TUNISIAN REUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and Technology

END OF YEAR PROJECT

Specialty: Networks and Telecommunications

Deep learning parallelization by neural networks for biological and medical analysis

Realized by:

Sarah BEN ABDALLAH
Mahdi BEN ZINOUBA
Mohamed Yassine ELKHADIRI

JURY

Reviewer: Dorsaf SEBAI

Supervisor: Sofiene OUNI

School year: 2021 / 2022

Acknowledgments

This work could not have been done without the precious contributions of many people, who we hope will find our gratitude through these lines.

Our thanks are addressed first of all to Mr. Sofiane OUNI, who supervised us throughout the project. It is thanks to his direction, his precious advice and his patience that we could finalize this project.

Special thanks go to Mr. Raed OUERFELLI who provided us with a Google Cloud account and to Mr. Hichem BEN ZINOUBA who bought a LambdaLab instance for us.

Finally, we would like to thank all those who have followed the tribulations that occurred during this project, to our families and friends.

May the members of the jury find our sincere gratitudes for the honor they give us by accepting to validate this work.

Abstract

Biological and medical databases have considerably grown since many years. A base request that only took a few minutes can now take days. At the same time, the bioinformatics research community has developed, and specialized laboratories have been established around the world. Using many data science methods to study these databases, training Convolutional Neural Networks (CNN) has been one of the most computationally intensive task whose parallelization has become critical in order to complete the training in an acceptable time.
However, there are two obstacles to developing a scalable parallel CNN in a distributed-memory computing environment. One is the high degree of data dependency exhibited in the model parameters across every two adjacent mini-batches and the other is the large amount of data to be transferred across the communication channel. Here, we present a parallelization strategy that maximizes the overlap of inter-process communication with the computation.
The overlapping is achieved by using a thread per compute node to initiate communication after the gradients are available. The output data of backpropagation stage is generated at each model layer, and the communication for the data can run concurrently with the computation of other layers.
To study the effectiveness of the overlapping and its impact on the scalability, we studied various model architectures and distributed training methods.

Contents

List of Figures

List of Tables

General Introduction

The exponential technological progress is radically changing human's daily life and attitudes. From smart applications that count daily steps or scan a QR Codes, to virtual assistants like Siri that can understand most of the commands a person can address to her. These daily actions that now represent our routine were once considered as science-fiction ideas. This remarkable progress is obtained thanks to decades of research and development. It is also important to note that Artificial Intelligence (AI) had played a big role in this advancement . Nowadays, with the huge amount of data, that can be collected easily, and the growth of new data types such as 3D images, videos and even text sequences, humans are in urgent need for complex algorithms to learn hidden patterns in this data. That is the problem that Deep Learning came to solve.

Deep learning was at first inspired by human brain neurons considering their structure and how messages are passing between them. That is why the fundamental structure of deep learning models is named Neural Networks. The birth of this approach has encouraged the researchers to develop new state-of-the-art models and architectures. It was also the apparition of new needs for AI in a lot of fields such as agriculture, weather prediction, medicine and marketing that made researchers invent new approaches to maximize the accuracy of these models. Images are the key factors of this progress since they represent a crucial node to which thousands of applications can be attached. Today, different types of data are present everywhere. It is used by social media recommendation systems, video surveillance software, medical analysis .. etc. Thus, it is no surprise that image classification and object detection one of the main focuses of deep learning. As a result, benchmark datasets such as MNIST [24] and CIFAR-10 [7] were created to encourage research in this field. There are also various approaches that have been developed for classification problems. One of the most important ones is Convolutional Neural Networks (CNNs) that outperformed all the previous architectures.

This project analyzes the topic of disease prediction in a distributed system. With data processed by the biomedical and medical communities, scrutiny of medical data is useful for early disease detection, patient care, and community services. The proposed system provides a deep learning algorithm to effectively predict the prevalence of Lymphocytic Leukemia.

This report is organized as follows: The first chapter aims to introduce the clinical context of our project, namely biological definitions of gene terms. We then present the objectives and motivation of this project. After that, we introduce by the second chapter the artificial neural networks as well as convolutional neural networks. In the third chapter, we draw up a state of the art of distributed training approaches used for data and model parallelism. Finally, we study the difference between our model and other benchmark models in distributed systems.

1

Background

Introduction

In this chapter, we first present the clinical context of our project, namely the scientific definitions necessary for the study of genetics and their methods of acquisition. Finally, we present the objectives and motivations of this project.

1 Definitions

1.1 DNA

DNA, which stands for Deoxyribonucleic Acid, is the molecule that contains the genetic code of an individual. This genetic information is present in every cell of the human body. The genetic code corresponds to a succession of 4 ACGT nucleotides: adenosine, cytidine, guanosine and thymidine. For each combination, there will be a correspondence: each triplet corresponds to an amino acid, the amino acids being what constitutes the proteins. DNA is a double strand that forms a double helix. Depending on the species, its organizational structure varies. DNA is the molecule that carries the genetic code of an individual, i.e. all the information necessary for the manufacture and development of a living being.

1.2 Genome

The word genome is a combination of the words gene and chromosome. The genome contains all the genetic information of an organism contained in each of its cells in the

form of chromosomes. The material support of the genome is DNA, except for some viruses where it is RNA.

1.3 Gene

A fragment of DNA containing all the information necessary to produce an RNA or, more often, a protein. A gene corresponds to an instruction to be carried out by the cell.

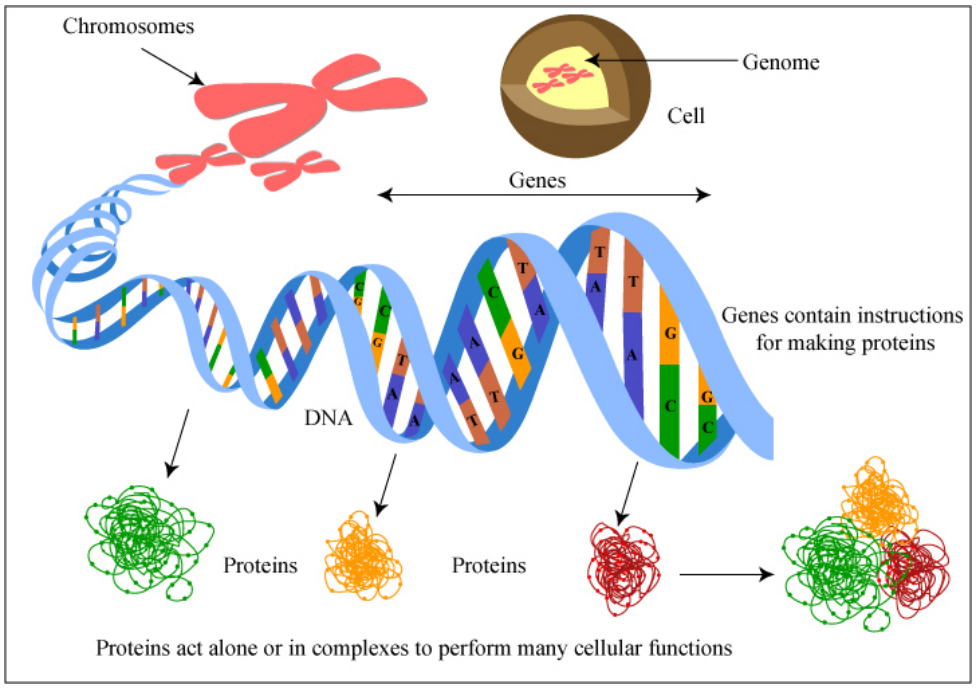


Figure 1.1: From genes to DNA. [18]

2 Method of acquisition

The study of DNA sequences is done by the principle of DNA sequencing. The latter is a method whose aim is to determine the linear succession of the A, C, G and T bases that make up the structure of DNA. The reading of this sequence makes it possible to study the biological information it contains. Due to the uniqueness and specificity of the DNA structure in each individual, the DNA sequence allows many applications in the field of medicine, such as diagnosis, genetic studies, paternity studies, criminology, understanding of physiopathological mechanisms, synthesis of drugs, epidemiological investigations. Science is now able to know the composition of the DNA of each species by detailing both its nucleotide sequence (the DNA that makes up the genome)and the expression of associated genes (the RNA that makes up the transcriptome). This knowledge has many applications, such as the analysis of the evolution of species, reproductive medicine or forensic medicine. To obtain the sequencing of a genome, it is first necessary to take a sample of blood, tissue, hair or saliva which will contain DNA. We will then isolate this DNA from the tissues using detergents and organic solvents, followed by a decantation

or centrifugation phase. Since a whole genome represents several billion nucleotides, it is necessary to combine a biological approach with bioinformatics to sequence it. The first step is to cut the DNA into fragments using restriction enzymes. These short fragments are then sequenced in a random manner and bioinformatics tools are then used to assemble the sequences in the right order by identifying the common sequences and thus knowing the complete nucleotide sequence that makes up our genome.

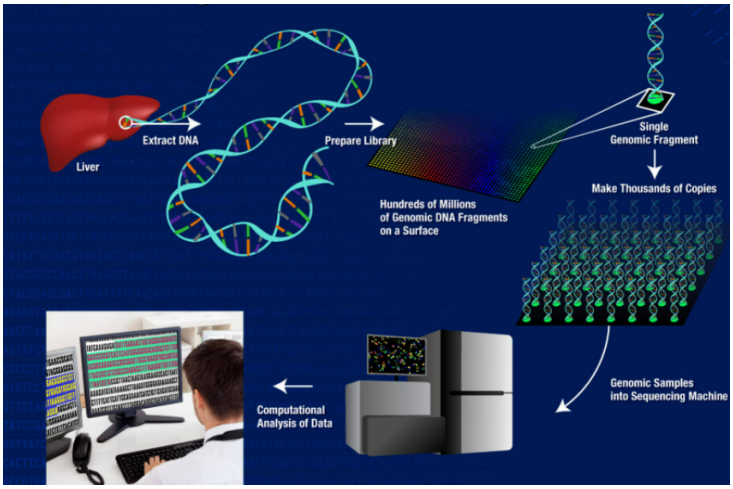


Figure 1.2: DNA Sequencing. [9]

3 Objectives and motivation

With the increase in genetic diseases, the study of multiple DNA sequences is becoming a major problem in terms of research time for physicians. Several approaches to segmentation and classification of anomalies based on gene sequence analysis have been proposed in the literature. With the development of new technologies, the evolution of the field of artificial intelligence, the availability of large databases and the increase in computing and storage power, new and more efficient approaches have appeared, especially in the medical field. To facilitate this study, these advanced technologies have greatly facilitated the proliferation of available sequence data and thus allowed machine learning to find its place in genomics projects. The composition of DNA guides the behavior of genes, and genes govern our appearance, our health, and more generally who we are. In most genomic application areas of machine learning, deep learning outperforms other machine learning procedures (SVM, RandomForest). In the last few years, we can see the exponential progress of deep learning approaches. This growth is supported by the leap in computing power that allows the process of information and by the growing community around artificial intelligence. Moreover, it gains in efficiency by processing large volumes of data, so its use in genomics problems seems particularly relevant. Our end-of-year project is part of this framework and therefore consists in exploring these new approaches, based on the parallelization of neural networks and deep learning, for the diagnosis of DNA sequences and the detection of diseases.

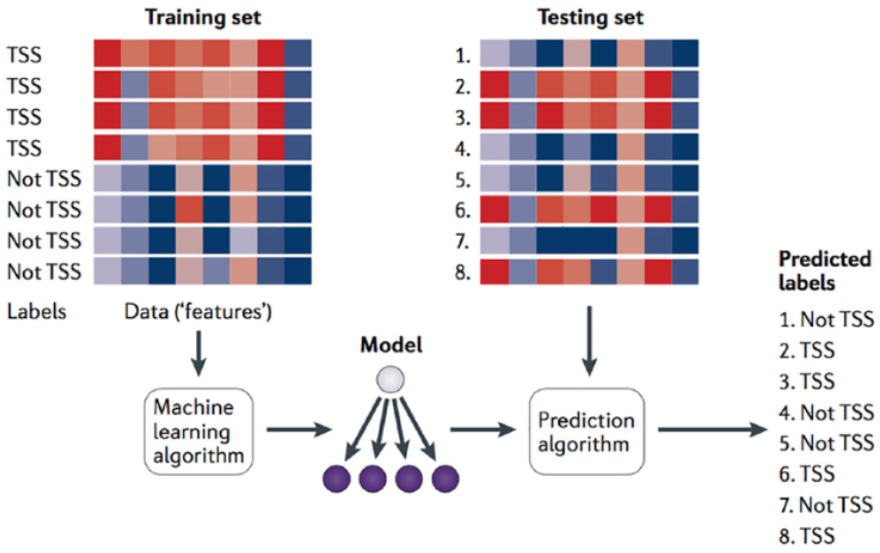


Figure 1.3: Example of "Toxic Shock Syndrome" detection. [15]

Conclusion

The technical ability to analyze thousands of genes by high-throughput sequencing has far exceeded our ability to interpret the data in a clinically meaningful way. In the next chapter, we will discuss the technologies used to get the most insights of this data.

2

Deep Learning methods

Introduction

Machine Learning (ML) techniques have shown significant performance in various medical and biological applications compared to traditional computer-aided systems for diagnosis. While these traditional techniques involve a manual step for feature extraction, which is sometimes very difficult and requires the intervention of an expert, Machine Learning is ideal for exploiting the hidden opportunities of Big Data. This technology makes it possible to extract value from massive and varied data sources without the need to rely on a human. Deep Learning (DL) models also automatically develop a learning process in an adaptive way and can extract features from input data taking into account the target output.
During this chapter, we will present the ANN (Artificial Neural Networks) and CNN (Convolutional Neural Network), their principles and functionalities.

1 ANN (Artificial Neural Networks)

1.1 Presentation

An artificial neural network is a system whose design was originally schematically inspired by the functioning of biological neurons, this modeling being to reproduce the capacities of the human brain to classify or make decisions, and which subsequently become closer to statistical methods. They are particularly applied to solve classification, prediction, categorization, optimization or pattern recognition problems.
A formal neuron, like a biological neuron, receives several stimuli via weights. It analyzes this information and provides a result by following:

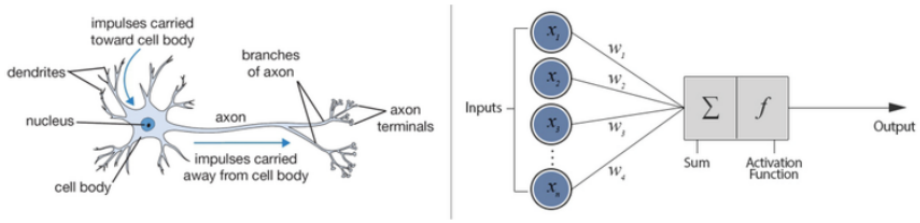


Figure 2.1: Biological Neuron versus Artificial Neural Network. [14]

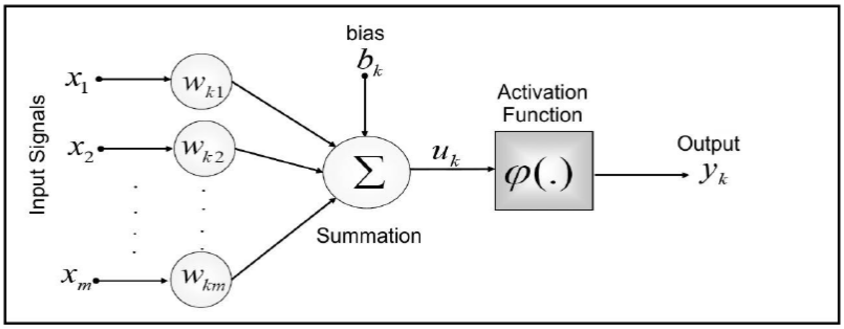


Figure 2.2: Artificial neural network. [13]

- Weights:

Each weight has a noted value Wij. This notation, the most widespread in the scientific literature, designates the weight going from a formal neuron i to the formal neuron j. Each weight transmits an information coming from the source neuron i denoted. This stimulus (its value) corresponding to the information sent by source neuron i is modulated by the weight linking neurons i and j. Mathematically, this translates to the following equation:

$$Wij * xi \tag{2.1}$$

- Combination functions:

Thus neuron j receives as many stimuli as weight, of which it makes the sum would be:

$$\sum_{i=0}^{n} Wij * xi \tag{2.2}$$

This expression then reads as follows: "the sum of all the multiplications of the values of the n source neurons by the weights associating these source neurons to the neuron j considered" (i taking the values: 0, 1, 2, ... , n).
It is this sum that the formal neuron j must then process. It uses the transfer function for this.

- Activation function

In the field of neural networks, this function can also bear the name of combination function, or thresholding function or else activation function. Biologically, the idea of

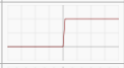
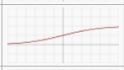
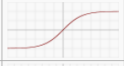
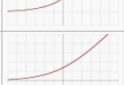
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 2.3: Example of activation functions. [16]

an activation function comes from the idea of mimicking the functioning of an action potential of a biological neuron: if all the input stimuli of a neuron reach its threshold of excitability, then this neuron provides an output.
Several mathematical functions can be used as shown in Figure 2.3.

1.2 Principle

A neural network works by:

1. Initialize the weights with random values, which are mostly between 0 and 1.
2. Calculate the output to calculate the loss or error term.
3. Next, adjust the weights to minimize the loss.

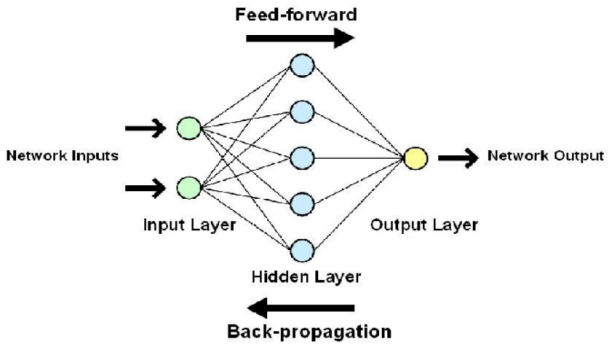


Figure 2.4: How a neural network works. [1]

We repeat these three steps until we reach the optimal solution of the minimum loss function or exhaust the predefined epochs (i.e. number of iterations). Now we can neither modify the input variables nor the actual Y values, but we can modify the other factors. The activation function and optimizers are the tuning parameters, and we can change them according to our needs.
The weights are updated using the gradient descent algorithm with the following equation:

$$Wnew = Wold(\alpha * \frac{dE}{dW}) \tag{2.3}$$

where:

- Wnew: The new weight of Xi.
- Wold: The old weight of Xi.
- α: The learning rate. It is a hyper-parameter between 0 and 1, it presents the speed of the response of the network to the changes it undergoes. It controls how much we adjust our network weights based on gradient descent as shown in Figure 2.5.

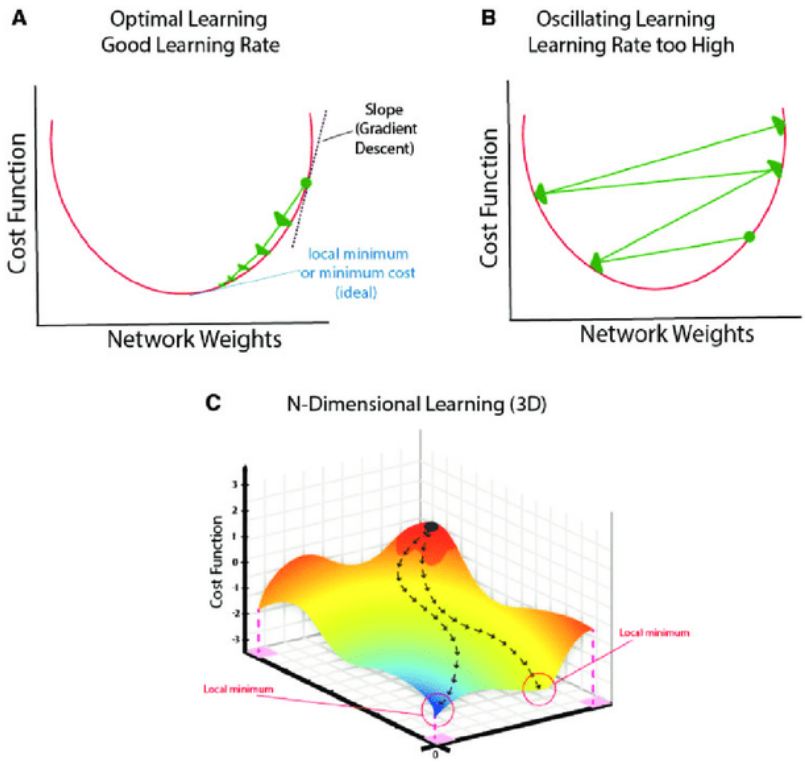


Figure 2.5: Functionning of the gradient descent. [10]

- E: Error function=

$$\frac{1}{2} * [Desired - Predicted]^2 \tag{2.4}$$

- $\frac{dE}{dW}$ is the partial derivative of the error for each of the X's. It is the rate of change of the error with respect to the change in weight.

In the first iteration, we randomly initialize the weights. In the second iteration, we modify the weights of the hidden layer closest to the output layer. In this case, we go from the output layer, hidden layer, then to the input layer.
The back-propagation algorithm is used to update the weights when they are not able to make the corrected predictions us shown in Figure 2.4.

2 CNN (Convolutional Neural Network)

2.1 Presentation

CNNs are neural networks that use convolution instead of matrix multiplication in at least one of their layers. This type of neural network is specialized in processing data having a grid topology.
CNNs have succeeded in becoming the most powerful tool for computer vision use cases, including video and image recognition. But CNN applications have not stopped there, they have also been used to develop recommender systems and solve certain natural language processing (NLP) problems.
The structuring of a CNN consists of a sequence of 3 main types of layers: convolutional, pooling and fully connected layer, in addition to input and output layers.
CNN draws its strength from the convolution layers to learn essential features on its own. This is done by sharing parameters (weights) and by local connectivity. The latter represents the fact that in each layer, each output value depends on only a small number of inputs. This prevents overfitting during network training and keeps the neural network size significantly small without affecting accuracy at the same time.

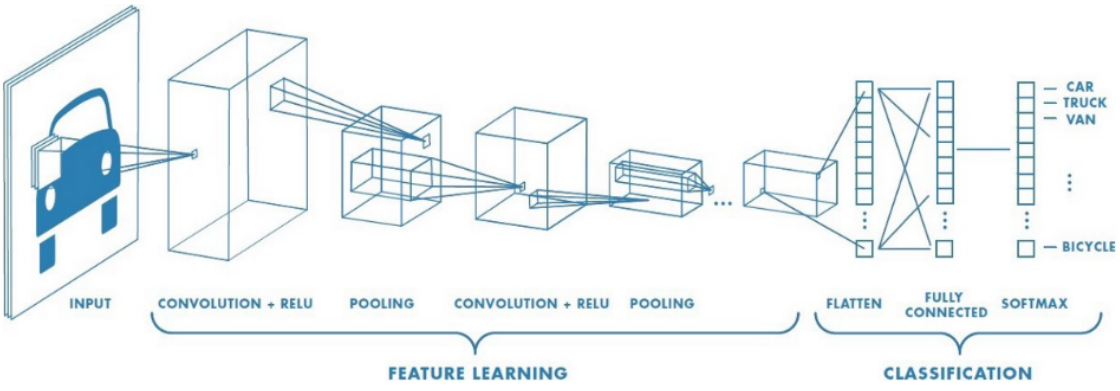


Figure 2.6: Example of a convolutional neural network. [17]

2.2 Principle

2.2.1 Filters

The upper layer is perceived as the mathematical layer. This is basically the convolutional layer and deals with understanding the number pattern it sees.
The filter is also called neuron or nucleus. It reads that part of the data and forms a conclusion of an array of numbers, multiplies the array and deduces a single number from this process.
With this process, we use methods to keep the original form of the data and not to lose information such as MaxPooling, AvgPooling, Zero-Padding ..

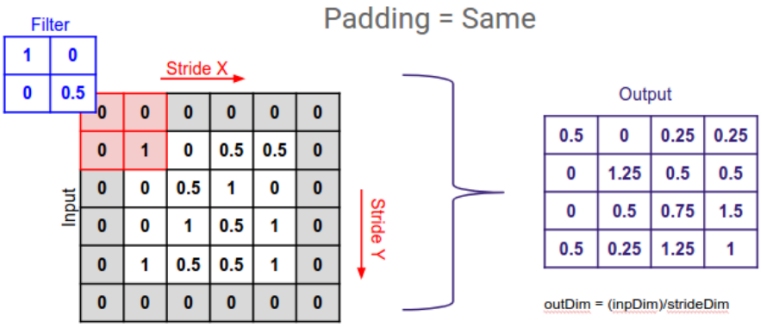


Figure 2.7: Zero-Padding with AvgPooling. [22]

2.2.2 Activation functions

In order to introduce non-linearity in the model, activation functions are applied. Since it makes the model able to learn complex functions between inputs and output variables. That's why, we apply an enable function after extracting the feature map, we usually use ReLU because of its simplicity.

2.2.3 Fully connected layers

As with any finished product, it is necessary to have a final layer encompassing all the interior complexities. This layer is the completion layer in a convolutional neural network. It takes the final output of the layer before it, whether it's a reread layer or a convolutional layer, and provides an N-dimensional vector output, N here means the number of classes from which the program chooses. In Figure 2.8, an example of a fully connected layer of CNN model.

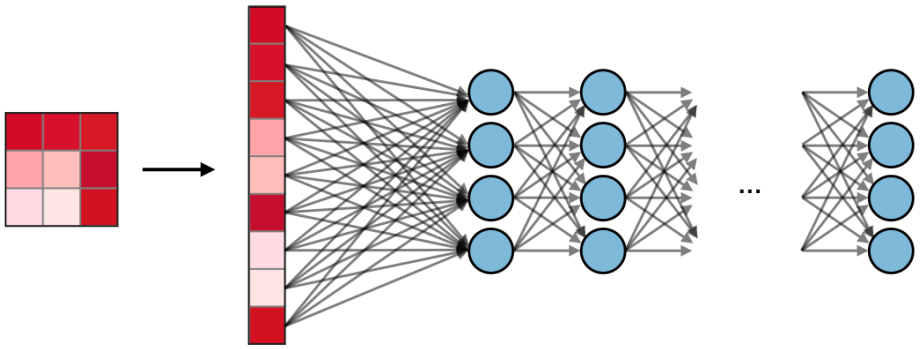


Figure 2.8: Fully connected layers in CNN. [6]

Conclusion

In this chapter we presented the ability of deep learning to process large volume data and the core difference between ANN and CNN , However as datasets size increases, it becomes very hard to train models within a limited time frame.In the next section, we will dive deeper into how parallel processing and distributed system can solve this problem.

3

On designing Parallel Processing and Distributed Training Systems

Introduction

Training a deep learning model is a computationally intensive task that needs to be completed in a reasonable amount of time. As the dataset size increases, it becomes very difficult to train a model within a limited time frame. To address this type of problem, distributed training methods are used. Distributed training allows us to train very large models and reduce training time. Parallelization of DNNs is considered a challenge due to the inherent sequential nature of stochastic gradients (SGD).
During this chapter, we will study the different types of parallelism and the difference between them. Then we will discuss the types of distributed strategies and their conceptions.

1 Types of parallelism

1.1 Data Parallelism

With data parallelism, the dataset is divided into N parts, where N is the number of GPUs. These parts are then distributed to parallel computers. After that, gradients are computed for each copy of the template, and then all templates exchange gradients. Finally, the values of these gradients are averaged.
Each GPU or node uses the same forward pass settings. A small batch of data is sent to each node, and gradients are calculated and sent back to the master node as usual.
Each node independently computes the error between its predictions for training samples

and the labelled output. Each node in turn updates its model based on the error, and must communicate all its changes to the other nodes to update their respective models. This means that worker nodes need to synchronize model parameters or gradients at the end of a batch computation to ensure that they train a consistent model.

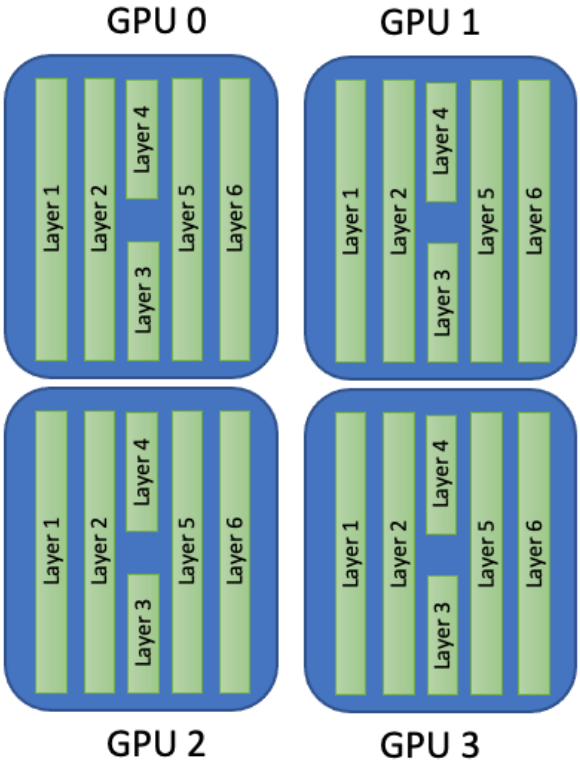


Figure 3.1: Data Parallelism. [19]

There are two strategies associated with distributed data parallelism, called synchronous and asynchronous:

1.1.1 Synchronous training

This strategy is supported by the All-Reduce architecture. All devices (GPUs) train synchronously on different input data slices, aggregating gradients at each step. That means, There is a root aggregator node in synchronous training. This node divides data into batches and forwards requests to other workers to process each batch and compute its gradient independently.
Once all machines have returned their results, the root aggregator node averages the gradients and sends them back to the workers to update the model's parameters. The root node repeats all this process for a specified number of epochs or based on predefined conditions.

1.1.2 Asynchronous training

This strategy is backed by parameter server architecture. All workers independently train on input data and update variables asynchronously. The worker reads all shared model parameters in parallel. A worker may see partial updates from one or more other workers. The worker computes the gradient locally based on a batch of input data and the parameter values, then workers send the gradients for each variable to the appropriate Server aggregator and apply the gradients to the respective variables using update rules determined by the optimization algorithm such as Adam, SGD and SGD with Momentum.

1.1.3 Synchronous vs Asynchronous

Distributed training is becoming more common in the MLOps ecosystem. Especially when the device is smaller, less reliable and more limited, researches prove that asynchronous training might be a better option while synchronous drives are more suitable if the device is more powerful and has strong connectivity.

1.2 Model Parallelism

With model parallelism, each model is divided into N parts, just like data parallelism, where N is the number of GPUs.
Each model is then placed on a single GPU. The GPU stack is then computed sequentially in this manner, starting with GPU 0, GPU 1, and ending with GPU N. This is forward propagation. Backpropagation on the other end starts at GPU N behind and ends at GPU 0.
Model parallelism has clear benefits. It can be used to train models so that they do not fit on a single GPU. But when the computations are done sequentially, like when GPU 1 is doing computations, the others just sit idle. This can be solved by switching to asynchronous GPU operation.
There are several running mini-batches in the pipeline. First, the initial mini-batch updates the weights, and subsequent mini-batches in the pipeline inherit the stale weights to derive gradients. With model parallelism, obsolescence can lead to model instability and poor accuracy.
The scalability of this method depends on the degree of task parallelization of the algorithm, and it is more complex to implement than data parallelism.

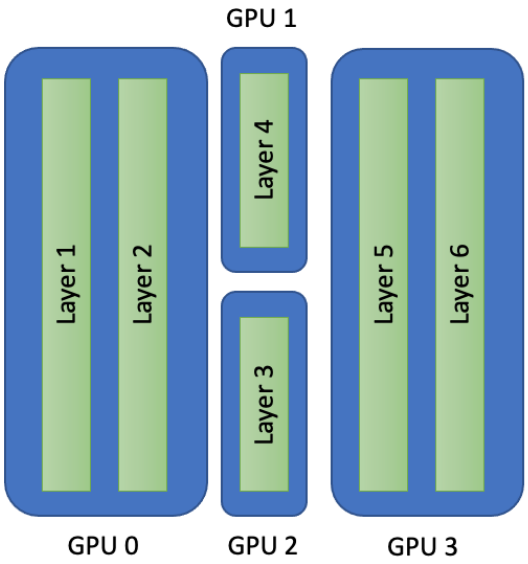


Figure 3.2: Model Parallelism [19].

1.3 Data Parallelism VS Model Parallelism

A study titled "Efficient and Robust Parallel DNN Training Using Model Parallelism as a Multi-GPU Platform" compared model parallelism with data parallelism, showing that the accuracy of models using data parallelism improves over time. As training progresses, the accuracy starts to vary with the parallelism of the model.
The study also shows that data parallelism should be a scalability issue. As more GPUs are added, model parallelism seems to be more suitable for DNN models. In a recent major case, Google AI's large-scale PaLM or Pathways language model used a combination of data and model parallelism as part of its cutting- edge training.
The model scales to two Cloud TPU v4 pods using pod-level data parallelism, each using model parallelism with standard data [5].

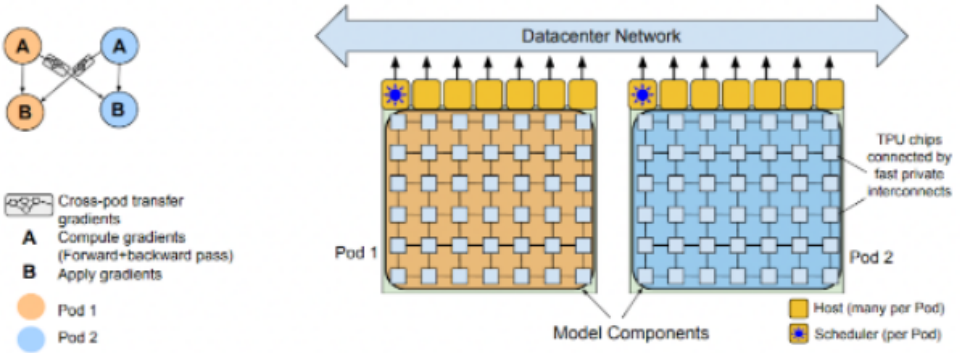


Figure 3.3: The pathway system scales training across two TPU v4 pods using two-way data parallelism at the pod level.

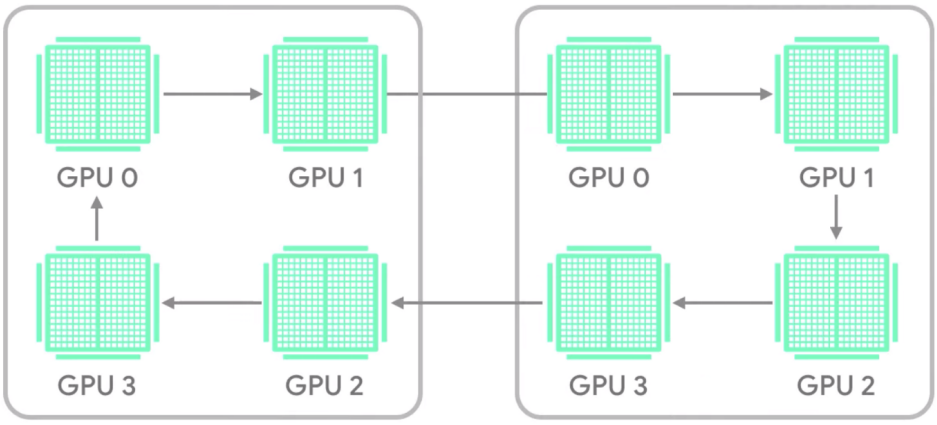
2 Types of of distributed strategies

2.1 Mirrored Strategy

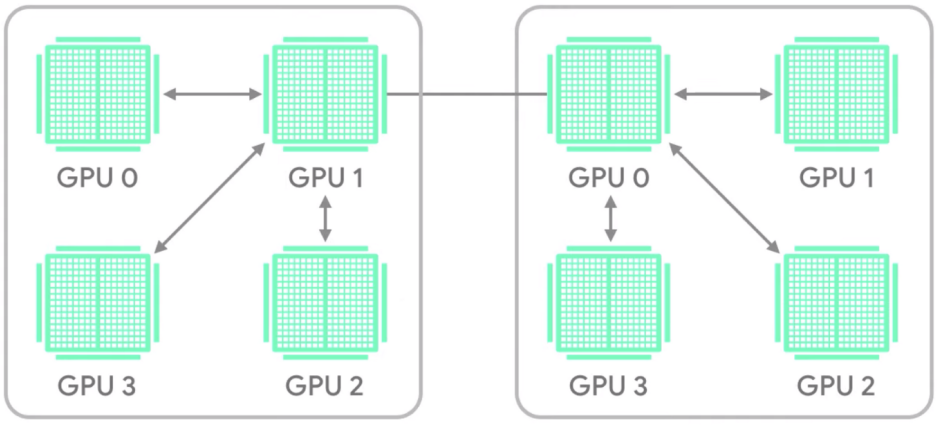
This strategy runs on one machine/worker and multiple GPUs. The model is replicated on all GPUs. Each model is trained on a different data slice and updates the weights using an efficient cross-device communication algorithm which are all-reduce algorithms. The model is trained on multiple machines with multiple GPUs. Workers run synchronously, synchronizing gradients at each step. The Communication between devices uses ring-all-reduce or hierarchical-all-reduce communication.

2.2 Multiworker Mirrored Strategy

The model is trained on multiple machines with multiple GPUs. Workers run synchronously, synchronizing gradients at each step. Communication between devices uses ring-all-reduce or hierarchical-all-reduce communication.



(a)



(b)

Figure 3.4: Multiworker mirrored strategy : (a) ring all-reduce between two workers and (b) Hierarchical all-reduce strategy between two workers.

As shown in Figure 3.5, all machines (workers) are connected to each other. In this case, instead of having a single machine to perform the aggregation task, we can distribute the aggregation task on all machines. In the first step, all machines start with their own copies of all parameters, then machine in each step every machine sends its parameters to the other machines. At the end, each machine has the full version of a subset of parameters and aggregates them.

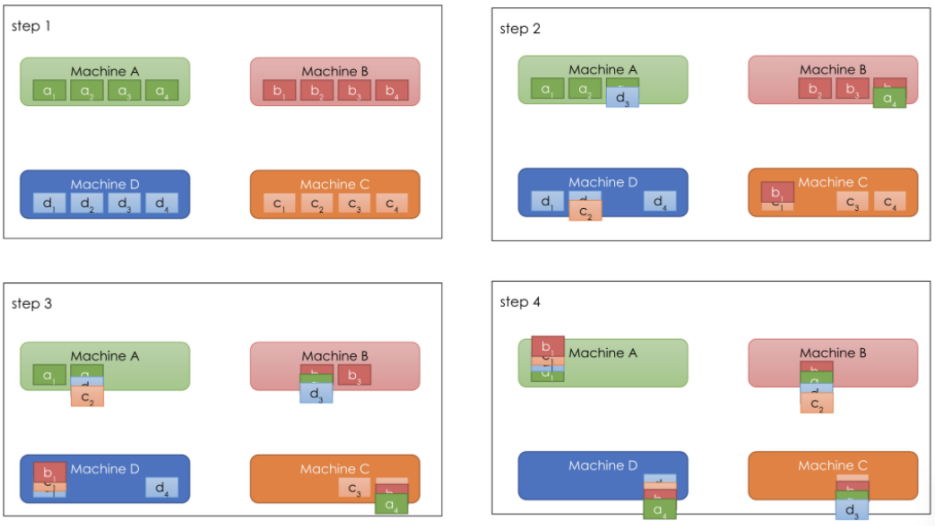


Figure 3.5: All-reduce architecture between four machines [11].

2.3 TPU strategy

TPU, stands for Tensor Processing Unit, is an AI accelerator application-specific integrated circuit (ASIC) developed by Google specifically for neural network machine learning, particularly using Google's own TensorFlow software.

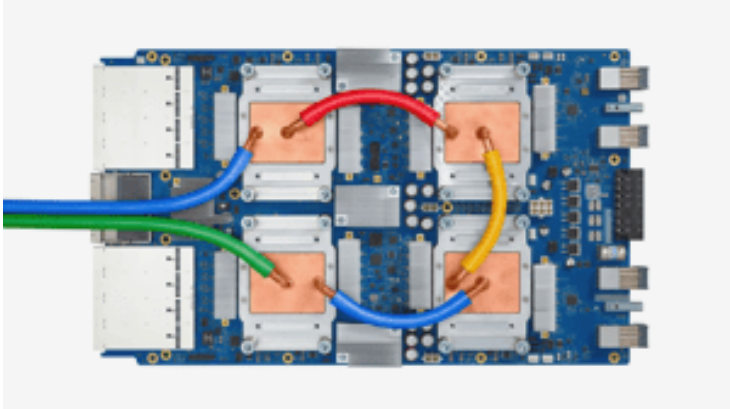


Figure 3.6: Tensor Processing Unit [4].

TPU strategy is the same as mirrored strategy, the only difference is that it runs on TPUs instead of GPUs.

2.4 Parameter Server Strategy

This strategy is enforced on multiple machines. In this setup, some machines are called workers, while others are called parameter servers. Each variable of the model is stored on the parameter server.
Computations are replicated on all GPUs across all workers. Work tasks read inputs, variables, compute forward and backward, and send updates to parameter servers.

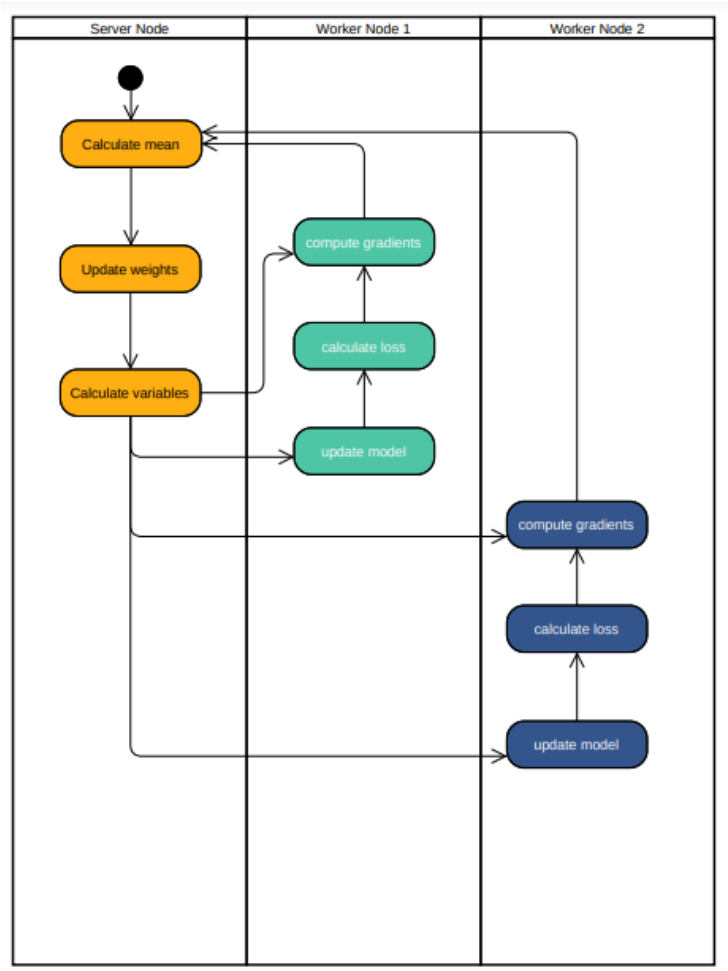


Figure 3.7: Activity Diagram of parameter server strategy between two workers.

Conclusion

In conclusion, we constat that in both data parallelism and model parallelism training time improvement is guaranteed. It is only a matter of choosing whether to use model or data parallelism and which distributed strategies to use based on hardware availability (one or more machines that contains one or multiple GPUS) and importance of scalability versus accuracy. In the next chapter, we will test data parallelism using benchmark datasets and our dataset

4

Implementation and results

Introduction

In this chapter, we are going to tackle DNA classification from CNN approach. We will first present the experimental setup than we will go through the obtained results and finally we will compare our modal with other benchmark dataset through distributed training algorithms.

1 Environment

The success of the field of artificial intelligence in general and Deep Learning in particular is due to the progress of machine learning libraries that are accessible and easy to import. Several frameworks are used to implement neural networks, and most of them support the Python language [3]. Among the best known are TensorFlow, Keras, Lasagne and PyTorch. Thanks to the libraries mentioned, the use of the GPUs and TPUs is now necessary to reduce calculation times. In our project, we are mainly interested in the keras and TensorFlow frameworks.

- Google Colaboratory [23]: It is a tool created by Google. It is free and easy to use, designed to learn about Deep Learning or to collaborate in a team on a data science project. The feature that distinguishes Google Colab from other services is the access to a to a GPU and a TPU, which they are provided for free with a runtime of up to 12 hours.
- Tensorflow [21]:
  Tensorflow is a dedicated programming framework for numerical computation. It has been made open source by Google in November 2015. Since its launch, TensorFlow has

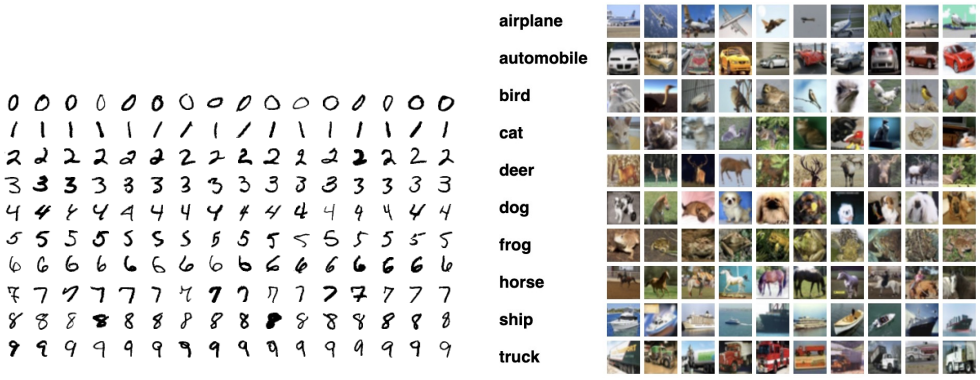
continued to gain popularity and has quickly become one of the most used frameworks.

- Tensorboard:
  Tensorboard provides the visualization solutions and tools needed for machine learning testing like tracking and visualization of metrics such as loss and correctness, visualization of the model graph (operations and layers), display of histograms of weights, biases or other Tensors as they evolve, projection of continuous vector representations into a lower dimensional space, display images, text and audio dataâ¦
- Keras [20]:
  Keras is a high-level neural network API, written in Python and able to run on TensorFlow or Theano. Its development is focused on rapid experimentation. In 2017, Google's TensorFlow team decided to support Keras in the main TensorFlow library. Its founder Chollet explained that Keras was designed as an interface rather than a learning environment. It provides a more advanced and intuitive set of abstractions, and can easily configure a neural network independently of the computer library.
- Lambdalabs [2]:
  Instant access to pay-by-the-second GPU instances. It is a GPU Cloud pre-installed with PyTorch TensorFlow. Lambda provides computation to accelerate human progress. Its customers include Microsoft, MIT, Los Alamos National Lab, Disney, Tencent, Kaiser Permanente, Stanford, Harvard, Caltech, and the Department of Defense.

2 Training Model

2.1 Benchmark datasets

In this section, we will present the benchmark datasets we worked on. In fact, since our goal is to identify on which type of images CNNs, we aimed to choose MNIST firstly as it is one of the most basic and beginner friendly dataset for image classification, since the images are simple and grayscale. CIFAR-10 dataset which contains colored objects.



(a) (b)

Figure 4.1: Benchmark Datasets : (a) MNIST. (b) CIFAR.

2.2 Dataset

We are going to use simulated data of 10,000 500 bp long sequences, with the positive set containing an instance of the TAL1 motif and the negative set will be random sequences. TAL1 is a T-cell acute lymphocytic leukemia protein.

Figure 4.2: TLA1 motif

The data were simulated using simDNA by Johnny Israeli and were deposited to The dragonn package [12] implements Deep RegulAtory GenOmic Neural Networks (DragoNNs) for predictive modeling of regulatory genomics, nucleotide-resolution feature discovery, and simulations for systematic development and benchmarking. The data is divided into training dataset, on which the model will be fitted, and a test dataset on which its performance will be assessed.
Each of the train and test datasets are splitted into instance column (X) which contains the DNA sequences and class column (Y) which contains True or False response.

Table 4.1: Dataset distribution

Dataset Parts	**Shape**
X train	(12800, 500, 4)
Y train	(12800, 1)
X test	(3200, 500, 4)
Y test	(3200, 1)

As you can see, there are 12.8k training examples and 3.2k test examples. Each training sample is composed of 500 DNA character (A, C, G or T). By calculating the mean of Y train and Y test we note the following results:

Table 4.2: Dataset Statistics

Class	**Mean**
Y train	0.501953125
Y test	0.5071875

As the means are close to 0.5, there are balanced classes, i.e. roughly the same number of positive and negative instances. This ratio is the same for training and testing. This is an ideal situation for training a classifier.
To visualize the train data, we one hot encoded sequences to DNA strings by the One Hot Encoder of the pysster package [8] which is a Python package for training and interpretation of convolutional neural networks on biological sequence data.

```
[ ]  df_train.head()
```

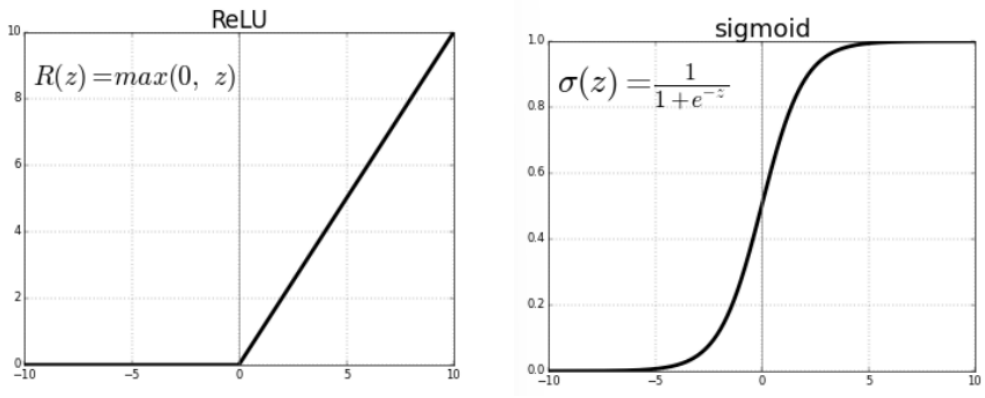
	0	0
0	CGTTATGAGTCATGACTGTTATTTTTACCCAAAACAGTGTCTAGCC...	False
1	TTGTATCCTTTTTTAGTACACTACGGATATGCTGTAATTTCCTGCA...	False
2	CGAAGTGGTCGAAGCTAATCGTCGGTATATATTAAAAACTGCTTAA...	True
3	TCCATCTGACTCAGTAGAAGTTGCTAGCACTAACAGGAAAACGATG...	True
4	CCAGACATGGGCCGTGAATCTTTTCAGTGTTTTGAATGGAGGAGAC...	True

Figure 4.3: Train subset

2.3 Model

2.3.1 Model definition

We will now implement a neural network consisting of 16 convolutional filters whose outputs are transformed by the ReLU activation function.On each of the 16 vectors, we keep the maximal value ("Max Pooling") giving a us 16-long vector. We then perform a linear transformation of this 16-long vector into a single scalar ("Dense layer") which we map to the [0,1] interval with sigmoid activation function.



(a) (b)

Figure 4.4: Relu vs Sigmoid functions: (a) ReLu. (b) Sigmoid.

The interpretation of the output is the probability that the sequence is bound by the transcription factor TAL1.

2.3.2 Model optimization

In order to optimize the performance of the model and to ensure its convergence in a reasonable time, we had to modify some parameters. In our case we use the adam optimizer. In machine learning, we use regularization forms to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration. In our case, we use the Early Stopping method which allows to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a holdout validation dataset. The model will stop training if the loss value doesnât decrease after 3 epochs.

2.3.3 Model compilation

Since this is a sequence detection problem, the evaluation criteria of the model must be adapted to this type of method thatâs why, we used the binary crossentropy as a loss function. We also used the accuracy as a metric. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a holdout validation dataset.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 486, 16)           976
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 993
Trainable params: 993
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.5: Model Summary

2.4 Experimental Results

2.4.1 Distributed Training on Benchmark Datasets

We initialize the benchmark dataset parameters as shown in the Table 4.3.
We trained the benchmark datasets on different numbers of GPUs. In this experience, we used the NVIDIA RTX A6000 GRAPHICS CARD 48GB GPU provided on the LambdaLabs instance.
We used as a distributed training method, the Mirrored Strategy algorithm.

Table 4.3: Benchmark datasets parameters initialization.

	Epochs	Batch size	Parameters
CIFAR	50	256	276138
MNIST	25	128	669706

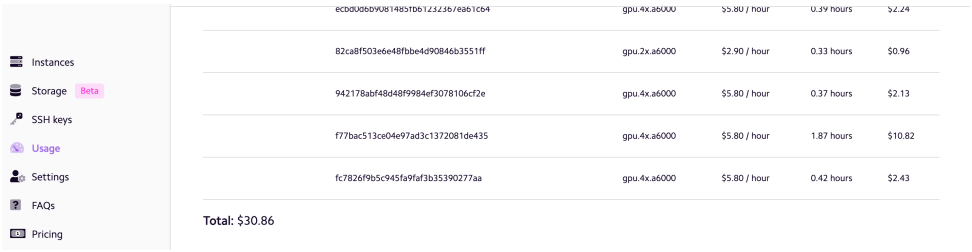


Figure 4.6: Lambda pay-by-the-second GPU instance expenses.

Table 4.4: Training time of benchmark datasets using distributed training on 1 GPU vs 2 GPU vs 4 GPU

	1 GPU	2GPU	4 GPU
CIFAR	4min24s	3min41s	2min51s
MNIST	3min16s	2min30s	1min47s

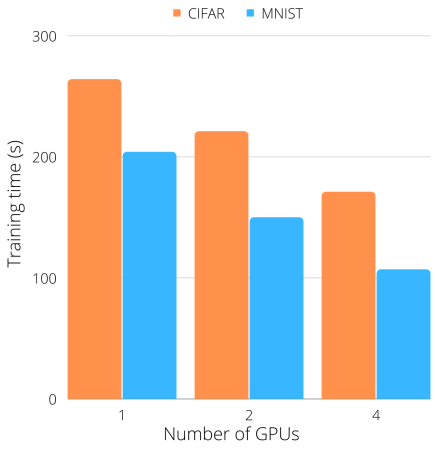


Figure 4.7: Benchmark datasets training time per numbers of GPUs.

2.4.2 Distributed Training on the DNA Dataset

- Training Time

  We used as a distributed training method, the Mirrored Strategy algorithm to compare with the benchmark datasets.

```
In [21]: strategy = tf.distribute.MirroredStrategy(['GPU:0'])
         with strategy.scope():
             model = Sequential([kl.Conv1D(filters=16, kernel_size=15, activation='relu', input_shape=x_train.shape[1:]),
                                 kl.GlobalMaxPooling1D(),
                                 kl.Dense(units=1),
                                 kl.Activation('sigmoid')
                                ])
             model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

         INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0',)
```

(a)

```
In [21]: strategy = tf.distribute.MirroredStrategy(['GPU:0', 'GPU:1'])
         with strategy.scope():
             model = Sequential([kl.Conv1D(filters=16, kernel_size=15, activation='relu', input_shape=x_train.shape[1:]),
                                 kl.GlobalMaxPooling1D(),
                                 kl.Dense(units=1),
                                 kl.Activation('sigmoid')
                                ])
             model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

         INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0', '/job:localhost/
         replica:0/task:0/device:GPU:1')
```

(b)

```
In [21]: strategy = tf.distribute.MirroredStrategy(['GPU:0', 'GPU:1', 'GPU:2', 'GPU:3'])
         with strategy.scope():
             model = Sequential([kl.Conv1D(filters=16, kernel_size=15, activation='relu', input_shape=x_train.shape[1:]),
                                 kl.GlobalMaxPooling1D(),
                                 kl.Dense(units=1),
                                 kl.Activation('sigmoid')
                                ])
             model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

         INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0', '/job:localhost/
         replica:0/task:0/device:GPU:1', '/job:localhost/replica:0/task:0/device:GPU:2', '/job:localhost/replica:0/task:0/devi
         ce:GPU:3')
```

(c)

Figure 4.8: Training the model on different number of GPUs: (a) 1 GPU ,(b) 2 GPUs ,(c) 4 GPUs.

Table 4.5: Training time of the DNA dataset using distributed training on 1 GPU vs 2 GPU vs 4 GPU

	1 GPU	2GPU	4 GPU
Training Time	54,7s	35s	25,9s

- Training results

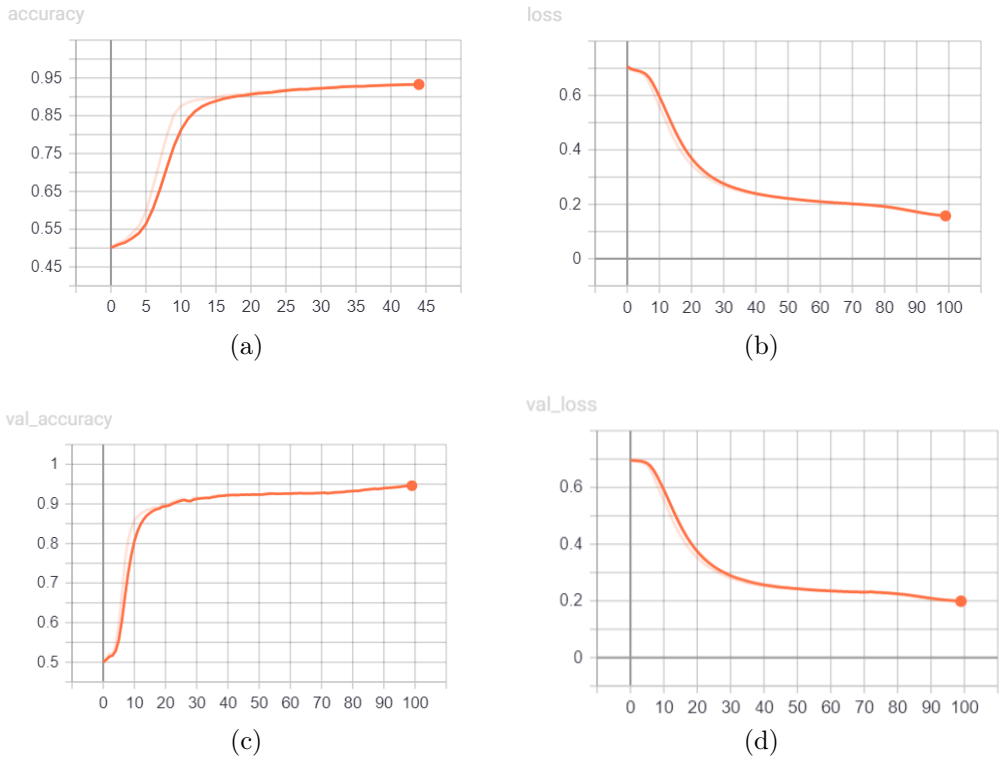


Figure 4.9: Tensorboard results graphs: (a) Train Accuracy.(b) Train Loss. (c) Test Accuracy. (d) Test Loss

2.4.3 Interpretations and Discussion

Multiple GPUs help when the requested computation has already saturated a single GPU however in some cases training on multiple GPUs may even need more time to train.This is due to the fact that we are using a very small batch size , in this case the cost of distributing the gradients/computations over two or four GPUs and fetching them back outweigh the parallel time advantage that you might gain using sequential training.

Conclusion

The results show that the more we distribute our data, we improve the training time. However, to see this fact in action, we need to have a large dataset and a complex model that uses an appropriate batch size in the training phase.

Conclusion and perspectives

As data and data engineering are getting increasingly crucial and as we are shifting to a digital era, we are collecting more data especially in the context of medical research. This evolution makes us wonder if our algorithms are sophisticated or if we have the computational capabilities to handle this gigantic amount of information to put it in good use. Thus, the importance of exploring deep learning architectures like CNN or ANN and more importantly speeding up the process of training models using large dataset and quite complex architecture.
We started by exploring different kinds of distributed learning : data parallelism and model parallelism.The first type consists of splitting the data into several smaller subsets and each subset is injected into a GPU, each unit simultaneously uses the same modal on each different subset. The second type of distributed learning is model parallelism in which we decouple the model into multiple layers where each layer or group of layers are places in a GPU.
Finally, to sum up we have discussed the need to parallelize deep learning models with the ways that can be used to solve the limitation of these traditional models.

Bibliography

[1] Artificial neural networks for machine learning â every aspect you need to know about. https://data-flair.training/blogs/artificial-neural-networks-for-machine-learning/.

[2] Lambda gpu cloud.

[3] Python 3.10.4 documentation. https://docs.python.org/3/download.html.

[4] Tensor processing unit. https://fr.wikipedia.org/wiki/Tensor_Processing_Unit.

[5] Jacob Devlin Noah Fiedel Aakanksha Chowdhery, Sharan Narang. Palm: Scaling language modeling with pathways.

[6] Shervine A Afshine A. Convolutional neural networks sticky notes. https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels.

[7] Vinod Nair Alex Krizhevsky and Geoffrey Hinton. The cifar-10 dataset.

[8] Stefan Budach. Pysster : a sequence structure classifier. https://github.com/budach/pysster.

[9] National human genetics research institute. Dna sequencing fact sheet.

[10] Kipp Johnson. Deep learning for cardiovascularmedicine: A practical primer.

[11] Ju. Distributed model training ii: Parameter server and allreduce. may 20th, 2020.

[12] kundajelab. Dragonn paper supplement contents. https://github.com/kundajelab/dragonn/tree/master/paper_supplement.

[13] Marcelo M Reginaldo M Mauricio V, SÃ©rgio F. Structure of artificial neurons used in the artificial neural network (ann).

[14] Brian Mwandau. Investigating keystroke dynamics as a two-factor biometric security.

[15] Nguyen QV Zhou Y Catchpoole DR January 2019 Qu Z, Lau CW. Visual analytics of genomic and cancer data, a systematic review.

[16] Sagar S. Activation functions in neural networks. Sep 6, 2017.

[17] S. Saha. A comprehensive guide to convolutional neural networks. dec 15, 2018.

[18] Dr. D. Barry Starr. What is the difference between a chromosome, a gene, a protein and dna?

[19] Alexander Statsenko. Model parallelism vs data parallelism in unet speedup.

[20] Keras team. Keras: Deep learning for humans. https://github.com/keras-team/keras.

[21] Tensorflow. Tensorflow documentation. https://github.com/tensorflow/docs.

[22] Yugesh Verma. Guide to different padding methods for cnn models. september 4, 2021.

[23] Nathaniel wendt. Welcome to colaboratory. https://colab.research.google.com/?utm_source=scs-index.

[24] Christopher J.C. Burges Yann LeCun, Corinna Cortes. The mnist database of handwritten digits.