Perceptón Backpropagation

Created	@October 17, 2024 11:03 PM
Reviewed	☐

1. Perceptrón

Algoritmo:

El perceptrón es el modelo más básico de una neurona artificial, utilizado principalmente para tareas de clasificación binaria. La idea detrás de este modelo es tomar un conjunto de entradas, multiplicarlas por pesos, sumarlas y luego pasarlas por una función de activación, usualmente una función escalón.

Algoritmo en Python:

```
import numpy as np

class Perceptron:
    def __init__(self, learning_rate=0.01, epochs=100):
        self.learning_rate = learning_rate
        self.epochs = epochs

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

    def fit(self, X, y):
        # Inicializar los pesos con ceros
        self.weights = np.zeros(X.shape[1] + 1)  # +1 para el
sesgo (bias)
```

```
        for _ in range(self.epochs):
            for xi, target in zip(X, y):
                update = self.learning_rate * (target - self.
predict(xi))
                self.weights[1:] += update * xi
                self.weights[0] += update  # Sesgo

    def predict(self, X):
        net_input = np.dot(X, self.weights[1:]) + self.weight
s[0]
        return self.step_function(net_input)

# Ejemplo de uso:
if __name__ == "__main__":
    # Datos de entrenamiento (AND gate)
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([0, 0, 0, 1])

    # Crear y entrenar el perceptrón
    perceptron = Perceptron(learning_rate=0.1, epochs=10)
    perceptron.fit(X, y)

    # Predicciones
    print("Predicciones:", [perceptron.predict(xi) for xi in
X])
```

Parámetros de entrada:

- `learning_rate`: tasa de aprendizaje para ajustar los pesos.
- `epochs`: número de iteraciones sobre el conjunto de datos.
- `X`: conjunto de características de entrada.
- `y`: etiquetas (0 o 1 para clasificación binaria).

Aplicaciones:

El perceptrón es adecuado para tareas de **clasificación lineal**. Algunas aplicaciones típicas incluyen:

- Clasificación de imágenes simples.
- Clasificación de patrones lineales.
- Implementación básica de compuertas lógicas como AND, OR.

2. Backpropagation en Redes Neuronales

Algoritmo:

Backpropagation es el algoritmo utilizado para entrenar redes neuronales multicapa ajustando los pesos mediante el gradiente descendente. Este algoritmo calcula el error en la capa de salida y lo propaga hacia atrás a través de las capas de la red, ajustando los pesos en cada capa.

Algoritmo en Python:

```
import numpy as np

class NeuralNetwork:
    def __init__(self, X, y, hidden_neurons=4, learning_rate=
0.1, epochs=10000):
        self.input = X
        self.weights1 = np.random.rand(self.input.shape[1], h
idden_neurons)
        self.weights2 = np.random.rand(hidden_neurons, 1)
        self.y = y
        self.output = np.zeros(self.y.shape)
        self.learning_rate = learning_rate
        self.epochs = epochs

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
```

```
        return x * (1 - x)

    def feedforward(self):
        self.layer1 = self.sigmoid(np.dot(self.input, self.we
ights1))
        self.output = self.sigmoid(np.dot(self.layer1, self.w
eights2))

    def backpropagation(self):
        # Calcular el error de la capa de salida
        error = self.y - self.output
        d_weights2 = np.dot(self.layer1.T, error * self.sigmo
id_derivative(self.output))

        # Calcular el error de la capa oculta
        error_hidden_layer = np.dot(error * self.sigmoid_deri
vative(self.output), self.weights2.T)
        d_weights1 = np.dot(self.input.T, error_hidden_layer
* self.sigmoid_derivative(self.layer1))

        # Actualizar los pesos
        self.weights1 += self.learning_rate * d_weights1
        self.weights2 += self.learning_rate * d_weights2

    def train(self):
        for _ in range(self.epochs):
            self.feedforward()
            self.backpropagation()

# Ejemplo de uso:
if __name__ == "__main__":
    # Datos de entrenamiento (XOR gate)
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [1], [1], [0]])

    # Crear y entrenar la red neuronal
```

```
    nn = NeuralNetwork(X, y)
    nn.train()

    # Predicciones
    print("Predicciones:", nn.output)
```

Parámetros de entrada:

- `X`: conjunto de características de entrada.
- `y`: etiquetas (valores esperados de salida).
- `hidden_neurons`: número de neuronas en la capa oculta.
- `learning_rate`: tasa de aprendizaje.
- `epochs`: número de iteraciones.

Aplicaciones:

Backpropagation se utiliza para tareas de **clasificación no lineal**, **regresión** y más. Ejemplos incluyen:

- Reconocimiento de patrones complejos como dígitos escritos a mano (MNIST).
- Predicción de series temporales.
- Clasificación de imágenes y textos.
- Control robótico.

Resumen de aplicaciones:

- **Perceptrón**: clasificación lineal binaria (ej., compuertas lógicas).
- **Backpropagation**: clasificación y regresión en redes neuronales profundas para tareas más complejas como reconocimiento de imágenes, procesamiento de lenguaje natural, y sistemas predictivos.