수치해석 프로젝트

20162820 김영민

모멘텀 최적화 알고리즘

목표 : 최적화 알고리즘을구체화하는 프로젝트

- 1. 선택한 최적화 알고리즘의 개요 및 동작원리(15점)

- 2. 선택한 최적화 알고리즘의 동작코드와 단위 테스트(20점)
  - 수업시간에소개된다양한파이썬패키지이용가능

- 3. 선택한 최적화알고리즘의구체화(40점)
  - 선택한 최적화 알고리즘 동작원리를 기능별로 나누어서 모듈화

- 4. 각 구체화 모듈의 단위테스트 작업 (15점)

- 5. 선택한 최적화알고리즘의검증(10점)
  -선택한 최적화 알고리즘의 성능검증을 로젠브록함수를 통해 진행

1) 모멘텀 최적화 알고리즘의 개요 및 동작원리

개요 : Momentum 은 Gradient descent(이하 GD) 기반의 optimization algorithm 이다. 그리고 '운동량'을 의미한다. 기울기에서 속도의 개념이 추가된 것으로 고등학교 물리 시간을 떠올려보면 자세히는 아니지만 지상의 마찰력 때문에 물체의 이동속도가 점점 감소한다고 배웠던 기억이 어렴풋이 기억이 난다. 속도가 크게 나올수록 기울기가 크게 업데이트 되어 확률적 경사하강법이 가지는 단점을 보완할 수 있다.

Momentum은 마찰력/공기저항 같은 것에 해당하며 기울기 업데이트 시 이 폭을 조절하는 역할을 한다. 이에 따라 속도 velocity가 변화한다.
모멘텀 알고리즘은 누적된 과거 그래디언트가 지향하고 있는 어떤 방향을 현재 그래디언트에 보정하려는 방식으로, 일종의 관성 또는 가속도처럼 생각하면 편하다.

동작원리 :

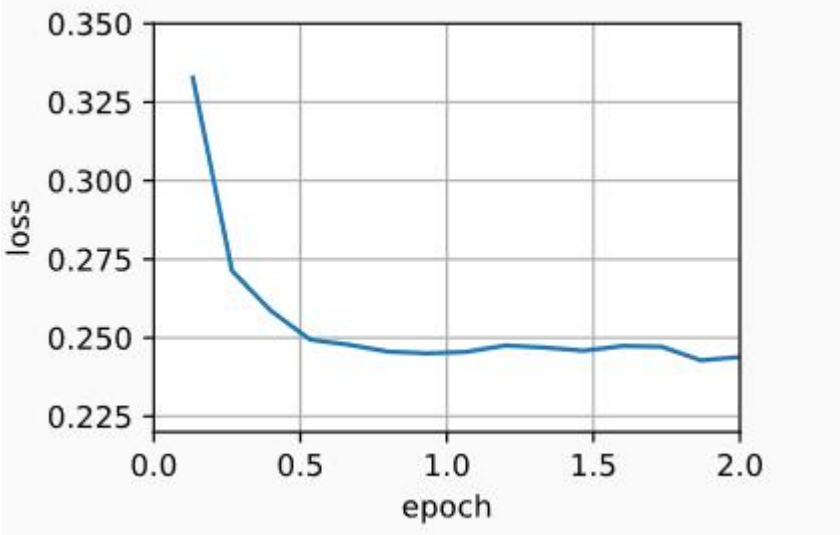
$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$
$$W \leftarrow W + v$$

이러한 수식을 이용하고 각 의미는 L : loss function value을 뜻하고 W : weights
η : learning rate , α : 가속도 같은 역할을 하는 hyper parameter, 0.9 등 1 이하의 값을 의미한다
파이썬 소스 코드는 v = m * v - learning_rate * gradient, weight[i] += v을 이용한다.
V는 물리에서 말하는 '속도' 라고 생각하면 편하다. 현재 Gradient Descent 알고리즘에 의해 계산된 이동 방향과 이동량을 이용해 속도를 변화시킨다. 그리고, 가중치를 그 '속도'만큼 변화시켜주고 있는 것을 볼 수 있다. 보통 에는 0.9등의 값을 준다고 한다.
이렇게 가중치를 업데이트하게 되면, 가중치는 현재 기울기()의 방향으로 가속 되어 업데이트된다. 따라서 최적값과 멀리 떨어져있을 때는 최적화에 가속이 붙고, 항이 이전의 속도를 기억하므로 최적화 이상으로 업데이트되어 다시 값이 튀었을 때 빠르게 잡아줄 수도 있다. 또한 기울기에 의한 갱신량이 0일 때, 항에 의해 이동하는 거리가 생기므로, Local Minimum에 빠질 확률을 줄여줄 수 있다.

2) 선택한 최적화 알고리즘의 동작코드와 단위 테스트

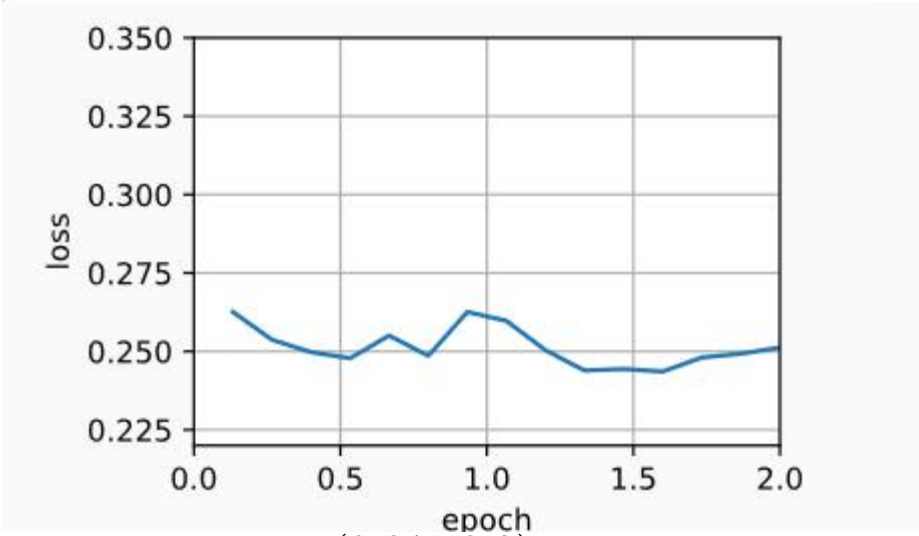
모멘텀 상수는 0.5 정도로 시작해서 어느 정도 감소 추세가 안정화되면 0.9로 늘려 사용한다. 다양한 안정화 기법이 나온 요즘에는 시작부터 0.9 로 진행한다고 한다. 같은 방향으로 많이 움직일 수록 속도도 빨라지게 된다. 즉, 모멘텀을 사용한다는 것은 학습 속도를 $1/1-\alpha$ 만큼으로 보정하는 것이라고 해석할 수 있습니다. α를 0.9로 한다는 것은, 기존 대비 약 10배 정도의 속도로 움직이도록 한다고 볼 수 있습니다.

```
def init_momentum_states(feature_dim):
    v_w = np.zeros((feature_dim, 1))
    v_b = np.zeros(1)
    return (v_w, v_b)

def sgd_momentum(params, states, hyperparams):
    for p, v in zip(params, states):
        v[:] = hyperparams['momentum'] * v + p.grad
        p[:] -= hyperparams['lr'] * v

def train_momentum(lr, momentum, num_epochs=2):
    d2l.train_ch11(sgd_momentum, init_momentum_states(feature_dim),
                   {'lr': lr, 'momentum': momentum}, data_iter,
                   feature_dim, num_epochs)

data_iter, feature_dim = d2l.get_data_ch11(batch_size=10)
train_momentum(0.02, 0.5)
```



train_momentum(0.02, 0.5)

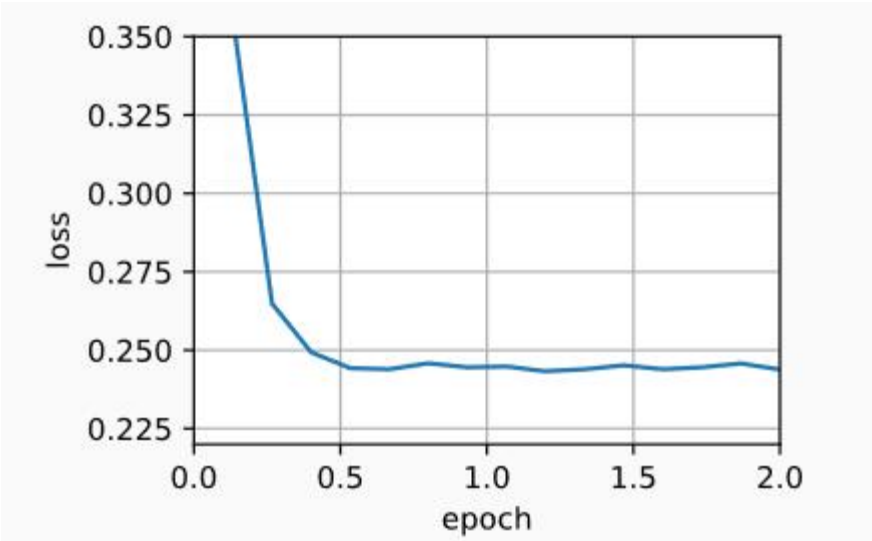
```python
def init_momentum_states(feature_dim):
    v_w = np.zeros((feature_dim, 1))
    v_b = np.zeros(1)
    return (v_w, v_b)

def sgd_momentum(params, states, hyperparams):
    for p, v in zip(params, states):
        v[:] = hyperparams['momentum'] * v + p.grad
        p[:] -= hyperparams['lr'] * v

def train_momentum(lr, momentum, num_epochs=2):
    d2l.train_ch11(sgd_momentum, init_momentum_states(feature_dim),
                   {'lr': lr, 'momentum': momentum}, data_iter,
                   feature_dim, num_epochs)

data_iter, feature_dim = d2l.get_data_ch11(batch_size=10)
train_momentum(0.01, 0.9)
```



train_momentum(0.01, 0.9)

학습 속도를 줄이면 매끄럽지 않은 최적화 문제의 문제를 더 많이 해결합니다. (0.005)로 설정하면 좋은 수렴 특성이 생성됩니다.

train_momentum(0.005, 0.9)



3. 선택한 최적화알고리즘의구체화

```
import numpy as np

def rosenbrock(x,y,a,b):
  return (a-x)**2 + b*(y-x**2)**2

def practice(a,b,c,d):
  moment = 0.9
  st = 0.12

  x = (rosenbrock(b[0]+0.15,b[1],c,d) - (rosenbrock(b[0]-0.0001,b[1],c,d))) / 0.0001
  y = (rosenbrock(b[0],b[1]+0.15,c,d) - (rosenbrock(b[0],b[1]-0.0001,c,d))) / 0.0001

  result = np.array([x,y], dtype=np.float64)
  a = moment * a - st * result
  b += st

  return a,b


a_1 = np.array([0,0], dtype=np.float64)
a_2 = np.array([-5,-5], dtype=np.float64)

a = 1
b = 100

for i in range(50):
  a_1,a_2 = practice(a_1,a_2,a,b)

  print(a_2[0],a_2[1])
```

def rosenbrock(x,y,a,b): // 이것을 통해 로젠브록으로 표현할 수 있게 return값을 설정해주었다. 로젠브록이란 수학적 최적화에서 최적화 알고리듬을 시험해볼 용도로 사용하는 비볼록함수로, 교재 pdf와 동일하게 return 값을 (a-x)**2 + b+(y-x**2)**2의 형식으로 주었다.

def practice(): // 이것은 본격적으로 모멘텀값 st = 학습률 for I range(학습횟수) 값을 변경해 로젠브록의 최적 값 (1,1)이 나오는 값을 찾아낸다. (1,1)이나 오기위해서는 계속적으로 학습률 , 학습횟수를 바꾸거나 a_2에 설정한 [-5,-5]같은 인덱스값을 바꾸어가며 설정할 수 있다. 그러면서 최적값이 나올때까지 올바른 값들을 찾아내면 된다.

for I in range(50): // 학습횟수를 50번을 해주어 지속적으로 로젠브록의 값들이 최적값에 도달해내는 과정이다 이과정을 50번해서 a_2[0]과, a_2[1]에 설정한 값이 최적값이 나올때까지 계속 반복문을 돌려주는 것이다.

각 구체화 모듈의 단위테스트 작업

1) moment=0.9 / 학습률 : 0.12 / 인덱스[-5 , -5] / 학습횟수 : 50번

```
import numpy as np

def rosenbrock(x,y,a,b):
  return (a-x)**2 + b*(y-x**2)**2

def practice(a,b,c,d):
  moment = 0.9
  st = 0.12

  x = (rosenbrock(b[0]+0.15,b[1],c,d) - (rosenbrock(b[0]-0.0001,b[1],c,d))) / 0.0001
  y = (rosenbrock(b[0],b[1]+0.15,c,d) - (rosenbrock(b[0],b[1]-0.0001,c,d))) / 0.0001

  result = np.array([x,y], dtype=np.float64)
  a = moment * a - st * result
  b += st

  return a,b


a_1 = np.array([0,0], dtype=np.float64)
a_2 = np.array([-5,-5], dtype=np.float64)

a = 1
b = 100

for i in range(50):
  a_1,a_2 = practice(a_1,a_2,a,b)

  print(a_2[0],a_2[1])
```

<결과>

```
-4.88 -4.88
-4.76 -4.76
-4.64 -4.64
-4.52 -4.52
-4.3999999999999995 -4.3999999999999995
-4.279999999999999 -4.279999999999999
-4.159999999999999 -4.159999999999999
-4.039999999999999 -4.039999999999999
-3.919999999999999 -3.919999999999999
-3.799999999999999 -3.799999999999999
-3.679999999999999 -3.679999999999999
-3.5599999999999987 -3.5599999999999987
-3.4399999999999986 -3.4399999999999986
-3.3199999999999985 -3.3199999999999985
-3.1999999999999984 -3.1999999999999984
-3.0799999999999983 -3.0799999999999983
-2.959999999999998 -2.959999999999998
-2.839999999999998 -2.839999999999998
-2.719999999999998 -2.719999999999998
-2.599999999999998 -2.599999999999998
-2.4799999999999978 -2.4799999999999978
-2.3599999999999977 -2.3599999999999977
-2.2399999999999975 -2.2399999999999975
-2.1199999999999974 -2.1199999999999974
-1.9999999999999973 -1.9999999999999973
-1.8799999999999972 -1.8799999999999972
-1.7599999999999971 -1.7599999999999971
-1.639999999999997 -1.639999999999997
-1.519999999999997 -1.519999999999997
-1.3999999999999968 -1.3999999999999968
-1.2799999999999967 -1.2799999999999967
-1.1599999999999966 -1.1599999999999966
-1.0399999999999965 -1.0399999999999965
-0.9199999999999965 -0.9199999999999965
-0.7999999999999965 -0.7999999999999965
-0.6799999999999965 -0.6799999999999965
-0.5599999999999965 -0.5599999999999965
-0.4399999999999965 -0.4399999999999965
-0.3199999999999965 -0.3199999999999965
-0.1999999999999651 -0.1999999999999651
-0.07999999999999652 -0.07999999999999652
0.04000000000000348 0.04000000000000348
0.16000000000000347 0.16000000000000347
0.28000000000000347 0.28000000000000347
0.40000000000000346 0.40000000000000346
0.5200000000000035 0.5200000000000035
0.6400000000000035 0.6400000000000035
0.7600000000000035 0.7600000000000035
0.8800000000000034 0.8800000000000034
1.0000000000000036 1.0000000000000036
```

2) moment=0.9 / 학습률 : 0.2 / 인덱스[-5 , -5] / 학습 횟수 : 30

```
import numpy as np

def rosenbrock(x,y,a,b):
  return (a-x)**2 + b*(y-x**2)**2

def practice(a,b,c,d):
  moment = 0.9
  st = 0.2

  x = (rosenbrock(b[0]+0.0001,b[1],c,d) - (rosenbrock(b[0]-0.0001,b[1],c,d))) / 0.01
  y = (rosenbrock(b[0],b[1]+0.001,c,d) - (rosenbrock(b[0],b[1]-0.0001,c,d))) / 0.01

  result = np.array([x,y], dtype=np.float64)
  a = moment * a - st * result
  b += st

  return a,b


a_1 = np.array([0,0], dtype=np.float64)
a_2 = np.array([-5,-5], dtype=np.float64)

a = 1
b = 100

for i in range(30):
  a_1,a_2 = practice(a_1,a_2,a,b)

  print(a_2[0],a_2[1])
```

<결과>

```
-4.8 -4.8
-4.6 -4.6
-4.3999999999999995 -4.3999999999999995
-4.199999999999999 -4.199999999999999
-3.999999999999999 -3.999999999999999
-3.799999999999999 -3.799999999999999
-3.5999999999999988 -3.5999999999999988
-3.3999999999999986 -3.3999999999999986
-3.1999999999999984 -3.1999999999999984
-2.9999999999999982 -2.9999999999999982
-2.799999999999998 -2.799999999999998
-2.599999999999998 -2.599999999999998
-2.3999999999999977 -2.3999999999999977
-2.1999999999999975 -2.1999999999999975
-1.9999999999999976 -1.9999999999999976
-1.7999999999999976 -1.7999999999999976
-1.5999999999999976 -1.5999999999999976
-1.3999999999999977 -1.3999999999999977
-1.1999999999999977 -1.1999999999999977
-0.9999999999999978 -0.9999999999999978
-0.7999999999999978 -0.7999999999999978
-0.5999999999999979 -0.5999999999999979
-0.39999999999999786 -0.39999999999999786
-0.19999999999999785 -0.19999999999999785
2.1649348980190553e-15 2.1649348980190553e-15
0.20000000000000218 0.20000000000000218
0.4000000000000022 0.4000000000000022
0.6000000000000022 0.6000000000000022
0.8000000000000023 0.8000000000000023
1.0000000000000022 1.0000000000000022
```

3) moment=0.9 / 학습률 : 0.06 / 인덱스[-5 , -5] / 학습 횟수 : 100

```
import numpy as np

def rosenbrock(x,y,a,b):
  return (a-x)**2 + b*(y-x**2)**2

def practice(a,b,c,d):
  moment = 0.9
  st = 0.06

  x = (rosenbrock(b[0]+0.15,b[1],c,d) - (rosenbrock(b[0]-0.0001,b[1],c,d))) / 0.0001
  y = (rosenbrock(b[0],b[1]+0.15,c,d) - (rosenbrock(b[0],b[1]-0.0001,c,d))) / 0.0001

  result = np.array([x,y], dtype=np.float64)
  a = moment * a - st * result
  b += st

  return a,b


a_1 = np.array([0,0], dtype=np.float64)
a_2 = np.array([-5,-5], dtype=np.float64)

a = 1
b = 100

for i in range(100):
  a_1,a_2 = practice(a_1,a_2,a,b)

  print(a_2[0],a_2[1])
```

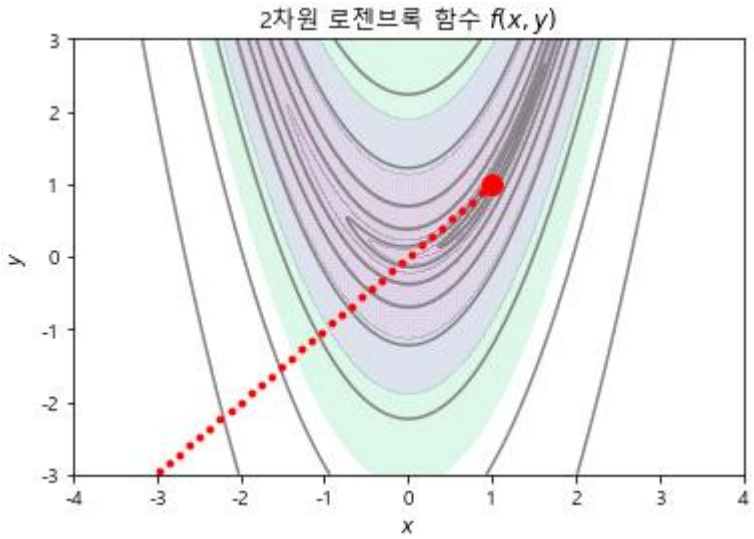
<결과>

```
-1.5200000000000004 -1.5200000000000004
-1.4600000000000004 -1.4600000000000004
-1.4000000000000004 -1.4000000000000004
-1.3400000000000039 -1.3400000000000039
-1.2800000000000038 -1.2800000000000038
-1.2200000000000037 -1.2200000000000037
-1.1600000000000037 -1.1600000000000037
-1.1000000000000036 -1.1000000000000036
-1.0400000000000036 -1.0400000000000036
-0.9800000000000035 -0.9800000000000035
-0.9200000000000035 -0.9200000000000035
-0.8600000000000034 -0.8600000000000034
-0.8000000000000034 -0.8000000000000034
-0.7400000000000033 -0.7400000000000033
-0.6800000000000033 -0.6800000000000033
-0.6200000000000032 -0.6200000000000032
-0.5600000000000032 -0.5600000000000032
-0.5000000000000031 -0.5000000000000031
-0.4400000000000031 -0.4400000000000031
-0.3800000000000031 -0.3800000000000031
-0.3200000000000031 -0.3200000000000031
-0.2600000000000031 -0.2600000000000031
-0.2000000000000312 -0.2000000000000312
-0.1400000000000312 -0.1400000000000312
-0.0800000000000312 -0.0800000000000312
-0.0200000000003126 -0.0200000000000003126
0.03999999999999687 0.03999999999999687
0.09999999999999687 0.09999999999999687
0.15999999999999687 0.15999999999999687
0.21999999999999686 0.21999999999999686
0.27999999999999686 0.27999999999999686
0.33999999999999686 0.33999999999999686
0.39999999999999686 0.39999999999999686
0.45999999999999686 0.45999999999999686
0.5199999999999969 0.5199999999999969
0.5799999999999997 0.5799999999999997
0.6399999999999997 0.6399999999999997
0.6999999999999971 0.6999999999999971
0.7599999999999971 0.7599999999999971
0.8199999999999972 0.8199999999999972
0.8799999999999972 0.8799999999999972
0.9399999999999973 0.9399999999999973
0.9999999999999973 0.9999999999999973
```

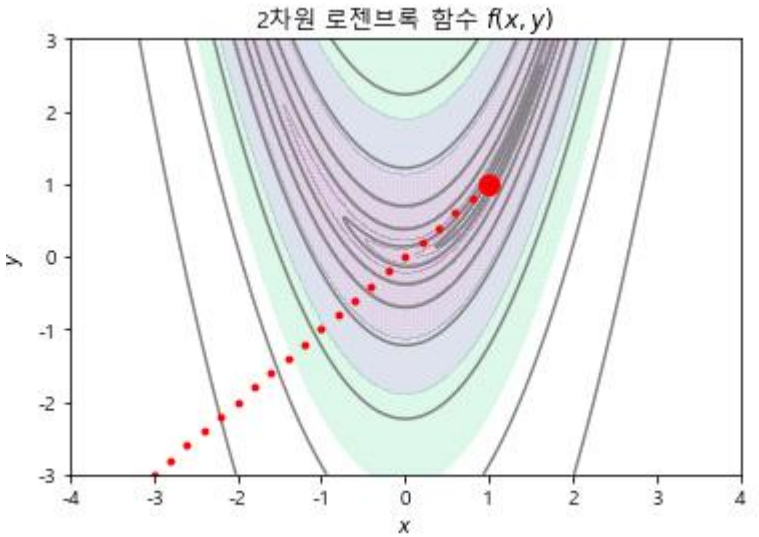
5. 선택한 최적화 알고리즘의 검증

설정 : (1,1) plt.plot size = 10, 모멘텀결과값 plt.plot size = 3

1) moment=0.9 / 학습률 : 0.12 / 인덱스[-5 , -5] / 학습횟수 : 50번



2) moment=0.9 / 학습률 : 0.2 / 인덱스[-5 , -5] / 학습 횟수 : 30번



3) moment=0.9 / 학습률 : 0.06 / 인덱스[-5 , -5] / 학습 횟수 : 100