

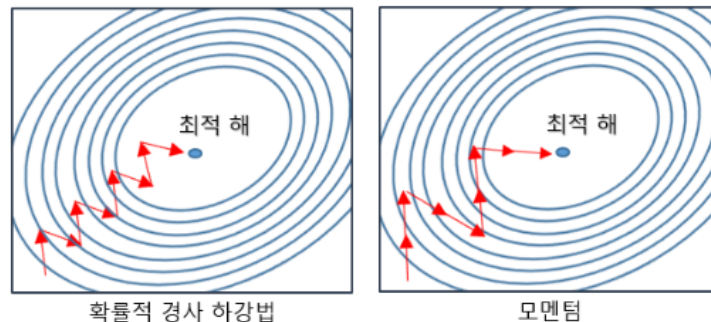
Momentum Project

20171620 문성찬

1. Momentum 최적화 알고리즘의 개요 및 동작원리

3. Momentum(모멘텀)

모멘텀(momentum)이란 단어는 관성, 탄력, 가속도라는 뜻입니다. 모멘텀 SGD는 경사 하강법에 관성을 더해 주는 것입니다. 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 가중치를 수정하기전 이전 수정 방향(+,-)를 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법입니다. 수정이 양(+) 방향, 음(-) 방향 순차적으로 일어나는 지그재그 현상이 줄어들고, 이전 이동 값을 고려하여 일정 비율만큼 다음 값을 결정하므로 관성의 효과를 낼 수 있습니다.



우선 아래 수식에서 α 는 Learning Rate, m 은 momentum 계수입니다.

$$\begin{aligned} V(t) &= m * V(t-1) - \alpha \frac{\partial}{\partial w} Cost(w) \\ W(t+1) &= W(t) + V(t) \end{aligned}$$

Momentum 방식은 말 그대로 Gradient Descent를 통해 이동하는 과정에 일종의 '관성'을 주는 것입니다.

현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식입니다.

경사하강법 알고리즘은 기울기 0인 점을 잘 탈출하지 못한다는 것 외에도 훈련이 너무 느리다는 단점이 있습니다.

이를 해결하기 위해서 보편적으로 사용되는 방법이 관성(momentum)을 적용하는 것입니다.

관성이란, 변수가 가던 방향으로 계속 가도록 하는 속도(velocity) 항을 추가하는 것입니다.

지금 상태의 기울기에도 당연히 영향을 받지만, 지금의 기울기는 가던 방향을 약간씩만 바꿔 주는 역할을 하게 됩니다.

바른 방향으로 가고 있다면 점점 더 속도가 빨라지게 되어 더 빨리 훈련이 될 수도 있고,

현재 기울기가 0인 안장점이라도 속도가 있으니 계속 이동해 안장점을 더 잘 탈출할 수 있게 됩니다.

새로운 속도는 이전 속도와 지수평균(exponential average)을 통해 계산됩니다.

학습속도처럼, 지수평균에 사용되는 모멘텀 상수(momentum rate)가 있습니다.

모멘텀 상수가 클수록 이전 속도를 더 따르게 됩니다.

모멘텀 상수는 0.5 정도로 시작해서 어느 정도 감소 추세가 안정화되면 0.9로 늘려 사용합니다.

다양한 안정화 기법이 나온 요즘에는 시작부터 0.9로 진행하기도 합니다.

같은 방향으로 많이 움직일수록 속도도 빨라지게 됩니다.

2. Tensorflow로 구현한 Momentum 동작 코드와 단위 테스트 코드

1) testTM_#1.py 실행 결과

testTM_#1.py에서는 unittest를 이용해 클래스 내부 함수들을 단위테스트 해주었습니다.

```
Ran 3 tests in 0.002s

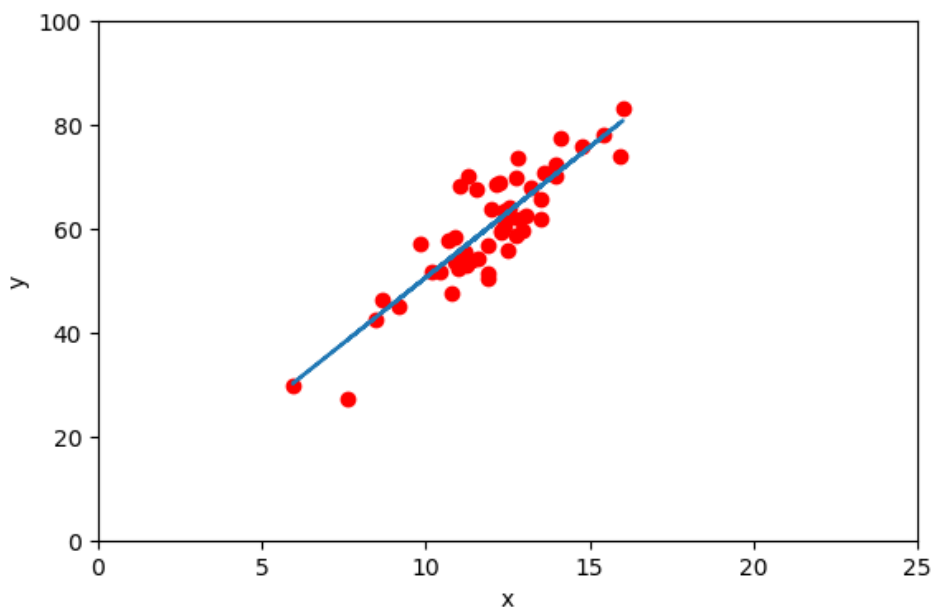
OK

Process finished with exit code 0
```

2) testTM_#2.py 실행 결과

testTM_#2.py에서는 구현한 momentum이 잘 작동되는지 간단한 난수 데이터를 통해 테스트했습니다.

```
Loss_data = [2324.1455    415.38824    63.510345   515.91016    750.10724    524.4789
184.04793    32.212822    64.82491    124.7607    119.888985   71.50212
 34.841053    29.62025    39.029713    43.074966    38.051094    31.284756
 28.538095    29.346313    30.499653    30.305082    29.307663    28.607632
 28.538542]
```



3) Source Code

- tensorflow_momentum.py (동작 코드)
- testTM_#1.py (단위 테스트 코드)
- testTM_#2.py (단위 테스트 코드)

3. Momentum 알고리즘의 구체화

Momentum 알고리즘과 경사하강법 알고리즘을 클래스화 시키고 기능별로 나누어 모듈화 작업해주었습니다.

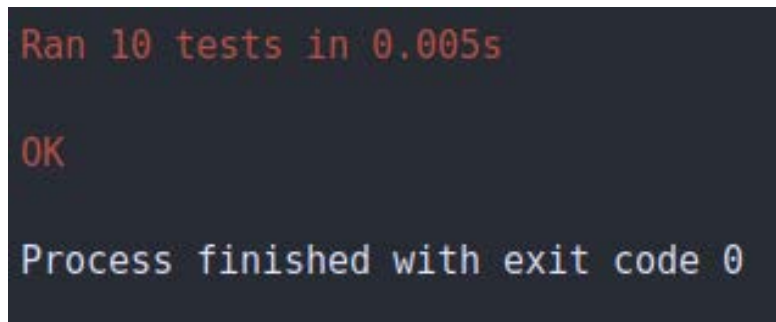
1) Source Code

- momentum.py

4. Momentum과 GD(경사하강법) 구체화 모듈의 단위 테스트 작업

unittest를 이용해 Momentum 클래스의 내부 함수들과 GD(경사하강법) 클래스의 내부 함수들을 단위테스트 해주었습니다.

1) testM.py 실행 결과



```
Ran 10 tests in 0.005s  
  
OK  
  
Process finished with exit code 0
```

2) Source Code

- testM.py (단위 테스트 코드)

5. Momentum 알고리즘 성능 검증

로젠브록 함수를 통해 두 알고리즘의 결과를 비교함으로써 성능을 검증했습니다.

1) checkMomentum_#1.py 실행 결과

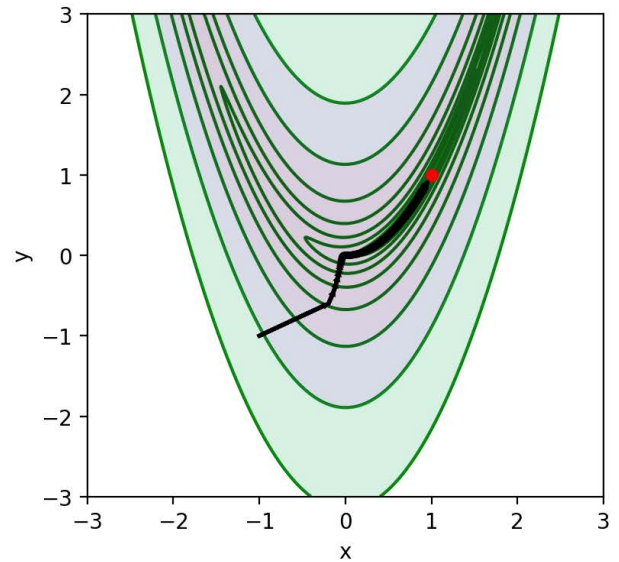
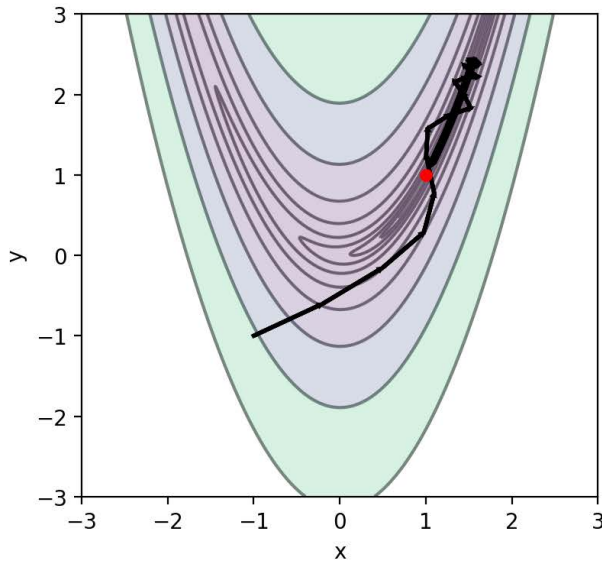
두 알고리즘 모두 시작값을 (-1, -1)로 설정하고 학습률도 0.001로 동일하게 설정한 후 성능 검증(비교)

```

< Momentum >
반복 수 : 1551
W = [1.064889, 1.134239]
dMSE = [0.023419, 0.049939]
-----
< GD >
반복 수 : 4299
W : [0.939765, 0.882909]
dMSE : [-0.026531, -0.049980]

Process finished with exit code 0

```



2) checkMomentum_#2.py 실행 결과

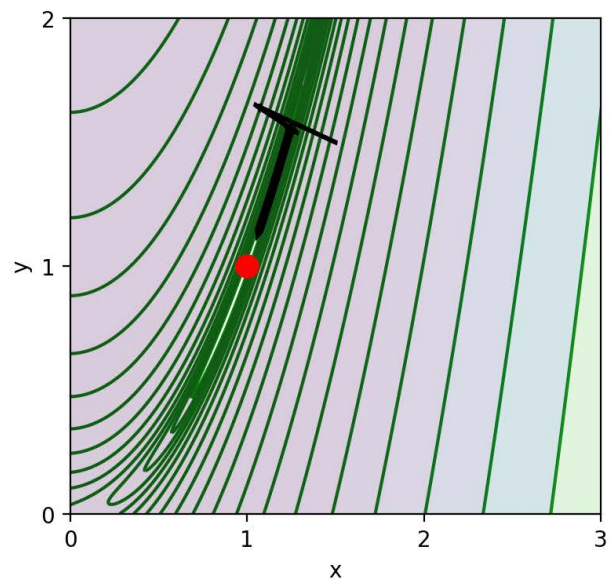
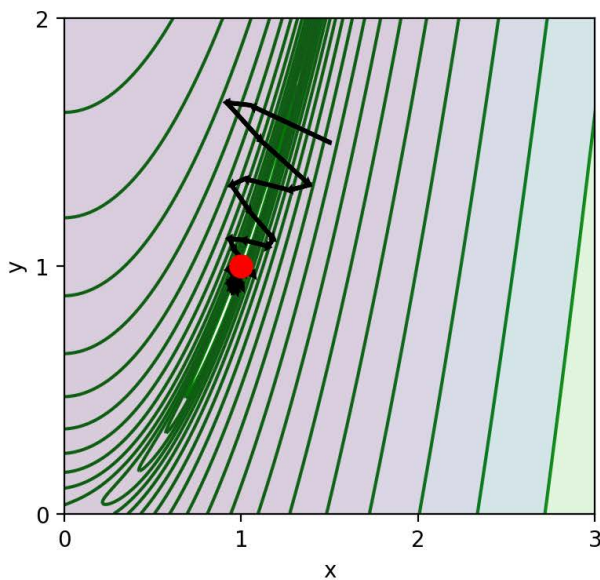
두 알고리즘 모두 시작값을 (1.5, 1.5) 로 설정하고 학습률도 0.001로 동일하게 설정한 후 성능 검증(비교)

```

< Momentum >
반복 수 : 79
W = [0.956447, 0.914671]
dMSE = [-0.041011, -0.024097]
-----
< GD >
반복 수 : 4114
W : [1.064964, 1.134398]
dMSE : [0.023434, 0.049999]

Process finished with exit code 0

```

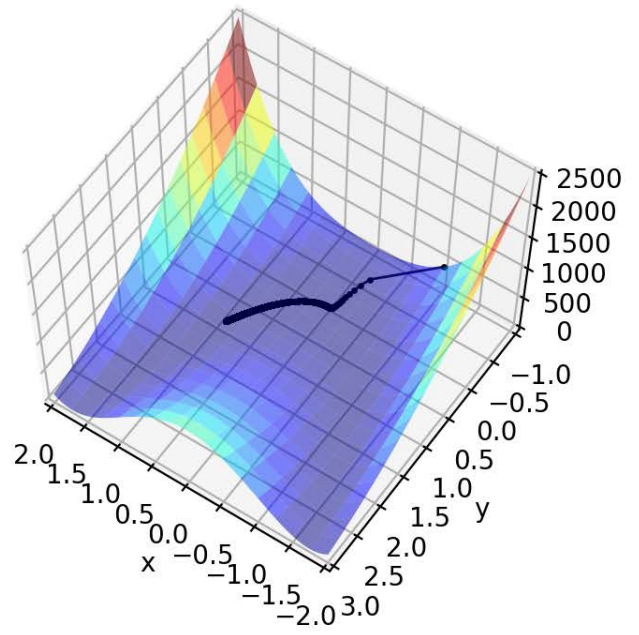
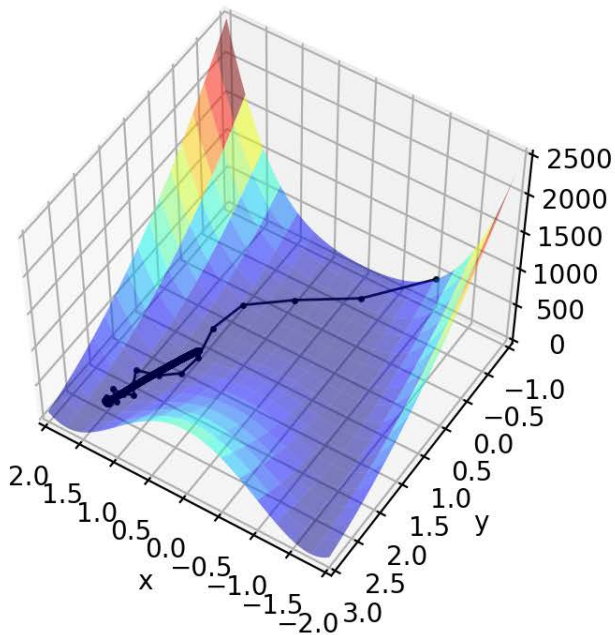
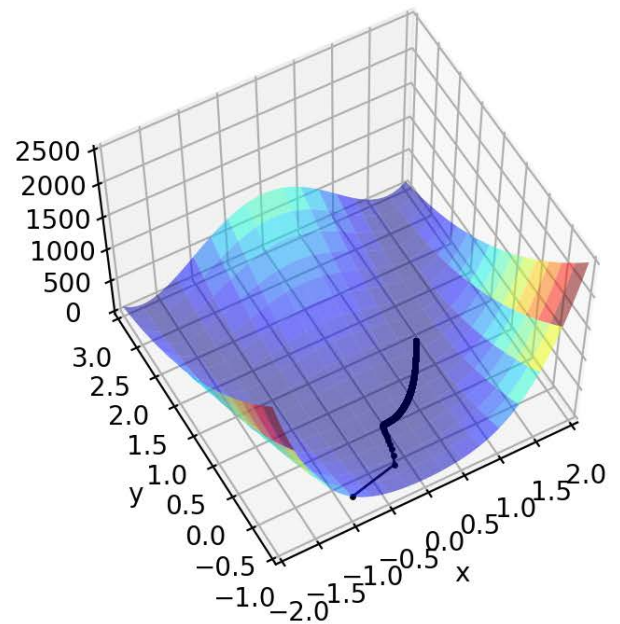
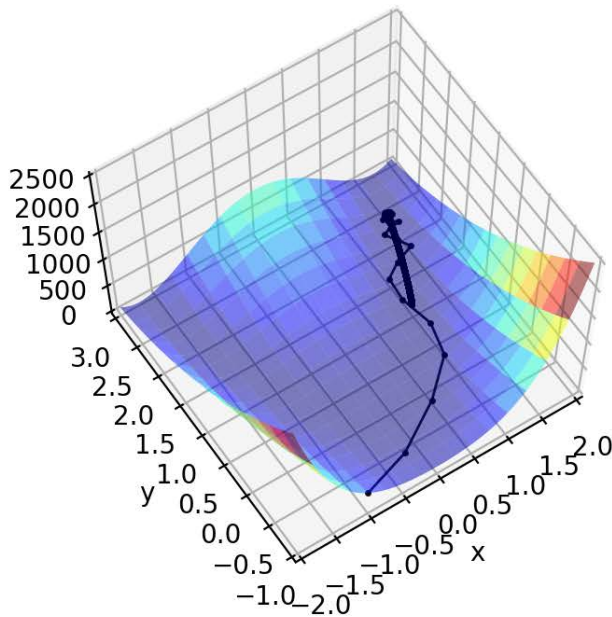


3) checkMomentum_#3.py 실행 결과

두 알고리즘 모두 시작값을 (-1, -1) 로 설정하고 학습률도 0.001로 동일하게 설정한 후
성능 검증 (checkMomentum_#1.py와 동일)
추가로 보다 자세한 비교를 위해 3차원 입체 그래프를 활용

```
< Momentum >
반복 수 : 1551
W = [1.064889, 1.134239]
dMSE = [0.023419, 0.049939]
-----
< GD >
반복 수 : 4299
W : [0.939765, 0.882909]
dMSE : [-0.026531, -0.049980]

Process finished with exit code 0
```



4) Source Code

- checkMomentum_#1.py
- checkMomentum_#2.py
- checkMomentum_#3.py

6. 소감

이번 프로젝트를 통해 모멘텀 알고리즘에 대해서 구체적으로 다루어 보았습니다. 기존에 배웠던 경사하강법과 비교해 더욱 빠르고 적은 횟수의 계산량으로 최적화하는 모멘텀의 알고리즘을 공부하면서 기존의 Gradient를 통해 이동하는 과정을 개선하기 위해 관성이라는 개념을 도입했다는 것에 대단함을 느꼈습니다.

또한 프로젝트 실습 과정에서 momentum을 tensorflow로도 구현해보고, 직접 numpy 레벨의 파이썬 패키지에서 구현해봄으로써, momentum에 개념에 대해 더욱 자세하게 배울 수 있었으며 직접 momentum 알고리즘을 구현해 보았다는 것 자체에 제 스스로에게 많은 뿌듯함을 느꼈습니다.

또한 각각의 알고리즘을 기능별로 나누어서 모듈화해보았더니 momentum과 경사하강법의 동작 과정을 다시 한번 집중있게 살펴 볼 수 있었습니다.

그리고 단위테스트를 통해 스스로 구현해 본 클래스나 함수가 잘 구현되었는지 확인해 봄으로써 unittest를 이용한 단위테스트의 중요성을 알 수 있었으며, momentum 알고리즘이 잘 동작하는지 알아보기 위해 특정 난수 데이터를 형성해 최적화를 진행해봄으로써 테스트에 대한 자신감도 얻을 수 있었습니다.

이후에 로젠브록 함수를 이용해 momentum 알고리즘과 경사하강법 알고리즘을 실행 결과를 통해 비교해봄으로써 이제까지 수업시간과 과제에서 실습해보았던 알고리즘 간의 비교를 스스로 진행해 볼 수 있었습니다.

진행 과정에서 약간의 어려움이 따르기도 했지만 악착같은 끈기와 노력으로 이겨낼 수 있었고, 평소 수업시간에 교수님이 말씀해주신 개념과 지식을 토대로 이번 프로젝트를 큰 어려움 없이 끝낼 수 있었습니다.

마지막으로 수업시간에 개념으로만 넘어갈 뻔했던 momentum과 확률적 경사하강법을 이렇게 프로젝트로 만나봄으로써, 추가로 다양한 방향으로 발전하는 최적화 알고리즘 계보를 찾아 볼 수 있었고, Adam, RMSProp 등 다양한 최적화 알고리즘도 배울 수 있었습니다.

감사합니다.