RingTMP: Locally Distributed Machine Learning on Low Power Devices

Prifysgol Abertawe
Swansea University

Christopher Hopkins

Department of Computer Science
Swansea University

This dissertation is submitted for the degree of
Bachelor

April 18, 2021

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 40,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 100 figures.

Christopher Hopkins
October 2020

Abstract

More data than ever is being produced by low power devices such as smart phones and Internet of Things (IoT) devices at the network edge. The data being produced is so enormous it would be infeasible to send it to a centralised location. Instead models can be trained from data distributed across multiple edge nodes, with machine learning algorithms being performed locally. In this paper I explore training multiple low power devices using a new distributed machine learning paradigm *RingTMP*. This paradigm focuses on low power usage and power efficiency while having the capacity for larger models than comparative systems.

Contents

1 Introduction

1.1 Motivation and Context

Machine Learning has become an invisible but ubiquitous part of modern life, and is being used in a plethora of fields and industries. The uses of this technology range from dystopian facial recognition [1] to lifesaving diagnoses [2] and many more purposes besides. Machine Learning leverages existing data to train Machine Learning models in order to perform a task or find patterns, that previously only a human could. The key difference between Machine Learning and conventional programs is that the data itself is used to develop the model. Therefore the quality and quantity of the data can affect the effectiveness of a machine learning model.

As the amount and complexity of the data we are collecting increases, so does the size and complexity of the machine learning models we use to make sense of our data. For example the Internet Archive as of 2020 contains over 70 petabytes of data, while labeled datasets such as AViD have video data in the order of terabytes. [3] We are now reaching a point where the limiting factor of creating a machine learning model is not the data, but the machine learning algorithm itself.

This problem is two pronged. First machine learning models are getting very, very large. For instance GPT-3 the largest NLP model ever trained contains 175 billion parameters. [4] And efforts are being made to create models with trillions of parameters. [5] We have reached the point where its no longer possible to store some machine learning models on a single machine. [6] The second problem is that training a machine learning model is increasingly taking longer and longer. This is because we have more data and larger models, but the algorithms used to train models have fundamentally stayed the same and are inherently sequential and difficult to parallelise.

The popular current solution is to use a parameter server model. In brief the paradigm is made of two different types of components. The parameter server and the workers. The parameter server holds the global parameters of the model. Workers are given the model parameters by the parameter server. The workers then perform an iteration of whichever machine learning algorithm they are performing, modifying the parameters. Then the modified parameters are sent to the parameter server where they are aggregated, the global model

parameters are updated and the cycle continues until the model has converged on an answer.

While this method has many benefits and is certainly faster than training using a single machine, it has two key limiting factors. First, every worker must communicate with a single parameter server, this limits scalability as eventually the network bandwidth becomes saturated severely impacting performance. [6] Secondly many parameter server models require the whole model to be replicated within each node. [7] This means that very large models simply cannot run on many machines.

In this paper I will outline an alternative distributed machine learning framework: *RingTMP*. RingTMP (Ring Topological Model Parallel) is a Ring Topological Model Parallel distributed machine learning framework focusing on optimising Distributed Gradient Descent. This is a novel design drawing in inspiration much research but particularly from the STRADS and DistBelief machine learning frameworks. [8, 9]

I believe my distributed framework may have some advantages over the current paradigm, these briefly are:

- There will be less communication between nodes
- A Potential for larger communication bandwidth between nodes
- Will be able to hold larger models
- Will be able to train neural networking models as faster or faster than a comparative parameter server, to the same level of accuracy

My aims more specifically for this project are to:

- Create a prototype RingTMP framework.
- Create a parameter server model framework.
- Demonstrate less communication between nodes
- Demonstrate that RingTMP is at least as scalable than a generic parameter server
- Demonstrate that RingTMP can hold larger models in comparison to a standard parameter server

- Demonstrate RingTMP can train neural networks to at least the same accuracy in at least the same amount of time

1.2 Overview

This document is split up into the following sections:

- **Introduction** Current section. Introduces the project and its aims.
- **Literature Review** Presents related research material in similar applications and areas.
- **Problem Description** Description of the problem aiming to be solved.
- **Solution Implementation** Details of the implementation of the solution.
- **Reflection** A small section reflecting on the project
- **Conclusion** Summary of the project and the paper.

2 Literature Review

2.1 Brief Introduction to Machine Learning and Neural Networks

To first understand Distributed Machine Learning you must first understand the fundamentals of machine learning and neural networks.

There are many machine learning methods some requiring training data which we call supervised and some being able to find patterns in data without being given solutions called unsupervised. [10] An example of a supervised system may be predicting house prices, using multiple factors about each house (market data, geographic area, square footage etc.) to come to a conclusion about what the house could sell for. This would be trained using data of previously sold houses to predict current ones. An example of an unsupervised system could be identification of new plant species. This could be done by taking as many features of a plant as possible, then apply a clustering algorithm to see if there are two distinct clusters in the data. If there were then that would suggest two different plant species. Neural networks tend to focus on supervised learning and use a form gradient descent called Stochastic Gradient Descent.

Many machine learning algorithms use a cost function to measure how well or badly they are solving a problem, these algorithms use parameters which are internal variables of a machine learning model and define how they solve the problem. If you map $costFunction(x)$, where x is the model parameter, for every x value. Then a graph will be produced $y = costFunction(x)$, the lowest point on the graph will be the global minimum. There may be other troughs higher than the global minimum these are called local minimums. A global minimum represents the lowest value of the cost function which indicates the parameter values produce the best solution for your problem. Initial model parameters are often randomised, which likely means they will start at a high point on the cost function graph, the goal is to get to the lowest point possible. To do this you must *descend* down the *gradient* to a local minima, the algorithm that does this is called gradient descent for that very reason. This often happens in little steps after the observation of each piece of data. However it is computationally expensive to step down the gradient after each example. It is more efficient to calculate the average step of a randomised selection of data. This is know as Stochastic Gradient Descent.

Neural networks are structures that can perform multi-variable gradient descent when provided with training data. Neural networks are comprised of layers of interconnected neurons in a lattice like structure. Each neuron holds parameter information the adjusting of which through gradient descent leads to the solving of a problem through reaching the local minimum of the cost function.

2.2 Limited History of Distributed Machine Learning

One of the first pieces of research into distributed machine learning was 'Distributed Inference for Latent Dirichlet Allocation' in 2008 [11] One of the first instances of distributed machine learning was used to categorise New York Times articles using Latent Dirichlet Allocation (LDA), which identifies the affiliations words have to certain topics. While the paper focused on parallelising the algorithm and running them over multiple artificially isolated cores the results showed that distributed machine learning could have scalability and didn't impact the rate of convergence of the model significantly. This was followed by a paper by Jia et al. [12] which produced much faster results than its predecessors by using memcache layer in every machine, every machine would message every other machine with updates of its local parameters to create an approximate global state, it was mentioned in passing that arranging the nodes in a star topology and caching the values that passed through it could make the system more scalable. After this followed a cambrian explosion of work in this area [9, 13–15] culminating in 2014 when the parameter server as it is known today [6] was produced. This parameter server is highly sophisticated and flexible, accommodating the difference in hardware components while spending more on computation and less time waiting.

2.3 The Communication Issue

Distributed neural networks must communicate with each other in some way in order to work together. This needs to be formalised to be able to measure the efficiency of our machine learning system. Parts of this reasoning already appears in these papers too. [16, 17]

If we consider how a neural network operates if we were to run it on a single node, we

could characterise its computation as such:

$$TIME = I_A(\epsilon) \times T_A \tag{1}$$

Where $I_A(\epsilon)$ is the number of iterations of the algorithm A it takes to reach accuracy ϵ and T_A is the time of each iteration of the algorithm. Here maximising the convergence per iteration or decreasing the time an iteration takes will decrease the runtime of the algorithm. In a distributed setting this equation changes to this:

$$TIME = I_A(\epsilon) \times (c + T_A) \tag{2}$$

In this equation we have the added variable c, this represents time taken for communication per iteration. In a distributed setting this will always remain above a non trivial amount of data. Unfortunately The majority of machine learning algorithms use a stochastic method which means a very large number of iterations ($I_A(\epsilon)$) that take a short amount of time (small T_A). You can see that no matter how small c is there will be a significant impact of the time taken. In fact with a naive approach of communication each iteration almost certainly $c > T_A$.

However this view doesn't take into account the possibility that communication could happen at the same time as an iteration. For example imagine a pipeline of nodes where each nodes performs an iteration but can communicate its previous iteration to the next node at the same time. Then the time taken could be described like so:

$$TIME = I_A(\epsilon) \times max(c, T_A) \tag{3}$$

Here you can see that if you can find a way of making the communication time equal to the time per iteration. Then c would have a negligible effect on the equation.

2.4 Model and Data Parallelism

When creating distributed machine learning models there two different methods for distributing training, model parallelism and data parallelism. These two methods are not mutually exclusive and can be used in conjunction with one another, such as in DistBelief. [9]. Model parallelism is when model parameters are split between the nodes. As data parallelism is when the data is split between the nodes. [18] Often with model parallelism the whole set of

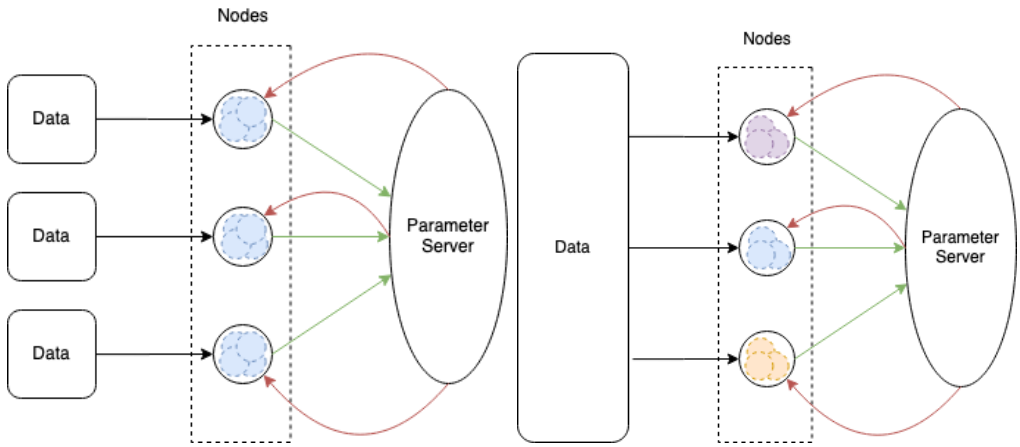

Figure 1: Left: Data Parallelism. Right: Model Parallelism. In both diagrams the green lines indicate local parameters being sent to the parameter server and red lines indicate parameters being sent to the worker nodes.

training data is passed through each node. While in data parallelism its common for each node to hold the whole machine learning model.

The key advantage of model parallelism is that machine learning models can be far larger as they no longer have to sit on one machine. However this one great advantage comes with some disadvantages. Some parameters may take more time to converge than others, this means that at times some nodes may be idle while others are still converging, so the spread of computation is not equal or efficient. [9] Because some parameters converge at different rates a scheduler can be used, which does improve model convergence. However this requires more computational overhead and communication and reduces iteration throughput. [8]

data parallelism has the benefit that data throughput can be very large, making processing using this method very fast. However with more nodes the communication overhead increases as the nodes must communicate the changes in their model parameters to each other. [19] The nodes can communicate with each other synchronously, but this means the computation is only as fast as the slowest node. If the nodes communicate with each other asynchronously then some of the calculations will be made on out of date model parameters so training examples may be needlessly wasted.

References

[1] W. U. Matt Burgess, "Some uk stores are using facial recognition to track shoppers."

[2] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafian, T. Back, M. Chesus, G. S. Corrado, A. Darzi, M. Etemadi, F. Garcia-Vicente, F. J. Gilbert, M. Halling-Brown, D. Hassabis, S. Jansen, A. Karthikesalingam, C. J. Kelly, D. King, J. R. Ledsam, D. Melnick, H. Mostofi, L. Peng, J. J. Reicher, B. Romera-Paredes, R. Sidebottom, M. Suleyman, D. Tse, K. C. Young, J. De Fauw, and S. Shetty, "International evaluation of an ai system for breast cancer screening," *Nature*, vol. 577, pp. 89–94, Jan 2020.

[3] A. Piergiovanni and M. S. Ryoo, "Avid dataset: Anonymized videos from diverse countries," 2020.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[5] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," 2020.

[6] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, (Broomfield, CO), pp. 583–598, USENIX Association, Oct. 2014.

[7] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," 2018.

[8] J. K. Kim, Q. Ho, S. Lee, X. Zheng, W. Dai, G. A. Gibson, and E. P. Xing, "Strads: A distributed framework for scheduled model parallel machine learning," in *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, (New York, NY, USA), Association for Computing Machinery, 2016.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1223–1231, Curran Associates, Inc., 2012.

[10] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[11] D. Newman, P. Smyth, M. Welling, and A. U. Asuncion, "Distributed inference for latent dirichlet allocation," in *Advances in neural information processing systems*, pp. 1081–1088, 2008.

[12] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proc. VLDB Endow.*, vol. 3, p. 703–710, Sept. 2010.

[13] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, (New York, NY, USA), p. 123–132, Association for Computing Machinery, 2012.

[14] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, pp. 19–27, 2014.

[15] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, p. 107–113, Jan. 2008.

[16] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[17] C. Ma, J. Konečný, M. Jaggi, V. Smith, M. I. Jordan, P. Richtárik, and M. Takáč, "Distributed optimization with arbitrary local solvers," *Optimization Methods and Software*, vol. 32, no. 4, pp. 813–848, 2017.

[18] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[19] A. Elgabli, J. Park, A. S. Bedi, M. Bennis, and V. Aggarwal, "Gadmm: Fast and communication efficient framework for distributed machine learning.," *Journal of Machine Learning Research*, vol. 21, no. 76, pp. 1–39, 2020.