Import Libraries

In [1]:
```
import numpy as np #for numerical compututations
import tensorflow as tf #for neural networks
```

```
WARNING:tensorflow:From C:\Users\Maria james\anaconda3\lib\site-packages
\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entrop
y is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entr
opy instead.
```

Create Dataset

In [2]:
```
# Define the dataset for the AND gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)  # Inputs
y = np.array([[0], [1], [1], [0]], dtype=np.float32)  # Outputs with size 1
```

Multi Layer Perceptron

In [3]:
```
# Define the multi layer perceptron model for XOR
class MLP(tf.keras.Model):
    def __init__(self):
        super(MLP, self).__init__()
        self.hidden = tf.keras.layers.Dense(2, activation='relu')  # Hidde
        self.output_layer = tf.keras.layers.Dense(1, activation='sigmoid')

    def call(self, inputs):
        x = self.hidden(inputs)  # Process through the hidden layer
        return self.output_layer(x)  # Final output
```

In [4]:
```
# Instantiate the model
model = MLP()
```

```
WARNING:tensorflow:From C:\Users\Maria james\anaconda3\lib\site-packages
\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. P
lease use tf.compat.v1.get_default_graph instead.
```

Compile The Model

In [5]:
```
# Compile the model using Binary Cross-Entropy for loss and Adam optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Train the MLP

In [7]:
```
model.fit(X, y, epochs=10, verbose=1)
```

```
Epoch 1/10
1/1 [==============================] - 0s 13ms/step - loss: 0.4155 - accu
racy: 0.7500
Epoch 2/10
1/1 [==============================] - 0s 10ms/step - loss: 0.4154 - accu
racy: 0.7500
Epoch 3/10
1/1 [==============================] - 0s 10ms/step - loss: 0.4153 - accu
racy: 0.7500
Epoch 4/10
1/1 [==============================] - 0s 11ms/step - loss: 0.4152 - accu
racy: 0.7500
Epoch 5/10
1/1 [==============================] - 0s 6ms/step - loss: 0.4151 - accur
acy: 0.7500
Epoch 6/10
1/1 [==============================] - 0s 8ms/step - loss: 0.4150 - accur
acy: 0.7500
Epoch 7/10
1/1 [==============================] - 0s 10ms/step - loss: 0.4148 - accu
racy: 0.7500
Epoch 8/10
1/1 [==============================] - 0s 7ms/step - loss: 0.4148 - accur
acy: 0.7500
Epoch 9/10
1/1 [==============================] - 0s 8ms/step - loss: 0.4146 - accur
acy: 0.7500
Epoch 10/10
1/1 [==============================] - 0s 9ms/step - loss: 0.4145 - accur
acy: 0.7500
```

Out[7]:
```
<keras.src.callbacks.History at 0x28c782fecb0>
```

In [8]:
```
# Extract the weights and bias
print("\nTrained Weights and Biases:")
for layer in model.layers:
    weights, biases = layer.get_weights()
    print(f"Layer: {layer.name}")
    print(f"Weights:\n{weights}")
    print(f"Biases:\n{biases}")
```

```
Trained Weights and Biases:
Layer: dense
Weights:
[[ 0.99008757  1.293399  ]
 [ 0.57743144 -1.2929447 ]]
Biases:
[-0.5777532  -0.00059488]
Layer: dense_1
Weights:
[[-1.9437944]
 [ 2.1150606]]
Biases:
[-0.00149703]
```

Test the model

In [9]:
```
print("\nTesting the XOR Gate MLP:")
for i, input_data in enumerate(X):
    raw_output = model.predict(input_data.reshape(1, -1))  # Model raw outp
    predicted_output = (raw_output > 0.5).astype(np.float32)  # Apply thres
    print(f"Input: {input_data}, Raw Output: {raw_output[0][0]:.4f}, "
          f"Predicted Output: {predicted_output[0][0]}, "
          f"Actual Output: {y[i][0]}")
```

```
Testing the XOR Gate MLP:
1/1 [==============================] - 0s 139ms/step
Input: [0. 0.], Raw Output: 0.4996, Predicted Output: 0.0, Actual Output:
0.0
1/1 [==============================] - 0s 39ms/step
Input: [0. 1.], Raw Output: 0.4996, Predicted Output: 0.0, Actual Output:
1.0
1/1 [==============================] - 0s 44ms/step
Input: [1. 0.], Raw Output: 0.8734, Predicted Output: 1.0, Actual Output:
1.0
1/1 [==============================] - 0s 41ms/step
Input: [1. 1.], Raw Output: 0.1273, Predicted Output: 0.0, Actual Output:
0.0
```

Model Evaluation

In [10]:
```
# Evaluate the model on the training data
train_loss, train_accuracy = model.evaluate(X, y, verbose=1)
print(f"Training Accuracy: {train_accuracy:.4f}")
```

```
Training Accuracy: 0.7500
```

With Regularization

In [18]:
```
class XOR_MLP(tf.keras.Model):
    def __init__(self):
        super(XOR_MLP, self).__init__()
        # Apply L2 regularization to the hidden layer weights
        self.hidden_layer = tf.keras.layers.Dense(2, activation='relu',
                                                  kernel_regularizer=regula
        self.output_layer = tf.keras.layers.Dense(1, activation='sigmoid',
                                                  kernel_regularizer=regula

    def call(self, inputs):
        x = self.hidden_layer(inputs)
        return self.output_layer(x)

model_reg = XOR_MLP()


model_reg.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model_reg.fit(X, y, epochs=10, verbose=1)
```

```
Epoch 1/10
1/1 [==============================] - 1s 983ms/step - loss: 0.7176 - acc
uracy: 0.7500
Epoch 2/10
1/1 [==============================] - 0s 13ms/step - loss: 0.7105 - accu
racy: 0.7500
Epoch 3/10
1/1 [==============================] - 0s 11ms/step - loss: 0.7040 - accu
racy: 0.7500
Epoch 4/10
1/1 [==============================] - 0s 8ms/step - loss: 0.6978 - accur
acy: 0.7500
Epoch 5/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6921 - accur
acy: 0.7500
Epoch 6/10
1/1 [==============================] - 0s 8ms/step - loss: 0.6867 - accur
acy: 0.7500
Epoch 7/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6815 - accur
acy: 0.7500
Epoch 8/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6766 - accur
acy: 0.7500
Epoch 9/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6719 - accur
acy: 0.7500
Epoch 10/10
1/1 [==============================] - 0s 10ms/step - loss: 0.6674 - accu
racy: 0.7500
```

Out[18]:
```
<keras.src.callbacks.History at 0x28c7d606ad0>
```

In [ ]: