인공 신경망

(주)인피닉스 – 강호용 연구원

Infinyx

경사 하강법 소개

Infinyx
THINK LIFE SYNC AI

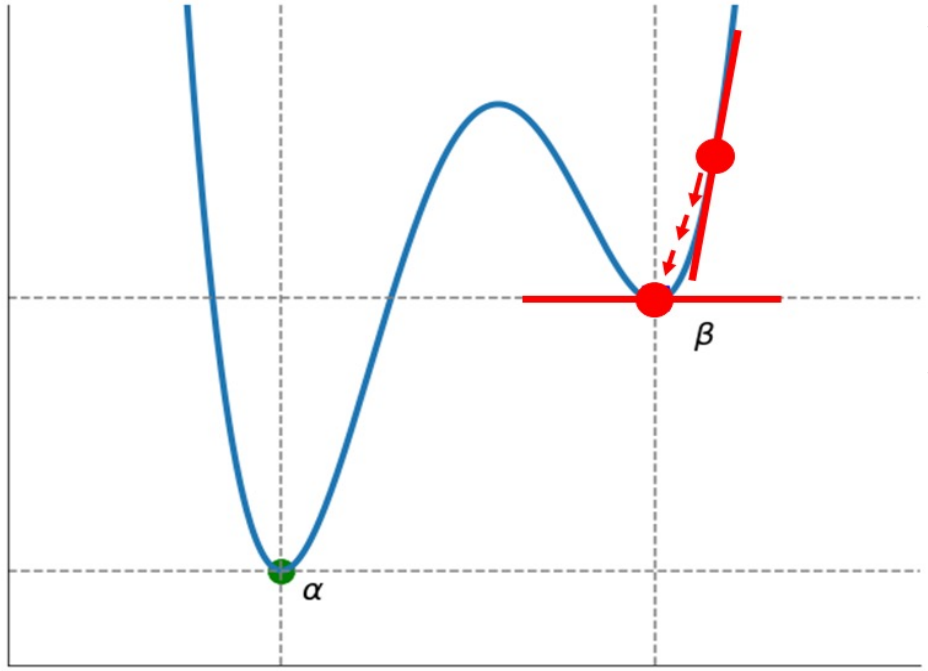
경사 하강법 개요

- ✓ 함수의 기울기(경사)를 이용하여 최솟값을 찾아가는 최적화 방법 중 하나입니다. 이 방법은 기계 학습에서 모델 학습을 위한 가중치(weight)와 bias(편향)을 찾을 때 많이 사용됩니다.

- ✓ 경사 하강법이 필요한 이유는 기계 학습에서 최적의 모델을 찾기 위해서는 모델의 손실 함수(loss function) 값을 최소화하는 가중치와 편향을 찾아야 합니다. 하지만 가중치와 편향을 수식으로 풀어내기 어려운 경우가 많아, 이를 해결하기 위해 경사 하강법을 이용해 최적의 가중치와 편향을 찾습니다.

- ✓ 초기 가중치와 편향 값을 설정하고, 손실 함수의 기울기(경사)를 이용해 가중치와 편향 값을 반복적으로 업데이트하는 과정을 거칩니다. 이때 학습률(learning rate)이라는 하이퍼파라미터를 이용하여 업데이트를 얼마나 크게 할지 결정합니다.

경사 하강법 개요



- ✓ 경사 하강법의 종류에는 Batch Gradient Descent, Stochastic Gradient Descent, Mini-batch Gradient Descent 등이 있으며, 이 중에서도 가장 일반적인 Batch Gradient Descent는 전체 데이터셋을 이용해 가중치와 편향을 업데이트하기 때문에 계산 비용이 크다는 단점이 있습니다.
- ✓ 따라서 대규모 데이터셋에서는 Stochastic Gradient Descent나 Mini-batch Gradient Descent를 주로 사용합니다.

경사 하강법 필요성 소개

✓ 빅 데이터 시대에서 모델의 학습 시간과 성능이 중요한 이유 ?

빅 데이터 시대에는 데이터의 양이 기하급수적으로 증가하고 있습니다. 이에 따라 학습에 필요한 계산량도 많아지고, 모델 학습 시간이 길어지는 문제가 발생하고 있습니다.

따라서 모델의 학습 시간을 줄이기 위해, 병렬 처리를 활용하거나 하드웨어 업그레이드를 하는 등의 대안이 제시되고 있습니다. 또한 모델의 성능 또한 중요한 문제입니다.

빅 데이터를 다루는 경우, 모델의 성능이 얼마나 높은지에 따라 예측의 정확도와 신뢰성이 결정됩니다.
따라서 모델의 성능을 향상시키기 위해, 다양한 알고리즘 및 기술을 활용하고 있습니다.

경사 하강법 필요성 소개

❑ 경사 하강법이 대용량 데이터에서 모델 학습에 사용되는 이유 ?

경사 하강법(Gradient Descent)은 대용량 데이터에서 모델 학습에 사용되는 이유 중 하나는 데이터를 한 번에 처리하지 않고 일부씩 처리하면서 최적의 모델 파라미터를 찾을 수 있기 때문입니다.

만약 데이터가 매우 크다면 모든 데이터를 한 번에 학습시키는 것은 시간이 오래 걸릴 뿐만 아니라 메모리도 많이 소모합니다.

따라서 일부 데이터를 무작위로 선택하여 모델을 학습시키는 미니배치 학습(Mini-Batch Learning)이나 온라인 학습(Online Learning) 방법이 사용됩니다. 이런 경우, 경사 하강법은 각 미니배치나 데이터 샘플마다 파라미터를 조금씩 업데이트하면서 모델을 학습시키는데 적합한 방법입니다.

또한, 경사 하강법은 모델 파라미터의 초기값에 따라 학습 속도와 수렴 여부가 달라지므로, 대부분의 경우 랜덤 초기값을 사용하게 됩니다. 이런 경우, 경사 하강법은 초기값에 관계없이 언젠가는 최적의 모델 파라미터를 찾아냅니다.

이러한 특성 때문에 대용량 데이터에서도 경사 하강법이 자주 사용되는 최적화 알고리즘 중 하나입니다.

경사 하강법 개념 원리 소개

경사 하강법(Gradient Descent)은 머신 러닝에서 가장 기본적인 최적화 알고리즘 중 하나입니다. 모델의 파라미터를 조정하여 손실 함수(Loss Function) 값을 최소화하는 방향으로 학습을 진행하는 알고리즘입니다.

경사 하강법은 현재 파라미터에서의 기울기(Gradient) 값을 구하고, 그 기울기가 감소하는 방향으로 파라미터를 업데이트하는 방식으로 동작합니다.

즉, 현재 위치에서 기울기를 구한 후, 기울기가 감소하는 방향으로 이동하여 새로운 위치를 찾아갑니다. 이 과정을 손실 함수의 값이 최소가 되는 지점을 찾을 때까지 반복합니다.

경사 하강법은 다양한 변형 알고리즘이 존재하며, 주어진 문제에 따라 적합한 알고리즘을 선택하여 사용합니다.

대표적인 변형 알고리즘으로는 확률적 경사 하강법(Stochastic Gradient Descent), 미니 배치 경사 하강법(Mini-batch Gradient Descent) 등이 있습니다.

경사 하강법 간단하게 수식으로 이해 해보기

경사 하강법은 최적화 알고리즘 중 하나로, 함수의 기울기를 활용하여 함수의 최솟값을 찾아가는 방법입니다.
예를 들어, 다음과 같은 함수가 있다고 가정해보겠습니다.

$$f(x) = x^2 - 6x + 5$$

이 함수의 최솟값을 구하고자 한다면, 함수의 도함수를 구하고 이 도함수의 극솟값이 위치하는 지점을 찾아가면 됩니다. 도함수는 다음과 같습니다.

$$f'(x) = 2x - 6$$

이 도함수를 통해 얻은 기울기 값을 활용하여 함수의 최솟값을 찾아가는 것이 경사 하강법입니다.
경사 하강법에서는 다음과 같은 식을 반복적으로 수행하면서 함수의 최솟값에 근접하게 됩니다.

$$x_{t+1} = x_t - \alpha f'(x_t)$$

경사 하강법 간단하게 수식으로 이해 해보기

이 도함수를 통해 얻은 기울기 값을 활용하여 함수의 최솟값을 찾아가는 것이 경사 하강법입니다.
경사 하강법에서는 다음과 같은 식을 반복적으로 수행하면서 함수의 최솟값에 근접하게 됩니다.

$$x_{t+1} = x_t - \alpha f'(x_t)$$

여기서 xt는 현재 위치, x{t+1}은 다음 위치, alpha는 학습률(learning rate)로, 이 값을 조절하여 경사 하강법이 얼마나 큰 보폭으로 이동할지를 결정할 수 있습니다.

이를 다시 f(x) = x^2 - 6x + 5 함수를 기준으로 설명하면, 초기값인 x0를 정한 후, f'(x0) 값을 계산하여 이 값의 부호에 따라 x1의 위치를 결정합니다. 예를 들어, x0가 5라고 하면 f'(5) = 4이므로 기울기가 양수이기 때문에 x1은 5 - alpha 4의 값이 됩니다. 이후에도 이 과정을 반복하면서 f(x)의 최솟값에 근접해가는 과정을 수행합니다.

 # 다중 변수 함수의 경우에는 경사 하강법이 각 변수에 대한 편도함수를 계산하여 이를 활용합니다.

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

여기서 theta_j는 j번째 변수의 값을 나타내며, h_theta(x^{(i)})는 모델의 예측값을 나타내고, y^{(i)}는 실제값을 나타냅니다. x_j^{(i)}는 i번째 샘플의 j번째 특성값을 나타내며, alpha는 학습률을 의미

경사 하강법은 비용 함수의 기울기가 낮아지는 방향으로 파라미터를 업데이트하기 때문에, 최소값에 수렴할 때까지 반복적으로 업데이트합니다.

단순한 경사 하강법은 모든 데이터를 하나의 반복문 안에서 처리하기 때문에, 대용량 데이터에서는 연산 속도가 느리고 메모리 부담이 큽니다.

이를 해결하기 위해 미니 배치 경사 하강법과 스토캐스틱 경사 하강법 등의 방법이 사용됩니다.

 # 경사 하강법에서 사용되는 하이퍼파라미터 → 학습률과 반복 횟수의 역할 ?

경사 하강법에서 사용되는 학습률(learning rate)과 반복 횟수(iterations)는 모델의 성능과 학습 시간을 조절하는 데 중요한 역할을 합니다.

학습률은 각 반복에서 이동하는 거리를 결정하며, 값이 너무 크면 최적점을 지나칠 수 있고, 값이 너무 작으면 학습 속도가 느려질 수 있습니다. 따라서 적절한 학습률을 설정하는 것이 중요합니다.

반복 횟수는 모델이 최적점에 수렴하는 데 필요한 시간을 결정합니다. 더 많은 반복을 허용하면 모델이 더 정확하게 수렴할 가능성이 높아지지만, 불필요한 계산을 초래할 수 있으므로 적절한 값을 설정해야 합니다. 일반적으로는 모델이 충분히 수렴할 때까지 반복을 진행하거나 미리 정한 최대 반복 횟수를 초과하지 않도록 설정합니다.

경사 하강법 종류 소개

경사 하강법 종류 소개	
배치 경사 하강법 (Batch Gradient Descent)	모든 학습 데이터를 사용하여 한 번의 업데이트 수행 전체 데이터에 대한 미분값을 계산하여 가중치를 업데이트 하므로 계산량이 크고, 전역 최소값에 도달할 가능성이 있음
확률적 경사 하강법 (Stochastic Gradient Descent)	매개변수를 하나씩 업데이트 하는 방법 학습 데이터를 무작위로 선택하여 가중치를 업데이트 하므로 계산량이 적고, 지역 최소값을 탈출할 가능성이 있음
미니 배치 경사 하강법 (Mini-batch Gradient Descent)	일정한 크기의 미니 배치를 사용하여 가중치를 업데이트하는 방법 배치 경사 하강법과 확률적 경사 하강법의 장점을 결합한 방법으로, 계산량과 정확도를 적절히 조절할 수 있음

이 세 가지 방법은 각각의 장단점이 있으며, 데이터의 크기, 모델의 복잡도, 학습 속도 등 여러 가지 요인에 따라 적절한 방법을 선택하여 사용해야 합니다.

경사 하강법 종류별 특징

경사 하강법 장단점 소개	
배치 경사 하강법 (Batch Gradient Descent)	장점 - 최적의 해에 도달할 때까지 수렴이 느리지만, 전역 최적점을 보장 - 전체 데이터를 통해. 학습시키기 때문에, 가장 업데이트 횟수가 적다. 단점 - 전체 데이터를 샘플을 한번에 처리하기 때문에 메모리가 가장 많이 필요
확률적 경사 하강법 (Stochastic Gradient Descent)	장점 - 각 샘플마다 가중치를 업데이트 - 전체 데이터를 처리하지 않기 때문에 빠른 속도로 학습 가능하며, 대용량 데이터에 적합 단점 - 수렴이 불안정하며, 지역 최적점에 수렴할 가능성 존재.
미니 배치 경사 하강법 (Mini-batch Gradient Descent)	장점 - 배치 경사 하강법과 확률적 경사 하강법의 장점을 모두 가짐 - 배치 크기를 잘 선택하면, 빠른 속도로 최적의 해를 수렴할 수 있음 단점 - 배치 크기에 따라 결과가 달라질 수 있음

경사 하강법을 이용한 선형 회귀 모델 구현 실습

종속 변수와 독립 변수 사이의 관계를 모델링하기 위해 경사 하강법을 사용하여 선형 회귀 모델을 구현할 수 있습니다.

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

다음으로 입력 데이터와 타깃 데이터를 생성합니다. 이 예시에서는 y = 2x + 1 관계를 따르는 데이터를 생성합니다.

```
x = np.array([1, 2, 3, 4, 5])
y = np.array([3, 5, 7, 9, 11])
```

모델의 예측값과 실제 타깃값 사이의 오차를 계산하는 함수를 정의합니다.

```
def compute_error(w0, w1, x, y):
    y_pred = w0 + w1 * x
    error = y - y_pred
    return error
```

경사 하강법을 이용한 선형 회귀 모델 구현 실습

경사 하강법을 수행하는 함수를 정의합니다.

```
def gradient_descent(x, y, alpha=0.05, iterations=1000):
    n = len(x)
    w0, w1 = 0, 0

    for i in range(iterations):
        error = compute_error(w0, w1, x, y)
        w0 -= alpha * (-2.0 / n) * np.sum(error)
        w1 -= alpha * (-2.0 / n) * np.sum(error * x)

    return w0, w1
```

X : 입력 데이터
Y : 목표 변수 데이터
Alpha : 학습률, 기본값은 0.05 입니다.
Iterations : 경사 하강법의 반복 횟수, 기본값은 1000
N : 데이터 총 개수
W0 : 절편 값 초기값은 0 설정
W1 : 기울기 값 초기값은 0 설정

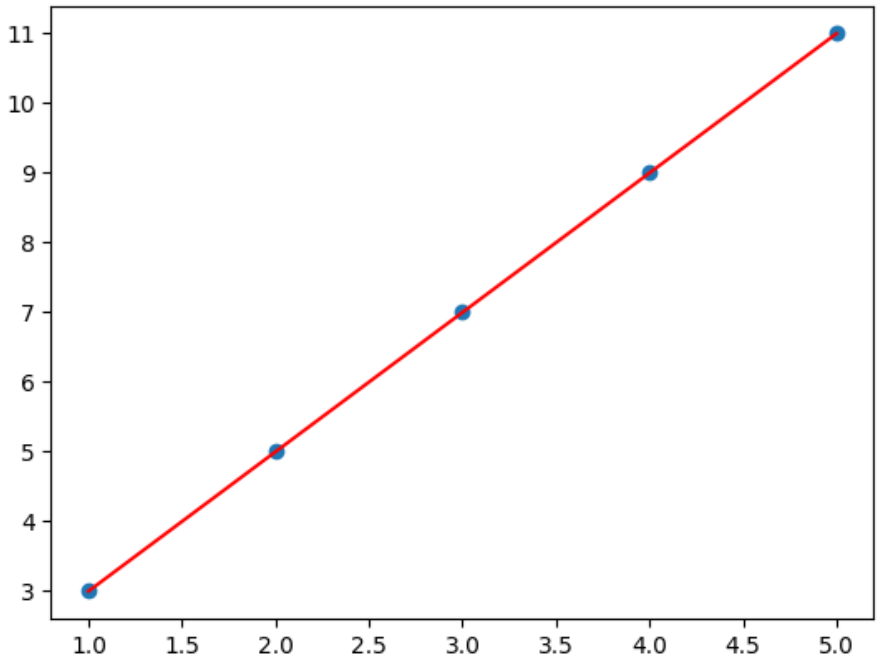
for 반복문에서는 iterations만큼 경사 하강법을 수행합니다. compute_error 함수를 이용하여 현재 가중치 값으로 예측한 값과 실제 목표 변수 값 사이의 오차를 계산합니다. 이 오차 값을 이용하여 w0과 w1을 갱신합니다.

이때, alpha는 학습률을 나타내며, 각 가중치의 변화량을 조절하는 역할을 합니다.
마지막으로 학습된 w0와 w1 값을 반환합니다. 이 값은 선형 회귀 모델의 절편과 기울기를 나타냅니다.

경사 하강법을 이용한 선형 회귀 모델 구현 실습

이제 모델을 학습시키고 결과를 시각화해보겠습니다.

```
# 모델 학습
w0, w1 = gradient_descent(x, y)

# 결과 시각화
plt.scatter(x, y)
plt.plot(x, w0 + w1 * x, color='red')
plt.show()
```



위 코드에서 plt.scatter 함수는 입력 데이터와 타깃 데이터를 산점도로 시각화합니다.
plt.plot 함수는 모델의 예측값을 직선으로 시각화합니다.

다중 선형 –실습 [경사 하강법을 이용한 다중 선형 회귀 구현 실습하기]

```python
import numpy as np

# 예시 데이터 생성
x1 = np.array([1, 2, 3, 4, 5])
x2 = np.array([0, 1, 0, 1, 0])
y = np.array([3, 5, 7, 9, 11])


def gradient_descent(x1, x2, y, alpha, iterations):
    # 초기값 설정
    n = len(y)
    beta_0 = 0
    beta_1 = 0
    beta_2 = 0

    # 경사 하강법 수행
    for i in range(iterations):
        y_pred = beta_0 + beta_1 * x1 + beta_2 * x2
        error = y_pred - y
        beta_0 -= alpha * (1/n) * np.sum(error)
        beta_1 -= alpha * (1/n) * np.sum(error * x1)
        beta_2 -= alpha * (1/n) * np.sum(error * x2)

    return beta_0, beta_1, beta_2


beta_0, beta_1, beta_2 = gradient_descent(x1, x2, y, 0.01, 1000)
print("beta_0:", beta_0)
print("beta_1:", beta_1)
print("beta_2:", beta_2)


x1_new = 1
x2_new = 0

y_pred = beta_0 + beta_1*x1_new + beta_2*x2_new

print("Predicted y value:", y_pred)
```

선형 회귀에서 설명한 단순 선형 회귀 코드와 매우 유사하며, 다만 입력 변수가 2개인 다중 선형 회귀에서는 입력 변수의 정규화가 필요합니다.

결과적으로 beta_0, beta_1, beta_2를 반환하고 출력합니다.

beta_0: 0.8907546215358821
beta_1: 2.0237699271134932
beta_2: 0.05788830985912915 6

Predicted y value: 2.9145245486493754

다중 선형 –실습 [경사 하강법을 이용한 다중 선형 회귀 구현 실습하기]

```python
import numpy as np

# 예시 데이터 생성
x1 = np.array([1, 2, 3, 4, 5])
x2 = np.array([0, 1, 0, 1, 0])
y = np.array([3, 5, 7, 9, 11])


def gradient_descent(x1, x2, y, alpha, iterations):
    # 초기값 설정
    n = len(y)
    beta_0 = 0
    beta_1 = 0
    beta_2 = 0

    # 경사 하강법 수행
    for i in range(iterations):
        y_pred = beta_0 + beta_1 * x1 + beta_2 * x2
        error = y_pred - y
        beta_0 -= alpha * (1/n) * np.sum(error)
        beta_1 -= alpha * (1/n) * np.sum(error * x1)
        beta_2 -= alpha * (1/n) * np.sum(error * x2)

    return beta_0, beta_1, beta_2


beta_0, beta_1, beta_2 = gradient_descent(x1, x2, y, 0.01, 1000)
print("beta_0:", beta_0)
print("beta_1:", beta_1)
print("beta_2:", beta_2)


x1_new = 1
x2_new = 0

y_pred = beta_0 + beta_1*x1_new + beta_2*x2_new

print("Predicted y value:", y_pred)
```

이 코드는 다중 선형 회귀에서 경사 하강법을 수행하여 회귀 계수 beta_0, beta_1, beta_2를 구하는 함수입니다.
먼저 iterations는 경사 하강법을 몇 번 반복할지를 정하는 파라미터입니다.

alpha는 학습률(learning rate)로, 경사 하강법에서 한 번에 얼마나 많이 이동할지를 조절합니다. 이 값이 너무 작으면 학습이 느리고, 너무 크면 발산할 가능성이 있습니다.

y_pred는 현재 beta_0, beta_1, beta_2 값으로부터 예측한 종속변수 y의 값입니다. error는 예측값과 실제값의 차이입니다.
이를 이용하여 각 회귀 계수에 대해 경사 하강법을 수행합니다.

beta_0의 경우, 단순 선형 회귀에서와 같이 오차의 평균값을 빼줍니다. beta_1과 beta_2는 오차와 독립 변수 x_1 및 x_2의 곱의 평균값을 빼줍니다.
마지막으로 계산된 beta_0, beta_1, beta_2를 반환합니다.

경사 하강법에서 발생할 수 있는 문제점과 해결방법

문제	원인	해결 방법
경사 하강법이 수렴하지 않는 문제	학습률이 너무 크거나 작거나, 목적 함수가 비선형 함수인 경우	1. 학습률 조정 2. 더 좋은 초기값을 선택 3. 다른 알고리즘 사용
지역 최소값 에 수렴하는 문제	목적 함수가 복잡하고 다중 국소 최소값이 존재하는 경우	1. 초기값을 다르게 설정 2. 모멘텀이나 다른 최적화 알고리즘 사용하기
과적합 문제	학습 데이터에 너무 맞춰져서 일반화 성능이 떨어진 경우	1. 규제 기법을 사용하여 모델 복잡도 제한 2. 데이터 양 늘리기 3. 일반화 데이터 추가 하기
수치적 불안정성 문제	경사 하강법에서 계산되는 수치값이 커지면 수치적 불안정성 문제가 발생	1. 정규화를 사용하여 수치값의 범위조절 2. 경사 하강법 대신 다른 최적화 알고리즘 사용

학습률과 반복 횟수에 따른 모델 성능 변화 평가 방법

학습률과 반복 횟수에 따른 모델 성능 변화 평가 방법	
검증 방법	학습 및 검증 데이터 셋으로 나누는 것입니다. 이를 위해 데이터를 무작위로 분할하여 일부는 학습 데이터로, 일부는 검증 데이터로 사용합니다. 학습 데이터를 이용하여 모델을 학습시키고, 검증 데이터를 이용하여 모델의 성능을 평가합니다.
성능 측정 방법	보통 학습률과 반복 횟수에 대한 그리드 서치나 랜덤 서치 등의 최적화 기법을 사용합니다. 검증 데이터를 이용하여 각 조합의 모델의 성능을 측정하고, 가장 좋은 조합을 선택합니다.
성능 평가 지표	평균 제곱 오차(MSE)나 결정 계수(R2) 등이 사용

 # 다양한 경사 하강법의 성능을 비교하고 평가하기 위해서는 일반적으로 다음과 같은 방법을 사용

성능 비교하고 평가하기 위한 일반적인 절차	
데이터셋 준비	- 사용하고자 하는 경사 하강법 알고리즘에 적합한 크기의 데이터셋을 준비 - 이 때, 훈련 세트와 테스트 세트를 분리하여 훈련 후 모델 성능을 평가
모델 학습	- 각각의 경사 하강법 알고리즘을 사용하여 모델을 학습 - 학습시키는 과정에서 학습률, 반복 횟수, 배치 크기 등의 하이퍼 파라미터를 조정 하.여 최적의 성능을 얻습니다.
모델 평가	- 각각의 모델에 대해 테스트 세트를 사용하여 예측 결과를 평가 - 이 때, 다양한 성능 지표를 사용하여 모델의 성능을 평가, 예를 들어서 평균 제곱 오차, 평균 제곱 절대 오차, R 제곱 값 등을 사용
성능 비교	- 각각의 모델의 성능을 비교하여 어떤 경사 하강법 알고리즘이 가장 우수한 성능을 보이는지 확인 - 이 때, 다양한 성능 지표를 종합적으로 고려하여 결정하는 것이 바람직하다.
하이퍼파라미터 조정	- 어떤 경사 하강법 알고리즘이 가장 우수한 성능을 보였다면, 해당 알고리즘의 하이퍼파라미터를 조정하여 성능을 더욱 개선할 수 있습니다. - 이 때, 적절한 하이퍼파리미터 값을 찾기 위해 그리드 탐색 등의 방법을 사용

그리드 탐색 (Grid Search) 소개

하이퍼파라미터를 조정하여 모델의 성능을 최적화하는 방법 중 하나입니다. 하이퍼파라미터란 모델링 과정에서 사람이 직접 지정해줘야 하는 매개변수로, 예를 들어 의사결정나무에서의 최대 깊이, SVM에서의 커널 종류 등이 있습니다.

그리드 탐색은 가능한 하이퍼파라미터의 조합을 모두 만들어내고 이를 순차적으로 모델 학습을 진행하여 가장 좋은 성능을 보이는 조합을 선택하는 방법입니다.

이 때 각 하이퍼파라미터의 조합을 만드는 과정에서는 가능한 모든 조합을 만드는 것이 일반적입니다. 하이퍼파라미터 조합이 많을수록 탐색 범위가 넓어지므로 모델 성능을 최적화하는 데 걸리는 시간이 늘어납니다.

그리드 탐색은 간단하고 직관적인 방법이지만, 하이퍼파라미터 공간이 클 경우에는 계산 비용이 크게 증가할 수 있으며 최적의 하이퍼파라미터를 찾지 못할 가능성도 있습니다. 이러한 문제를 해결하기 위해 랜덤 탐색(Random Search) 등의 방법도 사용됩니다.

경사 하강법이 사용되는 다양한 분야와 예시

분야	설명	예시
기계학습	경사 하강법은 기계 학습 알고리즘에서 최적화 문제를 해결하는 데 널리 사용	• 선형 회귀나 로지스틱 회귀 같은 분류 모델에서 파라미터를 조정하는 데 사용됩니다.
인공 신경망	신경망에서 역전파 알고리즘이 사용되는데, 이는 경사 하강법을 사용하여 가중치와 편향을 업데이트합니다.	• 역전파 알고리즘 (가중치 와 편향을 업데이트 용도)
자연어 처리	자연어 처리에서 최적화 문제를 해결하는 데에도 경사 하강법이 사용됩니다.	• 언어 모델링에서 파라미터를 조정
이미지 처리	이미지 분류나 객체 감지 같은 컴퓨터 비전 작업에서도 경사 하강법이 사용됩니다.	• CNN에서 학습된 모델의 파라미터를 최적화하는데 사용

경사 하강법이 사용되는 다양한 분야와 예시

분야	설명	예시
강화 학습	강화 학습에서도 경사 하강법이 사용됩니다.	• 강화 학습에서는 에이전트가 주어진 환경에서 행동을 수행하며 보상을 얻습니다. 경사 하강법은 이러한 보상에 기초하여 에이전트의 정책을 개선하는 데 사용됩니다.
행렬 분해	행렬 분해는 매우 큰 데이터셋을 처리할 때 유용합니다.	• 추천 시스템, 이미지 처리, 자연어 처리 등 다양한 분야에서 사용됩니다. 행렬 분해에서도 경사 하강법이 사용됩니다.
최적화 문제	경사 하강법은 최적화 문제를 해결하는 데에도 사용됩니다.	• 경제학에서 최적화 문제를 해결하는 데에 사용됩니다.

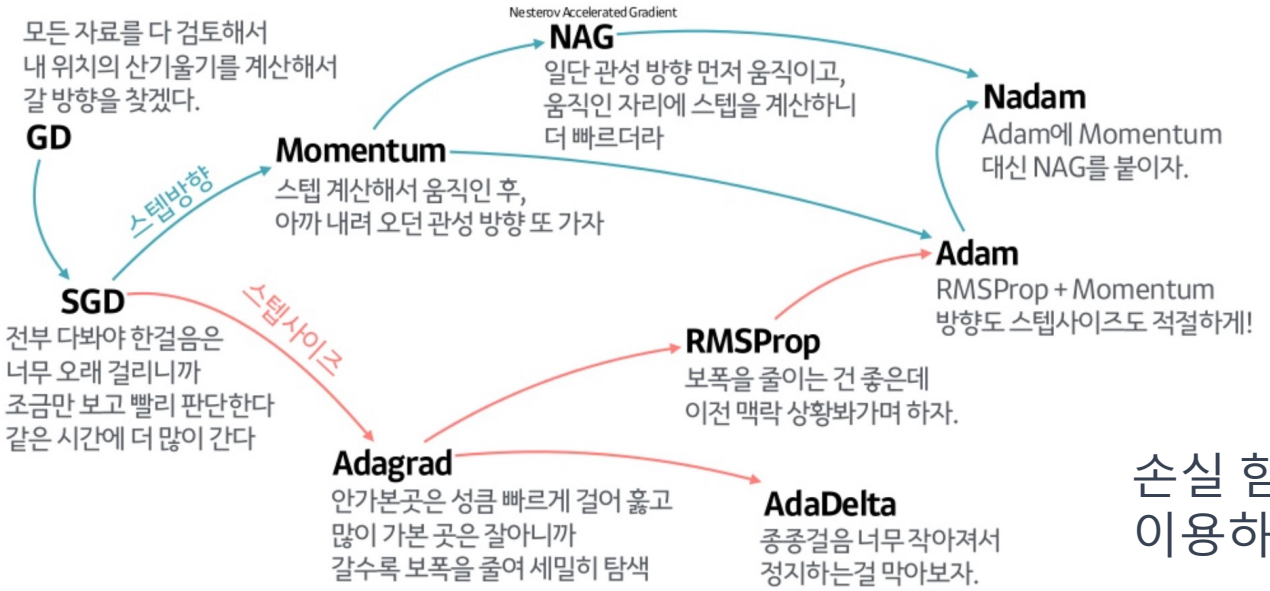
이 외에도, 경사 하강법은 다양한 분야에서 사용되고 있으며, 그 활용범위는 계속해서 확대되고 있습니다.

옵티마이저 소개

Infinyx
THINK LIFE SYNC AI

옵티마이저 소개

산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



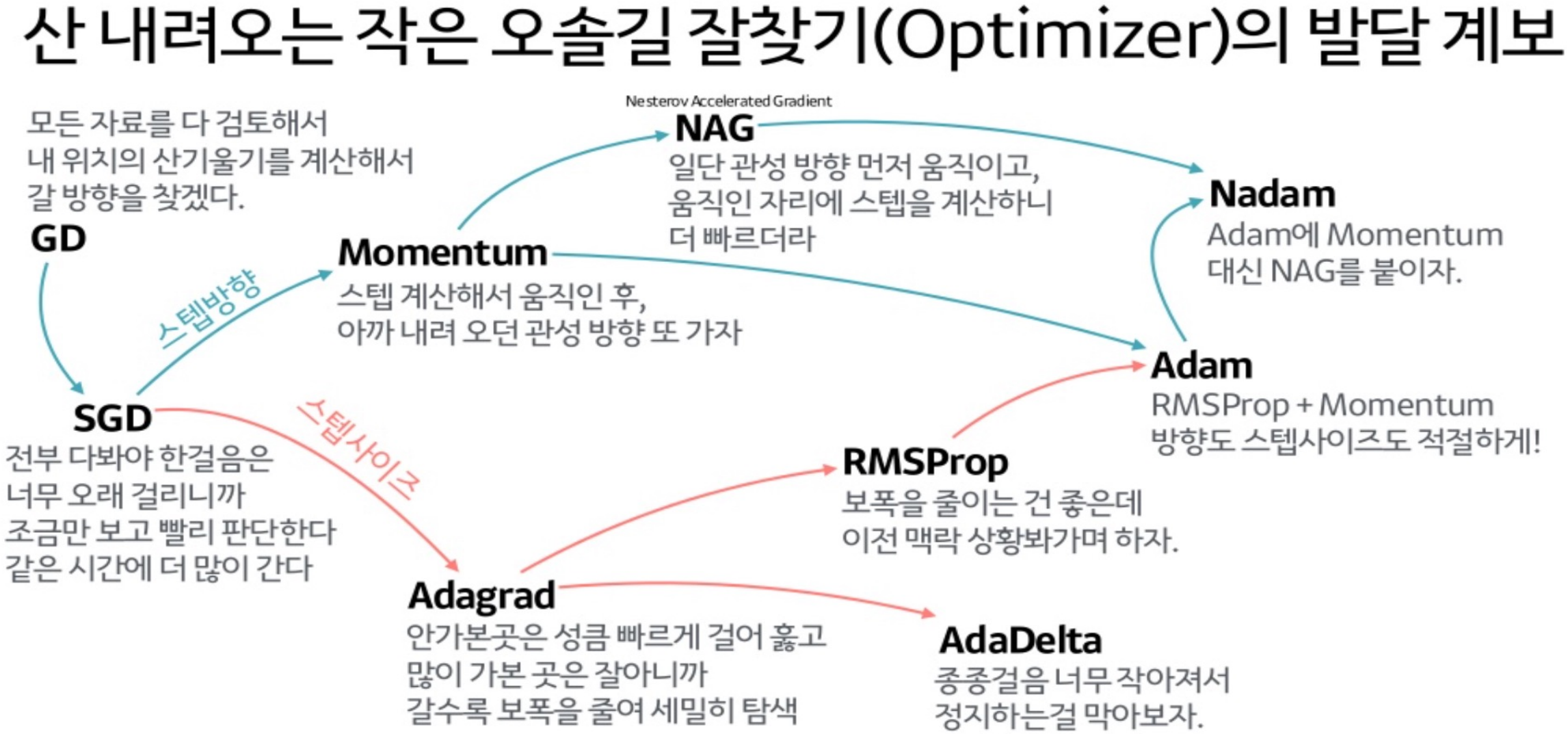
딥러닝 모델의 학습 과정에서 모델의 파라미터를 업데이트하는 알고리즘입니다.

즉, 모델이 학습 데이터에 대해 최적의 결과를 도출하기 위해 모델의 가중치(weight)와 편향(bias)을 조정하는데 사용됩니다.

손실 함수(loss function)에서 계산된 그래디언트(gradient)를 이용하여 모델의 파라미터를 업데이트합니다.

손실 함수는 모델의 예측 값과 실제 값의 차이를 계산하는 함수로, 이 값을 최소화하는 방향으로 모델의 파라미터를 조정합니다.

옵티마이저 종류 소개


산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보
모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
스텝사이즈
Adagrad
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색
RMSProp
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
AdaDelta
종종걸음 너무 작아져서
정지하는걸 막아보자.

 # Gradient Descent Optimization

가장 기본적인 최적화 알고리즘 중 하나로, 가중치를 조정할 때 매개변수에 대한 손실함수의 기울기(gradient)를 이용하여 최적화하는 방법입니다. 즉, 가중치 업데이트는 현재 가중치에서 기울기를 빼는 방식으로 이루어집니다.

종류	설명
Batch Gradient Descent	전체 데이터셋에 대해 기울기를 계산하고 가중치를 업데이트합니다. 즉, 전체 데이터셋을 한 번에 처리하므로 한 번에 메모리를 많이 사용하게 되고, 처리 시간이 오래 걸릴 수 있습니다. 하지만 전체 데이터셋에 대한 기울기를 계산하므로 최소값에 수렴하는 속도는 빠릅니다.
SGD	데이터 하나씩 기울기를 계산하고 가중치를 업데이트합니다. 즉, 데이터 한 개씩 처리하므로 전체 데이터셋보다 메모리 사용량이 적지만, 최소값에 수렴하는 속도가 느릴 수 있습니다. 또한, 경사 하강 방향이 매번 달라져서 최적 값에 수렴하는 과정에서 지그재그로 이동하는 현상이 발생할 수 있습니다.

 # Gradient Descent Optimization

종류	설명
Mini-batch Gradient Descent	전체 데이터셋의 일부(mini-batch)에 대해서만 기울기를 계산하고 가중치를 업데이트합니다. 즉, 전체 데이터셋보다 메모리 사용량이 적으면서, 한 개의 데이터를 처리하는 SGD보다 안정적으로 최적값에 수렴할 수 있습니다.
Gradient Descent Optimization	하이퍼파라미터인 learning rate는 가중치를 업데이트할 때 사용되는 step size로서, 학습률이 크면 최소값을 찾지 못하고 발산할 수 있고, 학습률이 작으면 최소값에 수렴하는 속도가 느려질 수 있습니다. 따라서 적절한 학습률을 선택하는 것이 중요합니다.

 # Momentum Optimization

Gradient Descent의 한계를 보완하기 위해 등장한 방법 중 하나입니다. 이전 기울기의 방향과 크기를 고려하여 새로운 기울기를 계산합니다. 이전 기울기의 방향이 현재 기울기와 일치하면 가중치를 더 크게 업데이트하고, 그렇지 않으면 더 작게 업데이트합니다.

예를 들어, 기울기가 대각선 방향으로 지속적으로 나오는 경우, 일반적인 Gradient Descent는 오랜 시간 동안 최적점에 수렴하지 못하고 지그재그로 움직이게 됩니다.

하지만 Momentum Optimization은 이전 기울기를 더해서 가중치 업데이트를 수행하기 때문에 이전 방향을 유지하면서 최적점에 빠르게 수렴할 수 있습니다.

이전 기울기와 현재 기울기의 비율을 조절하는 momentum 하이퍼파라미터를 설정할 수 있습니다. momentum 값이 0에 가까울수록 Gradient Descent와 유사해지며, 1에 가까울수록 이전 방향을 보존하는 정도가 높아집니다.

적절한 momentum 값을 설정하면 보다 빠르고 안정적인 학습을 할 수 있습니다.

 # Momentum Optimization

종류	설명
Standard Momentum	기본적인 Momentum Optimization으로 이전 기울기의 방향과 크기를 고려하여 가중치를 업데이트합니다.
NAG (Nesterov Accelerated Gradient)	Standard Momentum에 비해 더 빠른 수렴 속도를 가지는 방법입니다. 현재 위치에서 Momentum 방향으로 이동한 후, 이동한 위치에서 기울기를 계산하여 가중치를 업데이트합니다.
해비 볼 (Heavy-ball Momentum)	Standard Momentum에서 Momentum 방향의 속도를 감소시켜 overshooting(최적값을 지나쳐서 발산하는 현상)을 막는 방법입니다. 이전 기울기와 현재 기울기의 합과 차를 고려하여 가중치를 업데이트합니다.

 # AdaGrad Optimization

각각의 매개변수에 서로 다른 학습률을 적용하는 방식을 사용하여, 데이터 셋에서 매개변수에 대한 제곱된 그래디언트의 역사를 누적함으로써 각각의 매개변수에 대한 적응적인 학습률을 계산합니다.

기울기 제곱 값의 누적 합을 이용해 학습률을 조절하는 방식으로, 매개변수별로 학습률을 조절할 수 있는 방법입니다.

이전 기울기들의 제곱을 누적하여 학습률을 조절하므로 처음에는 크게 업데이트하다가 점차 학습률이 줄어들게 됩니다.

이러한 방식은 기울기의 크기가 큰 매개변수는 학습률이 감소하고, 기울기의 크기가 작은 매개변수는 학습률이 증가하는 경향을 보입니다.

 # AdaGrad Optimization 장단점, 최적화 기법 소개

매개변수의 전역 학습률을 조절하는 것이 아니라, 개별 매개변수의 학습률을 적절하게 조절할 수 있어, 매개변수의 스케일에 덜 민감해진다는 점이 있습니다.

하지만, 단점으로는 누적된 제곱 기울기 값이 계속해서 커져서 학습률이 점점 작아지는 문제가 있어, 학습이 오래될수록 업데이트가 매우 느려지는 경향이 있습니다.

이러한 단점을 보완하기 위해 RMSprop과 Adam이 등장하게 됩니다.

Adagrad 최적화 기법에는 다른 종류가 없습니다.
→ Adagrad는 하이퍼파라미터인 학습률(learning rate)을 매개변수마다 따로 설정하는 방식으로 동작하므로, 하나의 하이퍼파라미터만 설정하면 됩니다.

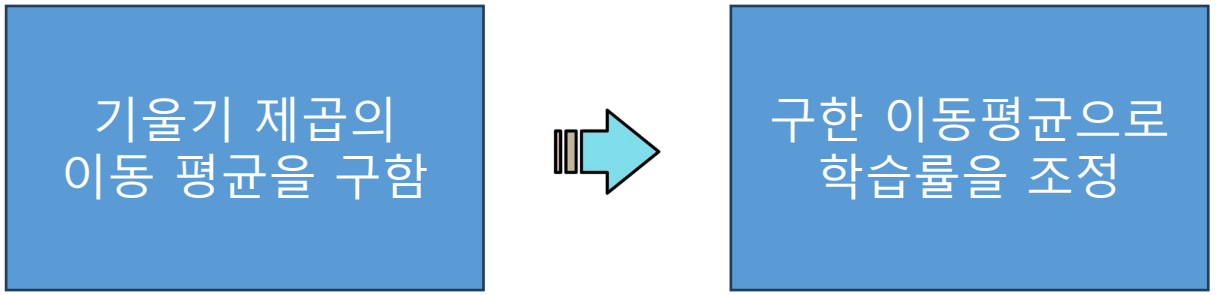
 # RMSprop Optimization

Adagrad의 단점을 보완한 방법 중 하나로, 기울기 제곱의 이동평균을 사용하여 학습률을 조절합니다.

이동평균을 구할 때 지수이동평균(Exponential Moving Average, EMA)을 사용합니다.

EMA는 이동평균을 구할 때 과거의 값들을 지수적으로 감소시키면서 현재 값을 계산하는 방식입니다.
이 때, 이동평균을 구하는 지수 가중치(decay rate)를 하이퍼파라미터로 설정할 수 있습니다.

 # RMSprop Optimization → 학습률 조절 방법



즉, RMSprop은 과거 기울기의 크기를 고려하여 적절한 학습률을 계산합니다. 이러한 방식으로 RMSprop은 Adagrad보다 빠르게 수렴하면서도 더 안정적으로 학습할 수 있습니다.

 # Adam

Momentum Optimization과 Adagrad Optimization의 아이디어를 결합한 옵티마이저입니다.

각 매개변수마다 적응적인 학습률을 사용하며, 이전 기울기의 지수 가중 이동 평균과 이전 기울기 제곱의 지수 가중 이동 평균을 계산하여 학습률을 조정합니다.

현재 시점의 기울기를 계산 → 이전 기울기의 지수 가중 이동 평균과 이전 기울기 제곱의 지수 가중 이동 평균을 계산합니다. → 학습률을 적응적(adaptive)으로 조정합니다. → 매개변수 업데이트

 # Adam

Momentum Optimization과 마찬가지로 이전 기울기의 방향과 크기를 고려하면서 매개변수를 업데이트하기 때문에, 경사면이 급격하게 변하는 지점에서 빠르게 최적점에 다가갈 수 있습니다.

또한, Adagrad Optimization처럼 각 매개변수마다 적응적인 학습률을 사용하기 때문에, 학습이 더욱 안정적으로 이루어질 수 있습니다.

Adam은 딥러닝 모델 학습에서 효과적인 옵티마이저 중 하나이며, 다양한 문제에 적용됩니다.

선형 회귀 모델의 학습에서 다양한 옵티마이저를 적용하여 결과를 비교 실습

❖ Auto MPG 데이터셋 활용 : 자동차 연비 예측을 위한 데이터 이용

```
# 다양한 옵티마이저를 적용하여 결과를 비교 실습
import pandas as pd
import requests
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import TensorDataset, DataLoader

# 파일 다운로드
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
filename = "auto-mpg.csv"

response = requests.get(url)
if response.status_code == 200:
    with open(filename, 'wb') as f:
        f.write(response.content)
    print("Downloaded successfully as", filename)
else:
    print("Failed to download:", response.status_code)

# CSV 파일을 DataFrame으로 읽기
df = pd.read_csv(filename, sep='\s+', header=None)
# sep='\s+'는 공백으로 구분된 데이터를 읽기 위한 옵션입니다.

# 열 이름 설정
df.columns = ["MPG", "Cylinders", "Displacement", "Horsepower", "Weight", "Acceleration", "Model_Year", "Origin", "Car_Name"]
print(df)
```

선형 회귀 모델의 학습에서 다양한 옵티마이저를 적용하여 결과를 비교 실습

❖ Auto MPG 데이터셋 활용 : 자동차 연비 예측을 위한 데이터 이용

```
x_data = df[["Cylinders", "Displacement", "Weight"]].values.astype(np.float32)
y_data = df["MPG"].values.astype(np.float32).reshape(-1, 1)

# 데이터 표준화
mean = x_data.mean(axis=0)
std = x_data.std(axis=0)
x_data_standardized = (x_data - mean) / std

# 텐서로 변환
x_tensor = torch.tensor(x_data_standardized)
y_tensor = torch.tensor(y_data)

# 데이터 로드 생성
dataset = TensorDataset( *tensors: x_tensor, y_tensor)
data_loader = DataLoader(dataset, batch_size=34, shuffle=True)
```

선형 회귀 모델의 학습에서 다양한 옵티마이저를 적용하여 결과를 비교 실습

❖ Auto MPG 데이터셋 활용 : 자동차 연비 예측을 위한 데이터 이용

```
# 선형 회귀 모델 정의
2 usages  new *
class LinearRegressionModel(nn.Module):
    new *
    def __init__(self, input_dim):
        super(LinearRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, out_features: 1)

    new *
    def forward(self, x):
        return self.linear(x)

# 모델 초기화
input_dim = x_data.shape[1]
model = LinearRegressionModel(input_dim)

num_epochs = 1000
learning_rate = 0.001

# 다양한 옵티마이저 설정
optimizers = {'SGD': optim.SGD(model.parameters(), lr=learning_rate),
              'Momentum': optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9),
              'Adagrad': optim.Adagrad(model.parameters(), lr=learning_rate),
              'RMSprop': optim.RMSprop(model.parameters(), lr=learning_rate),
              'Adam': optim.Adam(model.parameters(), lr=learning_rate)}

losses = {optimizer_name: [] for optimizer_name in optimizers.keys()}
```

선형 회귀 모델의 학습에서 다양한 옵티마이저를 적용하여 결과를 비교 실습

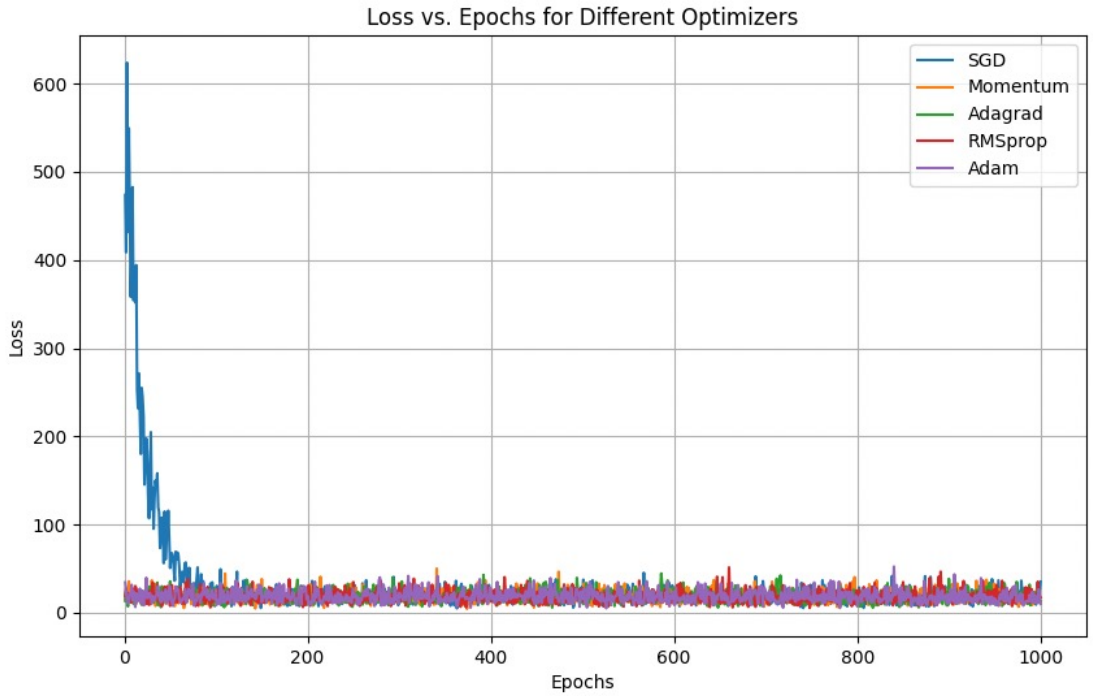
❖ Auto MPG 데이터셋 활용 : 자동차 연비 예측을 위한 데이터 이용

```
# 모델 학습
for optimizers_name, optimizer in optimizers.items() :
    criterion = nn.MSELoss()
    optimizer.zero_grad()

    for epoch in range(num_epochs) :
        for inputs, targets in data_loader:
            input = inputs
            target = targets
            optimizer.zero_grad()
            outputs = model(input)
            loss = criterion(outputs, target)
            loss.backward()
            optimizer.step()

        # 에폭마다 손실값 저장
        losses[optimizers_name].append(loss.item())

        if (epoch + 1) % 100 == 0:
            # print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss:.4f}')
            print(f"{optimizers_name} - Epoch [{epoch + 1}/{num_epochs}], Loss : {loss.item():.4f}")
```

선형 회귀 모델의 학습에서 다양한 옵티마이저를 적용하여 결과를 비교 실습

❖ Auto MPG 데이터셋 활용 : 자동차 연비 예측을 위한 데이터 이용

```
# 그래프로 손실값 시각화
plt.figure(figsize=(10, 6))
for optimizer_name, loss_values in losses.items():
    plt.plot( *args: range(1, num_epochs + 1), loss_values, label=optimizer_name)

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs for Different Optimizers')
plt.legend()
plt.grid(True)
plt.show()
```



옵티마이저의 특징과 장단점 정리 !!!

딥러닝 모델의 학습 과정에서 가중치(w)와 편향(b)의 값을 최적화하는데 사용되는 알고리즘입니다.
학습속도, 최적화 성능, 메모리 사용량 등의 측면에서 차이가 있으며, 각각의 특징과 장단점은 아래와 같습니다.

종류	특징과 장단점 정리
SGD	✓ 가장 기본적인 옵티마이저 ✓ 무작위로 선택한 샘플(미니배치)의 그래디언트를 이용해 파라미터를 갱신하는 방법 ✓ 속도가 빠르고 구현이 간단하지만, 수렴속도가 느리고 지역 최소값(local minimum)에 빠질 가능성이 있음
Momentum	✓ SGD와 비슷한 방식으로 가중치 갱신을 수행하지만, 그래디언트의 지수 이동 평균을 계산해 진동(oscillation)을 줄임 ✓ SGD에 비해 빠르게 수렴하며, 지역 최소값을 탈출하기 쉬움 ✓ 하지만, 모멘텀 값에 따라 최적값을 지나쳐서 수렴하는 overshooting 문제가 발생할 수 있음

 # 옵티마이저의 특징과 장단점 정리 !!!

종류	특징과 장단점 정리
Adagrad	✓ 학습이 진행됨에 따라 그래디언트의 크기에 따라 학습률(learning rate)을 조정해 가중치를 갱신하는 방법 ✓ 각 파라미터마다 개별적으로 학습률을 조정하기 때문에 수렴속도가 빠르고, 하이퍼파라미터의 튜닝이 필요 없음 ✓ 그러나, 학습이 진행됨에 따라 학습률이 감소해 더 이상 업데이트가 일어나지 않을 수 있음(overshooting)
RMSprop	✓ Adagrad의 단점을 보완한 방법 ✓ 학습률이 점점 줄어들도록 하는 대신, 지수 이동 평균을 사용하여 학습률을 유지함 ✓ 적응형 학습률 방법 중 하나로 SGD보다 빠르게 수렴하며, 불필요한 파라미터의 업데이트를 줄여줌

 # 옵티마이저의 특징과 장단점 정리 !!!

종류	특징과 장단점 정리
Adam	특징 ✓ 모멘텀과 RMSprop의 장점을 결합한 방법으로, 학습 속도가 빠르며 안정적인 수렴을 보장합니다. ✓ 미분 값이 0인 지점에서도 움직이도록 하기 위한 편향 보정을 수행합니다. ✓ 자동으로 학습률을 조정하며, 처음에는 큰 학습률로 시작하다가 이후에는 학습률을 점진적으로 감소시키므로 수렴 속도를 향상시킬 수 있습니다. 장점 ✓ RMSprop, Momentum의 장점을 결합하여 안정적이면서 빠른 수렴이 가능합니다. ✓ Gradient Descent에서 보여지는 최적점에서의 오실레이션 문제를 해결할 수 있습니다. ✓ 적응적인 learning rate를 제공하므로 빠르게 수렴할 수 있습니다. ✓ hyperparameter에 대한 민감도가 낮아서 하이퍼 파라미터를 튜닝하기 쉬운 편입니다. 단점 ✓ 일부 문제에서 다른 옵티마이저보다 성능이 떨어지는 경우가 있습니다. 또한, 일부 하이퍼 파라미터를 튜닝하지 않으면 성능이 떨어질 수 있습니다. ✓ 더욱 복잡한 문제에 대해서는 다른 옵티마이저와 비교했을 때 성능이 떨어질 수 있습니다.


감사합니다.