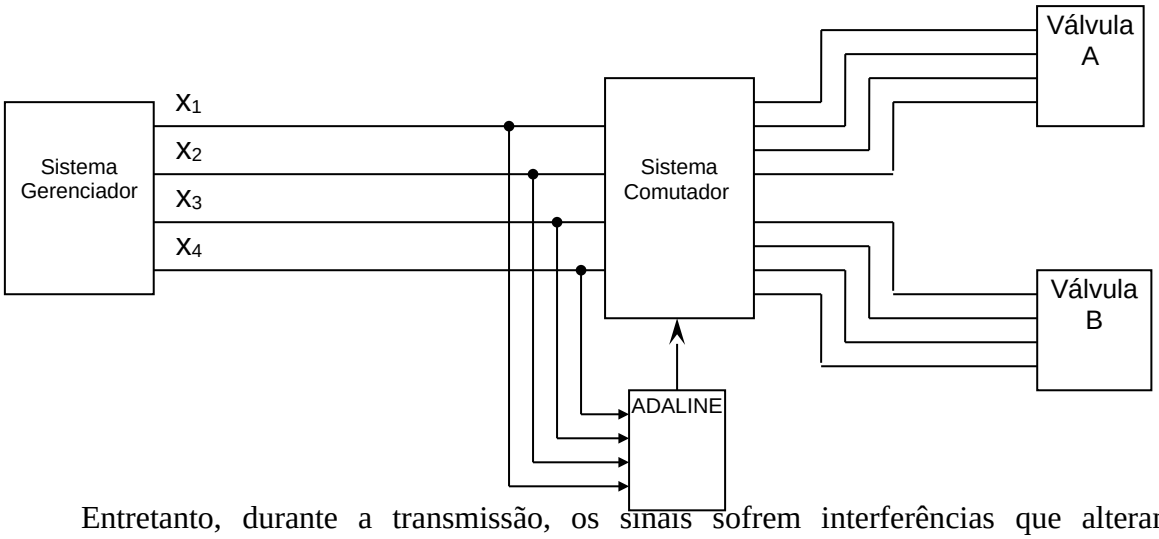
EXERCÍCIO ADALINE - VÁLVULAS:

Implementar a solução e responder as questões e forma de relatório. Utilizar Python no Google Colab para a resolução. Deve-se apresentar o código em anexo:

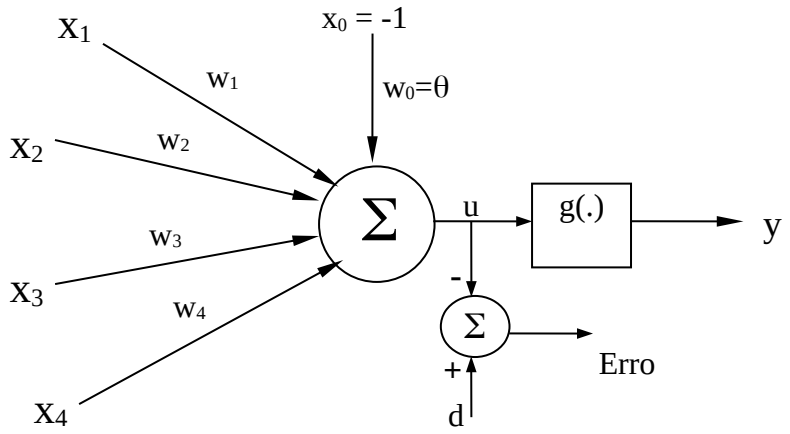
Descrição do Problema:

Um sistema de gerenciamento automático de controle de duas válvulas, situado a 500 metros de um processo industrial, envia um sinal codificado constituído de quatro grandezas {x_1, x_2, x_3 e x_4} que são necessárias para o ajuste de cada uma das válvulas. Conforme mostra a figura abaixo, a mesma via de comunicação é utilizada para acionamento de ambas as válvulas, sendo que o comutador localizado próximo das válvulas deve decidir se o sinal é para a válvula A ou B.



Entretanto, durante a transmissão, os sinais sofrem interferências que alteram o conteúdo das informações transmitidas. Para resolver este problema, a equipe de engenheiros e cientistas pretende treinar uma rede ADALINE para classificar os sinais ruidosos, confirmando ao sistema comutador se os dados devem ser encaminhados para o comando de ajuste da válvula A ou B.

Assim, baseado nas medições dos sinais já com ruídos, formou-se o conjunto de treinamento em anexo, tomando por convenção o valor –1 para os sinais que devem ser encaminhados para o ajuste da válvula A e o valor +1 se os mesmos devem ser enviados para a válvula B. Assim, a estrutura do ADALINE é mostrada na figura seguinte.



Utilizando o algoritmo de treinamento da Regra Delta para classificação de padrões no ADALINE, realize as seguintes atividades:

1. Execute 1 treinamento para a rede ADALINE inicializando o vetor de pesos em cada treinamento com valores aleatórios entre zero e um. Utilize taxa de aprendizado η = 0.001, épocas = 5000 e precisão ε = 0.000001.

```
rede = Adaline(eta = 0.001, epoch = 5000, goal = 0.000001)
rede.fit(X_train, y_train)
```
[62] ✓ 0.2s

```
<__main__.Adaline at 0x7e20191c00d0>
```

```
# Plotando o gráfico da descida do gradiente no processo de tr
plt.figure(figsize=(14,5))
ax = plt.subplot()
ax.plot(range(len(rede.mse_)), rede.mse_)
ax.set_ylabel('Erro Quadrático Médio')
ax.set_xlabel('Época')
ax.set_title('Gráfico de Erro no Treinamento da Rede')
plt.show()
```
[63] ✓ 0.3s

2. Registre os resultados do treinamento acima na tabela abaixo:

Treinamento	Vetor de Pesos Final					Número de Épocas
	Limiar	w_1	w_2	w_3	w_4	
T1	0.309	2.445	1.759	-0.662	-3.611	5000

```
# Testando a rede em lote de amostras
print("\nClassificações dos testes com amostras não apresentadas no treinamento:")

# For entrada, saida in zip(X_test, y_test):
nTests = len(y_test)
for i in range(nTests):
    rede.predict(X_test[i,:], y_t
```
(variable) errorsTests: int

```
print("Acurácia: ", (nTests-rede.errorsTests)/nTests*100, "%")
print("Quantidade de erros no teste em lote: ", rede.errorsTests)
print("Número de épocas do treinamento da RNA: ", rede.epochs)
print("Erro quadrático médio final (MSE - Eqm(w)): ", rede.mse)
print("Erro quadrático final (QE - E(w) - Custo): ", rede.cost)
print("Vetor de pesos finais da RNA treinada - Limiar = ", rede.weight_[0], "Pesos das entradas = ", rede.weight_[1:])
```
✓ 0.0s

```
Classificações dos testes com amostras não apresentadas no treinamento:
Acurácia:  66.66666666666666 %
Quantidade de erros no teste em lote:  5
Número de épocas do treinamento da RNA:  5000
Erro quadrático médio final (MSE - Eqm(w)):  0.580293541472354
Erro quadrático final (QE - E(w) - Custo):  10.155136975766196
Vetor de pesos finais da RNA treinada - Limiar =  0.3097888486601299 Pesos das entradas =  [ 2.44548896  1.75942841 -0.66207277 -3.6117866 ]
```

```
0.309
```

3. Para todos os treinamentos realizados, aplique então a rede ADALINE para classificar e indicar ao comutador se os sinais seguintes devem ser encaminhados para a válvula A ou B.

Amostra	x_1	x_2	x_3	x_4	d	y (T1)
1	0.9694	0.6909	0.4334	3.4965	-1	Acerto
2	0.5427	1.3832	0.6390	4.0352	-1	Acerto
3	0.6081	-0.9196	0.5925	0.1016	1	Erro
4	-0.1618	0.4694	0.2030	3.0117	-1	Acerto
5	0.1870	-0.2578	0.6124	1.7749	-1	Acerto
6	0.4891	-0.5276	0.4378	0.6439	1	Erro
7	0.3777	2.0149	0.7423	3.3932	1	Erro
8	1.1498	-0.4067	0.2469	1.5866	1	Erro
9	0.9325	1.0950	1.0359	3.3591	1	Erro
10	0.5060	1.3317	0.9222	3.7174	-1	Erro
11	0.0497	-2.0656	0.6124	-0.6585	-1	Acerto
12	0.4004	3.5369	0.9766	5.3532	1	Erro
13	-0.1874	1.3343	0.5374	3.2189	-1	Acerto
14	0.5060	1.3317	0.9222	3.7174	-1	Acerto
15	1.6375	-0.7911	0.7537	0.5515	1	Acerto

```
def Inference(row):
    base = np.array([row['x1'], row['x2'], row['x3'], row['x4']])
    return rede.predict(base, row['d'])
```

[75] ✓ 0.0s

```
base_test['T1'] = base_test[['x1', 'x2', 'x3', 'x4', 'd']].apply(Inference, axis=1)
```

[78] ✓ 0.0s

```
base_test
```

[79] ✓ 0.0s

	Amostra	x1	x2	x3	x4	d	y	T1
0	1	0.9694	0.6909	0.4334	3.4965	-1	T1	Acerto
1	2	0.5427	1.3832	0.6390	4.0352	-1	T1	Acerto
2	3	0.6081	-0.9196	0.5925	0.1016	1	T2	Erro
3	4	-0.1618	0.4694	0.2030	3.0117	-1	T1	Acerto
4	5	0.1870	-0.2578	0.6124	1.7749	-1	T1	Acerto
5	6	0.4891	-0.5276	0.4378	0.6439	1	T2	Erro
6	7	0.3777	2.0149	0.7423	3.3932	1	T2	Erro
7	8	1.1498	-0.4067	0.2469	1.5866	1	T2	Erro
8	9	0.9325	1.0950	1.0359	3.3591	1	T2	Erro
9	10	0.5060	1.3317	0.9222	3.7174	1	T1	Erro
10	11	0.0497	-2.0656	0.6124	-0.6585	-1	T1	Acerto
11	12	0.4004	3.5369	0.9766	5.3532	1	T2	Erro
12	13	-0.1874	1.3343	0.5374	3.2189	-1	T1	Acerto
13	14	0.5060	1.3317	0.9222	3.7174	-1	T1	Acerto
14	15	1.6375	-0.7911	0.7537	0.5515	1	T2	Acerto

[ ]

ANEXO – Conjunto de Treinamento.

Amostra	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{x_4}$	**d**
01	0.4329	-1.3719	0.7022	-0.8535	1.0000
02	0.3024	0.2286	0.8630	2.7909	-1.0000
03	0.1349	-0.6445	1.0530	0.5687	-1.0000
04	0.3374	-1.7163	0.3670	-0.6283	-1.0000
05	1.1434	-0.0485	0.6637	1.2606	1.0000
06	1.3749	-0.5071	0.4464	1.3009	1.0000
07	0.7221	-0.7587	0.7681	-0.5592	1.0000
08	0.4403	-0.8072	0.5154	-0.3129	1.0000
09	-0.5231	0.3548	0.2538	1.5776	-1.0000
10	0.3255	-2.0000	0.7112	-1.1209	1.0000
11	0.5824	1.3915	-0.2291	4.1735	-1.0000
12	0.1340	0.6081	0.4450	3.2230	-1.0000
13	0.1480	-0.2988	0.4778	0.8649	1.0000
14	0.7359	0.1869	-0.0872	2.3584	1.0000
15	0.7115	-1.1469	0.3394	0.9573	-1.0000
16	0.8251	-1.2840	0.8452	1.2382	-1.0000
17	0.1569	0.3712	0.8825	1.7633	1.0000
18	0.0033	0.6835	0.5389	2.8249	-1.0000
19	0.4243	0.8313	0.2634	3.5855	-1.0000
20	1.0490	0.1326	0.9138	1.9792	1.0000
21	1.4276	0.5331	-0.0145	3.7286	1.0000
22	0.5971	1.4865	0.2904	4.6069	-1.0000
23	0.8475	2.1479	0.3179	5.8235	-1.0000
24	1.3967	-0.4171	0.6443	1.3927	1.0000
25	0.0044	1.5378	0.6099	4.7755	-1.0000
26	0.2201	-0.5668	0.0515	0.7829	1.0000
27	0.6300	-1.2480	0.8591	0.8093	-1.0000
28	-0.2479	0.8960	0.0547	1.7381	1.0000
29	-0.3088	-0.0929	0.8659	1.5483	-1.0000
30	-0.5180	1.4974	0.5453	2.3993	1.0000
31	0.6833	0.8266	0.0829	2.8864	1.0000
32	0.4353	-1.4066	0.4207	-0.4879	1.0000
33	-0.1069	-3.2329	0.1856	-2.4572	-1.0000
34	0.4662	0.6261	0.7304	3.4370	-1.0000
35	0.8298	-1.4089	0.3119	1.3235	-1.0000