Universidade Estadual de Campinas
Instituto de Computação

Eva Maia Malta

Selecting Efficient Virtual Machines for Training Deep Learning Models on the Cloud

Seleção de Máquinas Virtuais Eficientes para o Treinamento de Modelos de Aprendizado Profundo na Nuvem

CAMPINAS
2021

Eva Maia Malta

Selecting Efficient Virtual Machines for Training Deep Learning Models on the Cloud

Seleção de Máquinas Virtuais Eficientes para o Treinamento de Modelos de Aprendizado Profundo na Nuvem

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestra em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Edson Borin
Co-supervisor/Coorientadora: Sandra Eliza Fontes de Avila

Este exemplar corresponde à versão final da Dissertação defendida por Eva Maia Malta e orientada pelo Edson Borin.

CAMPINAS
2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Malta, Eva Maia, 1993-
M298s Selecting efficient virtual machines for training deep learning models on the cloud / Eva Maia Malta. – Campinas, SP : [s.n.], 2021.

Orientador: Edson Borin.
Coorientador: Sandra Eliza Fontes de Avila.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Computação em nuvem. 2. Aprendizado de máquina. 3. Computação de alto desempenho. I. Borin, Edson, 1979-. II. Avila, Sandra Eliza Fontes de, 1982-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Seleção de máquinas virtuais eficientes para o treinamento de modelos de aprendizado profundo na nuvem
Palavras-chave em inglês:
Cloud computing
Machine learning
High performance computing
Área de concentração: Ciência da Computação
Titulação: Mestra em Ciência da Computação
Banca examinadora:
Edson Borin [Orientador]
João Paulo Papa
Lúcia Maria de Assumpção Drummond
Data de defesa: 10-03-2021
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-2085-122X
- Currículo Lattes do autor: http://lattes.cnpq.br/3183898522744068

Universidade Estadual de Campinas
Instituto de Computação

Eva Maia Malta

Selecting Efficient Virtual Machines for Training Deep Learning Models on the Cloud

Seleção de Máquinas Virtuais Eficientes para o Treinamento de Modelos de Aprendizado Profundo na Nuvem

Banca Examinadora:

- Prof. Dr. Edson Borin
  Universidade Estadual de Campinas
- Prof. Dr. João Paulo Papa
  Universidade Estadual Paulista
- Profa. Dra. Lúcia Maria de Assumpção Drummond
  Universidade Federal Fluminense

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 10 de março de 2021

Acknowledgements

This study was financed by Petrobras, FAPESP (CEPID 2013/08293-7), and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001/PROCAD 2966/2014. E. Borin is partially funded by CNPq (313012/2017-2) and S. Avila is partially funded by Google Research Awards for Latin America 2018, 2019, 2020 and FAPESP (2017/16246-0). The authors also thank the High-Performance Geophysics Lab (HPG), the Center for Computing in Engineering & Sciences (CCES), and the Laboratório Multidisciplinar de Computação de Alto Desempenho (LMCAD) team for technical support.

Resumo

Modelos de Aprendizado Profundo têm sido cada vez mais utilizados para a resolução de problemas complexos. Sua característica de análise hierárquica da informação permite a extração de relações complexas existentes em um conjunto de dados. No entanto, com o aumento da complexidade dos modelos e da quantidade de dados, o treinamento destes modelos tem exigido o uso de sistemas computacionais cada vez mais poderosos e com alto custo de aquisição. A Nuvem Computacional é um modelo de negócios que permite o acesso a diversos tipos de sistemas computacionais, incluindo sistemas de alto desempenho, mediante o pagamento pelo uso, sem que o usuário tenha que arcar com o custo de aquisição do equipamento. Contudo, escolher corretamente o sistema computacional mais adequado para o treinamento de um modelo de Aprendizado Profundo na nuvem é um desafio, pois a escolha deve levar em consideração fatores como tempo de execução e custo, por exemplo. Pensando nisso, este trabalho apresenta um estudo sobre o comportamento do treinamento de modelos de Aprendizado Profundo em máquinas virtuais com GPU na nuvem computacional. Neste estudo, nós observamos que a configuração do *batch size* afeta o tempo de treinamento do modelo e o número de épocas necessárias para que a acurácia do modelo estabilize. Além disso, observamos que os tempos de execução das iterações e dos processos de validação de cada época do treinamento são estáveis, com exceção da primeira iteração e da validação da primeira época. A partir destas observações, propusemos duas metodologias para identificar o tipo de máquina virtual mais adequada para treinar um dado modelo de Aprendizado Profundo na nuvem computacional. Por fim, validamos a acurácia das metodologias propostas com duas aplicações de Aprendizado Profundo distintas e mostramos que, em ambos os casos, as metodologias foram capazes de identificar o tipo de máquina virtual com menor custo e/ou mais rápida para realizar o treinamento.

Abstract

Deep Learning models are a popular tool to solve complex problems. Their analysis based on hierarchical analysis of information allows us to extract complex correlations among data. However, with the rise of models' complexity and amount of data, training these models is requiring powerful computational systems with high acquisition costs. Cloud Computing technology is a business model that allows us to access many computational systems, including high-performance systems, in a pay-per-use model, not requiring the user to pay for the equipment acquisition. Nevertheless, correctly choosing the appropriate computational system to train a Deep Learning model on the cloud is a challenge since the choice must consider variables like the execution time and cost. This work presents a behavior study of training Deep Learning models on virtual machines equipped with GPUs on the cloud. In this study, we observed that the batch size affects the training time and the necessary number of epochs to stabilize the model accuracy. We also observed that each epoch's iterations and validation times are stable, except for the first iteration and first epoch's validation. Based on these observations, we proposed two low-cost methodologies to identify the ideal virtual machine to train a Deep Learning model on cloud virtual machines. Finally, we validated the accuracy of the proposed methodologies with two Deep Learning applications, and we showed that, in both cases, the methods were capable of identifying the virtual machine types that provide the smallest cost and the shortest runtime to train the Deep Learning model.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The constant growth of available data from all different kinds of subjects is allowing companies and the government to make decisions based on complex data analysis insights. Data-driven researches are no longer supporting knowledge but plays a leading role in industry decision-making. The International Data Corporation (IDC) presented a research that suggests that, by around 2025, the world will have 175 ZB of data available, representing twice what we have today [27].

Two components have gained attention in this situation. The first one is the development of new algorithms and strategies to extract complex insights from these data. The second one is the development of new hardware systems to support the processing of such an amount of data.

Regarding the algorithms and strategies, we can mention Machine Learning as one of the main strategy used to acquire information from the data. This strategy has many different algorithms that can learn from data to identify patterns within the data and generalize them for new ones. Among all the Machine Learning algorithms, the Deep Learning approach is more suited to problems that deals with many variables [10].

On the one hand, Oussous *et al.* [25] argues that the hierarchical learning feature of Deep Learning is one of the main factors of why this strategy is so recommended to deal with large datasets since it simplifies the analysis of complex features "into a suitable internal representation or feature vectors". On the other hand, using Deep Learning techniques to learn from large datasets only became possible due to another factor: computational power growth. This factor, combined with code parallelization strategies and new types of hardware, such as Graphic Process Units (GPUs), allows us to train and use Deep Learning models with such an amount of data.

Cloud Computing is a business model that facilitates access to high-performance computational resources. The IDC's report [27] puts Cloud Computing as the new companies data center, shifting from the traditional one. Also, IDC predicts that in 2025 around 49% of the world's data will be in Cloud environments.

There are many advantages of using Cloud Computing infrastructures. We can cite minimizing operational costs for the companies, easy access to different computational resources, and a pay-per-use charging model. However, there are some challenges in the use of Cloud services as well. One of them is identifying the best practices when using Cloud Computing resources, especially when considering the services' cost-benefit

correlation.

For Deep Learning applications, these challenges are not different. The Amazon Web Services (AWS) cloud provider, for example, offers many services to run Deep Learning algorithms. One of them, the Elastic Compute Cloud (EC2), is based on the Infrastructure as a Service (IaaS) model and allows users to instantiate different virtual machines (VMs), with varying hardware types, to run their application. The IaaS service model plays a core role in Machine Learning application processing since it is the model that provides more autonomy to the user [22]. Additionally, it is usually used to develop other software and platforms to run Machine Learning programs on the cloud, as we can see in the work of Pop [26].

AWS provides several VM types, which are grouped in families according to their features. The p2 and p3 families, for example, contain virtual machine types that are equipped with K80 and V100 GPU models, respectively. We use VMs from these families to perform our study because they contain GPUs that are more appropriated to train Deep Learning models. The GPU type and the number of GPUs of each virtual machine instance also imply a difference in price per hour.

In total, AWS provides 22 instance types with GPU grouped in six families. The p2 and p3 families have a total of seven VM types. Given this amount of options, users have the challenge of choosing the instance that provides the best cost-benefit correlation to run their applications. Selecting the proper VM type is essential because, as indicated by our results in Chapter 4, a poor choice may lead to a high financial cost, or extended execution time, when training a Deep Learning model. Finally, selecting the best instance requires a strategy that can determine it at a low cost.

Therefore, this work's primary goal is to develop low-cost methods that allow users to determine the best cloud computing virtual machine type to train Deep Learning models. In this context, the best VM type depends on the Deep Learning model and the user interest, which may include reducing the execution time or the cost to train the model.

To achieve this goal, we started by analyzing the behavior of two Deep Learning applications on AWS VM instances and their impact on cost and execution time. We also observed the influence of variables related to the training on the execution time, like accuracy stabilization and sample batch size. Then, we proposed and validated a methodology to estimate the cost and the performance relationship of VM types taking into account different batch sizes. This methodology requires the user to perform the whole training process for each batch size at least once, in one of the VM types, to determine the number of epochs required to train the model with each batch size. Since this process may be too expensive depending on the Deep Learning model and the training dataset, we propose a second methodology capable of estimating the cost and the performance relationship of VM types without performing the full training process. This new methodology assumes a fixed batch size; however, it costs much less since it does not require the entire training process to be performed before choosing the best VM type. We validated the accuracy of the proposed methodologies with two Deep Learning applications and showed that, in both cases, the methods were capable of identifying the virtual machine types that provide the smallest cost and the shortest runtime to train the Deep Learning model.

The remaining of this text is organized as follows: Chapter 2 introduces the fundamen-

tal concepts to understand our results and related work. Chapter 3 discusses the services for Deep Learning provided by AWS, Microsoft Azure, and Google Cloud. Chapters 4 and 5 present the two methodologies proposed in this work and their evaluation. Finally, Chapter 6 summarizes our conclusions and discusses future work.

Chapter 2

Fundamental Concepts and Related Work

This chapter explains the main concepts of Machine Learning used in this work, narrowed to Deep Learning models and Cloud Computing. Section 2.1 presents the main idea and definitions of Machine Learning and Deep Learning areas. Section 2.2 describes the main components of a training model, which influences the model quality and the application runtime, and briefly explain the parallelization of a Deep Neural Network implementation in the context of popular frameworks. Section 2.3 introduces the Cloud Computing terms and service models.

2.1 Machine Learning & Deep Learning

Machine Learning is an Artificial Intelligence subset in which the algorithm must learn patterns through a dataset. The Mitchell [23] definition of Machine Learning is "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In other words, a Machine Learning algorithm learns from a dataset a model or hypothesis that is capable of relating the data's example features received as input to the output variable. An example, following the definition of Goodfellow *et al.* [10], "is a collection of features that have been quantitatively measured from some object or event that we want the machine learning system to process". Additionally, the algorithm must be capable of generalizing; this means that the same model must correctly relate different data features than the data presented to it initially.

Goodfellow *et al.* [10] define the task T of Machine Learning as tasks that "are too difficult to solve with fixed programs written and designed by human beings". It can be a prediction task that follows supervised learning or a description task that uses unsupervised learning. See the description of each type of problem.

- **Prediction task:** This kind of problem can be solved using the supervised learning approach, which means that the dataset's examples must have a label. This label

will serve as the problem's target variable, which is the baseline for the model output. The prediction tasks can be classification and regression. In the classification problem, the target variable is a discrete value, while in the regression problem, it is a continuous value.

- **Description task:** In a description task, the goal is to identify the intrinsic features of a dataset and qualify it based on these features. This category usually uses unsupervised learning, which means that it does not require labeled dataset samples. Clustering, Association, and Summarization are the tasks of this category.

Regarding the performance measure P, it is common to use the **accuracy** for tasks like classification. Accuracy is the proportion of examples with correct outputs made by the model. The **error rate** measures the proportion of examples with incorrect outputs. It is also a common performance measure P.

Based on a brief survey on Machine Learning researches [11] and blogs content found on the Internet [3] [18], we identified a pattern on Data Scientists (including Machine Learning researches) workflow, and we defined it as the sequence of four steps: data preparation, training, evaluation, and inference.

The **data preparation** is the process where the data scientist prepares the dataset for the training process, which may include cleaning missing data, changing variable types, removing features, among others. The **training process** is when the algorithm is adjusting the model parameters based on the information present on the dataset. This step is followed by the **evaluation**, where the resulting model is assessed to measure its quality. If the evaluation indicates that the model is generalizing, the model is considered complete and can be used for **inference** with new data; otherwise, it must be trained again. Therefore, the inference step is when the model is ready to be used in the real world.

Deep Learning is a field of Artificial Intelligence that is popular because, among other things, it is capable of building complex concepts out of simpler ones. The beginning of ideas that would culminate in Deep Learning as we know it today starts from back to the 1940s. At that time, the use of function approximation techniques was explored to solve few problems. From 1980 to 1990, researchers started to use back-propagation to train Neural Networks and initiated a study field called Connectionism. Only in 2006, the idea of Deep Learning, as we know it, has gained public attention, putting a new light on the subject [13].

We can summarize Deep Learning as a way to express object representations in terms of simpler ones. Neural Network is the core model of Deep Learning. It is composed of layers, where each one of them is a mathematical function that maps input values to output values. A Neural Network must have many layers, in other words, it must be a Deep Neural Network to be considered a Deep Learning model. The first layer is called the input layer, the last one is the output layer, and all other layers between the input and the output layers are the hidden layers [10]. Figure 2.1 shows an example of a Deep Neural Network architecture applied to an image classification problem. Notice how the components represented by each layer simplify complex features of an image into simpler ones.

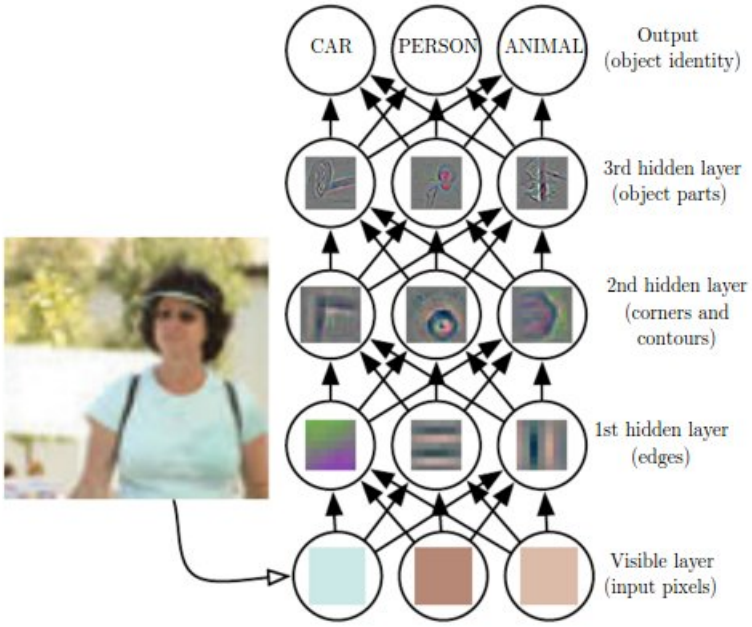

Figure 2.1: Example of a Deep Neural Network applied to an image classification problem. Figure reproduced from Zeiler and Fergus [39].

2.2 Training a Neural Network

The Feedforward Neural Network is called Feedforward because the connections among layers happen from the neurons of one layer towards neurons on subsequent layers. Therefore, the input flows in only one direction. Since it is the basis for other Deep Neural Networks, we use the Feedforward Neural Network as an example to explain essential concepts to understand the functioning of a Neural Network.

The goal of a Neural Network is to find a function f that maps the inputs x to the outputs y. The function can be defined as $y = f(x, \theta)$. Training a Neural Network model consists of finding the θ values — that we call parameters — to compose the function to be combined with the inputs x and discover the right output y.

The process of finding the parameters that optimize the model performance P is the **training process**. During this process, we are dealing with the training set, which is a subset of our dataset. The training is done by steps, also called **iterations**. In each iteration, a subset of the training set, known as a batch, is used to adjust the model parameters to improve its performance (*e.g.*, accuracy) accordingly to this subset. The amount of examples in the batch is known as the **batch size**.

Once the network process this batch of examples, it calculates the error rate, and the network walks through the layers in the reverse order to calculate how the parameters must be updated. When the network updates the parameters, it will process another batch of examples. The algorithm repeats these steps until the processing of the entire training set. At this point, the network completed one **epoch**. In other words, one epoch represents the processing of the entire training set.

After an epoch, the training process calculates the model performance P through accuracy by evaluating it using a validation dataset. Usually, the validation set has fewer

and different objects from the training set since this step is to check the parameters' efficiency.

The training process in a Deep Learning model may be composed of multiple epochs. The number of epochs is determined by the user or by heuristics that take into account how the validation accuracy is changing across epochs. Note that in the training process, the entire data set is accessed by the training process several times. This, plus the fact that the dataset is usually large, and the current Deep Neural Networks have several layers, which increases their depth, usually makes the training process a time-consuming process.

We describe the main components of Deep Neural Network training as follows.

- **Training set:** A subset of the entire dataset used in the training process;
- **Validation set:** A portion of the dataset, with different samples from training set, used to evaluate the model performance;
- **Validation accuracy:** A numerical value representing the performance P of the model. This variable is calculated at the end of each epoch.
- **Batch size:** The number of training set samples that the network process at a time when adjusting the θ parameter;
- **Iteration:** The processing of one batch of samples;
- **Epochs:** The processing of the entire dataset. One epoch is composed of n iterations, where the n is the size of the training set, *i.e.*, the number of samples in it, divided by the batch size;
- **Validation:** The evaluation the performance P of the model. It is usually performed at the end of each epoch.
- **Training:** The processing of a dataset during a number of epochs to achieve the best parameters θ to compose the final function $f(x) = \theta \times x$;

2.2.1 Parallelization of a Neural Network

Given the Deep Learning computing-intensive characteristic, users usually rely on GPUs to train their networks to exploit the inherent parallelism in their workload [4]. In recent years, the use of other strategies to run Deep Learning applications on parallel environments, such as *multi-machine parallelism*, has also gained force inside the scientific community [4].

The most popular frameworks for Deep Learning already allow us to train neural networks in multi-GPU environments. The Keras and TensorFlow frameworks, for example, provide tools that use the **synchronous parallelization** strategy to accelerate the process of training neural network models.

In this strategy, there is a **parameter server** that is responsible for sending several training samples to each processor (GPU's), which computes updates to the θ parameters.

Once these updates are computed, they are sent back to the parameter server to update the model parameters. The parameter server can be the CPU or one of the GPUs; in our experiments, it was the CPU.

Usually, the batch size is divided equally for each processor. For example, if we have a batch of 256 examples and an environment with 8 GPUs, each GPU will process 32 examples in one iteration. After the processing, the parameter server updates the θ parameters and sends a new amount of data to the GPUs.

Since this strategy is synchronous, the parameter server has to wait for all GPUs to complete their tasks, slowing down the process. Besides, there is intensive communication between the parameter server and the GPUs, which creates a communication overhead that may increase the total training time when the number of GPUs is increased. Figure 2.2 illustrates how a model may be trained in parallel using a parameter server (CPU) and two GPUs.

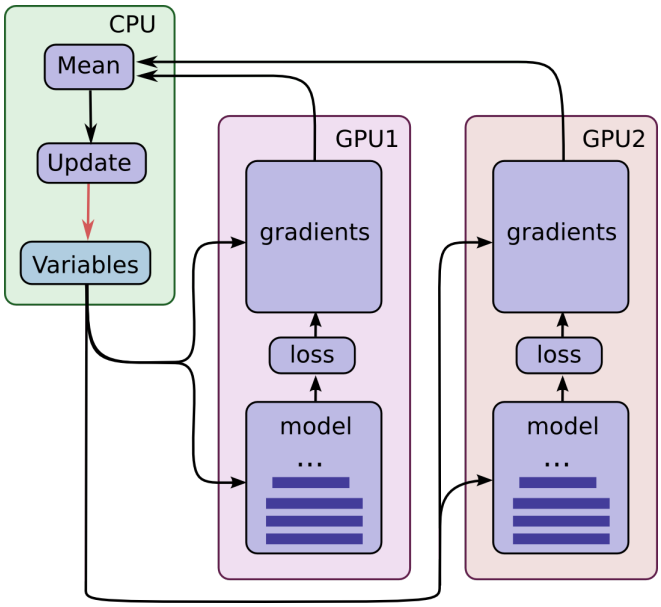

Figure 2.2: Example of a parallel training using a parameter server and two GPUs. Figure reproduced from `https://www.tensorflow.org/tutorials/images/deep_cnn`.

2.3 Cloud Computing Definitions

The National Institute of Standards and Technology (NIST) defines Cloud Computing as a "model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.*, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [22].

The NIST also defines the five main features of Cloud Computing as an on-demanding self-service, with broad network access, which provides different kinds of resources, rapid elasticity, and measured service. The NIST definition also establishes the following service models:

- **Software as a Service (SaaS):** The consumer uses an application that runs over a Cloud Computing infrastructure. The provider is responsible for all the infrastructure and software management.
- **Platform as a Service (PaaS):** The consumer has access to the cloud infrastructure for developing an application. All the infrastructure management and configuration is performed by the provider, as all the tools, frameworks, or programming languages. Here the consumer only worries about the application development and its use.
- **Infrastructure as a Service (IaaS):** This is the most basic service, where the consumer rents computing resources and is responsible for managing most of the infrastructure, including the operating system, software and framework installations, storage, and applications development.

The IaaS model allows users to rent Virtual Machines (VMs). In this model, the user selects a VM type, which defines the hardware configuration, and instantiates a VM of this type. We observed that the three providers we looked at (AWS, Microsoft Azure, and Google Cloud) have three ways of charging for VM instances.

1. **On-demand instances:** Each instance has a price per hour, and the user pays based on the time the instance is on;
2. **Spot instances [31]:** Providers charge less for the VM instance; however, they may shut them down whenever they want;
3. **Reserved instances:** The user pays for the resources in advance for at least one year.

In the context of Cloud Computing, it is crucial to understand metrics related to services' costs. The cost of using an EC2 instance, for example, is mainly defined by the cost of executing the VM, which is calculated by multiplying the VM type price (per hour) by the total time that the instance was on. The price of a VM type varies based on the instance characteristics, such as the number of virtual CPUs, memory capacity, bandwidth, and others. In AWS, VM type price may also vary according to the datacenter region. In this work, we use the term "price" to express the amount of money charged by time (*e.g.*, USD/h) and the term cost to express how much the user was charged for using a given computing resource (*e.g.*, VM instance) for a period of time. Unless stated otherwise, the cost is always computed by multiplying the VM type price by the amount of time it was on. The AWS website [34] contains the list of EC2 VM types and their prices.

To measure how adequate a VM type is to train a Deep Learning model, we measure (or estimate) its execution time. In this way, when we speak about *performance*, we are referring to the training time. Therefore, when we say that VM type x had a better performance than VM type y, it means that VM type x takes less time to execute a given task than VM type y.

2.4 Related Work

We found a few works that study Cloud Computing for Machine Learning applications. Some of them have the Deep Learning strategy as the main subject, similar to our work. Dube *et al.* [9], for instance, propose the AI Gauge, a service that estimates a Deep Learning application runtime. The AI Gauge provides the estimated runtime in an online and an offline way. The online estimation uses the same concepts that we used in this work, but it focuses on predicting the application execution time instead of determining the instance that provides the best cost-benefit correlation to the user, as we do.

The work of Kaplunovich *et al.* [14] proposes a Machine Learning model to predict Machine Learning algorithms runtime on EC2 instances. At the same time, their approach predicts the best instance for each algorithm and batch size. However, they do not include Deep Learning problems in their model, while we specifically focused on this kind of problem.

Venkataraman *et al.* [37] propose Ernest, a framework to predict the performance of analytics applications on the cloud. They give special attention to Machine Learning techniques and use them to create the prediction model. Their strategy is to use a small version of the dataset to build the model and then apply it to predict the performance of large-scale analytics applications. The resulting model achieved an error smaller than 20%.

More recently, Son *et al.* [35] created a method to predict matrix multiplication latency in cloud computing. Since the training and inference of Machine Learning models rely on several matrix multiplication operations, this work helps could be used to select estimate the training latency in cloud resources. They use Machine Learning algorithms to make the prediction as well. Our work differs from theirs mainly because we analyze real applications and, because of that, we take into account variables like the accuracy stabilization and epochs to converge, while they focus only on the algorithm's kernel.

The work of Alipourfard *et al.* [1] proposes a system called CherryPick to build a performance model for Big Data analytics. This system uses Bayesian Optimization to predict the best cloud configuration for various applications regarding the number of CPUs and instances, and even the RAM memory. Despite promising results in its prediction, this work did not analyze GPU instances either Machine Learning applications.

Similarly, the tool proposed by Samreen *et al.* [29], Daleel, uses Machine Learning to help the choice of cloud infrastructure for different applications. However, the focus of this work is not Machine Learning problems.

The Carneiro *et al.* work [5] analyzes the Google Colaboratory platform, or Google Colab, a Google Cloud environment to run Deep Learning and other GPU-based applications. The work's goal is to answer if Google Colab is feasible to run a Deep Learning application, and they present the service's strengths and limitations. As Google Colab is a free platform, its cost is not the subject of this work. Besides, Google Colaboratory is a tool that provides a single GPU to the user, so the multi-GPU running features of a Deep Learning application is not present in this paper.

The work of Kurkure *et al.* [17] presents a performance study of GPU virtualization solutions for running Machine Learning applications in the cloud environment. They test the virtual GPUs for running a Machine Learning application alone and combined with

other GPU-based applications. Their work also discusses the scaling metric in multiple GPUs, shows a comparison between virtualized and physical GPUs, and compares virtualized GPUs with CPUs as well. Unlike us, they did not focus on performance and cost trade-off but on analyzing the performance difference between a virtualized environment and a physical environment.

Regarding the cloud platforms for Machine learning analysis, we found the work of Yao *et al.* [38]. Their paper analyzed the platforms for Machine Learning of different cloud service providers, including AWS. Their analysis includes the correlation between performance in terms of accuracy and the difficulty of using each platform. They concluded that the bigger the user's power over a platform, the more significant is the chances of getting better accuracy. This work did not look at the runtime and cost components but focused on the usability and accuracy factors.

Based on the question we intend to answer in this work, that is, which cloud instance is the best for a Deep Learning problem, we compare the related work we found in Table 2.4, taking into account the aspects we considered in our proposal. They are

1. Did the work focus on Deep Learning applications?
2. Did the work analyze the applications' performance?
3. Did the work analyze the cost-benefit correlation?
4. Did the work provide tools to select the best instance for a given application?
5. The tool provided can determine the best instance with a low cost?
6. To make the prediction, the proposed tool only needs the user's ML application?

Table 2.1: Comparison between our research and the most correlated works. (NA: Not Applied, NF: Not Found)

Work	**Item 1**	**Item 2**	**Item 3**	**Item 4**	**Item 5**	**Item 6**
Yao *et al.* [38]	No	No	No	No	NA	NA
Kurkure *et al.* [17]	No	Yes	No	No	NA	NA
Samreen *et al.* [29]	No	Yes	No	Yes	NF	No
Kaplunovich *et al.* [14]	No	Yes	Yes	Yes	NF	No
Son *et al.* [35]	No	Yes	Yes	Yes	NF	No
Venkataraman *et al.* [37]	No	Yes	Yes	Yes	Yes	No
Alipourfard *et al.* [1]	No	Yes	Yes	Yes	Yes	No
Dube *et al.* [9]	Yes	Yes	No	Yes	NF	No
Carneiro *et al.* [5]	Yes	Yes	NA	NA	NA	NA
Ours	Yes	Yes	Yes	Yes	Yes	Yes

Chapter 3

Cloud Computing Services for Deep Learning

This chapter surveys the current Cloud Computing products, especially for Machine Learning problems. We investigate the three leading Cloud Computing providers: Amazon Web Services (AWS), Microsoft Azure, and Google Cloud. We started this study in 2018, and some of the products were discontinued, but we cite them here for record purposes. In the following sections, we describe the services of AWS, Microsoft Azure, and Google Cloud.

3.1 Amazon WEB Services (AWS)

The Amazon Web Services (AWS) provide many services that focus on Deep Learning applications [32]. In the SaaS model, they provide applications for Image problems, such as Image Classification and Image Segmentation, as for Natural Language Processing, such as Translation, Transcription, and others. In the PaaS model, they offer the Amazon SageMaker and the Amazon Machine Learning platforms; the last one is already discontinued. In this work, we focused on the IaaS (Infrastructure as a Service) model, which is provided by the Elastic Compute Cloud (EC2) product.

3.1.1 Elastic Compute Cloud (EC2)

The Elastic Compute Cloud (EC2) is an IaaS product. This means that the AWS only provides the infrastructure and manage it. The software installation, application development, and all other need is the user' responsibility. In this case, the user may select among a set of disk images, called Amazon Machine Images (AMI) [33] when instantiating a VM for use. These AMIs contain an operating system already installed and pre-configured and some software installed. There are several AMI options, and some of them already contain GPU drivers and Deep Learning frameworks to support the development of Deep Learning applications. After instantiating a VM with a given AMI, the user can access this instance through *ssh* connection and use it at will. Some of these AMIs have the software and drives already installed, but the users are responsible for all the other infrastructure preparation.

AWS provides different types of virtual machines. They group these machines based on their specialty. For example, some instances are optimized for computing while others have more memory or have better network capabilities.

For Deep Learning applications, AWS provides instances for accelerated computing. Most of them have GPUs, the primary hardware to run Machine Learning algorithms [4]. Moreover, they also provide an AMI with several Deep Learning frameworks, such as TensorFlow and Pytorch, the most popular Python libraries (*e.g.*, Pandas, Keras, Numpy), and NVIDIA drivers to use the instances' GPU. This AMI facilitates the user's job since they already have several Deep Learning tools at their disposal when they start the EC2 instance.

In this work, we used six on-demand instances with GPU hardware. They can be grouped into two families: the p2 family, whose instances use the K80 GPU type, and the p3 family that provides the V100 GPU type. Each instance also has a different quantity of GPUs and different prices per hour. Table 3.1 shows the main features of each analyzed instance.

Table 3.1: Main features of AWS EC2 instances types used. Prices on February 2021.

Instance	**GPU type**	**#GPUs**	**Price (U$/hour)**
p2.xlarge	K80	1	0.90
p2.8xlarge	K80	8	7.20
p2.16xlarge	K80	16	14.40
p3.2xlarge	V100	1	3.06
p3.8xlarge	V100	4	12.24
p3.16xlarge	V100	8	24.48

Deep Learning users commonly create a Jupyter Notebook[1] to develop and train a Deep Learning application. EC2 instances allow them to do this. To do so, the user can manually create an *ssh* tunnel between his/her local computer and the EC2 instance to open the Jupyter Notebook in their browser.

3.1.2 Amazon SageMaker

The Amazon SageMaker is a Platform as a Service (PaaS) product for Machine Learning applications. The product is split to meet three steps of a Machine Learning process: data preparation, training step, and deployment step. The user has the option of using the SageMaker for one or more steps.

The data preparation step allows the user to prepare the data for the training step. For example, the user may remove blank cells, convert the data to another format, remove some features, *etc.*. In this step, the user instantiates a Jupyter notebook in a VM type of his/her choice. The VM types available also have different prices and features, similar to the EC2 instances. Once the user configures the Jupyter notebook, he/she can start and access it through the browser without worrying about setting *ssh* tunnels.

[1] `https://jupyter.org`

For the training step, the user must create a training job. First, the user needs to choose a training algorithm. This algorithm can be one of all provided by the Amazon SageMaker itself. They provide many algorithm options for different kinds of Machine Learning problems, not only for Deep Learning. For Deep Learning, for example, they offer an Image Classification algorithm based on the ResNet network [12]. Then, the user must configure the algorithm hyperparameters. Each algorithm requires different hyperparameters, so the user must have the minimum knowledge about the algorithm functioning to do this task. The user also has the option of using his/her own implementation. In this case, he/she must put his/her code in a Docker container application to run on the Amazon SageMaker instances.

The Amazon SageMaker training instances have features similar to the ones present on EC2 VM types. However, Amazon affirms that they are optimized for Machine Learning problems, so their names have a *ml* at the beginning of the word (*e.g.*, ml.p2.xlarge is the equivalent to the *p2.xlarge* VM type). The *ml* instances' prices are also higher than the EC2 instances. Table 3.2 shows the price per hour for the ML GPU instances that we use in this work.

Table 3.2: Main features of AWS SageMaker instances types used. Prices on February 2021.

Instance	**GPU type**	**#GPUs**	**Price (U$/hour)**
ml.p2.xlarge	K80	1	1.26
ml.p2.8xlarge	K80	8	8.64
ml.p2.16xlarge	K80	16	16.6
ml.p3.2xlarge	V100	1	3.8
ml.p3.8xlarge	V100	4	14.7
ml.p3.16xlarge	V100	8	28.15

Amazon SageMaker also supports the Inference step. This step happens after the algorithm is trained and finalized. The resulting model is then implanted on the Cloud to generate predictions over new data. Figure 3.1 shows the dashboard of Amazon SageMaker. See the division of the tasks on its left side. Recently, AWS incorporated new functionalities on SageMaker, such as labeling data objects through the Ground Truth application.

3.1.3 Other Services

The AWS had another platform for Machine Learning, called Amazon Machine Learning. This service offered similar features to Amazon SageMaker, but it is even most automated. In this service, users do not need to have in-depth knowledge about Machine Learning. They basically need to know their dataset to indicate the target variable. The Amazon Machine Learning itself selected the proper algorithm for their dataset and only worked for prediction problems. Amazon Machine Learning was discontinued, and it is not available for new customers.

The AWS also launched the AutoScaling for EC2 instances. This service allows the

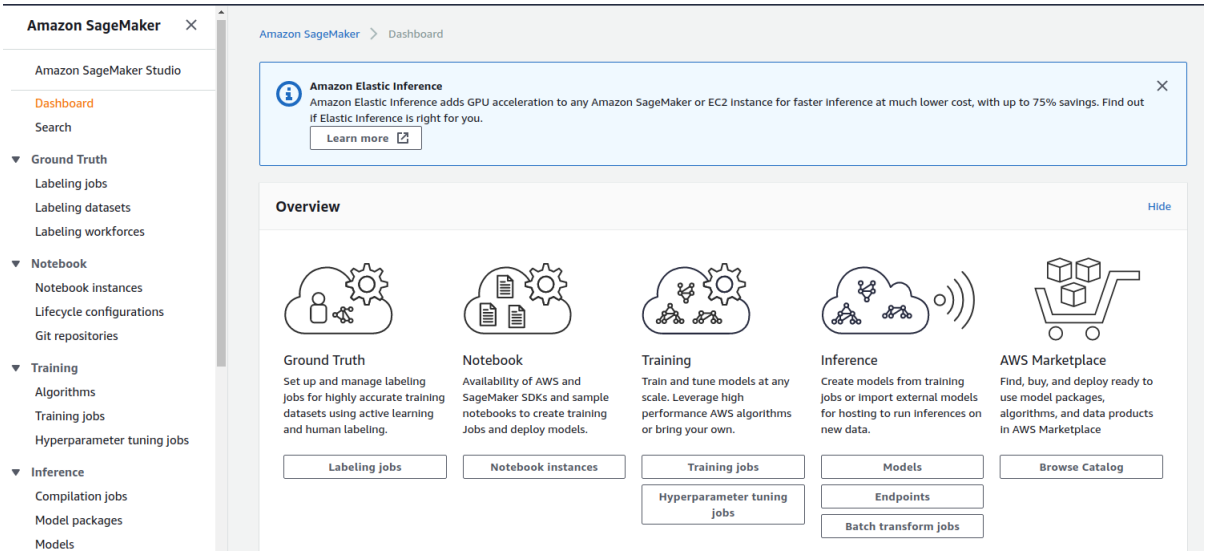

Figure 3.1: Dashboard of Amazon SageMaker from June 2020.

personalized scaling of EC2 instances for the prediction step. This feature avoids the unnecessary use of EC2 instances during the prediction.

Regarding Software as a Service, the AWS has solutions for different tasks. For example, the Amazon Rekognition is a product for Image and Video Analysis, such as image classification, face recognition, and inappropriate content detection. For the speech task, they have the Amazon Polly that converts text to audio speech and Amazon Transcribe, which does the inverse task. For language tasks, we found Amazon Lex, Amazon Translate, and Amazon Comprehend.

In 2019, AWS launched the Amazon Textrack, a service that extracts text and data from digital documents. They also launched the Amazon Forecast that makes accurate predictions through temporal data; this type of data considers variables that change with time, such as price, promotions, and number of employees.

3.2 Microsoft Azure

Microsoft Azure lists four pillars of its products: productivity, hybrid services, intelligence, and security. For the productivity aspect, Azure provides tools, languages, and frameworks to make the user's job easier. The hybrid services allow us to make some of the tasks on a local computer and then migrate to the Cloud. Regarding the intelligence aspect, Azure provides intelligent systems, tools to develop personalized models, and the infrastructure to run the user's application. For the security aspect, Azure ensures the privacy of the user's application and data.

We found Azure provides products that fit in three service models (IaaS, PaaS, and SaaS). As IaaS, Azure provides VMs specifically for Machine Learning. The Azure Machine Learning and Azure Machine Learning Studio are the PaaS services, and they also have SaaS services that solve vision, language, speech, knowledge, and search tasks. In the following section, we describe these services in more detail.

3.2.1 VMs for Machine Learning

Microsoft Azure provides virtual machine instances of multiple types and disk images to support Machine Learning. These disk images already contain the NVIDIA GPU drivers and popular frameworks for Deep Learning installed. Microsoft Azure also makes available the Microsoft tool kit, which is a differential compared with other providers.

Like AWS, Azure has different types of virtual machines and pricing models: on-demand, spot, and reserved instances. The on-demand instances charge per hour of usage. The spot instances are cheaper, but the provider may interrupt them at any time. The reserved instances are exclusive for clients that pay in advance for the resource.

Table 3.3 shows an example of on-demand prices for GPU instances of NC type in Azure. Instances of NC type have GPU hardware for accelerated computing and graphics processing.

Table 3.3: Main features of Azure instance types with GPU. Prices on February 2021.

Instance	**GPU type**	**#GPUs**	**Price (U$/hour)**
NC6	K80	1	0.90
NC12	K80	2	1.80
NC24	K80	4	3.60

3.2.2 Azure Machine Learning

Azure Machine Learning provides a full solution to run Machine Learning models. This platform supports several Deep Learning steps, including data processing, training, experimenting, and model implantation. It allows the user to use an IDE of its preference, train the model on the Cloud, and implant it on a container. The platform also provides support to all Python-based frameworks and libraries, such as TensorFlow[2] and scikit-learn[3].

Another feature of this platform is the introduction of a workspace. In this space, the user can manage all different models in the platform. It can also store the applications' execution history and their logs. A user can create a workspace and add many collaborators to the project.

3.2.3 Azure Machine Learning Studio

The Azure Machine Learning Studio is another Azure Platform as a Service product. This platform is different from Azure Machine Learning because it is more automated than the previous one. The users only need basic knowledge about a Machine Learning algorithm to use the Studio, things like the hyperparameters required for that algorithm and its evaluation metrics. The Azure Machine Learning Studio already provides implemented algorithms ready to use, but a user can also put its algorithm on the platform.

[2] `https://www.tensorflow.org`
[3] `https://scikit-learn.org`

The entire platform is based on a drag and drop system. That means that the user does not code while using it. We build the application by connecting blocks; these blocks correspond to an algorithm, and they can do data processing, select the Machine Learning algorithm, or even analyze the resulting model. Once the platform generates the ideal model, it converts it from a training experiment to an implementation experiment, and it will become a web service that others can access.

When initializing a session, the user must create an empty canvas to create a new experiment. The Studio provides some experiment templates or even dataset samples. Figure 3.2 shows the Azure Machine Learning Studio Dashboard and an example of a Machine Learning experiment implemented on the platform. The left bar contains the available options for data treatment and model creation. Each rectangle on the image's center corresponds to a module; we call them blocks. We must connect these modules to create a complete workflow, from data preparation until the model evaluation. When the user selects a block, the right bar shows the required setting for that module.

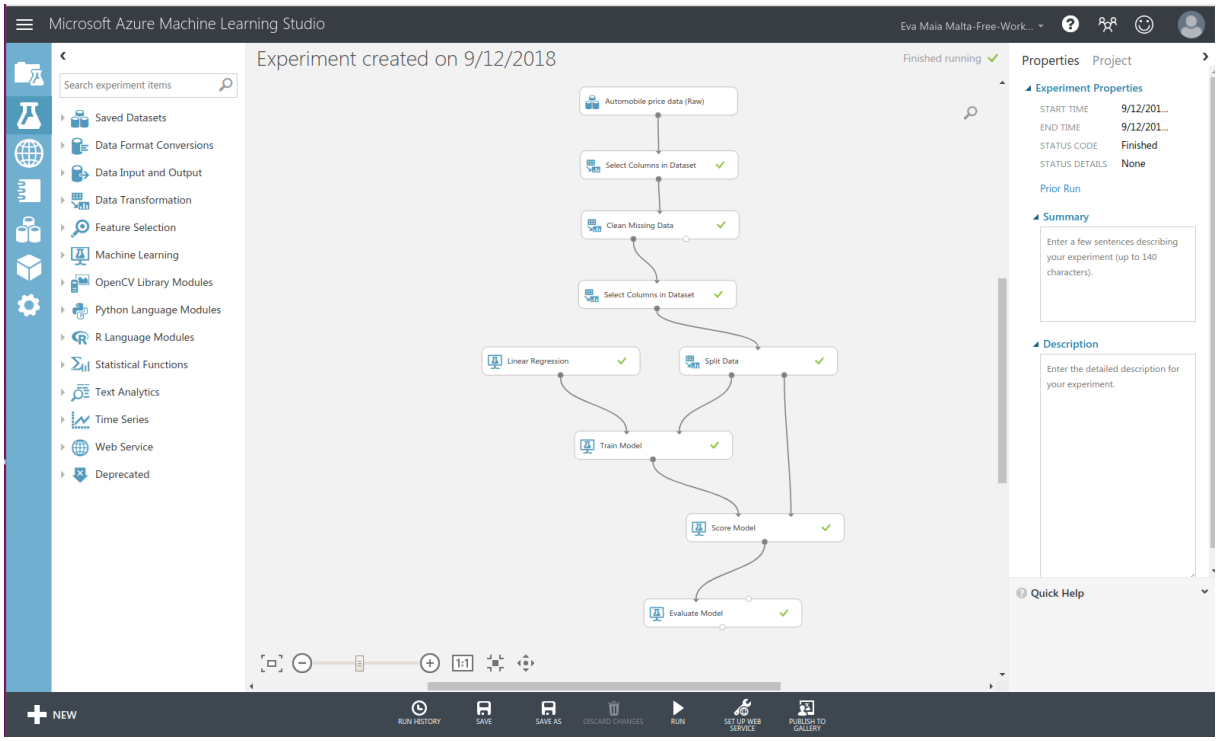

Figure 3.2: Dashboard of Azure Machine Learning Studio from June 2020.

For data preparation, the platform offers the "Data Transformation" option, which contains blocks for "Manipulation", "Filtering", and "Reduction". If the user selects the "Manipulation" option, it can do things like "Add Rows", "Clean Missing Data", and "Join Data". When the data preparation is completed, they can save the resulting dataset in the option "Saved Datasets".

Users may choose the Machine Learning algorithm by accessing the "Machine Learning" option, also on the left bar. There, they will find algorithms for classification, anomaly detection, clustering, and regression problems. Once the algorithm is trained, users can finally use the final model with the "Evaluate Model" block.

3.2.4 Other Services

Microsoft Azure provides products that fit the Software as a Service model for vision, languages, knowledge, and search tasks. The *computational vision* service provides information from an image, such as the image class, the image type, and its quality. It also detects the faces and their features and describes their content.

The Video Indexer, on the other hand, extracts insights from videos. The Face API service is focused on face recognition and can be used for security purposes, working to detect similar faces.

3.3 Google Cloud

Google Cloud is also a provider of Cloud Computing resources. At the time of this survey, they did not have so many resources specifically for Machine Learning as the other two providers, AWS and Microsoft Azure. Because of this, in the past year, the Google Cloud was the Cloud Computing provider the most presented news in their catalog for Machine Learning.

Google developed new hardware specifically to run Deep Learning called Tensor Processing Units (TPUs). This hardware is provided on the Google Cloud platform, and it is a differential of this provider.

Google Cloud's services for Deep Learning are the compute Engine instances that fit the IaaS service model, the AI Platform, and other Machine Learning software for generating predictions over the user data. The following sections describe these services.

3.3.1 Compute Engine

The Computing Engine is a Google Cloud IaaS product. It provides different VM types, which may contain GPUs and TPUs. Google Cloud also charges these instances differently based if they are on-demand, spot, or reserved instances. Table 3.4 shows the price per hour per GPU and TPU of Google Compute Engine. For Deep Learning, they have available virtual machine images with Deep Learning tools already installed and integrate with different Google Cloud products, such as storage and analysis services.

Table 3.4: Main features of Google Compute Engine instances with GPU and TPU. Prices on February 2021.

Instance	**Hardware**	**#GPUs**	**Price (U$/hour)**
GPU	K80	1	0.45
GPU	V100	1	2.48
TPU	v2	1	4.50
TPU	v3	1	8.00

3.3.2 AI Platform

The AI Platform product is composed of AI Platform Notebooks, AI Platform Training, and AI Platform Prediction services. These services help the user prepare the data, train models, and use them for prediction, respectively.

On the AI Platform Notebooks, the user instantiates a JupyterLab integrated with other Google Cloud services to prepare its dataset and develop its Machine Learning algorithm. The notebook comes with Machine Learning frameworks and other tools.

The AI Platform Training allows users to execute TensorFlow applications using the scikit-learn library on cloud computing resources. If the user wants to use another framework, he/she can create a container with its application and run it on Google Cloud computing resources in the same way. To run an application on AI Platform Training, the user must allow its application to receive the data from a storage service of Google Cloud, such as Cloud Storage or Cloud Bigtable. Once the application is ready, it must be on Google Storage to allow the platform to access it.

We noticed that the user must have knowledge about Machine Learning and implement its application because this platform does not provide implemented algorithms. The AI Platform Training was designed to minimize the interference in the user's code.

3.3.3 Other Services

The Cloud AutoML is a Google Cloud service that allows the user to train models easily. It is a set of services of Machine Learning for not experts, in which one can train a model using Google Cloud technology. Their segments are: AutoML Natural Language, AutoML Translation, and AutoML Vision.

The AutoML Vision trains models to extract information from images. It can classify the image, detect faces and objects, and obtain texts from it. The AutoML Translation can be used to translate a text file or identify the language of a file when it is not specified. The AutoML Natural Language extracts relevant information from a text (*e.g.*, blogs and papers) and can identify consumer feelings regarding a product or company.

Chapter 4

Selecting the Best GPU Instance through Epoch Time

Given the challenge of choosing the proper cloud instance, it is crucial to have a method that helps the user in the instance selection. This method should take into account variables like the instance's features and its price per hour, as the particularities of the Deep Learning application, like the network's complexity, dataset size, and the model configuration.

This chapter describes the first methodology we proposed to support the user in this task. This methodology helps the user identify the virtual machine type and batch size setting that provides the shortest training time or the lowest cost. To do so, we first analyze the behavior of Deep Learning applications on AWS' EC2 GPU instances. Then, we describe the proposed method its validation.

Our initial experimental results revealed that:

1. The cheapest instance does not always provide the lowest cost;
2. The instances with more GPUs do not provide the smallest runtime;
3. Choosing an inadequate instance can significantly increase the experiment's cost;
4. Choosing the proper batch size is important for the final cost and the model accuracy.

Based on these observations and understanding that they will not be necessarily valid for all Deep Learning problems and Cloud virtual machines, we concluded that a methodology to discover the best instance for a Deep Learning problem is needed. Our methodology takes into account the batch size influence on the application's performance and identifies the cost and performance relationship between EC2 instances and the ideal batch size. It can also discover the best instance for a Deep Learning application without training the model to its completion on all VM types.

This chapter is organized into five sections: Section 4.1 describes the experimental setup. Section 4.2 details the experimental results using two Deep Learning applications that allowed us to understand the behavior of an application on single and multi-GPU instances and the batch size influence in the total runtime. Section 4.3, describes the

methodology itself and explains the premises used to formulate it. Section 4.4 shows the methodology's validation using two Deep Learning benchmarks and the cost of using it to determine the best instance for these two examples. Finally, Section 4.6 provides the conclusions, discusses the methodology's strengths and limitations, and lists future work.

The methodology and the results presented in this chapter were published at the 12$^{\text{th}}$ International Conference on Utility and Cloud Computing (UCC '19) [21].

4.1 Experimental Setup

In the AWS' EC2 service, we selected GPU-based VM types from p2 and p3 families to perform the experiments. Table 3.1 lists the features and prices of these instances.

We chose two Deep Learning applications to serve as benchmarks. The first one trains a Full Convolutional LeNet network (FCN) [19] to identify diffraction apexes on seismic images [2]. This application is a binary classification problem. The training/validation dataset contains 40,512/212 images. These images have 64 $\times$ 64 pixels. Figure 4.1 shows a seismic data example. The green points show the presence of an apex, while the red ones show a not-apex. The application uses the Momentum Stochastic Gradient Descent as optimizer [28]; the learning rate starts with 0.1 and is gradually decreased to 0.01 during the training. The application also uses the Dropout regularization [36].

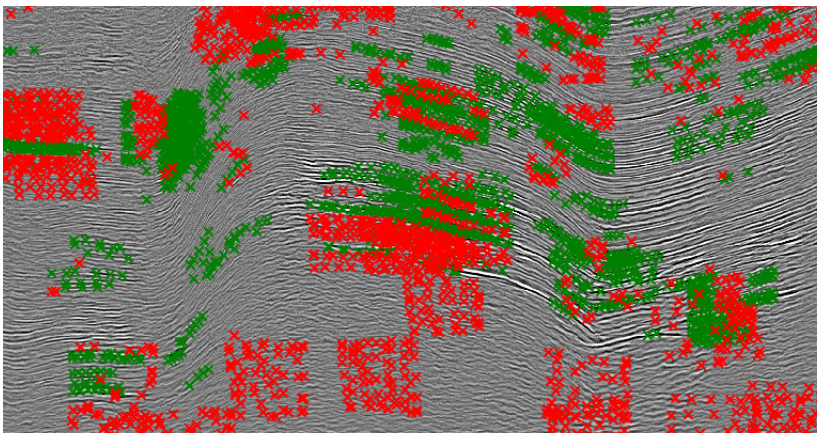

Figure 4.1: Seismic dataset examples.

The second application trains the ResNet-50 network [12] to classify images from the CIFAR-10 dataset [16], a multiclass classification problem. The CIFAR-10 dataset contains 60,000 images. The training dataset has 50,000 of these images, while the validation set has the 10,000 remaining images. The dataset contains 10 classes, such as airplane, automobile, bird, and cat. These images have 28 $\times$ 28 pixels. We can see an example of the CIFAR-10 dataset on Figure 4.2. This application also uses the Momentum SGD as optimizer [28], and the learning rate starts at 0.1 and is gradually decreased to 0.002 during the training. Both benchmarks start the training step with random parameters.

Initially, we run these applications in the instances described in Table 3.1. We varied the batch sizes in 256, 512, 1024, and 2048 images, and we execute the application five times for the seismic dataset/FCN benchmark and three times for the CIFAR-10

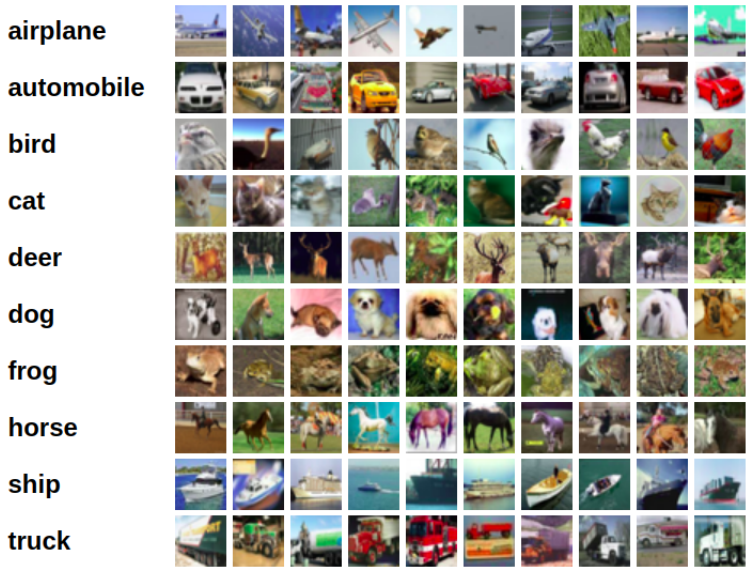

Figure 4.2: CIFAR-10 dataset examples. Figure reproduced from Krizhevsky *et al.* [16].

dataset/ResNet-50 benchmark for each batch size.

The application executes until its validation accuracy stabilizes. In the seismic dataset/FCN application, we used the Early Stopping callback[1], provided by Keras framework, to stop the training when the validation accuracy is not improved for five consecutive epochs. In the CIFAR-10/ResNet-50 benchmark, we ran it one time to discover the stabilization point, and we used this value as a boundary for the execution.

Once the execution was over, we measured each experiment's final runtime by calculating the average runtime for each batch size as its standard deviation. We used this average runtime to calculate the experiment's average cost as well. The entire analysis is based on these two variables.

4.2 Deep Learning on GPU Instances and the Batch Size Influence

We noted three things when we ran the seismic data/FCN and the CIFAR-10/ResNet-50 applications. First, the scalability of the applications was very poor. Second, the p3 instance family has a better performance when compared with the p2 instance family, which is expected. Third, the performance improved when we increased the batch size, but only to a certain point.

Figure 4.3 shows the correlation between batch size and runtime for all VM types with the seismic data experiments. Notice that VM types from the p3 family performed better than VM types from the p2 family with the same number of GPUs. The p3.8xlarge VM type achieved the lowest runtimes.

We also observed that the runtime is also affected when changing the batch size. The performance of the p2.16xlarge and p3.16xlarge instances improved when the batch size

[1] `https://keras.io/api/callbacks/early_stopping`

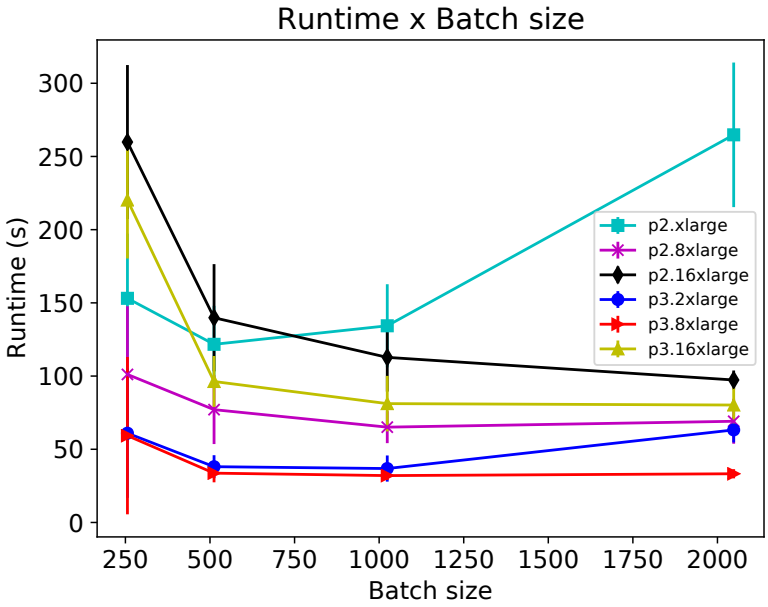

Figure 4.3: Runtime *vs* Batch size for the seismic dataset for all instances.

was increased. Notice that, with a batch size of 256 images, these two instances had a poor performance, but the runtime significantly decreased when the batch size was increased. In other cases, the performance was improved up to a point. This effect can be explained given the amount of work that each GPU is dealing with. As we used the synchronous parallelization strategy, with a batch size of 256 images, for example, and an instance with 16 GPUs, each GPU processes only 16 examples per time, which may cause GPU underutilization.

Also, after an iteration, all GPUs must communicate with the parameter server, using synchronous parallelization, each GPU does this communication after processing only 16 images, which leads to communication overhead. Therefore, increasing the batch size allows us to make better use of the computational resources and, in multi-GPU instances, also decreases the communication overhead between the GPUs and the parameter server. However, in some cases, just increasing the batch size does not fix the performance problem. For example, in single-GPU instances, increasing the batch size can make the performance worse, as we can see in the p2.xlarge runtime with a batch size of 1024 and 2048 images. With the p3.2xlarge, the same thing happens with 2048 images.

Figure 4.4 also shows similar results. This chart also relates the batch size with the runtime for all instances for the CIFAR-10 dataset. See that the p3.2xlarge, p3.8xlarge, ad p3.16xlarge achieved better performance once compared with the p2 instances family. We observed that when we increased the batch size from 256 to 512 images, almost every VM type had its performance improved.

Table 4.1 shows the details about the results for the seismic data experiments. Notice how the runtime becomes smaller when we increase the batch size for the p2.16xlarge and p3.16xlarge especially. The highlighted cells on the 'Speedup' columns show the lowest and the highest achieved runtime.

It is also important to notice that increasing the batch size does not always imply a

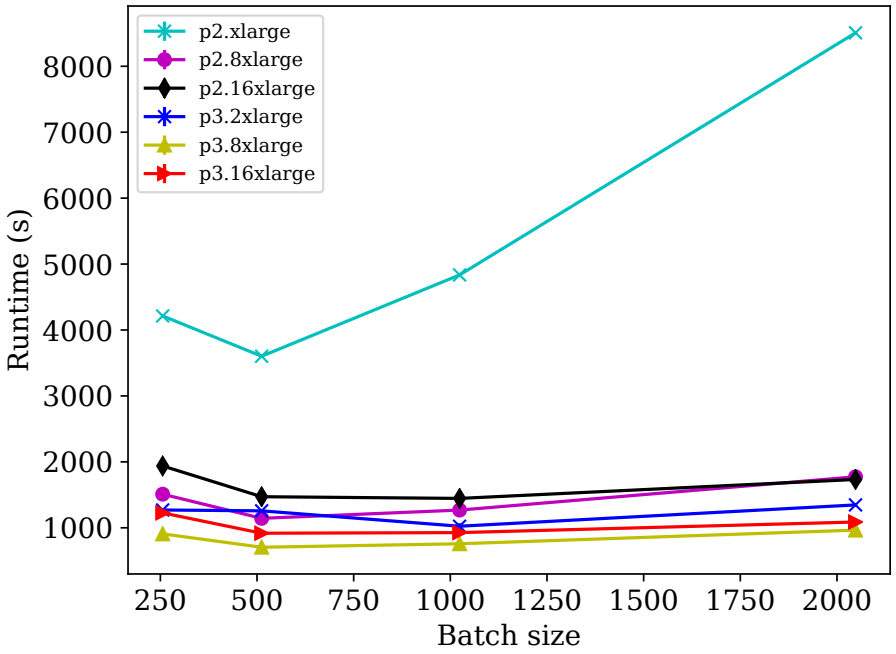

Figure 4.4: Runtime *vs* Batch size for the CIFAR-10 dataset for all instances.

faster execution. Sometimes, increasing the batch size can also harm the performance. This effect can be observed on the p2.xlarge VM type when we use a batch size of 1024 and 2048 images on both experiments (Figures A.1 and 4.4). Similar results happened with other instances as well. We hypothesize that the cause of this is the hardware limitation and framework implementation. We intend to investigate this aspect in future work.

Additionally, we also observed that naively increasing the batch size may not only harm the performance in terms of runtime but also harm the final accuracy. We noticed that when we increased the batch size, the final validation accuracy became smaller, especially in the CIFAR-10 experiment. Figure 4.5 shows the decrease in the final accuracy for both problems. Keskar *et al.* [15] presented some explanations for this phenomenon, so we did not get deeper into this subject. It is essential to understand that choosing the correct batch size is vital to get a good performance in terms of runtime and accuracy.

Table 4.1 also reveals another aspect of our results: how poor our scalability was. The speedup column is normalized by the fastest configuration on the p2.xlarge instance, with a batch size of 512 images. The instance and configuration that achieved the highest speedup based on this value were the p3.8xlarge virtual machine and a batch size of 1024 images, respectively. As we can see in the Table 4.1, this combination was 3.8 times faster than the p2.xlarge instance best performance.

We may argue that the p3.8xlarge instance has 4 GPUs, so this speedup is good since it almost achieved the ideal one. However, the GPU type of these two instances is different, and the p3.8xlarge instance price (12.24 USD per hour) is 13.6 times higher than the p2.xlarge virtual machine instance, which costs 0.90 USD per hour. Therefore, this speedup was not close to the ideal one.

Table 4.1: Average runtime and cost of using each instance with a seismic image dataset.

Instance type	Batch size	Average time (s)	Speedup	Average cost (U$)
p2.xlarge	256	153.1 ± 44.2	0.79	0.04
	512	121.7 ± 26.3	**1.0**	0.03
	1024	134.3 ± 28.4	0.91	0.03
	2048	264.8 ± 49.4	0.46	0.07
p2.8xlarge	256	101.1 ± 46.6	1.20	0.20
	512	77.1 ± 23.5	1.58	0.15
	1024	65.1 ± 10.9	1.87	0.13
	2048	69.1 ± 15.3	1.76	0.14
p2.16xlarge	256	259.8 ± 52.6	0.47	1.04
	512	139.8 ± 36.6	0.87	0.56
	1024	112.8 ± 23.4	1.08	0.50
	2048	97.3 ± 6.5	1.25	0.39
p3.2xlarge	256	61.0 ± 44.2	2.0	0.04
	512	38.1 ± 7.8	3.19	0.03
	1024	36.8 ± 8.9	3.30	0.03
	2048	63.2 ± 8.5	1.92	0.05
p3.8xlarge	256	59.3 ± 53.7	2.05	0.05
	512	33.7 ± 6.3	3.61	0.11
	1024	32.0 ± 2.3	**3.8**	0.11
	2048	33.3 ± 3.1	3.66	0.11
p3.16xlarge	256	220.2 ± 39.9	0.55	1.50
	512	96.3 ± 17.4	1.26	0.65
	1024	81.2 ± 18.9	1.5	0.55
	2048	80.2 ± 18.4	1.52	0.55

The fairest comparison could happen with the p3.2xlarge and the p3.8xlarge instances because they have the same GPU type. The increase in the price of these instances is proportional to their respective number of GPUs. But, as we can see in Table 4.1, the performances of both instances were almost the same. Thus, we can conclude that using the p3.8xlarge virtual machine instance is not justified for this problem.

For the CIFAR-10 dataset, we achieved similar results. Table 4.2 shows the average time and the average cost for each batch size on each instance. We normalize the speedup column by the smallest value on the p2.xlarge instance. In this case, this happened with a batch of 512 images. The instance that provided the best speedup was the p3.8xlarge instance. As we mentioned before, this instance cost 13.6 times more than the p2.xlarge virtual machine instance and was only five times faster, so the increase in cost may not justify its use.

Unlike the seismic data experiments, the p3.8xlarge VM type performed better than the p3.2xlarge VM type (the cheapest one among the p3 family). On the one hand, the p3.8xlarge VM type was only 1.46 times faster than the p3.2xlarge. On the other hand,

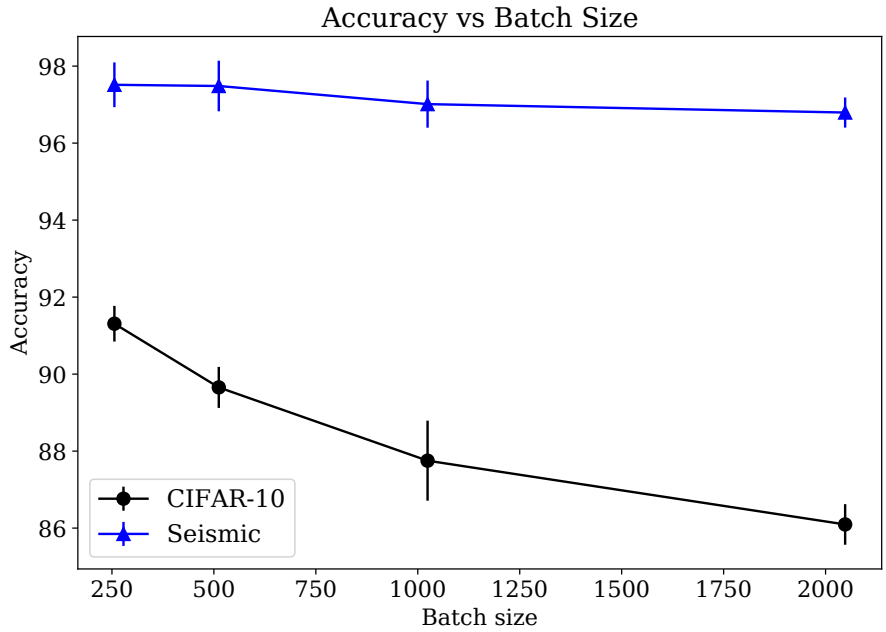

Figure 4.5: Final accuracy for seismic dataset and CIFAR dataset problems with different batch sizes.

despite costing 3.4 times more than the p2.xlarge VM type, this one performed 3.5 faster with a batch size of 1024 images, and the two instances had almost the same cost.

Figures 4.6 and 4.7 show the correlation between the cost and runtime for all the VM types when executing the seismic/FCN and CIFAR-10/ResNet-50 applications, respectively. For the seismic/FCN application, the p3.2xlarge and p3.8xlarge VM types offer similar performance in terms of runtime, but training with the p3.2xlarge VM type costs less. For the CIFAR-10/ResNet-50 application, the p3.8xlarge achieved the best performance among all instances. The p3.16xlarge VM type also offered good performance, but none of these instances achieved performance close to the expected, and their prices per hour make them more expensive than the p3.2xlarge VM type. So the p3.2xlarge VM type had the best cost-benefit correlation for this problem as well.

4.3 Description of the Proposed Methodology

During our experiments with the seismic/FCN and the CIFAR-10/ResNet-50 applications, we were able to see how expensive it may be to run a Deep Learning application on the cloud, especially if we choose the wrong instance. Simultaneously, we can not assume that the results we found, *i.e.*, the best VM types, will be the same for any other Deep Learning problem. This happens because the performance of a Deep Learning application on different computing systems depends on many variables. We also observed that the batch size has a significant influence on the application runtime and may harm the final model accuracy.

With these observations in mind, we propose a methodology to identify the best in-

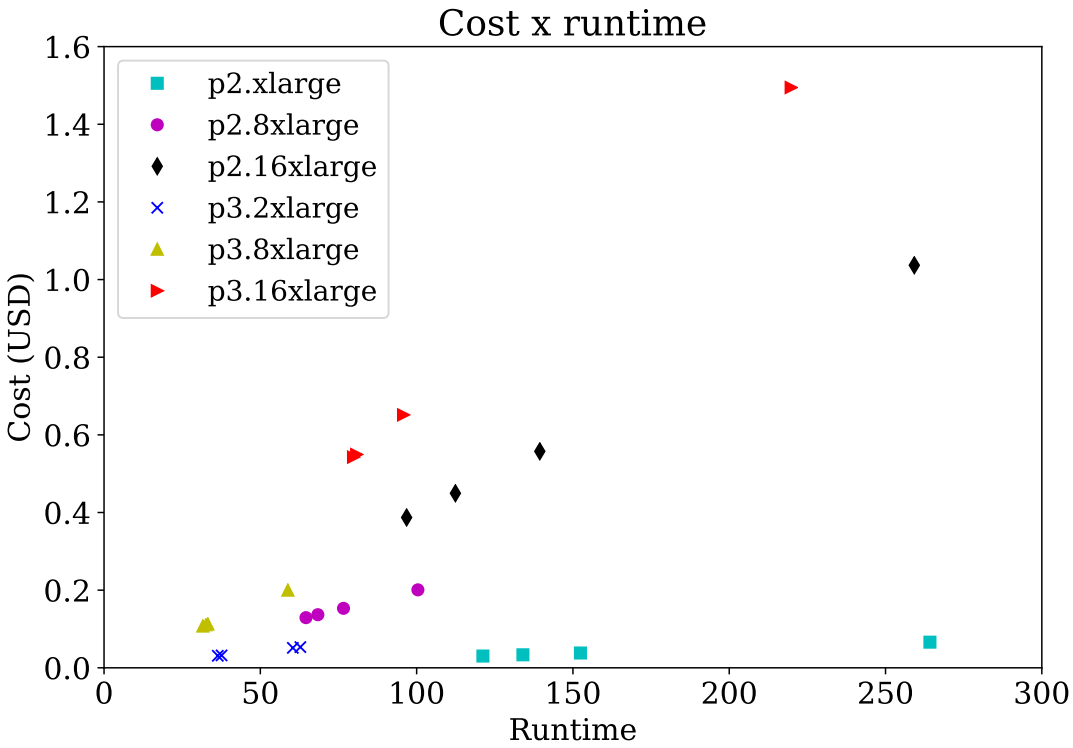

Figure 4.6: Cost *vs* runtime for the seismic dataset for all instances.

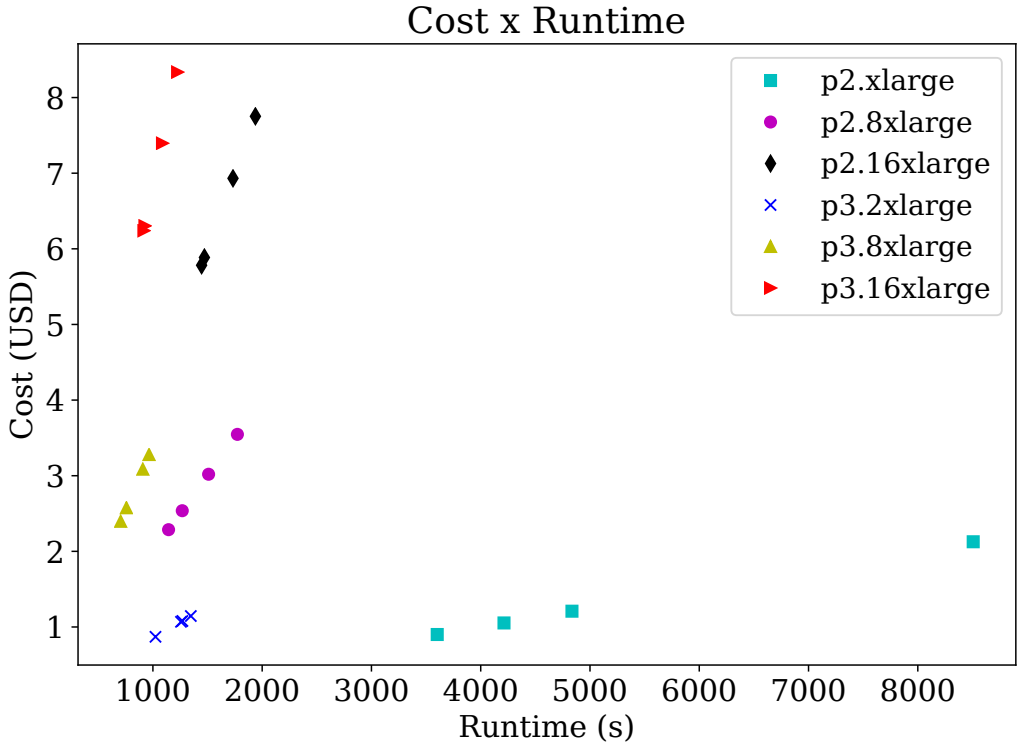

Figure 4.7: Cost *vs* runtime for the CIFAR-10 dataset for all instances.

Table 4.2: Average runtime and average cost for the CIFAR-10 dataset in all instances.The baseline for the speedup is the fastest configuration from p2.xlarge instance.

Batch size	Instance type	Average time (s)	Speedup	Average cost (U$)
p2.xlarge	256	4213.0 ± 7.9	0.85	1.05
	512	3602.0 ± 21.3	**1.0**	0.90
	1024	4835.0 ± 26.8	0.74	1.21
	2048	8507.0 ± 15.1	0.42	2.13
p2.8xlarge	256	1509.7 ± 93.9	2.4	3.02
	512	1143.3 ± 40.7	3.15	2.30
	1024	1268.7 ± 53.1	2.84	2.50
	2048	1773.3 ± 12.7	2.03	3.53
p2.16xlarge	256	1938.0 ± 86.2	1.86	7.78
	512	1471.3 ± 62.7	2.45	5.90
	1024	1445.7 ± 25.4	2.5	5.76
	2048	1733.0 ± 32.0	2.1	6.91
p3.2xlarge	256	1270.0 ± 1.7	2.84	1.07
	512	1257.0 ± 7.8	2.87	1.07
	1024	1023.7 ± 45.5	3.5	0.86
	2048	1347.0 ± 3.5	2.67	1.13
p3.8xlarge	256	907.7 ± 60.0	3.97	3.06
	512	705.0 ± 13.9	**5.11**	2.45
	1024	757.7 ± 10.0	4.75	2.57
	2048	964.7 ± 2.9	3.73	3.30
p3.16xlarge	256	1226.0 ± 16.7	2.94	8.32
	512	917.7 ± 33.9	3.93	6.10
	1024	927.0 ± 18.3	3.9	6.36
	2048	1087.7 ± 10.8	3.3	7.34

stance for a given Deep Learning problem, considering cost and runtime. This methodology uses the epoch time provided by a Deep Learning framework, and it takes into account the batch size variable to help the user choose the best configuration.

The first thing we observed was the fact that the epoch time is usually stable during the execution. Figure 4.8 shows an example of the epoch's time stability for the experiments made in a p2.xlarge instance with the seismic images dataset/FCN benchmark. The only exception is the first epoch that has a higher value than further epochs.

We also observed that the number of epochs required to stabilize the model accuracy depends on the batch size; nonetheless, given the same batch size, the application would take roughly the same amount of epochs to stabilize the model accuracy. This means that we can set the number of epochs necessary to stabilize the accuracy for a given batch size. Table 4.3 shows the number of epochs (and iterations) required to stabilize the model accuracy for each batch size and each application. This number is independent of the GPU type and if it is running on a single or a multi-GPU machine.

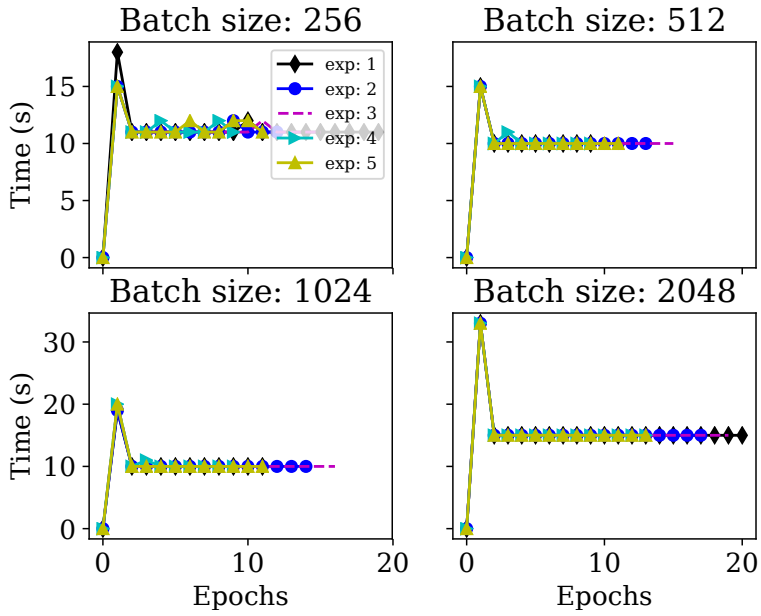

Figure 4.8: Epoch time for the seismic image dataset experiments for all instances.

Table 4.3: Number of epochs and iterations required for each batch size on both applications.

Application	Batch size	Epochs	Iterations
Seismic/FCN	256	13	2,067
	512	14	880
	1024	14	480
	2048	16	320
CIFAR-10/ResNet-50	256	2.7	17,000
	512	5.6	8,000
	1024	10	4,500
	2048	15	3,000

We conjectured that if we know the time of the first two epochs and the number of epochs necessary for each batch size, we can estimate the application's runtime. Based on this estimate, we can discover which instance has the best cost-benefit correlation. The remaining question is: how many epochs are required to train the model for a given batch size properly? There is no obvious answer to that because, as we explained before, it depends on the problem that we are dealing with.

Our proposal to answer this question is to run the application for all potential batch sizes in the cheapest VM type and measures the number of epochs that each batch size requires to stabilize the accuracy. Then, we apply this number to estimate the application's runtime on other VM types. We describe our methodology as follows:

1. Train the model in the cheapest VM type with all suggested batch sizes until the validation accuracy stabilizes;

2. Compute the average number of epochs required for each batch size;

3. Train the model again on other instance types but execute it for only two epochs;

4. Estimate the training runtime with the respective dataset by using Equation 4.1, where $T(Ep_1)$ is the time spent in the first epoch, N_{ep} is the average number of epochs to stabilize the accuracy for the given batch size, and $T(Ep_2)$ is the time spent in the second epoch;

5. Finally, we calculate the cost of using each experiment by multiplying the estimated execution time by the VM type price.

$$Total\ runtime = T(Ep_1) + (N_{ep} - 1) \times T(Ep_2). \tag{4.1}$$

4.4 Methodology's Validation

We validate the methodology presented in the previous section using the seismic/FCN and a new application, the MNIST/Simple-CNN, which trains a simple two-layer CNN to classify images from the MNIST dataset [8]. As the p2.xlarge is the cheapest VM type, we chose it to run the applications until the end to find the number of epochs for each batch size. Then we execute the application for two epochs in the other instances. Figure 4.9 shows the comparison between the real runtime and the estimated runtime for the seismic dataset. Notice that the estimated runtime was close to the real one. Figure 4.10 shows the estimated comparison between cost and runtime and shows that the methodology put the p3.2xlarge as the best instance for this problem, exactly as we saw with the previous experiments.

Figure 4.11 shows the results for the validation with the MNIST/Simple-CNN application. Notice that the estimation can predict the best instance for this case as well.

The CIFAR-10/ResNet-50 application, differently from the seismic/FCN and the MNIST/Simple-CNN applications, had its algorithm implemented with the TensorFlow framework, which provides a different kind of output that did not allow us to apply this methodology without changing its code. We intend to address this limitation in future work.

4.5 Cost of Applying the Proposed Methodology

One of the main advantages of this methodology is that it can discover the best instance by running only two training epochs in most of the VM types. Tables 4.4 and 4.5 show the difference in the cost of doing the experiments without the methodology and with the methodology. See that, for the MNIST/Simple-CNN application, the methodology achieved an economy of 62.2%, while for the seismic/FCN application, it achieved spent 86.4% less than just run the application entirely in each VM instance. This economy can be even higher with large datasets and networks.

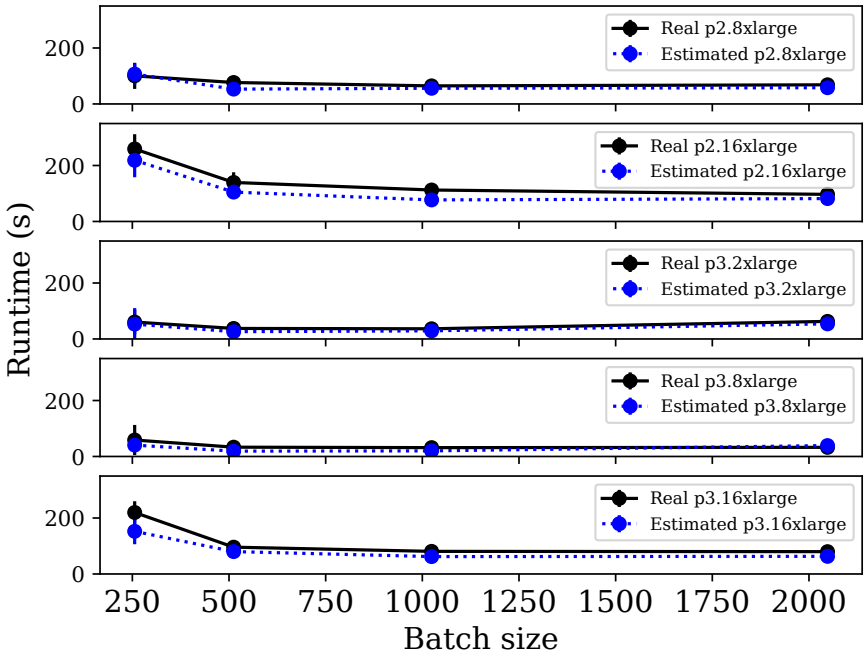

Figure 4.9: Comparison between the real runtime and the estimated runtime for the seismic dataset experiments.

4.6 Conclusions

Starting from the premise that GPUs are currently popular and one of the best hardware to train a Deep Learning algorithm, we focused our analyses on GPU-based VM types. However, based on our results, we can not assume that the VM type with more GPUs will bring the smallest runtime for a Deep Learning application. At the same time, the VM type that has the smallest price per hour does not always provide the smallest cost for execution.

Our work also shows that instances with V100 GPUs have better performance than the K80 GPUs, and this makes the single-GPU p3.2xlarge more indicated for our benchmarks because it brought small runtime with one of the lowest costs among all instances studied. The p3.2xlarge instance provided smaller runtime than instances with 8 GPUs, for example. This happened because the current Deep Learning algorithms used in our experiments exhibited poor scalability when training the models on multi-GPU environments. In our experiments, we could observe poor scalability for our three benchmarks that used the synchronous parallelization strategy with Keras and TensorFlow frameworks. Our results also indicated that increasing the batch size could reduce the total runtime, but this is not always true. Hence, we conclude that it is always important to test the batch sizes to verify which one is more indicated to the given problem.

Based on all these observations, we proposed a methodology to identify the EC2 VM types' cost-benefit correlation for Deep Learning problems. We build this methodology by verifying that the epoch time is stable during the networks' training, and the maximum number of epochs is dependent only on the batch size. Therefore, knowing the number of

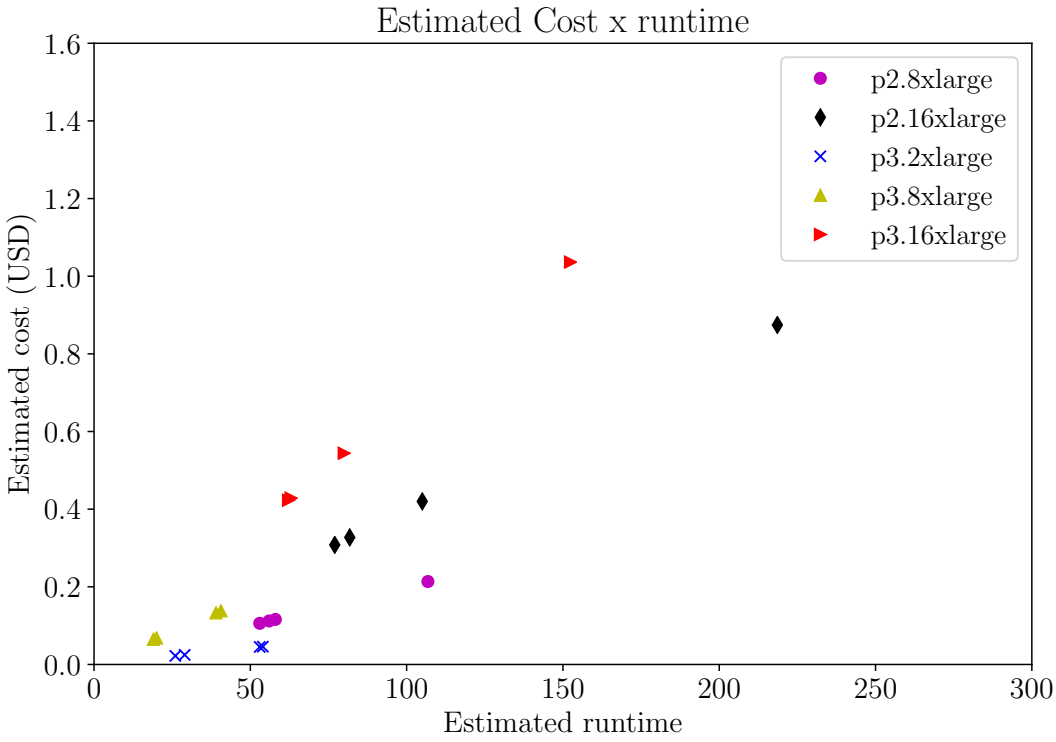

Figure 4.10: Estimated runtime *vs* estimated cost for the seismic dataset experiments.

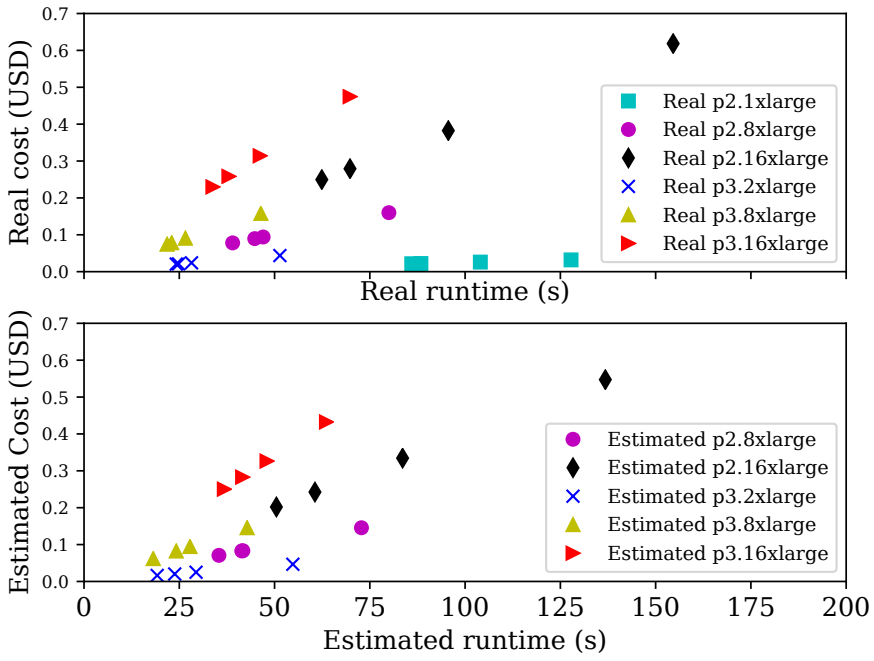

Figure 4.11: Comparison between the real runtime and the estimated runtime for the MNIST dataset experiments.

Table 4.4: Cost of performing the seismic data experiments with and without the proposed methodology.

Instance	**Cost without the methodology (U$)**	**Cost with the methodology (U$)**
p2.xlarge	0.84	0.84
p2.8xlarge	3.10	0.79
p2.16xlarge	12.20	1.41
p3.2xlarge	0.84	0.60
p3.8xlarge	2.66	0.50
p3.16xlarge	16.20	1.12
Total	**35.84**	**5.26**

Table 4.5: Cost of performing the MNIST data experiments with and without the proposed methodology.

Instance	**Cost without the methodology (U$)**	**Cost with the methodology (U$)**
p2.xlarge	0.51	0.51
p2.8xlarge	2.11	0.62
p2.16xlarge	7.65	2.30
p3.2xlarge	0.55	0.20
p3.8xlarge	2.00	0.80
p3.16xlarge	6.39	2.84
Total	**19.21**	**7.27**

epochs required by each batch size configuration and the time of the first two epochs on each VM type and batch size, we can estimate the application's runtime and cost among instances. We understand that this methodology contributes to the scientific community since it can help Machine Learning researchers to reduce the cost of experiments by choosing the best VM types and the best batch sizes for each Deep Learn problem.

However, the proposed methodology has some limitations. The first one is that we still need to run the application until its stabilization at least once for each batch size tested. This step is needed to identify the number of epochs necessary to stabilize the problem's validation accuracy. This limitation could be addressed in the future by techniques that try to predict the minimum number of epochs to stabilize the accuracy by inspecting the accuracy achieved in the first epochs, for example.

Another branch that can be explored from this work is to find out if this methodology can be applied for other Machine Learning problems. We understand that it is possible to do that for all cases where we have a repetition of a step (such as an epoch), in which time is stable during the entire execution. So it would be interesting to validate this hypothesis in future work as well.

Chapter 5

Selecting the Best GPU Instance through the Iteration Time

In the previous chapter, we discussed a methodology that relies on discovering the number of epochs to stabilize the accuracy for each batch size. We can then determine which instance is the fastest one and the optimum batch size configuration to achieve the shortest runtime. However, this methodology has the limitation of needing to run the application entirely at least one time. Then, we must run it on other virtual machine instances for two epochs. Running the whole training process may be too expensive. Also, when dealing with increasingly large datasets, running two the training process for epochs may also be a costly operation.

In this chapter, we introduce another approach to determine the best virtual machine type to train a Deep Learning model. This approach works by presuming that the users already know the batch size they want to use, and there is no need to train the model to its completion to find out how many epochs each batch size requires. Also, this approach uses the iteration's times as the core in determining the VM types that offer the best cost-benefit correlation.

This chapter is organized in four sections: Section 5.1 describes the experimental setup. Section 5.3 introduces the new approach and formalizes it. Section 5.4 presents the methodology's validation applying it to our case studies. Finally, Section 5.6 concludes the chapter with our observations and future work proposal.

5.1 Experimental Setup

The experiments presented in this chapter rely on the seismic/FCN and the MNIST/Simpl e-CNN applications, previously introduced in Chapter 4. We chose these applications because it is user friendly to use and instrument them to report the iteration times using the Keras framework. Besides, Keras is one of the most popular frameworks used by the Deep Learning community [24]. We intend to use benchmarks implemented with other frameworks in future work. So we did not use the CIFAR/ResNet-50 benchmark in these experiments.

As discussed in the previous chapter, the MNIST/Simple-CNN application trains a

simple two-convolutional layer model to classify images from the MNIST dataset [20], a well-known set of images of handwritten digits. Figure 5.1 shows a MNIST dataset example. We used the Dropout regularization [36] and the Adadelta optimizer [28], starting with a learning rate of 0.001.

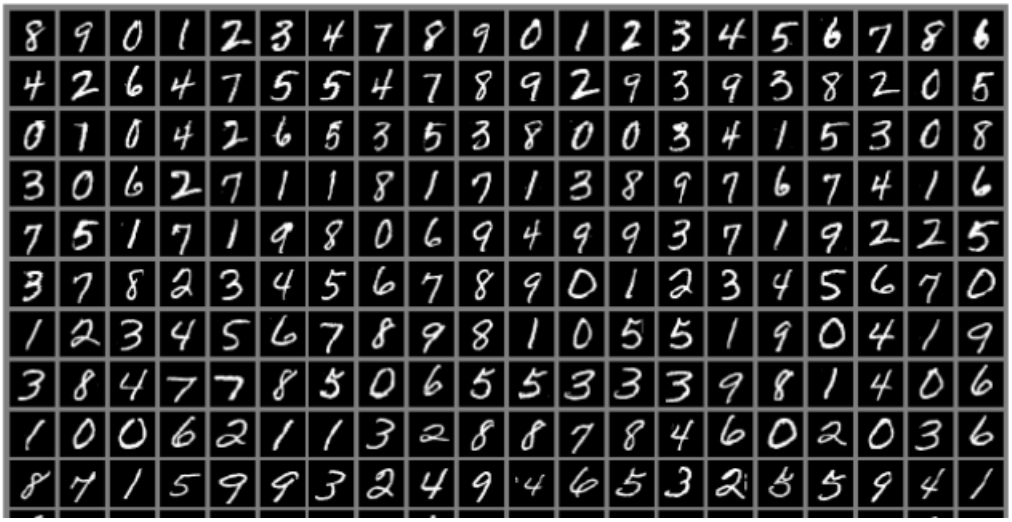

Figure 5.1: MNIST dataset sample. Figure reproduced from Goodfellow *et al.* [10].

To test and analyze this new approach, we run the MNIST/Simple-CNN application five times and the seismic/FCN application three times in each VM type analyzed. We tested both applications with batches of 256, 512, 1024, and 2048 images. We no longer employ the Early Stopping callback, as we did in the previous chapter. Instead, we set the number of epochs for each experiment and batch size based on the previous results, since we saw that the number of epochs necessary to stabilize the accuracy changes only based on the problem and the batch size, and it is not dependent on the GPU type. Table 5.1 shows the number of epochs required to train each model with the selected batch sizes.

Table 5.1: Number of iterations and epochs for each batch size for seismic and MNIST datasets. The iterations column is the number of iterations in each epoch.

Dataset	**Batch size**	**Epochs**	**Iterations**
Seismic	256	13	2,067
	512	14	880
	1024	14	480
	2048	16	320
MNIST	256	15	195
	512	17	98
	1024	22	49
	2048	28	25

We add custom callbacks to the applications' codes to measure the iteration and validation times during the program execution. We also measured the initialization time, which is the time spent by the program with instructions that precede the training function. With the output logs, we analyze the final runtime of the experiment and its cost.

Table 5.2: Comparison between the real runtime of an application with the estimated runtime based on Equation 4.1 for the p2.xlarge instance with seismic data experiments.

Batch size	First epoch (s)	Average epoch time (s)	Initialization time (s)	Epochs	Real time(s)	Estimated time(s)	Error (%)
256	13.69	11.05	0.18	13	148.15	146.47	1.15
512	14.22	9.96	0.19	11	116.86	114.01	2.5
1024	18.95	10.05	0.20	12	132.73	129.7	2.34
2048	33.06	15.6	0.19	16	268.75	267.25	0.56

The final runtime is the total training time, and we calculate the cost by multiplying the runtime by the VM type's price per hour. We base our cost-benefit analysis on these two variables.

5.2 Describing the Time-consuming Tasks of a Deep Learning Problem

As discussed in Chapter 2, the training of Deep Learning models is performed in epochs. Each epoch, in turn, is composed of iterations and the validation steps. Each iteration step processes a batch of dataset items at a time, while the validation step happens at the end of each epoch, and it calculates the model accuracy based on the validation dataset.

During our experiments, we noticed that the time required to perform each epoch is usually very stable during the whole execution, except for the first epoch. Therefore, we concluded that we can calculate the runtime of a Deep Learning application with the time of the first two epochs as we explained in the previous chapter (see Equation 4.1). For the current proposal, we also considered the initialization time. This variable corresponds to the time spent by the program executing instructions before the first epoch starts. We decided to take this variable into account because it can be significant for multi-GPU environments. Thus, we update Equation 4.1 into Equation 5.1. In this case, $T(Init)$ corresponds to the initialization time.

$$Total\ runtime = T(Init) + T(Ep_1) + (N_{ep} - 1) \times T(Ep_2). \tag{5.1}$$

Table 5.2 shows the validation of this equation in a p2.xlarge instance for the seismic/FCN application. The error column shows the error percentage of the estimated runtime, based on Equation 5.1, when compared to the real runtime. Therefore, the lower the error value, the more accurate the estimate is. As the error values are all smaller or equal to 2.5%, we concluded that this estimation is close to the real one.

We could also see that the iteration and the validation times are stable during the entire execution, except for the first iteration of the first epoch and the validation of the first epoch. Figure 5.2 shows that the iteration time from the second iteration was stable during the execution.

Figure 5.3 shows the execution time of each validation for the seismic/FCN application running on a p2.xlarge VM type. Notice that, except for the first validation, all validations take the same amount of time to execute.

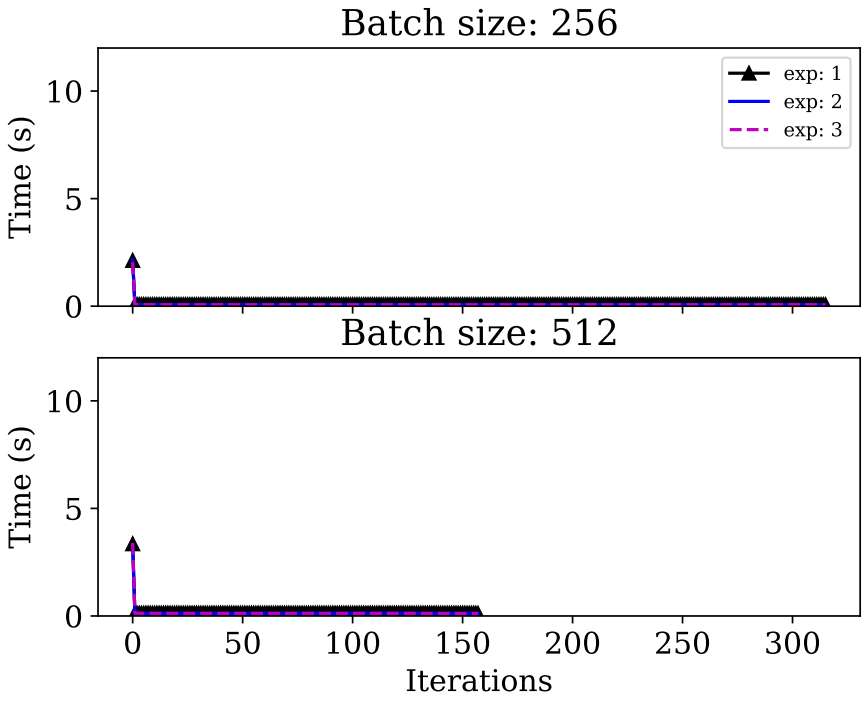

Figure 5.2: Iteration per time in seconds for the seismic data experiments on a p2.xlarge instance for batch sizes of 256 and 512 images.

As one epoch is composed of iterations plus the validation step, we can estimate an epoch time with the iteration and validation times. Assuming the iterations and validation times are stable, we can estimate an epoch time using only the execution time of the first iteration and the validation steps. Equation 5.2 shows how the n^{th} epoch time ($T(E_n)$) can be computed based on the number of iterations (N_{it}), the expected iteration time on epoch n ($T(E_nIt_1)$) and the expected validation time on epoch n ($T(Val_n)$).

$$T(E_n) = T(E_nIt_1) \times (N_{it}) + T(Val_n) \tag{5.2}$$

Since only the execution time of the first iteration ($T(E_1It_1)$) and the first validation ($T(Val_1)$) differs from the others, if we know the initialization time ($T(Init)$), the execution time of the first two iterations ($T(E_1It_1)$ and $T(E_1It_2)$) and the first two validations ($T(Val_1)$ and $T(Val_2)$), we can derive an equation that computes the total execution time ($Total\ runtime$) as a function of the number of epochs as follows:

$$\begin{aligned} Total\ runtime = & \overbrace{T(E_1It_1) + T(E_1It_2) \times (N_{it} - 1) + T(Val_1)}^{\text{First epoch execution time}} + \\ & (N_{ep} - 1) \times \underbrace{T(E_1It_2) \times N_{it} + T(Val_2)}_{\text{Execution time of other epochs}} \end{aligned} \tag{5.3}$$

Therefore, to estimate the runtime of a Deep Learning application, we basically need to know the time of the first two iterations of the first epoch and the validation time of the first and second epochs.

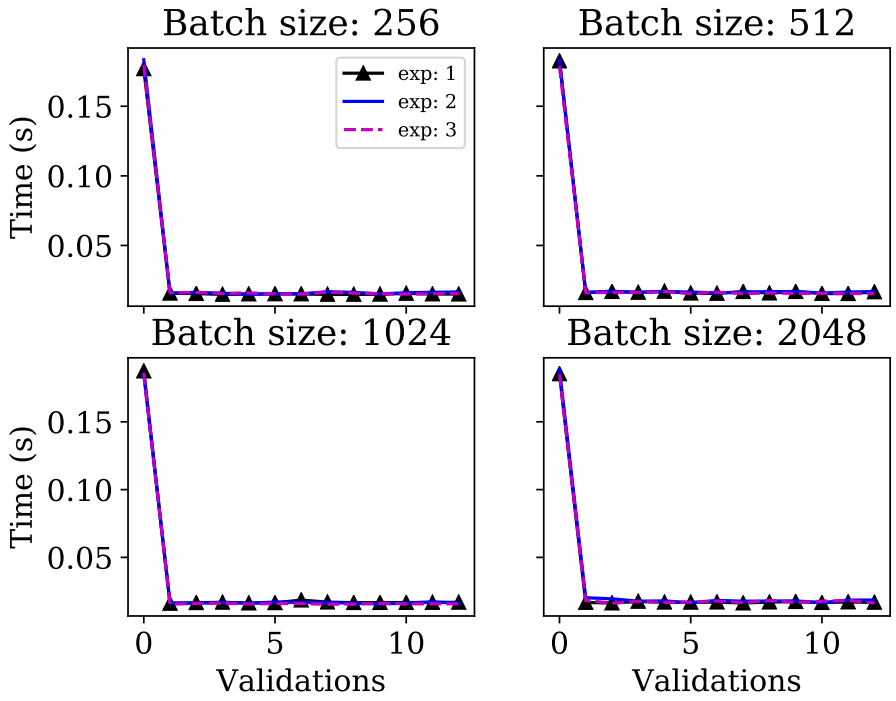

Figure 5.3: Time per validation for the seismic data experiments on a p2.xlarge instance.

We applied Equation 5.3 to predict the runtime of the seismic/FCN application. Table 5.3 shows how the runtime estimated with Equation 5.3 is close to real runtime for the p2.xlarge VM type. For this calculation, the validation time of the first epoch was 0.17 seconds for batch sizes equal to 256 and 512, and 0.18 seconds for 1024 and 2048 images. The validation of the second epoch was approximately 0.2 seconds for all batch sizes. We can see the other variables' values in Table 5.3.

Table 5.3: Comparison between the real runtime of an application with the estimated runtime based on Equation 5.3, that takes into account the iterations time, for the p2.xlarge instance with seismic data experiments. The values in the table are the average execution time of three experiments done for each batch size.

Batch size	**Initialization time (s)**	**First iteration (s)**	**Second iteration (s)**	**Real time(s)**	**Estimated time (s)**	**Error (%)**
256	0.63	14.0	0.08	195.85	170.6	14.8
512	0.19	3.41	0.13	145.67	148.95	2.20
1024	0.20	5.62	0.25	148.0	144.4	2.51
2048	0.18	10.2	0.74	248.5	243.95	1.85

The error column shows the percentage of how far the estimated time is from the real runtime. Using both the iteration and the validation times brought a higher error to our estimate when compared with the estimates performed using the epoch time, presented in Table 5.2, especially for a batch size for 256 images. However, since the goal is to determine the best instance and not predict the exact application runtime, we believed that the error margin would be enough to accomplish our goal, and we test this hypothesis

further in this research.

The high presented error with a batch of 256 images is caused by a high variance in the first iteration execution time, expressed through a standard deviation of 20.5 seconds. This variation happens because the first iteration of the first experiment in one VM instance usually takes longer to execute. We did not make experiments to discover why this happened, but we intend to investigate this in future work.

The validation time for all experiments performed was short, even in the first epoch, where it has its highest value. As a consequence, we expected it to have little influence on our estimation. In this sense, we did the same estimation that we did in Table 5.3 but discarding the validation time. Table 5.4 shows the results. When we compare the error column of this Table with the error of Table 5.3, we see that the validation time has little influence on the final runtime. The validation time is expected to be shorter than the sum of the iterations time for two reasons: 1) the validation dataset is usually smaller than the training dataset, and 2) operation performed for each sample on the validation step requires only the forward propagation pass, while the operation performed for each sample on the iteration step requires both the forward and the backward propagation pass. Because of this, we decided to use Equation 5.4 to go further in our analysis.

$$Total\ runtime = T(Init) + T(It_1) + \sum_{m=1}^{N_{ep}} \sum_{n=1}^{N_{it}} T(It_n). \tag{5.4}$$

Table 5.4: Comparison between the real runtime of an application with the estimated runtime based on Equation 5.3, that takes into account the iterations time, for the p2.xlarge instance with seismic data experiments without using the validation time. The values in the table are the average execution time of three experiments done for each batch size.

Batch size	**Initialization time (s)**	**First iteration (s)**	**Second iteration (s)**	**Real time(s)**	**Estimated time (s)**	**Error (%)**
256	0.63	14.0	0.08	195.85	170.3	13.05
512	0.19	3.41	0.13	145.67	148.6	1.99
1024	0.20	5.62	0.25	148.0	144.0	2.70
2048	0.18	10.2	0.74	241.7	243.5	1.99

We also estimate the total runtime based on the average time of iterations 2 to 10, *i.e.*, $\frac{\sum_{i=2}^{10} T(E_1 It_i)}{9}$, to check whether the average would make a better prediction of the real runtime by dissolving any distortion that may happen in one single iteration's time. The results are shown in Table 5.5. Except for the batch size of 256 images, the remaining errors were close to the previous estimation.

5.3 Proposed Methodology's Description and Formalization

Once we already know that we can estimate the real runtime of a Deep Learning application, we want to use this information to discover the performance and cost correlation

Table 5.5: Comparison between the real runtime of an application with the estimated runtime based on Equation 5.3, that takes into account the average time of iterations the 2 to 10, for the p2.xlarge instance with seismic data experiments without using the validation time. The values in the table are the average execution time of three experiments done for each batch size.

Batch size	**Initialization time (s)**	**First iteration (s)**	**Average iteration (s)**	**Real time(s)**	**Estimated time (s)**	**Error (%)**
256	0.63	14.0	0.067	195.85	151.7	29.1
512	0.19	3.41	0.126	145.67	142.5	2.19
1024	0.20	5.62	0.249	148.0	143.4	3.22
2048	0.18	10.2	0.733	248.5	241.7	2.78

among all VM types. For example, we wonder if the instance that has the shortest second iteration time will be necessarily the fastest. Furthermore, if the second iteration of a virtual machine instance x be two times higher than the second iteration of the fastest instance, the instance x is two times slower than the fastest virtual machine instance. Therefore, we performed some experiments to verify which combination would correspond to a better representation of the real correlation among all instances. We tested the following options.

1. Estimate the VM type performance using the time of the first iteration. We tested this option to verify whether, despite not representing the execution time of the remaining iterations, it could still produce a value that represents a good performance correlation among instances;

2. Estimate the VM type performance using the time of the second iteration. We wanted to check if the second iteration is enough to determine the best instance;

3. Estimate the VM type performance using the average time of iterations two to ten. If the second iteration presents some distortion, we want to test if this average of a few iterations could present a better representation of the real runtime.

Table 5.6 shows the comparison among the options above mentioned. Each column is normalized by its smallest value. Hence, the fastest VM type has the value of 1.00, and the others have values that indicate how slower they are when compared to the fastest one. The final column serves as the baseline for the comparison since it represents the real correlation using the application's real runtime. This table reveals that just using the iteration times was not accurate in determining the fastest instance for this application. The first iteration time did not work because it presents a significant distortion compared with the further iterations. The second and the average iteration times also did not work because they did not consider the initialization and the first iteration time; in our problem, these variables have a significant influence on the final runtime.

Because of this impasse, we thought that we could combine the importance of the iteration time from the second iteration, but that could also consider the initialization and the first iteration times in the estimate. Therefore, we decided to make the estimation based on all these variables by simulating the application execution.

Table 5.6: Comparison of strategies to determine the best instance for a batch size of 256 images for the FCN/seismic data experiments.

VM type	Relative performance using			
	First it.	**Second it.**	**Average**	**Real**
p2.xlarge	1.00	5.34	6.00	4.01
p2.8xlarge	4.27	3.82	4.00	3.07
p2.16xlarge	14.95	6.87	8.00	7.57
p3.2xlarge	1.10	1.15	1.40	1.00
p3.8xlarge	7.94	1.00	1.00	2.09
p3.16xlarge	10.28	1.91	3.00	2.87

Figure 5.4 shows the simulation of the behavior of the seismic/FCN application for different batch sizes. Note that if the application requires only one epoch to train the model, the p2.xlarge VM type would be one of the fastest. With five epochs, the expected correlation would still not be evident. Only after 25 epochs, the VM types performance correlation would be more definitive. We based our methodology on the principle that users do not know the number of epochs they need. So we chose a set of numbers of epochs to test. For this analysis, we will simulate the execution for 5, 10, and 20 epochs to discover which one will be related to our real runtime.

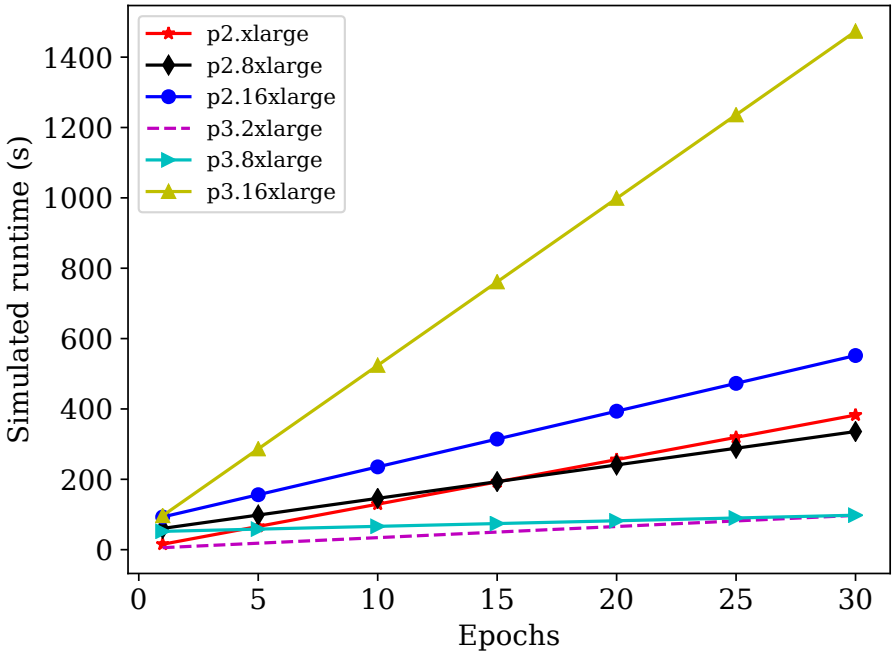

Figure 5.4: Simulated runtime for the seismic data experiments with batch size of 256 images.

We can see in Table 5.7 the performance of the three tests. Notice that the tests with 10 and 20 epochs were the closest to the real correlation. This result makes sense because the larger number of epochs amortize the initialization and first iteration time.

Appendix A shows all experimental results for the seismic/FCN application.

Table 5.7: Instances correlation of the simulated runtime for a batch size of 256 images for the seismic/FCN experiments.

VM type	Relative performance using			
	5 epochs	**10 epochs**	**20 epochs**	**Real runtime**
p2.xlarge	4.07	4.34	4.49	5.27
p2.8xlarge	6.05	4.81	4.11	4.47
p2.16xlarge	9.95	8.15	7.12	7.48
p3.2xlarge	1.00	1.00	1.00	1.00
p3.8xlarge	3.97	2.56	1.75	2.06
p3.16xlarge	4.54	3.23	2.48	2.84

Table 5.8 shows the application of these strategies to discover which instance has the smallest cost. The strategies that simulate the execution with 10 and 20 epochs had the best correlation with the real cost.

Table 5.8: Comparison of strategies to determine the instance that has the best cost for a batch size of 256 images for the seismic/FCN experiments.

VM type	Relative cost using			
	5 epochs	**10 epochs**	**20 epochs**	**Real runtime**
p2.xlarge	1.20	1.28	1.32	1.55
p2.8xlarge	14.2	11.3	9.66	10.5
p2.16xlarge	46.8	38.3	33.5	35.2
p3.2xlarge	1.00	1.00	1.00	1.00
p3.8xlarge	15.9	10.2	7.02	8.26
p3.16xlarge	36.3	25.8	19.9	22.7

Based on these observations, we can describe the proposed methodology as follows.

1. Run the application for a few iterations in all instances. Notice that the execution time of each iteration depends on the batch size, not the size of the entire training dataset. Hence, this step should not take cost much to perform on each VM type;
2. Take the initialization, first iteration, and second iteration times;
3. Estimate the application's runtime for three (or more) options of total epochs (*e.g.* 5, 10 or 20 epochs) using Equation 5.4.
4. List the estimated runtime for all options in a table and normalize the columns by the smallest value;
5. Do the same for the cost analysis by multiplying the estimated runtime per the instance's price per hour.

In the following section, we validate this methodology with the MNIST/Simple-CNN application. We also validate it with other batch sizes for the seismic/FCN application, which we show in Appendix A.

5.4 Methodology's Validation

In this section, we validate the methodology proposed in the previous section using the MNIST/Simple-CNN application. We tested the same assumptions and options that we did for the seismic/FCN application. Table 5.9 shows the comparison between the estimated runtime that takes into account the iteration time and the initialization time with the real runtime of this application. We also investigate the importance of using the validation time in this application. The error column shows the percentage difference between the estimated runtime and the real runtime.

Table 5.9: Comparison between the real runtime of an application with the estimated runtime based on Equation 5.3, that takes into account the iterations time, for the p2.16xlarge instance with MNIST experiments. The values in the table are the average execution time of three experiments done for each batch size.

With validation	**Batch size**	**Initialization time**	**First it.**	**Second it.**	**Real time**	**Estimated time**	**Error (%)**
Yes	256	23.2	23.3	0.03	167.17	167.17	0.14
	512	6.94	21.6	0.03	99.4	95.77	3.79
	1024	6.93	21.6	0.04	83.2	81.53	2.04
	2048	6.92	21.7	0.05	75.6	72.94	3.65
No	256	23.2	23.3	0.03	167.17	156.34	7.07
	512	6.94	21.6	0.03	99.4	89.07	11.59
	1024	6.93	21.6	0.04	83.2	75.93	9.57
	2048	6.92	21.7	0.05	75.6	68.1	11.0

Notice that the estimation error is smaller when using the validation time. This happens because, for this application, the validation time is more significant to the application's final runtime. When compared to the seismic/FCN application, the validation time is more expressive in the MNIST/Simple-CNN application because the size of its validation dataset (10,000 samples) is 20% of the size of its training dataset (50,000 samples), while the size of the seismic/FCN's validation dataset (212 samples) is approximately 0.5% of its training dataset (40,512 samples). It is worth noting that one may reason about the influence of the validation step on the application's runtime by comparing the validation and the training dataset sizes. For these experiments, we kept the MNIST's validation set with 10,000 samples to check how this amount can influence our analysis.

Regarding the instances correlation, we calculated it similarly to what we did with the seismic data experiments by simulating executions with 5, 10, and 20 epochs. These simulations used only the initialization and first and second iteration times, and we analyzed the influence of the validation time.

Table 5.10 shows the results of simulating an execution with 5, 10, and 20 epochs to determine the best instance for the MNIST/Simple-CNN benchmark. All columns are normalized by the smallest value, and the real runtime column contains the correct performance relationship, computed using the real runtime. The upper part of the table uses Equation 5.3 to do the estimation, while the bottom part uses the Equation 5.4 disregarding the validation time, as proposed in our methodology.

Table 5.10: Comparison of strategies to determine the best instance for a batch size of 256 images for the MNIST dataset.

With validation	**VM type**	Relative performance using **5 epochs**	**10 epochs**	**20 epochs**	**Real runtime**
Yes	p2.xlarge	2.79	3.23	3.57	2.58
	p2.8xlarge	3.84	3.41	3.07	2.24
	p2.16xlarge	4.96	4.71	4.51	3.35
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	2.11	1.76	1.49	1.13
	p3.16xlarge	2.96	2.55	2.23	1.69
No	p2.xlarge	2.80	3.26	3.62	2.58
	p2.8xlarge	3.87	3.44	3.09	2.24
	p2.16xlarge	4.92	4.66	4.44	3.35
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	2.14	1.79	1.50	1.13
	p3.16xlarge	2.97	2.55	2.22	1.69

The simulation with 20 epochs was the only one that could determine precisely the order of each VM type, from the fastest to the slowest one. Nonetheless, all of the approaches were able to identify the best VM type for this application, *i.e.*, the p3.2xlarge. Besides, comparing the results with and without the validation time, we can see that using the validation time did not significantly improve the prediction, even for this problem. We found similar results for the other batch sizes as well.

Regarding the cost prediction, the strategy that simulates an execution with ten epochs was the most accurate in determining the cheapest instance, as we can see in Table 5.11. The simulation with 20 epochs did not work well for this task, as it pointed out the p3.2xlarge as being the cheapest VM type, which is, in fact, 1.32 times more expensive than the p2.xlarge VM type. On the other hand, the simulation with 20 epochs offered the most accurate estimates for the other instances' correlation. These results suggest that it may be essential to carefully analyze each case and perform multiple simulations to understand how the number of epochs may influence the final decision.

With a batch size of 2048 images, we have a result that shows the importance of testing several number of epochs. Figure 5.5 shows that the instances correlation is more defined after 25 epochs; before this point, there was no definition between the p3.2xlarge and p3.8xlarge VM types. Because of this, we simulate the execution with 10, 20, and 25 epochs. Table 5.12 shows the results for the instances comparison based on these options. Notice that only the simulation with 25 epochs could correctly determine the fastest instance.

These results can make us think that it is worth using several epochs that guarantees the dissolution of the distortion that may appear in the initialization and first iteration times. However, there are cases, such as in the seismic/FCN application, that the total number of epochs cannot cause this dissolution, so these values influence the final runtime, and remove them may harm the final prediction.

Table 5.11: Comparison of strategies to determine the instance that has the smallest cost for a batch size of 256 images for the MNIST dataset.

With validation	**VM type**	Relative performance using **5 epochs**	**10 epochs**	**20 epochs**	**Real runtime**
Yes	p2.xlarge	1.00	1.00	1.05	1.00
	p2.8xlarge	10.99	8.45	7.23	6.97
	p2.16xlarge	28.4	23.4	21.2	20.8
	p3.2xlarge	1.22	1.05	1.00	1.32
	p3.8xlarge	10.3	7.43	5.95	5.96
	p3.16xlarge	28.8	21.5	17.8	17.9
No	p2.xlarge	1.00	1.00	1.06	1.00
	p2.8xlarge	11.05	8.44	7.26	6.97
	p2.16xlarge	28.1	22.9	20.9	20.8
	p3.2xlarge	1.21	1.04	1.00	1.32
	p3.8xlarge	10.4	7.47	6.01	5.96
	p3.16xlarge	28.9	21.3	17.7	17.9

Table 5.12: Comparison of strategies to determine the best instance for a batch size of 2048 images for the MNIST dataset.

With validation	**VM type**	Relative performance using **5 epochs**	**10 epochs**	**20 epochs**	**Real runtime**
Yes	p2.xlarge	4.53	4.88	5.15	5.27
	p2.8xlarge	2.48	2.10	2.09	2.04
	p2.16xlarge	4.03	3.00	2.87	2.98
	p3.2xlarge	1.00	1.00	1.04	1.10
	p3.8xlarge	1.37	1.04	1.00	1.00
	p3.16xlarge	2.32	1.57	1.45	1.46
No	p2.xlarge	4.55	4.91	5.20	5.27
	p2.8xlarge	2.49	2.08	2.07	2.04
	p2.16xlarge	4.08	2.99	2.86	2.98
	p3.2xlarge	1.00	1.00	1.04	1.10
	p3.8xlarge	1.40	1.04	1.00	1.00
	p3.16xlarge	2.39	1.59	1.47	1.46

Regarding the cost prediction, Table 5.13 shows the results of using the previously mentioned strategies. For the cost prediction, all strategies correctly revealed the cheapest instance. However, the simulation with 25 epochs was more precise than the others. Appendix B shows all results for the MNIST dataset experiments.

5.5 Cost of Applying the Proposed Methodology

Regarding the decrease in cost to perform the experiments with the proposed approach, we did an estimation that considers the first and second iteration runtime. Table 5.14 shows

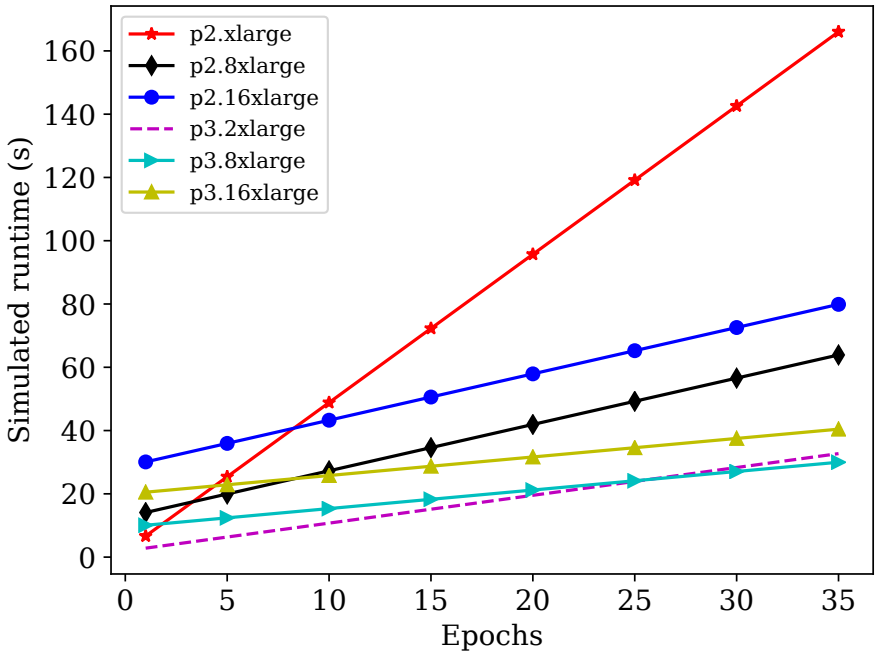

Figure 5.5: Simulated runtime for the MNIST data experiments with batch size of 2048 images.

the results. The "no methodology" column shows the cost to identify the instances' cost-benefit correlation by running the applications entirely on each instance. Methodology 1 is the methodology shown in Chapter 4, while Methodology 2 was the one presented in this chapter. This estimation does not consider the time it takes to boot the virtual machine. See that the methodology presented in this chapter was the cheapest option to do the experiments, as expected. It was 83.8% cheaper than the naïve option and 64% cheaper than the previous chapter's methodology.

Despite the evident economy that this methodology brings, we believe that the economy is even bigger for problems with large datasets. Because, in those cases, running the application entirely can take days, and one epoch can last for hours, so running it for a few iterations would be significantly cheaper.

5.6 Conclusions

The approach presented in this chapter can predict the performance and cost relationship of multiple VM types to train a Deep Learning model based only on the execution time of the first two iterations. This approach is an alternative to our previous methodology presented in Chapter 4. While the first methodology can determine a correlation among all VM types and also the best batch size configuration, the one presented in this chapter assumes that the user already chose the batch size. This brings the advantage of making the prediction in less time.

This approach accurately determined the best VM type for two applications evalu-

Table 5.13: Comparison of strategies to determine the instance that has smallest cost for a batch size of 2048 images for the MNIST dataset.

With validation	**VM type**	Relative cost using **5 epochs**	**10 epochs**	**20 epochs**	**Real runtime**
Yes	p2.xlarge	1.33	1.44	1.46	1.41
	p2.8xlarge	5.84	4.93	4.73	4.36
	p2.16xlarge	18.9	14.1	13.0	12.7
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	5.50	4.16	3.85	3.63
	p3.16xlarge	18.6	12.54	11.2	10.6
No	p2.xlarge	1.34	1.45	1.47	1.41
	p2.8xlarge	5.86	4.90	4.69	4.36
	p2.16xlarge	19.2	14.1	12.9	12.7
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	5.59	4.17	3.85	3.63
	p3.16xlarge	12.2	12.7	11.3	10.6

Table 5.14: Cost of performing the MNIST data experiments with and without the methodology proposed in this chapter.

VM type	**Cost with no methodology (U$)**	**Cost with methodology 1 (U$)**	**Cost with methodology 2 (U$)**
p2.xlarge	0.51	0.51	0.0009
p2.8xlarge	2.11	0.62	0.33
p2.16xlarge	7.65	2.30	0.88
p3.2xlarge	0.55	0.20	0.002
p3.8xlarge	2.00	0.80	0.38
p3.16xlarge	6.39	2.84	1.02
Total	**19.21**	**7.27**	**2.62**

ated: the MNIST/Simple-CNN and the seismic/FCN application. The approach can be described as a set of steps that the user must do to obtain the correct prediction. The user must consider, for example, the possibilities of the number of epochs to choose. Knowing this, the user will have no difficulty discovering the best instance for its problem.

We understand that our approach is a new way of determining the best cloud virtual machine type to train a Deep Learning model. At the same time, using it takes little time because it executes the application only for a few iterations.

Given time constraints, we could not execute experiments with more Deep Learning problems. Therefore, as future work, we understand that this approach should be evaluated with more problems, including problems with larger datasets and different Deep Learning models. We intend to develop an additional strategy that can help the user to correlate the runtime and cost variables easily. To do so, we intend to create a normalized equation where they can attribute weight to the variable more crucial to their project. It may also be interesting to embed it on other frameworks; especially on frameworks that are becoming popular for parallel execution, such as MXNet [6] and Horovod [30].

Chapter 6

Conclusions

In this work, we studied the behavior of applications that train Deep Learning models and proposed methodologies to estimate the performance and cost relationship of using different cloud VM types for this purpose. This way, users can better choose cloud resources to train their Deep Learning models.

We first show that the application settings, including the training batch size, may affect the performance and cost of different VM types; hence, they may affect the best resource choice. We also show that choosing a poor batch size can lead to improper use of virtual machine capabilities. It may not only slow down the application execution but also generate a waste of money since they may choose an expensive VM type and not use its power properly. Moreover, we showed that selecting a poor batch size may harm the final Deep Learning model accuracy.

This work also contributes with two methodologies to discover the best VM type to train a given Deep Learning model. The first methodology answered which GPU-based VM type provides the best cost-benefit correlation and also which batch size is more indicated to achieve the shortest runtime or the lowest cost. The second one presented the cost-benefit correlation among all instances in less time than the first one but presumed that the user already chose the application's batch size. Both methodologies were able to identify the cost-benefit correlation among VM instances for all benchmarks. And not only that, both ones were cheaper to analyze than run the application until entirely on the cloud.

We understand that the methodologies presented in this work are new ways to predict the best instance for a Deep Learning application. To the best of our knowledge, these methodologies are the only ones in the literature that need only the user's application to make the prediction.

For future work, it would be interesting to validate both methodologies for problems that deal with large datasets, such as ImageNet [7] and other Deep Learning architectures. To use large datasets is essential to check, for example, the influence of the initialization and first iteration's time on the final runtime and verify if more complex problems can disregard such variables. It is our wish to automatize these methodologies in a way that, given the dataset and the network architecture, the users will receive the information about which VM instance they must choose.

To identify the difference in performance and cost using the Spot instances, and check

if the proposed methodologies can analyze Spot instances. We also intend to check if these methodologies work for other Deep Learning frameworks, especially frameworks implemented to execute a Deep Learning application on distributed environments.

Bibliography

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 469–482, 2017. 22, 23

[2] Lucas Araújo, Fabíola Oliveira, Jorge Faccipieri, Tiago Coimbra, Sandra Avila, Martin Tygel, and Edson Borin. Detecção de estruturas em dados sísmicos com deep learning. *Boletim SBGf*, (104):18–21, 2018. 33

[3] Microsoft Azure. The team data science process lifecycle. `https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/lifecycle`. Accessed: 2018-04-15. 17

[4] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys*, 52(4), 2019. 19, 25

[5] Tiago Carneiro, Raul Victor Medeiros Da Nóbrega, Thiago Nepomuceno, Gui-Bin Bian, Victor Hugo C De Albuquerque, and Pedro Pedrosa Reboucas Filho. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6:61677–61685, 2018. 22, 23

[6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. In *Neural Information Processing Systems, Workshop on Machine Learning Systems*, 2016. 59

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 60

[8] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 42

[9] Parijat Dube, Tonghoon Suk, and Chen Wang. Ai gauge: Runtime estimation for deep learning in the cloud. In *IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 160–167, 2019. 22, 23

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 8, 13, 16, 17, 47

[11] Philip Jia Guo. *Software tools to facilitate research programming*. PhD thesis, Stanford University Stanford, CA, 2012. 17

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 26, 33

[13] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. 17

[14] Alex Kaplunovich and Yelena Yesha. Cloud big data decision support system for machine learning on AWS: Analytics of analytics. In *IEEE International Conference on Big Data*, pages 3508–3516, 2017. 22, 23

[15] Nitish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. 36

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset, 2014. http://www.cs.toronto.edu/kriz/cifar.html. 8, 33, 34

[17] Uday Kurkure, Hari Sivaraman, and Lan Vu. Machine learning using virtualized gpus in cloud environments. In *International Conference on High Performance Computing*, pages 591–604, 2017. 22, 23

[18] Randy Lao. A beginner's guide to the data science pipeline. https://towardsdatascience.com/a-beginners-guide-to-the-data-science-pipeline-a4904b2d8ad3. Accessed: 2018-04-10. 17

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998. 33

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 47

[21] Eva Maia Malta, Sandra Avila, and Edson Borin. Exploring the cost-benefit of aws ec2 gpu instances for deep learning applications. In *IEEE/ACM International Conference on Utility and Cloud Computing*, pages 21–29, 2019. 33

[22] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011. 14, 20

[23] Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997. 16

[24] Giang Nguyen, Stefan Dlugolinsky, Martin Bobák, Viet Tran, Álvaro López García, Ignacio Heredia, Peter Malík, and Ladislav Hluchỳ. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review*, 52(1):77–124, 2019. 46

[25] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Big data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences*, 30(4):431–448, 2018. 13

[26] Daniel Pop. Machine learning and cloud computing: Survey of distributed and SaaS solutions. *arXiv preprint arXiv:1603.08767*, 2016. 14

[27] David Reinsel, John Gantz, and John Rydning. The digitization of the world: from edge to core. *IDC White Paper*, 2018. 13

[28] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 33, 47

[29] F. Samreen, Y. Elkhatib, M. Rowe, and G. S. Blair. Daleel: Simplifying cloud instance selection using machine learning. In *IEEE/IFIP Network Operations and Management Symposium*, pages 557–563, 2016. 22, 23

[30] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018. 59

[31] Amazon Web Services. Amazon ec2 spot instances. `https://aws.amazon.com/ec2/spot/`. Accessed: 2020-05-25. 21

[32] Amazon Web Services. Definição de preço sob demanda do amazon ec2. `https://aws.amazon.com/pt/ec2/pricing/on-demand/`. Accessed: 2018-03-30. 24

[33] Amazon Web Services. Imagens de máquina da amazon (amis). `https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/AMIs.html`. Accessed: 2018-03-30. 24

[34] Amazon Web Services. Machine learning on aws. `https://aws.amazon.com/machine-learning/?nc2=h_l3_ai`. Accessed: 2018-03-30. 21

[35] Myungjun Son and Kyungyong Lee. Distributed matrix multiplication performance estimator for machine learning jobs in cloud computing. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 638–645, 2018. 22, 23

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. 33, 47

[37] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 363–378, 2016. 22, 23

[38] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Zhao. Complexity vs. performance: empirical analysis of machine learning as a service. In *Internet Measurement Conference*, pages 384–397, 2017. 23

[39] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833, 2014. 8, 18

Appendix A

All Results for the Seismic Data/FCN Experiments

Table A.1: Comparison of strategies to determine the best instance for a batch size of 512 images for the seismic/FCN experiments.

Analysis	**Instance type**	**5 epochs time**	**10 epochs time**	**20 epochs time**	**Real time**
Smallest Runtime	p2.xlarge	3.83	4.37	5.24	4.69
	p2.8xlarge	2.58	2.56	2.82	2.53
	p2.16xlarge	4.87	4.67	5.03	5.00
	p3.2xlarge	1.00	1.09	1.27	1.24
	p3.8xlarge	1.15	1.00	1.00	1.00
	p3.16xlarge	2.23	1.89	1.85	1.78
Smallest Cost	p2.xlarge	1.13	1.18	1.21	1.12
	p2.8xlarge	6.07	5.53	5.22	4.81
	p2.16xlarge	22.9	20.2	18.6	19.0
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	4.61	3.68	3.15	3.24
	p3.16xlarge	17.8	13.9	11.6	11.55

Table A.2: Comparison of strategies to determine the best instance for a batch size of 1024 images for the seismic/FCN experiments.

Analysis	**Instance type**	**5 epochs time**	**10 epochs time**	**20 epochs time**	**Real time**
Smallest Runtime	p2.xlarge	3.32	4.38	5.29	5.06
	p2.8xlarge	1.99	2.23	2.45	2.43
	p2.16xlarge	3.40	3.58	3.74	3.94
	p3.2xlarge	1.01	1.29	1.53	1.46
	p3.8xlarge	1.00	1.00	1.00	1.00
	p3.16xlarge	1.75	1.64	1.54	1.65
Smallest Cost	p2.xlarge	1.00	1.00	1.02	1.02
	p2.8xlarge	4.78	4.08	3.75	3.90
	p2.16xlarge	16.36	13.1	11.5	12.7
	p3.2xlarge	1.04	1.00	1.00	1.00
	p3.8xlarge	4.09	3.11	2.61	2.73
	p3.16xlarge	14.3	10.2	8.05	9.02

Table A.3: Comparison of strategies to determine the best instance for a batch size of 2048 images for the seismic/FCN experiments.

Analysis	**Instance type**	**5 epochs time**	**10 epochs time**	**20 epochs time**	**Real time**
Smallest Runtime	p2.xlarge	4.76	6.06	7.13	6.93
	p2.8xlarge	1.78	1.92	2.03	2.07
	p2.16xlarge	3.01	3.04	3.06	2.98
	p3.2xlarge	1.22	1.50	1.73	1.82
	p3.8xlarge	1.00	1.00	1.00	1.00
	p3.16xlarge	1.62	1.49	1.38	1.44
Smallest Cost	p2.xlarge	1.15	1.19	1.21	1.12
	p2.8xlarge	3.45	3.01	2.76	2.68
	p2.16xlarge	11.7	9.52	8.30	7.69
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	3.29	2.67	2.31	2.19
	p3.16xlarge	10.7	7.93	6.35	6.33

Appendix B

All Results for the MNIST Dataset Experiments

Table B.1: Comparison of strategies to determine the best instance for a batch size of 512 images for the MNIST dataset.

With validation	Instance type	5 epochs time	10 epochs time	20 epochs time	Real time
Yes	p2.xlarge	3.83	4.20	4.43	3.94
	p2.8xlarge	3.18	2.69	2.39	2.45
	p2.16xlarge	5.95	4.69	3.93	4.26
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	1.85	1.42	1.17	1.21
	p3.16xlarge	3.43	2.47	1.90	2.04
No	p2.xlarge	3.87	4.27	4.50	3.94
	p2.8xlarge	3.22	2.70	2.39	2.45
	p2.16xlarge	6.03	4.70	3.90	4.26
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	1.90	1.45	1.17	1.21
	p3.16xlarge	3.53	2.51	1.90	2.04

Table B.2: Comparison of strategies to determine the cheapest instance for a batch size of 512 images for the MNIST dataset.

With validation	Instance type	5 epochs time	10 epochs time	20 epochs time	Real time
Yes	p2.xlarge	1.13	1.24	1.30	1.16
	p2.8xlarge	7.47	6.32	5.63	5.77
	p2.16xlarge	28.0	22.1	18.5	20.0
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	7.39	5.70	4.68	4.83
	p3.16xlarge	27.5	19.8	15.2	16.3
No	p2.xlarge	1.14	1.26	1.32	1.16
	p2.8xlarge	7.58	6.36	5.63	5.77
	p2.16xlarge	28.4	22.1	18.3	20.0
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	7.58	5.78	4.70	4.83
	p3.16xlarge	28.3	20.1	15.2	16.3

Table B.3: Comparison of strategies to determine the best instance for a batch size of 1024 images for the MNIST dataset.

With validation	Instance type	5 epochs time	10 epochs time	20 epochs time	Real time
Yes	p2.xlarge	3.83	4.28	4.56	4.41
	p2.8xlarge	3.03	2.45	2.09	2.05
	p2.16xlarge	5.80	4.33	3.40	3.35
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	1.88	1.40	1.09	1.03
	p3.16xlarge	3.51	2.41	1.72	1.65
No	p2.xlarge	3.85	4.31	4.60	4.41
	p2.8xlarge	3.08	2.47	2.08	2.05
	p2.16xlarge	5.91	4.35	3.37	3.35
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	1.94	1.42	1.09	1.03
	p3.16xlarge	3.64	2.47	1.74	1.65

Table B.4: Comparison of strategies to determine the cheapest instance for a batch size of 1024 images for the MNIST dataset.

With validation	**Instance type**	**5 epochs time**	**10 epochs time**	**20 epochs time**	**Real time**
Yes	p2.xlarge	1.13	1.26	1.34	1.30
	p2.8xlarge	7.13	5.77	4.91	4.83
	p2.16xlarge	27.3	20.3	16.0	15.8
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	7.53	5.58	4.36	4.12
	p3.16xlarge	28.1	19.3	13.8	13.2
No	p2.xlarge	1.13	1.27	1.35	1.30
	p2.8xlarge	7.24	5.80	4.90	4.83
	p2.16xlarge	27.8	20.5	13.9	15.8
	p3.2xlarge	1.00	1.00	1.00	1.00
	p3.8xlarge	7.74	5.67	4.38	4.12
	p3.16xlarge	29.1	19.7	13.9	13.2