第六讲 正则化、规一化和预训练

一、数据、模型与特征工程

数据

- 简单神经网络难以在真实任务上发挥最大性能，因为模型和数据往往存在各种各样的问题，比较常见的数据问题有：
- 不同维度差异过大（数据中心偏置）

时间	1.1	2.1	3.1	4.1	5.1
用水量	2000	2300	2201	2102	2003
用电量	40	50	45	67	60

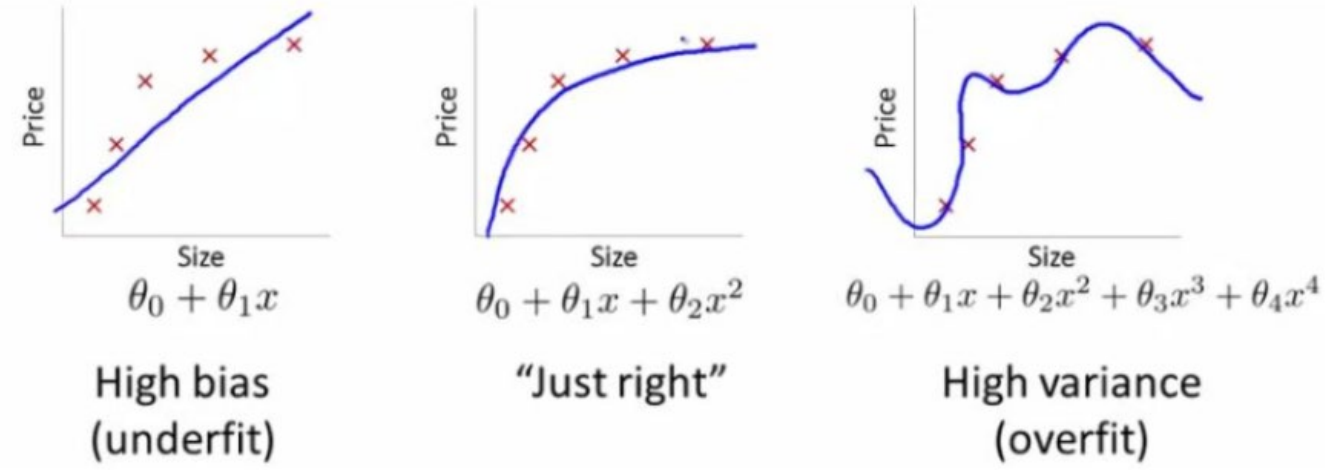
- 正负例样本不均衡

测试病人总数	患肺癌	不患肺癌
1520	18	1502

- ……

模型

- 模型问题
- 过/欠拟合



- 梯度消失/梯度爆炸
- 模型过大，难以重新训练等
- ……

- 解决上述问题常用的手段有：

1、正则化，规一化

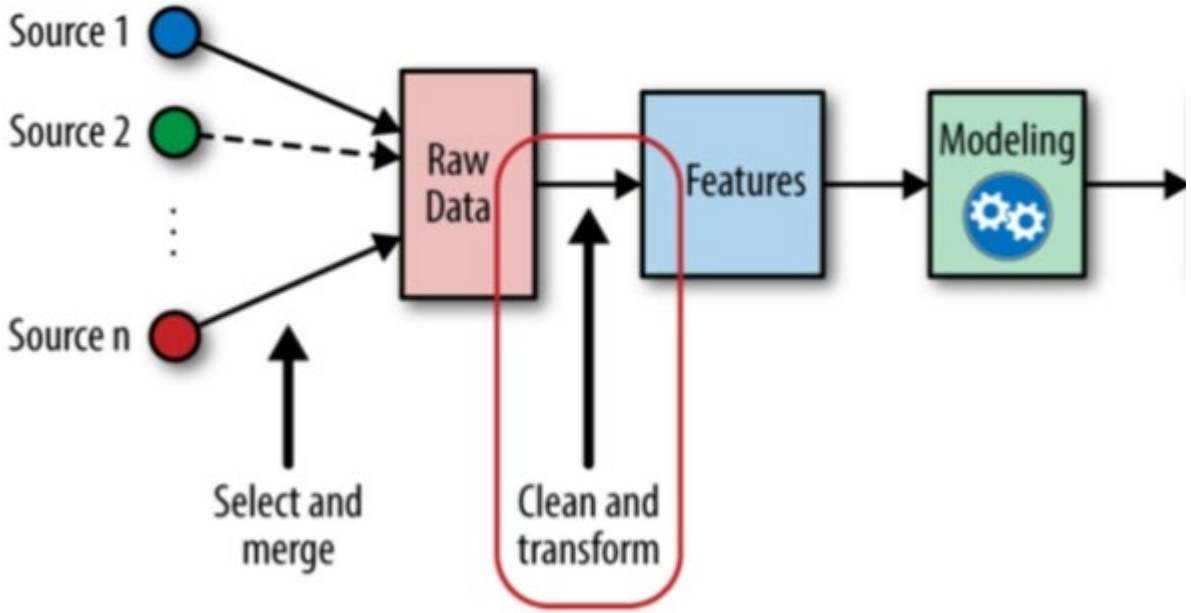
2、预训练和迁移学习

3、特征预处理（特征工程）

……

特征工程

- 特征工程是机器学习，甚至是深度学习中最为重要的一部分，是数据科学中最有创造力的一部分。
- 神经网络任务往往和具体的数据相结合，针对数据中存在的问题需要使用特殊的方法使得后续模型正常工作。
- 特征工程就是通过X，创造新的X'。



特征工程

从给定的特征集合中选出相关特征子集的过程称为特征选择(feature selection)。

1.对于一个学习任务，给定了属性集，其中某些属性可能对于学习来说很关键，但有些属性意义就不大。

- 对当前学习任务有用的属性或者特征，称为相关特征(relevant feature)；
- 对当前学习任务没用的属性或者特征，称为无关特征(irrelevant feature)。

2.特征选择可能会降低模型的预测能力，因为被剔除的特征中可能包含了有效的信息，抛弃这部分信息一定程度上会降低模型的性能。但这也是计算复杂度和模型性能之间的取舍：

- 如果保留尽可能多的特征，模型的性能会提升，但同时模型就变复杂，计算复杂度也同样提升；
- 如果剔除尽可能多的特征，模型的性能会有所下降，但模型就变简单，也就降低计算复杂度。

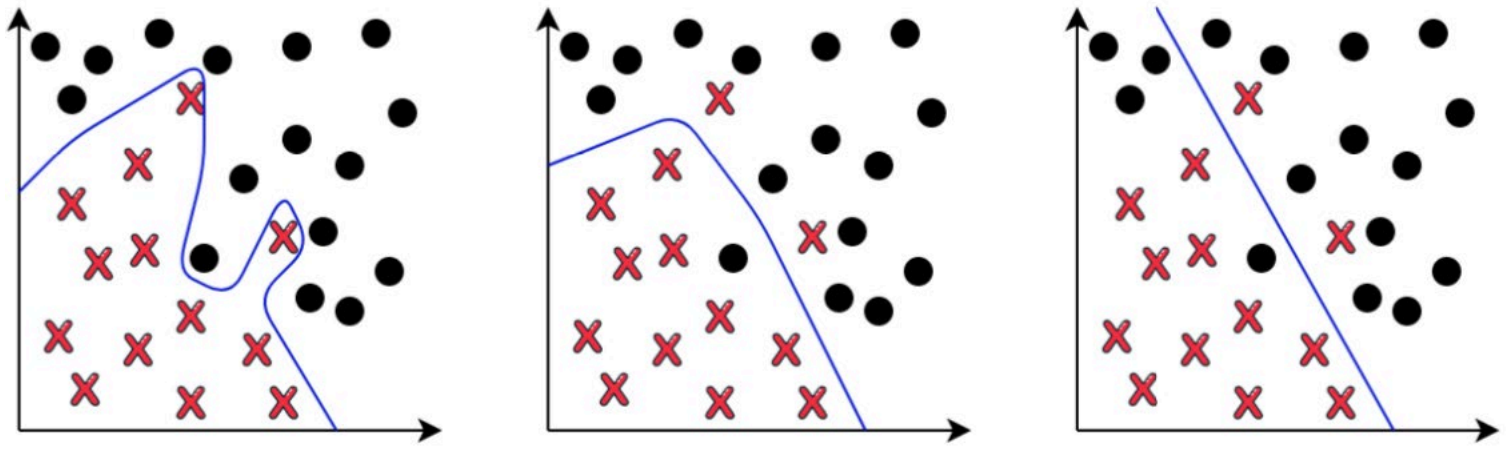
二、正则化与归一化

正则化

- 过拟合和欠拟合

过拟合：模型过于复杂，参数过多或训练数据过少，噪声过多。

欠拟合：模型比较简单，特征维度过少。



(a) 过拟合　(b)好拟合　（c）欠拟合

猫狗二分类，黑点代表猫，红色叉代表狗

正则化

- 偏差和方差：定量分析
- 偏差：衡量模型预测值和实际值之间的偏离关系，即模型在样本上拟合得好不好。
- 方差：描述模型在整体数据上表现的稳定情况，在训练集和验证集/测试集上表现是否一致。

情况	1	2	3	4
训练集错误率	16%	13%	3%	1%
验证集错误率	35%	17%	13%	2%

- 情况 1: 高偏差，高方差 (差模型)。
- 情况 2: 高偏差，低方差 (欠拟合)。
- 情况 3: 低偏差，高方差 (过拟合)。
- 情况 4: 低偏差，低方差 (好模型)。

猫狗二分类，训练集和验证机错误率的各种情况

正则化

- 正则化：通过限制模型的复杂度，避免过拟合，提高泛化能力。

不同的正则化方法

1、L1和L2正则化

2、权重衰减法

3、丢弃法

4、提前停止

5、数据增强

正则化：L1和L2正则化

- L_p 范数：$\|x\|_p = (\sum_i |x_i|^p)^{1/p}, p \in [1, \inf)$
- L_1 范数：$\|x\|_1 = \sum_i |x_i|$
- L_2 范数：$\|x\|_2 = (\sum_i |x_i|^2)^{1/2}$

优化目标：

$$\arg\min_{\theta} \frac{1}{N}(\sum_{i}^{N} \mathcal{L}(y_i, f(x_i, \theta)))$$

其中$\mathcal{L}(.)$为损失函数，$f(.)$为待学习的神经网络，θ 为参数，N为样本数量。

正则化：L1和L2正则化

- 加上L_1正则化后优化目标，λ为正则化系数：

$$\arg\min_{\theta} \frac{1}{N}\left(\sum_{i}^{N} \mathcal{L}(y_i, f(x_i, \theta)) + \lambda|\theta|\right)$$

- 每次更新θ，假设学习率为α

$$\begin{aligned}\theta &:= \theta - \alpha d\theta \\ &= \theta - \frac{\alpha}{N}\left(\frac{\partial \mathcal{L}}{\partial \theta} + \lambda sign(\theta)\right) \\ &= \theta - \frac{\alpha}{N}\frac{\partial \mathcal{L}}{\partial \theta} - \frac{\alpha\lambda}{N} sign(\theta)\end{aligned}$$

$$sign(\theta) = \begin{cases} 1, & \theta > 0 \\ 0, & \theta = 0 \\ -1, & \theta < 0 \end{cases}$$

- 目的：使θ更容易为0，整体权重矩阵更为稀疏，抑制过拟合。

正则化：L1和L2正则化

- 加上L_2正则化后优化目标，λ为正则化系数：

$$\arg\min_{\theta} \frac{1}{N}\left(\sum_{i}^{N} \mathcal{L}(y_i, f(x_i, \theta)) + \lambda\theta^2\right)$$

- 每次更新θ，假设学习率为α

$$\begin{aligned}\theta &:= \theta - \alpha d\theta \\ &= \theta - \frac{\alpha}{N}\left(\frac{\partial \mathcal{L}}{\partial \theta} + 2\lambda\theta\right) \\ &= \left(1 - \frac{2\alpha\lambda}{N}\right)\theta - \alpha\frac{\partial \mathcal{L}}{N\partial\theta}\end{aligned}$$

$0 < (1 - 2\alpha\lambda/N) < 1$
使得权重变的更小

正则化：权重衰减

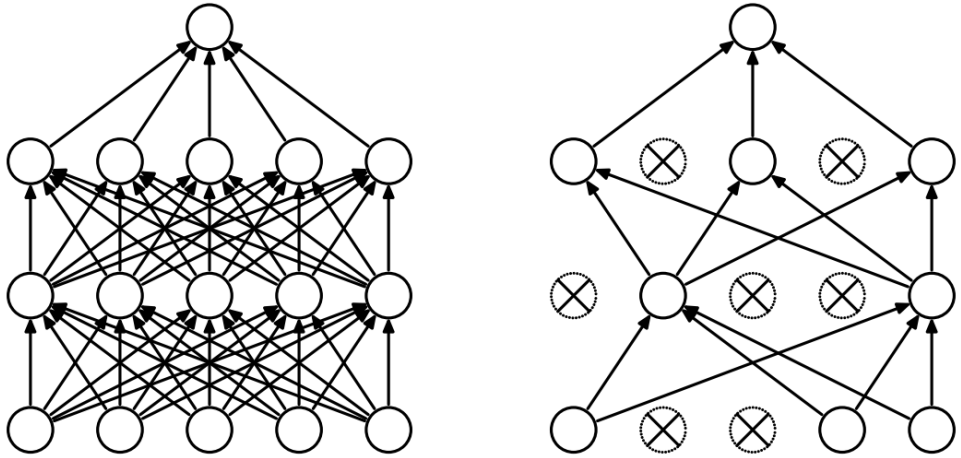
- 思路：每次参数更新时，都先对参数进行一定衰减。

$$\theta := (1 - w)\theta - \alpha d\theta$$

- 其中w为权重衰减系数。

正则化：丢弃法（Dropout）

- 思路：在训练时，以概率p随机丢弃部分神经元。



对一神经元层进行Dropout操作得到$y = f(Wd(x) + b)$，其中$d(.)$为丢弃函数：

$$d(\boldsymbol{x}) = \begin{cases} m * \boldsymbol{x}, & At\ training\ time \\ p\boldsymbol{x}, & At\ test\ time \end{cases}$$

m~Bernoulli(p)表示以概率p的伯努利分布生成0,1向量

正则化：丢弃法（Dropout）

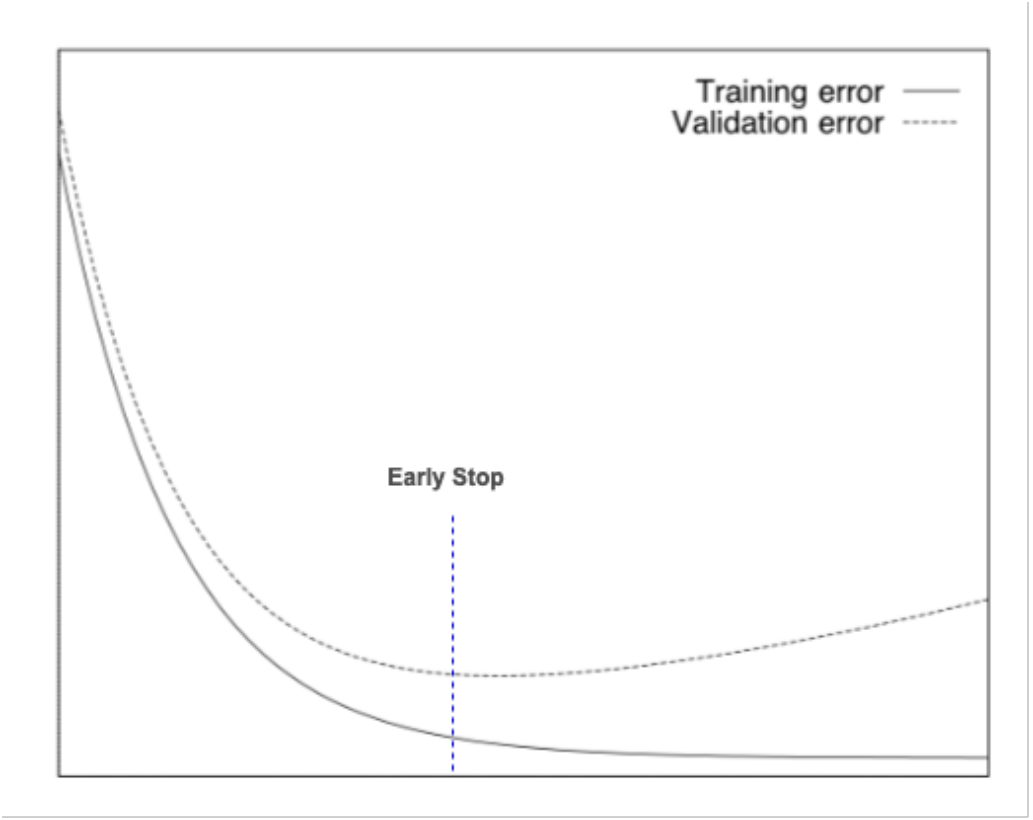
- 分析原因：

- 1.相当于取平均的作用，取每次丢弃后子网络的平均结果。

- 2.降低神经元之间的敏感度，增加整体鲁棒性。

正则化：提前停止

- 思路：提早结束训练
- 验证集错误率基本不下降时或有反增趋势时，可以提前停止训练。



正则化：数据增强

- 思路：在不实质增加数据的情况下，对当前数据执行一些操作达到数据增加的效果
- 图像数据：翻转、旋转、镜像、裁剪、增加高斯白噪声等
- 文本数据：同义词替换、随机插入、随机交换、随机删除等

归一化

- 为什么要归一化?

实例：从给定的房子面积$x_1(0\text{–}1000m^2)$和房间数量$x_2\ (0-10)$，来预测房子的总价格。

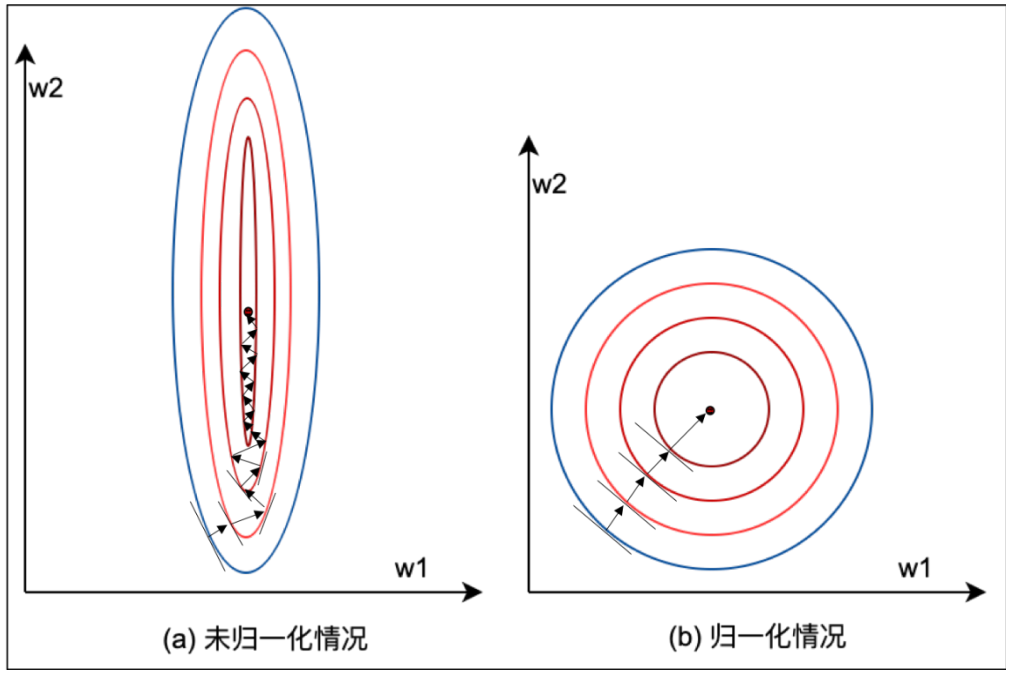
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

优化的目标函数:

$$\mathcal{L}(\theta) = (\theta_0 + \theta_1 x_1 + \theta_2 x_2 - y)^2$$

其中y为真实的房子总价。

x_1和x_2取值范围差别比较大:



数据归一化对梯度的影响

归一化

- 最常用的归一化方法

1、Min-max归一化：将结果映射到[0,1]之间。

$$\hat{x} = \frac{x - min(x)}{max(x) - min(x)}$$

2、Z-score归一化（标准归一化）：

$$\hat{x} = \frac{x - \mu}{\sigma}$$

其中μ为均值，σ为标准差

批归一化（Batch Normalization，BN）

- 思路：逐层归一化方法，对神经网络中任意的中间层进行归一化操作
  - 使得净输入的分布一致（例如正态分布），一般应用在激活函数之前
- 算法流程：

Algorithm 1: BN 在 Mini-Batch 上的归一化

Input: 每个 Mini-Batch 上的 x: $\mathcal{B} = \{x_{1...m}\}$;
需学习的参数: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// Mini-Batch 均值}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// Mini-Batch 方差}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// 进行归一化}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// 缩放参数 } \gamma\text{, 偏移参数 } \beta$$

层归一化（Layer Normalization，LN）

- 思路：对中间层的所有神经元进行归一化
- 算法流程:

1.计算网络层的均值和方差

$$\mu^{(l)} \leftarrow \frac{1}{n^{(l)}} \sum_{i=1}^{n^{(l)}} x_i^{(l)}$$ 求均值

$$\sigma^{(l)2} \leftarrow \frac{1}{n^{(l)}} \sum_{i=1}^{n^{(l)}} (x_i^{(l)} - \mu^{(l)})^2$$ 求方差

$n^{(l)}$:第l层神经元数量

$x_i^{(l)}$:第l层第i个神经元的输入

2. 进行归一化

$$\widehat{x}_i^{(l)} \leftarrow \frac{x_i^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)2} + \epsilon}}$$

3.设置可训练的缩放和偏移参数，映射数据

$$y_i^{(l)} \leftarrow \gamma \widehat{x}_i^{(l)} + \beta \equiv LN_{\gamma,\beta}(x_i^{(l)})$$

实例归一化（Instance Normalization，IN）

- 主要用于依赖于某个图像实例的任务。
- 假设输入的特征图为 $x \in R^{N*C*H*W}$ 。

N：batch维度，C：特征通道维度，H、W：特征图高和宽维度。

- IN思路：对每个样本的H和W的数据求均值和标准化，保留N、C维度。

$$\mu_{nc} \leftarrow \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H} x_{nchw} \quad \text{求均值}$$

$$\sigma_{nc}^2 \leftarrow \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H}(x_{nchw}-\mu_{nc})^2 \quad \text{求方差}$$

$$\widehat{x}_{nchw} \leftarrow \frac{x_{nchw}-\mu_{nc}}{\sqrt{\sigma_{nc}^2+\epsilon}}$$

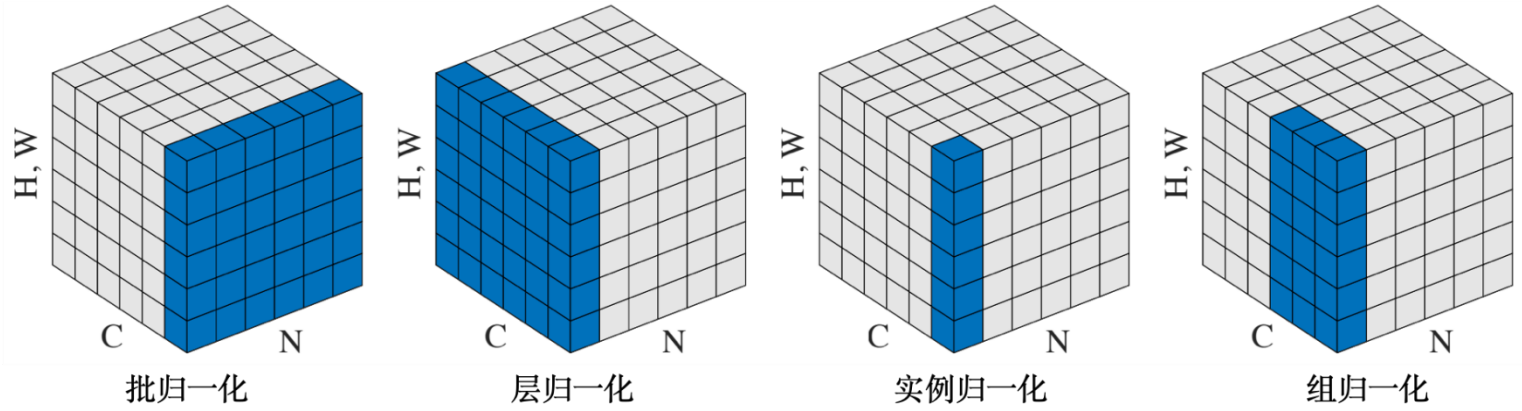
$$y_{nchw} \leftarrow \gamma\widehat{x}_{nchw}+\beta \equiv IN_{\gamma,\beta}(x_{nchw})$$

组归一化（Group Normalization，GN）

- 主要用于依赖于某个图像实例的任务。
- 输入的特征图为 $x \in R^{N*C*H*W}$。

N：batch维度，C：特征通道维度，H、W：特征图高和宽维度。

把特征通道分为G组，每组有C/G个特征通道，在组内归一



- 组归一化是层归一化和实例归一化的折中。

可转换归一化（Switchable Normalization，SN）

- 将批归一化、层归一化、实例归一化结合起来的方法，使网络自适应学习如何组合起来的权重。

- 思路：假设三种方法学习到的均值和方差分别为μ_{bn}, μ_{ln}, μ_{in}；σ_{bn}, σ_{ln}, σ_{in}

$$\mu_{sn} \leftarrow \omega_{bn}\mu_{bn} + \omega_{ln}\mu_{ln} + \omega_{in}\mu_{in} \quad \text{求均值}$$

$$\sigma^2_{sn} \leftarrow \omega'_{bn}\sigma^2_{bn} + \omega'_{ln}\sigma^2_{ln} + \omega'_{in}\sigma^2_{in} \quad \text{求方差}$$

进而归一化，映射数据。

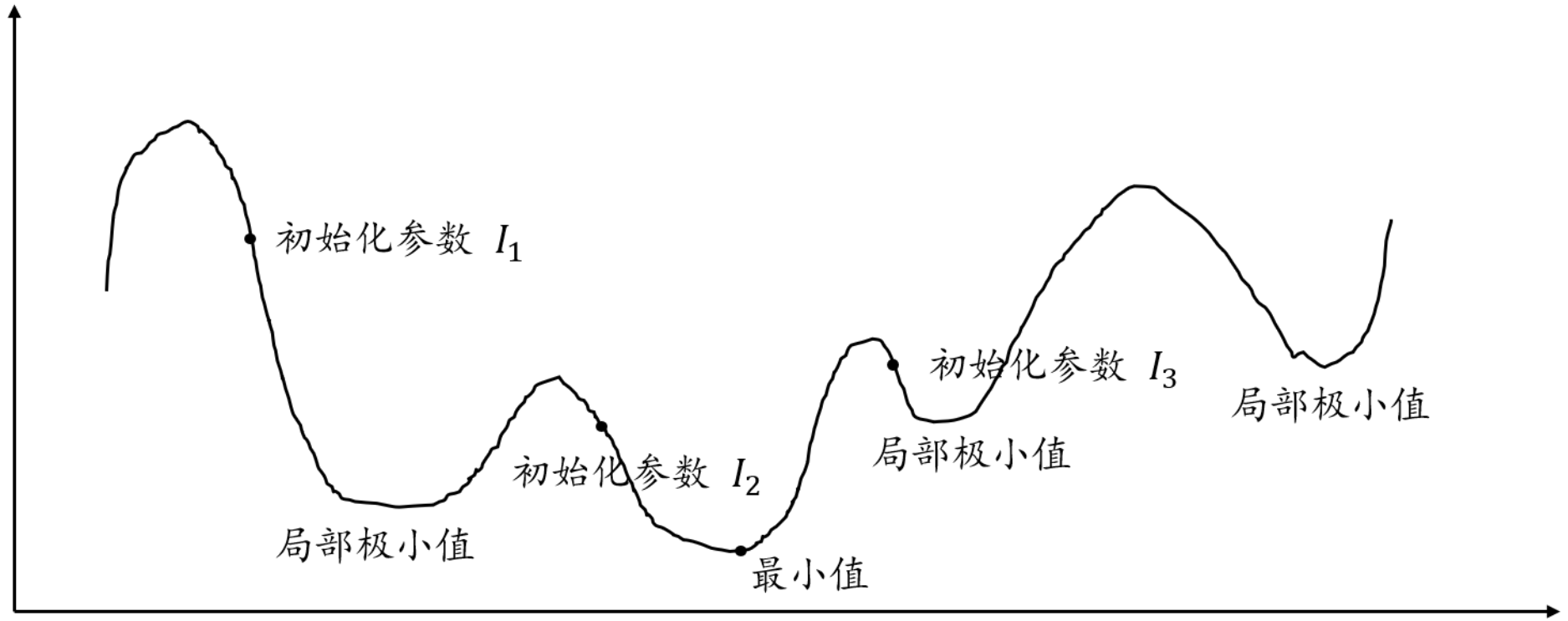
三、初始化、预训练和迁移学习

网络参数初始化

思考：根据之前的学习，考虑为什么要研究网络参数的初始化?

Hint：差的初始化会有什么结果?

网络参数初始化为什么重要?



不同的初始化参数，计算得到的“全局最小”差距很大

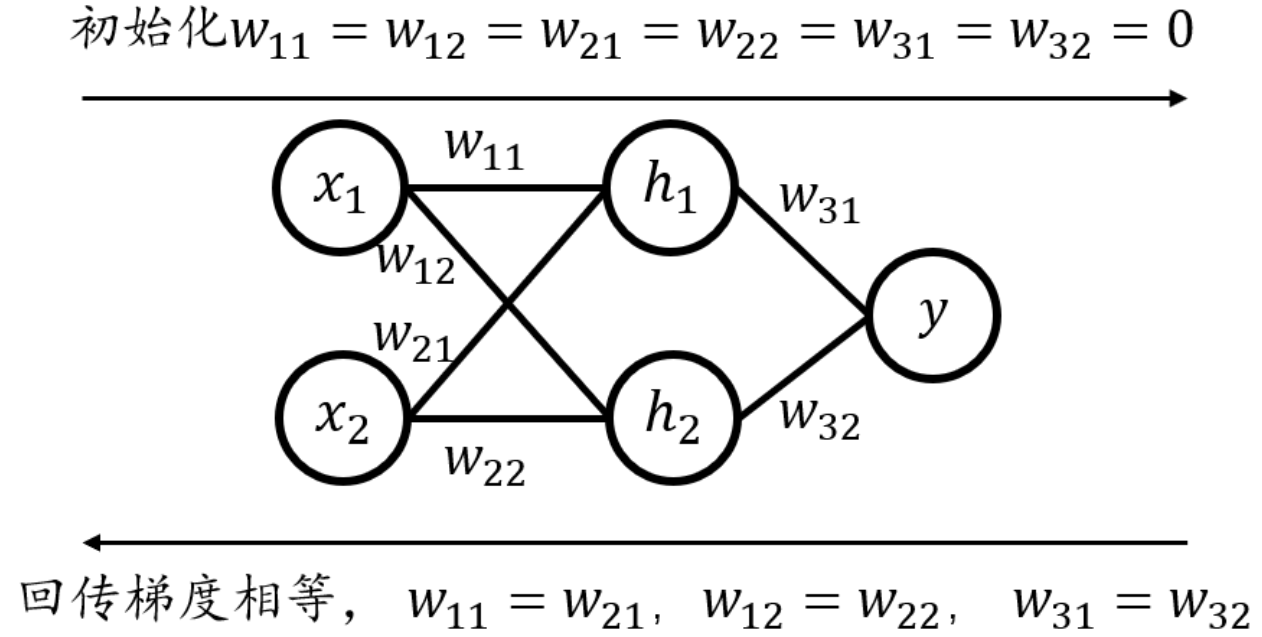
- 多层神经网络的目标函数通常是非凸函数，求解困难。

- 常用的优化算法的基本思想：针对目标函数的梯度下降，局部极小值点的存在使得优化的结果很难是最优解。

- **好的初始化值能够帮助网络更快地计算得到最优值，更容易收敛到目标函数。**

网络初始化

- 目前没有发现一种初始化方式可以适用于任何网络结构
- 初始化需要避免“对称权重”现象（唯一确知的特性）



权重矩阵的初始化

启发式思考：

- 考虑激活函数的学习曲线，初始化权重应该分布在梯度较大的区域；
- 初始化权重不应过大或者过小
  - 思考：权重过大或者过小分别有什么影响？
- 优化角度：权重应该足够大来传播信息
- 正则化角度：权重应该较小

权重矩阵的初始化

1. 均匀分布初始化

2. 高斯分布初始化

3. 稀疏初始化

4. 正交初始化

均匀分布初始化

主要思想：在区间(-r,r)的均匀分布U(-r,r)中随机选取所有网络权值

- 如何确定区间范围？
  - 尽量保持梯度能够在多层网络中传播

- 规则1：

  对于任意网络权值w_{ij}，从如下分布中随机选取初始值：

$$w_{ij} \sim U\left(-\frac{1}{\sqrt{n_i}}, \frac{1}{\sqrt{n_i}}\right)$$

  其中n_i表示第i层的神经元数量。

均匀分布初始化

- 规则2：Xavier 初始化

对于任意网络权值w_{ij}，从如下分布中随机选取初始值：

$$w_{ij} \sim U\left(-\sqrt{\frac{6}{n_i + n_j}}, \sqrt{\frac{6}{n_i + n_j}}\right)$$

其中n_i表示第i层的神经元数量，n_j 表示第j（j=i+1）层的神经元数量。

经验表明，对于激活函数设置为tanh的神经元，分布区间也可以为

$$w_{ij} \sim U\left(-4\sqrt{\frac{6}{n_i + n_j}}, 4\sqrt{\frac{6}{n_i + n_j}}\right)$$

这种初始化方法能够保持激活方差以及回传方差，从而能够保证网络的训练效果。

高斯分布初始化

主要思想：在固定均值和固定方差的高斯分布中随机选取所有网络权值

- 启发式规则1：

  采用均值为0，方差为常数k的正态分布，即：

  对于任意网络权值w_{ij}，从如下分布中随机选取初始值：

$$w_{ij} \sim N(0, k)$$

  缺陷：深层模型会非常难以收敛

高斯分布初始化

- 启发式规则2：He初始化

  主要针对非线性激活函数神经元（例如ReLU）

$$w_{ij} \sim N(0, \sqrt{\frac{2}{n_i}})$$

  如果考虑输出层神经元个数：

$$w_{ij} \sim N(0, \sqrt{\frac{2}{n_i + n_j}})$$

在较为浅层的网络中，Xavier和He两种初始化方法最终的训练结果相差不大，但是在更为深层（30层以上）的网络上，He初始化方法会获得更好的结果。

稀疏初始化

主要思想：稀疏初始化降低连接数量，使得初始化的数值不会太小

- 传统“稠密”初始化范围的大小和神经元数量有关，
  - 越多的神经元连接会导致越小的初始化范围

- 减少神经元数量可以增大初始化范围，使得神经元初始化之间的差异增大

- 启发式规则：
  - 每层神经元只与k个神经元连接，其余神经元连接初始化为0

正交初始化

主要思想：正交初始化可以避免训练开始时就出现梯度消失或梯度爆炸现象

- 启发式规则：

  初始化权重矩阵为正交矩阵，满足$W^T W = I$

具体：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，使用其中之一作为权重矩阵

实际中：通常需要将正交矩阵乘以一个缩放系数

偏置矩阵初始化

- 偏置矩阵通常不需要考虑破坏对称性的问题，通常我们可以把偏置矩阵初始化为全0矩阵；

- 通常情况下，偏置矩阵还是会初始化为全0矩阵，除了一些例外情况：
  - 偏置作为输出单元，初始化偏置以获得正确的输出边缘的统计是有利的；
  - 需要选择偏置以避免初始化引起的太大饱和

初始化参数对训练的优化程度

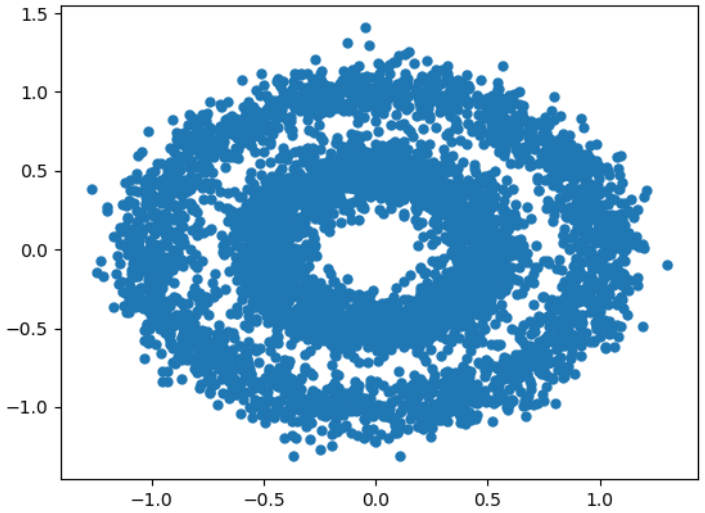
- 不合理的初始化会导致**梯度消失**或**爆炸**现象

（1）大的梯度会使网络十分不稳定，会导致权重成为一个特别大的值，最终导致溢出而无法学习

（2）小的梯度传过多层网络，达到靠近输入的隐藏层后会越来越小，导致隐藏层无法正常地进行学习。

初始化参数对训练的优化程度：实例

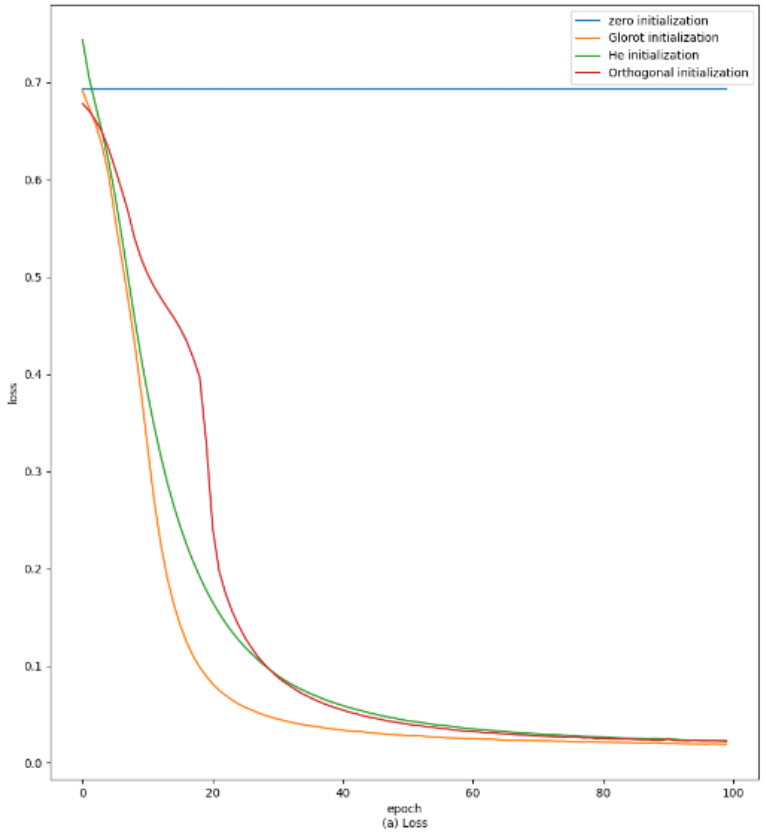
- 人工生成一个“同心圆环”数据集，数据共有2类，5000个样本。
- 数据分布如下图所示。

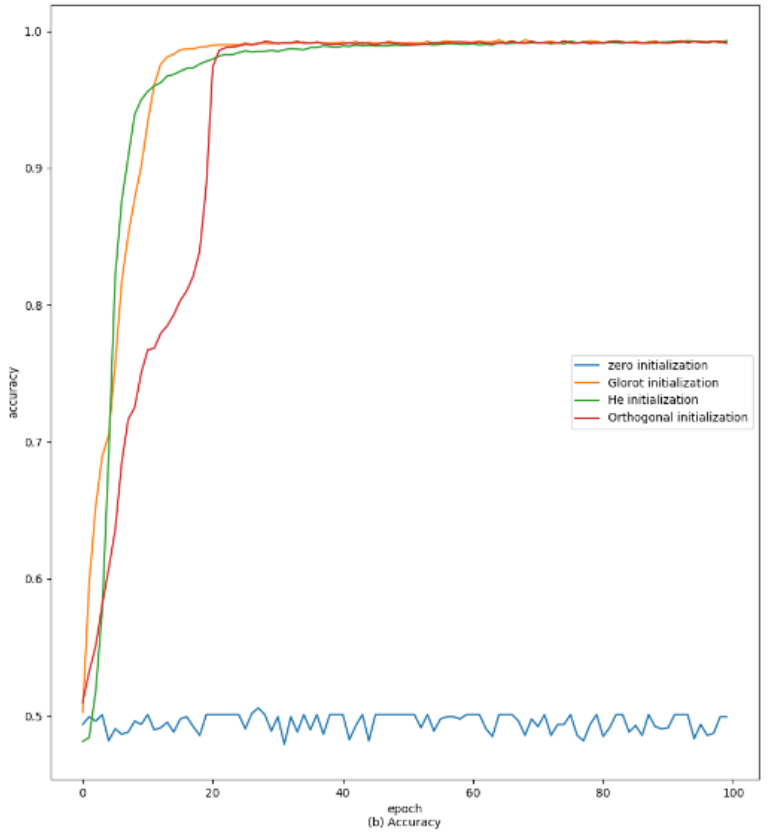


- 在这里，我们采用一个简单的多层神经网络，其输入输出层都是2个神经元，具体结构为每层[2,5,3,2]个神经元。输出层使用softmax进行分类，其余层使用relu函数作为激活函数。

初始化参数对训练的优化程度

- 对比全0初始化和前面介绍的三种初始化方法：Xavier初始化、He初始化和正交初始化在这个数据集上的表现。





- 全0初始化将使得网络无法正确学习分类结果，其余几种初始化方式在这个问题上能够达到差不多的水平。

网络预训练和迁移学习

网络预训练

- 网络预训练
  - 采用相同结构的，并且已经训练好的网络权值作为初始值，在当前任务上再次进行训练
- 为什么使用网络预训练?
  - 为了能够在更短时间内训练得到更好的网络性能
  - 相似的任务之间，训练好的神经网络可以复用，通常作为特征提取器

预训练方法

- 无监督预训练
  - 玻尔兹曼机
  - 自编码器
- 有监督预训练

无监督预训练

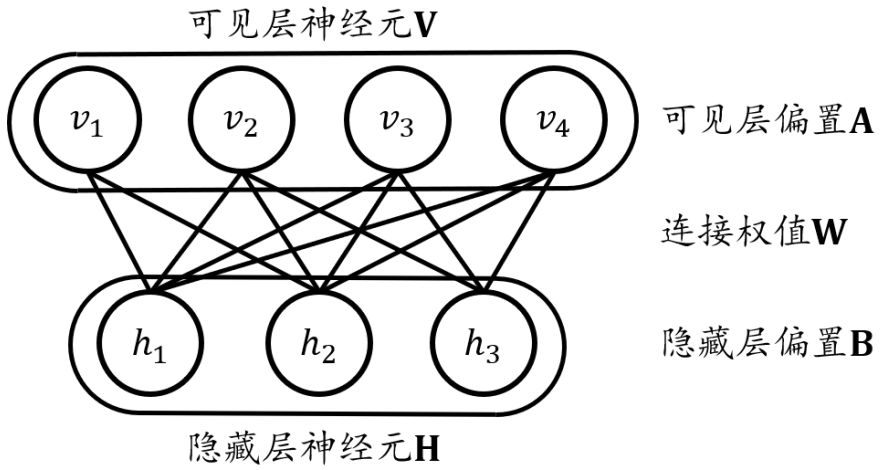
主要思想：通过无标签数据辅助神经网络的训练

- 梯度消失问题：
  - 梯度随着传递层数的增加迅速减小，靠近输入层的连接权值几乎没有进行学习；
  - 深层的网络难以训练
- 逐层预训练
  - 每次预训练只预训练一层；
  - 无监督训练网络：
    - 玻尔兹曼机
    - 自编码器

无监督预训练

玻尔兹曼机（Boltzmann Machines，BM）

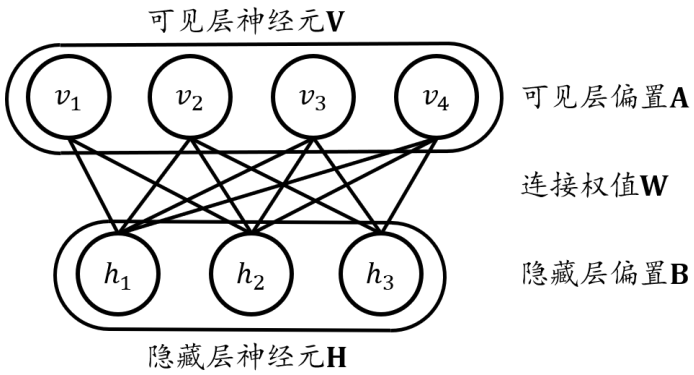
- 一种对称连接的，神经元根据能量函数计算概率进行激活的网络结构
- 常用的是改进版本：限制玻尔兹曼机（RBM）
- 单层RBM结构如图：



- 可见层神经元**V**接受输入，
- 隐藏层神经元**H**得到特征。

无监督预训练

- RBM是生成式神经网络
  - 可见状态向量（输入向量）$\boldsymbol{v}$，隐藏状态变量h
  - RBM描述的是(v,h)的联合分布



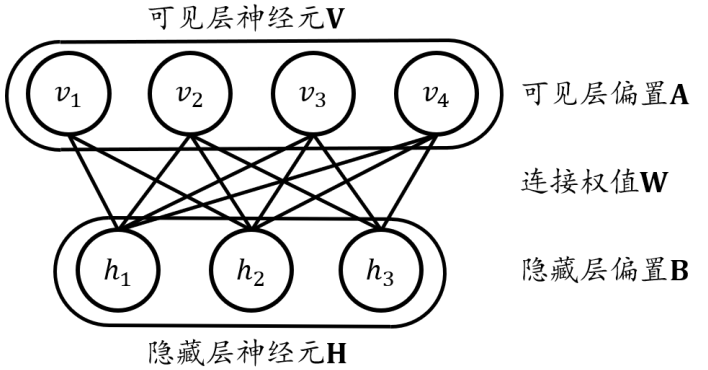
- 能量函数：
  - $E(\boldsymbol{v},\boldsymbol{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i w_{ij} h_j$

- 训练目标
  - 最小化RBM模拟的数据分布与数据实际分布之间的对比散度（contrastive divergence，CD）

玻尔兹曼机

- 逐层预训练方法，将多个预训练好的RBM堆叠起来就可以作为网络的初始化结果

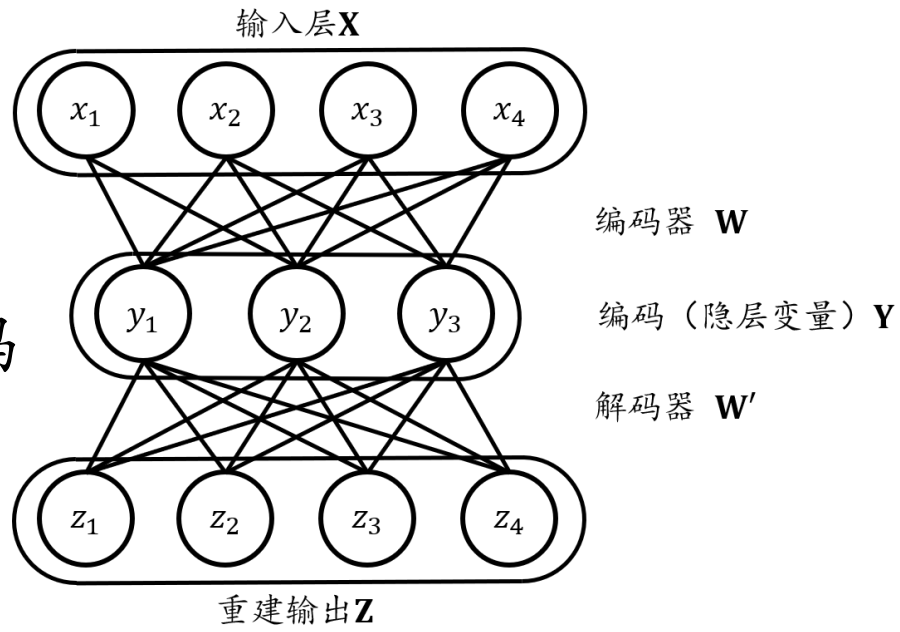
- 堆叠方式：
  - 将训练好的上一层输出作为下一层的输入
  - 将这个过程重复进行直到满足原始网络结构



自编码器

- 自编码器（Auto-encoder，AE）是无监督的全连接网络结构，其训练目标为重构误差最小化

- 单层AE的结构如图所示：
  - 编码器：高位数据转换成为低维特征编码
  - 解码器：还原原始数据



有监督预训练：迁移学习

主要思想：通过大量带标签的数据集，大幅减少网络收敛的训练时间

- ImageNet数据集

- 使用大型数据集预训练网络，可以使网络在其它任务上发挥更好的效果

站在巨人的肩膀上，复用已经得到的研究成果

迁移学习

- 使用预训练模型的方式：

（1）**直接作为特征提取网络**：即将网络直接用于数据的特征提取，将输出作为特征，根据任务目标进行后续的分类、回归等操作，

不再对这些预训练层进行进一步的学习，可以看作预训练模型“冻结（frozen）”了；

（2）**作为初始化模型进行微调**：部分或全部使用这个模型作为初始化模型，根据手头的数据对这个模型进行再次训练，这个过程被称为“精调（fine-tune）”。

根据**源领域与目标领域的关系**，以及我们所拥有的**带标签数据集大小**，我们可以通过不同的方式使用预训练模型。

四、实例

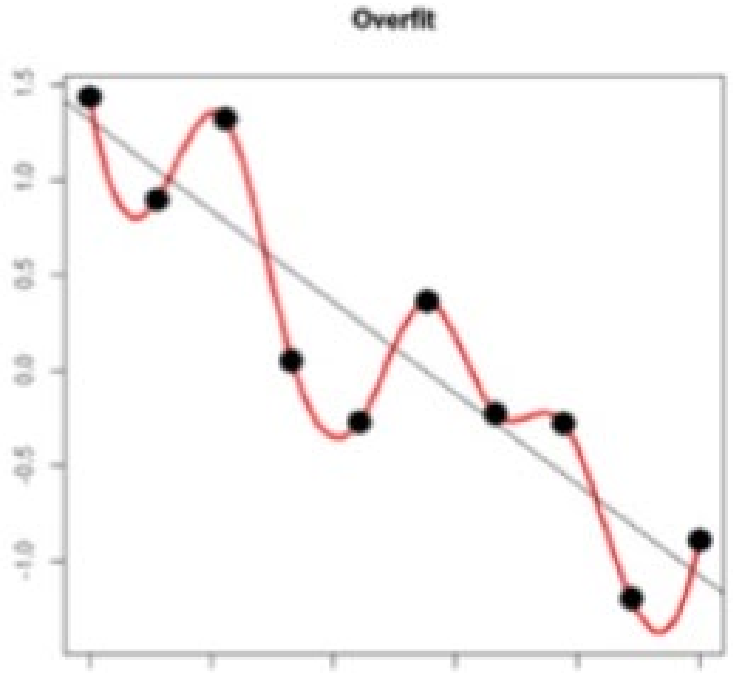
过拟合实例一

利用L1和L2正则化缓解过拟合现象:

过拟合的时候，拟合函数的系数（W）往往非常大，为什么?

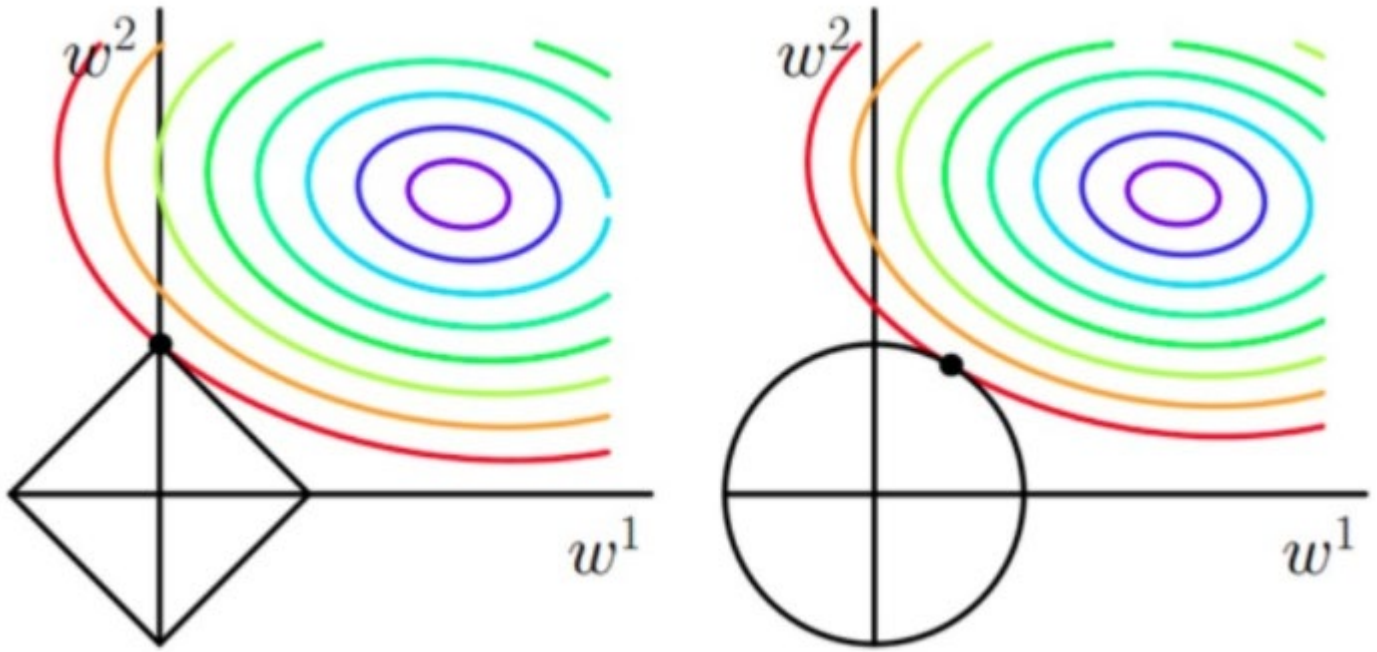
通过减小权重w的值来降低网络的复杂度。

以此缓解过拟合现象。



过拟合实例一

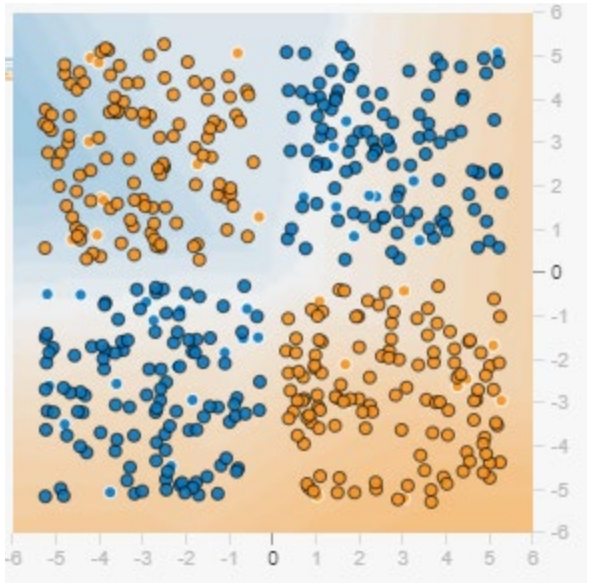
- L1和L2范数原理



- L1正则化使参数趋近坐标系（零值）。
- L2正则化使参数减小

过拟合实例一

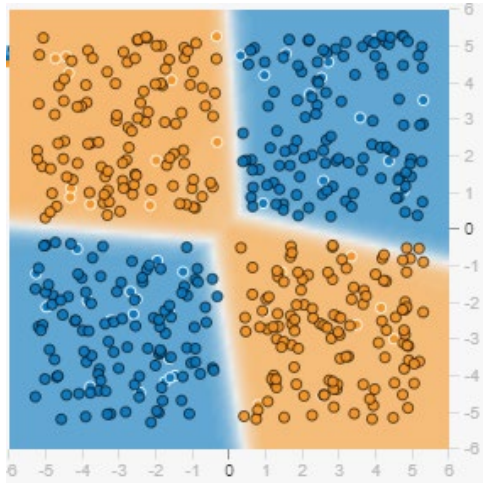
- 目前给定任务，二维空间中，点被分成了四个点集，代表了两类：



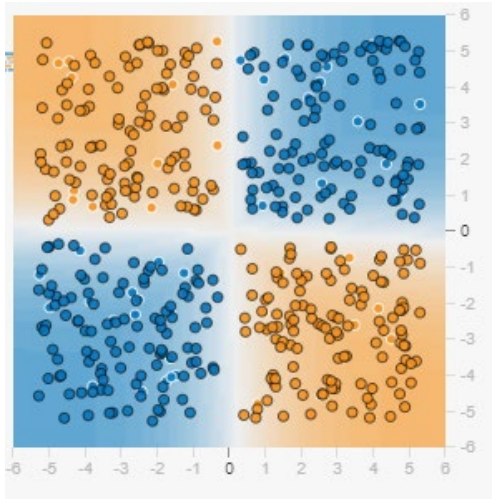
- 其中浅色小的点为训练集，深色粗的点为验证集，使用模型为具有一个隐含层，隐含单元为5的神经网络，激活函数为ReLU函数。
- 该任务中训练数据偏少，网络学到的分布容易偏离正确的分布。

过拟合实例一

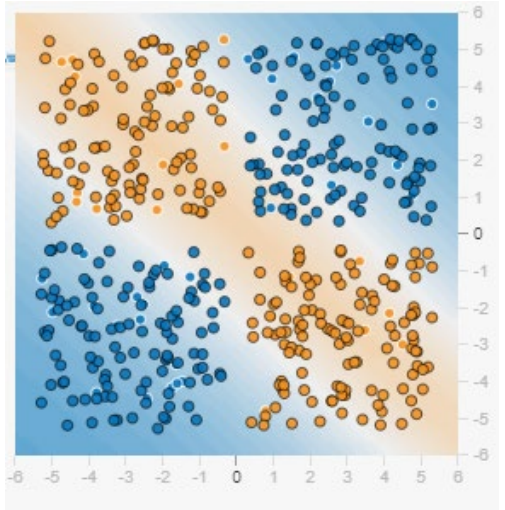
- 训练结果：



不采用正则化



L2正则化



L1正则化

- 训练后的权重（部分）：

None	-0.75	-0.85	0.87	0.34	0.75
L2	0.33	-0.25	0.33	-0.24	0.44
L1	0	0	-0.31	0.35	0

- 该实例可通过https://playground.tensorflow.org/自行验证和探讨。

BatchNorm实例一

这里展示BatchNorm在MINIST数据集上应用效果

（本例子来自论文：Ioffe S , Szegedy C . Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[J]. 2015.）

构建神经网络结构如下：

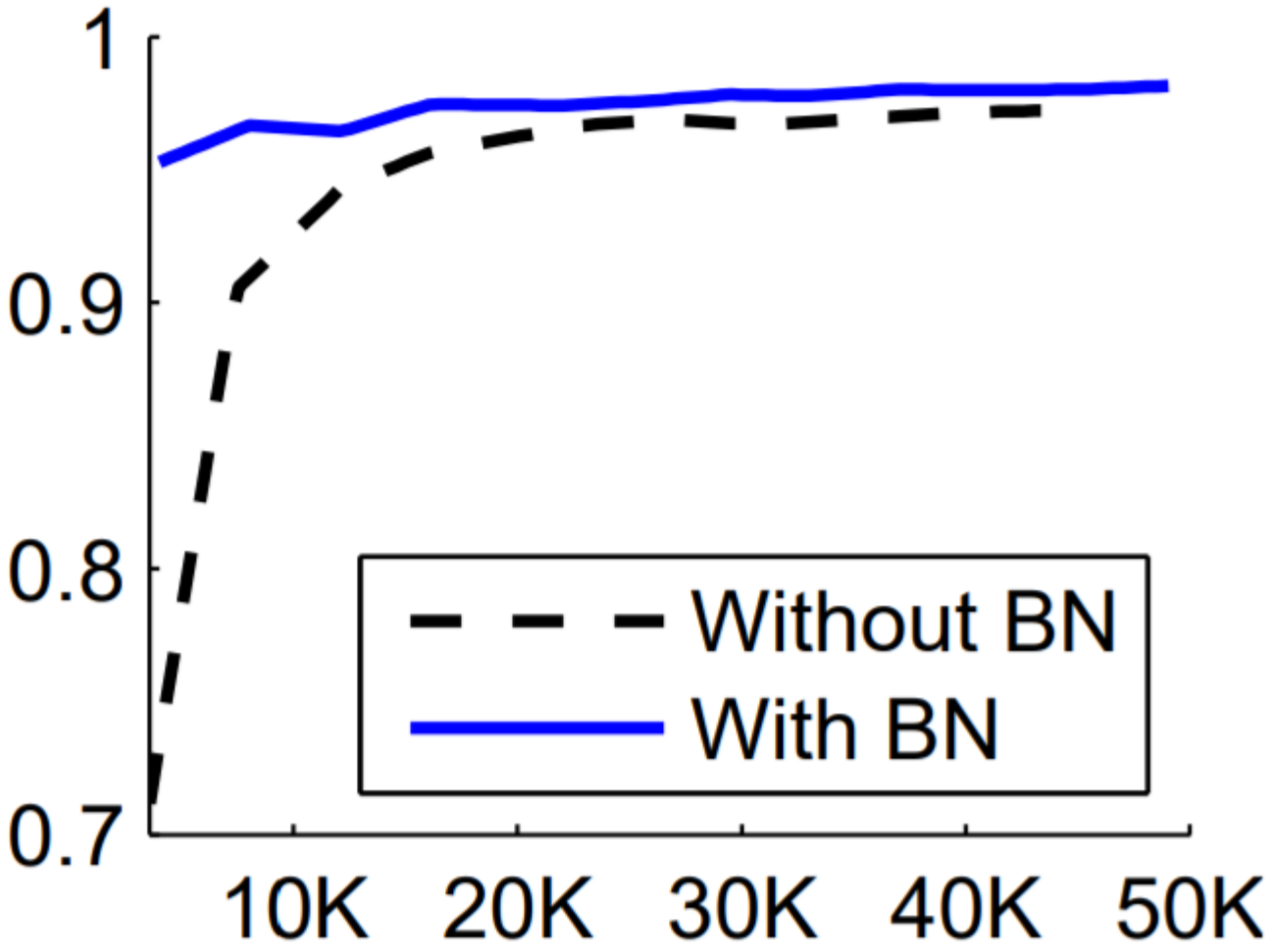
1. 基础版本：输入为28x28的图片。隐藏层为3个全连接层，每层包含100个神经元，使用sigmod激活函数。输出层是个包含10个神经元的全连接层，使用交叉熵作为损失函数

2. BatchNorm版本：在基础版本的基础上，每个隐藏层后面添加一个BatchNorm层。

实验效果

右图展示了测试准确率和训练步数之间的关系

可以看到，batchnorm能够有效加快网络收敛速度，并能够提高网络泛化性能。



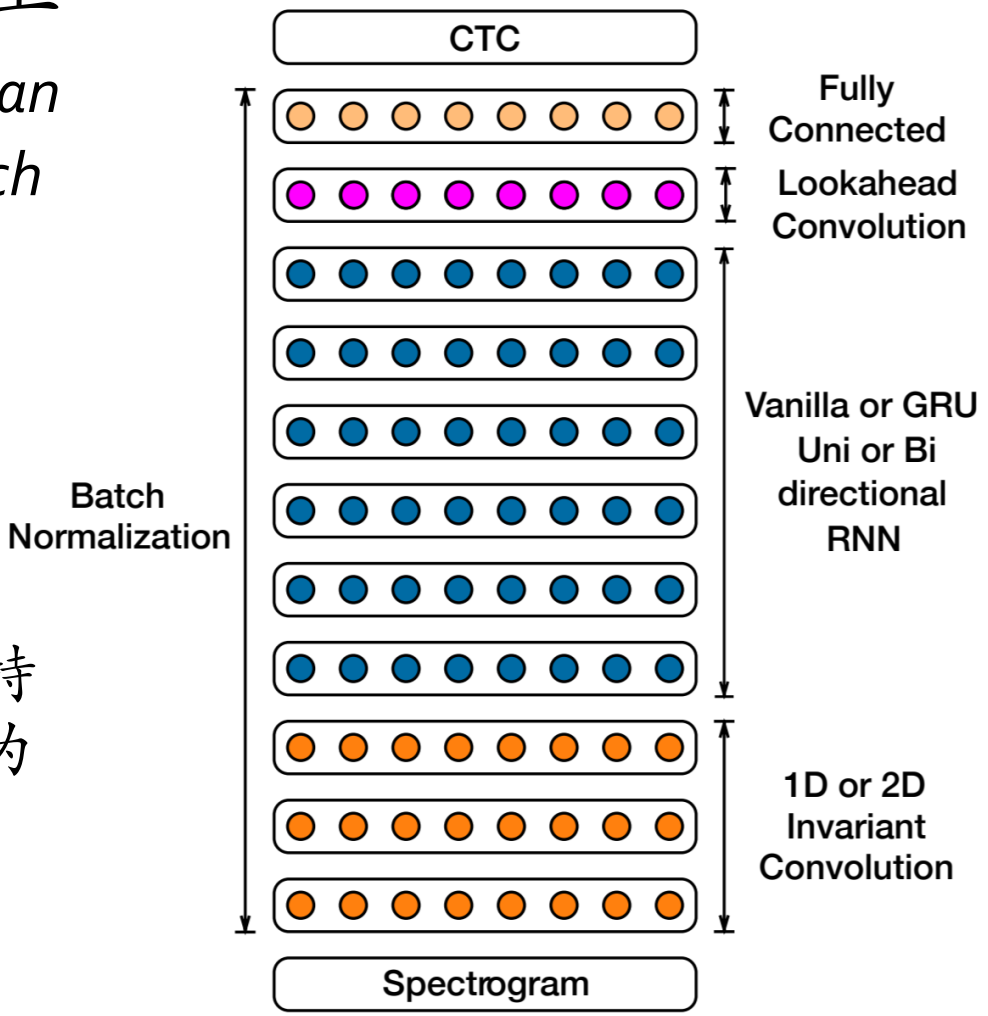
BatchNorm实例二

这里展示BatchNorm在语音识别任务上的效果（*实例来自：Amodei D , Ananthanarayanan S , Anubhai R , et al. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin[J]. Computer Science, 2015.*）

在该实例中，搭建了如右图所示的神经网络

基础版本：网络的输入在最下层，为语音的频谱特征。网络中间包含了可变数量的双向RNN层（标为蓝色）。输出层采用CTC损失函数。

BatchNorm版本：隐藏层后面都接上一BN层



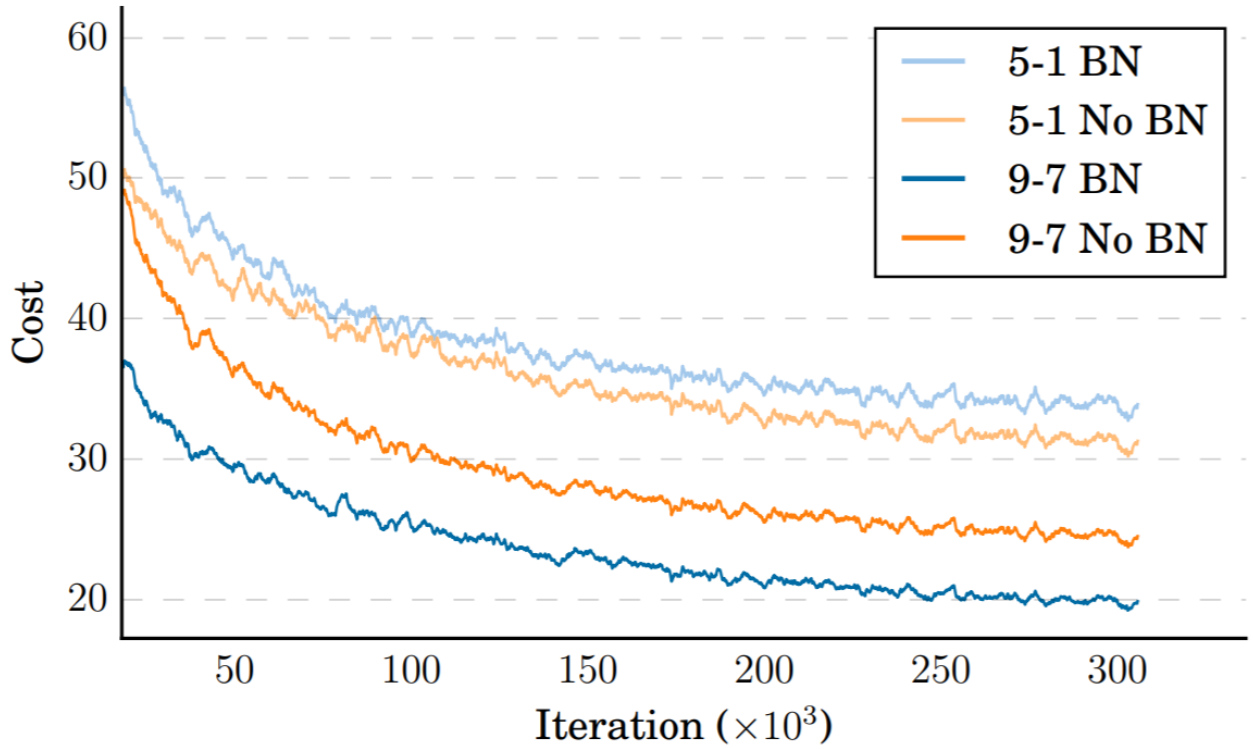
实验效果

右图展示了网络的训练损失函数和训练步数的关系

这里5-1指隐藏层总层数为5，其中双向RNN层数为1

9-7同理

可以看到：BatchNorm可以有效加快收敛速度并进一步降低训练损失。



五、作业

- Batch normalization的输出服从什么样的分布（指出分布中的具体参数）。
- Batch normalization为什么归一化后还有放缩（γ）和平移（β）？
- 目前有一批病人的身体数据（体重变化，血液指标等）和他们是否患有肺癌的真实标签，其中患肺癌的样本只占非常小的比例。数据直接送入一个神经网络中，求问应该使用什么样的初始化？数据中不同的特征数值差异过大，求问如何改进能够让网络更好地学习数据中地分布？

- 对神经网络损失函数添加正则化后形式如下，这里V为损失函数，R为正则化函数，给定输入xi，神经网络输出为f(xi)，yi为真实标记。

$$\min_f \sum_{i=1}^{n} V(f(x_i), y_i) + \lambda R(f)$$

  - （1）请说明参数λ的意义；
  - （2）参数λ的取值是越大越好还是越小越好，请结合playground（https://playground.tensorflow.org）上面的实例说明理由。
  - 注：在playground上，红框选中的区域用于选择正则化函数的种类以及设置参数λ的大小。