8-2009

ACCURACY AND MULTI-CORE PERFORMANCE OF MACHINE LEARNING ALGORITHMS FOR HANDWRITTEN CHARACTER RECOGNITION

Sumod Mohan
Clemson University, sumodm@clemson.edu

ACCURACY AND MULTI-CORE PERFORMANCE OF MACHINE LEARNING ALGORITHMS FOR HANDWRITTEN CHARACTER RECOGNITION

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Sumod K Mohan
August 2009

ABSTRACT

There have been considerable developments in the quest for intelligent machines since the beginning of the cybernetics revolution and the advent of computers. In the last two decades with the onset of the internet the developments have been extensive. This quest for building intelligent machines have led into research on the working of human brain, which has in turn led to the development of pattern recognition models which take inspiration in their structure and performance from biological neural networks. Research in creating intelligent systems poses two main problems. The first one is to develop algorithms which can generalize and predict accurately based on previous examples. The second one is to make these algorithms run fast enough to be able to do real time tasks. The aim of this thesis is to study and compare the accuracy and multi-core performance of some of the best learning algorithms to the task of handwritten character recognition. Seven algorithms are compared for their accuracy on the MNIST database, and the test set accuracy (generalization) for the different algorithms are compared. The second task is to implement and compare the performance of two of the hierarchical Bayesian based cortical algorithms, Hierarchical Temporal Memory (HTM) and Hierarchical Expectation Refinement Algorithm (HERA) on multi-core architectures. The results indicate that the HTM and HERA algorithms can make use of the parallelism in multi-core architectures.

DEDICATION

This work is dedicated to my grand-parents, parents and sister.

ACKNOWLEDGMENTS

This work would not have been possible without whole hearted support of many people. First and foremost I would like to thank my adviser, Dr. Tarek Taha for his patience, guidance, and support. I must thank Dr. Birchfield for the teaching me all the important things and for the positive guidance throughout. Next, I would like to thank Dr. Ligon for all the support. I would like to thank my friends Mohammed Ashraf, Rommel Jalasutram, Vidya N Murali, Kenneth Rice and Pavan Yalamanchali for all their help and support during my research.

I would like to thank Dexter Stowers, Scott Duckworth and Dr. Wayne Madison for all the support and the great ambiance at the CES Unix Help Desk. Last but not the least, I would like to thank research community in general, Clemson University and Georgia Institute of Technology for the resources which made this research possible.

TABLE OF CONTENTS

Table of Contents (Continued)

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Intelligent machines were part of myths in most of the ancient world. Considerable real development began with the cybernetics revolution and the advent of personal computers. Honda's humanoid robot 'Asimo', Stanford University's Autonomous vehicle 'Stanley', and IBM's chess playing computer 'Deep Blue' are its products. Improvements in inference capabilities and real time performance would benefit many areas of research including speech processing, computer vision, data mining, robotics, and computer games.

1.2 Related Work.

From the earliest models, we have taken inspiration in the structure and performance of biological neural networks. McCulloch and Pitts [38] proposed the McCulloch-Pitts model of a neuron, which performed weighted sum of inputs followed by thresholding. John McCarthy coined the term "artificial intelligence” and also invented the Lisp language. Rosenblatt [46] proposed the perceptron model, in which the perceptron learning rule adjusted the input weights, and Minsky et al. [40] demonstrated the limitations of the multilayer perceptron model. Kohonen came up with the idea of associative memories [29], Vapnik and Chervonenkis established the VC dimension theorem [49], which is a measure of capacity of a statistical classification algorithm. Adaptive Resonance Theory was formulated by Grossberg in 1976 [20], and Barto et al.

[6] originated the idea of reinforcement learning. Hopfield invented the recurrent neural networks in 1983 and in 1989 Judea Pearl formalized the concepts of Bayesian Networks [43]. In 1995, Cortes et al. [10] proposed the concepts of support vector machines. Convolutional Neural Networks was introduced by LeCun et al. [32], and Geoffrey E. Hinton along with Terry Sejnowski invented Boltzmann machines [23]. Some of the most interesting applications were ALVINN which stands for Autonomous Land Vehicle In a Neural Network by Dean Pomerleau, Honda's Humanoid robot 'Asimo', Stanford University's DARPA 2007 Grand Challenge winning autonomous vehicle 'Stanley' and IBM's chess playing computer 'Deep Blue'.

There has been number of studies on the comparison of classifiers. One of the oldest and most comprehensive was the works of King et al. [28], which compared symbolic, statistical and neural networks based algorithms. But this study was conducted in 1995, and hence many of the newer algorithms are missing. LeCun et al. [31], compared different classifiers for the problem of handwritten character recognition. Caruna et al. [9], compared different classifiers for many different problems and reported that performance was problem specific. More recently, Neeba et al. [42], compared different features and statistical and neural networks based classifiers for the problem of handwritten character recognition. But all the above studies have not included the newer hierarchical Bayesian based cortical models.

Recently, several models of the neocortex have been proposed that are based on modeling mini-columns/columns [3][11][18][27]. The models by Dean [11], George and Hawkins [18], and Anderson [3] are based on hierarchical graphical networks and concur well with experimental results. They describe the brain as a hierarchical device that computes by performing sophisticated pattern matching and sequence prediction. Johansson and Lansner [27] utilized a cluster of 442 dual Xeon processors to simulate a randomly connected brain model utilizing recurrently connected neural networks grouped into cortical columns. Anderson et. al [4] are examining the design and implementation of large scale cortical models based on the brain state in a box model [3].

Several studies have examined the acceleration of various models on multi-core architectures. Wu et al. [52] are examining the acceleration of the brain state in a box model [2] on the Cell processor in a Playstation 3. They achieve about 70% of the theoretical peak of the processor. Felch et al [17] examined the acceleration of the Brain Derived Vision algorithm on the Cell processor. They achieved a speedup of 140 times using a cluster of three Sony Playstation 3 systems over a serial implementation on 2.13 GHz Intel Core processor. Xia and Prasanna examined the parallelization of the exact inference algorithm in junction trees [53] and examined its acceleration on a Sony Playstation 3 based Cell processor [54]. They achieve a speedup of about four times a 3.0 GHz Intel Pentium 4 processor.

1.2 Overview

This thesis studies the two of the problems faced in artificial intelligence research, when applied to hand written character recognition. The first problem is accuracy of the generalization behavior and the second one is that of performance, speed, and the ability of the algorithm to work in real time.

For the first problem, seven classification algorithms from two different classes of algorithms are compared against each other. The first class of algorithms is based on statistical machine learning. Naïve Bayes (NB) is the first algorithm from this class, which is a simple probabilistic classifier with strong independence assumptions. The K Nearest Neighborhood classifier (KNN) labels a test case with the maximum occurring label of the k nearest training examples. The Neighborhood Component Analysis proposed by Goldberger et al., is an improvement of KNN which searches for the optimal distance parameter for the KNN algorithm in the Mahalanobis quadratic distance space. The next algorithm of this class, the Support Vector Machines invented by V. Vapnik et al. [50], is a binary classification algorithms which finds the optimal hyperplane which divides the two classes with the maximum margin. The last algorithm of this class of algorithms is the neural networks based algorithm which have taken inspiration from the biological neural networks. The Multilayer Perceptron Algorithm (MLP) which uses the back propagation algorithm, is a non linear statistical learning algorithms which adapts its structure during the learning phase based on the information flowing through the network.

The second class of algorithms are based on Hierarchical Bayesian Networks pioneered by Judea Pearl [43]. The first algorithm called Hierarchical Temporal Memory(HTM) and developed by Jeff Hawkins and Dileep George [18], models the structural and algorithmic properties of the neocortex. The second algorithm, Hierarchical Expectation Refinement Algorithm (HERA) was proposed by Thomas Dean [13]. All the above algorithms were compared on four different data sets of varying sizes.

The second problem of performance is studied by comparison of the Hierarchical Bayesian networks on different multi-core architectures. The HTM algorithm and HERA, both were implemented on four different multi-core architectures, the Intel Xeon Blade, Sony Play Station 3, IBM QS 20 and Sun SPARC T5140.

The main contributions of this work are:

1) An empirical study of the accuracy of the leading off the shelf learning algorithms for the problem of handwritten character recognition.
2) Parallelization study of two hierarchical Bayesian cortical models. Both thread level parallelization and the data level parallelization of the models are examined.
3) A study of different optimizations and parallelization strategies for multi-core implementations of the models. This thesis examines the performance of the models on three multi-core processors using four platforms (a Sony Playstation 3, an IBM QS20 blade, a Sun Enterprise 5140 server, and a dual processor Intel Xeon blade). Several differnt sizes of the model networks were implemented to examine the effect of scaling.

1.4 Organization

The second chapter explains the statistical machine learning based algorithms and the third chapter explains the Hierarchical Bayesian Networks based algorithms and the fourth chapter explains experimental setup and implementation. The fifth chapter discusses the results and the last chapter draws the conclusions and future work.

CHAPTER TWO

STATISTICAL MACHINE LEARNING ALGORITHMS

Statistical machine learning merges statistics with learning theory and is applied to large-scale, dynamical and heterogeneous data streams. One of the main research interest in this area is to understand the relation between inference and computational requirements. Some of the leading algorithms are described in the following sections. Section 2.1 and 2.2 describes the simplest statistical algorithms, namely the naïve Bayes algorithm and the K nearest neighbor algorithm. Section 2.3 describes a generalization to K nearest neighbor algorithm, and 2.4 explains the Support Vector Machines algorithm. Finally section 2.5 explains the Artificial Neural Network algorithm based on the Multilayer Perceptron.

2.1 Naïve Bayes

The simplest algorithm is the naïve Bayes (NB) classifier which makes the strong assumption of conditional independence of the each element of the feature space with respect to the class label.

Let X={x_1,x_2,..x_m} and x_1={x_{11},x_{12},...x_{1n}} represent m training examples of n sized features.

Let Y={y_1,y_2,..y_m} represents the labels of the m training examples.

$$\varphi_{(j\,|y=1)} = \frac{\sum \left(1\left\{x_{ij}=1 \wedge y_i=1\right\}\right)}{\sum \left(1\left\{y_i=1\right\}\right)}$$

$$\varphi_{(j|y=0)}=\frac{\sum\left(1\{x_{ij}=1\wedge y_i=0\}\right)}{\sum\left(1\{y_i=0\}\right)}$$

$$\varphi_y=\frac{\sum\left(1\{y_i=1\}\right)}{m}$$

The naïve Bayes assumption is that x_i's are conditionally independent given y. So for a binary classifier, the parameter to be learned is the maximum likelihood of $\varphi_{(j|y=0)}$ and $\varphi_{(j|y=1)}$. The parameter $\varphi_{(j|y=1)}$ is the fraction that feature j contributes to class with label 1, $\varphi_{(j|y=0)}$ is the fraction that the feature contributes to class with label 0 and $\varphi_{(y=1)}$ is the prior, or the probability of the class with label 1. Now the class label for a training set can be calculated by the Bayes rule.

$$p(y=1|x)=\frac{\left(p(x|y=1)*p(y=1)\right)}{\left(p(x)\right)}$$

In addition to the above in case of small training sets, some $\varphi_{(j)}$ may be calculated as zeros, which can severely distort the probabilities for some classes. So, Laplace smoothing is applied to avoid this.

2.2 K Nearest Neighbor (KNN)

The KNN is one of the simplest algorithms, which labels the test set example with the class label of the majority of the training set labels in its neighborhood. The distance parameter is used to determine the neighborhood of the test label. In our experiments, the study compares the Euclidean and Manhattan distance parameters. The advantages of this algorithm includes simplicity, non linear decision boundary, single parameter that needs to be tuned, and that accuracy increases with training set size. The disadvantages of the algorithm are that accuracy varies with the distance parameters, high computational cost with large databases, and the need to keep the database even for testing.

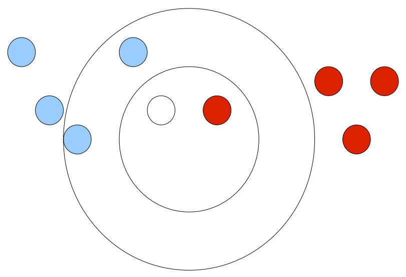

Fig 2.1 KNN Classification

2.3 Neighborhood Component Analysis (NCA)

This algorithm was proposed by J Goldberger et al. [62], it is an extension of the KNN algorithm. The idea is to convert the problem such that gradient descent can be done on the error function by varying the distance metric. The error function in case of the k nearest neighbors is discontinuous with respect to the distance metric, so the stochastic random neighbor selection function is used, where p_{ij} is the probability that a point i chooses j as its neighbor, by leave one out cross validation.

Training Algorithm

1. Start with A as identity matrix, LDA matrix
2. Continue steps 3 and 4 till convergence
3. Optimize by gradient descent

$$\varphi = \frac{1}{N} \sum \sum \frac{e^{-d_{ij}}}{\sum \exp^{-d_{ik}}}$$

$$d_{ij} = \left(x_i - x_j\right)' Q \left(x_i - x_j\right)$$

4. $$d_{ij} = \left(x_i - x_j\right)' AA' \left(x_i - x_j\right)$$

$$\frac{\partial \varphi}{\partial A} = 2\mathrm{A} \sum \left(p_i^{(+i)} \sum p_{ik} x_{ik} x_{ik}' - p_i^{(-i)} \sum p_{ij} x_{ij} x_{ij}' \right)$$

$$p_{ij} = \frac{e^{-d_{ij}}}{\sum \exp^{-d_{ik}}} \qquad p_{ii} = 0$$

Testing Algorithm

1. Convert the original input to the new feature space by multiplying with the learned kernel A.
2. Do KNN for the nearest neighbor with Euclidean distance.

2.4 Support Vector Machines

Support vector machines are a highly effective machine learning technique for many classes of problems. SVMs construct a hyperplane which maximizes the margin between the two classes. The problem of finding the optimal solution to the SVM problem is non convex by the basic formulation. Therefore the dual of the problem is considered which is convex. Then the Sequential Minimal Optimization (SMO) algorithm by John C. Platt [60] is used for the optimization of this convex problem. The memory requirement of SMO algorithm is between linear and quadratic in the training set size, depending on the problem. A simplified version of the SMO algorithm as explained in [60] is described below,

```
Algorithm
  C = Regularization parameter
  tol = numerical tolerance
  $\alpha \in R^m : Lagrange Multiplier$
  $b \in R : threshold$

  Repeat till convergence
    for $i = 1 \ldots m$
      $f(x_i) = \sum \alpha_i y_i \langle x_i, x \rangle + b$
      $E_i = f(x_i) - y_i$
      $if\left(\left(y_i E_i < -tol \,\&\&\, \alpha_i < C\right) \;||\; \left(y_i E_i > tol \,\&\&\, \alpha_i > C\right)\right)$
            $select\ j \neq i\ randomly$
            $E_j = f(x_j) - y_j$
            $\alpha_i^{old} = \alpha_i,\ \alpha_j^{old} = \alpha_j$
            $if\ y_i \neq y_j,\ L = max(0, \alpha_j - \alpha_i),\quad H = min(C, C + \alpha_j - \alpha_i)$
            $if\ y_i = y_j,\ L = max(0, \alpha_i + \alpha_j - C),\quad H = min(C, \alpha_i + \alpha_j)$

            $if\ (L = H)$
                  $continue$ to $next\ i$.
            $Compute\ \eta = 2\langle x_i, x_j \rangle - \langle x_i, x_i \rangle - \langle x_j, x_j \rangle$
            $if\ (\eta \geq 0)$
                  $continue$ to $next\ i$
            $\alpha_j = \alpha_j - y_j (E_i - E_j) / \eta$
```

```
                Set $\alpha_j = (H \quad if\, \alpha_j > H)$
                        $(\alpha_j \quad if\, L \le \alpha_j \le H)$
                        $(L \quad if\, \alpha_j < L)$
                if $(|\alpha_j - \alpha_j^{old}| < 10^{-5})$
                        continue to next i
                Determine value for $\alpha_i$ $b_1$ and $b_2$
                $\alpha_i = \alpha_i + y_i y_j (\alpha_j^{old} - \alpha_j)$
                $b_1 = b - E_i - y_i(\alpha_i - \alpha_i^{old})\langle x_i, x_i\rangle - y_j(\alpha_j - \alpha_j^{old})\langle x_i, x_j\rangle$
                $b_2 = b - E_j - y_i(\alpha_i - \alpha_j^{old})\langle x_i, x_i\rangle - y_j(\alpha_j - \alpha_j^{old})\langle x_j, x_j\rangle$
                                                                        Compute b
                $b = \{b_1 \quad if \quad 0 < \alpha_i < C\}$
                    $\{b_2 \quad if \quad 0 < \alpha_j < C\}$
                    $\{(b_1 + b_2)/2 \quad otherwise\}$
end loop
```

2.5 ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks are computational models which try to simulate the structural and functional aspects of biological neural networks. The network is formed by interconnected nodes, called neurons. Each neuron is presented by the weighted input, on which the neuron does a functional mapping to the output.

Let $x_1, x_2, \ldots x_n$ be the inputs to the neuron and $w_1, w_2, \ldots w_n$ be weights of the inputs to the neuron. Each neuron does two computations, summation followed by logistic function or sigmoid function. Therefore the output is

$$output \;=\; g\left(\sum x_i w_i\right), \;\; where g(x) = 1/\left(1 + e^{-x}\right)$$

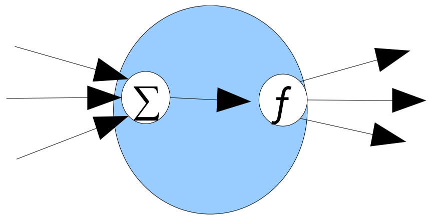

Fig 2.2 Node operation.

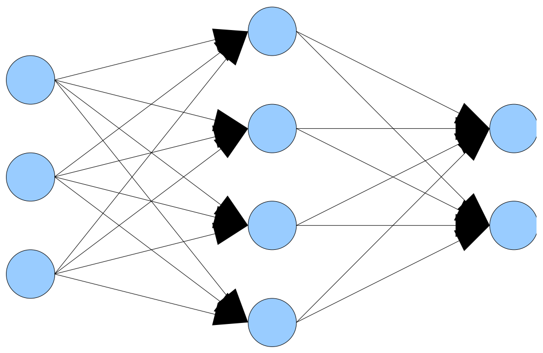
Fig 2.3 Network Architecture

The network architecture is completely connected, except in case of the neurons in the same layer as shown in figure 2.3 above.

Neural Network with one hidden layer

$x_{11}, x_{21}, \ldots x_{n1}$: n inputs to first neuron of the hidden layer

$w_{11}, w_{21}, \ldots w_{n1}$: weight of those inputs

$X = \{x_{11}, x_{21}, \ldots x_{n1}; x_{21} \ldots x_{nm}\}$

$W = \{w_{11}, w_{21}, \ldots w_{n1}; w_{21} \ldots W_{nm}\}$

$X = \{X; bias\}$

L is the learning rate, T is target labels.

```
                         Training Algorithm
Repeat till Convergence
1.   $Net_h = \sum X * W_h$                  : Net of the hidden layer.
2.   $O_h = 1/\left(1+\exp\left(-Net_h\right)\right)$        : Output of the hidden layer.
3.   $I_o = \{O_h ;\text{bias}\}$  : Input to output layer is output of hidden unit and the bias.
4.   $Net_o = \sum I_o W_o$                  : Net of the output layer.
5.   $O = 1/\left(1+\exp\left(-Net_o\right)\right)$          : Output of the output layer.
6.   $E = T - O$                             : Error
7.   $\delta_o = O.*(1-O).*E$                : Sensitivity of output units.
8.   $\delta_h = Oh.*(\delta_o * W_o)$       : Sensitivity of hidden units.
9.   $w_{\delta h} = L * X * \delta_h$       : Change in weights of hidden units.
10.  $w_{\delta o} = L * I_o * \delta_o$    : Change in weights of output units.
11.  $W_h = \mathrm{W}_h + \mathrm{w}_{\delta h}$     : Weight correction for hidden units.
12.  $W_o = \mathrm{W}_o + \mathrm{w}_{\delta o}$     : Weight correction for output units.
end

                         Testing Algorithm
 X is test input, Wh and Wo from training.
1.   $Net_h = \sum X * W_h$                  : Net of the hidden layer,
2.   $O_h = 1/\left(1+\exp\left(-Net_h\right)\right)$        : Output of the hidden layer.
```

3. $Io = \{O_h ; \text{bias}\}$: Input to output layer is output of hidden unit and the bias.

4. $Net_o = \sum I_o W_o$: Net of the output layer.

5. $O = 1/(1 + \exp(-Net_o))$: Output of the output layer.

CHAPTER THREE

HIERARCHICAL BAYESIAN CORTICAL MODELS

3.1 Hierarchical Temporal Memory model

George and Hawkins developed an initial mathematical model [18] of the neocortex based on the framework described by Hawkins in [21]. Their model utilizes a hierarchical collection of nodes that employ Pearl's Bayesian belief propagation algorithm [43]. As shown in Figure 3.1, each node has one parent and multiple children (hence there is no overlap in the input fields of any two nodes in a given layer). Input data is fed into the bottom layer of nodes (level 1) after undergoing some pre-processing. After a set of feed-forward and feedback belief propagations between nodes in the network, a final belief is available at the top level node. This belief is a distribution that indicates the degree of similarity between the input and the different items the network has been trained to recognize. The model is trained in a supervised manner by presenting a set of training data to the bottom layer of nodes multiple times.

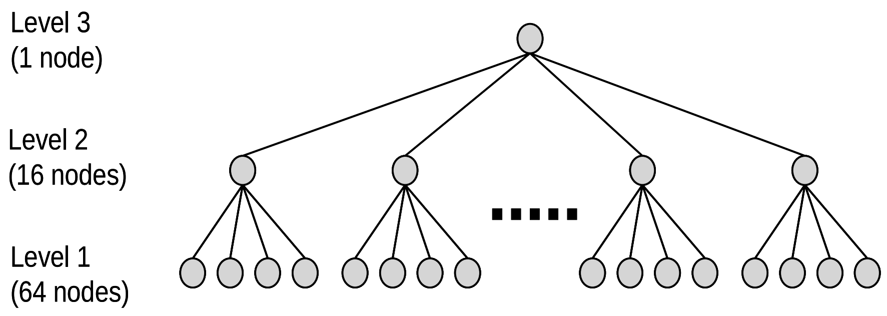

Fig 3.1 HTM Architecture

The computational algorithm within each node of the model is identical and follows equations 1 through 6 below. Before a node starts computing, it receives belief vectors from its parent (π) and children (λ) as shown in Figure 3.2 (a). The belief vectors from its children are all combined together as shown in equation 1. This combined belief vector from the children is then multiplied with an internal probability matrix, P_{xu} (generated in an offline training phase), and the belief vector from the parent (see equation 2). The matrix multiplications are carried out element-by-element. A set of belief vectors are then generated for the parent and child nodes (equations 3 to 6). These output belief vectors are then transmitted to the parent and children of the node as shown in Figure 3.2(b).

$$\lambda_{product}[i] = \prod \lambda_{in}[child][i] \qquad (1)$$
$$F_{xu}[j][k] = \pi_{in}[j] \times P_{xu}[j][k] \times \lambda_{product}[k] \qquad (2)$$
$$m_{row}[j] = max(m_{row}[j], F_{xu}[j][k]) \qquad (3)$$
$$m_{col}[k] = max(m_{col}[k], F_{xu}[j][k]) \qquad (4)$$
$$\lambda_{out}[j] = m_{row}[j] / \pi_{in}[j] \qquad (5)$$
$$\pi_{out}[child][k] = m_{col}[k] / \lambda_{in}[child][k] \qquad (6)$$

Table 3.1 Equations describing the HTM Model

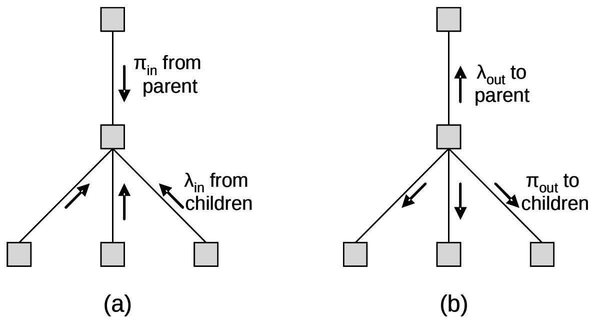

Fig 3.2 HTM Belief Propagation. (a) Node gathering beliefs (b) Node distributing beliefs.

3.2 Hierarchical Expectation Refinement Algorithm

Thomas Dean proposed a new hierarchical Bayesian model of the visual cortex, called the Hierarchical Expectation Refinement Algorithm[12]. This model consists of a layered collection of nodes as shown in Figure 3.3. Input data is presented to the bottom layer of nodes (generally after some pre-processing) and a final inference based on this input is produced by the top layer node. All the nodes in the network carry out the same set of computations and can be considered to be the functional equivalent of cortical columns. The model is trained in a supervised manner by presenting a set of training data to the bottom layer of nodes multiple times.

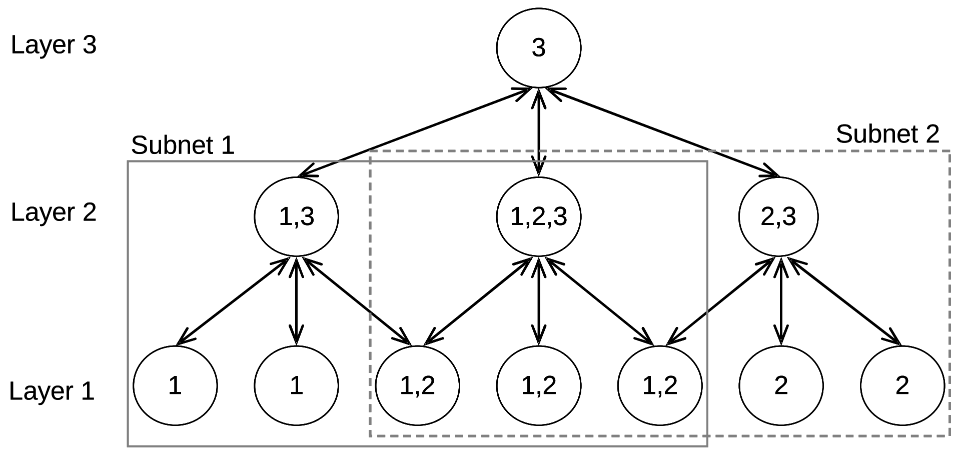

Figure 3.3: An Example of Thomas Dean's hierarchical Bayesian network model. This example can be divided into three subnets as shown and the nodes are numbered with the subnets they belong to.

In the example implementation of the model presented by Dean in [4], the model performs hand written character recognition on 28×28 pixel images. This example network consists of three layers of nodes connected in a pyramidal form, with the bottom layer consisting of 49 nodes (in a 7x7 layout), the middle layer of 9 nodes (in a 3x3 layout), and the top layer of 1 node. Each layer 2 node has nine layer 1 children (arranged in a 3x3 layout) forming a pyramidal collection. The field of view of each layer 2 node overlaps with its neighbors' by an edge that is one node thick. Thus, each layer one node can have up to four layer 2 parents.

The input image is preprocessed by a preprocessing layer before being fed to the layer 1 nodes. Each layer 1 node has a 4x4 patch of pixels corresponding to it. In the preprocessing layer, the 4x4 patch of pixels is transformed into a mixture of Gaussians and this mixture is matched against 16 predefined classes of mixtures of Gaussians. Thus each 4x4 pixel region is represented by a number between 1 and 16, with this number being fed to the corresponding layer 1 node by the preprocessing layer.

The network can be divided into a set of modular component subnets (as shown in Figure 3.3). Each subnet has two layers of nodes. A subnet can be defined as a node, its parents, and all the children of those parents in the same level as the original node [4]. The function of each subnet is to produce an abstract set of features that are seen by the lower level subnets feeding into it. Neighboring subnets have overlaps in their receptive fields to enable the network to more robustly recognize invariant features. Hence a node could belong to multiple subnets (as shown in Figure 3.3). The subnets are identified during the training process and only the largest subnets (those that would not be a subset of another subnet) are utilized. For any given input image, the network is processed through multiple bottom-to-top-to-bottom passes. In each pass, all the subnets for a certain layer are processed before moving to the next layer.

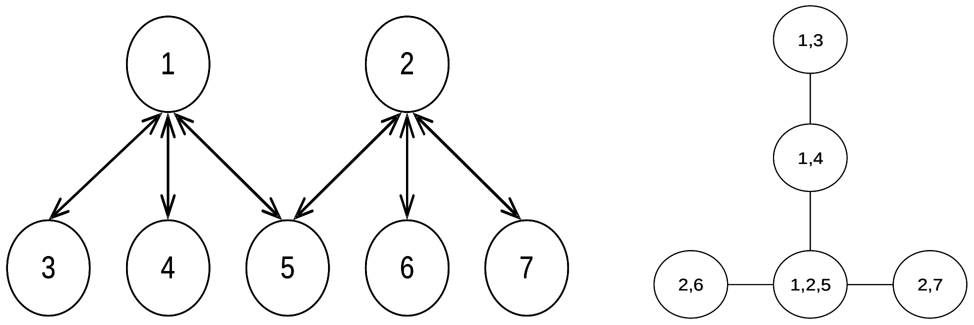

Figure 3.4: An Example of a junction tree derived from one of the lower level subnets shown in Figure 3.1. Part (a) shows the subnet with the nodes number 1 through 7. Part (b) shows the junction tree equivalent to this subnet. Each clique in part (b) is numbered with the corresponding nodes from the subnet that are used to build the clique.

In order to process a subnet, it is first converted to its equivalent junction-tree representation. The junction-tree consists of a set of nodes called cliques, where each clique is a collection of nodes in the original subnet. The connection between the cliques is called a separator and is labeled by the nodes in common between the two cliques. Figure 3.4 shows a simple subnet and its equivalent junction tree decomposition. Although this junction tree has only 5 cliques (with two that can be evaluated in parallel), the junction trees in the networks examined have up to 25 cliques (with up to 21 that can be evaluated in parallel). The subnet to junction tree mapping is carried out during training and does not have to be redone during inference (as the mapping is reused). The junction tree for a subnet is evaluated in a single bottom-to-top and then top-to-bottom pass. The Lauritzen and Spiegelhalter's junction-tree algorithm [30] is utilized for exact inference in the tree. The operations consist primarily of element by element multi-dimensional matrix adds, multiplies, and divides.

CHAPTER FOUR

EXPERIMENTAL SETUP AND IMPLEMENTATION

4.1 Algorithmic Implementation

All the algorithms were presented with the 60,000 training images from the MNIST training data set. Each image is 28x28 in size with 8 bit gray scale resolution, thus each algorithm was presented with a 784 bit vector for each training image. The naive Bayes was implemented in Matlab with Laplace smoothing to avoid distortion. The KNN algorithm was implemented in Matlab and was tested using Euclidean and Manhattan distances. The support vector machines implementation was based on the sequential minimal optimization [60], and the pseudo code as explained in [61] was used for the development of the algorithm. One versus all classification as explained in [50] was used for multi-class classification. The artificial neural networks was implemented by using the MLP algorithm in Matlab. The HTM algorithm was implemented using the Nupic 1.6 SDK, provided by Numenta Inc. and HERA was implemented by using the software provided by Thomas Dean [62].

4.2 Multi-core architectures

The problem of heat dissipation has put to hold the frequency scaling of processors by miniaturization of transistors and has taken the semiconductor industry to to explore multi-core architectures. This thesis examines three different multi-core architectures, Intel Xeon E5345, Sun Ultra SPARC T2 Plus, and STI Cell BE architectures.

The Intel Xeon E5345 processor examined contains four Intel Core based processing cores clocked at 2.33 GHz. These processors contain a 256 KB level one cache per core and an 8 MB shared level two cache. The processor can execute vector instructions (with four floating point operations) using the SSE3 instruction set.

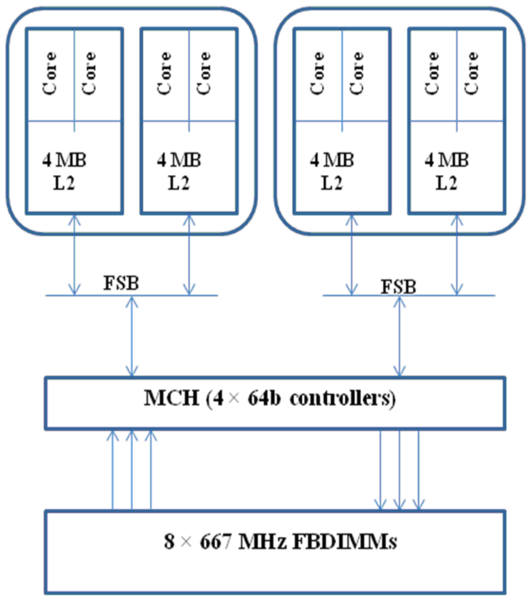

Fig 4.1 Intel Xeon E5345 processor architecture. [59]

The Sun Ultra SPARC T2 Plus processor [48] contains 8 cores running at 1.4 GHz. Each core can execute up to eight threads simultaneously, with up to two threads in each pipeline stage. Thus the entire processor can run a maximum of 64 threads concurrently. Each core contains 8 KB of data and 16 KB of instruction cache, and share a 4 MB level two cache.

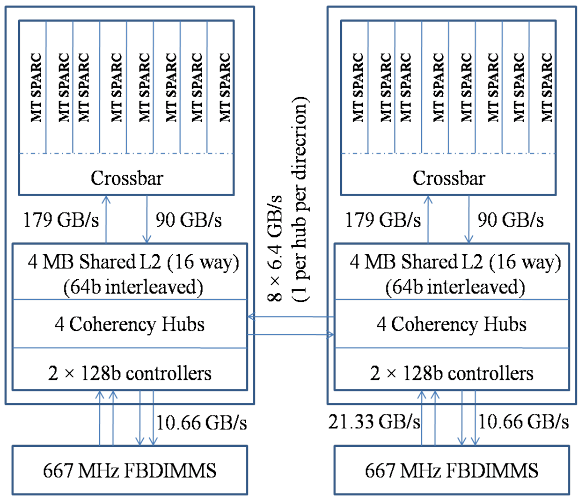

Fig 4.2 Sun UltraSPARC T2 Plus processor architecture. [59]

The Cell Broadband Engine developed by IBM, Sony, and Toshiba [20] is a multi-core processor that heavily exploits vector parallelism. The current generation of the IBM Cell processor consists of nine processing cores: a PowerPC based Power Processor Unit (PPU) and eight independent Synergistic Processing Units (SPU). The processor operates at 3.2 GHz. Each SPU is capable of processing up to four instructions in parallel each cycle (eight, if considering fused multiply-add instructions). The processing cores in the Cell utilize in-order execution with no branch prediction.

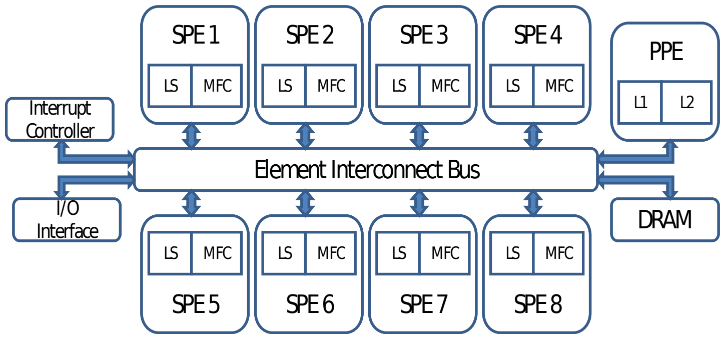

Fig 4.3 STI Cell processor architecture. [59]

Core Architecture	Intel Core2 (Xeon E5345)	Sun UltraSPARC T2 Plus	STI Cell			
			Sony PS3		IBM QS 20	
			PPE	SPE	PPE	SPE
Type	Superscalar out-of-order	Superscalar in-order	Multi Thread dual issue	SIMD dual issue	Multi Thread dual issue	SIMD dual issue
Clock (GHz)	2.33	1.16	3.2	3.2	3.2	3.2
Local store	-	-	-	256 KB		256 KB
L1 Data Cache per core	32 KB	8 KB	32 KB	-	32 KB	-
L2 Cache per core	-	-	512 KB	-	512 KB	
# Sockets	2	2	1		2	
Cores per Socket	4	8	1	8	1	8
DRAM Capacity	16 GB	64 GB	2 GB		1 GB	
Threading	Pthreads	Pthreads	Pthreads		Pthreads	
Compiler	gcc	cc	gcc	spu-gcc	gcc	spu-gcc

Table 4.1 Comparison of Multi-core architectures.

This simplified hardware design means that several software level optimizations are necessary to achieve high performance on the SPUs (these are generally not needed on traditional processors, such as the Intel Xeon). The optimizations include use of vectorization, reducing the frequency of branch instructions through loop unrolling and function in-lining, and explicit memory optimizations. Instead of a processor controlled data cache, each SPU contains a programmer controlled local store to explicitly optimize memory operations. This enables several memory level optimizations not possible on most high performance processors. Since high compute-to-I/O ratios are needed to achieve the full potential of the Cell processor [5], the programmer controlled memory stores are especially important.

4.3 Implementation

Four hardware platforms were utilized in this study, one was the Intel Xeon based, two were the STI Cell based, and one was the Sun UltraSPARC T2 Plus based. The Intel Xeon platform utilized was a blade on the Palmetto Cluster at Clemson University. Each blade on the system contained two quad core Intel Xeon processors running at 2.33 GHz (model E5345), had 12 GB of DRAM, and ran the Cent OS 5 operating system. The STI Cell platforms utilized was a Sony Playstation 3 at Clemson University and an IBM QS20 cluster at Georgia Tech. The Playstation 3 has one Cell processor on which six of the eight SPUs are available for use and contains 256 MB of DRAM. This platform was running Fedora Core 6 with IBM Cell SDK 2.1. The QS20 blade utilized had two Cell processors, each with all eight cores available, 2 GB of DRAM, and also used IBM Cell SDK 2.1. The Sun UltraSPRAC T2 Plus platform utilized was a Sun SPARC Enterprise T5140 running Solaris 10. This system contained 2 Sun UltraSPARC T2 Plus processors and 64 GB of DRAM. All the programs were compiled with -O3 optimizations using *gcc*. On the UltraSPARC platform, one processor was used for running the operating system, while the other was used to run the hierarchical Bayesian models, with each thread of the model bound to a specific core to ensure optimum performance.

Five networks with varying input image sizes were developed to examine the acceleration of the HTM model on the multi-core platforms. The overall network structure was kept similar to the design in [18], with three layers of nodes per network and each level 2 node having four level 1 children. The level 1 and 2 nodes were arranged in a square grid. Table 4.2 lists details about each of the networks examined including the number of nodes implemented in each network and the input image size. The smallest network was identical to the example presented by George and Hawkins. In order to train the different sized networks, the training algorithm described in [18] was used to generate the internal P_{xu} matrices for the networks. A subset of 76 of the 91 binary image categories presented in [18] was utilized for the training of these networks since these were used in the training example provided by the authors of the model. The set of images chosen would affect the runtimes on all the processors similarly. All the nodes in each layer are processed in parallel. The model was optimized separately for the different architectures. A set of nodes to be implemented was assigned to each thread (an SPU in the case of the cell processor), and these set of nodes were implemented in serial by each thread. The set of nodes to be assigned to each thread was pre-assigned to optimize the load on each thread.

Table 4.2. HTM configurations evaluated

Network input size	**32x32**	**48x48**	**64x64**	**80x80**	**96x96**
Total Nodes	81	181	321	501	721
Layer 3 nodes	1	1	1	1	1
Layer 2 nodes	16	36	64	100	144
Layer 1 nodes	64	144	256	400	576

Four networks with varying input image sizes were developed to examine the acceleration of the Dean model on the multi-core platforms. As shown in Table 4.3, all the networks had three layers. The smallest network was identical to the example presented by Dean in [8]. Dean utilized 10,000 images from the MNIST database [33] by Yann LeCun for training and testing of his model. These consist of one thousand versions of 10 objects (handwritten numerals 0 to 9 from the MNIST database), resulting in 10,000 images. The images are 28×28 pixels in dimension and with 8 bit resolution. The smallest network was trained with the 28×28 images in the database, while the larger networks were trained with zero padded versions of these images.

Table 4.3 Dean model configurations evaluated

Network input size		**28×28**	**36×36**	**40×40**	**52×52**
Nodes	Total	59	98	110	186
	Layer 3	1	1	1	1
	Layer 2	9	16	9	16
	Layer 1	49	81	100	169
Subnets	Total	6	11	6	11
	Layer 3	1	1	1	1
	Layer 2	1	1	1	1
	Layer 1	4	9	4	9

Dean's implementation of the model was in Matlab and utilized Kevin Murphy's Bayesian Network Toolbox (also written in Matlab) [37]. A C implementation of the model along with relevant parts of the Bayesian Network Toolbox was developed. Although C++ Bayesian Network libraries are available, they would need significant modifications in order to be utilized in our study. These include, parallelizing to run on multiple cores, vectorization using Cell SPU SIMD intrinsics, and being able to handle the DMA data transfers needed for the explicit memory management of the SPU local stores. The model and the relevant Bayesian Network libraries were optimized separately for the different architectures. In the Cell version, the PPU assigned a set of subnets or cliques to be processed to each SPU.

4.4 Parallelization and optimization

4.4.1 Hierarchical Temporal Memory model

Network parallelization

All the nodes in a particular layer are independent of each other and can therefore be evaluated in parallel. Therefore in this study, the HTM network was parallelized by assigning groups of nodes in a particular layer to separate processing cores. Nearly all computations in equations 1 through 6 are element-by-element matrix multiplies and divides (thus there are no addition operations needed). In order to accelerate the computations, the matrix values were converted into logarithmic form so that more expensive multiplies and divides could be replaced by less time consuming additions and subtractions. The comparisons involved in equations 3 and 4 could still be performed in logarithmic form and were thus unaffected by this change.

P_{xu} matrix compression and model vectorization

The P_{xu} matrix in equation 2 is large enough that it needs special consideration when examining the vectorization of the nodes. These matrices themselves are extremely sparse, being made up almost 90% zeros. The computations in equations 1 through 6 are element-by-element rather than dot products. Compressing the P_{xu} matrices can significantly speed up the algorithm computation by skipping over strings of zeros. Thus any vectorization approach needs to consider the compression of the P_{xu} matrix. Two possible approaches to utilize vectorization for the George Hawkins model were examined. The first involves vectorizing the code to process a single image more efficiently. The second approach involves vectorizing the code to process multiple images simultaneously.

Single image vectorization: In this case, equations 1 through 6 need to be vectorized for a single image. Equations 1, 5, and 6 can be vectorized easily if the variables for the equations are padded to be multiples of the vector width. Equation 2, however, cannot be vectorized as easily, given that the P_{xu} matrix is sparse. This study examined the feasibility of block compression [34] of the P_{xu} matrix to vectorize the computations in equation 2. In order to be efficient, there should be on high density of non zero elements in uncompressed blocks.

Two possible approaches for block compression are to compress along the rows or along the columns of the target matrix. Tables 4.4 and 4.5 show the density of non zero blocks for both row and column wise compression with block sizes of 4 and 8 respectively. Several network sizes are examined. The results indicate that with a vectorization factor of four, the average P_{xu} uncompressed block contains less than two non-zero elements per block, while a vectorization factor of eight yields at most 2 elements per block on average. Thus vectorizing the equations for single images is not very efficient.

Table 4.4 Block compression of P_{xu} with a block size of 4.
N/w Size : Network Size NZB: Non Zero Blocks; NZE: Non Zero Elements;

	Compression along rows			Compression along columns		
N/w	%NZB	Avg NZE in	%NZB>2	%NZB	Avg NZE in	%NZB>2 NZE

Size		NZB	NZE		NZB	
81	*3.84*	*1.2531*	*20.06*	*3.97*	*1.2605*	*17.72*
181	*5.17*	*1.3035*	*22.82*	*5.02*	*1.3891*	*24.89*
321	*6.23*	*1.3935*	*27.47*	*5.96*	*1.4940*	*28.79*
501	*7.41*	*1.4624*	*30.53*	*6.72*	*1.6483*	*34.48*
721	*7.82*	*1.4848*	*31.4*	*7.12*	*1.6659*	*35.02*

Table 4.5 Block compression of P_{xu} with a block size of 8.
N/w Size : Network Size NZB: Non Zero Blocks; NZE: Non Zero Elements;

	Compression along rows			Compression along columns		
N/w Size	%NZB	Avg NZE in NZB	%NZB>2 NZE	%NZB	Avg NZE in NZB	%NZB>2 NZE
81	*6.44*	*1.4587*	*27.89*	*6.97*	*1.4374*	*22.74*
181	*8.32*	*1.5837*	*32.66*	*8.38*	*1.6506*	*29.77*
321	*9.75*	*1.7418*	*36.69*	*9.73*	*1.8270*	*33.35*
501	*11.31*	*1.8754*	*39.91*	*10.75*	*2.0571*	*38.98*
721	*11.88*	*1.9125*	*40.38*	*11.36*	*2.0851*	*39.45*

Multiple image vectorization: The computations for any input image are identical throughout the network because each node in the network processes any input given in exactly the same manner. Therefore multiple images can also be evaluated in parallel using vectorization. In this case any compression scheme can be adopted for the P_{xu} matrices. The matrix is compressed by providing a coordinate for each nonzero value in the P_{xu} matrix. Two approaches for dealing with this are to treat the P_{xu} matrix as a linear vector (see Figure 4.4(b)) or to treat it as a two dimensional matrix (see Figure 4.4(c)). In the former case, only one coordinate is needed per nonzero element, while in the latter case, two coordinate values are needed. The first approach results in a higher compression level and thus lower data transfer time. It however does require the generation of a two dimensional (x,y) coordinate for each linear coordinate (for equation 2). Our studies indicate that a two dimensional representation provides the lowest overall execution time. For example, for the 721 node HTM model examined, the single dimensional approach required 18.26 ms on a Playstation 3, while the two dimensional approach required 10.96 ms.

1	**0**	**2**	**0**	**0**
0	**4**	**0**	**0**	**0**
0	**6**	**0**	**0**	**0**
1	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**9**

(a)

1	*0*	**2**	*2*	**4**
6	**6**	*11*	**1**	*15*
9	*24*			

(b)

1	*0*	*0*	**2**	*0*
2	**4**	*1*	*1*	**6**
1	*2*	**1**	*3*	*0*
9	*4*	*4*		

(c)

Figure 4.4 Restructuring the P_{xu} matrix. (a) Original P_{xu} Matrix. (b) Single dimensional position representation, [**p :value**, *x* :coordinate]. (c) Two dimensional position representation, [**p :value**, *x*,*y* : coordinates]

4.4.2 Dean model

Network parallelization

As shown in previous chapter, the nodes in a network in the Dean model can be grouped into subnets and the network would be processed by evaluating subnets rather than individual nodes. Also, as shown in the same section, each subnet was evaluated by processing its junction-tree representation. Each node of the junction tree is called a clique and has a clique potential associated with it. This potential is derived by combining the conditional probability tables of each node in the subnet that forms the clique (these tables are multi-dimensional with a maximum of five dimensions in our study).

There are two possible approaches to parallelize the evaluation of the Dean model: the first is at the subnet granularity and the second is at the clique granularity. This latter approach will yield a higher level of parallelism as there are more cliques than subnet (given that a subnet can be decomposed into multiple cliques). Dependencies between the cliques may limit the number of cliques that can be evaluated in parallel at any given level within a junction tree. This study evaluated both approaches and found that for the networks examined, the clique based approach had a better utilization of the available processing cores. In both approaches the order in which the subnets or cliques will be evaluated is predetermined and does not vary with the network inputs.

Vectorization

As with the HTM model, there are at least two approaches to vectorization for this model: vectorizing the operations for a single image and vectorizing to evaluate multiple images simultaneously. In the former case, matrix operations would have to be vectorized as a large portion of the junction tree evaluations consist of multi-dimensional matrix operations. In the networks examined, these matrices had up to five dimensions with each dimension being up to 16 elements wide. The matrix operations included element-by-element matrix multiplies and divides. There were also matrix dimension reductions which essentially were summations along a given dimension of the matrix. Not all of these operations can be vectorized efficiently, particularly as the matrix dimensions were of small widths (that were not always multiples of the vectorization factor).

Since the model evaluates any input data in precisely the same way, multiple inputs can be evaluated in parallel through vectorization. In case of a vectorization factor of four, there will be four versions of each matrix (one for each image). The same set of operations will be carried out for all four versions of each matrix. In this case vectorization can be applied to almost 100% of all the operations.

CHAPTER FIVE

RESULTS

5.1 Study of Accuracy.

The seven different algorithms were successfully implemented and tested with the MNIST Database [36]. The MNIST database contains handwritten digits with 6000 28x28 images per class for training and around 10000 test images.

	NB	**KNN**	**NCA**	**SVM**	**ANN**	**HTM**	**Dean**
MNIST	12.8	5.2	3.2	1.8	12.5	26	19
MNIST(6000 training examples)	18	9.00	8.5	8.1	16	30	24

Table 5.1 Comparison of Accuracy.

The naïve Bayes classifier was trained with the 784 sized features. Different values of thresholding were compared and it was found to be optimal at 185. The test set error of 12.8% was received. The advantages are ease of implementation and very fast test time, but the disadvantage is that the strong independence assumption makes the algorithm weak. The KNN algorithm was also trained with the feature size of 784 and the change in accuracy of the KNN classifier improved with the increase in K is shown in figure 5.1.The advantages of this method is that, it requires no training and implementation is very simple. The disadvantages are that, the test time is in the order of hours for MNIST data set (compared to seconds for all other algorithms) and the need to keep the complete database during testing.

The SVM algorithm was implemented with the sequential minimal optimization algorithm[61], with Gaussian kernel. SVM is a binary classifier, so multi-class classification is achieved by one-against all classification [50].

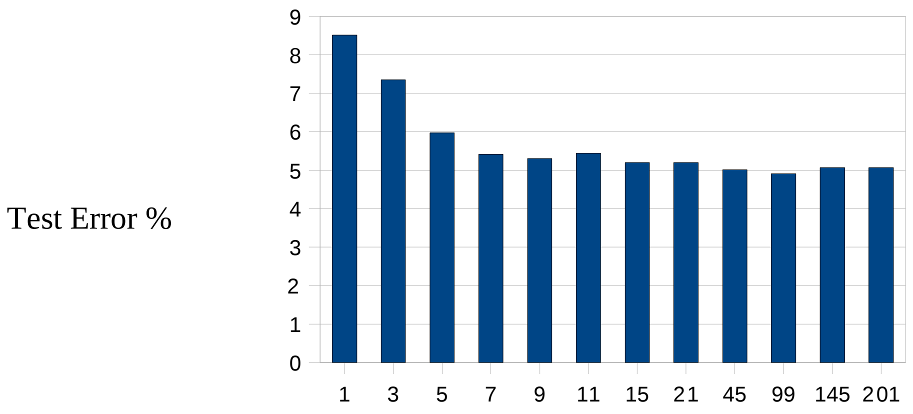

Fig 5.1 Test Error versus K, KNN Algorithm.

The SVM algorithm does not require the complete database to be stored. The runtime for testing is fast and has very good accuracy but takes long to converge while training. The multilayer perceptron algorithm implemented also does not require the complete database for testing and runtime for testing is fast. It was found that a network architecture with two hidden layers of 800 and 200 neurons, gave the most optimal performance in accordance with LeCun et al. [31]. HTM and Dean's model performed comparatively poorly, with a testing error of 26% and 19% respectively. But this is a respectable figure given the fact that, both these algorithms are in their nascent stages and have a large scope for improvement. The advantages of HTM and Dean are that, they do not require to have complete database for testing, fast test time execution, and are comparatively more biologically plausible [11][12][19]. It was found that, the SVM classifier performed the best and the test error achieved was comparable to that achieved by LeCun et al. [31]. The other interesting thing to note is that algorithms behave differently to training set size. Simpler algorithms like NB, KNN and NCA depends on the training set and accuracy improves with the training set size. This has been noted by Holte [59], that simple classifier do a very good job on large databases.

5.2 Study of Multi-core Performance.

Both the models were tested on the following four platforms with the given configurations.

1. Intel blade with 4 and 8 threads.
2. Playstation 3 with 6 SPU threads.
3. QS20 with 6, 8 and 16 SPU threads.
4. Sun UltraSPARC T2 Plus with 8, 16, 32, and 64 threads.

The six SPU thread implementation on the QS20 was examined inorder to compare the it against the Playstation 3 (PS3) performance. A serial version of the program was developed and tested on the Sony Playstation 3’s Cell PPU. All the implementations (both serial and all parallel) utilized the data structure optimizations for the models listed in earlier sections (such as P_{xu} matrix compression in the HTM model).

5.2.1 Speedup

Figures 5.2 and 5.3 present the speedup of each of the parallel implementations over the serial PPU implementation. From these figures it is seen that the parallel implementations of the models provide a significant performance gain over their serial implementations. This is mainly due to the use of multiple cores and the use of vectorization on the Intel and Cell architectures. There is sufficient parallelism in the models examined, so that for all of the platforms, use of more cores provided higher speedups. Our experiments showed that increasing the number of threads on the Intel Xeon blade beyond 8 provided no further improvement in performance. The Dean model produced a higher speedup than the HTM model for all the platforms examined. It is possible that the larger number of training categories in the HTM model produces larger potential tables, which translates to more data transfers, thus limiting its speedup over the Dean model.

For both models, it is seen that the Cell processor outperformed both the Intel Xeon and the Sun UltraSPARC T2 Plus processors. The Playstation 3 with 6 available SPU cores outperforms the Intel Xeon processor (with 4 cores) by about 1.9 times for the HTM model and by 2.4 times for the Dean model. As a result the Playstation 3 also outperformed the blade with two Intel Xeon processors. The speedup of the Cell processor on the QS20 with all 8 SPU cores available over a single Intel Xeon processor was about 2.3 times for the HTM model and about 3 times for the Dean model. Utilizing both Cell processors on the QS20 (16 threads) provides only a 11% performance gain for HTM model and a 22% performance gain for the Dean model over one Cell processor (8 threads). This is believed to be due to the memory accesses becoming a bottleneck as calculation times become close to data access times (as shown in Table 5.2). This effect is not seen on the Sun processor when going from 8 to 16 threads as the calculations take much longer on that system.

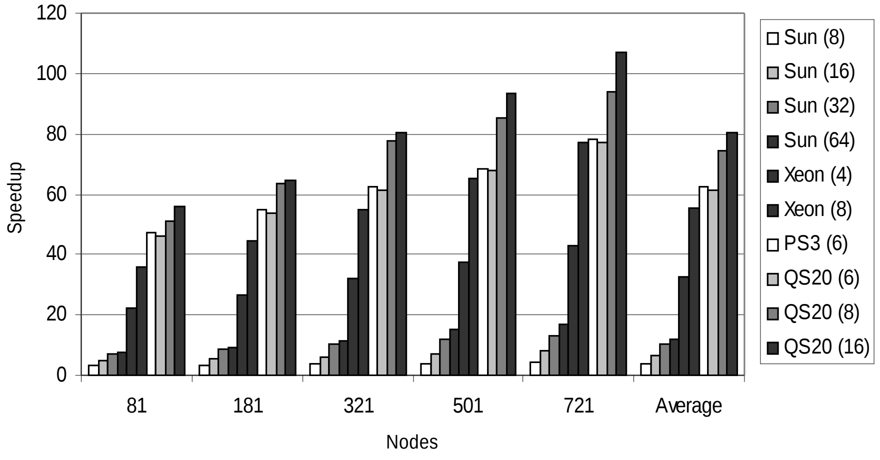

Figure 5.2 Speedup for the HTM model on different multi-core architectures over the Cell PPU. The numbers in parenthesis in the legend represent the number of threads utilized on each platform.

On the UltraSPARC processor, the Dean model provides speedups of about 2 when going from 8 to 16 threads and when going from 16 to 32 threads. The speedup from 32 to 64 threads is minor (about 1.1 times for the 186 node network). The HTM model provided lower speedups than the Dean model: 1.9 times for the largest model tested when going from 8 to 16 threads, 1.7 times when going from 16 to 32 threads, and 1.3 times when going from 32 to 64 threads. The Sun processor provides a lower speedup than the Xeon and Cell processors because of a lower clock frequency and a lack of vector capabilities. If multiple images were not available to process simultaneously (such as if there were only one small camera source), then it would not be possible to take advantage of the vectorization utilized. In this case the performance of the Intel and Cell architectures would be about one fourth of their current values.

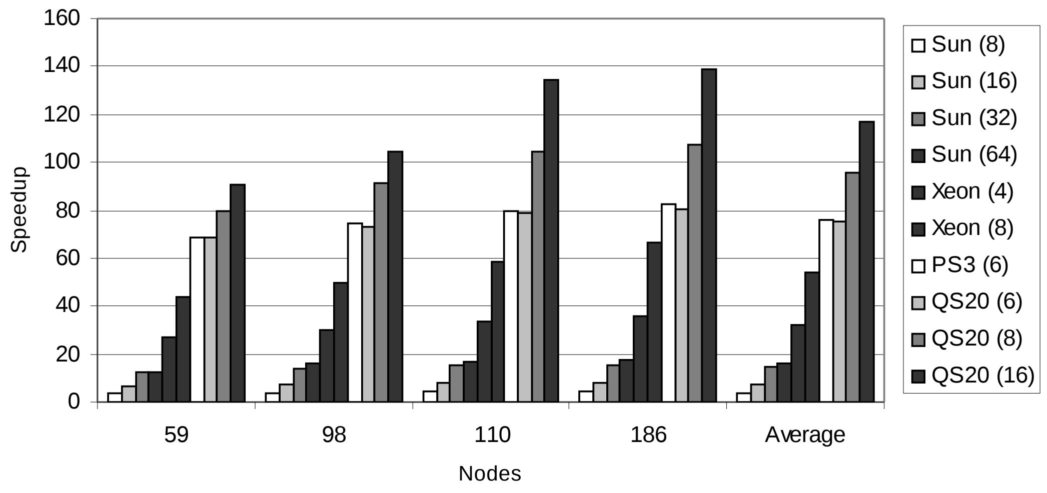

Figure 5.3. Speedup for the Dean model on different multi-core architectures over the Cell PPU. The numbers in parenthesis in the legend represent the number of threads utilized on each platform.

Since the Sun does not support vector operations, its performance would not be affected. In this situation, the Sun processor with 64 threads would actually be faster than the Xeon processor with 4 threads; about 2 times for the largest HTM model and 1.6 times for the largest Dean model.

5.2.2 Runtime breakdown of models

Figures 5.4 and 5.5 show the runtime breakdowns of the HTM and Dean models respectively on the Cell processor (on the Playstation 3) and the Intel Xeon processor (4 thread implementation). The runtime break downs are given for the smallest and the largest network sizes for both the models. This is done to compare the change in each part of the algorithm with the scaling of the model. The time for signaling between the different threads on all the platforms was insignificant due to the pre-assigning of nodes to different threads at the start of the program. Therefore this time is not listed separately in the timing breakdown.

For the Cell platform, the non-overlapped memory access time is calculated by taking the difference between the overall runtime of the application and the runtime with DMA data transfers commented out of the code. This is the part of the DMA accesses that could not be overlapped with computations (generally through double buffering). On the Intel Xeon platform, this time was calculated by taking the difference between the overall runtime and a version of the code with all global variables in the threads converted to local variables (synchronization barriers between threads were kept intact). The number of computations (array accesses and other operations) was kept the same in both cases.

The results show that on the Cell processor, DMA transfers that could not be overlapped can be a significant percentage of the overall runtime. However this fraction decreases as the network sizes increase since the nodes in the network become more complex and thus have more computations to be carried out per node. This is seen by the increase in the computation percentage for equations 2, 3, and 4 in the HTM model and in the percentage of time for getting evidence in the Dean model. Stalls due to global variable accesses on the Xeon processor (listed as non-overlapped memory access in Figures 5.4 and 5.5) showed similar trends as well.

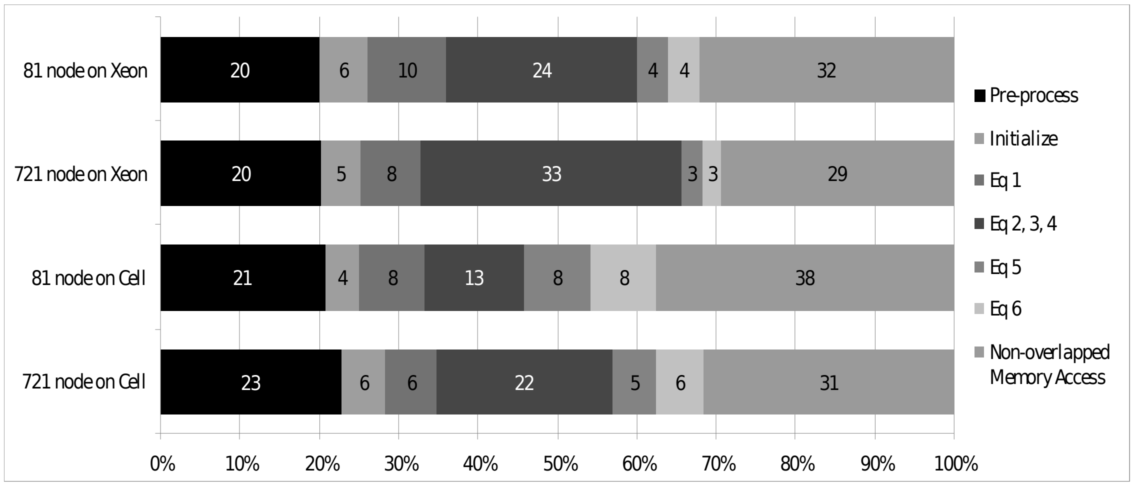

Figure 5.4 Runtime breakdowns for the HTM model on the PS3 and Xeon processors (a) Runtime breakdown for the 81 node network on the Playstation 3. (b) Runtime breakdown for the 721 node network on the Playstation 3. (c) Runtime breakdown for the 81 node network on the Xeon Processor. (d) Runtime breakdown for the 721 node network on the Xeon Processor.

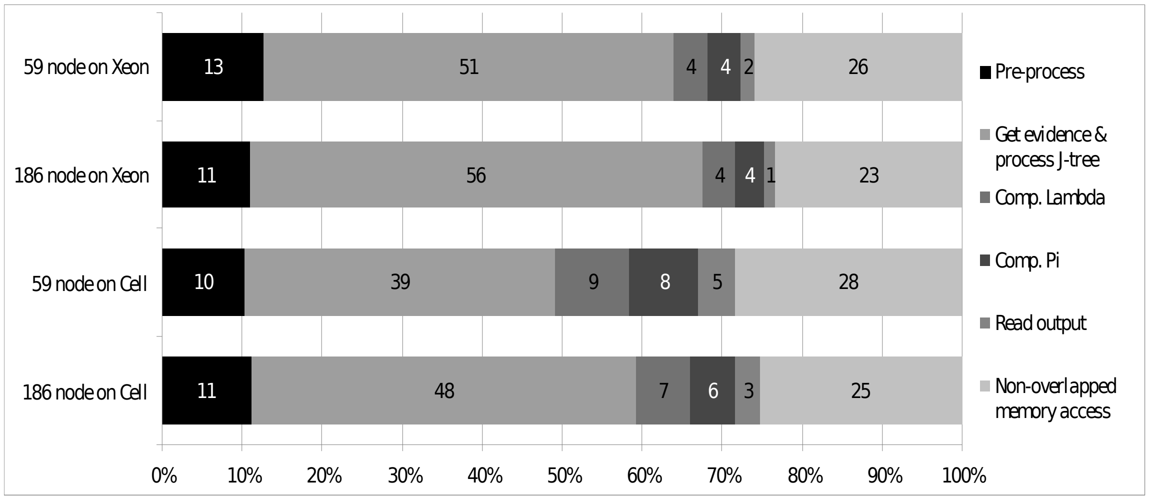

Figure 5.5 Runtime breakdowns for the Dean model on the PS3 and Xeon processors (a) Runtime breakdown for the 59 node network on the Playstation 3. (b) Runtime breakdown for the 186 node network on the Playstation 3. (c) Runtime breakdown for the 59 node network on the Xeon Processor. (d) Runtime breakdown for the 186 node network on the Xeon Processor.

The overall DMA time is unlikely to change with the number of cores used on the Cell processor as these accesses go to a centralized memory system. However the overall computation time is likely to decrease with more cores due to increased parallelism. Hence, as the number of cores increase, the DMA time can exceed the computation time, thus limiting the speedup seen with increasing cores. This effect is seen in figures 5.2 and 5.3: the speedup does not double when going from 8 to 16 cores. Although this could be due to the impact of off-chip memory buses, the results in Table 5.2 seem to indicate that it is due to a memory bottleneck. Table 5.2 shows the runtime breakdown of the largest HTM and Dean networks on the Cell processor platforms examined: Playstation 3 with 6 SPU, and QS20 with both 8 and 16 SPUs. While the computation time decreased with increasing numbers of cores, the non-overlapped DMA time increased slightly (since the computations started taking less time than data transfers). To alleviate this issue, a higher memory bandwidth would be needed. This would be seen by having each cell processor have access to its own dedicated memory. Figure 5.6 compares the two parallelization approaches examined for the Dean model: clique based and subnet based. All the subnets in a layer can be evaluated in parallel.

Table 5.2 Run time break down of the largest HTM and Dean models on the QS20 with 6, 8, and 16 threads. All times are in ms.

	HTM			Dean		
SPUs	6	8	16	6	8	16
Computation only (ms)	7.51	5.45	3.50	12.10	8.10	4.50
Non-overlapped DMA (ms)	3.45	3.60	4.45	4.10	4.34	5.12
Total (ms)	10.96	9.05	7.95	16.20	12.44	9.62
% of DMA in runtime	31.47	39.77	55.97	25.30	34.88	53.22

5.2.3 Parallelization strategy for the Dean model

The networks with 59 and 110 nodes had fewer subnets in level 1 than the 98 and 186 node networks. Thus the former set of networks provided lower speedups than the latter set when parallelized by subnets. For all the networks, there were more cliques that could be evaluated in parallel than subnets (since each subnet could be decomposed into multiple cliques). Thus the clique based parallelization approach provided higher speedups for all the network sizes evaluated.

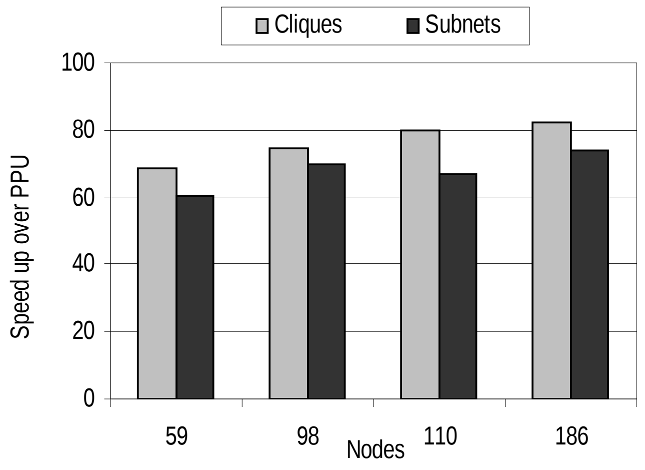

Figure 5.6 Parallelization of the Dean model by cliques vs. subnets. The 59 and 110 node networks are using only 4 SPUs because of the limited set of subnets on those networks. The other two are using six SPUs.

CHAPTER SIX

CONCLUSIONS AND FUTURE WORK

There is a significant interest in the research community to develop machine learning algorithms with good inference capabilities and with the ability to do real time tasks. Seven leading algorithms were compared for the task of handwritten character recognition. The study shows that Hierarchical Bayesian cortical models, which are a relatively new class of models, have lower performance compared to the leading algorithms on the complete MNIST database. The performance was compared for a smaller subset of MNIST dataset, and it was found that the accuracy of the Bayesian algorithms does not decrease considerably with the change in training set size. The Hierarchical Bayesian cortical models have inherent parallelism that make it easier to develop larger scale simulations of the cortex than traditional neural networks.

Since Hierarchical Bayesian cortical models are based on cortical columns as opposed to individual neurons, they have a significant computational advantage over the latter. Fewer nodes need to be modeled along with fewer node connections. Given that large scale cortical models can offer strong information processing capabilities, hierarchical Bayesian models are an attractive candidate for scaling. In the second part of this study, the parallelization and implementation on multi-core architectures of two hierarchical Bayesian models: the Hierarchical Temporal Memory model and the Dean's Hierarchical Bayesian model were done. Three multi-core processors were examined for implementation: the IBM Cell processor, the Intel Xeon processor, and the Sun

UltraSPARC T2 Plus. Both the models and their relevant libraries were implemented in C, parallelized, and vectorized. This is the first study of the acceleration of this class of models on multi-core architectures. It was shown that the hierarchical Bayesian cortical models can be parallelized onto multi-core architectures to provide significant speedups over serial implementations of the models. The speedups come primarily from the use of multiple processing cores and vector operations. The speedups increase as the models are scaled and as the number of processing cores is increased. The highest performance gain was seen from the Cell processor, with speedups of 1.9 times for the Dean model and 2.4 times for the HTM model over a parallel implementation on the Xeon Processor. It was shown that the Dean model can be parallelized based on the subnets that it contains, or based on the cliques contained in the junction-trees that the subnets can be converted into. Our results indicate that the latter approach provides slightly higher speedups as there is more parallelism exposed. This study also examined the vectorization of the two models and showed that it is easier to vectorize by processing multiple inputs simultaneously.

The results of this work can be applied to other multi-core processors. As future work, a study can examine:

1) Comparison of Convolutional Neural Networks[32] and LeCun Convolutional Network [31] for handwritten character recognition.
2) Comparison of all the above algorithms for more difficult recognition tasks like face recognition using standard databases.

3) The performance of much larger networks on clusters of multi-core processors.
4) The parallelization of the training phase of these models.
5) The parallelization of the newer version of these models [9][10][15] which incorporates temporal invariance in addition to spatial invariance.

Moreover these parallelized multi-core implementations of the models could be used for real time tasks and to make improvements to the algorithms itself.

REFERENCES

[1] R. Ananthanarayan, D. Modha, “Anatomy of a cortical simulator”, Proceedings of ACM/IEEE conference on Supercomputing, 2007.

[2] J.A. Anderson, “The BSB Network,” In: Hassoun, M.H. (ed.) *Associative Neural Networks*. Oxford University Press, New York, 77-103, 1993.

[3] J. A. Anderson, “Arithmetic on a parallel computer: Perception versus logic,” *Brain and Mind*, 4, 169–188, 2003.

[4] J. A. Anderson, P. Allopenna, G. S. Guralnik, D. Scheinberg, J. A. Santini, S. Dimitriadis, B. B. Machta, and B. T. Merritt, “Programming a Parallel Computer: The Ersatz Brain Project,” In W. Duch, J. Mandziuk, and J.M. Zurada (Eds.), *Challenges to Computational Intelligence*, Springer: Berlin, 2006.

[5] D.A. Bader, V. Agarwal, K. Madduri, and S. Kang, “High Performance Combinatorial Algorithm Design on the Cell Broadband Engine Processor,” *Parallel Computing*, *Elsevier,* 33(10-11):720-740, 2007.

[6] Barto, A.G., Sutton, R.S., & Anderson, C. (1983). Neuron-like adaptive elements that can solve difficult learning control problems, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13*: 834-846.

[7] A. Buttari, J. Dongarra, and J. Kurzak, “Limitations of the Playstation 3 for High Performance Cluster Computing,” University of Tennessee Computer Science Technical Report, CS-07-594, May 2007.

[8] Carpenter, G. and Grossberg, S. (1995). *Adaptive resonance theory* (ART). In Arbib, M., (Ed.), Handbook of Brain Theory and Neural Networks, pages 79--82. MIT Press.

[9] Rich Caruana and Alexandru Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms," The Proceedings of the 23rd International Conference on Machine Learning (ICML2006), June 2006, pp. 161-168.

[10] Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20:1-25.

[11] T. Dean, "A Computational Model of the Cerebral Cortex," *Proceedings of the Twentieth National Conference on Artificial Intelligence* (AAAI-05): 938-943, 2005.

[12] T. Dean. "Scalable inference in hierarchical generative models," Ninth International Symposium on Artificial Intelligence and Mathematics, 2006.

[13] T. Dean, "Learning invariant features using inertial priors," *Annals of Mathematics and Artificial Intelligence*, 47(3-4), 223–250, Aug. 2006.

[14] T. Dean, G. Carroll, and R. Washington, "On the prospects for building a working model of the visual cortex," *Proceedings the Twenty-Second Conference on Artificial Intelligence* (AAAI-07): 1597-1600, 2007.

[15] A. Delorme and S. J. Thorpe, "SpikeNET: an event-driven simulation package for modelling large networks of spiking neurons," *Network-computation in neural systems*, 14(4), 613–627, Nov. 2003.

[16] M. Djurfeldt, M. Lundqvist, C. Johansson, M. Rehn, O. Ekeberg, and A. Lansner, "Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer," *IBM Journal of Research and Development*, 52(1-2), 31–41, Jan.-Mar. 2008.

[17] A. Felch, J. Moorkanikara-Nageswaran, A. Chandrashekar, J. Furlong, N. Dutt, A. Nicolau, A. Veidenbaum, R. H. Granger. "Accelerating Brain Circuit Simulations of Object Recognition with a Sony PlayStation 3," *Proceedings of the International Workshop on Innovative Architectures*, 2007.

[18] D. George and J. Hawkins. "A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex," International Joint Conference on Neural Networks (IJCNN 2005), 2005.

[19] D. George, "How the brain might work: A hierarchical and temporal model for learning and recognition," Doctoral Dissertation, Dept. of Electrical Engineering, Stanford University, 2008.

[20] M. 16, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki, "Synergistic Processing in Cell's multi-core Architecture," *IEEE Micro*, 26(2), 10–24, Mar. 2006.

[21] J. Hawkins and S. Blakeslee, “On Intelligence,” Times *Books, Henry Holt and Company,* New York, NY 10011, Sept. 2004.

[22] J. Hawkins, D. George, “Hierarchical Temporal Memory: Concepts, Theory and Terminology”, Numenta, http://www.numenta.com/Numenta_HTM_Concepts.pdf , 2006.

[23] Hinton, G. E. and Sejnowski, T. J. (1983) Optimal perceptual inference. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, Washington DC.

[24] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and application to conduction and excitation in nerve,” *Journal of Physiology*, 117, 500–544, 1952.

[25] Hopfield, J. J. (1982). Neural networks and physical systems with emergent computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554.

[26] E. Izhikevich and G. Edelman, “Large-Scale Model of Mammalian Thalamocortical Systems,” *Proceedings of the National Academy of Sciences,* 105(9), 3593–3598, Mar. 2008.

[27] C. Johansson, A. Lansner, “Towards Cortex Sized Artificial Neural Systems,” *Neural Networks*, 20(1): 48-61, 2007.

[28] R. D. King, C. Feng, and A. Sutherland. STATLOG: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):289-334, 1995.

[29] Kohonen, T. An adaptive associative memory principle. *IEEE Transactions on Computers*, C-23:444-445, 1974.

[30] S. Lauritzen, D. Spiegelhalter, “Local computations with probabilities on graphical structures and their application to expert systems,” *Journal of the Royal Statistical Society*, 50(2):157–194, 1988.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278-2324, November 1998.

[32] Y. Le Cun and Yoshua Bengio. Convolutional networks for images, speech, and time series. In Michael A. Arbib, editor, The Handbook of Brain Theory and Neural Networks, pages 255–258. MIT Press, Cambridge, Massachusetts, 1995.

[33] Y. LeCun and C. Cortes, “The MNIST Database of handwritten images,” http://yann.lecun.com/exdb/mnist/.

[34] T. S. Lee and D. Mumford, “Hierarchical bayesian inference in the visual cortex,” *Journal of the Optical Society of America A*, 2(7), 1434–1448, 2003.

[35] H. Markram, “The Blue Brain Project,” *Nature Reviews Neuroscience*, 7, 153–160, 2006.

[36] W. Maas, “Networks of spiking neurons: the third generation of neural network models,” *Transactions of the Society for Computer Simulation International*, 14(4), 1659–1671, Dec. 1997.

[37] K. Murphy, “The Bayes Net Toolbox for Matlab.” *Computer Science and Statistics* 33(2): 1024-1034, 2001.

[38] McCulloch, W. S. and Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-133.

[40] Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge, MA.

[41] V. Mountcastle, “Introduction to the special issue on computation in cortical columns,” *Cerebral Cortex*, 13(1):2–4, 2003.

[42] Neeba N.V and C.V. Jawahar, "Empirical Evaluation of Character Classification Schemes," ICAPR, pp.310-313, Seventh International Conference on Advances in Pattern Recognition, 2009.

[43] J. Pearl, “Probabilistic Reasoning in Intelligent Systems,” *Networks of Plausible Inference*, Morgan Kaufmann, San Francisco, CA, 1988.

[44] W. Rall, “Branching dendritic trees and motoneuron membrane resistivity,” *Experimental Neurology*, 1, 503–532, 1959.

[45] J. Rickman, “Roadrunner supercomputer puts research at a new scale,” Jun. 2008, http://www.lanl.gov/news/index.php/fuseaction/home.story/story_id/13602.

[46] Rosenblatt F. “The Perceptron: A probabilistic model for information storage and organization in the brain,” Psycho-logical Review 65: 386–408, 1958.

[47] P. Salin, J. Bullier, “Corticocortical connections in the visual system: structure and function”, *Physiological Reviews*, 75(1):107-54, 1995.

[48] Sun Microsystems, “UltraSPARC T2™ Supplement to the UltraSPARC Architecture 2007”, http://opensparc-t2.sunsource.net/specs/UST2-UASuppl-current-draft-HP-EXT.pdf, 2007.

[49] V. N. Vapnik and A. Ya. Chervonenkis, On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities, Theory Probab. Appl. 16, 264 (1971).

[50]. Vapnik, V. (1995). *The Nature of Statistical Learning Theory*, chapter5. Springer-Verlag, New York.

[51] S. Vassiliadis, S. Cotofana, P. Stathis “Block based compression storage expected performance”, in the proceedings of HPCS2000, 2000.

[52] Q. Wu, P. Mukre, R. Linderman, T. Renz, D. Burns, M. Moore and Qinru Qiu, “Performance Optimization for Pattern Recognition using Associative Neural Memory,” Proceedings of the 2008 IEEE International Conference on Multimedia & Expo, June 2008.

[53] Y. Xia and V. Prasanna, “Junction Tree Decomposition for Parallel Exact Inference,” IEEE International Parallel & Distributed Processing Symposium (IPDPS'08), April 2008.

[54] Y. Xia and V. Prasanna, “Parallel Exact Inference on the Cell Broadband engine processor,” *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2008.

[55] R. Zemel, “Cortical belief networks,” in Hecht-Neilsen, R., ed., *Theories of the Cerebral Cortex*, New York, NY: Springer-Verlag, 2000.

[56] S. Zhu and D. Hammerstrom, “Simulation of associative neural networks,” *Proceedings of the 9th International Conference on Neural Information Processing*, 1639- 1643, Nov. 2002.

[57] Richard Linderman, “Early experiences with algorithm optimizations on clusters of

playstation 3's," *DoD HPCMP Users Group Conference*, Jul. 2008.

[58] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in Cell's multi-core architecture," *IEEE Micro*, vol. 26, pp. 10–24, Mar. 2006.[59] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–91, 1993.

[59] Samuel Williams, Jonas Carter, Leonid Oliker, John Shalf, Katherine Yelick, "Optimization of a lattice Boltzmann computation on state-of-art multicore platforms", Journal of Parallel Distributed Computing, 2009.

[60] John C. Platt, Fast training of support vector machines using sequential minimal optimization, Advances in kernel methods: support vector learning, MIT Press, Cambridge, MA, 1999.

[61] Andrew Ng, Stanford CS 229 Machine Learning, http://www.stanford.edu/class/cs229/notes/cs229-notes3.ps

[62] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis.In L. K. Saul, Y. Weiss, and L. Bottou, editors, Advances in Neural Information Processing Systems 17, pages 513–520, Cambridge, MA, 2005. MIT Press.