Embedded Artificial Intelligence

1. Introduction:

1.1 Overview of Artificial Intelligence:

Artificial intelligence (AI) refers to computer systems capable of performing complex tasks that historically only humans could do, such as reasoning, speech recognition, visual object identification, and detection.

It's the broad term that defines programs with the ability to learn and reason like humans, which is the general term that encompasses a wide variety of technologies, including machine learning, deep learning, and natural language processing (NLP).

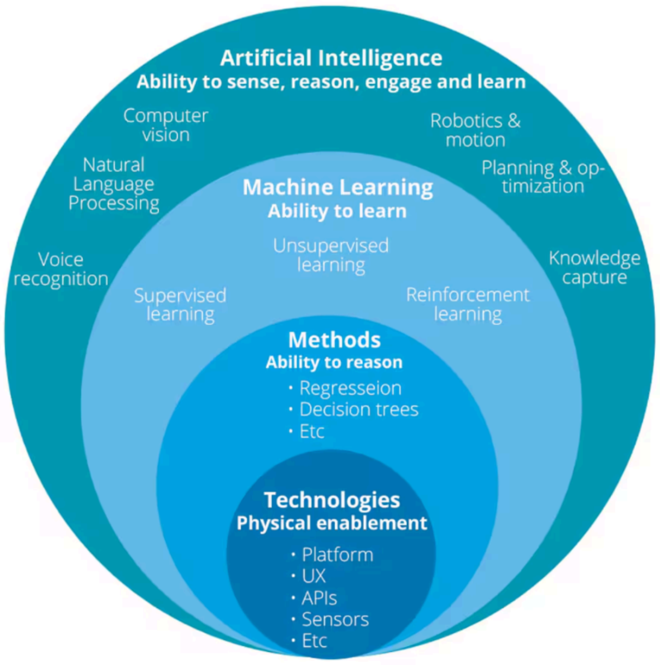


Figure 1.1: Diagram of Artifical Intelligence

The key components of AI applications are data, algorithms, and human feedback.

➢ Data is used to train the mathematical models, which are considered the magic black boxes that take an input, which can be sound, picture, or even a number, and predict an output detection or classification.

➢ Algorithms are the methods used to use the data to train the model to predict the right outputs, as initially, models tend to predict random outputs.

➢ Finally, human feedback is the true learning component, as without it, the model can't know if its output is correct or not.

1.2 Introduction to Embedded Systems:

Embedded systems are computer-based systems designed to perform a specific task or set of tasks, often within a larger system or device. It’s a microcontroller or a microprocessor-based system, often with real-time computing constraints and low computing resources, important for low power usage.

Embedded systems are all over the world; they're used in a wide range of applications, including automotive, industrial automation, consumer electronics, telecommunications, and aerospace.

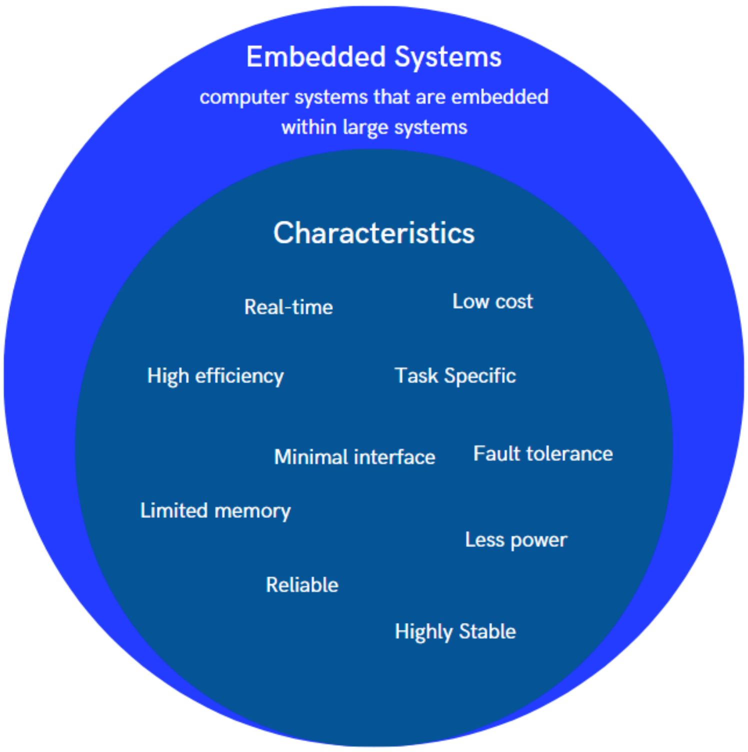


Figure 1.2: Diagram of Embedded Systems

2. Embedded Artificial Intelligence:

2.1 Definition and Scope:

Embedded Artificial Intelligence (EAI) refers to the integration of AI capabilities, particularly machine learning and deep learning, directly into small-scale devices, respecting the characteristics of an embedded system such as real-time constraints, low computing resources, and low power consumption.

This technology allows these devices to perform complex tasks autonomously without relying on constant internet connectivity or cloud computing.

The key components of embedded AI are sensors, microprocessors or microcontrollers, embedded software, and a user interface.

- Sensors are used as a gateway for the physical space; they are the data generator in this model. Sensors are used to collect data such as images, sounds, and any physical entity.
- Microprocessors, or microcontrollers, are used to process the data collected by sensors and run AI algorithms.
- Embedded software includes algorithms, machine learning libraries, and codes designed to run on embedded systems. The job of these algorithms is to take an input from a sensor and produce an action.

2.2 Why We Need Embedded AI:

The need for embedded AI comes from the characteristics of the embedded system itself; the most important is the real-time decision. The problem with centric AI, which means using centric servers and internet connectivity to run AI models, breaks the real-time notion, as networking algorithms and operating systems running on the cloud are non-predictive in nature.

Also, these solutions tend to enhance performance, improve resource efficiency, and improve general system security.

Examples:

- Enhanced automation: AI algorithms can learn to automate routine tasks and processes. Freeing up human operators to focus on more complex or strategic tasks.

- Predictive Maintenance: Embedded AI can perform analysis on stored data to identify scenarios. Wherein a failure might occur before the actual failure happens.

- Advanced, near--real-time signal processing methods: Embedded AI requires knowledge of devices, sensors, and advanced, near real-time signal processing methods for voice, audio, motion, or other signals.

2.3 Real-world Applications and Examples:

Amazon Echo: The Amazon Echo smart speaker uses an embedded AI system to process voice commands and provide responses. The Alexa Voice Service, which powers the Echo, runs on a low-power ARM-based microcontroller that is always listening for the “Alexa” wake word. When triggered, Alexa uses natural language processing and machine learning models to understand the user’s request and provide an appropriate response.

Figure 2.1: Amazon Echo

Tesla Autopilot: Tesla’s Autopilot system uses embedded AI to process data from cameras, radar, and ultrasonic sensors to enable features like traffic-aware cruise control, lane keeping, and automatic emergency braking. The AI algorithms run on Tesla’s proprietary Full Self-driving Computer, which is embedded directly into the vehicle.

Figure 2.2: Tesla Autopilot

Apple Watch Series 6: The Apple Watch Series 6 incorporates and embeds AI for features like fall detection and irregular heart rhythm notifications. The watch's accelerometer and heart rate sensor data is processed by an embedded machine learning model to detect potential health issues and alert the user to emergency contacts if necessary.

3. Neural Networks:

3.1 Definition:

A neural network, also known as an artificial neural network (ANN) or neural net, is a type of machine learning model inspired by the structure and function of biological neural networks in the human brain. It's a computational model that consists of interconnected nodes, or artificial neurons, which loosely model the neurons in a brain.

Key components of artificial neural networks are the artificial neurons, edges or weights, layers, and activation functions.

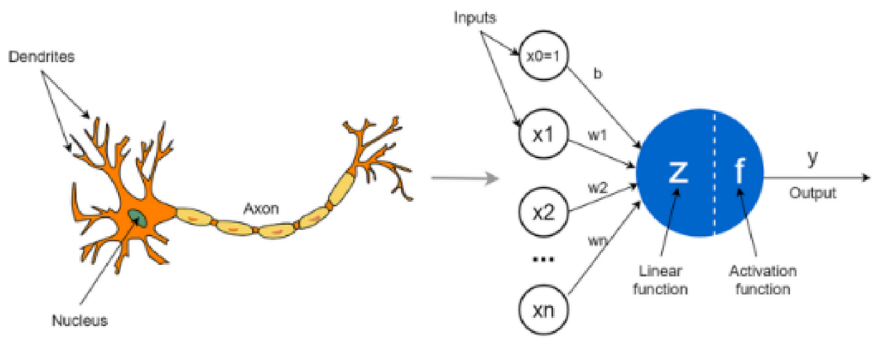
3.2 A Single Artificial Neuron:

A single artificial neuron is a mathematical model or function that takes an input, does a specific calculation based on weights and adds a bias

A neuron can be represented by a mathematical function as follows:

$$z = \sum_{i=1}^{inputs} w_i x_i + b$$

Here is a picture that illustrates such a computation:



And here is a simple Python code that implements such a model using NumPy:

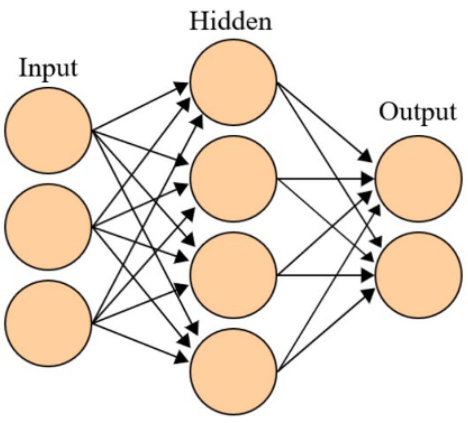
```python
import numpy as np

inputs = [1.0, 2.0, 3.0, 2.5]
weights = [0.2, 0.8, -0.5, 1.0]
bias = [2.0]

output = np.dot(inputs, weight) + bias
print(output)
```

3.3 Neural Network Layers:

Neural networks typically consist of multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer processes that data and sends it to the next layer.



We can use the same code as presented before since Numpy will take care of the multiplications; we just need to change the weights. Instead of being a single list, it will be a matrix, with each row representing the weights of a single neuron. So the total number of neurons will be equal to the number of rows of the matrix.

```python
import numpy as np

inputs = [1.0, 2.0, 3.0, 2.5]
weights = [
    [0.2, 0.8, -0.5, 1],
    [0.5, -0.91, 0.26, -0.5],
    [-0.26, -0.27, 0.17, 0.87]
]
biases = [2.0, 3.0, 0.5]

layer_outputs = np.dot(weights, inputs) + biases
print(layer_outputs)
```

3.4 Activation Functions:

The activation function is a mathematical function used in neural networks that can map the result of the weight and bias calculation into a controlled and bounded number.

3.4.1 Why Use Activation Functions?

- ➢ For a neural network to fit nonlinear function, we need it to contain two or more hidden layers.
- ➢ And we need those hidden layers to use a nonlinear activation function
- ➢ No matter how many layers we have, any network can only depict linear relationships if we use linear activation functions.

3.4.2 Different Activation functions

Linear Activation | Sigmoid Activation | ReLU (Rectified Linear Unit) Activation SoftMax Activation | Step Activation.

3.5 Loss Functions:

To train a neural network model, we tweak the weights and biases that connect neurons to improve the model's accuracy and confidence in predicting the expected value.

We use the loss function, also referred to as the cost function, as a metric to quantify how badly the model predicted what we expected.

The choice of the loss function depends on the type of problem we are solving. If we are solving a regression machine learning problem, then a **mean squared error (MSE) is** used. If the problem is a classification problem, we will be using the **cross-entropy loss function.**

A loss function can be easily implemented using any programming language; for example, the mathematical definition of the mean squared error is the following:

$$C = \frac{1}{n} \sum_{i=1}^{n} (\hat{y} - y)^2$$

A simple Python implementation of such a function can be done as follows:

```python
class Cost:
    # Calculate the cost depending on the loss function defined by the 
forward_pass method.
    def calculate(self, y_pred, y_true):
        # Get the sample losses array
        sample_losses = self.forward_pass(y_pred, y_true)
        # Average the losses values to get the cost value
        data_losses = np.mean(sample_losses)
        return data_losses

class LossMeanSquareError(Cost):
    # Forward pass
    def forward_pass(self, y_pred, y_true):
        # Calculate squared errors for each sample
        squared_errors = np.square(y_true - y_pred)
        return squared_errors
```

3.6 Backpropagation:

3.6.1 Definition

Backpropagation is a gradient-based optimization algorithm used to train deep learning neural networks.

It efficiently computes the gradients of a loss function with respect to the network's parameters (weights and bises) in order to update them.

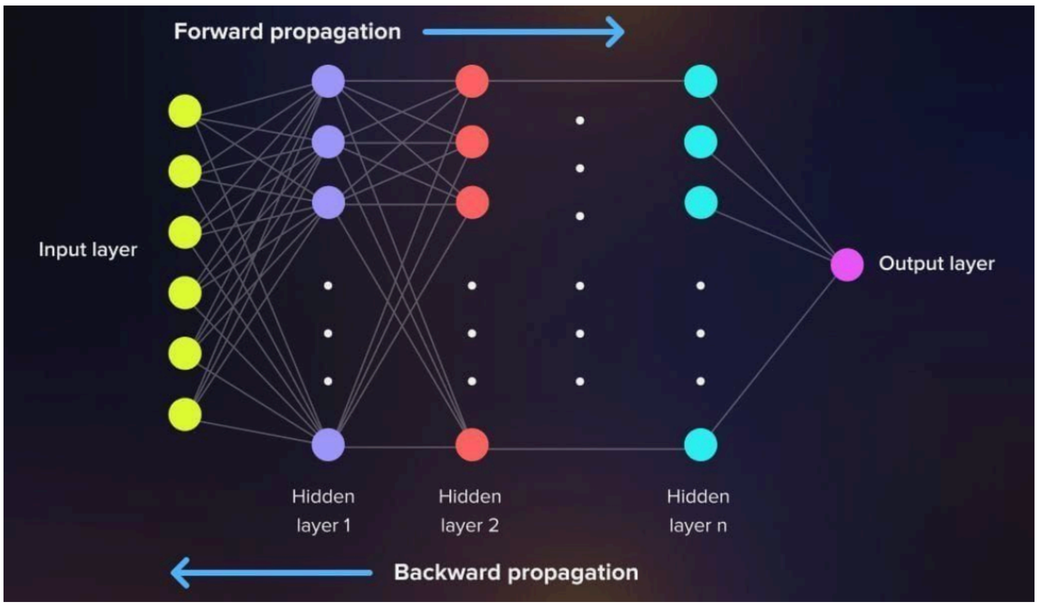
3.6.2 How Does Backpagation Work?

Backpropagation involves two passes through the different layers of a neural network. At first, a forward pass will be done to calculate the prediction based on a supervised input.

Then, we calculate the error, which will be done by the defined loss function for the network.

Then, going backward at each layer, we calculate the gradients, which are the partial derivatives of the loss of the previous layer with respect to the layer's weights and biases.

The resulting gradients will update the current layer and be passed to the previous layer to do its calculations.


Forward propagation
Input layer
Output layer
Hidden layer 1
Hidden layer 2
Hidden layer n
Backward propagation