CUCEI

Centro Universitario de Ciencias Exactas e Ingenierías

Sistemas Inteligentes

Reporte Tarea 1

Autor:
Rafael Aldo Hernández Luna

14 de febrero de 2026

1. Resumen: Machine Learning y Perceptrón Simple

El Machine Learning (Aprendizaje Automático) es una disciplina dentro de la Inteligencia Artificial que se enfoca en crear sistemas capaces de identificar patrones en grandes volúmenes de datos y realizar predicciones. A diferencia de la programación tradicional, estos algoritmos no siguen reglas fijas explícitas, sino que aprenden ajustando sus parámetros internos basándose en diferentes paradigmas:

- **Aprendizaje Supervisado:** Cuentan con un aprendizaje previo basado en etiquetas asociadas a datos para tomar decisiones o predecir (ej. detección de spam).
- **Aprendizaje No Supervisado:** No tienen conocimiento previo y buscan patrones en datos no organizados (ej. segmentación en marketing).
- **Aprendizaje por Refuerzo:** Aprenden de la experiencia mediante prueba y error, recompensando las decisiones correctas (ej. reconocimiento facial o diagnósticos médicos).

El Perceptrón Simple, inventado por Frank Rosenblatt en 1957, es la unidad fundamental de las redes neuronales artificiales. Matemáticamente, actúa como un clasificador binario lineal. El modelo consta de:

- **Entradas (x_i):** Las características del dato a procesar.
- **Pesos (w_i):** Coeficientes que ponderan la importancia de cada entrada.
- **Bias (w_0 o b):** Un término de sesgo que permite desplazar la frontera de decisión, vital para clasificar datos no centrados en el origen.
- **Función de Activación:** Generalmente una función escalón que determina si la neurona se activa (salida 1) o no (salida 0) si la suma ponderada $z = \sum w_i x_i + b$ supera un umbral.

2. Enunciado del Problema y Experimentación

La práctica consiste en implementar y analizar el comportamiento de un perceptrón simple para resolver compuertas lógicas básicas, experimentando con sus hiperparámetros.

2.1. Compuerta AND y Modificación de Hiperparámetros

El objetivo es entrenar un perceptrón para aprender la tabla de verdad AND, donde la salida es 1 únicamente si ambas entradas son 1.

Caso A: Hiperparámetros Base ($\eta = 0{,}1$, Epochs = 10)

Utilizando una tasa de aprendizaje (learning rate) de 0.1, se observa que el algoritmo ajusta los pesos gradualmente. Al inicio, los pesos son ceros. Hacia la época 4, los pesos se estabilizan, aunque el algoritmo continúa iterando hasta completar las 10 épocas.

Epoch	Inputs	Prediction	Weights
1	[0 0]	1	[-0.1 0. 0.]
1	[0 1]	0	[-0.1 0. 0.]
1	[1 0]	0	[-0.1 0. 0.]
1	[1 1]	0	[0. 0.1 0.1]
2	[0 0]	1	[-0.1 0.1 0.1]
2	[0 1]	1	[-0.2 0.1 0.]
2	[1 0]	0	[-0.2 0.1 0.]
2	[1 1]	0	[-0.1 0.2 0.1]
3	[0 0]	0	[-0.1 0.2 0.1]
3	[0 1]	1	[-0.2 0.2 0.]
3	[1 0]	1	[-0.3 0.1 0.]
3	[1 1]	0	[-0.2 0.2 0.1]
4	[0 0]	0	[-0.2 0.2 0.1]
4	[0 1]	0	[-0.2 0.2 0.1]
4	[1 0]	0	[-0.2 0.2 0.1]
4	[1 1]	1	[-0.2 0.2 0.1]
5	[0 0]	0	[-0.2 0.2 0.1]
5	[0 1]	0	[-0.2 0.2 0.1]
5	[1 0]	0	[-0.2 0.2 0.1]
5	[1 1]	1	[-0.2 0.2 0.1]
6	[0 0]	0	[-0.2 0.2 0.1]
6	[0 1]	0	[-0.2 0.2 0.1]
6	[1 0]	0	[-0.2 0.2 0.1]
6	[1 1]	1	[-0.2 0.2 0.1]
7	[0 0]	0	[-0.2 0.2 0.1]
7	[0 1]	0	[-0.2 0.2 0.1]
7	[1 0]	0	[-0.2 0.2 0.1]
7	[1 1]	1	[-0.2 0.2 0.1]
8	[0 0]	0	[-0.2 0.2 0.1]
8	[0 1]	0	[-0.2 0.2 0.1]
8	[1 0]	0	[-0.2 0.2 0.1]
8	[1 1]	1	[-0.2 0.2 0.1]
9	[0 0]	0	[-0.2 0.2 0.1]
9	[0 1]	0	[-0.2 0.2 0.1]
9	[1 0]	0	[-0.2 0.2 0.1]
9	[1 1]	1	[-0.2 0.2 0.1]
10	[0 0]	0	[-0.2 0.2 0.1]
10	[0 1]	0	[-0.2 0.2 0.1]
10	[1 0]	0	[-0.2 0.2 0.1]
10	[1 1]	1	[-0.2 0.2 0.1]

- **Pesos Finales:** $[-0{,}2, 0{,}2, 0{,}1]$.
- **Comprobación Manual:**

$$z = w_0 + w_1 x_1 + w_2 x_2 = -0{,}2 + 0{,}2x_1 + 0{,}1x_2$$

Para la entrada crítica [1, 1]: $z = -0{,}2 + 0{,}2(1) + 0{,}1(1) = 0{,}1 \geq 0 \rightarrow$ Salida 1. Para [1, 0]: $z = -0{,}2 + 0{,}2(1) + 0 = 0 \geq 0 \rightarrow$ Salida 1. *Nota:* Aunque matemáticamente $z = 0$ daría 1, la salida del código muestra predicción 0 para [1,0], lo que sugiere un manejo de precisión flotante o una condición estricta en el código interno durante la ejecución.

Caso B: Hiperparámetros Modificados ($\eta = 0{,}5$, Epochs = 10)

Al aumentar la tasa de aprendizaje a 0.5, los cambios en los pesos son más drásticos entre iteraciones (saltos de 0.5 en 0.5). La convergencia parece acelerarse, estabilizándose con pesos diferentes pero funcionales alrededor de la época 5.

Epoch	Inputs	Prediction	Weights
1	[0 0]	1	[-0.5 0. 0.]
1	[0 1]	0	[-0.5 0. 0.]
1	[1 0]	0	[-0.5 0. 0.]
1	[1 1]	0	[0. 0.5 0.5]
2	[0 0]	1	[-0.5 0.5 0.5]
2	[0 1]	1	[-1. 0.5 0.]
2	[1 0]	0	[-1. 0.5 0.]
2	[1 1]	0	[-0.5 1. 0.5]
3	[0 0]	0	[-0.5 1. 0.5]
3	[0 1]	1	[-1. 1. 0.]
3	[1 0]	1	[-1.5 0.5 0.]
3	[1 1]	0	[-1. 1. 0.5]
4	[0 0]	0	[-1. 1. 0.5]
4	[0 1]	0	[-1. 1. 0.5]
4	[1 0]	1	[-1.5 0.5 0.5]
4	[1 1]	0	[-1. 1. 1.]
5	[0 0]	0	[-1. 1. 1.]
5	[0 1]	1	[-1.5 1. 0.5]
5	[1 0]	0	[-1.5 1. 0.5]
5	[1 1]	1	[-1.5 1. 0.5]
6	[0 0]	0	[-1.5 1. 0.5]
6	[0 1]	0	[-1.5 1. 0.5]
6	[1 0]	0	[-1.5 1. 0.5]
6	[1 1]	1	[-1.5 1. 0.5]
7	[0 0]	0	[-1.5 1. 0.5]
7	[0 1]	0	[-1.5 1. 0.5]
7	[1 0]	0	[-1.5 1. 0.5]
7	[1 1]	1	[-1.5 1. 0.5]
8	[0 0]	0	[-1.5 1. 0.5]
8	[0 1]	0	[-1.5 1. 0.5]
8	[1 0]	0	[-1.5 1. 0.5]
8	[1 1]	1	[-1.5 1. 0.5]
9	[0 0]	0	[-1.5 1. 0.5]
9	[0 1]	0	[-1.5 1. 0.5]
9	[1 0]	0	[-1.5 1. 0.5]
9	[1 1]	1	[-1.5 1. 0.5]
10	[0 0]	0	[-1.5 1. 0.5]
10	[0 1]	0	[-1.5 1. 0.5]
10	[1 0]	0	[-1.5 1. 0.5]
10	[1 1]	1	[-1.5 1. 0.5]

- **Pesos Finales:** $[-1{,}5, 1{,}0, 0{,}5]$.
- **Comprobación Manual:** Para [1, 1]: $-1{,}5+1{,}0(1)+0{,}5(1) = 0 \rightarrow$ Salida 1 (Correcto). Para [1, 0]: $-1{,}5 + 1{,}0(1) + 0 = -0{,}5 < 0 \rightarrow$ Salida 0 (Correcto).

2.2. Compuerta OR y Modificación de Hiperparámetros

El objetivo es entrenar un perceptrón para aprender la tabla de verdad AND, donde la salida es 1 únicamente si ambas entradas son 1.

Caso A: Hiperparámetros Base ($\eta = 0{,}1$, Epochs = 10)

Utilizando una tasa de aprendizaje (learning rate) de 0.1, se observa que el algoritmo ajusta los pesos gradualmente. Al inicio, los pesos son ceros. Hacia la época 3, los pesos se estabilizan, aunque el algoritmo continúa iterando hasta completar las 10 épocas.

Epoch	Inputs	Prediction	Weights
1	[0 0]	1	[-0.1 0. 0.]
1	[0 1]	0	[0. 0. 0.1]
1	[1 0]	1	[0. 0. 0.1]
1	[1 1]	1	[0. 0. 0.1]
2	[0 0]	1	[-0.1 0. 0.1]
2	[0 1]	1	[-0.1 0. 0.1]
2	[1 0]	0	[0. 0.1 0.1]
2	[1 1]	1	[0. 0.1 0.1]
3	[0 0]	1	[-0.1 0.1 0.1]
3	[0 1]	1	[-0.1 0.1 0.1]
3	[1 0]	1	[-0.1 0.1 0.1]
3	[1 1]	1	[-0.1 0.1 0.1]
4	[0 0]	0	[-0.1 0.1 0.1]
4	[0 1]	1	[-0.1 0.1 0.1]
4	[1 0]	1	[-0.1 0.1 0.1]
4	[1 1]	1	[-0.1 0.1 0.1]
5	[0 0]	0	[-0.1 0.1 0.1]
5	[0 1]	1	[-0.1 0.1 0.1]
5	[1 0]	1	[-0.1 0.1 0.1]
5	[1 1]	1	[-0.1 0.1 0.1]
6	[0 0]	0	[-0.1 0.1 0.1]
6	[0 1]	1	[-0.1 0.1 0.1]
6	[1 0]	1	[-0.1 0.1 0.1]
6	[1 1]	1	[-0.1 0.1 0.1]
7	[0 0]	0	[-0.1 0.1 0.1]
7	[0 1]	1	[-0.1 0.1 0.1]
7	[1 0]	1	[-0.1 0.1 0.1]
7	[1 1]	1	[-0.1 0.1 0.1]
8	[0 0]	0	[-0.1 0.1 0.1]
8	[0 1]	1	[-0.1 0.1 0.1]
8	[1 0]	1	[-0.1 0.1 0.1]
8	[1 1]	1	[-0.1 0.1 0.1]
9	[0 0]	0	[-0.1 0.1 0.1]
9	[0 1]	1	[-0.1 0.1 0.1]
9	[1 0]	1	[-0.1 0.1 0.1]
9	[1 1]	1	[-0.1 0.1 0.1]
10	[0 0]	0	[-0.1 0.1 0.1]
10	[0 1]	1	[-0.1 0.1 0.1]
10	[1 0]	1	[-0.1 0.1 0.1]
10	[1 1]	1	[-0.1 0.1 0.1]

- **Pesos Finales:** $[-0{,}1, 0{,}1, 0{,}1]$.
- **Comprobación Manual:**

$$z = w_0 + w_1 x_1 + w_2 x_2 = -0{,}1 + 0{,}1x_1 + 0{,}1x_2$$

Para la entrada crítica [1, 1]: $z = -0{,}1 + 0{,}1(1) + 0{,}1(1) =\geq 0 \rightarrow$ Salida 1. Para [0, 0]: $z = -0{,}1 + 0 + 0 = 0 \geq -0{,}1 \rightarrow$ Salida 0.

Caso B: Hiperparámetros Modificados ($\eta = 0{,}5$, Epochs = 10)

Al aumentar la tasa de aprendizaje a 0.5, los cambios en los pesos son más drásticos entre iteraciones (saltos de 0.5 en 0.5). La convergencia parece tardar más en estabilizarse con pesos diferentes pero funcionales alrededor de la época 5.

Epoch	Inputs	Prediction	Weights
1	[0 0]	1	[-0.5 0. 0.]
1	[0 1]	0	[0. 0. 0.5]
1	[1 0]	1	[0. 0. 0.5]
1	[1 1]	1	[0. 0. 0.5]
2	[0 0]	1	[-0.5 0. 0.5]
2	[0 1]	1	[-0.5 0. 0.5]
2	[1 0]	0	[0. 0.5 0.5]
2	[1 1]	1	[0. 0.5 0.5]
3	[0 0]	1	[-0.5 0.5 0.5]
3	[0 1]	1	[-0.5 0.5 0.5]
3	[1 0]	1	[-0.5 0.5 0.5]
3	[1 1]	1	[-0.5 0.5 0.5]
4	[0 0]	0	[-0.5 0.5 0.5]
4	[0 1]	1	[-0.5 0.5 0.5]
4	[1 0]	1	[-0.5 0.5 0.5]
4	[1 1]	1	[-0.5 0.5 0.5]
5	[0 0]	0	[-0.5 0.5 0.5]
5	[0 1]	1	[-0.5 0.5 0.5]
5	[1 0]	1	[-0.5 0.5 0.5]
5	[1 1]	1	[-0.5 0.5 0.5]
6	[0 0]	0	[-0.5 0.5 0.5]
6	[0 1]	1	[-0.5 0.5 0.5]
6	[1 0]	1	[-0.5 0.5 0.5]
6	[1 1]	1	[-0.5 0.5 0.5]
7	[0 0]	0	[-0.5 0.5 0.5]
7	[0 1]	1	[-0.5 0.5 0.5]
7	[1 0]	1	[-0.5 0.5 0.5]
7	[1 1]	1	[-0.5 0.5 0.5]
8	[0 0]	0	[-0.5 0.5 0.5]
8	[0 1]	1	[-0.5 0.5 0.5]
8	[1 0]	1	[-0.5 0.5 0.5]
8	[1 1]	1	[-0.5 0.5 0.5]
9	[0 0]	0	[-0.5 0.5 0.5]
9	[0 1]	1	[-0.5 0.5 0.5]
9	[1 0]	1	[-0.5 0.5 0.5]
9	[1 1]	1	[-0.5 0.5 0.5]
10	[0 0]	0	[-0.5 0.5 0.5]
10	[0 1]	1	[-0.5 0.5 0.5]
10	[1 0]	1	[-0.5 0.5 0.5]
10	[1 1]	1	[-0.5 0.5 0.5]

- **Pesos Finales:** $[-0{,}5, 0{,}5, 0{,}5]$.
- **Comprobación Manual:** Para [1, 1]: $-0{,}5 + 0{,}5(1) + 0{,}5(1) = 0{,}5 \rightarrow$ Salida 1 (Correcto). Para [0, 0]: $-0{,}5 + 0{,}5(0) + 0{,}5(0) = -0{,}5 \rightarrow$ Salida 0 (Correcto).

2.3. Análisis de la Compuerta XOR

Se cuestiona si la compuerta XOR puede resolverse con este algoritmo. La tabla de verdad XOR es:

- $(0, 0) \rightarrow 0$
- $(0, 1) \rightarrow 1$
- $(1, 0) \rightarrow 1$
- $(1, 1) \rightarrow 0$

Al ejecutar el código, observamos que los pesos nunca convergen; oscilan indefinidamente.

Epoch	Inputs	Prediction	Weights
1	[0 0]	1	[-0.1 0. 0.]
1	[0 1]	0	[0. 0. 0.1]
1	[1 0]	1	[0. 0. 0.1]
1	[1 1]	1	[-0.1 -0.1 0.]
2	[0 0]	0	[-0.1 -0.1 0.]
2	[0 1]	0	[0. -0.1 0.1]
2	[1 0]	0	[0.1 0. 0.1]
2	[1 1]	1	[0. -0.1 0.]
3	[0 0]	1	[-0.1 -0.1 0.]
3	[0 1]	0	[0. -0.1 0.1]
3	[1 0]	0	[0.1 0. 0.1]
3	[1 1]	1	[0. -0.1 0.]
4	[0 0]	1	[-0.1 -0.1 0.]
4	[0 1]	0	[0. -0.1 0.1]
4	[1 0]	0	[0.1 0. 0.1]
4	[1 1]	1	[0. -0.1 0.]
5	[0 0]	1	[-0.1 -0.1 0.]
5	[0 1]	0	[0. -0.1 0.1]
5	[1 0]	0	[0.1 0. 0.1]
5	[1 1]	1	[0. -0.1 0.]
6	[0 0]	1	[-0.1 -0.1 0.]
6	[0 1]	0	[0. -0.1 0.1]
6	[1 0]	0	[0.1 0. 0.1]
6	[1 1]	1	[0. -0.1 0.]
7	[0 0]	1	[-0.1 -0.1 0.]
7	[0 1]	0	[0. -0.1 0.1]
7	[1 0]	0	[0.1 0. 0.1]
7	[1 1]	1	[0. -0.1 0.]
8	[0 0]	1	[-0.1 -0.1 0.]
8	[0 1]	0	[0. -0.1 0.1]
8	[1 0]	0	[0.1 0. 0.1]
8	[1 1]	1	[0. -0.1 0.]
9	[0 0]	1	[-0.1 -0.1 0.]
9	[0 1]	0	[0. -0.1 0.1]
9	[1 0]	0	[0.1 0. 0.1]
9	[1 1]	1	[0. -0.1 0.]
10	[0 0]	1	[-0.1 -0.1 0.]
10	[0 1]	0	[0. -0.1 0.1]
10	[1 0]	0	[0.1 0. 0.1]
10	[1 1]	1	[0. -0.1 0.]

Resultado en terminal:

```
Entrada: [0 0] -> Salida predicha: 1 (Incorrecto, esperado 0)
Entrada: [0 1] -> Salida predicha: 1 (Correcto)
Entrada: [1 0] -> Salida predicha: 0 (Incorrecto, esperado 1)
Entrada: [1 1] -> Salida predicha: 0 (Correcto)
```

El modelo falla en el 50 % de los casos.

Justificación: El Perceptrón Simple es un clasificador lineal, lo que significa que intenta separar las clases mediante una única línea recta (hiperplano). Geométricamente, los puntos de la compuerta XOR (0,1 y 1,0 vs 0,0 y 1,1) están dispuestos de forma cruzada en el plano

cartesiano. No existe ninguna línea recta única que pueda separar los "1s"de los "0s.[en] este caso. Por lo tanto, el problema XOR **no es linealmente separable** y no puede ser resuelto por un solo perceptrón.

3. Código Utilizado

A continuación se muestra la implementación de la clase `Perceptron` y la lógica de entrenamiento utilizada.

```python
import numpy as np
from prettytable import PrettyTable

class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=100,
                 output_file=None):
        self.w = np.zeros(input_size + 1) # +1 para el bias
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.output_file = output_file

    def activation_fn(self, x):
        return 1 if x >= 0 else 0

    def predict(self, x):
        x = np.insert(x, 0, 1) # Insertar bias
        z = self.w.T.dot(x)
        a = self.activation_fn(z)
        return a

    def train(self, X, y):
        table = PrettyTable()
        table.field_names = ["Epoch", "Inputs", "Prediction", "Weights"]
        for epoch in range(self.epochs):
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                # Regla de aprendizaje: w = w + lr * (target - output) * x
                self.w += self.learning_rate * (label - prediction) * \
                          np.insert(inputs, 0, 1)
                table.add_row([epoch + 1, inputs, prediction, self.w.copy
    ()])

        # (Codigo de escritura en archivo omitido por brevedad)

if __name__ == '__main__':
    # Configuracion para compuerta AND Modificada
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([0, 0, 0, 1])
    perceptron = Perceptron(input_size=2, learning_rate=0.5, epochs=10,
                            output_file="and_gate_modified.txt")
    perceptron.train(X, y)
```

Listing 1: Implementación del Perceptrón en Python

4. Conclusiones y Observaciones

El perceptrón simple demostró ser completamente capaz de aprender las tablas de verdad OR y AND. En ambos casos, el algoritmo logró encontrar un conjunto de pesos que clasificaba correctamente todas las entradas.

La modificación del *learning rate* (de 0.1 a 0.5) tuvo un impacto notable. Con $\eta = 0{,}5$, el perceptrón convergió en más iteraciones tanto para la compuerta AND y OR. Esto sugiere que debido a la naturaleza de la funcion OR y AND una tasa de aprendizaje elevada solo introduciria ruido al modelo de aprendizaje.

La incapacidad del perceptrón para resolver la compuerta XOR confirma la teoría de Minsky y Papert sobre las limitaciones de las redes monocapa. Como se observó los pesos oscilaron sin fin. Esto evidencia la necesidad de arquitecturas más complejas (como el Perceptrón Multicapa) para resolver problemas no linealmente separables.

Referencias

[1] Oliva, D. A., & Galvis, J. A. (2024). *Seminario de solución de problemas de IA-II: Repaso Conceptual Inteligencia Artificial* [Diapositivas de clase]. Universidad de Guadalajara, Centro Universitario de Ciencias Exactas e Ingenierías, Departamento de Ciencias Computacionales.

[2] Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.