}^+ &= W_{\lambda C} + \eta \delta_C \text{in}_\lambda & W_{\Omega C}^+ &= W_{\Omega C} + \eta \delta_C \text{in}_\Omega \end{aligned}$$

$$\Delta w_{j,m}^{(1)} = \eta \text{Error}_j \text{Output}_m = \eta \delta_j x_m$$

Consider sigmoid activation function

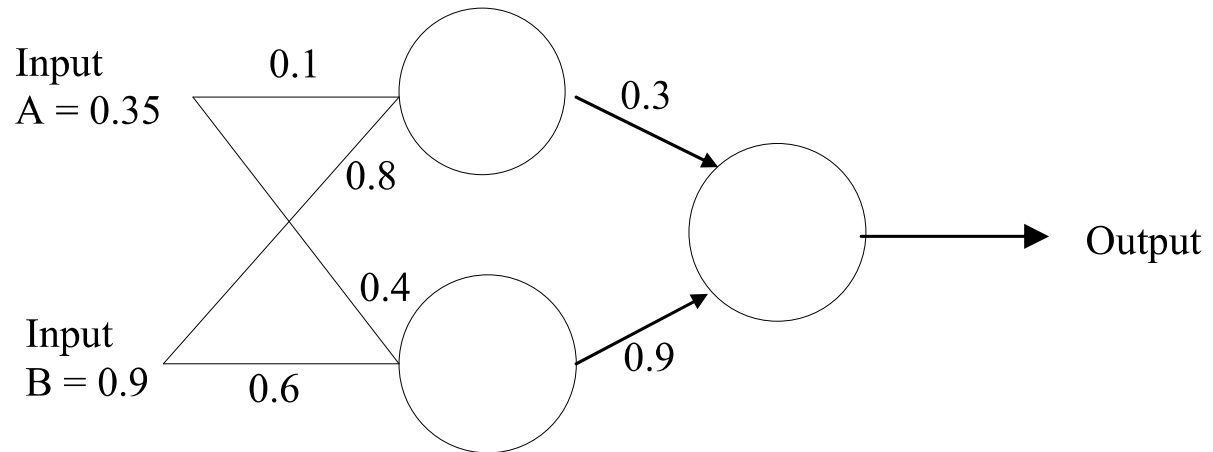
$$f_{\text{Sigmoid}}(x) = \frac{1}{1 + e^{-x}}$$



$$f'_{\text{Sigmoid}}(x) = f_{\text{Sigmoid}}(x)(1 - f_{\text{Sigmoid}}(x))$$

# Let us do some calculation

Consider the simple network below:



Assume that the neurons have a Sigmoid activation function and

1. Perform a forward pass on the network
2. Perform a reverse pass (training) once (target = 0.5)
3. Perform a further forward pass and comment on the result

# Let us do some calculation

Answer:

(i)

Input to top neuron =  $(0.35 \times 0.1) + (0.9 \times 0.8) = 0.755$ . Out = 0.68.

Input to bottom neuron =  $(0.9 \times 0.6) + (0.35 \times 0.4) = 0.68$ . Out = 0.6637.

Input to final neuron =  $(0.3 \times 0.68) + (0.9 \times 0.6637) = 0.80133$ . Out = 0.69.

(ii)

Output error  $\delta = (t - o)(1 - o)o = (0.5 - 0.69)(1 - 0.69)0.69 = -0.0406$ .

New weights for output layer

$w1^+ = w1 + (\delta \times \text{input}) = 0.3 + (-0.0406 \times 0.68) = 0.272392$ .

$w2^+ = w2 + (\delta \times \text{input}) = 0.9 + (-0.0406 \times 0.6637) = 0.87305$ .

Errors for hidden layers:

$\delta_1 = \delta \times w1 = -0.0406 \times 0.272392 \times (1 - o)o = -2.406 \times 10^{-3}$

$\delta_2 = \delta \times w2 = -0.0406 \times 0.87305 \times (1 - o)o = -7.916 \times 10^{-3}$

New hidden layer weights:

$w3^+ = 0.1 + (-2.406 \times 10^{-3} \times 0.35) = 0.09916$ .

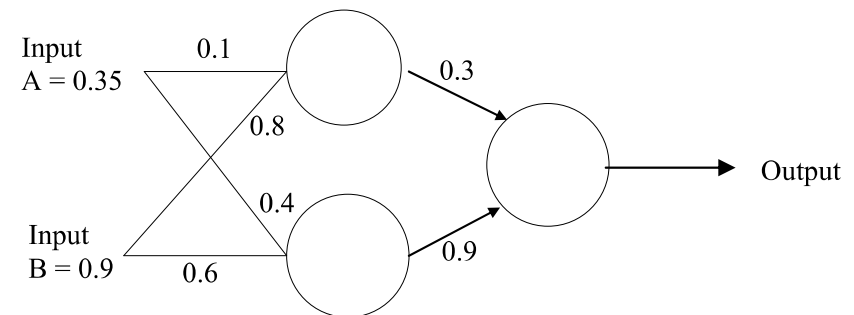
$w4^+ = 0.8 + (-2.406 \times 10^{-3} \times 0.9) = 0.7978$ .

$w5^+ = 0.4 + (-7.916 \times 10^{-3} \times 0.35) = 0.3972$ .

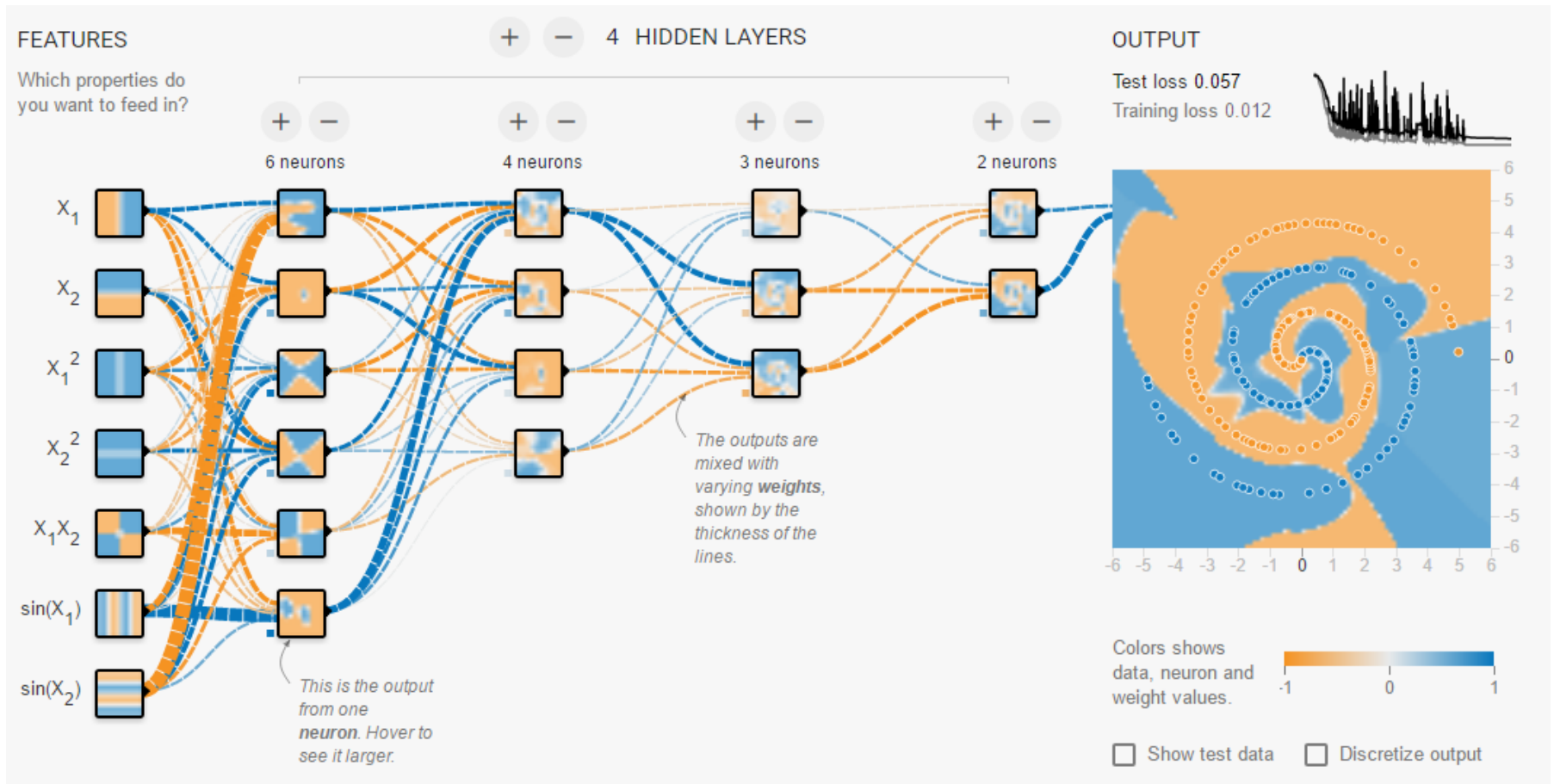
$w6^+ = 0.6 + (-7.916 \times 10^{-3} \times 0.9) = 0.5928$ .

(iii)

Old error was -0.19. New error is -0.18205. Therefore error has reduced.



# A demo from Google

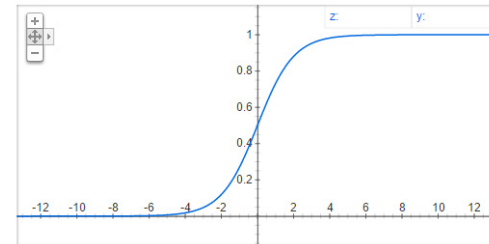


<http://playground.tensorflow.org/>

# Non-linear Activation Functions

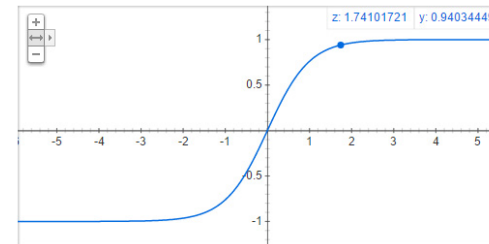
- Sigmoid

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



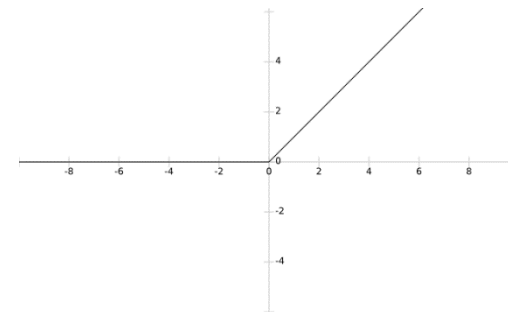
- Tanh

$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$



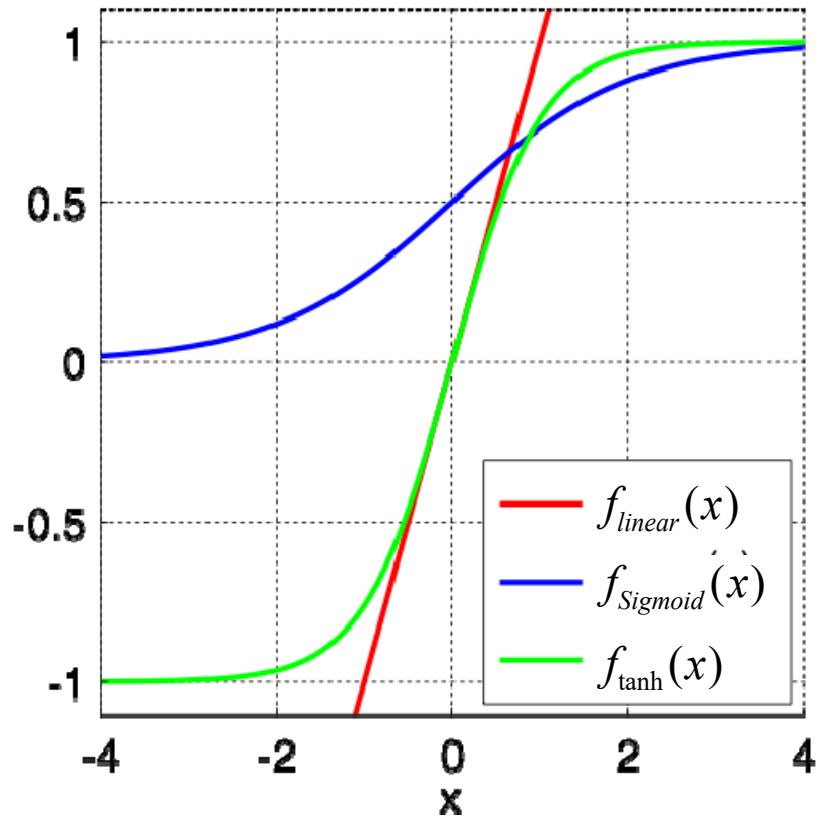
- Rectified Linear Unit (ReLU)

$$\text{ReLU}(z) = \max(0, z)$$

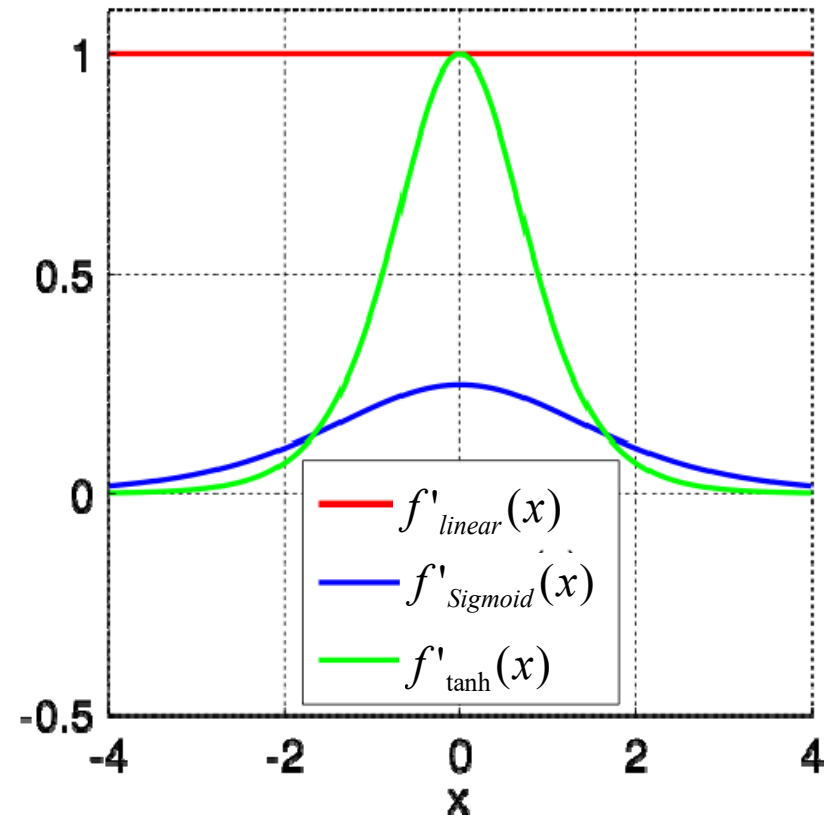


# Active functions

Some Common Activation Functions



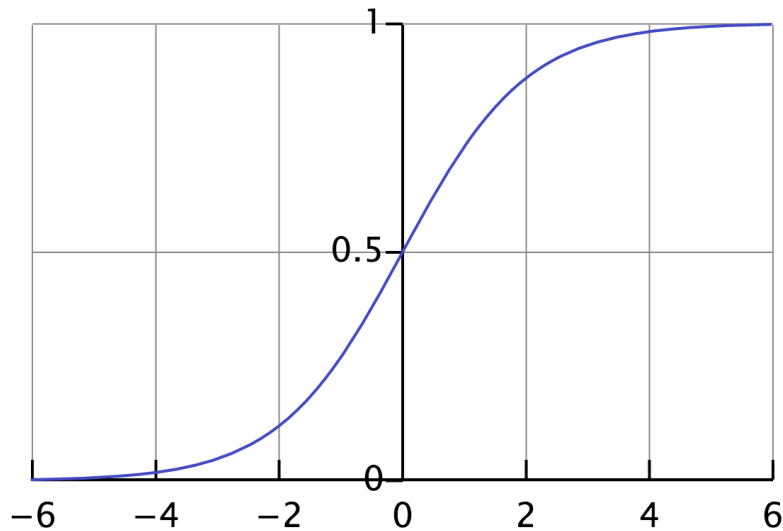
Activation Function Derivatives



# Activation functions

- **Logistic Sigmoid:**

$$f_{\text{Sigmoid}}(x) = \frac{1}{1 + e^{-x}}$$



Its derivative:

$$f'_{\text{Sigmoid}}(x) = f_{\text{Sigmoid}}(x)(1 - f_{\text{Sigmoid}}(x))$$

- Output range [0,1]
- Motivated by biological neurons and can be interpreted as the probability of an artificial neuron “firing” given its inputs
- However, saturated neurons make gradients vanished (**why?**)



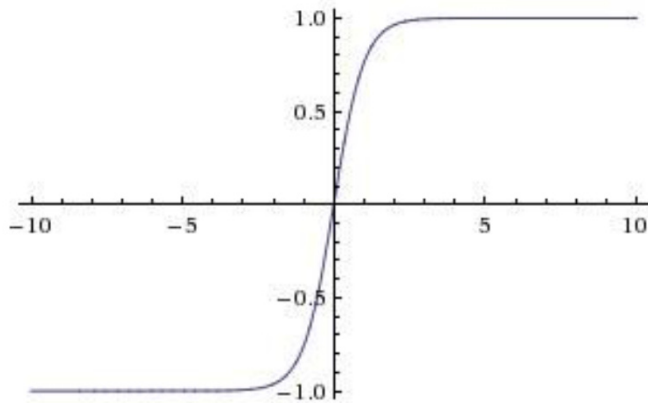
# Activation functions

- Tanh function

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

Its gradient:

$$f_{\tanh}(x) = 1 - f_{\tanh}(x)^2$$

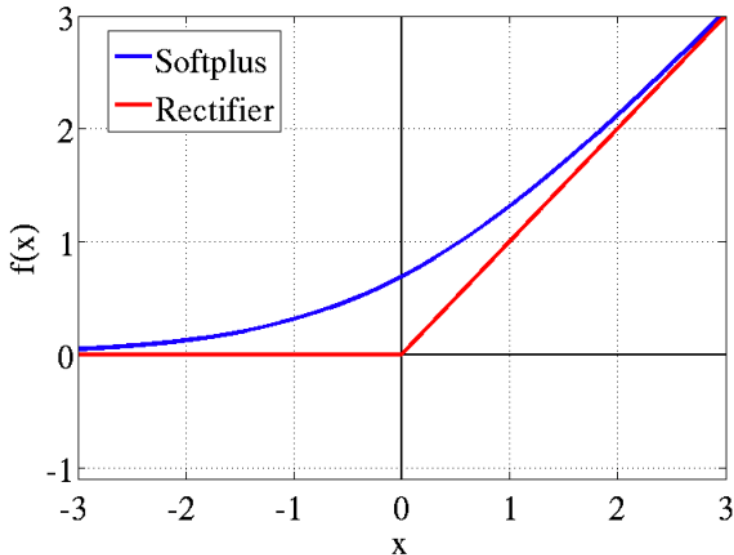


- Output range [-1,1]
- Thus strongly negative inputs to the tanh will map to negative outputs.
- Only zero-valued inputs are mapped to near-zero outputs
- These properties make the network less likely to get “stuck” during training

# Active Functions

- ReLU (rectified linear unit)

$$f_{\text{ReLU}}(x) = \max(0, x)$$



<http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf>

- The derivative:

$$f_{\text{ReLU}}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- Another version is

**Noise ReLU:**

$$f_{\text{NoisyReLU}}(x) = \max(0, x + N(0, \delta(x)))$$

- ReLU can be approximated by **softplus function**

$$f_{\text{Softplus}}(x) = \log(1 + e^x)$$

- ReLU gradient doesn't vanish as we increase x
- It can be used to model positive number
- It is fast as no need for computing the exponential function
- It eliminates the necessity to have a **"pretraining"** phase

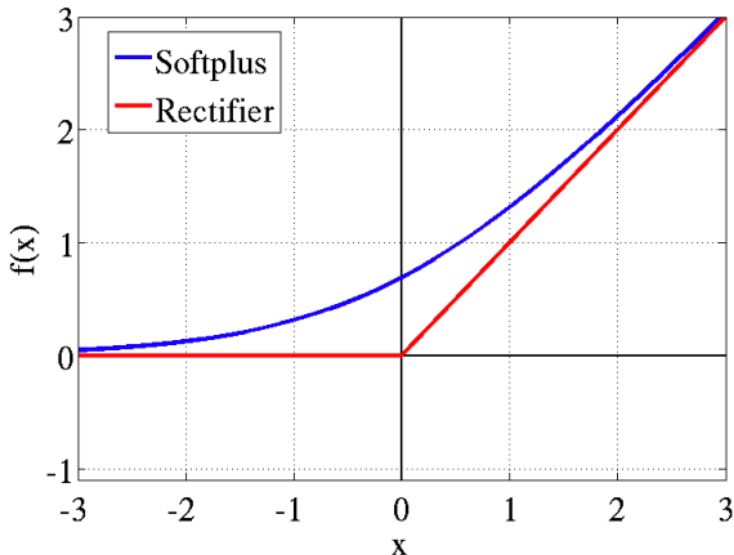
# Active Functions

- **ReLU (rectified linear unit)**

$$f_{\text{ReLU}}(x) = \max(0, x)$$

ReLU can be approximated by **softplus function**

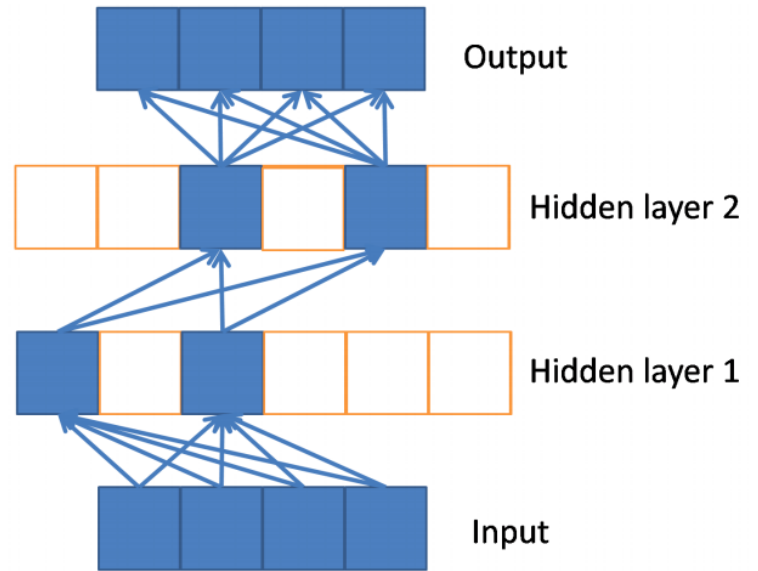
$$f_{\text{Softplus}}(x) = \log(1 + e^x)$$



Additional active functions:

**Leaky ReLU, Exponential LU, Maxout** etc

- The only non-linearity comes from the path selection with individual neurons being active or not
- It allows **sparse representations**:
  - for a given input only a subset of neurons are active



Sparse propagation of activations and gradients

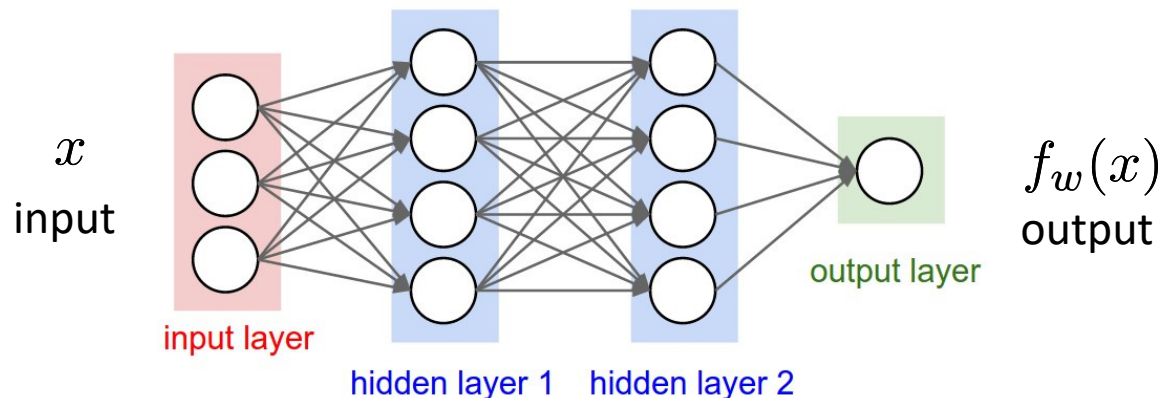
# Error/Loss function

- Recall stochastic gradient descent
  - Update from a randomly picked example (but in practice do a batch update)

$$w = w - \eta \frac{\partial \mathcal{L}(w)}{\partial w}$$

- Squared error loss for one binary output:

$$\mathcal{L}(w) = \frac{1}{2}(y - f_w(x))^2$$



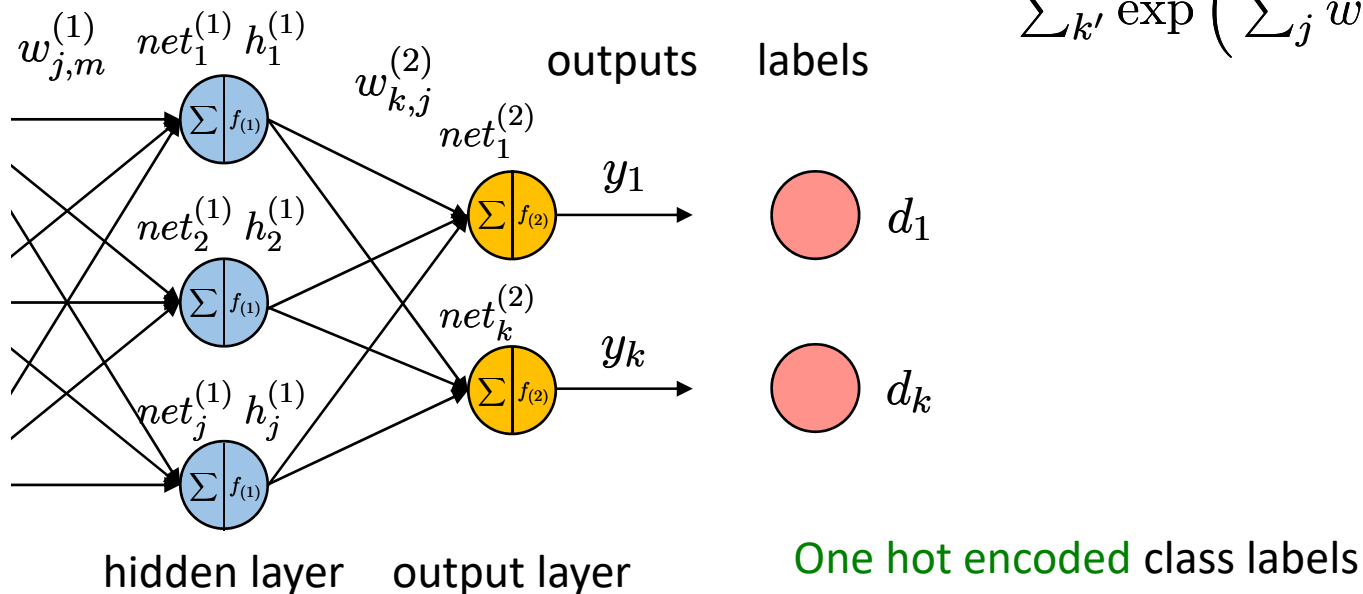
# Error/Loss function

- **Softmax** (cross-entropy loss) for multiple classes

(Class labels follow multinomial distribution)

$$\mathcal{L}(w) = - \sum_k (d_k \log \hat{y}_k + (1 - d_k) \log(1 - y_k))$$

$$\text{where } \hat{y}_k = \frac{\exp \left( \sum_j w_{k,j}^{(2)} h_j^{(1)} \right)}{\sum_{k'} \exp \left( \sum_j w_{k',j}^{(2)} h_j^{(1)} \right)}$$



For more machine learning details, you can check out my  
machine learning course at Zhiyuan College

# CS420 Machine Learning

Weinan Zhang

Course webpage:

<http://wnzhang.net/teaching/cs420/index.html>