一步一步学习Deep Learning (2) — Softmax Regression (Matlab)

by Lingxiao.Yang in 学习Deep Learning on 2015-04-22 | tags: Deep Learning Matlab Softmax Regression | comments

关于学习Softmax Regression

最近最火的深度神经网络模型 -- "Convolution Nueral Networks" (CNNs)中，其在顶层利用Softmax Regression (SR)来进行最终的预测过程。整个过程可以看作是利用SR进行的有监督的Fine-tuning的过程。所以，在进一步学习Deep Learning之前，个人觉得有必要去了解并且自己实现一下SR。本文主要涉及一些理论公式方面的东西。因本文的侧重点是去说明SR，为了能够方便快速的理解整个SR的过程，所以本文对于算法的实现采用了简单有效的`Matlab`工具，后续会将`Matlab`代码转换到`C++`中去以加深理解。 ***注: 为了能够方便理解，这里的SR应用在Supervised的场合，当然后续可能会讨论一些Unsupervised的情况。***

Softmax Regression理论部分理解

由于SR与另一个线性回归模型 ***Logistic Regression (LR)*** 类似，所以为了能够流畅的介绍SR，这里简单介绍一下LR模型。当然，为了较为方便的讨论，本文所涉及的回归问题一律转换为分类问题进行讨论，不过其背后的原理与逻辑基本类似。

Logistic Regression

从个人当前对机器学习不完整的理解来看，一般建立在数据之上的Supervised机器学习模型需要知道两方面的factor: **预测函数**以及**目标函数**。

- 预测函数一般是旨在告诉应用者们，如果我从这些数据中得到所设计的模型后，怎样去应用这个模型。
- 目标函数是设计者们怎样去利用现有的数据去学习到所设计的模型。

LR属于机器学习的一种，当然也应该具备上面的两者。这里对LR进行简单的定义：给定n个训练样本集合 $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$，其中输入特征 $x_i \in R^d$，其中d表示特征的纬度，而训练集合中的 y_i 表示每一个样本对应的label信息，也就是人为标定的信息。由于LR主要是针对2分类的情况，所以 $y_i \in (0, 1)$。其预测函数可以定义为：

$$h(x, w) = \frac{1}{1 + exp(-w^T x)}$$

这里的$h(x, w)$是参数为w的预测函数，就是我们需要从数据x中学到的，进而可以利用公式(1)去做后续的预测。当然，怎样去学习到参数w就是机器学习的本质性的问题了，这个本质性的问题一般通过一个目标函数实现，LR的目标函数被定义为：

$$C(w) = -\frac{1}{n}[\sum_{i=1}^{n} y_i log(h(x_i, w)) + (1 - y_i) log(1 - h(x_i, w))]$$

我们来仔细分析一下公式(2)，直接穷举y_i好了，当$y_i = 0$的时候，我们可以看到，上式中求和部分只有$(1 - y_i)log(1 - h(x_i, w))$项有作用，如果此时的预测值$h(x_i, w)$也为0，则整个目标函数为0，这里其实就是没有任何损失；相反，如果此时的预测函数为$(0, 1]$，则整个目标函数是有值，则此时认为目标函数在当前的参数w下，是有损失的。为了尽可能的降低整个目标函数的损失，需要不断的调整参数w，这个过程就是机器学习的过程。关于更多的LR信息可以参考百度以及Google的搜索。

LR的扩展 - Softmax Regression

SR可以认为是LR的一个扩展，将LR从二分类问题上扩展到多分类的情况。具体来说，我们有一组训练数据 $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$，其中输入特征$x_i$与LR的情况类似，而输出值$y_i \in \{1, 2, \ldots, k\}$，$k$为需要识别的类别数目，例如博文MNIST loading中读取的MNIST数据库有10个类别，分别是从0到9的数字，所以这里的k也就是10。有了数据之后，从上面的分析可以知道，重点就是了解SR的*预测*以及*目标*函数，以方便我们进一步的进行学习。SR的预测概率模型为$P(y = j \mid x, w)$，即在当前学习到的参数w下，预测x属于每一类的概率，而通常我们选择概率最大的一类作为最终的预测结果。完整的预测函数如下：

$$h(x_i, w) = \frac{1}{\sum_{j=1}^{k} e^{w_j^T x_i}} \begin{bmatrix} e^{w_1^T x_i} \\ e^{w_2^T x_i} \\ \vdots \\ e^{w_k^T x_i} \end{bmatrix}$$

在公式(3)中，$\sum_{j=1}^{k} e^{w_j^T x_i}$其实是每一个输入$x$的预测概率的总和，目的是归一化$x$的输出概率，使之各类别的概率之和等于1。而后面的$e^{w_k^T x}$则是$x$属于每一个类别输出概率情况。通过这样简单的扩展，SR具备了预测多类的能力。接下来要说明的是SR的目标函数：

$$C(w) = -\frac{1}{n}[\sum_{i=1}^{n} \sum_{j=1}^{k} 1\{y_i = j\} log \frac{e^{w_j^T x_i}}{\sum_{l=1}^{k} e^{w_l^T x_i}}]$$

此处$1\{\ldots\}$表示括号里面的表达式为真的时候为1，否则为0。此时关于目标函数的分析类似于LR，也可以采取穷举y_i的情况。不难看出，只要在优化的过程中找到最优的w即可。到此，基本上的SR模型介绍完毕。在针对真实的数据进行学习的时候，一般会在目标函数后面加上Regularization项，以防止参数中某些单个的值过大或者过小，从而在学习过程中造成了不必要的过拟合现象。关于过拟合以及加入Regularization项的一些解释，可以参考Bishop的书籍《Pattern Recognition and Machine Learning》中1.1节的详细解释。这里直接给出加入Regularization项后的SR：

$$C(w) = -\frac{1}{n}[\sum_{j=1}^{k} 1\{y_i = j\} log \frac{e^{w_j^T x_i}}{\sum_{l=1}^{k} e^{w_l^T x_i}}] + \frac{\lambda}{2} \sum_{i=1}^{k} \sum_{j=1}^{d} w_{ij}^2$$

λ是经验性的参数，需要通过具体的实验进行设定。好了，关于SR理论部分的解释基本完毕，关于怎样优化以及倒数的求解过程请参看实现部分。 ***注意：本文其中的大部分说明以及公式都来自UFLDL。只是做了一些简单的调整，关于更多的分析可以直接参考UFLDL。***

Softmax Regression的导数计算

这一部分主要介绍根据上面的理解，自己推到公式的过程。首先，让我们选择一种优化方法，由于本身只了解基本梯度下降以及随机梯度下降，这里选择**基本梯度下降**算法进行目标函数的优化过程。关于梯度下降的优化解释，可以看Andrew Ng的Machine Learing的课程。用**基本梯度下降**来进行迭代更新的话，需要知道目标函数的梯度。首先，我们来简单的求解一下公式(5)的梯度，计算梯度的时候我们将log看成ln，方便求导，但最终的结果不变。

先求出非Regularization项"$1\{y_i = j\} log \frac{e^{w_j^T x_i}}{\sum_{l=1}^{k} e^{w_l^T x_i}}$"的导数:

$$1\{y_i = j\} log \frac{e^{w_j^T x_i}}{\sum_{l=1}^{k} e^{w_l^T x_i}} = 1\{y_i = j\}[log(e^{w_j^T x_i}) - log(\sum_{l=1}^{k} e^{w_l^T x_i})]$$

上式利用$log\frac{a}{b} = log(a) - log(b)$展开，接着对$w_j$进行求导得：

$$\begin{aligned} & 1\{y_i = j\}[x_i - \frac{e^{w_j^T x_i}}{\sum_{l=1}^{k} e^{w_l^T x_i}} x_i] \\ &= x_i(1\{y_i = j\} - \frac{e^{w_j^T x_i}}{\sum_{l=1}^{k} e^{w_l^T x_i}}) \\ &= x_i(1\{y_i = j\} - p(y_i = j \mid x_i, w) \end{aligned}$$

上式我们将log看成ln，则$ln'x = \frac{1}{x}$， 另外，每一次的求导其实只是针对w中的某一项j，所以其他的w的非j项都为常数，所以求导以后都为0。而$e^x = e^x$，这个我觉得大家都知道吧。最后，我们看到$e^{w_j^T x_i}$ 除以 $\sum_{l=1}^{k} e^{w_l^T x_i}$其实就是$p(y_i = j \mid x_i, w)$。则关于非Regularization项的导数求解完成。而Regularization的导数相对比较简单，则整个目标函数关于w_j的导数为：

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_{i=1}^{n} [x_i(1\{y_i = j\} - p(y_i = j \mid x_i, w)] + \lambda w_j$$

好了导数也求好了，接下来利用`Matlab`来实现整个过程。

Softmax Regression Matlab实现

其实这一部分的实现测试可以在真实的MNIST数据上去做，但为了很好的验证上面的理解以及推导过程，仅仅利用虚拟的数据上跑。那么，首先需要的是产生一些数据：

```
clear all; clc; close all;

% params
params.maxIter = 100;
params.numData = 45;
params.numClasses = 3;
params.lambda = 1e-4;
params.learningRate = 0.01;

% generate data
train.X{1} = randi([1 100], 2, params.numData);
train.Y{1} = ones(1, params.numData) * 1;

train.X{2} = randi([200 255], 2, params.numData);
train.X{2}(1, :) = train.X{2}(1, :) - 180;
train.Y{2} = ones(1, params.numData) * 2;

train.X{3} = randi([180 255], 2, params.numData);
train.Y{3} = ones(1, params.numData) * 3;

train.X = cat(2, train.X{:});

% normalize data
train.X = double(train.X)/255;

% add bias so that the 1st row is the bias for each category classifier
% bias is very important in this example, no bias, wrong result
train.X = [ones(1, size(train.X,2)); train.X];
train.Y = cat(2, train.Y{:});
```

具体的`Matlab`代码的语法就不一句一句解释了，总之这里是为了产生3类数据，每一类数据45个点，特征纬度为2，即(x, y)的坐标值。关于加入**bias**的那一句，在这里的例子中，如果没有加入这一项的话，最终的结果会错误，但是我将SR用在真实的一些数据上，这个**bias**的影响并没有特别的大。例如在MNIST库上跑出来的结果是**92.3%** vs **92.2%**（bias vs no bias）。然后我们来初始化需要优化的参数w：

```
% w parameters initialization
w = randn(d, params.numClasses);

% convert real value label to label matrix
% 1, 2, 3 -> 001, 010, 100.
l = bsxfun(@(y, ypos) (y == ypos), train.Y', 1:params.numClasses);
```

上面第二部分主要是将原始的label数据转换成matrix的形式，例如在这个例子中，原始的label是[1, 2, 3, ..., 1]，那将这些label转换成[001, 002, 003, ..., 001]表示，即每一个label转换成一个向量，一般在写多分类的情况下，这么做的好处也是为了能够充分利用向量或者矩阵的运算以提高运算速度，无论是在`C++`还是在`Matlab`下，利用现有的一些矩阵运算要比自己写的快很多，特别是针对数据量比较大的时候，速度的提升就很明显了。

好了，这里基本上可以认为数据以及前期工作准备完毕，接下来就是要根据上面推导的公式一一计算并且更新参数&w&了。首先，我们来看看SR预测过程：

```
rsp = w' * train.X;
rsp = bsxfun(@minus, rsp, max(rsp, [], 1));
rsp = exp(rsp);
prob = bsxfun(@rdivide, rsp, sum(rsp));
```

上面4行代码主要是计算公式(3)，因为`exp`对于特别大的数并不十分稳定，所以在计算输出后，需要将每一输出减去一个数值，这里减去的是每一个样本概率的最大值。关于减去这个样本的值以后会不会影响最终的结果，这一部分在UFLDL[1]里面有具体的分析。接下来求出目标函数的导数，并进行参数的更新：

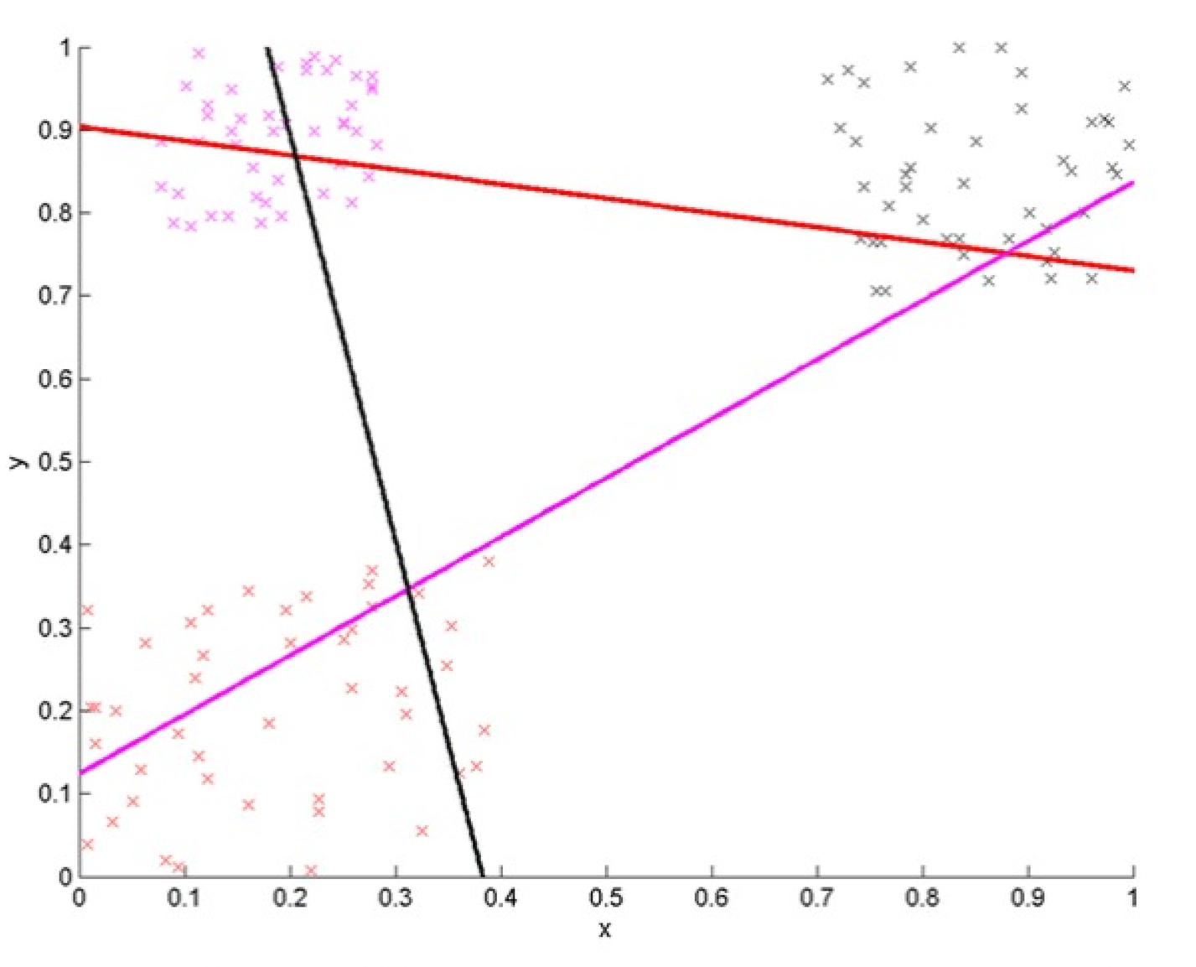
```
% compute gradient based on current probability
g = - train.X * (l - prob') / n + params.lambda * w;

% update w
w = w - params.learningRate * g;
```

其中，梯度的求导对应公式(8)，更新过程其实就是普通的梯度下降算法。最后一部分是求出目标函数的值如下，对应公式(5)：

```
% compute cost
logProb = log(prob);
idx = sub2ind(size(logProb), train.Y, 1:size(logProb, 2));
cost = - sum(logProb(idx)) / n + params.lambda * 0.5 * sum(sum(w.^2));
```

其实在优化的过程中，并没有用到目标函数的值，而一般我们在优化的过程中求这一部分的目的是为了能够方便的看出整个目标函数的变化趋势，比如，在最小化过程中，如果目标函数的变化大体趋势是减小，则证明我们的代码基本问题不大。在迭代一段时间后(本例设定迭代次数为200)，最终对于训练数据的识别率为100%。这里贴出一个GIF动态图来说明整个优化过程中w的变化过程：



上图中每一条线的颜色与点的类别一致，从上图可以看出，每一条线在迭代更新的过程中都力求将自己的类别与其他的类别尽可能的分开。训练完后，对于新来的点如何判断如下：

```
rsp = w' * newX;
[~, prediction] = max(rsp, [], 1);
```

首先计算新样本的响应，然后可以直接找到最大的值，而最大值的类别就是判断的类别。因为概率公式(3)是单调递增函数，我们在计算出输出响应后，则可以不用计算概率值而直接得到label。

后记

这篇BLOG写了一周的时间，主要还是自己对于softmax理解不够深刻，加上编程中遇到了**bias**的一个小bug。不过正文还是详细讲述了自己通过UFLDL学习softmax的过程以及理解。接下来一篇论文将会介绍自己在利用C++学习softmax的代码以及将该代码运用到MNIST数字库的过程。

Reference

[1] http://ufldl.stanford.edu/wiki/index.php/Softmax%E5%9B%9E%E5%BD%92

[2] Bishop C M. Pattern recognition and machine learning[M]. New York: springer, 2006.

Comments

york_hust • 3个月前
公式7将x_i提出的时候有误

Ein Verne • 1年前
感谢分享，总结的特别棒学习了