Redes Neuronales Artificiales

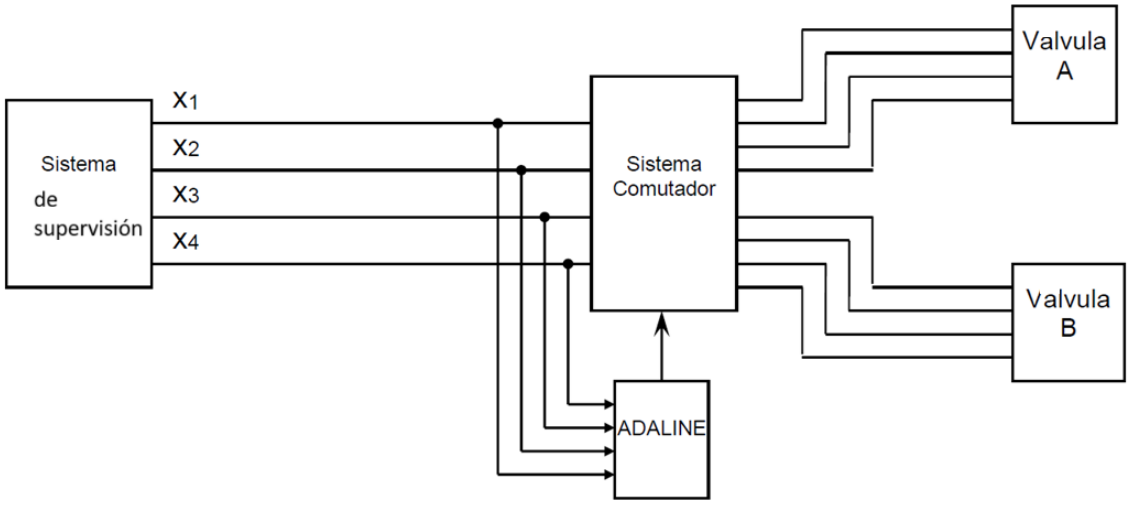
María Alejandra Bonilla Díaz - 20251595002

Youssef Alejandro Ortiz Vargas – 20251595004

Álvaro Alejandro Zarabanda Gutiérrez - 20251595006

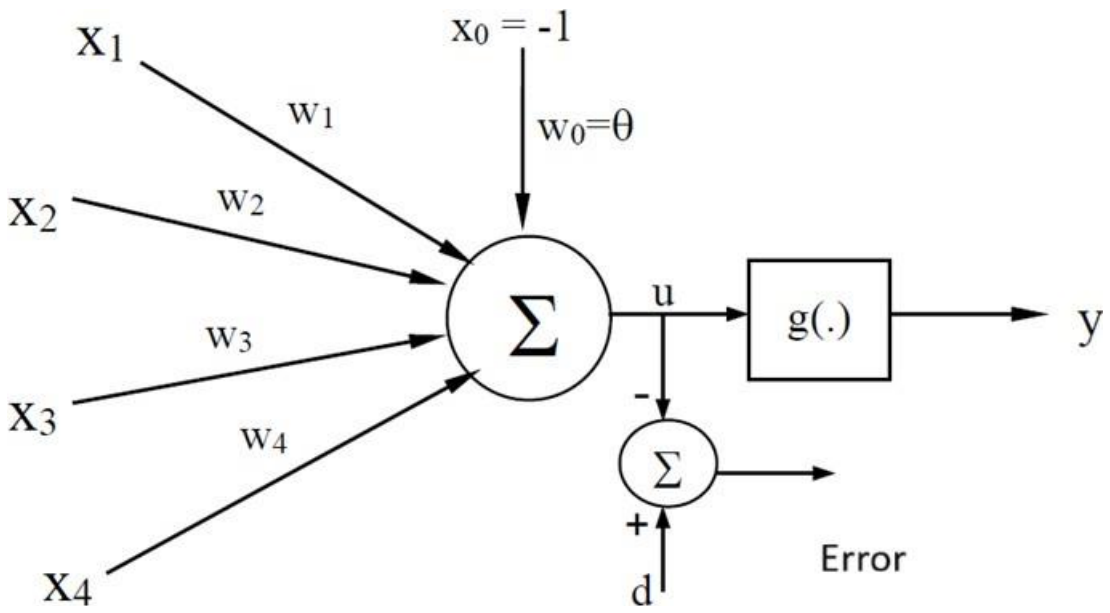
Tarea ADALINE

Un sistema de supervisión del control automático de dos válvulas, esta ubicado a 500 m de un proceso industrial, envía una señal codificada de cuatro mediciones $\{x_1, x_2, x_3, x_4\}$ que son necesarias para el ajuste de cada una de las válvulas, como se muestra en la figura. El mismo canal de comunicaciones es utilizado para realizar el accionamiento de ambas válvulas, siendo que el conmutador que esta localizado cerca de las válvulas, debe decidir si la señal es para la válvula A o B.



Durante el proceso de trasmisión las señales sufren interferencia que altera el contenido de la información. Para resolver este problema, un equipo de ingenieros y científicos pretende entrenar una red ADALINE para clasificar las señales ruidosas, indicándole al conmutador si direccionar la señal al comando de ajuste A o B.

Basado en las mediciones de las señales ruidosas, se formo un conjunto de datos de entrenamiento (Anexo), tomando como convención -1 para las señales que deben ser conmutadas para ajustar la válvula A y +1 para las señales que deben ser conmutadas para activar la válvula B. La estructura de la red ADALINE se muestra en la siguiente figura.



Utilizando el algoritmo de entrenamiento regla Delta para clasificación de patrones en ADALINE, realice las siguientes actividades.

1. Ejecute 5 entrenamientos para la red ADALINE, inicializando el vector de pesos en cada entrenamiento con valores aleatorios entre cero y uno. Si es necesario reinicie el generador de números aleatorios en cada entrenamiento para garantizar que sean diferentes. Utilice una taza de aprendizaje igual a η =0.0025 y una precisión $\varepsilon = 10^{-6}$.
2. Registre los resultados del entrenamiento en la siguiente tabla:

Training	Vector of weights (initial)					Vector of weights (final)					Number of epochs
	w_0	w_1	w_2	w_3	w_4	w_0	w_1	w_2	w_3	w_4	
#1 (T1)											
#2 (T2)											
#3 (T3)											
#4 (T4)											
#5 (T5)											

3. Para los dos primeros entrenamientos, realice la gráfica del error cuadrático medio (ECM) en función de cada época de entrenamiento. Imprima los dos gráficos en la misma hoja.
4. Después de terminar el entrenamiento del ADALINE clasifique los datos presentados en la siguiente tabla:

Table 4.3 Signal samples for classification by the ADALINE

Sample	x_1	x_2	x_3	x_4	y (T1)	y (T2)	y (T3)	y (T4)	y (T5)
1	0.9694	0.6909	0.4334	3.4965					
2	0.5427	1.3832	0.6390	4.0352					
3	0.6081	−0.9196	0.5925	0.1016					
4	−0.1618	0.4694	0.2030	3.0117					
5	0.1870	−0.2578	0.6124	1.7749					
6	0.4891	−0.5276	0.4378	0.6439					
7	0.3777	2.0149	0.7423	3.3932					
8	1.1498	−0.4067	0.2469	1.5866					
9	0.9325	1.0950	1.0359	3.3591					
10	0.5060	1.3317	0.9222	3.7174					
11	0.0497	−2.0656	0.6124	−0.6585					
12	0.4004	3.5369	0.9766	5.3532					
13	−0.1874	1.3343	0.5374	3.2189					
14	0.5060	1.3317	0.9222	3.7174					
15	1.6375	−0.7911	0.7537	0.5515					

5. Explique por qué los valores de los pesos son similares a pesar de que los entrenamientos consideran pesos iniciales y numero de épocas diferentes.

OBSERVACIONES:

1. La tarea puede ser presentada en grupos de máximo 3 personas.
2. Si la tarea requiere implementación computacional, adjunte el programa fuente.

Anexo- Conjunto de entrenamiento

Sample	x_1	x_2	x_3	x_4	d
1	0.4329	−1.3719	0.7022	−0.8535	1.0000
2	0.3024	0.2286	0.8630	2.7909	−1.0000
3	0.1349	−0.6445	1.0530	0.5687	−1.0000
4	0.3374	−1.7163	0.3670	−0.6283	−1.0000
5	1.1434	−0.0485	0.6637	1.2606	1.0000
6	1.3749	−0.5071	0.4464	1.3009	1.0000
7	0.7221	−0.7587	0.7681	−0.5592	1.0000
8	0.4403	−0.8072	0.5154	−0.3129	1.0000
9	−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

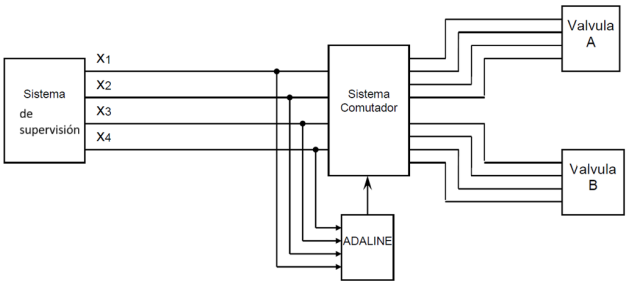
Solución

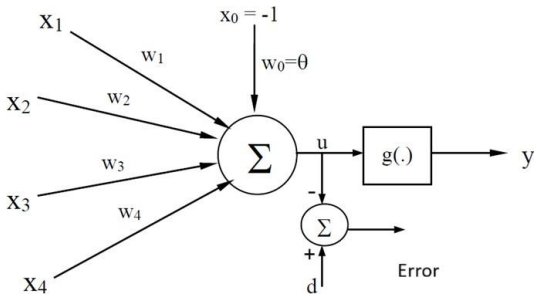
El sistema genera cuatro señales *(x1,x2,x3,x4)(x_1, x_2, x_3, x_4)*(x1 ,x2 ,x3 ,x4). Estas señales deben ser enviadas a una de dos válvulas (A o B). Debido al ruido en la comunicación, se usa una red ADALINE para decidir el destino correcto de cada muestra.

La arquitectura empleada corresponde al modelo ADALINE clásico: una suma ponderada

u=w_0 x_0 + w_1 x_1 + ... + w_4 x_4

X_0 = 1 como entrada de sesgo, la señal deseada está entre –1 y 1. La actualización de pesos se realiza con la regla Delta utilizando el error e = d – u





Conjunto de entrenamiento

Está compuesto por 35 muestras, cada una con cuatro entradas (x1,x2,x3,x4) y un valor deseado d. Las clases están codificadas como:

d = -1 --> la señal correspondiente a la válvula A

d = +1 --> la señal correspondiente a la válvula B

Se cargaron los datos del conjunto de entrenamiento desde un archivo CSV y se construyó la matriz de entradas *X*, agregando la entrada de sesgo *x_0 = -1*. También se generó el vector de salidas deseadas *d*.

```python
# ========== 2) Cargar datos ==========
csv_path = "/content/training.csv"
df = pd.read_csv(csv_path)
# Validar
print(df.shape)
df.head()
```

```
(35, 6)
```

	Sample	x1	x2	x3	x4	d
0	1	0.4329	-1.3719	0.7022	-0.8535	1.0
1	2	0.3024	0.2286	0.8630	2.7909	-1.0
2	3	0.1349	-0.6445	1.0530	0.5687	-1.0
3	4	0.3374	-1.7163	0.3670	-0.6283	-1.0
4	5	1.1434	-0.0485	0.6637	1.2606	1.0

```python
# ========== 3) Preparar matrices (X con bias, d) ==========
X = df[['x1','x2','x3','x4']].to_numpy()
# agregar bias x0 = -1 al inicio de cada vector
X_with_bias = np.hstack([ -np.ones((X.shape[0],1)), X ])  # shape: (N,5)
d = df['d'].to_numpy()   # target: -1 o +1
```

Funciones

Se implementa la función de entrenamiento del ADALINE mediante el algoritmo LMS (Regla Delta). El entrenamiento minimiza el error cuadrático medio usando descenso por gradiente en modo batch, repitiendo iteraciones hasta que

$|MSE_t - MSE_{t-1}| < \varepsilon$

Con

$\eta = 0.0025$

$\varepsilon = 10^{-6}$

```python
# ========== 4) Función de entrenamiento ADALINE (batch LMS) ==========
def train_adaline_batch(X, d, eta=0.0025, epsilon=1e-6, max_epochs=100000,
init_weights=None, verbose=False):
    """
    X: matriz (N, M) donde M incluye bias (x0=-1)
    d: vector (N,)
    devuelve: w (M,), mse_history (list), epochs (int), w0_init (copy)
    Algoritmo: batch gradient descent sobre error cuadrático medio (MSE).
    """
    N, M = X.shape
    if init_weights is None:
        w = np.random.rand(M)  # en [0,1)
    else:
        w = init_weights.copy()
    w_init = w.copy()
    mse_history = []
    prev_mse = np.inf

    for epoch in range(1, max_epochs+1):
        u = X.dot(w)                # salida lineal
        e = d - u                   # error
        mse = np.mean(e**2)         # MSE (sin 1/2)
        mse_history.append(mse)

        # criterio de parada
        if abs(prev_mse - mse) < epsilon:
            if verbose:
                print(f"converged epoch {epoch}, mse {mse:.8e}")
            break
        prev_mse = mse

        # actualización batch (regla delta / LMS)
        # grad J = -2/N * X^T (d - u)  -> update w += eta * 2/N * X^T * e
        # pero usamos factor 1: w += eta * X.T.dot(e)  (absorbe constante en eta)
        w = w + eta * X.T.dot(e)

    return w, mse_history, epoch, w_init
```

Se ejecutaron cinco entrenamientos independientes, cada uno con pesos iniciales aleatorios en el intervalo [0,1). Para cada entrenamiento se registraron

- Pesos iniciales
- Pesos finales
- Número de épocas hasta la convergencia

```
# ========== 5) Ejecutar 5 entrenamientos y registrar resultados ==========
eta = 0.0025
epsilon = 1e-6

results = []
seeds = [1,2,3,4,5]
for t in range(5):
    seed = seeds[t]
    if seed is not None:
        np.random.seed(seed)
    init_w = np.random.rand(X_with_bias.shape[1])  # pesos iniciales en [0,1)
    w_final, mse_hist, epochs, w_init = train_adaline_batch(
        X_with_bias, d, eta=eta, epsilon=epsilon, max_epochs=100000,
init_weights=init_w, verbose=False
    )
    results.append({
        'T': t+1,
        'seed': seed,
        'w0_init': w_init[0],
        'w1_init': w_init[1],
        'w2_init': w_init[2],
        'w3_init': w_init[3],
        'w4_init': w_init[4],
        'w0_final': w_final[0],
        'w1_final': w_final[1],
        'w2_final': w_final[2],
        'w3_final': w_final[3],
        'w4_final': w_final[4],
        'epochs': epochs,
        'mse_history': mse_hist
    })
```

En la Tabla X se presentan los resultados de los cinco entrenamientos ejecutados para el

ADALINE. Para cada entrenamiento se muestran los pesos iniciales, los pesos finales obtenidos tras la convergencia del algoritmo LMS y el número de épocas requeridas para cumplir con el criterio de parada $\varepsilon = 10^{-6}$.

Se observa que, a pesar de usar pesos iniciales aleatorios en el intervalo [0, 1], los pesos finales son bastante similares en los cinco entrenamientos.

	T	seed	w0_init	w1_init	w2_init	w3_init	w4_init	w0_final	w1_final	w2_final	w3_final	w4_final	epochs
0	1	1	0.417022	0.720324	0.000114	0.302333	0.146756	-0.766981	1.387590	0.924579	0.020845	-0.774580	794
1	2	2	0.435995	0.025926	0.549662	0.435322	0.420368	-0.767094	1.387628	0.924662	0.020805	-0.774635	799
2	3	3	0.550798	0.708148	0.290905	0.510828	0.892947	-0.767130	1.387705	0.924771	0.020921	-0.774700	830
3	4	4	0.967030	0.547232	0.972684	0.714816	0.697729	-0.767089	1.387812	0.924897	0.021181	-0.774768	843
4	5	5	0.221993	0.870732	0.206719	0.918611	0.488411	-0.767099	1.387790	0.924871	0.021125	-0.774754	816

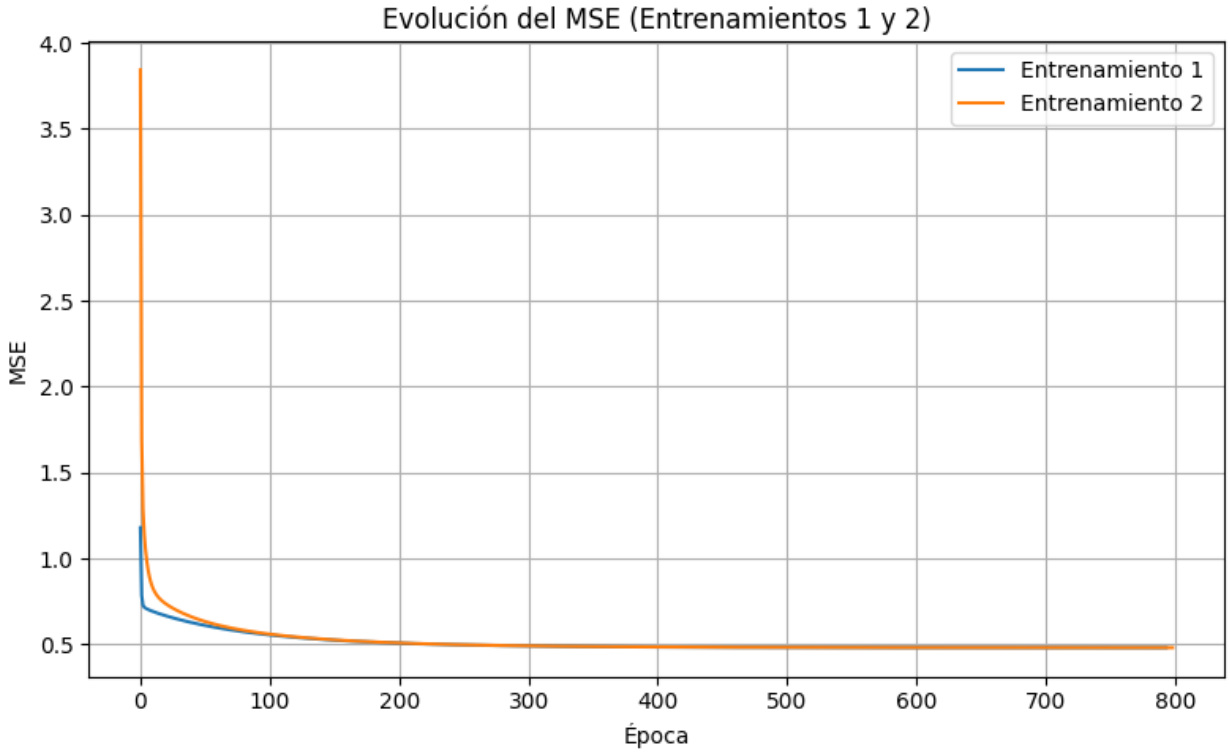
Primeros dos entrenamientos

La figura muestra la evolución del error cuadrático medio para los entrenamientos 1 y 2. Se observa que ambos convergen suavemente hacia el mínimo global, como se espera en el ADALINE debido a que el criterio de error es convexo

```
# ========== 6) Gráfica ECM para los dos primeros entrenamientos ==========

mse1 = results[0]['mse_history']
mse2 = results[1]['mse_history']

plt.figure(figsize=(8,5))
plt.plot(mse1, label='Entrenamiento 1')
plt.plot(mse2, label='Entrenamiento 2')
plt.xlabel('Época')
plt.ylabel('MSE')
plt.title('Evolución del MSE (Entrenamientos 1 y 2)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(OUTPUT / "mse_T1_T2.png", dpi=200)
plt.show()
```



Clasificación del conjunto de señales

Con los pesos obtenidos de los entrenamientos, se clasificaron 15 nuevas señales proporcionadas.
Por cada una se calculó:

$u = w^T x, \quad y = \text{sign}(u)$

```
# ========== 7) Clasificación de la tabla del punto 4 (usar los 5 entrenamientos)
==========
class_path = "/content/to_classify.csv"
to_classify = pd.read_csv(class_path)    # columnas x1,x2,x3,x4
Xc = to_classify[['x1','x2','x3','x4']].to_numpy()
Xc_bias = np.hstack([ -np.ones((Xc.shape[0],1)), Xc ])

# Generar predicciones con cada entrenamiento
pred_df = to_classify.copy()
for r in results:
    w = np.array([r['w0_final'], r['w1_final'], r['w2_final'], r['w3_final'],
r['w4_final']])
    u = Xc_bias.dot(w)
    y = np.where(u >= 0, 1, -1)   # regla de decisión
    pred_df[f'y(T{r["T"]})'] = y

pred_df.to_csv(OUTPUT / "clasificaciones.csv", index=False)
pred_df
```

Resultados de la clasificación

	Sample	x1	x2	x3	x4	y(T1)	y(T2)	y(T3)	y(T4)	y(T5)
0	1	0.9694	0.6909	0.4334	3.4965	1	1	1	1	1
1	2	0.5427	1.3832	0.6390	4.0352	-1	-1	-1	-1	-1
2	3	0.6081	-0.9196	0.5925	0.1016	1	1	1	1	1
3	4	-0.1618	0.4694	0.2030	3.0117	-1	-1	-1	-1	-1
4	5	0.1870	-0.2578	0.6124	1.7749	-1	-1	-1	-1	-1
5	6	0.4891	-0.5276	0.4378	0.6439	1	1	1	1	1
6	7	0.3777	2.0149	0.7423	3.3932	1	1	1	1	1
7	8	1.1498	-0.4067	0.2469	1.5866	1	1	1	1	1
8	9	0.9325	1.0950	1.0359	3.3591	1	1	1	1	1
9	10	0.5060	1.3317	0.9222	3.7174	-1	-1	-1	-1	-1
10	11	0.0497	-2.0656	0.6124	-0.6585	-1	-1	-1	-1	-1
11	12	0.4004	3.5369	0.9766	5.3532	1	1	1	1	1
12	13	-0.1874	1.3343	0.5374	3.2189	-1	-1	-1	-1	-1
13	14	0.5060	1.3317	0.9222	3.7174	-1	-1	-1	-1	-1
14	15	1.6375	-0.7911	0.7537	0.5515	1	1	1	1	1

Los pesos finales obtenidos en los cinco entrenamientos resultaron muy similares. Esto ocurre porque el ADALINE minimiza una función convexa (cuadrática) respecto a los pesos, lo que implica que existe un único mínimo global.

El punto de convergencia es independiente de los pesos iniciales, siempre que la tasa de aprendizaje sea suficientemente pequeña.

Diferencias menores entre los entrenamientos se deben a variaciones en los estados iniciales, en el número de épocas antes de cumplir la condición ε y en el orden del descenso.