CoE202
Fundamentals of Artificial intelligence
<Big Data Analysis and Machine Learning>

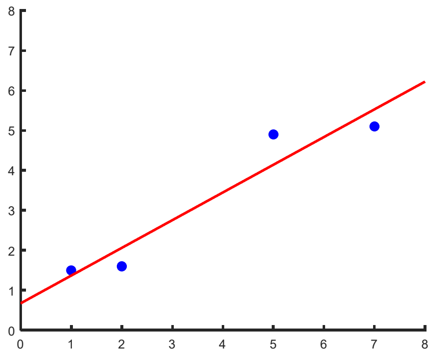
Neural Network

Prof. Young-Gyu Yoon
School of EE, KAIST

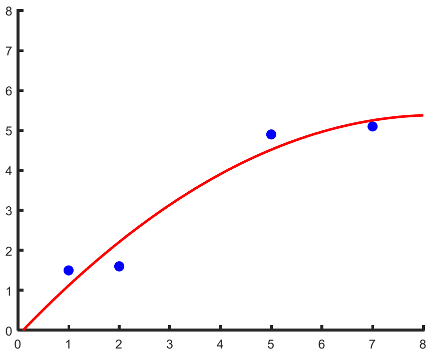
Contents

- Recap
  - Model selection problem
  - Bias-variance tradeoff
  - Generalization & Validation
  - Splitting dataset
- Biological neuron
- Artificial neuron
- Artificial neural network

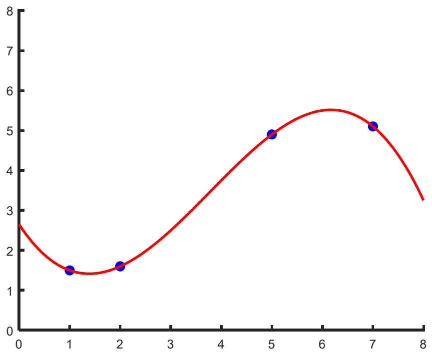
Recap: Model selection problem



$f(x) = 0.6934x + 0.6747$



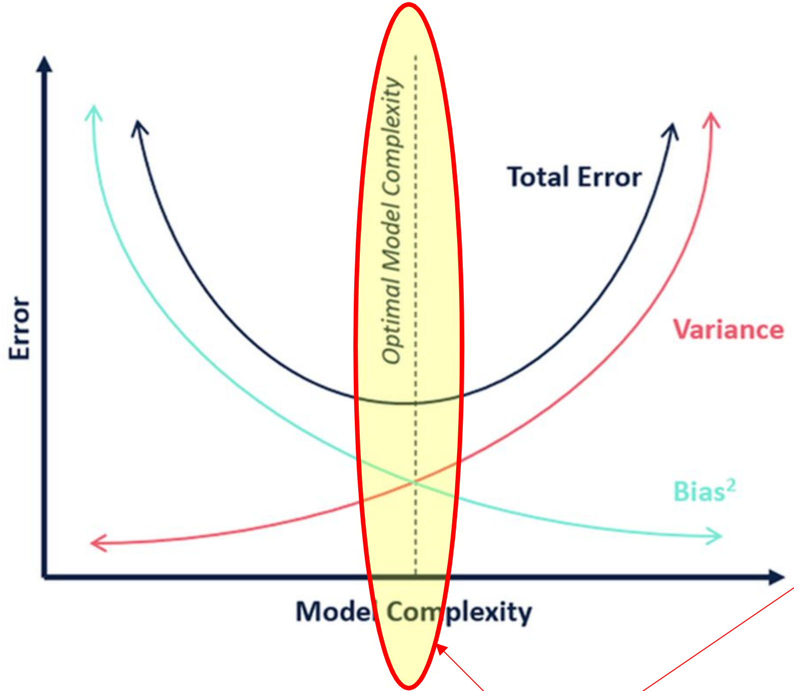
$f(x) = -0.0805x^2 + 1.3331x - 0.1339$

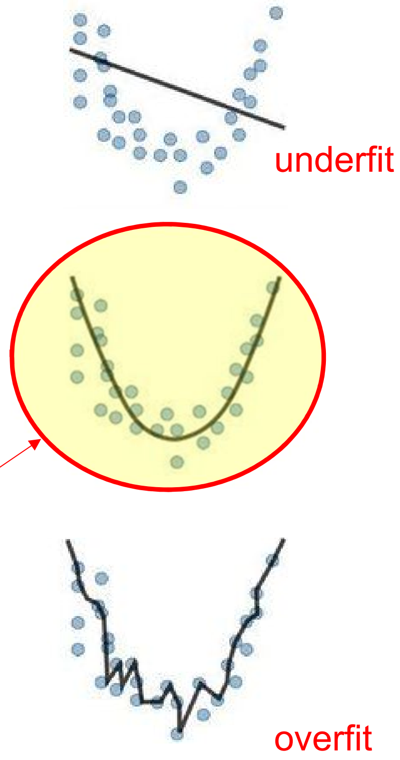


$f(x) = -0.075x^3 + 0.85x^2 - 1.925x + 2.65$

- For a same given data set…
- Higher order polynomial has more degree of freedom
  - 2nd order polynomical can be considered a just special case of 3rd order polynomial
- Higher order polynomial has lower loss value
  - Does it mean it is better?

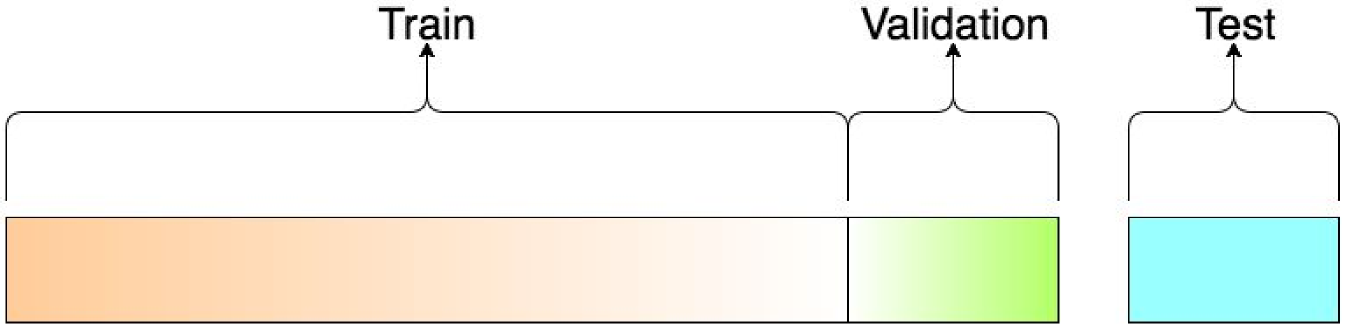
Recap: Bias-Variance Tradeoff





We want to be here

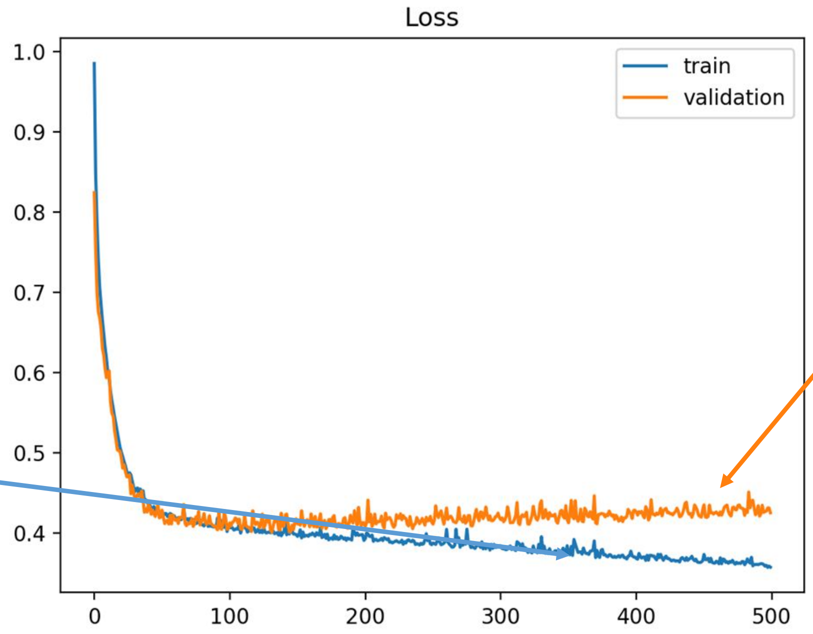
Recap: Training, Validation, Test



- **Depending on the availability and characteristics of the data set, the data can be split differently**
  - **80-10-10**
  - **50-25-25**
  - **Other ratios …**

Recap: Training & Validation

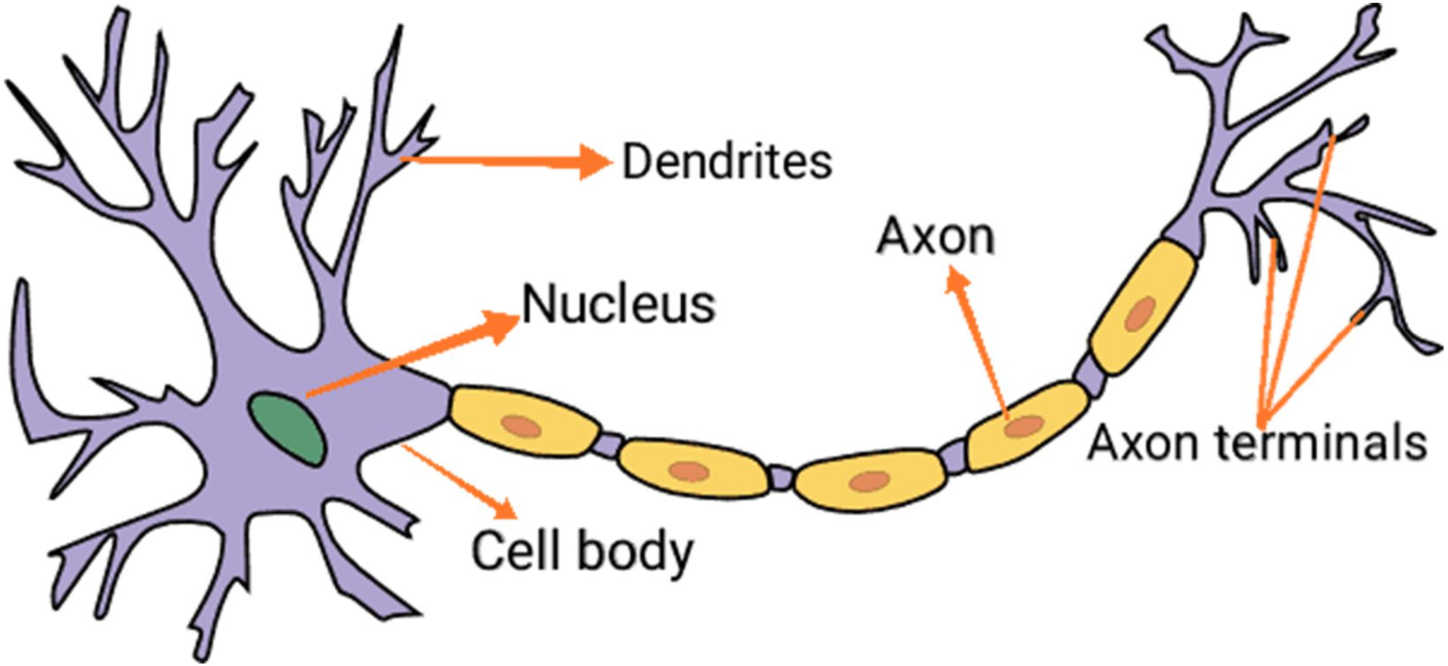
- Typical learning curve



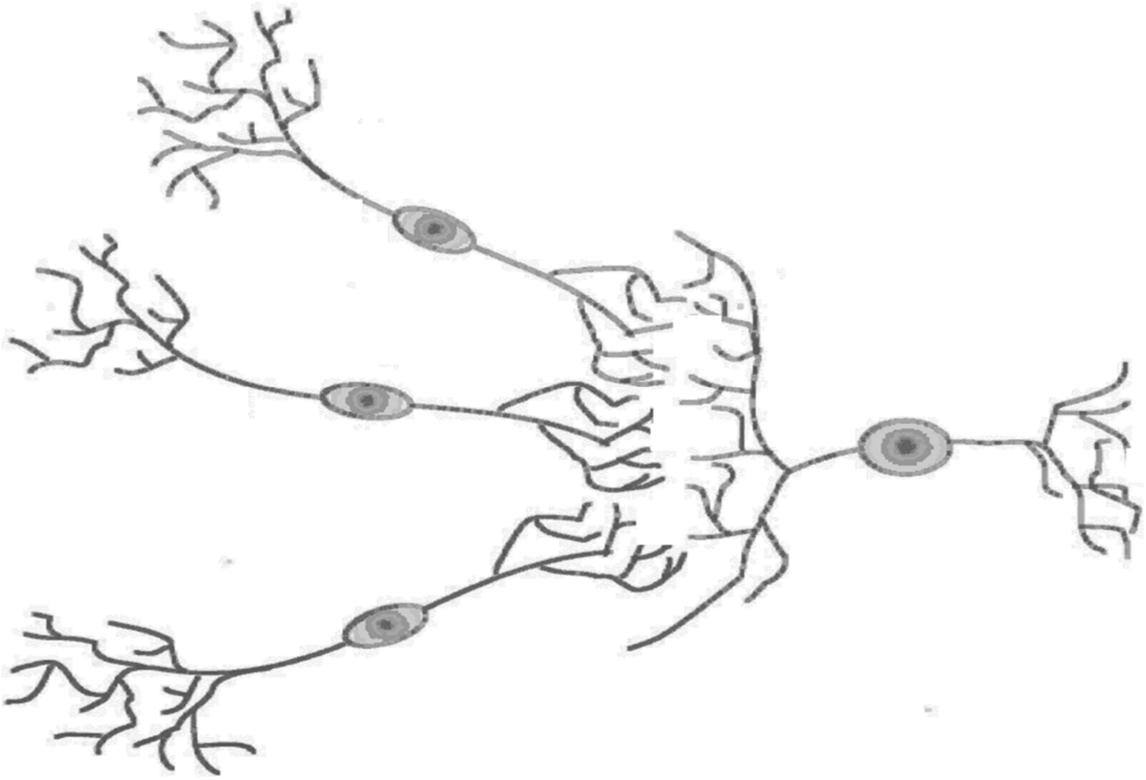
Our model is getting better and better over time for the training data set…

But from validation, we can see that overfitting is occurring (performance for unseen data is not getting better)

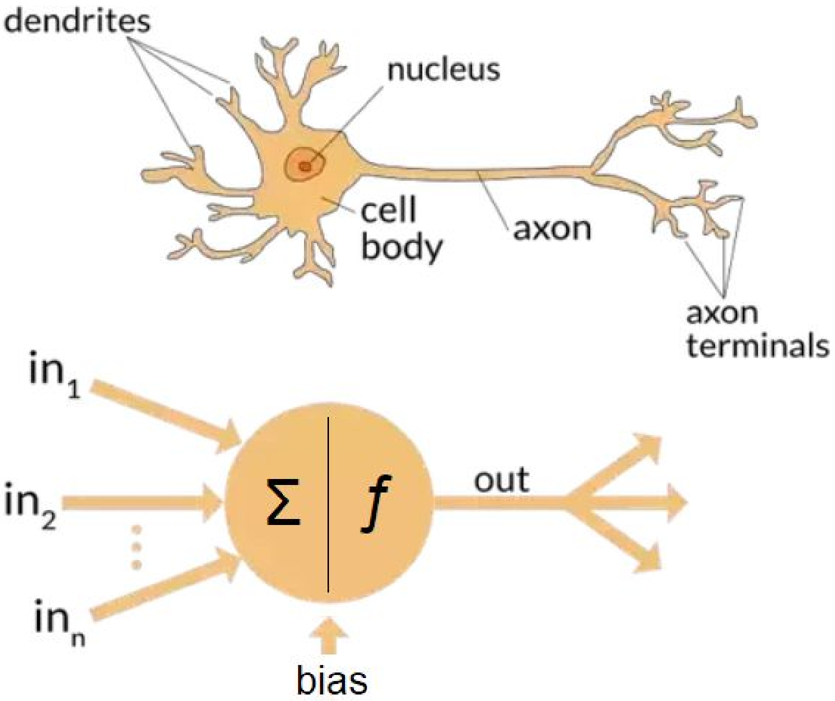
Biological neuron



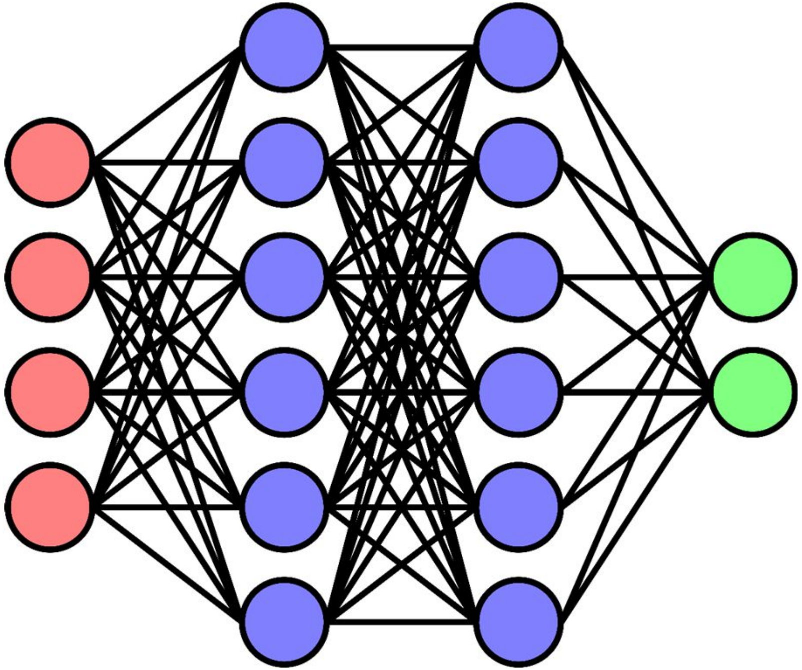
Biological neural network



Biological vs. Artificial Neuron

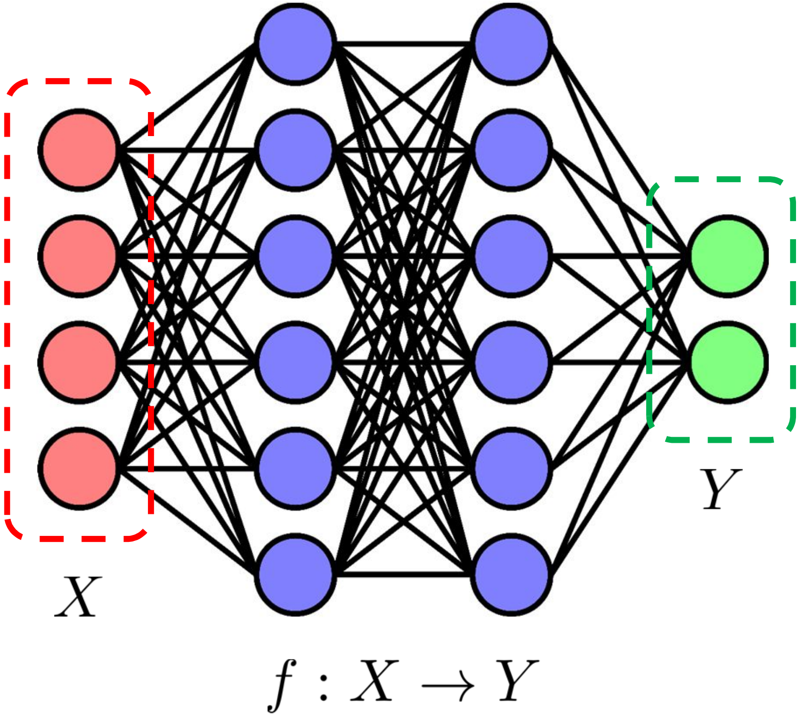


Artificial Neural Network



$$f : X \to Y$$

The "Neural Network"



For a data set

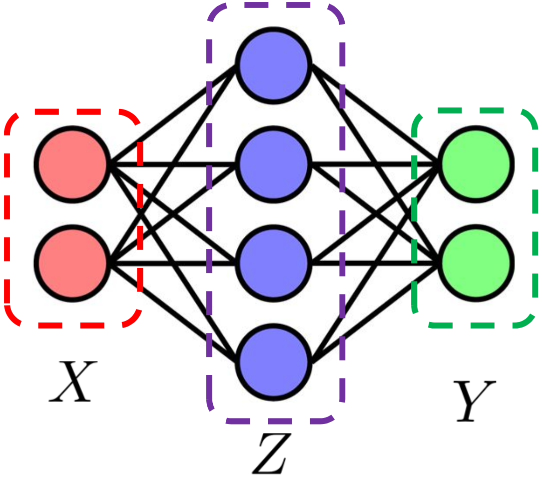
$\mathcal{D} = \{(\vec{x_1}, \vec{y_1}), (\vec{x_2}, \vec{y_2}), \cdots, (\vec{x_N}, \vec{y_N})\}$

Seeks a function $f : X \to Y$

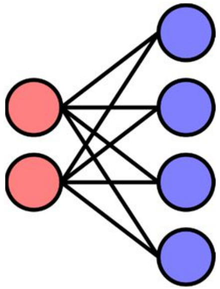
Such that a loss function

$\mathcal{L} : X \times Y \to \mathcal{R}$ is minimized

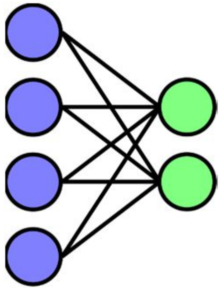
Neural Network as a composite function



$$Y = f(X) = f_2(Z) = f_2(f_1(X))$$



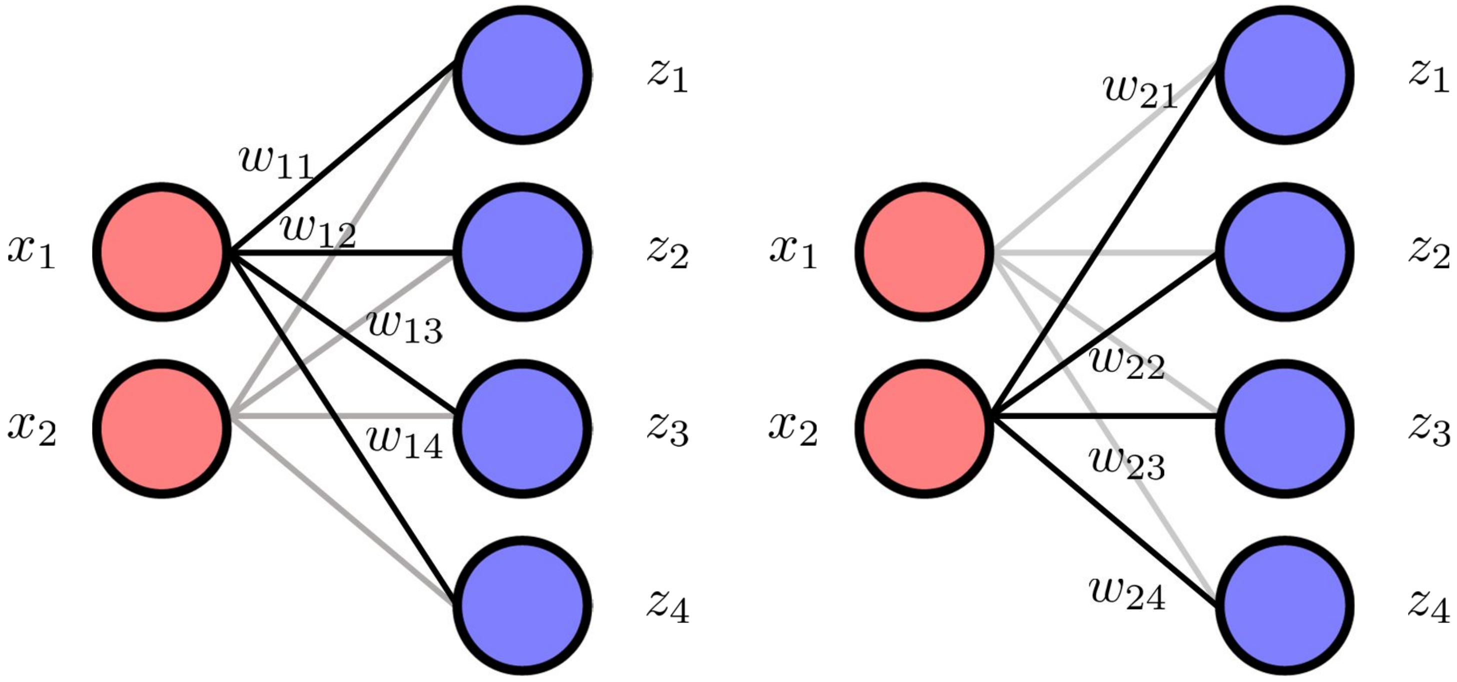
$$Z = f_1(X)$$



$$Y = f_2(Z)$$

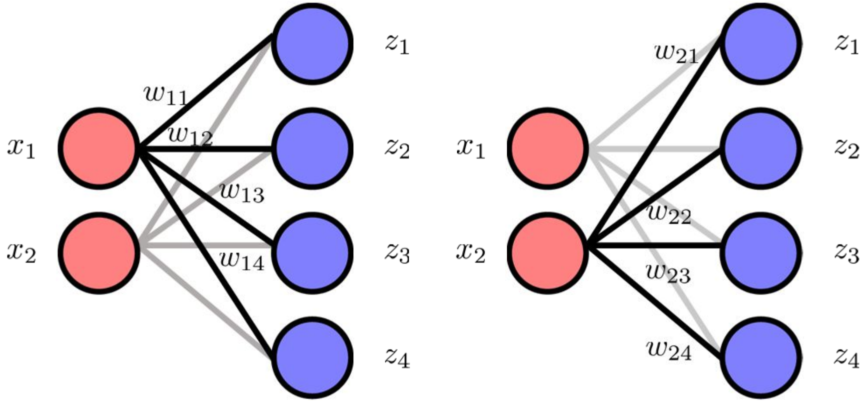
multiple layers of neurons
= composite function

Single layer in a neural network



$$Z = f_1(X)$$

Single layer in a neural network



$$z_1 = \mathbf{h}(w_{11}x_1 + w_{21}x_2 + b_1)$$

$$z_2 = \mathbf{h}(w_{12}x_1 + w_{22}x_2 + b_2)$$

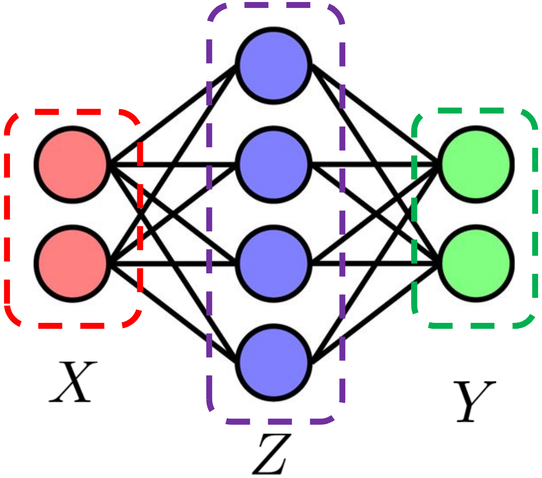
$$z_3 = \mathbf{h}(w_{13}x_1 + w_{23}x_2 + b_3)$$

$$z_4 = \mathbf{h}(w_{14}x_1 + w_{24}x_2 + b_4)$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \mathbf{h}(\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}) = \mathbf{h}(\begin{bmatrix} w_{11} & w_{21} & b_1 \\ w_{12} & w_{22} & b_2 \\ w_{13} & w_{23} & b_3 \\ w_{14} & w_{24} & b_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix})$$

$$Z = \mathbf{h}(W_{f1}X)$$

What is h?

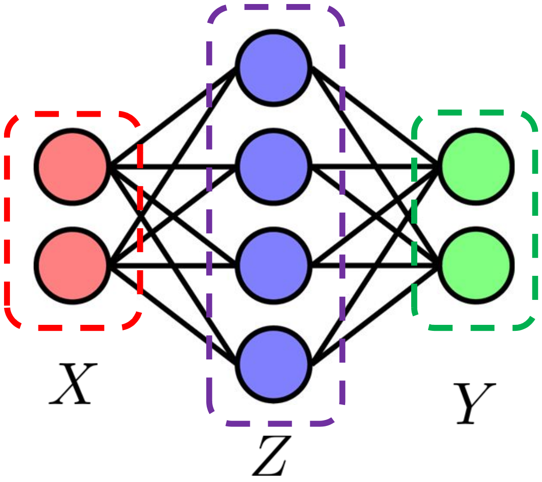


$$Y = f(X) = f_2(Z) = f_2(f_1(X))$$

$$Z = \mathbf{h}(W_{f1}X) = f_1(X)$$

$$Y = \mathbf{h}(W_{f2}Z) = f_2(Z)$$

Activation function



$$Y = f(X) = f_2(Z) = f_2(f_1(X))$$

$$Z = \mathbf{h}(W_{f1}X) = f_1(X)$$

$$Y = \mathbf{h}(W_{f2}Z) = f_2(Z)$$

- *h* is called the **activation function**
- If we choose *h* = I (identity function),

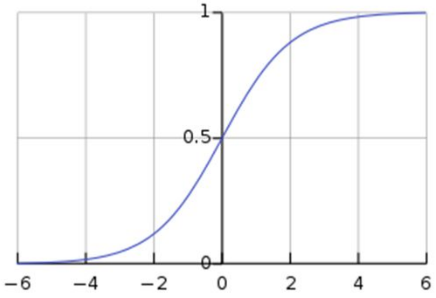
$$Y = W_{f2}Z = W_{f2}W_{f1}X = W_{f1,f2}X$$

the whole function h simply becomes a linear function

Activation function

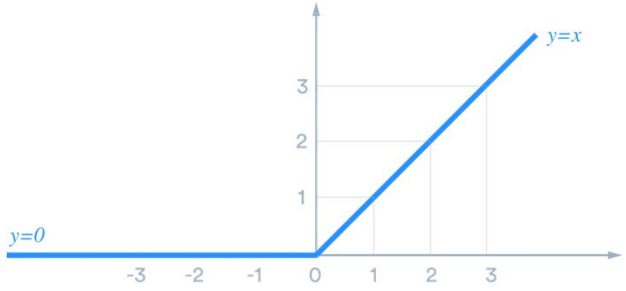
Sigmoid function

$$\boldsymbol{S}(z) = \frac{e^z}{e^z+1} = \frac{1}{1+e^{-z}}$$



ReLU (rectified linear unit) function

$$\boldsymbol{R}(z) = max(z, 0)$$



- These two functions are used often as the activation function
  - ReLU is the most popular choice these days
  - There are many other types of activation functions …

Why activation function?

$$Y = W_{f2}Z = W_{f2}W_{f1}X = W_{f1,f2}X$$

- Again, without the activation function, the neural network becomes just a linear function
- This nonlinear activation function allows us to model the nonlinear relationship between the input and the output

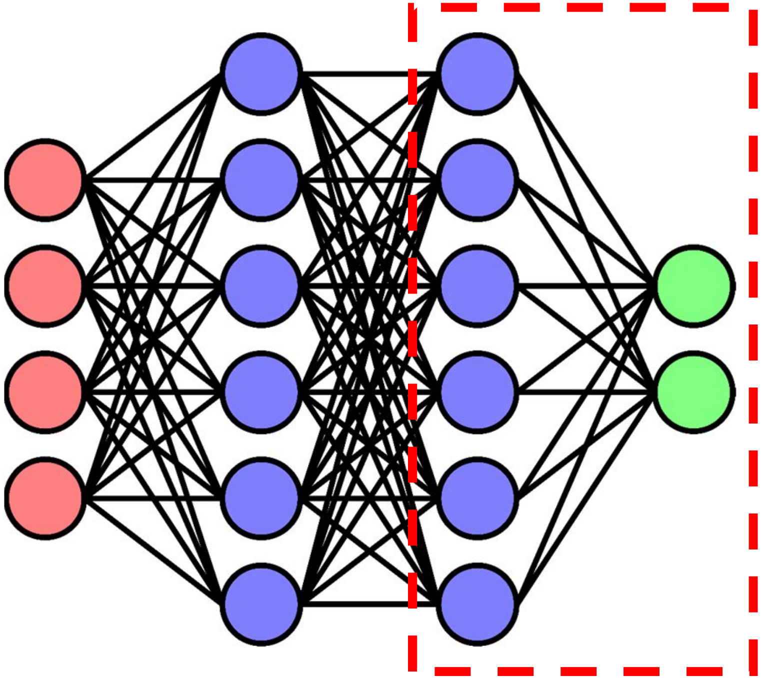
- Having multiple layers with these activation functions allow us to model even more complex relationship between the input and the output → “deep” learning

Why deep?

- Width vs. Depth
  - We can increase the “capacity” of a neural network either by
    - Increasing its depth (i.e., number of layers)
    - Increasing its width (i.e., number of neurons in each layer)
  - Both approaches can be used to make our network more capable of “approximating complex functions”

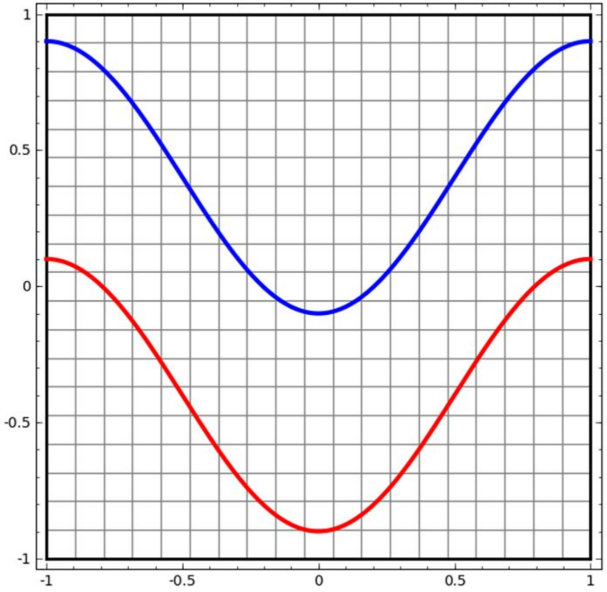
- “Deep network” comes with multiple chances to exploit non-linearity (more activation layers) to approximate nonlinear functions
  - Increasing depth is often more powerful way of increasing the capacity

Single layer in NN is linear

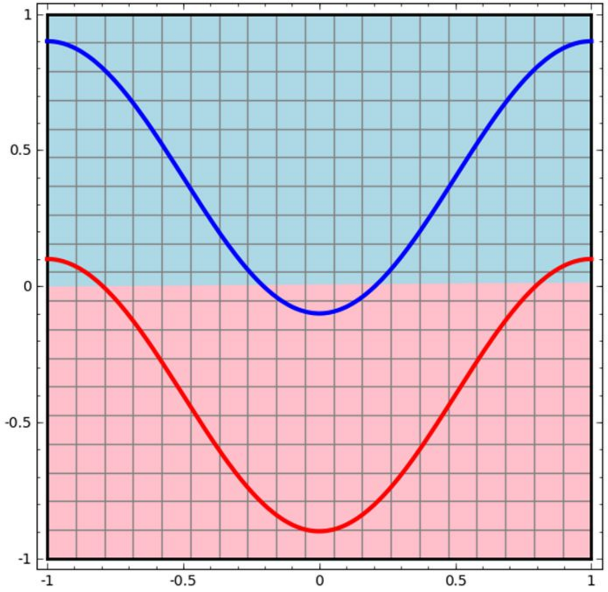


- Let's consider a classification example
- The last layer simple does "linear classification"
  - Input → nonlinearly transformed as an input to the last layer → last layer performs linear classification
- This means the "nonlinear transform" turns something not linearly separable into linearly separable thing
  - What does this mean?

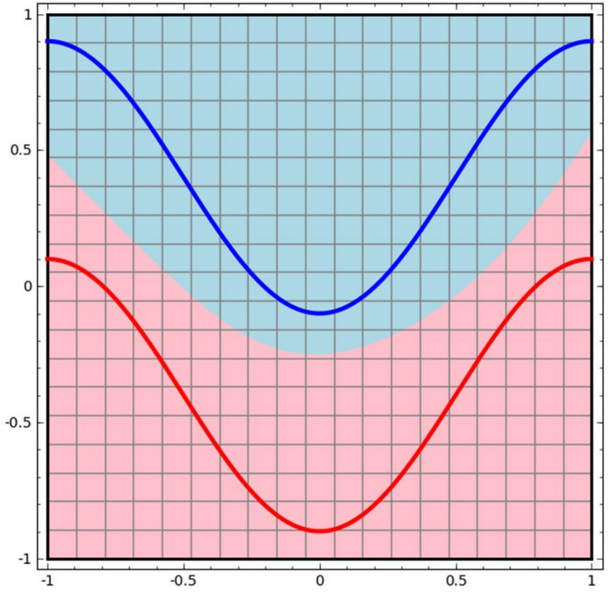
Power of nonlinearity



Data to be classified



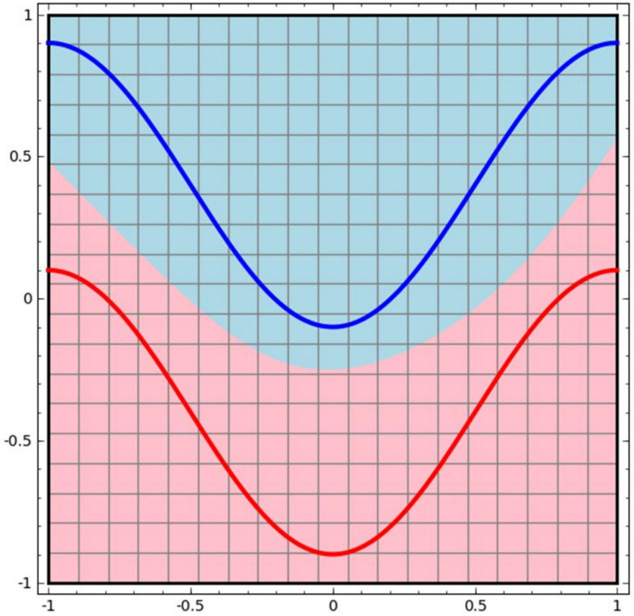
“best” linear classification

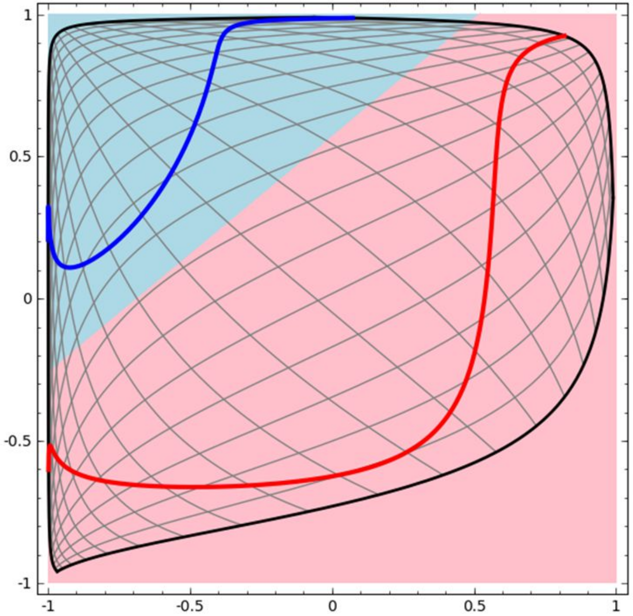


nonlinear classification

https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Power of nonlinearity

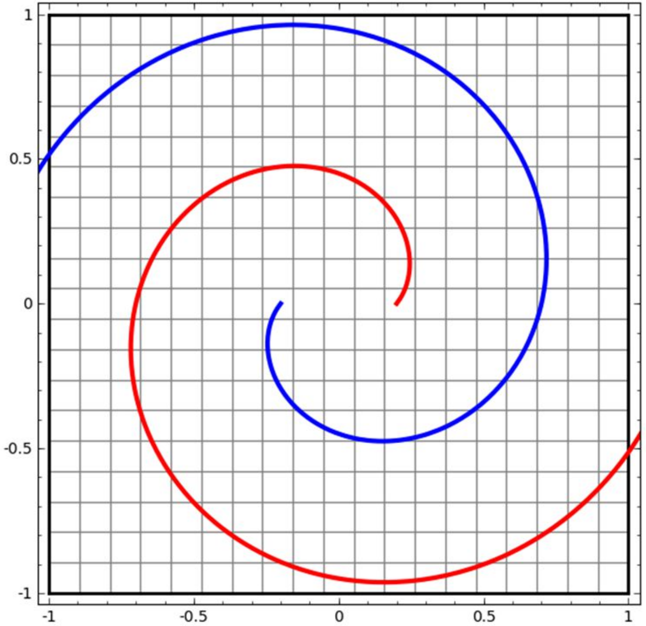




- Non-linear transform made the data ‘linearly separable’

https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

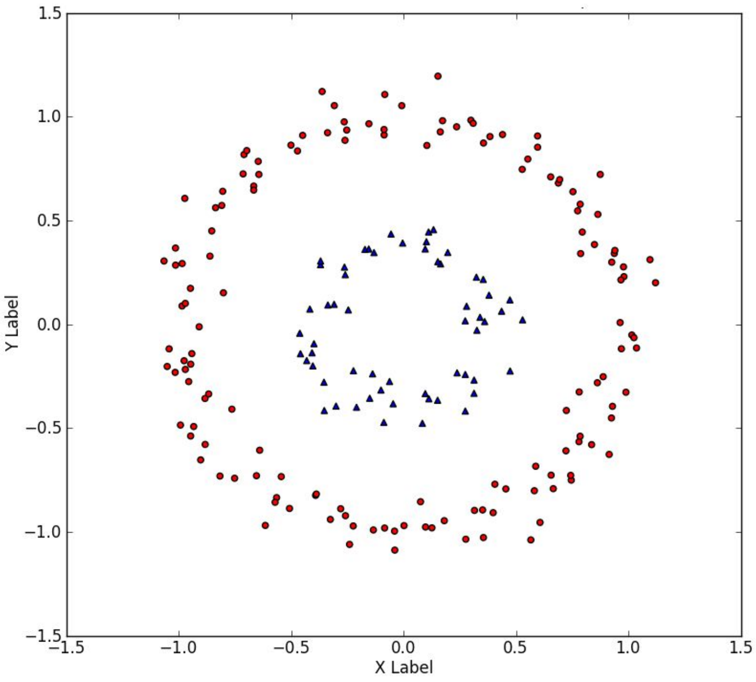
Power of nonlinearity



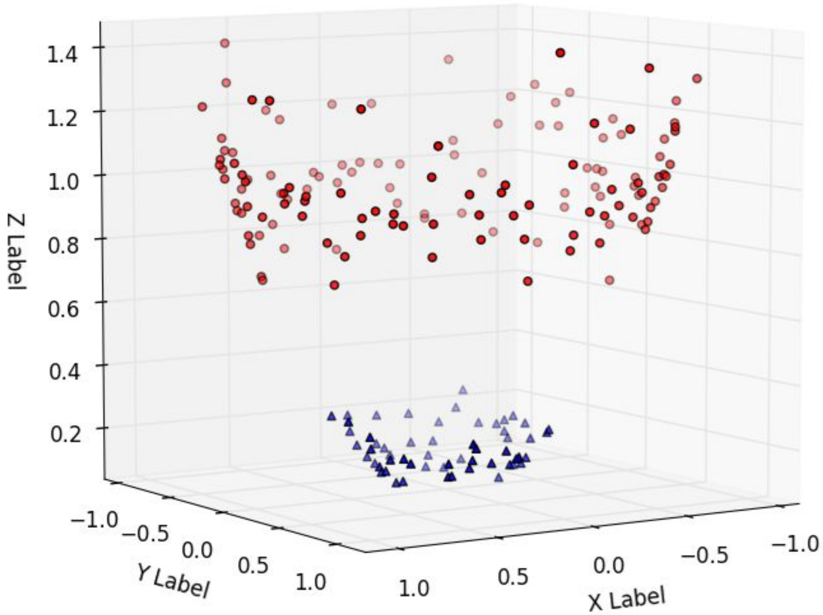
- Non-linear transform made the data ‘linearly separable’

https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Power of “nonlinearity & width & depth”



Not linearly separable



Non-linear transform & representation in higher dimension can make it linearly separable

Neural Network vs. Linear Regression

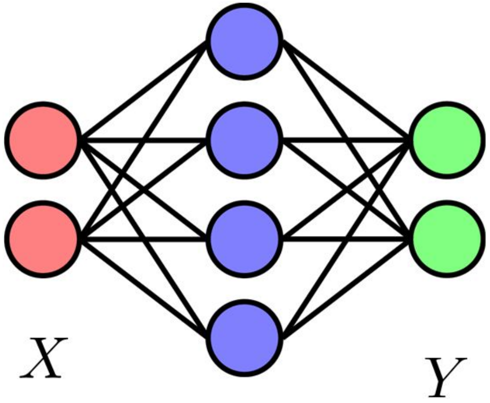
- Neural network was inspired by the structure of the biological neural network
- In supervised learning, we are simply using neural network as a parameterized function

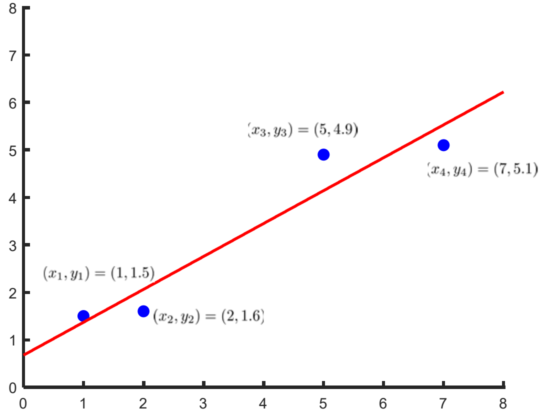
$$Y = f_W(X)$$

- In comparison, for linear regression, we are simply using a linear function as a parameterized function to “model” the relation between the input and the output of the data

$$y = f_\theta(x) = \theta_1 x + \theta_0$$

Neural Network vs. Linear Regression





$$Y = f_W(X)$$

Parameters W are optimized

to represent the relation between X and Y

$$Y = f_\theta(X)$$

Parameters θ are optimized

to represent the relation between X and Y

So, why are we using XXXXX?

- In linear regression,
  - We use a linear function, $Y = f_\theta(X)$, because we believe the input and the output have linear relationship
  - Otherwise, the regression will not be successful. In other words, the successfulness depends largely on the “model” we choose to represent the relationship between the input and the output

- With a neural network,
  - We use a nonlinear function, $Y = f_W(X)$, to represent a complex relationship between the input and the output

Neural network as a function approximator

$$Y = f_W(X)$$

- Conceptually, for an (almost) arbitrary data set, we are using a neural network to model the relationship between the input and the output
- **Universal approximation theorem**, in a nutshell, states that any continuous function can be approximated by a neural network (with a sufficient number of neurons)
- Simply put, neural network is a good model for almost any supervised learning tasks (which is why neural network is so popular)

Summary

- Artificial neural network is inspired by the architecture of biological neural networks
- Neural network can be thought of as just a parameterized function $f_W(x)$ with a set of parameters W, just like a polynomial $f_\theta(x)=\theta_2 x^2+ \theta_1 x\ \theta_0$, function with a set of parameters θ
- Neural network can perform non-linear transformation of data (exploiting depth, width, and nonlinearity)
- We will learn how a neural network can be trained in the next lecture and it is basically the same as how we train a polynomial function

References

- Lecture notes
  - CC229 lecture note
    - http://cs229.stanford.edu/notes2020fall/notes2020fall/deep_learning_notes.pdf
  - MIT 6.036 Intro to Machine Learning (Chapter 7)
    - https://www.mit.edu/~lindrew/6.036.pdf
- Website
  - CS231n course website: https://cs231n.github.io/