Parallel Backpropagation for Multilayer Neural Networks

Palash Goyal[1] Nitin Kamra[1] Sungyong Seo[1] Vasileios Zois[1]

[1]Department of Computer Science
University of Southern California

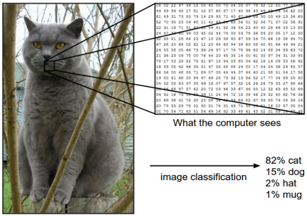
May 8, 2016

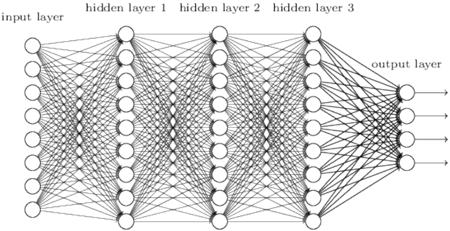
Outline

1. Problem statement
2. Why mini batch?
3. Parallelization Methods
   - POSIX Threads
   - CUDA
4. Experiments
5. Results and analysis

Problem statement

- Prevailing neural network architectures are implemented using several hidden layers, with each one consisting of thousands to millions of neurons in order to generalize well on diverse inputs.
- Can you imagine how many parameters need to be trained for every iterations?





Problem statement(contd.)

- Gradient Descent is the most commonly used optimization algorithm used to train neural networks in supervised settings.
- Having the gradient as a sum of partial gradients with respect to individual training examples opens up the possibility of parallelizing the gradient computation efficiently.

Gradient Descent

- Suppose we have some training set (x_i, y_i) for $i = 1, \cdots, N$:

$$\mathcal{L}_{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i; \theta))^2$$

- Then, the parameters are updated as follow:

$$\theta_j = \theta_j - \alpha \frac{\partial \mathcal{L}_{MSE}(\theta)}{\partial \theta_j}$$

Batch vs. Stochastic

- If the gradient is computed by using whole dataset (N), it is Batch Gradient Descent.
  - This is great for convex, or relatively smooth error manifolds.
  - Gradient is less noisy (averaged over a large number of samples).
  - For large or infinite datasets, batch gradient is impractical.
- Stochastic Gradient Descent(SGD) computes the gradient using a single example.
  - SGD works well for error manifolds that have lots of local maxima/minima because the somewhat noisy gradient helps to escape local minima into a region that hopefully is more optimal.
  - SGD may go "zig-zag" to a local minimum because of highly noisy gradient.

Alternative: Mini Batch

- Mini Batch Gradient Descent (MGD) is to compute the gradient against more than one training example at each step.
- M is the mini batch size.

$$\mathcal{L}_{MSE}^{k}(\theta) = \frac{1}{M} \sum_{i=s_k}^{s_k+M} (y_i - f(x_i;\theta))^2$$

$$\theta_k = \theta_k - \alpha \frac{\partial \mathcal{L}_{MSE}^{k}(\theta)}{\partial \theta_k}$$

- Parallelization : Distributing training examples across various processors and letting them compute a partial gradient over their own training examples.

Outline

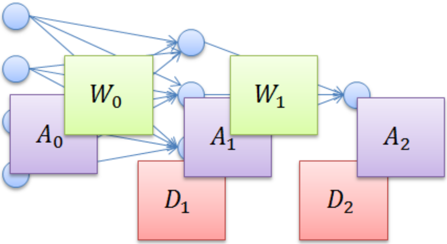
PThreads - POSIX Threads

- **Low level multithreading API** - Threads are created and each task is assigned to each thread in parallel.
- **TODO** - ABOUT Basic functions to show how Pthread is used in our project

Outline

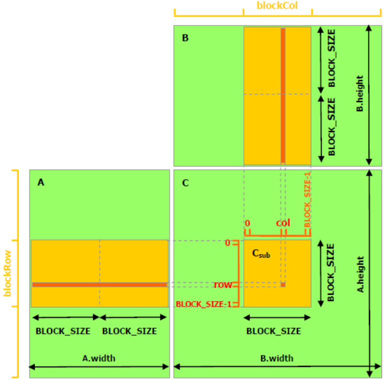
Design Overview

- Neural Network Training Steps
  - **Forward propagation:**
    $A_{i+1} = f(W_i \cdot A_i)$
  - **Back propagation:**
    $D_L = Y - A_L$,
    $D_i = (W_i)^T \cdot D_{i+1} \circ d(W_{i-1} \cdot A_{i-1})$
  - **Weight update:**
    $W_i = W_i + \frac{n}{b} \cdot \sum_{j=1}^{b} D_{i+1}^{j} \cdot A_i^{j}$



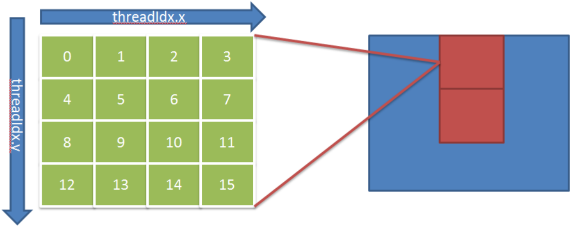
GPU Implementation Details

- Training Kernels
  - Variants of tiled matrix multiplication
  - Each thread is responsible for computing a single matrix cell
  - On-chip memory for amplifying



GPU Implementation Details (2)

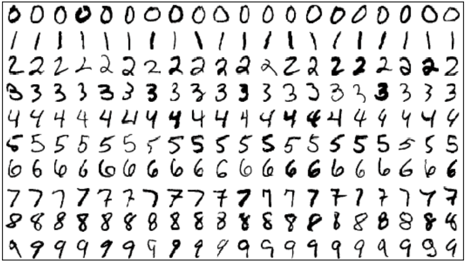
- Optimization
  - Coalesced access to global memory by iterating over the secondary dimension
  - Same strategy eliminates shared memory bank conflicts



- Neural Network Customization
  - Single or double precision arithmetic
  - User preferred activation function
  - Abstract definition of network architecture and training parameters (i.e. batch size, learning rate)

MNIST dataset

- MNIST, handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples.
- The digits have been size-normalized and centered in a fixed-size image to 28x28(784) image.
- 10,000 examples in 60,000 training set are used as validation set and left 50,000 images are used for training.
- The original labels values are 0 to 9 but it is vectorized by one-hot encoding.



Experiment parameters

- **Network structures**

	# Layers (In,**Hidden**,Out)	# Nodes (In,**Hidden**,Out)	Bias
Network1	1,**1**,1	784,**1024**,10	Yes
Network2	1,**2**,1	784,**1024,1024**,10	Yes

- **Hyperparameters**

Learning Rate	# Epochs	Size of Batch	Regularization
0.1	Let Me Know	2^n, $n \in [7, 12]$	None

- **Activation function**
  - Sigmoid function

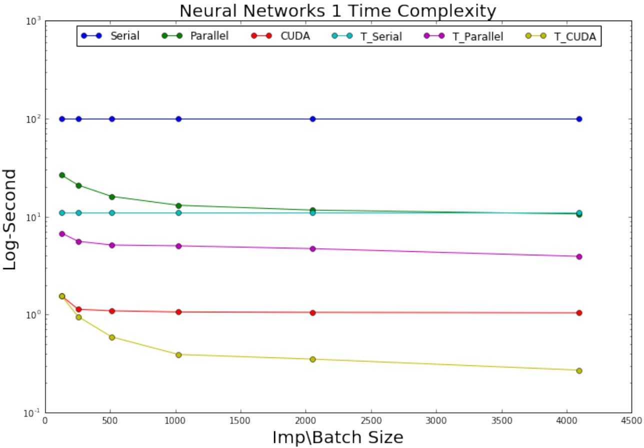
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Parallel experiments

- **PThreads**
  - About PThreads setting
- **CUDA**
  - About CUDA setting
- **theano**
  - About theano

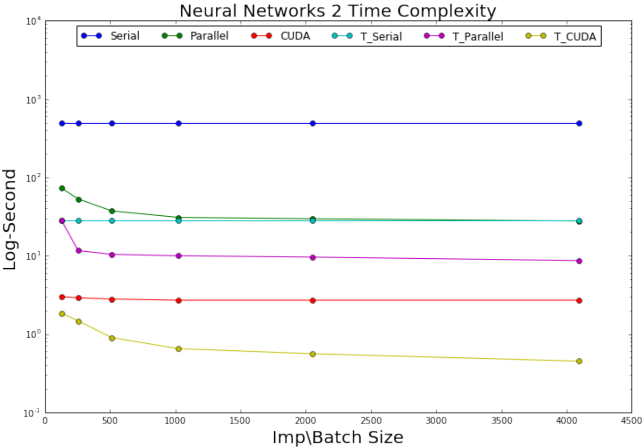
Results

- **Neural Networks 1**
- Accuracy is 97.78%

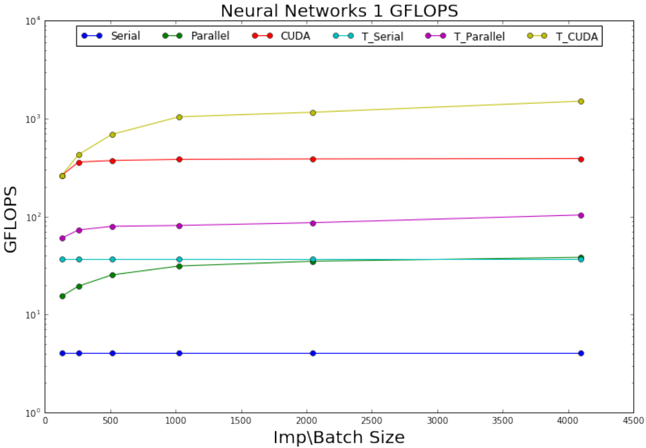


Results (contd.)

- **Neural Networks 2**
- Accuracy is ???%

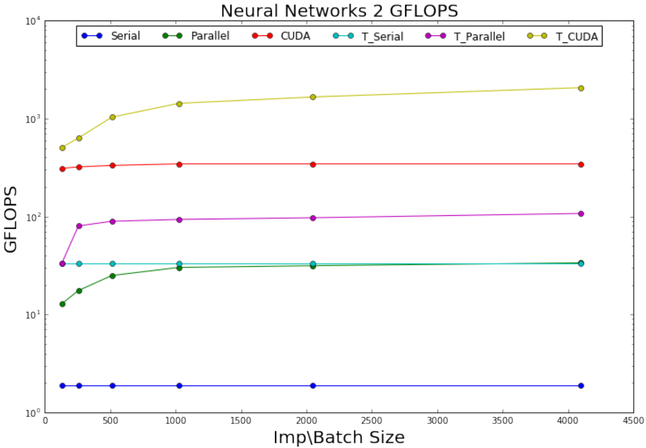


- **Neural Networks 1 - GFLOPS**



Results (contd.)

- **Neural Networks 2 - GFLOPS**



Analysis

- Discussion