Anatomy of Neural Networks and Design Choices

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

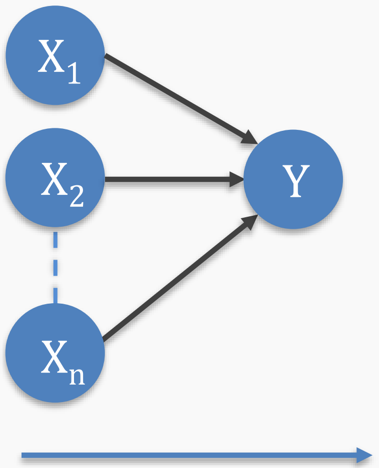
We build complex functions by composing simple functions of the form:

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

where f is the activation function.

Like sigmoid!

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

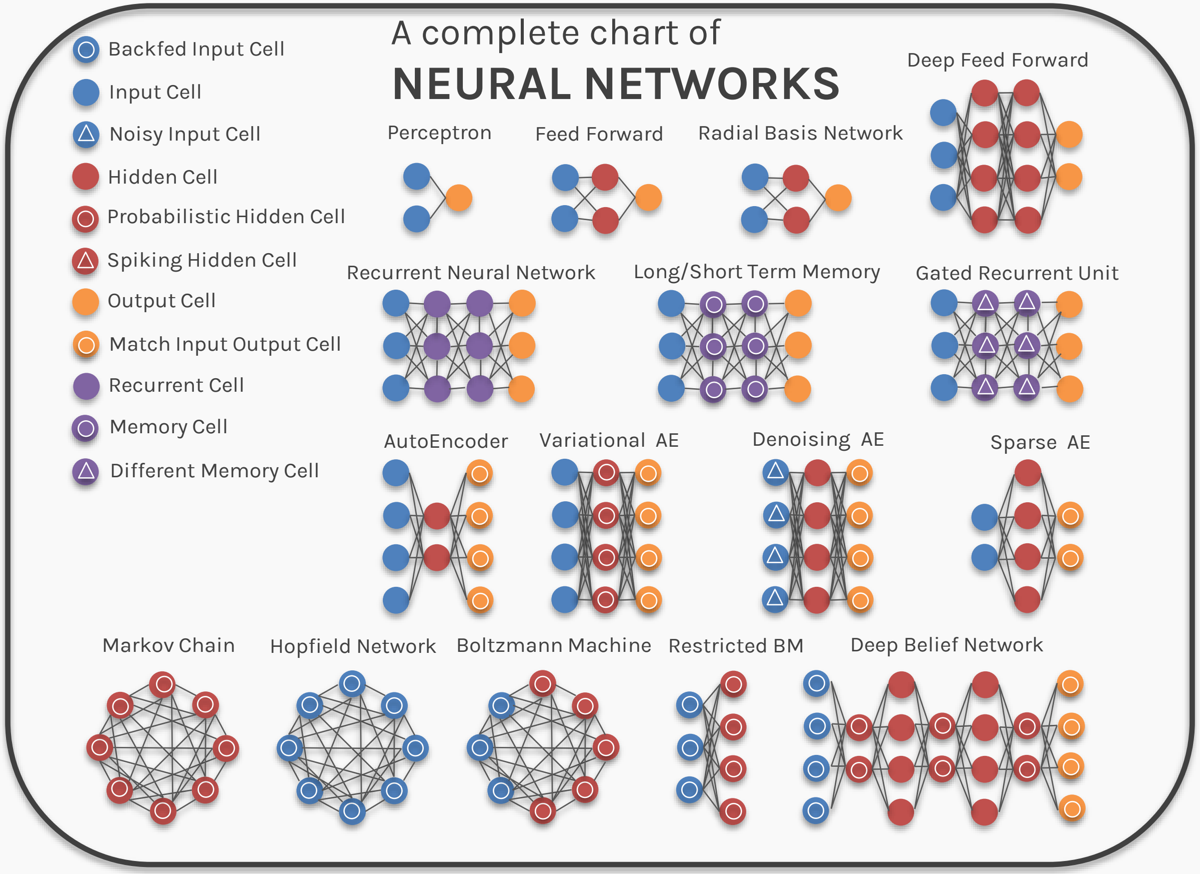


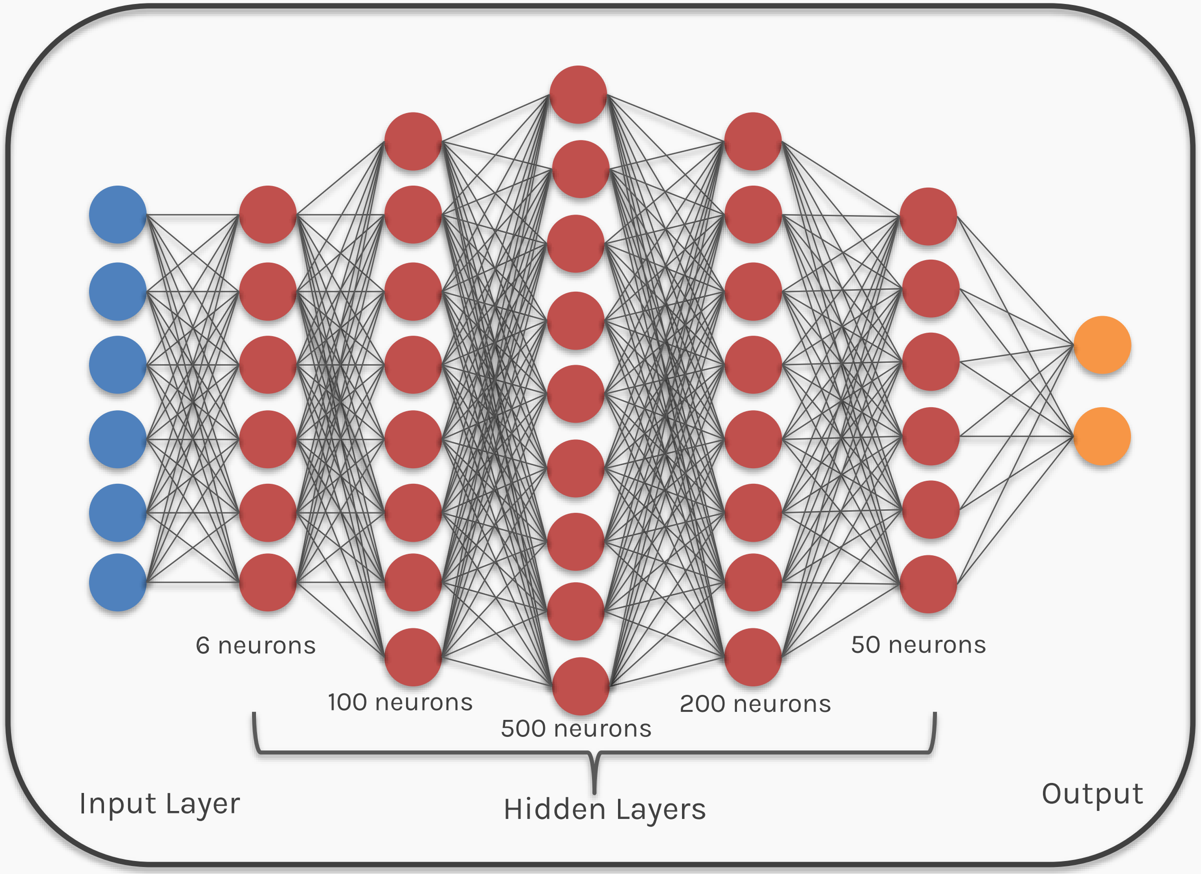
Read left to right →

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

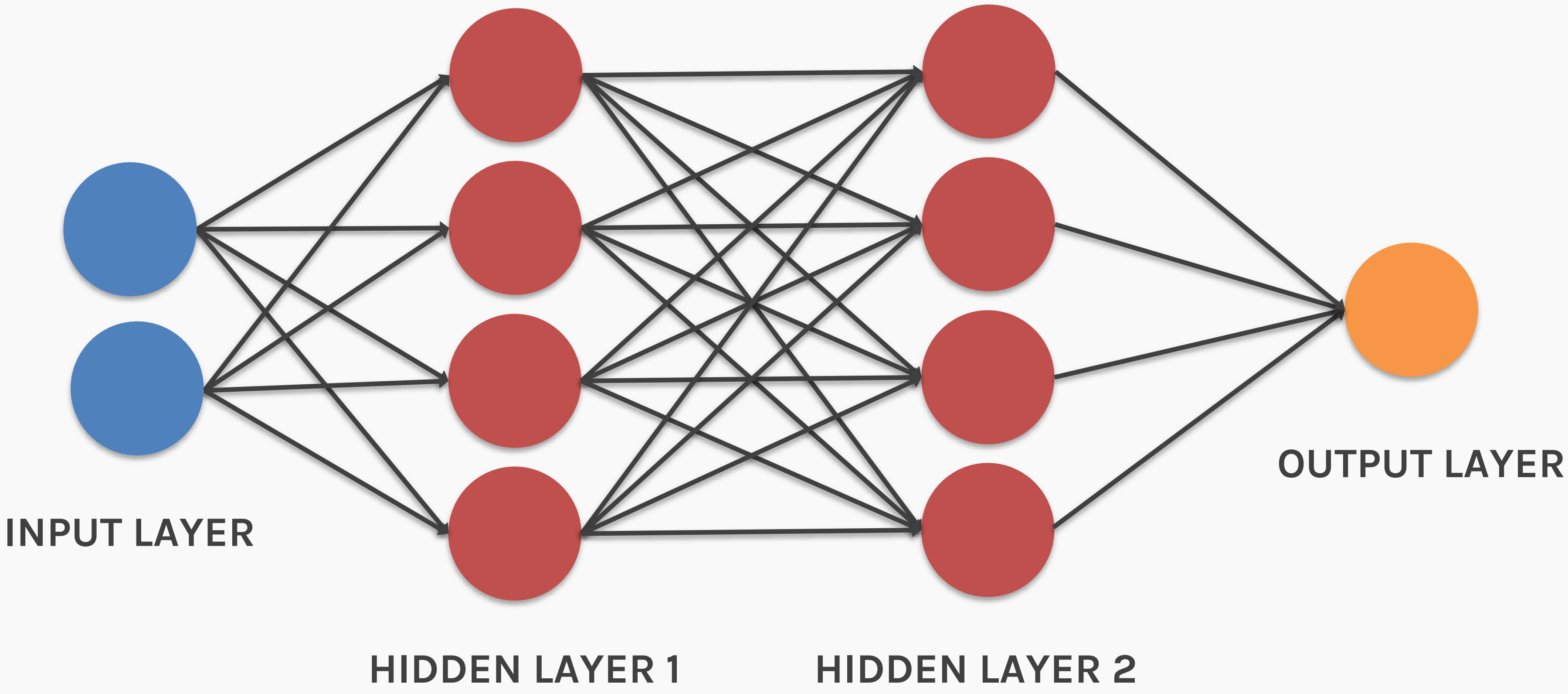




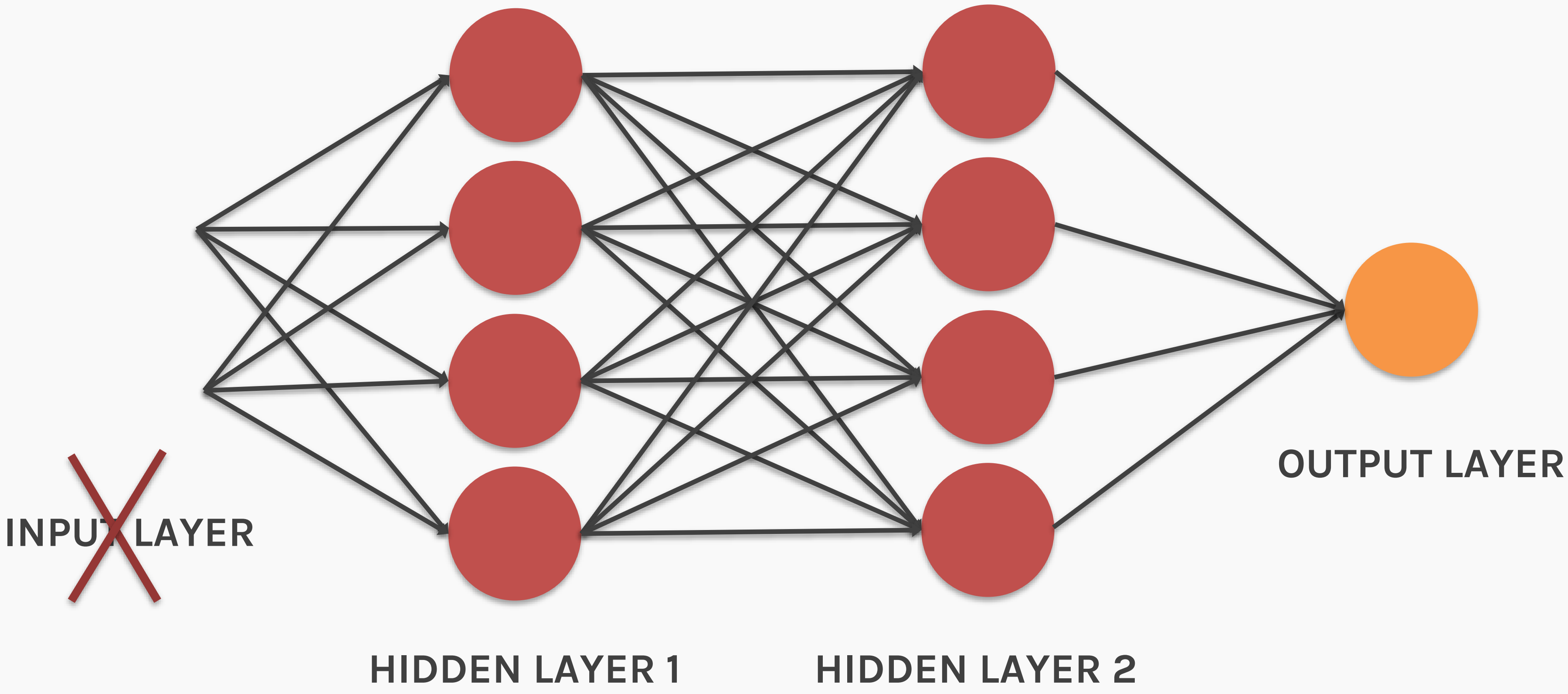
Different architectures result into functions with very different properties.

Larger networks can express more complex functions

Anatomy of artificial neural network (ANN)

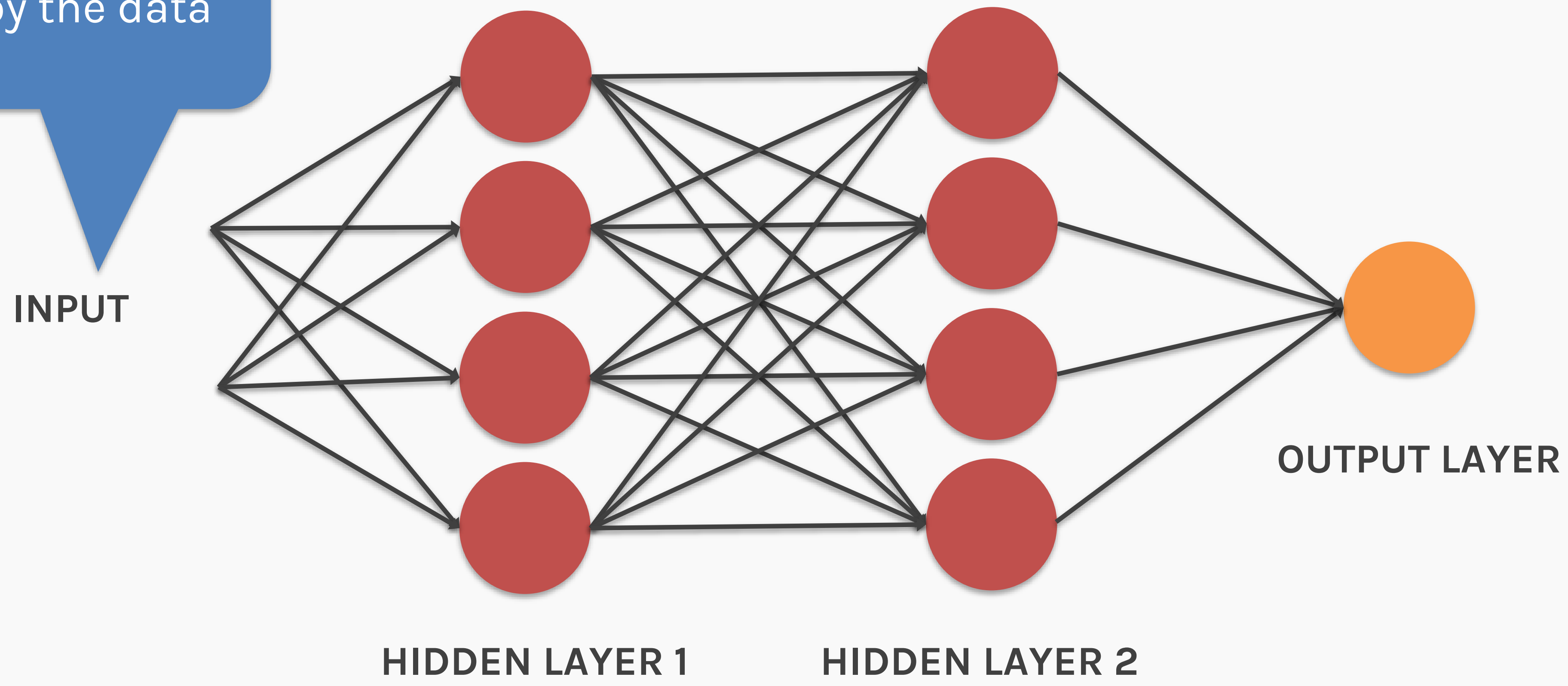


Anatomy of artificial neural network (ANN)

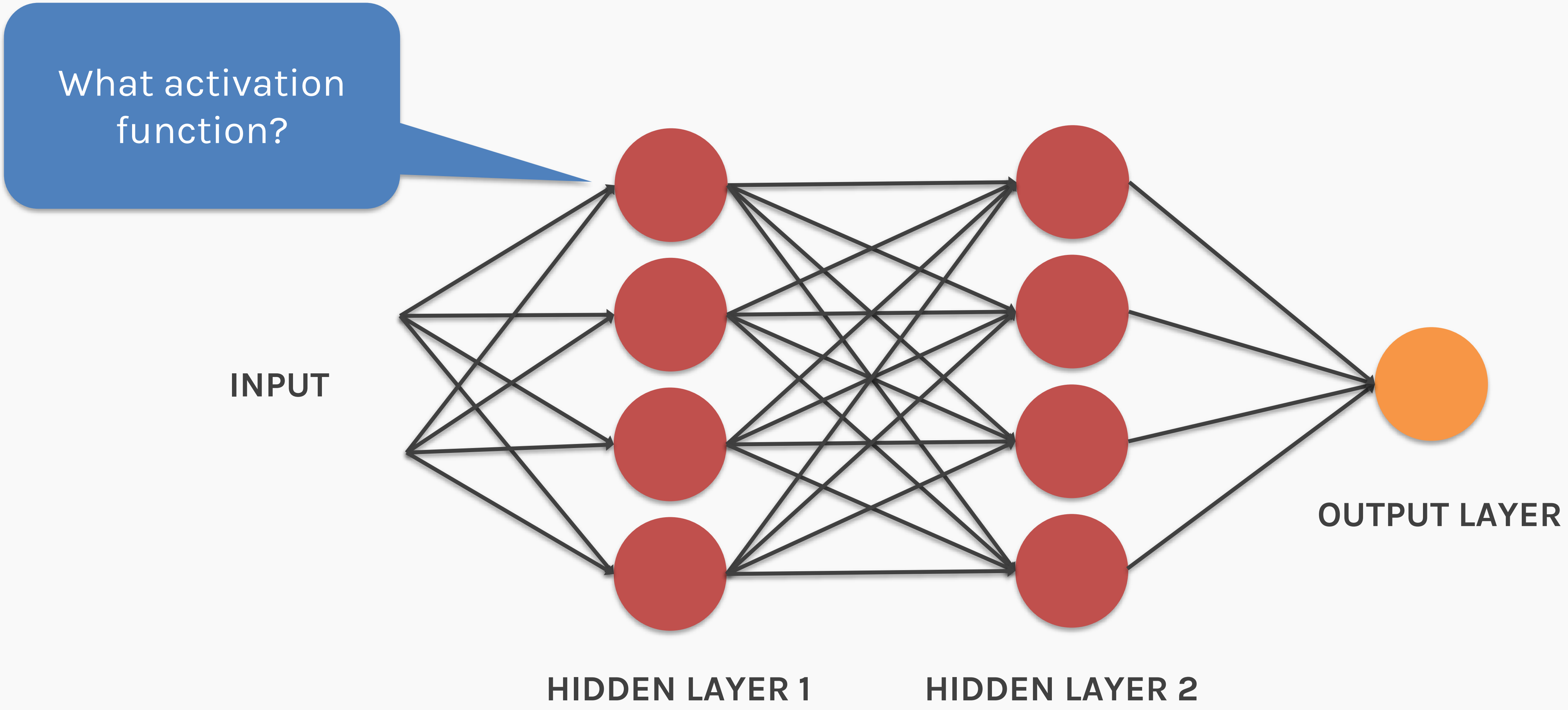


Anatomy of artificial neural network (ANN)

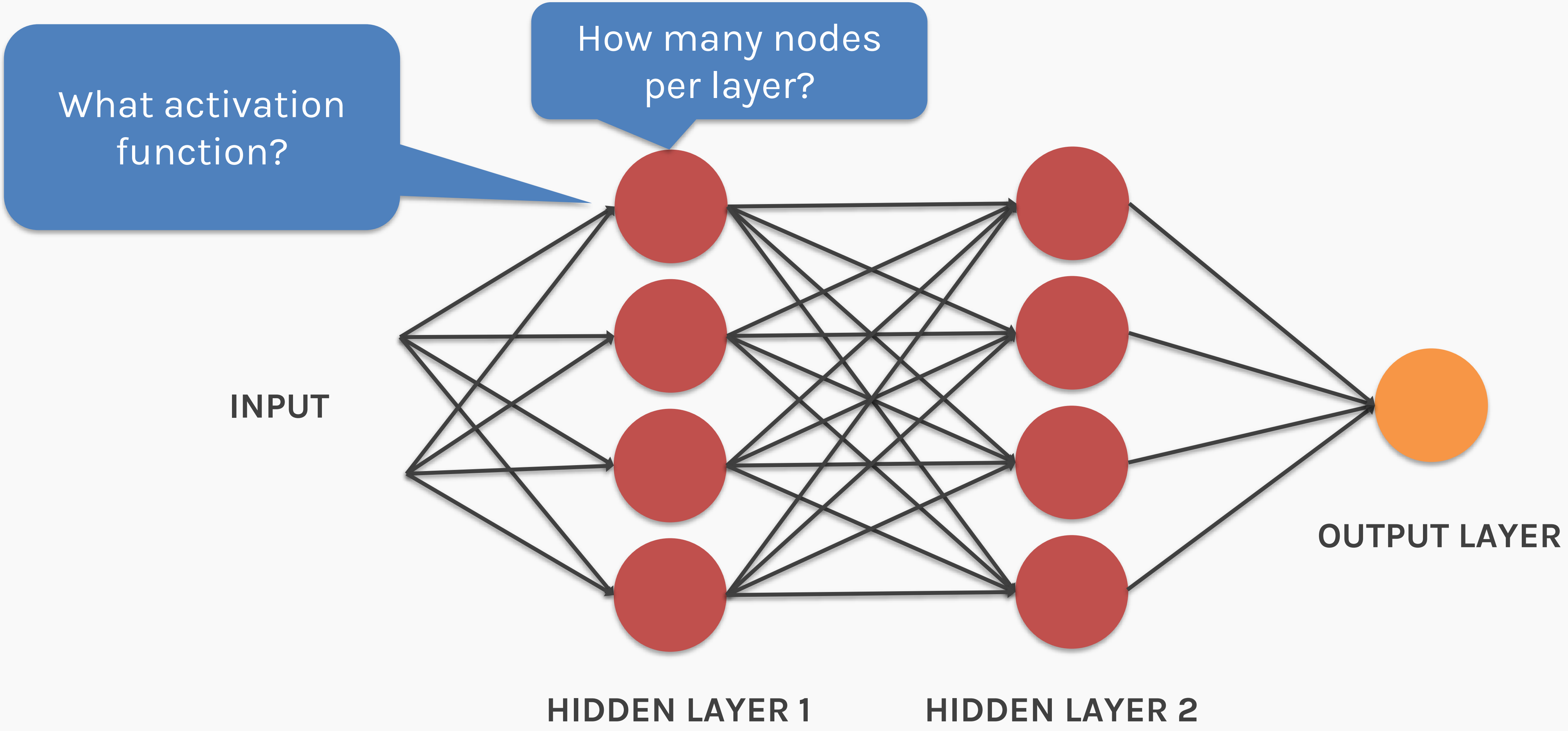
Size of the **INPUT** is specified by the data



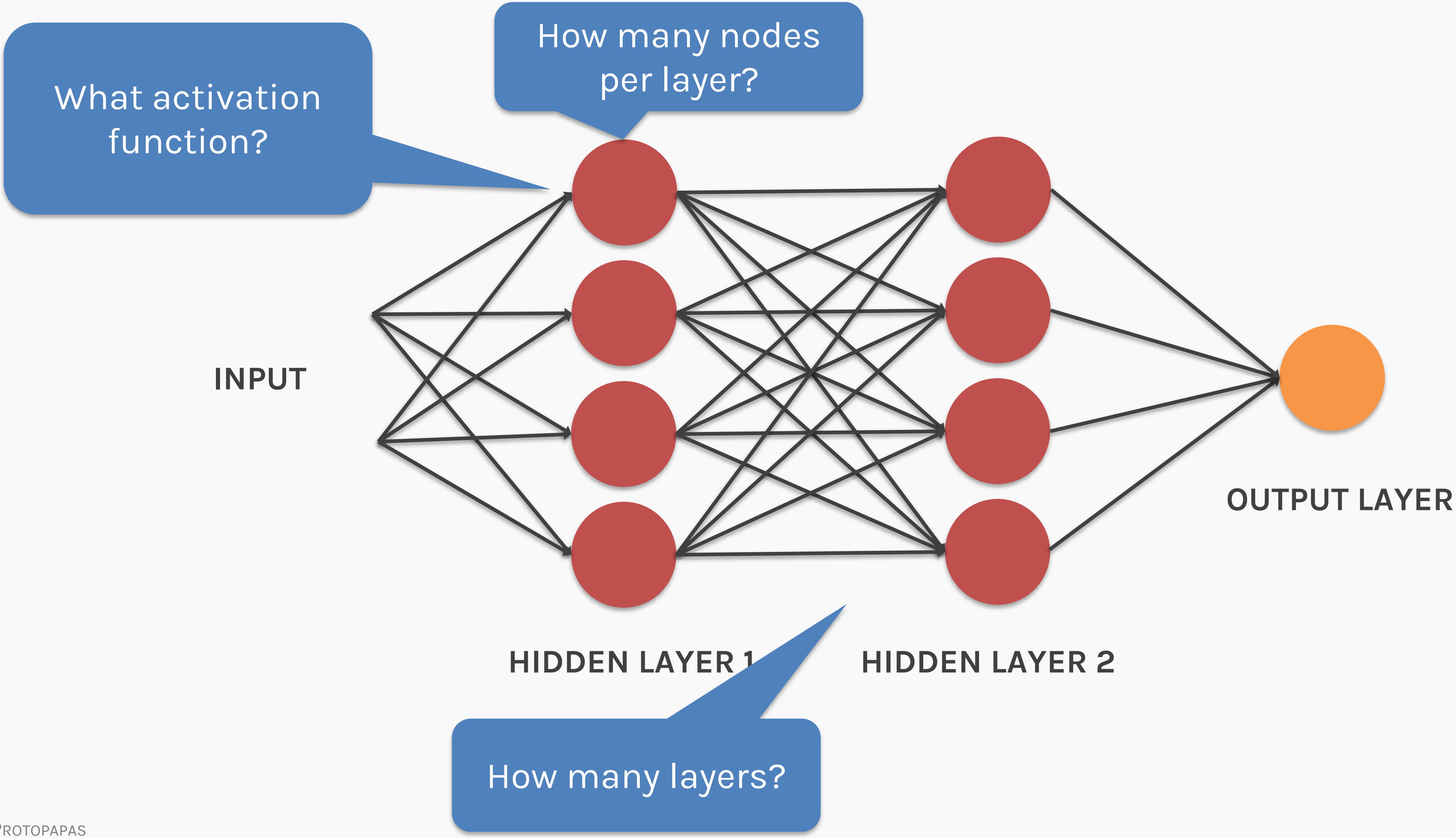
Anatomy of artificial neural network (ANN)



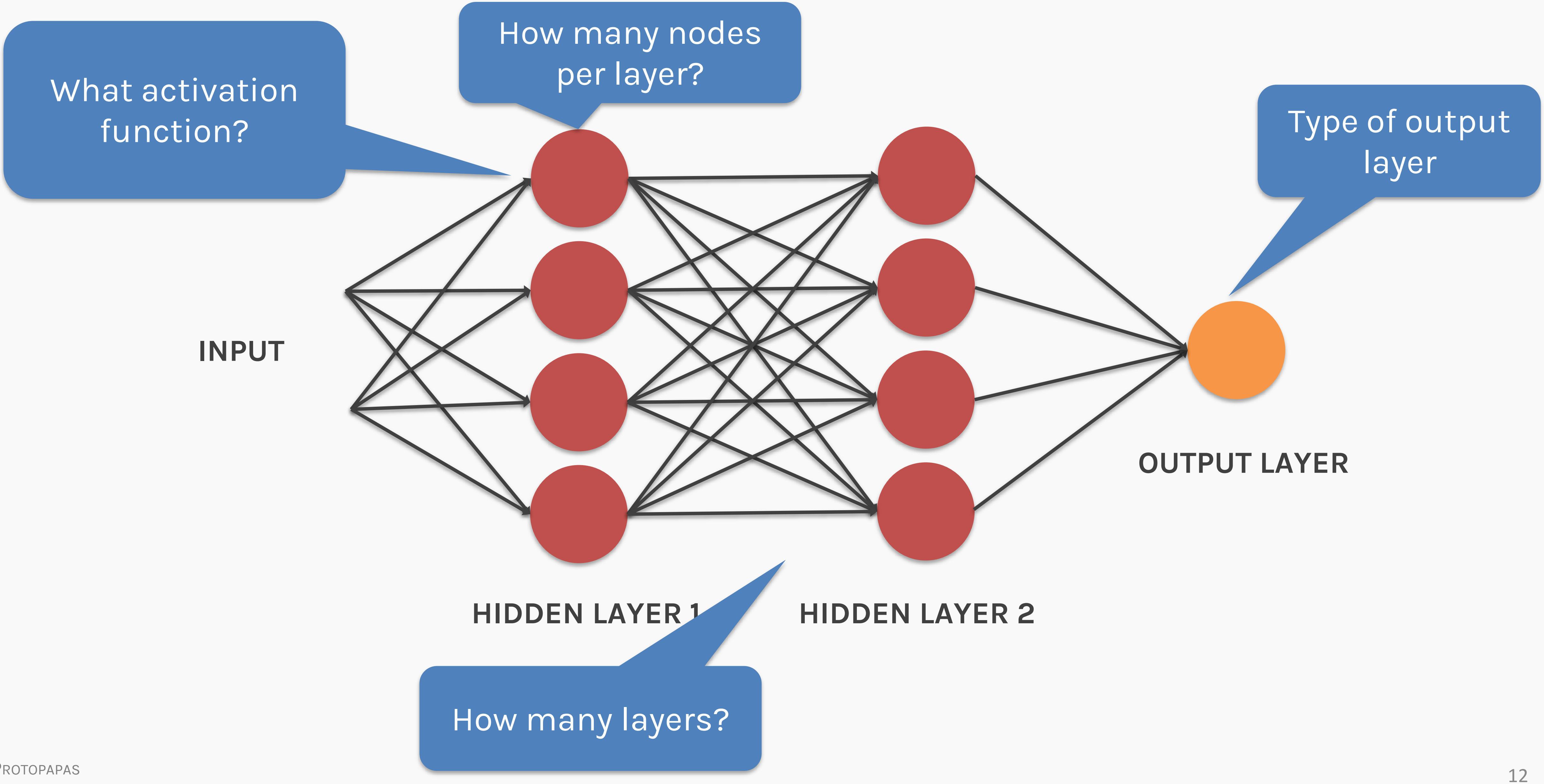
Anatomy of artificial neural network (ANN)



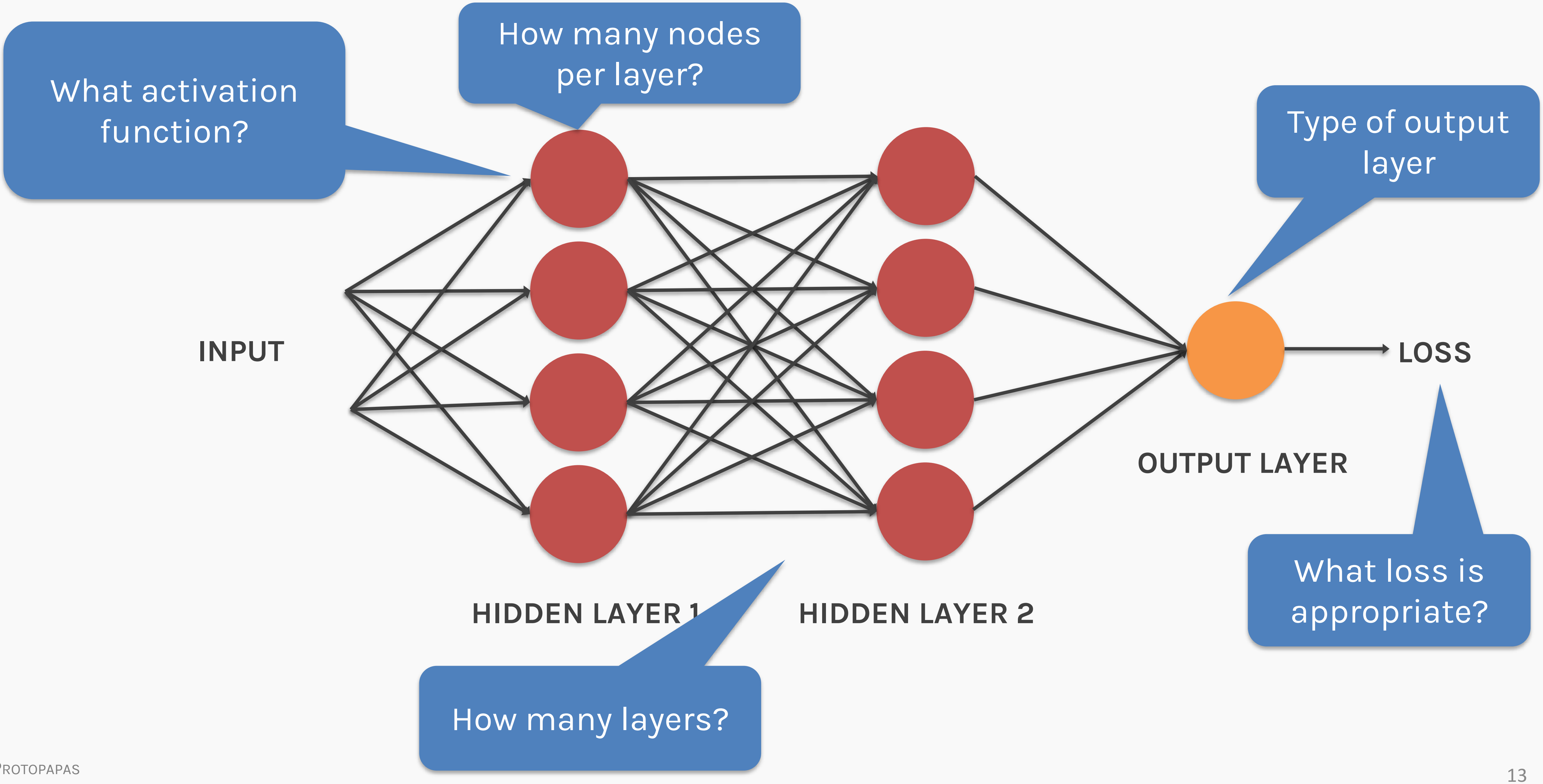
Anatomy of artificial neural network (ANN)



Anatomy of artificial neural network (ANN)



Anatomy of artificial neural network (ANN)



Design Choices

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

Activation Function

- Activation functions are like traffic officers, managing the flow of information in neural networks.

- They assess if a signal is strong enough to proceed, ensuring only relevant data moves forward.

- Acting as gatekeepers, they determine whether a neuron activates or stays silent based on incoming signals.

Quiz Time

According to you, how should an activation function be structured? (Select all that apply)

A. It should be non-linear

B. It should be simple

C. It should ensure that the gradients remain large

D. It should restrict the range of the outputs

Quiz Time

According to you, how should an activation function be structured? (Select all that apply)

A. It should be non-linear

B. It should be simple

C. It should ensure that the gradients remain large

D. It should restrict the range of the outputs

Activation function for hidden layers (not for the output unit)

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

The activation function should:

- Simple (non-complex).
- Provide non-linearity.
- Ensure gradients remain large through hidden units.

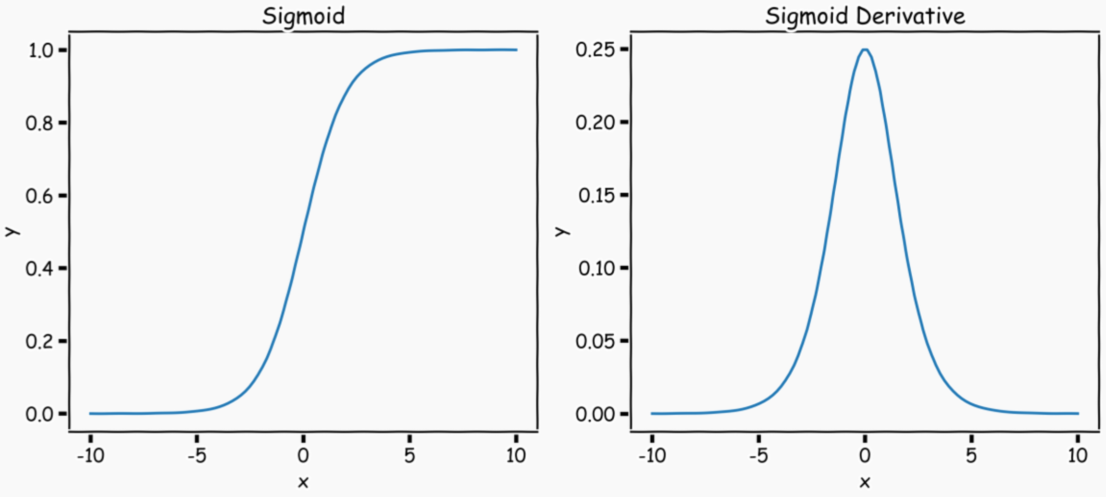
Common choices are

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

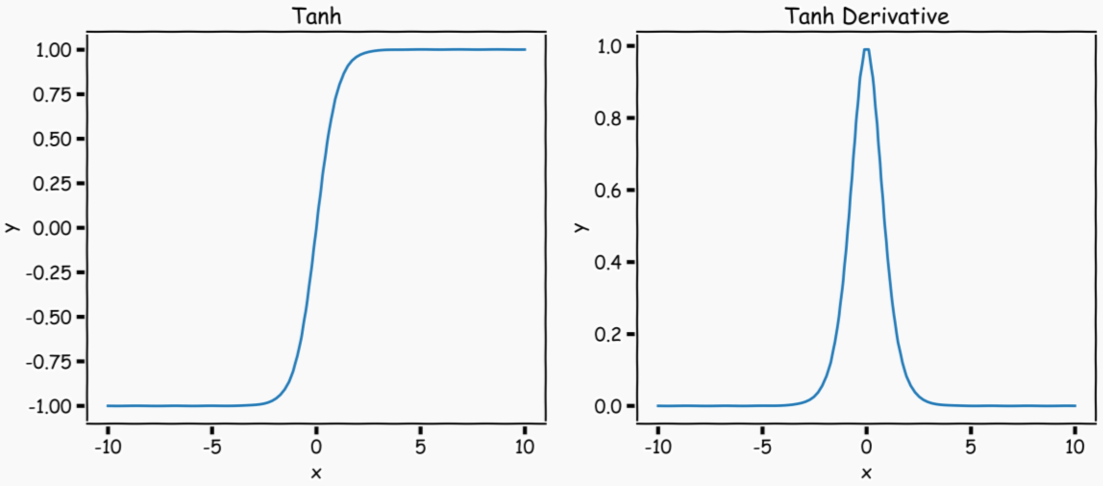
Swish is called SiLU in Pytorch

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

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



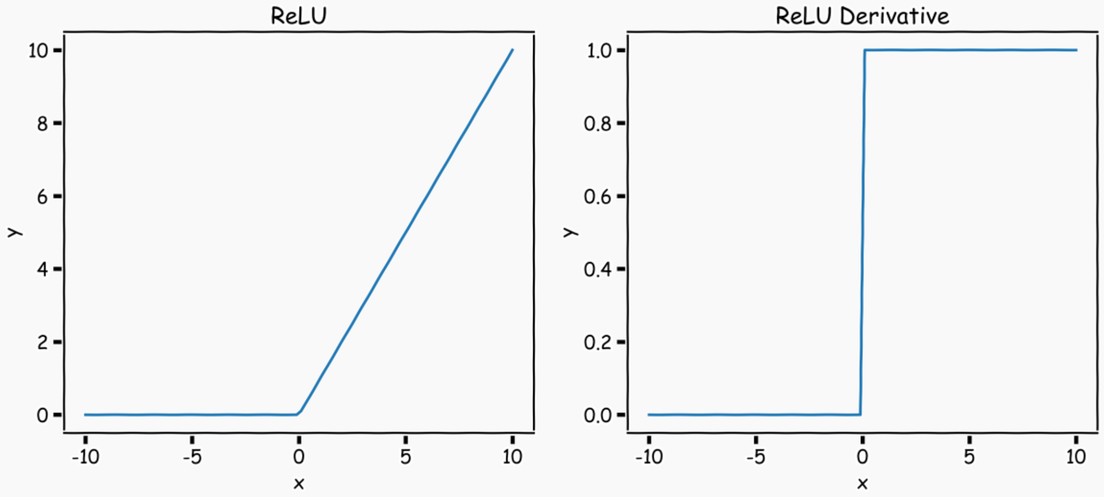
$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



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

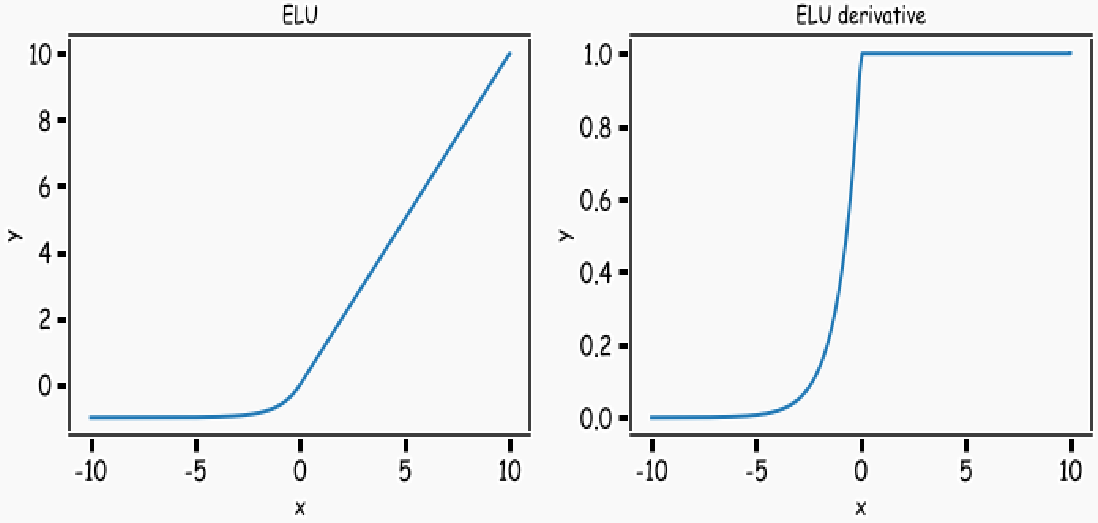
Rectified Linear Unit, ReLU(), Exponential ReLU (ELU)

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



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

where α takes a small value



Two major advantages:

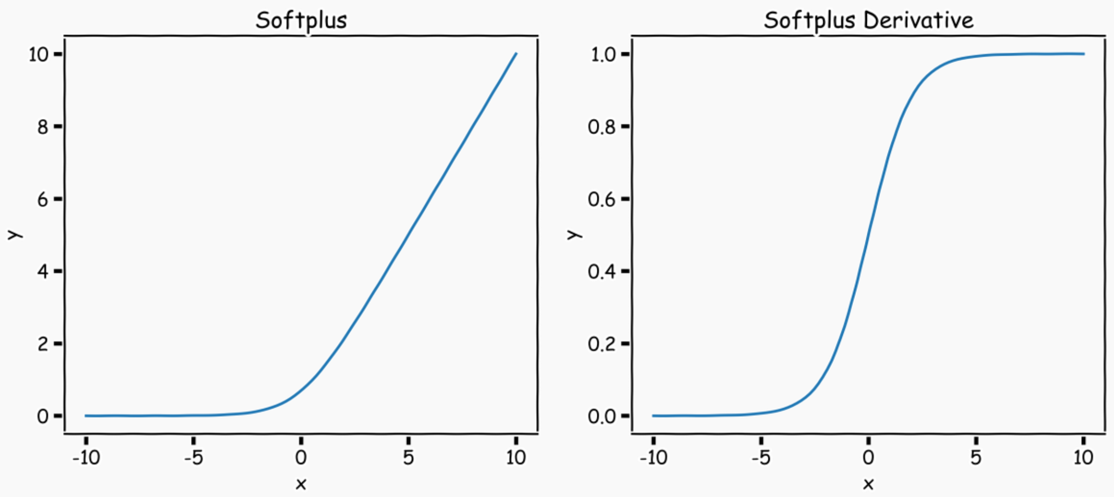
1. No vanishing gradient when x > 0
2. Provides sparsity (regularization) since y = 0 when x < 0

Less vanishing gradients and easy to calculate.

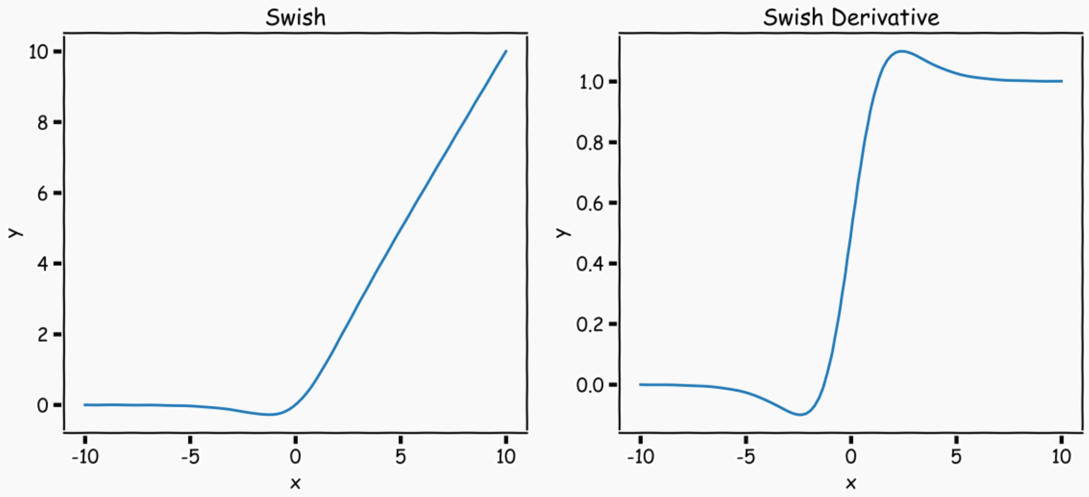
Note: Graph above is shown from α=1

Softplus and Swish

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



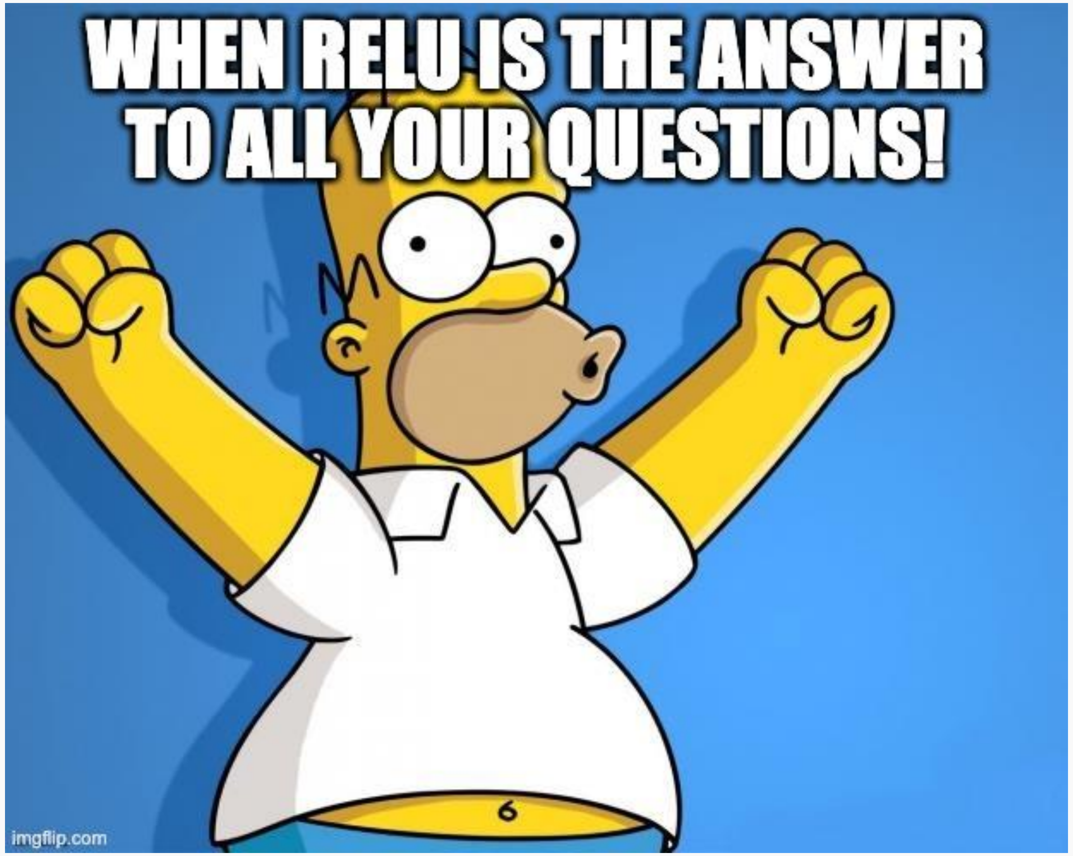
$$y = x\,\sigma(x)$$



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 several challenging datasets.

TL; DR



Activation Function Demo!

Visualising Activation Functions in Neural Networks - David Sheehan

Design Choices

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

Loss Function

TL;DR

- Regression: MSE

$$\mathcal{L}(W; X, Y) = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$$

- Binary Classification: Binary Cross Entropy

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

- Multi-class Classification: Cross Entropy

$$\mathcal{L}(W; X, Y) = -\frac{1}{n}\sum_i \sum_k I(y_i = k) \log p_{ik}$$

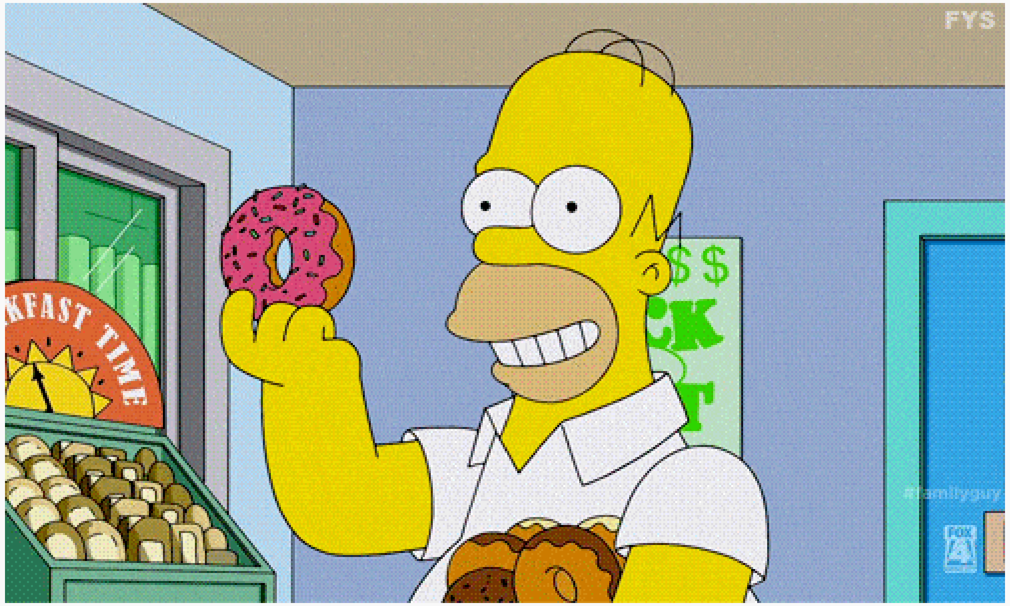
Design Choices

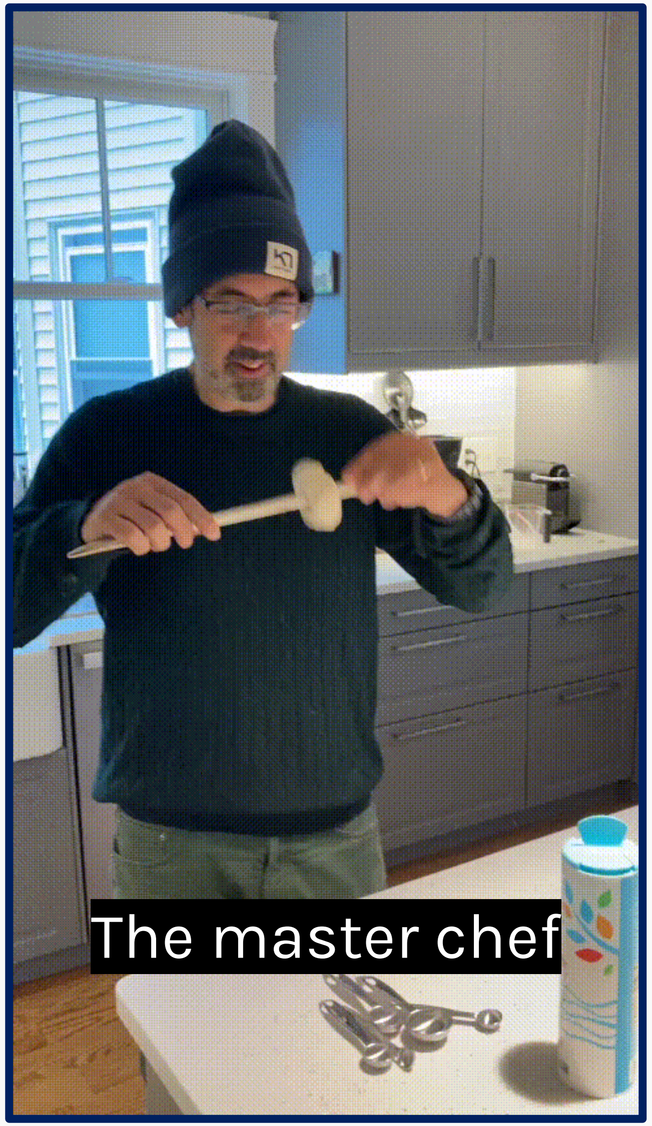
- Activation function
- Loss function
- **Output units**
- Architecture

Output Units

- Think of a neural network as a master chef preparing a donut.

- The output units are like the finished donuts that are served to the customers.

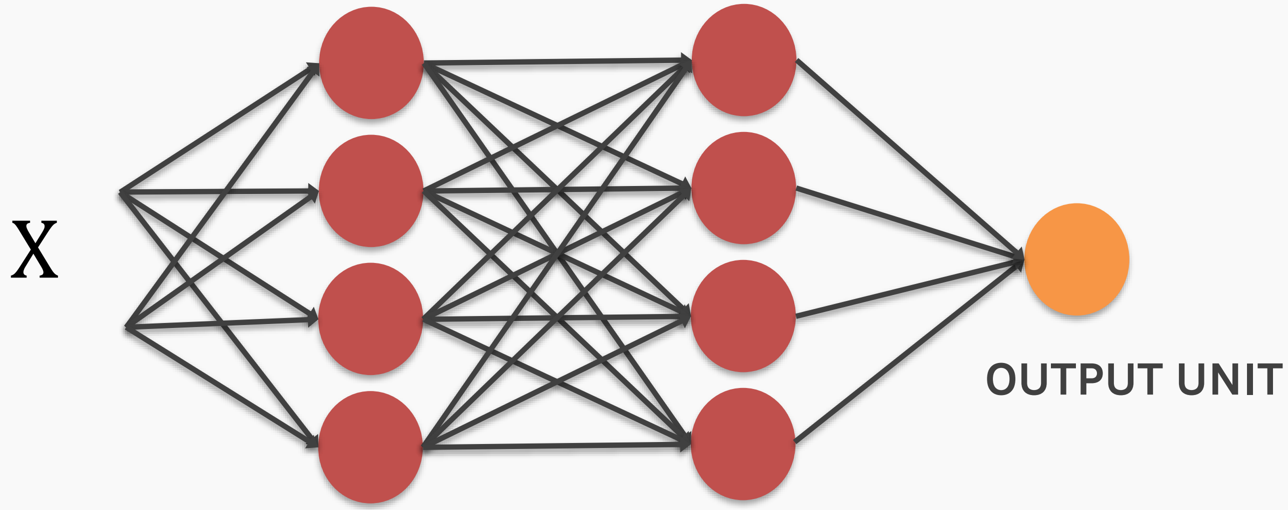




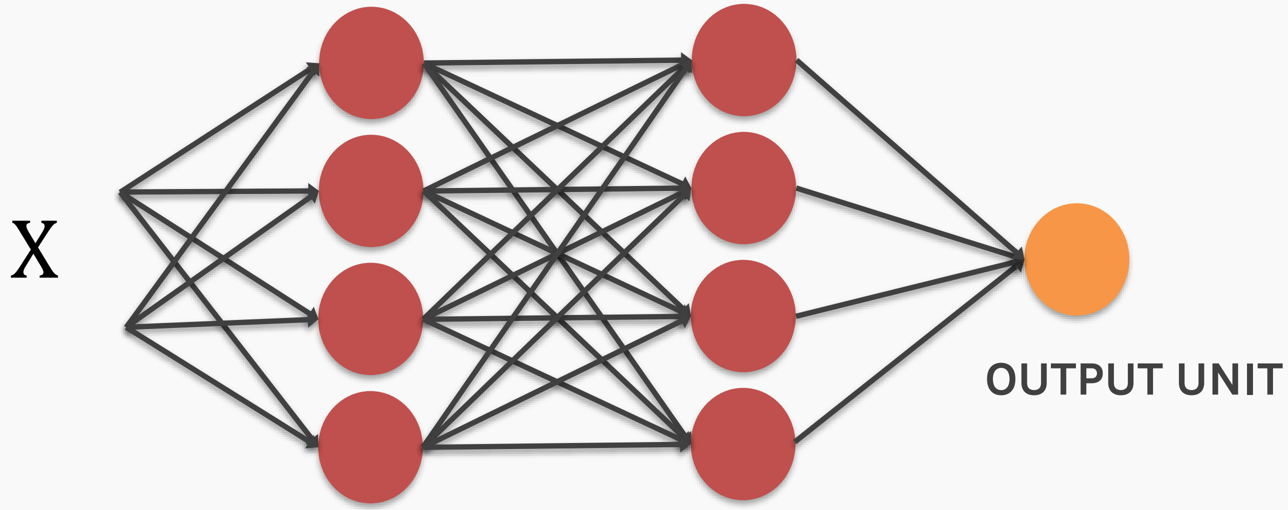
Output Units

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

Output Units - Binary Classification



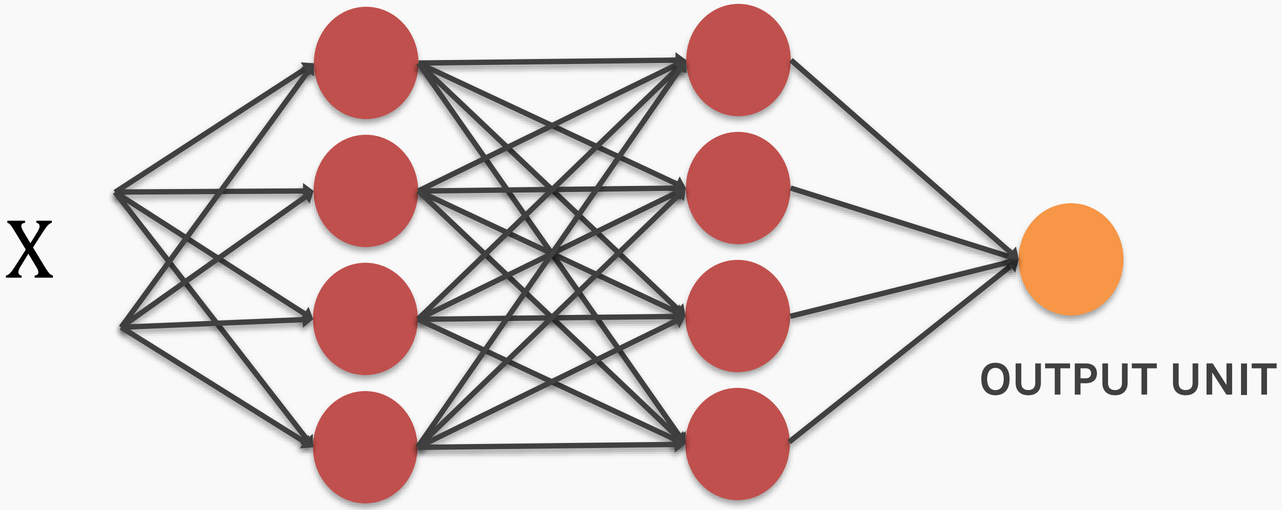
Output Units - Binary Classification



P(Y=1) must be [0,1]

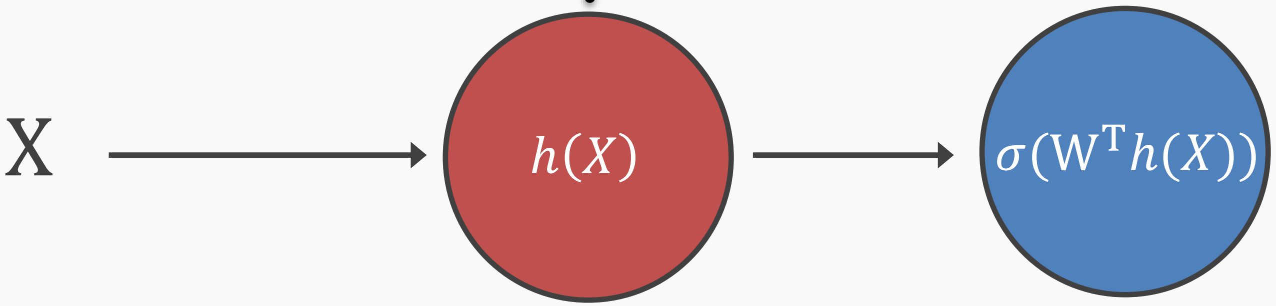
$$\hat{Y} = \mathrm{P}(y = 1)$$

Output Units - Binary Classification



P(Y=1) must be [0,1]

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



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

$$\boldsymbol{X \Longrightarrow h(X) \Longrightarrow P(y=1) = \frac{1}{1+e^{-W^T h(X)}}}$$

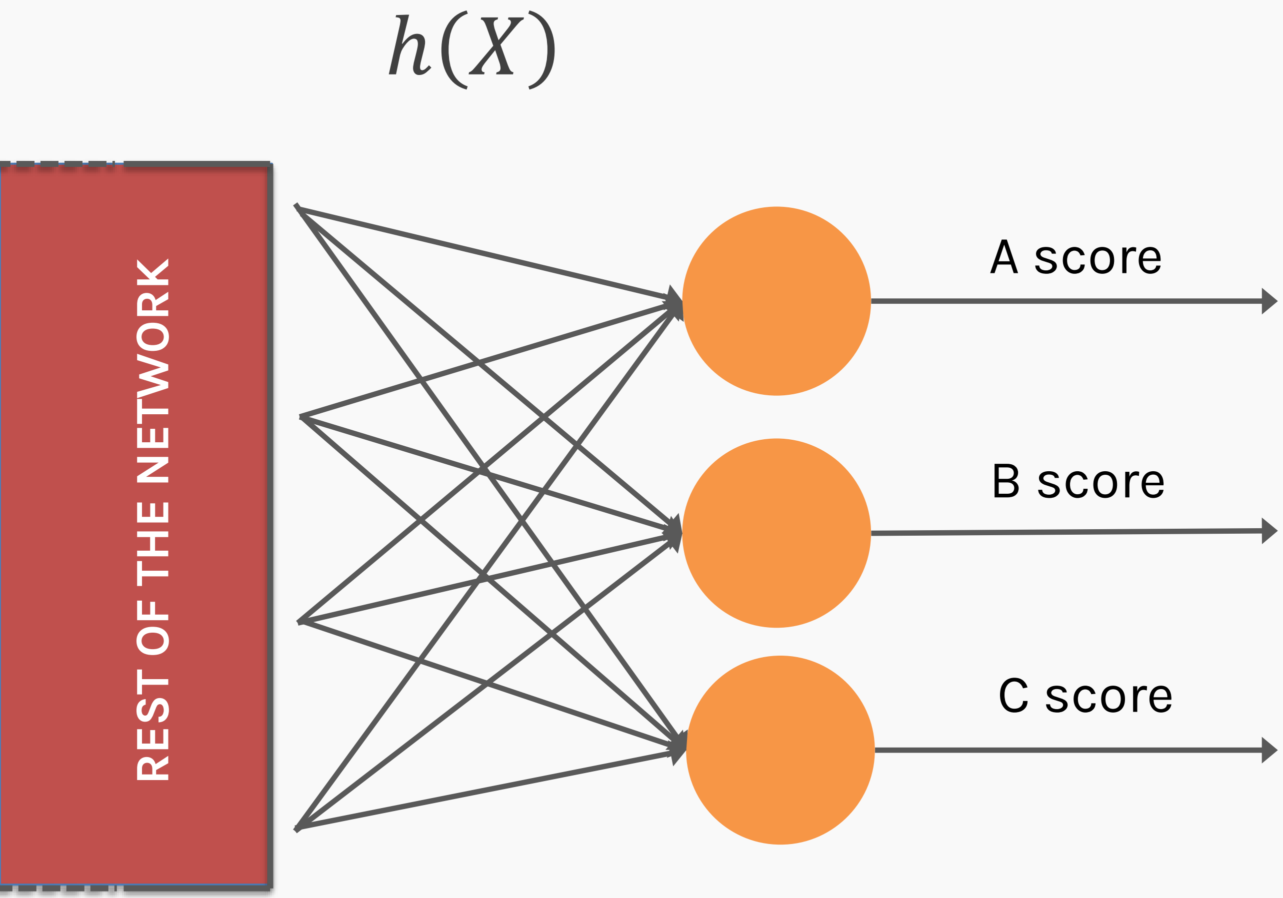
Output Units

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

Output Units

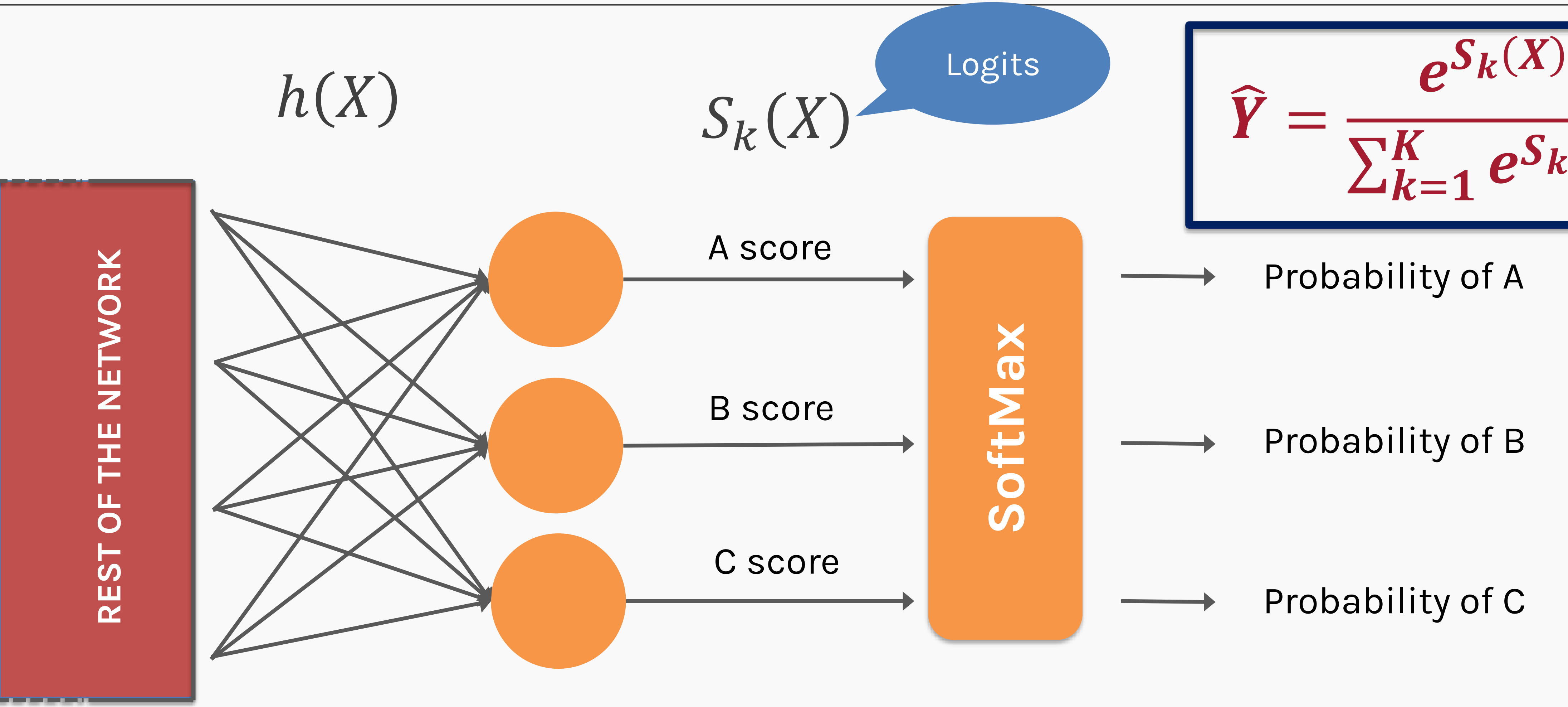
Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinouli	?	Cross Entropy

Output Units – Multiclass Classification (ex: 3 classes)

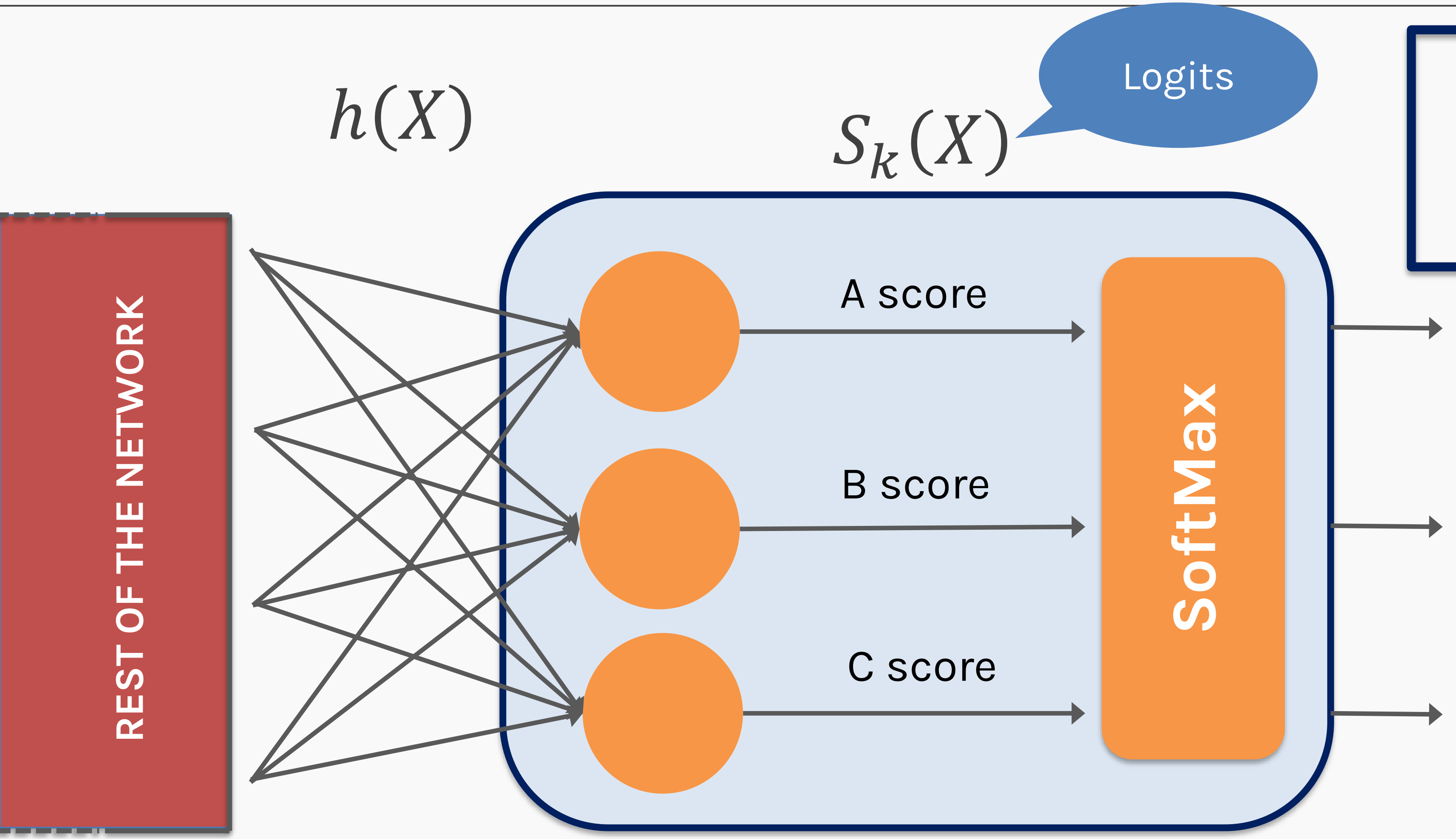


Output Units – Multiclass Classification (ex: 3 classes)

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



Output Units – Multiclass Classification (ex: 3 classes)



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

NOTE: In the case of multiclass classification, the number of output units will always be the same as the number of classes in the response variable.

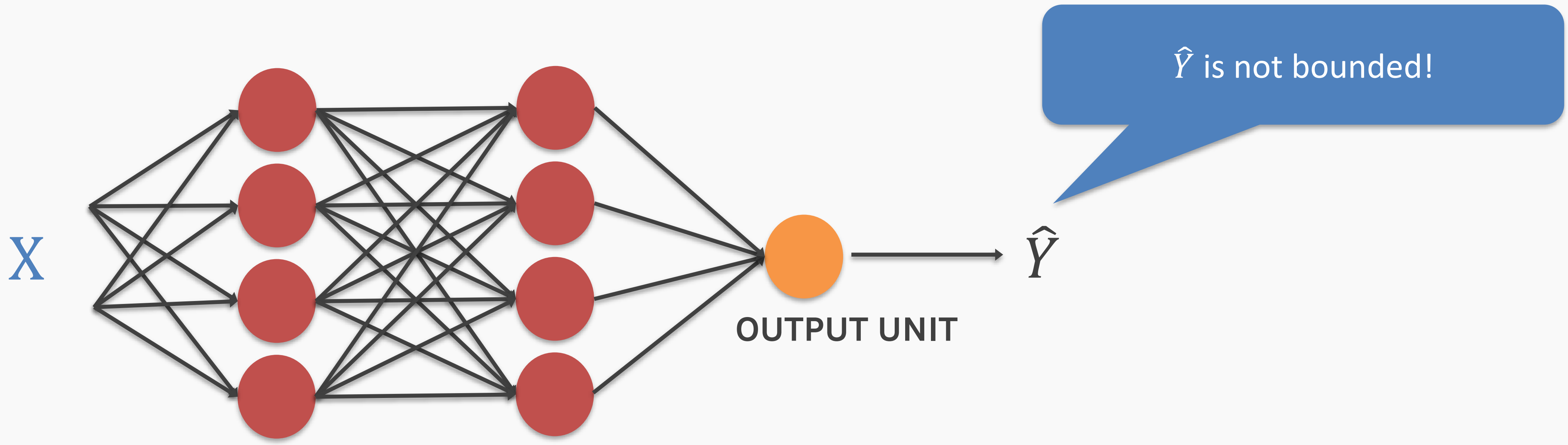
Output Units

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

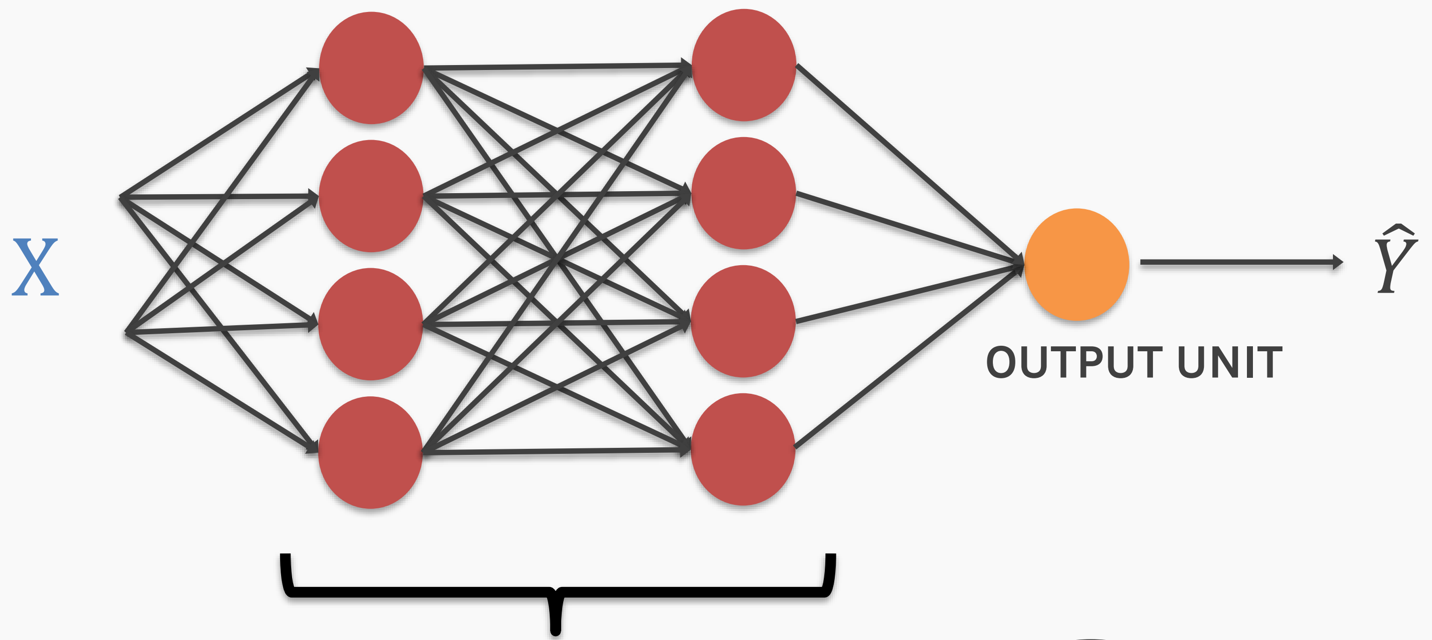
Output Units

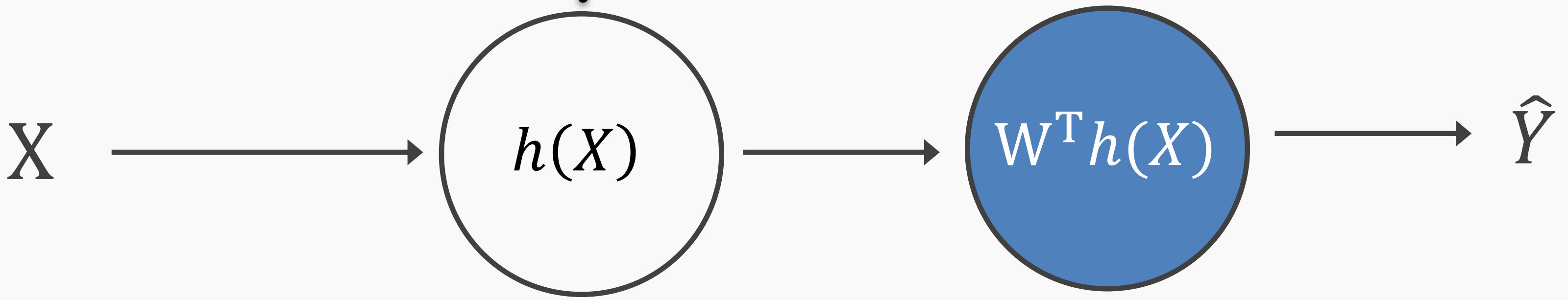
Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	?	MSE

Output Units - Regression



Output Units - Regression





$$X \Longrightarrow h(X) \Longrightarrow \hat{Y} = W^T h(X)$$

Output Units

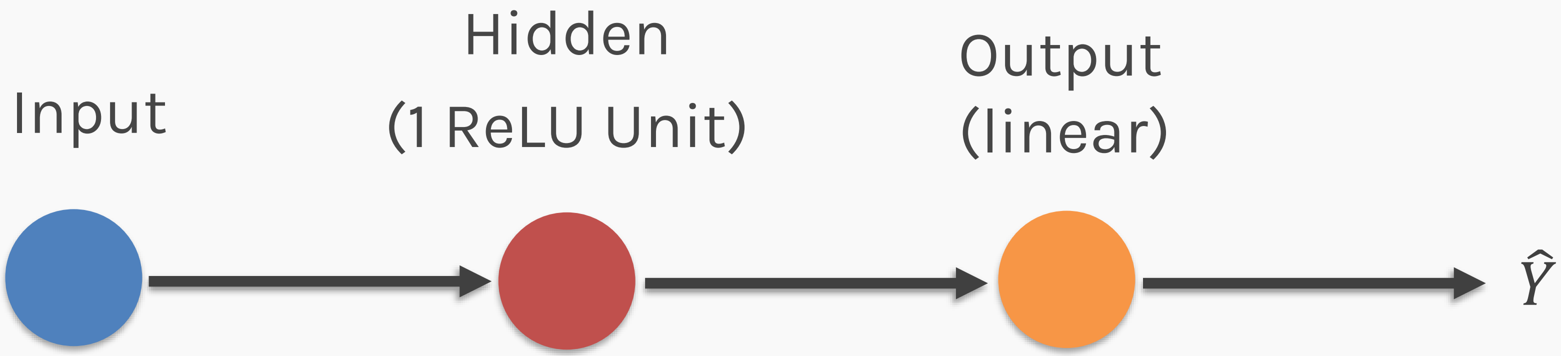
Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE

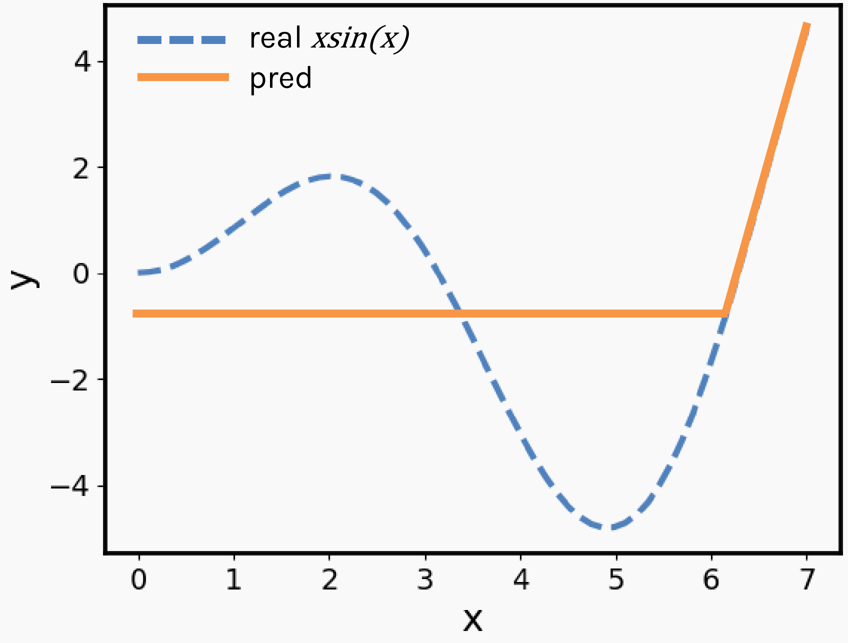
Design Choices

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

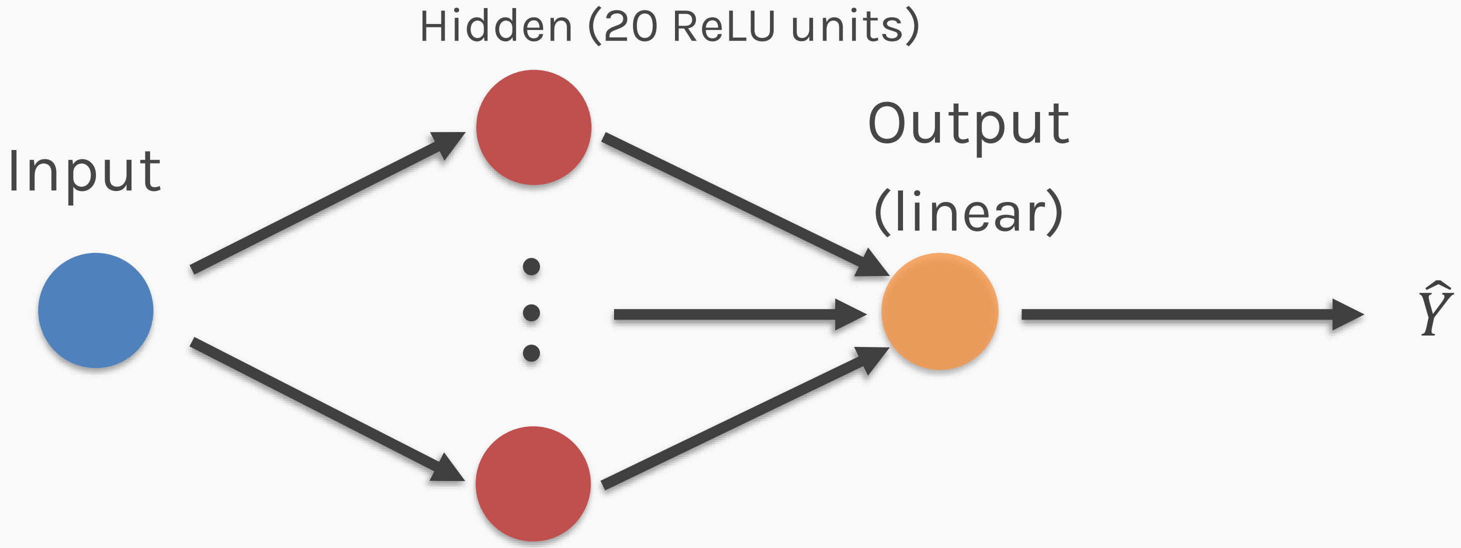
Architecture

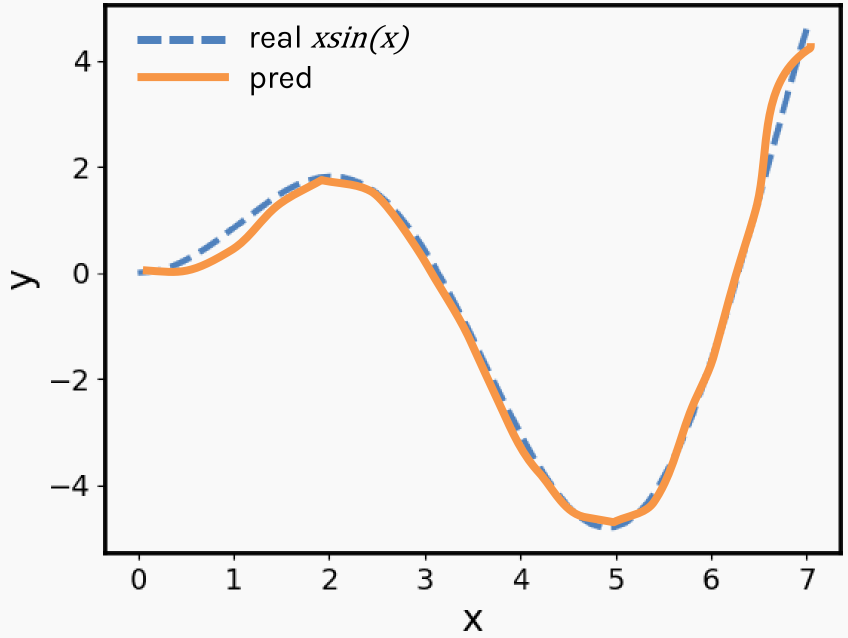
Number of nodes



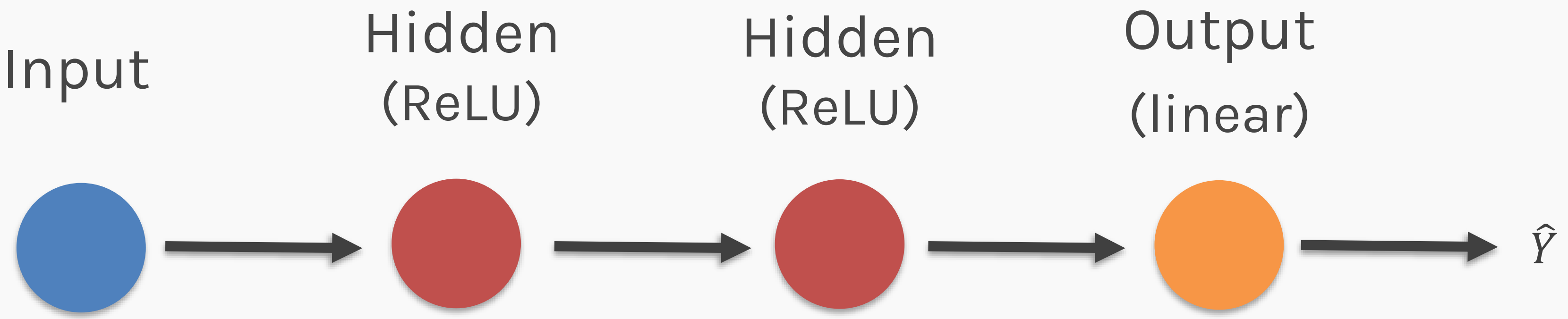


Number of nodes

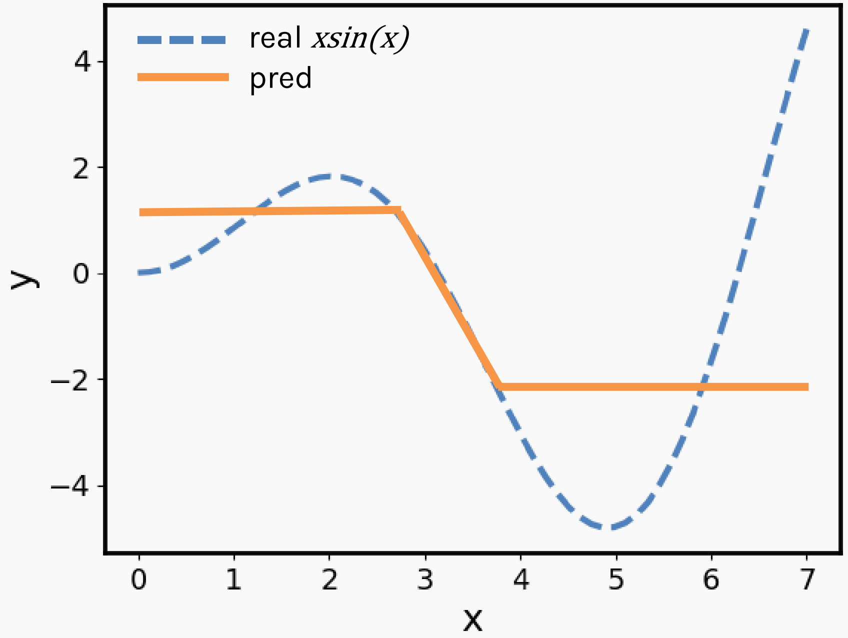




Layers

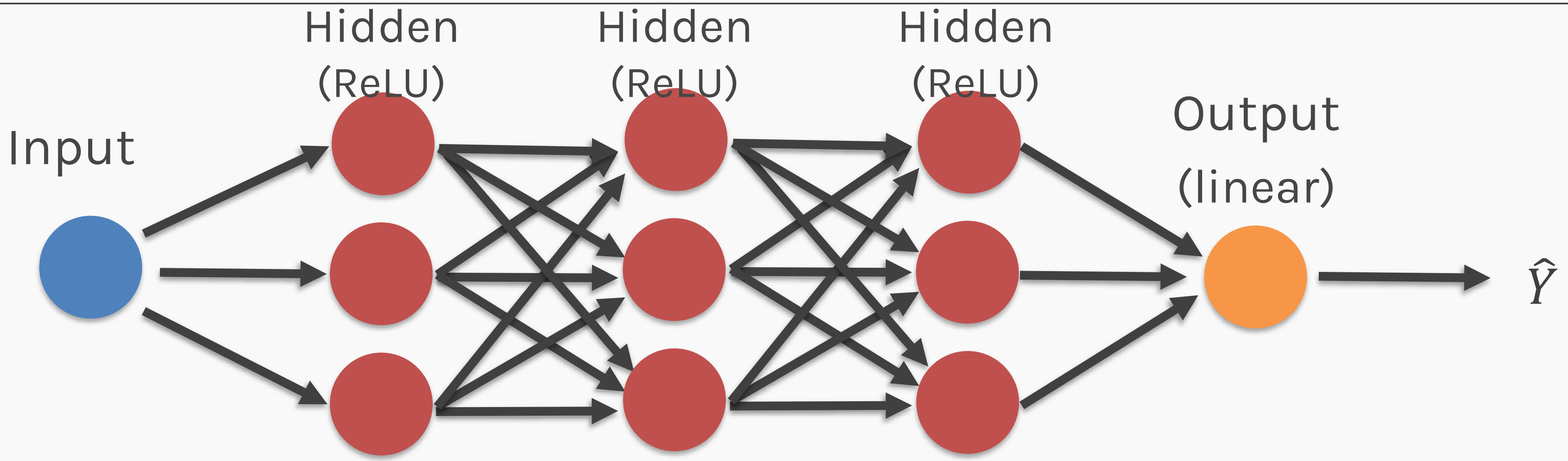


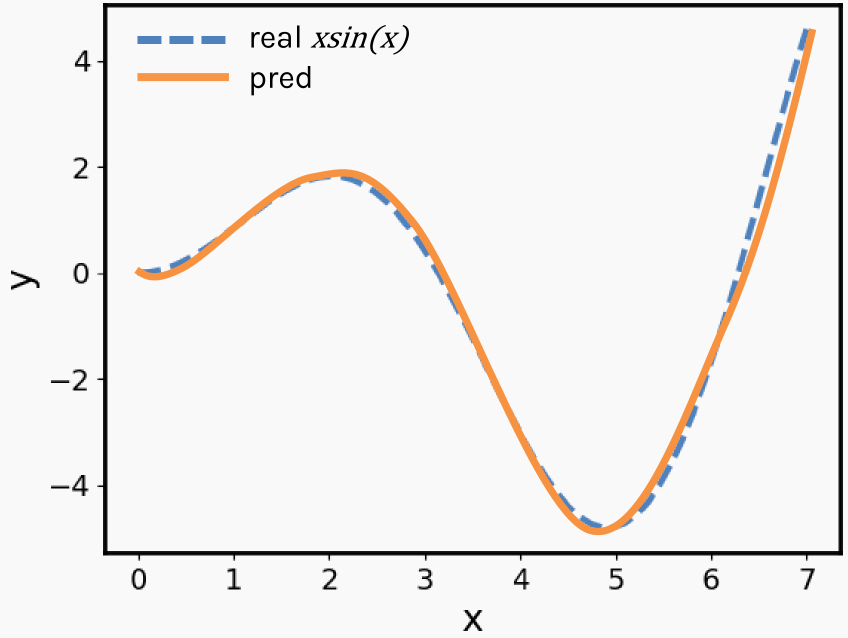
Input
Hidden
(ReLU)
Hidden
(ReLU)
Output
(linear)
$\hat{Y}$



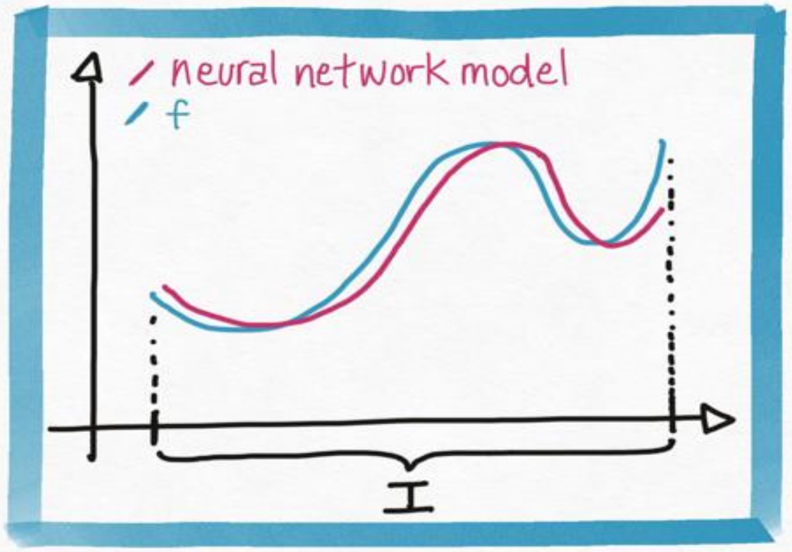
real $xsin(x)$
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

Layers


Input
Hidden
(ReLU)
Hidden
(ReLU)
Hidden
(ReLU)
Output
(linear)
$\hat{Y}$


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

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.

Layers

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

So, the question in your mind must be:

Quiz Time

Why do you think we need more layers?

A. To avoid overfitting

B. To enable the network to learn complex patterns through a hierarchical learning process step by step.

C. It is computationally faster

D. It works prof! I do not need to know why

Quiz Time

Why do you think we need more layers?

A. To avoid overfitting

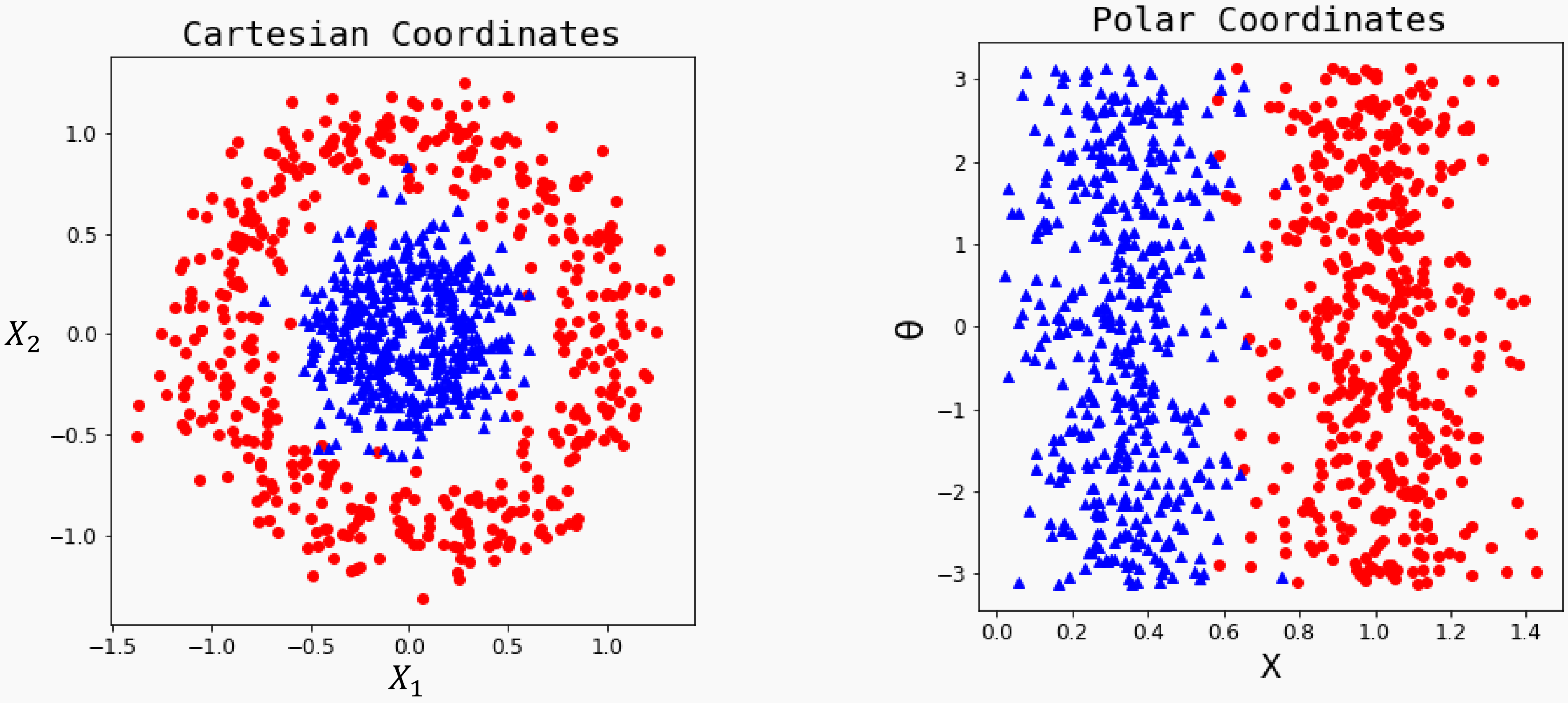
B. To enable the network to learn complex patterns through a hierarchical learning process step by step.

C. It is computationally faster

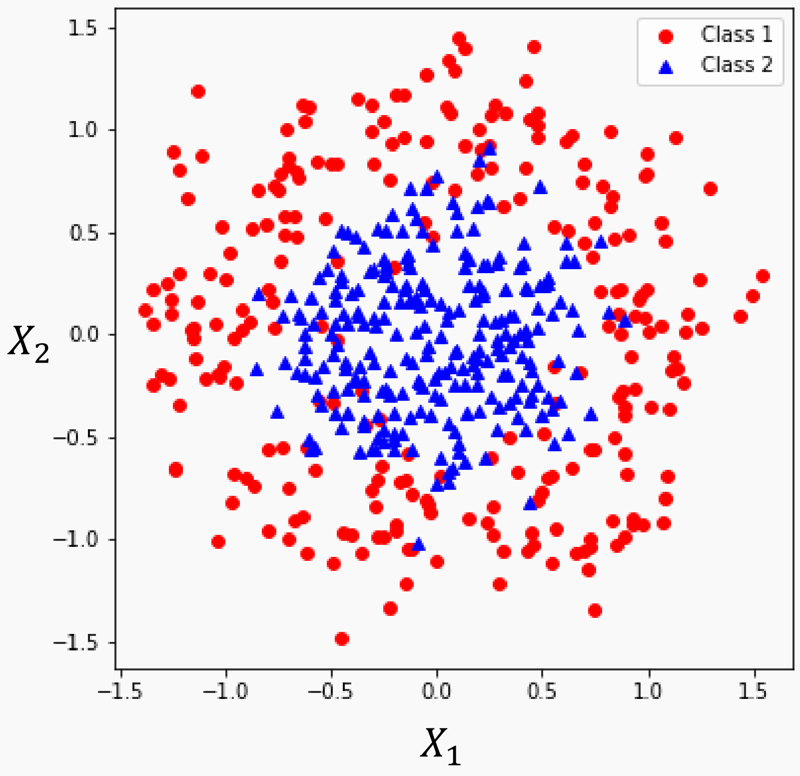
D. It works prof! I do not need to know why

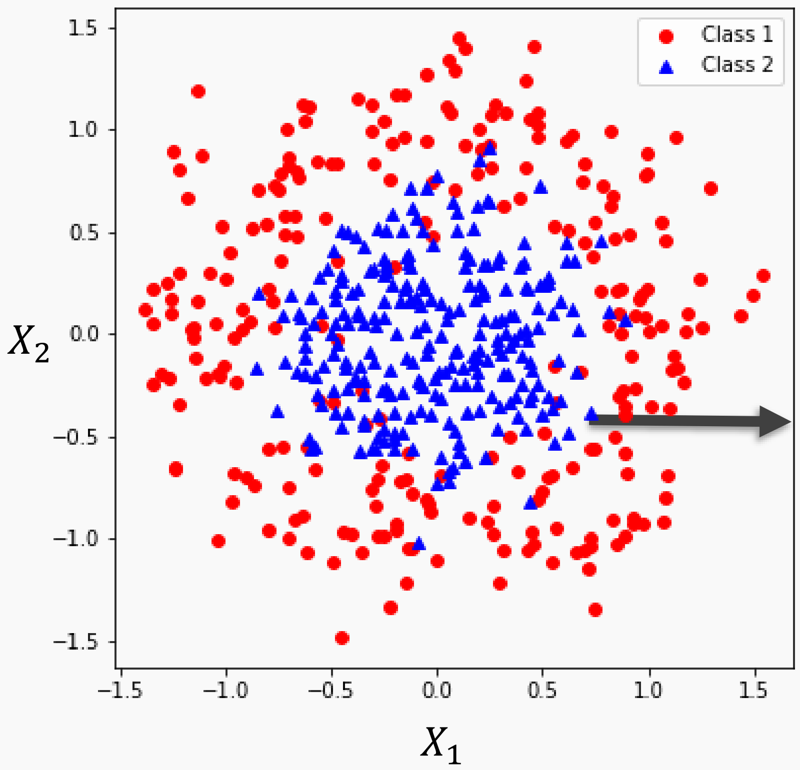
Why layers?

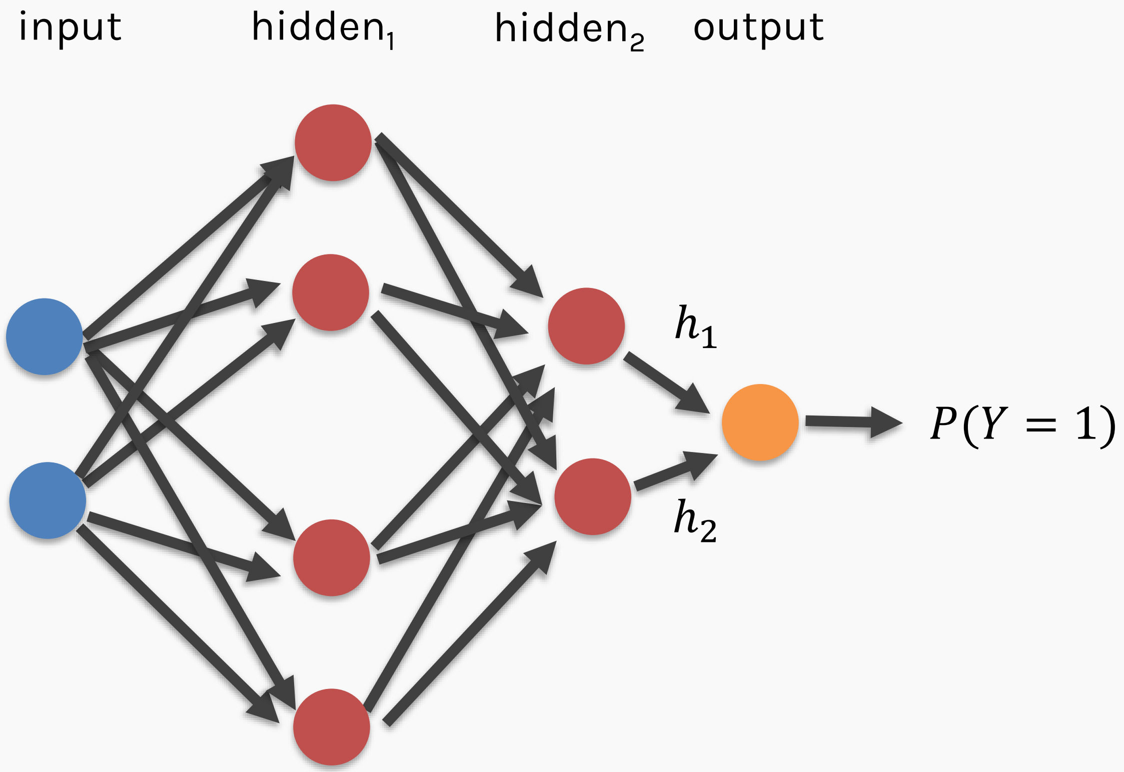
Representation matters!

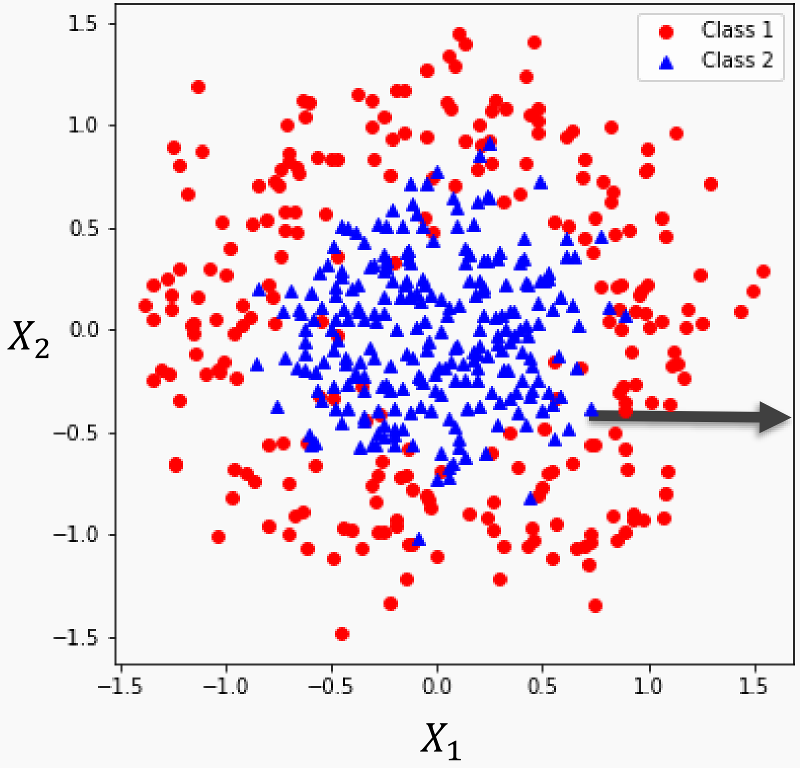


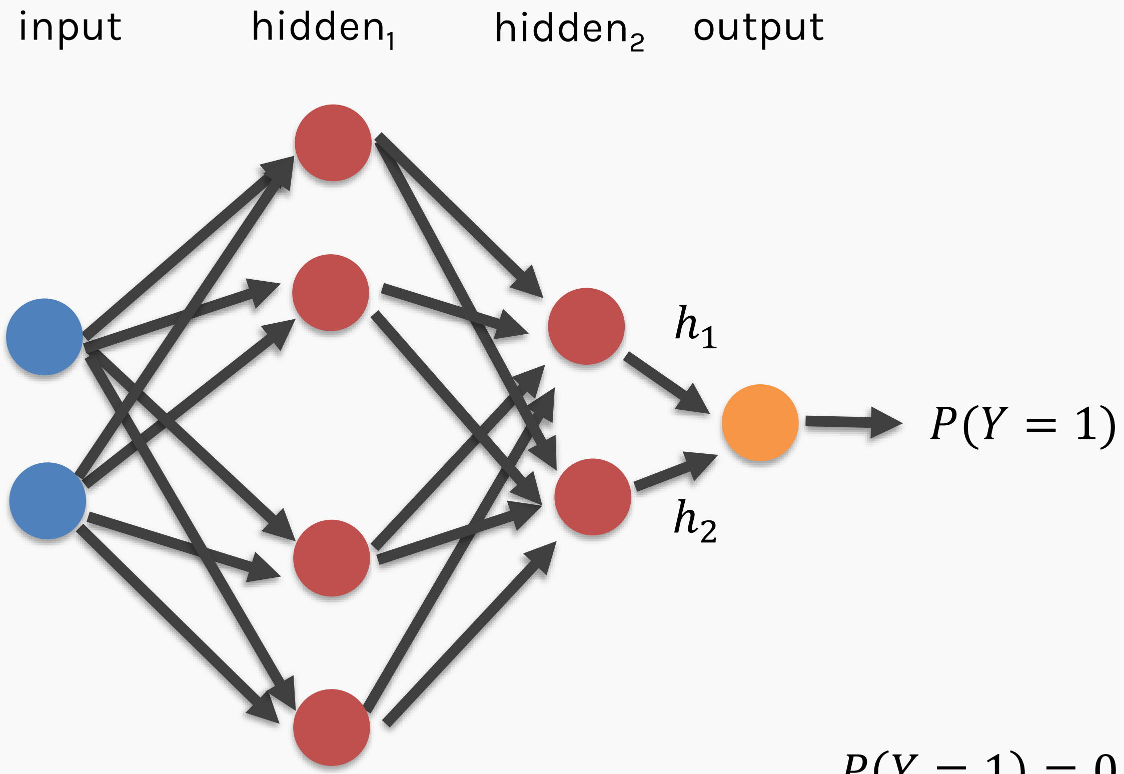
Neural networks can **learn useful representations** for the problem. This is another reason why they can be so powerful!

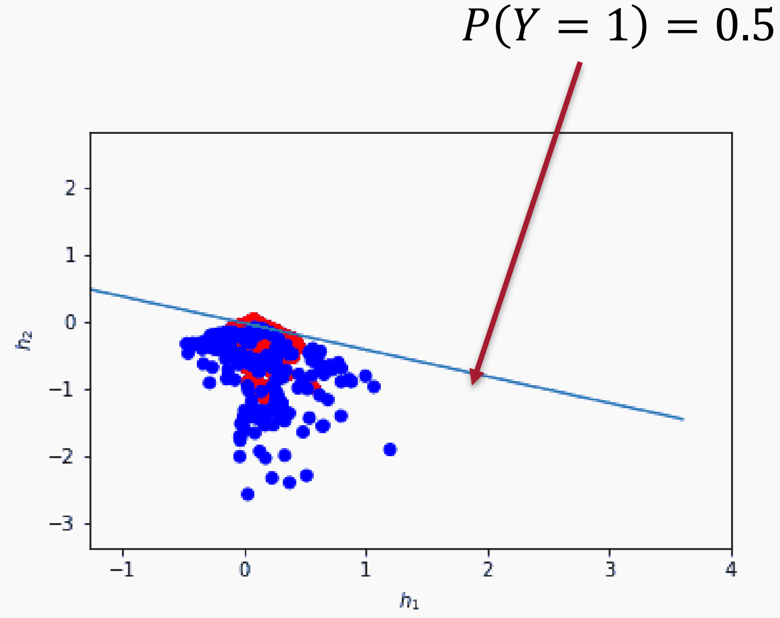

Class 1
Class 2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
X_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
X_1


Class 1
Class 2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
X_2
X_1

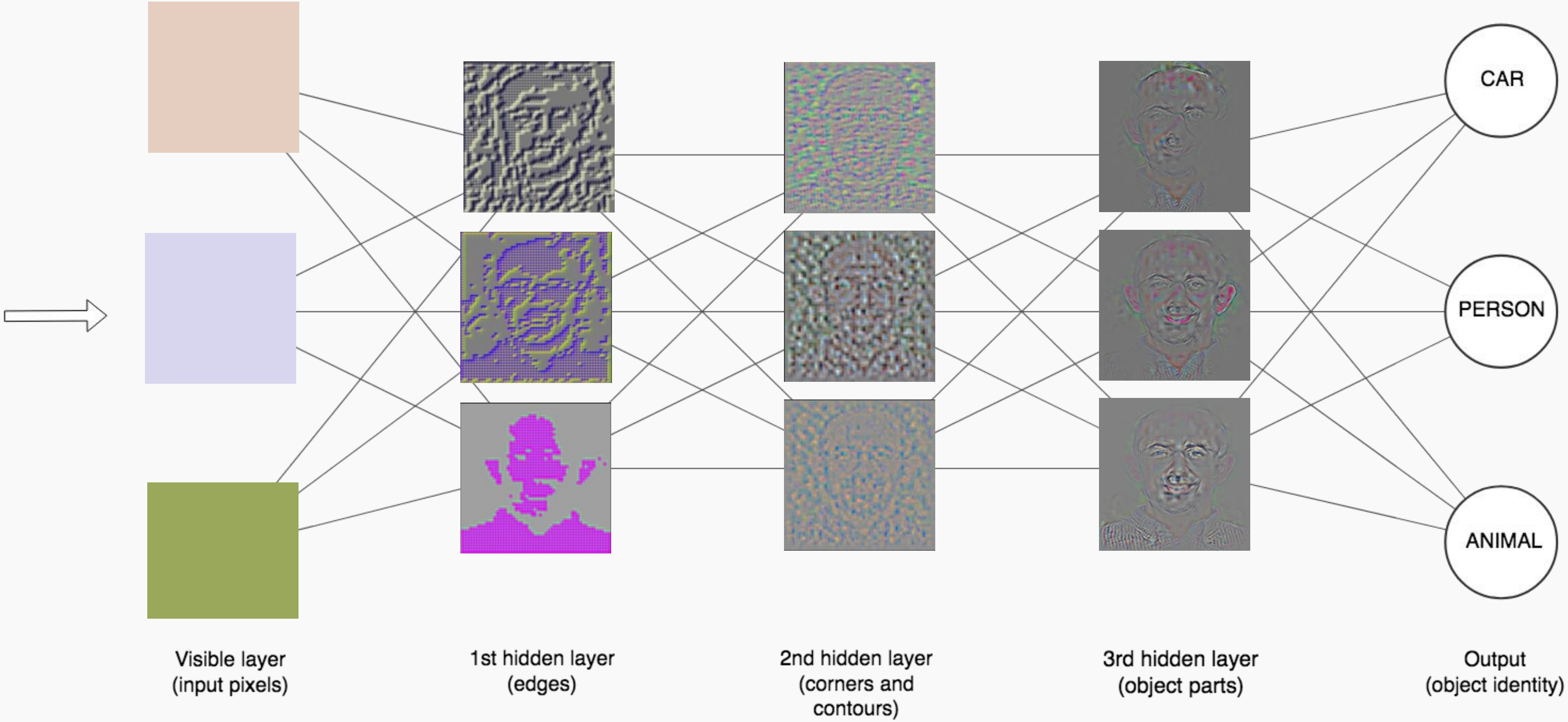

input
hidden$_1$
hidden$_2$
output
h_1
h_2
$P(Y = 1)$


Class 1
Class 2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
X_2
X_1


input
hidden_1
hidden_2
output
h_1
h_2
$P(Y = 1)$

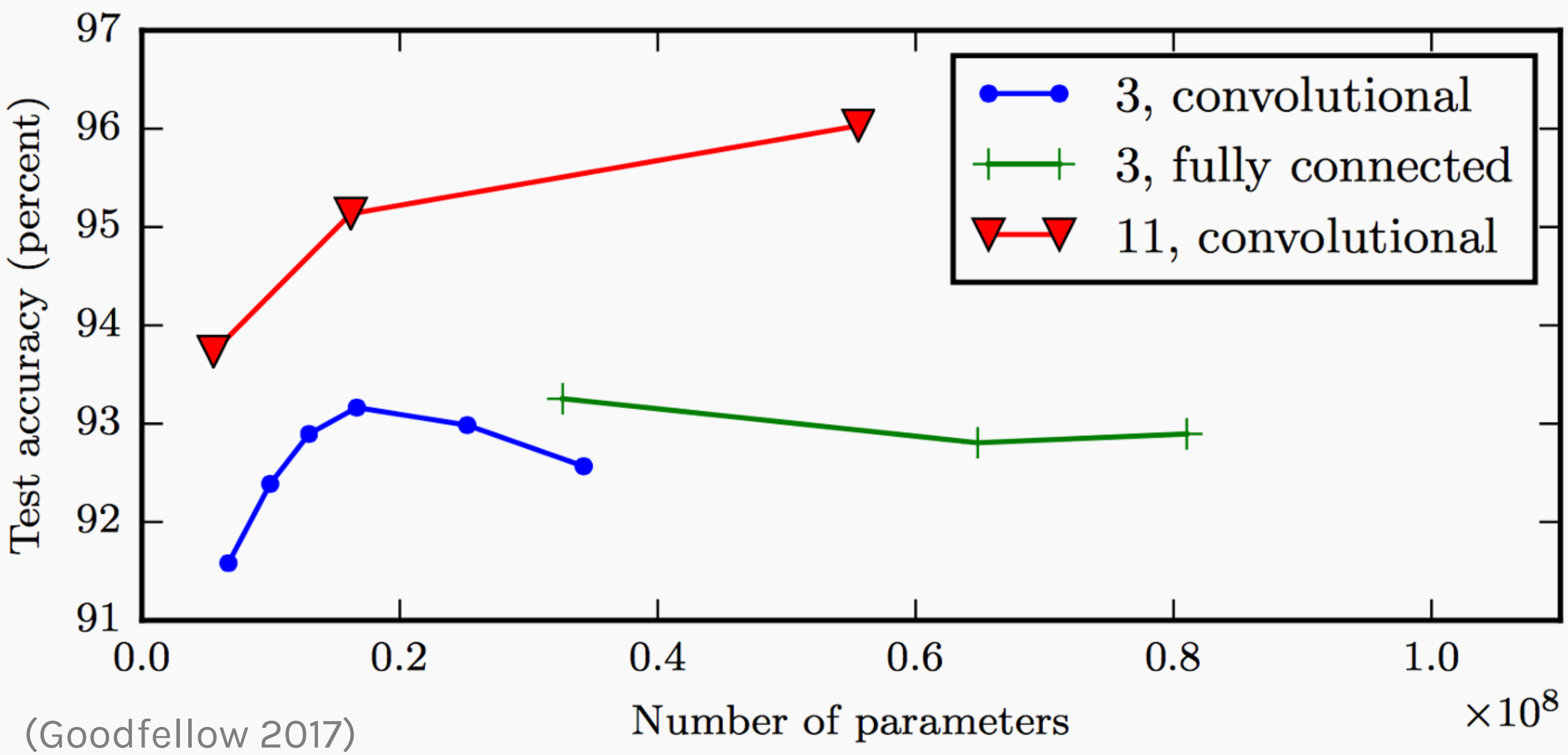

$P(Y = 1) = 0.5$
h_2
h_1

Depth Intuition



Shallow vs Deep Nets

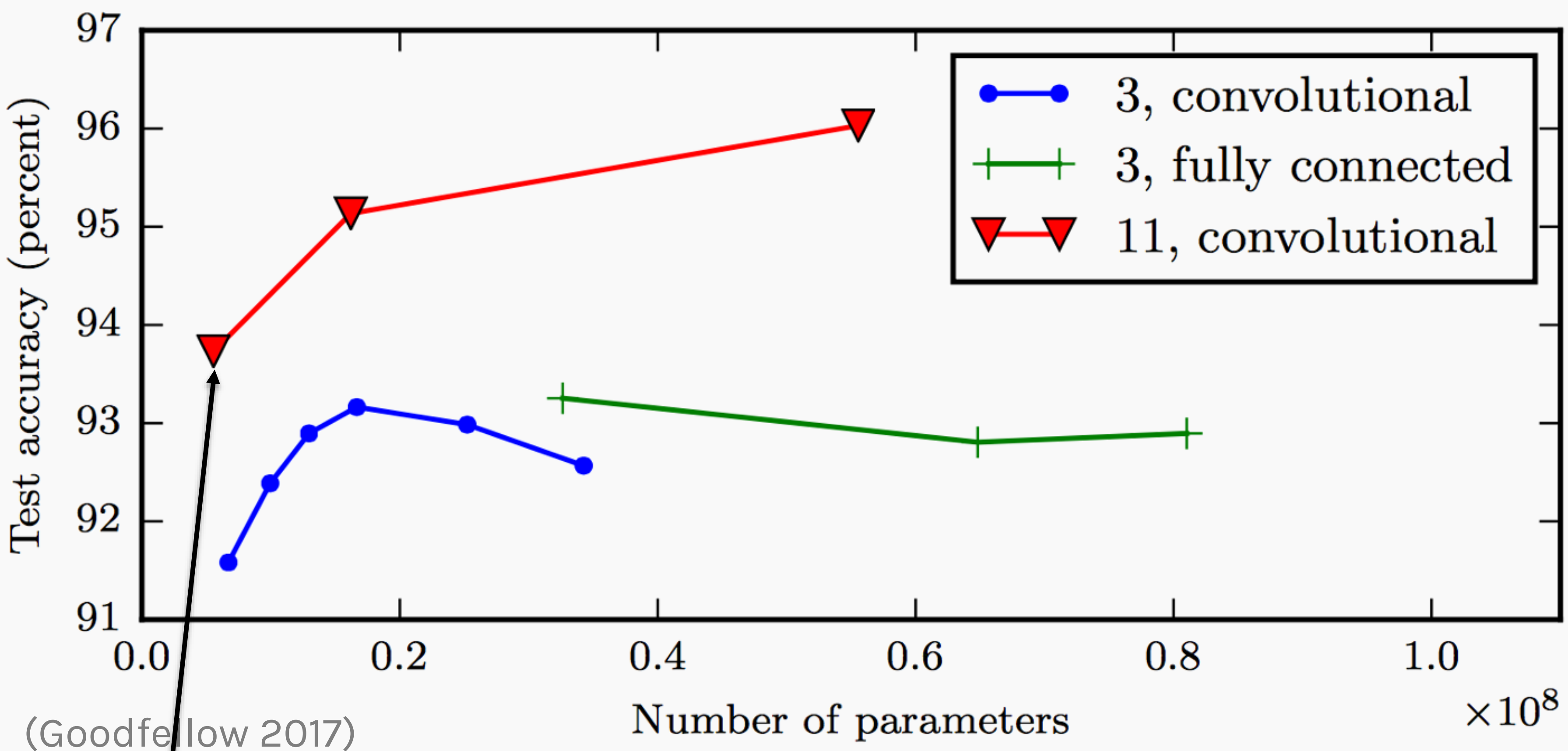
Depth helps, and it's not just because of more parameters



(Goodfellow 2017)

Shallow vs Deep Nets

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

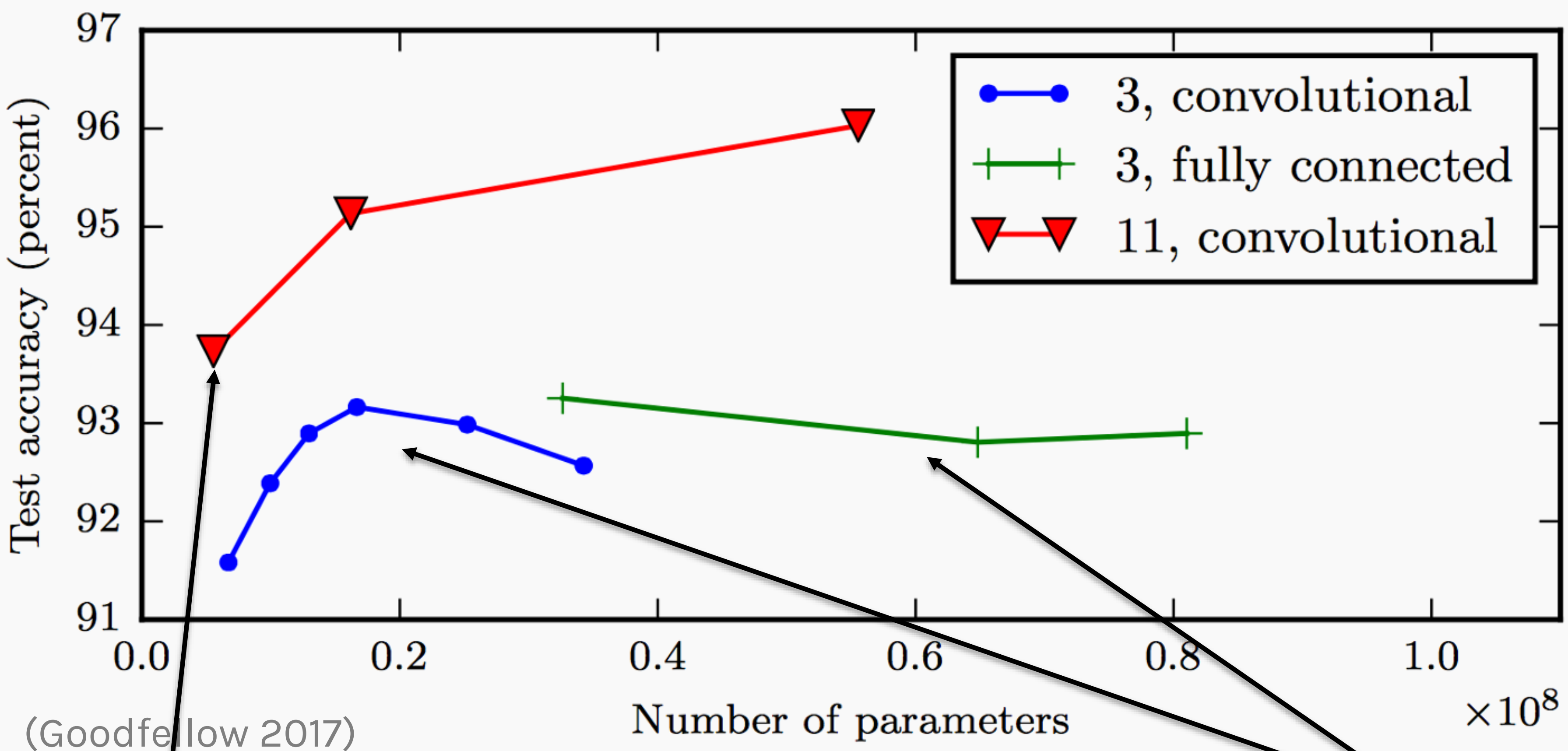


(Goodfellow 2017)

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

Shallow vs Deep Nets

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



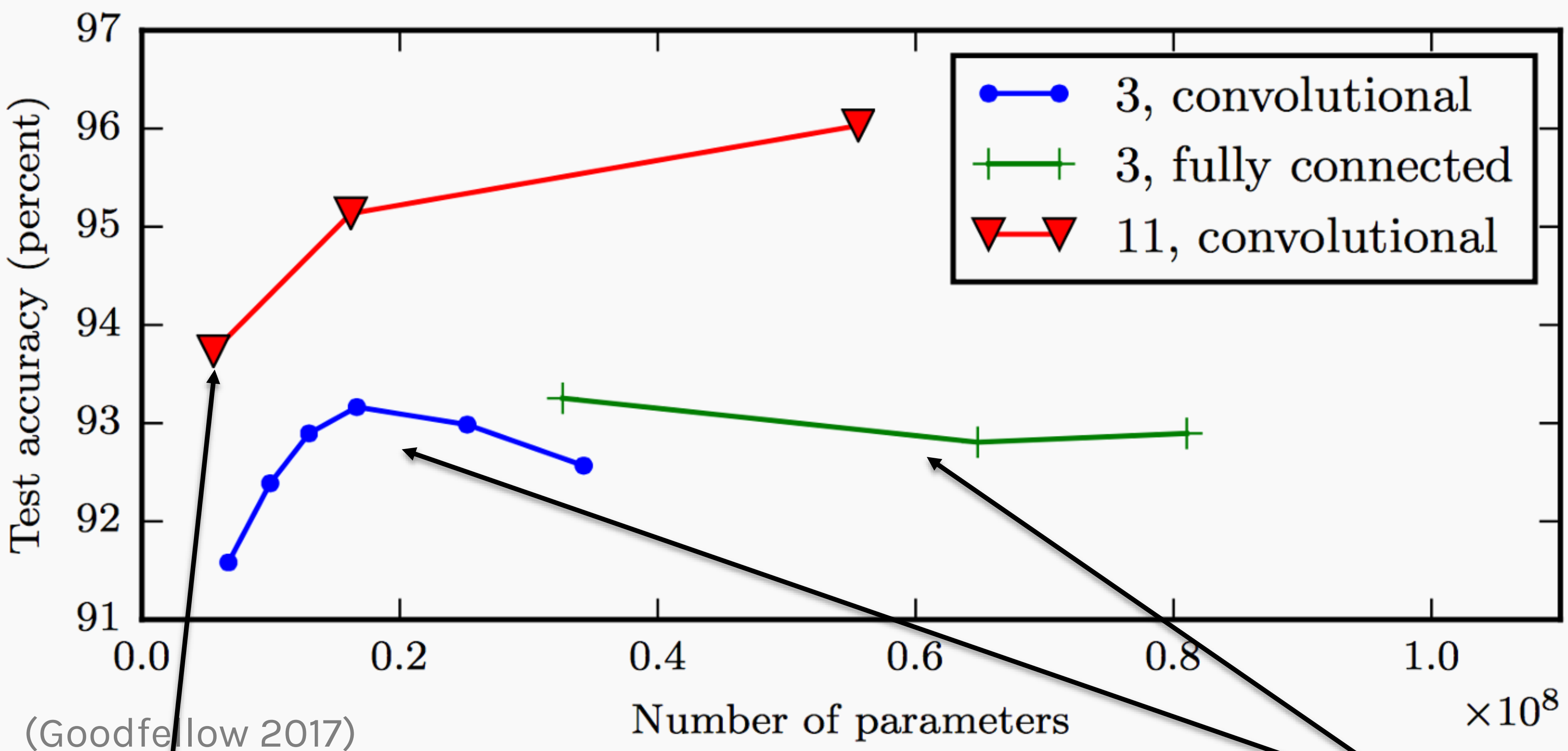
(Goodfellow 2017)

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 vs Deep Nets

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.

Thank you!