Nome : Arthur Felipe Reis Souza.
Data : 20/03/2024

Parte 1

O exercício 3 consiste em modelar um sistema por meio do modelo neural Adaline (Adaptive Linear Neuron). O Adaline é um modelo de neurônio artificial comumente utilizado em tarefas de regressão. É um modelo linear, ou seja, aprenderá apenas relações lineares entre entrada e saída. Na primeira parte, temos um conjunto de dados de entrada, aparentemente em forma senoidal, e uma função que altera as características dos mesmos. A ideia é usar o modelo neural Adaline para aprender as relações das entradas e saídas, de modo a ajustar os parâmetros do modelo e generalizar a função geradora.

A imagem abaixo ilustra o código em python do treinamento do modelo Adaline.

[14]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

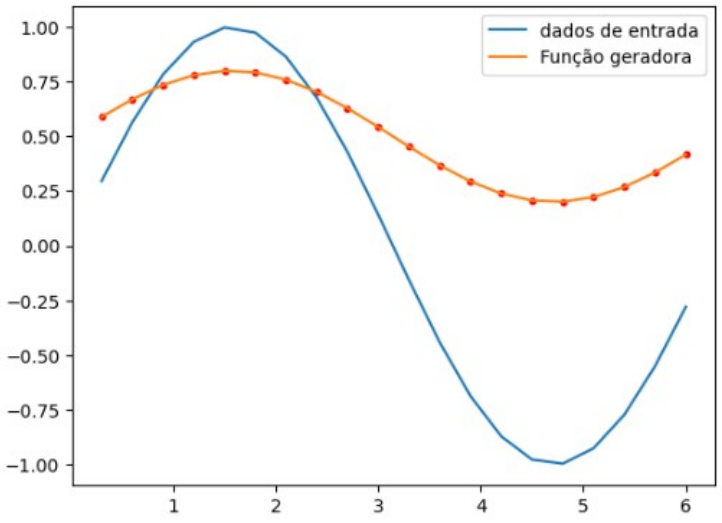
[15]:
```python
def train_adaline(x_inputs : np.array, yd : np.array, learning_rate : float, tol : float, maxepochs : int, control_var : bool):
    dimentions = list(x_inputs.shape)
    try:
        N_inputs = dimentions[0]
        n_val_inputs = dimentions[1]
    except Exception as error:
        print(f" The error {error} is happening. It's happening because n_val_inputs has 0 columns.")
        print(f" So, we will change it to 1")
        print(f"Changing ...")
        n_val_inputs = 1
        print(f"Now, n_val_inputs is {n_val_inputs}")
    finally:
        if control_var == True:
            w = np.random.uniform(size = n_val_inputs + 1) - 0.5
            # Estou organizando minha entrada x em colunas, e adicionando 1 coluna extra de 1s.
            aux = np.column_stack([np.ones_like(x_inputs)])
            x_inputs = np.column_stack([x_inputs, aux])
        else:
            w = np.random.uniform(size = n_val_inputs) - 0.5

        n_epochs = 0 # É o número de vezes que estou treinando usando TODOS os dados de entrada.
        erro_epoch = tol + 1
        lst_errors_grad = np.zeros((maxepochs))

        # Loop while que resultará no treino do meu modelo.
        while ((n_epochs < maxepochs) and (erro_epoch > tol)):
            erro_grad = 0

            # Alterando a ordem de dados de treino, no fito de o gradiente descendente não ficar estático em 1 lugar específico.
            change_order_train = np.random.permutation(N_inputs)
            for i in range(N_inputs):
                i_rand = change_order_train[i]
                x_val_train = x_inputs[i_rand, : ]
                # Não é necessário fazer o np.transpose(), pois já fiz implicitamente através do column_stacks.
                y_hat = np.dot(x_val_train, w) #ŷ = ([Xt] @ w)
                err = (yd[i_rand] - y_hat)
                dw = (learning_rate*err* x_inputs[i_rand, :])
                w = w + dw
                erro_grad = erro_grad + (err * err)

            lst_errors_grad[n_epochs] = erro_grad / N_inputs
            n_epochs += 1
        return (w, lst_errors_grad)
```

Pode-se observar que o treinamento do modelo consistiu em ajustar os parâmetros seguindo a regra delta, dada pela equação $w(t+1)=w(t)+ei(t)*\eta*xi(t)$. Portanto, a cada iteração de entrada, os parâmetros são ajustados e a saída do modelo se aproxima cada vez mais da saída da função geradora.

A imagem abaixo destaca a geração dos dados que foram utilizados no treinamento do modelo.

```
[25]: t = pd.read_csv('Ex1_t', delimiter = ' ')
      x = pd.read_csv('Ex1_x', delimiter = ' ')
      y = pd.read_csv('Ex1_y', delimiter = ' ')
      t = t['x']
      x = x['x']
      y = y['x']
      plt.plot(t, x, label = 'dados de entrada')
      plt.scatter(t, y, color = 'red', s = 10)
      plt.plot(t, y, label = 'Função geradora')
      plt.legend()
```

```
[25]: <matplotlib.legend.Legend at 0x7f700756c790>
```



Os pontos em vermelho são os pontos gerados para treinamento. Portanto, iremos aproximar a função que gerou esses dados.

Treinamento do Adaline :

```
w, lst_err_grad = train_adaline(x, y, learning_rate = 0.01, tol = 0.01, maxepochs = 10, control_var = True)
```

Analisando os parâmetros de treinamento e os hiperparâmetros do modelo :

Learning Rate : É um hiperparâmetro que utilizamos no processo de otimização ao treinar uma rede neural. Esse hiperparâmetro é usado para controlar a taxa de aprendizado do modelo, ou seja, ele controla a velocidade o qual um modelo pode ou não convergir para uma solução ótima. Um learning rate alto pode significar uma rápida convergência, mas também há a possibilidade de passar da solução ótima e divergir. Um baixo learning rate garante a convergência do modelo, porém é um processo demorado. Essa demora implica um alto custo computacional e energético.

Tol : Esse hiperparâmetro serve para garantir que o modelo não pare em pontos de mínimo local. Ele é utilizado como critério de saída do loop while que é utilizado para o treinamento do Adaline.

Max Epochs : É o máximo de vezes que o modelo irá treinar sobre um mesmo conjunto de entradas completo. A cada treinamento completo em 1 conjunto de dados de entrada, irei somar 1 ao valor da variável n_epochs.

Control Var : É uma variável de controle que indicará se terá ou não uma coluna extra na matriz de entrada. Essa coluna extra resultará, no resultado final, na adição de um parâmetro w0.

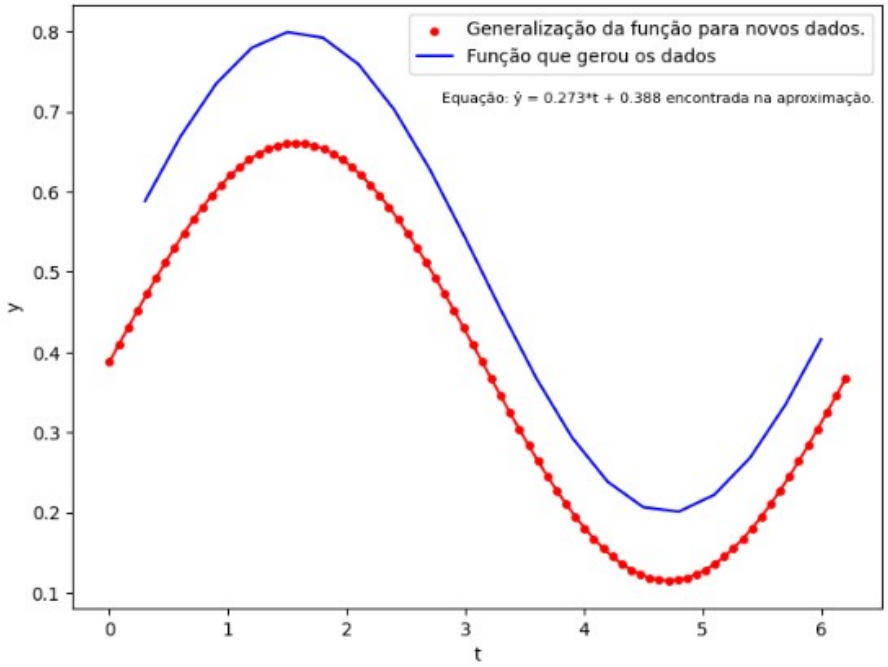
Após treinar o modelo, com diferentes valores de entradas, o testamos, e obtemos aproximações que podem ser visualizadas no gráfico abaixo :

[5]:
```
t_test = np.arange(start = 0, stop = 2*np.pi, step = 0.025*np.pi)
x_test =  np.sin(t_test)
x_test = np.column_stack([x_test, np.ones_like(x_test)])
y_hat = x_test @ w
w
```

[5]:
```
array([0.27283821, 0.38810889])
```

[8]:
```
plt.figure(figsize = (8, 6))
plt.scatter(t_test, y_hat, color = 'red', s = 15, label = 'Generalização da função para novos dados.')
plt.plot(t_test, y_hat, color = 'red')
plt.plot(t, y, color = 'blue', label = 'Função que gerou os dados')
plt.xlabel('t')
plt.ylabel('y')

# Adjust text coordinates to be within the plot area
plt.text(min(t_test) + 2.8, max(y_hat) + 0.05, f'Equação: ŷ = {np.round(w[0],3)}*t + {np.round(w[1],3)} encontrada na aproximação.', fontsi

plt.legend()
```

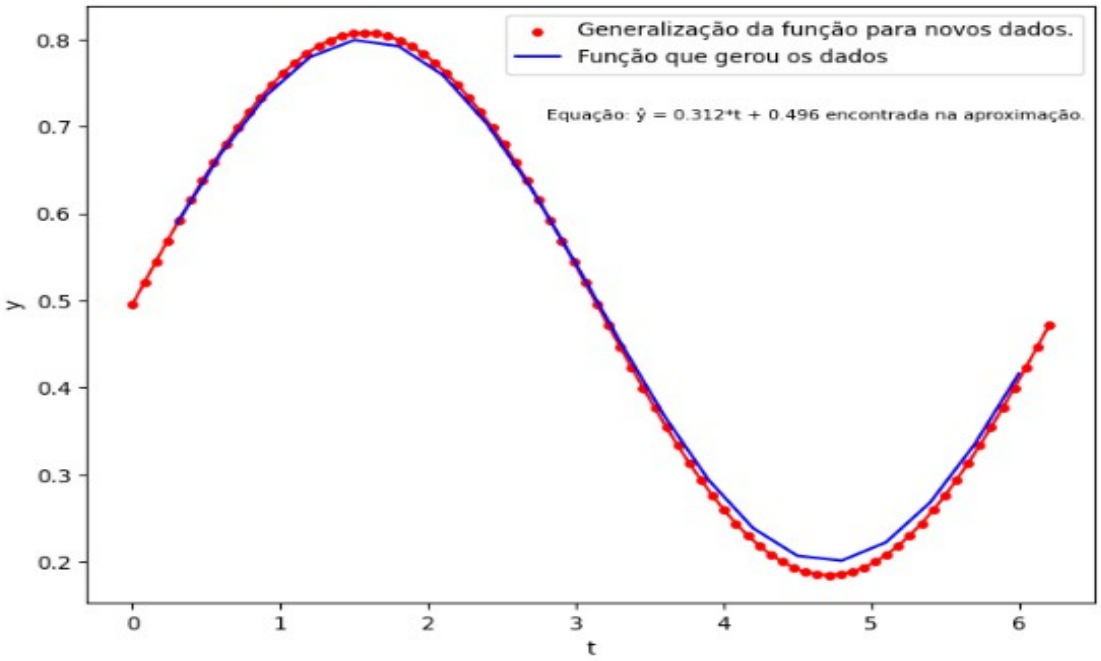
[8]:
```
<matplotlib.legend.Legend at 0x7f78ffad40a0>
```



É notório que a generalização do modelo (função vermelho) se aproxima da função geradora dos dados, exceto por um offset de aproximadamente 0.2 unidade no eixo y. No intuíto de ter uma resposta mais semelhante ao modelo, foi alterado o numero máximo de epocas, de 10 para 30 epocas. Isso implica em um maior tempo de treino e uma possível melhor generalização da função geradora.

A imagem abaixo mostra a nova generalização do modelo, após alterar a qualidade do treino :

```
: <matplotlib.legend.Legend at 0x7ff69b08f1c0>
```



Observando o resultado, pode-se afirmar que os resultados se assemelham muito e é visualmente uma boa aproximação para os dados de treino. Mas um ponto a se observar é que é válido utilizar novas entradas de teste para testar a generalização do modelo e verificar se o mesmo não está superajustado aos dados de entrada (overfitting).

Parte 2

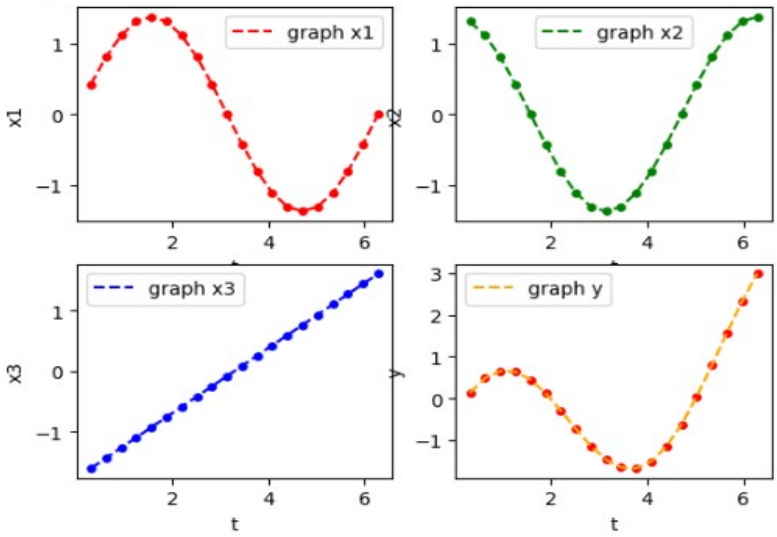
Na parte 2, deseja-se que utilize o modelo Adaline para fazer a aproximação de uma combinação linear de diversas funções não lineares de entrada. Para a extração e manipulação de dados, foi utilizada a biblioteca pandas, que é muito útil no campo de Ciência de Dados e Aprendizado de Máquina. Os dados utilizados no treinamento foram obtidos pelo arquivo disponilizado no moodle. A variável x contém os dados de treinamento organizados em colunas.

```
t = np.array(pd.read_csv('t', delimiter = ' '))
x = pd.read_csv('x', delimiter = ' ')
y = np.array(pd.read_csv('y', delimiter = ' '))
x1 = np.array(x['V1'])
x2 = np.array(x['V2'])
x3 = np.array(x['V3'])
x = np.column_stack([x1, x2, x3])
y_plot = x1 + x2 + x3
```

Utilizando as variáveis criadas acima, iremos representar os gráficos de entrada e uma possível representação de como será a saída do modelo. Portanto, os gráficos abaixo representam os dados e a possível representação da saída do modelo :

[174]:
```
fig, axs = plt.subplots(2, 2)
axs[0, 0].scatter(t, x1, color = 'red', s = 15)
axs[0, 0].plot(t, x1, color = 'red', linestyle = '--', label = 'graph x1')
axs[0, 0].set_xlabel('t')
axs[0, 0].set_ylabel('x1')
axs[0, 0].legend()
axs[0, 1].scatter(t, x2, color = 'green', s = 15)
axs[0, 1].plot(t, x2, color = 'green', linestyle = '--', label = 'graph x2')
axs[0, 1].set_xlabel('t')
axs[0, 1].set_ylabel('x2')
axs[0, 1].legend()
axs[1, 0].scatter(t, x3, color = 'blue', s = 15)
axs[1, 0].plot(t, x3, color = 'blue', linestyle = '--', label = 'graph x3')
axs[1, 0].set_xlabel('t')
axs[1, 0].set_ylabel('x3')
axs[1, 0].legend()
axs[1, 1].scatter(t, y_plot, color = 'red', s = 15)
axs[1, 1].plot(t, y_plot, color = 'orange', linestyle = '--', label = 'graph y')
axs[1, 1].set_xlabel('t')
axs[1, 1].set_ylabel('y')
axs[1, 1].legend()
```

[174]:
```
<matplotlib.legend.Legend at 0x7f2a1038f070>
```



É importante citar que os coeficientes da saída do modelo são os parâmetros que iremos descobrir e, portanto, o último gráfico é apenas uma aproximação do que será a generalização do modelo.

De forma igual à primeira parte, treinamos o modelo a partir dos dados de entrada disponibilizados no moodle :

```
w, lst_err_grad = train_adaline(x, y, learning_rate = 0.001, tol = 0.01, maxepochs = 30, control_var = True)
```

Nesse treinamento, o número máximo de épocas é 30, e o learning rate contém 1 valor de 0.001, que é considerado 1 valor baixo. É necessário tomar cuidado para não colocar o learning rate baixo e um grande número de épocas, pois isso implicará em uma grande demora no treinamento e uma possível divergência.

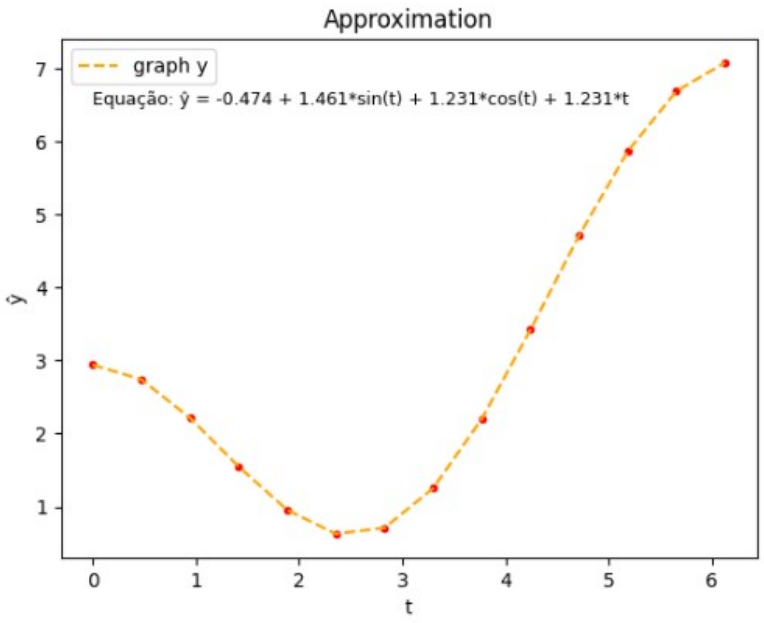
Após treinar o modelo, dados de teste foram gerados e utilizados para testar a eficiência do mesmo. A imagem abaixo mostra a saída y generalizada aos dados de entrada.

```
t_test = np.arange(start = 0, stop = 2*np.pi, step = 0.15*np.pi)
x1 = 1.37*np.sin(t_test)
x2 = 1.37*np.cos(t_test)
x3 = 0.5383*t_test
x_final = np.column_stack([x1, x2, x3, np.ones_like(x1)])
y_hat = x_final @ w
w
```

```
array([-0.47382252,  1.46056024,  1.23071306,  0.94173859])
```

```
plt.scatter(t_test, y_hat, color = 'red', s = 10)
plt.plot(t_test, y_hat, color = 'orange', linestyle = '--', label ='graph y')
plt.xlabel('t')
plt.ylabel('ŷ')
plt.title("Approximation")
plt.text(0, 6.5, f'Equação: ŷ = {np.round(w[0],3)} + {np.round(w[1],3)}*sin(t) + {np.round(w[2],3)}*cos(t) + {np.round(w[2],3)}*t', fontsize
plt.legend()
```

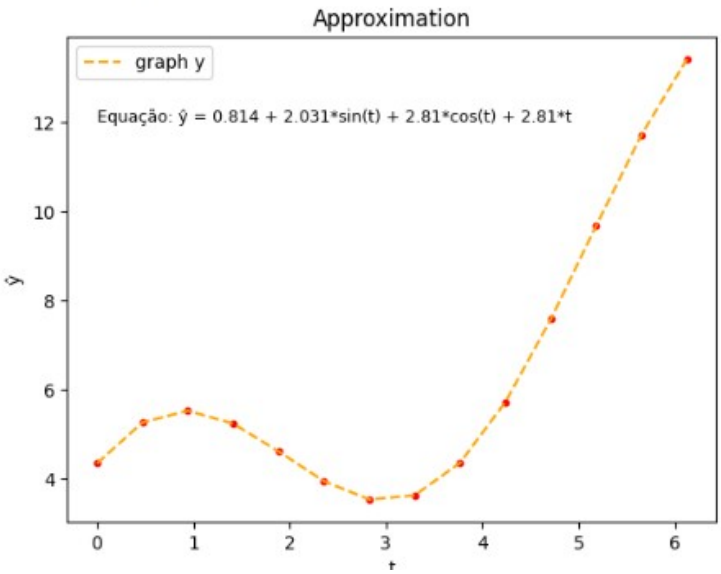
```
<matplotlib.legend.Legend at 0x7f2a0f39bd90>
```



Comparando o gráfico abaixo com o o possível gráfico da função geradora, é possível afirmar que, apenas de algumas diferenças, os gráficos são bem semelhantes e o modelo aparentemente generalizou bem a função geradora.

No fito de testar o modelo neural Adaline, m novo treinamento foi realizado, agora com um número máximo de épocas 50, e um learning rate de 0.01. O resultado obtido foi :

```
[23]: <matplotlib.legend.Legend at 0x7fa201f172e0>
```



O gráfico acima se assemelha ainda mais ao gráfico da função geradora.