AMSGrad

این الگوریتم بر پایه الگوریتم Adam بنا شده است و از ایده اصلاح شده شدن نسبت به مشکل عدم همگرایی Adam بهره می برد.

درAdam ، میانگین مربعات گرادیان‌ها (متغیر دومی) در محاسبه میزان بهبود وزن‌ها استفاده می‌شود. اما این موجودیت ممکن است در برخی مواقع رشد بیش از حدی داشته باشد که باعث عدم همگرایی می‌شود AMSGrad .با این مشکل مواجه نمی‌شود چون از میانگین مربعات گرادیان ها با حفظ آنچه که تا کنون بیشتر بوده است استفاده می‌کند.

مراحل AMSGrad :

1- محاسبه گرادیان و وزن‌های آپدیت شده

2-محاسبه میانگین مربعات گرادیان ها براساس گرادیان های فعلی

3- مقایسه این میانگین با یک مقدار ذخیره شده به نام V_t و به روزرسانی V_t به این صورت که بیشترین مقدار را انتخاب می کند (پس از روند آماری وزن دار)

4- محاسبه نرخ یاد گیری براساس مقادیر به روزرسانی شده

5- به روزرسانی وزن‌ها با استفاده از نرخ یادگیری محاسبه شده

AMSGrad در برخی موارد نسبت به Adam عملکرد بهتری دارد و مشکلات همگرایی را کاهش می‌دهد .

کد پیاده سازی AMSGrad :

صفحه بعد

```
import numpy as np

class AMSGradOptimizer:
    def __init__(self, learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-
8):
        self.learning_rate = learning_rate
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.m = None
        self.v = None
        self.v_hat = None
        self.t = 0

    def update(self, params, grads):
        if self.m is None:
            self.m = [np.zeros_like(param) for param in params]
            self.v = [np.zeros_like(param) for param in params]
            self.v_hat = [np.zeros_like(param) for param in params]

        self.t += 1

        for i in range(len(params)):
            self.m[i] = self.beta1 * self.m[i] + (1 - self.beta1) * grads[i]
            self.v[i] = self.beta2 * self.v[i] + (1 - self.beta2) * (grads[i] **
2)
            self.v_hat[i] = np.maximum(self.v_hat[i], self.v[i])
            m_hat = self.m[i] / (1 - self.beta1 ** self.t)
            v_hat = self.v_hat[i] / (1 - self.beta2 ** self.t)
            params[i] -= self.learning_rate * m_hat / (np.sqrt(v_hat) +
self.epsilon)

def objective_function(x):
    return 1 / x

def gradient_function(x):
    return -1 / (x ** 2)

initial_params = [10.0]
learning_rate = 0.1

optimizer = AMSGradOptimizer(learning_rate)

num_iterations = 100
```