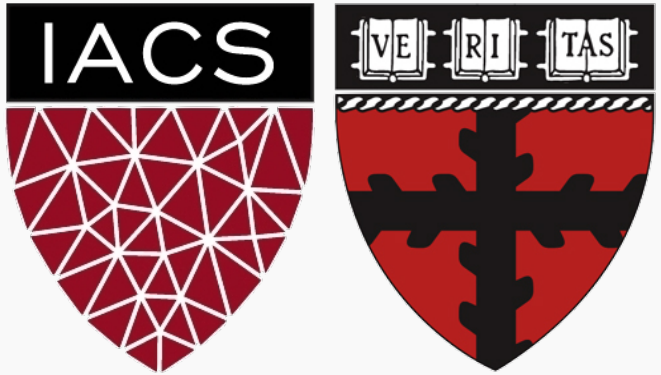
Lecture 19: Anatomy of NN

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

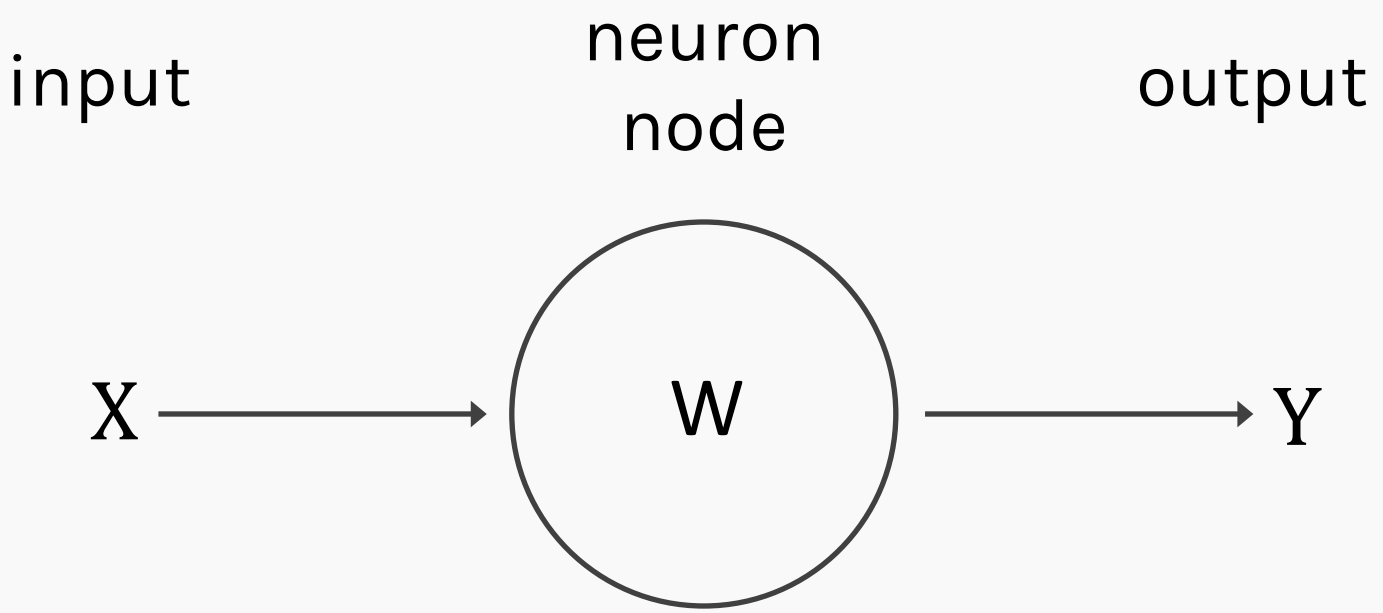
Outline

Anatomy of a NN

Design choices

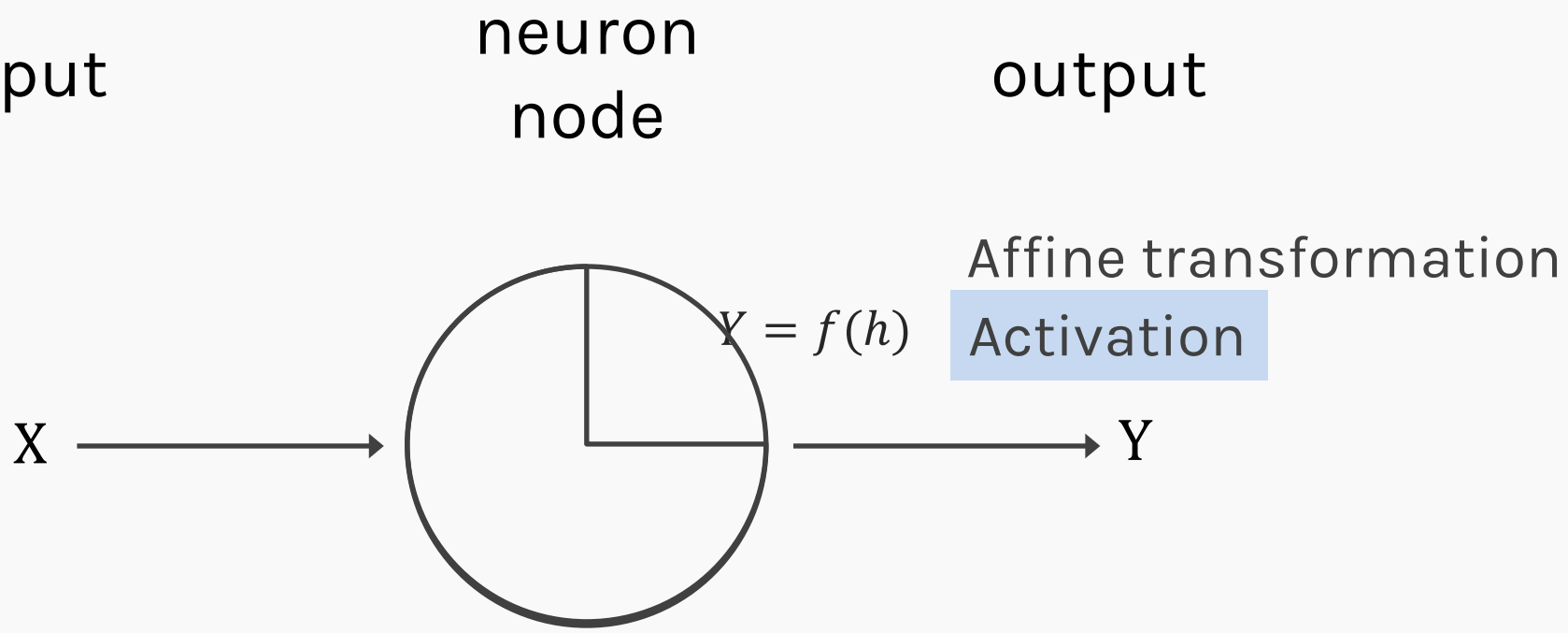
- Activation function
- Loss function
- Output units
- Architecture

Anatomy of artificial neural network (ANN)



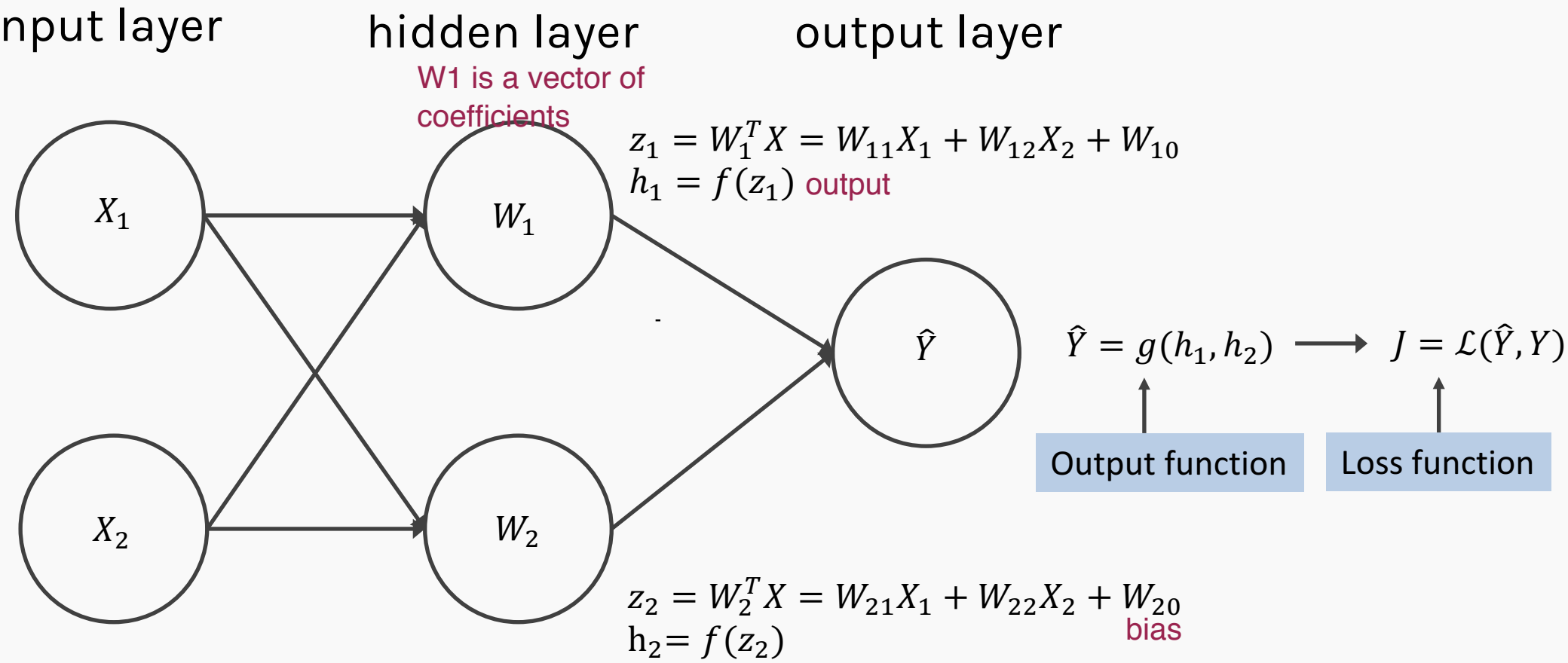
affine transformation: multiply input by coef (called W now, B previously) and add a constant
put affine transformed vars thru activation function

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)

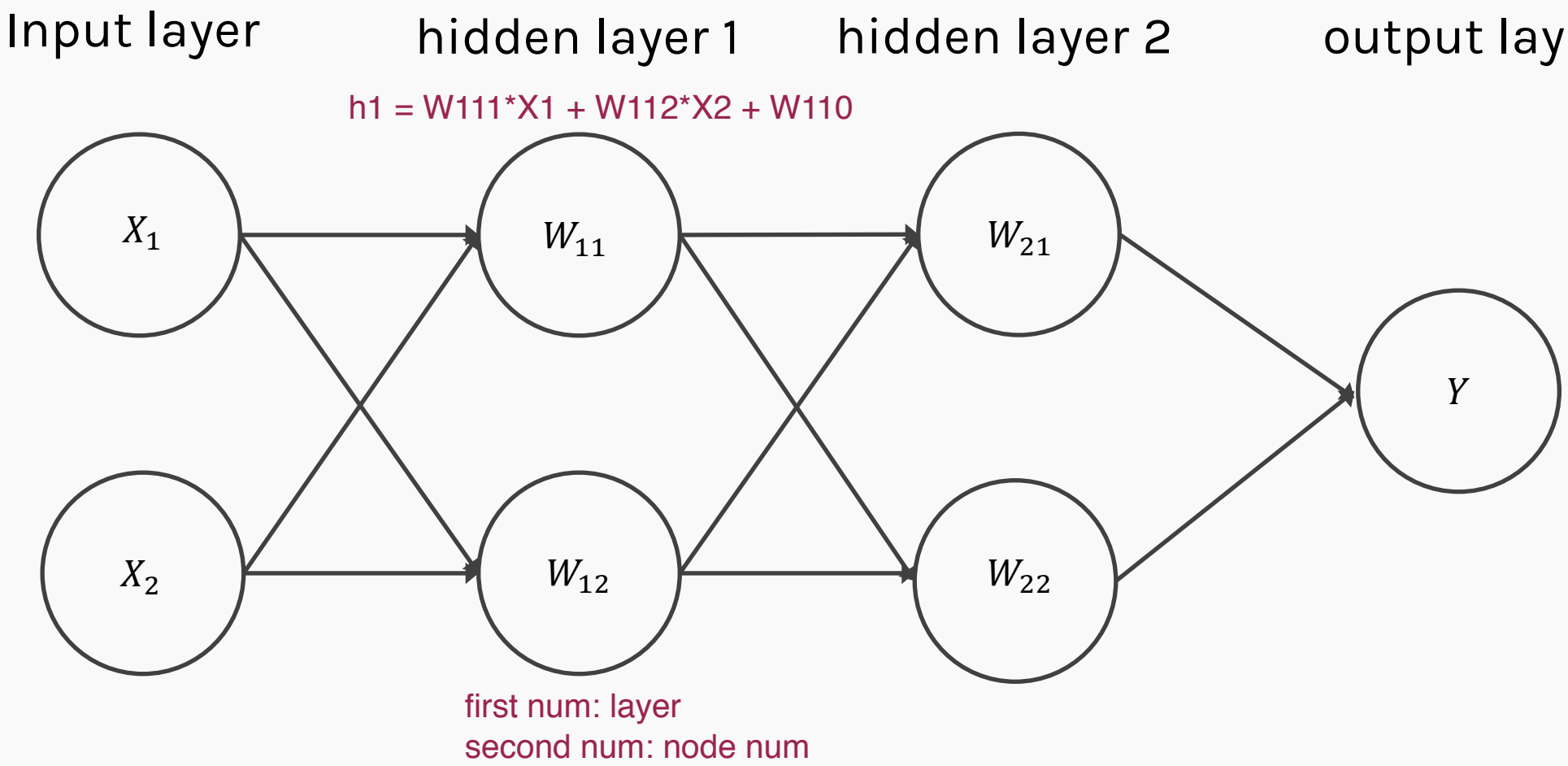


You only have control over the weights (W), the coef by other name once you choose your activation fxn.

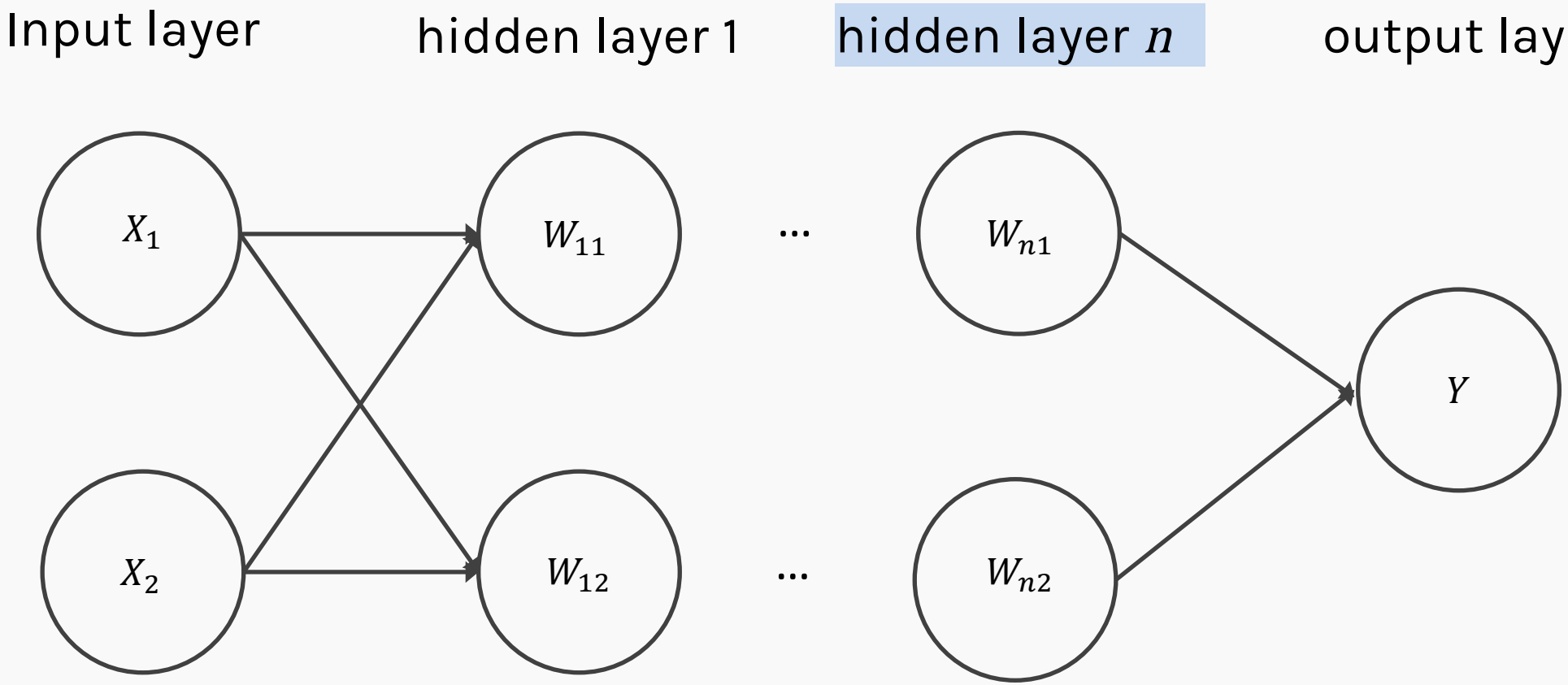
Thats why we call the whole circle W

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.

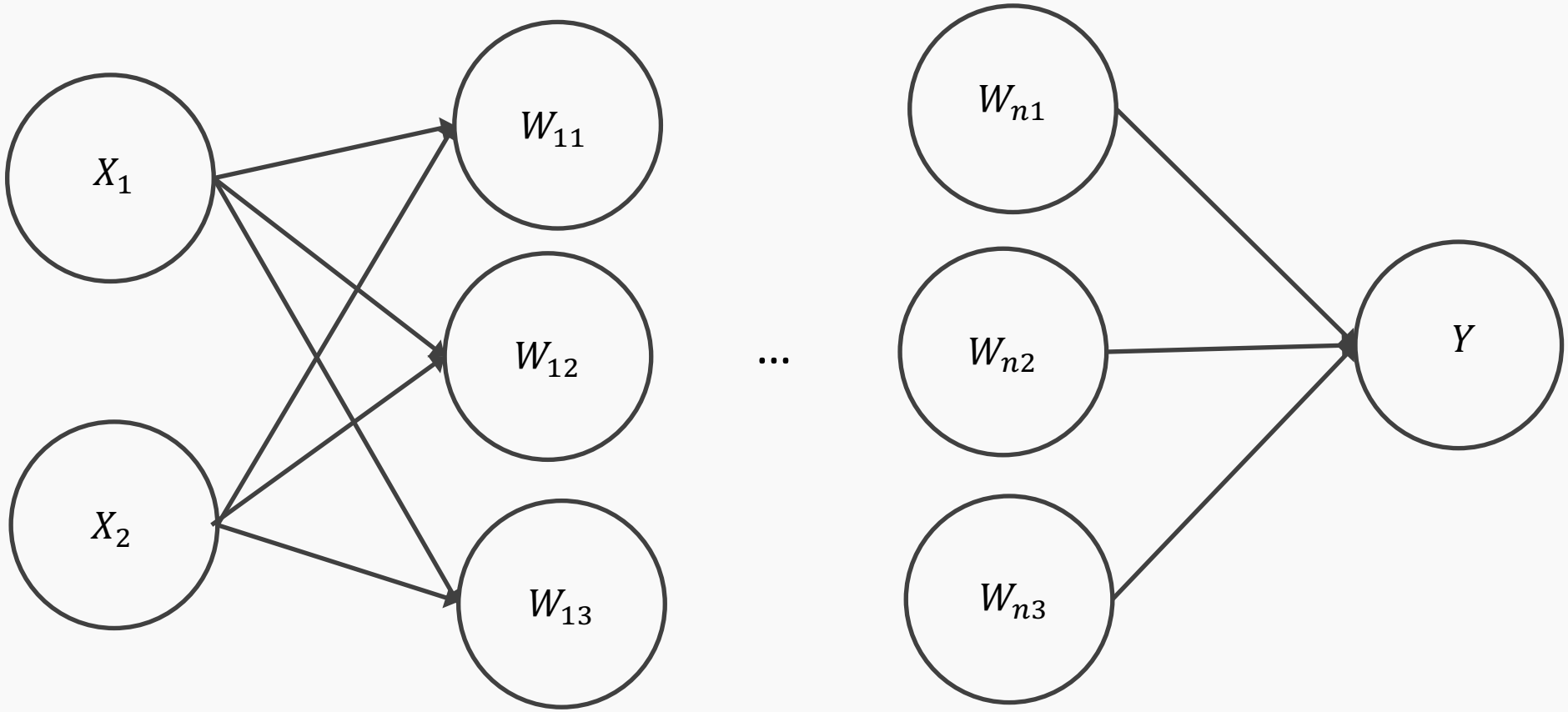
Anatomy of artificial neural network (ANN)

Input layer

hidden layer 1,
3 nodes

hidden layer n
3 nodes

output layer



Anatomy of artificial neural network (ANN)

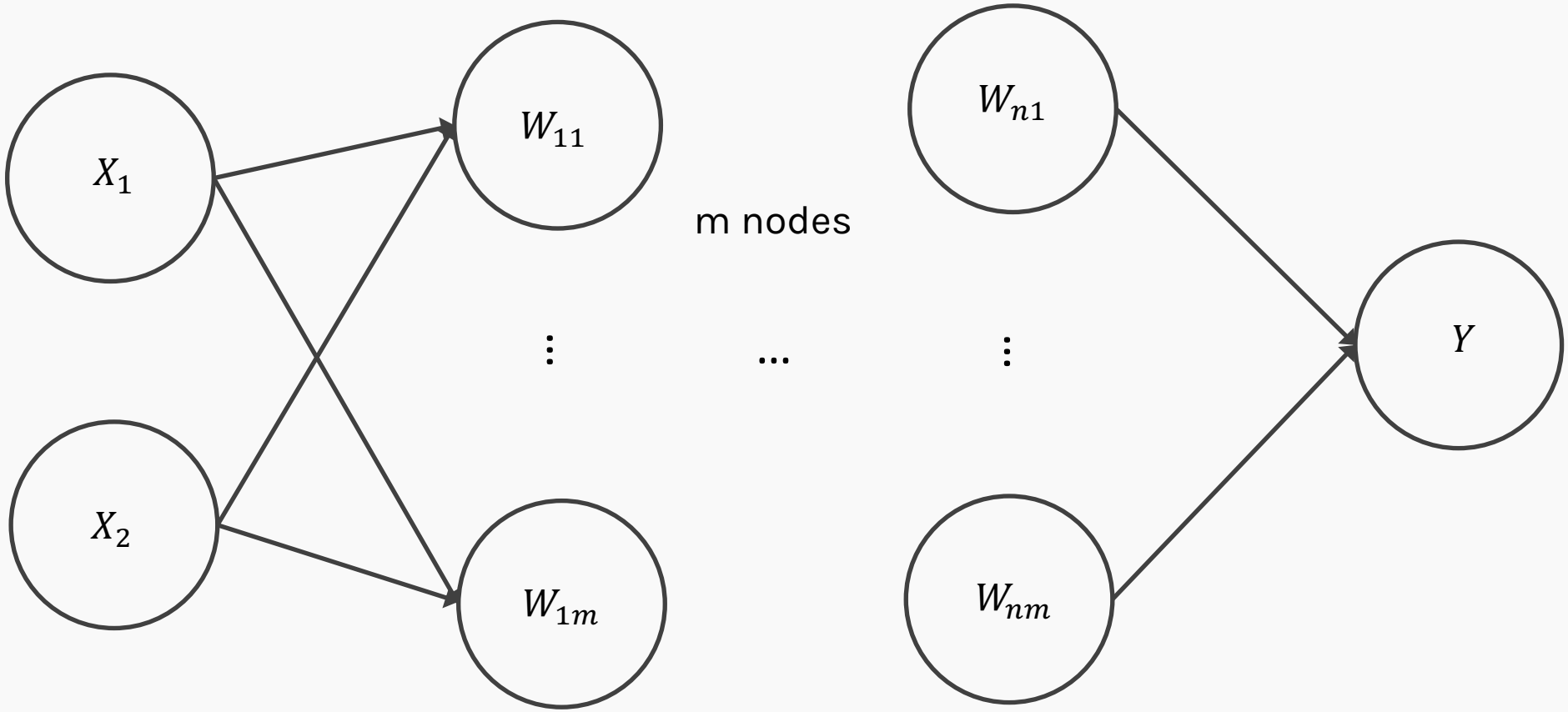
Input layer

hidden layer 1,

m nodes

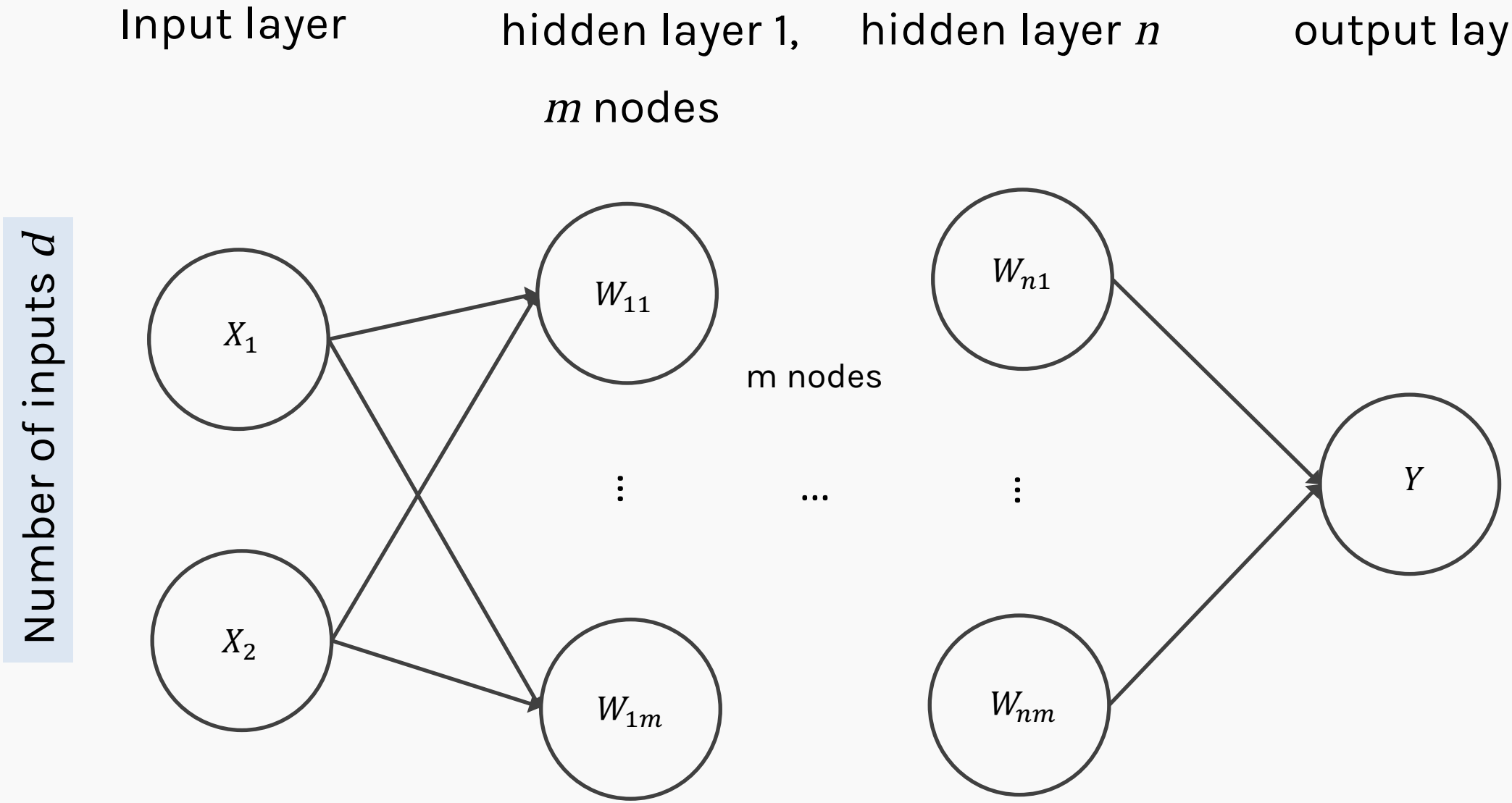
hidden layer n

output layer



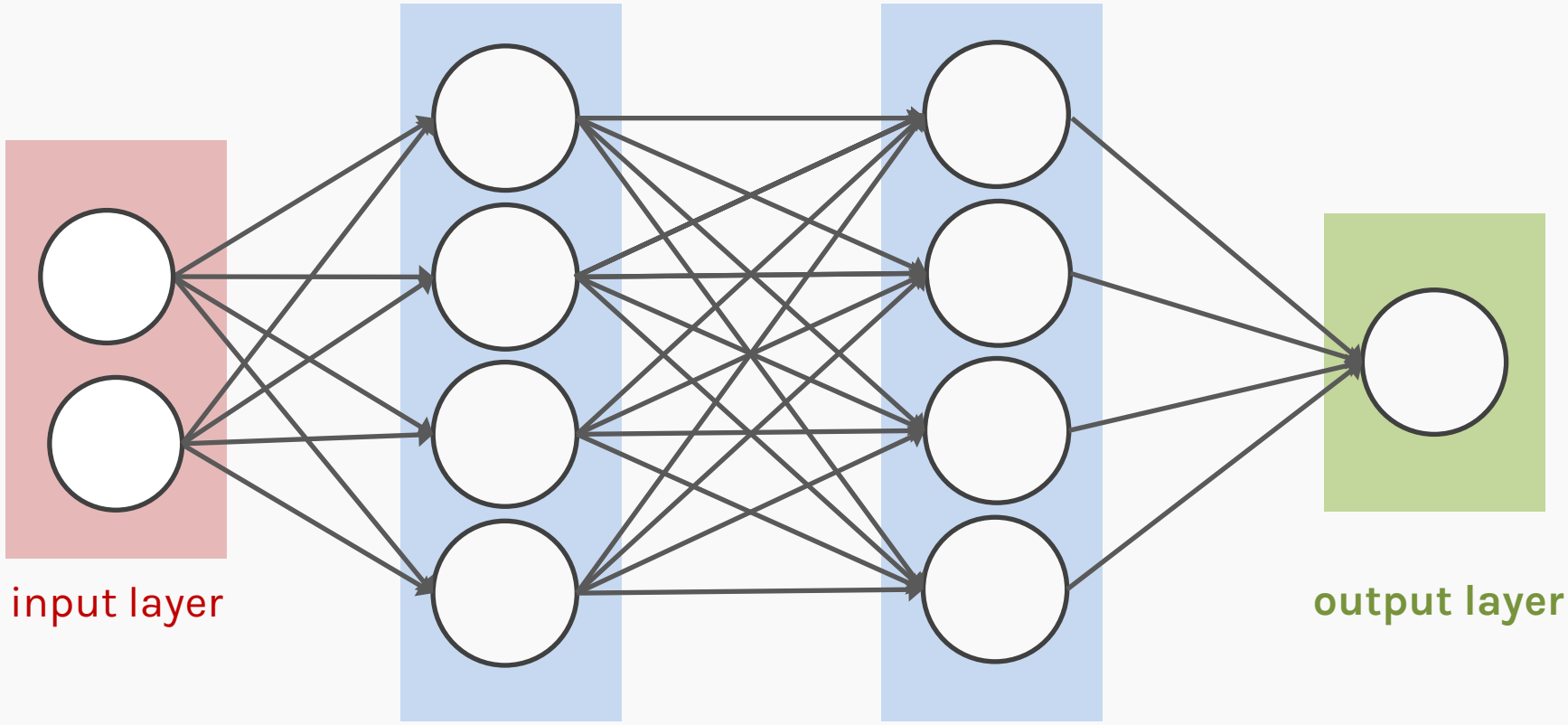
We will talk later about the choice of the number of nodes.

Anatomy of artificial neural network (ANN)



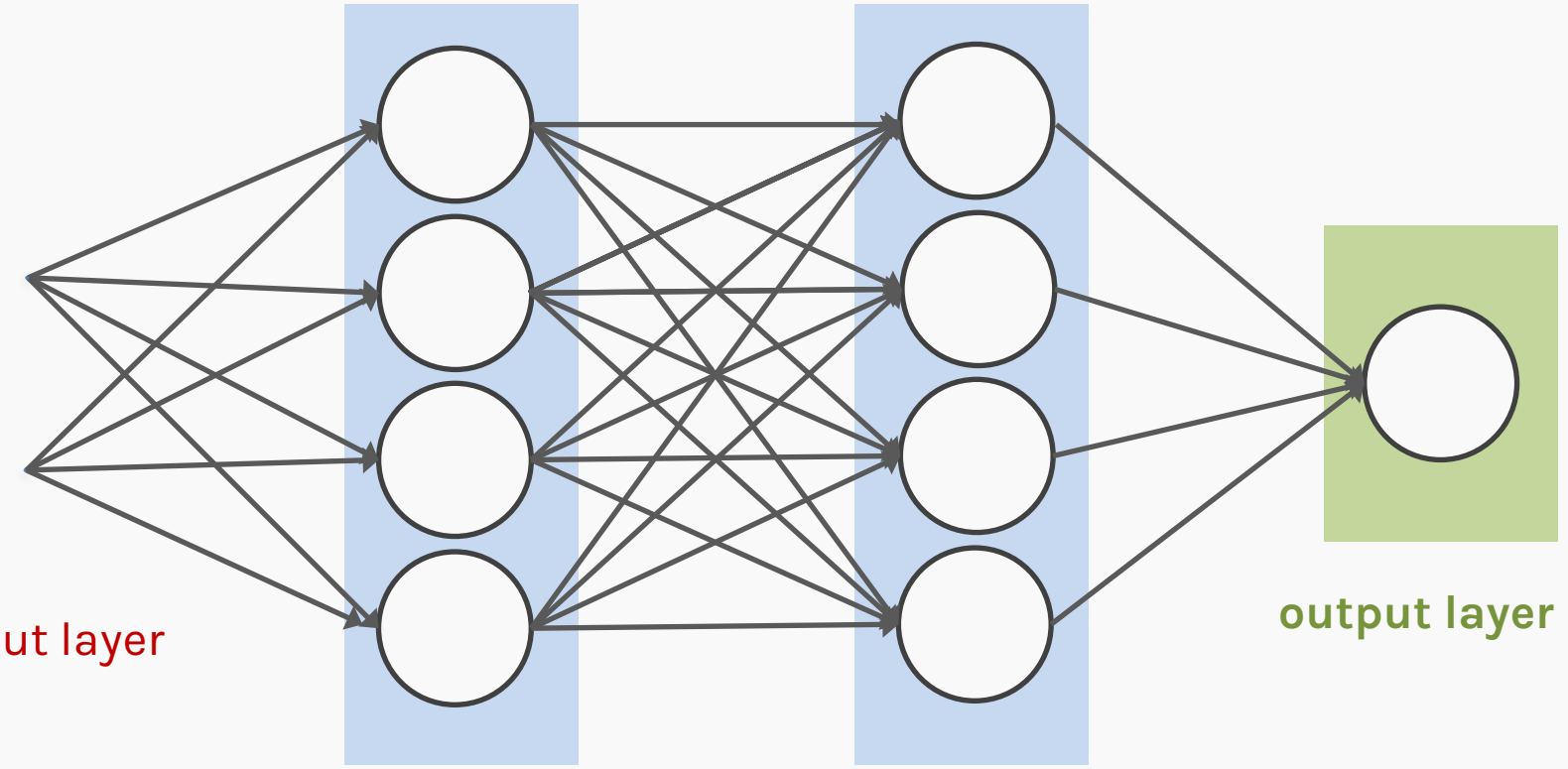
Number of inputs is specified by the data

Anatomy of artificial neural network (ANN)



Anatomy of artificial neural network (ANN)

feedforward
multi layer preceptor
dense network



input layer

input layer not usually shown as circles

hidden layer 1

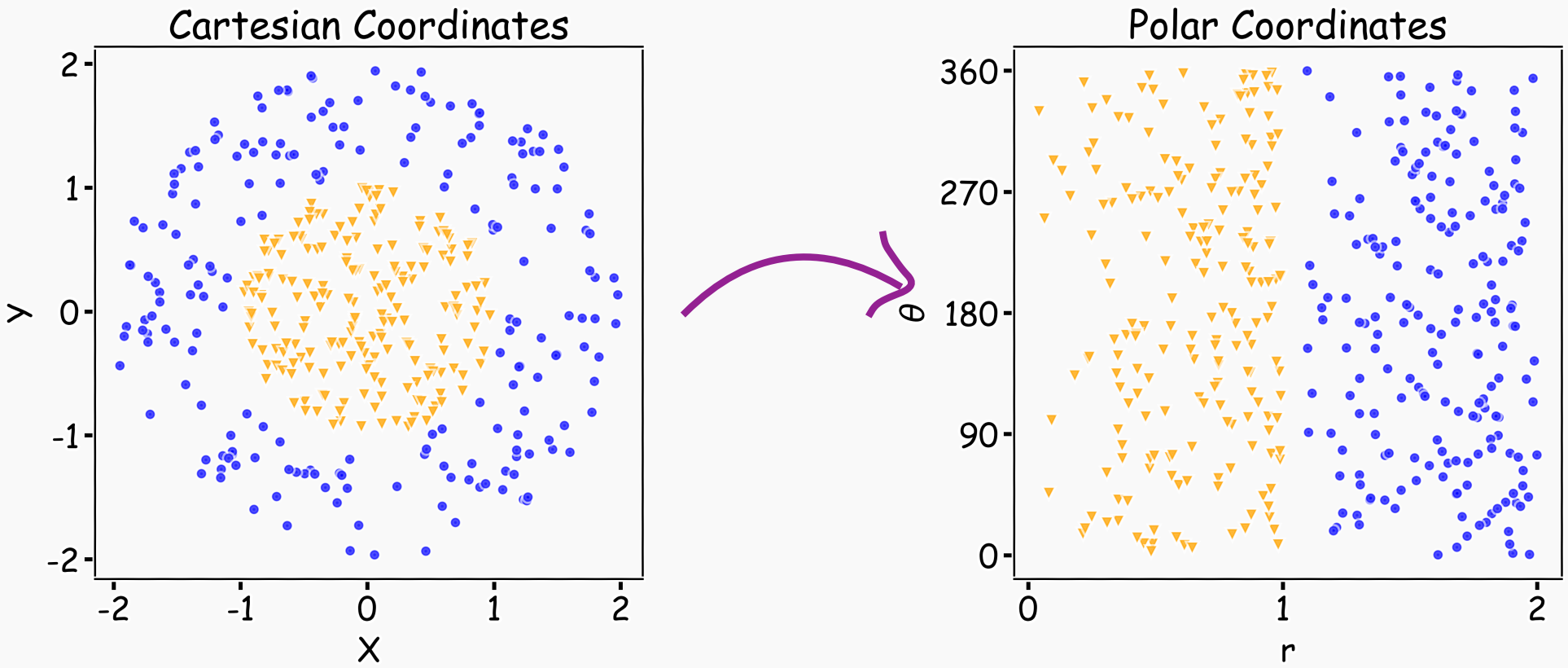
hidden layer 2

output layer

Why layers? Representation

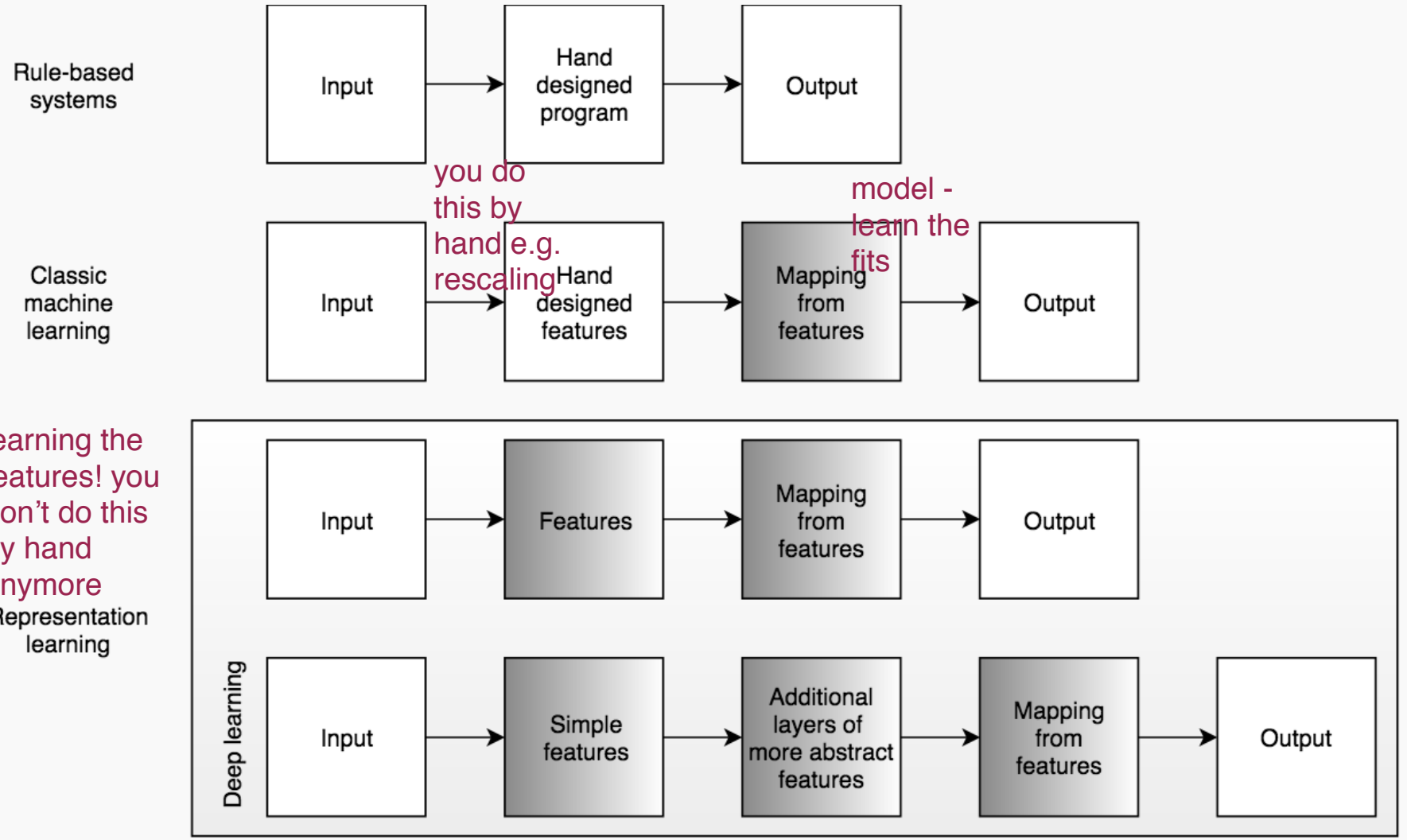
Representation matters!

circle is hard boundary to work with
BUT
if we find a 'magic' transformation, the task is much easier

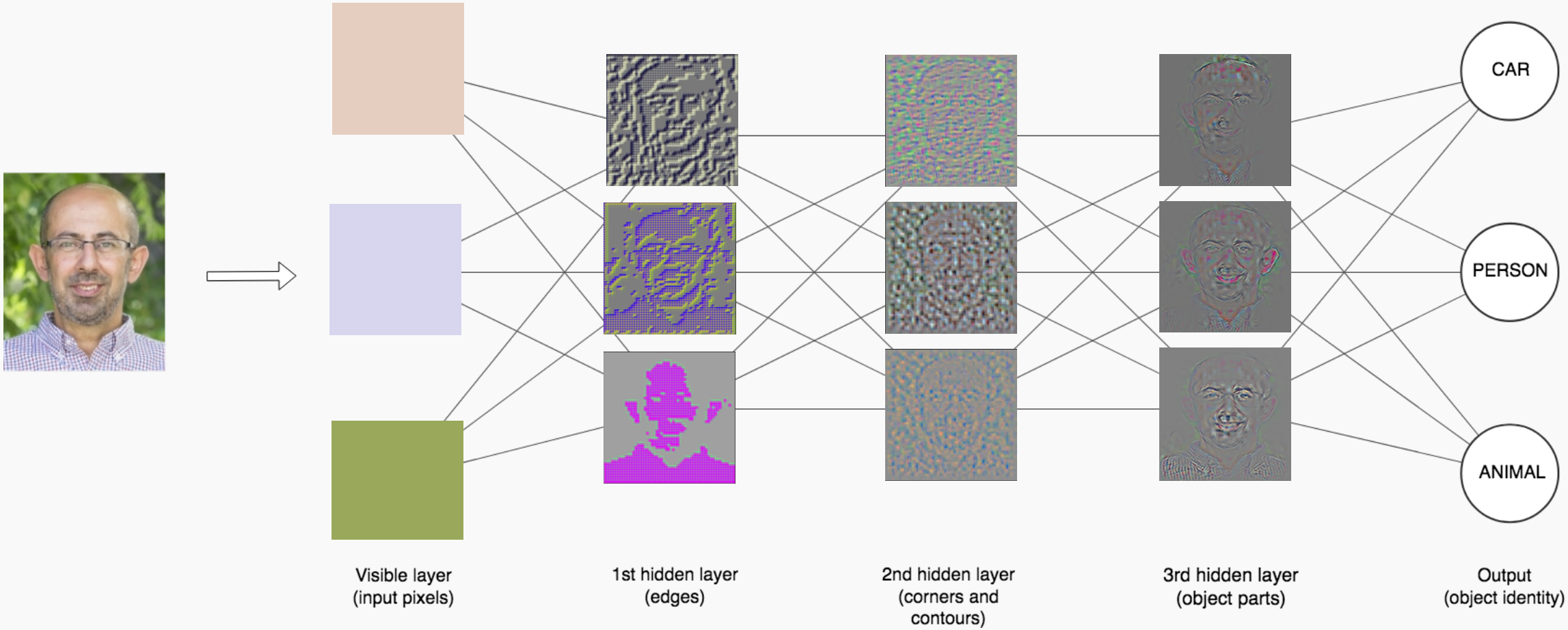


Learning Multiple Components

hand designed transform



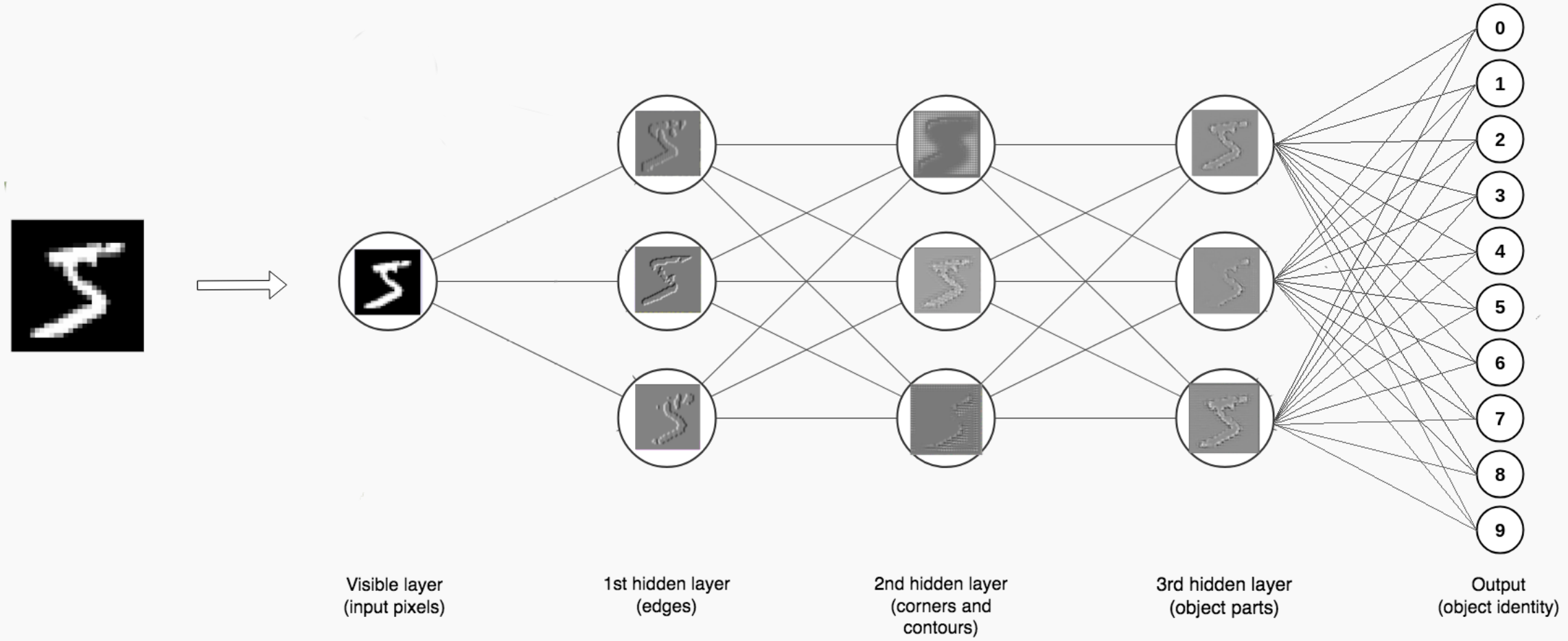
Depth = Repeated Compositions



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: we do something semi-linear: turn nonlinear into linear e.g. polynomial regression

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

Where ϕ is a *non-linear transform*

Traditional ML

Manually engineer ϕ

- Domain specific, enormous human effort

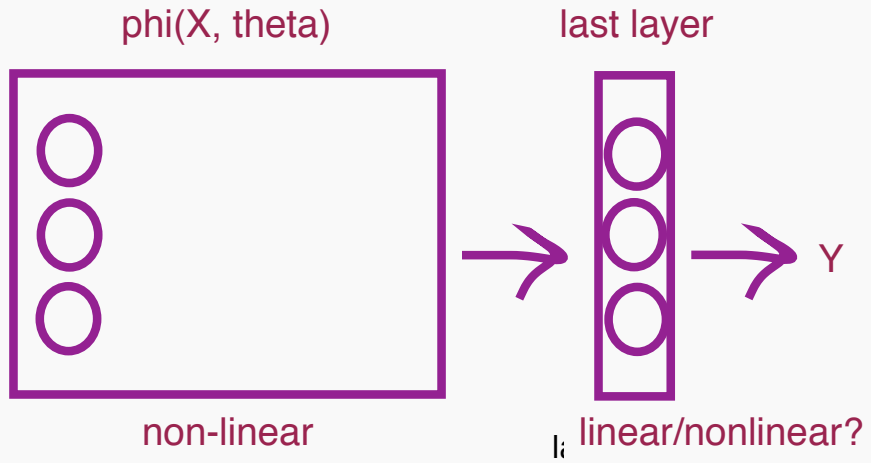
Generic transform

- 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

nonlinear

of the non-linear function

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

derivatives have to have direction for gradient descent to work; have to be large enough

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:

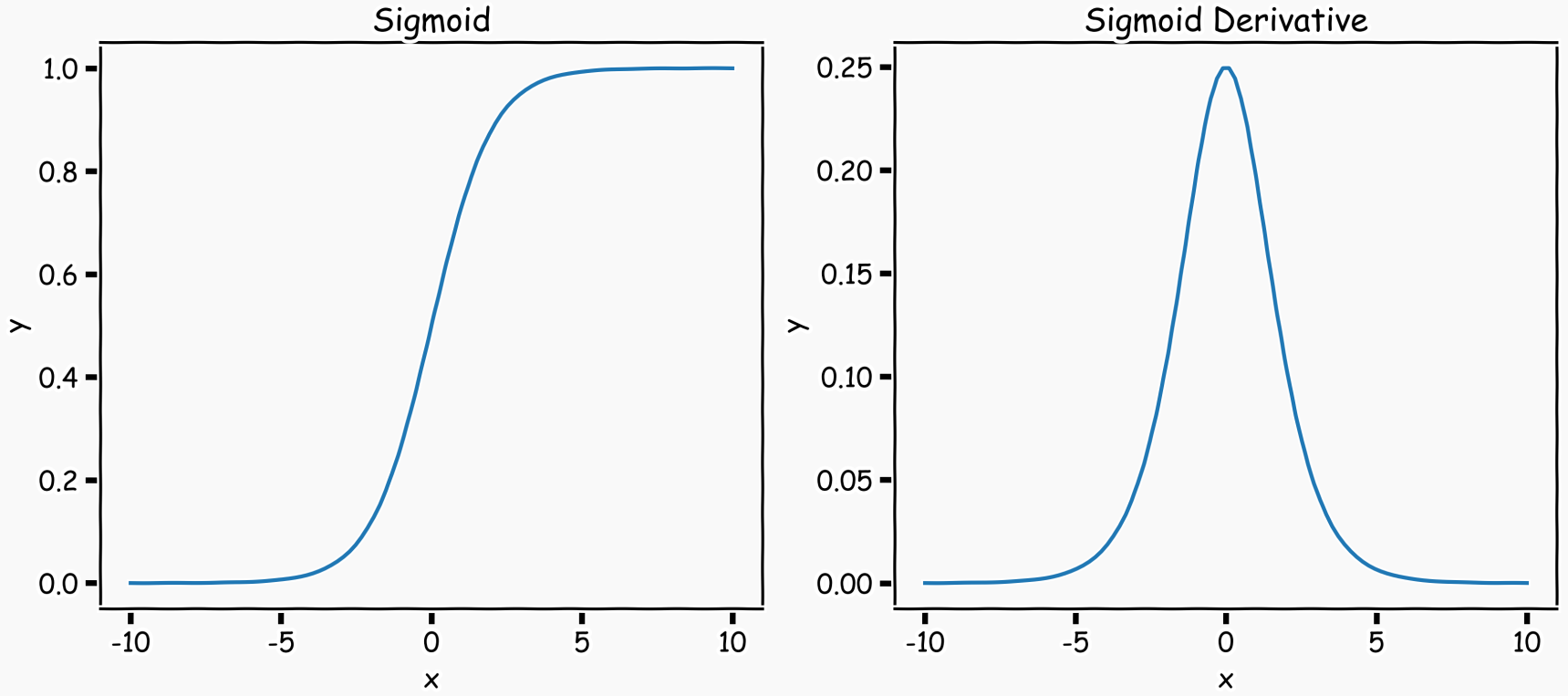
- Provide **non-linearity**
- Ensure gradients remain large through hidden unit

Common choices are

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

Sigmoid (aka Logistic)

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

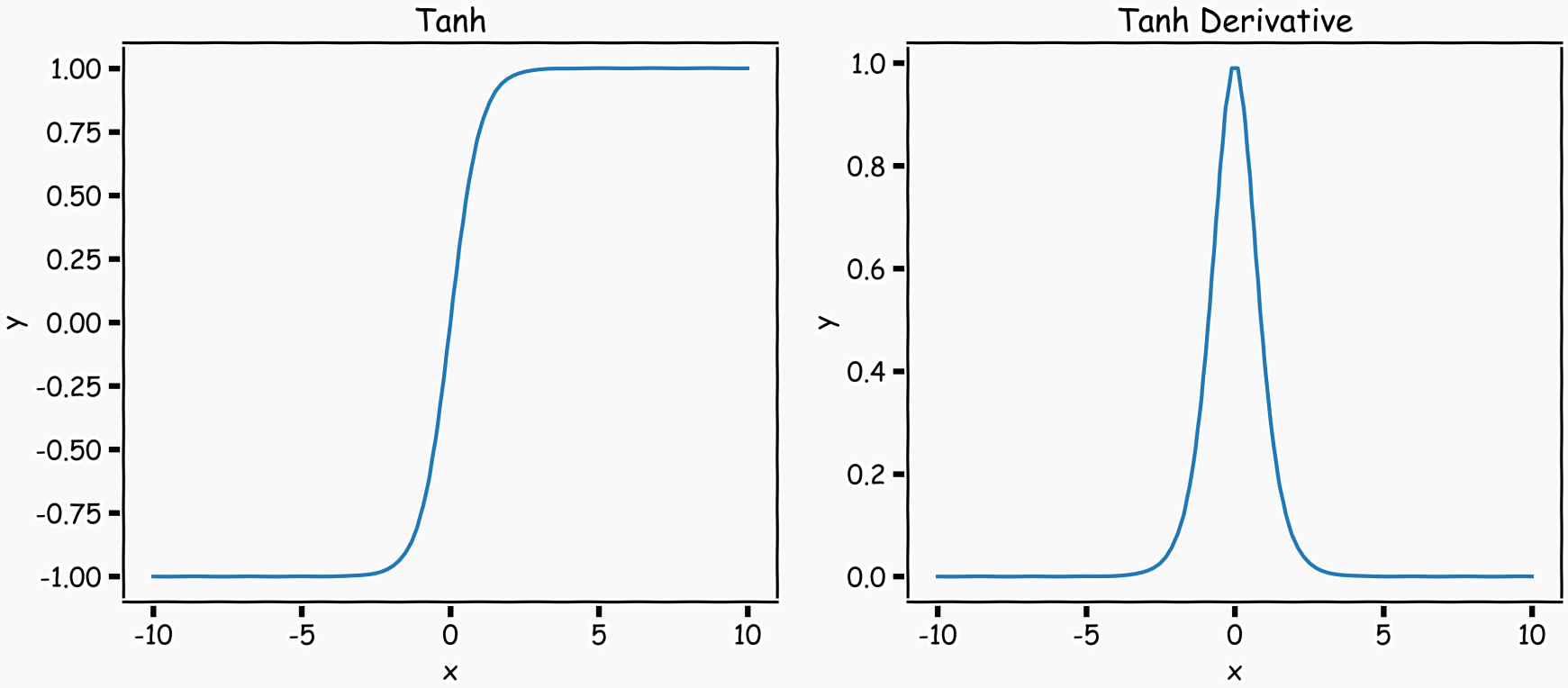


Derivative is **zero** for much of the domain. This leads to "vanishing gradients" in backpropagation.

prefer to have activation fxns where we don't have derivative as zero everywhere

Hyperbolic Tangent (Tanh)

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



Same problem of "vanishing gradients" as sigmoid.

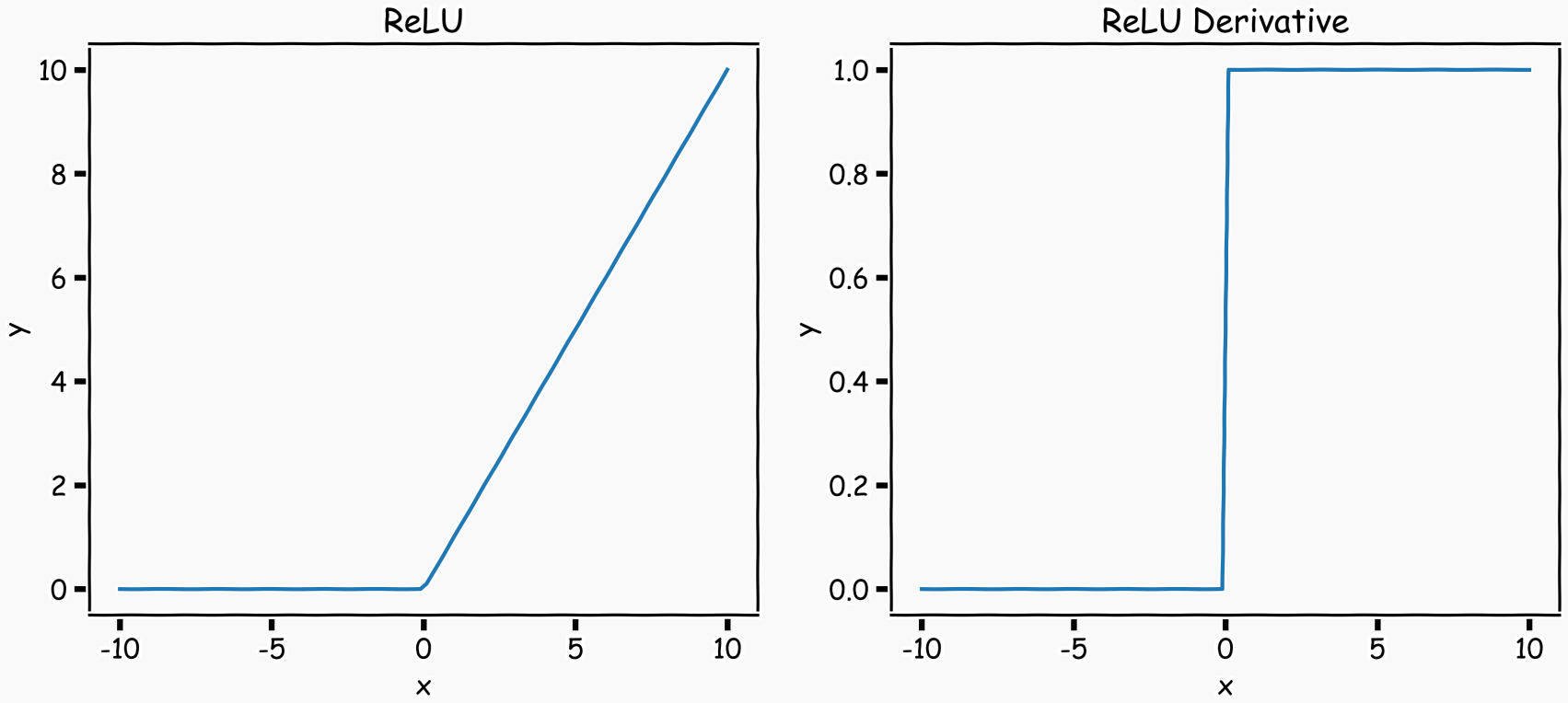
Rectified Linear Unit (ReLU)

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

the kink is the nonlinearity

derivative at 0 is undefined - we kinda ignore it and just pick derivative = 0 or 1

why is it good that we have zero derivative? its fantastic for making networks less complex - parts of the network get 'deactivated' since output is 0



Two major advantages:

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

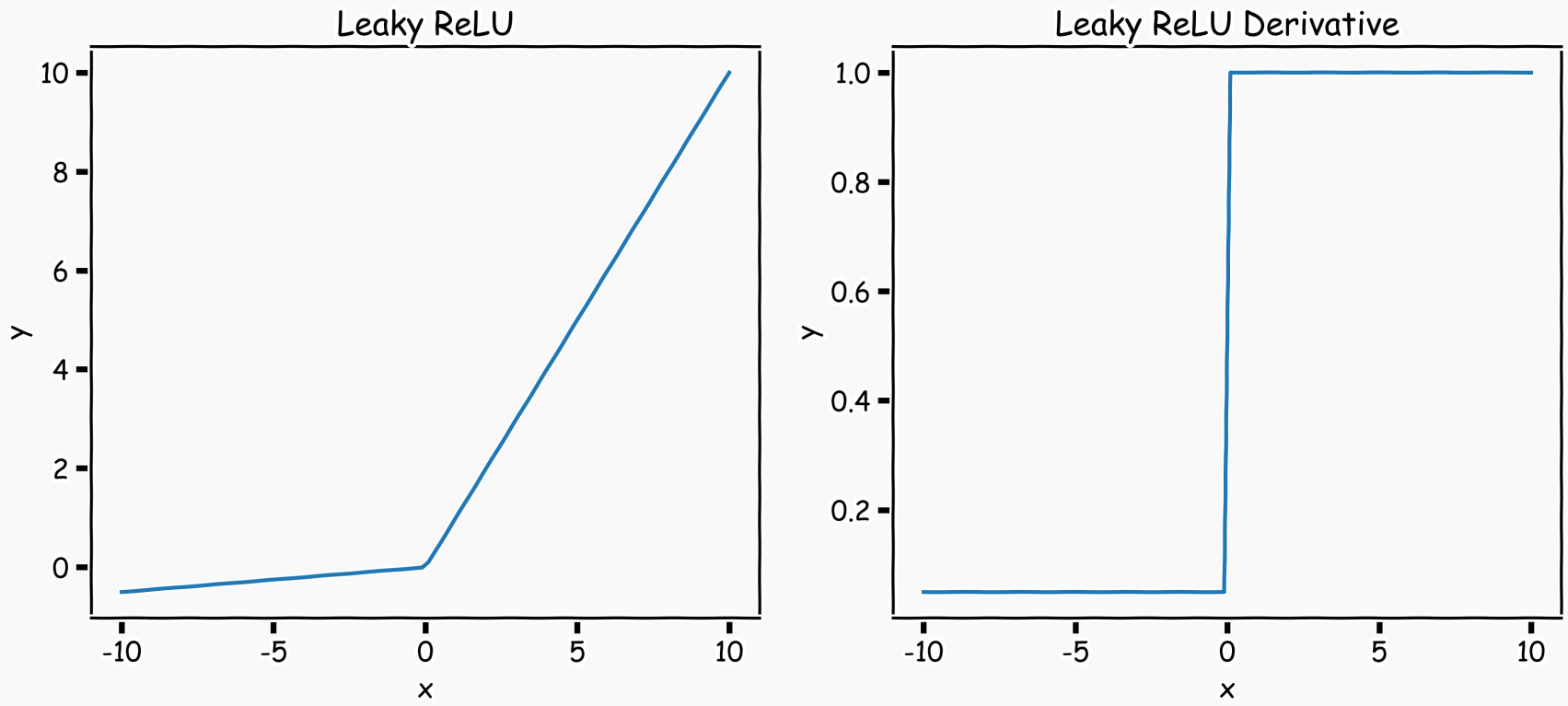
Leaky ReLU

(0, X)

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

where α takes a small value

alpha is typically 0.1

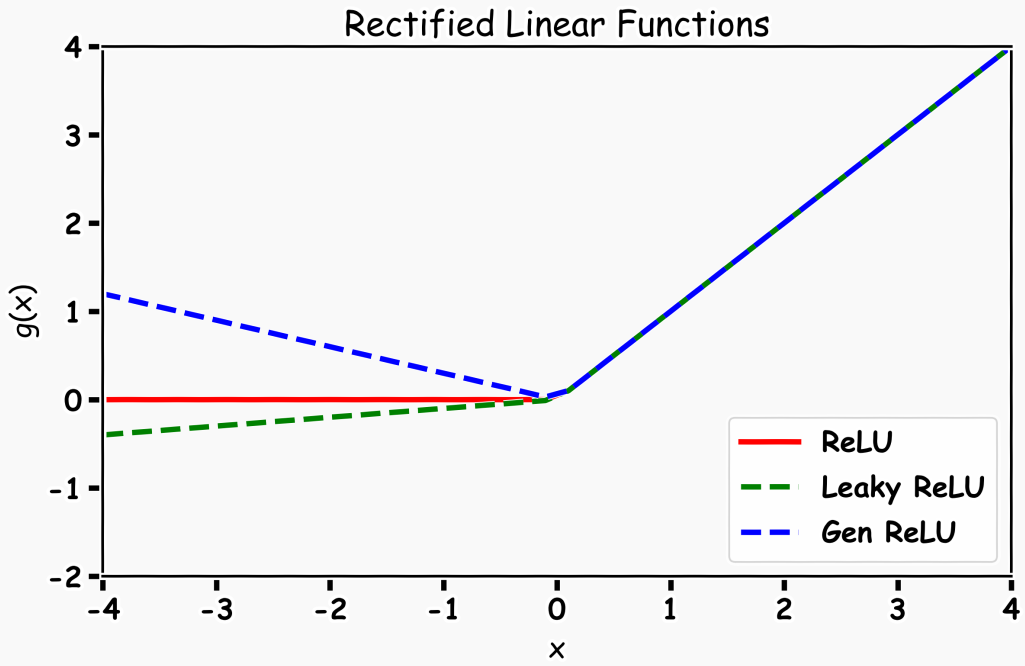


- Tries to fix "dying ReLU" problem: derivative is non-zero everywhere.
- Some people report success with this form of activation function, but the results are not always consistent

Generalized ReLU

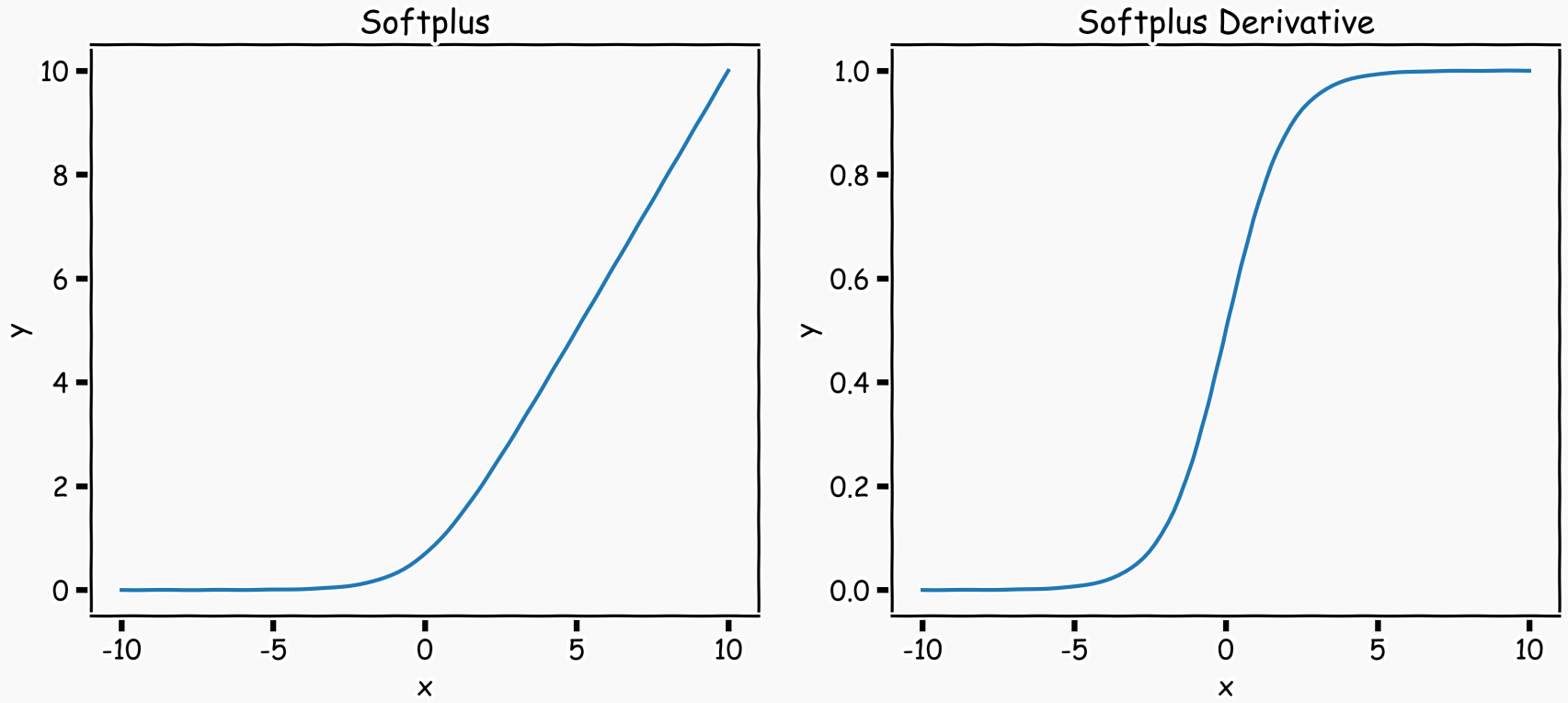
Generalization: For $\alpha_i > 0$

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



softplus

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

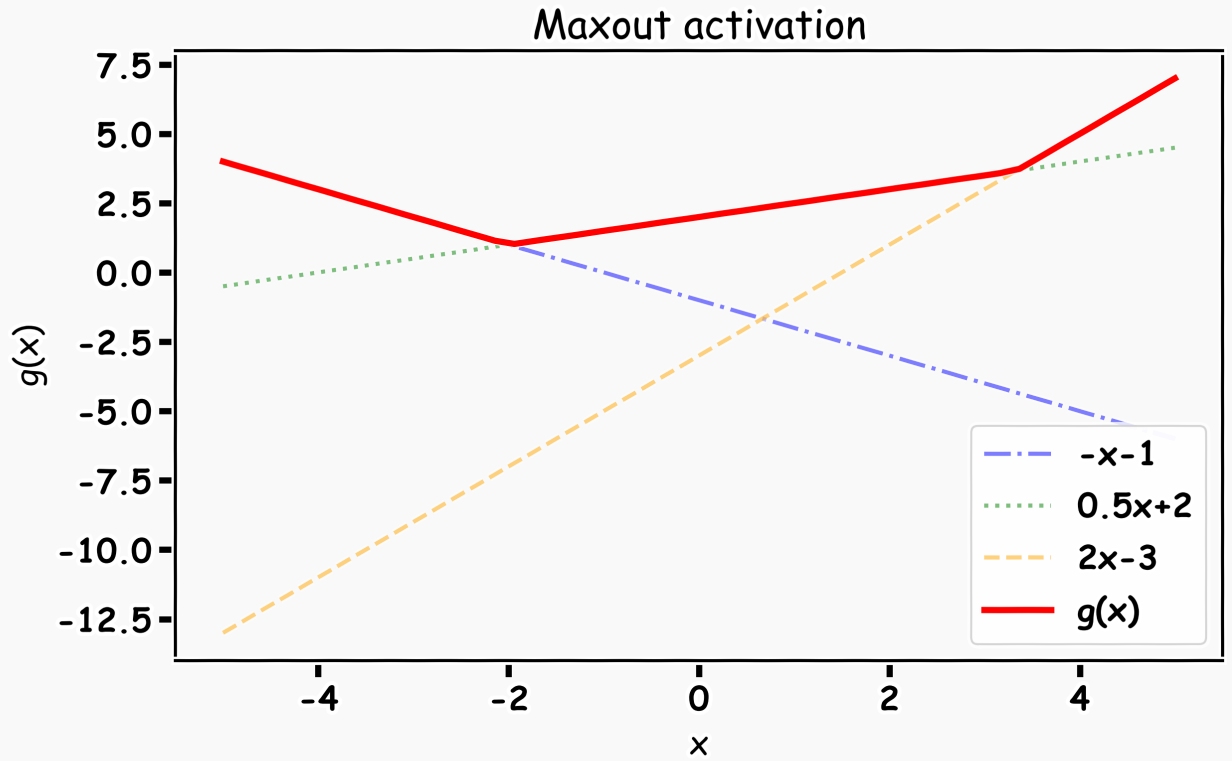


The logistic sigmoid function is a smooth approximation of the derivative of the rectifier

Maxout

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

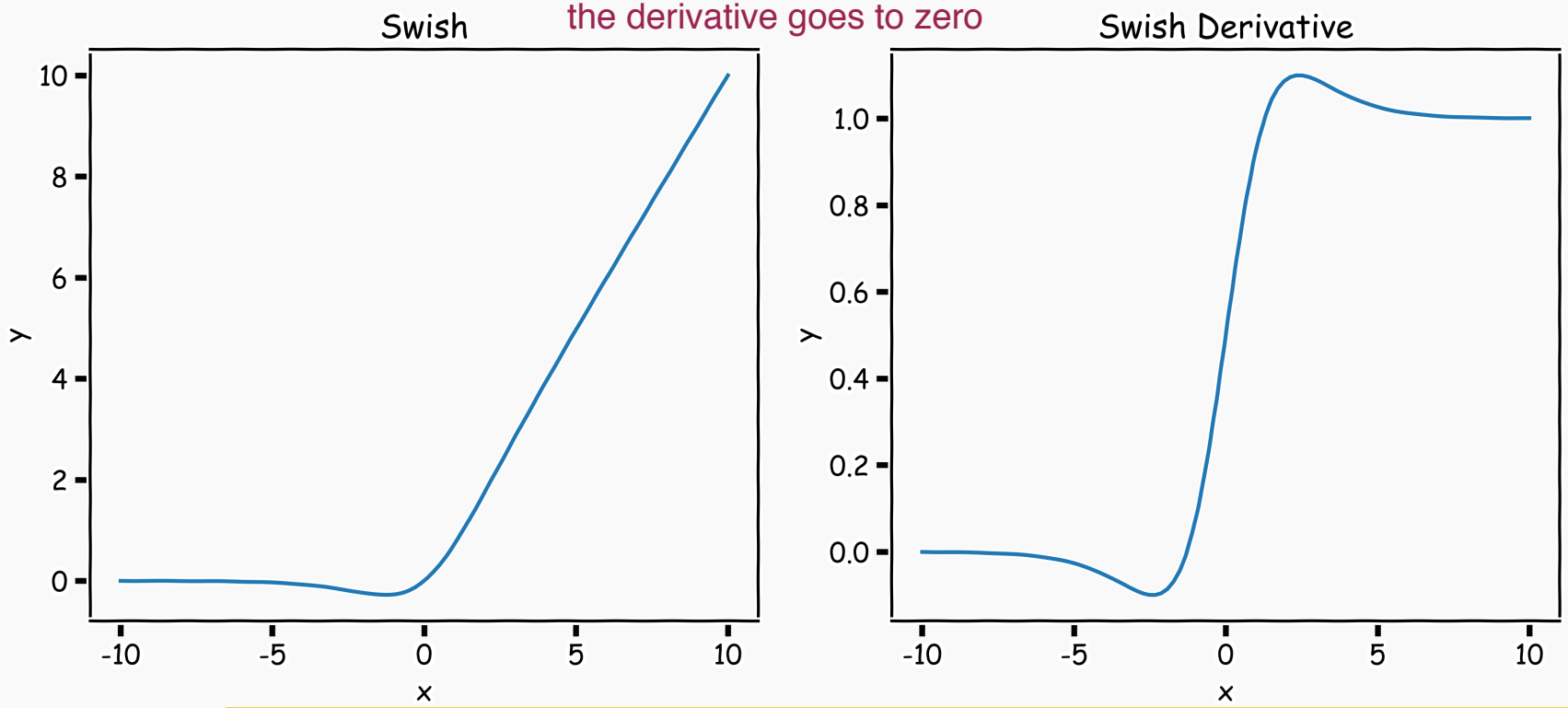
$$g(x) = \max_{i \in \{1,\ldots,k\}} \alpha_i x_i + \beta$$



Swish: A Self-Gated Activation Function

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

some more control over where the derivative goes to zero



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:

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

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

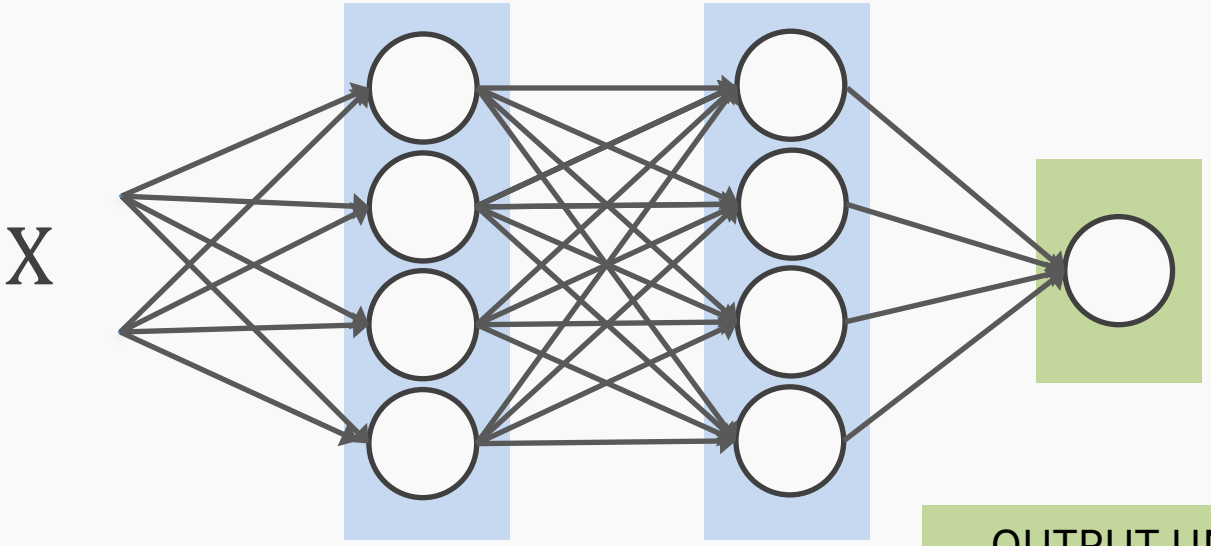
Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli		Binary Cross Entropy

Output Units

what do we do at the output layer for a classification problem???

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

Output unit for binary classification

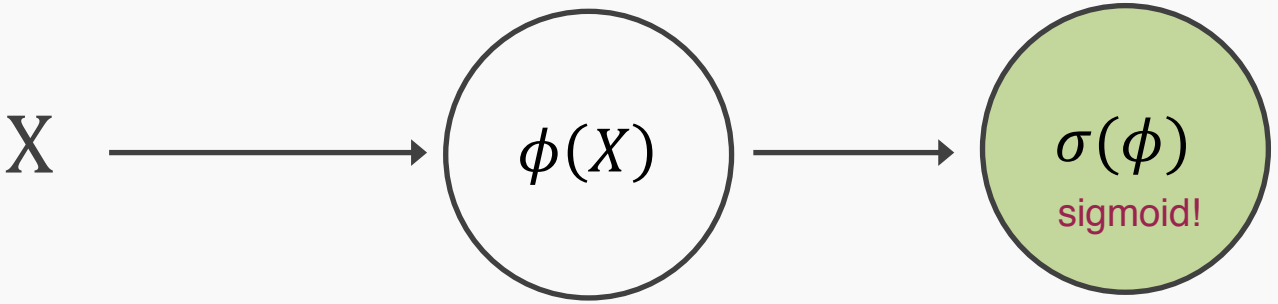


output should be probability of class 1; needs to be bound by 0 and 1

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

OUTPUT UNIT

output units are special - ReLU doesn't apply here - only to the hidden layers



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

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

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		

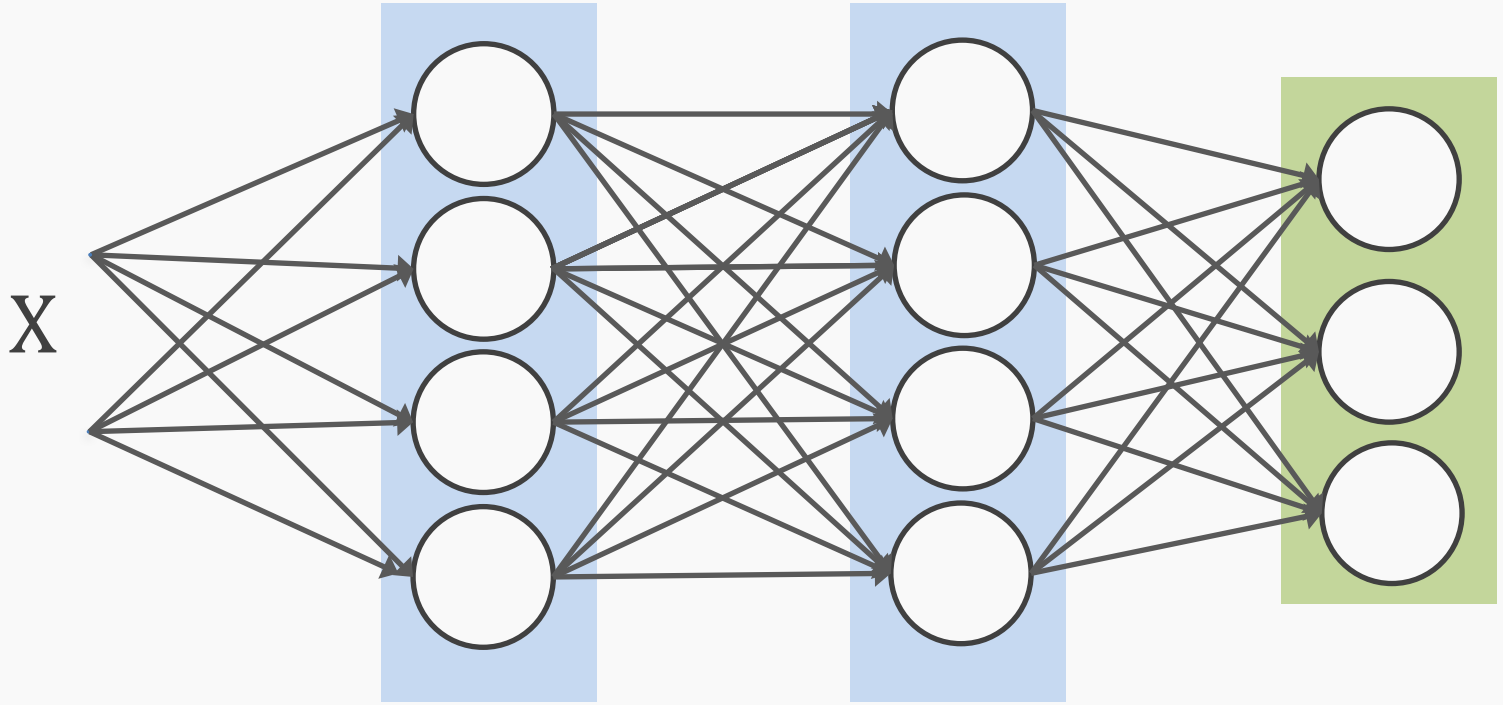
Output Units

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



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

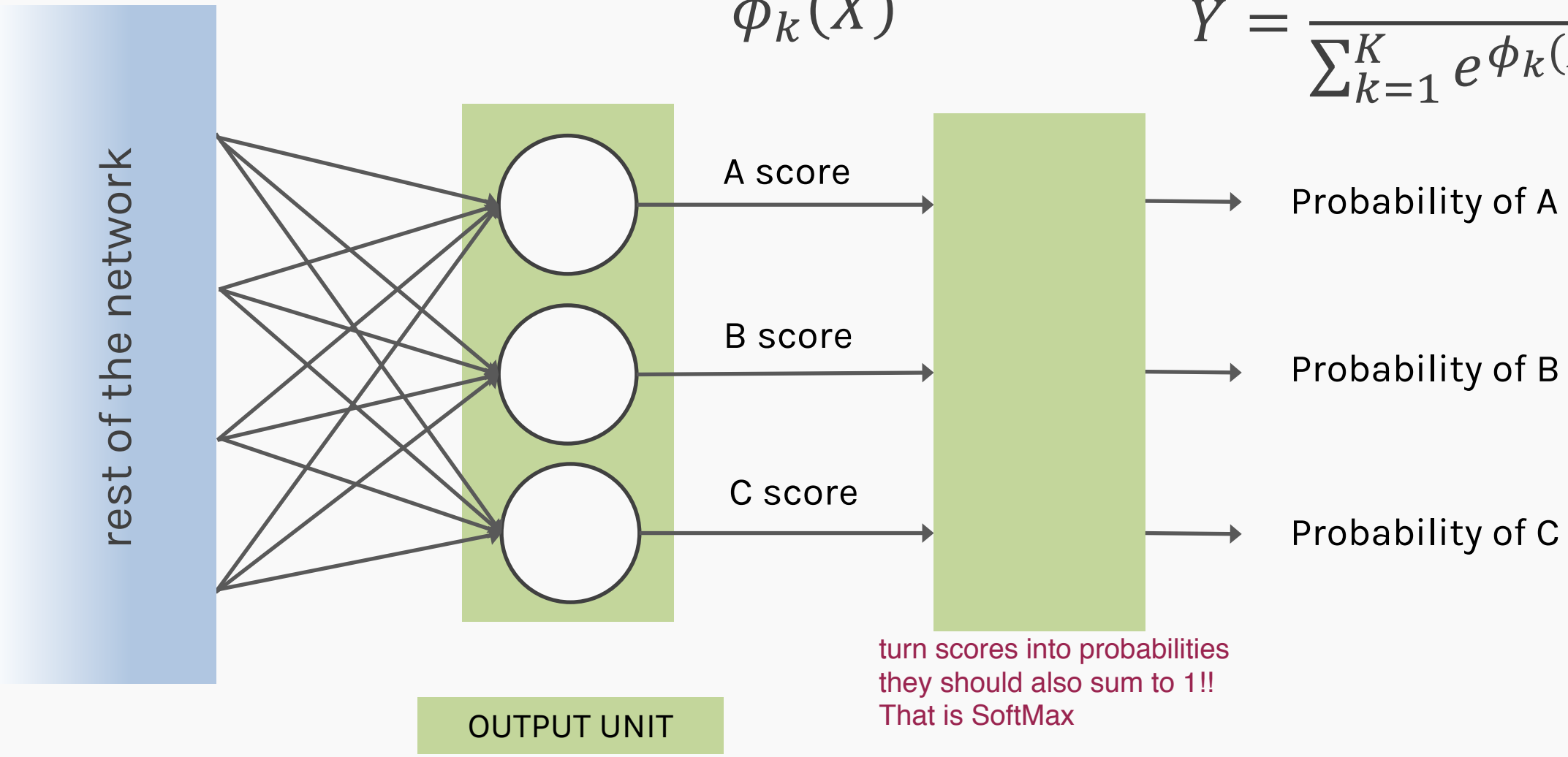
OUTPUT UNIT

3 object classification

SoftMax

$$\phi_k(X)$$

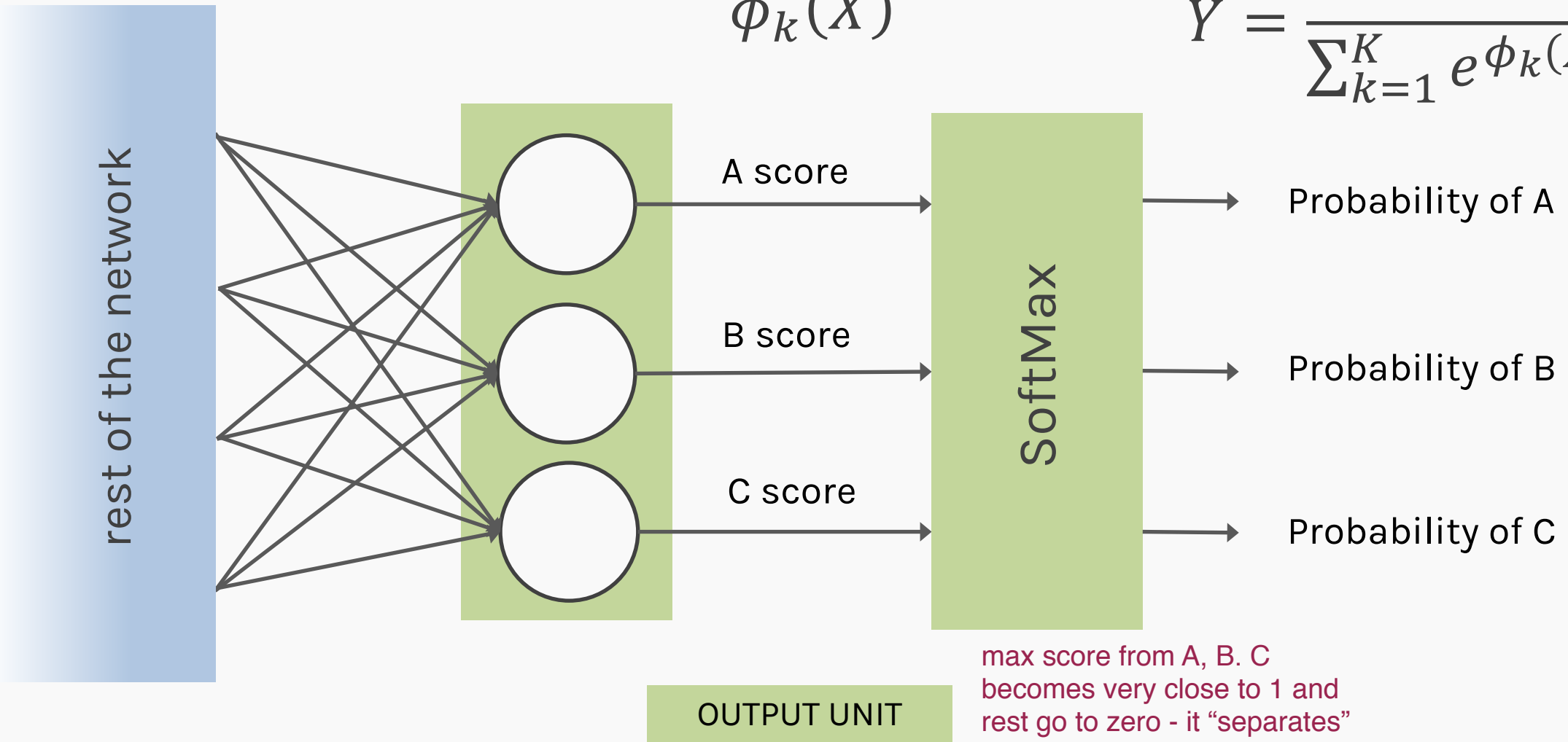
$$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$$



SoftMax

$$\phi_k(X)$$

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

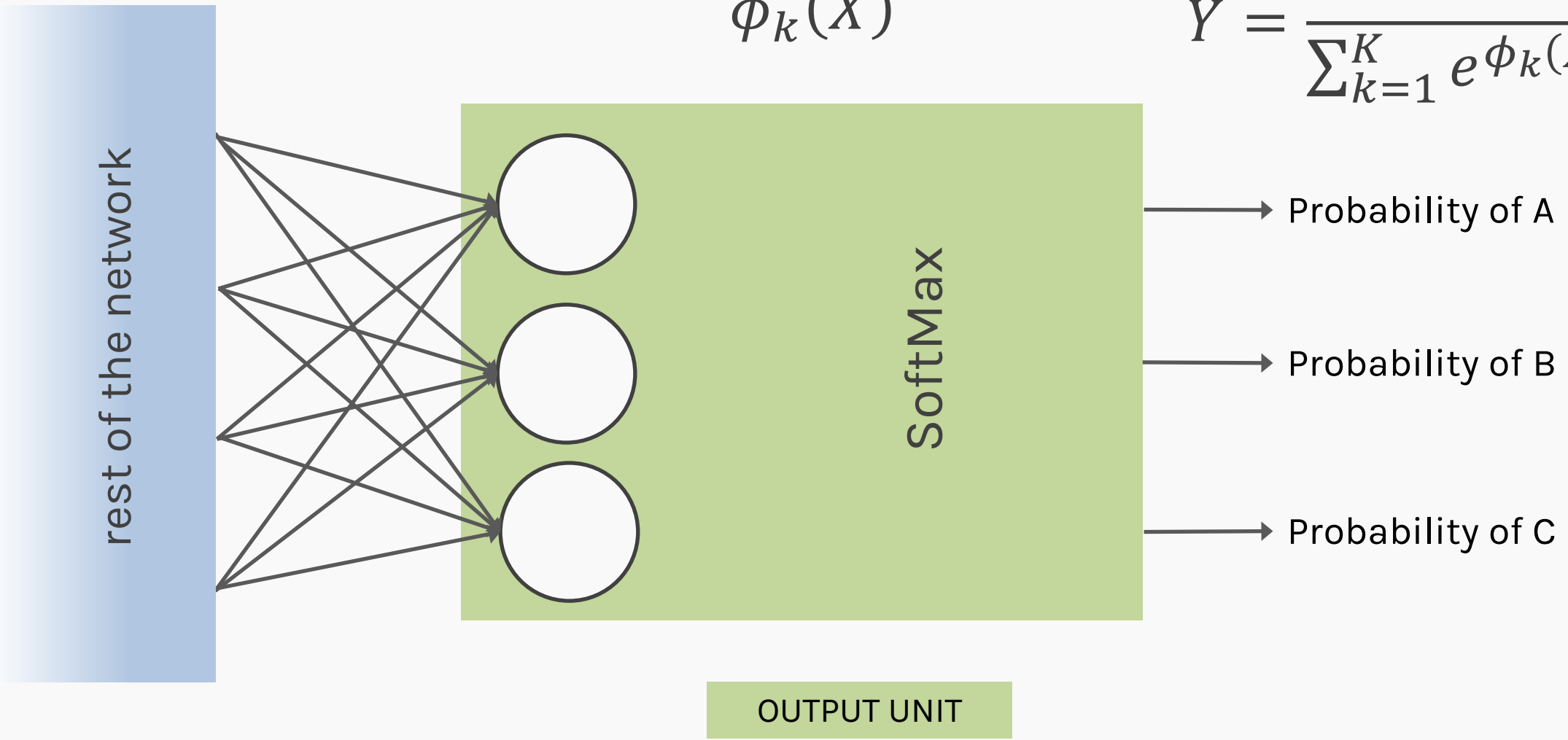


max score from A, B. C becomes very close to 1 and rest go to zero - it "separates" the classes due to the exponential term

SoftMax

$$\phi_k(X)$$

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



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			

Output Units

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

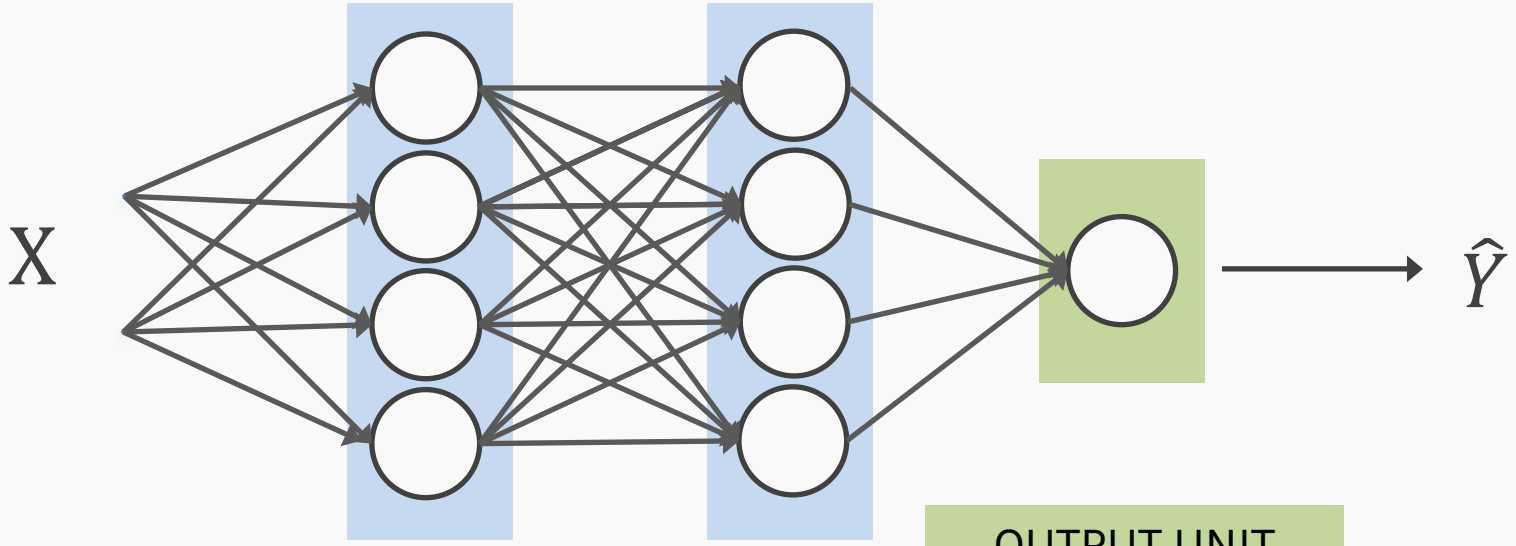
Output Units

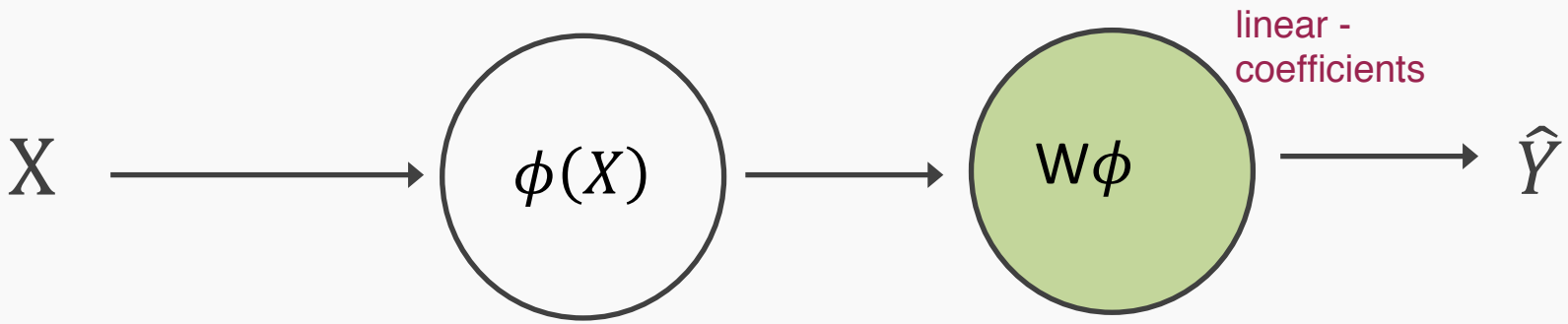
Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian		MSE

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

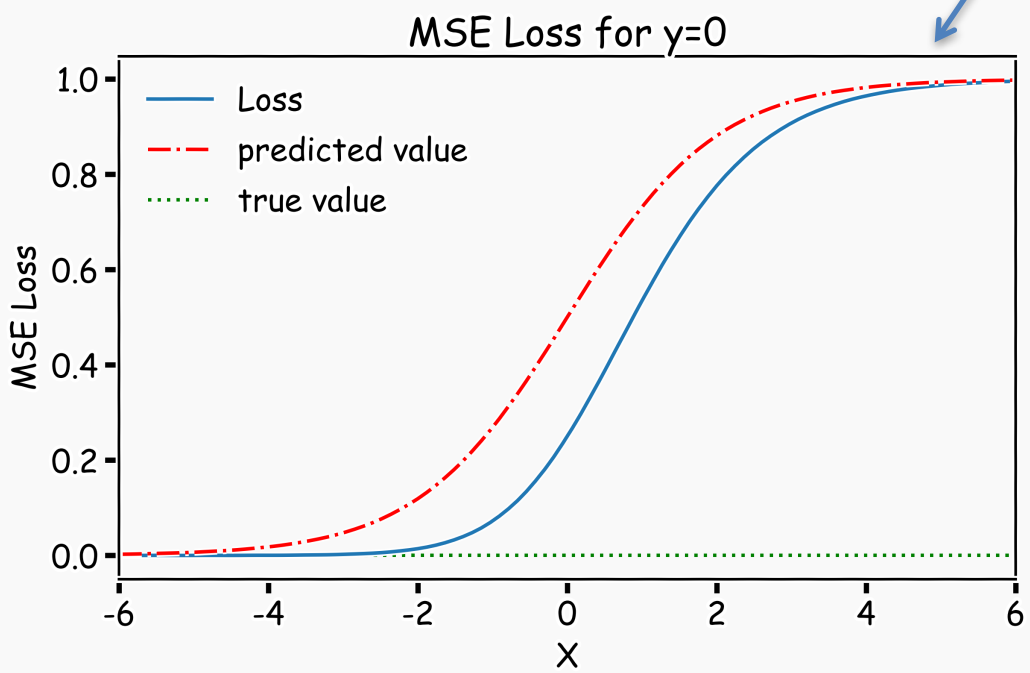
Loss Function

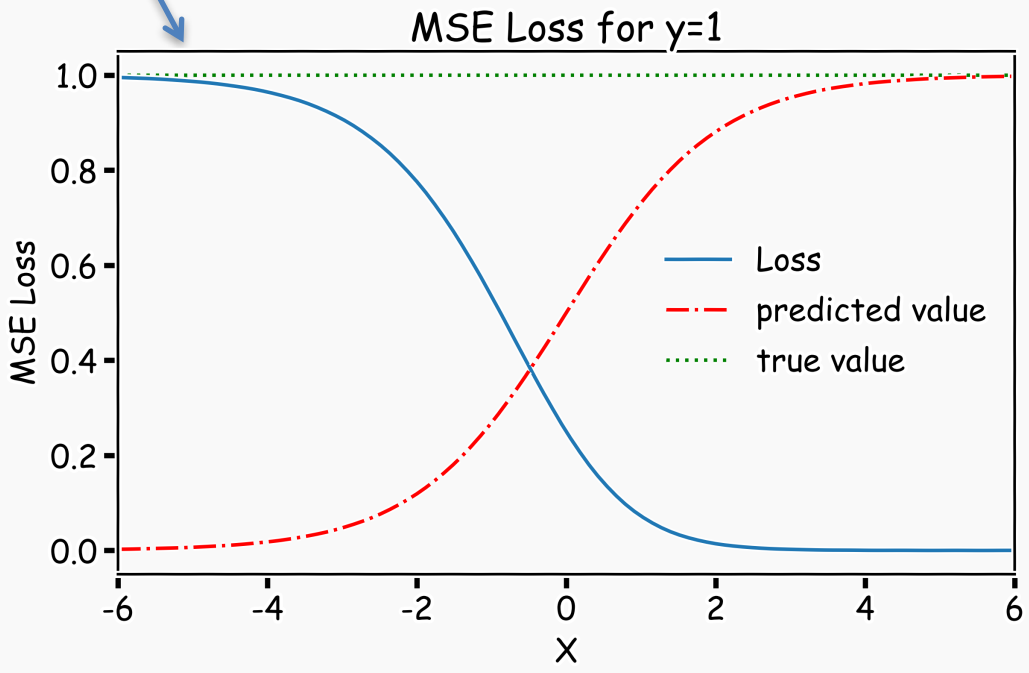
Example: sigmoid output + squared loss

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

derivative is zero

Flat surfaces

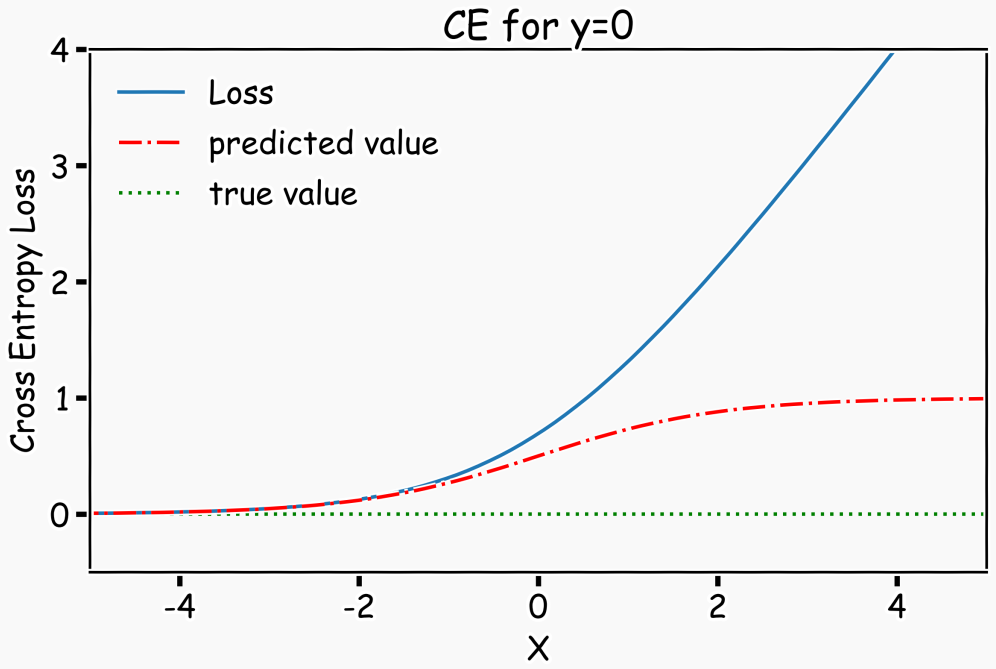


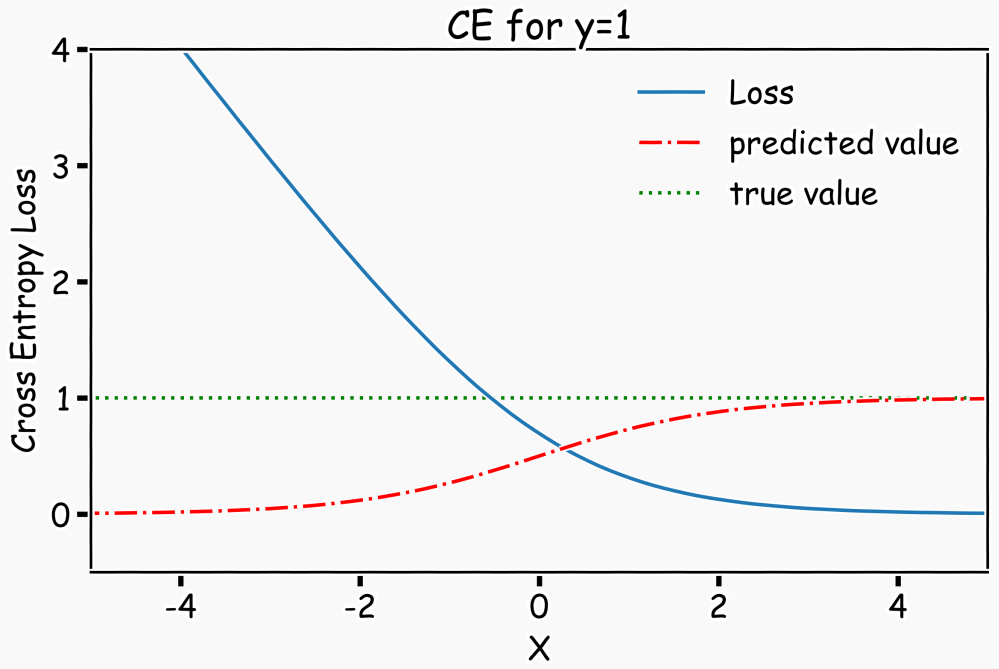


Cost Function

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

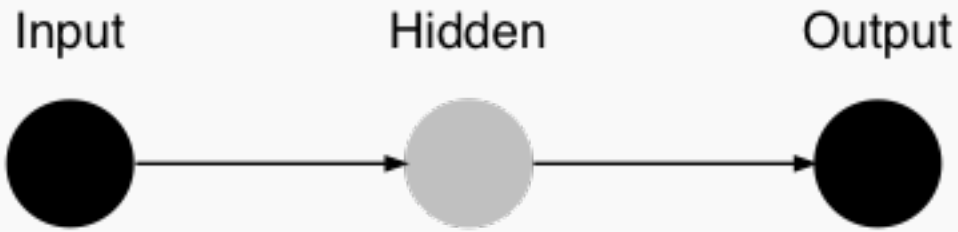
Loss function

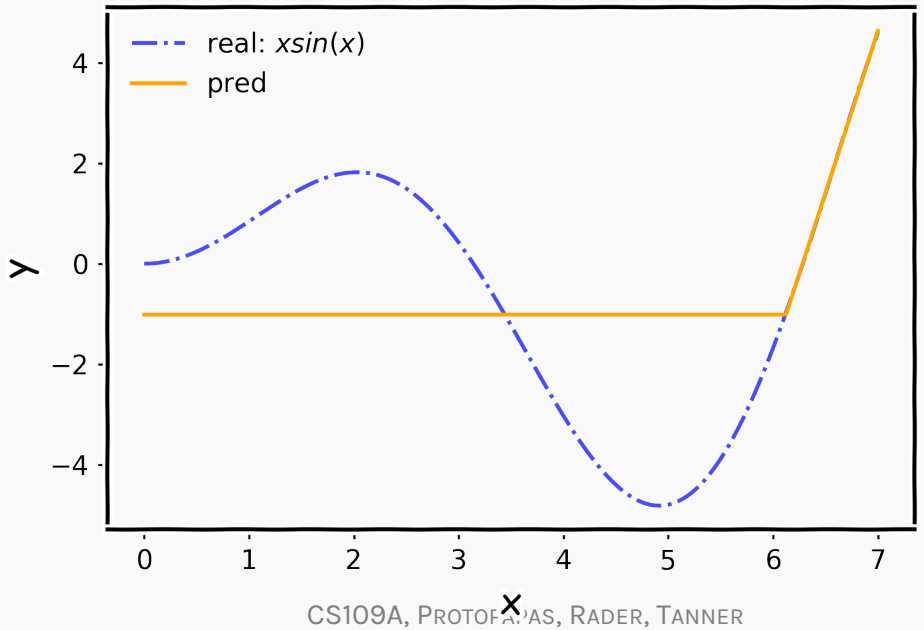
Output units

Architecture

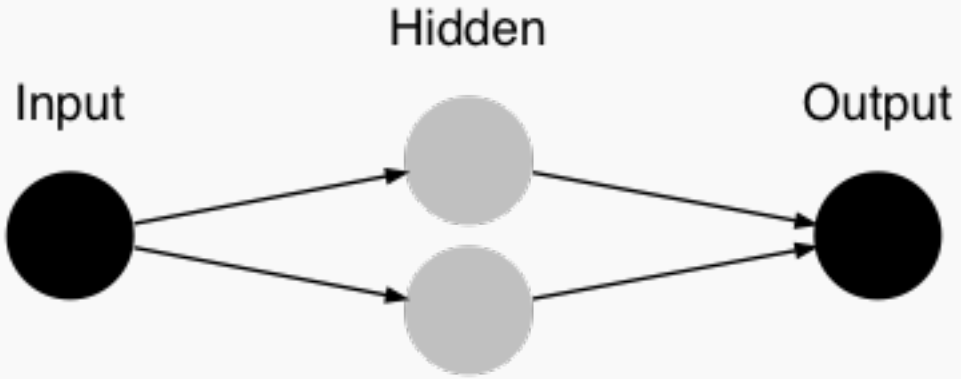
Optimizer

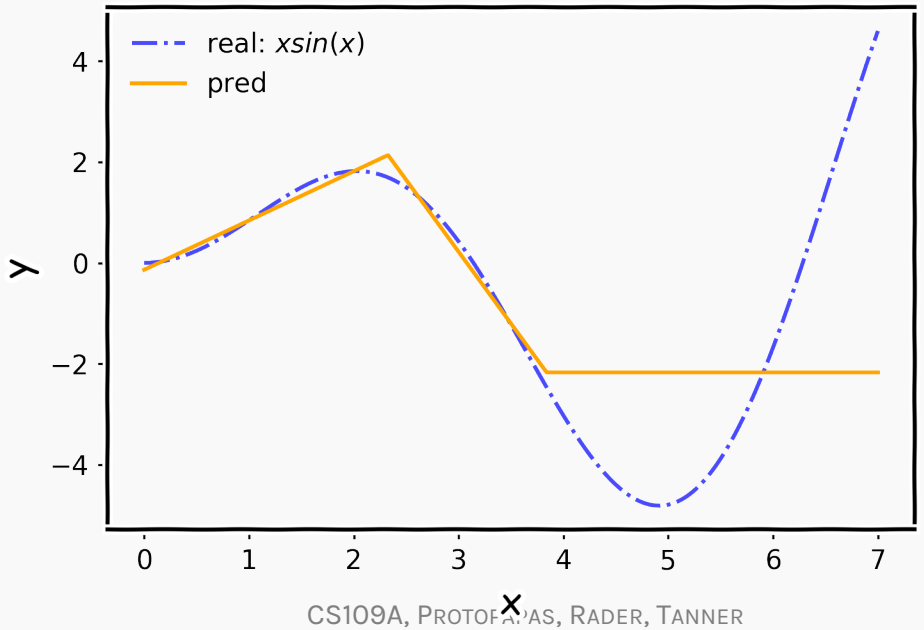
NN in action



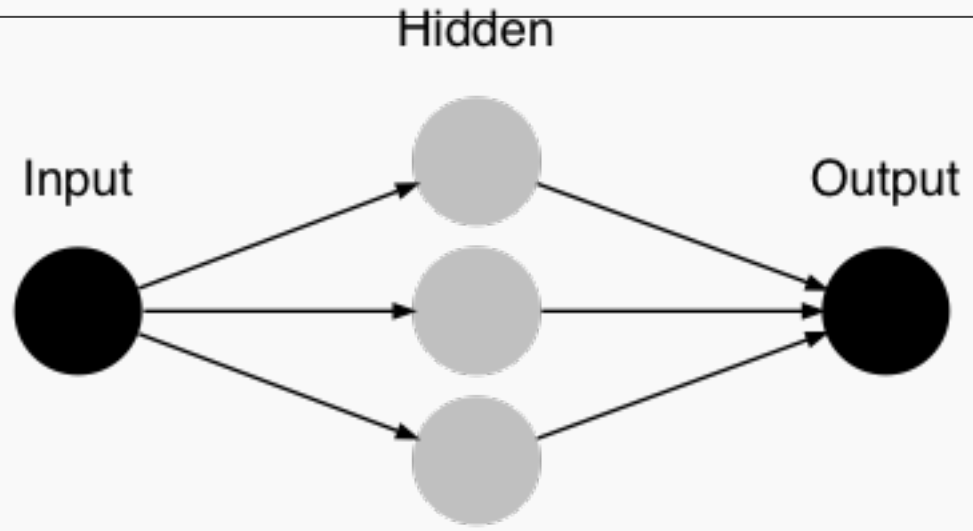


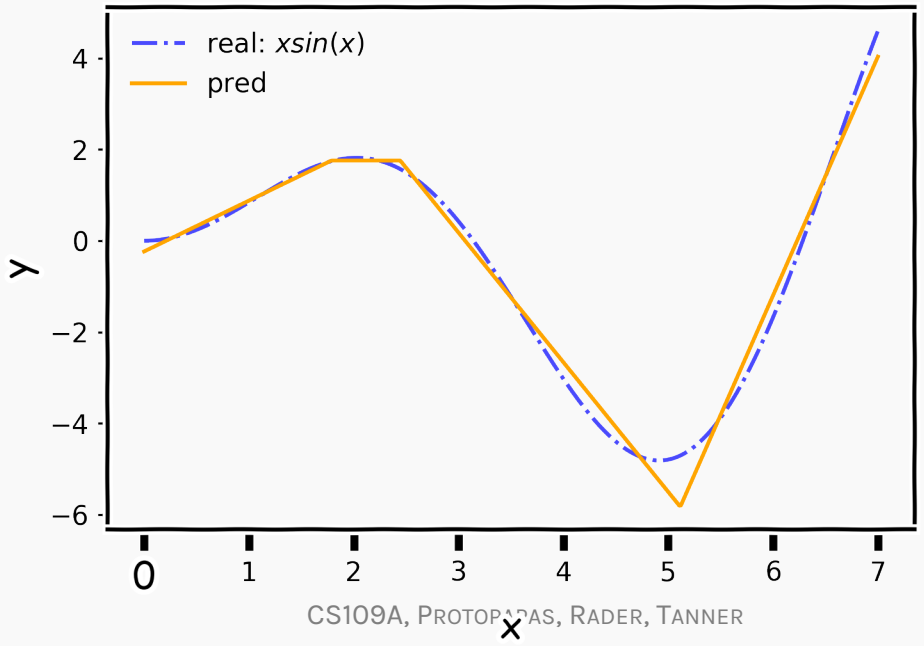
NN in action



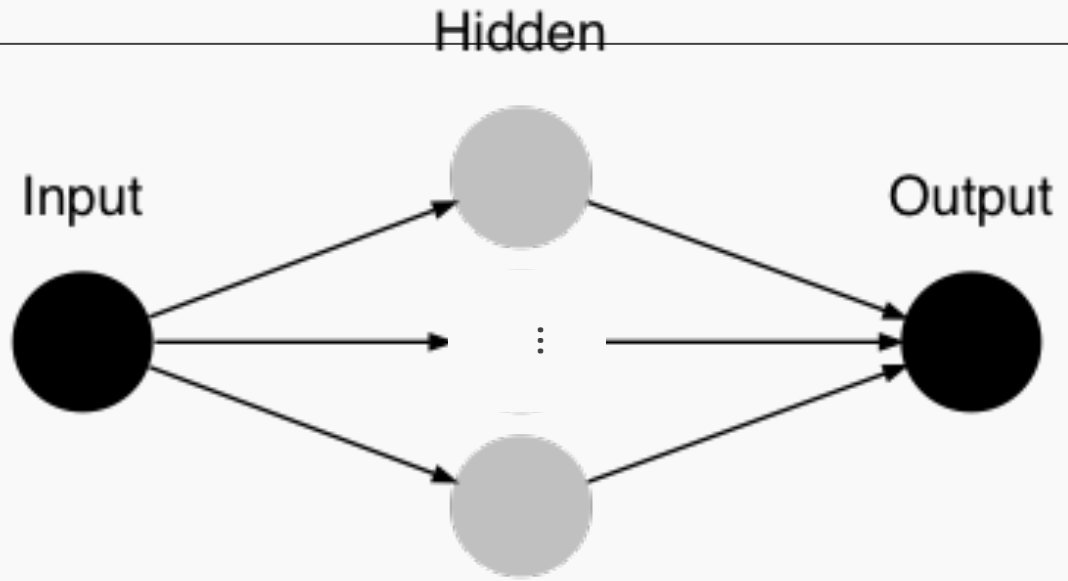


NN in action

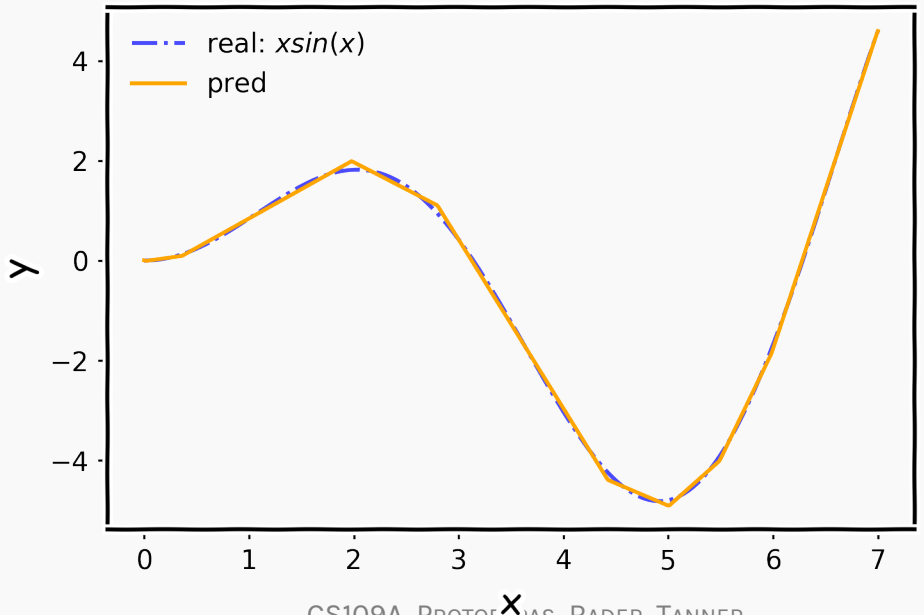




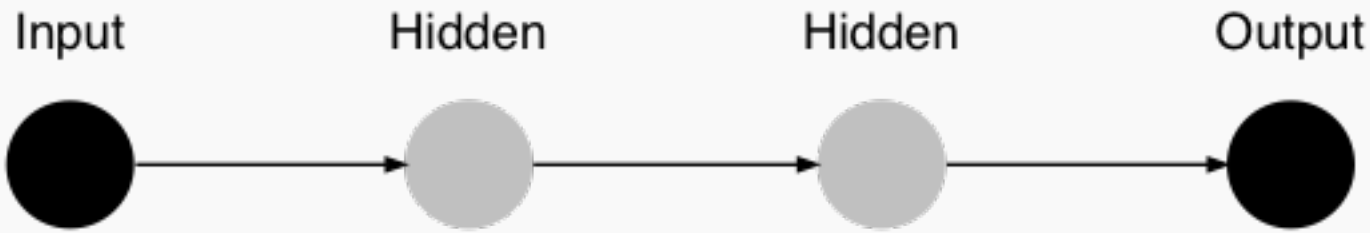
NN in action

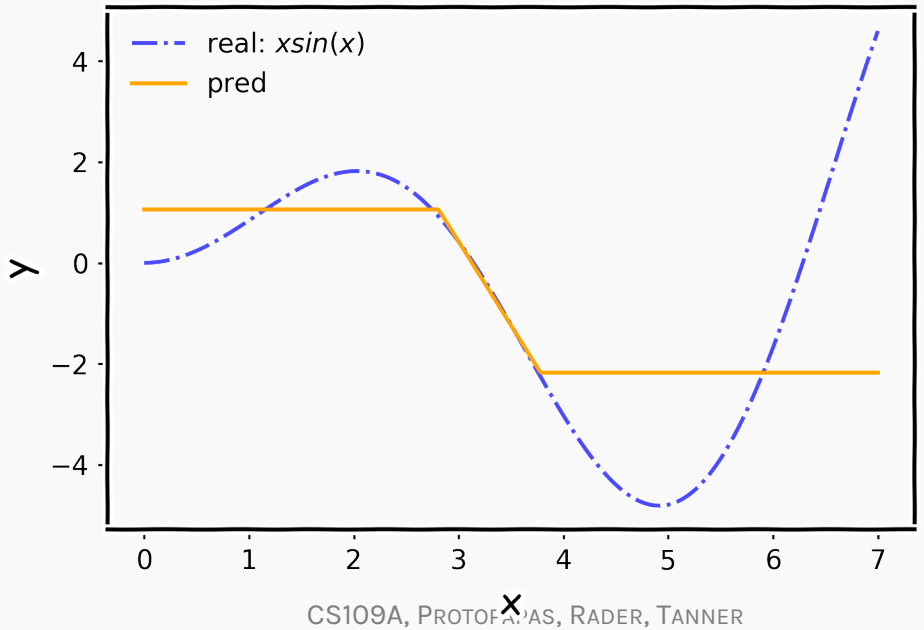


a lot of nodes - you can basically do whatever you want (exactly what you want!)
BUT it can overfit, take forever to fit

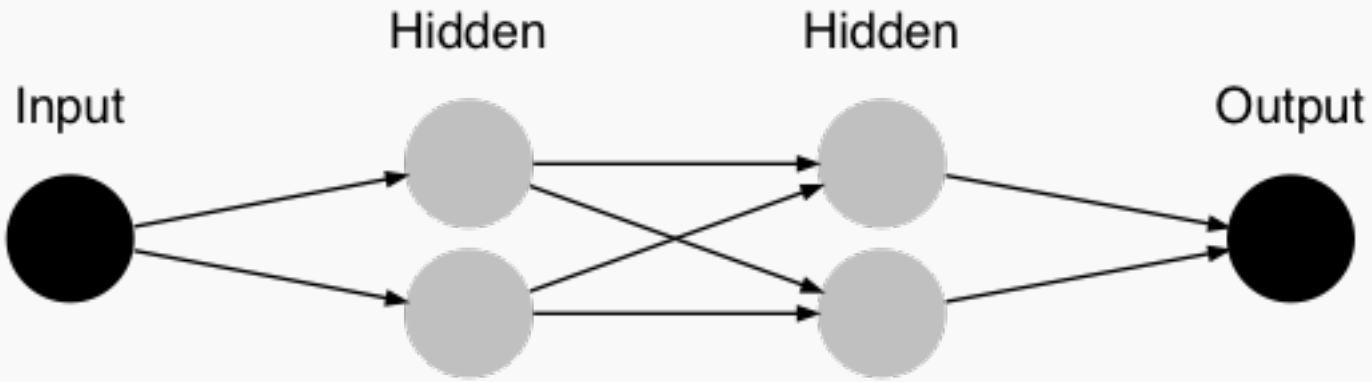


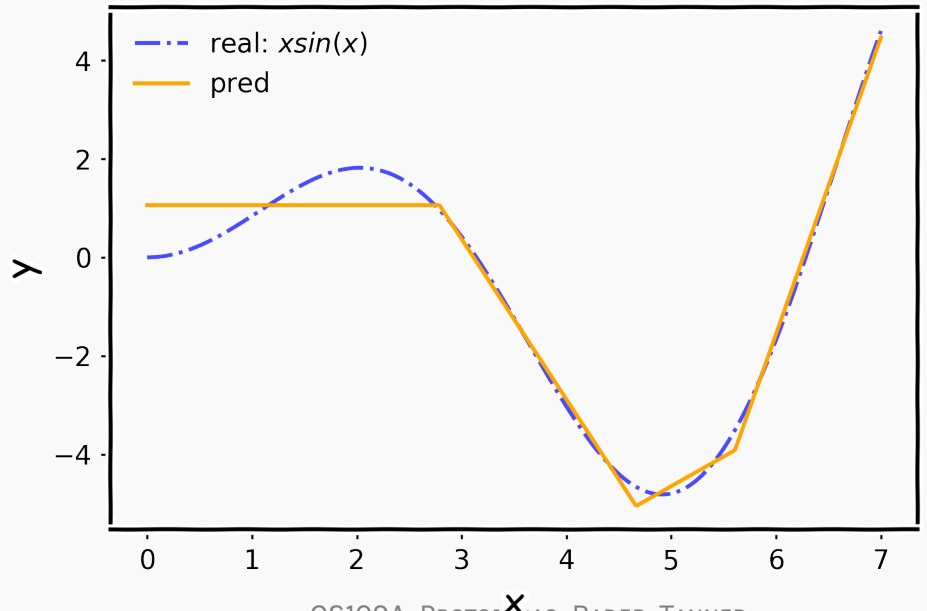
NN in action


Input
Hidden
Hidden
Output

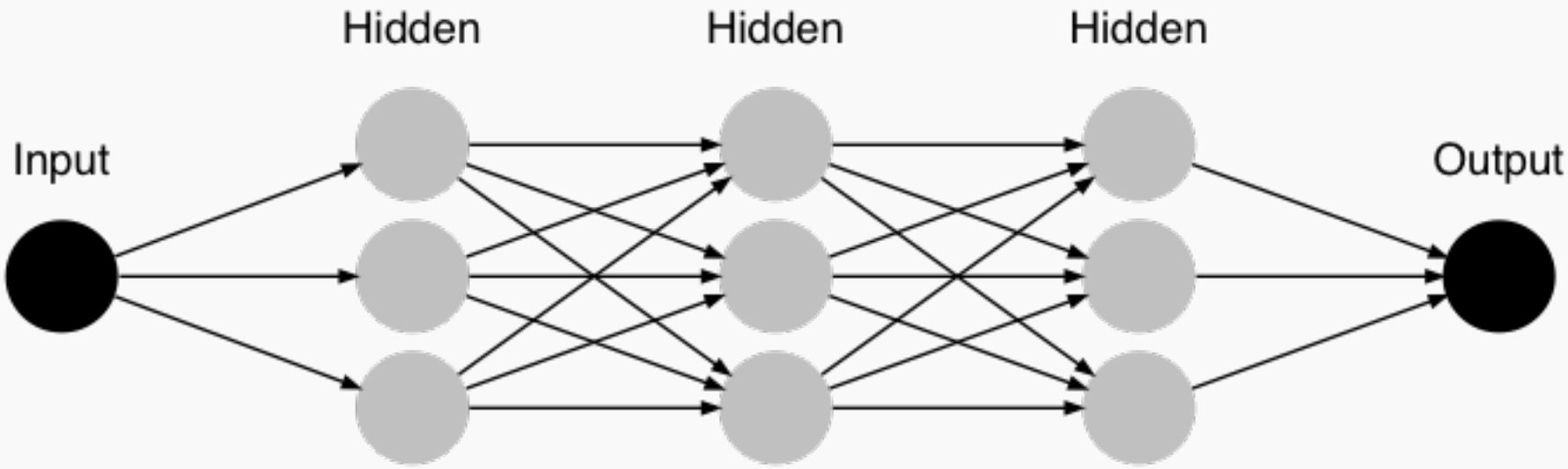

real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

NN in action

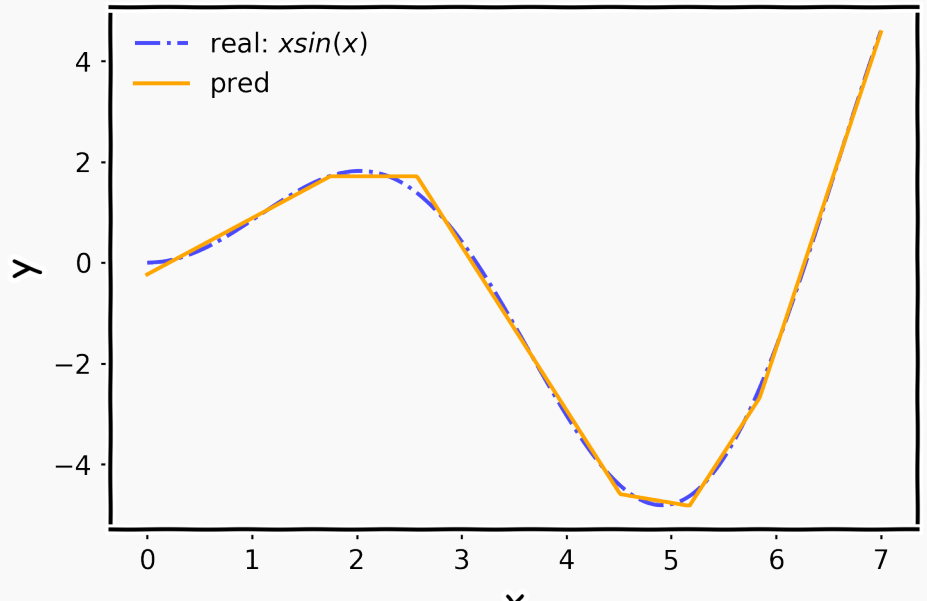

Input
Hidden
Hidden
Output


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

NN in action



Depth vs Width of network



Universal Approximation Theorem

Think of a Neural Network as function approximation.

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

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

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

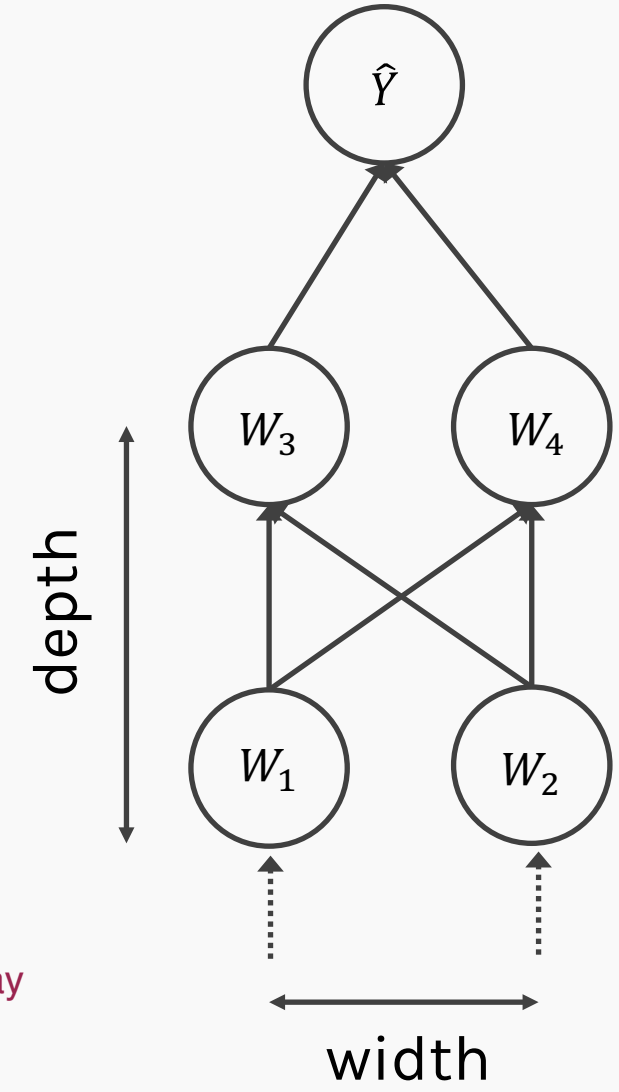
One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy

as you increase the number of nodes, you get better, better better

So why deeper?

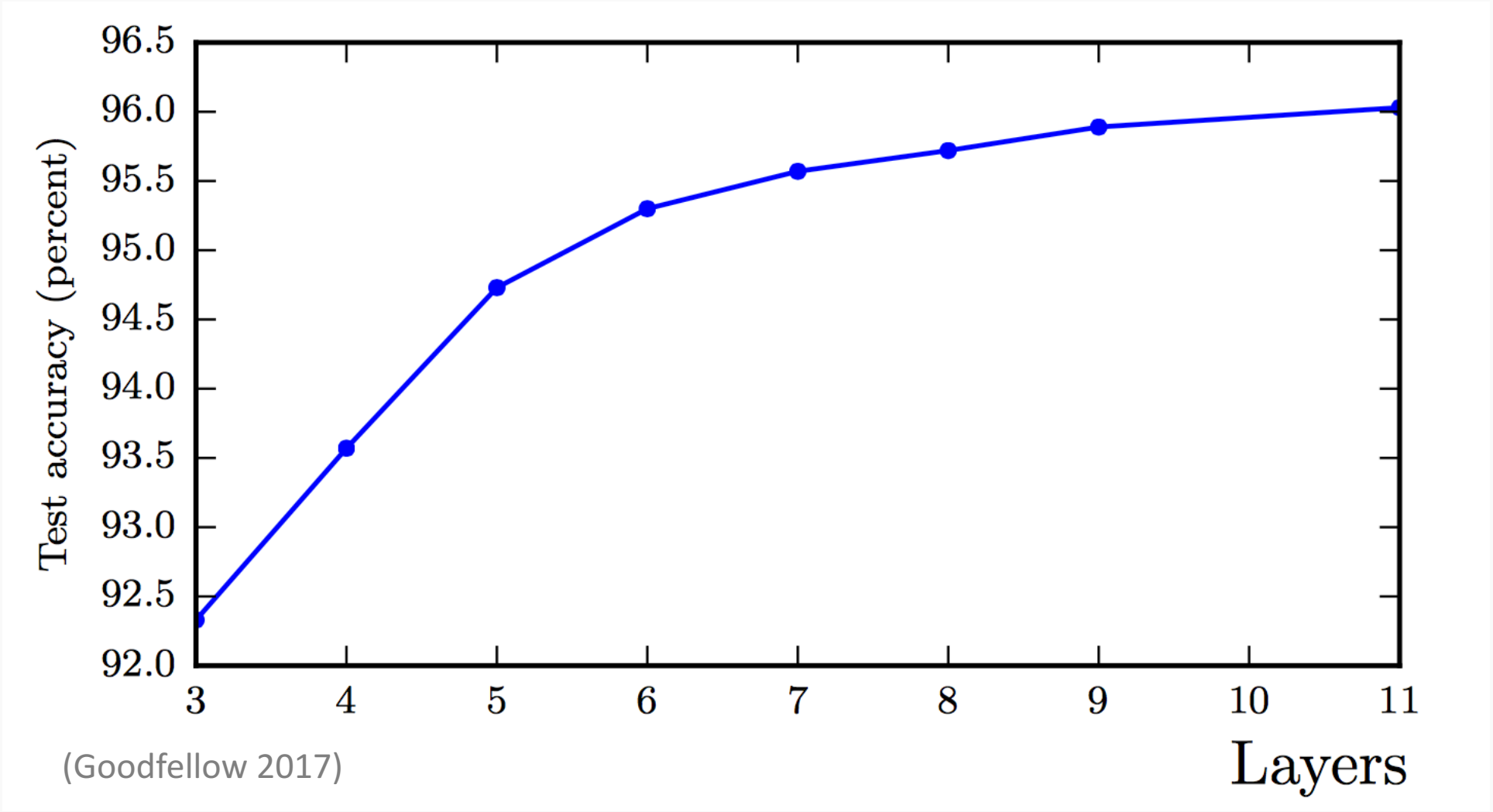
- Shallow net may need (exponentially) more width
- Shallow net may overfit more

with too many nodes and one layer, you may specialize each node on a little part of the data —> overfitting



Better Generalization with Depth

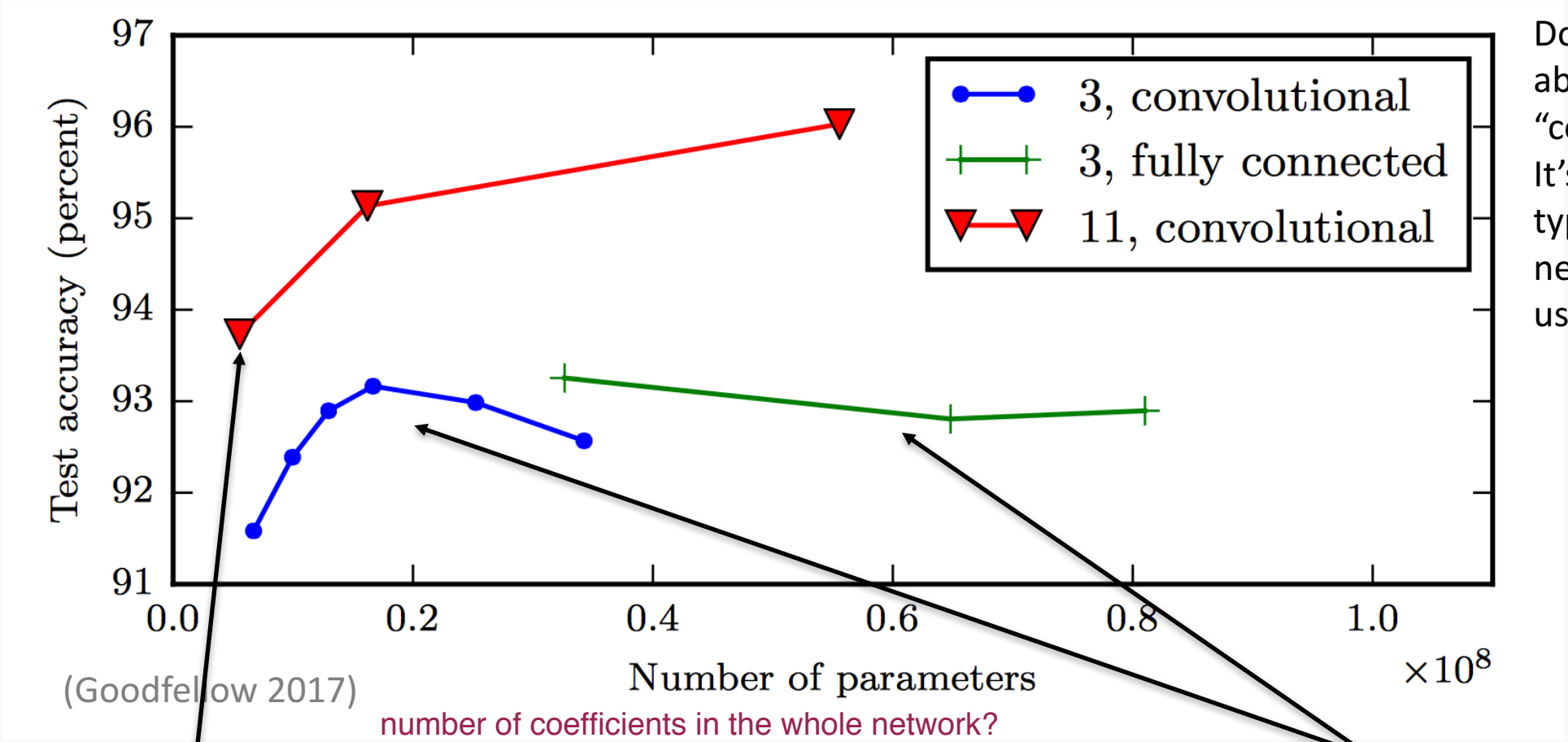
we prefer depth over width for generalization



(Goodfellow 2017)

Shallow Nets Overfit More

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



(Goodfellow 2017)

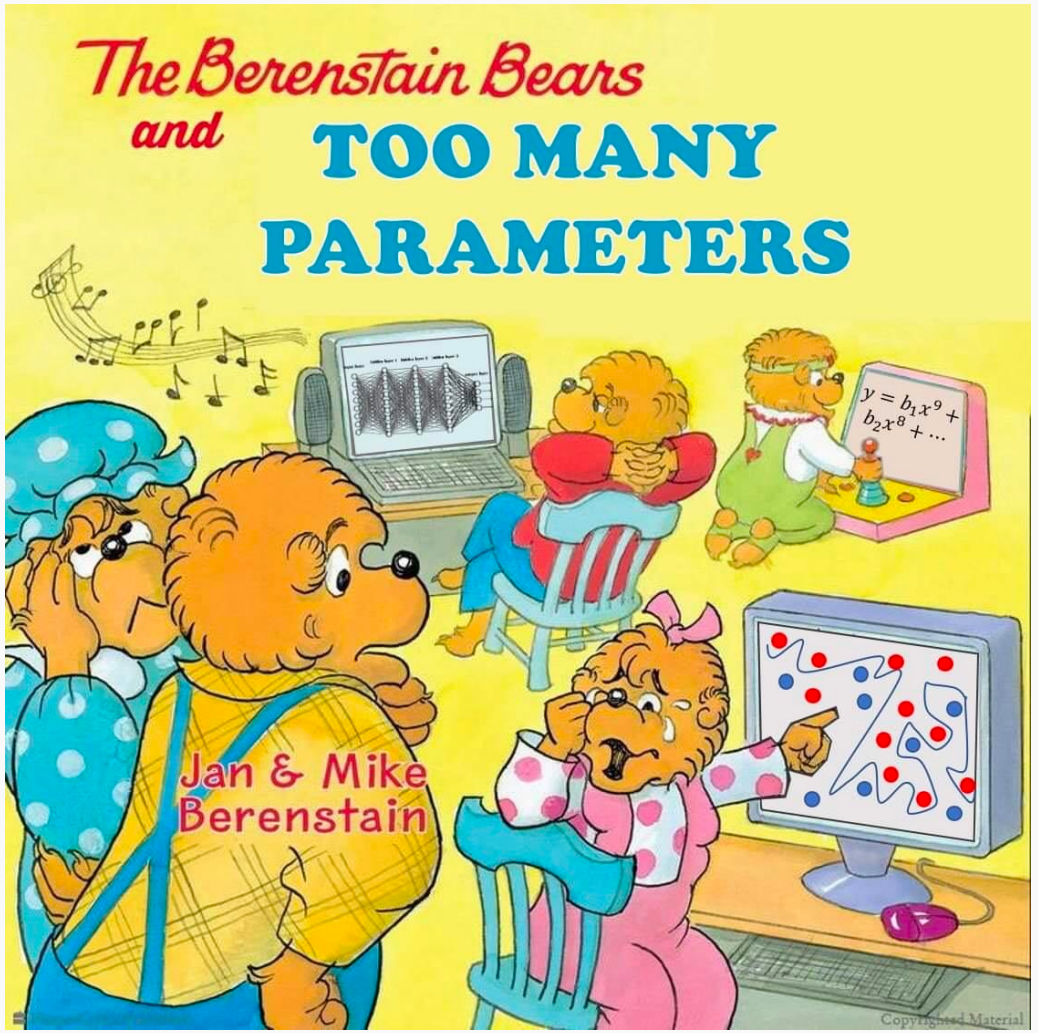
Don't worry about this word "convolutional". It's just a special type of neural network, often used for images.

number of coefficients in the whole network?

proportional to complexity of the network; how much you have to learn

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.