1

Programación de Perceptrones 101

25-10-2024

Autores:
ANTONIA PAZ PAREDES LEON (21194240121) ,
ARTURO ALONSO AVALOS PASCUAL (21482639922),
LIAN ALEJANDRO CANIO SAN MARTIN (21341802522).

Docente:
JULIO ERNESTO FENNER LOPEZ

Resumen - Este informe detalla la implementación de un perceptrón para la clasificación de imágenes binarias de 10x10 píxeles, representando líneas y círculos. Se describen los fundamentos del perceptrón, el proceso de forward propagation y el uso de funciones de activación escalón y softmax para la clasificación. Se incluyen ejemplos de código y se analiza la precisión del modelo.

Índice de Términos - Perceptrón, Función de activación, Softmax, Redes neuronales.

I. INTRODUCCIÓN

El perceptrón es uno de los algoritmos de aprendizaje supervisado más simples, y es la base de las redes neuronales artificiales. Fue desarrollado por Frank Rosenblatt en 1958 y sigue siendo una herramienta fundamental en el campo del aprendizaje automático y la inteligencia artificial. El perceptrón tiene la capacidad de resolver problemas de clasificación linealmente separables mediante el ajuste de sus pesos internos basados en los ejemplos de entrenamiento.

En este proyecto, se implementa un perceptrón para clasificar imágenes de 10x10 píxeles que representan líneas horizontales, verticales y círculos. El objetivo es que el perceptrón aprenda a distinguir entre estos dos tipos de figuras geométricas, utilizando la capa oculta para transformar las entradas y aplicar una función de activación que determine la clase de la imagen.

1

II. Análisis de conceptos

A. Fundamentos matemáticos

El perceptrón realiza la clasificación a través de una serie de pasos matemáticos:

A.1. Cálculo del producto punto:

La entrada a cada neurona en la capa oculta se calcula como el producto punto entre los valores de entrada (los píxeles de la imagen, aplanados en un vector) y los pesos de la red. El producto punto se define como:

$$z = \sum_{1}^{n} x_i \cdot w_i$$

Donde: xi es el valor del píxel i, wi es el peso asociado a ese píxel y n es el número total de píxeles en la imagen.

A.2. Función de activación:

En la capa oculta se aplica la función de activación escalón. Esta función convierte el valor de entrada en una salida binaria (0) dependiendo de si el valor es positivo o negativo. La función escalón se define como:

$$f(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

En la capa de salida, se utiliza la función softmax para convertir los valores de salida de la capa oculta en probabilidades. La función softmax se define como:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

A.3. Funcion sigmoide:

Una de la primeras funciones de activación que se usaron para sustituir a la función escalón fue la sigmoide, función definida por la siguiente expresión:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Comprobamos que, para valores muy negativos de x, la función tiende al valor 0, y que, para valores muy positivos de x, la función tiende al valor 1.

A.4. Propagación hacia adelante (Forward Propagation):

El proceso de forward propagation en el perceptrón implica pasar los datos de entrada a través de la red, realizar el producto punto en la capa oculta, aplicar la función de activación escalón y finalmente, calcular las probabilidades con la función softmax en la capa de salida.

III. METODOLOGÍA

A. Generación de datos

Se generan dos tipos de imágenes binarias de 10x10 píxeles: líneas (horizontales o verticales) y círculos. Las líneas se generan aleatoriamente, mientras que los círculos se dibujan utilizando la ecuación de un círculo en una cuadrícula discreta.

B. Arquitectura del perceptrón

El perceptrón diseñado para esta tarea tiene la siguiente arquitectura:

B.1. *Capa de entrada:* 100 neuronas, una por cada píxel de la imagen (10x10 = 100).

B.2. *Capa oculta:* 50 neuronas con la función de activación escalón.

B.3. *Capa de salida:* 2 neuronas con la función de activación softmax, una para cada clase (línea o círculo).

C. Función de activación

C.1. *Función escalón en la capa oculta:* Esta función introduce no linealidades en el modelo, permitiendo que la red pueda clasificar correctamente imágenes que no son separables linealmente.

C.2. *Softmax en la capa de salida:* Convierte las salidas de la red en probabilidades para cada clase, lo que facilita la clasificación.

D. Evaluación del modelo

La precisión del modelo se calcula utilizando la siguiente fórmula:

$$\text{Precisión} = \left(\frac{\text{Número de predicciones correctas}}{\text{Número total de ejemplos}} \right) \times 100$$

IV. Implementación

A continuación se presenta el código Python utilizado para implementar el perceptrón, generar las imágenes de líneas y círculos, y evaluar el rendimiento del modelo.

A. Librerias:

La librería numpy se utiliza para realizar operaciones numéricas eficientes con matrices y vectores, esenciales para los cálculos en el perceptrón. Por otra parte, la librería matplotlib se utiliza para visualizar las imágenes generadas y los resultados de las predicciones.

```
1    import numpy as np
2    import matplotlib.pyplot as plt
```

Fig. 1. Importación de bibliotecas necesarias

B. Función de activación escalón:

Esta función toma los valores de entrada (en este caso, el resultado del producto punto entre los datos de entrada y los pesos) y aplica una activación escalón. Si el valor es mayor o igual a 0, devuelve 1; de lo contrario, devuelve 0. Esto convierte el valor en una salida binaria.

```
5    def funcion_escalon(x):  1 usage
6        return np.where(x >= 0, 1, 0)
```

Fig. 2. Función escalón

C. Función Softmax:

La función softmax convierte los valores de salida de la red en probabilidades, normalizando los resultados. Esto es útil cuando tenemos múltiples clases y queremos que la salida represente la probabilidad de pertenecer a cada clase. Se utiliza principalmente en la capa de salida de una red neuronal.

```
9    def softmax(x):  1 usage
10       exp_x = np.exp(x - np.max(x))
11       return exp_x / exp_x.sum(axis=1, keepdims=True)
```

Fig. 3. Función softmax

D. Inicialización de pesos:

Esta función inicializa los pesos de las conexiones entre las capas del perceptrón. En este caso, genera pesos aleatorios para las conexiones entre la capa de entrada y la capa oculta (pesos_entrada_oculta), y entre la capa oculta y la capa de salida (pesos_oculta_salida). El parámetro np.random.seed(42) asegura que los resultados sean reproducibles.

```
14   def inicializar_pesos(entrada_dim, oculta_dim, salida_dim):  1 usage
15       np.random.seed(42)
16       pesos_entrada_oculta = np.random.randn(entrada_dim, oculta_dim)
17       pesos_oculta_salida = np.random.randn(oculta_dim, salida_dim)
18       return pesos_entrada_oculta, pesos_oculta_salida
```

Fig. 4. Función inicializar pesos

E. Forward propagation:

Esta es la función principal para la propagación hacia adelante (forward propagation).

1. Producto punto (dot product) entre la entrada y los pesos de la capa oculta: z_oculta = np.dot(x, pesos_entrada_oculta).
2. Aplicar la función de activación escalón en la capa oculta: a_oculta = funcion_escalon(z_oculta).
3. Producto punto entre la capa oculta y los pesos de la capa de salida: z_salida = np.dot(a_oculta, pesos_oculta_salida).
4. Aplicar softmax a la capa de salida para obtener probabilidades: a_salida = softmax(z_salida).

El resultado final es la salida del perceptrón, que corresponde a las probabilidades de cada clase.

```
21    def forward_propagation(x, pesos_entrada_oculta, pesos_oculta_salida):  1 usage
22        # Capa oculta
23        z_oculta = np.dot(x, pesos_entrada_oculta)
24        a_oculta = funcion_escalon(z_oculta)
25
26        # Capa de salida
27        z_salida = np.dot(a_oculta, pesos_oculta_salida)
28        a_salida = softmax(z_salida)
29
30        return a_salida
```

Fig. 5. Función forward_propagation

F. Generación de ejemplos de líneas y círculos:

Se generan ejemplos de líneas en una cuadrícula de 10x10 píxeles. La línea puede ser horizontal o vertical, y se representa como una matriz de ceros (0s) y unos (1s). La imagen se aplana (flatten) en un vector de 100 elementos antes de ser devuelta. El número de ejemplos a generar es especificado por el parámetro n.

```
33    def generar_lineas(n):  1 usage
34        ejemplos = []
35        for _ in range(n):
36            imagen = np.zeros((10, 10))
37            linea = np.random.randint( low: 0,  high: 10)  # Línea aleatoria horizontal o vertical
38            if np.random.rand() > 0.5:  # Horizontal
39                imagen[linea, :] = 1
40            else:  # Vertical
41                imagen[:, linea] = 1
42            ejemplos.append(imagen.flatten())  # Aplanamos la imagen 10x10 a un vector de 100
43        return np.array(ejemplos)
```

Fig. 6. Función generar líneas

Además, se generan ejemplos de círculos en una cuadrícula de 10x10 píxeles. Los círculos se crean utilizando la ecuación de un círculo en un plano cartesiano, centrados en el punto (5,5) con un radio de 4. Al igual que las líneas, las imágenes se devuelven como vectores aplanados.

```python
def generar_circulos(n):  1 usage
    ejemplos = []
    for _ in range(n):
        imagen = np.zeros((10, 10))
        centro = (5, 5)
        radio = 4  # Aumentar un poco el radio para diferenciar mejor
        for i in range(10):
            for j in range(10):
                if (i - centro[0]) ** 2 + (j - centro[1]) ** 2 <= radio**2:  # Radio del círculo
                    imagen[i, j] = 1
        ejemplos.append(imagen.flatten())  # Aplanamos la imagen 10x10 a un vector de 100
    return np.array(ejemplos)
```

Fig. 7. Función generar círculos

G. Perceptron:

Esta función recibe los ejemplos a clasificar y los pesos del modelo, y realiza la clasificación en función de los resultados de la propagación hacia adelante.

```python
def perceptron(ejemplos, pesos_entrada_oculta, pesos_oculta_salida):  1 usage
    salidas = forward_propagation(ejemplos, pesos_entrada_oculta, pesos_oculta_salida)
    return np.argmax(salidas, axis=1) + 1  # Retorna 1 para líneas, 2 para círculos
```

Fig. 8. Función perceptrón

H. Calcular Precision:

Esta función calcula el porcentaje de aciertos del modelo comparando las predicciones con las etiquetas reales. La función retorna la precisión del modelo en porcentaje, lo que indica qué tan bien el modelo está clasificando las imágenes.

```python
def calcular_precision(predicciones, etiquetas):  1 usage
    return np.mean(predicciones == etiquetas) * 100
```

Fig. 9. Función cálculo de precisión

V. RESULTADOS

Se entrenó el perceptrón con 30 ejemplos de líneas y 30 ejemplos de círculos. El modelo alcanzó una precisión del 76.67% al clasificar correctamente las imágenes. En la figura siguiente, se presentan algunos ejemplos de imágenes clasificadas junto con las etiquetas reales y las predicciones del modelo.

A. Visualización de resultados:

Las imágenes clasificadas correctamente y sus predicciones pueden visualizarse utilizando la función mostrar_imagenes. A continuación se muestran algunos ejemplos generados por el modelo:

```
Precisión del modelo: 76.67%
Predicciones: [1 2 2 1 1 2 2 1 1 1 2 2 2 1 1 2 2 2 1 1 2 1 1 2 1 1 2 1 2 1 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
Etiquetas reales: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Fig. 10. Salida de la ejecución del ejemplo.

```python
def mostrar_imagenes(ejemplos, etiquetas, predicciones, num_imagenes=5):  1 usage
    for i in range(num_imagenes):
        plt.subplot( *args: 2, num_imagenes, i+1)
        plt.imshow(ejemplos[i].reshape(10, 10), cmap='gray')
        plt.title(f"Etiqueta: {etiquetas[i]} \nPred: {predicciones[i]}")
        plt.axis('off')

    for i in range(num_imagenes):
        plt.subplot( *args: 2, num_imagenes, i+num_imagenes+1)
        plt.imshow(ejemplos[i+30].reshape(10, 10), cmap='gray')
        plt.title(f"Etiqueta: {etiquetas[i+30]} \nPred: {predicciones[i+30]}")
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

Fig. 11. Mostrar imágenes

Fig. 12. Plot.

VI. Conclusiones

El perceptrón implementado es capaz de clasificar de manera efectiva imágenes binarias simples de líneas y círculos. Los resultados obtenidos muestran que el modelo puede aprender a distinguir entre dos tipos de figuras geométricas utilizando una capa oculta con 50 neuronas. Sin embargo, para mejorar la precisión del modelo, se podrían probar arquitecturas más profundas o técnicas avanzadas de optimización de pesos.

VII. Comentarios

Durante el proceso de programación, se tuvo que realizar varios ajustes para reducir el porcentaje de error (se realizaron pruebas con diferentes pesos y radios de los círculos,), ya que al inicio el porcentaje de precisión del modelo era de solo 38.33%, mostrando los siguientes resultados:

```
Precisión del modelo: 38.33%
Predicciones: [1 1 1 2 1 1 1 2 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Etiquetas reales: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Fig. 13. Salida de las primeras ejecuciones.

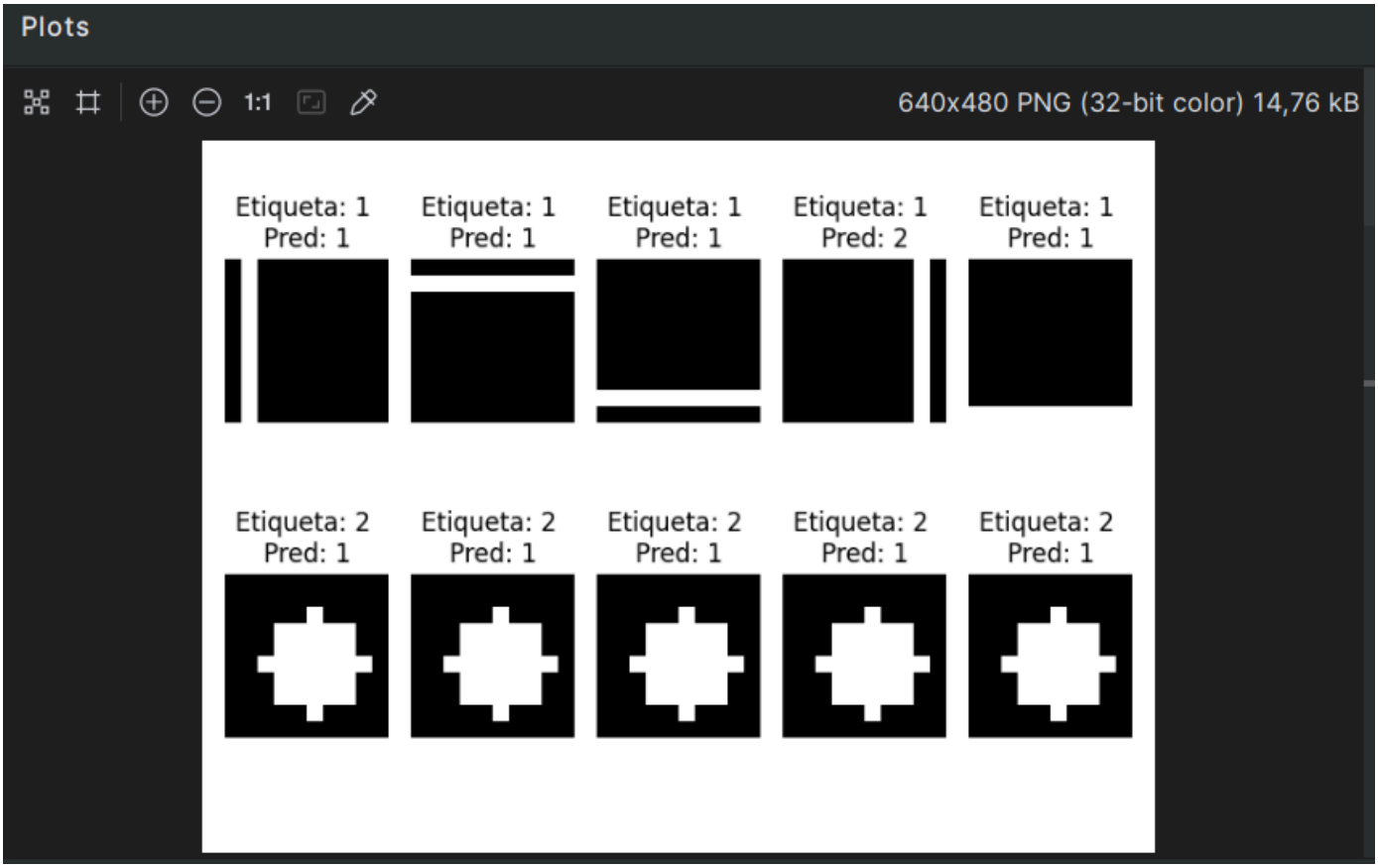

Fig. 14. Plot.

VIII. Anexos

A. Referencias

[1] "Función de activación," *Interactive Chaos*, https://interactivechaos.com/es/manual/tutorial-de-deep-learning/funcion-de-activacion

[2] "Las matemáticas del machine learning: funciones de activación," *Telefónica Tech*, https://telefonicatech.com/blog/las-matematicas-del-machine-learning-funciones-de-activacion

[3] G. Coma, "Redes Neuronales", Grupo de Tratamiento de la Información y Telecomunicaciones, Universidad de Sevilla. https://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm

B. Código fuente

[1] Link github: https://github.com/ChiaxNiro/Perceptron.git