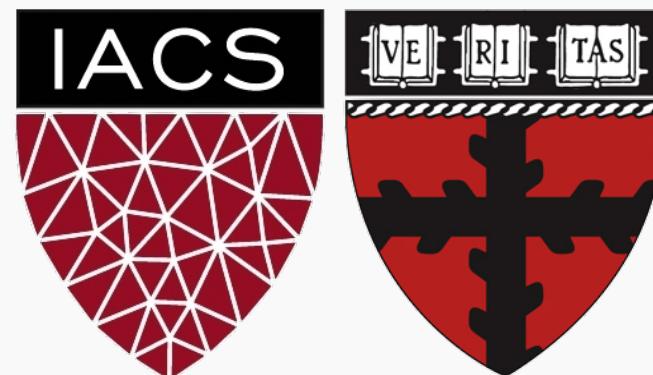


# Lecture 18: Anatomy of NN

CS109A Introduction to Data Science  
Pavlos Protopapas and Kevin Rader



# Outline

---

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture



# Outline

---

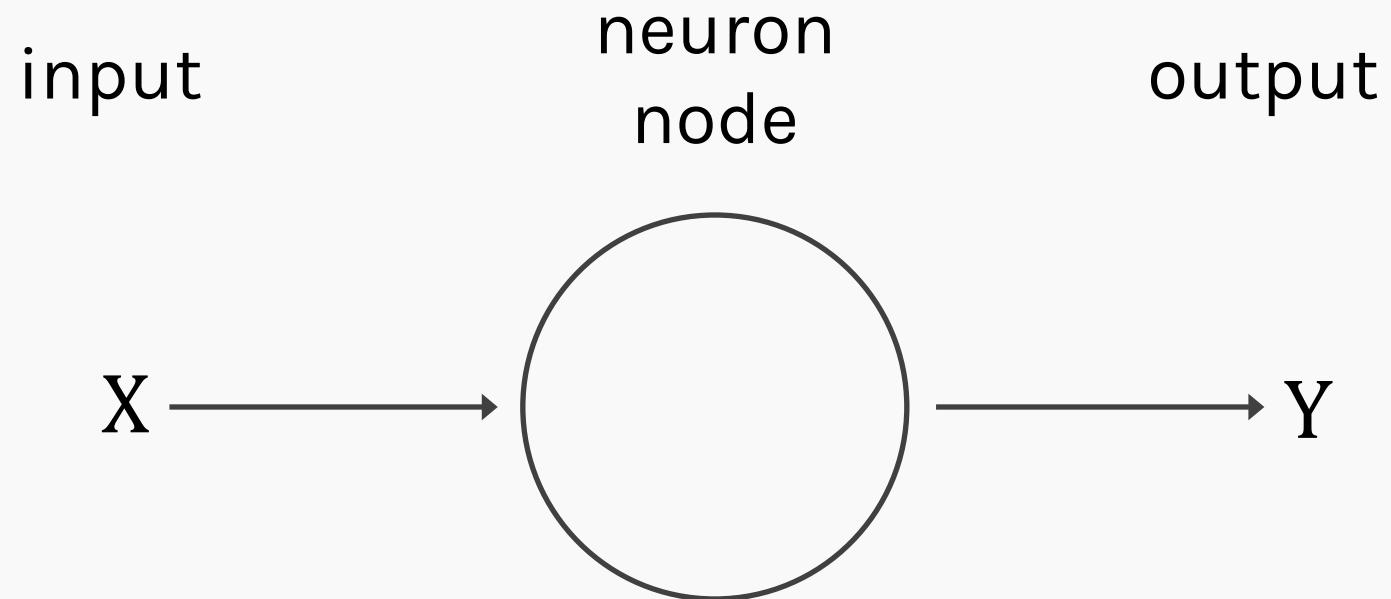
## Anatomy of a NN

### Design choices

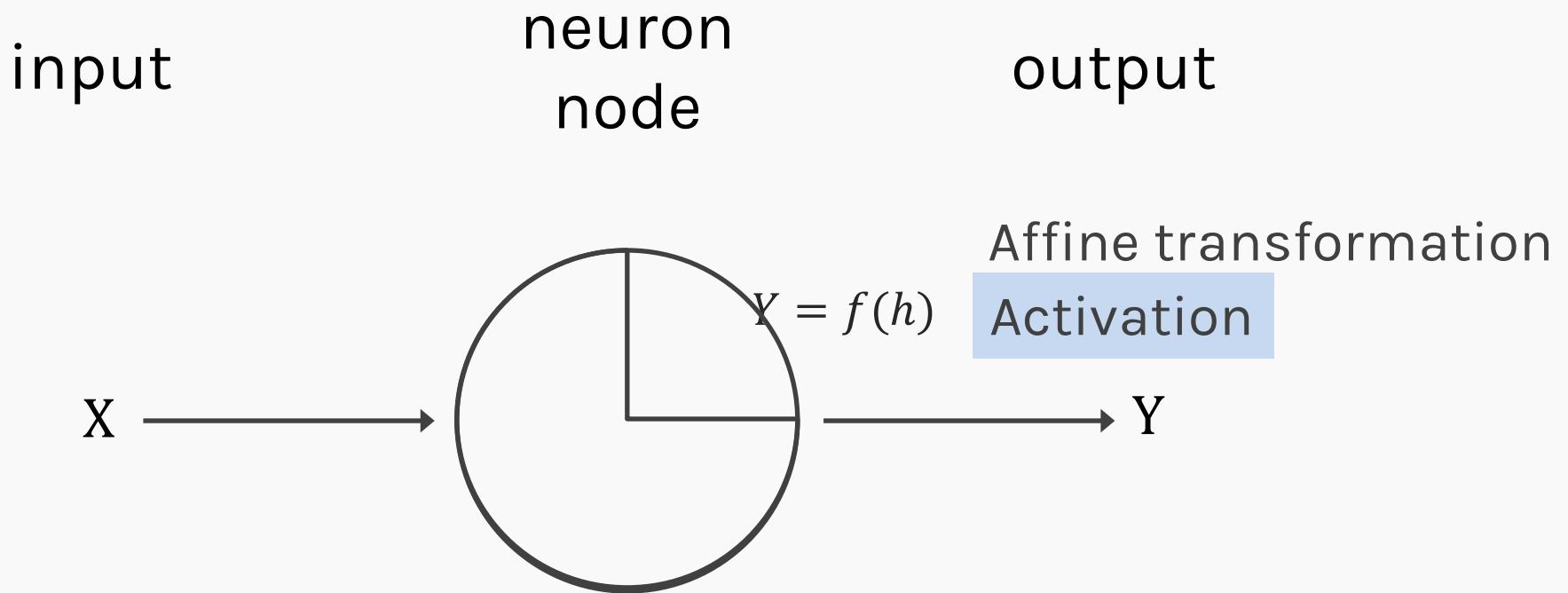
- Activation function
- Loss function
- Output units
- Architecture



# Anatomy of artificial neural network (ANN)

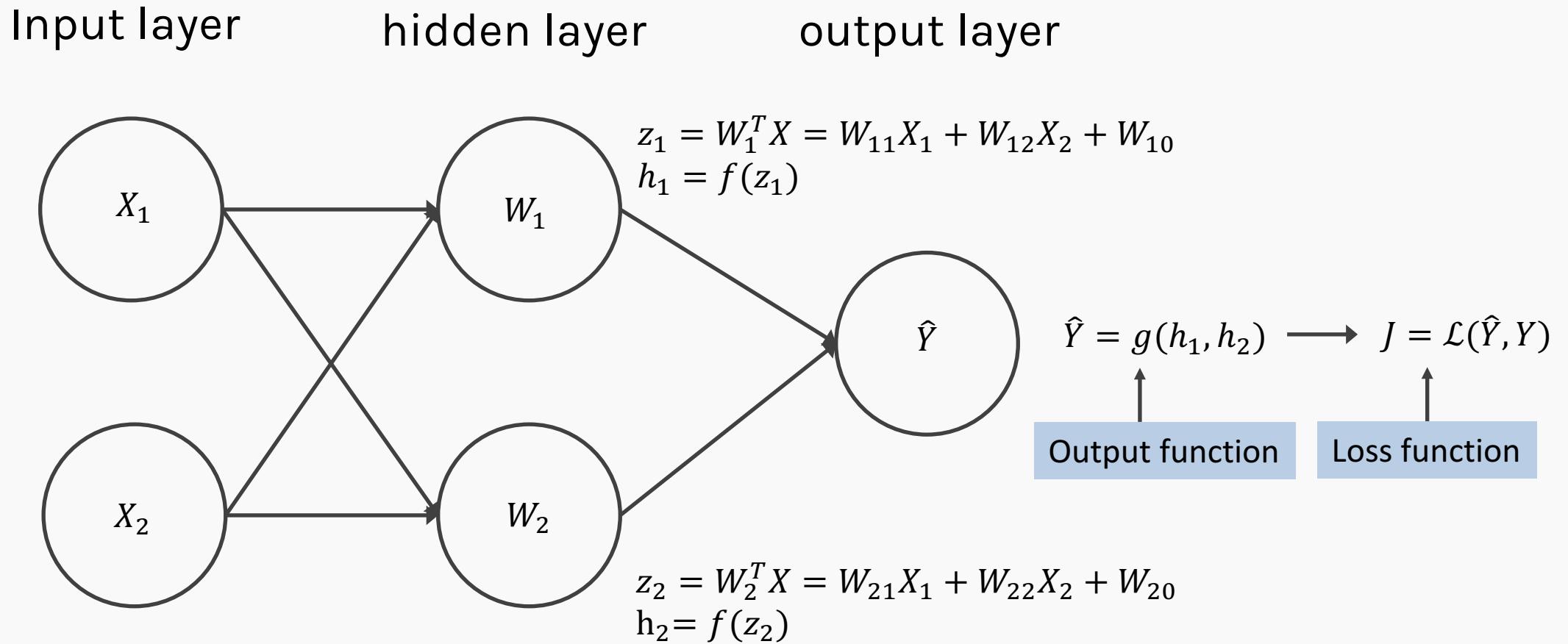


# Anatomy of artificial neural network (ANN)



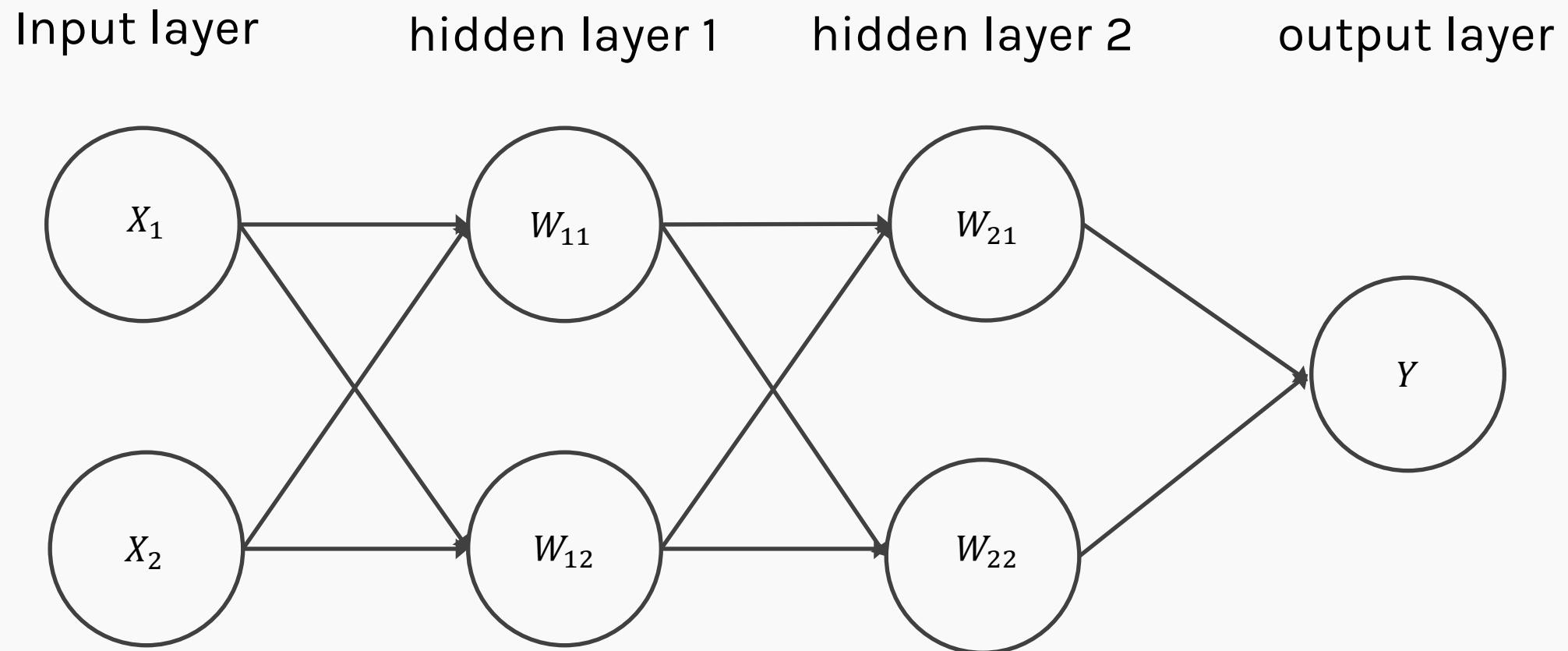
We will talk later about the choice of activation function. So far we have only talked about sigmoid as an activation function but there are other choices.

# Anatomy of artificial neural network (ANN)

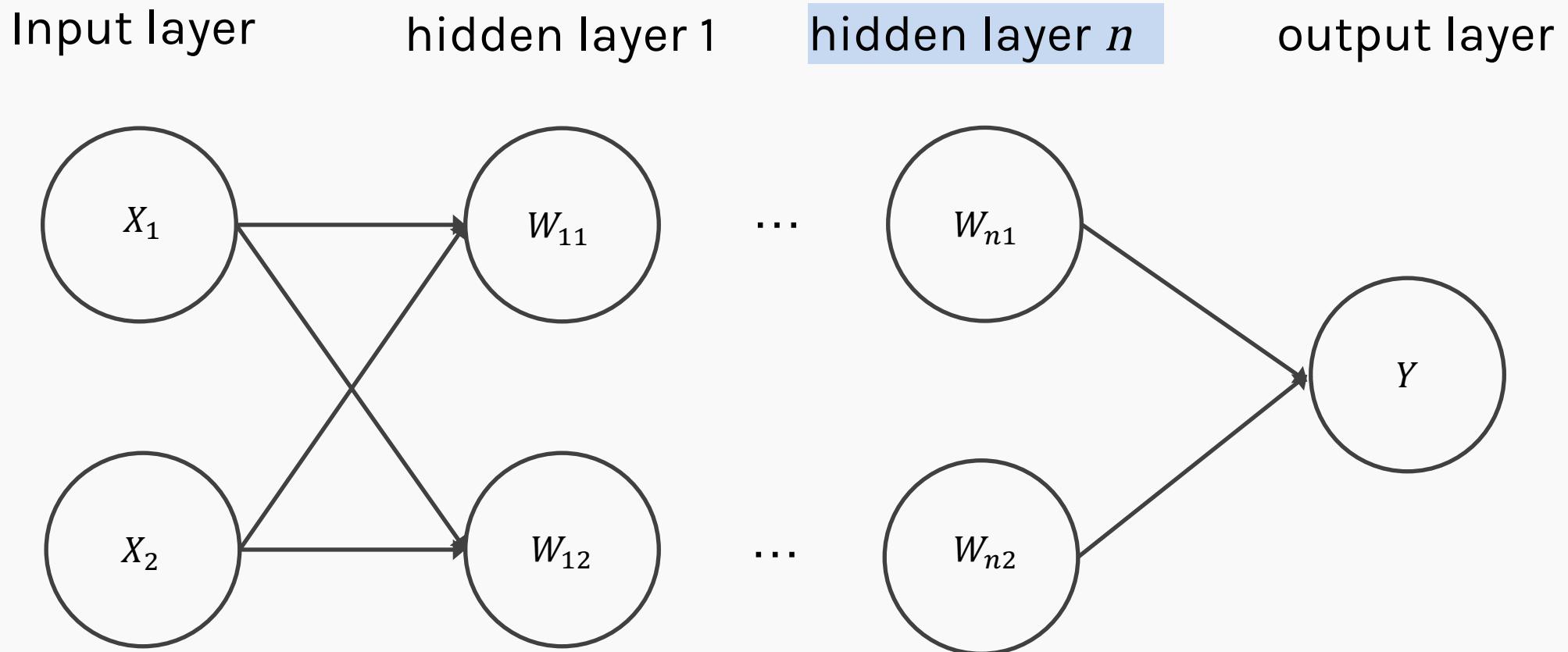


We will talk later about the choice of the output layer and the loss function. So far we consider sigmoid as the output and log-bernoulli.

# Anatomy of artificial neural network (ANN)



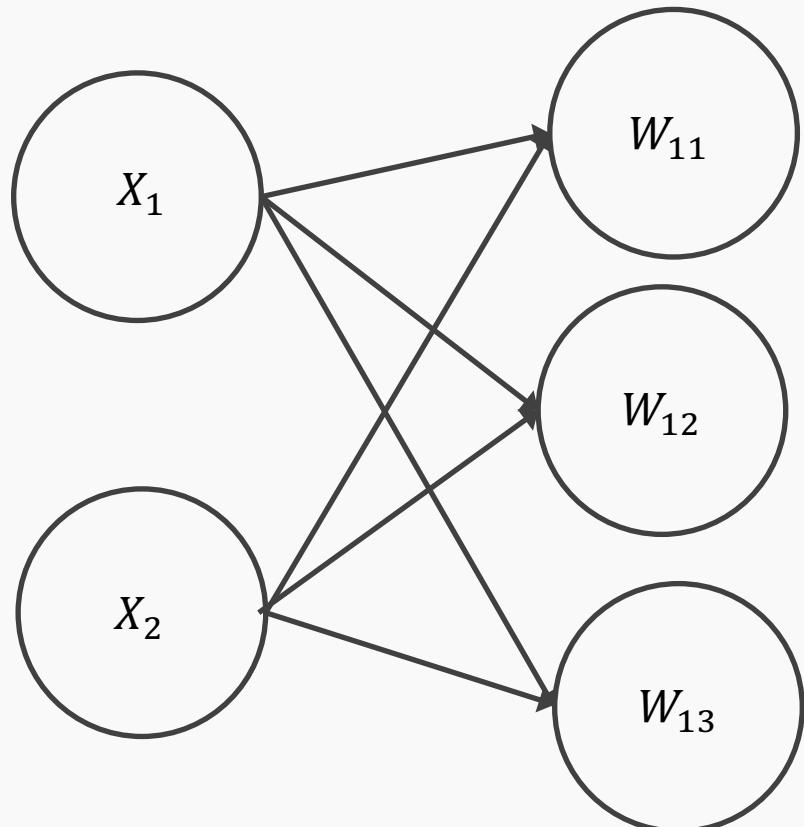
# Anatomy of artificial neural network (ANN)



We will talk later about the choice of the number of layers.

# Anatomy of artificial neural network (ANN)

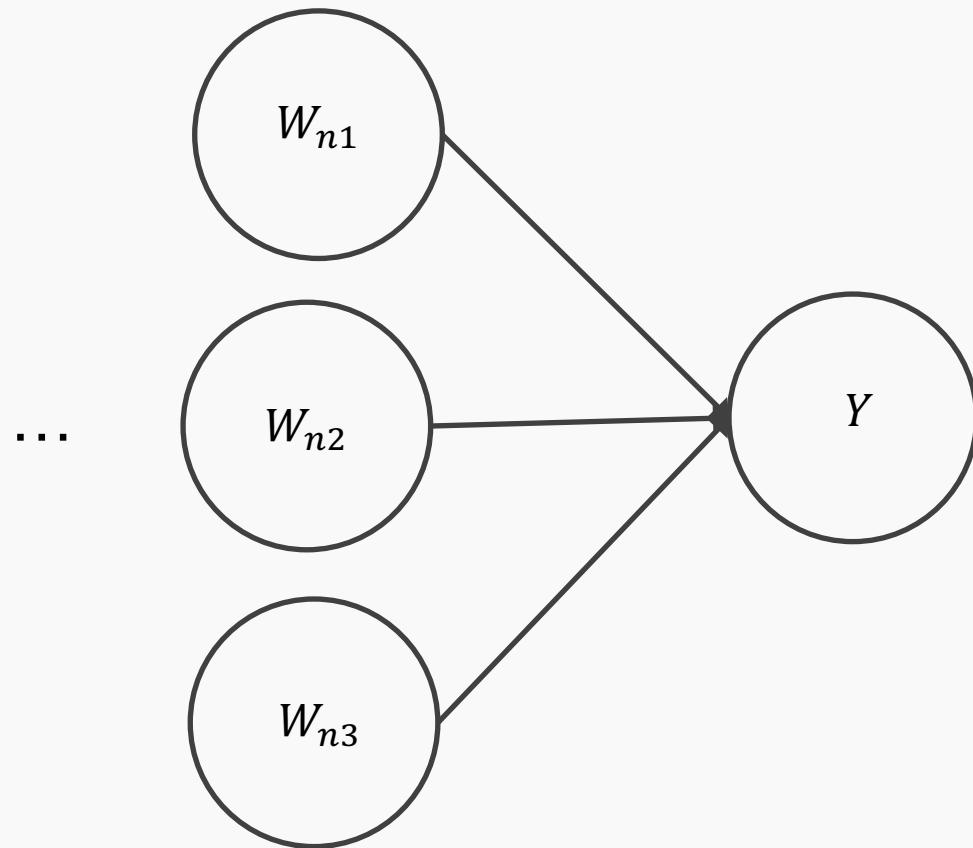
Input layer



hidden layer 1,  
3 nodes

hidden layer  $n$   
3 nodes

output layer



# Anatomy of artificial neural network (ANN)

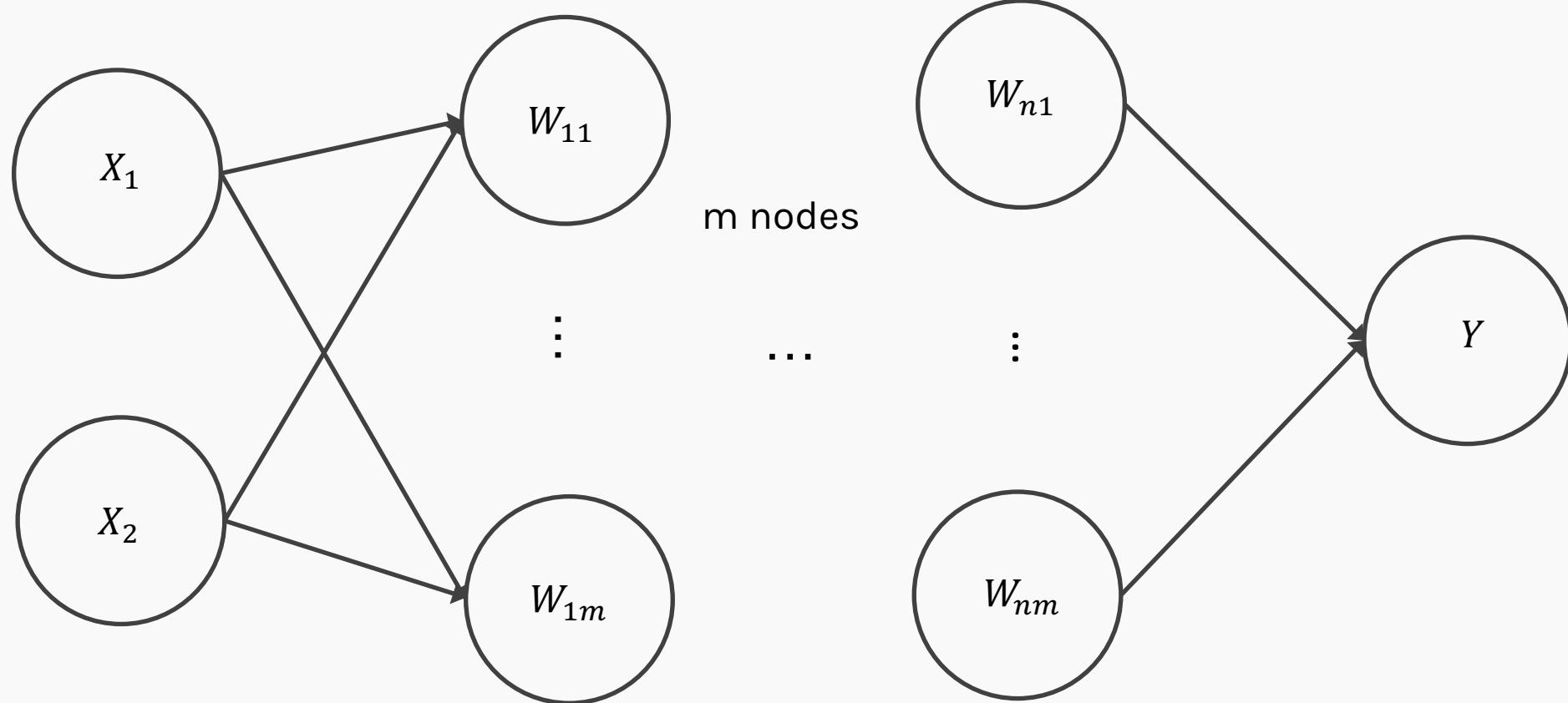
Input layer

hidden layer 1,

hidden layer  $n$

output layer

$m$  nodes



We will talk later about the choice of the number of nodes.

# Anatomy of artificial neural network (ANN)

Input layer

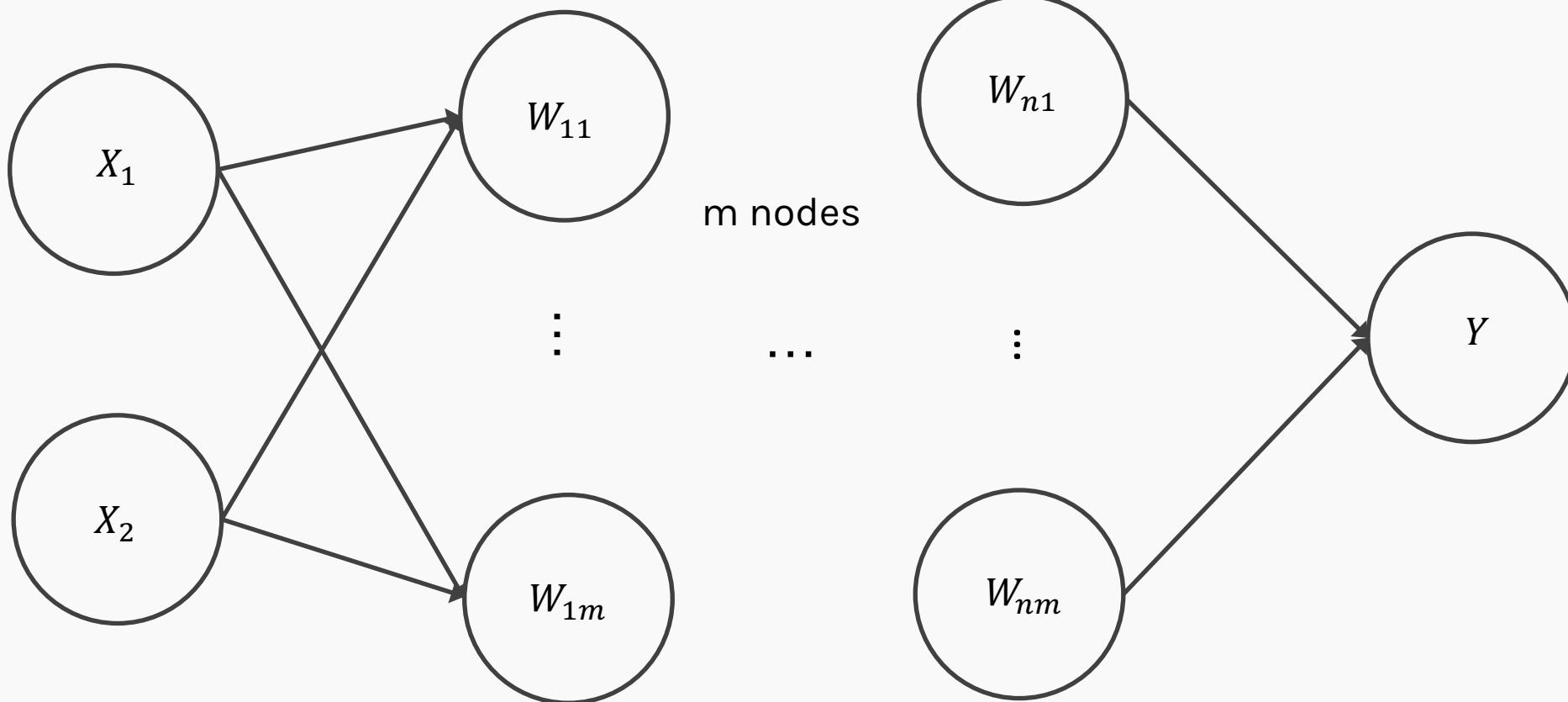
hidden layer 1,

hidden layer  $n$

output layer

$m$  nodes

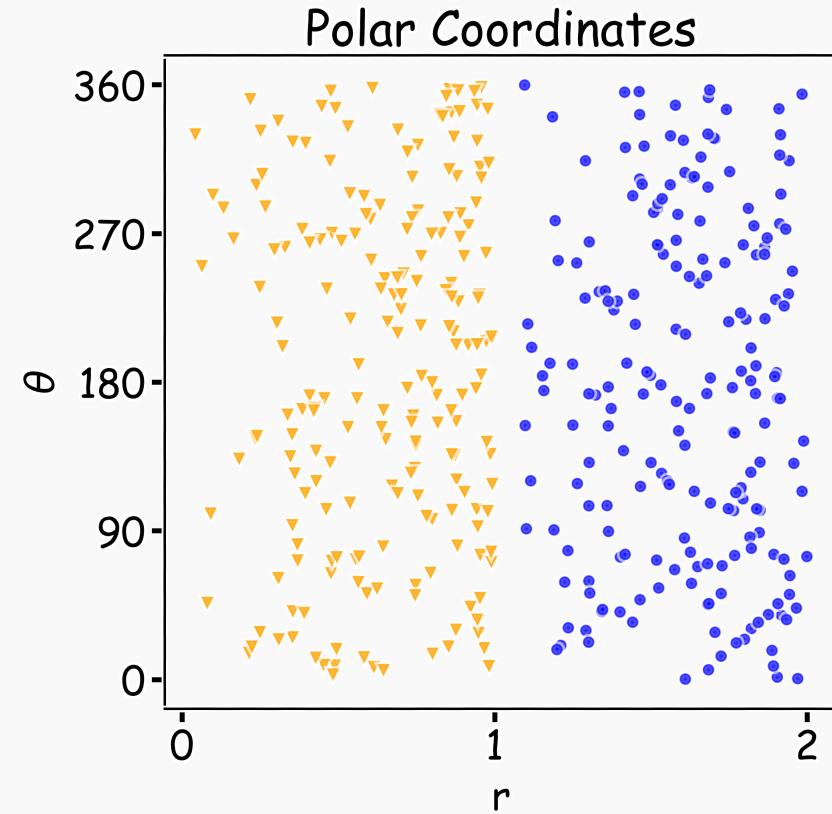
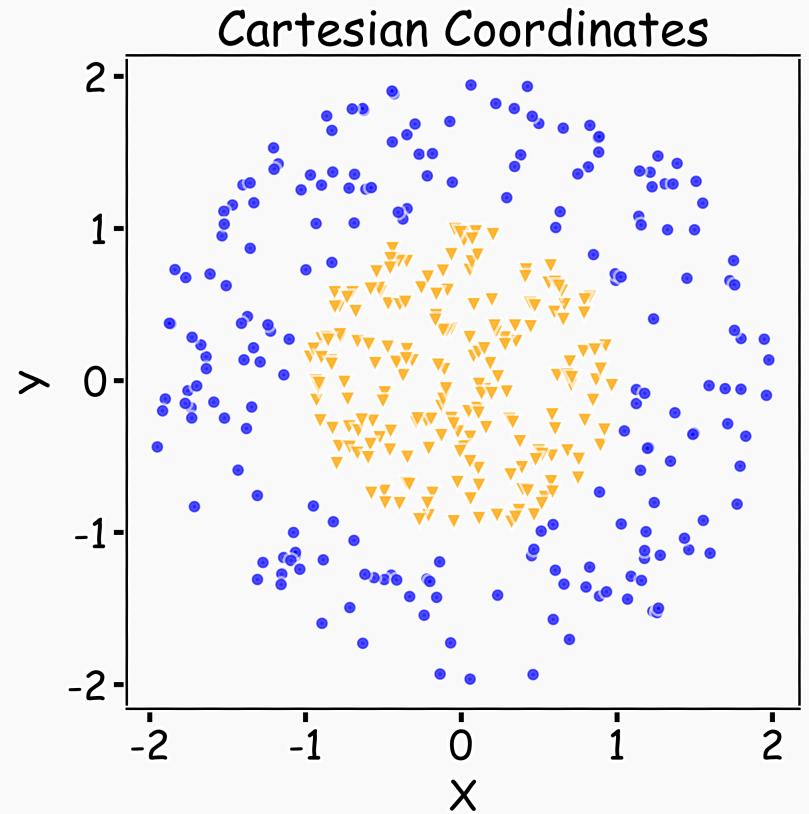
Number of inputs  $d$



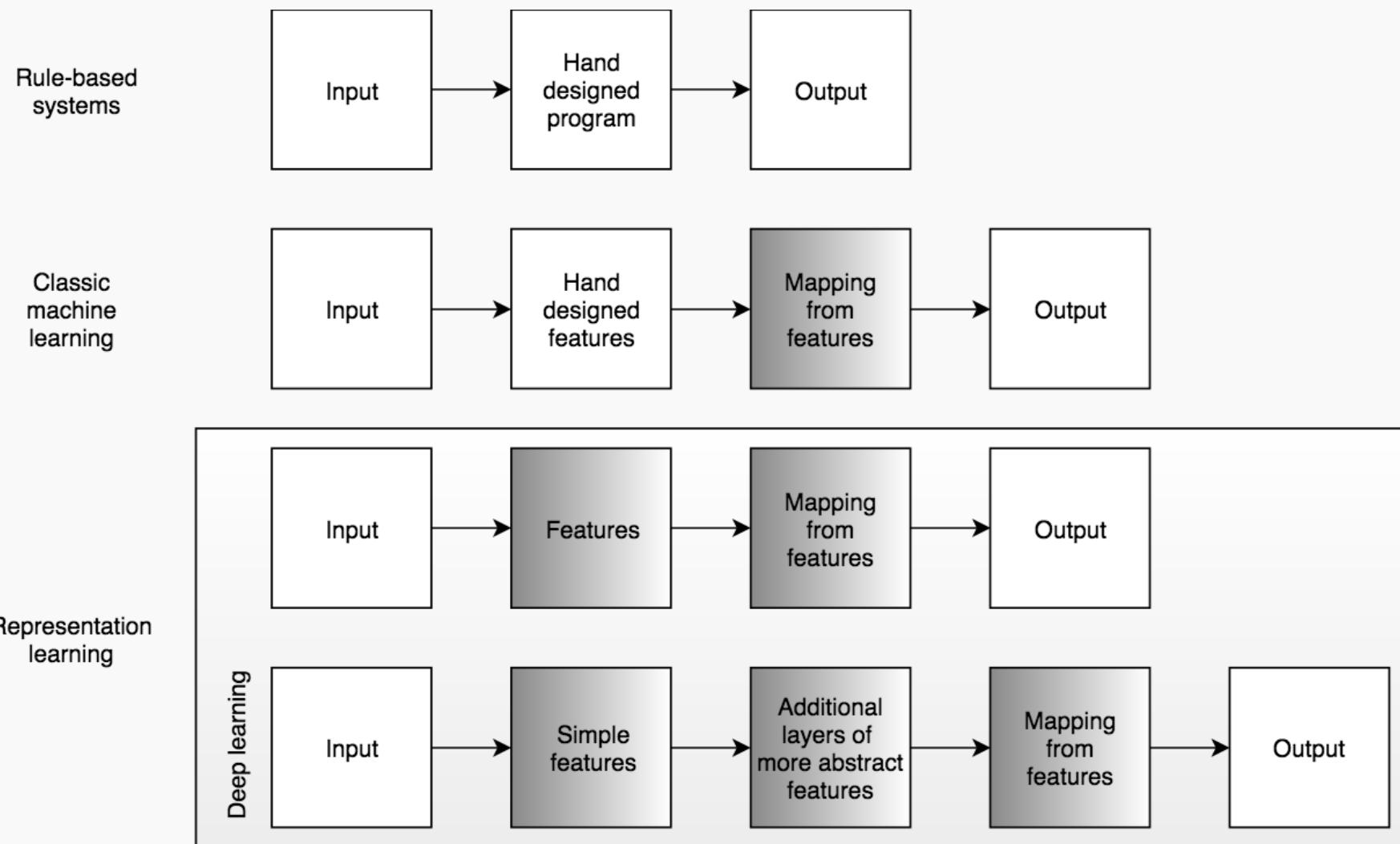
Number of inputs is specified by the data

# Why layers? Representation

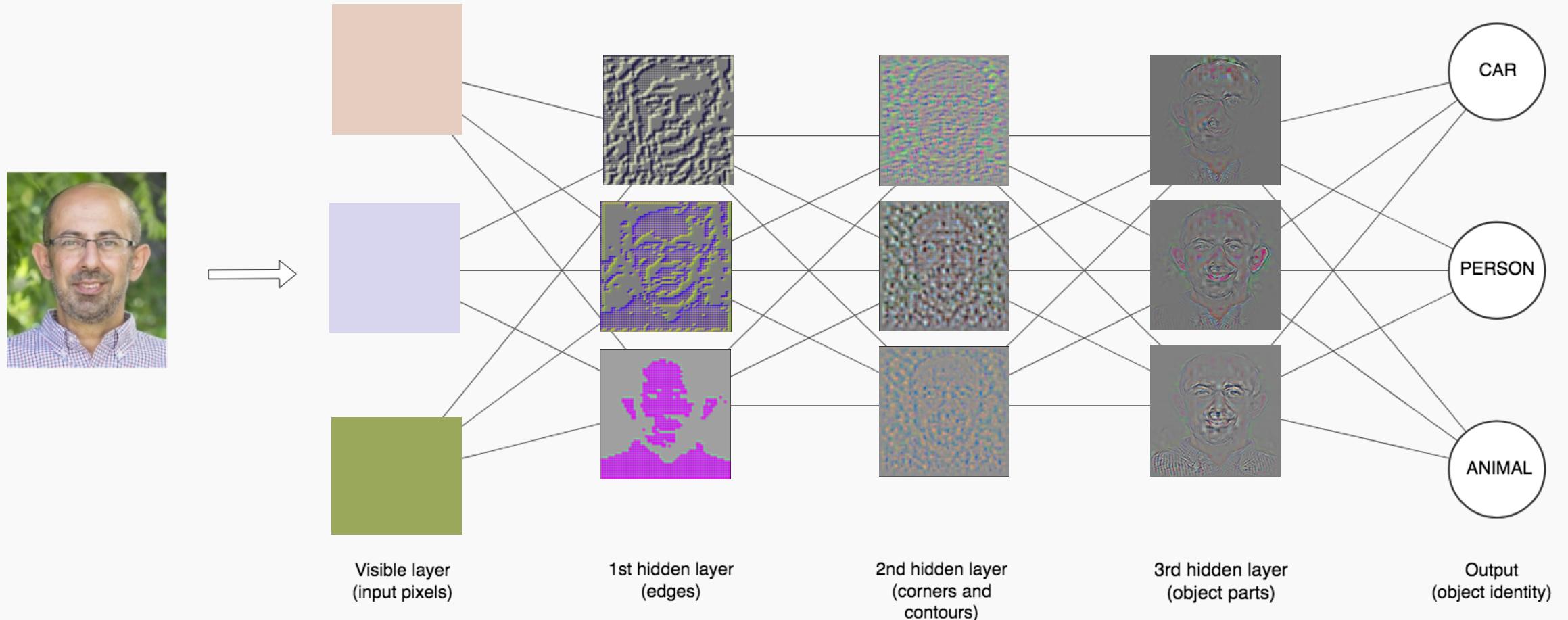
Representation matters!



# Learning Multiple Components



# Depth = Repeated Compositions

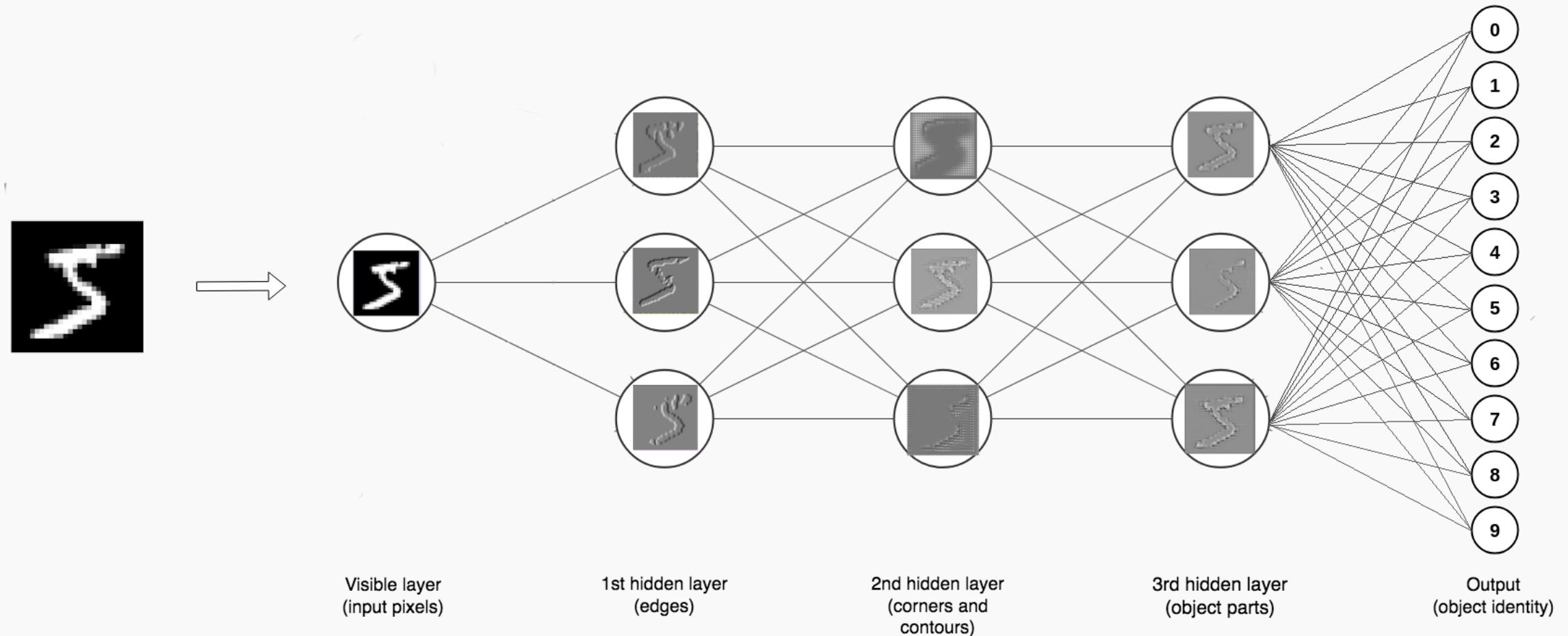


# Neural Networks

Hand-written digit recognition: MNIST data



# Depth = Repeated Compositions



# Outline

---

## Anatomy of a NN

## Design choices

- Activation function
- Loss function
- Output units
- Architecture



# Outline

---

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture



# Activation function

---

$$h = f(W^T X + b)$$

The activation function should:

- Ensures non-linearity
- Ensure gradients remain large through hidden unit

Common choices are

- Sigmoid
- Relu, leaky ReLU, Generalized ReLU, MaxOut
- softplus
- tanh
- swish

# Activation function

---

$$h = f(W^T X + b)$$

The activation function should:

- Ensures **not linearity**
- Ensure gradients remain large through hidden unit

Common choices are

- Sigmoid
- Relu, leaky ReLU, Generalized ReLU, MaxOut
- softplus
- tanh
- swish

# Beyond Linear Models

---

Linear models:

- Can be fit efficiently (via convex optimization)
- Limited model capacity

Alternative:

$$f(x) = w^T \phi(x)$$

Where  $\phi$  is a *non-linear transform*

# Traditional ML

---

## Manually engineer $\phi$

- Domain specific, enormous human effort

## Generic transform

- Maps to a higher-dimensional space
- Kernel methods: e.g. RBF kernels
- Over fitting: does not generalize well to test set
- Cannot encode enough prior information



# Deep Learning

---

- Directly learn  $\phi$

$$f(x; \theta) = W^T \phi(x; \theta)$$

- where  $\theta$  are parameters of the transform
- $\phi$  defines hidden layers
- Non-convex optimization
- Can encode prior beliefs, generalizes well



# Activation function

---

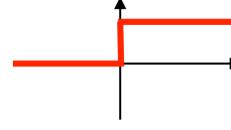
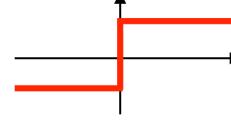
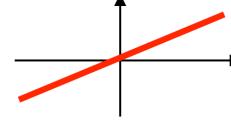
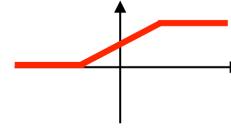
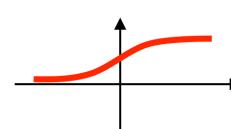
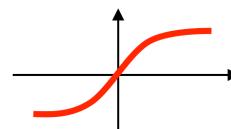
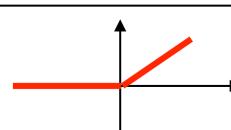
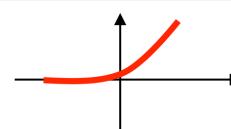
$$h = f(W^T X + b)$$

The activation function should:

- Ensures **not linearity**
- Ensure gradients remain large through hidden unit

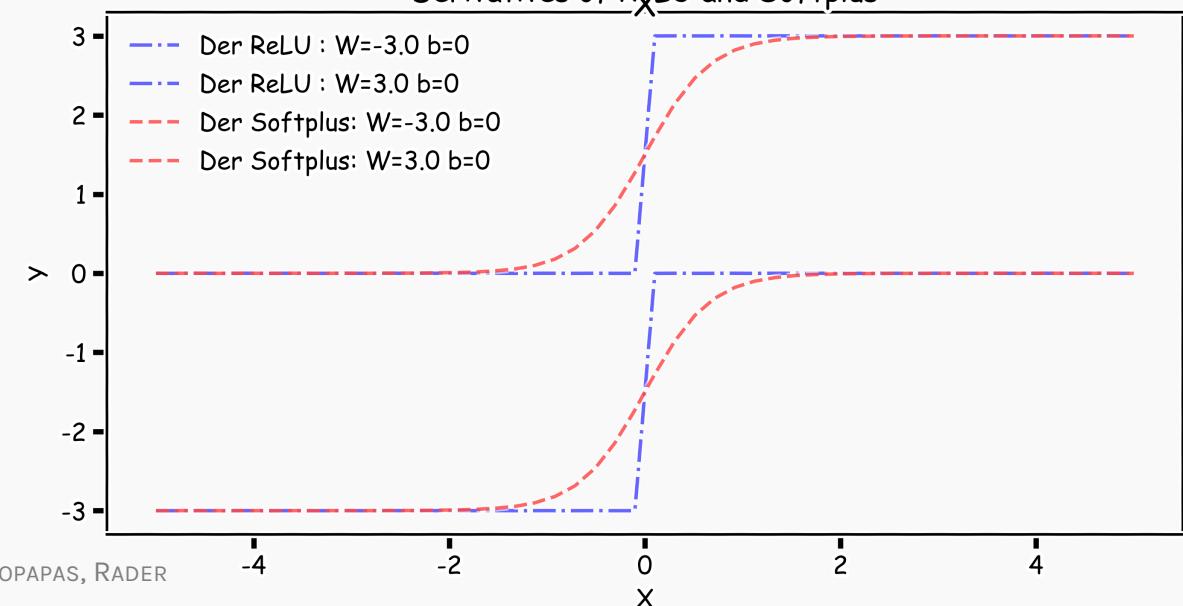
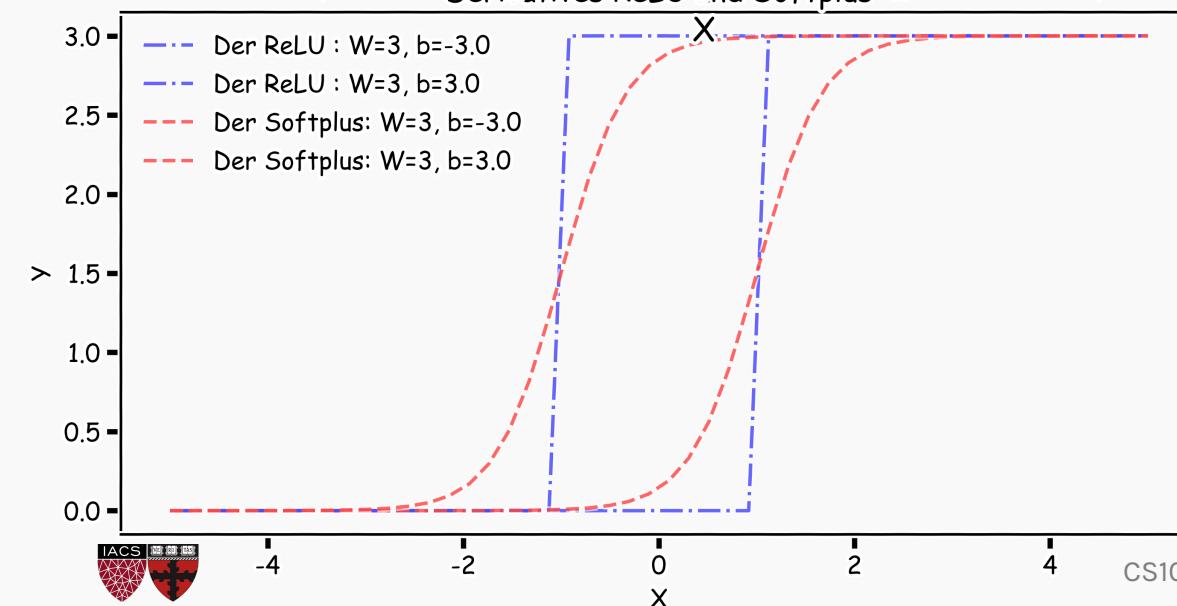
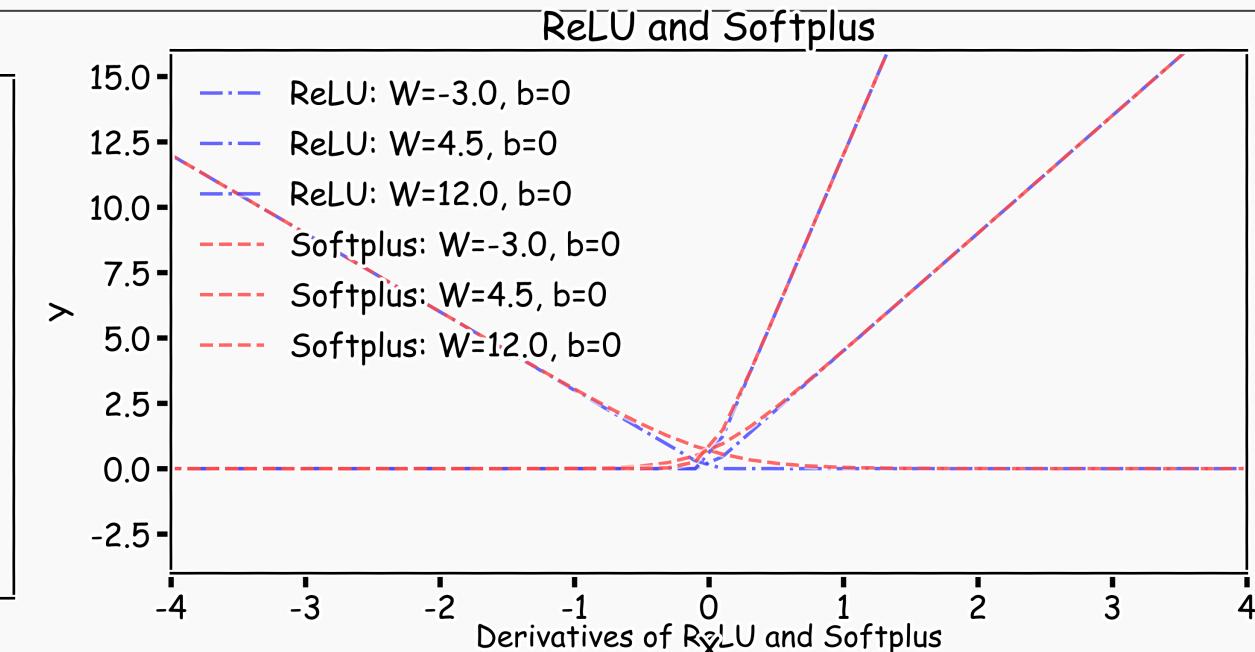
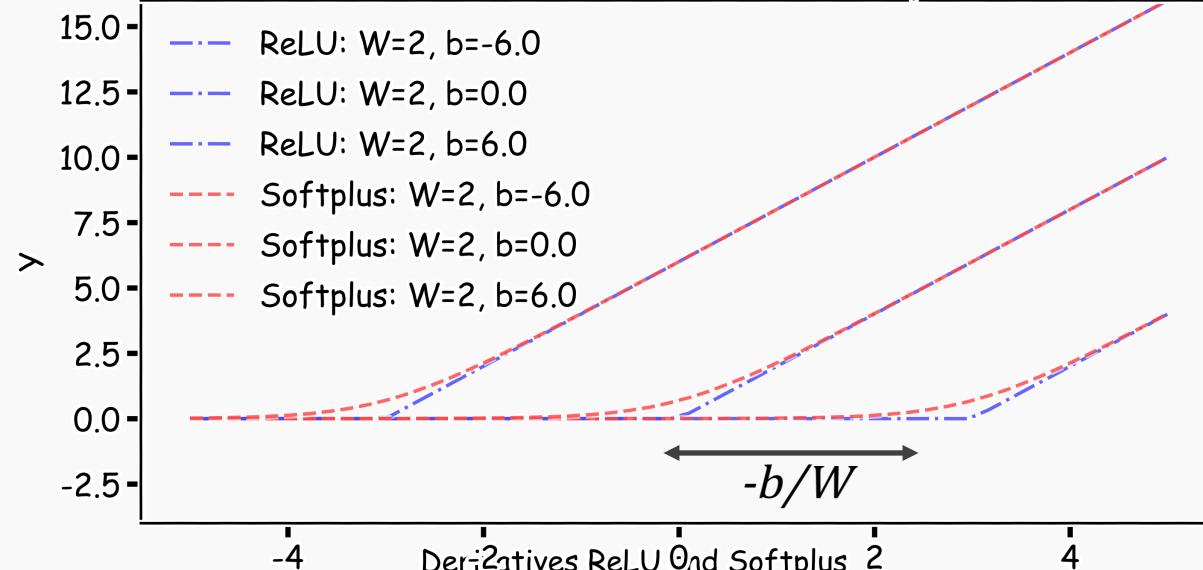
Common choices are

- Sigmoid
- Relu, leaky ReLU, Generalized ReLU, MaxOut
- softplus
- tanh
- swish

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

# ReLU and Softplus

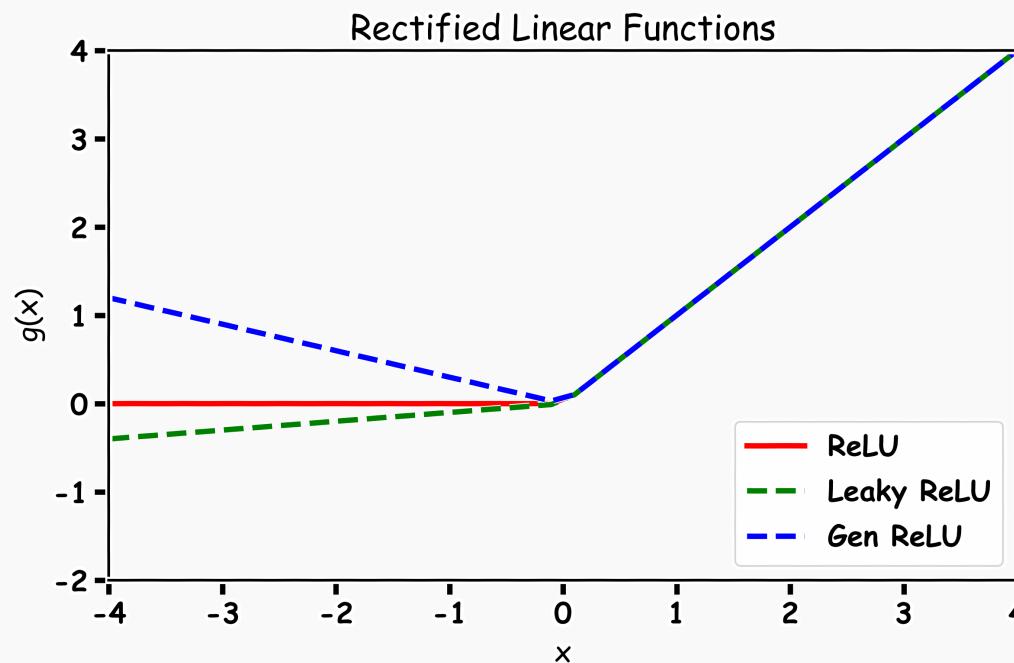
Derivatives of ReLU and Softplus



# Generalized ReLU

Generalization: For  $\alpha_i > 0$

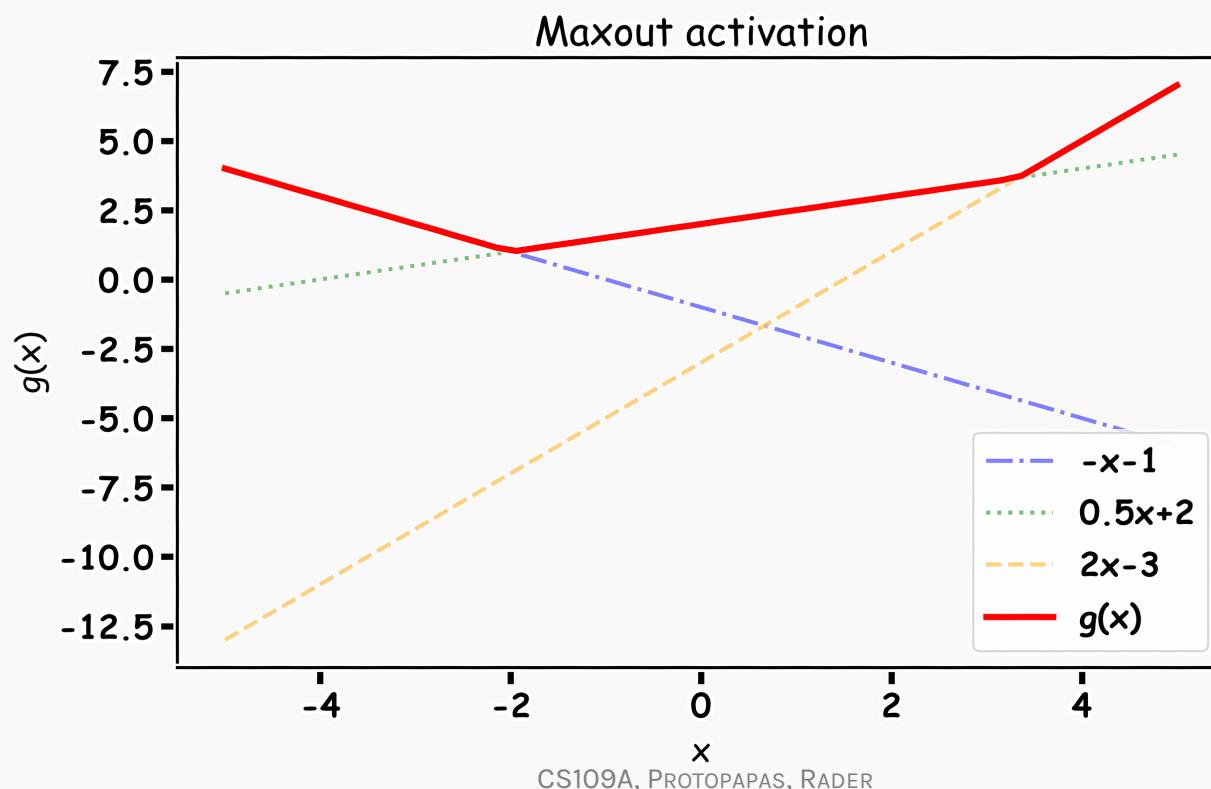
$$g(x_i, \alpha) = \max\{a, x_i\} + \alpha \min\{0, x_i\}$$



# Maxout

Max of  $k$  linear functions. Directly learn the activation function.

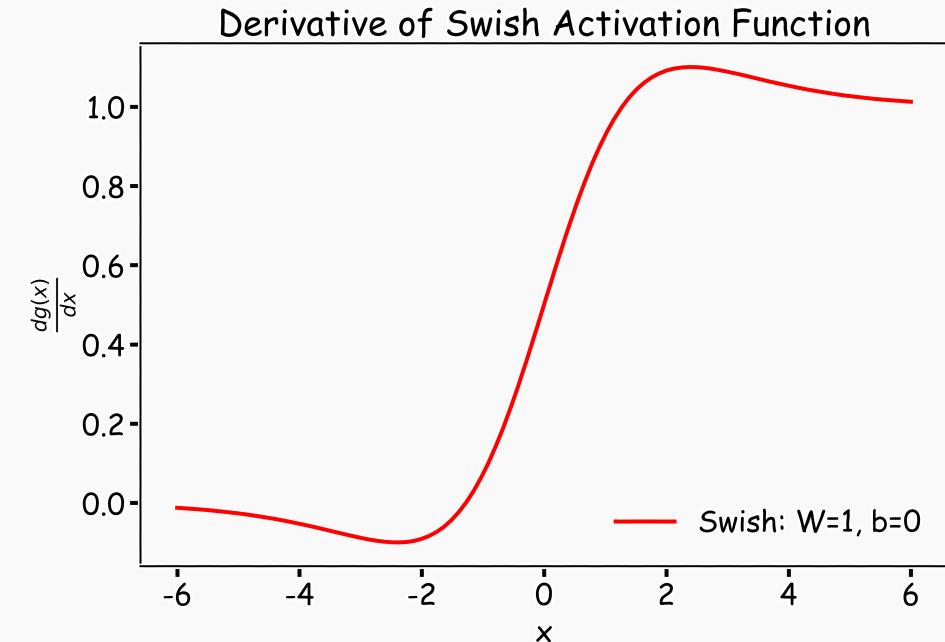
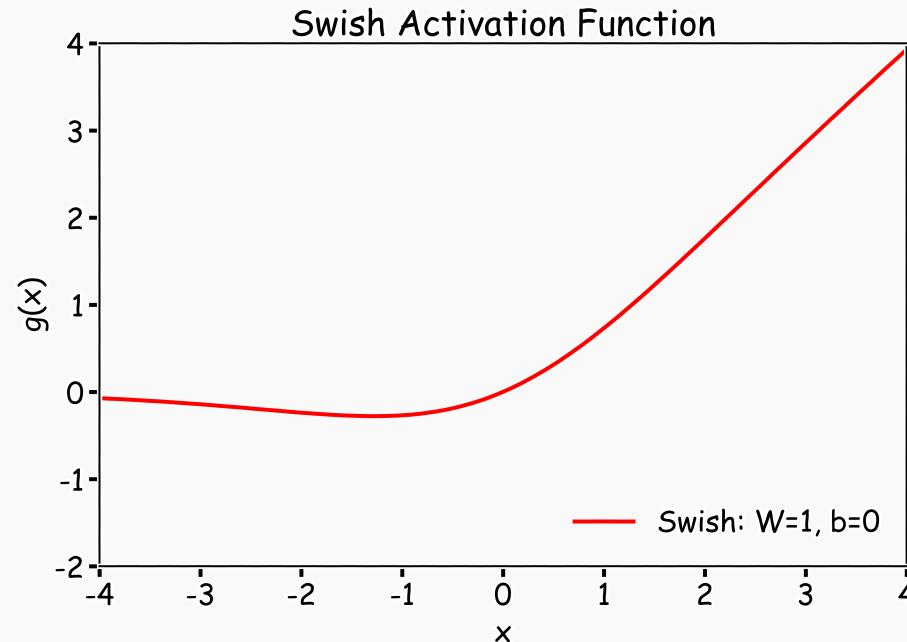
$$g(x) = \max_{i \in \{1, \dots, k\}} \alpha_i x_i + \beta$$



# Swish: A Self-Gated Activation Function

Currently, the most successful and widely-used activation function is the ReLU. Swish tends to work better than ReLU on deeper models across a number of challenging datasets.

$$g(x) = x \sigma(x)$$



# Outline

---

Anatomy of a NN

Design choices

- Activation function
- **Loss function**
- Output units
- Architecture



# Loss Function

---

Cross-entropy between training data and model distribution (i.e. negative log-likelihood)

$$J(W) = -\mathbb{E}_{x,y \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(y|x)$$

Do not need to design separate loss functions.

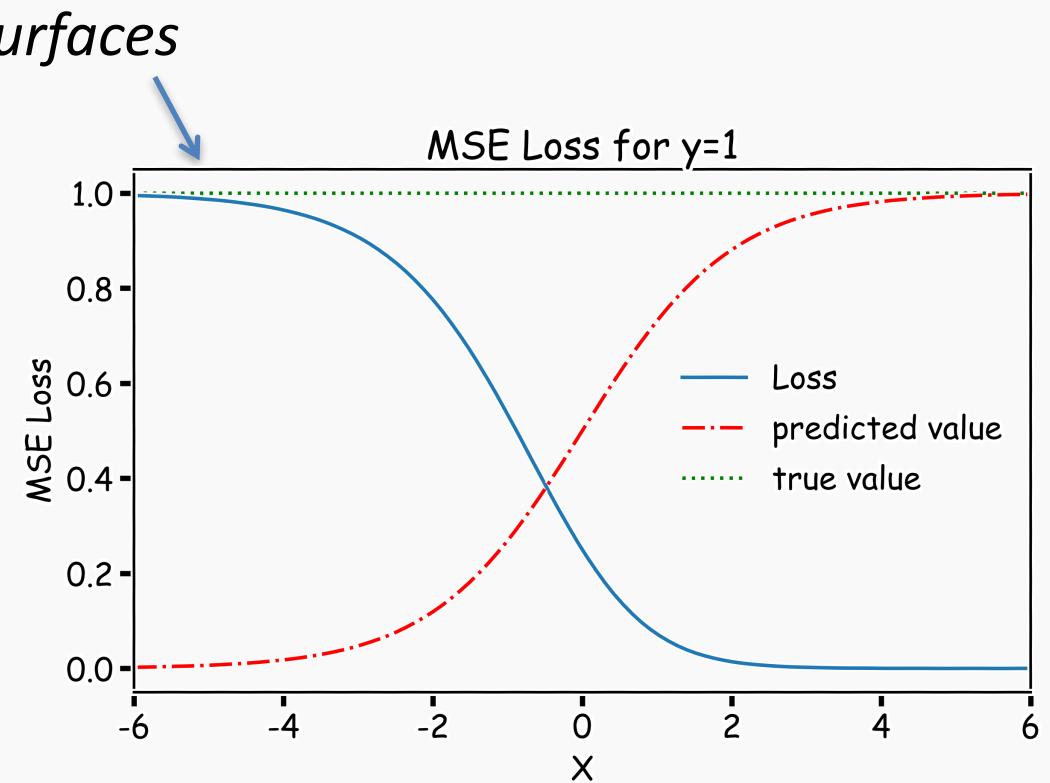
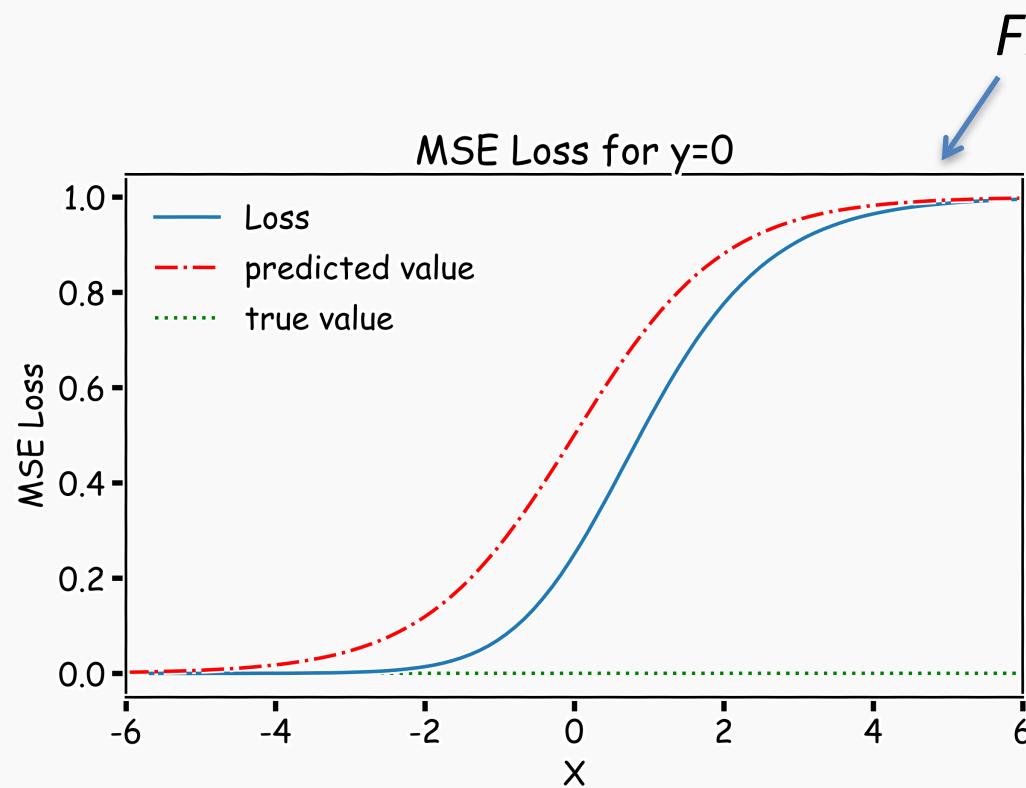
Gradient of cost function must be large enough



# Loss Function

Example: sigmoid output + squared loss

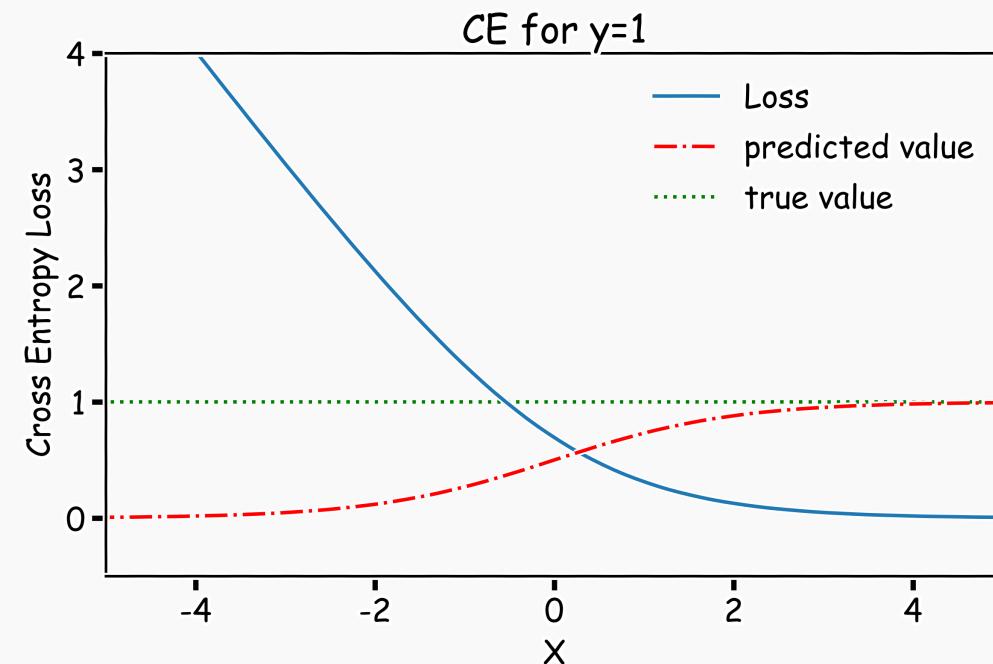
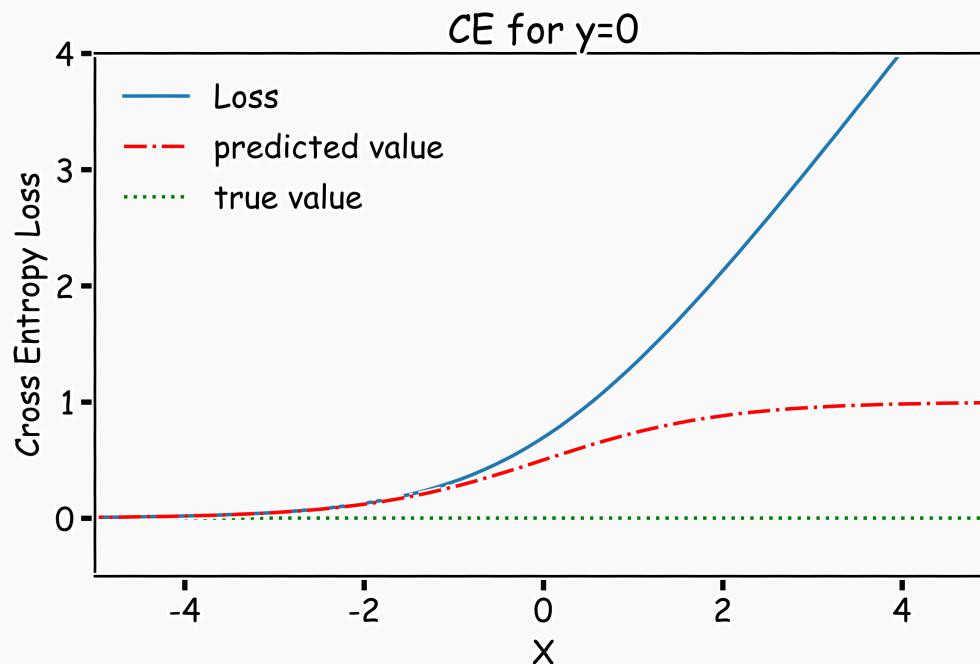
$$L_{sq} = (y - \hat{y})^2 = (y - \sigma(x))^2$$



# Cost Function

Example: sigmoid output + cross-entropy loss

$$L_{ce}(y, \hat{y}) = -\{ y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \}$$



# Design Choices

---

Activation function

Loss function

Output units

Architecture

Optimizer



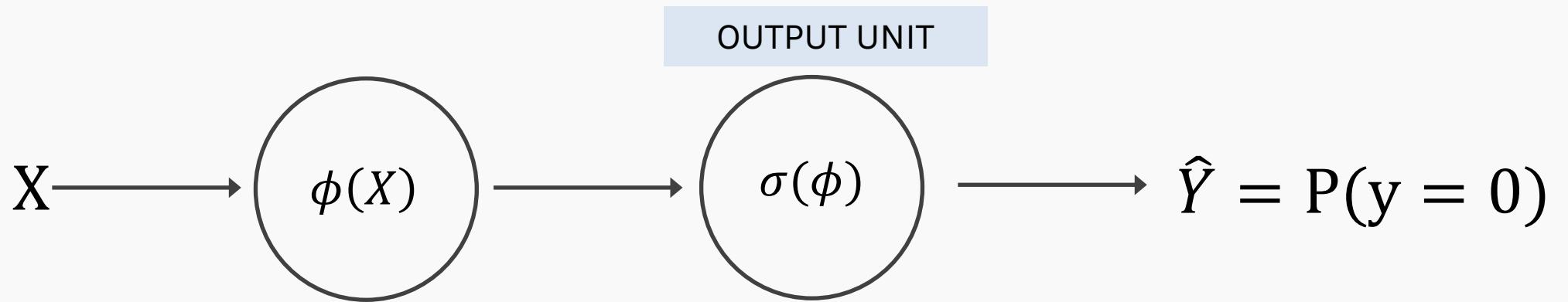
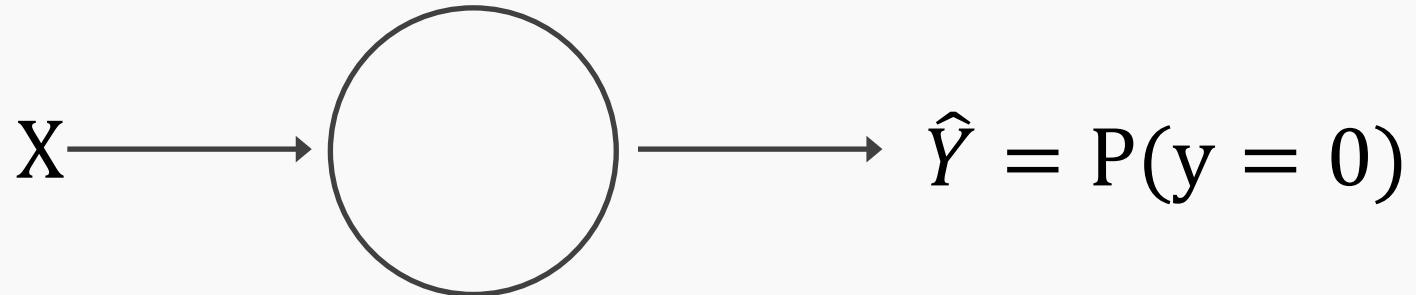
# Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary			



# Link function

$$X \Rightarrow \phi(X) = W^T X \Rightarrow P(y = 0) = \frac{1}{1 + e^{\phi(X)}}$$



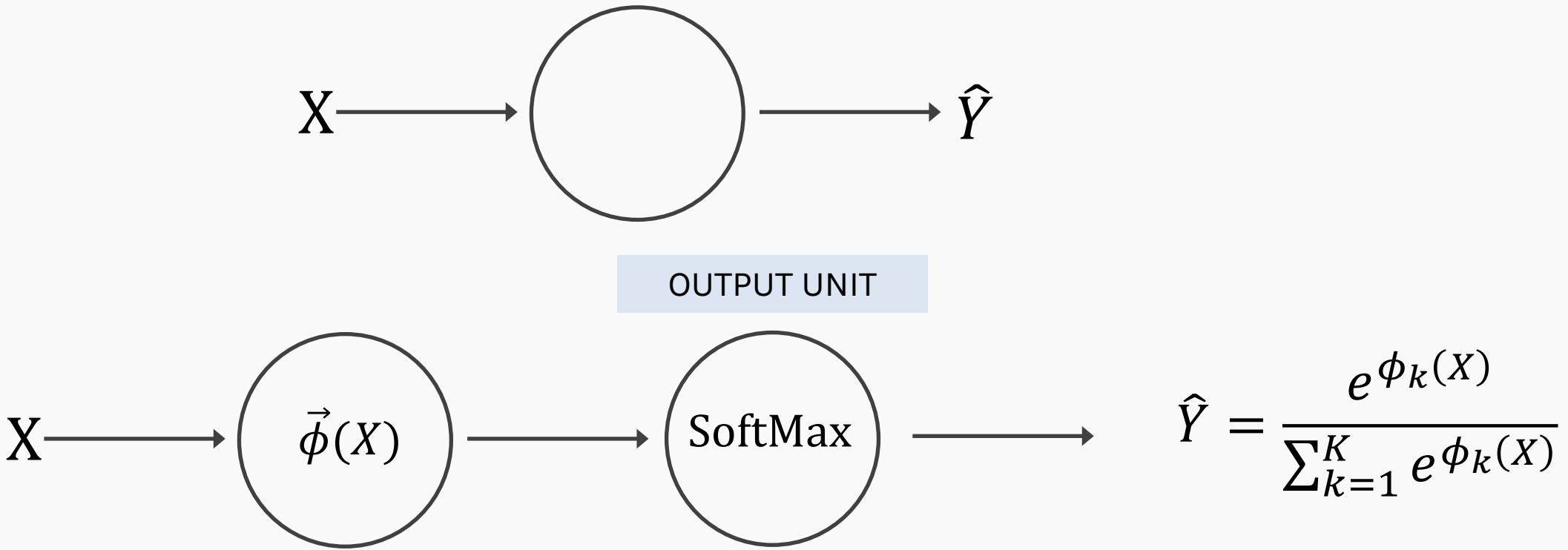
# Output Units

<b>Output Type</b>	<b>Output Distribution</b>	<b>Output layer</b>	<b>Cost Function</b>
Binary	Bernoulli	Sigmoid	Binary Cross Entropy

# Output Units

<b>Output Type</b>	<b>Output Distribution</b>	<b>Output layer</b>	<b>Cost Function</b>
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete			

# Link function multi-class problem



# Output Units

<b>Output Type</b>	<b>Output Distribution</b>	<b>Output layer</b>	<b>Cost Function</b>
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy



# Output Units

<b>Output Type</b>	<b>Output Distribution</b>	<b>Output layer</b>	<b>Cost Function</b>
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE



# Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE
Continuous	Arbitrary	-	GANS

# Design Choices

---

Activation function

Loss function

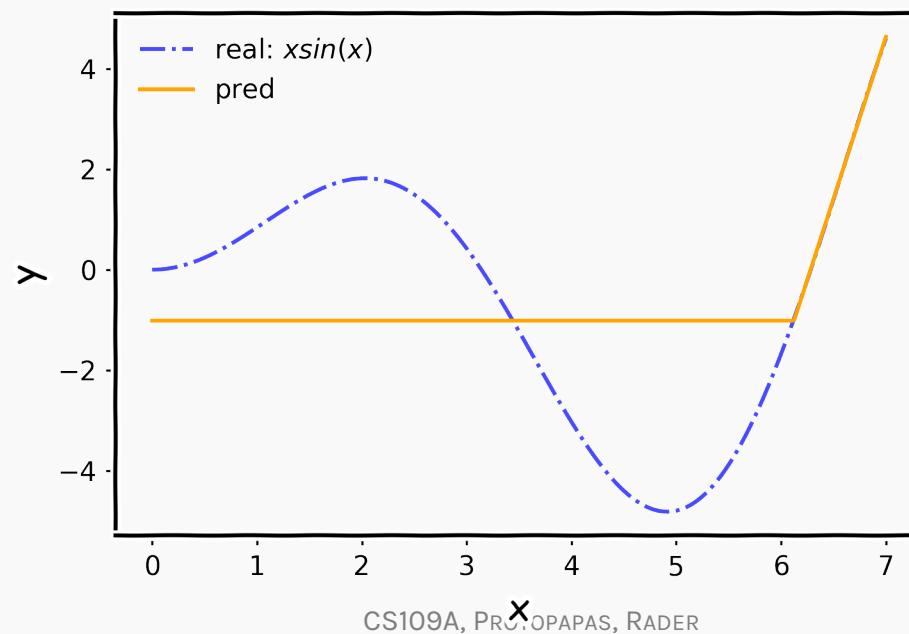
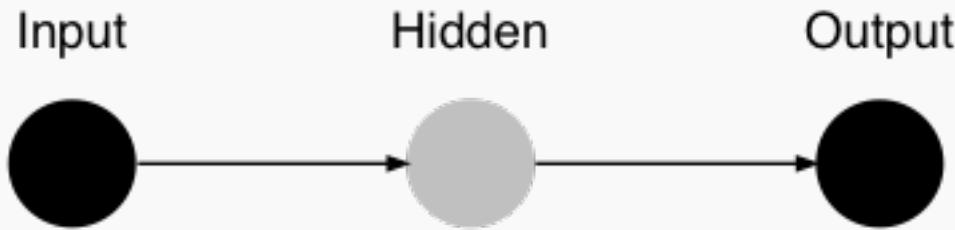
Output units

Architecture

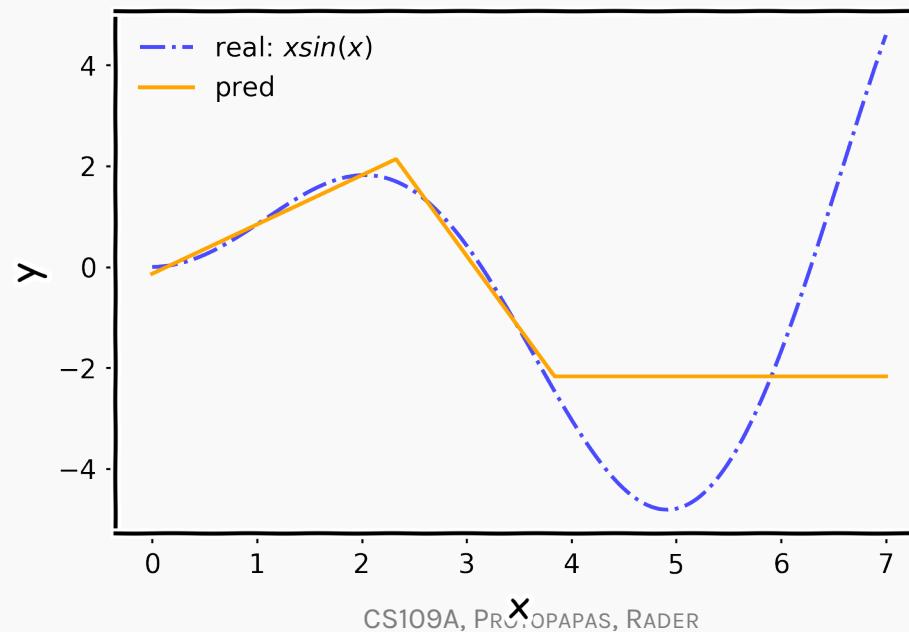
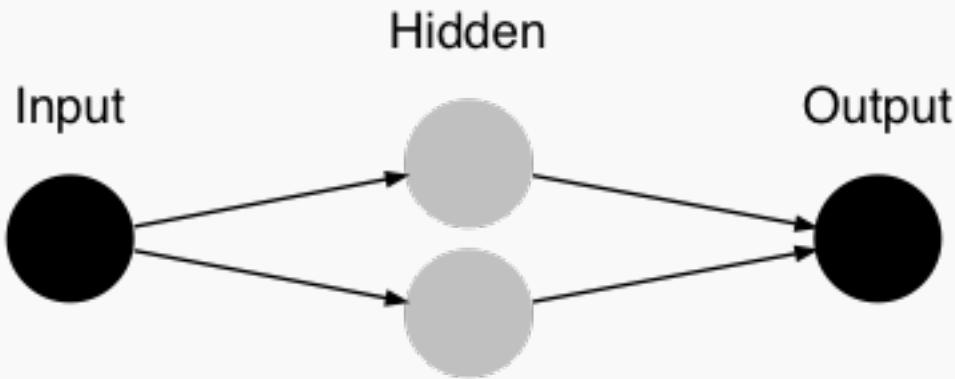
Optimizer



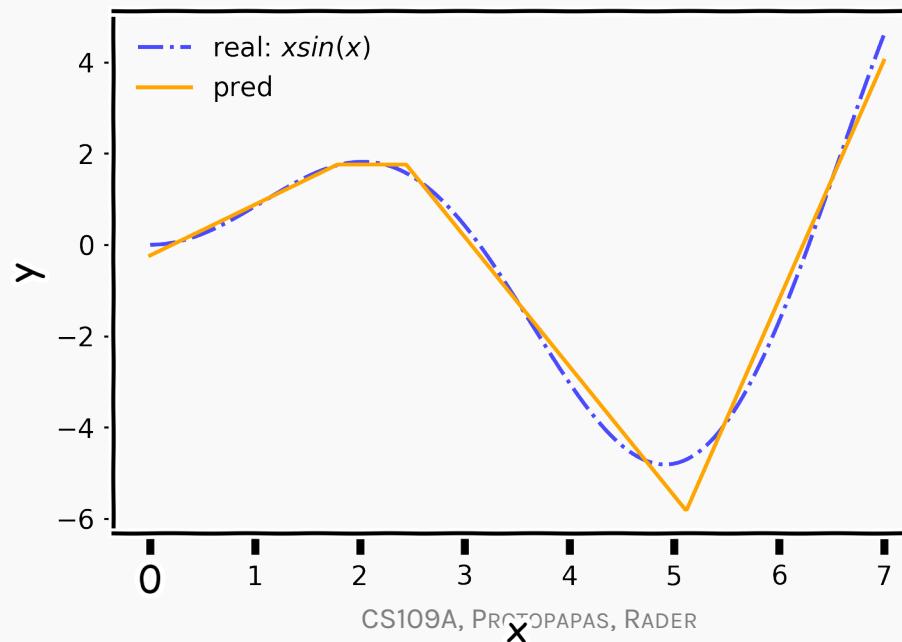
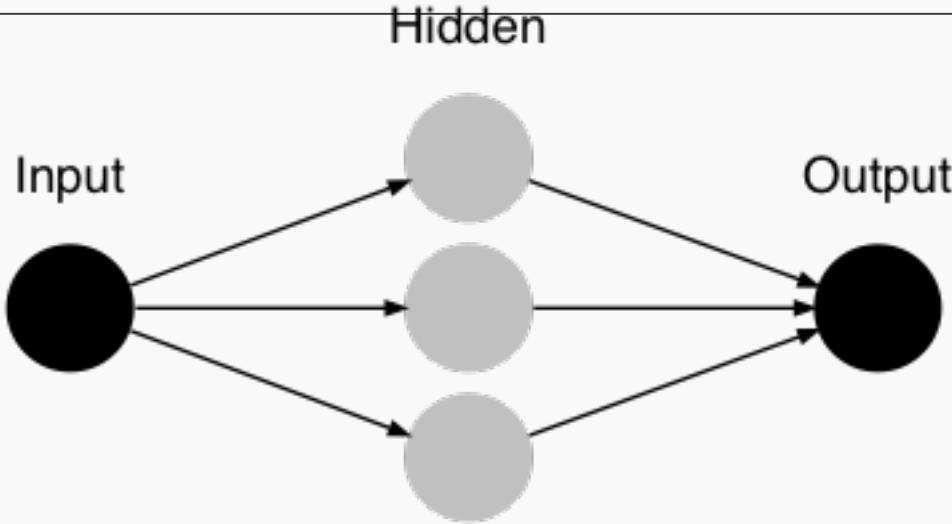
# NN in action



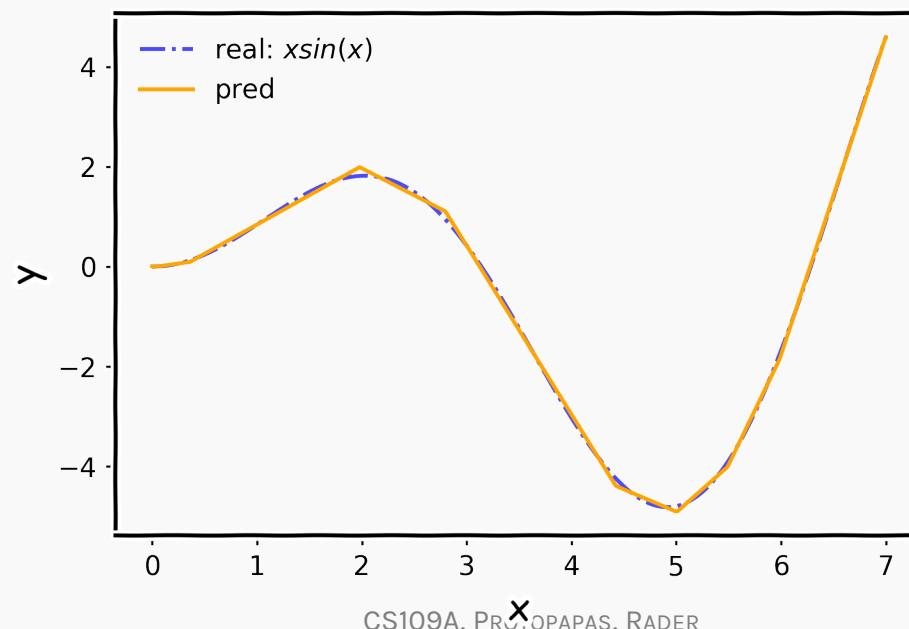
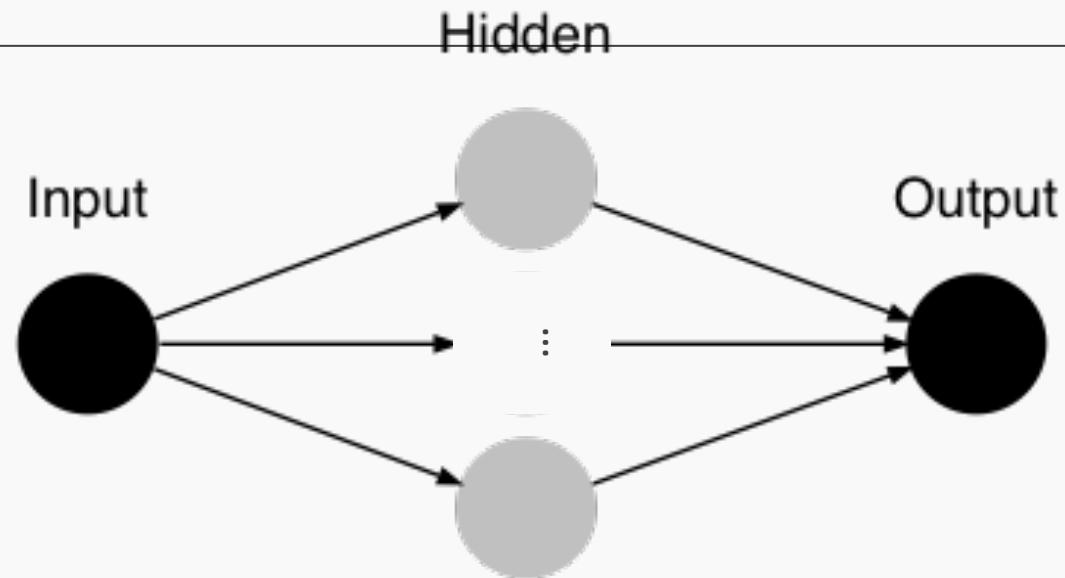
# NN in action



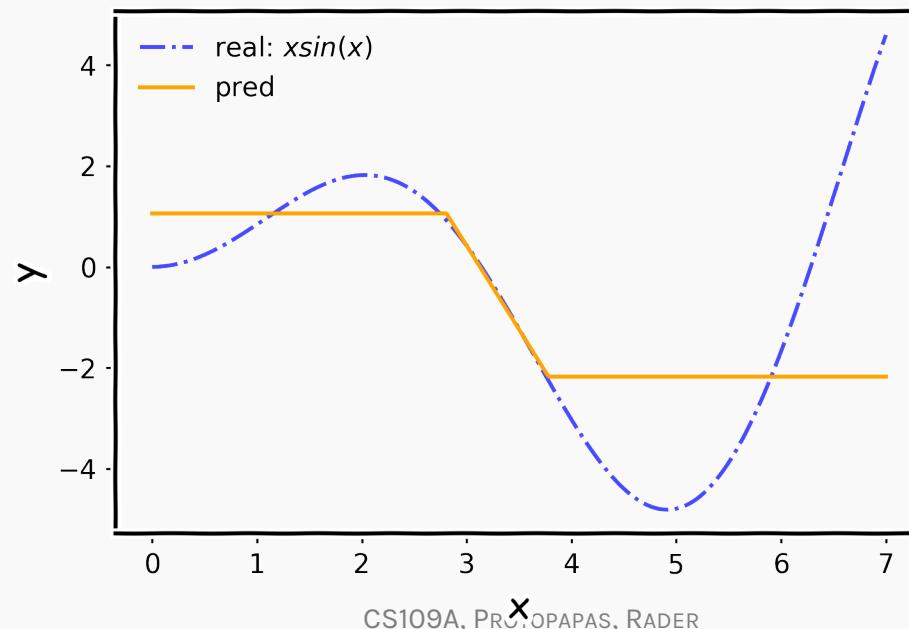
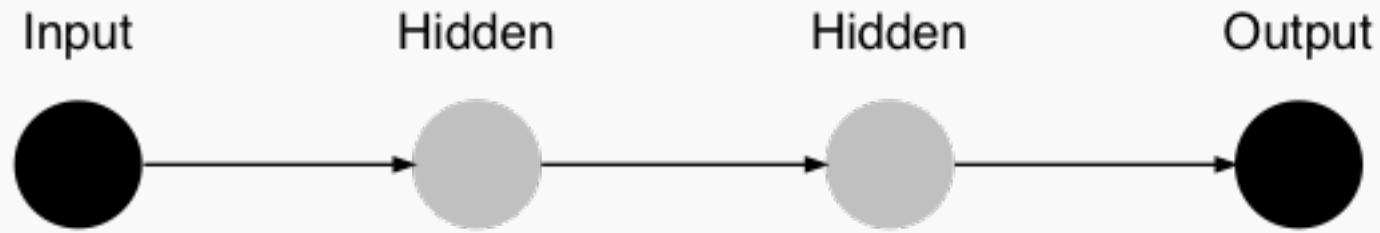
# NN in action



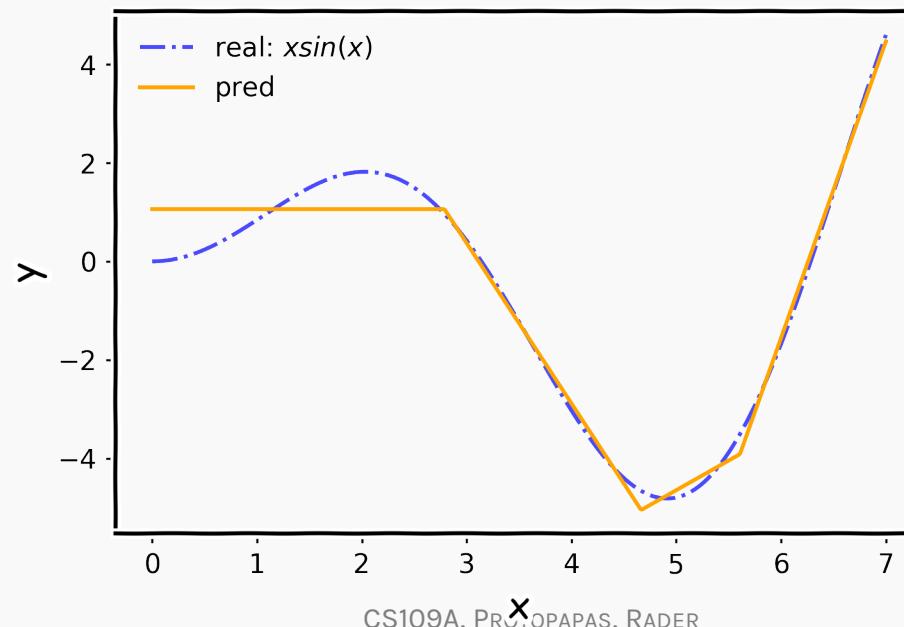
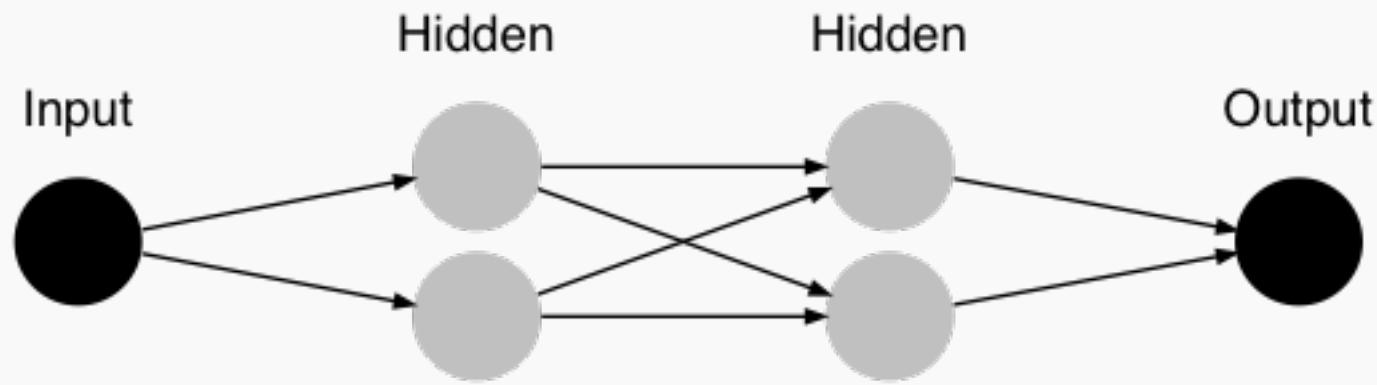
# NN in action



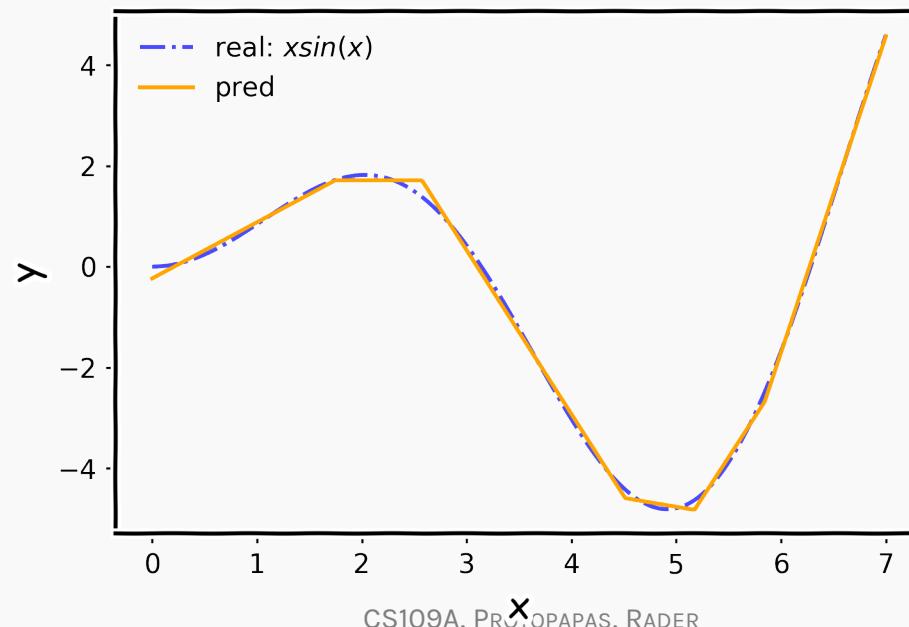
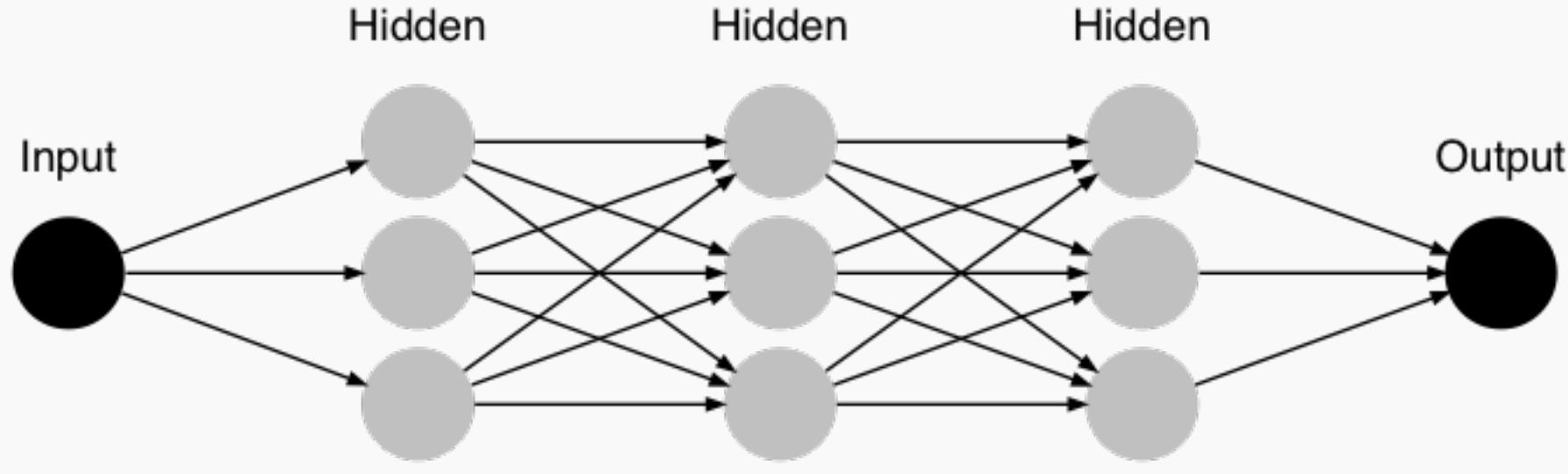
# NN in action



# NN in action



# NN in action



# Universal Approximation Theorem

Think of a Neural Network as function approximation.

$$Y = f(x) + \epsilon$$

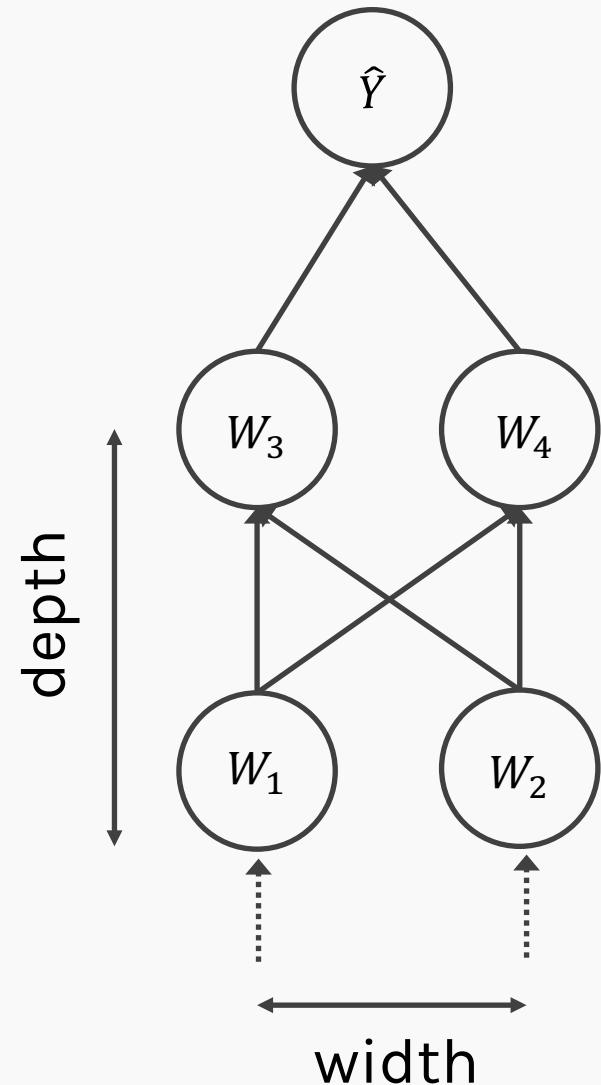
$$Y = \hat{f}(x) + \epsilon$$

$$\text{NN: } \Rightarrow \hat{f}(x)$$

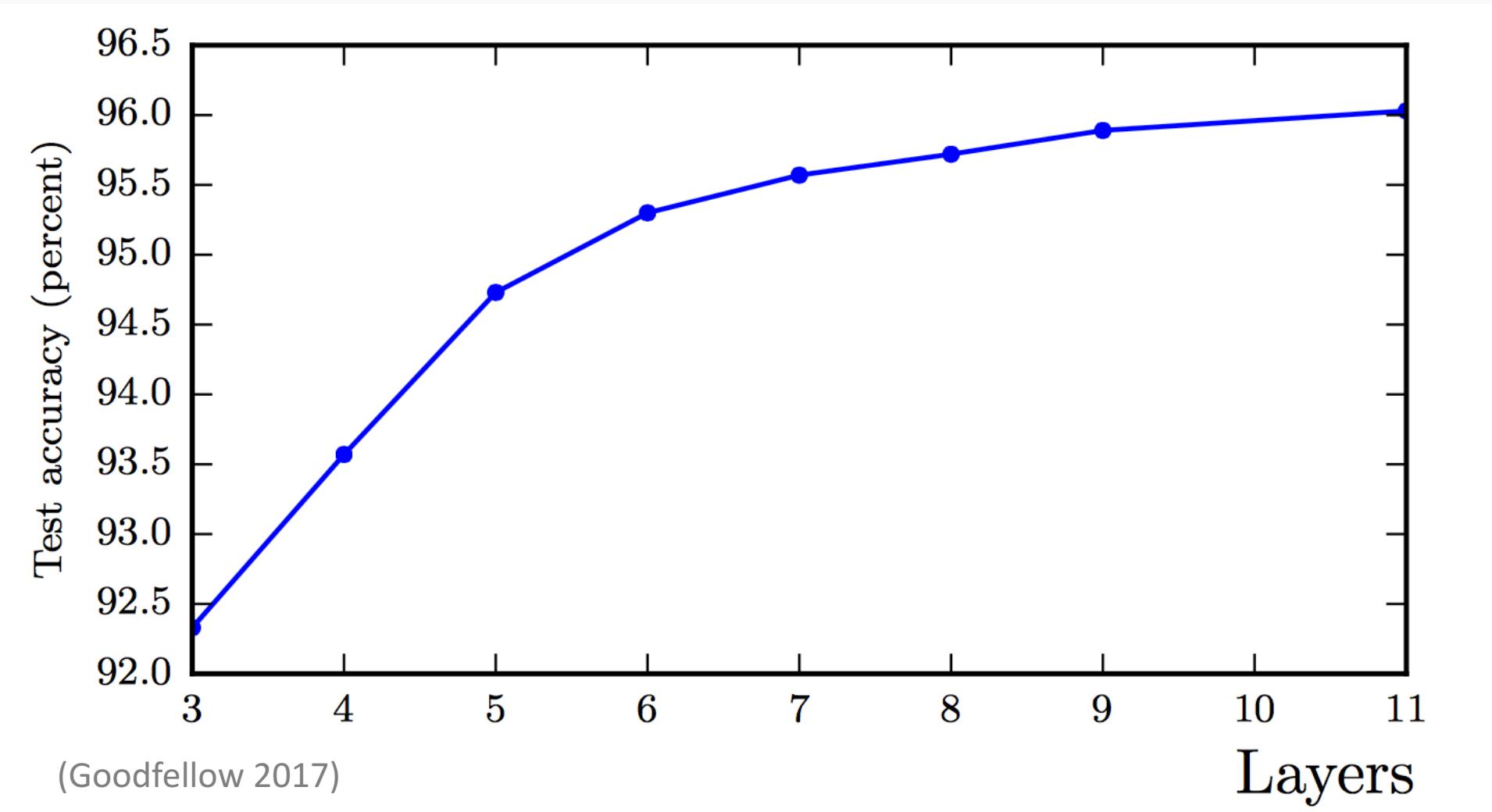
One hidden layer is enough to represent an approximation of any function to an arbitrary degree of accuracy

So why deeper?

- Shallow net may need (exponentially) more width
- Shallow net may overfit more



# Better Generalization with Depth



# Large, Shallow Nets Overfit More

