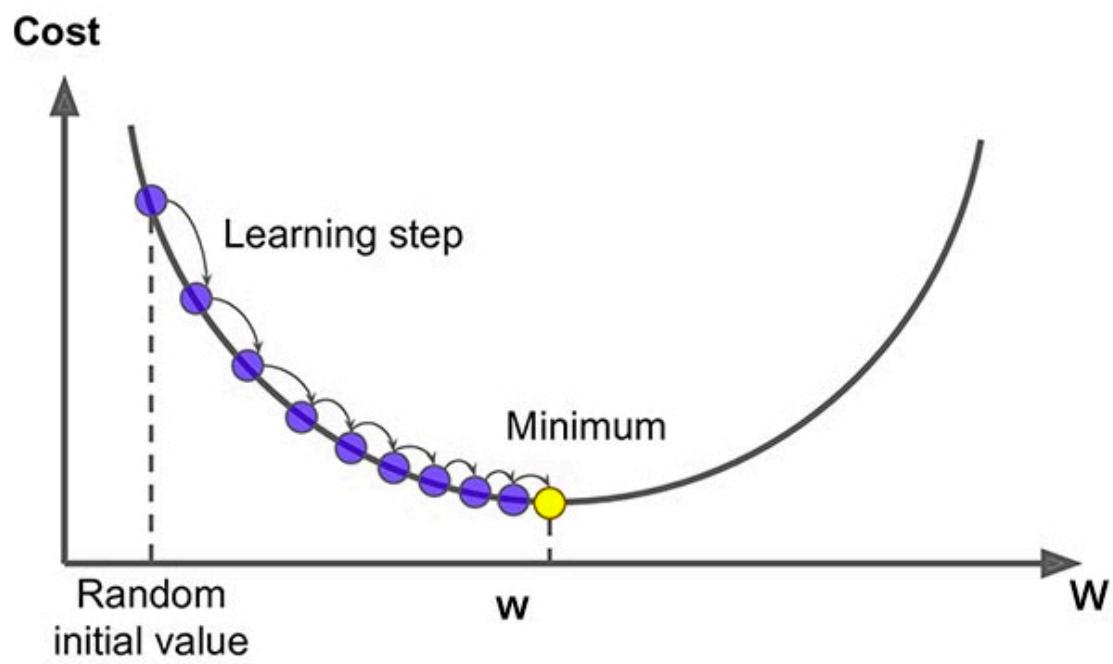


梯度下降优化算法



- 梯度下降法变体
 - 批处理梯度下降
 - 随机梯度下降法
 - 迷你批处理梯度下降法
- 面临的挑战
- 常用的梯度下降法
 - Momentum
 - Nesterov
 - Adagrad
 - AdaDelta
 - RMSprop
 - Adam
- 如何选择
- 小技巧

梯度下降优化算法

- 梯度下降法是训练神经网络最常用的优化算法
- 梯度下降法（英语：Gradient descent）是一个一阶最优化算法，通常也称为最速下降法。要使用梯度下降法找到一个函数的局部极小值，必须向函数上当前点对应梯度（或者是近似梯度）的反方向的规定步长距离点进行迭代搜索
- 梯度下降方法基于以下的观察：如果实值函数 $f(x)$ 在 a 点处可微且有定义，那么函数 $f(x)$ 在点 a 沿着梯度相反的方向 $-\nabla f(a)$ 下降最快

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$$

偏导数

- For a multivariable function, like $f(x, y) = x^2y$, computing partial derivatives looks something like this:

$$\frac{\partial f}{\partial x} = \underbrace{\frac{\partial}{\partial x} x^2}_{\text{Treating } y \text{ as constant;}} y = 2xy$$

Treat y as constant;
take derivative.

$$\frac{\partial f}{\partial y} = \underbrace{\frac{\partial}{\partial y} x^2}_{\text{Treating } x \text{ as constant;}} y = x^2 \cdot 1$$

Treat x as constant;
take derivative.

梯度

- The gradient of a scalar-valued multivariable function $f(x, y, \dots)$, denoted ∇f , packages all its partial derivative information into a vector:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix}$$

In particular, this means ∇f is a vector-valued function.

If $f(x, y) = x^2 - xy$, which of the following represents ∇f ?

Choose 1 answer:

(A)
$$\begin{bmatrix} 2x - x \\ x^2 - y \end{bmatrix}$$

(B)
$$\begin{bmatrix} 2x - y \\ -x \end{bmatrix}$$

If $f(x, y) = x^2 - xy$, which of the following represents ∇f ?

Choose 1 answer:

(A)
$$\begin{bmatrix} 2x - x \\ x^2 - y \end{bmatrix}$$

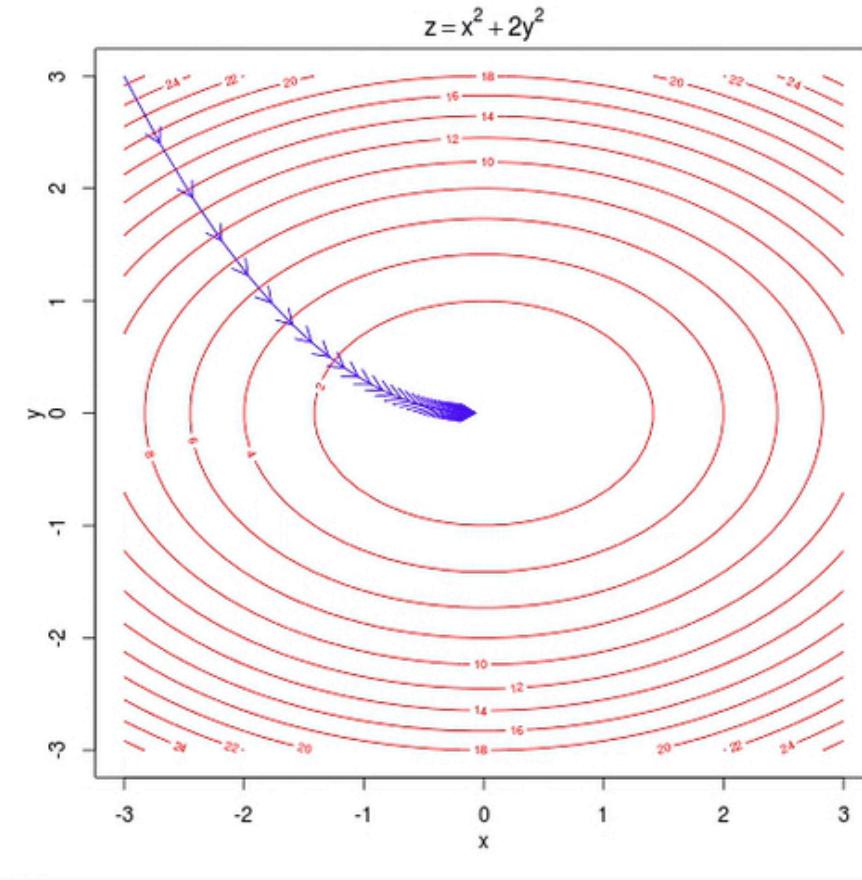
(B)
$$\begin{bmatrix} 2x - y \\ -x \end{bmatrix}$$

Check

[Hide explanation.]

$$\nabla f(x, y) = \left[\begin{array}{c} \frac{\partial}{\partial x} (x^2 - xy) \\ \frac{\partial}{\partial y} (x^2 - xy) \end{array} \right] = \left[\begin{array}{c} 2x - y \\ -x \end{array} \right]$$

梯度下降优化算法



$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$$

批处理梯度下降

```
for i in range(nb_epochs):
    sum_grad = 0
    for x, y in data:
        grad = gradient(loss_function, x, y, params)
        sum_grad += grad
    avg_grad = sum_grad / len(data)
    params = params - learning_rate * avg_grad
```

批处理梯度下降

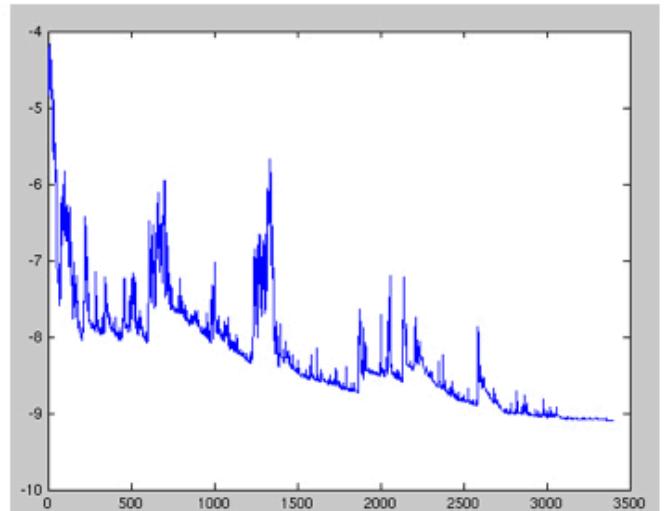
1. 在凸优化(Convex Optimization)的情况下, 一定会找到最优解
2. 在非凸优化的情况下, 一定能找到局部最优解
3. 单次参数调整计算量大
4. 不适合在线(Online)的情况

随机梯度下降

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for x, y in data:
        grad = gradient(loss_function, x, y, params)
        params = params - learning_rate * grad
```

随机梯度下降

1. 适合Online的情况
2. 通常比批处理梯度下降法快 (在批处理的情况下, 有可能许多的数据点产生的梯度是相似的, 这些计算是冗余的, 并不会有实际的帮助)
3. 通常目标函数震荡严重. 在神经网络优化情况下(没有全局最优解), 这种震荡反而有可能让它避免被套牢在一个局部最小值, 而找到更好的局部最优解
4. 通过调节学习率, 能够找到和批处理相似的局部或者全局最优解



迷你批处理梯度下降

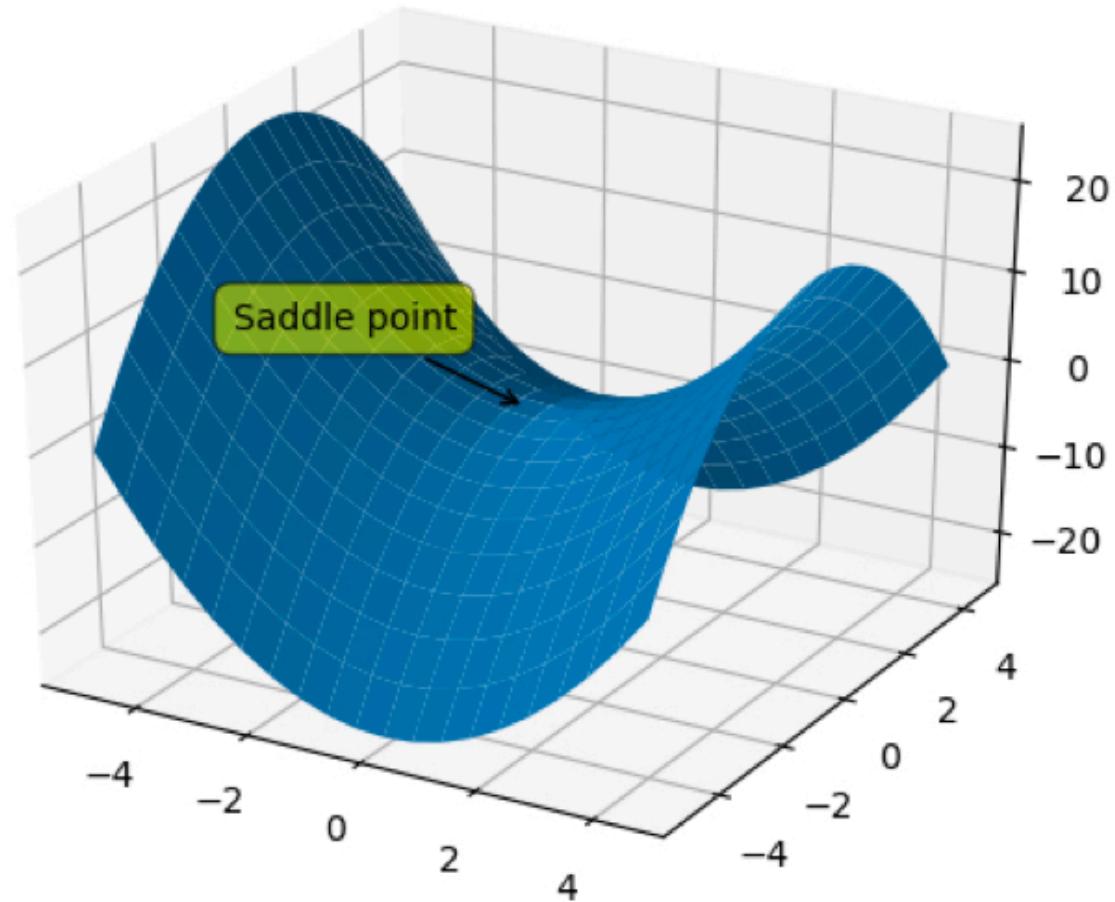
```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for mini_batch in get_mini_batch(data, batch_size=50):
        sum_grad = 0
        for x, y in mini_batch:
            grad = gradient(loss_function, x, y, params)
            sum_grad += grad
        avg_grad = sum_grad / len(data)
        params = params - learning_rate * avg_grad
```

迷你批处理梯度下降

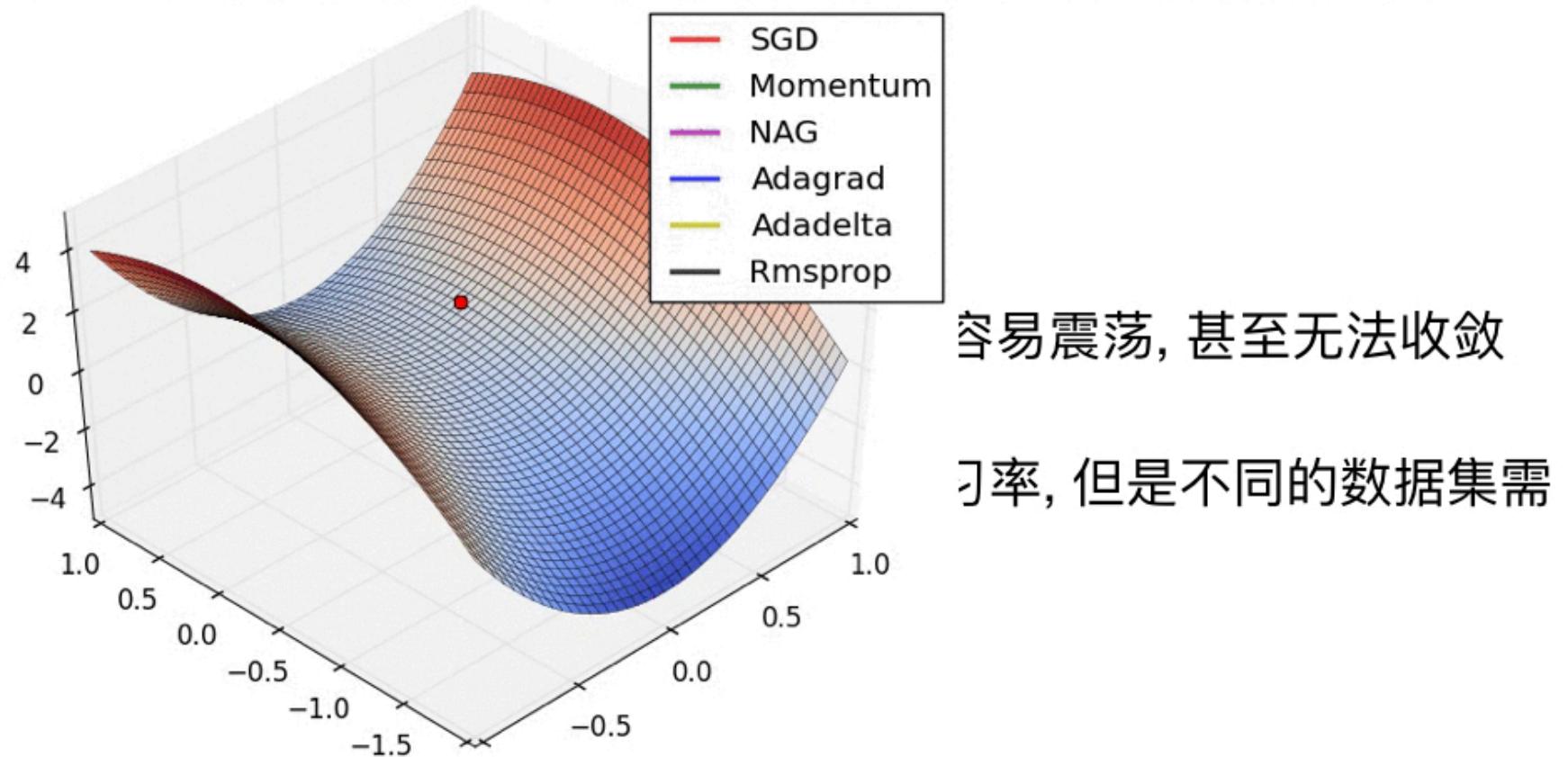
1. 结合了批处理和随机梯度下降法的优点
2. 减弱了目标函数震荡, 更加稳定
3. 易于硬件加速实现, 常用的机器学习库都利用了这个特性提供了高性能的计算速度
4. 一般的迷你批大小为50至256, 取决于不同的应用

传统梯度下降法面临的挑战

- 传统迷你批处理不能保证能够收敛
- 当学习率太小, 收敛会很慢, 学习率太高容易震荡, 甚至无法收敛
- 可以按照某个公式随着训练逐渐减小学习率, 但是不同的数据集需要不同的学习率变化曲线, 不容易估计
- 所有的参数使用同样的学习率并不合适
- 容易被套牢在马鞍点



传统梯度下降法面临的挑战



- 容易被套牢在马鞍点

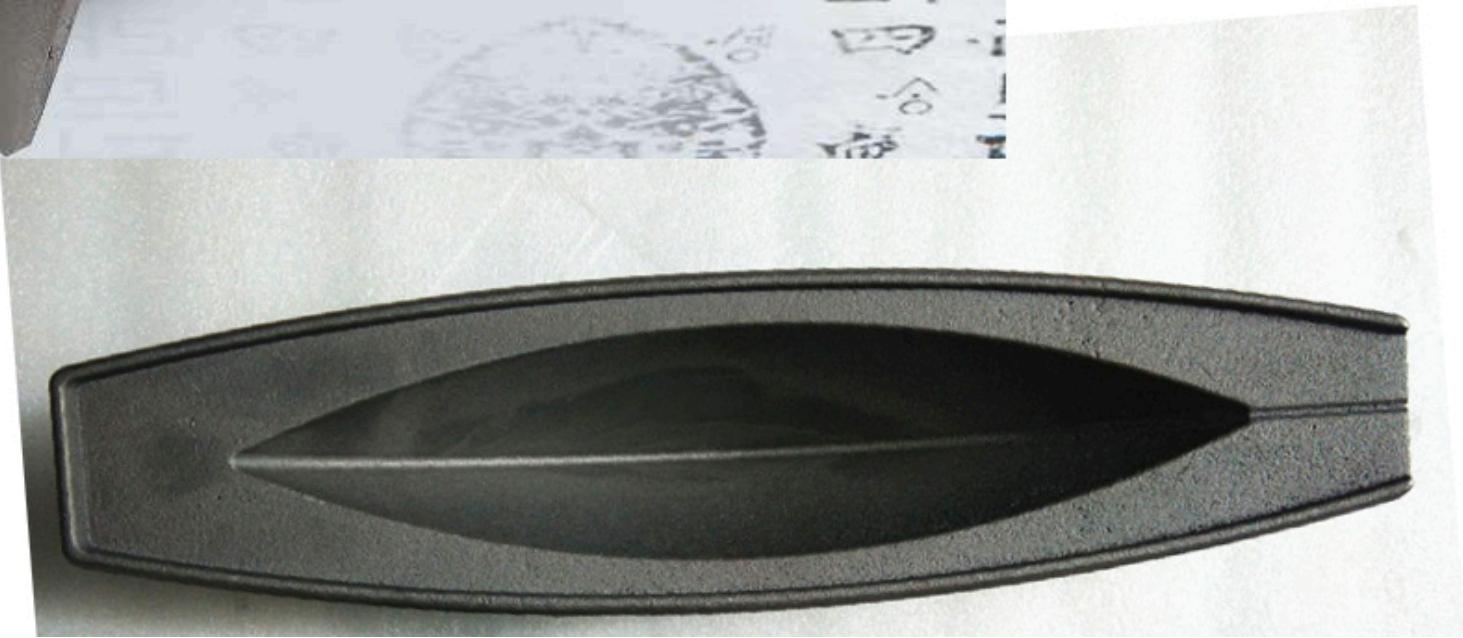
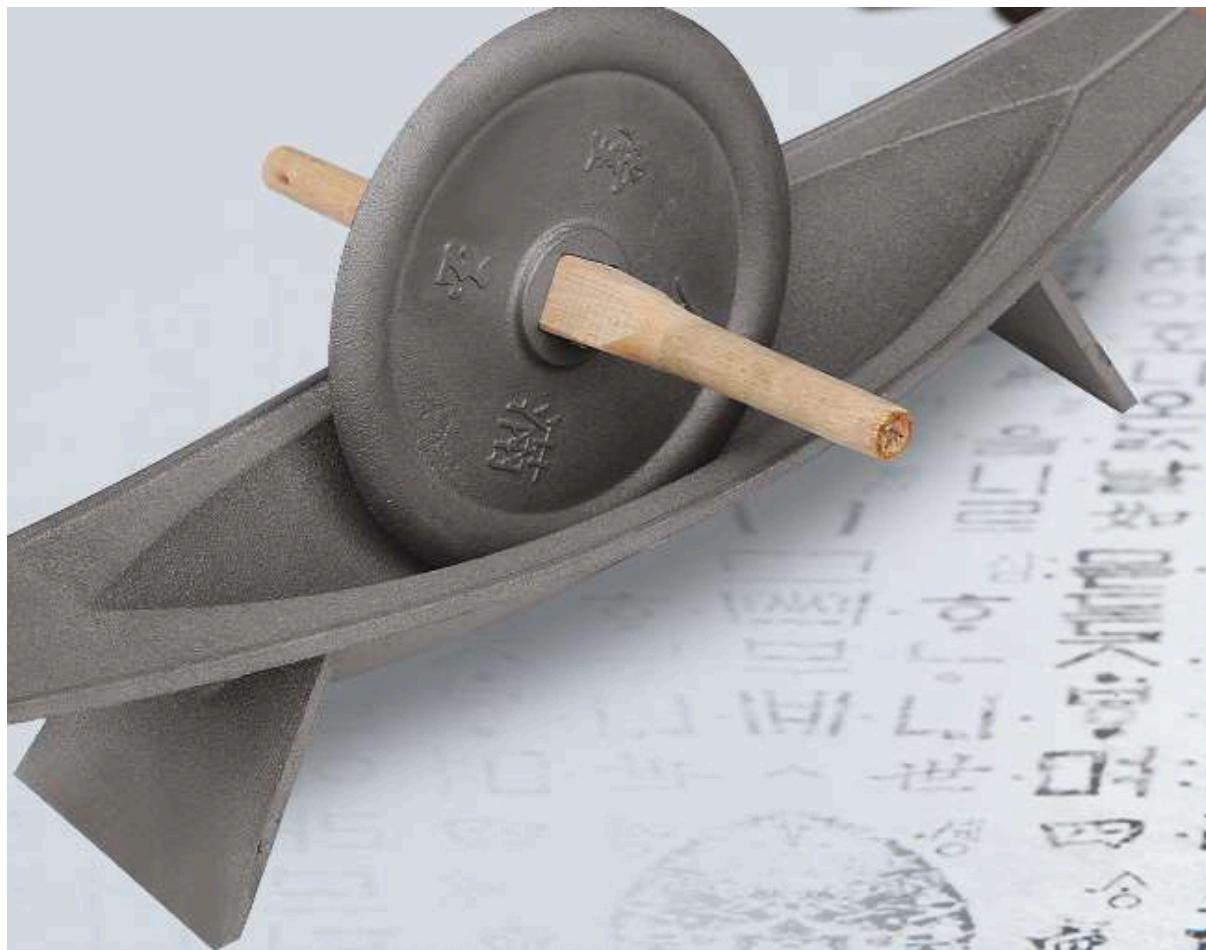
Momentum

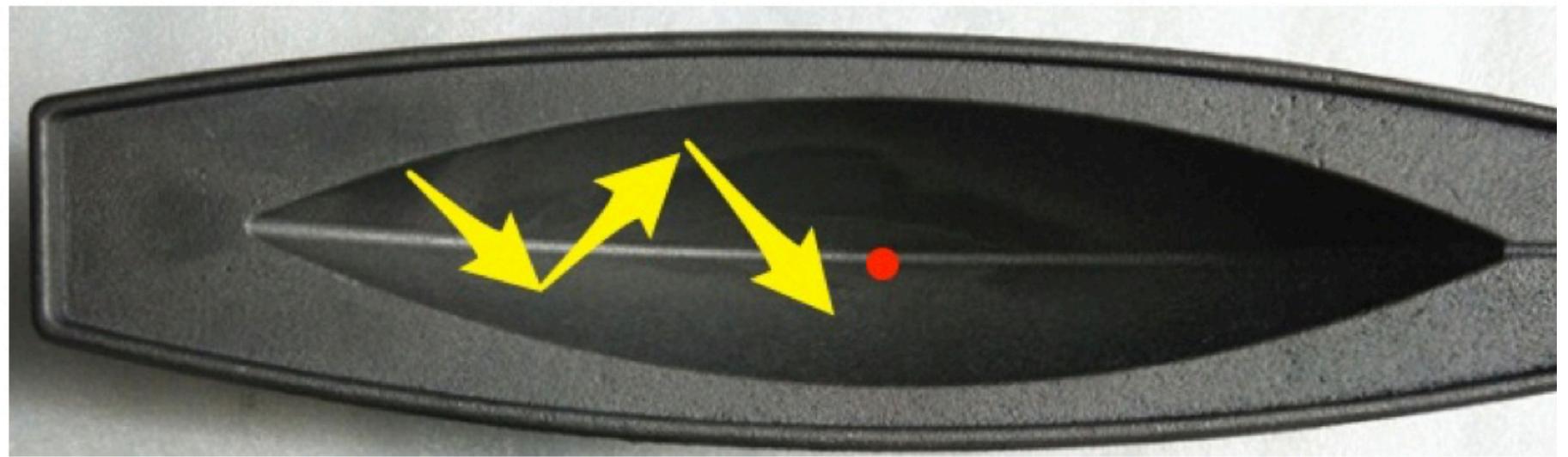
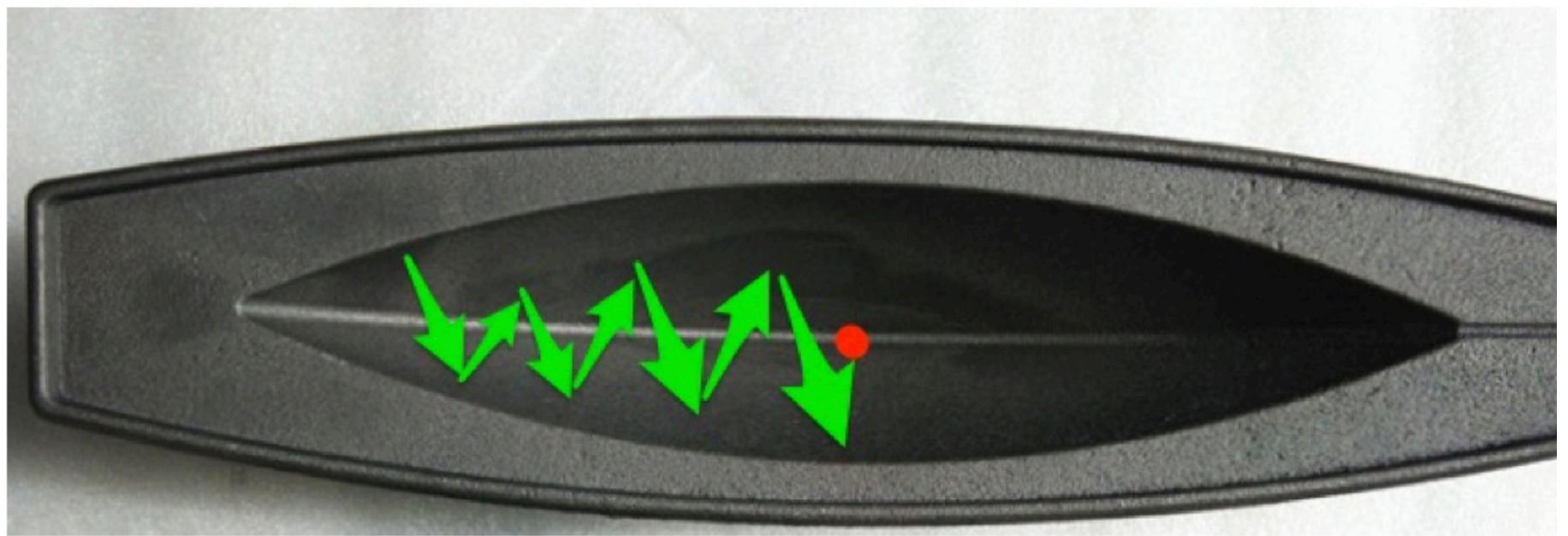
- 不同dimension的变化率不一样
- 动量在梯度在某一维度上的投影指向同一方向上增强, 在维度上的指向不断变化的方向抵消

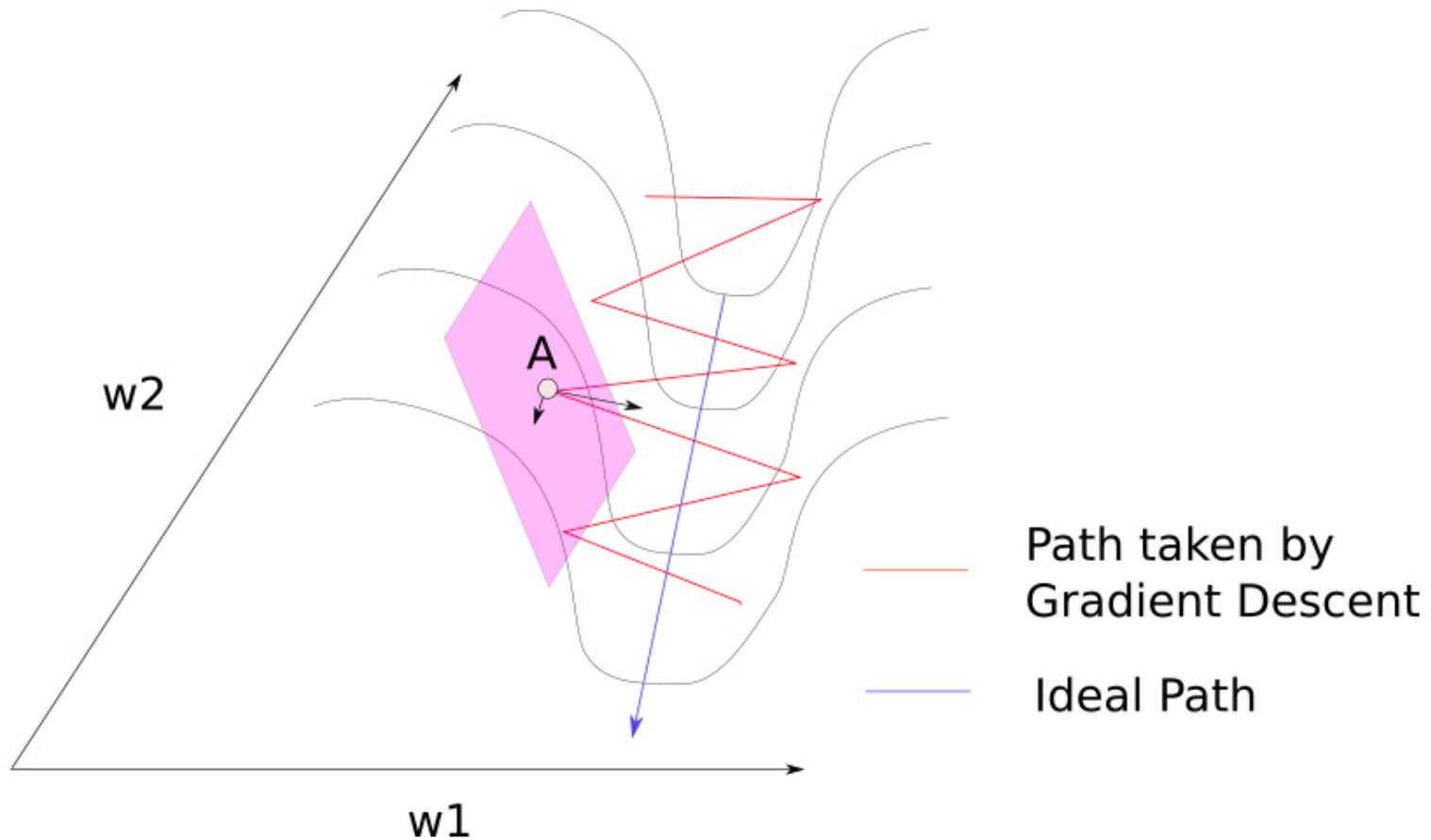
$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

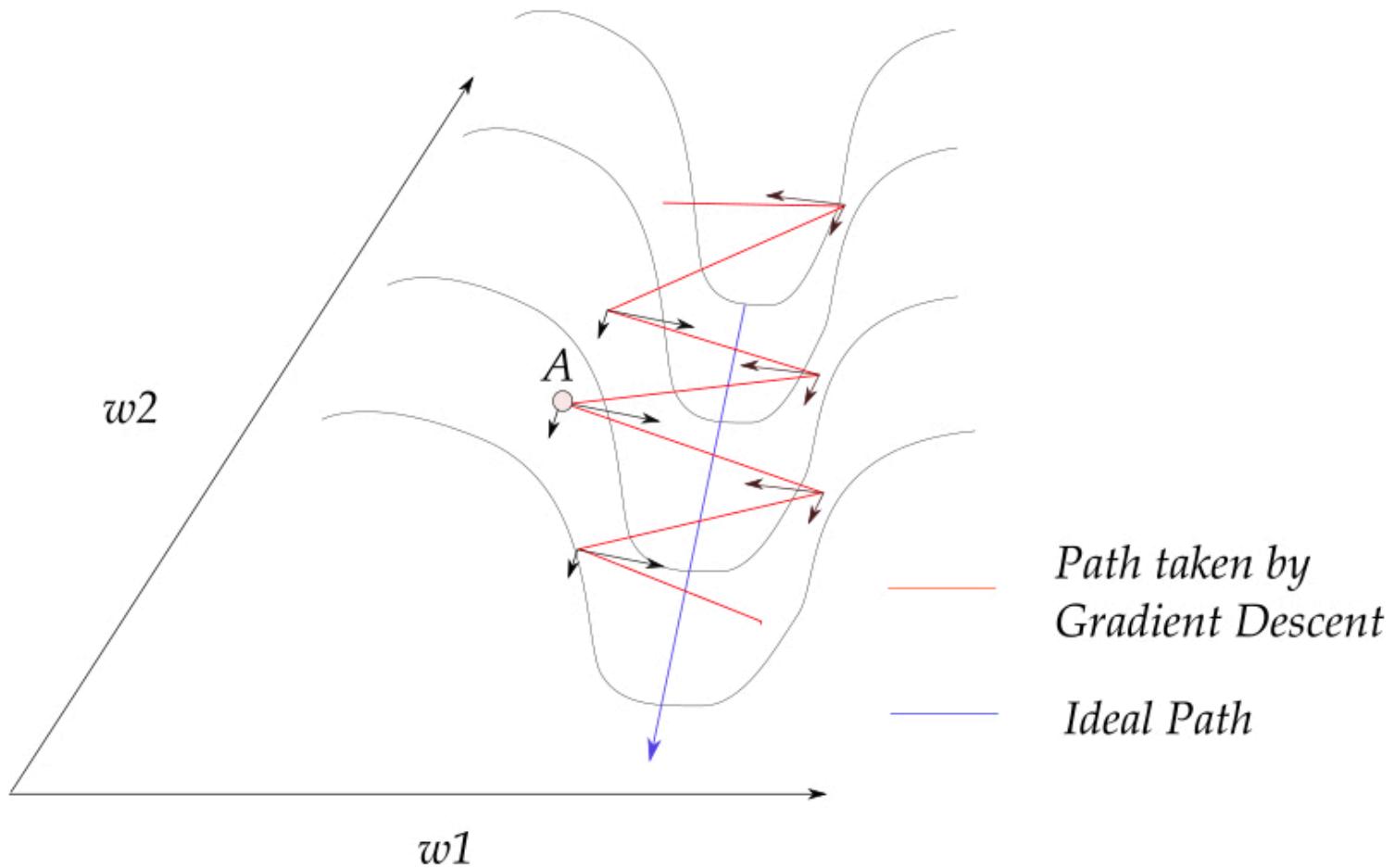
$$\theta = \theta - v_t$$

$$\gamma := 0.9$$









Nesterov accelerated gradient

- 动量 + 预测的前方的梯度
- 在多个RNN的任务中表现突出

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

$$\theta = \theta - v_t$$

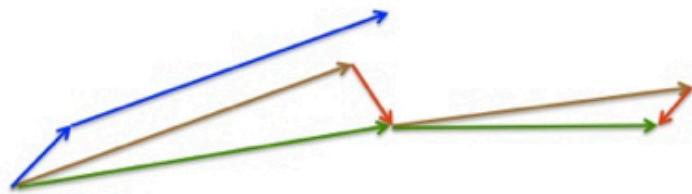


Image 4: Nesterov update (Source: [G. Hinton's lecture 6c](#))

对参数各个击破

- 前面的算法是整体提升梯度下降法的速度, 接下来的算法是针对各个参数采用不同的策略

Adagrad

- 对频繁出现的参数, 采用小的步长
- 对不频繁出现的参数, 采用大的步长
- 对稀疏数据集非常有用 (文本数据). Google在训练从Youtube视频自动识别猫用到了. Pennington et al训练词嵌入的GloVe也用到了

Adagrad

$$g_{t,I} = \nabla_{\theta} J(\theta_{t,I})$$

$$\theta_{t+1,I} = \theta_{t,I} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,I}$$

$G_t \in \mathbb{R}^{d \times d}$ here is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t [12], while ϵ is a smoothing term that avoids division by zero (usually on the order of $1e - 8$).

Adagrad

- 优势:
 - 无需手动调整learning rate步长
 - 设置初始步长为0.01即可
- 劣势:
 - 随着训练, 分母总是增大, 步长会越来越小, 算法无法收敛

Adadelta

- 只是累积过去的一段时间的梯度平方值
- 完全无需设置步长
- 为了便于实现, 采用类使用动量的策略:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2.$$

Adadelta

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

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma) \Delta\theta_t^2.$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}.$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

RMSprop

- Geoff Hinton Lecture 6e of his Coursera Class $\eta=0.001$

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Adam

- 记录过去一段时间的梯度平方和(类似Adadelta和RMSprop), 以及梯度的和(类似Momentum动量)
- 把优化看作铁球滚下山坡, Adam定义了一个带动量和摩擦的铁球

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

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

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \text{0.9 for } \beta_1, \text{0.999 for } \beta_2$$

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

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

如何选择

- 1. 如果数据集是稀疏的, 选择自适应学习率的方法会更快收敛, 而且免去了需要调试学习率的烦恼
- 2. RMSprop, Adadelta, Adam的效果非常相似, 大多数情况下Adam略好

Tips:

- 每一个epoch之前重新随机洗牌训练数据

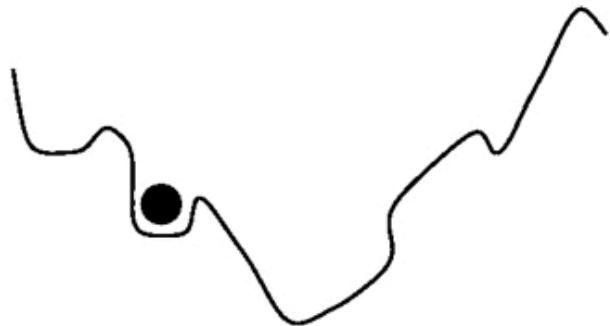
Tips:

- 使用Batch Normalization:
 - 我们一般会对训练数据做正则化, 但是随着数据前馈, 后面 layers的输入已经不是正则化的了, Batch Normalization就是在后面layer之间做正则化
 - 使得训练可使用更大的学习率, layer参数的初始化可以更加随意
 - BN还有regularization的作用, 还可以减少对Dropout的依赖

Tips:

- Early Stopping: Early stopping (is) beautiful free lunch" (NIPS 2015 Tutorial slides, slide 63)

Tips:



$$g_{t,i} = g_{t,i} + N(0, \sigma_t^2)$$

$$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

- 添加随机噪声到梯度
 - 使得layer参数初始化更加随意,
 - 使得model可以找到新的局部最小值

完

Tips:

- 每一个epoch之前重新随机洗牌训练数据
 - 使用Batch Normalization:
 - 我们一般会对训练数据做正则化, 但是随着数据前馈, 后面layers的输入已经不是正则化的了, Batch Normalization就是在后面layer之间做正则化
 - 使得训练可使用更大的学习率, layer参数的初始化可以更加随意
 - BN还有regularization的作用, 还可以减少对Dropout的依赖
 - Early Stopping: Early stopping (is) beautiful free lunch" (NIPS 2015 Tutorial slides, slide 63)
 - 添加随机噪声到梯度
 - 使得layer参数初始化更加随意,
 - 使得model可以找到新的局部最小值
- $$g_{t,i} = g_{t,i} + N(0, \sigma_t^2)$$
- $$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$