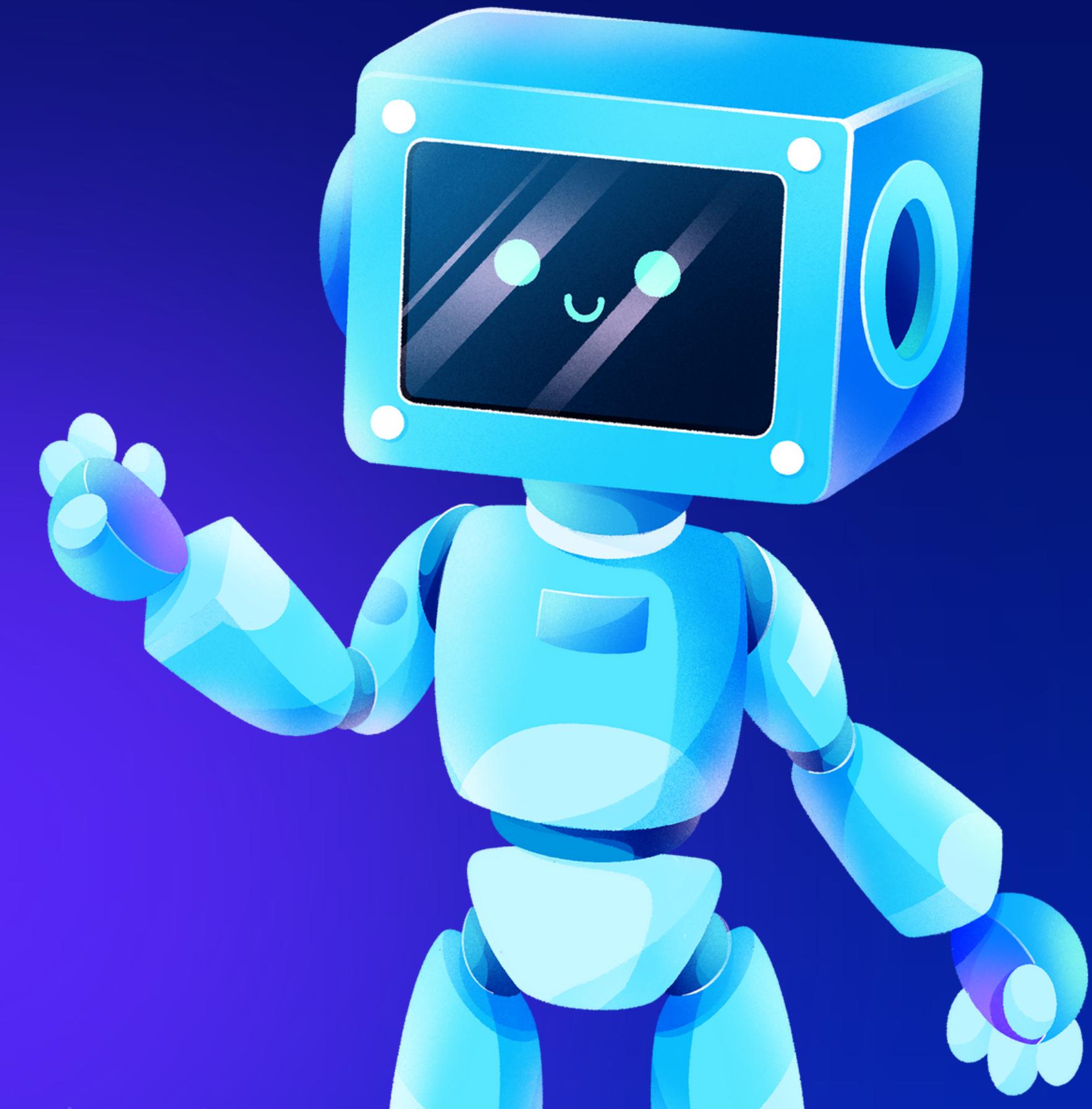


# REDES NEURAIS

Erick e Eryka





# Projeto 3

Esse projeto consiste na implementação de uma RNA do tipo MLP, cuja o objetivo é classificar pessoas como tendo crédito bom ou ruim. Dois data sets são fornecidos. O **data set original**, no formato fornecido pelo Prof. Hofmann, que contém atributos categóricos/simbólicos e está no arquivo "**german.data**". Para algoritmos que necessitam de atributos numéricos, pode-se usar o arquivo "**german.data-numeric**". Este arquivo foi editado para torná-lo adequado para algoritmos que não conseguem lidar com variáveis categóricas. Vários atributos ordenados categóricos (como o atributo 17) foram codificados como inteiros.

Previa do dataset:

1	1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1	1
2	2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1	2
3	4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0	1
4	1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	0	1
5	1	24	3	49	1	3	3	4	4	53	3	2	2	1	1	1	0	1	0	0	0	0	0	0	1
6	4	36	2	91	5	3	3	4	4	35	3	1	2	2	1	0	0	1	0	0	0	0	0	1	1
7	4	24	2	28	3	5	3	4	2	53	3	1	1	1	1	0	0	1	0	0	1	0	0	1	1
8	2	36	2	69	1	3	3	2	3	35	3	1	1	2	1	0	1	1	0	0	1	0	0	0	1
9	4	12	2	31	4	4	1	4	1	61	3	1	1	1	1	0	0	1	0	0	1	0	1	0	1
10	2	30	4	52	1	1	4	2	3	28	3	2	1	1	1	1	0	1	0	0	1	0	0	0	2

## Código - Pré Processamento e Criação de funções

```
# Carregue seus dados do 'test.csv'  
data = 'test.csv'  
  
df = pd.read_csv(data)  
df['1.7'] = df['1.7'].replace({1: 0, 2: 1})  
  
# Defina a função de ativação sigmoid  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
# Defina a derivada da função sigmoid  
# 2 usages  
  
def sigmoid_derivative(x):  
    return x * (1 - x)
```

Ao converter rótulos de 1 e 2 para 0 e 1, o código está alinhando os rótulos com a convenção geralmente usada em problemas de classificação binária e tornando-os compatíveis com a função sigmoid e a função de custo associada.

Parte I: Defina e implemente os componentes necessários para o treinamento da rede neural:

```
# Defina a função de treinamento para a MLP com duas camadas ocultas
1 usage
def initialize_weights(input_size, hidden_units1, hidden_units2, output_size):
    limit_xavier_hidden1 = np.sqrt(2 / (input_size + hidden_units1))
    weights_hidden1 = np.random.uniform(-limit_xavier_hidden1, limit_xavier_hidden1, size=(input_size, hidden_units1))

    limit_xavier_hidden2 = np.sqrt(2 / (hidden_units1 + hidden_units2))
    weights_hidden2 = np.random.uniform(-limit_xavier_hidden2, limit_xavier_hidden2, size=(hidden_units1, hidden_units2))

    limit_xavier_output = np.sqrt(2 / (hidden_units2 + output_size))
    weights_output = np.random.uniform(-limit_xavier_output, limit_xavier_output, size=(hidden_units2, output_size))

    return weights_hidden1, weights_hidden2, weights_output
```

A função Calcula os limites de inicialização para cada camada oculta e de saída usando a fórmula de Xavier, e logo em seguida inicializa os pesos para cada camada com valores aleatórios entre os limites calculados

## Parte II: Implemente a propagação para frente e calcule o custo;

```
# Parte II: Implemente a propagação para frente e calcule o custo
2 usages
def forward_propagation(X, weights_hidden1, weights_hidden2, weights_output):
    hidden1 = sigmoid(np.dot(X, weights_hidden1))
    hidden2 = sigmoid(np.dot(hidden1, weights_hidden2))
    output = sigmoid(np.dot(hidden2, weights_output))
    return hidden1, hidden2, output

2 usages
def calculate_cost(y_true, y_pred):
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    m = len(y_true)
    cost = -(1/m) * np.sum(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return cost
```

### forward\_propagation:

- Calcula as saídas da rede neural aplicando a função sigmoid nas camadas ocultas e de saída.
- Recebe como entrada os dados X e os pesos das camadas ocultas e de saída.

### calculate\_cost:

- Calcula o custo da rede usando entropia cruzada binária.
- Recebe como entrada os rótulos verdadeiros e as previsões da rede neural.

## Parte III: Calcule o gradiente da função de custo

```
# Parte III: Calcule o gradiente da função de custo
1 usage
def calculate_gradients(X, y, hidden1, hidden2, output, weights_hidden1, weights_hidden2, weights_output):
    delta_output = output - y
    delta_hidden2 = delta_output.dot(weights_output.T) * sigmoid_derivative(hidden2)
    delta_hidden1 = delta_hidden2.dot(weights_hidden2.T) * sigmoid_derivative(hidden1)

    grad_output = hidden2.T.dot(delta_output)
    grad_hidden2 = hidden1.T.dot(delta_hidden2)
    grad_hidden1 = X.T.dot(delta_hidden1)

    return grad_hidden1, grad_hidden2, grad_output
```

A função `calculate_gradients` calcula os gradientes durante a retropropagação na rede neural, utilizando a técnica de descida de gradiente. Esses gradientes são essenciais para atualizar os pesos da rede e melhorar seu desempenho durante o treinamento.

Parte IV: Implemente a retropropagação. (Use a regra de atualização para o gradiente descendente);

```
# Parte IV: Implemente a retropropagação. (Use a regra de atualização para o gradiente descendente)
1 usage
def backpropagation(weights_hidden1, weights_hidden2, weights_output, grad_hidden1, grad_hidden2, grad_output, learning_rate):
    weights_hidden1 -= learning_rate * grad_hidden1
    weights_hidden2 -= learning_rate * grad_hidden2
    weights_output -= learning_rate * grad_output

    return weights_hidden1, weights_hidden2, weights_output
```

Essa função `backpropagation` implementa a atualização dos pesos durante a fase de retropropagação, utilizando a regra do gradiente descendente. Essa etapa é fundamental para ajustar os pesos da rede e minimizar o custo durante o treinamento. O `learning rate` controla a taxa de atualização, influenciando a convergência do modelo.

## Resultados

### Custo a cada epoch

```
Epoch 1/100 - Cost: 0.6911993857481155
Epoch 11/100 - Cost: 0.6028347623121959
Epoch 21/100 - Cost: 0.5159544419228047
Epoch 31/100 - Cost: 0.4867661982630222
Epoch 41/100 - Cost: 0.4805172290808845
Epoch 51/100 - Cost: 0.4772920580236428
Epoch 61/100 - Cost: 0.47479539683790467
Epoch 71/100 - Cost: 0.4723583452181209
Epoch 81/100 - Cost: 0.4695638445351714
Epoch 91/100 - Cost: 0.46601302974034287
```

O objetivo durante o treinamento é reduzir gradualmente o custo. Se os valores estiverem diminuindo, isso sugere que a rede está aprendendo padrões nos dados. No entanto, se os custos começarem a se estabilizar, pode ser um sinal de que a rede atingiu uma espécie de limite em sua capacidade de aprendizado.

## Resultados

### Acurácia e média do custo

```
Acurácia da Rede Neural: 0.74
```

```
Média da Função Custo no Conjunto de Teste: 8.980113846573985
```

A acurácia de 0.74 indica que a rede neural alcançou uma precisão de aproximadamente 74% no conjunto de teste. Isso significa que, das amostras no conjunto de teste, cerca de 74% foram classificadas corretamente.

Quanto à média da função custo no conjunto de teste (8.980113846573985), representa o custo médio associado às previsões da rede neural. Em problemas de classificação, geralmente se deseja minimizar esse custo, pois ele reflete o quão bem as previsões da rede correspondem aos rótulos reais.