Neural networks and Reinforcement learning review

CS 540

Yingyu Liang

Neural Networks

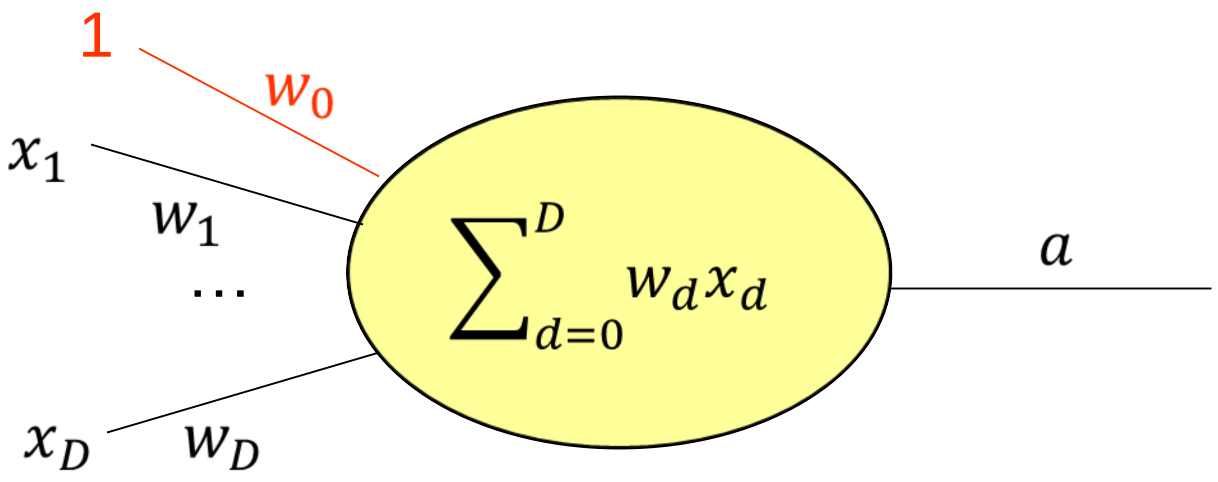
Outline

- Building unit: neuron
  - Linear perceptron
  - Non-linear perceptron
  - The power/limit of a single perceptron
  - Learning of a single perceptron

- Neural network: a network of neurons
  - Layers, hidden units
  - Learning of neural network: backpropagation (gradient descent)

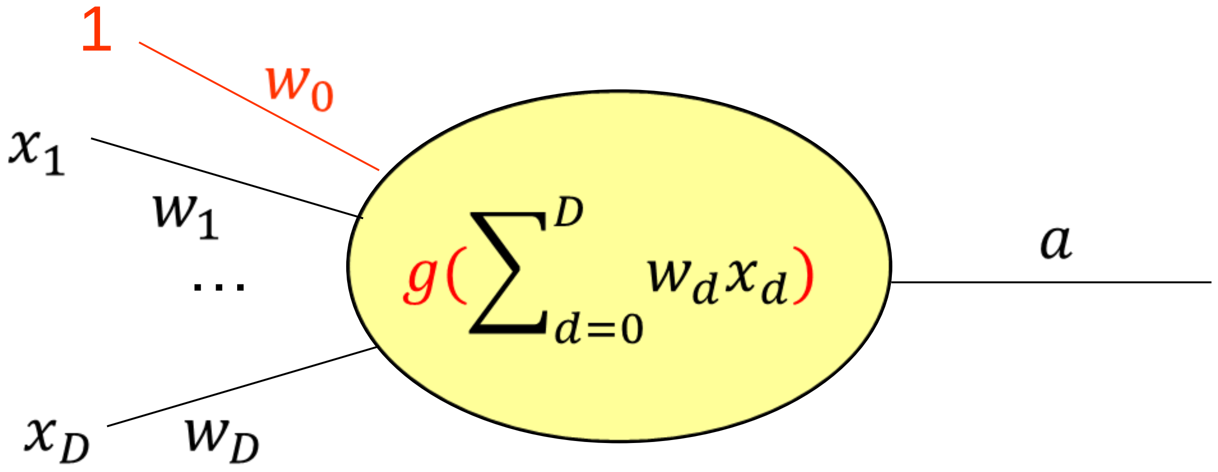
Linear perceptron

- Input: $x_1, x_2, \ldots, x_D$ (For notation simplicity, define $x_0 = 1$)
- Weights: $w_1, w_2, \ldots, w_D$
- Bias: w_0
- Output: $a = \sum_{d=0}^{D} w_d x_d$



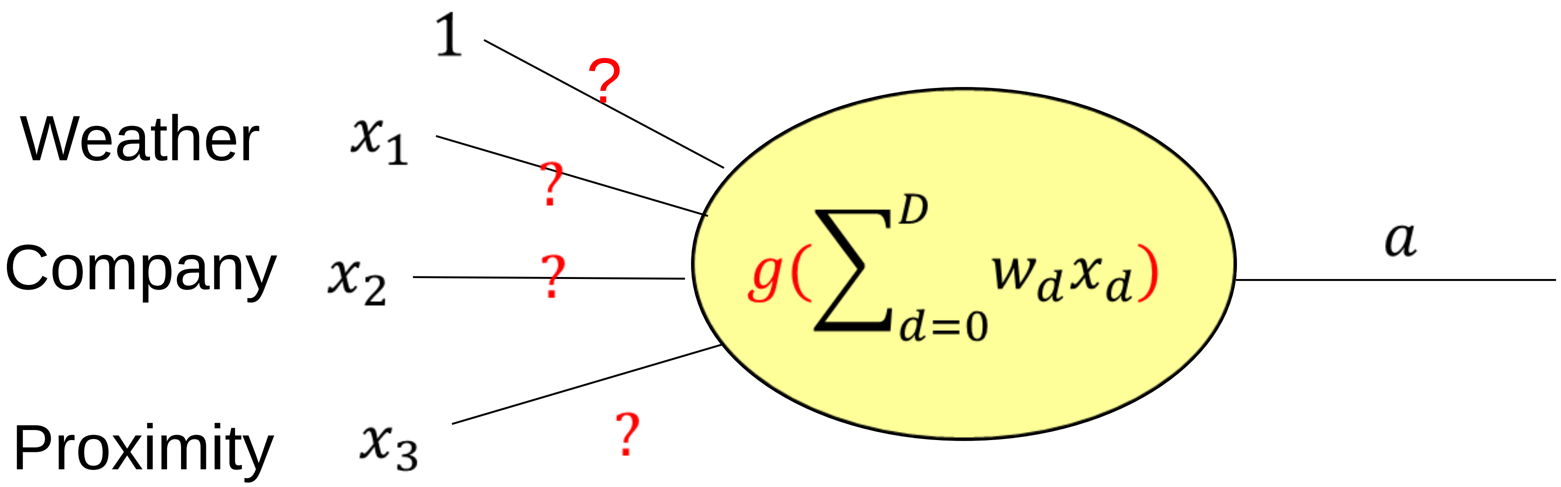
Nonlinear perceptron

- Input: $x_1, x_2, \ldots, x_D$ (For notation simplicity, define $x_0 = 1$)
- Weights: $w_1, w_2, \ldots, w_D$
- Bias: w_0
- Activation function: $g(z) = \mathrm{step}(z), \mathrm{sigmoid}(z), \mathrm{relu}(z), \ldots$
- Output: $a = g(\sum_{d=0}^{D} w_d x_d)$



Example Question

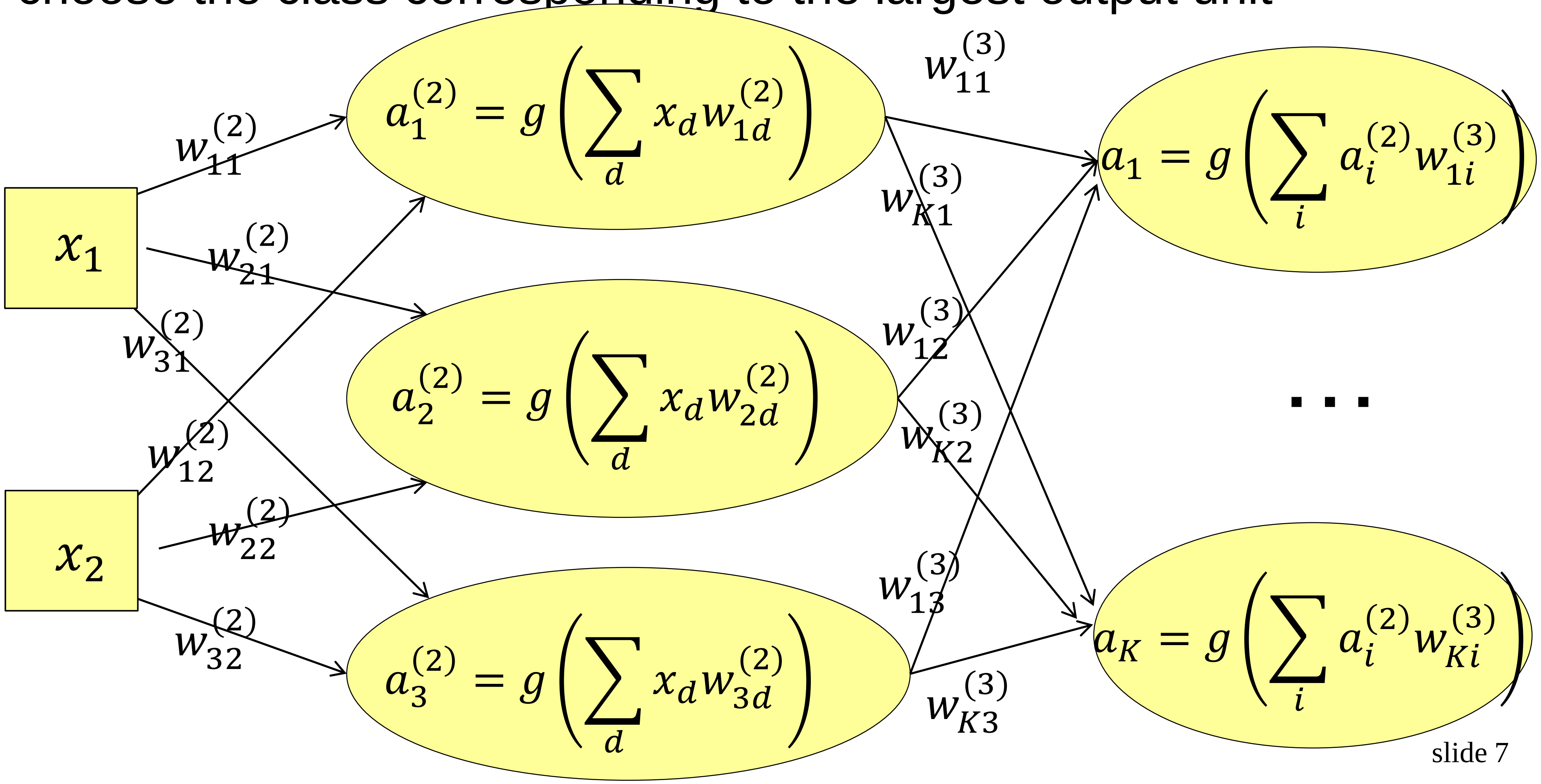
- Will you go to the festival?
- Go only if Weather is favorable and at least one of the other two conditions is favorable



All inputs are binary; 1 is favorable

Multi-layer neural networks

- Training: encode a label y by an indicator vector
  - class1=(1,0,0,…,0), class2=(0,1,0,…,0) etc.
- Test: choose the class corresponding to the largest output unit

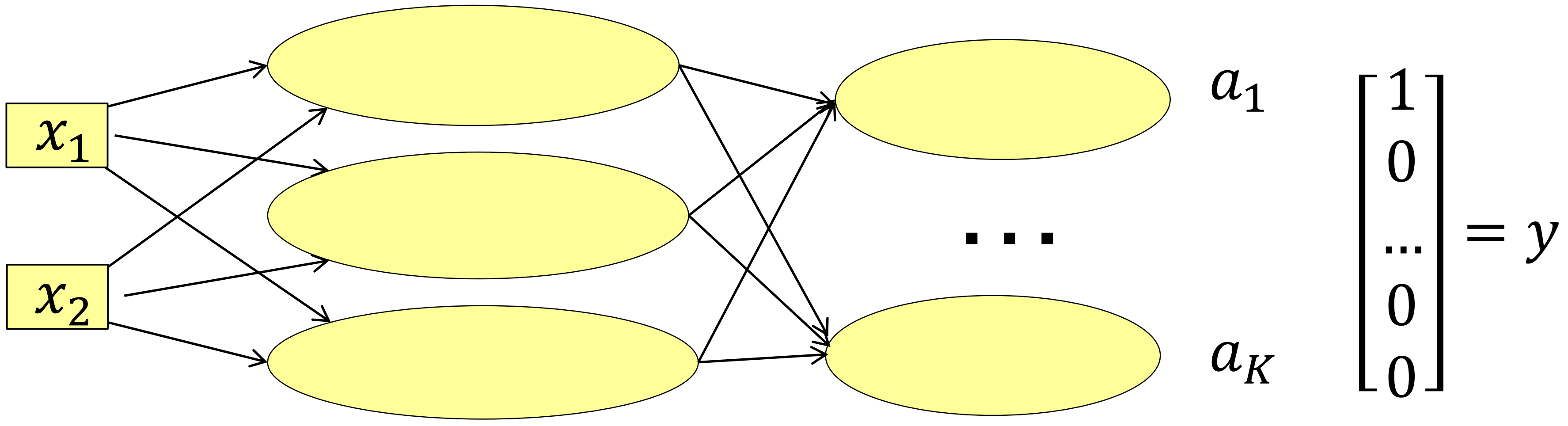


Learning in neural network

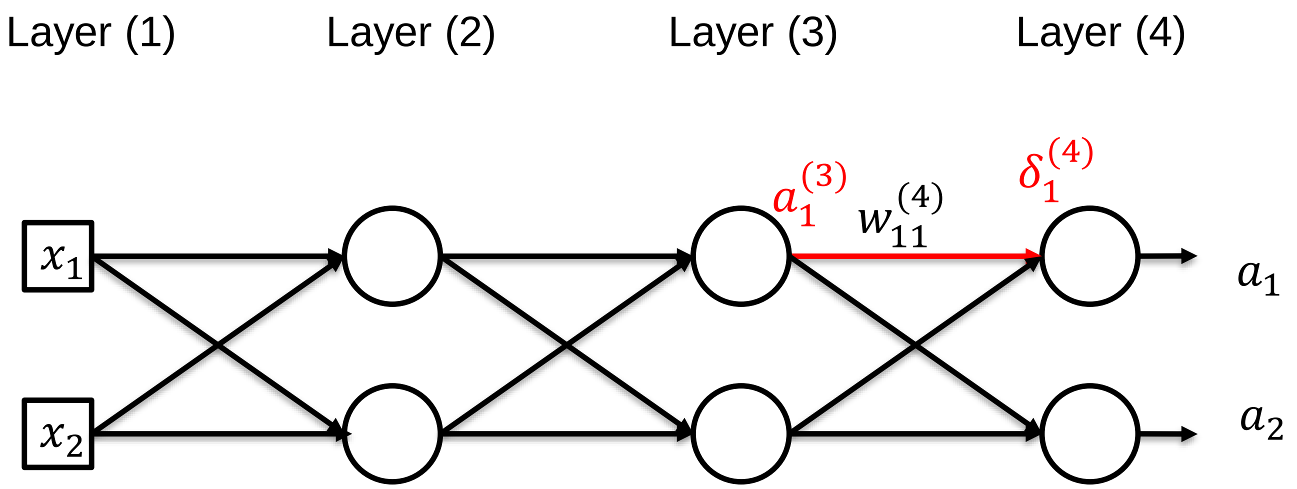
- Again we will minimize the error (K outputs):

$$E = \frac{1}{2}\sum_{x \in D} E_x , \qquad E_x = \|y - a\|^2 = \sum_{c=1}^{K} (a_c - y_c)^2$$

- x: one training point in the training set D
- a_c: the c-th output for the training point x
- y_c: the c-th element of the label indicator vector for x



Backpropagation



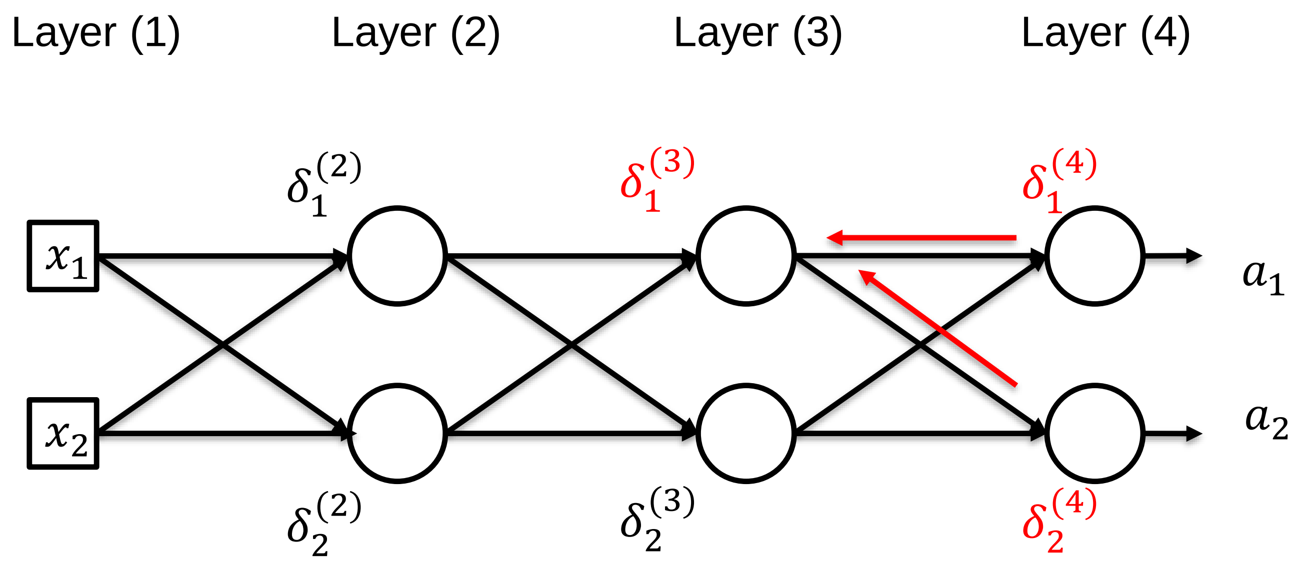
$$E_x = \|y - a\|^2$$

$$\left.\begin{array}{l} w_{11}^{(4)} a_1^{(3)} \\ w_{12}^{(4)} a_2^{(3)} \end{array}\right\} \xrightarrow{+} z_1^{(4)} \longrightarrow E_x$$

$$\delta_1^{(4)} = \frac{\partial E_x}{\partial z_1^{(4)}} = 2(a_1 - y_1)g'\left(z_1^{(4)}\right)$$

By Chain Rule: $$\frac{\partial E_x}{\partial w_{11}^{(4)}} = \delta_1^{(4)} a_1^{(3)}$$

Backpropagation of δ



$$E_x = \|y - a\|^2$$

Thus, for any neuron in the network:

$$\boxed{\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} \quad g'\left(z_j^{(l)}\right)}$$

$\delta_j^{(l)}$: δ of j^{th} Neuron in Layer l

$\delta_k^{(l+1)}$: δ of k^{th} Neuron in Layer $l+1$

$g'\left(z_j^{(l)}\right)$: derivative of j^{th} Neuron in Layer l w.r.t. its linear combination input

$w_{kj}^{(l+1)}$: Weight from j^{th} Neuron in Layer l to k^{th} Neuron in Layer $l+1$

Example Question

15. (Gradient descent) Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. We want to minimize the objective function $f(\mathbf{x}) = \sum_{i=1}^d ix_i = x_1 + 2x_2 + \ldots + dx_d$ using gradient descent. Let the stepsize $\eta = 0.1$. If we start at the all-zero vector $\mathbf{x}^{(0)} = (0, \ldots, 0)$, what is the next vector $\mathbf{x}^{(1)}$ produced by gradient descent?

Example Question

15. (Gradient descent) Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. We want to minimize the objective function $f(\mathbf{x}) = \sum_{i=1}^d ix_i = x_1 + 2x_2 + \ldots + dx_d$ using gradient descent. Let the stepsize $\eta = 0.1$. If we start at the all-zero vector $\mathbf{x}^{(0)} = (0, \ldots, 0)$, what is the next vector $\mathbf{x}^{(1)}$ produced by gradient descent?

A: $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \eta \nabla f(\mathbf{x})$, and $\nabla f(\mathbf{x}) = (1, 2, \ldots, d)$. So $(-0.1, -0.2, \ldots, -0.1 \cdot d)$.

Example Question

23. (ReLU) Consider a rectified linear unit with input $x \in \mathbb{R}$ and a bias term. The output can be written as $y = \max(0, w_0 + w_1 x)$. Write down the input value x that produces a specific output $y > 0$.

Example Question

23. (ReLU) Consider a rectified linear unit with input $x \in \mathbb{R}$ and a bias term. The output can be written as $y = \max(0, w_0 + w_1 x)$. Write down the input value x that produces a specific output $y > 0$.

A: since $y > 0, y = w_0 + w_1 * x$. Thus $x = (y - w_0)/w_1$. However, we will also accept the interpretation that we asked for the range $y > 0$. In that case you must show all four branches:

$$\begin{cases} x > -w_0/w_1, & w_1 > 0 \\ x < -w_0/w_1, & w_1 < 0 \\ x \in \mathbb{R}, & w_1 = 0, w_0 > 0 \\ \emptyset, & w_1 = 0, w_0 \leq 0 \end{cases}$$

Convolution: discrete version

- Given array u_t and w_t, their convolution is a function s_t

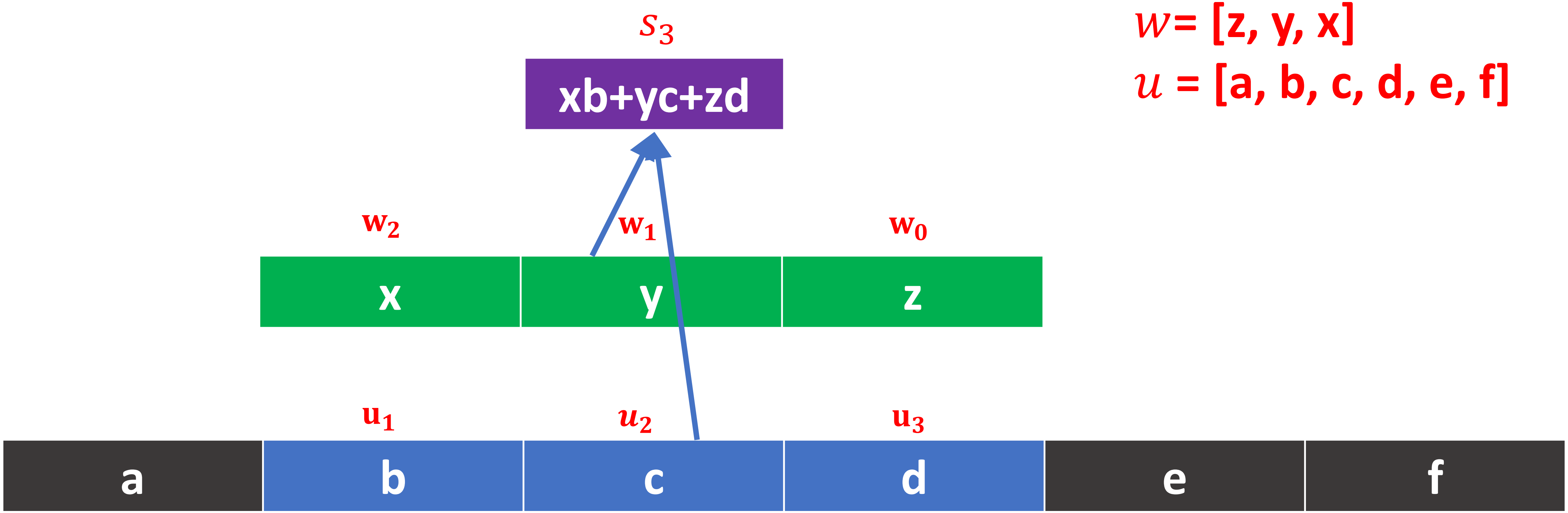
$$s_t = \sum_{a=-\infty}^{+\infty} u_a w_{t-a}$$

- Written as

$$s = (u * w) \quad \text{or} \quad s_t = (u * w)_t$$

- When u_t or w_t is not defined, assumed to be 0

Convolution illustration

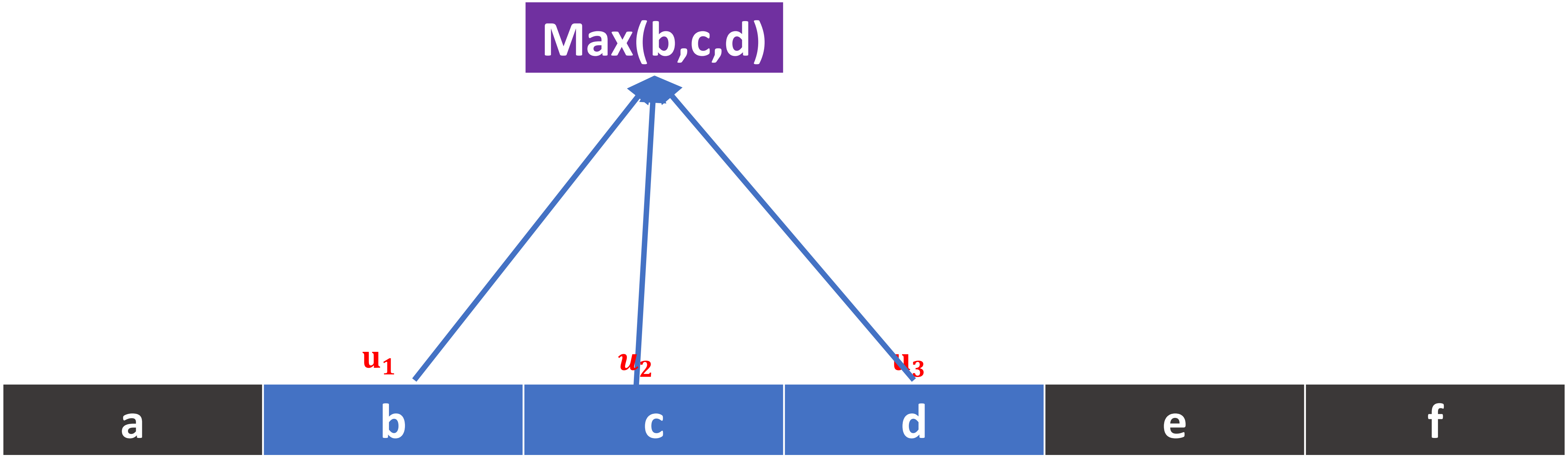


w= [z, y, x]

u = [a, b, c, d, e, f]

Pooling illustration

u = [a, b, c, d, e, f]



Example question

What is the value $s = (u * w)$? (Valid padding)

w= [-1,1,1]
u = [1,2,3,4,5,6]

w_2	w_1	w_0
1	1	-1

1	2	3	4	5	6

Reinforcement Learning

Outline

- The reinforcement learning task
- Markov decision process
- Value functions
- Value iteration
- Q functions
- Q learning

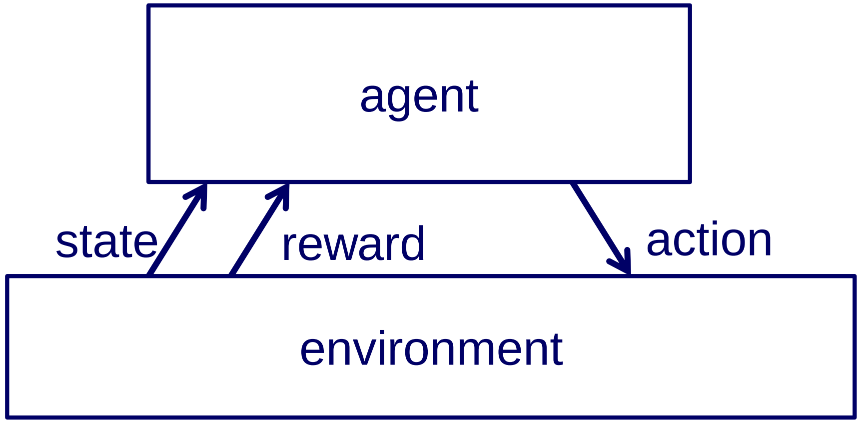
Reinforcement learning as a Markov decision process (MDP)

- Markov assumption

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} \mid s_t, a_t)$$

- also assume reward is Markovian

$$P(r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(r_{t+1} \mid s_t, a_t)$$



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2}$$

Goal: learn a policy $\pi : S \rightarrow A$ for choosing actions that maximizes

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \quad \text{where } 0 \leq \gamma < 1$$

for every possible starting state s_0

Value function for a policy

- given a policy $\pi : S \rightarrow A$ define

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^{t} E[r_t]$$

assuming action sequence chosen according to π starting at state s

- we want the optimal policy π^* where

$$p^* = \arg\max_p V^p(s) \quad \text{for all } s$$

we'll denote the value function for this optimal policy as $V^*(s)$

Value iteration for learning $V^*(s)$

initialize $V(s)$ arbitrarily

loop until policy good enough

{

    loop for $s \in S$

    {

        loop for $a \in A$

        {

$$Q(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$$

        }

$$V(s) \leftarrow \max_a Q(s,a)$$

    }

}

Q function

define a new function, closely related to V^*

$$Q(s,a) \leftarrow E[r(s,a)] + \gamma E_{s'|s,a}[V^*(s')]$$

if agent knows $Q(s, a)$, it can choose optimal action without knowing $P(s' \mid s, a)$

$$\pi^*(s) \leftarrow \arg\max_a Q(s,a) \qquad V^*(s) \leftarrow \max_a Q(s,a)$$

and it can learn $Q(s, a)$ without knowing $P(s' \mid s, a)$

Q learning for deterministic worlds

for each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$

observe current state s

do forever

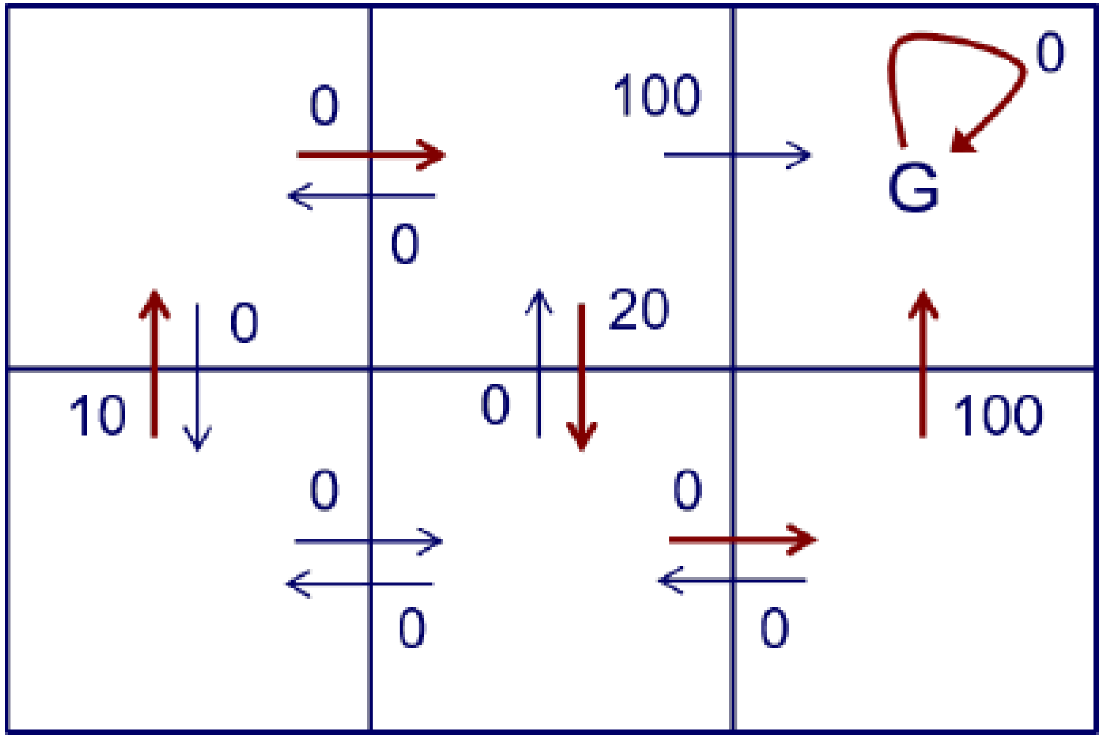
- select an action a and execute it
- receive immediate reward r
- observe the new state s'
- update table entry

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

- $s \leftarrow s'$

Example question

Suppose a policy π is shown by red arrows, the discount factor $\gamma = 0.9$. Compute the value function $V^{\pi}(s)$ for all states s.



Example question