

# Anatomy of NN

Pavlos Protopapas

# 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



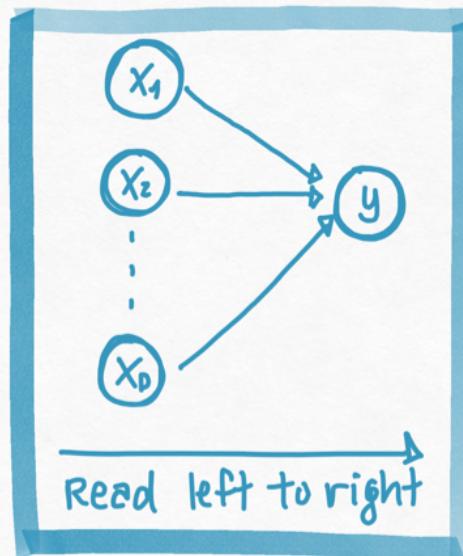
# Graphical representation of simple functions

We build complex functions by composing simple functions of the form:

$$h_w(x) = f(XW + b)$$

where  $f$  is the activation function.

We represent our simple function as a **graph**

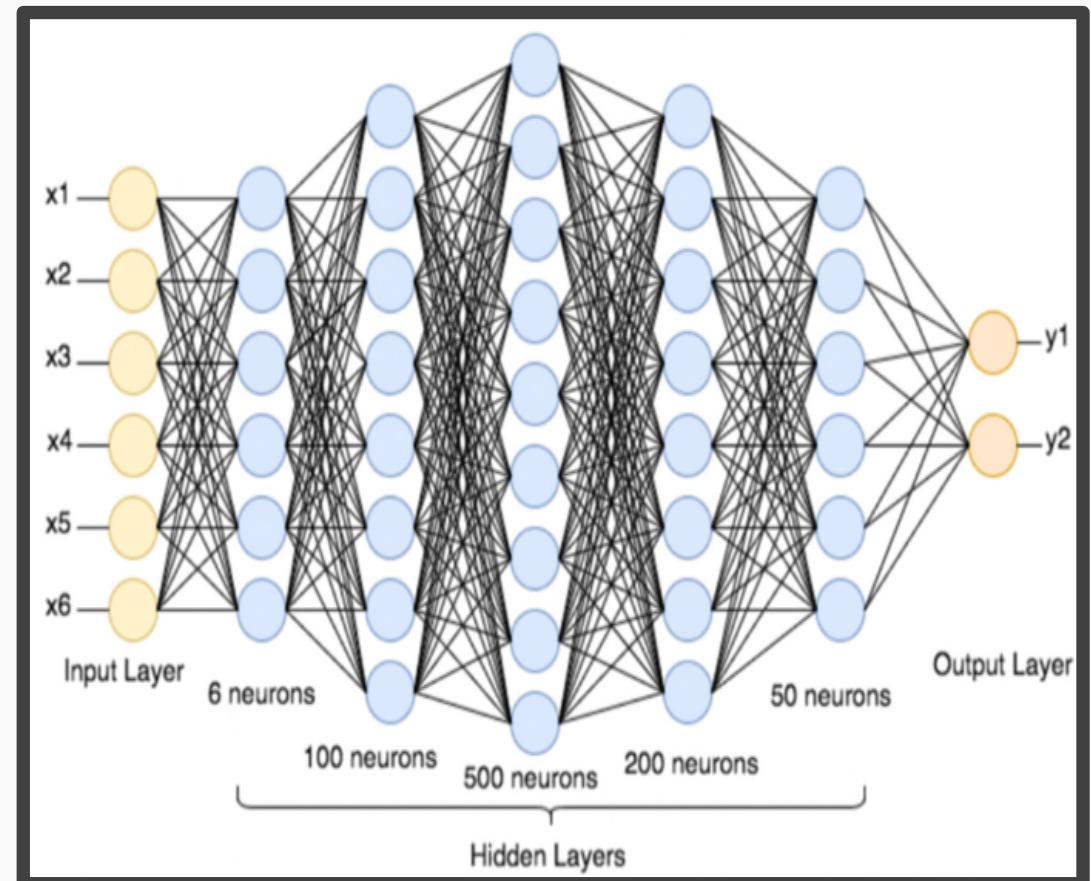
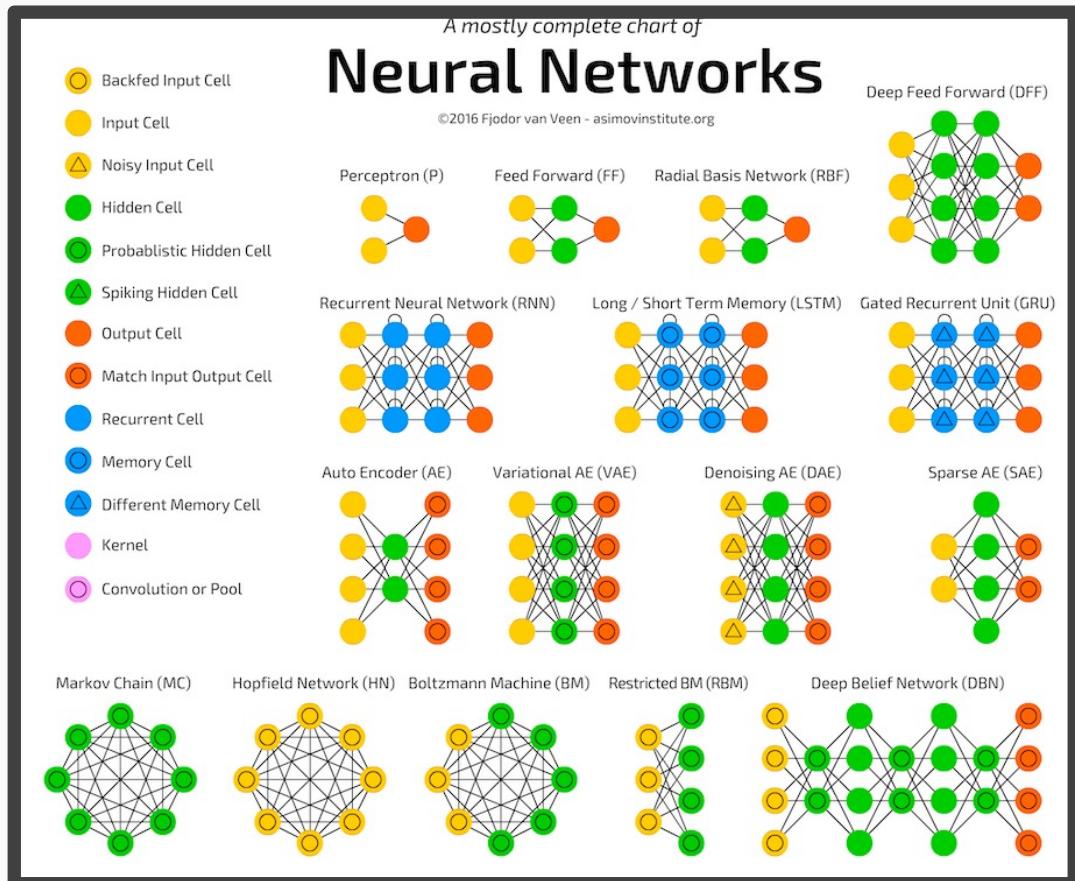


Each edge in this graph represents multiplication by a different constant  $w_d$ .

We call each  $w_d$  a **weight**.



# The zoo of neural network architectures

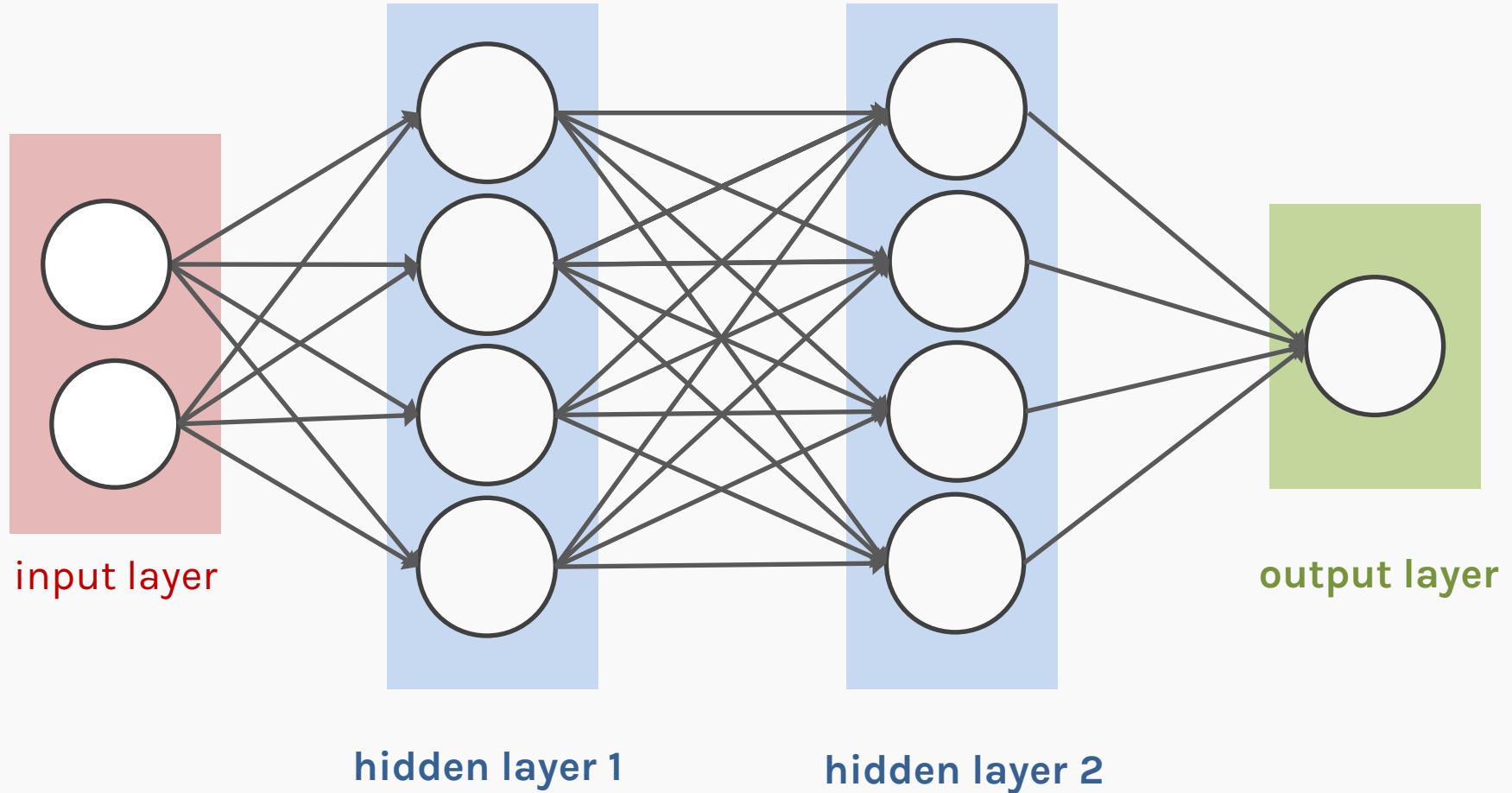


Different architectures result into functions with very different properties.

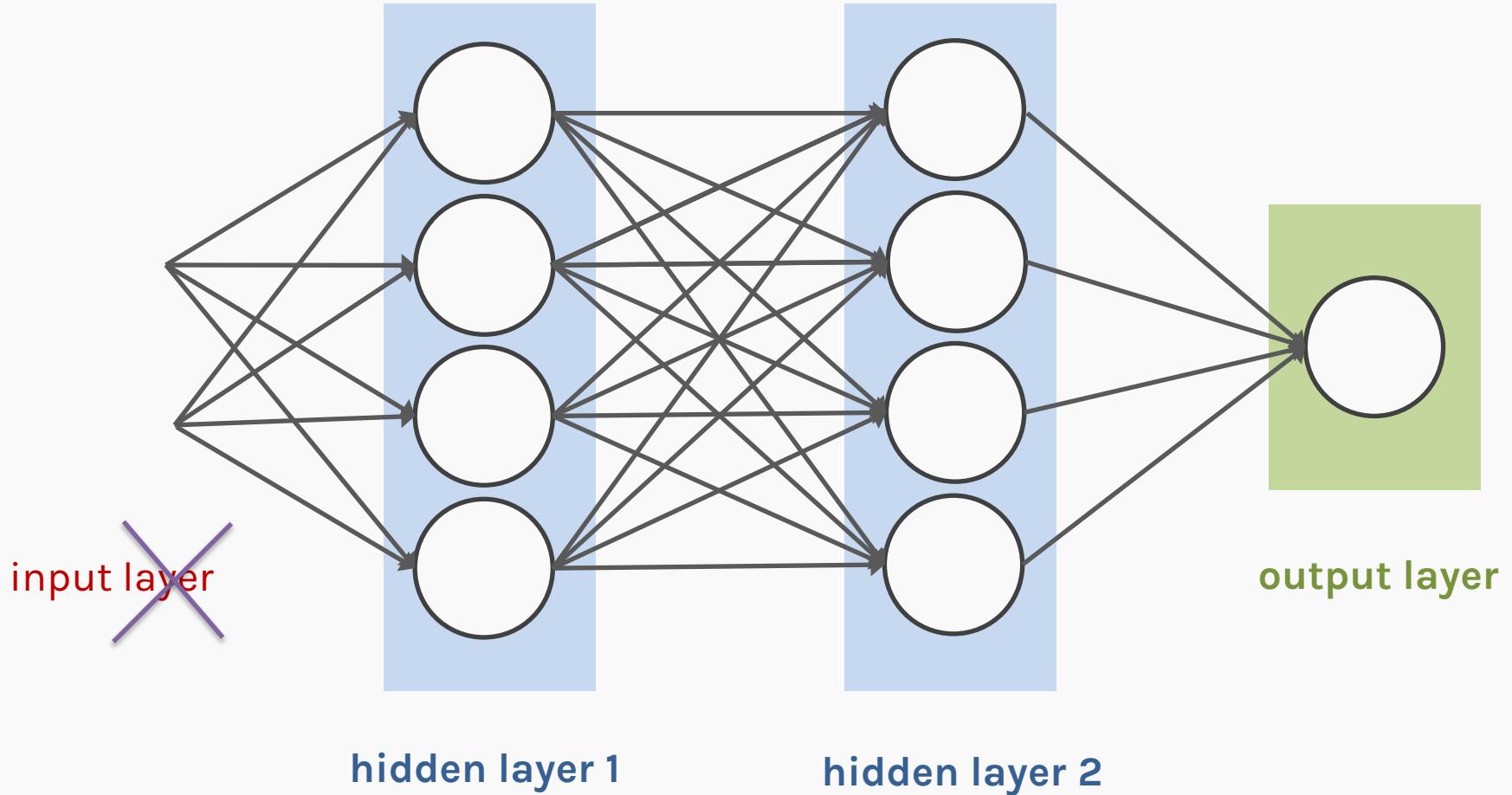
Larger networks can express more complex functions



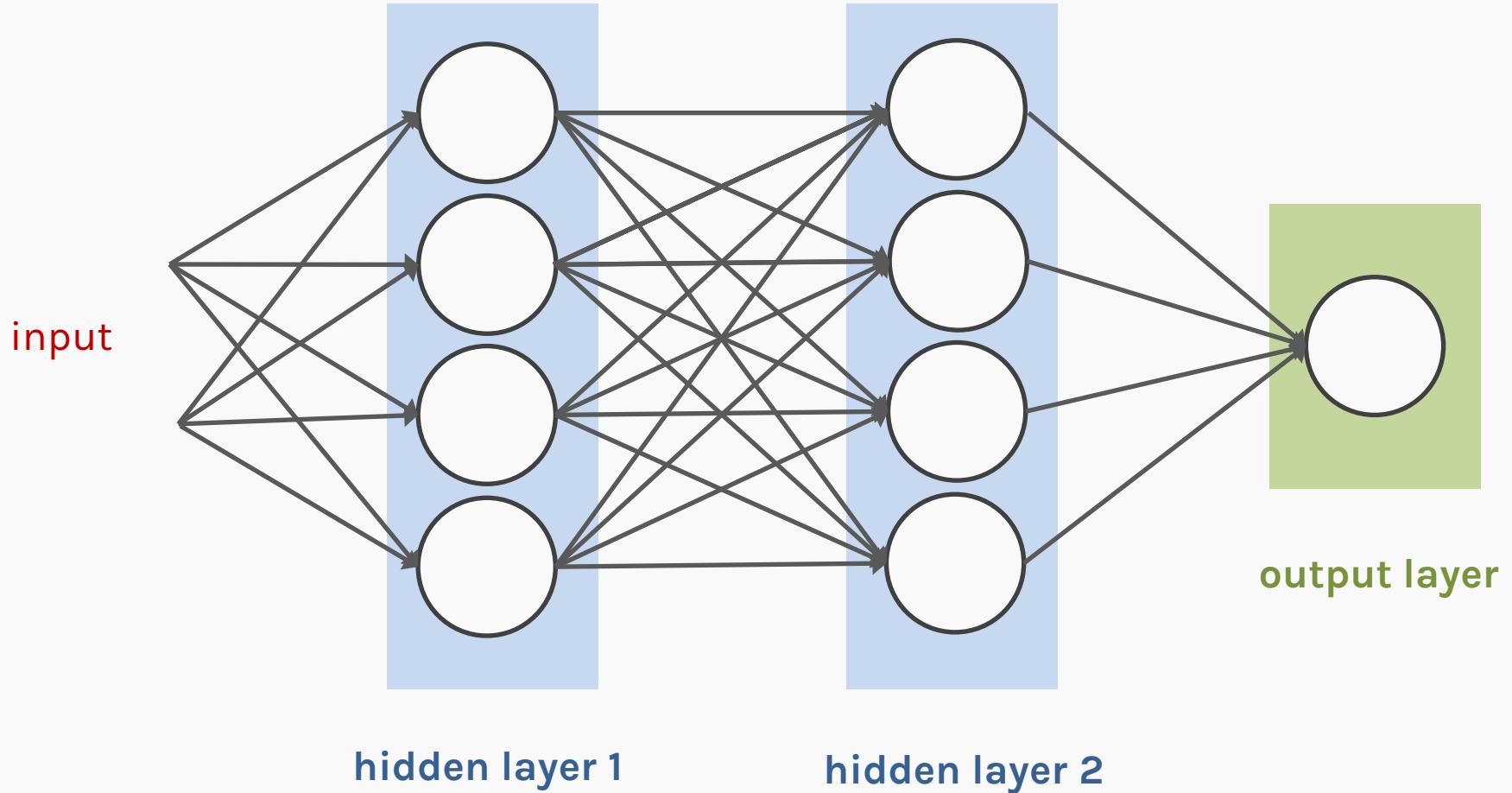
# Anatomy of artificial neural network (ANN)



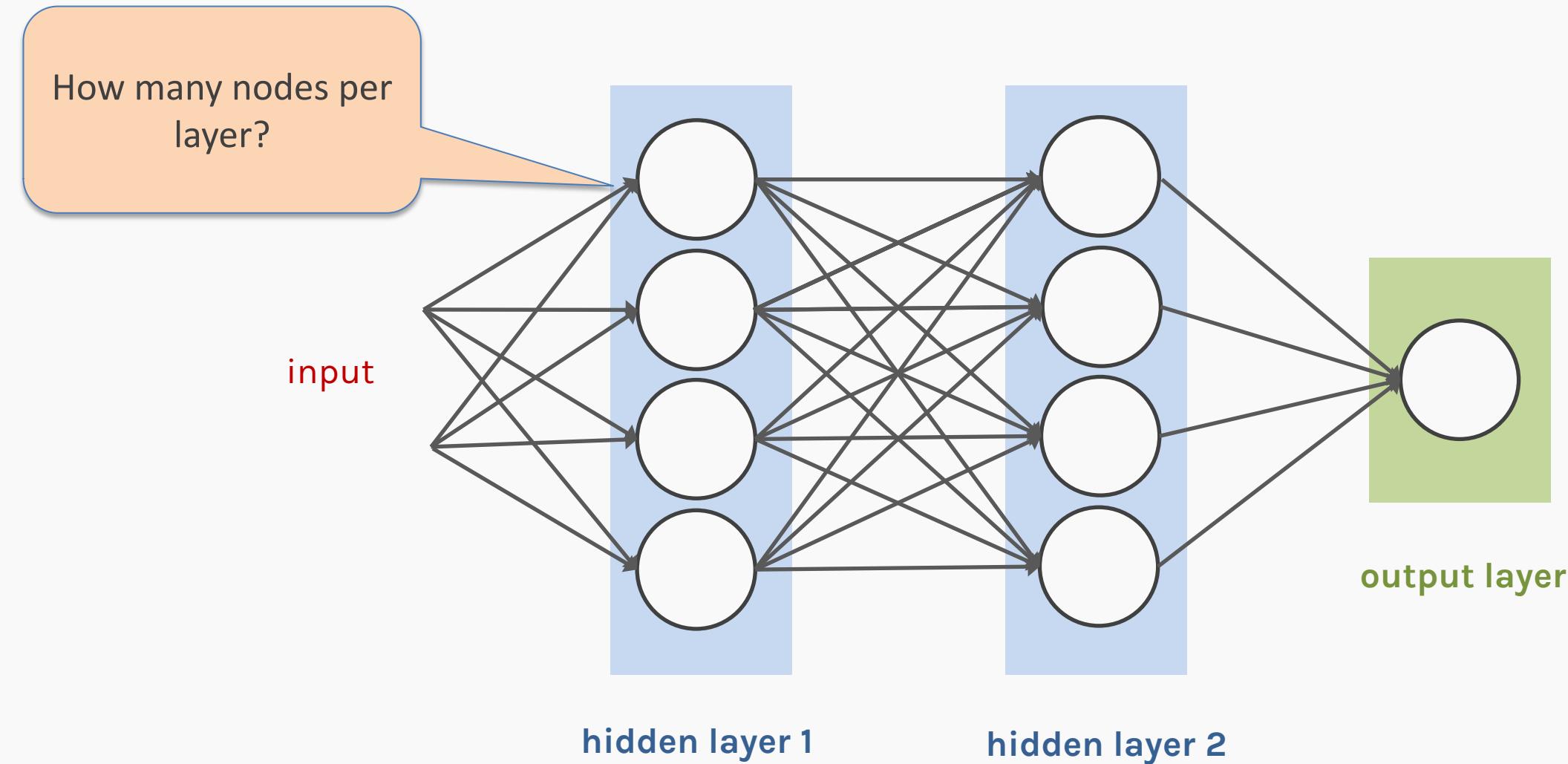
# Anatomy of artificial neural network (ANN)



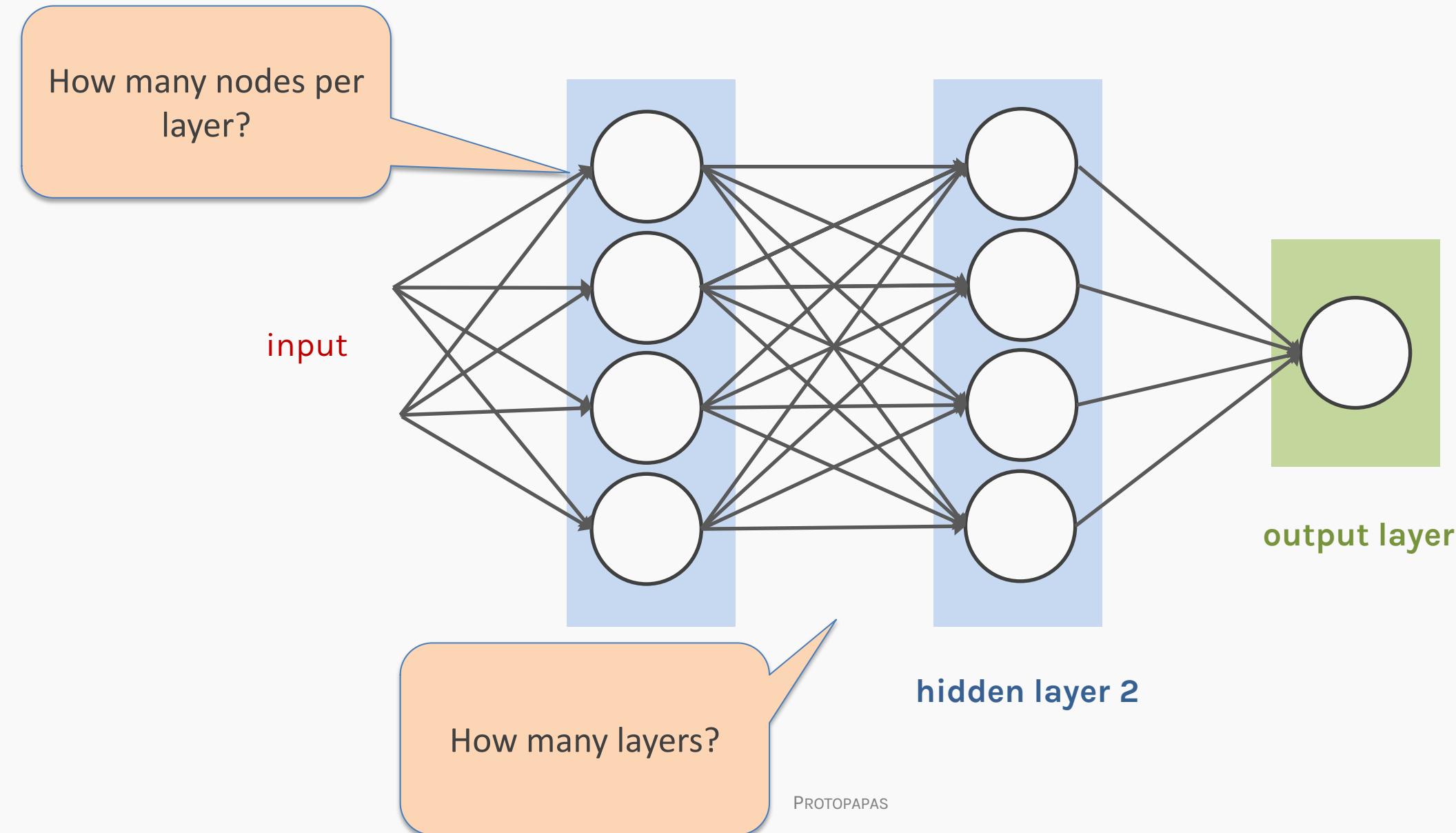
# Anatomy of artificial neural network (ANN)



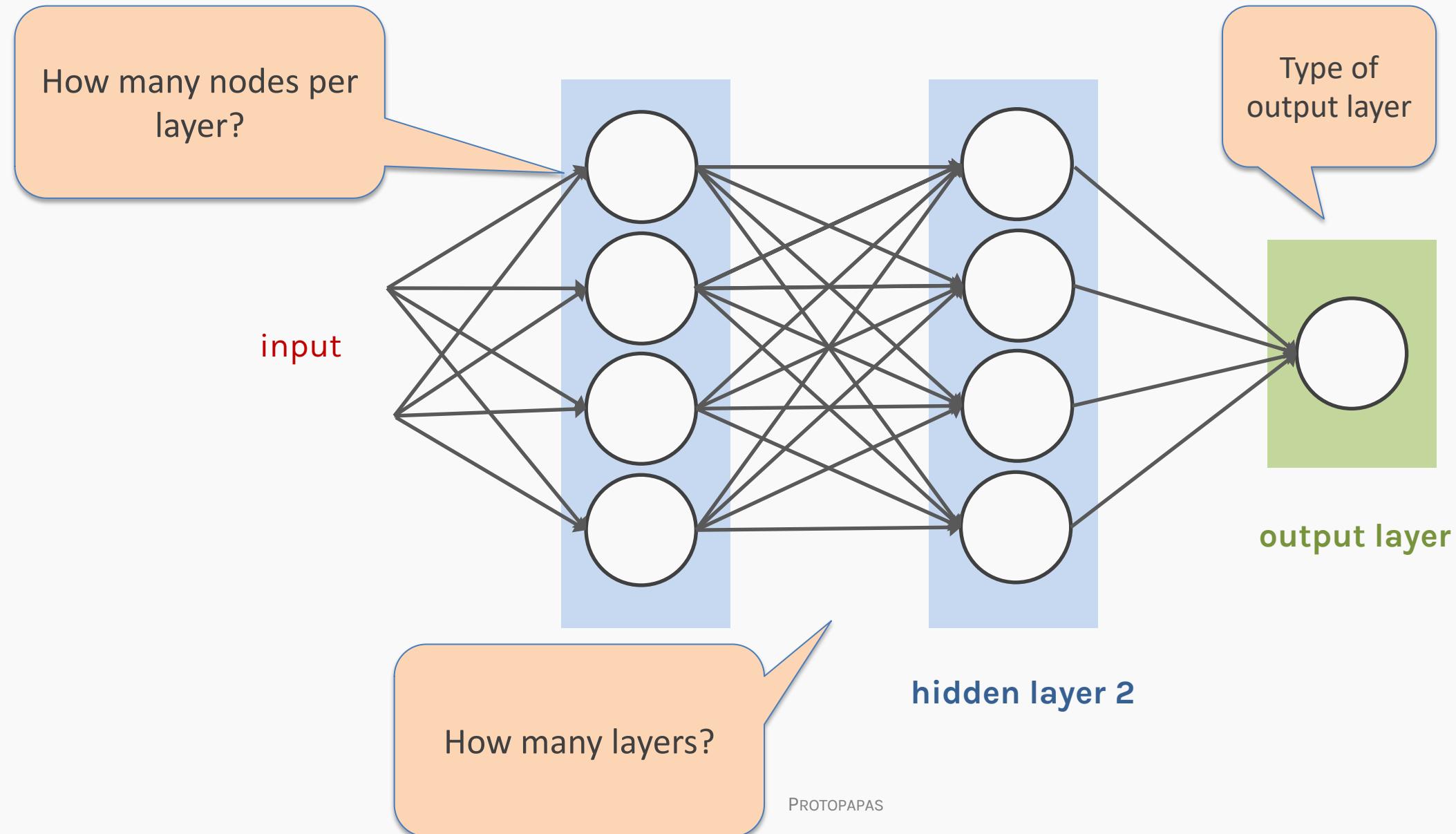
# Anatomy of artificial neural network (ANN)



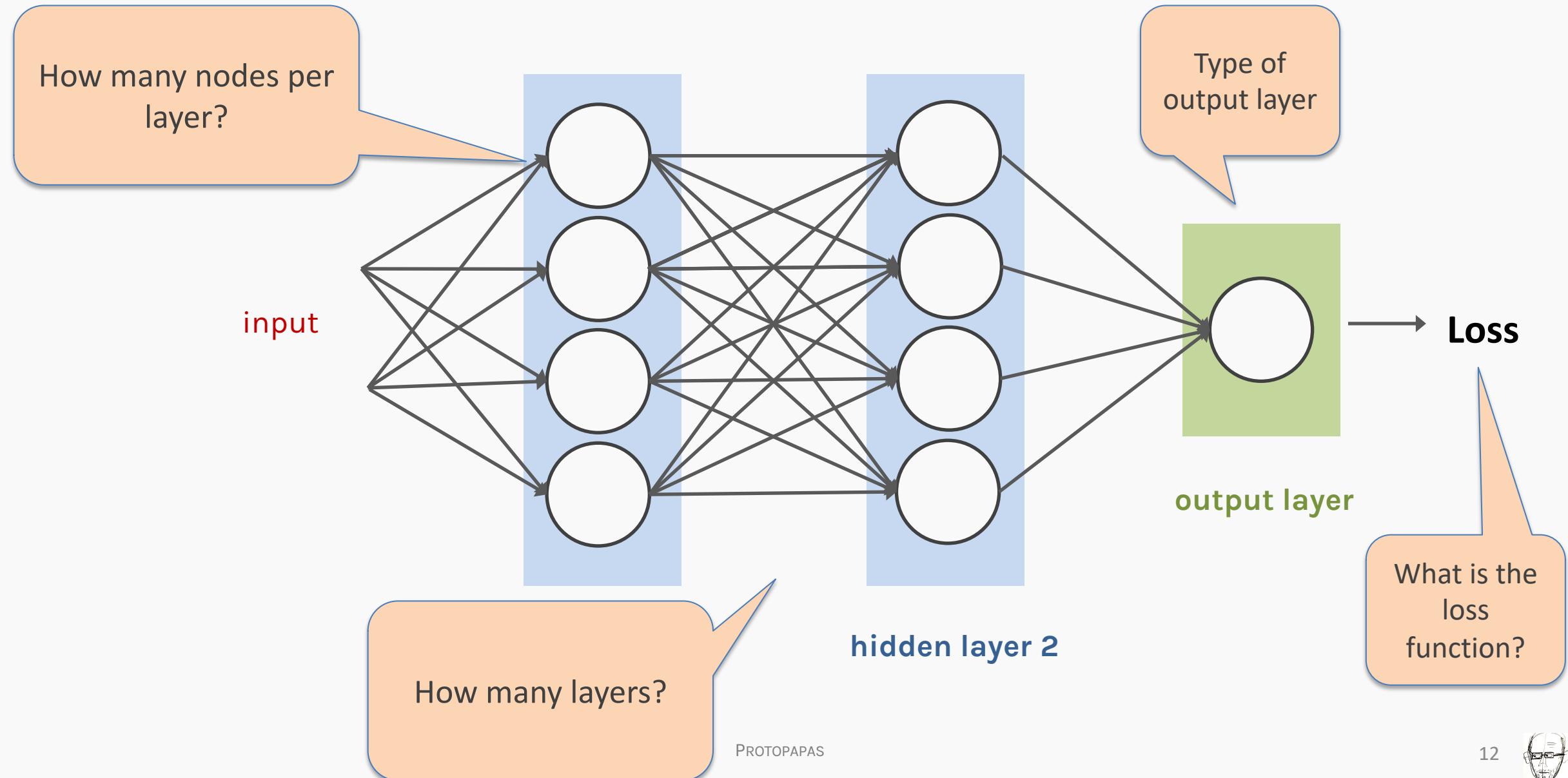
# Anatomy of artificial neural network (ANN)



# Anatomy of artificial neural network (ANN)



# Anatomy of artificial neural network (ANN)



# 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

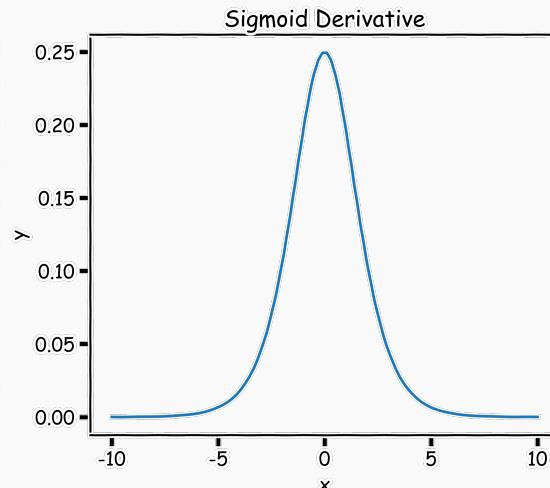
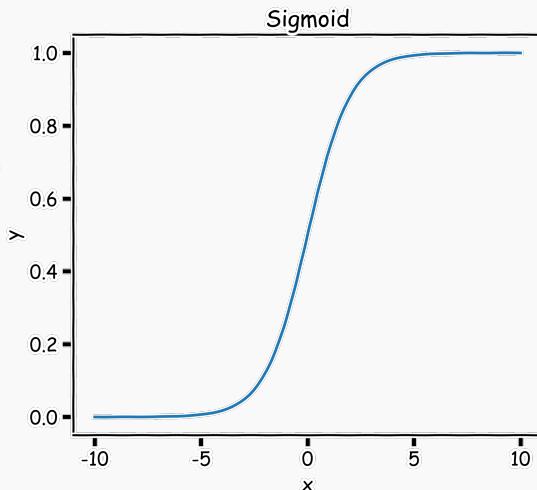
Common choices are

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

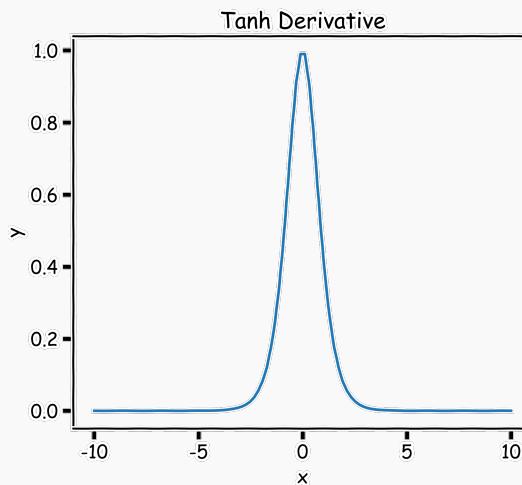
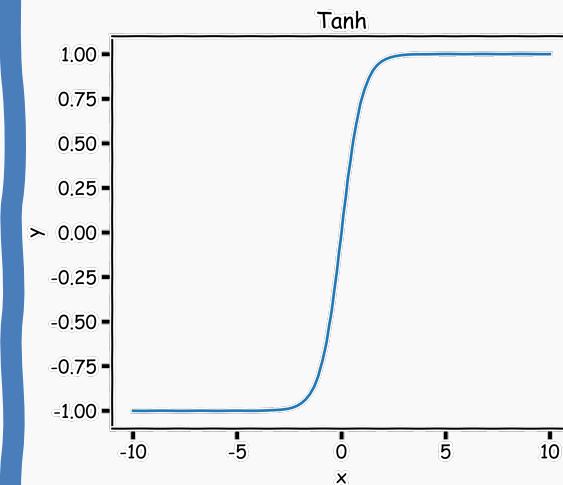


# Sigmoid, $\sigma()$ (aka logistic) and tanh

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



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

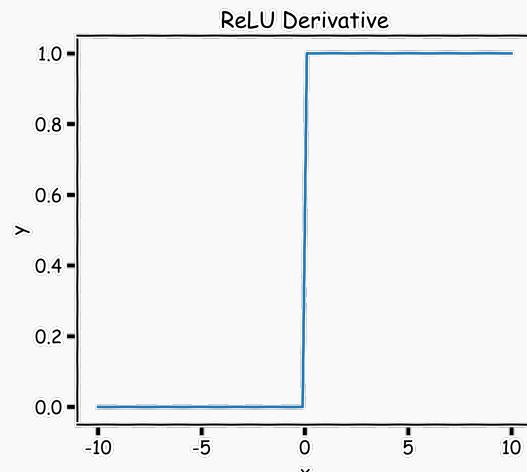
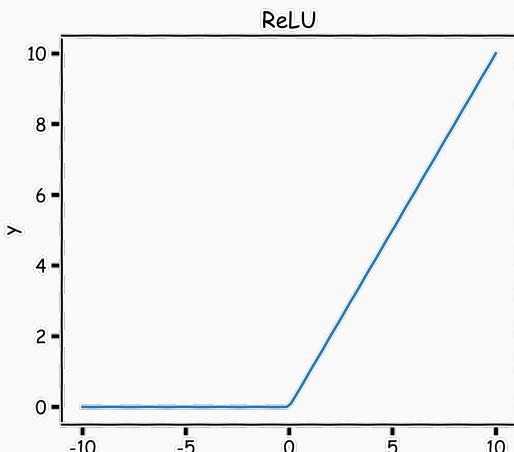


Derivative is **zero** for much of the domain. This leads to “vanishing gradients” in backpropagation.



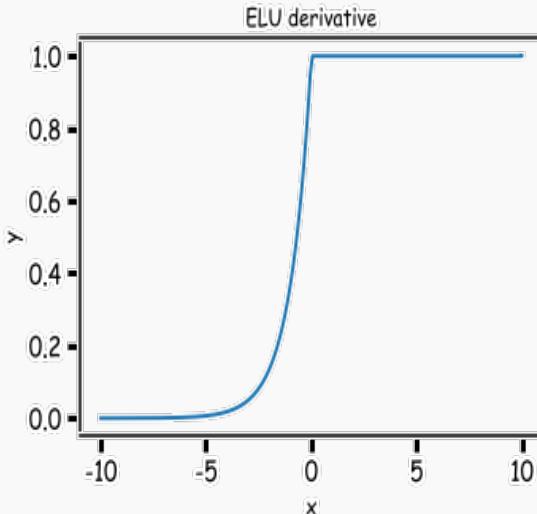
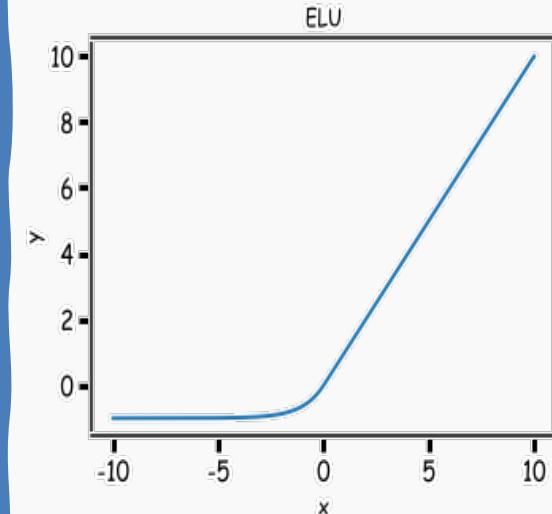
# Rectified Linear Unit, ReLU( ), Exponential ReLU (ELU)

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



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

where  $\alpha$  takes a small value



**Two major advantages:**

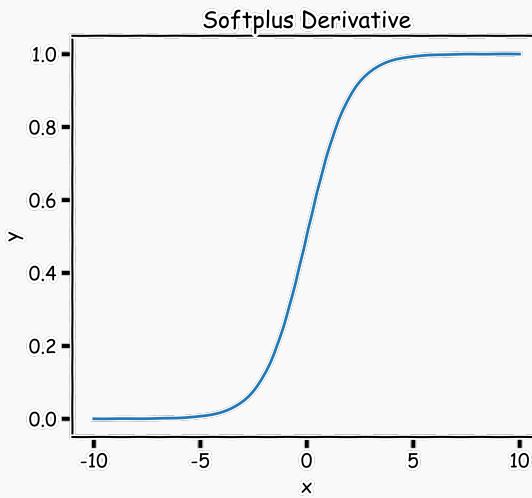
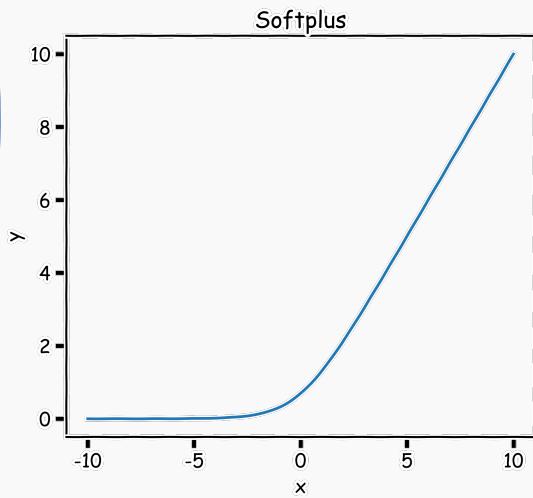
1. No vanishing gradient when  $x > 0$
2. Provides sparsity (regularization) since  $y = 0$  when  $x < 0$

No vanishing gradients and easy to calculate.

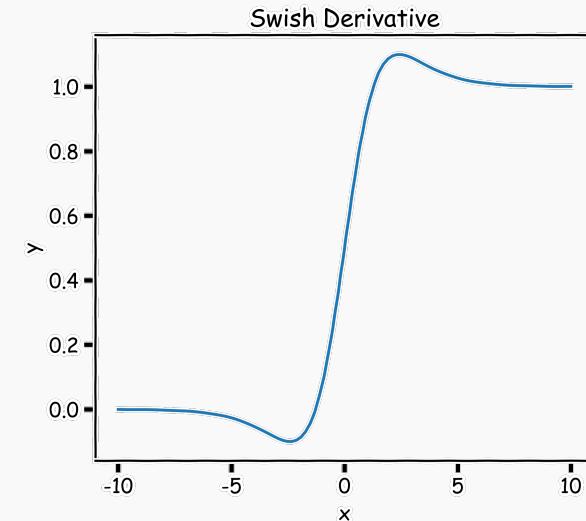
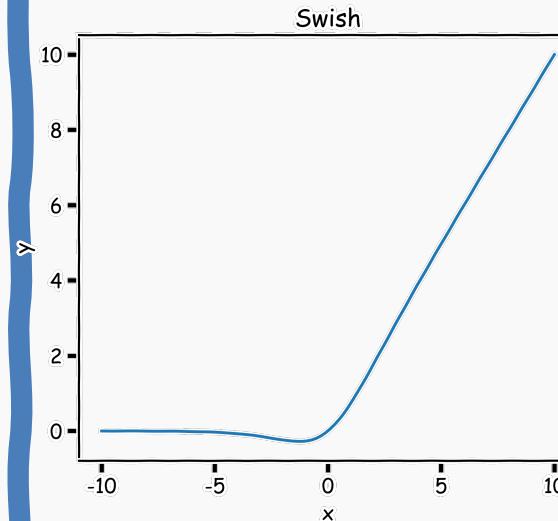


# Softplus and Swish

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



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



The derivative of the softplus is the sigmoid logistic function, which is a smooth approximation of the derivative of the rectifier. So the derivative of the softplus is continuous.

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

## Probabilistic modeling

Likelihood for a given measurement:

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

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 minimizing the -ve log-likelihood:

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



# Loss Function

Do not need to design separate loss functions if we follow the probabilistic modeling approach, i.e. minimize the -ve likelihood function.

## Examples:

- Distribution is **Normal** then -ve log-likelihood is **MSE** :

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

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

- Distribution is **Bernouli** then -ve log-likelihood is **Binary Cross-Entropy**:

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

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



# Design Choices

---

Activation function

Loss function

Output units

Architecture

Optimizer

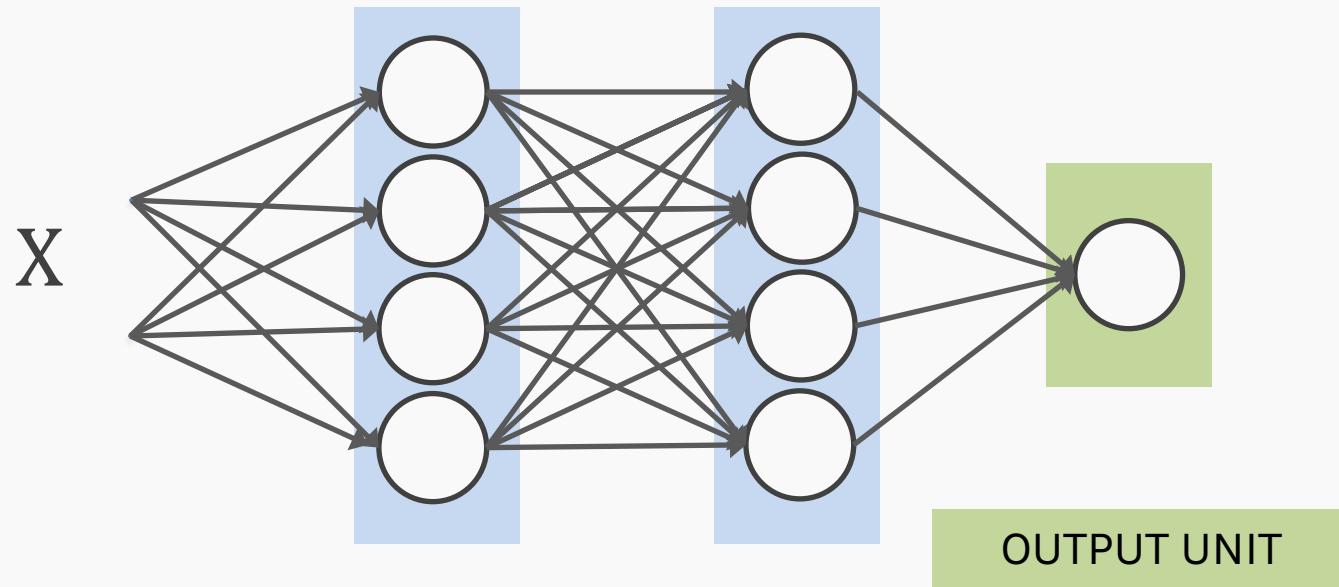


# Output Units

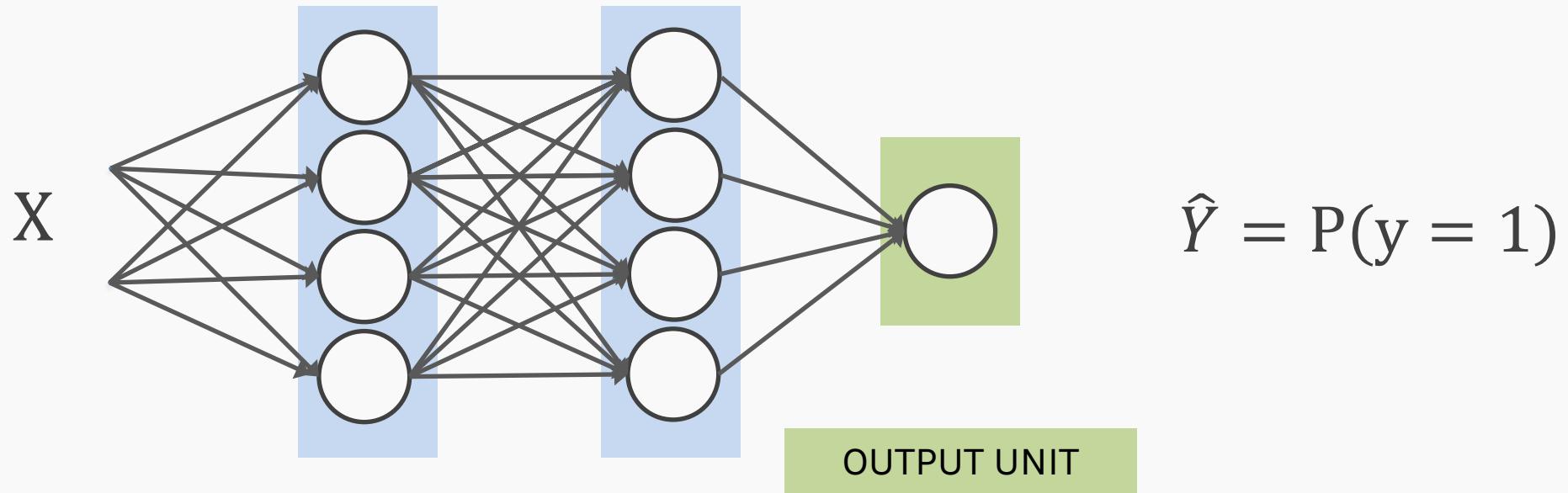
<b>Output Type</b>	<b>Output Distribution</b>	<b>Output layer</b>	<b>Loss Function</b>
Binary	Bernoulli	?	Binary Cross Entropy



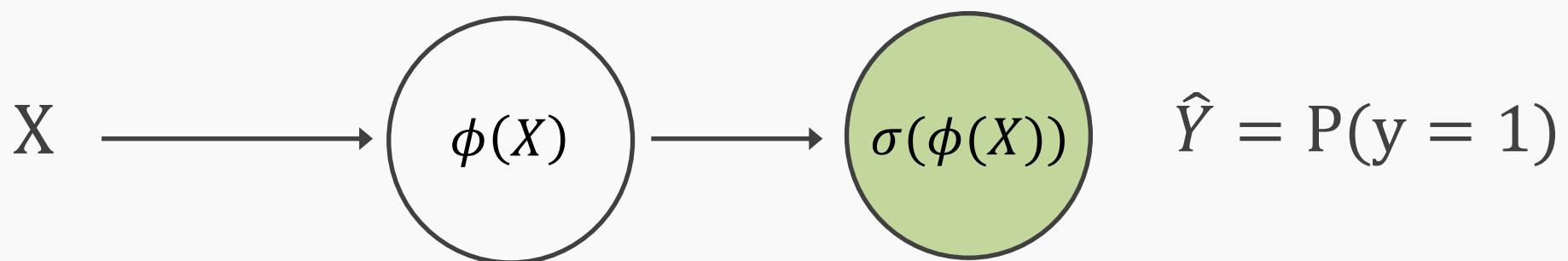
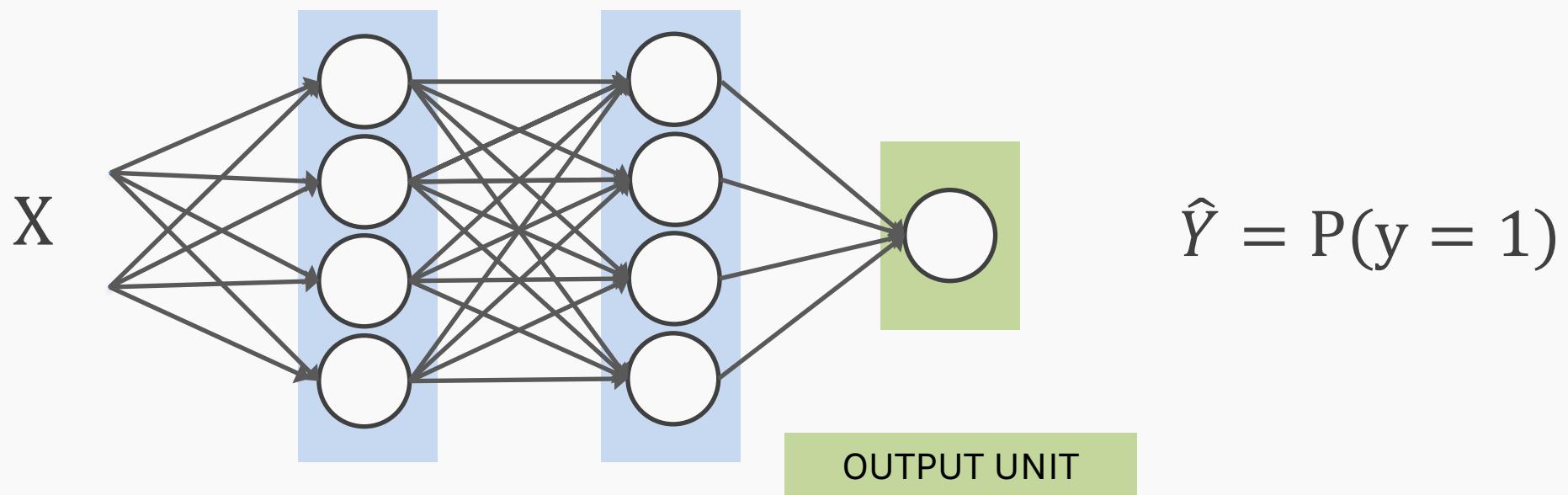
# Output unit for binary classification



# Output unit for binary classification



# Output unit for binary classification



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

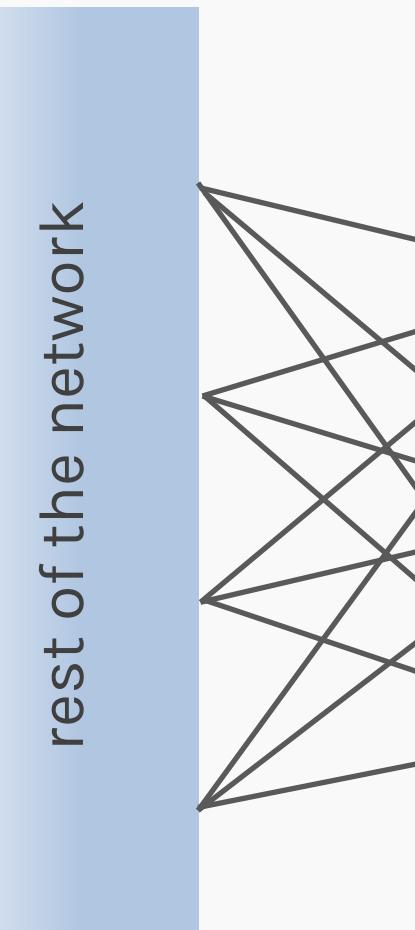


# Output Units

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



# SoftMax



$$\phi_k(X)$$

A score

B score

C score

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

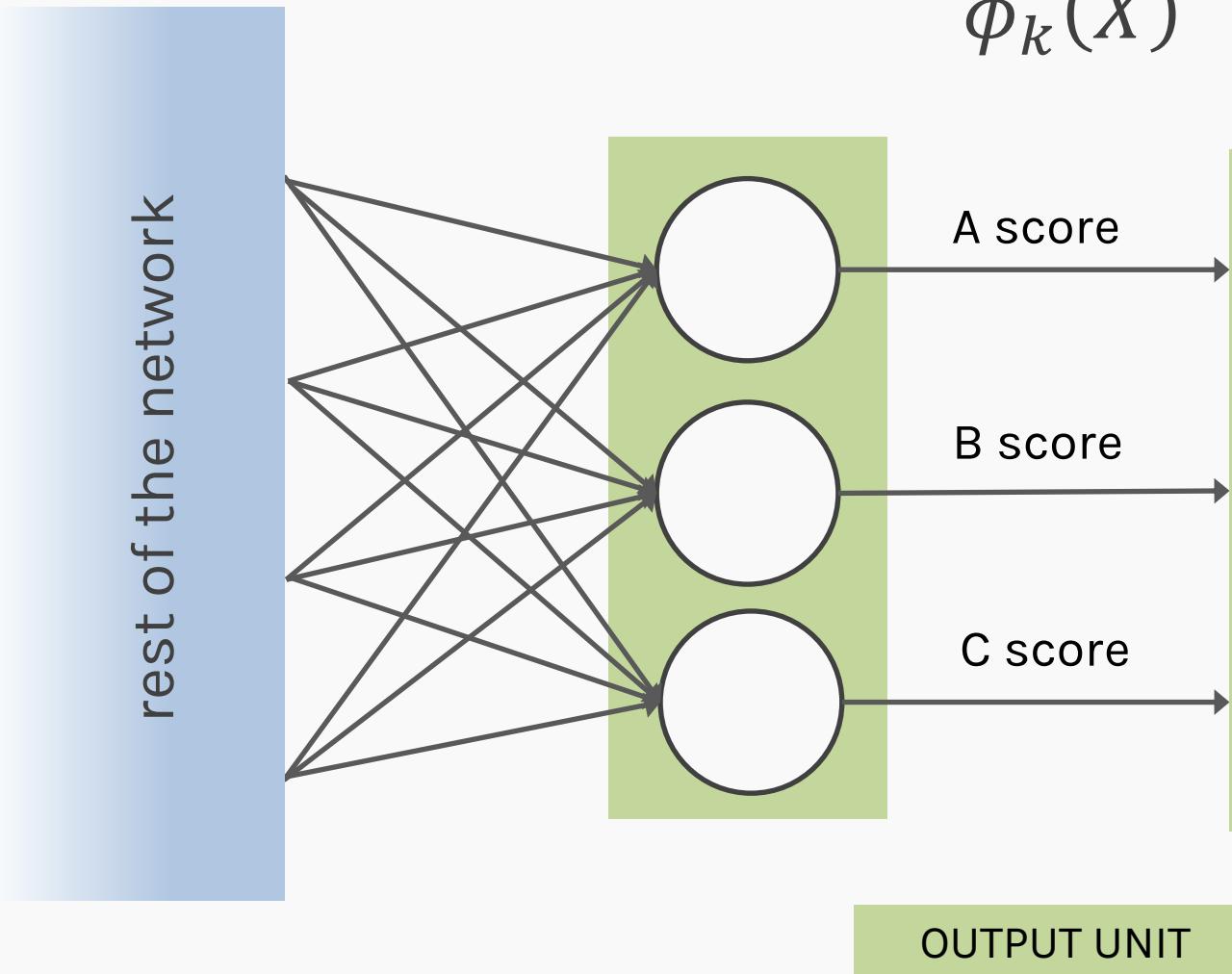
Probability of A

Probability of B

Probability of C



# SoftMax



$$\phi_k(X)$$

A score

B score

C score

OUTPUT UNIT

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

Probability of A

Probability of B

Probability of C

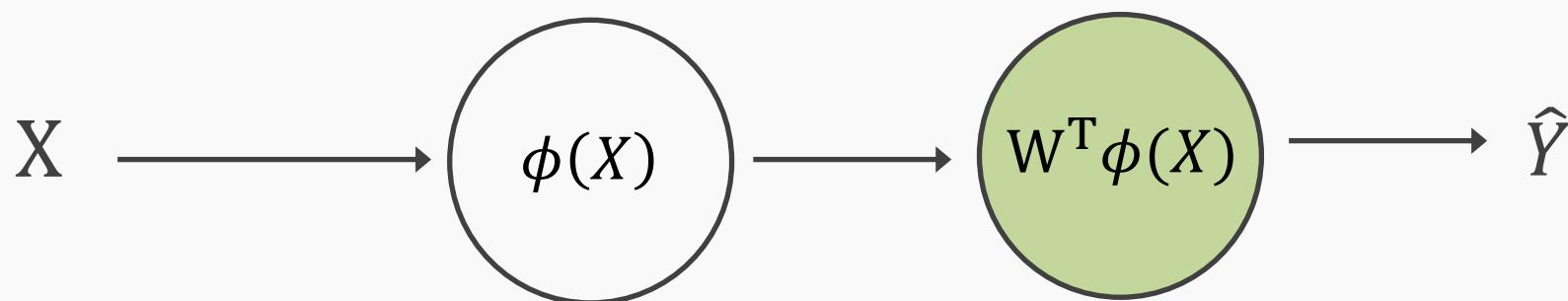
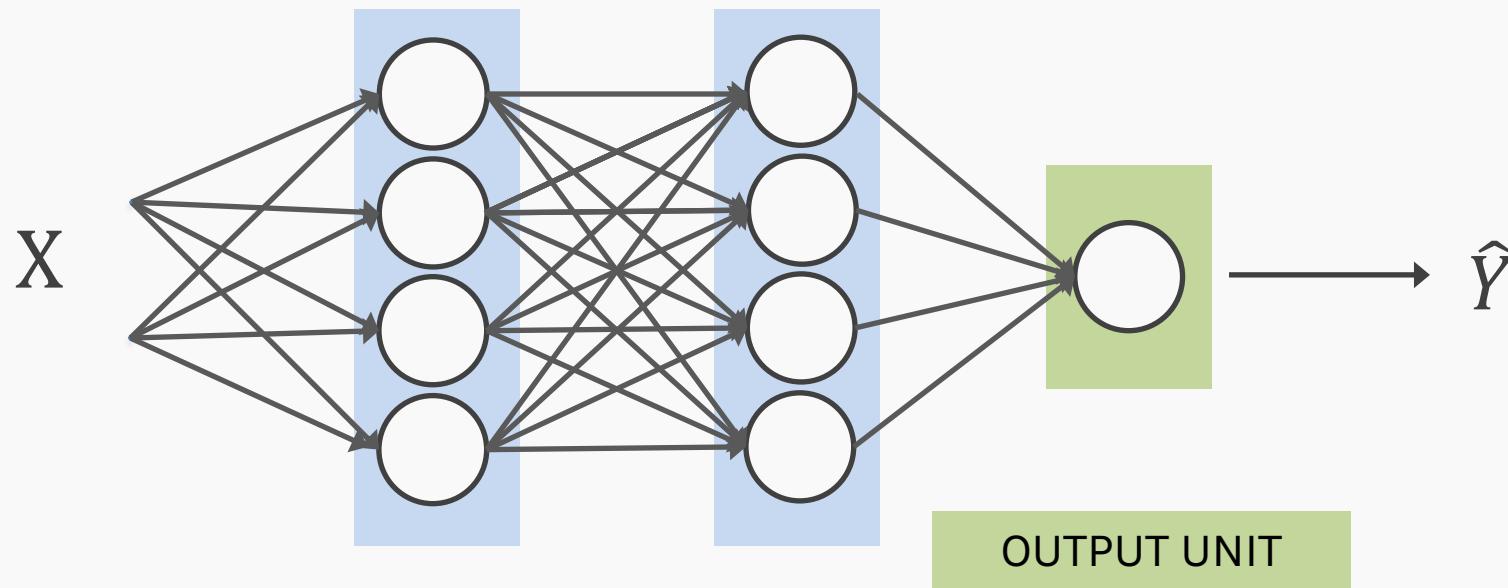


# Output Units

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



# Output unit for regression



$$X \Rightarrow \phi(X) \Rightarrow \hat{Y} = W^T \phi(X)$$



# Output Units

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



# Design Choices

---

Activation function

Loss function

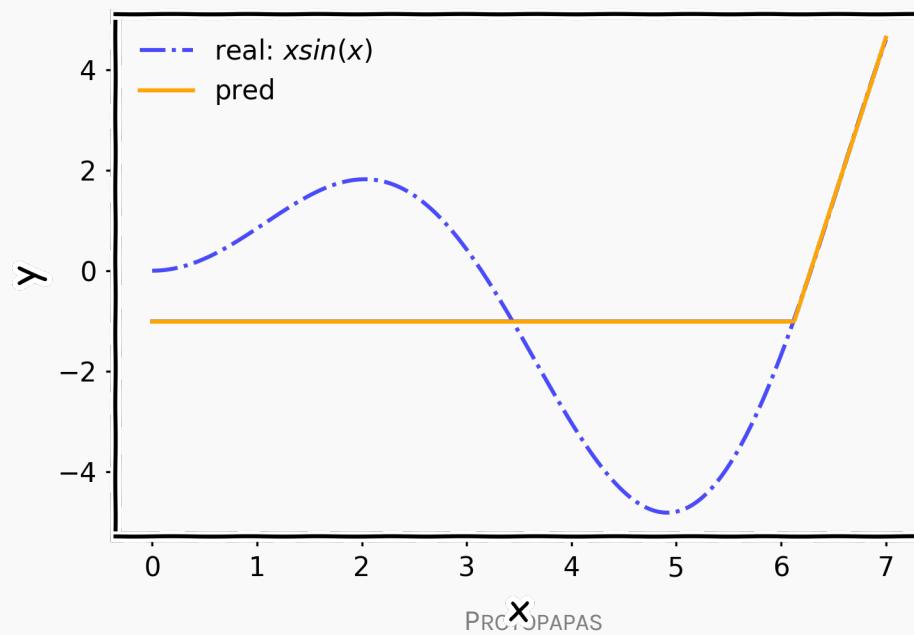
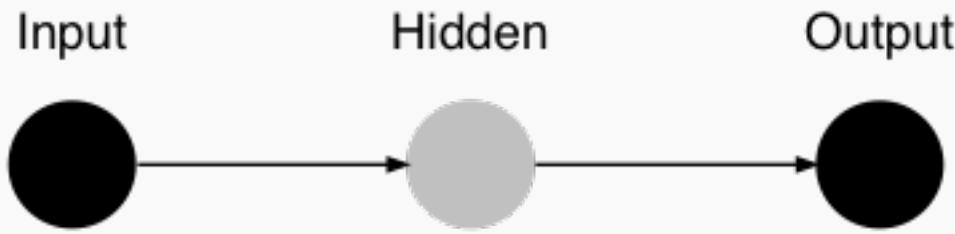
Output units

Architecture

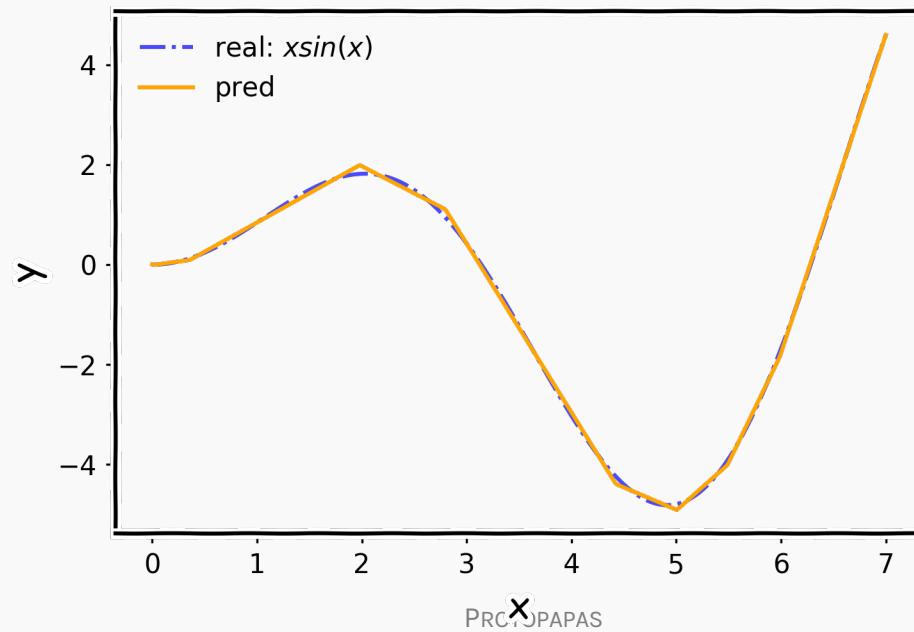
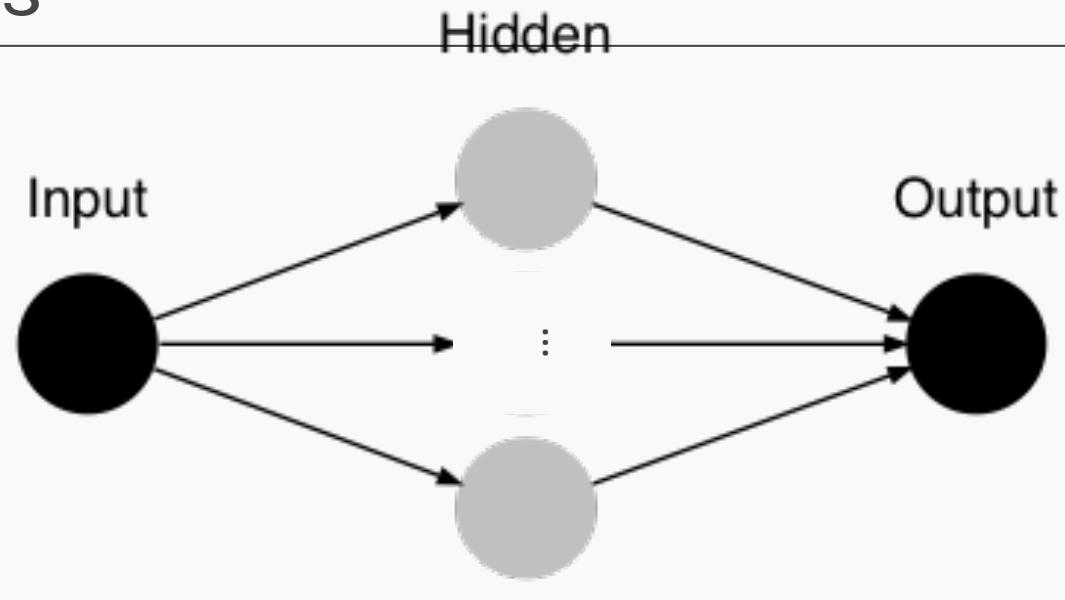
Optimizer



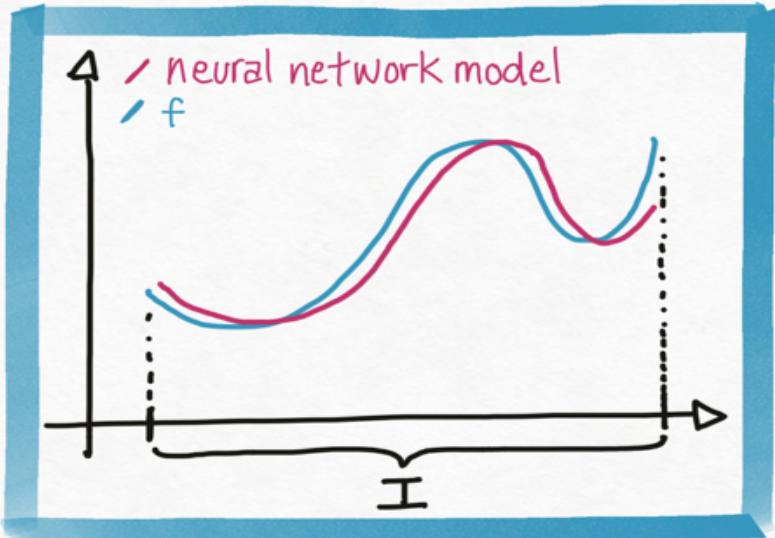
# Number of nodes



# Number of nodes



# Neural Networks as Universal Approximators



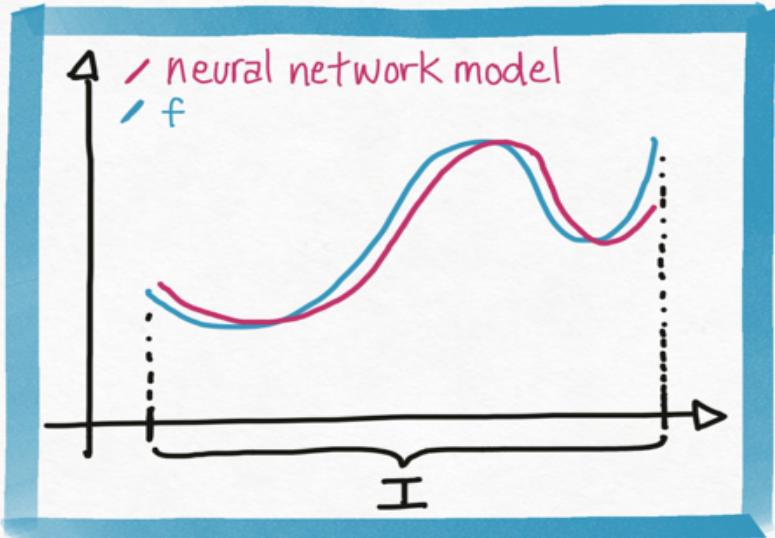
We have seen that neural networks can represent complex functions, but are there limitations on what a neural network can express?

## Theorem:

For any continuous function  $f$  defined on a bounded domain, we can find a neural network that approximates  $f$  with an arbitrary degree of accuracy.



# Neural Networks as Universal Approximators



We have seen that neural networks can represent complex functions, but are there limitations on what a neural network can express?

## Theorem:

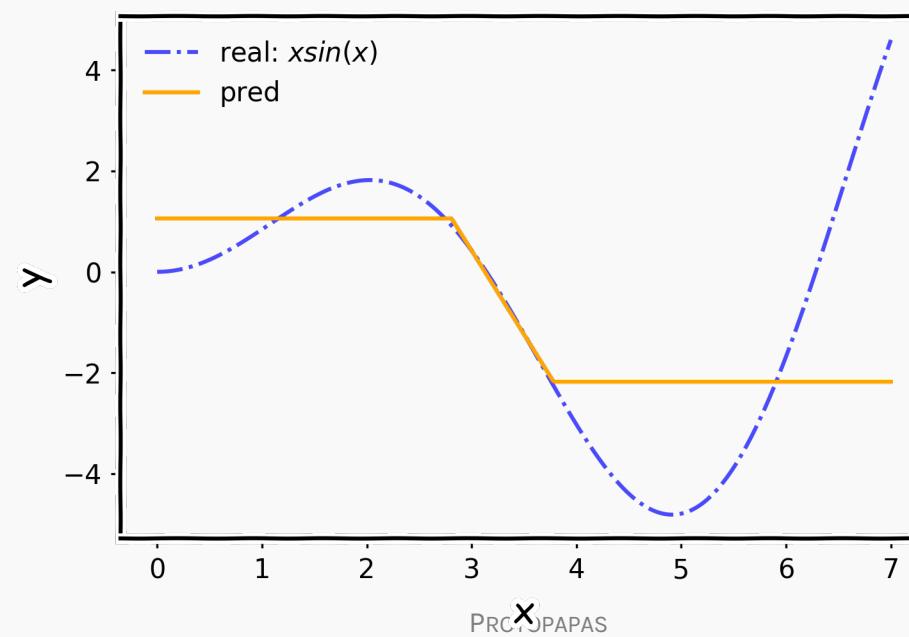
For any continuous function  $f$  defined on a bounded domain, we can find a neural network that approximates  $f$  with an arbitrary degree of accuracy.

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

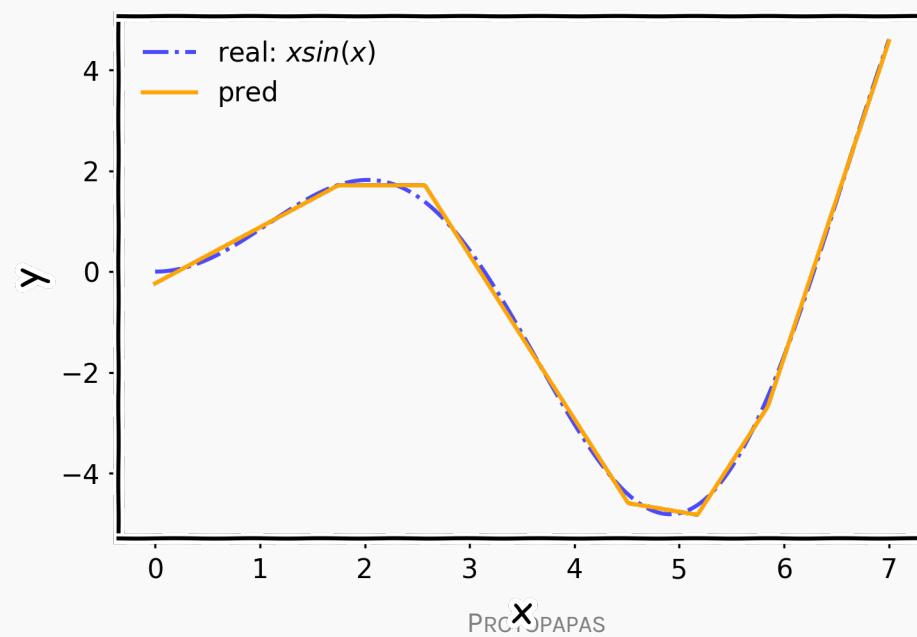
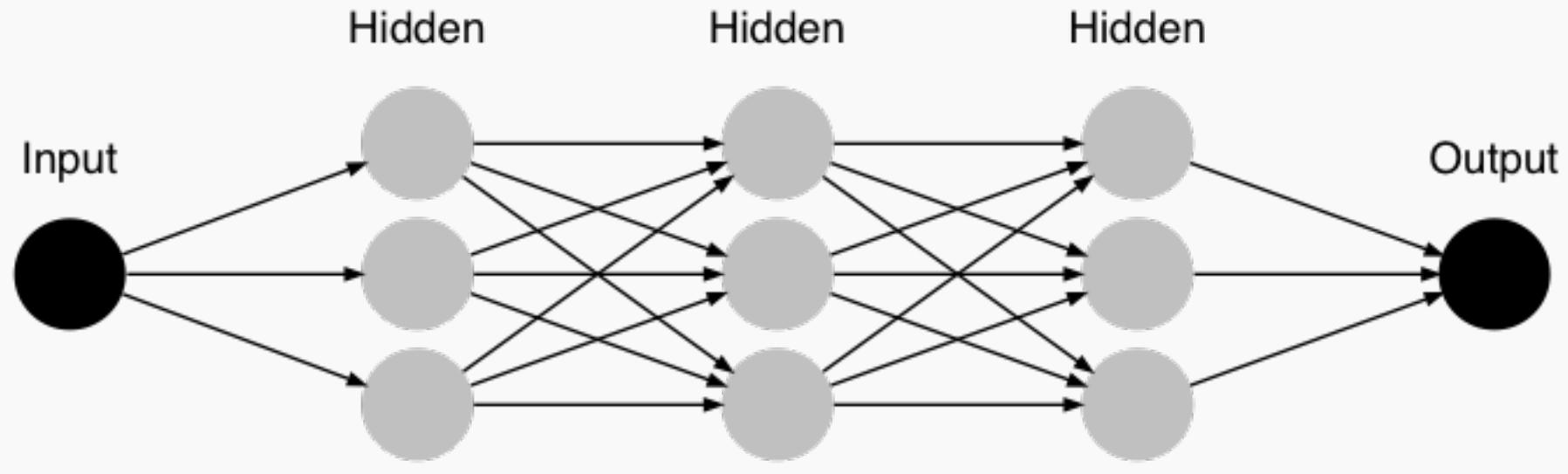
So why deeper?



# Layers

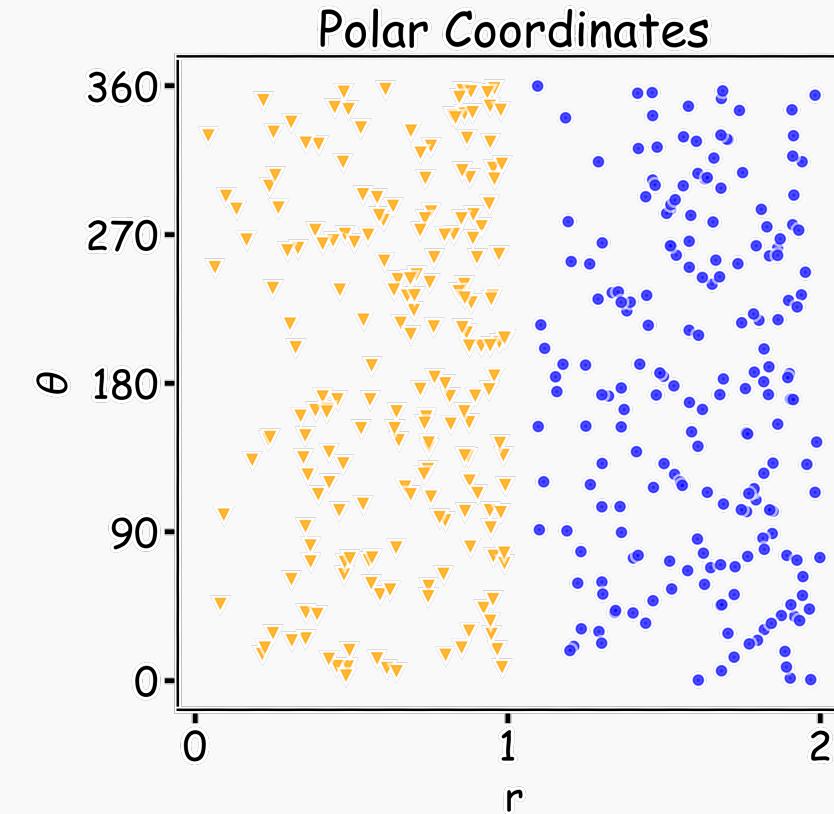
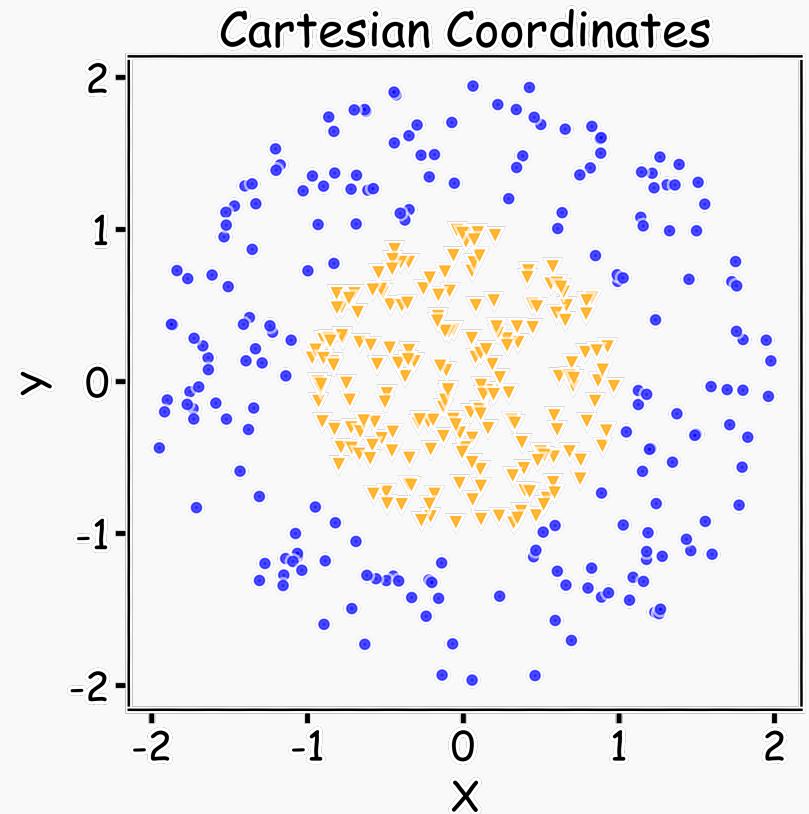


# Layers



# Why layers?

Representation matters!

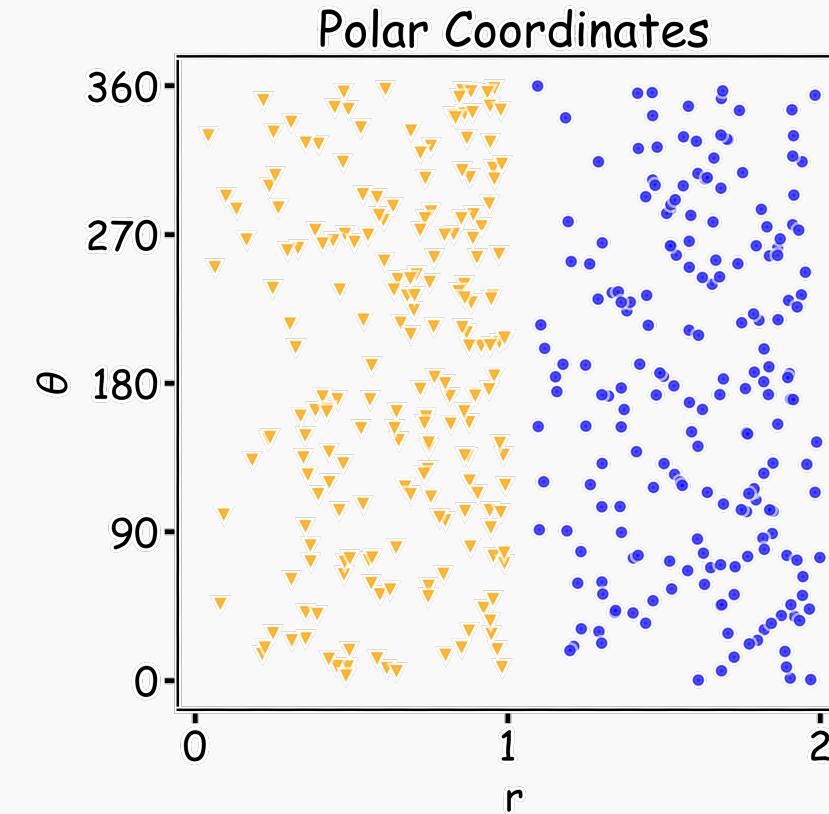
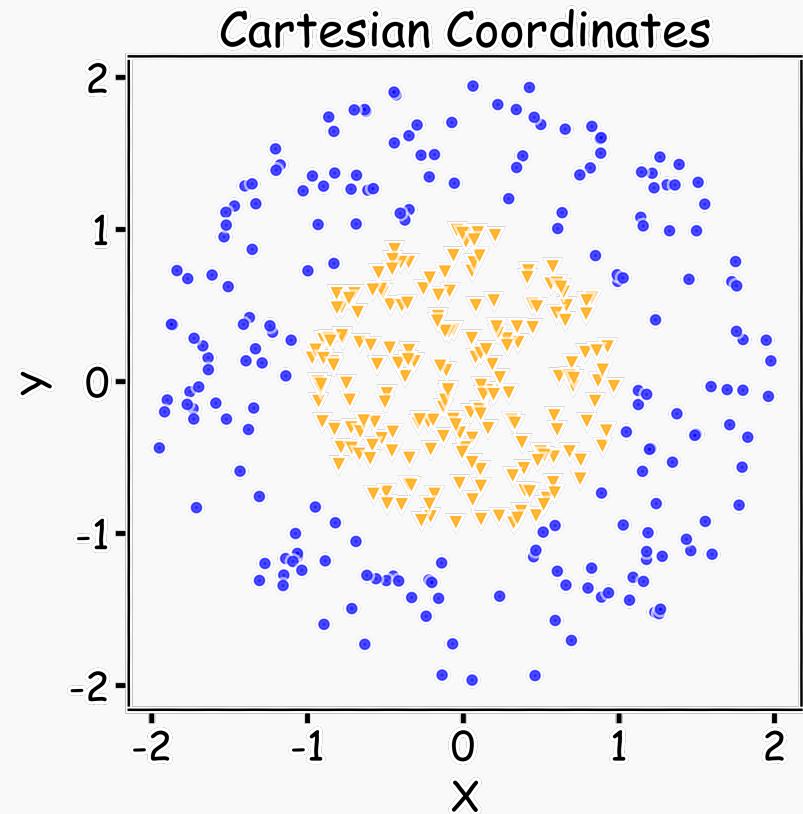


Neural networks can **learn useful representations** for the problem. This is another reason why they can be so powerful!



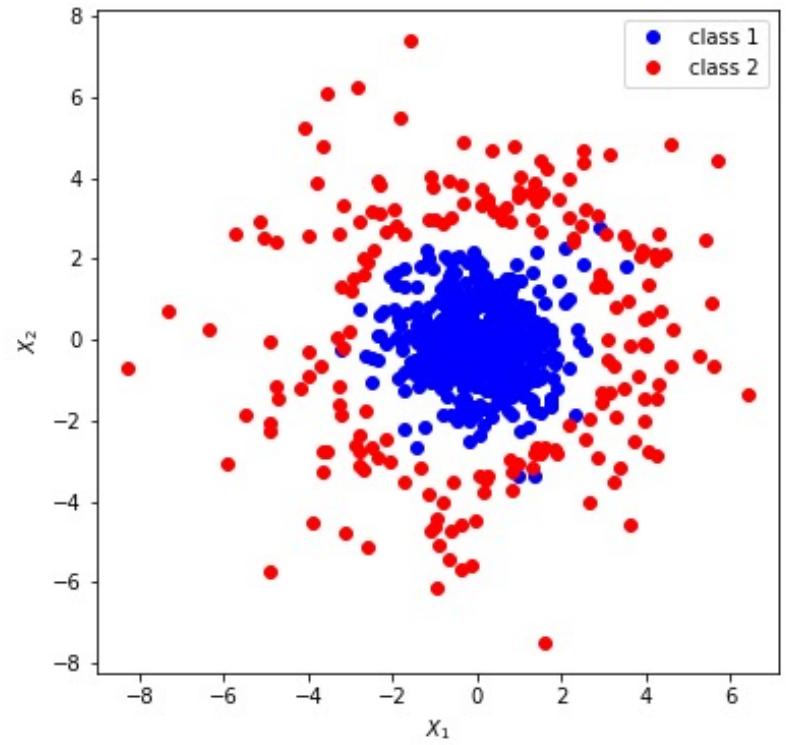
# Why layers?

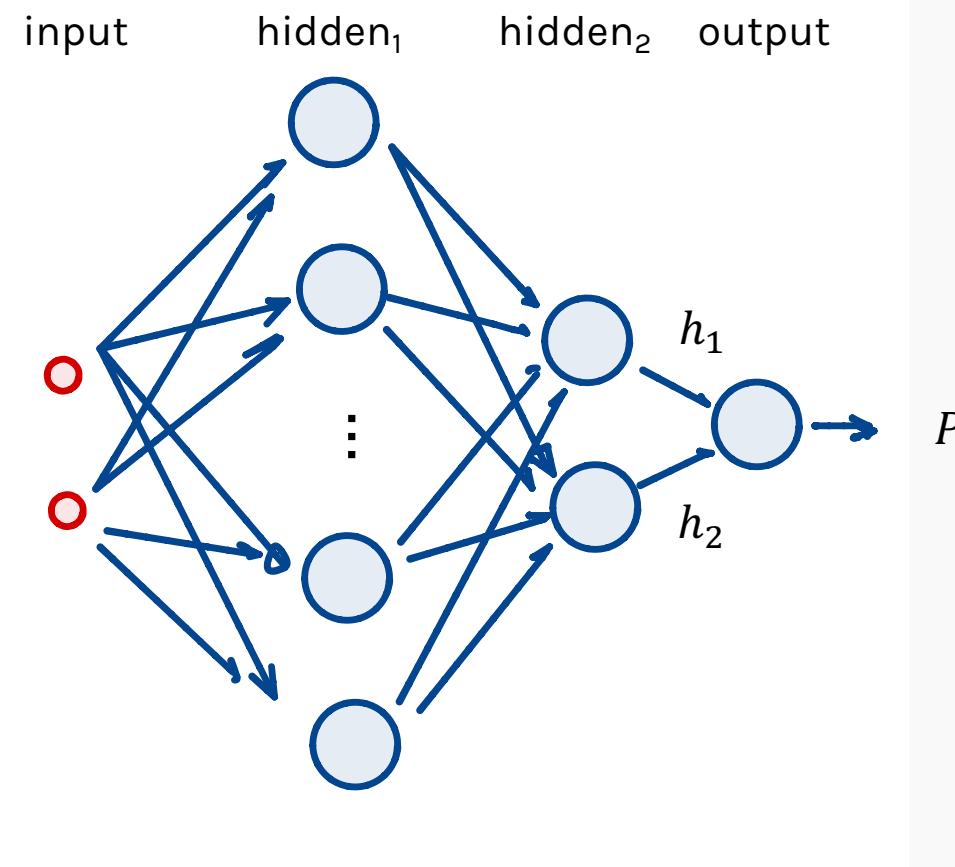
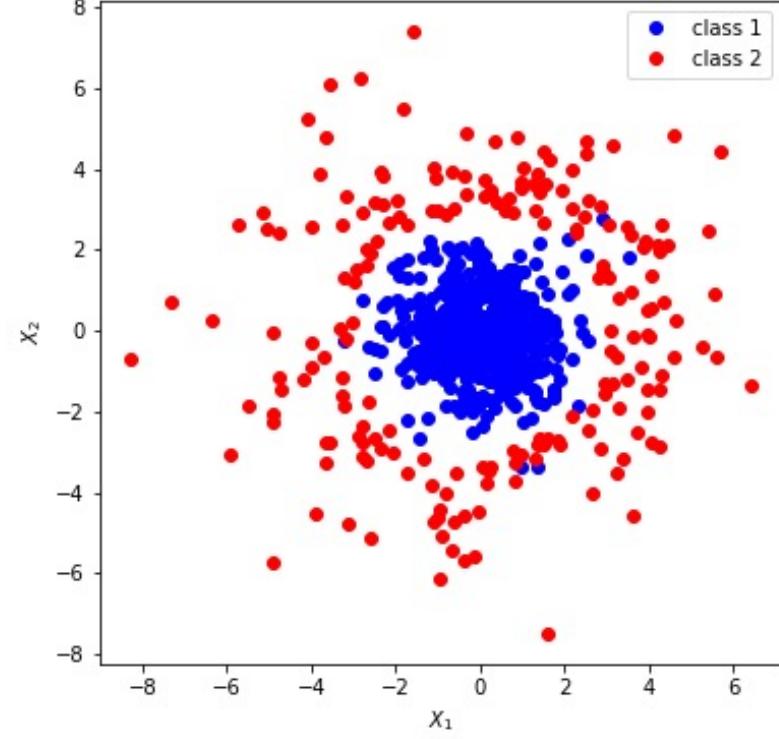
Representation matters!

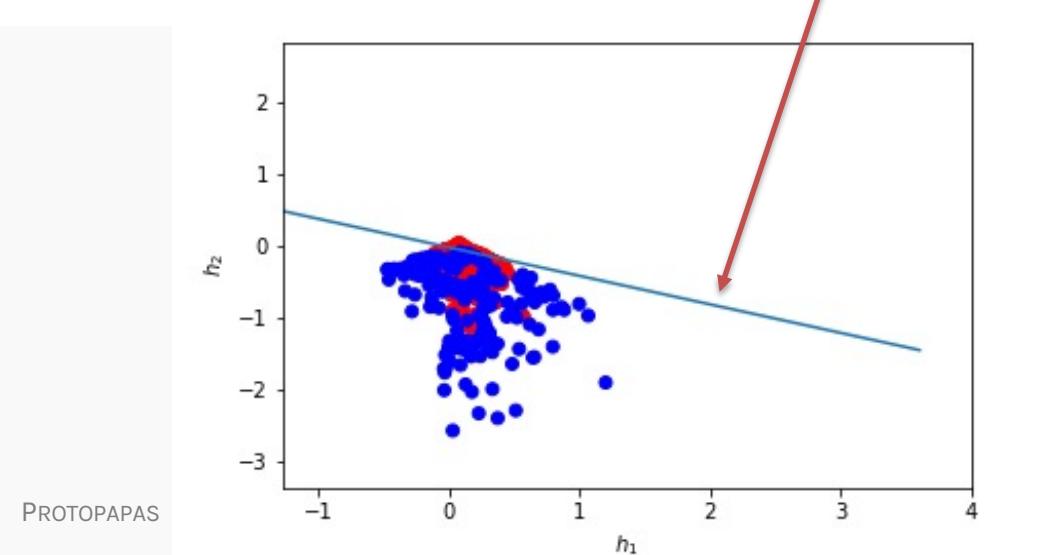
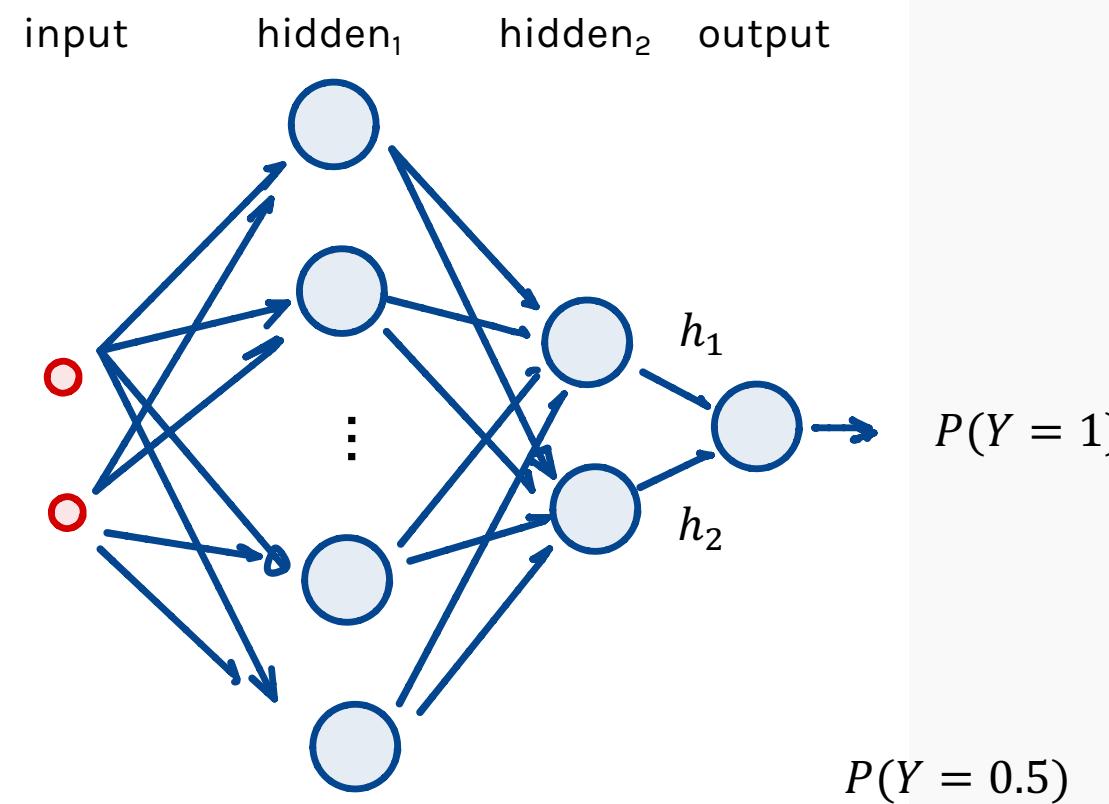
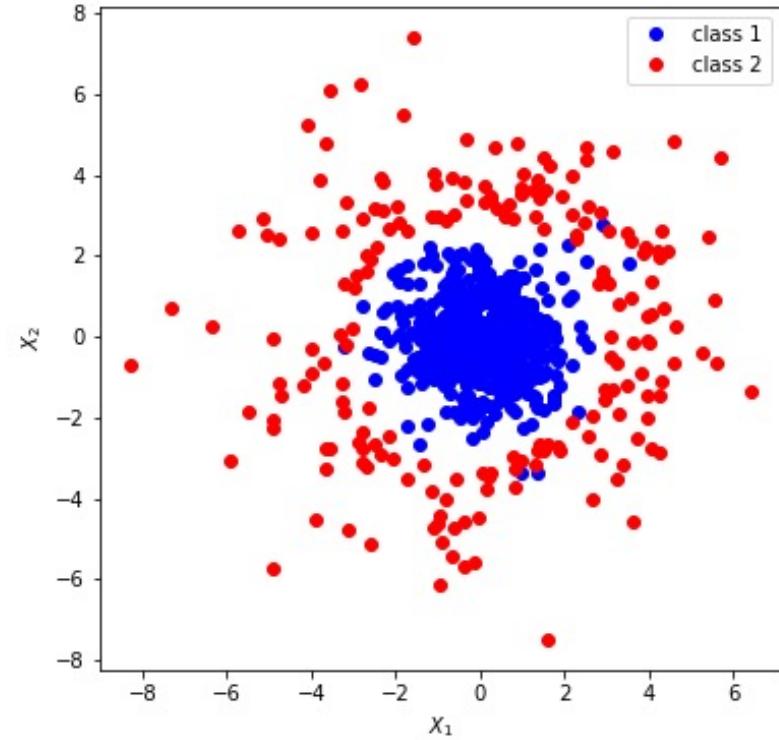


Neural networks can **learn useful representations** for the problem. This is another reason why they can be so powerful!





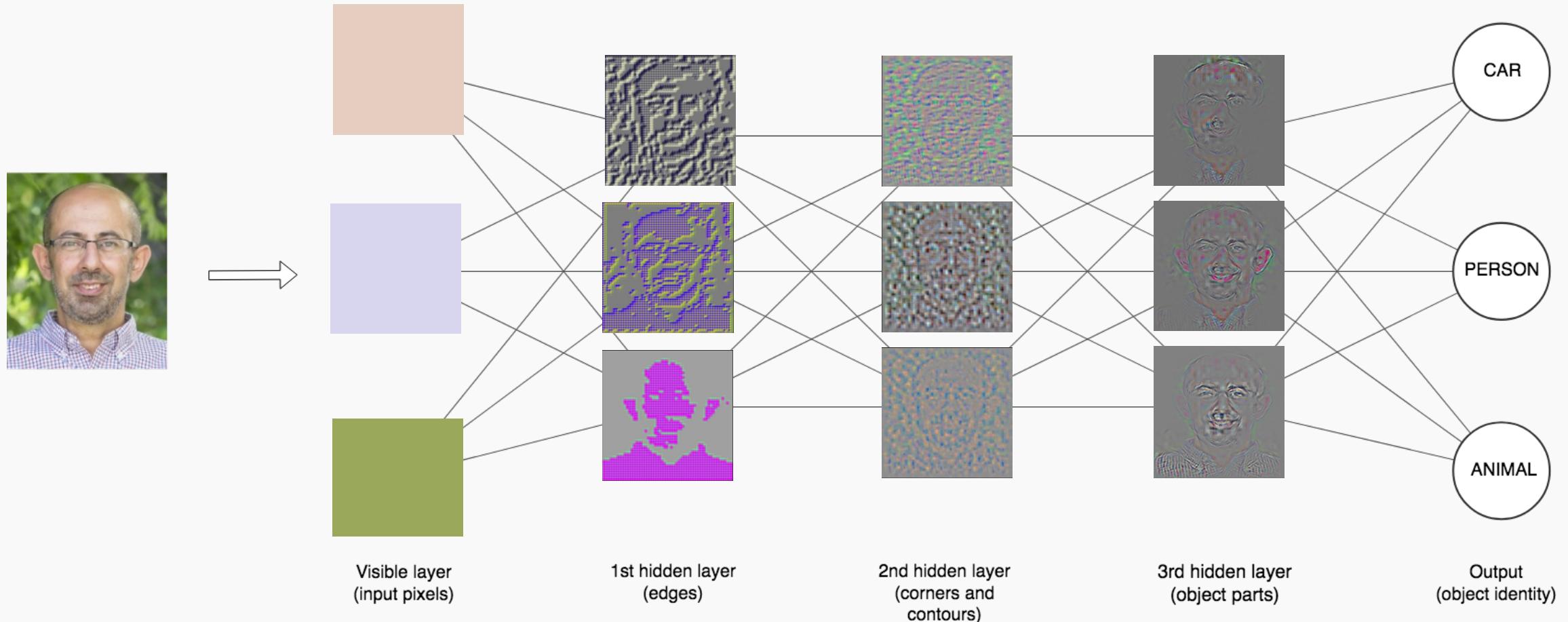




PROTOPAPAS

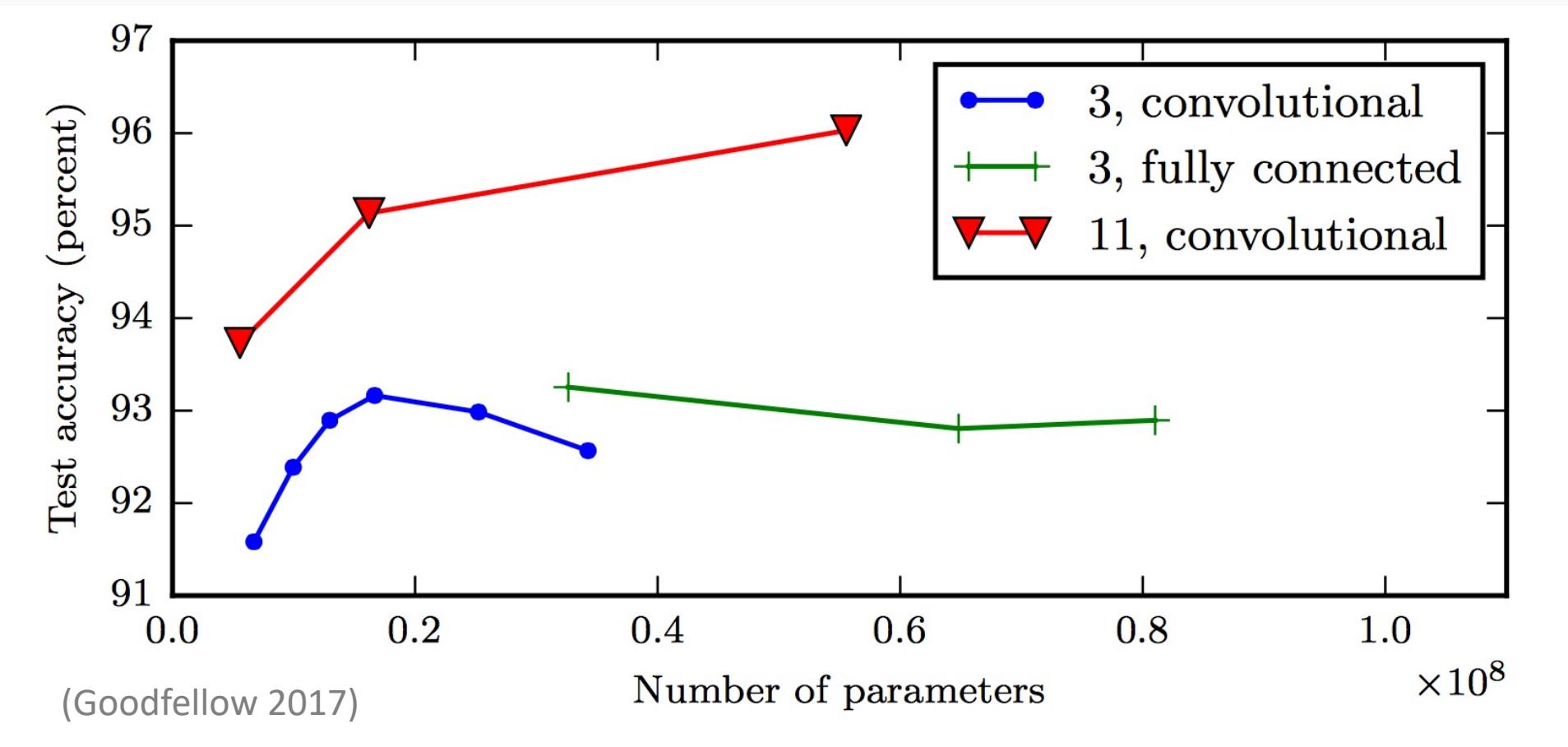


# Depth = Repeated Compositions



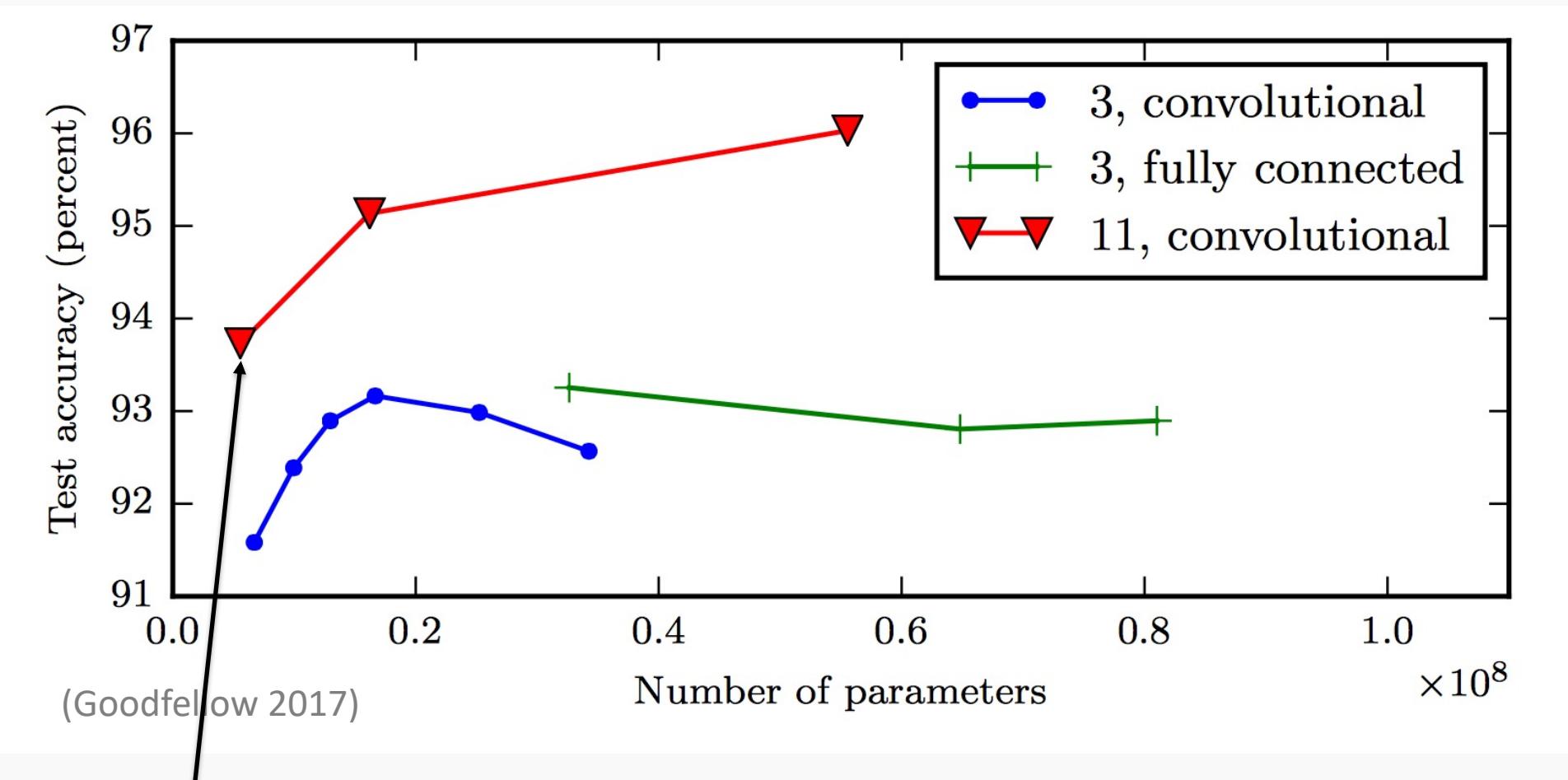
# Shallow Nets Overfit More

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



# Shallow Nets Overfit More

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

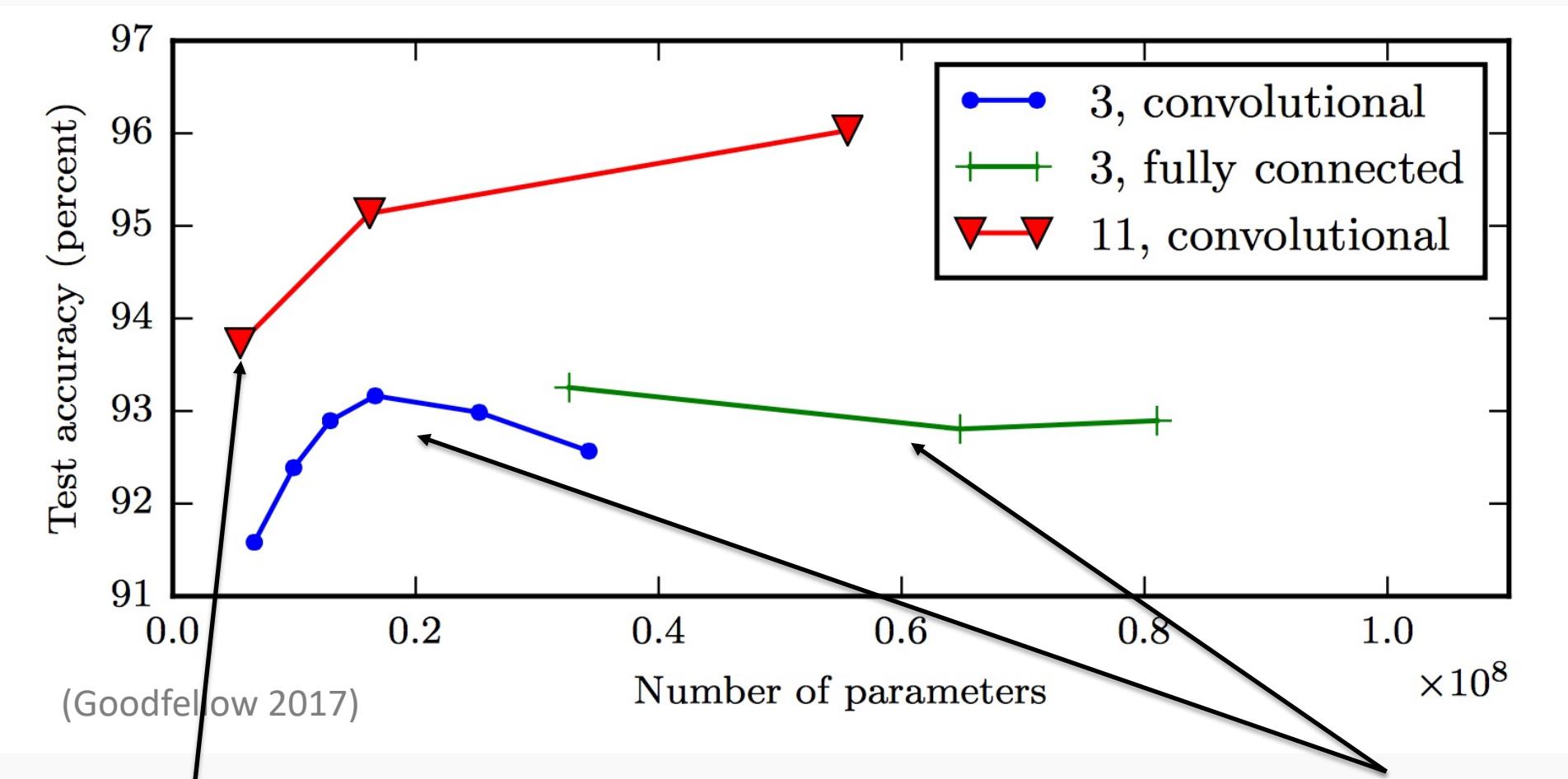


The **11-layer net** generalizes better on the test set when controlling for number of parameters.



# Shallow Nets Overfit More

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



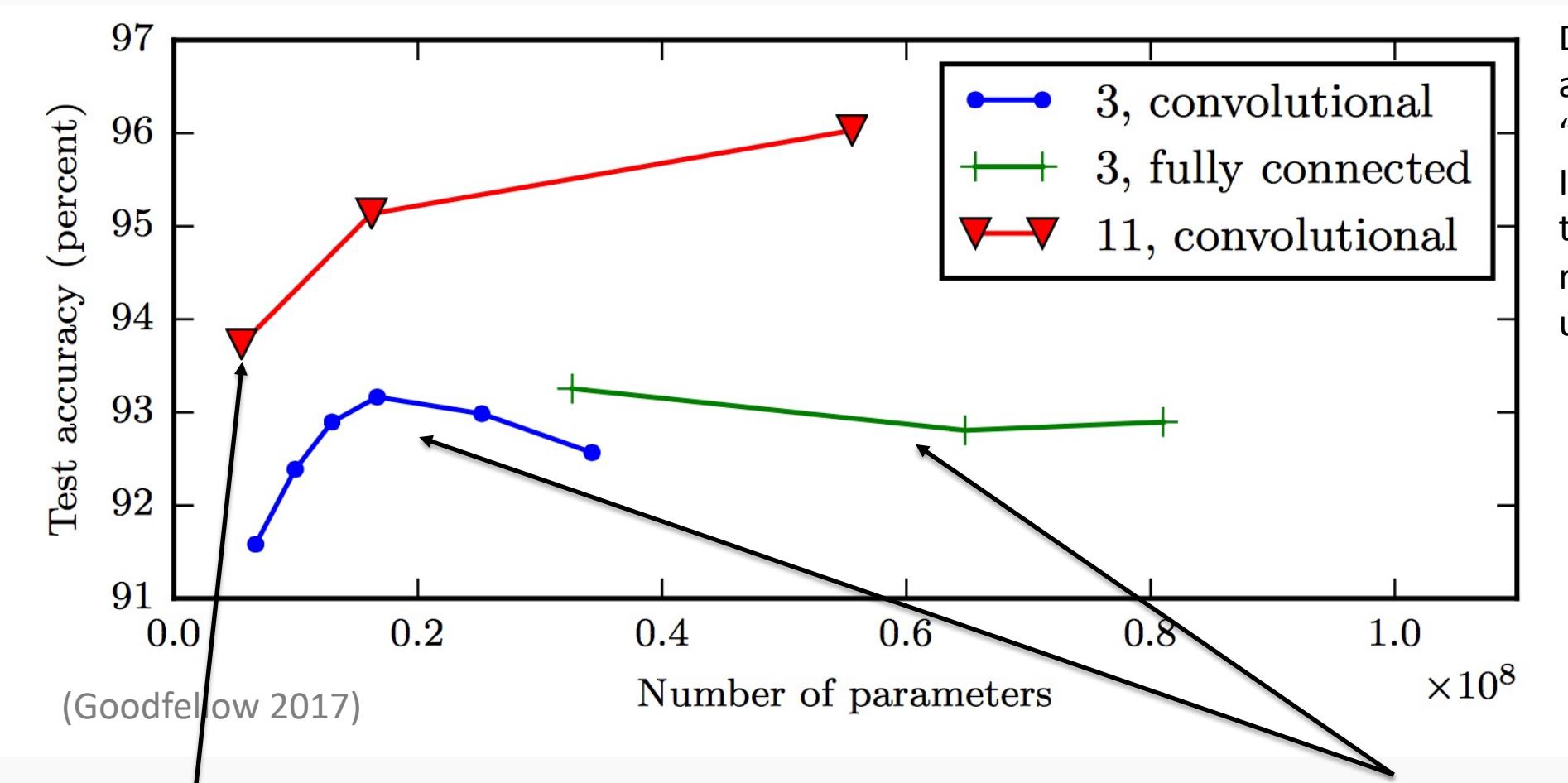
The **11-layer net** generalizes better on the test set when controlling for number of parameters.

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



# Shallow Nets Overfit More

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



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.

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

