



# CS 540 Introduction to Artificial Intelligence **Review**

University of Wisconsin-Madison  
Fall 2025 Sections 1 & 2

# Announcements: Final Information

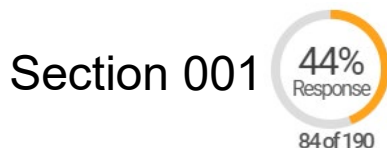
- **Time: December 13th 12:25 PM - 2:25 PM**
- **Location for sections 001 AND 002 (Instructor Blerina Gkotse)**  
**based on your last name:**
  - Chemistry S413 (Last name begins with A-J inclusive)
  - Engineering Hall 1800 (Last name begins with K-Sa inclusive)
  - Van Vleck B130 (Last name begins with Sc-Z inclusive)
- Location for **section 003 (Instructor Gavin Brown)**: Chemistry S429
- Students with McBurney accommodations or alternate requests should have received an email with additional information.

# Announcements: Final Information

- **Topics:** The exam is **cumulative**; everything covered in the slides or homework.
- **Practice questions:** Canvas -> Files -> Practice Questions
- **Format:** MCQ.
- **Cheat Sheet:** You will be allowed a handwritten note sheet of a single piece of paper (8.5" x 11", front and back).
- **Calculator:** Calculators are allowed if they don't have an internet connection. A calculator will not be necessary though it may be useful to double check simple arithmetic.
- **Bring:** your WISC ID, pencil (No 2 or softer) and your cheat sheet.

# Announcements

- **Homework:**
  - HW10 due today at 11:59PM
- **Course evaluation until December 10:**
  - If participation reaches 50%, we'll reveal more information for the final.
  - If participation reaches 70%, even more.





NLP

# Language Models

- Basic idea: use probabilistic models to **assign a probability to a sentence  $W$**

$$P(W) = P(w_1, w_2, \dots, w_n) \text{ or } P(w_{\text{next}} | w_1, w_2 \dots)$$

- Goes back to Shannon
  - Information theory: letters

Zero-order approximation	XFOML RXKHRJFFJUJ ALPWXFJWXYJ FFJEYVJCQSGHYD QPAAMKBZAACIBZLKJQD
First-order approximation	OCRO HLO RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL
Second-order approximation	ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE
Third-order approximation	IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE
First-order word approximation	REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE

# Training: Make Assumptions

- Markov assumption with shorter history:

$$P(w_i | w_{i-1} w_{i-2} \dots w_1) = P(w_i | w_{i-1} w_{i-2} \dots w_{i-k})$$

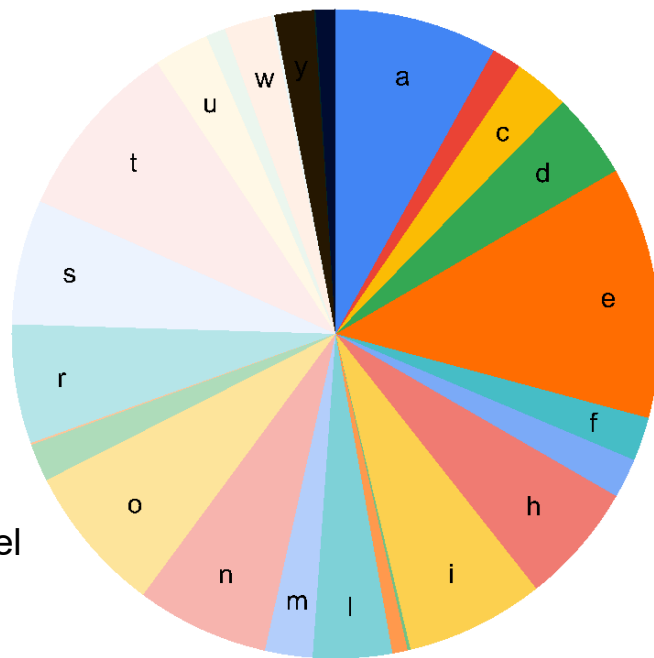
- Present doesn't depend on whole past
  - Just recent past, i.e., *context*.
  - What's ***k=0?***

# k=0: **Unigram** Model

- Full independence assumption:
  - (Present doesn't depend on the past)

$$P(w_1, w_2, \dots, w_n) = P(w_1)P(w_2) \dots P(w_n)$$

The English letter frequency wheel





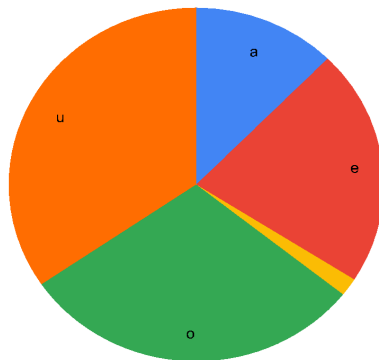
# k=1: **Bigram Model**

- Markov Assumption:
  - (Present depends on immediate past)

$$P(w_1, w_2, \dots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2) \dots P(w_n|w_{n-1})$$



$p(.|q)$ : the “after q” wheel



$p(.|j)$ : the “after j” wheel

texaco, rose, one, in, this, issue,  
is, pursuing, growth, in, a, boiler,  
house, said, mr., gurria, mexico, 's,  
motion, control, proposal, without,  
permission, from, five, hundred,  
fifty, five, yen outside, new, car,  
parking, lot, of, the, agreement,  
reached this, would, be, a, record,  
november

## k=n-1: **n**-gram Model

Can do trigrams, 4-grams, and so on

- More expressive as  $n$  goes up
- Harder to estimate

Training: just count? I.e, for bigram:

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

# n-gram Training

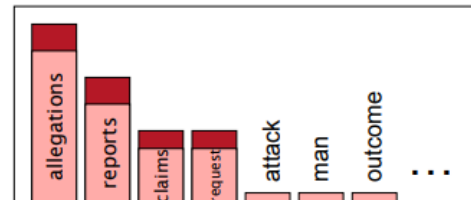
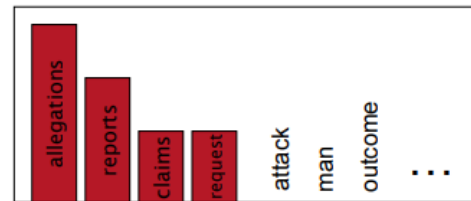
Issues:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

- **1.** Multiply tiny numbers?
  - **Solution:** use logs; add instead of multiply
- **2.** n-grams with zero probability?
  - **Solution:** (Laplace) smoothing

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + V}$$

*P(w|denied the)*



# Break & Quiz

**Q 1.2:** Smoothing is increasingly useful for n-grams when

- A.  $n$  gets larger
- B.  $n$  gets smaller
- C. always the same
- D.  $n$  larger than 10

# Break & Quiz

**Q 1.2:** Smoothing is increasingly useful for n-grams when

- **A. n gets larger**
- B. n gets smaller
- C. always the same
- D. n larger than 10



# Neural Networks

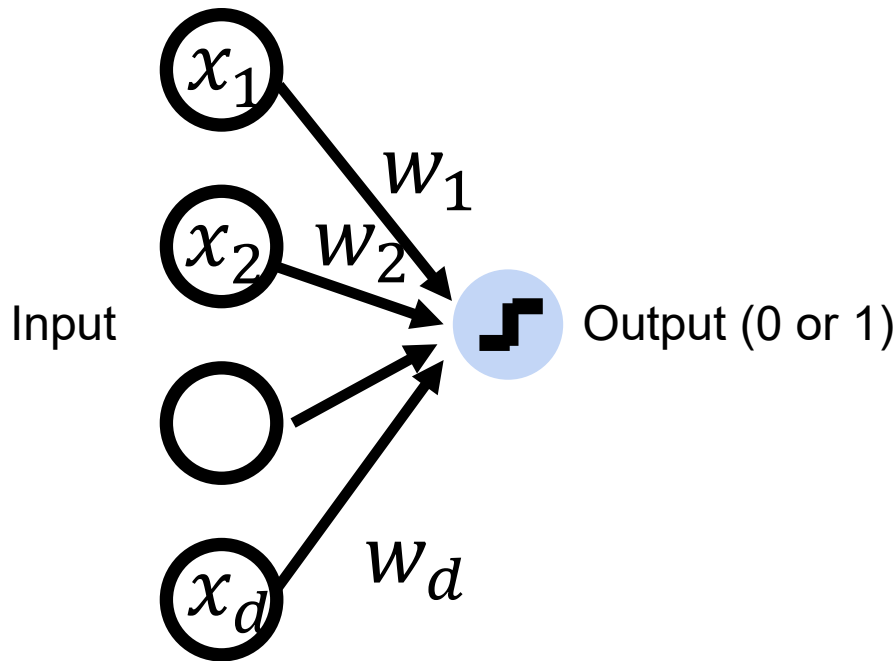
# Perceptron

- Given input  $\mathbf{x}$ , weight  $\mathbf{w}$  and bias  $b$ , perceptron outputs:

$$o = \sigma(\mathbf{w}^T \mathbf{x} + b) \quad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

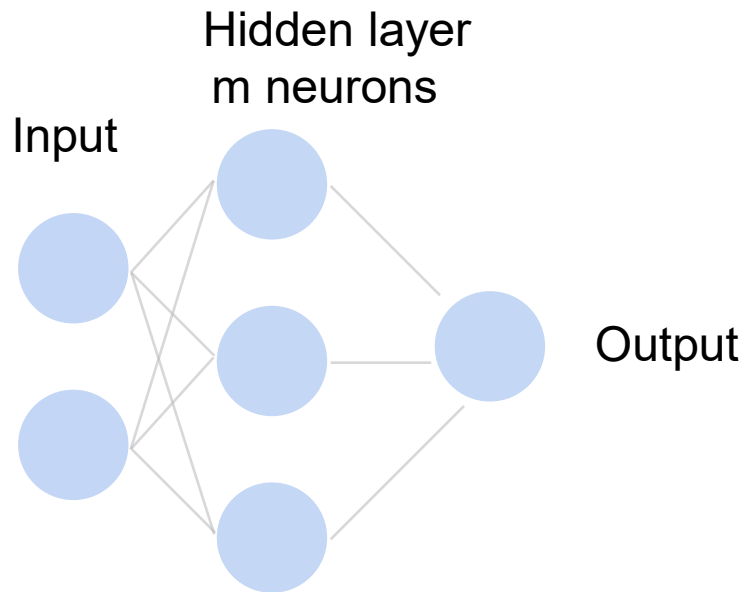
Activation function

Cats vs. dogs?



# Single Hidden Layer

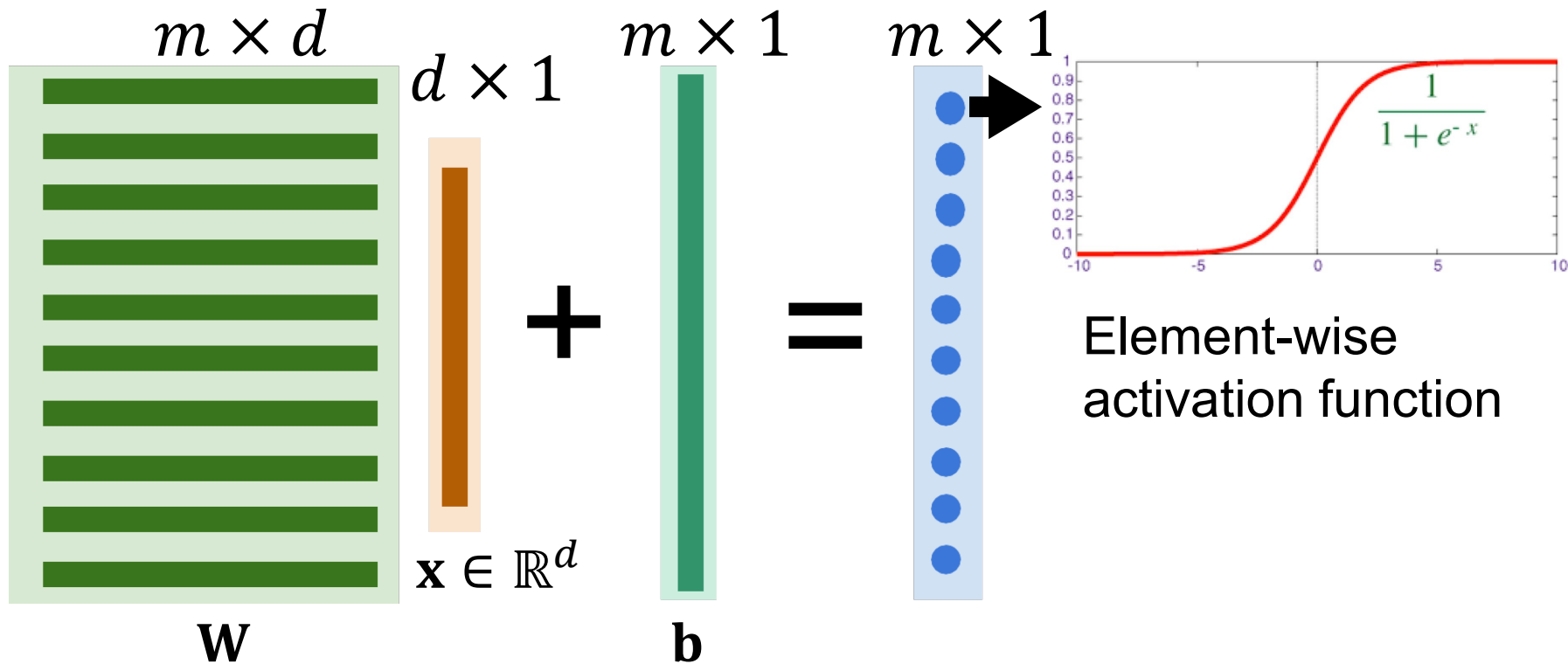
**How to classify  
Cats vs. dogs?**





# Neural networks with one hidden layer

**Key elements:** linear operations + Nonlinear activations

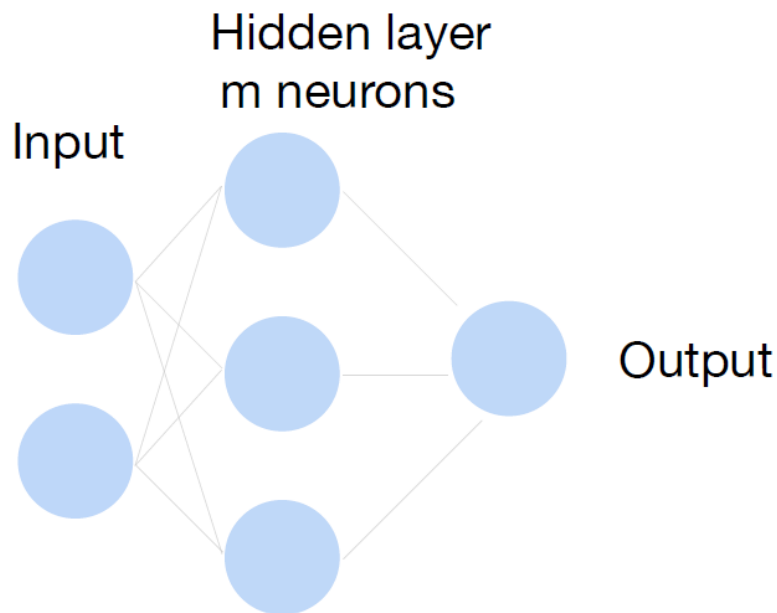
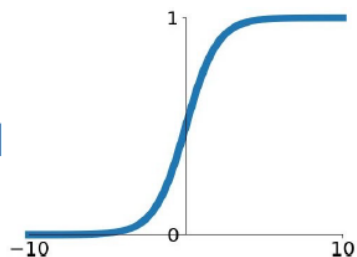


# Single Hidden Layer

- Output  $f = \mathbf{w}_2^\top \mathbf{h} + b_2$
- Normalize the output into probability using sigmoid

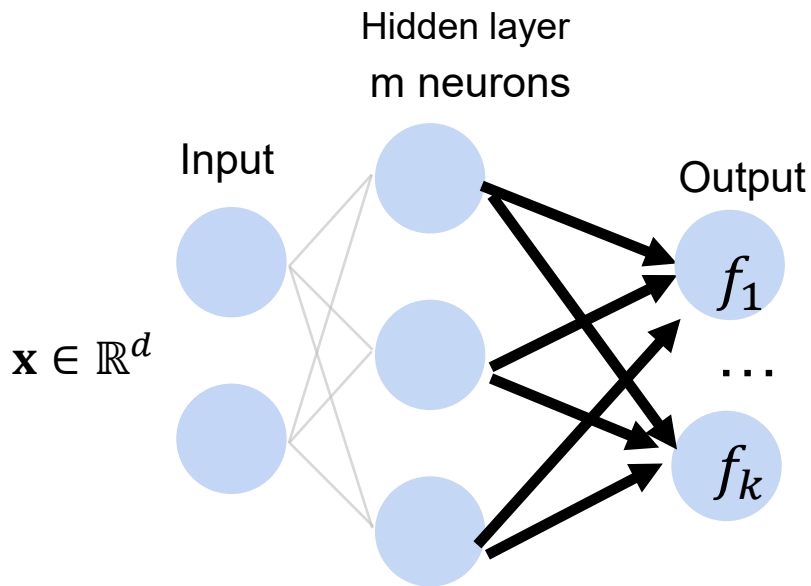
$$p(y = 1 | \mathbf{x}) = \frac{1}{1 + e^{-f}}$$

**Sigmoid**



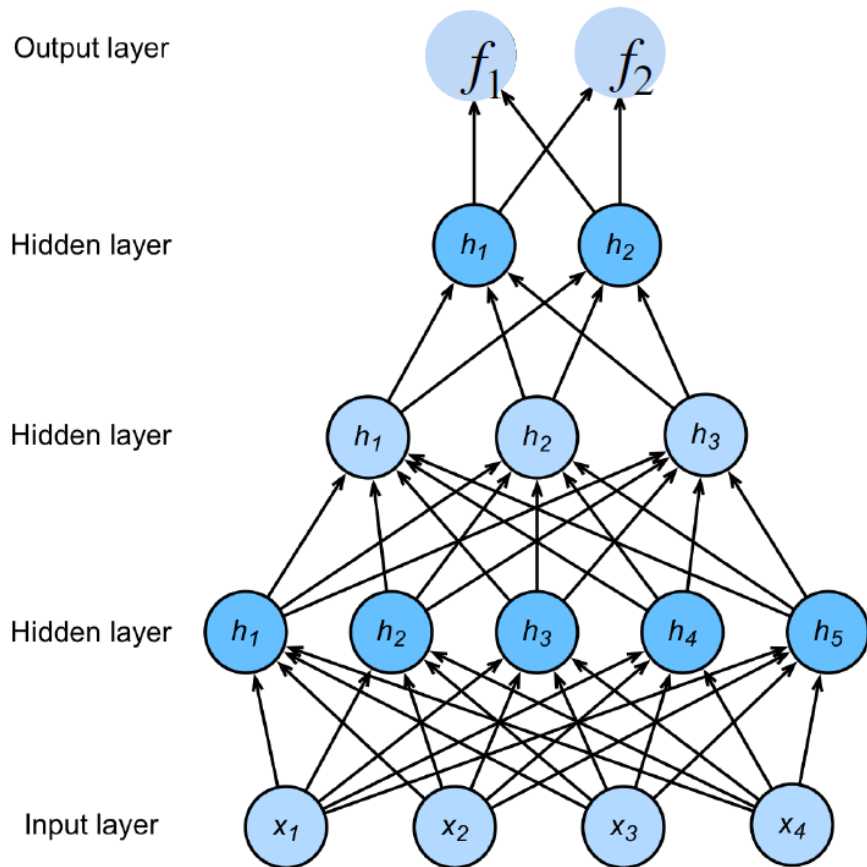
# Multi-class classification

Turns outputs  $f$  into  $k$  probabilities (sum up to 1 across  $k$  classes)



$$\begin{aligned} p(y|\mathbf{x}) &= \textit{softmax}(\mathbf{f}) \\ &= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)} \end{aligned}$$

# Deep neural networks (DNNs)



$$\mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3)$$

$$\mathbf{f} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$

$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

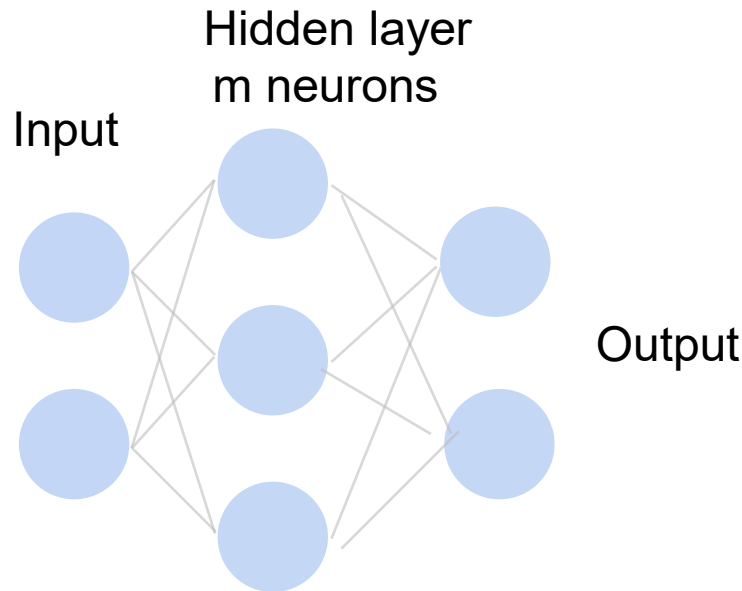
NNs are composition  
of nonlinear  
functions

# How to train a neural network?

Update the weights  $W$  to minimize the loss function

$$L = \frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$$

**Use gradient descent!**



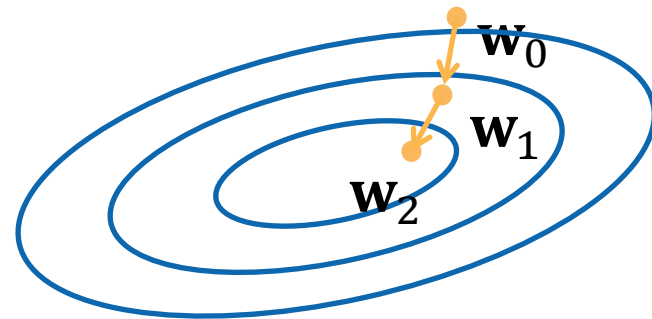
# Gradient Descent

- Choose a learning rate  $\eta > 0$
- Initialize the model parameters  $w_0$
- For  $t = 1, 2, \dots$ 
  - Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{\partial L}{\partial \mathbf{w}_{t-1}}$$

$$= \mathbf{w}_{t-1} - \eta \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

- Repeat until converges

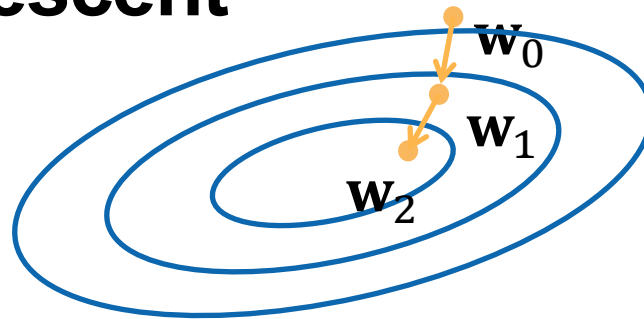


D can be very large. Expensive per iteration

The gradient w.r.t. all parameters is obtained by concatenating the partial derivatives w.r.t. each parameter

# Minibatch Stochastic Gradient Descent

- Choose a learning rate  $\eta > 0$
- Initialize the model parameters  $w_0$
- For  $t = 1, 2, \dots$



- **Randomly sample a subset (mini-batch)  $B \subset D$**
- Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{1}{|B|} \sum_{(\mathbf{x}, y) \in B} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

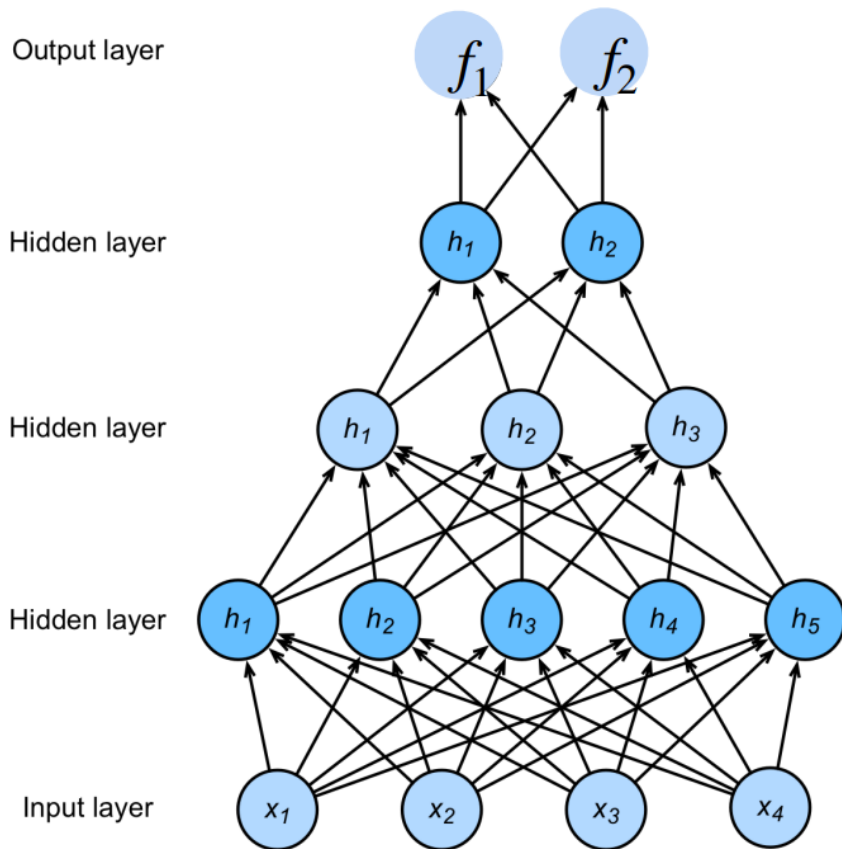
- Repeat until converges



# Neural Networks as a Computational Graph



# Deep neural networks (DNNs)



$$\mathbf{h}_1 = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)})$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$

$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

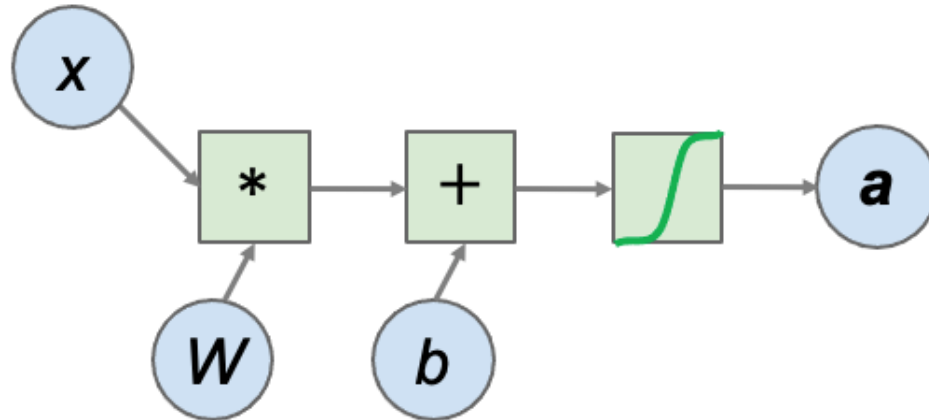
$$\mathbf{p} = \text{softmax}(\mathbf{f})$$

NNs are composition  
of nonlinear  
functions

# Neural networks as variables + operations

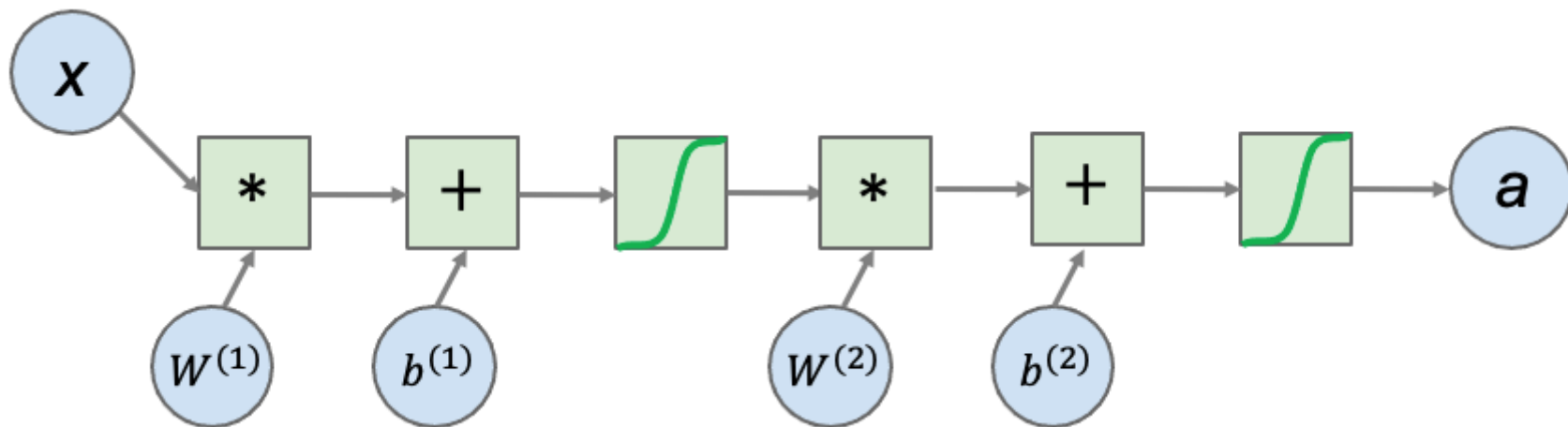
$$\mathbf{a} = \textit{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Can describe with a **computational graph**
- Decompose functions into atomic operations
- Separate data (**variables**) and computing (**operations**)



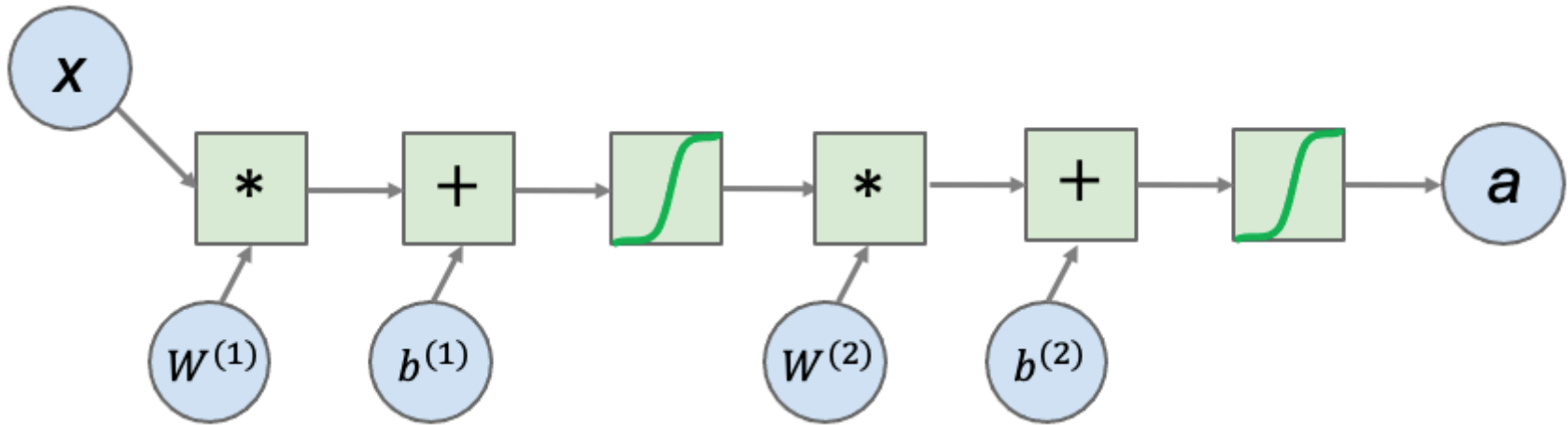
# Neural networks as a computational graph

- A two-layer neural network



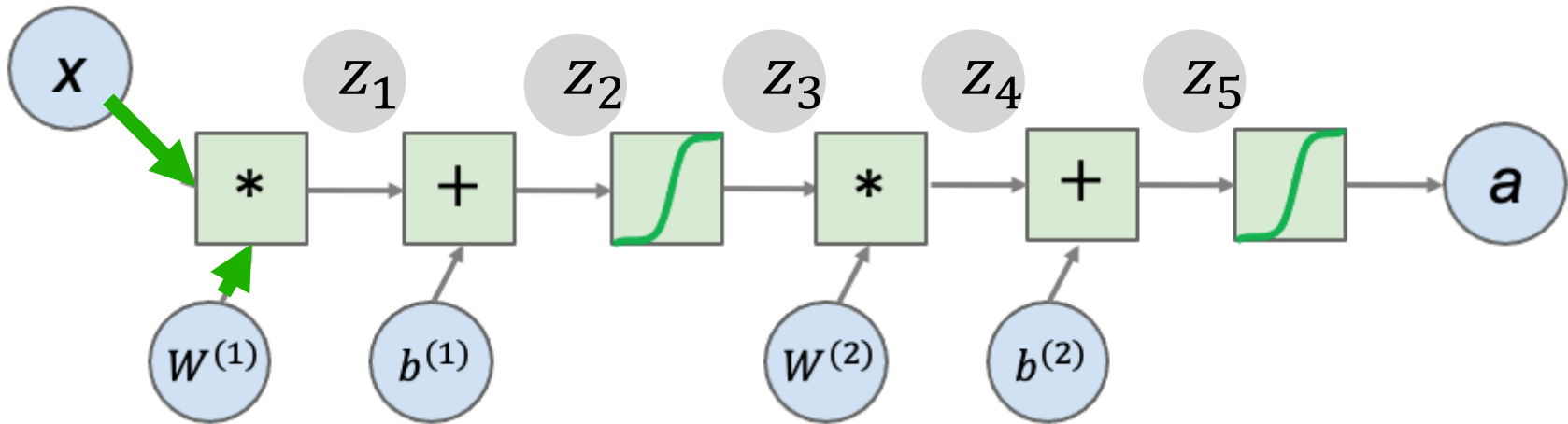
# Neural networks as a computational graph

- A two-layer neural network
- Forward propagation vs. backward propagation



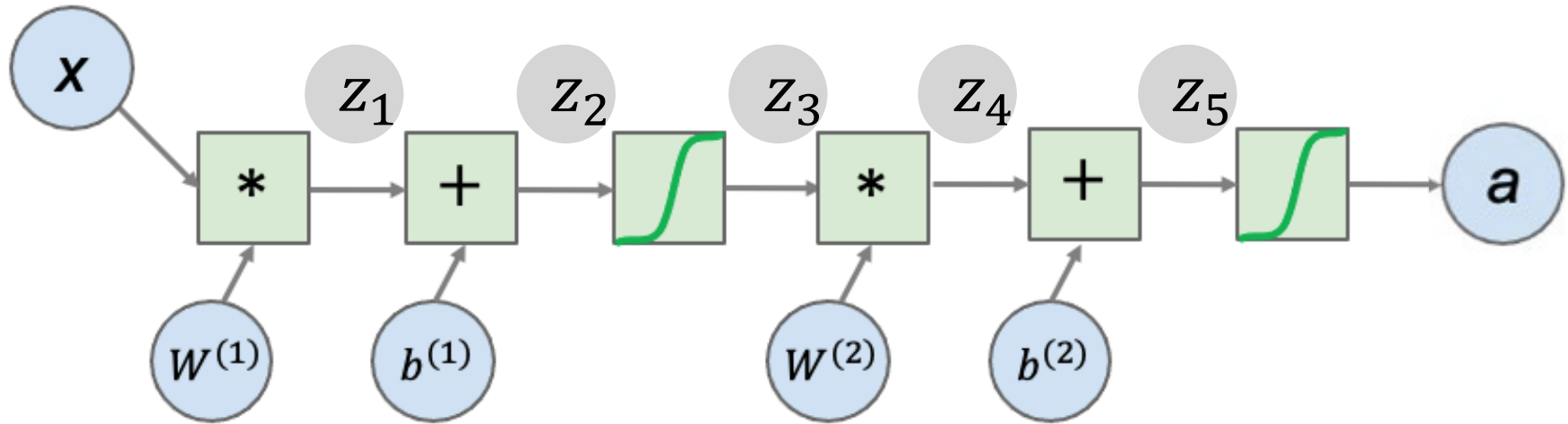
# Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables  $Z$



# Neural networks: backward propagation

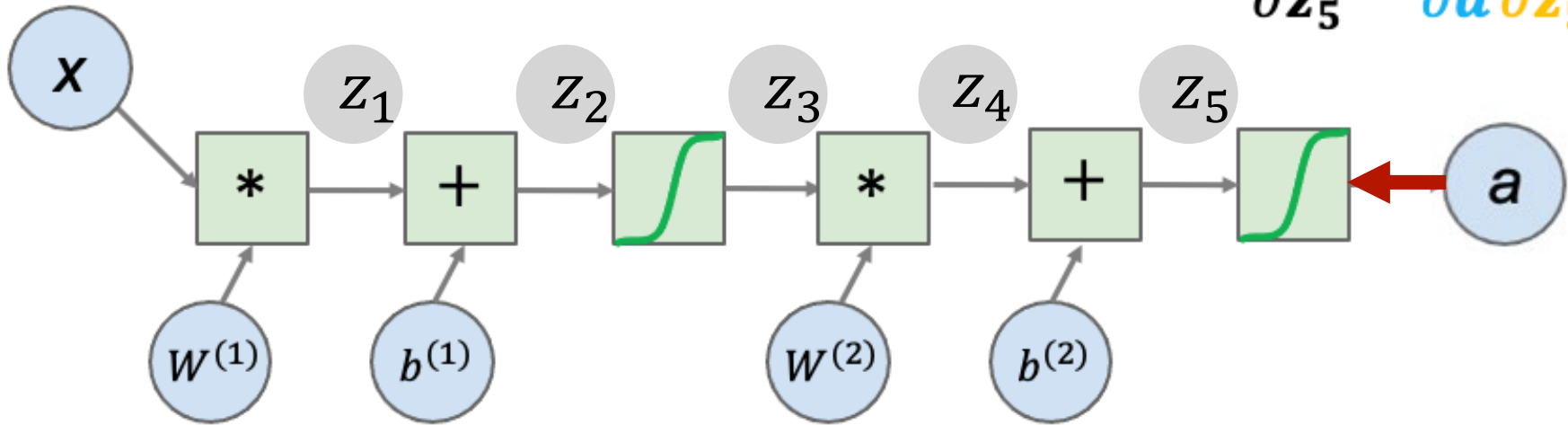
- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function**  $L$



# Neural networks: backward propagation

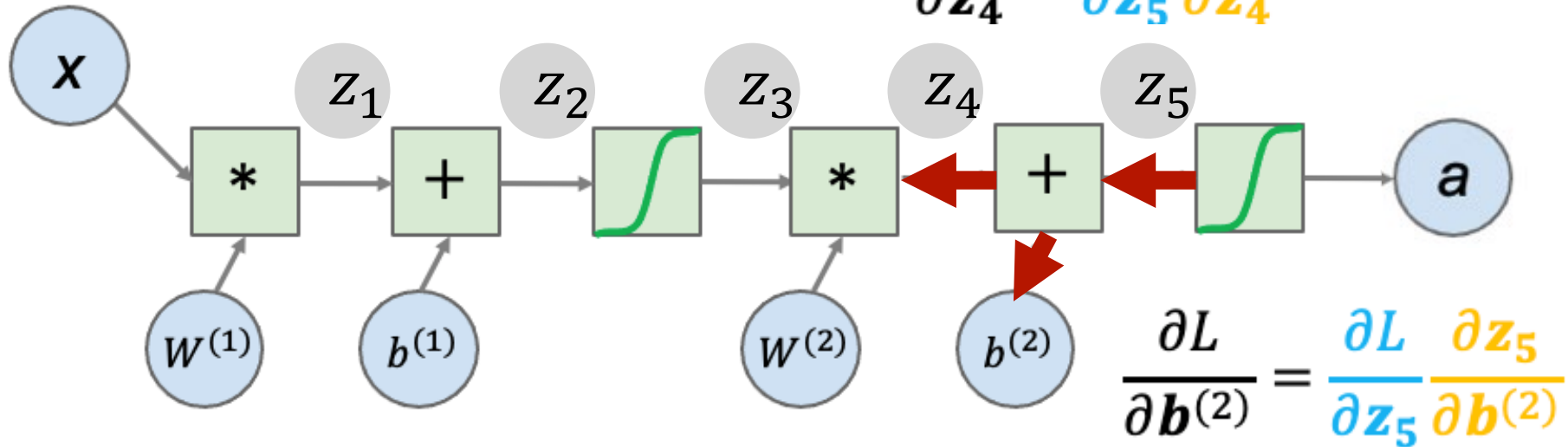
- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function**  $L$

$$\frac{\partial L}{\partial \mathbf{z}_5} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{z}_5}$$



# Neural networks: backward propagation

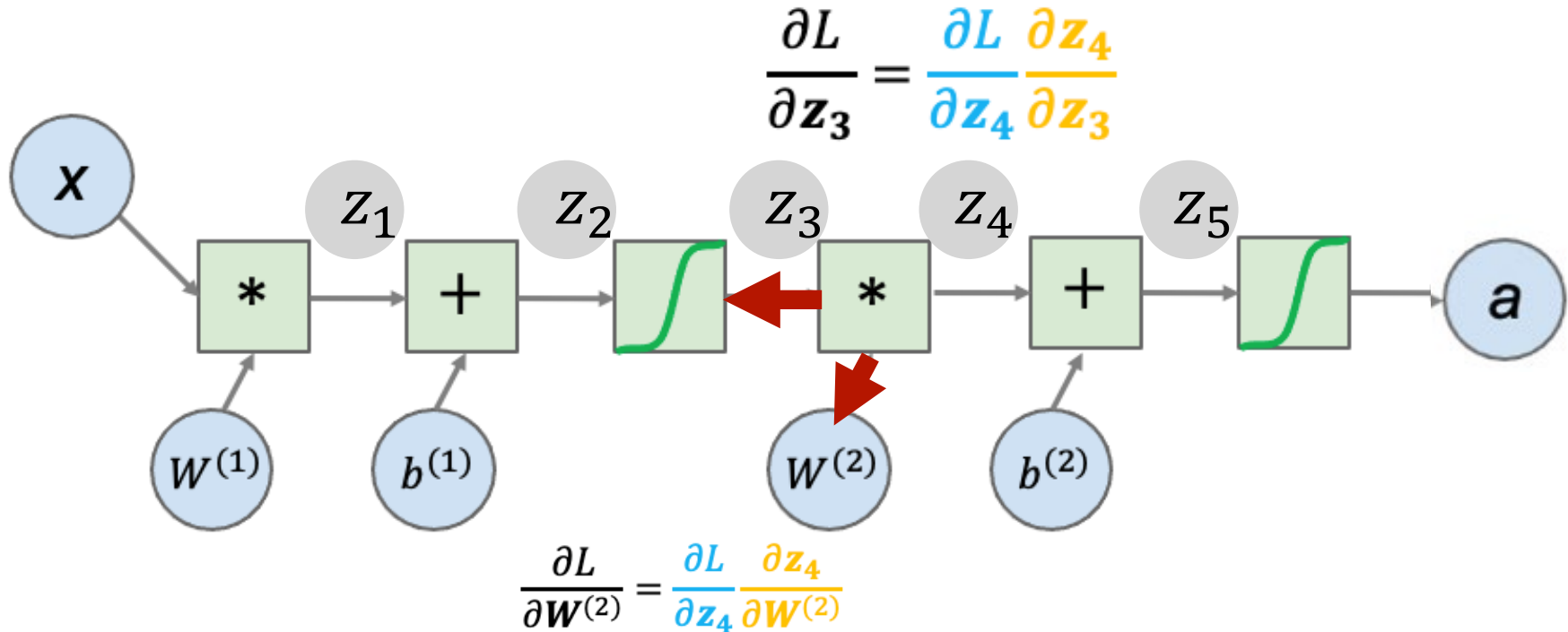
- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function**  $L$





# Neural networks: backward propagation

- A two-layer neural network
- Assuming forward propagation is done





# Numerical Stability

# Gradients for Neural Networks

- Compute the gradient of the loss  $\ell$  w.r.t.  $\mathbf{W}_t$

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}}_{\text{Multiplication of many matrices}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

Multiplication of *many*  
matrices



Wikipedia

# Two Issues for Deep Neural Networks

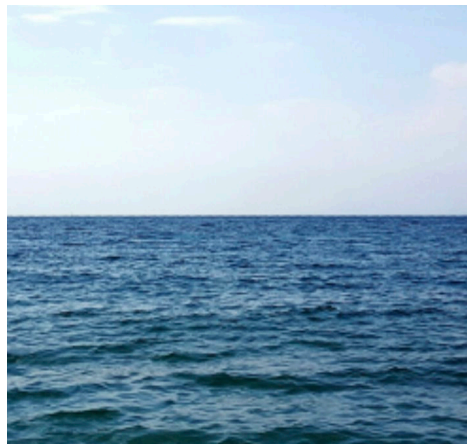
$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}$$

Gradient Exploding



$$1.5^{100} \approx 4 \times 10^{17}$$

Gradient Vanishing



$$0.8^{100} \approx 2 \times 10^{-10}$$

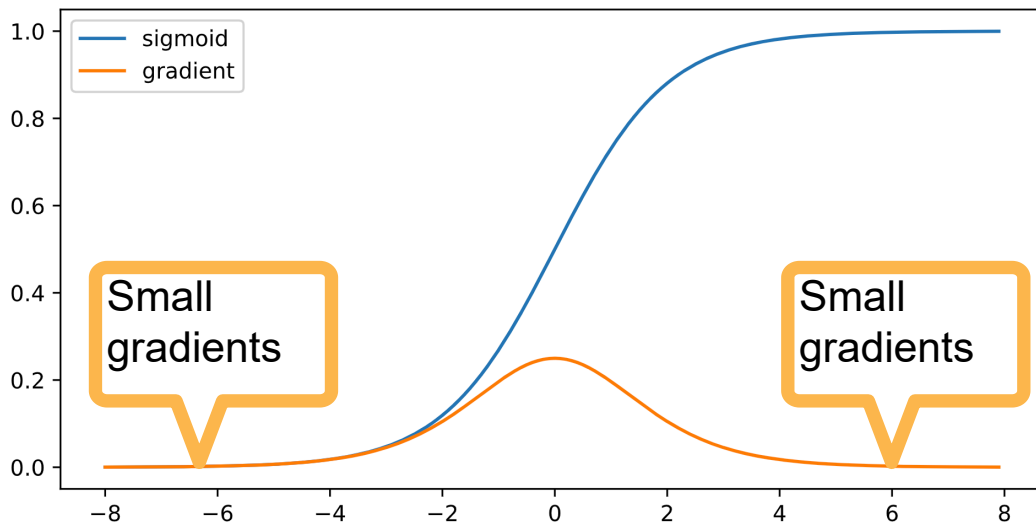
# Issues with Gradient Exploding

- Value out of range: infinity value (NaN)
- Sensitive to learning rate (LR)
  - Not small enough LR  $\rightarrow$  larger gradients
  - Too small LR  $\rightarrow$  No progress
  - May need to change LR dramatically during training

# Gradient Vanishing

- Use sigmoid as the activation function

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



# Issues with Gradient Vanishing

- Gradients with value 0
- No progress in training
  - No matter how to choose learning rate
- Severe with bottom layers (those near the input)
  - Only top layers (near output) are well trained
  - No benefit to make networks deeper

**How to stabilize training?**





# Stabilize Training: Practical Considerations

- Goal: make sure gradient values are in a proper range
  - E.g. in  $[1e-6, 1e3]$
- Multiplication  $\rightarrow$  plus
  - Architecture change (e.g., ResNet)
- Normalize
  - Batch Normalization, Gradient clipping
- Proper activation functions

Quiz. Which of the following are TRUE about the vanishing gradient problem in neural networks? Multiple answers are possible.

- A. Deeper neural networks tend to be more susceptible to vanishing gradients.
- B. Using the ReLU function can reduce this problem.
- C. If a network has the vanishing gradient problem for one training point due to the sigmoid function, it will also have a vanishing gradient for every other training point.
- D. Networks with sigmoid functions don't suffer from the vanishing gradient problem if trained with the cross-entropy loss.

Quiz. Which of the following are TRUE about the vanishing gradient problem in neural networks? Multiple answers are possible?

- A. Deeper neural networks tend to be more susceptible to vanishing gradients.
- B. Using the ReLU function can reduce this problem.
- C. If a network has the vanishing gradient problem for one training point due to the sigmoid function, it will also have a vanishing gradient for every other training point.
- D. Networks with sigmoid functions don't suffer from the vanishing gradient problem if trained with the cross-entropy loss.

Quiz. Let's compare sigmoid with rectified linear unit (ReLU). Which of the following statement is NOT true?

- A. Sigmoid function is more expensive to compute
- B. ReLU has non-zero gradient everywhere
- C. The gradient of Sigmoid is always less than 0.3
- D. The gradient of ReLU is constant for positive input

Quiz. Let's compare sigmoid with rectified linear unit (ReLU). Which of the following statement is NOT true?

- A. Sigmoid function is more expensive to compute
- B. ReLU has non-zero gradient everywhere
- C. The gradient of Sigmoid is always less than 0.3
- D. The gradient of ReLU is constant for positive input

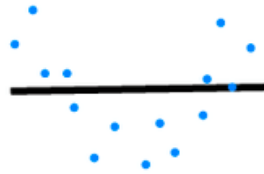


# **Generalization & Regularization**

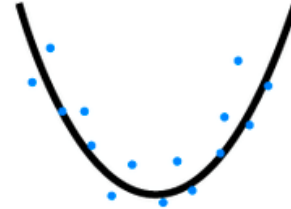
# Training Error and Generalization Error

- Training error: model error on the training data
- **Generalization error**: model error on new data
- Example: practice a future exam with past exams
  - Doing well on past exams (training error) doesn't guarantee a good score on the future exam (generalization error)

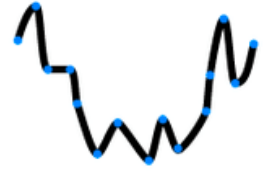
# Underfitting Overfitting



Underfitting



Desired

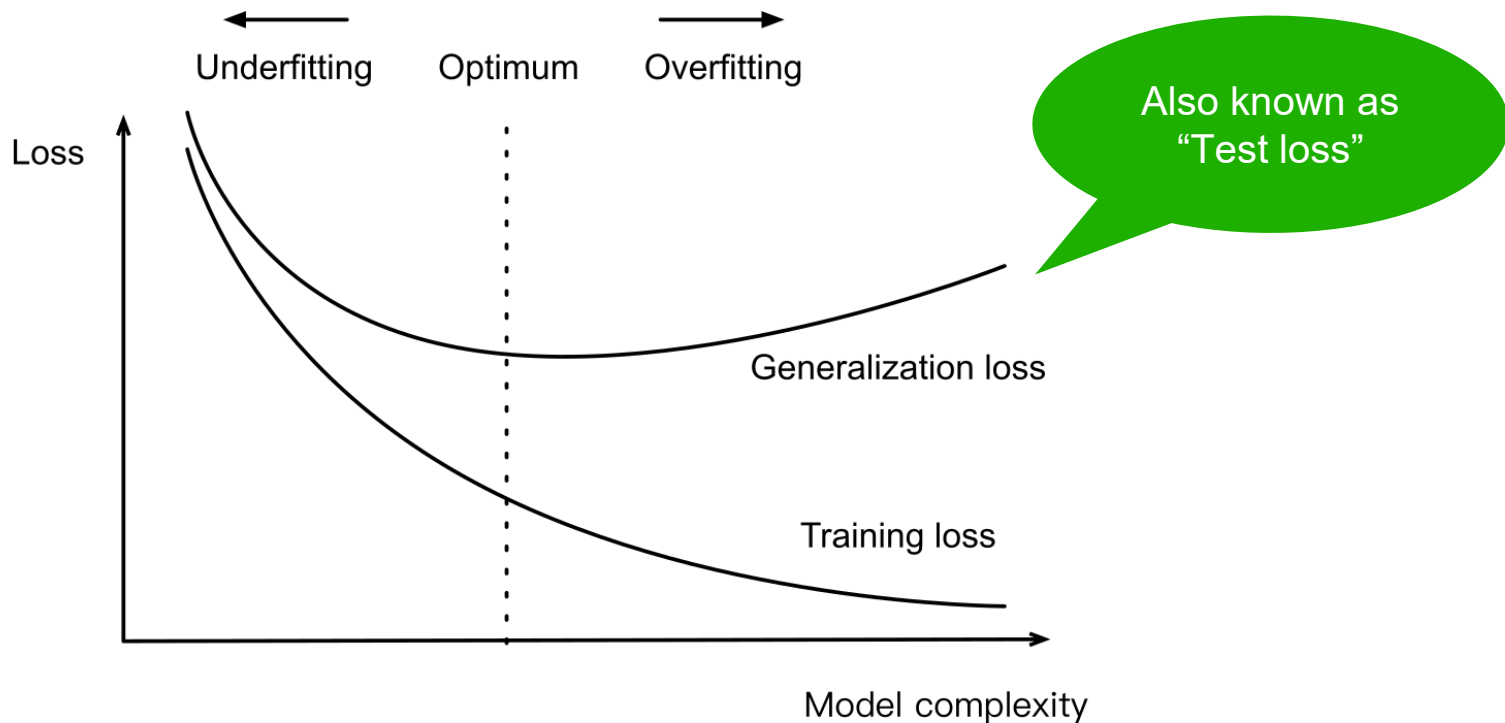


Overfitting

Image credit: hackernoon.com



# Influence of Model Complexity



Quiz Break: Adding more layers to a multi-layer perceptron may cause \_\_\_\_\_.

- A. Vanishing gradients during back propagation.
- B. A more complex decision boundary.
- C. Underfitting.
- D. Higher test loss.
- E. None of these.

Quiz Break: Adding more layers to a multi-layer perceptron may cause \_\_\_\_\_. (Multiple answers)

- A. Vanishing gradients during back propagation.
- B. A more complex decision boundary.
- C. Underfitting.
- D. Higher test loss.
- E. None of these.



# Convolutional Neural Networks (CNNs)

# How to classify

Cats vs. dogs?

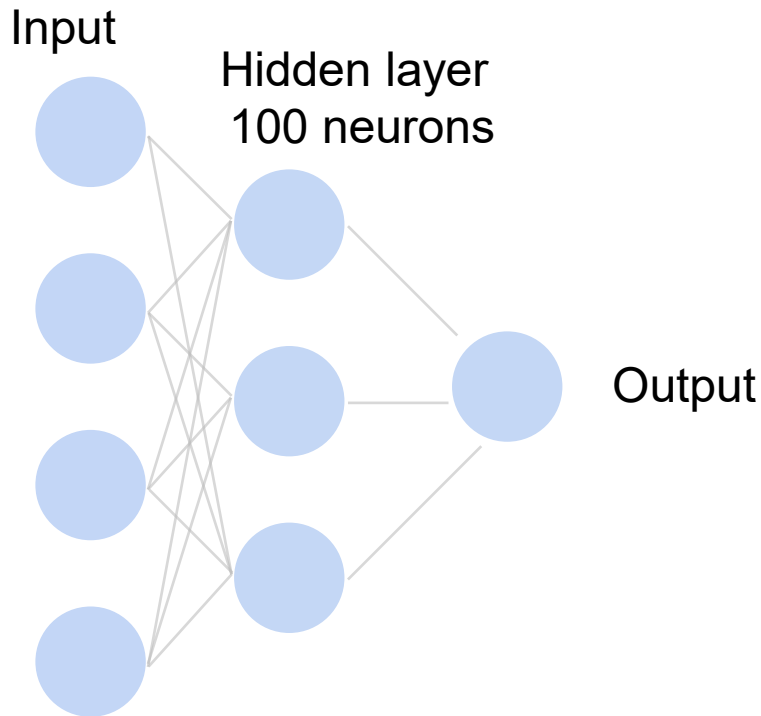


Dual  
**12MP**  
wide-angle and  
telephoto cameras

**36M floats in a RGB  
image!**

# Fully Connected Networks

Cats vs. dogs?



$\sim 36\text{M elements} \times 100 = \sim \mathbf{3.6B}$  parameters!

# Why Convolution?

- Translation Invariance
- Locality



# 2-D Convolution

Input

0	1	2
3	4	5
6	7	8

Kernel

0	1
2	3

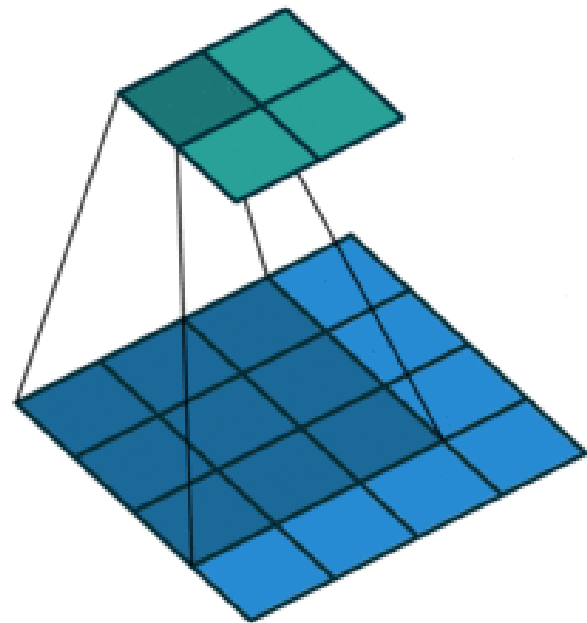
\*

=

Output

19	25
37	43

$$\begin{aligned}0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 &= 19, \\1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 &= 25, \\3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 &= 37, \\4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 &= 43.\end{aligned}$$



(vdumoulin@ Github)



## 2-D Convolution Layer

0	1	2
3	4	5
6	7	8

 \* 

0	1
2	3

 = 

19	25
37	43

- $\mathbf{X}$ :  $n_h \times n_w$  input matrix
- $\mathbf{W}$ :  $k_h \times k_w$  kernel matrix
- $b$ : scalar bias
- $\mathbf{Y}$ :  $(n_h - k_h + 1) \times (n_w - k_w + 1)$  output matrix

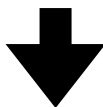
$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- $\mathbf{W}$  and  $b$  are learnable parameters

# 2-D Convolution Layer with Stride and Padding

- Stride is the #rows/#columns per slide
- Padding adds rows/columns around input
- Output shape

Kernel/filter size



$$\lfloor (n_h - k_h + p_h + s_h) / s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w) / s_w \rfloor$$



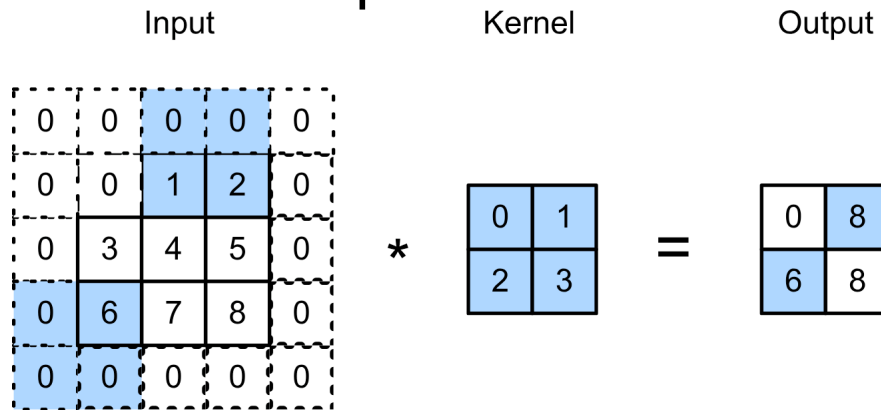
Input size



Pad



Stride

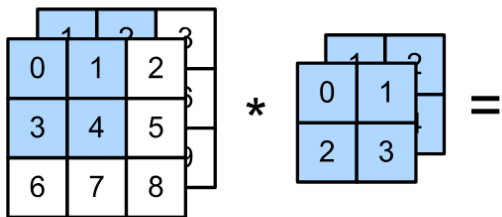


# Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels

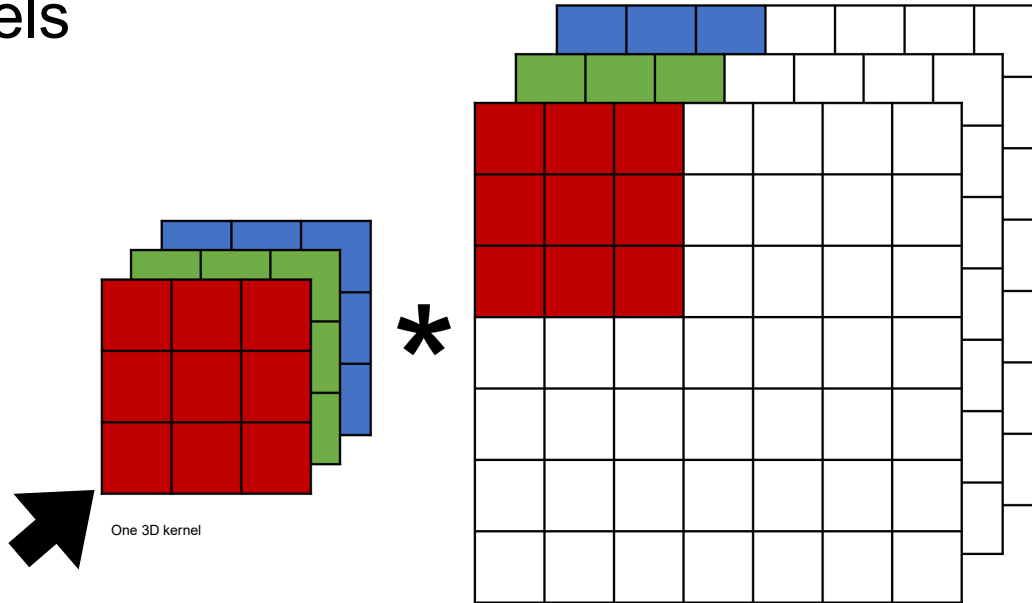
Input

Kernel



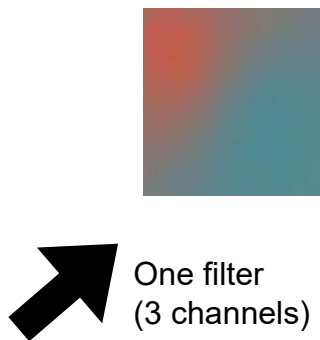
# Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a 2D kernel for each channel, and then sum results over channels

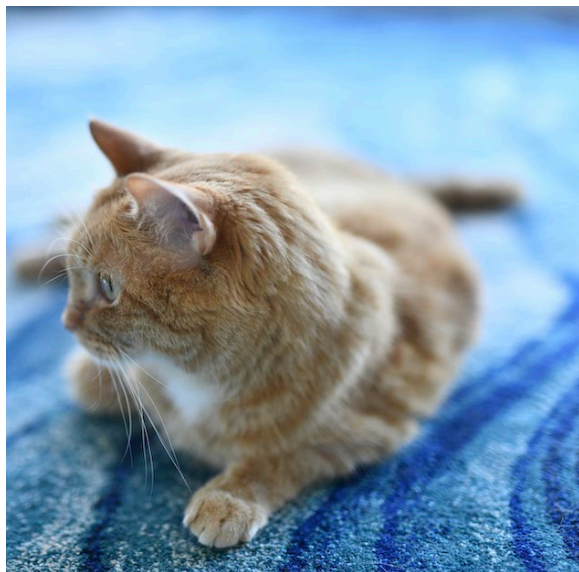


## Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Also call each 3D kernel a “**filter**”, which produce only **one** output channel (due to summation over channels)



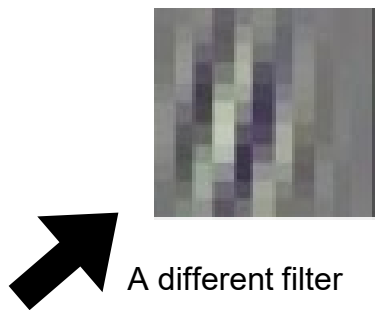
\*



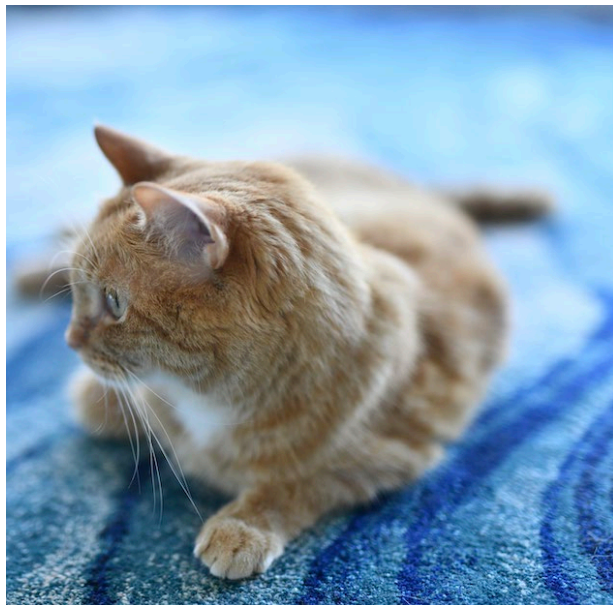
RGB (3 input channels)

## Multiple filters (in one layer)

- Apply multiple filters on the input
- Each filter may learn different features about the input
- Each filter (3D kernel) produces one output channel



\*



RGB (3 input channels)

# Multiple Output Channels

- The # of output channels = # of filters

- Input  $\mathbf{X}: c_i \times n_h \times n_w$

- Kernel  $\mathbf{W}: c_o \times c_i \times k_h \times k_w$

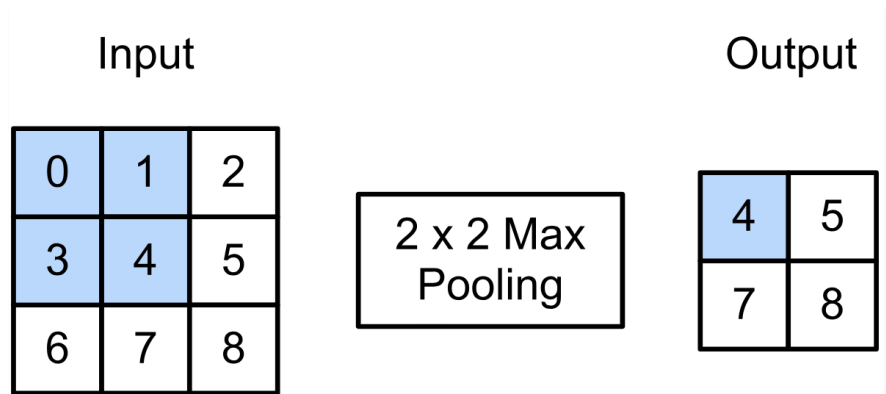
- Output  $\mathbf{Y}: c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

$$\text{for } i = 1, \dots, c_o$$

## 2-D Max Pooling

- Returns the maximal value in the sliding window



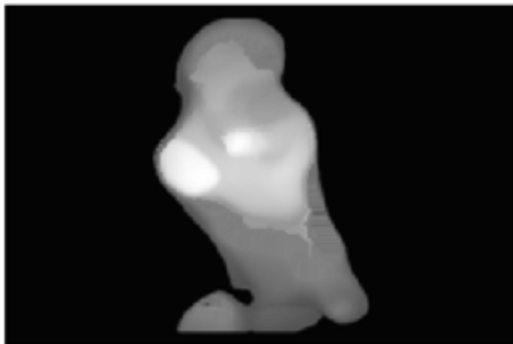
$$\max(0, 1, 3, 4) = 4$$



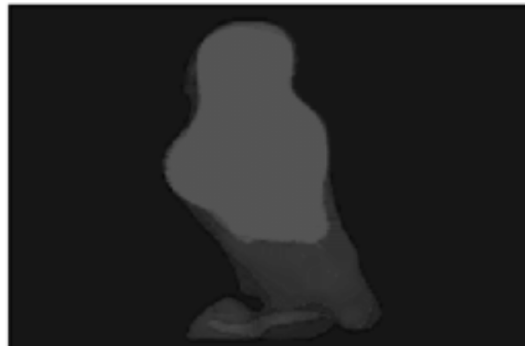
# Average Pooling

- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window

Max pooling



Average pooling



# Quiz

Calculate the convolution of matrix A with kernel matrix B, for no padding and a stride:

A.  $\begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}$

B.  $\begin{bmatrix} 4 & 3 \\ 0 & 5 \end{bmatrix}$

C.  $\begin{bmatrix} 3 & 2 \\ 0 & 6 \end{bmatrix}$

D.  $\begin{bmatrix} 8 & 6 \\ 1 & 12 \end{bmatrix}$

E. None of the above

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 1 \\ 3 & 0 & 2 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

# Quiz

Calculate the convolution of matrix A with kernel matrix B, for no padding and a stride:

A.  $\begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}$

B.  $\begin{bmatrix} 4 & 3 \\ 0 & 5 \end{bmatrix}$

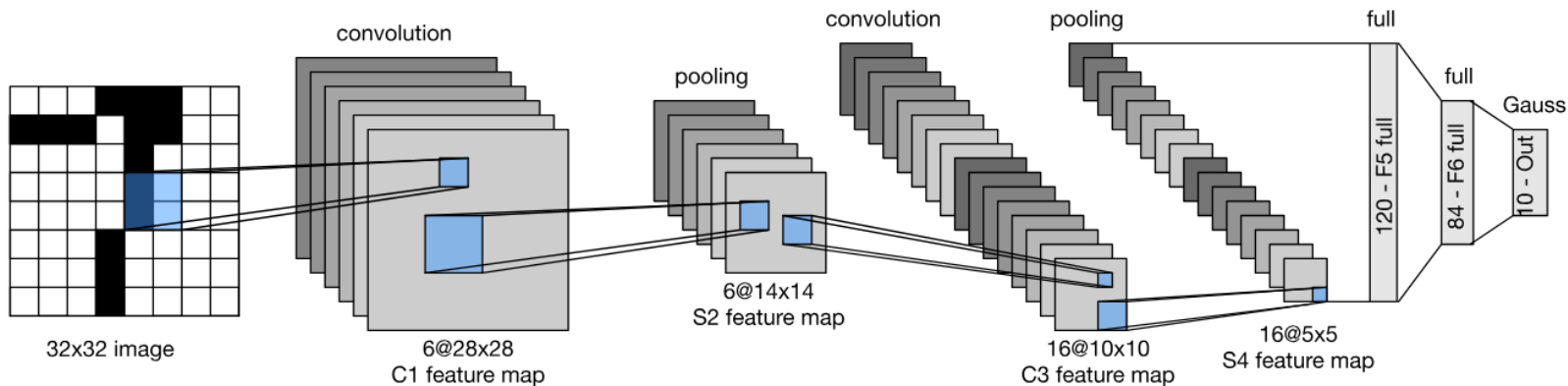
C.  $\begin{bmatrix} 3 & 2 \\ 0 & 6 \end{bmatrix}$

D.  $\begin{bmatrix} 8 & 6 \\ 1 & 12 \end{bmatrix}$

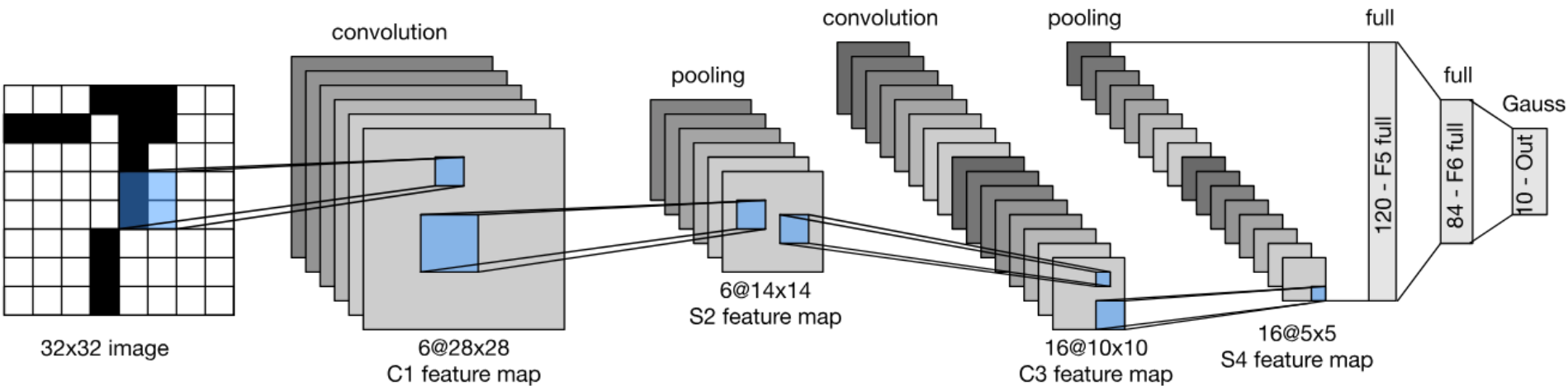
E. None of the above

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 1 \\ 3 & 0 & 2 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

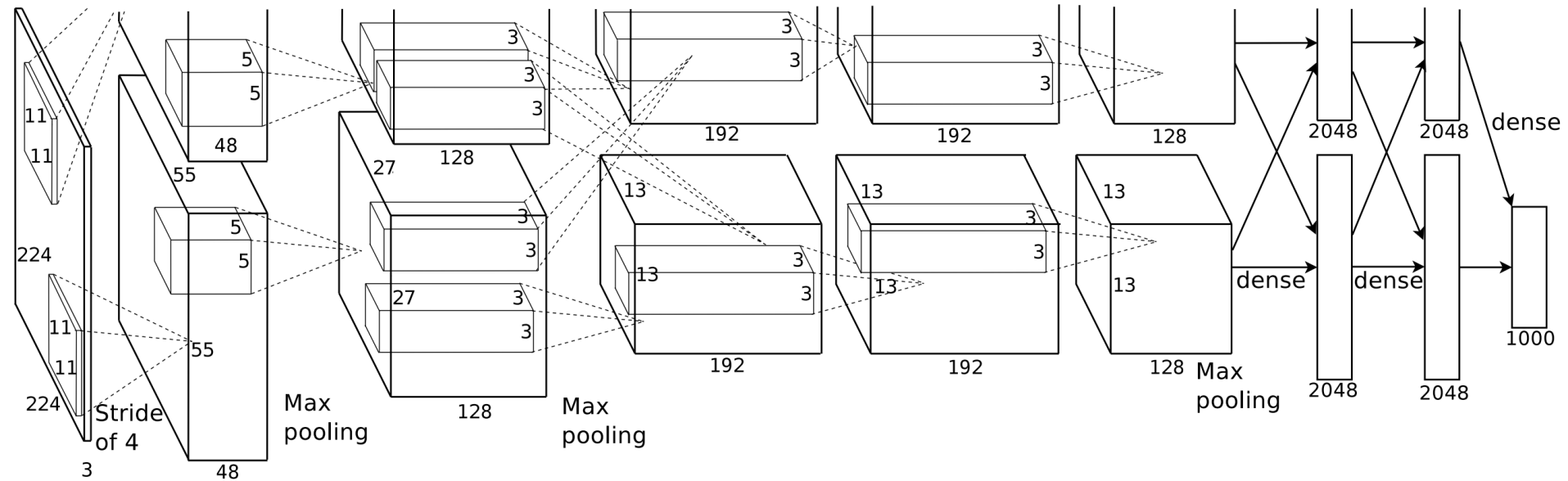
# Convolutional Neural Network Architecture



# LeNet Architecture (first convolutional neural net; 1989)

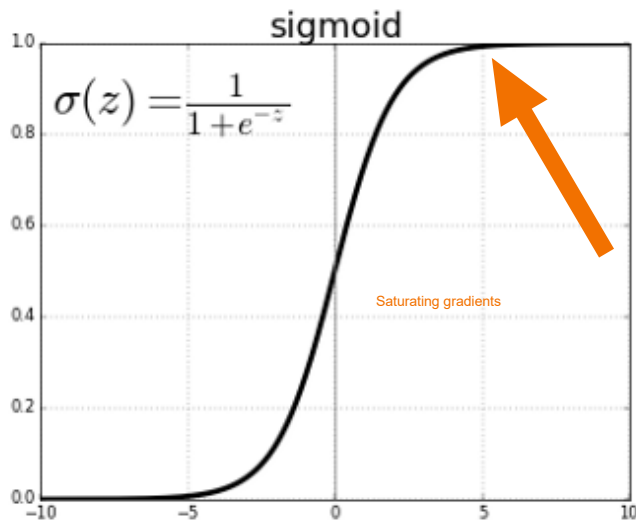


# AlexNet



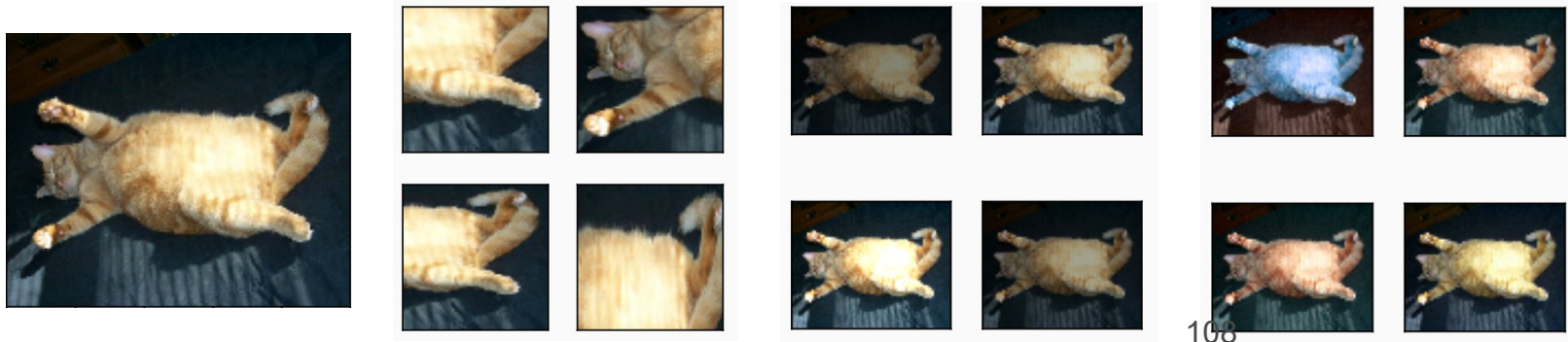
## More Differences...

- Change activation function from sigmoid to ReLu (no more vanishing gradient)



## More Differences...

- Change activation function from sigmoid to ReLu (no more vanishing gradient)
- Data augmentation



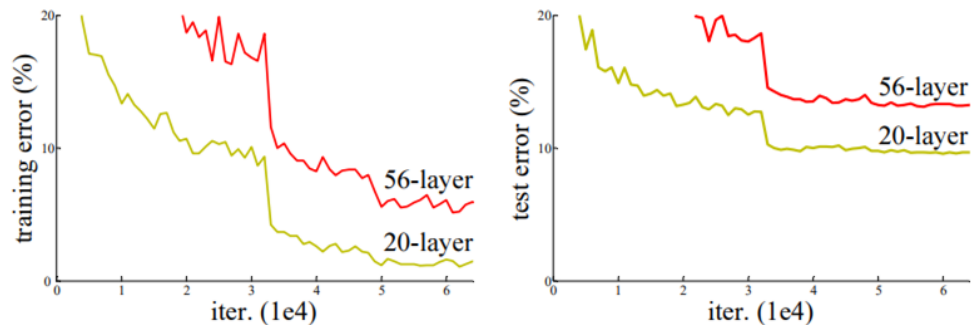


# Simple Idea: Add More Layers

VGG: 19 layers. ResNet: 152 layers. **Add more layers...**  
sufficient?

- No! Some problems:
  - i) Vanishing gradients: more layers → more likely
  - ii) Instability: deeper models are harder to optimize

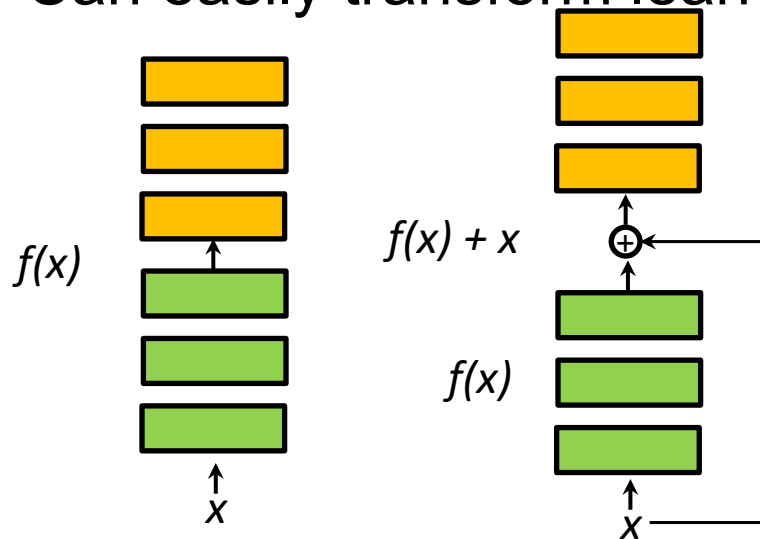
**Reflected in training error:**



# Residual Connections

**Idea:** Identity might be hard to learn, but zero is easy!

- Make all the weights tiny, produces zero for output
- Can easily transform learning identity to learning zero:



**Left:** Conventional layers block

**Right:** **Residual** layer block

To learn identity  $f(x) = x$ , layers now need to learn  $f(x) = 0 \rightarrow$  easier

# Quiz

Which of the below can you implement to solve the gradient vanishing problem?

- A. Reduce the batch size
- B. Increase the depth of network
- C. Oversample majority classes and downsample minority classes
- D. Use SGD optimization
- E. None of the above.

# Quiz

Which of the below can you implement to solve the gradient vanishing problem?

A. Reduce the batch size

B. Increase the depth of network

C. Oversample majority classes and downsample minority classes

D. Use SGD optimization

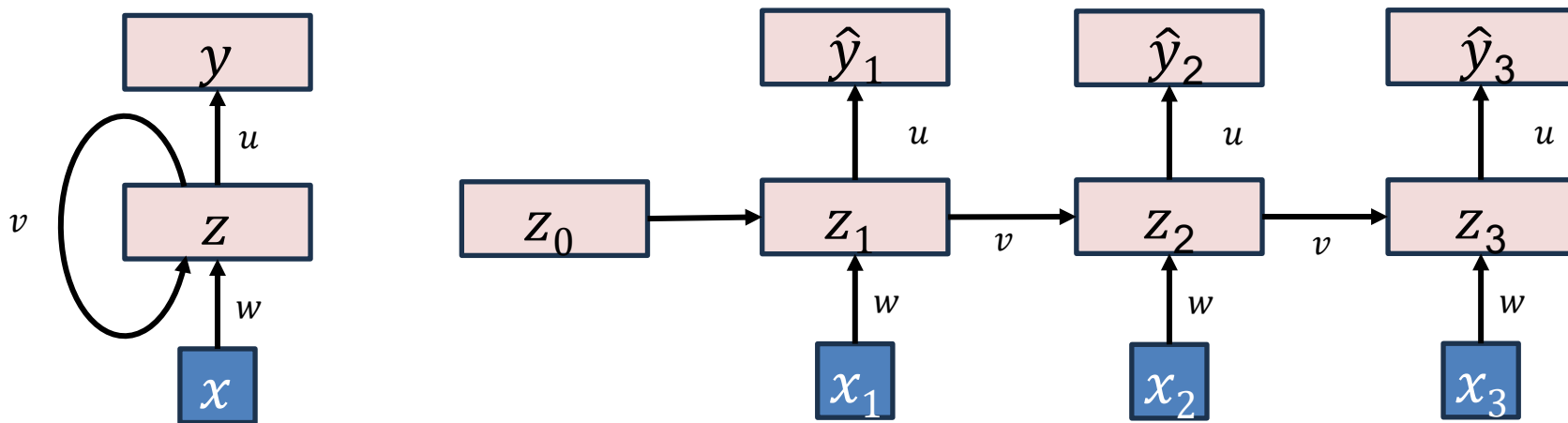
**E. None of the above.**



# **Recurrent Neural Networks (RNNs)**

# Recurrent Neural Networks (RNNs)

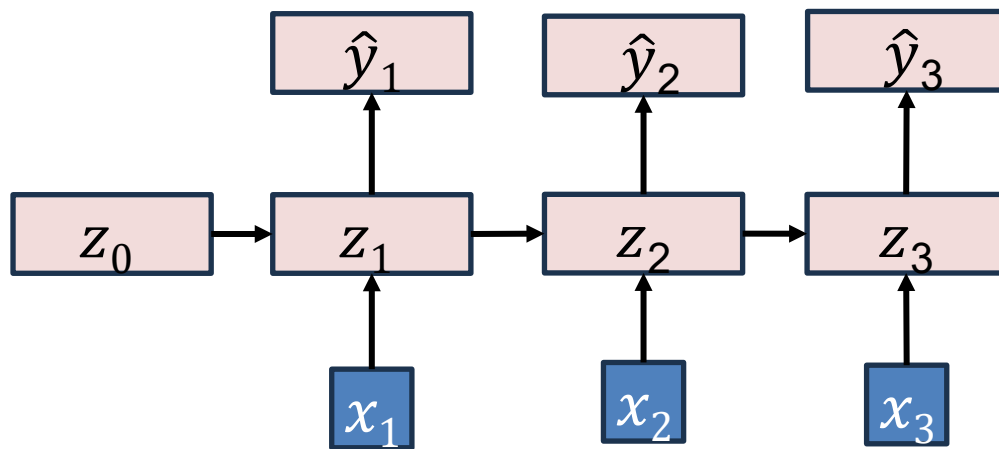
- RNNs introduce **cycles** in the computational graph
- Allowing information to persist; **memory**



# RNNs: High Level

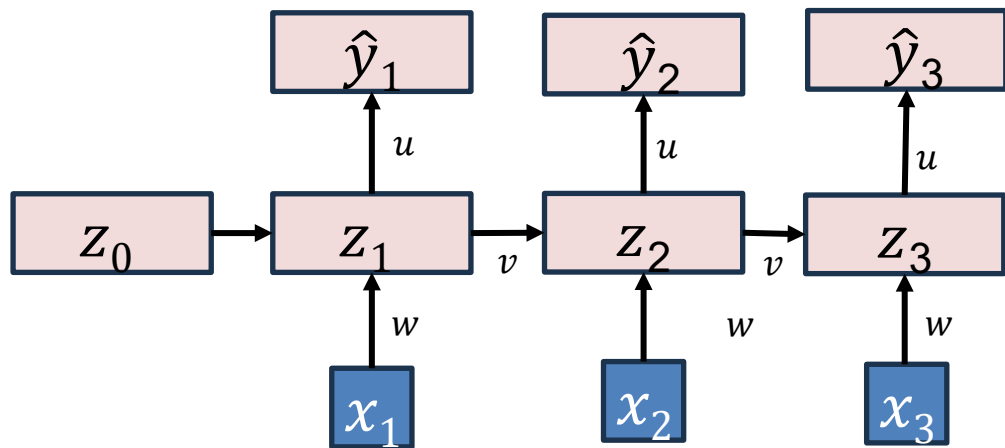
At each time step  $t$ :

- Receive input token  $x_t$
- Receive old hidden state  $z_{t-1}$
- Use  $x_t$  and  $z_{t-1}$  to compute new hidden state  $z_t$
- Use  $z_t$  to predict  $\hat{y}_t$



# Recurrent Neural Networks (RNNs)

- In each time step, the **input value** and the **output of previous hidden state** are used in the computation
- Internal state, **memory** – inputs received at earlier time steps affect the RNN's response to the current input.



$$\hat{y}_t = g_y(Uz_t)$$

$$z_t = g_z(Vz_{t-1} + Wx + b)$$



## Q2.1 Quiz Break

What is the primary characteristic that distinguishes Recurrent Neural Networks (RNNs) from standard feedforward networks?

- A) They use convolutional layers to process spatial data.
- B) They have loops in their architecture, allowing information to persist.
- C) They cannot be trained using backpropagation.
- D) They can only have a single hidden layer.

## Q2.1 Quiz Break Quiz Break

What is the primary characteristic that distinguishes Recurrent Neural Networks (RNNs) from standard feedforward networks?

- A) They use convolutional layers to process spatial data.
- B) They have loops in their architecture, allowing information to persist.**
- C) They cannot be trained using backpropagation.
- D) They can only have a single hidden layer.



# Transformers

# From RNNS to Transformers

- **RNNs** handle sequences but **struggle with long-term dependencies**
- Transformers allow parallelization and efficient context handling
- Example: “The cat the dog chased ran away.”
  - Who ran away?
  - Need to remember/attend to earlier words
- Goal: **decide which words matter most!**



# Word Representations & Context

“The monkey ate the banana. It was \_\_\_\_\_, wasn’t it?”

Could be:

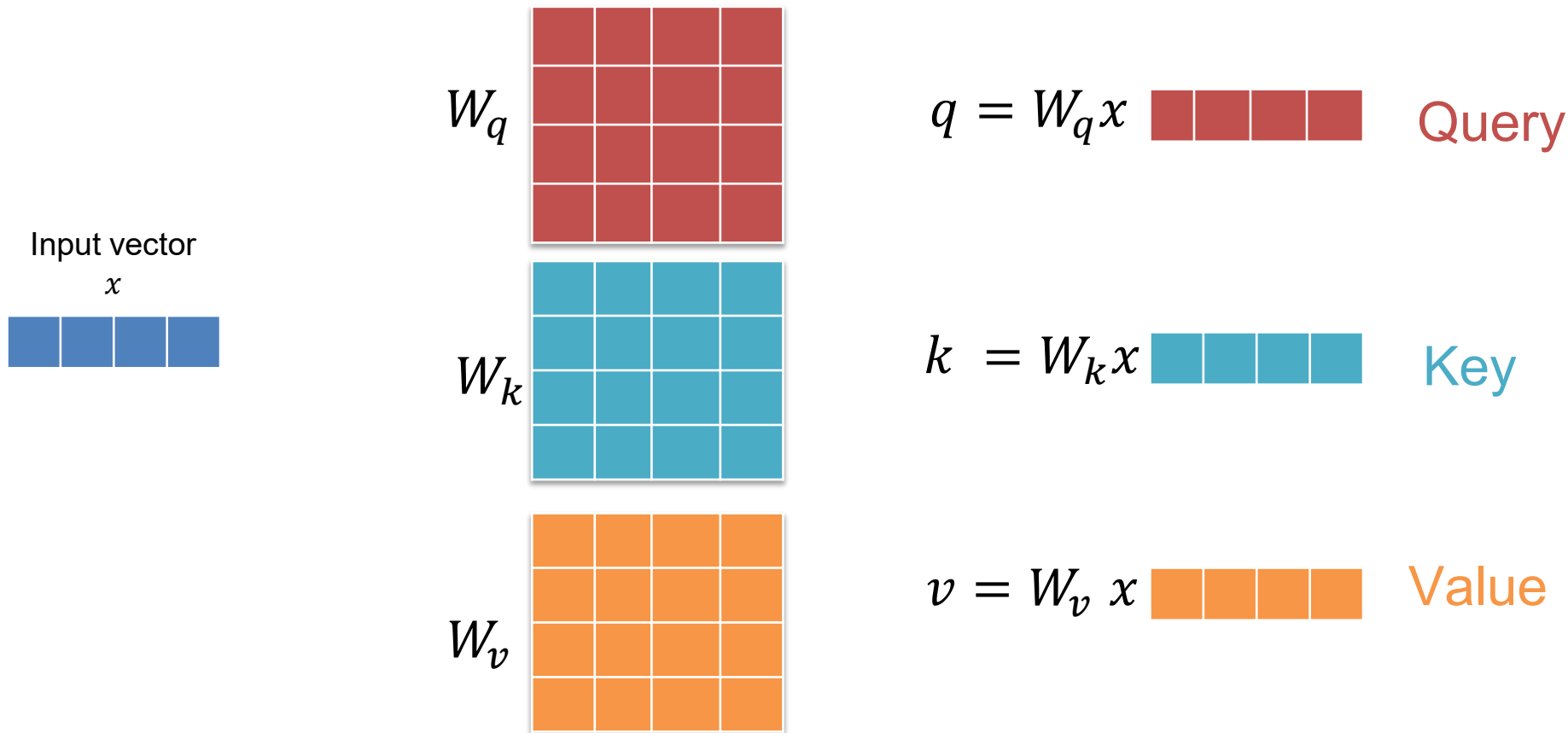
- The monkey ate the banana. It was **ripe**, wasn’t it?
- The monkey ate the banana. It was **hungry**, wasn’t it?

Does “it” mean  
**monkey** or  
**banana**?

Meaning depends on past words (**context**)

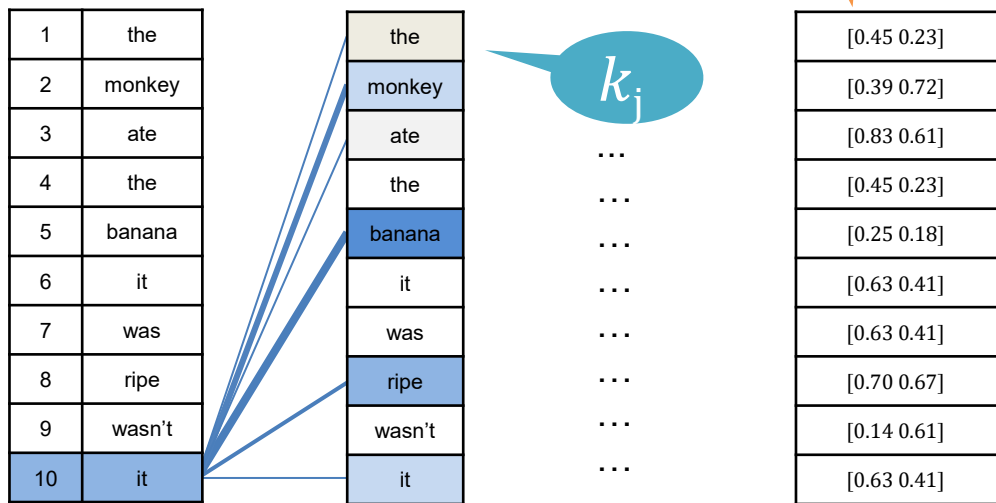
The attention mechanism produces **contextual embeddings**.

# Query, Key, and Value Matrices



# The Attention Mechanism

Each token attends to all other tokens in the same sequence



$$r_{ij} = \frac{\langle q_i, k_j \rangle}{\sqrt{d}}$$



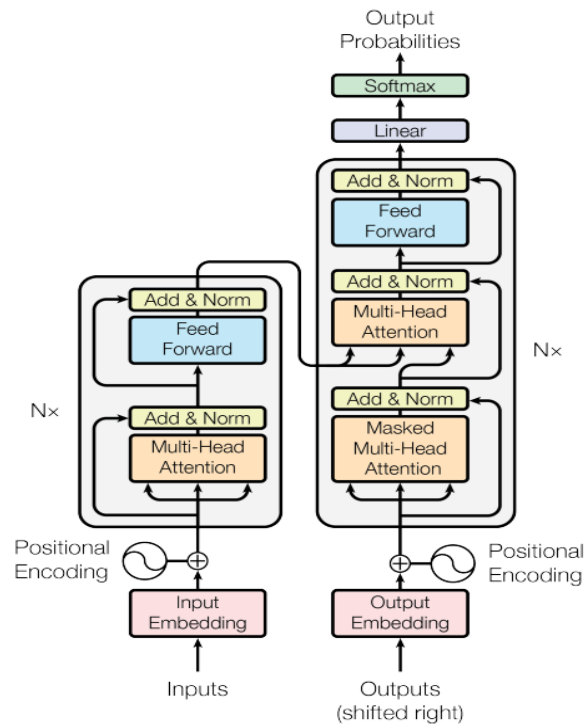
$$p_{i,:} = \text{softmax}(r_{i,:})$$



$$c_i = \sum_j p_{ij} \cdot v_j$$

# Transformer Architecture

- Encoder–Decoder structure
- **Encoder:** maps an input sequence to a sequence of continuous representations  $z$ .
  - Useful for classification
- **Decoder:** Given  $z$ , the decoder generates an output sequence of symbols one element at a time.
  - Useful for generation





## Q3.1 Quiz Break

What is the primary function of the self-attention mechanism?

- A) To track the position of each word in the sequence.
- B) To process the input sequence strictly from left to right.
- C) To weigh the importance and relationship of all words in a sequence relative to each other.
- D) To reduce the size of the model by using fewer parameters.

## Q3.1 Quiz Break

What is the primary function of the self-attention mechanism?

- A) To track the position of each word in the sequence.
- B) To process the input sequence strictly from left to right.
- C) To weigh the importance and relationship of all words in a sequence relative to each other.**
- D) To reduce the size of the model by using fewer parameters.

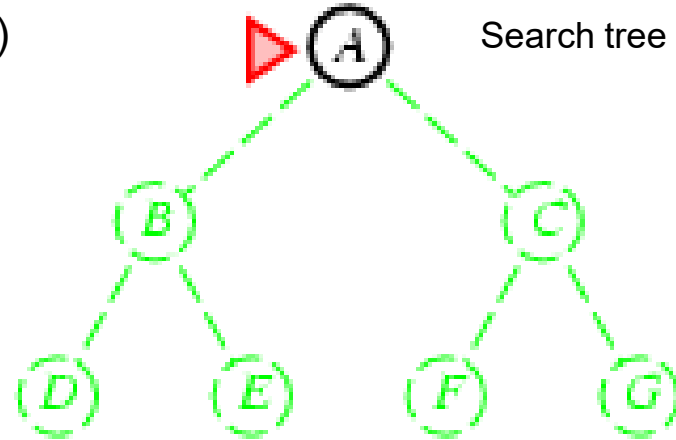


# Uninformed Search

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. enqueue(Initial states)
2. While (queue not empty)
3.   s = dequeue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   enqueue(T)
7. endwhile



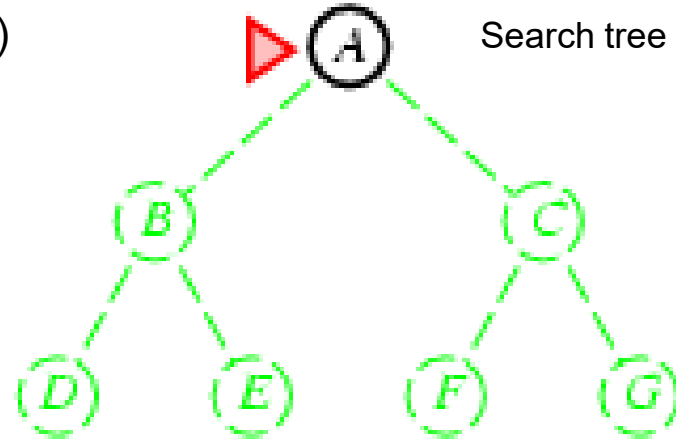
Initial state: **A**

Goal state: **G**

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, **OPEN**)  
→ [A] →

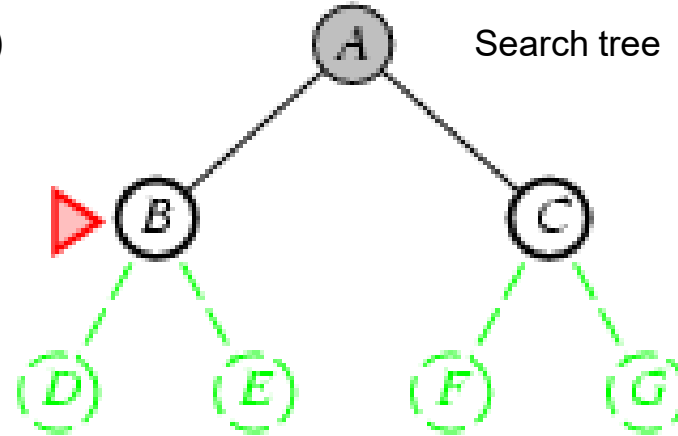
Initial state: **A**

Goal state: **G**

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, OPEN)  
→ [CB] → A

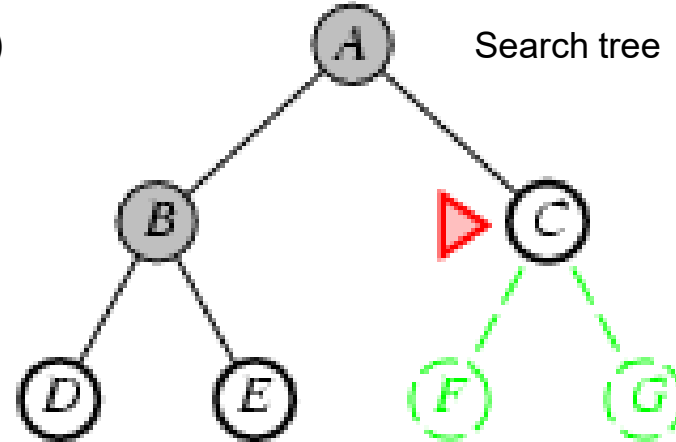
Initial state: **A**

Goal state: **G**

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, **OPEN**)  
→ [EDC] → B

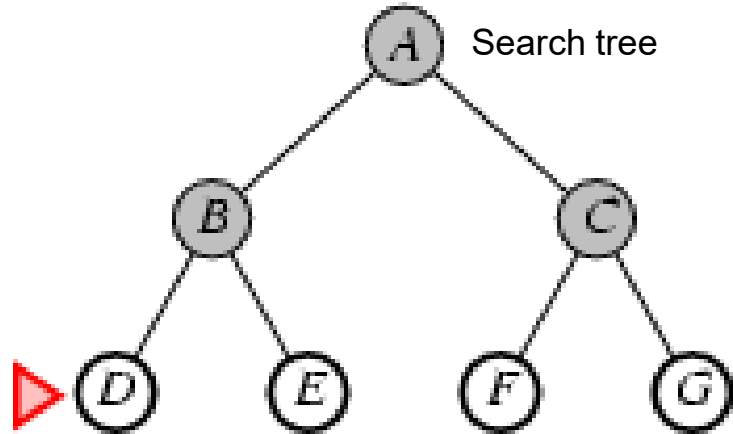
Initial state: **A**

Goal state: **G**

## Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, OPEN)

□[GFED] → C

If G is a goal, we've seen it, but we don't stop!

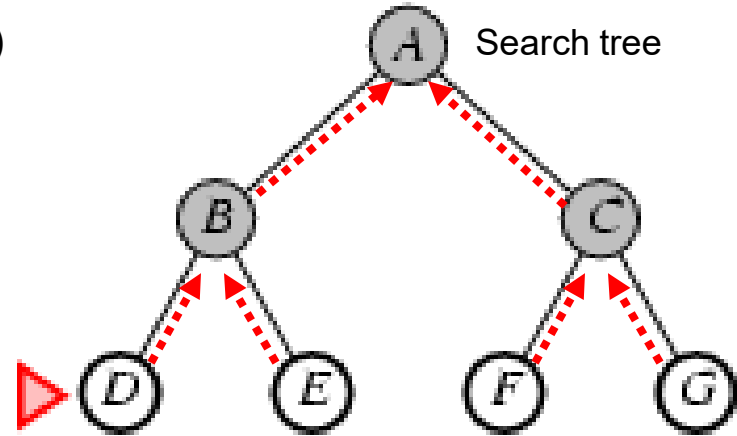
Initial state: **A**



## Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. enqueue(Initial states)
2. While (queue not empty)
3.   s = dequeue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   enqueue(T)
7. endwhile



queue

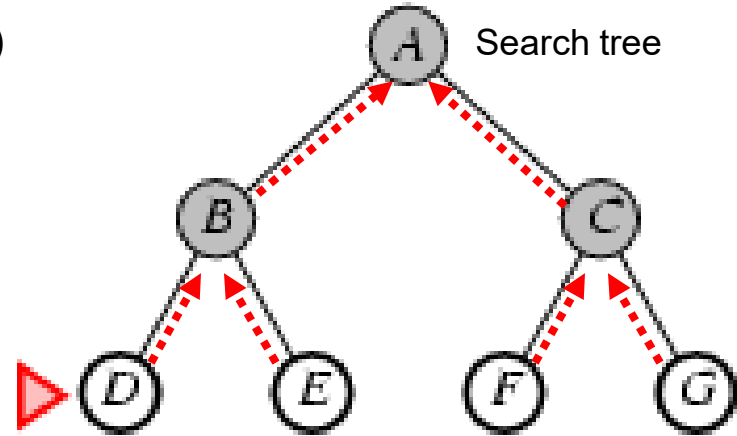
□ → G

... until much later we pop G.

## Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. enqueue(Initial states)
2. While (queue not empty)
3.   s = dequeue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   enqueue(T)
7. endwhile



queue

□ → G

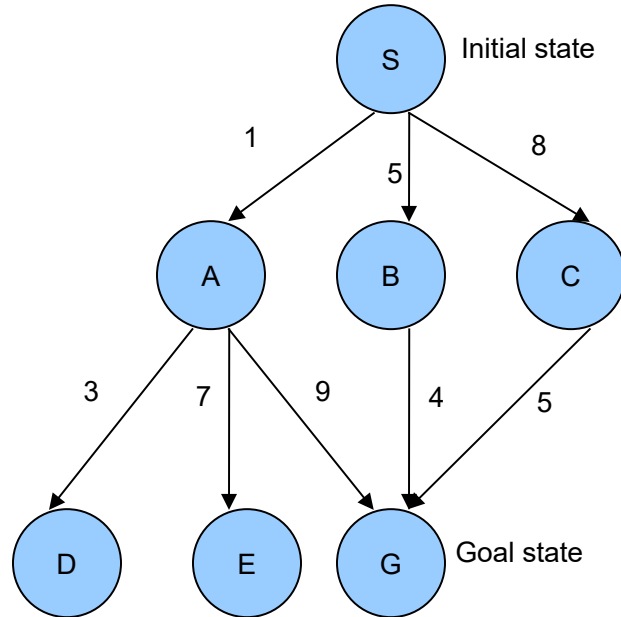
... until much later we pop G.

We need **back pointers** to recover the solution path.

## Uniform-cost search

- Find the least-cost goal
- Each node has a path cost from start (= sum of edge costs along the path).
- Expand the least cost node first.
- Use a **priority queue** instead of a normal queue
  - Always take out the least cost item

## Example

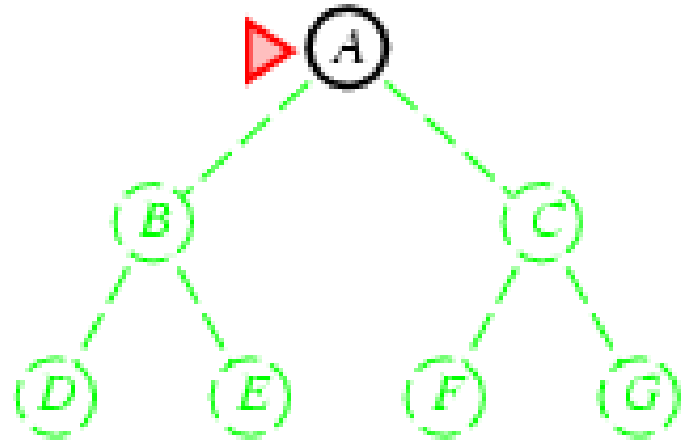


- 1: (S,0), [(A,1), (B,5), (C,8)]
- 2: (A,1), [(B,5), (C,8), (D,4), (E,8), (G,10)]
- 3: (D,4), [(B,5), (C,8), (E,8), (G,10)]
- 4: (B,5), [(C,8), (E,8), (G,9)]
- 5: (C,8), [(E,8), (G,9)]
- 6: (E,8), [(G,9)]
- 7: (G,9), []: Success!

## Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



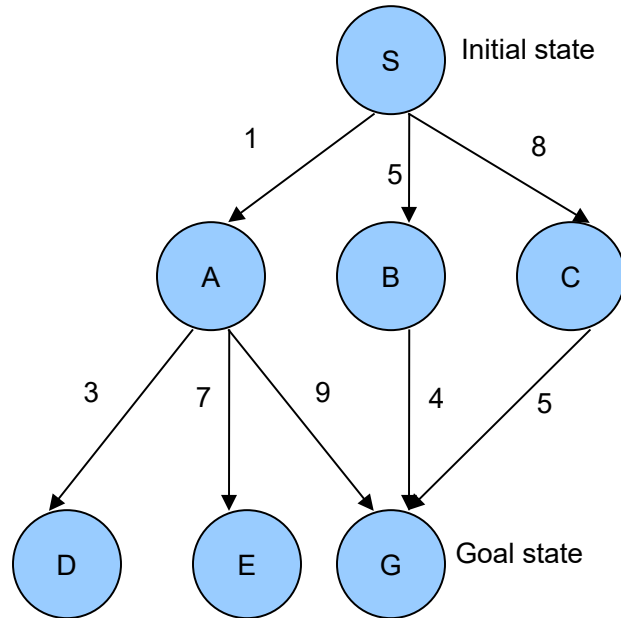
stack (**fringe**)

1. A, [B, C]
2. B, [D, E, C]
3. D, [E, C]

## Iterative deepening

- Search proceeds like BFS, but fringe is like DFS
  - Complete, optimal like BFS
  - Small space complexity like DFS
  - Time complexity like BFS
- Preferred uninformed search method

## Example



## Nodes expanded by:

- Breadth-First Search: S A B C D E G

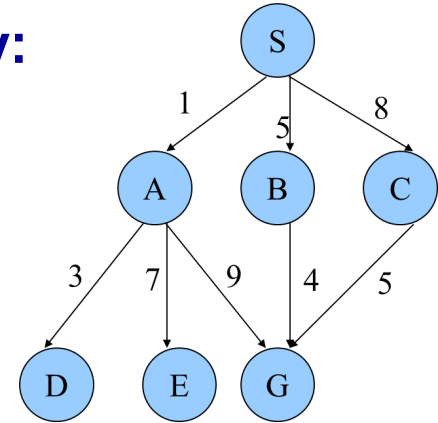
Solution found: S A G

- Uniform-Cost Search: S A D B C E G

Solution found: S B G (This is the only uninformed search that worries about costs.)

- Depth-First Search: S A D E G

Solution found: S A G





## Performance of search algorithms on trees

b: branching factor (assume finite)

d: goal depth m: graph depth

	Complete	optimal	time	space
Breadth-first search	Y	Y, if <sup>1</sup>	$O(b^d)$	$O(b^d)$
Uniform-cost search <sup>2</sup>	Y	Y	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$
Depth-first search	N	N	$O(b^m)$	$O(bm)$
Iterative deepening	Y	Y, if <sup>1</sup>	$O(b^d)$	$O(bd)$

<sup>1</sup> only if cost constant, and positive max. depth given in advance

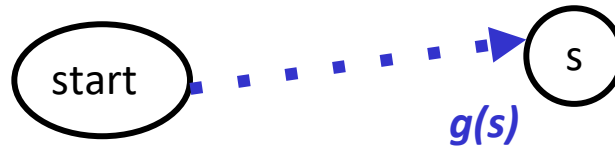


# Informed Search

# Uninformed vs Informed Search

Uninformed search (all of what we saw). Know:

- Path cost  $g(s)$  from start to node  $s$
- Successors.



Informed search. Know:

- All uninformed search properties, plus
- Heuristic  $h(s)$  from  $s$  to goal (recall game heuristic)

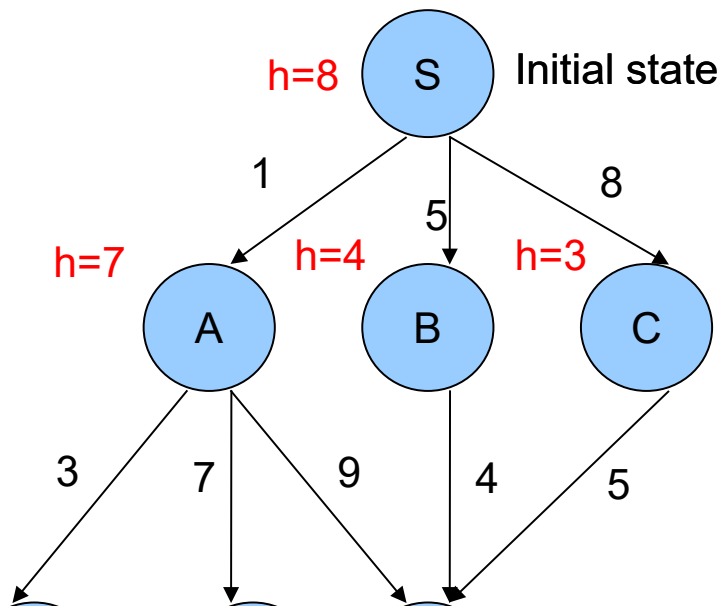
## Attempt 3: A\* Search

Same idea, use  $g(s) + h(s)$ , with one **requirement**

- Demand that  $h(s) \leq h^*(s)$  where  $h^*(s)$  is true cost from  $s$  to goal.
- If heuristic has this property, it is called “admissible”
  - Optimistic! Never over-estimates

# Recap and Examples

**Example for A\*:**



# Recap and Examples

## Example for A\*:

OPEN

S(0+8)

A(1+7) B(5+4) C(8+3)

B(5+4) C(8+3) D(4+inf) E(8+inf) G(10+0)

C(8+3) D(4+inf) E(8+inf) G(9+0)

C(8+3) D(4+inf) E(8+inf)

CLOSED

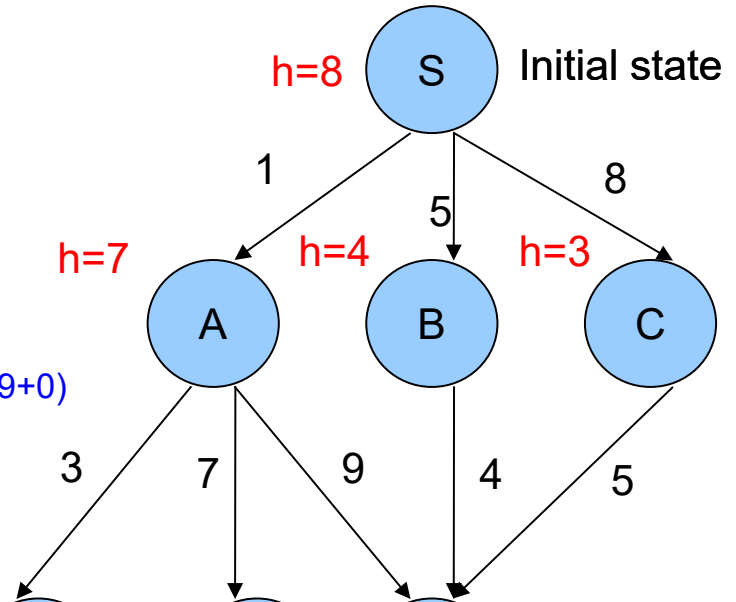
-

S(0+8)

S(0+8) A(1+7)

S(0+8) A(1+7) B(5+4)

S(0+8) A(1+7) B(5+4) G(9+0)



# Break & Quiz

**Q 1.2:** Which of the following are admissible heuristics?

- i.  $h(s) = h^*(s)$
- ii.  $h(s) = \max(2, h^*(s))$
- iii.  $h(s) = \min(2, h^*(s))$
- iv.  $h(s) = h^*(s) - 2$
- v.  $h(s) = \sqrt{h^*(s)}$

- A. All of the above
- B. (i) (iii) (iv)

# Break & Quiz

**Q 1.2:** Which of the following are admissible heuristics?

- i.  $h(s) = h^*(s)$
- ii.  $h(s) = \max(2, h^*(s))$
- iii.  $h(s) = \min(2, h^*(s))$
- iv.  $h(s) = h^*(s) - 2$
- v.  $h(s) = \sqrt{h^*(s)}$

- A. All of the above
- B. (i) (iii) (iv)



# Break & Quiz

**Q 1.2:** Which of the following are admissible heuristics?

- i.  $h(s) = h^*(s)$
- ii.  $h(s) = \max(2, h^*(s))$  No:  $h(s)$  might be too big
- iii.  $h(s) = \min(2, h^*(s))$
- iv.  $h(s) = h^*(s) - 2$  No:  $h(s)$  might be negative
- v.  $h(s) = \sqrt{h^*(s)}$  No: if  $h^*(s) < 1$  then  $h(s)$  is bigger

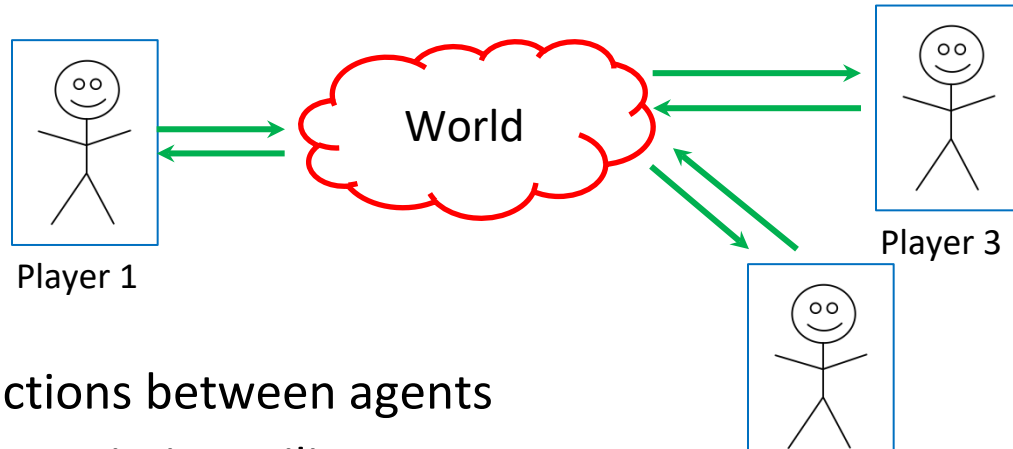
- A. All of the above
- B. (i) (iii) (iv)



# Games

# Games Setup

Games setup: **multiple** agents



- Now: interactions between agents

# Normal Form Game

Mathematical description of simultaneous games.

- $n$  players  $\{1, 2, \dots, n\}$
- Player  $i$  chooses strategy  $a_i$  from action space  $A_i$ .
- Strategy profile:  $\mathbf{a} = (a_1, a_2, \dots, a_n)$
- Player  $i$  gets rewards  $u_i(\mathbf{a})$ 
  - **Note:** reward depends on other players!

# Example of Normal Form Game

## Ex: Prisoner's Dilemma

Player 1	Player 2	
	<i>Stay silent</i>	<i>Betray</i>
<i>Stay silent</i>	-1, -1	-3, 0
<i>Betray</i>	0, -3	-2, -2

- 2 players, 2 actions: yields 2x2 payoff matrix

# Strictly Dominant Strategies

Let's analyze such games. Some strategies are better than others!

- Strictly dominant strategy: if  $a_i$  strictly better than  $b$  *regardless* of what other players do,  $a_i$  is **strictly dominant**
- I.e.,  $u_i(a_i, a_{-i}) > u_i(b, a_{-i}), \forall b \neq a_i, \forall a_{-i}$



## Dominant Strategy Equilibrium

$a^*$  is a (strictly) dominant strategy equilibrium (DSE), if every player  $i$  has a strictly dominant strategy  $a_i^*$

- Rational players will play at DSE, if one exists.

Player 2		
Player 1	<i>Stay silent</i>	<i>Betray</i>

# Break & Quiz

Two firms, A and B, are deciding whether to launch a new product. Each firm can either launch or not launch. Their profits depend on their choices, and the payoff matrix is as follows:

	<i>B: Launch</i>	<i>B: Not Launch</i>
<i>A: Launch</i>	(20, 20)	(40, 10)
<i>A: Not Launch</i>	(10, 40)	(30, 30)

What is the strictly dominant strategy for each firm:

- A's dominant strategy is to launch, and B's dominant strategy is not to launch.



# Break & Quiz

Two firms, A and B, are deciding whether to launch a new product. Each firm can either launch or not launch. Their profits depend on their choices, and the payoff matrix is as follows:

	<i>B: Launch</i>	<i>B: Not Launch</i>
<i>A: Launch</i>	(20, 20)	(40, 10)
<i>A: Not Launch</i>	(10, 40)	(30, 30)

What is the strictly dominant strategy for each firm:

- A's dominant strategy is to launch, and B's dominant strategy is not to launch.

# Dominant Strategy Equilibrium

Dominant Strategy Equilibrium does not always exist.

Player 2		<i>L</i>	<i>R</i>
Player 1			
<i>T</i>		2, 1	0, 0

# Nash Equilibrium

$a^*$  is a Nash equilibrium if no player has an incentive to **unilaterally deviate**

$$u_i(a_i^*, a_{-i}^*) \geq u_i(a_i, a_{-i}^*) \quad \forall a_i \in A_i$$

Player 2		
	$L$	$R$
Player 1		

## Nash Equilibrium: Best Response to Each Other

$a^*$  is a Nash equilibrium:

$$\forall i, \forall b \in A_i: u_i(a_i^*, a_{-i}^*) \geq u_i(b, a_{-i}^*)$$

(no player has an incentive to **unilaterally deviate**)

- Pure Nash equilibrium:
  - A **pure strategy** is a deterministic choice (no randomness).
  - Later: we will consider **mixed** strategies

# Pure Nash Equilibrium may not exist

So far, pure strategy: each player picks a deterministic strategy. But:

Player 2				
		<i>rock</i>	<i>paper</i>	<i>scissors</i>
Player 1				
<i>rock</i>		0, 0	<u>-1, 1</u>	<u>1, -1</u>
<i>paper</i>		<u>1, -1</u>	0, 0	<u>-1, 1</u>

# Mixed Strategy Nash Equilibrium

Example:  $x_1^*(\cdot) = x_2^*(\cdot) = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$

Player 1 \ Player 2			
	<i>rock</i>	<i>paper</i>	<i>scissors</i>
<i>rock</i>	0, 0	-1, 1	1, -1

# Properties of Nash Equilibrium

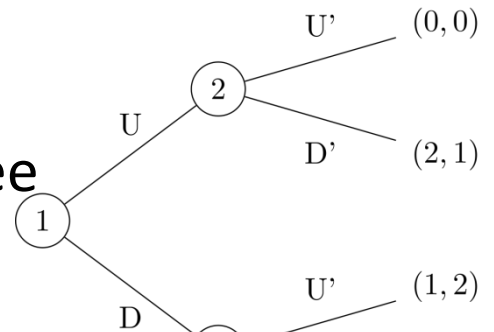
Major result: (John Nash '51)

- Every **finite** (players, actions) game has at least one Nash equilibrium
  - But not necessarily **pure** (i.e., deterministic strategy)
- Could be more than one
- Searching for Nash equilibria: computationally **hard**.

# Sequential-Move Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- **Rewards at leaves**
- Find strategies: perform search over the tree





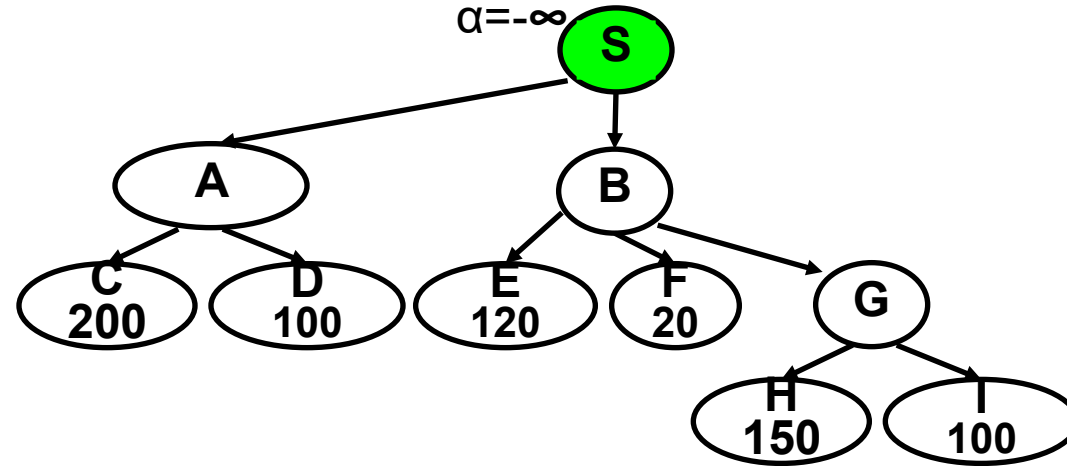
# Minimax algorithm in execution

max

min

max

min



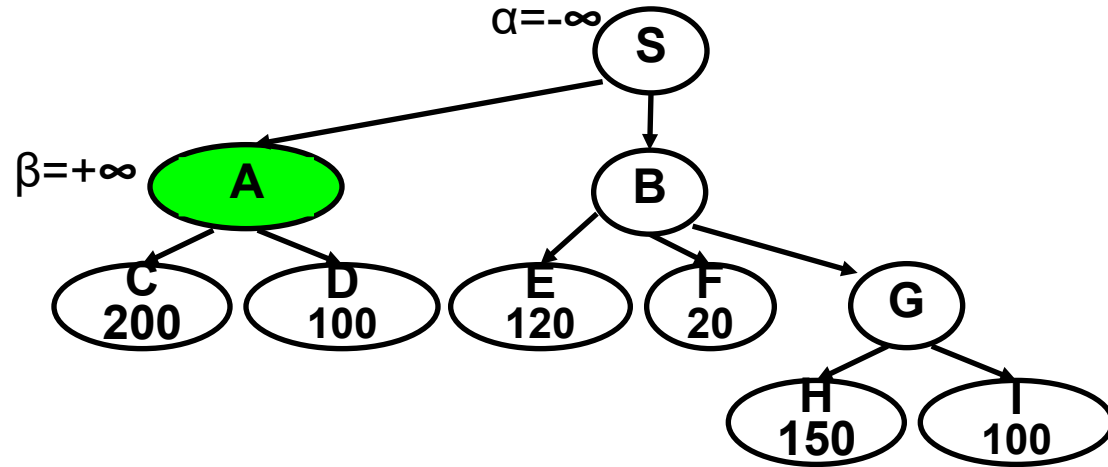
# Minimax algorithm in execution

max

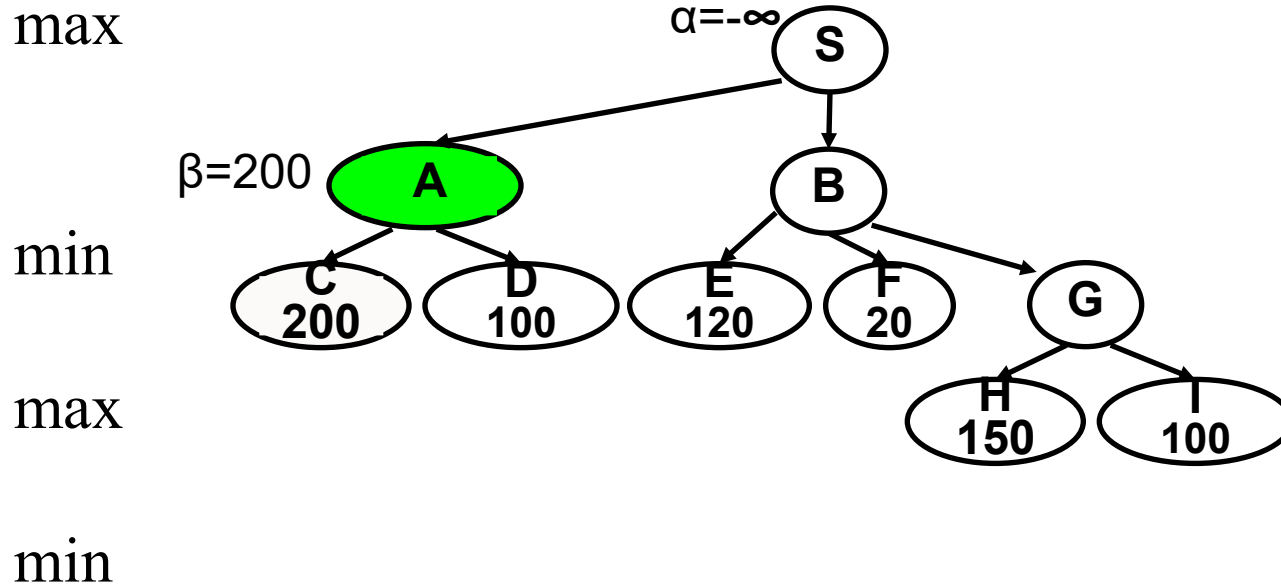
min

max

min

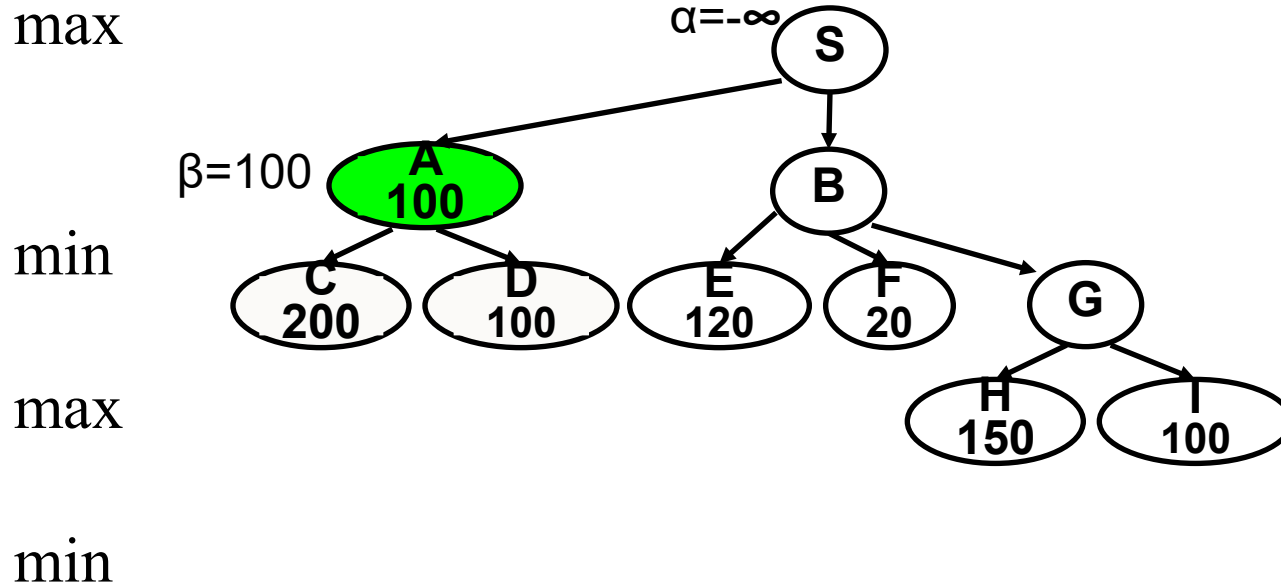


# Minimax algorithm in execution

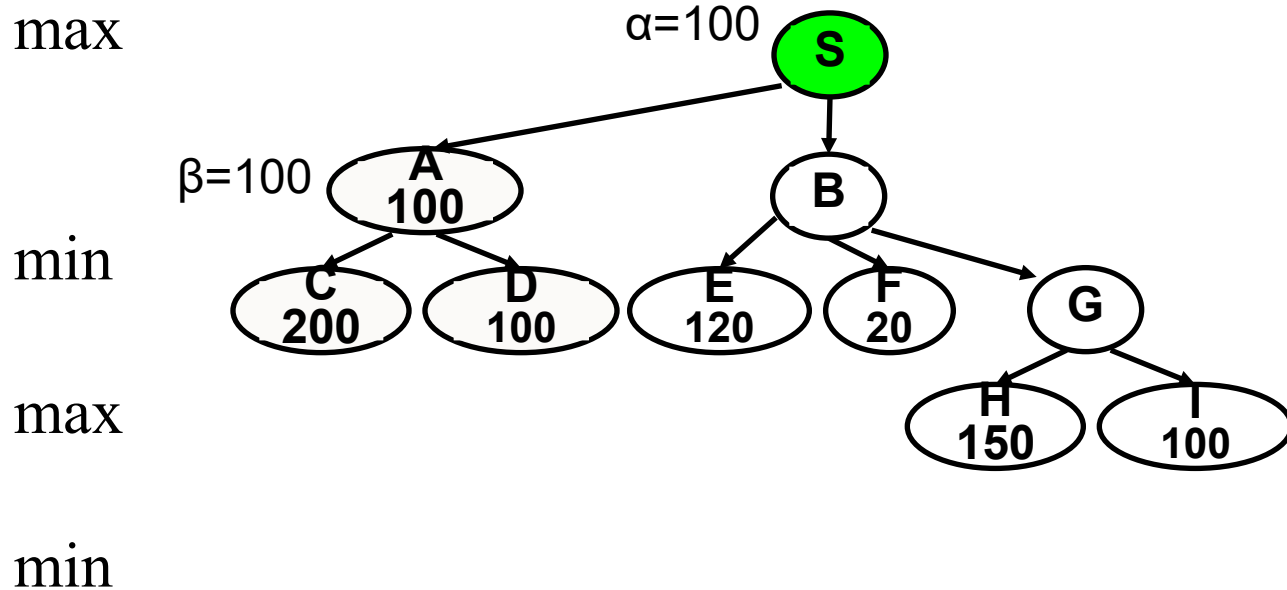


The execution on the

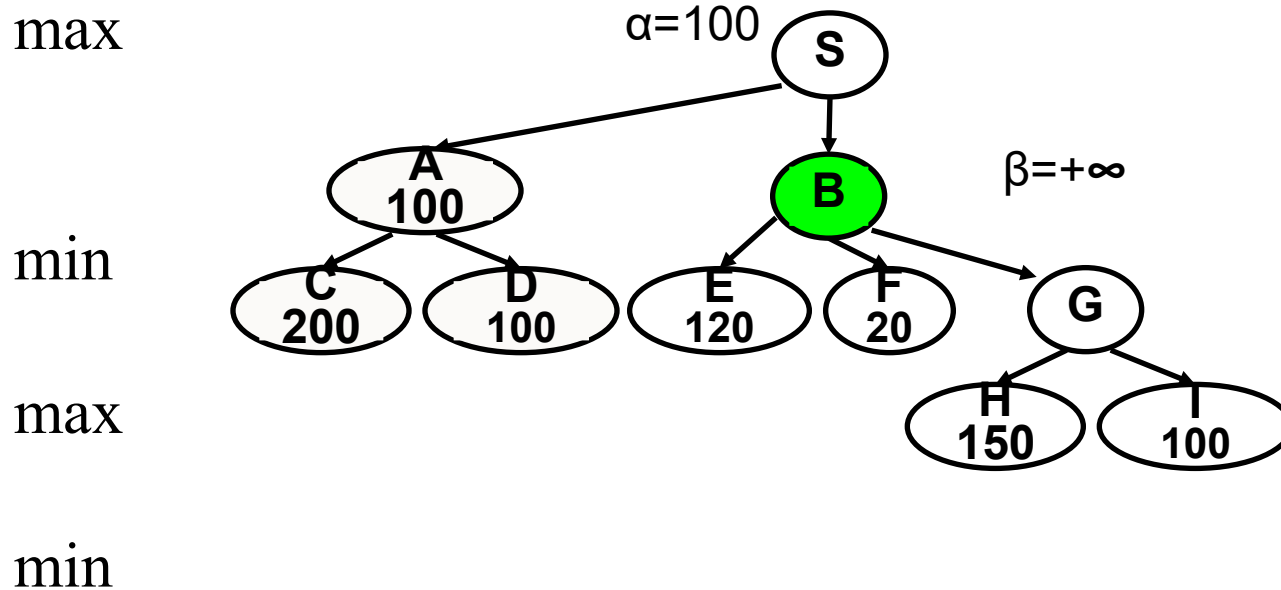
# Minimax algorithm in execution



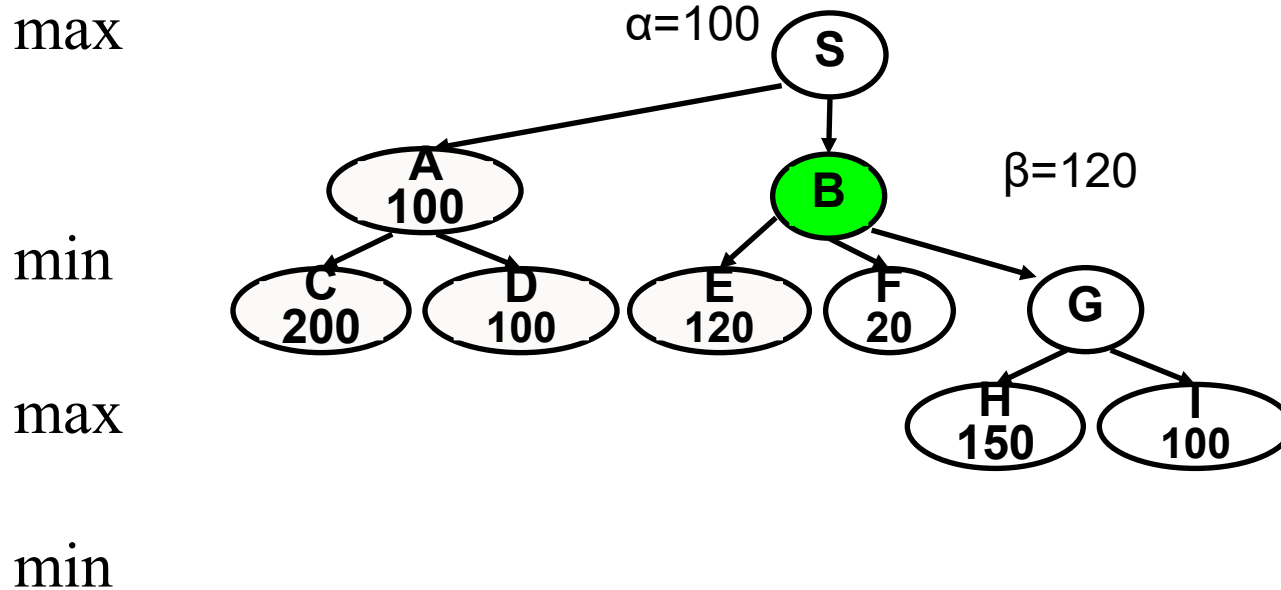
# Minimax algorithm in execution



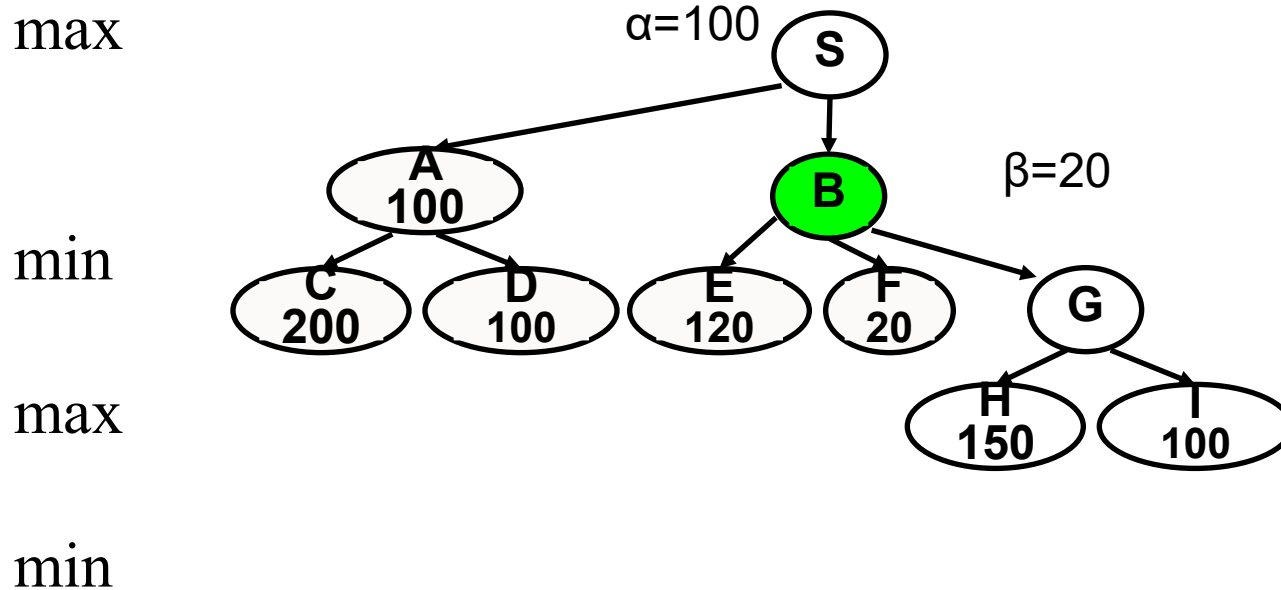
# Minimax algorithm in execution



# Minimax algorithm in execution

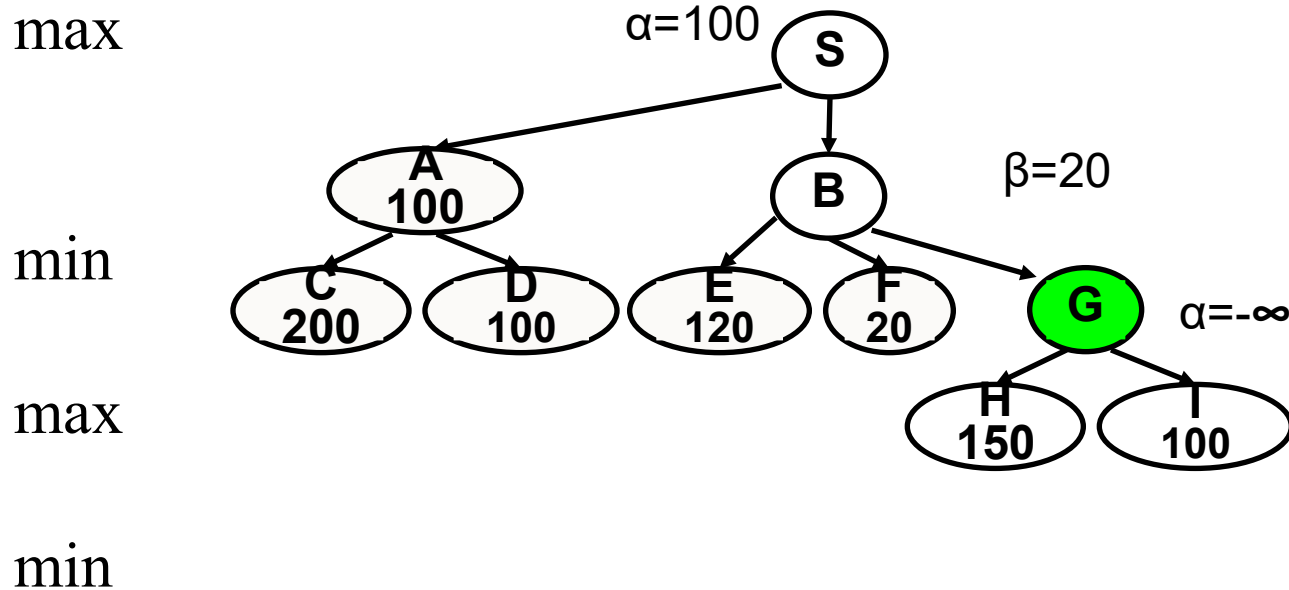


# Minimax algorithm in execution

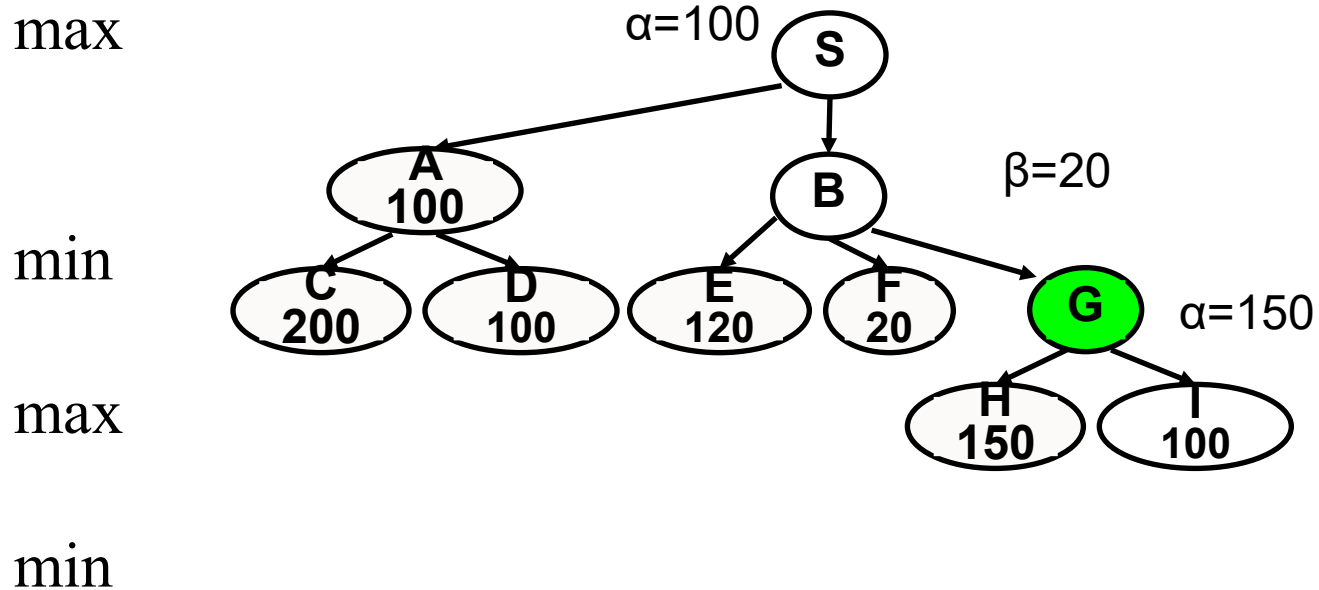




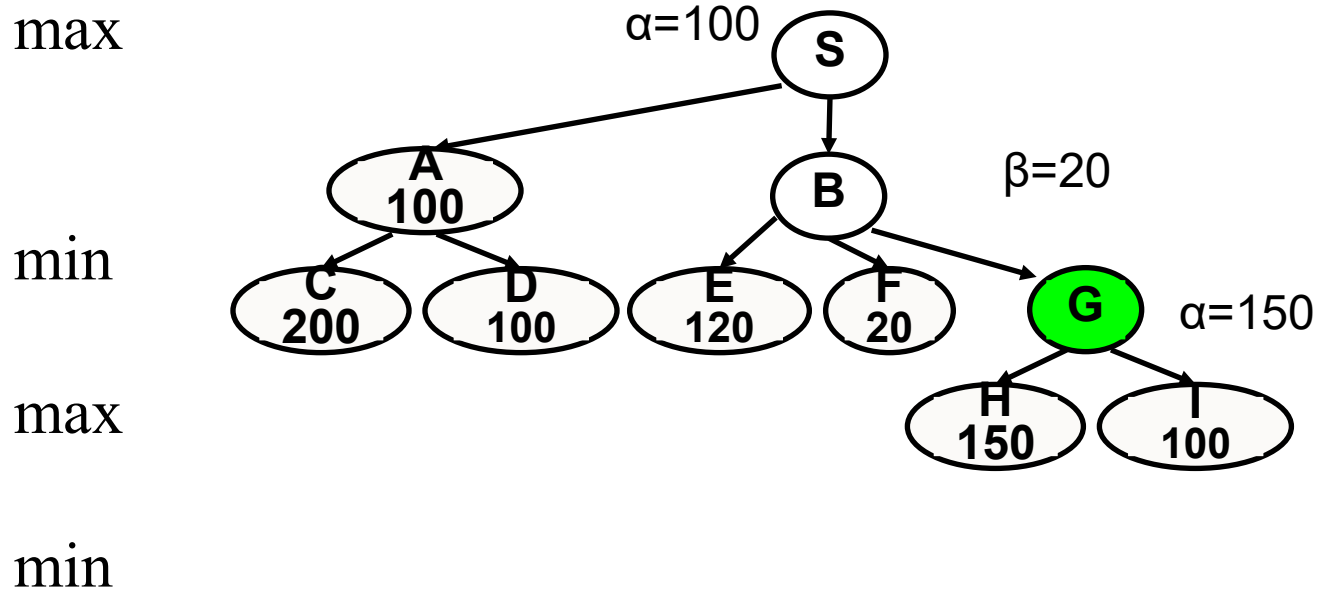
# Minimax algorithm in execution



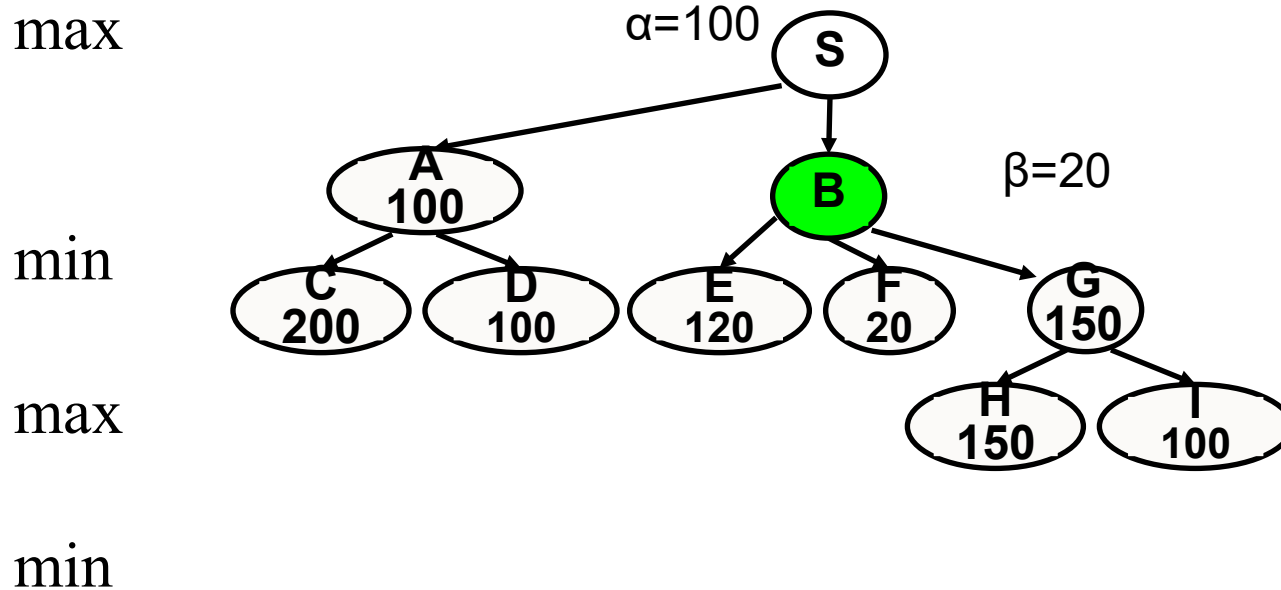
# Minimax algorithm in execution



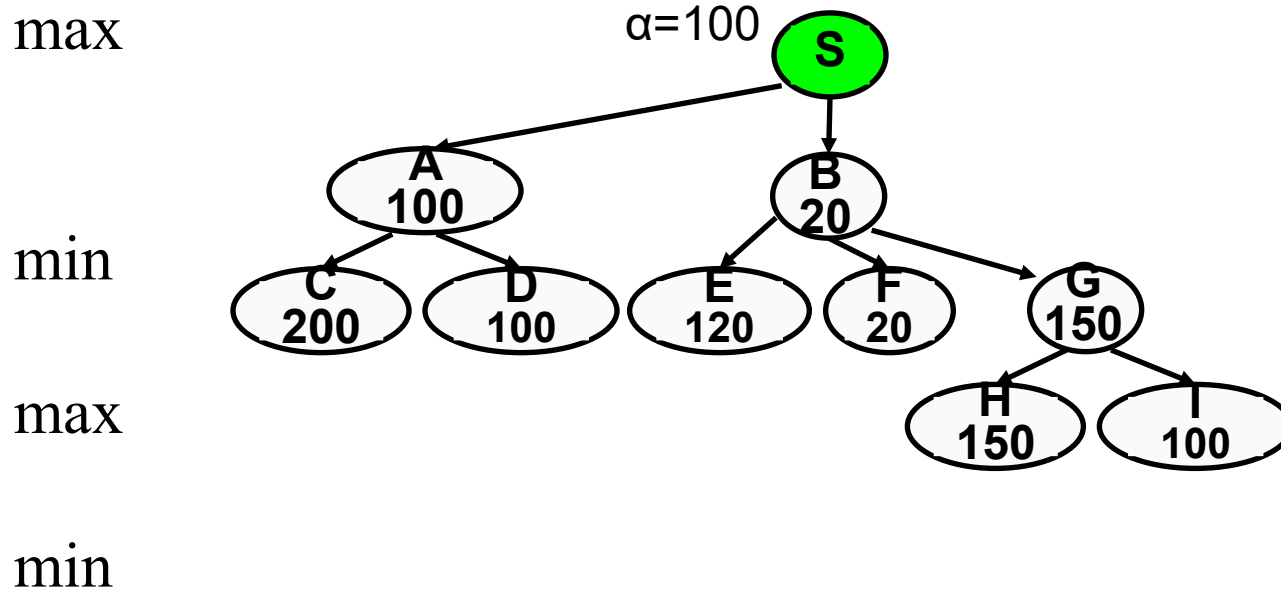
# Minimax algorithm in execution



# Minimax algorithm in execution



# Minimax algorithm in execution



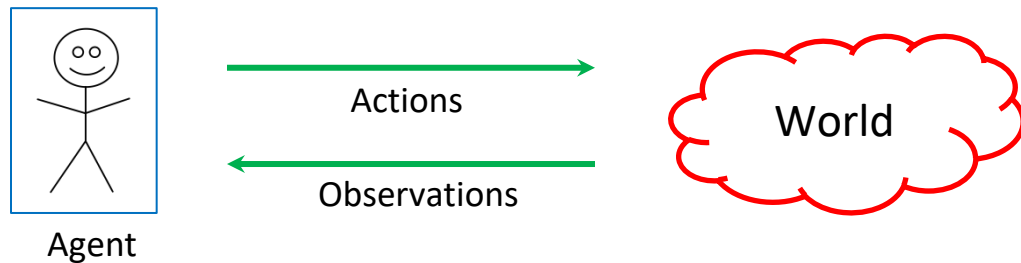


# Reinforcement Learning

# Building The Theoretical Model

Basic setup:

- Set of states,  $S$
- Set of actions  $A$
- Information: at time  $t$ , observe state  $s_t \in S$ . Get reward  $r_t$
- Agent makes choice  $a_t \in A$ . State changes to  $s_{t+1}$ , continue



Goal: find a map from **states to actions** maximize rewards.

# Markov Decision Process (MDP)

The formal mathematical model:

- **State set**  $S$ . Initial state  $s_0$ . **Action set**  $A$
- **State transition model:**  $P(s_{t+1} | s_t, a_t)$ 
  - Markov assumption: transition probability only depends on  $s_t$  and  $a_t$ , and not previous actions or states.
- **Reward function:**  $r(s_t)$
- **Policy:**  $\pi(s) : S \rightarrow A$ , action to take at a particular state.



# Discounting Rewards

One issue: these are infinite series. **Convergence?**

- Solution


$$U(s_0, s_1 \dots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \dots = \sum_{t \geq 0} \gamma^t r(s_t)$$

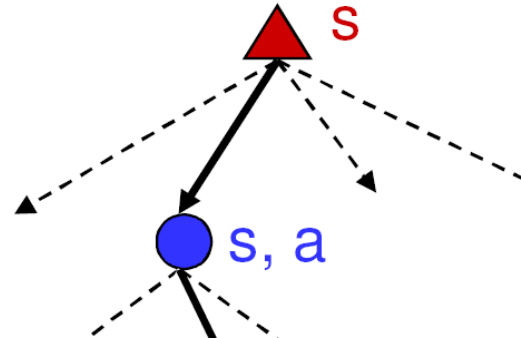
- Discount factor  $\gamma$  between 0 and 1
  - Set according to how important **present** is VS **future**

# Obtaining the Optimal Policy

Assume, we know the expected utility of an action.

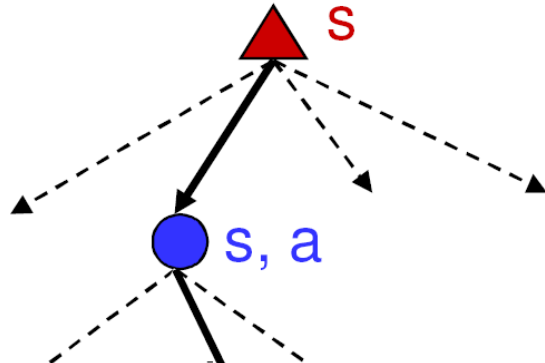
- So, to get the optimal policy, compute

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a) V^*(s')$$




# Bellman Equations

Let's walk over one step for the value function:



$$V^*(s) = \underset{\text{Current state reward}}{\color{green}r(s)} + \gamma \max_a \underbrace{\sum_{s'} P(s'|s, a) V^*(s')}_{\text{Discounted expected future rewards}}$$

Current state  
reward

Discounted expected  
future **rewards**

# Break & Quiz

Supposed you have the following information about an environment:

1. The discount factor is 0.8
2. The reward in  $s1$  taking action  $a1$  is 3
3. The transition probabilities are:  $P(s2|s1, a1) = 0.6$  and  $P(s3|s1, a1) = 0.4$

Currently,  $V(s2) = 10$  and  $V(s3) = 6$

Remember the update for value iteration is  $V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a) V_i(s')$

After a single iteration of value iteration, what is the value for state  $s1$  (what is  $V(s1)$ )?

- A. 8
- B. 10

# Break & Quiz

Supposed you have the following information about an environment:

1. The discount factor is 0.8
2. The reward in  $s1$  taking action  $a1$  is 3
3. The transition probabilities are:  $P(s2|s1, a1) = 0.6$  and  $P(s3|s1, a1) = 0.4$

Currently,  $V(s2) = 10$  and  $V(s3) = 6$

Remember the update for value iteration is  $V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a) V_i(s')$

After a single iteration of value iteration, what is the value for state  $s1$  (what is  $V(s1)$ )?  
Choose the closest option.

A. 8

B. 10

$V(s1) = 3 + 0.8(0.6 \times 10 + 0.4 \times 6) = 9.72 \approx 10$


# Q-Learning

- Our **next** reinforcement learning algorithm.
- Does not require knowing  $r$  or  $P$ . Learn from data of the form:  $\{(s_t, a_t, r_t, s_{t+1})\}$ .
- Learns an action-value function  $Q^*(s, a)$  that tells us the expected value of taking  $a$  in state  $s$ .
  - Note:  $V^*(s) = \max_a Q^*(s, a)$ .
- Optimal policy is formed as  $\pi^*(s) \equiv \operatorname{argmax}_a Q^*(s, a)$

# Q-Learning

Estimate  $Q^*(s, a)$  from data  $\{(s_t, a_t, r_t, s_{t+1})\}$ :

1. Initialize  $Q(.,.)$  arbitrarily (eg all zeros)
  1. Except terminal states  $Q(s_{\text{terminal}},.)=0$
2. Iterate over data until  $Q(.,.)$  converges:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$


# Example

A -10	G 10
S	B



# Example

$A^{-10}$	$G^{10}$
S	B

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

discount factor  $\gamma = 0.9$

learning rate  $\alpha = 0.1$

	left	right	up	down
S			-1	
A				
B				

$$Q(S, up) = (1 - 0.1) * 0 + 0.1(-10) = -1$$

# Example

$A^{-10}$	$G^{10}$
S	B

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

discount factor  $\gamma = 0.9$

learning rate  $\alpha = 0.1$

	left	right	up	down
S		0	-1	
A				
B				

$$Q(S, right) = (1 - 0.1) * 0 + 0.1(0 + 0.9 * 0) = 0$$

# Example

$A^{-10}$	$G^{10}$
S	B

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

discount factor  $\gamma = 0.9$

learning rate  $\alpha = 0.1$

	left	right	up	down
S		0	-1	
A				
B			1	

$$Q(S, right) = (1 - 0.1) * 0 + 0.1(0 + 0.9 * 0) = 0$$

$$Q(B, up) = (1 - 0.1) * 0 + 0.1(10) = 1$$

# Example

$A^{-10}$	$G^{10}$
S	B

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

discount factor  $\gamma = 0.9$

learning rate  $\alpha = 0.1$

	left	right	up	down
S		0.09	-1	
A				
B			1.9	

$$Q(S, right) = (1 - 0.1) * 0 + 0.1(0 + 0.9 * 1) = 0.09$$

$$Q(B, up) = (1 - 0.1) * 1 + 0.1(10) = 1.9$$

# Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
  - **Pros:**
    - Get a more accurate model of the environment
    - Discover higher-reward states than the ones found so far
  - **Cons:**
    - When exploring, not maximizing your utility
    - Something bad might happen
- **Exploitation:** go with the best strategy found so far
  - **Pros:**
    - Maximize reward as reflected in the current utility estimates

# Q-Learning: $\epsilon$ -Greedy Behavior Policy

Getting data with both **exploration** and **exploitation**

- With probability  $\epsilon$ , take a random action; else the action with the highest (current)  $Q(s, a)$  value.

$$a = \begin{cases} \operatorname{argmax}_{a \in A} Q(s, a) & \text{uniform}(0, 1) > \epsilon \\ \text{random } a \in A & \text{otherwise} \end{cases}$$

# Q-learning Algorithm

Input: step size  $\alpha$ , exploration probability  $\epsilon$

1. set  $Q(s,a) = 0$  for all  $s, a$ .
2. For each episode:
3. Get initial state  $s$ .
4. While ( $s$  not a terminal state):
  - 5. Perform  $a = \epsilon$ -greedy( $Q, s$ ), receive  $r, s'$
  - 6.  $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$

Explore: take action to  
see what happens.

Update action-value



Thank you and good luck!