국민대학교 2020 수치해석

기말 프로젝트

< Momentum 최적화 알고리즘 구체화 >

김찬미 [20191574]

소프트웨어융합대학

목차

1 모멘텀의 개요 및 동작 원리

1.1 모멘텀 개요

모멘텀(momentum) 이란 단어의 뜻은 '운동량'을 의미한다. 즉, 기울기에서 속도의 개념이 추가 되었다고 보면 된다. 모멘텀을 설명하는 쉬운 방법으로는 물리적으로 비유하는 방법이 있다. 예를 들어, 공이 굴러갈 때 지상의 마찰력 때문에 공의 속도가 점점 감소하는 것을 다들 경험 해봤을 것이다. 운동량이 없으면 공은 원하는 최종 목적지에 도달하지 못한다. 이를 보완하기 위해서 알고리즘에 운동량을 추가하는 방법이 필요하다. 이 방법은 운동량이 크게 나올수록 기울기가 크게 업데이트 되어 확률적 경사하강법이 가지는 단점을 보완할 수 있다. 모멘텀은 마찰력 혹은 공기저항 같은 것에 해당하며 기울기 업데이트 시 이 폭을 조절하는 역할을 하고, 그에 따라 속도(velocity)가 변화한다.

1.2 모멘텀 동작 원리

모멘텀의 동작 원리는 SGD(확률적 경사 하강법)과는 차이가 있다. SGD 식에서는 현재의 기울기 만을 이용했다면, 모멘텀은 이전 시간의 기울기까지도 고려하여 weight를 계속 업데이트 시켜줘야 한다. 이에 대한 식은 다음과 같다.

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - v_{ij}^{(t)}$$

식 1.1

이 때 모멘텀을 나타내는 $v_{ij}^{(t)}$ 는 다음과 같이 계산된다.

$$v_{ij}^{(t)} = \beta v_{ij}^{(t-1)} + \eta \frac{\partial L}{\partial w_{ij}^{(t)}}$$

식 1.2

이 식에서 $\beta \in [0, 1]$ 는 일반적으로 0.9로 설정되는 hyperparameter로써 이전 기울기와 현재 기울기 간의 영향력을 조절해 준다. 모멘텀의 이러한 업데이트 방식은 이전 시간에 A라는 방향으로 움직이고 있던 물체를 B라는 방향으로 움직이도록 변경하여도 일정 부분은 계속 A방향으로 움직이려고 하는 힘의 작용을 나타낸 것이다.

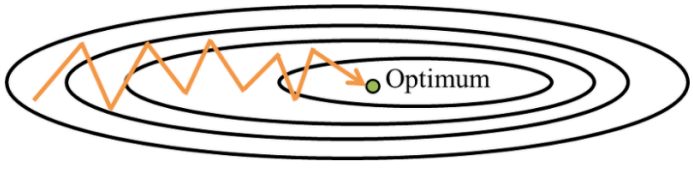


그림 1.1

위의 그림은 기본적인 SGD를 통한 최적화 과정이다. 경사 하강법은 기본적으로 'learning rate'의 크기만큼 기울기의 방향이 이동하기 때문에 최적점의 방향으로 곧장 이동하는 것이 아니라, 그림 1.1과 같이 진동하며 이동하게 된다. 모멘텀을 사용하는 경우 이와 같은 비효율적인 이동을 최소화 할 수 있다.

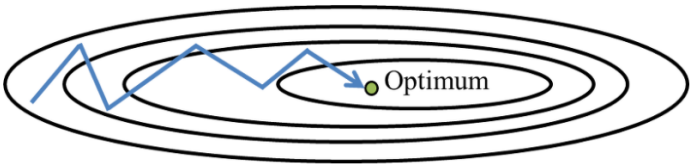


그림 1.2

그림 1.2는 모멘텀의 개념을 이용한 최적화 과정이다. 위의 그림을 보면 알 수 있듯이 진동을 하면서도 모든 진동 방향에는 최적점으로 이동하고자 하는 직선 방향의 이동이 존재한다. 모멘텀은 이전 시간의 이동 방향을 저장하기 때문에 모멘텀에는 직선 방향으로 이동하려는 성질 또한 강하게 저장된다. 이러한 작용에 의해 그림 1.2의 최적화 방식은 그림 1.1의 방법보다 더 빠르게 최적점을 찾아갈 수 있다.

2 모멘텀 방식의 알고리즘 코드

2.1 모멘텀의 동작 코드

methods.py

```python
import sys
import numpy as np

def __update_learning_rate__(lr, beta1, beta2):
    return lr * (np.sqrt(1 - beta2)/(1 - beta1))

def __update__(x, lr, new_value):
    return x - lr * new_value

def __weighted_average__(rho, grad_weight_aver, gradient):
    return rho*grad_weight_aver + (1-rho)*gradient

def __moment__(beta, current_gradient, gradient):
    return beta * current_gradient + gradient

def __reciprocal_sqrt__(val):
    epsilon = sys.float_info.epsilon
    return 1 / np.sqrt(np.array(val + epsilon, dtype=np.float))
```

< methods.py >

Momentum.py

```python
from . import methods

class Momentum:
    def __init__(self, lr=0.0013, momentum=0.1):
        self.lr = lr
        self.momentum = momentum
        self.gradient_moment = 0

    def update(self, x, gradient):
        # 기울기 구하기
        self.gradient_moment = methods.__moment__(self.momentum, self.gradient_moment, gradient)
        # 새로운 좌표 구하기
        new_x = methods.__update__(x, self.lr, self.gradient_moment)
        return new_x
```

< Momentum.py >

2.2 모멘텀 최적화 알고리즘 코드

Optimization.py

```python
import numpy as np
import copy

class Optimization:
    def __init__(self, optimizer, funct, early_stop=0.00001, iter_max=100):
        self.funct = funct
        self.early_stop = early_stop
        self.iter_max = iter_max

        self.optimizer_list = list()

        for i in range(self.funct.dim):
            self.optimizer_list.append(copy.copy(optimizer))

    def optimize(self, init):
        val_prev = np.array(init, dtype=np.float)
        val_new = np.array(init, dtype=np.float)
        iter_X = list()
        iter_Y = list()

        iter_X.append(init)
        iter_Y.append(self.funct.get_value(init))

        iter_cnt = 0
        for iter_cnt in range(self.iter_max):

            gradient = self.funct.get_diff_value(val_prev)
            for i in range(len(self.optimizer_list)):
                val_new[i] = self.optimizer_list[i].update(val_prev[i], gradient[i])
            iter_X.append(val_new.copy())
            iter_Y.append(self.funct.get_value(val_new))
            diff = abs(self.funct.get_value(val_new) - self.funct.get_value(val_prev))

            if diff < self.early_stop or np.isnan(diff):
                break
            else:
                val_prev = val_new.copy()

        return iter_X[-1], iter_X, iter_Y, iter_cnt
```

< Optimization.py >

2.3 구체화 모듈의 단위 테스트 코드

unittest.py

```python
import tensorflow as tf
import numpy as np
import Momentum.Optimization as Optimization
from Momentum.drawing import *


class Model(object):
    def __init__(self, a, b, init):
        self.x = tf.Variable(float(init[0]))
        self.y = tf.Variable(float(init[1]))
        self.a = tf.Variable(float(a))
        self.b = tf.Variable(float(b))

    def __call__(self):
        return (self.a - self.x) ** 2 + self.b * (self.y - self.x ** 2) ** 2

def test_uniitest(optimizer, f, init):
    Opt = Optimization.Optimization(optimizer=optimizer, funct=f)
    result, iter_X, iter_Y, iter_cnt = Opt.optimize(init)
    print(type(optimizer).__name__, " ==> ", f"position(x,y): {result}, z: {iter_Y[-1]:.3f}, iteration: {iter_cnt}")
    show(f, iter_X, iter_Y, iter_cnt)
```

< unittest.py >

2.4 로젠브록 함수를 통한 알고리즘 검증 코드

rosenbrock.py

```python
class Rosenbrock:
    def __init__(self, a=1, b=100):
        self.a = a
        self.b = b
        self.dim = 2
    def get_value(self, X):
        return (self.a-X[0])**2 + self.b*(X[1]-X[0]**2)**2

    def get_diff_value(self, X):
        x_diff_value = 2*X[0] - self.a*2 - 4 * self.b * (X[1] - X[0] ** 2) * X[0]
        y_diff_value = self.b*(2*X[1] - 2*X[0]**2)
        return [x_diff_value, y_diff_value]
```

< rosenbrock.py >

3 코드 실행 결과 화면

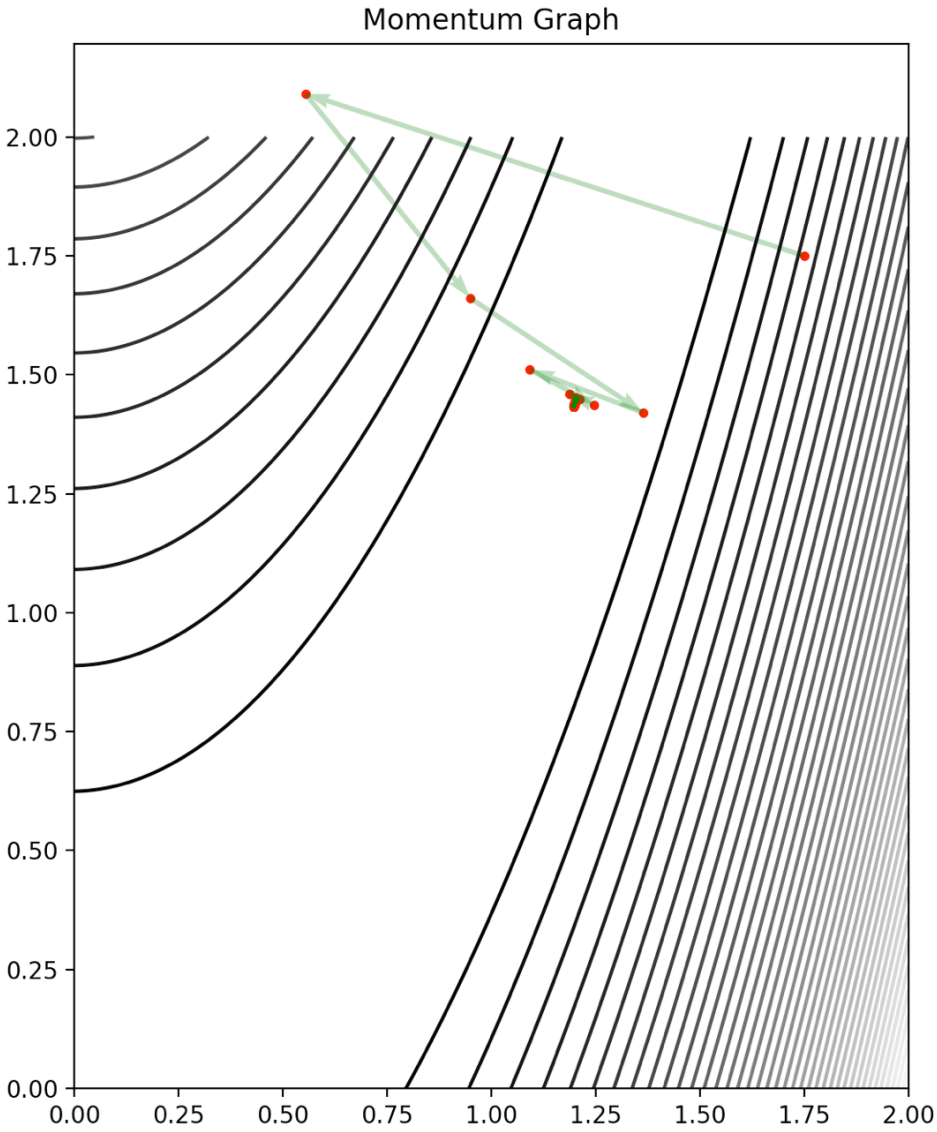
3.1 모멘텀 그래프 그리기

drawing.py

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

def show(function, iter_X, iter_z, iter_count):
    iter_x = np.empty(0)
    iter_y = np.empty(0)
    x = np.linspace(0, 2, 250)
    y = np.linspace(0, 2, 250)
    X_, Y_ = np.meshgrid(x, y)
    Z_ = np.zeros((250,250))
    for i in range(250):
        for j in range(250):
            Z_[i,j] = function.get_value([X_[i,j], Y_[i,j]])
    for X in iter_X:
        iter_x = np.append(iter_x, X[0])
        iter_y = np.append(iter_y, X[1])


    angle_x = iter_x[1:] - iter_x[:-1]
    angle_y = iter_y[1:] - iter_y[:-1]

    fig = plt.figure(figsize=(16, 8))

    ax = fig.add_subplot(1, 2, 2)
    ax.set_title('Momentum Graph')
    ax.scatter(iter_x, iter_y, color='r', marker='.')
    ax.contour(X_, Y_, Z_, 50, cmap='gist_gray')
    ax.quiver(iter_x[:-1], iter_y[:-1], angle_x, angle_y, scale_units='xy', angles='xy', scale=1, color='g', alpha=.3)

    plt.show()
```

< drawing.py >

3.2 출력 화면 결과



4 소감

사실 기말 프로젝트를 처음 시도해보려고 했을 땐 정말 막막했습니다. 너무 생소한 분야의 개념이어서 열심히 구글링도하고 주변 사람들에게 자문도 구하여 여러 시도 끝에 나름의 방법으로 성공한 것 같습니다. 처음엔 점들이 뒤죽박죽 생기는 등 여러 문제를 겪었지만 포기하지 않고 계속해서 수정해 나간 결과 이번 기말 프로젝트를 무사히 완료하였기에 스스로 굉장히 뿌듯합니다. 수업시간에 배운 머신러닝의 한 발짝 더 나아가게 된 것 같아 좋습니다. 비록 이번엔 비대면 수업이라 교수님께 여러 질문 등 교류를 하지 못해 아쉬움이 남지만 이번 학기에 배운 것을 토대로 한 단계 성장한 것 같습니다. 한 학기 동안 수고 많으셨고 감사합니다 ☺