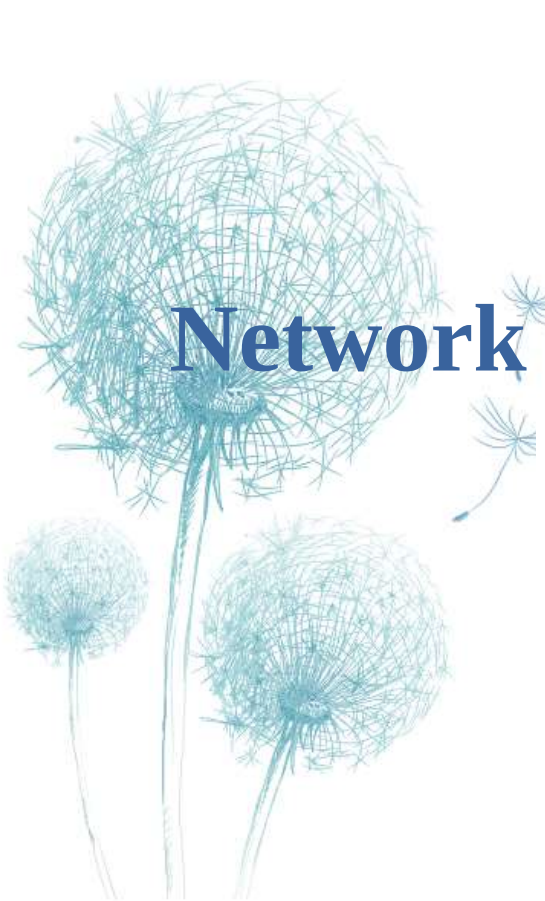
《神经网络与深度学习》



网络优化与正则化

Network Optimization and Regularization

王逍

xiaowang@ahu.edu.cn

计算机科学与技术学院 安徽大学

目 录

1. 网络优化

什么是网络优化?

- 网络优化是指寻找一个神经网络模型来使得经验（或结构）风险最小化的过程，包括模型选择以及参数学习等.

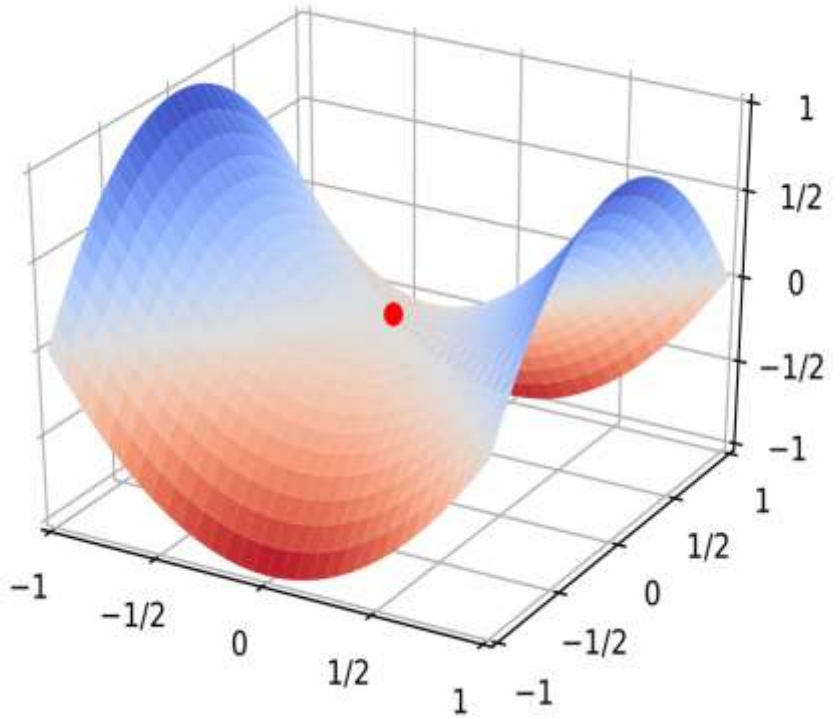
- 难点?

  - 结构差异大
    - 没有通用的优化算法
    - 超参数多
  - 非凸优化问题
    - 参数初始化
    - 逃离局部最优
  - 梯度消失（爆炸）问题

高维空间的非凸优化问题

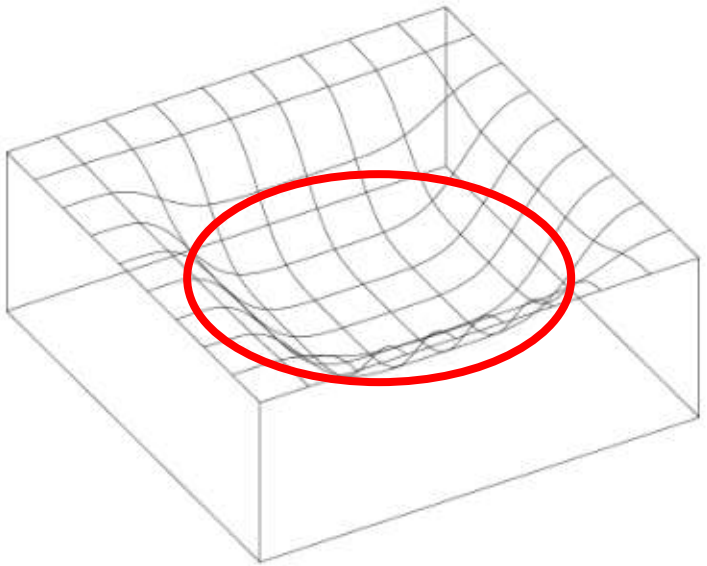
- 鞍点（Saddle Point）
  - 驻点（Stationary Point）：梯度为 0 的点。
  - 在高维空间中，大部分驻点都是鞍点。

基于梯度下降的优化方法会在**鞍点**附近接近于停滞，很难从这些鞍点中逃离．因此，随机梯度下降对于高维空间中的非凸优化问题十分重要，**通过在梯度方向上引入随机性，可以有效地逃离鞍点．**

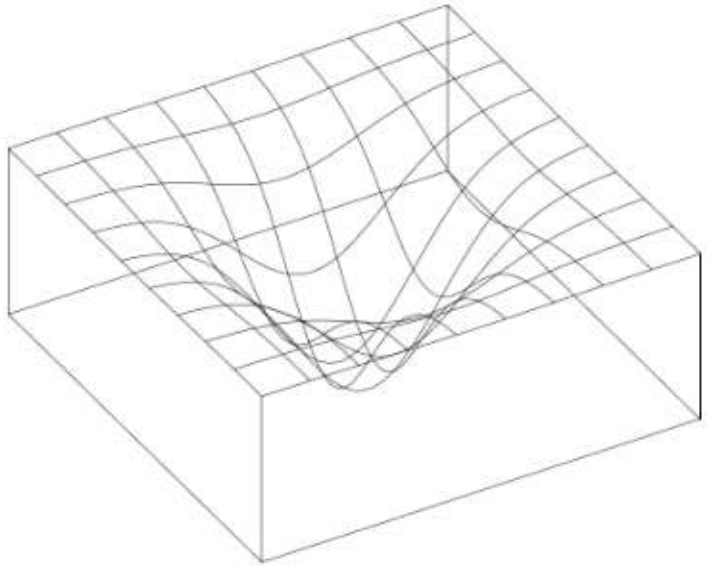


高维空间的非凸优化问题

- 平坦最小值（Flat Minima）
  - 一个平坦最小值的邻域内，所有点对应的训练损失都比较接近
  - 大部分的局部最小解是等价的
  - 局部最小解对应的训练损失都可能非常接近于全局最小解对应的训练损失



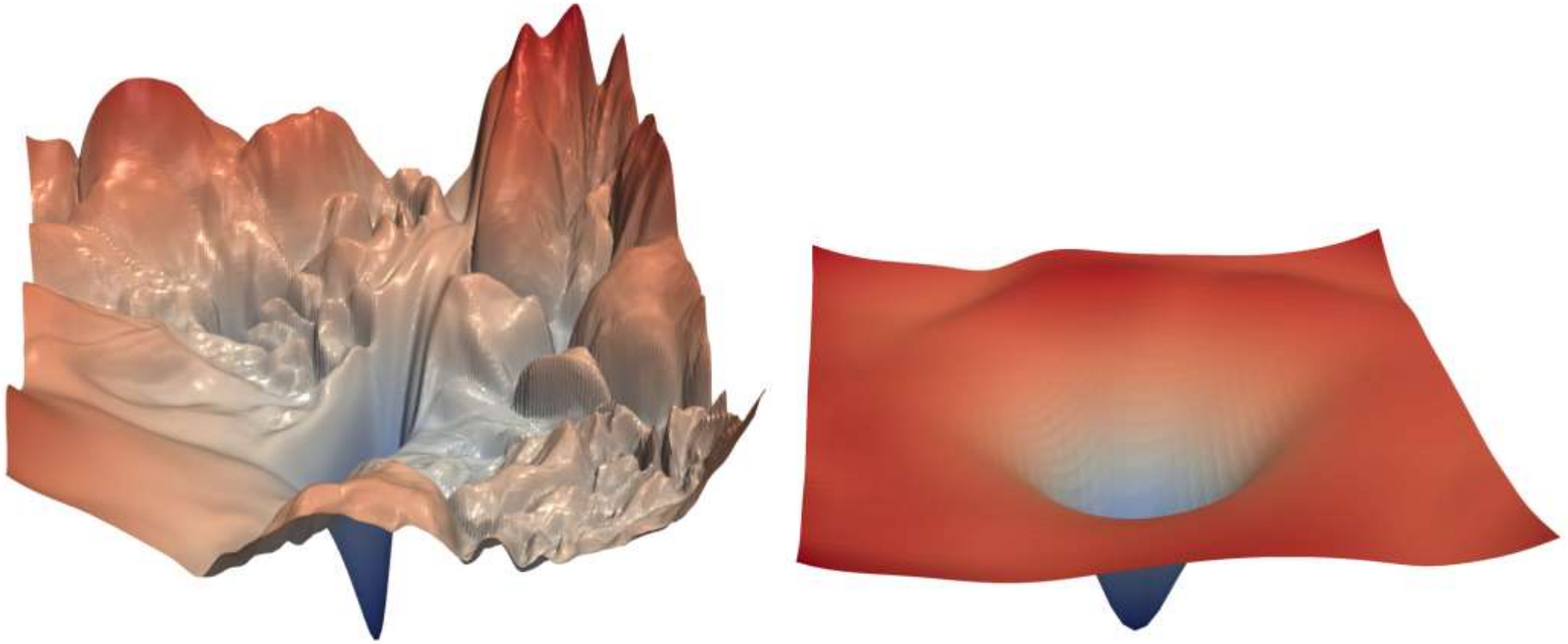
(a) 平坦最小值



(b) 尖锐最小值

Visualizing the Loss Landscape of Neural Nets

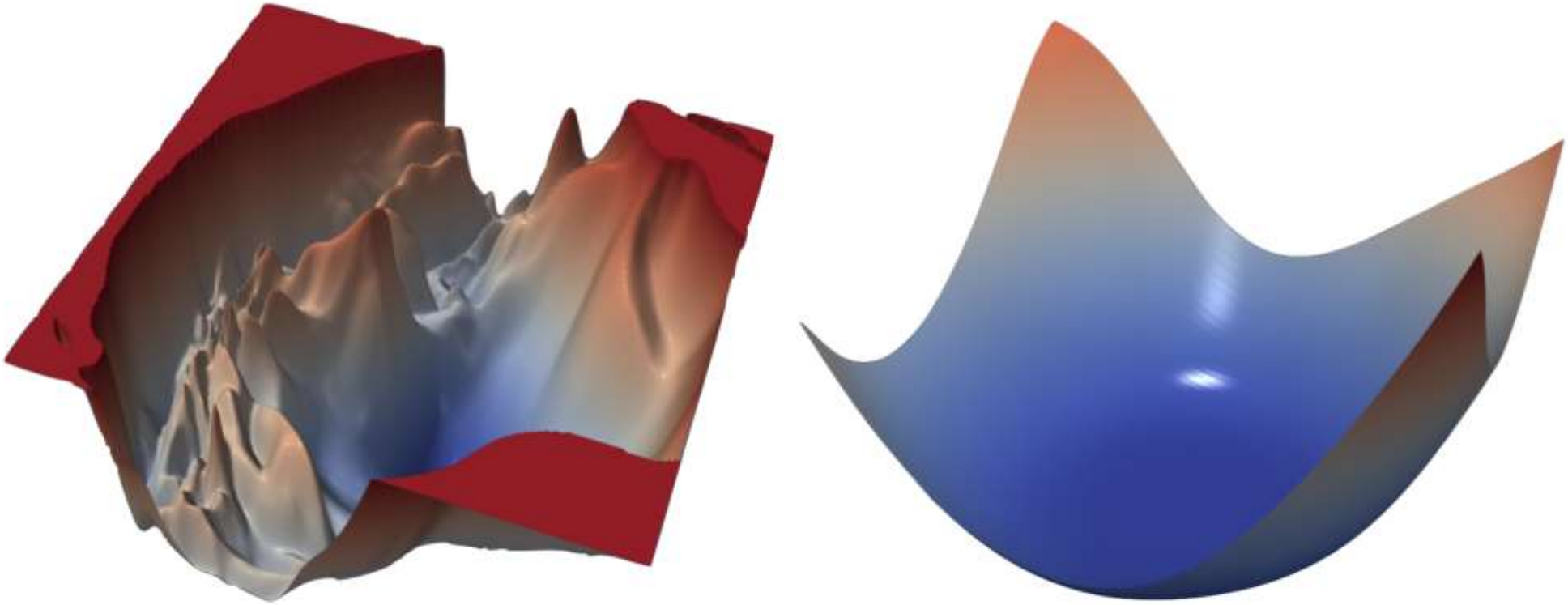
使用了随机线性投影（Random Linear Projections）来可视化神经网络的损失函数表面。



(a) without skip connections　　(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Visualizing the Loss Landscape of Neural Nets



(a) ResNet-110, no skip connections (b) DenseNet, 121 layers

Figure 4: The loss surfaces of ResNet-110-noshort and DenseNet for CIFAR-10.

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. https://arxiv.org/pdf/1712.09913.pdf

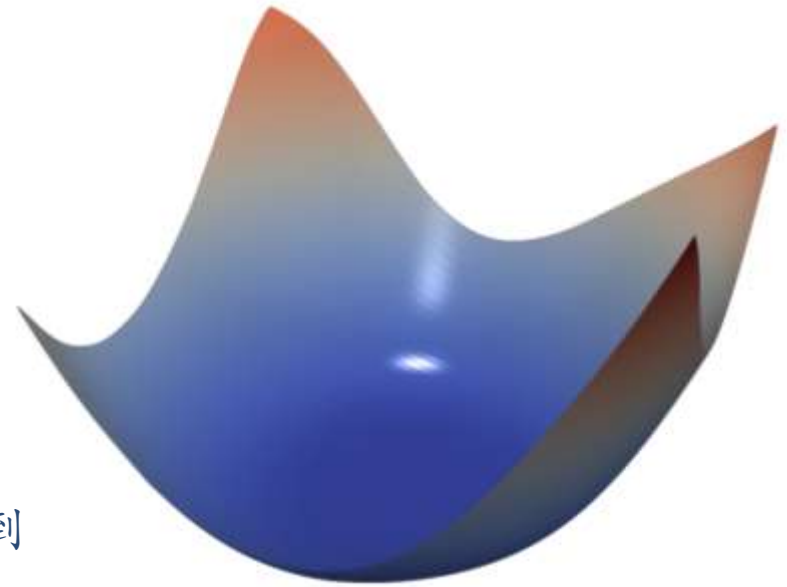
高维空间的非凸优化问题

- 在一个平坦最小值的邻域内，所有点对应的训练损失都比较接近，表明我们在训练神经网络时，不需要精确地找到一个局部最小解，只要在一个局部最小解的邻域内就足够了.

- 什么是神经网络的解?

- 平坦最小值通常被认为和模型泛化能力有一定的关系.

一般 而言，当一个模型收敛到一个平坦的局部最小值时，其鲁棒性会更好，即微小的参数变动不会剧烈影响模型能力；而当一个模型收敛到一个尖锐的局部最小值 时，其鲁棒性也会比较差.



高维空间的非凸优化问题

局部最小解的等价性

- 在非常大的神经网络中，大部分的局部最小解是等价的，它们在测试集上性能都比较相似.

- 神经网络有一定概率收敛于比较差的局部最小值，但随着网络规模增加，网络陷入比较差的局部最小值的概率会大大降低. 在训练神经网络时，我们通常没有必要找全局最小值，这反而可能导致过拟合.

神经网络优化的改善方法

- 更有效的优化算法来提高优化方法的效率和稳定性
  - 动态学习率调整
  - 梯度估计修正
- 更好的参数初始化方法、数据预处理方法来提高优化效率
- 修改网络结构来得到更好的优化地形
  - 优化地形（Optimization Landscape）指在高维空间中损失函数的曲面形状。好的优化地形通常比较平滑
  - 使用 ReLU 激活函数、残差连接、逐层归一化等
- 使用更好的超参数优化方法

2. 优化算法改进

优化算法：随机梯度下降

算法 2.1　随机梯度下降法

输入: 训练集 $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，验证集 $\mathcal{V}$，学习率 α

随机初始化 θ;

repeat

　对训练集 $\mathcal{D}$ 中的样本随机排序;

　for $n = 1 \cdots N$ **do**

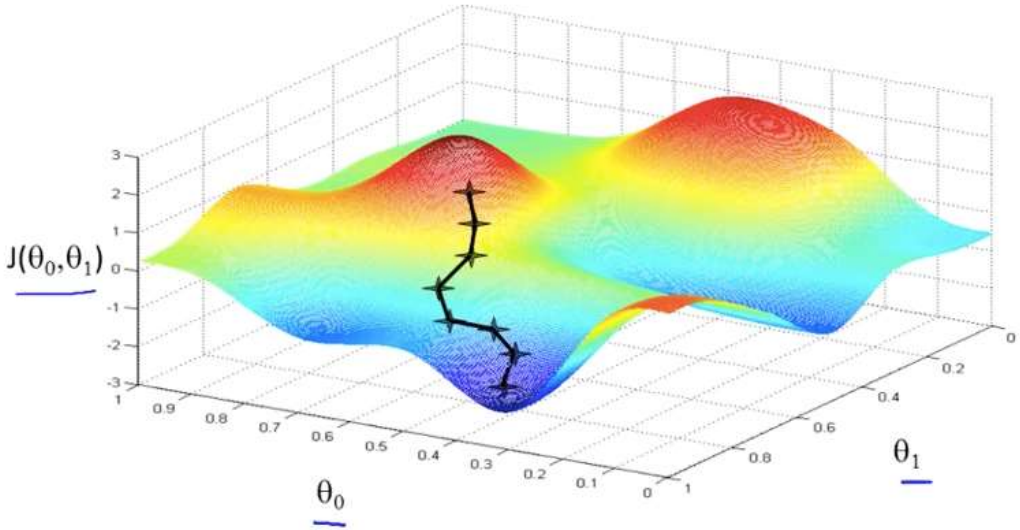
　　从训练集 $\mathcal{D}$ 中选取样本 $(\boldsymbol{x}^{(n)}, y^{(n)})$;

　　$\theta \leftarrow \theta - \alpha \dfrac{\partial \mathcal{L}(\theta; x^{(n)}, y^{(n)})}{\partial \theta}$;

　end

until 模型 $f(\boldsymbol{x}; \theta)$ 在验证集 $\mathcal{V}$ 上的错误率不再下降;

输出: θ



优化算法：小批量随机梯度下降 Mini-Batch

- 选取K个训练样本$\{\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\}_{k=1}^{\mathrm{K}}$，计算偏导数

$$\mathbf{g}_t(\theta) = \frac{1}{K} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{S}_t} \frac{\partial \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x};\theta))}{\partial \theta}$$

- 定义梯度

$$\mathbf{g}_t \triangleq \mathbf{g}_t(\theta_{t-1})$$

- 更新参数

$$\theta_t \leftarrow \theta_{t-1} - \alpha \mathbf{g}_t$$

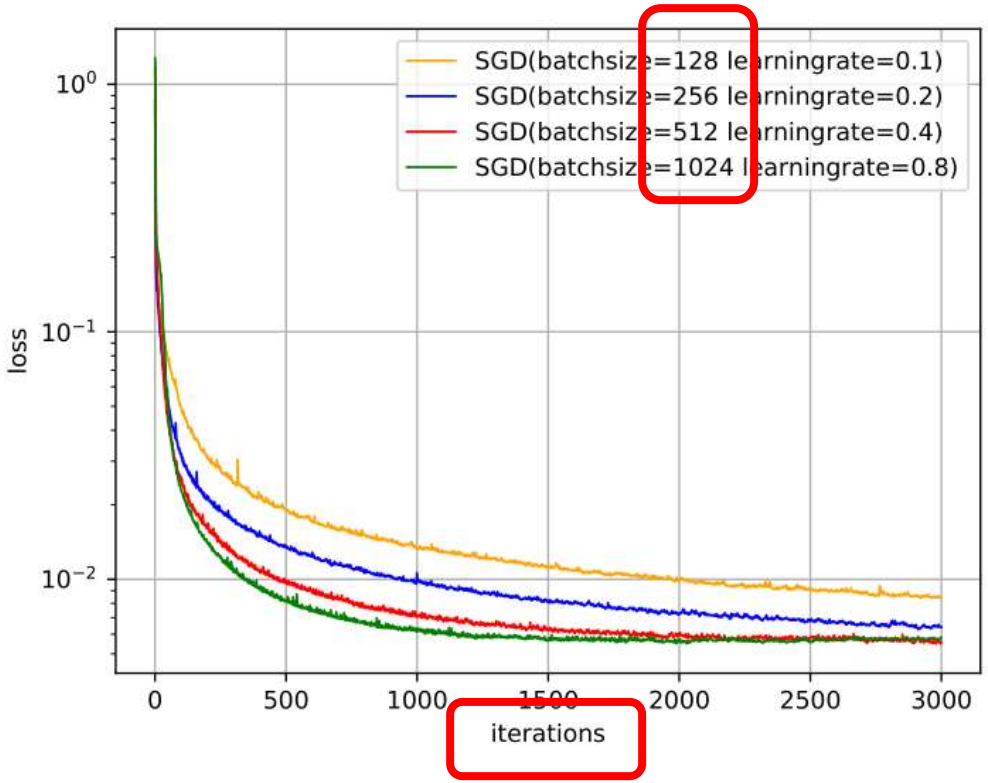
其中$\alpha > 0$为学习率

几个关键因素：

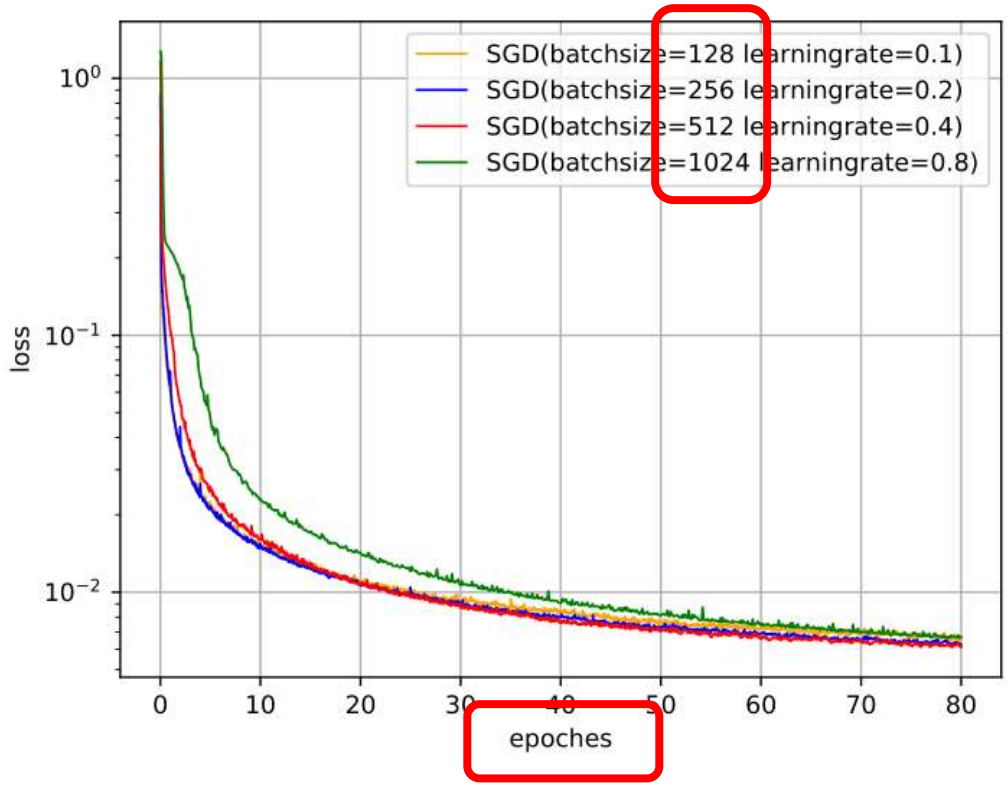
- 小批量样本数量
- 梯度
- 学习率

批量大小的影响

- 批量大小不影响随机梯度的期望，但是会影响随机梯度的方差.
  - 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

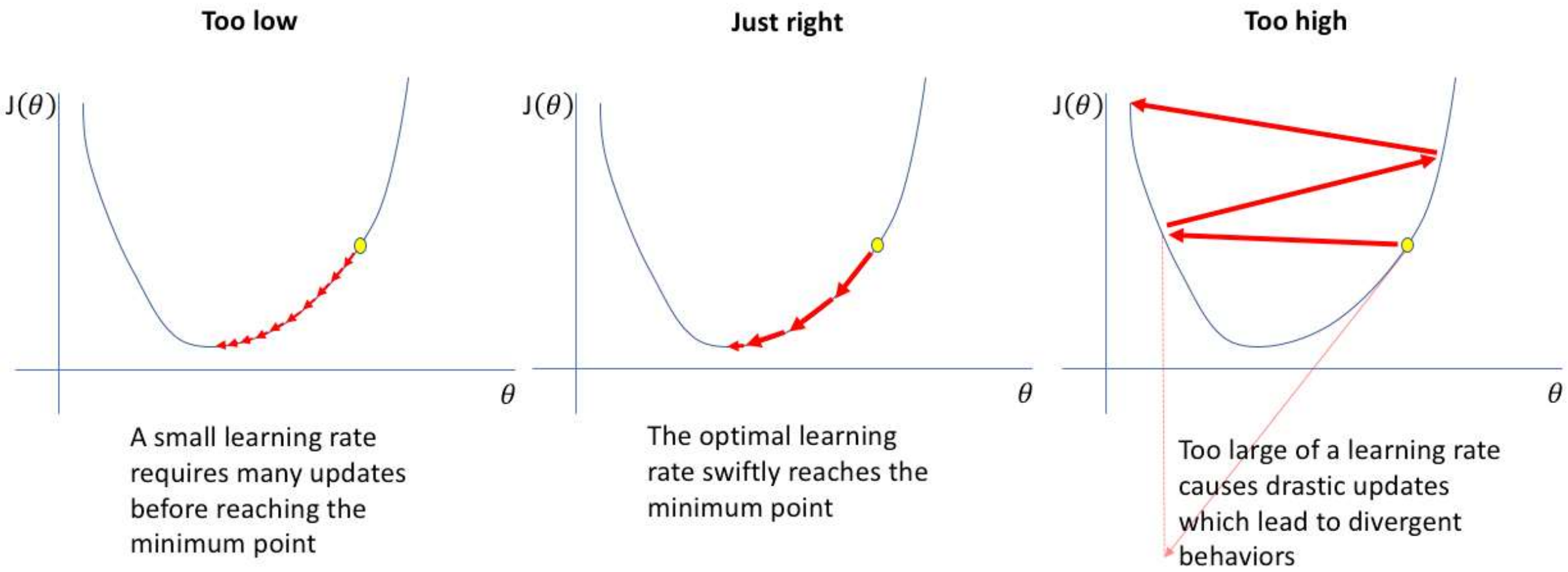
批量大小的影响

- 学习率通常要随着批量大小的增大而相应地增大.
  - 一个简单有效的方法是**线性缩放规则（Linear Scaling Rule）**[Goyal et al., 2017]: 当批量大小增加 m 倍时，学习率也增加 m 倍.
  - 线性缩放规则往往在批量大小比较小时适用，当批量大小非常大时，线性缩放会使得训练不稳定.

- 迭代（Iteration）与回合（Epoch）
  - 每一次小批量更新为一次迭代（Iteration），
  - 所有训练集的样本更新一遍为一个回合（Iteration）

$$1\text{回合（Epoch）} = \left(\frac{\text{训练样本的数量}N}{\text{批量大小}K}\right) \times \text{迭代（Iteration）}.$$

学习率 (Learning Rate) 的影响



https://www.jeremyjordan.me/nn-learning-rate/

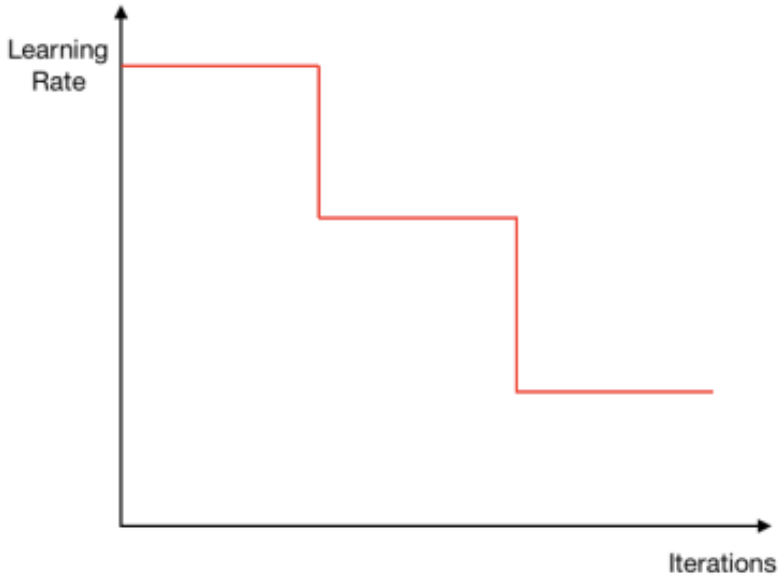
如何改进?

Reference:
1. An overview of gradient descent optimization algorithms
2. Optimizing the Gradient Descent

- 标准的（小批量）梯度下降
- 学习率
  - 学习率衰减，学习率预热，周期性学习率调整
  - 自适应调整学习率的方法：Adagrad, Adadelta, RMSprop
- 梯度
  - Momentum
    - 计算负梯度的“加权移动平均”作为参数的更新方向
  - Nesterov accelerated gradient
  - 梯度截断

学习率衰减

- 分段常数衰减（Piecewise Constant Decay）



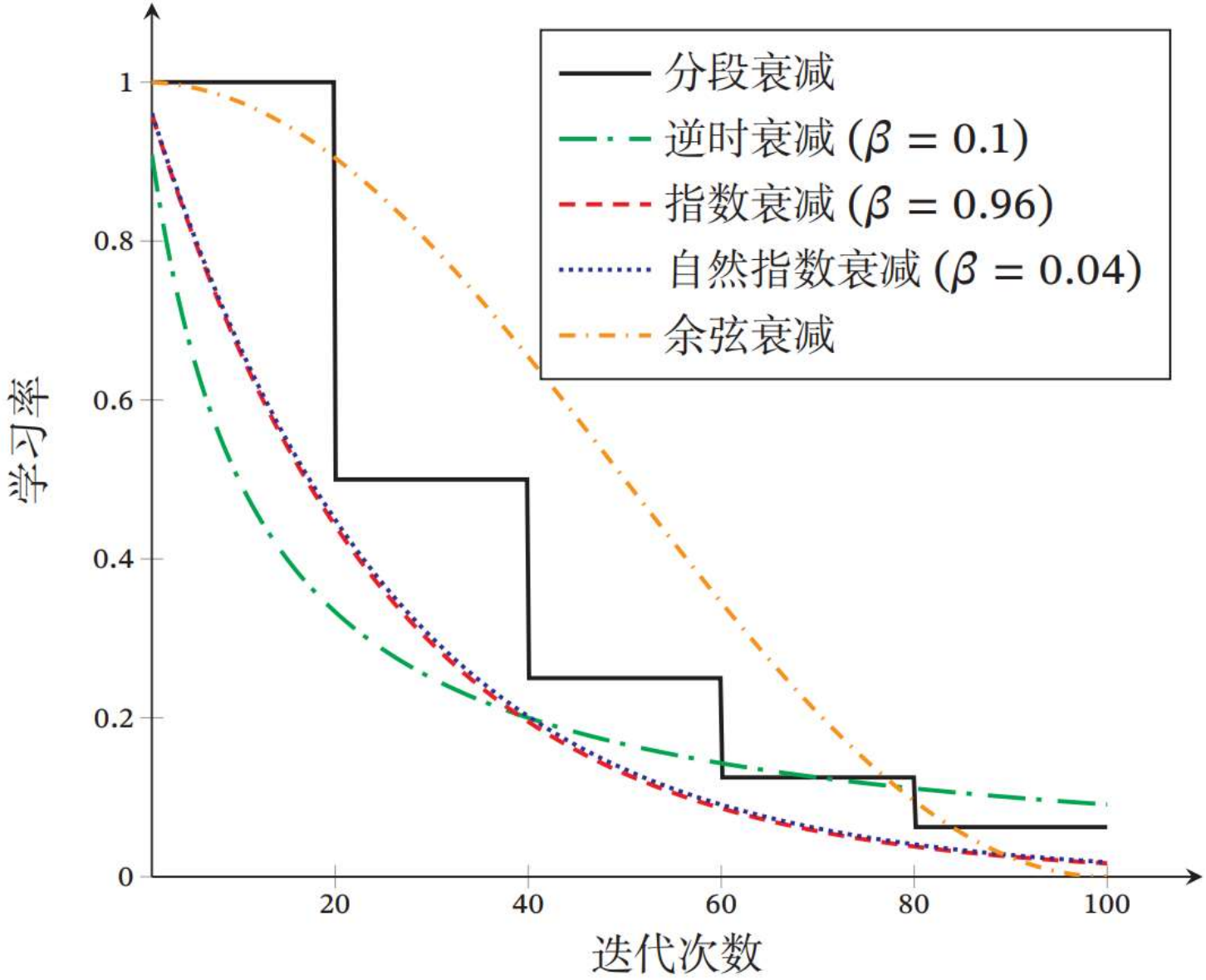
- 逆时衰减（Inverse Time Decay） $\alpha_t = \alpha_0 \frac{1}{1+\beta \times t},$
- 指数衰减（Exponential Decay） $\alpha_t = \alpha_0 \beta^t,$
- 自然指数衰减（Natural Exponential Decay）

$$\alpha_t = \alpha_0 \exp(-\beta \times t),$$

- 余弦衰减（Cosine Decay）

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

学习率衰减


分段衰减
逆时衰减 ($\beta = 0.1$)
指数衰减 ($\beta = 0.96$)
自然指数衰减 ($\beta = 0.04$)
余弦衰减
学习率
迭代次数
1
0.8
0.6
0.4
0.2
0
20
40
60
80
100

学习率预热 Learning Rate WarmUp

在小批量梯度下降法中，当批量大小的设置比较大时，通常需要比较大的学习率. 但在刚开始训练时，由于参数是随机初始化的，梯度往往也比较大，再加上 比较大的初始学习率，会使得训练不稳定.

为了提高训练稳定性，我们可以**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再恢复到初始的学习率，这种方法称为学习率预 热（Learning Rate Warmup）.**

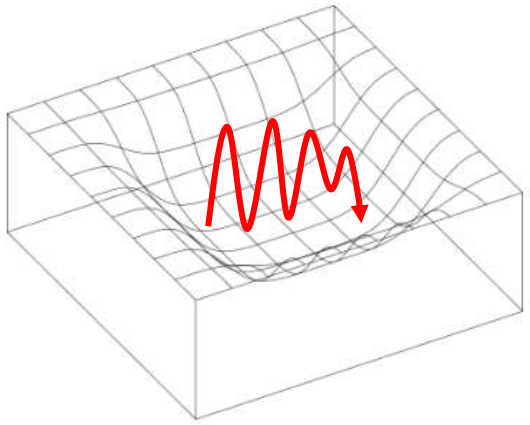
【案例】逐渐预热（Gradual Warmup）[Goyal et al., 2017]. 假设预热的迭代次数为T'，初始学习率为α_0，在预热过程中，每次更新的学习率为:

$$\alpha'_t = \frac{t}{T'}\alpha_0, \qquad 1 \leq t \leq T'.$$

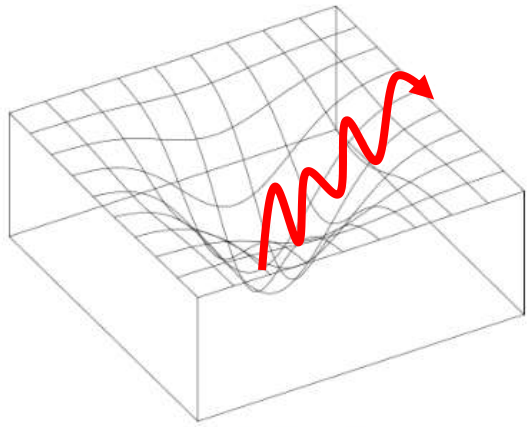
周期性学习率调整 Cyclical Learning Rates

为了使得梯度下降法能够逃离鞍点或尖锐最小值，一种经验性的方式是**在训练过程中周期性地增大学习率.**

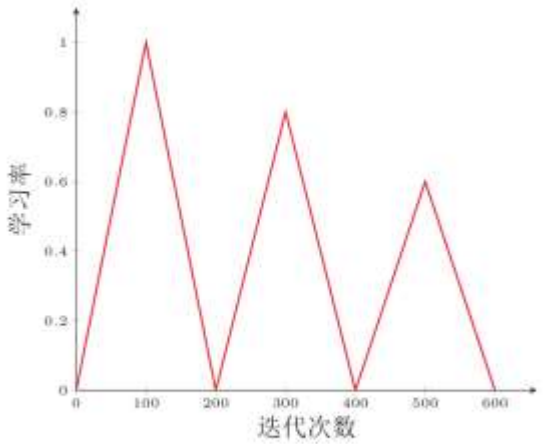
- 当参数处于尖锐最小值附近时，增大学习率有 助于逃离尖锐最小值;
- 当参数处于平坦最小值附近时，增大学习率依然有可能在 该平坦最小值的吸引域（Basin of Attraction）内.



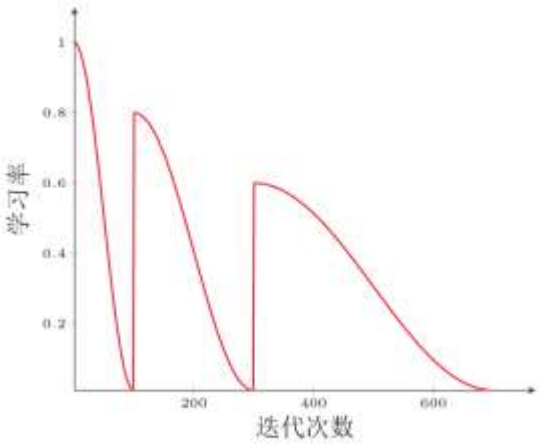
(a) 平坦最小值



(b) 尖锐最小值

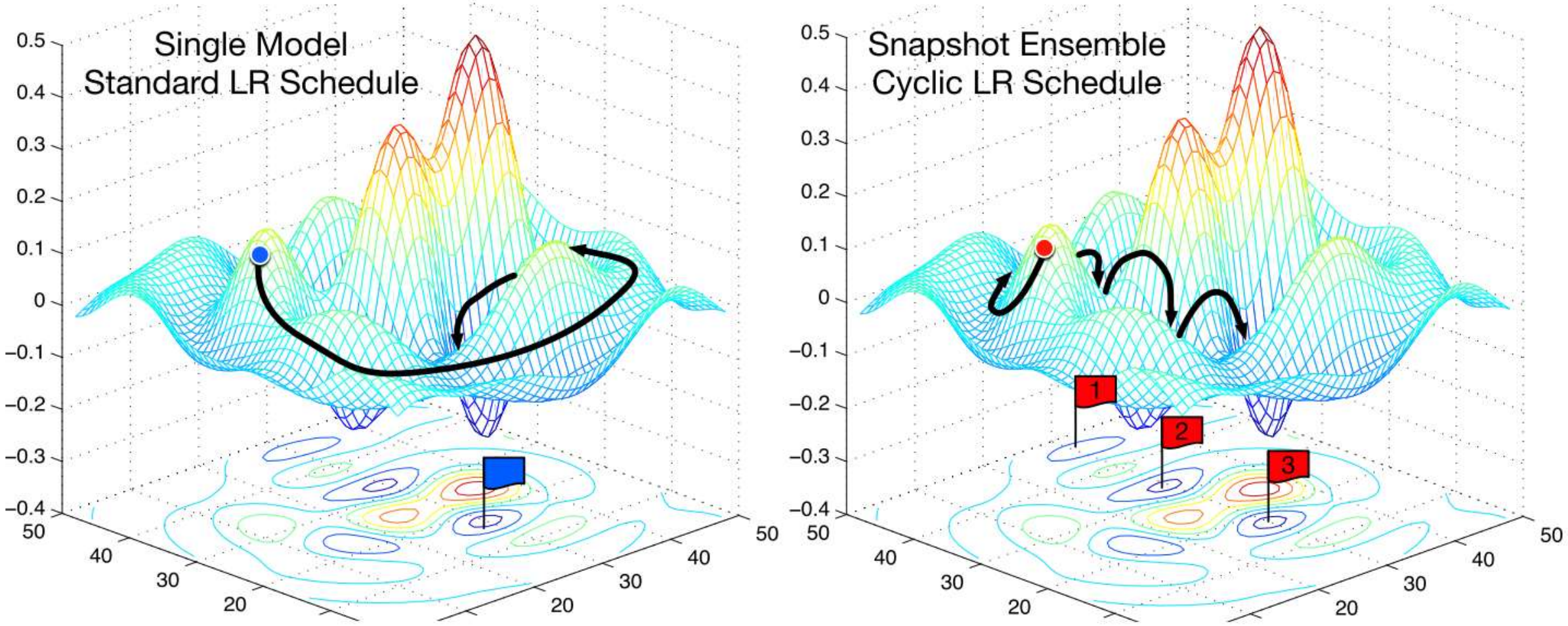


(a) 三角循环学习率



(b) 带热重启的余弦衰减

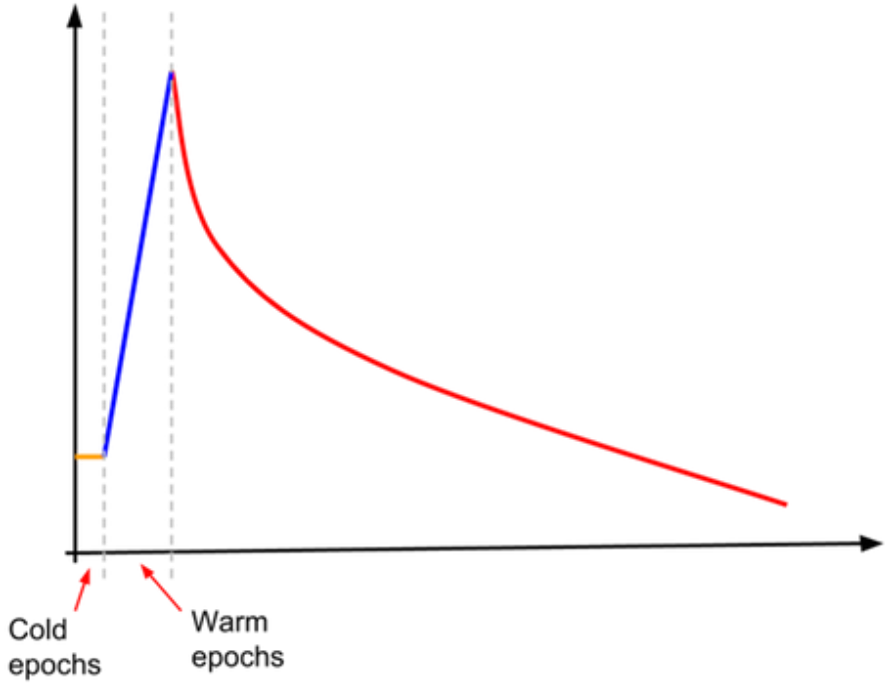
周期性学习率调整 Cyclical Learning Rates



Others

- Don't Decay the Learning Rate, Increase the Batch Size
  - https://openreview.net/pdf?id=B1Yy1BxCZ

- Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
  - Warmup
  - https://arxiv.org/abs/1706.02677



自适应学习率

AdaGrad算法（Adaptive Gradient Algorithm）[Duchi et al., 2011]是借鉴 $\ell 2$ 正则化的思想，每次迭代时自适应地调整每个参数的学习率.

在第t 次迭代时，先计算每个参数梯度平方的累计值 $G_t = \sum_{\tau=1}^{t} \mathbf{g}_\tau \odot \mathbf{g}_\tau,$

AdaGrad算法的参数更新差值为 $\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t,$

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.**

自适应学习率

RMSprop算法是Geoff Hinton提出的一种自适应学习率的方法[Tieleman et al., 2012]，可以在有些情况下避免AdaGrad算法中学习率不断单调下降以至于过早衰减的缺点。

RMSprop算法首先计算每次迭代梯度 $\boldsymbol{g}_t$ 平方的**指数衰减移动平均**，

$$G_t = \beta G_{t-1} + (1-\beta)\boldsymbol{g}_t \odot \boldsymbol{g}_t = (1-\beta)\sum_{\tau=1}^{t} \beta^{t-\tau}\boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau,$$

RMSprop算法的参数更新差值为

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t,$$

背景知识：指数衰减移动平均

指数衰减移动平均（Exponential Moving Average，EMA）是一种用于平滑时间序列数据的方法。它通过对数据进行加权平均来减少噪声和突发性波动，使得趋势更加明显。

在指数衰减移动平均中，新的数据点会受到更大的权重，而过去的数据点会受到指数级别的衰减，因此称之为"指数衰减"。

具体来说，对于一个**时间序列数据点 X_t**，其指数衰减移动平均值 EMA_t 可以用下面的公式计算：

$$EMA_t = \alpha \cdot X_t + (1 - \alpha) \cdot EMA_{t-1}$$

其中，$0<\alpha<1$ 是平均的衰减因子，决定了过去数据点的衰减程度。一般来说，α 越接近 1，新的数据点的权重越大，过去数据点的影响越小。

指数衰减移动平均的一个重要特性是，它可以用于动态地调整模型参数，尤其在训练神经网络等深度学习模型时，可以平滑损失函数的变化趋势，使得模型在训练过程中更加稳定。在深度学习中，通常会用指数衰减移动平均来计算模型的参数变化，特别是在优化算法（如Adam、RMSProp等）中，用于更新参数的梯度信息往往会与EMA结合在一起以平滑优化过程。

自适应学习率

Adagrad

RMSprop

Adadelta

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \boldsymbol{g}_t$$

$$G_t = \sum_{\tau=1}^{t} \boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau$$

$$G_t = \beta G_{t-1} + (1-\beta)\boldsymbol{g}_t \odot \boldsymbol{g}_t$$

$$= (1-\beta)\sum_{\tau=1}^{t} \beta^{t-\tau} \boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau$$

$$\Delta\theta_t = -\frac{\sqrt{\Delta X_{t-1}^2 + \epsilon}}{\sqrt{G_t + \epsilon}} \mathbf{g}_t$$

$$\Delta X_{t-1}^2 = \beta_1 \Delta X_{t-2}^2 + (1-\beta_1)\Delta\theta_{t-1} \odot \Delta\theta_{t-1}$$

梯度方向优化

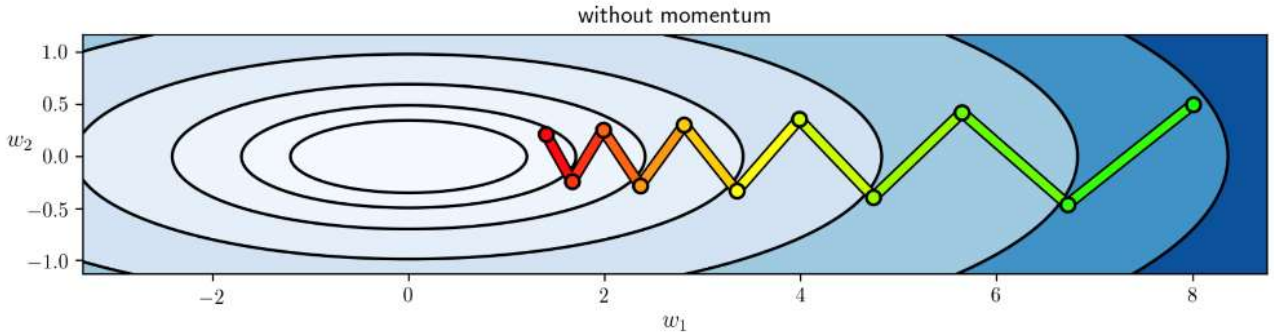
动量法（Momentum Method）

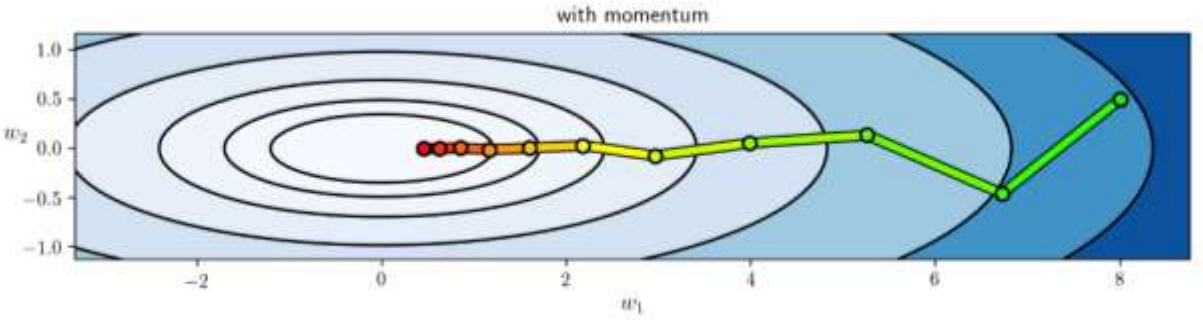
- **动量**：是模拟物理中的概念。一个物体的动量是指该物体在它运动方向上保持运动的趋势，是该物体的质量和速度的乘积。
- **动量法**：是用之前**积累动量**来代替**真正的梯度**，每次迭代的梯度可以看做加速度。

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，

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathbf{g}_t = -\alpha\sum_{\tau=1}^{t}\rho^{t-\tau}\mathbf{g}_\tau,$$

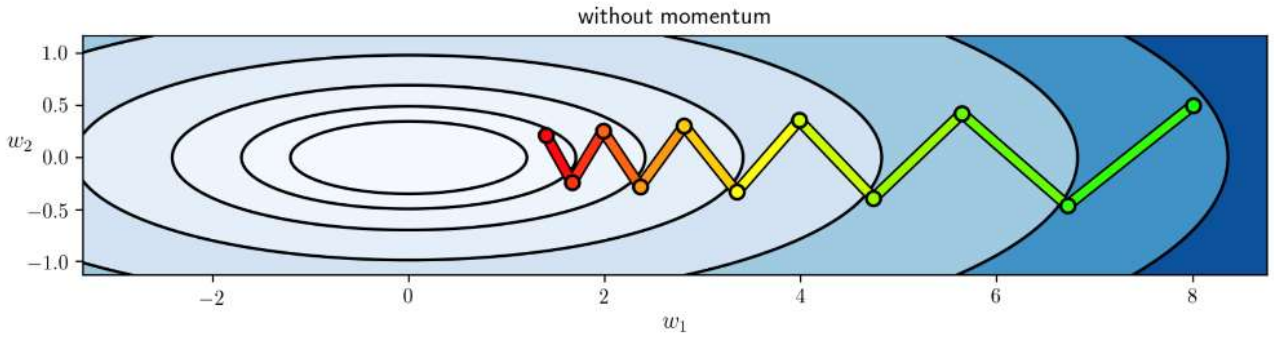
动量因子　　学习率

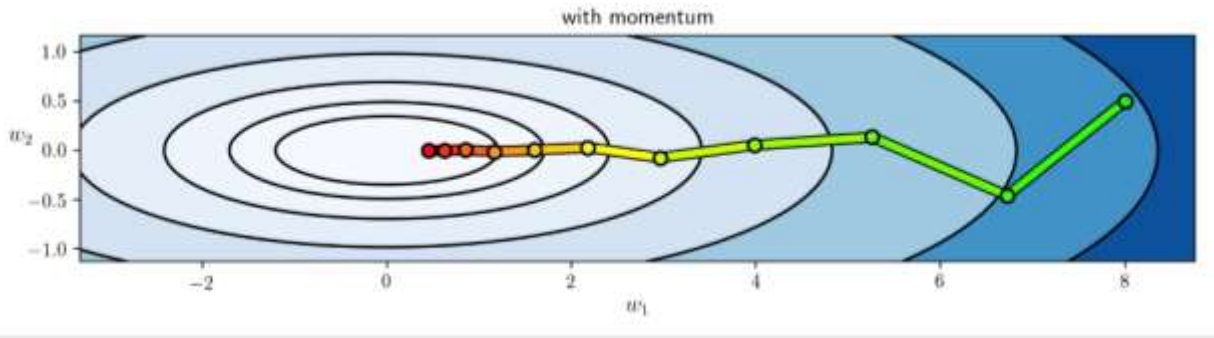




梯度方向优化

▸动量法（Momentum Method）





- **每个参数的实际更新差值取决于最近一段时间内梯度的加权平均值.**
  当某个参数在最近一段时间内的梯度方向不一致时，其真实的参数更新幅度变小；
  当在最近一段时间内的梯度方向都一致时，其真实的参数更新幅度变大，起到加速作用.

- **一般而言，在迭代初期，梯度方向都比较一致，动量法会起到加速作用，可以更快地到达最优点.**
  在迭代后期，梯度方向会不一致，在收敛值附近振荡，动量法会起到减速作用，增加稳定性.
  从某种角度来说，当前梯度叠加上部分的上次梯度，一定程度上可以近似看作**二阶梯度**.

梯度方向优化

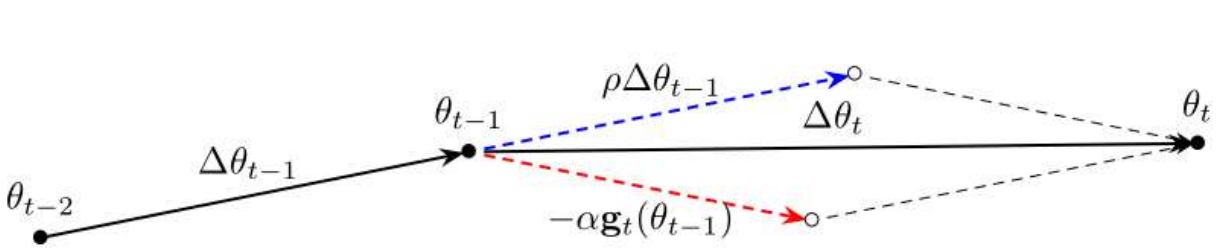
Nesterov 加速梯度（Nesterov Accelerated Gradient，NAG）

在动量法中，实际的参数更新方向 $\Delta\theta_t$ 为上一步的参数更新方向 $\Delta\theta_{t-1}$ 和当前梯度的反方向 $-\mathbf{g}_t$ 的叠加. 这样，$\Delta\theta_t$ 可以被拆分为两步进行，先根据 $\Delta\theta_{t-1}$ 更新一次得到参数 $\hat{\theta}$，再用 $-\mathbf{g}_t$ 进行更新.

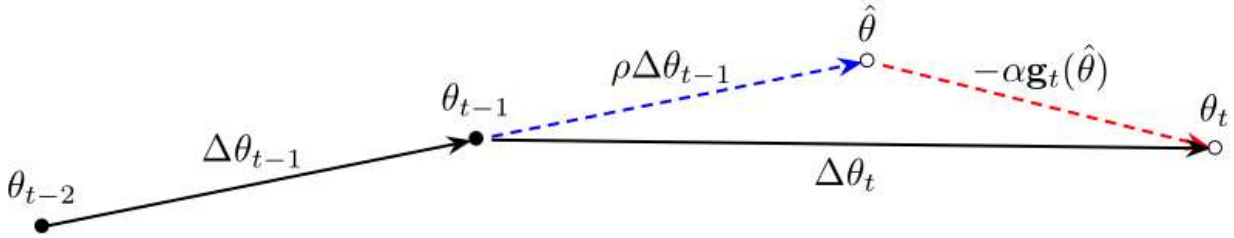
$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha g_t(\theta_{t-1} + \rho\Delta\theta_{t-1}),$$

$$\hat{\theta} = \theta_{t-1} + \rho\Delta\theta_{t-1}, \tag{7.22}$$

$$\theta_t = \hat{\theta} - \alpha\mathbf{g}_t, \tag{7.23}$$



(a) 动量法



(b) Nesterov 加速梯度

梯度方向优化+自适应学习率

- Adam算法 ≈ 动量法 + RMSprop
  - 先计算两个移动平均

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)\mathbf{g}_t,$$

$$G_t = \beta_2 G_{t-1} + (1-\beta_2)\mathbf{g}_t \odot \mathbf{g}_t,$$

1. 计算梯度平方的指数加权平均；
2. 计算梯度的指数加权平均。

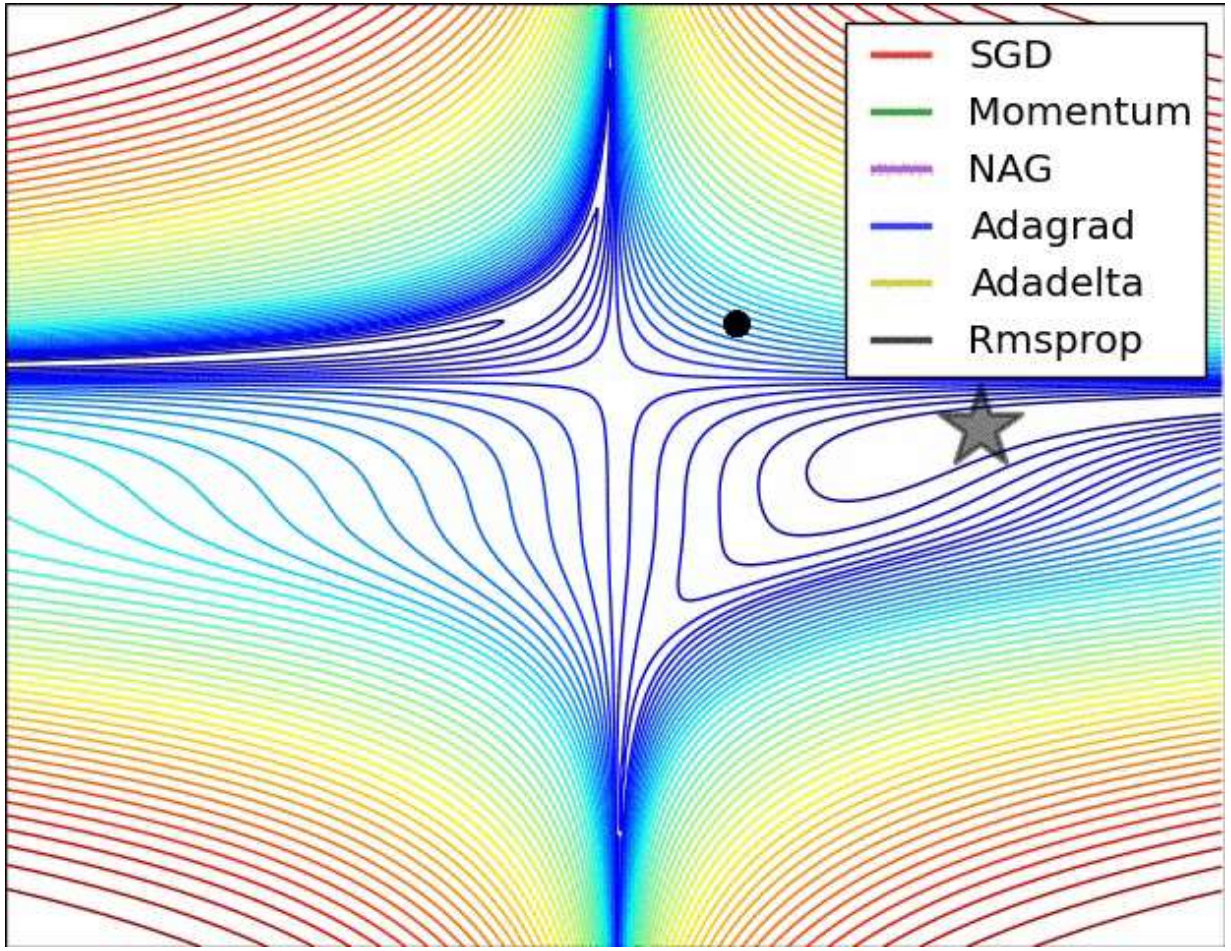
  - 偏差修正

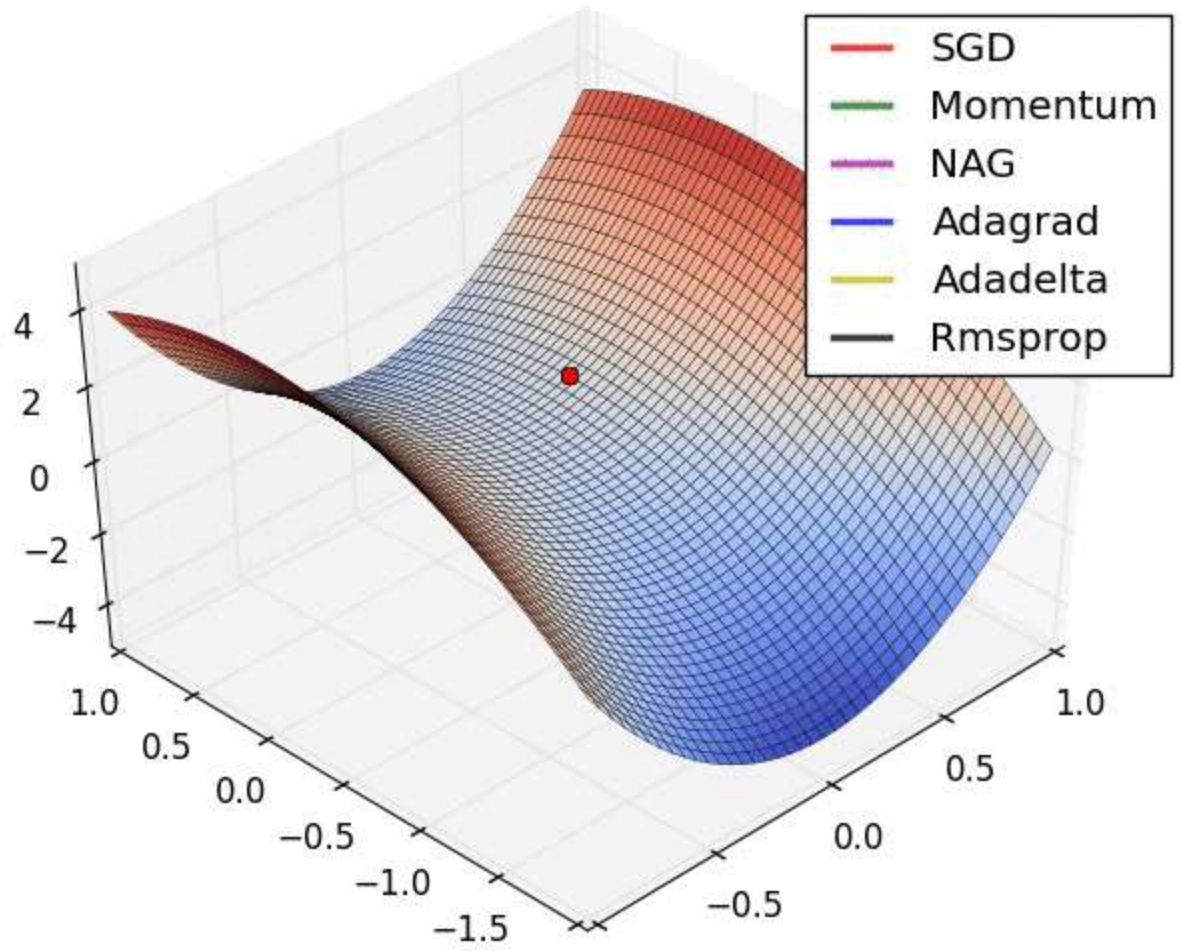
$$\hat{M}_t = \frac{M_t}{1-\beta_1^t}, \qquad \hat{G}_t = \frac{G_t}{1-\beta_2^t}.$$

  - 参数更新差值

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{\hat{G}_t + \epsilon}}\hat{M}_t$$

优化


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

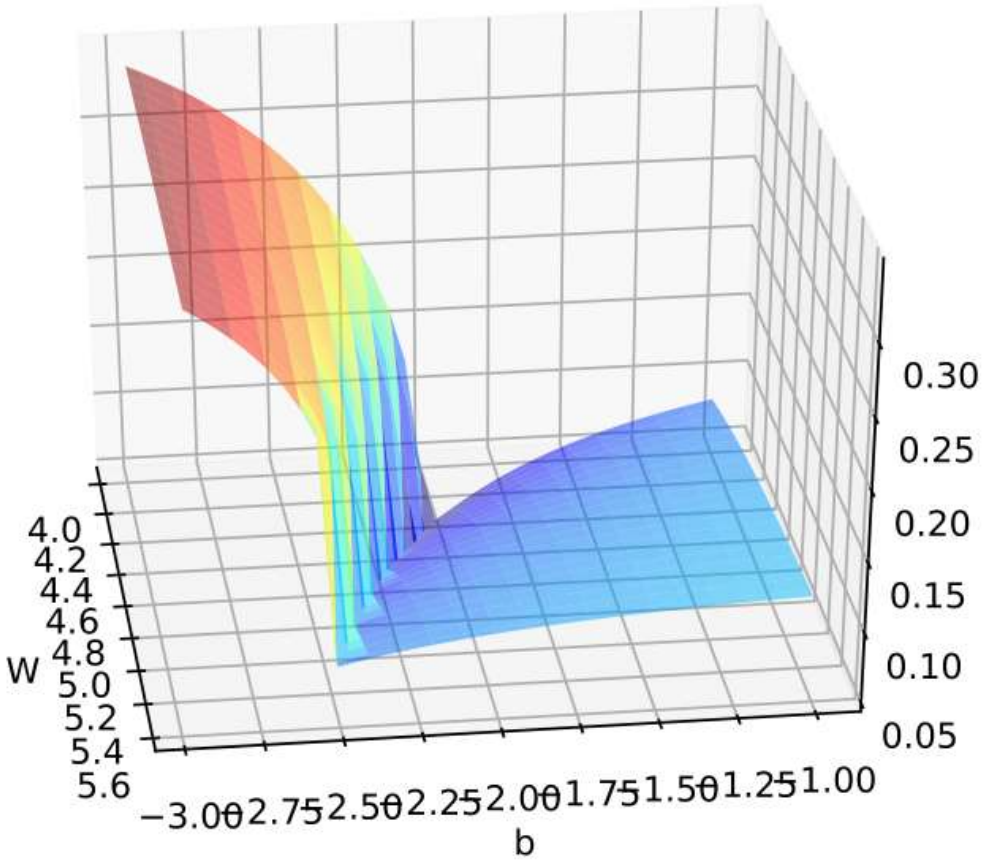

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

梯度截断

在深度神经网络或循环神经网络中，除了**梯度消失**之外，**梯度爆炸**也是影响学习效率的主要因素．**在基于梯度下降的优化过程中，如果梯度突然增大，用大的梯度更新参数反而会导致其远离最优点．**

- 梯度截断是一种比较简单的启发式方法，把梯度的模限定在一个区间，当梯度的模小于或大于这个区间时就进行截断。

右图的曲面为只有一个隐藏神经元的**循环神经网络** $h_t = \sigma(wh_{t-1} + b)$的损失函数，其中$w$ 和 b 为参数．



梯度爆炸问题示例

梯度截断

在深度神经网络或循环神经网络中，除了**梯度消失**之外，**梯度爆炸**也是影响学习效率的主要因素．**在基于梯度下降的优化过程中，如果梯度突然增大，用大的梯度更新参数反而会导致其远离最优点．**

- 梯度截断的两种方法
  - **按值截断** $\boldsymbol{g}_t = \max(\min(\boldsymbol{g}_t, b), a).$
  - **按模截断**

$$\text{如果} \|\boldsymbol{g}_t\|^2 \leq b, \text{保持} \boldsymbol{g}_t \text{不变.}$$

$$\text{如果} \|\boldsymbol{g}_t\|^2 > b, \text{令} \ \boldsymbol{g}_t = \frac{b}{\|\boldsymbol{g}_t\|}\boldsymbol{g}_t.$$

梯度爆炸问题示例

优化算法改进小结

- 大部分优化算法可以使用下面公式来统一描述概括：

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

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

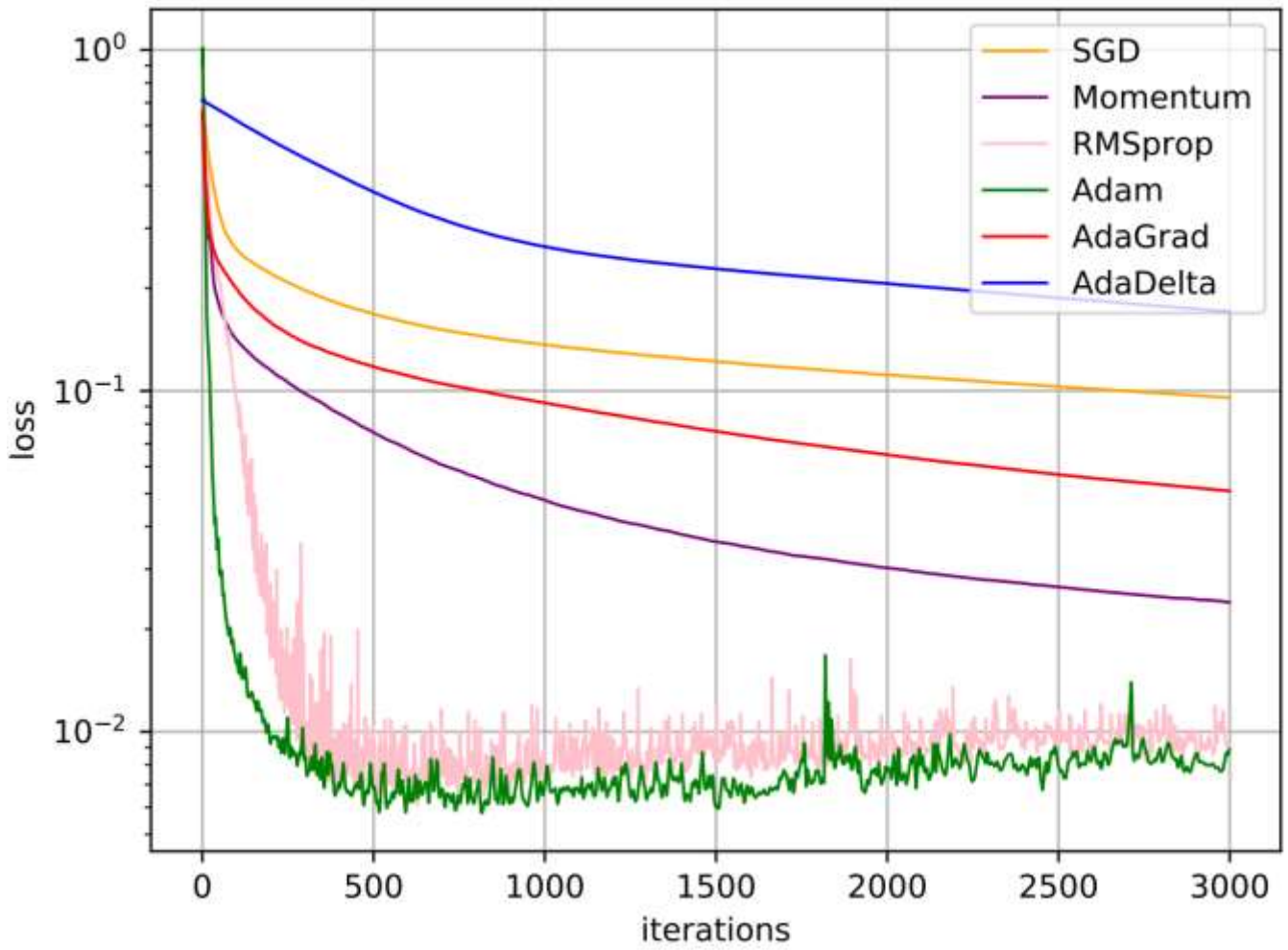
$$M_t = \phi(\boldsymbol{g}_1, \cdots, \boldsymbol{g}_t),$$

g_t 为第t步的梯度，
α_t 为第t步的学习率

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

优化算法改进小结

不同优化方法在 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/数据预处理

参数初始化

初始化方法

预训练初始化：通常情况下，一个已经在大规模数据上训练过的模型可以提供一个好的参数初始值，这种初始化方法称为预训练初始化（Pre-trained Initialization）

随机初始化：

是否可以将网络的参数全部初始化为0？为什么？

如果参数都为 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**．这种现象也称为**对称权重现象**．

固定值初始化：

- 对于一些特殊的参数，可以根据经验用一个特殊的固定值来进行初始化．

比如偏置（Bias）通常用 0 来初始化，但是有时可以设置某些经验值以提高优化效率．

- ✓ LSTM 的遗忘门，偏置可以设置为 1或者2，使得时序上的梯度变大；
- ✓ 使用ReLU的神经元，偏置可以设置为 0.01，使得 ReLU 神经元在训练初期更容易激活，从而获得一定的梯度来进行误差反向传播．

随机初始化

- Gaussian分布初始化
  - Gaussian初始化方法是最简单的初始化方法，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。

- 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。

参数初始化

- 基于**方差缩放**的参数初始化
  - Xavier 初始化、He 初始化

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	$r = \sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2 = \frac{2}{M_{l-1}}$

基于方差的初始化方法都是对权重矩阵中的每个参数进行独立采样．由于采样的随机性，采样出来的权重矩阵依然可能存在梯度消失或梯度爆炸问题．

随机初始化

范数保持性（Norm-Preserving）

- 一个 M 层的等宽线性网络

$$\boldsymbol{y} = \boldsymbol{W}^{(L)}\boldsymbol{W}^{(L-1)}\cdots\boldsymbol{W}^{(1)}\boldsymbol{x}$$

- 为了避免梯度消失或梯度爆炸问题，我们希望误差项

$$\|\delta^{(l-1)}\|^2 = \|\delta^{(l)}\|^2 = \|(\boldsymbol{W}^{(l)})^{\mathrm{T}}\delta^{(l)}\|^2$$

正交初始化 $\boldsymbol{W}^{(l)}(\boldsymbol{W}^{(l)})^T = I$

1）用均值为 0、方差为 1 的高斯分布初始化一个矩阵；
2）将这个矩阵用**奇异值分解**得到两个正交矩阵，并使用其中之一作为权重矩阵。

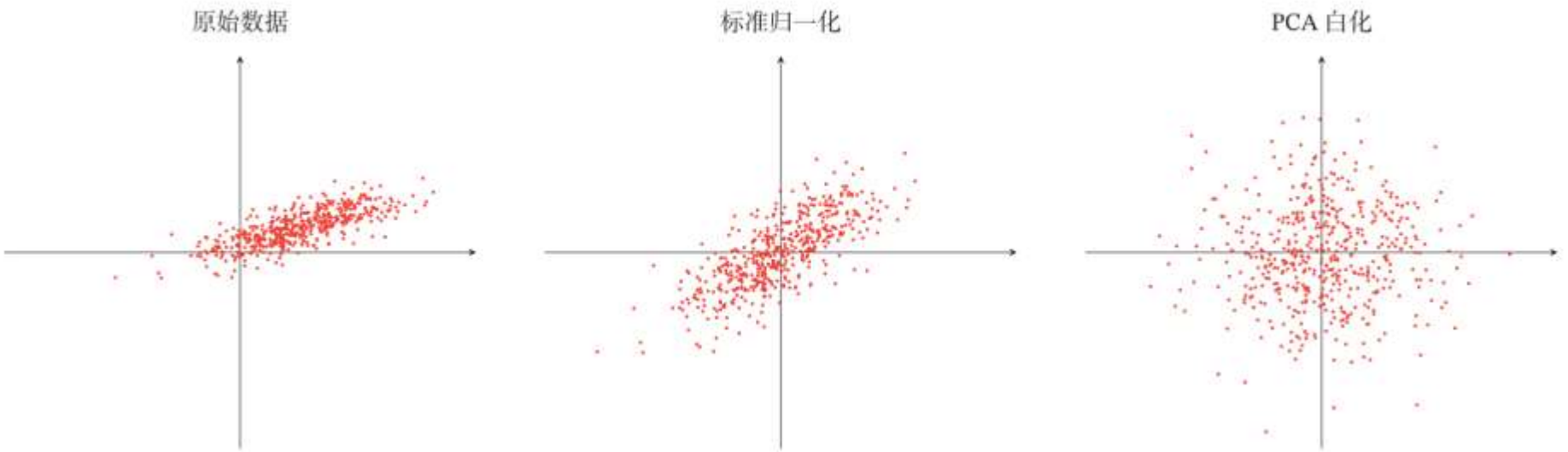
数据预处理

- 数据归一化
  - 最小最大值归一化

    通过缩放将每一个特征的取值范围归一到[0, 1]或[−1, 1] 之间
  - 标准化

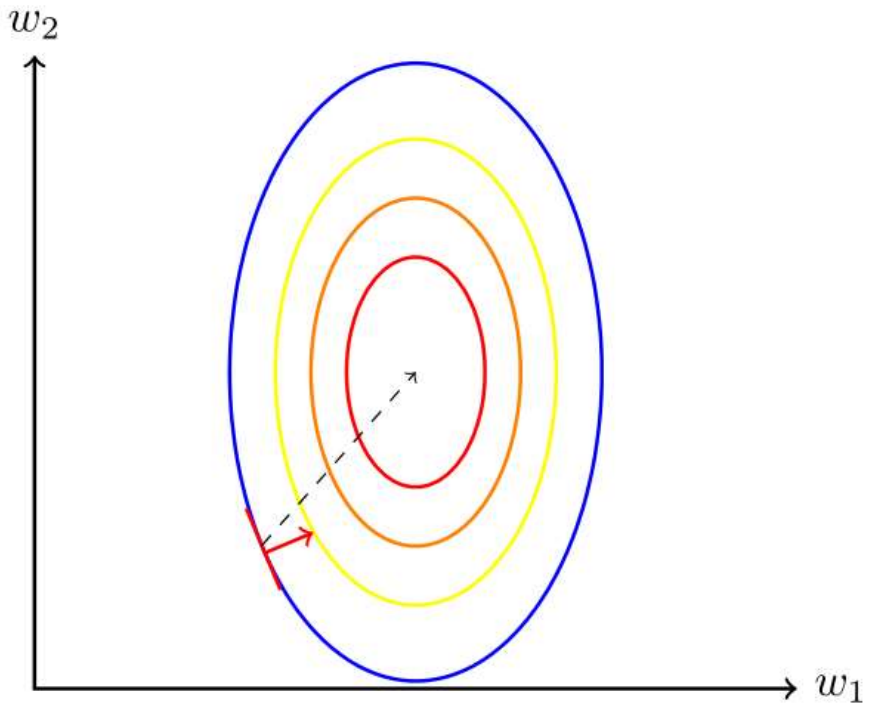
$$\mu = \frac{1}{N}\sum_{n=1}^{N} x^{(n)}, \quad \sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x^{(n)} - \mu)^2. \qquad \hat{x}^{(n)} = \frac{x^{(n)} - \mu}{\sigma},$$

  - 白化（Whitening）　通过 PCA 降低输入数据特征之间的冗余性。

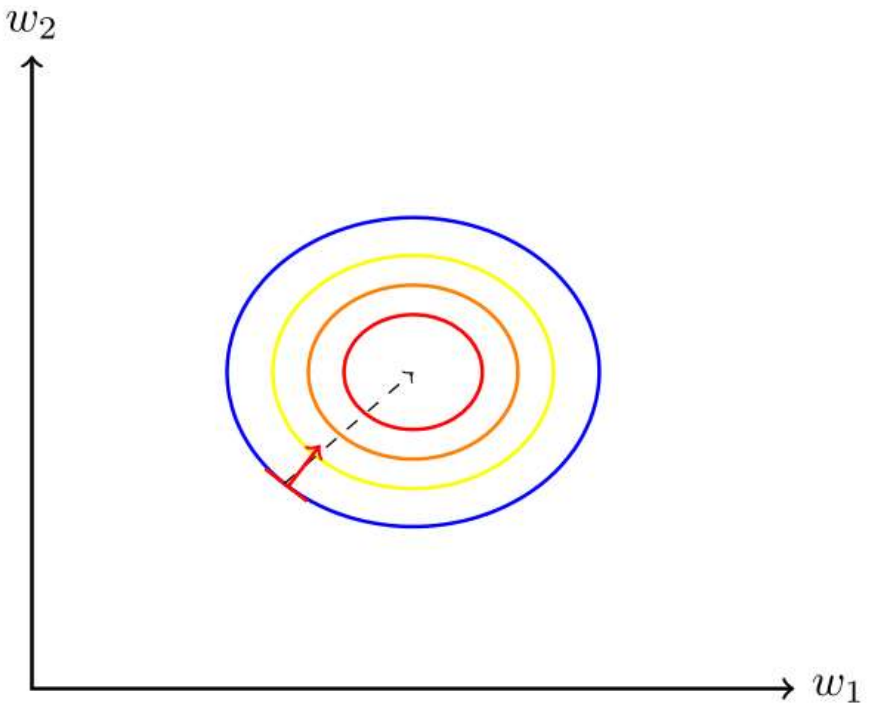


数据预处理

- 数据归一化对梯度的影响



(a) 未归一化数据的梯度



(b) 归一化数据的梯度

超参数优化

超参数优化

- 超参数
  - 层数
  - 每层神经元个数
  - 激活函数
  - 学习率（以及动态调整算法）
  - 正则化系数
  - mini-batch 大小

- 优化方法
  - 网格搜索
  - 随机搜索
  - 贝叶斯优化
  - 动态资源分配
  - 神经架构搜索

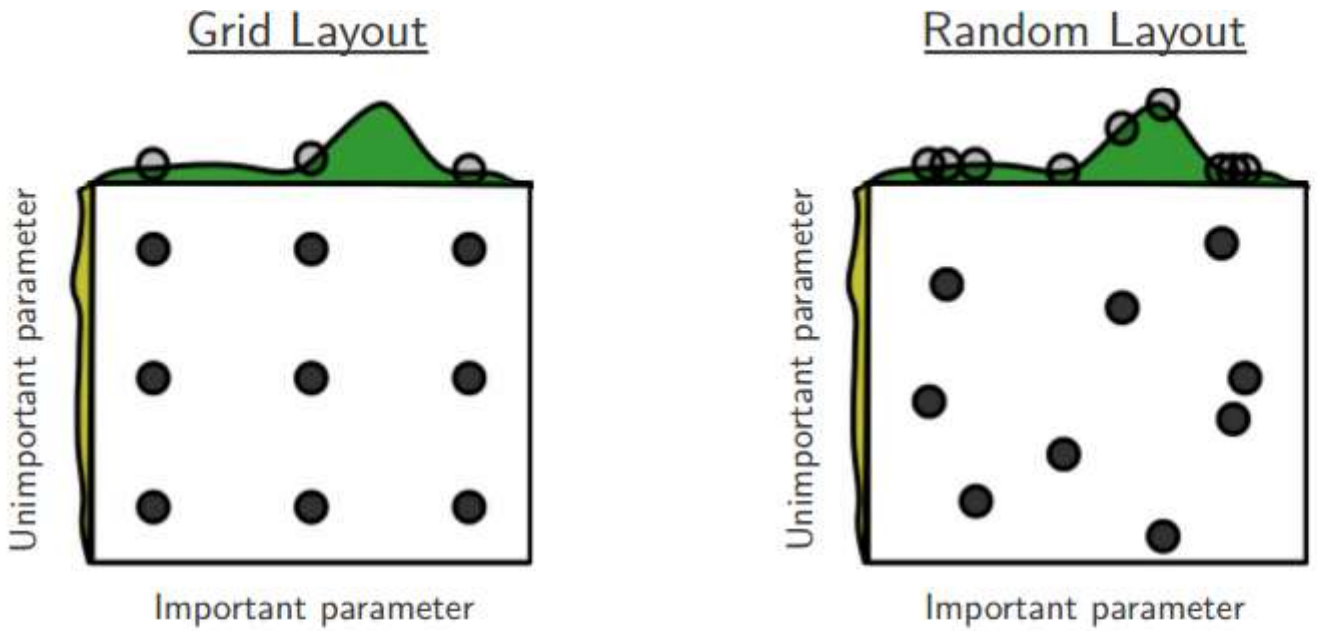
超参数优化

- 网格搜索（Grid Search）
  - 假设总共有K 个超参数，第k个超参数的可以取m_k 个值。
  - 如果参数是连续的，可以将参数离散化，选择几个“经验”值。比如学习率α，我们可以设置

$$\alpha \in \{0.01, 0.1, 0.5, 1.0\}$$

  - 这些超参数可以有 $m_1 \times m_2 \times \cdots \times m_K$ 个取值组合。

超参数优化



Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

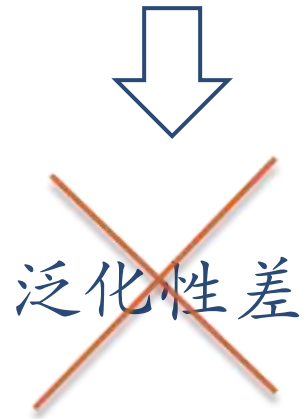
超参数优化

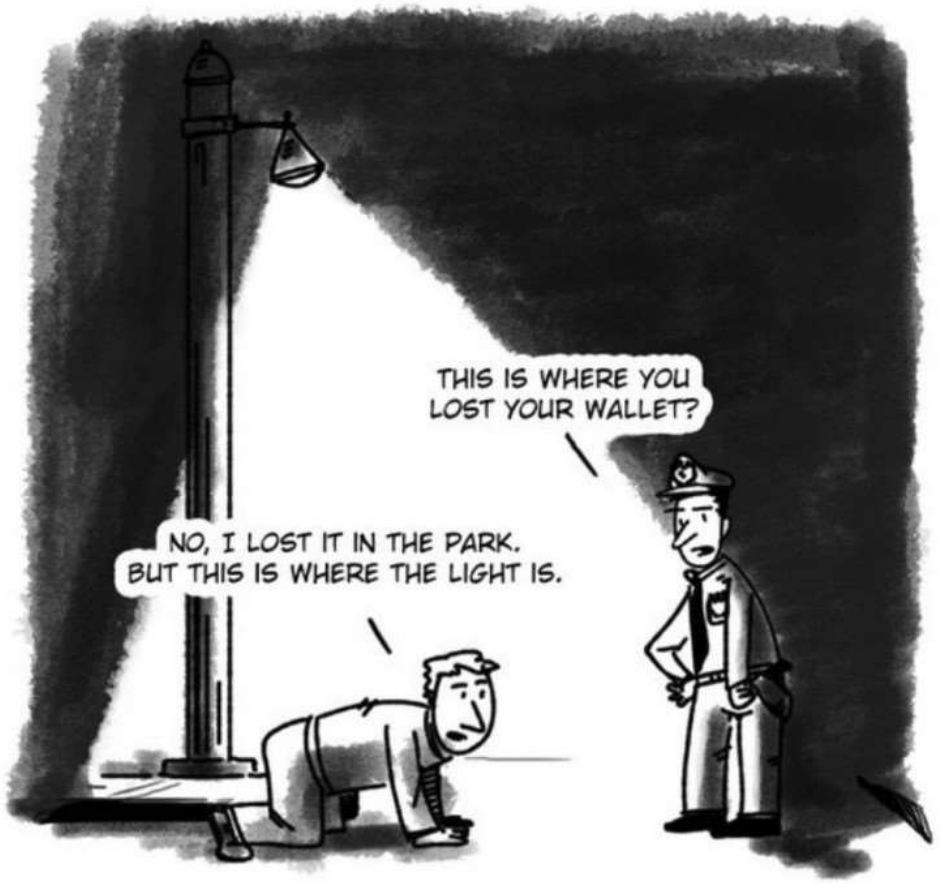
- 贝叶斯优化
- 动态资源分配
- 神经架构搜索

网络正则化

重新思考泛化性

- 神经网络
  - 过度参数化
  - 拟合能力强



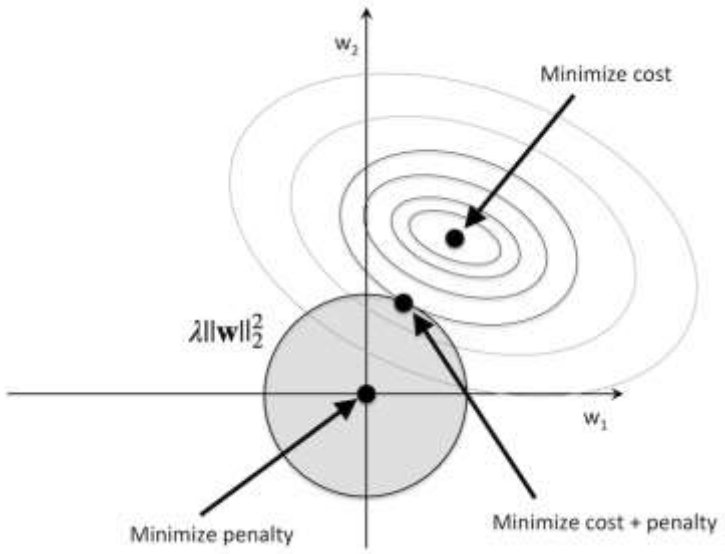


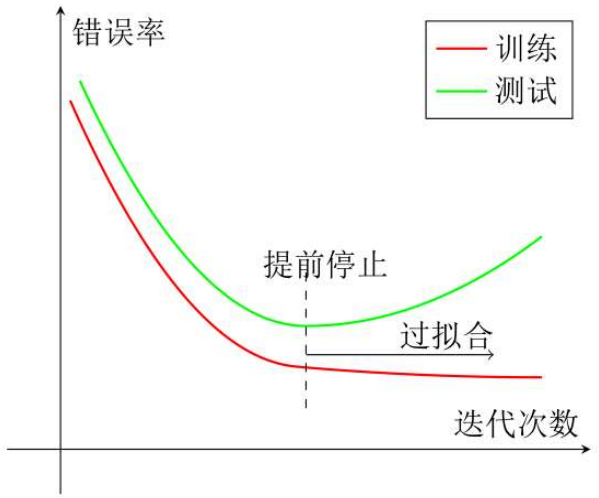
Zhang C, Bengio S, Hardt M, et al. Understanding deep learning requires rethinking generalization[J]. arXiv preprint arXiv:1611.03530, 2016.

正则化（regularization）

所有损害优化的方法都是正则化。

- 增加优化约束：L1/L2约束、数据增强
- 干扰优化过程：权重衰减、随机梯度下降、提前停止





正则化

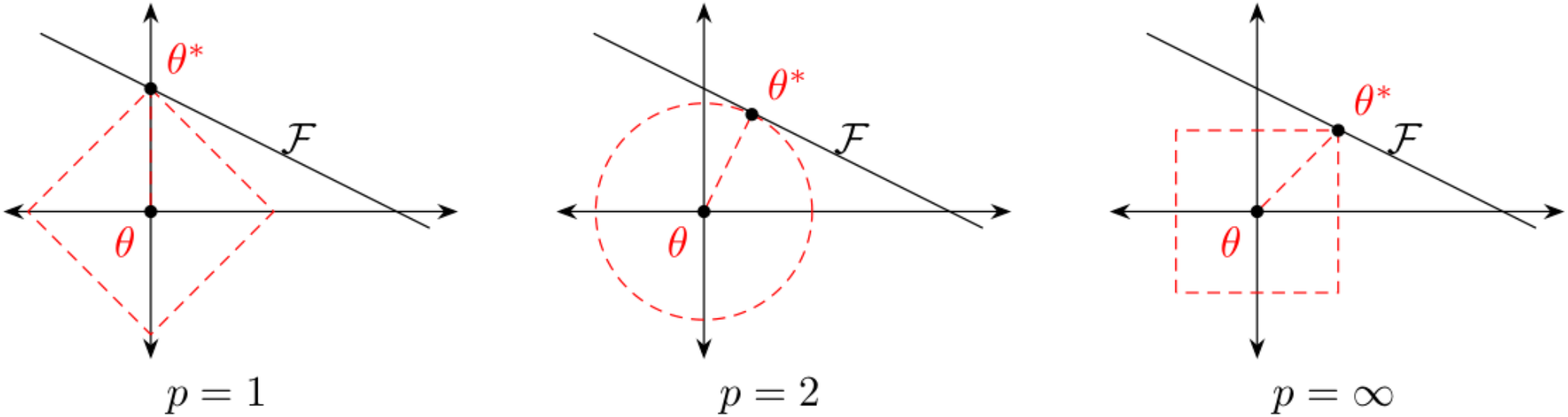
- 如何提高神经网络的泛化能力
  - ℓ_1和ℓ_2正则化
  - early stop
  - 权重衰减
  - SGD
  - Dropout
  - 数据增强

ℓ_1和ℓ_2正则化

- 优化问题可以写为

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}\left(y^{(n)}, f(\mathbf{x}^{(n)}, \theta)\right) + \lambda \ell_p(\theta)$$

- ℓ_p 为范数函数，p的取值通常为{1,2}代表ℓ_1和ℓ_2范数，λ为正则化系数。

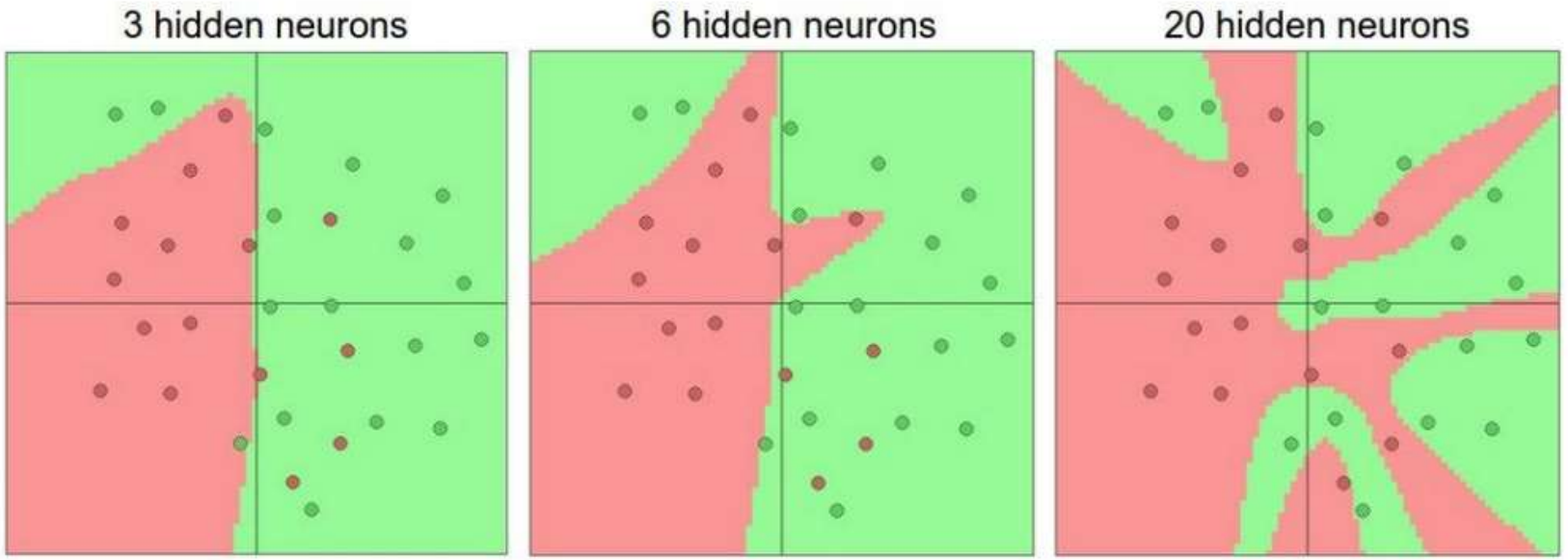


$p=1$　　$p=2$　　$p=\infty$

神经网络示例

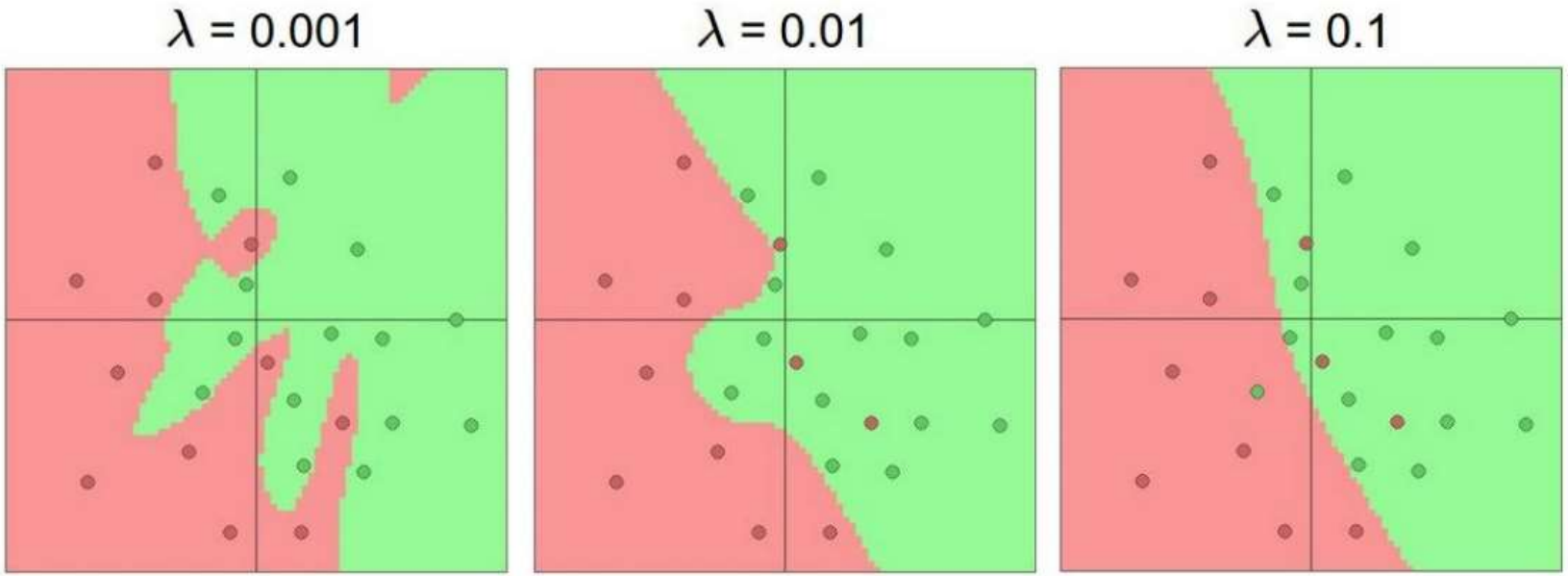
http://playground.tensorflow.org/

- 隐藏层的不同神经元个数



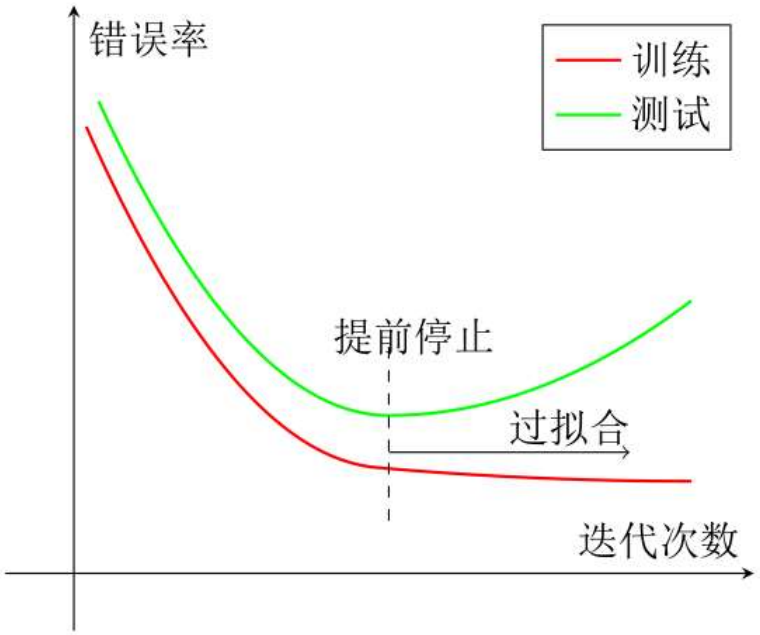
神经网络示例

- 不同的正则化系数



提前停止

- 我们使用一个验证集（Validation Dataset）来测试每一次迭代的参数在验证集上是否最优。如果在验证集上的错误率不再下降，就停止迭代。



权重衰减（Weight Decay）

- 在每次参数更新时，引入一个衰减系数w。

$$\theta_t \leftarrow (1-w)\theta_{t-1} - \alpha \mathbf{g}_t$$

- 在标准的随机梯度下降中，权重衰减正则化和 ℓ_2 正则化的效果相同。

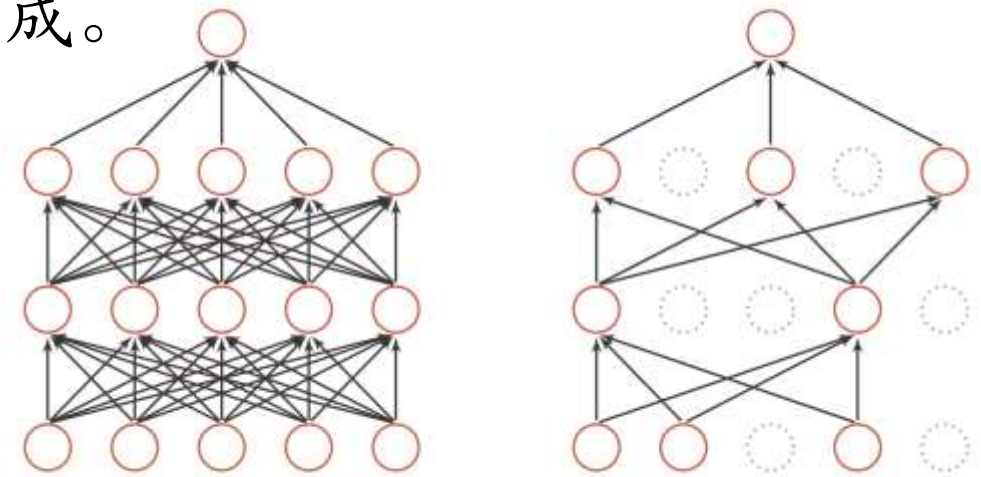
- 在较为复杂的优化方法（比如Adam）中，权重衰减和ℓ_2正则化并不等价。

丢弃法（Dropout Method）

- 对于一个神经层$y = f(Wx + b)$，引入一个丢弃函数$d(\cdot)$使得$y = f(Wd(x) + b)$。

$$d(\mathbf{x}) = \begin{cases} \mathbf{m} \odot \mathbf{x} & \text{当训练阶段时} \\ p\mathbf{x} & \text{当测试阶段时} \end{cases}$$

- 其中$m \in \{0,1\}^d$是丢弃掩码（dropout mask），通过以概率为p的贝努力分布随机生成。



(a) 标准网络　　(b) Dropout 后的网络

Dropout意义

- 集成学习的解释
  - 每做一次丢弃，相当于从原始的网络中采样得到一个子网络。如果一个神经网络有n个神经元，那么总共可以采样出2n个子网络。

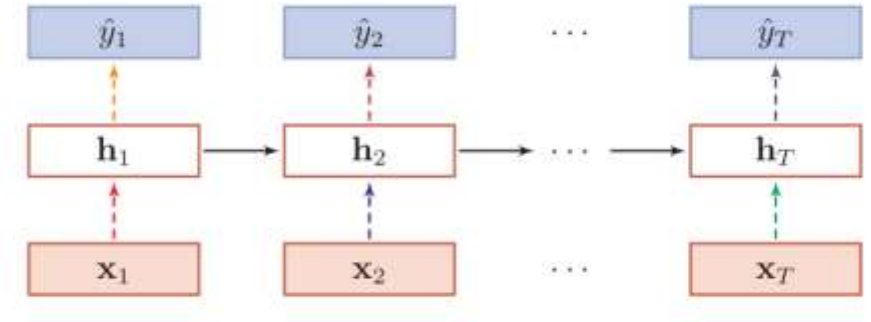
- 贝叶斯学习的解释

$$\mathbb{E}_{q(\theta)}[y] = \int_q f(\mathbf{x}, \theta) q(\theta) d\theta$$
$$\approx \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}, \theta_m),$$

  - 其中$f(x,\theta_m)$为第m次应用丢弃方法后的网络。

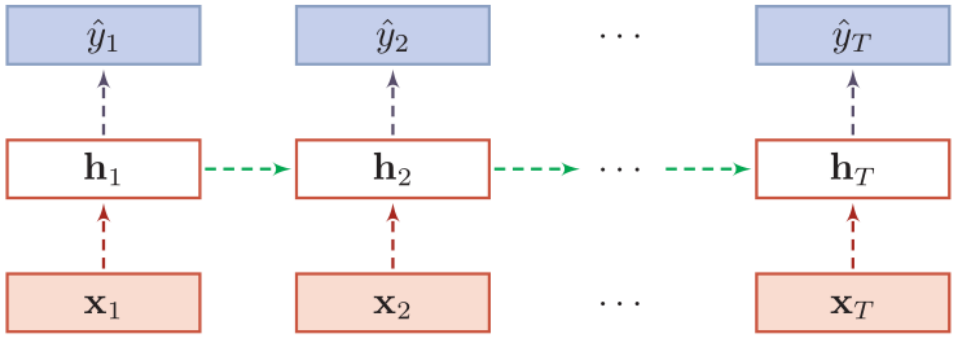
循环神经网络上的丢弃法

- 当在循环神经网络上应用丢弃法，不能直接对每个时刻的隐状态进行随机丢弃，这样会损害循环网络在时间维度上记忆能力。



虚线边表示进行随机丢弃，不同的颜色表示不同的丢弃掩码。

- 变分Dropout
  - 根据贝叶斯学习的解释，丢弃法是一种对参数θ的采样。
    - 每次采样的参数需要在每个时刻保持不变。因此，在对循环神经网络上使用丢弃法时，需要对参数矩阵的每个元素进行随机丢弃，并在所有时刻都使用相同的丢弃掩码。



相同颜色表示使用相同的丢弃掩码

数据增强（Data Augmentation）

- 图像数据的增强主要是通过算法对图像进行转变，引入噪声等方法来增加数据的多样性。
  - 图像数据的增强方法：
    - 旋转（Rotation）：将图像按顺时针或逆时针方向随机旋转一定角度；
    - 翻转（Flip）：将图像沿水平或垂直方法随机翻转一定角度；
    - 缩放（Zoom In/Out）：将图像放大或缩小一定比例；
    - 平移（Shift）：将图像沿水平或垂直方法平移一定步长；
    - 加噪声（Noise）：加入随机噪声。

标签平滑（Label Smoothing）

- 在输出标签中添加噪声来避免模型过拟合。
- 一个样本x的标签一般用onehot向量表示

$$\mathbf{y} = [0, \cdots, 0, 1, 0, \cdots, 0]^{\mathrm{T}}$$

硬目标（Hard Targets）

- 引入一个噪声对标签进行平滑，即假设样本以ϵ的概率为其它类。平滑后的标签为

$$\tilde{\mathbf{y}} = [\frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}, 1-\epsilon, \frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}]^{\mathrm{T}}$$

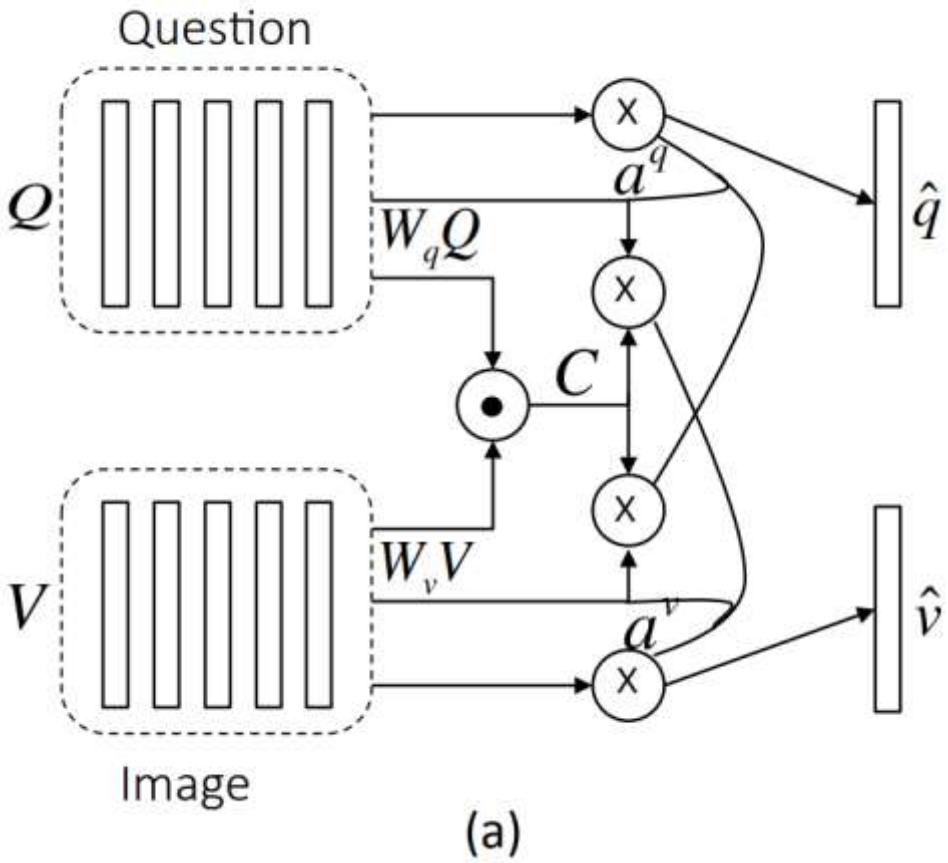
总结

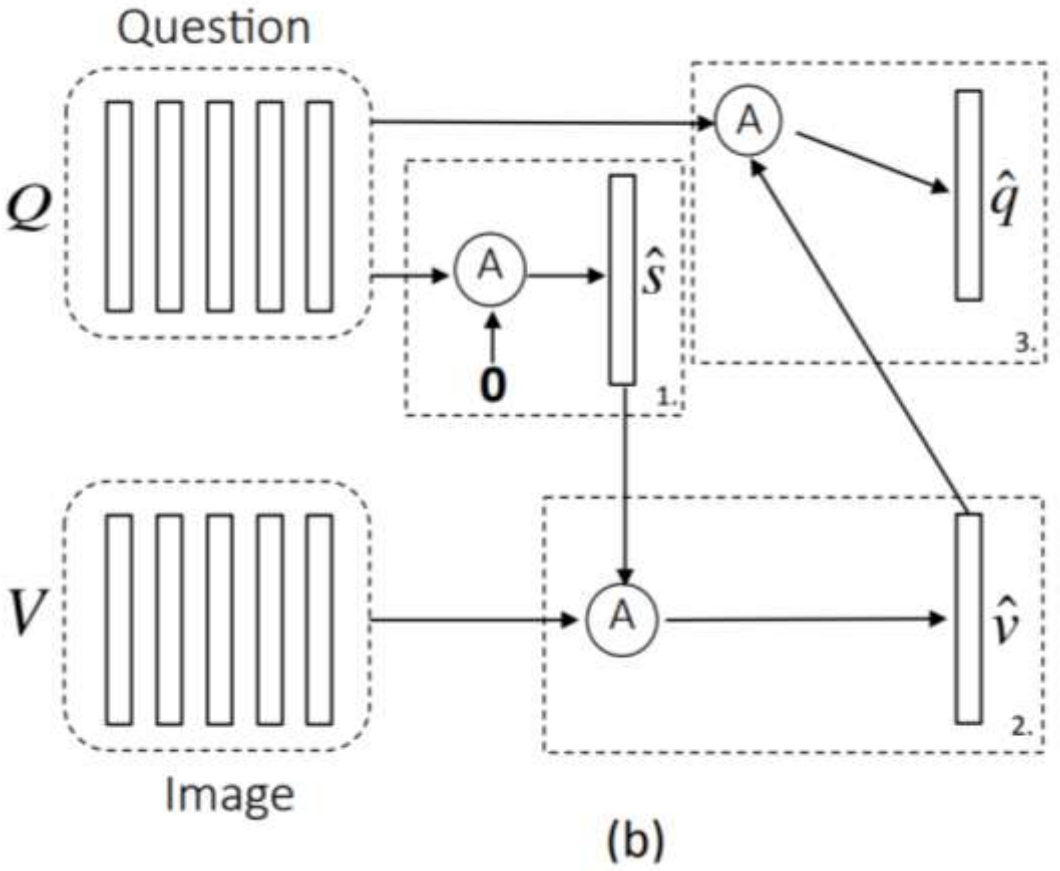
- 模型
  - 用 ReLU 作为激活函数
  - 分类时用交叉熵作为损失函数
  - 逐层归一化
- 优化
  - SDG+mini-batch（动态学习率、Adam算法优先）
  - 每次迭代都重新随机排序
  - 数据预处理（标准归一化）
  - 参数初始化
- 正则化
  - ℓ_1和ℓ_2正则化（跳过前几轮）
  - Dropout
  - Early-stop
  - 数据增强

谢谢！

随堂小测试

请写出 co-attention 的Python代码实现：





https://github.com/jiasenlu/HieCoAttenVQA/blob/master/misc/attention.lua
https://github.com/karunraju/VQA/blob/master/coatt/coattention_net.py