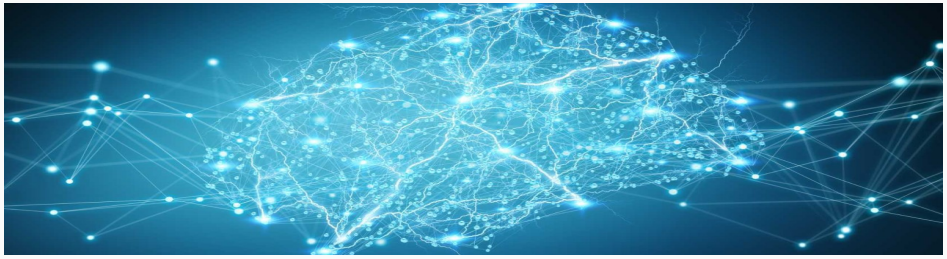


Lecture slides - Week 1

Introduction to ANNs

Dr. Aamir Akbar

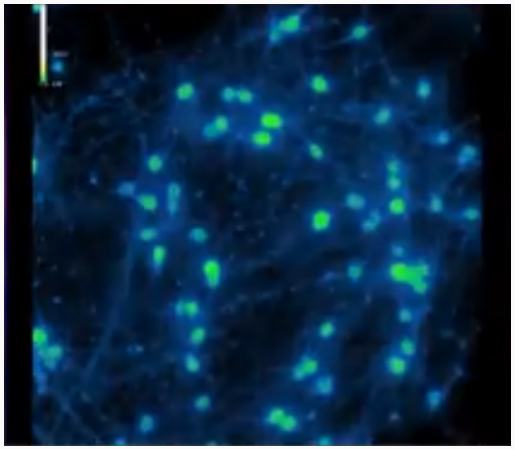
Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

Contents

Introduction

Neurons in Human Brains

The human brain contains billions of neurons, forming complex networks. Neurons are specialized cells that process and transmit information through electrical and chemical signals. `The firing of a neuron` is the fundamental mechanism by which neurons communicate and transmit information in the nervous system.

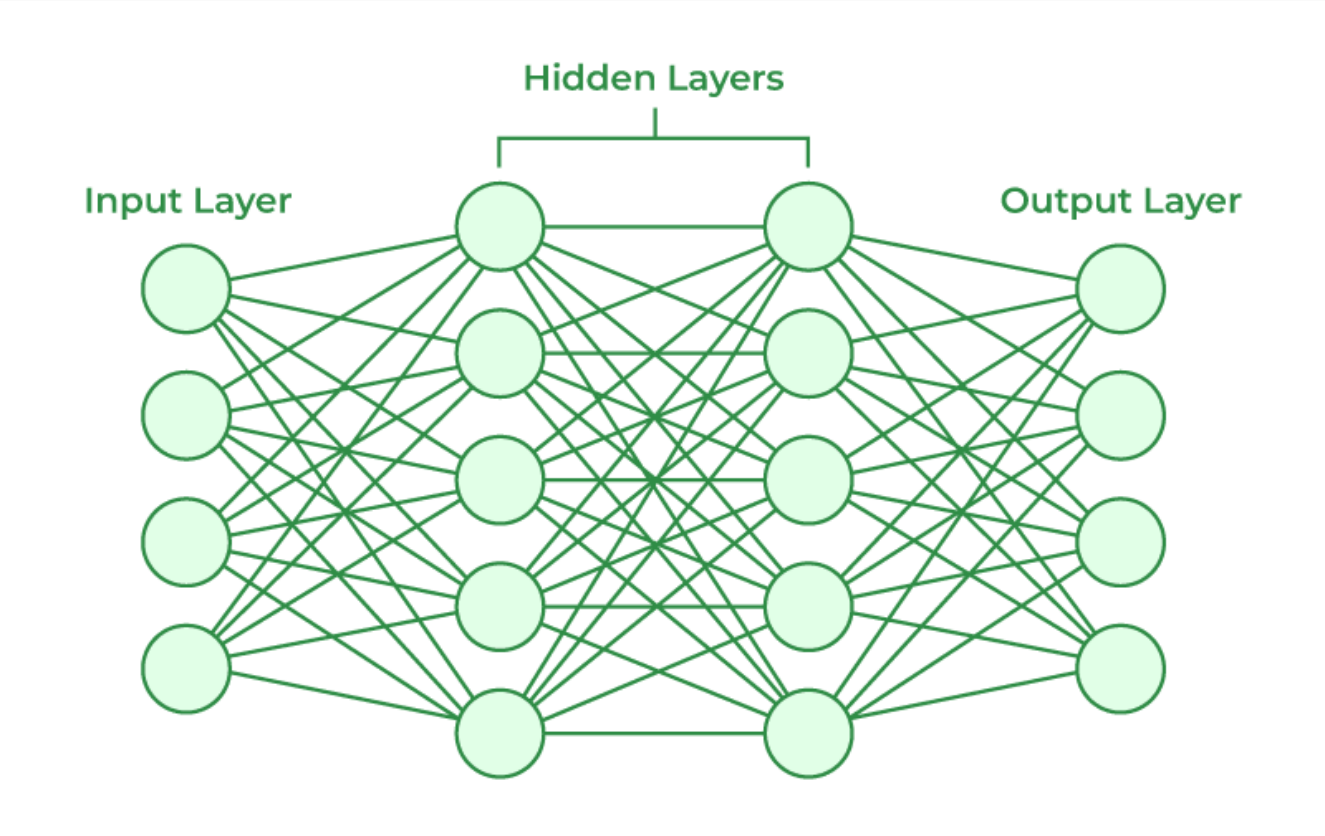


Neurons are responsible for various functions including sensing stimuli from the environment, processing information, and coordinating responses.

Artificial Neural Networks (ANNs) i

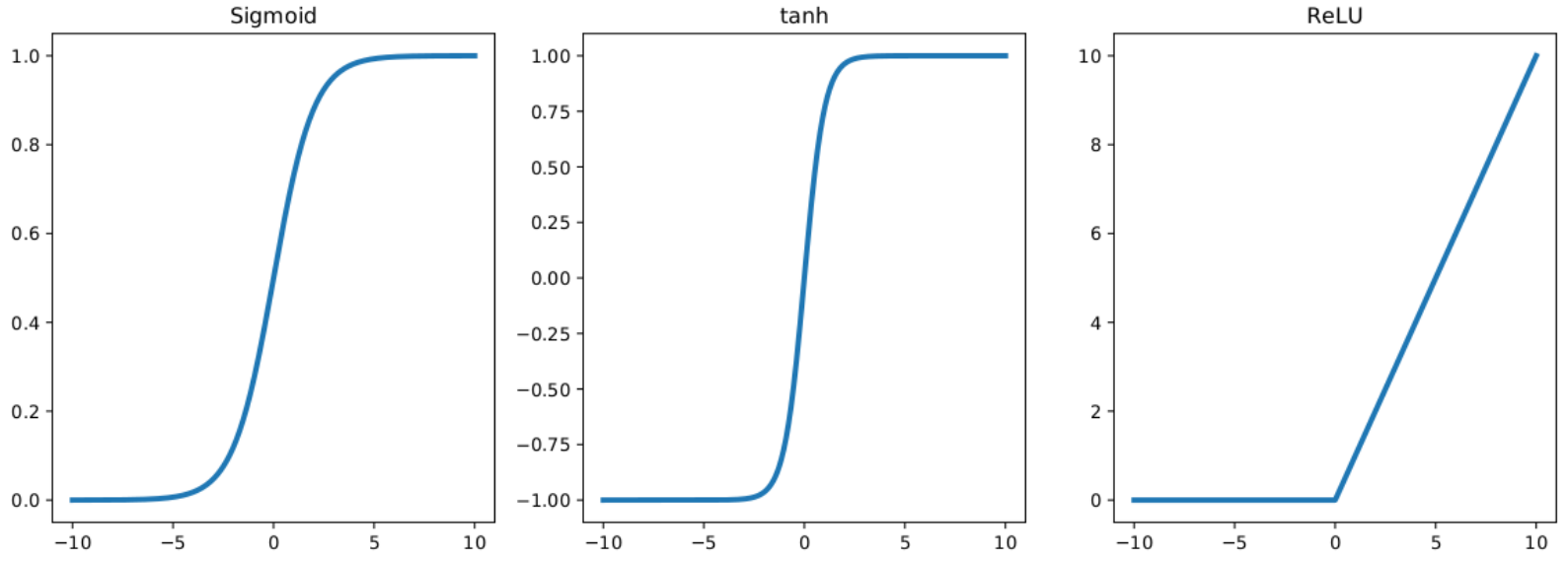
- Artificial Neural Networks (ANNs) are simplified mathematical models inspired by the complex biological neurons in human brains.
- ANNs consist of interconnected nodes called neurons, organized into layers: input layer, hidden layers, and output layer.
- Neurons in ANNs perform simple operations on incoming data, similar to how biological neurons process signals.
- ANNs are trained using algorithms like backpropagation, where errors are propagated backward through the network to adjust the connections (weights) between neurons.
- ANNs have shown remarkable success in various tasks such as image recognition, natural language processing, and game playing.

Artificial Neural Networks (ANNs) ii



Neuron Activation

In ANNs, neuron activation refers to the output generated by an individual neuron in response to its inputs. This activation is determined by applying an `activation function` to the `weighted sum` of the neuron's inputs. Some Examples:

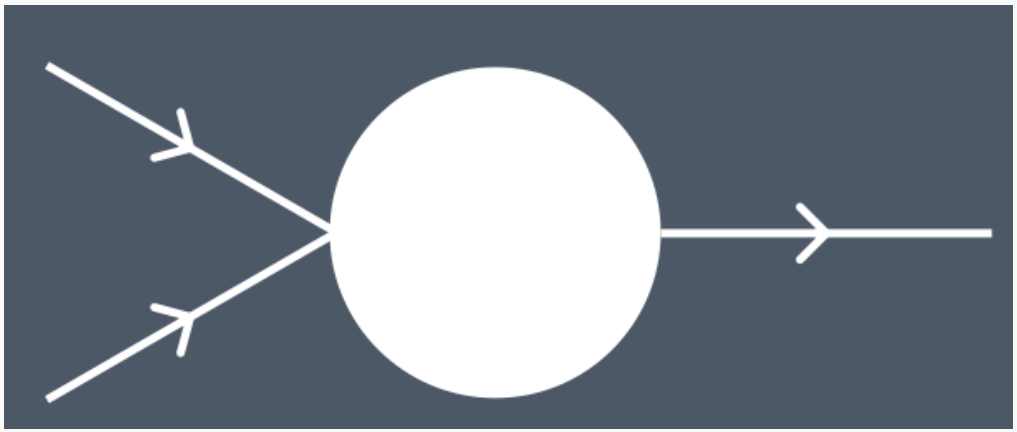


The choice of the activation function depends on the specific application (classification, regression). For instance, `ReLU` is often used in deep learning.

Building ANNs

A Neuron

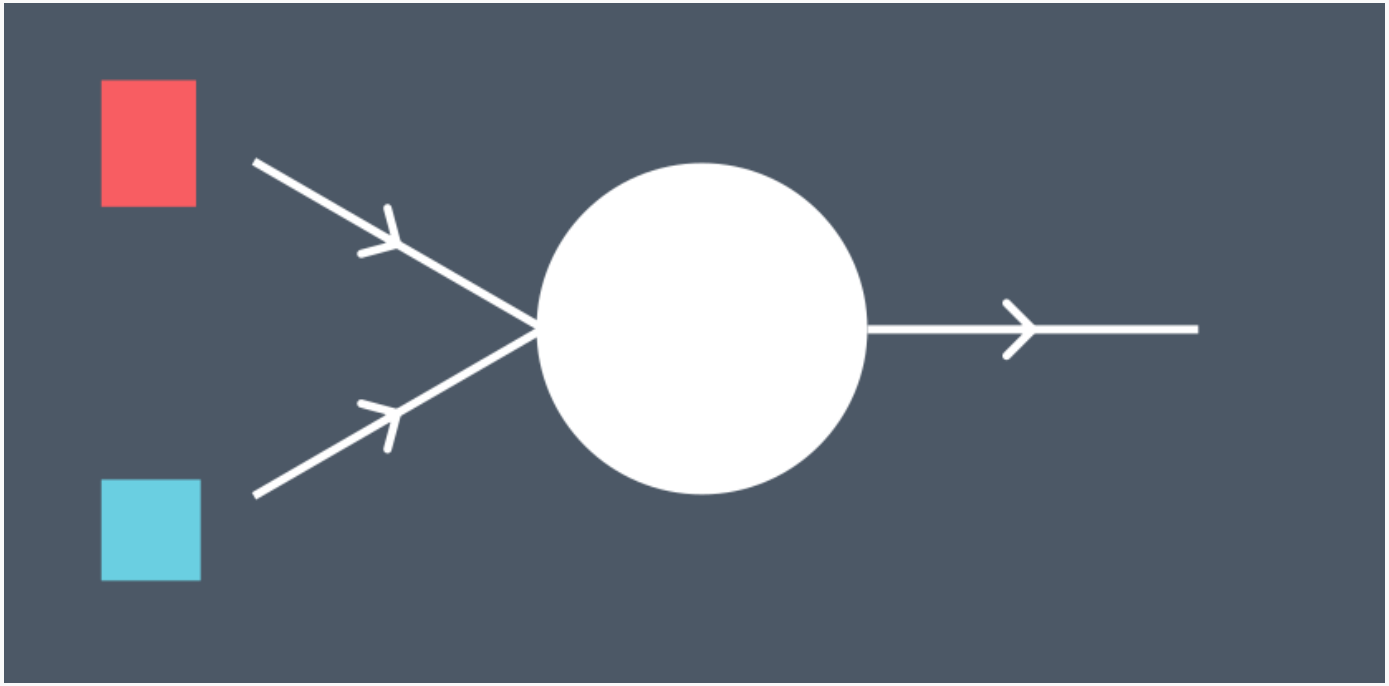
The neuron is the smallest unit and the building block of a neural network.



A neuron takes a set of inputs, performs some mathematical computations, and gives an output.

Inputs to a Neuron

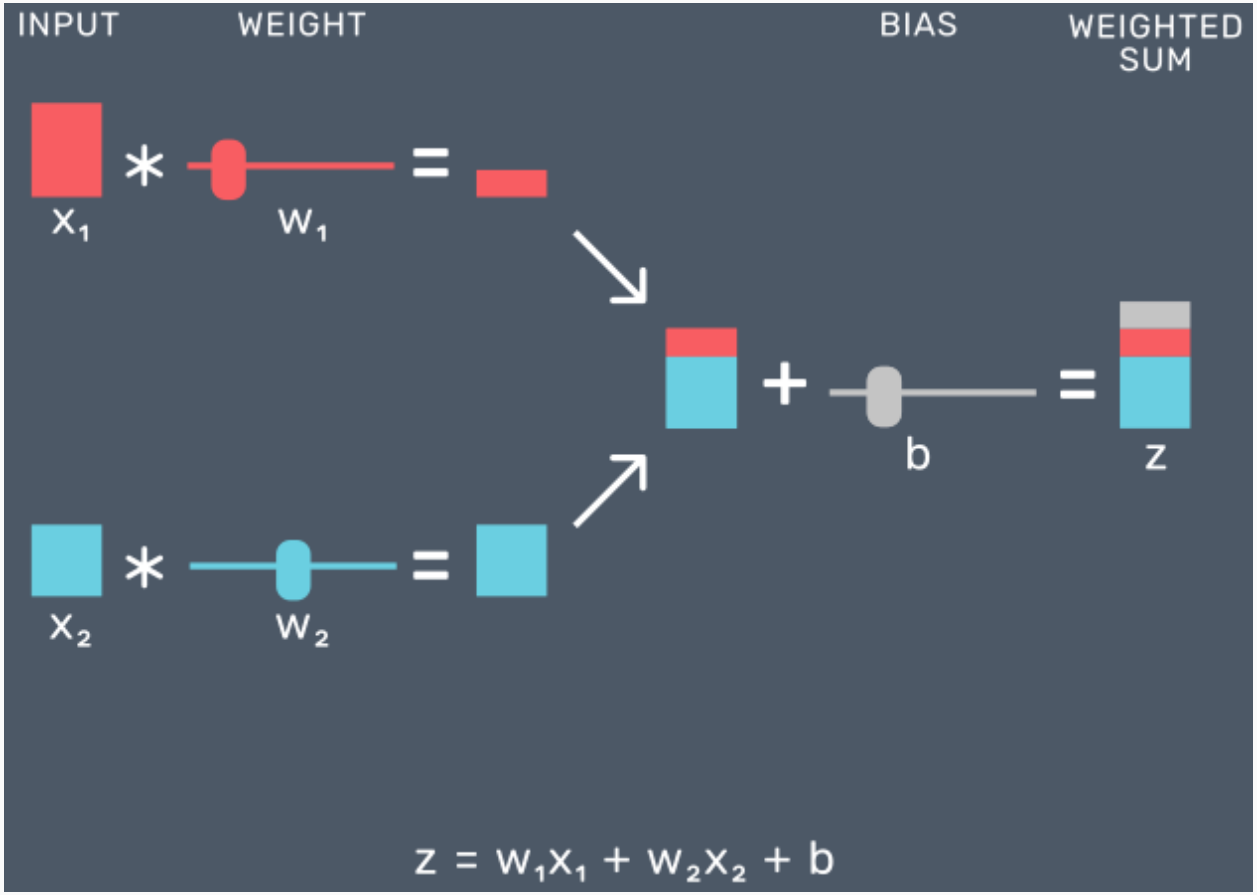
The inputs (and outputs) are numbers, either positive or negative.



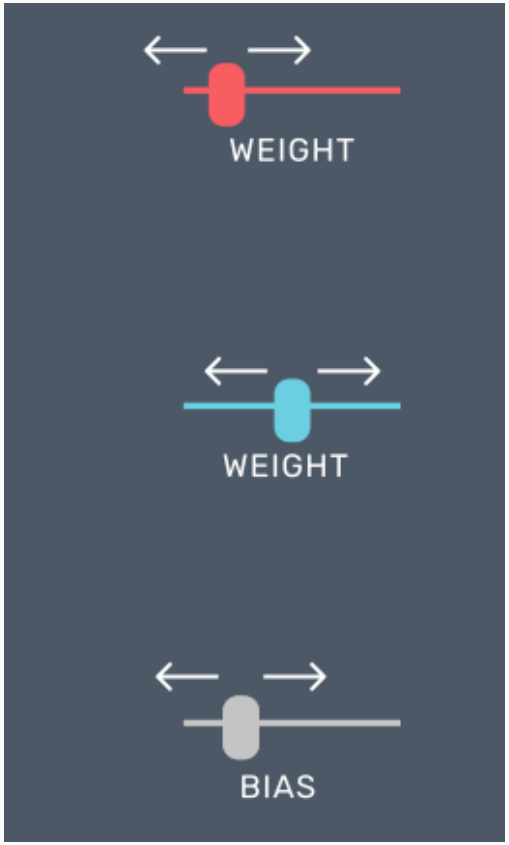
In this example, the neuron takes two inputs. However, there is no limit to the number of inputs a neuron can take.

Weighted Sum - weights and biases i

The first computation that a neuron performs is the `weighted sum`. It multiplies each input by its corresponding `weight`. Then all the inputs are summed and a term called `bias` is added.



Weighted Sum - weights and biases ii

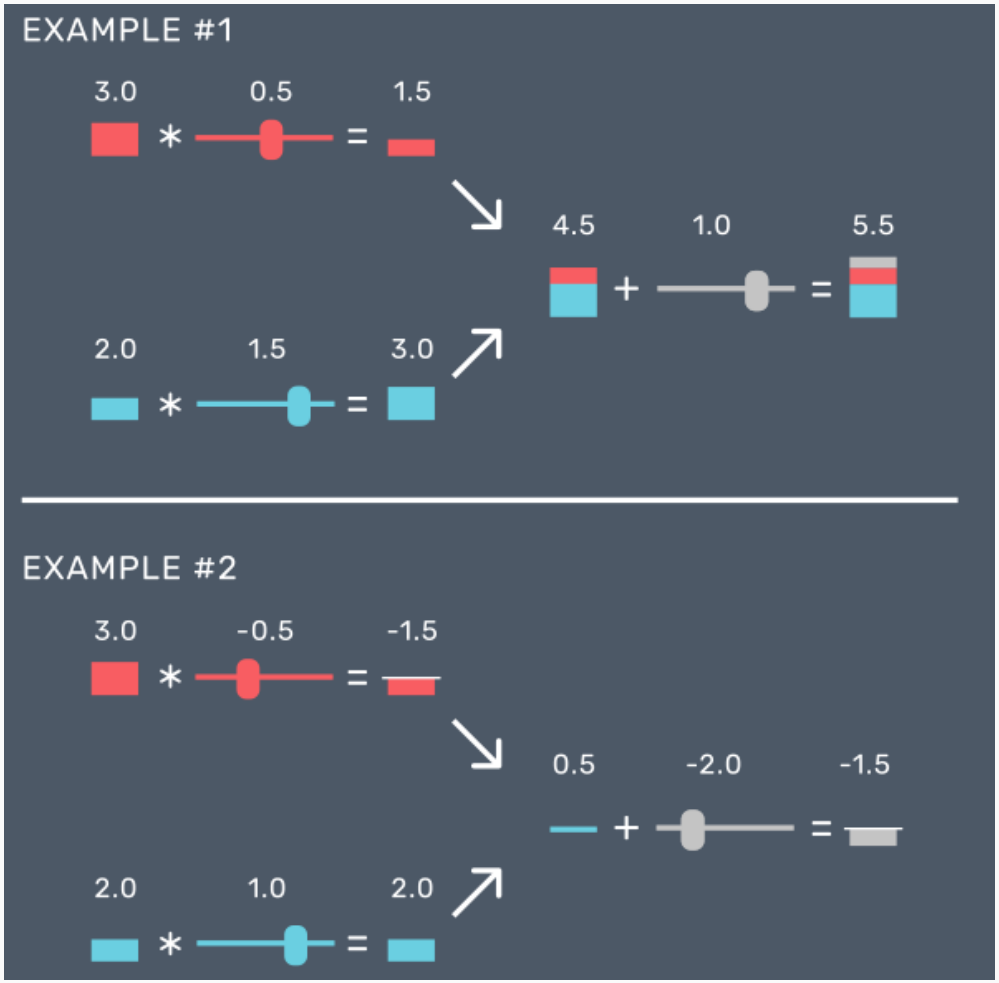


Weights and Biases

These weights and biases are called the `parameters` of the neural network. These adjustable parameters are the medium through which a neural network learns.

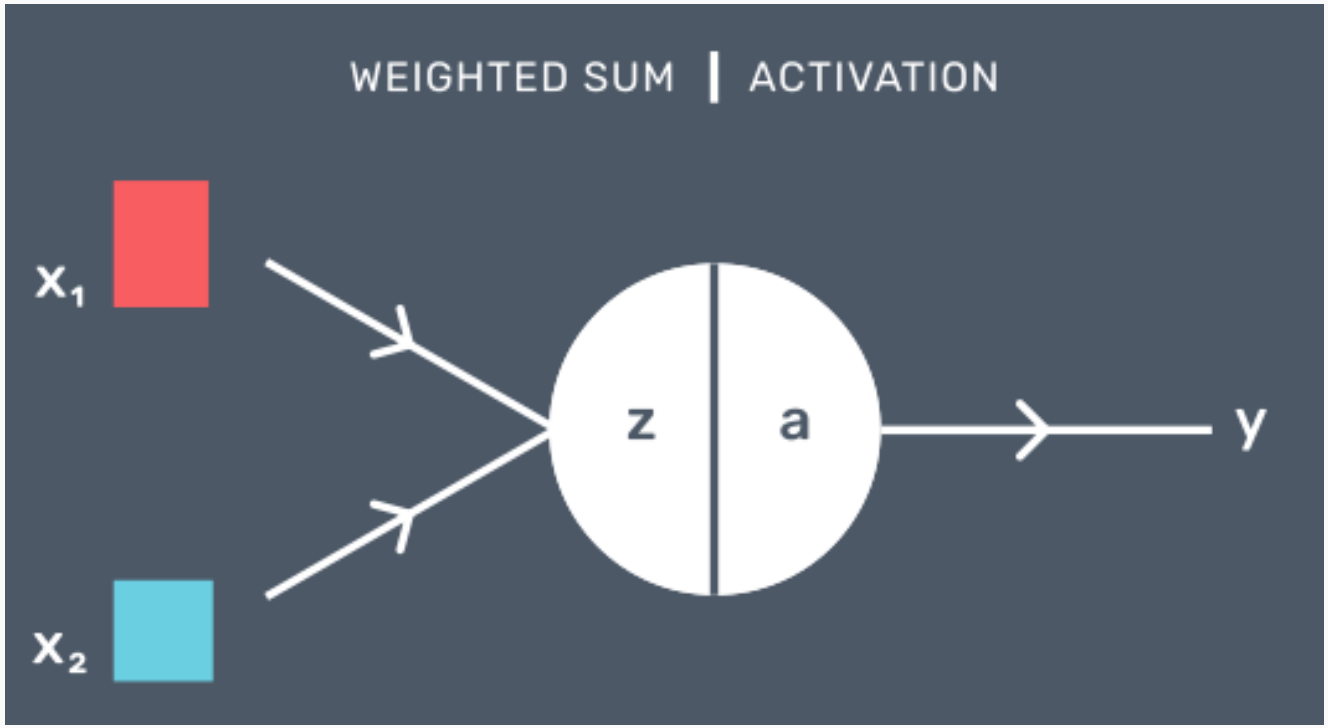
Example

A neuron with two inputs, 3.0 and 2.0 and with different weight values, it will correspondingly output different values.



Activation

The second computation performed by the neuron is called an activation. This is done by taking the output of the `weighted sum` and passing it through an `activation function`.



Linear Regression

What is a Linear Regression i

The purpose of the activation function is to introduce non-linearity into the output of a neuron. Therefore, the non-linear transformation to the input makes a neuron capable of learning and performing more complex tasks.

In other words, without an activation function, a neuron (or a neural network) is just a Linear Regression model.

Linear Regression Model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$

Where:

- y is the dependent variable (target).
- $x_1, x_2, ..., x_n$ are the independent variables (features).
- $\beta_0, \beta_1, \beta_2, ..., \beta_n$ are the coefficients (parameters) to be estimated.
- ε is the error term that represent the difference between the observed values of the dependent variable (y) and the values predicted by the model.

What is a Linear Regression ii

Linear Regression Model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

A Neuron Without Activation Function:

$$y = b + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

- Both models involve linear transformations of input features $(x_1, x_2, ..., x_n)$.
- In linear regression, the relationship between y and the independent variables is assumed to be linear, limiting its ability to capture complex patterns.

What is a Linear Regression iii

- Without activation functions, a neural network collapses into a series of linear transformations, unable to model non-linear relationships or interactions in the data.
- Linear regression is suitable for tasks where relationships are linear, while a neural network with activation functions can learn non-linear mappings, making it more adept at handling complex patterns and non-linearities.

McCulloch-Pitts (M-P) Model

What is McCulloch-Pitts model of a neuron? i

The McCulloch-Pitts (M-P) model of a neuron, proposed by Warren McCulloch and Walter Pitts in 1943, is one of the earliest mathematical models of biological neurons.

Key Components:

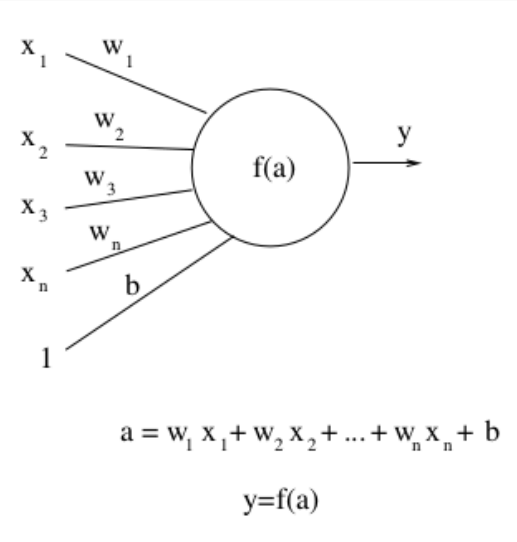
- Neurons are represented as binary threshold units.
- Inputs to the neuron are binary (0 or 1).
- The neuron sums the weighted inputs and compares the sum to a threshold.
- If the sum exceeds the threshold, the neuron outputs 1; otherwise, it outputs 0.

What is McCulloch-Pitts model of a neuron? ii

Mathematical Modeling of M-P: Let $x_1, x_2, ..., x_n$ be the binary inputs to the neuron, $w_1, w_2, ..., w_n$ be the corresponding weights, b be the bias term, and θ be the threshold.

The output y of the neuron is computed as:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n}(w_i x_i) + b \geq \theta \\ 0 & \text{otherwise} \end{cases}$$



Python Implementation

```python
def mcculloch_pitts_neuron(inputs, weights, bias, threshold):
    # Compute the weighted sum of inputs
    weighted_sum = sum(w * x for w, x in zip(weights, inputs))

    # Add bias term
    weighted_sum += bias

    # Compute the output of the neuron based on threshold
    if weighted_sum >= threshold:
        return 1
    else:
        return 0

# Example usage:
inputs = [1, 0, 1]
weights = [0.5, -0.3, 0.8]
bias = 0.2
threshold = 0.5

output = mcculloch_pitts_neuron(inputs, weights, bias, threshold)
print("Output of the neuron:", output)
```

Perceptron

Perceptron

The McCulloch-Pitts (M-P) model and the perceptron both represent early conceptualizations of artificial neurons. The M-P model laid the groundwork for subsequent developments in neural network theory but had limitations, such as its binary nature and lack of a learning mechanism.

The perceptron, proposed by Frank Rosenblatt in 1957, is a specific implementation of the M-P model with some enhancements:

- The perceptron introduces real-valued weights instead of binary weights used in the M-P model.
- It uses a different activation function, typically a step function, to determine the output based on the weighted sum of inputs.
- The perceptron is designed to learn from data using a supervised learning algorithm called the perceptron learning rule, which adjusts the weights based on errors in the output.

Algorithm: Training with Perceptron Learning Rule

1: **Initialization:** Initialize weights (w_i) to small random values or zero.
2: **Iterative Training:**
3: **for** each input pattern **do**
4: Compute the weighted sum of inputs ($\sum w_i x_i$) and apply activation function.
5: **if** output matches desired target output **then**
6: No weight adjustments are made.
7: **else**
8: Adjust weights to reduce error:
9: Update weights using perceptron learning rule:
10: $w_i \leftarrow w_i + \eta \times (target - output) \times input_i$
11: **end if**
12: **end for**
13: **Repeat** until convergence or fixed number of iterations.
14: **Classification:** Once trained, classify new input patterns.

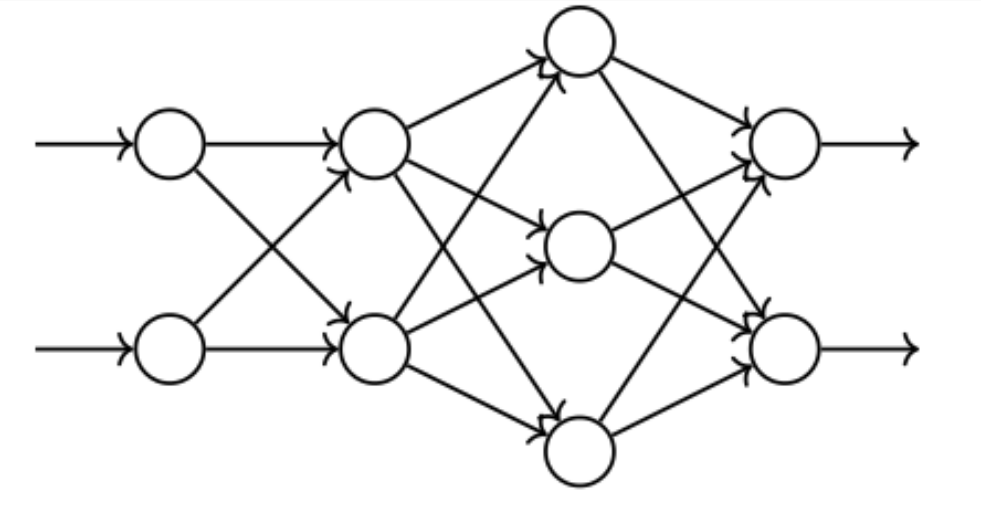
Perceptron from Scikit-learn (a.k.a `sklearn`)

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score

# Declare random_state variable
random_state_value = 100

# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, \
random_state=random_state_value)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, \
random_state=random_state_value)

# Initialize and train the perceptron model
perceptron = Perceptron(random_state=random_state_value)
perceptron.fit(X_train, y_train)

# Predictions on test set
y_pred = perceptron.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Feed Forward Neural Network

What is a Feed-Forward Network? i

A `feed-forward neural network` arises by combining various perceptrons by feeding the outputs of a series of perceptrons into a new perceptrons.



We interpret a neural network as consisting of different layers. The first layer is the input layer, while the last layer is the output layer. The layers between the input and output layer are called the hidden layers.

What is a Feed-Forward Network? ii

Linear Transformation: For the `k-th` layer of the network, a linear transformation $W^k : \mathbb{R}^{d_{k-1}} \rightarrow \mathbb{R}^{d_k}$ is applied to the input vector x, where d_{k-1} is the dimensionality of the input features to the `k-th` layer, and d_k is the dimensionality of the output features of that layer. This linear transformation is represented by a weight matrix W^k. Each row of W^k corresponds to a neuron in the current layer, and each column corresponds to the weights associated with the incoming connections from the previous layer.

Bias Addition: Additionally, a bias vector $b^k \in \mathbb{R}^{d_k}$ is added to the result of the linear transformation $W^k x$. This bias vector allows the network to shift the transformed features, providing flexibility in fitting the data.

What is a Feed-Forward Network? iii

Activation Function: After the linear transformation and bias addition, an activation function σ is applied element-wise to the result as:

$$\sigma(W^k x + b^k). \tag{1}$$

The purpose of the activation function is to introduce non-linearity into the network, allowing it to learn and approximate complex relationships in the data.

A neural network with $\mathcal{L}$ layers is a function of the form $F^{\mathcal{L}}(x)$, where the F^k are recursively defined as:

$$F^1(x) = \sigma(W^1 x + b^1) \tag{2}$$

$$F^{k+1}(x) = \sigma(W^{(k+1)} F^k(x) + b^{k+1}) \tag{3}$$

and $W^k \in \mathbb{R}^{d_k \times d_{k+1}}$, $b^k \in \mathbb{R}^{d_k}$, for $1 \leq k \leq \mathcal{L}$.

Resources

Resources

To download the source codes used in the previous slides, follow the link:

▸ Download Resources

Import the codes into your preferred development environment, such as Visual Studio Code (VS Code), to practice and explore further.

To learn programming in Python, follow my comprehensive 15-week Programming in Python course at:

▸ Programming in Python