神经网络的参数优化

主讲：王皓 副研究员| 硕士导师

邮箱：wanghao3@ustc.edu.cn

主页：http://staff.ustc.edu.cn/~wanghao3

本章内容

➢网络优化基本介绍

➢优化算法

- SGD
- AdaGrad
- RMSProp
- Adam等

➢参数初始化

- 随机初始化
- Xavier初始化
- He初始化

神经网络优化的挑战

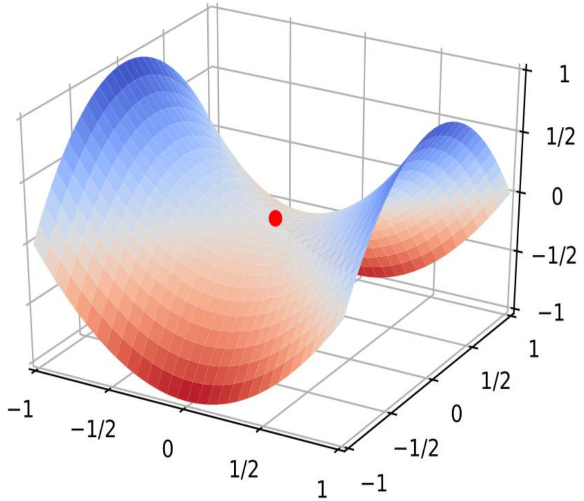
- 局部极小值非常多甚至不可数无限多的局部极小值

- 对于足够大的神经网络而言，大部分局部极小值具有很小的损失

- 很多高维非凸函数的局部极小值事实上都远少于鞍点

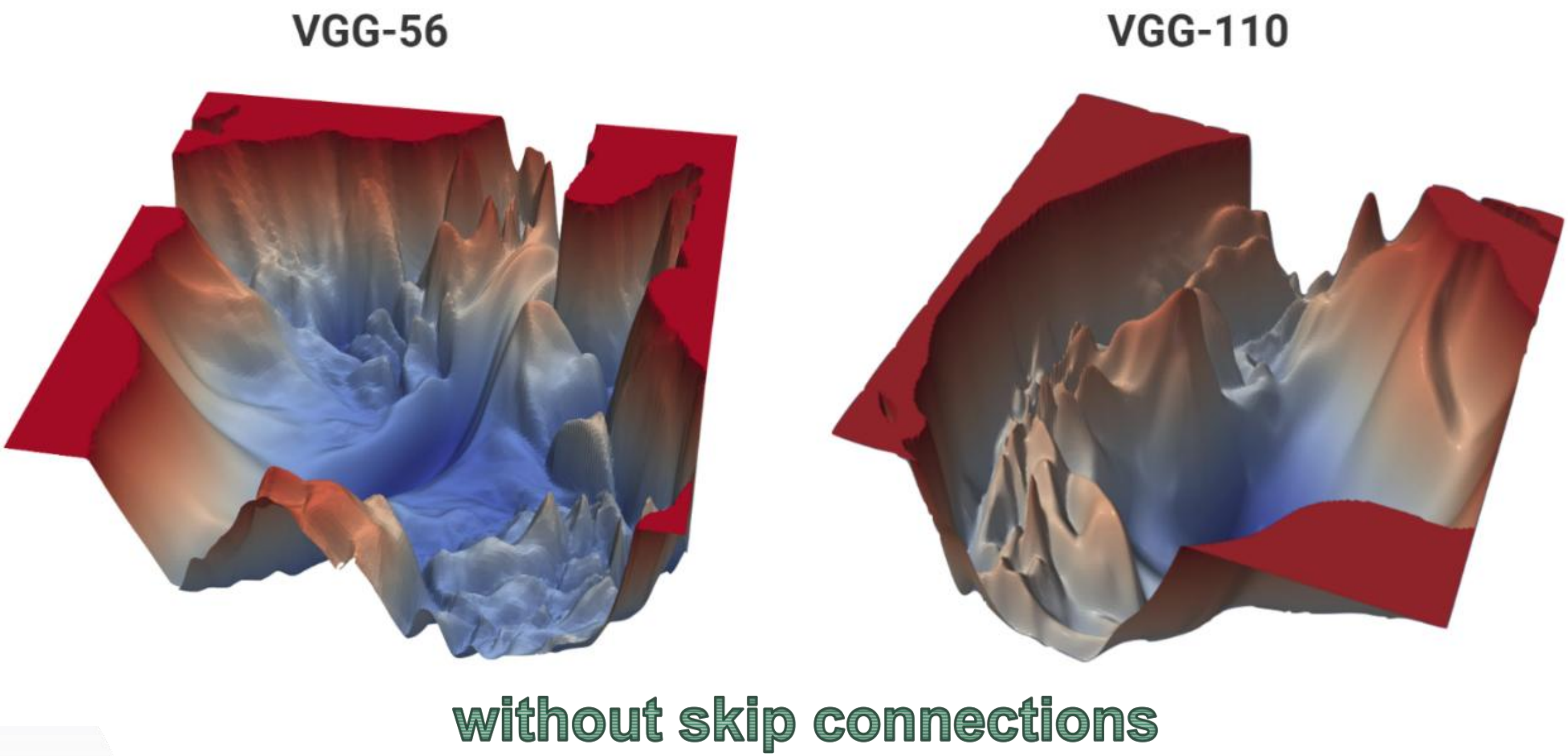
- 真实的神经网络也存在包含很多高代价鞍点的损失函数

鞍点：梯度为 0 的点



神经网络优化的挑战

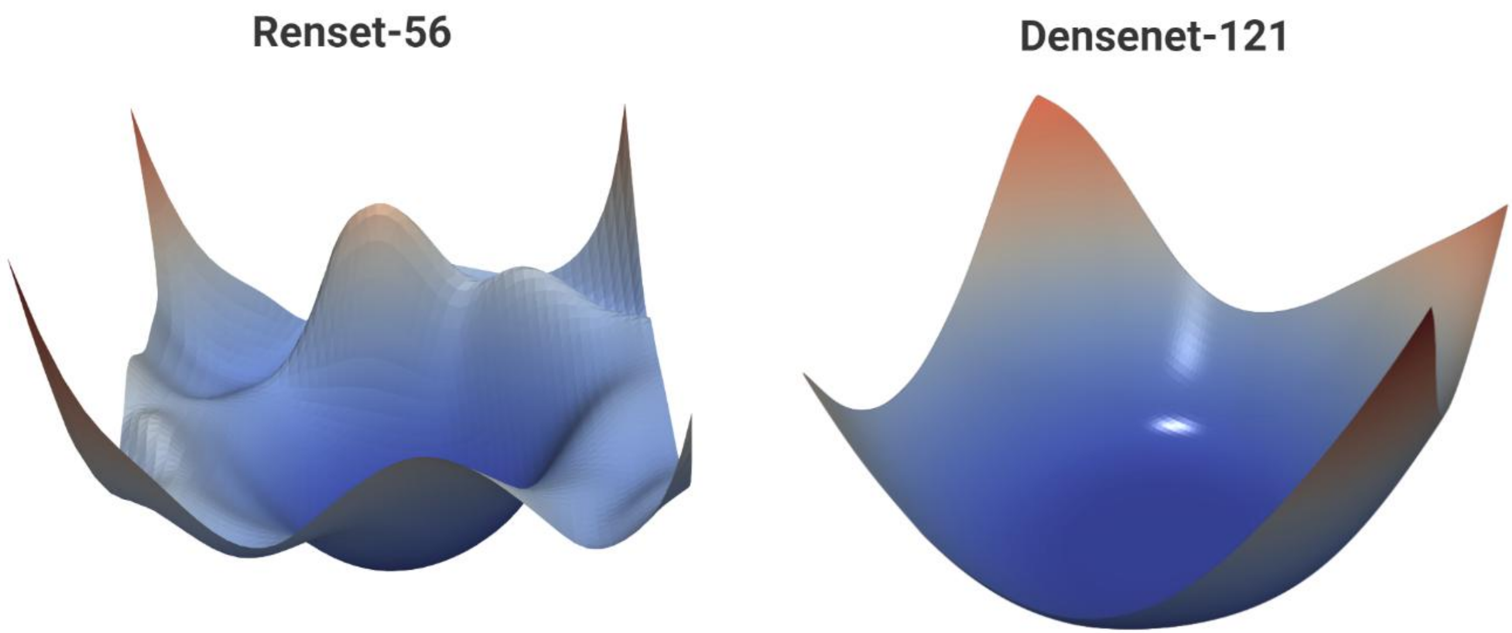
优化地形（Optimization Landscape）：在高维空间中损失函数的曲面形状



Li H, Xu Z, Taylor G, et al. Visualizing the loss landscape of neural nets[C] //Advances in Neural Information Processing Systems. 2018: 6389-6399.

修改网络结构获得更好的优化地形

- 好的优化地形通常比较平滑，更容易优化
- 使用 ReLU 激活函数、残差连接、逐层归一化等



with skip connections

神经网络优化的其他改善方法

- 更有效的优化算法来提高优化方法的效率和稳定性
  - 例如：动态学习率调整，梯度估计修正
- 更好的参数初始化方法
- 更好的数据预处理方法

提高优化效率

- 使用更好的超参数优化方法

优化算法

随机梯度下降

- 由于数据集可能很大，无法全部放入内存计算梯度
- 一般采用小批量随机梯度下降法，每次从数据集中采样一部分样本（称为batch），计算batch上的梯度，并进行参数更新
- 给定训练集为$D=\{(\boldsymbol{x}^{(n)},\boldsymbol{y}^{(n)})\}_{n=1}^{N}$，每次采样B个样本

梯度下降

小批量随机梯度下降

$$\mathcal{L}_D(\boldsymbol{W},\boldsymbol{b})=\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}\left(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)}\right)+\frac{1}{2}\lambda\|\boldsymbol{W}\|_F^2 \Rightarrow \mathcal{L}_B(\boldsymbol{W},\boldsymbol{b})=\frac{1}{B}\sum_{i=1}^{B}\mathcal{L}\left(\boldsymbol{y}^{(i)},\hat{\boldsymbol{y}}^{(i)}\right)+\frac{1}{2}\lambda\|\boldsymbol{W}\|_F^2$$

$$\frac{\partial\mathcal{L}_D(\boldsymbol{W},\boldsymbol{b})}{\partial\boldsymbol{W}^{(l)}}=\frac{1}{N}\sum_{n=1}^{N}\frac{\partial\mathcal{L}\left(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)}\right)}{\partial\boldsymbol{W}^{(l)}} \Rightarrow \frac{\partial\mathcal{L}_B(\boldsymbol{W},\boldsymbol{b})}{\partial\boldsymbol{W}^{(l)}}=\frac{1}{B}\sum_{i=1}^{B}\frac{\partial\mathcal{L}\left(\boldsymbol{y}^{(i)},\hat{\boldsymbol{y}}^{(i)}\right)}{\partial\boldsymbol{W}^{(l)}}$$

样本随机性的影响

- 在每次迭代时，随机选择B个样本，这里的随机性非常重要

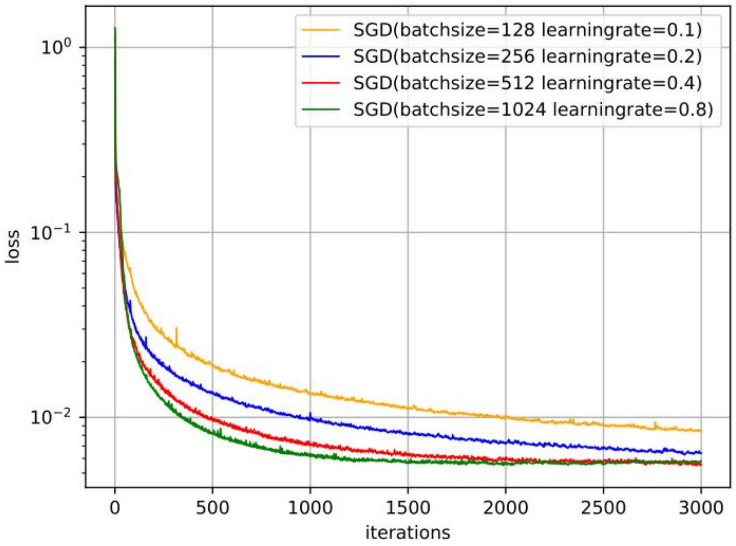
- 但随机性在大规模数据情况下很难满足

- 实践中通常将样本顺序打乱一次，然后按照这个顺序存储起来

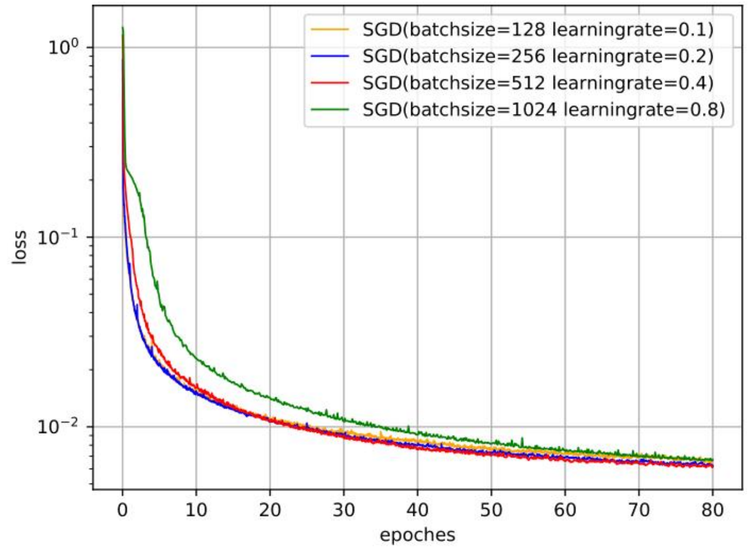
- 虽然偏离真实随机采样，但不会有严重的有害影响

批量大小的影响

- 批量大小不影响梯度期望，但会影响梯度方差，一般为2的幂数
  - 批量越大，随机梯度的方差越小，引入的噪声也越小，训练也越稳定，因此可以设置较大的学习率
  - 批量较小时，需要设置较小的学习率，否则模型会不收敛



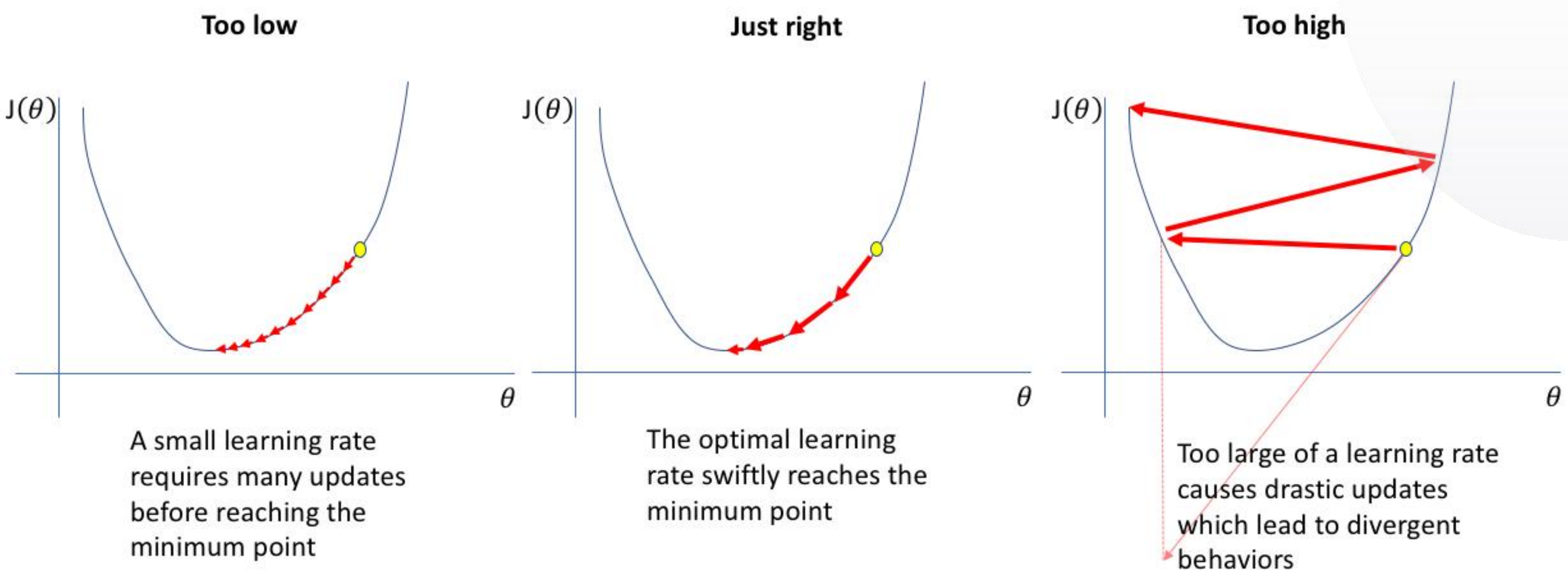
(a) 按 Iteration 的损失变化



(b) 按 Epoch 的损失变化

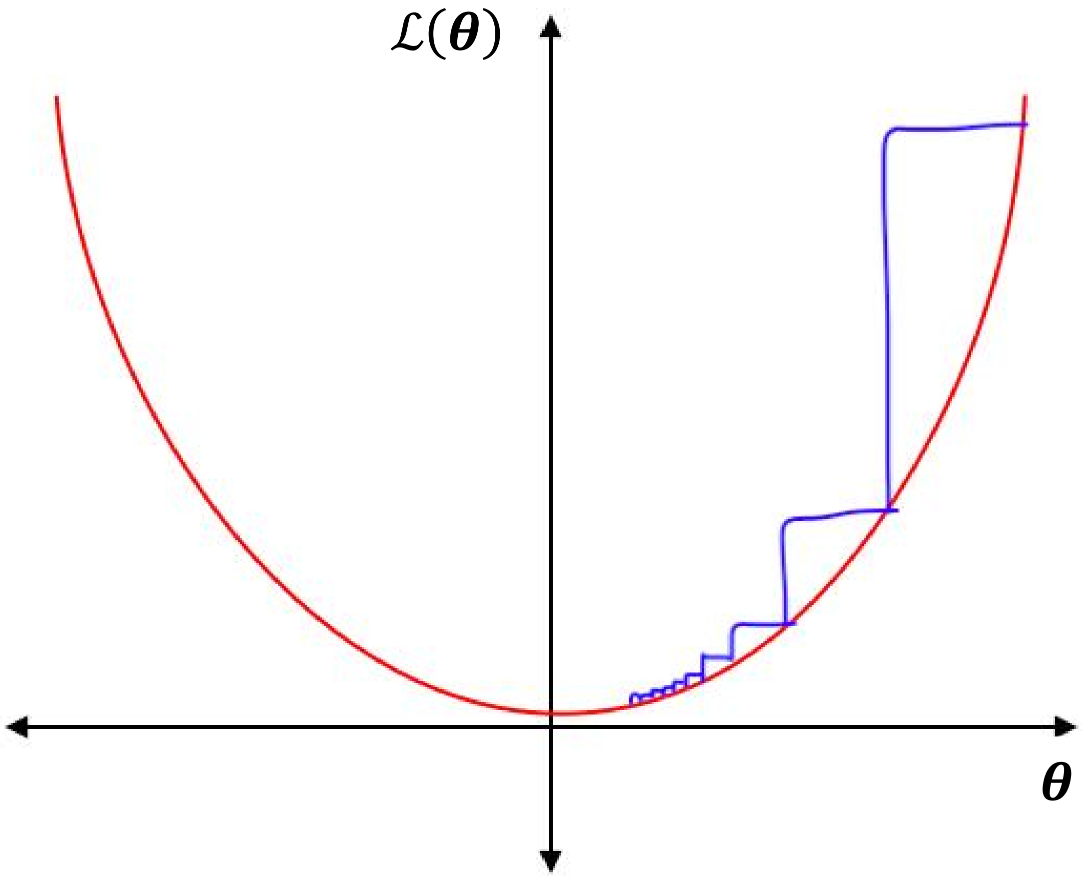
小批量梯度下降中，每次选取样本数量对损失下降的影响

学习率的影响



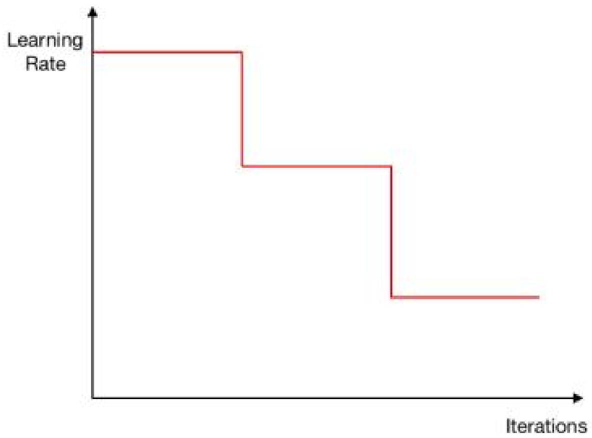
- 学习率调整方法：
  - 学习率衰减
  - 学习率预热
  - 周期性学习率调整
  - 自适应调整学习率

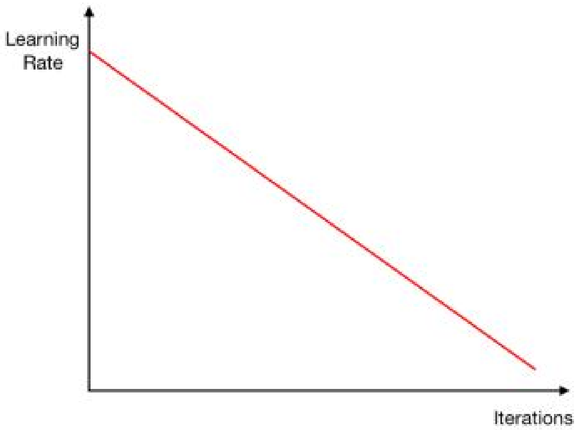
学习率衰减

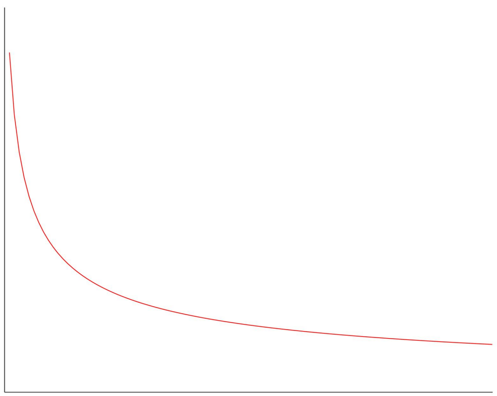


- 学习率一开始要保持大些保证收敛速度
- 在收敛到最优点附近时要小些以避免来回振荡

学习率衰减



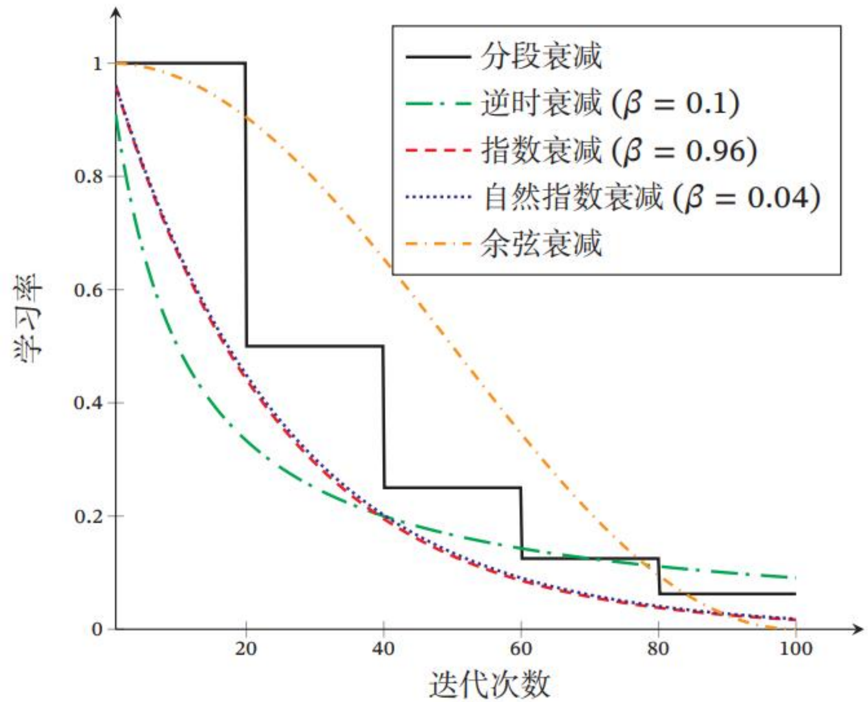




梯级衰减（step decay） 线性衰减（Linear Decay） 1/t衰减（1/t decay）

学习率衰减

β为衰减率



➢假设初始学习率为α_0，在第t次迭代时的学习率为α_t

- 分段衰减
- 逆时衰减 $\alpha_t = \alpha_0 \frac{1}{1+\beta \times t}$
- 指数衰减 $\alpha_t = \alpha_0 \beta^t$ $\beta < 1$
- 自然指数衰减 $\alpha_t = \alpha_0 \exp(-\beta t)$
- 余弦衰减

$$\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$$

学习率预热

➢刚开始训练时，参数是随机初始化的，梯度往往也比较大，比较大的初始学习率使得训练不稳定

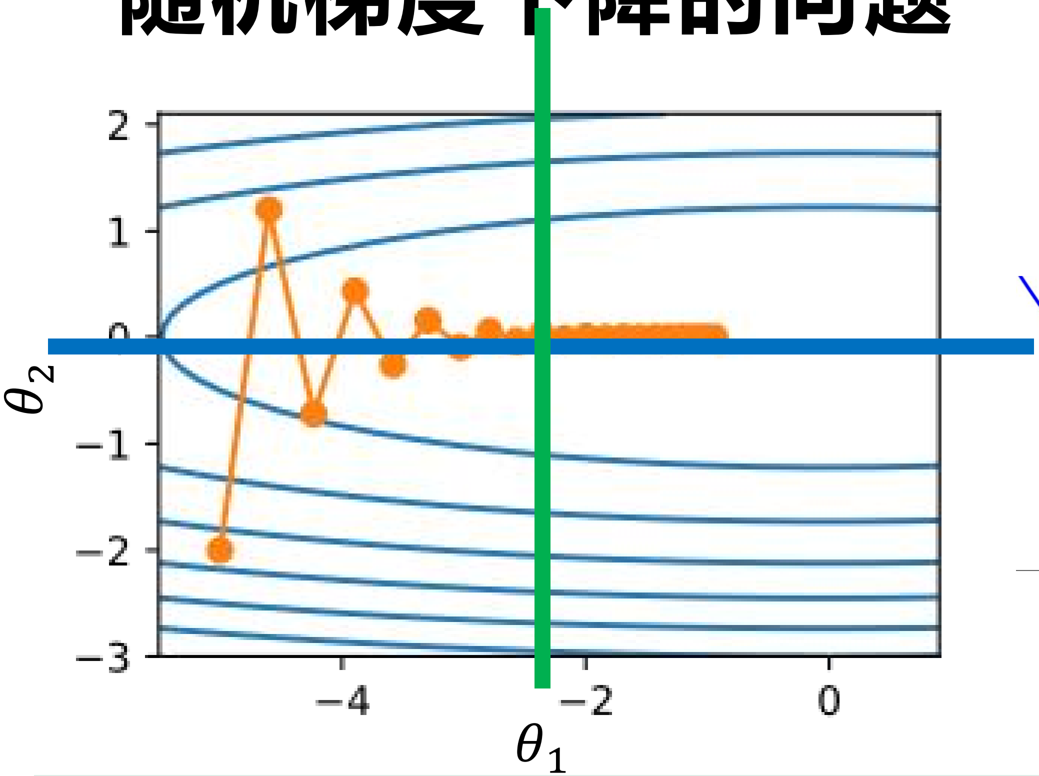
➢为了提高训练稳定性，可以在最初几轮迭代时，采用比较小的学习率，等梯度下降到一定程度后在恢复到初始学习率

➢采用的方法是：在最初的T'迭代中，初始学习率为α_0，每次更新的学习率为

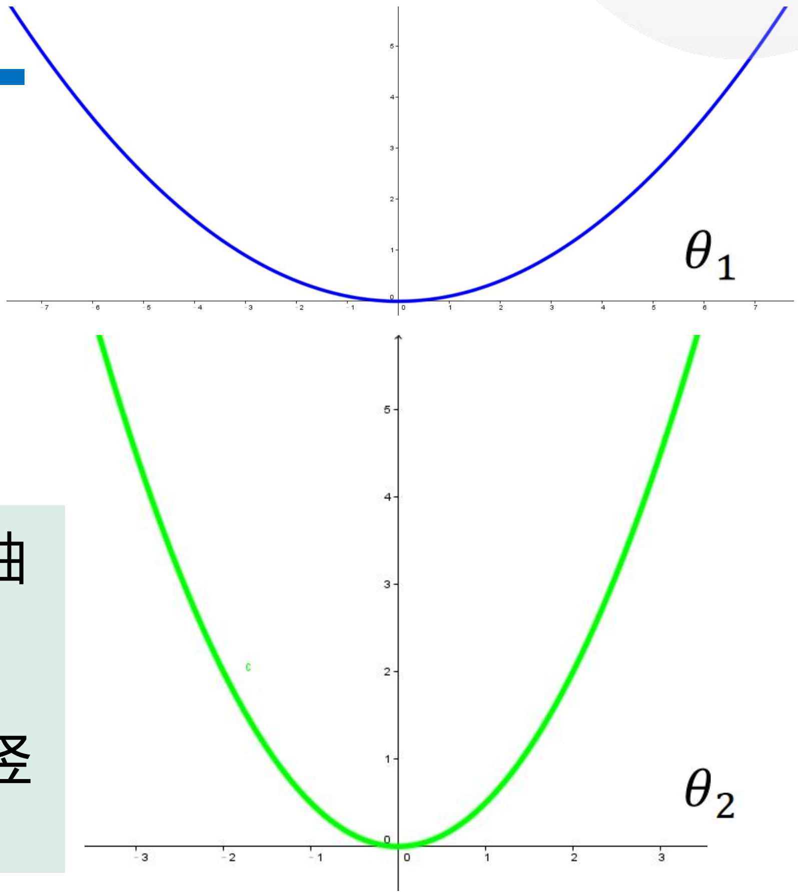
$$\alpha'_t = \frac{t}{T'}\alpha_0, 1 \le t \le T'$$

➢在这个预热阶段结束后，再选择另外一种学习率衰减方法来逐渐降低学习率

随机梯度下降的问题



不同参数不同学习率



- 同一位置，损失在θ_2 轴方向比在 θ_1 轴方向的斜率的绝对值更大
- 给定学习率，梯度下降迭代参数时在竖直方向会比在水平方向移动幅度更大

自适应学习率—AdaGrad

➢将学习率除以每个参数历史梯度的平方根

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \qquad \sigma^0 = \sqrt{(g^0)^2}$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{(\sigma^0)^2 + (g^1)^2}$$

$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{(\sigma^1)^2 + (g^2)^2}$$

$$\vdots \qquad \vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \qquad \sigma^t = \sqrt{(\sigma^{t-1})^2 + (g^t)^2}$$

$$g^t = \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{L}\left(\boldsymbol{y}^{(i)}, f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)\right)}{\partial w}$$

w 是其中一个参数

σ^t是参数w的历史梯度平方根

自适应学习率—AdaGrad

- AdaGrad参数更新流程：
  - 计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \mathcal{L}\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  - 累积平方梯度：$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
  - 计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ ⟶ 逐元素地应用除和求平方根操作
  - 应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
- 在 AdaGrad算法中，如果某个参数的偏导数累积比较大，其学习率相对较小；相反，如果其偏导数累积较小，其学习率相对较大。但整体是随着迭代次数的增加，学习率逐渐缩小。
- AdaGrad缺点：在经过一定次数的迭代依然没有找到最优点时，由于这时的学习率已经非常小，很难再继续找到最优点。

经验上发现，对于训练深度神经网络模型而言，从训练开始时积累梯度平方会导致**有效学习率过早和过量的减小**

自适应学习率—RMSProp

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \qquad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2}$$

$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots \qquad \vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

指数加权的移动平均

$$\boxed{(\sigma^t)^2 = (\alpha)^t g_0^2 + (1-\alpha)\sum_{i=1}^{t}(\alpha)^{t-i}\left(g^i\right)^2}$$

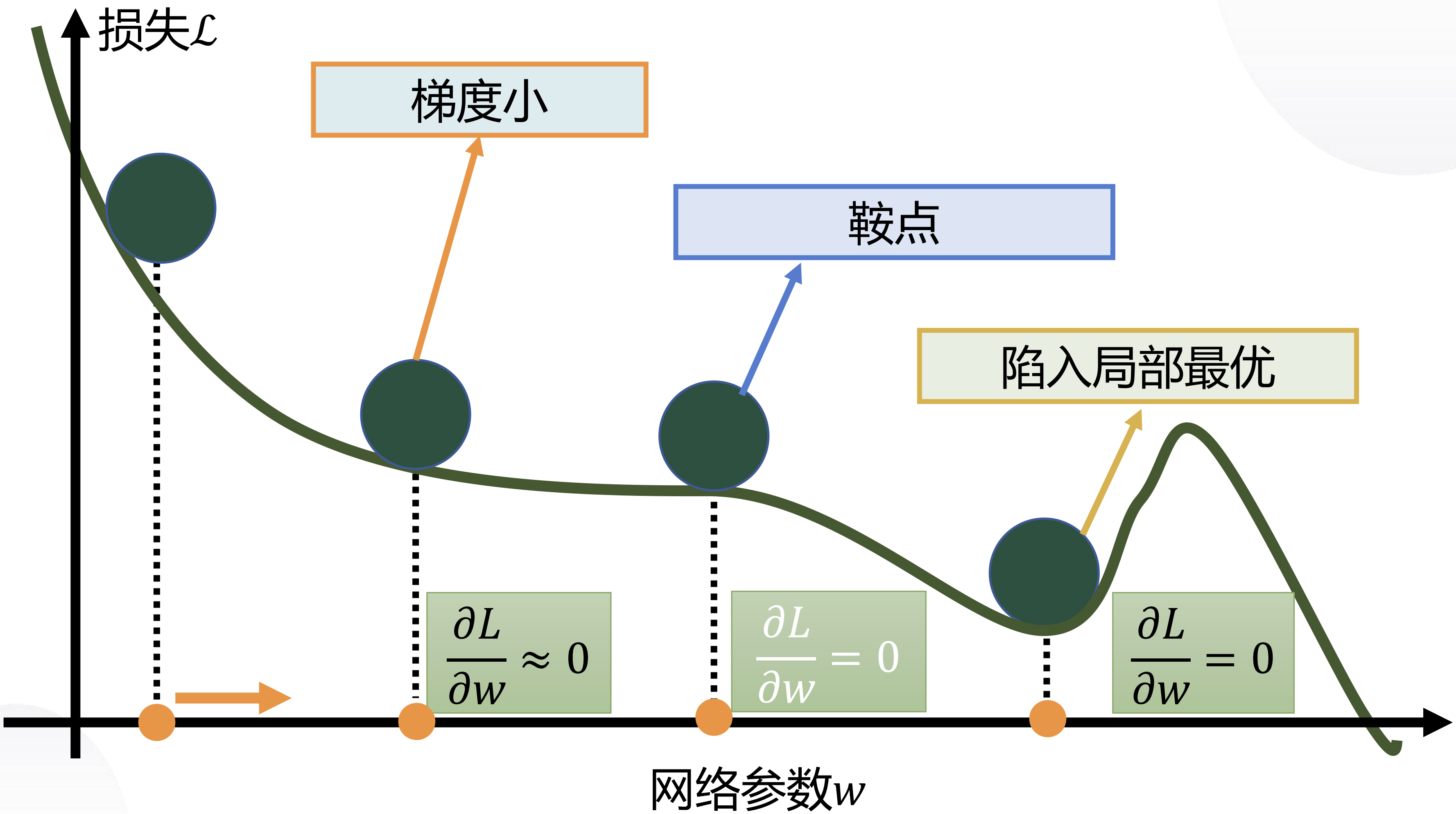
自适应学习率—RMSProp

- RMSProp参数更新流程：
  - 计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \mathcal{L}\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  - 累积平方梯度：$\boxed{\boldsymbol{r} \leftarrow \rho\, \boldsymbol{r} + (1-\rho) \boldsymbol{g} \odot \boldsymbol{g}}$
  - 计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$
  - 应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

Adagrad 累积平方梯度：$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

在RMSProp迭代过程中，每个参数的学习率并不是呈衰减趋势，既可以变小也可以变大

神经网络优化的挑战

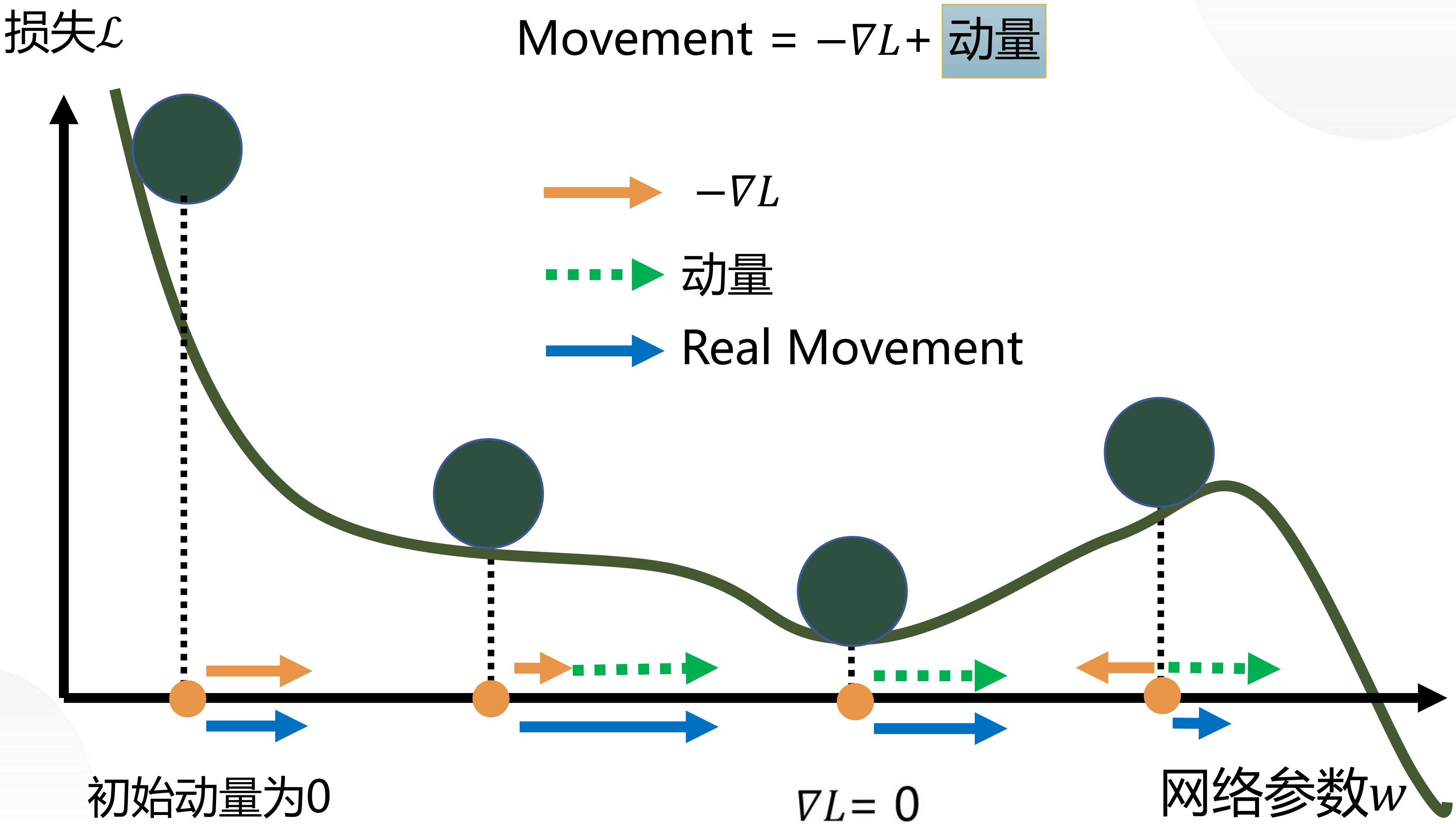


损失$\mathcal{L}$
梯度小
鞍点
陷入局部最优
$\frac{\partial L}{\partial w} \approx 0$
$\frac{\partial L}{\partial w} = 0$
$\frac{\partial L}{\partial w} = 0$
网络参数w

动量法的动机

一个物体的动量指的是该物体在它运动方向上保持运动的趋势，是该物体的质量和速度的乘积

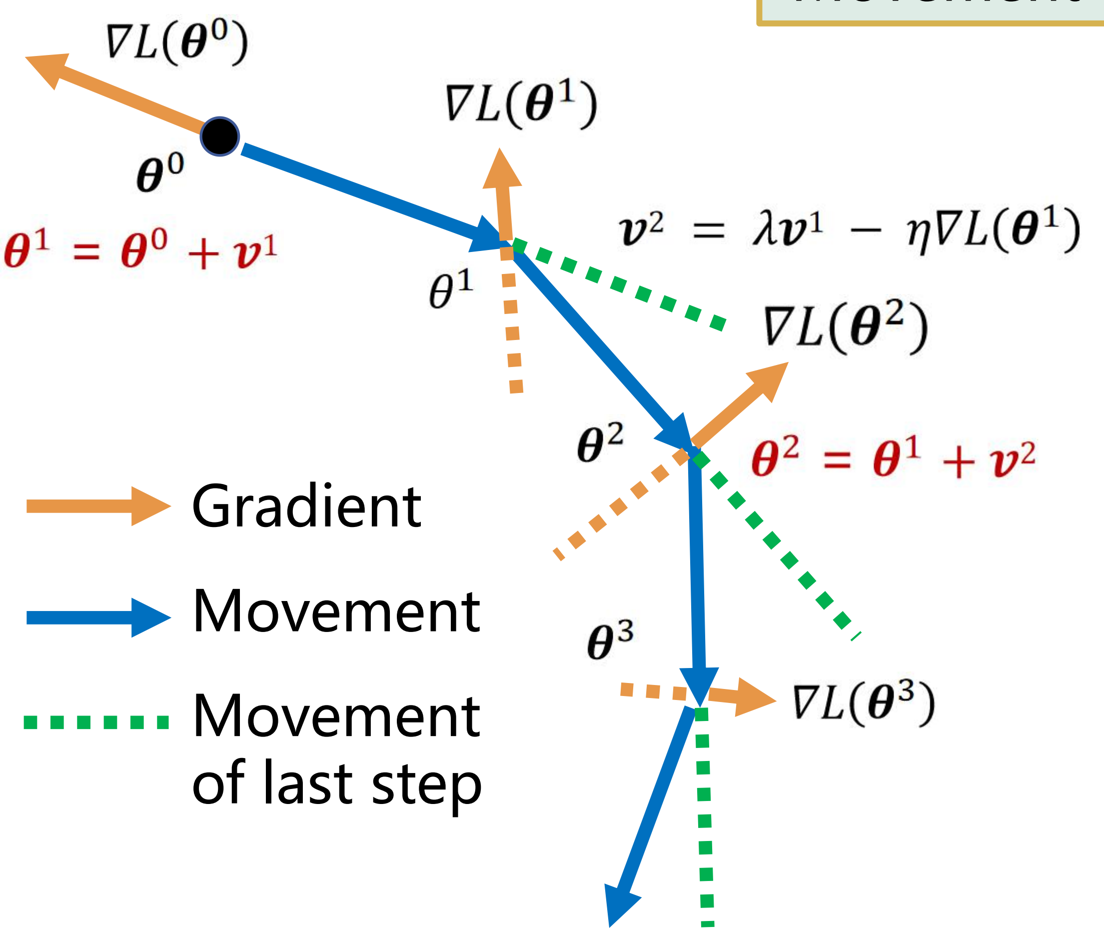
$$\text{Movement} = -\nabla L + \boxed{\text{动量}}$$



动量法

$\boldsymbol{v}^1 = \lambda \boldsymbol{v}^0 - \eta \nabla L(\boldsymbol{\theta}^0)$

Movement = $-\nabla L$+ 动量



动量法

➢ $\boldsymbol{v}^i$是历史负梯度的加权和

$$\boldsymbol{v}^0 = 0$$

$$\boldsymbol{v}^1 = \lambda \boldsymbol{v}^0 - \eta \nabla L(\boldsymbol{\theta}^0)$$

$$\boldsymbol{v}^2 = \lambda \boldsymbol{v}^1 - \eta \nabla L(\boldsymbol{\theta}^1)$$

$$\boldsymbol{v}^3 = \lambda \boldsymbol{v}^2 - \eta \nabla L(\boldsymbol{\theta}^2)$$

$$\vdots$$

$$\boldsymbol{v}^i = \lambda \boldsymbol{v}^{i-1} - \eta \nabla L\left(\boldsymbol{\theta}^{i-1}\right)$$

$$\boldsymbol{v}^0 = 0$$

$$\boldsymbol{v}^1 = -\eta \nabla L(\boldsymbol{\theta}^0)$$

$$\boldsymbol{v}^2 = -\lambda \eta \nabla L(\boldsymbol{\theta}^0) - \eta \nabla L(\boldsymbol{\theta}^1)$$

$$\boldsymbol{v}^3 = -\lambda^2 \eta \nabla L(\boldsymbol{\theta}^0) - \lambda \eta \nabla L(\boldsymbol{\theta}^1) - \eta \nabla L(\boldsymbol{\theta}^2)$$

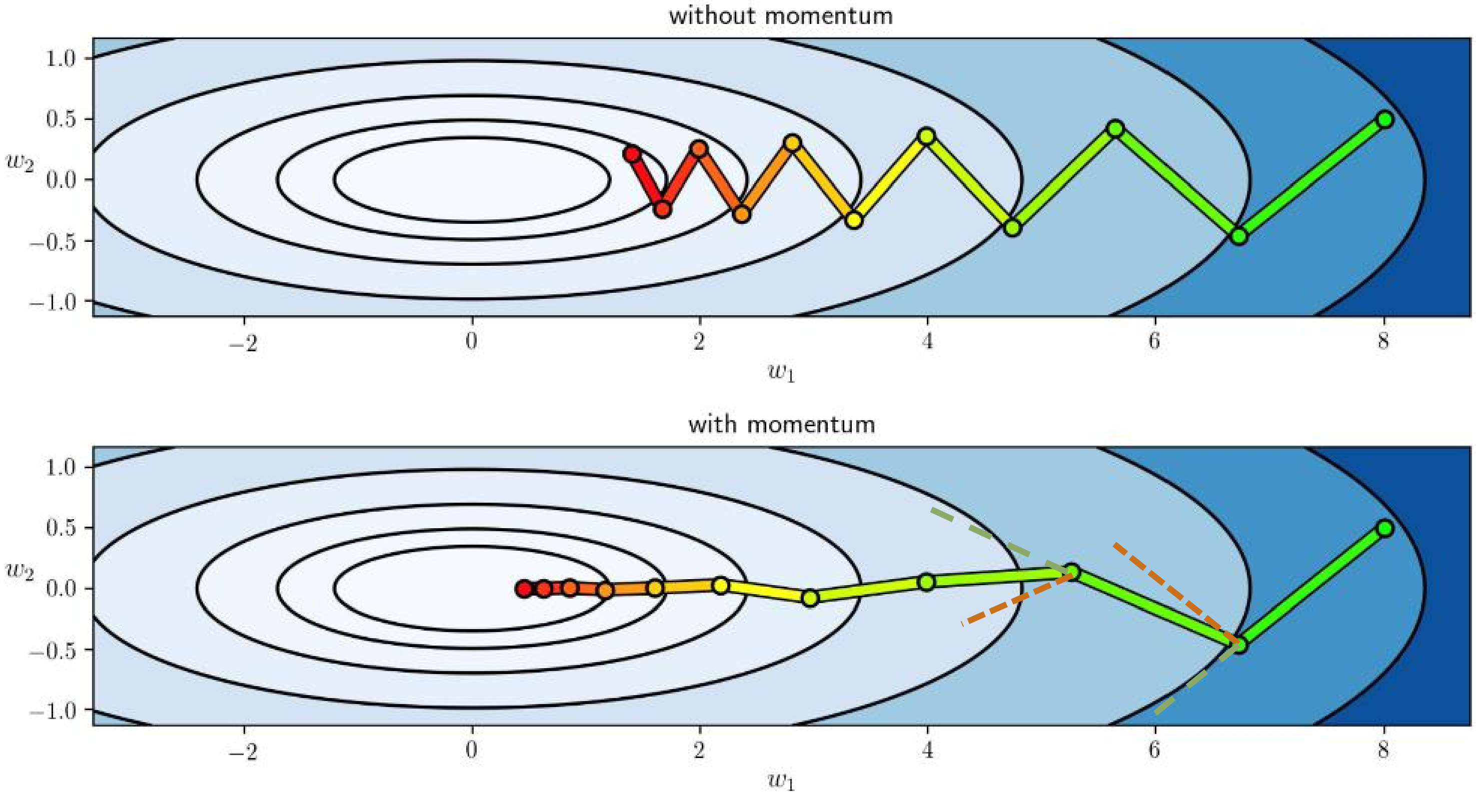
$$\vdots$$

$$\boldsymbol{v}^i = -\eta \sum_{k=1}^{i} \lambda^{i-k} \nabla L(\boldsymbol{\theta}^{k-1})$$

$$\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} + \boldsymbol{v}^t$$

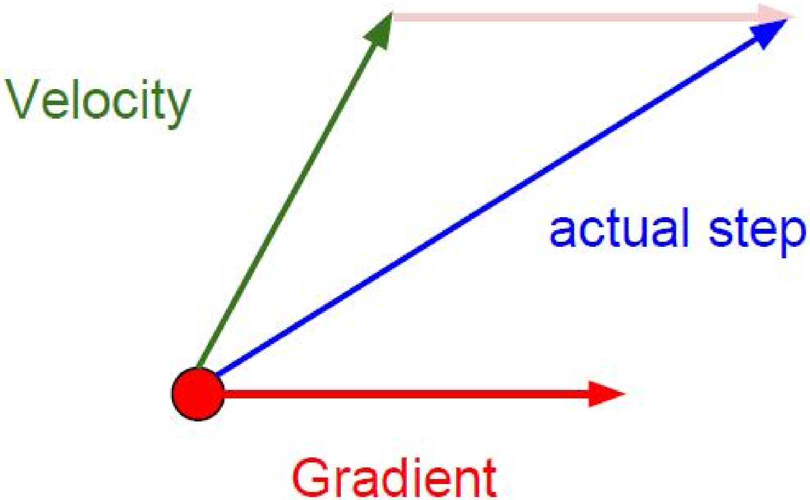
在第t次迭代时，计算负梯度的“加权移动平均”作为参数的更新方向

动量法的作用


without momentum
1.0
0.5
w_2 0.0
−0.5
−1.0
−2
0
2
4
6
8
w_1
with momentum
1.0
0.5
w_2 0.0
−0.5
−1.0
−2
0
2
4
6
8
w_1

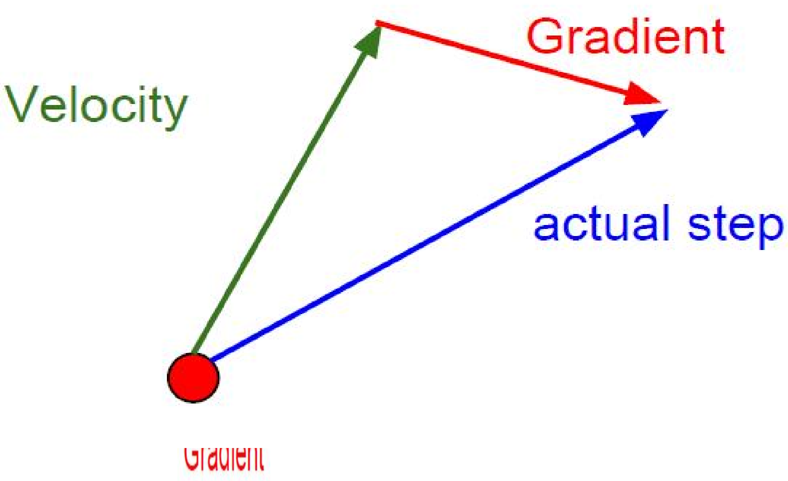
Nesterov动量

动量更新



结合梯度和速度来更新权重

Nesterov动量更新



沿着速度方向试走一步再计算梯度，再结合速度来更新权重

Nesterov动量法

动量法

计算梯度估计：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \mathcal{L}\left(\boxed{f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)}, y^{(i)}\right)$

计算动量更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

Nesterov动量法

应用临时更新：$\widetilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

计算梯度(临时点)：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \mathcal{L}\left(\boxed{f\left(\boldsymbol{x}^{(i)}; \widetilde{\boldsymbol{\theta}}\right)}, y^{(i)}\right)$

计算动量更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

RMSProp+Nesterov动量

RMSProp

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \mathcal{L}\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$

累积平方梯度：$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho) \boldsymbol{g} \odot \boldsymbol{g}$

计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta+\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

RMSProp +Nesterov 动量

计算临时更新：$\widetilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \mathcal{L}\left(f\left(\boldsymbol{x}^{(i)}; \widetilde{\boldsymbol{\theta}}\right), y^{(i)}\right)$

累积平方梯度：$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho) \boldsymbol{g} \odot \boldsymbol{g}$

计算动量更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

动量作用于缩放后的梯度

Adam≈动量法+RMSprop

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i \mathcal{L}(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$

$t \leftarrow t+1$

更新有偏一阶矩估计：$\boxed{\boldsymbol{s}} \leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g}$

更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g}\odot\boldsymbol{g}$

修正一阶矩的偏差：$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$

修正二阶矩的偏差：$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$

计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta+\sqrt{\hat{\boldsymbol{r}}}}\odot\hat{\boldsymbol{s}}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

类似于动量

偏置修正

$\rho_1 = 0.9$
$\rho_2 = 0.999$

优化算法小结

➢大部分优化算法可以使用下面公式来统一描述概括：

$$\Delta\theta_t = -\frac{\alpha_t}{\sqrt{G_t + \epsilon}} M_t,$$

$$G_t = \psi(\mathbf{g}_1, \cdots, \mathbf{g}_t),$$

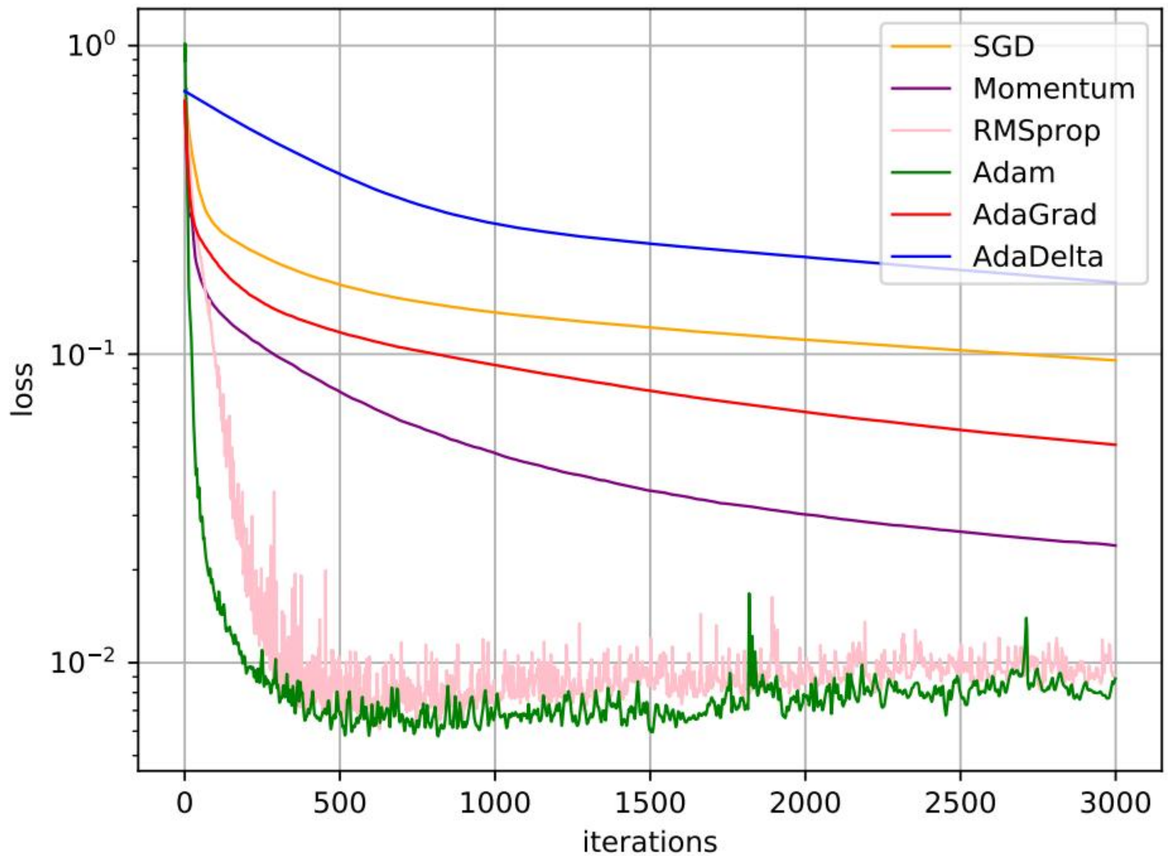
$$M_t = \phi(\mathbf{g}_1, \cdots, \mathbf{g}_t),$$

类别		优化算法
学习率调整	固定衰减学习率	分段常数衰减、逆时衰减、(自然)指数衰减、余弦衰减
	周期性学习率	循环学习率、SGDR
	自适应学习率	AdaGrad、RMSprop、AdaDelta
梯度估计修正		动量法、Nesterov 加速梯度、梯度截断
综合方法		Adam≈动量法+RMSprop

g_t 为第t步的梯度
α_t为第t步的学习率

优化算法小结

➢以下为这几种优化方法在 MNIST 数据集上收敛性的比较（学习率为0.001，批量大小为128）



参数初始化

参数初始化

- 参数不能初始化为0！为什么？
  - 对称权重问题：如果参数都初始化为0，在第一遍前向计算时，所有的隐藏层神经元的激活值都相同；在反向传播时，所有权重的更新也都相同，这样会导致隐藏层神经元没有区分性

- 初始化方法
  - 预训练初始化（Fine-Tuning）
  - 随机初始化
  - 固定值初始化（偏置-Bias-通常用0来初始化）

随机初始化

➤Gaussian分布初始化

- 参数从一个固定均值（比如0）和固定方差（比如0.01）的Gaussian分布进行随机初始化。

$$\boldsymbol{W} = \sigma * \mathrm{np.random.randn(fan\_in,\ fan\_out)}$$

若方差为σ^2，
那么$r = \sqrt{3\sigma^2}$

$$\mathrm{var}(x) = \frac{(b-a)^2}{12}$$

➤均匀分布初始化

- 参数可以在区间$[-r, r]$内采用均匀分布进行初始化

$$\boldsymbol{W} = \mathrm{r} * \mathrm{np.random.rand(fan\_in,\ fan\_out)}$$

在浅层网络时效果不错，但是深层网络就有较大问题

随机初始化的问题

➢如果参数范围取的太小

- 会导致神经元的输出过小，经过多层之后信号就慢慢消失
- 使得 Sigmoid 型激活函数丢失非线性的能力

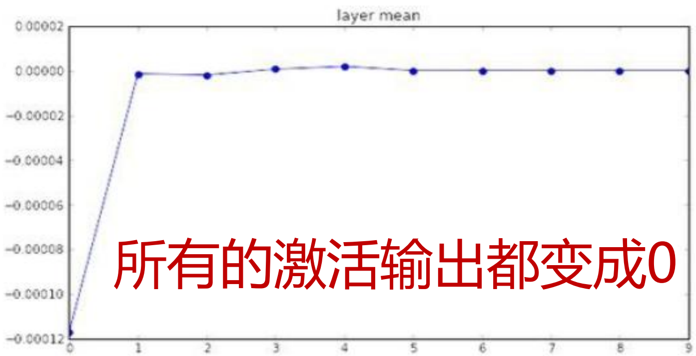
➢如果参数范围取的太大

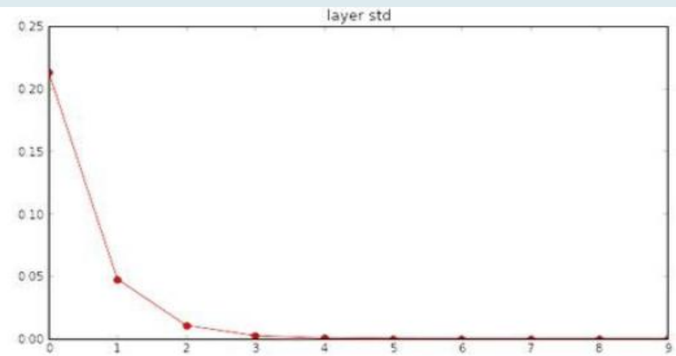
- 对于Sigmoid型激活函数，激活值变得饱和，梯度接近于0，从而导致梯度消失问题

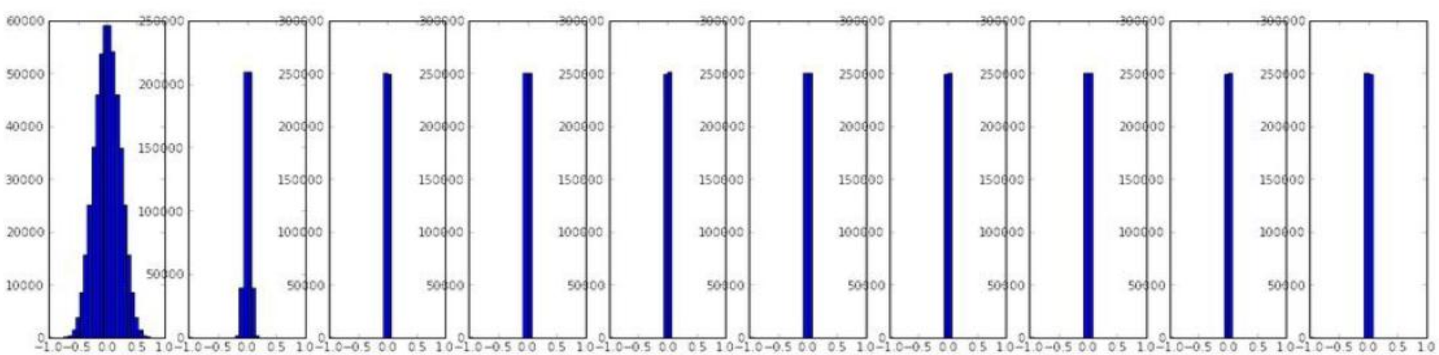
随机初始化的问题

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```

- 看看激活函数的输出统计
- 10层，每层500个神经元，用tanh非线性函数，用小随机数初始化





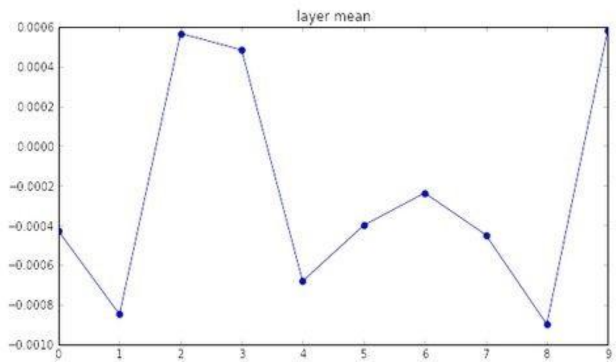


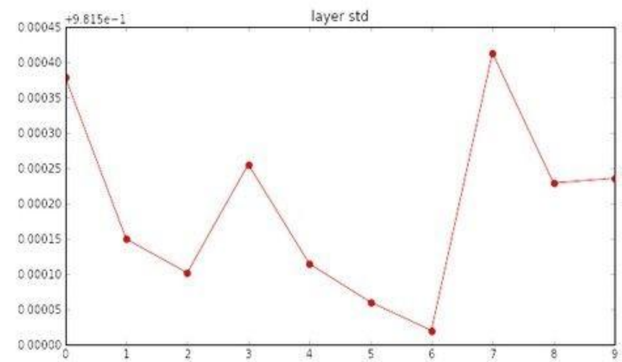
$$\boldsymbol{W} = 0.01 * \text{np.random.randn(fan\_in, fan\_out)}$$

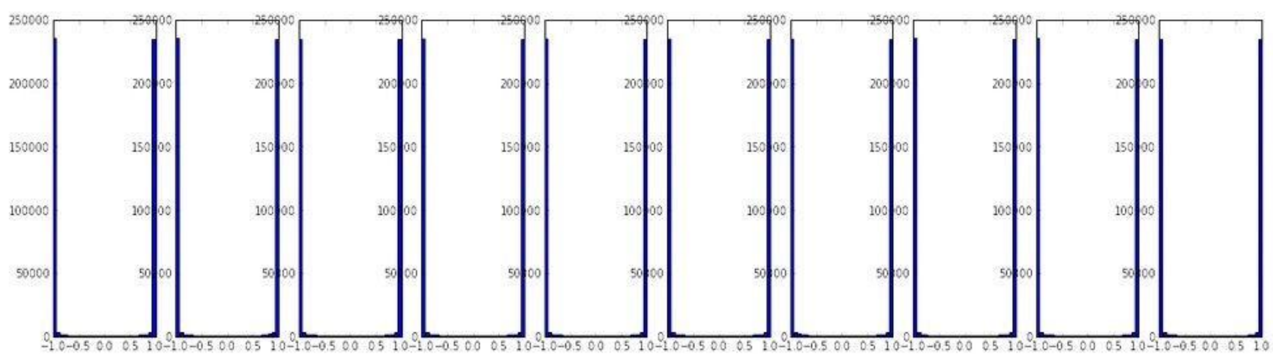
随机初始化的问题

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

激活函数输出饱和，输出1或者-1，**梯度为0**







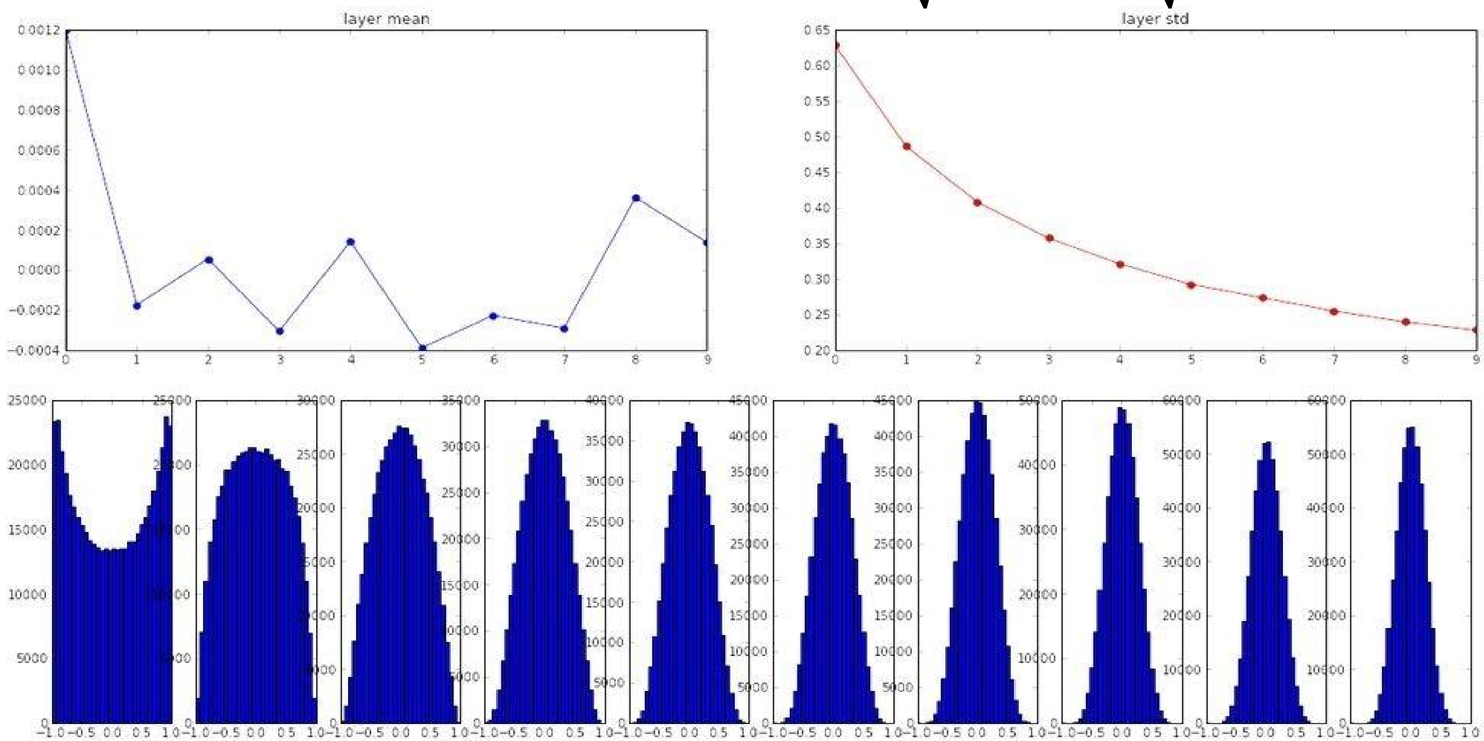
$$\boldsymbol{W} = 1 * \mathrm{np.random.randn(fan\_in,\ fan\_out)}$$

Xavier 初始化

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

也可以为均匀分布

$$U\left(-\frac{\sqrt{3}}{\sqrt{\text{fan\_in}}}, \frac{\sqrt{3}}{\sqrt{\text{fan\_in}}}\right)$$



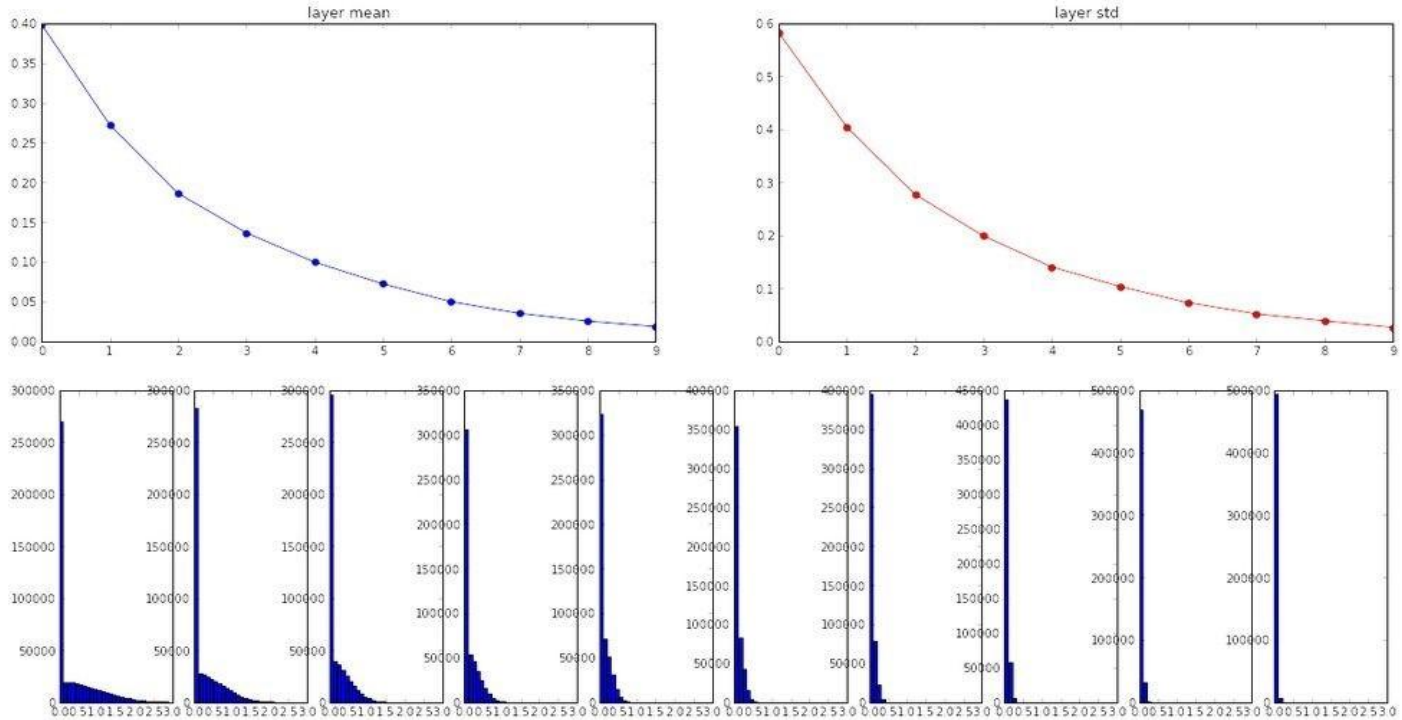
$$\boldsymbol{W} = \text{np.random.randn(fan\_in, fan\_out)/np.sqrt(fan\_in)}$$

[Glorot et al., 2010]

Xavier 初始化的问题

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

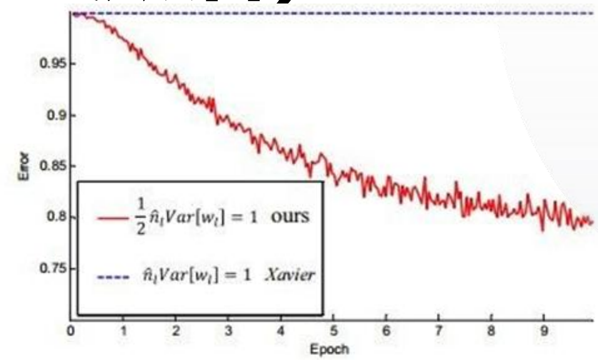
当使用ReLU时，
方差又接近0

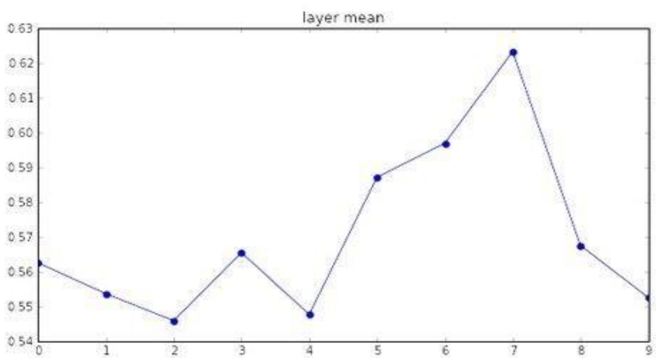


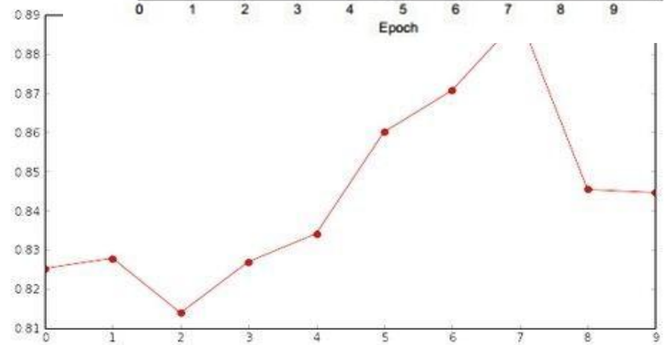
$$\boldsymbol{W} = \mathrm{np.random.randn(fan\_in,\ fan\_out)/np.sqrt(fan\_in)}$$

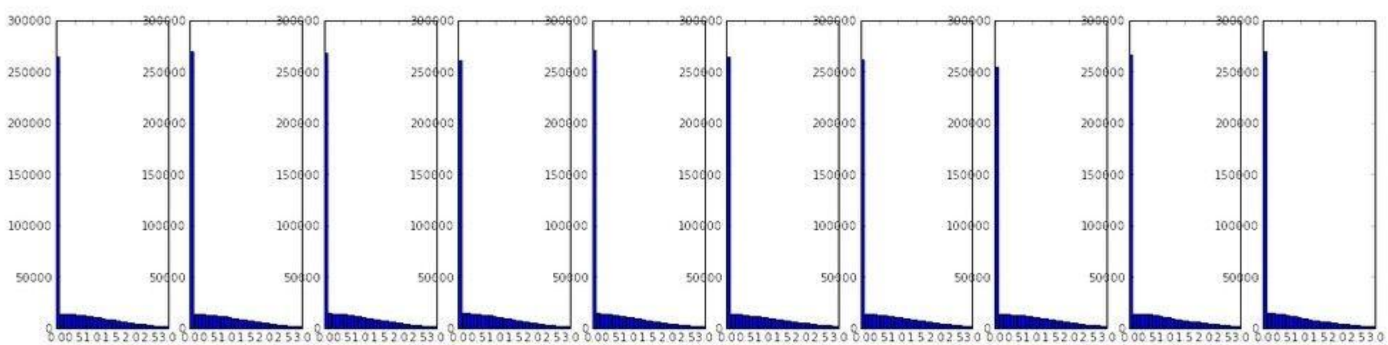
He 初始化（针对ReLU激活）

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```









$$\boldsymbol{W} = \text{np.random.randn(fan\_in, fan\_out)/np.sqrt(}\mathbf{2}\text{/fan\_in)}$$

[He et al., 2015]

权重初始化的总结

初始化方法	激活函数	均匀分布 $[-r, r]$	高斯分布 $\mathcal{N}(0, \sigma^2)$
Xavier 初始化	Logistic	$r = 4\sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2 = 16 \times \frac{2}{M_{l-1}+M_l}$
Xavier 初始化	Tanh	$r = \sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2 = \frac{2}{M_{l-1}+M_l}$
He 初始化	ReLU	$r = \sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2 = \frac{2}{M_{l-1}}$

偏置设置的一些方案

- 通常情况设偏置为0
- 输出单元的偏置可以用softmax(b)=c来初始化，c为类别分布
- ReLU的偏置为0.1
- 门控结构中，门单元的偏置设为门输出接近1
  - 单元输出u，门单元$h \in [0,1]$， 输出为uh
  - $h \approx 1$

权重初始化的参考文献

1. ***Understanding the difficulty of training deep feedforward neural networks*** by Glorot and Bengio, 2010
2. ***Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*** by Saxe et al, 2013
3. ***Random walk initialization for training very deep feedforward networks*** by Sussillo and Abbott, 2014
4. ***Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*** by He et al., 2015
5. ***Data-dependent Initializations of Convolutional Neural Networks*** by Krähenbühl et al., 2015
6. ***All you need is a good init***, by Mishkin and Matas, 2015