吴恩达深度学习第二课 学习笔记

> 二、网络正则化以及优化方法
> （1）视频教程：学习吴恩达deeplearning.ai系列教程中第二课和第三课所有内容。
> （2）学习目标：了解正则化方法、优化方法、数据集划分方式，学习超参数调节技巧。重点掌握mini-batch梯度下降法和batch norm正则化方法。
> （3）动手实验：完成第二课对应的课后编程作业并撰写报告，报告主要记录实验原理（mini-batch、batch norm理论推导等）、实验环境、实验结果、结果分析等。

相关链接
吴恩达深度学习第二课
作业链接1
作业链接2

week-1

1.1 数据集

训练集 training set
验证集 development set (验证不同算法的效果)
测试集 test set (评估性能)

确保 验证集和测试集 来源与同一分布
测试集可有可无

1.2 & 1.3 偏差 和 方差

train set error
dev set error

训练集的误差比验证集小很多，过拟合，高方差
训练集和验证集误差都较大，欠拟合，高偏差
训练集和验证集误差都较大 且训练集的误差比验证集小很多，高方差，高偏差
训练集和验证集误差都较小，低方差，低偏差

偏差：训练集错误率与0%(基本/最优错误率)的差别
方差：训练集错误率和测试集错误率之间的差别

解决高偏差方法
1. 训练更长时间
2. 选择更大的网络
3. 找到更合适的神经网络框架

解决高方差方法
1. 采用更多数据
2. 正则化减少过拟合
3. 找到更合适的神经网络框架

1.4 正则化 Regularization

cost函数 L2正则化

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(y\hat{^{(i)}},y^{(i)})+\frac{\lambda}{2m}\cdot\|w\|_2^2$$
$$\|w\|_2^2=\sum_{j=1}^{n_x}{w_j}^2=w^Tw$$

λ: 正则化参数 使用验证集来配置 权衡训练集和验证集 避免过拟合 是一个超参数
$\|w\|_2^2$: w的欧几里得范数 (L2范数)

省略b参数的正则化原因
w是高纬度参数，加上b影响不大

L1正则化

$$\frac{\lambda}{m}\|w\|_1^2=\sum_{j=1}^{n_x}w_j$$

若用L1正则化，w参数有正有负，求和后为0，使模型变得稀疏

一般使用L2模型

Forbenius 范数

$$\left\|W^{[l]}\right\|_F^2=\sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}(w_{ij}^{[l]})^2$$
$$(W^{[l]}:n^{[l-1]}\times n^{[l]})$$

正则化后

$$J(W^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y^{(i)}}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left\|W^{[l]}\right\|_F^2$$
$$\mathrm{d}W^{[l]} = (\dots) + \frac{\lambda}{m}W^{[l]}$$
$$w^{[l]} := w^{[l]} - \alpha \mathrm{d}w^{[l]} = w^{[l]}(1 - \frac{\alpha\lambda}{m}) - \alpha(\dots)$$

亦称权重衰减

缺点
需要训练多次找到合适的 λ 值

1.5 正则化减少过拟合(减小方差)原因

直观理解：正则化权重衰减了一些节点，每层网络变简单

$\lambda\uparrow, \quad W^{[l]}\downarrow \quad :\rightarrow \quad Z^{[l]}\downarrow$
对于tanh激活函数，z小近似线性，网络近似线性，网络复杂性降低

若添加了正则化范数，进行梯度下降，计算cost时，亦需要加上正则化范数

1.6 & 1.7 dropout正则化

随机失活正则化
随机消除每层网络中的节点，得到更简单的网络

实现dropout方法

inverted dropout 反向随机失活

```
keep-prob = 0.8 # keep-prob : 保留节点的概率
d = np.random.rand(a.shape[0], a.shape[1]) < keep-prob # 布尔值
a = np.multiply(a, d)
a /= keep-prob
```

a参数最后需要除keep-prob，防止z过小(z = wa + b)，这样也不影响测试时候的w参数

dropout减小过拟合原理
由于随机消除节点，l层的节点不能仅依靠l-1层的任意节点
即w权值不会在一个节点上过大，w权值在l-1上分散开，达到了收缩权重的效果

对于节点多的层，使用小的keep-prob以减小过拟合
第一层最好不要用dropout

dropout在计算机视觉中常用
dropout在发生过拟合的时候才用

dropout缺点
cost函数J定义不明确
解决办法：先不使用dropout，确保cost函数下降后，再使用

1.8 其他正则化方法

图像翻转，裁剪，变形等

early stopping
在dev set error 的极小值点停止训练
开始时 w初始值很小，训练多次后w值很大，提前停止训练使得w值适中
缺点
提前结束训练，J不能再小

1.9 归一化输入

归一化输入以加速训练

零均值化

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$
$$x := x - \mu$$

归一化方差

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} [x^{(i)}]^2$$
$$x := x/\sigma^2$$

归一化后有利于梯度下降

1.10 梯度消失和梯度爆炸

计算的导数(梯度)变得非常大或非常小，训练难度增加

多层神经网络，w参数以层数L指数增长或指数减小

1.11 初始化参数

初始化参数能减小梯度消失和梯度爆炸

较为合理的方法
设置 w_i 的方差为 $\frac{1}{n^{[l-1]}}$

$$W^{[l]} = np.random.randn(shape) * np.sqrt(\frac{1}{n^{[l-1]}})$$

若l层使用ReLU激活函数，方差使用 $\frac{2}{n^{[l-1]}}$ 更好
若l层使用tanh激活函数，方差使用 $\frac{1}{n^{[l-1]}}$ 或者 $\frac{2}{n^{[l-1]}+n^{[l]}}$

这个方差可以成为一个超参数

1.12 & 1.13 & 1.14 梯度检验

检验反向传播是否正确

计算梯度的数值逼近
拉格朗日中值定理(两边加小量)计算导数

梯度检验
将W和b 重塑为 θ
将dW和db重塑为 $\mathrm{d}\theta$

计算某点数值逼近的导数值和该点导数值的范数(欧式距离)

$$\left\|\mathrm{d}\theta_{approx} - \mathrm{d}\theta\right\|_2$$

即求误差平方和后开根号

归一化

$$\frac{\left\|\mathrm{d}\theta_{approx} - \mathrm{d}\theta\right\|_2}{\left\|\mathrm{d}\theta_{approx}\right\|_2 + \left\|\mathrm{d}\theta\right\|_2}$$

设置$\varepsilon = 10^{-7}$
检验是否在范围内

$$\frac{\left\|\mathrm{d}\theta_{approx} - \mathrm{d}\theta\right\|_2}{\left\|\mathrm{d}\theta_{approx}\right\|_2 + \left\|\mathrm{d}\theta\right\|_2} \leq \varepsilon$$

此值非常小，则认为神经网络正常

注意事项
梯度检验仅在调试中使用，不要在训练中使用
如果梯度检验失败，需要检查所有项，以找出bug
如果使用正则化，注意正则化项(在计算梯度时需要带上正则化项)
梯度检验和dropout不能同时使用
若随机初始化值较小，训练不会使w和b变大，可以在初始化时进行梯度检验，再进行训练

week-2

2.1 & 2.2 mini-batch梯度下降法

对于大量的训练集数据(m很大)，如果训练完整个训练集再梯度下降才进行梯度下降，效率低
可以先梯度下降处理一部分，算法速度更快
即把训练集切分成几部分，各自执行梯度下降

把训练集分割为训练子集 mini-batch

$$\begin{aligned} X^{\{t\}}, &\quad (n_x \times m') \\ Y^{\{t\}}, &\quad (1 \times m') \end{aligned}$$

每次 使用$X^{\{t\}}, Y^{\{t\}}$进行梯度下降法
每代(one epoch)训练进行T次梯度下降法

单次训练中，每代的梯度下降后，cost震荡下降

使用mini-batch需要决定mini-batch的大小

若设置mini-batch大小为m，即batch梯度下降法
每次梯度下降朝着全局最小值走，但每次算力要求较高，耗时较长

若设置mini-batch大小为1，即stochastic 梯度下降法(随机梯度下降法)，每个样本都是一个mini-batch
每次梯度下降不一定朝着全局最小值走，有很多噪声(通过减小学习率可以改善)，不能加速，效率低，不一定收敛

实际选择合适的mini-batch大小，每次梯度下降不一定总向最小值走，会有小的波动，可以减小学习率改善

选择mini-batch大小的准则
对于m较小的训练集(m<=2000)，直接batch梯度下降

对于m较大的训练集，设置mini-batch的大小为64-512 (2的幂次计算快) 可以尝试找合适的
确保mini-batch适合内存

2.3 & 2.4 & 2.5 指数加权平均

指数平均迭代公式

$$
\begin{gathered}
v_\theta = 0 \\
v_\theta := \beta \cdot v + (1-\beta)\theta_1 \\
v_\theta := \beta \cdot v + (1-\beta)\theta_2 \\
\cdots
\end{gathered}
$$

写进循环

$$
\begin{gathered}
v_\theta = 0 \\
v_\theta := \beta \cdot v + (1-\beta)\theta_t, \quad (t = 1 \ldots T)
\end{gathered}
$$

使用迭代，节省内存

偏差修正
设置$v_0 = 0$时，估计初期(前面的值)数据过小
使用

$$
\frac{v_t}{1-\beta^t}
$$

用于修正t较小时v的值，t增大后，分母近似为1

2.6 动量梯度下降法

momentum法运行速度快于标准梯度下降法

有波动，学习率不能太大

$$
\begin{gathered}
v_{\mathrm{d}W} = \beta \cdot v_{\mathrm{d}W} + (1-\beta)\mathrm{d}W \\
v_{\mathrm{d}b} = \beta \cdot v_{\mathrm{d}b} + (1-\beta)\mathrm{d}b
\end{gathered}
$$

即求 $\mathrm{d}W$ 和 $\mathrm{d}b$ 的指数加权平均，用来更新W和b

$$
\begin{gathered}
W := W - \alpha \cdot v_{\mathrm{d}W} \\
b := b - \alpha \cdot v_{\mathrm{d}b}
\end{gathered}
$$

由此可以减缓梯度下降的幅度
(向两边波动的分量抵消，摆动变小，向极值点移动更快)

β 亦为超参数 (常用$\beta = 0.9$)

注意事项
1.在使用momentum时，可以不用进行偏差修正
2.初始值 $v_{\mathrm{d}W} = 0, \quad v_{\mathrm{d}b} = 0$ 需要分别与 $\mathrm{d}W$ 和 $\mathrm{d}b$ 有相同维度

2.7 RMSprop

减慢b的学习，加速w的学习

$$S_{\mathrm{d}W} = \beta \cdot S_{\mathrm{d}W} + (1-\beta)(\mathrm{d}W)^2$$
$$S_{\mathrm{d}b} = \beta \cdot S_{\mathrm{d}b} + (1-\beta)(\mathrm{d}b)^2$$

即求 $(\mathrm{d}W)^2$ 和 $(\mathrm{d}Wb)^2$ 的指数加权平均，用来更新W和b

$$W := W - \alpha \frac{\mathrm{d}W}{\sqrt{S_{\mathrm{d}W}}}$$
$$b := b - \alpha \frac{\mathrm{d}b}{\sqrt{S_{\mathrm{d}b}}}$$

相当于将dW和db归一化后来更新W和b，以减小波动

注意事项
1.使用 β_2 以区分momentum的 β
2.为防止 $S_{\mathrm{d}W}$ 趋近于0
使得 $\frac{\mathrm{d}W}{\sqrt{S_{\mathrm{d}W}}}$ 过大
可将其改为 $\frac{\mathrm{d}W}{\sqrt{S_{\mathrm{d}W}+\varepsilon}}$ 以使其稳定

这样可以使用较大的学习率以得到更好的效果

2.8 Adam 算法

结合momentum和RMSprop方法

初始化

$$V_{\mathrm{d}W} = 0, \quad V_{\mathrm{d}b} = 0$$
$$S_{\mathrm{d}W} = 0, \quad S_{\mathrm{d}b} = 0$$

迭代

$$
\begin{gathered}
V_{\mathrm{d}W} = \beta_1 \cdot V_{\mathrm{d}W} + (1-\beta_1)\mathrm{d}W \\
V_{\mathrm{d}b} = \beta_1 \cdot V_{\mathrm{d}b} + (1-\beta_1)\mathrm{d}b \\
S_{\mathrm{d}W} = \beta_2 \cdot S_{\mathrm{d}W} + (1-\beta_2)(\mathrm{d}W)^2 \\
S_{\mathrm{d}b} = \beta_2 \cdot S_{\mathrm{d}b} + (1-\beta_2)(\mathrm{d}b)^2
\end{gathered}
$$

修正偏差

$$
\begin{gathered}
V_{\mathrm{d}W}^{\text{corrected}} = \frac{V_{\mathrm{d}W}}{1-\beta_1^t} \\
V_{\mathrm{d}b}^{\text{corrected}} = \frac{V_{\mathrm{d}b}}{1-\beta_1^t} \\
S_{\mathrm{d}W}^{\text{corrected}} = \frac{S_{\mathrm{d}W}}{1-\beta_2^t} \\
S_{\mathrm{d}b}^{\text{corrected}} = \frac{S_{\mathrm{d}b}}{1-\beta_2^t}
\end{gathered}
$$

更新w和b

$$
\begin{gathered}
W := W - \alpha \frac{V_{\mathrm{d}W}^{\text{corrected}}}{\sqrt{S_{\mathrm{d}W}^{\text{corrected}}} + \varepsilon} \\
b := bW - \alpha \frac{V_{\mathrm{d}b}^{\text{corrected}}}{\sqrt{S_{\mathrm{d}b}^{\text{corrected}}} + \varepsilon}
\end{gathered}
$$

在学习中常用，对大所属结构有效

超参数
α: 学习率
β_1: 对 $\mathrm{d}W$ 加权平均的系数 (一般设为0.9)
β_2: 对 $(\mathrm{d}W)^2$ 加权平均的系数 (一般设为0.999)
ε: 使算法稳定，不影响算法表现(一般设为10^{-8})

使用Adam方法的时候，$\beta_1, \beta_2, \varepsilon$使用默认值，调整不同的$\alpha$值使效果更好

2.9 学习率衰减

使用固定的学习率进行mini-batch时，由于有噪声，训练到后面，在极值附近摆动，不收敛
在训练中缓慢减小学习率，使得训练到最后，结果在极值附近小区域内波动

epoch:每代(对整个数据集训练完一次)

设置学习率为

$$\alpha = \frac{\alpha_0}{1 + \text{decayRate} * \text{epochNum}}$$

decayRate 也是一个超参数

也可以设置为指数衰减

$$\alpha = 0.95^{\text{epochNum}} \cdot \alpha_0$$

也可以设为

$$\alpha = \frac{k}{\sqrt{\text{epochNum}}} \cdot \alpha_0$$
$$\alpha = \frac{k}{\sqrt{t_{\text{miniBatch}}}} \cdot \alpha_0$$

也可以设置为离散下降值
也可以手动减小

2.10 局部最优问题

高维空间中，梯度为0的点不一定是极值，也可能是鞍点，且大概率是鞍点

平稳段学习缓慢，需要用momentum，RMS，Adam等方法快速走出平稳段

week-3

3.1 & 3.2 & 3.3 超参数的设置

α -最重要T1

β_{momentum} -次重要T2
隐藏层单元数 -次重要T2
mini-batch size -次重要T2

层数 -重要T3
学习率下降函数(衰减率) -重要T3

$\beta_1, \beta_2, \varepsilon$ -不太重要T4

在范围内随机选择
先粗略找出表现较好的范围，再在新的范围内随机选择

选择合适的标尺
对于学习率，在对数轴上随机选择
r = -4 * np.random.rand()
alpha = 10 ** r

对于beta的取值，也用对数
beta = 1 - 10 ** r

3.4 & 3.5 & 3.6 & 3.7 batch归一化

对网络中每层的a归一化，以更快训练
通常对z归一化

对于一层的一个节点

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$
$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$
$$z_{\text{norm}}^{(i)} = \frac{z^i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

这样，Z近似服从均值0，方差1的正态分布
我们想要有所改变

$$\tilde{z}^{(i)} = \gamma \cdot z_{\text{norm}}^{(i)} + \beta$$

其中， γ 和 β 为可学习的参数，可以用各种优化器来更新
即随意设置 $\tilde{z}^{(i)}$ 的均值和方差

当 $\gamma = \sqrt{\sigma^2 + \varepsilon}, \quad \beta = \mu$ 时
$\tilde{z}^{(i)} = z_{\text{norm}}^{(i)}$

使用以上4个式子，即可改变Z的均值和方差

若将Z的均值和方差分别设置为0和1
对于sigmoid，tanh等激活函数，Z的取值可能大部分在线性区间

batch norm (BN)

batch norm通常与mini-batch结合使用

使用batch norm的时候，可以将b参数舍去 (对z归一化b参数被消除，相当于beta取代了b)

$$\gamma^{[l]} : (n^{[l]} \times 1)$$
$$\beta^{[l]} : (n^{[l]} \times 1)$$

mini-batch时，每次分批的数据在均值和方差上有偏移
BN相当于添加了噪声，类似正则化，有正则化的效果，但添加的噪声较小，可以和dropout结合使用

对于测试数据(单个数据) 如何处理
对于单个样本，不能得到 μ 和 σ^2 ，需要估算

典型方法，使用指数加权平均来估算

对于mini-batch的每一个子集，都有一个 μ
对其指数加权平均，得到 μ 的估计值
对 σ^2 同理

预测的时候，使用估计的 μ 和 σ^2 来计算

3.8 & 3.9 softmax 回归

多种分类问题

C: 类别总数(包括其他)

输出层有C个节点 $n^{[L]} = C$
每个节点的值代表各自类别的概率
节点值之和为1

softmax层激活函数
在输出层

$$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}, \quad (C \times 1)$$
$$t = e^{Z^{[l]}}, \quad (C \times 1)$$
$$A^{[L]} = \frac{e^{Z^{[l]}}}{\sum_{j=1}^{C} e^{Z^{[l](i)}}} = \frac{e^{Z^{[l]}}}{\sum_{j=1}^{C} t_j} = \frac{e^{Z^{[l]}}}{\text{np.sum}(t)}, \quad (C \times 1)$$

以上构成softmax层的激活函数
这样得到的A和为1，原本Z中最大的值经过激活函数后仍最大
不同于之前的激活函数(仅传单值)，softmax激活函数需要传入一个向量(C个值)

hardmax
hardmax函数将Z参数直接转为布尔值，最大元素为1，其余为0

相比，softmax更加温和，将Z转为概率值

C = 2 时，softmax退化为logistic回归

使用softmax时的损失函数
与logistic相似，由最大似然导出

3.10 & 3.11 深度学习框架

tensorflow

关于tensorflow环境的链接

tensorflow官网
cuda-toolkit官网
cudnn官网
配置tf-gpu