Code Reading Report of Caffe - Version 1

计 45 孟垂正 2013010952

May 2, 2016

1 Caffe 简介

Caffe 是一个基于易表达性, 速度和通用性开发的深度学习框架. 它由 Berkeley Vision and Learning Center (BVLC) 开发并由社区维护. 贾扬清在 UC Berkeley 读博期间开创了此项目 [1].

2 编译, 安装与运行

本节内容主要参考官网的安装指南 [2], 同时结合了个人实践. 以下操作均在 Arch Linux[1] 上进行, 与指南中 Ubuntu 下的操作略有差异.

2.1 依赖库安装

Caffe 主要用到的依赖库有:

- protobuf(Google 出品的一种数据序列化工具)
- leveldb(Google 出品的一种键值存储工具)
- snappy(Google 出品的一种快速压缩/解压缩工具)
- opencv(著名的计算机视觉计算库)
- hdf5-serial(支持多种数据类型的数据组织结构和文件格式)
- boost(著名 C++ 增强库)
- protobuf-compiler(protobuf 的编译器)
- BLAS(可从 atlas. OpenBLAS 或者 MKL 中选择一种, 以下以 atlas 为例)
- gflags(提供命令行参数管理功能的库)
- glog(提供 Log 功能的库)
- CUDA(可选,NVIDIA 显卡的 GPU 计算库, 若 CPU-only 则无需安装)
- Python(可选, 若需要 Caffe 的 Python 接口)

[1] Arch Linux 是一个通用的 i686/x86-64 GNU/Linux 发行版.Arch 采用滚动升级模式, 尽全力提供最新的稳定版软件. 与 Ubuntu 不同,Arch Linux 通过其包管理工具 pacman 安装的大部分软件包附带有必要的头文件和动态链接库, 因此一般无需安装相应的 libxxx-dev.

Arch Linux 下的安装指令为:

```
sudo pacman -S protobuf leveldb snappy opencv hdf5 boost boost-libs
sudo pacman -S atlas-lapack-base
sudo pacman -S gflags google-glog
sudo pacman -S cuda python
```

2.2 编译与安装

从 https://github.com/BVLC/caffe.git 获取源码并切换到对应目录.

```
cd ~
git clone https://github.com/BVLC/caffe.git
cd ./caffe
```

修改配置文件并进行编译. 如需要开启 CUDNN, 开启 CPU-only 模式, 更改默认 BLAS 库 (atlas) 或者使用 Python 接口, 在 Makefile.config 中参照注释进行必要修改即可.

```
cp Makefile.config.example Makefile.config
# Adjust Makefile.config (for example, if using Anaconda Python, or if
    cuDNN is desired)
make all -j4
make test
make runtest
```

runtest 结束后可检验安装是否成功.

2.3 运行 Demo

安装成功后可尝试运行 Caffe 附带的 LeNet Demo.LeNet-5[3] 是由 Y.LeCun 等人设计的用于识别手写和打印字符的卷积神经网络. 其主要结构如下图:

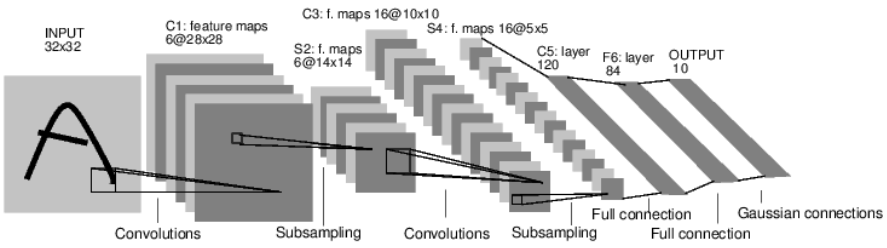

Figure 1: LeNet-5 的结构图

切换到 caffe 根目录并执行脚本获取数据集. 然后执行脚本进行训练.

```
cd ~/caffe
./data/mnist/get_mnist.sh
./examples/mnist/create_mnist.sh
./examples/mnist/train_lenet.sh
```

最终结果如下图.

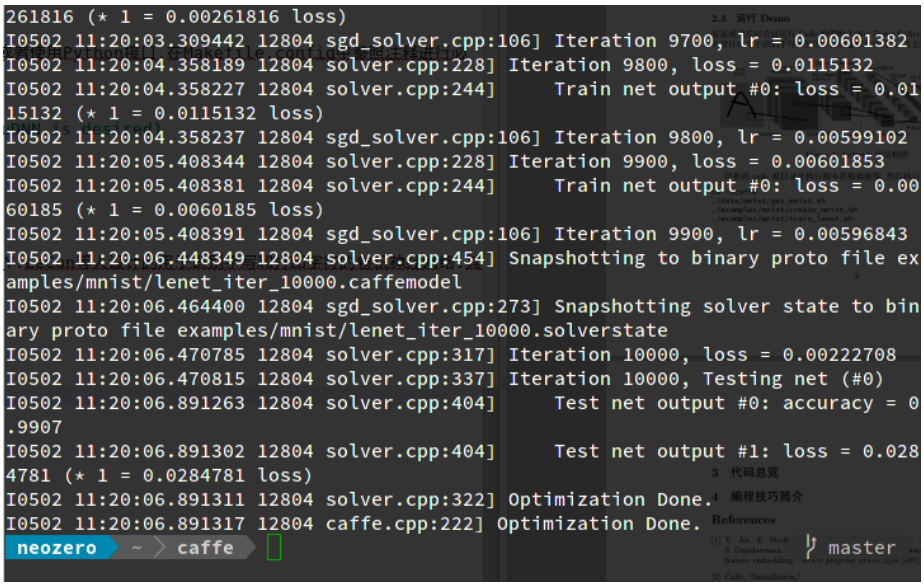

Figure 2: LeNet 的训练结果

3 系统总览

Caffe 是一个用于搭建深度神经网络的框架. 为了了解其功能, 此处有必要对深度神经网络做简要的介绍.

3.1 深度神经网络

深度神经网络是在神经网络的基础上发展而来. 由于篇幅限制, 此处对神经网络不再过多介绍, 读者如有需要可参考任何一本机器学习/人工智能/数学建模方面的基础书籍. 概括而言, 初级的神经网络接受 n 个变量的输入, 经过输入层-隐层-输出层的路径, 在各层的权重和激励函数作用下得到 m 个输出变量. 而训练目标即为求解最优的权重值组合使得输出与真实值之间的误差最小.

现实应用中, 这种简单的神经网络结构表现的好坏与输入的选择密切相关. 以围棋为例, 若欲训练出一神经网络, 使之接受当前局面并输出下一步落子位置, 则简单地将 < 所有棋盘位置上的落子情况, 下一步落子位置 > 组成的训练数据作为输入将不能得到理想的结果. 在深度学习出现之前, 通常需要人类专家结合先验知识进行输入的合理选择和设计 (称为"feature engineering")[4], 而这并不容易.

深度学习的出现, 使得特征学习成为可能. 典型的深度学习模型, 即为很多层的神经网络 (存在多个隐层), 若采用 BP 神经网络的训练方法, 计算开销将无法忍受, 并且计算结果常常会发散而无法达到稳定. 一种节省开销的方法为 CNN(卷积神经网络), 例如 Figure 1中的 LeNet-5. 网络的输入为图像的各个像素灰度值, 经过多个卷积层-采样层对输入进行加工后得到卷积层数据, 最后输入到一个普通神经网络进行输出. 在训练过程中, 同一层的每个神经元均采用相同的权值, 从而需要计算的参数数目大幅减少.

3.2 Caffe 提供的功能

作为一个深度学习框架,Caffe 提供了以下基本功能来帮助使用者较容易地搭建自己的深度神经网络.

1. 易于编辑的表达方式 模型和优化算法均用纯文本 (.protobuf) 定义而非固化在代码中.
2. 高速计算 支持使用 GPU 加速计算.
3. 扩展性 使用者可方便地根据需求添加自己需要的扩展功能.

4 系统结构设计

Caffe 实现了建模与求解的解耦, 故其系统结构可从这两方面分析.

4.1 建模

建模方面,Caffe 采用的网络结构范式为:Blob-Layer-Net 三层结构.

1. Blob.Blob 为 Caffe 中数据的基本单元, 以数组形式存储数据, 提供底层的内存管理功能.
2. Layer.Layer 对应深度神经网络中" 层" 的概念, 是构成整个网络的基础构件, 进行网络中的计算与数据处理.
3. Net.Net 是多个 Layer 的组合, 管理网络中各层的连结关系.

4.2 优化

求解方面,Caffe 实现了 SGD(Stochastic Gradient Descent) 等常用优化方法, 从 Net 中接收当前的代价函数和梯度, 输出各参数的更新量. 求解器还提供检验训练效果和保存训练阶段性成果的功能.

5 数据设计

Caffe 通过 Blob-Layer-Net 三层结构存储输入数据和配置文件, 使用 Solver 对模型进行优化求解.

5.1 数据描述

由于神经网络的结构差异很大, 为尽可能简单地说明功能, 以单层感知机为例说明 Caffe 处理数据的流程.

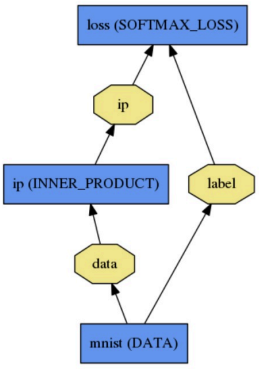

Figure 3: 单层感知机在 Caffe 中的结构示意图 [5]

5.1.1 外部数据输入网络

Caffe 采用一种专门的 Layer-DataLayer 进行数据的读取. 通过使用相关库,Caffe 支持读取 leveldb/lmdb 等数据库, 内存数据,HDF5 类型数据和图片格式数据. 使用时通过 protobuf 配置文件为 DataLayer 设置 2 个 top Blob(data 和 label) 作为该层的输出.Figure 3中底部的 mnist 层即为 DataLayer.

DataLayer 通过内部的 DataReader 类将数据读进内存并传入输出 Blob.DataReader 类通过一个 map 为每个数据源维护一个用于读取此数据源的 Body. 每个 Body 为当前需要该数据源的所有 Net 维护一个 QueuePair 队列, 其中 QueuePair 同样是一个队列, 用于依次存放解析得到的数据.Body 内部开启一个主循环线程将数据不断读入每个网络所在的队列. 如此设计主要原因是适应同时训练多个网络的情况.

当一个 Net 中的 DataLayer 需要读取数据时, 它通过 map 寻找到读取所需数据源的 Body, 并从相应队列中取出数据, 根据数据种类将其分为 data 和 label 并分别传至 top 的 Blob 中.

5.1.2 数据在网络中的流动

除了数据层外,Net 中的每一个 Layer 均需定义其输入 (bottom Blobs) 和输出 (top Blobs).Figure 3中的 ip 层 (全连接层) 的输入为 data, 输出为 ip. 有必要注意的是,Layer 内部同样采用 Blob 存储网络权重等结构信息.

每个 Layer 需要实现 2 类函数:Forward 和 Backward(实际上每类还要分为 cpu 和 gpu, 此处只讨论 cpu 编程), 分别对应 BP 神经网络的前向传播和反向传播. 这类函数中,Layer 通过 Blob 提供的接口获取其中数据的指针并进行运算. 以 Forward 函数为例, 其形式一般为 $f(X, W, Y)$, 其中 X 为 bottom Blob 的数据,W 为 Layer 中的权重, 偏移量等数据,Y 为输出数据.$f(X, W, Y)$ 为矩阵运算函数,Caffe 在此处直接调用安装时指定的 BLAS 库接口进行计算.

为了适应不同的传播函数,Caffe 在此处大量运用了 C++ 的继承和多态特性, 也为使用者开发自己需要的 Layer 提供了方便. 开发者需要继承 Layer 类并实现其中的初始化, 正/反向传播函数.

5.1.3 数据从网络中输出

Solver 的配置文件中可设置每过一定的训练间隔即可将当前最优的网络保存.Solver 将 Net 中所有 Layer 的权重和连接关系保存为二进制文件以便于使用和继续训练.

6 用户接口

Caffe 提供了可执行程序 caffe 用于网络的训练, 检验和时间测试. 进行训练的结果见 Figure 2.

```
# train LeNet
caffe train -solver examples/mnist/lenet_solver.prototxt
# score the learned LeNet model on the validation set as defined in the
# model architeture lenet_train_test.prototxt
caffe test -model examples/mnist/lenet_train_test.prototxt -weights
    examples/mnist/lenet_iter_10000.caffemodel -gpu 0 -iterations 100
# time LeNet training on CPU for 10 iterations
caffe time -model examples/mnist/lenet_train_test.prototxt -iterations
    10
```

Caffe 也提供了 Python 接口, 需要编译 pycaffe 并安装.

7 编程技巧

7.1 继承与多态的运用

OOP 中的继承与多态在 Caffe 中得到大量运用. 以 Layer 类为例, 其中 LayerSetup(),Reshape(),Forward(),Backward() 等成员函数均设置为虚函数. 由于神经网络中 Layer 用到的运算繁多, 因此需要借助虚函数在其子类中实现各种运算. 这一特性也方便了新 Layer 的开发, 增强了系统的通用性.

7.2 拷贝构造函数和赋值运算符的禁用

在 caffe/common.hpp 中 Caffe 通过宏的形式禁用了某些类的拷贝构造函数和赋值运算符.

```
// Disable the copy and assignment operator for a class.
#define DISABLE_COPY_AND_ASSIGN(classname) \
private:\
  classname(const classname&);\
  classname& operator=(const classname&)
```

通过禁用二者可以使得所有以该类为参数的函数采用传引用方式, 避免了传值方式带来的额外开销, 对于 Caffe 中内部存储较多数据的结构其意义更为明显.

7.3 命名空间的合理使用

在头文件中使用命名空间很容易造成其使用的混乱, 例如:

```
// header.h
...
using namespace caffe;
...
```

多次 include 之后将难以确定某段代码所使用的命名空间. 但完全不使用命名空间会造成很多不方便, 因此 Caffe 采用限定其作用域的方法解决这一问题.

```
// header.h
...
namespace caffe {
    ...
} // namespace caffe
```

References

[1] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[2] http://caffe.berkeleyvision.org/installation.html, "Installation,"

[3] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks,"

[4] Z. Zhou, *Machine Learning*.

[5] http://caffe.berkeleyvision.org/tutorial/net_layer_blob.html, "Blobs, layers, and nets: anatomy of a caffe model,"