

— “笔记”

April 30, 2017; rev. Saturday 16th September, 2017

Qian Wang

第一章 Adaboost

Introduction

本章内容来自于网络以及张志华老师的就《机器学习导论》课程

本章基本内容就是给定一堆弱分类器，然后通过各种组合组成一个人强的分类器

1.1 离散的 Adaboost

离散的 AdaBoost 算法步骤：

w 表示给数据的权值， α 表示给分类器的权值

1. start with weights $w_i = \frac{1}{N} \quad i = 1 \dots N$

2. repeat for $m=1$ to M

- 使用输入数据训练一个分类器 $G_m(x) \in (-1, 1)$
- 计算误差 err :

$$E_w[I(y_i \neq G_m(x_i))] = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$

- 输出 $\alpha_m = \frac{1}{2} \log(\frac{1-err}{err})$, 从这里可以看出分类器的误差越大，权值越小
- set $w_i = \frac{w_i}{Z_m} \exp(\alpha_m I(y \neq G_m(x)))$, 其中 Z_m 是规范化因子,
 $Z_m = \sum_{i=1}^N w_i \exp(-\alpha_m y_i G_m(x_i))$ 如果一个数据点分类错了，那么给这个点的权值大一点。

3. 这样我们就得到了 $G(x) = \text{sign}[\sum_{m=1}^M \alpha_m G_m(x)]$

1.2 Forward Stagewise Additive Modeling

考虑加法模型 (additive model)

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m) \quad (2.1)$$

其中, $b(x; \gamma_m)$ 为基函数, γ_m 为基函数的参数, β_m 为基函数的系数。

在给定训练数据以及 Loss Function 的情况下, 相当于一个加法模型 $f(x)$ 相当于一个 minimize Loss Function 的问题:

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^N L(y_i, \sum_{m=1}^M \beta_m b(x; \gamma_m)) \quad (2.2)$$

从前向后, 每一步值学习一个基函数及其系数, 即每一步只需优化如下的损失函数:

$$\min_{\beta, \gamma} \sum_{i=1}^N L(y_i, \sum_{m=1}^M \beta b(x; \gamma)) \quad (2.3)$$

前向分布算法:

输入: 训练数据 $T = (x_1, y_1), \dots, (x_N, y_N)$, Loss Function $L(x, f(x))$, 基函数 $b(x; \gamma)$

输出: 加法模型 $f(x)$

1. 初始化 $f_0(x) = 0$

2. repeat for $m=1$ to M

- 计算参数 β_m, γ_m

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x; \gamma)) \quad (2.4)$$

- 更新 $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

3. 得到加法模型

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m) \quad (2.5)$$

第二章 Logistic Regression

Introduction

本章节内容主要来自林轩田的《机器学习课程》

2.1 离散的 Adaboost

没有风险

第三章 Deep Learning

Introduction

本章节内容主要是来自于深度学习中遇到的一些坑以及问题

3.1 一些小的 trick

1. 一定要对数据进行归一化

2. 训练可能会出现 loss 一开始迅速下降，然后稳定在一定的值上，这时候不要以为网络已经收敛了，有时候会出现一个拐点，即在训练很后面的时候会重新出现一个新的下降的区间，这时候网络基本上会收敛。但是不排除会出现多的拐点的情况。结论：训练过程中一定要耐心耐心再耐心。

3.2 weight decay

在损失函数中，weight decay 是放在正则项（regularization）前面的一个系数，正则项一般指示模型的复杂度，所以 weight decay 的作用是调节模型复杂度对损失函数的影响，若 weight decay 很大，则复杂的模型损失函数的值也就大。我所理解的 weight decay 就是一个调节正则化项的系数，增大则对正则化项的依赖会更高，不容易过拟合，反之则反之。

利用 weight decay 给损失函数加了个惩罚项，使得在常规损失函数值相同的情况下，学习算法更倾向于选择更简单（即权值和更小）的 NN。是一种减小训练过拟合的方法。

Weight decay is equivalent to L2 regularizer.

在训练神经网络的时候，可以先设置 weight decay 为 0，然后使网络过拟合，查看测试结果。结果正确后，设置 weight decay 为大一点的值，即加入正则化项，这样可以防止过拟合现象。具体大小需要在网络中调试。

遇到的问题：在训练 U-net 的时候，出现训练结果全是黑图的情况。原因是 weight decay 设置过大，导致网络结果没有过拟合。解决方案：设置 weight decay 为更小的值。

3.3 momentum

momentum 是梯度下降法中一种常用的加速技术。对于一般的 SGD，其表达式为 $x \leftarrow x - \alpha * dx$, x 沿负梯度下降。而带 momentum 项的 SGD 则写成如下形式： $v = \beta * v - \alpha * dx$, $x \leftarrow x + v$ ，其中 β 是 momentum 系数，通俗的理解上面式子就是，如果上一次的 momentum（即 v ）与这一次的负梯度方向是相同的，那这次下降的幅度就会加大，所以这样做能够达到加速收敛的过程。

神经网络的训练过程（也就是梯度下降法）是在高维曲面上寻找全局最优解的过程（也就是寻找波谷），每经过一次训练 epoch，搜寻点应该更加靠近最优点所在的区域范围，这时进行权重衰减便有利于将搜寻范围限制在该范围内，而不至于跳出这个搜索圈，反复进行权重衰减便逐渐缩小搜索范围，最终找到全局最优解对应的点，网络收敛。momentum 是冲量单元，也就是下式中的 m ，作用是有助于训练过程中逃离局部最小值，使网络能够更快速地收敛，也是需要经过反复地 trial and error 获得的经验值

主要作用：防止陷入局部最小值

第四章 Caffe

Introduction

本章节内容主要是关于 caffe 框架的一些知识。

4.1 lmdb

lmdb 格式的文件在使用 Python 程序进行生成的时候，如果需要重复生成，则需要先删除原来的。否则会在原先的 lmdb 上重新添加文件。

4.2 net protobuf

此部分内容有待添加

4.3 solver

solver 文件为 caffe 的训练参数的文件，主要存储一些训练的超参数

运行代码为： `caffe train -solver=*solver.prototxt`

一个例子：

```
1  train_net: "lenet_train.prototxt"
2  test_net: "lenet_test.prototxt"
3      test_iter: 100
4      test_interval: 500
5
6      base_lr: 0.01
7      lr_policy: "fixed"
8
9      momentum: 0.9
```

```

10         type: SGD
11         weight_decay: 0.0005
12         display: 100
13         max_iter: 20000
14         snapshot: 5000
15         snapshot_prefix: "models"
16         solver_mode: CPU

```

下面来一个一个解释这些程序的意思

1.

`train_net: "lenet_train.prototxt"`

`test_net: "lenet_test.prototxt"`

这两行用于定于训练网络和测试网络，可以是同一个网络，用 `net: train_test.prototxt` 来表示，为上一节的内容。注意：文件的路径要从 `caffe` 的根目录开始，其他所有的配置都是这样的。

2. 接下来 `test_iter` 表示一次训练需要加载多少个数据，训练中的 `batch size` 是一致的；`test_ival` 表示经过多少此训练的 `Iteration` 后进行一次测试，如 500 表示没经过 500 个 `Iteration` 进行一次测试。另外，如果网络不想进行测试的话，可以在 `solver` 文件中加入如下的参数 `test_initialization: false` 这样不管经过多少次的 `Iteration` 都不会进行测试。

3. 有关于 `learning rate` 的东西

`base_lr` 表示初始的一个 `learning rate`，如果 `lr_policy` 如果设置为 `fixed`，训练过程中会一直维持这个 `learning rate` 不再改变，其他的都是会在训练过程中逐渐变化的。

`lr_policy` 可设置的值如下所示：

- `fixed`: 保持 `base_lr` 不变
- `step`: 如果设置为 `step`，则还需要设置一个 `stepsize`，返回 $base_lr * gamma^{\lfloor \frac{iter}{stepsize} \rfloor}$ ，其中 `iter` 表示当前的迭代次数。如设置 `gamma=0.9`，`stepsize=100` 表示在第一百次迭代后 `learning rate` 下降到原来的 0.9 倍
- `exp`: 返回 $base_lr * gamma^{iter}$ ，`iter` 为当前迭代次数
- `inv`: 如果设置为 `inv`，还需要设置一个 `power`，返回 $base_lr * (1 + gamma * iter)^{-power}$
- `multistep`: 如果设置为 `multistep`，则还需要设置一个 `stepvalue`。这个参数和 `step` 很相似，`step` 是均匀等间隔变化，而 `multistep` 则是根据 `stepvalue` 值变化
- `poly`: 学习率进行多项式误差，返回 $base_lr(1 - iter/max_iter)^{power}$
- `sigmoid`: 学习率进行 `sigmoid` 衰减，返回 $base_lr(1/(1 + exp(-gamma * (iter - stepsize))))$

需要设置参数的数量随着 lr policy 的不同而有所变化。如设置为 fixed 则不需要添加任何参数，设置为 step 则需要添加 gamma 和 stepsize 两个参数，设置为 step 的策略后 solver 配置如下所示：

```

1      base_lr: 0.01
2      lr_policy: "step"
3      gamma: 0.9
4      stepsize: 100

```

4. 对于 momentum，一般取值在 0.5–0.99 之间。通常设为 0.9，momentum 可以让使用 SGD 的深度学习方法更加稳定以及快速。详细的资料，参考 Hinton 的论文《A Practical Guide to Training Restricted Boltzmann Machines》

5. type:SGD

表示优化算法，总共有六种：SGD、AdaDelta、AdaGrad、Adam、NAG、RMSprop

6.weight_decay 为权重衰减项，详细的内容已经在上一章解释过了。

7.display: 100 表示每训练 100 个 Iteration 显示一次 loss

8.max_iter:2000 表示最大的迭代此时为 2000

9.

snapshot:500

snapshot_prefix : "models"

表示每训练 500 个 Iteration，保存一次网络的参数数据，保存路径为 models。同时会保存另外一个 solverstate 文件，以便下次训练的时候可以从这一步继续训练。

10.solver_mode: CPU 设置运行模式为 CPU

Bibliography