第七章 神经网络优化

修贤超

https://xianchaoxiu.github.io

神经网络优化

- 深度学习的矛与盾
  - 优化: 经验风险最小
  - 正则化: 降低模型复杂度

目录

参数学习

- 给定训练集为 $D = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{N}$，将每个样本 $\mathbf{x}^{(n)}$ 输入给前馈神经网络，得到网络输出为 $\hat{\mathbf{y}}^{(n)}$，其在数据集 D 上的结构化风险函数为

$$R(W, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{\lambda}{2} \|W\|_F^2$$

- 梯度下降

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial R(W, \mathbf{b})}{\partial W^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \frac{\partial R(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}$$

- 根据链式法则设计反向传播算法
- 一个更加通用的计算方法——自动微分（Automatic Differentiation，AD）

链式法则

- 若 $x \in \mathbb{R}$，$\mathbf{u} = u(x) \in \mathbb{R}^s$，$\mathbf{g} = g(\mathbf{u}) \in \mathbb{R}^t$，则

$$\frac{\partial \mathbf{g}}{\partial x} = \frac{\partial \mathbf{u}}{\partial x} \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \in \mathbb{R}^{1 \times t}$$

- 若 $\mathbf{x} \in \mathbb{R}^p$，$\mathbf{y} = g(\mathbf{x}) \in \mathbb{R}^s$，$\mathbf{z} = f(\mathbf{y}) \in \mathbb{R}^t$，则

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \in \mathbb{R}^{p \times t}$$

- 若 $X \in \mathbb{R}^{p \times q}$，$\mathbf{y} = g(X) \in \mathbb{R}^s$，$z = f(\mathbf{y}) \in \mathbb{R}$，则

$$\frac{\partial z}{\partial X_{ij}} = \frac{\partial \mathbf{y}}{\partial X_{ij}} \frac{\partial z}{\partial \mathbf{y}} \in \mathbb{R}$$

反向传播算法

- 考虑 $\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$，则

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

- 计算

$$\begin{aligned}
\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_m^{(l)}}{\partial w_{ij}^{(l)}}\right] \\
&= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right] \\
&= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right]
\end{aligned}$$

反向传播算法

- 令 $\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$ 为误差项，则 $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层权重 $W^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W^{(l)}} = \delta_i^{(l)} (\mathbf{a}^{(l-1)})^\top$$

- 对于 $\mathbf{z}^{(l+1)} = W^{(l+1)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)}$, $\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$，考虑误差项

$$\begin{aligned}
\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\
&= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^\top \cdot \delta^{(l+1)} \\
&= f_l'(\mathbf{z}^{(l)}) \odot (W^{(l+1)})^\top \cdot \delta^{(l+1)}
\end{aligned}$$

反向传播算法

- 同理，考虑

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} &= \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial (W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}\end{aligned}$$

- 于是 $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层权重 $\mathbf{b}^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

- 计算复杂，能否自动梯度计算?

计算图与自动微分

- 自动微分是利用链式法则来自动计算一个复合函数的梯度

$$f(x;w,b)=\frac{1}{\exp(-(wx+b))+1}$$

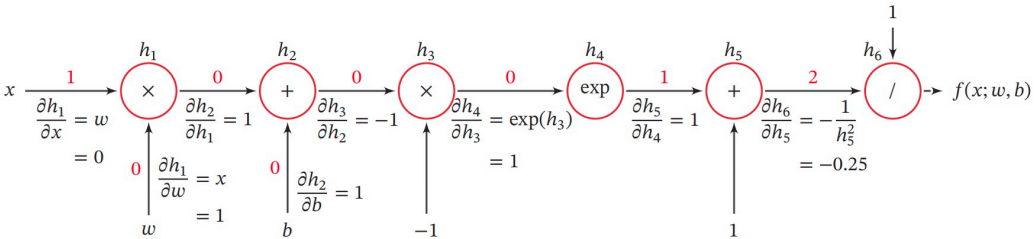
- 基本函数及其导数

函数	导数	
$h_1=x\times w$	$\frac{\partial h_1}{\partial w}=x$	$\frac{\partial h_1}{\partial x}=w$
$h_2=h_1+b$	$\frac{\partial h_2}{\partial h_1}=1$	$\frac{\partial h_2}{\partial b}=1$
$h_3=h_2\times -1$	$\frac{\partial h_3}{\partial h_2}=-1$	
$h_4=\exp(h_3)$	$\frac{\partial h_4}{\partial h_3}=\exp(h_3)$	
$h_5=h_4+1$	$\frac{\partial h_5}{\partial h_4}=1$	
$h_6=1/h_5$	$\frac{\partial h_6}{\partial h_5}=-\frac{1}{h_5^2}$	

计算图与自动微分

- 当 $x = 1,\ w = 0,\ b = 0$ 时，可以得到

$$\begin{aligned}\frac{\partial f(x;w,b)}{\partial w} &= \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w} \\ &= 1 \times (-0.25) \times 1 \times 1 \times (-1) \times 1 \times 1 = 0.25\end{aligned}$$



自动微分

- 前馈神经网络的训练过程可以分为以下三步
  - 前向计算每一层的状态和激活值，直到最后一层
  - 反向计算每一层的参数的偏导数
  - 更新参数
- 静态计算图
  - 在构建时可以进行优化，并行能力强，但灵活性比较差
  - 如 Tensorflow、Theano
- 动态计算图
  - 在程序运行时动态构建，灵活性比较高，但不容易优化，难以并行计算
  - 如 PyTorch、DyNet

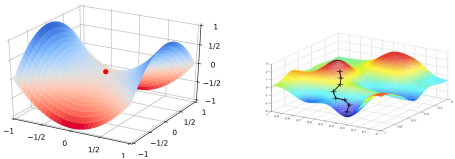
优化难点

- 深度学习的三个步骤

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

高维变量的非凸优化

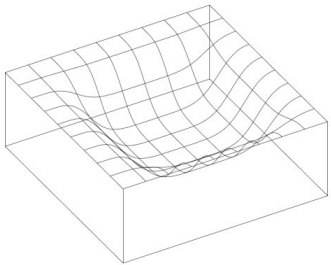
- 高维空间中非凸优化的难点不在于逃离局部最优点，而是如何逃离鞍点

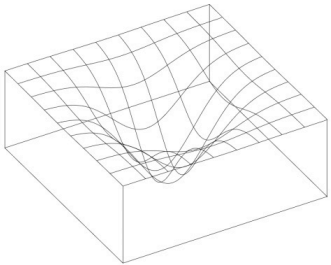


- 鞍点的梯度是 0，但是在一些维度上是最高点，在另一些维度上是最低点
- 通过在梯度方向上引入随机性，可以有效地逃离鞍点

高维变量的非凸优化

- 平坦最小值（Flat Minima ）
  - 大部分的局部最小解是等价的
  - 所有点对应的训练损失都比较接近
  - 局部最小解的训练损失可能非常接近全局最小解的训练损失





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

目录

优化算法

算法 随机梯度下降法

给定训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，验证集 $\mathcal{V}$，学习率 α，随机初始化 θ
while 未达到收敛准则 **do**
从训练集 $\mathcal{D}$ 中选取样本 $(\mathbf{x}^{(n)}, y^{(n)})$
$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}(\theta; \mathbf{x}^{(n)}, y^{(n)})}{\partial \theta}$
end while
输出θ

==========

- 随机梯度法仅计算选取样本处的梯度
- 要保证随机梯度的条件期望恰好是全梯度
- 回合（Epoch）= 训练样本的数量 N/批量大小 K × 迭代（Iteration）

小批量随机梯度下降

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

$$g_t(\theta) = \frac{1}{K} \sum \frac{\partial \mathcal{L}(\mathbf{y}; f(\mathbf{x}; \theta))}{\partial \theta}$$

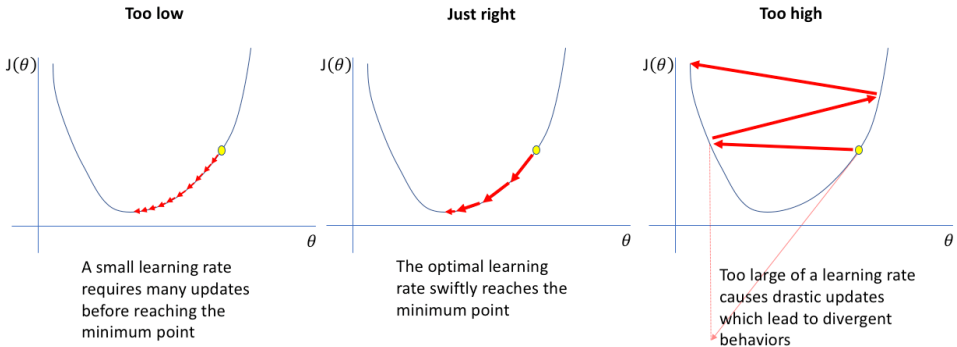
- 定义梯度 $g_t = g_t(\theta_{t-1})$，则更新参数

$$\theta_t \leftarrow \theta - \alpha g_t$$

- 几个关键因素
  - 小批量样本数量 K
  - 学习率 α: 学习率衰减、Adagrad、Adadelta、RMSprop
  - 梯度 g_t: Momentum、Nesterov、梯度截断

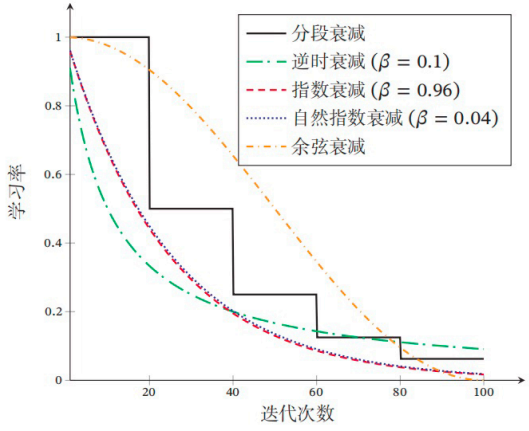
学习率的影响

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

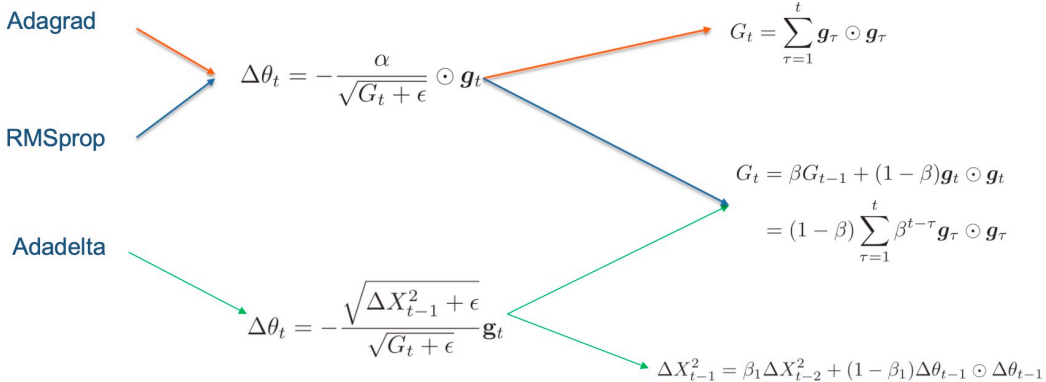


学习率衰减

- 学习率衰减也称为学习率退火



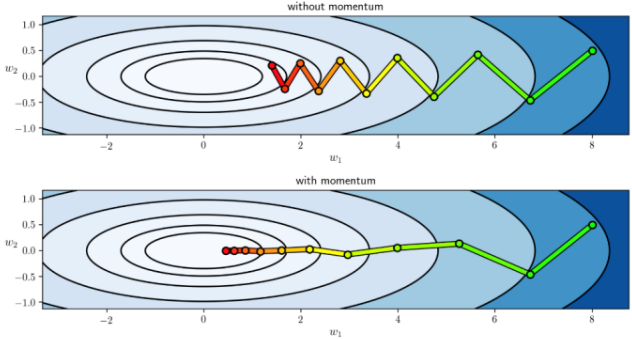
自适应学习率


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）
- 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

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha g_t = -\alpha \sum_{\tau=1}^{t} \rho^{t-\tau} g_\tau$$



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

$$M_t = \beta_1 M_{t-1} + (1-\beta_1) g_t$$
$$G_t = \beta_2 G_{t-1} + (1-\beta_2) g_t \odot g_t$$

  - 偏差修正

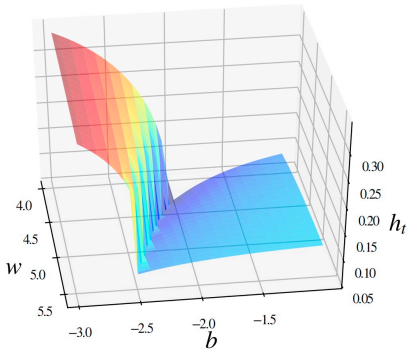
$$\hat{M}_t = \frac{M_t}{1-\beta_1^t},\ \hat{G}_t = \frac{G_t}{1-\beta_2^t}$$

  - 更新

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

梯度截断

- 把梯度的模限定在一个区间，当小于或大于这个区间时就进行截断
  - 按值截断　　$g_t = \max(\min(g_t, b), a)$
  - 按模截断　　$g_t = b \cdot g_t / \|g_t\|$



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

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

其中 $G_t = \psi(g_1, \cdots, g_t)$, $M_t = \phi(g_1, \cdots, g_t)$

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

目录

参数初始化

- 预训练初始化化（Pre-trained Initialization），又称精调（Fine-Tuning）
- 随机初始化（Random Initialization）
  - Gaussian 分布初始化: $\mu = 0, \sigma = 0.01$
  - 均匀分布初始化: $[-r, r]$
  - 基于方差缩放的参数初始化
  - 正交初始化方法
- 固定值初始化

正交初始化

- 范数保持性（Norm-Preserving ）
- 一个 M 层的等宽线性网络

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

- 为了避免梯度消失或梯度爆炸问题，希望误差项满足

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

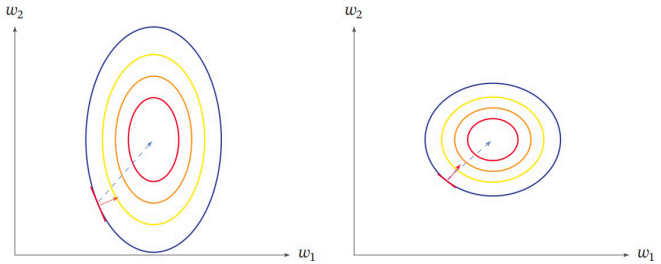
- 正交初始化化（Orthogonal Initialization）
  - 用均值为 0 、方差为 1 的高斯分布初始化一个矩阵
  - 用奇异值分解得到两个正交矩阵，并使用其中之一作为权重矩阵

数据归一化

- 最小最大值归一化（Min-Max Normalization）
- 标准化（Standardization）也叫 Z 值归一化（Z-Score Normalization）

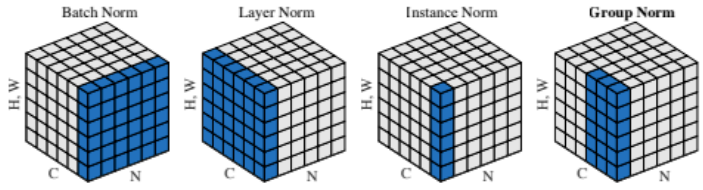
$$\mu = \frac{1}{N}\sum_{n=1}^{N} x^{(n)},\ \sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x^{(n)} - \mu)^2$$

- 白化（Whitening），用来降低输入数据特征之间的冗余性，如 PCA



逐层归一化

- 目的: 更好的尺度不变性、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

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

- 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

$$\mu_{\mathcal{B}} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{z}^{(k,l)}$$

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

- 批量归一化

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

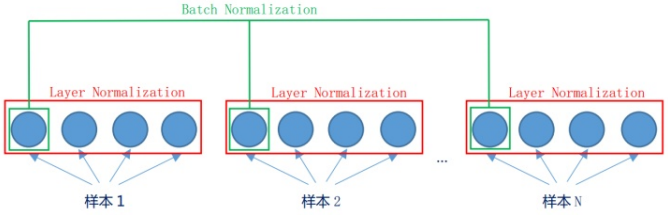
层归一化

- 第 l 层神经元的净输入为 $\mathbf{z}^{(l)}$

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

- 层归一化定义为

$$\hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \mu^{(l)}}{\sqrt{\mu^{(l)^2} + \epsilon}} \odot \gamma + \beta \triangleq \mathrm{LN}_{\gamma,\beta}(\mathbf{z}^{(l)})$$



超参数优化

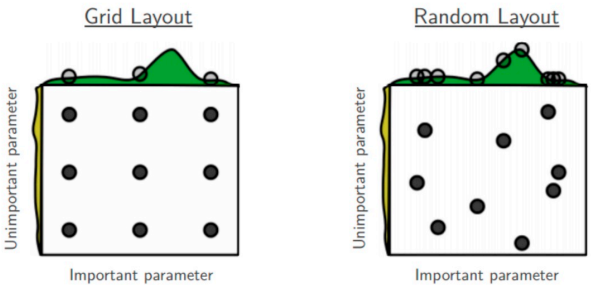
- 超参数
  - 层数
  - 每层神经元个数
  - 激活函数
  - 学习率（以及动态调整算法）
  - 正则化系数
  - mini-batch 大小
- 优化方法
  - 网格搜索、随机搜索
  - 贝叶斯优化、动态资源分配、神经架构搜索

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，可以离散化选择几个经验值。如学习率 α 设置为

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

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



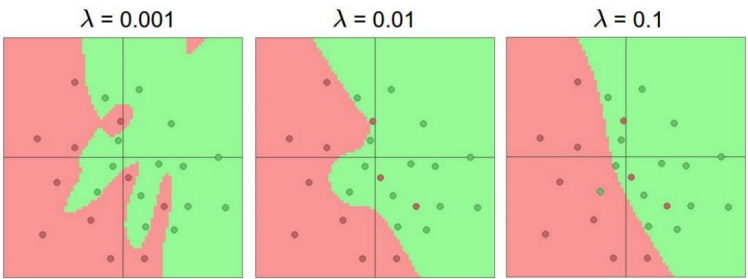
目录

正则化（Regularization）

- 优化问题可以写为

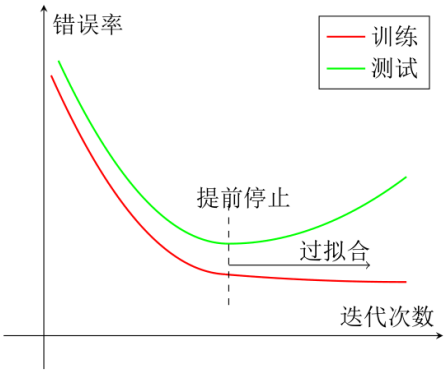
$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y^{(n)}, f(\mathbf{x}^{(n)}, \theta)) + \lambda \ell_p(\theta)$$

- http://playground.tensorflow.org



提前停止（Early-stop）

- 使用验证集（Validation Dataset）测试每一次迭代的参数是否最优
- 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 ω

$$\theta_t \leftarrow (1-\omega)\theta_{t-1} - \alpha g_t$$

- 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

[PDF] Fixing weight decay regularization in adam

I Loshchilov, F Hutter - arXiv preprint arXiv:1711.05101, 2017 - arxiv.org

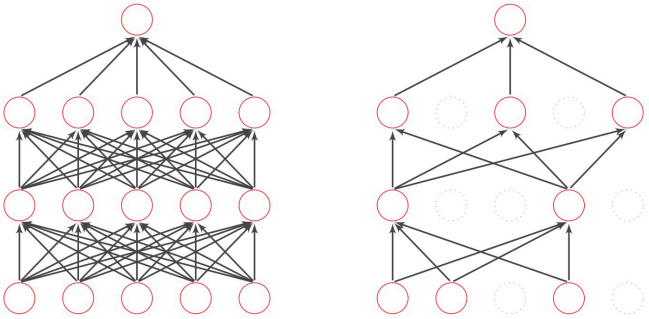
L2 regularization and weight decay regularization are equivalent for standar
gradient descent (when rescaled by the learning rate), but as we demonstra
case for adaptive gradient algorithms, such as Adam. While common deep l

丢弃法（Dropout）

- 对于神经层 $y = f(W\mathbf{x} + \mathbf{b})$，引入丢弃函数 $d(\cdot)$ 使得 $y = f(Wd(\mathbf{x}) + \mathbf{b})$

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

- 这里 $\mathbf{m} \in \{0, 1\}^d$ 是丢弃掩码，通过以概率为 p 的贝努力分布随机生成



Dropout 意义

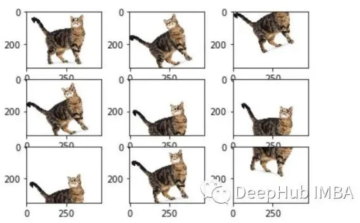
- 集成学习的解释
  - 每做一次丢弃，相当于从原始的网络中采样得到一个子网络
  - 如果一个神经网络有 n 个神经元，那么总共可以采样出 2^n 个子网络
- 贝叶斯学习的解释

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

其中 $f(\mathbf{x}, \theta_m)$ 为第 m 次应用丢弃方法后的网络

数据增强（Data Augmentation）

- 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
  - 旋转: 将图像按顺时针或逆时针方向随机旋转一定角度
  - 翻转: 将图像沿水平或垂直方法随机翻转一定角度
  - 缩放: 将图像放大或缩小一定比例
  - 平移: 将图像沿水平或垂直方法平移一定步长
  - 噪声: 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 $\mathbf{x}$ 的标签可用 onep-hot 向量表示

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

- 引入噪声对标签进行平滑，即设样本以 ϵ 的概率为其它类，平滑后的标签为

$$\hat{\mathbf{y}} = [\frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}, 1-\epsilon, \frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}]^\top$$

- 一种更好的做法是按照类别相关性来赋予不同的概率，称为知识蒸馏

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
- **优化**
  - SDG + mini-batch（动态学习率、Adam 算法优先）
  - 数据预处理（标准归一化、逐层归一化）
  - 参数初始化
- **正则化**
  - ℓ_1 和 ℓ_2 正则化
  - Dropout、Early-stop、数据增强

Q&A

Thank you!

感谢您的聆听和反馈