

## 《神经网络与深度学习》



### 网络优化与正则化

<https://nndl.github.io/>

# 机器学习的矛与盾

---

优化

经验风险最小

正则化

降低模型复杂度



# 内容

---

- ▶ 网络优化
- ▶ 优化算法
- ▶ 参数初始化
- ▶ 数据预处理
- ▶ 逐层归一化
- ▶ 超参数优化
- ▶ 网络正则化



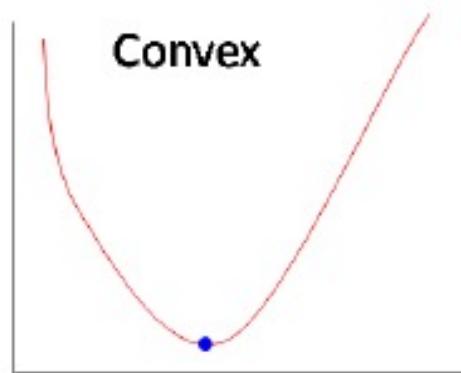
# 网络优化的难点

---

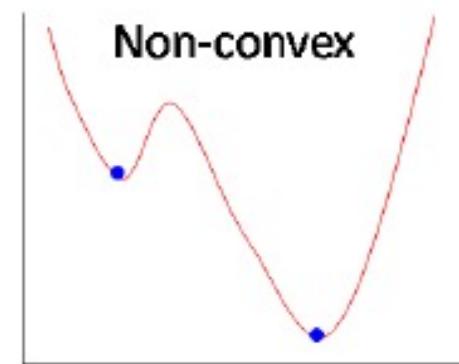
- ▶ 结构差异大
  - ▶ 没有通用的优化算法
  - ▶ 超参数多
- 
- ▶ 非凸优化问题
  - ▶ 参数初始化
  - ▶ 逃离局部最优
- 
- ▶ 梯度消失（爆炸）问题

# 最优化问题

► 机器学习问题转化成为一个最优化问题



Convex

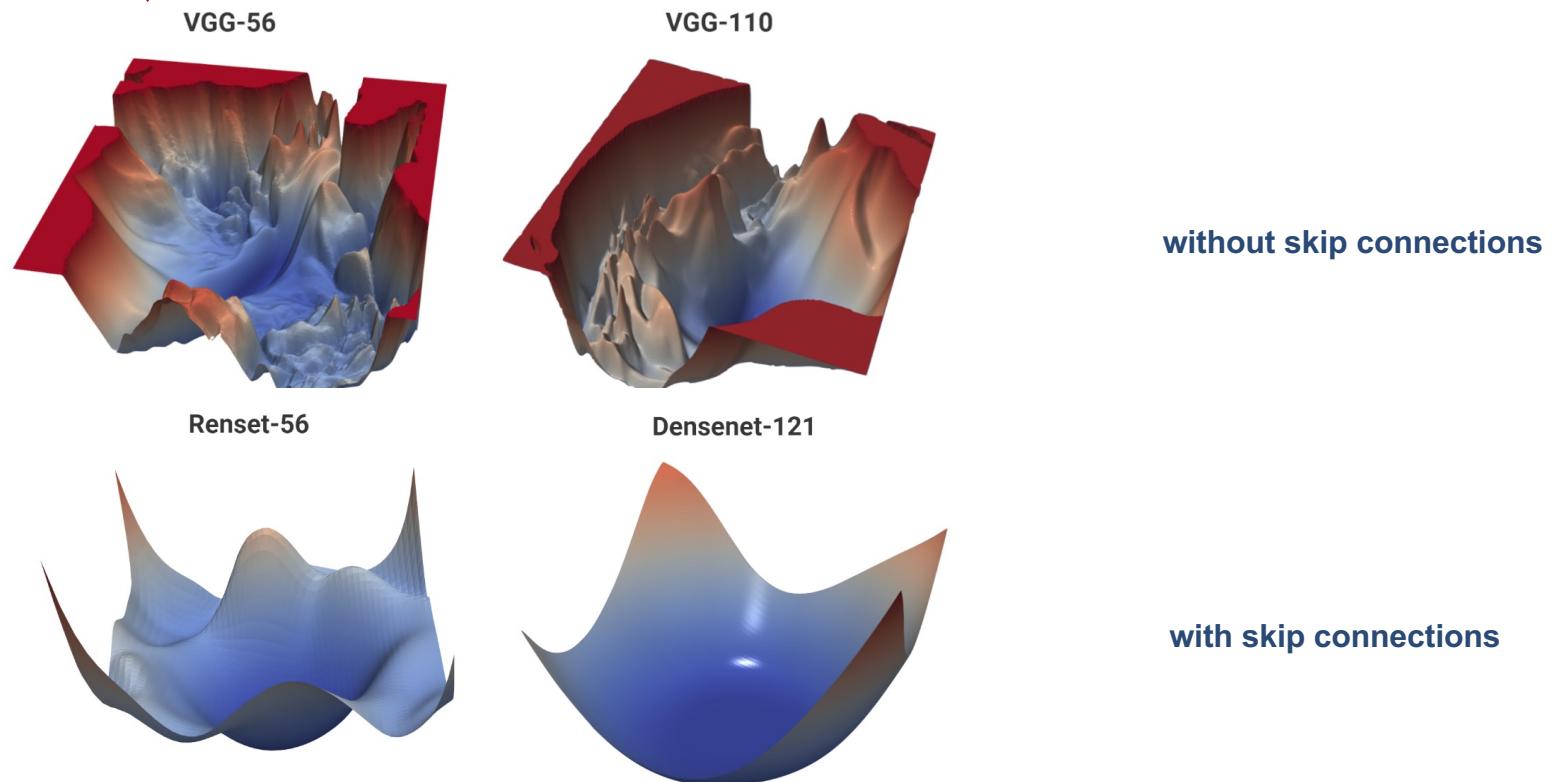


Non-convex

$$\min_{\mathbf{x}} f(\mathbf{x})$$

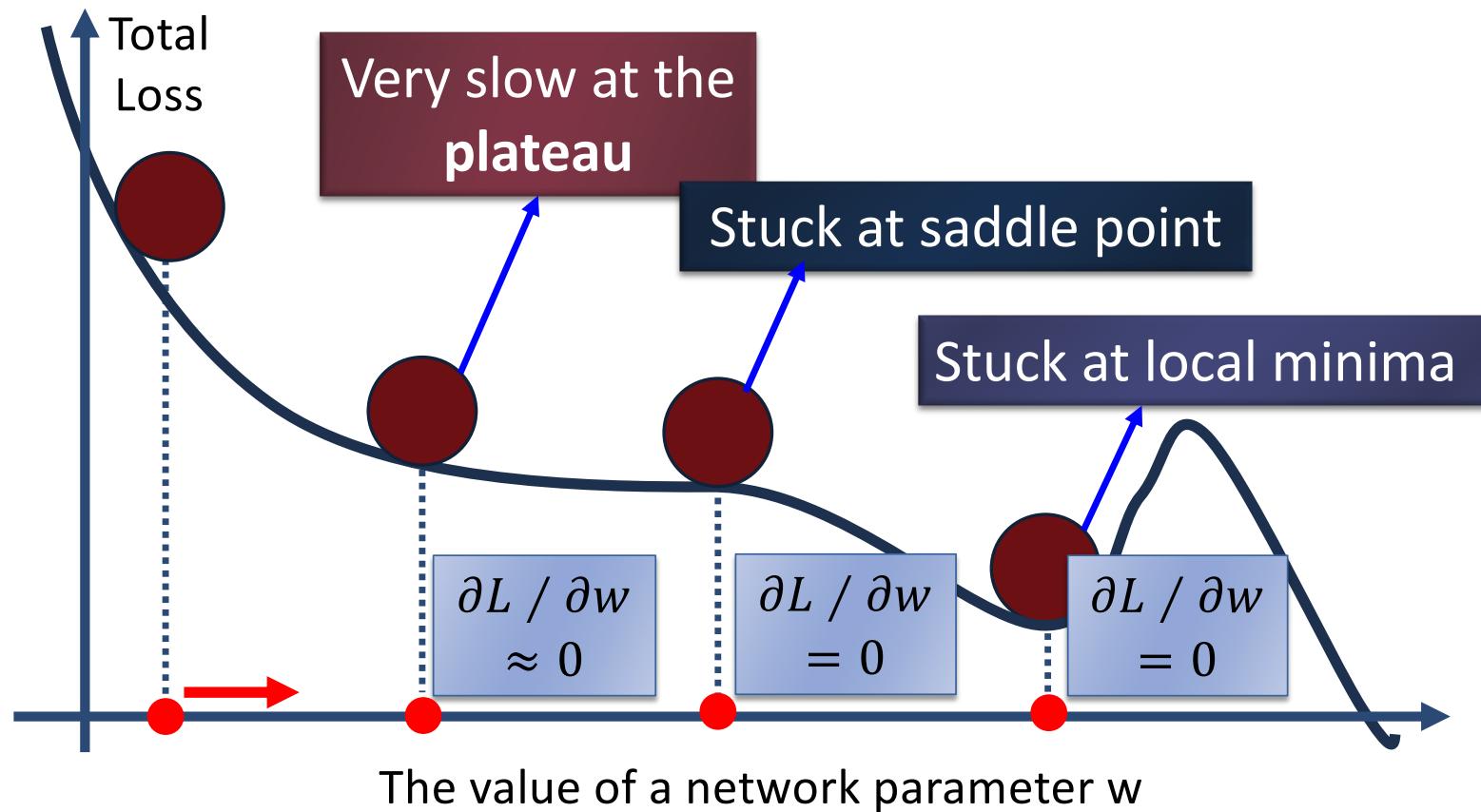
# VISUALIZING THE LOSS LANDSCAPE OF NN

## ►深度学习的最优化问题



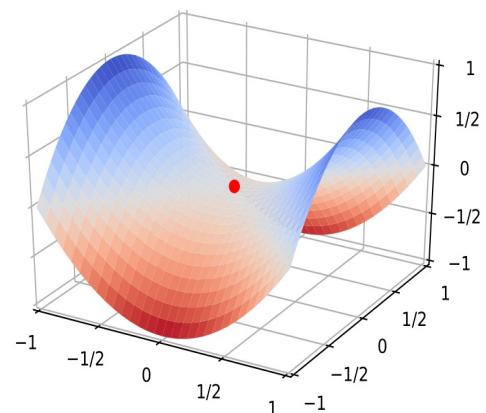
Li H, Xu Z, Taylor G, et al. Visualizing the loss landscape of neural nets[C]//Advances in Neural Information Processing Systems. 2018: 6389-6399.

# 高维空间的非凸优化问题

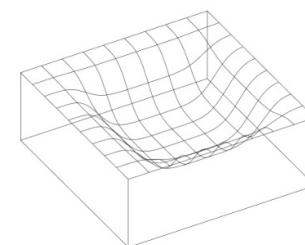


# 高维空间的非凸优化问题

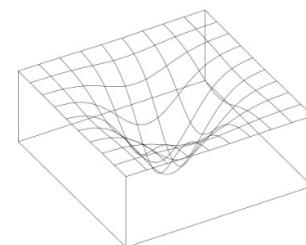
- ▶ 鞍点 ( Saddle Point ) : 梯度为 0(驻点: Stationary Point), 一些维度上是最高点, 另一些维度上是最低点



- ▶ 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
  - ▶ 在训练神经网络时, 我们找到局部最小解即可



(a) 平坦最小值



(b) 尖锐最小值

# 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

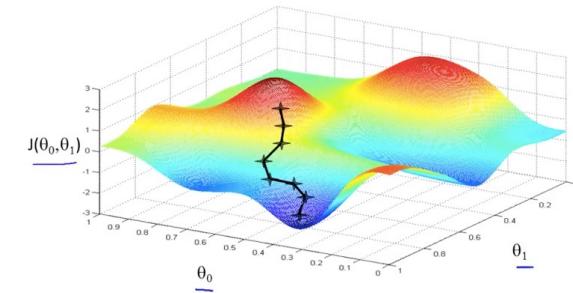


# 优化算法：随机梯度下降

## 算法 2.1 随机梯度下降法

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

```
1 随机初始化  $\theta$ ;  
2 repeat  
3   对训练集  $\mathcal{D}$  中的样本随机排序;  
4   for  $n = 1 \dots N$  do  
5     从训练集  $\mathcal{D}$  中选取样本  $(x^{(n)}, y^{(n)})$ ;  
6      $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}(\theta; x^{(n)}, y^{(n)})}{\partial \theta}$ ;           // 更新参数  
7   end  
8 until 模型  $f(x; \theta)$  在验证集  $\mathcal{V}$  上的错误率不再下降;  
输出:  $\theta$ 
```



# 优化算法：小批量随机梯度下降 MiniBatch

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

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

- 定义梯度

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

- 更新参数

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

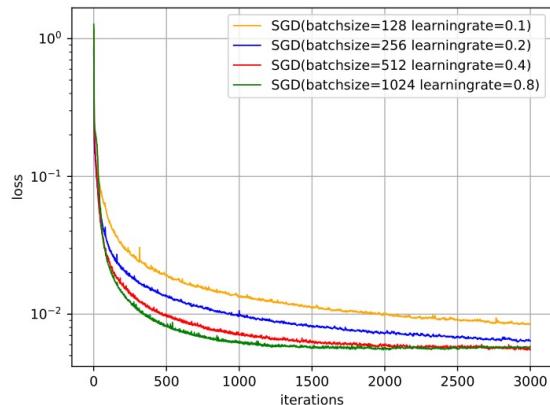
几个关键因素：

- 小批量样本数量
- 梯度
- 学习率

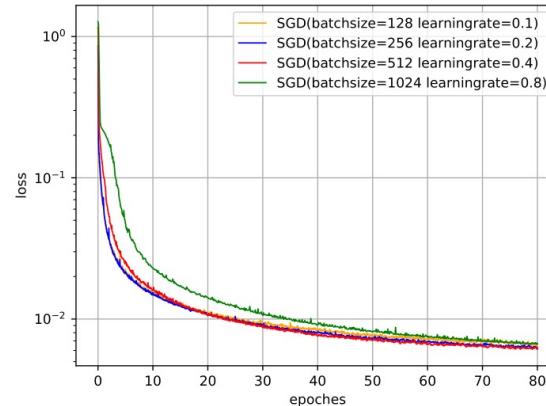
- 其中  $\alpha > 0$  为学习率

# 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。
- ▶ 由于硬件特性，批量大小一般是2的次方
- ▶ 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。
- ▶ 而批量较小时，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

4 种批量大小对应的学习率设置不同，因此并不是严格对比。

小批量梯度下降中，每次选取样本数量对损失下降的影响。

# 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

► 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

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

$$\Delta\theta_t = -\alpha \mathbf{g}_t$$

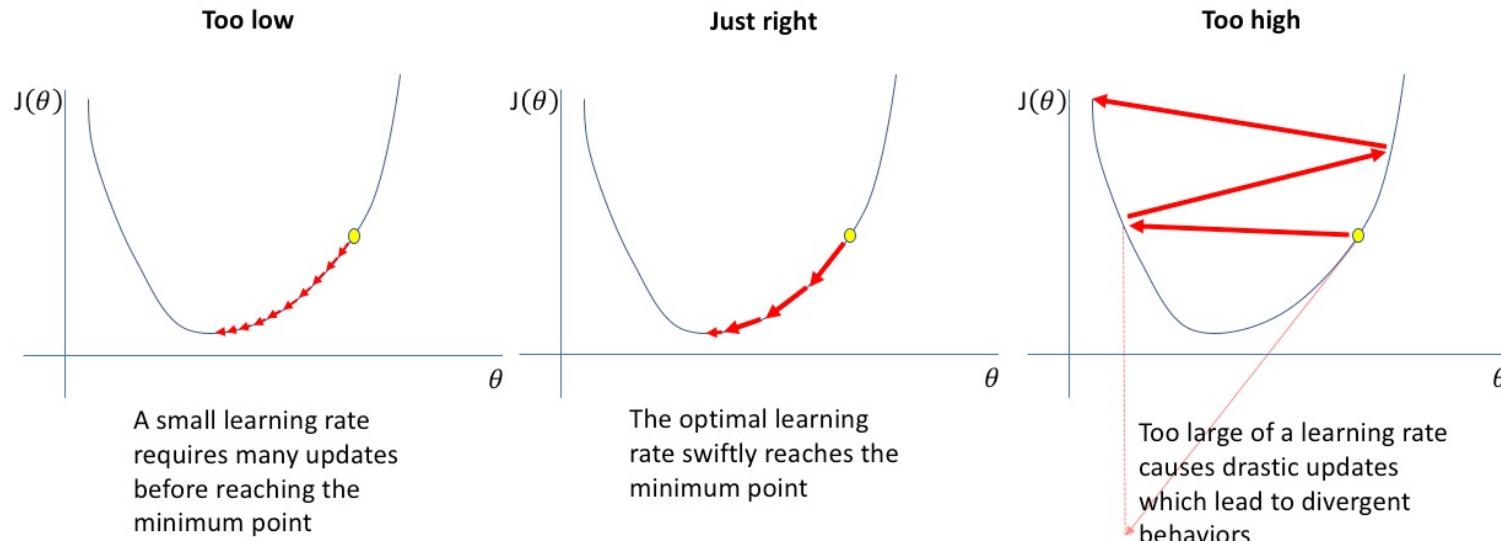
实际更新方向

梯度方向

Adam

Adam is better choice!

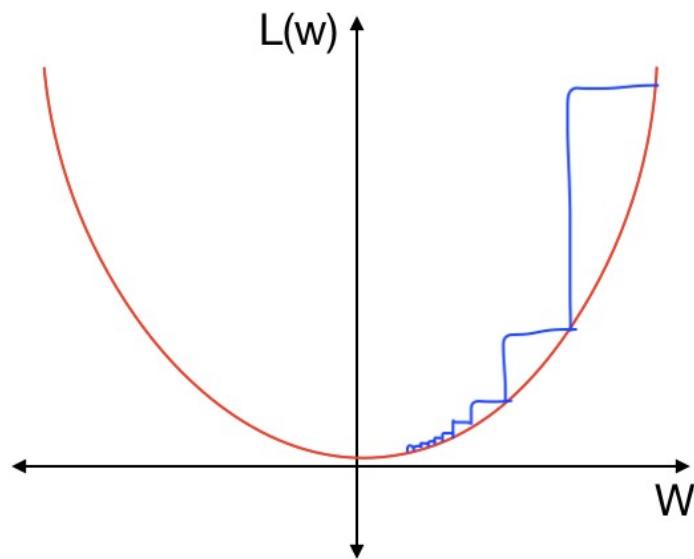
# 学习率的影响



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

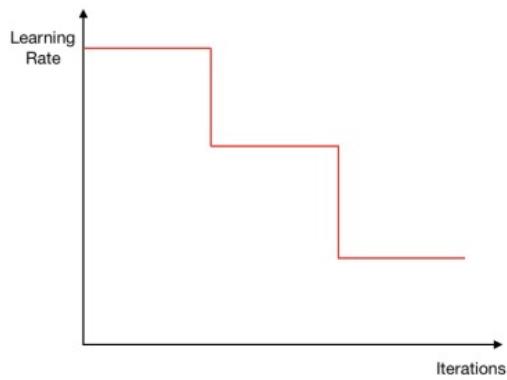
# 学习率衰减

$$\Delta\theta_t = -\underline{\alpha} g_t$$

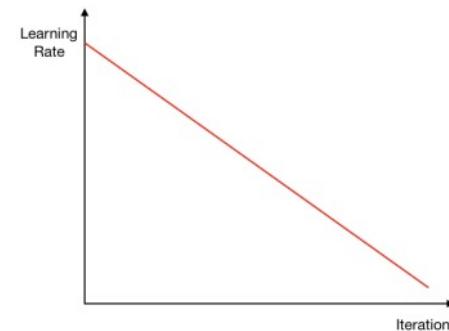


# 学习率衰减

$$\alpha = f(t)$$



梯级衰减 (step decay)



线性衰减 (Linear Decay)

# 学习率衰减

分段常数衰减 ( Piecewise Constant Decay ) : 即每经过  $T_1, T_2, \dots, T_m$  次迭代将学习率衰减为原来的  $\beta_1, \beta_2, \dots, \beta_m$  倍, 其中  $T_m$  和  $\beta_m < 1$  为根据经验设置的超参数. 分段常数衰减也称为阶梯衰减 ( Step Decay ).

逆时衰减 ( Inverse Time Decay ) :

$$\alpha_t = \alpha_0 \frac{1}{1 + \beta \times t},$$

其中  $\beta$  为衰减率.

指数衰减 ( Exponential Decay ) :

$$\alpha_t = \alpha_0 \beta^t,$$

其中  $\beta < 1$  为衰减率.

自然指数衰减 ( Natural Exponential Decay ) :

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

其中  $\beta$  为衰减率.

余弦衰减 ( Cosine Decay ) :

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

其中  $T$  为总的迭代次数.

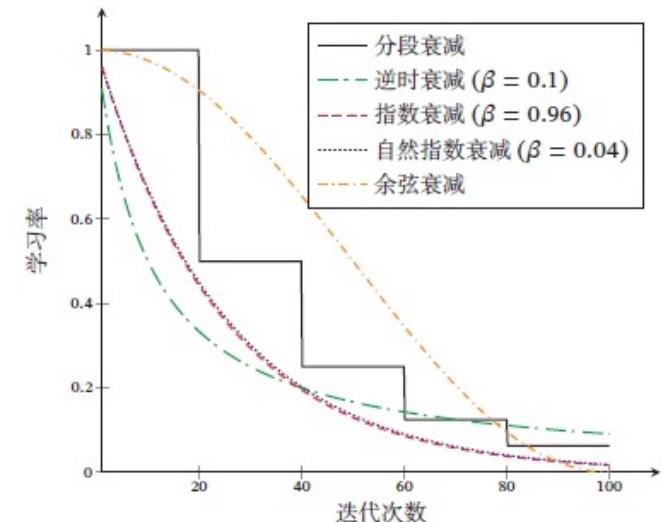
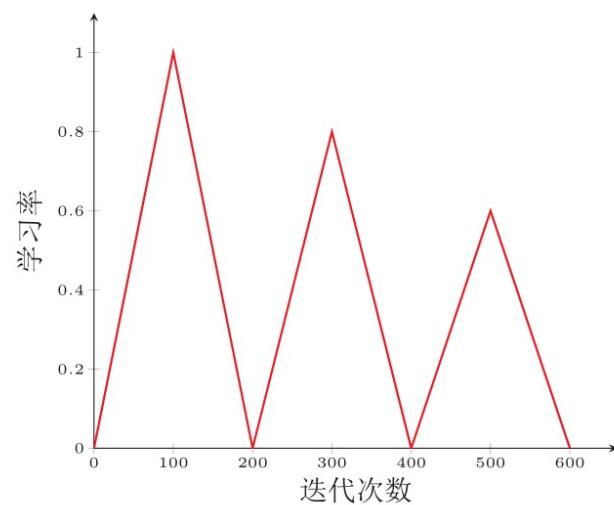
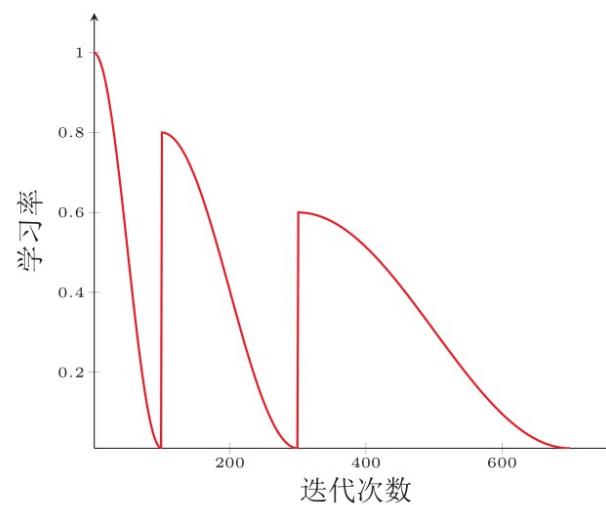


图 7.4 不同学习率衰减方法的比较

# 周期性学习率调整 Cyclical Learning Rates

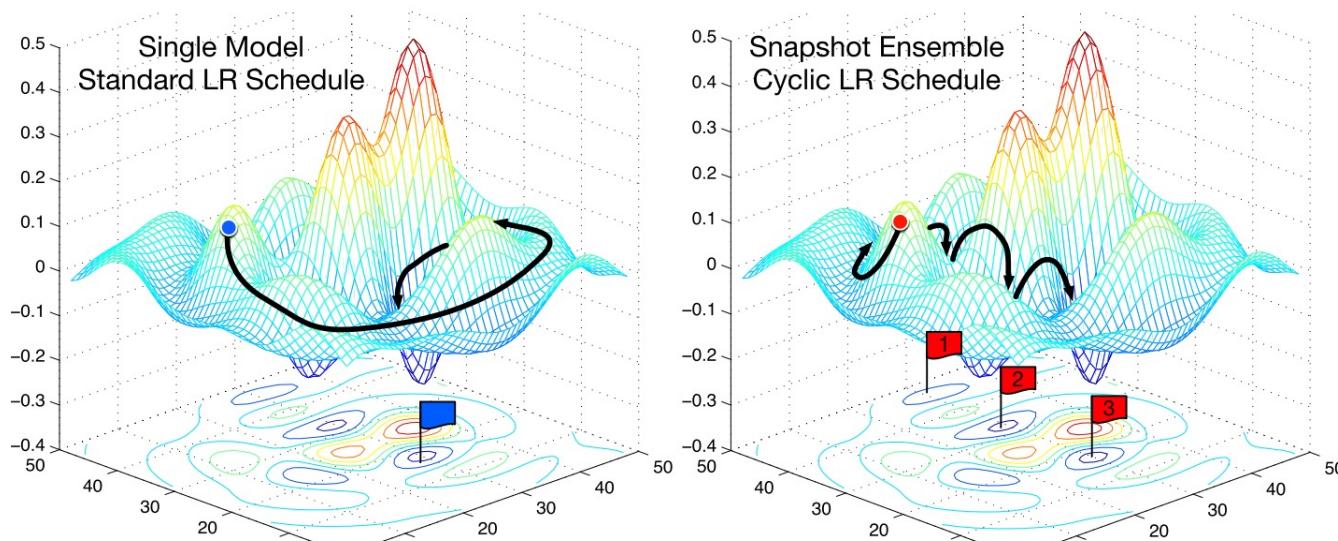


(a) 三角循环学习率



(b) 带热重启的余弦衰减

# Cyclical Learning Rates

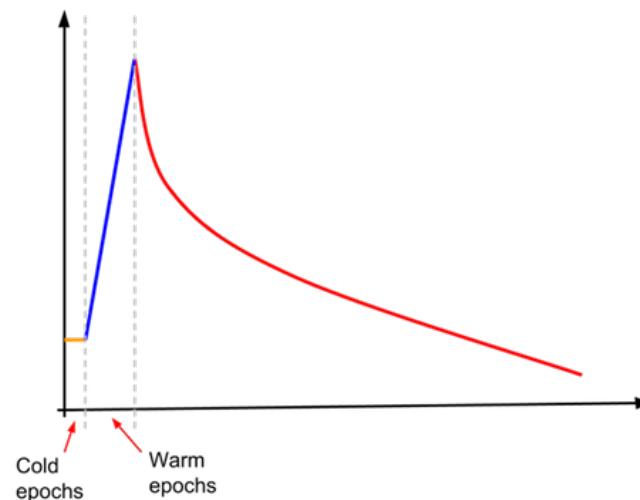


## Others

---

- ▶ 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 (SGDR)
    - ▶ <https://arxiv.org/abs/1706.02677>

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



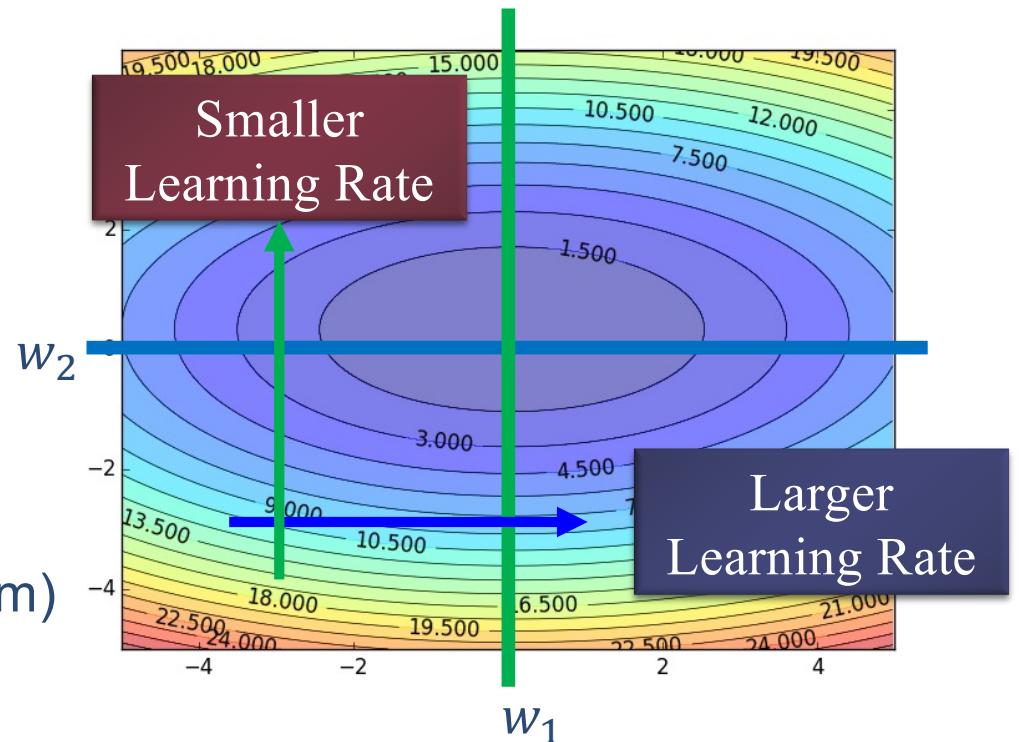
# 自适应学习率

$$\begin{aligned}\alpha &= f(t) \\ \downarrow \\ \alpha &= f(g_1, \dots, g_t)\end{aligned}$$

Adagrad 算法(Adaptive Gradient Algorithm)

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

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



# 自适应学习率

---

RMSprop 算法

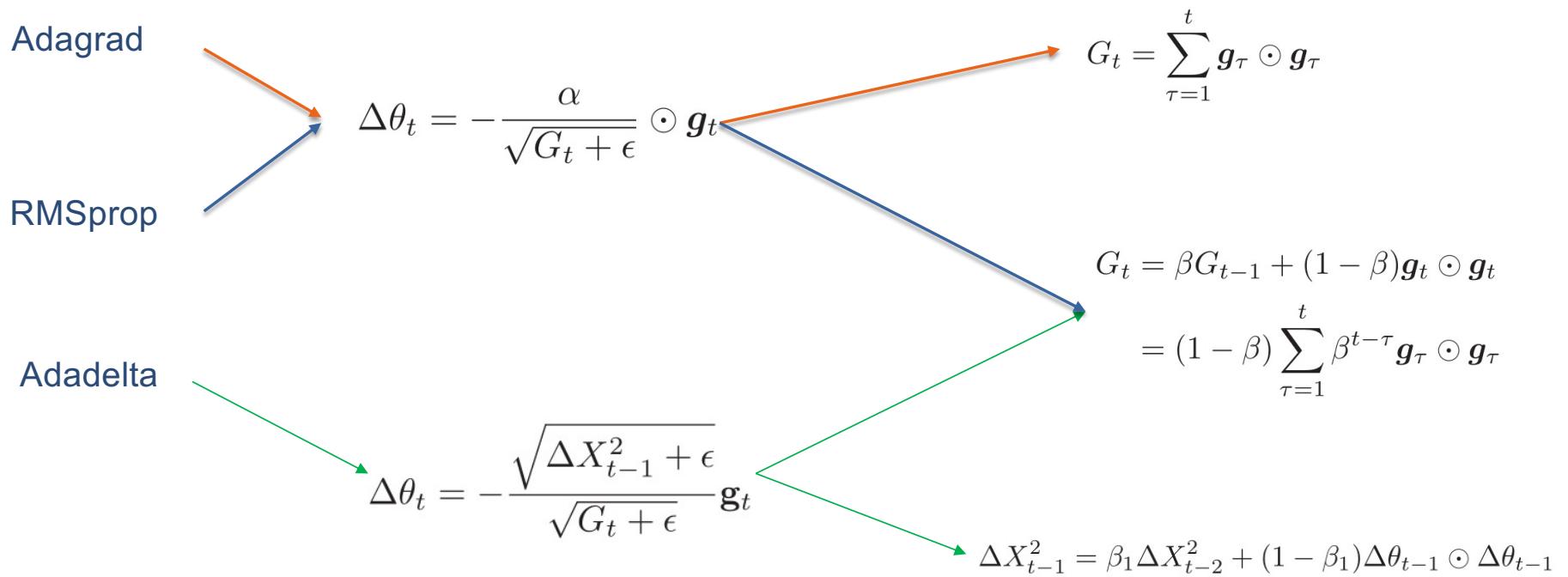
$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot g_t$$

$$G_t = \beta G_{t-1} + (1 - \beta) g_t \odot g_t$$

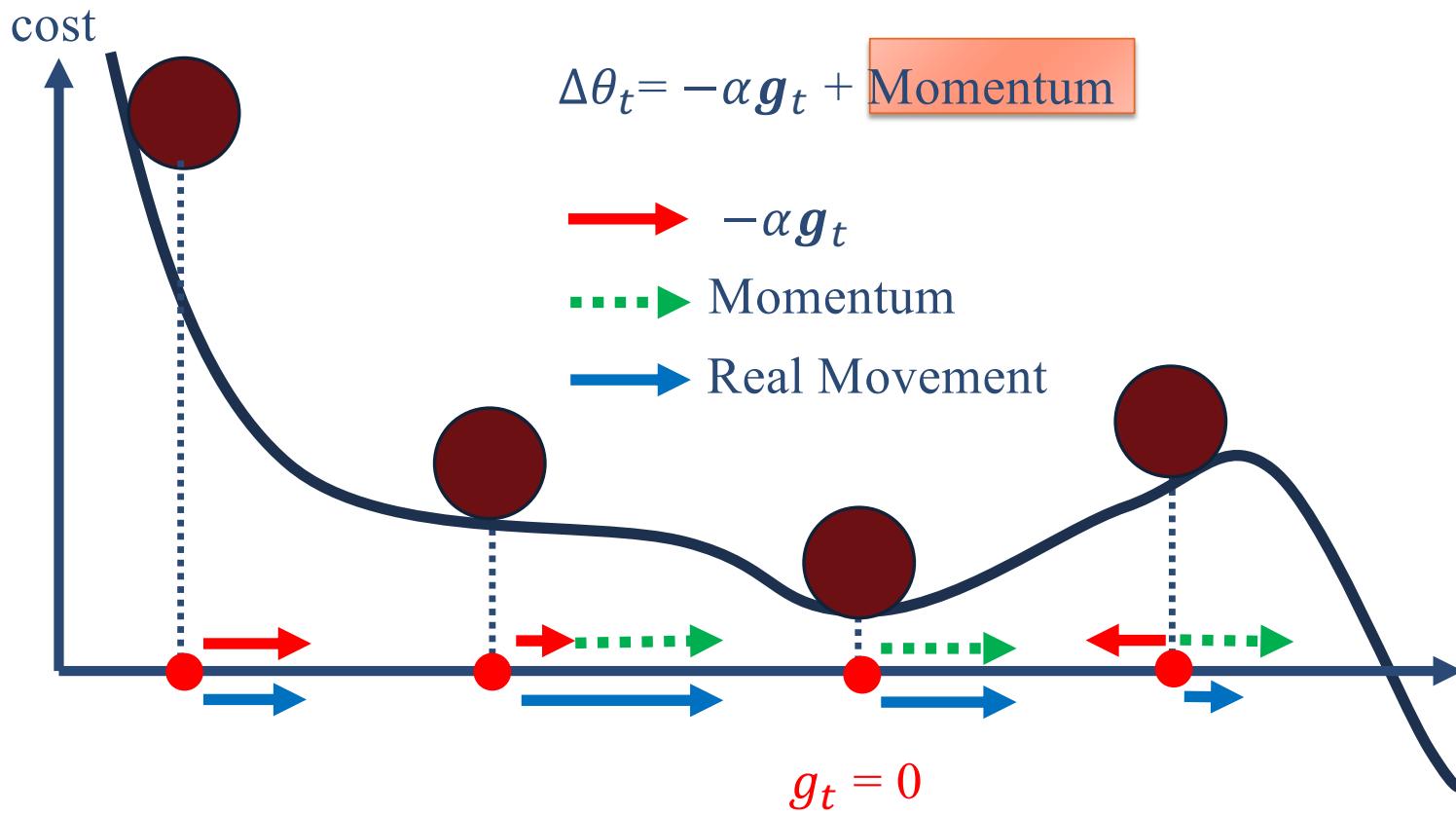
$$= (1 - \beta) \sum_{\tau=1}^t \beta^{t-\tau} g_\tau \odot g_\tau,$$

其中  $\beta$  为衰减率,一般取值为 0.9.

# 自适应学习率



# 梯度方向优化



# 梯度方向优化

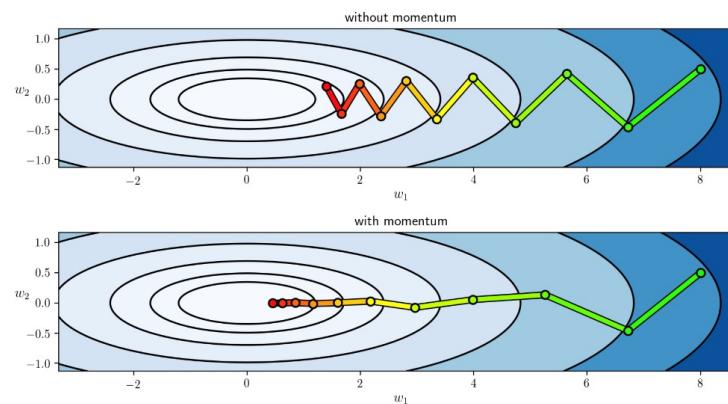
## ► 动量法 (Momentum Method)

► 用之前积累动量来替代真正的梯度。每次迭代的梯度可以看作是加速度。

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，

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha g_t, = -\alpha \sum_{\tau=1}^t \beta^{t-\tau} g_\tau, \quad (7.15)$$

其中 $\rho$ 为动量因子，通常设为0.9， $\alpha$ 为学习率。



<https://www.quora.com/What-exactly-is-momentum-in-machine-learning>

# 梯度方向优化

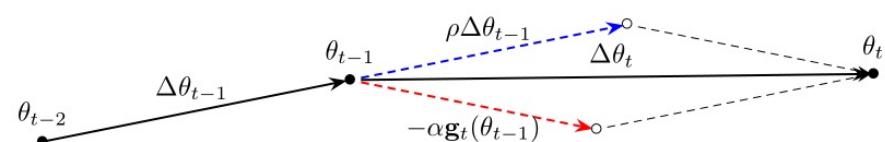
## ► Nesterov 加速梯度算法(Nesterov Accelerated Gradient)

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

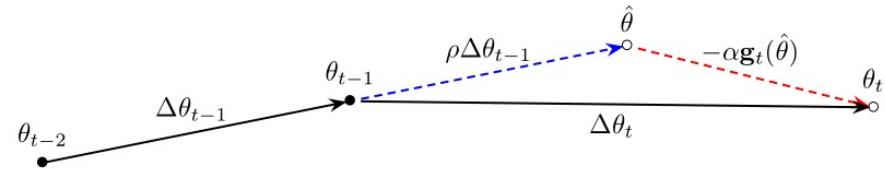
$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathbf{g}_t$$



$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathbf{g}_t(\theta_{t-1} + \rho\Delta\theta_{t-1})$$



(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

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

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

► 偏差修正

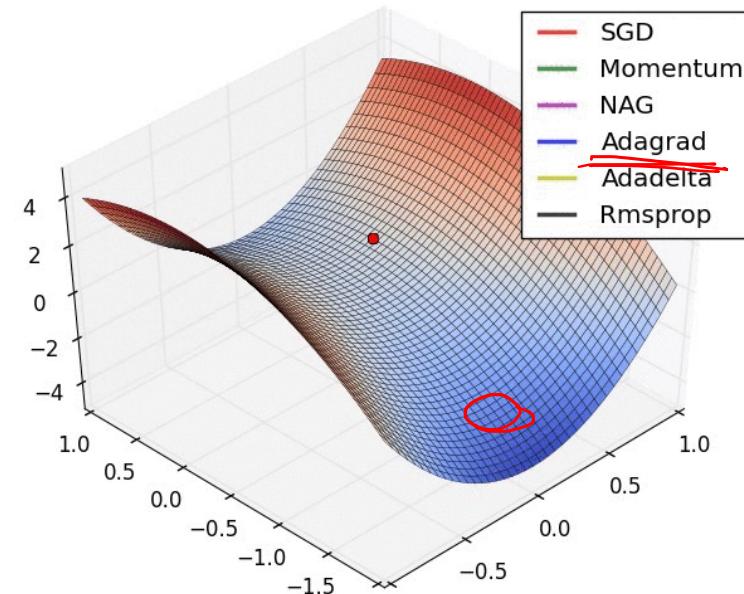
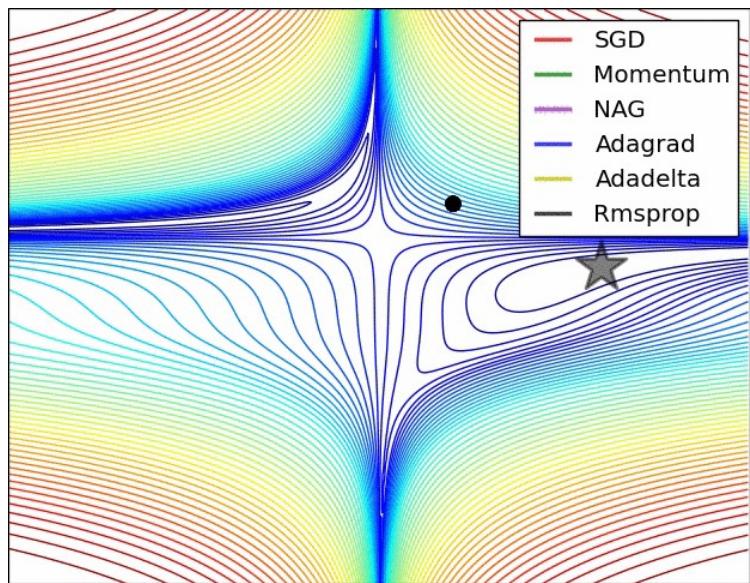
$$\hat{M}_t = \frac{M_t}{1 - \beta_1^t},$$

$$\hat{G}_t = \frac{G_t}{1 - \beta_2^t}.$$

► 更新

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

# 优化



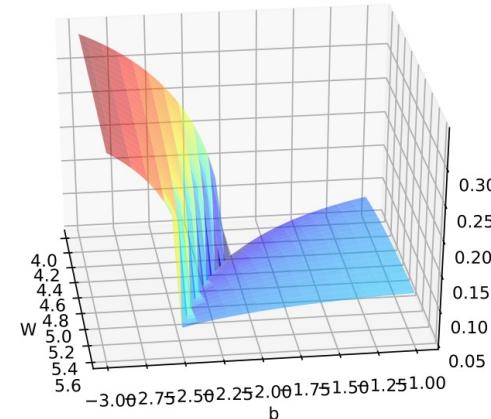
鞍点

## 梯度截断

► 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。

► 按值截断  $\mathbf{g}_t = \max(\min(\mathbf{g}_t, b), a)$

► 按模截断  $\mathbf{g}_t = \frac{b}{\|\mathbf{g}_t\|} \mathbf{g}_t$



# 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：

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

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

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

$\mathbf{g}_t$  为第  $t$  步的梯度

$\alpha_t$  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
	周期性学习率
	自适应学习率
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

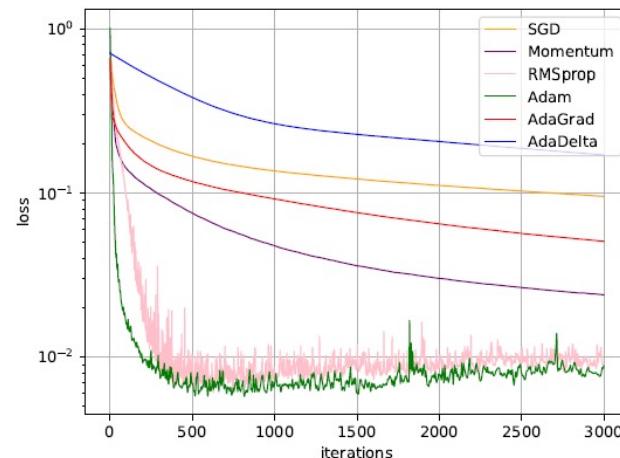


图 7.8 不同优化方法的比较



# 参数初始化

---

- ▶ 参数不能初始化为0！为什么？
- ▶ 对称权重问题！

```
self.W_xz = torch.nn.Parameter(torch.tensor(np.random.normal(0, 0.01, size=(num_inputs, num_hiddens)), dtype=torch.float32))
self.W_hz = torch.nn.Parameter(torch.tensor(np.random.normal(0, 0.01, size=(num_hiddens, num_hiddens)), dtype=torch.float32))
self.b_z = torch.nn.Parameter(torch.zeros(num_hiddens))
```

# 参数初始化

---

- ▶ 初始化方法
  - ▶ 预训练初始化
  - ▶ 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

- 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。

基于固定方差的参数初始化

# 参数初始化

## ▶ 基于方差缩放的参数初始化

### ▶ Xavier 初始化和 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	$r = \sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2 = \frac{2}{M_{l-1}}$

根据每层的神经元数量来自动计算初始化参数方差的方法

考虑到激活神经元比例计算初始化参数方差的方法

# 随机初始化

---

- ▶ 范数保持性 ( Norm-Preserving )
- ▶ 一个 L 层的等宽数线性网络

$$\mathbf{y} = \mathbf{W}^{(L)} \mathbf{W}^{(L-1)} \dots \mathbf{W}^{(1)} \mathbf{x}$$

- ▶ 为了避免梯度消失或梯度爆炸问题，我们希望误差项

$$\|\delta^{(l-1)}\|^2 = \|\delta^{(l)}\|^2 = \|(\mathbf{W}^{(l)})^\top \delta^{(l)}\|^2$$

$$(AX)^T(AX) = X^T A^T AX = X^T X$$

# 参数初始化

---

## ► 正交初始化

- 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

# 数据预处理

---

- ▶ 样本特征由于来源以及度量单位不同，尺度(scale)也不同
- ▶ 一个机器学习算法在缩放全部或部分特征后不影响它的学习和预测，该算法具有尺度不变性
- ▶ 线性分类器 vs. 最近邻分类器

# 数据预处理

► 数据归一化

$$\hat{x}^{(n)} = \frac{x^{(n)} - \min_n(x^{(n)})}{\max_n(x^{(n)}) - \min_n(x^{(n)})},$$

► 最小最大值归一化

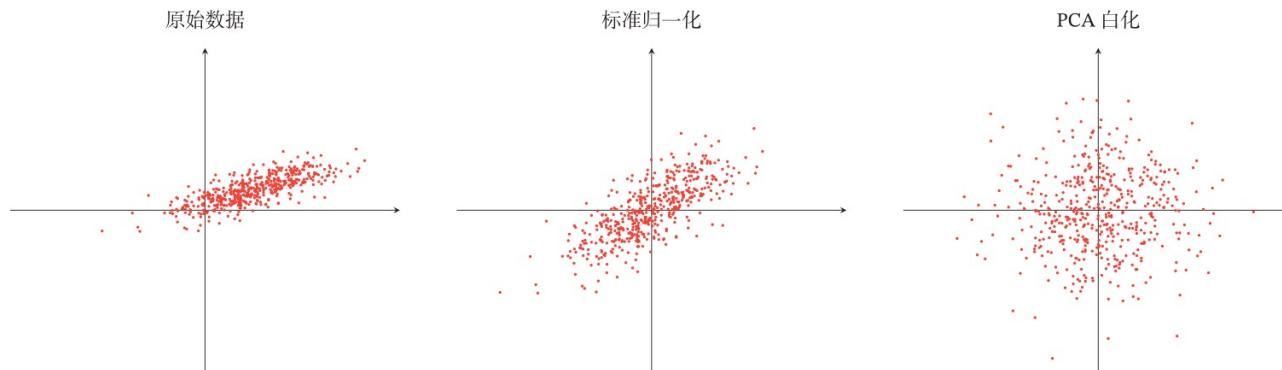
► 标准化



$$\mu = \frac{1}{N} \sum_{n=1}^N x^{(n)},$$

► PCA

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





# 逐层归一化

---

## ▶ 目的

- ▶ 更好的尺度不变性
- ▶ 使大部分神经层的输入处于不饱和区域，更平滑的优化地形

## ▶ 归一化方法

- ▶ 批量归一化 (Batch Normalization, BN)
- ▶ 层归一化 (Layer Normalization)
- ▶ 权重归一化 (Weight Normalization)
- ▶ 局部响应归一化 (Local Response Normalization, LRN)

# 批量归一化

对于一个深层神经网络，令第  $l$  层的净输入为  $\mathbf{z}^{(l)}$ ，神经元的输出为  $\mathbf{a}^{(l)}$ ，即

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) = f(W\mathbf{a}^{(l-1)} + \mathbf{b}), \quad (7.42)$$

其中  $f(\cdot)$  是激活函数， $W$  和  $\mathbf{b}$  是可学习的参数。

► 给定一个包含  $K$  个样本的小批量样本集合，计算均值和方差

$$\begin{aligned}\mu_{\mathcal{B}} &= \frac{1}{K} \sum_{k=1}^K \mathbf{z}^{(k,l)}, \\ \sigma_{\mathcal{B}}^2 &= \frac{1}{K} \sum_{k=1}^K (\mathbf{z}^{(k,l)} - \mu_{\mathcal{B}}) \odot (\mathbf{z}^{(k,l)} - \mu_{\mathcal{B}}).\end{aligned}$$

► 批量归一化

$$\begin{aligned}\hat{\mathbf{z}}^{(l)} &= \frac{\mathbf{z}^{(l)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \odot \gamma + \beta \\ &\triangleq \text{BN}_{\gamma, \beta}(\mathbf{z}^{(l)}),\end{aligned}$$

# 层归一化

---

► 第 $l$ 层神经元的净输入为 $z^{(l)}$

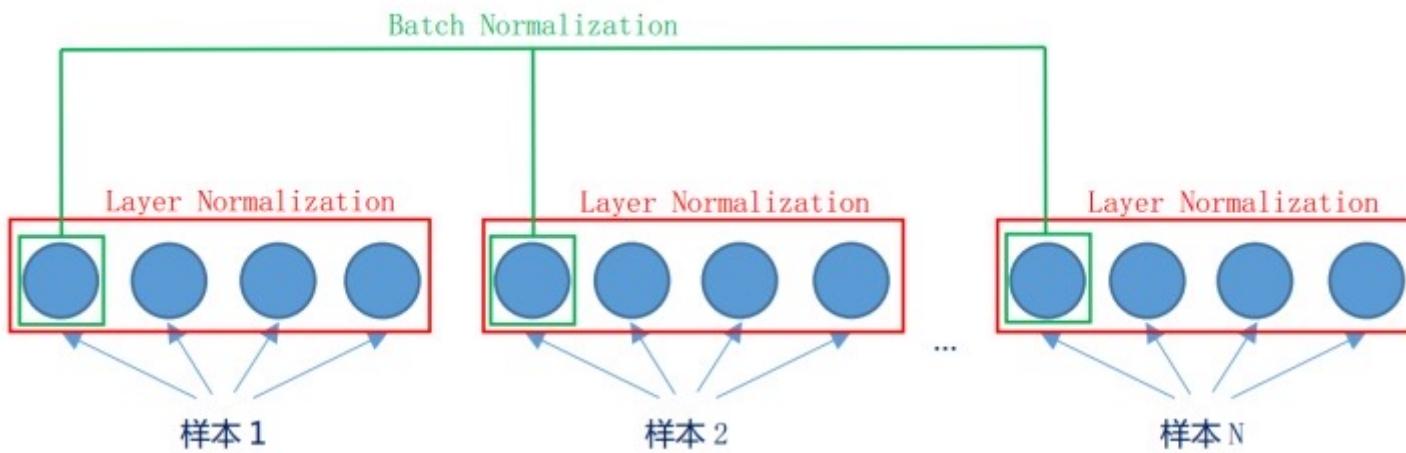
$$\mu^{(l)} = \frac{1}{n^l} \sum_{i=1}^{n^l} z_i^{(l)},$$

$$\sigma^{(l)2} = \frac{1}{n^l} \sum_{i=1}^{n^l} (z_i^{(l)} - \mu^{(l)})^2,$$

► 层归一化定义为

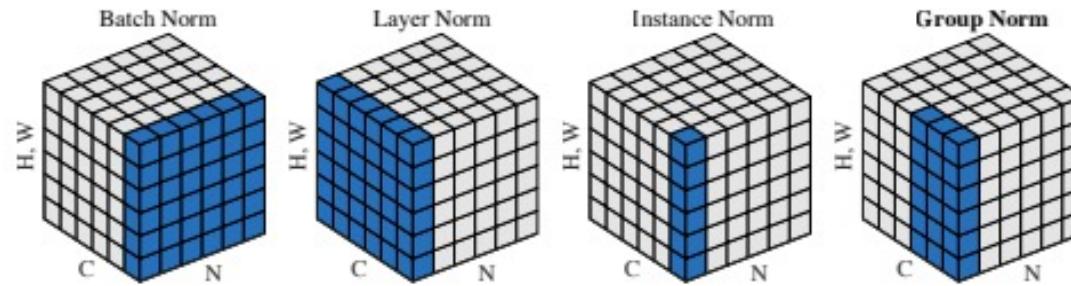
$$\begin{aligned}\hat{z}^{(l)} &= \frac{z^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)2} + \epsilon}} \odot \gamma + \beta \\ &\triangleq \text{LN}_{\gamma, \beta}(z^{(l)}),\end{aligned}$$

# 批量归一化VS层归一化

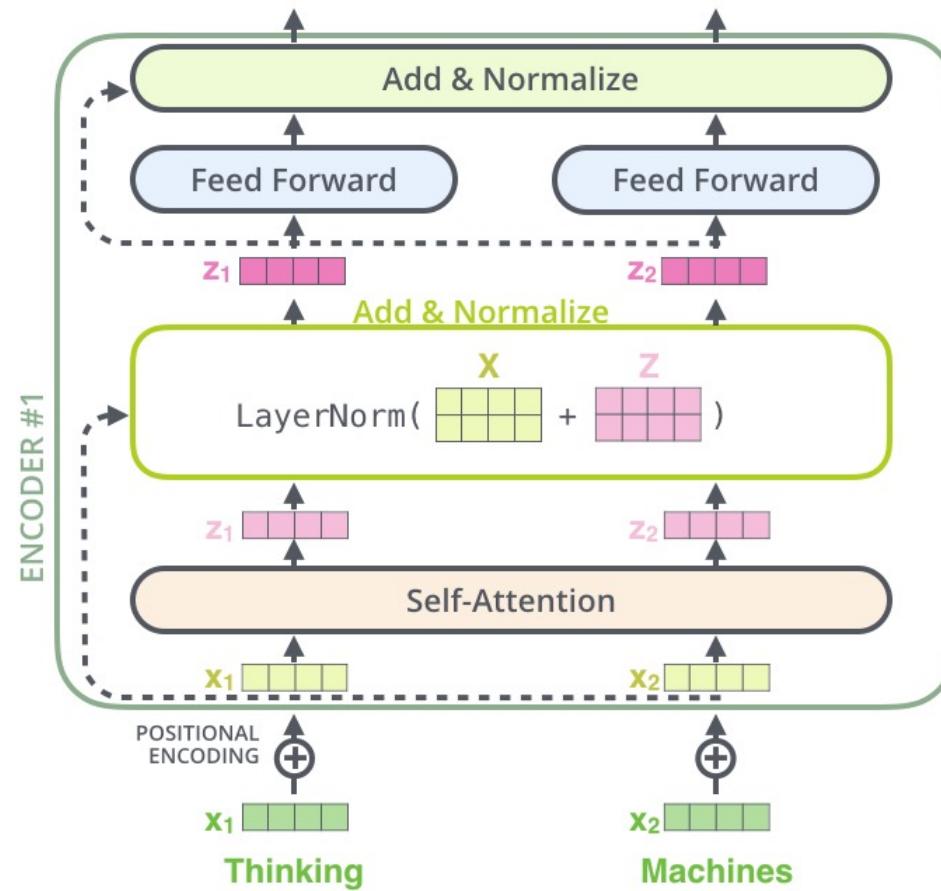


# 更多的归一化

---



# Transformer





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

# 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  $\alpha$ ，我们可以设置

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

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



# 重新思考泛化性

- ▶ 神经网络
- ▶ 过度参数化
- ▶ 拟合能力强

↓  
泛化性差



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)

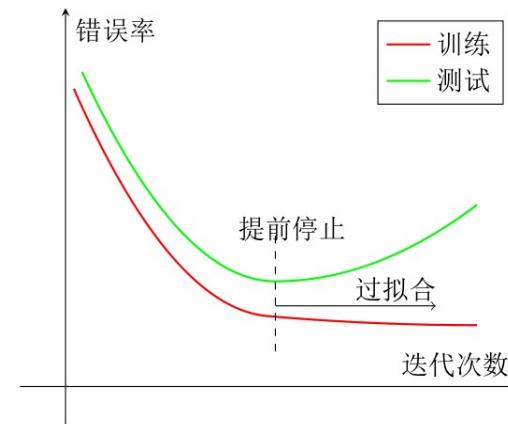
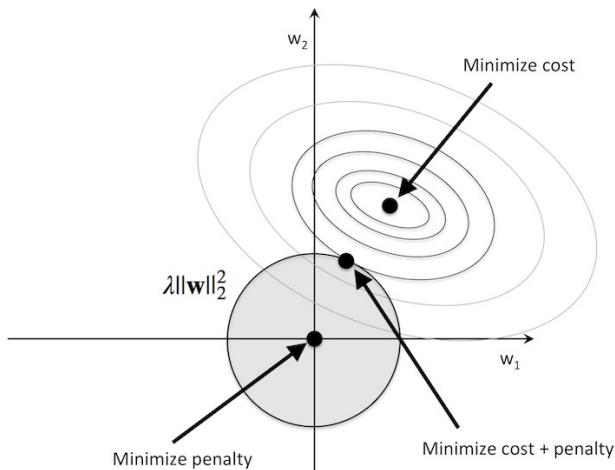
所有损害优化的方法都是正则化。

增加优化约束

干扰优化过程

L1/L2约束、数据增强

权重衰减、随机梯度下降、提前停止



# 正则化

---

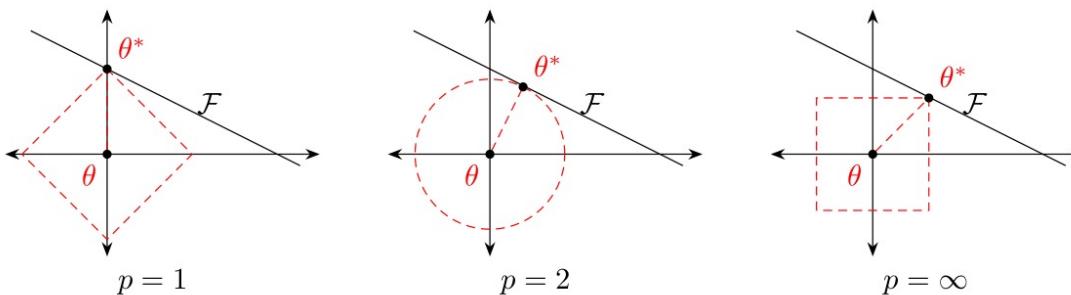
- ▶ 如何提高神经网络的泛化能力
  - ▶  $\ell_1$  和  $\ell_2$  正则化
  - ▶ early stop
  - ▶ 权重衰减
  - ▶ Dropout
  - ▶ 数据增强

## $\ell_1$ 和 $\ell_2$ 正则化

► 优化问题可以写为

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

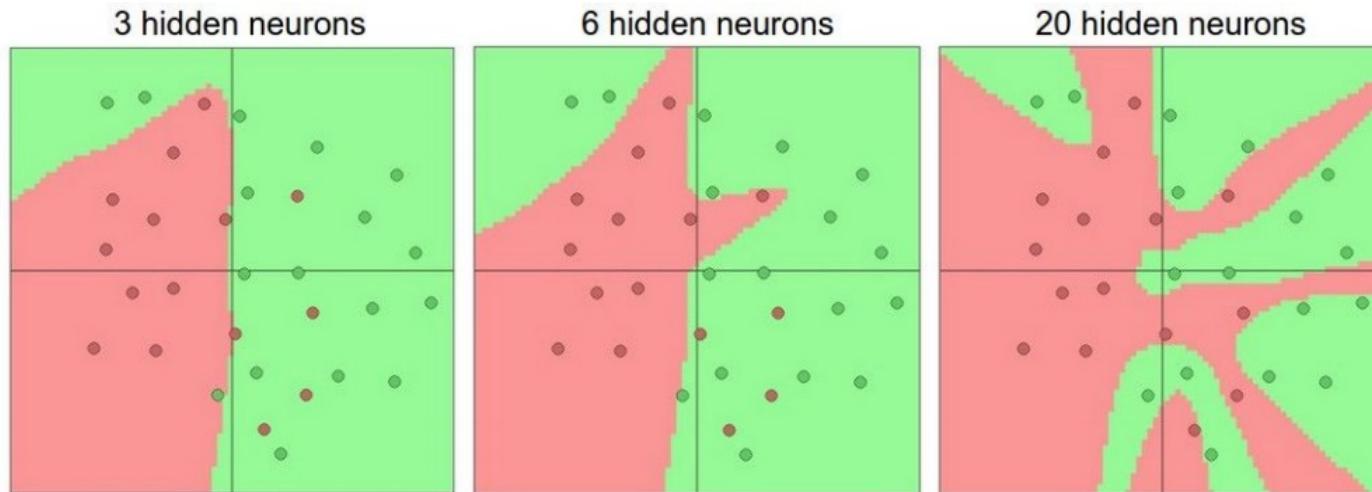
►  $\ell_p$  为范数函数，p的取值通常为{1,2}代表 $\ell_1$ 和 $\ell_2$ 范数， $\lambda$ 为正则化系数。



# 神经网络示例

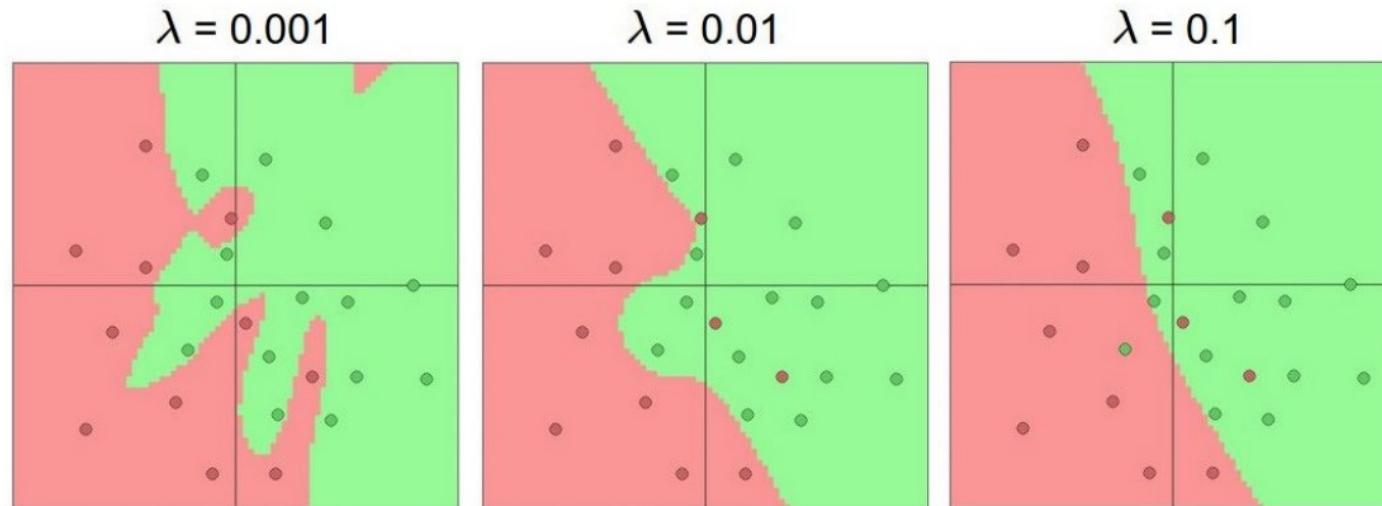
<http://playground.tensorflow.org/>

## ► 隐藏层的不同神经元个数



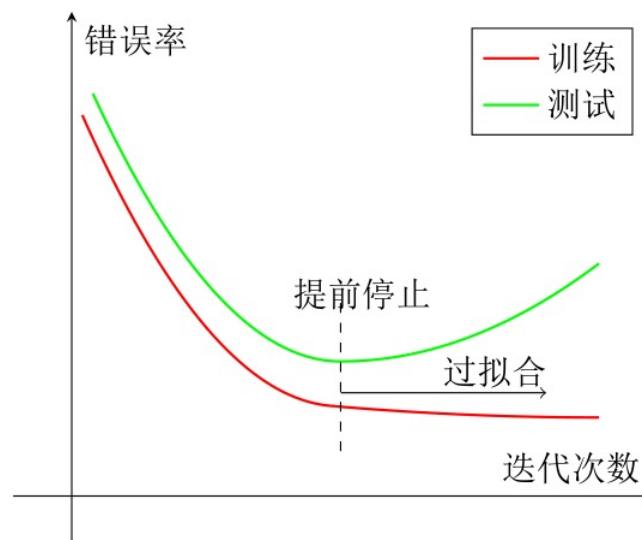
# 神经网络示例

## ► 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 $w$ 。

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

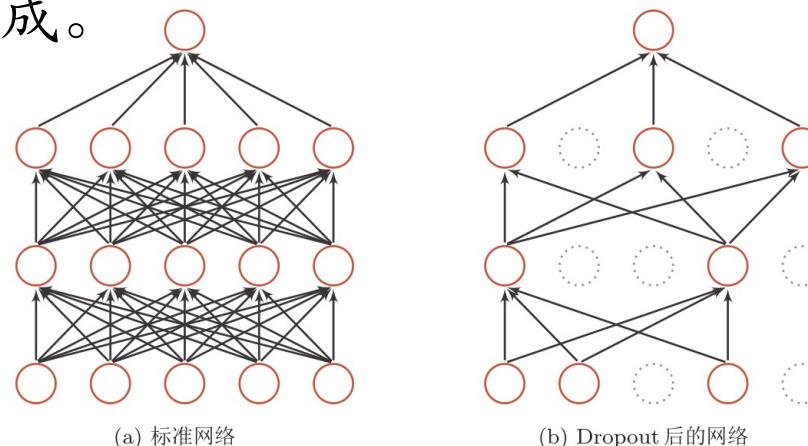
- ▶ 在标准的随机梯度下降中，权重衰减正则化和 $\ell_2$ 正则化的效果相同。
- ▶ 在较为复杂的优化方法（比如Adam）中，权重衰减和 $\ell_2$ 正则化并不等价。

# 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  $p$  的伯努利分布随机生成。



# 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 $2^n$ 个子网络。

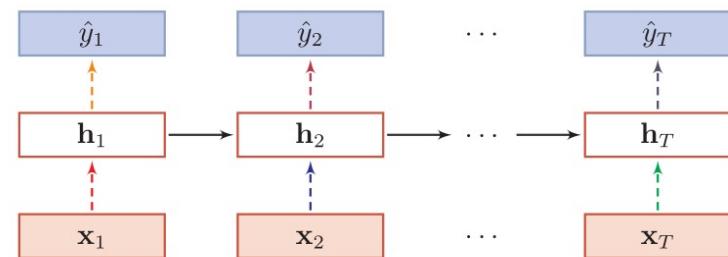
## ▶ 贝叶斯学习的解释

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

- ▶ 其中 $f(\mathbf{x}, \theta_m)$ 为第 $m$ 次应用丢弃方法后的网络。其参数 $\theta_m$ 为对全部参数 $\theta$ 的一次采样。

# 循环神经网络上的丢弃法

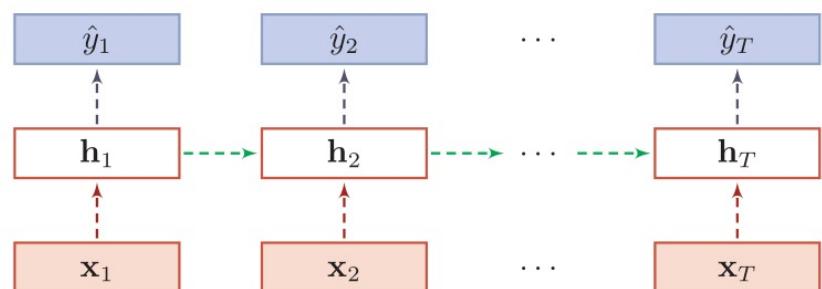
► 当在循环神经网络上应用丢弃法，不能直接对每个时刻的隐状态进行随机丢弃，这样会损害循环网络在时间维度上记忆能力。



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 $\theta$ 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。

# 数据增强

<https://zhuanlan.zhihu.com/p/145521255>

- ▶ 文本数据的增强主要是通过算法对文本进行转变，引入噪声等方法来增加数据的多样性。
- ▶ 文本数据的增强方法：
  - ▶ 词汇替换
  - ▶ 回译(back translation)
  - ▶ 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one-hot向量表示

$$\mathbf{y} = [0, \dots, 0, 1, 0, \dots, 0]^T \quad \text{硬目标 (Hard Targets)}$$

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

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

# 教学内容

---

- ▶ 网络优化
- ▶ 优化算法
  - ▶ 学习率优化(Adagrad, RMSprop, Adadelta), 梯度修正(Momentum, Nesterov), Adam
- ▶ 参数初始化
- ▶ 数据预处理
- ▶ 逐层归一化
  - ▶ Batch normalization, Layer normalization
- ▶ 超参数优化
- ▶ 网络正则化