深度学习中的正则化

概览

1. 什么是正则化?
2. 参数范数惩罚。
3. 数据集增强
4. 多任务学习。
5. 提前终止。
6. Bagging和其它集成方法。
7. Dropout。
8. 其它正则化方法。

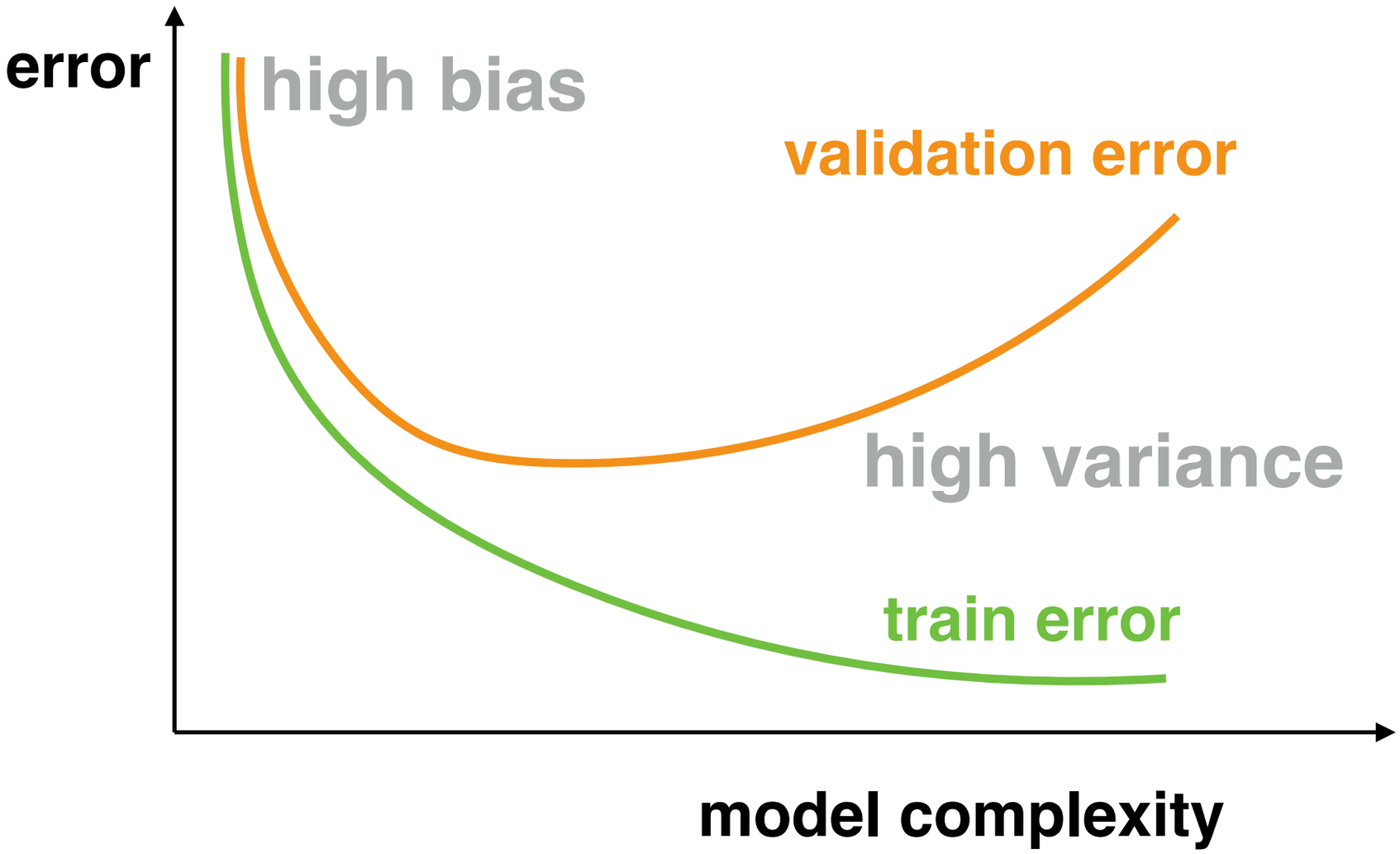
1. 什么是正则化?

什么是正则化

思考：是否模型在训练数据集上的误差越小就代表模型越优秀?

我们希望在训练集中模型误差较低的情况下，泛化误差越低越好。在机器学习中，很多策略显示地被设计来减少泛化误差，这些策略被统称为正则化。即正则化旨在降低泛化误差。

正则化



有效的正则化可以显著的减少方差，且不过度增加偏差。

并不是。对于复杂模型并不一定包含真实数据生成过程，真实的生成过程涉及模拟整个宇宙，所以难以甚至无法控制模型规模。

思考：控制模型的复杂度是否意味着我们需要找到一个合适规模的模型?

2. 参数范数惩罚

参数范数惩罚

参数范数惩罚，通过对目标代价函数J添加一个参数范数惩罚，限制模型的学习能力。正则化后的总体代价函数为：

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}),$$

其中 $\Omega(\boldsymbol{\theta})$ 表示惩罚项，

$\alpha \in [0, \infty)$ 表示惩罚项与标准代价函数J的相对贡献。

参数范数正则化注意事项

- 通常仅仅对神经网络中的连接权重进行范数惩罚。
- 正则化偏置值可能导致欠拟合。
- 对每一层神经元分别进行正则化时寻找超参数α的代价较大，通常仅设置全局的α。

2.1 L^2 参数范数惩罚

L^2 参数范数惩罚

L^2 参数范数惩罚，通常也称为权重衰减（weight decay），其中，

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{w}\|_2^2$$

即，总体代价函数为：

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2} \boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

参数更新规则为：

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha \boldsymbol{w} + \nabla_w J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})).$$

L^2 参数范数惩罚的特点

- L^2 正则化能对模型中不重要的参数施加较大惩罚力度。
- 超参数a的设置太大可能导致模型欠拟合。
- 超参数a的设置太小可能无法起到正则化效果。

练习

1. 使用TensorFlow构造二元线性回归模型，模型包含连接权重与偏置值，对连接权重使用 L^2 参数范数惩罚。

2. （作业）查找并学习TensorFlow高层库tf.layer中使用 L^2 参数范数惩罚的方法，并重新完成上面的“实验 1”。

主要使用方法：tf.contrib.layers.l2_regularizer, tf.contrib.layers.apply_regularization

2.2 L^1参数范数惩罚

L^1 参数范数惩罚

L^1 参数范数惩罚，也是常见的限制模型参数规模的方法，其中，

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

即，总体代价函数为：

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

参数更新规则为：

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha \text{sign}(\boldsymbol{w}) + \nabla_w J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})).$$

L^1 参数范数惩罚的特点

- 相比2范数参数惩罚，1范数参数惩罚可以使模型参数更加稀疏（此处的稀疏指最优值中一些参数为0）。

- 利用1范数参数惩罚可以使得参数稀疏的性质，可以进行**特征选择**，忽略不重要的特征，著名的LASSO回归就是一个例子。

c : $y = 0.5\ (x - 2)^2 + \sin(x)$

B = 极值点 (c, −2.96, 11.68)

→ (2.99, 0.64)

f : $y = 0.5\ (x - 2)^2 + \sin(x) + x^2$

C = 极值点 (f, −2.96, 11.68)

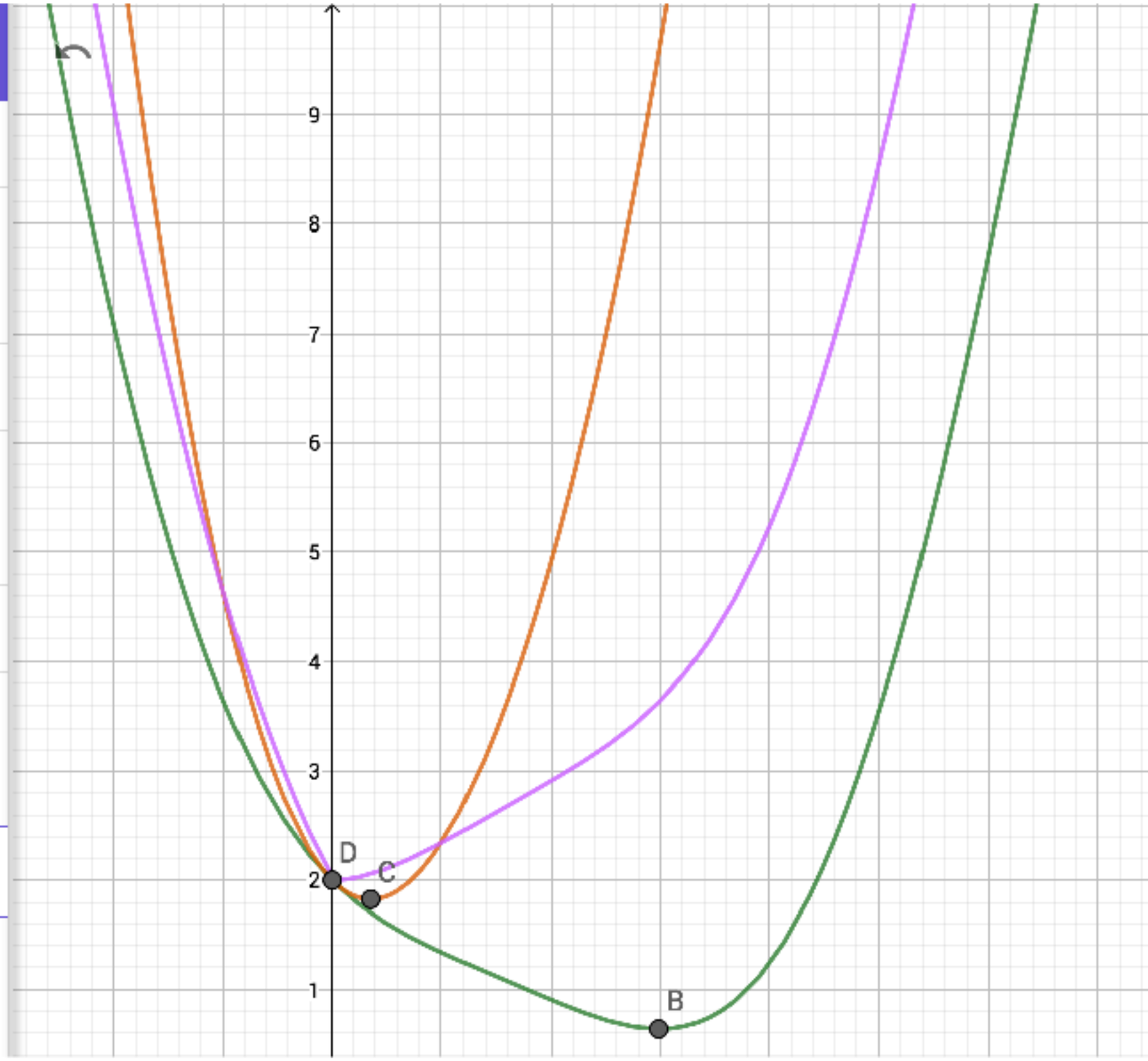
→ (0.35, 1.83)

g : $y = 0.5\ (x - 2)^2 + \sin(x) + |x|$

D = 极值点 (g, −2.96, 11.68)

→ (0, 2)

输入...



练习

1. 使用TensorFlow构造二元线性回归模型，模型包含连接权重与偏置值，对连接权重使用 L^1 参数范数惩罚。

2. （作业）查找并学习TensorFlow高层库tf.layer中使用 L^1 参数范数惩罚的方法，并重新完成上面的“实验 1”。

主要使用方法：tf.contrib.layers.l1_regularizer, tf.contrib.layers.apply_regularization

3. 数据集增强

数据集增强

更多的数据集可以提高模型泛化能力。

数据集获取方法：

- 搜集更多的训练数据。操作成本较高。
- 创建假数据并添加到训练集。适用范围较窄。

图像数据集增强方法

- 加入高斯噪声。
- 随机裁剪。
- 图片翻转（OCR任务中，要谨慎使用）。
- 亮度、对比度、色调、饱和度的调整。单通道图片仅可调节亮度、对比度。
- 其它。

4. 多任务学习

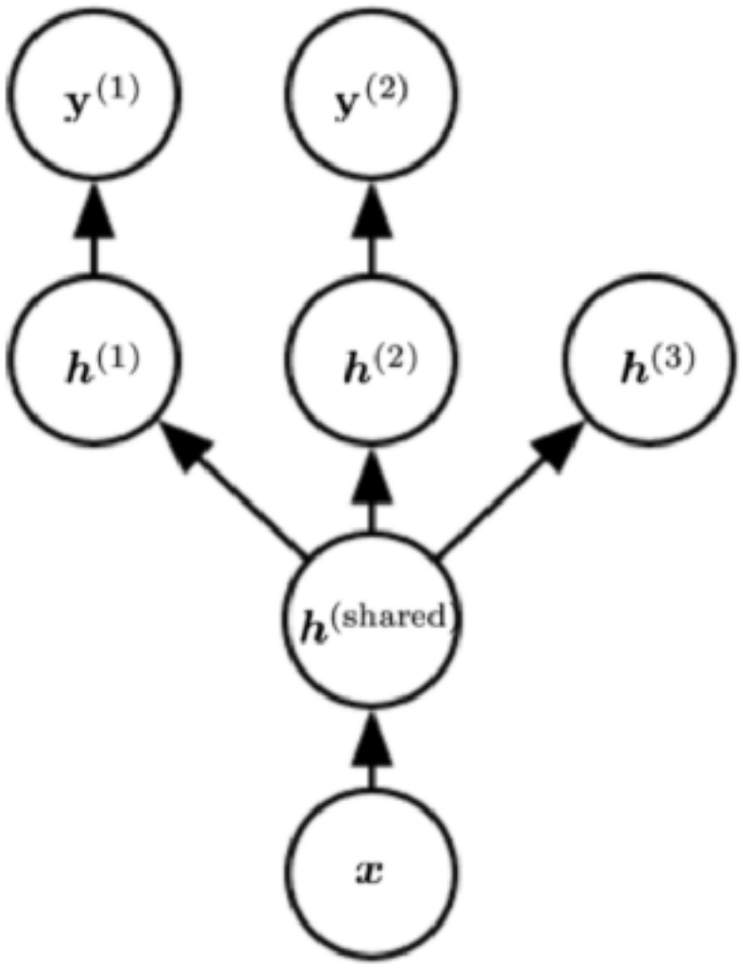
多任务学习

多任务学习（Multitask Learning，MTL）：利用共享表示并行训练多个相关任务，来提高泛化性能。

结构特点：

- 一个模型完成多个任务。
- 所有任务共享一部分模型参数。
- 具体任务拥有特定参数。

多任务学习



模型共享表示部分为： $h^{(shared)}$

具体任务的独特表示为： $h^{(1)}$ $h^{(2)}$ $h^{(3)}$

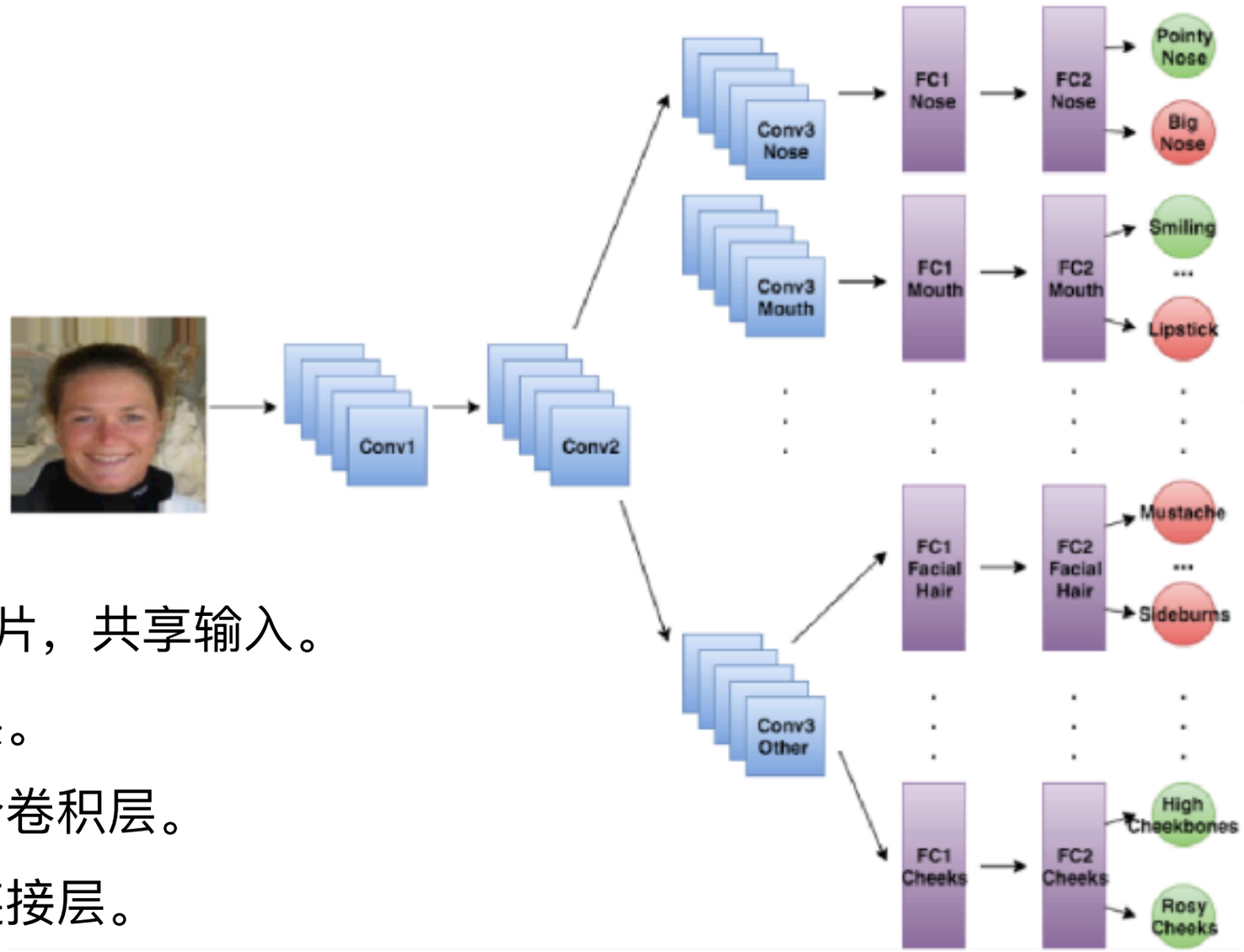
监督学习任务预测输出为： $y^{(1)}$ $y^{(2)}$

无监督学习任务为： $y^{(3)}$

注意：多任务学习在深度学习模型中可以以多种方式运行。此任务共享相同输入，但涉及不同目标随机变量。较为常见。

多任务学习例子

MCNN结构：



结构特点：

- 输入为227*227大小的图片，共享输入。
- 共享第一、第二个卷积层。
- 各个任务拥有特定第三个卷积层。
- 各个任务拥有特定的全连接层。
- 输出是否拥有某个特征，绿色表示有。

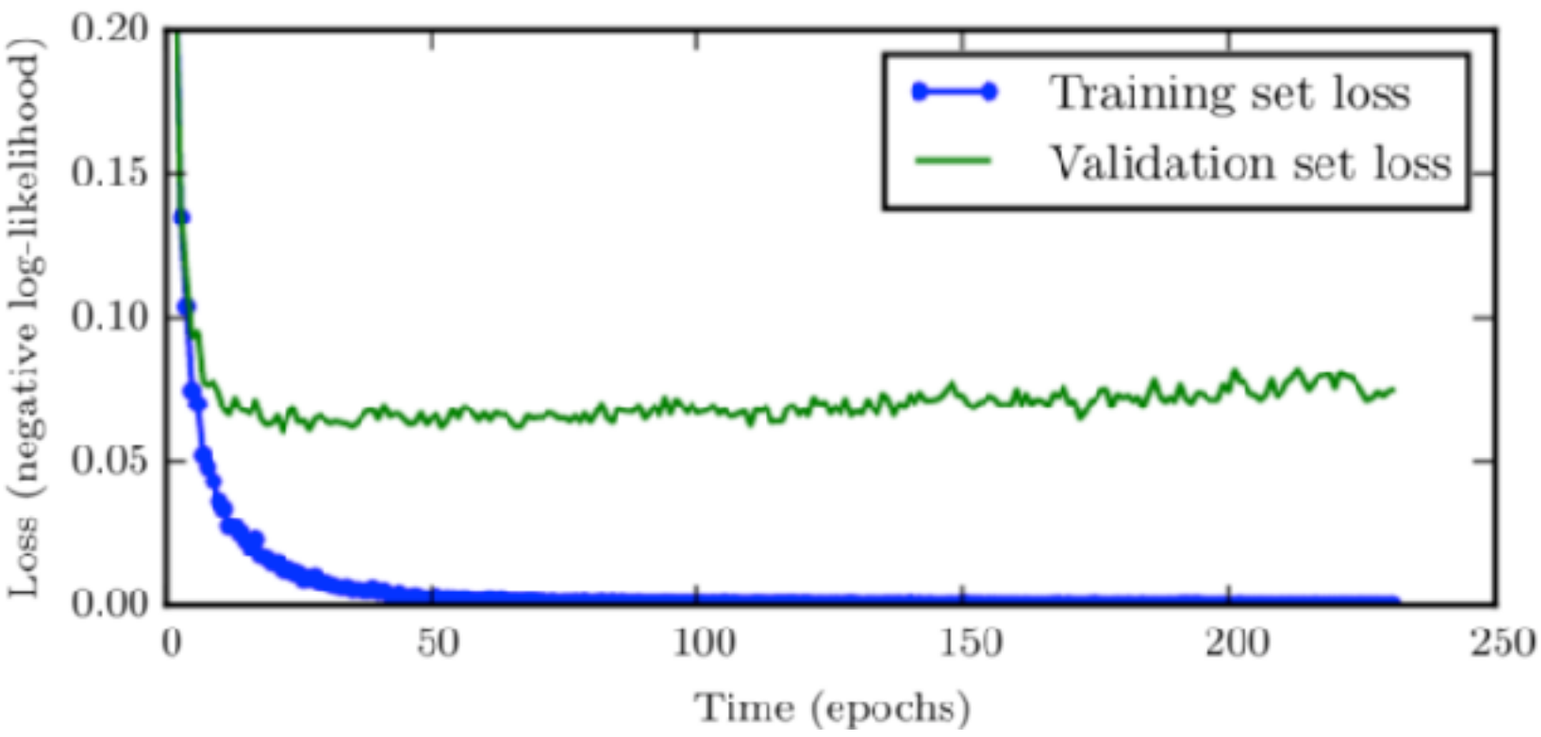
实验一

- 了解并读取BioID数据集；
- 构建神经网络模型，使用BioID数据集训练多任务（20关键点回归）模型；
- 在模型中加入参数范数正则化；
- （可选）增强数据集。

5. 提前终止

提前终止

提前终止（early stopping），当验证集误差不再降低时（准备再次上升之前）就证明当前训练方法下模型的泛化能力已经达到最大（或接近最大），此时需要终止训练。再接着训练，可能会降低训练集误差，但验证集误差反而可能升高。



过度训练，几乎都会出现过拟合，从而降低泛化能力。所以**提前终止是最常用的正则化方法**。

提前终止算法

提前终止算法中的“提前”需要使用验证集对当前算法的表现进行评估，当观察到p次坏的表现（验证集代价连续增大p次）之后，就终止训练。并使用p次之前

右图中：

n 表现评估间隔步数；

i 表示最终训练步数；

j 表示第j次看到坏表现；

v 表示验证集代价。

令 n 为评估间隔的步数。
令 p 为“耐心 (patience)”，即观察到较坏的验证集表现 p 次后终止。
令 $\boldsymbol{\theta}_o$ 为初始参数。
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  运行训练算法 n 步，更新 $\boldsymbol{\theta}$ 。
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
最佳参数为 $\boldsymbol{\theta}^*$，最佳训练步数为 i^*

提前终止的优缺点

- 🙂 节约了计算能力，避免了不必要的额外计算。
- 🙂 正则化方法简便、常用。
- ☹️ 需要从训练集中抽出一部分数据作为验证集。
- ☹️ 为高效使用数据集中的验证集，会付出一些代价。

有效利用验证集的两种策略

策略一：首先使用验证集确定最佳训练步数；然后再次初始化模型，并使用所有训练数据再次训练，训练的步数为上一次确定的最佳步数。

缺点：

1. 训练2次、计算代价大。
2. 由于训练集变大，第二轮训练会更多次的更新参数。

策略二：保持第一轮训练获得的参数，并记录第一轮结束后训练集的平均代价。接着使用全部的数据集继续训练，直到在验证集上的平均代价低于记录的目标值。

缺点：

1. 验证集的目标不一定能达到之前的目标值，即不能保证训练终止。
2. 效果不明显。

策略—算法

令 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 为训练集。

将 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 分别分割为 $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ 和 $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$。

从随机 $\boldsymbol{\theta}$ 开始，使用 $\boldsymbol{X}^{(\text{subtrain})}$ 和 $\boldsymbol{y}^{(\text{subtrain})}$ 作为训练集，$\boldsymbol{X}^{(\text{valid})}$ 和 $\boldsymbol{y}^{(\text{valid})}$ 作为验证集，运行（提前终止算法）。这将返回最佳训练步数 i^*。

将 $\boldsymbol{\theta}$ 再次设为随机值。

在 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 上训练 i^* 步。

策略二算法

令 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 为训练集。

将 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 分别分割为 $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ 和 $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$。

从随机 $\boldsymbol{\theta}$ 开始，使用 $\boldsymbol{X}^{(\text{subtrain})}$ 和 $\boldsymbol{y}^{(\text{subtrain})}$ 作为训练集，$\boldsymbol{X}^{(\text{valid})}$ 和 $\boldsymbol{y}^{(\text{valid})}$ 作为验证集，运行（提前终止算法）。这会更新 $\boldsymbol{\theta}$。

$\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{subtrain})})$

while $J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{valid})}, \boldsymbol{y}^{(\text{valid})}) > \epsilon$ **do**

　在 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 上训练 n 步。

end while

实验二

- 将BioID数据集以8:1:1的比例划分为训练集、验证集、测试集。

- 改造实验一中的模型训练方法，加入提前终止算法，保存泛化能力最佳的模型参数。

- （作业）加入策略一或者策略二中的高效利用验证集的方法，

6. bagging和其它集成方法

Bagging 算法简述

Bagging（bootstrap aggregating）是通过结合几个模型降低泛化误差的技术(Breiman, 1994) 。主要想法是分别训练几个不同的模型，然后让所有模型表决测试样本的输出，也被称之为**模型平均（model average）**。

Bagging 算法流程

算法：Bagging算法流程

令 D 为训练集，训练集大小为 m 。

从 D 中均匀有放回的选出 k 个大小为 n 的子集记为 D_i 。

在这 k 个训练集上使用分类、回归算法，得到 k 个模型。

使用 k 个模型预测结果，通过取平均值、取多数票等方法，得到最终结果。

当m=n时，构造出的数据集的子集中大概有原始数据集中2/3的实例，这意味着每个子集中以高概率缺少一些来自原始数据集的例子，还包含若干重复例子。

注意：预测时，回归模型取平均值；分类模型取多数票。

思考：Bagging 算法为什么会奏效?

Bagging 算法

假设我们有 k 个回归模型。假设每个模型在每个例子上的误差是 ε_i，这个误差服从零均值方差为 $\mathrm{E}\left[\varepsilon_i^2\right]=v$ 且协方差为 $\mathrm{E}\left[\varepsilon_i \varepsilon_j\right]=c$ 的多维正态分布。通过所有集成模型的平均预测所得误差是 $\frac{1}{k}\sum_i \varepsilon_i$。集成预测器平方误差的期望是 $\mathrm{E}\left[\left(\frac{1}{k}\sum_i \varepsilon_i\right)^2\right]$。

$$\mathrm{E}\left[\left(\frac{1}{k}\sum_i \varepsilon_i\right)^2\right] = \mathrm{E}\left[\frac{1}{k^2}\left(\sum_i \varepsilon_i\right)^2\right] = \frac{1}{k^2}\mathrm{E}\left[\left(\sum_i \varepsilon_i\right)^2\right]$$

$$= \frac{1}{k^2}\mathrm{E}\left[\sum\nolimits_i \left(\varepsilon_i^2 + \sum\nolimits_{j\neq i} \varepsilon_i \varepsilon_j\right)\right]$$

$$= \frac{1}{k^2}\mathrm{E}\left[\sum\nolimits_i \varepsilon_i^2 + \sum\nolimits_i \sum\nolimits_{j\neq i} \varepsilon_i \varepsilon_j\right]$$

$$= \frac{1}{k^2}\mathrm{E}\left[\sum\nolimits_i \varepsilon_i^2\right] + \frac{1}{k^2}\mathrm{E}\left[\sum\nolimits_i \sum\nolimits_{j\neq i} \varepsilon_i \varepsilon_j\right]$$

$$= \frac{1}{k^2}\sum\nolimits_i \mathrm{E}\left[\varepsilon_i^2\right] + \frac{1}{k^2}\sum\nolimits_i \sum\nolimits_{j\neq i} \mathrm{E}\left[\varepsilon_i \varepsilon_j\right]$$

$$= \frac{1}{k}v + \frac{k-1}{k}c$$

bagging 算法

当 $c=v$ ，即误差完全相关，每个模型输出一样时：

$$\mathrm{E}\left[\left(\frac{1}{k}\sum_i \varepsilon_i\right)^2\right]=\frac{1}{k}v+\frac{k-1}{k}c=v$$ 此时，模型平均没有任何帮助。

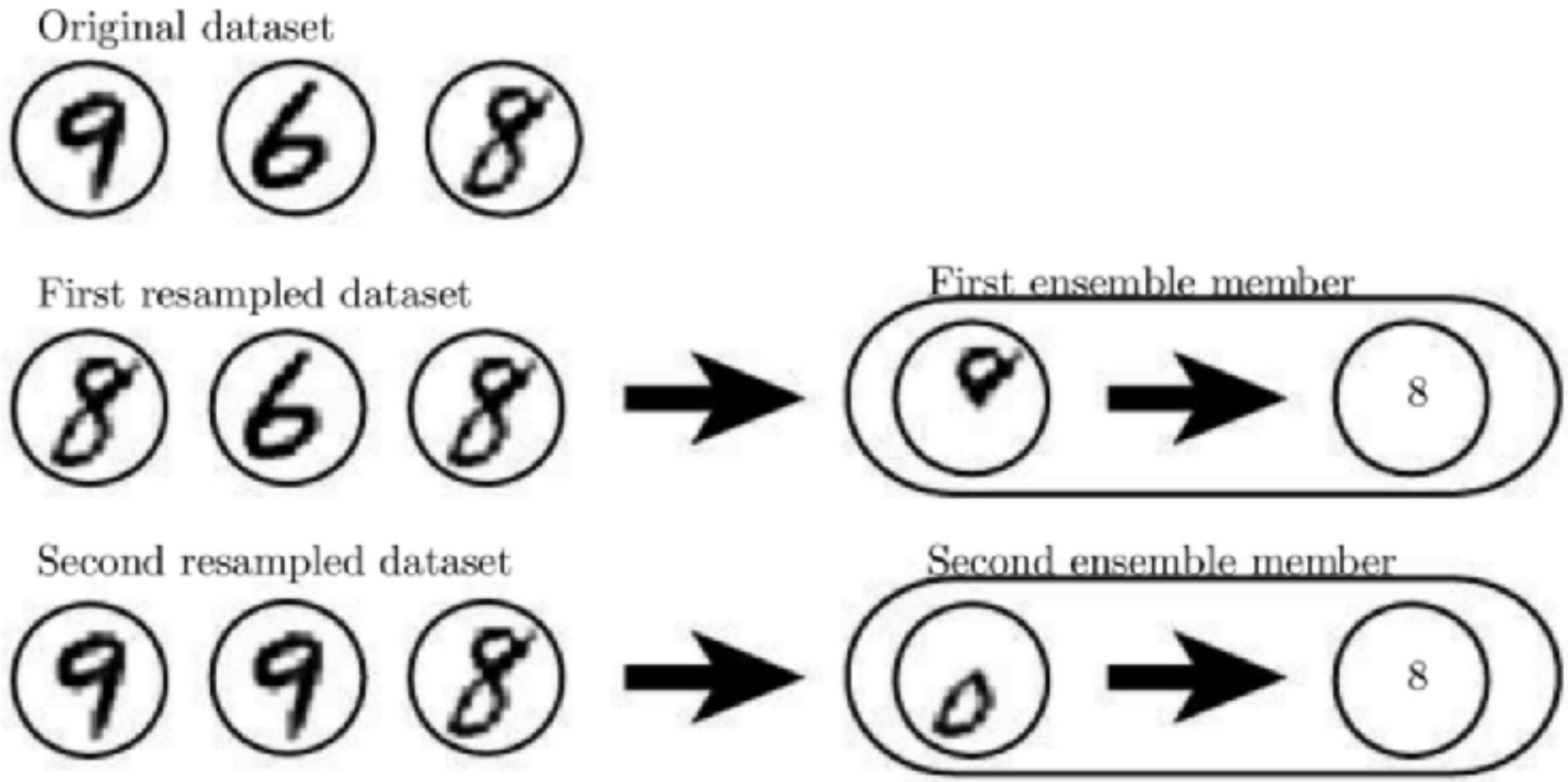
当 $c=0$ ，即误差完全不相关，每个模型输出与其它模型完全无关时：

$$\mathrm{E}\left[\left(\frac{1}{k}\sum_i \varepsilon_i\right)^2\right]=\frac{1}{k}v+\frac{k-1}{k}c=\frac{1}{k}v$$ 此时，模型平均线性减少误差。

NN构建不同模型的方法

- 使用不完全相同的数据集训练模型；
- 使用不完全相同的算法和代价函数；
- 有差异的初始化方法；
- 小批量的随机选择；
- 超参数差异；
- 其它

Bagging示例



在原始数据集（包含9，6，8三个样本）上训练数字8的检测。有放回的采样两个子数据集。第一个子数据集不包含9，第二个子数据集不包含6。使用这两个数据集训练模型，最终第一个模型通过检测图像上部是否有环结构来判断是否为8，第二个模型通过检测图像下部是否有环结构来判断是否为8。平均他们的输出，可以得到鲁棒的检测器（两个模型都认为是8才输出8）。

Bagging应用在NN中优缺点

- 模型平均是一个减少泛化误差的非常强大可靠的方法，任何机器学习算法都可以从模型平均中大幅获取增益；

- Bagging算法的性能提升是以增加计算和存储为代价，通常机器学习比赛中的取胜算法都是是以超过几十种模型平均的方法。例如Netflix Grand Prize(Koren, 2009)；

- 大规模的NN进行集成需要花费很多运行时间和内存，通常我们只能集成五至十个NN。

实验三

1. 从实验二中BioID的训练集中有放回的随机抽样，生成3-5个新的训练集。

2. 使用新的训练集训练3-5个简单的CNN。

3. 使用多个新模型投票进行预测，计算其在测试集上的正确率。

了解其它集成方法

不是所有构建集成的技术都是为了让集成模型比单一模型更加正则化。例如，一种被称为**Boosting**的技术构建比单个模型容量更高的集成模型。通过向集成逐步添加神经网络，Boosting已经被应用于构建神经网络的集成。通过逐渐增加神经网络的隐藏单元，Boosting也可以将单个神经网络解释为一个集成。

7. Dropout

7.1 Dropout简介

Dropout (Srivastava *et al.*, 2014) 提供了正则化一大类模型的方法，计算方便，但功能强大。具体而言在训练神经网络时随机丢弃一部分神经元，在训练完毕时使用全部神经元进行预测（需要对使用Dropout的相关部分进行处理）。Dropout可以被看做集成大量深层神经网络的实用Bagging方法。

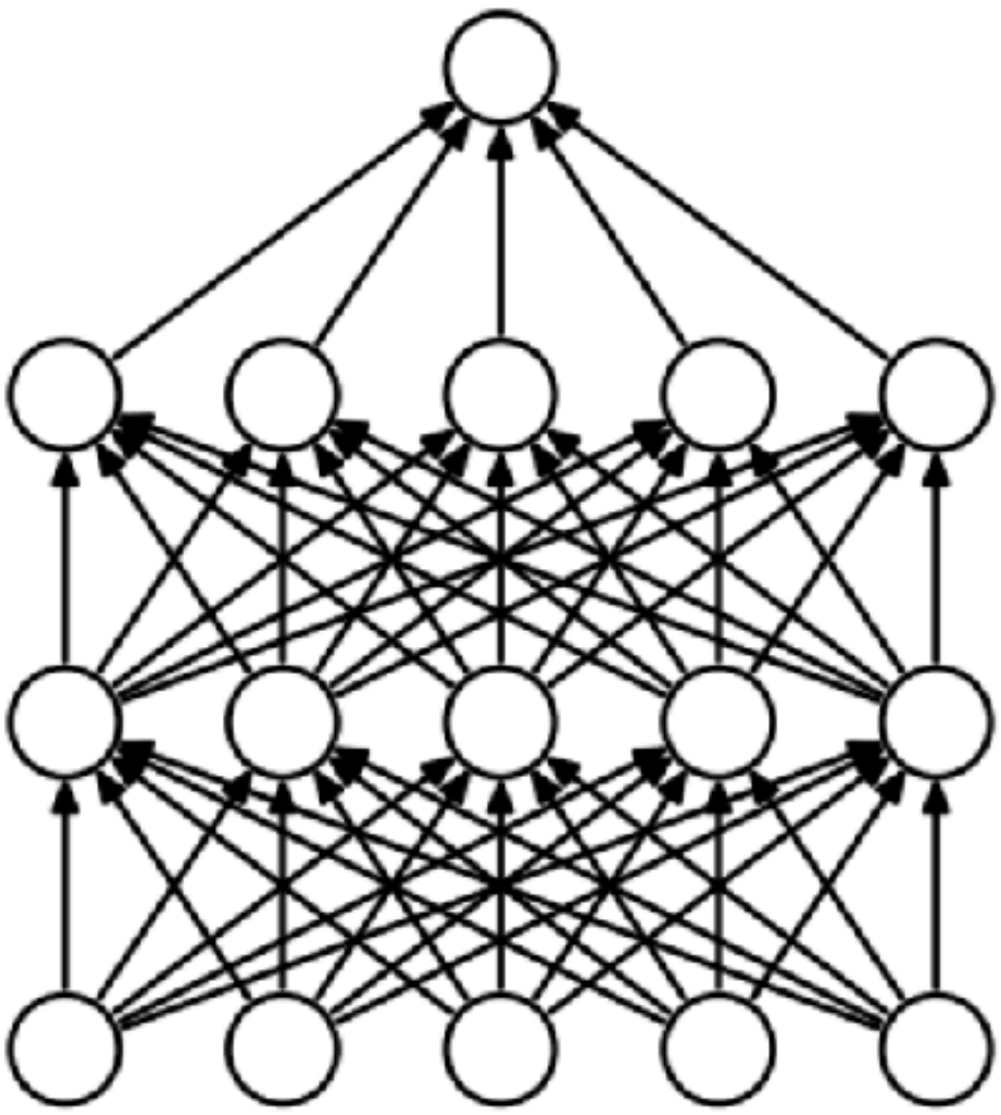
7.2 Dropout动机与背景

- 自然界中，（有性生殖中）基因的混合比（无性生殖中）基因的直接继承更具竞争力，虽然这违反了知觉。

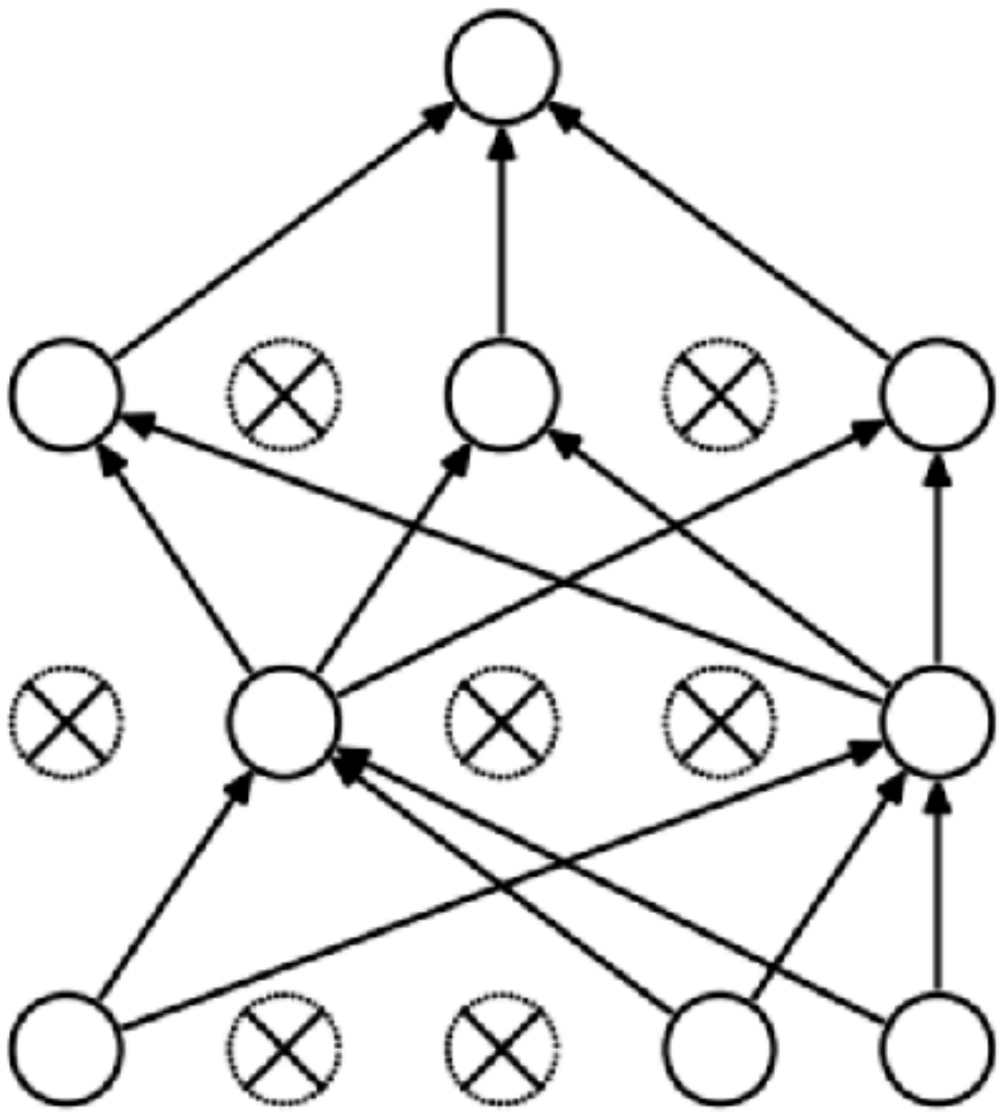
- 传统的ANN中，每一个神经元都强依赖于前面层的所有神经元，任何一个神经元输出的微小改变，都有可能引发之后所有神经元的巨大改变，即神经元对输入不够鲁棒。

7.3 Dropout 训练时

标准网络与Dropout网络对照

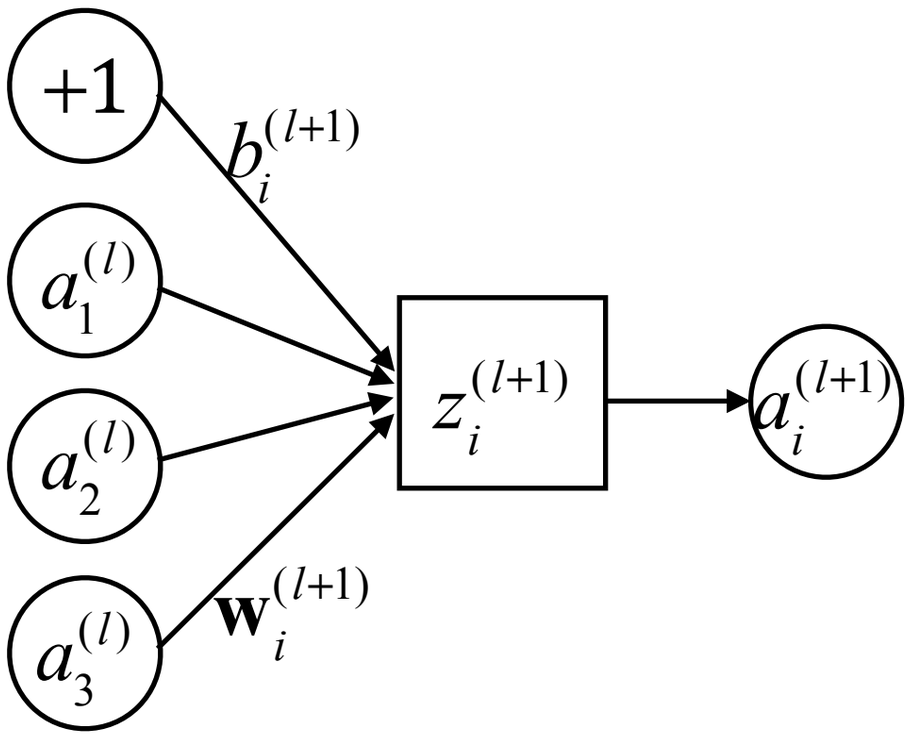


(a) Standard Neural Net

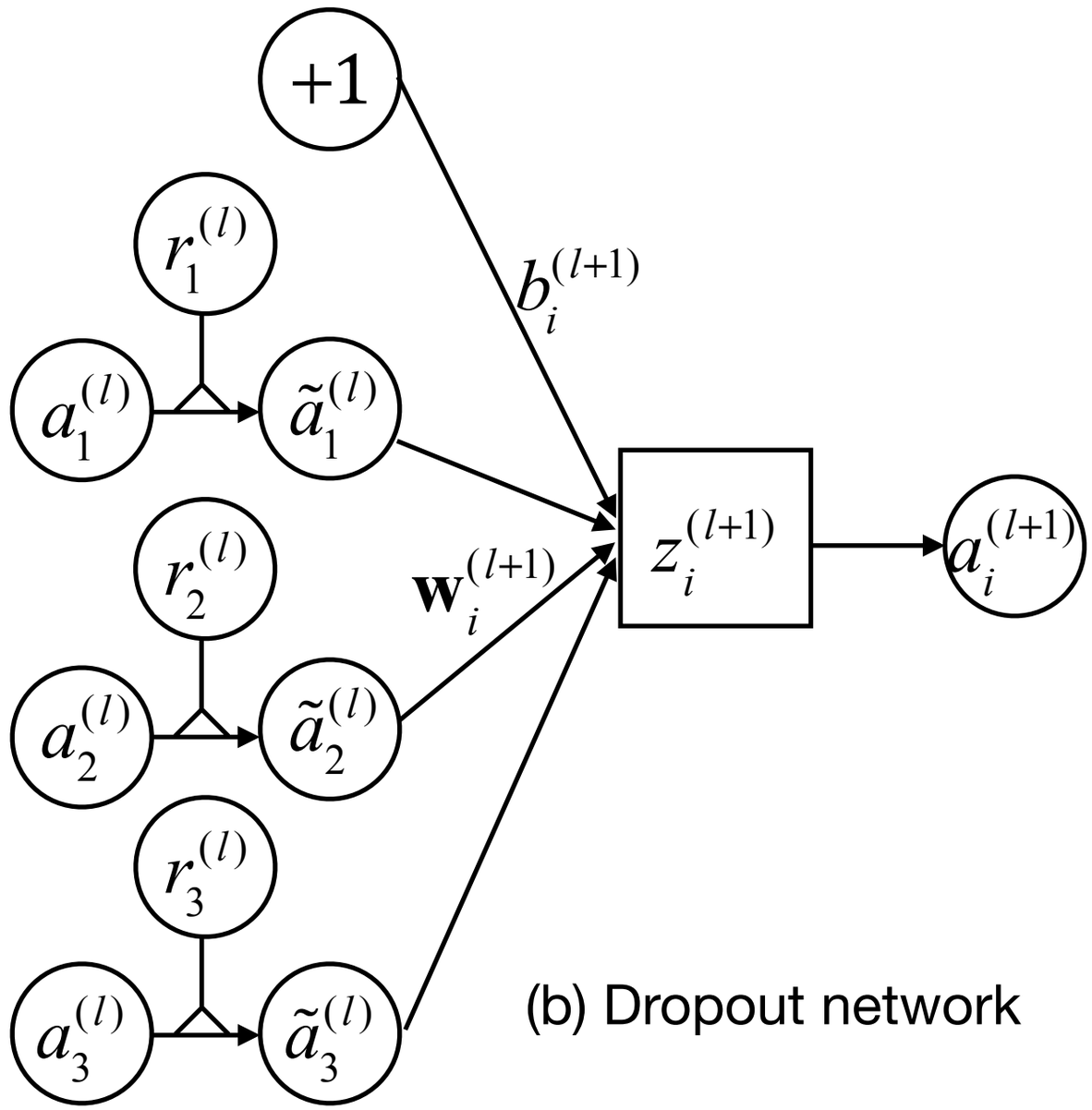


(b) After applying dropout.

标准网络与Dropout网络对照



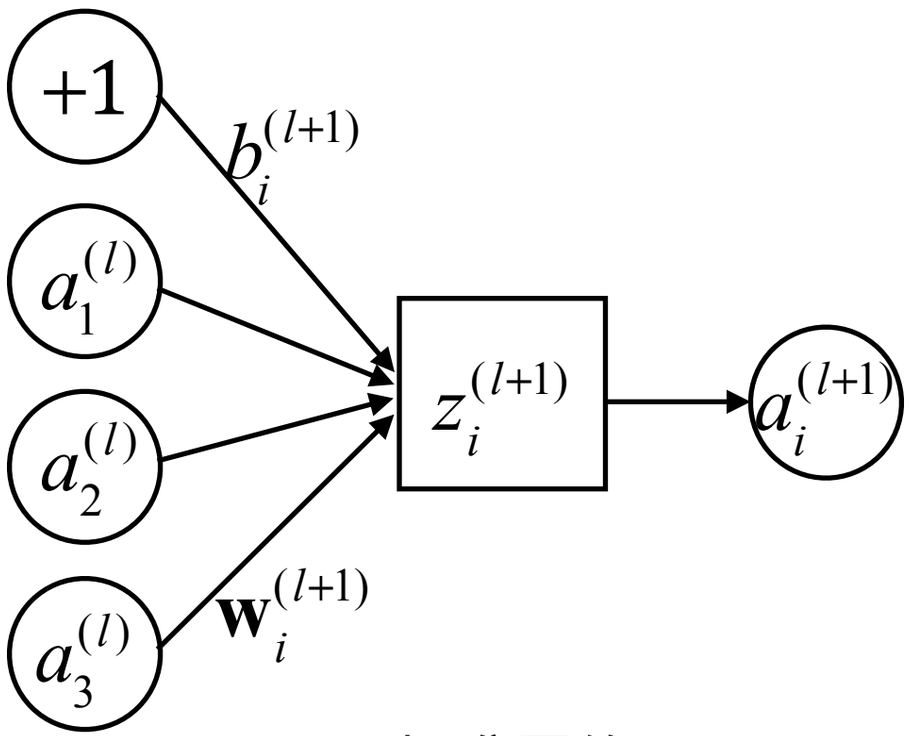
(a) Standard network



(b) Dropout network

左图表示标准网络，右图表示使用Dropout的网络。

标准网络的前向传播



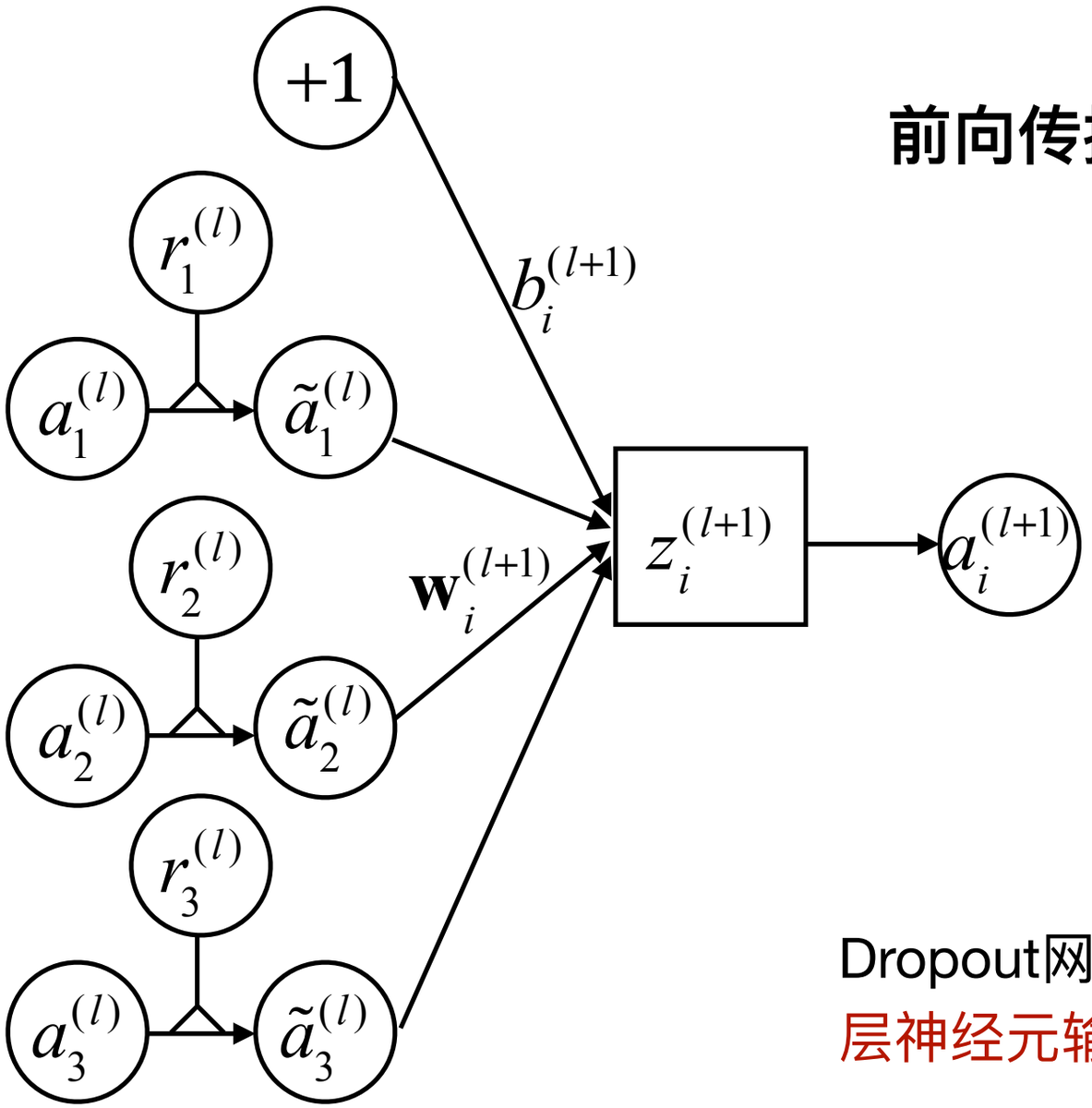
标准网络

前向传播：

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)}\mathbf{a}^{(l)} + b_i^{(l+1)}$$

$$a_i^{(l+1)} = \sigma(z_i^{(l+1)})$$

Dropout网络前向传播



前向传播： $r_j^{(l)} \sim \text{Bernoulli}(p)$

$$\tilde{\mathbf{a}}^{(l)} = \mathbf{r}^{(l)} * \mathbf{a}^{(l)}$$

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \tilde{\mathbf{a}}^{(l)} + b_i^{(l+1)}$$

$$a_i^{(l+1)} = \sigma(z_i^{(l+1)})$$

Dropout网络训练时，随机的使一部分隐藏层或输入层神经元输出变为0。

思考：Dropout网络反向传播时会发生什么?

使用小批量梯度下降训练

算法：Dropout网络使用小批量梯度下降法训练流程

设置 keep_prob 为神经元保留概率。

while 训练未完成条件：

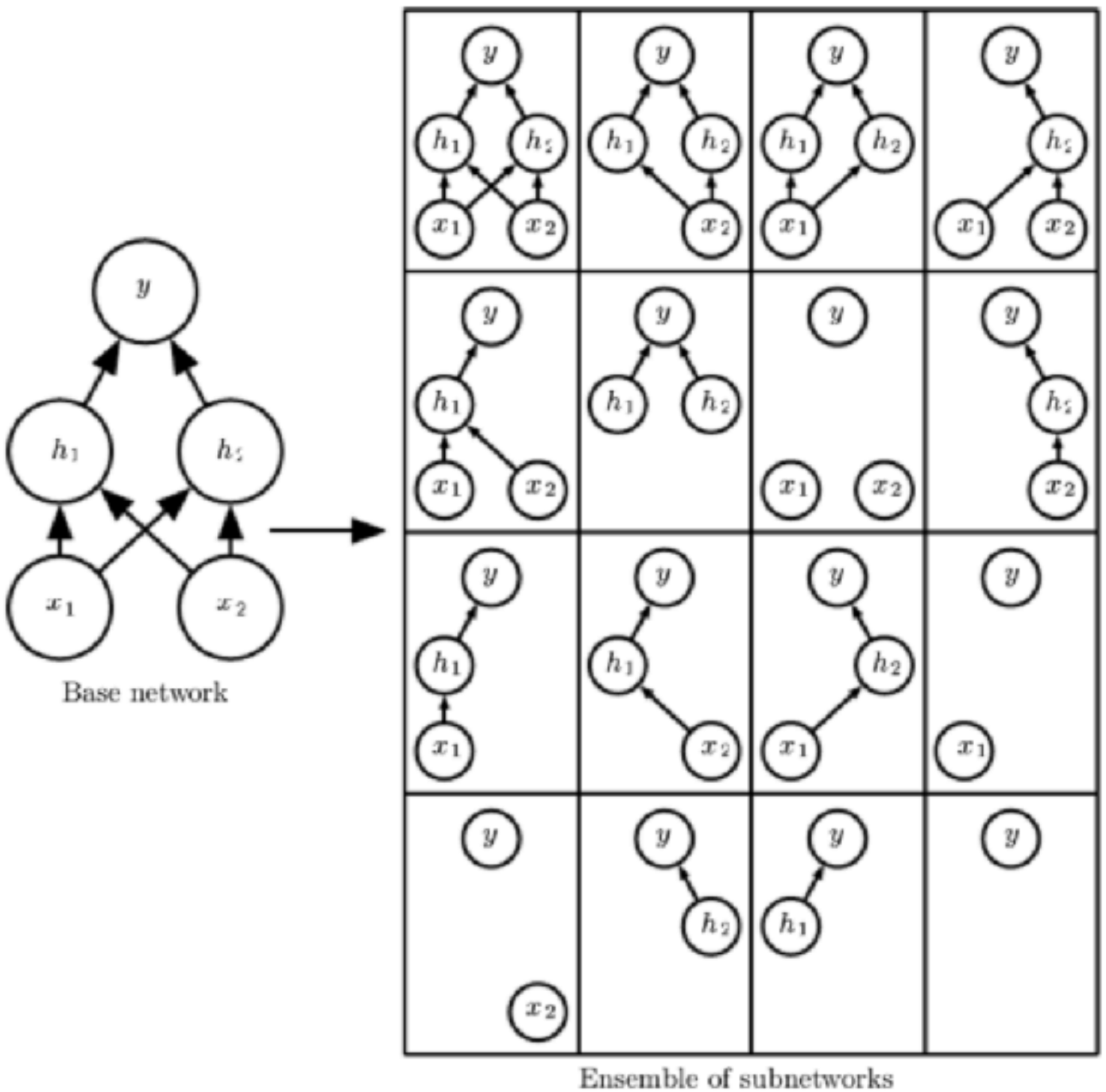
根据 keep_prob 生成神经网络掩码 **r** 。

获取小批量训练数据执行一次训练。

end while

通常输入层保留输出的概率为80%，隐藏层保留输出的概率为50%。

Dropout训练的特点与意义



- 每次训练都从基础网络中抽出一个子网络；
- 输出层不丢弃单元；
- 共有2^n中可能的子网络（n表示除输出层以外的神经元数量）；
- 在较大的规模的神经网络中丢弃所有从输入到输出的可能路径的概率变小；
- 当训练完成时，等价于训练好了2^n个共享参数的子模型。

思考：在上面的例子中，当keep_prob设置为50%且掩码生成服从均匀分布时：

1. 丢弃所有神经元（不包括输出层）的可能性存在吗？
2. 如果存在，其概率有多大?
3. 丢弃2个神经元的概率有多大?

7.4 Dropout 预测时

Dropout训练时的特点

- 随机使用掩码来改变神经网络的结构；
- 训练完成的神经网络，本质上主要训练出来的是概率较大的子网络；
- 子网络之间共享参数；
- 可以使用某一个子网络进行预测（思考使用基础网络可以吗？）。

思考：结合Bagging集成方法，思考是否可以通过一些方法提高Dropout网络的性能?

7.4.1 像Bagging一样集成

像Bagging一样集成

Bagging集成必须根据**所有成员**的累积投票做一个预测，即**推断**。在Bagging下，每个模型 i 产生一个概率分布 $p^{(i)}(y \mid \boldsymbol{x})$。集成预测由这些分布的算术平均值给出：

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y \mid \boldsymbol{x})$$

在Dropout下，通过掩码 $\boldsymbol{r}$ 定义每个子模型的概率分布 $p(y \mid \boldsymbol{x}, \boldsymbol{r})$ 。所有的掩码的算术平均为：

$$\sum_{\boldsymbol{r}} p(\boldsymbol{r}) p(y \mid \boldsymbol{x}, \boldsymbol{r})$$

其中 $p(\boldsymbol{r})$ 是训练时采样 $\boldsymbol{r}$ 的概率分布。

像Bagging一样集成

式子 $\sum_{\boldsymbol{r}} p(\boldsymbol{r})p(y\,|\,\boldsymbol{x},\boldsymbol{r})$ 包含多达指数级的项，并且目前没有可以简化的形式。

通常，我们可以通过采样近似推断，即平均许多掩码的输出。一般的10-20个掩码就足以获得不错的表现。这样做的缺点是：需要很多次的前向传播才能完成。

像Bagging一样集成

例子：一个2分类模型的输入层以keep_prob=0.8，隐藏层以keep_prob=0.5的比例使用Dropout，并在测试时，得到了以下五组结果：

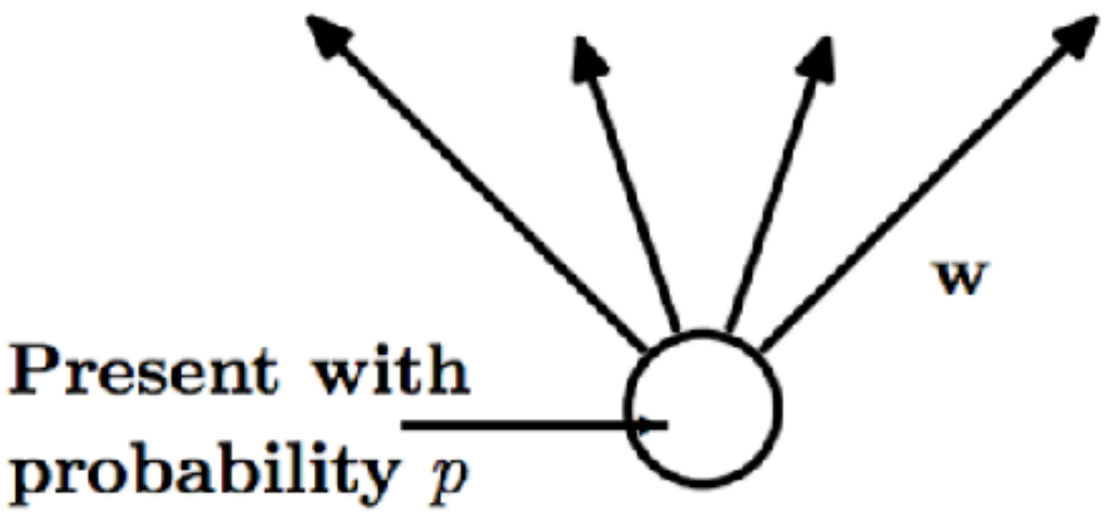
1. [0.1, 0.9]
2. [0.2, 0.8]
3. [0.6, 0.4]
4. [0.15, 0.85]
5. [0.5, 0.5]

则，模型最终的预测每个类别的概率为多少？预测类别是哪个？

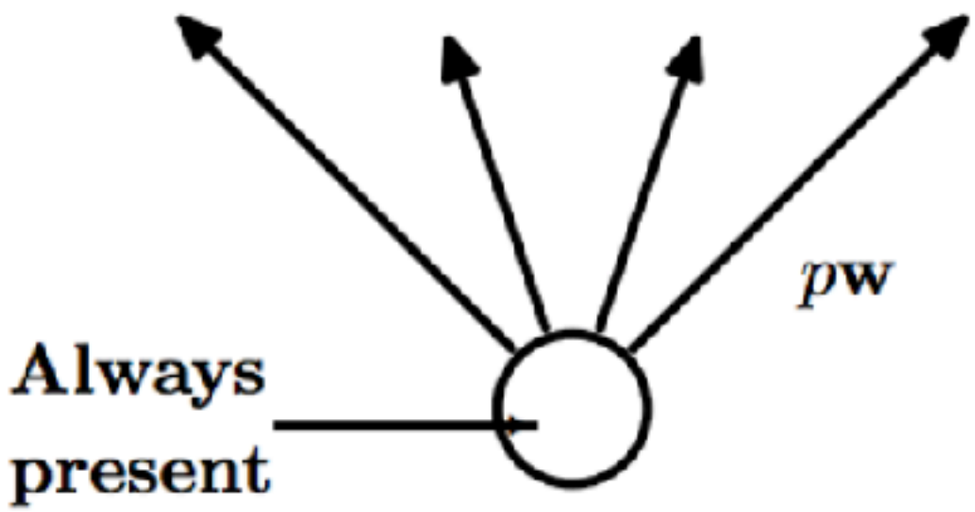
7.4.2 权重比例推断

权重比例推断

更高效的集成方法是**只使用一次前向传播**。保留所有神经元，并使连接权重乘以其保留概率，以保证输出与训练时的期望输出相同。这种方法称之为**权重比例推断规则（weight scaling inference rule）**。目前还没有在深度非线性网络上对这种近似推断的准确性作任何理论分析，但经验上表现的很好。



(a) At training time



(b) At test time

权重比例推断用法

- 方法一：训练结束后，将权重缩小一定比例。例如通常使用1/2的包含概率，则权重比例规则一般相当于在训练结束后将权重除以2。

- 方法二：在训练期间将神经元的输出值乘以2。

权重比例推断——例子

假如一个3层的神经网络中隐藏层有2个神经元，输出层有一个神经元，隐藏层与输出层的连接权重分别使用w1、w2表示。训练时隐藏层以50%的保留概率进行训练，其中第一个神经元输出的期望值为4，第二个神经元输出的期望值为2。

1. 计算输出层输入的期望；
2. 使用权重比例推断计算输出层输入的期望。

证明

权重比例推断规则在不具有非线性隐藏单元的模型中是精确的。为了证明这一点，首先需要使用几何平均代替算术平均。Warde- Farley *et al.* (2014) 提出的论点和经验证据表明，在这个情况下几何平均与算术平均表现差不多。

$$\sum_{\boldsymbol{r}} p(\boldsymbol{r}) p(y \mid \boldsymbol{x}, \boldsymbol{r})$$

算术平均

$$\sqrt[2^d]{\prod_{\boldsymbol{r}} p(\boldsymbol{r}) p(y \mid \boldsymbol{x}, \boldsymbol{r})}$$

几何平均，其中d表示可被丢弃的单元数。

Warde-Farley, D., Goodfellow, I. J., Courville, A., and Bengio, Y. (2014). An empirical analysis of dropout in piecewise linear networks. In *In ICLR'2014*.

证明

为了简化推导，我们使用均匀分布的 $\boldsymbol{r}$ ，但是非均匀分布也是可以的。此时几何平均为：

$$\tilde{P}_{ensemble}(y \mid \boldsymbol{r}) = \sqrt[2^d]{\prod_{\boldsymbol{r}} p(y \mid \boldsymbol{x}, \boldsymbol{r})}$$

由于几何平均之后的结果不再是概率分布，我们需要重新标准化集成结果，即：

$$P_{ensemble}(y \mid \boldsymbol{x}) = \frac{\tilde{P}_{ensemble}(y \mid \boldsymbol{x})}{\sum_{y'} \tilde{P}_{ensemble}(y' \mid \boldsymbol{x})}$$

这里，我们以softmax回归为例，其中 $\boldsymbol{v}$ 表示d个输入，则：

$$P(Y = y \mid \boldsymbol{v}) = \text{softmax}(\boldsymbol{W}^{\mathrm{T}} \boldsymbol{v} + \boldsymbol{b})_y$$

根据二值向量 $\boldsymbol{r}$ 与输入 $\boldsymbol{v}$ 逐元素乘法得到一类子模型：

$$P(Y = y \mid \boldsymbol{v}; \boldsymbol{r}) = \text{softmax}(\boldsymbol{W}^{\mathrm{T}} (\boldsymbol{r} \odot \boldsymbol{v}) + \boldsymbol{b})_y$$

证明

将 $P(Y=y\mid \boldsymbol{v};\boldsymbol{r})=\mathrm{softmax}(\boldsymbol{W}^{\mathrm{T}}(\boldsymbol{r}\odot\boldsymbol{v})+\boldsymbol{b})_y$ 带入可得：

$$\tilde{P}_{ensemble}(Y=y\mid \boldsymbol{v})=\sqrt[2^d]{\prod_{\boldsymbol{r}}P(Y=y\mid \boldsymbol{v};\boldsymbol{r})}$$

$$=\sqrt[2^d]{\prod_{\boldsymbol{r}}\mathrm{softmax}(\boldsymbol{W}^{\mathrm{T}}(\boldsymbol{r}\odot\boldsymbol{v})+\boldsymbol{b}))_y}$$

$$=\sqrt[2^d]{\prod_{\boldsymbol{r}}\frac{\exp(\boldsymbol{W}_{y,:}^{\mathrm{T}}(\boldsymbol{r}\odot\boldsymbol{v})+b_y)}{\sum_{y'}\exp(\boldsymbol{W}_{y',:}^{\mathrm{T}}(\boldsymbol{r}\odot\boldsymbol{v})+b_{y'})}}$$

$$=\frac{\sqrt[2^d]{\prod_{\boldsymbol{r}}\exp(\boldsymbol{W}_{y,:}^{\mathrm{T}}(\boldsymbol{r}\odot\boldsymbol{v})+b_y)}}{\sqrt[2^d]{\prod_{\boldsymbol{r}}\sum_{y'}\exp(\boldsymbol{W}_{y',:}^{\mathrm{T}}(\boldsymbol{r}\odot\boldsymbol{v})+b_{y'})}}$$

由于需要进行归一化操作，所以相同的分母将会被约去。即分母的形式不影响之后的结果。

证明

所以我们只关注 $\tilde{P}_{ensemble}$ 的分子即可，所以可得：

$$\tilde{P}_{ensemble}(Y=y\mid \boldsymbol{v}) \propto \sqrt[2^d]{\prod_{\boldsymbol{r}} \exp(\boldsymbol{W}_{y,:}^{\mathrm{T}}(\boldsymbol{r}\odot \boldsymbol{v})+b_y)}$$

$$= \exp\left(\frac{1}{2^d}\sum_{\boldsymbol{r}} \boldsymbol{W}_{y,:}^{\mathrm{T}}(\boldsymbol{r}\odot \boldsymbol{v})+b_y\right)$$

若神经元保留概率为 q 可化简为：

$$\tilde{P}_{ensemble}(Y=y\mid \boldsymbol{v}) \propto \exp\left(q\boldsymbol{W}_{y,:}^{\mathrm{T}}\boldsymbol{v}+b_y\right)$$

证明

最终可得到一个权重比例缩小的softmax输出，如下：

$$P_{ensemble}(Y=y\,|\,\boldsymbol{x})=\frac{\tilde{P}_{ensemble}(Y=y\,|\,\boldsymbol{x})}{\sum_{y'}\tilde{P}_{ensemble}(Y=y'\,|\,\boldsymbol{x})}$$

$$=\frac{\exp\left(q\boldsymbol{W}_{y,:}^{\mathrm{T}}\boldsymbol{v}+b_{y}\right)}{\sum_{y'}\exp\left(q\boldsymbol{W}_{y',:}^{\mathrm{T}}\boldsymbol{v}+b_{y'}\right)}$$

$$=\mathrm{softmax}(\mathrm{q}\boldsymbol{W}^{\mathrm{T}}\boldsymbol{v}+b)_{y}$$

所以在softmax回归中，权重比例推断时精确的（不考虑使用几何平均近似算术平均的时候）。

Dropout说明

- 使用Dropout训练时的随机性不是此方法成功的必要条件，它仅仅是近似子模型总和的一个方法。fast dropout与dropout boosting等实验证明了这一点。

- 通过随机行为训练并平均多个随机决策进行预测，实现了一种参数共享的Bagging形式。

- Dropout强大的大部分原因来自施加到隐藏单元的掩码噪声。

- Dropout使得隐藏层神经元之间的交互表现更加稳健。

- Dropout是一个正则化技术，它减少了模型的有效容量，为了抵消这种影响，必须增加模型规模。

实验四

- 改进实验二或者实验三构建好的神经网络，在其最后的一个隐藏层（其它层亦可）上以50%的概率均匀的随机丢弃神经元进行训练；

- 将训练好的模型的最后一个隐藏层与输出层之间的连接权重除以2完成模型构建；

- 在测试集上测试模型性能；

- （作业）查找并学习TensorFlow中在训练与预测时使用Dropout的方法，并对实验四进行改进：输入层使用20%的概率丢弃，每一个隐藏层使用50%的概率随机丢弃神经元，进行训练模型，并测试其在测试集上的性能与上述实验进行对比。事实上，**在极少的数据量的情况下，Dropout的作用并不明显**。此处仅仅为了学习如何使用。

实验四 部分内容示例

生成伯努利分布，并应用于隐藏层：

```
keep_prob = 0.5
drop_prob = 1. - keep_prob
rdu = tf.random_uniform([3], 0, 1)
r = tf.cast(tf.equal(tf.minimum(rdu, drop_prob), drop_prob), tf.float32)
hidden_output = tf.constant([[1, 2, 3.],
                             [4, 5, 6]])
res = tf.multiply(hidden_output, r)
```

8. 其它正则化方法

其它正则化方法

- 噪声鲁棒性。例如给样本、模型的神经元、输出目标等注入噪声。

- 半监督学习。例如构建一个共享参数的生成模型与判别模型，通过训练生成模型以赋予模型对某些任务的先验知识，通过训练共享参数的判别模型来强化模型。

- 参数绑定与共享。例如卷积神经网络中，使用了参数共享，提高了网络复杂度并显著降低了参数数量；

- 稀疏表示。

- 对抗训练。例如通过构造对抗样本增强模型训练集。

- 其它方法。

小结

- 所有能够提高模型泛化能力的方法都是正则化方法；
- 参数范数正则化可用来惩罚对模型贡献不重要的参数，其中超参数a的设置很关键；
- 数据集增强需要根据具体任务类型选择不同的方法；
- 多任务学习中共享表示部分“吸取”了不同任务的公共表示，更具鲁棒性；
- 提前终止避免了过度训练造成的过拟合，节省了计算资源；
- Bagging利用投票等方式得到最优结果。
- Dropout方法相当于很多共享参数的子模型集成的方法。

THANKS

Author:WangQi
Email:wangqikaixin@gmail.com