Atividade: *Perceptron* - Petróleo

INSTITUTO FEDERAL DE MINAS GERAIS

Departamento de Engenharia e Computação

Professor: Ciniro Nametala Leite

Aluno: Luan Carlos dos Santos

Pacotes necessários

In [68]:
```
import numpy as np
import pandas as pd
import pygwalker as pyg
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report,  ConfusionM
import seaborn as sns
import plotly.graph_objects as go
```

Funções

- Degrau Bipolar

In [69]:
```
def degrau_bipolar(u):
   if u >= 0:
      y = 1 # óleo P1
   else:
      y = -1 # óleo P2
   return y
```

- Previsão

In [70]:
```
def previsao(w, x):
   u = np.dot(w.T, x)
   yhat = degrau_bipolar(u)
   return yhat # esse y hat é a previsão da perceptron
```

Conjunto de dados

- Coletando os dados de treinamento;

In [71]:
```
dados = pd.read_csv('../data-sets/tabela_treino.csv', sep=';')
```

- Armazenando a quantidade de amostras, variáveis e matriz de treinamento;

In [72]:
```
n_amostras = dados.shape[0]
n_variaveis = dados.shape[1] -1
x = dados.iloc[:, 1:n_variaveis].values
pd.DataFrame(x)[:5]
```

Out[72]:

	0	1	2
0	-0.6508	0.1097	40.009
1	-14.4920	0.8896	44.005
2	2.0850	0.6876	79.845
3	0.2626	11.4760	12.071
4	0.6418	10.2340	77.985

- Inserindo o "*bias*" na matriz de treinamento;

In [73]:
```
# Criando um vetor preenchido com '1s' * -1
bias = np.ones((n_amostras, 1)) * -1
x = np.hstack((x, bias))
pd.DataFrame(x)[:5]
```

Out[73]:

	0	1	2	3
0	-0.6508	0.1097	40.009	-1.0
1	-14.4920	0.8896	44.005	-1.0
2	2.0850	0.6876	79.845	-1.0
3	0.2626	11.4760	12.071	-1.0
4	0.6418	10.2340	77.985	-1.0

- Armazenando o vetor de saida;

In [74]:
```
y = dados['d'].values
print(y)

print(f"Quantidade de amostras tipo P1(1): {(y == 1).sum()}")
print(f"Quantidade de amostras tipo P2(-1): {(y== -1).sum()}")
```

```
[-1. -1. -1. -1.  1.  1. -1.  1. -1.  1. -1. -1. -1. -1. -1. -1.  1.  1.
  1.  1. -1.  1.  1.  1.  1. -1. -1.  1. -1.  1.]
Quantidade de amostras tipo P1(1): 14
Quantidade de amostras tipo P2(-1): 16
```

- Inicializando o vetor de pesos w com valores aleatórios de 0 a 1;

In [75]:
```
w = np.random.uniform(0, 1, n_variaveis)
print(w)
```

```
[0.68970268 0.91044318 0.46510501 0.18456377]
```

Pré configurações de Treinamento

- **Definindo a taxa de aprendizagem η, a tolerância e o erro médio inicial;**

In [76]:
```
eta = 0.01
tolerancia = 0.01
erro_medio = tolerancia + 0.1
```

- Criando um vetor de épocas, contador e o máximo de épocas;

In [77]:
```
erros_epocas = []
epoca = 0
max_epocas = 400
```

- Inicializando os critérios de parada do treinamento;

In [78]:
```
criterio_erro = True
criterio_epoca = True
```

1. Treinamento da RNA com Arquitetura *Perceptron*

"5 Treinamentos com pesos aleatórios entre 0 e 1"

In [79]:
```
n_treinamentos = 5
pesos_iniciais = []
pesos_finais = []
erros_treinamentos = []

for treinamento in range(n_treinamentos):

   # reinicializando os pesos aleatórios
   w = np.random.uniform(0, 1, n_variaveis)
   pesos_iniciais.append(w.copy())

   # reinicia variáveis de controle
   criterio_erro = True
   criterio_epoca = True
   epoca = 0
   erros_epocas = []

   while criterio_erro and criterio_epoca:
      erro_atual = 0
      epoca += 1

      # embaralhando os indíces pra usar na rede
      indice_aleatorios = np.random.permutation(n_amostras)

      # para cada amostra no conjunto de amostras
      for i in range(n_amostras):
```

```
        # pegando indice_aleatorio pra amostra i
        i_amostra = indice_aleatorios[i]

        # capturando uma amostra da matriz de treinamento vinculada ao indice c
        amostra_atual = x[i_amostra,:]

        # calculando o potencial de ativação
        u = np.dot(w.T, amostra_atual)

        # obtendo previsão da rede pela Degrau Bipolar
        yhat = degrau_bipolar(u)

        # calculando o erro entre a previsão e o esperado
        e = y[i_amostra] - yhat

        # atualizando os pesos pela regra de HEBB
        w = w + ((eta * e) * amostra_atual)

        # acumulando os erros de de todas as amostras
        erro_atual += np.abs(e)

    # calcular o erro médio da época
    erro_medio = erro_atual/n_amostras

    # adicionando o erro medio ao vetor de erros do treinamento
    erros_epocas.append(erro_medio)
    # verificando e atualizando os critérios de parada
    criterio_erro = (erro_medio > tolerancia)
    criterio_epoca = (epoca < max_epocas)
  erros_treinamentos.append(erros_epocas) # pegando os erros_epocas de cada tre
  pesos_finais.append(w.copy())
  # print(f'Época: {epoca}')
```

2. Registro dos pesos w antes e depois

In [80]:

```
# criando nomes das colunas
colunas_iniciais = [f'w{i}' for i in range(n_variaveis)]
colunas_finais = [f'w{i}' for i in range(n_variaveis)]
colunas = colunas_iniciais + colunas_finais

# juntando pesos iniciais e finais
dados = np.hstack([pesos_iniciais, pesos_finais])

# cria uma coluna de treinos
treinos = [f'Treino {i}' for i in range(n_treinamentos)]

# Cria DataFrame simples
df_pesos = pd.DataFrame(dados, columns=colunas)
df_pesos.insert(0, 'Treino', treinos)

df_pesos
```

Out[80]:

	Treino	w0	w1	w2	w3	w0	w1	w2	
0	Treino 0	0.895190	0.603932	0.491345	0.848153	-1.839922	1.297238	-0.125495	6.248
1	Treino 1	0.310746	0.401330	0.999289	0.704244	-0.991368	1.072610	-0.522831	7.424
2	Treino 2	0.042412	0.813961	0.378428	0.640372	-0.609266	1.150101	-0.844452	6.640
3	Treino 3	0.029720	0.260290	0.988193	0.195458	-1.752714	1.642508	0.524633	5.935
4	Treino 4	0.934188	0.979633	0.542214	0.760576	-1.219260	1.261433	0.113934	6.120

3. Implementação do *Perceptron*

In [81]:

```
# lê os dados de teste
dados_classificacao = pd.read_csv('../data-sets/tabela_classificacao.csv', sep='
n_amostras_teste = dados_classificacao.shape[0]
bias = np.ones((n_amostras_teste, 1)) * -1
x_teste = np.hstack((dados_classificacao, bias))

previsoes_treinamento = []
for i in range(n_treinamentos):
   # vetor que recebe as previsões da rna para o treino i
   yhat_teste = np.zeros(n_amostras_teste)
   # fazendo as previsoes pra cada modelo de treino
   for j in range(n_amostras_teste):
      yhat_teste[j] = previsao(pesos_finais[i], x_teste[j, :])
   previsoes_treinamento.append(yhat_teste) # guardando as

# converte a lista de previsões em array 2D (treinos x_teste amostras)
previsoes_array = np.array(previsoes_treinamento)

# transpõe pra (amostras x treinos)
previsoes_transpostas = previsoes_array.T

# junta tudo: amostras + dados_classificacao + previsões
resultado = np.hstack((dados_classificacao, previsoes_transpostas))

# nomes das colunas
colunas_x = ['X1', 'X2', 'X3']
colunas_prev = [f'ŷ(T{i+1})' for i in range(previsoes_transpostas.shape[1])]
colunas = colunas_x + colunas_prev

# cria o DataFrame final
df_final = pd.DataFrame(resultado, columns=colunas)

df_final
```

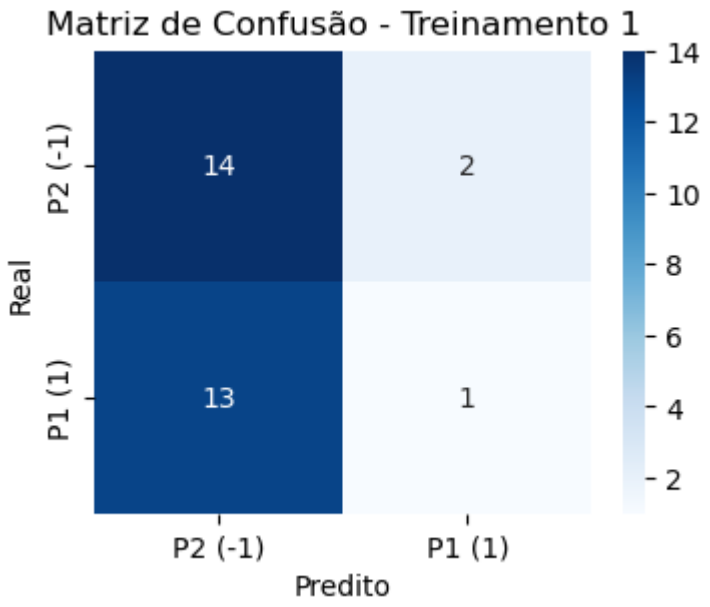
Out[81]:

	X1	X2	X3	ŷ(T1)	ŷ(T2)	ŷ(T3)	ŷ(T4)	ŷ(T5)
0	-0.3665	0.0620	5.9891	-1.0	-1.0	-1.0	-1.0	-1.0
1	-0.7842	1.1267	5.5912	-1.0	-1.0	-1.0	1.0	-1.0
2	0.3012	0.5611	5.8234	-1.0	-1.0	-1.0	-1.0	-1.0
3	0.7757	1.0648	8.0677	-1.0	-1.0	-1.0	-1.0	-1.0
4	0.1570	0.8028	6.3040	-1.0	-1.0	-1.0	-1.0	-1.0
5	-0.7014	1.0316	3.6005	-1.0	-1.0	-1.0	-1.0	-1.0
6	0.3748	0.1536	6.1537	-1.0	-1.0	-1.0	-1.0	-1.0
7	-0.6920	0.9404	4.4058	-1.0	-1.0	-1.0	-1.0	-1.0
8	-1.3970	0.7141	4.9263	-1.0	-1.0	-1.0	1.0	-1.0
9	-1.8842	0.2805	1.2548	-1.0	-1.0	-1.0	-1.0	-1.0

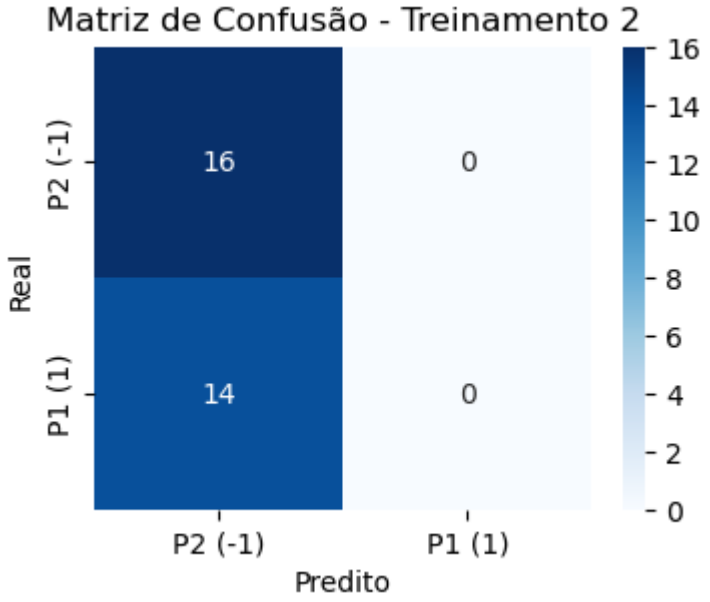
4. Taxa de acerto percentual de cada modelo

- Matriz de confusão

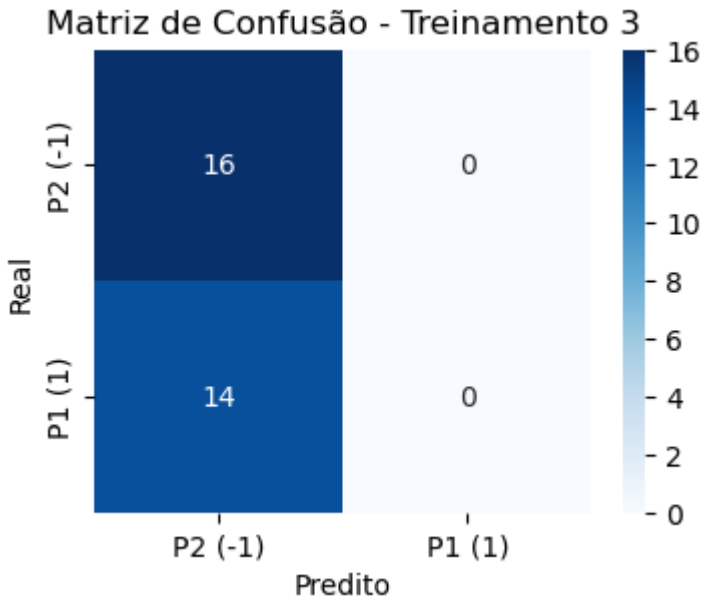
In [82]:
```
# converte -1 -> 0 apenas para visualização
y_display = np.where(y == -1, 0, y)

for i in range(n_treinamentos):
    # previsões no conjunto de treinamento
    yhat_treino = np.array([degrau_bipolar(np.dot(pesos_finais[i].T, x[j,:]))
                            for j in range(x.shape[0])])

    # converter -1 → 0 para exibir na matriz
    yhat_display = np.where(yhat_treino == -1, 0, yhat_treino)

    # gerar matriz de confusão
    cm = confusion_matrix(y_display, yhat_display)

    # calcular acurácia
    acuracia = np.sum(yhat_treino == y) / len(y) * 100

    # plotar matriz de confusão
    plt.figure(figsize=(4,3))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['P2 (-1)', 'P1 (1)'],
                yticklabels=['P2 (-1)', 'P1 (1)'])
    plt.title(f'Matriz de Confusão - Treinamento {i+1}')
    plt.xlabel('Predito')
    plt.ylabel('Real')
    plt.show()
    print(f"Acurácia: {acuracia:.2f}%")
```



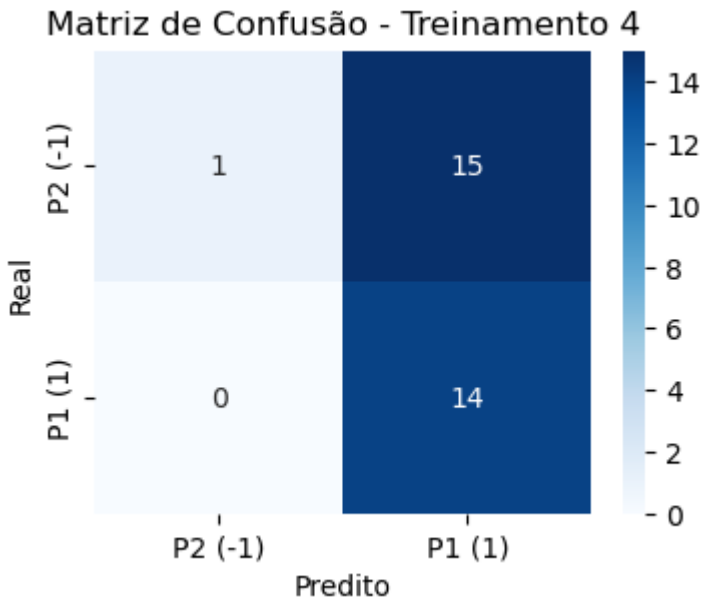
Acurácia: 50.00%



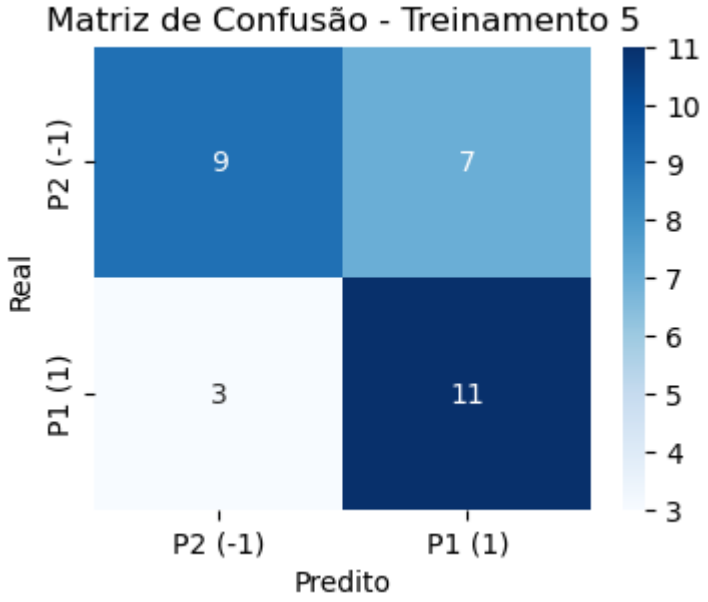
Acurácia: 53.33%



Acurácia: 53.33%




```
Acurácia: 50.00%
```

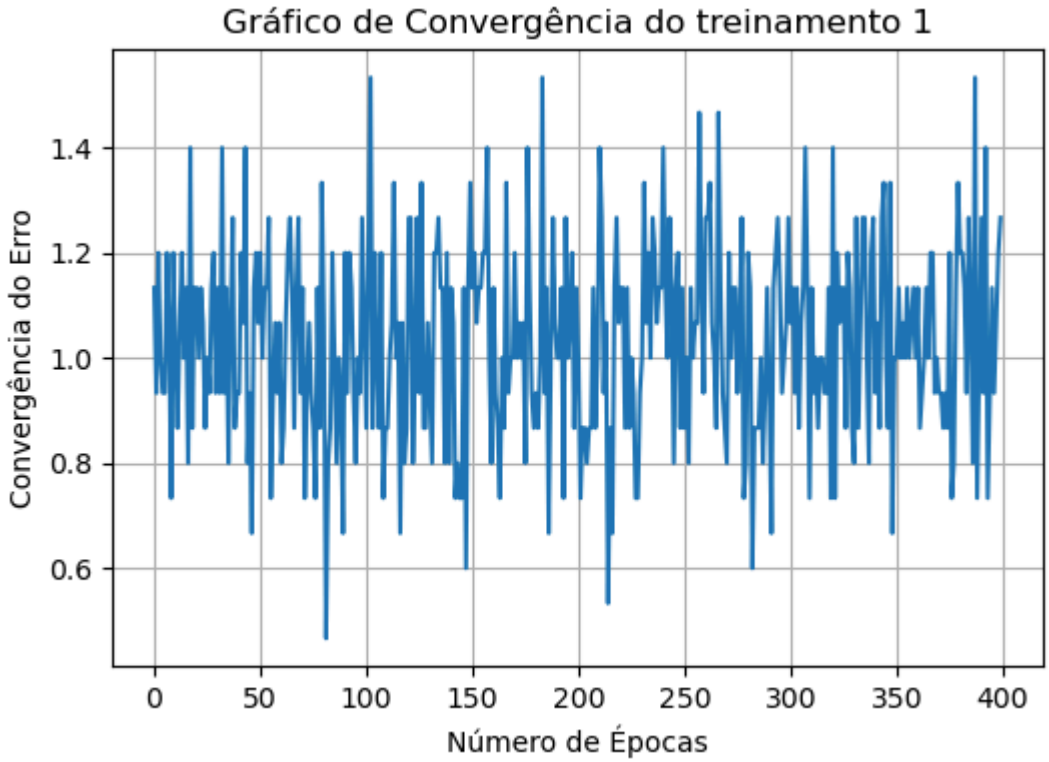


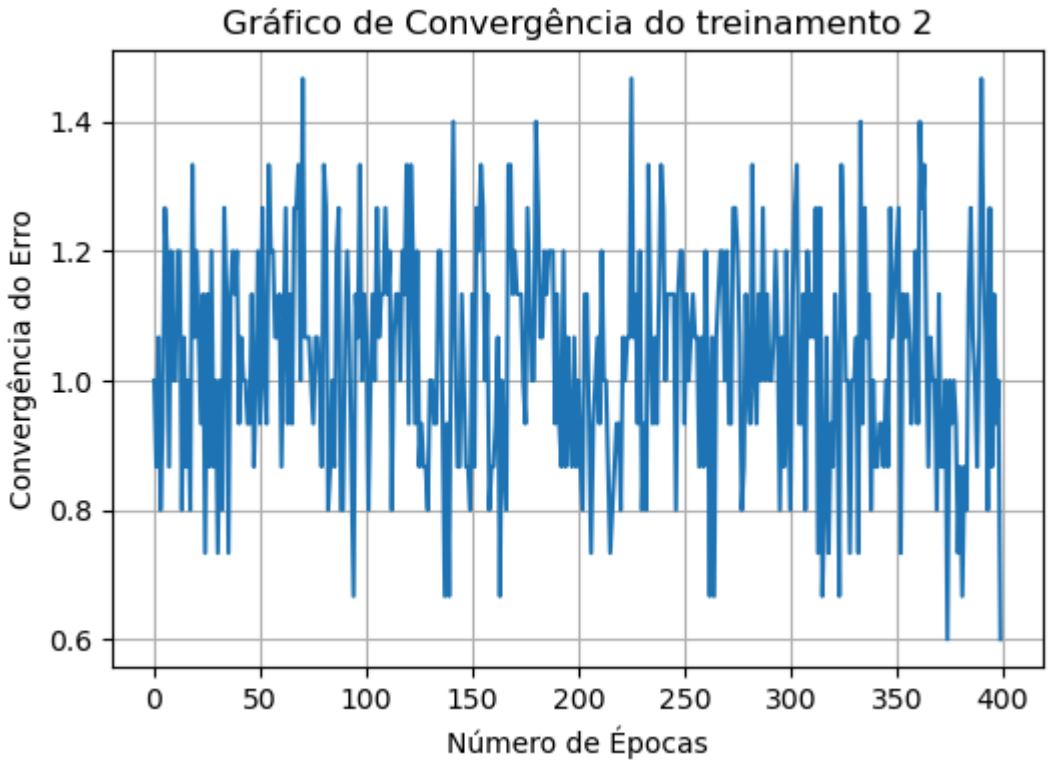

```
Acurácia: 66.67%
```

- Análise do Gráfico de Convergência do Algorítmo com base no erro de cada época:

In [83]:
```python
previsoes_treinamento = []

# --- Geração das previsões ---
for i in range(n_treinamentos):
    yhat_teste = np.zeros(n_amostras)
    for j in range(n_amostras):
        yhat_teste[j] = previsao(pesos_finais[i], x[j, :])
    previsoes_treinamento.append(yhat_teste)

acuracia = []

# --- Plot e cálculo da acurácia ---
for i in range(n_treinamentos):
    plt.figure(figsize=(6,4))
    plt.plot(erros_treinamentos[i][:max_epocas])
    plt.title(f'Gráfico de Convergência do treinamento {i+1}')
```

```
    plt.xlabel('Número de Épocas')
    plt.ylabel('Convergência do Erro')
    plt.grid(True)
    plt.show()
```





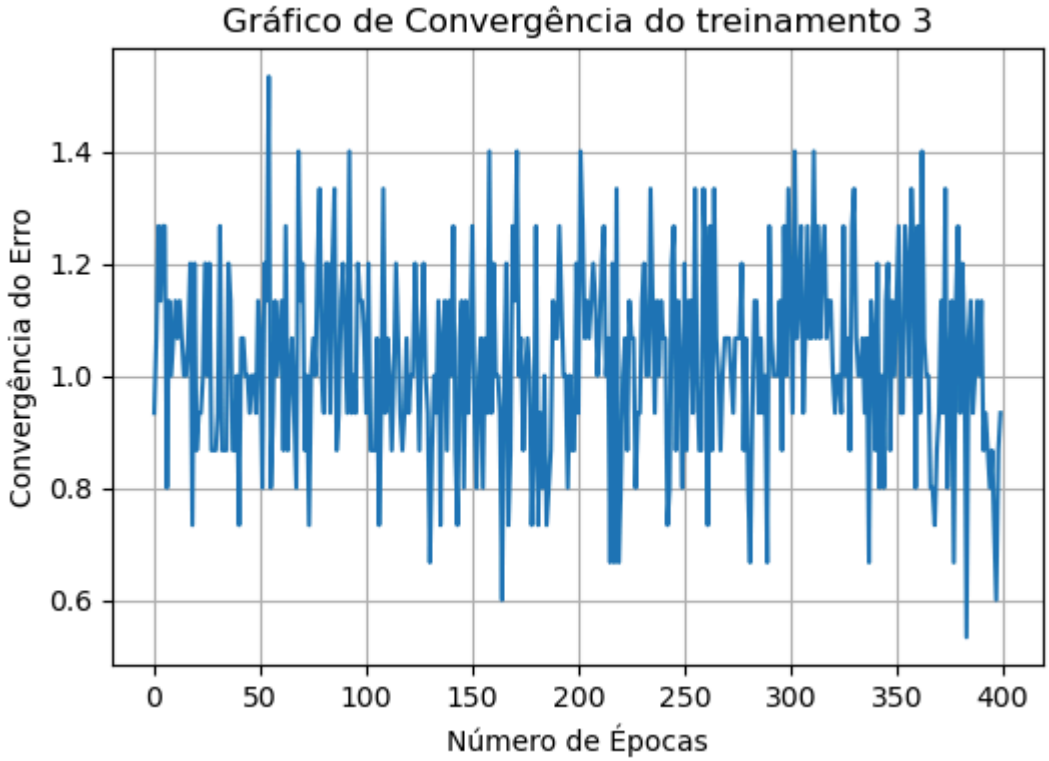


Gráfico de Convergência do treinamento 3
Convergência do Erro
1.4
1.2
1.0
0.8
0.6
0
50
100
150
200
250
300
350
400
Número de Épocas

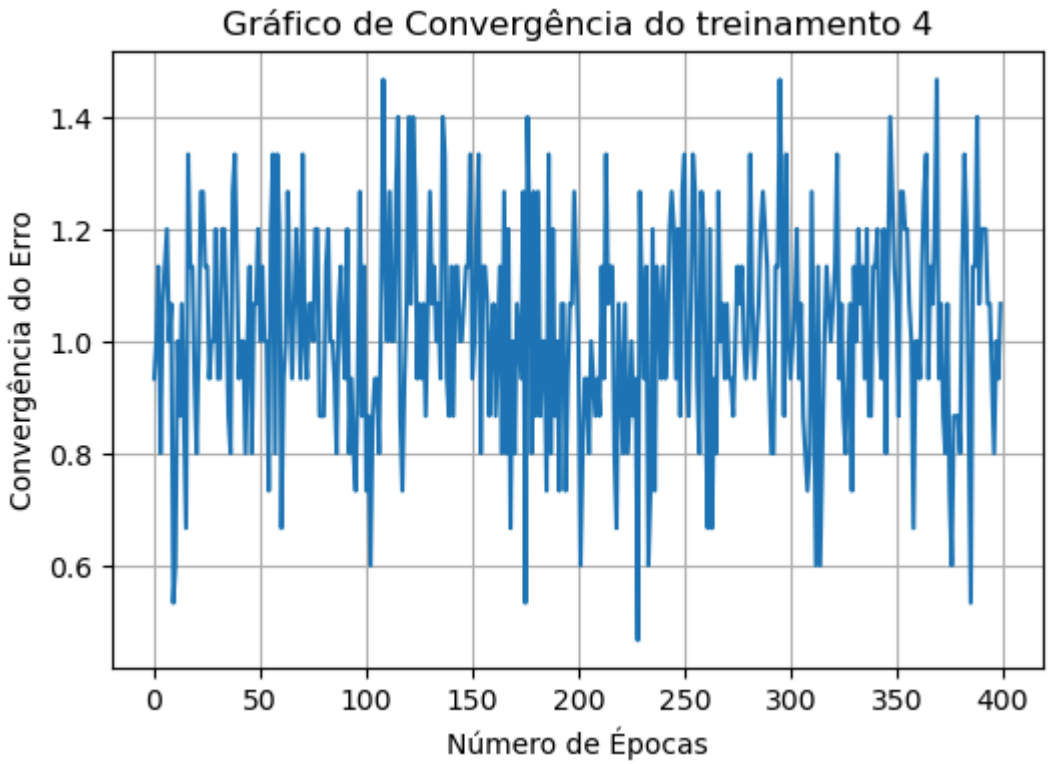
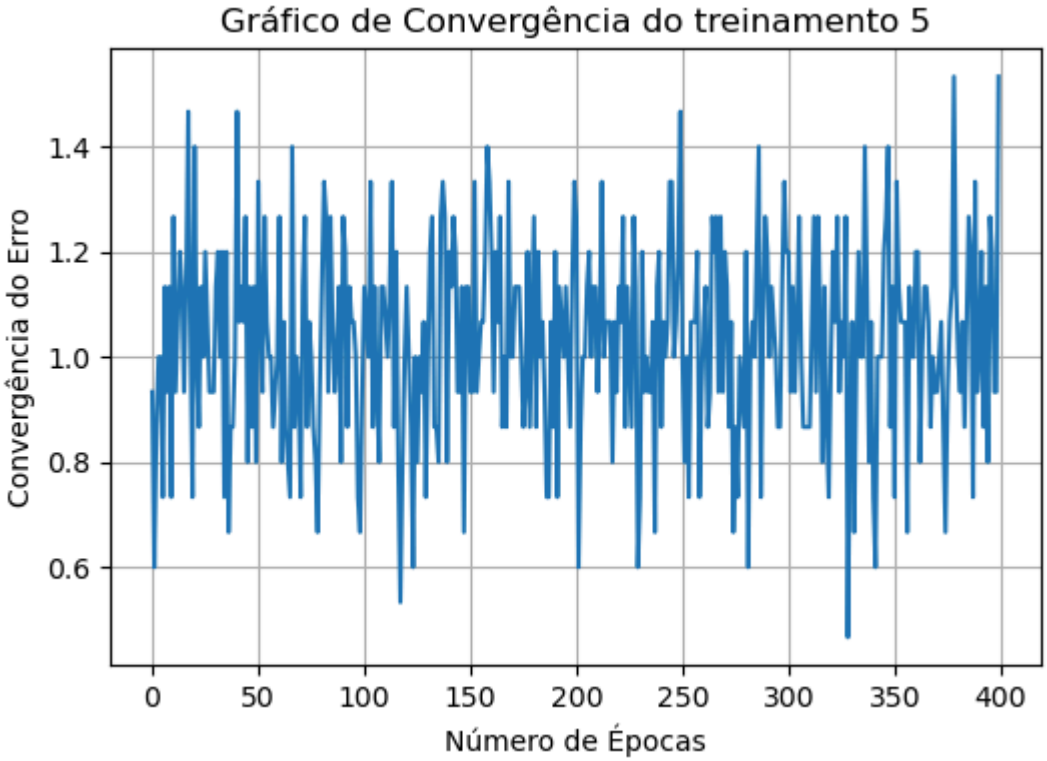


Gráfico de Convergência do treinamento 4
Convergência do Erro
1.4
1.2
1.0
0.8
0.6
0
50
100
150
200
250
300
350
400
Número de Épocas



5. Qual o efeito de aumentar ou diminuir o número de épocas na qualidade dos resultados?

Nenhum pois os dados não podem ser classificados;

6. Qual o efeito de aumentar ou diminuir a taxa de aprendizagem na qualidade dos resultados?

Independente do valor da taxa de aprendizagem, a qualidade dos resultado fica inferior a 60%;

7. Discorra se é possível afirmar se as suas classes, neste problema, são linearmente separáveis.

Não é possível afirmar pois em todos os testes a taxa de acertos foi imprecisa e em nenhum dos casos convergiu!