网络优化与正则化

优化问题：神经网络模型是一个非凸函数，再加上在深度网络中的梯度消失问题，很难进行优化；另外，深层神经网络模型一般参数比较多，训练数据也比较大，会导致训练的效率比较低。

泛化问题：因为神经网络的拟合能力强，反而容易在训练集上产生过拟合。因此，在训练深层神经网络时，同时也需要通过一定的正则化方法来改进网络的泛化能力。

网络优化

深层神经网络是一个高度非线性的模型，其风险函数是一个非凸函数，因此风险最小化是一个非凸优化问题，会存在很多局部最优点。

网络优化的难点

网络结构多样性

神经网络的种类非常多，不同参数在网络中的作用也有很大的差异，网络的超参数一般也比较多。

网络结构的多样性，我们很难找到一种通用的优化方法。不同的优化方法在不同网络结构上的差异也都比较大。

高维变量的非凸优化

低维空间的非凸优化问题主要是存在一些局部最优点。基于梯度下降的优化方法会陷入局部最优点，因此低维空间非凸优化的主要难点是如何选择初始化参数和逃离局部最优点。深层神经网络的参数非常多，其参数学习是在非常高维空间中的非凸优化问题，其挑战和在低维空间的非凸优化问题有所不同。

鞍点 在高维空间中，非凸优化的难点并不在于如何逃离局部最优点，而是如何逃离鞍点（Saddle Point）。鞍点的梯度是 0，但是在一些维度上是最高点，在另一些维度上是最低点。基于梯度下降的优化方法会在鞍点附近接近于停滞，同样很难从这些鞍点中逃离。

平坦底部 深层神经网络的参数非常多，并且有一定的冗余性，这导致每单个参数对最终损失的影响都比较小，这导致了损失函数在局部最优点附近是一个平坦的区域，称为平坦最小值（Flat Minima）。并且在非常大的神经网络中，大部分的局部最小值是相等的。虽然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，网络陷入局部最小值的概率大大降低。

优化算法

梯度下降法可以分为：批量梯度下降、随机梯度下降以及小批量梯度下降。

小批量梯度下降

令 $f(\mathbf{x},\theta)$ 表示一个深层神经网络，θ 为网络参数，在使用小批量梯度下降进行优化时，每次选取 K 个训练样本 $\mathcal{I}_t = \{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})\}_{k=1}^{K}$。第 t 次迭代（iteration）

时损失函数关于参数θ的偏导数为

$$\mathcal{g}_t(\theta) = \frac{1}{K} \sum_{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) \in \mathcal{I}_t} \frac{\partial \mathcal{L}(\mathbf{y}^{(k)}, f(\mathbf{x}^{(k)}, \theta))}{\partial \theta}, \tag{7.1}$$

其中$\mathcal{L}(\cdot)$为可微分的损失函数，K称为批量大小（Batch Size）。

第t次更新的梯度$\mathbf{g}_t$定义为

$$\mathbf{g}_t \triangleq \mathcal{g}_t(\theta_{t-1}). \tag{7.2}$$

使用梯度下降来更新参数，

$$\theta_t \leftarrow \theta_{t-1} - \alpha \mathbf{g}_t, \tag{7.3}$$

其中$\alpha > 0$为学习率。

每次迭代时参数更新的差值$\Delta\theta_t$定义为

$$\Delta\theta_t \triangleq \theta_t - \theta_{t-1}. \tag{7.4}$$

$\Delta\theta_t$和梯度$\mathbf{g}_t$并不需要完全一致。$\Delta\theta_t$为每次迭代时参数的实际更新方向，即$\theta_t = \theta_{t-1} + \Delta\theta_t$。在标准的小批量梯度下降中，$\Delta\theta_t = -\alpha \mathbf{g}_t$。

每次迭代选取的批量样本数越多，下降效果越明显，并且下降曲线越平滑。当每次选取一个样本时（相当于随机梯度下降），损失整体是下降趋势，但局部看会来回震荡。如果按整个数据集上的迭代次数（Epoch）的来看损失变化情况，则是批量样本数越小，下降效果越明显。

为了更有效地进行训练深层神经网络，在标准的小批量梯度下降方法的基础上，也经常使用一些改进方法以加快优化速度：学习率衰减和梯度方向优化。这些改进的优化方法也同样可以应用在批量或随机梯度下降方法上。

学习率衰减

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震荡。

逆时衰减：$\alpha_t = \alpha_0 \frac{1}{1 + \beta \times t}$

指数衰减：$\alpha_t = \alpha_0 \beta^t$

自然指数衰减：$\alpha_t = \alpha_0 \exp(-\beta \times t)$

AdaGrad（Adaptive Gradient）算法

借鉴 L2 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 迭代时，先计算每个参数梯度平方的累计值

$$G_t = \sum_{\tau=1}^{t} \mathbf{g}_\tau \odot \mathbf{g}_\tau, \tag{7.8}$$

AdaGrad 算法的参数更新差值为

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t, \tag{7.9}$$

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。同时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。

RMSprop 算法

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

$$G_t = \beta G_{t-1} + (1-\beta)\mathbf{g}_t \odot \mathbf{g}_t \tag{7.10}$$

$$= (1-\beta)\sum_{\tau=1}^{t}\beta^{t-\tau}\mathbf{g}_\tau \odot \mathbf{g}_\tau, \tag{7.11}$$

RMSprop 算法的参数更新差值为

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t+\epsilon}} \odot \mathbf{g}_t, \tag{7.12}$$

AdaDelta 算法

通过梯度平方的指数衰减移动平均来调整学习率，还引入了每次参数更新差 Δθ 的平方的指数衰减权移动平均。

第 t 次迭代时，每次参数更新差 $\Delta\theta_\tau, 1 \le \tau \le t-1$ 的指数衰减权移动平均为

$$\Delta X_{t-1}^2 = \beta_1 \Delta X_{t-2}^2 + (1-\beta_1)\Delta\theta_{t-1} \odot \Delta\theta_{t-1}. \tag{7.13}$$

AdaDelta 算法的参数更新差值为

$$\Delta\theta_t = -\frac{\sqrt{\Delta X_{t-1}^2+\epsilon}}{\sqrt{G_t+\epsilon}}\mathbf{g}_t \tag{7.14}$$

梯度方向优化

使用最近一段时间内的平均梯度来代替当前时刻的梯度来作为参数更新的方向。用梯度的移动平均来代替每次的实际梯度，并提高优化速度，这就是动量法。

动量法（Momentum Method）

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，

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathbf{g}_t, \tag{7.15}$$

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；相反，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，梯度方法都比较一致，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，梯度方法会取决不一致，在收敛值附近震荡，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。

Nesterov 加速梯度（Nesterov Accelerated Gradient， NAG）

$$\hat{\theta} = \theta_{t-1} + \rho\Delta\theta_{t-1}, \tag{7.16}$$

$$\theta_t = \hat{\theta} - \alpha\mathbf{g}_t, \tag{7.17}$$

其中梯度 $\mathbf{g}_t$ 为点 θ_{t-1} 上的梯度，因此在第二步更新中有些不太合理。更合理的更新方向应该为 $\hat{\theta}$ 上的梯度。

这样，合并后的更新方向为

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathbf{g}_t(\theta_{t-1} + \rho\Delta\theta_{t-1}), \tag{7.18}$$

其中 $\mathbf{g}_t(\theta_{t-1} + \rho\Delta\theta_{t-1})$ 表示损失函数在点 $\hat{\theta} = \theta_{t-1} + \rho\Delta\theta_{t-1}$ 上的偏导数。

自适应动量估计（Adaptive Moment Estimation， Adam）

动量法和 RMSprop 的结合，不但使用动量作为参数更新方向，而且可以自适应调整学习率。

Adam 算法一方面计算梯度平方 $\mathbf{g}_t^2$ 的指数加权平均（和 RMSprop 类似），另一方面计算梯度 $\mathbf{g}_t$ 的指数加权平均（和动量法类似）。

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)\mathbf{g}_t, \tag{7.19}$$

$$G_t = \beta_2 G_{t-1} + (1-\beta_2)\mathbf{g}_t \odot \mathbf{g}_t, \tag{7.20}$$

M_t可以看作是梯度的均值（一阶矩），G_t可以看作是梯度的未减去均值的方差（二阶矩）。

Adam 算法的参数更新差值为

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{\hat{G}_t + \epsilon}}\hat{M}_t, \tag{7.23}$$

梯度截断

在基于梯度下降的优化过程中，如果梯度突然增大，用大的梯度进行更新参数，反而会导致其远离最优点。为了在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）。

按值截断：在第 t 次迭代时，梯度为g_t，给定一个区间[a,b]，如果一个参数的梯度小于 a 时，就将其设为 a；如果大于 b 时，就将其设为 b。

$$g_t = \max(\min(g_t, b), a)$$

按模截断:按模截断是将梯度的模截断到一个给定的截断阈值 b。

如果 $\|\mathbf{g}_t\|^2 \leq b$，保持 $\mathbf{g}_t$ 不变。如果 $\|\mathbf{g}_t\|^2 > b$，令

$$\mathbf{g}_t = \frac{b}{\|\mathbf{g}_t\|}\mathbf{g}_t. \tag{7.25}$$

优化算法小结

一是调整学习率，使得优化更稳定；二是调整梯度方向，优化训练速度。

学习率衰减	梯度方向优化
AdaGrad、RMSprop、AdaDelta	动量法、Nesterov 加速梯度、梯度截断

Adam≈动量法 +RMSprop

参数初始化

所有的隐层神经元的激活值都相同，会导致深层神经元没有区分性。这种现象也称为对称权重现象。如果一个神经元的输入连接很多，它的每个输入连接上的权重就应该小一些，以避免神经元的输出过大（当激活函数为 ReLU 时）或过饱和（当激活函数为 sigmoid 函数时）。

Gaussian 分布初始化

初始化一个深度网络时，一个比较好的初始化方案是保持每个神经元输入的方差为一个常量。当一个神经元的输入连接数量为n_{in}时，可以设置其输入连接权

重以$N(0,\sqrt{\frac{1}{n_{in}}})$的 Gaussian 分布进行初始化。如果同时考虑输出连接的数量n_{out}，则可以按$N(0,\sqrt{\frac{2}{n_{in}+n_{out}}})$的 Gaussian 分布进行初始化。

均匀分布初始化

在一个给定的区间[-r,r]内采用均匀分布来初始化参数。超参数 r 的设置也可以按神经元的连接数量进行自适应的调整。

Xavier 初始化方法:一个自动计算超参数 r 的方法，参数可以在[-r,r]内采用均匀分布进行初始化。

如果神经元激活函数为 logistic 函数，对于第 l-1 到 l 层的权重参数区间 $r=\sqrt{\frac{6}{n^{l-1}+n^{l}}}$；若是 tanh 函数，$r=4\sqrt{\frac{6}{n^{l-1}+n^{l}}}$。

假设第 l 层的一个隐藏层神经元 z^l，其接受前一层的 n^{l-1} 个神经元的输出 $a_i^{(l-1)}$，$i\in[1,n^{(l-1)}]$，

$$z^l=\sum_{i=1}^{n^{(l-1)}}w_i^l a_i^{(l-1)}. \tag{7.28}$$

为了避免初始化参数使得激活值变得饱和，我们需要尽量使得 z^l 处于激活函数的线性区间，也就是其绝对值比较小的值。这时该神经元的激活值为 $a^l=f(z^l)\approx z^l$。

假设 w_i^l 和 $a_i^{(l-1)}$ 都是相互独立，并且均值都为0，则 a^l 的均值为

$$\mathbb{E}[a^l]=\mathbb{E}[\sum_{i=1}^{n^{(l-1)}}w_i^l a_i^{(l-1)}]=\sum_{i=1}^{n^{(l-1)}}\mathbb{E}[\mathbf{w}_i]\mathbb{E}[a_i^{(l-1)}]=0. \tag{7.29}$$

a^l 的方差为

$$\begin{aligned}\mathrm{var}[a^l]&=\mathrm{var}[\sum_{i=1}^{n^{(l-1)}}w_i^l a_i^{(l-1)}] &(7.30)\\ &=\sum_{i=1}^{n^{(l-1)}}\mathrm{var}[w_i^l]\,\mathrm{var}[a_i^{(l-1)}] &(7.31)\\ &=n^{(l-1)}\,\mathrm{var}[w_i^l]\,\mathrm{var}[a_i^{(l-1)}]. &(7.32)\end{aligned}$$

也就是说，输入信号的方差在经过该神经元后被放大或缩小了 $n^{(l-1)}\,\mathrm{var}[w_i^l]$ 倍。为了使得在经过多层网络后，信号不被过分放大或过分减弱，我们尽可能保持每个神经元的输入和输出的方差一致。这样 $n^{(l-1)}\,\mathrm{var}[w_i^l]$ 设为1比较合理，即

$$\mathrm{var}[w_i^l]=\frac{1}{n^{(l-1)}}. \tag{7.33}$$

同理，为了使得在反向传播中，误差信号也不被放大或缩小，需要将 w_i^l 的方差保持为

$$\mathrm{var}[w_i^l]=\frac{1}{n^{(l)}}. \tag{7.34}$$

作为折中，同时考虑信号在前向和反向传播中都不被放大或缩小，可以设置

$$\mathrm{var}[w_i^l]=\frac{2}{n^{(l-1)}+n^{(l)}}. \tag{7.35}$$

假设随机变量 x 在区间 $[a,b]$ 内均匀分布，则其方差为：

$$\text{var}[x] = \frac{(b-a)^2}{12}. \tag{7.36}$$

因此，若让 $w_i^l \in [-r,r]$，并且 $\text{var}[w_i^l] = 1$，则 r 的取值为

$$r = \sqrt{\frac{6}{n^{l-1}+n^1}}. \tag{7.37}$$

数据预处理

对于基于相似度比较的机器学习方法，必须先对样本进行预处理，将各个维度的特征归一化到同一个取值区间，并且消除不同特征之间的相关性。虽然神经网络可以通过参数的调整来适应不同特征的取值范围，但是会导致训练效率比较低。

缩放归一化 缩放归一化是一种非常简单的归一化方法，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。对于每一维特征 x，

$$\hat{x}^{(i)} = \frac{x^{(i)} - \min_i(x^{(i)})}{\max_i(x^{(i)}) - \min_i(x^{(i)})}, \tag{7.38}$$

其中 $\min(x)$ 和 $\max(x)$ 分别是特征 x 在所有样本上的最小值和最大值。

标准归一化 标准归一化也叫z-score归一化，来源于统计上的标准分数。将每一个维特征都处理为符合标准正态分布（均值为0，标准差为1）。假设有 N 个样本 $\{\mathbf{x}^{(i)}\}, i = 1, \cdots, N$，对于每一维特征 x，我们先计算它的均值和标准差：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}, \tag{7.39}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)} - \mu)^2. \tag{7.40}$$

然后，将特征 $x^{(i)}$ 减去均值，并除以标准差，得到新的特征值 $\hat{x}^{(i)}$。

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu}{\sigma}, \tag{7.41}$$

这里 σ 不能为0。如果标准差为0，说明这一维特征没有任务区分性，可以直接删掉。

在标准归一化之后，每一维特征都服从标准正态分布。

白化（Whitening）：一种重要的预处理方法，用来降低输入数据特征之间的冗余性。输入数据经过白化处理后，特征之间相关性较低，并且所有特征具有相同的方差。主成分分析（Principal Component Analysis，PCA）方法去除掉各个成分之间的相关性。

逐层归一化

如果某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，这种现象叫做内部协变量偏移（Internal Covariate Shift）。

批量归一化（Batch Normalization， BN）

逐层归一化方法，可以对神经网络中任意的中间层进行归一化操作。

为了使得归一化不对网络的表示能力造成负面影响，可以通过一个附加的缩放和平移变换改变取值区间。

$$\hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \odot \gamma + \beta \tag{7.46}$$

$$\triangleq \mathrm{BN}_{\gamma,\beta}(\mathbf{z}^{(l)}), \tag{7.47}$$

批量归一化操作可以看作是一个特殊的神经层，加在每一层非线性激活函数之前，即

$$\mathbf{a}^{(l)} = f(\mathrm{BN}_{\gamma,\beta}(\mathbf{z}^{(l)})) = f\left(\mathrm{BN}_{\gamma,\beta}(W\mathbf{a}^{(l-1)})\right), \tag{7.48}$$

当训练完成时，用整个数据集上的均值 μ 和方差 σ 来分别代替每次小批量样本的μ_B和方差$\sigma_B{}^2$。在实践中，μ_B和$\sigma_B{}^2$也可以用移动平均来计算。

层归一化（Layer Normalization）

对一个中间层的所有神经元进行归一化。

$$\hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)2} + \epsilon}} \odot \gamma + \beta \tag{7.51}$$

$$\triangleq \mathrm{LN}_{\gamma,\beta}(\mathbf{z}^{(l)}), \tag{7.52}$$

循环神经网络中的层归一化 层归一化可以应用在循环神经网络中，对循环神经层进行归一化操作。假设在时刻 t，循环神经网络的隐藏层为h_t，其层归一化的更新为

$$\mathbf{z}_t = U\mathbf{h}_{t-1} + W\mathbf{x}_t, \tag{7.53}$$

$$\mathbf{h}_t = f(\mathrm{LN}_{\gamma,\beta}(\mathbf{z}_t)), \tag{7.54}$$

其它归一化方法

权重归一化(Weight Normalization):对神经网络的连接权重进行归一化，通过再参数化（Reparameterization）方法，将连接权重分解为长度和方向两种参数。假设第 l 层神经元$\mathrm{a}^{(l)} = f(Wa^{(l-1)} + b)$，我们将 W 再参数化为

$$W_{i,:} = \frac{g_i}{\|\mathbf{v}_i\|}\mathbf{v}_i, \qquad 1 \le i \le n^l \tag{7.55}$$

由于在神经网络中权重经常是共享的，权重数量往往比神经元数量要少，因此权重归一化的开销会比较小。

局部响应归一化（Local Response Normalization， LRN）:对邻近的特征映射进行局部归一化。

$$\hat{Y}^p = Y^p \Big/ \left(k + \alpha \sum_{j=\max(1,p-\frac{r}{2})}^{\min(P,p+\frac{r}{2})} (Y^j)^2\right)^{\beta} \tag{7.56}$$

$$\triangleq \mathrm{LRN}_{n,k,\alpha,\beta}(Y^p), \tag{7.57}$$

局部响应归一化和层归一化都是对同层的神经元进行归一化。不同的是局部响应归一化应用在激活函数之后，只是对邻近的神经元进行局部归一化，并且不减去均值。最大汇聚是对同一个特征映射中的邻近位置中的神经元进行抑制，而

局部响应归一化是对同一个位置的邻近特征映射中的神经元进行抑制。

超参数优化

- 网络结构，包括神经元之间的连接关系、层数、每层的神经元数量、激活函数的类型等；
- 优化参数，包括优化方法、学习率、小批量的样本数量等；
- 正则化系数。

网格搜索（grid search）

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。

随机搜索（Random Search）

贝叶斯优化（Bayesian optimization）

自适应的超参数搜索方法，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。

假设超参数优化的函数 f(x)服从高斯过程，则 p(f(x)|x)为一个正态分布。贝叶斯优化过程是根据已有的 N 组试验结果$H=\{x_n,y_n\}_{n=1}^N$(y_n为$f(x_n)$的观测值)来建模高斯过程，并计算 f(x)的后验分布$p_{gp}(f(x)|x,H)$。

为了使得$p_{\mathcal{GP}}(f(\mathbf{x})|\mathbf{x},\mathcal{H})$接近其真实分布，就需要对样本空间进行足够多的采样。但是超参数优化中每一个样本的生成成本很高，需要用尽可能少的样本来使得$p_\theta(f(\mathbf{x})|\mathbf{x},\mathcal{H})$接近于真实分布。因此，需要通过定义一个收益函数（acquisition function）$a(x,\mathcal{H})$来判断一个样本是否能够给建模$p_\theta(f(\mathbf{x})|\mathbf{x},\mathcal{H})$提供更多的收益。收益越大，其修正的高斯过程会越接近目标函数的真实分布。

算法 7.1: 时序模型优化，一种贝叶斯优化方法

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

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

输出: $\mathcal{H}$

高斯过程建模需要计算协方差矩阵的逆，时间复杂度是$O(n^3)$。

动态资源分配

如果一组超参数配置的学习曲线不收敛或者收敛比较差，我们可以应用早期停止（early-stopping）策略来中止当前的训练。

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 $\{\mathbf{x}_n\}_{n=1}^N$

$T \leftarrow \lceil \log_2(N) \rceil - 1$;

随机初始化 $\mathcal{S}_0 = \{\mathbf{x}_n\}_{n=1}^N$;

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 $\mathbf{y}_t$;

  根据评估结果，选取 $|S_t|/2$ 组最优的配置

  $\mathcal{S}_t \leftarrow \arg\max(\mathcal{S}_t, \mathbf{y}_t, |\mathcal{S}_t|/2)$; // $\arg\max(\mathcal{S}, \mathbf{y}, m)$ 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 $\mathcal{S}_K$

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。

神经架构搜索（neural architecture search， NAS）:通过神经网络来自动实现网络架构的设计。利用元学习的思想，神经架构搜索利用一个控制器来生成另一个子网络的架构描述。控制器可以由一个循环神经网络来实现。控制器的训练可以通过强化学习来完成，其奖励信号为生成的子网络在开发集上的准确率。

网络正则化

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，包括引入一些约束规则，增加先验、提前停止等。

L1 和 L2 正则化

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(y^{(n)}, f(\mathbf{x}^{(n)}, \theta)) + \lambda \ell_p(\theta), \tag{7.59}$$

带正则化的优化问题等价于下面带约束条件的优化问题，

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(y^{(n)}, f(\mathbf{x}^{(n)}, \theta)), \tag{7.60}$$

$$\text{subject to } \ell_p(\theta) \leq 1. \tag{7.61}$$

弹性网络正则化（Elastic Net Regularization），同时加入 L1 和 L2 正则化。

权重衰减（Weight Decay）

在每次参数更新时，引入一个衰减系数： $\theta_t \leftarrow (1-w)\theta_{t-1} - \alpha \mathbf{g}_t$

在标准的随机梯度下降中，权重衰减正则化和 L2 正则化的效果相同。但是，在较为复杂的优化方法（比如 Adam）中，权重衰减和 L2 正则化并不等价。

提前停止（early stop）

丢弃法（Dropout Method）

对每一个神经元都一个概率 p 来判定要不要保留。对于一个神经层$y = f(Wx + b)$，我们可以引入一个丢弃函数 d(•)使得$y = f(Wd(x) + b)$。

对于输入层的神经元，其丢弃率通常设为更接近 1 的数，使得输入变化不会太大。对输入层神经元进行丢弃时，相当于给数据增加噪声，以此来提高网络的鲁棒性。

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,对非时间维度的连接（即非循环连接）进行随机丢失。

数据增强（Data Augmentation）

- 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。