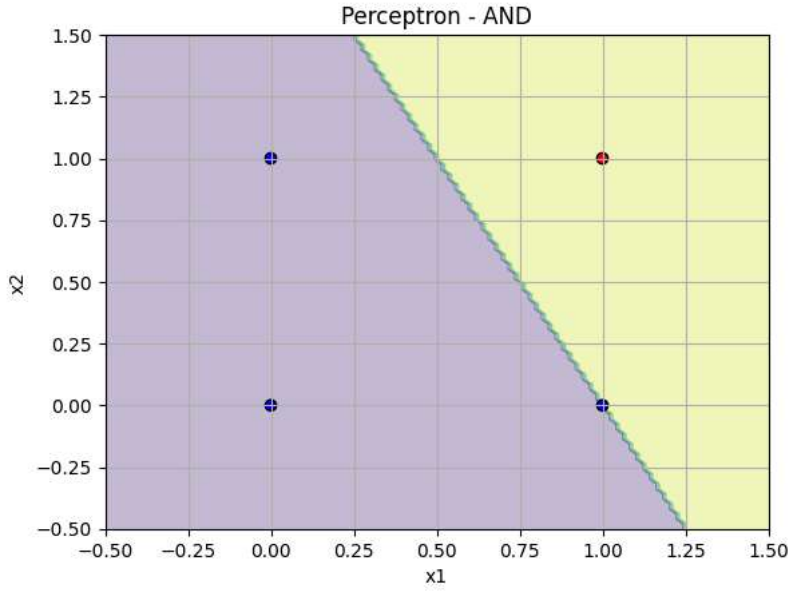
# Lista 10 - Redes Neurais Artificiais

## Perceptron e Backpropagation

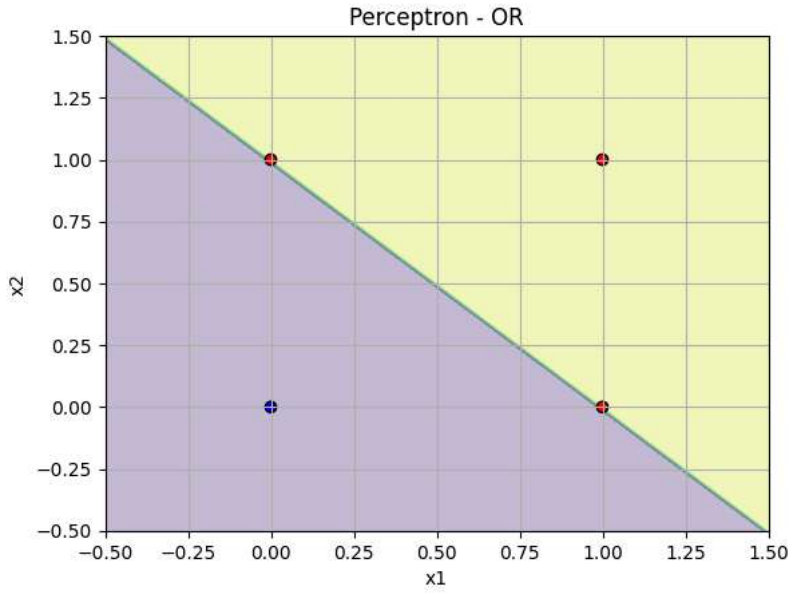
## Questão 1 - Perceptron

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import product

def step(x):
    return 1 if x >= 0 else 0

class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=100):
        self.weights = np.zeros(input_size + 1)
        self.lr = learning_rate
        self.epochs = epochs

    def predict(self, x):
        x = np.insert(x, 0, 1)
        return step(np.dot(self.weights, x))

    def train(self, X, y):
        X = np.insert(X, 0, 1, axis=1)
        for _ in range(self.epochs):
            for xi, target in zip(X, y):
                output = step(np.dot(self.weights, xi))
                update = self.lr * (target - output)
                self.weights += update * xi

def generate_dataset(func, n):
    X = np.array(list(product([0, 1], repeat=n)))
    if func == "AND":
        y = np.array([int(all(x)) for x in X])
    elif func == "OR":
        y = np.array([int(any(x)) for x in X])
    elif func == "XOR":
        y = np.array([sum(x) % 2 for x in X])
    return X, y

def plot_2d_decision_boundary(model, X, y, title):
    if X.shape[1] != 2:
        print("Plot disponível apenas para 2 entradas.")
        return
    x_min, x_max = -0.5, 1.5
    y_min, y_max = -0.5, 1.5
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
    grid = np.c_[xx.ravel(), yy.ravel()]
    Z = np.array([model.predict(pt) for pt in grid]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.bwr)
    plt.title(title)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.grid(True)
    plt.show()
for func in ["AND", "OR", "XOR"]:
    print(f"\n==== Perceptron - Função {func} ====")
    X, y = generate_dataset(func, n=2)
    model = Perceptron(input_size=2)
    model.train(X, y)
    for xi, yi in zip(X, y):
        pred = model.predict(xi)
        print(f"Entrada: {xi} -> Saída: {pred} (Esperado: {yi})")
    plot_2d_decision_boundary(model, X, y, f"Perceptron - {func}")
```

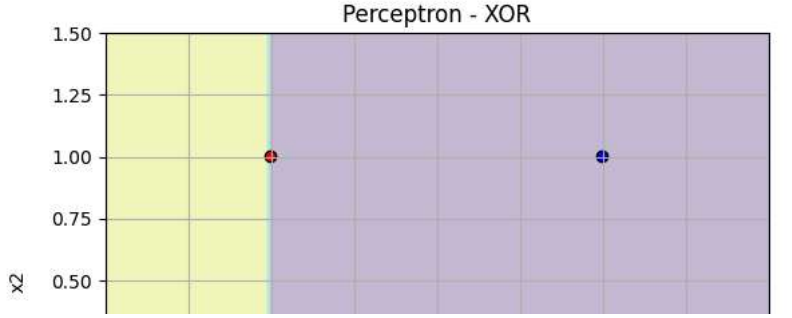
```
==== Perceptron - Função AND ====
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 0 (Esperado: 0)
Entrada: [1 0] -> Saída: 0 (Esperado: 0)
Entrada: [1 1] -> Saída: 1 (Esperado: 1)
```

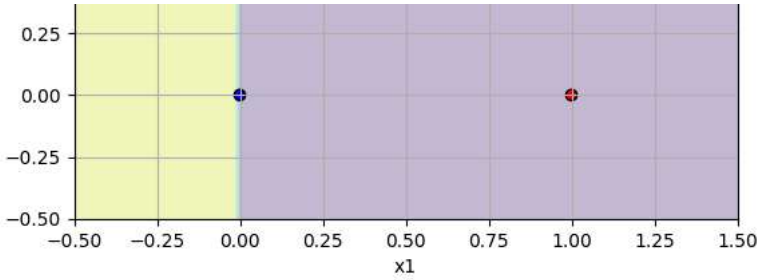



```
==== Perceptron - Função OR ====
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 1 (Esperado: 1)
```




```
==== Perceptron - Função XOR ====
Entrada: [0 0] -> Saída: 1 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 0 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)
```






## Questão 2 - Backpropagation

```
## Questão 2 - Backpropagation


### 1) A Importância da Taxa de Aprendizado

# A taxa de aprendizado (learning rate) influencia diretamente na velocidade de convergência da rede.
# - **Taxas muito baixas**: convergência lenta.
# - **Taxas muito altas**: podem causar oscilação e não convergência.

#python:
for lr in [0.01, 0.1, 0.5]:
    print(f"\n-- Taxa de aprendizado = {lr} --")
    X, y = generate_dataset("XOR", n=2)
    y = y.reshape(-1, 1)
    mlp = MLP(n_inputs=2, hidden_size=4, lr=lr, activation="sigmoid", bias=True)
    mlp.train(X, y, epochs=10000)
    for xi, yi in zip(X, y):
        pred = int(mlp.predict(xi))
        print(f"Entrada: {xi} -> Saída: {pred} (Esperado: {yi[0]})")

### 2) A Importância do Bias

# O bias permite deslocar a função de ativação e torna o modelo mais flexível. Sem o bias, a rede pode ter dificuldades para aprender padrões

#python:
for bias_flag in [True, False]:
    print(f"\n-- Bias = {bias_flag} --")
    X, y = generate_dataset("XOR", n=2)
    y = y.reshape(-1, 1)
    mlp = MLP(n_inputs=2, hidden_size=4, lr=0.1, activation="sigmoid", bias=bias_flag)
    mlp.train(X, y, epochs=10000)
    for xi, yi in zip(X, y):
        pred = int(mlp.predict(xi))
        print(f"Entrada: {xi} -> Saída: {pred} (Esperado: {yi[0]})")

### 3) A Importância da Função de Ativação

#python:
for act in ["sigmoid", "tang", "relu"]:
    print(f"\n-- Função de ativação: {act} --")
    X, y = generate_dataset("XOR", n=2)
    y = y.reshape(-1, 1)
    mlp = MLP(n_inputs=2, hidden_size=4, lr=0.1, activation=act, bias=True)
    mlp.train(X, y, epochs=10000)
    for xi, yi in zip(X, y):
        pred = int(mlp.predict(xi))
        print(f"Entrada: {xi} -> Saída: {pred} (Esperado: {yi[0]})")
```

```
-- Taxa de aprendizado = 0.01 --
<ipython-input-6-089704a4116d>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error ir
  pred = int(mlp.predict(xi))
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 1 (Esperado: 0)

-- Taxa de aprendizado = 0.1 --
```

```
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)

-- Taxa de aprendizado = 0.5 --
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)

-- Bias = True --
<ipython-input-6-089704a4116d>:34: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error ir
  pred = int(mlp.predict(xi))
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)

-- Bias = False --
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)

-- Função de ativação: sigmoid --
<ipython-input-6-089704a4116d>:52: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error ir
  pred = int(mlp.predict(xi))
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)

-- Função de ativação: tanh --
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 1 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)

-- Função de ativação: relu --
Entrada: [0 0] -> Saída: 0 (Esperado: 0)
Entrada: [0 1] -> Saída: 0 (Esperado: 1)
Entrada: [1 0] -> Saída: 1 (Esperado: 1)
Entrada: [1 1] -> Saída: 0 (Esperado: 0)
```