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

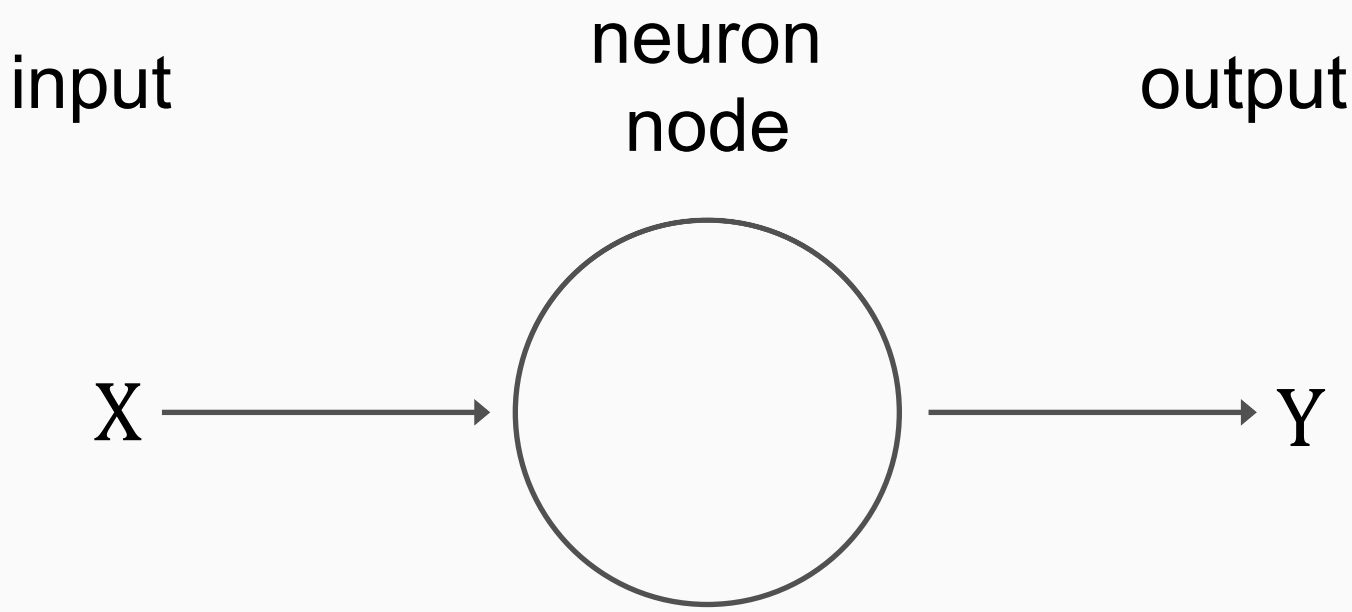
Outline

Anatomy of a NN

Design choices

- Activation function
- Loss function
- Output units
- Architecture

Anatomy of artificial neural network (ANN)

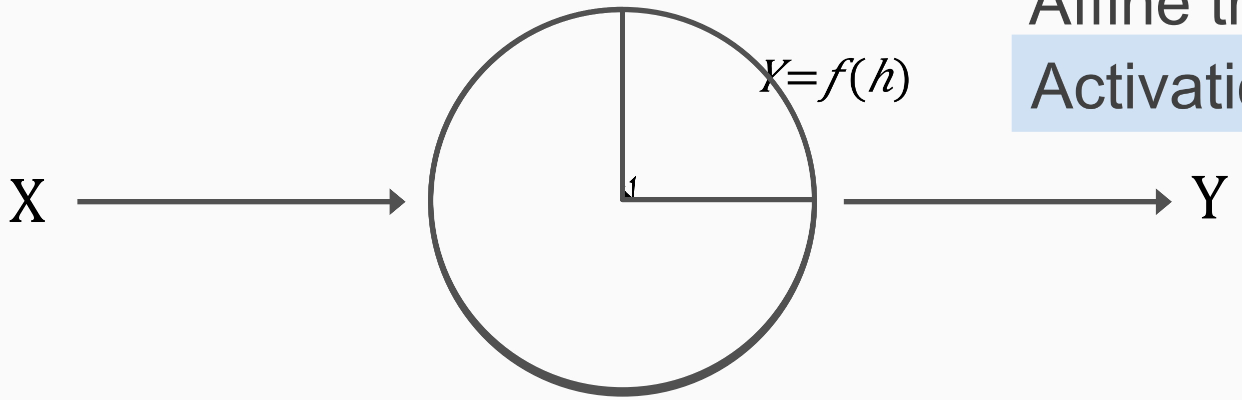


Anatomy of artificial neural network (ANN)

input

neuron
node

output



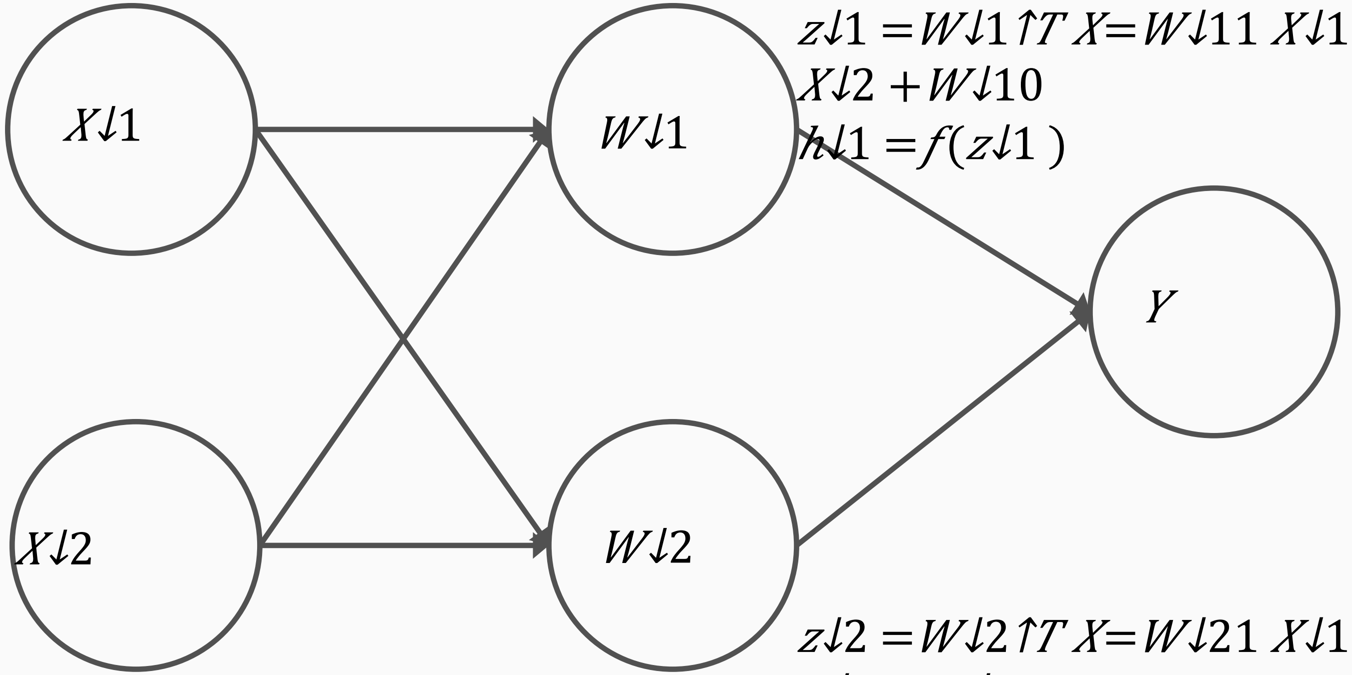
Affine transformation

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)

Input layer   hidden layer   output layer



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

$h_1 = f(z_1)$

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

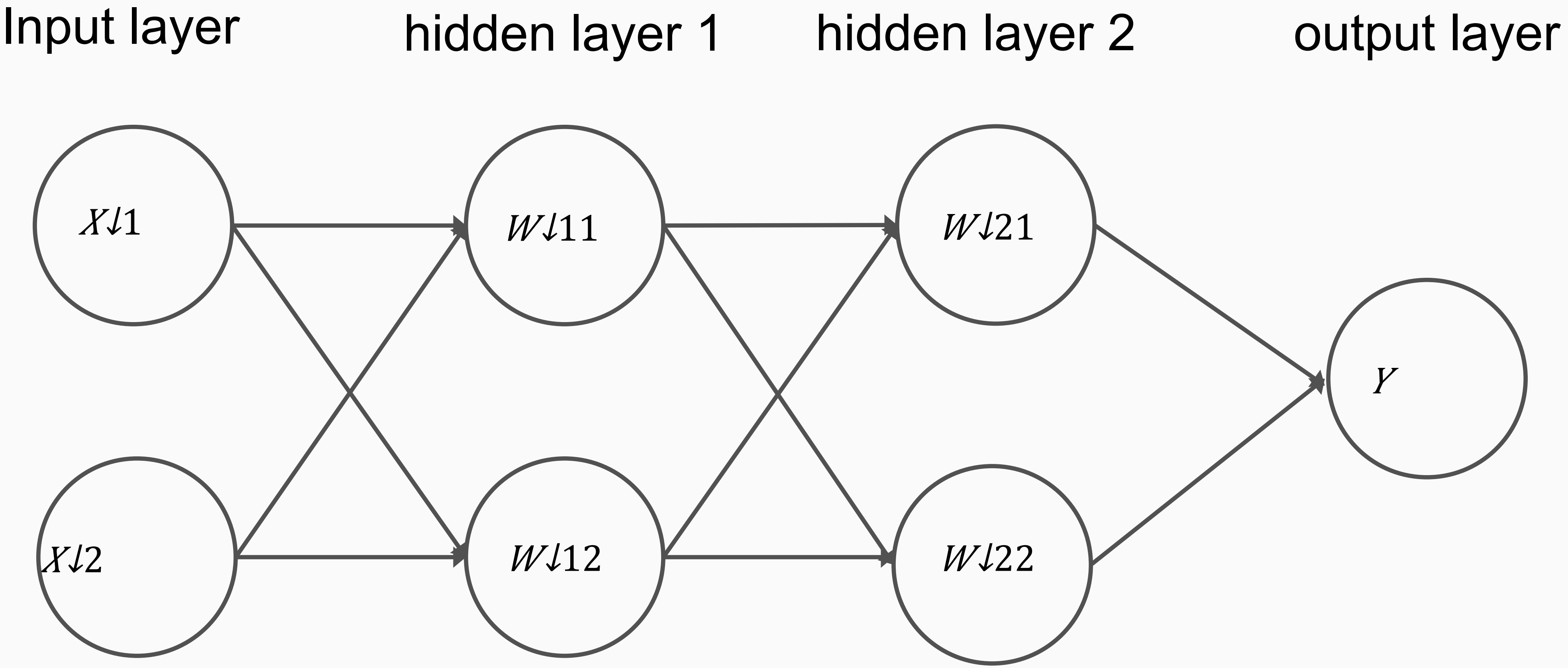
$\mathrm{h}_2 = f(z_2)$

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

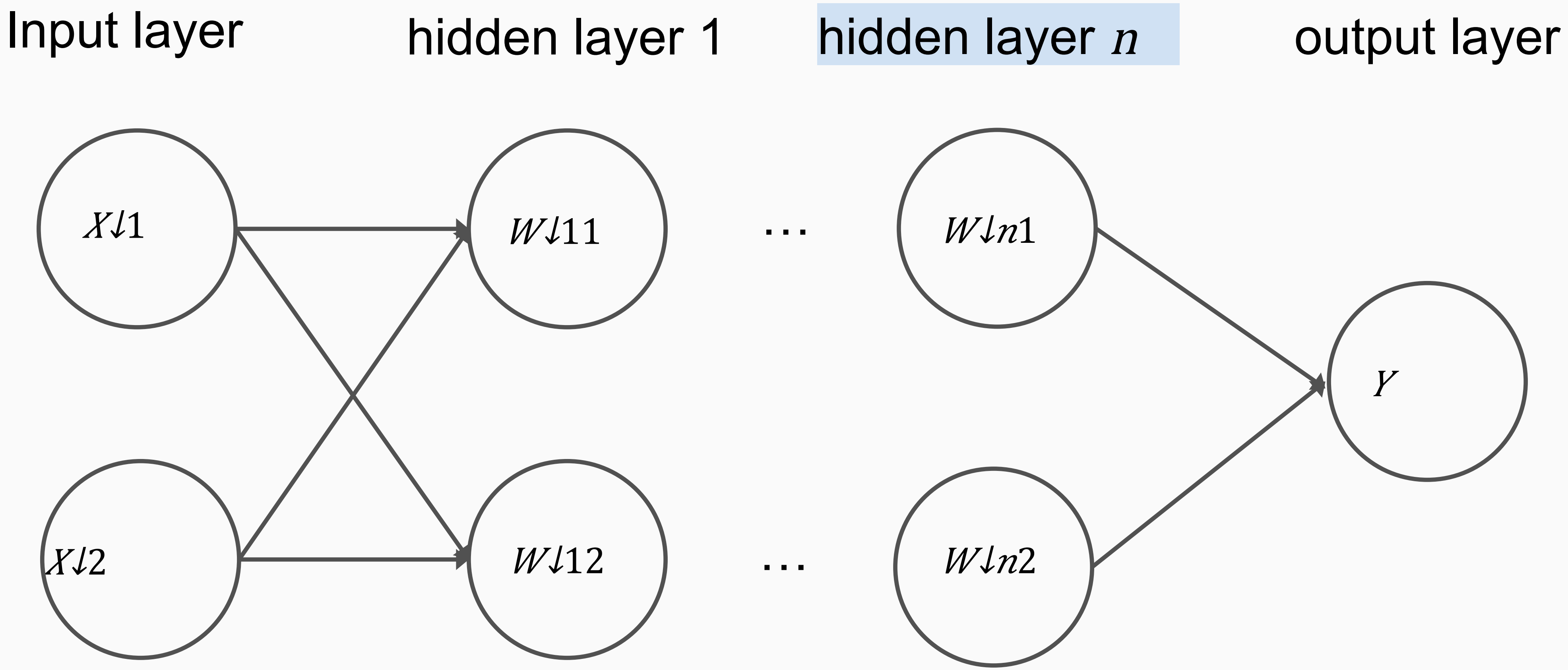
Output function   Loss function

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.

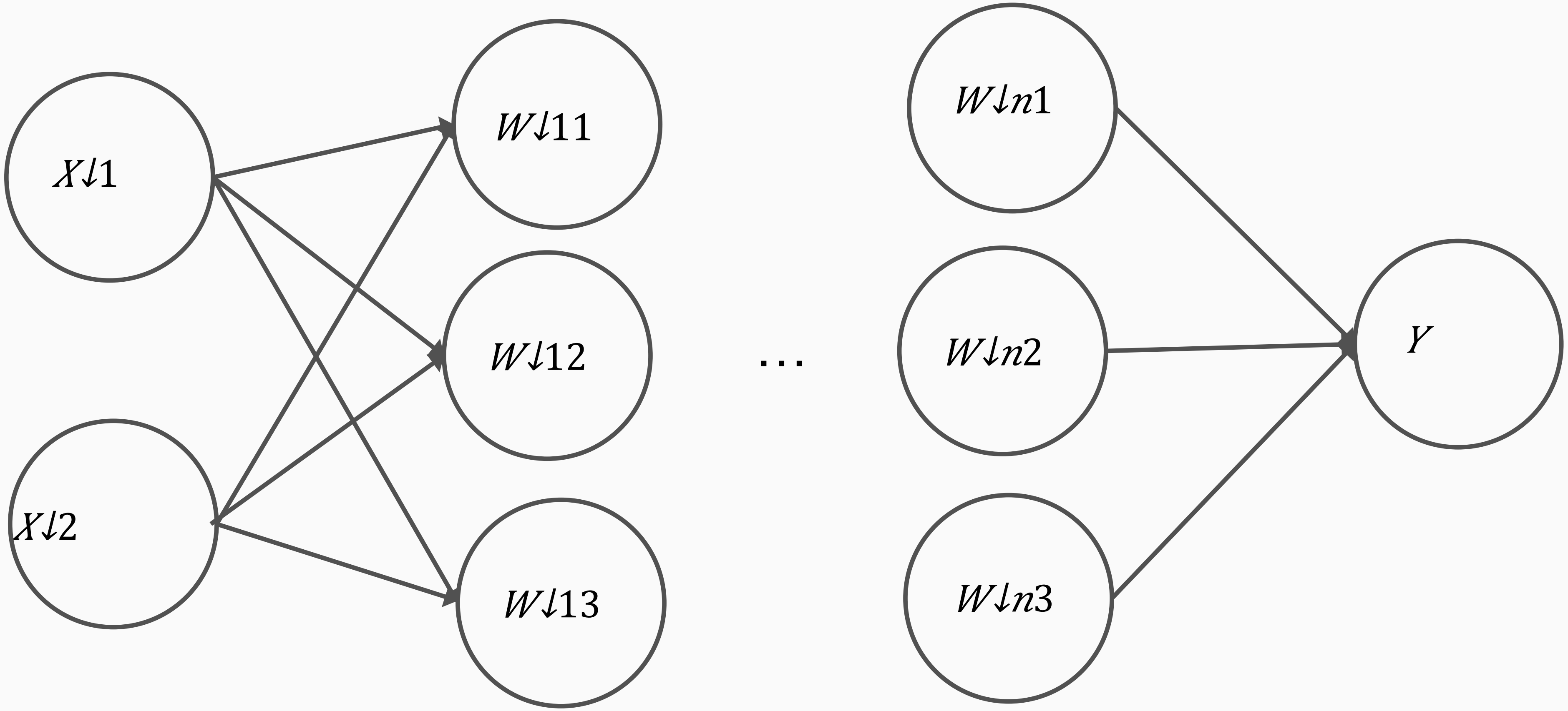
Anatomy of artificial neural network (ANN)

Input layer

hidden layer 1, 3 nodes

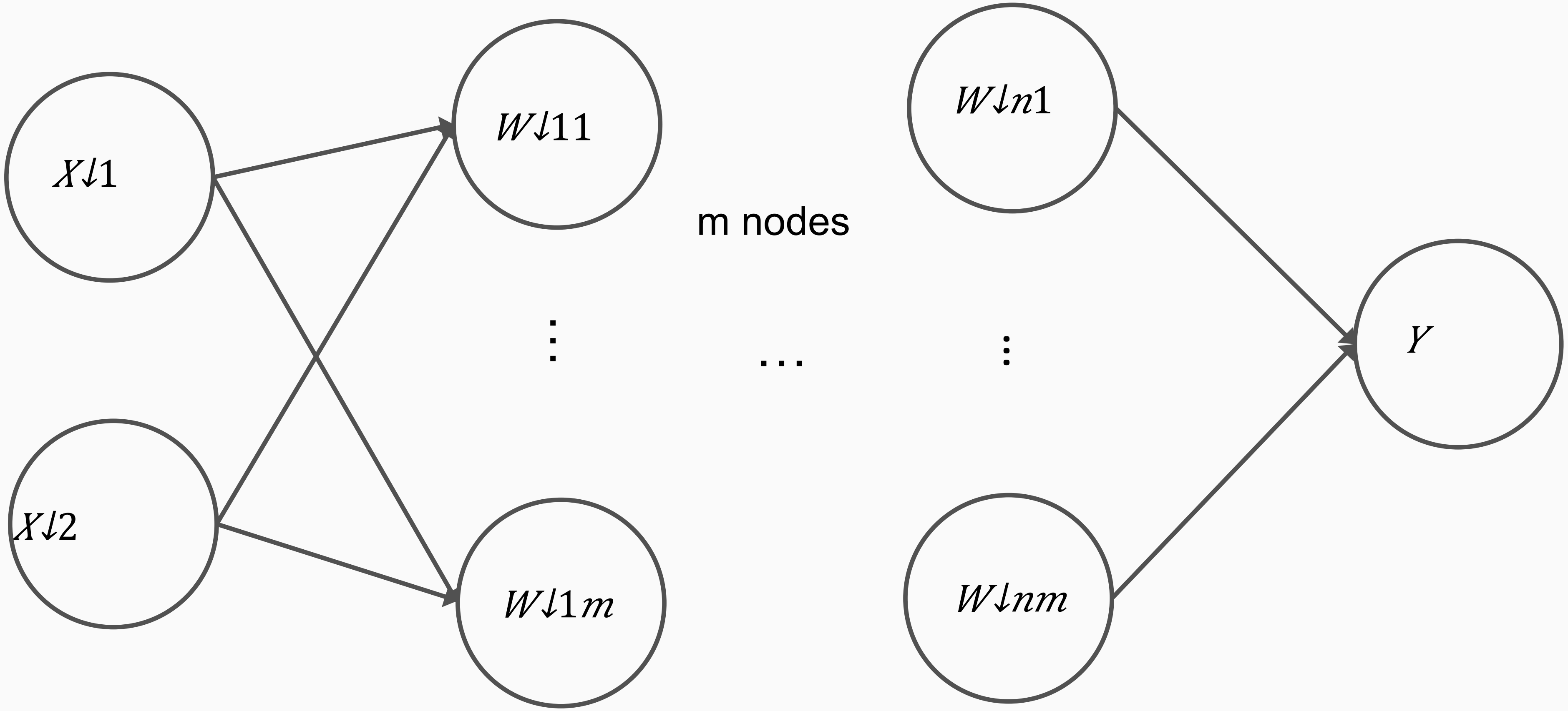
hidden layer n *3 nodes*

output layer



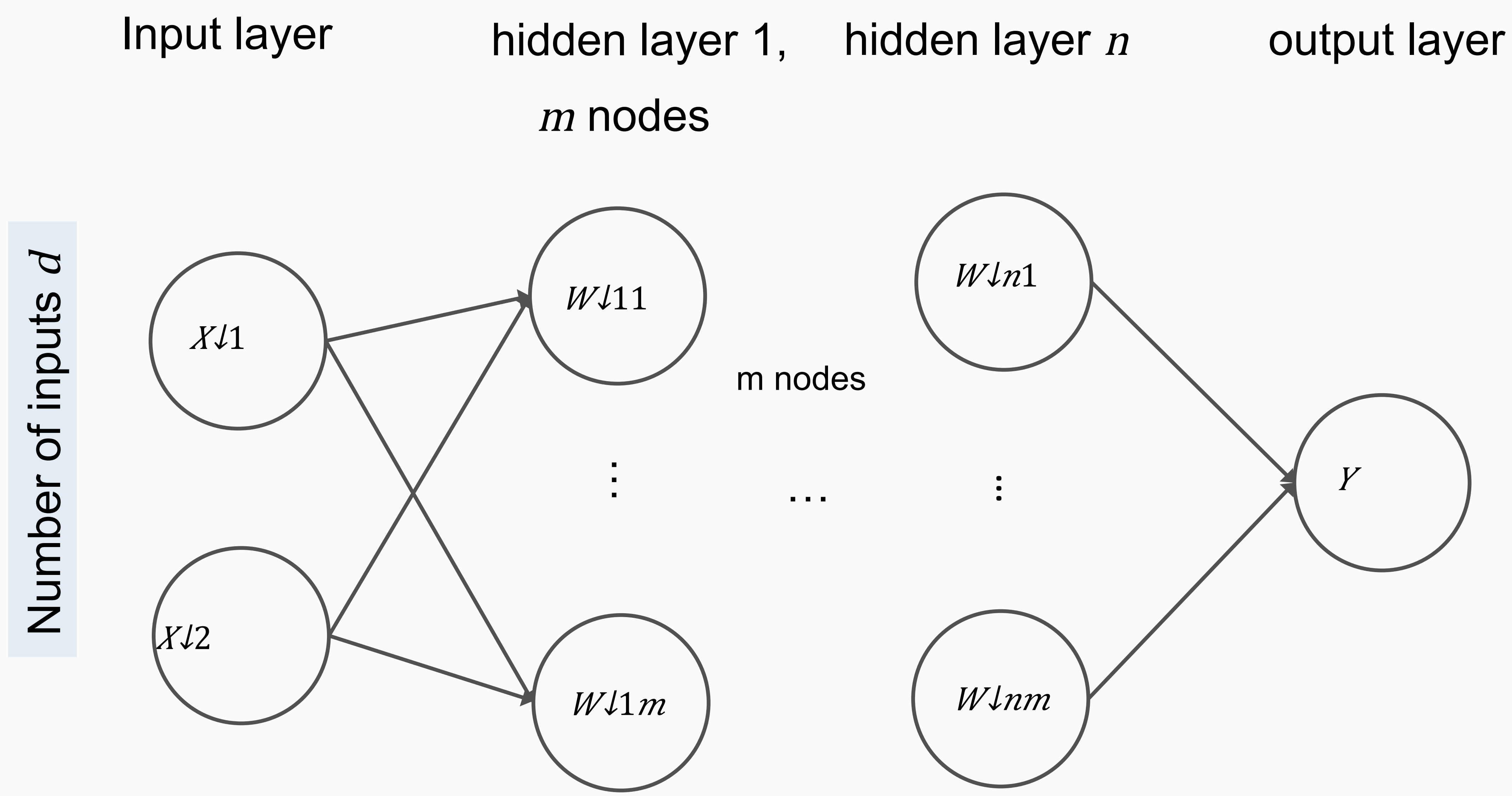
Anatomy of artificial neural network (ANN)

Input layer | hidden layer 1, m nodes | hidden layer n | output layer



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

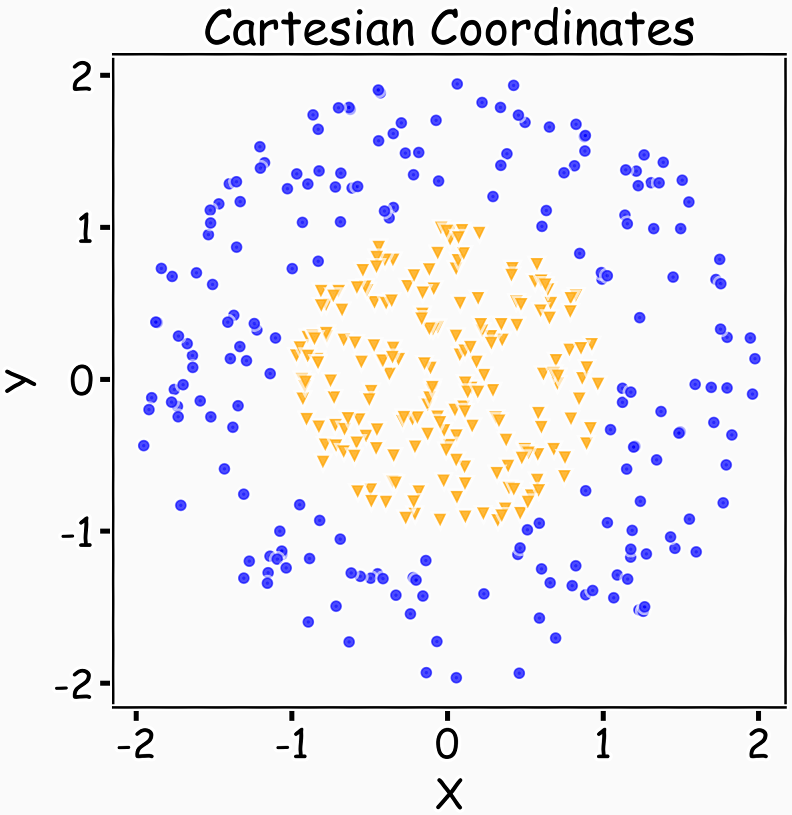
Anatomy of artificial neural network (ANN)

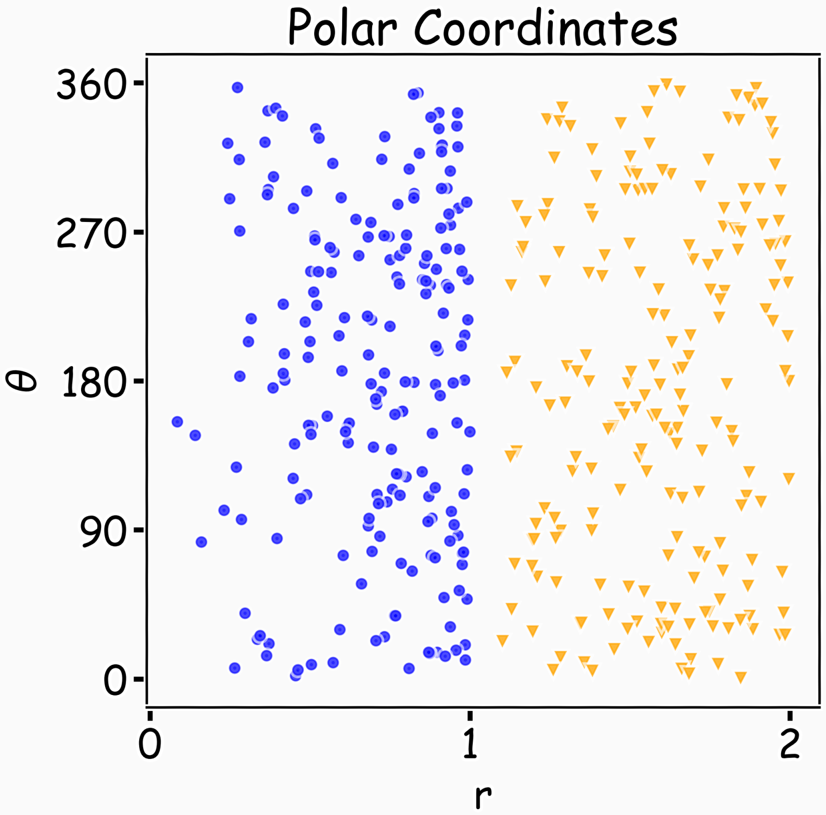


Number of inputs is specified by the data

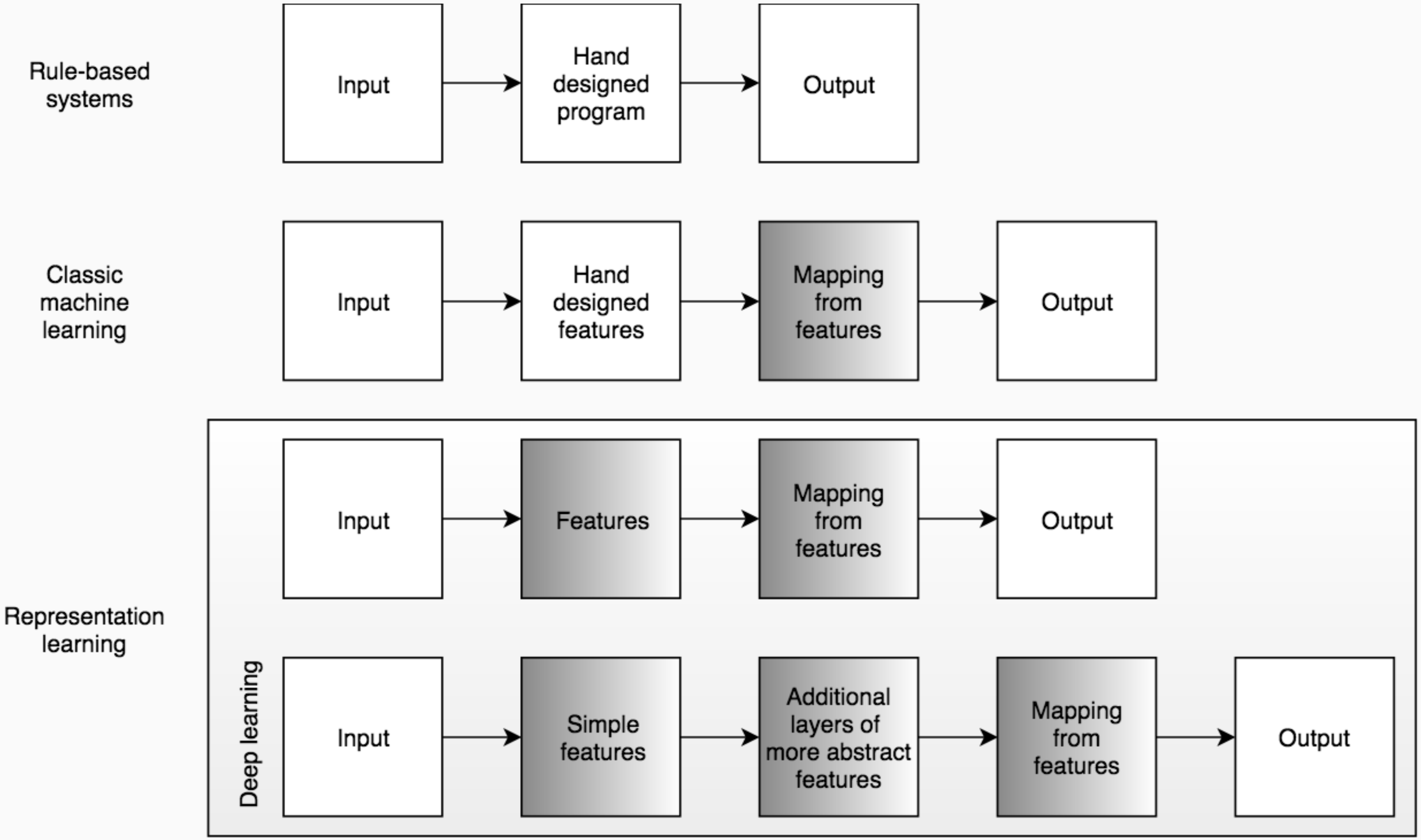
Why layers? Representation

Representation Matters

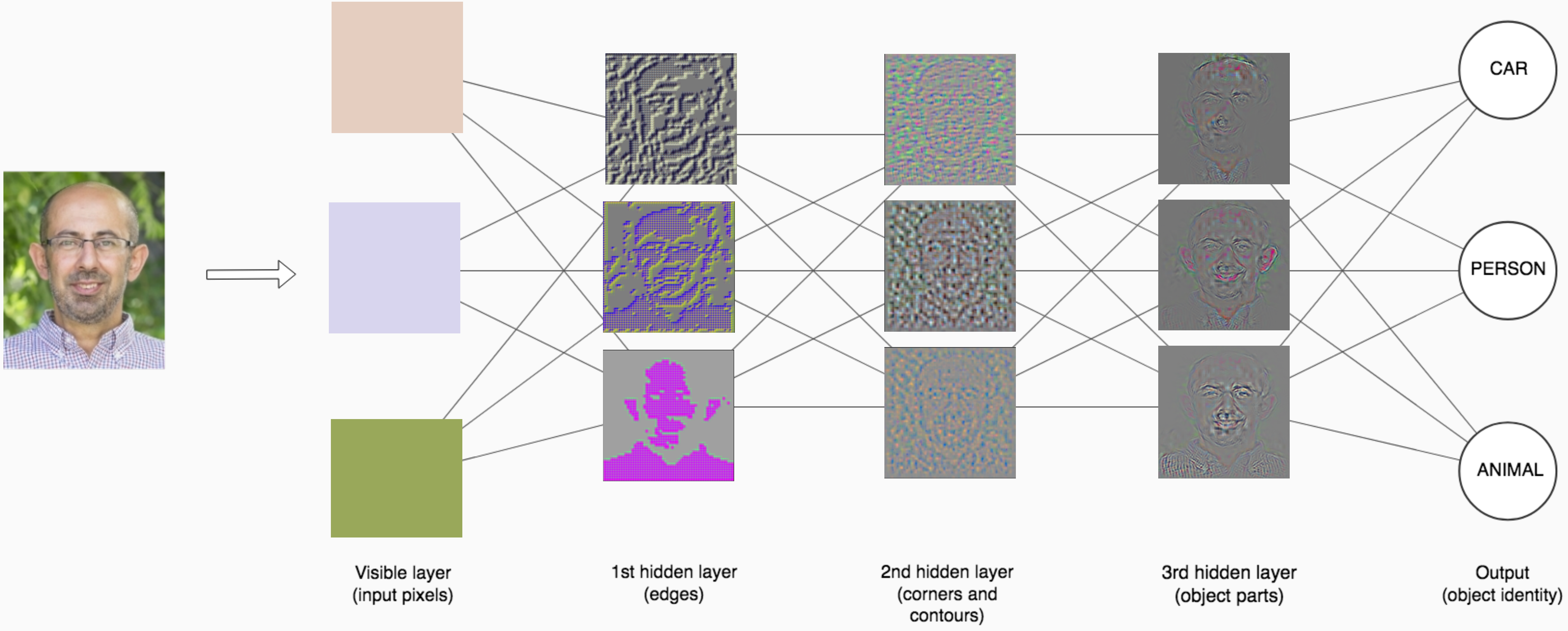




Learning Multiple Components



Depth = Repeated Compositions



Neural Networks

Hand-written digit recognition: MNIST data

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 not 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 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

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

- where θ are parameters of the transform
- ϕ defines hidden layers
- Non-convex optimization
- Can encode prior beliefs, generalizes well

Activation function

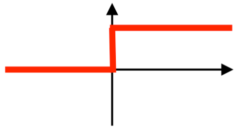
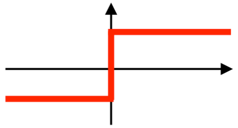
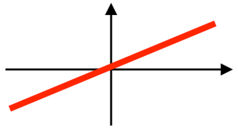
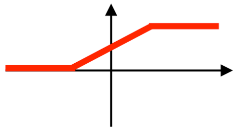
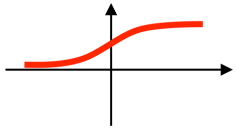
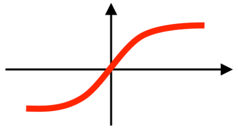
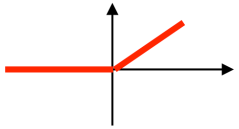
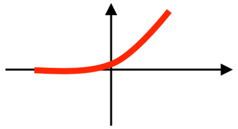
$$h = f(W^T X + b)$$

The activation function should:

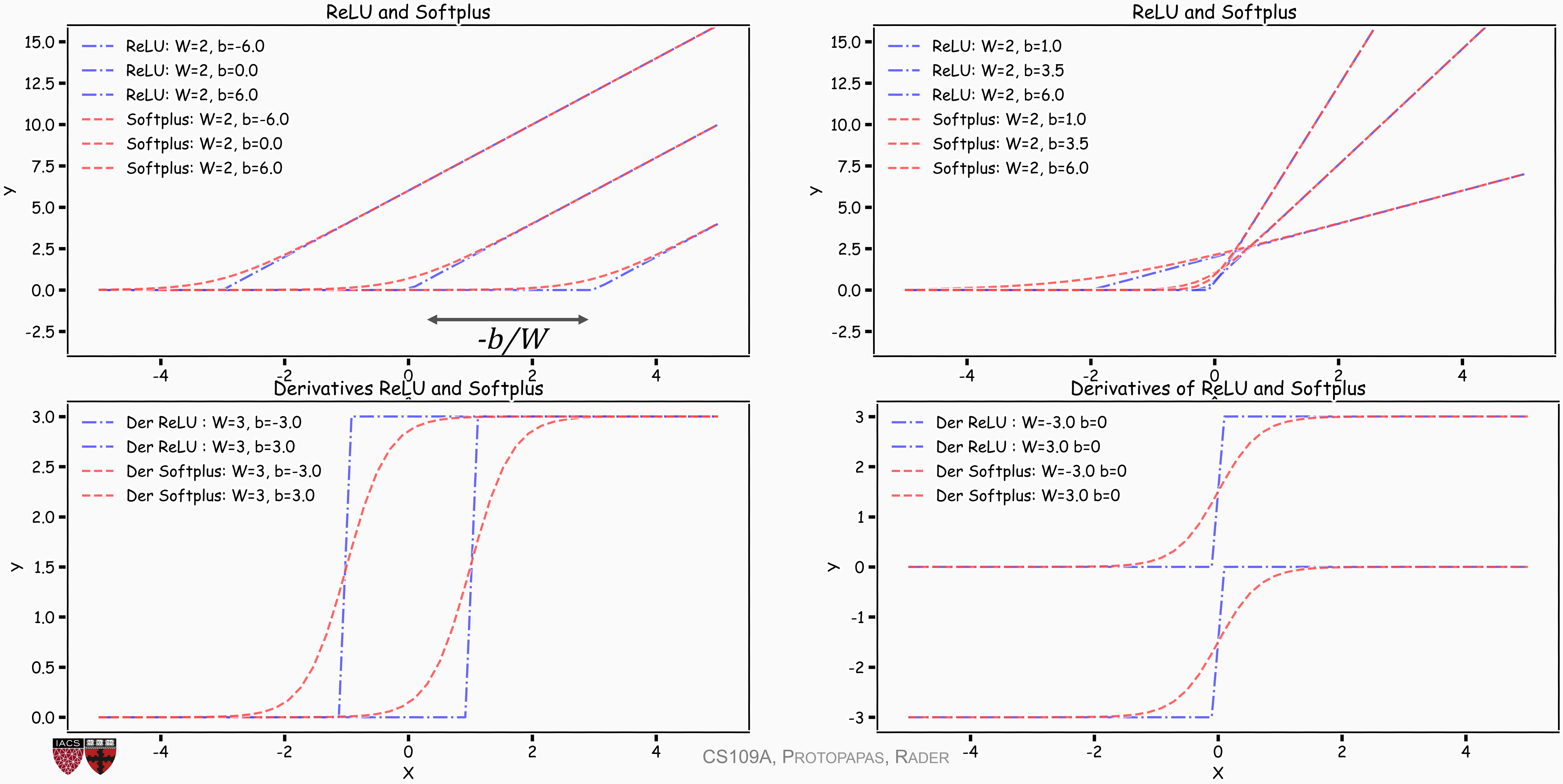
- Ensures **not linearity**
- Ensure gradients remain large through hidden unit

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

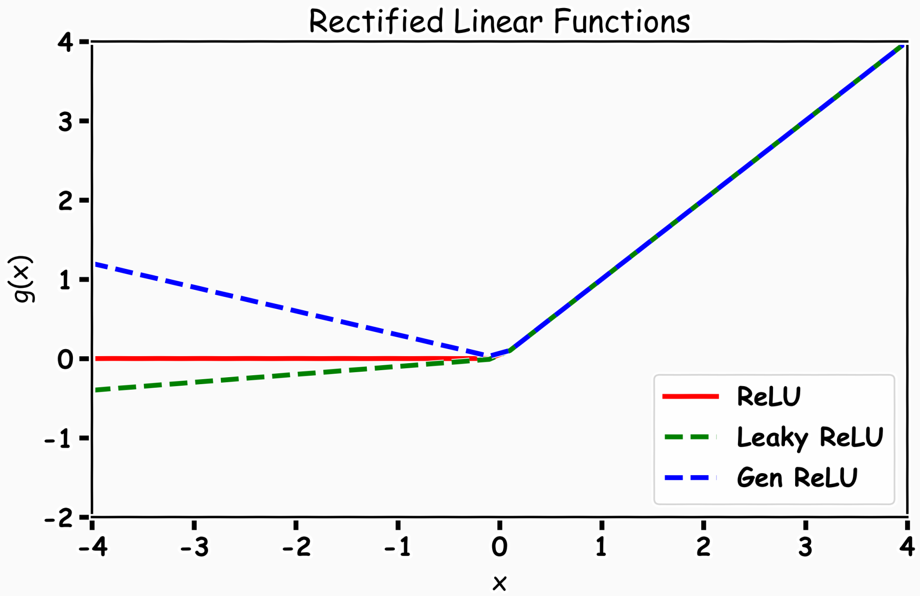


Generalized ReLU

Generalization: For $\alpha_i > 0$

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

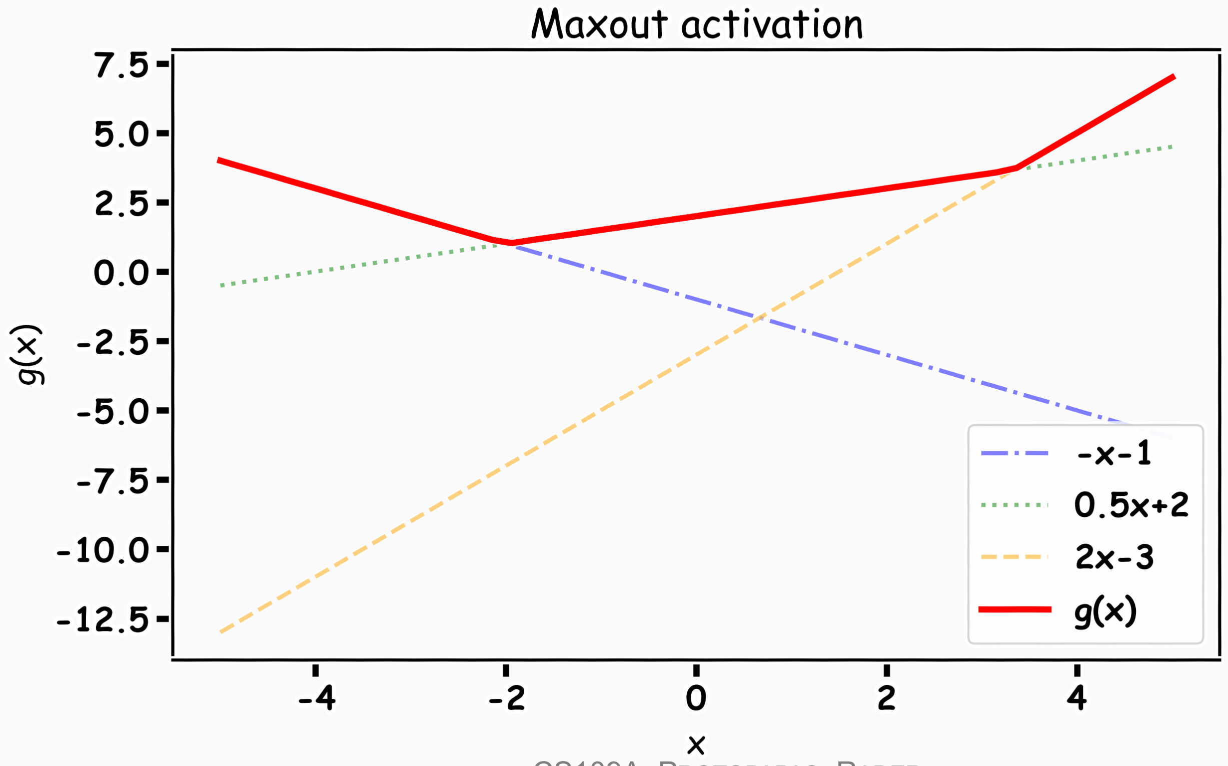
E.g. Absolute value ReLU: $\alpha_i = -1 \;\Rightarrow\; g(z) = |z|$



Maxout

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

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



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.

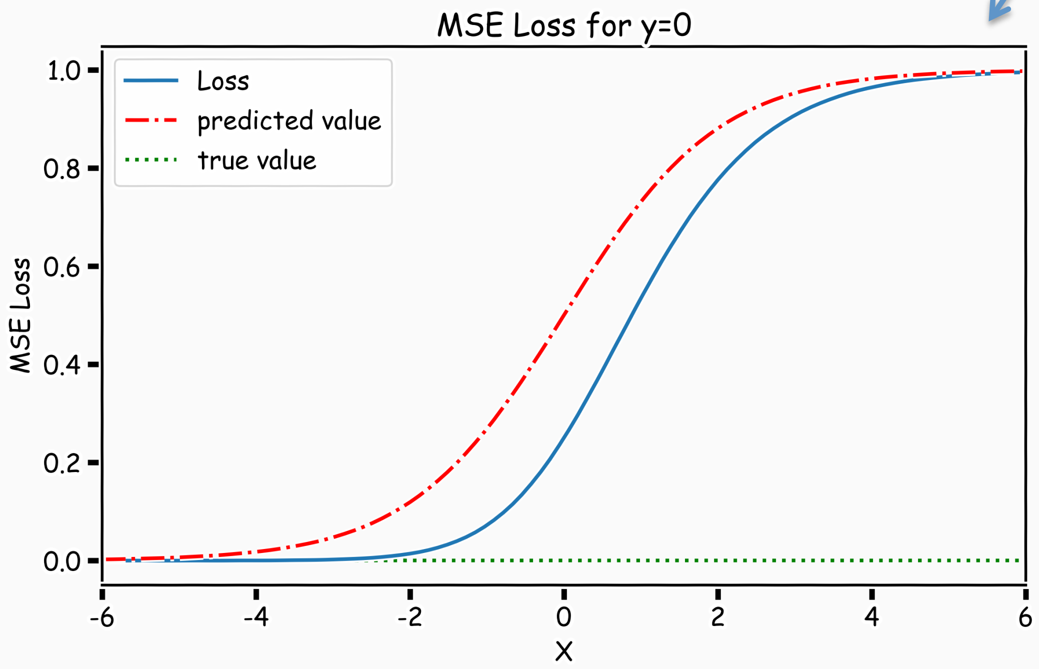
Gradient of cost function must be large enough

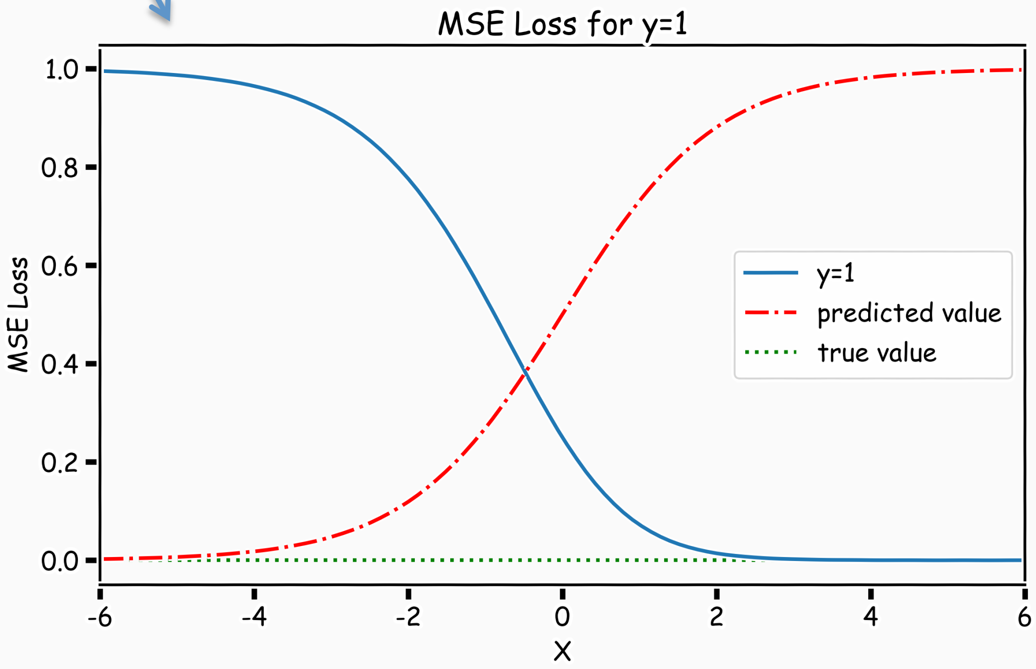
Loss Function

Example: sigmoid output + squared loss

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

Flat surfaces

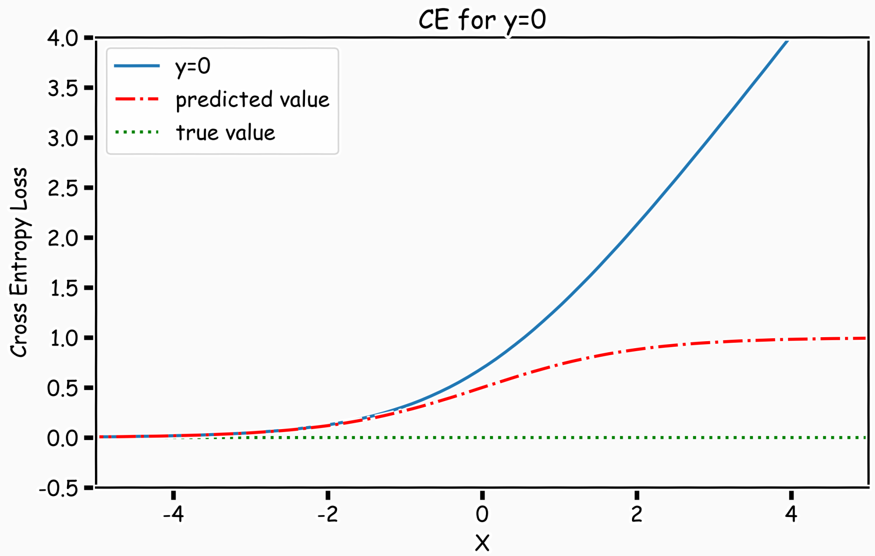


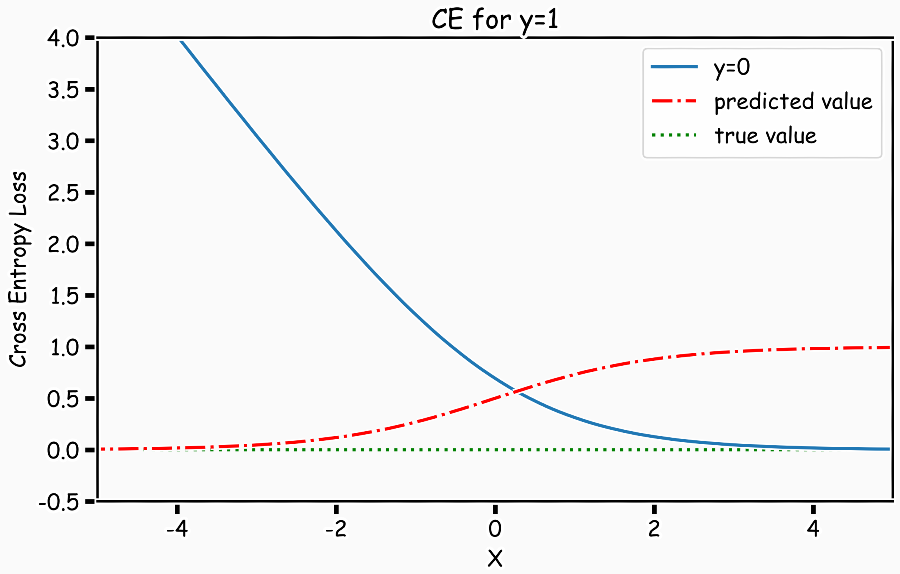


Cost Function

Example: sigmoid output + cross-entropy loss

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





Design Choices

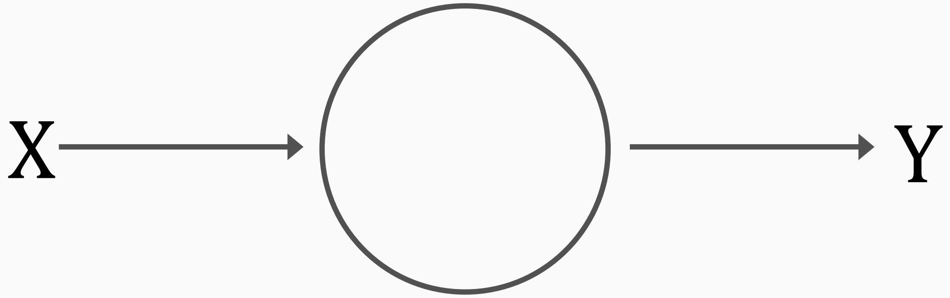
- Activation function
- Loss function
- Output units
- Architecture
- Optimizer

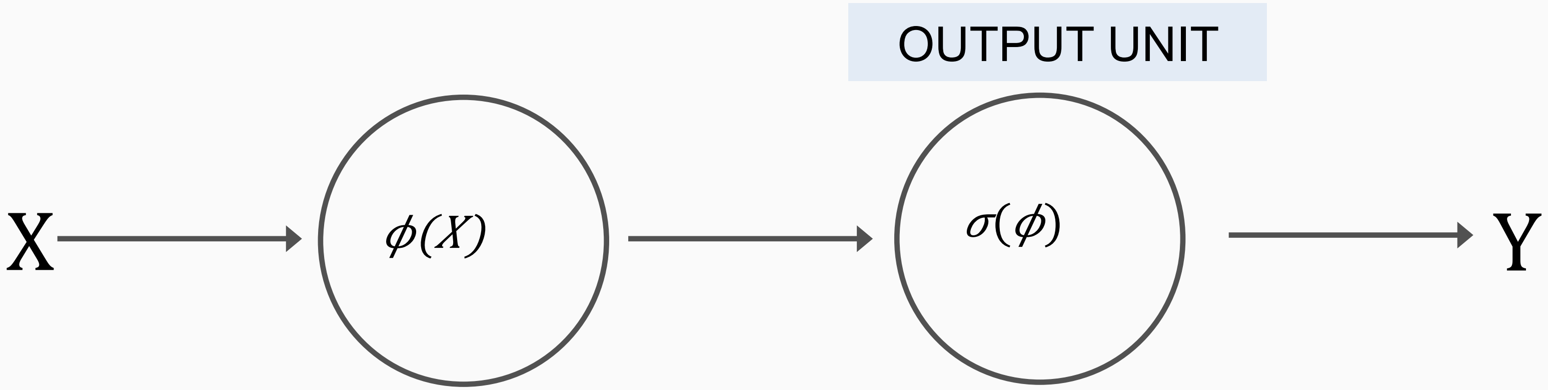
Output Units

Output Type	Output Distribution	Output layer	Cost Function
Binary			

Link function

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





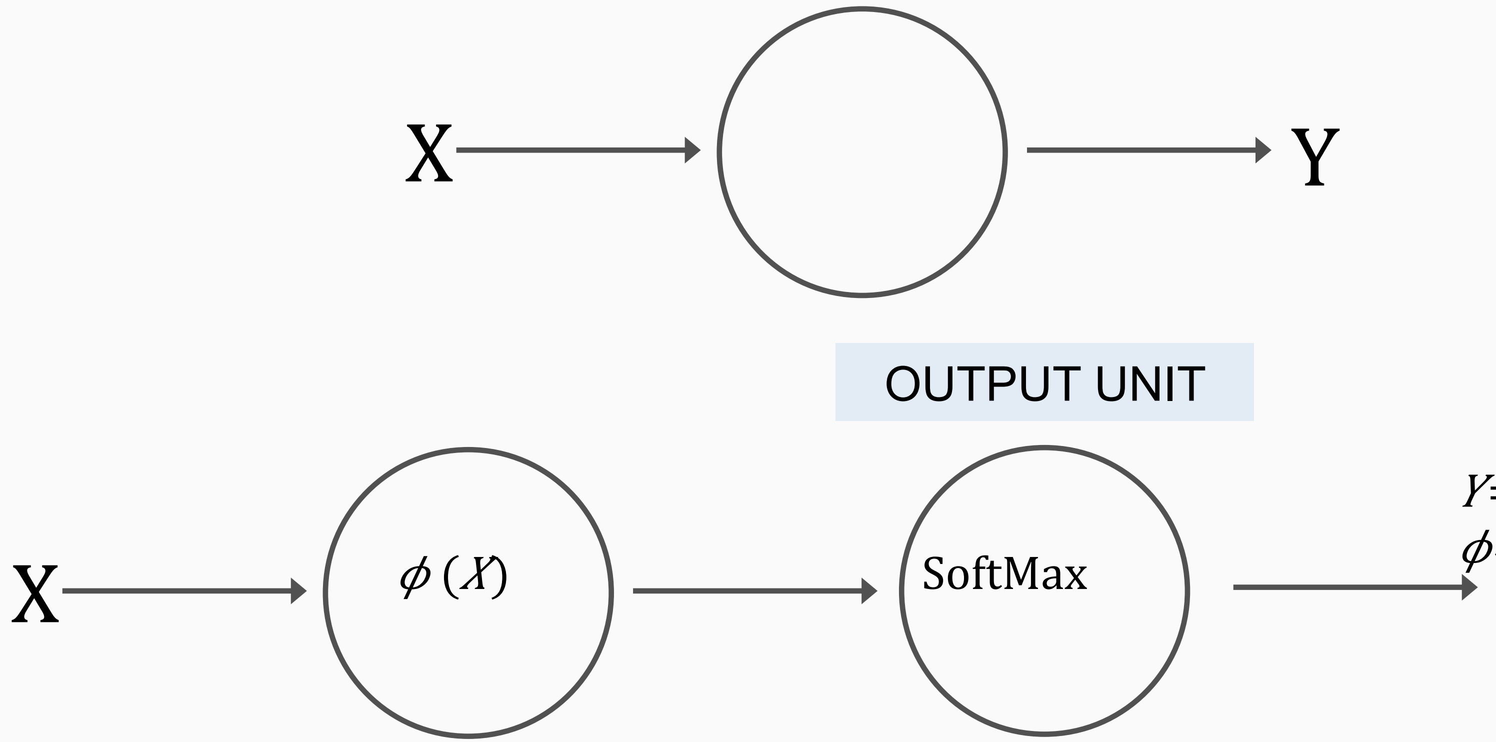
Output Units

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



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

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

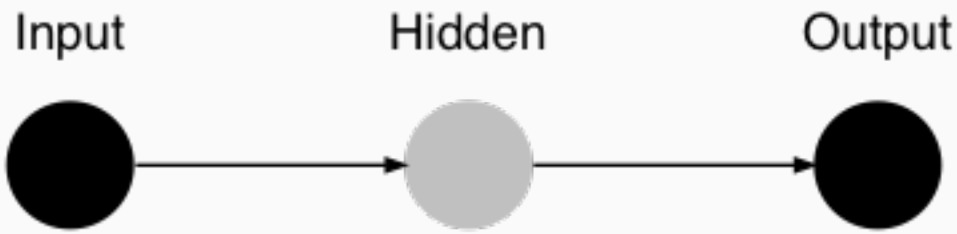
Loss function

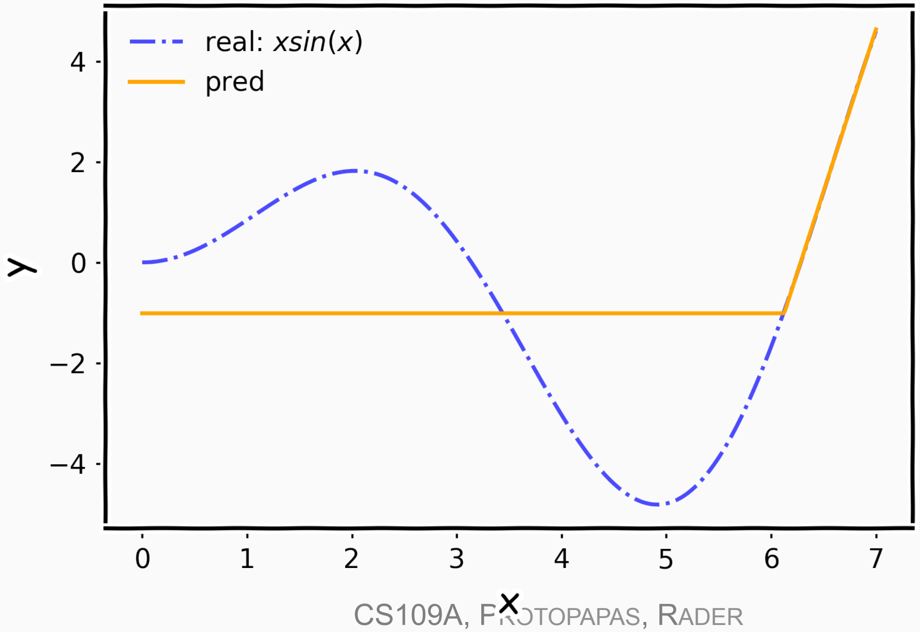
Output units

Architecture

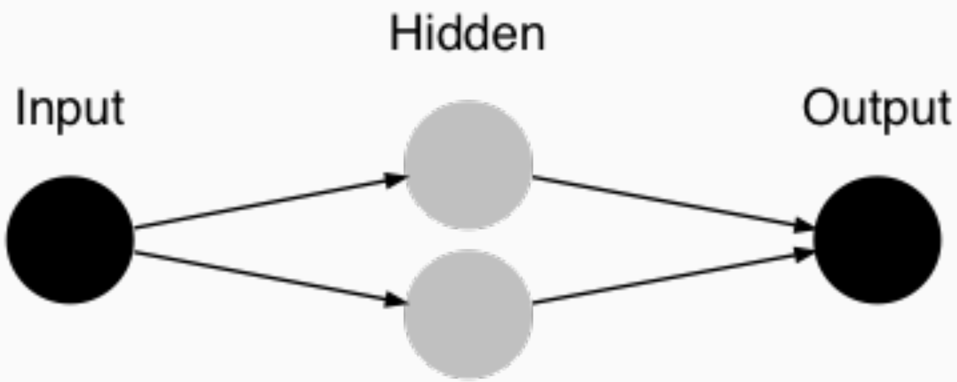
Optimizer

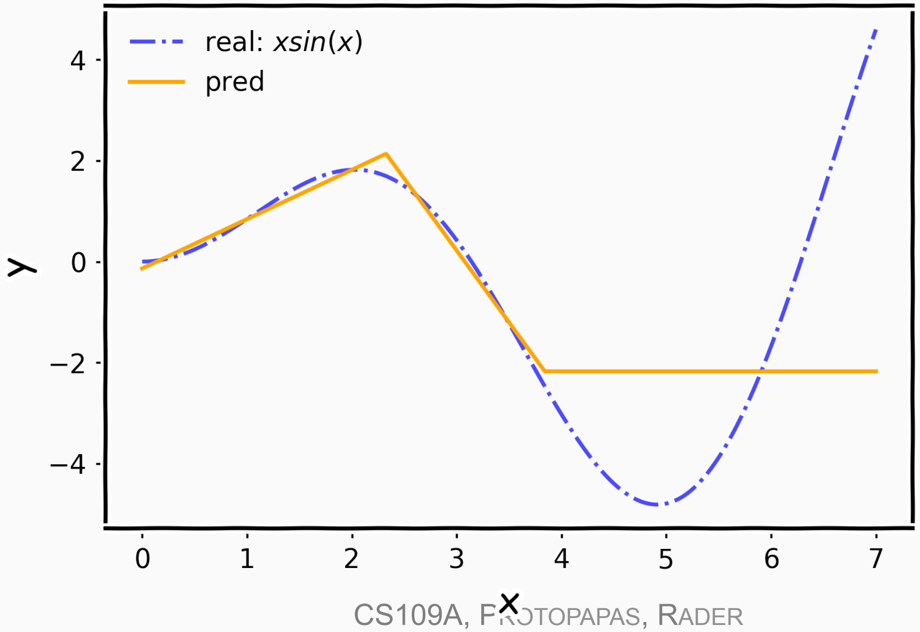
NN in action



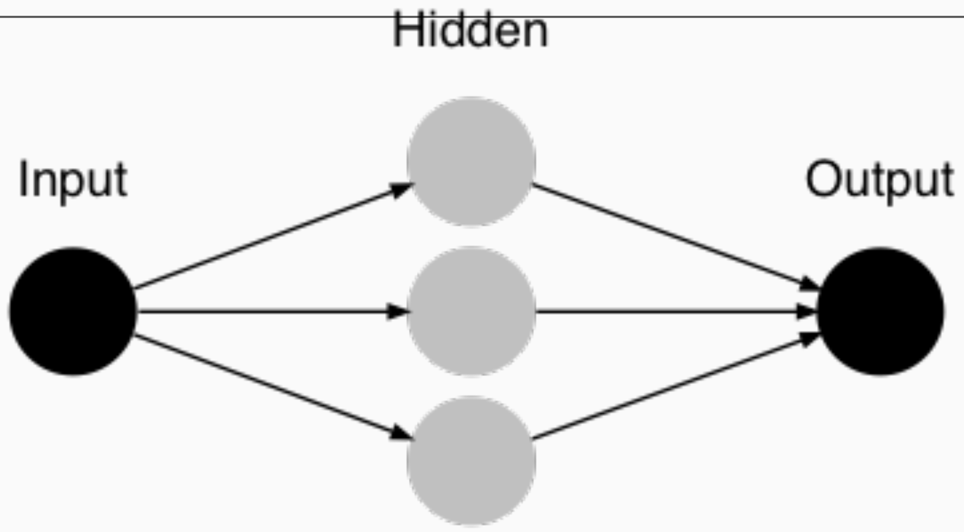


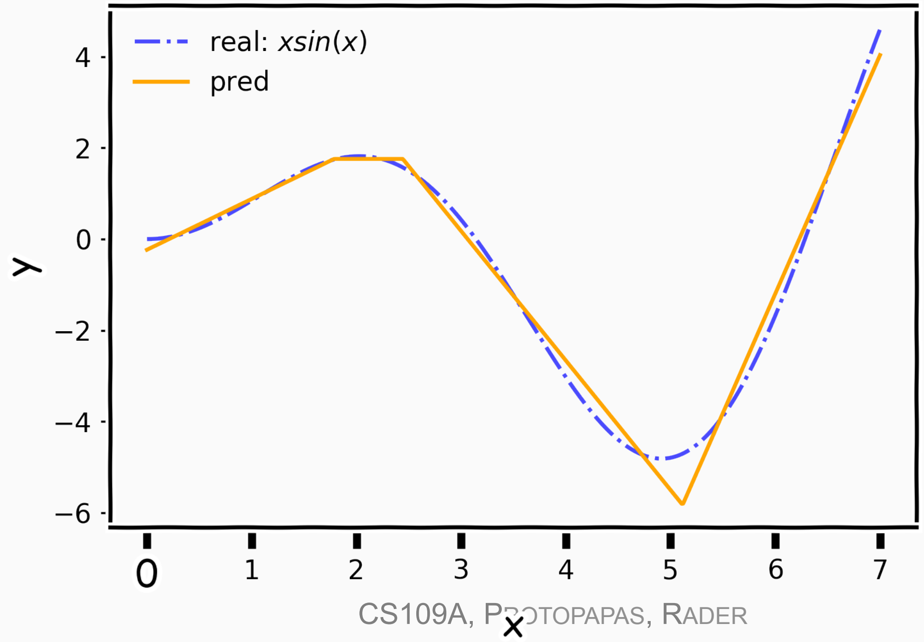
NN in action



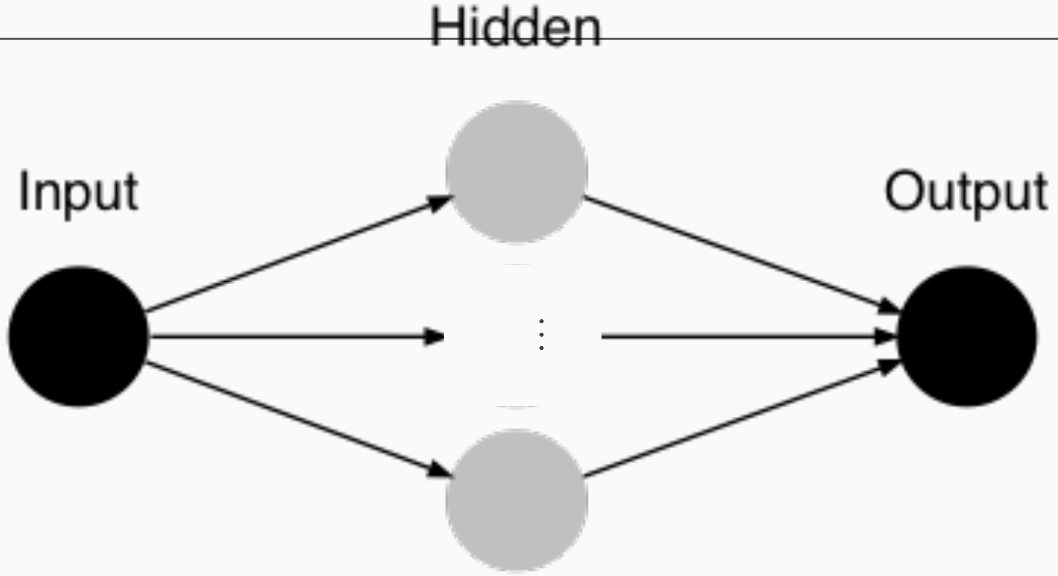


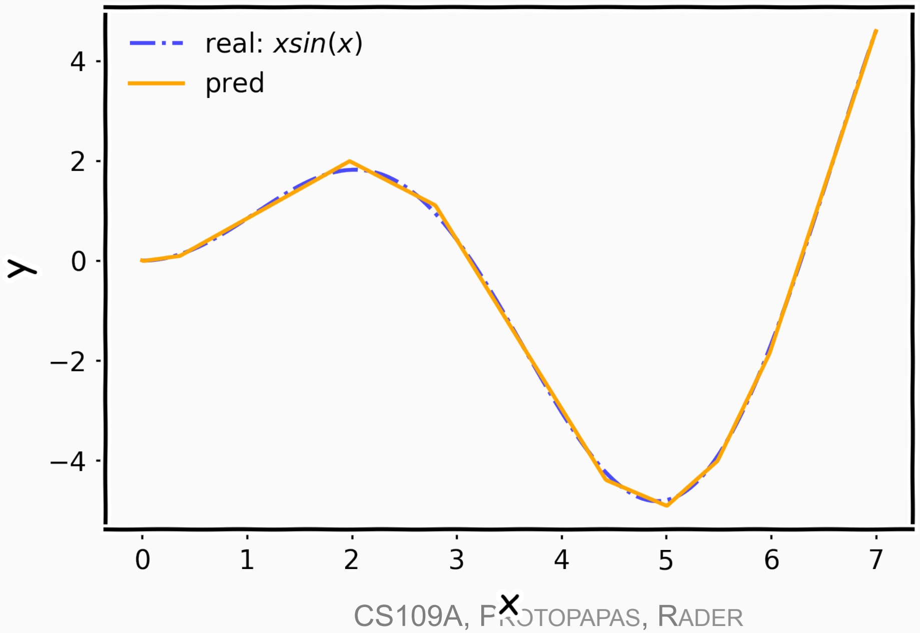
NN in action



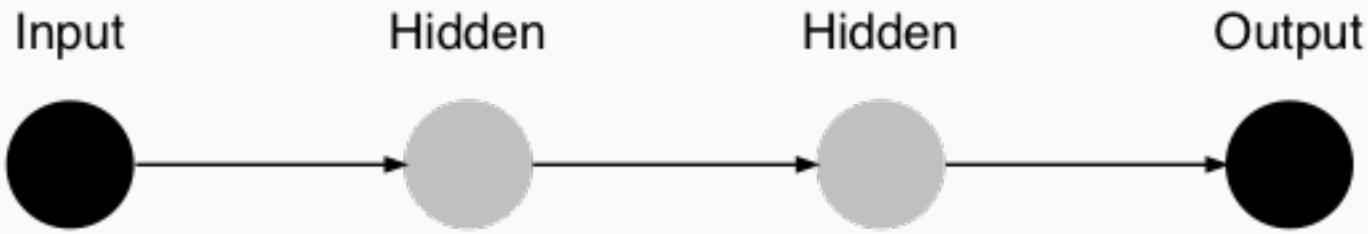


NN in action

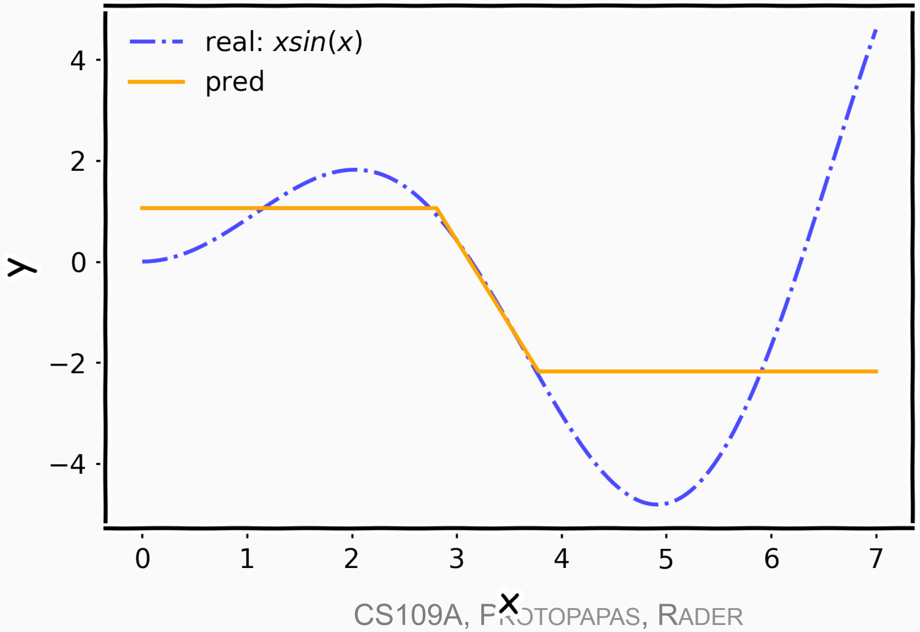




NN in action

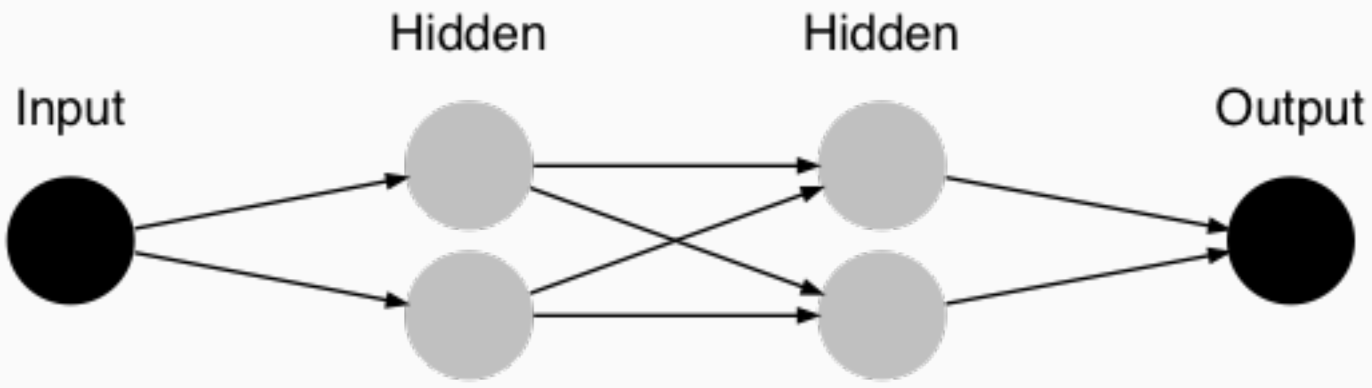


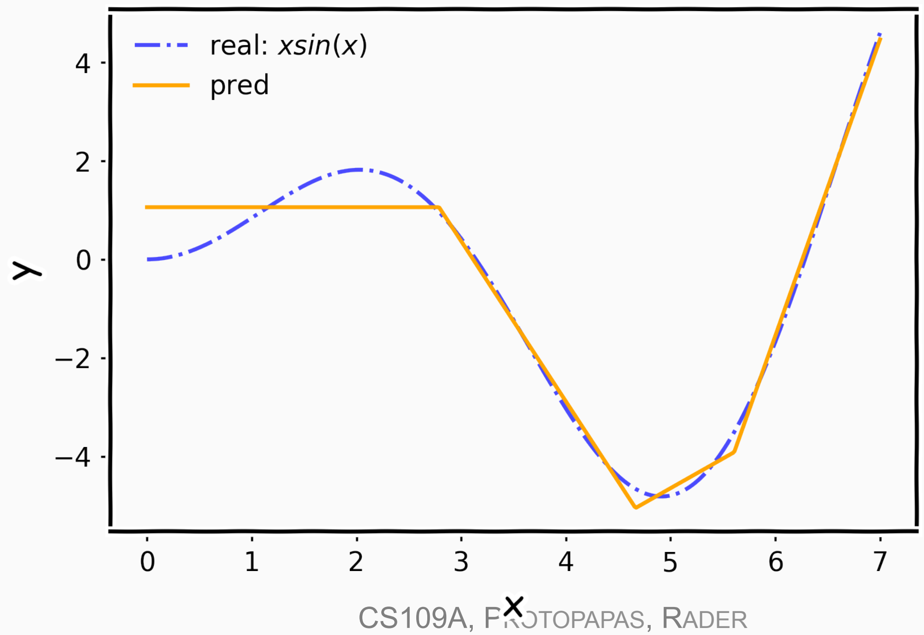
Input
Hidden
Hidden
Output



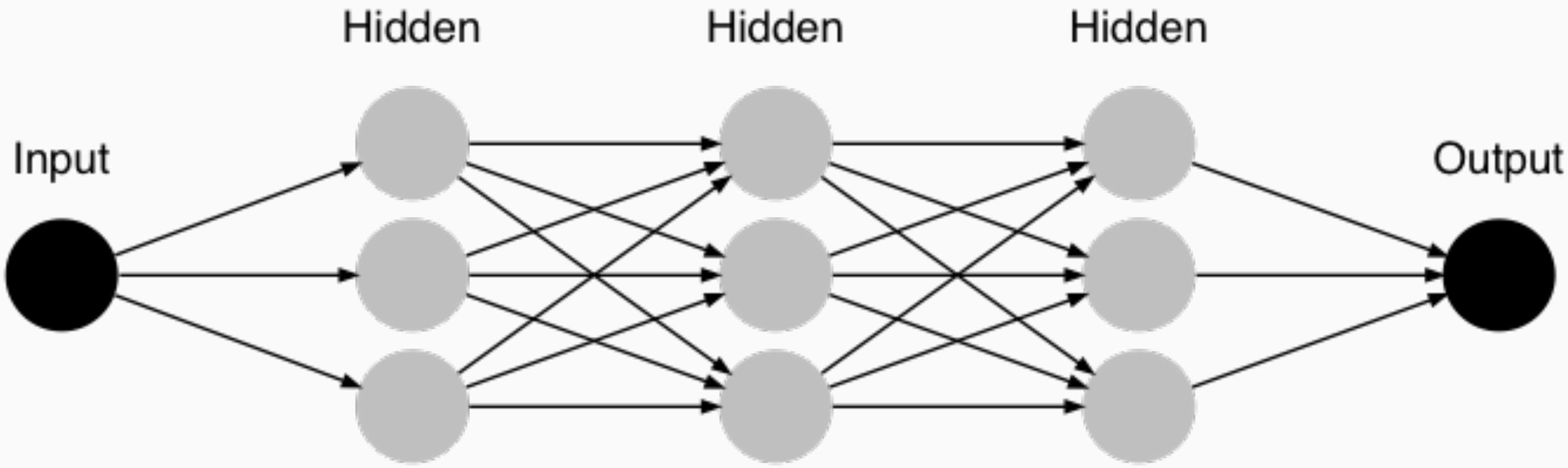
real: $xsin(x)$
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

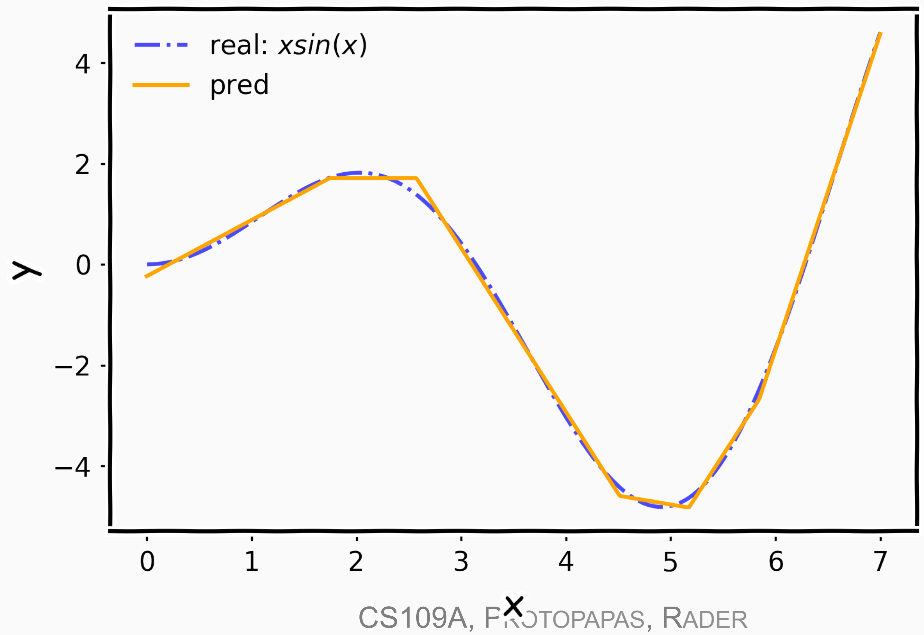
NN in action





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 Neural Network as function approximation.

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

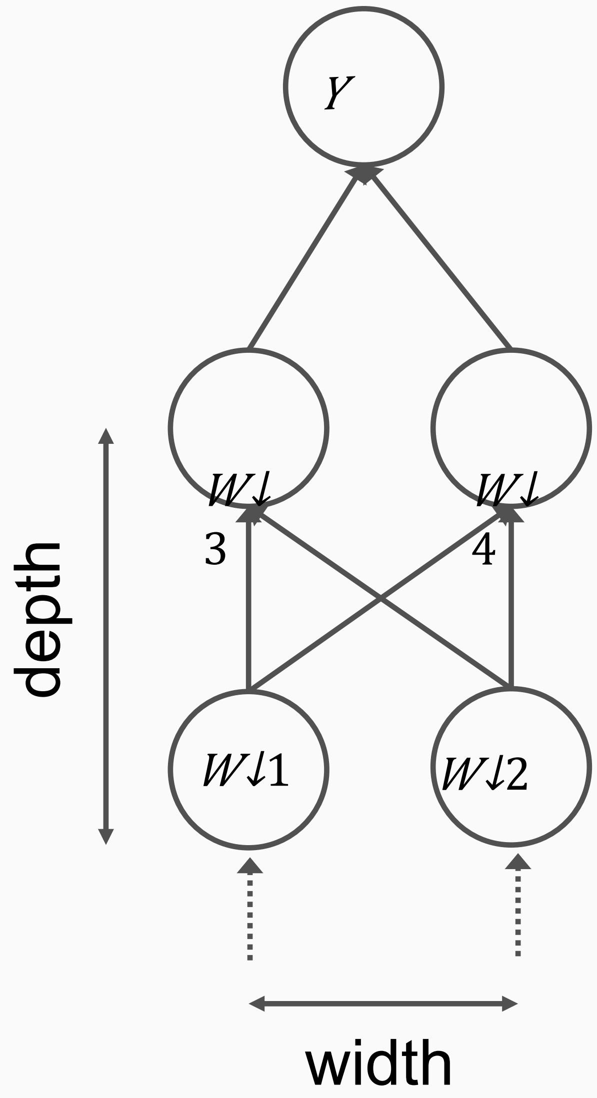
$$Y = f(x) + \epsilon$$

NN: $\Longrightarrow f(x)$

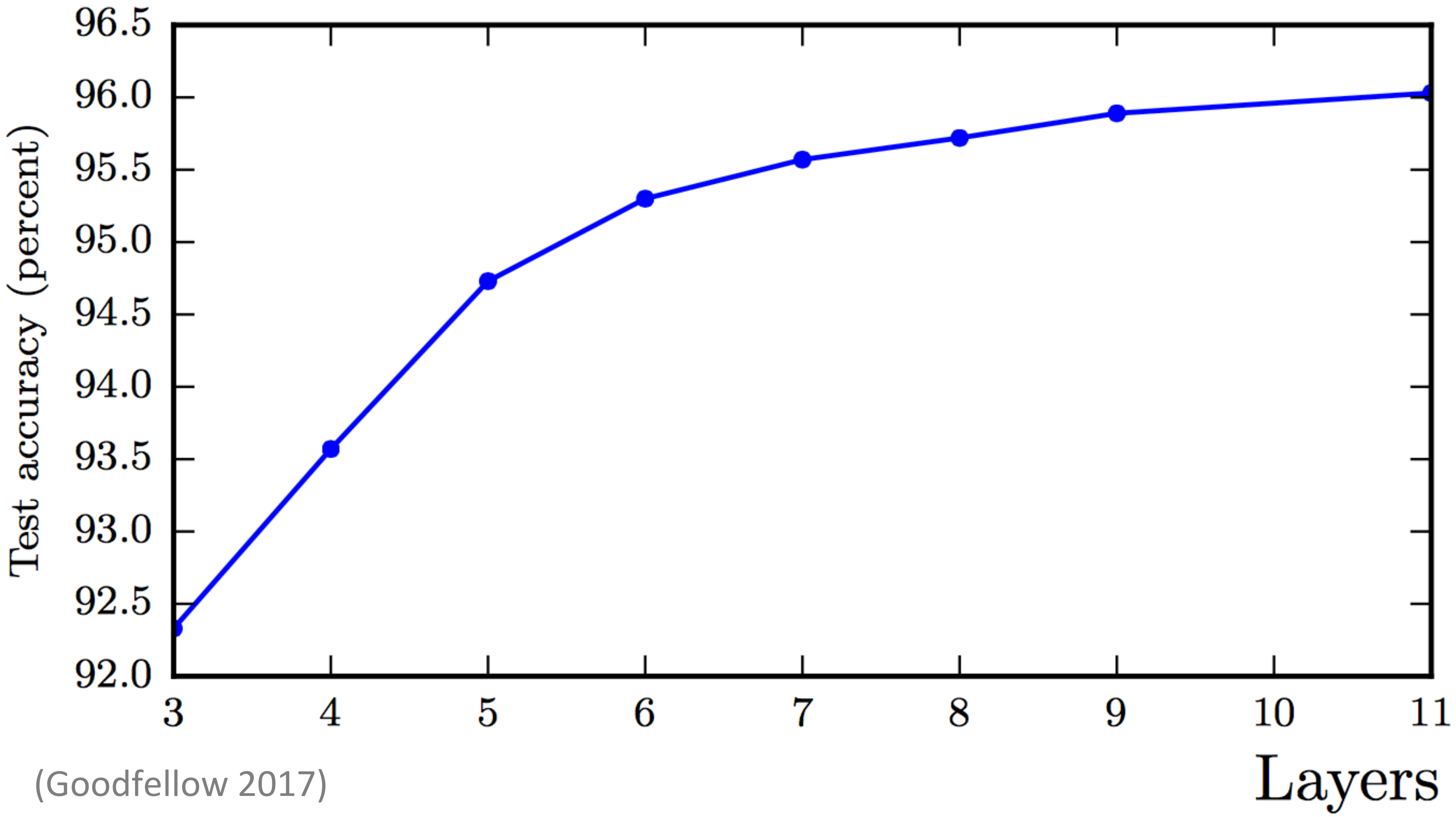
One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy

So why deeper?

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

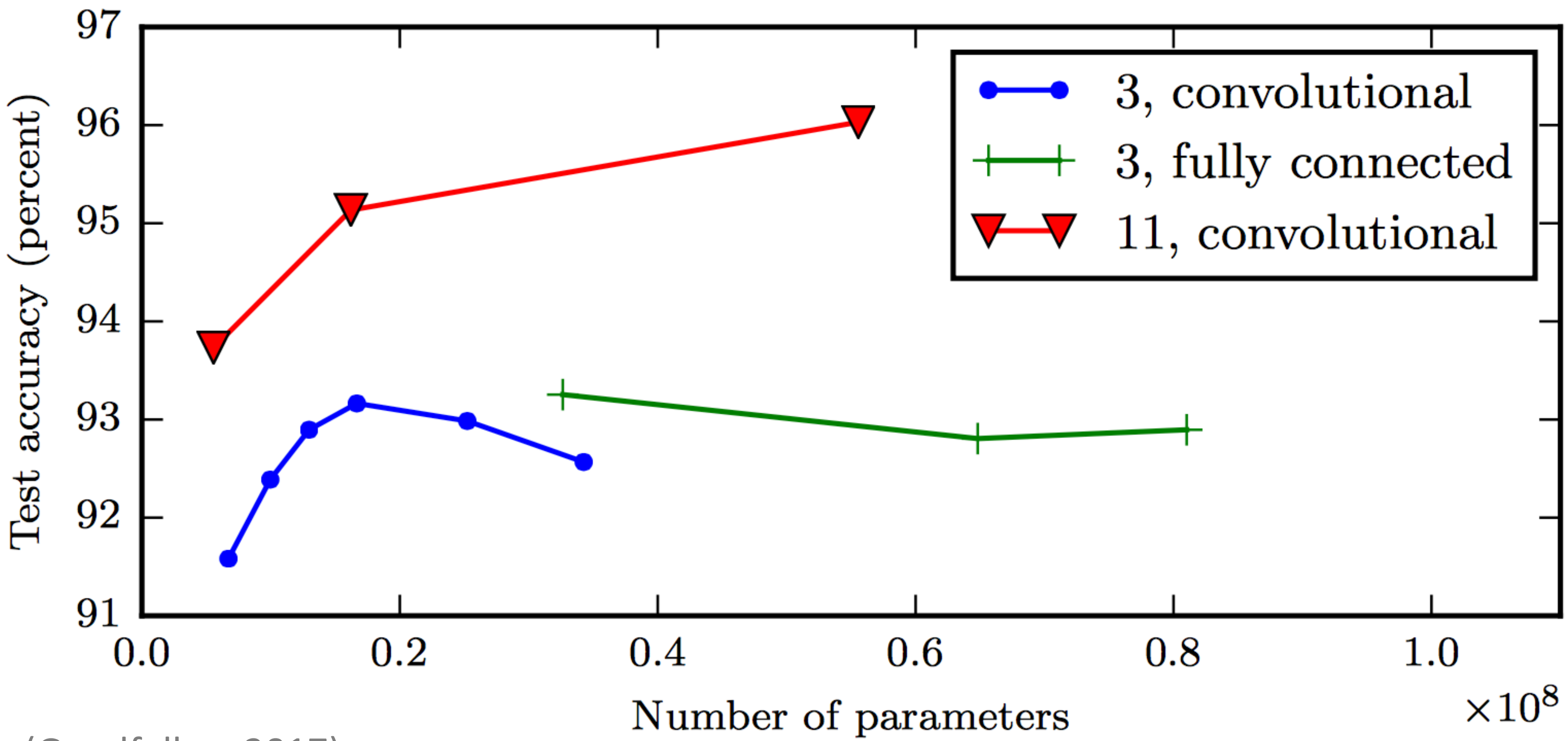


Better Generalization with Depth



(Goodfellow 2017)

Large, Shallow Nets Overfit More



(Goodfellow 2017)