**前言:**

　　最近打算稍微系统的学习下deep learing的一些理论知识，打算采用Andrew Ng的网页教程[UFLDL Tutorial](http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial)，据说这个教程写得浅显易懂，也不太长。不过在这这之前还是复习下machine learning的基础知识，见网页：<http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=DeepLearning>。内容其实很短，每小节就那么几分钟，且讲得非常棒。

**教程中的一些术语:**

　　Model representation:

　　其实就是指学习到的函数的表达形式，可以用矩阵表示。

　　Vectorized implementation:

　　指定是函数表达式的矢量实现。

　　Feature scaling：

　　指是将特征的每一维都进行一个尺度变化，比如说都让其均值为0等。

　　Normal equations:

　　这里指的是多元线性回归中参数解的矩阵形式，这个解方程称为normal equations.

　　Optimization objective:

　　指的是需要优化的目标函数，比如说logistic中loss function表达式的公式推导。或者多元线性回归中带有规则性的目标函数。

　　Gradient Descent、Newton’s Method：

　　都是求目标函数最小值的方法。

　　Common variations:

　　指的是规则项表达形式的多样性。

**一些笔记：**

　　模型表达就是给出输入和输出之间的函数关系式，当然这个函数是有前提假设的，里面可以含有参数。此时如果有许多训练样本的话，同样可以给出训练样本的平均相关的误差函数，一般该函数也称作是损失函数（Loss function）。我们的目标是求出模型表达中的参数，这是通过最小化损失函数来求得的。一般最小化损失函数是通过梯度下降法（即先随机给出参数的一组值，然后更新参数，使每次更新后的结构都能够让损失函数变小，最终达到最小即可）。在梯度下降法中，目标函数其实可以看做是参数的函数，因为给出了样本输入和输出值后，目标函数就只剩下参数部分了，这时可以把参数看做是自变量，则目标函数变成参数的函数了。梯度下降每次都是更新每个参数，且每个参数更新的形式是一样的，即用前一次该参数的值减掉学习率和目标函数对该参数的偏导数（如果只有1个参数的话，就是导数），为什么要这样做呢？通过取不同点处的参数可以看出，这样做恰好可以使原来的目标函数值变低，因此符合我们的要求（即求函数的最小值）。即使当学习速率固定(但不能太大)，梯度下降法也是可以收敛到一个局部最小点的，因为梯度值会越来越小，它和固定的学习率相乘后的积也会越来越小。在线性回归问题中我们就可以用梯度下降法来求回归方程中的参数。有时候该方法也称为批量梯度下降法，这里的批量指的是每一时候参数的更新使用到了所有的训练样本。

      Vectorized implementation指的是矢量实现，由于实际问题中很多变量都是向量的，所有如果要把每个分量都写出来的话会很不方便，应该尽量写成矢量的形式。比如上面的梯度下降法的参数更新公式其实也是可以用矢量形式实现的。矢量形式的公式简单，且易用matlab编程。由于梯度下降法是按照梯度方向来收敛到极值的，如果输入样本各个维数的尺寸不同（即范围不同），则这些参数的构成的等高线不同的方向胖瘦不同，这样会导致参数的极值收敛速度极慢。因此在进行梯度下降法求参数前，需要先进行feature scaling这一项，一般都是把样本中的各维变成0均值，即先减掉该维的均值，然后除以该变量的range。

     接下来就是学习率对梯度下降法的影响。如果学习速率过大，这每次迭代就有可能出现超调的现象，会在极值点两侧不断发散，最终损失函数的值是越变越大，而不是越来越小。在损失函数值——迭代次数的曲线图中，可以看到，该曲线是向上递增的。当然了，当学习速率过大时，还可能出现该曲线不断震荡的情形。如果学习速率太小，这该曲线下降得很慢，甚至在很多次迭代处曲线值保持不变。那到底该选什么值呢？这个一般是根据经验来选取的，比如从…0.0001,0.001,.0.01,0.1,1.0…这些参数中选，看那个参数使得损失值和迭代次数之间的函数曲线下降速度最快。

     同一个问题可以选用不同的特征和不同的模型，特征方面，比如单个面积特征其实是可以写成长和宽2个特征的。不同模型方面，比如在使用多项式拟合模型时，可以指定x的指数项最多到多少。当用训练样本来进行数据的测试时，一般都会将所有的训练数据整理成一个矩阵，矩阵的每一行就是一个训练样本，这样的矩阵有时候也会叫做是“design matrix”。当用矩阵的形式来解多项式模型的参数时，参数w=inv(X’\*X)\*X’\*y,这个方程也称为normal equations. 虽然X’\*X是方阵，但是它的逆不一定存在（当一个方阵的逆矩阵不存在时，该方阵也称为sigular）。比如说当X是单个元素0时，它的倒数不存在，这就是个Sigular矩阵，当然了这个例子太特殊了。另一个比较常见的例子就是参数的个数比训练样本的个数还要多时也是非可逆矩阵。这时候要求解的话就需要引入regularization项，或者去掉一些特征项（典型的就是降维，去掉那些相关性强的特征）。另外，对线性回归中的normal equations方程求解前，不需要对输入样本的特征进行feature scale（这个是有理论依据的）。

　　上面讲的函数一般都是回归方面的，也就是说预测值是连续的，如果我们需要预测的值只有2种，要么是要么不是，即预测值要么是0要么是1，那么就是分类问题了。这样我们需要有一个函数将原本的预测值映射到0到1之间，通常这个函数就是logistic function，或者叫做sigmoid function。因为这种函数值还是个连续的值，所以对logistic函数的解释就是在给定x的值下输出y值为1的概率。

　　Convex函数其实指的是只有一个极值点的函数，而non-convex可能有多个极值点。一般情况下我们都希望损失函数的形式是convex的。在分类问题情况下，先考虑训练样本中值为1的那些样本集，这时候我的损失函数要求我们当预测值为1时，损失函数值最小（为0），当预测值为0时，此时损失函数的值最大，为无穷大，所以这种情况下一般采用的是-log(h(x)),刚好满足要求。同理，当训练样本值为0时，一般采用的损失函数是-log(1-h(x)).因此将这两种整合在一起时就为-y\*log(h(x))-(1-y)\*log(1-h(x))，结果是和上面的一样，不过表达式更紧凑了，选这样形式的loss函数是通过最大释然估计(MLE)求得的。这种情况下依旧可以使用梯度下降法来求解参数的最优值。在求参数的迭代公式时，同样需要求损失函数的偏导，很奇怪的时，这时候的偏导函数和多元线性回归时的偏导函数结构类似，只是其中的预测函数一个是普通的线性函数，一个是线性函数和sigmoid的复合的函数。

　　梯度下降法是用来求函数值最小处的参数值，而牛顿法是用来求函数值为0处的参数值，这两者的目的初看是感觉有所不同，但是再仔细观察下牛顿法是求函数值为0时的情况，如果此时的函数是某个函数A的导数，则牛顿法也算是求函数A的最小值（当然也有可能是最大值）了，因此这两者方法目的还是具有相同性的。牛顿法的参数求解也可以用矢量的形式表示，表达式中有hession矩阵和一元导函数向量。

　　下面来比较梯度法和牛顿法，首先的不同之处在于梯度法中需要选择学习速率，而牛顿法不需要选择任何参数。第二个不同之处在于梯度法需要大量的迭代次数才能找到最小值，而牛顿法只需要少量的次数便可完成。但是梯度法中的每一次迭代的代价要小，其复杂度为O(n),而牛顿法的每一次迭代的代价要大，为O(n^3)。因此当特征的数量n比较小时适合选择牛顿法，当特征数n比较大时，最好选梯度法。这里的大小以n等于1000为界来计算。

　　如果当系统的输入特征有多个，而系统的训练样本比较少时，这样就很容易造成over-fitting的问题。这种情况下要么通过降维方法来减小特征的个数（也可以通过模型选择的方法），要么通过regularization的方法，通常情况下通过regularization方法在特征数很多的情况下是最有效，但是要求这些特征都只对最终的结果预测起少部分作用。因为规则项可以作用在参数上，让最终的参数很小，当所有参数都很小的情况下，这些假设就是简单假设，从而能够很好的解决over-fitting的问题。一般对参数进行regularization时，前面都有一个惩罚系数，这个系数称为regularization parameter，如果这个规则项系数太大的话，有可能导致系统所有的参数最终都很接近0，所有会出现欠拟合的现象。在多元线性回归中，规则项一般惩罚的是参数1到n（当然有的也可以将参数0加入惩罚项，但不常见）。随着训练样本的增加，这些规则项的作用在慢慢减小，因此学习到的系统的参数倾向而慢慢增加。规则项还有很多种形式，有的规则项不会包含特征的个数，如L2-norm regularization(或者叫做2-norm regularization).当然了，还有L1-norm regularization。由于规则项的形式有很多种，所以这种情形也称为规则项的common variations.

　　在有规则项的线性回归问题求解中，如果采用梯度下降法，则参数的更新公式类似（其中参数0的公式是一样的，因为规则项中没有惩罚参数0），不同之处在于其它参数的更新公式中的更新不是用本身的参数去减掉后面一串，而是用本身参数乘以（1-alpha\*lamda/m）再减掉其它的，当然了这个数在很多情况下和1是相等的，也就很前面的无规则项的梯度下降法类似了。它的normal equation也很前面的类似，大致为inv(X’\*X+lamda\*A)\*X’\*y,多了一项，其中A是一个对角矩阵，除了第一个元素为0外，其它元素都为1（在通用规则项下的情形）。这种情况下前面的矩阵一般就是可逆的了，即在样本数量小于特征数量的情况下是可解的。当为logistic回归的情况中（此时的loss函数中含有对数项），如果使用梯度下降法，则参数的更新方程中也和线性回归中的类似，也是要乘以（1-alpha\*lamda/m），nomal equation中也是多了一个矩阵，这样同理就解决了不可逆问题。在牛顿法的求解过程中，加了规则项后的一元导向量都随着改变，hession矩阵也要在最后加入lamda/m\*A矩阵，其中A和前面的一样。

　　logistic回归与多充线性回归实际上有很多相同之处，最大的区别就在于他们的因变量不同，其他的基本都差不多，正是因为如此，这两种回归可以归于同一个家族，即广义线性模型（generalized linear model）。这一家族中的模型形式基本上都差不多，不同的就是因变量不同，如果是连续的，就是多重线性回归，如果是二项分布，就是logistic回归，如果是poisson分布，就是poisson回归，如果是负二项分布，就是负二项回归，等等。只要注意区分它们的因变量就可以了。logistic回归的因变量可以是二分类的，也可以是多分类的，但是二分类的更为常用，也更加容易解释。所以实际中最为常用的就是二分类的logistic回归。

**参考资料：**

<http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=DeepLearning>

<http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial>

前面的文章已经介绍过了2种经典的机器学习算法：线性回归和logistic回归，并且在后面的练习中也能够感觉到这2种方法在一些问题的求解中能够取得很好的效果。现在开始来看看另一种机器学习算法——神经网络。线性回归或者logistic回归问题理论上不是可以解决所有的回归和分类问题么，那么为什么还有其它各种各样的机器学习算法呢？比如这里马上要讲的神经网络算法。其实原因很简单，在前面的一系列博文练习中可以发现，那些样本点的输入特征维数都非常小（比如说2到3维），在使用logistic回归求解时，需要把原始样本特征重新映射到高维空间中，如果特征是3维，且指数最高为3时，得到的系数最高维数应该是20维。但是一般现实生活中的数据特征非常大，比如一张小的可怜的灰度图片50\*50，本身就只有2500个特征，如果要采用logistic回归来做目标检测的话，则有可能达到上百万的特征了。这样不仅计算量复杂，而且因为特征维数过大容易是学习到的函数产生过拟合现象。总的来说，只有线性回归和logistic回归在现实生活中是远远不够的，因此，神经网络由于它特有的优势就慢慢被研究了。

　　神经网络模型的表达结构是比较清晰的，输入值和对应的权重相乘然后相加最终加上个偏移值就是输出了。只是数学公式比较繁琐，容易弄错。假设第j层网络有Sj个节点，而第j+1层网络有S(j+1)个节点，则第j层的参数应该是个矩阵，矩阵大小为S(j+1)\*(Sj+1)，当然了，此时是因为那个权值为1的那个网络节点没有算进去。很显然，为了方便公式的表达，神经网络中经常使用矢量化的数学公式。为什么神经网络最有学习功能呢？首先从生物上来讲，它模拟了人的大脑的功能，而人的大脑就有很强大的学习机制。其次从神经网络的模型中也可以看出，如果我们只看输出层已经和输出层相连的最后一层可以发现，它其实就是一个简单的线性回归方程（如果使输出在0~1之间，则是logistic回归方程），也就是说前面那么多的网络只是自己学习到了一些新的特征，而这些新的特征是很适合作为问题求解的特征的。因此，说白了，神经网络是为了学习到更适合问题求解的一些特征。

　　表面上看，神经网络的前一层和当前层是直接连接的，前一层的输出值的线性组合构成了当前层的输出，这样即使是有很多层的神经网络，不也只能学习到输入特征的线性组合么？那为什么说神经网络可以学习任意的非线性函数呢？其实是刚才我犯了一个本质错误，因为前一层输出的线性组合并不直接是本层的输出，而是一般还通过一个函数复合，比如说最常见的函数logistic函数（其它的函数比如双曲正切函数也是很常用的），要不然可就真是只能学习到线性的特征了。神经网络的功能是比较强大的，比如说单层的神经网络可以学习到”and”,”or”，,”not”以及非或门等，两层的神经网络可以学习到”xor”门（通过与门和非或门构成的一个或门合成），3层的神经网络是可以学习到任意函数的（不包括输入输出层）等，这些在神经网络的发展过程中有不少有趣的故事。当然了，神经网络也是很容易用来扩展到多分类问题的，如果是n分类问题，则只需在设计的网络的输出层设置n个节点即可。这样如果系统是可分的话则总有一个学习到的网络能够使输入的特征最终在n个输出节点中只有一个为1，这就达到了多分类的目的。

　　神经网络的损失函数其实是很容易确定的，这里以多分类的神经网络为例。当然了，这里谈到损失函数是在有监督学习理论框架下的，因为只有这样才能够知道损失了多少（最近有发展到无监督学习框架中也是可以计算损失函数的，比如说AutoEncoder等）。假设网络中各个参数均已学到，那么对于每个输入样本，就能够得出一个输出值了，这个输出值和输入样本标注的输出值做比较就能够得到一个损失项。由于多分类中的输出值是一个多维的向量，所以计算它的损失时需要每一维都求（既然是多分类问题，那么训练样本所标注的值也应该为多维的，至少可以转换成多维的）。这样的话，神经网络的损失函数表达式与前面的logistic回归中损失函数表达式很类似，很容易理解。

　　有了损失函数的表达式，我们就可以用梯度下降法或者牛顿法来求网络的参数了，不管是哪种方法，都需要计算出损失函数对某个参数的偏导数，这样我们的工作重点就在求损失函数对各个参数的偏导数了，求该偏导数中最著名的算法就是BP算法，也叫做反向传播算法。在使用BP算法求偏导数时，可以证明损失函数对第l层的某个参数的偏导与第l层中该节点的误差，以及该参数对应前一层网络编号在本层的输出（即l层）的输出值有关，那么此时的工作就转换成了每一层网络的每一个节点的误差的求法了（当然了，输入层是不用计算误差的）。而又可通过理论证明，每个节点的误差是可以通过下一层网络的所以节点反向传播计算得到（这也是反向传播算法名字的来源）。总结一下，当有多个训练样本时，每次输入一个样本，然后求出每个节点的输出值，接着通过输入样本的样本值反向求出每个节点的误差，这样损失函数对每个节点的误差可以通过该节点的输出值已经误差来累加得到，当所有的样本都经过同样的处理后，其最终的累加值就是损失函数对应位置参数的偏导数了。BP算法的理论来源是一个节点的误差是由前面简单的误差传递过来的，传递系数就是网络的系数。

　　一般情况下,使用梯度下降法解决神经网络问题时是很容易出错,因为求解损失函数对参数的偏导数过程有不少矩阵，在程序中容易弄错,如果损失函数或者损失函数的偏导数都求错了的话,那么后面的迭代过程就更加错了,导致不会收敛，所以很有必要检查一下偏导数是否正确。Andrew Ng在课程中告诉大家使用gradient checking的方法来检测，即当求出了损失函数的偏导数后，取一个参数值，计算出该参数值处的偏导数值，然后在该参数值附近取2个参数点，利用损失函数在这个两个点值的差除以这2个点的距离（其实如果这2个点足够靠近的话，这个结果就是导数的定义了），比较这两次计算出的结果是否相等，如果接近相等的话，则说明很大程度上，这个偏导数没有计算出错，后面的工作也就可以放心的进行了，这时候一定要记住不要再运行gradient checking，因为在运行gradient checking时会使用BP进行每层的误差等计算，这样很耗时（但是我感觉即使不计算gradient checking,不也要使用BP算法进行反向计算么？）。

　　在进行网络训练时，千万不要将参数的初始值设置成一样的，因为这样学习的每一层的参数最终都是一样的，也就是说学习到的隐含特征是一样的，那么就多余了，且效果不好。因此明智的做法是对这些参数的初始化应该随机，且一般是满足均值为0，且在0左右附近的随机。

　　如果采用同样的算法求解网络的参数的话（比如说都是用BP算法），那么网络的性能就取决于网络的结构（即隐含层的个数以及每个隐含层神经元的个数），一般默认的结构是：只取一个隐含层，如果需要取多个隐含层的话就将每个隐含层神经元的个数设置为相同，当然了隐含层神经元的个数越多则效果会越好。