An overview of gradient descent optimization algorithms

梯度下降法:

目的：最小化模型的目标函数$J(\theta)$
效果:对于凸函数可以达到最小值，对于非凸函数可以达到局部最小值

Batch gradient descent

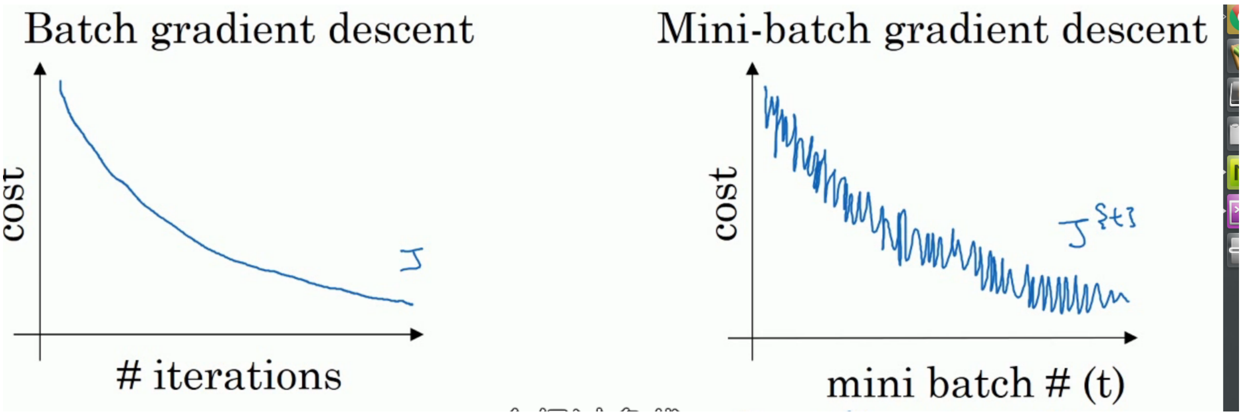
$\theta = \theta - \eta \nabla_\theta J(\theta)$
每次都使用全部数据更新。
缺点：每次更新的代价太大
η为学习率

mini-batch gradient descent

$\theta = \theta - \eta \nabla_\theta J(\theta; x^{i:i+n}; y^{i:i+n})$
每次都使用一个batch_size(n) 的数据进行更新。
通常n选择为50 到256

Stochastic gradient descent

$\theta = \theta - \eta \nabla_\theta J(\theta; x^i; y^i)$
每次只用一对数据进行更新，
缺点：容易陷入局部最小值
if mini-batch size =1 => stochastic gradient descent
if min-batch size = m(训练集总数) => batch gradient descent
如何选择min-batch size :
如果训练集较小(<2000)，使用batch
如果训练集较大，使用mini-batch,大小为64-512.使用 2 的幂次，代码运行速度更快
也可以把mini-batch size 当做一个超参数，试验几个不同的值，找到使cost funtion下降最快的mini batch size

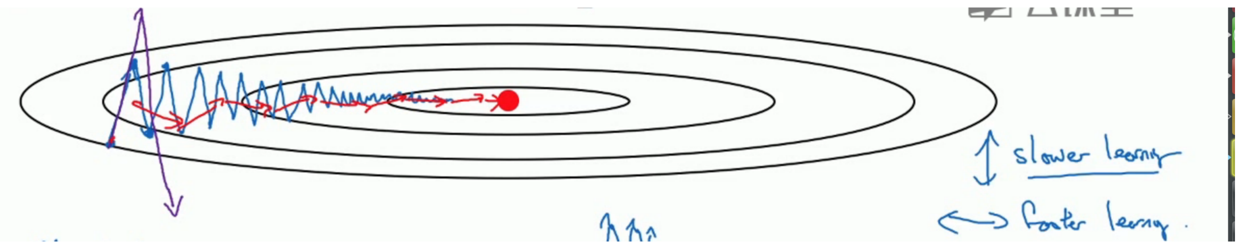


Challenges

- 选择学习率的大小
- 每个轮训练中学习率大小的变化，不同epoch 的学习率的变化， 训练停止的threshold
- 数据稀疏或者feature 出现频率差异大时，可能不想同时更新全部的参数，而是对频率小的feature进行更大的更新
- 如何避免陷入局部最优点

Gradient descent optimization algorithms

Momentum

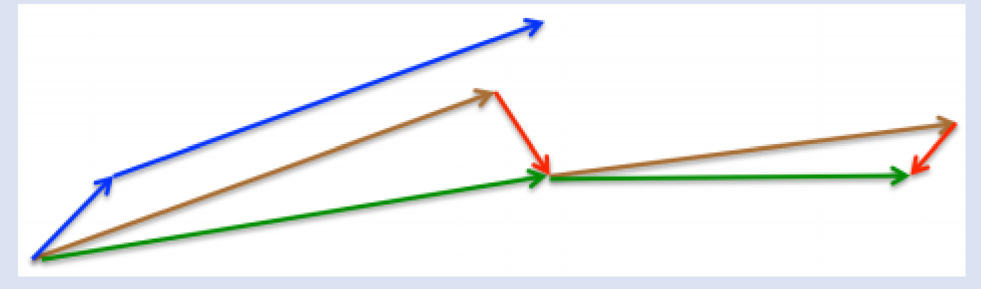


$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$
$\theta = \theta - v_t$
γ通常选0.9

在grdient descent 过程中，每次计算结果的梯度值变化太大，如果选取大的learning rate，可能结果无法收敛。所以只能使用小的learning rate。使用滑动平均值代替原本的梯度，每次的结果更平滑。不会出现剧烈波动，所以可以使用大的学习率，从而更快的收敛。
把梯度递减算法看成一个球从一个碗的边缘滚下去，算出的∇J等可以算为其下滚的加速度，加入了momentum可以看做是为其加了一个摩擦力。

Nesterov accelerated gradient

$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
$\theta = \theta - v_t$



Momentum:蓝色的线：先计算当前的梯度$\eta \nabla_\theta J(\theta)$，然后往“惯性”的方向$\gamma v_{t-1}$
NAG：先往“惯性”的方向γv_{t-1}（棕色），再往当前的梯度的方向$\eta \nabla_\theta J(\theta - \gamma v_{t-1})$（红）
可以看做为一个聪明的小球，能注意到它将要去哪，并且在上坡再次向上倾斜时小球应该进行减速。
对RNN效果显著

Adagrad

之前算法更新时每个参数的学习率都是一样的大小，adagrad 每个参数的学习率都是不同的
对稀疏参数进行大幅更新和对频繁参数进行小幅更新
SGD:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \nabla_{\theta_t} J(\theta_{t,i})$$

Adagrad:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla_{\theta_t} J(\theta_{t,i})$$

$G_{t,ii} = G_{t-1,ii} + \nabla_{\theta_t} J(\theta_{t,i})^2$
（之前的所有梯度的平方的和）
用G对学习率进行修正。因为G是正的，所以所有参数的学习率都是递减的。出现次数多的参数减得更快。
ϵ用于避免分母为0.一般是1e-8
缺点：
到最后学习率会变得太低

Adadelta

将梯度用g表示
$g_{ti} = \nabla_{\theta_t} J(\theta_{t,i})$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

将不断累积的梯度平方用滑动平均代替（跟上面的momentum类似)
$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$

RMSprop

基本与Adadelta相同
Hinton建议γ 0为.9, η 为 0.001.

Adaptive Moment Estimation (Adam)

最常用的优化方法
将momentum和RMSprop方法结合

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t可以看做是对梯度均值的估计，v_t可以看做是对方差的估计
因为是非中心化的，所以需要进行修正

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

β_1 usually 0.9.β_2 usually 0.999.ϵdoesn’t matter but recommend 1e-8

AdaMax

$$u_t = \beta_2 u_{t-1} + (1-\beta_2) g_t^{\infty}$$

发现用无穷范数代替二范数结果更稳定

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} m_t$$

Nadam (Nesterov-accelerated Adaptive Moment Estimation)

将Adam 与NAG结合
将Adam 的公式合并：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} (\beta_1 \hat{m}_{t-1} + \frac{(1 - \beta_1 g_t)}{1 - \beta_1^t})$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} (\beta_1 \hat{m}_t + \frac{(1 - \beta_1 g_t)}{1 - \beta_1^t})$$

只是将$\hat{m}_{t-1}$换为$\hat{m}_t$

Visualize

Visualize demo

summarize

SGD->Adagrad->Adadelta==RMSprop
RMSprop + Momentum = Adam
Adam +NAG = Nadam
Adam Best!