Engenharia de Redes Neurais Artificiais

Aluno: Matheus Paul Lopuch

Exercícios das Aulas

Aula 01 (18.09)

Representação de Redes Neurais Artificiais

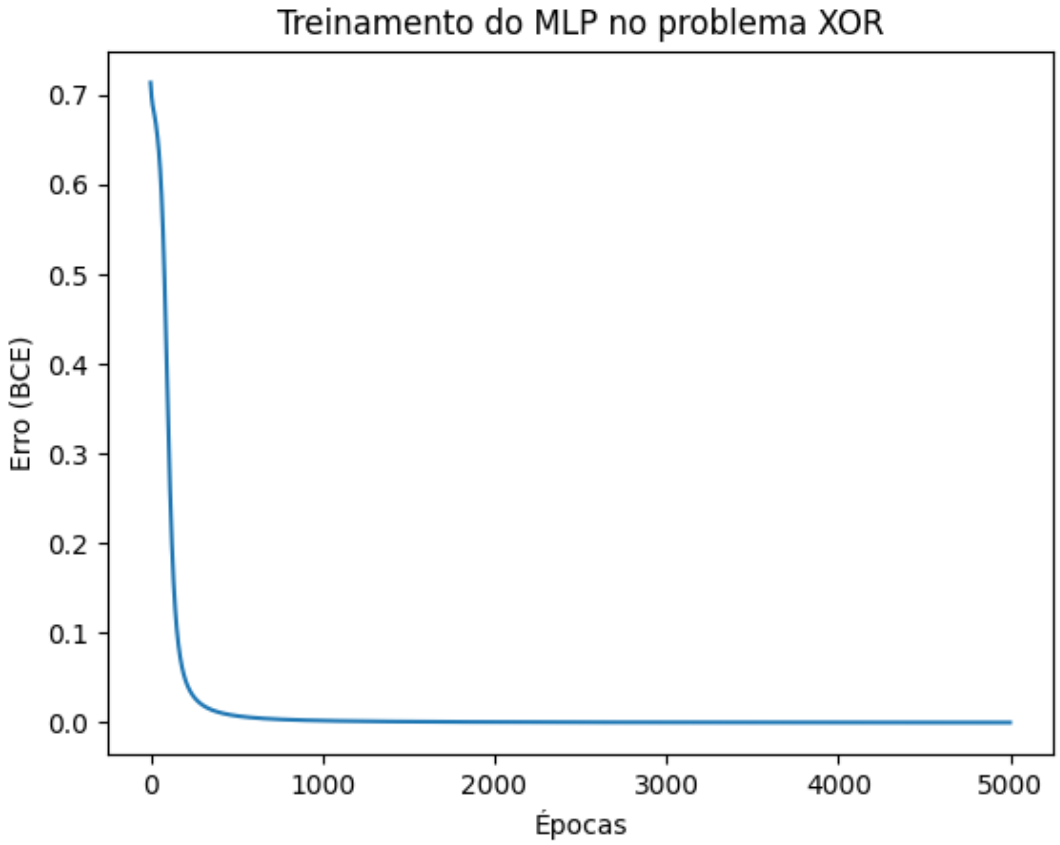
Exercícios:

1. **Execute a célula de treino do *perceptron* algumas vezes e verifique os resultados. Depois, realize o mesmo com o treino do perceptron multicamadas. Houveram diferenças? Discuta quais são as diferenças e justifique.**
2. **Modifique a chamada do Perceptron Multicamadas para utilizar apenas um *perceptron* na camada oculta. Depois, utilize apenas dois *perceptrons*. Foi possível resolver o problema não-linear? Discuta a resposta e justifique.**
3. **Reproduza o código do Multilayer Perceptron utilizando Pytorch.**

Respostas:

1. **Sim, existem diferenças importantes que foram observadas. No Perceptron simples ele só consegue resolver problemas linearmente separáveis como AND, OR por exemplo. Ele não consegue resolver problemas não lineares, como XOR. Outro ponto também é que ele depende fortemente dos pesos iniciais, onde pode convergir ou não. Agora no Perceptron multicamadas ele consegue modelar fronteiras de decisão não lineares porque ele combina múltiplos neurônios ocultos e funções de ativação não lineares. Ele é capaz de resolver problemas não linearmente separáveis e os resultados tendem a ser mais estáveis entre execuções ainda que variem com inicializações aleatórias. Em resumo, o Perceptron simples ele tem como limitação gerar apenas hiperplanos lineares e o Perceptron multicamadas ao combinar vários perceptrons ocultos ele gera regiões de decisão complexas e não lineares.**

2. Com apenas 1 neurônio oculto não é possível resolver problemas não lineares, o modelo continua gerando apenas uma fronteira linear.
   Já com 2 neurônios ocultos, é possível aproximar alguns problemas não lineares mais ainda existem limitações, por exemplo para o problema clássico do XOR, 2 neurônios já são suficientes para gerar a fronteira de decisão correta, então em problemas mais complexos deve ter mais neurônios.
   Em resumo, a camada oculta aumenta a capacidade de representação da rede. Com apenas 1 neurônio, não há poder expressivo. Com 2 ou mais, a rede consegue compor funções não-lineares e resolver o XOR.

3. 


```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt

# Dataset XOR
X = torch.tensor([[0,0],[0,1],[1,0],[1,1]],
dtype=torch.float32)
y = torch.tensor([[0],[1],[1],[0]], dtype=torch.float32)
```

```
# Definição do MLP
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
         self.hidden = nn.Linear(2, 4)   # mais neurônios
ocultos (4 em vez de 2)
        self.output = nn.Linear(4, 1)   # saída
          self.activation_hidden = nn.Tanh()    # ativação
melhor na camada oculta
       self.activation_output = nn.Sigmoid() # saída binária
entre 0 e 1

    def forward(self, x):
        x = self.activation_hidden(self.hidden(x))
        x = self.activation_output(self.output(x))
        return x

# Instancia modelo
model = MLP()
criterion = nn.BCELoss()  # perda binária mais adequada
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Treinamento
epochs = 5000
losses = []
for epoch in range(epochs):
    optimizer.zero_grad()
    outputs = model(X)
    loss = criterion(outputs, y)
    loss.backward()
    optimizer.step()
    losses.append(loss.item())

# Plot do erro
plt.plot(losses)
plt.xlabel("Épocas")
plt.ylabel("Erro (BCE)")
plt.title("Treinamento do MLP no problema XOR")
plt.show()

# Predições
with torch.no_grad():
    preds = model(X)
    print("Saídas previstas (valores contínuos):\n", preds)
    print("Saídas arredondadas (0 ou 1):\n", preds.round())
```