RMSProp 算法

1. 产生背景

在 AdaGrad 算法中，当学习率在迭代早期下降较快且解依然不佳时，AdaGrad 算法在迭代后去由于学习率过小，可能较难找到一个有用的解。

2. 算法介绍

不同于 AdaGrad 算法中状态变量S_t是截至时间步 t 所有小批量随机梯度g_t按原始平方和，RMSProp 苏纳法将这些梯度按元素平方做指数加权移动平均，即：
给定超参数$0 \leq \gamma \leq 1$,RMSProp 算法在时间步 t>0 计算：

$$\boldsymbol{s}_t \leftarrow \gamma \boldsymbol{s}_{t-1} + (1-\gamma)\boldsymbol{g}_t \odot \boldsymbol{g}_t$$

和 AdaGrad 算法一样，RMSProp 算法将目标函数自变量中每个严肃的学习率按元素运算重新调整，然后更新自变量：

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$$

其中η是学习率，ϵ是维持数值稳定性的较小常熟，如10^{-6}。因为 RMSProp 算法的状态变量S_t是对平方项$\boldsymbol{g}_t \odot \boldsymbol{g}_t$ 的指数加权移动平均，所以可以看作是最近$\frac{1}{1-\gamma}$ 个时间步的小批量随机梯度平方的加权平均。这样就可以使得每个元素的学习率在迭代过程中不再一直降低（或不变）。
代码实现：

```
%matplotlib inline
import d2lzh as d2l
import math
from mxnet import nd

def rmsprop_2d(x1, x2, s1, s2):
    g1, g2, eps = 0.2 * x1, 4 * x2, 1e-6
    s1 = gamma * s1 + (1 - gamma) * g1 ** 2
    s2 = gamma * s2 + (1 - gamma) * g2 ** 2
    x1 -= eta / math.sqrt(s1 + eps) * g1
    x2 -= eta / math.sqrt(s2 + eps) * g2
    return x1, x2, s1, s2

def f_2d(x1, x2):
    return 0.1 * x1 ** 2 + 2 * x2 ** 2

eta, gamma = 0.4, 0.9
d2l.show_trace_2d(f_2d, d2l.train_2d(rmsprop_2d))
```

在同样学习率下，RMSProp 算法可以更快逼近最优解。

3. 特点

RMSProp 算法和 AdaGrad 算法的不同在于，RMSProp 算法使用了小批量随机梯度按元素平方的指数加权移动平均来调整学习率