Parallelization of a Learning Algorithm for Neural-Network Decision Support

Nathan DeBardeleben
William Jones

Parallel Architecture Research Laboratory (PARL)
Clemson University, Clemson, South Carolina
http://www.parl.clemson.edu/
Neural network overview (Jones)
Algorithm from paper (Jones)
Parallel computer overview (DeBardeleben)
Parallelize algorithm (DeBardeleben)
Neural Network Overview

- Determine relationships
- Problem solving
- System features not known *a priori*
- Mapping from input to output space
- Identify salient features of system
Basic Building Block—Node

- Artificial neuron or unit
- Weighted Linear Input Combination (WLIC)
- Net activation: $net_i$
- Squashing function: $f(net_i)$
- Propagate output
Network Characterization

- Topology
  - Feed-forward (no feedback)
  - Recurrent (feedback)

- Node characteristics

- Number of nodes

- Layers, *if any*
Feed-Forward Neural Network
Training Algorithm

Training Set H

Adjust Network Parameters

Input Pattern

Output Pattern

Hidden Layers

Input Layer

Output Layer

Training Strategy
Training Algorithm Depends On:

- Node characteristics
  - squashing function
  - WLIC
- Topology
- Effectiveness metric
  - Input to output space mapping
  - Error measure (objective function)
Gradient Descent

- Initialize all weights – Gauss(0,1)
- Apply input vector to network
- Propagate vector forward and obtain unit outputs
- Compare output layer response with desired outputs
- Compute and propagate error measure backward, correcting weights layer by layer
- Iterate until ”good” mapping is achieved
Derivation of Gradient Descent

\[ E = \sum_p E_p = \frac{1}{2} \sum_p \sum_j (o_{pj} - a_{pj})^2, \]

\[ net_j = \sum_i w_{ij}a_i \quad (1) \]

\[ a_i = f(net_i) = \frac{1}{1 + e^{-net_i}}, \]

\[ \frac{\partial net_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj}a_{pk} = a_{pi}, \]

\[ \frac{\partial a_{pj}}{\partial net_{pj}} = f'(net_{pj}), \]

\[ \Delta_p w_{ij} \propto \frac{\partial E_p}{\partial w_{ij}} = -\frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ij}}. \quad (2) \]

\[ \delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial a_{pj}} \frac{\partial a_{pj}}{\partial net_{pj}}. \quad (3) \]

\[ \frac{\partial E_p}{\partial a_{pj}} = -(o_{pj} - a_{pj}). \quad (4) \]

\[ \delta_{pj} = (o_{pj} - a_{pj})f'(net_{pj}). \quad (5) \]
The objective function:

$$E^p = \frac{1}{2} \sum_p \sum_j (o_{pj} - a_{pj})^2.$$  \hspace{1cm} (6)

The weight correction is

$$\Delta_p w_{ij} = \epsilon \delta_j^p a_i^p,$$ \hspace{1cm} (7)

where $\epsilon$ is the learning rate.

For output units the $\delta$'s are

$$\delta_j^p = (o_j^p - a_j^p) f'_j(net_j^p).$$ \hspace{1cm} (8)

For internal units the $\delta$'s are

$$\delta_j^p = f'_j(net_j^p) \sum_n \delta_n^p w_{nj},$$ \hspace{1cm} (9)

where $\delta_n$'s are from layer $(L_{k+1})$.

Assuming a sigmoid activation (squashing) function,

$$f'_j(net_j^p) = \frac{d}{dnet_j^p} \left( \frac{1}{1 + e^{-net_j^p}} \right) = a_j^p(1 - a_j^p).$$ \hspace{1cm} (10)
Learning Convergence Rates

- Depends on:
  - Topology (layers, num. nodes)
  - Random initial weights
  - Size of training set

- When to stop?
  - Memorization
  - Generalization
Computation

- Expensive (time)
- Limited resources
- Speed-up
- Research paper
Motivation: speed-up training

Requirements

- Maintain learning convergence
- Maintain Accuracy

Outlines new BP algorithms

Modifications to GDR equations (output layer)

\[
\delta_j^p = 4(o_j^p - a_j^p)^3 e^{(o_j^p-a_j^p)^2} f'_j(\text{net}_j^p). \tag{11}
\]
Project: Phase I

- Implement
  - Common gradient descent algorithm
  - Modified BP algorithm
  - Verify results in paper

- Can we make it faster?
  - Sequential execution
  - Parallel execution
Why Parallelism?

- Further speed-up of training
- Neural networks exhibit high degree of parallelism

Process of parallelism:

- What type of machine?
- How to parallelize?
Parallel Computing

- Multiple processors

- Two paradigms:
  - Shared memory
  - Distributed memory
Shared Memory
Distributed Memory

Interconnection Network

PE1  PE2  PE3  \ldots  PEn
Memory  Memory  Memory  \ldots  Memory
Parallel Systems

Composed of two things:

- Architecture
- Algorithm
Problem Granularity

- Time between communication
  - Fine grain
    - Frequent communication
    - Shared memory solutions
  - Coarse grain
    - Infrequent communication
    - Distributed memory solutions
Parallelism Considerations

- What type of architecture is appropriate?
- Can the problem be broken into independent pieces?
- What are the independent pieces?
- How much communication will be required between pieces?
- How can we minimize network traffic?
Neural Network Parallelism

- Aberdeen et al. use vector operations
- Parallelism across the training set
- Communication overhead
Decomposition

```
Blocked Training Set

PE1 PE2 PE3 PEn
...
Master
...
HnH3H1 H2 ...
Slaves
return delta wij’s
distribute weights

```

```
Blocked
Training Set

PE1 PE2 PE3 PEn
...
Master
...
HnH3H1 H2 ...
Slaves
return delta wij’s
distribute weights

```
Implementation

- Cluster of computers
- Master / slave C program
- Message Passing Interface (MPI)

Potential problems:

- Quantity of data
- Overlap of communication and computation