Deep Learning

Sathish Vadhiyar

Introduction
Compressed Neural Networks (CNNs)

- CNN composed of a sequence of tensors (generalized matrices with dynamical properties)
- The tensors are referred to as weights
- Input fed to CNN
- A series of tensor-matrix operations
  - Could be matrix-matrix multiplication, matrix-vector multiplication, FFT, non-linear transform
- Output obtained
- To get correct classification, need to get a set of working weights

CNN Training

- Weights need to be trained
- Training process consists of three steps:

1. Forward propagation: Input passed from first to last layer. Output is predicted
2. Backward propagation: Numerical prediction error passed from last to first layer and gradient of W, delta, obtained
3. Weight update: W = W-n.delta [n is the learning rate]

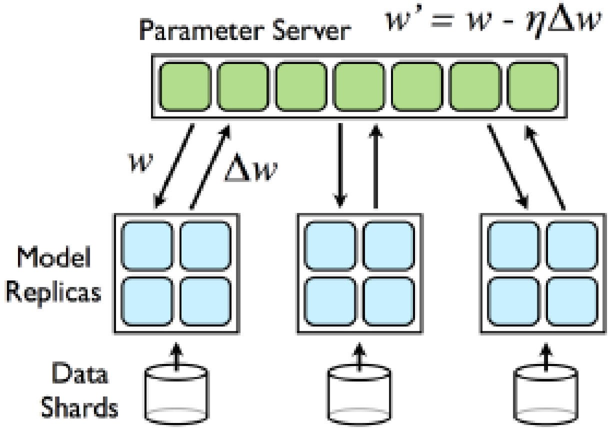
- Above three steps iterated until model is optimized
- Using stochastic gradient descent
- Randomly pick a batch of samples

Parallelism

- Data parallelism
  - Data set partitioned into P parts
  - Each machine has a copy of a neural network and the Ws
  - The master updates W by the sum of all the subgradients, delta, from all the processors
  - The master broadcasts W to all machines
- Model parallelism
  - Partitions the neural network across P processors
  - i.e., parallelizes the matrix operations across the processors
- Most methods follow data parallelism since the matrix sizes are small for model parallelism

Parallel Algorithms

- Parameter server or asynchronous SGD
  - All workers complete their iteration step (local updates, sending to master, and receiving W from master) asynchronously
  - Master process uses lock to avoid weight update conflicts
  - One worker at a time



- Hogwild (lock-free)
  - Removes the above lock
  - Multiple workers at a time
- EAGSD (round-robin)

$$W_{t+1}^i = W_t^i - \eta(\Delta W_t^i + \rho(W_t^i - \bar{W}_t)) \qquad (1)$$

Local updates by workers

$$\bar{W}_{t+1} = \bar{W}_t + \eta \sum_{i=1}^{P} \rho(W_t^i - \bar{W}_t) \qquad (2)$$

Global updates by master

Multi-GPU Implementation

Algorithm 1: Original EASGD on Multi-GPU system
master: CPU, workers: GPU_1, GPU_2, ..., GPU_P

Input: samples and labels: $\{X_i, y_i\}$ $i \in 1, ..., n$
#iterations: T, batch size: b, #GPUs: G

Output: model weight W

Normalize X on CPU by standard deviation: $E(X) = 0$ (mean) and $\sigma(X) = 1$ (variance)
Initialize W on CPU: random and Xavier weight filling
for $j = 1; j <= G; j{+}{+}$ **do**
    create **local** weight W_j on j-th GPU, copy W to W_j
create **global** weight $\bar{W}_1$ on 0-th GPU, copy W to $\bar{W}_1$
for $t = 1; t <= T; t{+}{+}$ **do**
    j = t mod G
    CPU **randomly** picks b samples
    CPU **asynchronously** copies b samples to j-th GPU
    CPU sends $\bar{W}_t$ to j-th GPU
    Forward and Backward Propagation on j-th GPU
    CPU gets W_t^j from j-th GPU
    j-th GPU updates W_t^j by Equation (1)
    CPU updates $\bar{W}_t$ by $\bar{W}_{t+1} = \bar{W}_t + \eta\rho(W_t^j - \bar{W}_t)$

Thank You