Brett Duncan
CSc 8530 Parallel Algorithms
Detailed Annotated Bibliography and Classification

1. **Training on GPU**
    1. **Parallel Implementation of Neural Networks Training on Graphic Processing Unit**
    Backpropagation is used as a fine-tune technique of a deep belief network, and pre-training is not considered. In the training algorithm, sum and max are reduced on the GPU. The CPU and GPU implement different instructions in parallel, with the CPU handling logic instructions and the GPU handling complexity computing.
    2. **Parallel Training of a Back-Propagation Neural Network Using CUDA**
    The approach here is to implement a parallel back-propagation algorithm for neural networks on CUDA. Matrix and vector operations are performed on the GPU. There are two types of parallel operations: vector-matrix products and arithmetic operations. In the first type, the CUDA Basic Linear Algebra Subroutines (CUBLAS) library is used on 1 x n matrices. In the second type, a kernel is launched, and a function is applied over all the elements of a vector.
    For the parallel algorithm, CUBLAS is initialized with cublasAlloc instead of cudaMalloc. CUBLAS computes the optimum number of blocks and threads to perform its functions. The matrix-vector product is utilized to propagate the data forward to the hidden layer. A sigmoid kernel is launched, then the matrix-vector product is performed, then another sigmoid kernel is called again when the network is propagated to the output. Then, the output error is computed. Next, the error is computed for each hidden neuron using a kernel call. Next, the weights are updated using kernel calls (one of which does a matrix multiplication).
    3. **Parallel Implementation of Feedforward Neural Networks on GPUs**
    This paper presents a set of techniques for allowing efficient implementation of feedforward neural networks on GPUs. The techniques are based on the idea of executing each layer in parallel for many input cases at once, for training and network use, as opposed to mapping each neuron in a network to a single computing core. GPU kernels are responsible for calculating a single layer of the network, with all input cases in a dataset processed in parallel.
    For forward propagation, a kernel is used that takes as inputs the network weights and the inputs for each neuron, and produces as output the output levels of each neuron. The kernel must be launched in a grid with a number of blocks equal to the number of input cases, and the number of threads per block must be equal to the number of nodes in the layer. The CPU executes a copy of the kernel for each thread. The output array for a layer is used as the input for the next level.
    For backward propagation, two kernels implement the delta calculation: one for computing deltas for the output layer, and the other for the hidden layers. The kernel to calculate error derivatives is launched with a grid composed of a number of blocks equal to the number of input cases, and a number of threads per block equal to the number of weights in the whole network. For the weight update, the kernel uses the array of overall derivatives resulting from the reduction (the reduction is the derivatives for all input cases calculated as the sum of derivatives of each case) and the network weights.
    4. **GPU Asynchronous Stochastic Gradient Descent to Speed Up Neural Network Training**
    In this paper, two types of parallelism are used: model parallelism and data parallelism. Using a system called GPU A-SGD, the training of large convolutional neural networks for

computer vision is sped up.
In model parallelism, stochastic gradient descent (SGD) can be used when the dataset is too large. In SGD, the gradient of the objective function is calculated over a small random partition of the dataset called a minibatch. The structure of the neural network computations is exploited to speed up the calculation of the minibatch gradient.
In data parallelism, the stochastic gradient descent steps are parallelized by using distributed versions of stochastic gradient descent.
For the GPU A-SGD, the GPUs are used for model parallelism, and A-SGD for data parallelism.

5. **Stochastic Data Sweeping for Fast DNN Training**
   In this paper, a novel framework is proposed to speed up BP training, known as stochatic data sweeping (SDS). In this framework, the training data is selected stochastically from the whole set and the quantity is reduced at each training epoch. The SDS approach is integrated into multi-GPU parallelization approaches for a much faster BP training framework. Using a fixed amount of random selected subsets throughout the entire BP training process will not yield good trade-off between saving training time and keeping recognition accuracy, so the dynamic subset selection using the data sweeping function is used in this paper.
6. **Asynchronous Stochastic Gradient Descent for DNN Training**
   In this paper, an effective approach to speed up DNN training for speech recognition is proposed. The real parallelization of BP is prohibitive due to the sequential property of stochastic gradient descent (SGD), so asynchronous stochastic gradient descent is used to address this issue, by approximating BP. Multiple GPUs are used that work asynchronously. Each GPU calculates gradients and update the global model parameters independently. This is done by the CPU initializing the model and storing it in the server, then the model training on multiple-GPU starts. The GPU calculates the gradient based on minibatch. The model is updated by the CPU.
7. **Pipelined Back-Propagation for Context-Dependent Deep Neural Networks**
   This paper takes a step towards parallelizing the back-propagation algorithm for deep neural network training. Pipelined back-propagation is used, which updates models with delayed data and allows network layers to be computed concurrently. Multiple GPGPU cards are utilized in a single server.
   First, the data can be partitioned using map-reduce. For each minibatch, it requires accumulation/redistribution of gradients/models of the dimension of the entire model to and from a master to the other GPUs. Next, each layer's model parameters are partitioned into stripes and parallelized across these. Each GPU holds a vertical stripe of each layer's parameters and gradients (also known as node parallelism).
8. **Parallel Neural Network Training with OpenCL**
   OpenCL is used to implement two parallel neural network training algorithms: parallel backpropagation, and parallel particle swarm optimization (PSO). Programming elements in OpenCL are based on work items or kernels, and are processed in parallel by a compute unit called a workgroup. Maximum efficiency is achieved on a GPU, but it could also be executed on a general purpose CPU. Training is more efficient with larger networks.
   To parallelize the neural network training, the network output is computed layer by layer at a time. Each layer has a set of work items, where each work item computes the output of a single neuron.

2. **<u>Map Reduce/Hadoop</u>**
   1. **The Improved BP Algorithm Based on MapReduce and Genetic Algorithm**

This paper combines MapReduce and a genetic algorithm to improve upon previous algorithms based on MapReduce. In in the MapReduce BP algorithm, **batch** training is used, in which weights are updated after all records in training have been processed. In the genetic algorithm, all the weights of the BP neural network are encoded as one chromosome. The reduce function is where the selection, crossover, and mutation of the genetic algorithm occur.

2. **Parallel Implementation of Multilayered Neural Networks Based on Map-Reduce on Cloud Computing Clusters**
   This paper presents an efficient mapping scheme for a fully connected multilayered neural network trained using back-propagation based on Map-Reduce of cloud computing clusters. Batch-training (or epoch-training) is used.
   For the BP algorithm based on Map-Reduce, each computing node has a complete ANNs and initial state of the network is consistent. The BP algorithm is divided into two parts on Hadoop. In the Map phased, samples are split to train the network in the batch-pattern, and get output weights after a certain number of iterations. In the Reduce phase, all the outputs of each Map phase are colligated to obtain new weights.
3. **A Method for Text Categorization Using BP Network Based on Hadoop**
   In this paper, a BP network text categorization model based on data parallel method on the Hadoop platform that uses the MapReduce programming model is designed to overcome the high time costs when using large amounts of texts to train the BP network. It uses batch training and adjusts the network weights after getting the accumulated error by summing every sample training error on each node. The categorization of text is done in parallel. First, preprocessing is done to reduce the feature dimension using x^2-statistic. Then text vectorization must be performed by by calculating the weight of every feature word in each text, which will construct the vector space model of every text. Term frequency-inverse document frequency (TFIDF) is used to calculate the weight of the feature words to constuct the vector space model of every text.
   After text vectorization, the text vectors are used to train the BP network for text categorization. The training of the BP network is done by utilizing data-parallel and batch training on a Hadoop cluster. In data-parallel, every node has a whole BP network. In batch training, network weights are adjusted after calculating the accumulated error of all samples. Before training the BP network using MapReduce, the initial network weights matrix is written into the global Distributed Cache. In the mapping phase, each TaskTracker reads the sample data in parallel, executes forward propagation, and calculates the errors and sends them to the Reducer. In the reduce phase, the accumulated error of all samples is calculated, and error back propagation is used to adjust the network weights.

3. **Batch Training**
   1. **Parallel Batch Pattern Training Algorithm for Deep Neural Network**
      Each layer is treated as an unsupervised Restricted Boltzmann Machine (RBM) and pre-trained one at a time. Afterwards, the standard supervised backpropagation is used to train the layers. The batch pattern training algorithm updates neurons' weights and thresholds after all training patterns are processed (an epoch), instead of after each training pattern.
   2. **Parallel Batch Pattern Training of Neural Networks on Computational Clusters**
      A batch pattern training algorithm is used instead of implementing a parallel MLP of the standard sequential training algorithm, because the parallelization using the standard sequential training algorithm has high synchronization and communication overhead. Batch pattern training updates neurons' weights and thresholds at the end of an epoch. Experiments using the ccNuma architecture showed performance improvements.

3. **Efficient Parallelization of Batch Pattern Training Algorithm on Many-core and Cluster Architectures**
   Batch training is also used in this paper, in which the weights and thresholds of each neuron is updated after all training patterns are processed (after an epoch). A recirculation neural network is used, which compresses the input pattern space of X to an output vector $\overline{X}$ containing compressed data. Open MPI, Mvapich, and Intel message passing libraries are used in the experiments and results compared.
4. **Parallel Batch Pattern BP Training Algorithm of Recurrent Neural Network**
   The parallel batch pattern BP training algorithm divides all computational work among the master and workers processors. The master assigns functions and calculations, and the workers execute the calculations.
   The master defines the number of patterns in the training data set and the number of processors used for the parallel executing of the training algorithm. Then, all patterns are divided into equal parts corresponding to the number of workers and assigns one part of the patterns to himself. After all assigned patterns are processed, the global operation of reduction and summation is executed. The sum of all delta weights and delta thresholds are sent to all processors that are working in parallel.

4. **<u>Sequence Training</u>**
   1. **Sequence Training of Multiple Deep Neural Networks for Better Performance and Faster Training Speed**
      A multiple deep neural network (mDNN) is used for acoustic modeling. Sequence training on mDNN is used, based on the maximum mutual information (MMI) criterion. Sequence training has 3 steps: i) DNN forward past, in which posterior probabilities of all HMM states are computed for all feature frames. ii) Word graph processing: performs forward-backward algorithm in each word graph. iii) DNN back propagation.
   2. **Sequence-discriminative Training of Deep Neural Networks**
      Sequence-discriminative training of a deep neural network (DNN)-hidden Markov model (HMM) hybrid is experimented with on a standard 300 hour American English conversational telephone speech task. Different sequence-discriminative critera are used: maximum mutual information (MMI), minimum phone error (MPE), state-level minimum Bayes risk (sMBR), and boosted MMI are compared. Two heuristics are investigated to improve the performance of the DNNs trained using sequence-based criteria: lattices are regenerated after the first iteration of training, and for MMI and BMMI, the frames where the numerator and denominator hypothesis are disjoint are removed from the gradient computation.
      For a DNN-HMM hybrid system, the DNN is trained to provide posterior probability estimates for the HMM states.
      In one setup, DNNs are trained using cross-entropy. In another setup, sequence-discriminative training is used.

   3. **A Comparison of Two Optimization Techniques for Sequence Discriminative Training of Deep Neural Networks**
      Two lattice-based sequence discriminative training of neural network acoustic models are compared: distributed Hessian-free (DHF) and stochastic gradient descent (SGB). This neural network was used to experiment on large-vocabulary continuous speech recognition (LVCSR) tasks. An improved modified forward-backward algorithm for computing lattice-based expected loss function and gradients is presented that results in a 34% speedup for SGD.

4. **State-Clustering Based Multiple Deep Neural Networks Modeling Approach for Speech Recognition**
   In this paper, a novel DNN-based acoustic modeling framework for speech recognition is proposed. It is shown that the training procedure of multiple deep neural networks (mDNNs) under frame-level cross-entropy and sequence-level discriminative training, can be parallelized for significant speedup using multiple GPUs.
   Sequence training of DNNs is composed of three main steps:
   i) DNN forward pass, in which the posterior probabilities of all HMM states are computed for all feature frames in each utterance
   ii) word graph processing, in which the forward-backward algorithm in each word graph is used to compute statistics,
   iii) DNN back-propagation, in which the error signals in all DNN layers are computed and all DNN weights are updated based on the error signals.

5. **<u>Other Training Methods</u>**
   1. **Parallel Training of An Improved Neural Network for Text Categorization**
      This paper proposes to parallelize the improved neural network. The improved neural network is an improvement over the traditional back-propagation learning algorithm to make learning faster, to avoid local minima, and to improve generalization ability. There are generally two approaches for parallelizing neural networks: pattern parallel and network parallel. Pattern parallel is used in this paper.
   2. **A Parallel Computing Platform for Training Large Scale Neural Networks**
      This paper implements cNeural, a parallel computing platform for training large scale neural network. Training is divided into two phases: training data loading and training process executing. Large scale training datasets are stored in Hbase to reduce the time cost of data loading. The platform could be deployed on Amazon EC2 or general PCs interconnected in a network.
   3. **Parallel Deep Neural Network Training for Big Data on Blue Gene/Q**
      Training in deep neural networks is slow, so the Blue Gene/Q computer system is used to parallelize it. The deep neural network is trained using the Hessian-free 2nd order optimization algorithm. In what would take a month to train a deep neural network with 100 million+ parameters, it took 6 hours with two racks of Blue Gene.
      In the Hessian-free method, gradients are computed over all the training data. A master/worker architecture is used in which worker processes distributed over a compute cluster perform data-parallel computation of gradients and matrix-vector produces, and the master implements the Hessian-free optimization and coordinates the activity of the workers.
   4. **Parallelization of Ensemble Neural Networks for Spatial Land-Use Modeling**
      The hybrid parallel ensemble neural network combines the shared-memory paradigm and the embarrassingly parallel method by using multicore computer clusters. The Fuzzy ARTMAP (ART meaning adaptive resonance theory) is parallelized. It helps solve both the computation and and data intensity issues when applying fuzzy ARTMAP (FAM) to spatial land-use modeling.
      There is an ensemble containing a set of FAM networks, each of which is trained independently on a different subset of data, thus using the embarrassingly parallel paradigm. During the training stage, the whole dataset is divided into n folds to train n networks in parallel.
   5. **Predicting Object-Oriented Software Maintainability using Hybrid Neural Network with Parallel Computing Concept**

A neural-genetic algorithm (a hybrid approach of the neural network and genetic algorithms), is proposed for estimating the maintainability of object-oriented software. The effectiveness of feature reduction techniques such as rough set analysis (RSA) and principal component analysis (PCA) are also explored. To parallelize the neural network, training dataset parallelism is used. Each computing node performs full training of the network for one set of the training dataset. It is based on the master-slave approach. The master uses 'scheduler' for distributing the job among the available computing nodes.

6. **An Enhanced Parallel & Distributed Implementation of the Harmony Search Based Supervised Training of Artificial Neural Networks**
   This paper proposed an enhanced parallel & distributed implementation for the master-slave heterogeneous platform. To achieve maximum performance on a heterogeneous set of processing nodes, parallelism and distributed processing with good load balancing must be exploited. A benchmarking approach that considers the average elapsed time taken by each node to complete the same size workload is used. The master would send a workload to the different heterogeneous slave nodes and record the elapsed time needed by each node to complete its task.