

并行计算实验四

PB19030888 张舒恒

问题描述

使用参数服务器架构训练一个机器学习模型，对MNIST手写数字图片进行分类。自行选择参数服务器的同步模式。自行选择机器学习模型以及训练算法。使用MPI实现。

算法设计

采用三层神经网络的机器学习模型。^[1]

显层全连接层(Fully Connected Layer)采用正态分布生成初始参数，正向传播函数forward计算 $XW['val'] + b$ ，反向传播函数backward计算梯度 $W['grad'] = X.Tdout$, $b['grad'] = \text{sum}(dout)$ ，并更新模型参数。

```
class FC:

    def __init__(self, D_in, D_out):
        self.cache = None
        self.W = {'val': np.random.normal(0.0, np.sqrt(2 / D_in), (D_in,
D_out)), 'grad': 0}
        self.b = {'val': np.random.randn(D_out), 'grad': 0}

    def _forward(self, X):
        out = np.dot(X, self.W['val']) + self.b['val']
        self.cache = X
        return out

    def _backward(self, dout):
        X = self.cache
        dx = np.dot(dout, self.W['val'].T).reshape(X.shape)
        self.W['grad'] = np.dot(X.T, dout)
        self.b['grad'] = np.sum(dout, axis=0)
        self._update_params()
        return dx

    def _update_params(self, lr=0.001):
        self.W['val'] -= lr * self.W['grad']
        self.b['val'] -= lr * self.b['grad']
```

隐层激活函数采用ReLU，对输入值进行取绝对值。

```
class ReLU:

    def __init__(self):
        self.cache = None

    def _forward(self, X):
        out = np.maximum(0, X)
        self.cache = X
        return out
```

```
def _backward(self, dout):
    x = self.cache
    dx = np.array(dout, copy=True)
    dx[x <= 0] = 0
    return dx
```

活化层只需用到正向传播函数forward，其输入是三层神经网络正向输出，求出每个样本不同数字预测值的最大值 $maxes$ ，再讲所有数字的预测值映射到0-1，即 $Y = e^{X-maxes}$ ，最后求出Y中每个数字预测值占总预测值的比重即为预测概率。

```
class Softmax:

    def __init__(self):
        self.cache = None

    def _forward(self, X):
        maxes = np.amax(X, axis=1)
        maxes = maxes.reshape(maxes.shape[0], 1)
        Y = np.exp(X - maxes)
        Z = Y / np.sum(Y, axis=1).reshape(Y.shape[0], 1)
        self.cache = (X, Y, Z)
        return Z

    def _backward(self, dout):
        X, Y, Z = self.cache
        dZ = np.zeros(X.shape)
        dY = np.zeros(X.shape)
        dX = np.zeros(X.shape)
        N = X.shape[0]
        for n in range(N):
            i = np.argmax(Z[n])
            dZ[n, :] = np.diag(Z[n]) - np.outer(Z[n], Z[n])
            M = np.zeros((N, N))
            M[:, i] = 1
            dY[n, :] = np.eye(N) - M
        dx = np.dot(dout, dZ)
        dx = np.dot(dx, dY)
        return dx
```

交叉熵损失函数获取样本的预测数字概率分布和真实数字，将每个样本的预测数字概率分布和**热独码**编码的真实数字分布点乘，若结果x为0说明误差很大，给损失值加上一个很大的评估值，若结果x不为0则损失值为 $-\ln x$ 。将预测数字概率分布中真实数字的概率-1来得到负反馈矩阵。返回**误差估计损失值**和**负反馈矩阵**。

```
class CrossEntropyLoss:

    def __init__(self):
        pass

    def get(self, Y_pred, Y_true):
        N = Y_pred.shape[0]
        softmax = Softmax()
        prob = softmax._forward(Y_pred)
        loss = NLLLoss(prob, Y_true)
        Y_serial = np.argmax(Y_true, axis=1)
```

```

        dout = prob.copy()
        dout[np.arange(N), Y_serial] -= 1
        return loss, dout

def NLLLoss(Y_pred, Y_true):
    loss = 0.0
    N = Y_pred.shape[0]
    M = np.sum(Y_pred * Y_true, axis=1)
    for e in M:
        if e == 0:
            loss += 500
        else:
            loss += -np.log(e)
    return loss / N

```

参数服务器模型如下：

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

- 1: issue LoadData() to all workers
- 2: **for** iteration $t = 0, \dots, T$ **do**
- 3: issue WORKERITERATE(t) to all workers.
- 4: **end for**

总体的并行训练流程

1. 分发数据到每一个worker节点
2. 每一个worker节点并行执行 WORKERITERATE方法，一共迭代T轮，完成模型的并行训练

Worker $r = 1, \dots, m$:

- 1: **function** LOADDATA()
- 2: load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
- 3: pull the working set $w_r^{(0)}$ from servers
- 4: **end function**
- 5: **function** WORKERITERATE(t)
- 6: gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
- 7: push $g_r^{(t)}$ to servers
- 8: pull $w_r^{(t+1)}$ from servers
- 9: **end function**

worker的初始化过程

- 1、每个worker载入一部分训练数据
- 2、每个woker将全部初始模型参数w0从 servers节点上拉下来

worker节点的迭代计算过程

- 1、仅利用本节点的训练数据计算梯度
- 2、将计算好的梯度gr(t) push到servers节点
- 3、将新一轮的模型参数w(t+1)拉到本地 worker节点

Servers:

- 1: **function** SERVERITERATE(t)
- 2: aggregate $g^{(t)} \leftarrow \sum_{r=1}^m g_r^{(t)}$
- 3: $w^{(t+1)} \leftarrow w^{(t)} - \eta (g^{(t)} + \partial \Omega(w^{(t)}))$
- 4: **end function**

server节点的迭代计算过程

- 1、在收到m个worker节点计算的梯度后，汇总梯度形成总梯度
- 2、利用汇总梯度g(t)，融合正则化项梯度，计算出新梯度w(t+1)

实验配置

软硬件配置


CPU参数:

Processor (CPU)	
CPU Name	11th Gen Intel® Core™ i7-11600H @ 2.90GHz
Threading	1 CPU - 6 Core - 12 Threads
Frequency	4192.09 MHz (42 * 99.81 MHz) - Uncore: 3493.4 MHz
Multiplier	Current: 42 / Min: 8 / Max: 46
Architecture	Tiger Lake / Stepping: R0 / Technology: 10 nm
CPUID / Ext.	6.D.1 / 6.8D
IA Extensions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, AVX512F, FMA3, SHA
Caches	L1D : 48 KB / L2 : 1280 KB / L3 : 18432 KB
Caches Assoc.	L1D : 12-way / L2 : 20-way / L3 : 12-way
Microcode	Rev. 0x34
TDP / Vcore	45 Watts / 1.117 Volts
Temperature	94 °C / 201 °F
Type	Retail
Cores Frequencies	#00: 4192.09 MHz #01: 4192.09 MHz #02: 4192.09 MHz #03: 4192.09 MHz #04: 4192.09 MHz #05: 4192.09 MHz

GPU参数:

Graphic Card (GPU)	
GPU #1 Type	NVIDIA GeForce RTX 3050 Laptop GPU (GA107) @ 1500 MHz
GPU #1 Brand	ASUSTeK Computer Inc.
GPU #1 VRAM	4096 MB @ 6001 MHz
GPU #2 Type	Intel(R) UHD Graphics
GPU #2 Brand	ASUSTeK Computer Inc.

编译器参数:

Python 解释器:	 Python 3.9 (pythonProject1) C:\Users\凝雨\AppData\Local\Programs\Python\Python39\python.exe
-------------	---

数据集配置

采用<http://yann.lecun.com/exdb/mnist/>上的60000份训练样本和10000份测试样本

实验结果

正确性验证

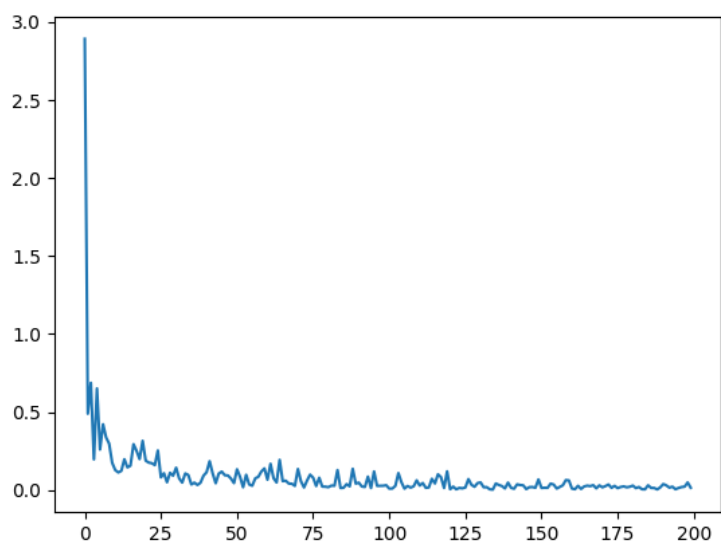
测试训练集的60000份样本，模型正确率99.92%。测试测试集的10000份样本，模型正确率98.01%。详细数据见test.log。

```

199 98.0% iter: 19600, loss: 0.00330213423011231
200 98.5% iter: 19700, loss: 0.008323898589869412
201 99.0% iter: 19800, loss: 0.006180728099093682
202 99.5% iter: 19900, loss: 0.006489142132216091
203 TRAIN--> Correct: 59952 out of 60000, acc=0.9992
204 TEST--> Correct: 9801 out of 10000, acc=0.9801
205
206 进程已结束,退出代码0
207

```

20000次迭代过程交叉熵损失函数变化曲线:



加速比分析

为了保证测试集的正确率在95%以上，模型至少需要经过约10000次迭代，以下分别测试10000次，15000次，20000次迭代时2线程，4线程，8线程，16线程的运行时间以及加速比。

10000次迭代:

线程数	训练集正确率	测试集正确率	运行时间	相对于2线程的加速比
2	97.31%	95.34%	12.467s	1
4	97.07%	96.01%	9.561s	1.304
8	96.81%	95.61%	7.340s	1.699
16	97.29%	96.09%	5.791s	2.153

15000次迭代:

线程数	训练集正确率	测试集正确率	运行时间	相对于2线程的加速比
2	98.12%	96.98%	16.207s	1
4	98.70%	96.54%	11.462s	1.414
8	98.42%	97.21%	8.026s	2.019
16	98.01%	97.19%	6.451s	2.512

20000次迭代:

线程数	训练集正确率	测试集正确率	运行时间	相对于2线程的加速比
2	99.01%	97.39%	21.207s	1
4	99.07%	97.18%	12.781s	1.648
8	99.92%	98.01%	9.014s	2.337
16	99.30%	97.41%	7.012s	3.005

当迭代次数较少时加速比较小，随着迭代次数增加，训练集正确率和测试集正确率均有所提高，且加速比也有所改善。这主要是因为迭代次数越多，MPI准备开销以及模型准备开销占比降低。

参考文献

[1] [MNIST For Machine Learning Beginners With Softmax Regression | DataScience+ \(datascienceplus.com\)](https://datascienceplus.com/mnist-for-machine-learning-beginners-with-softmax-regression/)