8、Code Optimization with OpenMP

Caffe的设计将数据抽象为四维的Blob，将所有对数据的操作（包括卷积、全连接、Pooling、激活函数、Softmax）抽象为层（Layer），Blob数据在Layer中被处理，在Layer间流动。各种Layer的实现存放在src/caffe/layers/下，CPU版本的源文件后缀为.cpp，GPU版本的源文件后缀为.cu（NVIDIA CUDA通用计算技术）。本文在CPU上执行OpenMP并行化优化，故优化对象主要为src/caffe/layers/下的各.cpp文件。

8.1、热点分析

为了确定优化目标，我们需要分析Caffe中占资源较高的热点函数。

根据Intel公司对Caffe的完整基准测试表格（测试基于CIFAR-10网络）【图、来源请求】，Caffe运行时占用计算资源最大、运行总时间最长的几个热点，由高到低分别为：

·gemm\_omp\_driver\_v2

选择Intel的MKL库作为Caffe底层BLAS计算库时，该函数是MKL库libmkl\_intel\_thread.so中的一个函数。GEMM为General Matrix-Matrix的缩写，即“通用矩阵-矩阵相乘”。如前文所述，Caffe将难以直接优化的卷积层的计算过程以线性代数矩阵相乘的形式表达，而全连接层计算过程本身即可写作矩阵相乘形式，故该函数是Caffe进行卷积、全连接计算的核心函数，占用时间最多。

·caffe::im2col\_cpu<float>

该函数名称im2col\_cpu是“image to column on cpu”的缩写，即“从特征图图像到列向量的CPU操作”。如前文所述，Caffe将滤波器对图像的卷积表达为矩阵与列向量相乘的形式，二维图像需要转换到可供稀疏矩阵相乘的列向量。该过程将图像原地展开，补0填充为非常长的列向量，运算量较大。

·caffe::PoolingLayer<float>::Forward\_cpu

该函数为Caffe执行Pooling层前向计算的函数。Pooling是一个对特征图下采样降维的过程，在特征图较大、特征图频道数较多时，该层的运算量不可忽略。

·caffe::col2im\_cpu<float>

上一个函数的逆操作，将列向量转回图片。运算量同理。

8.2、函数调用关系分析

·gemm\_omp\_driver\_v2函数

在BLAS库选用OpenBLAS时，该函数对应cblas\_sgemm函数。分析调用栈：

> caffe-time.bin!caffe::caffe\_cpu\_gemm<float>(const CBLAS\_TRANSPOSE TransA, const CBLAS\_TRANSPOSE TransB, const int M, const int N, const int K, const float alpha, const float \* A, const float \* B, const float beta, float \* C) 行 21 C++

caffe-time.bin!caffe::BaseConvolutionLayer<float>::forward\_cpu\_gemm(const float \* input, const float \* weights, float \* output, bool skip\_im2col) 行 271 C++

caffe-time.bin!caffe::ConvolutionLayer<float>::Forward\_cpu(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 34 C++

caffe-time.bin!caffe::Layer<float>::Forward(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 460 C++

caffe-time.bin!caffe::Net<float>::ForwardFromTo(int start, int end) 行 550 C++

caffe-time.bin!caffe::Net<float>::Forward(float \* loss) 行 570 C++

caffe-time.bin!time() 行 361 C++

caffe-time.bin!main(int argc, char \* \* argv) 行 444 C++

> caffe-time.bin!caffe::caffe\_cpu\_gemm<float>(const CBLAS\_TRANSPOSE TransA, const CBLAS\_TRANSPOSE TransB, const int M, const int N, const int K, const float alpha, const float \* A, const float \* B, const float beta, float \* C) 行 16 C++

caffe-time.bin!caffe::InnerProductLayer<float>::Forward\_cpu(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 92 C++

caffe-time.bin!caffe::Layer<float>::Forward(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 460 C++

caffe-time.bin!caffe::Net<float>::ForwardFromTo(int start, int end) 行 550 C++

caffe-time.bin!caffe::Net<float>::Forward(float \* loss) 行 570 C++

caffe-time.bin!time() 行 361 C++

caffe-time.bin!main(int argc, char \* \* argv) 行 444 C++

其中，caffe::caffe\_cpu\_gemm<float>为Caffe封装的矩阵相乘函数。由调用栈可分析出，卷积层前向计算与BLAS函数的调用关系为：ConvolutionLayer调用其基类BaseConvolutionLayer的前向计算函数，BaseConvolutionLayer的前向计算成员函数再调用BLAS；全连接层前向计算相应的调用关系则为：InnerProductLayer前向计算成员函数直接调用BLAS函数。

综上，需要优化base\_conv\_layer.cpp。

·im2col函数

分析调用栈：

> caffe-time.bin!caffe::im2col\_cpu<float>(const float \* data\_im, const int channels, const int height, const int width, const int kernel\_h, const int kernel\_w, const int pad\_h, const int pad\_w, const int stride\_h, const int stride\_w, const int dilation\_h, const int dilation\_w, float \* data\_col) 行 24 C++

caffe-time.bin!caffe::BaseConvolutionLayer<float>::conv\_im2col\_cpu(const float \* data, float \* col\_buff) 行 106 C++

caffe-time.bin!caffe::BaseConvolutionLayer<float>::forward\_cpu\_gemm(const float \* input, const float \* weights, float \* output, bool skip\_im2col) 行 264 C++

caffe-time.bin!caffe::ConvolutionLayer<float>::Forward\_cpu(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 34 C++

caffe-time.bin!caffe::Layer<float>::Forward(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 460 C++

caffe-time.bin!caffe::Net<float>::ForwardFromTo(int start, int end) 行 550 C++

caffe-time.bin!caffe::Net<float>::Forward(float \* loss) 行 570 C++

caffe-time.bin!time() 行 361 C++

caffe-time.bin!main(int argc, char \* \* argv) 行 444 C++

需要优化im2col.cpp。

·Pooling Layer前向计算

分析调用栈：

> caffe-time.bin!caffe::PoolingLayer<float>::Forward\_cpu(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 129 C++

caffe-time.bin!caffe::Layer<float>::Forward(const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & bottom, const std::vector<caffe::Blob<float> \*,std::allocator<caffe::Blob<float> \*> > & top) 行 460 C++

caffe-time.bin!caffe::Net<float>::ForwardFromTo(int start, int end) 行 550 C++

caffe-time.bin!caffe::Net<float>::Forward(float \* loss) 行 570 C++

caffe-time.bin!time() 行 361 C++

caffe-time.bin!main(int argc, char \* \* argv) 行 444 C++

需要优化pooling\_layer.cpp。

·col2im函数

同im2col函数，需要优化im2col.cpp。

8.3、对卷积层的OpenMP并行化优化

卷积层使用训练好的滤波器，在输入的特征图上做卷积操作。卷积神经网络的学习过程使用梯度下降法，让待学习的参数向着数值梯度的反方向前进，从而使参数达到损失函数取极小值点。但训练数据集很大时，整个数据集无法全部装入内存，这时计算训练中某一时刻的数值梯度的时间空间代价太高，难以承受。实际应用中的做法是规定超参数“Batch Size”，认为从训练集中的一小批样本计算出的梯度可以代表当前整个网络整个训练集的梯度。同时处理成批图片也有助于提高矩阵运算的局部性，将计算资源充分利用起来。

在定义卷积层OpenMP并行化计算的线程数时，我们必须充分考虑Batch Size参数的影响。因为如果同批处理的样本数量少于所开线程，那么过多线程将引入冗余的计算量，造成计算资源的浪费。线程数多于所需实际数量时，线程之间的通信协调也将拖慢整体的计算时间。

故，我们在基类BaseConvolutionLayer中维护成员变量num\_of\_threads\_为设定的线程数，并设定其初始值为1。解析网络结构配置文件之后，我们取num\_of\_threads\_为OpenMP在平台上最多支持的线程数和Batch Size二者之间较小者。

即num\_of\_threads = min (omp\_get\_max\_threads(), Batch Size)

具体到代码上的表达为：

num\_of\_threads\_ = 1;

#ifdef \_OPENMP

num\_of\_threads\_ = omp\_get\_max\_threads() < bottom[0]->shape(0) ?

omp\_get\_max\_threads() : bottom[0]->shape(0);

if (num\_of\_threads\_ < 1) {

LOG(WARNING) << "Base Conv layer: omp\_get\_max\_threads() ="

<< num\_of\_threads\_;

num\_of\_threads\_ = 1;

继承BaseConvolutionLayer的ConvolutionLayer在前向计算中即可使用继承的成员变量num\_of\_threads\_进行OpenMP并行化：

#ifdef \_OPENMP

#pragma omp parallel for num\_threads(this->num\_of\_threads\_)

#endif

for (int n = 0; n < this->num\_; ++n) {

this->forward\_cpu\_gemm(bottom\_data + n \* this->bottom\_dim\_, weight,

top\_data + n \* this->top\_dim\_);

if (this->bias\_term\_) {

const Dtype\* bias = this->blobs\_[1]->cpu\_data();

this->forward\_cpu\_bias(top\_data + n \* this->top\_dim\_, bias);

}

}

}

}

反向传播同理：

#ifdef \_OPENMP

this->clear\_weight\_mt();

#pragma omp parallel num\_threads(this->num\_of\_threads\_)

#endif

{

#ifdef \_OPENMP

#pragma omp for

#endif

for (int n = 0; n < this->num\_; ++n) {

// gradient w.r.t. weight. Note that we will accumulate diffs.

this->weight\_cpu\_gemm(bottom\_data + n \* this->bottom\_dim\_,

top\_diff + n \* this->top\_dim\_, weight\_diff);

}

#ifdef \_OPENMP

this->sum\_weight\_mt(weight\_diff);

#endif

}

}

if (propagate\_down[i]) {

#ifdef \_OPENMP

#pragma omp parallel for num\_threads(this->num\_of\_threads\_)

#endif

for (int n = 0; n < this->num\_; ++n) {

// gradient w.r.t. bottom data, if necessary.

this->backward\_cpu\_gemm(top\_diff + n \* this->top\_dim\_, weight,

bottom\_diff + n \* this->bottom\_dim\_);

}

}

通过并行化GEMM矩阵相乘操作，我们完成了对卷积层的并行优化。

8.4、对im2col、col2im函数的优化

im2col函数将输入特征图原地展开为列向量，以将计算过程转化为稀疏矩阵与列向量相乘的形式。col2im是im2col的逆操作，将列向量还原为特征图。Caffe在实现这个函数时并未用到BLAS库，而是以五重循环的形式直接实现。由于循环过深，难以从代码层面直接进行优化。且Intel提出的代码优化【来源请求】已经被整合到了GitHub上更新后的Caffe中。

而且循环中包含较为复杂的条件判断（需要通过下标判定列向量该维补零填位还是填入特征图像素），并行化带来的代价得不偿失。我们保持这两个函数原样，不做优化。

8.5、对Pooling层的优化

Pooling层将特征图做下采样处理，达到降维的目的。Caffe目前只实现了两种较为常用的Pooling操作：Average Pooling与Max Pooling。Average Pooling取下采样格子中各特征图像素的平均值作为输出，Max Pooling取下采样格子中各特征图像素的最大值作为输出。这种Pooling操作可以降维，同时大大增加卷积神经网络的鲁棒性、平移不变性。

Caffe使用四重循环实现这两种Pooling操作，循环指标分别为图片批次、图片频道、长、宽。其中，后两重循环对每个下采样格子进行操作，各下采样格子之间完全独立。故，每个线程对应一个下采样格子，即可在保证代码正确性的前提下实现并行优化。

具体到代码上的表达为：

#ifdef \_OPENMP

#pragma omp parallel for collapse(2)

for (int n = 0; n < bottom[0]->num(); ++n) {

for (int c = 0; c < channels\_; ++c) {

//每个下采样格子的下采样操作

}

}

#endif

其中，预编译宏“collapse(2)”意为对下列代码的两重for循环都做展开，进行并行。由于外面两重循环分别代表对批次中的不同图片、图片中的不同频道迭代，这个预编译宏能够达到令每个下采样格子互相并行计算的效果。

8.6、对ReLU层的优化

ReLU是目前卷积神经网络较为常用的神经单元激活函数。如前文所述，Caffe将数据组织为Blob在层之间流动，ReLU层每次处理一批神经元，计算它们的激活程度，作为输出传导到下一层。显然，每个神经元之间的激活程度只和其激活函数的输入有关，互相完全独立。故我们对ReLU层的前向和后向计算作如下优化：

#ifdef \_OPENMP

#pragma omp parallel for

#endif

for (int i = 0; i < count; ++i) {

top\_data[i] = std::max(bottom\_data[i], Dtype(0))

+ negative\_slope \* std::min(bottom\_data[i], Dtype(0));

}

和

#ifdef \_OPENMP

#pragma omp parallel for

#endif

for (int i = 0; i < count; ++i) {

bottom\_diff[i] = top\_diff[i] \* ((bottom\_data[i] > 0)

+ negative\_slope \* (bottom\_data[i] <= 0));

}