



机器学习与数据挖掘

神经网络的训练与优化



课程大纲

- ◆ 神经网络训练的总体框架
- ◆ 随机梯度下降算法

随机梯度下降 (SGD)

- 假设 \mathbf{w} 是要优化的参数。随机梯度下降 (SGD) 中的更新方式是：

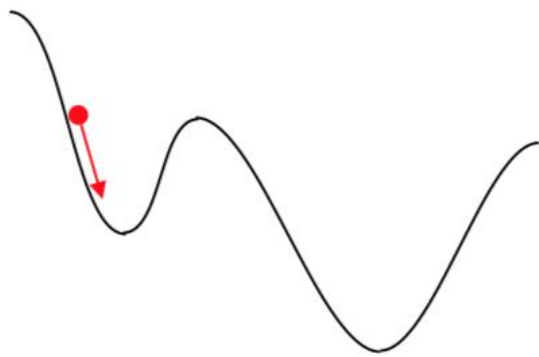
$$\mathbf{w}_{t+1} = \mathbf{w}_t - lr * \nabla f(\mathbf{w}_t)$$

- $\nabla f(\mathbf{w})$ 是损失 \mathcal{L} 的随机梯度，仅使用部分训练数据进行估计
- lr 是学习率

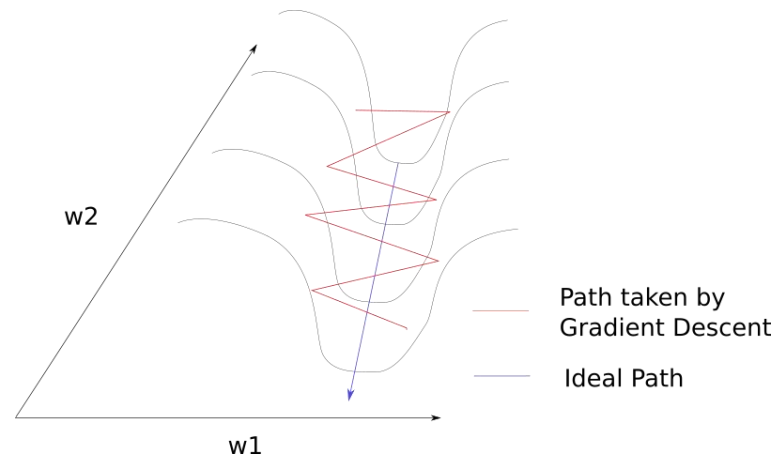
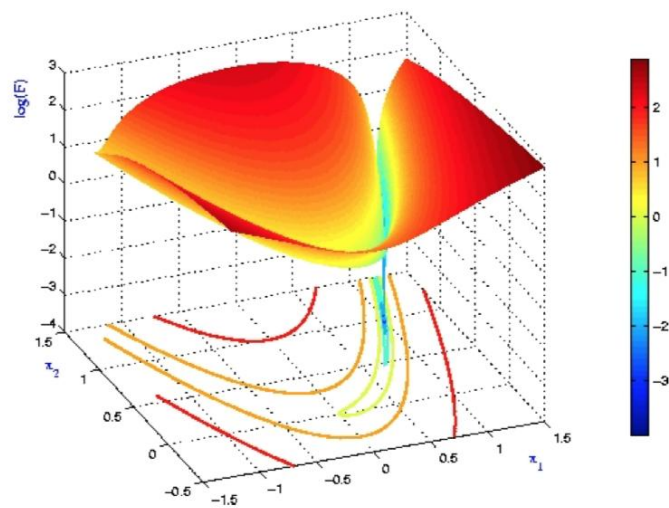


想象一步步地沿着山坡行走以降低高度，最终走到山脉的最低处

- 随机梯度下降存在的问题
 - 非常容易陷入局部最优



- 由于神经网络（NNs）中存在病态曲率，导致收敛缓慢

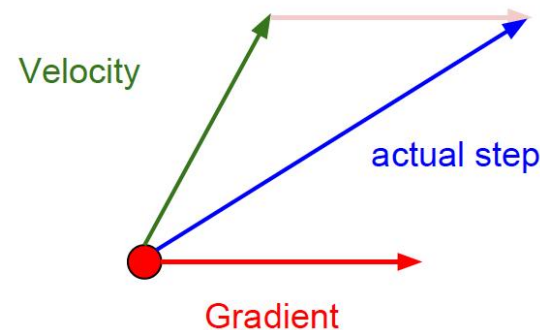


SGD + 动量

- 更新方程

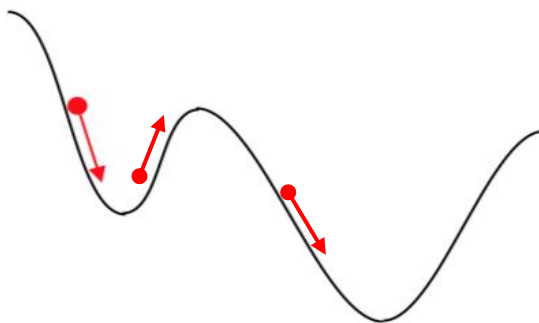
$$\mathbf{v}_t = \rho \mathbf{v}_{t-1} + \nabla f(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - lr \cdot \mathbf{v}_t$$



其中 $\rho \in (0, 1)$ 是衰减率，通常选择为 0.9, 0.95, 0.99

这个更新过程可以理解为一个小球从崎岖的山坡上滚下，其摩擦力与 ρ 成正比



- 更新方程

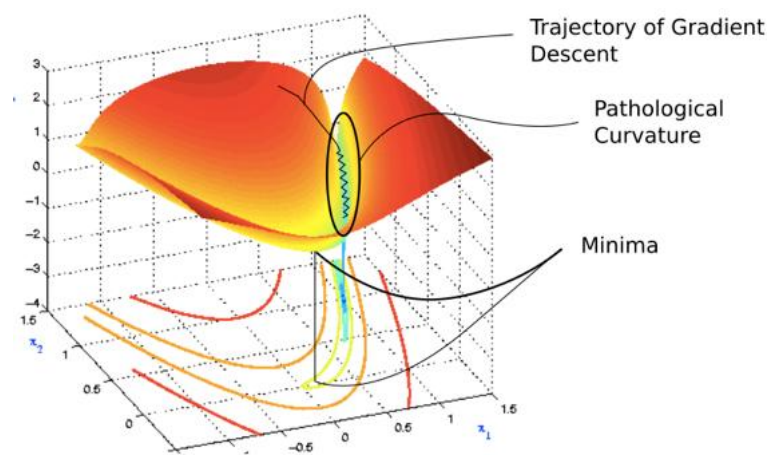
滑动平均

$$\mathbf{s}_t = \rho \cdot \mathbf{s}_{t-1} + (1 - \rho)(\nabla f(\mathbf{w}_t))^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - lr \cdot \nabla f(\mathbf{w}_t) \oslash \sqrt{\mathbf{s}_t}$$

其中 \mathbf{s}_t 是截至当前时间步的平方梯度加权平均； \oslash 表示逐元素除法

它 *将不同维度的梯度重新缩放到相同的水平*，从而缓解了病态曲率问题



补充：滑动平均

$$\mathbf{s}_1 = \rho \cdot \mathbf{s}_0 + (1 - \rho)(\nabla f(\mathbf{w}_1))^2$$

$$\mathbf{s}_2 = \rho^2 \cdot \mathbf{s}_0 + \rho(1 - \rho)(\nabla f(\mathbf{w}_1))^2 + (1 - \rho)(\nabla f(\mathbf{w}_2))^2$$

$$\mathbf{s}_3 = \rho^3 \cdot \mathbf{s}_0 + \rho^2(1 - \rho)(\nabla f(\mathbf{w}_1))^2 + \rho(1 - \rho)(\nabla f(\mathbf{w}_2))^2 + (1 - \rho)(\nabla f(\mathbf{w}_3))^2$$

\vdots

对于较早的记录，其权重会衰减

可以验证，所有权重之和为 1，即：

$$(1 - \rho) + \rho(1 - \rho) + \rho^2(1 - \rho) + \rho^3(1 - \rho) + \dots$$

$$= \frac{1 - \rho^n}{1 - \rho} (1 - \rho) \rightarrow 1$$

加权

Adam

- 更新方程

$$\mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \nabla f(\mathbf{w}_t)$$

$$\mathbf{s}_t = \beta_2 \cdot \mathbf{s}_{t-1} + (1 - \beta_2) (\nabla f(\mathbf{w}_t))^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - lr \cdot \mathbf{m}_t \oslash \sqrt{\mathbf{s}_t}$$

与 RMSProp 相同，
除了 \mathbf{m}_t

其中 \mathbf{m}_t 是过去梯度的滑动平均

它是 RMSProp 的一个改进版本，*用梯度的滑动平均 \mathbf{m}_t 替代了精确梯度 $\nabla f(\mathbf{w}_t)$*

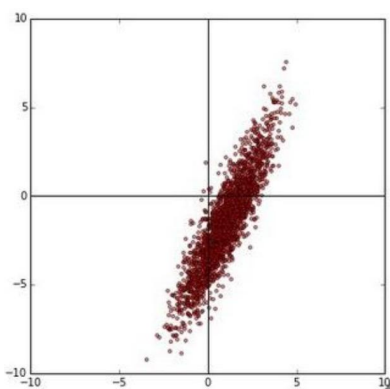
- 默认设置： $\beta_1 = 0.9$, $\beta_2 = 0.999$ 和 $lr = 10^{-3}$
- 通过将 $\beta_1 = 0$ ，Adam 退化为 RMSProp

课程大纲

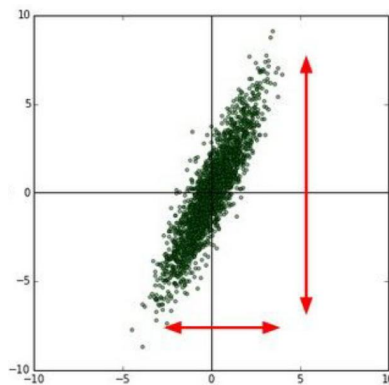
- 神经网络训练的总体框架
- 随机梯度下降算法

数据预处理

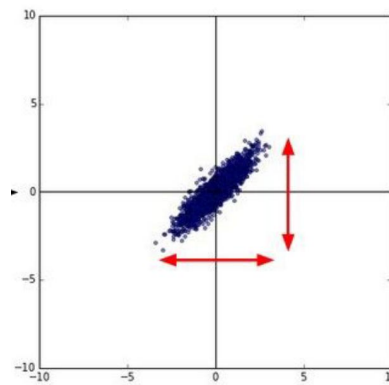
- 常用的数据预处理方法：
 - 1) 数据**中心化**，即减去数据的均值
 - 2) **方差标准化**，即对每个维度进行方差归一化
 - 3) 数据**白化**



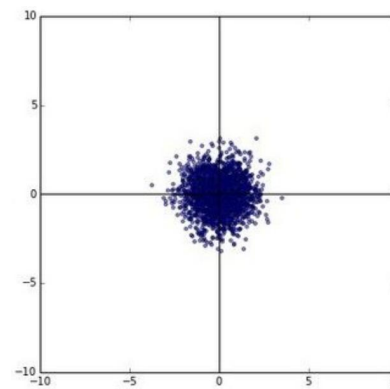
原始



中心化后



方差归一化后



白化后

- 根据数据的特性，我们可以使用上述一种或几种方法来预处理数据

参数初始化

1) 通过从固定的高斯分布中随机抽样来初始化所有层的权重，例如：

$$\mathbf{W} \sim \mathcal{N}(0, 0.01^2)$$

- 这种方法对于不那么深的神经网络效果不错（例如，层数小于 8）
- 如果使用 ReLU 作为激活函数，有些人也建议使用截断高斯分布，即：

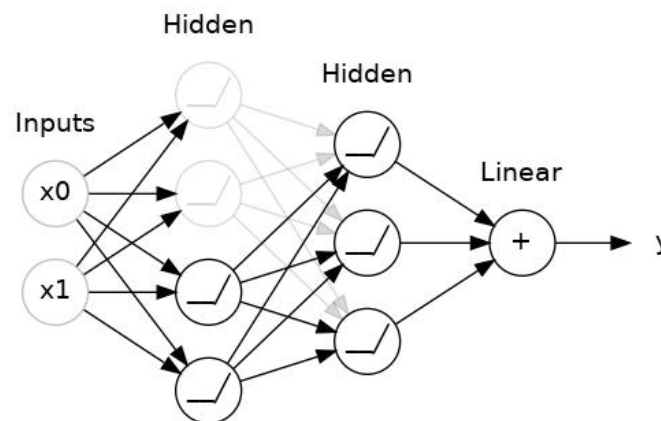
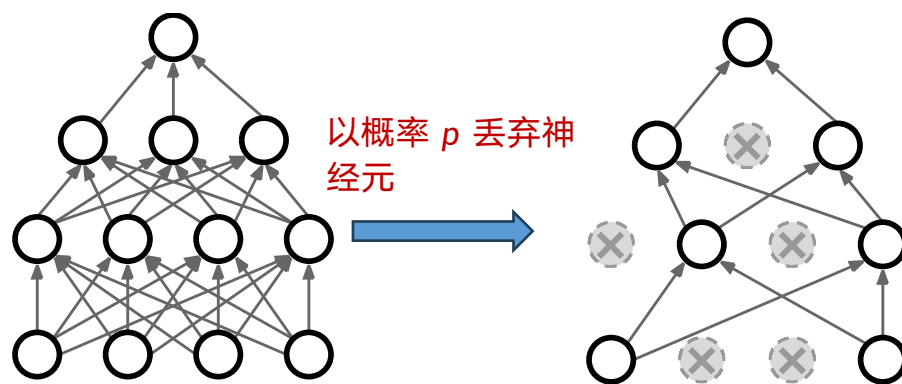
$$\mathbf{W} \sim \mathcal{N}_T(0, 0.01^2)$$

2) 当网络变得更深时，可以通过一个高斯分布 $\mathbf{W} \sim \mathcal{N}(0, \sigma_i^2)$ 进行初始化，其中每层的方差与前一层神经元的数量成反比，即：

$$\sigma_i = \frac{1}{\sqrt{\# \text{ neurons in } (i-1)\text{th layer}}}$$

Dropout

- 在每次前向传播中，随机将一些神经元设置为零
 - 丢弃神经元的概率是一个超参数；0.5 是一个常用的值

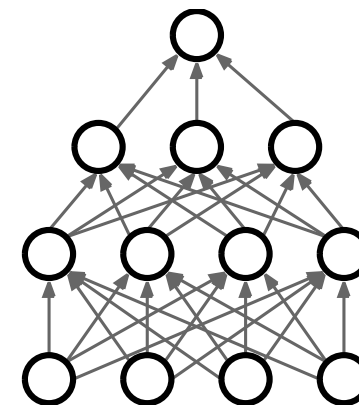


Dropout 可以有效缓解深度神经网络中常见的过拟合问题

- 测试时的 Dropout

- 在测试期间，我们倾向于让模型对一个输入输出一个固定的值

- 因此，在测试时应该移除神经元上的随机性，将它们全部打开



- 为了保持每个神经元在测试时的输入与训练时一致，我们需要用丢弃概率 p 来重新缩放这个值

$$\begin{array}{ccc} h_{\ell+1} = \sigma(W_{\ell} h_{\ell-1}) & \longrightarrow & h_{\ell+1} = \sigma(p \cdot W_{\ell} h_{\ell-1}) \\ \text{w/o dropout} & & \text{w/ dropout} \end{array}$$

批量归一化

- 虽然好的初始化可以让网络在一开始处于正常状态，但其影响会随着训练的进行而减弱
- 一种更有效的方法是，在训练过程中有目的地将隐藏状态 $\{\tilde{\mathbf{h}}_\ell^{(i)}\}_{i=1}^N$ 归一化到标准正态分布，如下所示：

$$\hat{\mathbf{h}}_\ell^{(i)} = (\tilde{\mathbf{h}}_\ell^{(i)} - \boldsymbol{\mu}_\ell) \oslash \boldsymbol{\sigma}_\ell$$

其中

$$\boldsymbol{\mu}_\ell = \frac{1}{N} \sum_{i=1}^N \tilde{\mathbf{h}}_\ell^{(i)} \quad \boldsymbol{\sigma}_\ell^2 = \frac{1}{N} \sum_{i=1}^N (\tilde{\mathbf{h}}_\ell^{(i)} - \boldsymbol{\mu}_\ell)^2$$

- 然后，归一化后的 $\hat{\mathbf{h}}_{\ell}^{(i)}$ 会通过一个缩放系数 \mathbf{y}_{ℓ} 和一个偏移系数 $\boldsymbol{\beta}_{\ell}$ 进行进一步变换

$$\mathbf{z}_{\ell}^{(i)} = \mathbf{y}_{\ell} \odot \hat{\mathbf{h}}_{\ell}^{(i)} + \boldsymbol{\beta}_{\ell}$$

- 可以很容易地验证，如果 \mathbf{y}_{ℓ} 和 $\boldsymbol{\beta}_{\ell}$ 分别被设置为 $\boldsymbol{\sigma}_{\ell}$ 和 $\boldsymbol{\mu}_{\ell}$ ，那么 $\mathbf{z}_{\ell}^{(i)}$ 将恢复到输入状态 $\tilde{\mathbf{h}}_{\ell}^{(i)}$
 - 系数 \mathbf{y}_{ℓ} 和 $\boldsymbol{\beta}_{\ell}$ 与模型权重同时进行学习
- 计算精确的均值 $\boldsymbol{\mu}_{\ell}$ 和方差 $\boldsymbol{\sigma}_{\ell}^2$ 成本很高。在实践中，它们是通过当前的小批量 \mathcal{B} 来估计的

$$\boldsymbol{\mu}_{\ell} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \tilde{\mathbf{h}}_{\ell}^{(i)} \quad \boldsymbol{\sigma}_{\ell}^2 = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\tilde{\mathbf{h}}_{\ell}^{(i)} - \boldsymbol{\mu}_{\ell})^2$$

- 在测试阶段，我们不希望看到输出结果依赖于输入与哪个小批量相关联

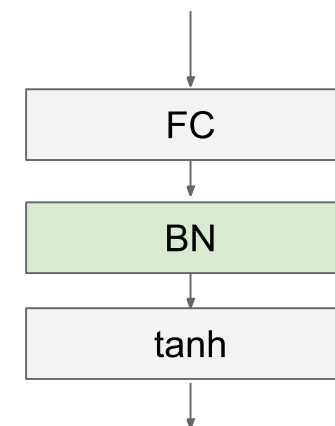
- 有两种解决方案：

➤ 在整个训练数据集上计算精确的均值 $\mu_\ell = \frac{1}{N} \sum_{i=1}^N \tilde{h}_\ell^{(i)}$ 和方差 $\sigma_\ell^2 = \frac{1}{N} \sum_{i=1}^N (\tilde{h}_\ell^{(i)} - \mu_\ell)^2$ 。只需计算一次

➤ 使用滑动平均来根据基于小批量的均值和方差，估计出精确的均值和方差

- 批量归一化可以被看作一个用 $BN(\cdot)$ 表示的层，即：

$$\gamma_\ell \odot (\tilde{h}_\ell^{(i)} - \mu_\ell) \oslash \sigma_\ell + \beta_\ell = BN(\tilde{h}_\ell^{(i)})$$



- 批量归一化在 MLP 和 CNN 中的区别

Batch Normalization for
fully-connected networks

$$\begin{array}{l}
 \mathbf{x}: \mathbf{N} \times \mathbf{D} \\
 \text{Normalize} \quad \downarrow \\
 \boldsymbol{\mu}, \sigma: 1 \times \mathbf{D} \\
 \gamma, \beta: 1 \times \mathbf{D} \\
 \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta
 \end{array}$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$\begin{array}{l}
 \mathbf{x}: \mathbf{N} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W} \\
 \text{Normalize} \quad \downarrow \quad \downarrow \quad \downarrow \\
 \boldsymbol{\mu}, \sigma: 1 \times \mathbf{C} \times 1 \times 1 \\
 \gamma, \beta: 1 \times \mathbf{C} \times 1 \times 1 \\
 \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta
 \end{array}$$

- 层归一化

Batch Normalization for
fully-connected networks

$$\begin{array}{l} \mathbf{x}: \mathbf{N} \times \mathbf{D} \\ \text{Normalize} \downarrow \\ \mu, \sigma: 1 \times \mathbf{D} \\ \gamma, \beta: 1 \times \mathbf{D} \\ \mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta \end{array}$$



Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$$\begin{array}{l} \mathbf{x}: \mathbf{N} \times \mathbf{D} \\ \text{Normalize} \downarrow \\ \mu, \sigma: \mathbf{N} \times 1 \\ \gamma, \beta: 1 \times \mathbf{D} \\ \mathbf{y} = \gamma(\mathbf{x} - \mu) / \sigma + \beta \end{array}$$

- 实例归一化

Batch Normalization for
convolutional networks

$$\begin{array}{l} \mathbf{x}: \mathbf{N} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W} \\ \text{Normalize} \quad \downarrow \quad \downarrow \quad \downarrow \\ \boldsymbol{\mu}, \sigma: \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1} \\ \gamma, \beta: \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1} \\ \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta \end{array}$$

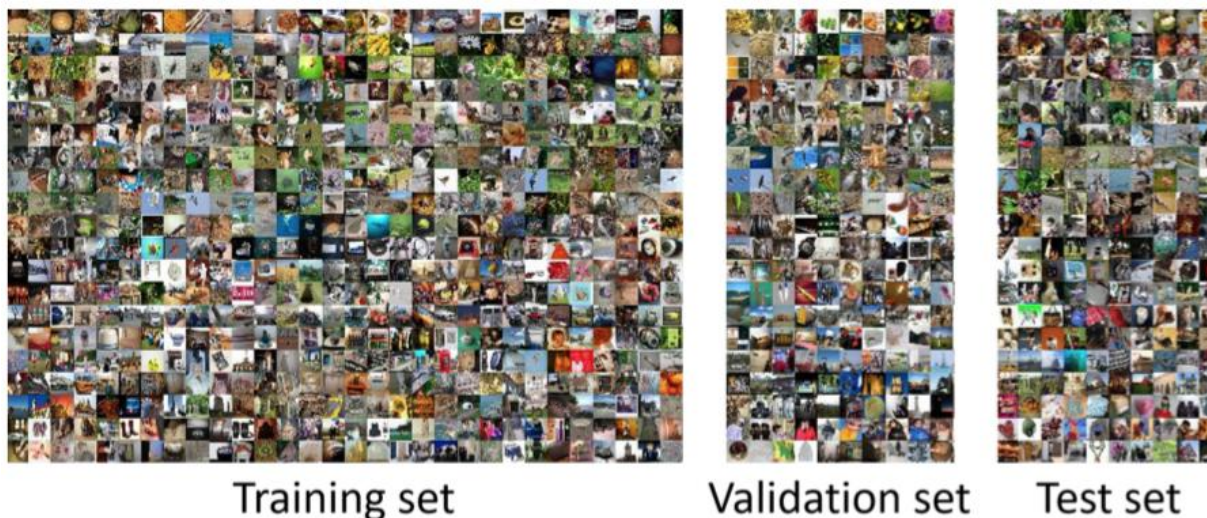


Instance Normalization for
convolutional networks
Same behavior at train / test!

$$\begin{array}{l} \mathbf{x}: \mathbf{N} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W} \\ \text{Normalize} \quad \quad \quad \downarrow \quad \downarrow \\ \boldsymbol{\mu}, \sigma: \mathbf{N} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1} \\ \gamma, \beta: \mathbf{1} \times \mathbf{C} \times \mathbf{1} \times \mathbf{1} \\ \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \sigma + \beta \end{array}$$

超参数调优

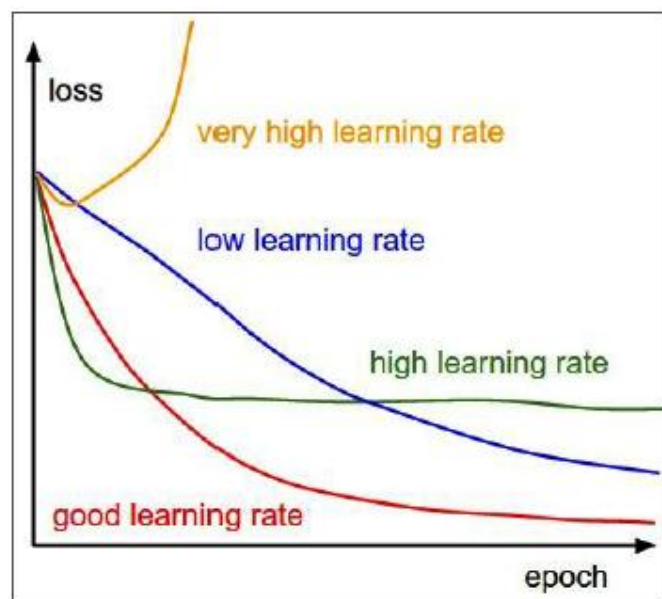
- 在开始训练之前，有很多参数需要设置，例如学习率、RMSProp 和 Adam 中的衰减参数、丢弃率等
- 为了给这些超参数设置合适的值，我们需要将整个数据集分为三部分，即训练集、验证集和测试集



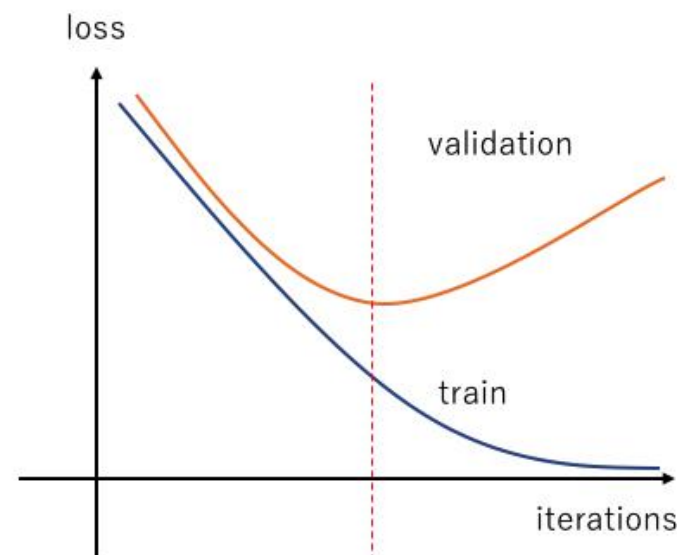
- 在训练集上训练模型
- 根据在验证集上的表现调优超参数
- 在测试集上测试性能，并将其作为最终性能进行报告

- 超参数的调整不是盲目的，而是可以遵循一些指导原则

- 1) 首先尝试一些在许多任务上表现良好的默认值
- 2) 根据训练过程中观察到的现象来选择值



训练损失曲线暗示了良好学习率的可能范围



训练集和验证集上的表现关系表明模型过于灵活