Contents

- Supervised learning
- Introduction to Neural Network (NN)
- Algorithm of Feedforward Neural Network (FFNN)
- Applications
- Physics Informed Neural Network (PINN)
- Convolutional Neural Network (CNN)

Classification

	Supervised Learning	Unsupervised Learning
Methods	Linear regression, nonlinear regression, etc.	K-means clustering, etc.
Data	Input and output variables will be given.	Only input data will be given
Goal	To determine the relationship between inputs and outputs so that we can predict the output when a new dataset is given	To capture the hidden patterns or underlying structure in the given input data
Uses	Regression, classification, etc.	Clustering, dimension reduction, etc.

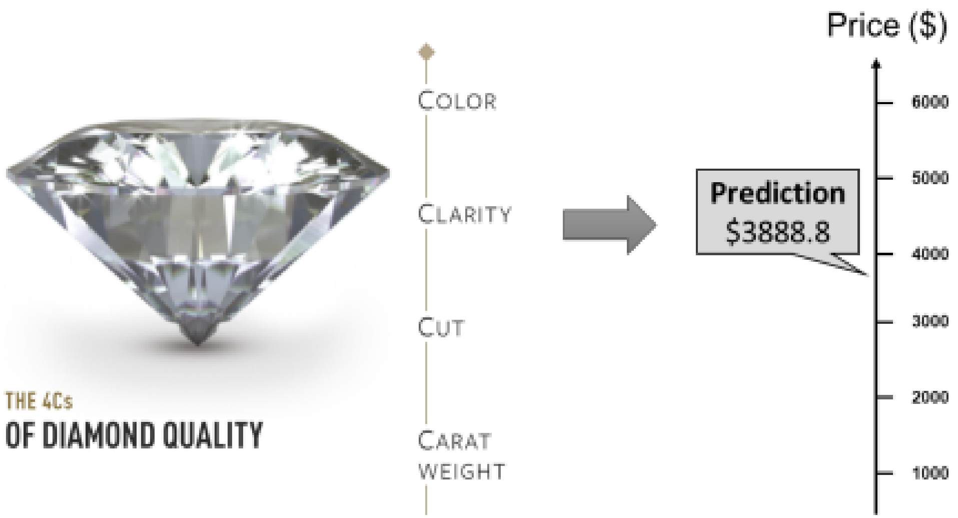
https://www.educba.com/supervised-learning-vs-unsupervised-learning/

Supervised Learning

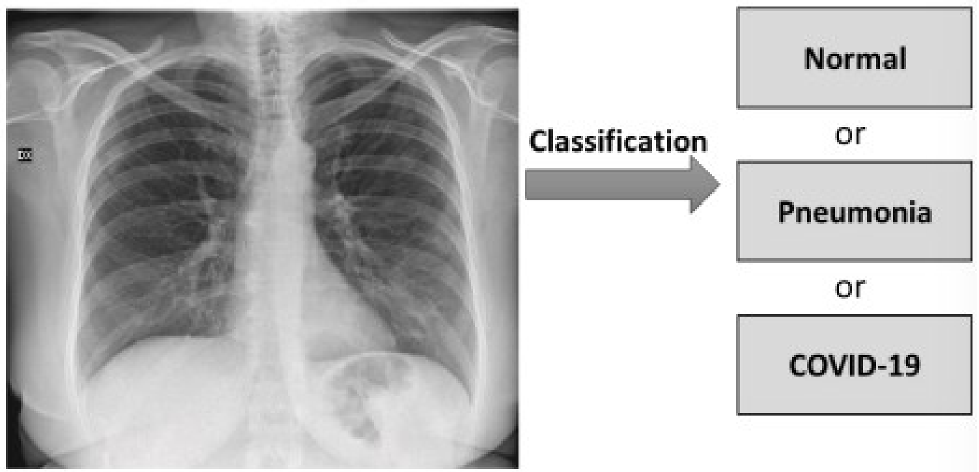
- problem of learning input-output mappings from empirical data (the training dataset)

$$\begin{cases} \text{input vector: } \mathbf{x} \Leftrightarrow \text{output } (\text{target}):\ y \rightarrow \begin{cases} \text{continuous: regression} \\ \text{discrete: classification} \end{cases} \\ \text{dataset } D \text{ of } n \text{ data points: } D = \{(\mathbf{x}_i, y_i) | i = 1, 2, \ldots, n\} \end{cases}$$

Regression:
What is the price of a diamond based on its 4Cs?

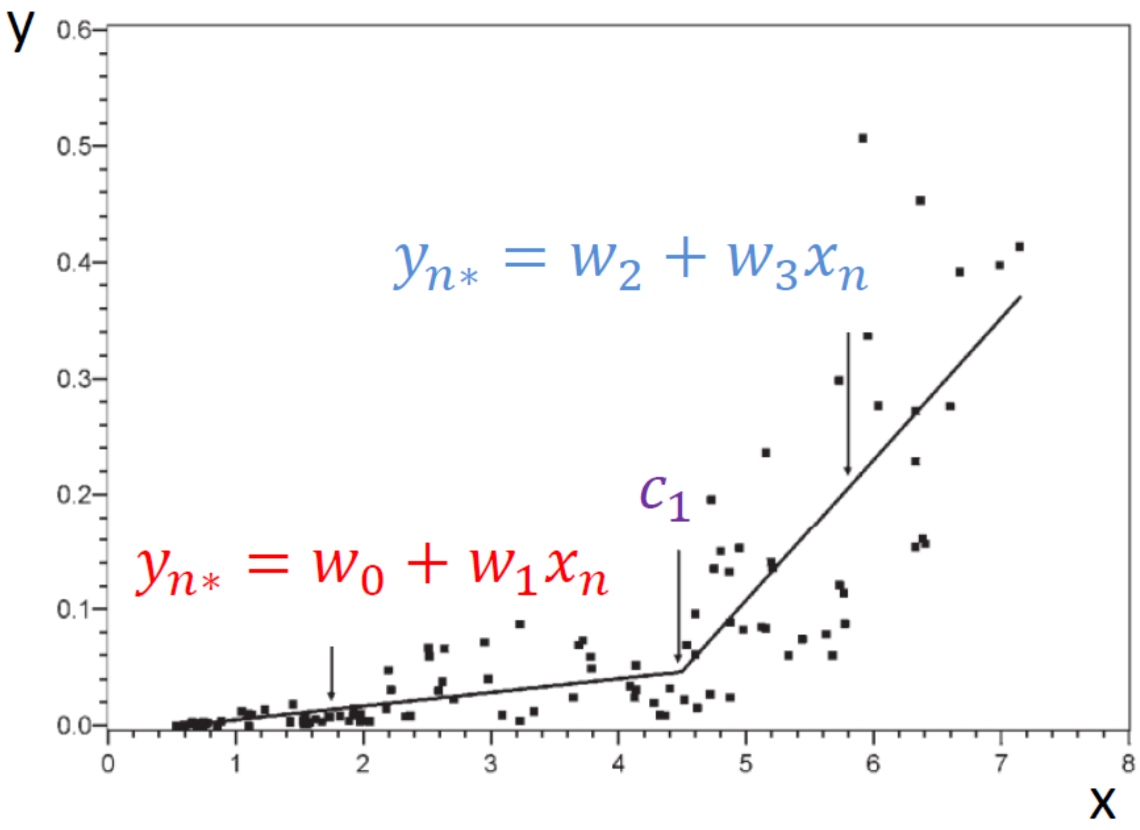


Classification:
Whether a patient has been infected with COVID-19 based on their chest X-ray image?



Recall Piecewise Linear Regression

– Another straightforward regression model is Piecewise Linear Regression
– We will show the methodology for 1 split point , but this can be extended for an arbitrary number of points



Split the data into two sets: N_1 and N_2

linear regression for N_1: $y_n^* = w_0 + w_1 x_n \quad (x < c_1)$

linear regression for N_2: $y_n^* = w_2 + w_3 x_n \quad (x > c_1)$

calculate for x-coordinate of the intercept

$$\xrightarrow[c_1 = -\frac{w_0 - w_2}{w_1 - w_3}]{w_0 + w_1 c_1 = w_2 + w_3 c_1} \begin{cases} y_n^* = w_0 + w_1 x_n & (x < c_1) \\ y_n^* = w_0 + (w_1 - w_3) c_1 + w_3 x_n & (x > c_1) \end{cases}$$

use appropriate cost function (residual sum of squares, RSS)

for N data, split into two sets of N_1 and N_2

$$c(\mathbf{w}) = \frac{1}{N_1} \sum_{n=1}^{N_1} \left(y_n^* - y_n \right)^2 + \frac{1}{N_2 - N_1} \sum_{n=N_1+1}^{N_2} \left(y_n^* - y_n \right)^2$$

We can solve this problem using Neural Network!

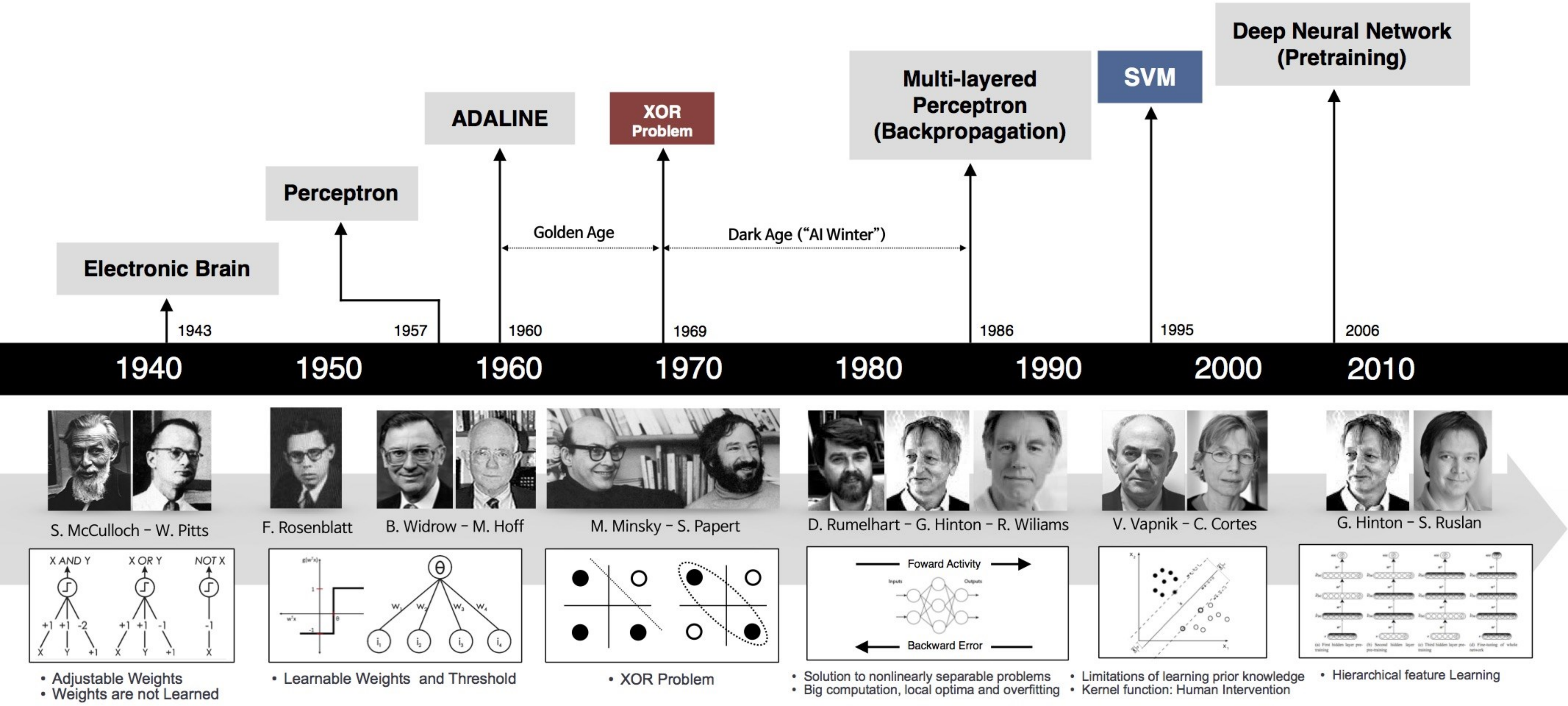
- Artificial neural networks (often called neural networks) learn (or are trained) by a set of data, which contain known inputs and outputs.
  - determine the difference between the output of the neural networks (often a prediction) and a target output (an error)
  - update its parameters (such as weights and biases) according to a learning rule based on this error value
  - Successive adjustments will cause the neural network to produce output which is increasingly similar to the target output
- Such neural networks “learn” to perform tasks by considering data without being programmed with task-specific rules.
  - X-ray images → “COVID-19” or “Normal”

Brief History (1)

- Alexey Ivakhnenko and Lapa (1976)
  - First feedforward multilayer neural networks for supervised learning
- Rina Dechter (1986)
  - The term “Deep Learning” was first introduced
- Yann LeCun et al. (1989)
  - Applied the standard backpropagation algorithm to a deep convolutional neural network (CNN) for recognizing handwritten ZIP codes on mail
- Brendan Frey and co-developer Peter Dayan and Geoffrey Hinton (1995)
  - demonstrated that it was feasible to train a network containing six fully-connected hidden layers with several hundred neurons using a wake-sleep algorithm
- Hochreiter and Schmidhuber (1997)
  - recurrent neural network (RNN) was published and called long short-term memory (LSTM), which avoided the longstanding vanishing gradient problem in deep learning

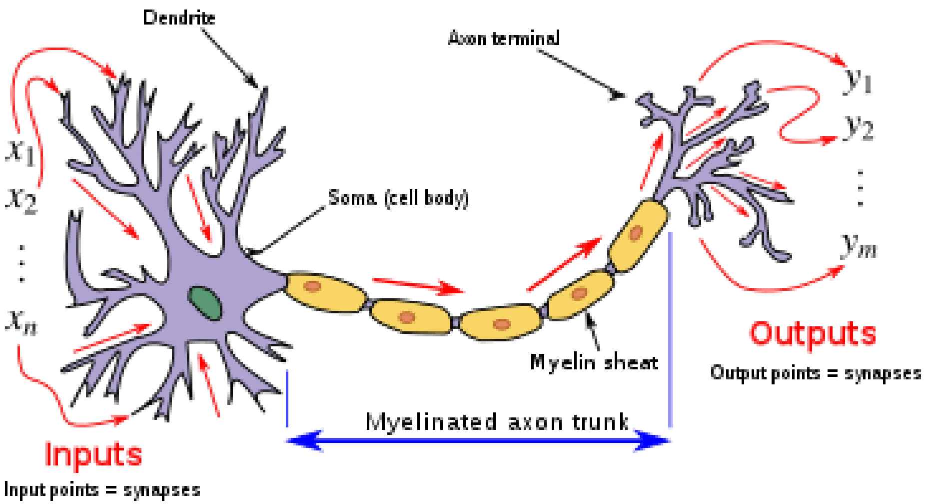
Brief History (2)

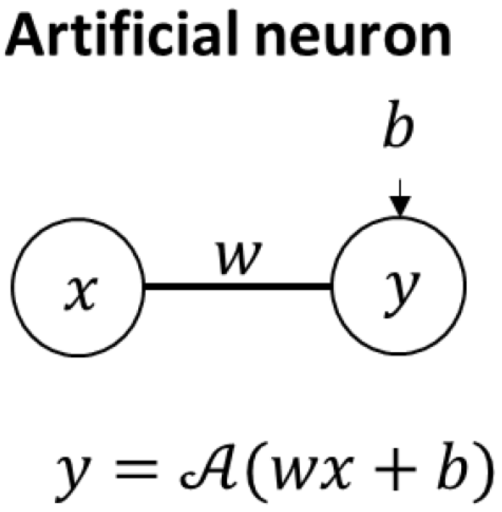
- In the early 2000s, the deep learning began to significantly impact industry
- Industrial applications of deep learning to large-scale speech recognition started around 2010
  - Advances in computational hardware have driven more interest in deep learning
- Andrew Ng (2009)
  - demonstrated that graphics processing units (GPUs) could accelerate the learning process of deep learning by more than 100 times
- Yoshua Bengio, Geoffrey Hinton, and Yann LeCun (2019)
  - recipients of the 2018 Association for Computing Machinery (ACM) Turing Award (the “Nobel Prize of Computing”) for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing


Electronic Brain
1943
Perceptron
1957
ADALINE
1960
Golden Age
XOR Problem
1969
Dark Age ("AI Winter")
Multi-layered Perceptron (Backpropagation)
1986
SVM
1995
Deep Neural Network (Pretraining)
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

A First Look at Feed Forward Neural Network (1)

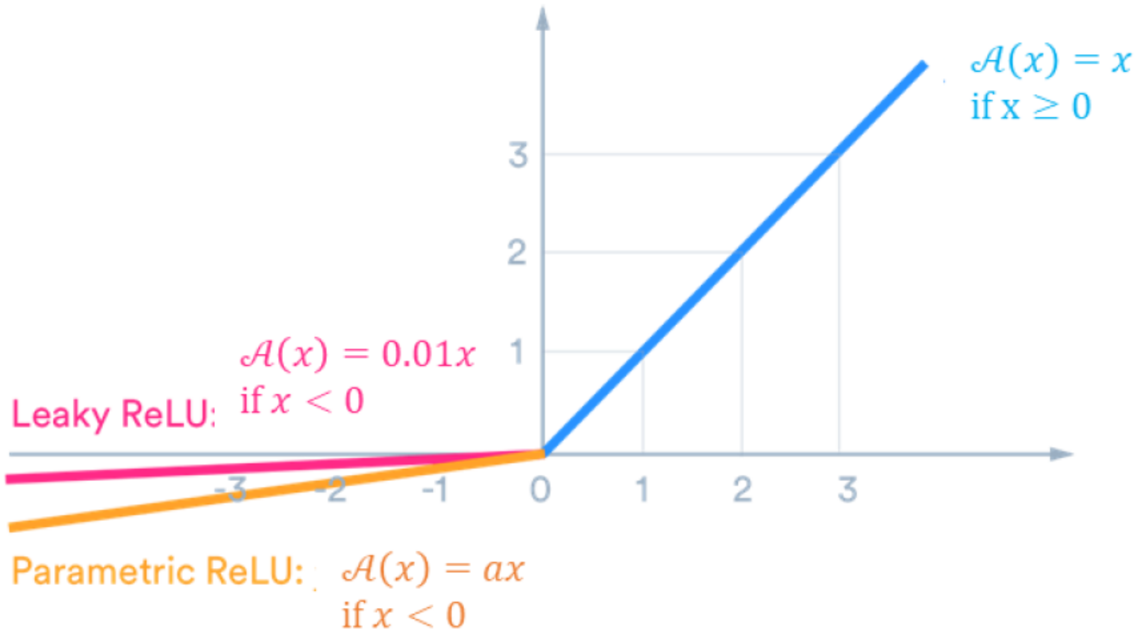
- What is Neural Networks (NNs)?
  - A network of "**artificial neurons**", a mathematical model that mimics biological neurons.
- Artificial neuron (also called a unit):
  - An output activation of a neuron a is calculated by sum of a bias b (optional) and "**linear combination**" of inputs x_i's, passing through a non-linear "**activation function**" g.

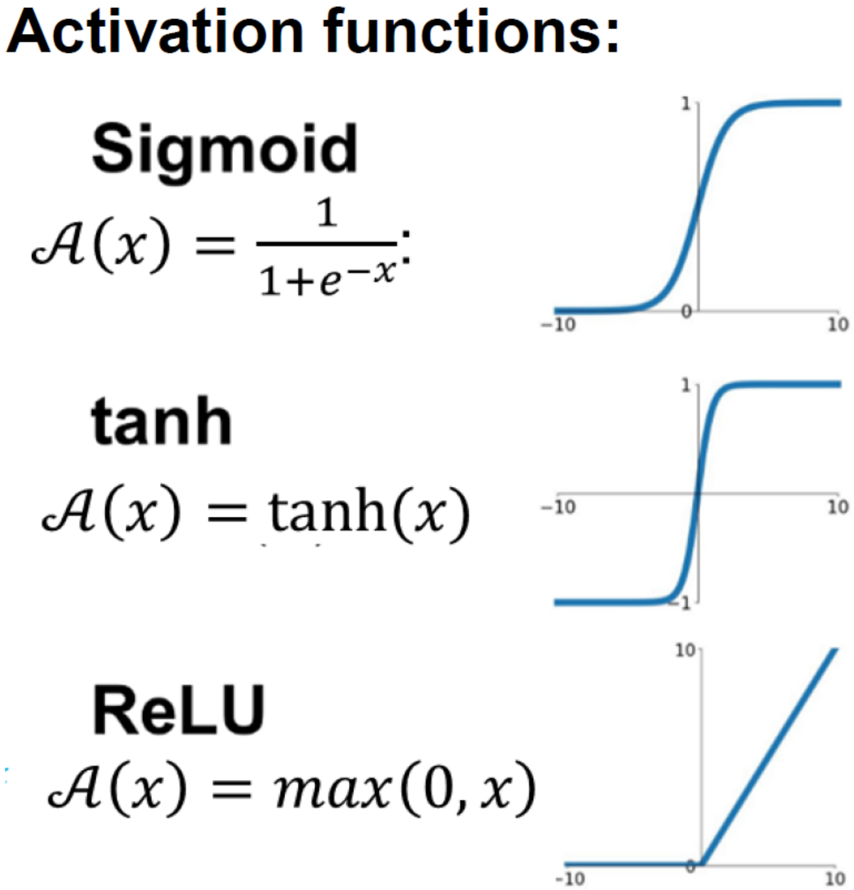




A First Look at Feed Forward Neural Network (2)

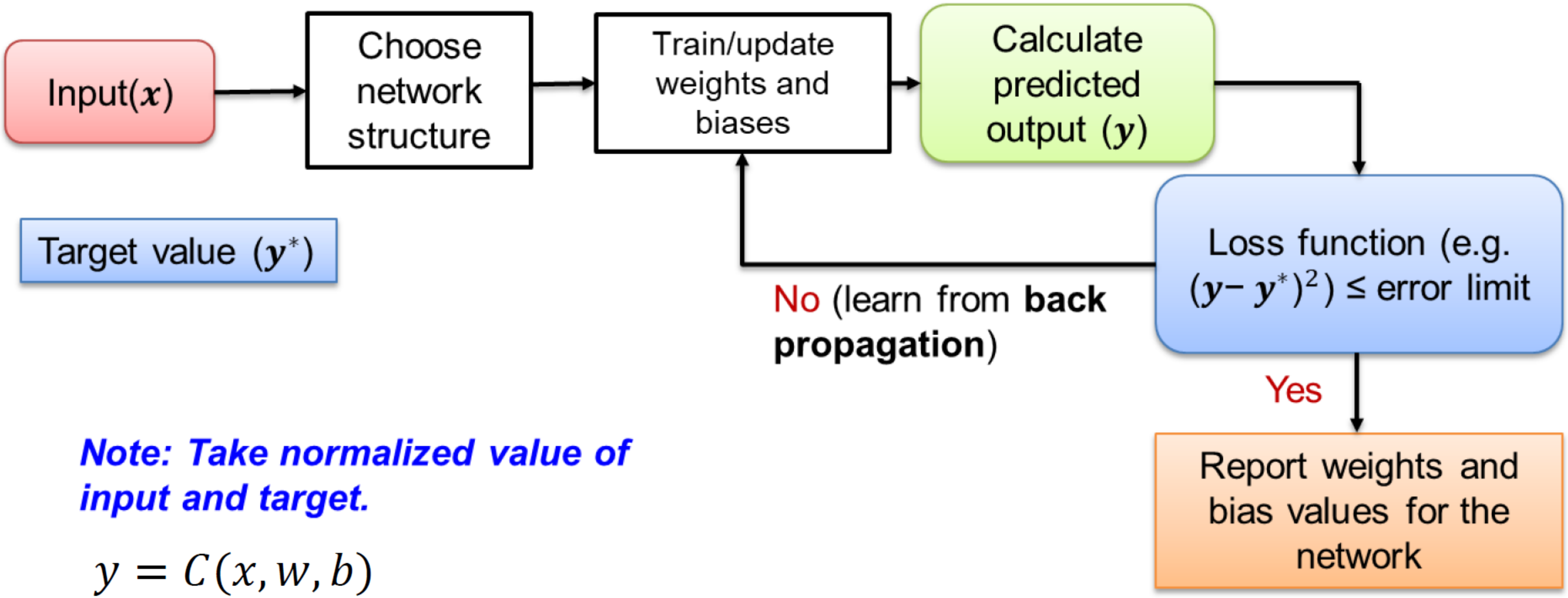
- An activation function introduces "**<u>non-linear complexities</u>**" to a model. Most common ones include Sigmoid, hyperbolic tangent, and Rectified Linear Unit, etc.





A First Look at Feed Forward Neural Network (3)

- A flowchart for training a neural network (NN)

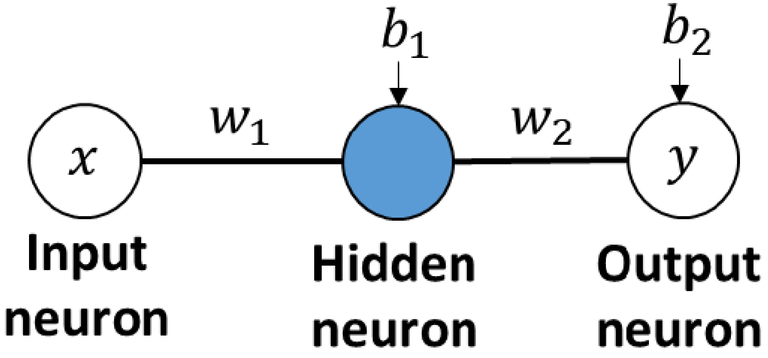


Note: Take normalized value of input and target.

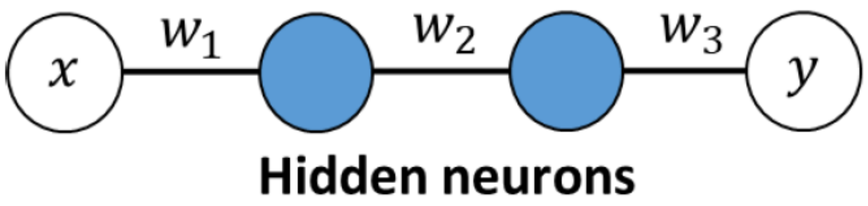
$$y = C(x, w, b)$$

Network Structure: one hidden neuron

- Step 1: initialize the weights by arbitrary values
- Step 2: compute NN output and error
- Step 3: compute increments of weights based on the gradient decent
- Step 4: update the weights
- Repeat Step 1~4: until the weights are unchanged or the error is less than a criterion



$$\mathcal{A}(w_2\mathcal{A}(w_1 x + b_1) + b_2)$$



$$y = w_3 w_2 w_1 x$$

Result of Weights: one hidden neuron

i	1	2	3	4	5
w_1	10	11.875	13.98	15.07	15.24
w_2	5	8.75	11.6	12.92	13.12
y	5	10.39	16.22	19.48	19.996
$y^* - y$	15	9.61	3.78	0.52	0.004

data point: $x^* = 0.1,\ y^* = 20$

$w_1 = 10,\ w_2 = 5,\ b_1 = 0,\ b_2 = 0$

$y = w_1 w_2 x = (10)(5)(0.1) = 5$

$L = y^* - y = y^* - w_1 w_2 x = 20 - 5 = 15$

$$\begin{cases} \Delta w_1 = -\alpha \dfrac{\partial L}{\partial w_1} = \alpha\left(y^* - y\right) w_2 x = (0.25)(15)(5)(0.1) = 1.875 \\ \Delta w_2 = -\alpha \dfrac{\partial L}{\partial w_2} = \alpha\left(y^* - y\right) w_1 x = (0.25)(15)(10)(0.1) = 3.75 \\ \alpha:\ \text{learning rate} \end{cases}$$

$$\begin{cases} w_1 = w_1 + \Delta w_1 = 10 + 1.875 = 11.875 \\ w_2 = w_2 + \Delta w_2 = 5 + 3.75 = 8.75 \end{cases}$$

Network Structure: one hidden layer, two hidden neurons

data point: $x^* = 0.1,\ y^* = 20$

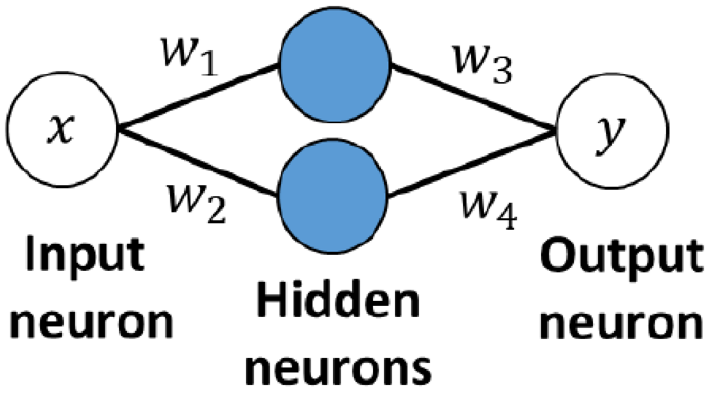
$$w_1 = 10,\ w_2 = 10,\ w_3 = 5,\ w_4 = 5,\ b_1 = 0,\ b_2 = 0,\ b_3 = 0$$

$$y = w_1 w_3 x + w_2 w_4 x = (10)(5)(0.1) + (10)(5)(0.1) = 5 + 5 = 10$$

$$L = y^* - y = y^* - w_1 w_3 x - w_2 w_4 x = 20 - 10 = 10$$

$$\begin{cases} \Delta w_1 = -\alpha \dfrac{\partial L}{\partial w_1} = \alpha\left(y^* - y\right) w_3 x = (0.25)(10)(5)(0.1) = 1.25 \\ \Delta w_2 = -\alpha \dfrac{\partial L}{\partial w_2} = \alpha\left(y^* - y\right) w_4 x = (0.25)(10)(5)(0.1) = 1.25 \\ \Delta w_3 = -\alpha \dfrac{\partial L}{\partial w_3} = \alpha\left(y^* - y\right) w_1 x = (0.25)(10)(10)(0.1) = 2.5 \\ \Delta w_4 = -\alpha \dfrac{\partial L}{\partial w_4} = \alpha\left(y^* - y\right) w_2 x = (0.25)(10)(10)(0.1) = 2.5 \\ \alpha:\ \text{learning rate} \end{cases}$$

$$\begin{cases} w_1 = w_1 + \Delta w_1 = 10 + 1.25 = 11.25 \\ w_2 = w_2 + \Delta w_2 = 10 + 1.25 = 11.25 \\ w_3 = w_3 + \Delta w_3 = 5 + 2.5 = 7.5 \\ w_4 = w_4 + \Delta w_4 = 5 + 2.5 = 7.5 \end{cases}$$

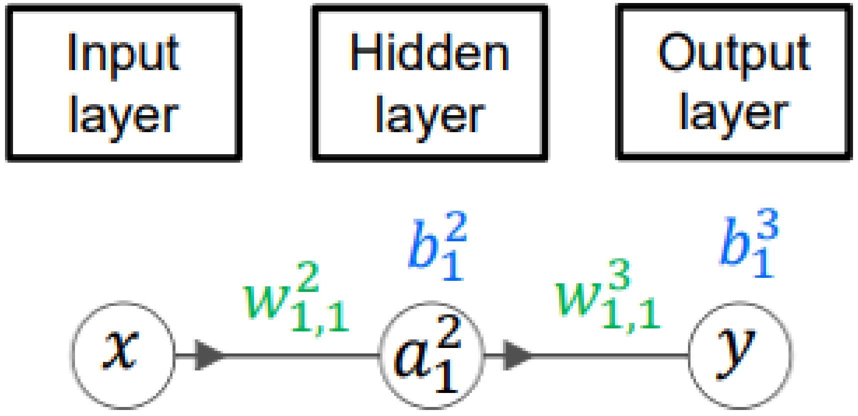


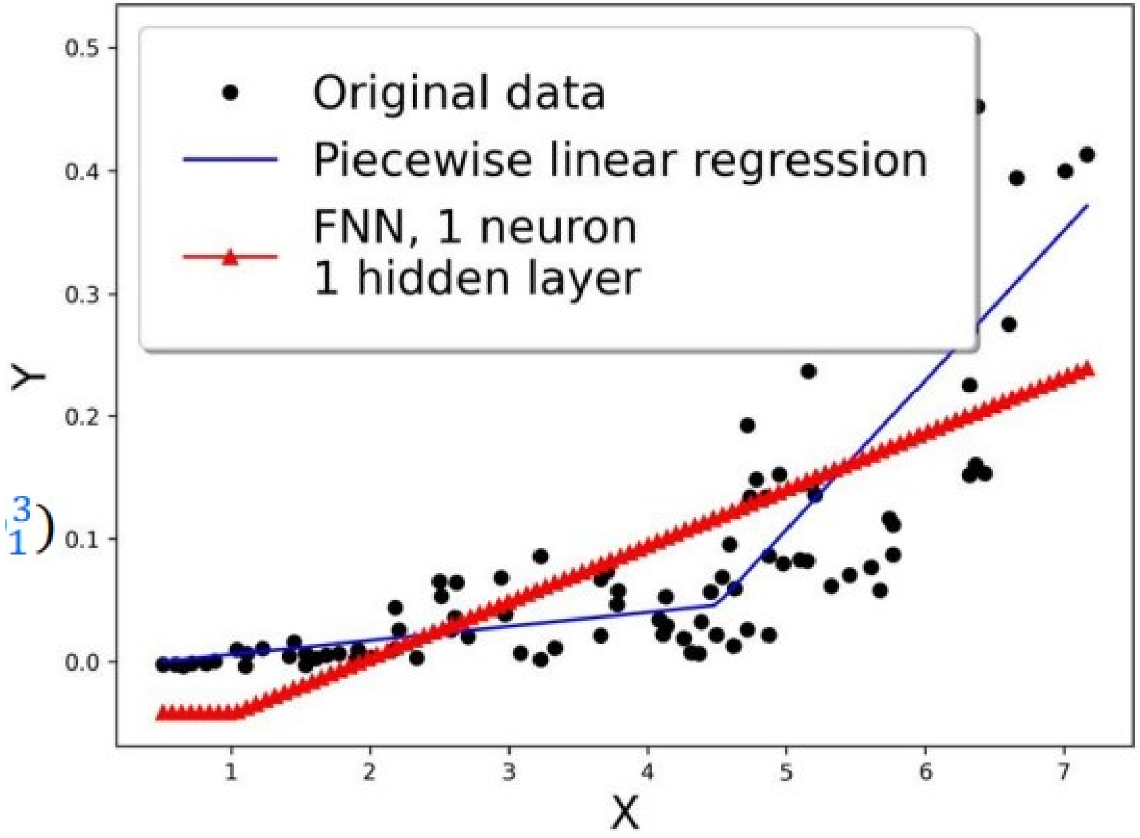
$$y = \mathcal{A}[w_3 \mathcal{A}(w_1 x + b_1) + w_4 \mathcal{A}(w_2 x + b_2) + b_3]$$

Result of Weights: one hidden layer, two hidden neurons

i	1	2	3	4
w_1	10	11.25	11.84	11.87
w_2	10	11.25	11.84	11.87
w_3	5	7.5	8.38	8.428
w_4	5	7.5	8.38	8.428
y	10	16.88	19.83	20.009
$y^* - y$	10	3.13	0.17	-0.009

1 hidden layer, 1 neuron





$$a_1^2 = A\left(w_{1,1}^2 x + b_1^2\right)$$

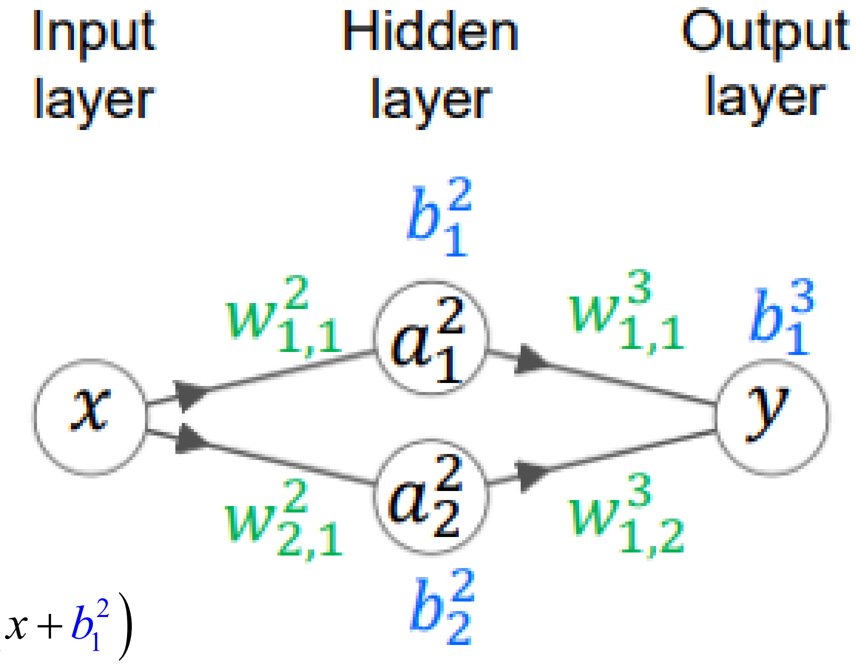
$$y = A\left(w_{1,1}^3 a_1^2 + b_1^3\right)$$

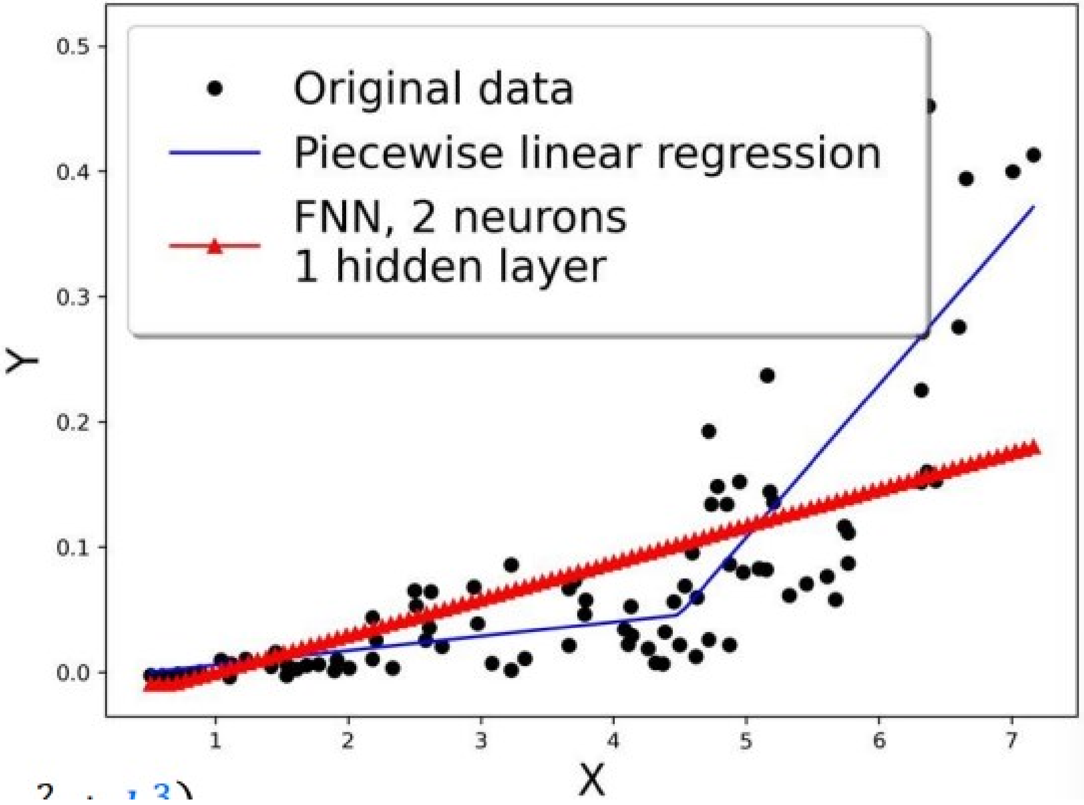
$$= A\left(w_{1,1}^3 A\left(w_{1,1}^2 x + b_1^2\right) + b_1^3\right)$$

$w_{j,k}^l$: weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer

b_j^l : bias of the j^{th} neuron in the l^{th} layer

1 hidden layer, 2 neuron





$$a_1^2 = A\left(w_{1,1}^2 x + b_1^2\right)$$

$$a_2^2 = A\left(w_{2,1}^2 x + b_2^2\right)$$

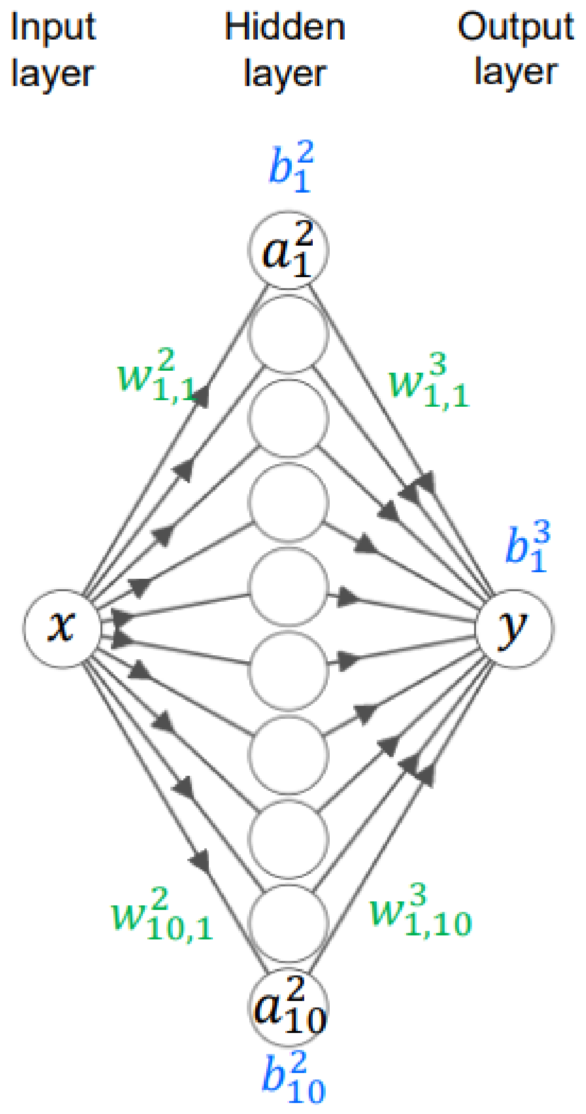
$$y = A\left(w_{1,1}^3 a_1^2 + b_1^3\right) + A\left(w_{1,2}^3 a_2^2 + b_1^3\right)$$

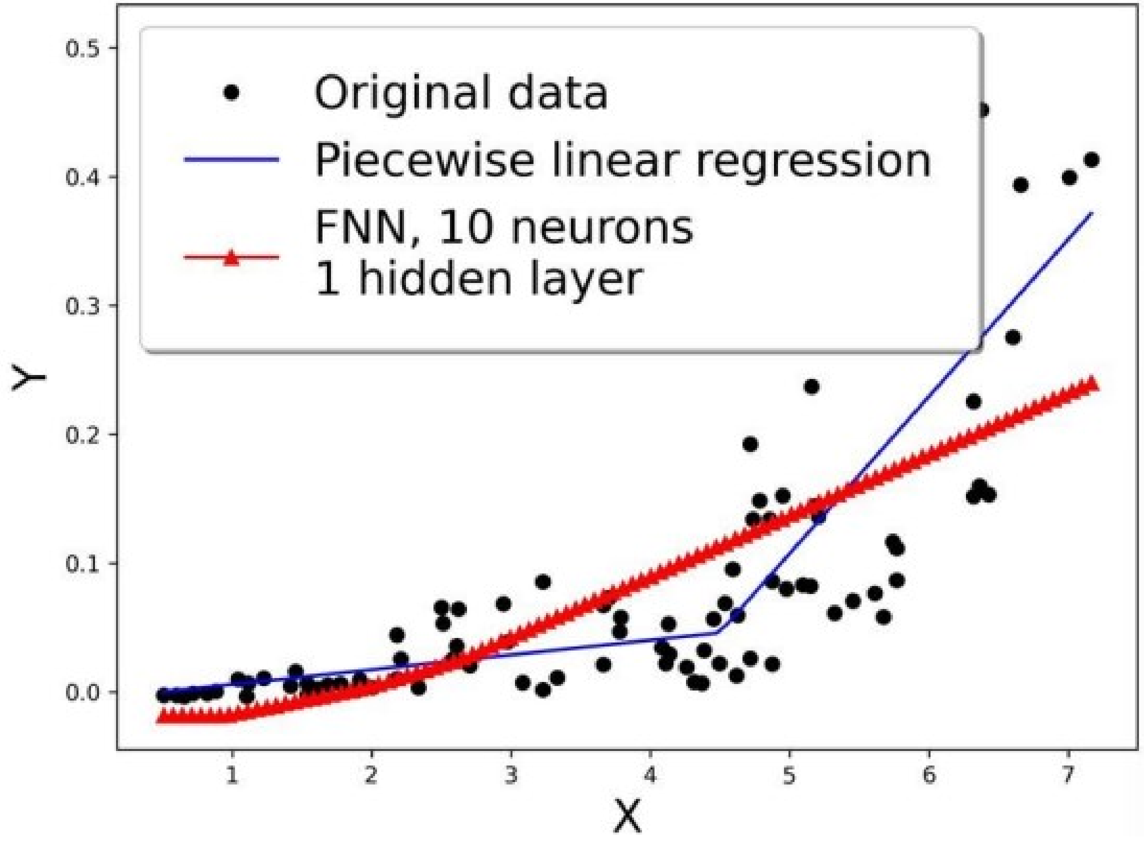
$$= A\left(w_{1,1}^3 A\left(w_{1,1}^2 x + b_1^2\right) + b_1^3\right) + A\left(w_{1,2}^3 A\left(w_{2,1}^2 x + b_2^2\right) + b_1^3\right)$$

$w_{j,k}^l$: weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer

b_j^l : bias of the j^{th} neuron in the l^{th} layer

1 hidden layer, 10 neuron

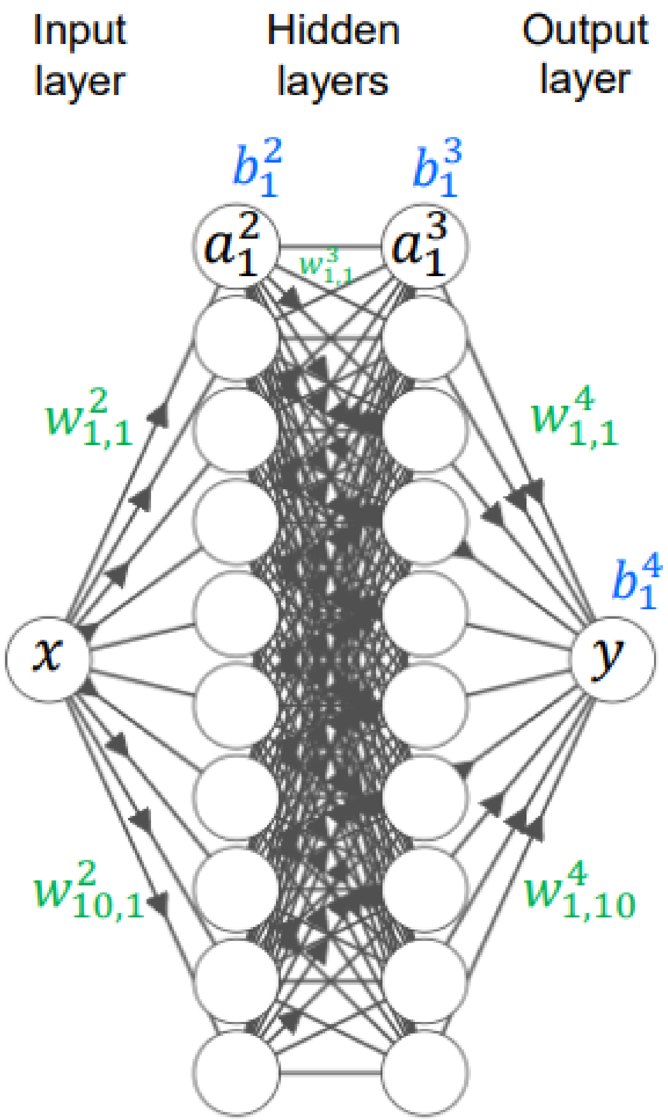


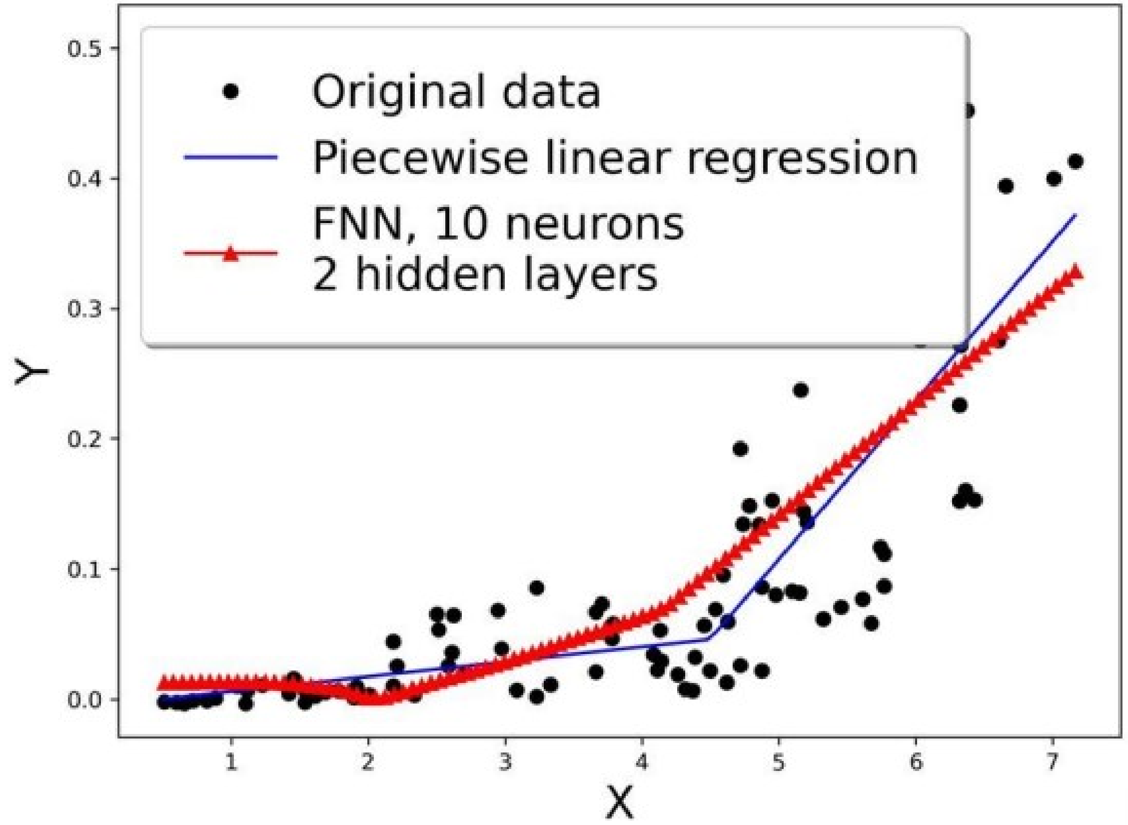


$$y = A\left(w_{1,1}^3 a_1^2 + \cdots + w_{1,10}^3 a_{10}^2 + b_1^3\right) = A\left(\sum_{j=1}^{10}\left(w_{1,j}^3 a_j^2\right) + b_1^3\right)$$

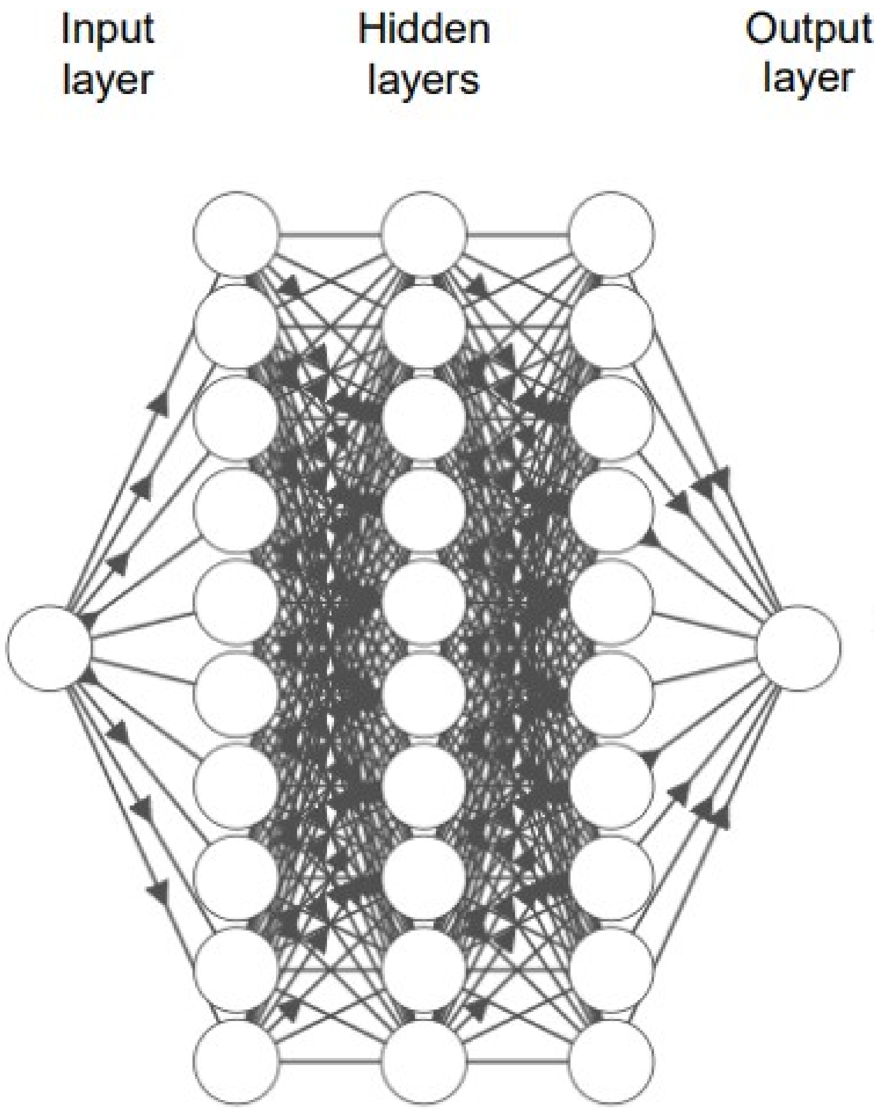
$$a_j^2 = A\left(w_{j,1}^2 x + b_j^2\right)$$

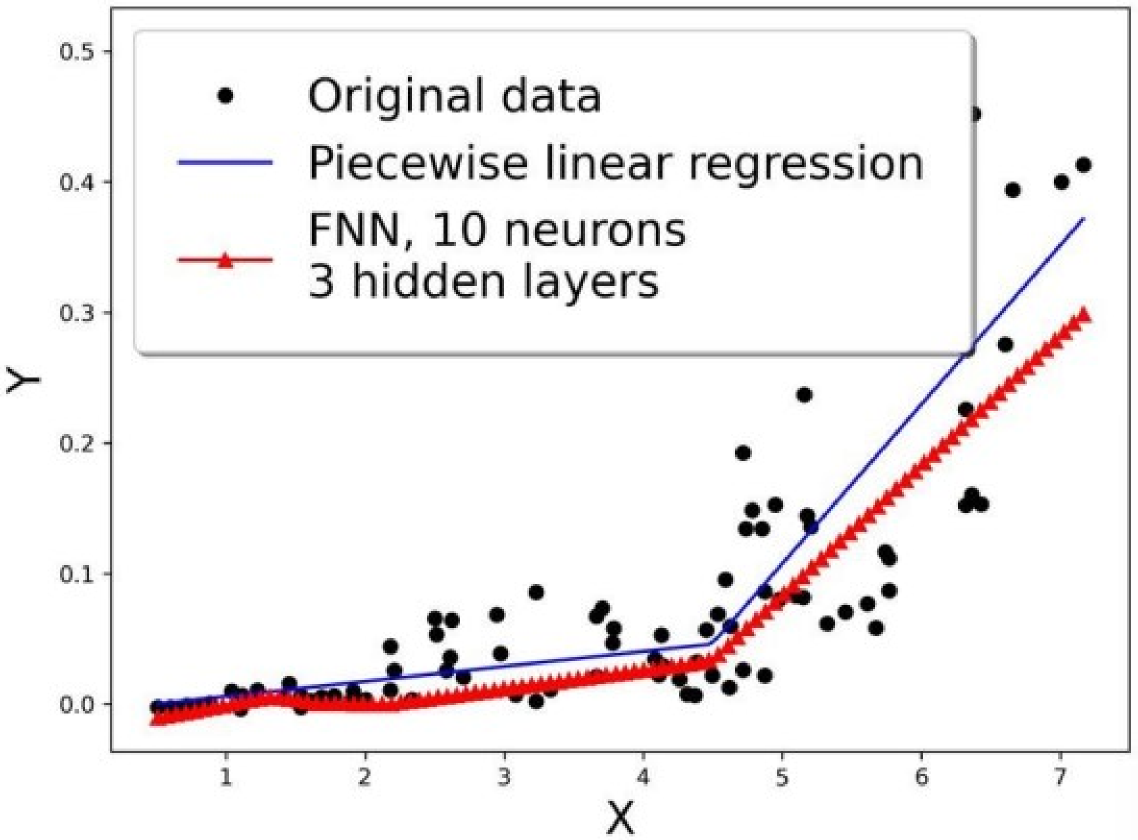
2 hidden layer, 10 neuron



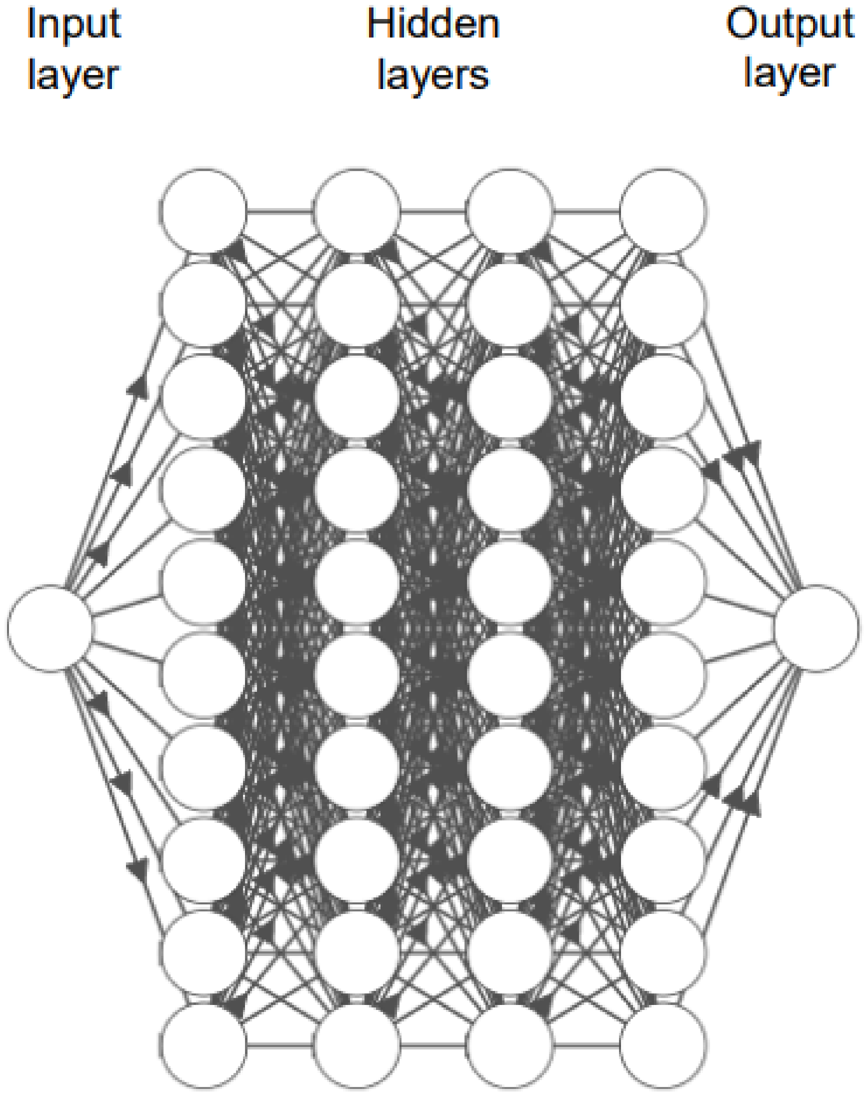


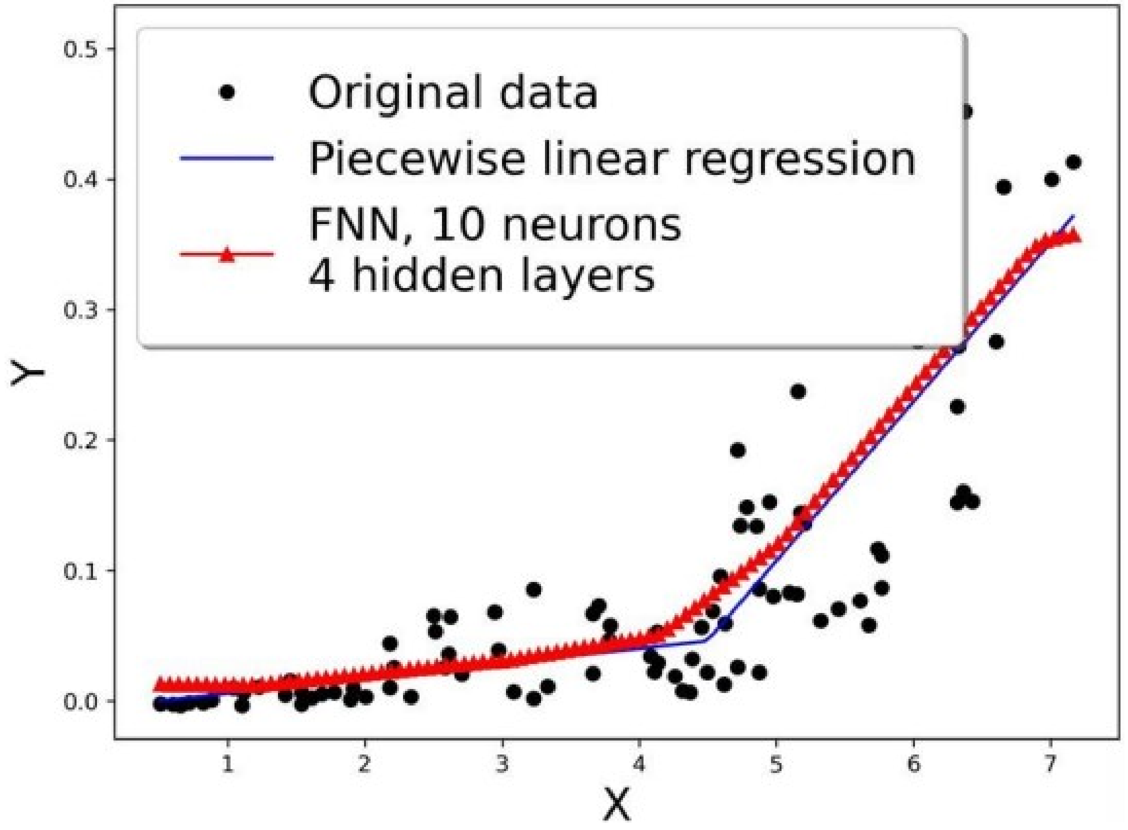
3 hidden layer, 10 neuron



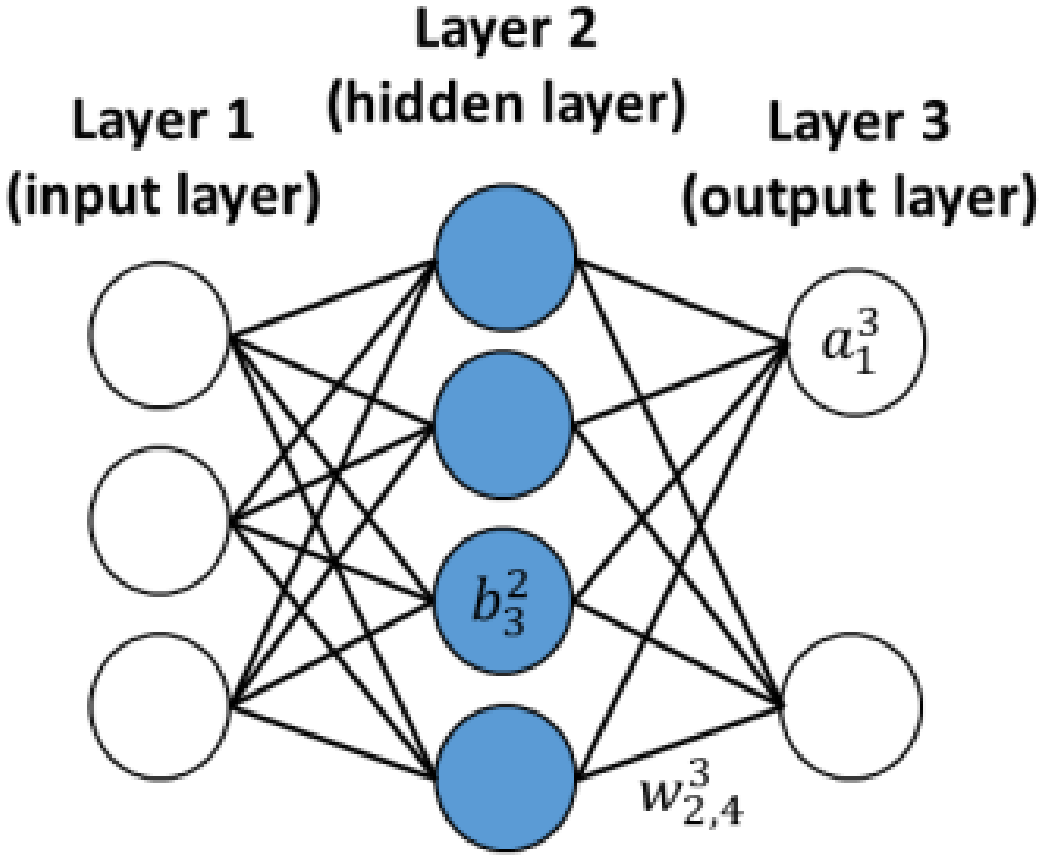


4 hidden layer, 10 neuron





General Notation for FFNN



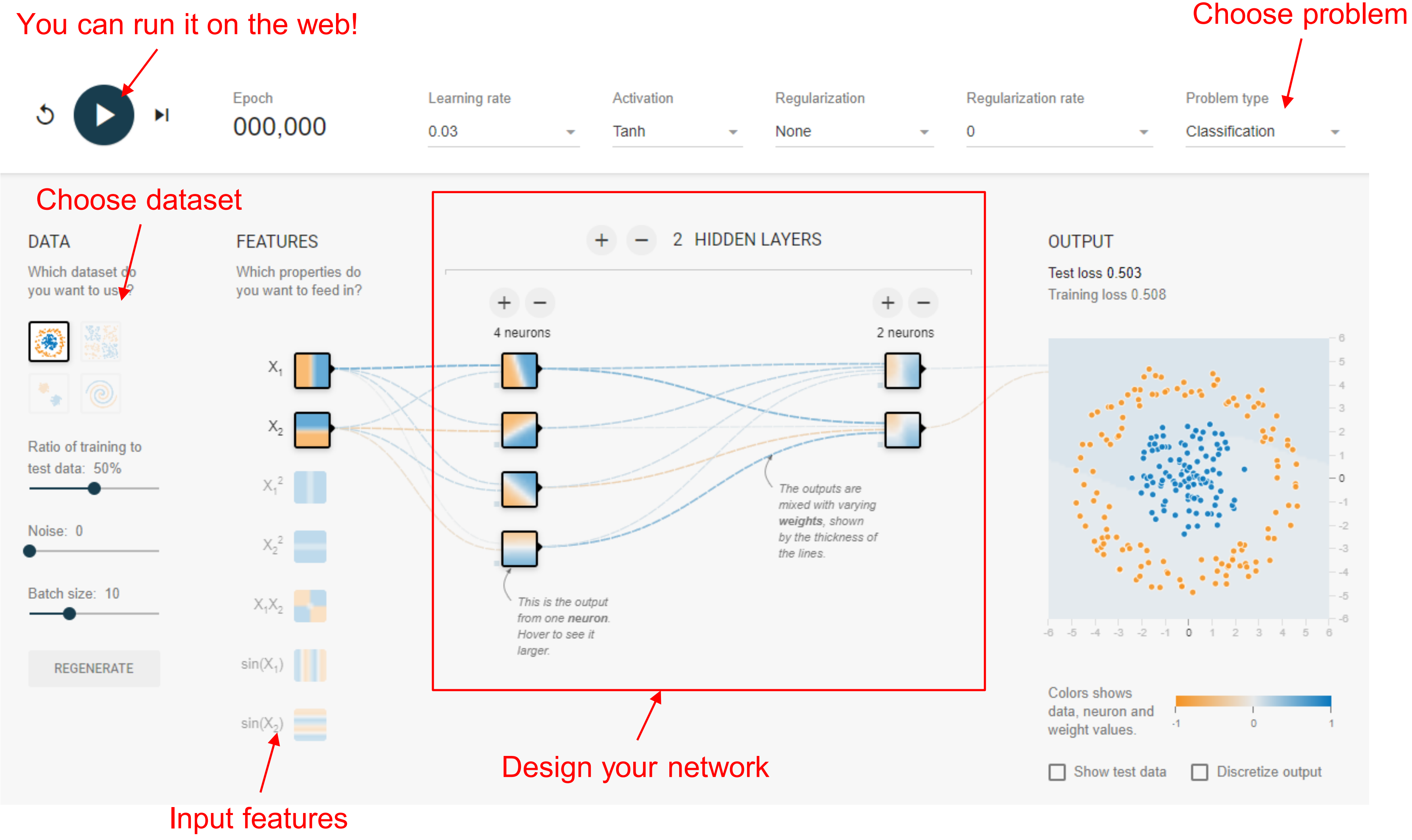
$w_{j,k}^{l}$: weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer

b_j^l : bias of the j^{th} neuron in the l^{th} layer

a_j^l : activation (or output) of the j^{th} neuron in the l^{th} layer

$$a_j^l = A\left(\sum_k \left(w_{j,k}^l a_k^{l-1}\right) + b_j^l\right)$$

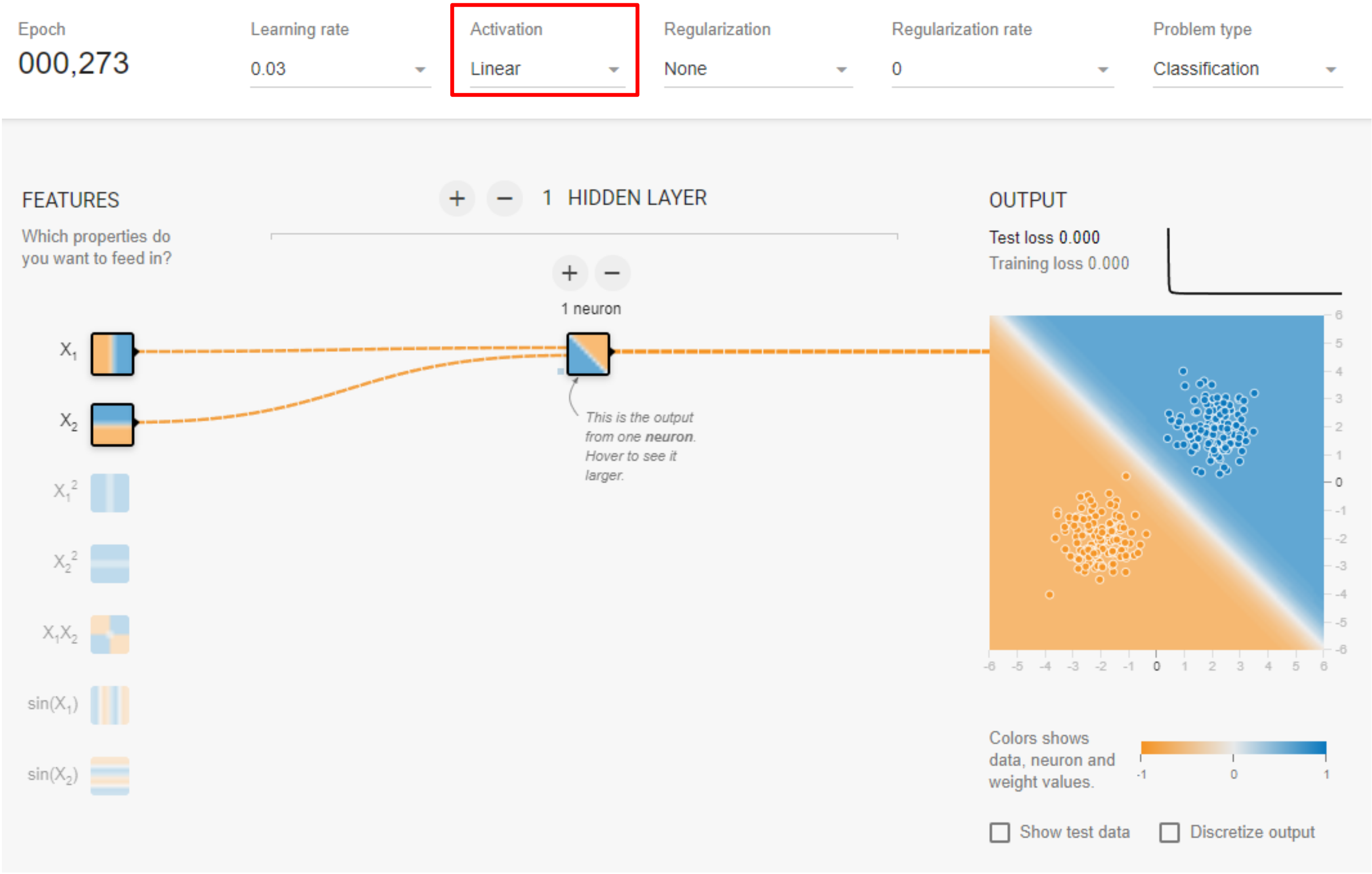
A Neural Networks Playground



https://playground.tensorflow.org/

A Neural Network Playground

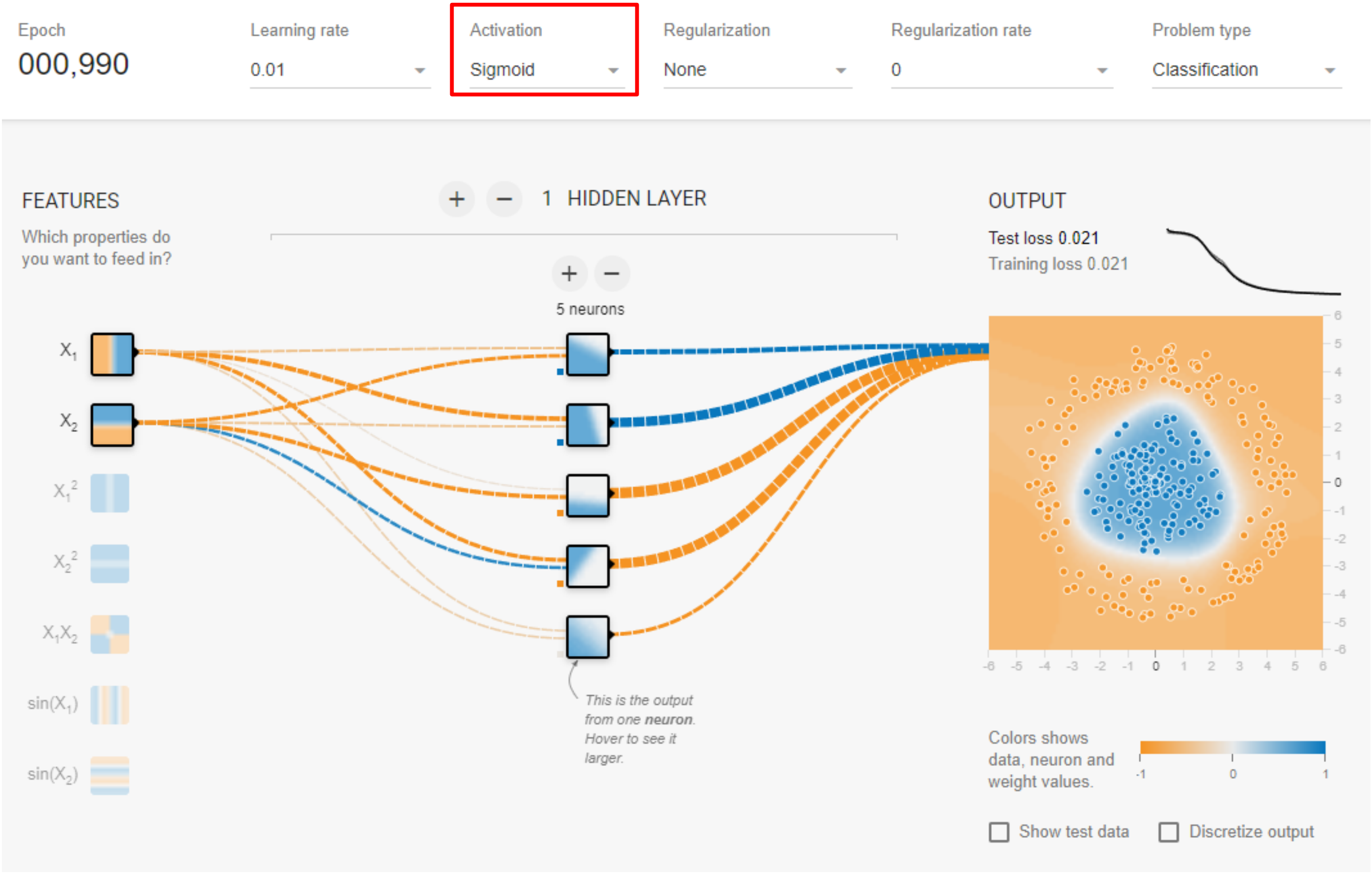
- Simple problem can be easily solved even with a linear model.



https://playground.tensorflow.org/

A Neural Network Playground

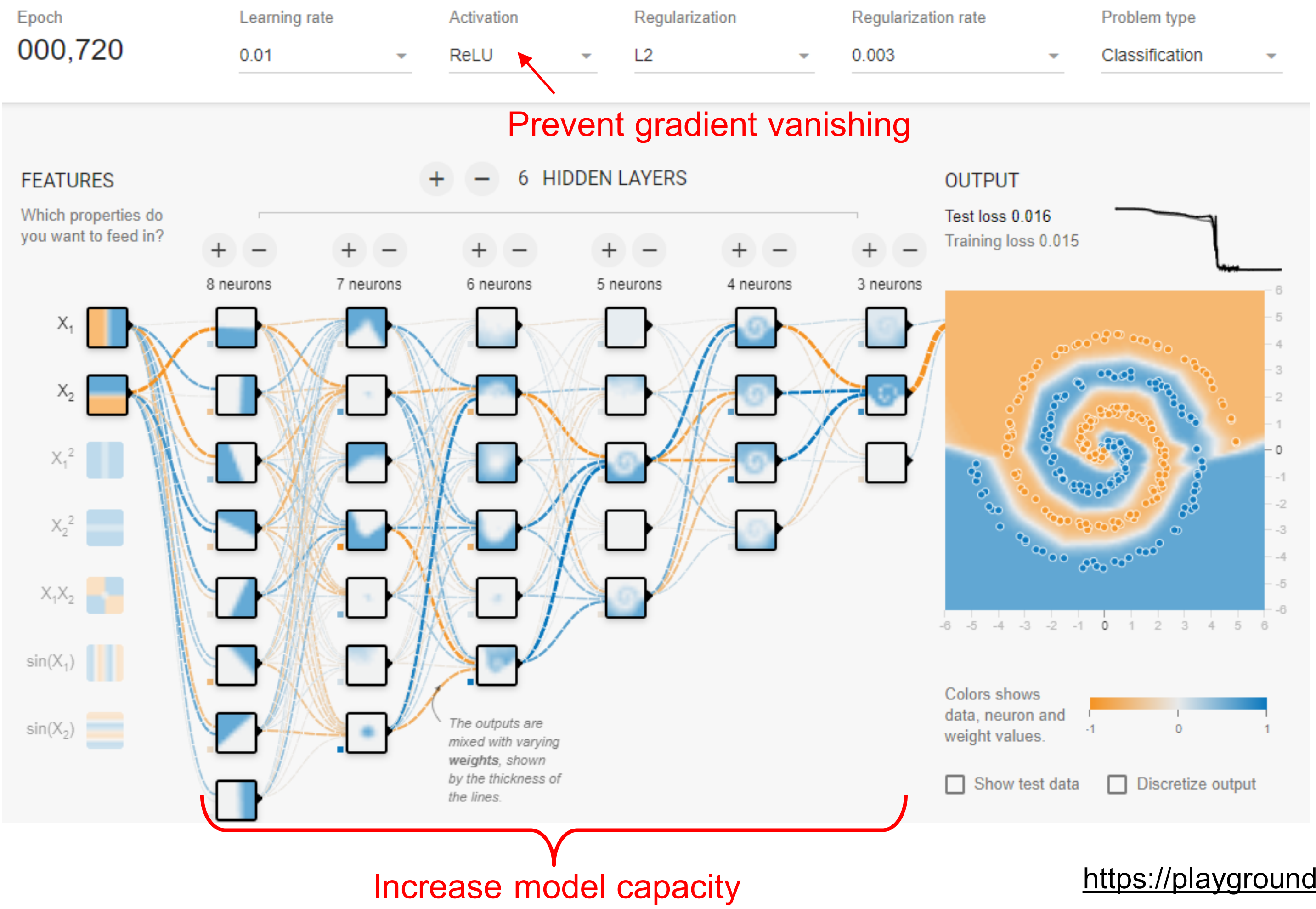
- Some problem requires non-linear complexity.



https://playground.tensorflow.org/

A Neural Network Playground

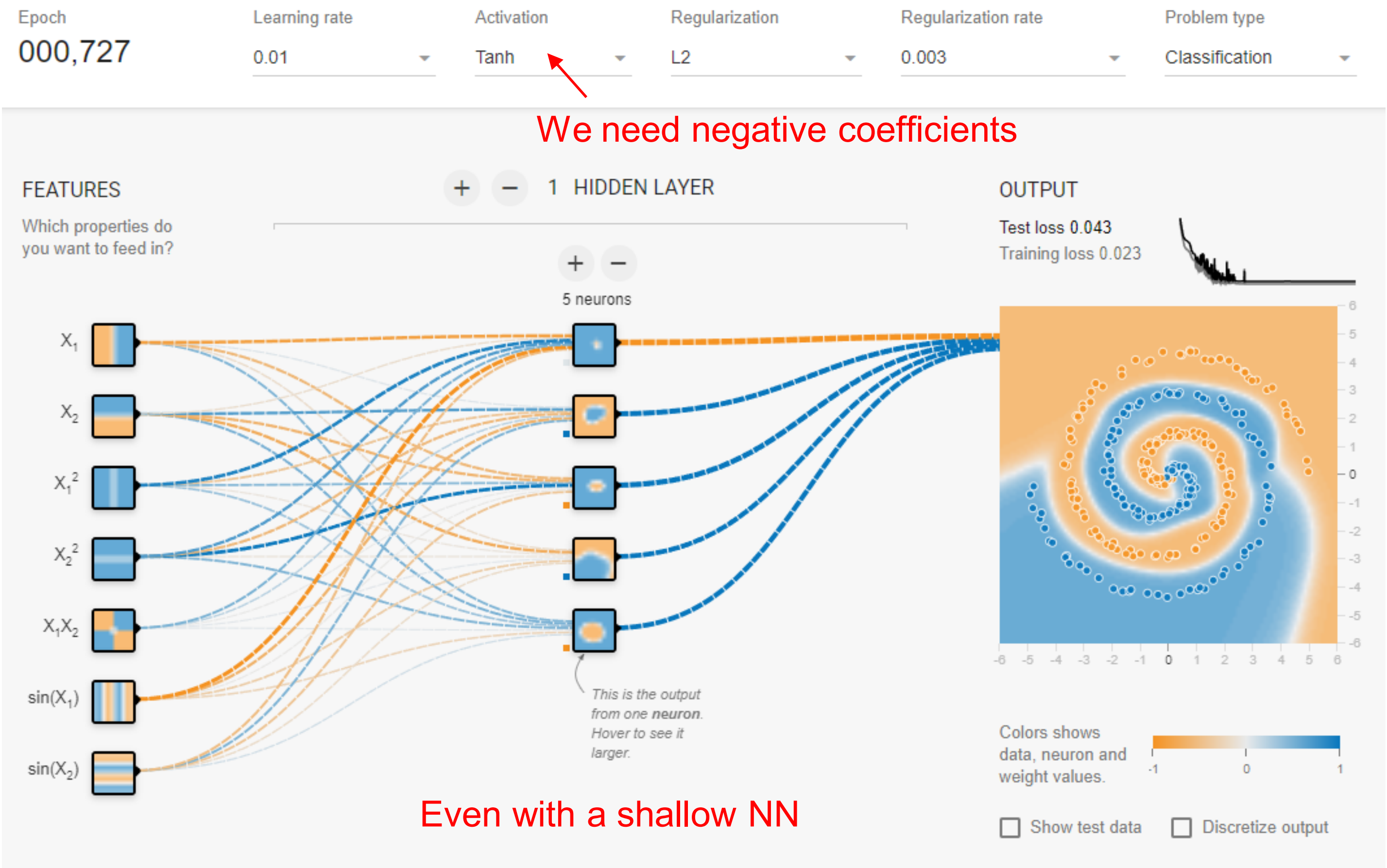
- Only delicately designed model can solve difficult problems.



https://playground.tensorflow.org/

A Neural Network Playground

- By adding more (engineered) feature, it has been solved!



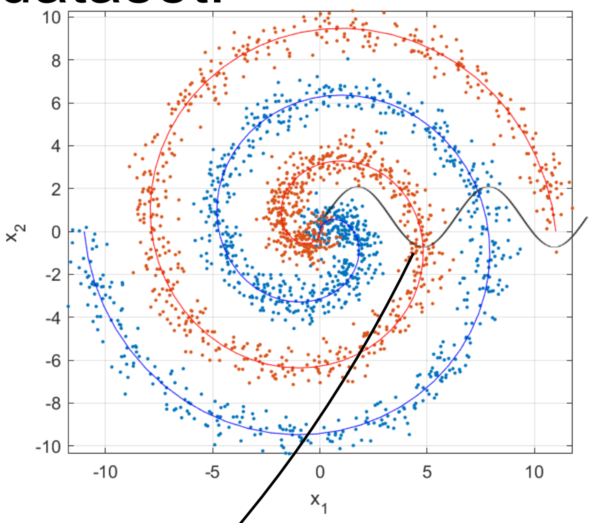
https://playground.tensorflow.org/

A Neural Network Playground

- In fact, we know the rule of "spiral" dataset.

$$x_1 = \begin{cases} t\sin(t) & \text{for label 1} \\ -t\sin(t) & \text{for label 2} \end{cases}$$

$$x_2 = \begin{cases} t\cos(t) & \text{for label 1} \\ -t\cos(t) & \text{for label 2} \end{cases}$$

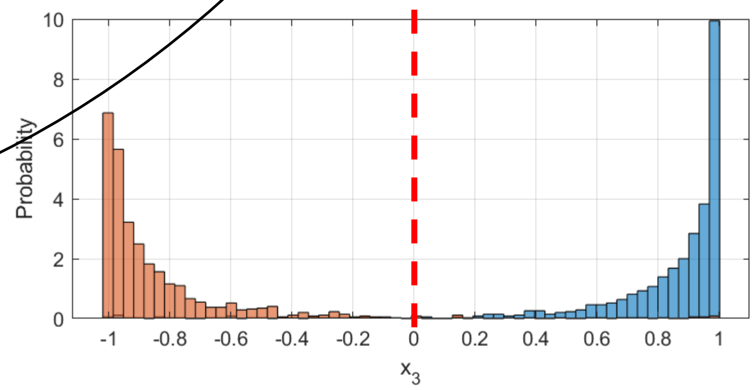


- By using engineered feature, even a linear model can classify.

$$r = \sqrt{x_1^{\ 2} + x_2^{\ 2}}$$

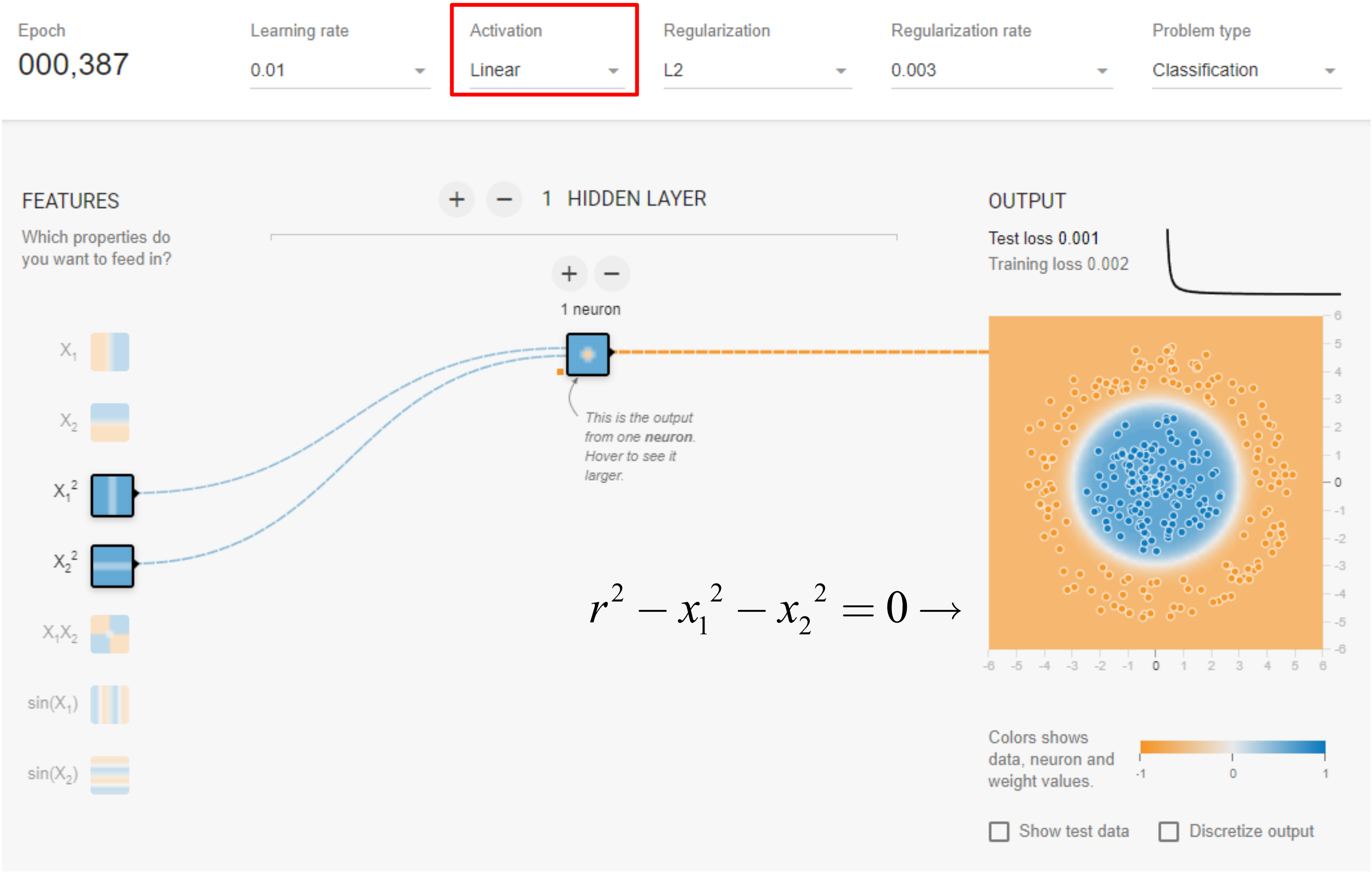
$$\varphi = \mathrm{atan}\left(x_1 / x_2\right)$$

$$x_3 = \sin\left(r + \varphi\right)$$



A Neural Network Playground

- Another well engineered feature input.



$$r^2 - x_1{}^2 - x_2{}^2 = 0 \rightarrow$$

https://playground.tensorflow.org/

Problems 11–13 use the blue ball, orange ring example on playground.tensorflow.org with one hidden layer and activation by ReLU (not Tanh). When learning succeeds, a white polygon separates blue from orange in the figure that follows.

11 Does learning succeed for $N = 4$? What is the count $r(N, 2)$ of flat pieces in $F(x)$? The white polygon shows where flat pieces in the graph of $F(x)$ change sign as they go through the base plane $z = 0$. How many sides in the polygon ?

12 Reduce to $N = 3$ neurons in one layer. Does F still classify blue and orange correctly ? How many flat pieces $r(3, 2)$ in the graph of $F(v)$ and how many sides in the separating polygon ?

13 Reduce further to $N = 2$ neurons in one layer. Does learning still succeed ? What is the count $r(2, 2)$ of flat pieces ? How many folds in the graph of $F(v)$? How many sides in the white separator ?

14 Example 2 has blue and orange in two quadrants each. With one layer, do $N = 3$ neurons and even $N = 2$ neurons classify that training data correctly ? How many flat pieces are needed for success ? Describe the unusual graph of $F(v)$ when $N = 2$.

15 Example 4 with blue and orange spirals is much more difficult ! With one hidden layer, can the network learn this training data ? Describe the results as N increases.

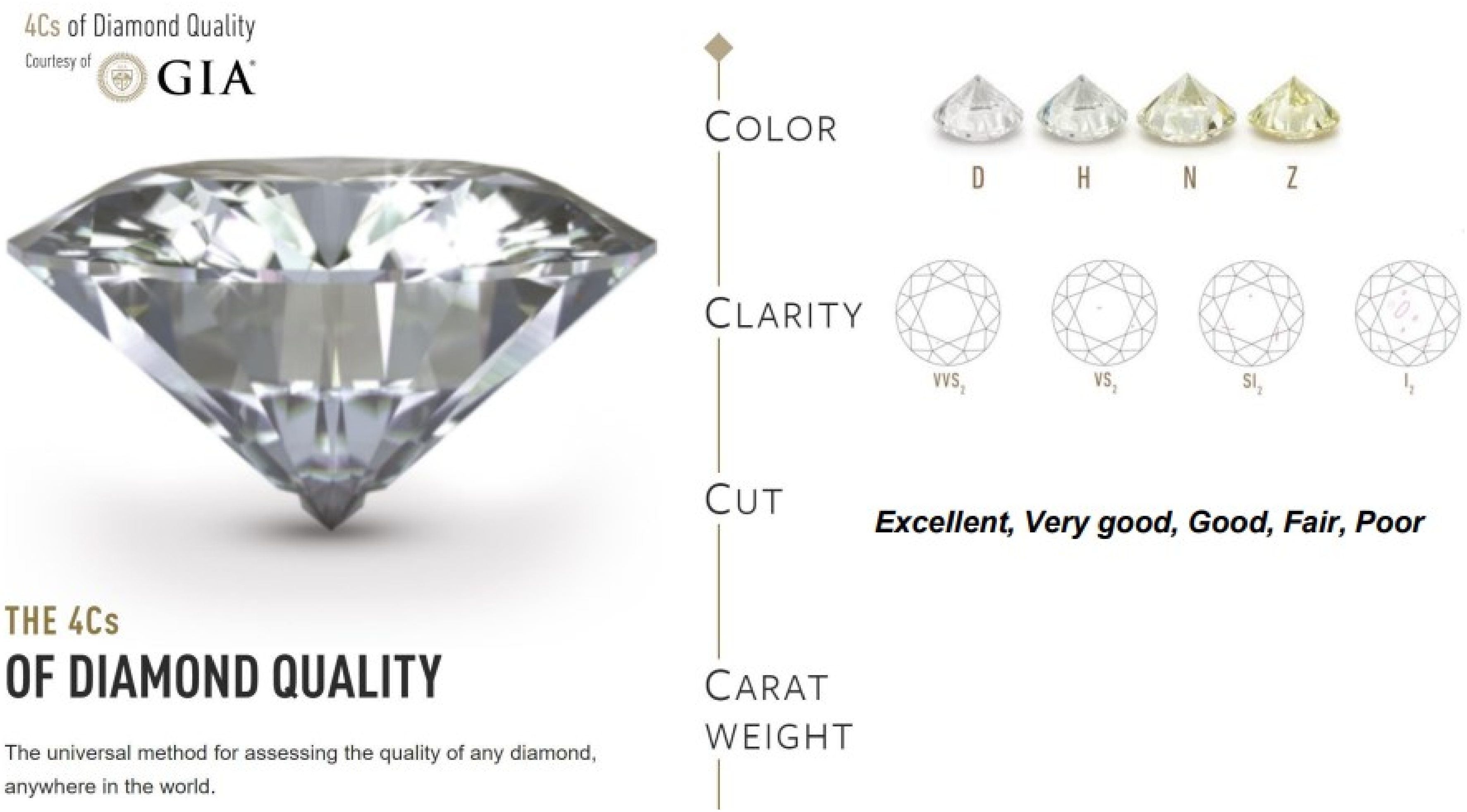
16 Try that difficult example with two hidden layers. Start with $4 + 4$ and $6 + 2$ and $2 + 6$ neurons. Is $2 + 6$ better or worse or more unusual than $6 + 2$?

17 How many neurons bring complete separation of the spirals with two hidden layers ? Can three layers succeed with fewer neurons than two layers ?

I found that $4 + 4 + 2$ and $4 + 4 + 4$ neurons give very unstable iterations for that spiral graph. There were spikes in the training loss until the algorithm stopped trying. playground.tensorflow.org (on our back cover!) was a gift from Daniel Smilkov.

Application: Diamond Price Regression

4Cs:



Excellent, Very good, Good, Fair, Poor

Diamond Dataset

- Kaggle (Datasets/Diamonds)
  - 53,940 diamonds with 10 features

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63
5	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61	338	4	4.05	2.39

Price: ($326--$18,823)
Carat: (0.2--5.01)
Cut: (Fair, Good, Very Good, Premium, Ideal)
Color: (J (worst) to D (best))
Clarity: (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
Size in x direction in mm (0--10.74)
Size in y direction in mm (0--58.9)
Size in z direction in mm (0--31.8)
Depth: **z** / mean(x, y) = 2 * **z** / (x + y) (43--79) (%)
Table: width of top of diamond relative to widest point (43--95) (%)

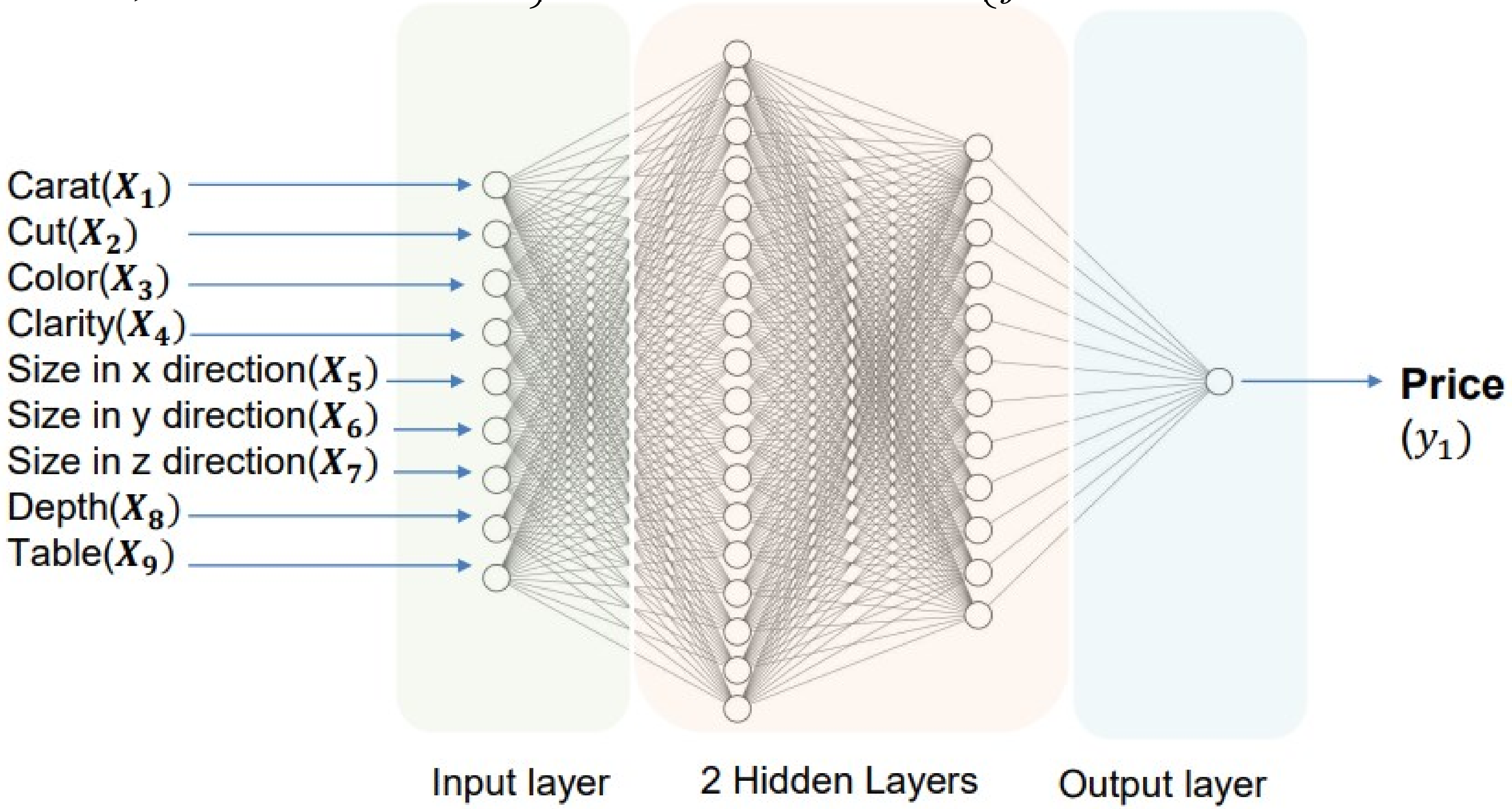
Color (D, E, F, G, H, I, J) → (1, 2, 3, 4, 5, 6, 7)
D: colorless ~ Z: light yellow or brown

Cut Rating	Numerical value
Premium	1
Ideal	2
Very Good	3
Good	4
Fair	5

Clarity Rating	Numerical value
IF—Internally Flawless	1
VVS1,2—Very, Very Slightly Included 1,2	2
VS1,2—Very Slightly Included 1,2	3
SI1,2—Slightly Included 1,2	4
I1—Included 1	5

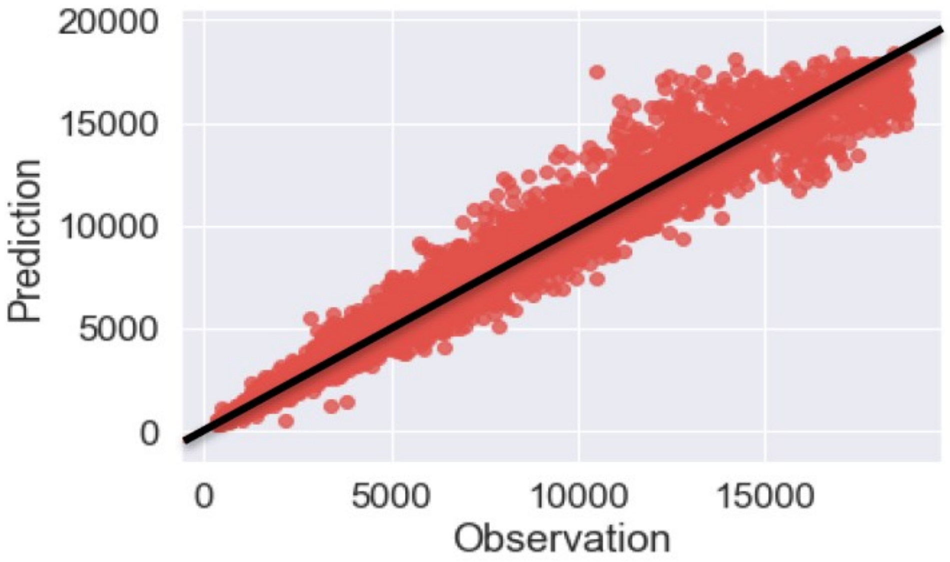
Neural Network Architecture

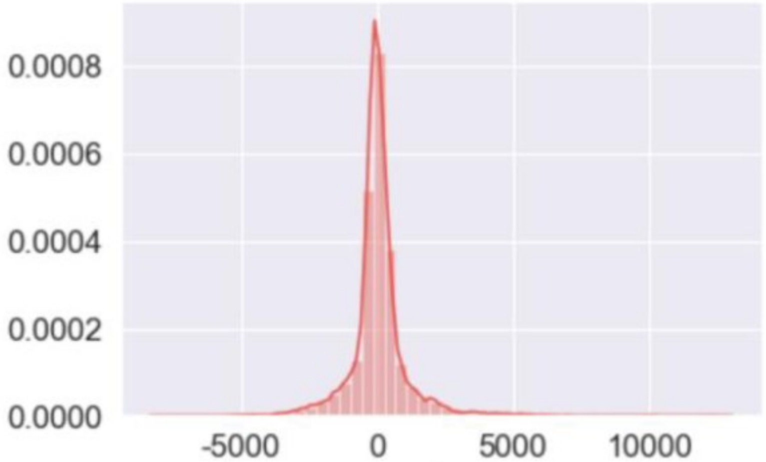
9 input variables $\left(\mathbf{x}_i^N\right)$, $i = 1, \ldots 9$ and $N = 53940$ —Find the unknown function→ 1 output variables $\left(\mathbf{y}_j^N\right)$, $j = 1$ and $N = 53940$



Network Training and Prediction

- Loss: MSE(mean squared error)
- Optimizer: Adam
- Architecture:
- Metrics
  - Mean absolute error: 615.97
  - Mean squared error: 1139999.82
  - R-squared: 0.93





Prediction of Unseen data

	carat	cut	color	clarity	depth	table	x	y	z
0	4.0	1	0	0	65.5	59	10.74	10.54	6.98
1	2.0	1	0	0	65.5	59	10.74	10.54	6.98
2	0.5	2	2	1	61.4	56	4.33	4.37	2.67
3	0.3	0	1	2	59.7	58	4.13	4.14	2.47

```
Prediction: [12148.8  9818.9  1573.8   503.2]
```