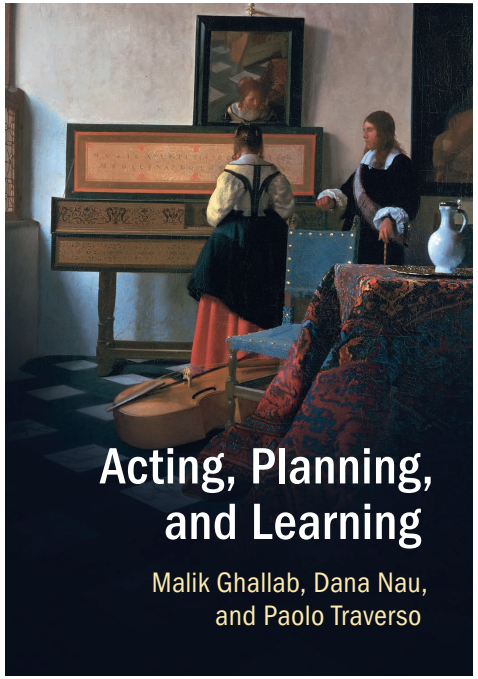
Last update: 2:49 PM, March 12, 2025



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

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

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

- Update rule: $Q(a) \leftarrow Q(a) + \alpha\,(r(a) - Q(a))$
  - $\alpha = 1/(k+1)$ = *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

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

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

- *Temporal difference* update rule:
  - $Q(a) \leftarrow Q(a) + \alpha\,(r(a) - Q(a))$
  - α = *learning rate*
- Which a to run next:

  $\text{argmax}_a\, Q(a)$ with probability $1-\varepsilon$

  other random with probability ε

Poll: What to use for α?

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

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

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

- *Temporal difference* update rule:
  - $Q(a) \leftarrow Q(a) + \alpha\,(r(a) - Q(a))$
  - α = *learning rate*
- Which a to run next:

  $\text{argmax}_a\, Q(a)$ with probability $1-\varepsilon$

  other random with probability ε

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

A. ∞ 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 γ
- Adapt temporal-difference learning
  - Modified MDP model: maximize reward

- Same as before:
  - $\Sigma = (S, A, \gamma, \Pr, \text{cost})$
  - $\gamma(s,a)$ = {all possible "next states" after applying action a in state s}
  - $\Pr(s' \mid s, a)$ = probability that a takes us to s'
    - $\Pr(s' \mid s, a) \neq 0$ iff $s' \in \gamma(s,a)$

- Instead of cost(s,a,s'), we have
  - $r(s,a,s')$ = reward if a takes us to s' from s
- Want a policy π that maximizes
  - E(reward for getting from s_0 to S_g)

- Value function:
  - was $V^\pi(s) = \sum_{s' \in \gamma(s,\pi(s))} \Pr(s' \mid s, \pi(s))[\text{cost}(s,\pi(s),s') + V^\pi(s')]$
  - now $V^\pi(s) = \sum_{s' \in \gamma(s,\pi(s))} \Pr(s' \mid s, \pi(s))[r(s,\pi(s),s') + V^\pi(s')]$

- Bellman-update:
  - was $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s' \mid s,a)\ [\text{cost}(s,a,s') + V(s')]$
    - where $V(s') = \min_a Q(s',a)$
  - now $Q(s,a) \leftarrow \sum_{s' \in \gamma(s,a)} \Pr(s' \mid s,a)\ [r(s,a,s') + V(s')]$
    - 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 γ
- 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
  - $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, \ldots, N\}$
- Parametric function $f_\theta(x)$
  - θ is a parameter that changes f
  - adjust θ to try to make $f_\theta(x^{(i)}) \approx y^{(i)}$ for each i

- Example: *linear regression*
  - $f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x$
  - $\boldsymbol{\theta} = (\theta_0, \theta_1)$
- Empirical loss function

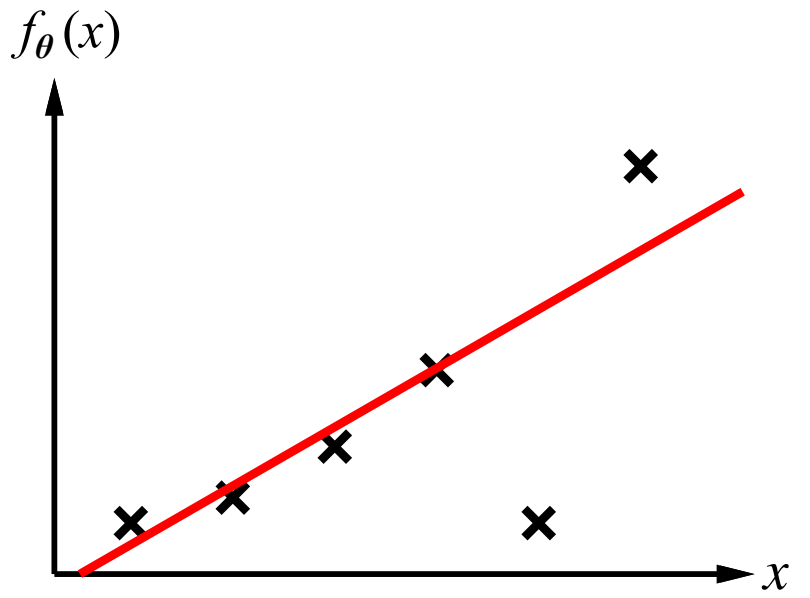
$$Loss = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} (f_\theta(x) - y)^2$$

$$= \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} (\theta_0 + \theta_1 x - y)^2$$

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

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

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

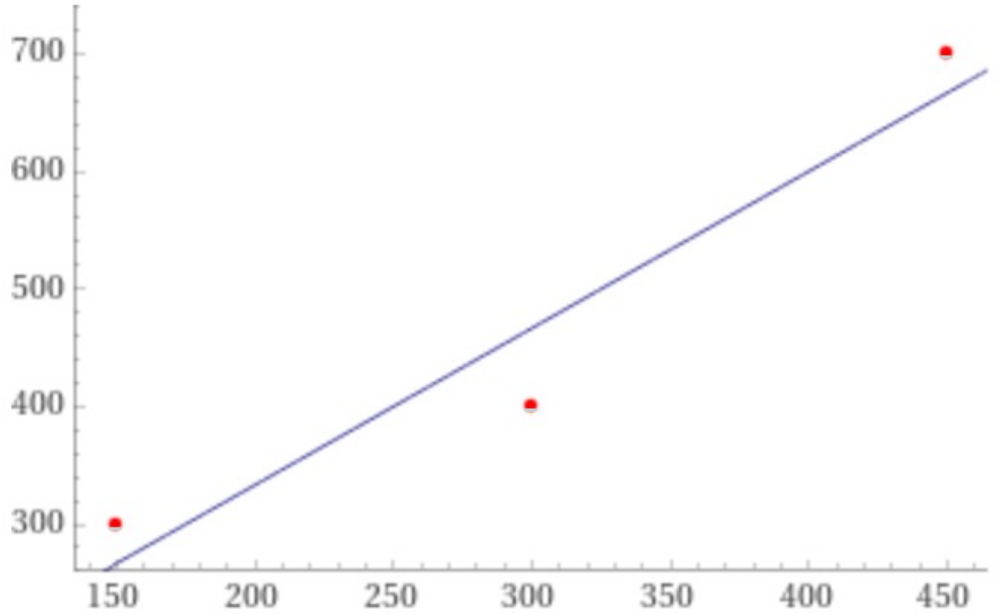


Credit: Stuart Russell

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

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

GradientDescent($\{(\mathbf{x}^{(i)}, y^{(i)}) \mid 1 \le i \le N\}$)
  until convergence **do**
    foreach $0 \le j \le n$ **do**
      $\theta_j \leftarrow \theta_j - \alpha \sum_{1 \le i \le N} (\boldsymbol{\theta} \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$

- In each iteration of the outer loop
  - Move each θ_j in the direction of $\partial Loss/\partial \theta_j$
  - How far depends on $\partial Loss/\partial \theta_j$ and α
- 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* $a \in Applicable(s)$
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 $\boldsymbol{\theta}$ to make $Q_\theta(s,a)$ fit observed rewards
  - Use gradient descent

Parametric Q-Learning

- Use gradient descent to learn Q_θ

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

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

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 -2α instead?

A. Yes B. No C. don't know

Q-learning

initialize Q and a starting state s

until *Termination* **do**

1. $a \leftarrow$ Select(s) // *selects* $a \in Applicable(s)$
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. $s \leftarrow s'$

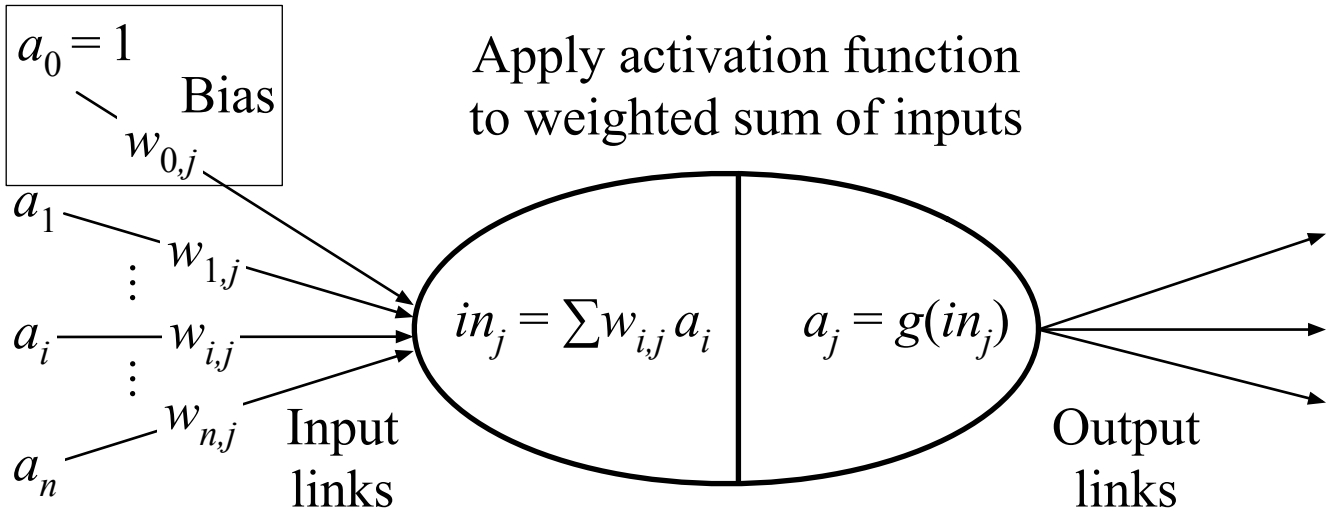
Q-learning

initialize $\boldsymbol{\theta}$ and a starting state s

until *Termination* **do**

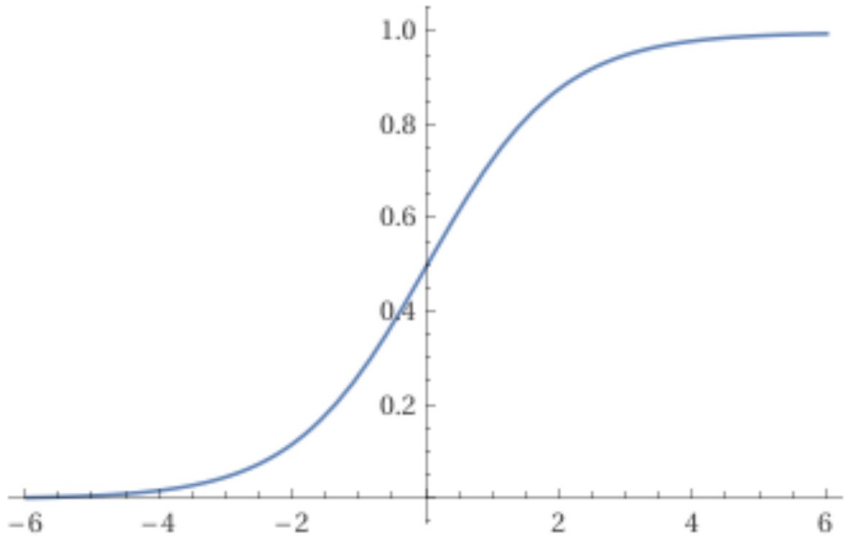
1. $a \leftarrow$ Select(s)
2. perform action a
3. observe resulting state s' and reward $r(s,a,s')$
4. **forall** parameters θ_j **do**
5. $\theta_j \leftarrow \theta_j + \alpha[r(s,a,s') + \max_{a'}\{Q_\theta(s',a')\} - Q_\theta(s,a)]\frac{\partial Q_\theta(s,a)}{\partial \theta_j}$
6. $s \leftarrow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

McCulloch-Pitts "unit"

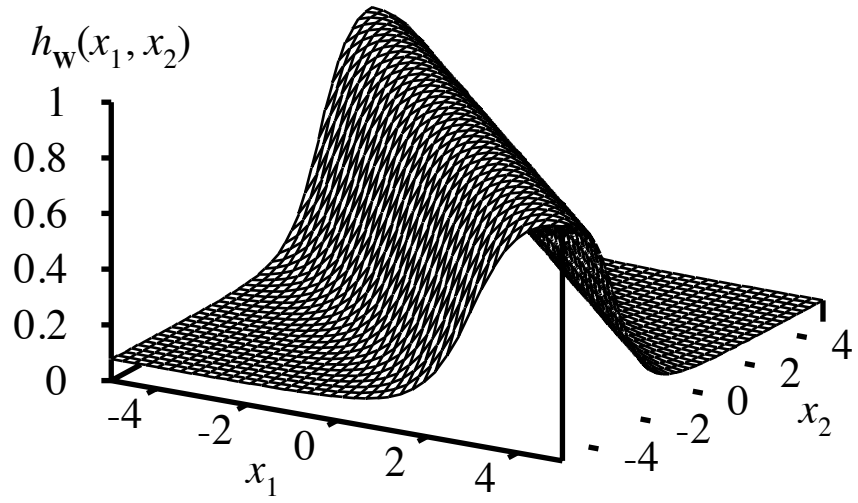


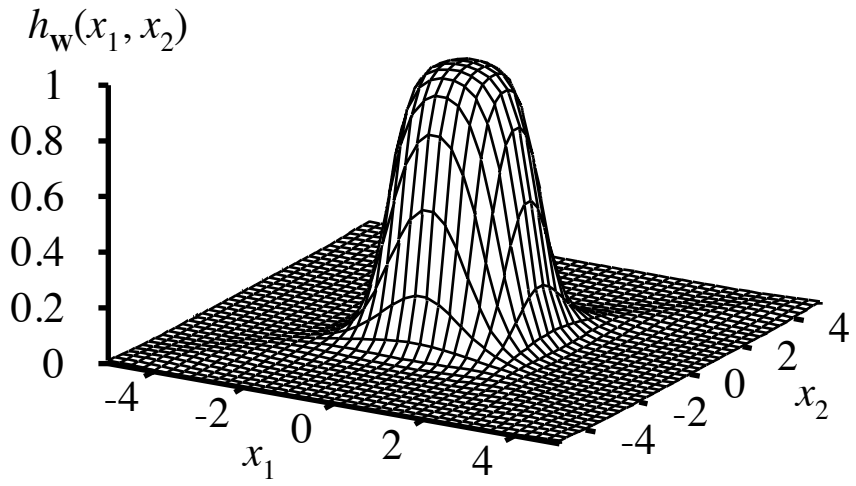
- Output: $a_j = g(in_j)$
- g: *activation function*
  - nondecreasing
  - usually positive, often nonlinear
  - e.g., logistic (sigmoid):
    $g(z) = 1/(1+e^{-z})$

- Simple mathematical model
  - Inspired by neurons, but much simpler
- Output of unit j depends on weighted sum of inputs
  - $\mathbf{w} = \begin{bmatrix} w_{0,j} \\ \vdots \\ w_{n,j} \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix}$
  - $in_j = \sum_i w_{i,j} a_i = \mathbf{w}^\top \mathbf{x}$



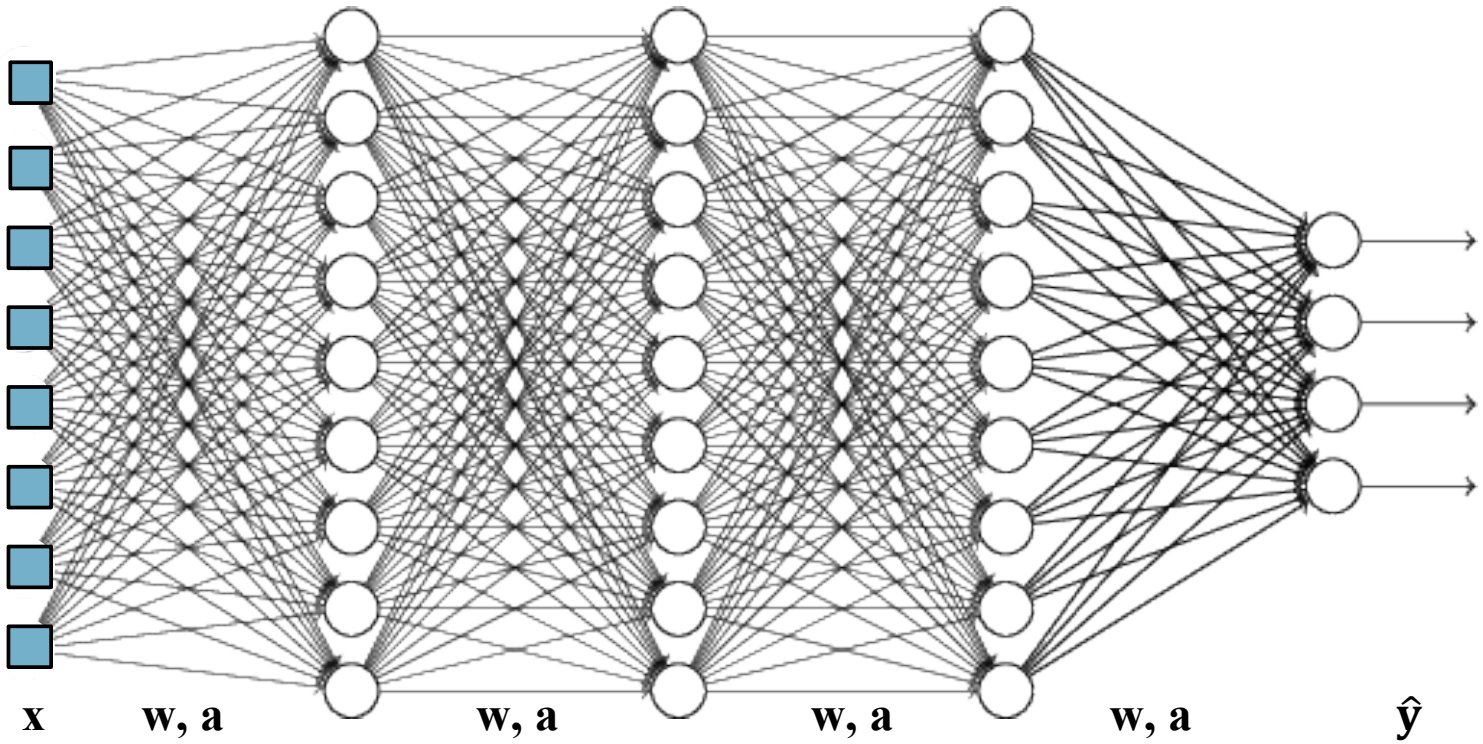
Expressiveness





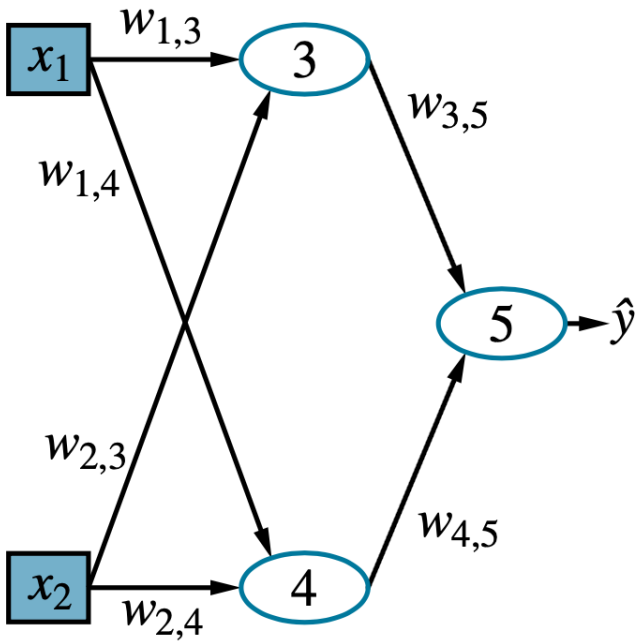
- With 2 layers and enough units in those layers, can approximate all continuous functions
  - with exponentially many units, can get any desired accuracy
- With 3 layers, all functions
- Combine two opposite-facing threshold functions to make a ridge
- Combine two perpendicular ridges to make a bump
- 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 $\mathbf{x} = [x_1, \ldots, x_n]$ Output $\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_k]$

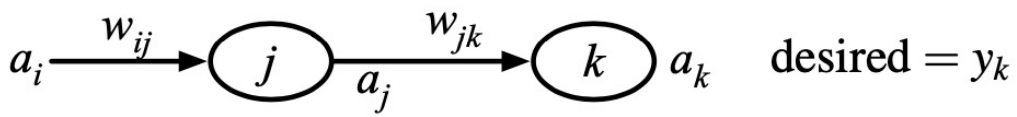
Example



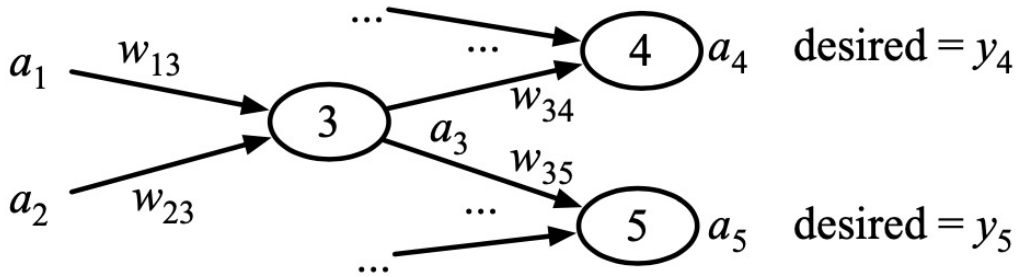
- Let $\mathbf{W}^{(i)}$ = matrix of weights in i'th layer
  - $\mathbf{W}^{(1)} = \begin{bmatrix} w_{1,3} & w_{1,4} \\ w_{2,3} & w_{2,4} \end{bmatrix}$
  - $\mathbf{W}^{(2)} = \begin{bmatrix} w_{3,5} \\ w_{4,5} \end{bmatrix}$
- Let $\mathbf{g}^{(i)}$ = column vector of activation functions
  - $\mathbf{g}^{(1)} = \begin{bmatrix} g_3 \\ g_4 \end{bmatrix}$
  - $\mathbf{g}^{(2)} = [g_5]$
- 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

$$
\begin{aligned}
a_3 &= g_3(w_{1,3}\,x_1 + w_{2,3}\,x_2) \\
a_4 &= g_4(w_{1,4}\,x_1 + w_{2,4}\,x_2) \\
\hat{y} &= g_5(w_{3,5}\,a_3 + w_{4,5}\,a_4) \\
&= g_5[w_{3,5}\,g_3(w_{1,3}\,x_1 + w_{2,3}\,x_2) + w_{4,5}\,g_4(w_{1,4}\,x_1 + w_{2,4}\,x_2)] \\
&= h_{\mathbf{w}}(\mathbf{x})
\end{aligned}
$$

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

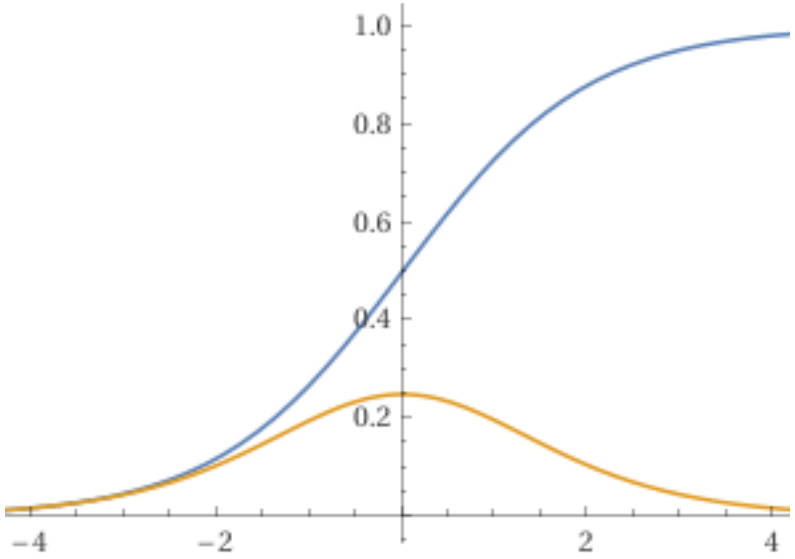
Activation Functions

- logistic (sigmoid):

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

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

- In deep multilayer networks, sigmoid has a *vanishing gradient* problem
  - g' too small to be useful

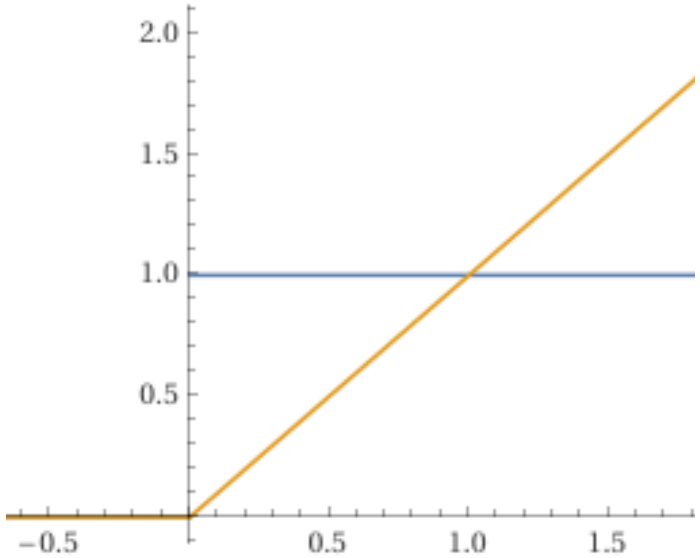


- Rectified linear (ReLU) or softplus are widely used instead

- ReLU:

  $g(z) = \max(0,z)$

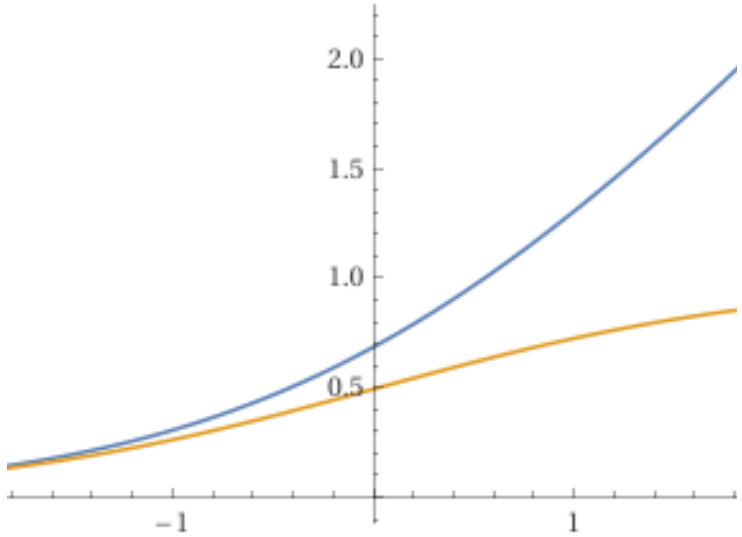
  $g'(z) = 0$ if $z < 0$, 1 if $z > 0$



- softplus:

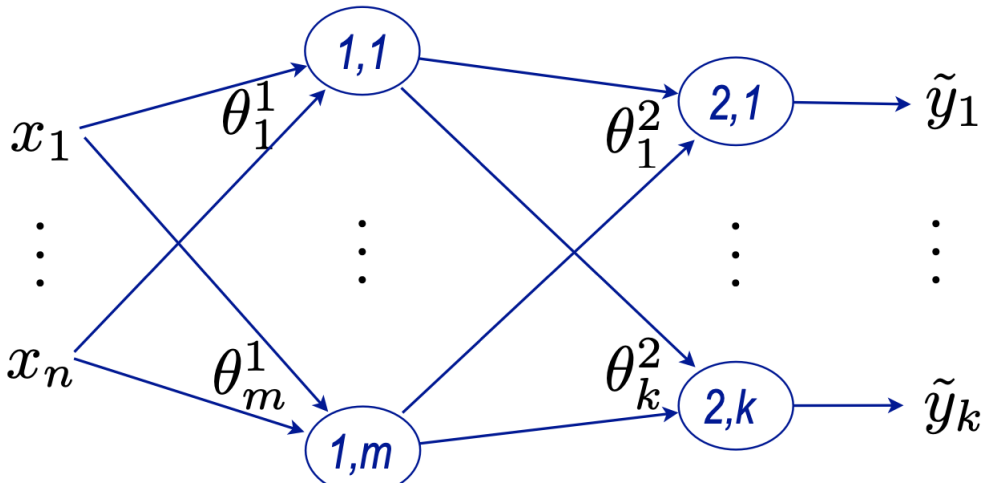
  $g(z) = \log(1+e^{z})$

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



Relation to the Book

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



Backpropagation(x, y)

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

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

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

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

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

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

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

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

Summary

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

