Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant

Introduction to Deep Learning

Basics of neural networks, TensorFlow, and Keras

AGENDA

1. Intro to Deep Learning
2. Neural Networks
3. Working of Neural Networks
4. Intro to TensorFlow and Keras

If you were given a million dollars but only 24 hours to spend it, what would you do?

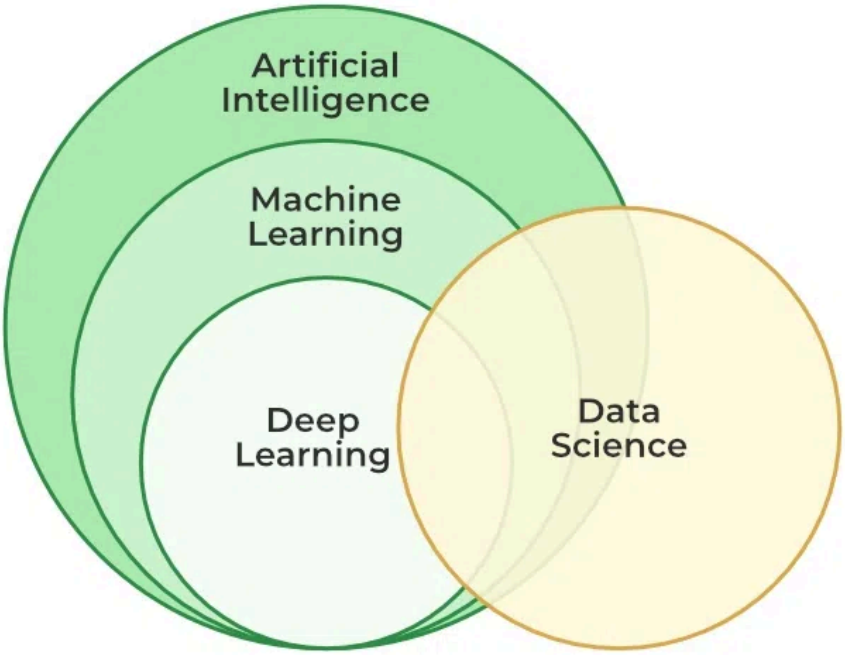
1. Intro to Deep Learning

Deep Learning is a subset of Machine Learning (ML) that uses artificial neural networks(ANN) to **simulate the way humans learn** from **large amounts** of **data**.

It is called "deep" because these networks consist of multiple layers, allowing the system to learn complex patterns.

Relationship with Machine Learning and AI:

- **AI**: The broader concept of machines performing tasks that require human intelligence.
- **Machine Learning**: A subset of AI that involves training algorithms to make predictions without being explicitly programmed.
- **Deep Learning**: A subset of ML focused on learning from data using neural networks with multiple layers.



Differences Between Machine Learning (ML) and Deep Learning (DL)

Aspect	Machine Learning (ML)	Deep Learning (DL)
Level of Complexity	Relies on manual feature engineering and simpler models.	Uses deep neural networks to learn directly from data.
Data Requirements	Performs well with small to medium datasets.	Requires large datasets for optimal performance.
Feature Engineering	Manual feature extraction by domain experts.	Features are automatically extracted during training.
Computational Power	Works on regular CPUs; requires less computational power.	Needs GPUs/TPUs due to high computation demands.
Algorithm Structure	Includes decision trees, SVMs, regression, etc.	Uses CNNs, RNNs, transformers, and deep architectures.
Applications	Structured data tasks like fraud detection or segmentation.	Unstructured data tasks like image and speech recognition.
Training Time	Typically faster to train.	Longer training times due to model complexity.
Interpretability	Models like decision trees are interpretable.	Operates as a "black box" with limited interpretability.
Hardware Dependency	Can run on traditional hardware setups.	Requires specialized hardware like GPUs/TPUs.
Performance	Good for smaller datasets and simple tasks.	Excels with large, complex datasets and unstructured data.

Real-World Applications of Deep Learning

1. **Image Recognition:**

Images are made up of millions of pixels, forming high-dimensional data.

Deep learning, specifically Convolutional Neural Networks (CNNs), can efficiently detect intricate patterns, edges, and shapes by learning hierarchical features directly from the raw pixel data.

Examples:

- Face detection in smartphones.
- Object detection in self-driving cars.

2. **Natural Language Processing (NLP):**

Deep learning models like Recurrent Neural Networks (RNNs) or Transformers (e.g., BERT, GPT) are designed to handle sequential data and understand context across words or sentences.

This enables tasks like converting spoken language into meaningful responses.

Examples:

- Voice assistants like Siri and Alexa.
- Machine translation (e.g., Google Translate).

Current Groundbreaking Work in Deep Learning

1. GPT-4 and Generative AI Models (NLP)

- Models like ChatGPT are revolutionizing how humans interact with machines by holding human-like conversations, drafting content, and solving problems.

2. Neuralink and Brain-Computer Interfaces (Healthcare and Neural Engineering)

- Neuralink, founded by Elon Musk, is developing devices that enable direct communication between the human brain and computers.

3. AlphaFold (Healthcare and Drug Discovery)

- Developed by DeepMind, AlphaFold predicts protein structures with remarkable accuracy, solving a 50-year-old grand challenge in biology. Its insights are accelerating drug discovery and enabling scientists to understand diseases at a molecular level, offering breakthroughs in treating conditions like cancer and Alzheimer's.

4. Tesla Autopilot (Autonomous Driving)

- Tesla's Autopilot system employs DL to analyze camera feeds, radar, and other sensor inputs for real-time decision-making in self-driving vehicles. It's making

strides toward safer transportation, reducing accidents, and paving the way for fully autonomous fleets.

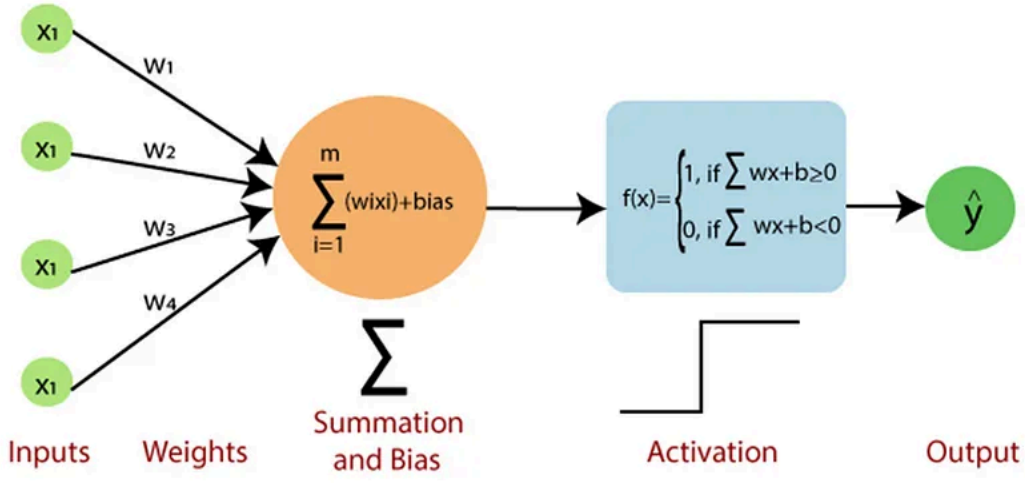
Similarly:

- Deepfake Detection (Cybersecurity)
- Climate Change Mitigation (Environmental Science)
- AI in Agriculture (AgTech)
- AI in Space Exploration (Aerospace)
- Multi-Modal AI Models (models combine data from different modalities—such as text, images, and video—into a single model,)
- Federated Learning (Privacy-Preserving AI)(models are trained across decentralized devices or servers without needing to share sensitive data between them)
- Neuromorphic Computing (Hardware Development) (design computer chips that mimic the human brain's architecture)

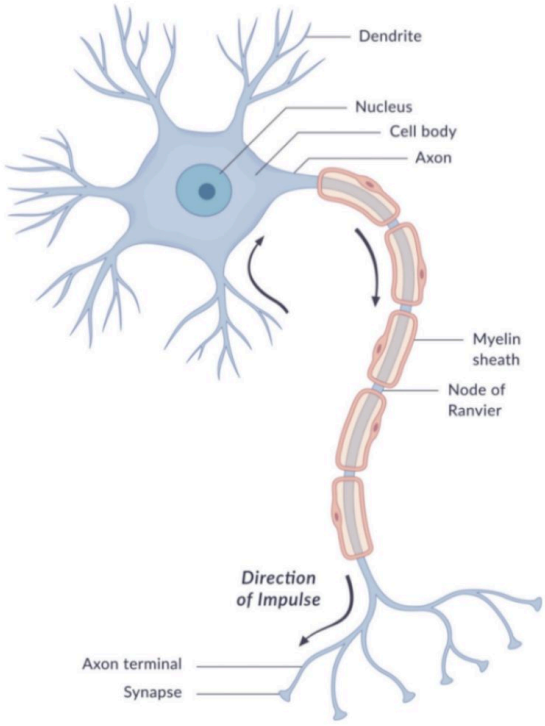
2. Neural Networks

A neuron in a neural network mimics a biological neuron in the brain. It is the basic unit that processes information.

A perceptron, the simplest type of artificial neural network (ANN) unit:



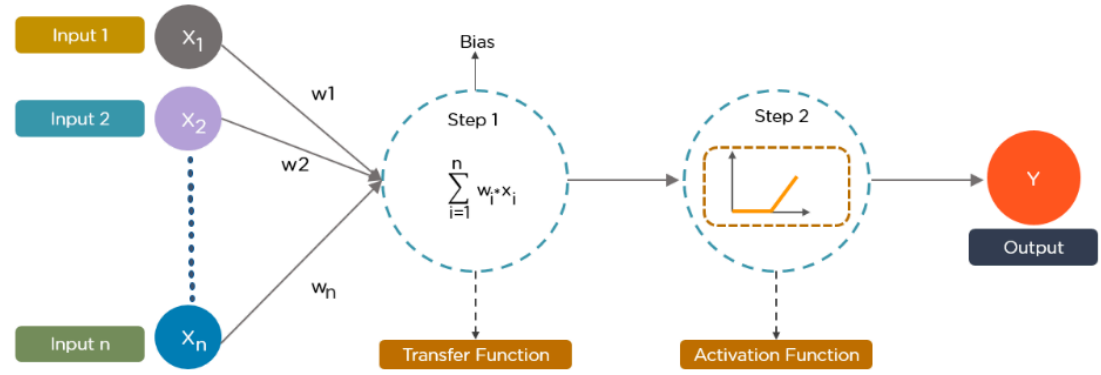
Here is how a human brain cell (Neuron) looks like:

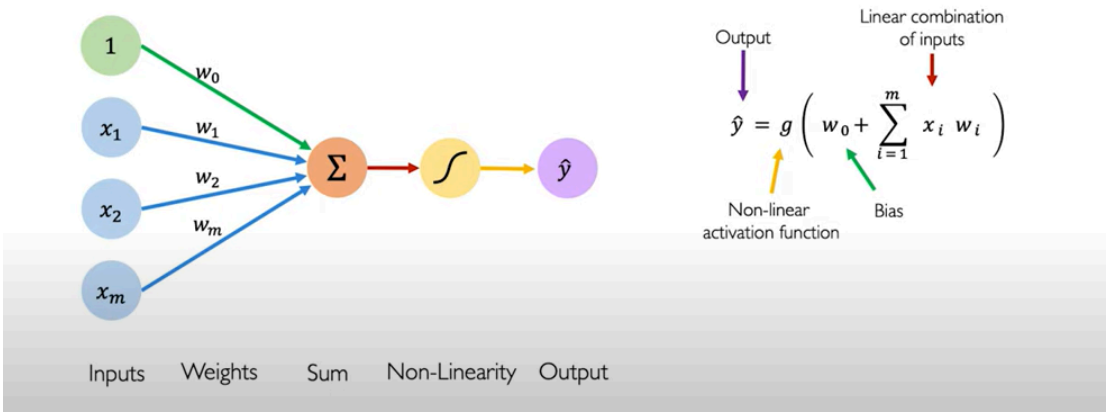


Neuron: The Building Block of a Neural Network

Perceptron - W3 Schools

A neuron in a neural network mimics a biological neuron in the brain. It is the basic unit that processes information. Let's break it down step-by-step:





Inputs:

- A neuron receives information in the form of **inputs** (e.g., features like pixel values in an image or numerical data in a spreadsheet).

Weights and Bias:

- **Weights** determine how important each input is. A higher weight gives more significance to that input.
- **Bias** is like a threshold that helps the neuron adjust how much influence the input has.

Processing: The neuron multiplies each input by its weight, adds the bias, and then sums up these values:

$z = (w_1 \cdot x_1) + (w_2 \cdot x_2) + \ldots + b$

Activation Function: The result (z) is passed through an **activation function** to introduce non-linearity. This helps the neuron decide whether to activate (output a signal) or not.

Think of a neuron as a decision-making unit:

- It processes inputs and decides the strength of its signal based on the inputs' significance and its activation function.

Layers in a Neural Network

A neural network is like a team of neurons working together, organized into layers:

Input Layer:

- This is where the raw data enters the network.
- Example: In an image recognition task, the input layer might take pixel values as input.

Hidden Layers:

- These are the "thinking" layers of the network, where most of the computation happens.
- Each neuron in a hidden layer processes data from the previous layer to learn patterns and features.
- Example: In an image, hidden layers might identify edges, textures, or shapes.

Output Layer:

- This layer produces the final result of the network.
- Example: In a binary classification task, the output layer might produce a single value (e.g., 0 or 1).

Activation Functions

Activation functions help neurons decide whether to "fire" or not. They introduce non-linearity, enabling the network to learn complex patterns. Here are the common ones:

Sigmoid Function:

- Outputs a value between 0 and 1.
- Ideal for binary classification problems (e.g., predicting "yes" or "no").
- Formula:
  $Sigmoid(z) = 1 / (1 + e^{-z})$
- Example: A model predicting whether an email is spam might use sigmoid in the output layer.

ReLU (Rectified Linear Unit):

- Outputs the input directly if it's positive; otherwise, it outputs 0.
- Formula:
  $ReLU(z) = max(0, z)$
- **Advantage:** Helps prevent vanishing gradients and is widely used in hidden layers.
- Example: ReLU allows hidden layers to identify complex features like shapes in an image.

Softmax Function:

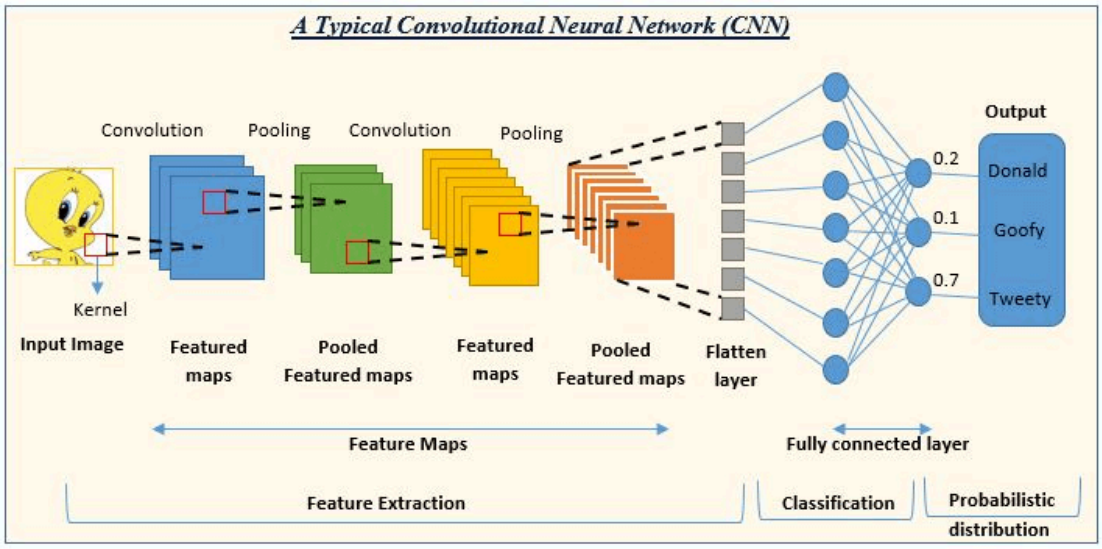
- Converts raw scores into probabilities, with the sum of all probabilities equal to 1.
- Ideal for multi-class classification problems (e.g., categorizing handwritten digits from 0 to 9).
- Formula:
  $Softmax(z_i) = e^{zi} / \sum_j e^{zj}$

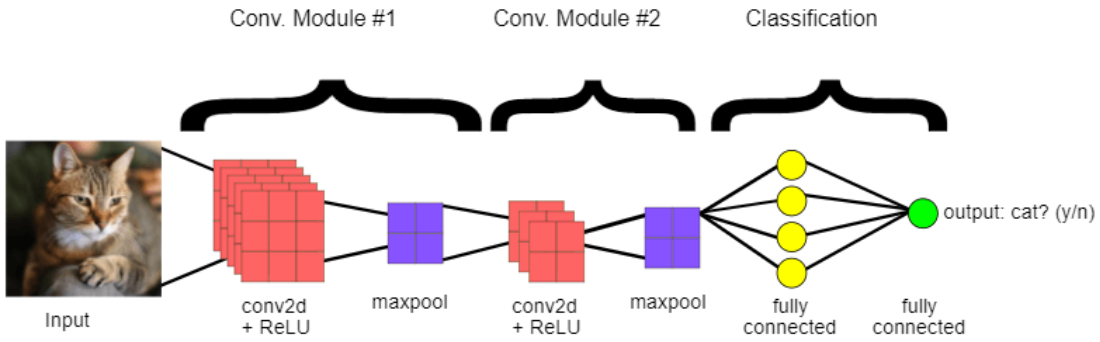
Commonly used algorithms

Learn more from here

1. Convolutional Neural Networks (CNNs):

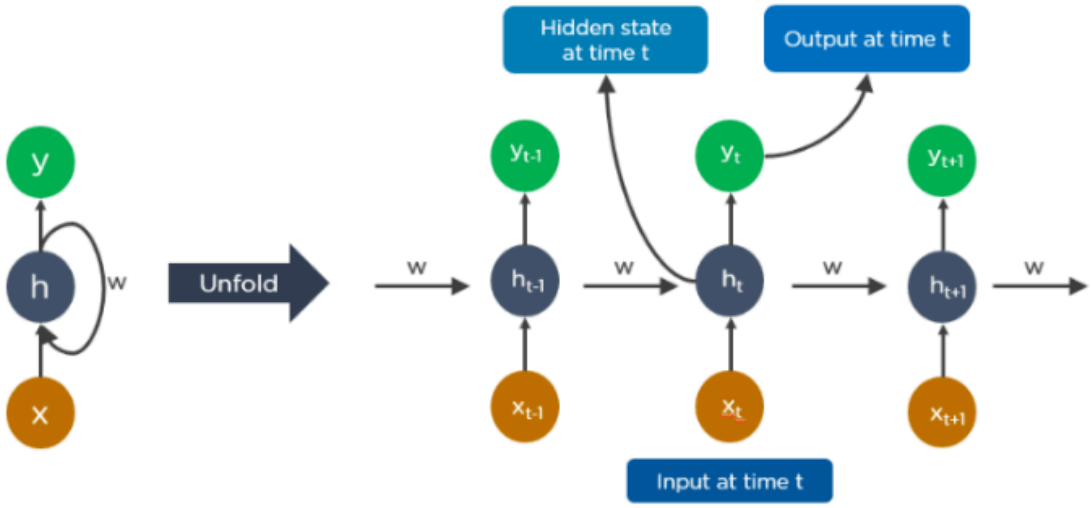
- Designed to process data with a grid-like topology, such as images.
- Uses convolutional layers to detect spatial hierarchies in data, identifying patterns like edges, textures, and shapes.
- Widely used in image classification, object detection, and computer vision tasks.





2. Recurrent Neural Networks (RNNs):

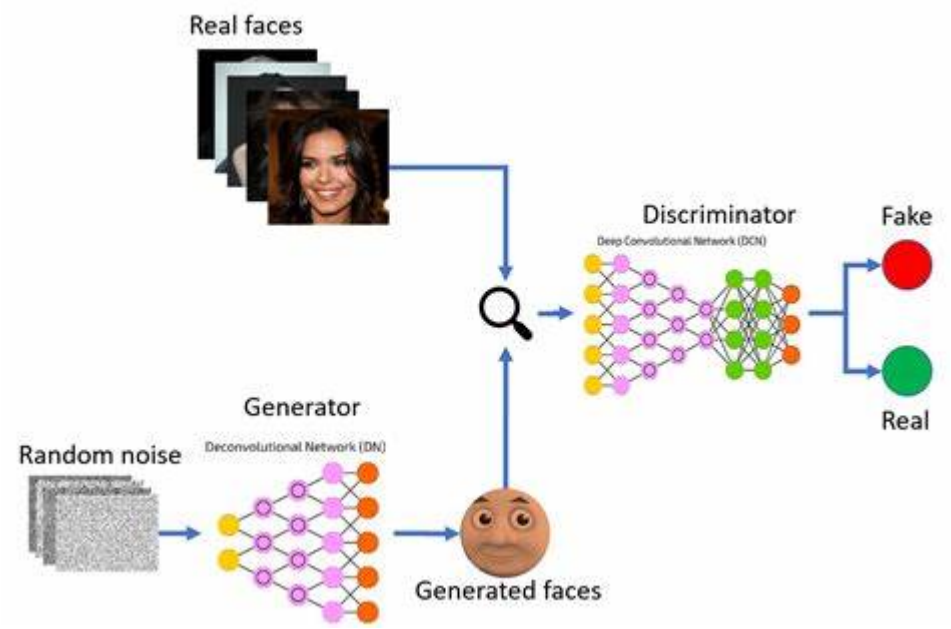
- Specialized for sequence data, such as time series or text.
- Maintains a "memory" of previous inputs using loops within its architecture.
- Commonly applied in natural language processing (NLP) tasks like sentiment analysis, machine translation, and speech recognition.

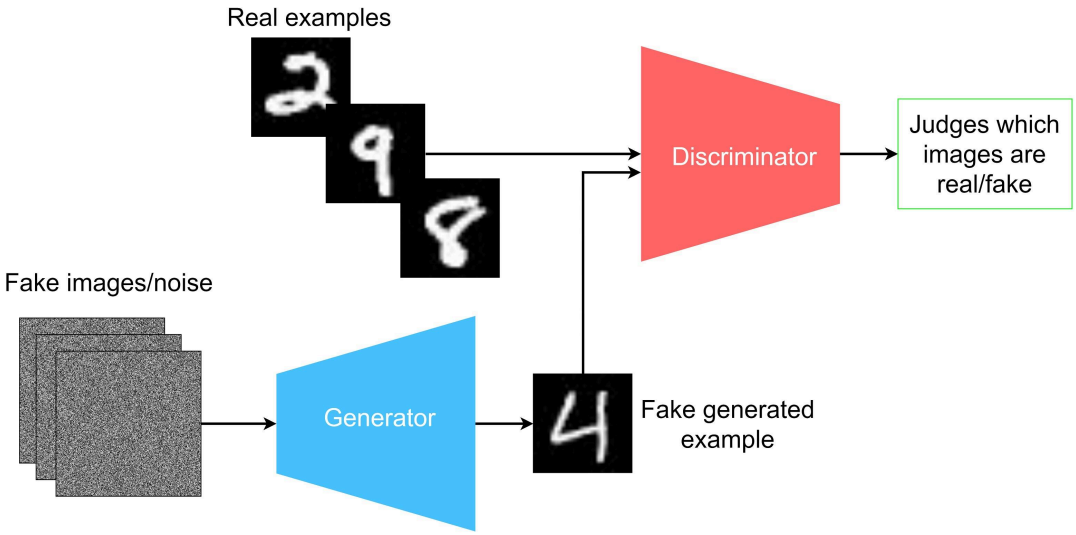


3. Generative Adversarial Networks (GANs):

- Consists of two neural networks: a generator (creates data) and a discriminator (evaluates data).

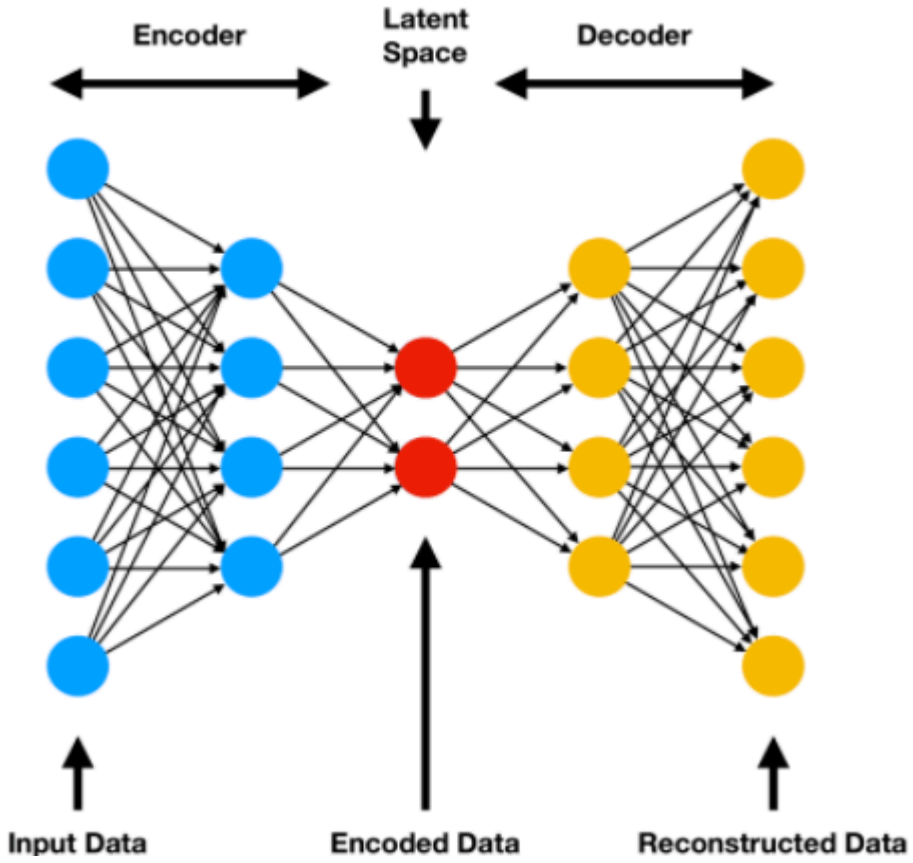
- Both networks compete, resulting in realistic data generation (e.g., synthetic images or videos).
- Popular in tasks like image generation, style transfer, and data augmentation.





4. Autoencoders:

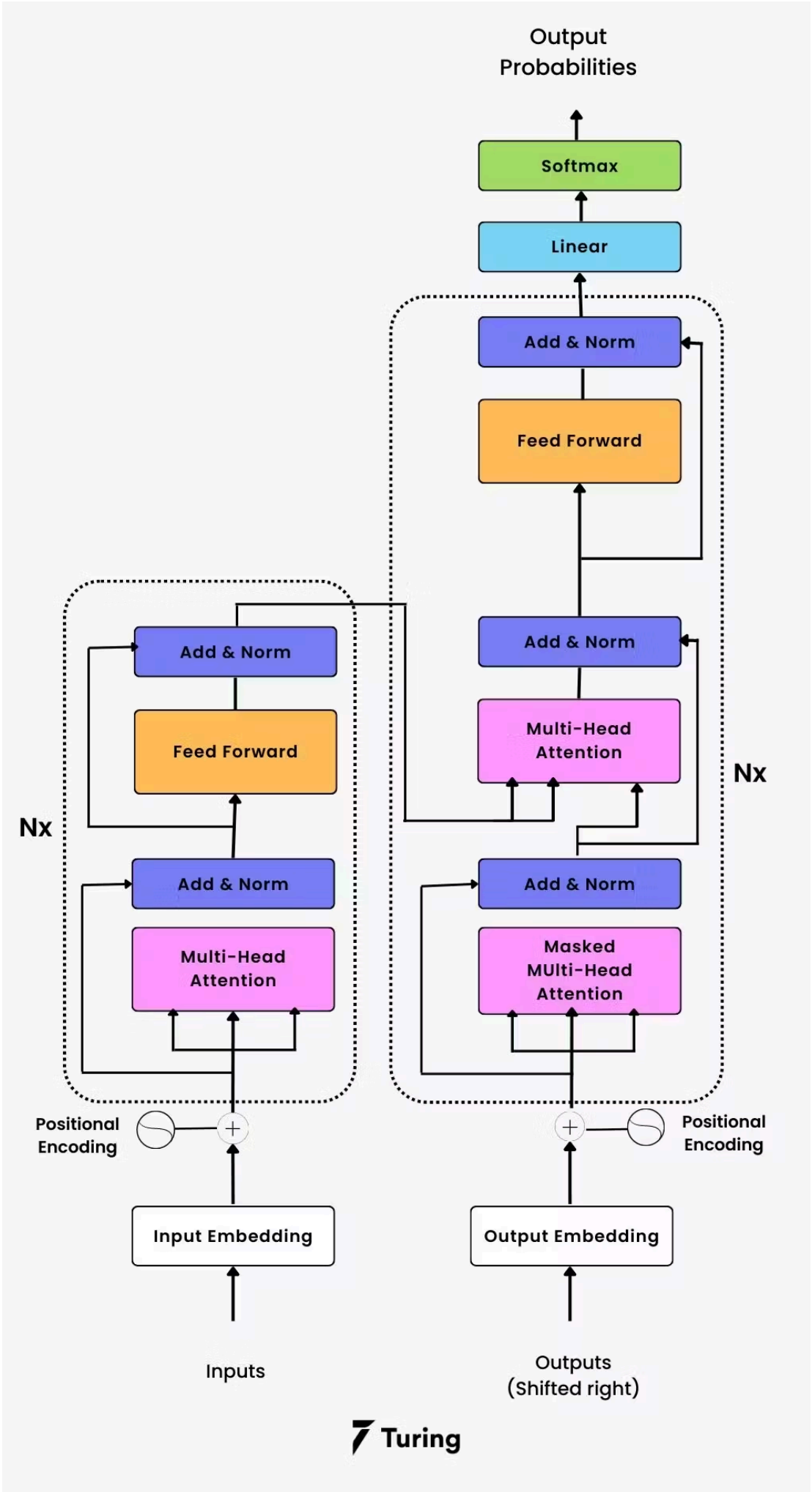
- Unsupervised learning models that encode input data into a compressed representation and then decode it back to the original form.
- Useful for dimensionality reduction, anomaly detection, and data denoising.



5. Transformer Models:

- Transformers are neural networks that learn context and understanding through sequential data analysis. The Transformer models use a modern and evolving mathematical techniques set, generally known as attention or self-attention.
- Focus on attention mechanisms to process sequences without relying on recurrence.
- Revolutionized NLP tasks (e.g., BERT, GPT) by enabling better understanding of context and long-term dependencies.
- Also gaining popularity in computer vision and other domains.

A Transformer, unlike an RNN, does not perform data processing in sequential order, allowing for greater parallelization and faster training.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked MUlti-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (Shifted right)
Turing

This figure illustrates the Transformer deep learning model's overall architecture. There are two main components of the Transformer:

- The encoder stacks — Nx identical encoder layers (in the original published paper, Nx = 6).
- The decoder stacks — Nx identical decoders layers (in the original published paper, Nx =6)

Others algorithms:

- Long Short Term Memory Networks (LSTMs)
- Radial Basis Function Networks (RBFNs)
- Deep Belief Networks (DBNs)
- Neural Style Transfer
- Deep Q-Networks (DQNs)

3. Working of Neural Networks

1. Forward Propagation

- Data flows through the network layer by layer, starting from the input layer to the output layer.
- Each neuron processes the data using weights, biases, and an activation function, passing the output to the next layer.
- The final layer produces predictions or outputs based on the input data.

---

2. Backward Propagation

- After forward propagation, the model's predictions are compared with the actual values to calculate the error (loss).
- The error is propagated backward through the network, adjusting weights and biases in each layer to reduce the loss.
- This process uses the **Gradient Descent algorithm**, which finds the optimal weights by minimizing the loss function.

---

3. Training a Neural Network

Training involves several steps repeated iteratively until the model performs well:

1. **Initialize Weights and Biases:** Start with random values.
2. **Forward Propagation:** Pass input data through the network to make predictions.
3. **Calculate Loss:** Compute the difference between the predicted and actual values using a loss function.
4. **Backward Propagation:** Adjust weights and biases using the Gradient Descent algorithm to minimize the loss.

5. **Repeat:** Iterate the process with multiple data batches (epochs) until the network learns adequately.

---

4. Testing a Neural Network

- Once trained, the model is tested on unseen data to evaluate its performance.
- This step ensures the model generalizes well to new data and is not overfitting.
- Metrics like accuracy, precision, recall, or mean squared error are used to assess performance.

---

4. Intro to TensorFlow and Keras

TensorFlow

TensorFlow is an open-source framework developed by Google for building and deploying machine learning models.

It provides powerful tools for training deep learning models and is widely used in both research and industry.

TensorFlow supports a range of machine learning tasks, from neural networks to more advanced models.

Keras

Keras is a high-level API that sits on top of TensorFlow, making it simpler to build and train neural networks.

It provides an intuitive interface for creating deep learning models with minimal code, allowing you to focus on model design rather than low-level implementation details.

We'll use TensorFlow and Keras to build and train deep learning models. You'll learn how to set up environments, implement basic neural networks, and experiment with different architectures

Tensorflow and Keras

Let's observe how we can quickly see them in action:

In [9]:
```
# pip install tensorflow # might take upto 5 minutes to install
```

In [10]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
```

1. Dummy Data:

In this demonstration, we will use **dummy data** instead of a real dataset. Dummy data is simply random data generated for the purpose of illustration. In this case, we'll create some 2D input data (features) and corresponding output data (labels or targets).

In [11]:
```
import numpy as np

# Dummy input data (5 data points, 2 features each)
X_train = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])

# Dummy output data (5 target values)
y_train = np.array([0, 1, 0, 1, 0])
```

2. Build a Simple Model with Keras:

Using Keras, we can easily define a neural network. We start by creating a **Sequential model**. This model consists of layers, where each layer performs a specific operation on the input data. In our simple model:

- The **input layer** takes in data with 2 features.
- The **hidden layer** processes the data and identifies patterns using 3 neurons and a ReLU activation function.
- The **output layer** generates predictions, which in this case will be a binary value (0 or 1) using a sigmoid activation function.

In [12]:
```
model = models.Sequential([
    layers.Dense(3, input_dim=2, activation='relu'),  # Hidden Layer with 3 neur
    layers.Dense(1, activation='sigmoid')             # Output Layer (binary clas
])
```

3. Compile the Model:

Before training, the model needs to be compiled. Compiling a model means configuring the learning process by selecting an optimizer, loss function, and performance metric:

- **Optimizer (Adam):** An algorithm to update the weights of the network to minimize the loss.
- **Loss Function (Binary Crossentropy):** A function that measures how well the model's predictions match the actual data (suitable for binary classification).
- **Metrics (Accuracy):** Used to evaluate the performance of the model during training.

In [13]:
```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

4. Train the Model:

Training involves feeding the input data to the model and allowing it to adjust the weights using the loss function and optimizer. The model is trained iteratively over multiple epochs until it learns the patterns in the data.

In [14]:
```
model.fit(X_train, y_train, epochs=1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 834ms/step - accuracy: 0.6000 - loss: 0.6811
```

Out[14]: <keras.src.callbacks.history.History at 0x2631c4a0920>

5. Evaluate the Model:

After training, we test the model on the same dummy data to see how well it performs. We use metrics like **accuracy** to determine how well the model is making predictions compared to the actual data.

In [15]:
```
loss, accuracy = model.evaluate(X_train, y_train)
print(f"Model accuracy: {accuracy}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step - accuracy: 0.6000 - loss: 0.6805
Model accuracy: 0.6000000238418579
```

Thus, accuracy is 0.4, means that the model correctly predicted 40% of the time

---

Additional Reading Resources:

- https://www.datacamp.com/tutorial/tensorflow-tutorial#installing-tensorflow
- https://neptune.ai/blog/deep-learning-visualization

Live Exercise

Now it's your turn!

Task 1: description of task

```
- instructions
```

END

THANK YOU!

Live Exercise Solutions

In [16]:
```
solutions
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 solutions

NameError: name 'solutions' is not defined
```

Programming Interveiw Questions

1. topic:
    - question

In [ ]:

Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant