8. 확률적 경사하강법과 선형대수학

1. 경사하강법과 선형대수학의 관계

   경사하강법은 함수의 기울기(즉, 경사)를 따라 함수 값을 최소화하는 방법입니다. 이때, 기울기는 미분을 통해 구할 수 있으며, 다차원 공간에서는 기울기 벡터가 됩니다. 이 과정에서 선형대수학적 연산, 즉 행렬과 벡터 연산이 필수적으로 사용됩니다.

   선형대수학의 기본개념

   1. 벡터(Vector): 1차원 배열로 이해할 수 있습니다. 예를 들어, 변수들의 집합 $w = [w_1, w_2, ..., w_n]$는 벡터로 표현됩니다.
   2. 행렬(Matrix): 2차원 배열로, 여러 벡터의 집합으로 이해할 수 있습니다. 행렬은 여러 변수의 상호 관계를 표현하는 데 사용됩니다.
   3. 행렬 곱셈(Matrix Multiplication): 경사하강법에서 다중 변수의 가중치를 업데이트할 때, 주로 행렬곱셈을 통해 계산이 이루어집니다.
   4. 전치 행렬(Transpose): 행렬을 업데이트할 때 사용되며, 행과 열을 바꾸는 작업입니다.

2. 경사하강법의 수학적 개념

   경사하강법 기본 수식

   경사하강법의 목표는 주어진 손실 함수를 최소화하는 것입니다.
   예를 들어,
   $L(w)$라는 손실함수가 주어졌다고 하면, 경사하강법은 아래와 같이 가중치 w를 업데이트 합니다.

   $$w := w - \alpha \nabla L(w)$$

   - α는 학습률(learning rate) 입니다.
   - ∇L(w)는 손실함수 $L(w)$에 대한 기울기(gradient)를 의미하며, 이 기울기는 벡터로 표현 됩니다.

   기울기의 계산

   기울기는 벡터 민분을 통해 계산됩니다. 예를들어, $L(w)$가 다음과 같은 이차함수 일 때,

$$L(w) = \frac{1}{2} w^T A w - b^T w + c$$

여기서 A는 대칭 행렬, b는 벡터, c는 상수입니다. 이 손실 함수에 대한 기울기 ∇L(w)는 다음과 같이 구할 수 있습니다.

$$\nabla L(w) = Aw - b$$

이 결과는 행렬과 벡터 연산을 기반으로 한 선형대수적 결과입니다.

3. 확률적 경사하강법(SGD)

   확률적 경사하강법은 경사하강법의 변형으로, 전체 데이터셋을 사용하는 대신 데이터의 부분 집합을 사용하여 기울기를 계산합니다. 이 방ㅂㅂ은 매 반복마다 비용을 줄일 수 있어 큰 데이터셋에서 더 효율적입니다.

   확률적 경사하강법 수식

$$w := w - \alpha \nabla L(w; x^{(i)}, y^i)$$

   x^i, y^i는 데이터의 i번째 샘플입니다.

   왜 행렬 연산이 중요한가 ?

   - 다수의 입ㄹ력과 출력이 존재하는 신경망에서는, 행렬 연산을 통해 데이터를 빠르게 처리하고 병렬 계산을 수행할 수 있습니다.
   - 신경망의 가중치 업데이트는 주로 행렬 곱을 통해 이루어지며, 이는 벡터들의 변환 과정과 밀접하게 연결되어 있습니다.

```
import numpy as np

# 데이터셋 (x, y)
X = np.array([[1, 2], [2, 4], [3, 6], [4, 8]])  # 입력 데이터
y = np.array([2, 4, 6, 8])  # 실제 출력값

# 가중치 초기화
W = np.random.randn(2)  # 2차원 벡터
b = 0  # 편향
```

```
# 학습률과 반복 횟수
learning_rate = 0.01
epochs = 1000

# 경사하강법 함수
def gradient_descent(X, y, W, b, learning_rate, epochs):
    m = len(y)
    for epoch in range(epochs):
        for i in range(m):
            xi = X[i]
            yi = y[i]

            # 예측 값
            y_pred = np.dot(xi, W) + b

            # 오차
            error = y_pred - yi

            # 가중치와 편향 업데이트
            W -= learning_rate * error * xi
            b -= learning_rate * error

        if epoch % 100 == 0:
            cost = (1/m) * np.sum((np.dot(X, W) + b - y) ** 2
            print(f"Epoch {epoch}, Cost: {cost}")

    return W, b

# 경사하강법 실행
W_final, b_final = gradient_descent(X, y, W, b, learning_rate
print(f"최종 가중치: {W_final}, 최종 편향: {b_final}")
```

설명

1. 데이터셋: X는 2차원 입력 데이터이며, y는 실제 값입니다.
2. 가중치 업데이트: 각 데이터 포인트에 대해 예측 값과 실제 값의 차이(오차)를 계산하고, 이를 바탕으로 가중치와 편향을 업데이트 합니다.
3. 파라미터 업데이트: 학습률과 오차를 곱해 가중치와 편향을 갱신합니다.