

COMP2211 Exploring Artificial Intelligence

Artificial Neural Network - Neurons

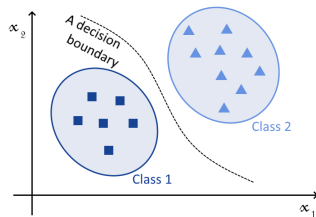
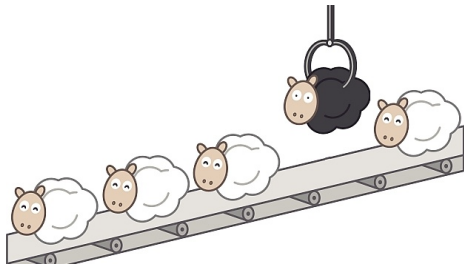
Huiru Xiao

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

Classification

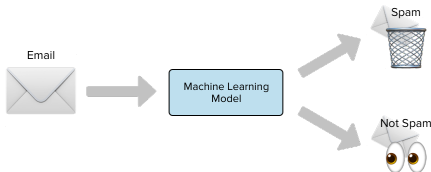
Recap: What is Classification?

- **Classification** is the process of **predicting the class of given data**.
- **Classes** are also called as **labels/categories**.

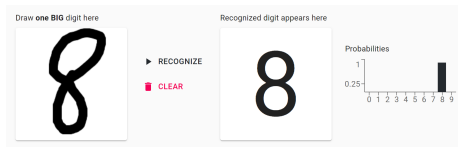


Classification Examples

- Spam detection in email service: 2 classes - spam and not spam

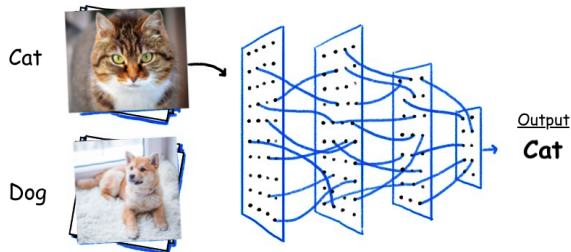


- Handwritten digit recognition: 10 classes - 0, 1, 2, ..., 9



Classification Examples

- **Images classification:** 3 classes - cat, dog, none of them



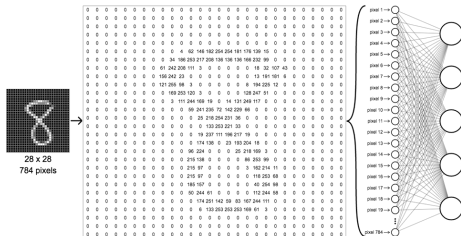
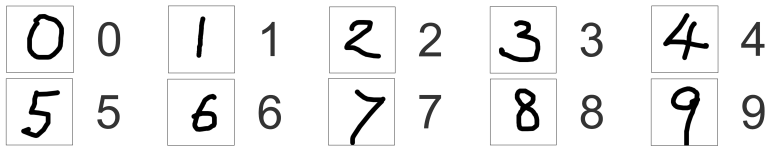
Why Image Classification?

- **Image recognition and classification** is a **subdomain of computer vision**. It is an algorithm that looks at an image and **assigns it a label** from a collection of predefined labels or categories (e.g., a dog, a cat).
- Image classification and recognition are **vital components in robotics**, such as autonomous vehicles or domestic robots.
- It is also important in **security systems such as face recognition**, image search engines such as Google or Bing image search, and medical imaging such as cancer detection.



Handwritten Digits Recognition using MLP

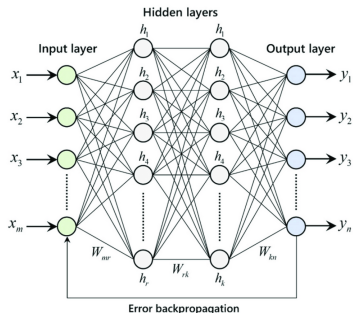
We will build an **Artificial Neural Network** to **recognize/classify** handwritten digits.



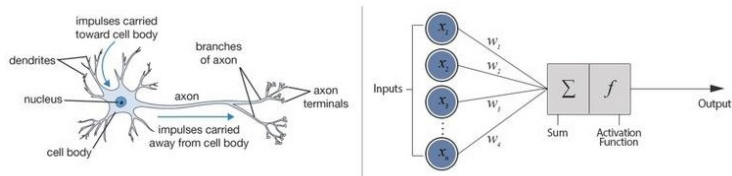
Artificial Neural Network and Artificial Neuron

Artificial Neural Network

- Artificial neural networks (ANN) are one of the most powerful artificial intelligence and machine learning algorithms.
- An ANN is considered a universal function approximator that transforms inputs into outputs.
- As the name suggests, it draws inspiration from neurons in our brain and the way they are connected.



Biological Neuron vs Artificial Neuron



Biological Neuron	Artificial Neuron
Dendrites	Inputs
Cell Nucleus (Computation unit)	Node
Axon	Output
Synapse	Weight

- **Neuron Node** = a linear function & an activation function

Artificial Neuron

What is an Artificial Neuron?

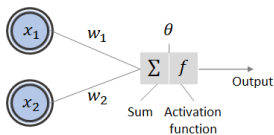
- An **artificial neuron** is a **simple biological neuron model** in an artificial neural network.
- It performs certain **calculations to detect input data capabilities**.
- One of the most simplest models of neuron is called **perceptron** (or **threshold logic unit**).

Neuron = a **linear function** & an **activation function**

- **Linear function:** a linear transformation, the weighted sum of input and bias.
- **Activation function:** a mathematical operation, deciding whether a perceptron/neuron should be activated or not.

Note: The primary role of the activation function is to transform the summed weighted input from the node into an output value to be fed to the next hidden layer or as output.

An Example of Neuron



$$output = f(w_1 \times x_1 + w_2 \times x_2 + \theta)$$

Suppose $x_1 = 3$, $x_2 = 5$, $w_1 = 0.2$, $w_2 = 0.3$, $\theta = 0.3$, $f(x) = \frac{1}{1+e^{-x}}$

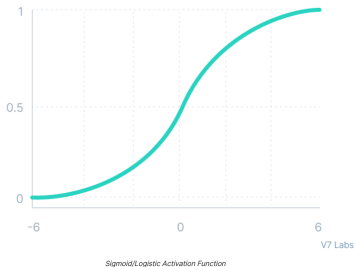
$$\begin{aligned} output &= f(0.2 \times 3 + 0.3 \times 5 + 0.3) \\ &= f(2.4) \\ &= \frac{1}{1 + e^{-2.4}} \\ &= 0.91683 \end{aligned}$$

Sigmoid/Logistic Activation Function

Sigmoid/Logistic Activation Function: $f(x) = \frac{1}{1+e^{-x}}$

- Taking any real value as input and outputs values in the range of 0 to 1.
- The larger the input, the closer the output value will be to 1, whereas the smaller the input, the closer the output will be to 0.

Sigmoid / Logistic



- One of the most widely used activation functions.
- Commonly used for models where we have to predict the probability as an output.
- Differentiable with a smooth gradient.
- The limitations will be discussed later after we learn MLP.

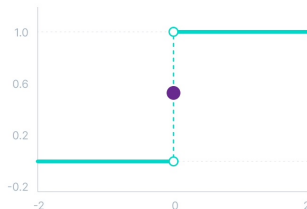
Binary Step Activation Function

Binary Step Activation Function:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

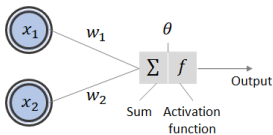
- When using the binary step activation function, a neuron is called **Perceptron** (or **threshold logic unit**).

Binary Step Function



- Cannot provide multi-value outputs, thus cannot be used for multi-class problems.
- The gradient of the step function is zero, which causes a hindrance in the backpropagation process in MLP (will be covered later).
- Suitable to represent some logical operations in a neuron.

Perceptron - Logical AND Example



- Suppose we will work on a problem of AND logical operation. The truth table of logical AND is as follows.

x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

- Assume the weights and bias are randomly generated, say $w_1 = 0.1$, $w_2 = 0.5$, $\theta = -0.8$. Also, we set learning rate $\eta = 0.2$.
- Activation function is binary step function:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

Perceptron Learning Rules

- x_1 and x_2 are inputs
 - θ is the bias
 - w_1 and w_2 are weights
 - O is the computed output
 - T is the target (label)
 - Δw_1 is the change of w_1
 - Δw_2 is the change of w_2
 - $\Delta \theta$ is the change of θ
 - η is the learning rate
- 1 If the **output is correct** (i.e., T is the same as O), the weights w_1 and w_2 are **not updated**.
 - 2 If the **output is incorrect** (i.e., T is different to O), the weights w_1 and w_2 are **updated** according to the following rules such that the output of the perceptron for the new weights is closer to T .

$$\Delta w_i = \eta(T - O)x_i$$

$$\Delta \theta = \eta(T - O)$$

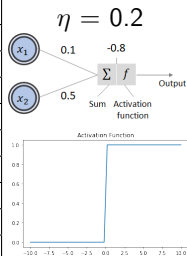
$$w_i = w_i + \Delta w_i$$

$$\theta = \theta + \Delta \theta$$

where $i \in \{1, 2\}$.

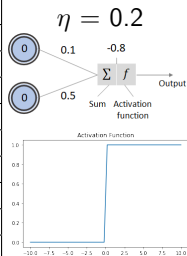
Initial

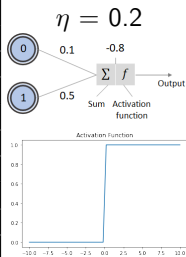
x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8



Round 1 - Step 1

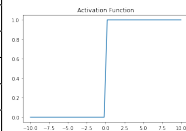
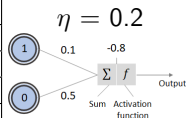
x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8



[illegible]

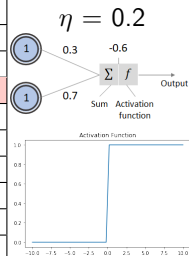
Round 1 - Step 3

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8



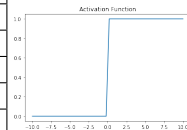
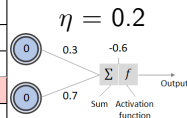
Round 1 - Step 4

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6



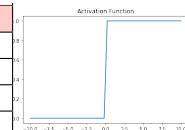
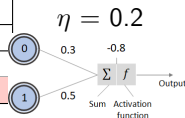
Round 2 - Step 1

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6



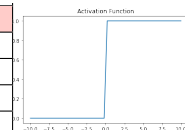
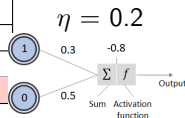
Round 2 - Step 2

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6
0	1	0	1	0	0.3	-0.2	0.5	-0.2	-0.8



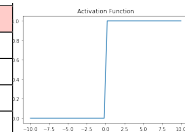
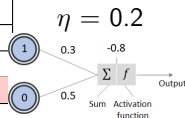
Round 2 - Step 3

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6
0	1	0	1	0	0.3	-0.2	0.5	-0.2	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8



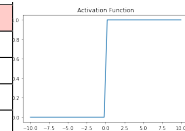
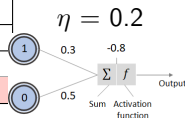
Round 2 - Step 4

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6
0	1	0	1	0	0.3	-0.2	0.5	-0.2	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8
1	1	1	1	0	0.3	0	0.5	0	-0.8



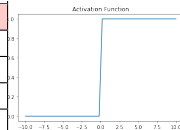
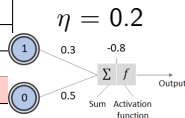
Round 3 - Step 1

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6
0	1	0	1	0	0.3	-0.2	0.5	-0.2	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8
1	1	1	1	0	0.3	0	0.5	0	-0.8
0	0	0	0	0	0.3	0	0.5	0	-0.8



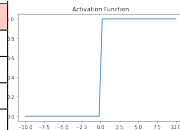
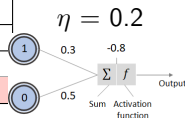
Round 3 - Step 2

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6
0	1	0	1	0	0.3	-0.2	0.5	-0.2	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8
1	1	1	1	0	0.3	0	0.5	0	-0.8
0	0	0	0	0	0.3	0	0.5	0	-0.8
0	1	0	0	0	0.3	0	0.5	0	-0.8



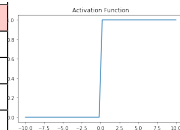
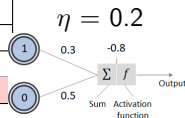
Round 3 - Step 3

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6
0	1	0	1	0	0.3	-0.2	0.5	-0.2	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8
1	1	1	1	0	0.3	0	0.5	0	-0.8
0	0	0	0	0	0.3	0	0.5	0	-0.8
0	1	0	0	0	0.3	0	0.5	0	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8



Round 3 - Step 4

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	0	0	0	0.1	0	0.5	0	-0.8
1	0	0	0	0	0.1	0	0.5	0	-0.8
1	1	1	0	0.2	0.3	0.2	0.7	0.2	-0.6
0	0	0	0	0	0.3	0	0.7	0	-0.6
0	1	0	1	0	0.3	-0.2	0.5	-0.2	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8
1	1	1	1	0	0.3	0	0.5	0	-0.8
0	0	0	0	0	0.3	0	0.5	0	-0.8
0	1	0	0	0	0.3	0	0.5	0	-0.8
1	0	0	0	0	0.3	0	0.5	0	-0.8
1	1	1	1	0	0.3	0	0.5	0	-0.8



Perceptron Implementation from Scratch

```
import math # Import math module

class Perceptron:
    def __init__(self):
        """ Perceptron initialization """
        self.w = [0.1,0.5]      # Weights
        self.theta = -0.8       # Bias
        self.learningRate = 0.2 # Eta

    def response(self,x):
        """ Perceptron output """
        # Calculate weighted sum
        y = x[0] * self.w[0] + x[1] * self.w[1] + self.theta
        # If weighted sum >= 0, return 1. Otherwise return 0
        if y >= 0:
            return 1
        else:
            return 0
```

Perceptron Implementation from Scratch

```
def updateWeights(self,x,iterError):    # iterError:  $T - 0$ 
    """ Weights update """
    #  $w_i = w_i + \eta * (T-0) * x_i$ 
    self.w[0] += self.learningRate * iterError * x[0]
    self.w[1] += self.learningRate * iterError * x[1]

def updateBias(self,iterError):
    """ Bias update """
    #  $\theta = \theta + \eta * (T-0)$ 
    self.theta += self.learningRate * iterError

def train(self,data):
    """ Training """
    learned = True    # Should perform training
    round = 0          # Initialize round to 0
```

Perceptron Implementation from Scratch

```

while learned:                                # While learned
    totalError = 0.0                          # Initialize
    for x in data:                            # For each data
        r = self.response(x)                 # Calculate output
        if x[2] != r:                        # If different
            roundError = x[2] - r             # Error
            self.updateWeights(x,roundError)  # Update weights
            self.updateBias(roundError)        # Update bias
            totalError += abs(roundError)      # Update total error
    round += 1

if math.isclose(totalError, 0) or round >= 100:
    print("Total number of rounds (epochs): ", round)
    print("Final weights: ", self.w)
    print("Final bias: ", self.theta)
    learned = False

```


Perceptron Implementation from Scratch

""" Main function """

```
perceptron = Perceptron()
```

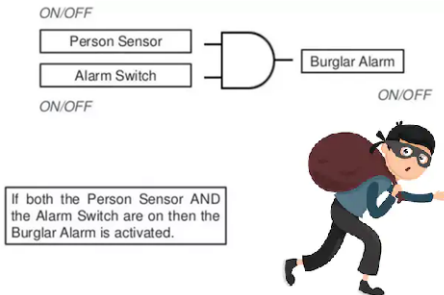
```
trainset = [[0,0,0], [0,1,0], [1,0,0], [1,1,1]]
```

```
perceptron.train(trainset)
```

Create Perceptron object

Define training set

Perform training



Perceptron Implementation using Scikit-Learn

```
import numpy as np # Import NumPy
from sklearn.linear_model import Perceptron # Import Perceptron class from Scikit-Learn

inputs = np.array([[0,0], [0,1], [1,0], [1,1]]) # Inputs
outputs = np.array([0, 0, 0, 1]) # Expected outputs

# Create and fit a perceptron model
# Set learning rate (eta0)
model = Perceptron(eta0=0.2)
model.fit(inputs, outputs)

# Use the trained model to predict the outputs
predicted_outputs = model.predict([[0,0], [1,0], [1,1], [0,1]])
print(predicted_outputs) # Print the predicated outputs

print(model.coef_) # Print the final weights
print(model.intercept_) # Print the bias
```

Stopping Rules

- Use maximum training time

The training may go a bit beyond the specified time limit in order to complete the current cycle.

- The maximum number of training cycles allowed

If the maximum number of cycles is exceeded, then training stops.

- Use minimum accuracy

Training will continue until the specified accuracy is attained.



Terminologies - Learning and Epoch

- **Learning** is the **process of updating weights** in the perceptron.
 - We set weights w_1 to 0.1, w_2 to 0.5 initially, but it causes some errors. Then, we update the weight values to 0.5 and predict all instances correctly.
 - The whole process takes 4 rounds/epochs.
- **Epoch** refers to **one cycle through the full training dataset**.

Observation

According to the results of the perceptron learning procedure, $w_1 = 0.3, w_2 = 0.5, \theta = -0.8$. The relationship between the inputs, i.e., x_1, x_2 , and the output y is

$$y = \begin{cases} 0 & \text{if } 0.3x_1 + 0.5x_2 - 0.8 < 0 \\ 1 & \text{otherwise} \end{cases}$$
$$= \begin{cases} 0 & \text{if } 3x_1 + 5x_2 < 8 \\ 1 & \text{otherwise} \end{cases}$$

Decision Boundary

$$y = \begin{cases} 0 & \text{if } 3x_1 + 5x_2 < 8 \\ 1 & \text{otherwise} \end{cases}$$

The straight line defined by $3x_1 + 5x_2 = 8$ is called **Decision Boundary** of the perceptron.

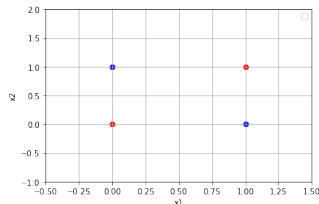
Decision Boundary

In a binary classification problem, a decision boundary or decision surface is a hypersurface that partitions the underlying vector space into two sets, one for each class. The classifier will classify all the points on one side of the decision boundary as belonging to one class and all those on the other side as belonging to the other class.

Problem

- Question: Can we apply the same perceptron learning procedure for the XOR gate, which has the truth table on the right? If so, show all the steps. If not, explain why.
- Answer: No, a perceptron cannot implement XOR. The reason is that the labels in XOR are not linearly separable, i.e. we cannot draw a straight line to separate the points (0,0),(1,1) from the points (0,1),(1,0).

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0



To solve the problem, we need a multi-layer perceptron.

Practice Problem

- Please apply the perceptron learning procedure for the OR gate. The truth table of logical OR is as follows.

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	1

- Assume the weights and bias are randomly generated, say $w_1 = 0.1$, $w_2 = 0.5$, $\theta = -0.8$. Also, we set learning rate $\eta = 0.2$.
- Activation function is

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



Initial, Round 1, Round 2, & Round 3

x_1	x_2	T	O	Δw_1	w_1	Δw_2	w_2	$\Delta \theta$	θ
-	-	-	-	-	0.1	-	0.5	-	-0.8
0	0	0	0	0	0.1	0	0.5	0	-0.8
0	1	1	0	0	0.1	0.2	0.7	0.2	-0.6
1	0	1	0	0.2	0.3	0	0.7	0.2	-0.4
1	1	1	1	0	0.3	0	0.7	0	-0.4
0	0	0	0	0	0.3	0	0.7	0	-0.4
0	1	1	1	0	0.3	0	0.7	0	-0.4
1	0	1	0	0.2	0.5	0	0.7	0.2	-0.2
1	1	1	1	0	0.5	0	0.7	0	-0.2
0	0	0	0	0	0.5	0	0.7	0	-0.2
0	1	1	1	0	0.5	0	0.7	0	-0.2
1	0	1	1	0	0.5	0	0.7	0	-0.2
1	1	1	1	0	0.5	0	0.7	0	-0.2

The training converges in 3 epochs if the initial weights are $w_1 = 0.1$, $w_2 = 0.5$, initial bias is $\theta = -0.8$ and learning rate is $\eta = 0.2$. The final weights and bias are: $w_1 = 0.5$, $w_2 = 0.7$, $\theta = -0.2$.

That's all!
Any question?



**Welcome
Back!**

Acknowledgments

- The lecture notes are developed based on Dr. Desmond Tsoi's lecture slides.