Multi-node Training with Tensorflow

1st Kontaras Marinos
Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
el17050@mail.ntua.gr

2nd Bouras Dimitrios-Stamatios
Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
el17072@mail.ntua.gr

3rd Theodoros Siozos
Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
0000-0002-0361-1646

***Abstract*—Nowadays, deep learning is thriving due to the vast amount of data that are available. The main drawback of deep learning leans on the high amount of time that is required for the models' training. In order to alleviate this phenomenon this document investigates the effect of distributed training of deep neural networks on training time. The development is based on the TensorFlow library and in order to extract a more general conclusion, several machine learning problems were studied with a different neural network for each of them. Each model was trained on one, two and three nodes for the same number of epochs, which is a crucial condition in order to produce comparable results. The training distribution is achieved by synchronously training multiple workers (each node represents a worker) according to data parallelism. In other words a fraction of the dataset is assigned to each worker. This technique consists the most popular technique that TensorFlow provides for distributed training on multiple CPUs.**

***Index Terms*—distributed training, multiple nodes, neural networks, TensorFlow**

I. Introduction

In order to handle the amount of data that are generated everyday as well as to solve complex machine learning problems in the field of computer vision and NLP, large neural networks have been developed that require a vast amount of time to be trained. The BERT transformer [1] initial training lasted four days and it required 4 cloud TPUs. This provides an indication of how crucial it us to distribute the training process. The difference between traditional and distributed training time-wise is even more obvious when it comes to well-established computer vision models [2]. The resnet50 managed to be trained in an astonishing time of 224, using 2176 GPUs [3] while training with 8 GPUs took 29 hours.

As the role of distributed training became clear, many different techniques of distribution were introduced. The main categories of distributed training consist of data parallelism [4]–[6] (Figure 1) and model parallelism [7]–[9] (Figure 2). In the first case - which is the case that is investigated in the current document - the dataset is split into 'N' parts, where 'N' is the number of worker -nodes . These parts are then assigned to parallel computational machines. Post that, gradients are calculated for each copy of the model, after which all the models exchange the gradients. In the end, the values of these gradients are averaged. For every node, the same parameters are used for the forward propagation. A small batch of data is sent to every node, and the gradient is computed normally and sent back to the main node.

In the case of model parallelism though the model is partitioned into 'N' parts, just like data parallelism, where 'N' is the number of worker - nodes . Each model part is then placed on an individual node. The batch of nodes is then calculated sequentially in this manner, starting with node#0, node#1 and continuing until node#N, regarding forward propagation. Backward propagation on the other end begins with the reverse, node#N and ends at node#0.

Model parallelism has some obvious benefits. It can be used to train a model such that it does not fit into just a single node. But when computing is moving in a sequential fashion, for example, when node#1 is in computation, the others simply lie idle which can be hardware inefficient and requires methods of model parallelism pipelining for the training speed to increase. [10], [11]. It is worth mentioning, that since most models can fit on two machines , the advantages of model parallelism do not scale as will as the number of nodes increases .Thus the focus of this work will be placed on model parallelism [12].

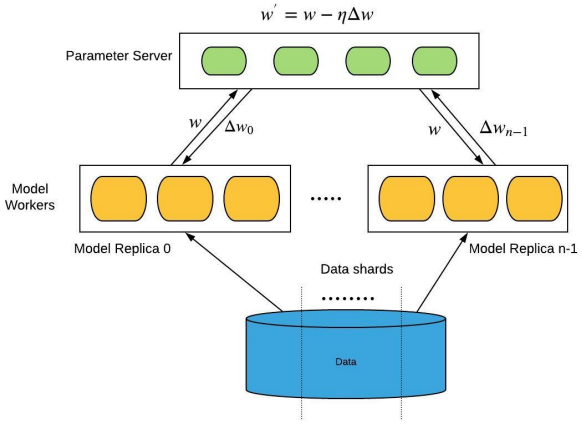

Fig. 1. Example of data parallelism

The main strategies supported by TensorFlow and belong in the data parallelism category are the mirrored strategy, the TPU strategy, the multi-worker mirrored strategy and the parameter server strategy. The mirrored strategy supports synchronous distributed training on multiple GPUs on one machine. Each GPU contains a replica of the model and the variables are mirrored across all the replicas and are kept in sync, by applying identical updates of the variable values

across the GPUs. The TPU strategy is the same as the GPU strategy with the only notable difference that the TPU strategy provides the chance of training on Tensor Processing Units (TPUs).

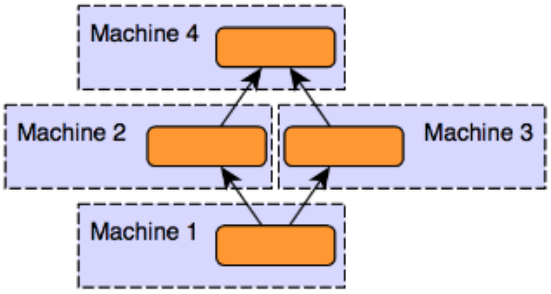

Fig. 2. Model parallelism

The multi-worker mirrored strategy is also very similar to the mirrored strategy. Its key difference consists of the fact that in the former the training is implemented across various machines (nodes). That is the reason why this strategy fits perfectly to the goal of this document, as we would like to investigate the effect of multi-node training. Last but not least, the parameter server strategy creates a training cluster that consists of workers and parameter servers. This type of training is asynchronous, on the grounds that parameter servers create the variables and worker servers read them and update them in each step without synchronizing with each other. The reason that this strategy is not studied in this document is that it requires many nodes (one would be the coordinator, some would be the workers and the remaining would be the parameter servers), while we had at most 3 nodes at our disposal.

The rest of this document is organized as follows. Section II describes the hardware and software setup, as well as a profiling for each neural network / dataset that was included in the conducted experiments. Section III consists of the results and their corresponding analysis. Finally, Section IV contains the discussion presenting future work that would improve the presented results, while Section V provides the conclusion of the document and Section VI contains the link for the developed code.

II. Materials and Methods

A. Node specifications

For the purpose of the study, we were given 3 different Virtual Machines (VMs) from Okeanos, which is the cloud service for the greek research and academic community. The VMs were created using the okeanos framework and according to the project hardware limits. As a result each of the VMs is defined by the following hardware specifications:

- 4-core CPU
- 8gb RAM
- 30gb storage
- Ubuntu 16.04.3 LTS OS

Following the capabilities of these VMs we adjusted the models to be able to run under these specifications.

B. Communication Setup

Initially, a public IP was assigned to a VM, via the Okeanos framework, so it can have access to the Internet. Then, a private network was created and all three VMs were added and received the following hostnames (for ease of use) and private IPs:

- VM 1 : $hostname$ slave1 - $IPv4$ 192.168.1.1
- VM 2 : $hostname$ master - $IPv4$ 192.168.1.2
- VM 3 : $hostname$ slave2 - $IPv4$ 192.168.1.3

The experimental layout of the machines is shown in Figure 3.

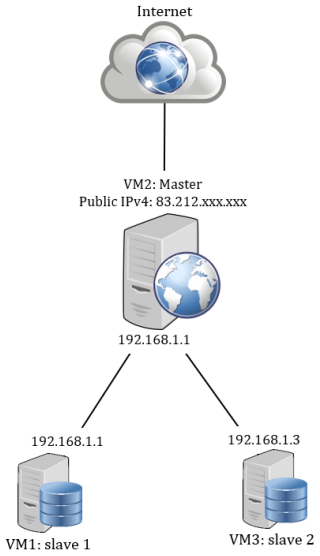

Fig. 3. Experimental network layout of virtual machines

The master-slave communication is maintained via SSH/TCP (port 22). In order to enable communication between slaves and master for distributed training via ssh, it is necessary to configure key-based authentication for SSH. Instead of the remote system asking for a password at each connection, authentication can be negotiated automatically using a public and private SSH key pair.

To authenticate using SSH keys, the master must have an SSH key pair on its local machine, created with the $ssh - keygen$ command in Linux and located in the .ssh hidden directory within the master's home directory. The public key must then be copied to a file within the slaves' home directories at /.ssh/authorized_keys. This file contains a list of the public keys, one per line, that are authorized to connect to this account. For convenience, we copied the entire .ssh folder to the slaves using the scp command.

Because only one public IP is allocated (from Okeanos), which is assigned to the master as described above, the only way the slaves can access the internet is through the master. For this reason, it is necessary to configure the master to act as a router gateway for the slaves. For this purpose, NAT is enabled on the master machine with the following Linux commands:

```
echo "1" > /proc/sys/net/ipv4/ip\_forward
iptables -F
iptables -t nat -F
iptables -t nat -A POSTROUTING -o eth0
         -j MASQUERADE
```

After that, a default gateway must be configured on the slaves' machines to point to the master's private ip address (Linux command: route add default gw) so that they can access the Internet through the router VM (master).

Information about the distributed training is exchanged between VMs via a randomly predefined port on each machine (in our case port 2222 was used).

C. Software infrastructure

Since the node network is set up, it is now possible to set up the distribution protocol which will enable the training across all the nodes. The distributed training is based on the TensorFlow distributed API and more specifically on its MultiWorkerMirrored strategy (Figure 4).

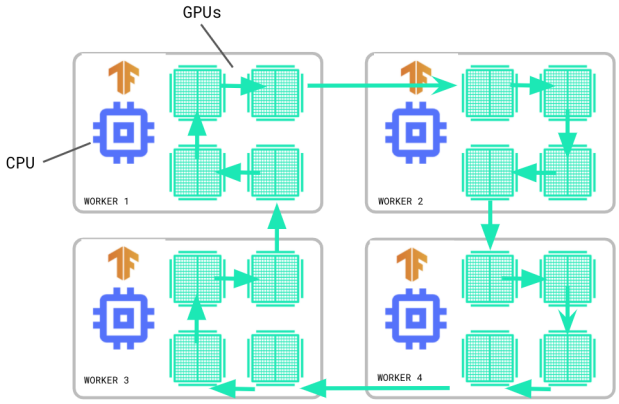

Fig. 4. TensorFlow Multi-Worker Mirrored strategy

In this subsection the full pipeline of the multi-worker mirrored strategy will be explained in detail:

Initially the data are divided into n number of partitions, where n is the total number of available workers in the compute cluster. In the experiments studied in this document n will be up to 3 as 3 Virtual Machines are provided, which will act as the worker nodes.

The model is replicated across each worker node, that then trains on its own subset of the data, that was produced by the initial data partition. Since the Mirror Multi worker strategy is a synchronous training method , the forward propagation begins at the same time in all of the workers and they compute a different output and gradients. Here each worker waits for all other workers to complete their training loops so as to calculate their respective gradients.

Now after all the workers have completed computing gradients, all of them start communicating with each other and aggregate the gradients using the all-reduce algorithm. Now after getting the updated gradients using the all-reduce algorithm, each worker continues with the backward propagation and updates the local copy of the weights. The next forward propagation begins after all the workers update their weights, and that is the reason that this technique is called synchronous.

Note that the last points mentioned are actually the main reasons that there is a time overhead in distributed compared to conventional neural network training. In other words, the communication and synchronization processes, create a time delay.

Taking a closer look into the all-reduce algorithm that is used in the multi-mirrored worker strategy, it directs all the workers to send their gradients to a single worker known as driver worker (in this case the master, aka VM 2) which is responsible for the reduction of gradients and sending the updated gradients to all workers.

As it is shown in Figure 5 the dataset is split in batches and trained in different nodes, as explained previously.

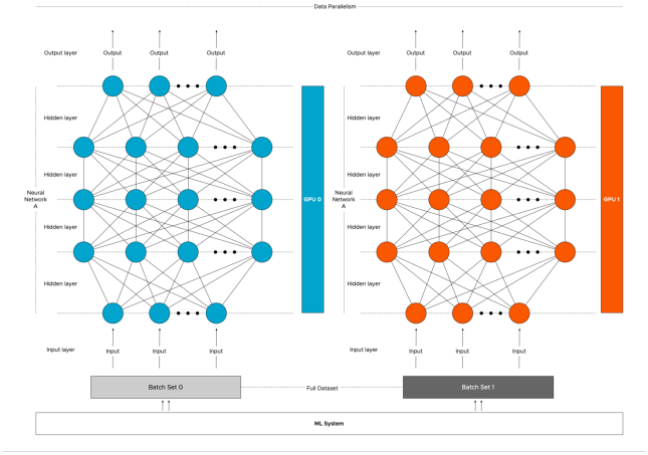

Fig. 5. Splitting of dataset in multiple nodes

In order to execute the Multi-Worker Mirrored strategy every node should execute the worker python file, under the TF_CONFIG environment variable that contains the worker's corresponding task type and task id. The worker python file is the file that runs the models under the chosen strategy. It is worth noting that a conda environment was created, for the project at hand, so as to keep all the dependencies in line.

To get even more technical, the TF_CONFIG variable is defined as a cluster dictionary that contains two keys: the workers (which consist of the IPs of all the nodes that we want to train across), and the task which contains the specific node's index (0 for the 1st node, 1 for the 2nd, etc.). As it is already mentioned the worker file, which contains the model's

training code, should run for every node. So for example if we want to train in one node (traditional training), only the script for the first VM will be executed, while if we want to train across two nodes, the script will be executed for both of the VMs under their corresponding TF_CONFIG. A crucial point regarding this strategy is based on the fact that the global batch size the model is compiled with should be equal to the initial batch size times the number of nodes taking part in the training process:

$$global\_batch\_size = batch\_size * nodes$$

The reason behind this choice lies on the equal division of the dataset to the nodes. Hence, for the node to have the intended batch size the global batch size should be the number of nodes times greater.

Following these steps, distributed training is successfully implemented.

D. Models and Datasets

The distributed training effect on the training time and accuracy is evaluated by implementing it for various dataset/model pairs.

1) Cifar 10: Cifar 10 is considered one of the most popular dataset on the field of computer vision and it was developed by the University of Toronto. The features of the dataset are the rgb pixel values of images with size 32x32, while the labels are the class of each image. There are 10 different classes which are presented in Figure 6. A total of 50,000 different examples were used for the model's training.

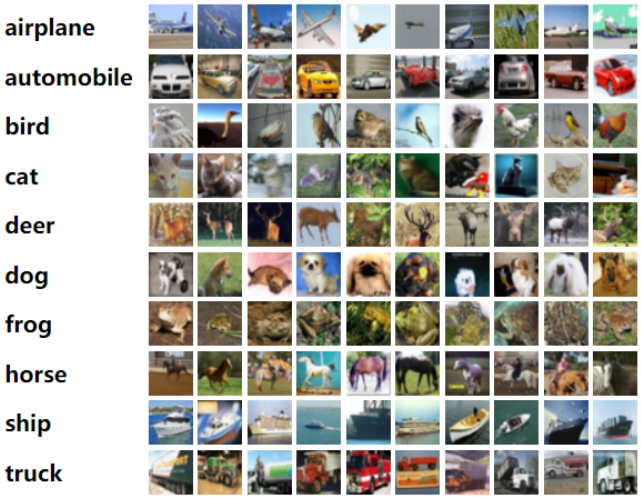

Fig. 6. Cifar 10 dataset

For the Cifar 10 dataset we chose a model that achieves one of the highest accuracy scores. This model is consisted of 2,396,330 trainable parameters and 896 non-trainable. The model's architecture is defined by multiple convolutional layers with batch normalization and maximum pooling layers in between them. One dense layer is used between the output layer, which is also a dense layer with a softmax activation function, completing the architecture of the model (Figure 7).

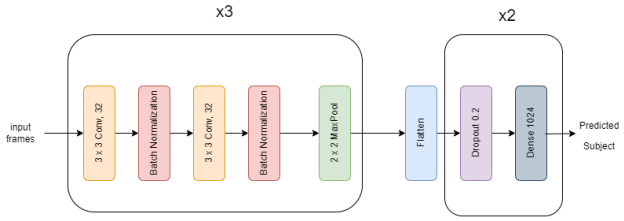

Fig. 7. Model architecture for the Cifar 10 Dataset

2) Natural Images: The natural images dataset (Figure 8) consist of 8 different classes of images and their corresponding images. It is similar to the Cifar 10 dataset, but with different classes and images with size 128x128 pixels. The training set size is 5519.

Fig. 8. Natural Images dataset

The model corresponding to this dataset is the the DenseNet121 network [13] which achieves very high accuracy and is a member of the Densenet CNN family .In more detail a DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers .

Due to RAM limitations we fine tuned the last 2 layers of the DenseNet model, keeping the other layers frozen to their pretrained values. After the DenseNet network some pooling, dropout and dense layers were added before the final output, which is a dense layer with size 8 and softmax activation function. The total parameters of the model are

8,622,152 while the trainable parameters are 1,581,576 due to the freezing of the majority of the DenseNet's layers. The complete architecture is presented in Figure 9.

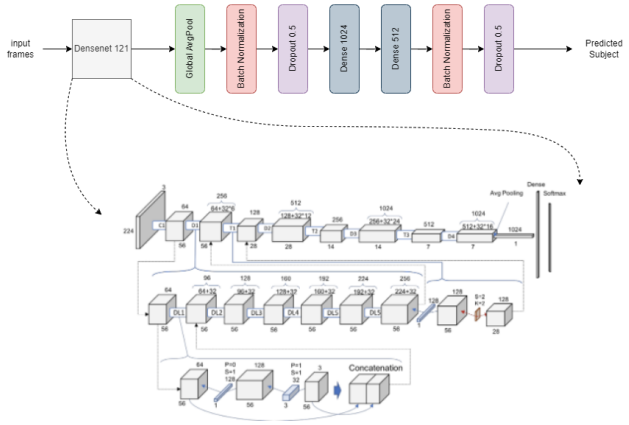

Fig. 9. Model architecture for the Natural Images Dataset

3) Fashion MNIST: Fashion-MNIST dataset (Figure 10) is also a computer vision dataset consists of Zalando's article images which are grayscale images with size 28x28 pixels associated with a label from 10 classes of clothing. The training set includes 60,000 of these images.

A much simpler model (Figure 11) was implemented for the fashion-MNIST dataset containing one convolutional, one max pooling, one flatten and two dense layers the last of which is the output with a softmax activation. The total parameters are 1,256,150 and all trainable.

Fig. 10. Fashion MNIST dataset

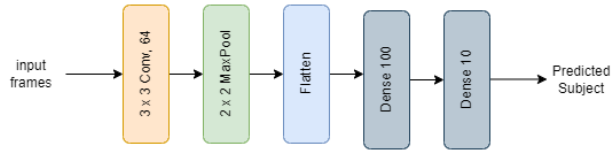

Fig. 11. Model architecture for the Fashion MNIST dataset

4) MNIST: MNIST (Figure 12) is the most basic and popular dataset in the field of machine learning. It contains 60,000 training images with size 28x28 pixels that correspond to handwritten digits and the labels are the intended digits of the handwritten ones.

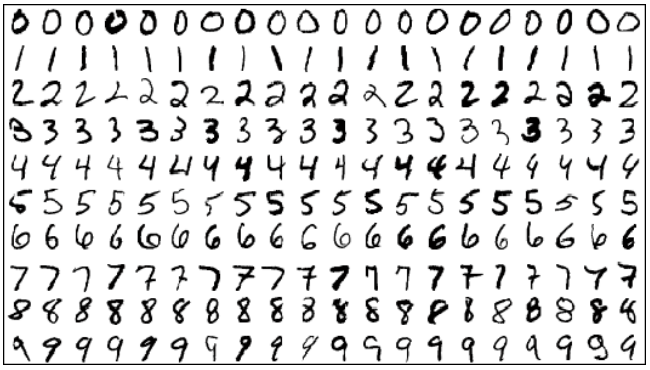

Fig. 12. MNIST Dataset

The MNIST dataset was picked as a use case for the ResNet50 model, on the grounds that we wanted to study the effect of distributed training on such a well established model in the field of computer vision.The Resnet50 is a version of the Resnet model which was one of the most groundbreaking works in computer vision . It is based on the idea of residual training .In more detail , while in general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers, on the other hand in the case of residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of nth layer to some (n+x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved.

We were able to fit the entire ResNet50 model in the VMs without RAM problems and the final model consists of the ResNet50 with a max pooling layer and a dense layer as the output (Figure 13). The total parameters for the model are 23,608,202 with 23,555,082 of them being trainable.

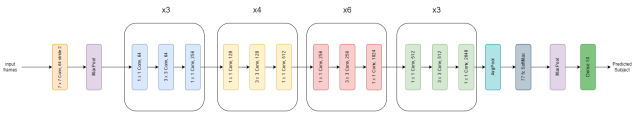

Fig. 13. Model architecture for the MNIST dataset

5) NLTK Movie Reviews: The NLTK corpus movie reviews dataset (Figure 14) will be used to perform sentimental analyses on reviews from films and categorise them as postitive or negative. The training set contains 1,000 positive and 1,000 negative reviews.

Fig. 14. NLTK Movie Reviews Dataset

The movie reviews dataset was chosen because we want to test the BERT model (Figure 15).

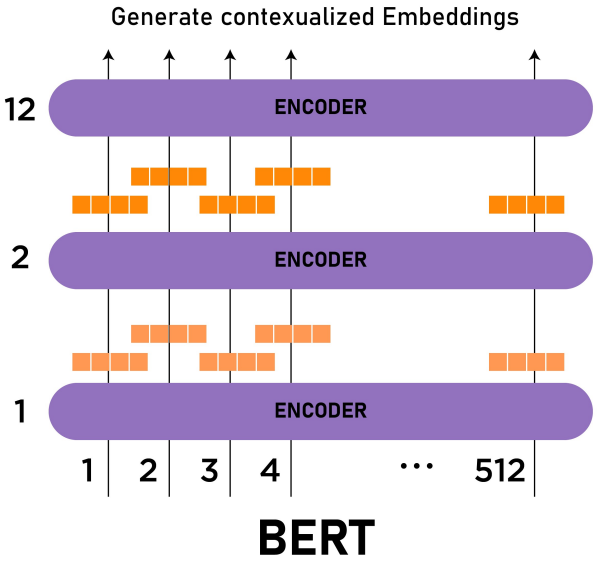

Fig. 15. BERT model architecture

The BERT Transformer is one the most popular Transformer models and produces great for a wide variety of Natural Language Processing tasks. By applying the bidirectional training of Transformers (an attention mechanism that learns contextual relations between words) to language modelling it has yielded state of the art results.

In contrast to other transformer based models though ,which read the text input sequentially (left-to-right or right-to-left), the BERT Transformer reads the entire sequence of words at once [1].This characteristic allows the model to learn the context of a word based on all of its surroundings and that is why it performs so well.

Although, BERT was the start of many high achieving Transformer models, it is one of the most demanding when it comes to computational power, providing a very fitting example for the objective of this study.

The BERT baseline model that we are using has 109,483,778 total parameters, but due to RAM and CPU limitations we froze the BERT layers except the last one, ending up with 7,089,410 trainable parameters.

III. Results and Analysis

A. Results presentation

Since the process of multi-node training is made clear and the experiment setup is analyzed in detail, the results of these experiments will be presented and analyzed in depth. Table I contains the main parameters of each model that affect the training time, such as the epochs, the initial batch size and the size of the model, while table II contains the total results of the experiments. As it is already mentioned the multi-node batch size is greater than the initial batch size by nodes times.

TABLE I
TABLE WITH MODEL PARAMETERS

Dataset	Epochs	Initial batch size	Trainable weights	Total weights
Cifar 10	5	64	2.396.330	2.397.226
Natural Images	5	16,64,256	1.583.624	8.622.152
Fashion MNIST	5	32,64,128,256,512	1.256.150	1.256.150
MNIST	2	64,256	23.555.082	23.608.202
Movie Reviews	3	32	7.089.410	109.483.778

By running multi-node training of the model built for the Cifar dataset, we notice a significant improvement on training time (Figure 16), while the training accuracy remains roughly the same (Figure 17).

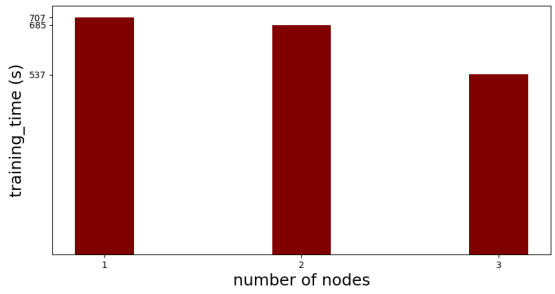

Fig. 16. Training time across nodes when training cifar_10 model

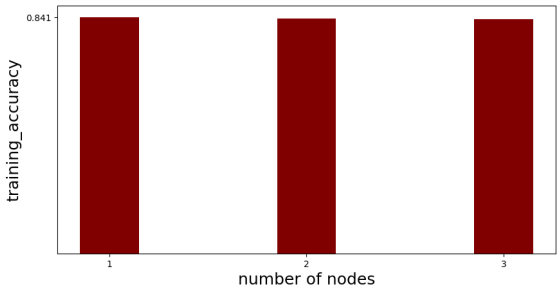

Fig. 17. Training accuracy across nodes when training cifar_10 model

In the case of the model built for the natural images dataset, a significant improvement on training time can be notices

TABLE II
TABLE WITH OVERALL RESULTS

Dataset	Nodes	Batch Size	Training Time (s)	Training Accuracy
Cifar 10	1	64	707	**0.841**
	2		685	0.837
	3		**537**	0.835
Natural Images	1	16	656	**0.981**
	2		410	0.976
	3		**402**	0.978
	1	64	530	0.986
	2		397	0.986
	3		**384**	**0.986**
	1	256	455	**0.989**
	2		360	0.985
	3		**216**	0.985
Fashion MNIST	1	32	**142**	**0.932**
	2		476	0.925
	3		418	0.920
	1	64	**125**	**0.930**
	2		249	0.922
	3		209	0.914
	1	128	**104**	**0.916**
	2		150	0.902
	3		127	0.897
	1	256	115	**0.904**
	2		106	0.889
	3		**83**	0.883
	1	512	118	**0.889**
	2		81	0.875
	3		**66**	0.867
MNIST	1	64	**1308**	0.978
	2		2052	0.982
	3		1794	**0.986**
	1	256	987	0.989
	2		832	0.992
	3		**683**	**0.992**
Movie Reviews	1	32	5951	0.896
	2		3104	**0.898**
	3		**2070**	0.874

(Figure 18), mainly between traditional training and 2-node training for small batch sizes (16 and 64). The same batch sizes training across 3 nodes slightly decreases the training time from the one that 2-node training produced. On the other hand, for batch size 256, 3-node training is by far the best in terms of training time, with substantial reduction from training time of previous batch sizes and a big difference from 2-node (1.7 times bigger than 3-node) and traditional method (2 times bigger than 3-node) of the same batch size. It is very important that in terms of training accuracy (Figure 19), we do not notice any difference between the three different training methods. Comparing the traditional training time of batch size 16 with 3-node training time of batch size 256 we observe a reduction of about 70% that is not accompanied by notable reduction in training accuracy, which makes 3-node training with big batch size the best training solution for this specific dataset.

Concerning the Fashion MNIST dataset, as observed from Figure 20 and Figure 21, increasing the batch size decreases

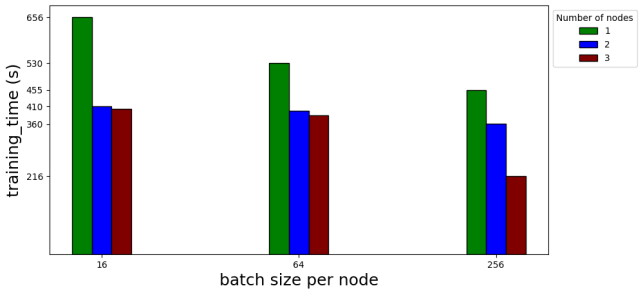

Fig. 18. Training time across nodes when training natural_images model

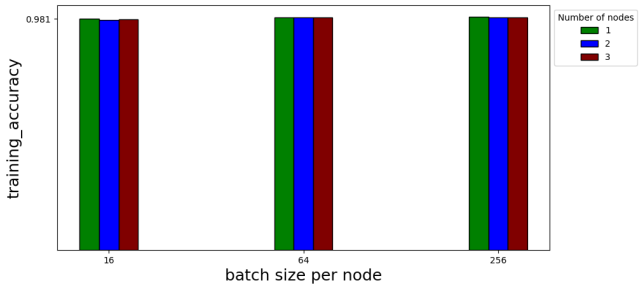

Fig. 19. Training accuracy across nodes when training natural_images model

the training time, but at the same time it decreases training accuracy. It can be seen, also, that traditional training time not only does not decrease at the same rate as 2- and 3-node training time while increasing the batch size, but it slightly increases. We therefore conclude that for smaller batch sizes (32, 64, 128) traditional training method with 1 node is preferable, whereas on the other hand for bigger batch sizes (256, 512) 2- and 3-node training are significantly better. For every batch size, 3-node is way more efficient than 2-node training in terms of training time, but for 2-node training we observe the biggest reduction of about 85% in training time from batch size 32 to 512.

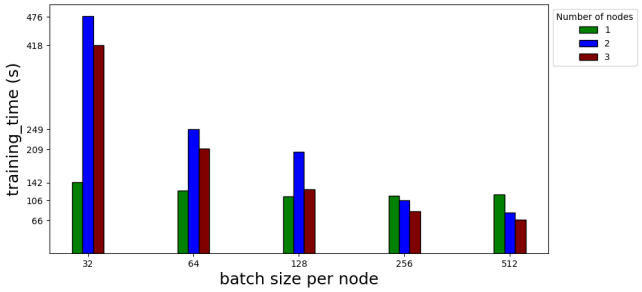

Fig. 20. Training time across nodes when training fasion_MNIST model

Similar behavior to the Fashion MNIST model with respect to training time is observed for the MNIST model (Figure 22). The only difference is that this time the training accuracy remains roughly the same for every batch size and number of nodes (Figure 23) and training time does not increase for traditional training while increasing batch size. As observed,

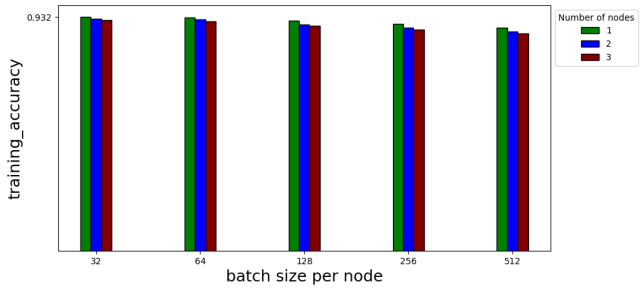

Fig. 21. Training accuracy across nodes when training fashion_MNIST model

there is huge reduction in training time in MNIST model for multi-node training, which makes multi-node training better and more efficient for bigger batch sizes. As in the Fashion MNIST model, traditional training is preferable for smaller batch sizes. This conclusion is logical since as the batch size increases, the communication overhead between master and slaves decreases just like in the communication networks. In other words, the number of times master and slaves have to exchange information about the training process is decreasing and at the same time the information aggregation that takes place in order to update weights (of neural network) is increasing.

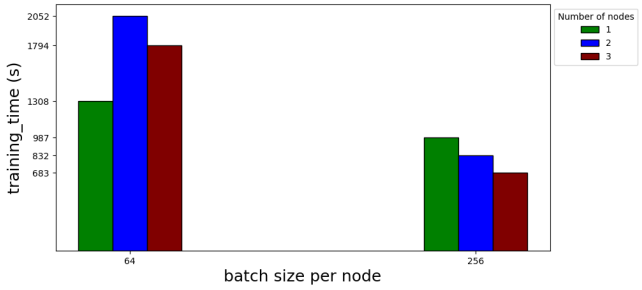

Fig. 22. Training time across nodes when training MNIST model

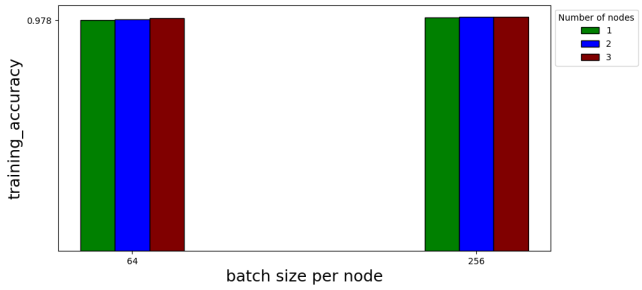

Fig. 23. Training accuracy across nodes when training MNIST model

Distributed training on BERT Transformers leads to astonishing results as training across 2 nodes nearly doubles down the training time and training across 3 nodes almost triples it down (Figure 24), while keeping the accuracy on the same level (Figure 25).

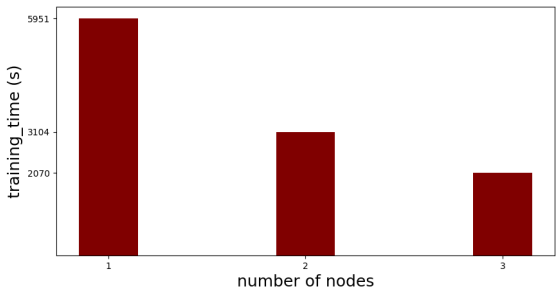

Fig. 24. Training time across nodes when training NLTK_movie_reviews model

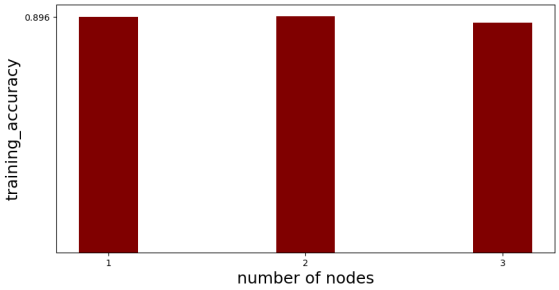

Fig. 25. Training accuracy across nodes when training NLTK_movie_reviews model

The presented results can be summed up in Figure 26 and Figure 27. In Figure 26 the training time improvement in comparison with the conventional training is presented for each model, picking the best batch size value for the models that were trained multiple times for different batch values. In Figure 27 the average time improvement is presented for the results presented in Figure 26.

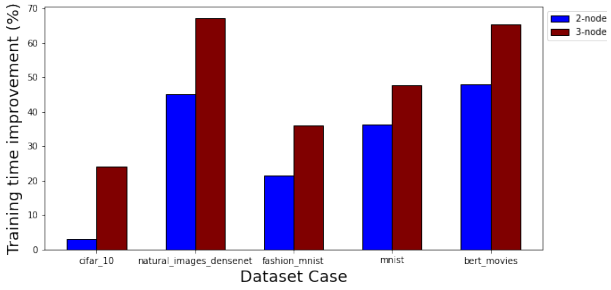

Fig. 26. Training time percentage improvement of each dataset

B. General Analysis

A simple approach to distributed training that involves splitting the dataset and training each part in different node might lead one to assume that the training speed would be inversely proportional to the training time as indeed the number of mathematical operations, that must now be performed by each node are actually the total computational load of the neural

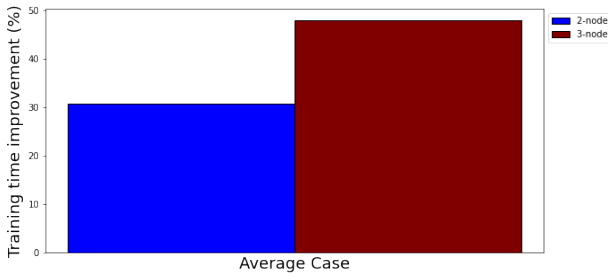

Fig. 27. Training time average percentage improvement across datasets

network training divided by the number of nodes. Nonetheless a closer examination of the mechanisms that perform the distributed training make it rather obvious that an overhead is introduced when training on multiple nodes compared to the baseline technique. This overhead is associated with the aggregation and update of the weights that are stored in each worker – node and can prove to be very important for the time outcome of distributed training.

The overhead that node communication induce is mostly visible when comparing the 2-node distributed training to the convetional one. As we can see in the MNIST and fashion MNIST cases for smaller batch sizes the performance of the distributed training method provides inferior results compared to the conventional training. The cause of this phenomenon is that having smaller batch sizes leads to more times that nodes need to communicate with each other and synchronize, as the parameter updating and the synchronization trigger after every batch-size training. Increasing the batch size without caution though, could lead to lower accuracy due to fewer back-propagation processes. The solution is provided by tuning the batch size parameter and finding the balance point, in which the distributed training time is the lowest possible without significant reduction of the accuracy. These statements can be validated by observing the Natural Images, MNIST and Fashion MNIST figures. With close investigation and when regarding the parameter number as well, it is excluded that for simpler models (e.g. for the fashion MNIST model), the distributed overhead brings high time penalties with smaller batch sizes as the forward and backward propagation require less time than the communication and synchronization (training time for 2-node training significantly higher than the one for conventional training). In the cases studied, it is manages to find batch sizes that utilize the distribution and do not take their toll to the training accuracy. In larger and more complex models, the improvement that distributed techniques bring is more straight-forward (e.g. BERT case), and sometimes they present the only viable choice.

To sum up these observations, in order for simpler models and smaller datasets to make use of distributed training, the batch size hyperparameter should be tuned and still in some cases the distributed training might not prove useful. On the other hand models with high complexity, utilise in the maximum way the distributed methods.

It is also observed that there are no cases were there is a decrease in training time when comparing 2 node training and 3 node training. This leads to the conclusion that the overhead that is added to the distributed strategies compared to the conventional ones is not proportional to the number of nodes. So by increasing the number of worker nodes the overhead effects concering training time are gradually mitigating. So one might assume that for each model there is a certain number of nodes that is large enough to decrease the effect of the the aggregation and update of the weights time, but not large enough for the over-splitting of the dataset to start to cause saturation of the time improvement.

Due to limited resources, testing with more than 3 different machines was not possible, but using statistical methods a chart has been constructed to visualise the speedup that an average dataset might experience on distributed training where the number of nodes is larger than 3 , with the ranging reaching up to ten worker - nodes (Figure 28).

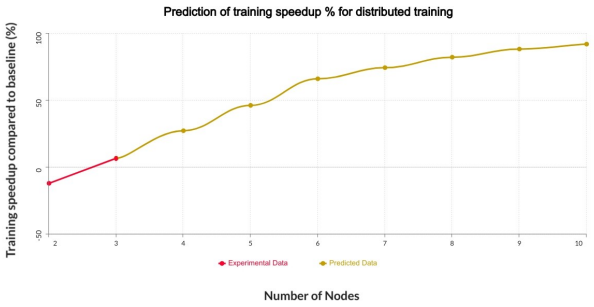

Fig. 28. Training time speedup predictions for distributed training(2 to 10 nodes)

As we can observe in Figure 26 the speedup slope is higher for node numbers two up to six , with six nodes being actually a good compromise between sufficient training speedup (67 %) and number of computational units required.After six worker nodes the benefit start to diminish for every new worker node added to the distributed training system and a saturation is observed when a ten node strategy is reached. The higher node projection is just a predicted result for the average of the five models that we examined and the results might different when examining different datasets and models as far as the number of nodes to reach saturation and the ideal node number but the form of the slope is expected to be similar in most cases and it is backed up theoretically by the weight aggregation and update overhead's effect on training time relationship with the number of worker nodes, that was discussed earlier in this section.

IV. Discussion

According to the presented study, the are two main points that future work should focus on. At first, producing results from more use cases as well as training across a higher number of nodes is very important so as to prove the hypothesis.

Furthermore, mathematical formulas should be defined in order to describe the distributed training overhead and the optimal number of nodes.

Experimenting with more datasets and models should make clearer the advantages and the drawbacks of distributed training. It is also worth, studying how the multi-node training is affected by the model's architecture and not only the number of parameters.

V. Conclusion

Concluding the study, distributed training is a powerful tool in the field of neural networks, especially on computational-demanding cases. The training batch size has also proved the parameter with the most importance, while tuning it right can benefit the training time results greatly.

VI. Developed Code

The code developed for this project is publicly availiable in the following link: https://github.com/Thodorissio/distributed-training. The repository provides clear instructions for the use of the code. The setup should be created in a similar way as described in the document.

References

[1] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[2] Jia, Xianyan, et al. "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes." arXiv preprint arXiv:1807.11205 (2018).

[3] Tanaka, Yoshiki, and Yuichi Kageyama. "ImageNet/ResNet-50 Training in 224 Seconds.

[4]] Leon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta,editors, Proceedings of COMPSTAT'2010, pages 177–186, Heidelberg, 2010. Physica-Verlag HD.

[5]] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In Advances in Neural Information Processing Systems, pages 19–27, 2014.

[6]] Leslie G Valiant. A bridging model for parallel computation. Communications of the ACM, 33(8):103–111, 1990.

[7]] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. On model parallelization and scheduling strategies for distributed machine learning. In Advances in neural information processing systems, pages 2834–2842, 2014

[8]]Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. arXiv preprint arXiv:1404.5997, 2014.

[9]]An Xu et al. On the Acceleration of Deep Learning Model Parallelism with Staleness, 2022

[10]]Chi-Chung Chen et al. Efficient and robust parallel DNN training through model parallelism on multi- GPU platform, 2019.

[11]]Lei Guan et al. XPipe: Efficient Pipeline Model Parallelism for Multi-GPU DNN Training, 2020

[12] Chahal, Karanbir Singh, et al. "A hitchhiker's guide on distributed training of deep neural networks." Journal of Parallel and Distributed Computing 137 (2020): 65-76.

[13] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.