

# Chapter 10

## Reinforcement Learning

Dana S. Nau  
University of Maryland



# Introduction

- Idea: agent interacts directly with the world
  - ▶ Improves performance by trial and error
  - ▶ Might or might not have a domain model
- Requires a *reward function*
  - ▶  $r(s, a, s')$  = reward for going from  $s$  to  $s'$  with action  $a$
- Agent tries to
  - ▶ remember which states/actions led to good/bad rewards
  - ▶ generalize
  - ▶ maximize long-term perceived benefits of its actions
- Simple RL: no teacher

# Value-Based RL

- Estimate  $V^*$  or  $Q^*$  by trial and error
- Example:
  - ▶ Three paella recipes  $a_1, a_2, a_3$
  - ▶ Rewards = ratings by guests

recipe	trial 1	trial 2	trial 3
$a_1$	7.2		
$a_2$	5.4	8	7.6
$a_3$	6.8		
$a_2$ avg	5.4	6.7	7

- $r(a, i)$  = reward for running  $a$  the  $i$ 'th time

$$Q_k(a) = \frac{1}{k} \sum_{i=1}^k r(a, i)$$

$$\begin{aligned} Q_{k+1}(a) &= \frac{1}{k+1} [r(a, k+1) + \sum_{i=1}^k r(a, i)] \\ &= \frac{1}{k+1} [r(a, k+1) + kQ_k(a)] \\ &= Q_k(a) + \frac{1}{k+1} [r(a, k+1) + kQ_k(a) - (k+1)Q_k(a)] \\ &= Q_k(a) + \frac{1}{k+1} [r(a, k+1) - Q_k(a)] \end{aligned}$$

- Update rule:  $Q(a) \leftarrow Q(a) + \alpha (r(a) - Q(a))$ 
  - ▶  $\alpha = 1/(k+1) = \text{learning rate}$
- Begin with initial value  $Q(a) = q_0$

# Value-Based RL

- Estimate  $V^*$  or  $Q^*$  by trial and error
- Example:
  - ▶ Three paella recipes  $a_1, a_2, a_3$
  - ▶ Rewards = ratings by guests

recipe	trial 1	trial 2	trial 3
$a_1$	7.2		
$a_2$	5.4	8	7.6
$a_3$	6.8		
$a_2$ avg	5.4	6.7	7

- Let  $r(a, i)$  = reward for running  $a$  the  $i$ 'th time

$$Q_k(a) = \frac{1}{k} \sum_{i=1}^k r(a, i)$$

$$\begin{aligned} Q_{k+1}(a) &= \frac{1}{k+1} [r(a, k+1) + \sum_{i=1}^k r(a, i)] \\ &= \frac{1}{k+1} [r(a, k+1) + kQ_k(a)] \\ &= Q_k(a) + \frac{1}{k+1} [r(a, k+1) + kQ_k(a) - (k+1)Q_k(a)] \\ &= Q_k(a) + \frac{1}{k+1} [r(a, k+1) - Q_k(a)] \end{aligned}$$

- *Temporal difference* update rule:
  - ▶  $Q(a) \leftarrow Q(a) + \alpha (r(a) - Q(a))$
  - ▶  $\alpha = \text{learning rate}$
- Which  $a$  to run next:
  - $\operatorname{argmax}_a Q(a)$  with probability  $1 - \epsilon$
  - other random with probability  $\epsilon$

**Poll:** What to use for  $\alpha$ ?

- A.  $1/(k+1)$
- B. any monotonically decreasing  $f(k) > 0$
- C. any constant  $> 0$
- D. any of the above
- E. don't know

# Value-Based RL

- Estimate  $V$  and  $Q$  by trial and error
- Example:
  - ▶ Three paella recipes  $a_1, a_2, a_3$
  - ▶ Rewards = ratings by guests

recipe	trial 1	trial 2	trial 3
$a_1$	7.2		
$a_2$	5.4	8	7.6
$a_3$	6.8		
$a_2$ avg	5.4	6.7	7

- Let  $r(a, i)$  = reward for running  $a$  the  $i$ 'th time

$$Q_k(a) = \frac{1}{k} \sum_{i=1}^k r(a, i)$$

$$\begin{aligned} Q_{k+1}(a) &= \frac{1}{k+1} [r(a, k+1) + \sum_{i=1}^k r(a, i)] \\ &= \frac{1}{k+1} [r(a, k+1) + kQ_k(a)] \\ &= Q_k(a) + \frac{1}{k+1} [r(a, k+1) + kQ_k(a) - (k+1)Q_k(a)] \\ &= Q_k(a) + \frac{1}{k+1} [r(a, k+1) - Q_k(a)] \end{aligned}$$

- *Temporal difference* update rule:
  - ▶  $Q(a) \leftarrow Q(a) + \alpha (r(a) - Q(a))$
  - ▶  $\alpha$  = learning rate

- Which  $a$  to run next:
  - $\operatorname{argmax}_a Q(a)$  ← with probability  $1 - \epsilon$
  - other random with probability  $\epsilon$

**Poll:** What to use for  $Q(a)$  here if  $a$  hasn't been tried yet?

A.  $\infty$       B. max possible  $r$

C. 0            D. anything  $> -\infty$

E. other        F. don't know

# Q-learning

- Consider an MDP in which we don't know  $\gamma$
- Adapt temporal-difference learning
  - ▶ Modified MDP model: maximize reward
- Same as before:
  - ▶  $\Sigma = (S, A, \gamma, \text{Pr}, \text{cost})$
  - ▶  $\gamma(s, a) = \{\text{all possible "next states" after applying action } a \text{ in state } s\}$
  - ▶  $\text{Pr}(s' | s, a) = \text{probability that } a \text{ takes us to } s'$ 
    - $\text{Pr}(s' | s, a) \neq 0$  iff  $s' \in \gamma(s, a)$
- Instead of  $\text{cost}(s, a, s')$ , we have
  - ▶  $r(s, a, s') = \text{reward if } a \text{ takes us to } s' \text{ from } s$
- Want a policy  $\pi$  that maximizes
  - ▶  $E(\text{reward for getting from } s_0 \text{ to } S_g)$
- Value function:
  - ▶ was  $V^\pi(s) = \sum_{s' \in \gamma(s, \pi(s))} \text{Pr}(s' | s, \pi(s)) [\text{cost}(s, \pi(s), s') + V^\pi(s')]$
  - ▶ now  $V^\pi(s) = \sum_{s' \in \gamma(s, \pi(s))} \text{Pr}(s' | s, \pi(s)) [r(s, \pi(s), s') + V^\pi(s')]$
- Bellman-update:
  - ▶ was  $Q(s, a) \leftarrow \sum_{s' \in \gamma(s, a)} \text{Pr}(s' | s, a) [\text{cost}(s, a, s') + V(s')]$ 
    - where  $V(s') = \min_a Q(s', a)$
  - ▶ now  $Q(s, a) \leftarrow \sum_{s' \in \gamma(s, a)} \text{Pr}(s' | s, a) [r(s, a, s') + V(s')]$ 
    - where  $V(s') = \max_a Q(s', a)$
- Temporal-difference update rule:
  - ▶  $Q(a) \leftarrow Q(a) + \alpha (r(a) - Q(a))$
- Temporal-difference update for MDPs:
  - ▶  $Q(s, a) \leftarrow Q(s, a) + \alpha (r(s, a, s') + V(s') - Q(s, a))$
- Equivalently:
  - ▶  $Q(s, a) \leftarrow Q(s, a) + \alpha (r(s, a, s') + \max_{a'} Q(s', a') - Q(s, a))$

# Q-learning

- Consider an MDP in which we don't know  $\gamma$
- Adapt temporal-difference learning
  - ▶ Modified MDP model: maximize reward

From previous slide

```
Q-learning
  initialize  $Q$  and a starting state  $s$ 
  until Termination do
1   |  $a \leftarrow \text{Select}(s)$ 
2   | perform action  $a$ 
3   | observe resulting state  $s'$  and reward  $r(s, a, s')$ 
4   |  $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a, s') + \max_{a'}\{Q(s', a')\} - Q(s, a)]$ 
5   |  $s \leftarrow s'$ 
```

**Poll.** Is line 2 practical?

A. Yes   B. No   C. don't know

# Q-learning

- Learn in two stages:
  1. Learn from simulated acting:
    - Replace lines 2, 3 with  
 $(s', r(s, a, s')) \leftarrow \text{Simulate}(a, s)$
  2. Learn from real-world acting
    - Use lines 2, 3 as shown

## Q-learning

```
initialize  $Q$  and a starting state  $s$ 
until Termination do
1    $a \leftarrow \text{Select}(s)$ 
2   perform action  $a$ 
3   observe resulting state  $s'$  and reward  $r(s, a, s')$ 
4    $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a, s') + \max_{a'}\{Q(s', a')\} - Q(s, a)]$ 
5    $s \leftarrow s'$ 
```

**Poll.** Is Stage 2 needed?

A. Yes   B. No   C. don't know

**Poll.** Is it practical to store  $Q(s, a)$  for every  $s$  and  $a$ ?

A. Yes   B. No   C. don't know



# Parametric Learning

- Input: a set of numeric  $(x,y)$  pairs
  - ▶  $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, \dots, N\}$
- Parametric function  $f_{\theta}(x)$ 
  - ▶  $\theta$  is a parameter that changes  $f$
  - ▶ adjust  $\theta$  to try to make  $f_{\theta}(x^{(i)}) \approx y^{(i)}$  for each  $i$

- Example: *linear regression*

- ▶  $f_{\theta}(x) = \theta_0 + \theta_1 x$
- ▶  $\theta = (\theta_0, \theta_1)$

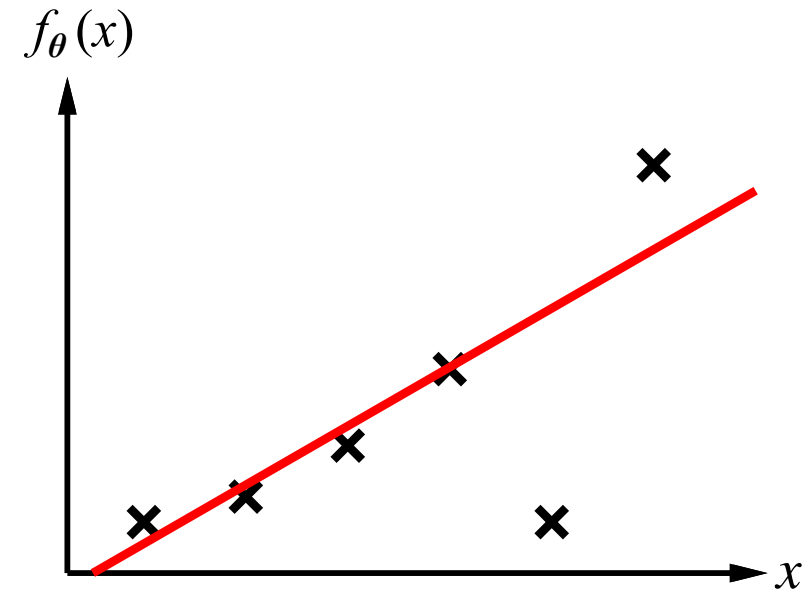
- Empirical loss function

$$\begin{aligned} Loss &= \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} (f_{\theta}(x) - y)^2 \\ &= \frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} (\theta_0 + \theta_1 x - y)^2 \end{aligned}$$

- *Loss* is minimized when both of these are 0:

$$\frac{\partial Loss}{\partial \theta_0} = \frac{2}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} (\theta_0 + \theta_1 x - y)$$

$$\frac{\partial Loss}{\partial \theta_1} = \frac{2}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} x(\theta_0 + \theta_1 x - y)$$



Credit: [Stuart Russell](#)

# Linear Regression Example

- Example: (*seafood, rice*) proportions in paella

- ▶  $\mathcal{D} = \{(150, 300), (300, 400), (450, 700)\}$

- ▶ 
$$\frac{\partial Loss}{\partial \theta_0} = \frac{2}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} (\theta_0 + \theta_1 x - y)$$
$$= (2/3) [(\theta_0 + 150 \theta_1 - 300) + (\theta_0 + 300 \theta_1 - 400) + (\theta_0 + 450 \theta_1 - 700)]$$
$$= (2/3) [3 \theta_0 + 900 \theta_1 - 1400]$$

- ▶ 
$$\frac{\partial Loss}{\partial \theta_1} = \frac{2}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} x(\theta_0 + \theta_1 x - y)$$
$$= (2/3) [150(\theta_0 + 150 \theta_1 - 300) + 300(\theta_0 + 300 \theta_1 - 400) + 450(\theta_0 + 450 \theta_1 - 700)]$$
$$= (2/3) [900 \theta_0 + 315,000 \theta_1 - 480,000]$$

- Solve:

$$3 \theta_0 + 900 \theta_1 - 1400 = 0$$

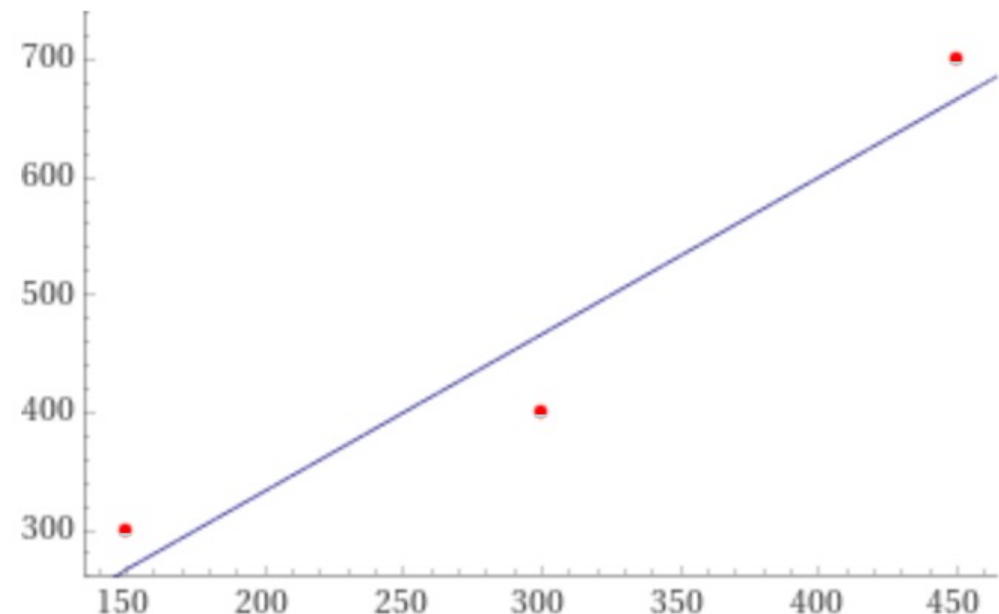
$$900 \theta_0 + 315,000 \theta_1 - 480,000 = 0$$

- Answer:

- $\theta_0 = 200/3 \approx 66.6$

- $\theta_1 = 4/3 \approx 1.33$

- $f_{\theta}(x,y) = \theta_0 + \theta_1 x = 1.33x + 66.6$



# Multivariable Linear Regression

- $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, \dots, N\}$ 
  - ▶ Each  $\mathbf{x}^{(i)}$  is a column vector  $[x_1, \dots, x_n]^\top$
- $\boldsymbol{\theta}$  is a column vector  $[\theta_0, \dots, \theta_n]^\top$ 
  - ▶  $f_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \sum_{1 \leq j \leq n} \theta_j x_j$
- Extend  $\mathbf{x}$  to  $[x_0, x_1, \dots, x_n]^\top$ 
  - ▶  $x_0 = 1$  for every  $\mathbf{x} \in \mathcal{D}$
  - ▶  $f_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{0 \leq j \leq n} \theta_j x_j = \boldsymbol{\theta} \cdot \mathbf{x}$
  - ▶  $Loss = (1/|\mathcal{D}|) \sum_{(\mathbf{x}, y) \in \mathcal{D}} (\boldsymbol{\theta} \cdot \mathbf{x} - y)^2$
- Partial derivatives:
  - ▶  $\partial Loss / \partial \theta_j = (2/|\mathcal{D}|) \sum_j (\boldsymbol{\theta} \cdot \mathbf{x} - y) x_j$
  - ▶  $\nabla Loss = [\partial Loss / \partial \theta_1, \dots, \partial Loss / \partial \theta_n]^\top$
- Can solve analytically to minimize  $Loss$ 
  - ▶ but it's more complicated

GradientDescent( $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid 1 \leq i \leq N\}$ )

**until** convergence **do**

**foreach**  $0 \leq j \leq n$  **do**

$\theta_j \leftarrow \theta_j - \alpha \sum_{1 \leq i \leq N} (\boldsymbol{\theta} \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$

- In each iteration of the outer loop
  - ▶ Move each  $\theta_j$  in the direction of  $\partial Loss / \partial \theta_j$
  - ▶ How far depends on  $\partial Loss / \partial \theta_j$  and  $\alpha$
- Stop when each  $\boldsymbol{\theta}$  stops changing very much
- Two ways to call GradientDescent:
  - ▶ *Batch*: argument is all of  $\mathcal{D}$
  - ▶ *Stochastic*: argument is random subset of  $\mathcal{D}$
- Can generalize to nonlinear  $f_{\boldsymbol{\theta}}$ 
  - ▶ update rule  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla Loss(f_{\boldsymbol{\theta}})$

# Parametric Q-Learning (Motivation)

- From before:

## Q-learning

initialize  $Q$  and a starting state  $s$

**until** *Termination* **do**

```
1    $a \leftarrow \text{Select}(s)$  // selects  $a \in \text{Applicable}(s)$ 
2   perform action  $a$ 
3   observe resulting state  $s'$ , reward  $r(s, a, s')$ 
4    $Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a, s') +$   
     $\max_{a'} \{Q(s', a')\} - Q(s, a)]$ 
5    $s \leftarrow s'$ 
```

- Problems with Q-learning:
  - ▶ Need a large table to store  $Q(s, a)$
  - ▶ Doesn't generalize to new situations
- Idea: develop a parameterized formula  $Q_\theta(s, a) \approx Q(s, a)$ 
  - ▶ Adjust  $\theta$  to make  $Q_\theta(s, a)$  fit observed rewards
  - ▶ Use gradient descent

# Parametric Q-Learning

- Use gradient descent to learn  $Q_\theta$

$$Loss = [r(s, a, s') + \max_{a'} \{Q_\theta(s', a')\} - Q_\theta(s, a)]^2$$

$$\frac{\partial Loss}{\partial \theta_j} = -2[r(s, a, s') + \max_{a'} \{Q_\theta(s', a')\} - Q_\theta(s, a)] \frac{\partial Q_\theta(s, a)}{\partial \theta_j}$$

Update rule:

$$\theta_j \leftarrow \theta_j + \alpha [r(s, a, s') + \max_{a'} \{Q_\theta(s', a')\} - Q_\theta(s, a)] \frac{\partial Q_\theta(s, a)}{\partial \theta_j}$$

**Poll.** Should this be  $-2\alpha$  instead?

A. Yes   B. No   C. don't know

## Q-learning

initialize  $Q$  and a starting state  $s$

**until Termination do**

- 1  $a \leftarrow \text{Select}(s)$  // selects  $a \in \text{Applicable}(s)$
- 2 perform action  $a$
- 3 observe resulting state  $s'$ , reward  $r(s, a, s')$
- 4  $Q(s, a) \leftarrow Q(s, a) + \alpha [r(s, a, s') + \max_{a'} \{Q(s', a')\} - Q(s, a)]$
- 5  $s \leftarrow s'$

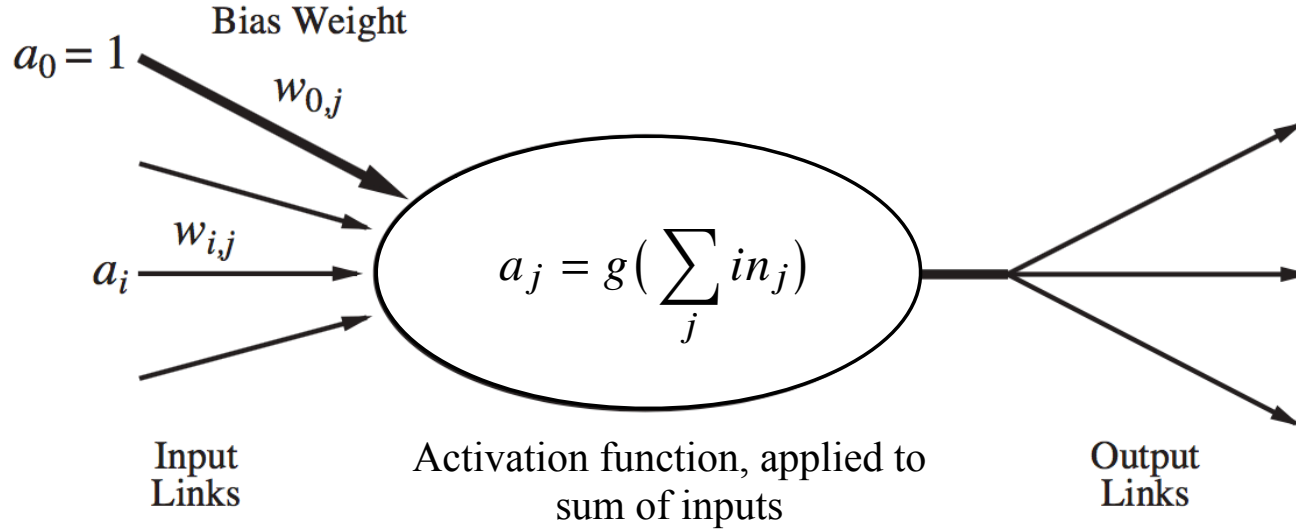
## Q-learning

initialize  $\theta$  and a starting state  $s$

**until Termination do**

- 1  $a \leftarrow \text{Select}(s)$
- 2 perform action  $a$
- 3 observe resulting state  $s'$  and reward  $r(s, a, s')$
- 4 **forall** parameters  $\theta_j$  **do**
- 5  $\theta_j \leftarrow \theta_j + \alpha [r(s, a, s') + \max_{a'} \{Q_\theta(s', a')\} - Q_\theta(s, a)] \frac{\partial Q_\theta(s, a)}{\partial \theta_j}$
- 6  $s \leftarrow s'$

# McCulloch-Pitts “unit”

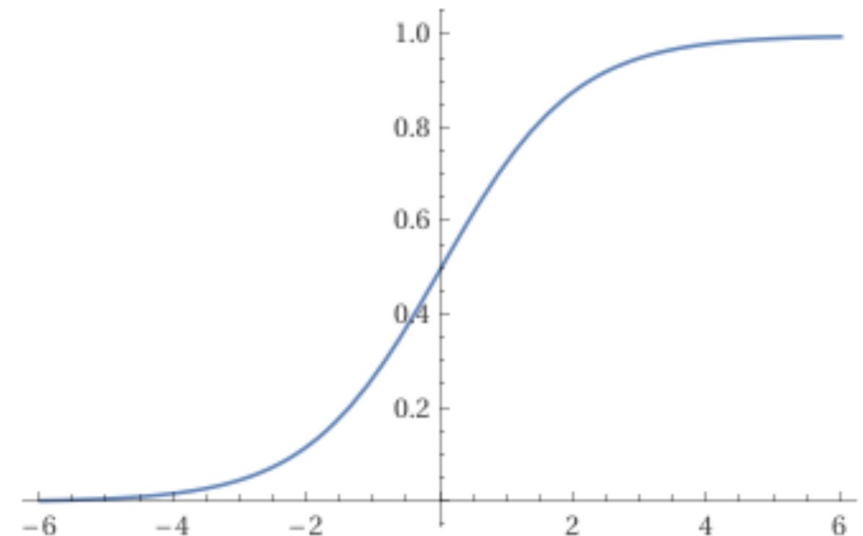


- Output:  $a_j = g(in_j)$
- $g$ : activation function
  - ▶ nondecreasing
  - ▶ usually positive, often nonlinear
  - ▶ e.g., logistic (sigmoid):  
 $g(z) = 1/(1+e^{-z})$

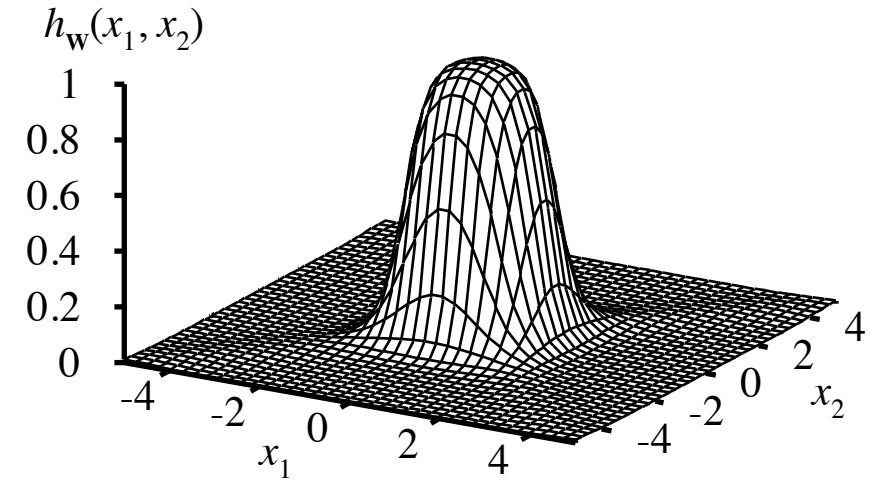
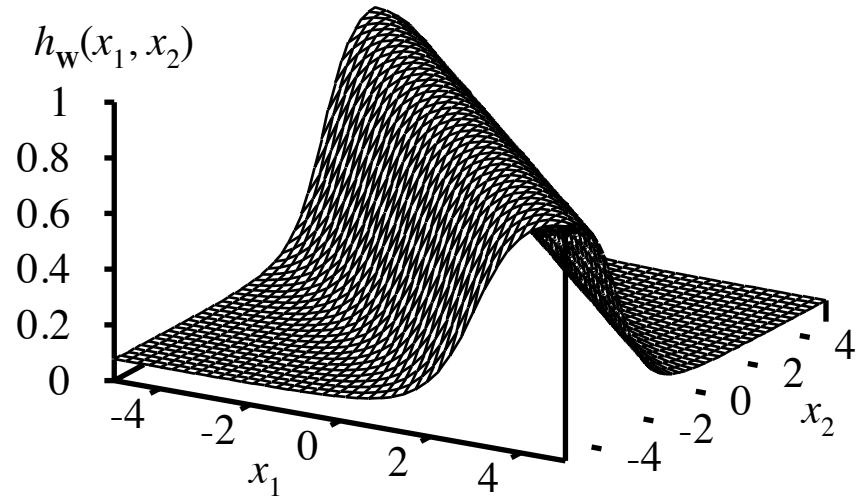
- Simple mathematical model
  - ▶ Inspired by neurons, but much simpler
- Output of unit  $j$  depends on weighted sum of inputs

- ▶  $\mathbf{w} = \begin{bmatrix} w_{0,j} \\ \vdots \\ w_{n,j} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix}$

- ▶  $in_j = \sum_i w_{i,j} a_i = \mathbf{w}^\top \mathbf{x}$

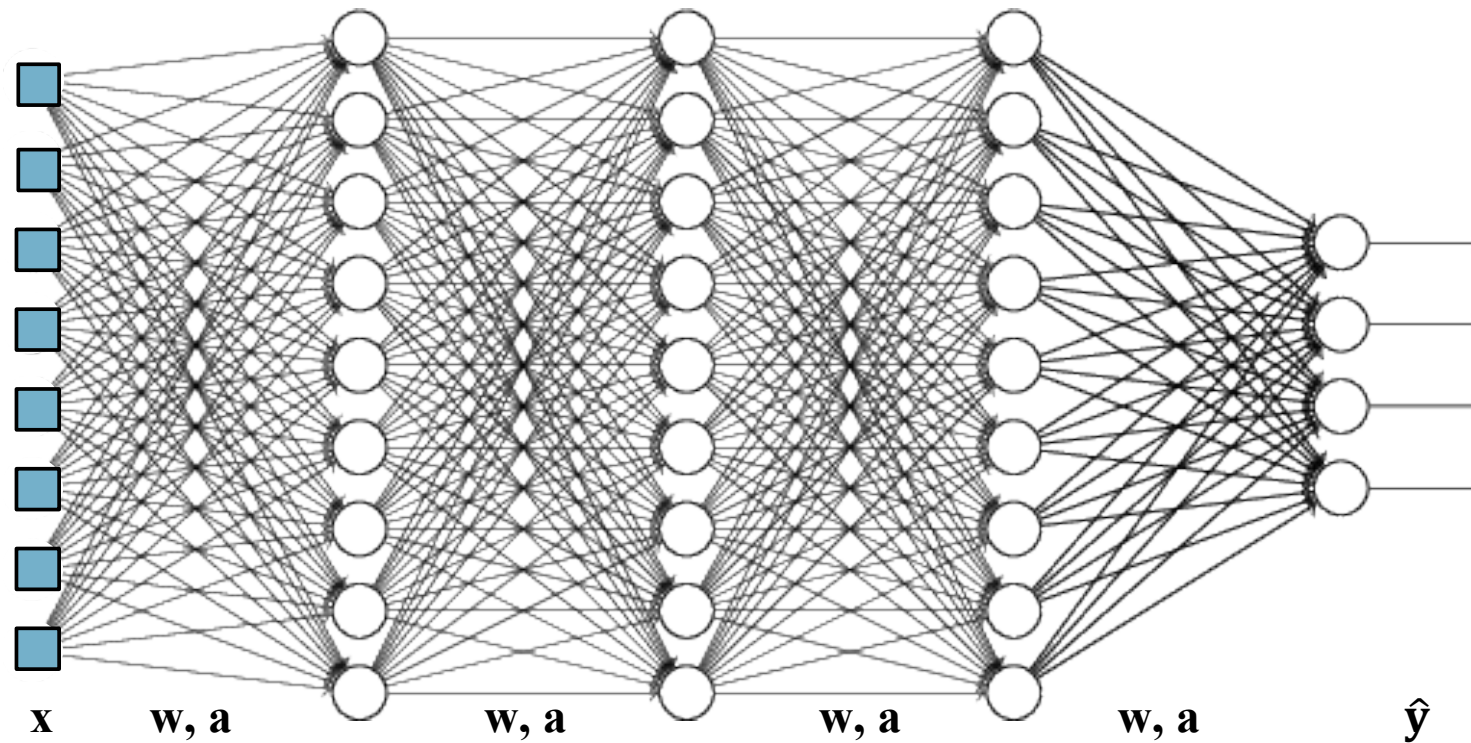


# Expressiveness



- With 2 layers and enough units in those layers, can approximate all continuous functions
  - ▶ with exponentially many units, can get any desired accuracy
- With 3 layers, all functions
- 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

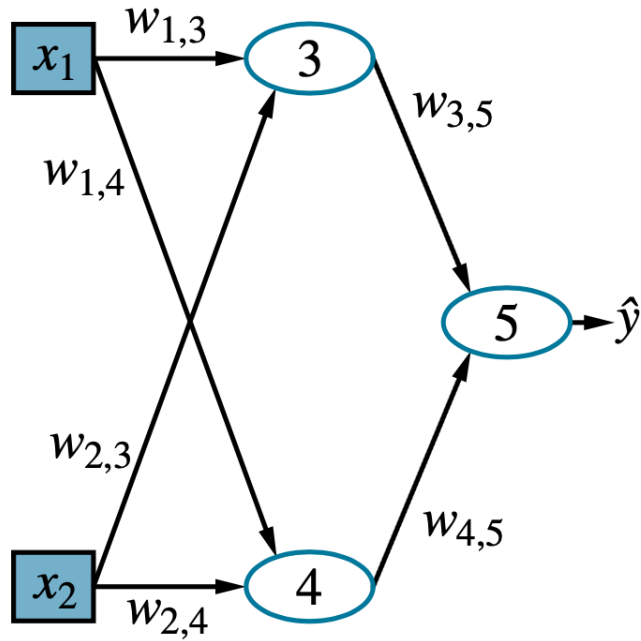
# Multilayer Feed-Forward Networks



- Acyclic network of units
  - ▶ usually organized into layers
  - ▶ connections go from left to right
  - ▶ how many layers, and units in each layer, typically chosen by hand
- Input  $\mathbf{x} = [x_1, \dots, x_n]$       Output  $\hat{\mathbf{y}} = [\hat{y}_1, \dots, \hat{y}_k]$



# Example



$$a_3 = g_3(w_{1,3} x_1 + w_{2,3} x_2)$$

$$a_4 = g_4(w_{1,4} x_1 + w_{2,4} x_2)$$

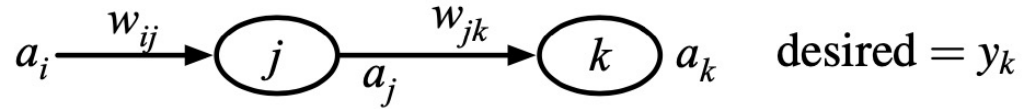
$$\hat{y} = g_5(w_{3,5} a_3 + w_{4,5} a_4)$$

$$= g_5[w_{3,5} g_3(w_{1,3} x_1 + w_{2,3} x_2) + w_{4,5} g_4(w_{1,4} x_1 + w_{2,4} x_2)]$$

$$= h_{\mathbf{w}}(\mathbf{x})$$

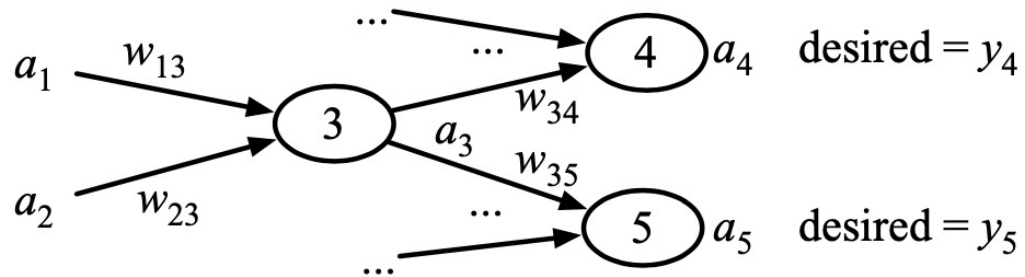
- Let  $\mathbf{W}^{(i)}$  = matrix of weights in  $i$ 'th layer
  - $\mathbf{W}^{(1)} = \begin{bmatrix} w_{1,3} & w_{1,4} \\ w_{2,3} & w_{2,4} \end{bmatrix}$
  - $\mathbf{W}^{(2)} = \begin{bmatrix} w_{3,5} \\ w_{4,5} \end{bmatrix}$
- Let  $\mathbf{g}^{(i)}$  = column vector of activation functions
  - $\mathbf{g}^{(1)} = \begin{bmatrix} g_3 \\ g_4 \end{bmatrix}$
  - $\mathbf{g}^{(2)} = [g_5]$
- Then  $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{g}^{(2)} (\mathbf{W}^{(2)} \mathbf{g}^{(1)} (\mathbf{W}^{(1)} \mathbf{x}))$ 
  - Parameterized family of functions
- Adjusting weights changes the function
  - do learning this way

# Back-Propagation Learning



- Backward from the output layer

- ▶ output layer:  $\Delta_k = (y_k - a_k) g'(in_k); \quad w_{jk} \leftarrow w_{jk} + \alpha a_j \Delta_k$
- ▶ hidden layer:  $\Delta_j = (\sum_k w_{jk} \Delta_k) g'(in_j); \quad w_{ij} \leftarrow w_{ij} + \alpha a_i \Delta_j$



$$\Delta_4 = (y_4 - a_4) g'(in_4); \quad w_{34} \leftarrow w_{34} + \alpha a_3 \Delta_4$$

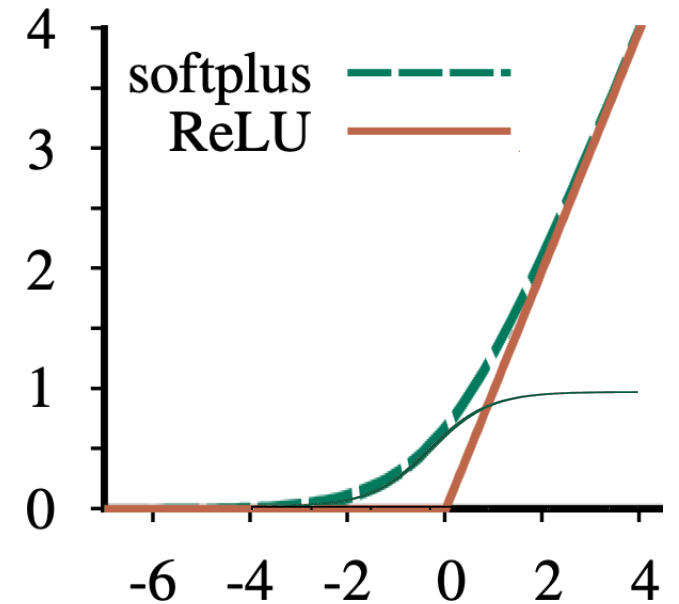
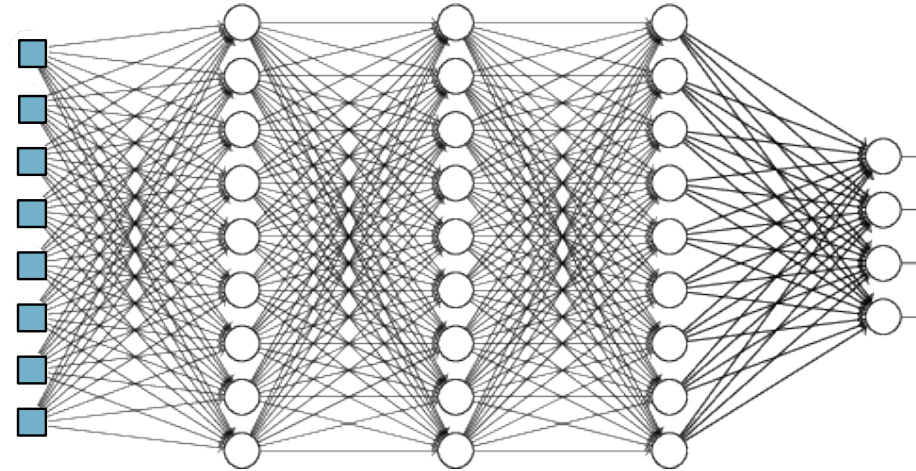
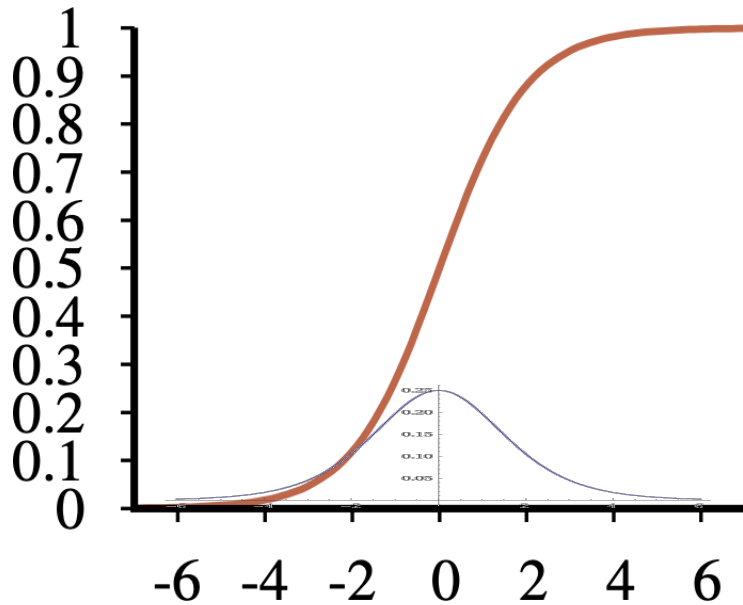
$$\Delta_5 = (y_5 - a_5) g'(in_5); \quad w_{35} \leftarrow w_{35} + \alpha a_3 \Delta_5$$

$$\Delta_3 = (w_{34} \Delta_4 + w_{35} \Delta_5) g'(in_3); \quad \begin{cases} w_{13} \leftarrow w_{13} + \alpha a_1 \Delta_3 \\ w_{23} \leftarrow w_{23} + \alpha a_2 \Delta_3 \end{cases}$$

Two ways:

- Batch:
  - ▶ At each *epoch*, compute the updates for all examples, then apply
  - ▶ Don't apply any of the updates until you've seen all the examples
- Online:
  - ▶ Update after each example
- Can lead to different results
  - ▶ At one time, batch was thought to be theoretically superior
  - ▶ In practice, online seems to work better

# Activation Functions



- logistic (sigmoid):

$$g(z) = 1/(1+e^{-z})$$

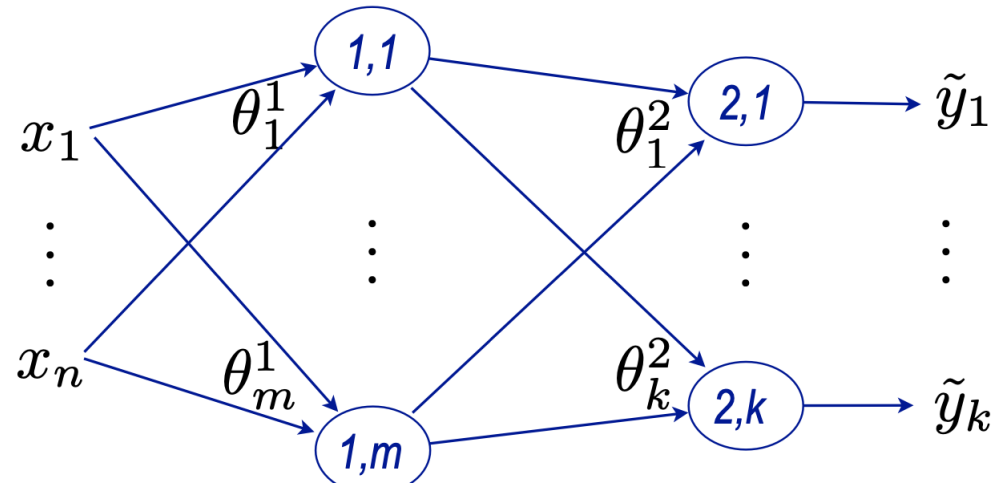
$$g'(z) = e^{-z}/(1+e^{-z})^2$$

- Problem with sigmoid in deep multilayer networks
  - ▶ *Vanishing gradient:*  
 $g'$  too small to be useful
- ReLU or softplus are widely used instead

- rectified linear (ReLU):  
 $g(z) = \max(0, z)$   
 $g'(z) = 0$  if  $z < 0$ ; 1 if  $z > 0$
- softplus:  
 $g(z) = \log(1+e^z)$   
 $g'(z) = 1/(1+e^{-z})$

# Relation to the Book

- $\hat{y} \rightarrow \tilde{y}$
- $w \rightarrow \theta$  (with subscript, superscript)
- $w \rightarrow \theta$



## Backpropagation( $x, y$ )

$$\mathbf{x}^1 \leftarrow \mathbf{x}$$

**for**  $l = 1$  to  $L$  **do**

$$\left[ \mathbf{x}^{l+1} \leftarrow g(\Theta^l \times \mathbf{x}^l) \right.$$

$$\delta^{L+1} \leftarrow (\mathbf{x}^{L+1} - \mathbf{y}) \times g'(\Theta^L \times \mathbf{x}^L)$$

$$\Theta^L \leftarrow \Theta^L - \alpha[\delta^{L+1} \otimes (\mathbf{x}^L)^\top]$$

**for**  $l = L$  to  $2$  **do**

$$\left[ \delta^l \leftarrow [(\Theta^l)^\top \times \delta^{l+1}] \times g'(\Theta^{l-1} \times \mathbf{x}^{l-1}) \right.$$

$$\left. \Theta^{l-1} \leftarrow \Theta^{l-1} - \alpha[\delta^l \otimes (\mathbf{x}^{l-1})^\top] \right]$$

# Summary

- McCulloch-Pitts “units”
  - ▶ weighted input, bias weight
  - ▶ activation function
- Expressivity
- Multi-layer feed-forward networks
- Back-propagation learning
  - ▶ Gradient descent
  - ▶ ReLU