用GPU并行计算加速卷积玻尔兹曼机的训练

李冯福 应用数学所

2015-12-23

目录

- GPU在深度学习中的应用背景
- 预备知识
- 两种并行思路
- 实例：批处理数据的valid卷积
- 实验：加速比
- 部分结果展示
- Reference

GPU在深度学习中的应用背景

- 10年： GPU 加速DNN --- 最多 20x 倍加速
- 12年： Deep Convolutional Neural Networks
  - 8 层，6000万参数，120万高像素图像，65万神经元
  - 2 × GTX 580 GPU
  - 6~8 天训练时间
- 14年： Caffe 框架 (模块化) -> 重写GPU接口
- 15年： TensorFlow、MXNET (多GPU、分布式)
- 产品： 百度识图、科大讯飞语音输入法、DeepSpeech、谷歌/百度大脑等

预备知识 – 机器学习

图1： Supervised Learning System

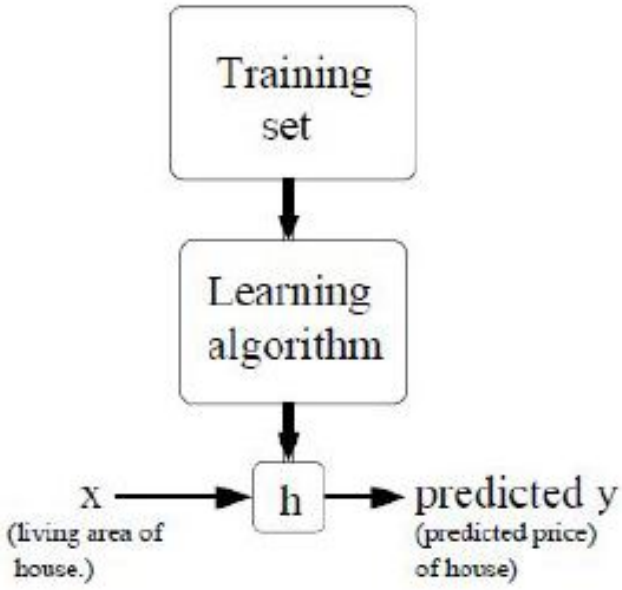


图 1: 有监督机器学习系统

三个基本要素：

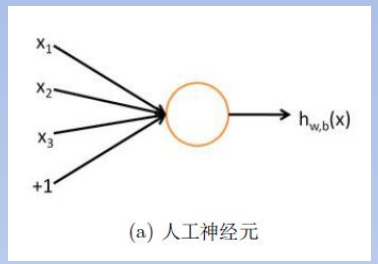
- 经验: (X, Y)
- 性能评价准则：$J(X, Y; \theta)$
- 学习：通过经验和对性能的评价来改进 θ （动态）

目标函数/损失函数：

$$\hat{\theta} = \arg\min_{\theta} J(X, Y; \theta) = \frac{1}{2} \sum_{i=1}^{m} f(h_\theta(x^{(i)}), y^{(i)})$$

预备知识 – 神经网络

神经网络：一种特殊的带有层次性的机器学习系统。



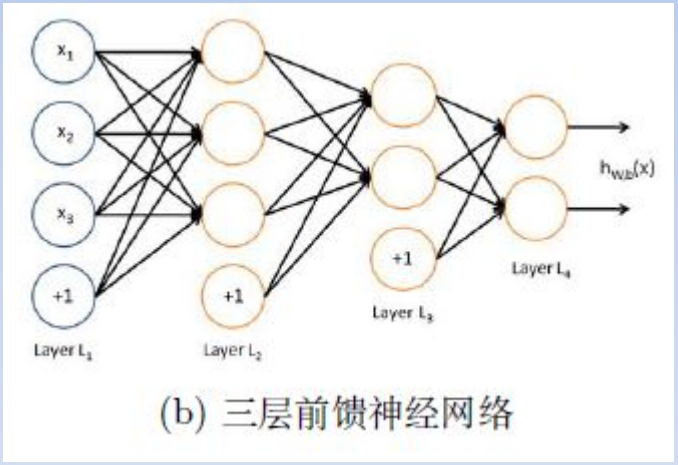
(a) 人工神经元

(a). **人工神经元**示意图。
它的推断公式如下：

$$h_{W,b}(x) = g(W^T x) = g(\sum_{i=1}^{3} W_i x_i + b)$$

这里g(x)一般是非线性的递增函数，常取为 sigmoid 函数，即：

$$g(x) = \frac{1}{1+e^{-x}}$$



(b) 三层前馈神经网络

(b). **三层前馈神经网络**。
每一个神经元的值仍然由上面的公式得到。

预备知识 – 批梯度下降

学习算法：**梯度下降算法**。但是数据量m比较大的时候一般用**批梯度下降**来近似。

算法 1 批梯度下降算法求解问题 (1)

输入： m_b 和 ϵ

输出： $\hat{\theta}$

1: 初始化: $t = 0$, $\theta^{(t)} := \theta_0$

2: **while** $J(X, Y; \theta)$ 没有收敛到最小值 **do**

3: 从 (X, Y) 中随机选取 m_b 个样本点组成集合 (X_b, Y_b) ;

4: 用链式法则求 $J(X_b, Y_b; \theta)$ 关于 θ 的导数 :

$$\frac{\partial J(X_b, Y_b; \theta)}{\partial \theta} = \sum_{i=1}^{m_b} \frac{\partial f(h, y)}{\partial h} \frac{\partial h_\theta(x^{(i)})}{\partial \theta} \tag{5}$$

5: 更新 θ:

$$\theta^{(t+1)} := \theta^{(t)} - \epsilon \cdot \frac{\partial J(X_b, Y_b; \theta)}{\partial \theta} \tag{6}$$

6: 更新迭代步数: $t := t + 1$

7: **end while**

8: 输出 $\hat{\theta} = \theta^{(t)}$

预备知识 – 卷积玻尔兹曼机

卷积玻尔兹曼机：

- 英文名：Convolutional Restricted Boltzmann Machine
- 一种非监督的神经网络模型，很方便处理二维图像

推断公式：

$$h_{i,j}^k = (W^k * v)_{ij} + b_k$$

V => H ： valid 卷积
H => P ： 下采样
H => V ： full 卷积

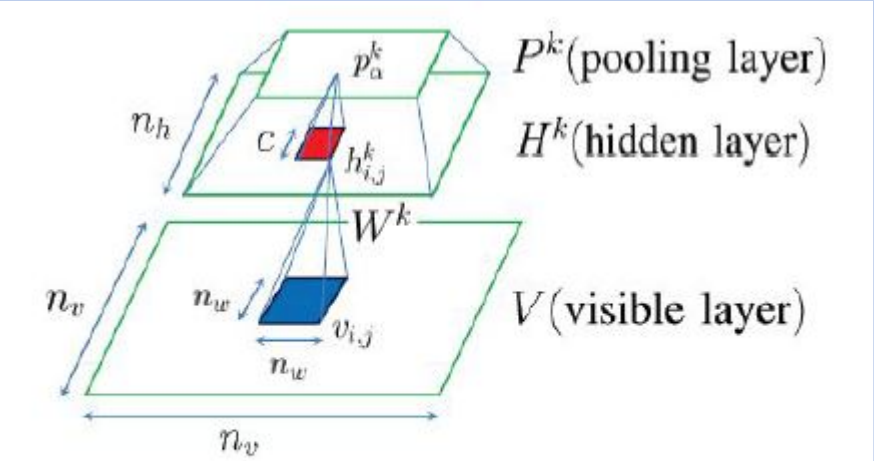


图 3: 卷积玻尔兹曼机示意图

两种并行思路

- **模型并行**：考虑到各个 $h_{i,j}^k$ 之间求解相互不干扰，可以并行的求解 $h_{i,j}^k (0 \leq i,j \leq n_v)$
- **数据并行**：各个数据得到的 $h_{i,j}^k$ 也不同，可以对不同的数据并行的求解
- CUDA 实现：两级并行框架

➢模型并行 <-> thread block

➢数据并行 <-> block grid

实例：批处理数据的valid卷积

```
============================================================
#define WIDTH 8  // 预先固定共享内存的宽度
__global__ void conv4d_valid_kernel
(float *d_H, float *d_V, float *d_W, int nv, int nw){
    // 获取线程 id 和 block id
    const int tx = threadIdx.x;  // 对应于 i
    const int ty = threadIdx.y;  // 对应于 j
    const int bx = blockIdx.x;   // 对应于 v: Kin id
    const int by = blockIdx.y;   // 对应于 k: Kout id
    const int bz = blockIdx.z;   // 对应于 l: mb id

   // 声明共享内存Ws并将 W 对应的部分读入 Ws 供当前 block 使用
    __shared__ float Ws[WIDTH][WIDTH];

    if(ty < nw && tx < nw){
        Ws[ty][tx] = d_W[tx, ty, bx, by];
    }
    __syncthreads();  // 同步；因为后面都得用到这个共享的 Ws

    const int nh = nv - nw + 1;
    // tiles_x：当前 block 沿 x 方向平移来覆盖 nh*nh 的区域
    const int tiles_x = nh / blockDim.x + 1;
    // tiles_y：当前 block 沿 y 方向平移来覆盖 nh*nh 的区域
    const int tiles_y = nh / blockDim.y + 1;
```

```
    for iy = 0 to tiles_y-1
        for ix = 0 to tiles_x-1
            // 当前计算的 H 中元素对应的位置
            int idH_y = ty + iy * blockDim.y;
            int idH_x = tx + ix * blockDim.x;

            // 线程越界则抛弃
            if(idH_x >= nh || idH_y >= nh)
                continue;

            // 计算 V 与 Ws 的卷积
            float temp = conv(d_V, Ws);

            // 将当前卷积值加入到H中
            // 对 Kout 个 block 都得执行这一步，因此用原子操作
            atomicAdd(&d_H[idH_x, idH_y, by, bz], temp);
        endfor
    endfor
}
============================================================
```

批处理数据的 valid 卷积伪代码

kernel 函数名	block 尺寸	grid 尺寸
valid 卷积	(16, 16)	(K_{in}, K_{out}, m_b)
full 卷积	(16, 16)	(K_{out}, K_{in}, m_b)
pooling	(16, 16)	(K_{out}, m_b)
element-wise opration	256	(K_{out}, m_b)

表 1: 主要 kernel 函数的配置参数

实验：加速比

数据：10,000 个 31 × 31 的灰度图像

迭代次数：50,000 次 （尽可能消除随机性）

K_{out} 尺寸：从2到256，以2倍速增长

m_b 大小：从1到64，以2倍速增长

卷积核大小： 8 × 8

CPU 配置： Intel® Core™ i7-4700 CPU @ 3.4GHz

GPU 配置： Tesla K20c, 2496 Cores, @ 706 MHz

加速比：

$$\text{speed up} := \frac{t_{cpu}}{t_{gpu}}$$

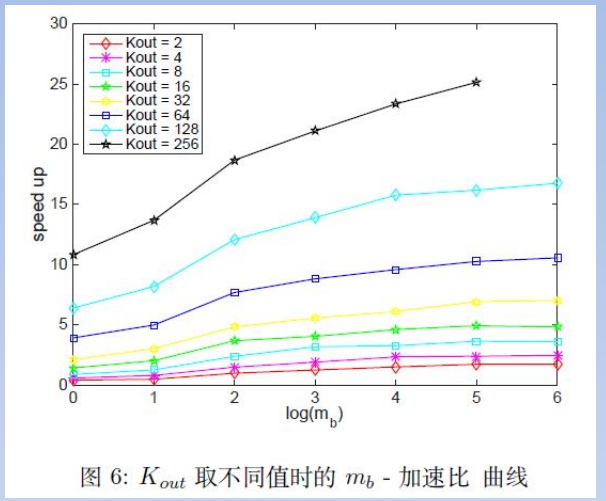


图 6: K_{out} 取不同值时的 m_b - 加速比 曲线

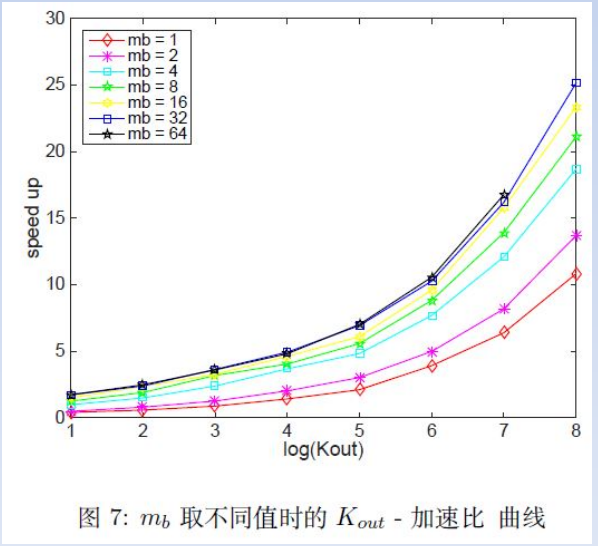


图 7: m_b 取不同值时的 K_{out} - 加速比 曲线

部分结果展示

4.5 部分结果展示

最后学习到的参数 W 经过可视化后如图 8 所示。它是一种底层的 Garbor 特征提取器，可以用作高层特征的提取。

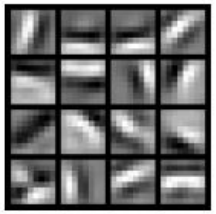
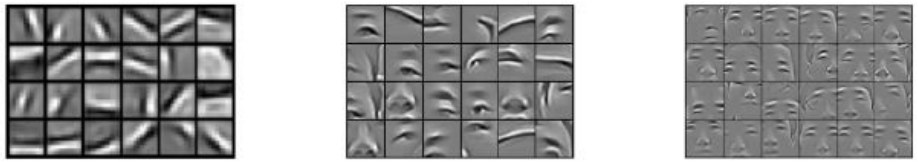


图 8: 参数 W 的可视化 ($K_{out} = 16, m_b = 1$)

作为拓展，图 9 也展示了我们工作中学习到的一些高层的人脸特征，请参考 [6]。



(a) (b) (c)

Fig. 7: Visualization of the first (a), second (b), and third (c) layer bases. The first layer bases are visualized as "images". The second and the third layers bases are visualized as the patterns (which are obtained by hierarchically inference of the CDBN) most liked by the second and third layer's hidden units respectively.

图 9: 高层特征的可视化

Reference

- 深度学习应用：
  http://www.csdn.net/article/2015-09-25/2825806
- Caffe: http://caffe.berkeleyvision.org/
- MXNET: http://mxnet.readthedocs.org/en/latest/
- TensorFlow: https://www.tensorflow.org/
- H. Qiao, X. Xi, Y. Li, W. Wu, and F. Li, *Biologically Inspired Visual Model With Preliminary Cognition and Active Attention Adjustment*, IEEE Transactions on Cybernetics, 2014, vol. 45, pp. 2612 - 2624.
- GPU并行加速CRBM代码：
  https://github.com/fengfu-chris/parallel-crbm

Thank you!
That's all.