COM510 - Redes Neurais

Exercício de Apoio - Semana 03 (Parte 01)

Rede Neural Multilayer Perceptron MLP (Classificação)

- Carregamento dos pacotes necessários
- Carregamento dos dados
- Definição do modelo
- Treinamento de modelo
- Gráficos e Avaliação
- Exercício de Apoio

Pacotes

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import sklearn.datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
```

Dados

Conjunto de dados Iris

Tiia Monto, CC BY-SA 4.0, via Wikimedia Commons
Charles de Mille-Isles from Mille-Isles, Canada, CC BY 2.0, via Wikimedia Commons
Robert H. Mohlenbrock. Courtesy of USDA NRCS, Public domain, via Wikimedia Commons

Este dataset contém 150 exemplos, sendo 50 de cada classe: Setosa, Virginica e Versicolor. Os exemplos são caracterizados por quatro atributos: comprimento e largura das pétalas e sépalas.

https://en.wikipedia.org/wiki/Iris_ _ower_data_set

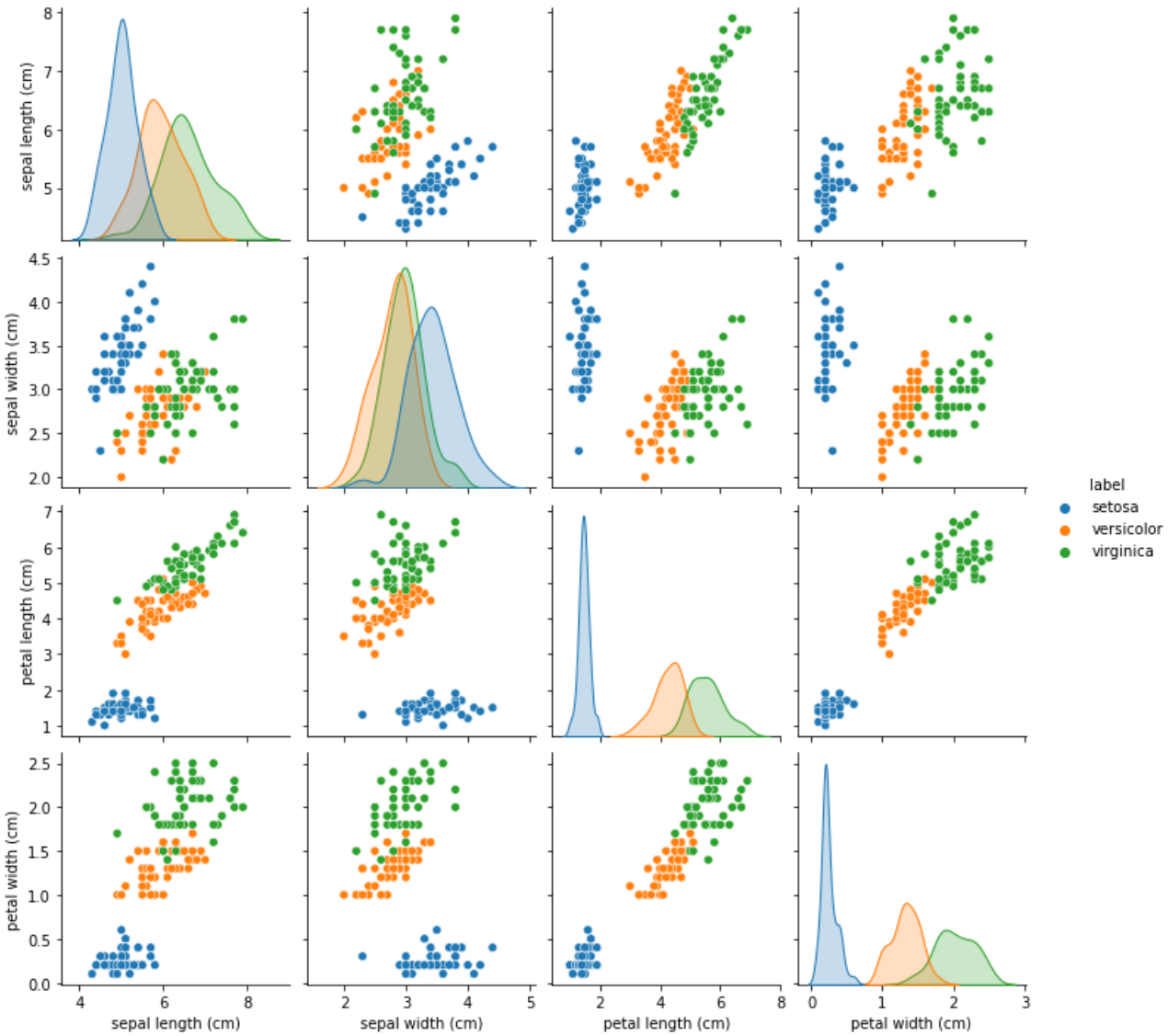
Carga dos Dados

```
dados = sklearn.datasets.load_iris()
# Imprimindo algumas informações sobre o conjunto de dados
print("Atributos:",dados['feature_names'])
print("Classes (labels):",dados['target_names'])
print("Dimensões:", dados['data'].shape)

df = pd.DataFrame(dados.data, columns = dados.feature_names)
df['label'] = [dados.target_names[i] for i in dados.target]
```

```
Atributos: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Classes (labels): ['setosa' 'versicolor' 'virginica']
Dimensões: (150, 4)
```

```
# Pair Plot
sns.pairplot(df, hue = 'label')
```

```
<seaborn.axisgrid.PairGrid at 0x7fee5f5e36a0>
```



Preprocessamento dos Dados

```
X = df.drop(['label'], axis=1)
# Normalização dos dados (Min-Max)
normalizador = MinMaxScaler()
X_norm = pd.DataFrame(normalizador.fit_transform(X), columns=X.columns)
labels = df.label.unique()
print(labels)

#Transformação dos atributos (classes) categóricos em
#               numéricos (1-de-c / one-hot-encoding)
# setosa -> 0 0 1
# versicolor -> 0 1 0
# virginica -> 1 0 0

Y = pd.get_dummies(df.label)
print(Y)
```

```
['setosa' 'versicolor' 'virginica']
     setosa  versicolor  virginica
0         1           0          0
1         1           0          0
2         1           0          0
3         1           0          0
4         1           0          0
..      ...         ...        ...
145       0           0          1
146       0           0          1
147       0           0          1
148       0           0          1
149       0           0          1

[150 rows x 3 columns]
```

Separação Treino/Validação/Teste

```
# Separação desenvolvimento (90) e teste (10)
X_dev, X_test, Y_dev, Y_test = train_test_split(X_norm, Y, test_size=0.1, random_state=1)

# Separação treino (80) e validação (20)
X_train, X_val, Y_train, Y_val = train_test_split(X_dev, Y_dev, test_size=0.2, random_state=1)

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(108, 4)
(27, 4)
(15, 4)
```

Modelo

```
class MLP(nn.Module):
    def __init__(self,input_dim,output_dim):
        super(MLP,self).___init___()
        self.rede = nn.Sequential(
            nn.Linear(input_dim,10),
            nn.Tanh(),
            nn.Linear(10,output_dim),
            nn.Sigmoid(),
        )
    def forward(self,x):
        out =  self.rede(x)
        return out
```

Instanciação do Modelo

```
input_dim  = 4 # número de atributos do Iris
output_dim = 3 # número de classes

modelo = MLP(input_dim,output_dim) # Criação do modelo (rede)
```

```
from torchsummary import summary

print(modelo)
summary(modelo, (150,4))
```

```
MLP(
  (rede): Sequential(
    (0): Linear(in_features=4, out_features=10, bias=True)
    (1): Tanh()
    (2): Linear(in_features=10, out_features=3, bias=True)
    (3): Sigmoid()
  )
)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1              [-1, 150, 10]              50
              Tanh-2              [-1, 150, 10]               0
            Linear-3               [-1, 150, 3]              33
           Sigmoid-4               [-1, 150, 3]               0
================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.03
Params size (MB): 0.00
Estimated Total Size (MB): 0.03
----------------------------------------------------------------
```

Otimizador e Função de Custo

```
eta = 0.2
loss_function = nn.MSELoss()
optimizer = torch.optim.SGD(modelo.parameters(),lr=eta)
```

Treinamento

Transformação dos dados em tensores Pytorch

```
x_train = torch.FloatTensor(X_train.values)
y_train = torch.FloatTensor(Y_train.values)

x_val = torch.FloatTensor(X_val.values)
y_val = torch.FloatTensor(Y_val.values)

x_test = torch.FloatTensor(X_test.values)
y_test = torch.FloatTensor(Y_test.values)
```

```
# verificando disponibilidade da gpu
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cpu')
```

Laço de treinamento da rede

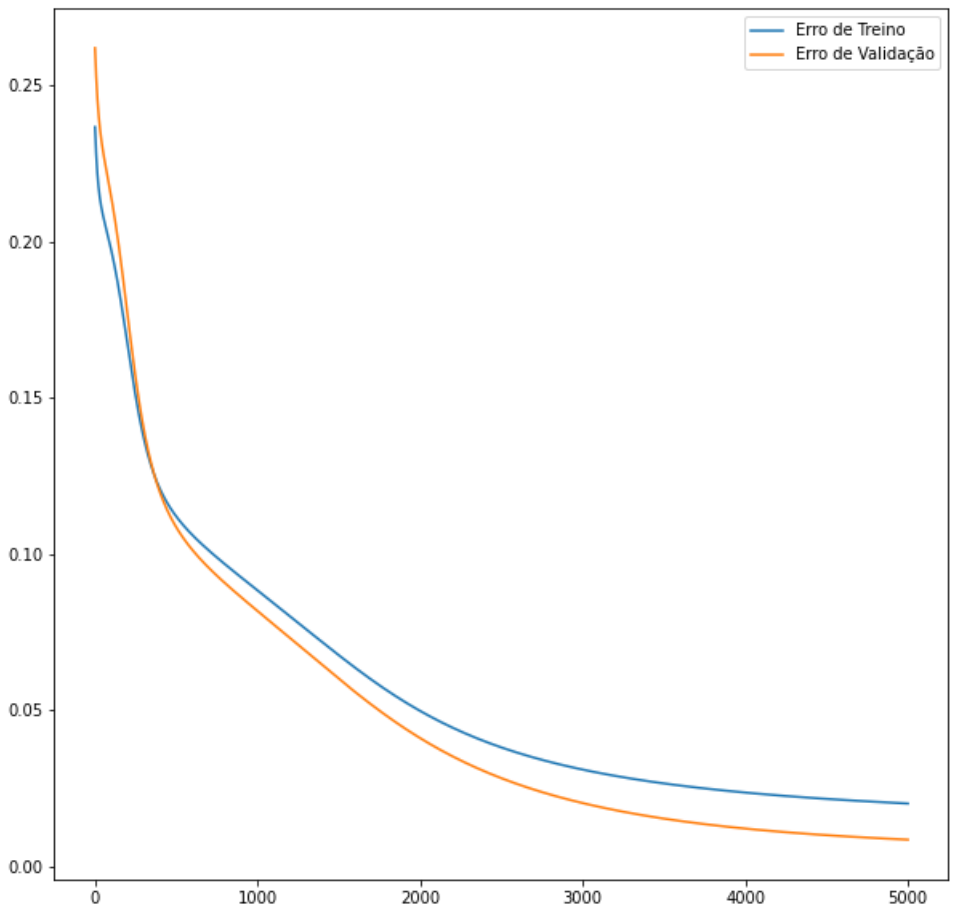
```
def train_network(model,optimizer,loss_function,x_train,y_train,x_val,y_val,num_epochs,train_losses,val_
    for epoch in range(num_epochs):
        # zerando os gradientes da época anterior
        optimizer.zero_grad()

        # fase de propagação
        output_train = model(x_train)

        # cálculo do erro (função de custo - loss function)
        loss_train = loss_function(output_train, y_train)

        # fase de retroprogação
        loss_train.backward()

        # atualização dos pesos da rede
        optimizer.step()

        # avaliando o modelo com o conjunto de validação
        output_val = model(x_val)
        loss_val = loss_function(output_val,y_val)

        train_losses[epoch] = loss_train.item()
        val_losses[epoch] = loss_val.item()

        if (epoch + 1) % 100 == 0:
            print(f"Epoch {epoch+1}/{num_epochs}, Erro Treino: {loss_train.item():.4f},  Erro Validação:


num_epochs = 5000
train_losses = np.zeros(num_epochs)
val_losses  = np.zeros(num_epochs)

train_network(modelo,optimizer,loss_function,x_train,y_train,x_val,y_val,num_epochs,train_losses,val_los
```

```
Epoch 100/5000, Erro Treino: 0.1974,  Erro Validação: 0.214
Epoch 200/5000, Erro Treino: 0.1673,  Erro Validação: 0.177
Epoch 300/5000, Erro Treino: 0.1371,  Erro Validação: 0.140
Epoch 400/5000, Erro Treino: 0.1208,  Erro Validação: 0.120
Epoch 500/5000, Erro Treino: 0.1120,  Erro Validação: 0.109
Epoch 600/5000, Erro Treino: 0.1060,  Erro Validação: 0.101
Epoch 700/5000, Erro Treino: 0.1011,  Erro Validação: 0.095
Epoch 800/5000, Erro Treino: 0.0966,  Erro Validação: 0.091
Epoch 900/5000, Erro Treino: 0.0924,  Erro Validação: 0.086
Epoch 1000/5000,  Erro Treino: 0.0882,  Erro Validação: 0.082
Epoch 1100/5000,  Erro Treino: 0.0841,  Erro Validação: 0.077
Epoch 1200/5000,  Erro Treino: 0.0800,  Erro Validação: 0.073
Epoch 1300/5000,  Erro Treino: 0.0758,  Erro Validação: 0.069
Epoch 1400/5000,  Erro Treino: 0.0716,  Erro Validação: 0.064
Epoch 1500/5000,  Erro Treino: 0.0675,  Erro Validação: 0.060
Epoch 1600/5000,  Erro Treino: 0.0635,  Erro Validação: 0.056
Epoch 1700/5000,  Erro Treino: 0.0597,  Erro Validação: 0.052
Epoch 1800/5000,  Erro Treino: 0.0561,  Erro Validação: 0.048
Epoch 1900/5000,  Erro Treino: 0.0527,  Erro Validação: 0.044
Epoch 2000/5000,  Erro Treino: 0.0497,  Erro Validação: 0.041
Epoch 2100/5000,  Erro Treino: 0.0468,  Erro Validação: 0.038
Epoch 2200/5000,  Erro Treino: 0.0443,  Erro Validação: 0.035
Epoch 2300/5000,  Erro Treino: 0.0420,  Erro Validação: 0.033
Epoch 2400/5000,  Erro Treino: 0.0399,  Erro Validação: 0.030
Epoch 2500/5000,  Erro Treino: 0.0380,  Erro Validação: 0.028
Epoch 2600/5000,  Erro Treino: 0.0363,  Erro Validação: 0.026
Epoch 2700/5000,  Erro Treino: 0.0347,  Erro Validação: 0.024
Epoch 2800/5000,  Erro Treino: 0.0333,  Erro Validação: 0.023
Epoch 2900/5000,  Erro Treino: 0.0320,  Erro Validação: 0.021
Epoch 3000/5000,  Erro Treino: 0.0309,  Erro Validação: 0.020
```

```
Epoch 3100/5000, Erro Treino: 0.0298,   Erro Validação: 0.019
Epoch 3200/5000, Erro Treino: 0.0289,   Erro Validação: 0.018
Epoch 3300/5000, Erro Treino: 0.0280,   Erro Validação: 0.017
Epoch 3400/5000, Erro Treino: 0.0272,   Erro Validação: 0.016
Epoch 3500/5000, Erro Treino: 0.0264,   Erro Validação: 0.015
Epoch 3600/5000, Erro Treino: 0.0258,   Erro Validação: 0.014
Epoch 3700/5000, Erro Treino: 0.0251,   Erro Validação: 0.014
Epoch 3800/5000, Erro Treino: 0.0245,   Erro Validação: 0.013
Epoch 3900/5000, Erro Treino: 0.0240,   Erro Validação: 0.013
Epoch 4000/5000, Erro Treino: 0.0235,   Erro Validação: 0.012
Epoch 4100/5000, Erro Treino: 0.0231,   Erro Validação: 0.011
Epoch 4200/5000, Erro Treino: 0.0226,   Erro Validação: 0.011
Epoch 4300/5000, Erro Treino: 0.0222,   Erro Validação: 0.011
Epoch 4400/5000, Erro Treino: 0.0218,   Erro Validação: 0.010
Epoch 4500/5000, Erro Treino: 0.0215,   Erro Validação: 0.010
Epoch 4600/5000, Erro Treino: 0.0212,   Erro Validação: 0.010
Epoch 4700/5000, Erro Treino: 0.0208,   Erro Validação: 0.009
Epoch 4800/5000, Erro Treino: 0.0206,   Erro Validação: 0.009
Epoch 4900/5000, Erro Treino: 0.0203,   Erro Validação: 0.009
Epoch 5000/5000, Erro Treino: 0.0200,   Erro Validação: 0.008
```

Resultados

```
plt.figure(figsize=(10,10))
plt.plot(train_losses, label='Erro de Treino')
plt.plot(val_losses, label='Erro de Validação')
plt.legend()
plt.show()
```




```
predictions_train = []
predictions_val = []
predictions_test =  []
with torch.no_grad():
    predictions_train = modelo(x_train)
    predictions_val = modelo(x_val)
    predictions_test = modelo(x_test)
```

```
# Cálculo do erro (Função de Custo)
erro_train = loss_function(predictions_train,y_train)
erro_val = loss_function(predictions_val,y_val)
erro_test = loss_function(predictions_test,y_test)

print(f"Erro de Treino: {erro_train}")
print(f"Erro de Validação: {erro_val}")
print(f"Erro de Teste: {erro_test}")
```

```
Erro de Treino: 0.02001219242811203
Erro de Validação: 0.008439882658421993
Erro de Teste: 0.007400526665151119
```

```
# Cálculo da Acurácia de Classificação:

pred_train = torch.argmax(predictions_train, dim=1)
label_train = torch.argmax(y_train, dim=1)
pred_val = torch.argmax(predictions_val, dim=1)
label_val = torch.argmax(y_val, dim=1)
pred_test = torch.argmax(predictions_test, dim=1)
label_test = torch.argmax(y_test, dim=1)

from sklearn.metrics import accuracy_score
acc_train = accuracy_score(label_train, pred_train)
acc_val = accuracy_score(label_val, pred_val)
acc_test = accuracy_score(label_test, pred_test)


print(f"Acurácia de Treino: {acc_train*100:.2f}%")
print(f"Acurácia de Validação: {acc_val*100:.2f}%")
print(f"Acurácia de Teste: {acc_test*100:.2f}%")
```

```
Acurácia de Treino: 97.22%
Acurácia de Validação: 100.00%
Acurácia de Teste: 100.00%
```

Problema XOR

Modelo

```
#  torch.manual_seed(7) # aprende as portas Not-AND e Not-OR
torch.manual_seed(9) # aprende as portas AND e OR

class MLPXor(nn.Module):
    def __init__(self):
        super(MLPXor,self).__init__()
        self.hidden = nn.Linear(2, 2)
        self.output = nn.Linear(2, 1)
    def forward(self,x):
        hidden = self.hidden(x)
        x1 = torch.sigmoid(hidden)
        output = self.output(x1)
        x2 = torch.sigmoid(output)
        return x2, x1


modeloXOR = MLPXor() # Criação do modelo (rede)
```

```
MLPXor(
  (hidden): Linear(in_features=2, out_features=2, bias=True)
  (output): Linear(in_features=2, out_features=1, bias=True)
)
```

Otimizador e Função de Custo

```
loss_function = nn.MSELoss()
# optimizer = torch.optim.Adam(modeloXOR.parameters(),lr=eta)
optimizer = torch.optim.SGD(modeloXOR.parameters(),lr=0.02, momentum=0.9)
```

Dados (conjunto XOR)

```
x_xor = torch.Tensor([[0,0],[0,1], [1,0], [1,1]])
y_xor = torch.Tensor([0,1,1,0]).view(-1,1)
print(x_xor)
print(y_xor)
```

```
tensor([[0., 0.],
        [0., 1.],
        [1., 0.],
        [1., 1.]])
tensor([[0.],
        [1.],
        [1.],
        [0.]])
```

Treinamento

```
from torch.autograd import Variable

def train_xor(model,optimizer,loss_function,x_train,y_train,num_epochs,train_losses):
    for epoch in range(num_epochs):
      for j in range(4):
        exemplo = np.random.randint(4)
        x = Variable(x_xor[exemplo], requires_grad=False)
        y = Variable(y_xor[exemplo], requires_grad=False)

        optimizer.zero_grad()
        y_hat, _ = modeloXOR(x)
        loss_train = loss_function.forward(y_hat, y)
        loss_train.backward()
        optimizer.step()
        train_losses[epoch] = loss_train.item()

      if (epoch + 1) % 1000 == 0:
        print(f"Epoch {epoch+1}/{num_epochs}, Erro Treino: {loss_train.item():.4f}")


num_epochs = 5000
train_losses = np.zeros(num_epochs)
train_xor(modeloXOR,optimizer,loss_function,x_xor,y_xor,num_epochs,train_losses)
```

```
Epoch 1000/5000, Erro Treino: 0.3352
Epoch 2000/5000, Erro Treino: 0.0082
Epoch 3000/5000, Erro Treino: 0.0027
Epoch 4000/5000, Erro Treino: 0.0017
Epoch 5000/5000, Erro Treino: 0.0010
```

Resultados

```
predictions, hidden = modeloXOR(x_xor)
pred = predictions > 0.5
hid = hidden > 0.5
print(x_xor)
print(predictions)
print(hidden)
```

```
tensor([[0., 0.],
        [0., 1.],
        [1., 0.],
        [1., 1.]])
tensor([[0.0358],
        [0.9685],
        [0.9688],
        [0.0332]], grad_fn=<SigmoidBackward0>)
tensor([[0.0018, 0.0662],
        [0.0997, 0.9706],
        [0.0993, 0.9715],
        [0.8740, 0.9999]], grad_fn=<SigmoidBackward0>)
```

```
print("Saídas do modelo: ")
print(pred)

print("Saídas das Camadas Ocultas: ")
print(hid)
```

```
Saídas do modelo:
tensor([[False],
        [ True],
        [ True],
        [False]])
Saídas das Camadas Ocultas:
tensor([[False, False],
        [False,  True],
        [False,  True],
        [ True,  True]])
```

Analisando os resultados acima, podemos ver, pela variável hid, que ilustra os valores dos dois neurônios ocultos, que o neurônio H1 representa uma porta AND (E) e o neurônio H2 representa uma porta OR (OU)

É importante destacar que outras con guraçõ es de portas podem ser obtidas, por exemplo, a porta Not-AND e Not-OR. A condição inicial do modelo de ne as portas que o modelo irá aprender.

Exercício de Apoio

Realizar as seguintes atividades:

1. Treinar o modelo com outras con guração (variar o número de camadas e neurônios
2. Avaliar o processo de treinamento considerando outros valores para a taxa de aprendizagem