

# DEEP LEARNING

## Lecture 4

### Perceptron and Multilayers Perceptron

*Instructor: Zafar Iqbal*

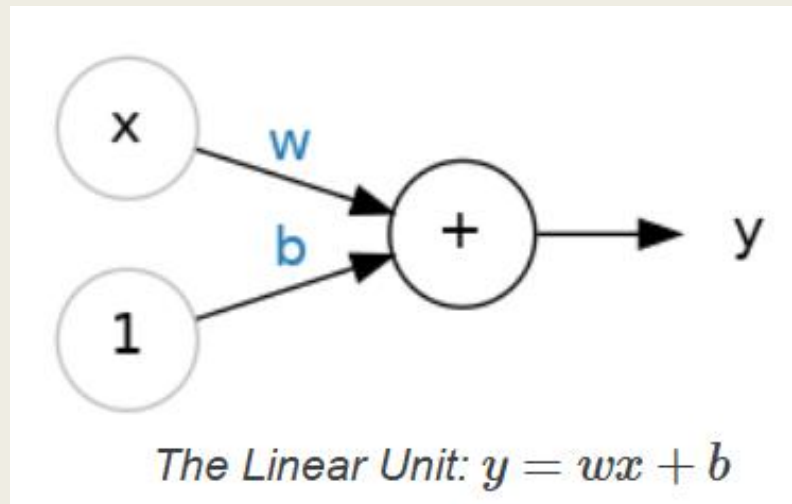
# Agenda

- Topics:
  - The linear unit
  - Perceptron
  - Multilayer Perceptron (MLP)
  - Example



# The Linear Unit

- A neuron is also called a unit.
- It receives input, performs a calculation, and produces an output.
- With one input, a neuron can be shown in a simple diagram.



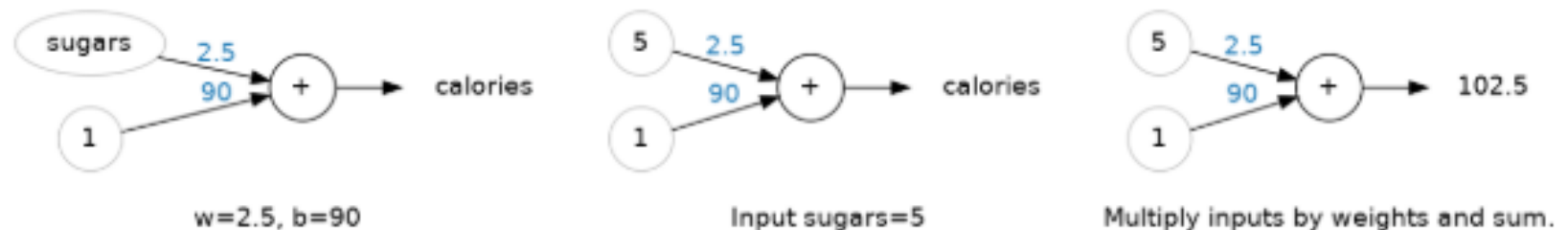
- The input to the neuron is called  $x$ .
- The input connection has a weight, labeled  $w$ .
- When a value flows through this connection, it gets multiplied by the weight  $\rightarrow$  the neuron receives  $w \times x$ .

# The Linear Unit

- Neural networks learn by adjusting these weights ( $w$ ) during training.
- There is also a special value called the bias, labeled  $b$ .
  - It doesn't depend on any input data.
  - It's included by connecting a constant 1 to the neuron.
  - So the neuron receives  $b$  (because  $1 \times b = b$ ).
- The bias allows the neuron to shift the output, even if the input is zero.
- The neuron adds up everything it receives:
  - $y = w \times x + b$
- This final value  $y$  is the neuron's output (also called the activation).

# Example – The Linear Unit as a Model

- Individual neurons usually work as part of a larger neural network.
- However, studying a single neuron model is a useful starting point.
- A single neuron is essentially a linear model.
- Example: using the 80 Cereals dataset.
  - Input: 'sugars' (grams of sugar per serving).
  - Output: 'calories' (calories per serving).



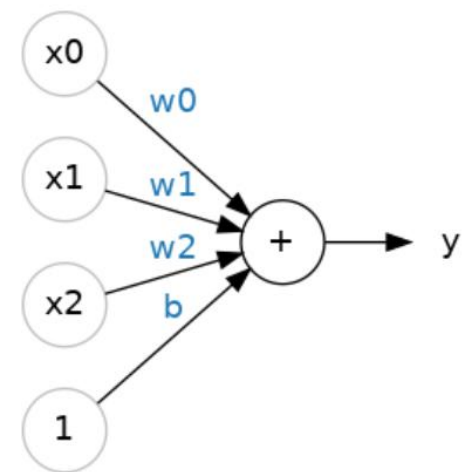
*Computing with the linear unit.*

# Example – The Linear Unit as a Model

- After training, suppose we find:
  - Weight ( $w$ ) = 2.5
  - Bias ( $b$ ) = 90
- To estimate calories for a cereal with 5 grams of sugar:
  - Use the formula:  $y = w \times x + b$
  - Calculation:  $y = 2.5 \times 5 + 90 = 102.5$
- So, the estimated calorie content is **102.5 calories per serving**.

# Multiple Inputs

- The 80 Cereals dataset includes many features besides just 'sugars'.
- To include more features (like fiber or protein), we can:
  - Add more input connections to the neuron—one for each feature.
  - $x_0, x_1, x_2$  are the inputs (e.g., sugar, fiber, protein)
  - $w_0, w_1, w_2$  are the corresponding weights
  - $b$  is the bias
- Each input is multiplied by its own weight, and all the products are added together along with the bias.
- The formula for the output becomes:
- $Y = w_0x_0 + w_1x_1 + w_2x_2 + b$



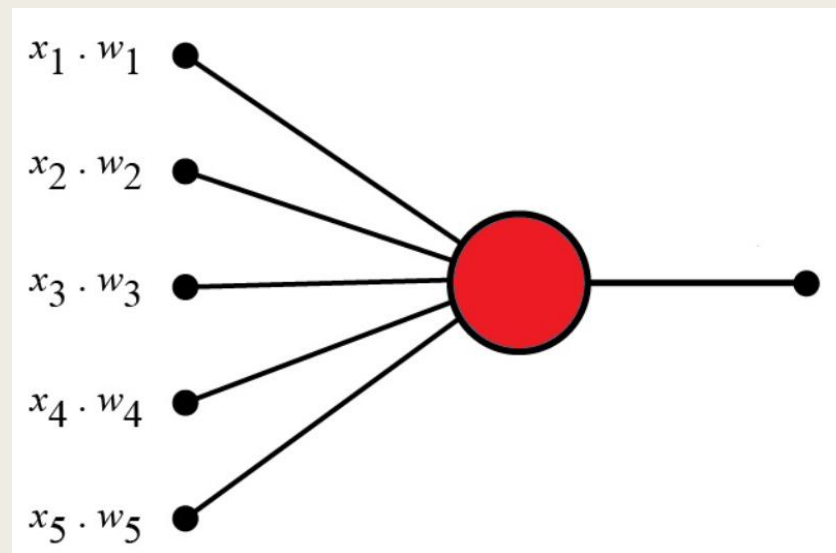
*A linear unit with three inputs.*

# The Perceptron

- A Perceptron is the simplest form of an Artificial Neural Network - essentially an artificial neuron.
- Frank Rosenblatt introduced the Perceptron in 1957 using an IBM 704 computer at Cornell Aeronautical Laboratory.
- The concept was inspired by how biological neurons work - receiving electrical inputs, processing them, and making decisions.
- Rosenblatt's idea was that Perceptrons could mimic brain behavior, allowing machines to learn and make decisions like humans.
- A **perceptron** and **linear unit** are **not** exactly the **same**, though they are closely related

# The Perceptron

- The original Perceptron takes several binary inputs and produces a single binary output (0 or 1).
- Each input is assigned a weight that indicates its importance.
- The weighted sum of all inputs is compared to a threshold value.
- If the sum exceeds the threshold, the output is 1 (True/Yes); otherwise, it is 0 (False/No).



# Perceptron Example

- Imagine a perceptron (in your brain).
- The perceptron tries to decide if you should go to a concert.
- Is the artist good? Is the weather good?
- What weights should these facts have?

Criteria	Input	Weight
Artists is Good	$x_1 = 0 \text{ or } 1$	$w_1 = 0.7$
Weather is Good	$x_2 = 0 \text{ or } 1$	$w_2 = 0.6$
Friend will Come	$x_3 = 0 \text{ or } 1$	$w_3 = 0.5$
Food is Served	$x_4 = 0 \text{ or } 1$	$w_4 = 0.3$
Drink is Served	$x_5 = 0 \text{ or } 1$	$w_5 = 0.4$

# The Perceptron Algorithm

- Set a threshold value
- Multiply all inputs with its weights
- Sum all the results
- Activate the output

## ■ 1. Set a threshold value:

- Threshold = 1.5

## ■ 2. Multiply all inputs with its weights:

- $x_1 * w_1 = 1 * 0.7 = 0.7$
- $x_2 * w_2 = 0 * 0.6 = 0$
- $x_3 * w_3 = 1 * 0.5 = 0.5$
- $x_4 * w_4 = 0 * 0.3 = 0$
- $x_5 * w_5 = 1 * 0.4 = 0.4$

## ■ 3. Sum all the results:

- $0.7 + 0 + 0.5 + 0 + 0.4 = 1.6$  (The Weighted Sum)

## ■ 4. Activate the Output:

- Return true if the sum  $> 1.5$  ("Yes I will go to the Concert")

# Task 1: Example

```
# Define inputs and weights
inputs = [1, 0, 1, 0, 1]
weights = [0.7, 0.6, 0.5, 0.3, 0.4]
threshold = 1.5

# Compute weighted sum
sum_val = 0
for i in range(len(inputs)):
    sum_val += inputs[i] * weights[i]

# Apply step activation (threshold)
activate = sum_val > threshold

# Print results
print(f"Weighted sum: {sum_val}")
print(f"Activated: {activate}")
```

# The Output

- The final output of the perceptron is the result of the activation function.
- It represents the perceptron's decision or prediction based on the input and the weights.
- The activation function maps the the weighted sum into a binary value.
- The binary **1** or **0** can be interpreted as **true** or **false** / **yes** or **no**.
- The output is **1** because:  $(\text{sum} > \text{treshold}) == \text{true}$ .

# Perceptron Learning

- The perceptron learns from examples through training, adjusting its weights based on errors.
- It compares its predicted output with the desired output and updates weights to minimize error.
- Learning algorithms such as the Perceptron Learning Rule or Backpropagation guide this adjustment process.
- This enables the perceptron to make accurate predictions for new inputs.
- However, single-layer perceptrons can only learn linearly separable patterns.
- Multi-layer perceptrons with non-linear activation functions overcome this limitation and can learn complex patterns, forming the foundation of modern neural networks.

## Task 2: Loan Approval System (AND Logic)

- An AI model decides whether to approve a loan based on two conditions:
  - Applicant's credit score  $\geq 700$
  - Applicant's income  $\geq \$50,000$
- The loan should be approved only if both conditions are met.
- Implement this using a perceptron model that performs an AND operation and test it on four applicants.

# Code: Loan Approval System (AND Logic)

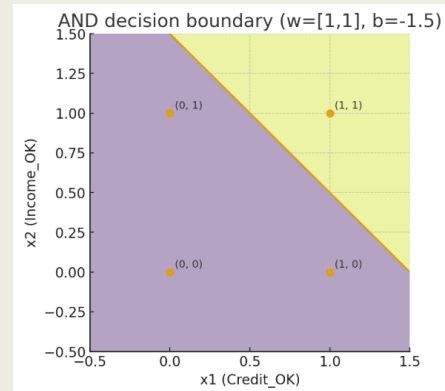
```
import numpy as np
```

```
def perceptron(x, w, b):  
    return int(np.dot(x, w) + b >= 0)
```

```
# Inputs: (credit_score_ok, income_ok)  
# 1 = condition satisfied, 0 = not satisfied  
applicants = [(0,0), (0,1), (1,0), (1,1)]
```

```
# AND logic parameters  
w = np.array([1, 1])  
b = -1.5
```

```
print("Loan Approval Results:")  
for i, x in enumerate(applicants, start=1):  
    decision = perceptron(x, w, b)  
    print(f"Applicant {i}: Credit_OK={x[0]}, Income_OK={x[1]}  
→ Approved={decision}")
```



Perceptron decision boundary

Loan Approval Results:

Applicant 1: Credit\_OK=0, Income\_OK=0 → Approved=0  
Applicant 2: Credit\_OK=0, Income\_OK=1 → Approved=0  
Applicant 3: Credit\_OK=1, Income\_OK=0 → Approved=0  
Applicant 4: Credit\_OK=1, Income\_OK=1 → Approved=1

## Task 3: Spam Filter System (OR Logic)

- A spam detection AI checks two conditions:
  - The email contains **suspicious keywords** (1 = yes, 0 = no)
  - The email is from an **unknown sender** (1 = yes, 0 = no)
- If **any one** of these is true, the email is **spam**.  
Simulate this using a **perceptron performing OR logic**.

# Task 3: Spam Filter System (OR Logic)

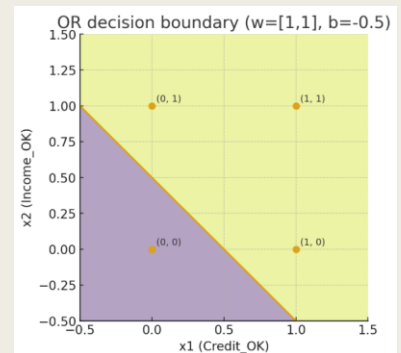
```
import numpy as np
```

```
def perceptron(x, w, b):  
    return int(np.dot(x, w) + b >= 0)
```

```
# Inputs: (suspicious_keywords, unknown_sender)  
emails = [(0,0), (0,1), (1,0), (1,1)]
```

```
# OR logic parameters  
w = np.array([1, 1])  
b = -0.5
```

```
print("\nSpam Filter Results:")  
for i, x in enumerate(emails, start=1):  
    result = perceptron(x, w, b)  
    print(f"Email {i}: Keywords={x[0]}, UnknownSender={x[1]}  
→ Spam={result}")
```



Perceptron decision boundary

Spam Filter Results:

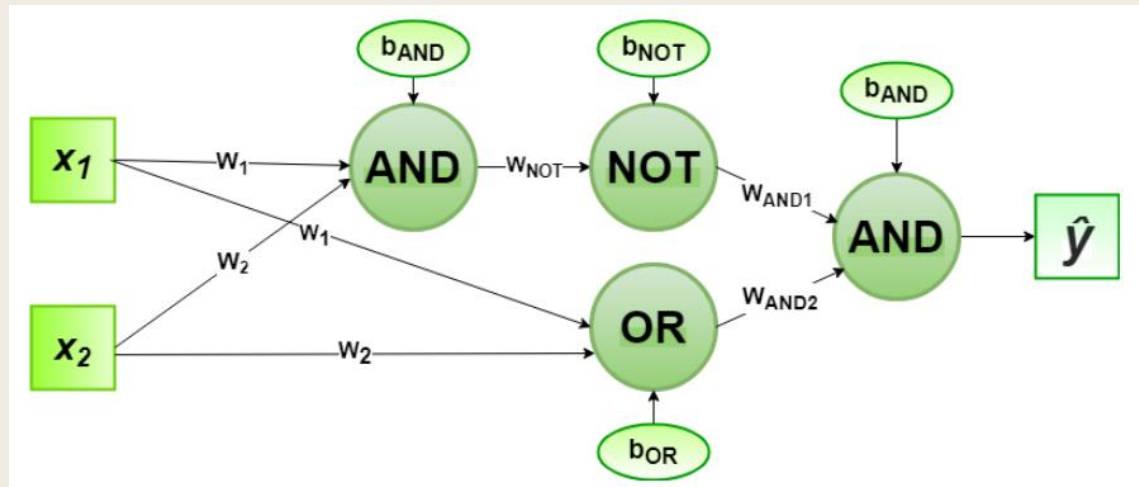
```
Email 1: Keywords=0, UnknownSender=0 → Spam=0  
Email 2: Keywords=0, UnknownSender=1 → Spam=1  
Email 3: Keywords=1, UnknownSender=0 → Spam=1  
Email 4: Keywords=1, UnknownSender=1 → Spam=1
```

# Multilayer Perceptron (MLP)

- Perceptron Works for AND / OR but Fails for XOR
- A single-layer perceptron can only learn linearly separable patterns
- To solve XOR, we need to combine multiple perceptrons (layers) - this leads to the invention of the Multilayer Perceptron (MLP) or Neural Network.

$$XOR = (x_1 \text{ OR } x_2) \text{ AND NOT } (x_1 \text{ AND } x_2)$$

- Perceptron → draws straight lines (linear decisions)
- XOR → needs a curve or combination of lines (nonlinear)



# Task 4: Security Access XOR

```
import numpy as np

# Trained MLP weights (from previous training or similar
# setup)
def sigmoid(x): return 1 / (1 + np.exp(-x))

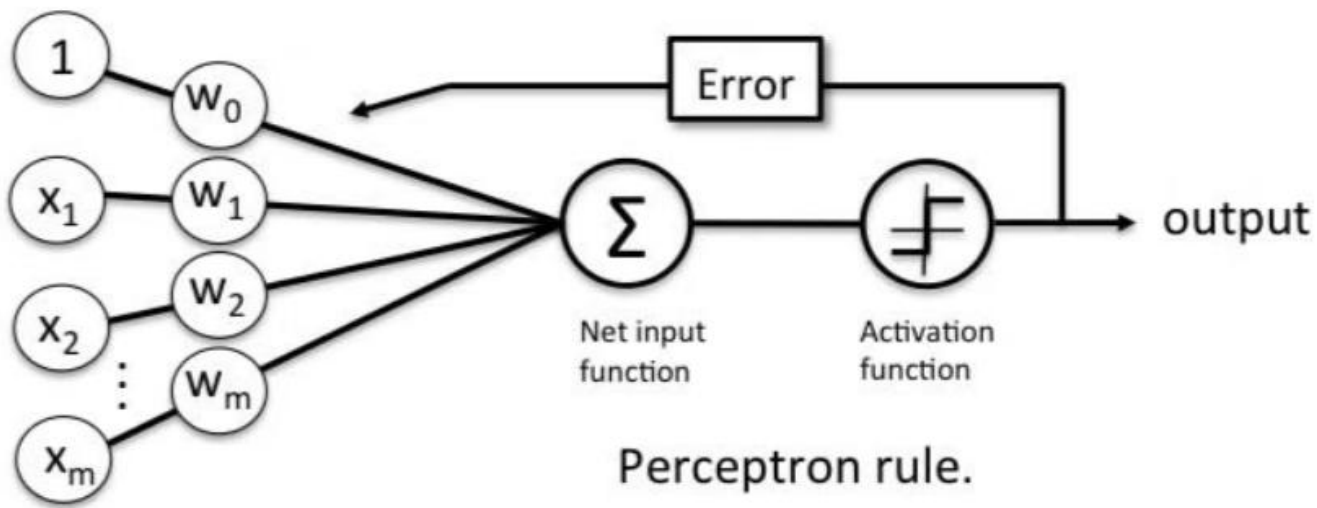
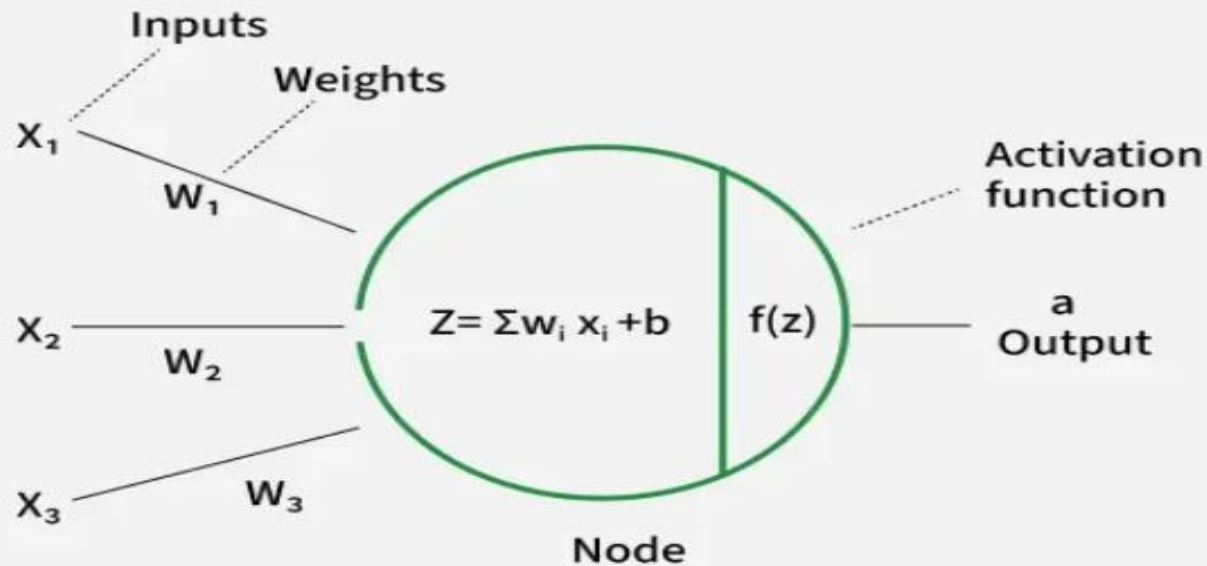
# Define custom XOR MLP (using learned weights)
def xor_nn(x1, x2):
    # Hidden layer
    h1 = sigmoid(20*x1 - 20*x2 - 10)    # neuron 1
    h2 = sigmoid(-20*x1 + 20*x2 - 10)    # neuron 2
    # Output layer (combine)
    output = sigmoid(20*h1 + 20*h2 - 10)
    return int(output >= 0.5)

# Test cases
cases = [(0,0), (0,1), (1,0), (1,1)]
print("Security Access System Results:")
for case in cases:
    print(f"Keycard={case[0]}, Face={case[1]} →
Access={xor_nn(*case)}")
```

Security Access System Results:  
Keycard=0, Face=0 → Access=0  
Keycard=0, Face=1 → Access=1  
Keycard=1, Face=0 → Access=1  
Keycard=1, Face=1 → Access=0

# Activation Functions

## Activation functions in Neural Networks



# Cat vs Not-Cat Classification

## Using a Simple Neural Network

- Perceptron with hidden layer
- Step-by-step neuron calculations
- Activation functions (ReLU & Sigmoid)

### ■ Problem Overview

- **Goal:** Predict whether an animal is a cat or not.

### ■ Input Features:

- x1: Fluffy (1 = yes, 0 = no)
- x2: Meows (1 = yes, 0 = no)

### ■ Output:

- Probability of being a CAT

# Cat vs Not-Cat Classification

Using a Simple Neural Network

## ■ Network Architecture

- A tiny neural network:
- Input layer (2 features)
- Hidden layer (2 neurons)
- Output layer (1 neuron)

## ■ Input Example

### ■ We classify:

- $x = [1, 1]$
- Fluffy = Yes
- Meows = Yes
- This will be fed through the network.

# Cat vs Not-Cat Classification

Using a Simple Neural Network

## ■ Hidden Neuron $h_1$

- Weights & bias:
- $w_1 = [0.8, 0.4]$
- $b_1 = -0.2$

## ■ Calculation:

- $z_1 = 0.8 * 1 + 0.4 * 1 - 0.2$
- $z_1 = 1.0$

## ■ Activation:

- $h_1 = \text{ReLU}(1.0) = 1.0$

# Cat vs Not-Cat Classification

## Using a Simple Neural Network

### ■ Hidden Neuron h2

- Weights & bias:
- $w2 = [0.3, 0.9]$
- $b2 = -0.1$

### ■ Calculation:

- $z2 = 0.3 * 1 + 0.9 * 1 - 0.1$
- $z2 = 1.1$

### ■ Activation:

- $h2 = \text{ReLU}(1.1) = 1.1$

### ■ Hidden layer outputs:

- $h1 = 1.0$
- $h2 = 1.1$

# Cat vs Not-Cat Classification

## Using a Simple Neural Network

### ■ Hidden layer outputs:

- $h1 = 1.0$
- $h2 = 1.1$

### ■ Calculation:

- $z\_out = 1.2 * h1 + 0.5 * h2 - 0.4$
- $z\_out = 1.35$

### ■ Sigmoid Activation

- The sigmoid function converts  $z\_out$  into a probability:
- $prob = 1 / (1 + e^{(-1.35)})$
- $prob \approx 0.794$

### ■ Output: 79.4% chance the animal is a CAT

### ■ Final Classification

- Since  $0.794 > 0.5 \rightarrow$  The network predicts:
- **CAT**

# Cat vs Not-Cat Classification

## Using a Simple Neural Network

### ■ Python Code

```
import numpy as np
def relu(x): return np.maximum(0, x)
def sigmoid(x): return 1 / (1 + np.exp(-x))
x = np.array([1, 1])
W_hidden = np.array([[0.8, 0.4], [0.3, 0.9]])
b_hidden = np.array([-0.2, -0.1])
z_hidden = np.dot(W_hidden, x) + b_hidden
h = relu(z_hidden)
W_out = np.array([1.2, 0.5])
b_out = -0.4
z_out = np.dot(W_out, h) + b_out
prob = sigmoid(z_out)
print("Hidden outputs:", h)
print("CAT probability:", prob)
```

Hidden outputs: [1. 1.1]

CAT probability: 0.7941296281990528