Parallel Gradient Descent for Multilayer Feedforward Neural Networks

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

[1]Department of Computer Science
University of Southern California

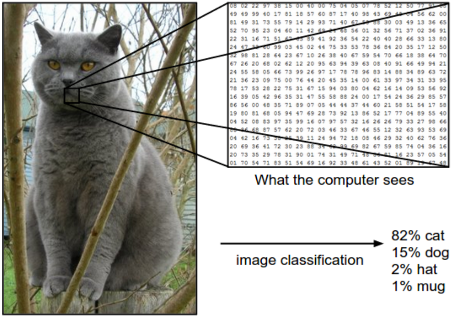
May 9, 2016

Outline

Outline

Introduction

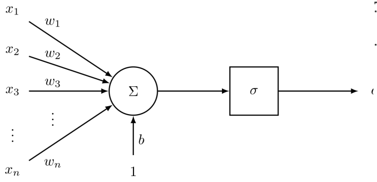
- How to learn to classify objects from images?
- What algorithms to use?
- How to scale up these algorithms?

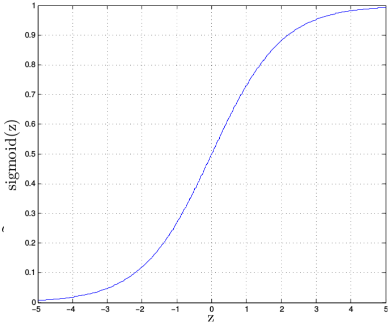


Classification

- Dataset $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1:N}$ with $x^{(i)} \in \mathbb{R}^D$ and labels $y^{(i)} \in \mathbb{R}^P$
- Make accurate prediction $\hat{y}$ on unseen data point x
- Classifier (parameters θ) approximates label as: $y \approx \hat{y} = F(x; \theta)$
- Classifier learns parameters (θ) from data $\mathcal{D}$ to minimize a pre-specified loss function

Neuron

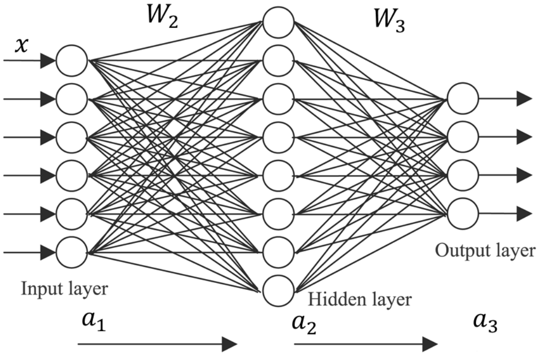




$$a = f(w^T x + b)$$

- $w \in \mathbb{R}^n$ = Weight vector
- $b \in \mathbb{R}$ = Scalar bias

Classifier: Neural Network



For each layer,

$$z_l = (W_l)^T x_l + b_l; \qquad a_l = f(z_l)$$

- $W^l \in \mathbb{R}^{n_{l-1} \times n_l}$ = Weight vector
- $b_l \in \mathbb{R}^{n_l}$ = Scalar bias

Outline

Gradient Descent

Minimize the Mean-Squared Error loss:

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

Algorithm: Gradient Descent

1. Initialize all weights (θ) randomly with small values close to 0.
2. Repeat until convergence {

$$\theta_k := \theta_k - \alpha\frac{\partial\mathcal{L}_{MSE}}{\partial\theta_k} \qquad \forall k \in \{1, 2, ..., K\}$$

}

Minibatch gradient descent considers a subset of examples

Outline

Forward Propagation

Algorithm 3 Forward Propagation

Input: Example x, parameters $[W_{\{2:L\}}, b_{\{2:L\}}]$
Output: $z_l(x), a_l(x) \qquad \forall l = 1 : L$

$z_1(x) := x, a_1(x) := x$
for $l = 2 : L$ **do**
  $z_l(x) = (W_l)^T a_{l-1}(x) + b_l$
  $a_l(x) = \sigma(z_l)$
end for

Backpropagation

Algorithm 4 Backpropagation

Input: Example x, label y, parameters $[W_{\{2:L\}}, b_{\{2:L\}}]$

Output: Derivatives $\{\frac{\partial \mathcal{L}_{MSE}}{\partial b_l}\}_{l=2:L}, \{\frac{\partial \mathcal{L}_{MSE}}{\partial W_l}\}_{l=2:L}$

Compute $z_l(x), a_l(x)\ \forall l = 1 : L$ with a forward pass

$\delta_L := \frac{\partial \mathcal{L}_{MSE}}{\partial a_L} \circ \sigma'(z_L(x))$

for $l = L : 2$ **do**

$\quad \frac{\partial \mathcal{L}_{MSE}}{\partial b_l} := \delta_l$

$\quad \frac{\partial \mathcal{L}_{MSE}}{\partial W_l} := a_{l-1}\delta_l^T$

$\quad \delta_{l-1} := (W_l \delta_l) \circ \sigma'(z_{l-1}(x))$

end for

Outline

Parallelizing Gradient Descent

Two ways to parallelize:

- **Parallelize Gradient Descent**:
  Derivative of the loss function has the following form:

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

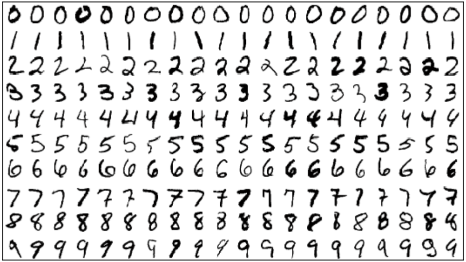
  Distribute training examples, compute partial gradients, sum up partial gradients

- **Parallelize Backpropagation**:
  Parallelize matrix vector multiplications in forward propagation and backpropagation algorithms

Outline

MNIST dataset

- 28x28 images of handwritten digits
- 50,000 training examples, 10,000 test examples, 10,000 validation examples
- Labels: 0 to 9 (one-hot encoding)



- **Network structures**

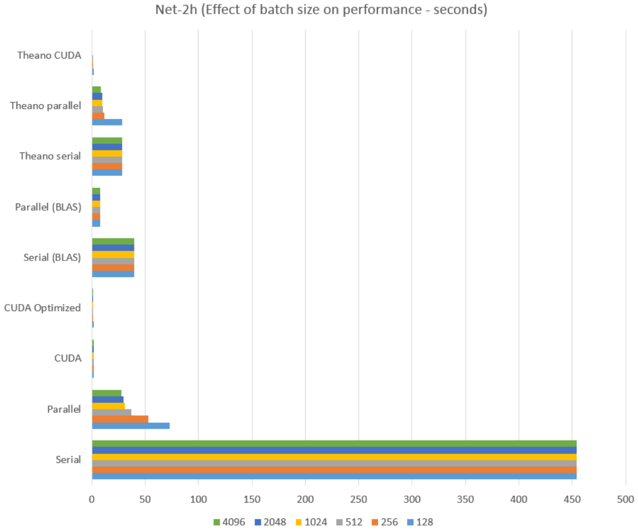
	# Layers (In,**Hidden**,Out)	# Nodes (In,**Hidden**,Out)	# Num Params
Network1	1,**1**,1	784,**1024**,10	800,000
Network2	1,**2**,1	784,**1024,1024**,10	1,860,000

- Serial, Parallelize over examples (Pthreads, CUDA)
- Serial (BLAS), Parallelize matrix computations (BLAS)
- Serial (Keras:Theano), Parallel (Keras:Theano), GPU (Keras:Theano)

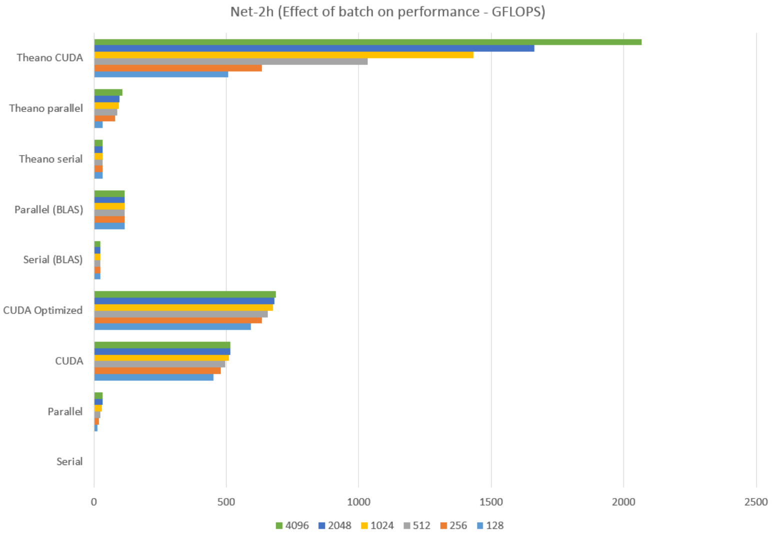
Analyze time per epoch, gigaflops for each implementation
Analyze speedup from parallelization over serial counterparts

Outline

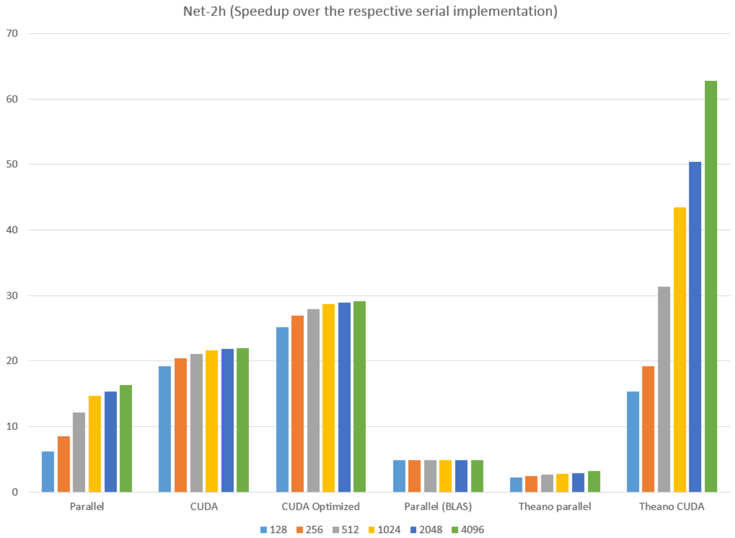
Results - Time per Epoch



Results - Gigaflops



Results - Speedup



Analysis

- **Our implementation**
  - Parallel computing average speedup ≈ 10
  - Training time decreases as minibatch size decreases
- **BLAS**
  - Parallelizing each matrix vector product gives even faster results
  - Speedup independent of batch size, but less than our implementation
- **CUDA**
  - Our CUDA implementation gives about $\approx 20x$ speedup
  - If # neurons per layer are not perfect multiple of 32 then some threads do not participate in computation
- **Theano**
  - Apparently combines both types of parallelization
  - Theano CUDA scales very fast with batch size

Future Work

Combine the two parallelization techniques: Split training examples amongst threads, further hierarchically parallelize matrix computations for each individual example.

Thank you

Questions?