FDU 神经网络 4. 神经网络优化

本文参考以下教材:

- 神经网络与深度学习 (邱锡鹏) 第 7 章
- Deep Learning (I. Goodfellow, Y. Bengio, A. Courville) Chapter 8
- 深度学习 (I. Goodfellow, Y. Bengio, A. Courville, 赵申剑等译) 第 8 章

欢迎批评指正!

4.1 An Introduction

虽然神经网络具有非常强的表达能力，但实际应用时依然存在一些问题:

- **① 优化问题:**
  深度神经网络的优化十分困难.
  首先，神经网络的损失函数是一个非凸函数，找到全局最优解通常比较困难;
  其次，深度神经网络的参数通常非常多，训练数据也比较多，
  因此无法使用计算代价高的二阶优化方法，而一阶优化方法的训练效率通常比较低;
  此外，深度神经网络存在梯度消失或爆炸问题，导致基于梯度的优化方法经常失效.
- **② 泛化问题:**
  由于深度神经网络的复杂度比较高，并且拟合能力很强，很容易在训练集上产生过拟合.
  因此在训练深度神经网络时，我们需要引入正则化方法来提升网络的泛化能力.
  这些从实践中总结的经验方法可以帮助我们在神经网络的拟合能力和泛化能力之间找到比较好的平衡.

4.1.1 鞍点

深度神经网络的参数非常多，其参数学习是高维空间中的非凸优化问题.
其难点不在于如何逃离局部最优点，而是如何逃离**鞍点** (saddle point).
目标函数在鞍点的梯度是 0，但在一些维度上是极大值点，在另一些维度上是极小值点:

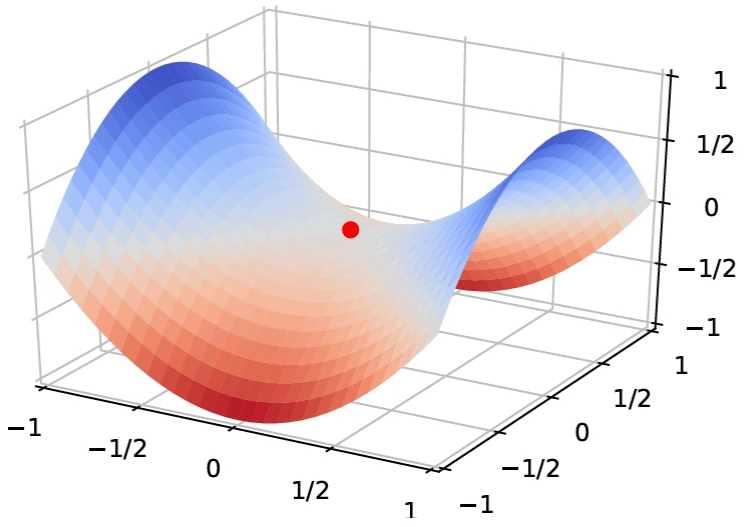
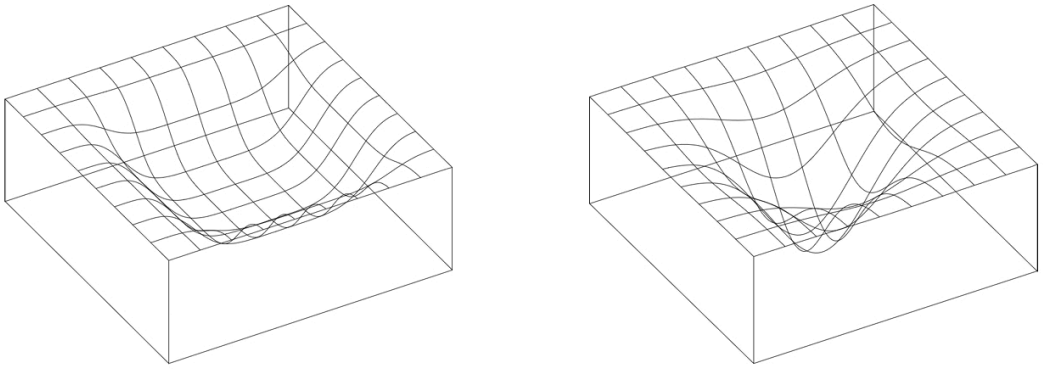


图 7.1　鞍点示例

在高维空间中，**局部最小点** (local minima) 要求在每一维度上都是极小值点，这种概率非常低.
因此在高维空间中，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，很难从鞍点处逃离.
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.

4.1.2 平坦最小值

深度神经网络的参数非常多，并且有一定的冗余性，这使得单个参数对最终损失的影响都比较小，
这会导致损失函数在局部最小点附近通常是一个平坦的区域，称为**平坦最小值** (flat minima)



(a) 平坦最小值　　(b) 尖锐最小值

图 7.2　平坦最小值和尖锐最小值的示例 (图片来源 : [Hochreiter et al., 1997])

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.
这意味着我们在训练神经网络时，不需要精确地找到一个局部最小点，只要进入一个局部最小点的邻域就足够了.
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点时，其稳健性会更好，即微小的参数扰动不会显著影响模型能力;
而当一个模型收敛到一个尖锐的局部最小点时，其稳健性会比较差.

4.1.3 局部最小点的等价性

在深度神经网络中，大部分的局部最小点在测试集上的性能是相近的.
此外，局部最小点对应的训练损失都可能非常接近于全局最小点对应的训练损失.
虽然神经网络有一定概率收敛于比较差的局部最小点，
但随着网络规模增加，网络陷入比较差的局部最小点的概率会大大降低.
在训练神经网络时，我们通常没有必要找全局最小点，这反而可能导致过拟合.

4.1.4 神经网络优化

改善神经网络优化的目标是找到更好的局部最小点和提高优化效率.
目前比较有效的经验方法如下:

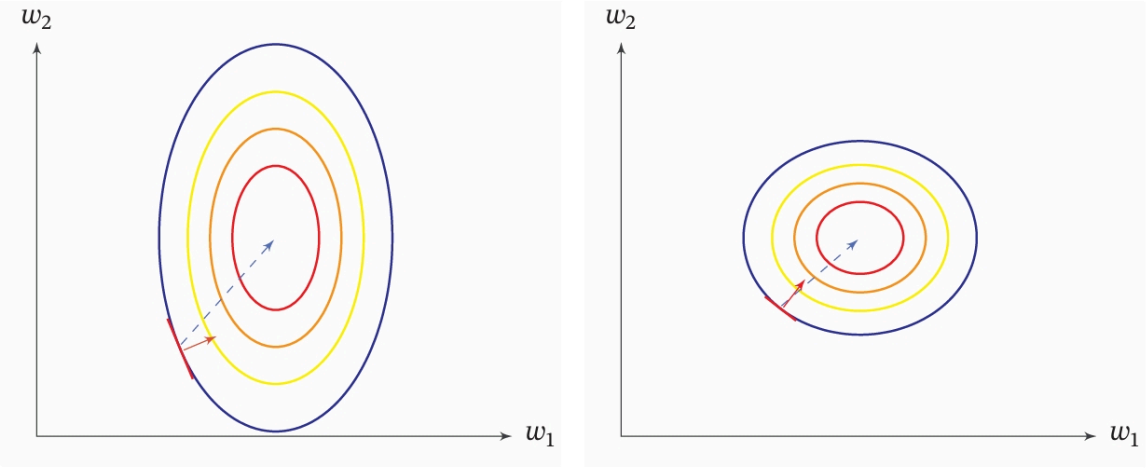
- ① 使用更有效的优化算法来提高梯度下降优化方法的效率和稳定性，比如动态学习率调整、梯度修正等.

- ② 使用更好的参数初始化方法、数据预处理方法来提高优化效率.
- ③ 修改网络结构来得到更好的优化地形 (optimization landscape),
  比如使用 ReLU 激活函数、残差连接、逐层归一化等.
  (优化地形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)
- ④ 使用更好的超参数优化方法.

4.2 数据准备

4.2.1 数据预处理

一般而言，样本特征由于来源以及度量单位不同，它们的尺度 (scale) 往往差异很大.
不同机器学习模型对数据特征尺度的敏感程度不一样.
如果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，
我们就称该算法具有**尺度不变性** (Scale Invariance)，
比如线性分类器是尺度不变的，而最近邻分类器就是尺度敏感的.
当我们计算不同样本之间的 Euclid 距离时，尺度大的特征会起到主导作用.
因此对于尺度敏感的模型，必须先对样本进行预处理，
将各个维度的特征转换到相同的取值区间，并且消除不同特征之间的相关性，才能获得比较理想的结果.
理论上，神经网络具有尺度不变性，可以通过参数的调整来适应不同特征的尺度.
但尺度不同的输入特征会增加训练难度.



(a) 未归一化数据的梯度　　(b) 归一化数据的梯度

图 7.9　数据归一化对梯度的影响

归一化 (normalization) 指把数据特征转换为相同尺度的方法:

- **最小最大值归一化 (Min-Max Normalization)**
  将每一维特征的取值范围归一到 $[0,1]$ 或 $[-1,1]$ 之间，以前者的一维情况为例:

$$\tilde{x} := \frac{x - \min}{\max - \min}$$

- **标准化 (Standardization)**
  将每一维特征的分布的均值标准化为 0，方差标准化为 1:

$$\tilde{x} := \frac{x - \mathrm{mean}}{\mathrm{standard\_error}}$$

  其中标准差不能为零，如果为零，则说明这一维特征没有区分性，可以直接删掉.

- **白化 (whitening)**
  使用主成分分析 (Principal Component Analysis, PCA)
  去除各成分的相关性 (降低冗余)，并且让各成分具有相同的方差.

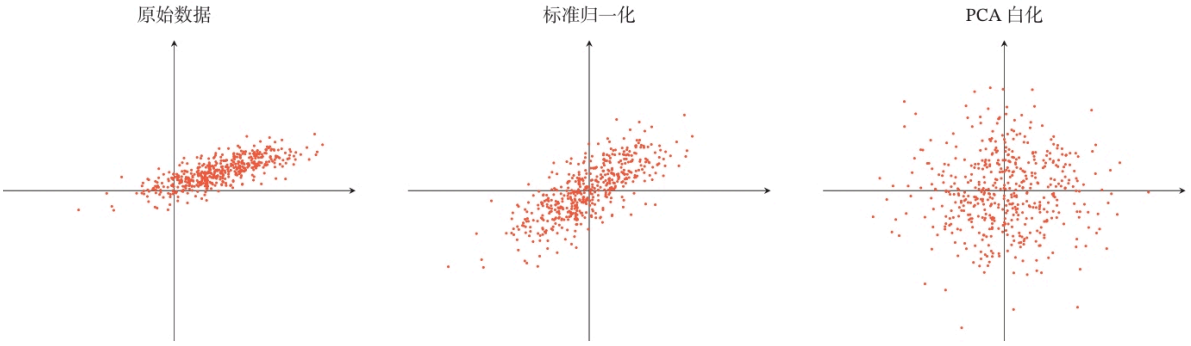


图 7.10　标准归一化和 PCA 白化

4.2.2 数据增强

深度神经网络一般都需要大量的训练数据才能获得比较理想的效果.
在数据量有限的情况下，可以通过数据增强 (data augmentation) 来增加数据量，
提高模型稳健性，避免过拟合.
目前数据增强主要应用在图像数据上，在文本等其他类型的数据上还没有太好的方法.

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.
增强的方法主要有几种:

- 旋转 (rotation): 将图像随机旋转一定角度
- 翻转 (flip): 将图像沿水平或垂直方向随机翻转
- 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
- 噪声 (noise): 加入随机噪声

4.2.3 标签平滑

在数据增强中，我们可以给样本特征加入随机噪声来避免过拟合.
同样我们也可以给样本的标签引入一定的噪声.
假设训练数据集中有一些样本的标签是被错误标注的，
那么最小化这些样本上的损失函数会导致过拟合.
一种正则化方法是**标签平滑** (label smoothing)，即在输出标签中添加噪声来避免模型过拟合.

一个样本 x 的标签可用 one-hot 向量表示:

$$y = [0, \dots, 0, 1, 0, \dots, 0]^{\mathrm{T}} \in \{0, 1\}^K$$

这种标签可以看作**硬目标** (hard target).
如果使用 $\mathrm{softmax}$ 分类器并使用交叉熵损失函数，
最小化损失函数会使得正确类和其他类的权重差异变得很大.
根据 $\mathrm{softmax}$ 函数的性质可知，如果要使得某一类的输出概率接近于 1，
其未归一化的得分需要远大于其他类的得分，可能会导致其权重越来越大，导致过拟合.
此外，如果样本标签是错误的，会导致更严重的过拟合现象.

为了改善这种情况，我们可以引入一个噪声对标签进行平滑.
假设样本以 ε 的概率为其他类，平滑后的标签为:

$$\tilde{y} := \left[\frac{\varepsilon}{K-1}, \ldots, \frac{\varepsilon}{K-1}, 1-\varepsilon, \frac{\varepsilon}{K-1}, \ldots, \frac{\varepsilon}{K-1}\right]^{\mathrm{T}} \in \mathbb{R}^K$$

其中 K 为类别总数.
这种标签可以看作**软目标** (soft target).
标签平滑可以避免模型的输出过拟合到硬目标上，并且通常不会损害其分类能力.

上面的标签平滑方法是给其他 $K-1$ 个标签相同的概率 $\frac{\varepsilon}{K-1}$，没有考虑标签之间的相关性.
一种更好的做法是按照类别相关性来赋予其他标签不同的概率，
比如先训练另外一个更复杂 (一般为多个网络的集成) 的**教师网络** (teacher network)，
并使用大网络的输出作为软目标来训练**学生网络** (student network).
这种方法也称为**知识蒸馏** (knowledge distillation)

4.3 网络初始化

神经网络的参数学习是一个非凸优化问题.
当使用梯度下降法来进行优化网络参数时，参数初始值的选取十分关键.
参数初始化的方式通常有以下三种:

- **① 预训练初始化 (pre-trained initialization)**
  一个已经在大规模数据上训练过的模型可以提供好的参数初始值.
  预训练任务可以为监督学习或无监督学习任务.
  由于无监督学习任务更容易获取大规模的训练数据，因此被广泛采用.
  预训练模型在目标任务上的学习过程也称为**精调** (fine-tuning)
- **② 随机初始化 (random initialization)**
  在线性模型的训练中，我们一般将参数全部初始化为 0
  但这在神经网络的训练中会存在一些问题，因为如果参数都为 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**.
  为了打破这个平衡，比较好的方式是对每个参数都随机初始化.
- **③ 固定值初始化 (constant initialization)**
  对于一些特殊的参数，我们可以根据经验用一个特殊的固定值来进行初始化.
  例如偏置通常用 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.
  在 LSTM 网络的遗忘门中，偏置通常初始化为 1 或 2，使得时序上的梯度变大.
  对于使用 ReLU 的神经元，有时也可以将偏置设为 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.

4.3.1 固定方差随机初始化

最简单的初始化方法是从固定均值 (通常为 0) 和方差为 σ^2 的分布中采样生成初始值:

- Gauss 分布初始化: $N(0, \sigma^2)$
- 均匀分布初始化: $\mathrm{uniform}(-r, r)$

在基于固定方差的随机初始化方法中，比较关键的是如何设置方差 σ^2.
如果参数范围取的太小，一是会导致神经元的输出过小，经过多层之后信号就慢慢消失了;
二是还会使得 Sigmoid 型激活函数丢失非线性的能力，因为 Sigmoid 型函数在 0 附近是近似线性的.
这样多层神经网络的优势也就不存在了.
如果参数范围取的太大，会导致输入状态过大，
使得 Sigmoid 型激活函数饱和，梯度接近于 0，从而导致梯度消失问题.
为降低固定方差对网络性能以及优化效率的影响，
基于固定方差的随机初始化方法一般需要配合逐层归一化来使用.

4.3.2 自适应方差随机初始化

初始化一个深度网络时，为了缓解梯度消失或爆炸问题，
我们尽可能保持每个神经元的输入和输出的方差一致，
根据神经元的连接数量来自适应地调整初始化分布的方差，
这类方法称为**方差缩放** (variance scaling)

- **① Xavier Glorot 初始化**
  它的想法是让激活值在前向传播中方差不被放大或缩小，
  同时让梯度在反向传播中方差也不被放大或缩小.
  在计算出参数的理想方差后，就可用 Gauss 分布或均匀分布来随机初始化参数.

  记第 l 层神经元数量为 M_l，激活函数为 Sigmoid 型函数.
  第 l 层的连接权重可按 $N(0, 2\rho/(M_{l-1} + M_l))$ 初始化，
  或按 $\text{uniform}(-\sqrt{6\rho/(M_{l-1} + M_l)}, \sqrt{6\rho/(M_{l-1} + M_l)})$ 初始化.
  其中 ρ 是根据经验设定的缩放因子.

- **② He Kaiming 初始化**
  记第 l 层神经元数量为 M_l，激活函数为 ReLU 型函数.
  注意到通常只有一半的神经元输出为 0.
  假设只考虑前向传播，不考虑反向传播.
  第 l 层的连接权重可按 $N(0, 2/M_{l-1})$ 初始化，
  或按 $\text{uniform}(-\sqrt{6/M_{l-1}}, \sqrt{6/M_{l-1}})$ 初始化.

表 7.2　Xavier 初始化和 He 初始化的具体设置情况

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

4.3.3 正交初始化

上述随机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，初始化得到的权重矩阵依然可能存在梯度消失或梯度爆炸问题.
正交初始化的目标是使激活值在前向传播中 Euclid 范数近似不变，梯度在反向传播中的 Euclid 范数近似不变.
因此我们可以将第 l 层的权重矩阵 $W^{(l)}$ 直接初始化为行 (标准) 正交或列 (标准) 正交的矩阵.
可以先使用标准 Gauss 分布 $N(0,1)$ 生成一个与 $W^{(l)}$ 同形的矩阵，
然后进行奇异值分解，取其左奇异向量或右奇异向量 (并乘以某个缩放系数 ρ) 构成 $W^{(l)}$.

4.4 优化算法

4.4.1 小批量梯度下降

在训练深度神经网络时，训练数据的规模通常都比较大.
如果在梯度下降时，每次迭代都要计算整个训练数据上的梯度，这就需要比较多的计算资源.
实际上，大规模训练集中的数据通常会非常冗余，没有必要在整个训练集上计算梯度.
因此在训练深度神经网络时，我们经常使用**小批量梯度下降法** (mini-batch gradient descent)
具体来说，我们每轮迭代只使用小批量的样本计算梯度，每遍历完一次训练数据，就称为一**回合** (epoch):

$$1\ \text{Epoch}\ :=\left(\frac{\text{number of training data}}{\text{batch size}}\right)\times\text{Iteration}$$

记深度神经网络为 $f(x;\theta)$，损失函数为 $\mathcal{L}(y,f(x;\theta))$，其中 θ 为网络参数.
记第 k 轮迭代的网格参数为 $\theta^{(k-1)}$
我们每次选取 batch_size 个训练样本，记第 k 轮迭代选取的样本为 $\mathcal{S}_k:=\{(x^{(i_j)},y^{(i_j)})\}_{j=1}^{\text{batch\_size}}$
于是第 k 轮损失函数关于 θ 的梯度为:

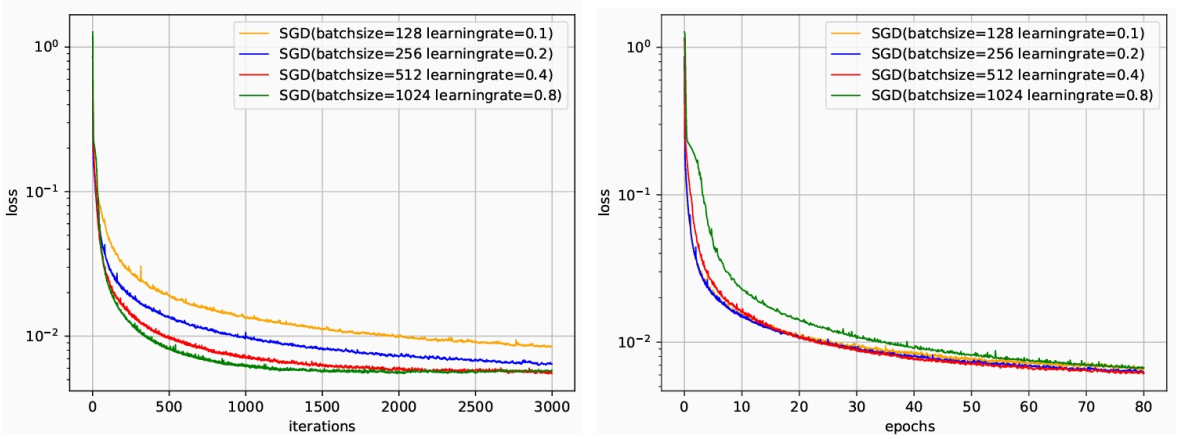
$$g^{(k)}:=\frac{1}{\text{batch\_size}}\sum_{(x,y)\in\mathcal{S}_k}\frac{\partial\mathcal{L}(y,f(x;\theta^{(k-1)}))}{\partial\theta}$$

则第 k 轮参数的迭代格式为:

$$\theta^{(k)}=\theta^{(k-1)}-\alpha_k g^{(k)}$$

其中 $\alpha_k>0$ 是第 k 轮的学习率.

批量大小 batch_size 对网络优化的影响也很大.
一般而言，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.
批量大小越大，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.
当批量大小较小时，需要设置较小的学习率，否则模型会不收敛.
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): 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.



(a) 按迭代（Iteration）的损失变化　　(b) 按回合（Epoch）的损失变化

图 7.3　在 MNIST 数据集上批量大小对损失下降的影响

从左图可以看出，批量大小越大，下降效果越明显，下降曲线也越平滑.
但从右图可以看出，如果按整个数据集上的回合数来看，则是批量大小越小，下降效果越明显.
适当小的批量可以加速收敛.

此外，批量大小和模型的泛化能力也有一定的关系.
批量越大，越有可能收敛到尖锐最小值;
批量越小，越有可能收敛到平坦最小值.

4.4.2 学习率调整

(1) 学习率衰减

从经验上看，学习率在一开始要保持大些来保证收敛速度，在收敛到最优点附近时要小些以避免来回振荡.
比较简单的学习率调整策略是**学习率衰减** (learning rate decay)
学习率衰减可以按每次迭代进行，也可以按每若干次迭代或每个回合进行.
简单起见，我们这里默认按每次迭代进行，记第 k 轮迭代的学习率为 α_k.

- ① 分段常数衰减 (piecewise constant decay)

$$\alpha_k := \beta_k \alpha_0, \quad \text{if } k \in [T_k, T_{k+1})$$
$$\text{where } \{\beta_k\} \text{ is a decreasing sequence within the interval } (0, 1)$$

- ② 逆时衰减 (inverse time decay)

$$\alpha_k := \alpha_0 \frac{1}{1 + \beta k}$$
$$\text{where } \beta > 0$$

- ③ 指数衰减 (exponential decay)

$$\alpha_k := \alpha_0 \beta^k$$
$$\text{where } 0 < \beta < 1$$

也可采取以下形式:

$$\alpha_k := \alpha_0 \exp(-\beta k)$$
$$\text{where } \beta > 0$$

- ④ 余弦衰减 (cosine decay)

$$\alpha_k := \frac{1}{2}\alpha_0 \left(1 + \cos(\frac{k\pi}{\text{max\_iter}})\right)$$

其中 max_iter 是最大迭代次数.

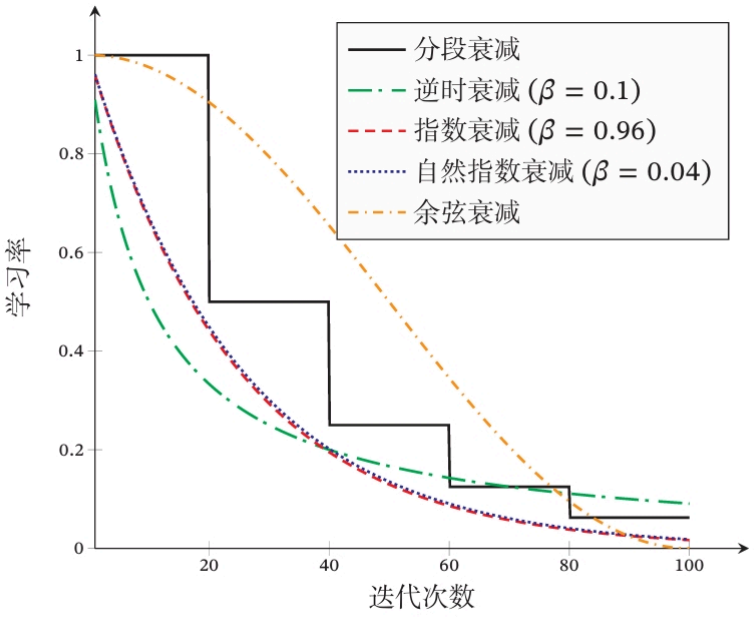


图 7.4 不同学习率衰减方法的比较

(2) 学习率预热

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)
在预热过程中，每次更新的学习率为:

$$\alpha_k := \frac{k}{\text{warm\_up\_iter}}\alpha_0$$
$$\text{where } 1 \leq k \leq \text{warm\_up\_iter}$$

当预热过程结束，再选择一种学习率衰减方法来逐渐降低学习率.

(3) 周期性学习率调整

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内.
因此周期性地增大学习率虽然可能短期内损害优化过程，使得网络收敛的稳定性变差，
但从长期来看有助于找到更好的局部最优解.

- **① 三角循环学习率**
  假设每个循环周期的长度相等都为 $2\Delta T$，
  其中前 ΔT 步为学习率线性增大阶段，后 ΔT 步为学习率线性缩小阶段.
  在第 k 次迭代时，其所在的循环周期数为:

$$m := \left\lfloor 1 + \frac{k}{2\Delta T} \right\rfloor$$

  于是第 k 次迭代的学习率为:

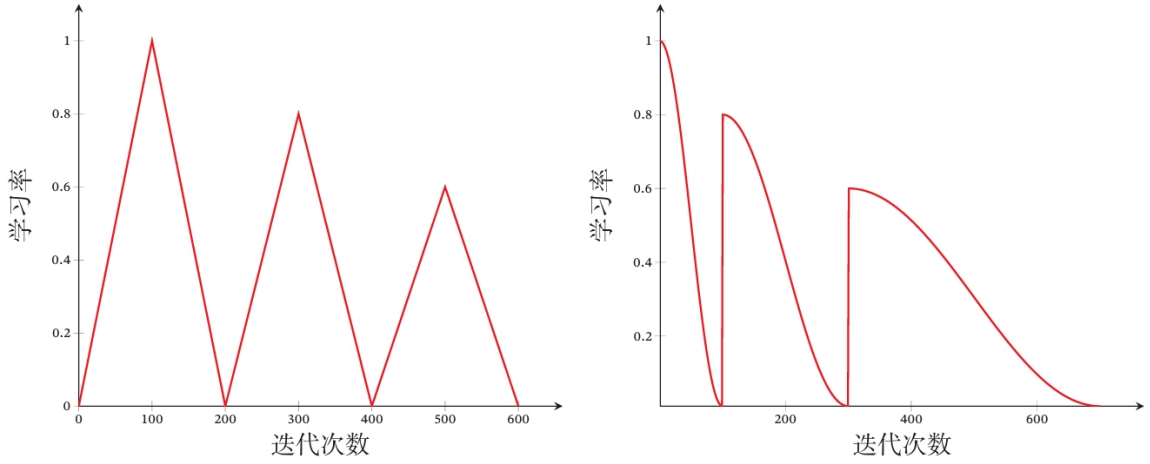
$$\alpha_k := \alpha_{\min}^{(m)} + (\alpha_{\max}^{(m)} - \alpha_{\min}^{(m)}) \max(0, 1 - b_k)$$
$$\text{where } b_k := \left| \frac{k}{\Delta T} - 2m + 1 \right|$$

  其中 $\alpha_{\max}^{(m)}$ 和 $\alpha_{\min}^{(m)}$ 分别是第 m 个周期中学习率的上界和下界，可随 m 的增大逐渐降低.

- **② 带热重启的随机梯度下降**
  第 m 次重启在第 $m-1$ 次重启后第 T_m 个回合后进行，采样余弦衰减来降低学习率.
  第 k 次迭代的学习率为:

$$\alpha_k := \alpha_{\min}^{(m)} + \frac{1}{2}(\alpha_{\max}^{(m)} - \alpha_{\min}^{(m)}) \left(1 + \cos(\frac{T_{\text{epoch}}}{T_m}\pi)\right)$$

  其中 $\alpha_{\max}^{(m)}$ 和 $\alpha_{\min}^{(m)}$ 分别是第 m 个周期中学习率的上界和下界，可随 m 的增大逐渐降低.
  T_{epoch} 是上次重启后的回合数 (可取成小数，例如 0.1，这样可以在回合内部进行学习率衰减)
  重启周期 T_m 可随着重启次数增大而逐渐增加，例如 $T_m = T_{m-1} \times \kappa$，其中 $\kappa > 1$ 是放大因子.



(a) 三角循环学习率　　(b) 带热重启的余弦衰减

图 7.5　周期性学习率调整

(4) AdaGrad 算法

在标准的梯度下降法中，每个参数在每次迭代时都使用相同的学习率.
这似乎有些不太合理，因此我们可以根据不同参数的收敛情况分别设置学习率.
AdaGrad 算法 (Adaptive Gradient Algorithm) 在每轮迭代自适应地调整每个参数的学习率.
在第 k 次迭代时，先计算每个参数梯度平方的累计值:

$$\text{sum}_k := \text{sum}_{k-1} + g^{(k)} \odot g^{(k)} = \sum_{t=1}^{k} g^{(t)} \odot g^{(t)}$$

$$\text{where } \odot \text{ denotes element-wise (Hadamard) multiplication}$$

第 k 次迭代的迭代格式为:

$$\theta_k := \theta_{k-1} - \frac{\alpha}{\sqrt{\text{sum}_k + \varepsilon}} \odot g^{(k)}$$

其中 $\alpha > 0$ 是初始学习率向量，$\varepsilon > 0$ 是为避免零除而设置的小常数 ($1 \times 10^{-10} \sim 1 \times 10^{-7}$)
这里的开方、除、加运算都是逐元素进行的操作.

在 AdaGrad 算法中，如果某个参数的偏导数累积比较大，则其学习率相对较小;
如果其偏导数累积较小，则其学习率相对较大.
但整体是随着迭代次数的增加，学习率逐渐缩小.
其缺点是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.

(5) RMSprop 算法

RMSprop 算法在某些情况下避免 AdaGrad 算法中学习率不断单调下降以至于过早衰减的缺点.
在第 k 次迭代时，它首先计算每个参数梯度平方的指数衰减移动平均:

$$\begin{aligned}\text{sum}_k &:= \beta \text{sum}_{k-1} + (1-\beta) g^{(k)} \odot g^{(k)} \\ &= (1-\beta) \sum_{t=1}^{k} \beta^{k-t} g^{(t)} \odot g^{(t)}\end{aligned}$$

其中 β 为衰减率，一般取 $\beta = 0.9$，其余设置与 AdaGrad 算法相同.
从上式可以看出，RMSProp 算法和 AdaGrad算法 的区别在于 sum_k 的计算由累积方式变成了指数衰减移动平均.
在迭代过程中，每个参数的学习率既可以变小也可以变大，并不是呈单调衰减趋势.

4.4.3 梯度修正

我们可以通过修正小批量梯度下降中的梯度来加速收敛.

(1) 动量法

动量法 (momentum method) 是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加速度.
在第 k 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:

$$\Delta\theta_k := \rho \Delta\theta_{k-1} - \alpha_k g^{(k)} = -\sum_{t=1}^{k} \alpha_t \rho^{k-t} g^{(t)}$$

其中动量因子 ρ 通常设为 0.9，α_k 为第 k 轮的学习率.

- 如果 $\alpha_k \equiv \alpha$，则是经典的动量法.
- 如果 α_k 根据 RMSprop 算法生成，则称为 Adam 算法.

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.

(2) Nesterov 动量法

Nesterov 对动量法进行了改进.
动量法的迭代格式可以写为:

$$g^{(k)} := \frac{1}{\text{batch\_size}} \sum_{(x,y)\in\mathcal{S}_k} \frac{\partial \mathcal{L}(y, f(x;\theta^{(k-1)}))}{\partial \theta}$$

$$\hat{\theta} := \theta^{(k-1)} + \rho \Delta\theta^{(k-1)}$$

$$\theta^{(k)} := \hat{\theta} - \alpha_k g^{(k)}$$

其中 $\mathcal{S}_k$ 是第 k 轮选取的小批量样本集.
注意到 $g^{(k)}$ 是根据 $\theta^{(k-1)}$ 计算的，但更合理的选择是根据 $\hat{\theta}$ 计算.
因此我们可以将迭代格式调整为:

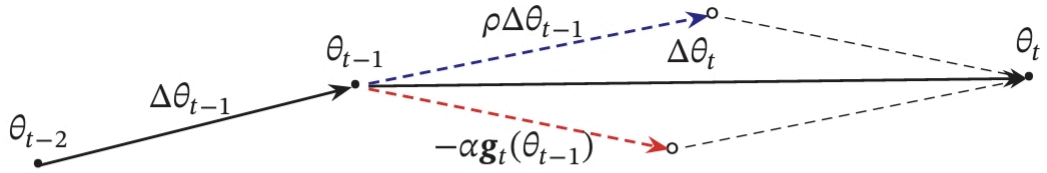
$$\hat{\theta} := \theta^{(k-1)} + \rho \Delta\theta^{(k-1)}$$

$$g^{(k)} := \frac{1}{\text{batch\_size}} \sum_{(x,y)\in\mathcal{S}_k} \frac{\partial \mathcal{L}(y, f(x;\hat{\theta}))}{\partial \theta}$$

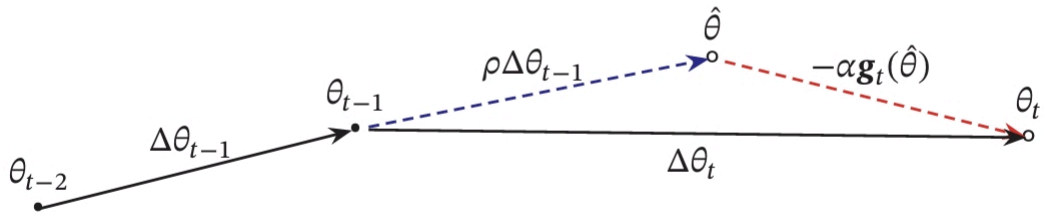
$$\theta^{(k)} := \hat{\theta} - \alpha_k g^{(k)}$$

其中 α_k 为第 k 轮的学习率.

- 如果 $\alpha_k \equiv \alpha$，则是经典的 Nesterov 动量法.
- 如果 α_k 根据 RMSprop 算法生成，则称为 Nadam 算法.



(a) 动量法



(b) Nesterov 加速梯度

图 7.6　动量法和 Nesterov 加速梯度的比较

(3) 梯度截断

在深度神经网络中，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.
为避免这种情况，当梯度的模大于一定阈值时，就对梯度进行截断，称为**梯度截断** (gradient clipping)

- ① 按值截断
  将第 k 轮的梯度 $g^{(k)}$ 的每个元素截断到区间 $[a, b]$:

$$g_{\text{clipped}}^{(k)} := \max(\min(g^{(k)}, b), a)$$

- ② 按模截断
  将第 k 轮的梯度 $g^{(k)}$ 的模截断到一个给定的阈值 b:

$$g_{\text{clipped}}^{(k)} := \begin{cases} g^{(k)}, & \text{if } \|g^{(k)}\| \leq b \\ b\frac{g^{(k)}}{\|g^{(k)}\|}, & \text{if } \|g^{(k)}\| > b \end{cases}$$

  在训练循环神经网络时，按模截断是避免梯度爆炸问题的有效方法.
  截断阈值 b 是一个超参数，也可以根据一段时间内的平均梯度来自动调整.
  训练过程对阈值 b 并不十分敏感，通常一个小的阈值就可以得到很好的效果.

4.4.4 小结

优化方法主要有两类:

- ① 调整学习率，使得优化更稳定
- ② 梯度估计修正，优化训练速度

表 7.1　神经网络常用优化方法的汇总

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

图7.8给出了这几种优化方法在 MNIST 数据集上收敛性的比较（学习率为 0.001，批量大小为 128）.

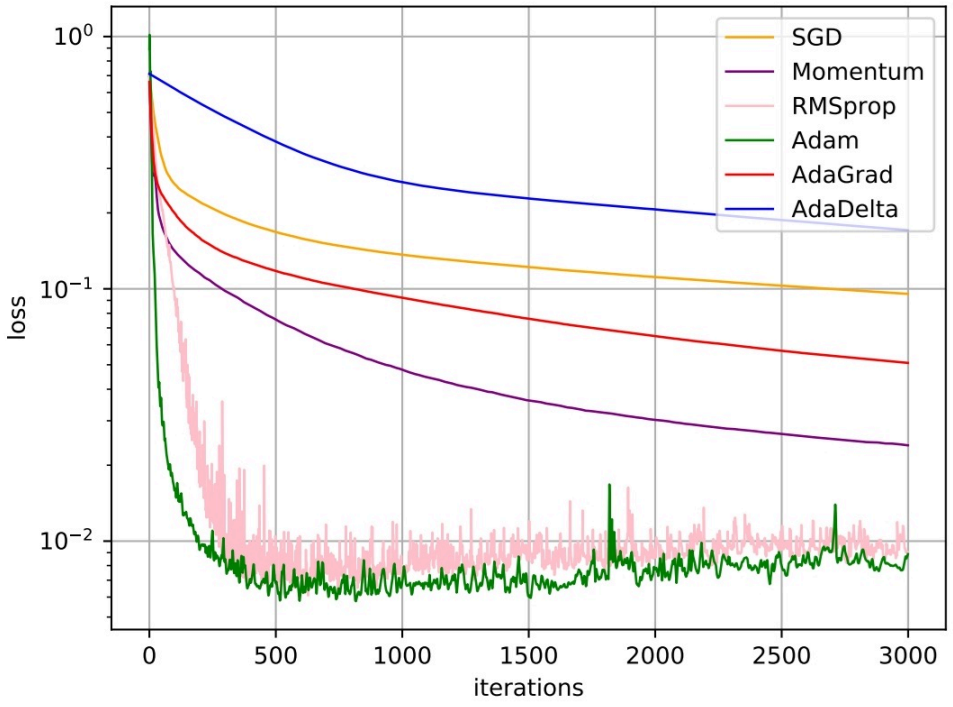


图 7.8　不同优化方法的比较

4.5 逐层归一化

逐层归一化 (layer-wise normalization) 是将传统机器学习中的数据归一化方法应用到深度神经网络中，对神经网络中隐藏层的输入进行归一化，从而使得网络更容易训练.
逐层归一化可以有效提高训练效率的原因有以下几个方面:

- **① 更好的尺度不变性:**
  在深度神经网络中，一个神经层的输入是之前神经层的输出.
  第 l 层之前的第 $1, \ldots, l-1$ 层的参数变化会导致其输入的分布发生较大的改变.
  当使用随机梯度下降来训练网络时，每次参数更新都会导致该神经层的输入分布发生改变.
  越高的层，其输入分布会改变得越明显.
  就像一栋高楼，低楼层发生一个较小的偏移，可能会导致高楼层较大的偏 移.
  从机器学习角度来看，如果一个神经层的输入分布发生了改变，
  那么其参数需要重新学习，这种现象叫作**内部协变量偏移** (internal covariate shift).
  为缓解这个问题，我们可以对每一个神经层的输入进行归一化操作，使其分布保持稳定.

  把每个神经层的输入分布都归一化为标准正态分布，
  可以使得每个神经层对其输入具有更好的尺度不变性.
  不论低层的参数如何变化，高层的输入保持相对稳定.
  另外，尺度不变性可以使得我们更加高效地进行参数初始化以及超参选择.

- **② 更平滑的优化地形:**
  逐层归一化一方面可以使得大部分神经层的输入 处于不饱和区域，从而让梯度变大，避免梯度消失问题;
  另一方面还可以使得神经网络的**优化地形** (optimization landscape) 更加平滑，
  以及使梯度变得更加稳定，从而允许我们使用更大的学习率，并提高收敛速度.

4.5.1 批量归一化

批量归一化 (Batch Normalization, BN) 可以对神经网络中任意的中间层进行归一化操作.
考虑第 l 层神经元:

$$a^{(l)} := f(z^{(l)}) = f(W^{(l)}a^{(l-1)} + b^{(l)})$$

在实践中我们通常对净输入 $z^{(l)}$ 进行归一化，即在仿射变换之后，作用激活函数之前:

$$\hat{z}^{(l)} := \alpha \odot \frac{z^{(l)} - \mathrm{E}[z^{(l)}]}{\sqrt{\mathrm{Var}(z^{(l)}) + \varepsilon}} + \beta \overset{\Delta}{=} \mathrm{BN}_{\alpha,\beta}(z^{(l)})$$

其中 α, β 为缩放和平移的参数向量，用于缓解归一化对网络表示能力的负面影响.
(例如标准归一化可能导致 Sigmoid 激活函数的 "非线性" 向 "线性" 退化)
值得一提的是，逐层归一化不但可以提高优化效率，还可以作为一种隐形的正则化方法.
在训练时，神经网络对一个样本的预测不仅和该样本自身相关，也和同一批次中的其他样本相关.
由于在选取批次时具有随机性，因此使得神经网络不会 "过拟合" 到某个特定样本，从而提高网络的泛化能力.

4.5.2 层归一化

(待补充, 等循环神经网络整理完再写)

4.6 正则化

机器学习模型的关键是泛化问题，即在样本真实分布上的期望风险最小化.
而训练数据集上的经验风险最小化和期望风险并不一致.
由于神经网络的拟合能力非常强，其在训练数据上容易过拟合.
正则化 (regularization) 是一类通过限制模型复杂度，
从而避免过拟合，提高泛化能力的方法，比如引入约束、增加先验、提前停止等.

4.6.1 l_p 正则化

引入 l_p 正则化的优化问题可表示为:

$$\theta_\star := \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{n=1}^{N} \mathrm{loss}(y^{(n)}, f(x^{(n)}; \theta)) + \lambda \|\theta\|_p$$

其中 $D_{\text{train}} := \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$ 为训练数据，$p \geq 1$ (通常取 $p = 1, 2$)
在传统的机器学习中，提高泛化能力的方法主要是限制模型复杂度，例如采用 l_1 和 l_2 正则化等方式.
而在训练深度神经网络时，特别是在过度参数化 (即模型参数量远远大于训练数据量) 时，
l_1 和 l_2 正则化的效果往往不如浅层机器学习模型好.
因此在训练深度学习模型时，往往还会使用其他的正则化方法，例如数据增强、提前停止、丢弃法、集成法等.

4.6.2 权重衰减

权重衰减 (weight decay) 在每次参数更新时引入衰减系数:

$$\theta^{(k)} \leftarrow (1-\beta)\theta^{(k-1)} + \alpha_k g^{(k)}$$

衰减系数 β 通常取极小值，例如 $\beta = 0.0005$
其中 α_k 为第 k 步的学习率，$g^{(k)}$ 为下降方向.

4.6.3 提前停止

提前停止 (early stop) 使用一个和训练集独立的样本集合，称为验证集 (validation set)
当模型在验证集上的错误率不再下降时，就停止迭代.
然而在实际操作中，验证集上的错误率变化曲线很可能是先升高再降低.
因此提前停止的具体停止标准需要视情况而定.

4.6.4 丢弃法

丢弃法 (dropout) 通过随机丢弃一部分神经元 (同时丢弃对应的连接边) 来避免过拟合.
最简单的方法是设置一个固定的概率 p，对每一个神经元都以概率 p 来判定要不要保留.
对于一个神经层 $a^{(l)} = f(Wa^{(l-1)} + b)$，我们可以引入掩蔽函数 $\mathrm{mask}(\cdot)$ 得到
$a^{(l)} = f(W\mathrm{mask}(a^{(l-1)}) + b)$
其中掩蔽函数 $\mathrm{mask}(\cdot)$ 的定义为:

$$\mathrm{mask}(x) := \begin{cases} m \odot x, & \text{if training} \\ px, & \text{if testing} \end{cases}$$

其中 $m \in \{0,1\}^{\dim(a^{(l-1)})}$ 为丢弃掩码 **(待补充)**

4.7 超参数优化

4.7.1 网格搜索

网格搜索 (grid search) 是一种通过尝试所有超参数的组合来寻址合适一组超参数配置的方法.
假设总共有 K 个超参数，第 k 个超参数的可以取 m_k 个值，那么总组合数为 $m_1 \times \cdots \times m_K$.
如果超参数是连续的，则可以按其特点将其离散化，选择几个经验值.

4.7.2 随机搜索

不同超参数对模型性能的影响有很大差异.
有些超参数 (如正则化系数) 对模型性能的影响有限，而另一些超参数 (如学习率) 对模型性能影响比较大.
在这种情况下，采用网格搜索会在不重要的超参数上进行不必要的尝试.
一种在实践中比较有效的改进方法是对超参数进行随机组合，

然后选取一个性能最好的配置，这就是**随机搜索** (random search).
随机搜索在实践中更容易实现，一般会比网格搜索更加有效.

4.7.3 动态资源分配

网格搜索和随机搜索都没有利用不同超参数组合之间的相关性，
即如果模型的超参数组合比较类似，其模型性能也是比较接近的.
因此这两种搜索方式一般都比较低效.

如果我们可以在较早的阶段就估计出一组配置的效果会比较差，
那么我们就可以中止这组配置的评估，将更多的资源留给其他配置.
这个问题可以归结为多臂赌博机问题的一个泛化问题: **最优臂问题** (best-arm problem)，
即在给定有限的机会次数下，如何玩这些赌博机并找到收益最大的臂.
和多臂赌博机问题类似，最优臂问题也是在利用和探索之间找到最佳的平衡.

由于目前神经网络的优化方法一般都采取随机梯度下降，
因此我们可以通过一组超参数的学习曲线来预估这组超参数配置是否有希望得到比较好的结果.
如果一组超参数配置的学习曲线不收敛或者收敛比较差，
我们可以应用**早停** (early-stopping) 策略来中止当前的训练.

动态资源分配的关键是将有限的资源分配给更有可能带来收益的超参数组合.
一种有效方法是逐次减半 (successive halving) 方法，将超参数优化看作一种非随机的最优臂问题.
假设要尝试 N 组超参数配置，总共可利用的资源预算 (摇臂的次数) 为 B，
我们可以通过 $T = \lceil \log_2(N) \rceil - 1$ 轮逐次减半的方法来选取最优的配置:

算法 7.2　一种逐次减半的动态资源分配方法

输入: 预算 B，N 个超参数配置 $\{\boldsymbol{x}_n\}_{n=1}^N$

$T \leftarrow \lceil \log_2(N) \rceil - 1$;
随机初始化 $\mathcal{S}_0 = \{\boldsymbol{x}_n\}_{n=1}^N$;
for $t \leftarrow 1$ **to** T **do**
　$r_t \leftarrow \lfloor \frac{B}{|\mathcal{S}_t| \times T} \rfloor$;
　给 S_t 中的每组配置分配 r_t 的资源;
　运行 S_t 所有配置，评估结果为 $\boldsymbol{y}_t$;
　根据评估结果，选取 $|S_t|/2$ 组最优的配置 $\mathcal{S}_t \leftarrow \arg\max(\mathcal{S}_t, \boldsymbol{y}_t, |\mathcal{S}_t|/2)$;
　// $\arg\max(\mathcal{S}, \boldsymbol{y}, m)$ 为从集合 $\mathcal{S}$ 中选取 m 个元素，对应最优的 m 个评估结果.
end

输出: 最优配置 $\mathcal{S}_K$

在逐次减半方法中，尝试的超参数配置数量 N 十分关键.
如果 N 越大，得到最佳配置的机会也越大，但每组配置分到的资源就越少，这样早期的评估结果可能不准确.
反之如果 N 越小，每组超参数配置的评估会越准确，但有可能无法得到最优的配置.
因此如何设置 N 是平衡 "利用-探索" 的一个关键因素.
一种改进的方法是 HyperBand 方法，通过尝试不同的 N 来选取最优参数.

4.7.4 Bayes 优化

Bayes 优化 (Bayesian optimization) 是一种自适应的超参数优化方法，
根据当前已经试验的超参数组合，来预测下一个可能带来最大收益的组合.
一种比较常用的 Bayes 优化方法为**时序模型优化** (Sequential Model-Based Optimization, SMBO)
(待补充, 等循环神经网络整理完再写)

算法 7.1　时序模型优化（SMBO）方法

输入: 优化目标函数 $f(\boldsymbol{x})$, 迭代次数 T, 收益函数 $a(x, \mathcal{H})$

$\mathcal{H} \leftarrow \varnothing$;
随机初始化高斯过程, 并计算 $p_{\mathcal{GP}}(f(\boldsymbol{x})|\boldsymbol{x}, \mathcal{H})$;
for $t \leftarrow 1$ **to** T **do**
    $\boldsymbol{x}' \leftarrow \arg\max_x a(x, \mathcal{H})$;
    评价 $y' = f(\boldsymbol{x}')$;　// 代价高
    $\mathcal{H} \leftarrow \mathcal{H} \cup (\boldsymbol{x}', y')$;
    根据 $\mathcal{H}$ 重新建模高斯过程, 并计算 $p_{\mathcal{GP}}(f(\boldsymbol{x})|\boldsymbol{x}, \mathcal{H})$;
end

输出: $\mathcal{H}$

The End