




Lista de exercícios 01

Aprendizado profundo



João Honorato
Maria Raquel
Samila Garrido

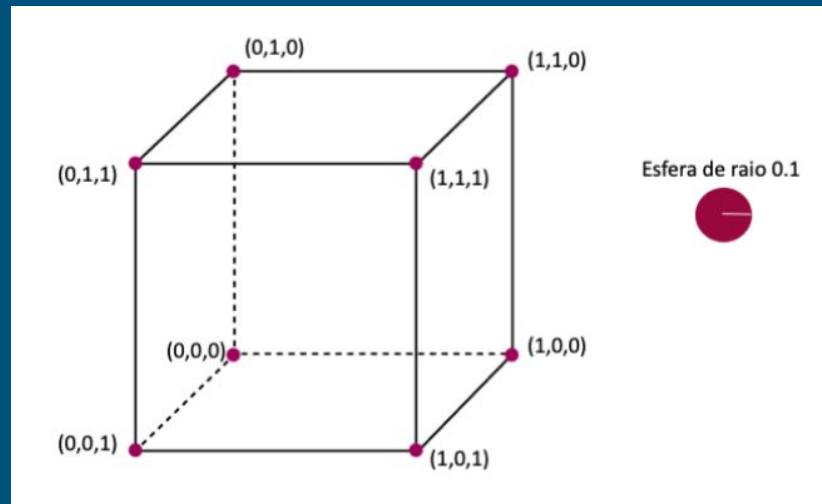
QUESTÃO

0 01

Questão 01

Classificação de pontos em oito padrões, representado pelos vértices de um cubo de lado 1

Dados com ruído de $\pm 0,1$ por componente de coordenada



Questão 01

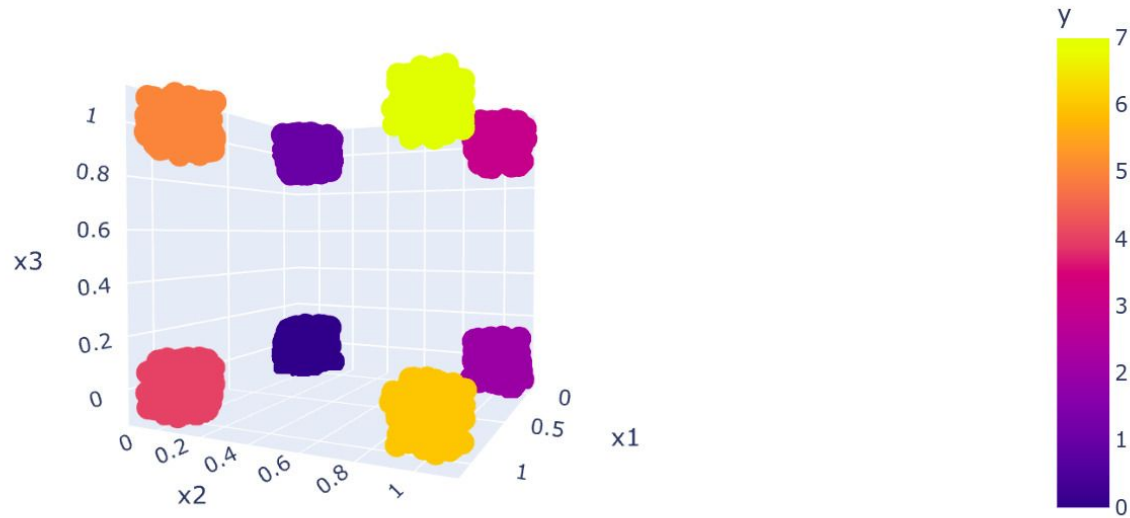
Arquitetura

Um perceptron de Rosenblatt que recebe pontos de um espaço 3D e retorna oito possíveis classificações

```
class RoseblattPerceptron(nn.Module):  
    def __init__(self):  
        super(RoseblattPerceptron, self).__init__()  
        self.perceptron = nn.Linear(3, 8, dtype=torch.float64)  
  
    def forward(self, x):  
        x = self.perceptron(x)  
        output = F.softmax(x, dim=1, dtype=torch.float64)  
        return output
```

Arquitetura da rede com PyTorch
(Acurácia 1.00)

Questão 01



Plot dos dados classificados

QUESTÃO

0 02

Questão 02

Usando PyTorch, criamos duas funções para a rede perceptron de múltiplas camadas: ***train e evaluate***

train é responsável pelo treinamento e pela validação do modelo, prevendo os rótulos dos dados e calculando as perdas para os dois conjuntos de dados para cada época

evaluate aplica os dados de teste e avalia a acurácia do modelo treinado

Usamos as duas funções para todas as questões, com ajustes na arquitetura de cada modelo

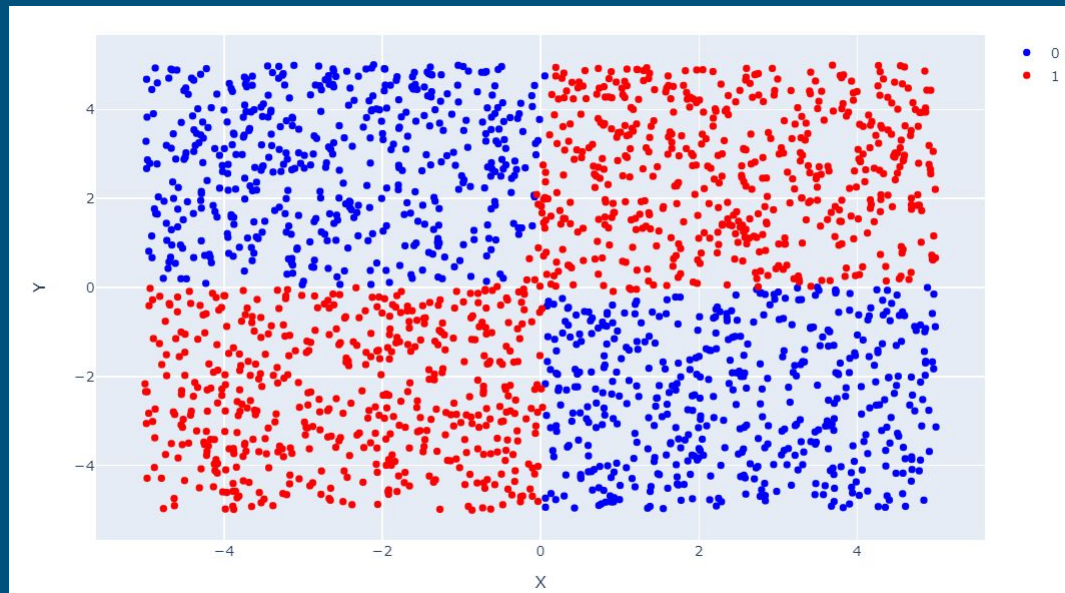
Questão 02

a) XOR

```
class PerceptronXOR(nn.Module):
    def __init__(self):
        super(PerceptronXOR, self).__init__()
        self.fc1 = nn.Linear(2, 4)  ## Camada de entrada
        self.fc2 = nn.Linear(4, 1)  ## Camada de saída

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x
```

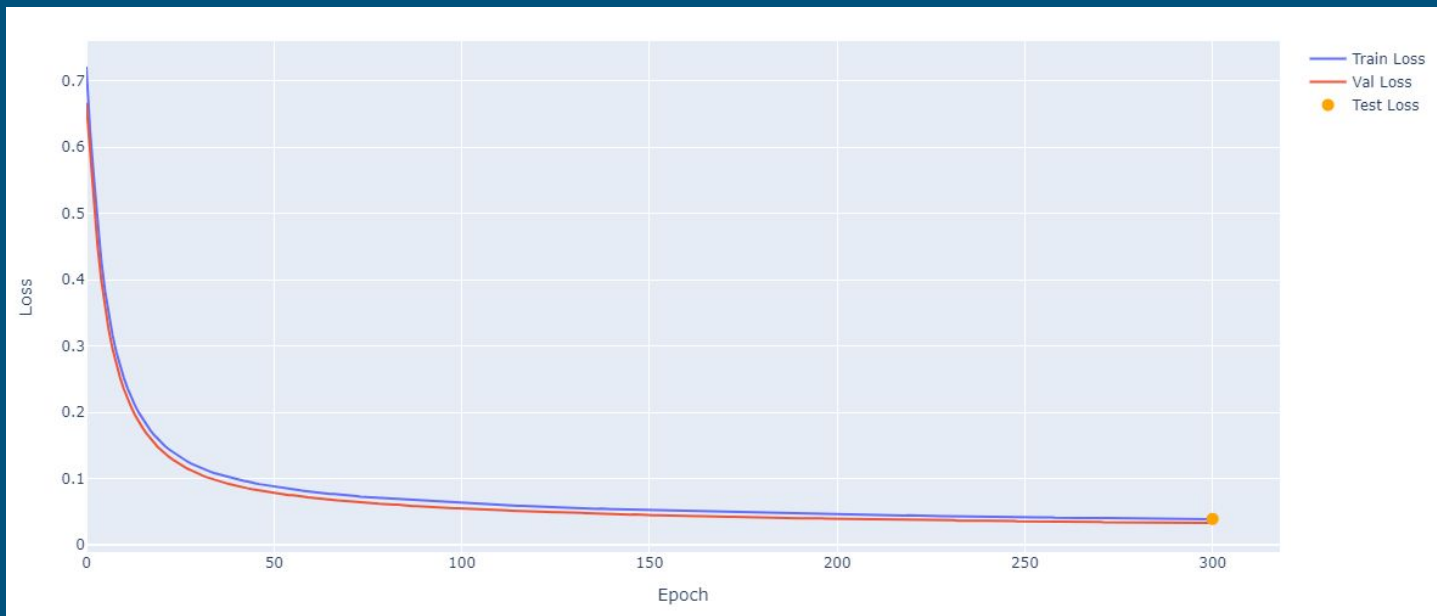
Arquitetura da rede



Plot dos dados classificados
(acurácia de 0.99)

Questão 02

a) XOR



Erro por época para os dados de
treino e validação

Questão 02

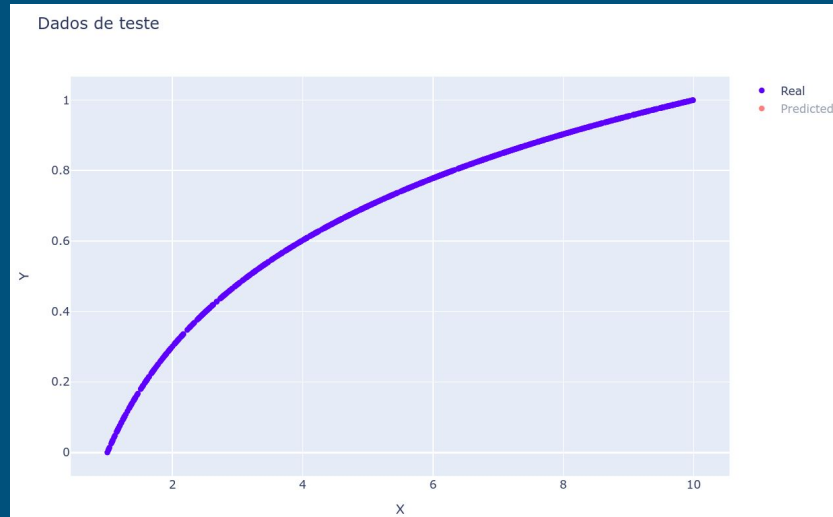
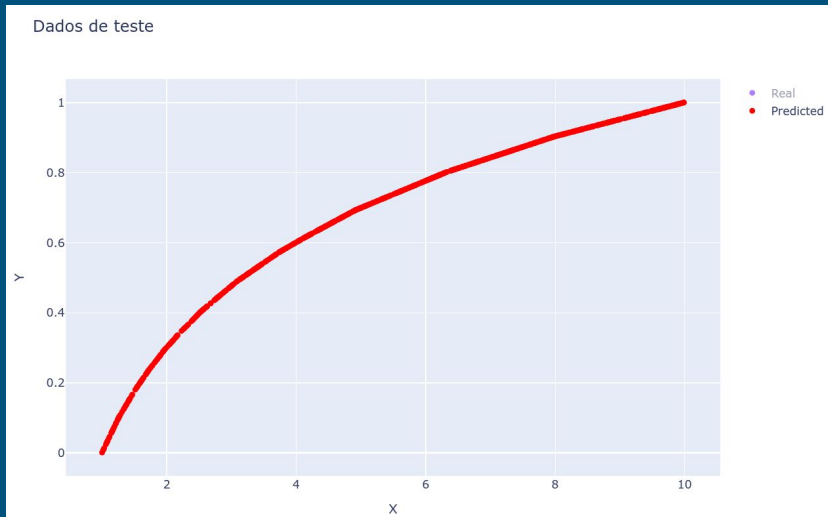
b) $\log_{10}(x)$

```
class PerceptronLog10(nn.Module):  
    def __init__(self):  
        super(PerceptronLog10, self).__init__()  
        self.fc1 = nn.Linear(1, 100)  
        self.fc2 = nn.Linear(100, 1)  
  
    def forward(self, x):  
        x = torch.relu(self.fc1(x))  
        x = self.fc2(x)  
        return x
```

Arquitetura da rede

Questão 02

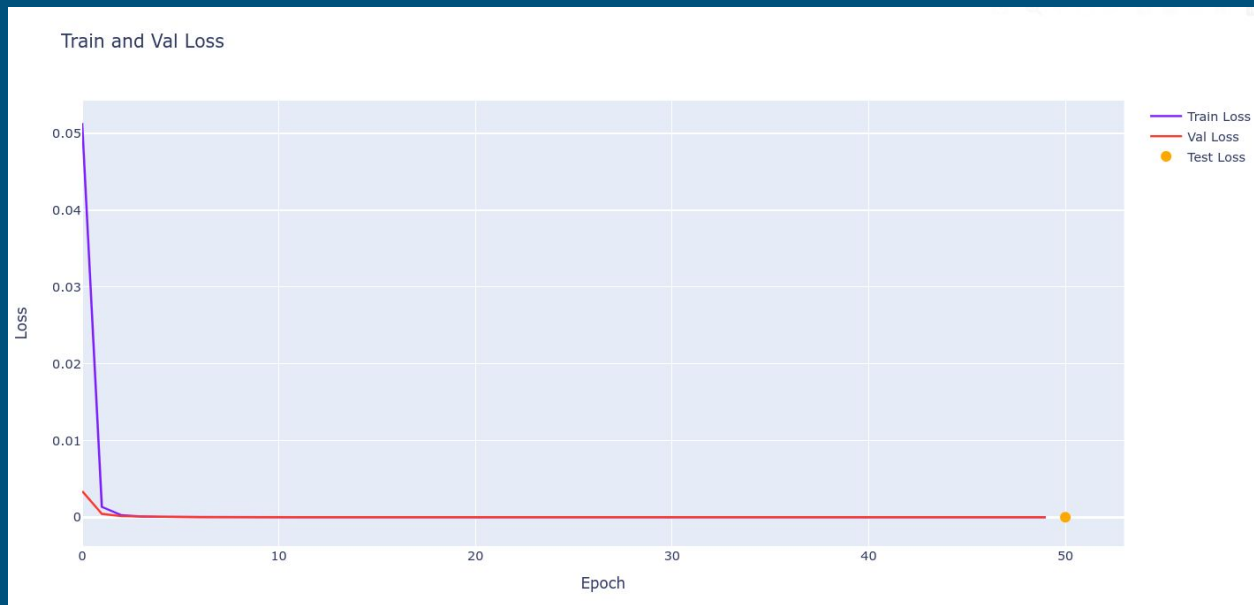
b) $\log_{10}(x)$



Plot da curva esperada e da curva encontrada pelo modelo

Questão 02

b) $\log_{10}(x)$



Erro por época para os dados de
treino e validação

Questão 02

c) $10x^5 + 5x^4 + 2x^3 - 0.5x^2 + 3x + 2$

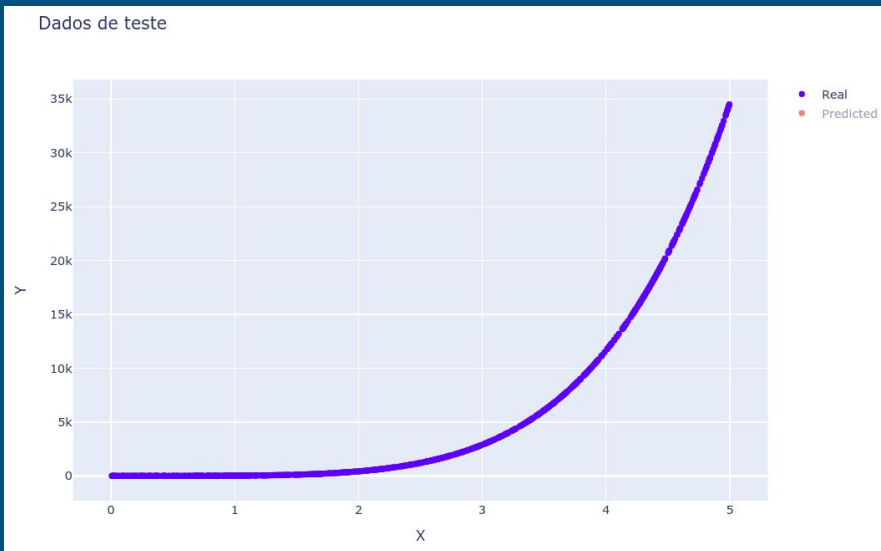
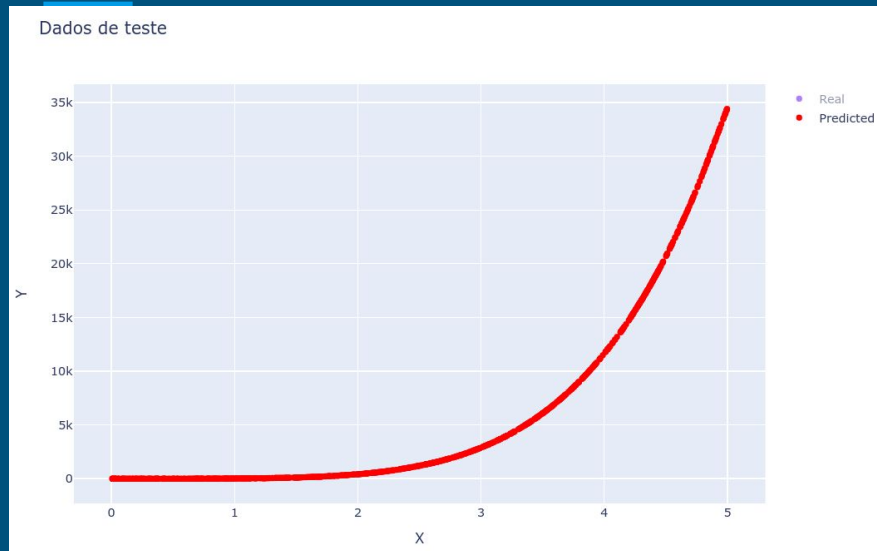
```
class PerceptronPol(nn.Module):
    def __init__(self):
        super(PerceptronPol, self).__init__()
        self.fc1 = nn.Linear(1, 100)
        self.fc2 = nn.Linear(100, 100)
        self.fc3 = nn.Linear(100, 100)
        self.fc4 = nn.Linear(100, 100)
        self.fc5 = nn.Linear(100, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = self.fc5(x)
        return x
```

Arquitetura da rede

Questão 02

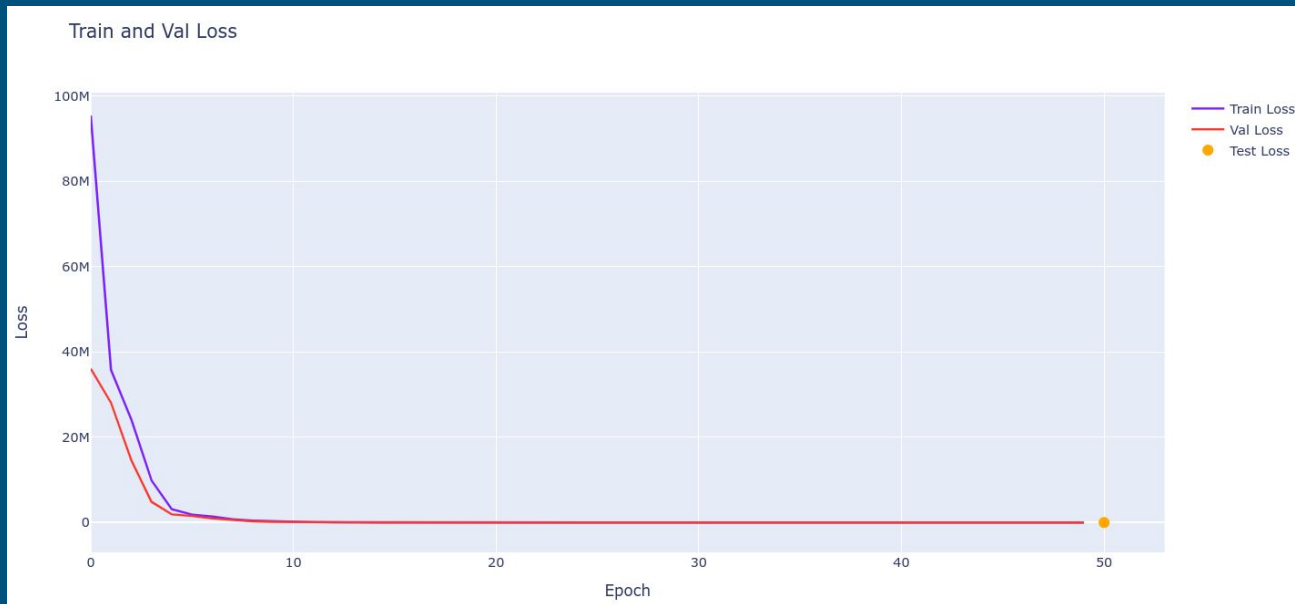
c) $10x^5 + 5x^4 + 2x^3 - 0.5x^2 + 3x + 2$



Plot da curva esperada e da curva encontrada pelo modelo

Questão 02

c) $10x^5 + 5x^4 + 2x^3 - 0.5x^2 + 3x + 2$



Erro por época para os dados de
treino e validação

QUESTÃO

0 03

Questão 03

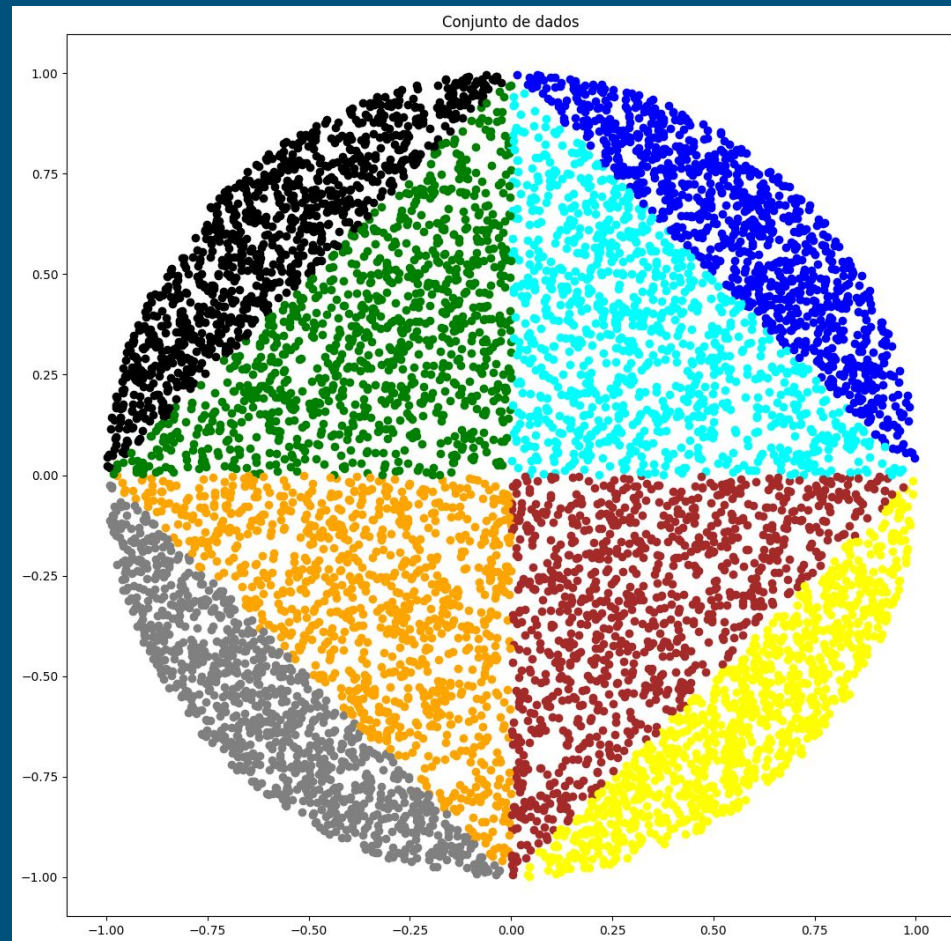
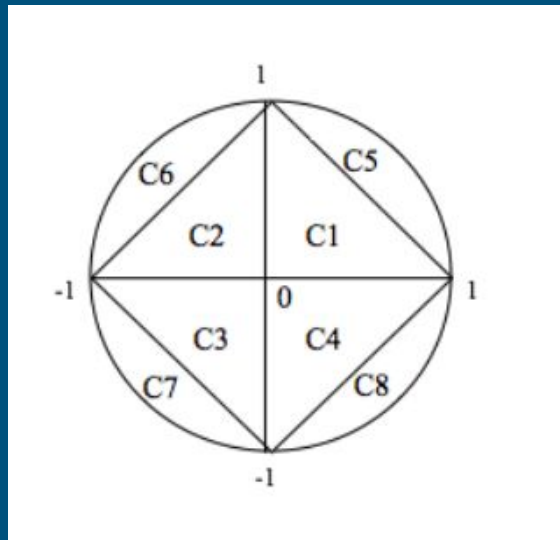
Usando a biblioteca Shapely e Math para operações geométricas, foi criada a classe CreateData. Para o treinamento da rede, foi utilizado o TensorFlow.

class CreateData

- Restrições:
 - Círculo centrado na origem de raio unitário;
 - Losango (contido no círculo), centrado na origem e com lados iguais à raiz de 2;
- Resultado:
 - Cria 4 polígonos (C1, C2, C3 e C4);
 - Cria 4 setores (C5, C6, C7, C8);

Por fim são atribuído os labels aos pontos, indicando a qual polígono ou setor cada ponto pertence.

Questão 03



Regra Delta Convencional

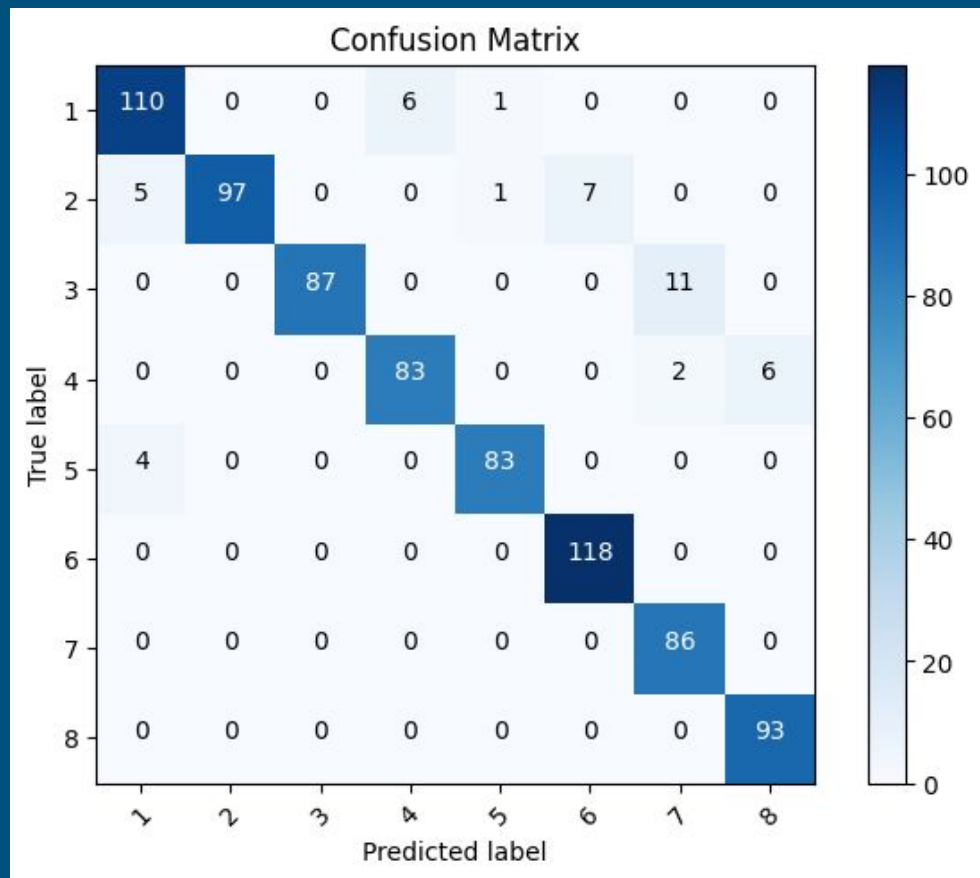
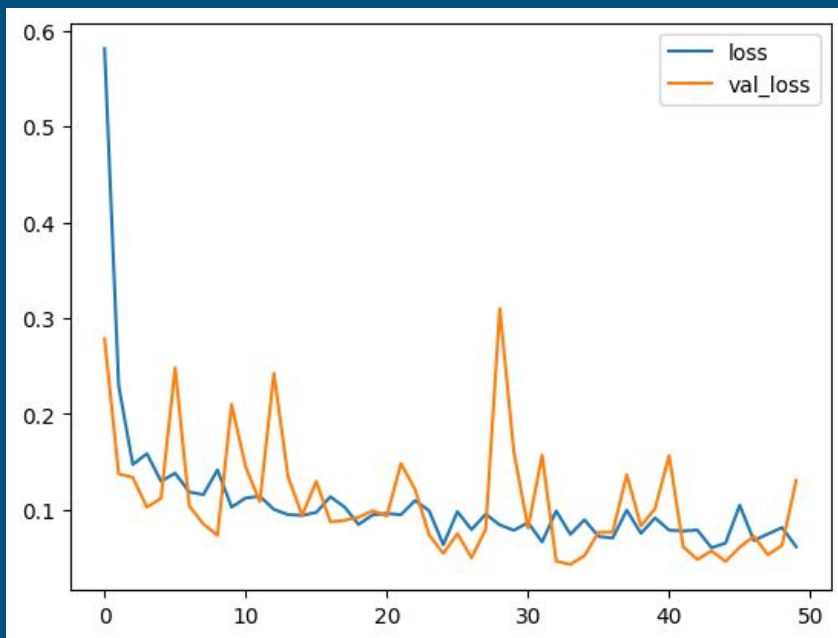
Usando a biblioteca do TensorFlow, foi importada a classe Adam.

Adam

- Tamanho do conjunto de validação: 0.2
- Tamanho do batch: 22
- Tamanho da época: 50
- Taxa de aprendizado: 0.01
- Função de Perda: Entropia Cruzada

```
model_convencional = Sequential([  
    Dense(units=16, input_shape=(2,)), activation='relu'),  
    Dense(units=32, activation='relu'),  
    Dense(units=8, activation='softmax'),  
])
```

Acurácia: 94,625%



Regra Delta com Termo do Momento

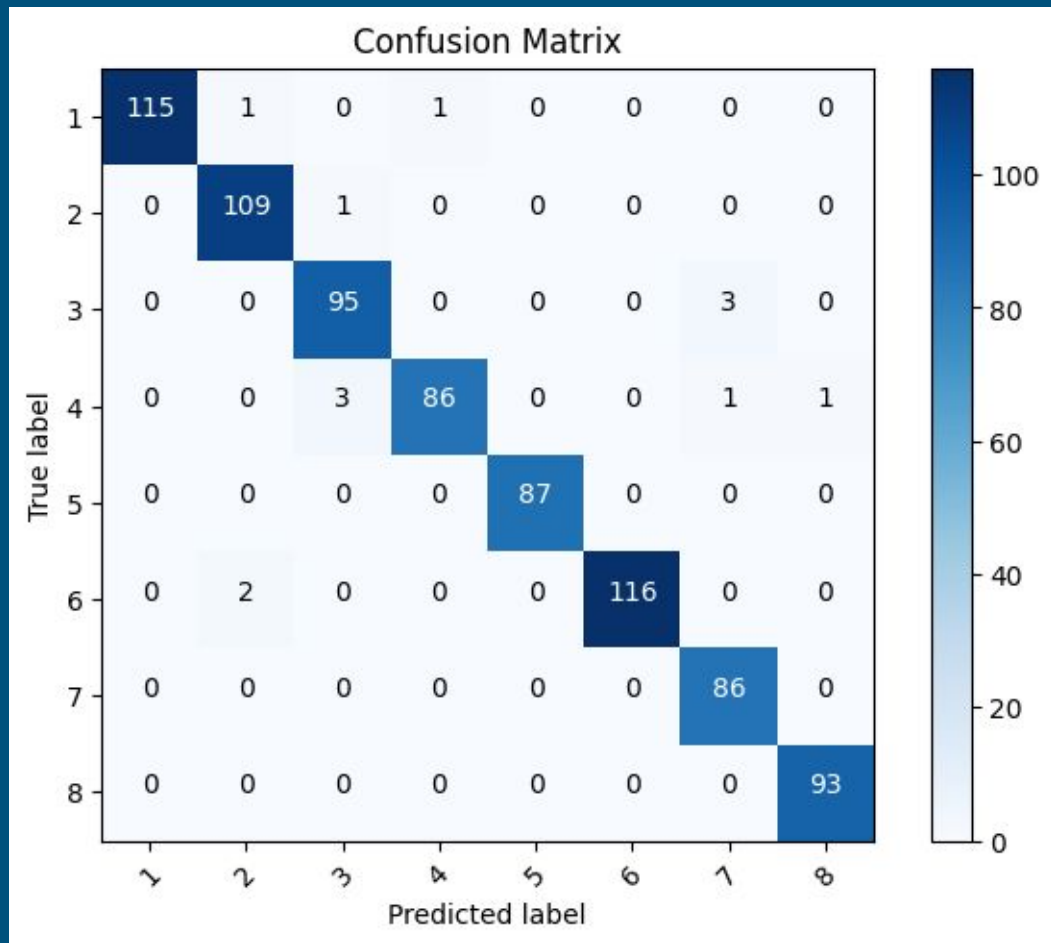
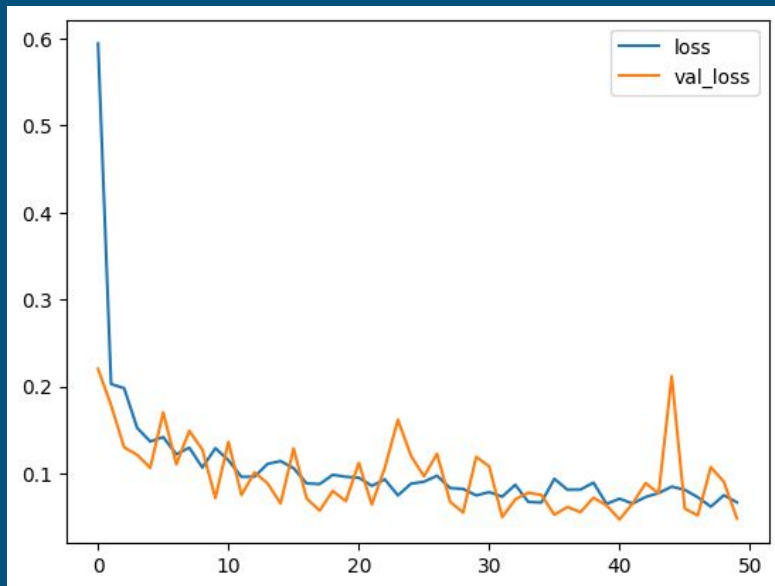
Adam

Usando a biblioteca do TensorFlow, foi importada a classe Adam.

- Tamanho do conjunto de validação: 0.2
- Tamanho do batch: 22
- Tamanho da época: 50
- Taxa de aprendizado: 0.01
- Função de Perda: Entropia Cruzada
- Beta_1: 0.9

```
model_momento = Sequential([
    Dense(units=16, input_shape=(2,), activation='relu'),
    Dense(units=32, activation='relu'),
    Dense(units=8, activation='softmax'),
])
```

Acurácia: 98,375%



QUESTÃO

0 04

Questão 04

Usa a biblioteca TensorFlow.

def generate_time_series_data

Para uma sequência de 10 pontos de entrada, gera uma sequência de 3 pontos de saída

- Tamanho do conjunto de validação: 0.2
- Tamanho do batch: 50
- Tamanho da época: 100
- Taxa de aprendizado: 0.01
- Função de perda: Erro Médio Quadrático

```
neural_network_model = Sequential([
    Dense(units=5, input_shape=(34,), activation='relu'),
    Dense(units=10, activation='linear'),
])
```


Erro Médio Absoluto:
2,941%

