Dr. Denton Bobeldyk

CIS 365 Artificial Intelligence

Perceptron Learning

Week in Review

Blackboard Check-in

Delivery Methods

Lecture

Videos

Lab Time

Small Groups

Decision Making

- Multiple factors weigh into determining the outcome of a decision
- Multiple factors weigh into determining the action to take

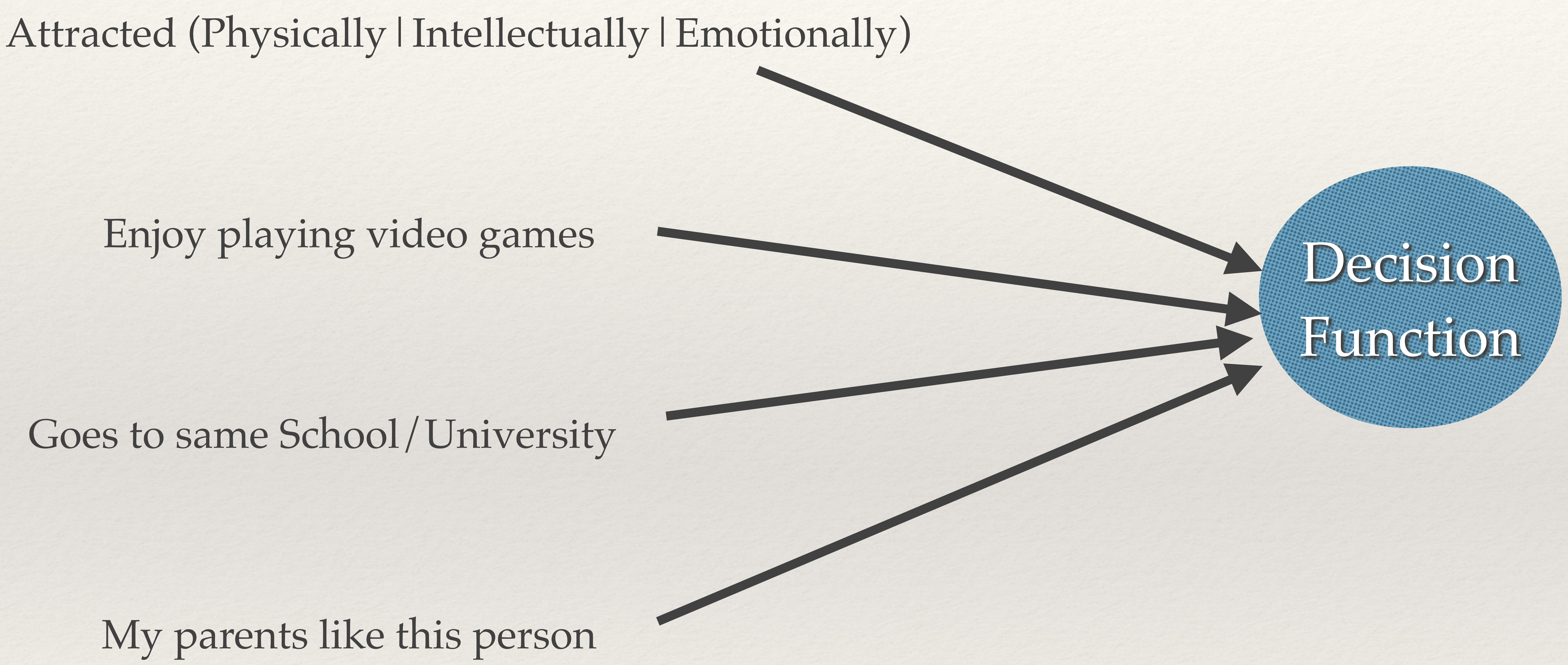
Decision Making

- Would I like to date this person?

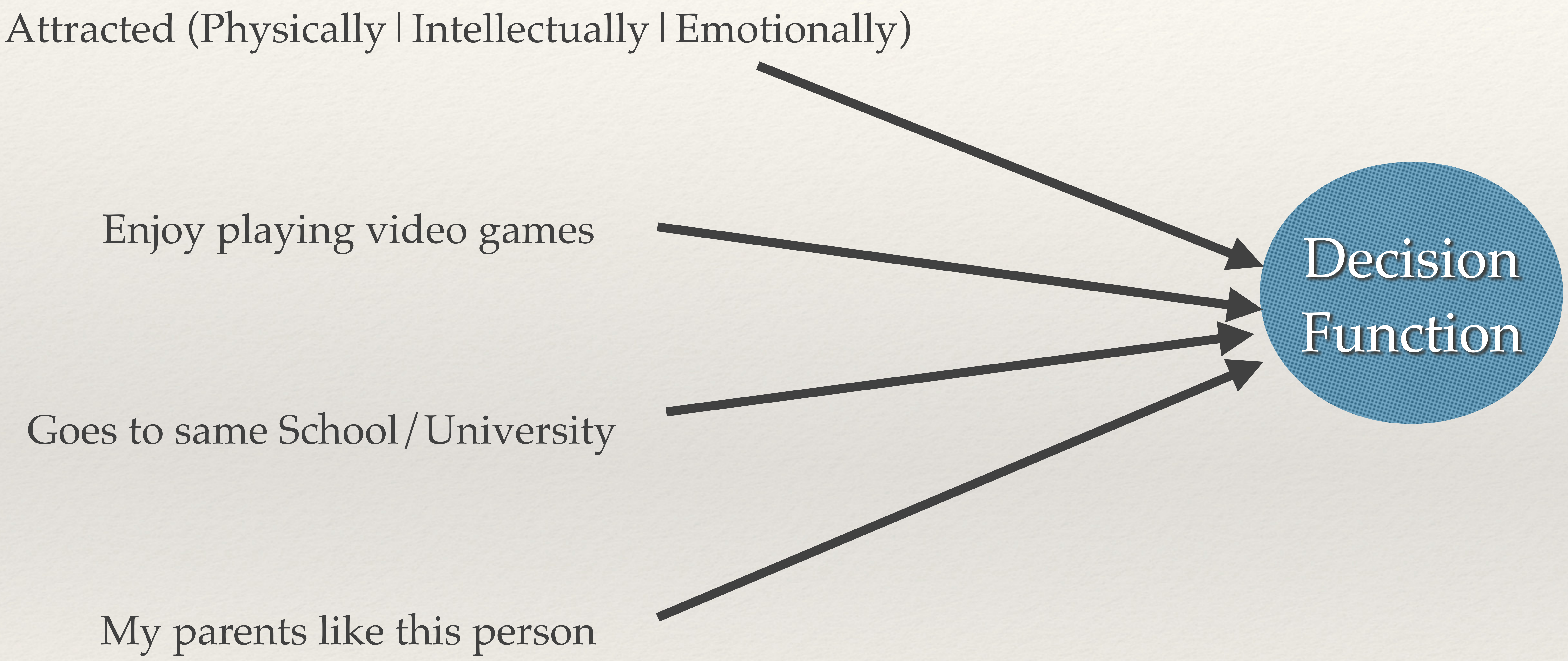
Decision Making - Class Exercise

- Would I like to date this person?
    - What considerations go into this decision?

Decision Making

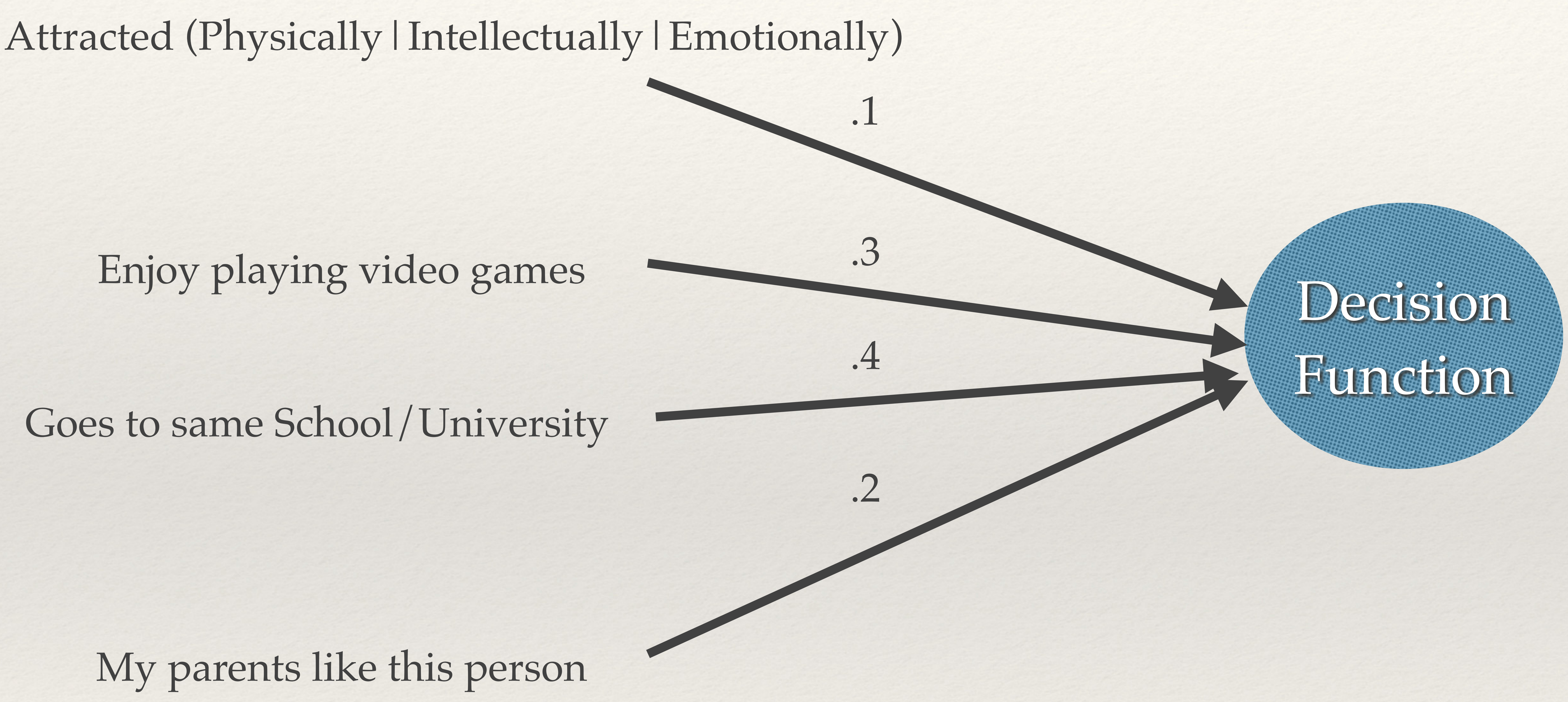


Decision Making

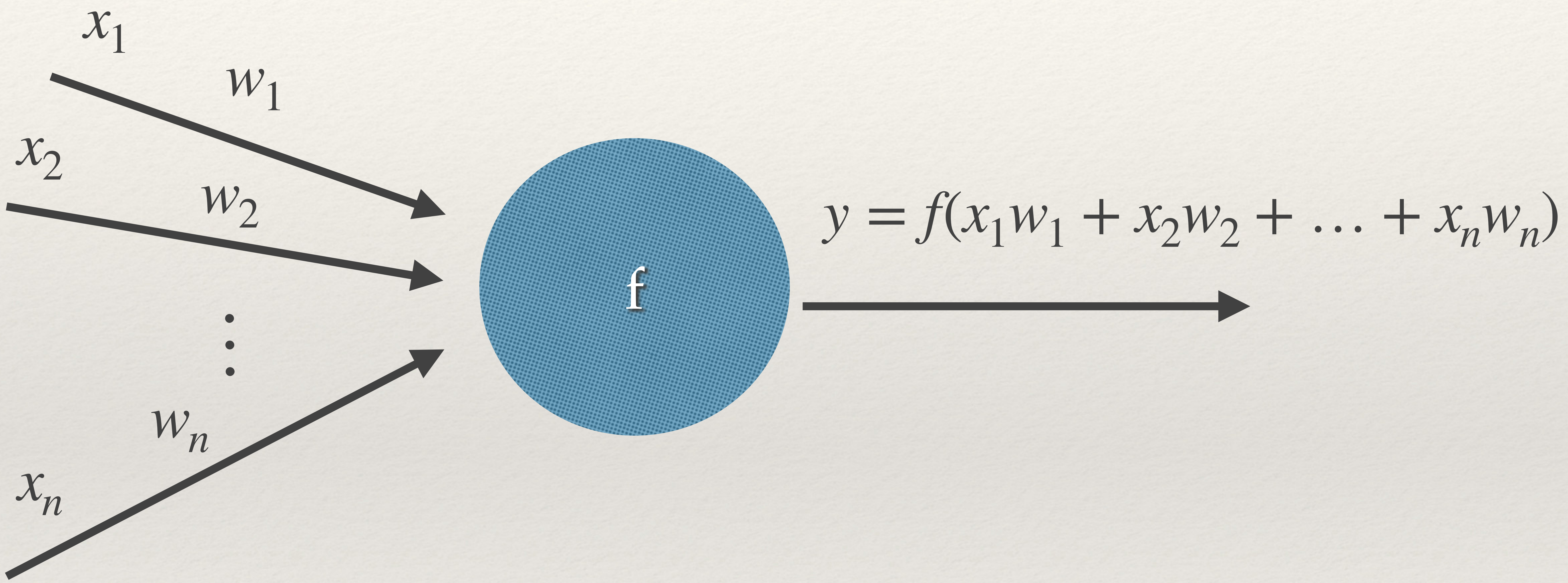


Can I weigh how much each of these factor into the decision?

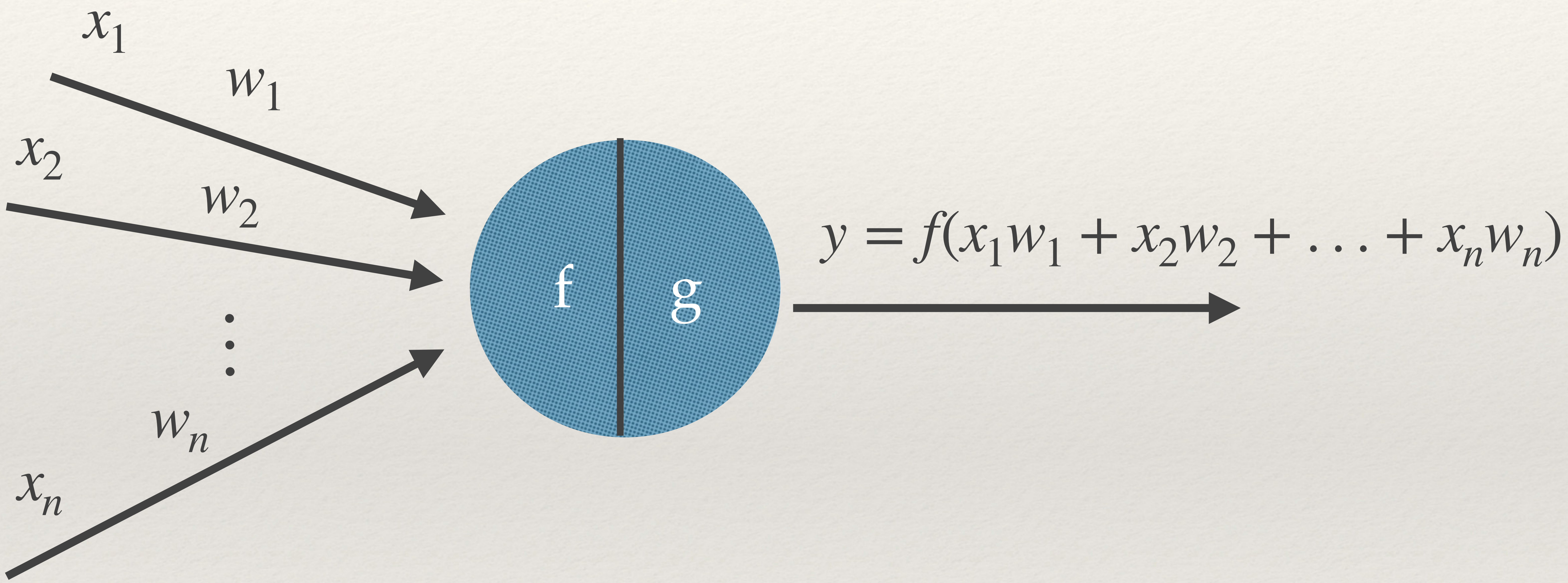
Decision Making



Neuron



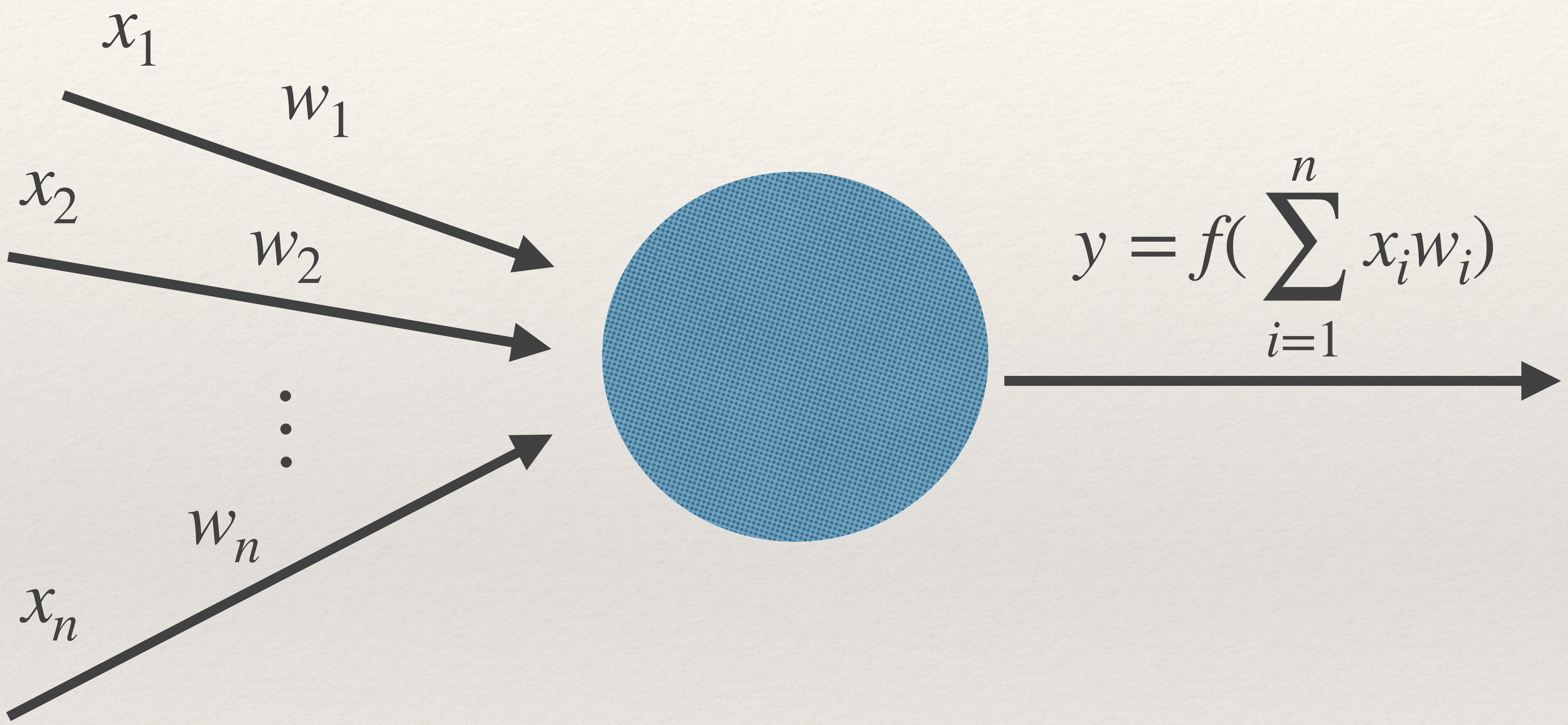
Mc Culloch-Pitts Neuron



$$g = \bar{x} \cdot \bar{w}$$

F is an activation function

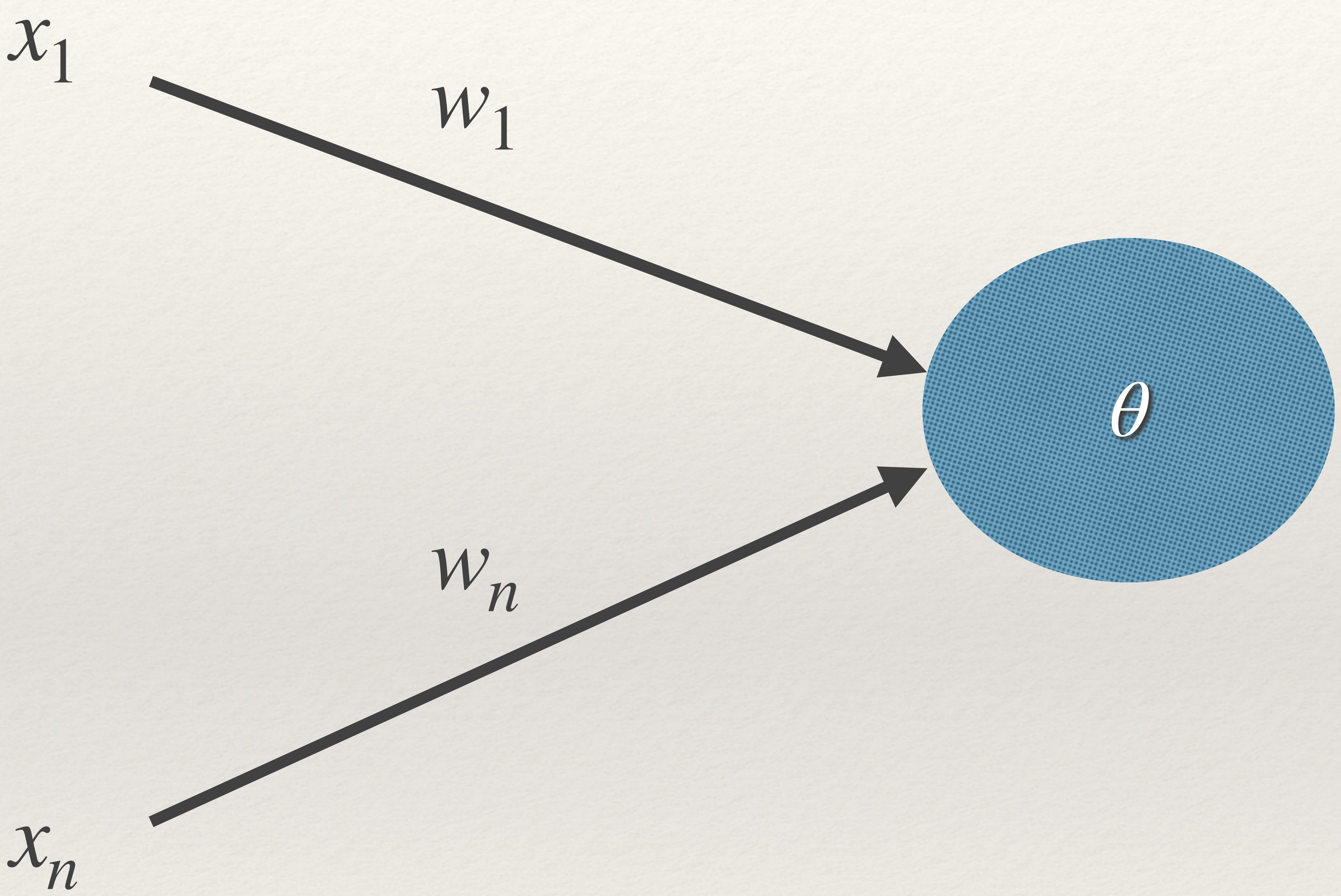
Perceptron - General Notation



F is an activation function

Perceptron with Thresholding

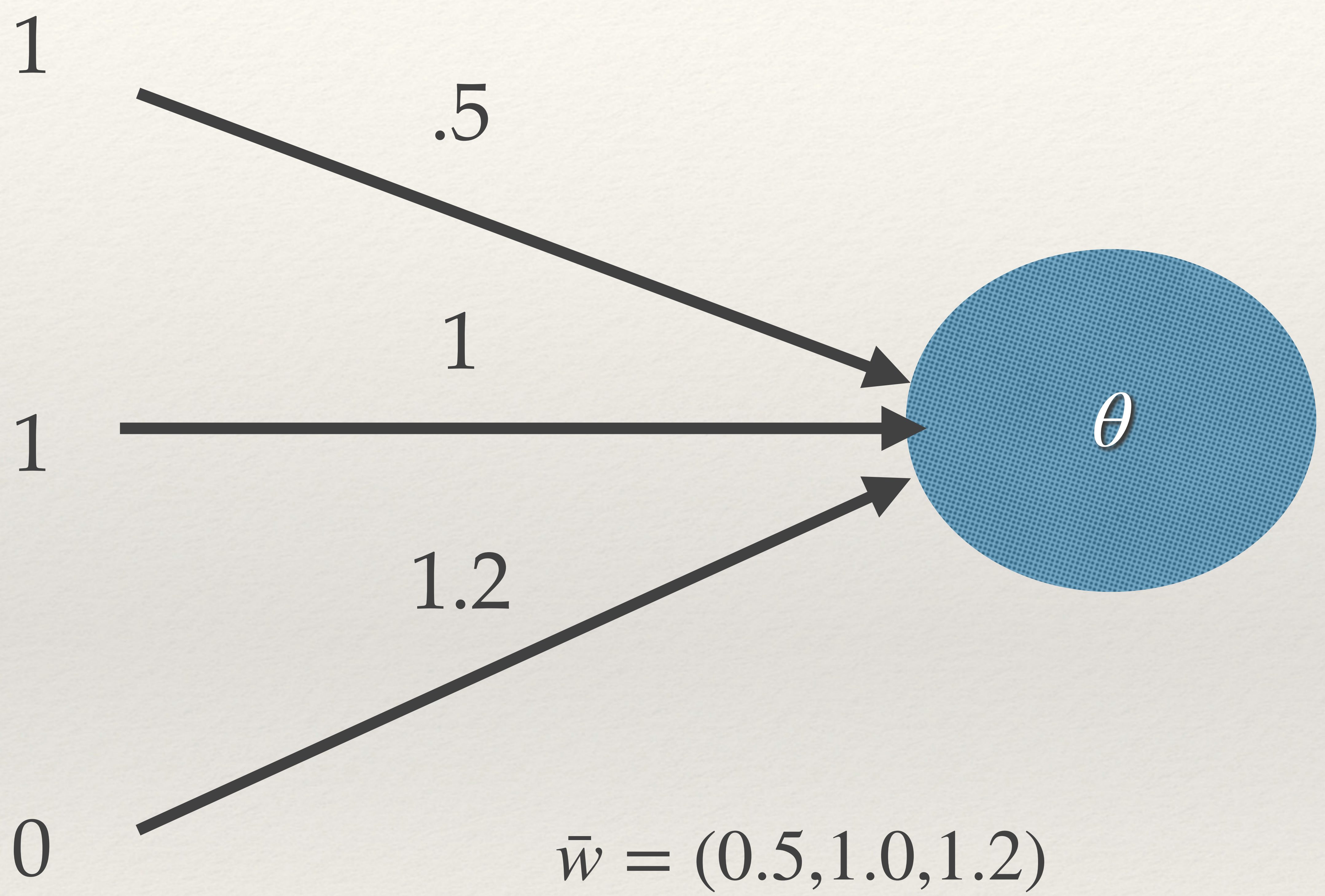
Percepton



An abstract neuron referred to as a Threshold Logic Unit

Percepton

$\bar{x} = (1,1,0)$



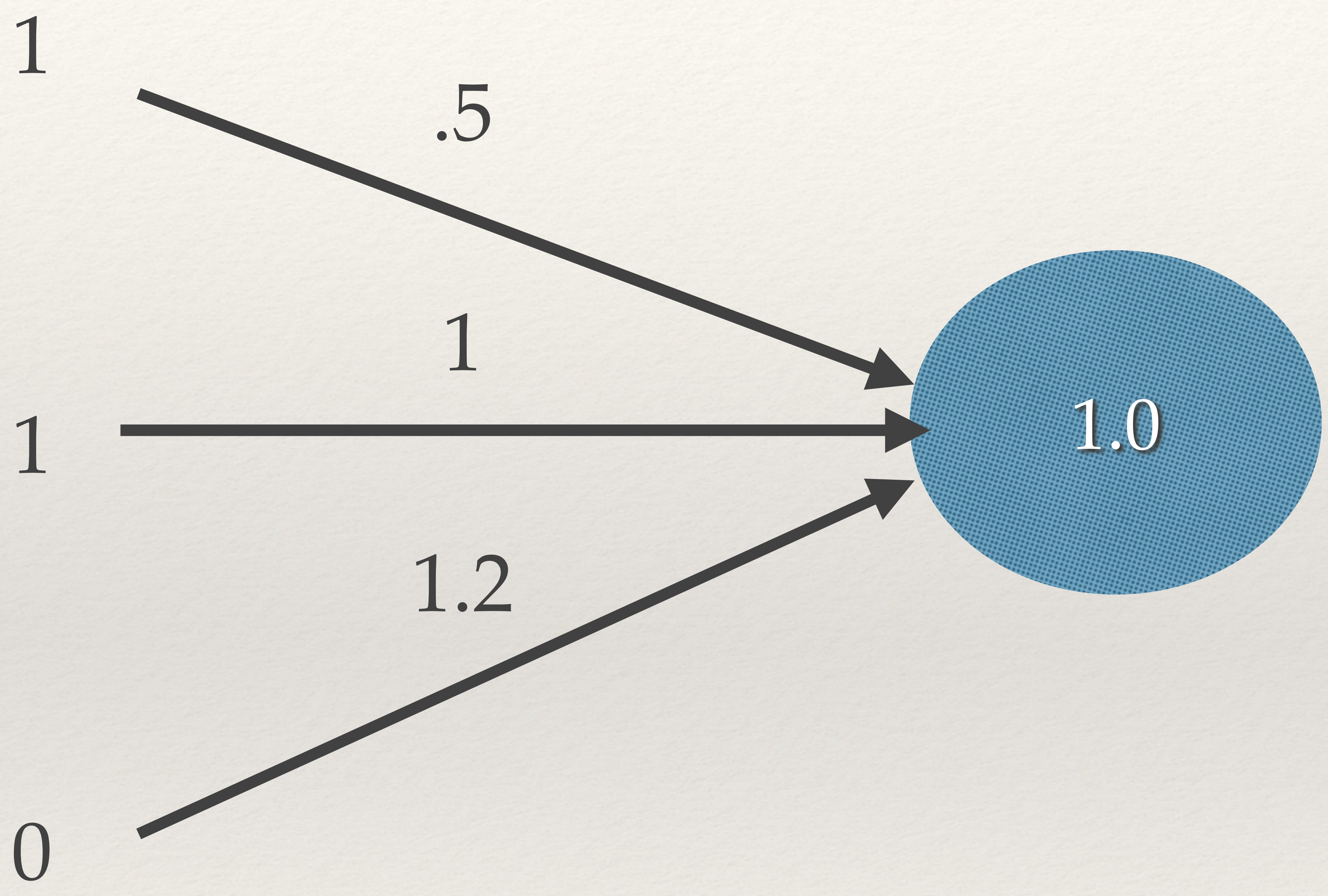
$\bar{w} = (0.5,1.0,1.2)$

An abstract neuron referred to as a Threshold Logic Unit

Threhold Logic Unit

$\bar{x} = (1,1,0)$

$\bar{w} = (0.5,1.0,1.2)$

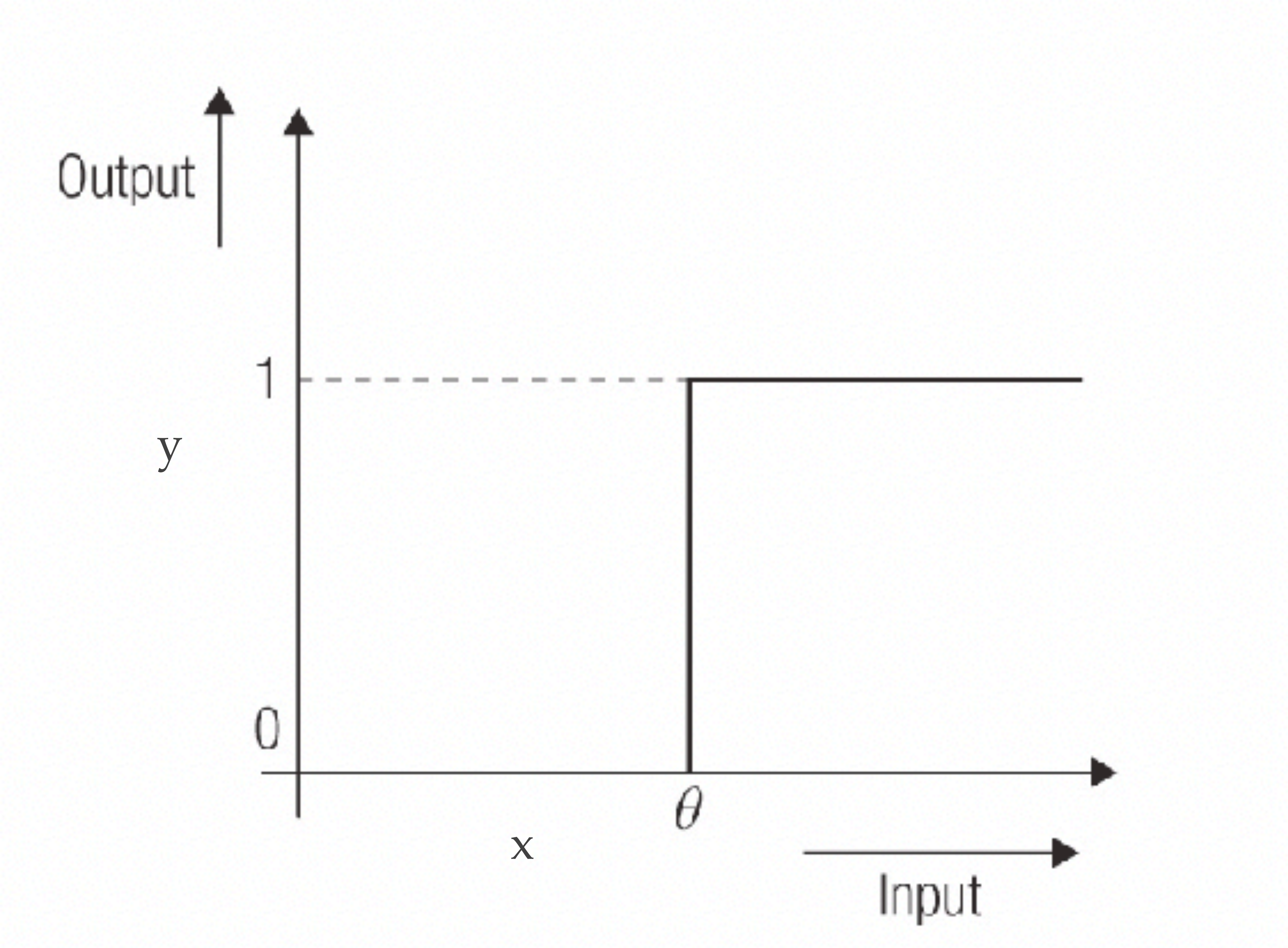


If the output greater than or equal to the threshold send output of 1

$$\bar{x} \cdot \bar{w} = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 =$$

$$(1 * 0.5) + (1 * 1.0) + (0 * 1.2) = 1.5$$

Activation Function - Threshold Function



$$y = \begin{cases} 1 & \text{if } x \geq \text{threshold} \\ 0 & \text{if } x < \text{threshold} \end{cases}$$

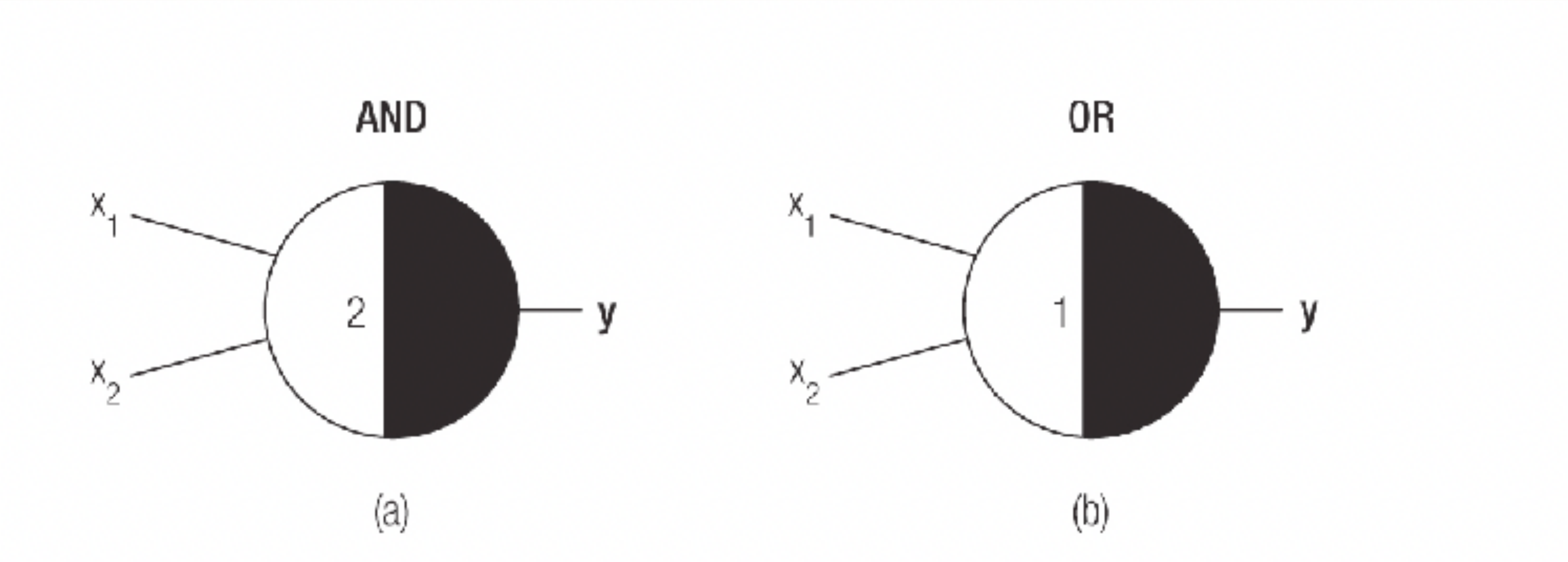
If the value is greater than threshold θ, the out put is 1

Class Exercise 10-12 minutes

- Create a threshold logic unit for:
  - AND Gate
  - OR Gate

Answer Next Slide

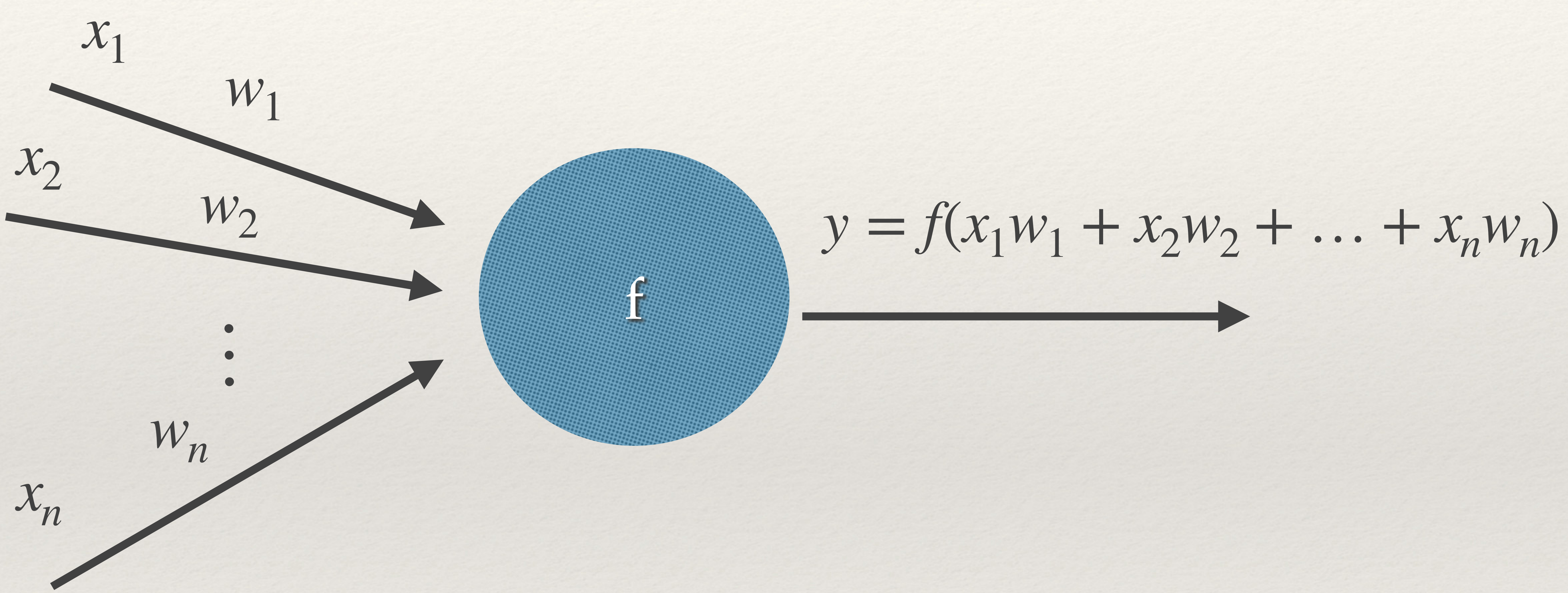
McCulloch-Pits two input gates



Threshold for AND: 2
Threshold for OR: 1

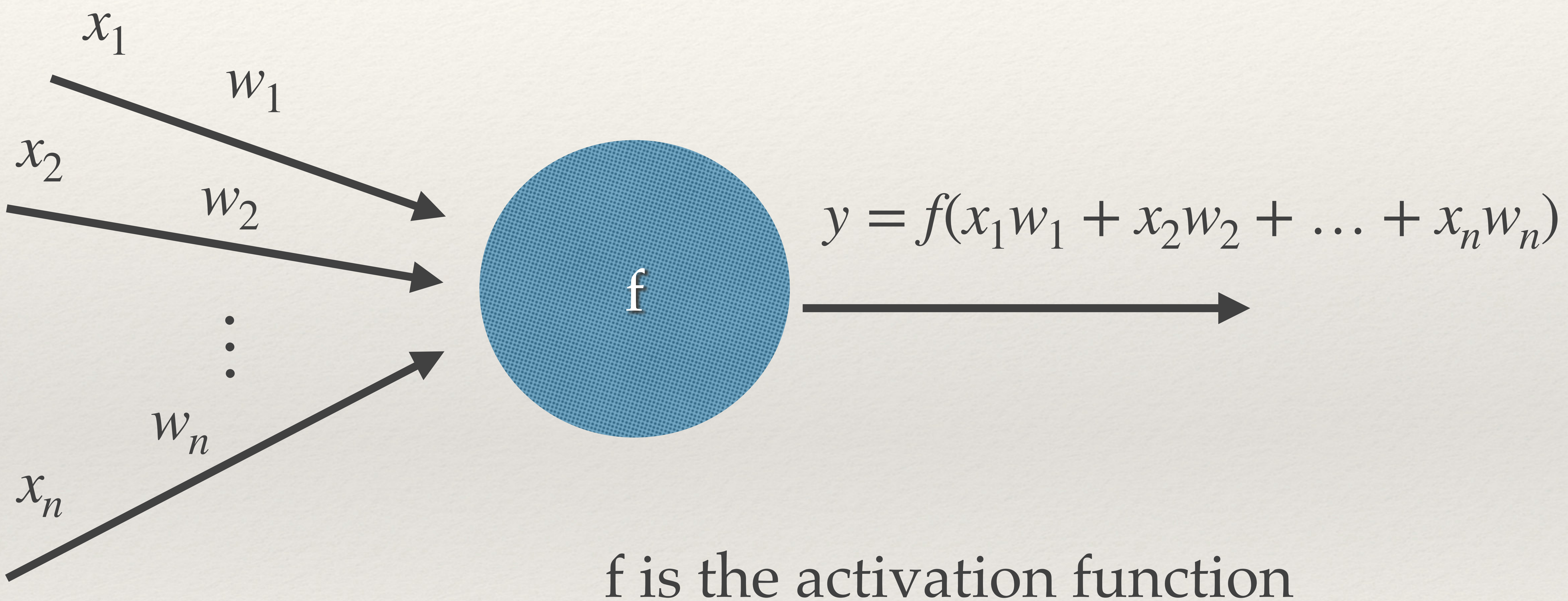
NOTE: No weights

Neuron



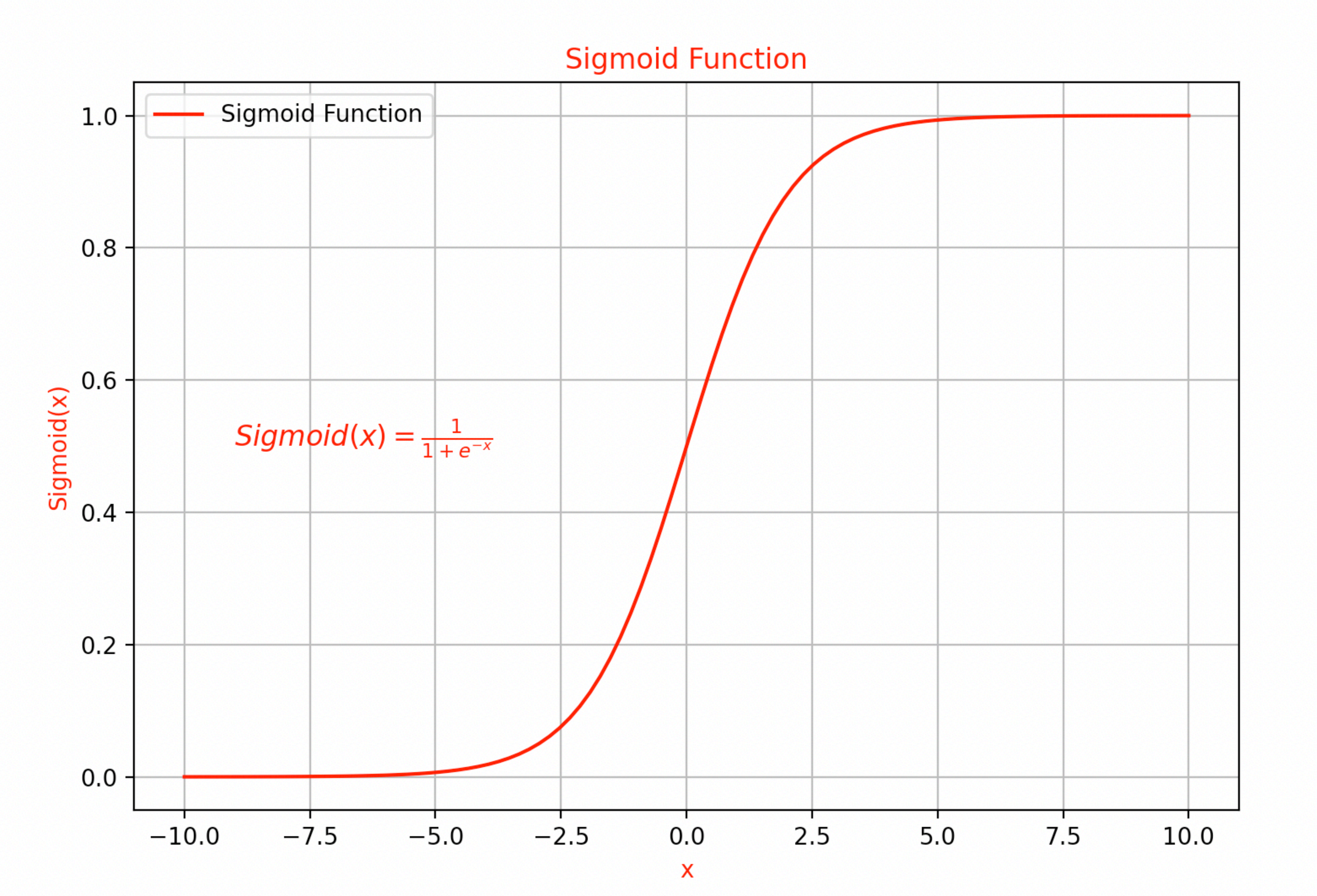
Activation Functions

- Sigmoid
- Tanh
- ReLu



f is the activation function

Activation Functions - Sigmoid Function



https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

Sigmoid Activation Function Properties

- Range: Output of the sigmoid function is contained between 0 and 1
- Monotonicity: as the input increases, the output increases also
- Differentiability: Smooth and differentiable at all points. This is useful for gradient based optimization methods like backpropagation
- Derivative: $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$

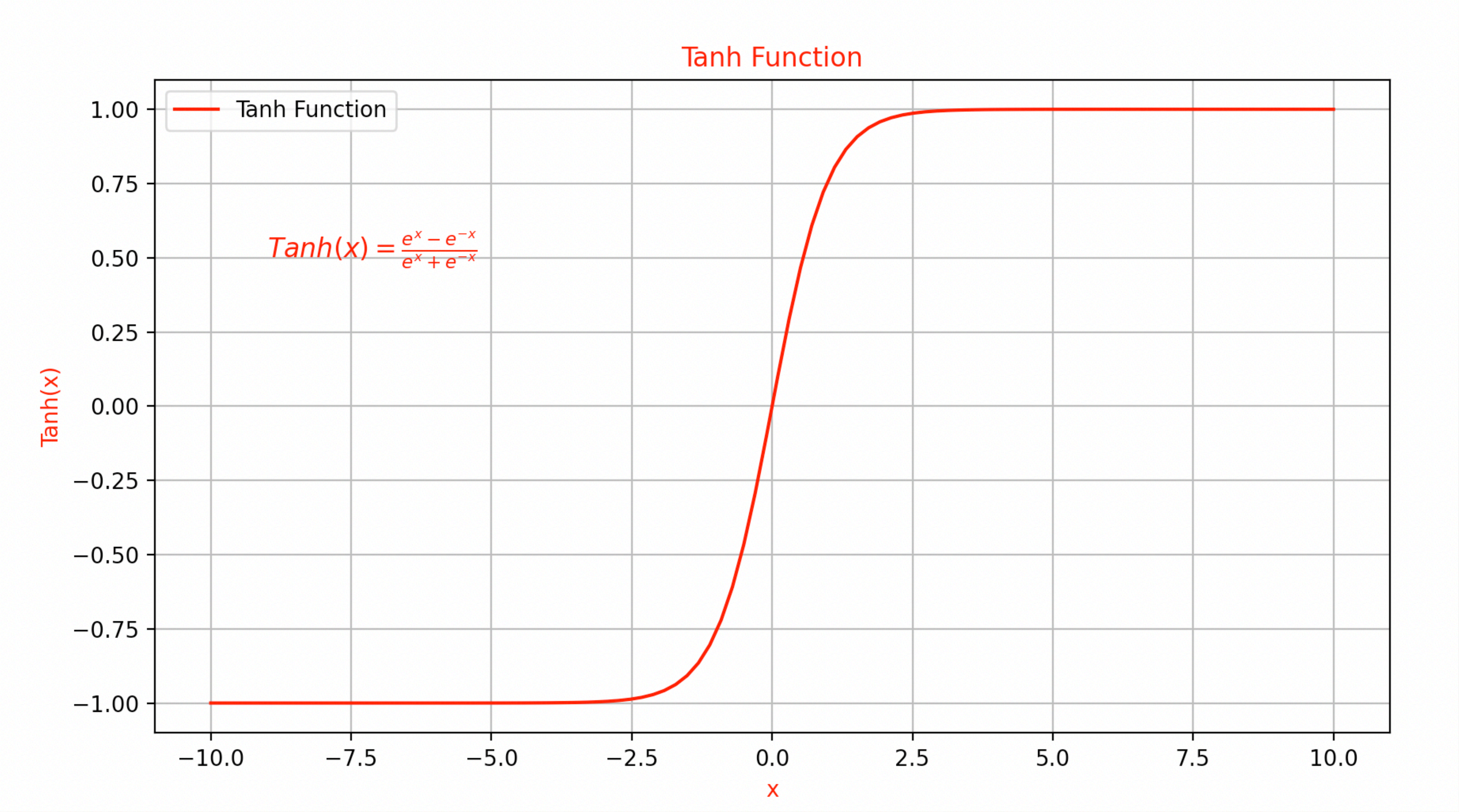
$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid Activation Function Properties

- Gradient Saturation: For large positive or large negative inputs, the sigmoid function saturates. This can lead to the vanishing gradient problem where update to weights become very small slowing down or stopping the learning process
- Output bound: Sigmoid outputs values close to 0 or 1, which makes it useful in binary classification where we want probabilities as outputs
- Non-zero-centered: The outputs are always positive, which can sometimes slow down training

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

Activation Functions - Tanh Function



https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

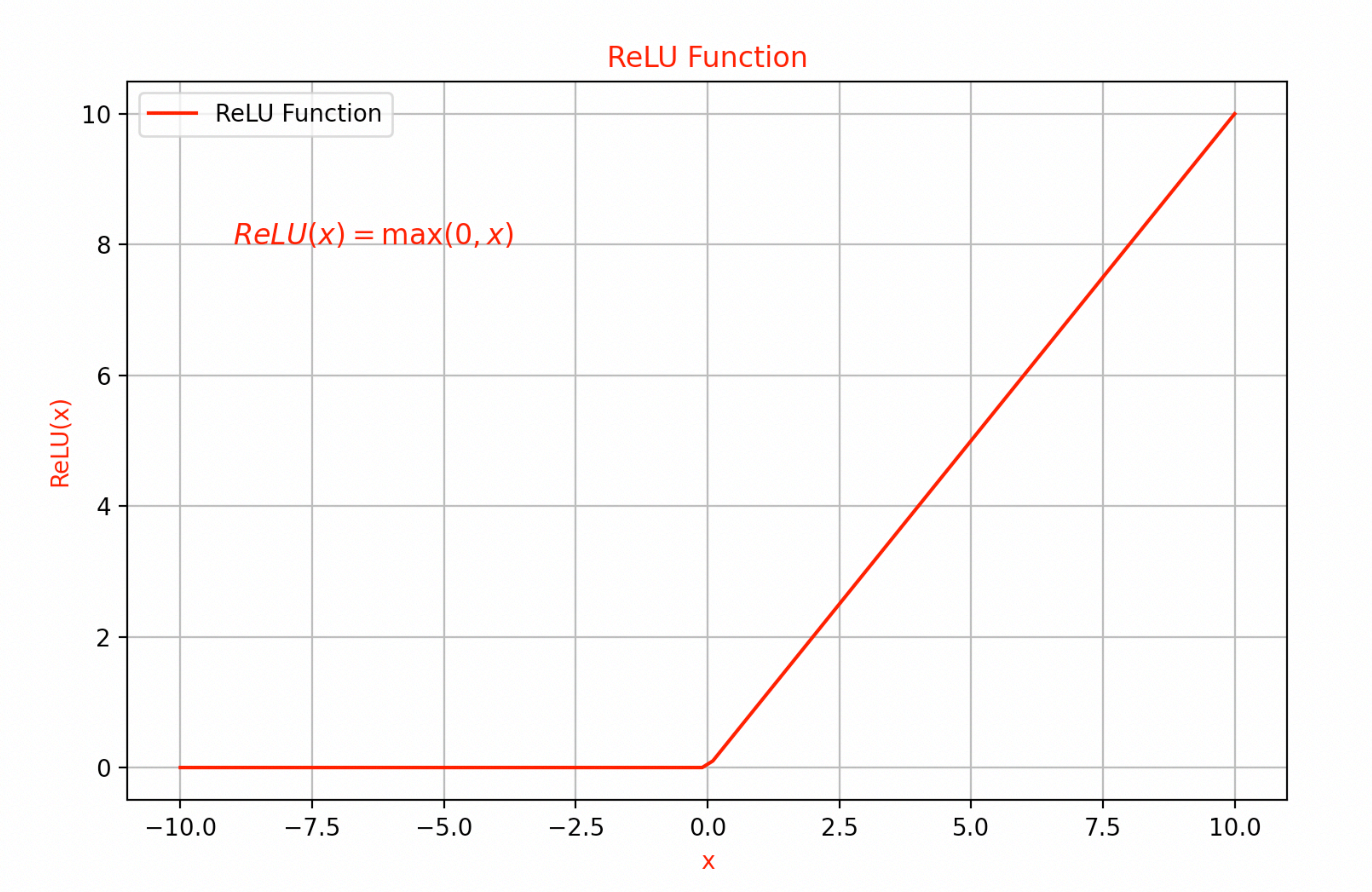
Tanh Activation Function Properties

- Range: Output of the tanh function is contained between -1 and 1, making it zero-centered
- Zero-centered: Gradient updates are more balanced, leading to more efficient training as weight updates can flow both positively and negatively.
- S-shaped curve: Smoothly transitions between the bounds -1 and 1
- Monotonicity: as the input increases, the output increases also
- Differentiability: Smooth and differentiable at all points. This is useful for gradient based optimization methods like backpropagation
- Derivative: $tanh'(x) = 1 - tanh^2(x)$

Tanh Activation Function Properties

- Symmetry: The tanh function is odd-symmetric about the origin. This means for any input x, $tanh(-x) = -tanh(x)$. This symmetry around zero helps avoid bias in the activation
- Gradient Saturation: For large positive or large negative inputs, the tanh function saturates. This can lead to the vanishing gradient problem where update to weights become very small slowing down or stopping the learning process
- Output bound: Sigmoid outputs values close to -1 or 1, tanh is effective in normalizing data and limiting the range of activation values, which can help stabilize learning in certain cases
- Better for Hidden Layers: Compared to the sigmoid function because of it's zero centered output, however in modern deep learning architectures ReLU and its variants are often used instead due to their ability to mitigate the vanishing gradient problem

Activation Functions - Relu Function



https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

ReLU Activation Function Properties

- Range: The ReLU function outputs the values in the range of
- Sparsity: One of ReLU's defining features is that it outputs 0 for any negative input. This leads to sparsity in the activations, meaning any neurons can have zero output at any given time. This sparsity can improve computational efficiency and reduce the complexity of the model, especially in large neural networks.
- Non-linearity: introduces non-linearity into the model, which is essential for learning complex patterns.
- Efficient gradient propagation: ReLU alleviates the vanishing gradient problem faced by sigmoid and tang functions. For positive input values, the gradient of ReLU is always 1, ensuring that gradients can flow through the network efficiently. This helps in faster convergence during training, especially in deep networks

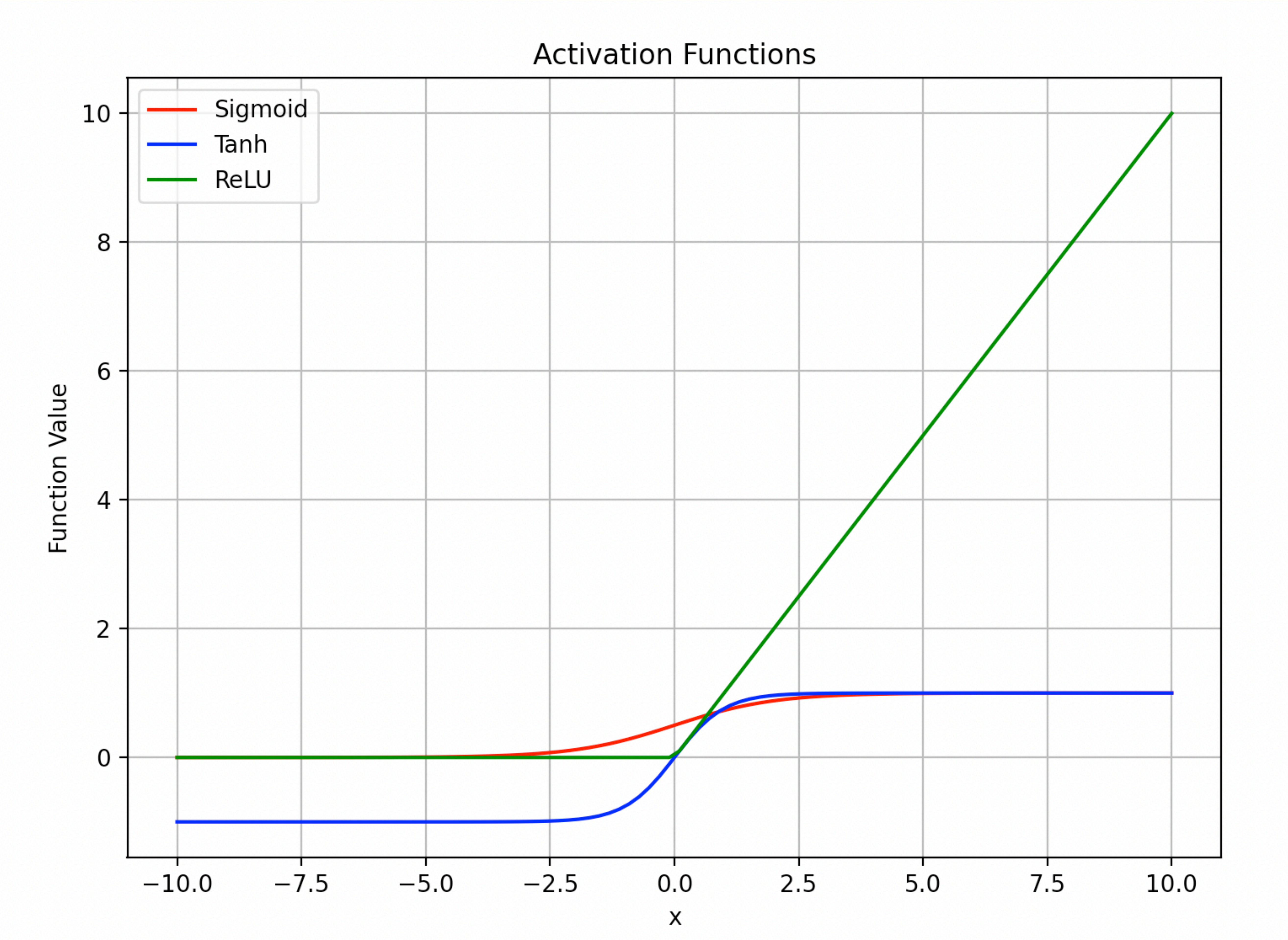
ReLU Activation Function Properties

- Derivative: The derivate of ReLU is simple
- Computational efficiency: it only involves a thresholding operation, making it very fast to compute unlike sigmoid and tang which involve exponentiation
- Not zero-centered: like the sigmoid function, ReLU is not zero-centered, meaning its output is always non-negative. This can potentially lead to some inefficiencies during training, but this issue is generally outweighed by its other advantages
- Dying ReLU problem: Key drawback of ReLU, if a neuron receives only negative inputs during training, it will output 0, and its gradient will also be 0, preventing a neuron form learning. Once the neuron can get stuck in this state and never activate again. This can lead to dead neurons that never fire for any input reducing the model's capacity.

ReLU Activation Function Properties (cont'd)

- Variants of ReLU such as Leaky ReLU and Parametric ReLU (PReLU) have been introduced. Leaky ReLU allows a small, non-zero gradient for negative inputs, reducing the chances of neurons dying
- Used in Hidden layers - more on this to come!
- Unbounded output: The output of the ReLU can grow indefinitely for positive inputs. While this can be useful for allowing the model to learn larger activations, it can also lead to potential issues like exploding gradients if not properly managed (e.g., using techniques like gradient clipping)

Activation Functions



https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

Activation Functions

- Which one should I use?
  - Designing neural networks can feel like more of an ‘art’ than a ‘science’
  - The more we’re learning about them, the easier it is to make these decisions
  - https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function

TLU with two inputs

By setting $\bar{x} \cdot \bar{w} = \theta$ we obtain: $x_1{}^* \, w_1 + x_2{}^* \, w_2 = \theta$. We solve for x_2 in terms of

x_1, w_1, w_2 and θ,

yielding:

$x_2{}^* \, w_2 = \theta - x_1{}^* \, w_1$.

Some algebraic manipulation gives us:

$$x_2 * w_2 = -x_1 * w_1 + \theta$$

$$x_2 = -\frac{w_1}{w_2} * x_1 + \frac{\theta}{w_2}.$$

TLU with two inputs

Recall that the equation of a straight line is: $y = m * x + b$ where m is the slope (m equals $\frac{\Delta y}{\Delta x}$ or the change in y divided by the change in x), and b is the y-intercept. Hence, we have a straight line whose slope equals $-\frac{w_1}{w_2}$ and intercept equals $\frac{\theta}{w_2}$ Substituting the values for w_1, w_2, θ, shown in Figure 11.13 we have: $x_2 = -x_1 + 2$. This line is shown in Figure 11.14.

TLU with two inputs

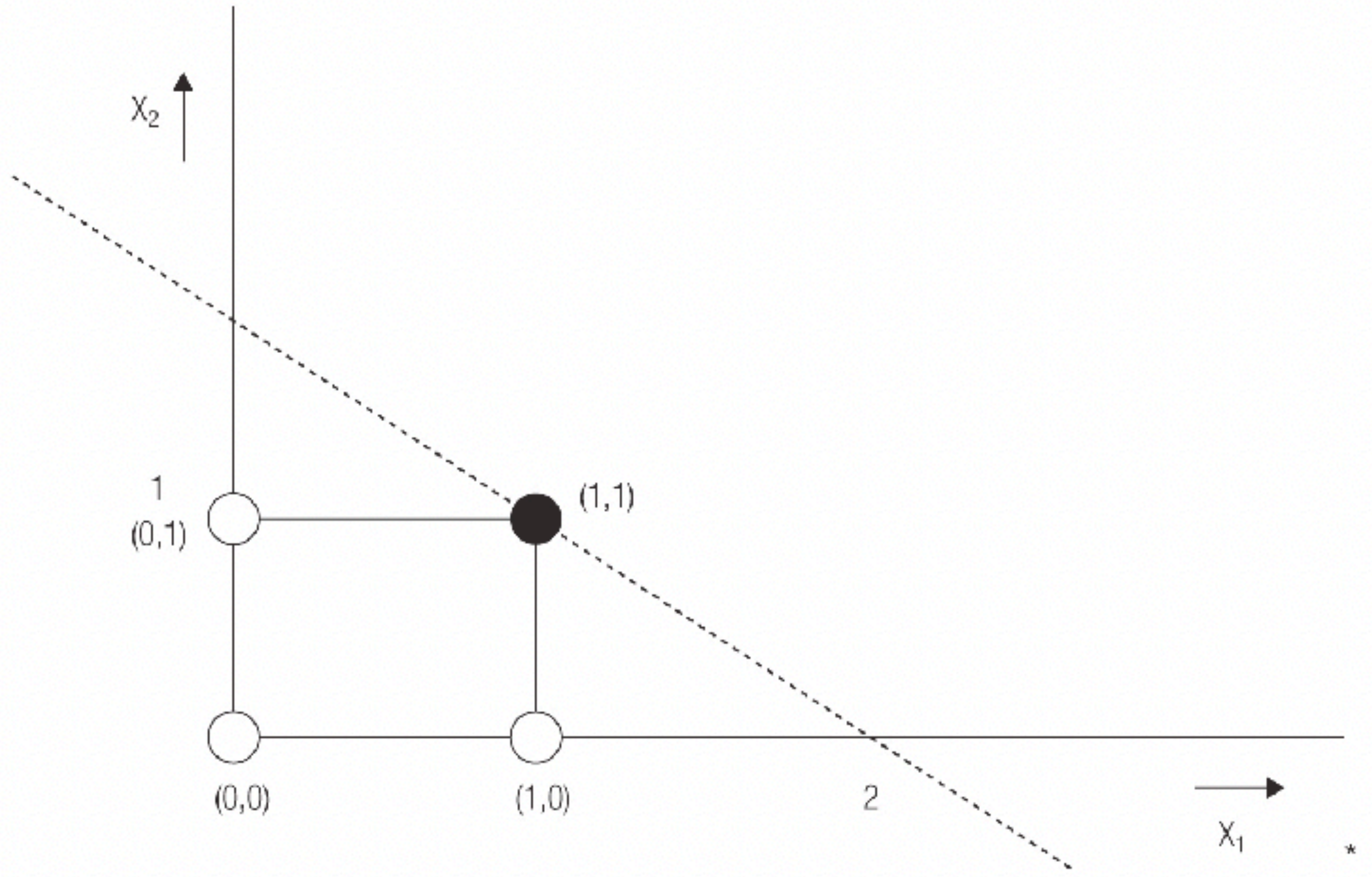


Figure 11.14

The straight line obtained from the TLU in Figure 11.13 when the excitation equals the threshold.

Input/Output Behavior for the TLU

Where x_1 and x_2 are the input and y is the output

x_1	x_2	$\overline{X} \cdot \overline{W}$	y
0	0	[redacted]	
0	1		
1	0		
1	1		

Figure 11.15

Input/output behavior for the TLU in Figure 11.13.

Fill in the columns
Each weight = 0.5

Input/Output Behavior for the TLU

Where x_1 and x_2 are the input and y is the output

x_1	x_2	$\overline{X} \cdot \overline{W}$	y
0	0	0.0	0
0	1	0.5	0
1	0	0.5	0
1	1	1.0	1

Figure 11.15

Input/output behavior for the TLU in Figure 11.13.

Perceptron Learning Algorithm Pseudocode

Inputs:
  X: set of input patterns {x_1, x_2, ..., x_p}
  D: set of desired outputs {d_1, d_2, ..., d_p}
  W: augmented weight vector (randomly initialized)

```
Algorithm:
  // Initialize a variable to check if weights need further updates
  not_all_equal = true

  // Loop until the algorithm converges
  while (not_all_equal) do:
     not_all_equal = false

     // Loop over all input patterns
     for i = 1 to p do:
        // Compute the perceptron output (using the current weights)
        y_i = thetaActivation(W · x_i)  // returns 1 if ≥ theta, 0 if < theta

        // If the output is not equal to the desired output
        if (y_i ≠ d_i) then:
           // Update the weight vector: W = W + ΔW
           W = W + η * (d_i - y_i) * x_i  // η is the learning rate
          // Set flag to true to continue training
           not_all_equal = true

  // If no weights were updated in the loop, the model is trained
  return "TLU has successfully been trained"
```

END