



Samsung Innovation Campus

| Artificial Intelligence Course

Together for Tomorrow!
Enabling People

Education for Future Generations

Chapter 8.

Neural Network and Deep Learning

Artificial Intelligence Course

Chapter Description

◆ Chapter objectives

- ✓ Build and train the deep neural networks.
- ✓ Optimize the deep learning neural networks preventing the overfitting and vanishing gradient problems.
- ✓ Gain proficiency with the deep learning libraries such as TensorFlow and Keras.

◆ Chapter contents

- ✓ Unit 1. Basics of Neural Network
- ✓ Unit 2. TensorFlow Basics
- ✓ Unit 3. Deep Learning Methods with TensorFlow and Keras

Unit 1.

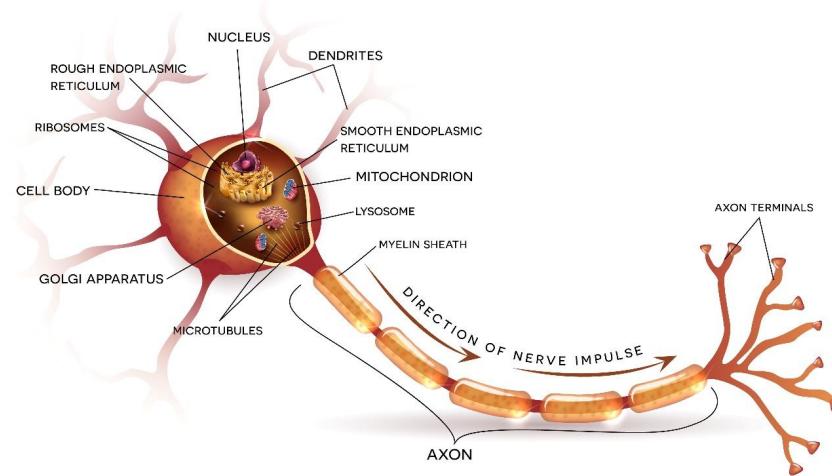
Basics of Neural Network

- 1.1. Understanding Perceptron with Excel
- 1.2. Understanding Multilayer Perceptron with Excel
- 1.3. From Multilayer Perceptron to Deep Learning
- 1.4. Error Backpropagation and Gradient Descent to Reduce Errors
- 1.5. Activation Function
- 1.6. Deep Learning
- 1.7. Problems with Deep Learning and the Solutions

Overview

Biological origin:

- Human's neuron is the smallest information processing unit that composes the neural system. It consists of a cell body, dendrites, and an axon. Below is a more simplified version of the above figure.

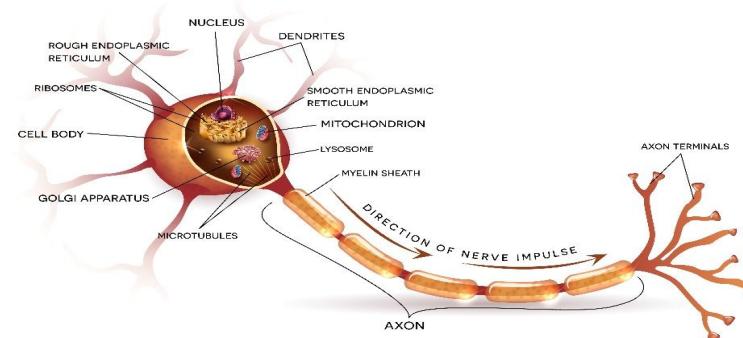


- A cell body transmits simple operations, a dendrite transmits received signals, and an axon transmits the executed results.

Overview

Biological origin:

- They receive and store various information by exchanging chemical signals with adjacent neurons through a structure named synapse.
- The number of neurons in the brain of human being is about 10^{11} neurons. And the average neuron has around 1,000 synapses. So in the human brain 100trillion (10^{14}) synapses are interconnected.
- In the 1940's, several researchers who studied biological neural network began to investigate how to emulate the mechanisms of the neural network. Perceptron is one of the early artificial neural network models.
- From here on, the term 'neural network' refers to 'artificial' neural network.



History of Neural Network

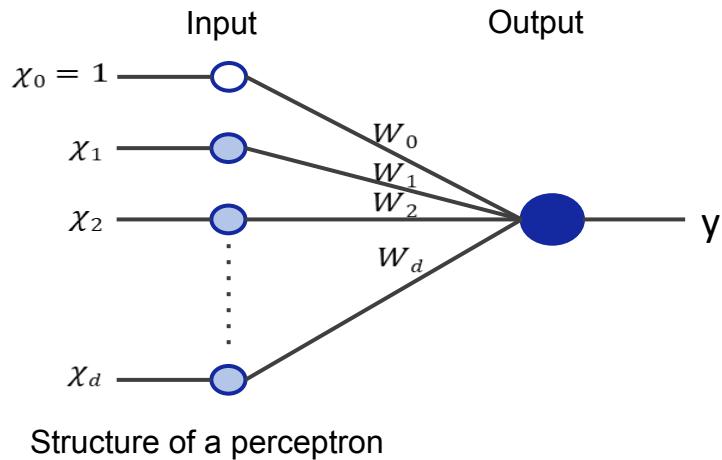


Frank Rosenblatt
(1928~1971)

- In 1943, McCulloch and Pitts first delineated the working mechanisms of neurons.
- Rosenblatt's perceptron is based on the McCulloch Pitts' neuron model.
- In 1958, Rosenblatt first proposed perceptron.
- In 1969, Minsky and Papert mathematically proved the limitations of the perceptron in their book 『Perceptrons』. It was found that a perceptron could not solve the XOR problem because it was only a linear classifier.

Perceptron

The Structure of a Perceptron



- Comprehend the structure of a perceptron by solving the OR function of the truth table.

Perceptron

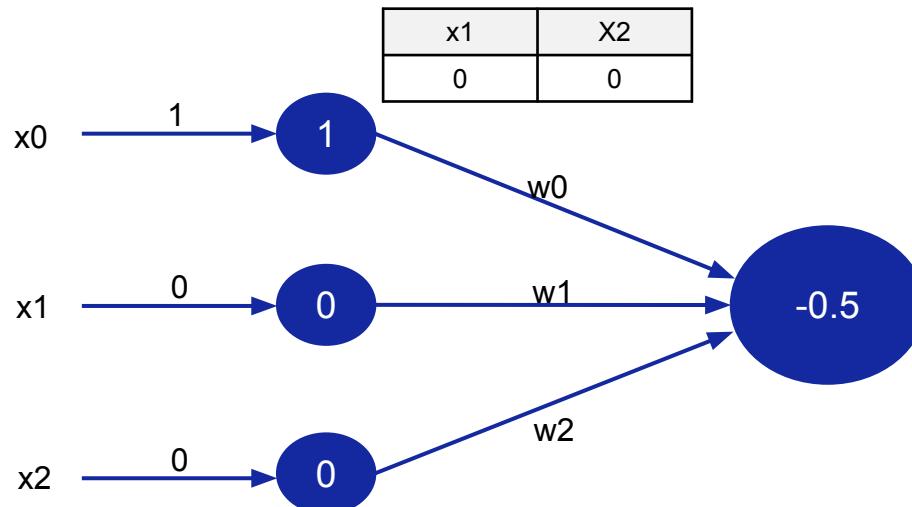
- First, you need to understand the OR operation of the truth table. The results of the OR operation are below.

X1	X2	OR operation
False	False	F
True	False	T
False	True	T
True	True	T

- What are the weight values to solve the OR operation problem?

Perceptron

- Suppose that we know the 'w' values to solve a problem. w_0 is -0.5, w_1 is 1, w_2 is 1.
(w : a weight vector)
- Execute Excel function after substituting x_1 with 0, and x_2 with 0.

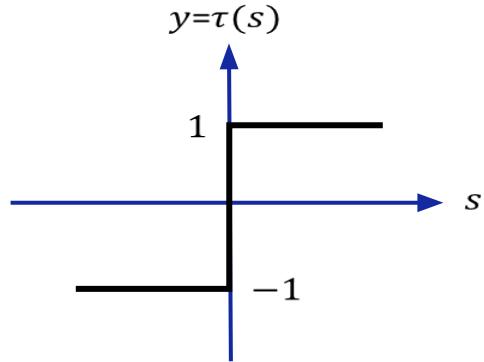


w_0	w_1	w_2
-0.5	1	1

$x_0 * w_0$	-0.5
+	
$x_0 * w_1$	0
+	
$x_0 * w_2$	0
Sum	-0.5

Perceptron

- Put the value of the sum into a function.
- Here, that function is the step function.



Use threshold function as an activation function $\tau(s)$

Perceptron

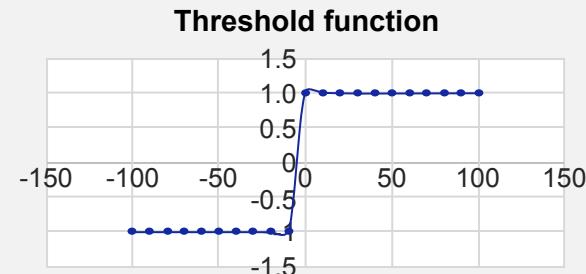
- A threshold function can be defined as follows, and we can create a graph by putting random values in the Excel table.

A threshold can be defined as follows,

$$y = \tau(s)$$

$$\text{here } s = w_0 + \sum_{i=1}^d w_i x_i, \quad \tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$$

-100	-1
-90	-1
-80	-1
-70	-1
-60	-1
-50	-1
-40	-1
-30	-1
-20	-1
-10	-1
0	1
10	1
20	1
30	1
40	1
50	1
60	1
70	1
80	1
90	1
100	1



Perceptron

- Put the result from the previous slide, -0.5 as s, and run it through the threshold function, the return is -1.
- Let's learn about the truth table.

A truth table displays truth or false for all results of the propositions or the combination of their [Boolean functions](#).

For example, in case of the [conjunction](#) of two statements P and Q, $P \wedge Q$, the truth table can be constructed as below.

In addition, true-false is also notated as T·F or 1·0.

Proposition P	Proposition Q	
True	True	True
True	False	False
False	True	False
False	False	False

Perceptron

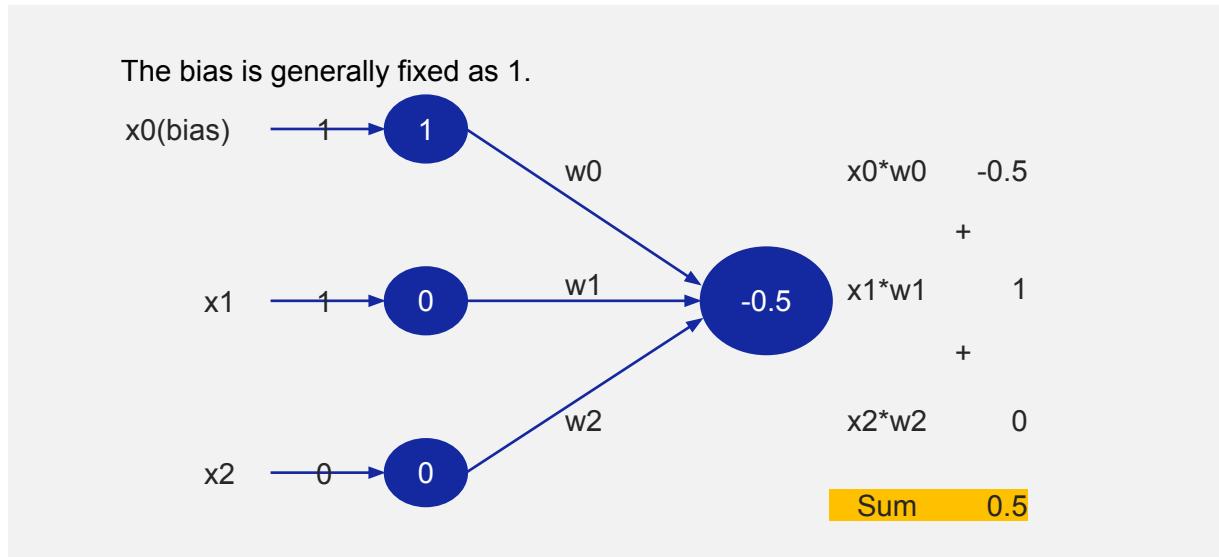
- A truth table can be expressed with 1 and 0 as follows.

x1	x2
0	0
1	0
0	1
1	1

- The figure from Slide 9 put 0 for the first variable x1, and 0 for x2 as well.
- x0 is called a bias, and it initializes as 1.

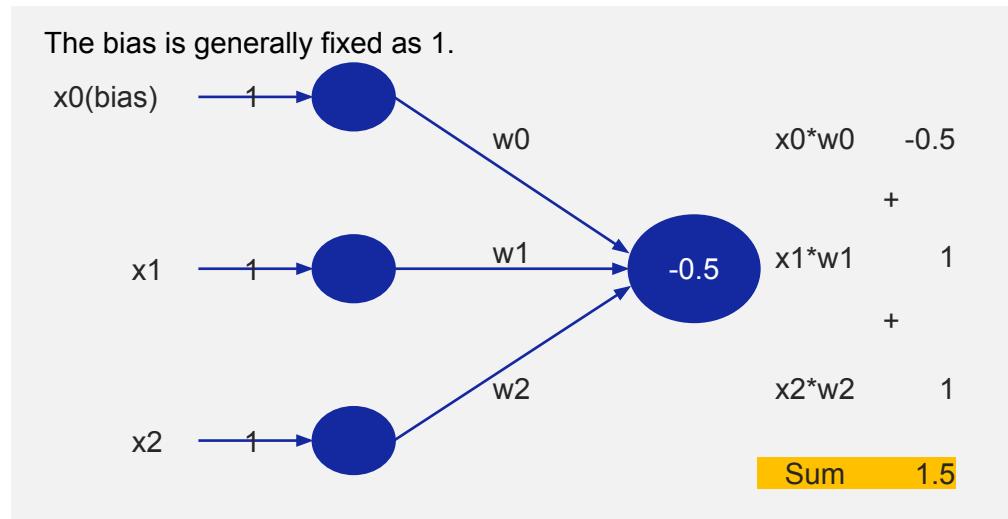
Perceptron

- Substitute x_1 with 1, and x_2 with 0.



Perceptron

- The sum is 0.5. Run this through the threshold function, and the result is 1.
- Likewise, if we process all values in the truth table through the perceptron, the results are as follows.

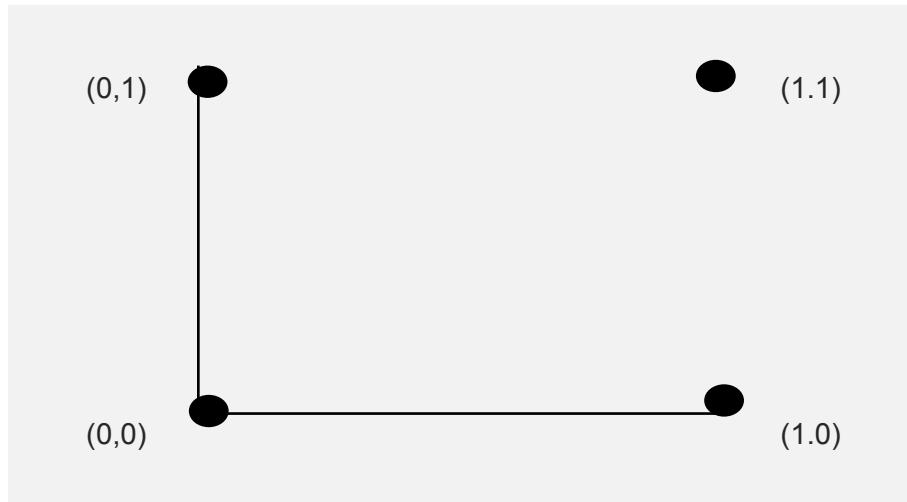


- 4 results can be displayed in a table.

x1	x2		
0	0	-0.5	-1
1	0	0.5	1
0	1	0.5	1
1	1	1.5	1

Perceptron

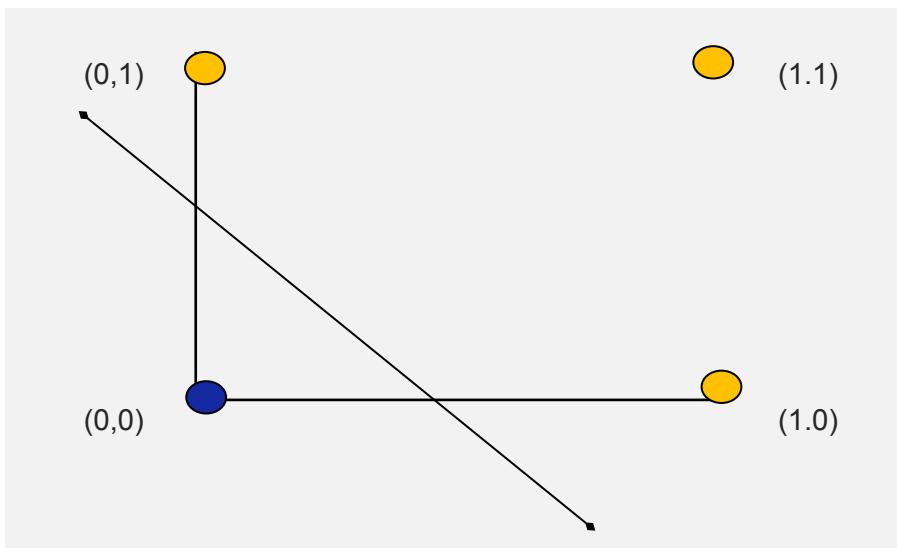
- These results can be shown in the coordinate plane as follows.
- The results can be grouped into that of point (0,0) or the rest.



x1	x2		
0	0	-0.5	-1
1	0	0.5	1
0	1	0.5	1
1	1	1.5	1

Perceptron

- Such classification can be presented as a geometrical figure as follows.
- If you recall the basic notion of machine learning through linear regression, machine learning is a process that builds regression equation with the predicted values of the slope and y-intercept. The figure below shows how linear regression forms a line that separate the two groups.



x1	x2		
0	0	-0.5	-1
1	0	0.5	1
0	1	0.5	1
1	1	1.5	1

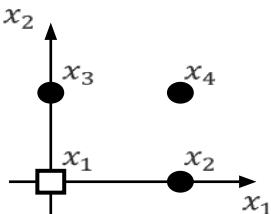
- This table displays the OR operation of a truth table.

OR operation

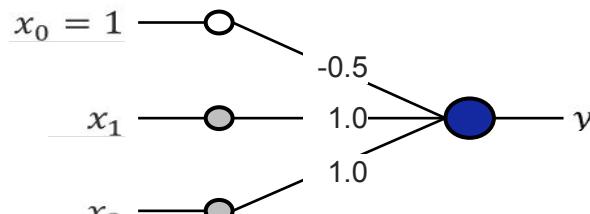
x1	x1	x2	Value before running threshold function	Value after running threshold function
0	0	0	=A\$2*\$C\$7+B2*\$C\$8+C2*\$C\$9	=IF(D2>=0,1,-1)
	0	1	=A\$2*\$C\$7+B3*\$C\$8+C3*\$C\$9	=IF(D3>=0,1,-1)
1	0	0	=A\$2*\$C\$7+B4*\$C\$8+C4*\$C\$9	=IF(D4>=0,1,-1)
1	1	1	=A\$2*\$C\$7+B8*\$C\$8+C5*\$C\$9	=IF(D5>=0,1,-1)

w1	-0.5
w2	1
w3	1

$$x_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, y_1 = -1, x_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, y_2 = 1, x_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, y_3 = 1, x_4 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, y_4 = 1$$



(a) Train set



(b) Perceptron

Example of a perceptron's operation using the OR logic gate

OR operation

Let's input four samples to the perceptron and check the results.

$$x_1: s = -0.5 + 0 * 1.0 + 0 * 1.0 = -0.5, \quad \tau(-0.5) = -1$$

$$x_2: s = -0.5 + 1 * 1.0 + 0 * 1.0 = 0.5, \quad \tau(0.5) = 1$$

$$x_3: s = -0.5 + 0 * 1.0 + 1 * 1.0 = 0.5, \quad \tau(0.5) = 1$$

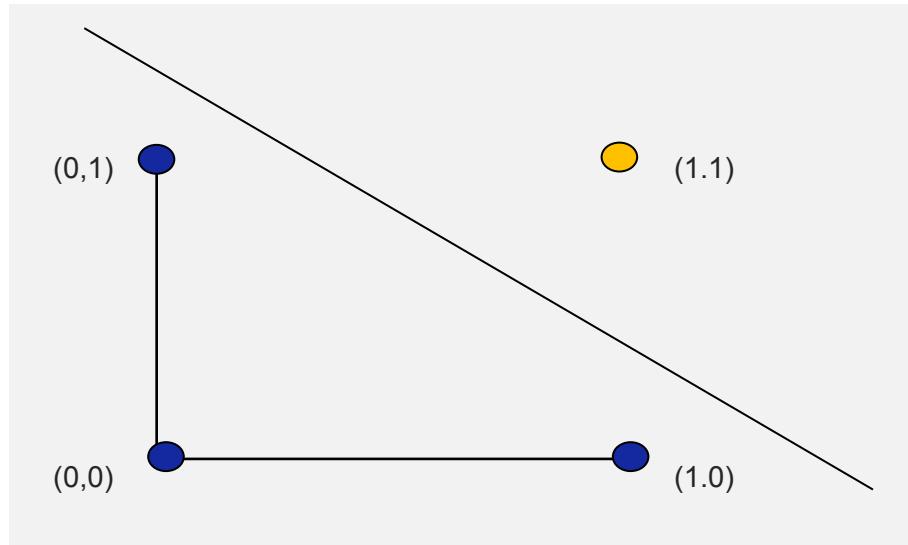
$$x_4: s = -0.5 + 1 * 1.0 + 1 * 1.0 = 1.5, \quad \tau(1.5) = 1$$

- As you've seen in the previous slide, the perceptron delivered correct results for all four samples.
- It can be said that this perceptron classifies the train set with 100 % performance.

AND operation

- To recap, you can solve the OR problem by perceptron with the appropriate w values.
- Now, the AND operation. The truth table is on the left, and the geometrical solution is the line that separates the units into two groups.

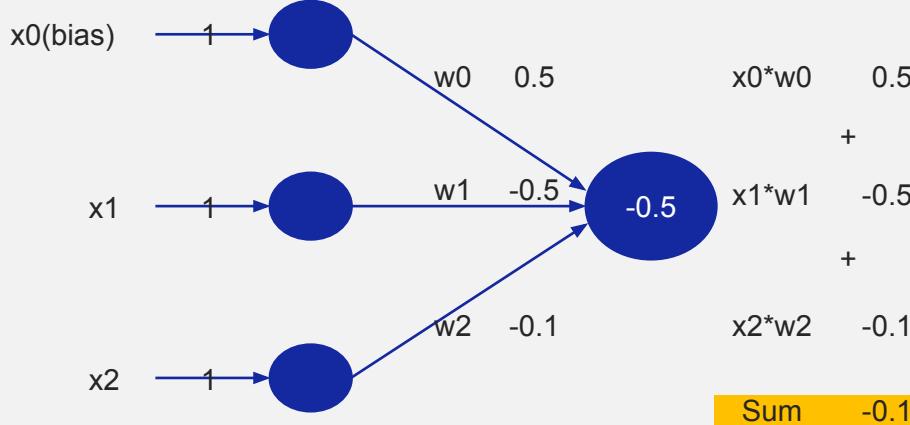
x1	x2	AND operation
0	0	F
1	0	F
0	1	F
1	1	T



AND operation

- Find the values of w_0 , w_1 , w_2 that return the result from the previous slide.
- First, let's apply a random value, and then gradually change the values of w_0 , w_1 , and w_2 to find the solution.

The bias is generally fixed as 1.



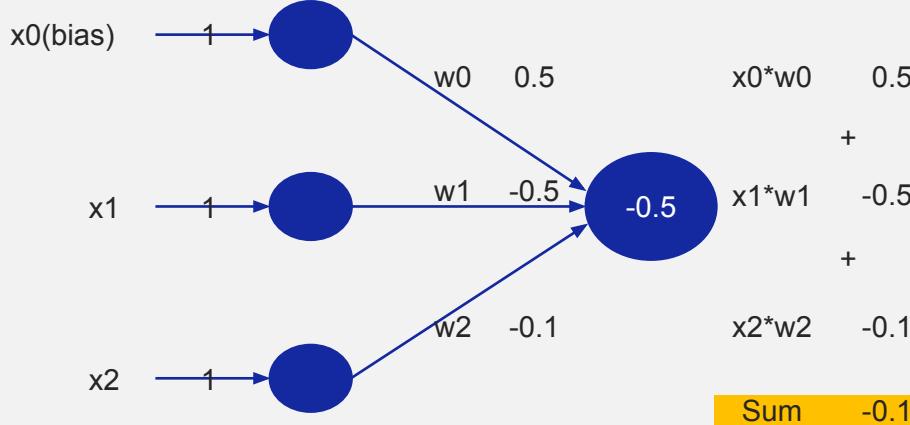
- 4 results can be displayed in a table.

x1	x2		
0	0	0.5	1
1	0	0	1
0	1	0.4	1
1	1	-0.1	-1

- AND operation of a truth table.

AND operation

The bias is generally fixed as 1.



- 4 results can be displayed in a table.

x1	x2		
0	0	0.5	1
1	0	0	1
0	1	0.4	1
1	1	-0.1	-1

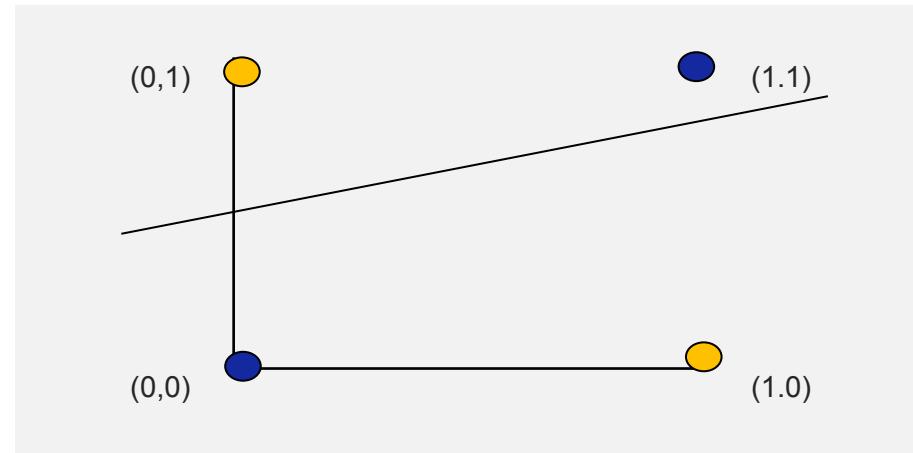
- AND operation of a truth table.

- By substituting w0 with 0.5, w1 with -0.5, w2 with -0.1, the equation solves the AND operation.
- You found a structure that with appropriate values for the w vector, solves a certain problem.
- Although the perceptron solved the OR and AND problem, it could not solve the XOR. Let's find out why.

XOR Operation

- XOR operation in a truth table.
- In XOR, the result is true if two propositions are the opposite.

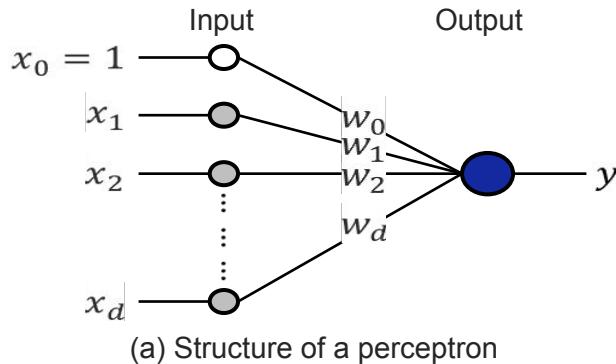
x1	x2	XOR Operation
0	0	F
1	0	T
0	1	T
1	1	F



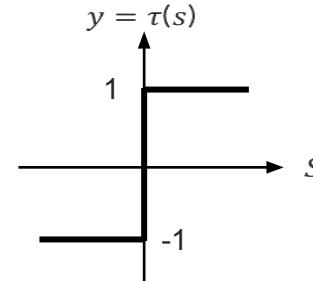
- Since the perceptron is a linear classifier, you cannot find a line that separate the blue dots and the red dots.
- A multilayer perceptron solved the problem.

Summary of Perceptron's Structure

- It has an input layer and an output layer.
- The input layer does not operate, so the perceptron is considered a single layer structure.
- The i th node of the input layer takes x_i from the feature vector $x = (x_1, x_2, \dots, x_d)^T$.
- The bias node always takes 1 as input.
- The output layer has a single node.
- The connection of the i th node of the input later and the output layer has weight w_i .



(a) Structure of a perceptron

(b) Use threshold function as an activation function $\tau(s)$

Mechanism of a perceptron

Unit 1.

Basics of Neural Network

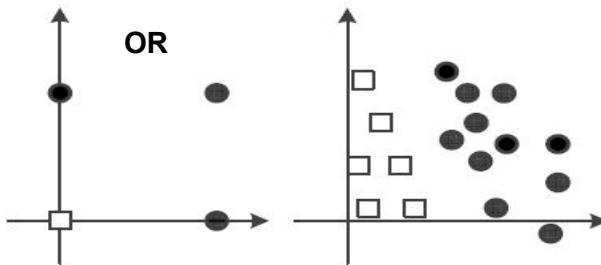
- 1.1. Understanding Perceptron with Excel
- 1.2. Understanding Multilayer Perceptron with Excel
- 1.3. From Multilayer Perceptron to Deep Learning
- 1.4. Error Backpropagation and Gradient Descent to Reduce Errors
- 1.5. Activation Function
- 1.6. Deep Learning
- 1.7. Problems with Deep Learning and the Solutions

Understand Multilayer Perceptron with Excel

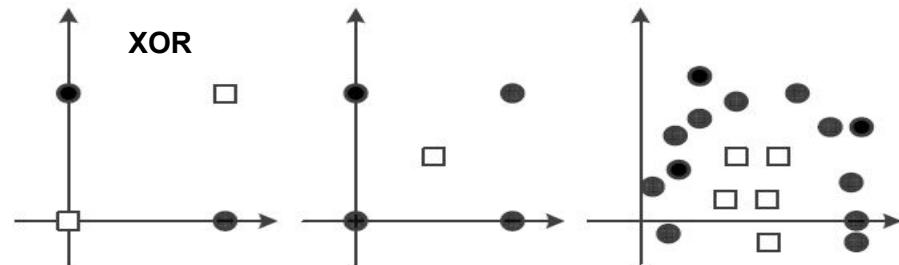
Perceptron cannot compute the XOR function.

- Linear classification is impossible, and certain amount of errors occur.
- 75% accuracy is the limit in XOR function.

Linearly separable case and linearly inseparable case



(a) Linearly separable



(b) Linearly inseparable

Multilayer Perceptron

- In XOR, the inputs are not linearly separable.
- It could not solve a simple problem like XOR.
 - This was claimed by Marvin Minsky, an MIT professor who was the pioneer of the AI field, in his book <Perceptrons>, published in 1969.
 - People realized that emulating the diagram of ‘neuron □ neural network □ intelligence’ through the diagram of ‘perceptron □ artificial neural network □ artificial intelligence’ was not an easy task.

After this publication, research on AI took a downturn.

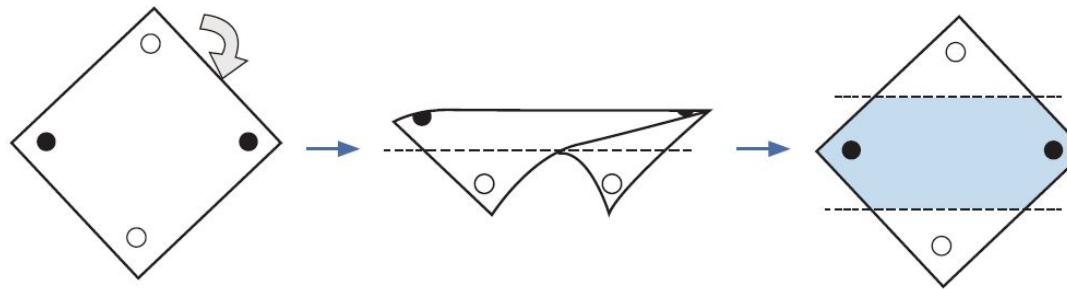
After 10 years, this problem was solved by ‘multilayer’ perceptron.

Multilayer Perceptron

- What are the ways you can separate the inputs that are not linearly separable?
- First, the inputs can be separated in a non-linear form.
- Second, a two-dimensional figure on a plane, could be modified into a three-dimensional figure to enable separation.

Solution for the XOR problem

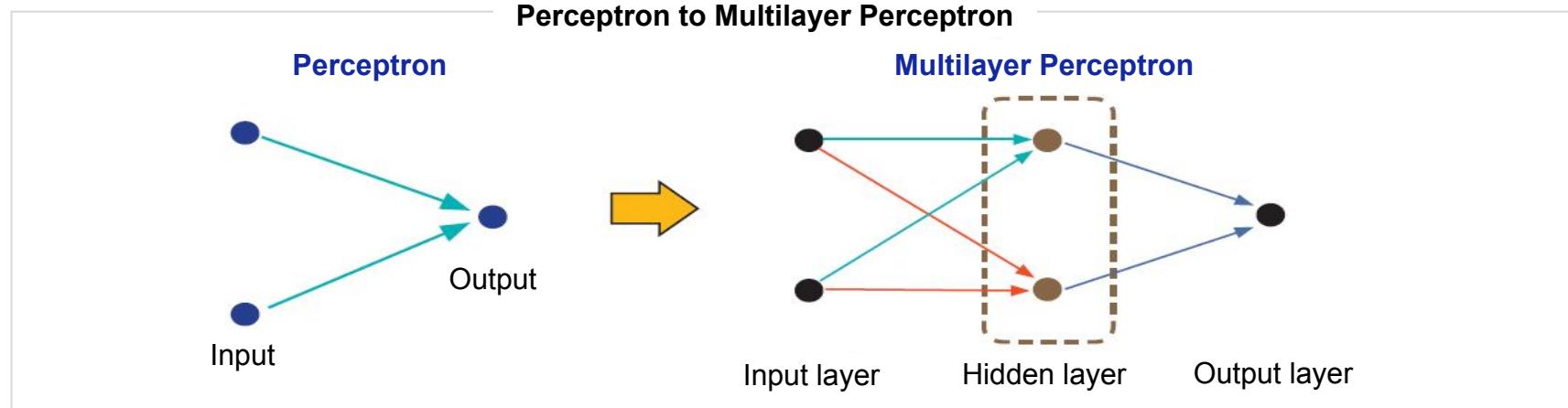
- Artificial Intelligence scientists had to overcome the XOR problem to develop an artificial neural network.
- An innovative idea provided hints for the solution.



- You had to bend the plane!

Solution for the XOR problem

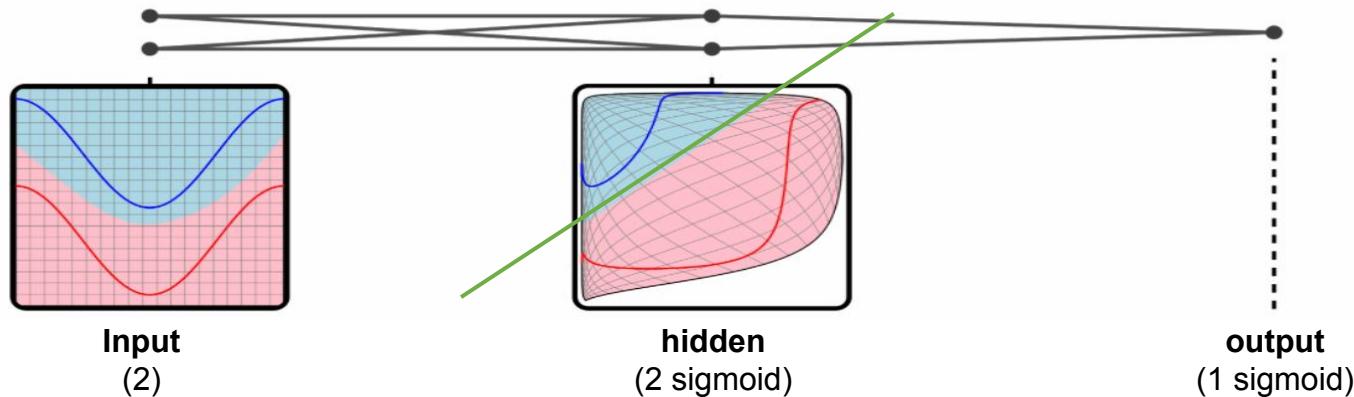
- ▶ Modify the coordinate plane itself.
- ▶ To solve the XOR, you need to run two perceptrons simultaneously.
- ▶ To make this possible, a hidden layer is necessary.



Solution for the XOR problem

- If you divide the plane into sections of blue and red with regards to the inputs, no line can separate the two. However, if a hidden layer warps the plane, the curve line separating the two sections becomes a straight line.

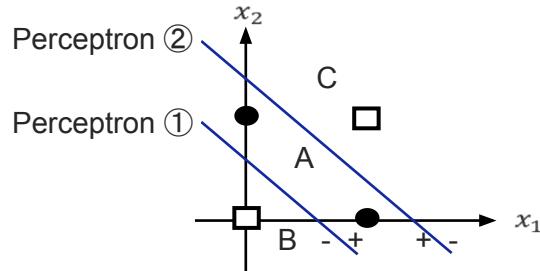
Perceptron to Multilayer Perceptron



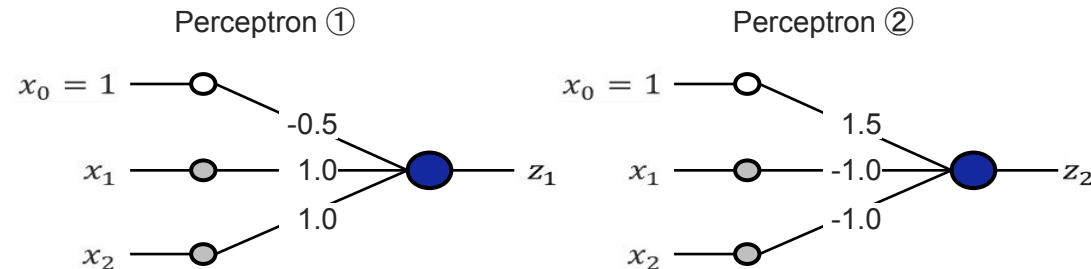
Solution for the XOR problem

Solving the XOR with 2 perceptrons

- If perceptron ① and perceptron ② is both +1, the unit is ● . If not, □ .



(a) Separating the plane
by two perceptrons



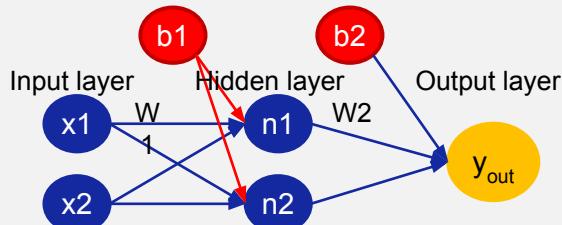
(b) Two perceptrons

Solution for the XOR problem

Solution for the XOR problem

- Solve the XOR problem by applying multilayer perceptron in Excel.

Solving XOR



Sigmoid function

$$=1/(1+\exp(-x))$$

x_1	x_2
0	0
0	1
1	0
1	1

W_1
-2
2
-2
2

W_2
1
1
1
3

n_1 After running sigmoid function		
-1	0.268941	0
1	0.731059	1
1	0.731059	1
3	0.952574	1

n_2
0.268941
0.731059
0.731059
0.952574

$output$
0
1
1
0

Desired value
0
1
1
0

b_1
3
-1

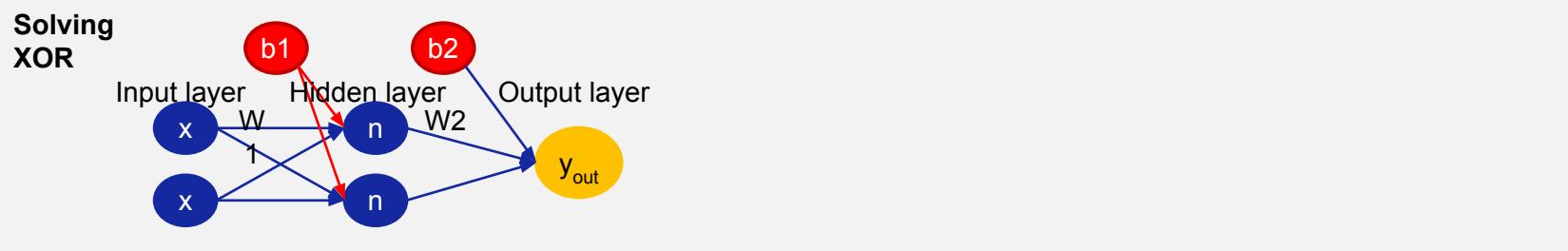
b_2
-1

x_1	x_2	or	and	nand
0	0	0	0	1
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

or	nand
0	1
1	1
1	1
1	0

Solution for the XOR problem

- Compute the XOR by applying multilayer perceptron in Excel.



x1	x2
0	0
0	1
1	0
1	1

W1
-2
2
-2
2

W2
1
1

n1 After running sigmoid function				
=C12*\$F\$12+D12*\$F\$14+\$F\$18	=1/(1+EXP(-K12))	1		
=C13*\$F\$12+D13*\$F\$14+\$F\$18	=1/(1+EXP(-K13))	1		
=C14*\$F\$12+D14*\$F\$14+\$F\$18	=1/(1+EXP(-K14))	1		
=C15*\$F\$12+D15*\$F\$14+\$F\$18	=1/(1+EXP(-K15))	0		

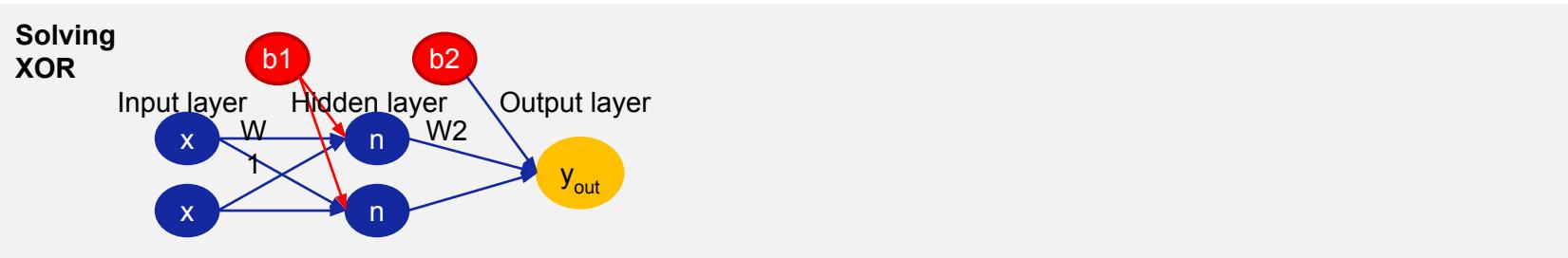
x1	x2	or	and	nand
0	0	0	0	1
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

n2		
=C12*\$F\$13+D12*\$F\$15+\$F\$19	=1/(1+EXP(-O12))	0
=C13*\$F\$13+D13*\$F\$15+\$F\$19	=1/(1+EXP(-O13))	1
=C14*\$F\$13+D14*\$F\$15+\$F\$19	=1/(1+EXP(-O14))	1
=C15*\$F\$13+D15*\$F\$15+\$F\$19	=1/(1+EXP(-O15))	1

or	nand
0	1
1	1
1	1
1	0

Solution for the XOR problem

- Compute the XOR by applying multilayer perceptron in Excel.



x_1	x_2
0	0
0	1
1	0
1	1

W_1
-2
2
-2
2

W_2
1
1

n1 After running sigmoid function		
$=C12*\$F\$12+D12*\$F\$14+\$F\18	$=1/(1+EXP(-K12))$	1
$=C13*\$F\$12+D13*\$F\$14+\$F\18	$=1/(1+EXP(-K13))$	1
$=C14*\$F\$12+D14*\$F\$14+\$F\18	$=1/(1+EXP(-K14))$	1
$=C15*\$F\$12+D15*\$F\$14+\$F\18	$=1/(1+EXP(-K15))$	0

n_2		
$=C12*\$F\$13+D12*\$F\$15+\$F\19	$=1/(1+EXP(-O12))$	0
$=C13*\$F\$13+D13*\$F\$15+\$F\19	$=1/(1+EXP(-O13))$	1
$=C14*\$F\$13+D14*\$F\$15+\$F\19	$=1/(1+EXP(-O14))$	1
$=C15*\$F\$13+D15*\$F\$15+\$F\19	$=1/(1+EXP(-O15))$	1

b_1
3
-1

b_2
-1

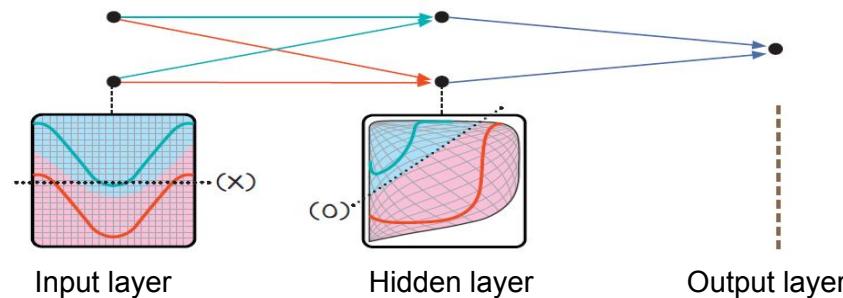
x_1	x_2	or	and	nand
0	0	0	0	1
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

or	$nand$	
0	1	0
1	1	1
1	1	1
1	0	0

Solution for the XOR problem

- The hidden layer warps the coordinate plane.

x1	x2	or	and	nand	or	nand	xor
0	0	0	0	1	0	1	0
0	1	1	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	1	1	0	1	0	0



- Hidden layer warps the plane. (<https://goo.gl/8qEGHD> for more reference)

『Perceptrons』 by Minsky

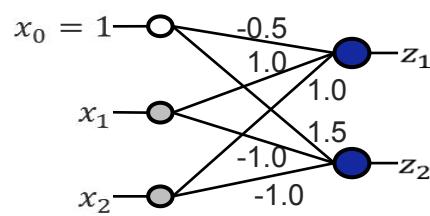
- Minsky pointed out the limitations of the perceptron and suggested a multilayer structure as a solution. The level of technology at the time was not advanced enough to materialize his idea.
- In 1974, Paul Werbos proposed backpropagation algorithm in his doctoral thesis.
- In 1986, Rumelhart established the multilayer theory in his book 『Parallel Distributed Processing』 and revived interest on the neural network.

Central Idea to the Multilayer Perceptron

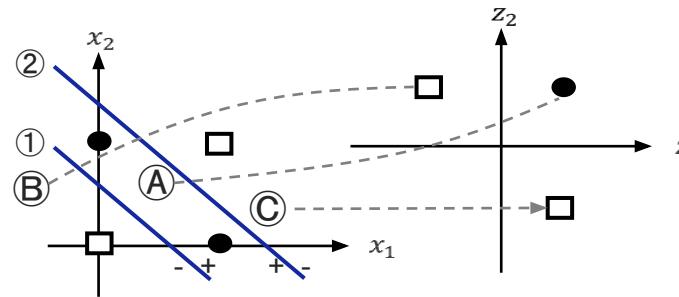
- Insert a hidden layer. It transforms a given feature space into a new one much favorable for classification.
- Use a sigmoid activation function. Perceptron used a step function as an activation function. This function is categorized as a hard decision decoder. However, a multilayer perceptron utilizes sigmoid function, which is a soft decision decoder, as an activation function. In soft-decision decoding, the output is continuous, and indicates confidence. This enables a more flexible decision making.
- Use backpropagation algorithm. Multilayer perceptron consists of many layers in consecutive order. Thus, backpropagation algorithm, which progresses in reverse order, and computes gradients and renews weights in each layer, is effective.

Central Idea to the Multilayer Perceptron

- ▶ Parallel combine two perceptrons
 - Transform a space $x = (x_1, x_2)^T$ into a new feature space $z = (z_1, z_2)^T$
 - The new feature space z is linear separable.



(a) Parallel combine two perceptrons

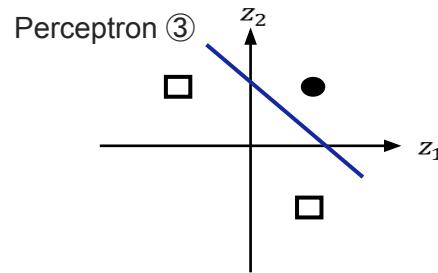
(b) Transform a feature space x into a new feature space z

Transformation of a feature space

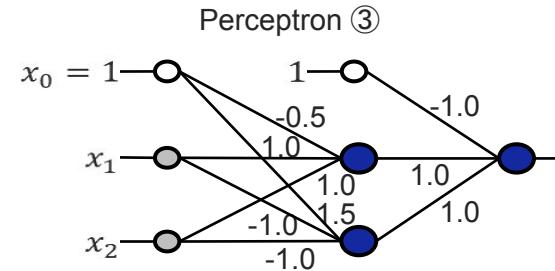
- Equivalent to a human-being manually executing feature learning

Central Idea to the Multilayer Perceptron

- ▶ If you serial combine a single perceptron
 - In feature space z , if you serial combine perceptron 3 which performs linear classification, it becomes a **multilayer perceptron**.



(a) Separation in a new feature space

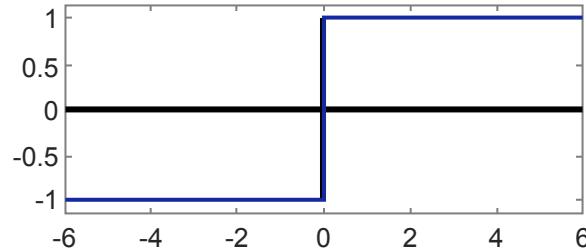


(b) Multilayer perceptron that combines three perceptrons

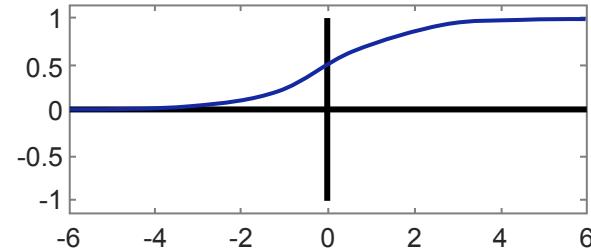
Multilayer Perceptron

Central Idea to the Multilayer Perceptron

- A threshold function is a hard-decision decoder (transforms an area into a dot). Other activation functions are soft-decision decoders (transforms an area into an area).



(a) Threshold function



(b) Logistic sigmoid

Activation function used in neural network

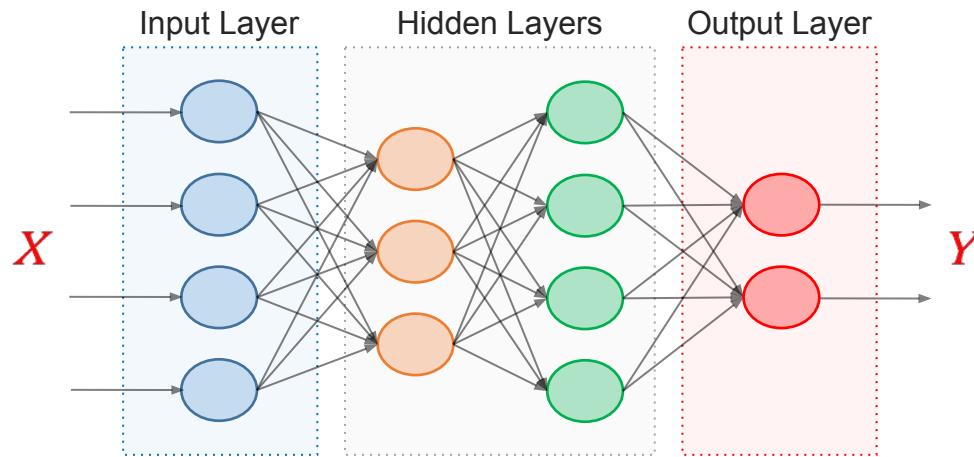
Unit 1.

Basics of Neural Network

- 1.1. Understanding Perceptron with Excel
- 1.2. Understanding Multilayer Perceptron with Excel
- 1.3. From Multilayer Perceptron to Deep Learning
- 1.4. Error Backpropagation and Gradient Descent to Reduce Errors
- 1.5. Activation Function
- 1.6. Deep Learning
- 1.7. Problems with Deep Learning and the Solutions

Artificial Neural Network (ANN)

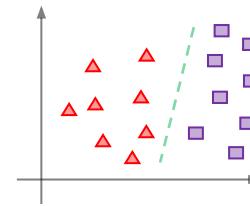
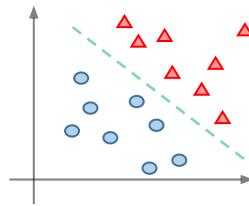
| About the artificial neural network (ANN):



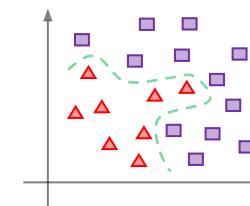
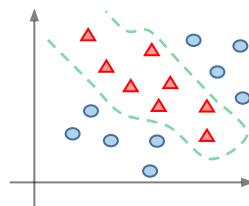
- Mimics the neural connections of a biological brain.
- There can be several hidden layers.

I Why Artificial Neural Network (ANN)?

- For a logistic regression (even for the multi-class variant), the decision boundaries are linear.



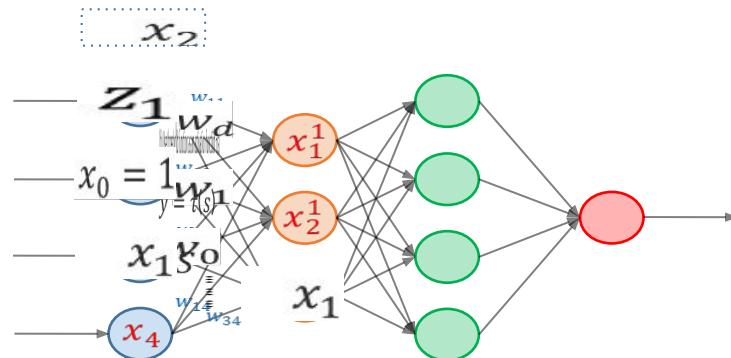
- How about the cases that require non-linear decision boundaries? \Rightarrow ANN with hidden layers!



ANN training: forward propagation.

- When the values of X_i are given, apply the weights and propagate the signal forward to the next layer.

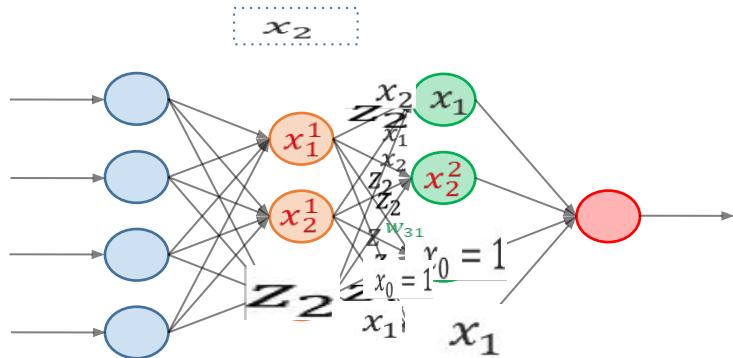
Example



$$\begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix} = Activation \left(\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right)$$

- ANN training: forward propagation.
2) Propagate forward to the next layer.

Example



Unit 1.

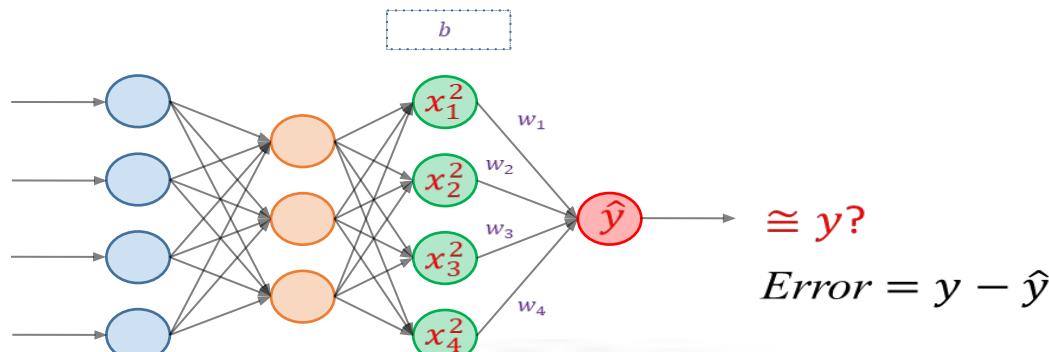
Basics of Neural Network

- 1.1. Understanding Perceptron with Excel
- 1.2. Understanding Multilayer Perceptron with Excel
- 1.3. From Multilayer Perceptron to Deep Learning
- 1.4. Error Backpropagation and Gradient Descent to Reduce Errors
- 1.5. Activation Function
- 1.6. Deep Learning
- 1.7. Problems with Deep Learning and the Solutions

ANN training: forward propagation

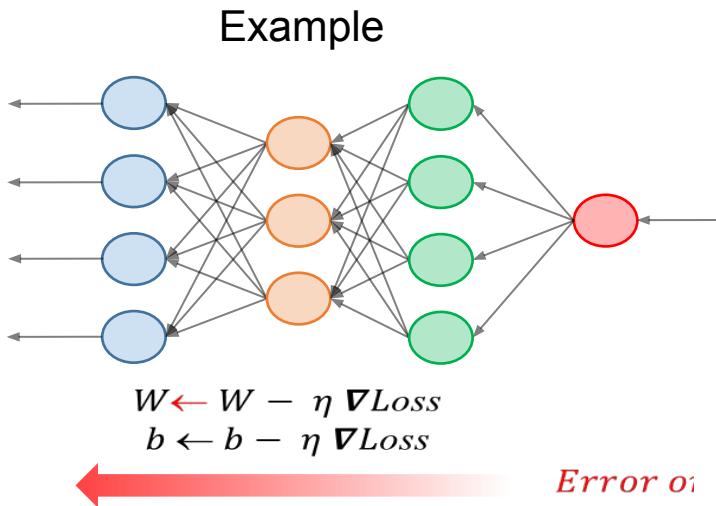
- Propagate all the way; the difference between the estimated value \hat{y} and the true value y is the error.

Example



ANN training: backward propagation

- Propagate the error backward and update the parameters by gradient descent algorithm. Repeat from 1).



Unit 1.

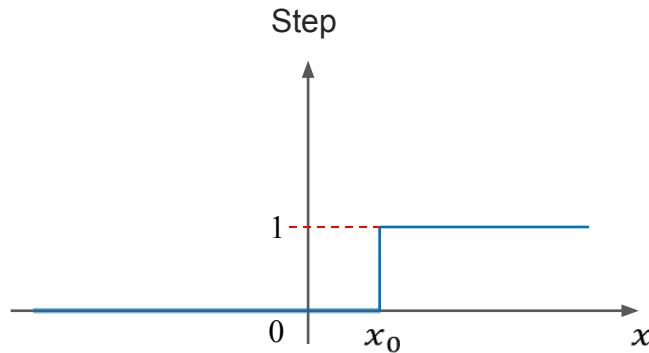
Basics of Neural Network

- 1.1. Understanding Perceptron with Excel
- 1.2. Understanding Multilayer Perceptron with Excel
- 1.3. From Multilayer Perceptron to Deep Learning
- 1.4. Error Backpropagation and Gradient Descent to Reduce Errors
- 1.5. Activation Function
- 1.6. Deep Learning
- 1.7. Problems with Deep Learning and the Solutions

Activation Function

Activation function: Linear threshold or Step.

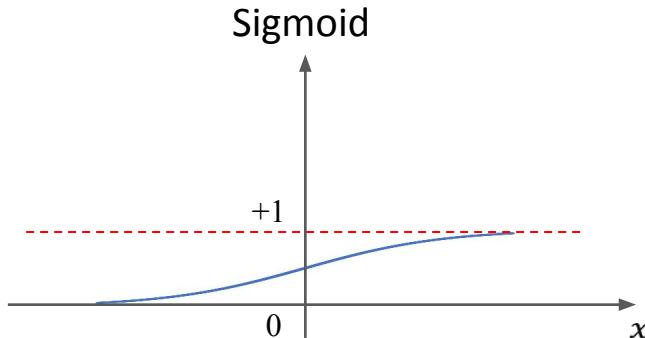
- ▶ $\text{Step}(x) = \theta(x - x_0)$
- ▶ An activation function that was frequently used in the early days of neural network is the step function.



Activation function: Sigmoid $\sigma(x)$.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- The Sigmoid is another widely used activation function. Note that the output value ranges between 0 and 1.

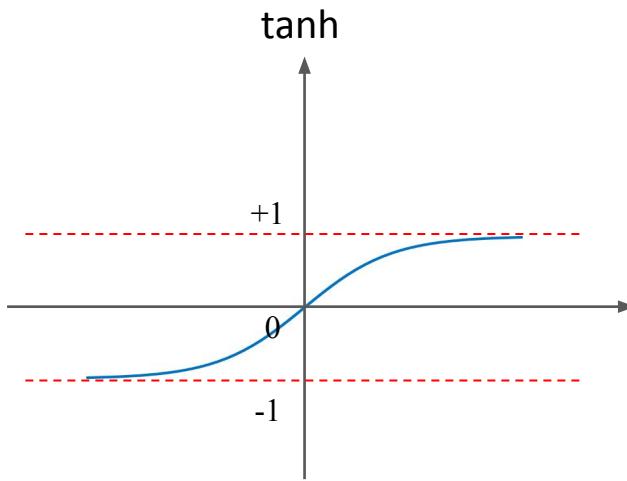


- When we have more than two classes, it can be generalized to the "Softmax".

Activation function: Hyperbolic tangent or tanh.

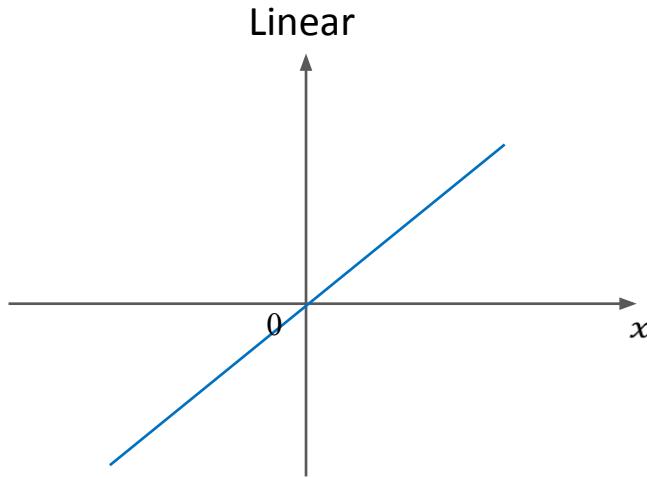
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- The hyperbolic tangent (tanh) looks similar to the Sigmoid. However, the output value ranges between -1 and +1.



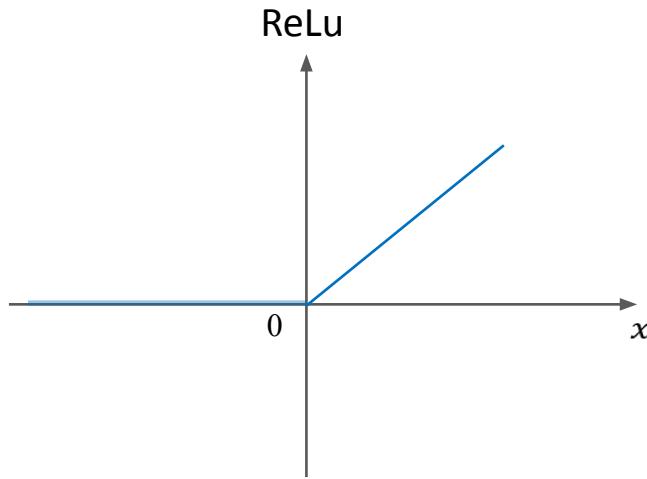
| Activation function: Linear.

- ▶ $\text{Linear}(x) = x$
- ▶ The linear activation's output is the same as the input. The linear activation function is used in the regression.



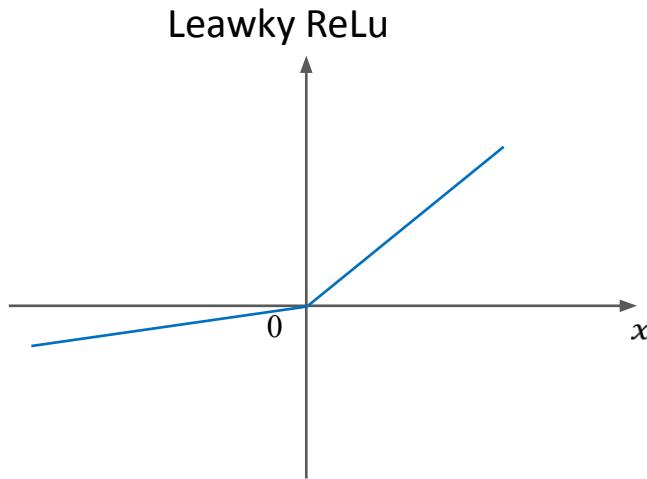
Activation function: ReLu (Rectifier Linear Unit).

- ▶ $\text{ReLU}(x) = \max(0, x)$
- ▶ ReLu is another commonly used activation function. It can be expressed in a simple functional form.



| Activation function: Leaky ReLu.

- ▶ $\text{Leaky ReLu}(x) = \max(\alpha x, x)$ where $\alpha \in (0,1)$
- ▶ Leaky ReLu is a slightly “tweaked” version of ReLu.



Activation function:

- This is a summary of some of the most commonly used activation functions.
- Softmax is often used at the output layer of neural networks that do classification.

Name	Formula	TensorFlow
Step		
Sigmoid		<code>tf.math.sigmoid(x)</code>
Tanh		<code>tf.math.tanh(x)</code>
ReLU		<code>tf.nn.relu(x)</code>
Softmax		<code>tf.nn.softmax(x)</code>

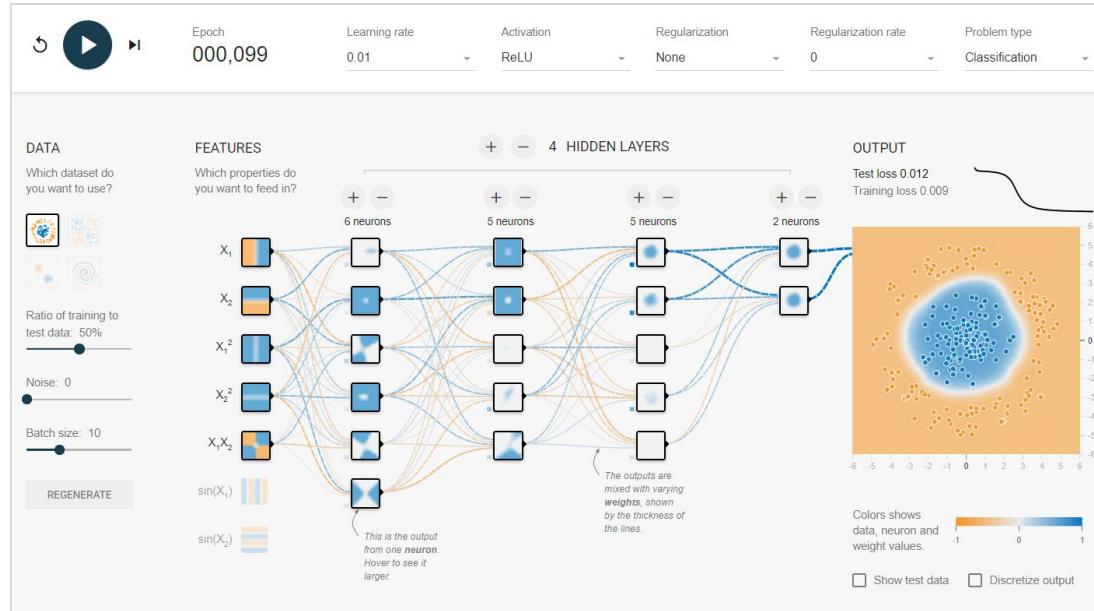
Unit 1.

Basics of Neural Network

- 1.1. Understanding Perceptron with Excel
- 1.2. Understanding Multilayer Perceptron with Excel
- 1.3. From Multilayer Perceptron to Deep Learning
- 1.4. Error Backpropagation and Gradient Descent to Reduce Errors
- 1.5. Activation Function
- 1.6. Deep Learning
- 1.7. Problems with Deep Learning and the Solutions

Artificial neural network: a recommended site

- Access the website below to check artificial neural network.



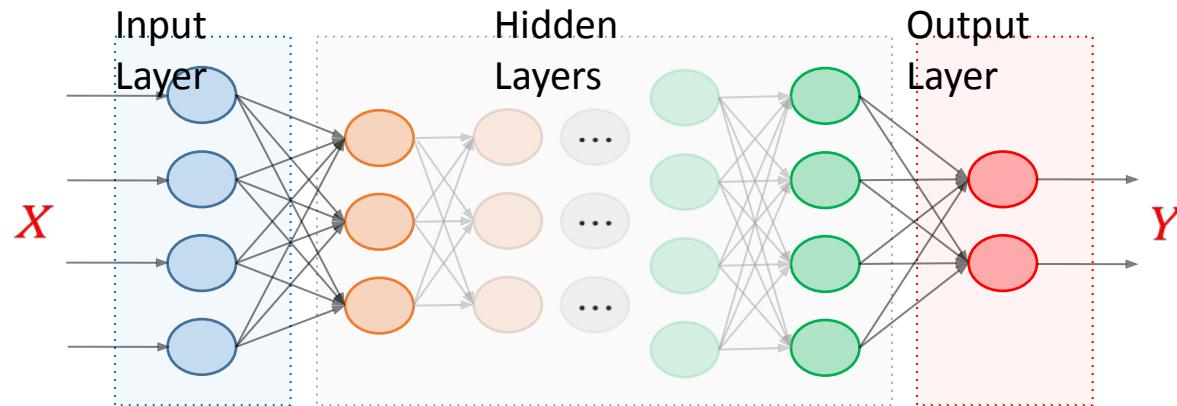
Website: <https://playground.tensorflow.org>

Artificial neural network: a recommended site

- The GUI of the website (<https://playground.tensorflow.org>) is intuitive.
- Select data from the left side panel in the website.
- Adjust the features, layers and nodes.
- From the top drop down menus, customize the learning rate, activation function, regularization, etc.
- Press the Play button to start training ⇒ **training** loss decreases as the training progresses.
- The thickness of an edge line represents the weight that can also be changed manually.

About the Deep Neural Network (DNN):

- There may be several hidden layers.
- With more hidden layers, we have a higher degree of abstraction.



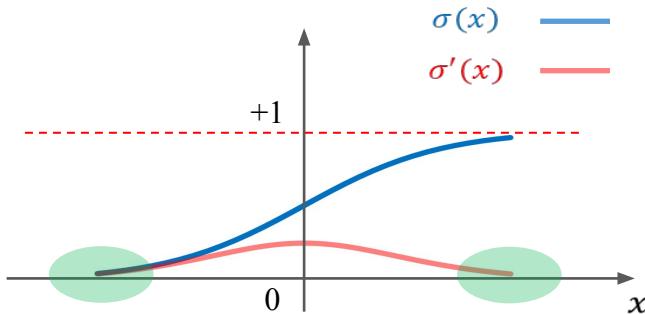
Unit 1.

Basics of Neural Network

- 1.1. Understanding Perceptron with Excel
- 1.2. Understanding Multilayer Perceptron with Excel
- 1.3. From Multilayer Perceptron to Deep Learning
- 1.4. Error Backpropagation and Gradient Descent to Reduce Errors
- 1.5. Activation Function
- 1.6. Deep Learning
- 1.7. Problems with Deep Learning and the Solutions

I Common problems with the DNN:

- As the ‘signal’ propagates through the layers, it can die out (vanish) or grow uncontrollably.
 - a) We should avoid using the Sigmoid $\sigma(x)$ as the activation function and use the ReLu instead.
 - The derivative $\sigma'(x)$ becomes very small as the absolute value of x or $|x|$ increases.
 - Small $\sigma'(x)$ may cause the so-called vanishing gradient problem.



I As the ‘signal’ propagates through the layers, it can die out (vanish) or grow uncontrollably.

▶ As the ‘signal’ propagates through the layers, it can die out (vanish) or grow uncontrollably.

b) We should initialize the weights such that the variance of the internal nodes is roughly constant.

→ Initializing as a constant such as 1 is not optimal.

→ Better to randomly initialize the weights with center at 0 and standard deviation $\sim \frac{1}{\sqrt{N_{nodes}}} .(*)$

Here, N_{nodes} = average number of nodes in the neighboring layers (before and after).

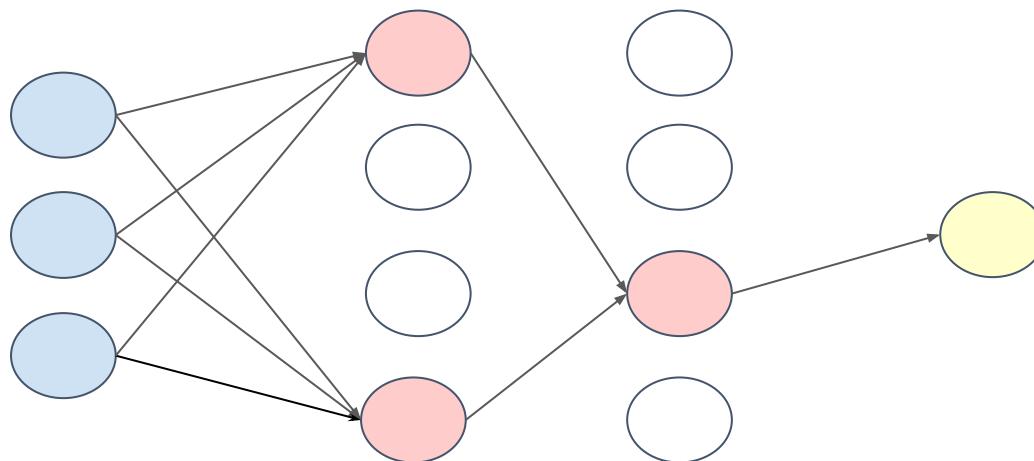
▶ (*) Refer to the research paper at <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>

Common problems with the DNN:

- Also, DNNs are prone to the overfitting. To alleviate this problem we can do:
 - a) Regularization of the weights: L1 or L2 regularization similar to the Lasso or Ridge.
 - b) Dropout: randomly exclude certain nodes during the training step to avoid over dependency.
 - c) Data augmentation: add noise, create new data by transformation, etc.

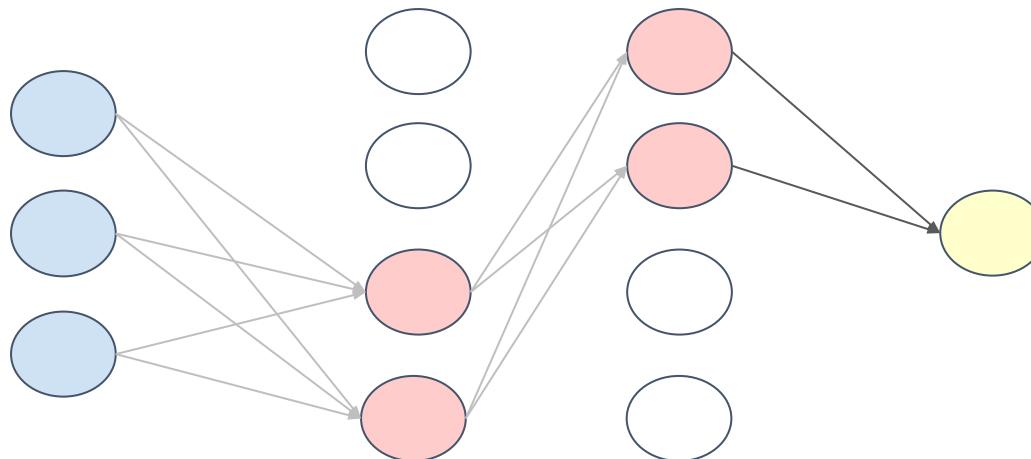
| Dropout:

- Randomly exclude certain nodes during the training step to avoid over dependency.



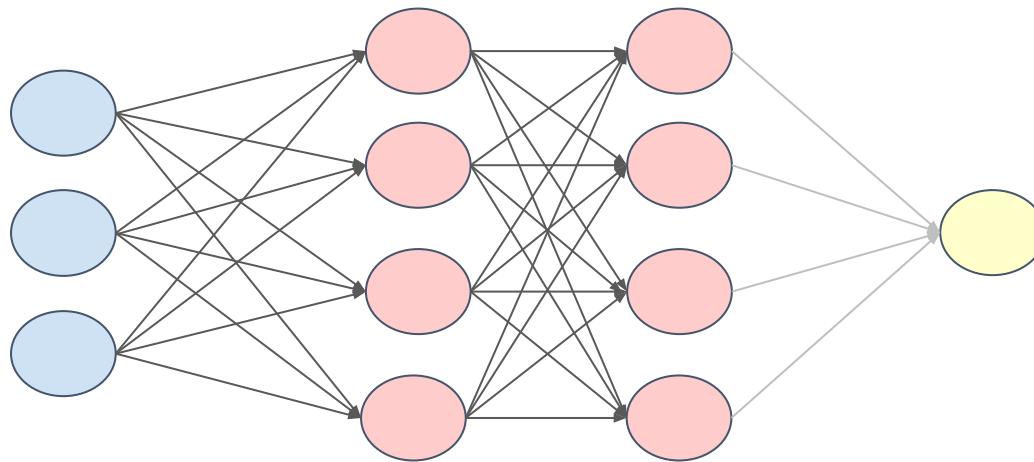
| Dropout:

- Randomly exclude certain nodes during the training step to avoid over dependency.



| Dropout:

- When predicting, use all the nodes and edges without excluding.



Unit 2.

TensorFlow Basics

- | 2.1. Introduction to TensorFlow
- | 2.2. TensorFlow dataset
- | 2.3. Machine Learning with TensorFlow

Introduction to TensorFlow

About the TensorFlow

- Tensorflow can utilize both CPU and GPU to improve the training performance of a machine learning model.
- It can produce maximum performance only with GPU
- Tensorflow officially supports CUDA-capable GPU.
- Support for OpenCL GPU is still on progress.
- It's likely that OpenCL will be officially supported soon.
- Tensorflow is a front-end interface that supports various programming languages.



About the TensorFlow

- Since Tensorflow's Python API is fully mature, it is popular to deep learning and machine learning technologists.
- Tensorflow has its official C ++ API as well.
- In addition, two new TensorFlow-based libraries, Tensorflow.js and Tensorflow Lite were introduced.
- APIs for Java, Haskell, Node.js or Go are not yet stable.
- Tensorflow developers and open source community are constantly working to resolve such issues.

I What is TensorFlow?

- Tensorflow generates a computational graph composed of a series of nodes.
- Each node expresses an operation that can denote more than 0 input or output.
- Tensor was created as a symbolic handle to refer to inputs and outputs of such operation.

I Why TensorFlow?

- Tensorflow immensely expedites the machine learning process.
 - Performance issue
 - Since the performance of computer processors consistently improved in the past years, it can train more complex and powerful machine learning systems.
 - It can enhance machine learning model's prediction accuracy.
 - Even the cheapest computer hardwares carry processors with a number of cores.
 - Numerous functions from Scikit-learn can disperse an operation into numbers of processes.
 - Generally, Python can only run one core due to GIL (Global Interpreter Lock).
 - Multiprocessing libraries can disperse an operation into multiple cores, but even the most advanced desktop PCs do not carry more than 8 or 16 cores.

Why TensorFlow?

- Even the simplest multilayer perceptron has a hundred units in a single hidden layer.
- You need about 80,000 weight parameters ($[784 \times 100 + 100] + [100 \times 10 + 10] = 79,510$) to train a model to classify even the simplest images.
 - MNIST is a relatively small image (28×28 pixel)
- The number of weight parameters will radically increase when processing high-pixel images or adding hidden layers.
- Such tasks are difficult to operate on a single process.
- How can you effectively resolve such an issue?

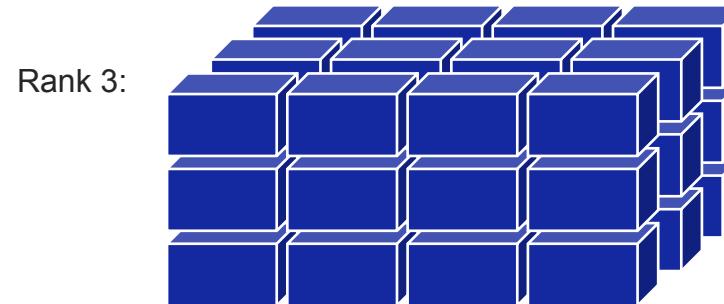
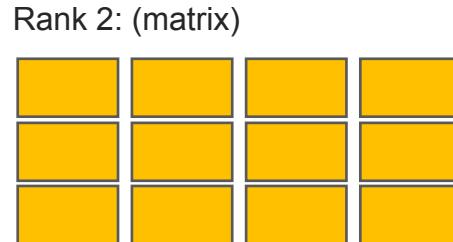
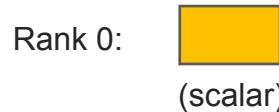
Why TensorFlow?

- A foolproof solution to this issue is a GPU of exceptional performance level.
- A graphic card could be considered as a small computer cluster within a computer.
- You can buy a GPU that carries 290 times more core and executes 10 times more floating point operations per second, with 70% price of the latest CPU.
- Writing a code for a certain CPU is not as easy as running a code on a Python interpreter.
- There are packages such as CUDA or OpenCL that enable usage of designated GPUs.
- However, CUDA or OpenCL does not offer the best environment to write code for machine learning algorithms.
- TensorFlow was developed from such background.

What is TensorFlow?

- Refer to the figure below to precisely understand the notion of a tensor.
- First row shows rank 0 and rank 1 tensors. Second row shows rank 2 and rank 3 tensors.

Ex Tensors



I What is TensorFlow?

- Tensorflow creates a computational graph composed of a series of nodes.
- Each node expresses an operation that can denote more than 0 input or output.
- Tensor was created as a symbolic handle to refer to inputs and outputs of such operation.
- A tensor can be understood in mathematical forms of scalar, vector, matrix etc.
- To be more precise, a scalar can be defined as a rank 0 tensor, vector as rank 1 tensor, matrix as rank 2 tensor, and a matrix with three dimensions a rank 3 tensor.
- The actual values are stored as NumPy arrays, and the tensor provides a reference for this array.

I What is TensorFlow?

- Let's discuss programming mechanisms of TensorFlow.
- Learn how to create and modify tensors.
- Learn how to load data and use TensorFlow Dataset objects which efficiently circulate datasets.
- Study the datasets in the tensorflow datasets module.
- Let's get acquainted with tf.keras API and build a machine learning model.
- Learn how to compile and train this model and store the trained model in a drive.

What is TensorFlow?

- In TensorFlow version 1.x, it supported static computational graph that expressed flow of data.
- Many users had a difficult time utilizing the static computational graph. In the recent 2.0 version, this issue has been resolved, and the user can create and train neural network model more easily.
- TensorFlow 2.0 still supports static computational graphs, but dynamic computational graphs are the default.

Installing TensorFlow

- ▶ Use Python's pip installer to install TensorFlow from PyPI.
- ▶ Run following command on the terminal.

```
> pip install tensorflow
```

- ▶ This command installs the most recent, stable version
- ▶ Version 2.5.0 at the time of writing.
- ▶ Recommend using TensorFlow 2.5.0 for codes (in this chapter) to show expected results.
- ▶ The command below installs a designated version.

```
> pip install tensorflow==[desired-version]
```

Checking TensorFlow version

- Run code as below to check the version

```
In [1]: import tensorflow as tf
```

```
In [2]: tf.__version__
```

```
Out[2]: '2.5.0'
```

Installing TensorFlow

- For GPU, you need an NVIDIA compatible graphic card, and installation of CUDA Toolkit and cuDNN library.
(GPU is recommended for neural network training)
- If above conditions are fulfilled, you can install TensorFlow GPU version.

```
> pip install tensorflow-gpu
```

- Refer to the official website for more information on installation. (<https://www.tensorflow.org/install/gpu>)

TensorFlow learning procedure

- New version is released every few months, due to TensorFlow's fast developing rate.
- Let's study several ways to create a tensor.
- Then, study the features of TensorFlow, and methods for modification.
- First, use **tf.convert to tensor** to create a tensor in list or NumPy array.

```
In [3]: import numpy as np
```

```
In [4]: np.set_printoptions(precision=3)
```

TensorFlow learning procedure

- New version is released every few months, due to TensorFlow's fast developing rate.
- Let's study several ways to create a tensor.
- Then, study the features of TensorFlow, and methods for modification.
- First, use `tf.convert_to_tensor` to create a tensor in list or NumPy array.

```
In [5]: a = np.array([1, 2, 3], dtype=np.int32)
b = [4, 5, 6]

t_a = tf.convert_to_tensor(a)
t_b = tf.convert_to_tensor(b)

print(t_a)
print(t_b)

tf.Tensor([1 2 3], shape=(3,), dtype=int32)
tf.Tensor([4 5 6], shape=(3,), dtype=int32)
```

```
In [6]: tf.is_tensor(a), tf.is_tensor(t_a)
out[6]: (False, True)
```

Creating a tensor in TensorFlow

```
In [7]: t_ones = tf.ones((2, 3))  
t_ones.shape
```

```
Out[7]: TensorShape([2, 3])
```

```
In [8]: t_ones.numpy()
```

```
Out[8]: array([[1., 1., 1.],  
               [1., 1., 1.]], dtype=float32)
```

```
In [9]: const_tensor = tf.constant([1.2, 5, np.pi], dtype=tf.float32)  
print(const_tensor)
```

```
tf.Tensor([1.2 5. 3.142], shape=(3,), dtype=float32)
```

- `tf.convert_to_tensor` function supports `tf.Variable` objects, unlike the `tf.constant` function (which will be discussed soon)
- `tf.fill` function and `tf.one_hot` function creates a tensor as well.

Creating a tensor in TensorFlow

- `tf.fill` function generates a tensor composed of scalar inputs.
- First parameter delivers the shape of the tensor like the `tf.ones` function.
- Second parameter delivers the desired scalar input. The code below returns the following result identical to that of `tf.ones((2,3))`.

```
In [10]: tf.fill((2, 3), 1)
out[10]: <tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[1, 1, 1],
       [1, 1, 1]])>
```

- `tf.fill` function is more efficient than the `tf.ones` to generate tensors of larger size.

Creating a tensor in TensorFlow

- tf.one_hot is a useful function to generate a one-hot encoding matrix.
- First parameter : index that displays the location for one-hot encoding.
- Second parameter : the length of the one-hot encoding vector.
- The size of the generated matrix is (length of the first parameter x second parameter)

Ex The code below generates a one-hot encoding matrix of (3 x 4) size.

```
In [11]: tf.one_hot([0, 1, 2], 4)
Out[11]: <tf.Tensor: shape=(3, 4), dtype=float32, numpy=
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]], dtype=float32)>
```

Modifying shape and data type of a tensor

```
In [12]: t_a_new = tf.cast(t_a, tf.int64)
          print(t_a_new.dtype)
          <dtype: 'int64'>

In [13]: t = tf.random.uniform(shape=(3, 5))
          t_tr = tf.transpose(t)
          print(t.shape, '-->', t_tr.shape)
          (3, 5) --> (5, 3)
```

Modifying shape and data type of a tensor

```
In [14]: t = tf.zeros((30,))

t_reshape = tf.reshape(t, shape=(5, 6))

print(t_reshape.shape)

(5, 6)
```

```
In [15]: t = tf.zeros((1, 2, 1, 4, 1))

t_sqz = tf.squeeze(t, axis=(2, 4))

print(t.shape, ' --> ', t_sqz.shape)

(1, 2, 1, 4, 1) --> (1, 2, 4)
```

| Applying math operation on a tensor

```
In [16]: tf.random.set_seed(1)

t1 = tf.random.uniform(shape=(5, 2),
                      minval=-1.0,
                      maxval=1.0)
```

```
t2 = tf.random.normal(shape=(5, 2),
                      mean=0.0,
                      stddev=1.0)
```

```
In [17]: t3 = tf.multiply(t1, t2).numpy()
print(t3)
```

```
[[ -0.27 -0.874]
 [-0.017 -0.175]
 [-0.296 -0.139]
 [-0.727  0.135]
 [-0.401  0.004]]
```

Applying math operation on a tensor

```
In [18]: t4 = tf.math.reduce_mean(t1, axis=0)  
print(t4)  
  
tf.Tensor([0.09  0.207], shape=(2,), dtype=float32)
```

```
In [19]: t5 = tf.linalg.matmul(t1, t2, transpose_b=True)  
print(t5.numpy())  
  
[[ -1.144  1.115 -0.87  -0.321  0.856]  
 [ 0.248 -0.191  0.25   -0.064 -0.331]  
 [ -0.478  0.407 -0.436  0.022  0.527]  
 [ 0.525 -0.234  0.741 -0.593 -1.194]  
 [ -0.099  0.26   0.125 -0.462 -0.396]]
```

```
In [20]: t6 = tf.linalg.matmul(t1, t2, transpose_a=True)  
print(t6.numpy())  
  
[[ -1.711  0.302]  
 [ 0.371 -1.049]]
```

Applying math operation on a tensor

```
In [21]: norm_t1 = tf.norm(t1, ord=2, axis=1).numpy()  
print(norm_t1)
```

```
[1.046 0.293 0.504 0.96 0.383]
```

```
In [22]: np.sqrt(np.sum(np.square(t1), axis=1))
```

```
Out[22]: array([1.046, 0.293, 0.504, 0.96 , 0.383], dtype=float32)
```

Applying math operation on a tensor

- NumPy function loads `__array__()` method of the pertaining object before processing the input parameter. This part of the process makes the object compatible with NumPy.
- For example, you can use Series object from pandas in NumPy API. Since this method is embodied in the tensor as well, you can put a tensor as an input in a NumPy function.
- Many mathematical functions are referable in the most advanced level. For example, they can be referred as `tf.multiply()`, `tf.reduce_mean()`, `tf.reduce_sum()`, `tf.matmul()`. For Python 3.5 version or above, it can perform matrix operations using `@` operator. For example, The computation below return the results identical to that of the previous slide.

```
In [23]: t1 @ tf.transpose(t2)
```

```
Out[23]: <tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[-1.144,  1.115, -0.87 , -0.321,  0.856],
       [ 0.248, -0.191,  0.25 , -0.064, -0.331],
       [-0.478,  0.407, -0.436,  0.022,  0.527],
       [ 0.525, -0.234,  0.741, -0.593, -1.194],
       [-0.099,  0.26 ,  0.125, -0.462, -0.396]], dtype=float32)>
```

| split(), stack(), concatenate() function

```
In [24]: tf.random.set_seed(1)

t = tf.random.uniform((6,))

print(t.numpy())

t_splits = tf.split(t, 3)

[item.numpy() for item in t_splits]

[0.165 0.901 0.631 0.435 0.292 0.643]

out[24]: [array([0.165, 0.901], dtype=float32),
          array([0.631, 0.435], dtype=float32),
          array([0.292, 0.643], dtype=float32)]
```

| split(), stack(), concatenate() function

```
In [26]: A = tf.ones((3,))  
B = tf.zeros((2,))  
  
C = tf.concat([A, B], axis=0)  
print(C.numpy())  
  
[1. 1. 1. 0. 0.]
```

```
In [27]: A = tf.ones((3,))  
B = tf.zeros((3,))  
  
S = tf.stack([A, B], axis=1)  
print(S.numpy())  
  
[[1. 0.]  
 [1. 0.]  
 [1. 0.]]
```

Coding Exercise #0601



Follow practice steps on 'ex_0601.ipynb' file

Unit 2.

TensorFlow Basics

- | 2.1. Introduction to TensorFlow
- | 2.2. TensorFlow dataset
- | 2.3. Machine Learning with TensorFlow

Building TensorFlow dataset in a tensor

```
In [28]: a = [1.2, 3.4, 7.5, 4.1, 5.0, 1.0]
ds = tf.data.Dataset.from_tensor_slices(a)
print(ds)
<TensorSliceDataset shapes: (), types: tf.float32>
```

```
In [29]: for item in ds:
    print(item)

tf.Tensor(1.2, shape=(), dtype=float32)
tf.Tensor(3.4, shape=(), dtype=float32)
tf.Tensor(7.5, shape=(), dtype=float32)
tf.Tensor(4.1, shape=(), dtype=float32)
tf.Tensor(5.0, shape=(), dtype=float32)
tf.Tensor(1.0, shape=(), dtype=float32)
```

```
In [30]: ds_batch = ds.batch(3)

for i, elem in enumerate(ds_batch, 100):
    print('batch {}: {}'.format(i), elem.numpy())

batch 100: [1.2 3.4 7.5]
batch 101: [4.1 5.  1. ]
```

Connecting two tensors in one dataset (1/2)

I Approach 1

```
In [31]: tf.random.set_seed(1)

t_x = tf.random.uniform([4, 3], dtype=tf.float32)
t_y = tf.range(4)

In [32]: ds_x = tf.data.Dataset.from_tensor_slices(t_x)
ds_y = tf.data.Dataset.from_tensor_slices(t_y)

ds_joint = tf.data.Dataset.zip((ds_x, ds_y))

for example in ds_joint:
    print('  x: ', example[0].numpy(),
          '  y: ', example[1].numpy())

  x:  [0.165 0.901 0.631]  y:  0
  x:  [0.435 0.292 0.643]  y:  1
  x:  [0.976 0.435 0.66 ]  y:  2
  x:  [0.605 0.637 0.614]  y:  3
```

Connecting two tensors in one dataset (2/2)

I Approach 2

```
In [33]: ds_joint = tf.data.Dataset.from_tensor_slices((t_x, t_y))

for example in ds_joint:
    print(' x: ', example[0].numpy(),
          ' y: ', example[1].numpy())

x: [0.165 0.901 0.631]  y:  0
x: [0.435 0.292 0.643]  y:  1
x: [0.976 0.435 0.66 ]  y:  2
x: [0.605 0.637 0.614]  y:  3
```

```
In [34]: ds_trans = ds_joint.map(lambda x, y: (x*2-1.0, y))

for example in ds_trans:
    print(' x: ', example[0].numpy(),
          ' y: ', example[1].numpy())

x: [-0.67   0.803  0.262]  y:  0
x: [-0.131 -0.416  0.285]  y:  1
x: [ 0.952 -0.13   0.32 ]  y:  2
x: [ 0.21   0.273  0.229]  y:  3
```

shuffle(), batch(), repeat() method (1/6)

```
In [35]: tf.random.set_seed(1)
ds = ds_joint.shuffle(buffer_size=len(t_x))

for example in ds:
    print(' x: ', example[0].numpy(),
          ' y: ', example[1].numpy())

x:  [0.976 0.435 0.66 ]  y:  2
x:  [0.435 0.292 0.643]  y:  1
x:  [0.165 0.901 0.631]  y:  0
x:  [0.605 0.637 0.614]  y:  3
```

```
In [36]: ds = ds_joint.batch(batch_size=3,
                         drop_remainder=False)

batch_x, batch_y = next(iter(ds))

print('batch x: \n', batch_x.numpy())
print('batch y:   ', batch_y.numpy())

batch x:
 [[0.165 0.901 0.631]
 [0.435 0.292 0.643]
 [0.976 0.435 0.66 ]]
batch y:   [0 1 2]
```

shuffle(), batch(), repeat() method (2/6)

```
In [37]: ds = ds_joint.batch(3).repeat(count=2)

for i,(batch_x, batch_y) in enumerate(ds):
    print(i, batch_x.shape, batch_y.numpy())

0 (3, 3) [0 1 2]
1 (1, 3) [3]
2 (3, 3) [0 1 2]
3 (1, 3) [3]
```

```
In [38]: ds = ds_joint.repeat(count=2).batch(3)

for i,(batch_x, batch_y) in enumerate(ds):
    print(i, batch_x.shape, batch_y.numpy())

0 (3, 3) [0 1 2]
1 (3, 3) [3 0 1]
2 (2, 3) [2 3]
```

shuffle(), batch(), repeat() method (3/6)

```
In [39]: tf.random.set_seed(1)

## Step 1 : shuffle -> batch -> repeat
ds = ds_joint.shuffle(4).batch(2).repeat(3)

for i,(batch_x, batch_y) in enumerate(ds):
    print(i, batch_x.shape, batch_y.numpy())

0 (2, 3) [2 1]
1 (2, 3) [0 3]
2 (2, 3) [0 3]
3 (2, 3) [1 2]
4 (2, 3) [3 0]
5 (2, 3) [1 2]
```

shuffle(), batch(), repeat() method (4/6)

```
In [40]: tf.random.set_seed(1)

## Step 1': shuffle -> batch -> repeat
ds = ds_joint.shuffle(4).batch(2).repeat(20)

for i,(batch_x, batch_y) in enumerate(ds):
    print(i, batch_x.shape, batch_y.numpy())
```

```
0 (2, 3) [2 1]
1 (2, 3) [0 3]
2 (2, 3) [0 3]
3 (2, 3) [1 2]
4 (2, 3) [3 0]
5 (2, 3) [1 2]
6 (2, 3) [1 3]
7 (2, 3) [2 0]
8 (2, 3) [1 2]
9 (2, 3) [3 0]
10 (2, 3) [3 0]
11 (2, 3) [2 1]
12 (2, 3) [3 0]
13 (2, 3) [1 2]
14 (2, 3) [3 0]
15 (2, 3) [2 1]
16 (2, 3) [2 3]
```

shuffle(), batch(), repeat() method (4/6)

```
In [41]: tf.random.set_seed(1)

## Step 2 : batch -> shuffle -> repeat
ds = ds_joint.batch(2).shuffle(4).repeat(3)

for i,(batch_x, batch_y) in enumerate(ds):
    print(i, batch_x.shape, batch_y.numpy())

0 (2, 3) [0 1]
1 (2, 3) [2 3]
2 (2, 3) [0 1]
3 (2, 3) [2 3]
4 (2, 3) [2 3]
5 (2, 3) [0 1]
```

shuffle(), batch(), repeat() method (5/6)

```
In [42]: tf.random.set_seed(1)

## Step 2 : batch -> shuffle -> repeat
ds = ds_joint.batch(2).shuffle(4).repeat(20)

for i,(batch_x, batch_y) in enumerate(ds):
    print(i, batch_x.shape, batch_y.numpy())
```

```
0 (2, 3) [0 1]
1 (2, 3) [2 3]
2 (2, 3) [0 1]
3 (2, 3) [2 3]
4 (2, 3) [2 3]
5 (2, 3) [0 1]
6 (2, 3) [2 3]
7 (2, 3) [0 1]
8 (2, 3) [2 3]
9 (2, 3) [0 1]
10 (2, 3) [2 3]
11 (2, 3) [0 1]
12 (2, 3) [2 3]
13 (2, 3) [0 1]
14 (2, 3) [2 3]
15 (2, 3) [0 1]
16 (2, 3) [0 1]
```

shuffle(), batch(), repeat() method (6/6)

```
In [43]: tf.random.set_seed(1)

## Step 3 : batch -> repeat -> shuffle
ds = ds_joint.batch(2).repeat(3).shuffle(4)

for i,(batch_x, batch_y) in enumerate(ds):
    print(i, batch_x.shape, batch_y.numpy())

0 (2, 3) [0 1]
1 (2, 3) [0 1]
2 (2, 3) [2 3]
3 (2, 3) [2 3]
4 (2, 3) [0 1]
5 (2, 3) [2 3]
```

Building dataset from a file stored in a local drive (1/6)

```
In [44]: import pathlib  
imgdir_path = pathlib.Path('cat_dog_images')  
file_list = sorted([str(path) for path in imgdir_path.glob('*.*')])  
print(file_list)  
['cat_dog_images\\cat-01.jpg', 'cat_dog_images\\cat-02.jpg', 'cat_dog_images\\cat-03.jpg', 'cat_dog_images\\dog-01.jpg',  
'cat_dog_images\\dog-02.jpg', 'cat_dog_images\\dog-03.jpg']
```

Building dataset from a file stored in a local drive (2/6)

```
In [45]: import matplotlib.pyplot as plt
import os

fig = plt.figure(figsize=(10, 5))
for i,file in enumerate(file_list):
    img_raw = tf.io.read_file(file)
    img = tf.image.decode_image(img_raw)
    print( 'Image Size: ', img.shape)
    ax = fig.add_subplot(2, 3, i+1)
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(img)
    ax.set_title(os.path.basename(file), size=15)

# plt.savefig('images/13_1.png', dpi=300)
plt.tight_layout()
plt.show()
```

Building dataset from a file stored in a local drive (3/6)

Image size: (900, 1200, 3)
Image size: (900, 1200, 3)
Image size: (900, 742, 3)
Image size: (800, 1200, 3)
Image size: (800, 1200, 3)
Image size: (900, 1200, 3)
Image size:

cat-01.jpg



cat-02.jpg



cat-03.jpg



dog-01.jpg



dog-02.jpg



dog-03.jpg



Building dataset from a file stored in a local drive (4/6)

```
In [46]: labels = [1 if 'dog' in os.path.basename(file) else 0
                 for file in file_list]
print(labels)

[0, 0, 0, 1, 1, 1]
```

```
In [47]: ds_files_labels = tf.data.Dataset.from_tensor_slices(
            (file_list, labels))

for item in ds_files_labels:
    print(item[0].numpy(), item[1].numpy())

b'cat_dog_images\\cat-01.jpg' 0
b'cat_dog_images\\cat-02.jpg' 0
b'cat_dog_images\\cat-03.jpg' 0
b'cat_dog_images\\dog-01.jpg' 1
b'cat_dog_images\\dog-02.jpg' 1
b'cat_dog_images\\dog-03.jpg' 1
```

Building dataset from a file stored in a local drive (5/6)

```
In [48]: def load_and_preprocess(path, label):
    image = tf.io.read_file(path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [img_height, img_width])
    image /= 255.0

    return image, label

img_width, img_height = 120, 80

ds_images_labels = ds_files_labels.map(load_and_preprocess)

fig = plt.figure(figsize=(10, 5))
for i,example in enumerate(ds_images_labels):
    print(example[0].shape, example[1].numpy())
    ax = fig.add_subplot(2, 3, i+1)
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(example[0])
    ax.set_title('{}'.format(example[1].numpy()),
                 size=15)

plt.tight_layout()

plt.show()
```

Building dataset from a file stored in a local drive (6/6)

```
(80, 120, 3) 0  
(80, 120, 3) 0  
(80, 120, 3) 0  
(80, 120, 3) 1  
(80, 120, 3) 1  
(80, 120, 3) 1
```



Loading datasets from tensorflow.datasets library (1/10)

```
In [49]: !pip install --upgrade tensorflow-datasets
Collecting tensorflow-datasets
  Downloading tensorflow_datasets-4.4.0-py3-none-any.whl (4.0 MB)
Requirement already satisfied: tqdm in c:\users\emcast\anaconda3\lib\site-packages (from tensorflow-datasets) (4.59.0)
Collecting dill
  Downloading dill-0.3.4-py2.py3-none-any.whl (86 kB)
Requirement already satisfied: six in c:\users\emcast\anaconda3\lib\site-packages (from tensorflow-datasets) (1.15.0)
Requirement already satisfied: absl-py in c:\users\emcast\anaconda3\lib\site-packages (from tensorflow-datasets) (0.13.0)
Collecting promise
  Downloading promise-2.3.tar.gz (19 kB)
```

Loading datasets from tensorflow.datasets library (2/10)

```
In [50]: import tensorflow_datasets as tfds  
  
print(len(tfds.list_builders()))  
print(tfds.list_builders()[:5])  
  
278  
['abstract_reasoning', 'accentdb', 'aeslc', 'aflw2k3d', 'ag_news_subset']
```

Loading datasets from tensorflow.datasets library (3/10)

The following command generates the entire list.

```
In [51]: tfds.list_builders()  
Out[51]: ['abstract_reasoning',  
          'accentdb',  
          'aeslc',  
          'aflw2k3d',  
          'ag_news_subset',  
          'ai2_arc',  
          'ai2_arc_with_ir',  
          'amazon_us_reviews',  
          'anli',  
          'arc',  
          'bair_robot_pushing_small',  
          'bccd',  
          'beans',  
          'big_patent',  
          'bigearthnet',  
          'billsum',  
          'binarized_mnist',  
          'binary_alpha_digits',  
          'blimp',  
          'bool_q',  
          'c4',  
          'caltech101',  
          'caltech_birds2010',  
          'caltech_birds2011',  
          'cars196',  
          'cassava']
```

Loading datasets from tensorflow.datasets library (4/10)

Downloading CelebA dataset.

```
In [52]: celeba_bldr = tfds.builder('celeb_a')

print(celeba_bldr.info.features)
print('\n', 30*"-", '\n')
print(celeba_bldr.info.features.keys())
print('\n', 30*"-", '\n')
print(celeba_bldr.info.features['image'])
print('\n', 30*"-", '\n')
print(celeba_bldr.info.features['attributes'].keys())
print('\n', 30*"-", '\n')
print(celeba_bldr.info.citation)

FeaturesDict({
    'attributes': FeaturesDict({
        '5_o_Clock_Shadow': tf.bool,
        'Arched_Eyebrows': tf.bool,
        'Attractive': tf.bool,
        'Bags_Under_Eyes': tf.bool,
        'Bald': tf.bool,
        'Bangs': tf.bool,
        'Big_Lips': tf.bool,
        'Big_Nose': tf.bool,
        'Black_Hair': tf.bool,
        'Blond_Hair': tf.bool,
```

Loading datasets from tensorflow.datasets library (5/10)

- Download and save data in the local drive.

```
In [53]: celeba_bldr.download_and_prepare()
          Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to
          C:\Users\emcast\tensorflow_datasets\celeb_a\2.0.1...
          DL Completed...: 100%  4/4 [08:34<00:00, 197.62s/ url]
          DL Size...:  1414/0 [08:34<00:00, 2.65 MiB/s]
          Dataset celeb_a downloaded and prepared to C:\Users\emcast\tensorflow_datasets\celeb_a\2.0.1. Subsequent calls will reuse
          this data.
```

Loading datasets from tensorflow.datasets library (6/10)

- | Load data from the drive by tf.data.Datasets

```
In [54]: datasets = celeba_bldr.as_dataset(shuffle_files=False)
datasets.keys()
Out[54]: dict_keys([Split('train'), Split('validation'), Split('test')])
```

Loading datasets from tensorflow.datasets library (7/10)

```
In [55]: ds_train = datasets['train']
assert isinstance(ds_train, tf.data.Dataset)

example = next(iter(ds_train))
print(type(example))
print(example.keys())

<class 'dict'>
dict_keys(['attributes', 'image', 'landmarks'])
```

Loading datasets from tensorflow.datasets library (8/10)

```
In [56]: ds_train = ds_train.map(lambda item:  
    (item['image'], tf.cast(item['attributes']['Male'], tf.int32)))  
  
In [57]: ds_train = ds_train.batch(18)  
images, labels = next(iter(ds_train))  
  
print(images.shape, labels)  
(18, 218, 178, 3) tf.Tensor([0 1 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 1], shape=(18,), dtype=int32)
```

Loading datasets from tensorflow.datasets library (9/10)

```
In [58]: fig = plt.figure(figsize=(12, 8))
for i,(image,label) in enumerate(zip(images, labels)):
    ax = fig.add_subplot(3, 6, i+1)
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(image)
    ax.set_title('{}' .format(label), size=15)

# plt.savefig('images/13_3.png', dpi=300)
plt.show()
```

Loading datasets from tensorflow.datasets library (10/10)



Another approach to loading datasets (1/3)

```
In [59]: mnist, mnist_info = tfds.load('mnist', with_info=True,
                                         shuffle_files=False)

print(mnist_info)
print(mnist.keys())

Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to
C:\Users\emcast\tensorflow_datasets\mnist\3.0.1...

DI Completed...: 100% [ 4/4 [00:09<00:00, 2.57s/ url]

DI Size....: 100% [ 10/10 [00:09<00:00, 1.47 MiB/s]

Extraction completed...: 100% [ 4/4 [00:09<00:00, 2.77s/ file]
```

Another approach to loading datasets (2/3)

```
In [60]: ds_train = mnist['train']

assert isinstance(ds_train, tf.data.Dataset)

ds_train = ds_train.map(lambda item:
    (item['image'], item['label']))

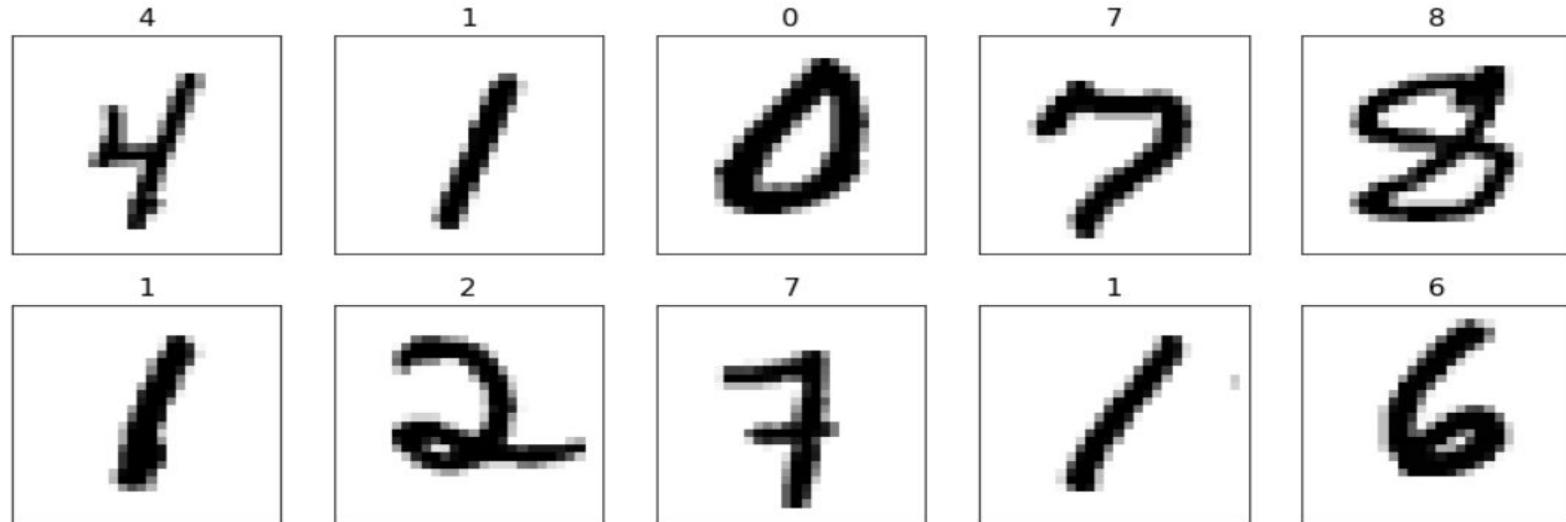
ds_train = ds_train.batch(10)
batch = next(iter(ds_train))
print(batch[0].shape, batch[1])

fig = plt.figure(figsize=(15, 6))
for i,(image,label) in enumerate(zip(batch[0], batch[1])):
    ax = fig.add_subplot(2, 5, i+1)
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(image[:, :, 0], cmap='gray_r')
    ax.set_title('{0}'.format(label), size=15)

plt.show()
```

Another approach to loading datasets (3/3)

```
(10, 28, 28, 1) tf.Tensor([4 1 0 7 8 1 2 7 1 6], shape=(10,), dtype=int64)
```



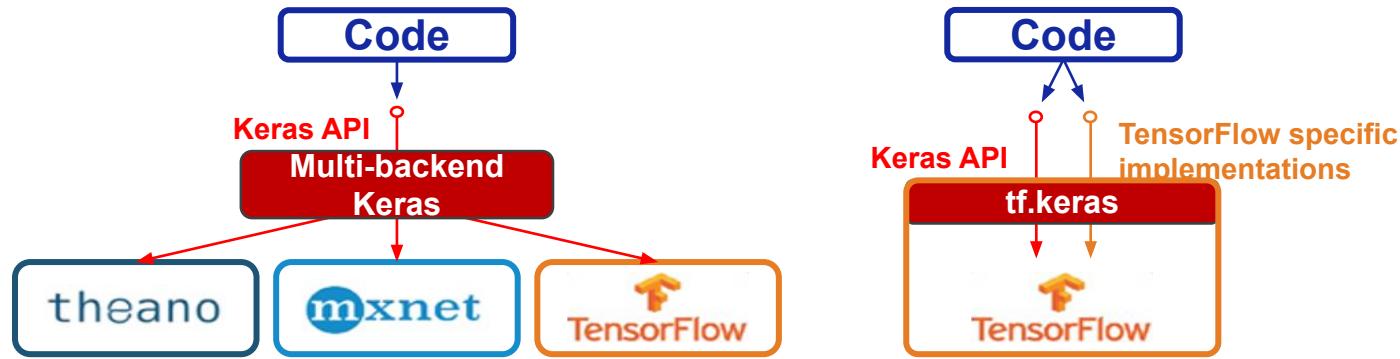
Unit 2.

TensorFlow Basics

- | 2.1. Introduction to TensorFlow
- | 2.2. TensorFlow Dataset
- | 2.3. Machine Learning with TensorFlow

Keras

- high-level API that allows users to build, train, evaluate and execute all kinds of neural networks.



Building a neural network with TensorFlow

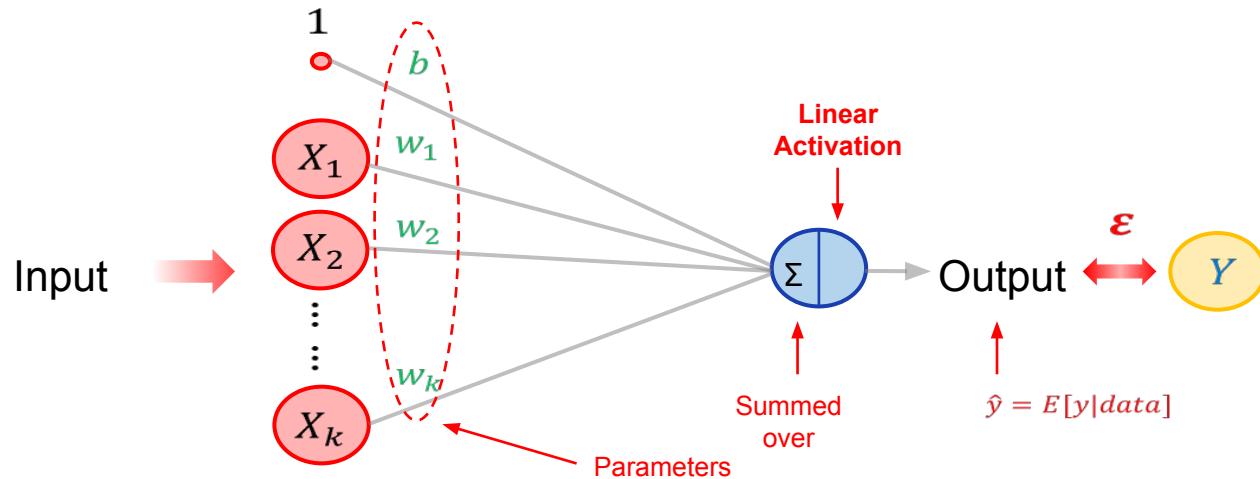
Linear regression with TensorFlow:

- ▶ There is one or more explanatory variables: X_1, X_2, \dots, X_k
- ▶ There is one response variable: Y
- ▶ The variables X_i and Y are connected by a linear relation:

$$Y = b + w_1 X_1 + w_2 X_2 + \dots + w_k X_k + \varepsilon$$

- ▶ The coefficients previously denoted as β_{-i} , from now and on will be called “weights” w_{-i} .
- ▶ The intercept previously denoted as β_{-0} , from now and on will be called “bias” b .
- ▶ We will use the “linear” activation function.

| Linear regression with TensorFlow:



- ▶ The disagreement between the predicted \hat{y} and the true y has to be minimized by training.

Linear regression with TensorFlow:

- ▶ As we have an over determined system of linear equations, the exact solution does not exist.
- ▶ We can minimize the loss (cost) defined as $|\mathbf{Y} - \hat{\mathbf{Y}}|^2$ or $|\mathbf{Y} - \hat{\mathbf{Y}}|$ and get the “best” solution (\mathbf{b}, \mathbf{W}) .
- ▶ If we define the loss as $L = |\mathbf{Y} - \hat{\mathbf{Y}}|^2$, the solution (\mathbf{b}, \mathbf{W}) can be calculated using matrices.
 - ▶ However, with TensorFlow we will take a different approach: **Gradient descent algorithm**.

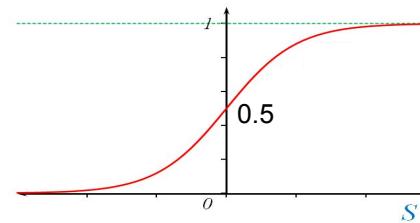
Logistic regression with TensorFlow:

- The linear combinations of variables X_i is the so-called “Logit” denoted here as S :

$$S = b + w_1 X_1 + w_2 X_2 + \dots + w_k X_k$$

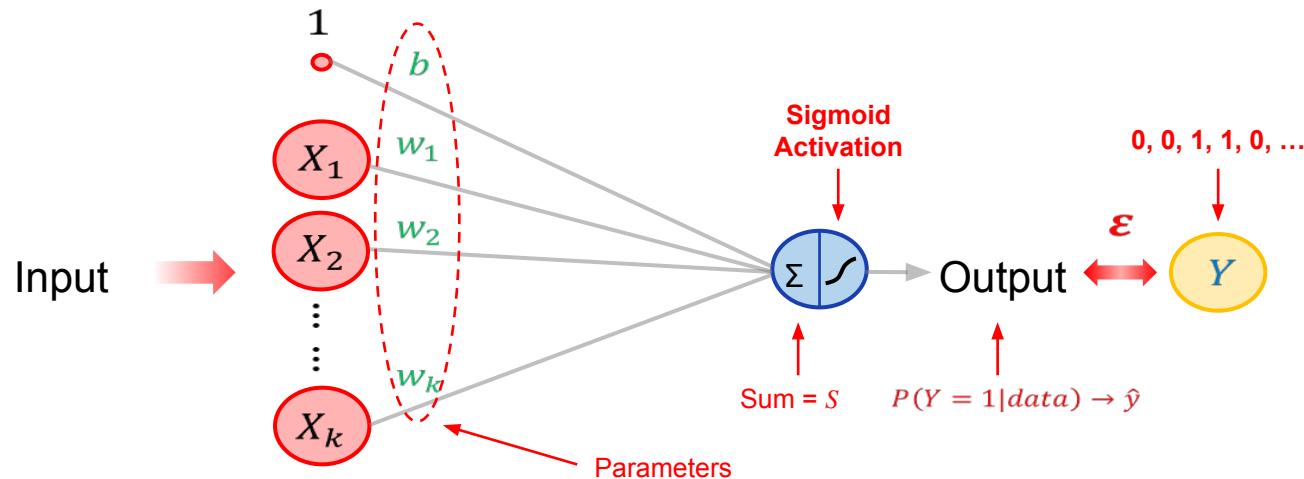
- The conditional probability of Y being equal to 1 is denoted as $P(Y=1|x_i)$.
- A “Sigmoid” function connects the probability with the logit:

$$\sigma(S) = \frac{e^S}{1 + e^S}$$



- This Sigmoid is the “activation function” \leftarrow the biggest difference with the linear regression.

| Logistic regression with TensorFlow:



- ▶ The mismatch between the predicted \hat{y} and the true y has to be minimized by training.

Logistic regression with TensorFlow:

- ▶ We can recall the negative of logarithmic likelihood with the redefined $y_i = -1$ or $+1$. (*)

$$-\sum_{i=1}^n \text{Log}(1 + e^{-y_i s})$$

- ▶ We now rename this as the “entropy” and express it in terms of the predicted probability \hat{p}_i .

$$L = -\frac{1}{n} \sum_{i=1}^n (y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i))$$

- ▶ We went back to the definition $y_i = 0$ or 1 .
- ▶ This is the loss function we’d like to minimize by the gradient descent algorithm.
- ▶ Binary logistic regression can be generalized to the **multi-class version** using the Softmax activation and multi-class cross entropy as the loss function.

I Loss (cost) functions in TensorFlow:

Name	Formula	TensorFlow Expression
Squared Error (L2 Error)		<code>tf.reduce_sum(tf.square(y_true - y_model))</code>
Absolute Error (L1 Error)		<code>tf.reduce_sum(tf.abs(y_true - y_model))</code>
Binary Cross Entropy		<code>tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=y_true, logits=y_model))</code>
Multi-Class Cross Entropy		<code>tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true, logits=y_model))</code>

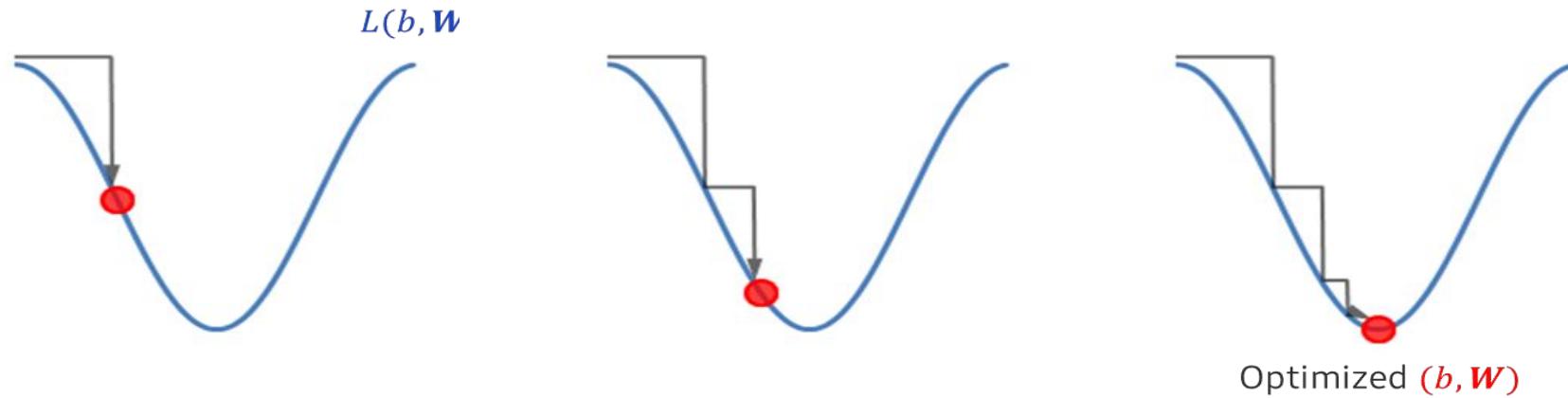
- Cross entropy is the appropriate loss function for the classification.

Gradient descent algorithm:

- ▶ $L(b, \mathbf{W})$ is minimized iteratively in small “steps” pushing (b, \mathbf{W}) along the direction $-\nabla L(b, \mathbf{W})$:
- 1) (b, \mathbf{W}) is randomly initialized.
 - 2) Calculate the gradient $\nabla L(b, \mathbf{W})$.
 - 3) Update (b, \mathbf{W}) by one step: $(b, \mathbf{W}) \leftarrow (b, \mathbf{W}) - \eta \nabla L(b, \mathbf{W})$.
Convergence speed is controlled by the “Learning rate” η .
 - 4) Repeat from the step 2) for a fix number of times (epochs).

| Gradient descent algorithm:

- $L(b, W)$ is minimized iteratively in small “steps” pushing (b, W) along the direction $-\nabla L(b, W)$:



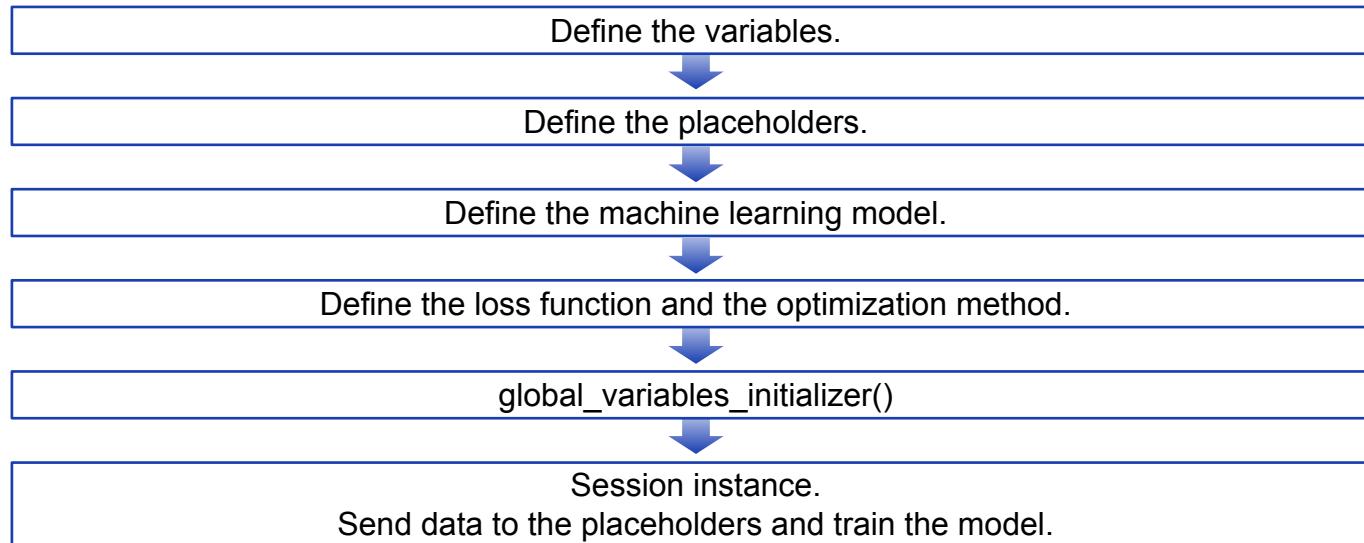
I Batch learning (or mini-batch gradient descent):

- Instead of using the whole training dataset for calculating the gradient, only a subset (batch) is used.
- Batch is randomly sampled at every gradient descent step.
- Batch size is small independent of the whole training data size.
- It is advantageous for the generalization (testing).
- It requires less computing power.

Gradient descent algorithms:

Algorithm	Explanation	TensorFlow
Gradient Descent	Gradient descent algorithm with batch learning.	<code>tf.train.GradientDescentOptimizer()</code>
Momentum	Takes into account the gradient from the previous step.	<code>tf.train.MomentumOptimizer(nesterov=False)</code>
Nesterov Momentum	Minimizes unnecessary movements.	<code>tf.train.MomentumOptimizer(nesterov=True)</code>
Adagrad	Adaptative step size.	<code>tf.train.AdagradOptimizer()</code>
RMSProp	Improves upon the Adagrad.	<code>tf.train.RMSPropOptimizer()</code>
Adam	Combines the best properties of Adagrad and RMSProp.	<code>tf.train.AdamOptimizer()</code>

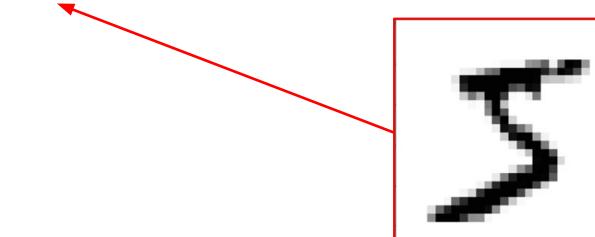
I Training work flow:



I MNIST data of handwritten digits:

- Handwritten images of digits (0~9). Image size is 28×28 (748 pixels).
- There are 55,000 training and 10,000 testing cases.
- The digit label is one-hot-encoded.

X0	X1	X2	X3	X4	X5	X6	X7	X8	X9
0	0	0	0	0	1	0	0	0	0



The diagram illustrates the one-hot encoding of a handwritten digit. A 10-column table represents the input features (X0 to X9) and the output label. The digit '4' is shown in the X5 column, indicating it is the correct class. Below the table, a red arrow points from the value '1' in the X5 column to a small 28x28 pixel image of a handwritten digit '4' enclosed in a red box.

Coding Exercise #0602



Follow practice steps on 'ex_0602.ipynb' file

Coding Exercise #0603



Follow practice steps on 'ex_0603.ipynb' file

Coding Exercise #0604



Follow practice steps on 'ex_0604.ipynb' file

Unit 3.

Deep Learning Methods with TensorFlow and Keras

- | 3.1. Main Features of TensorFlow
- | 3.2. Keras Basics
- | 3.3. AI with Keras

Main features of TensorFlow (1/3)

- TensorFlow is a programming interface used to build and execute machine learning algorithms. It supports many platforms, and it is flexible for expansion.
- TensorFlow API has become more stable and mature after the initial release of 1.0 in 2017.
- It was renewed after the release of 2.0 in 2019.

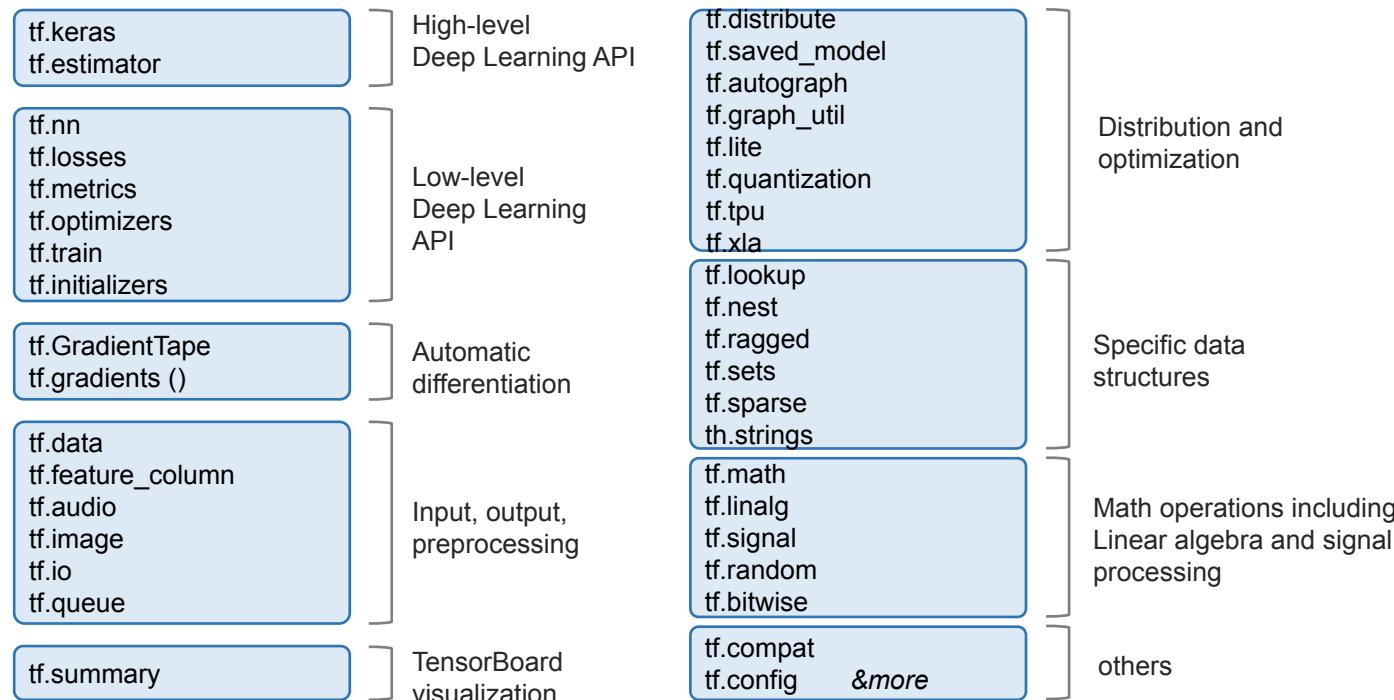
Main features of TensorFlow (2/3)

- Based on user feedbacks, the TensorFlow team decided to support dynamic computational graph for TensorFlow 2.0.
- Thus, Building and training neural networks became much easier.
- Since dynamic computational graphs iterate graph execution and graph declaration, TensorFlow 2.0 will be much more natural and comfortable for Python and NumPy users.
- TensorFlow 2.0 offers TensorFlow 1.x API in `tf.compat` module.
- Using the above model, users will be able to gradually migrate their codes to TensorFlow v.2 API.

Main features of TensorFlow (3/3)

- Another central feature of TensorFlow is capability to use one or multiple GPUs.
- You can efficiently train deep learning models in large-scale datasets and systems.
- TensorFlow is an open-source library for the public, but it is sponsored and supported by Google.
- A large team of software engineers is consistently expanding and improving the library.
- Since it is an open-source library, many developers are supporting TensorFlow.

Summary of the entire Python API in TensorFlow



Unit 3.

Deep Learning Methods with TensorFlow and Keras

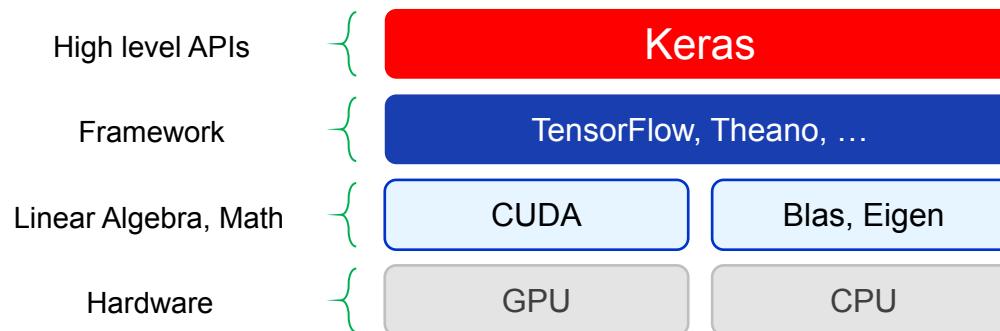
- | 3.1. Main Features of TensorFlow
- | 3.2. Keras Basics
- | 3.3. AI with Keras

Keras basics

- The most popular backend up to this day was TensorFlow which was the default backend for Keras.
- Since more and more TensorFlow users worked with Keras for high-level APIs, TensorFlow developers had to consider containing Keras project in a separate module (`tf.keras`).
- In TensorFlow 2.0, Keras and `tf.keras` are synchronized.

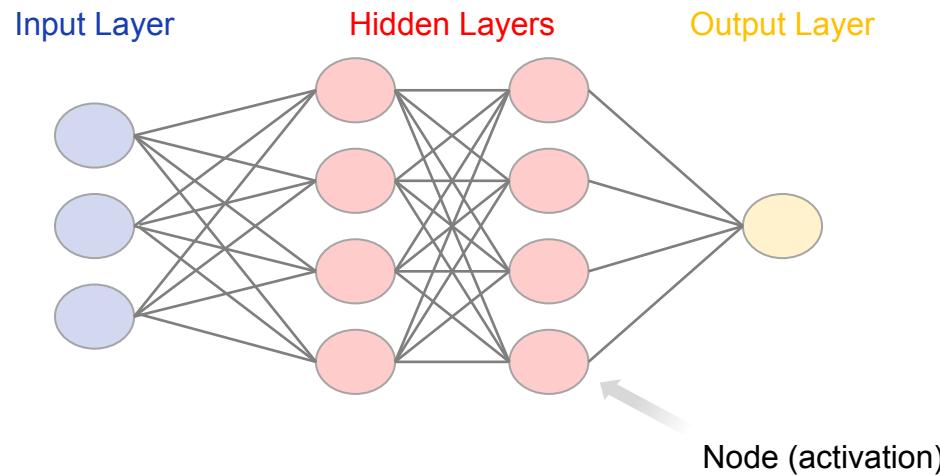
About the Keras library

- A high-end deep learning library.
- Does not support the low-end tensor operations.
- Runs on top of TensorFlow.



About the Keras library

- It is easy to model deep neural networks with Keras.



I Training work flow:

- It is easy to model deep neural networks with Keras.

Define a model's layer structure (specify the activation functions).



Define the loss function and the optimization method.



Compile the model.



Specify the batch learning method and start the training (gradient descent).

Loss (cost) functions in Keras:

- It is easy to model deep neural networks with Keras.

Name	Formula	Keras Expression
Squared Error (L2 Error)		<code>loss = "mse"</code>
Absolute Error (L1 Error)		<code>loss = "mae"</code>
Binary Cross Entropy		<code>loss = "binary_crossentropy"</code>
Multi-Class Cross Entropy		<code>loss = "categorical_crossentropy"</code>

Gradient descent algorithms:

Algorithm	Explanation	Keras
Stochastic Gradient Descent	Gradient descent algorithm with batch learning.	<code>SGD(lr=0.1)</code>
Momentum	Takes into account the gradient from the previous step.	<code>SGD(momentum=0.9)</code>
Nesterov Momentum	Minimizes unnecessary movements.	<code>SGD(lr=0.01, momentum=0.9, nesterov=True)</code>
Adagrad	Adaptative step size.	<code>Adagrad(lr=0.01)</code>
RMSProp	Improves upon the Adagrad.	<code>RMSProp(lr=0.01)</code>
Adam	Combines the best properties of Adagrad and RMSProp.	<code>Adam(lr=0.01)</code>

Activation functions:

Name	Formula	Keras
Linear		activation="linear"
Sigmoid		activation="sigmoid"
Tanh		activation="tanh"
ReLU		activation="relu"
Softmax		activation="softmax"

Layers:

Keras	Explanation
Dense()	Dense layer.
Conv1D()	1D convolution layer.
Conv2D()	2D convolution layer.
MaxPooling1D()	1D max pooling layer.
MaxPooling2D()	2D max pooling layer.
Dropout()	Applies dropout to the input.
Flatten()	Flattens the input.
Embedding()	Turns positive integers (label encoding) into dense vectors.
SimpleRNN()	RNN where the output is fed back to input.
LSTM()	LSTM layer.
TimeDistributed()	A wrapper that applies a layer to every time slice.
Input()	The first layer for a functional API model.

Two ways of building models with Keras:

1). Sequential API:

- Layers are added to a “Sequential” object.
- Easy but less flexible.

2). Functional API:

- The first layer must be defined as “Input” where the data shape is specified.
- Each layer defined as a function that takes in as argument the previous layer.
- A “Model” object takes in as arguments the first and the last layers.
- More flexible.

I Datasets provided by Keras:

- Useful for learning/practicing purpose.
- Provided by the “keras.datasets” module.

Name	Description
cifar10	
cifar100	
imdb	25000 movie reviews from IMDB with binary labeling (positive or negative).
reuters	11228 newswires from Reuters, labeled over 46 topics.
mnist	
fashion_mnist	
boston_housing	13 attributes of houses at different locations around Boston suburbs in the late 1970s.

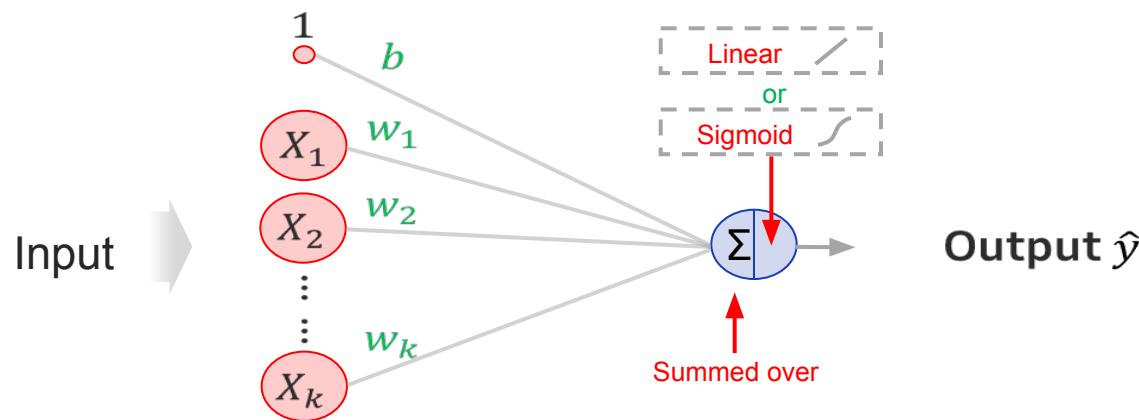
Unit 3.

Deep Learning Methods with TensorFlow and Keras

- | 3.1. Main Features of TensorFlow
- | 3.2. Keras Basics
- | 3.3. AI with Keras

AI with Keras

- Linear/Logistic regression with Keras:



- The major difference between linear and logistic regression lies in the choice of activation function.

Keras Sequential API model:

```
In [83]: # Import the necessary classes.  
from keras.models import Sequential  
from keras.layers import Dense
```

```
In [84]: keras.__version__  
out[84]: '2.5.0'
```

```
In [86]: import pandas as pd  
import numpy as np  
import os  
import warnings  
import matplotlib.pyplot as plt  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import MinMaxScaler  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.optimizers import Adam, RMSprop, SGD  
warnings.filterwarnings('ignore')  
%matplotlib inline  
# Turn the warnings off.
```

Read in the data and explore:

- read

```
In [87]: df = pd.read_csv('data_boston.csv', header='infer', encoding = 'latin1')
X = df.drop(columns=['PRICE'])
y = df['PRICE']
```

```
In [88]: df.head(5)
```

Out[88]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

| Read in the data and explore:

- Scale the X data.

```
In [89]: scaler = MinMaxScaler()  
X = scaler.fit_transform(X)
```

- Split the data into training and testing

```
In [90]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)  
n_vars = X_train.shape[1]
```

| Define a Sequential API model:

- Add layers on a sequential objects.

```
In [91]: my_model1 = Sequential()
my_model1.add(Dense(input_dim = n_vars, units = 1, activation="linear"))
```

```
In [92]: my_model1.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	14

Total params: 14

Trainable params: 14

Non-trainable params: 0

**Line 91**

- Add an output layer for linear regression.

**Line 92**

- Summary of the model.

Define the hyperparameters and optimizer:

```
In [93]: n_epochs = 2000  
batch_size = 10  
learn_rate = 0.002
```

```
In [94]: my_optimizer=Adam(lr=learn_rate)  
my_model1.compile(loss = "mae", optimizer = my_optimizer, metrics=["mse"])
```

Line 93

- Hyperparameters.

Line 94

- Define the optimizer and then compile.

| Train the model and visualize the history:

- Train the model.

```
In [95]: my_summary = my_model1.fit(X_train, y_train, epochs=n_epochs, batch_size = batch_size,  
                                 validation_split = 0.2, verbose = 0)
```

```
In [96]: my_summary.history.keys()
```

```
Out[96]: dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```



Line 95

- verbose = 0 means no output. verbose = 1 to view the epochs.



Line 96

- View the keys.

| Train the model and visualize the history:

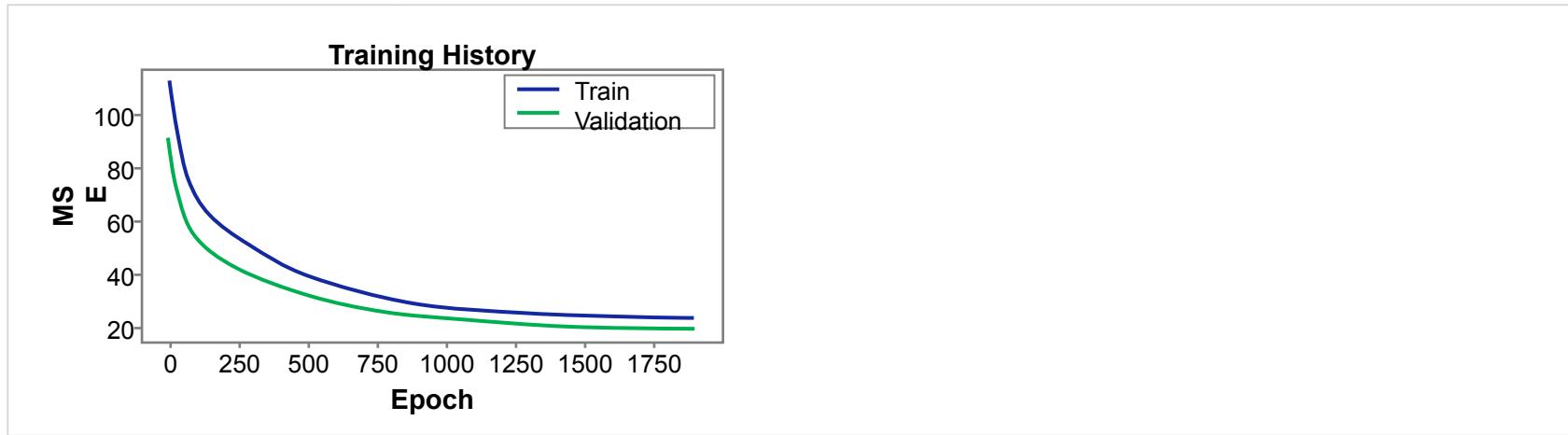
- Visualize the training history.

```
In [97]: 1 n_skip = 100
          2 plt.plot(my_summary.history['mse'][n_skip:], c="b")
          3 plt.plot(my_summary.history['val_mse'][n_skip:], c="g")
          4 plt.title('Training History')
          5 plt.ylabel('MSE')
          6 plt.xlabel('Epoch')
          7 plt.legend(['Train', 'Validation'], loc='upper right')
          8 plt.show()
```

 Line 97-1

- Skip the first few steps.

- I Train the model and visualize the history:
 - ▶ Visualize the training history.



Testing:

- Predict and test using a formula.

```
In [99]: y_pred = my_model1.predict(X_test)[:,0]
RMSE = np.sqrt(np.mean((y_test-y_pred)**2))
np.round(RMSE,3)

out[99]: 5.647
```

- Use the evaluate() method.

```
In [100]: MSE = my_model1.evaluate(X_test, y_test, verbose=0)[1]
RMSE = np.sqrt(MSE)
print("Test RMSE : {}".format(np.round(RMSE,3)))

Test RMSE : 5.647
```

Line 100

- Returns the 0 = loss value and 1 = metrics value.

| Keras Functional API model:

```
In [101]: from keras.models import Model  
from keras.layers import Input, Dense
```

| Define a Functional API model:

```
In [102]: 1 my_input = Input(shape=(n_vars,))
2 my_output = Dense(units=1,activation='linear')(my_input)
```

 **Line 102-1**

- Input layer

 **Line 102-2**

- Output layer

| Define a Functional API model:

- The model & The summary of the model.

```
In [103]: my_model2 = Model(inputs=my_input,outputs=my_output)
```

```
In [104]: my_model2.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 13)]	0
dense_1 (Dense)	(None, 1)	14
<hr/>		
Total params: 14		
Trainable params: 14		
Non-trainable params: 0		
<hr/>		

Define a Functional API model:

- Define the optimizer and then compile.

```
In [105]: my_optimizer=Adam(lr=learn_rate)
my_model2.compile(loss = "mae", optimizer = my_optimizer, metrics=["mse"])
```

Line 105

- Loss = MAE (L1) and Metrics = MSE (L2).

| Train the model and visualize the history:

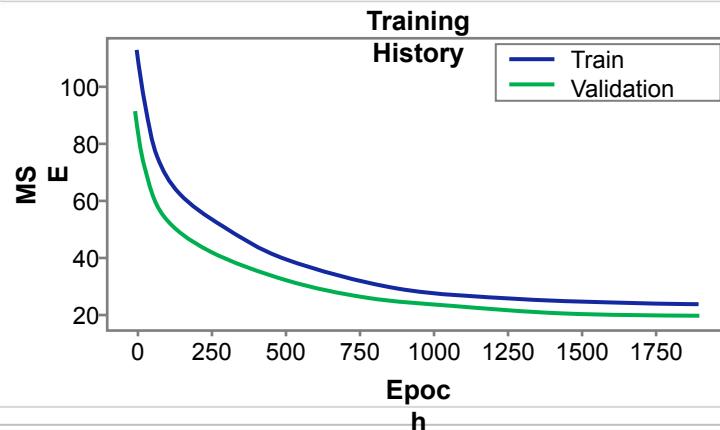
- › Train the model

```
In [107]: my_summary = my_model2.fit(X_train, y_train, epochs=n_epochs, batch_size = batch_size,
                                   validation_split = 0.2, verbose = 0)
```

| Train the model and visualize the history:

- › Visualize the training history.

```
In [108]: n_skip = 100
plt.plot(my_summary.history['mse'][n_skip:], c="b")
plt.plot(my_summary.history['val_mse'][n_skip:], c="g")
plt.title('Training History')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```



| Train the model and visualize the history:

- Use the evaluate() method.

```
In [109]: MSE = my_model2.evaluate(X_test, y_test, verbose=0)[1]
RMSE = np.sqrt(MSE)
print("Test RMSE : {}".format(np.round(RMSE,3)))
```

Test RMSE : 5.641

Line 109

- Returns the 0 = loss value and 1 = metrics value.

I Softmax regression with Keras:

```
In [113]: import os
import pandas as pd
import numpy as np
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam, RMSprop, SGD
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore') # Turn the warnings off.
%matplotlib inline
```

Softmax regression with Keras:

- Read in the data and prepare:

```
In [114]: data_raw = load_iris()  
X = data_raw['data']  
y0 = data_raw['target'].reshape(-1,1)  
  
In [117]: import tensorflow as tf
```

Softmax regression with Keras:

- One-hot-encoding for y

```
In [119]: y = tf.keras.utils.to_categorical(y0, num_classes=3)
```

- View as DataFrame

```
In [120]: header = ['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Species_0', 'Species_1', 'Species_2']
df = pd.DataFrame(np.concatenate([X,y],axis=1),columns=header)
df.head(5)
```

Out[120]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species_0	Species_1	Species_2
0	5.1	3.5	1.4	0.2	1.0	0.0	0.0
1	4.9	3.0	1.4	0.2	1.0	0.0	0.0
2	4.7	3.2	1.3	0.2	1.0	0.0	0.0
3	4.6	3.1	1.5	0.2	1.0	0.0	0.0
4	5.0	3.6	1.4	0.2	1.0	0.0	0.0

Softmax regression with Keras:

- Scale the X data.

```
In [121]: scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

- Split the data into training and testing.

```
In [122]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
n_vars = X_train.shape[1]
```

Softmax regression with Keras:

- Define a Sequential API model

```
In [123]: my_model = Sequential()
my_model.add(Dense(input_dim=n_vars, units = 3, activation="softmax"))
```

```
In [124]: my_model.summary()

Model: "sequential_2"

Layer (type)          Output Shape         Param #
=====
dense_2 (Dense)      (None, 3)            15
=====
Total params: 15
Trainable params: 15
Non-trainable params: 0
```

Line 123

- Add layers on a Sequential object.
- units = N# of output variables.

Line 124

- Summary of the model.

Softmax regression with Keras:

- Define the hyperparameters and optimizer:

```
In [125]: n_epochs = 500  
batch_size = 10  
learn_rate = 0.005
```

```
In [126]: my_optimizer=Adam(lr=learn_rate)  
my_model.compile(loss = "categorical_crossentropy", optimizer = my_optimizer, metrics=["accuracy"])
```

Softmax regression with Keras:

- Train the model and visualize the history

```
In [128]: my_summary = my_model.fit(X_train, y_train, epochs=n_epochs, batch_size = batch_size,  
                                validation_split = 0.2, verbose = 0)
```

```
In [129]: my_summary.history.keys()
```

```
Out[129]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Line 128

- Verbose = 0 means no output. Verbose = 1 to view the epochs.

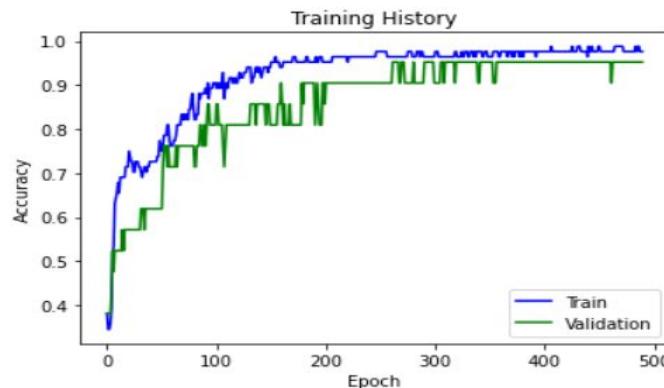
Line 129

- View the keys

Softmax regression with Keras:

- ▶ Visualize the training history

```
In [130]: n_skip = 10
plt.plot(my_summary.history['accuracy'][n_skip:], c="b")
plt.plot(my_summary.history['val_accuracy'][n_skip:], c="g")
plt.title('Training History')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



Softmax regression with Keras:

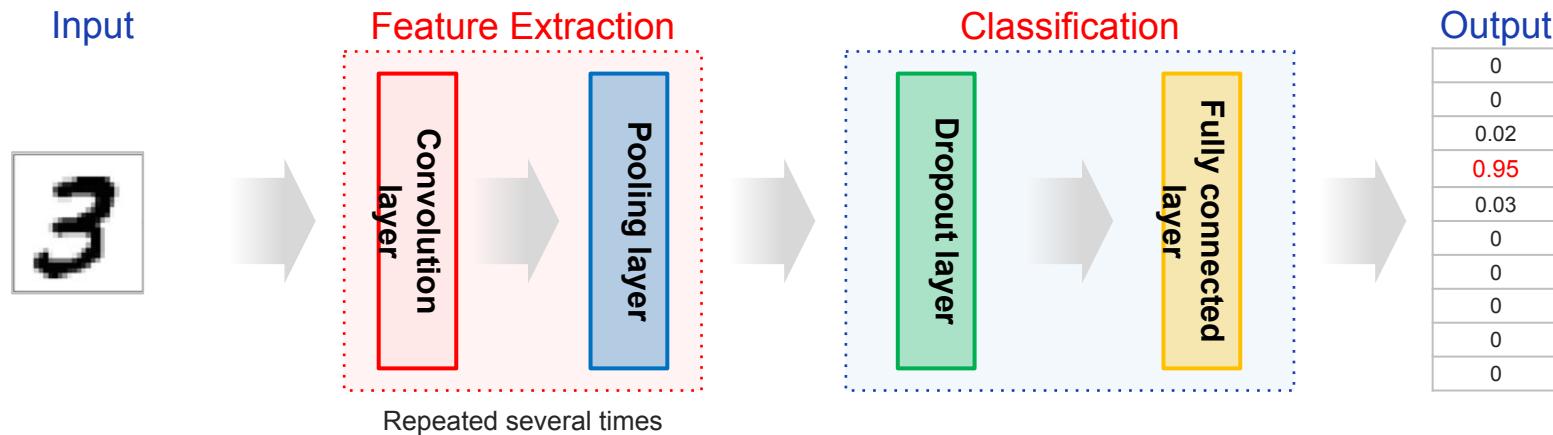
- Testing

```
In [131]: ACC = my_model.evaluate(X_test, y_test, verbose=0)[1]
print("Test Accuracy : {}".format(np.round(ACC,3)))
```

```
Test Accuracy : 0.956
```

| Diagram of a Convolutional Neural Network (CNN):

- The convolution and pooling layers are repeated several times.
- There is a fully connected layer just before the output.



| Read in the data

```
In [132]: import numpy as np
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from keras.datasets.mnist import load_data
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D

from keras.optimizers import Adam, RMSprop, SGD
warnings.filterwarnings('ignore')                      # Turn the warnings off.
%matplotlib inline
```

Read in the data

- Bring in the data

```
In [133]: (X_train, y_train), (X_test, y_test) = load_data()  
n_train_size = X_train.shape[0]
```

- View the shapes.

```
In [134]: print("-"*50)  
print("Training data X shape: {}".format(X_train.shape))  
print("Training data y shape: {}".format(y_train.shape))  
print("-"*50)  
print("Test data X shape: {}".format(X_test.shape))  
print("Test data y shape: {}".format(y_test.shape))  
print("-"*50)
```

```
-----  
Training data X shape: (60000, 28, 28)
```

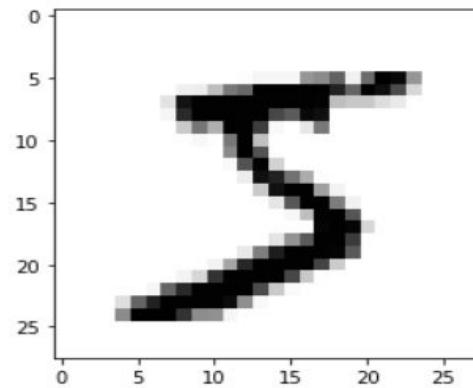
```
Training data y shape: (60000,)
```

```
-----  
Test data X shape: (10000, 28, 28)
```

```
Test data y shape: (10000,)
```

I Visualize the data:

```
In [135]: 1 i_image = 0
2 plt.imshow(X_train[i_image,:,:],cmap="Greys")
3 plt.show()
```



Line 135-1

- You may change this at will.

| Prepare the data:

- Scaling

```
In [136]: X_train = X_train/255  
X_test = X_test/255
```

- Reshaping

```
In [137]: X_train = X_train.reshape(-1,28,28,1)  
X_test = X_test.reshape(-1,28,28,1)
```

Prepare the data:

- One-hot-encoding.

```
In [139]: y = np.concatenate([y_train,y_test],axis=0)
y = tf.keras.utils.to_categorical(y,10)
y_train = y[:n_train_size,:]
y_test = y[n_train_size:,:]
```

Define a CNN model

```
In [140]: 1 drop_prob = 0.5
2 my_model = Sequential()
3 # 1st convolution + pooling.
4 my_model.add(Conv2D(input_shape=(28,28,1),filters=32,kernel_size=(5,5),padding='same',activation="relu"))
5 my_model.add(MaxPooling2D(pool_size=2))
6 # 2nd convolution + pooling.
7 my_model.add(Conv2D(filters=64,kernel_size=(5,5), padding='same',activation="relu"))
8 my_model.add(MaxPooling2D(pool_size=2))
9 # Flattened fully connected layer.
10 my_model.add(Flatten())
11 my_model.add(Dense(units = 1024, activation="relu"))
12 # Apply dropout.
13 my_model.add(Dropout(rate=drop_prob))
14 # Output layer.
15 my_model.add(Dense(units = 10, activation="softmax"))
```

 Line 140-1

- 1 channel of grayscale.

Define a CNN model

- View the summary

```
In [141]: my_model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 28, 28, 32)	832
<hr/>		
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
<hr/>		
conv2d_1 (Conv2D)	(None, 14, 14, 64)	51264
<hr/>		
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
<hr/>		
flatten (Flatten)	(None, 3136)	0
<hr/>		
dense_3 (Dense)	(None, 1024)	3212288
<hr/>		
dropout (Dropout)	(None, 1024)	0
<hr/>		
dense_4 (Dense)	(None, 10)	10250
<hr/>		
Total params: 3,274,634		
Trainable params: 3,274,634		
Non-trainable params: 0		

| Define the hyperparameters and optimizer

```
In [142]: n_epochs = 10  
batch_size = 200  
learn_rate = 0.001
```

```
In [143]: my_optimizer=Adam(lr=learn_rate)  
my_model.compile(loss = "categorical_crossentropy", optimizer = my_optimizer, metrics=["accuracy"])
```

| Train the model and visualize the history:

```
In [144]: my_summary = my_model.fit(X_train, y_train, epochs=n_epochs, batch_size = batch_size, validation_split = 0.2, verbose = 1)

Epoch 1/10
240/240 [=====] - 42s 162ms/step - loss: 0.4622 - accuracy: 0.8538 - val_loss: 0.0525 - val_accuracy: 0.9837
Epoch 2/10
240/240 [=====] - 39s 162ms/step - loss: 0.0607 - accuracy: 0.9819 - val_loss: 0.0533 - val_accuracy: 0.9835
Epoch 3/10
240/240 [=====] - 39s 164ms/step - loss: 0.0419 - accuracy: 0.9873 - val_loss: 0.0360 - val_accuracy: 0.9893
Epoch 4/10
240/240 [=====] - 39s 161ms/step - loss: 0.0296 - accuracy: 0.9908 - val_loss: 0.0441 - val_accuracy: 0.9872
Epoch 5/10
240/240 [=====] - 39s 160ms/step - loss: 0.0233 - accuracy: 0.9923 - val_loss: 0.0349 - val_accuracy: 0.9923
```

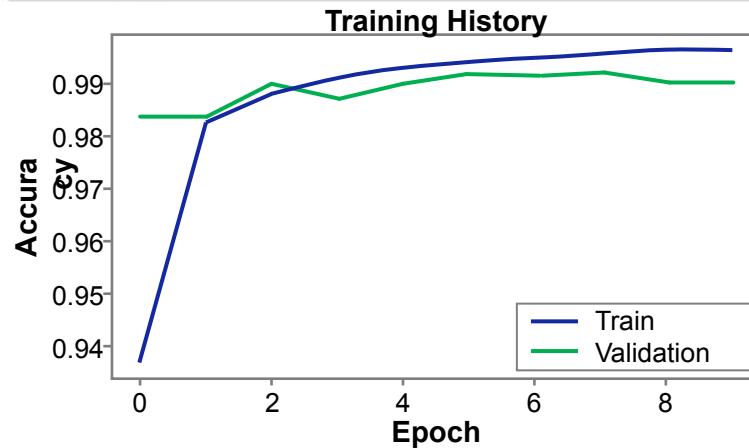


Line 144

- `verbose = 0` means no output. `verbose = 1` to view the epochs.

| Train the model and visualize the history:

```
In [145]: n_skip = 0
plt.plot(my_summary.history['accuracy'][n_skip:], c="b")
plt.plot(my_summary.history['val_accuracy'][n_skip:], c="g")
plt.title('Training History')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```



| Testing:

```
In [146]: ACC = my_model.evaluate(X_test, y_test, verbose=0)[1]
print("Test Accuracy : {}".format(np.round(ACC,3)))
```

```
Test Accuracy : 0.992
```

| Convolutional Neural Network with Keras (color images):

```
In [147]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from keras.datasets.cifar10 import load_data
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D

from keras.optimizers import Adam, RMSprop, SGD
warnings.filterwarnings('ignore') # Turn the warnings off.
%matplotlib inline
```

| Read in the data:

```
In [148]: (X_train, y_train), (X_test, y_test) = load_data()  
n_train_size = X_train.shape[0]  
  
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz  
170500096/170498071 [=====] - 11s 0us/step  
170508288/170498071 [=====] - 11s 0us/step
```

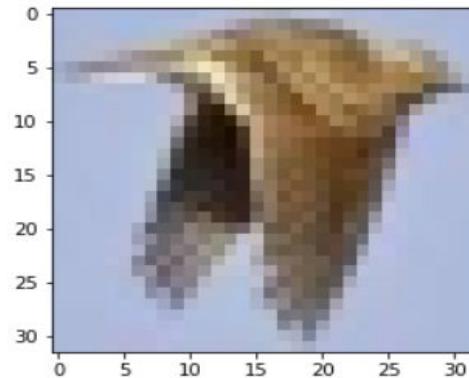
```
In [149]: print("-"*50)  
print("Training data X shape: {}".format(X_train.shape))  
print("Training data y shape: {}".format(y_train.shape))  
print("-"*50)  
print("Test data X shape: {}".format(X_test.shape))  
print("Test data y shape: {}".format(y_test.shape))  
print("-"*50)
```

```
-----  
Training data X shape: (50000, 32, 32, 3)  
Training data y shape: (50000, 1)
```

```
-----  
Test data X shape: (10000, 32, 32, 3)  
Test data y shape: (10000, 1)
```

Visualize the data:

```
In [150]: i_image = 123  
plt.imshow(X_train[i_image,:,:])  
plt.show()  
# You may change this at will.
```



Prepare the data

- Scaling

```
In [151]: X_train = X_train/255  
X_test = X_test/255
```

- Reshaping

```
In [152]: X_train = X_train.reshape(-1,32,32,3)  
X_test = X_test.reshape(-1,32,32,3)
```

- One-hot-encoding

```
In [154]: y = np.concatenate([y_train,y_test],axis=0)  
y = tf.keras.utils.to_categorical(y,10)  
y_train = y[:n_train_size,:]  
y_test = y[n_train_size:,:]
```

Define a CNN model

```
In [155]: drop_prob = 0.7
my_model = Sequential()
# 1st convolution + pooling.
my_model.add(Conv2D(input_shape=(32,32,3),filters=32,kernel_size=(5,5),padding='same',activation="relu"))
my_model.add(MaxPooling2D(pool_size=2))
# 2nd convolution + pooling.
my_model.add(Conv2D(filters=64,kernel_size=(5,5), padding='same',activation="relu"))
my_model.add(MaxPooling2D(pool_size=2))
# Flattened fully connected layer.
my_model.add(Flatten())
my_model.add(Dense(units = 1024, activation="relu"))
# Apply dropout.
my_model.add(Dropout(rate=drop_prob))
# Output layer.
my_model.add(Dense(units = 10, activation="softmax"))
```



Line 155

- 3 channels of color

Define a CNN model

```
In [156]: my_model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 32, 32, 32)	2432
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	51264
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_5 (Dense)	(None, 1024)	4195328
dropout_1 (Dropout)	(None, 1024)	0
dense_6 (Dense)	(None, 10)	10250
<hr/>		
Total params: 4,259,274		
Trainable params: 4,259,274		
Non-trainable params: 0		

| Define the hyperparameters and optimizer:

```
In [157]: n_epochs = 20  
batch_size = 20  
learn_rate = 0.0001
```

```
In [158]: my_optimizer=Adam(lr=learn_rate)  
my_model.compile(loss = "categorical_crossentropy", optimizer = my_optimizer, metrics=["accuracy"])
```

Train the model and visualize the history:

```
In [159]: my_summary = my_model.fit(X_train, y_train, epochs=n_epochs, batch_size = batch_size, validation_split = 0.3, verbose = 1)

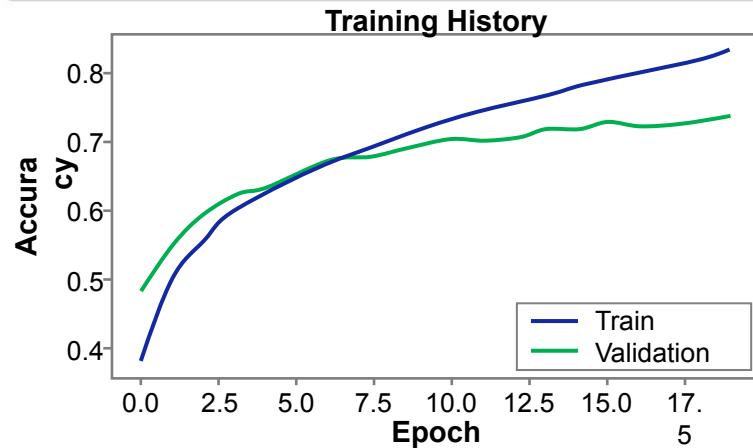
Epoch 1/20
1750/1750 [=====] - 78s 44ms/step - loss: 1.8939 - accuracy: 0.3080 - val_loss: 1.4602 - val_accuracy: 0.4831
Epoch 2/20
1750/1750 [=====] - 75s 43ms/step - loss: 1.4393 - accuracy: 0.4840 - val_loss: 1.2612 - val_accuracy: 0.5509
Epoch 3/20
1750/1750 [=====] - 75s 43ms/step - loss: 1.2723 - accuracy: 0.5427 - val_loss: 1.1496 - val_accuracy: 0.5935
Epoch 4/20
1750/1750 [=====] - 76s 43ms/step - loss: 1.1506 - accuracy: 0.5904 - val_loss: 1.0775 - val_accuracy: 0.6227
Epoch 5/20
1750/1750 [=====] - 75s 43ms/step - loss: 1.0781 - accuracy: 0.6181 - val_loss: 1.0373 - val_accuracy: 0.6330
```

Line 159

- verbose = 0 means no output. verbose = 1 to view the epochs.

Train the model and visualize the history:

```
In [160]: n_skip = 0
plt.plot(my_summary.history['accuracy'][n_skip:], c="b")
plt.plot(my_summary.history['val_accuracy'][n_skip:], c="g")
plt.title('Training History')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.show()
```

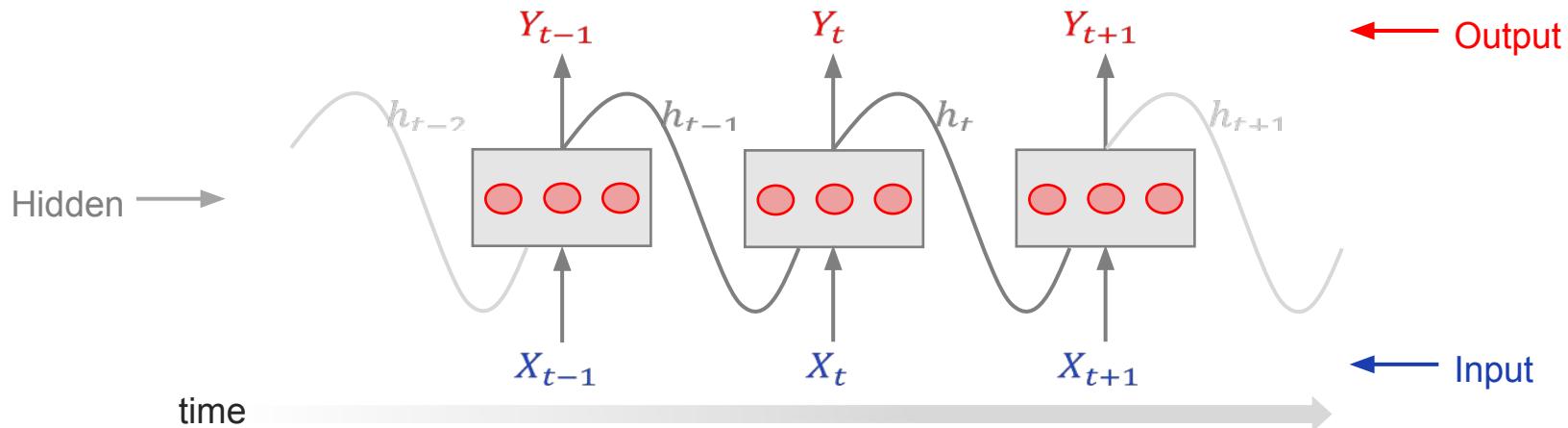


Testing:

```
In [161]: ACC = my_model.evaluate(X_test, y_test, verbose=0)[1]
print("Test Accuracy : {}".format(np.round(ACC,3)))
```

```
Test Accuracy : 0.732
```

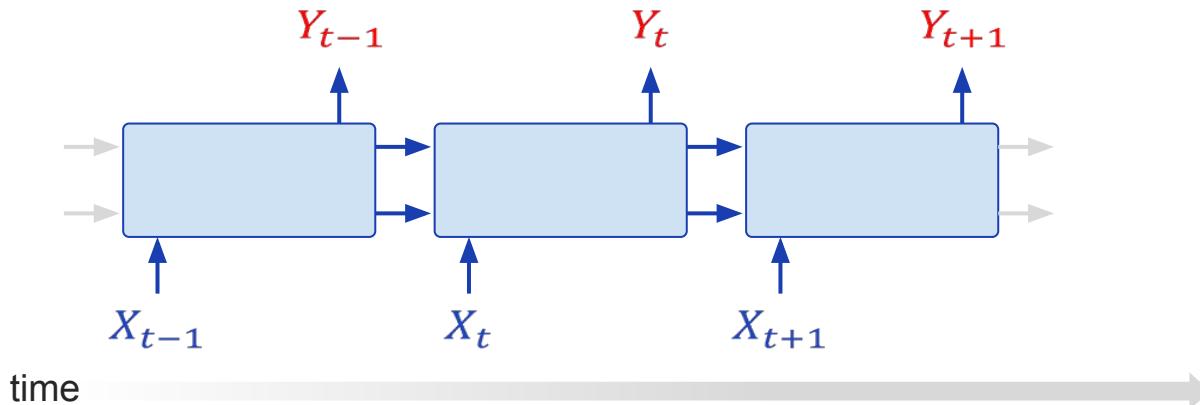
I Recurrent Neural Network (RNN)



- This is a “Sequence in and Sequence out” model.

I Long Short Term Memory (LSTM) network:

- LSTM cells can form a sequence just like the regular RNN cells.
- The following is a “Sequence in and Sequence out” model.



| Read in the data and explore:

```
In [164]: df = pd.read_csv('data_time_series.csv', header='infer', encoding = 'latin1')
n_time_steps = df.shape[0]
print(df.shape)

(41, 1)
```

▶ Display the dataframe

```
In [165]: df.head()
```

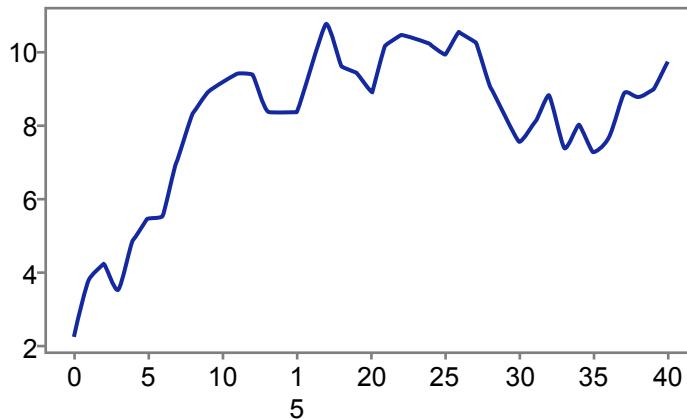
Out[165]:

	Value
0	2.26483
1	3.80588
2	4.21088
3	3.48790
4	4.86365

| Read in the data and explore:

- › Visualize the time series.

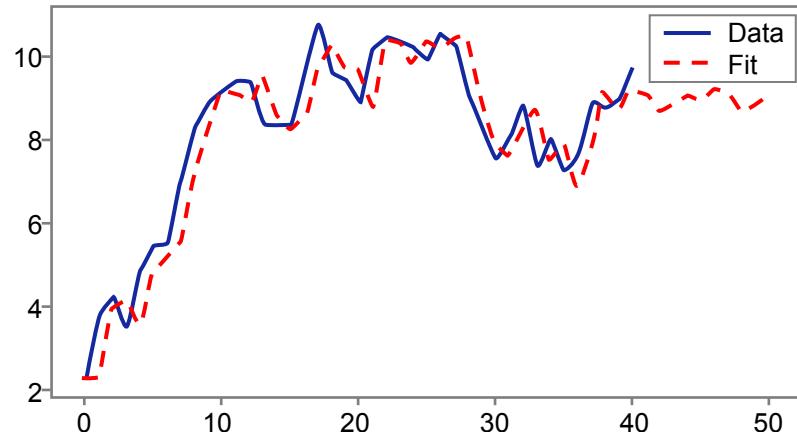
```
In [166]: plt.plot(df.Value, c="b", linewidth=2, linestyle="--")
plt.show()
```



| Read in the data and explore:

- Multiplicative exponential smoothing model.

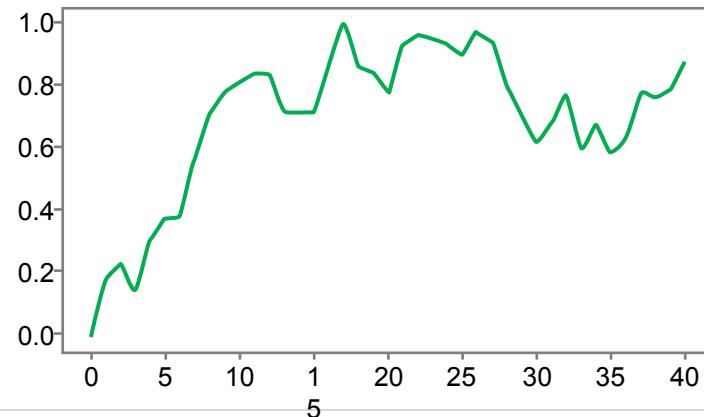
```
In [167]: model = ExponentialSmoothing(df['Value'][0:40], seasonal='mul',seasonal_periods=6).fit()
pred = model.predict(start= 0, end=50)
plt.plot(df,c='b',linewidth=2, linestyle="--", label="Data")
plt.plot(pred,c='r',linewidth=2, linestyle="--", label="Fit")
plt.legend()
plt.tight_layout()
plt.show()
```



Recurrent neural network (RNN):

- Pre-processing:
- Scaling (please pay attention to the vertical scale of the plot)

```
In [168]: scaler = MinMaxScaler()  
ts_scaled = scaler.fit_transform(df)  
plt.plot(ts_scaled,c = "g", linewidth=2, linestyle="--")  
plt.show()
```



Recurrent neural network (RNN):

- Reshaping (batch_size, time series length, n_input)

```
In [169]: ts_scaled_2 = ts_scaled.reshape(1,-1,1)
```

| Do the necessary definitions:

- Hyperparameters

```
In [170]: 1 n_epochs = 1001
            2 batch_size = 1
            3 learn_rate = 0.0002
```

 **Line 170-2**

- There is only 1 time series data. No other choice but 1

| Do the necessary definitions:

- Hyperparameters

```
In [171]: 1 n_input = 1
            2 n_neurons = 100
            3 n_output = 1
```

 **Line 171-1**

- Scalar input

 **Line 171-2**

- N# of neurons per layer

 **Line 171-3**

- Scalar output

Recurrent neural network (RNN):

- RNN or LSTM network

```
In [172]: my_model = Sequential()
my_model.add(SimpleRNN(units=n_neurons, return_sequences=True, input_shape=(None, n_input)))      # RNN.
my_model.add(TimeDistributed(Dense(units=n_output, activation="linear")))    # !           # LSTM.
```

Line 172

- Return sequences = True: means “Sequence to Sequence”.
- Input shape = (None, n inputs) : variable length of the time series.

I Recurrent neural network (RNN):

- View the summary.

```
In [173]: my_model.summary()
```

```
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, None, 100)	10200
time_distributed (TimeDistr)	(None, None, 1)	101
<hr/>		
Total params: 10,301		
Trainable params: 10,301		
Non-trainable params: 0		

Recurrent neural network (RNN):

- Define the optimizer and compile

```
In [174]: my_optimizer=Adam(lr=learn_rate)
my_model.compile(loss = "mse", optimizer = my_optimizer, metrics=["mse"])
```

Recurrent neural network (RNN):

- Train the model:

```
In [176]: my_summary = my_model.fit(ts_scaled_2[:, :-1, :], ts_scaled_2[:, 1:, :], epochs=n_epochs,  
batch_size = batch_size, verbose = 0)
```

```
In [177]: my_summary.history.keys()
```

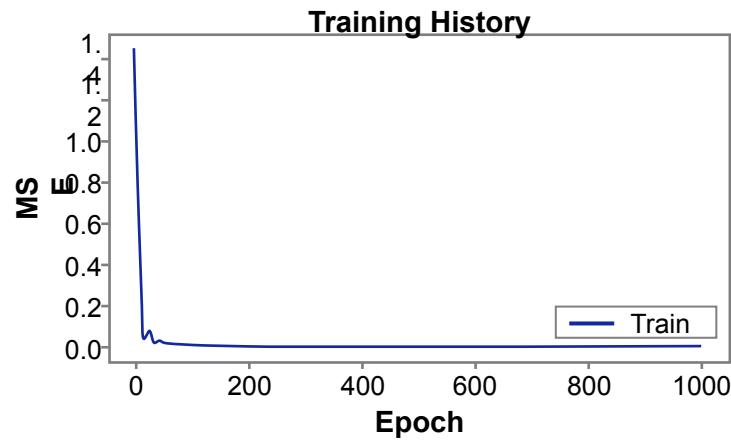
```
out[177]: dict_keys(['loss', 'mse'])
```

Line 176

- No validation.
- CAUTION: y is X shifted by +1

Recurrent neural network (RNN):

```
In [178]: plt.plot(my_summary.history['mse'], c="b")
plt.title('Training History')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='lower right')
plt.show()
```



Recurrent neural network (RNN):

- Predict the future

```
In [180]: 1 n_ts_seed = 5  
           2 n_predict_time_steps = 55
```

Line 180-1

- Seed length

Line 180-2

- Prediction length

Recurrent neural network (RNN):

- Prediction loop

```
In [181]: 1 ts_seed = ts_scaled[0:n_ts_seed]
2 for i in range(n_predict_time_steps):
3     X = ts_seed.reshape(1,-1,1)
4     y_pred = my_model.predict(X)
5     y_last= y_pred[0,-1,0]
6     ts_seed = np.concatenate((ts_seed, np.array([y_last]).reshape(1,1)), axis=0)
```

 **Line 181-3**

- Reshape

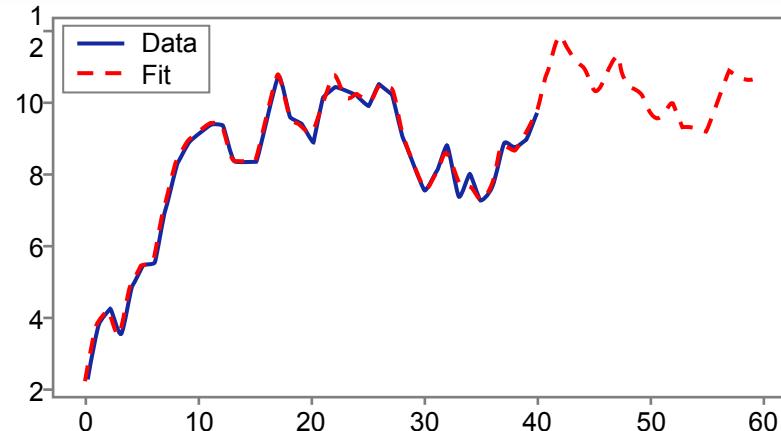
 **Line 181-5**

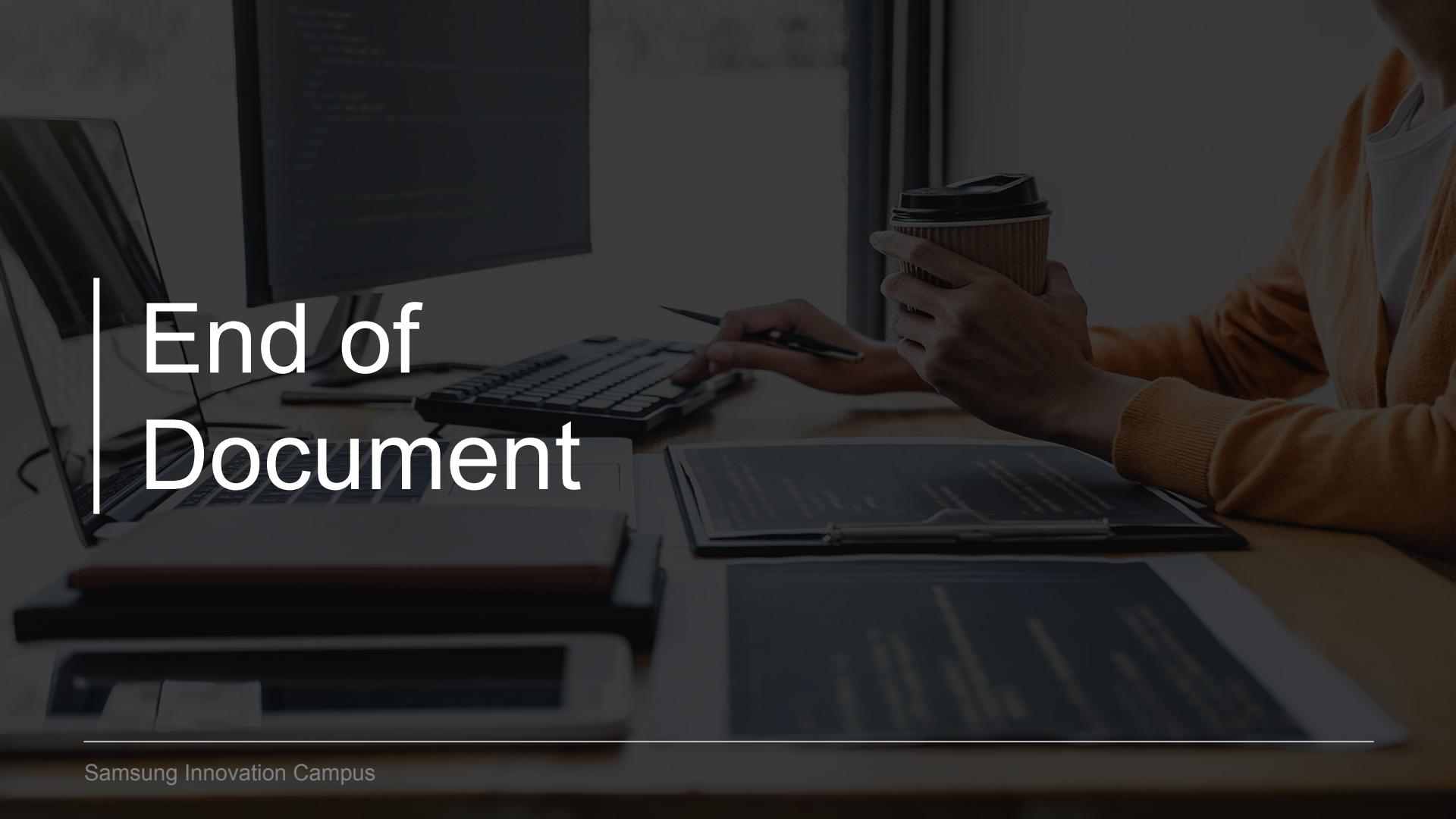
- The last output is the predicted y

Recurrent neural network (RNN):

- Do the inverse transform and show the times series.

```
In [182]: ts = scaler.inverse_transform(ts_seed)
plt.plot(df.Value,c='b',linewidth=2, linestyle="-",label="Data")
plt.plot(ts,c='r',linewidth=2, linestyle="--",label="Fit")
plt.legend()
plt.tight_layout()
plt.savefig("out.png")
plt.show()
```



A photograph of a person's hands and arms as they work at a desk. They are wearing a light-colored long-sleeved shirt and are holding a brown paper coffee cup with a black lid in their left hand. Their right hand is on a black computer keyboard. In front of them is a stack of papers and a clipboard with a pen resting on it. A computer monitor is visible in the background, displaying some text. The scene is lit from the side, creating shadows and highlights.

End of Document



Together for Tomorrow!
Enabling People

Education for Future Generations

©2022 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this book other than the curriculum of Samsung Innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.