2. 改善深层神经网络：超参数调试、正则化以及优化

深度学习

1. 深度学习的实用层面

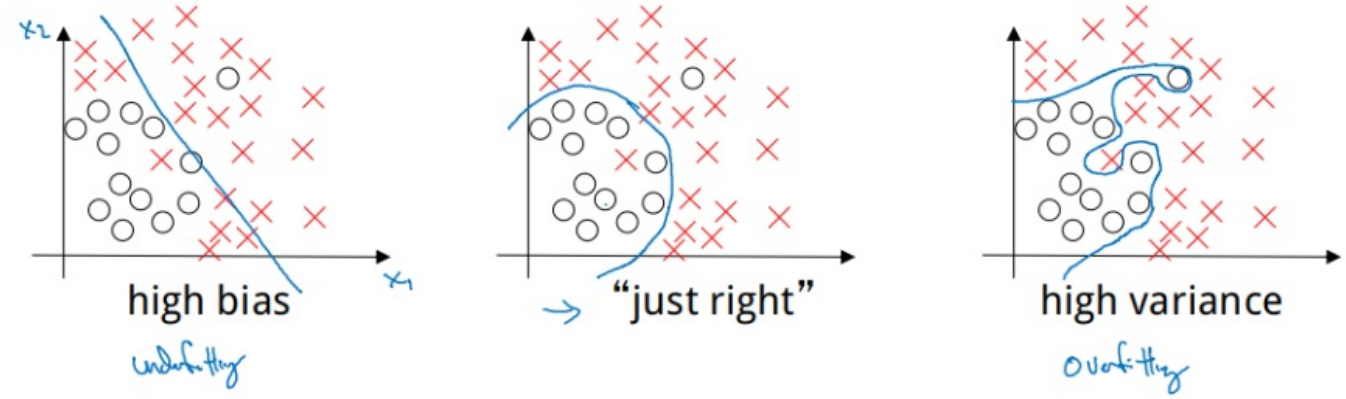
1.1 机器学习应用

训练/开发/测试集

在实际应用中，深度学习项目一般都需要大量的试验性工作，即使是有相关经验的专家也很难一次就找到最合适的模型框架和超参数。
为了方便试验，一般将数据划分为训练集(Training Sets)、开发集(Development Sets)和测试集(Test Sets)。我们一般采用训练集训练模型，用开发集查看不同超参数下的模型表现，并选择最好的超参数训练出的模型来应用到测试集上。注意需要保持各数据集中的数据来源和分布一致。

偏差和方差

在训练模型的过程中，如果使用一个非常简单的模型(例如线性模型)，模型可能不能很好地对数据进行拟合，存在较大的偏差（Bias），也叫作欠拟合（Underfitting）；如果使用复杂的模型(例如深度神经网络)，模型可能过度拟合训练数据，使得模型存在较大的方差（Variance），也叫作过拟合（Overfitting）。



我们训练出的模型一般有四种可能：

- **过拟合**：训练集的错误率较小，而开发集的错误率却较大，存在较大方差
- **欠拟合**：训练集和开发集的错误率都都较大时，且两者相当，存在较大偏差
- 训练集的错误率较大时，且开发集的错误率远较训练集大时，方差和偏差都较大
- **较优模型**：训练集和开发集的错误率都较小，且两者的相差也较小，此时方差和偏差都较小。

模型存在较大偏差时，可增加神经网络的隐含层数、隐含层中的节点数，或训练更多轮次来预防欠拟合。存在较大方差时，可以增加训练样本、或者用正则化（Regularization）等方法来预防过拟合。

1.2 防止过拟合与正则化

L1/L2正则化

L2正则化是向损失函数中加入惩罚项，惩罚项会倾向于使参数W缩小，因此也叫作权重衰减法：

$$J(W^{[1]},b^{[1]},W^{[2]},b^{[2]},\dots,W^{[L]},b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i,y_i) + \frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_F^2$$

其中 $||w^{[l]}||_F^2 = \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}(w_{ij}^{[l]})^2$

因此在计算梯度的过程中，梯度原来的梯度$dW^{[l]}$也会变为 $\frac{\partial L}{\partial W} + \frac{\lambda}{m}W^{[L]}$

λ确定了惩罚的强度，当其很大时，参数W会变得很小。

除了L2正则化外，也可以使用L1正则化

$$J(W^{[1]},b^{[1]},W^{[2]},b^{[2]},\dots,W^{[L]},b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i,y_i) + \frac{\lambda}{2m}\sum_{i=1}^{L}\sum_{l=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}|w_{ij}^{[l]}|$$

正则化对神经网络的作用

- 考虑极限的情况，正则化会使部分权重W变得很小。当它们接近0时，正则化相当于消除了一些节点的作用，因此防止了过拟合。
- 当权重W变小后，线性计算结果Z=WX+b的结果也相对较小。考虑到sigmoid等激活函数会在Z较小时接近线性，正则化也起到了使模型偏向线性的作用，不会过于复杂。
- 当权重W很小时，输入样本X的变化给模型带来的波动也会变小，这也使模型方差较小。

丢弃法Dropout

Dropout方法在传播过程中让一些节点随机失活。由于每个节点都有可能在某次传播过程中失活，因此整体模型不会过分依赖某个节点，达到了降低方差的目的。

反向随机失活(Inverted dropout)是一种目标比较常见的dropout方法，它的执行过程如下：

1) 预先设置神经元的保留概率，在每个训练样本i经过网络时，都利用这个概率选定失活的神经元，也就是将该节点的激活值$a_i^{[l]}$记为0

2) 对于保留的节点位置，它们的值要乘以保留概率的倒数，保持传播过程中的期望值不变

根据每层参数的多少，可以设置不同的留存率；一般不会让原始特征(输入层单元)失活。

```
# 正向传播过程
keep_prob = 0.8     # 设置神经元保留概率
dl = np.random.rand(al.shape[0], al.shape[1]) < keep_prob # 随机选择失活神经元
al = np.multiply(al, dl) # 将选定的失活神经元设置为0
al /= keep_prob # 增大保留神经元的数值，保持传播过程中的期望值不变

# 反向传播过程
```

```
dA2 = dA2*D2                # 反向传播中失活的位置应该和正向传播的相同
dA2 = dA2/keep_prob         # 对于没有失活的位置扩大其权重，保持传播过程中的期望值不变
```

注意：dropout不是针对权重W而是每层的激活后计算结果A。$A^{[l]}$的形式为$[n^{[l]}, m]$，因此在传播过程中，每个样本经过神经网络时，失活的节点都是不一样的。

其他预防过拟合的方法

- **增加样本**：获取更多数据、根据分布模拟出新数据、数据增强(限用于图像等问题，如对称或反转图片等方法)
- **调整网络结构**：减少层次、减少神经元个数
- **提前停止(early stopping)**：在每个epoch结束后，用验证集进行测试，如果accuracy不再提高，那么就停止继续训练。也可以设置容忍度，如2或3次训练不提高再停止
- **集成模型**：bagging等

各种方法对比

- 增加样本对模型没有坏处，但现实中收集数据比较困难。
- early stopping可以提升速度，但是无法分解最小化损失函数和控制过拟合两个任务，因此不满足正交化；同时，使用es会影响调参效果，因为无法控制每个模型的iteration相同。
- dropout里的损失函数无法被明确定义，因此无法使用查看成本函数是否递减的方法来监测学习情况，也无法进行梯度检查。
- 在神经网络中，L1正则化理论上可以形成稀疏的特征矩阵，但实际应用中收效甚微。
- L2正则化普遍比L1正则化好，因此比较推荐；不过整体而言，L1/L2正则化方法的速度都比较慢，因为需要不断寻找最佳λ。(**Andrew Ng推荐L2正则化**)

3. 最优化的准备工作

标准化

使用z-score标准化方法：$x^{(i)} = \frac{x^{(i)} - \bar{x}}{\frac{1}{m}\sum_{i=1}^{m} x^{(i)2}}$

原因：当特征的范围差距很大时，优化的速度会很慢，标准化有助于更快地进行优化。另外，进行标准化几乎是没有任何负面作用的。

梯度消失/爆炸与初始化

当网络层数很深时，尽管权重W可能并不大(仅略大于或略小于1)，但每层计算结果Z将以指数级递增或者递减，导致训练神经网络十分困难。

缓解方法：合理的初始化参数

- 对于部分激活函数，随机参数矩阵*$\sqrt{\frac{1}{n^{[l-1]}}}$

- 对于ReLU，随机参数矩阵*$\sqrt{\frac{2}{n^{[l-1]}}}$

梯度检查

根据导数的定义，函数$J(\theta)$在某个具体的θ处导数应为：$J'(\theta) = lim_{\epsilon \to 0} \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$
利用此方法可以进行梯度检验。对于某个参数θ_i，手动近似计算该维度在该点上的导数(使用双边方法)，再通过与计算出的$d\theta_i$比较来检查梯度计算是否正确：
$d\theta_{approx}[i] = \frac{J(\theta_1,\theta_2,...,\theta_i+\varepsilon,...)-J(\theta_1,\theta_2,...,\theta_i-\varepsilon,...)}{2\varepsilon}$
计算$\frac{||d\theta_{approx}[i]-d\theta[i]||_2}{||d\theta_{approx}[i]||_2+||d\theta[i]||_2}$的结果，如果很小说明目前的梯度计算正确。

2. 优化算法

2.1 小批量梯度下降

(1) 批量梯度下降(batch)

普通的梯度下降形式，即每次迭代使用整个训练集，速度非常慢

(2) 小批量梯度下降(mini-batch)

每次迭代时，使用所有样本中的一部分进行梯度下降，这部分样本称为一个mini-batch。具体执行中，会将全部样本分为多个组，每次迭代选择一个组进行训练。当训练集的所有样本都使用一次后，称为完成了一个epoch。
小批量梯度下降的损失函数变化呈震荡下降的趋势，最终一般不会停止在最优点上，而是在其附近波动。

(3) 随机梯度下降(SGD)

每次使用一个样本进行梯度下降，可以看做每批样本数等于1的小批量梯度下降。使用这种方法时，需要选择较小的学习速率。由于每次只输入一个样本，SGD失去了向量计算带来的加速效果，因此效率是比较低的。

优化算法 选择

- 小训练集(例如小于2000)时，直接使用批量梯度下降。
- 大训练集时，使用小批量梯度下降。选择2^n，例如128/256/512等，作为每次迭代使用的样本数。

2.2 动量梯度下降(Momentum)

指数加权平均

计算方法

指数加权平均是一种常见的时间序列处理方法。假设有时间序列θ_t，我们可以计算出它的指数平均加权序列V_t：当t=0时，$V_t = 0$；当t>0时，$V_t = \beta V_{t-1} + (1-\beta)\theta_t$。

使用效果

指数加权平均相当于对时间序列进行平滑处理，它考虑到序列之前几个时间点上的值，而不是仅由当前值决定。

β的取值在0~1之间，效果近似相当于过去$\frac{1}{1-\beta}$个时期的加权平均。例如当$\beta = 0.9$时，V_t相当于过去10个时间点的加权平均数。因为如果将V_{t-1}的逐层带入t式，会得到

$V_t = (1-\beta)\theta_t + (1-\beta)\beta\theta_{t-1} + \ldots + (1-\beta)\beta^n\theta_{t-n}$。

$\frac{1}{1-\beta}$个时期后，权重已经缩减到1/e以下，因此可以忽略不计。

优点

相比于直接计算前t个时间点的加权平均数，指数平均加权的优点是节省内存。因为每一个时间点的v_t都是仅基于上一个时间点的，因此理论上只需要保存v_t值。

偏差修正

v_t在初始阶段偏差会比较大。当t=1时，$V_1 = \beta V_0 + (1-\beta)\theta_1$，由于$V_0 = 0$，$V_1$的值就会很小。随着t的增长，误差会越来越小直至接近于无。

这种误差可以考虑使用$\frac{v_t}{1-\beta^t}$来修正。修正方法在t较小时作用明显，但当t比较大时，β^t接近0，对v_t几乎没有影响。

动量梯度下降(Momentum)

Momentum方法运用了指数加权平均的思想。对于dW或db：

- $v_{dW} = \beta v_{dW} + (1-\beta)dW, W = W - \alpha v_{dW}$
- $v_{db} = \beta v_{db} + (1-\beta)db, b = b - \alpha v_{db}$

α和β都是超参数。α表示学习速率，β控制指数加权平均的效果，表示之前梯度衰减的速度。一般来说，没有必要对v进行偏差修正。

相比于梯度下降中每次梯度都独立于前一次的，Momentum中的v_{dw}考虑到了之前几次的梯度计算结果。如果本次梯度方向与前几次相同，则产生了协同效应加速梯度更新；如果相反，则互相抵消，避免了没必要的反复震荡。因此实际应用中，Momentum的效果几乎总是好于一般梯度下降的。

2.3 均方根传递(RMSprop)

能够减小在某个方向上的震荡程度；ε可以避免分母为0

- $S_{dW} = \beta S_{dW} + (1-\beta)dW^2, W = W - \alpha \frac{dW}{\sqrt{S_{dW}}+\varepsilon}$
- $S_{db} = \beta S_{db} + (1-\beta)db^2, b = b - \alpha \frac{db}{\sqrt{S_{db}}+\varepsilon}$

2.4 Adam(Adaptive Moment Estimation)

Adam是Momentum和RMSprop的结合形式。因为两种方法都含有β参数，为了避免混淆，Momentum中的β记为β_1，RMSprop中的β记为β_2。

- 初始状态：$v_{dW} = 0$，$s_{dW} = 0$，$v_{db} = 0$，$s_{db} = 0$
- 计算Momentum中的v：$v_{dW} = \beta_1 v_{dW} + (1-\beta_1)dW, v_{db} = \beta_1 v_{db} + (1-\beta_1)db$
- 计算RMSprop中的s：$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2, S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$
- v的偏差修正：$v_{dW}^{correct} = v_{dW}/(1-\beta_1)^t, v_{db}^{correct} = v_{db}/(1-\beta_1)^t$，t是迭代次数
- s的偏差修正：$s_{dW}^{correct} = s_{dW}/(1-\beta_2)^t, s_{db}^{correct} = s_{db}/(1-\beta_2)^t$，t是迭代次数
- $W = W - \alpha \frac{v_{dW}^{correct}}{\sqrt{S_{dW}^{correct}}+\varepsilon}$
- $b = b - \alpha \frac{v_{db}^{correct}}{\sqrt{S_{db}^{correct}}+\varepsilon}$

调参

- 学习速率α：主要调试的参数
- $beta_1$：常用值为0.9
- $beta_2$：常用值为0.999
- ε：不重要，几乎不会影响算法表现，可以选10^{-8}

2.5 学习速率衰减

- 衰减率方法：$\alpha = \frac{1}{1+decay\_rate*epoch\_num} * \alpha_0$
- 指数衰减：$alpha = r^{epoch\_num} * \alpha_0$，r在0~1之间
- $alpha = \frac{k}{\sqrt{epoch\_num}} * \alpha_0$，k为常数
- 手动调节

3. 超参数调试和Batch标准化

3.1 选择超参数

重要性

- 最重要：学习速率α
- 次重要：动量衰减参数β_1、各层隐藏神经元的、mini-batch size大小等
- 一般重要：Adam的其他超参数、神经网络层数和学习速率衰减率等

调参方法

当超参数很多时，Grid Search不太适用，一般考虑使用Random Search。
调参时，一般会遵循从粗略到精细的过程，也就是说，先在一个大的参数空间里粗略搜索，之后逐渐缩小参数空间，在更小的范围里选择最佳超参数组合。

3.2 Batch Normalization

标准化是指处理输入的特征，将所有特征转换为数据范围相近的形式，可以加快最优化。
除了对初始特征进行标准化为，也可以对隐藏层的计算结果进行标准化，一般而言是直接对线性计算结果Z记性标准化，也称为批量标准化(BN)。

- 计算Z的平均值：$\mu = \frac{1}{m}\sum z^{(i)}$
- 计算Z的标准差：$\sigma^2 = \frac{1}{m}\sum (z^{(i)} - \mu)^2$
- 标准化计算：$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$，$\varepsilon$是一个非常小的常数，可以避免分母为0
- 改进计算结果：$\tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$。主要是为了防止结果过于全部是以0为均值，1为标准差的数列，过于均匀。γ和β是参数(不是超参数)，是可以通过优化算法训练出来的，不需要预先设定。

使用BN时，主要参数是权重系数W和BN中的参数γ和β，偏置项b可以省略，因为它在减去均值一步中会被消去。
BN限制了在前层的参数更新对数值分布的影响程度，使后层的学习更为容易。同时，BN减少了各层的关联性，使它们更加独立，因此也起到了微弱的正则化效果。
在test阶段，由于每次只输入一个样本，无法计算μ和σ^2来进行标准化。因此，应在训练阶段保存它们的指数加权平均结果，在test时直接使用。

3.3 用softmax进行多分类

对于多分类问题，最后一层应有多个节点，每个节点代表一类。softmax函数对输出值进行转换。假设多分类问题有C类，则输出层应用C个节点，每个节点激活后的值应为$a_j^{[L]} = \frac{e^{z_j^{[L]}}}{\sum_{i=1}^{C} e^{z_i^{[L]}}}$。通过计算可以发现，最后一层所有节点的输出值之和为1，因此softmax后的结果可以看作概率。模型将C个节

点中输出值最大的点作为预测的结果。
softmax的损失函数形式为： $L(\hat{y},y)=-\sum_{j=1}^{C}y_j log\hat{y}_j$ 。

3.4 深度学习框架

主流框架

- **Keras**
- **Tensorflow**
- Caffe
- CNTK
- DL4J
- Lasagne
- mxnet
- PaddlePaddle
- Theano
- **Torch**

选择标准

- 便于编程：包括神经网络的开发和迭代、配置产品
- 运行速度：特别是训练大型数据集时
- 是否真正开放：不仅需要开源，而且需要良好的管理，能够持续开放所有功能