Parallel Machine Learning

GROUP 18 – PROJECT 3

Dion Balmforth, Dylan Fu, and Torrance Kam

https://github.com/dylanfu/parallel-ml

TABLE OF CONTRIBUTIONS

WORK ITEM	DION BALMFORTH	DYLAN FU	TORRANCE KAM
DATASET GATHERING	50%	40%	10%
CLOUD DISTRIBUTION	10%	60%	30%
PARAMATER EXPERIMENTING	40%	20%	40%
REPORT	20%	20%	60%
SEQUENTIAL CODE	30%	30%	40%
PRESENTATION	33%	33%	33%

INTRODUCTION

This report gives a brief overview of machine learning, recent technologies and research relevant to the field, and how the different work relates to the training of neural networks. We then discuss the approaches made to parallelise such training, and how the approaches and methods that we have done work to build on that. We discuss the results that we see from our experiments and its significance, and conclude with any future work that should be done.

OVERVIEW

Types of Learning

There are many ways to approach machine learning, but they all fall under four categories. Each approach differs with the type of data that they handle, and also what sort of applications they are suited for.

Supervised Learning - Learning that involves the usage of pre-labelled data to perform inference on unseen, unlabelled data. This pre-labelled data consists of input-output pairs, where particular inputs are already assigned to their outputs, which act as their labels. After training, utilising the pre-labelled data, is done, the algorithm can use what has been learned to map new, unlabelled input data to a possible output. This is done by inspecting different characteristics of the data that were identified during the training, and by looking at how similarly characterised data was labelled [1].

Unsupervised Learning - Similar to supervised learning, but omits the training that is done with pre-labelled data. Because no training is undergone, the algorithm is required to classify the input data with its own labels. The algorithm uses characteristics that it can identify, and creates groups that comprise of data that have similar characteristics to each other. When classifying new data, it compares the new data to the characteristics that are present or absent in the groups, and assigns the data to its most fitting group [2].

Semi-supervised Learning - Merely describes the approach that falls somewhere between supervised and unsupervised learning. In this approach, the input data is not all labelled, however some are. The degree to which data is labelled depends on to what extent the algorithm should learn the pre-defined labels, and how much freedom the algorithm should have when mapping data to their outputs [1].

Reinforcement Learning - Involves the defining of an environment with weights being associated with different actions, and having "agents" attempt to achieve their goals through said actions. The weights can be positive or negative values, and the agents attempt to maximise the cumulative weights of their actions. The agents are usually encouraged to explore the environment to find actions that more positively affect their cumulative weight, in order to find the best possible actions to achieve their goals in the defined environment [3].

Models

Machine learning processes can be expressed as models, which are used to provide a high-level representation.

Bayesian Networks - Models that map the probability of possible reasons for an event to have occurred. In machine learning specifically, it models the probabilities of different features being present in a data sample, given the classification of that data sample [4].

Support Vector Machines - models that have been trained to classify new data as one of two categories. These models are created by separating the initial data into two clusters, and then assigning new data to the cluster that they are found to be closest to [5].

Neural Networks - Systems that have been modelled to use its hidden layers to process data through its input layer, to provide values out of the output layer which can then be interpreted to give results. Neural networks consist of an input and output layer, and hidden layers. Layers are comprised of neurons, where the neurons of a particular layer are connected to the neurons of the adjacent layers via edges. As input data enters the network through its input layer, it is converted into several different real values that represent the different features and characteristics of the data. These values then propagate through the network, which produce new values and eventually give output values that can be interpreted as the classification of the data [6].

During the training of neural networks, there are different metrics that are calculated. One is loss, which is the cumulative value of mistakes that the model has made when trying to classify data. Another one is accuracy, which is the approximate rate at which the model can correctly classify input data [6].

RECENT TECHNOLOGY

Autonomous Vehicles - Vehicles that are able to perform the act of driving without human intervention. They have seen increasing use, and the technology underlying them has been steadily growing. To develop this technology, many machine learning algorithms need to be utilised to ensure correct operation. One algorithm in particular, named YOLO [7] is used for real-time detection of objects while the vehicle is driving, and involves the usage of a neural network. A single neural network is applied to an image (captured rapidly during driving), and predicts the presence of objects. The network takes the whole image into consideration, so not only are the objects looked at, but the environment and context they are within is also considered when the network is classifying the objects in the image [8].

Chatbots - Robots that are trained to create conversation with another human, and to mimic humans while doing so to improve. Popular examples include ChatBot and Evie. Chatbots, like autonomous vehicles, require the assistance of neural networks in order to operate and improve. During conversation, the chatbot receives messages from the human user. The chatbot learns from these messages by analysing how the human user responds to what the chatbot itself is saying, and attempts to mimic the human responses in future conversations. It utilises neural networks to save particular responses to different words and phrases to memory, and will return the mapped response or similar to any human input [9].

RECENT WORK

Background

Data Parallelism – This is parallelisation focused on the partitioning of the data itself across different processors. These techniques tend to be fast and simple, but are limited to smaller models due to requiring entire models being stored in each processor.

Model Parallelism – This is parallelisation that is related to partitioning the model itself across different processors, so that these different processors are responsible for process relating only to the parts of the model that they have stored. These techniques allow extremely large models, however are much more complicated to implement than data parallelism.

Mesh

Built on top of the TensorFlow libraries released by Google in 2015, which are libraries that provide access to many APIs relating to machine learning, Mesh seeks to combine data and model parallelisation processes to improve and simplify the process to train large models.

They discuss the drawbacks of current techniques to parallelise models, such as data parallelism. Data parallelism is limited as alone it cannot parallelise extremely large models, which are required for greater performances of more complex models. It also results in high latency, and is inefficient with smaller batch sizes. Model parallelism techniques mitigate these disadvantages, however unlike data parallelism, it is very difficult to implement and apply to models, due to it being more complicated.

With the introduction of Mesh, not only can data and model parallelism techniques be combined, but it can be utilised by the user easily to specify how a particular model should be parallelised. The difficulty of the implementation of such techniques is taken away from the user, and the user only needs to specify which dimensions to parallelise the model across, having Mesh do all of the complicated implementation on its own.

Mesh also performs these parallelism techniques efficiently, and it has been tested on large, existing models, returning results that surpass current state-of-the-art technology [10].

FlexFlow

FlexFlow is a deep learning framework that was introduced in 2018, and is an execution simulator that seeks to find the most optimal set of parallelisation techniques for any given model. The algorithms that are used result in FlexFlow achieving a much greater speed in finding an optimal parallelisation than existing work, and it does this by using a delta simulation algorithm.

A delta simulation algorithm is an algorithm that is aware of past information, and avoids redundancy by utilising past relevant information as much as possible. Using this, time used running simulations of parallelisation techniques on the input model is significantly reduced, as after completion of a single simulation, it can be inferred from when making adjustments to the parallelisation strategy. It removes the need to fully run simulations on the model when making adjustments to the parallelisation techniques.

FlexFlow investigates new parallelisation techniques within the SOAP search space. Existing techniques only focus on data and model parallelisation, however SOAP identifies more dimensions in which models can be parallelised. Not only does SOAP consider the dimensions relevant to data and model parallelisation, it also considers the dimensions within individual data samples, and how the samples can be parallelised across multiple processors. Defining more dimensions adds complexity, but also increases the potential for better parallelisation performance [11].

METHODOLOGY AND RESULTS

For all local experiments and tests, a machine with the following specifications was used:

- Intel i7 8850H - 12 Hyperthreads (vCPUs)
- 16GB RAM

Fashion_mnist Dataset

We chose the fashion_mnist dataset as it was relatively small and simple, which made it the perfect example to experiment with as it did not consume too many resources. We also decided that since when tweaking different parameters we were only looking at the results relative to each other, we were still able to consider the results applicable when looking at other datasets. This gave us much more time to experiment with different aspects, and explore different possibilities.

Sequential

First we wanted to set a benchmark by recording results for the performance of sequential implementations for the fashion_mnist dataset. It was run five times, and each epoch and the evaluation step was averaged to get a more fair and accurate measurement. These results are given below:

Epoch	**Accuracy**	**Time (s)**
1 (60000 Images)	0.8249	1.92
2 (60000 Images)	0.8662	1.80
3 (60000 Images)	0.8770	1.80
4 (60000 Images)	0.8837	1.80
5 (60000 Images)	0.8914	1.92
Evaluation (10000 Images)	0.8728	0.20

Parameter Experiments

To experiment with how different parameters affect the performance of the training of a neural network model, we decided to create a script that would allow us to tweak the values of different parameters and see how the outputs would change. Various parameters were tweaked, such as the number of cores, number of hidden layers, number of neurons in each hidden layer, and the number of epochs used. As we tweaked a certain parameter, we ensured that the other parameters remained the same, so that we could test the given parameter in isolation. The results are given below:

# Cores	**Loss**	**Accuracy**	**Time (s)**
Benchmark (1)	100.0%	100.0%	100.0%
2	103.5%	99.76%	110.3%
3	105.0%	99.30%	105.1%
4	103.0%	99.74%	102.6%

# Neurons per layer	**Loss**	**Accuracy**	**Time (s)**
Benchmark (32)	100.0%	100.0%	100.0%
128	87.05%	101.5%	173.9%
256	89.17%	101.3%	267.8%
512	84.01%	102.2%	503.4%

# Hidden layers	**Loss**	**Accuracy**	**Time (s)**
Benchmark (1)	100.0%	100.0%	100.0%
4	102.4%	99.71%	118.2%
8	107.0%	98.75%	161.7%

# Epochs	**Loss**	**Accuracy**	**Time (s)**
Benchmark (1)	100.0%	100.0%	100.0%
5	78.16%	103.9%	445.4%
10	76.36%	104.7%	861.7%
20	77.75%	106.1%	1739%

Flower Dataset

As we found that the fashion_mnist dataset may have been too small and simple for us to justify parallelising attempts, we looked to find a dataset that was more complex in nature. A more complex model would give us more room to parallelise, as the communication overhead introduced with parallelisation would be more insignificant when compared to the time required to train the model. The flower dataset had required more hidden layers to use it to train, and thus added more complexity which made it a good model to test parallelisation techniques.

Sequential

The results for the sequential implementation of training using this dataset are given below:

Epoch	**Training Accuracy**	**Time (s)**
1 (2936 Images)	0.6124	270.112
2 (2936 Images)	0.8975	270.112
3 (2936 Images)	0.9437	267.176
4 (2936 Images)	0.9576	261.304
5 (2936 Images)	0.9666	261.304

Evaluation (12 Images)	0.6611	0.516

Distributed Parallelism with Google Cloud

After looking at the results of the tests that we ran for testing the effect of different parameters for the fashion_mnist dataset, we decided to use the two parameters "number of epochs" and "number of cores" to drive our approach of utilising a distributed implementation to increase the performance of the training of our model using the flower dataset. We utilised the Google Cloud servers to assist us in parallelising the model across different devices, with the following specifications:

- Cloud (Single Node):
    - Intel Xeon Scalable Processor (Skylake) - 4 Hyperthreads (vCPUs)
    - 15GB RAM
- Cloud (Distributed):
    - Master:
        - Intel Xeon Scalable Processor (Skylake) - 32 Hyperthreads (vCPUs)
        - 28.8GB RAM
    - 6 Workers:
        - Intel Xeon Scalable Processor (Skylake) - 16 Hyperthreads (vCPUs)
        - 14.4GB RAM
    - 3 Parameter Servers:
        - Intel Xeon Scalable Processor (Skylake) - 8 Hyperthreads (vCPUs)
        - 52GB RAM

Using the distributed method in Google Cloud, the master node handles all of the processes and assigns other nodes tasks. The worker nodes are responsible for the training of the model, and the parameter servers are responsible for maintaining all of the values of the parameters of the model, and joins different instances of the same parameters when necessary.

Three different methods were used to compare the distributed implementation across the cloud, and the results are given below:

Method (1 Epoch)	**Loss**	**Time (s)**
Baseline (Local)	100.0%	100.0%
Cloud (Single Node)	99.75%	278.3%
Cloud (Distributed)	99.88%	36.17%

Method (3 Epochs)	**Loss**	**Time (s)**
Baseline (Local)	100.0%	100.0%
Cloud (Single Node)	100.1%	263.7%
Cloud (Distributed)	100.2%	29.97%

DISCUSSION

Parameter Experimentation

When experimenting with the different parameters to see how each one affected the overall performance of the training of the model, we looked at: the number of cores; the number of neurons per layer; the number of hidden layers; and the number of epochs that was done.

Cores – Compared to the benchmark of the performance of one core, it was found that utilising two cores gave the worst performance in terms of time. We can ignore the fluctuating loss and accuracy measurements, as these metrics differ to this degree naturally when running multiple tests. We believe that two cores give the worst time, as it is the minimum number of cores required to implement parallelisation, meaning that it introduces the communication overhead between cores, while having as little parallelisation as possible. This is supported by the fact that despite all uses of multiple cores returning worse times than a single core, the time is shown to be decreasing with the number of cores that we add, suggesting that if we went beyond four cores, we would be seeing times better than what was achieved with a single core.

Neurons per layer – Here we can see that the loss measurements have been found to be better than the benchmark, when we use a greater number of neurons per layer. However, we do not seem to see any trend, as there is not much evidence of a decrease when we increase the number of neurons per layer past 128. This may be because for the dataset fashion_mnist, 128 neurons per layer could be the or around the limit that the data samples can be optimised, and that adding neurons would not have any effect on the loss and accuracy. We also find that the time taken to train the model increases with the number of neurons, showing that it is important that we do not needlessly extend the number of neurons per layer past what can be used by the dataset, as we may suffer large increases in time, while not having much of an increase in the quality of the trained model.

Hidden Layers – From the results, it is evident that this parameter had the most negative effect. An increase in the number of layers showed negative impacts on all measured metrics. This may be due to the same reasons that more neurons per layer showed diminishing results, where the fashion_mnist dataset cannot utilise any more than a set number of hidden layers, given the simplicity of the dataset compared to others. It shows that it is important that we do not try to force more hidden layers to train a better model, as some datasets may not be able to utilise the extra hidden layers.

Epochs – As expected, positive changes in the loss and accuracy metrics occurred when we increased the number of epochs. The more epochs we do, the more the model gets to train and the better it gets at classifying data. Although we do seemingly end up with a better-trained model, we do suffer from drastic increases in time, which gives us the trade-off between time and accuracy. What cannot be seen from the results however is the amount overfitting that may have occurred due to a large number of epochs. Unfortunately not included in the tests was the validation step, which is required to see how well our model has been trained, before testing the model. As the testing step was done with similar objects that were used to train, the impacts of overfitting were not seen. We need to be careful that we do not train a model too much with too narrow of a dataset, or overfitting may occur.

Distributed Implementation

Looking at the results of the distributed implementation, we can see large speedup percentages when we utilise Google Cloud for our distributed implementation. Distributed training on Google Cloud allowed efficient data parallelisation, due to many workers being able to be used for the parallelisation, and the greater specifications of these nodes.

The method using the single node on Google Cloud was found to be much slower than local training. This may be due to the slightly worse specifications of the single node, compared to the local machine, however it is most likely due to network bandwidth limitations of Google Cloud. As the was only a single node being utilised on Google Cloud, parts of the model were required to be saved to the cloud in the form of checkpoints, which occurred every 500 steps during training. As saving took a substantial amount of time (around 20-30 seconds) this is most likely the reason why training using a single node on Google Cloud was far worse than using the local machine.

CONCLUSION

This report discusses the efforts into parallelising the processes of machine learning – specifically the training of neural networks utilising datasets. An overview of neural networks and machine learning as a whole was given, in addition to recent work and technologies in the area, and aspects relating to neural networks were discussed. Sequential benchmarks were made using the fashion_mnist and flower datasets, and the results from these benchmarks were used to compare performance using parallelisation.

It was found that running more epochs seemingly improves the quality of the trained model, and increasing the number of cores would eventually result in speedups. However, care needs to be taken when adjusting the number of hidden layers, and neurons within these layers, as depending on the complexity of the dataset and the model, the layers and neurons can only be utilised to a degree, meaning that there is a risk of decreasing speed with no potential benefit.

Using Google Cloud services, we found that data parallelism through the approach of distributed implementation across different nodes/devices results in large speedups, without the loss of any quality of the trained model. However, more experiments should be done in the future with more controlled variables, to ensure that the approach itself is causing improvements.

REFERENCES

[1] Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[2] Tucker, A. B. (Ed.). (2004). *Computer science handbook*. CRC press.

[3] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., & Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, No. 2). Belmont, MA: Athena scientific.

[4] Champ, C. W., & Shepherd, D. K. (2007). Encyclopedia of statistics in quality and reliability.

[5] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273-297.

[6] Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009, June). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning* (pp. 609-616). ACM.

[7] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[8] https://www.intellias.com/who-takes-the-lead-in-the-autonomous-driving-race/

[9] https://blog.statsbot.co/chatbots-machine-learning-e83698b1a91e

[10] Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., ... & Sepassi, R. (2018). Mesh-tensorflow: Deep learning for supercomputers. In Advances in Neural Information Processing Systems (pp. 10414-10423).

[11] Lu, W., Yan, G., Li, J., Gong, S., Han, Y., & Li, X. (2017, February). Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 553-564). IEEE.