《神经网络基础》第四次课程作业

截至日期：2017 年 11 月 5 日 23:59

1 Convolution 求导关系

项目人数　1 人
提交形式　电子提交，可以是电子文档或者拍摄的手写版

在神经网络中，有时需要进行反卷积操作(deconvolution)，以便从 feature map 重建输入。而在实践中，反卷积常常用`ConvBackwardData`来实现；其在训练过程中也需要求导，而导数就是`ConvForward`和`ConvBackwardFilter`。事实上，这三个操作互为对方的导数。

具体而言，对于函数 $f : \mathbb{R}^m \times \mathbb{R}^n \mapsto \mathbb{R}^k$，定义求导算子 ∇_1 和 ∇_2 满足如下条件：

$$\begin{aligned} &\forall L : \mathbb{R}^k \mapsto \mathbb{R} \\ &\forall x \in \mathbb{R}^m,\, y \in \mathbb{R}^n \\ &\frac{\partial L(f(x,y))}{\partial x} = \nabla_1^f(y, \frac{\partial L(z)}{\partial z}) \\ &\frac{\partial L(f(x,y))}{\partial y} = \nabla_2^f(x, \frac{\partial L(z)}{\partial z}) \\ &\text{where } z \triangleq f(x,y) \end{aligned} \tag{1}$$

用 $f_p(x, w)$ 表示超参数为 p 的`ConvForward` 操作，其中 x 是形状为 $[N, IC, IH, IW]$ 的张量，表示被卷积的输入图片；w 是形状为 $[OC, IC, FH, FW]$ 的张量，表示卷积核(filter)。$f_p(x, w)$ 返回形状为 $[N, OC, OH, OW]$ 的张量。超参数 p 包含 $p.sh$, $p.sw$, $p.ph$, $p.pw$，分别表示卷积操作在高度和宽度方向上的步长(stride) 和填充(padding)。`ConvForward` 步骤如下：

1. 在 x 的 IH 和 IW 两维上，分别按照 $p.ph$ 和 $p.pw$ 在两端对称填 0，得到 $\hat{x}$，其形状为 $[N, IC, IH + 2 \times p.ph, IW + 2 \times p.pw]$。用 numpy 表示为：

   $\hat{x}$ `= np.zeros((N, IC, IH+2*p.ph, IW+2*p.pw))`
   $\hat{x}$`[:, :, p.ph:-p.ph, p.pw:-p.pw] =` $\hat{x}$

2. 对于 $0 \le n < N$ 和 $0 \le oc < OC$，计算

   $$\hat{y}[n, oc] = \sum_{0 \le i < IC} \hat{x}[n, i] * w[oc, i]$$

   其中 $*$ 表示普通的二维卷积操作。

3. 在 $\hat{y}$ 中按照步长取出最终结果 y：

   y `=` $\hat{y}$`[:, :, ::p.sh, ::p.sw]`

用 $g_p(w,d)$ 和 $h_p(x,d)$ 分别表示超参数为 p 的`ConvBackwardData` 和`ConvBackwardFilter` 操作，于是两者可定义如下：

$$
\begin{aligned}
g_p(w,d) &\triangleq \nabla_1^{f_p}(w,d) \\
h_p(x,d) &\triangleq \nabla_2^{f_p}(x,d)
\end{aligned} \tag{2}
$$

现在请你解决以下两个问题：

1. 推导 OH 与 $IH, FH, p.ph, p.sh$ 的关系，以及 OW 与 $IW, FW, p.pw, p.sw$ 的关系
2. 令 $y = g_p(w,d)$, $z = h_p(x,d)$，y' 和 z' 分别表示当前损失函数相对 y 和 z 的梯度。证明：

$$
\begin{aligned}
\nabla_1^{g_p}(d,y') &= h_p(y',d) \\
\nabla_2^{g_p}(w,y') &= f_p(y',w) \\
\nabla_1^{h_p}(d,z') &= g_p(z',d) \\
\nabla_2^{h_p}(x,z') &= f_p(x,z')
\end{aligned}
$$

2 矩阵乘优化

项目人数　1 人
提交形式　电子提交，代码 + 实验报告

课堂上已经讲到了矩阵乘的简单优化。现在，请你在自己的计算机上，测量这种优化策略在不同矩阵大小下的加速比。具体而言，需要在实验报告里写明：

- 实验环境：硬件型号、操作系统、编译器版本、编译选项等信息
- 优化前和优化后的代码在不同矩阵大小下的 FLOPS
- 计算自己的计算机的理论 FLOPS 峰值
- 测量一个成熟的数学库中的矩阵运算操作，计算其 FLOPS (包括但不限于 MKL、OpenBLAS、matlab 等，可以通过 C-API 或者 numpy 等方式调用，不限制库和调用方式)
- (可选) 进一步优化自己的矩阵乘实现

2.1 速度测量

要准确的测量一段代码的速度并不是一件容易的事，编译器优化、操作系统和系统库的策略、运行环境的随机绕动等都会影响测量结果。可以参考以下测量方法：

1. 将被测量代码和计时代码分别写在两个`.cpp` 里，防止编译器根据计时逻辑进行了不恰当的优化
2. 输入矩阵要有合理的值(例如从标准正态分布采样)；如果含有大量 0、`NaN`、denormalized number 等，会严重影响计算速度
3. 第一次运行时由于 page fault 等会影响运行时间，可以丢弃；较短的时间测量也不准确，应多次运行取均值
4. 可以参考以下计时模板：

```
float *A = new float[N * N];
float *B = new float[N * N];
float *C = new float[N * N];
fill(A); fill(B);

matmul(A, B, C, N); // warmup

clock_t start = clock();
for (int i = 0; i < run_times; ++ i) {
    matmul(A, B, C, N);
}
printf("avg run time: %.3fs\n",
    static_cast<double>(clock() - start) /
    CLOCKS_PER_SEC / run_times);
```