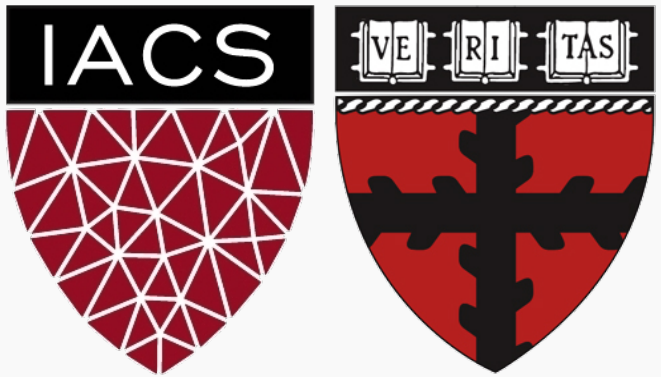
Lecture 3: Anatomy of NN

Pavlos Protopapas
Institute for Applied Computational Science
Harvard

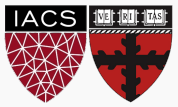


Outline

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture

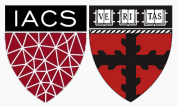


Outline

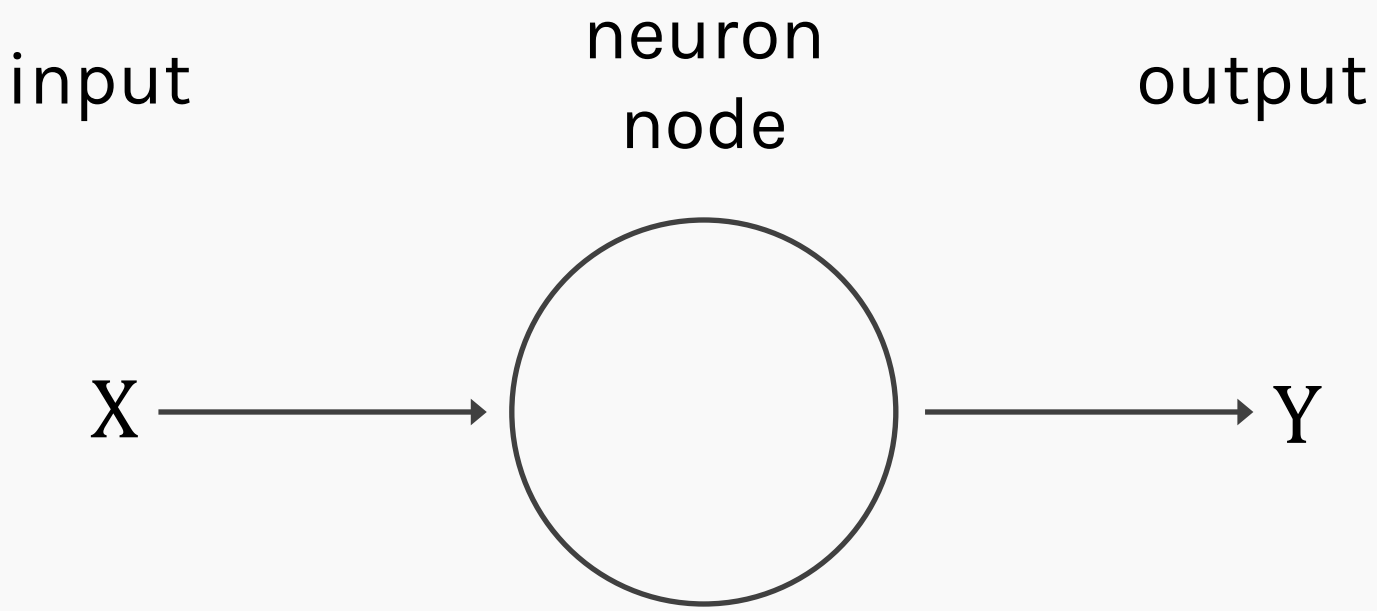
Anatomy of a NN

Design choices

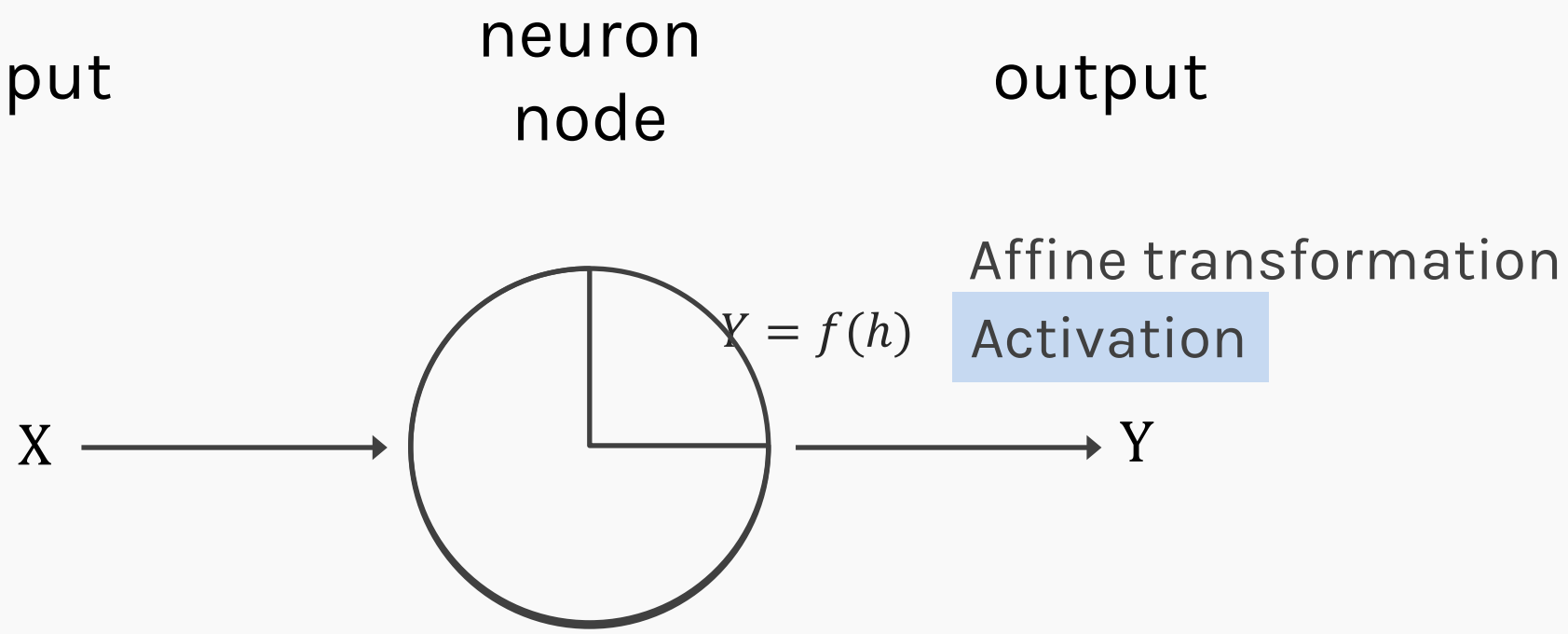
- Activation function
- Loss function
- Output units
- Architecture



Anatomy of artificial neural network (ANN)

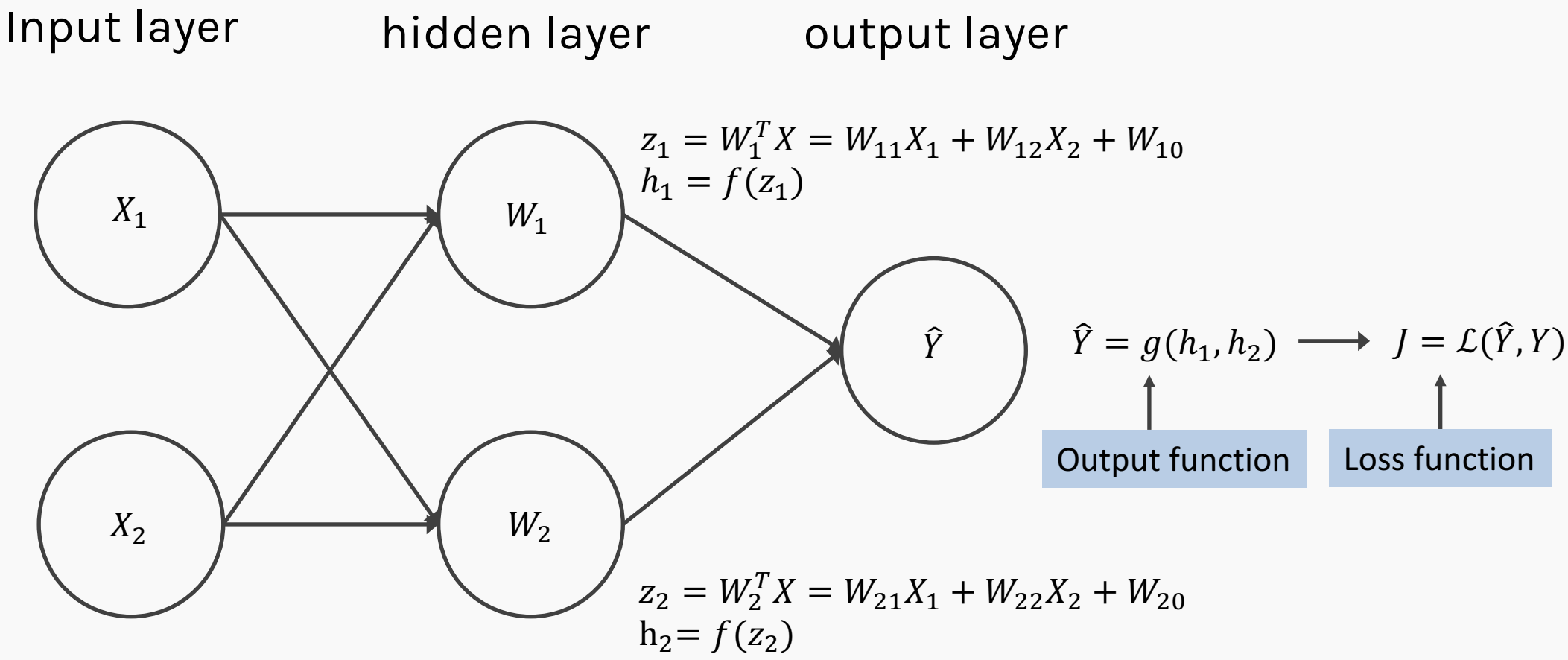


Anatomy of artificial neural network (ANN)



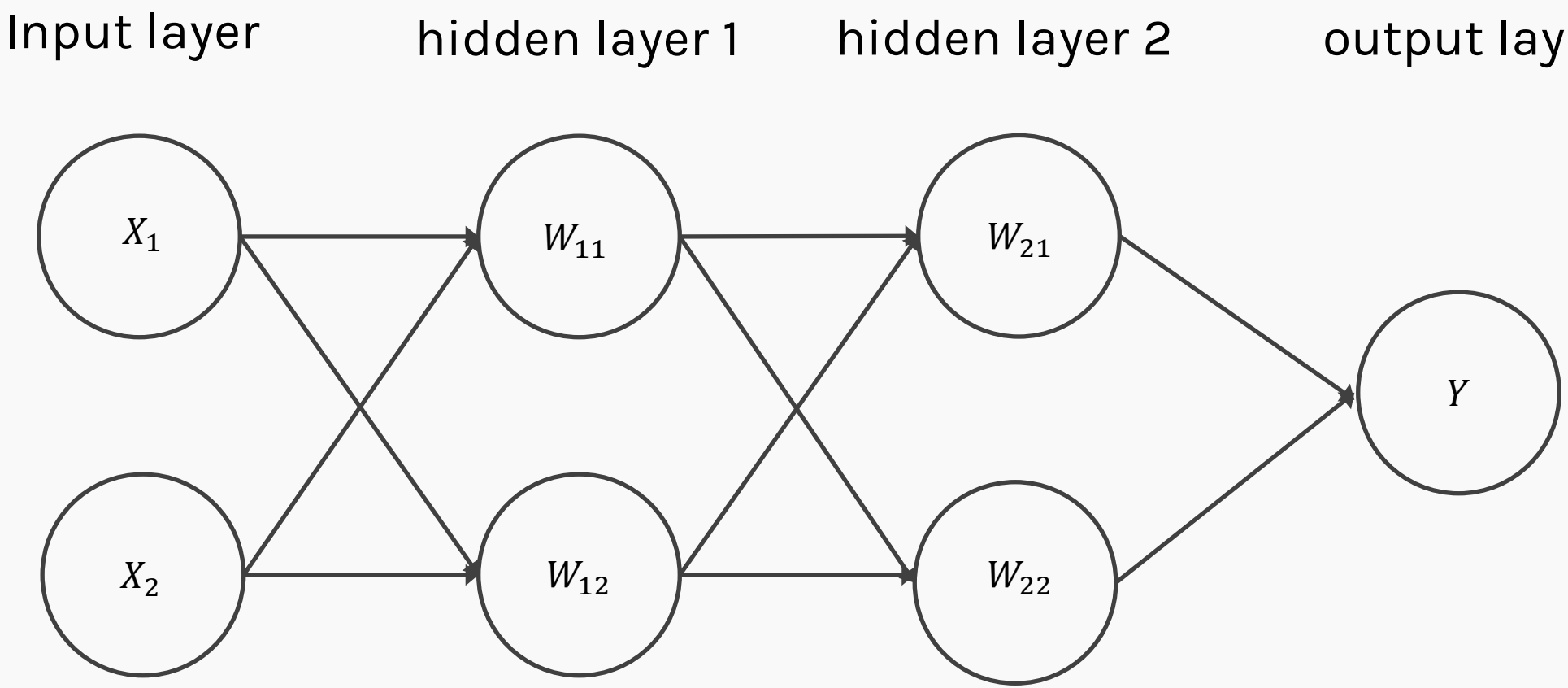
We will talk later about the choice of activation function. So far we have only talked about sigmoid as an activation function but there are other choices.

Anatomy of artificial neural network (ANN)

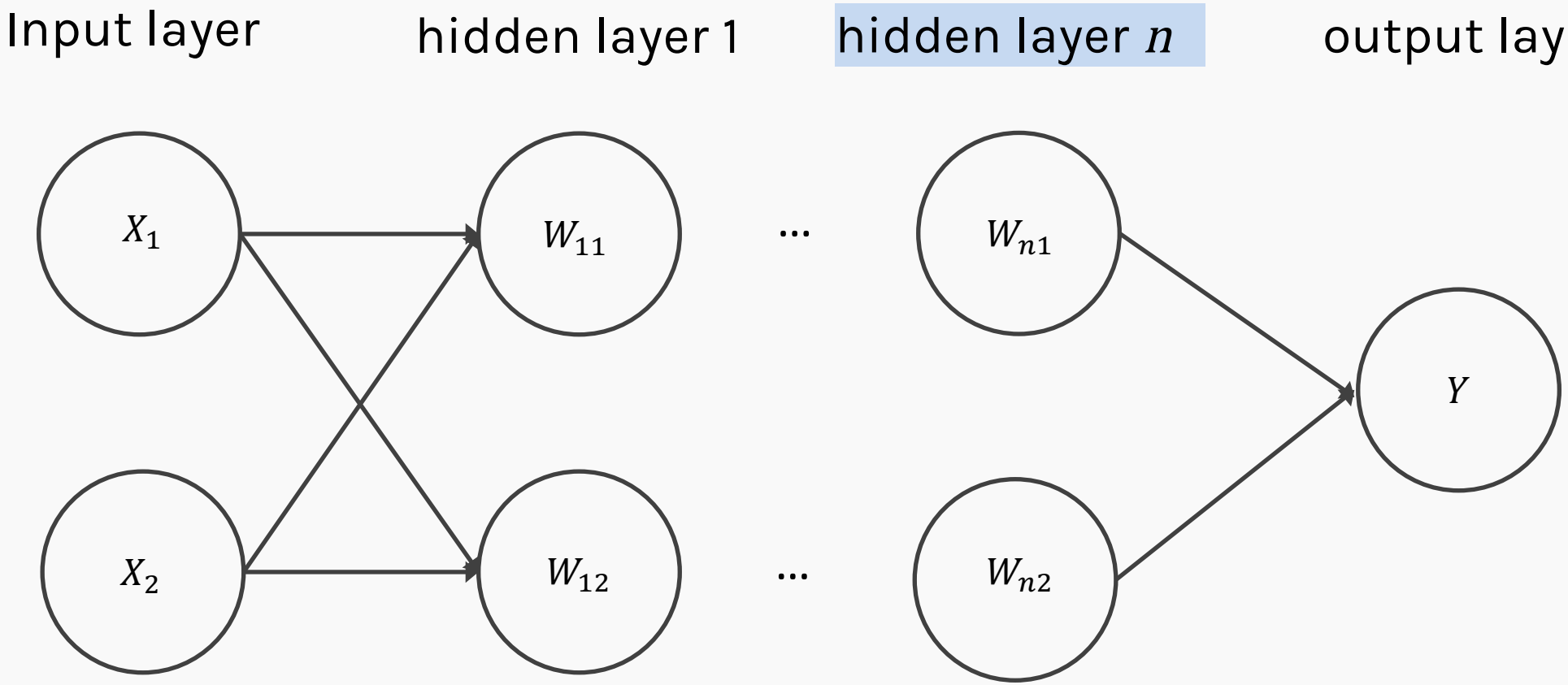


We will talk later about the choice of the output layer and the loss function. So far we consider sigmoid as the output and log-bernouli.

Anatomy of artificial neural network (ANN)



Anatomy of artificial neural network (ANN)



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

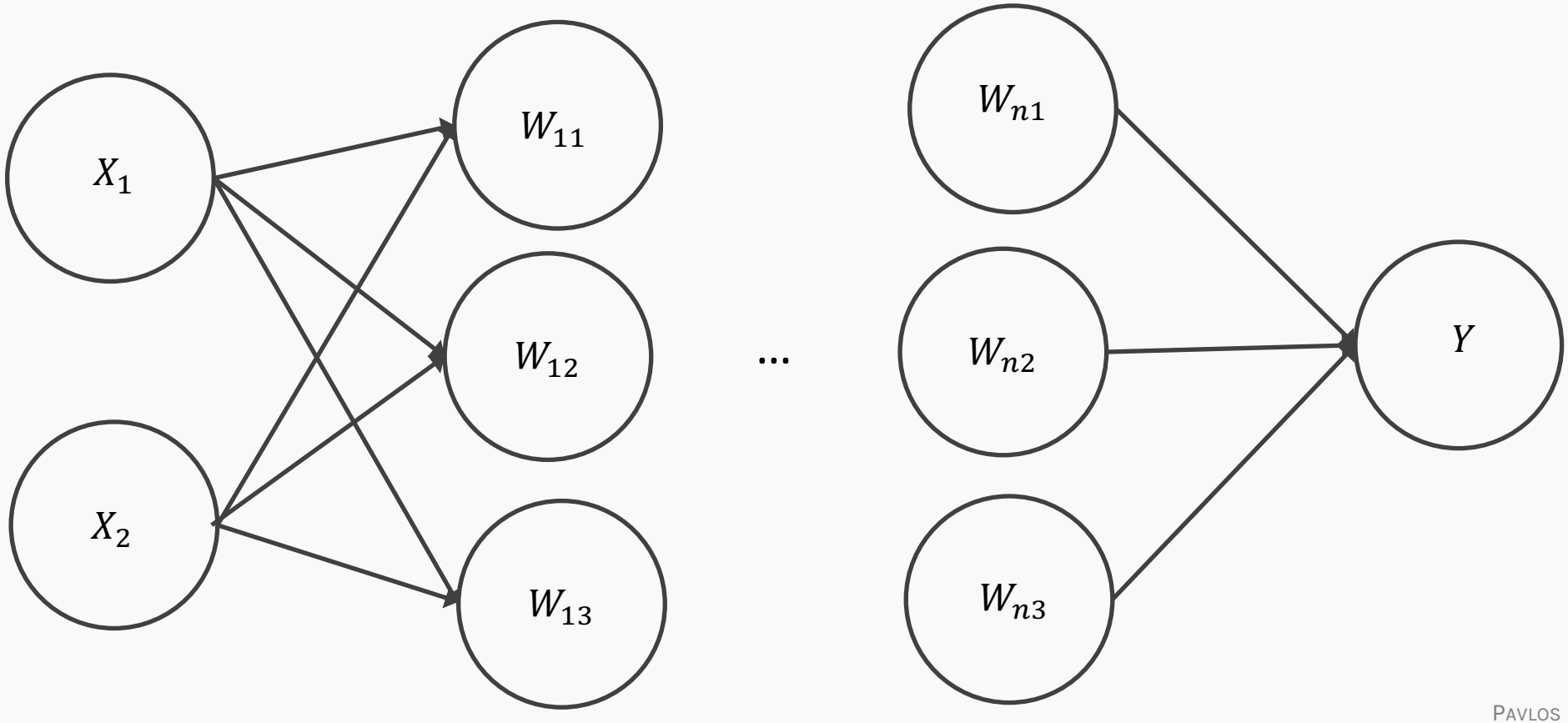
Anatomy of artificial neural network (ANN)

Input layer

hidden layer 1,
3 nodes

hidden layer n
3 nodes

output layer



Anatomy of artificial neural network (ANN)

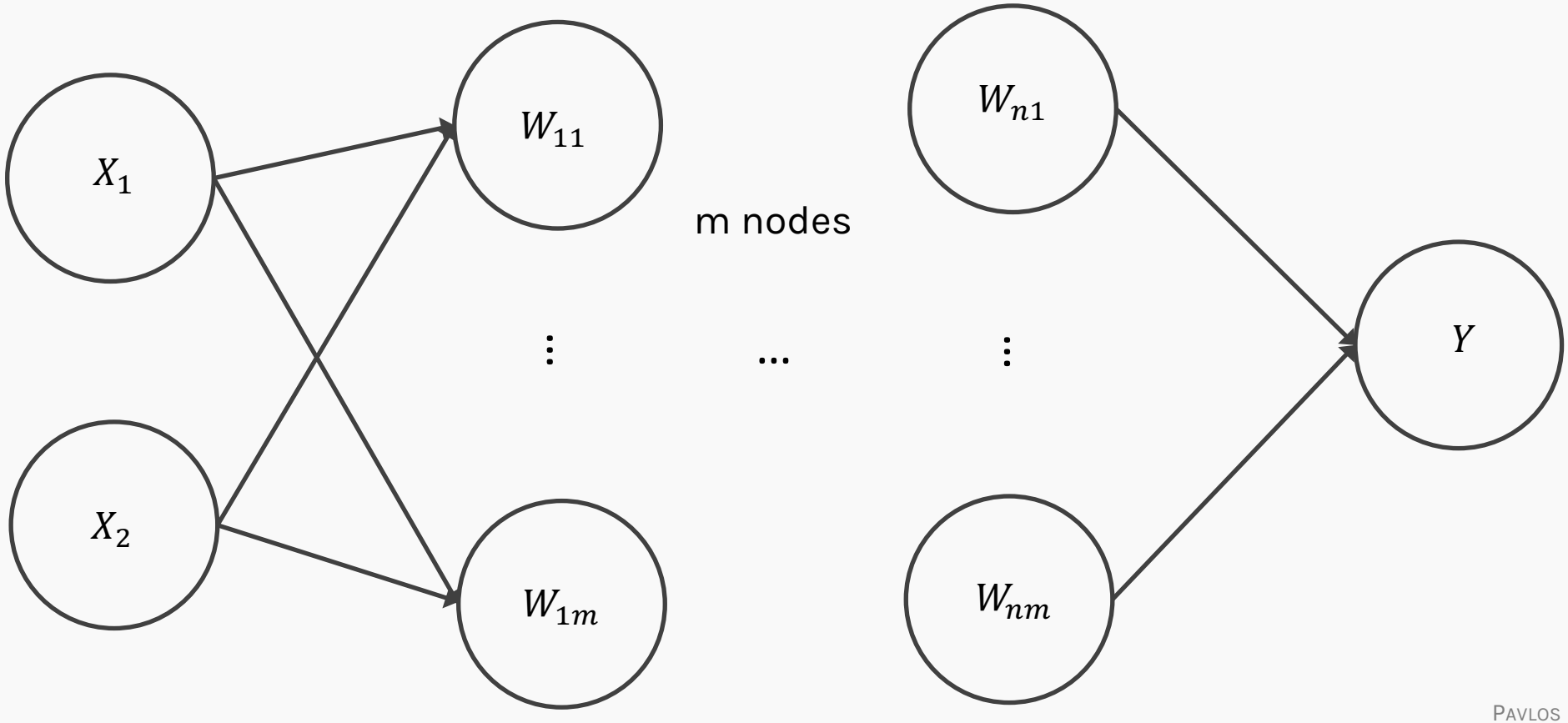
Input layer

hidden layer 1,

m nodes

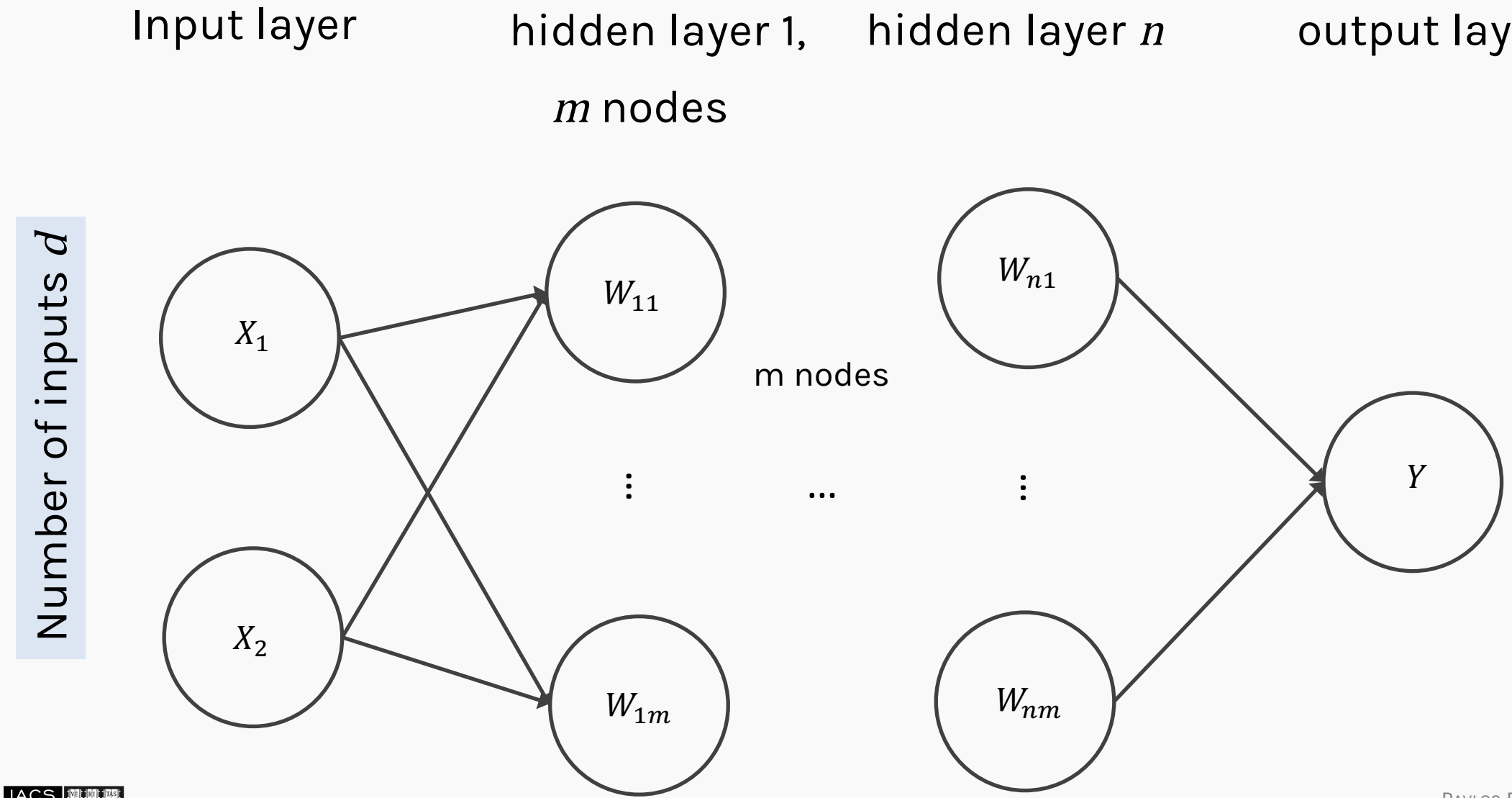
hidden layer n

output layer



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

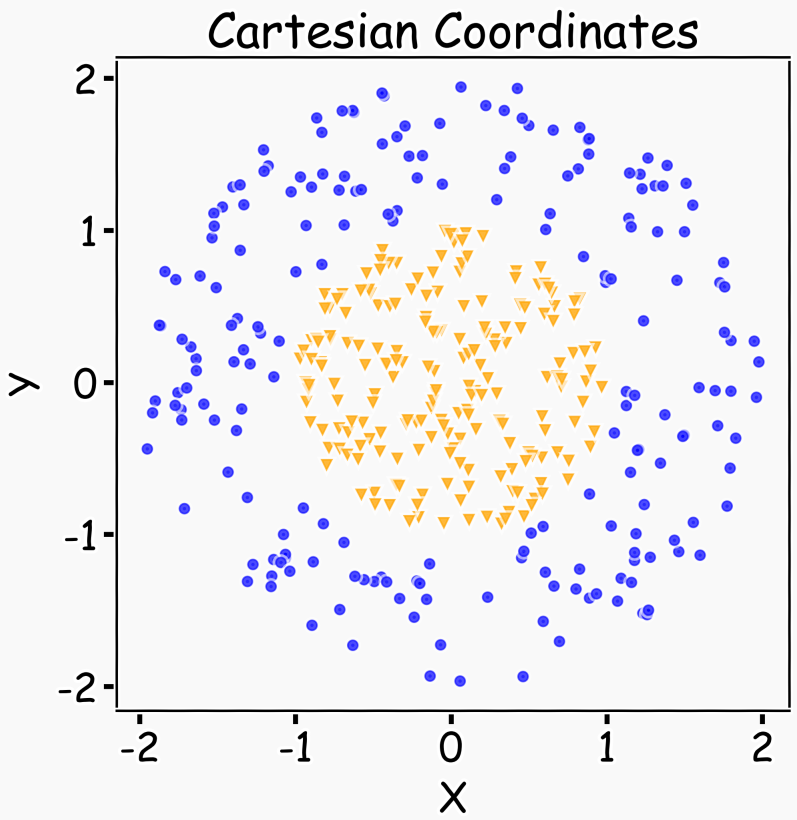
Anatomy of artificial neural network (ANN)

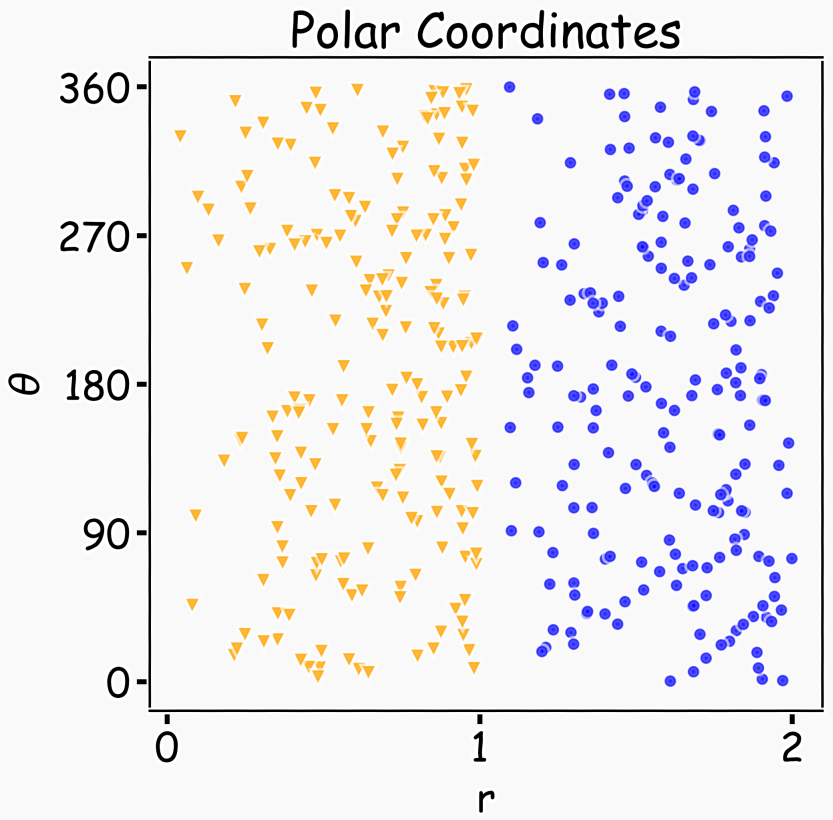


Number of inputs is specified by the data

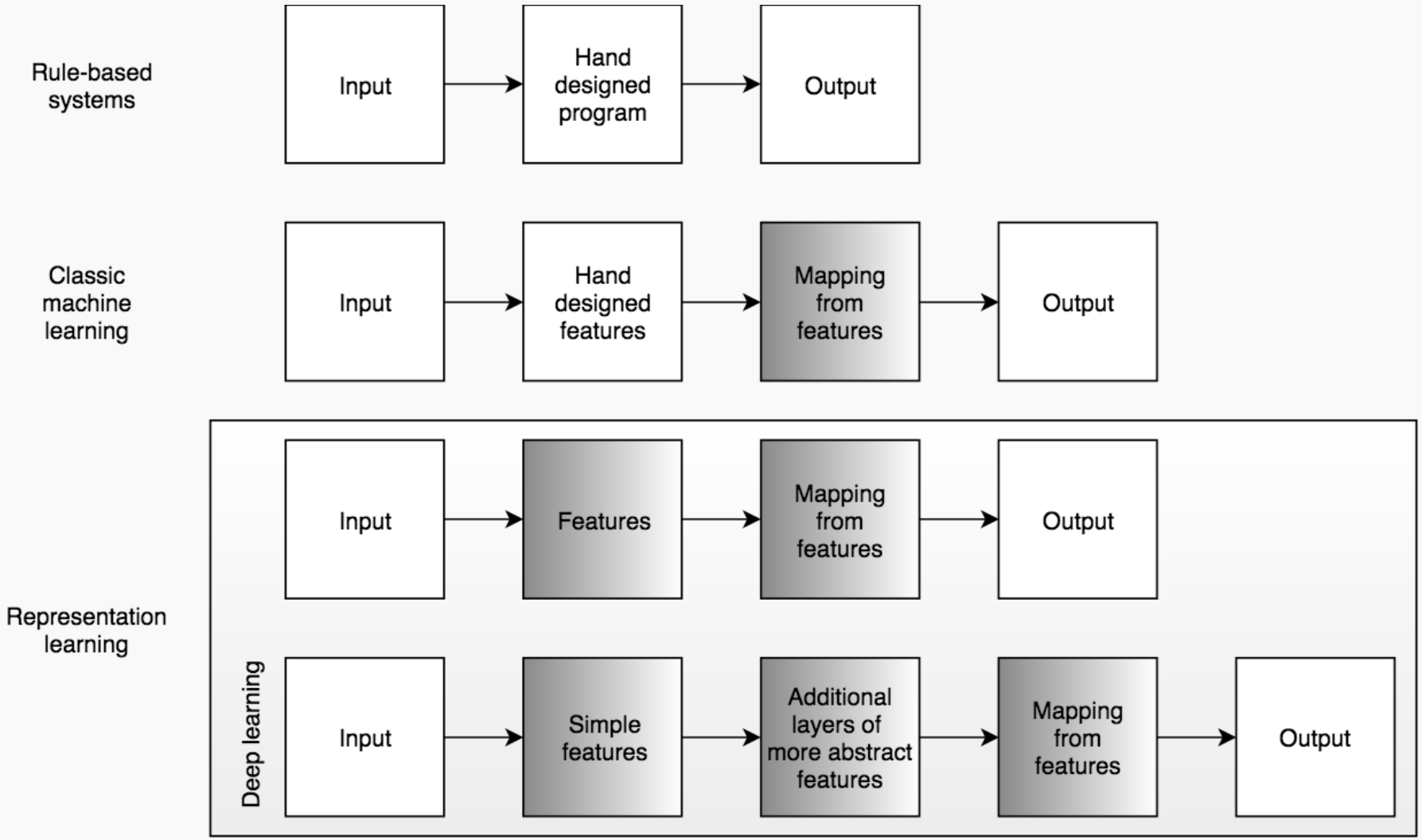
Why layers? Representation

Representation matters!

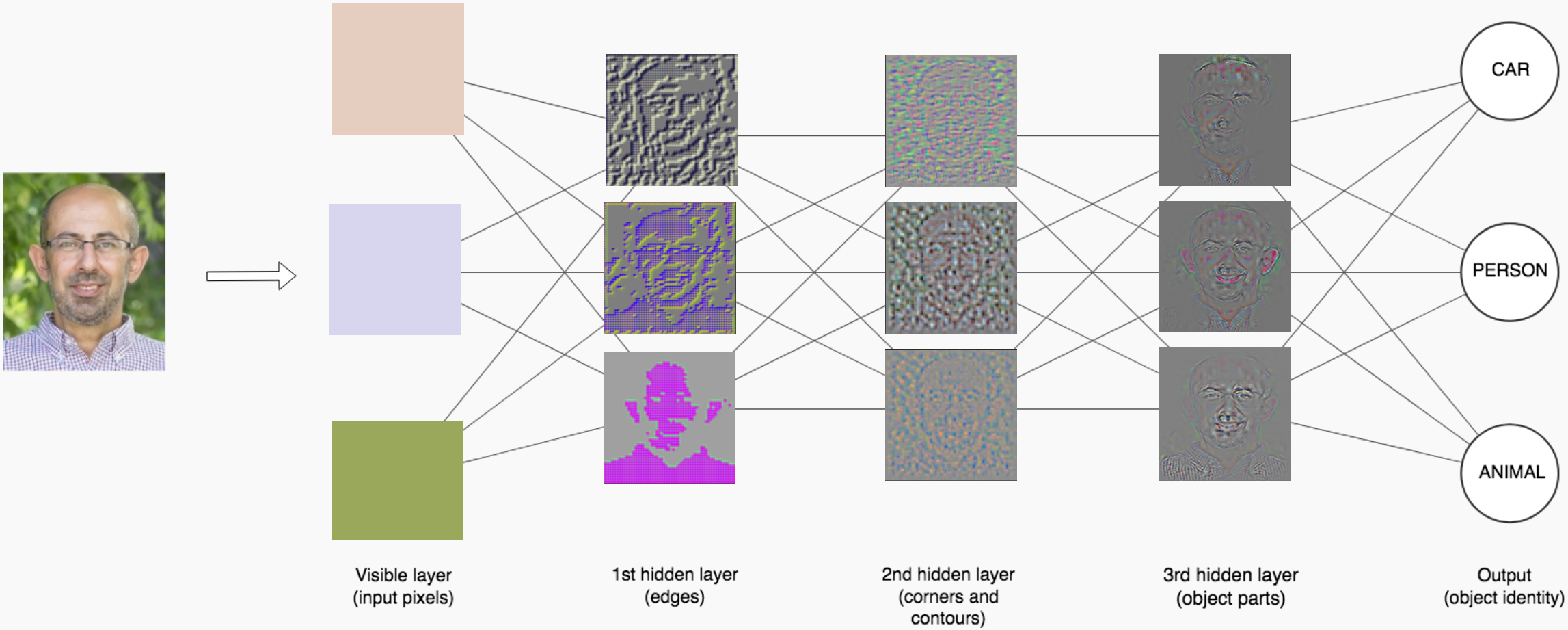




Learning Multiple Components



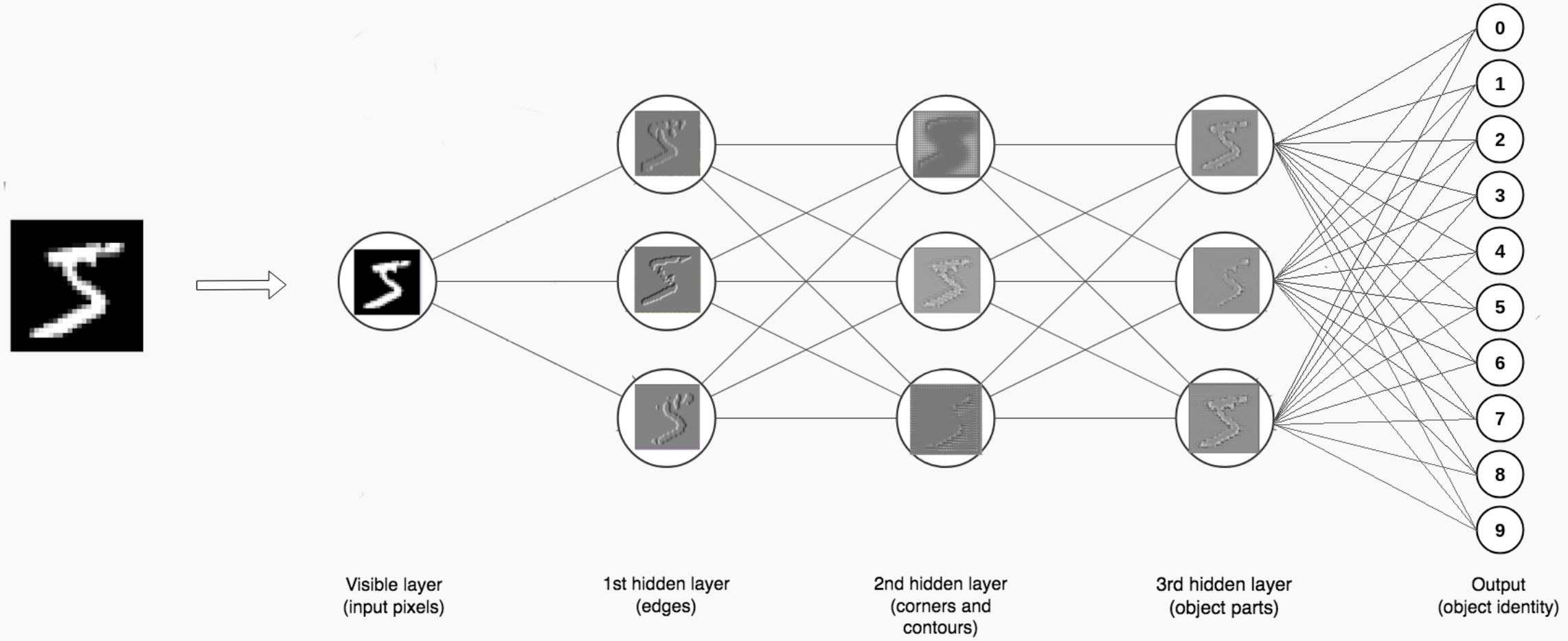
Depth = Repeated Compositions



Neural Networks

Hand-written digit recognition: MNIST data

Depth = Repeated Compositions



Beyond Linear Models

Linear models:

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

Alternative:

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

Where ϕ is a *non-linear transform*

Traditional ML

Manually engineer ϕ

- Domain specific, enormous human effort

Generic transform

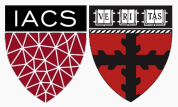
- Maps to a higher-dimensional space
- Kernel methods: e.g. RBF kernels
- Over fitting: does not generalize well to test set
- Cannot encode enough prior information

Deep Learning

- Directly learn ϕ

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

- $\phi(x;\theta)$ is an automatically-learned **representation** of x
- For **deep networks**, ϕ is the function learned by the **hidden layers** of the network
- θ are the learned weights
- Non-convex optimization
- Can encode prior beliefs, generalizes well

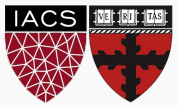


Outline

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture

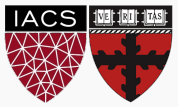


Outline

Anatomy of a NN

Design choices

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



Activation function

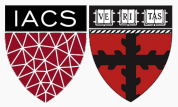
$$h = f(W^T X + b)$$

The activation function should:

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

Common choices are

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



Activation function

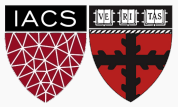
$$h = f(W^T X + b)$$

The activation function should:

- Provide **non-linearity**
- Ensure gradients remain large through hidden unit

Common choices are

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



Activation function

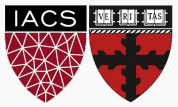
$$h = f(W^T X + b)$$

The activation function should:

- Provide **non-linearity**
- Ensure gradients remain large through hidden unit

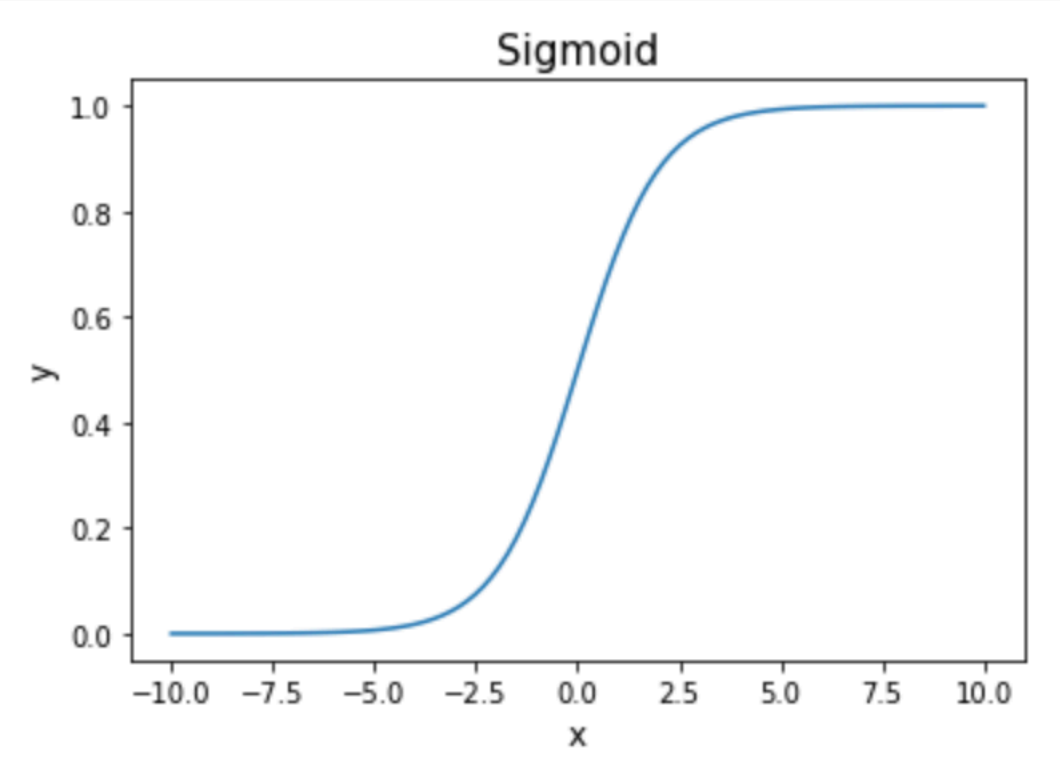
Common choices are

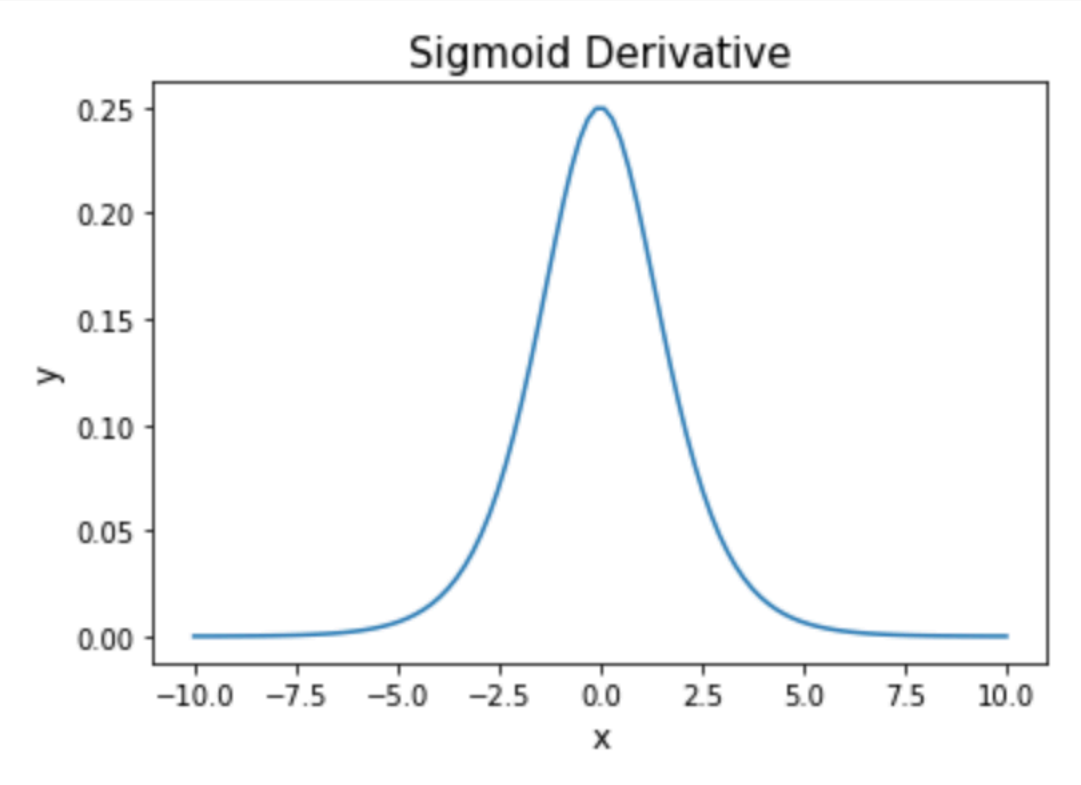
- Sigmoid
- Relu, leaky ReLU, Generalized ReLU, MaxOut
- softplus
- tanh
- swish



Sigmoid (aka Logistic)

$$y = \frac{1}{1 + e^{-x}}$$

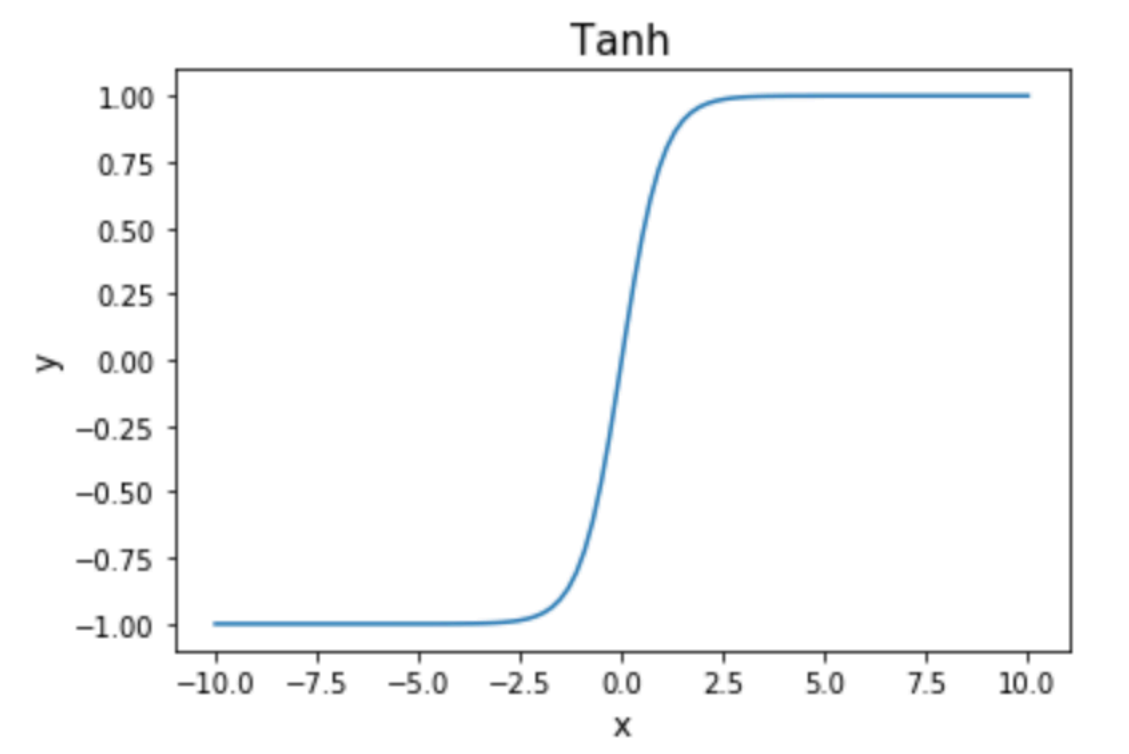


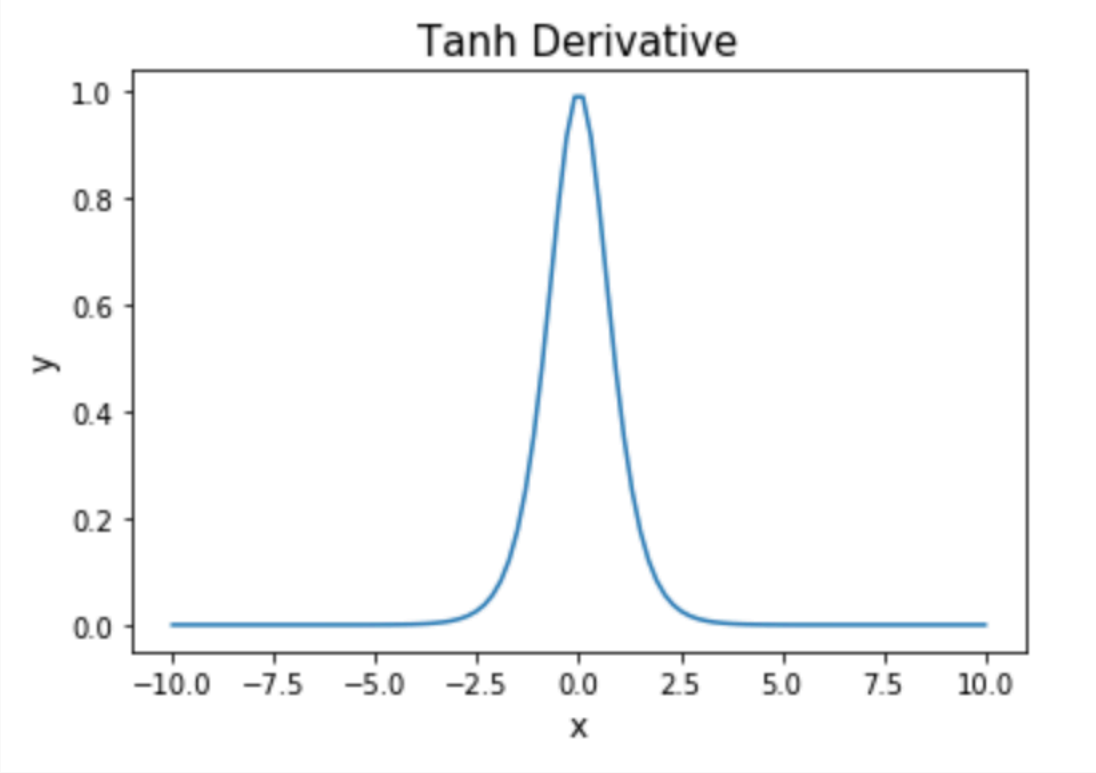


Derivative is **zero** for much of the domain. This leads to "vanishing gradients" in backpropagation.

Hyperbolic Tangent (Tanh)

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

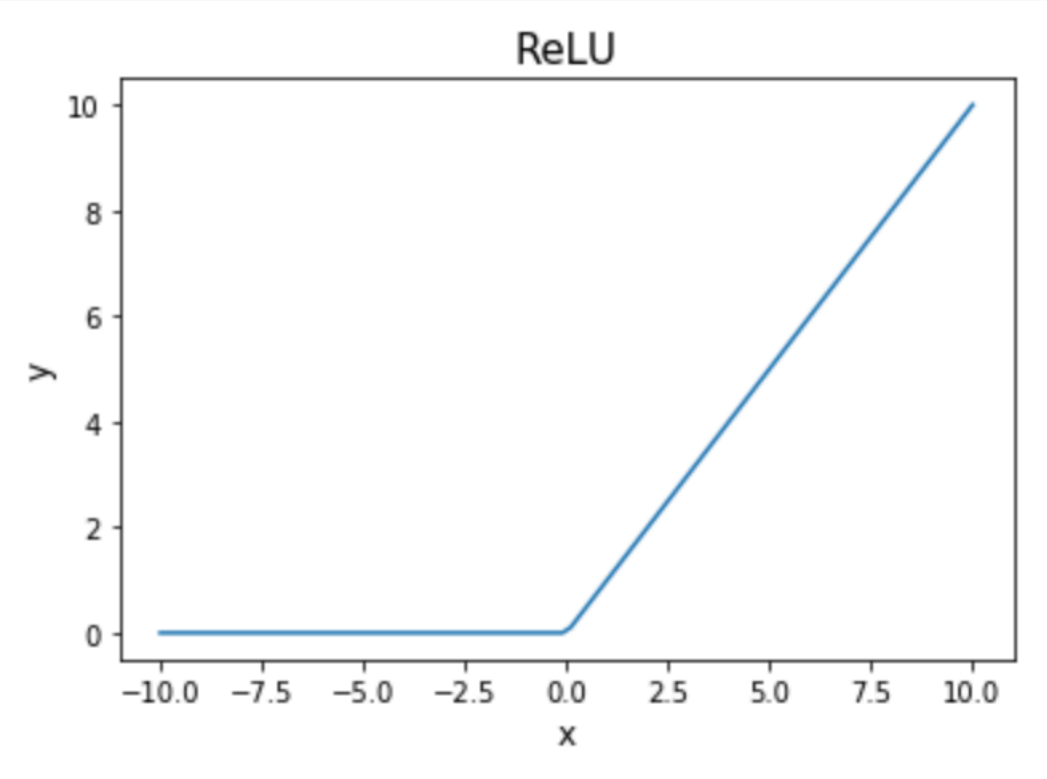


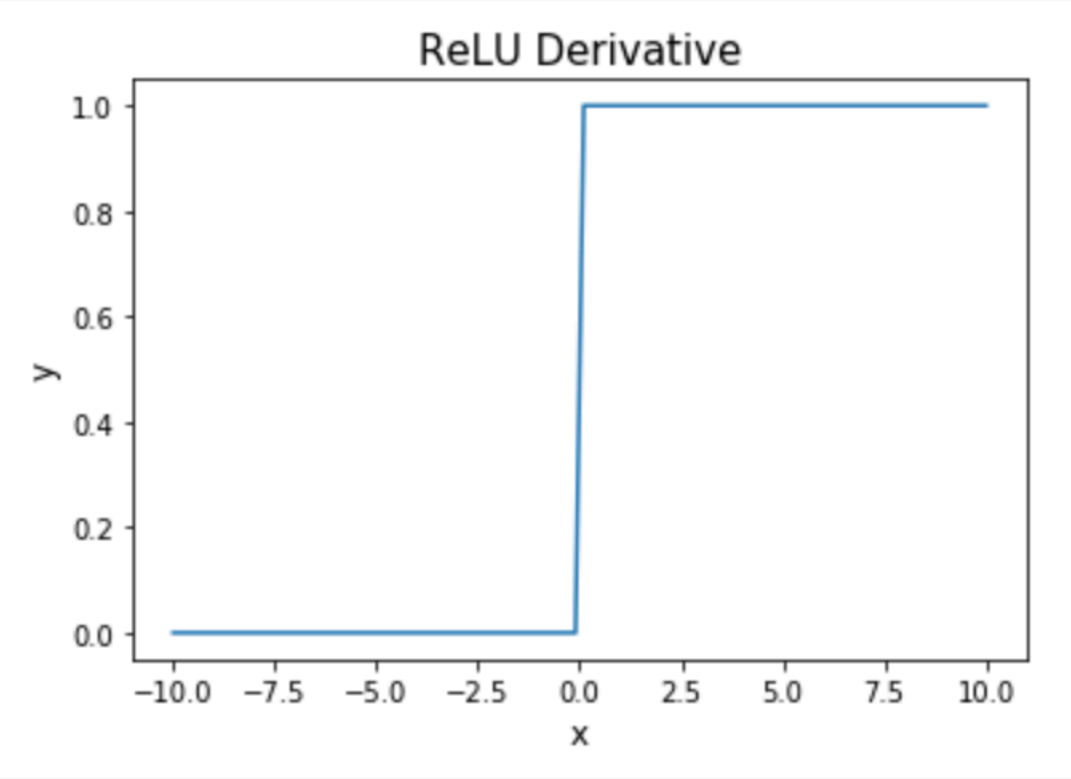


Same problem of “vanishing gradients” as sigmoid.

Rectified Linear Unit (ReLU)

$$y = \max(0, x)$$



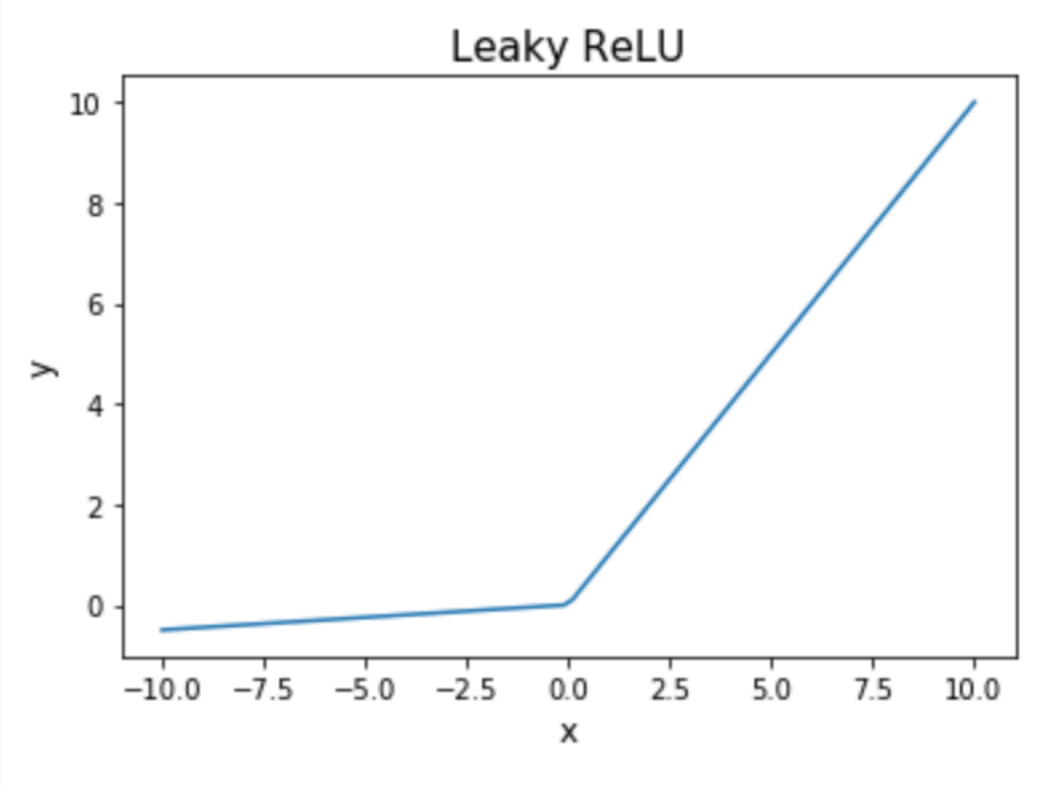


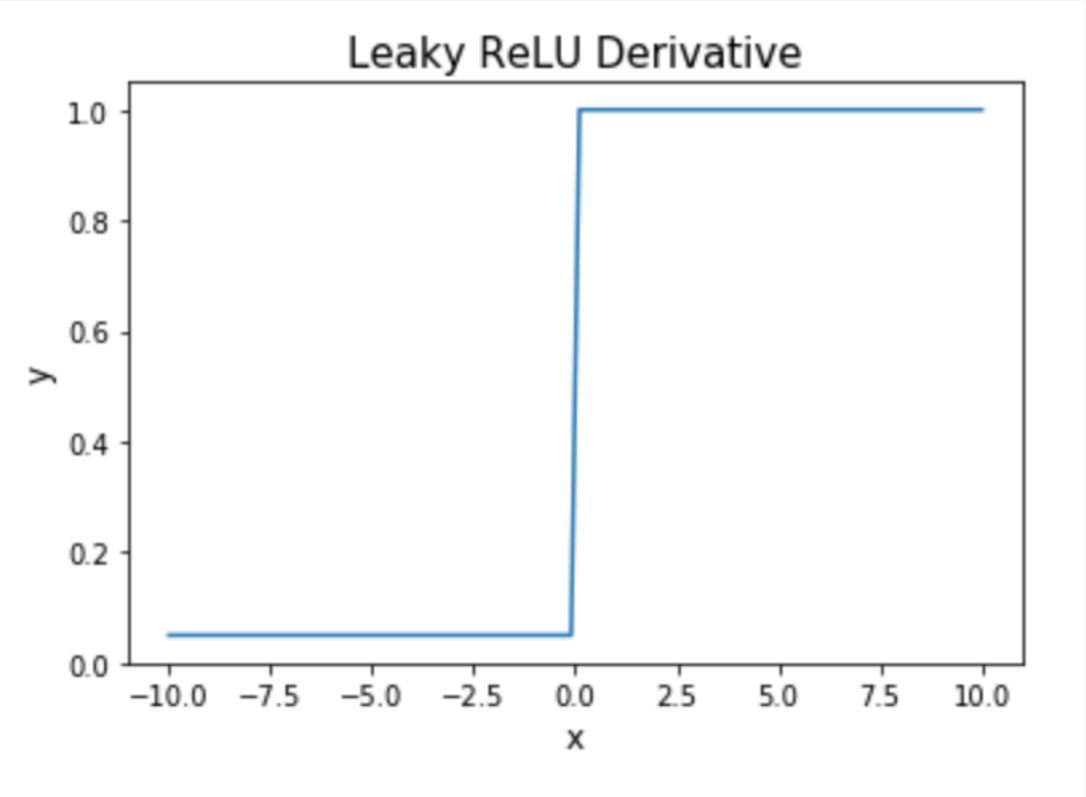
Two major advantages:

1. No vanishing gradient when x > 0
2. Provides sparsity (regularization) since y = 0 when x < 0

Leaky ReLU

$$y = \max(0, x) + \alpha * \min(0, x)$$
Where α is a small constant (negative slope)



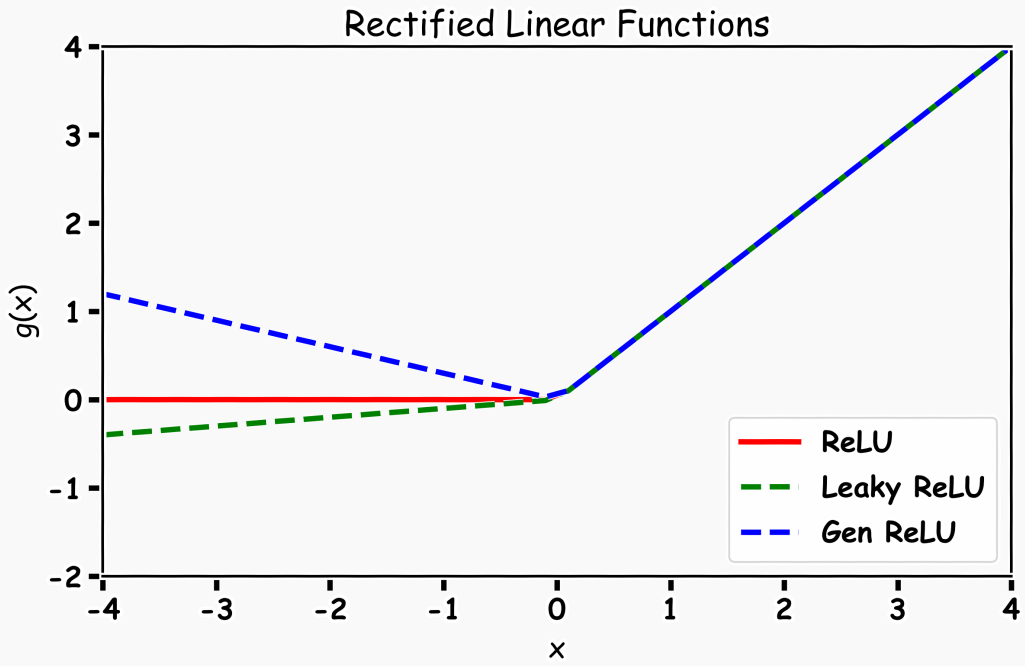


- Tries to fix “dying ReLU” problem: derivative is non-zero everywhere.
- Some people report success with this form of activation function, but the results are not always consistent

Generalized ReLU

Generalization: For $\alpha_i > 0$

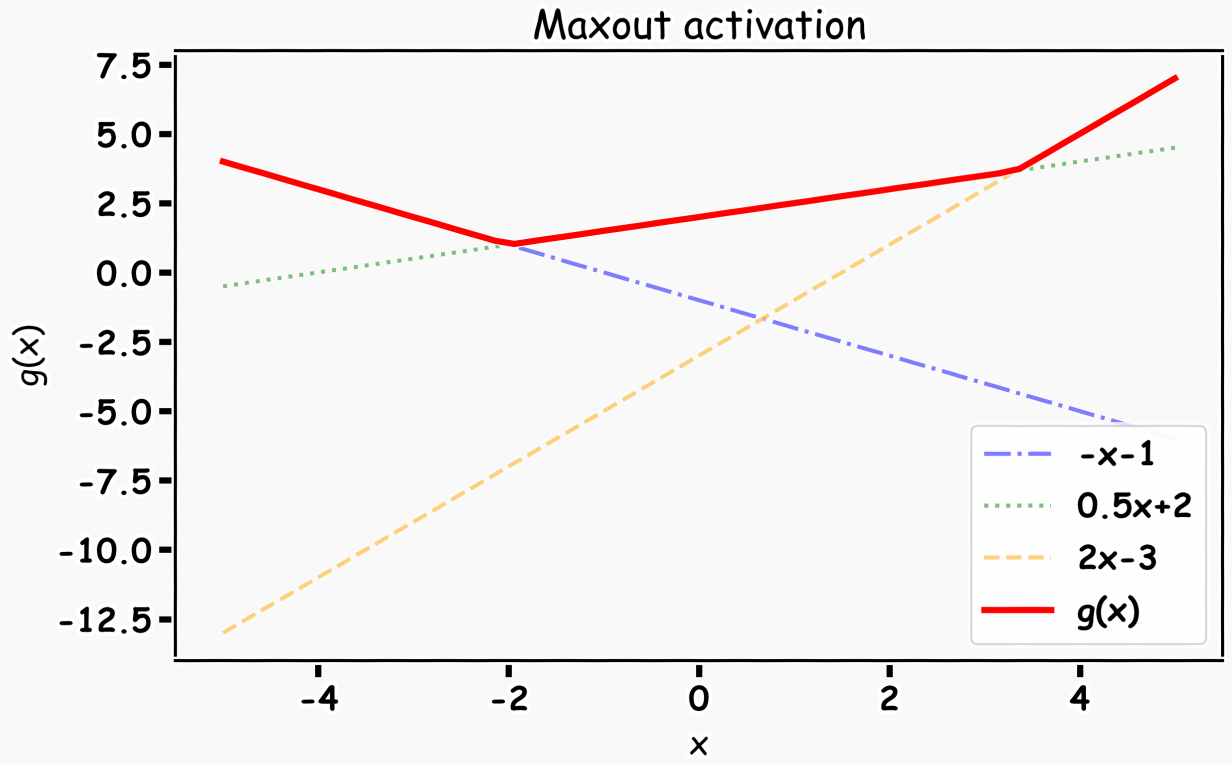
$$g(x_i, \alpha) = \max\{a, x_i\} + \alpha \min\{0, x_i\}$$



Maxout

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

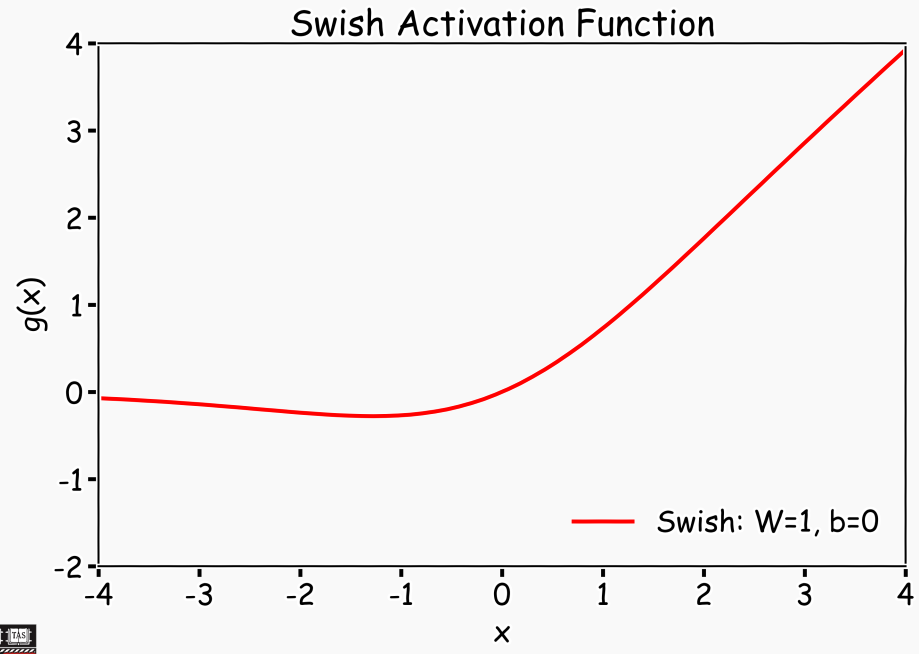
$$g(x) = \max_{i \in \{1,\ldots,k\}} \alpha_i x_i + \beta$$

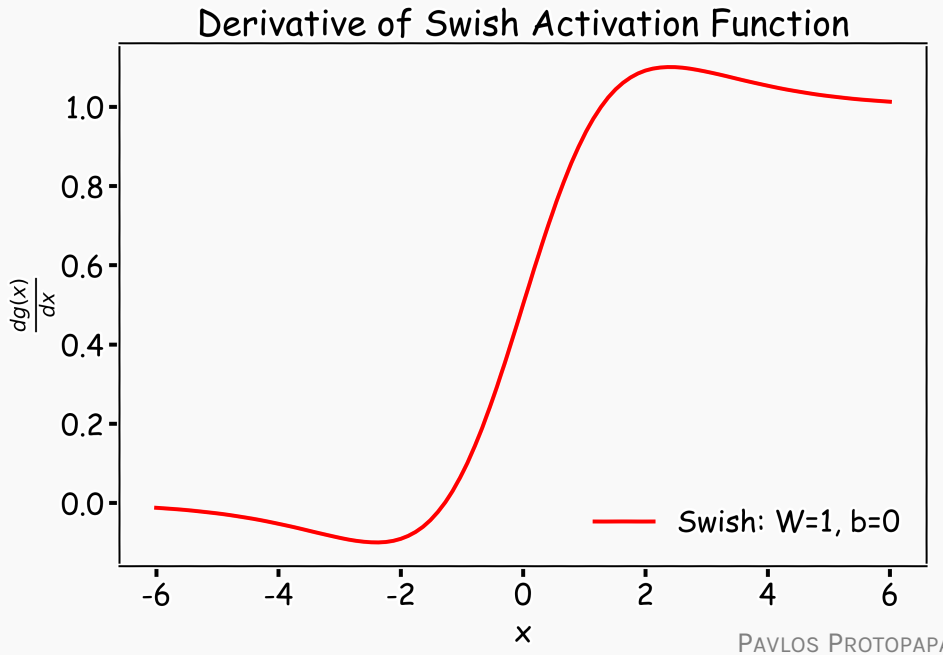


Swish: A Self-Gated Activation Function

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.

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



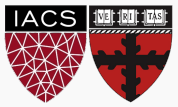


Outline

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)

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

Do not need to design separate loss functions.

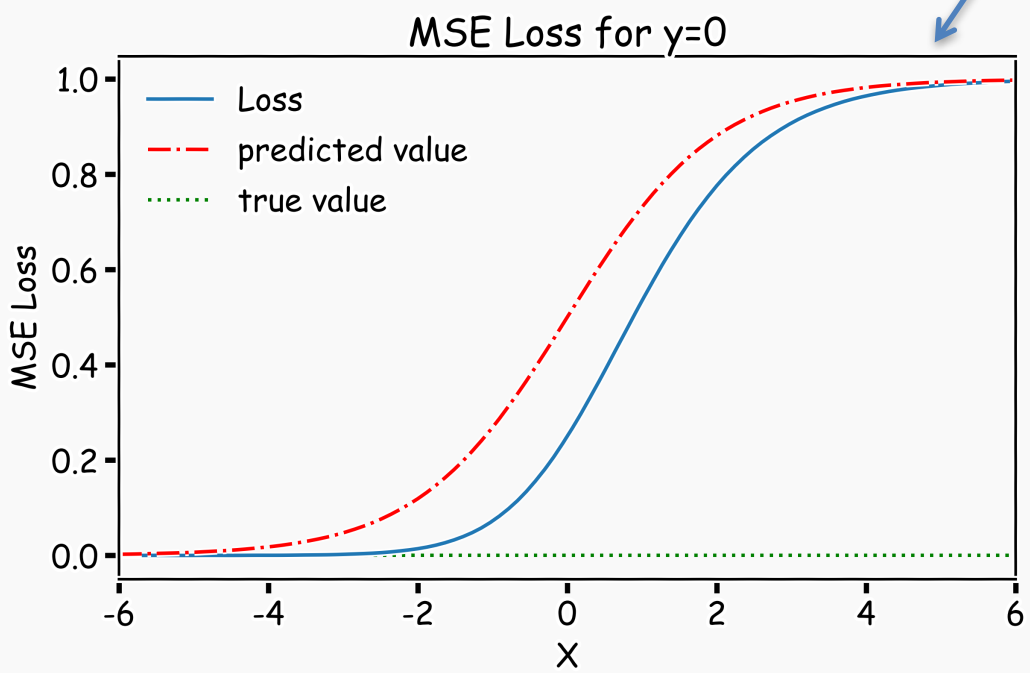
Gradient of cost function must be large enough

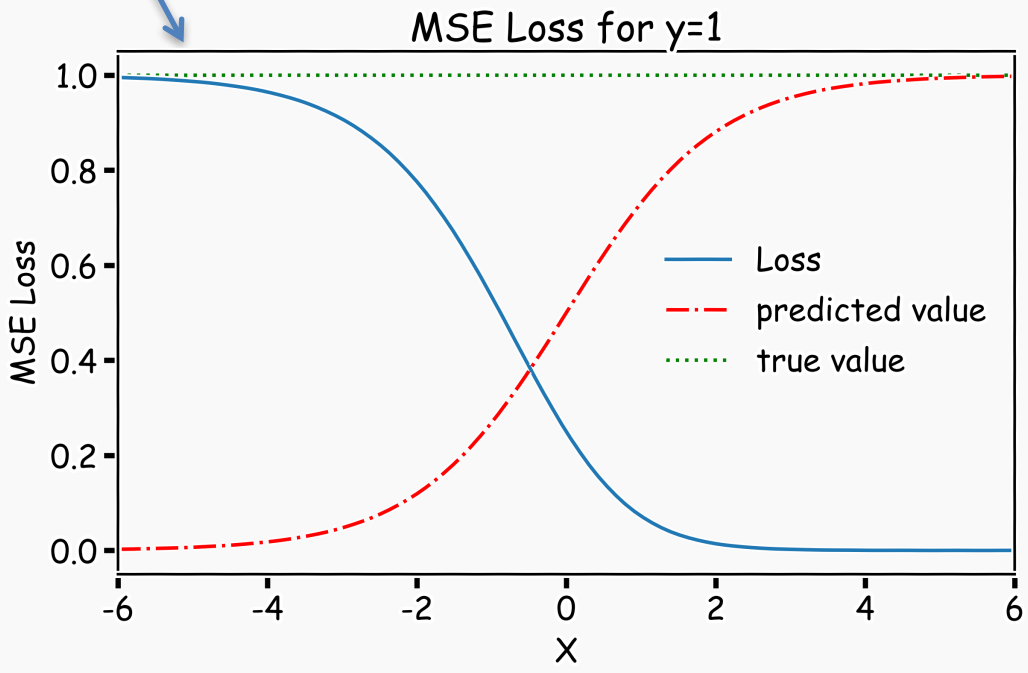
Loss Function

Example: sigmoid output + squared loss

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

Flat surfaces

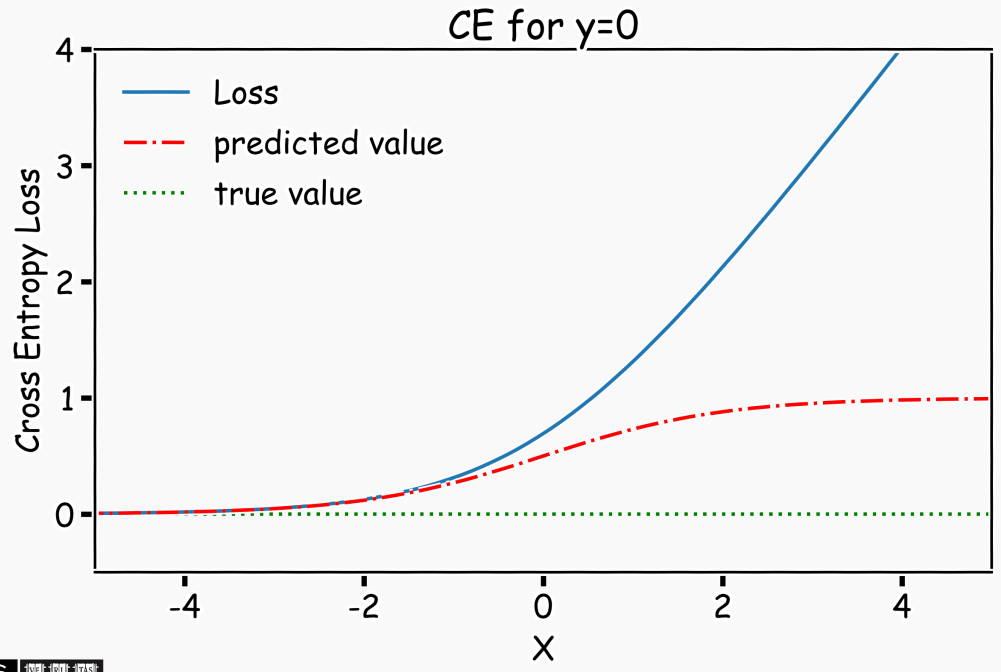


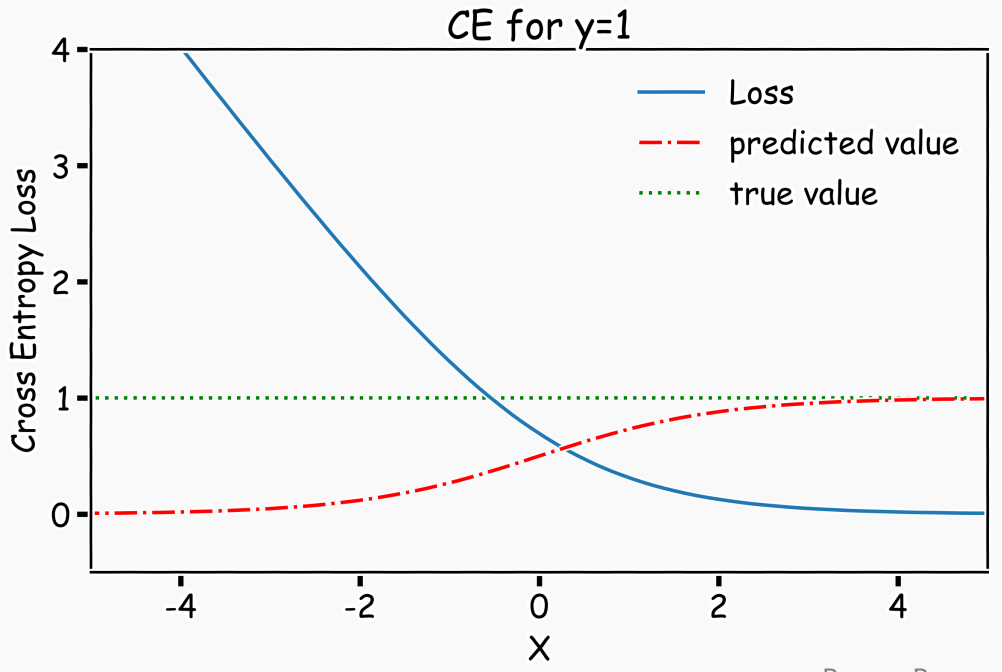


Cost Function

Example: sigmoid output + cross-entropy loss

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





Design Choices

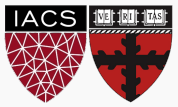
Activation function

Loss function

Output units

Architecture

Optimizer

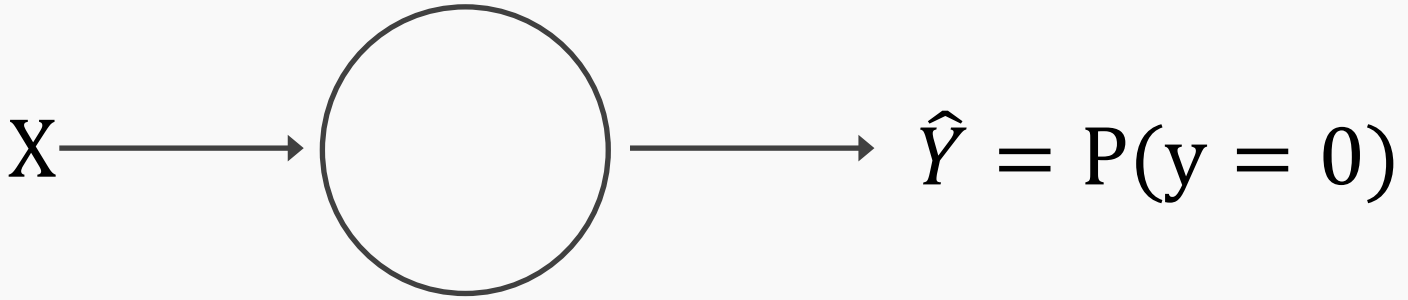


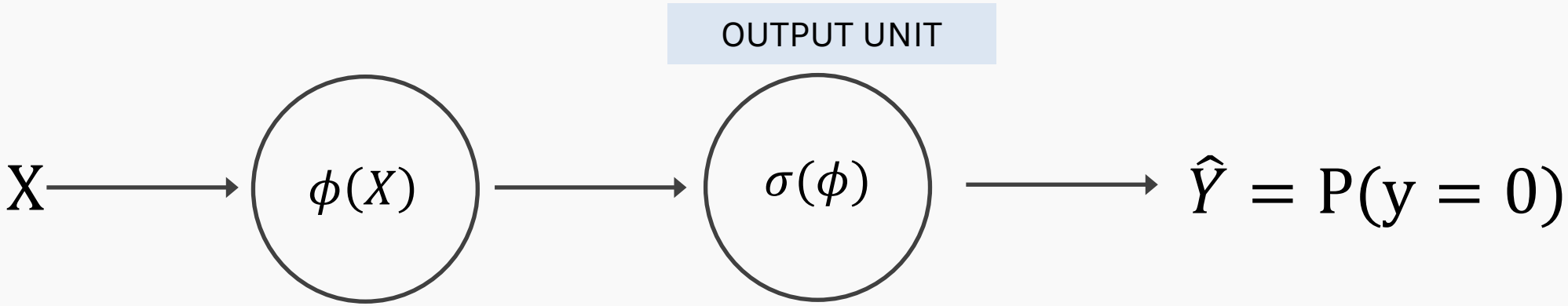
Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary			

Link function

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





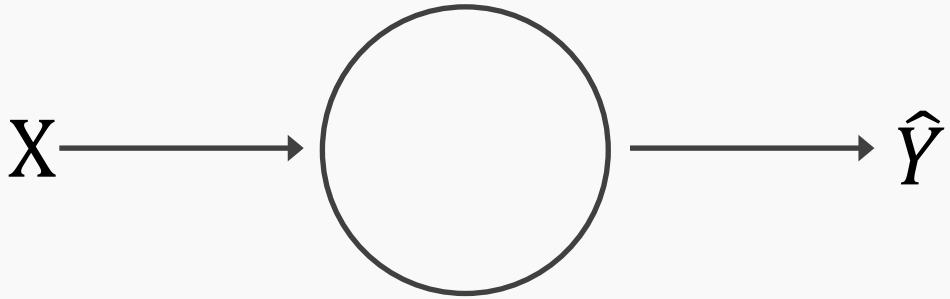
Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy

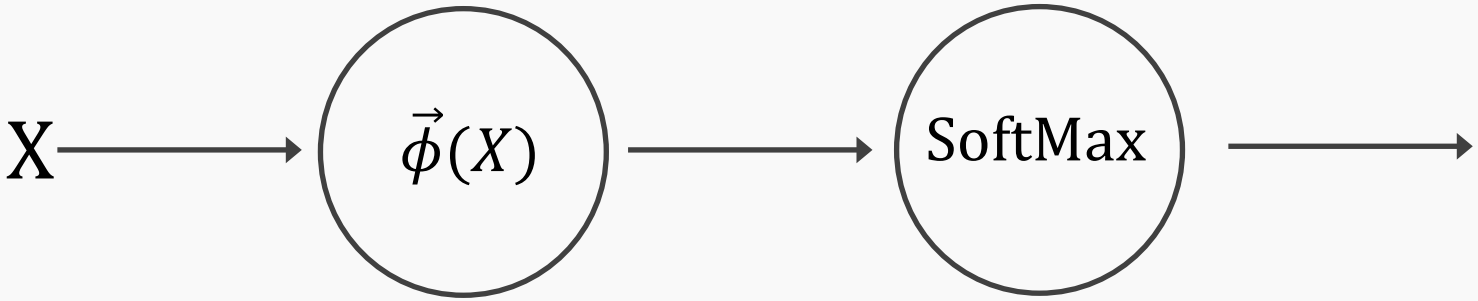
Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete			

Link function multi-class problem



OUTPUT UNIT



$$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$$

Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy

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

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

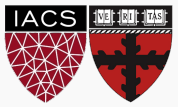
Activation function

Loss function

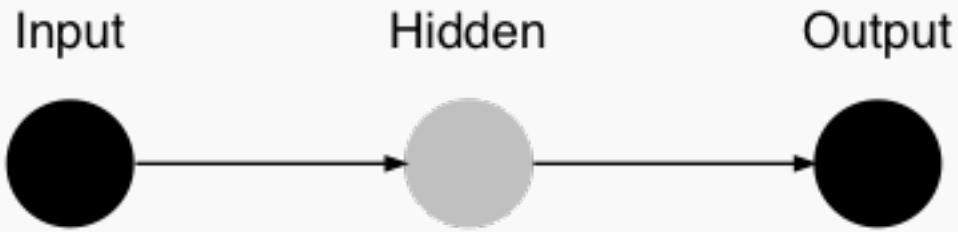
Output units

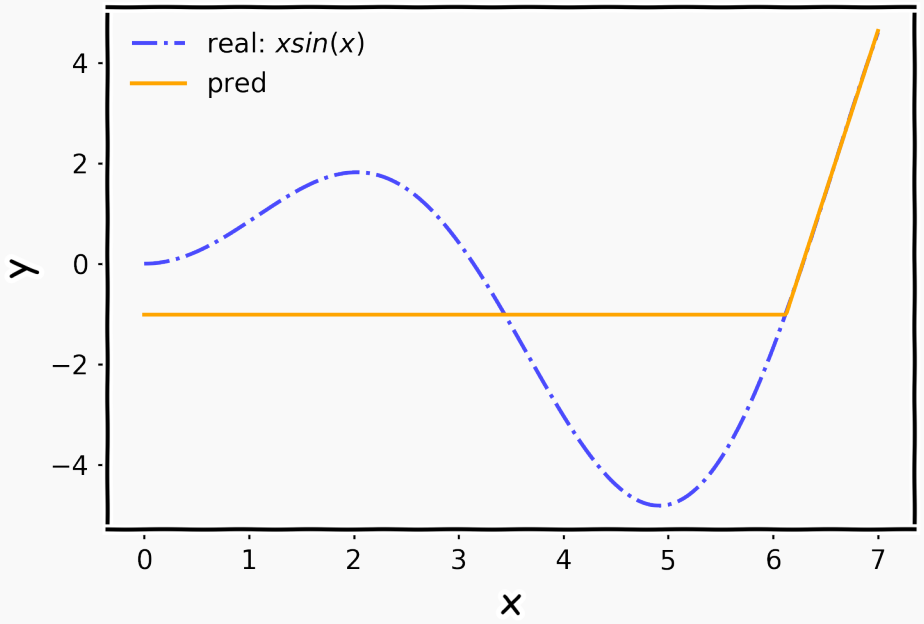
Architecture

Optimizer

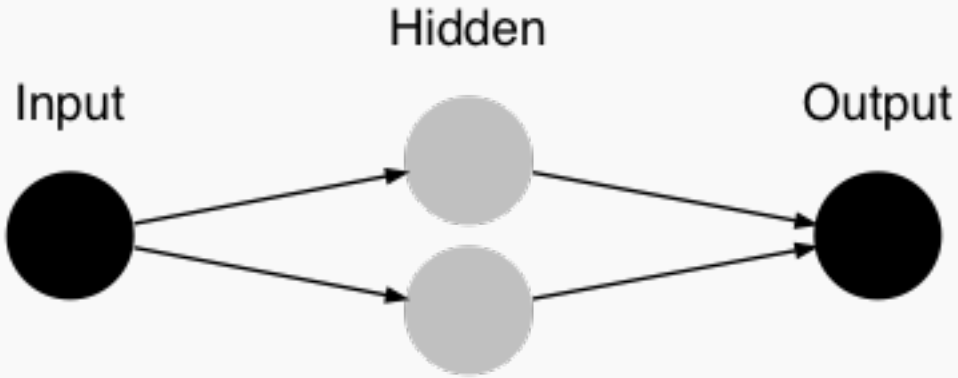


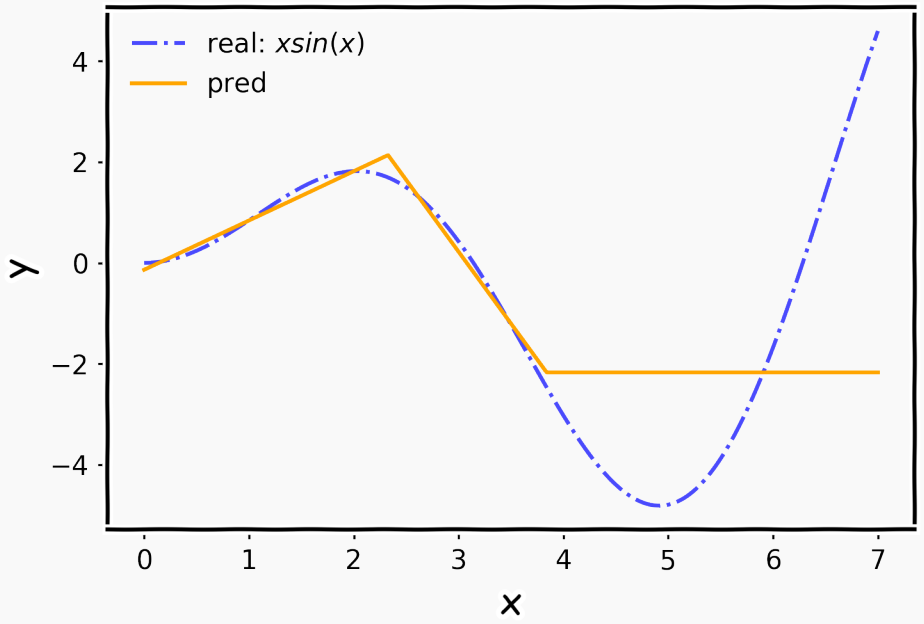
NN in action



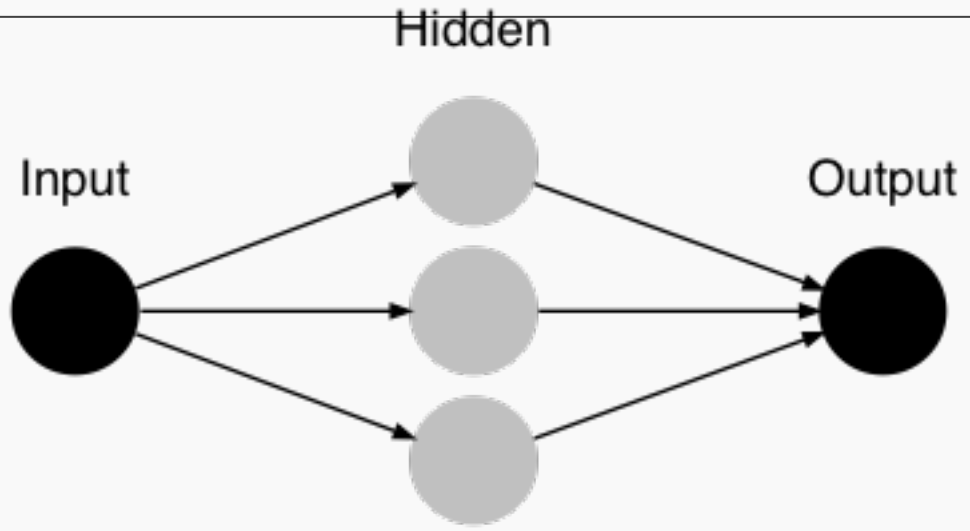


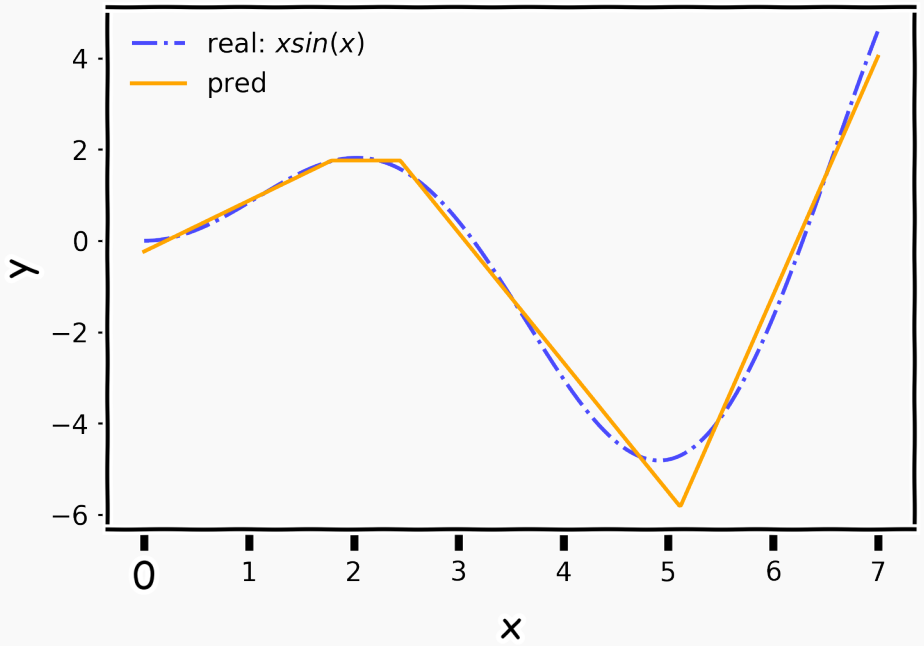
NN in action



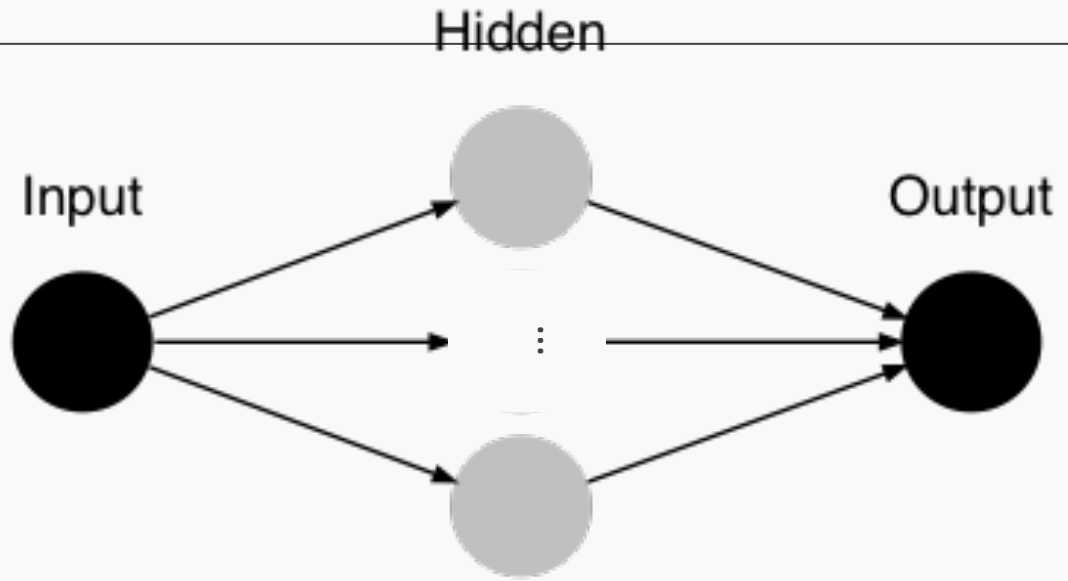


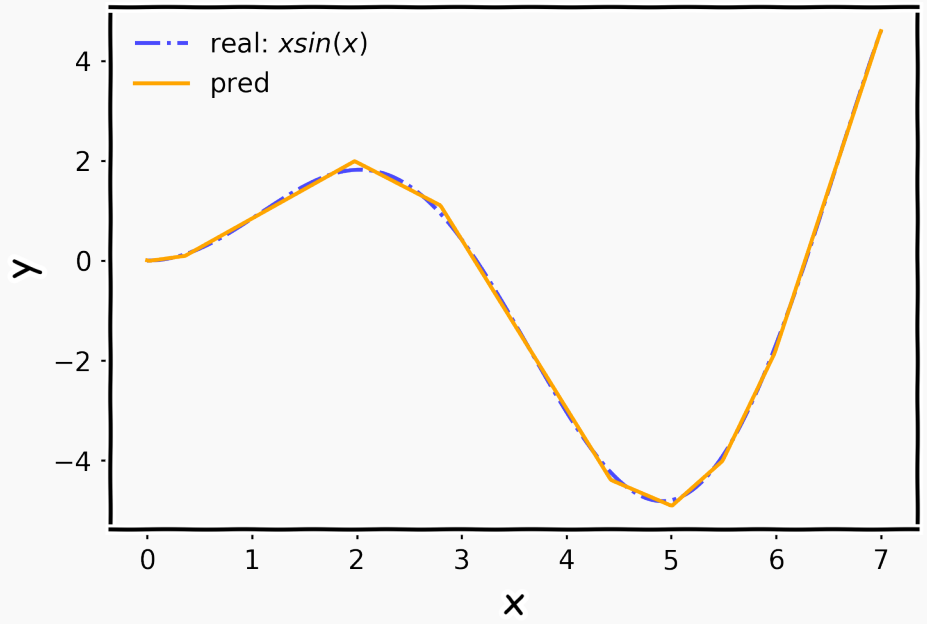
NN in action



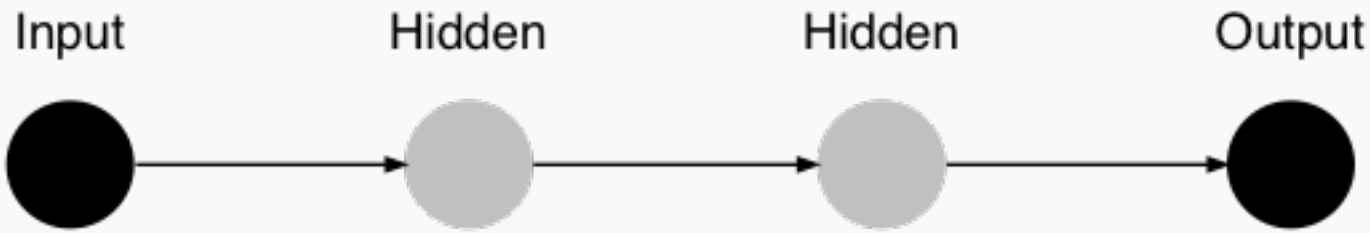


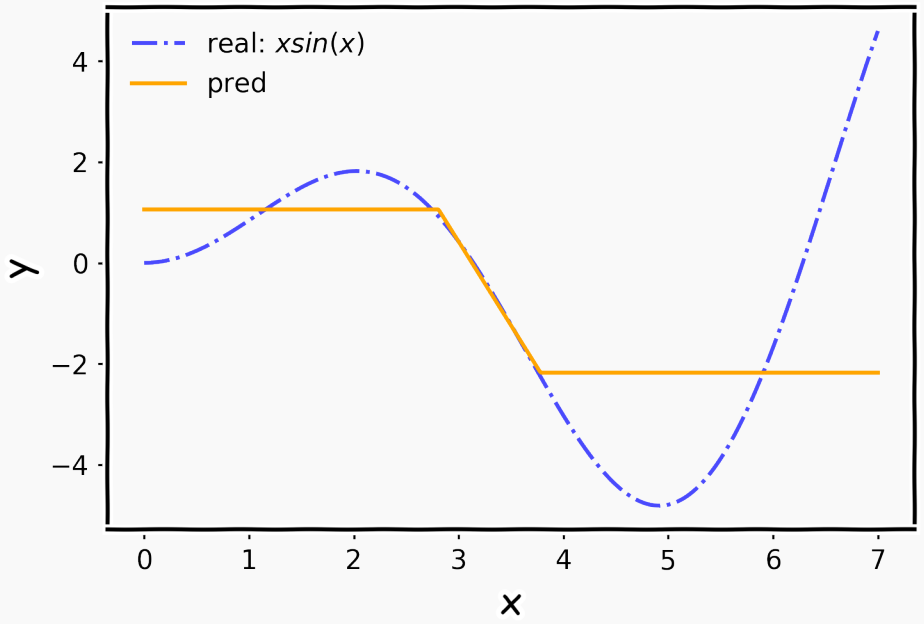
NN in action



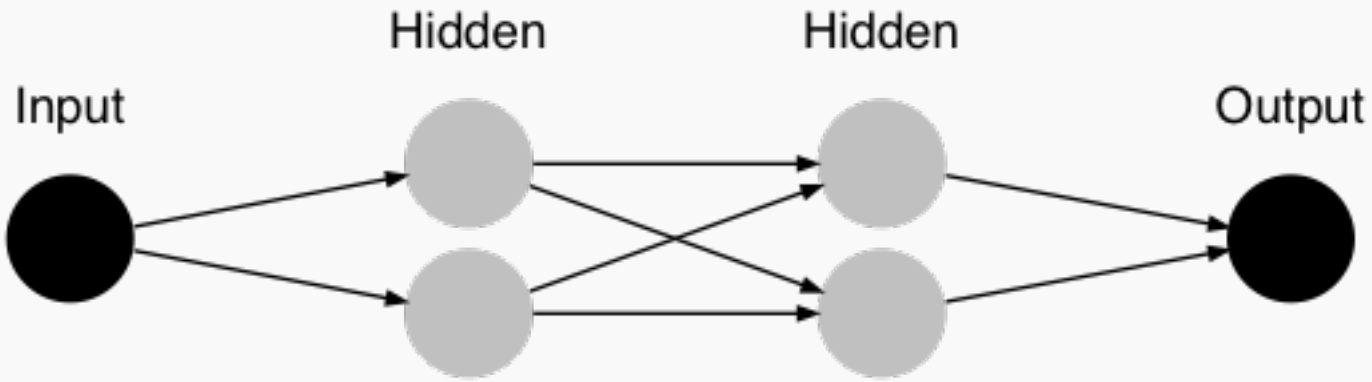


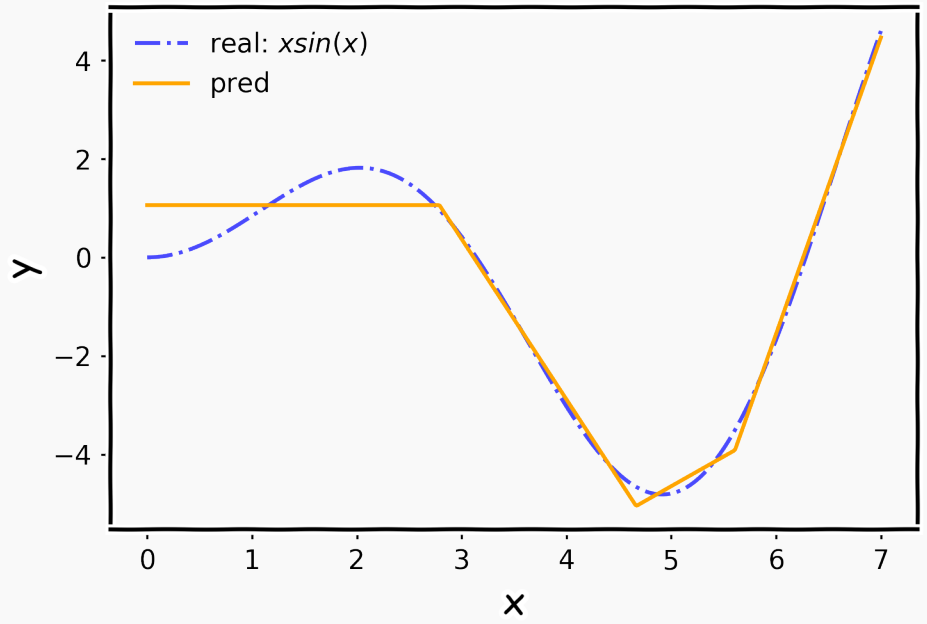
NN in action


Input
Hidden
Hidden
Output

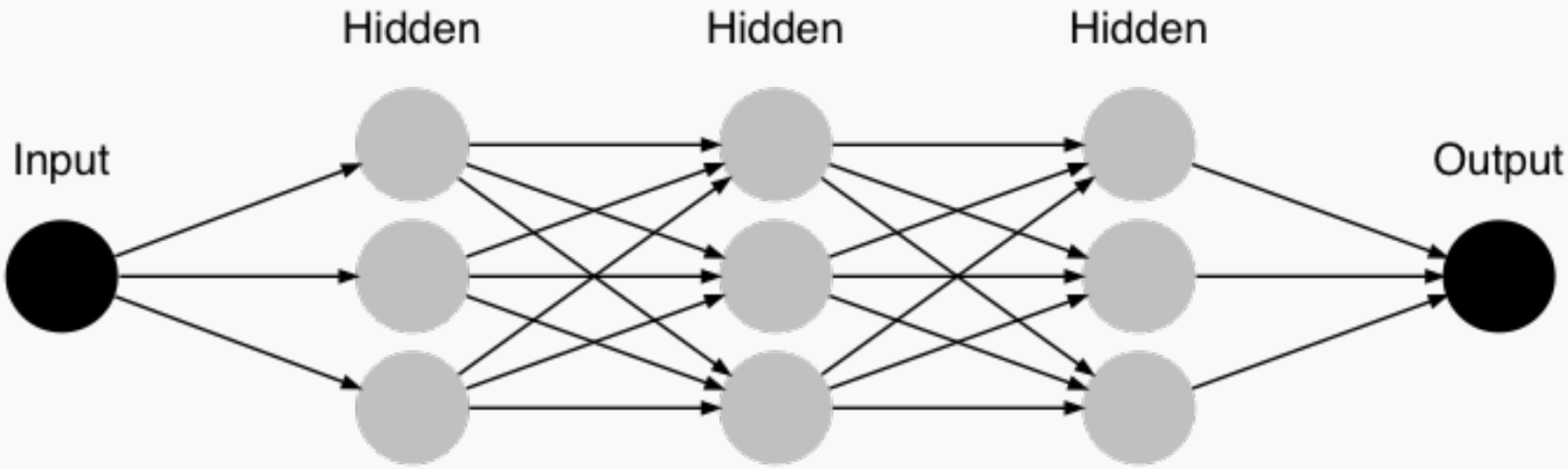

real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

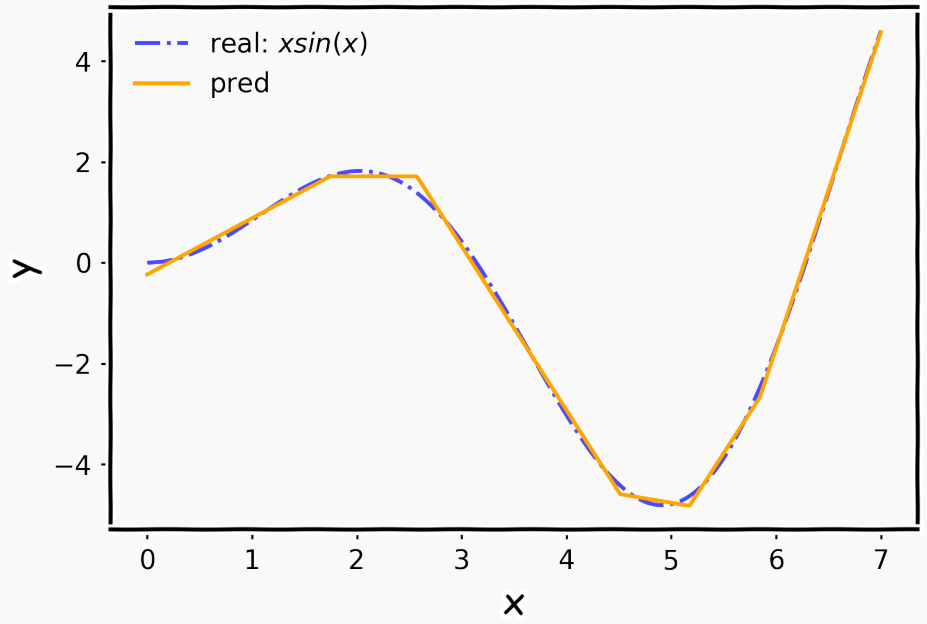
NN in action


Input
Hidden
Hidden
Output


real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

NN in action


Hidden
Hidden
Hidden
Input
Output


real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

Universal Approximation Theorem

Think of a Neural Network as function approximation.

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

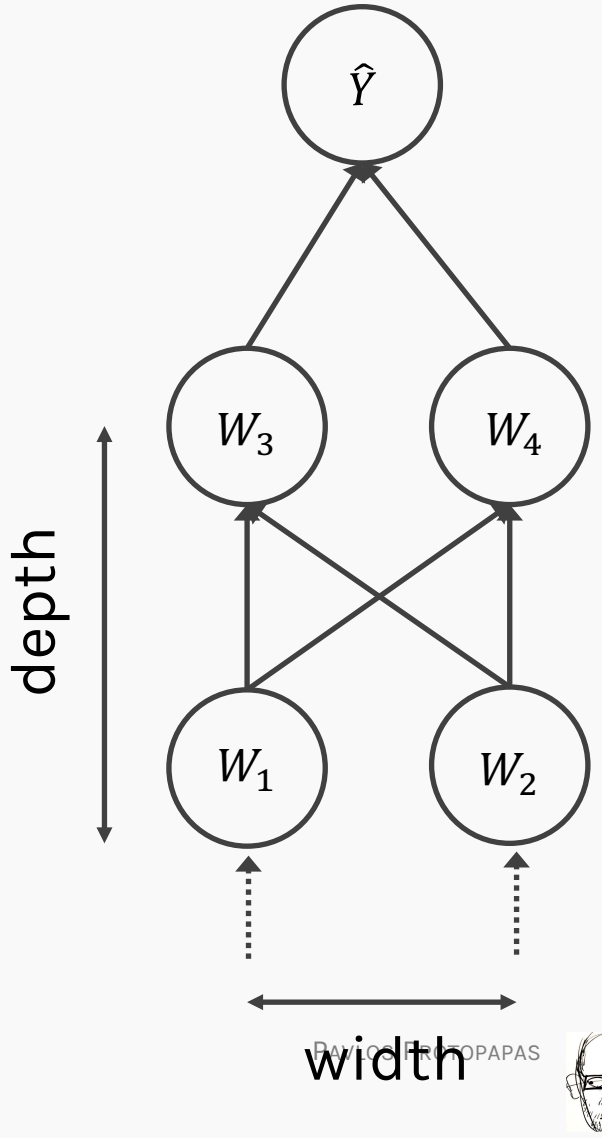
$$Y = \hat{f}(x) + \epsilon$$

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

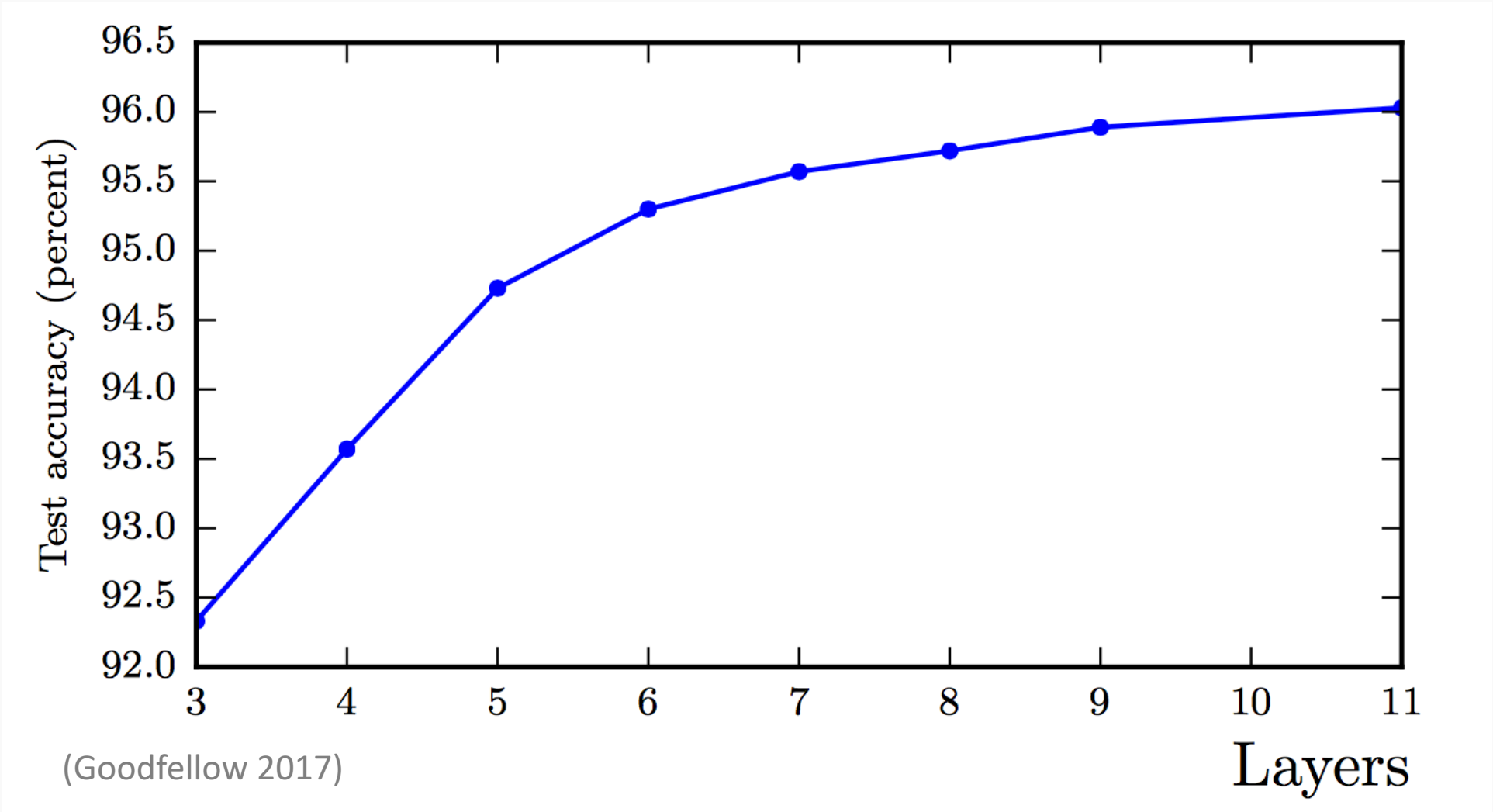
One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy

So why deeper?

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



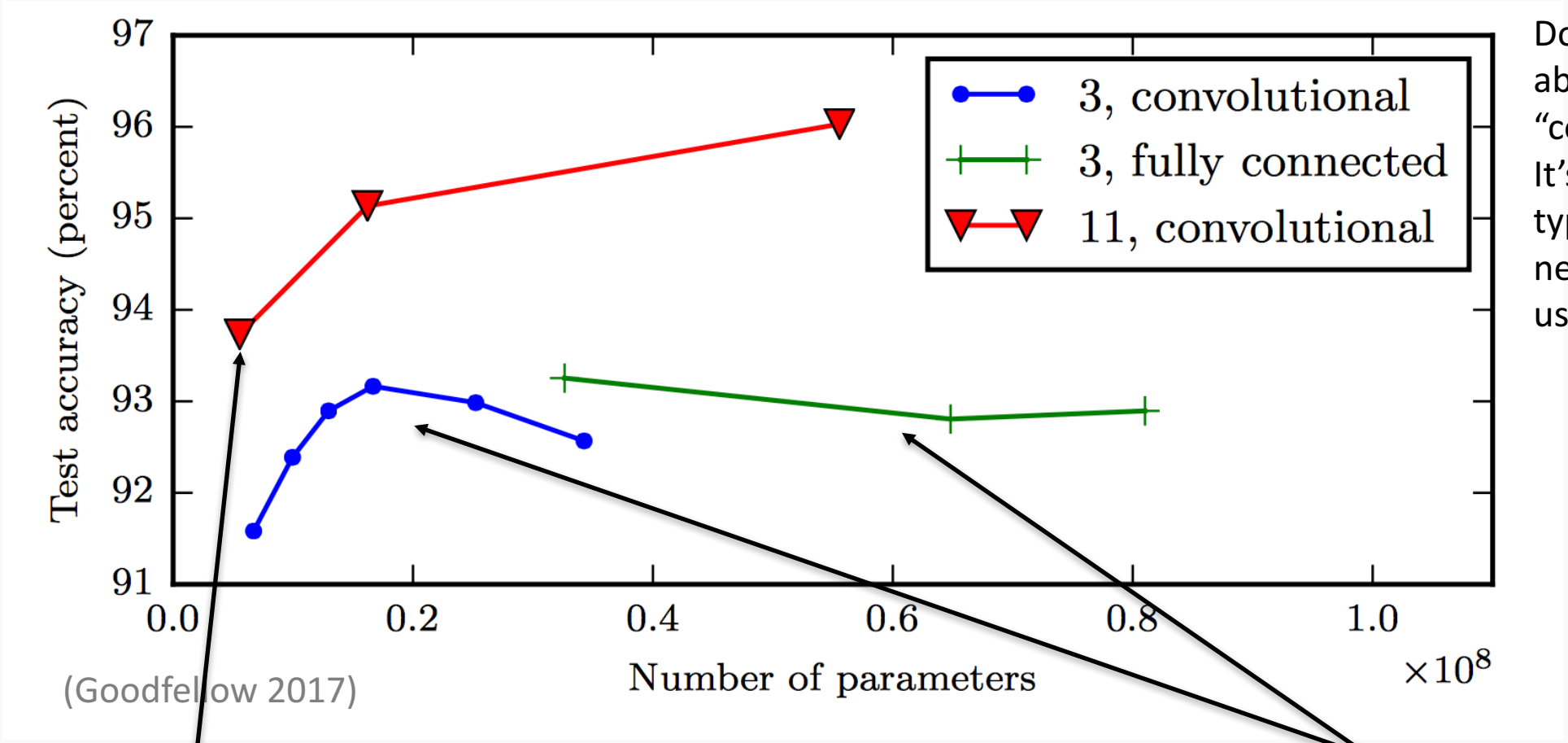
Better Generalization with Depth



(Goodfellow 2017)

Shallow Nets Overfit More

Depth helps, and it's not just because of more parameters



(Goodfellow 2017)

Don't worry about this word "convolutional". It's just a special type of neural network, often used for images.

The **11-layer net** generalizes better on the test set when controlling for number of parameters.

The 3-layer nets perform worse on the test set, even with similar number of total parameters.