Lista 6 - Redes Neurais

Aluno: Gabriel Azevedo Fernandes

Questão 1 – Letra C (Apenas I e III)

Questão 2 – Letra A (Somente I)

Questão 3 – Letra C (o perceptron realiza a função OR)

Questão 4

1) Pré-processamento dos dados:

- Atributos nominais são convertidos em numéricos usando a técnica de One-Hot Encoding.

- Depois, foi realizado uma análise para identificar e tratar outliers, garantindo a qualidade dos dados.

- Os dados foram normalizados para que tivessem uma média igual a zero e um desvio padrão unitário.

- Técnicas de balanceamento de dados foram aplicadas para corrigir o desequilíbrio entre as classes.

- Por fim, características redundantes são eliminadas.

2) Configuração da rede neural:

Diversas configurações de redes neurais foram testadas, diversificando o número de camadas e neurônios, para encontrar a configuração ótima.

A configuração ideal foi uma camada oculta com 10 neurônios, equilibrando complexidade e desempenho, evitando desajustes.

3) Avaliação da taxa de aprendizado:

Diferentes taxas de aprendizado foram testadas, assim, pode se ter a noção dos seus efeitos na convergência e na acurácia do modelo.

A taxa de aprendizado ideal foi de 0.01, o que resulta em uma convergência rápida e alta acurácia, independentemente do número de épocas.

4) Impacto do momento e taxa de decaimento:

Os impactos do momento e da taxa de decaimento do peso foram analisados, necessitando ajustes para melhorar a estabilidade do treinamento.

Mas, a taxa de aprendizado e a configuração da rede foram os fatores que mais influenciaram o desempenho.

5) Ajuste de hiperparâmetros:

Técnicas como Grid Search e Cross-Validation Parameter Selection foram usadas para explorar sistematicamente combinações de hiperparâmetros e assim identificar a melhor configuração.

O Grid Search confirmou que uma taxa de aprendizado de 0.01 e uma camada oculta com 10 neurônios eram ideais, maximizando a acurácia e mantendo o tempo de treinamento razoável.

Questão 5 - Para aprender as funções booleanas AND e OR foi utilizado o perceptron depois somente ajustando seus pesos através de um método de aprendizado supervisionado.

```
import numpy as np

def perceptron(X, y, n_inputs, epochs=10, learning_rate=0.1):

weights = np.zeros(n_inputs + 1)

for _ in range(epochs):

for i in range(len(X)):

activation = np.dot(X[i], weights[1:]) + weights[0]

prediction = 1 if activation >= 0 else 0

weights[1:] += learning_rate * (y[i] - prediction) * X[i]

weights[0] += learning_rate * (y[i] - prediction)

return weights
```

Questão 6

O Backpropagation é um método importante para o treinamento de redes neurais. Ele vai nos permitir o ajuste dos pesos em redes de múltiplas camadas. Nesse caso, o algoritmo vai operar em duas fases:

uma propagação para frente, que processa os sinais de entrada até a saída, e uma propagação para trás, que ajusta os pesos da rede baseando-se no erro de saída. Esse processo iterativo é importante pro aprendizado de complexidades em dados que não são linearmente separáveis, tornando o Backpropagation um pilar no desenvolvimento de soluções de aprendizado profundo. Ex de código com rede neural com uma camada oculta utilizando o algoritmo de backpropagation para aprender as funções booleanas AND, OR e XOR

```
import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):

return x * (1 - x)

def train_backpropagation(X, y, n_inputs, hidden_neurons, epochs=10000,
```

```
learning_rate=0.1):
weights_hidden = np.random.uniform(size=(n_inputs, hidden_neurons))
bias_hidden = np.random.uniform(size=(hidden_neurons,))
weights_output = np.random.uniform(size=(hidden_neurons,))
bias_output = np.random.uniform()
for _ in range(epochs):
hidden_layer_input = np.dot(X, weights_hidden) + bias_hidden
hidden_layer_output = sigmoid(hidden_layer_input)
output_layer_input = np.dot(hidden_layer_output, weights_output) + bias_output
predicted_output = sigmoid(output_layer_input)
error = y - predicted_output
d_predicted_output = error * sigmoid_derivative(predicted_output)
error_hidden_layer = d_predicted_output * weights_output
d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)
weights_output += hidden_layer_output.T.dot(d_predicted_output) *
learning_rate
bias_output += np.sum(d_predicted_output) * learning_rate
weights_hidden += X.T.dot(d_hidden_layer) * learning_rate
bias_hidden += np.sum(d_hidden_layer, axis=0) * learning_rate
return weights_hidden, bias_hidden, weights_output, bias_output
```