Redes Neurais Artificiais - IFES - PPCOMP

Exercicio 05

Arquitetura capaz de resolver um problema XOR (forward)

In [45]:

```
import sklearn
import numpy as np

from sklearn.base import BaseEstimator,ClassifierMixin

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [46]:

```
print('Versão do scikit-learn {}.'.format(sklearn.__version__))
```

```
Versão do scikit-learn 0.21.2.
```

In [47]:

```
# Combinações de entradas - XOR
X = np.array([
    [0, 1],
    [1, 0],
    [1, 1],
    [0, 0]
])
```

In [48]:

```
# Labels (y) - XOR
y = np.array([1,1,0,0])
```

In [49]:

```
X.shape
```

Out[49]:

```
(4, 2)
```

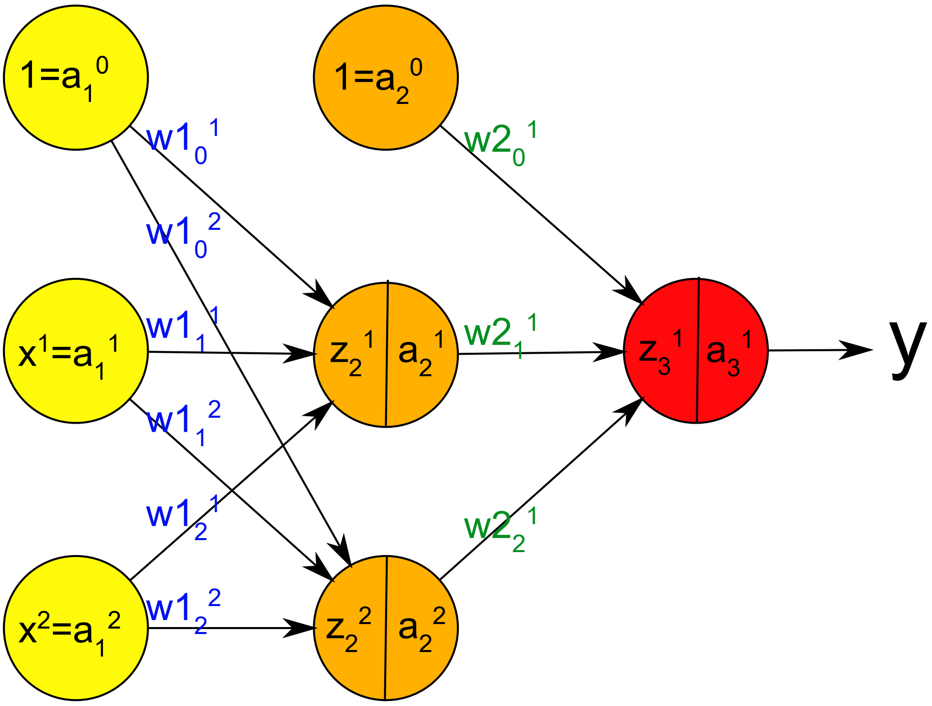
In [50]:

```
y.shape
```

Out[50]:

```
(4,)
```

Arquitetura de Referência



In [51]:

```
#funcoes de ativação

def sigmoid(x):
    return 1 / (1 + np.exp(-1* x))

def sigmoid_deriv(x):
    return x * (1 - x)

def tanh(x):
    return (1.0 - np.exp(-2*x))/(1.0 + np.exp(-2*x))

def tanh_deriv(x):
    return (1 + x)*(1 - x)
```

In [52]:

```
class SolverXorForward(BaseEstimator, ClassifierMixin):

    def __init__(self):

        # Pesos - inicialização randomica (serão usados na implementação da fase bacward)
        #self.W1 = np.random.uniform(-1, 1, (2, 2)) # 2x2 - entrada x escondida
        #self.W2 = np.random.uniform(-1, 1, (1, 2)) # 1x2 - escondida x saida

        # Bias associados - inicialização randomica (serão usados na implementação da fase
        #self.W1bias = np.random.random((2, 1)) # 2x1
        #self.W2bias = np.random.random((1, 1)) # 1x1

        # Pesos manuais
        self.W1=np.array([[-7.98691197,  7.80228261],
                          [ 7.24071749, -7.55675496]])
        self.W2=np.array([[14.71044493, 14.81647932]])

        # Bias manuais
        self.W1bias=np.array([[-4.22584175],[-3.906862  ]])
        self.W2bias=np.array([[-7.27388762]])
        return

    def predict_1(self, x):
        a1 = x.reshape(x.shape[0], 1)
        z2 = self.W1.dot(a1) + self.W1bias
        a2 = sigmoid(z2)
        z3 = self.W2.dot(a2) + self.W2bias
        a3 = sigmoid(z3)
        return a3[0]

    def predict(self,X):
        Y = np.array([]).reshape(0, 1)
        for x in X:
            y = np.array([self.predict_1(x)])
            Y = np.vstack((Y,y))
        return Y

    def fit(self, X, y):

        for j in range(X.shape[0]):
            # Implementação do Forward
            a1 = X[j].reshape(X[j].shape[0], 1)
            z2 = self.W1.dot(a1) + self.W1bias
            a2 = sigmoid(z2)
            z3 = self.W2.dot(a2) + self.W2bias
            a3 = sigmoid(z3)

        return self
```

In [53]:

```
clf = SolverXorForward()
clf.fit(X,y)
```

Out[53]:

```
SolverXorForward()
```

In [54]:

```
print("Predição com fase Forward")
for x in X:
    print(x, clf.predict_1(x))
```

```
Predição com fase Forward
[0 1] [0.99912147]
[1 0] [0.99911811]
[1 1] [0.00102371]
[0 0] [0.00114635]
```

In [55]:

```
clf.predict(X)
```

Out[55]:

```
array([[0.99912147],
       [0.99911811],
       [0.00102371],
       [0.00114635]])
```

In [56]:

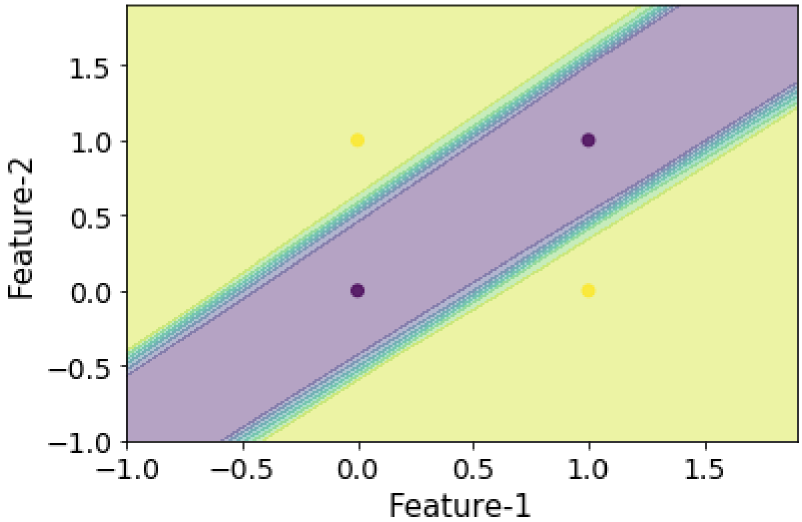
```
#Plot de Fronteira de decisão para o ELM implementado
#Referência: https://towardsdatascience.com/easily-visualize-scikit-learn-models-decision-b

def plot_decision_boundaries(X, y, model_class, **model_params):
    try:
        X = np.array(X)
        y = np.array(y).flatten()
    except:
        print("Coercing input data to NumPy arrays failed")
    reduced_data = X[:, :2]
    model = model_class(**model_params)
    model.fit(reduced_data, y)
    h = .02     # point in the mesh [x_min, m_max]x[y_min, y_max].
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8)
    plt.xlabel("Feature-1",fontsize=15)
    plt.ylabel("Feature-2",fontsize=15)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    return plt
```

In [57]:

```
plot_decision_boundaries(X, y, SolverXorForward)
```

Out[57]:

```
<module 'matplotlib.pyplot' from 'C:\\Users\\leandro\\Anaconda3\\lib\\site-p
ackages\\matplotlib\\pyplot.py'>
```



In [ ]:

```

```