Lecture 19: Anatomy of NN

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Tanner

Outline

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture

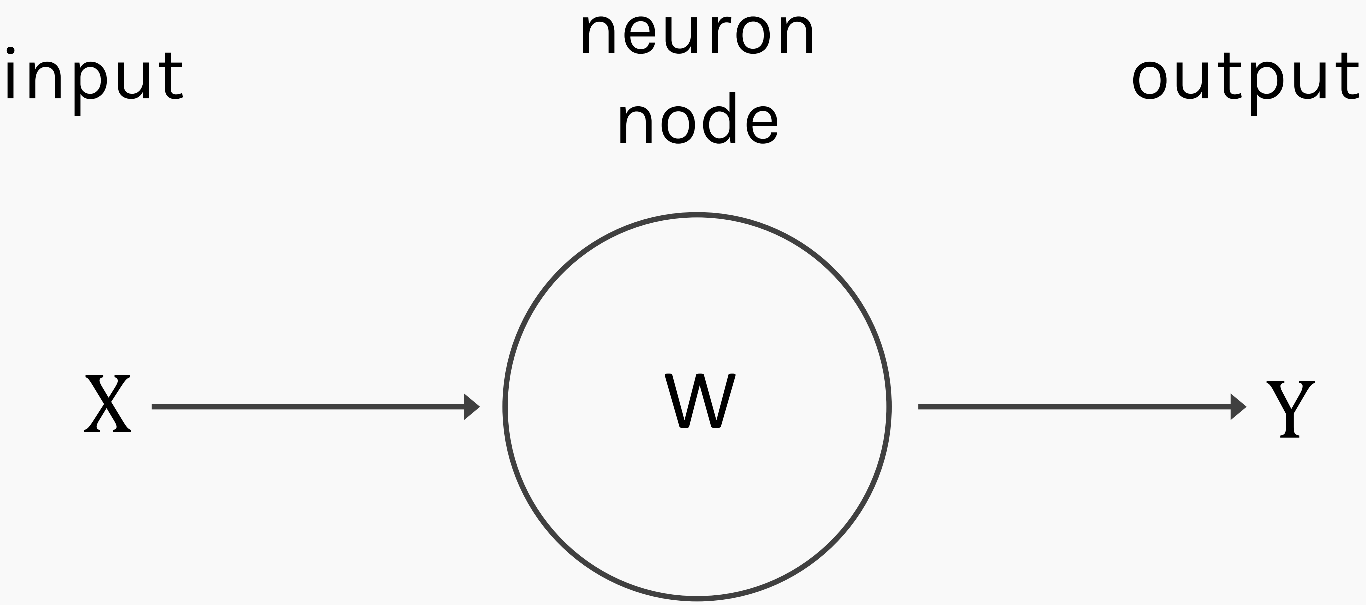
Outline

Anatomy of a NN

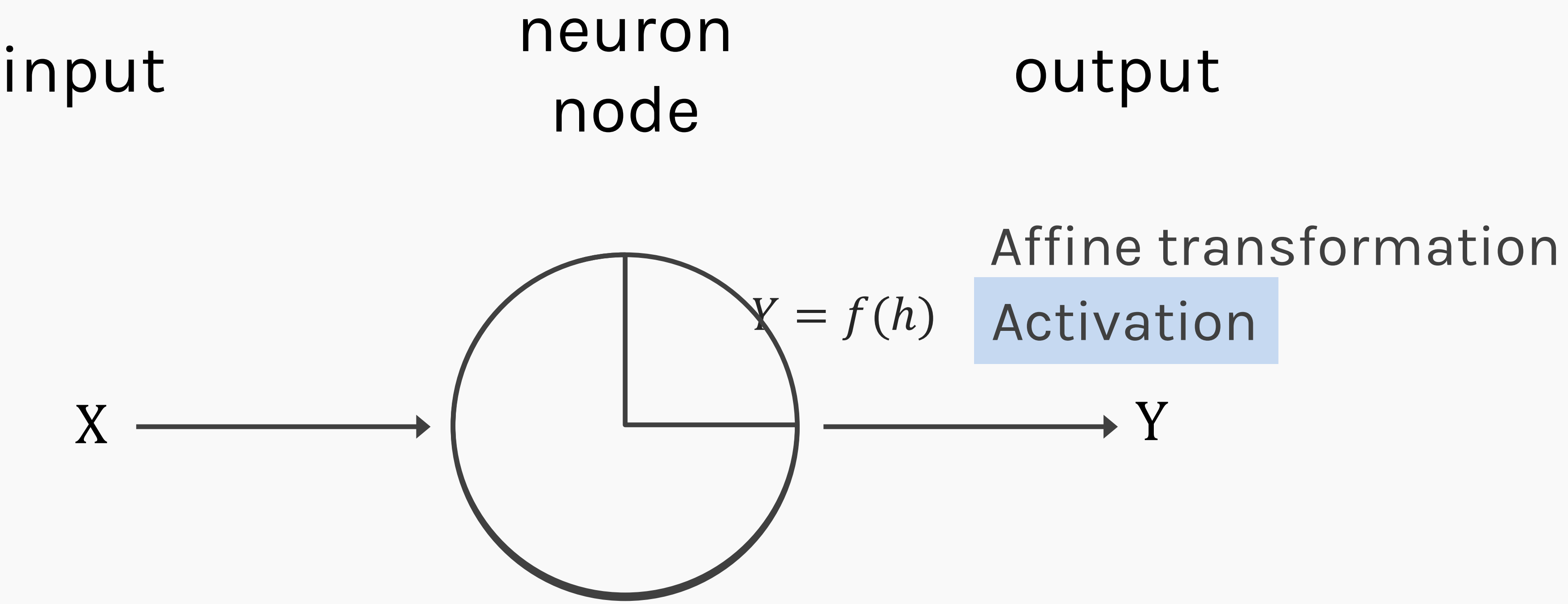
Design choices

- Activation function
- Loss function
- Output units
- Architecture

Anatomy of artificial neural network (ANN)



Anatomy of artificial neural network (ANN)

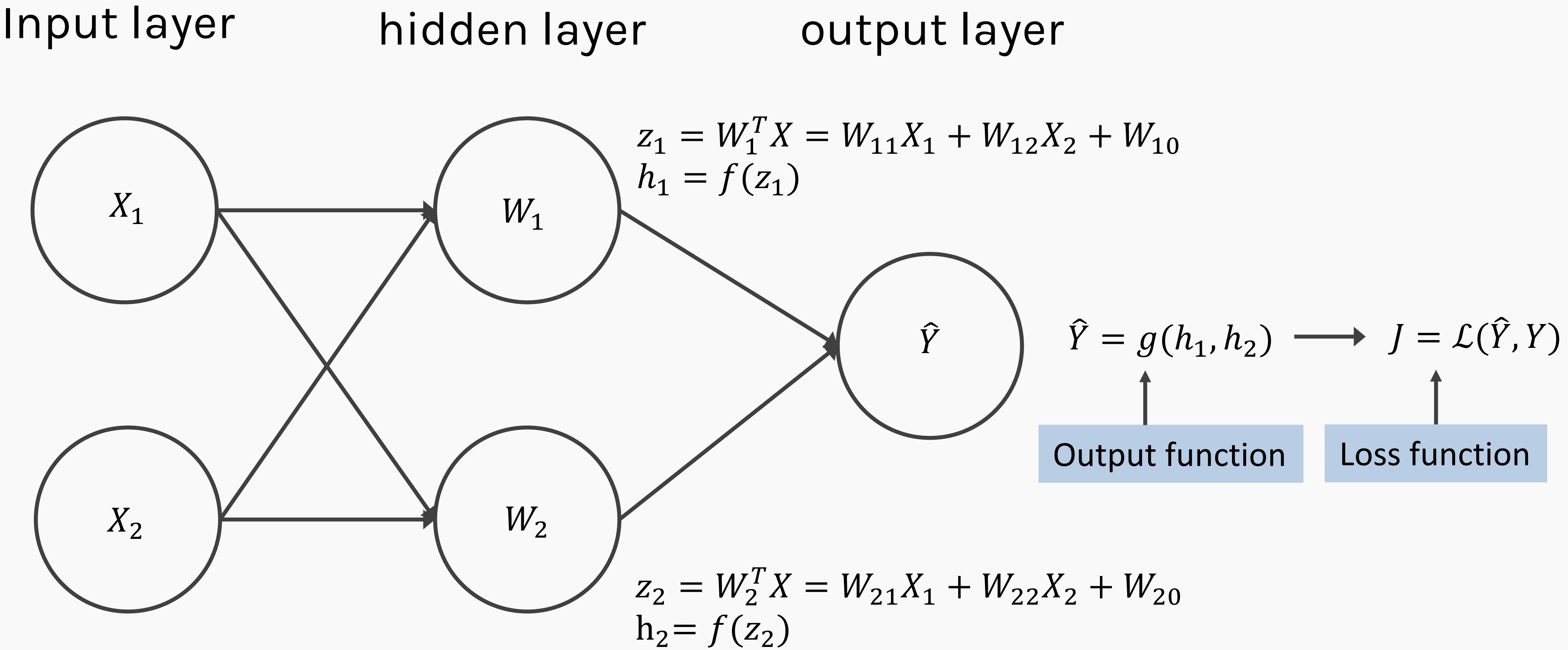


Affine transformation

Activation

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)



$z_1 = W_1^T X = W_{11}X_1 + W_{12}X_2 + W_{10}$
$h_1 = f(z_1)$

$z_2 = W_2^T X = W_{21}X_1 + W_{22}X_2 + W_{20}$
$h_2 = f(z_2)$

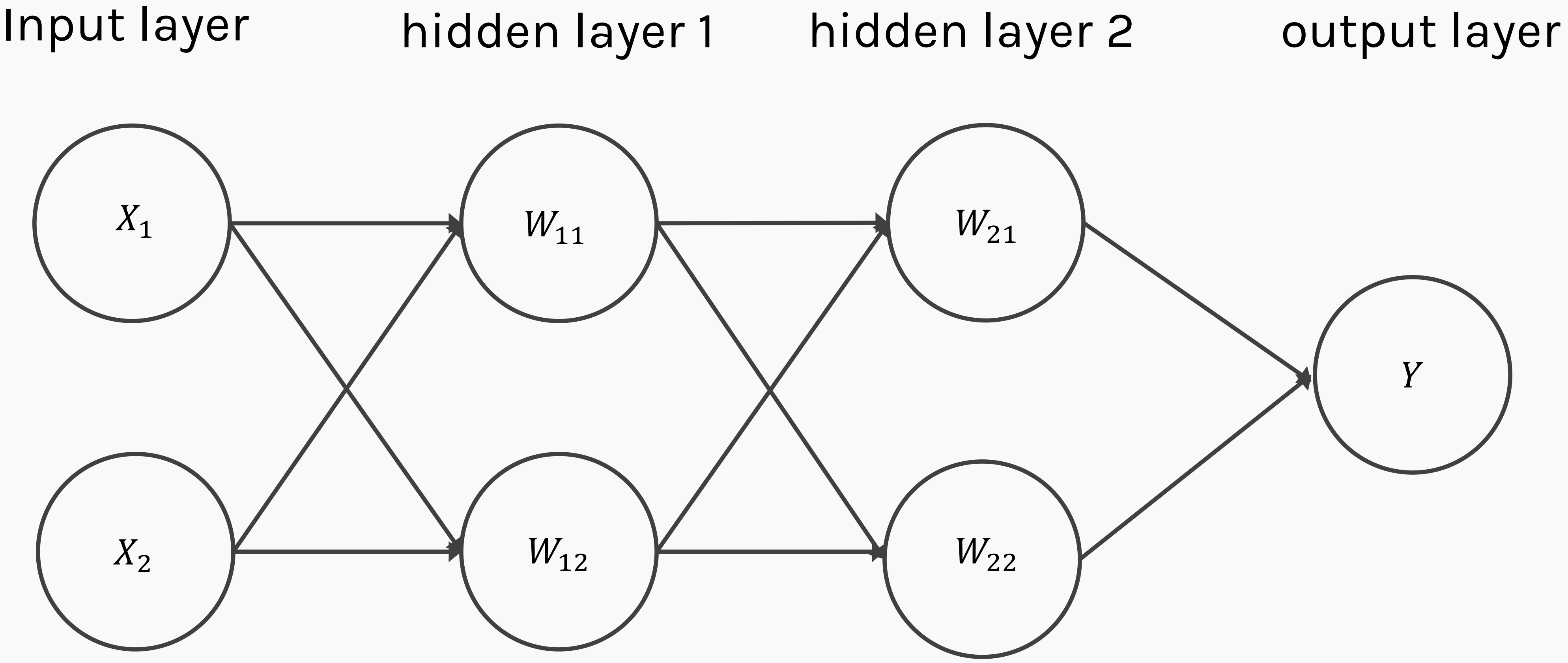
$\hat{Y} = g(h_1, h_2) \longrightarrow J = \mathcal{L}(\hat{Y}, Y)$

Output function

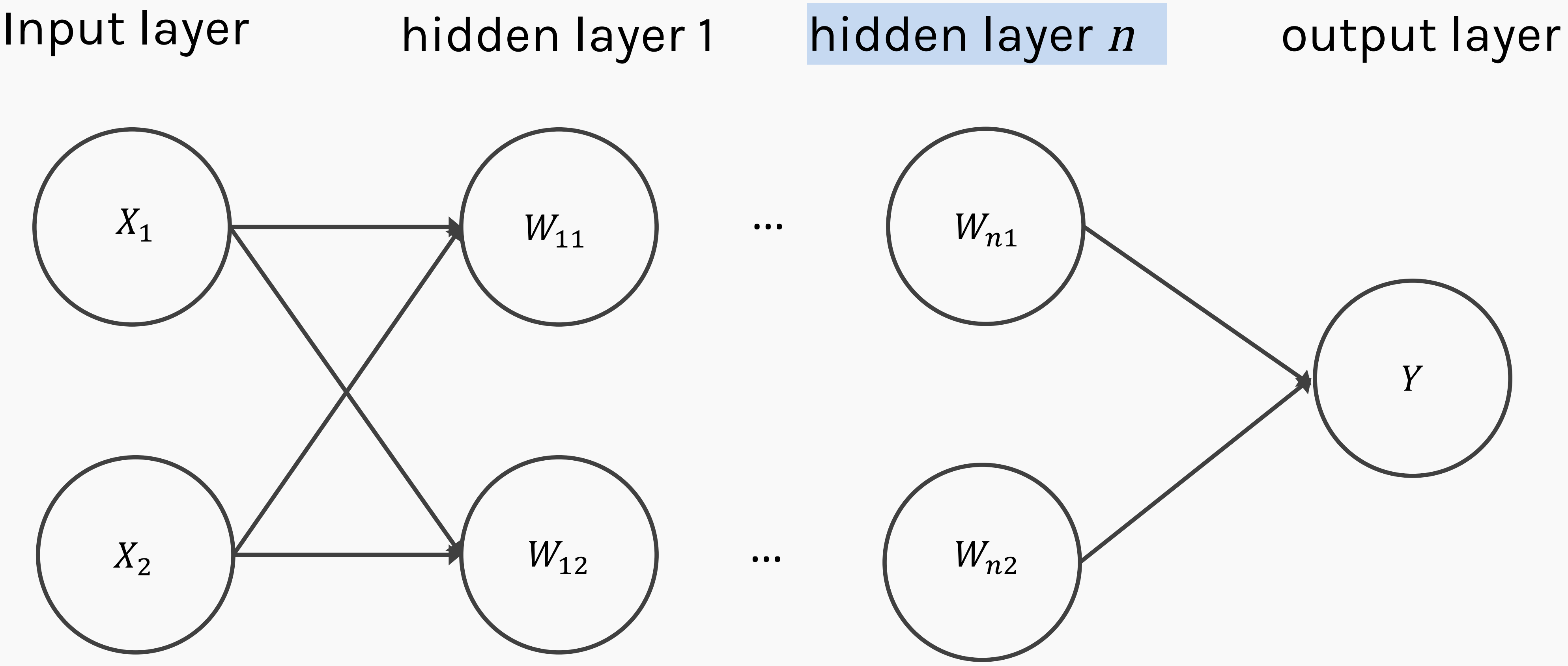
Loss function

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)

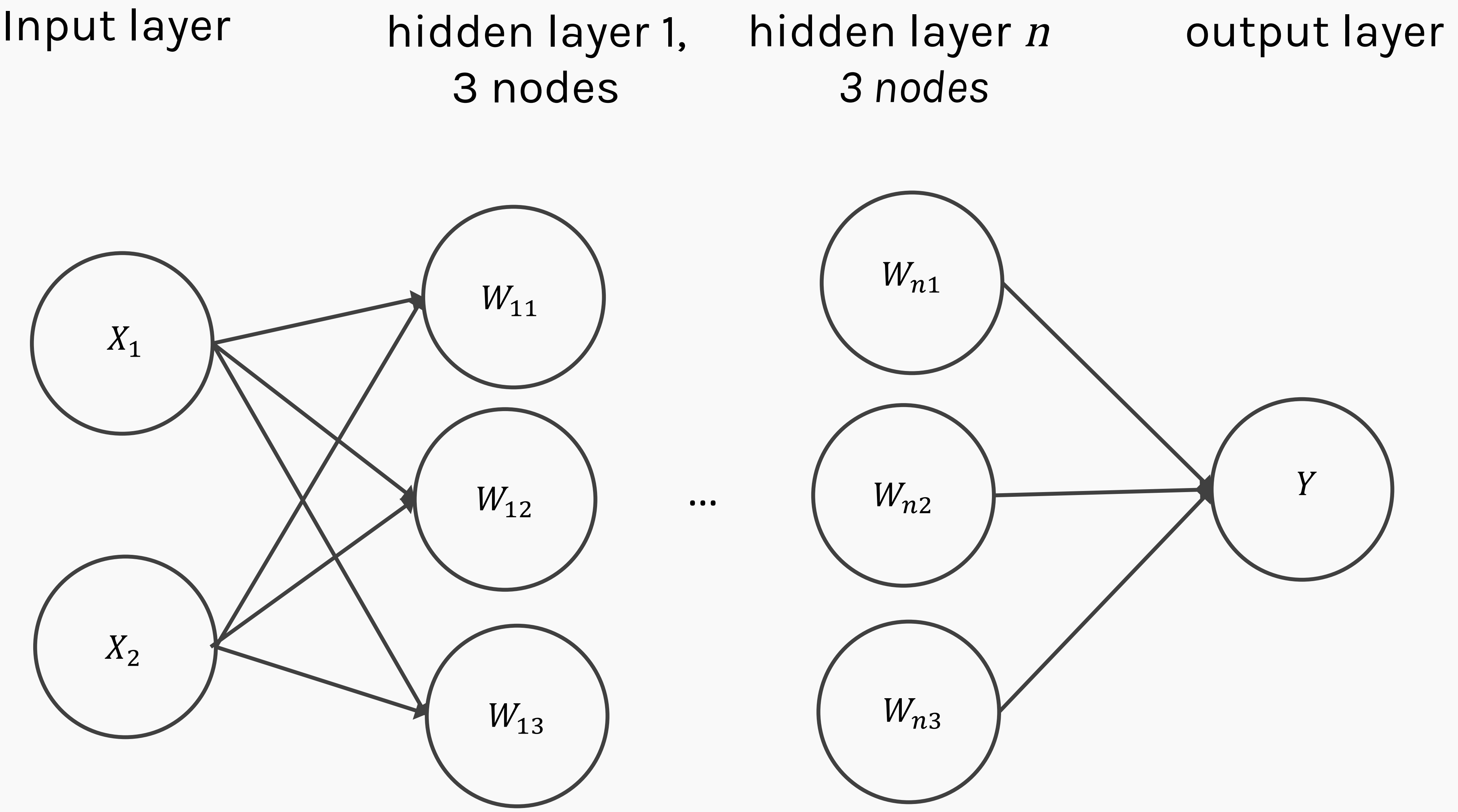


Anatomy of artificial neural network (ANN)



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

Anatomy of artificial neural network (ANN)



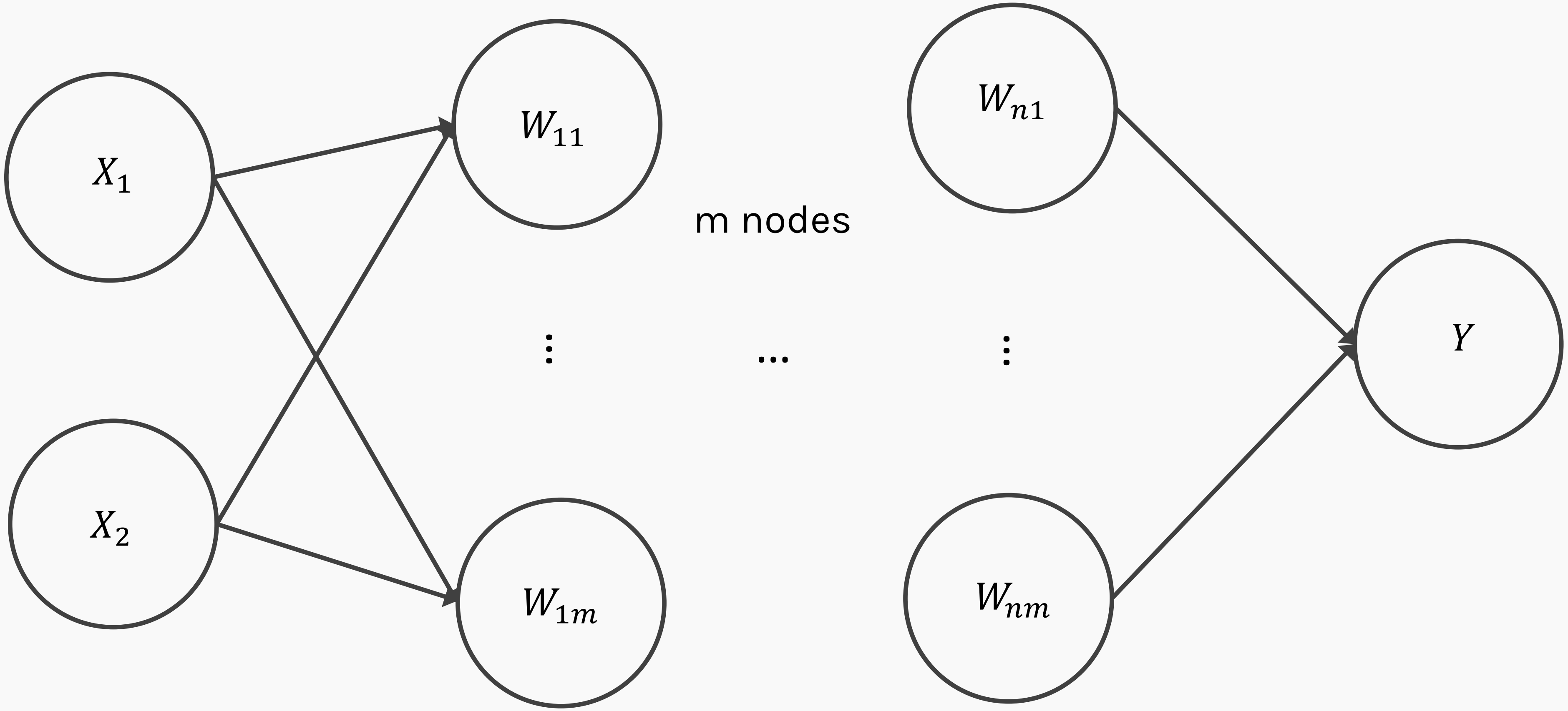
Anatomy of artificial neural network (ANN)

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)

Input layer

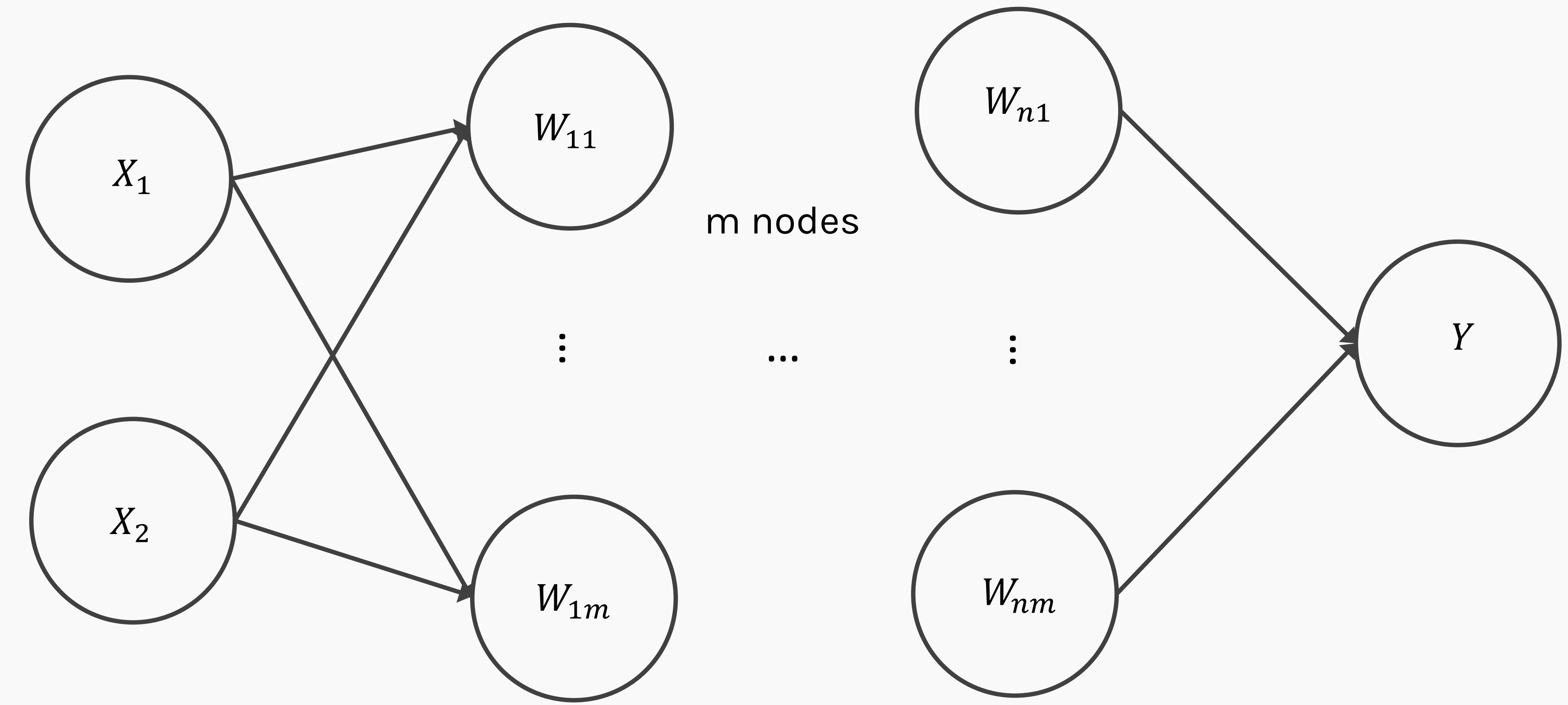
hidden layer 1,

m nodes

hidden layer n

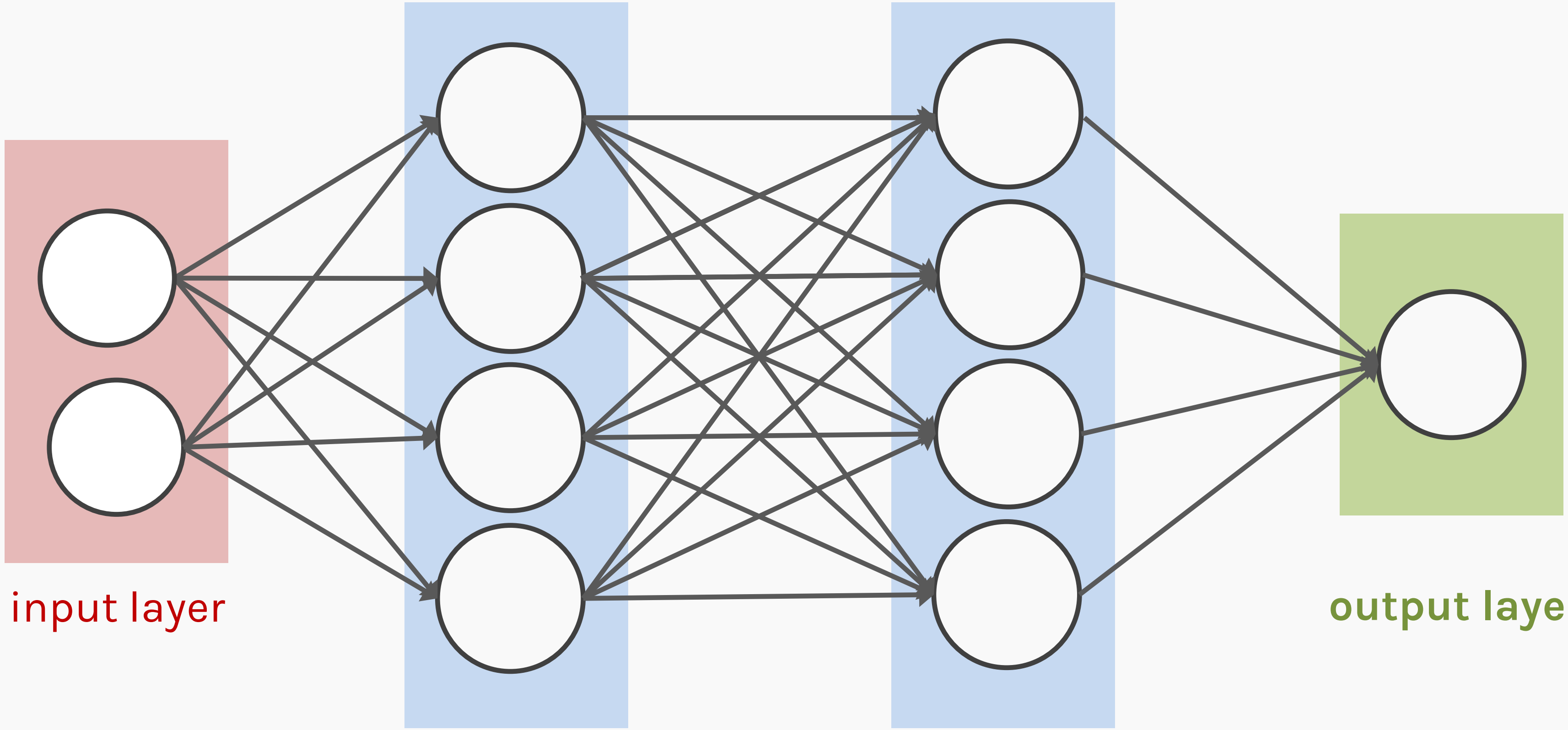
output layer

Number of inputs d



Number of inputs is specified by the data

Anatomy of artificial neural network (ANN)



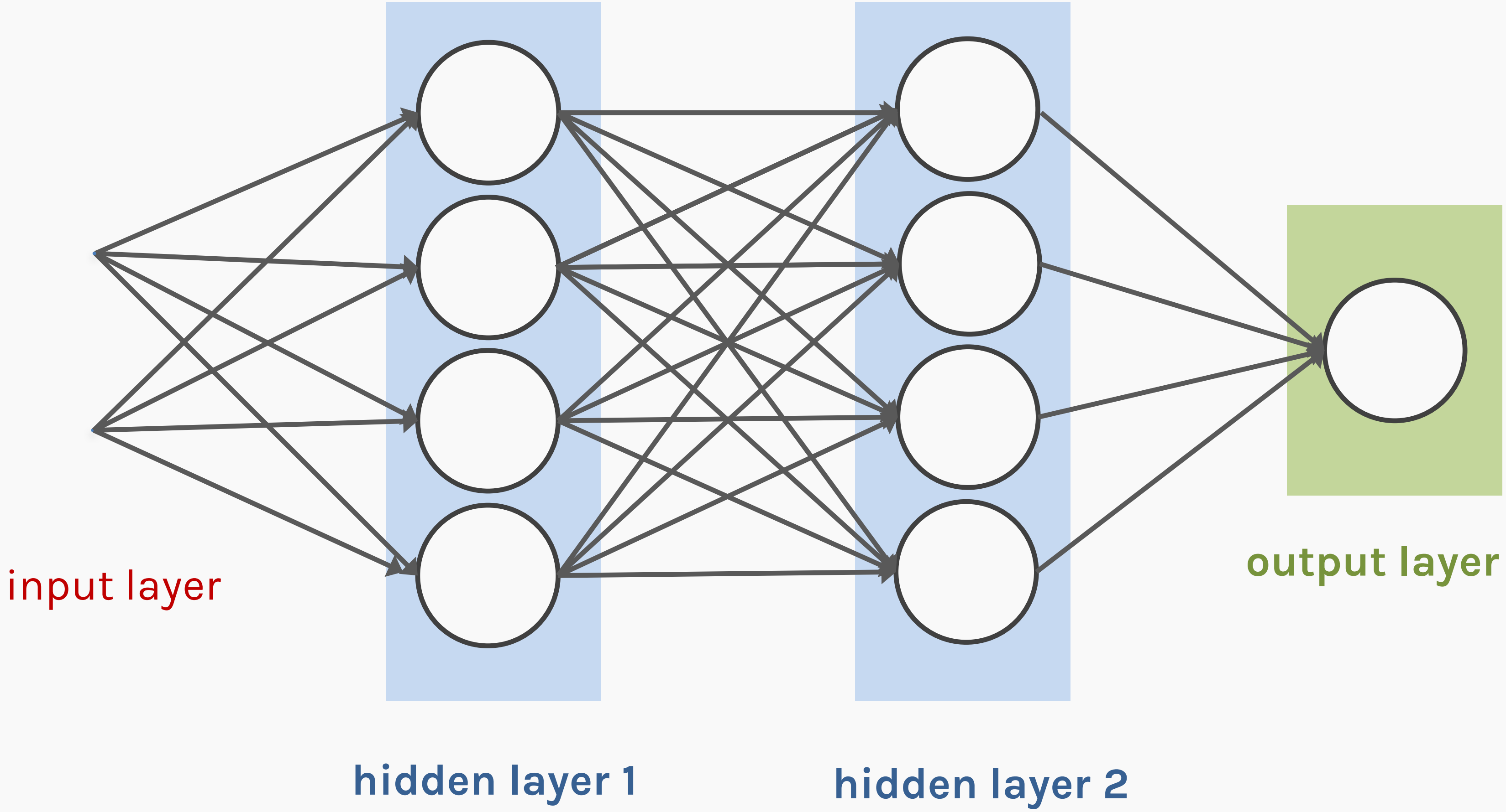
input layer

output layer

hidden layer 1

hidden layer 2

Anatomy of artificial neural network (ANN)



input layer

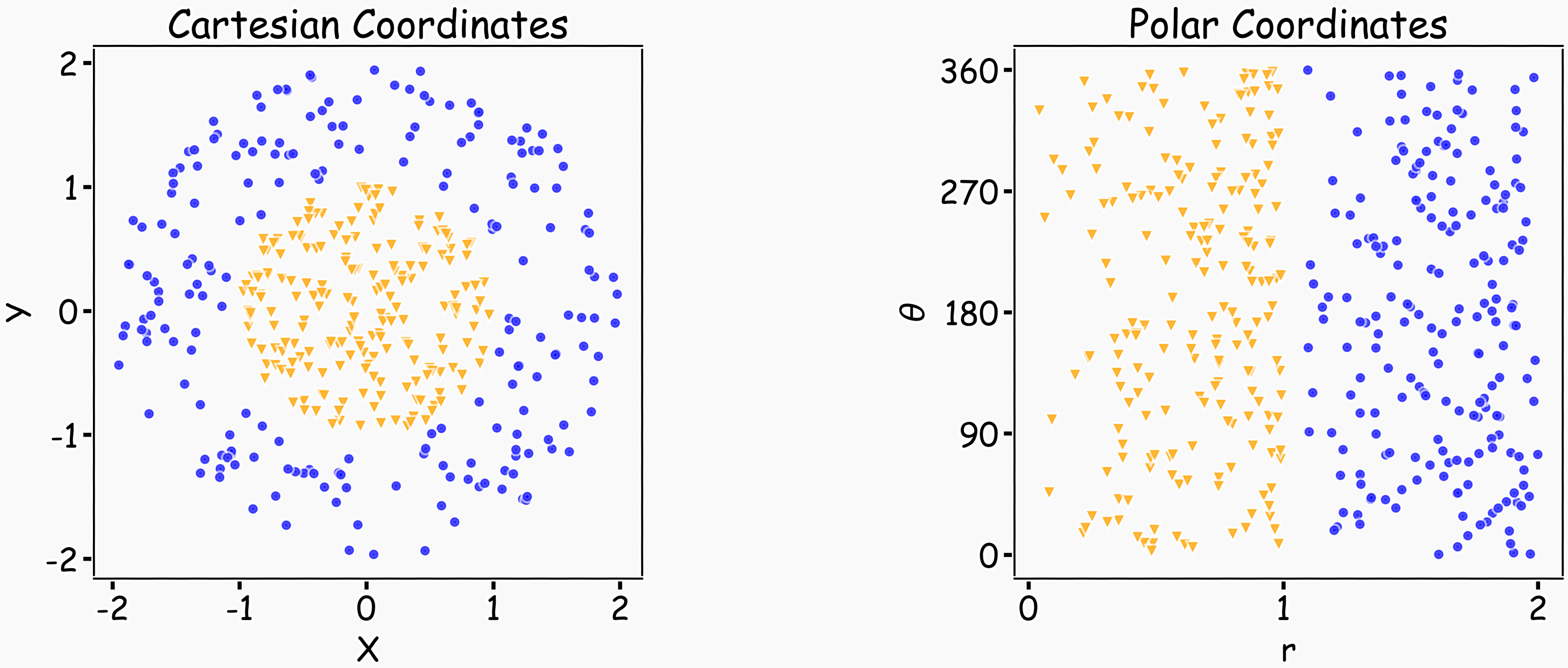
hidden layer 1

hidden layer 2

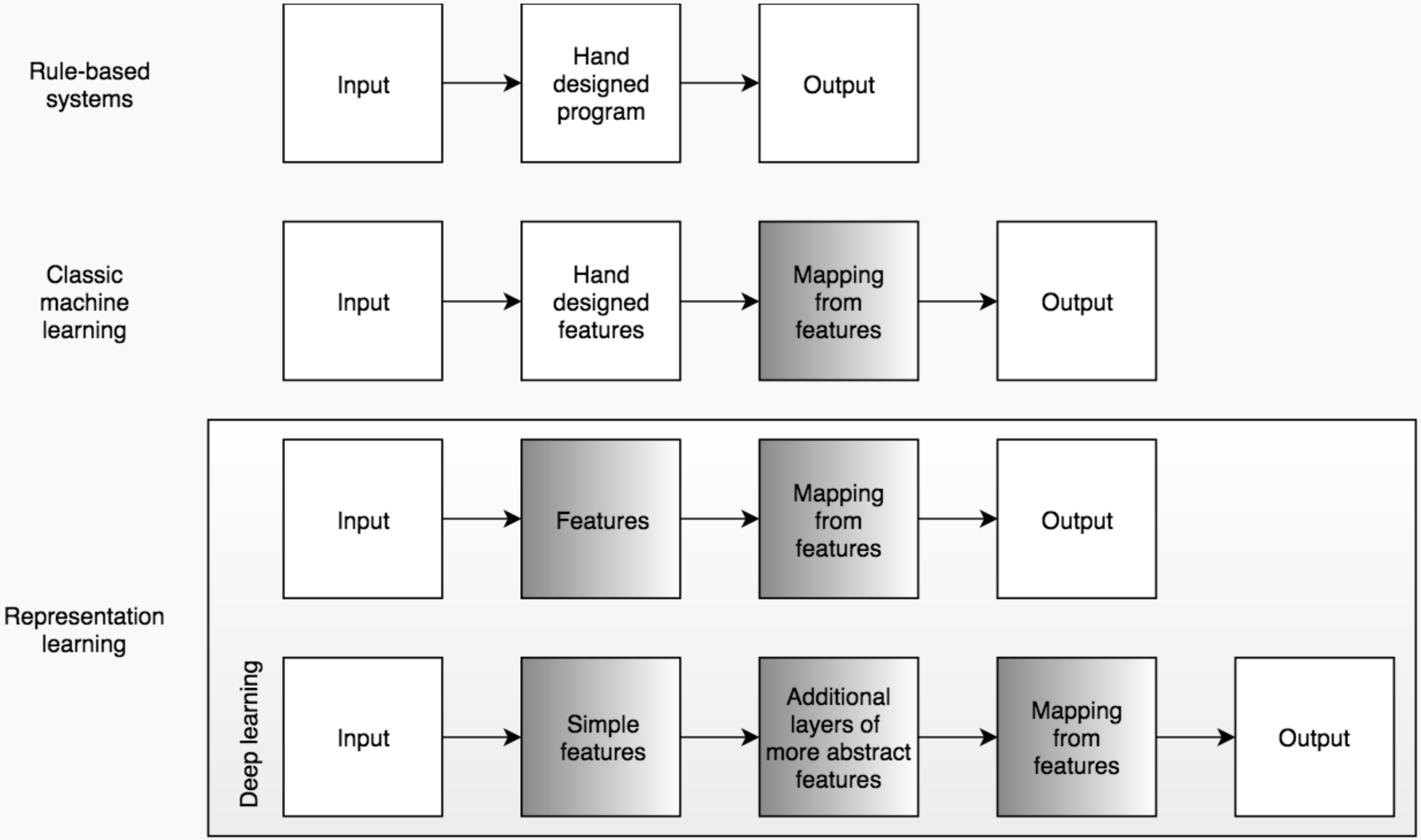
output layer

Why layers? Representation

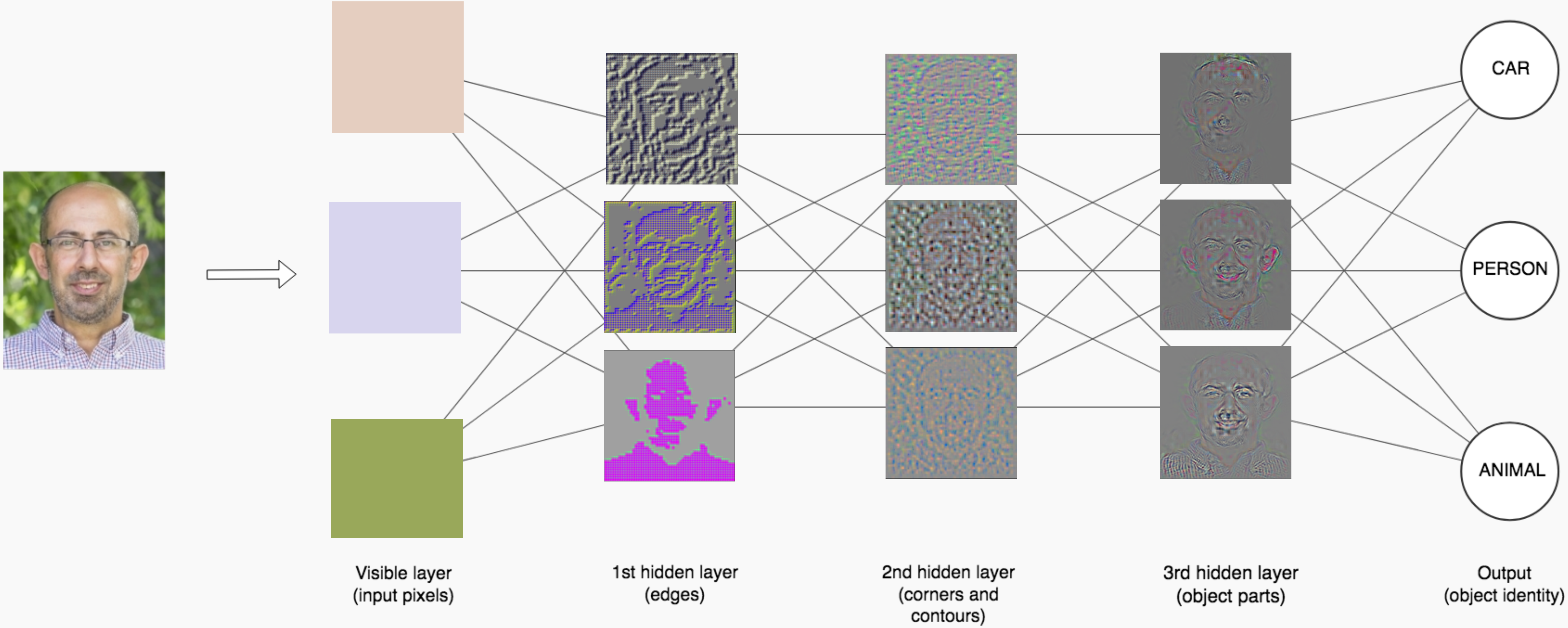
Representation matters!



Learning Multiple Components



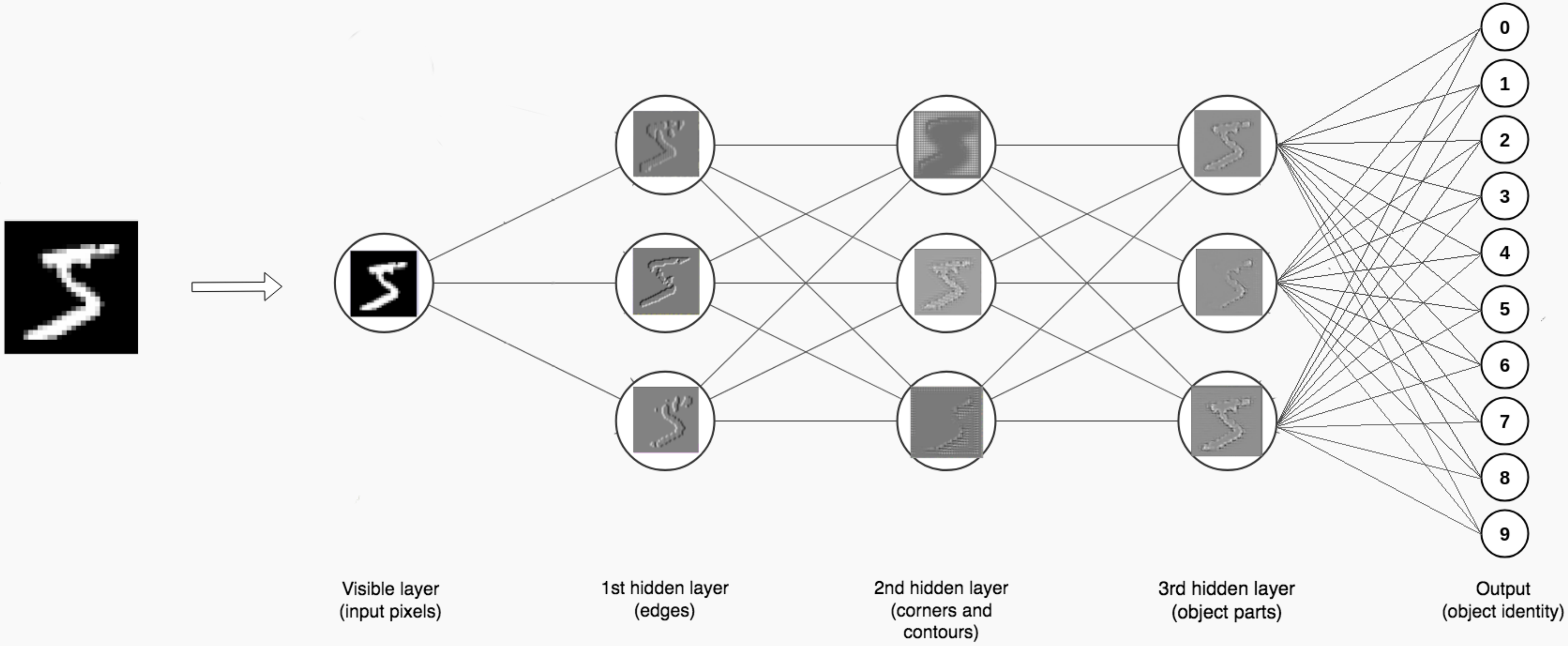
Depth = Repeated Compositions



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

Activation function

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

The activation function should:

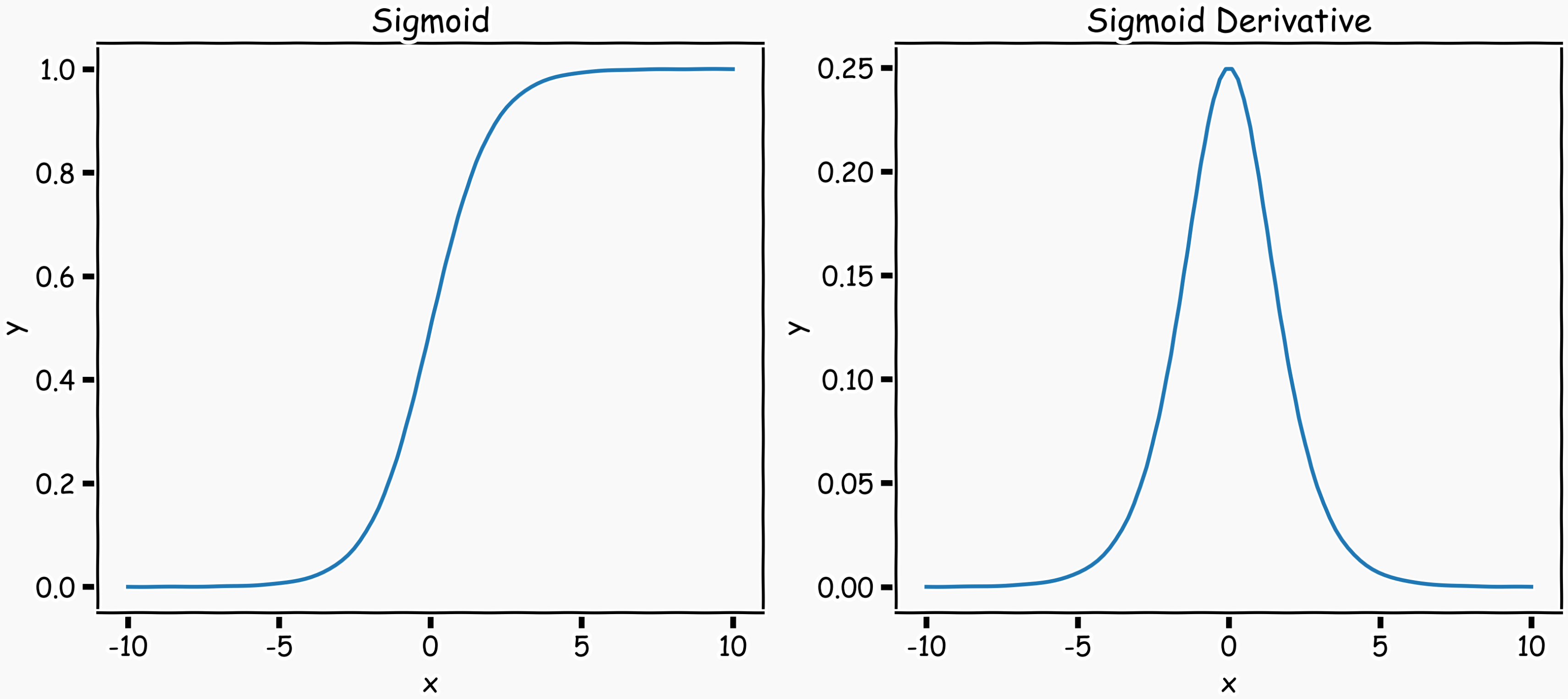
- Provide **non-linearity**
- Ensure gradients remain large through hidden unit

Common choices are

- sigmoid
- tanh
- ReLU, leaky ReLU, Generalized ReLU, MaxOut
- softplus
- swish

Sigmoid (aka Logistic)

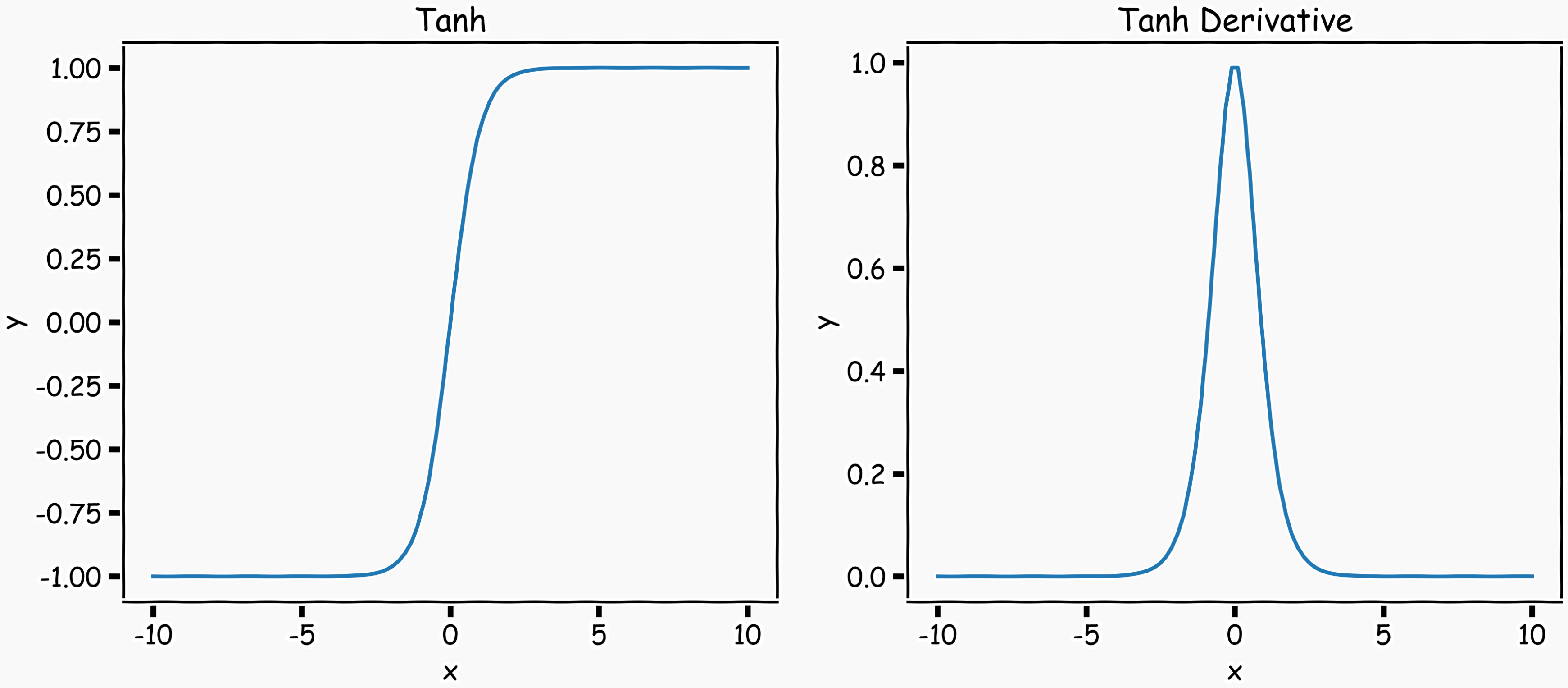
$$y = \frac{1}{1+e^{-x}}$$



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

Hyperbolic Tangent (Tanh)

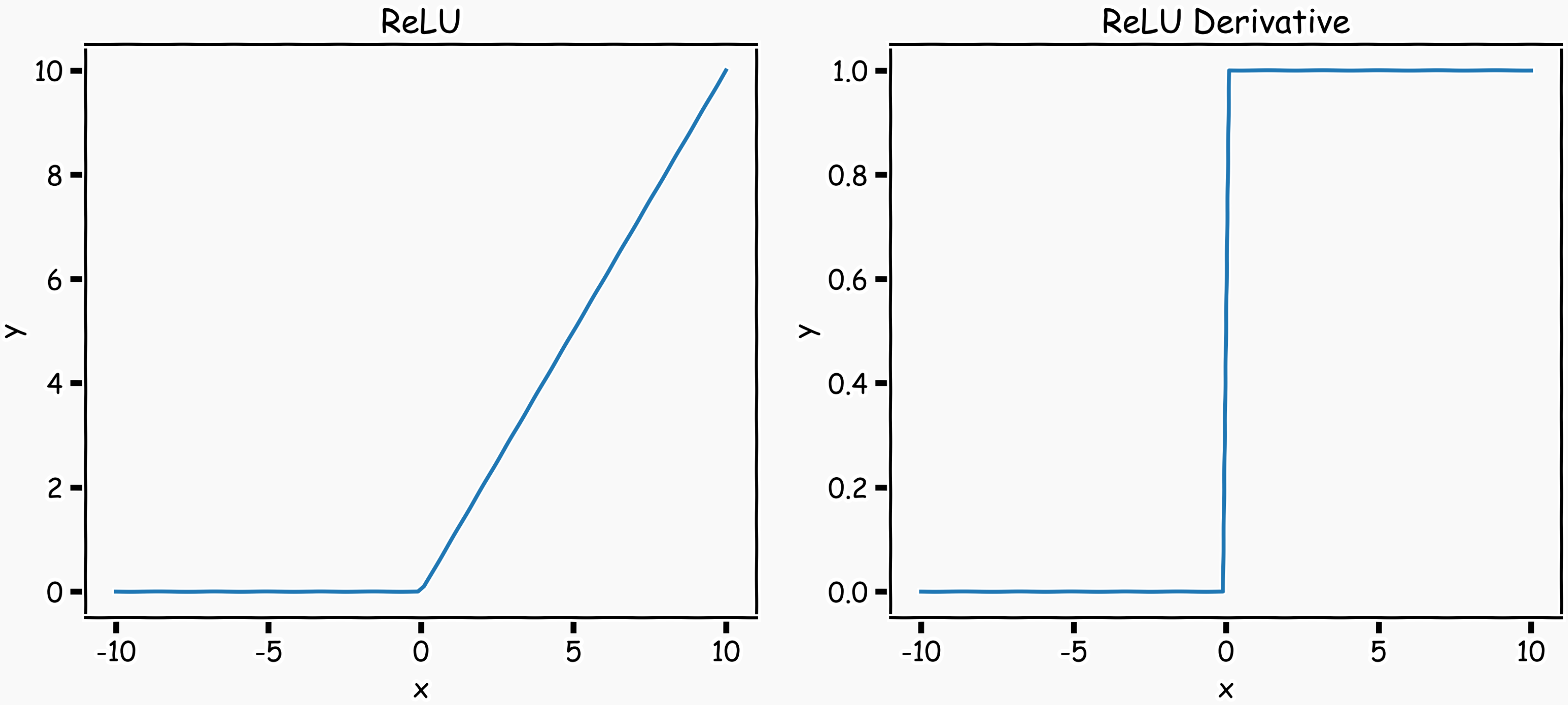
$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Same problem of “vanishing gradients” as sigmoid.

Rectified Linear Unit (ReLU)

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



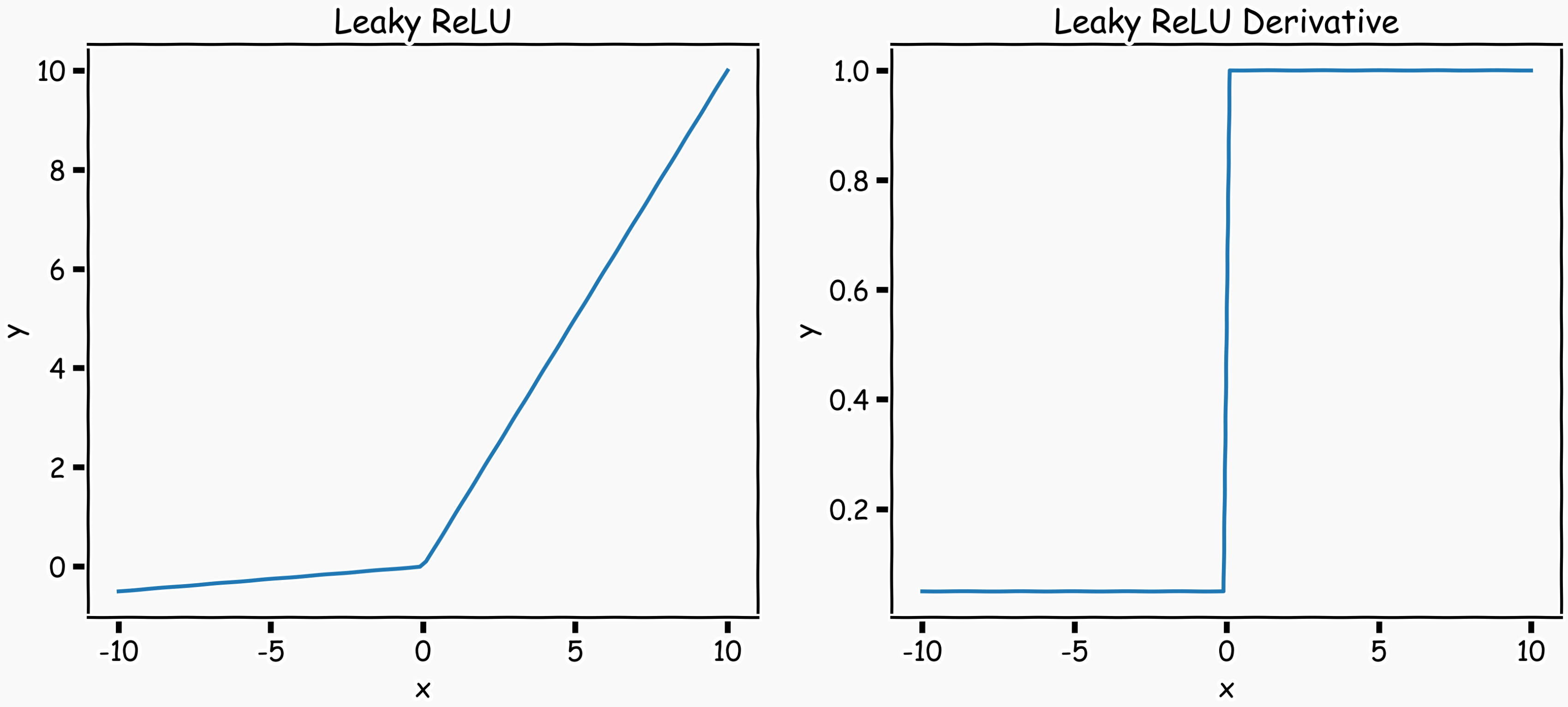
Two major advantages:

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

Leaky ReLU

$$y = \max(0, x) + \alpha \min(0,1)$$

where α takes a small value

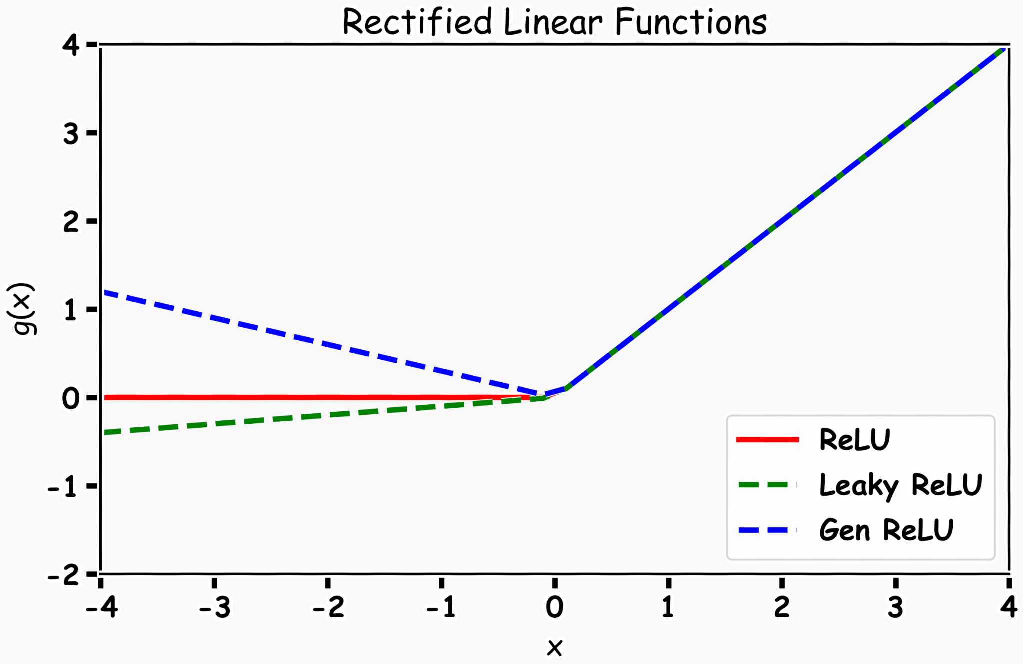


- Tries to fix “dying ReLU” problem: derivative is non-zero everywhere.
- Some people report success with this form of activation function, but the results are not always consistent

Generalized ReLU

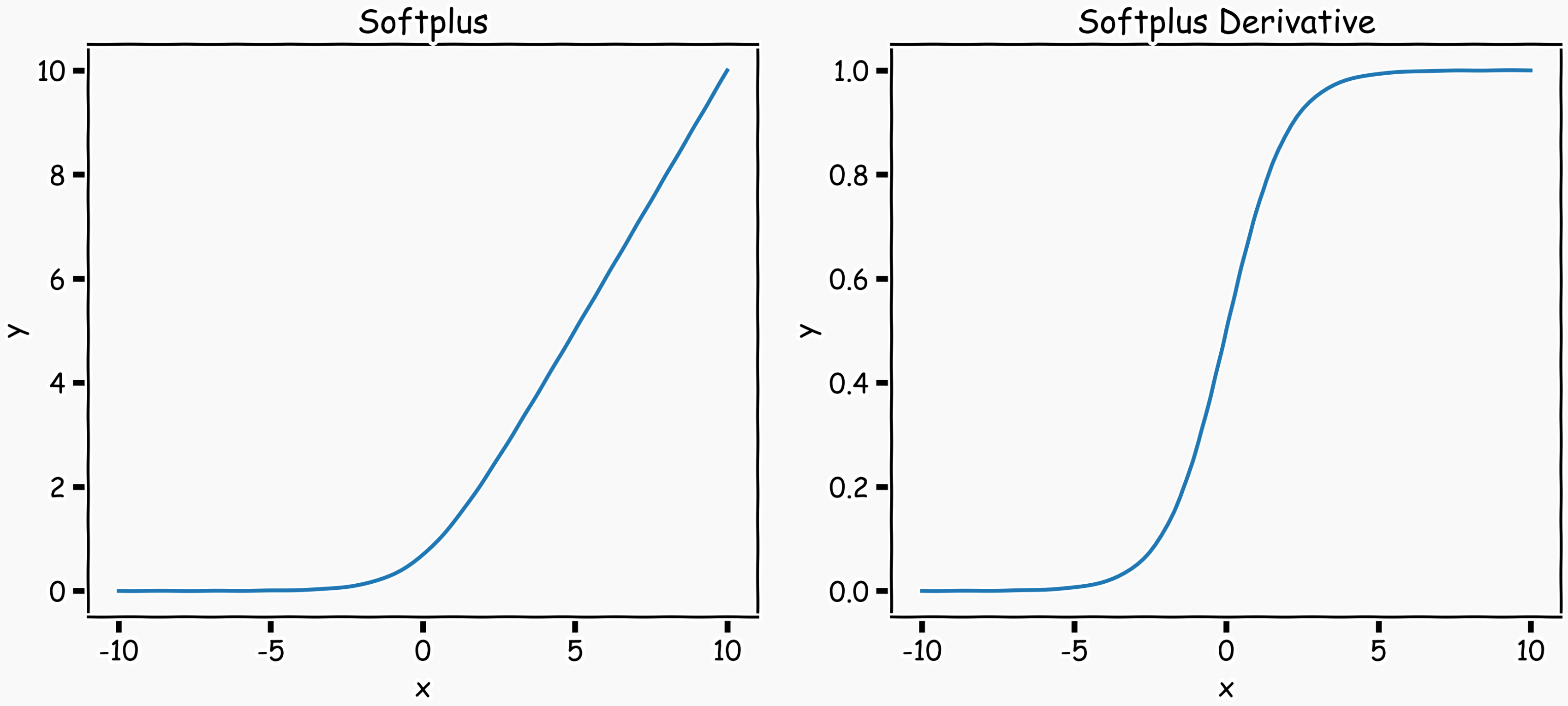
Generalization: For $\alpha_i > 0$

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



softplus

$$y = \log(1 + e^{x})$$

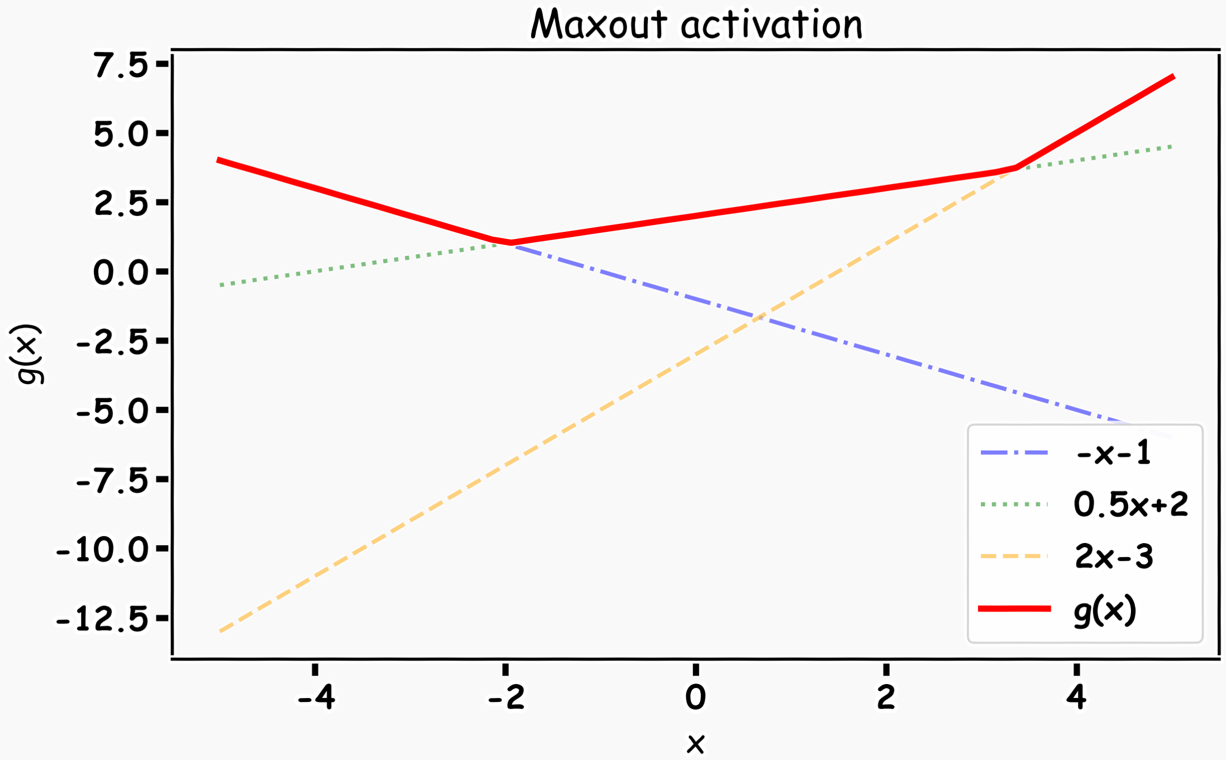


The logistic sigmoid function is a smooth approximation of the derivative of the rectifier

Maxout

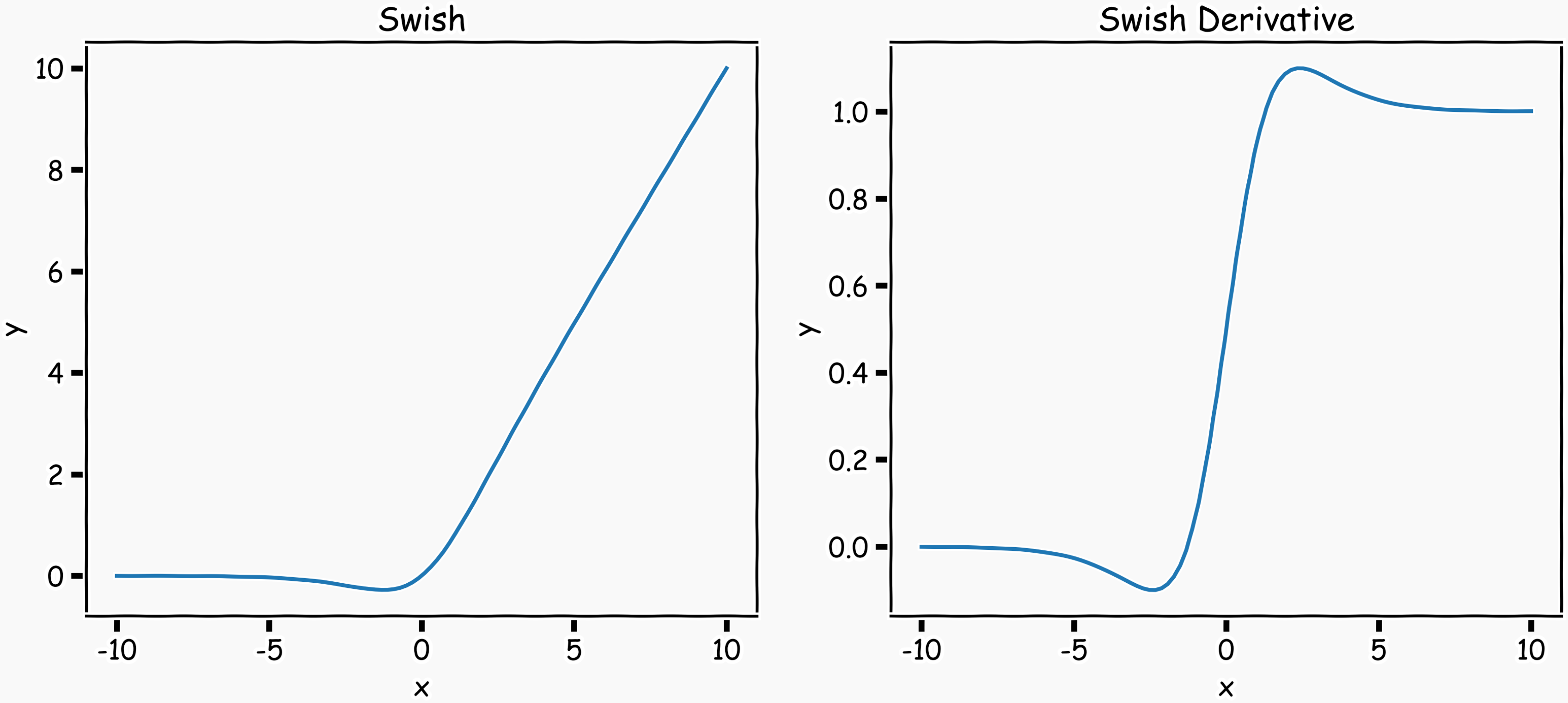
Max of k linear functions. Directly learn the activation function.

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



Swish: A Self-Gated Activation Function

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



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.

Outline

Anatomy of a NN

Design choices

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

Loss Function

Likelihood for a given point:

$$p(y_i|W;x_i)$$

Assume independency, likelihood for all measurements:

$$L(W;X,Y) = p(Y|W;X) = \prod_i p(y_i|W;x_i)$$

Maximize the likelihood, or equivalently maximize the log-likelihood:

$$\log L(W;X,Y) = \sum_i \log p(y_i|W;x_i)$$

Turn this into a loss function:

$$\mathcal{L}(W;X,Y) = -\log L(W;X,Y)$$

Loss Function

Do not need to design separate loss functions if we follow this simple procedure

Examples:

- Distribution is **Normal** then likelihood is:

$$p(y_i|W;x_i) = \frac{1}{\surd\{2\pi^2\sigma\}} e^{-\frac{(y_i-\hat{y}_i)^2}{2\sigma{}^\wedge 2}}$$

MSE

$$\mathcal{L}(W;X,Y) = \sum_i (y_i - \hat{y}_i)^2$$

- Distribution is **Bernouli** then likelihood is:

$$p(y_i|W;x_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

Cross-Entropy

$$\mathcal{L}(W;X,Y) = -\sum_i [y_i \log p_i + (1-y_i)\log(1-p_i)]$$

Design Choices

- Activation function
- Loss function
- Output units
- Architecture
- Optimizer

Output Units

Output Type	Output Distribution	Output layer	Loss Function
Binary			

Output Units

Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli		

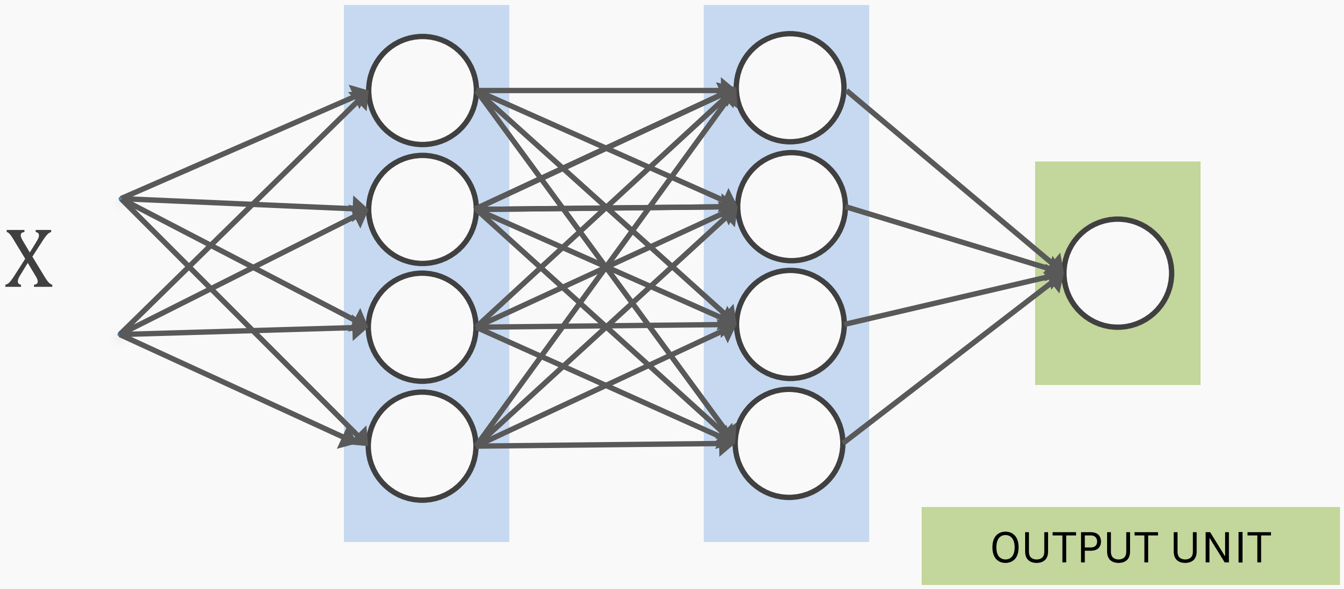
Output Units

Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli		Binary Cross Entropy

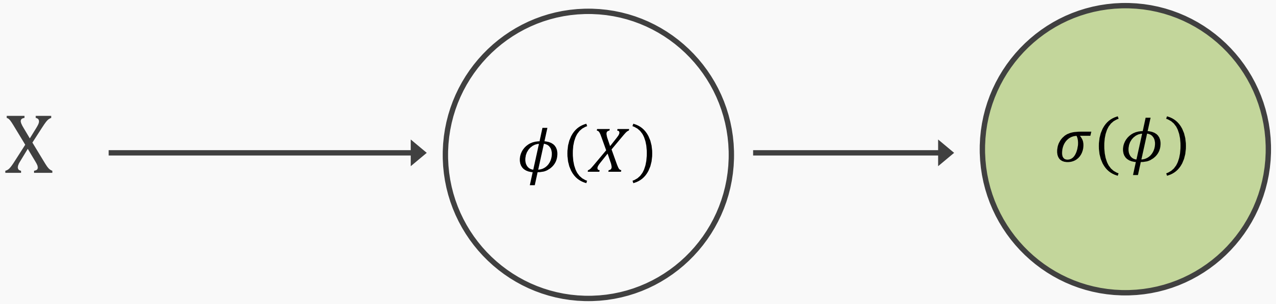
Output Units

Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli	?	Binary Cross Entropy

Output unit for binary classification



$\hat{Y} = \mathrm{P}(y = 1)$



$\hat{Y} = \mathrm{P}(y = 1)$

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

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			

Output Units

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

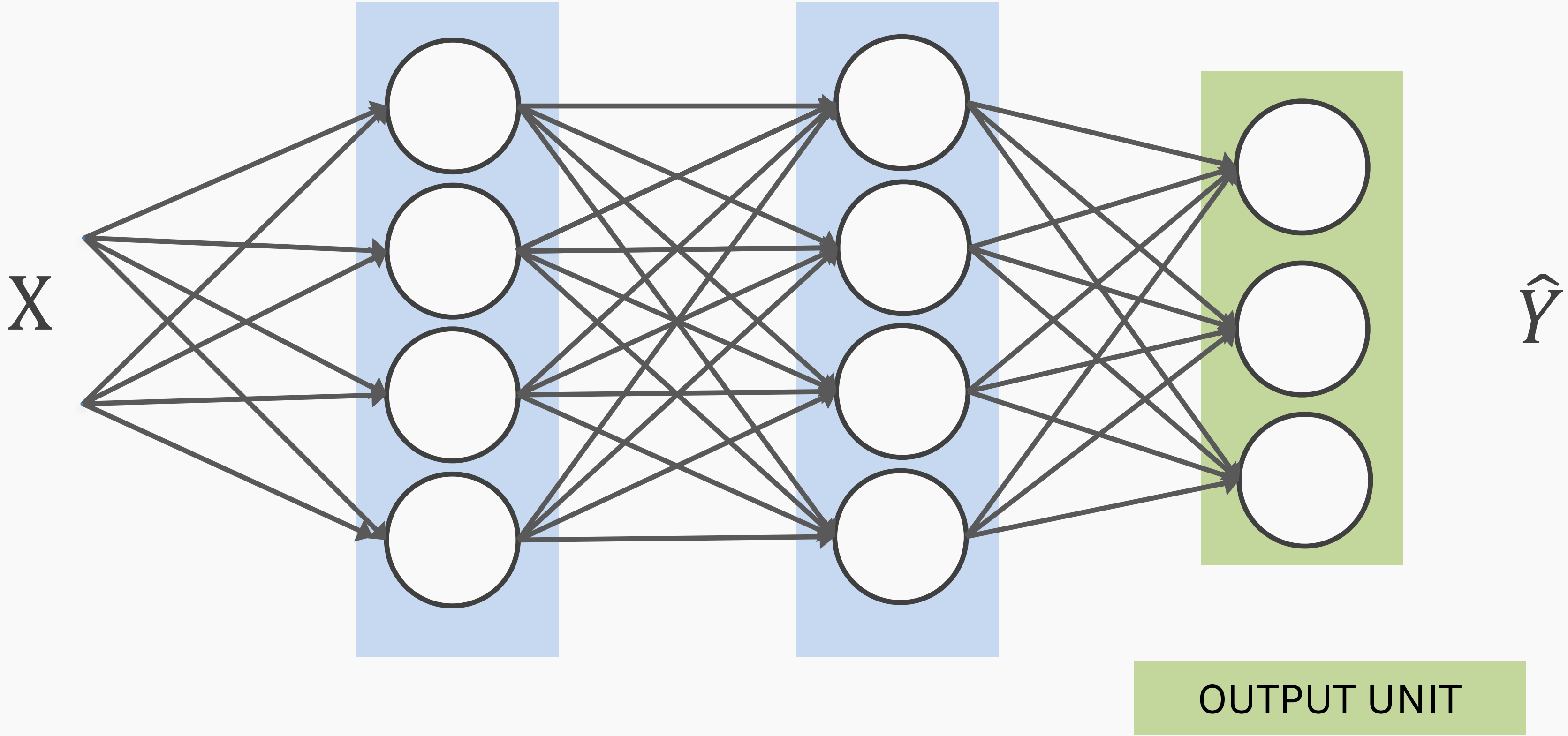
Output Units

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

Output Units

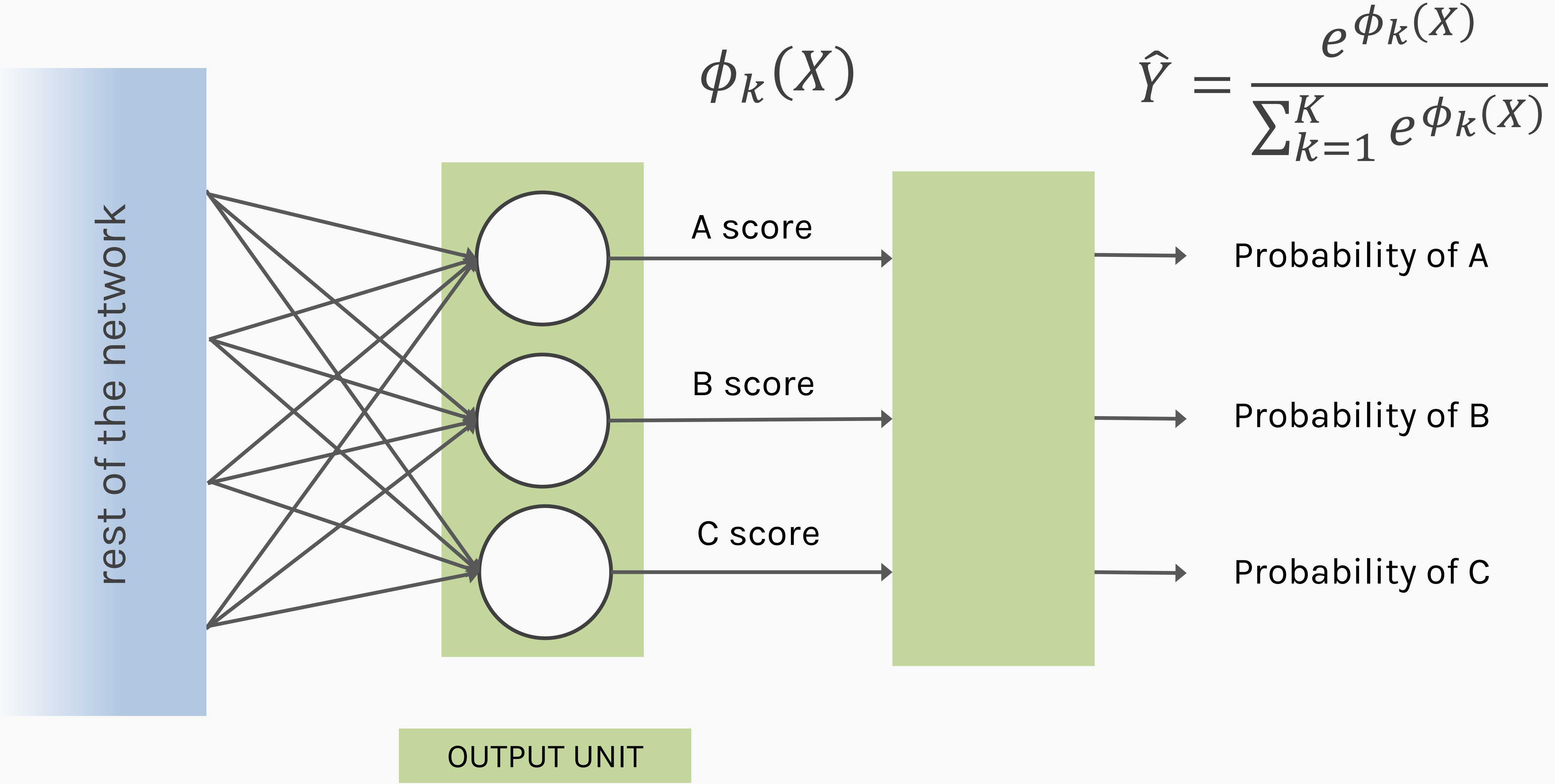
Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinouli	?	Cross Entropy

Output unit for multi-class classification



$\hat{Y} = [P_1, P_2, P_3]$

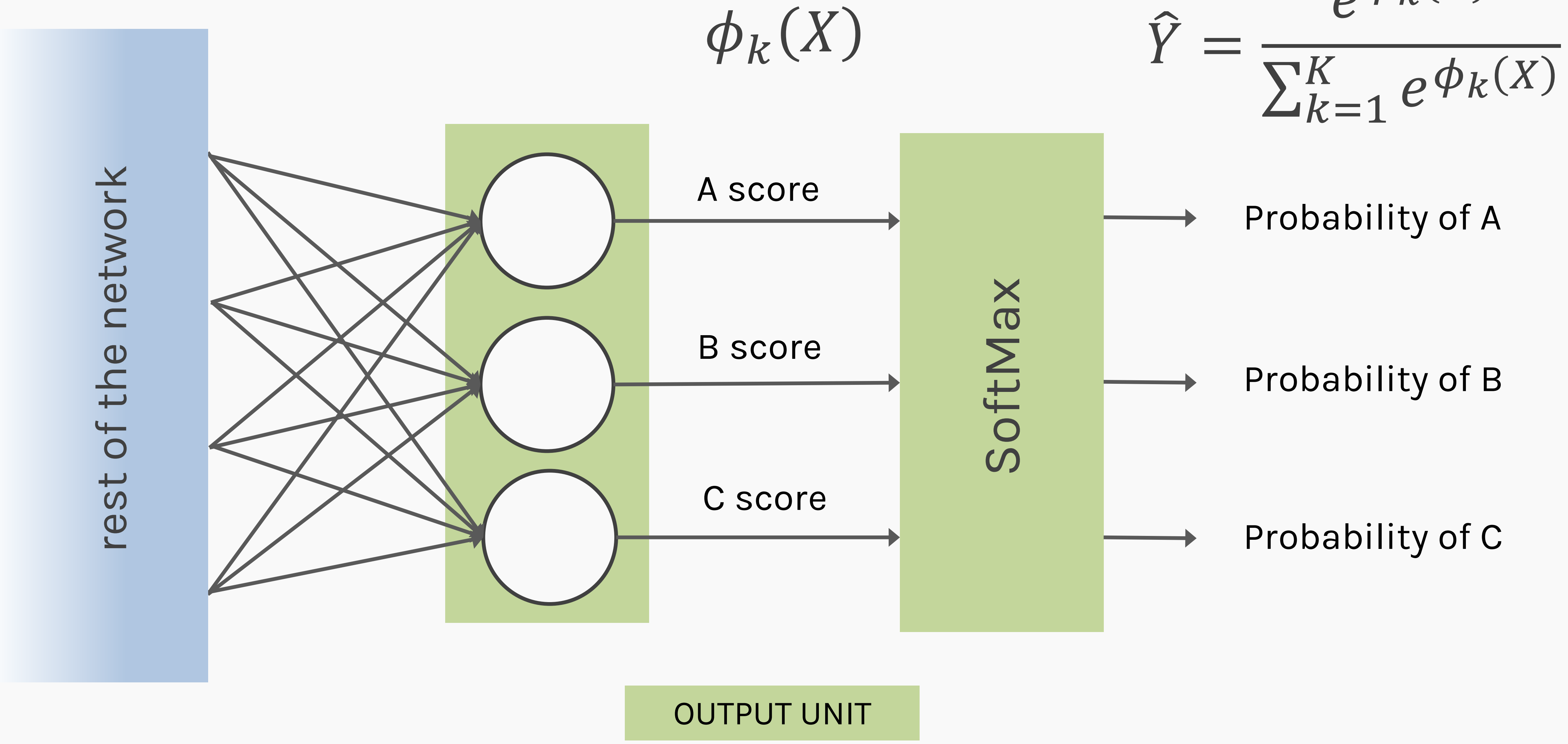
SoftMax



$$\phi_k(X)$$

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

SoftMax

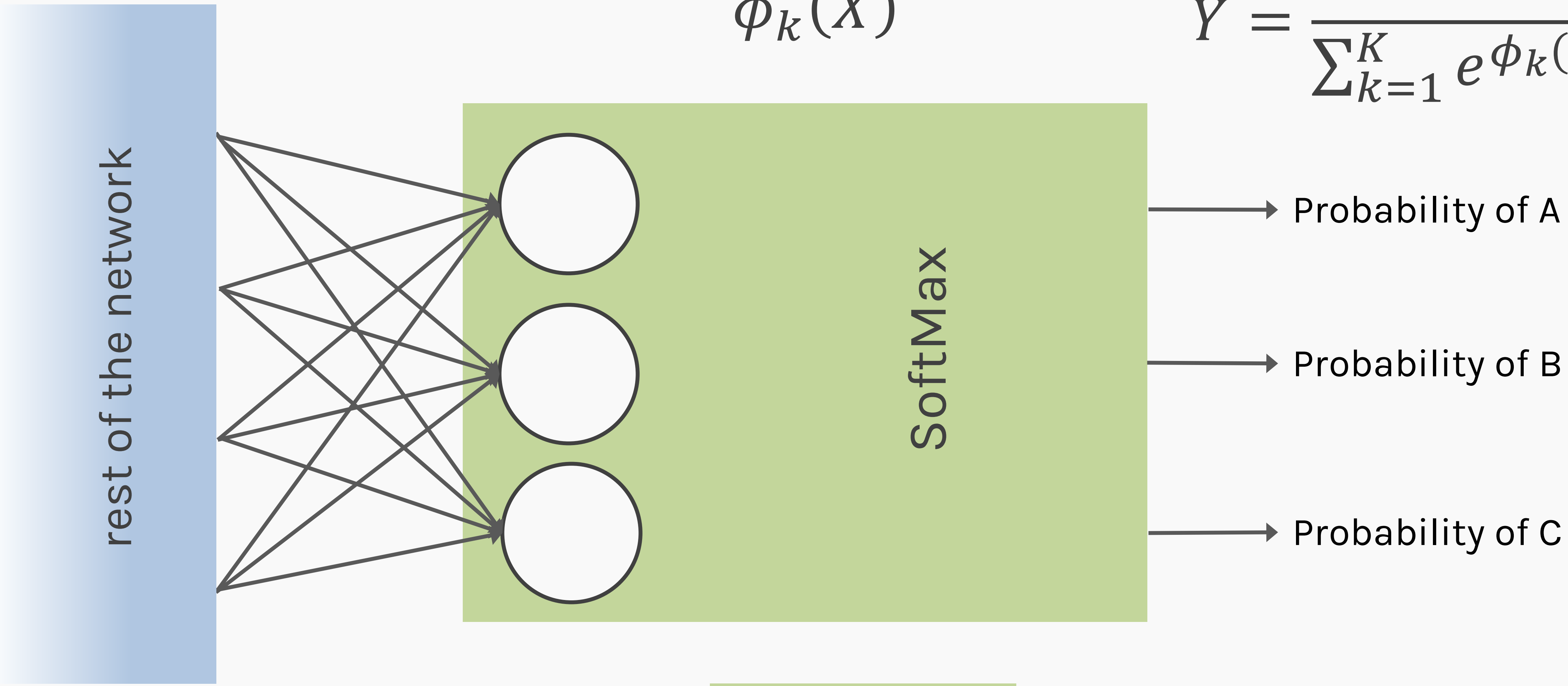


$$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$$

SoftMax

$$\phi_k(X)$$

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



OUTPUT UNIT

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			

Output Units

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

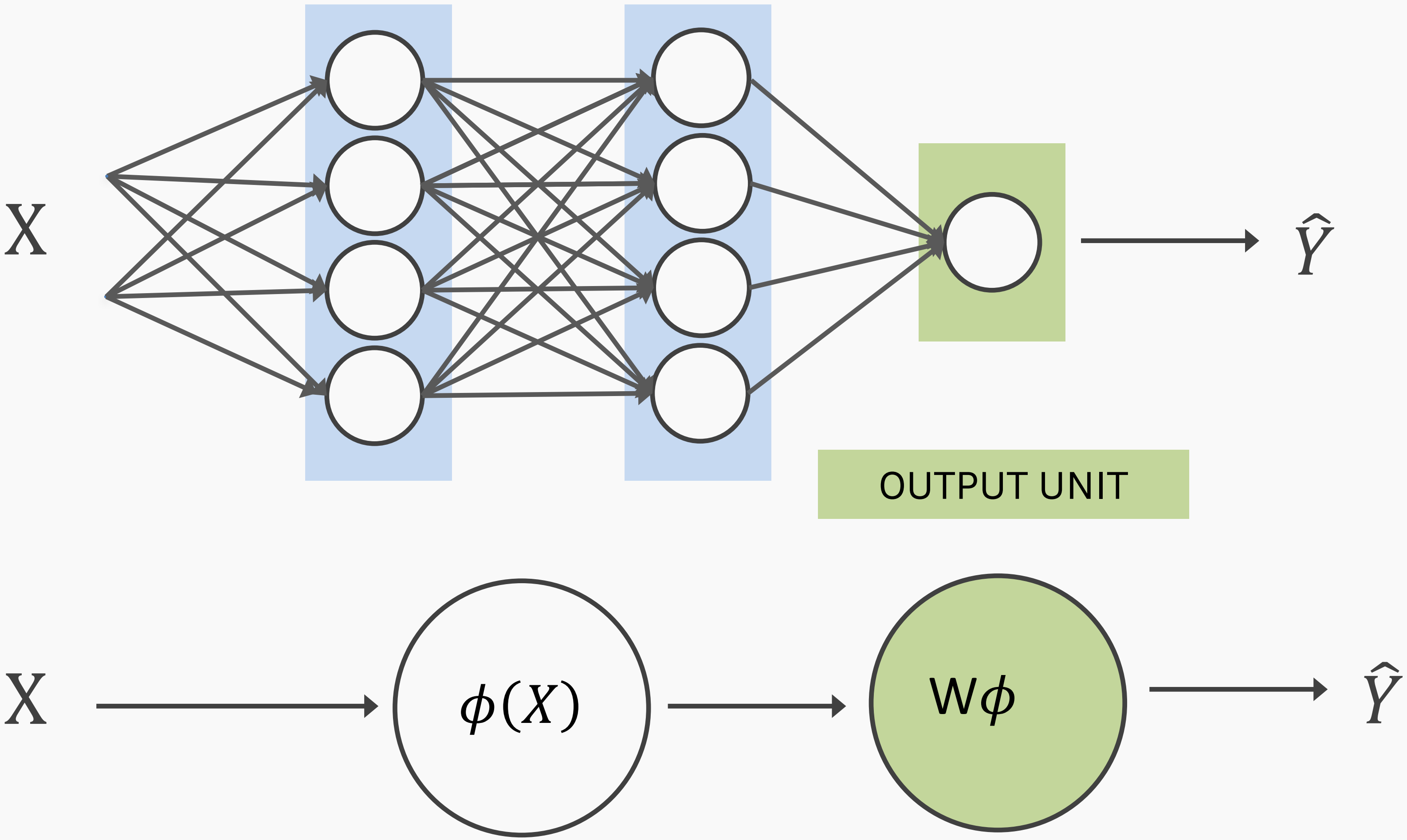
Output Units

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

Output Units

Output Type	**Output Distribution**	**Output layer**	**Cost Function**
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	?	MSE

Output unit for regression



$$X \Longrightarrow \phi(X) \Longrightarrow \hat{Y} = W\phi(X)$$

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	-	

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

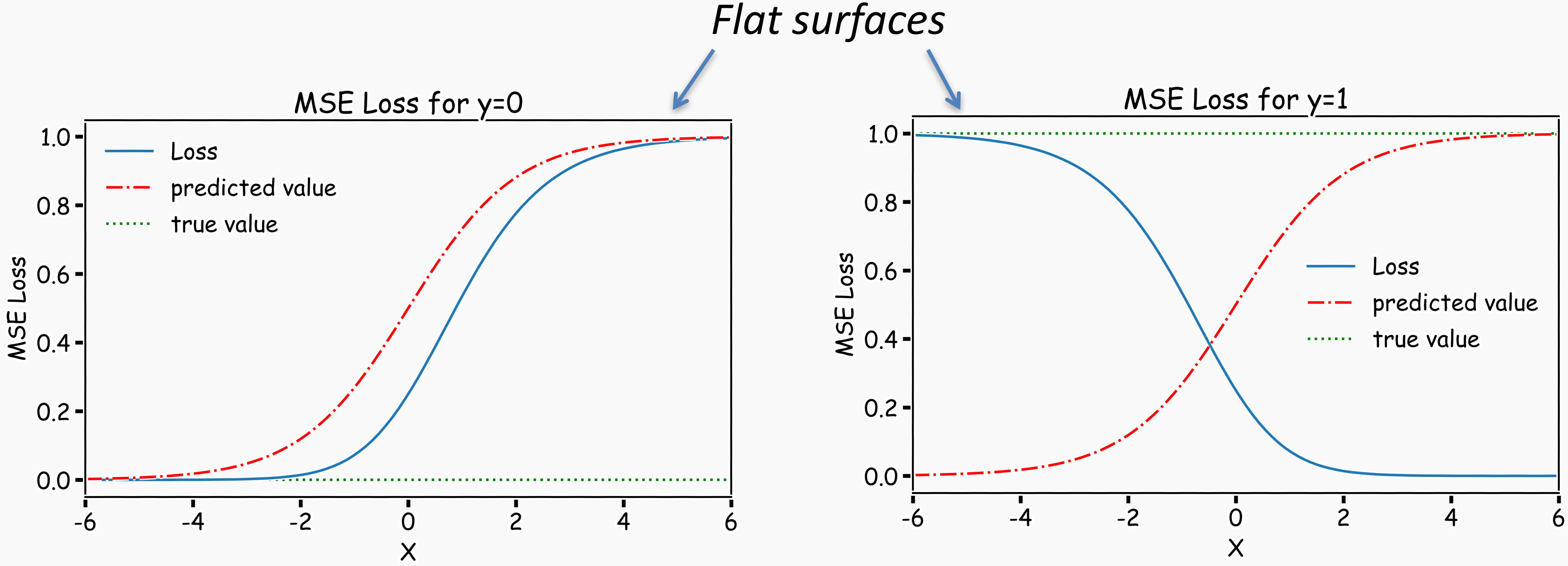
Lectures 18-19 in CS109B

Loss Function

Example: sigmoid output + squared loss

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

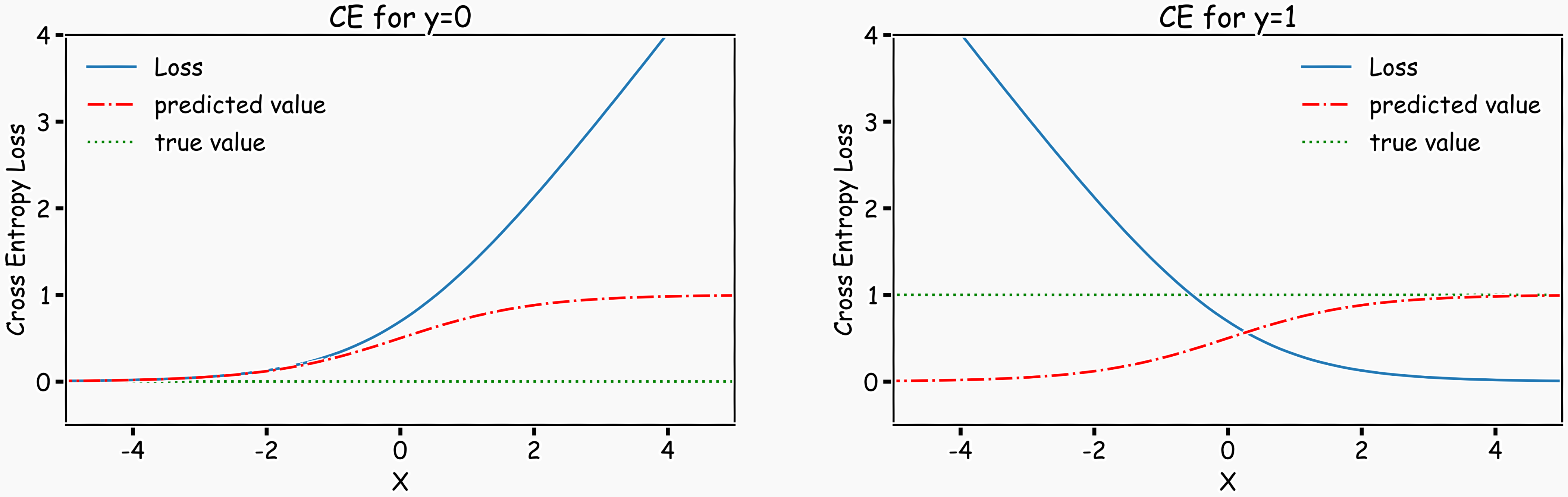
Flat surfaces



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

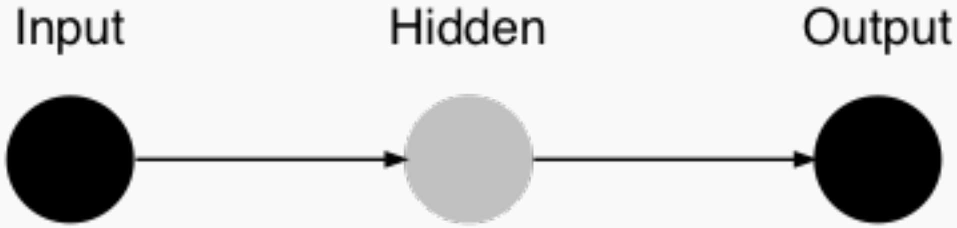
Loss function

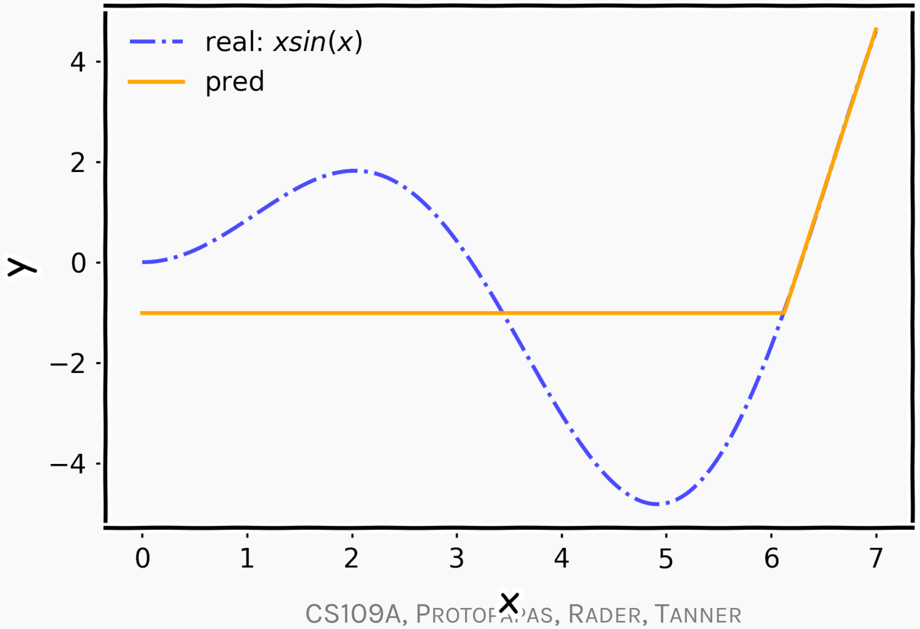
Output units

Architecture

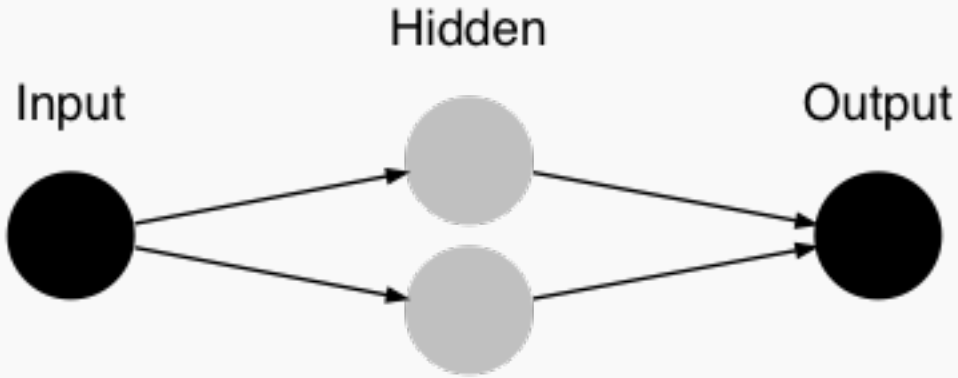
Optimizer

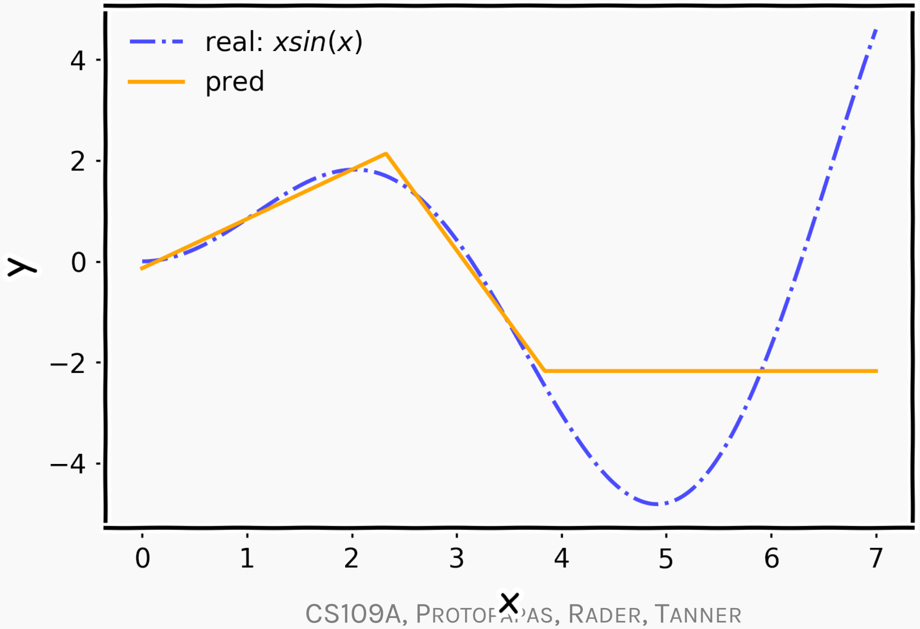
NN in action



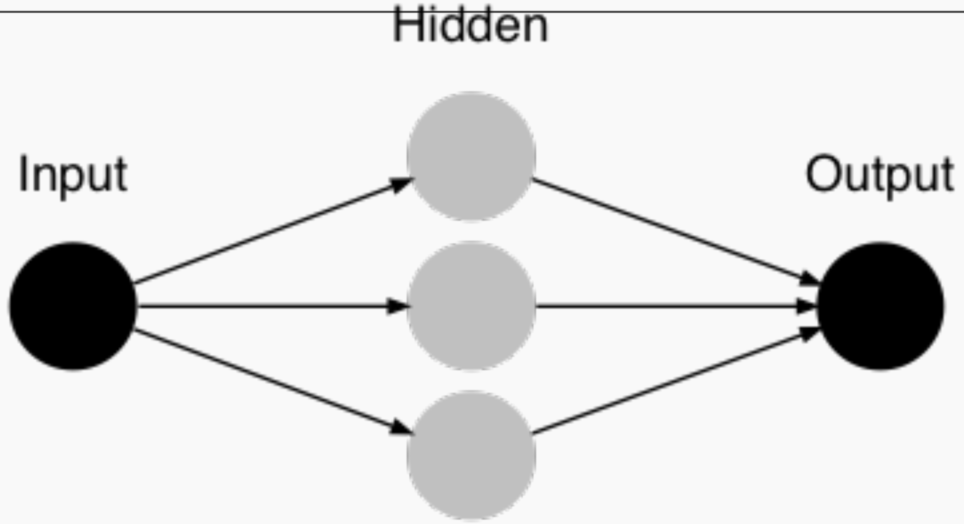


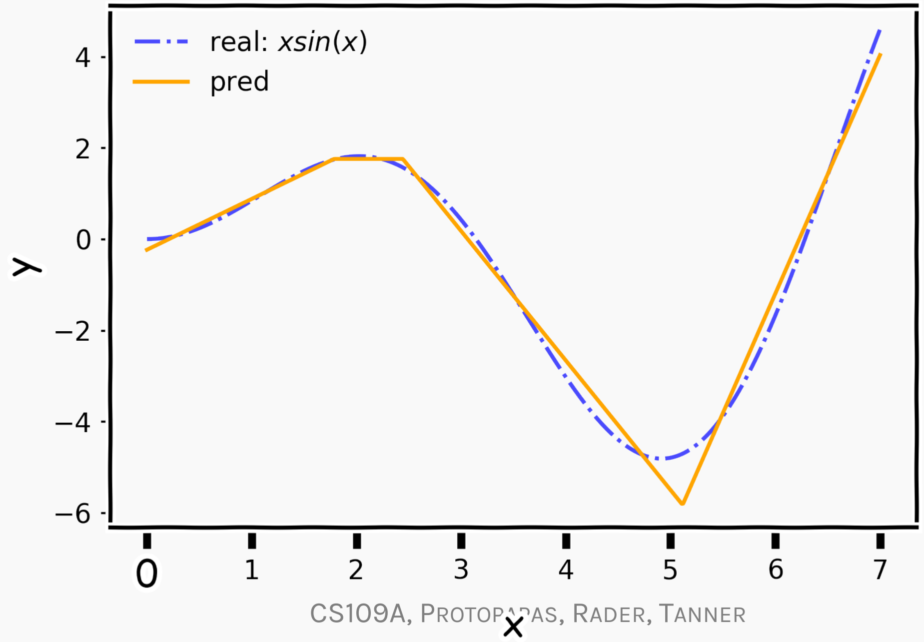
NN in action



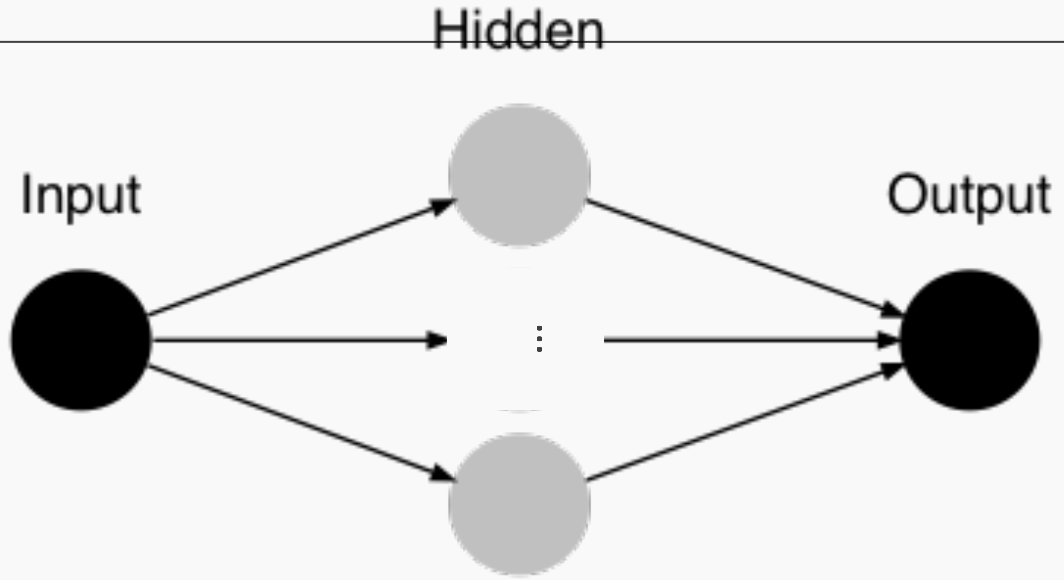


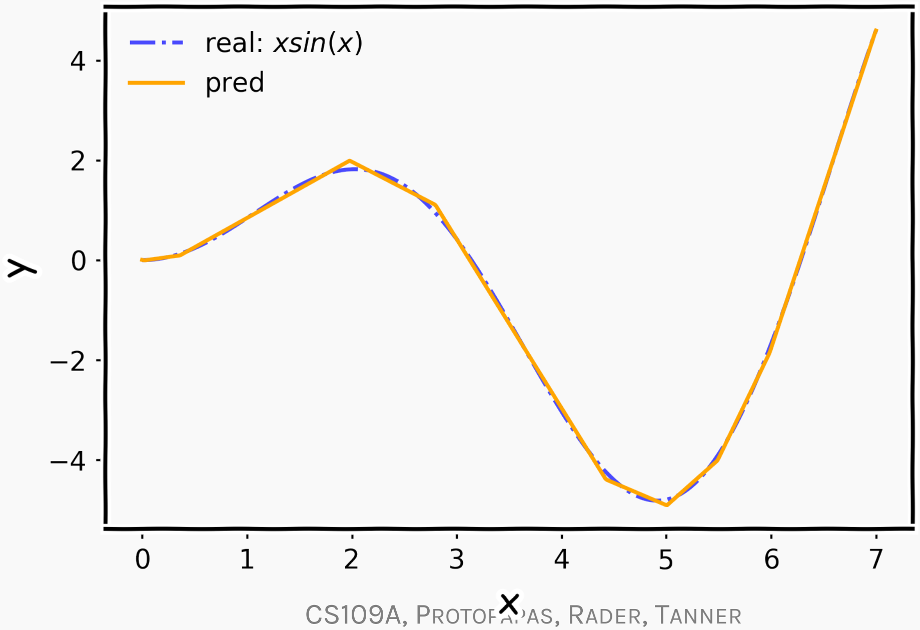
NN in action



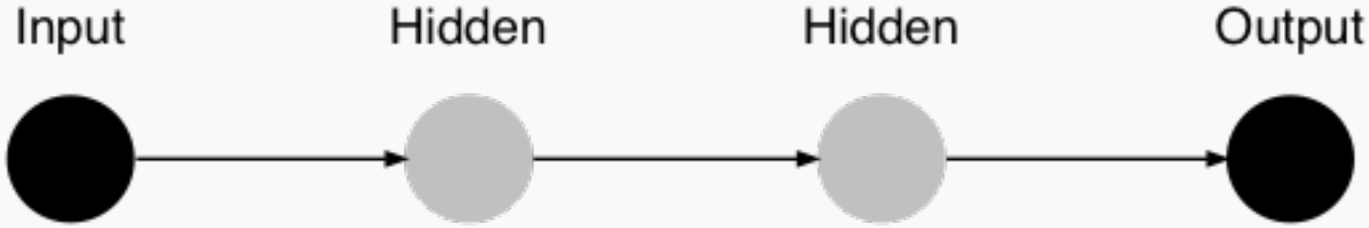


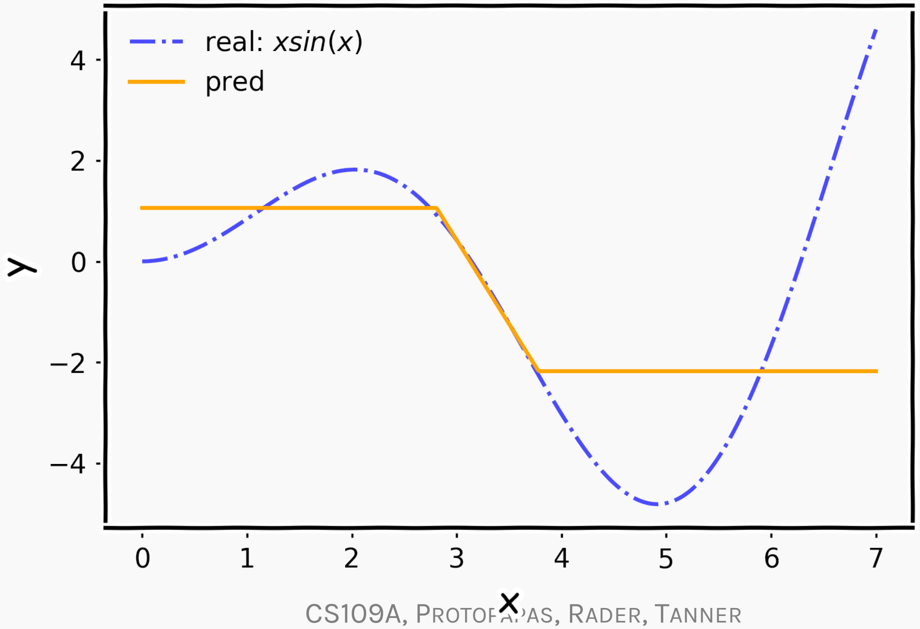
NN in action



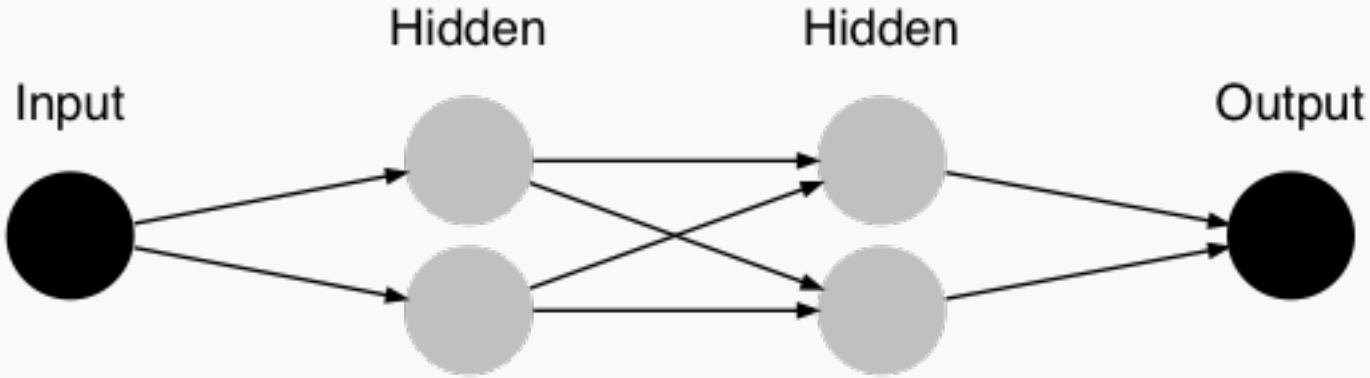


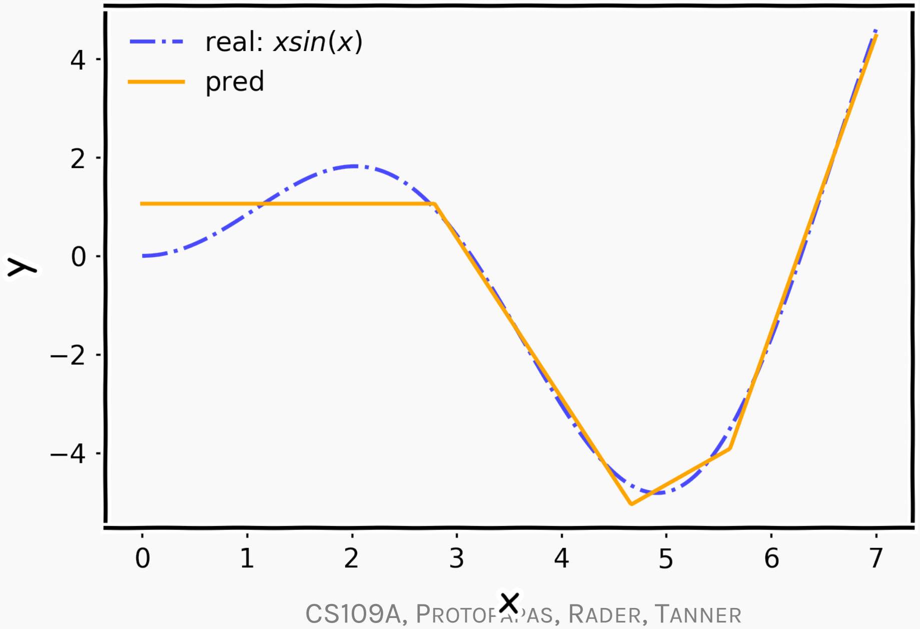
NN in action


Input
Hidden
Hidden
Output

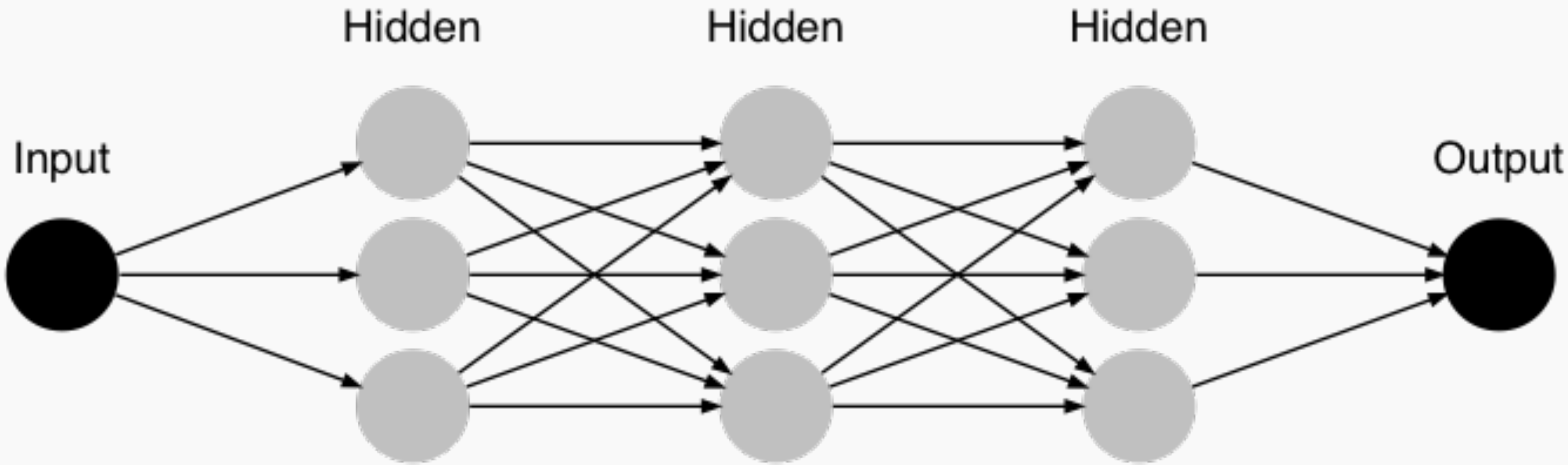

real: $xsin(x)$
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

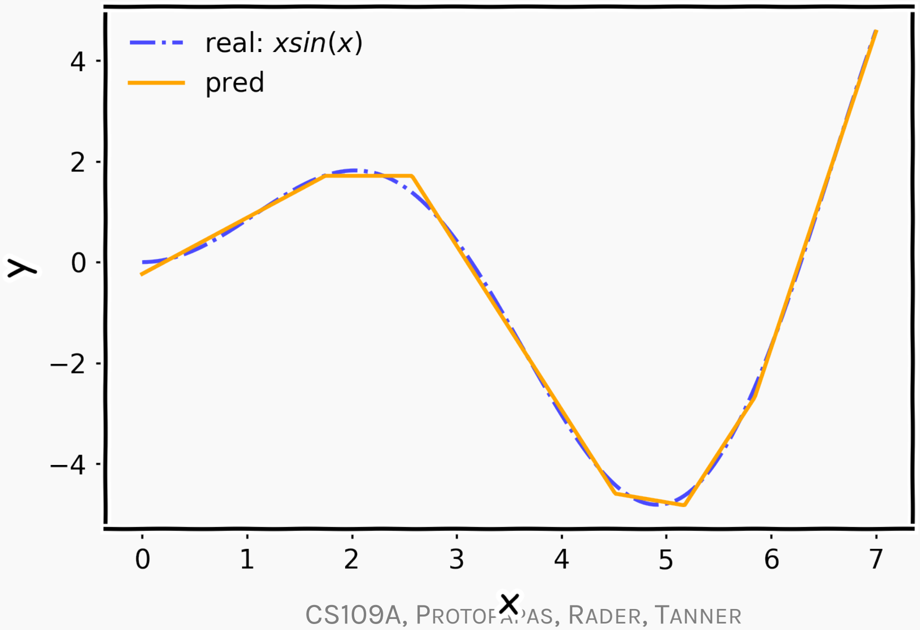
NN in action


Input
Hidden
Hidden
Output


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

NN in action


Hidden
Hidden
Hidden
Input
Output


real: $x sin(x)$
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.

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

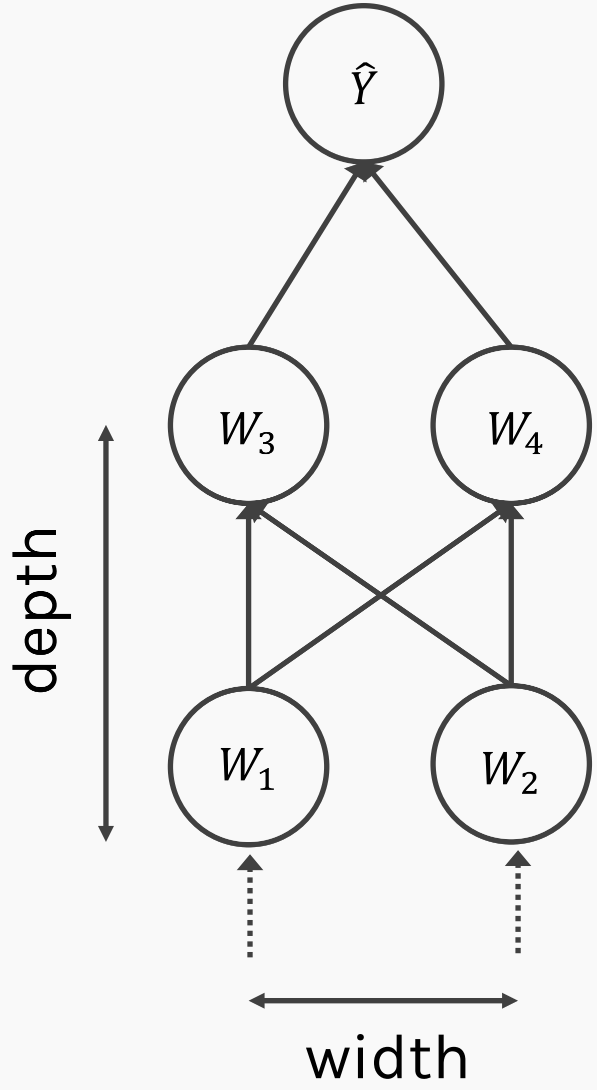
$$Y = \hat{f}(x) + \epsilon$$

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

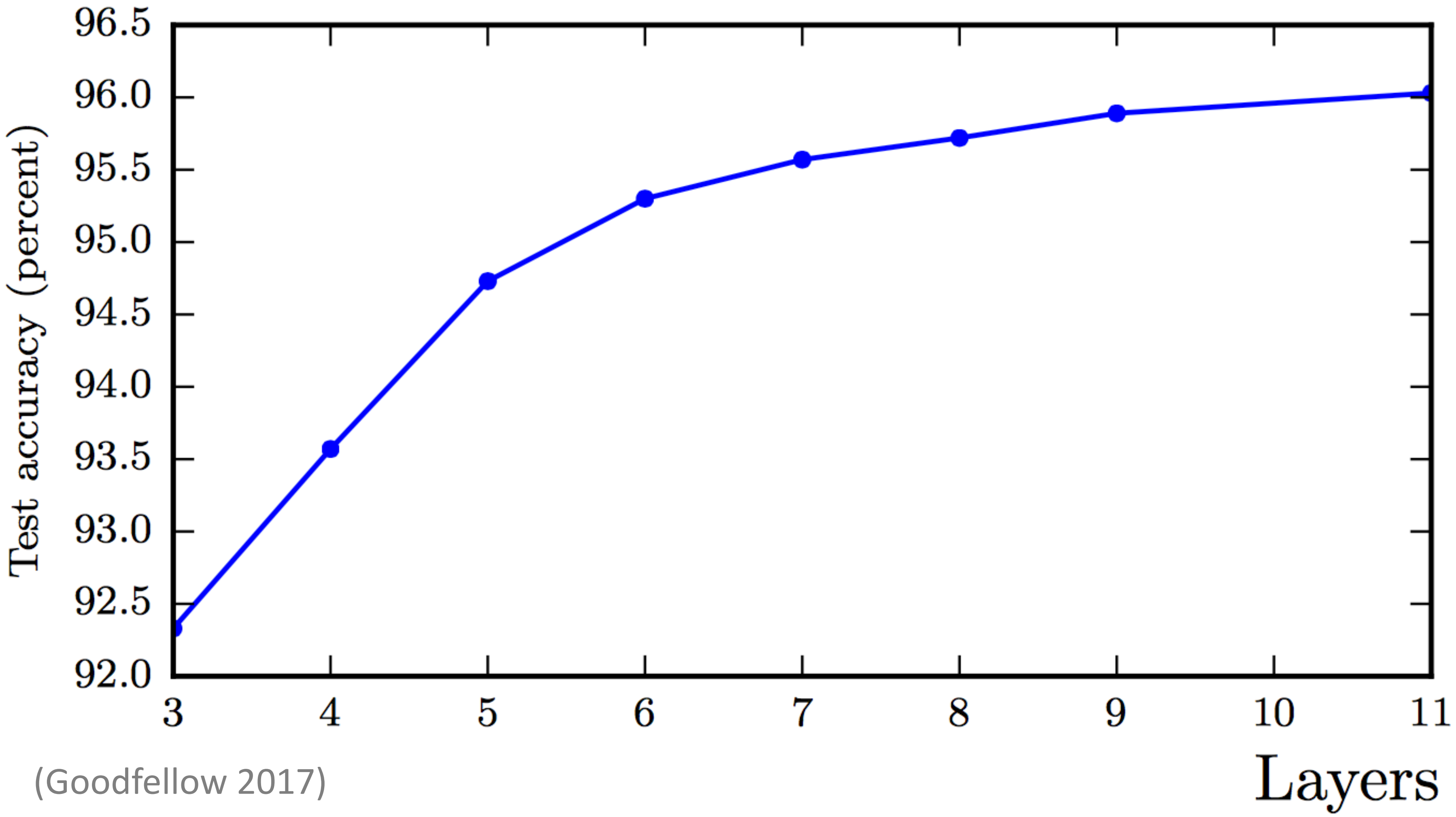
One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy

So why deeper?

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



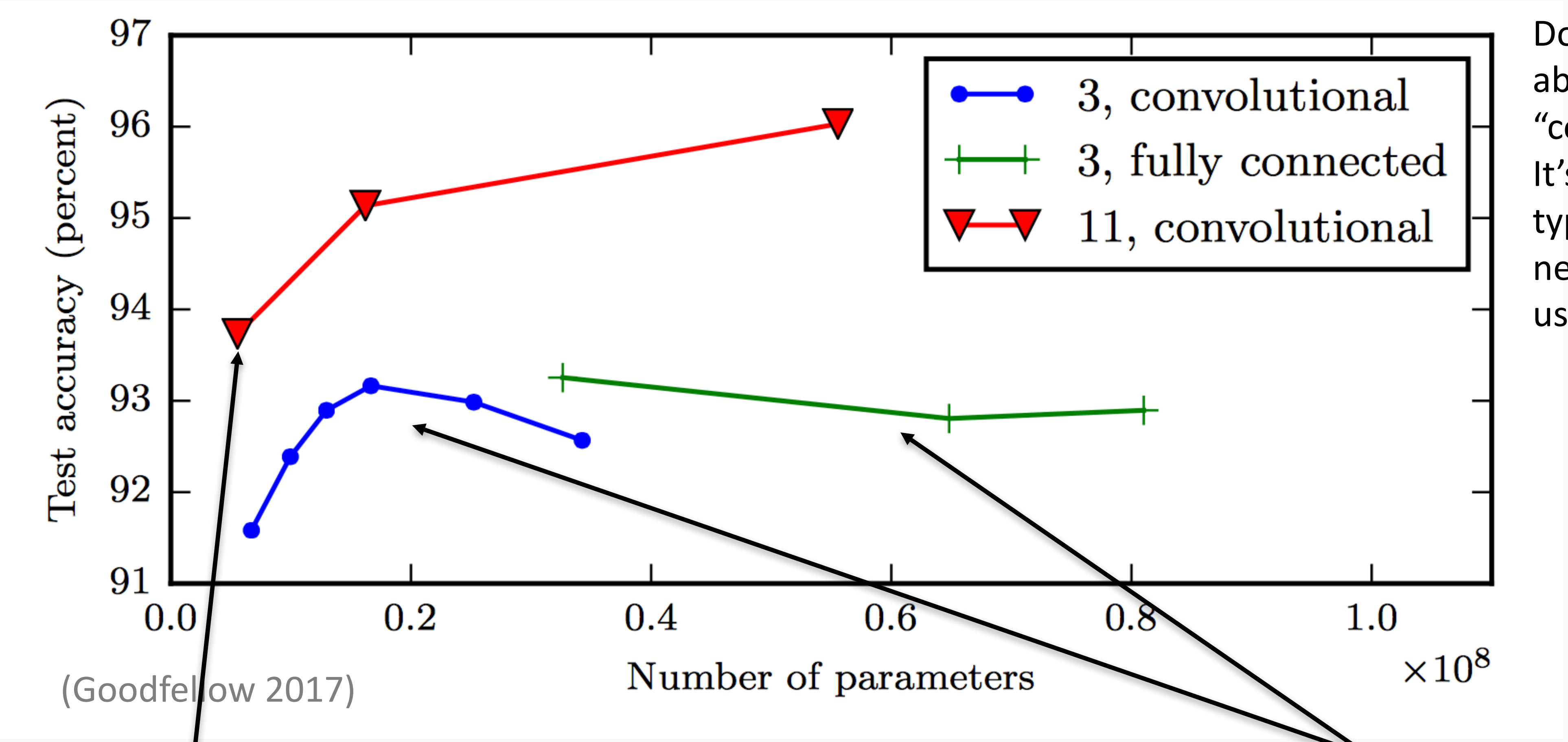
Better Generalization with Depth



(Goodfellow 2017)

Shallow Nets Overfit More

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

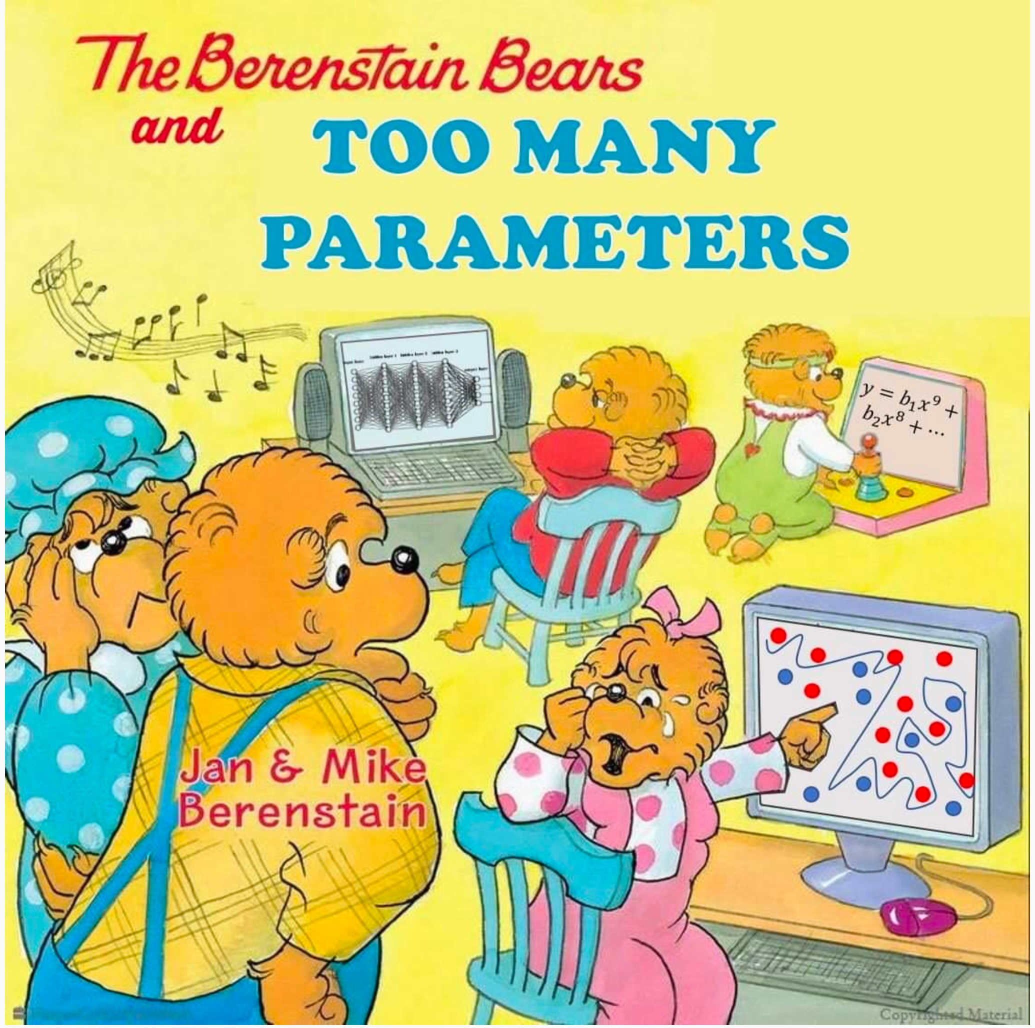


(Goodfellow 2017)

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.


The Berenstain Bears
and TOO MANY PARAMETERS
$y = b_1x^9 + b_2x^8 + \ldots$
Jan & Mike Berenstain
Copyrighted Material

Lab time with Pavlos

1. Install Keras or tensorboard 2
2. Build the same thing we did for exercise from Lecture 18 but now with Keras.