# Redes Neurais Artificiais

January 7, 2022

## 1 Exercício 6 - Aplicação das Máquinas de Aprendizado Extremo (ELM)

**Aluno:** Vítor Gabriel Reis Caitité
**Matrícula:** 2021712430

### 1.1 Objetivos

O objetivo dos exercícios desta semana é utilizar as ELMs para resolver problemas multidimensionais, a partir de bases de dados reais.

As bases de dados utilizadas pertencem ao repositório *UCI Machine Learning Repository* [1]. A primeira base de dados a ser estudada é a base Breast Cancer (diagnostic). Para esta base, os dados serão divididos de forma aleatória os dados entre treinamento e teste e comparar as acurácias de treinamento e teste para diferentes valores do hiperparâmetro que controla o número de neurônios. Os valores de acurácia serão apresentados na forma de *media +/- desvio padrão* para 10 execuções diferentes. O mesmo será feito para a base *Statlog (Heart)*.

Além das Extreme Learning Machines, também será utilizado um modelo baseado no perceptron, e assim o desempenho na solução dos dois problemas, poderá ser comparado às ELMs.

### 1.2 Importando Bibliotecas

[43]:
```
# Imports
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
from mpl_toolkits.mplot3d import Axes3D
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
```

### 1.3 Implementação da ELM

Nesse tipo de modelo é feita a escolha de uma matriz de pesos Z aleatória que, através do aumento de dimensão no espaço da camada intermediária, busca garantir a separação linear nesse espaço.

Resumidamente, o intuito da ELM é que o número de funções $g_i(\mathbf{x}, \mathbf{z}_i)$ seja suficientemente grande para garantir a separabilidade no espaço da camada intermediária. Após isso, uma solução direta de erro mínimo pode ser obtida.

Como descrito em [2], a ELM possui um método de aprendizado simples, descrito a seguir.

Dado um *dataset* de treinamento com N amostras $\{(\mathbf{x}_i, \mathbf{y}_i) \backslash \mathbf{x}_i \in \mathbf{R}^n, \mathbf{y}_i \in \mathbf{R}^m, i = 1, ..., N\}$, uma função de ativação $g(x)$ e um número de neurônios da camada escondida P:

- Gerar uma matriz de pesos de entrada Z tal que $Z \in R^{nxP}$.
- Calcular a matriz de mapeamento H (saída da camada escondida), tal que $ H = g(**XZ**) $.
- Calcular os pesos da camada de saída, $\mathbf{W} = \mathbf{H}^{+}\mathbf{Y}$, sendo $\mathbf{H}^{+}$ a pseudo-inversa de $\mathbf{H}$.

[ ]:
```
import random
class ELM:

    def __init__(self, n_neurons):
        self.n_neurons = n_neurons

    def fit(self, X, y):
        # Adding polarization term
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        n = X_new.shape[1]
        self.Z = np.array([random.uniform(-0.5, 0.5) for i in range(n*self.
↪n_neurons)]).reshape(n, self.n_neurons)
        H = np.tanh(np.dot(X_new, self.Z))
        H_new = np.ones((H.shape[0], H.shape[1]+1))
        H_new[:,1:] = H
        self.w = np.dot(np.linalg.pinv(H_new), y)
        return self.w, H, self.Z

    def predict(self, X):
        X_new = np.ones((X.shape[0], X.shape[1]+1))
        X_new[:,1:] = X
        H = np.tanh(np.dot(X_new, self.Z))
        H_new = np.ones((H.shape[0], H.shape[1]+1))
        H_new[:,1:] = H
        y_predicted = np.sign(np.dot(H_new, self.w))
        y_predicted[y_predicted==0]=1
        return y_predicted
```

## 1.4 Implementação do Perceptron Simples

O Perceptron de uma única camada é utilizado para dividir duas classes linearmente separáveis. O funcionamento do Perceptron de camada única é muito simples, as entradas (Xi) representam as informações do processo que desejamos mapear, sendo que cada uma das entradas terá um peso ponderado (Wi) que representa a importância de cada entrada em relação ao valor de saída desejado (y). O resultado da somatória das entradas ponderadas será somado ao limiar

de ativação ($\theta$) e então repassado como argumento da função de ativação g(.), a qual terá como resultado a saída desejada. Normalmente a função de ativação costuma ser do tipo função degrau ou degrau bipolar.

[53]:
```
class Perceptron:

    def __init__(self, learning_rate=0.1, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.activation_func = self._unit_step_func
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0
        y_ = np.array([1 if i > 0 else 0 for i in y])
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_func(linear_output)
                # Perceptron update rule
                update = self.lr * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update
        return (np.concatenate(([self.bias], self.weights)))

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_func(linear_output)
        return y_predicted

    def _unit_step_func(self, x):
        return np.where(x>=0, 1, 0)
```

### 1.5 Funções para Captação de Resultados da ELM e Perceptron

[54]:
```
def resultsELM(X, y, max_iterations, p):
    train_accuracy_ELM = np.zeros(max_iterations)
    test_accuracy_ELM = np.zeros(max_iterations)
    for i in range(0, max_iterations):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        # Normalizing data:
        normalizer = preprocessing.Normalizer()
        X_train = normalizer.fit_transform(X_train)
```

```
        X_test = normalizer.transform(X_test)

        # ELM
        clf = ELM(p)
        clf.fit(X_train, y_train)
        y_hat_train=clf.predict(X_train)
        y_hat=clf.predict(X_test)
        train_accuracy_ELM[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy_ELM[i] = accuracy_score(y_test, y_hat)

    print(f"********* Results ELM (p = {p})**************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy_ELM.mean())+ "+/-" + '{:
↪.4f}'.format(train_accuracy_ELM.std()))
    print("Acc test: " + '{:.4f}'.format(test_accuracy_ELM.mean()) + "+/-" + '{:.
↪4f}'.format(test_accuracy_ELM.std()))
```

[59]:
```
def results_perceptron(X, y, max_iterations):
    train_accuracy = np.zeros(max_iterations)
    test_accuracy = np.zeros(max_iterations)
    for i in range(0, max_iterations):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        # Normalizing data:
        normalizer = preprocessing.Normalizer()
        X_train = normalizer.fit_transform(X_train)
        X_test = normalizer.transform(X_test)

        # Perceptron
        clf = Perceptron(learning_rate=0.1, n_iters=5000)
        clf.fit(X_train, y_train)
        y_hat_train=clf.predict(X_train)
        y_hat=clf.predict(X_test)
        train_accuracy[i] = accuracy_score(y_train, y_hat_train)
        test_accuracy[i] = accuracy_score(y_test, y_hat)

    print(f"********* Results Percetron **************")
    print("Acc train: " + '{:.4f}'.format(train_accuracy.mean())+ "+/-" + '{:.
↪4f}'.format(train_accuracy.std()))
    print("Acc test: " + '{:.4f}'.format(test_accuracy.mean()) + "+/-" + '{:.
↪4f}'.format(test_accuracy.std()))
```

### 1.6 Aplicação da ELM na base Breast Cancer

[61]:
```
wdbc_dataset = pd.read_csv('data/WDBC/wdbc.data', names=list(range(0,32)))
# convert to array


,
y = wdbc_dataset[1].to_numpy()
```

```
X = wdbc_dataset.drop([0, 1],axis='columns').to_numpy()
y[np.where(y=='B')] = 1
y[np.where(y=='M')] = -1
y = np.array(y.tolist())
for p in [5, 10, 30, 50, 100, 200, 300]:
    resultsELM(X, y, 10, p)
```

```
********* Results ELM (p = 5)**************
Acc train: 0.9262+/-0.0089
Acc test: 0.9061+/-0.0278
********* Results ELM (p = 10)**************
Acc train: 0.9479+/-0.0080
Acc test: 0.9368+/-0.0251
********* Results ELM (p = 30)**************
Acc train: 0.9721+/-0.0050
Acc test: 0.9623+/-0.0147
********* Results ELM (p = 50)**************
Acc train: 0.9769+/-0.0034
Acc test: 0.9632+/-0.0110
********* Results ELM (p = 100)**************
Acc train: 0.9846+/-0.0033
Acc test: 0.9447+/-0.0147
********* Results ELM (p = 200)**************
Acc train: 0.9982+/-0.0016
Acc test: 0.9114+/-0.0276
********* Results ELM (p = 300)**************
Acc train: 1.0000+/-0.0000
Acc test: 0.8623+/-0.0380
```

### 1.7 Aplicação do Perceptron na base Breast Cancer

[63]:
```
y[y==-1] = 0
results_perceptron(X, y, 10)
```

```
********* Results Percetron **************
Acc train: 0.9196+/-0.0221
Acc test: 0.9096+/-0.0394
```

### 1.8 Aplicação da ELM na base Statlog (Heart)

[56]:
```
statlog_dataset = pd.read_csv('data/statlog/heart.dat', sep="\s+",␣
 ↪engine='python', header=None)
X = statlog_dataset.drop((13), 1).to_numpy()
y = statlog_dataset.iloc[:, 13].to_numpy()
y[y==2] = -1
for p in [5, 10, 30, 50, 100, 200, 300]:
    resultsELM(X, y, 10, p)
```

```
********* Results ELM (p = 5)**************
Acc train: 0.7528+/-0.0547
Acc test: 0.7426+/-0.0558
********* Results ELM (p = 10)**************
Acc train: 0.8394+/-0.0177
Acc test: 0.8037+/-0.0455
********* Results ELM (p = 30)**************
Acc train: 0.8704+/-0.0142
Acc test: 0.8037+/-0.0333
********* Results ELM (p = 50)**************
Acc train: 0.8861+/-0.0126
Acc test: 0.7815+/-0.0502
********* Results ELM (p = 100)**************
Acc train: 0.9384+/-0.0088
Acc test: 0.7315+/-0.0525
********* Results ELM (p = 200)**************
Acc train: 1.0000+/-0.0000
Acc test: 0.6167+/-0.0469
********* Results ELM (p = 300)**************
Acc train: 1.0000+/-0.0000
Acc test: 0.5852+/-0.0484
```

### 1.9 Aplicação do Perceptron na base Statlog (Heart)

[60]:
```
y[y==-1] = 0
results_perceptron(X, y, 10)
```

```
********* Results Percetron **************
Acc train: 0.7287+/-0.1171
Acc test: 0.7148+/-0.1281
```

### 1.10 Perguntas:

1) Com quantos neurônios (aproximadamente) a acurácia de teste aparenta ser máxima?

A acurácia máxima de teste para a base de dados *Breast Cancer* foi atingida para a ELM com 50 neurônios na camada escondida (0.9632+/-0.0110). Já para a base de dados *Statlog (Heart)*, essa acurácia máxima foi de 0.8037+/-0.0333, e foi obtida para as redes com 10 e 30 neurônios na camada escondida.

2) O que acontece com os valores de acurácia de treinamento e teste conforme aumentamos progressivamente o número de neurônios (por exemplo, para 5, 10, 30, 50, 100, 300 neurônios)?

Percebeu-se que a acurácia de treinamento aumenta progressivamente de acordo com o aumento do número de neurônios. Contudo esse comportamente não é o mesmo no caso da acurácia de teste. Essa aumenta até certo momento de acordo com o número de neurônios, porém a partir de um certo ponto percebe-se que o modelo sofre *overfitting*, ocasionando uma menor acurácia de teste.

## 1.11 Discussão Perceptron x ELM

Pôde-se perceber que a ELM obteve resultados superiores ao Perceptron Simples. A principal causa disso se deve ao fato de as bases de dados não serem linearmente separáveis. Dessa forma um modelo como a ELM (capaz de lidar com bases não linearmente separáveis) tende a obter resultados superiores.

## 1.12 Referências

[1] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[2] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," Neurocomputing, vol. 70, no. 1-3, pp. 489–501, 2006.