UNIVERSIDADE DE SÃO PAULO
Faculdade de Zootecnia e Engenharia de Alimentos
Departamento de Ciências Básicas

Perceptron Em Camadas (MLP) em python

Introdução

Os perceptrons, interligados por meio de conexões conhecidas como sinapses, formam uma rede neural na qual cada perceptron desempenha uma função simples. No entanto, a rede como um todo possui capacidade computacional para resolver problemas complexos. Assim, as redes neurais atuam como aproximadores universais de funções multivariáveis contínuas.

A quantidade de neurônios nas camadas de entrada e saída é influenciada pela dimensionalidade dos dados, enquanto o número de neurônios nas camadas escondidas é determinado pela complexidade do problema. A escolha da estrutura da rede tem um impacto direto na qualidade do modelo resultante.

Nesta prática, abordaremos a arquitetura neural conhecida como Multi-Layer Perceptron (MLP). No entanto, é importante ressaltar que existe uma ampla variedade de arquiteturas e aplicações disponíveis. Entre elas, destacam-se as redes de Hopfield, as Redes Pulsadas, as arquiteturas com base radial, as arquiteturas do tipo ART (Adaptative Resonance Theory), entre outras.

Considere o exemplo do conjunto de dados extraído de flores Iris (Lírio) conforme apresentado por Fisher em 1936. Esta base de dados compila quatro características morfológicas dos lírios, possibilitando a classificação da flor em três espécies: Iris setosa, Iris virginica e Iris versicolor.

A base de dados Iris é composta por 50 amostras de cada uma das três espécies. Para cada amostra, foram realizadas medições do comprimento e largura das sépalas e pétalas, expressas em centímetros.

A tabela 1 contém uma amostragem do conjunto de dados Iris.

Tabela 1

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)		target
5.5	2.4	3.7	1.0	1	Versicolor
5.4	3.7	1.5	0.2	0	Setosa
7.7	2.6	6.9	2.3	2	Virginica
4.9	3.0	1.4	0.2	0	Setosa
5.1	3.8	1.5	0.3	0	Setosa
5.5	2.4	3.7	1.0	1	Versicolor

O objetivo desta aula prática é criar um modelo MLP no ambiente Google Colab, com a capacidade de classificar amostras de Lírio como Setosa, Versicolor ou Virginica, com base nas medições de comprimento e largura de suas sépalas e pétalas.

Roteiro

1. No Google Colab crie um notebook e importe as seguintes bibliotecas:

```
import pandas as pd #Bibliotea para manipulação de frames de dados
import seaborn as sns #Utilizada para plotar os dados
import matplotlib.pyplot as plt #Utilizada para plotar os dados
from sklearn import datasets #Biblioteca para gerar dados de exemplo
from sklearn.neural_network import MLPClassifier #Classificador MLP
from sklearn.model_selection import train_test_split #Serve para dividir o conjunto de dados (Treino e teste)
```

2. Carregue e formate o conjunto de dados Iris

```
# Carregando os dados de exemplo
data = datasets.load_iris() # Carregand os dados 'iris'
iris = pd.DataFrame(data['data'], columns=data.feature_names) #Cria uma tabela (iris) com as entradas (features)
iris['target'] = data.target #Acrescenta as classes à tabela iris
target = iris.pop('target') # Extrai a columa target da tabela iris
```

A biblioteca SKLEARN permite gerar automaticamente conjuntos de dados para testar modelos de IA. Utilizando o comando **datasets.load_iris()**, é possível carregar o conjunto de dados "Iris" de Fisher (1936) na variável denominada "**data**"..

A biblioteca Pandas permite criar estruturas para manipular dados em forma de tabelas. No código acima, a tabela (ou DataFrame) **iris** é preenchida com 4 colunas, uma coluna para cada característica das amostras de lírio.

Através da estrutura de um DataFrame do Pandas, podemos, por exemplo, visualizar os dados das cinco primeiras linhas da tabela "iris" utilizando o comando iris.head(5).

```
iris.head(5)
```

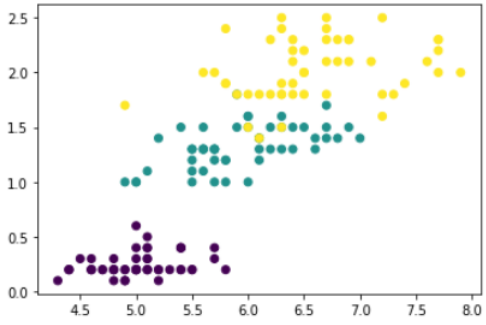
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Do mesmo modo, podem ser utilizados os comandos **iris.tail(5),** para exibir as últimas 5 linhas e **iris.sample(5)** para exibir 5 exemplos do conjunto de dados escolhidos de forma aleatória.

3. Vamos agora plotar um gráfico 2D que nos permita visualizar os dados da tabela **iris**.

```
plt.scatter(iris['sepal length (cm)'],iris['petal width (cm)'], c=target)
```

```
<matplotlib.collections.PathCollection at 0x7fdb15952e90>
```



A biblioteca matplotlib foi utilizada para plotar um gráfico do tipo “scatter” com as características: “sepal length (cm)” e “petal width (cm)”. O vetor “target” foi utilizado como atributo de cor permitindo a visualização das três classes de lírio.

4. O próximo passo consiste em dividir o conjunto de dados em duas partes: **o conjunto de treinamento e o conjunto de teste**. Isso é realizado por meio do comando "**train_test_split**" da biblioteca **sklearn**. Este comando realiza a divisão de maneira aleatória, e o parâmetro "**test_size**" determina a porcentagem de dados alocada para o conjunto de teste, enquanto o restante é atribuído ao conjunto de treinamento.

```
#Divide o conjunto de dados em treino e teste (70% treino e 30% teste)
X_train, X_test, y_train, y_test = train_test_split(iris, target, stratify=target,random_state=1, test_size = 0.3)
```

As variáveis X_train, X_test, y_train e y_test recebem os conjuntos de dados após a divisão.

5. Vamos agora criar a rede MLP que vai servir de modelo classificador utilizando a função **MLPClassifier**, onde o parâmetro “**solver**” é utilizado para configurar a estratégia de treinamento da rede que pode ser: “**lbfgs**” (Limited Memory - BFGS); “**adam**” (adaptative moment estimation); “**sgd**” (Stochastic gradient descent).

```
 # solver='lbfgs', 'sgd', 'adam';
clf = MLPClassifier(solver='adam',alpha=1e-4, random_state=1,early_stopping=False, max_iter=1000) #Cria o modelo classificador MLP
clf.fit(X_train, y_train) #Treina a rede MLP utilizando o conjnto de dados de treino
```

Nesse trecho de código, foi criado o objeto **clf** (classificador) que é o modelo da rede MLP gerada.
A função **clf.fit(X_train, y_train)** executa o treinamento da rede utilizando a parcela de dados reservada para o treinamento.

6. Após o treinamento, vamos verificar a precisão da rede utilizando o conjunto de dados de teste. A função **clf.predict (X_test)** serve para estimar as classes para as características das amostras do conjunto de dados de teste.

```
prediction = clf.predict(X_test) #Predição das classes para o conjunto de dados de teste
print('Precisão: {%}',clf.score(X_test, y_test)*100) #Quantifica a eficiência da rede
```

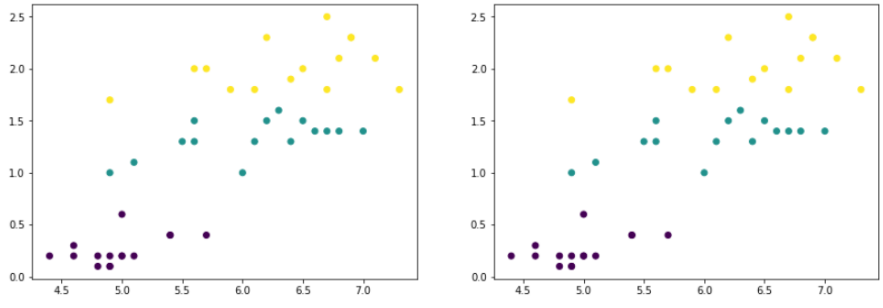
```
Precisão: {%} 100.0
```

A função **clf.score(X_test, y_test)** quantifica em porcentagem a acurácia na predição das classes do conjunto de dados de teste.

7. O próximo código, plota dois gráficos. O da direita mostra os dados do conjunto de teste com as classes extraídas da própria base de dados e o da esquerda mostra os mesmos dados com as classes atribuídas pela rede. As classes são representadas pela cor de cada ponto.

```
fig, axes = plt.subplots(1, 2,figsize=(15, 5)) #Cria uma Figure com duas subplots
axes[0].scatter(X_test['sepal length (cm)'],X_test['petal width (cm)'], c=y_test) #Plota o conjunto de teste com a classe esperada
axes[1].scatter(X_test['sepal length (cm)'],X_test['petal width (cm)'], c=prediction) #Plota o conjunto de teste com a classe atribuida pela rede
```

```
<matplotlib.collections.PathCollection at 0x7f5190e875d0>
```



8. Vamos agora montar uma estrutura que permita introduzir as características de um lírio qualquer para que a rede faça a predição da sua classe.

```
5#////////////////////////////////////////////////
#//   Predição para uma nova entrada            ///
#////////////////////////////////////////////////

sl=input("comprimento da sepala: ")
sw=input("largura da sepala: ")
pl=input("comprimento da petala: ")
pw=input("largura da petala: ")

#Cria uma variável  dicionário com as entradas
amostra = {
  "sepal length (cm)": [sl],
  "sepal width (cm)": [sw],
  "petal length (cm)":[pl],
  "petal width (cm)":[pw]
}

df = pd.DataFrame(amostra) # Cria uma tabela a partir do dicionário
```

O comando **input** serve para possibilitar a entrada de dados pelo usuário. No caso do código acima, o programa vai aguardar digitar as medidas de sépalas e pétalas de uma amostra qualquer e as armazena nas variáveis **sl, sw, pl** e **pw**. A variável "**amostra**" é um tipo especial de estrutura de dados utilizada em python chamada de "dicionário" e serve para vincular cada característica com sua respectiva descrição.
Finalmente, é criado um dataFrame (**df**) pandas baseado no dicionário "**amostra**".

9. Agora podemos utilizar o dataFrame criado, como parâmetro para que a rede atribua uma classe para essas características de lírio:

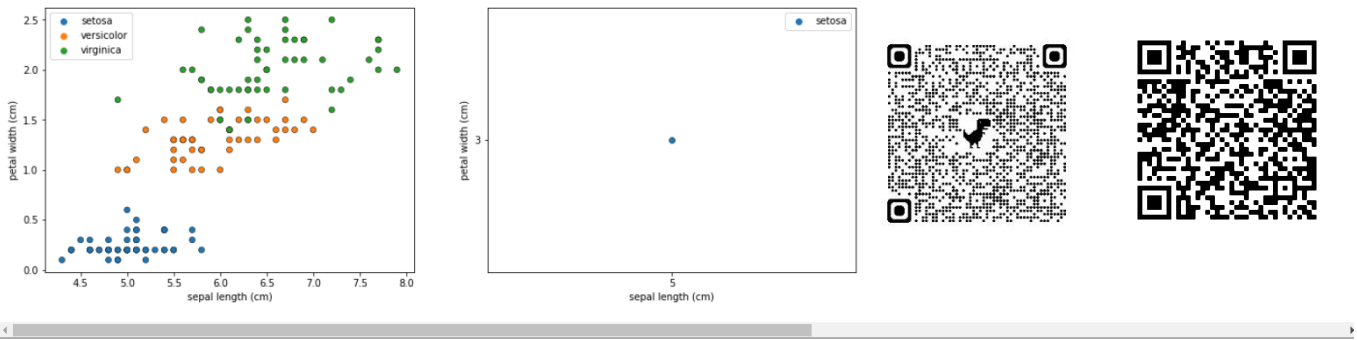
```
prediction = clf.predict(df) # Classifica o novo conjunto de características
print(prediction) #Mostra a classe atribuida
```

```
[0]
```

10. Por último, o próximo código utiliza dois gráficos scatter gerados com suporte da biblioteca "**seaborn**". O da esquerda mostra todos os dados da base de dados iris com suas classes e o da direita mostra a nova amostra com a classe que a rede estimou para suas características.
O código QR direciona para a documentação da biblioteca MLPClassifier.

```
# Exibe a nova entrada classificada
fig, axes2 = plt.subplots(1, 2,figsize=(15, 5)) #Cria uma Figure com duas subplots
sns.scatterplot(iris['sepal length (cm)'],iris['petal width (cm)'],hue=data.target_names[target],alpha=1,edgecolor="black", ax=axes2[0]) #Plota os dados usados para modelar a rede
sns.scatterplot(df['sepal length (cm)'],df['petal width (cm)'],hue=data.target_names[prediction],alpha=1,edgecolor="black", ax=axes2[1]) #plota o novo ponto com sua classe estimada
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional arg
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional arg
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f5190e201d0>
```



*R. A. Fisher (1936). «The use of multiple measurements in taxonomic problems». Annals of Eugenics. **7**: 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x*
https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html
https://www.w3schools.com/python/pandas/pandas_dataframes.asp