

Lecture4-1 Regularization

机器学习中的一个核心问题是设计不仅在训练数据上表现好，并且能在新输入上泛化好的算法

正则化 (regularization)：对学习算法的修改——旨在减少泛化误差而不是训练误差

- 一个有效的正则化是有利的“交易”，也就是能显著减少方差而不过度增加偏差

1. 参数范数惩罚

许多正则化方法通过对目标函数 J 添加一个参数范数惩罚 $\Omega(\theta)$ 限制模型（如神经网络、线性回归或逻辑回归）的学习能力

我们将正则化后的目标函数记为 \tilde{J}

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

- $\alpha \in [0, \infty)$: 范数惩罚项的权衡
- 注意，在神经网络中，参数 θ 包含每一层仿射变换的权重和偏置，我们只对权重做惩罚，而不对偏置做惩罚
- 使用向量 \mathbf{w} 表示应受范数惩罚影响的权重，而向量 θ 表示所有参数

L^2 参数正则化

最简单而最常见的参数范数惩罚，通常被称为权重衰减 (weight decay) 的 L^2 参数范数惩罚， L^2 正则化也被称为岭回归或 Tikhonov 正则

正则化策略：向目标函数添加一个正则化项 $\Omega(\theta) = \frac{1}{2} \|\mathbf{w}\|_2^2$ ，使权重更靠近原点

正则化后线性回归的解

在之前的章节中，我们使用均方误差损失函数 MSE 作为线性回归的代价函数

$$J(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

添加了 L^2 正则项后，目标函数变为

$$J(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \frac{1}{2} \alpha \mathbf{w}^T \mathbf{w}$$

这使得普通方程的解从

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

变成了

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X} + \alpha \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

正则化后目标函数的梯度

首先对于这样的模型，具有以下总的目标函数

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

与之对应的梯度为

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

使用梯度下降更新权重，即执行以下更新

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

L^1 参数正则化

正则化策略：向目标函数添加一个正则化项 $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$ ，即各个参数的绝对值之和

相比 L^2 正则化， L^1 正则化会产生更**稀疏 (sparse)** 的解

- 稀疏性指的是最优值中的一些参数为 0
- 由 L^1 正则化导出的稀疏性质已经被广泛地用于**特征选择 (feature selection)**

正则化后目标函数的梯度

首先对于这样的模型，具有以下总的目标函数

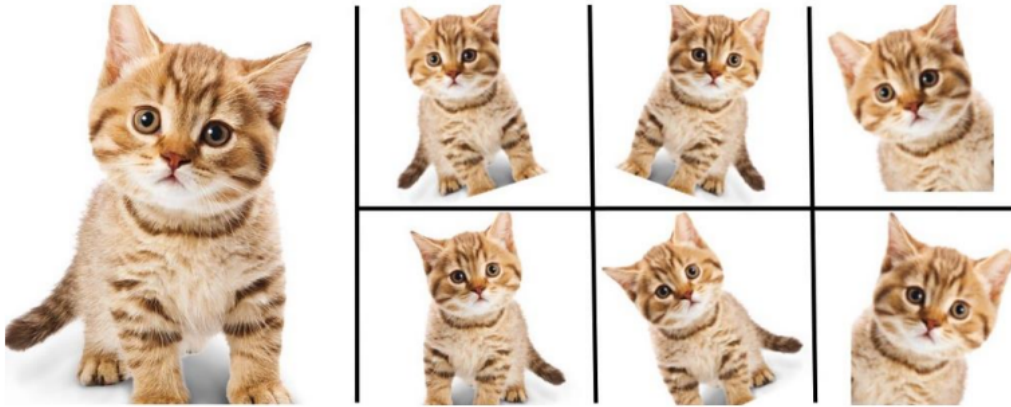
$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha \|\boldsymbol{w}\|_1$$

与之对应的梯度为

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \nabla_{\boldsymbol{w}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha \text{sign}(\boldsymbol{w})$$

- $\text{sign}(w)$ 只是简单地取 w 各个元素的正负号

2. 数据集增强



让机器学习模型泛化更好的最好办法是使用**更多的数据**进行训练，在数据集有限的情况下，可以**创建一些假数据并添加到训练集中**

数据集增强对于某些具体的分类问题是特别有效的方法

- 对象识别
- 语音识别

混淆 Mixup

mixup是一种运用在**计算机视觉**中的对图像进行混类增强的算法，它可以将不同类之间的图像进行混合，从而扩充训练数据集

- 通过**线性插值**构造虚拟训练样本
- 假设：增加混合插值的强度会产生离训练样例更远的虚拟样例，使得记忆更加困难

选取一个从 Beta 分布中的出来的参数 λ

$$\lambda \sim \text{Beta}(\alpha, \alpha)$$

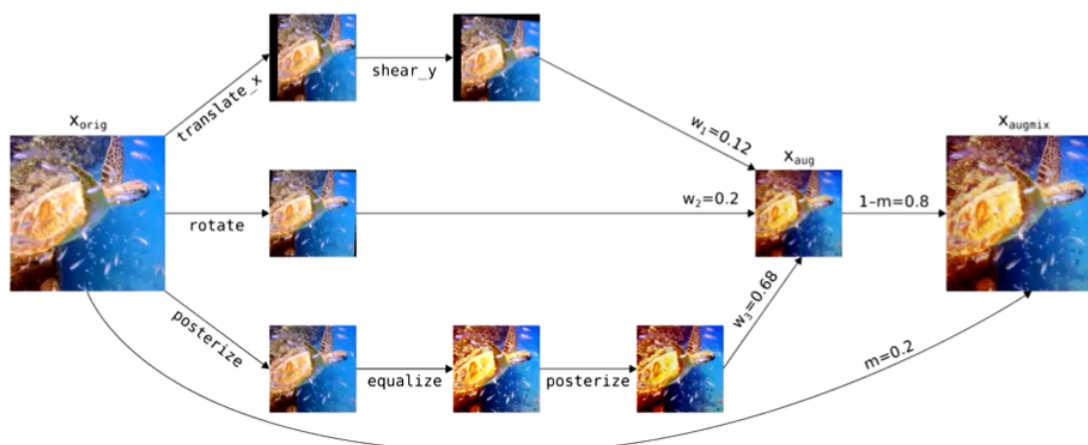
- α : 超参数，可以自己设置

假设有两个样本 $(x_i, y_i), (x_j, y_j)$ ，那么混合的结果为

$$\begin{aligned}\tilde{x} &= \lambda x_i + (1 - \lambda) x_j \\ \tilde{y} &= \lambda y_i + (1 - \lambda) y_j\end{aligned}$$

AugMix




AugMix 是一种数据增强方法，它随机采样各种增强操作后进行加权组合，允许我们探索原始图像周围语义等效的输入空间



对抗训练 Adversarial training

在精度达到人类水平的神经网络上通过优化过程故意构造数据点，其上的误差率接近100%，模型在这个输入点 x' 的输出与附近的数据点 x 非常不同

在许多情况下， x' 与 x 非常近似，人类观察者不会察觉原始样本和**对抗样本 (adversarial example)** 之间的差异，但是网络会作出非常不同的预测

	$+ .007 \times$		$=$	
x		$\text{sign}(\nabla_x J(\theta, x, y))$		$x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$
$y = \text{"panda"}$		"nematode"		"gibbon"
w/ 57.7%		w/ 8.2%		w/ 99.3 %
confidence		confidence		confidence

对抗样本对神经网络产生极大影响的主要原因之一是**过度线性**，神经网络主要是基于线性块构建的它们实现的整体函数被证明是**高度线性**的，这些线性函数很容易优化

但是如果一个线性函数具有许多输入，那么它的值可以**非常迅速地改变**

快速梯度符号方法 (FGSM)

- 目前，提高对抗鲁棒性最有效的方法是对抗性训练，即在标准训练中加入对抗样本
- 会在后面的章节讲到

$$\boldsymbol{x}^{adv} = \boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}), y)$$

- $\epsilon > 0$: 控制扰动幅度
- 简单来说，这是 FGSM 的想法是对抗训练产生的样本是基于**让样本向着样本的损失函数增大的方向进行偏移**

3. 提前终止 early stopping

当训练有足够的表示能力甚至会**过拟合**的大模型时，我们经常观察到，**训练误差会随着时间的推移逐渐降低但验证集的误差会再次上升**

在每次**验证集**误差有所改善后，我们存储模型参数的副本

当验证集上的误差在事先指定的循环次数内**没有进一步改善**时，算法就会**终止**

注意

提前终止需要验证集，这意味着某些训练数据不能被馈送到模型

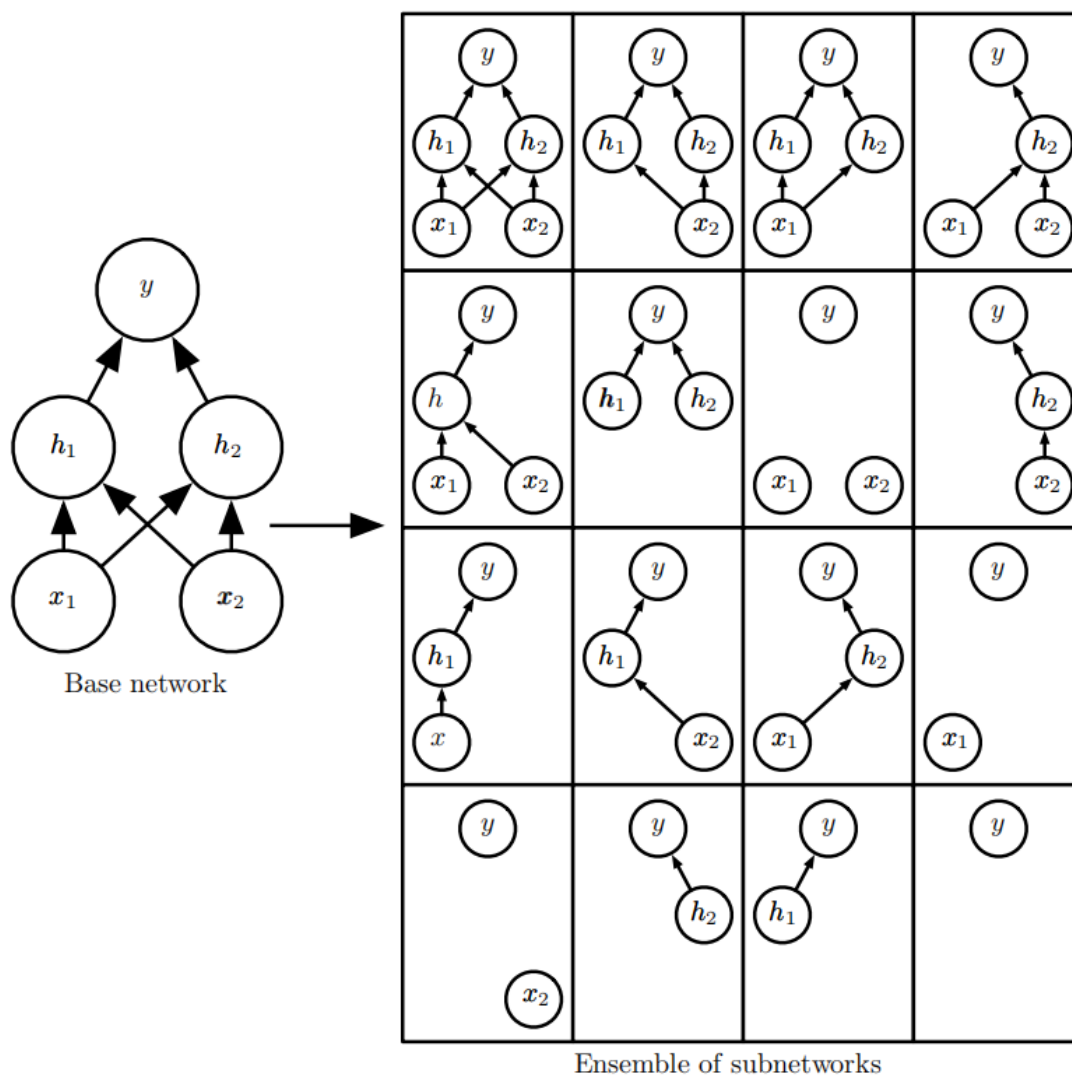
为了更好地利用这一额外的数据，我们可以在完成提前终止的首次训练之后，进行**额外的训练**

有两种测了进行额外的训练

- 再次初始化模型，然后使用所有的数据再次训练
 - 在第二轮训练中，我们使用第一轮提前终止训练所确定的最佳步数
- 保持第一轮训练获得的参数，然后使用全部的数据继续训练
 - 因为没有验证集指导什么时候停止，可以通过监控验证集的平均损失函数，直到它低于提前终止过程终止时的目标值
 - 但是，这种策略甚至不能保证种植，因为验证集的目标不一定能达到之前的目标值

4. Dropout

Dropout 训练的集成包括所有从基础网络除去非输出单元后形成的子网络



- 在训练中使用Dropout时，我们会使用基于**小批量**产生较小步长的学习算法，如**随机梯度下降**
- 每次在小批量中加载一个样本，然后随机抽样应用于网络中所有输入和隐藏单元的不同**二值掩码**
 - 每个单元，掩码是**独立采样**的
 - 掩码值为 1 的采样概率（导致 100% 包含一个单元）是训练开始前一个固定的超参数
 - 通常在每一个小批量训练的神经网络中
 - 一个**输入单元**被包括的概率为 0.8
 - 一个**隐藏单元**被包括的概率为 0.5
 - 在训练过程中，根据掩码随机“停用”一些单元
 - 有一定的概率不更新某些参数

优点

- 计算方便
- 不限制适用的模型或训练过程
 - 几乎在所有使用分布式表示且可以用随机梯度下降训练的模型上都表现很好
- 更快的训练
- 更少的过拟合
- 各个单元的鲁棒性更高

