UCLouvain

École polytechnique de Louvain

Secure architectures implementing trusted coalitions for blockchained distributed learning (TCLearn)

Author: **Aurélien Boland**
Supervisors: **Benoît Macq, Axel Legay**
Readers: **Sébastien Lugan, Ramin Sadre**
Academic year 2019–2020
Master [120] in Mathematical Engineering

Acknowledgements

I would like to thank my supervisors Prof. Benoît Macq and Prof. Axel Legay as well as Dr. Sébastien Lugan and Dr. Paul Desbordes for their guidance but also their support throughout my work in this master thesis. I am particularly grateful for their reviews during the writing of my master thesis that helped me targeting the attended work.

Moreover, I would like to express my gratitude to Dr. Sébastien Lugan that shares with me his implementation of the TCLearn architecture and accepted to be reader of my work. I also thank Prof. Ramin Sadre for accepting to be reader of my work.

Introduction

The emergence of machine learning and the data problem

Recent years have been marked by a growing interest in machine learning. Industries are investing more and more money in this technology [1]. Machine Learning is a class of Artificial Intelligence (AI) methods that takes decisions based on data training. For example, if we want a computer to classify images, the machine learning methodology would be to build the classifier based on previously classified images that constitute the training set. A machine learning method is designed such that if the training set is representative of the problem's data then we may expect that the performances of the method will be good on data outside of the training set. A lot of machine learning algorithms can be found in the literature [2]. The choice of the method depends on the problem but also on the quantity of data available during the training phase and on the required computation resources. In some applications, interpretability is also an important characteristic. For example, in a linear regression we can easily see what is the impact of each features on the output but in other case such as neural networks the impact of each feature is less clear. One of the reasons why machine learning interest is growing inside of our society is the growth of available data and computation power. Techniques that are data hungry and computationally intensive, such as deep neural networks, are more and more used.

Over the years, data has become an important resource, some economists call it the "new oil" [3]. But data related to a person or a company contains sometimes sensitive information that the data owner does not want to share. The European Union has recently established a regulation called GDPR (General Data Protection and Regulation) [4]. This regulation allows people to be more informed and have more control on the information that a company has on them. Thereby, acquiring data could be a hard task in some fields. The development of AI solutions in these fields may be strongly impacted by this lack of data. An example could be

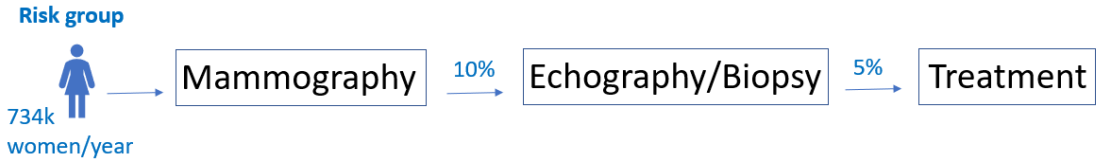


Figure 1: Workflow of the breast cancer screening in Belgium

the medical domain. Data driven algorithms could be helpful in many medical applications. It could be used to monitor the impact of a treatment on a huge amount of people and on different time scales [5]. There is also an important need of annotated medical images in order to build algorithms able to annotate images by itself [6]. But centralizing data from different hospitals, in order to train a model, is difficult because of privacy concerns even more if the hospitals are from different countries.

We may illustrate this with a more concrete example: breast cancer screening. Cancer is the second cause of death in Belgium, 25.8% of the death in 2017 were due to cancer. The sooner a cancerous tumor is detected, the easier is cure. Breast cancer is the most common cancer for women. In this logic, the Belgian government offers free breast cancer screening to the women between 50 and 69 years old. It is recommended to do a mammography every two years. The workflow of the breast cancer screening is given in Fig. 1. If every woman followed the recommendation, there would be 734 000 of woman to screen each year in Belgium. After the mammography, the corresponding image is analyzed by two experts that will decide if there could be a cancerous tumor. The use of two experts instead of one aims to reduce the risk of not perceiving a tumor or signaling a tumor when it is not the case (false alarm). There is a probability of 10% that the experts decide that there is a potential cancerous tumor and in this case further tests are advised. Then an echography is done if necessary and the experts will finish with a biopsy in order to analyze the potentially cancerous cells. These tests will lead with a probability of 5% to the diagnosis of a cancerous tumor. This is a simplified scheme of the breast cancer workflow that could change depending of the case and the hospital, for example some practitioners use Magnetic Resonance Imaging (MRI) but the Belgian Health Care Knowledge Centre does not recommend a routine use of this technology (in the case of breast cancer) [7]. This represents a lot of work with a low rate of positive case and a risk of false negative. As explained by MA Mazurowski et al. [8], an AI that is able to successfully categorize even half of the mammogram images as surely negative would significantly reduce the amount of work for the radiologists. Moreover, we may expect that a good algorithm will sometime find cancerous tumor that has not been found by the

experts. A solution of AI-based decision making support has been developed by Therapixel to enhance the breast cancer workflow [9]. As said before, one of the main obstacle for building good AI program that analyse mammogram images is the difficulty to collect data. My master thesis is focused on this problematic.

Several companies and research groups are working on solutions for this data acquisition problem. Most of these solutions are built using a framework called federated learning. The idea of federated learning is that each data owner (e.g. an hospital) keeps his data and the AI model is trained in a distributed way. In term of vocabulary, we will speak of a coalition in which the members have valuable data and want to train a machine learning model together. The design of a federated learning solution has to deal with four main challenges:

1. **Data and model privacy**
   In order to train the model in a distributed way, the coalition members will have to communicate with each other. It is important to avoid that these communications leak sensitive information about the data used during the training. Furthermore, in some applications, the trained model has to be hidden from the collation members. In some less restrictive configurations, we may agree that a member see the current model but we want to avoid that the model goes outside of the collation by using, for example, a method for tracing it.

2. **Good performances**
   When we train a model, the final performance obtained at the end is obviously an important criterion. We ideally want that the performances obtained with federated learning are not too far from what would be obtained if the model was trained classically with a centralized data set. An important criterion is also the amount of computation required to train the model.

3. **Resilience against wrong behaviours**
   It may happen that some members of the coalition do not act correctly. For example, a member trains the model on wrong data (bad annotation, irrelevant data, ...) or he does not trained the model with the requested method. The coalition has to be resilient to wrong behaviours in order to avoid degradation (deliberate or not) of the model.

4. **Trust in the training process**
   Do we trust a third party to update model based on the information shared by the members? In some applications, we want to avoid that the proper functioning of the process relies on a single entity. Instead, we decide that each decision during the training process relies on a member's consensus.

My contribution

I worked on an improvement of the TCLearn architecture which is a federated learning solution [10]. This solution addresses the four challenges explained before. I focused my work on the third one: the resilience against bad behaviours. The resilience mechanism that has been presented in the TCLearn paper does not take into account every possible bad behaviours. It allows a member to decrease the evaluated performances of the model but not too much according to a threshold. Moreover, the presented mechanism is some kind of heuristic that has no mathematical basis to decide of the rejection or not of a model increment proposal. I developed an other resilience mechanism based on statics that will be the main subject of this thesis.

Contents

Chapter 1

Background

1.1 Machine Learning

There is two main classes of problem treated by machine learning:

1. **Quantitative problem**
   Given an input vector x we want to predict a continuous value y. The commonly used notation is to write the true value y and the predicted value $\hat{y}$. An example would be to predict the price of a stock tomorrow (y) based on the information that we have today (x).

2. **Qualitative problem**
   We still have an input vector x, but now we want to predict a class. So we have:
   $$y \in \mathbf{C} = \{1, ..., C\}. \tag{1.1}$$
   Most of the time, instead of directly computing a predicted class, we will estimate the probability of each class which is written:
   $$p(c) = P(y = c) \ \forall c \in \mathbf{C} \tag{1.2}$$
   and then compute:
   $$\hat{y} = \underset{c \in \mathbf{C}}{\operatorname{argmax}} \left(p(c)\right). \tag{1.3}$$
   An example would be detecting the digit on a image (10 classes).

A machine learning algorithm can be represented by function that take x as input and that take w as a parameter. We usually call w the weights vector. In the quantitative case, we write:
$$\hat{y} = f(x|w) \tag{1.4}$$

and in the qualitative one, the notation becomes:

$$\hat{p} = (p(1), ..., p(C))^\top = f(x|w). \tag{1.5}$$

The couple (w, f) will be referred further as a machine learning model. Given a training set $D = \{(x_1, y_1), ..., (x_n, y_n)\}$, we will evaluate the quality of model using an evaluation metric $E(w, D, f)$. By convention we will consider that we have a good model if the evaluation metric is small.

In a quantitative problem a common evaluation metric is:

$$E(w, D, f) = \frac{\sum_{i=1}^{n} (y_i - f(x_i|w))^2}{n} \tag{1.6}$$

This metric is called the MSE for Mean Squared Error. In the case of a qualitative problem, the most intuitive metric would be to work with accuracy that is defined as follow:

$$E(w, D, f) = 1 - \frac{\sum_{i=1}^{n} I(y_i = \hat{y}_i)}{n} \tag{1.7}$$

where $\hat{y}_i$ is the most plausible class according to $f(x_i|w)$ and the function $I(.)$ is defined such that:

$$I(x = y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{1.8}$$

Despite the simplicity and the easy interpretation of this metric, it does not take into account the degree of certainty ($p(y)$). In other words, a 2 classes function that always attribute a probability of 60% to the right class will have the same metric than a classifier that attribute a probability of 100% to the right class. In order to take into account the degree of certainty a commonly used metric is cross-entropy, it is define as follow:

$$E(w, D, f) = -\frac{\sum_{i=1}^{n} log(p_i(y_i))}{n} \tag{1.9}$$

where $p_i(y_i)$ is the probability attributed to y_i by $f(x_i|w)$.

Training a machine learning model consist to computing w such that

$$w \approx \underset{w}{\operatorname{argmin}}\, E(w, D, f) \tag{1.10}$$

Due to the difficulty of solving some optimization problems, it is sometime admitted to choose a local optimum which is not the global one. It is also common to use an heuristic method. Moreover, we sometimes want to stay sufficiently far from the

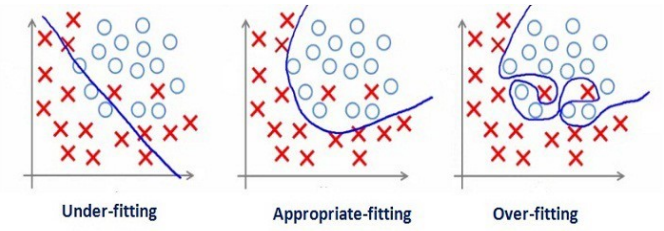


Figure 1.1: The over-fitting problem is illustrated with a 2 class problem (the blue circles and the red crosses)[1]

optimal solution to avoid a phenomena called over-fitting which is illustrated in Fig. 1.1. When we train a model, the final purpose is to have good performances not on the training set but on new data points. So if we fit too much on the training set, we may loose in term of performances when we generalize to other data points.

One of the simplest example of a machine learning method is the linear regression. In the case we have:

$$\hat{y} = f(x|w) = w^\top x \tag{1.11}$$

The associated evaluation metric is the MSE (see Eq. 1.6). In this particular case, the evaluation metric is quadratic in w and so the global optimum is fast to compute (in contrast to other methods that we will see further). The method is illustrated on Fig. 1.2 in the case where the input is a scalar.

1.2 Convolutional Neural Network (CNN)

In the field of image processing, the most current machine learning architectures are based on convolutional neural networks. This method is a class of deep neural networks that we will explain further. Convolutional neural network, commonly abbreviated by CNN, is a method that is often data and computational intensive but this method is a very powerful tool for processing images such as, for example, tumors. A good illustration of the strength of this technology in the field, could be the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). ILSVRC is a competition where participant purpose an algorithm to classify and detect object in images (the challenge changes each year). In 2012, a CNN called AlexNet highly exceeded all the competitors and the following years every top ranked solutions were CNN. This breakthrough is visible on Fig. 1.3.

[1] `https://medium.com/ml-research-lab/under-fitting-over-fitting-and-its-solution-dc6191e34250`

[2] `https://nextjournal.com/intelrefinery/simple-linear-regression`

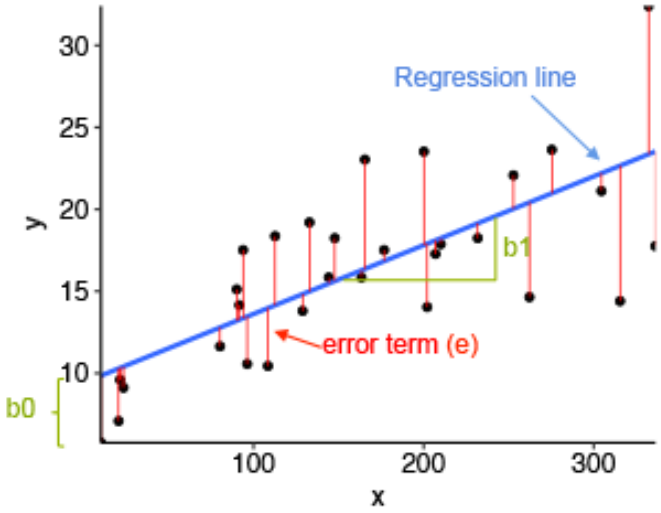


Figure 1.2: Illustration of a linear regression[2]

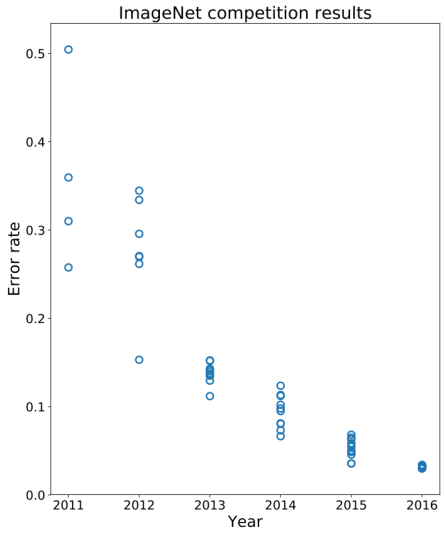


Figure 1.3: Error rate history on ImageNet [3]

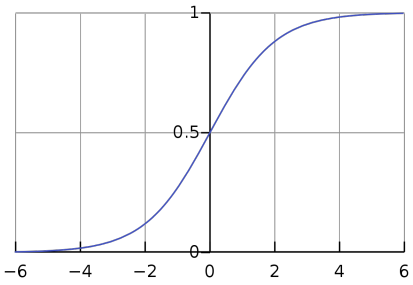


Figure 1.4: The sigmoid function [4]

1.2.1 Perceptron

A central notion to understand how a deep neural network works is the notion of perceptron. The model is the following:

$$\hat{y} = f(x|w) = \phi(x^T w) \tag{1.12}$$

where $\phi(.)$ is called the activation function. A common use of the perceptron is for two class classification where:

$$\phi(z) = \frac{1}{1 + e^{-z}}. \tag{1.13}$$

This function is called the sigmoid function and has two interesting proprieties: the function is monotone and is bounded between 0 and 1. So the output of a perceptron with the sigmoid function can be interpreted as a probability. The sigmoid function is plotted on Fig. 1.4. Other activation functions could be used depending on the problem. A common graphical representation of the perceptron is given in Fig. 1.5. This representation where the activation function is applied by a neuron will be useful to understand neural networks.

1.2.2 Deep neural network (mutlilayer perceptron)

In this section we will see the concept of multilayer perceptron which is the most popular type of deep neural network. It can be divided in three parties as illustrated on Fig. 1.6: an input layer, multiple hidden layers and an output layer. The number of hidden layers is called the depth of the networks. Each ball on the input layer

[3] https://commons.wikimedia.org/wiki/File:ImageNet_error_rate_history_(just_systems).svg

[4] https://commons.wikimedia.org/wiki/File:Logistic-curve.svg

[5] https://towardsdatascience.com/machine-learning-fundamentals-ii-neural-networks-f1e7b2cb3eef

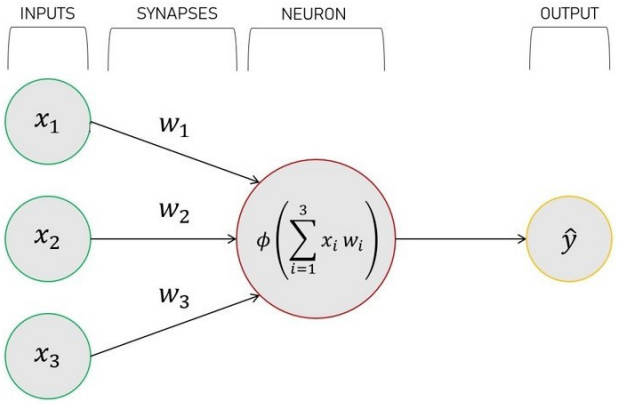


Figure 1.5: Graphical repesentation of the perceptron

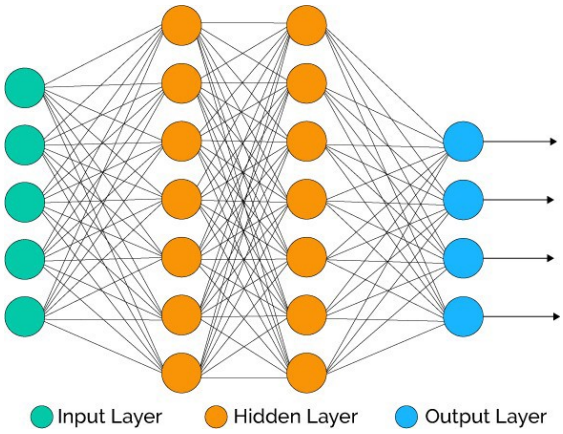


Figure 1.6: Graphical representation of a deep neural network (multilayer perceptron)[5]

correspond to a component of the input vector x. The balls of the hidden layers and the output layer are neurons (see Fig. 1.5). As with the perceptron, each input of a neuron is multiplied by a weight w_i and then an activation function is applied to the linear combination. The output of the neuron is broadcast to the next layer. One of the main strength of a neural network is given by the universal approximation theorem [11]. According to this theorem, a one hidden layer neural network can approximate any function (the precision of the approximation depend of the number of neurons in the hidden layer). Some counterparts are that training a neural networks can be data and computational intensive. The weights w of a network are regularly computed using the Stochastic Gradient Descent (SGD) algorithm. This optimization method requires to divide the training set D in batches (generally of a fixed size). The weights are updated by applying a gradient descent to each batch one by one. For a given batch, the gradient descent is done by applying N times the following formula:

$$w^* = w - \alpha \frac{\partial E(w, D_i, f)}{\partial w} \tag{1.14}$$

where w^* is the updated value of w, D_i is the i^{th} batch and α is the learning rate (this rate can be adjusted during the process). One of the main reason why batches are used is to decrease the memory consumption and the computation required. The parameter N is called the epoch number.

1.2.3 Use of convolutional layers

If we want to process images, we will try to capture information about the image structure: edges, curves, ... In order to do that convolution functions will be applied to the images before the neuron's layers. In this section we will focus on the 2D function that is used for processing black and white images. An black and white image is represented by a matrix where each element correspond to a pixel. Let X be the input matrix (image) of size $M \times N$, an let F be a $m \times n$ matrix that we call the the filter. The 2D convolution operation is written:

$$Z = X * F \text{ where } Z(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} X(k+i-1, l+j-1) F(k,l) \tag{1.15}$$

This operation is illustrated in Fig. 1.7. Another important operation in CNN is the pooling function that is used to reduce the size of the image outputted by the convolution. The image is divided in rectangles, and an operation is applied to each one. The most common pooling functions are max-pooling and average-pooling

[6] `https://towardsdatascience.com/a-beginners-guide-to-convolutional-neural-networks-cnns-14649dbddce8`

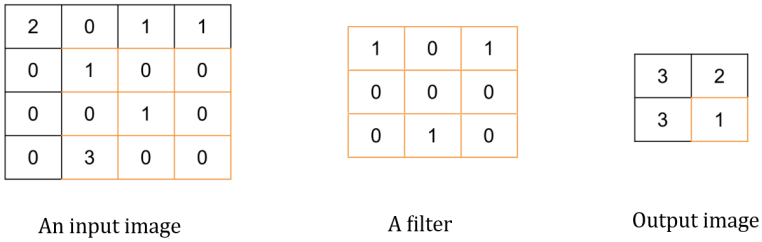


Figure 1.7: Illustration of the 2D convolution[6]

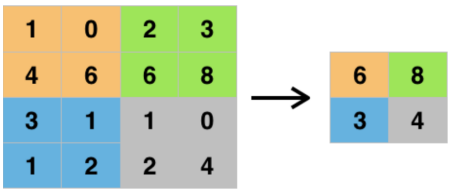


Figure 1.8: Illustration of the max-pooling function[7]

that will respectively compute the maximum and the average of each rectangle. Max-pooling is illustrated in Fig. 1.8. An example of convolutional neural network is given in Fig. 1.9. First, several convolutions are applied to the input image leading to several convoluted images. Each convolution has its associated filter, the elements of the filters are weights that will be computed during the training process. An activation function (See section 1.2.1) is generally applied to each element of the convoluted images. Then, each image is pooled. The last step will be to build a vector containing each elements of the pooled images (flattening operation). This vector is the input of a multilayer perceptron as seen in the section 1.2.2. Depending of the CNN design, there may be multiple pairs of convolution and pooling layers before the multilayer perceptron.

[7] https://deepai.org/machine-learning-glossary-and-terms/max-pooling

[8] https://medium.com/machine-learning-researcher/convlutional-neural-network-cnn-2fc4faa7bb63

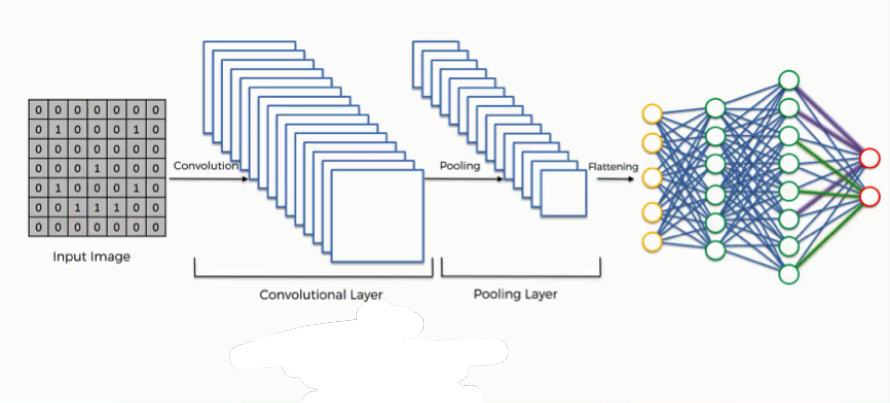


Figure 1.9: Example of convolutional neural network [8]

1.3 Hypothesis testing

Let the vector $X = (X_1, X_2, ...X_n)^T$ be independently distributed samples that follow a distribution that is partially known (for example a normal distribution with unknown mean) or totally unknown. We have one hypothesis H_0 about the distribution and an alternative hypothesis H_1. In our case we will be interested by the following hypothesis where μ is the mean of our distribution:

1. H_0: $\mu = 0$
2. H_1: $\mu > 0$

When we are doing hypothesis testing, we are computing $P(H_0|X)$ which is called the p-value. In order to compute this p-value different methods exist based on different assumptions.

The most popular method for testing hypothesis is the Student's t-test. Let us consider the following estimators:

$$\hat{\mu}(X) = \frac{\sum_{i=1}^{n} X_i}{n} \tag{1.16}$$

$$\hat{\sigma}(X) = \sqrt{\frac{\sum_{i=1}^{n} (\hat{\mu}(X) - X_i)^2}{n-1}} \tag{1.17}$$

If we make the assumption that the samples are normally distributed then we have this propriety:

$$\text{If } H_0 \text{ is true then } T = \sqrt{n} * \frac{\hat{\mu}}{\hat{\sigma}} \sim t_{n-1} \tag{1.18}$$

where t_{n-1} is the t-distribution with $n-1$ degrees of freedom. We compute the p-value with the following formula:

$$P(H_0|X) = P(T \geq t(X)|X) \tag{1.19}$$

T is the random variable and $t(X)$ is a value computed from the available samples with the formula bellow:

$$t(X) = \sqrt{n} * \frac{\hat{\mu}(X)}{\hat{\sigma}(X)} \tag{1.20}$$

1.4 Blockchain

Before explaining how the TCLearn architecture has been designed, it is important to explain the notion of blockchain. A blockchain is an immutable decentralized ledger. It is a database that is not managed by a central authority but by several peoples. As the name suggests, a blockchain consists of data blocks that are linked by using cryptography. The idea is that a group of people want to build a database together but they do not trust a single entity to manage this database. So, A block is added on the blockchain thanks to a consensus protocol and the fact that the majority of the users are honest. This technology has been popularised with the Bitcoin cryptocurrency where it is used to able people to exchange money without the need of a central bank. But blockchains have other applications than the financial one. It can be used for supply chain management, microgrids, elections, ... and federated learning.

To understand how a blockchain works, it is important to understand the notion of cryptographic hash function. An cryptographic hash function transforms a variable length input into a fixed size output. This function is a one-way function which means that given the output it is computationally hard[9] to find a corresponding input. Another important propriety of a cryptographic hash function is that it is computationally hard to find two inputs that gives the same hash. This propriety is called collision resistance. The structure of a blockchain is illustrated in Fig. 1.10. This illustration is a simplified version of a blockchain structure. There is generally other elements in a block such as a timestamp or elements used for the consensus protocol discussed later such as a nonce (element to compute during proof of work) or signatures (voting scheme). The purpose of adding a block to the blockchain is to add new data to the decentralized database, in the figure this data is called transactions. We observed that the data is stored by using a structure called Merkle tree. Every transactions are hashed, then the resulting hashes are concatenated two by two and hashed again and so on. The

[9]Ideally, the only way would be to use brute force

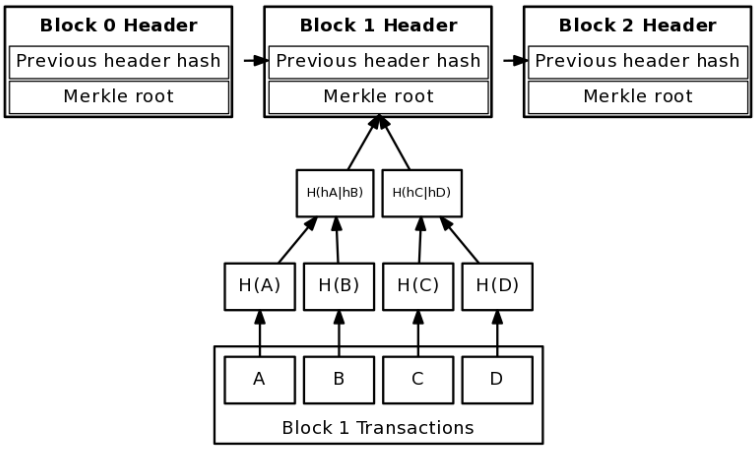


Figure 1.10: Simplfied blockchain structure [10]

block header only contains the root of this tree. The use of a Merkle tree allows to efficiently verify that a transaction is in a block. For example, if we want to verify that the transaction D is in the block, we do not have to reconstruct the Merkle tree by starting from all the transactions but we may simply download $H(C)$ and $H(H(A)|H(B))$ and then check if we obtain the same Merkle root. Because of the collision resistance, it is computationally hard to compute a fake value of $H(H(A)|H(B))$ or $H(C)$ that will aim to validate a false transaction D. In order to link blocks, an hash of the previous block header will be stored and so because of collision resistance it is hard to fake previous blocks.

1.4.1 Consensus protocol

Because there is no central authority, the users have to agree on the next blocks together. For example, in a financial blockchain they have to verify that the transactions are valid (enough money to do the transaction, the transaction is signed, ...). We want that all honest users have the same blockchain. The choice of the consensus protocol will depend if the blockchain is public or not.

In a public blockchain such as Bitcoin, everyone can go inside or outside of the coalition. A simple voting scheme for the new block agreement is not possible because a user can easily control multiple nodes of the blockchain network. This is called the Sybil attack. To avoid this attack, consensus where users have to allocate some resources during the protocol are used. The two commonly used methods

[10] `https://dtrt.org/posts/blockchain-pruning-problems-and-solutions/`

are proof of work and proof of stake where the users have to respectively allocate computational resource and money during the consensus protocol.

In a private blockchain, each node in the network corresponds to an identified user. A user need a permission to join the coalition. In this configuration Sybil attacks are not possible and it is easier to reach a consensus. Agreement among agents where some are not honest has been studied in this paper of 1982 called the generals byzantine problem. [12] An efficient way to reach agreement in an asynchronous network has been created under the name of Practical Byzantine fault tolerance. [13] An asynchronous network is a network where there is no upper bound on the message delay. This a common hypothesis when a blockchain runs on a large networks such as internet, there could be some lost messages or users disconnected. Some blockchain such as *Algorand* [14], are performed in configuration called partially synchronous. The protocol start in a an asynchronous setting and the second part is done in a synchronous setting. This configuration will be useful for TCLearn as explained later.

1.5 TCLearn

Thanks to the use of the SGD method, a deep learning model can easily be trained without centralizing the data. In fact, the SGD algorithm is based on the division of data in batch on which the model is trained successively. So at each iteration of the coalition, one of the coalition members will train the current model with several batches and send the results to the other members. To avoid confusion, we will say that a member train the model on a lot of data during his turn in the coalition protocol and this training is done by dividing the lot in batches as required by the SGD method.

Concerning the data privacy, we would avoid the possibility to reconstruct the lot of data used for training based on the model update (the difference between the new and the old model weights). This threat can be studied in the framework of differential privacy [15]. A randomized function $f : D \to R$ satisfies (ϵ, δ)-differential privacy if for any two data set $d, d' \in D$ differencing by a single item (adjacent) and for any subset of outputs $S \subseteq R$:

$$Pr[f(d) \in S] \leq e^{\epsilon} Pr[f(d') \in S] + \delta \tag{1.21}$$

In our case f will be the difference between the model weights before and after the training of a member. A differential private SGD algorithm has been studied in the following paper [16]. By adding noise and training on sufficiently large lots, (ϵ, δ)-differential privacy can be guaranteed. We observe that there will be a trad-off

between the training performances and the privacy.

The model is trained inside a closed coalition where each member are identified and need a permission to go inside. A coalition member has two data set, a training set and a test set. As the names suggest, the training set will be used to train the model and the test set will be used to evaluate the quality of the model training done by another member. The successive versions of the model are stored on a server that is managed by the supervisor. But because the members do not trust the supervisor, a blockchain that contains an hash of each model is built inside of the coalition. So when a member download a model from the supervisor server, he can verify the authenticity of the model using the model hash present on the blockchain. The coalition begins with an random model or a pretrained model that will be stored by the supervisor and the hash of this model will be in the first block of the blockchain. At each iteration of the coalition, a member want to train the model and is called the partner. Moreover, during this iteration, a set of members are randomly selected and will be called the validators. This random selection can be done without relaying on a third party by using the same method that the one used in the Algorand blockchain [14]. The method is based on the principle of verifiable random functions [17]. The steps followed by the coalitions at each iteration are illustrated in Fig. 1.11:

1. The partner requests the last accepted model to the supervisor. We call this model M_i. The blockchain will be used to verify the integrity of the received model.

2. The partner trains the model M_i. The resulting model is written M_{i+1}.

3. $H(M_{i+1})$, an hash of M_{i+1}, is send to the validators and the supervisor. This communication is signed by the partner, this will be helpful for the step 5.

4. M_{i+1} is send to the supervisor.The supervisor verify that the model correspond to the received hash.

5. The validators and the partner want now to verify that they have received the same model hash. In order to do that, they run the signed message protocol of the paper on the byzantine general problem [12]. Fast implementations of this synchronous protocol have been studied as presented in [18] or in the Algorand paper [14].

6. The validators downloads M_{i+1} and M_i. They verify the authenticity of the models using the corresponding hash.

7. The last step is to evaluate the training quality based on M_{i+1}, M_i and the test sets of the validators. This is done by the validators using the *model evaluation protocol*. This protocol will be the subject of the next chapters of this master thesis. The results, such as the acceptance of the model, are written in a new block of the blockchain. One of the validators submit a block proposal to the others. Then a synchronized byzantine protocol will be used and the block will be accepted if 2/3 of the validators agree as done in Algorand. The choice of a synchronous protocol is justified by the fact that we want all validators to be available for the model evaluation protocol and so we consider that they are still available after.

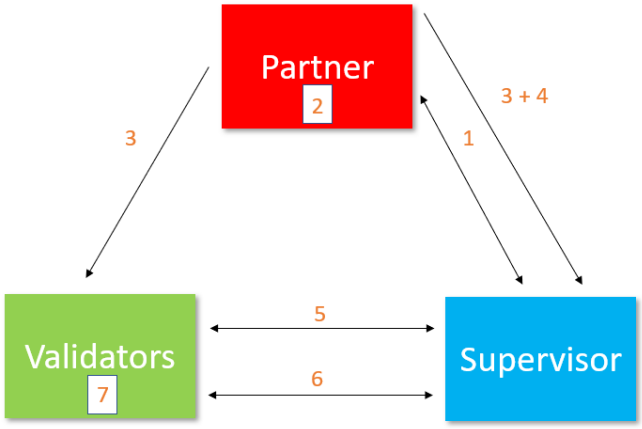


Figure 1.11: TCLearn protocol

Each block of the blockchain corresponds to a model proposal. A block contain at least the ID of the partner, the model hash, the results of the model evaluation protocol and the signatures of the validators that has agree on the block. There is no need to use a Merkle tree in our case because each block corresponds to only one model. The idea of basing the blockchain protocol on Algorand comes from another federated learning solution called Deepchain [19]. Despite this commonality, Deepchain is a solution with a different purpose, it relies on a public blockchain with financial incentives.

The purpose of this section was to give the main principles of the TCLearn architecture in which the model evaluation protocol (that is the main subject of this thesis) occurs. This is not an exhaustive explanation of what has been written in the TCLearn paper, I choose to focus on what will be useful for the rest of the master thesis. Moreover, this version of TCLearn is not exactly the one of the paper. For example, in the paper a global test set is distributed among the

members. This has been removed because we observed that it is hard to detect if a coalition member trains the model with some data from this global test set. Moreover, the use of a blockchain protocol based on Algorand is not mentioned in the paper.

Chapter 2

Model Evaluation Protocol

2.1 Objectives

The purpose of the model Evaluation Protocol (MEP) is to evaluate the quality of a partner training and take a decision on the acceptance of the model proposal. During this protocol, two models will be evaluated. The first one, the current model (M_i), is the last model that has been accepted. The second one, the model proposal (M_{i+1}), is the model proposed by the partner and that the coalition have to accept or reject. The evaluation of these two models is done by the validators (see section 1.5), each one use his own test set to evaluate these models. Based on these evaluations, a score is attributed to the model proposal and the acceptance of the model is decided using this score. We refer the coalition's model as the last model accepted by the coalition. There are four objectives that a good evaluation protocol has to reach:

1. **Avoiding degradation of the model**
   The acceptance of model proposal resulting from bad training could lead, in the long run, to a degradation of the coalition's model. One of the goal of the protocol is to detect, with a sufficiently low error rate, the model proposal that has lower performances than the current model.

2. **Avoiding fitting on the test sets**
   We want to avoid as much as possible that the successive accepted models fit on the test sets. This fitting could be intentional if a malicious partner knows a test set or an approximation of it. But it could also be due to a model acceptance criterion that is too restrictive.

3. **Resilience against malicious validators**
   There is no guarantee that every validator will be honest during the protocol. More precisely, the protocol assumes that there is at most m malicious

validators. The protocol has to be designed such the malicious validators are not able to change the result by sending false information. This is referred as resilience to Byzantine failures in the literature. [20]

4. **Good overall training of the model**
   If avoiding too much acceptance of bad model proposal (false positive) is an important criterion, avoiding too much rejection of good model proposal (false negative) is also important. Indeed, if the protocol is too restrictive and rejects a lot of good training, the performances of the coalition's model will be impacted. A model trained on less data points is generally less effective. A good way to measure the impact of a too restrictive coalition would be to compare the performances of a model trained with good data using TCLearn (and so the protocol) with a model trained with the same data in the classical centralized way.

2.2 The problem to solve

The purpose of the protocol is to decide if the model proposal is better than the current model. Before going more into details, it is important to define the meaning of "better" in terms of machine learning models. Let $D = (X, Y)$ be a random variable that represents a data point from the machine learning's problem that the model has to solve (for example, detecting breast cancer). The distribution of this variable is the distribution of the problem's data. Given an evaluation metric $E(.)$ and a model M, we may build a random variable $E(M) = E(M, D)$ which is the evaluation of D with the model M. The mean of the variable, written $\overline{E(M)}$, corresponds to the true performance of the model by opposition to the estimated performances obtained by using a test set. As reminder, we decided in section 1.1 that a small evaluation metric corresponds to a good model. So, we will consider that the model M_{i+1} is better than M_i if $\overline{E(M_{i+1})} < \overline{E(M_i)}$.

The problem of deciding if a model proposal (M_{i+1}) is better than the current model (M_i) will be treated as a hypothesis testing problem. The hypothesis are the following:

1. H_0: $\overline{E(M_{i+1})} = \overline{E(M_i)}$, the two models have the same performance.
2. H_1: $\overline{E(M_{i+1})} < \overline{E(M_i)}$, the model proposal is better than the current model.

The data available for the hypothesis test is the evaluations of the two models on each data point of the test sets. We consider N_v validators. We write:

$$D^{(j)} = \{D_1^{(j)}, ..., D_n^{(j)}\} = \{(X_1^{(j)}, Y_1^{(j)}), ..., (X_n^{(j)}, Y_n^{(j)})\} \tag{2.1}$$

the test set of the validators number j. The corresponding model evaluations are written:

$$\alpha^{(j)} = \{\alpha_1^{(j)}, ..., \alpha_n^{(j)}\} = \{E\left(M_i, D_1^{(j)}\right), ..., E\left(M_i, D_n^{(j)}\right)\} \tag{2.2}$$

$$\beta^{(j)} = \{\beta_1^{(j)}, ..., \beta_n^{(j)}\} = \{E\left(M_{i+1}, D_1^{(j)}\right), ..., E\left(M_{i+1}, D_n^{(j)}\right)\} \tag{2.3}$$

This hypothesis testing problem is referred in the the literature as a two sample hypothesis testing problem because we have to compare samples from two different distributions. In our case, there is a link between the two sample sets, they come from the evaluations of two models on the same data. We say that we have paired samples. In this case, the problem can be transformed in a one sample problem. Indeed, we may build the following sample set from the two others:

$$\gamma^{(j)} = \{\gamma_1^{(j)}, ..., \gamma_n^{(j)}\} = \{\alpha_1^{(j)} - \beta_1^{(j)}, ..., \alpha_n^{(j)} - \beta_n^{(j)}\} \tag{2.4}$$

Instead of computing the probability that the mean of one distribution is not greater than the mean of another, we will compute the probability that the mean of $E(M_i) - E(M_{i+1})$ is not greater than 0. To simplify the further notations we will define:

$$\mu(\gamma) \triangleq \overline{E(M_i)} - \overline{E(M_{i+1})} \tag{2.5}$$

It is important not to confuse $\mu(\gamma)$ with $\hat{\mu}(\gamma)$. The last one is an estimator of the mean given a sample vector γ as defined in Eq. 1.16. The hypothesis finally become:

1. H_0: $\mu(\gamma) = 0$
2. H_1: $\mu(\gamma) > 0$

2.3 The challenges

Training a machine learning model is an empirical process, there is no guarantee that training a model on a batch of images will lead to a better performance evaluated on a test set. However, we expect that if we train the model on enough data and if the data is good and representative of the problem, then the trained model will perform better. At the beginning of the model training process, when the model has only been trained on a small amount of data, there is a lot of chance that training the model on new data will improve his performance. But, after a certain amount of data used for training, it becomes hard to improve the model performance and we observe that some training will decrease the performance of the model.

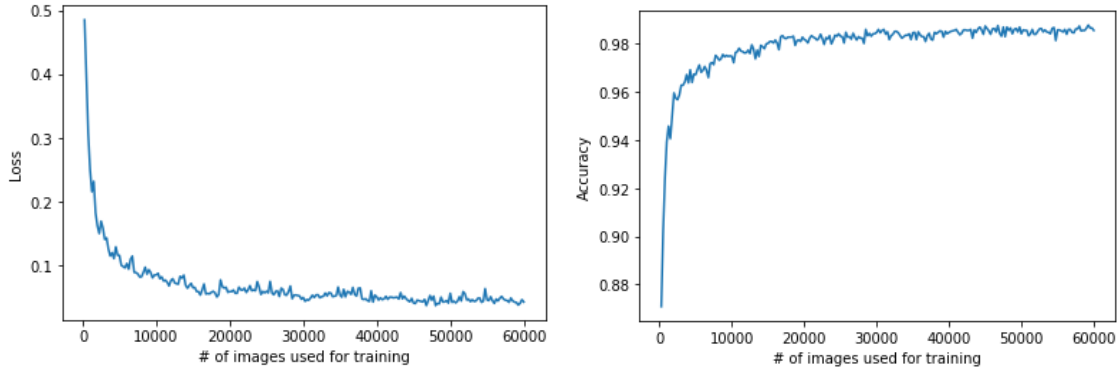


Figure 2.1: Evolution of a CNN performance metrics on 60 000 MNIST images

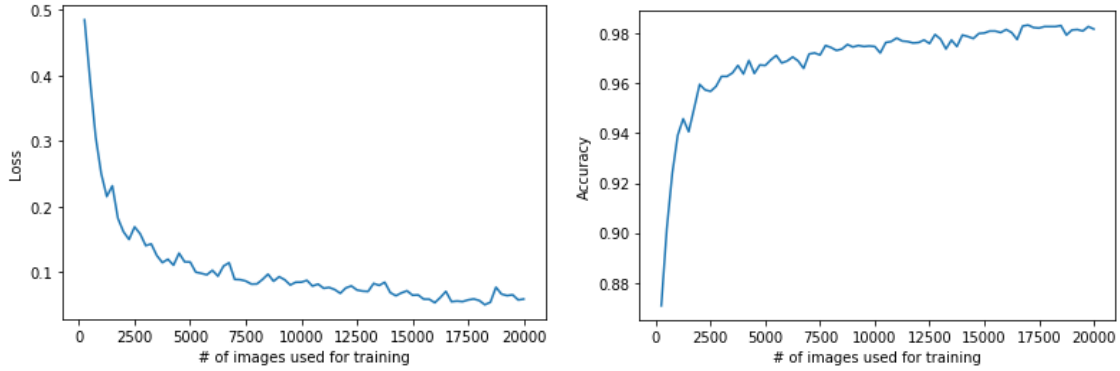


Figure 2.2: Evolution of a CNN performance metrics on 20 000 MNIST images

To illustrate this phenomena, a CNN has been trained on the MNIST database. This database contains 70 000 images of handwritten digits and the corresponding labels (the corresponding digits), 60 000 are used for training the model and 10 000 for the performance evaluation. The details about the neural network and the algorithm used to train the model are given at the appendix A. Each time the model has been trained on 250 images, the loss and the accuracy are evaluated on a test set. The loss corresponds to the evaluation metric that is optimized during the training process. This is illustrated in Fig. 2.1. As predicted, we observe an important decrease of the loss during the beginning of the training process. Then the performance of the model evolve slowly. However, this second part is still an important on, it is during this part that the model will be refine to go from a 6% to a 2% error rate. After 20 000 images, the evolution of the loss becomes more "noisy" and it decreases even slowly. We consider that the model has converged at this stage.

Another observation that will be interesting for the MEP is illustrated in Fig.

2.3 and table 2.1. The probability to observe a decrease of the loss after training the model on a fixed amount of data increases with this amount. This implies that it will be easier to observe an improvement of the model by a partner if he trained the model on a large amount of data. However, it is not possible to obtain a good approximation of the amount of data used for training the model based on the growing rate as shown in Fig. 2.4. Indeed, despite the fact that the training size has an impact on the growing rate, we observe that the distributions of these rates overlap a lot when the model is trained on 250, 1000 and 2500 images.

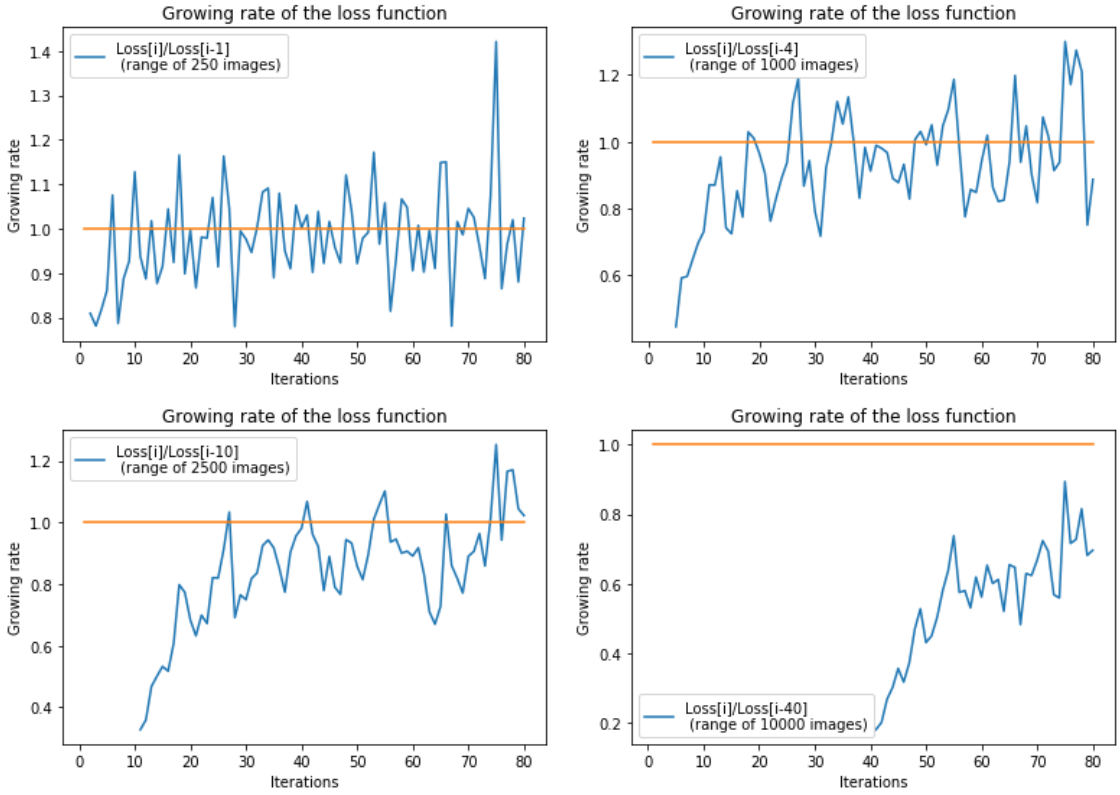


Figure 2.3: Each iteration corresponds to a training of the CNN on 250 images. The growing rate after k iteration is computed at the iteration i by $Loss[i]/Loss[i-k]$

Until now, we only looked at the average loss evaluated on a whole test set. But as explained in the section 2.2, we are interested by the distribution of the model evaluation on each image of the sets. At figure 2.5, I evaluated the cross entropies of 10 000 images on the CNN trained with 1250 images. We observe that the distribution is far from normal. A great majority of the points are close from 0 despite that the mean is 0.236. It is partly due to the fact that the neural network assigns the right label to 95% of the images. We observe the same phenomena

range size (in images)	Percentage of growing rates lower than 1
250	58.2%
1000	69.7%
2500	82.9%
10000	100%

Table 2.1: Percentage of growing rates lower than 1 when the CNN is trained on 20 000 images

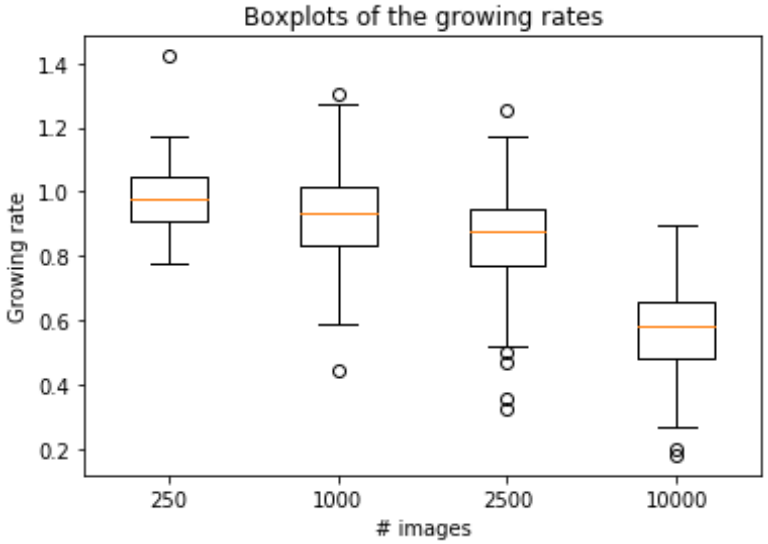


Figure 2.4: Boxplots of the growing rates for different training sizes

when we compute γ presented in section 2.2. At figure 2.6, we computed a model proposal by training the model above (already trained with 1250 images) with 250 images, and based on the two models γ is computed. Applying a normality test is not advised when we have a large amount of data. But this non normality can be deduced from the large amount of outliers on the boxplot of the figure. As reminder, the outliers of a boxplot are the point x_i such that:

$$x_i \notin [median - 1.5 * IQR, median + 1.5 * IQR] \tag{2.6}$$

where IQR (Interquartile range) is the difference between the third and second quartile. If the data is normally distributed the outliers represent 0.7% of the data. The convention of a boxplot is given in Fig. 2.7. This particular distributions of images cross entropy make the use of hypothesis testing methods challenging. Indeed, this method are usually designed for distributions not to far from the normal one.

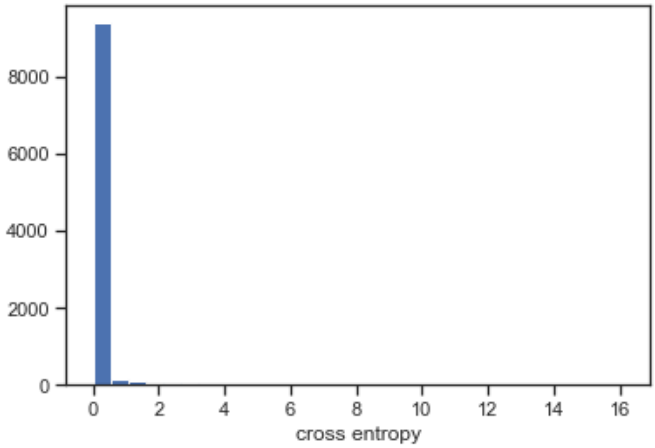
2.4 The different hypothesis testing methods

2.4.1 *t*-test

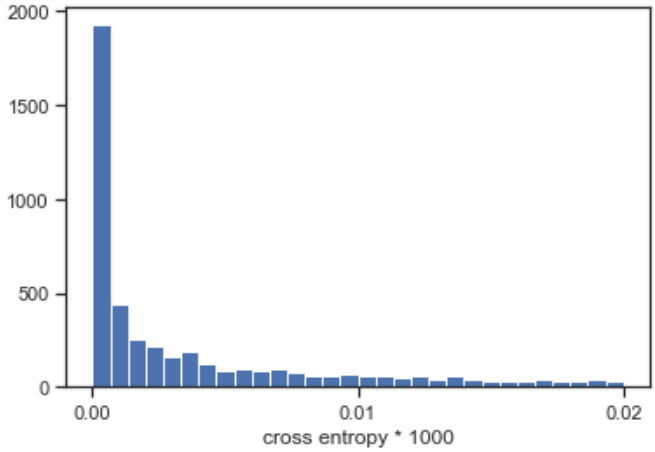
Despite the fact that γ is not normally distributed, we may expect that $T = \sqrt{|\gamma|} * \frac{\hat{\mu}(\gamma)}{\hat{\sigma}(\gamma)}$ (as define in Eq. 1.18) will follow a distribution close from $\mathcal{N}(0,1)$ if H_0 is true. Indeed, it has been shown in [21] that the t-test is robust to non normal distribution in the case of large sample size. This result is closely related to the central limit theorem but it is not a direct implication of this theorem where the estimated mean is divided by the standard deviation and not an estimate of it. [22]

A good way to see if we have a sample size large enough is to check if $\hat{\mu}(\gamma)$ is normally distributed. Based on the same two models than on section 2.3, M_i trained with 1250 images and M_{i+1} with 1500, we evaluated the two models performances on 60 000 images. This gives us a vector γ of 60 000 samples. Four histograms have been computed from γ in Fig. 2.8. For each histogram we decided of a sample size s. Then we divided the samples vector γ in vectors of size s and computed the mean value of each vector. These means are the elements of the histogram and gives us the empirical distribution of $\hat{\mu}(\gamma)$ when $|\gamma| = s$. The number of computed means decreases with s, which implies that for a value of s too large, there is not enough computed values to be able to guess their distribution. Besides the histograms, a normality test has been applied for the different values of s. The test chosen is the one provided by the *scipy* library of python which is based on the D'Agostino and

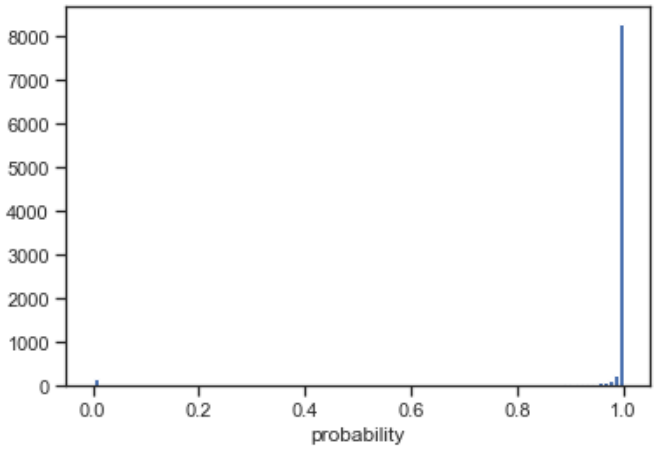
[1] `https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51`



(a) Distribution of the images cross entropy



(b) Distribution of the images cross entropy in the range [0;0.00002]



(c) Distribution of the correct label probability of each image

Figure 2.5: Scores distribution with MNIST test set

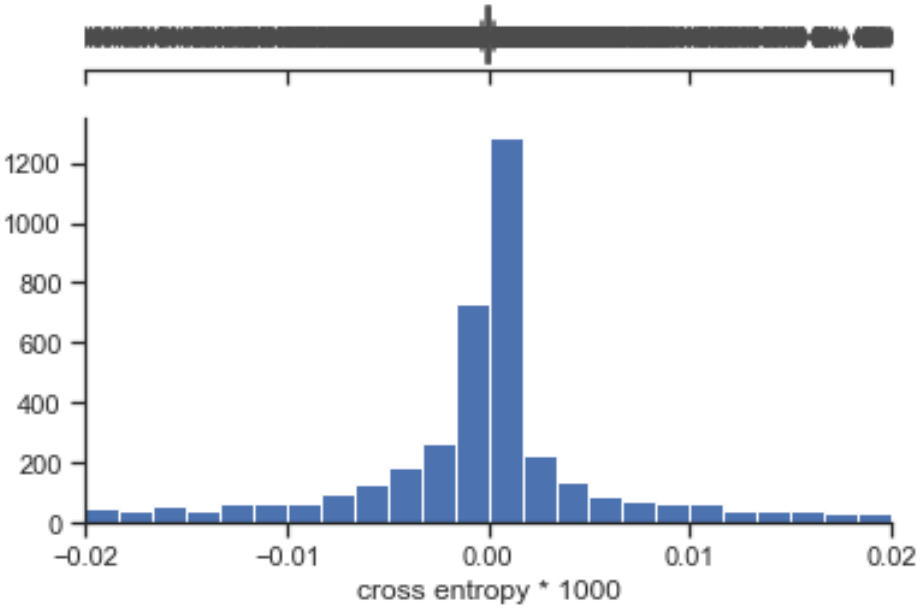


Figure 2.6: Distribution of γ (histrogram and boxplot)

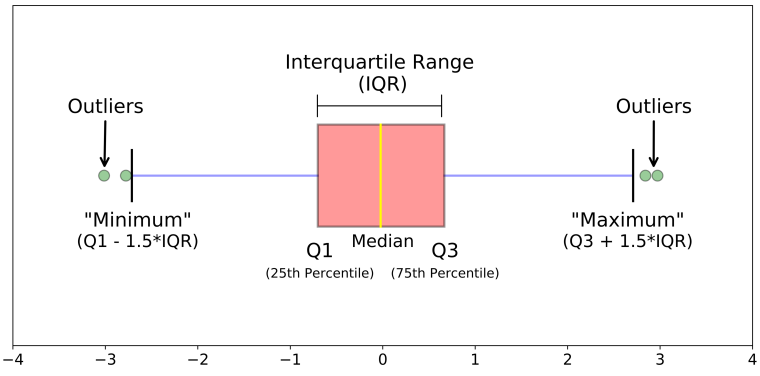
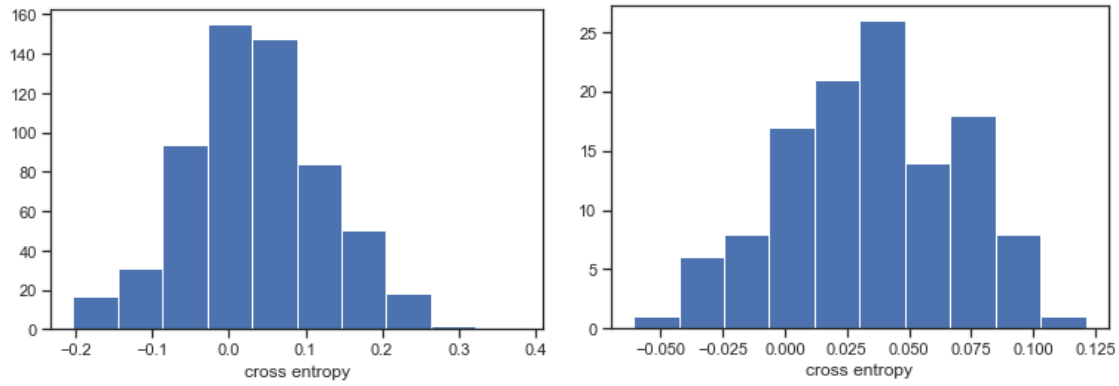
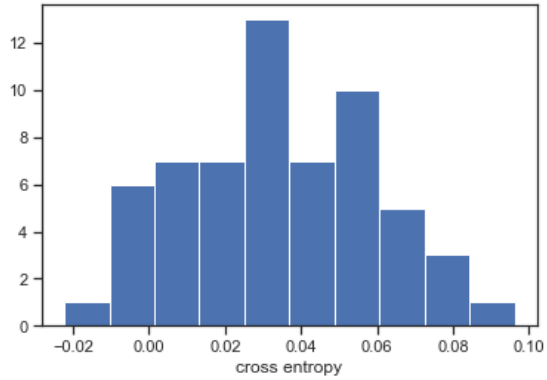


Figure 2.7: boxplot's convention[1]

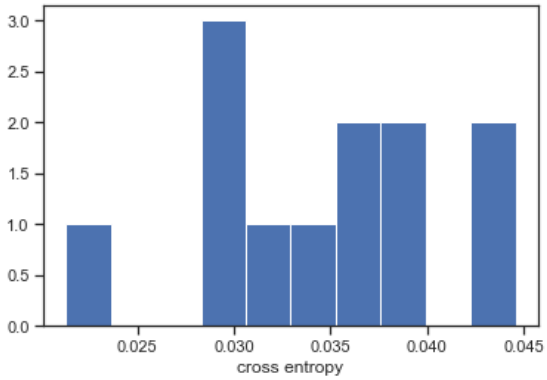


(a) $s = 100$, $p = 0.136$

(b) $s = 500$, $p = 0.575$



(c) $s = 1000$, $p = 0.823$



(d) $s = 5000$, the test is not consistent

Figure 2.8: Empirical distribution of $\hat{\mu}(\gamma)$ with $|\gamma| = s$. p corresponds to the p-value of the normality test

Pearson's test. [23]. The null hypothesis of this test is that the distribution is normal.

If we compare the graphs that we obtained with the one of Fig. 2.6, we observe that averaging enables to spread the distribution. The high concentration of values around zero vanishes already with $s = 100$. Despite the fact that the probability assigned to the normal hypothesis when we take the average of 100 values is not high (but not a small enough to reject the hypothesis), this probability increases with s and takes the value of 82% when $s = 1000$. In the case of $s = 5000$, we are not able to compute more than 12 means, this is not enough to compute the normality test (at least 50 means are advised according to the authors of the test). Unfortunately, we have a limited amount of images and so we are not able to conclude about normality with a high probability.

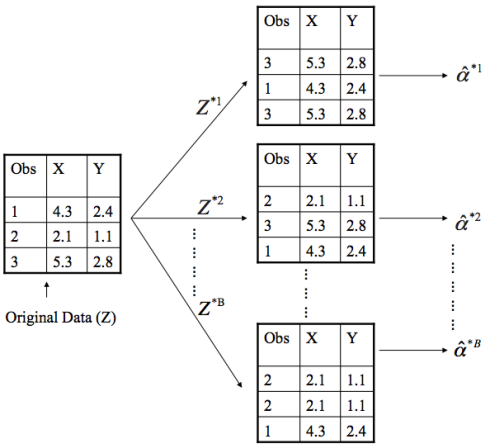


Figure 2.9: Illustration of the bootstrap procedure [24]

2.4.2 Bootstrap method

If the normality assumption can not be made, a solution would be to use simulation methods. One of these methods is bootstrap hypothesis testing.

We keep the estimator $\hat{\mu}(\gamma)$. Now, we are interested to know the accuracy of this estimator. First, we will generate m vectors of $|\gamma|$ samples $\gamma^1, .., \gamma^m$ that follow the distribution of our original vector γ. Because we consider that the distribution is unknown, we will have to generate the m vectors from the original one. We use a method called bootstrapping. Samples are randomly picked inside γ with replacement to form the m vectors. This is illustrated in Fig. 2.9. Then, we compute $\hat{\mu}(\gamma^1), .., \hat{\mu}(\gamma^m)$. The p-value is computed using the following principle:

$$P(H_0|\gamma) \approx P(\hat{\mu}(\gamma^B) \leq 0|\gamma) \approx \frac{\sum_{k=1}^{m} I(\hat{\mu}(\gamma^k) \leq 0)}{m} \tag{2.7}$$

where γ^B is a random variable that follows the distribution of a vector built from γ using bootstrap and $I(b) = 1$ if b is true and 0 otherwise. We applied the bootstrap method for testing the improvement of the two models seen before. We chose to use 1000 testing images and built 1000 bootstrapped vectors γ^i with $i = 1, ..., 1000$. The average of each vector is plotted on a histogram in Fig. 2.10. We observe that there is 30 values smaller or equal to 0. This implies a p-value of 3%.

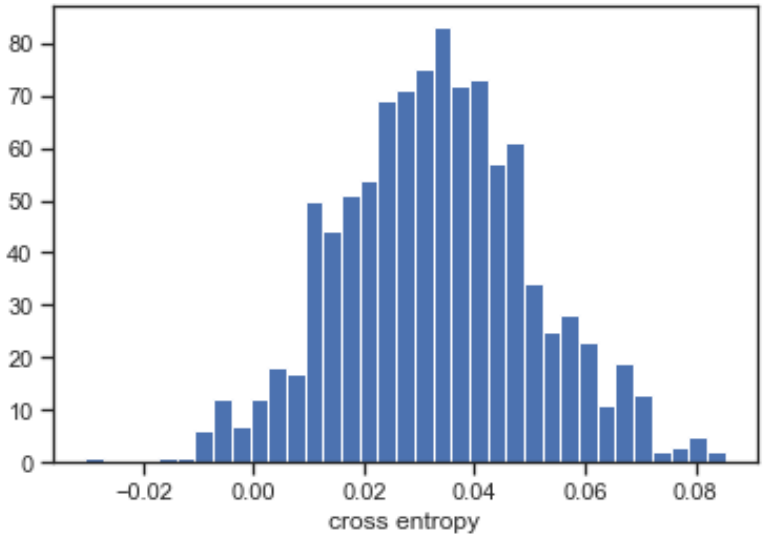


Figure 2.10: Illustration of the bootstrap method

2.4.3 Wilcoxon signed-rank test

This test is a commonly used test when the data is not normally distributed. No assumptions are made on the distribution of α and β. We still consider that the samples are paired but instead of testing if the distribution of α have a greater mean than the one of β, we want to know if the two sets of samples follow the same distribution. The tested hypothesis is slightly different:

1. H_0': $\mu(\gamma) = 0$ and the distribution of γ is symmetric
2. H_1: $\mu(\gamma) > 0$

The symmetry of the γ's distribution is an implication of the fact that α and β follow the same distribution.
The test consists of the following steps:

1. We remove all zeros from γ to build γ'.
2. The values of γ' are ranked from the smallest absolute value to the largest one. The rank of γ_i' is written R_i. The smallest absolute value rank is 1.
3. We compute:
$$w(\gamma') = \sum_{i=1}^{|\gamma'|} (sign(\gamma_i') * R_i) \tag{2.8}$$
where $sign(x)$ equal 1 if $x > 0$ and -1 if $x < 0$.

4. Under the null hypothesis H_0', $w(\gamma')$ follows a specific distribution (with no simple expression) with mean zero and a variance of $\frac{|\gamma'|(|\gamma'|+1)(2|\gamma'|+1)}{6}$. This distribution tends to the normal one when $|\gamma'|$ increases. The normal approximation is generally considered as reasonable for $|\gamma'| \geq 20$. Let W be a random variable that follows this distribution, we have that:

$$p = P(W \geq w(\gamma')) \tag{2.9}$$

where p is the p-value associated to H_0.

2.4.4 Permuted t-test

In a paper called Bootstrapping and permuting paired t-test type statistics [25], the authors are interested combining an analytical method (t-test) with a simulation method (bootstrapping or permuting). They classify their method as a semi-parametric method. The combination of the t-test with a permutation method will improve the robustness of the t-test to non normality. Or in other words, less data is required to obtain a correct p-value. This semi-parametric framework has been studied and tested in the paper with six different simulations methods, each one is either a bootstrapping or a permutation method. The method chosen for MEP is a permutation method that will be detailed further. This choice has been done in accordance with the paper's recommendations. Indeed, the authors recommend the method to face non normality.

The first step is to build m vectors $\gamma^1, .., \gamma^m$ from the original vector γ. The vectors are randomly built such that $\forall i = 1, ..., |\gamma|$ and $k = 1, ..., m$:

$$P(\gamma_i^k = \gamma_i) = 0.5 \text{ and } P(\gamma_i^k = -\gamma_i) = 0.5 \tag{2.10}$$

These vectors are called permutations of γ. We define:

$$T = \sqrt{|\gamma|} * \frac{\hat{\mu}(\gamma)}{\hat{\sigma}(\gamma)} \tag{2.11}$$

$$T_k^* = \sqrt{|\gamma^k|} * \frac{\hat{\mu}(\gamma^k)}{\hat{\sigma}(\gamma^k)} \tag{2.12}$$

which are the t-test scores of γ and his permutations. We compute the p-value p attributed to H_0 as:

$$p = \frac{\sum_{k=1}^{m} I(T_k^* \geq T)}{m} \tag{2.13}$$

It has been proved that the method is consistent, which means that if H_1 is true, the p-value will tend to 0 when $|\gamma|$ grows.[2] It has also been shown that the p-value

[2] Assuming that the elements of γ are independently distributed.

given by this method tends to be the same as the one given by the t-test when $|\gamma|$ grows. The proofs of this properties are given in the paper.

2.5 Implementation

As seen before the MEP is not centralized, the test images are shared among the validators and they do not want to share them. Moreover, we may not trust each member. Some of them could lie in order to improve the probability of acceptance or rejection of the model. In this section we will investigate how to implement the hypothesis testing methods presented above in a distributed way that allows to keep the test images confidential and that take into account malicious validators. As reminder, we consider at most m malicious validators.

2.5.1 Combining scores

In the t-test and Wilcoxon signed-rank test, we obtain a score respectively T and W that follows the standard normal distribution if the null hypothesis is true. Let Z_j be a random variable that represents the score obtained by the validator j after running either the t-test or Wilcoxon signed-rank test. Under the assumption that the tests are independent we obtain:

$$\text{If } H_0 \text{ is true then } Z = \frac{\sum_{j=1}^{N_v} Z_j}{\sqrt{N_v}} \sim \mathcal{N}(0,1) \tag{2.14}$$

where N_v is the number of validators. Thanks to this propriety, we may define a global hypothesis test where the p-value p is computed as follow:

$$p = P(Z \geq z(\gamma)) \tag{2.15}$$

with $Z \sim \mathcal{N}(0,1)$ and $z(\gamma)$ is define as:

$$z(\gamma) = \frac{\sum_{j=1}^{N_v} z\left(\gamma^{(j)}\right)}{\sqrt{N_v}} \tag{2.16}$$

where $z\left(\gamma^{(j)}\right)$ is the score obtained by applying the test on $\gamma^{(j)}$. To simplify further notations, we will write $z_j \triangleq z\left(\gamma^{(j)}\right)$.

At this stage, the method is not resilient to adversarial inputs. Indeed a validator could enforce a small p-value by sending a high value of z_j. If he wants a large p-value, he will send a high negative value of z_j. To avoid these attacks, a method would be to filter the inputs as it has been suggested in [26]. The inputs z_j for

$j = 1, ..., N_v$ are ordered, the m smallest and the m largest values are removed. z' is computed on the remaining inputs. If we define $\mathcal{Z}(\gamma, m)$ as the set of partner's scores without the m smallest and the m largest value, we are then able to define z' as :

$$z'(\gamma, m) = \frac{\sum_{z \in \mathcal{Z}(\gamma,m)} z}{\sqrt{N_v - 2m}} \tag{2.17}$$

We are now able to compute a global hypothesis test resilient to adversaries. The p-value of this test is written p' and is defined as before but with $z'(\gamma, m)$ instead of $z(\gamma)$:

$$p'(m) = P(Z \geq z'(\gamma, m)) \tag{2.18}$$

2.5.2 Fisher's method

An other way to obtain a global p-value is to use the Fisher's method. [27] Under the null hypothesis, the p-value of an hypothesis test is uniformly distributed. This is directly related to the definition of a probability and is a useful propriety that allows to build the Fisher's method.

Let $P_1, ..., P_{N_v}$ be random variables that represent the p-values of each validator's test. Under the assumption that the tests are independent we obtain:

$$\text{If } H_0 \text{ is true then } F = -2 * \log\left(\Pi_{j=1}^{N_v} P_j\right) \sim \mathcal{X}^2_{2N_v} \tag{2.19}$$

where $\log(.)$ is the natural logarithm and $\mathcal{X}^2_{2N_v}$ is the chi-squared distribution with $2N_v$ degrees of freedom. We define $f(\gamma)$ as:

$$f(\gamma) = -2 * \log\left(\Pi_{j=1}^{N_v} p\left(\gamma^{(j)}\right)\right) \tag{2.20}$$

where $p\left(\gamma^{(j)}\right)$ is the p-value obtained by applying the test on $\gamma^{(j)}$. To simplify further notations, we will write $p_j \triangleq p\left(\gamma^{(j)}\right)$.

We observe that high p-values implies a low value of f. Based on that, we may compute a global p-value p from this metric as follow:

$$p = P(F \geq f(\gamma)) \tag{2.21}$$

where $F \sim \mathcal{X}^2_{2N_v}$.

As it has been done with the previous method, we have to add a filtering step if we want the global hypothesis test to be resilient to adversary inputs. We define a set $\mathcal{P}(\gamma, m)$ that contains the partner's p-values without the m smallest and the m largest. The Byzantine resilient global p-value $p'(m)$ is then:

$$p'(m) = P(F \geq f'(\gamma, m)) \tag{2.22}$$

2.5.3 Global test

A solution to reduce the amount of computation is to execute one global test among the coalition. In this case, the samples are not the evaluation metric of one image but the evaluation metric of the whole test set of a validator. We have seen in section 2.4.1 that loss of a large set of images is normally distributed and so the t-test is a good method in this setting. Let $l_j \triangleq E(M_i, D^{(j)}) - E(M_{i+1}, D^{(j)})$ be the difference in loss between the current and the proposal model evaluated by the partner j. The metric t defined in Eq. 1.18 becomes:

$$t(l) = \sqrt{N_v} * \frac{\hat{\mu}(l)}{\hat{\sigma}(l)} \tag{2.23}$$

where $l = (l_1, ..., l_{N_v})$.

This metric can be transformed in Byzantine resulting by using filtering. Let $l'(l, m)$ be the set l without the m smallest and the m largest values. The Byzantine resilient metric t' is then:

$$t'(l, m) = \sqrt{N_v - 2m} * \frac{\hat{\mu}\left(l'(l, m)\right)}{\hat{\sigma}\left(l'(l, m)\right)} \tag{2.24}$$

The Byzantine resilient global p-value $p'(m)$ is:

$$p'(m) = P(T \geq t'(l, m)) \tag{2.25}$$

where $T \sim t_{N_v - 2m - 1}$ (the Student's distribution with $N_v - 2m - 1$ degrees of freedom). It is important to not make the normal approximation in this case unless there is a huge amount of validators.

2.5.4 How to fix the threshold ?

An important point that has not been discussed yet is the threshold under which the p-value has to be for the model acceptance. In the TCLearn paper, a model proposal with a worst evaluated performance may potentially be accepted. In this paper, the acceptance criterion is that the growing rate is bellow a threshold and this threshold is greater than 1. As we seen in Fig. 2.3, training a model (with good data) leads sometimes to a growing rate greater than 1. So, the main reason for tolerating models with a worst evaluated performance, is to avoid as much as possible the rejection of models trained on good data. We except that a model trained with wrong data will imply, with a high probability, a growing rate over the threshold. But this principle only works in a configuration where every member follows the rules which means that they trained the model on the requested quantity

of data and that they use the same optimisation method. In this case the only wrong behaviour that may happen is training on wrong data. This assumption is not compatible with the ones defined in this thesis (see Introduction). Indeed, we consider that there could be malicious members doing wrong behaviours that are not limited to training on wrong data. For example, in this configuration, a member with wrong data could train the model on less data in order to force the acceptance of the model. More generally, this allows a member to send a lot of model proposals consecutively that are each time slightly worst. This will implies a degradation of the coalition's model in the long run.

To avoid what has been described above, it is important to implement the following rule where M_1, M_2 are two models: If the MEP accepts M_2 when M_1 is the current model then the MEP will not accept M_1 when M_2 is the current model. This is equivalent to say that the MEP has an preference order between the models and that on condition of acceptance (necessary but not sufficient) is that the model proposal is preferred over the current one. And so thanks to this rule, there is will be no acceptance of models considered as worst by the MEP.

Let p_1 be the p-value attributed when M_1 is the current model and M_2 is the model proposal. And let p_2 be the p-value attributed when M_2 is the current model and M_1 is the model proposal. If we consider a fixed test set, we have that for each of the four hypothesis testing methods[3]: $p_1 = 1 - p_2$. The combination score method conserves this propriety but this is not the case with the Fisher's method (because this method takes a decision based on the product the validators p-values). Due to this propriety, it can be shown that the only way to define preference among a MEP that use combine score is to consider:

$$M_2 \text{ is preferred to } M_1 \Leftrightarrow p_1 < 0.5 \tag{2.26}$$

Because we want the preference to be a necessary condition for acceptance, we have that the highest allowed threshold is 0.5. Based on the same logic, we have that the highest allowed threshold for the ratio method is 1. We will see in the following chapter that the Fisher's method and combination score method give close results when they are associated with the t-test or Wilcoxon test and so in this case it seems reasonable to keep the 0.5 upper bound also for the Fisher method.

[3]In the case of the bootstrap and permuted t-test, we consider that we used the same simulated sample sets for p_1 and p_2

Chapter 3

Evaluation and comparison of the different methods

3.1 Methodology

In order to evaluate the performance of the different MEP methods and compare them, we write a python library that aim the simulation of TCLearn's coalitions. These simulations are done with the MNIST dataset and the same neural network and optimizer as on chapter 3 (see the appendix A). This library is available on the TCLearn GitHub (`https://github.com/slugan/TCLearn`). The part of the library used for training the CNN model on lots of data and taking intermediary measurements of the performances was already written. We extend the library to allows the simulation a coalition, this includes the implementation of the different methods of the MEP presented before.

A coalition is emulated by using the following instruction :

```
coalition_MEP.train(folder,trainingSize,localSize,nMembers,
validationSize,HTmethod,CombineMethod,nSim,m=0,dataSize=20000,
mnist="mnist/mnist_dataset.h5",threshold = 0.05,badFactor=None)
```

where :

`folder`	String that corresponds to the folder's path where the results will be stored
`trainingSize`	The number of image used by a partner to train the model before submitting a proposal (lot's size)
`localSize`	The size of each partner's test set
`nMembers`	Number of members in the coalition
`validationSize`	Size of the validation set. The validation set is a set of images that are not used by the members of the coalition. This set allows to evaluate the performance of the model after each step of the coalition.
`HTmethod`	String that corresponds to the hypothesis testing method to use. 6 values are possible : ttest, bootstrap, wilcoxon, perm_ttest, global_ttest, ratio. The ratio option corresponds to the test of the TCLearn paper where the decision is taken based on the ratio between the (average) loss evaluated on the model proposal and the current model (the growing rate)
`combineMethod`	String that corresponds to the method used for combining the validator's p-value. 2 values are possible : combineScore, fisher
`nSim`	Number of simulated test sets in the simulation hypothesis methods (bootstrap or permuted t-test)
`m`	Upper bound on the number of malicious validators
`dataSize`	Size of the training set which is divided among the members
`mnist`	String that corresponds to the MNIST database's path
`threshold`	A model proposal is accepted if the p-value given by the MEP is bellow the threshold.
`badFactor`	If the value is not `None`, one of the member data is modified. A portion badFactor of the images labels are shuffled. The purpose is to obtain a member with wrong data and see if his model proposals will be accepted.

The first step of the simulation is to divide the training set equally in `nMembers` parts. Each part represents a member and will be divided in lots of size `trainingSize`. An important aspect of this simulation is the order of the member's model proposal and when to stop the coalition. The role of partner is attributed to the members of the coalition in a cycling order (1,2,3,1,2,3,1,2,....). The partner trains the model with an images lot (a training set of size `trainingSize`) and submits it for approval. If the model is accepted, it is the turn of the next member to be the partner. Else he continues training the model with an other lot until the model is accepted or he has no more data. The coalition stops when nobody is able to submit a model

that will be accepted. A data lot that has been used for the training of a model that has not been accepted will be used again when the current model change. The role of validator is given to all the members except the partner. This procedure is described by the Algorithm 1, 2 and 3. In these algorithms, each data lot is marked as unused, used or accepted.

Algorithm 1 *train* with input model M and data lot D

$M' \leftarrow M$ trained with D
D is marked as used
return M'

Algorithm 2 *mep* with input model M and partner p. The threshold t and the hypothesis testing and combination methods used in the MEP are parameters of the algortihm.

Every members expected the partner p are designated as validators
$p-$value is computed by the validators according to the MEP
if $p-$value $< t$ **then**
    All used data lots of partner p are marked as accepted
    All used data lots of the coalition members are marked as unused
    $statue \leftarrow$ "$accepted$"
else
    $statue \leftarrow$ "$refused$"
return *statue*

Concerning the `combineScore` option to combine p-values, the method presented in section 2.5.1 has to be adapted for the bootstrap and permuted t-test methods. Indeed, these methods are not associated to a score that follows a normal distribution under the null hypothesis. Instead, we choose to average the p-values for these methods.

Because the simulations are computational intensive and so it takes time to obtain results, we were limited in the number of tests done. Moreover, due to the use of GPU parallel computing, the results are not reproducible (running two times the same script may lead to different results).[1]

[1] `https://keras.io/getting_started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development`

Algorithm 3 *coalition*. The current model is written M_i and the model proposal is written M_{i+1}

The model M_i is initialized with random weights
All the data is marked as unused
repeat
    According to the cyclic order, select a member p with unused data as partner
    $D \leftarrow$ an unused data lot of the partner p
    $M_{i+1} \leftarrow train(M_i, D)$
    $statue \leftarrow mep(M_{i+1}, p)$
    while $statue == "refused"$ **and** p has unused data **do**
        $D \leftarrow$ an unused data lot of the partner p
        $M_{i+1} \leftarrow train(M_{i+1}, D)$
        $statue \leftarrow mep(M_{i+1}, p)$
    if $statue == "accepted"$ **then** $M_i \leftarrow M_{i+1}$
until All the data is marked as used or accepted

3.2 Comparison of the methods on the same training

We will begin the comparison of the methods without simulating the model's acceptances or rejections. We trained a CNN on 20 000 MNIST images and we save the model after each training on a lot of 250 images. The loss and the p-values according to the different methods will be evaluated for each model. This is done with 9 validators (which corresponds to the number of validators in a coalition of 10 members) and each validator has a test set of 1000 images. The parameter `nSim` is equal to 200. No malicious validators are considered at this stage.

We observe on Fig. 3.1, 3.2, 3.3 and 3.4 the global p-value evolution for each hypothesis testing method (except the global t-test) and each p-values combination method. A first interesting thing to notice is that the choice of the combination methods does not change a lot the global p-value in the case of the t-test and the Wilcoxon test. This is not observed for the bootstrap and the permutation method for which, as reminder, the combine score method corresponds to averaging. With these two hypothesis testing methods, the p-value generally closer from 0.5 with the averaging than with the Fisher's method. In Tab. 3.1, we compute the standard deviation of the difference between the p-value given by the combine score and Fisher's method for a given hypothesis testing method. We observe a standard deviation around 5% for the t-test and the Wilcoxon test, and around 20% for the others. Another point that we may notice is that the p-value given by the Wilcoxon

Hypothesis testing method	Standard deviation of the p-value
t-test	6.5%
bootstrap test	22.5%
Wilcoxon test	5.3%
permuted t-test	22.6%

Table 3.1: Standard deviation of the difference between the p-value obtained by combining scores and the p-value obtained using the Fisher's method

	percentage of p-values lower than:				
MEP method	0.5	0.4	0.3	0.2	0.1
t-test with combine score	76%	75%	66%	61%	52%
t-test with Fisher	77%	69%	66%	60%	50%
global t-test	62%	56%	51%	50%	46%
bootstrap test with combine score	55%	45%	31%	22%	7%
bootstrap test with Fisher	57%	56%	56%	50%	42%
Wilcoxon test with combine score	55%	55%	55%	52%	50%
Wilcoxon test with Fisher	55%	55%	55%	55%	49%
permuted t-test with combine score	55%	45%	26%	21%	7%
permuted t-test with Fisher	57%	56%	52%	46%	42%

	percentage of growing rates lower than:				
	1	0.9	0.8	0.7	0.6
ratio	55%	29%	5%	2.5%	1.25%

Table 3.2: Percentage of p-values lower than a given threshold for the different hypothesis testing methods

test is either very close from 1 or from 0. On Fig. 3.5, we observe that global t-test, which is done on averaged samples, does not give a good approximation of the t-test computed with each image of the test sets.

In Tab. 3.2, we observe the percentage of training (done with 250 images) that would be accepted in the MEP for different hypothesis testing method and threshold. We observe that with the non global t-test methods more than 75% of the p-values are smaller than 0.5. Based on the same overall CNN's training there is 55% of the model proposal that will lead to a growing rate smaller than 1. This means that a larger part of the model proposals is more likely to be better than the current one based on the t-test than based on the growing rate. The percentage of accepted models stays close from the previous value when the threshold becomes 0.4. We see that the other hypothesis testing methods will accept less models.

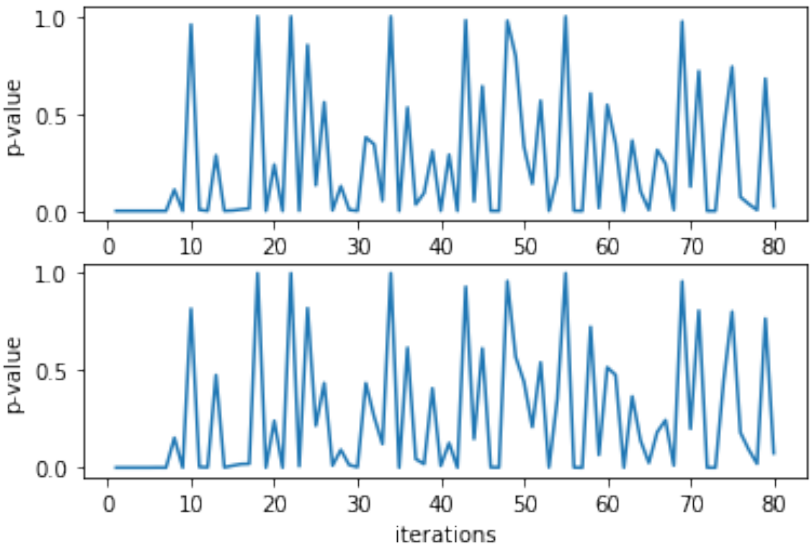


Figure 3.1: Global p-value of the t-test with combine score (above) vs Fisher (bellow)

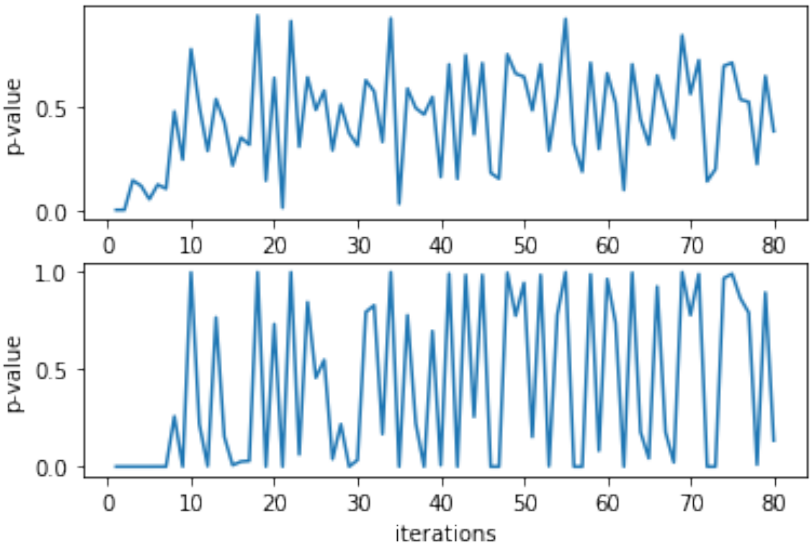


Figure 3.2: Global p-value of the bootstrap test with combine score (above) vs Fisher (bellow)

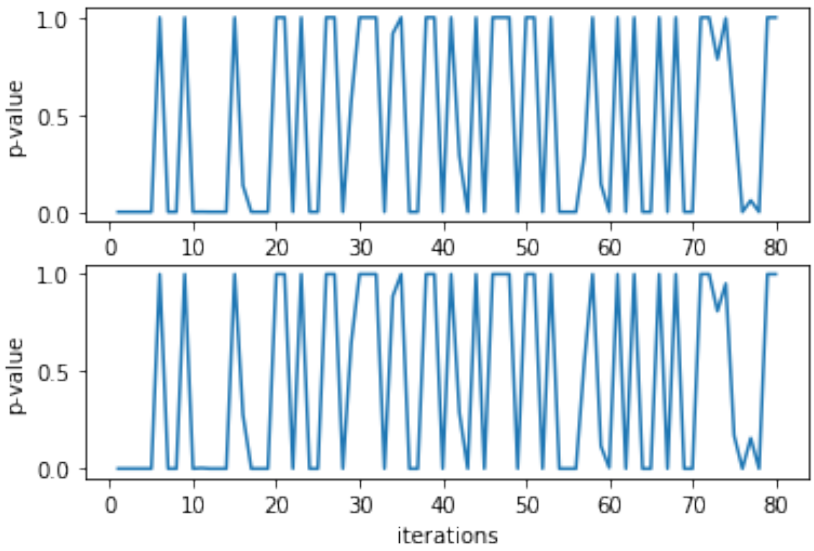


Figure 3.3: Global p-value of the Wilcoxon test with combine score (above) vs Fisher (bellow)

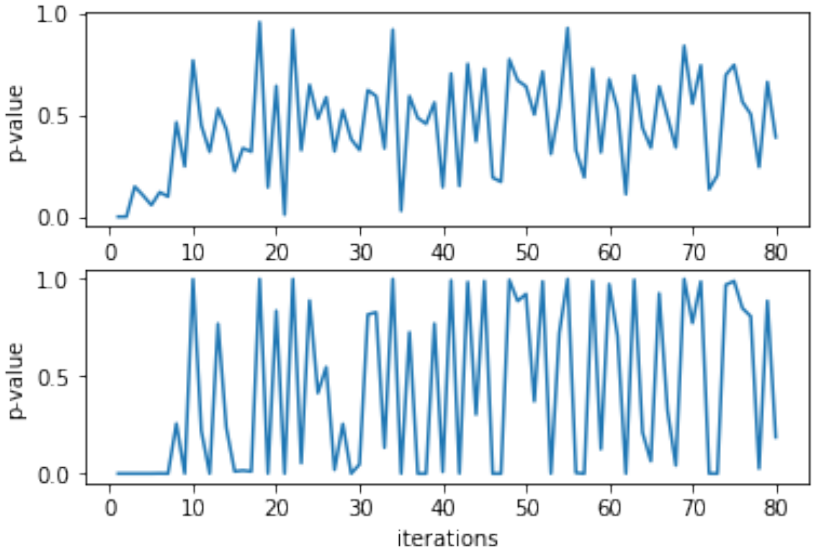


Figure 3.4: Global p-value of the permuted t-test with combine score (above) vs Fisher (bellow)

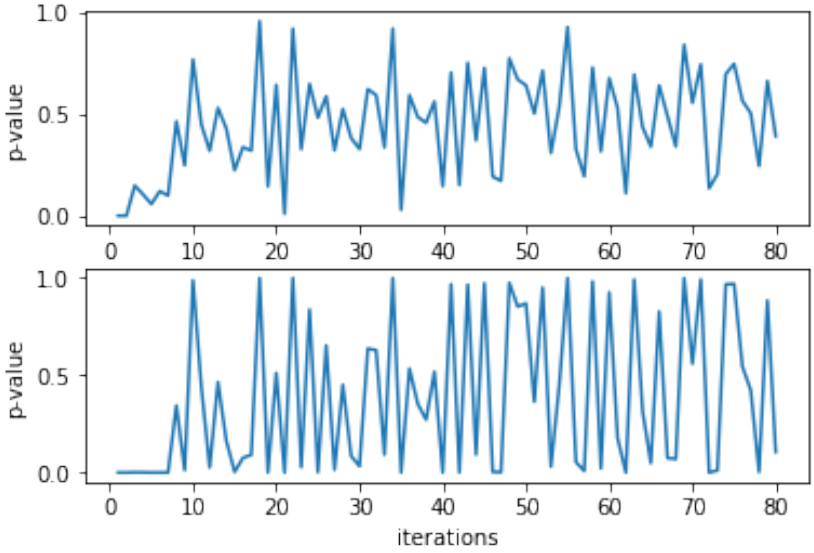


Figure 3.5: Global p-value of the t-test with combine score (above) vs Fisher (bellow)

3.3 Comparison of the methods inside a coalition

Now that preliminary tests have been done, we know which tests are more permissive and which are more restrictive. But we do not know already what it implies in term of performance or quantity of accepted data. In this section, we will test the different methods towards the coalition mechanism presented at the beginning of this chapter. We still use the same CNN as before with the same optimization method. There is 10 members, each one with a training set of 2000 images and a test set of 1000 images. So, the total of training images is 20 000. The parameter $nSim$ is equal to 200 and there is no malicious behaviours.

For each hypothesis method and p-values combination methods, we simulated four coalitions with the thresholds 0.4 and 0.1 and the training sizes of 250 and 1000 images. Concerning the ratio method, the thresholds are 0.9 and 0.7. In the next lines we will refer to four configurations:

- Configuration 1: a training size of 250 images and a threshold of 0.1 except for the ratio method where the threshold is 0.7
- Configuration 2: a training size of 250 images and a threshold of 0.4 except for the ratio method where the threshold is 0.9
- Configuration 3: a training size of 1000 images and a threshold of 0.1 except

MEP method	Configuration: 1	2	3	4
t-test with combine score	11000	15250	16000	19000
t-test with Fisher	13750	18500	14000	20000
global t-test	10750	14250	9000	14250
ratio	2250	7750	4000	6000
bootstrap test with combine score	5750	9500	4000	9000
bootstrap test with Fisher	8000	13000	9000	15000
Wilcoxon test with combine score	5500	8500	13000	14000
Wilcoxon test with Fisher	8250	12250	13000	15000
permuted t-test with combine score	2000	7500	4000	10000
permuted t-test with Fisher	4000	13500	9000	13000

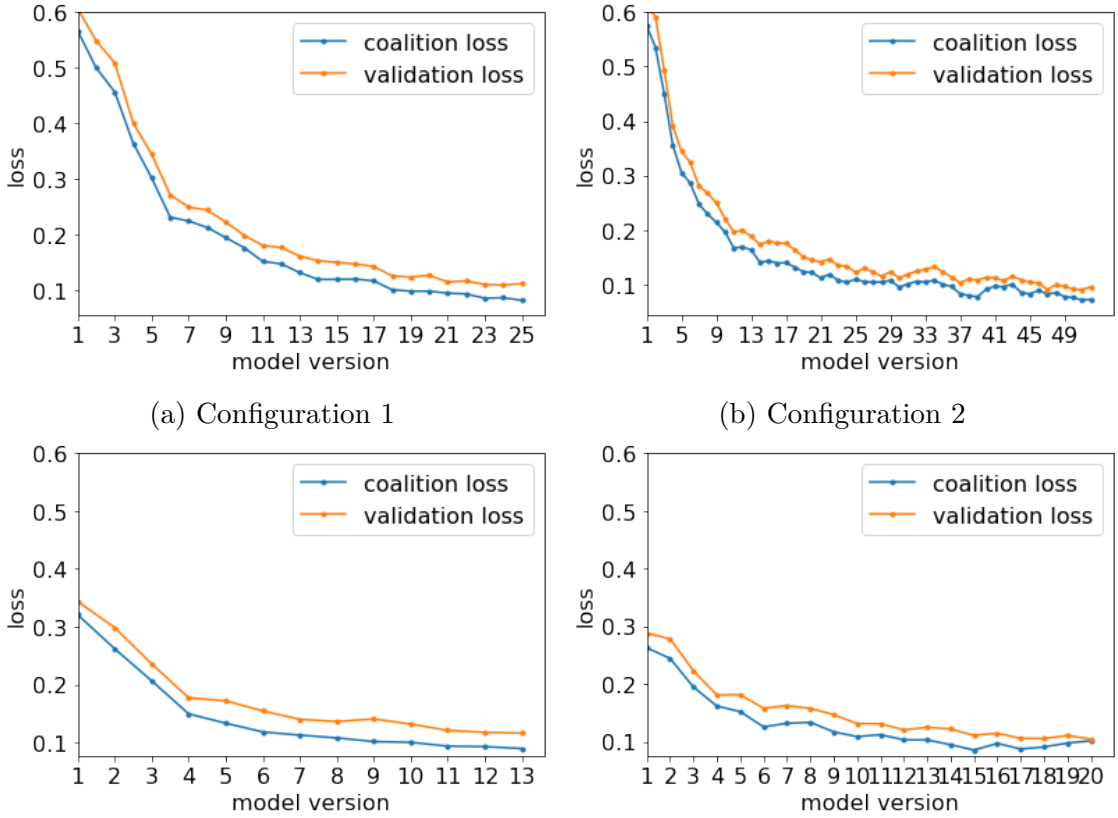
Table 3.3: Quantity of accepted data for the different MEP methods and configurations

for the ratio method where the threshold is 0.7

- Configuration 4: a training size of 1000 images and a threshold of 0.4 except for the ratio method where the threshold is 0.9

The quantity of accepted data resulting from each simulation is given in Table 3.3. Because, at this stage, only one simulation has been done for each configuration, we do not take strong conclusion of the performance of a configuration. But these tests allow to decide which methods we will keep for further analysis. We observe that the t-test method associated with Fisher or combine score allows to train the model on more data than the other methods. The loss in performance due to the global t-test method is not negligible. The highest quantity of accepted data for the ratio method is 7750 images (less than 40% of the training set), it happens with the 0.9 threshold and a training size of 250 images. Based on these results, we will restrain the further analysis to the t-test method with associated with Fisher or combine score and the ratio method. We decide to keep the t-test methods because they have the best performance in term of accepted data and the ratio method because it is the method that is used in the TCLearn paper.

In Fig. 3.6 we observe the performance of the coalition for different configuration of the t-test with Fisher method. The coalition loss corresponds to the loss evaluated on the members test sets and the validation loss corresponds to the loss evaluated on the validation set, a set of images that are not used for training or for the MEP. As explained in the MEP objectives (section 2.1), because the members test sets are used for the models acceptance, there is a risk that the



(a) Configuration 1

(b) Configuration 2

(c) Configuration 3

(d) Configuration 4

Figure 3.6: Evolution of the validation loss and the coalition loss using the t-test with Fisher method in different configurations

coalition model will indirectly fit on these test sets. On the four plots, we observe that the performance of the model are better on the members test sets than on the validation set. The validation set gives us an better estimate of the "true" performance of the model ($E(M)$) because it is not used to make decision during the coalition.

In order to obtain a better estimate of the coalition's performance for the different configurations and methods, we run multiple simulations. For each configuration and method, 5 simulations has been done with different training sets and test sets each time. Indeed, 5 MNIST data sets have been build from the original one. These data sets contain the images than the MNIST data set but in an different order. For each data set, the training set and the test set have been shuffled (separately). The resulting validation losses and quantities of accepted data of these simulation is given at the Fig. 3.7 and 3.8. In table 3.4, we computed the average loss, accuracy and quantity for each configuration and MEP method.

MEP method	Configuration number: 1	2	3	4
without coalition	0.082			
t-test with combine score	0.111	0.109	0.110	0.105
t-test with Fisher	0.115	0.106	0.112	0.102
ratio	0.214	0.140	0.194	0.144

(a) Validation loss

MEP method	Configuration number: 1	2	3	4
without coalition	98.56%			
t-test with combine score	97.48%	97.57%	97.75%	97.85%
t-test with Fisher	97.3%	97.6%	97.77%	97.89%
ratio	94.67%	96.68%	96.06%	97.08%

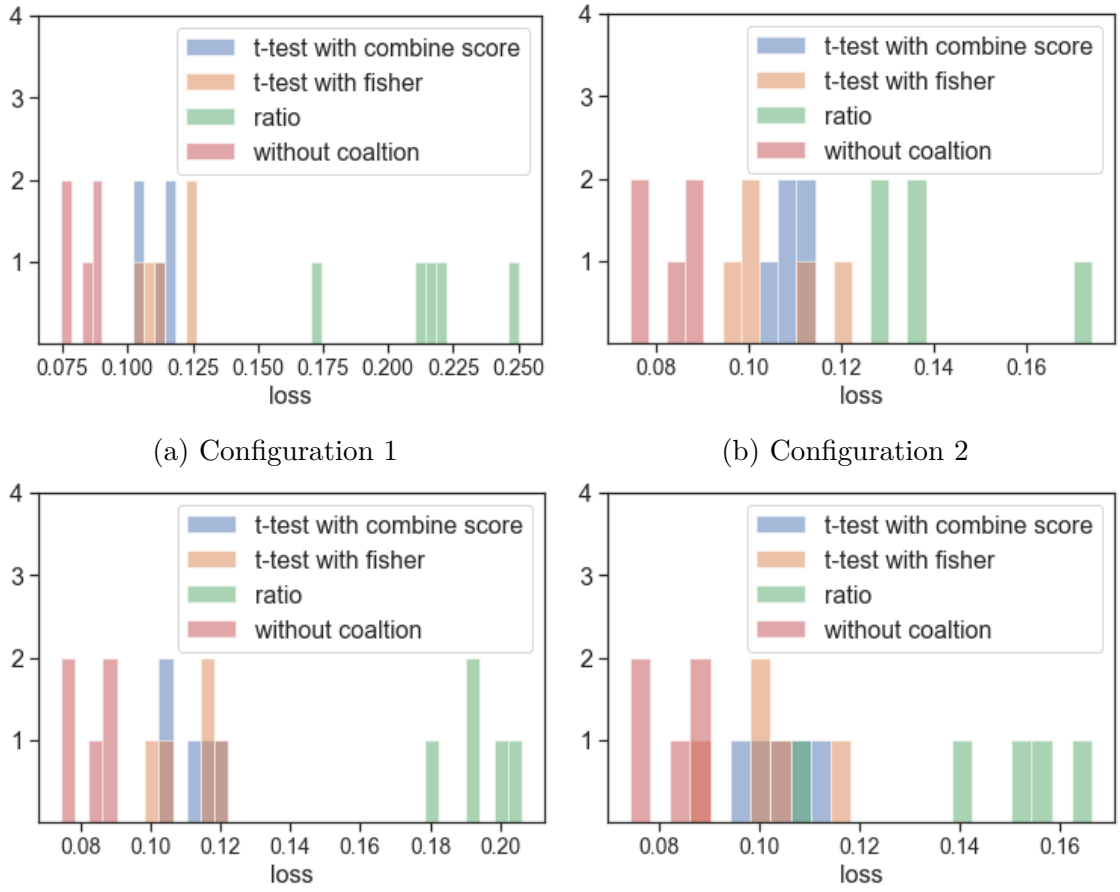
(b) Validation accuracy

MEP method	Configuration number: 1	2	3	4
t-test with combine score	12850	17400	13400	16000
t-test with Fisher	11700	18250	14000	17200
ratio	2700	6500	3400	7200

(c) Quantity of accepted

Table 3.4: Average of different coalition performance metrics applied to MEP methods with the four configurations

These results confirm the fact that the t-test methods are better than the ratio one. However, we are not able to conclude if one of the t-test method is better than the other. Indeed, by applying a t-test on the resulting losses of the two methods, we obtain a p-value between 46% and 66% depending of the configuration. We observe that on average an important part of the training data is accepted with t-test on configuration 2 and 4 (corresponding to a 0.4 threshold), 86% in average. Despite this good results, we notice an significant variability between the results from one simulation to another. For example, in the the configuration 4 of the t-test with combine score, we observe that the quantity of accepted data goes from 11000 to 20000 images. The configuration 4 leads to more variability than the configuration 2. As reminder, the difference between this configurations is the training size. With the ratio method, the quantity of accepted is way bellow. In the better configuration this quantity reach 7200 images which correspond to 36% of the whole training set.



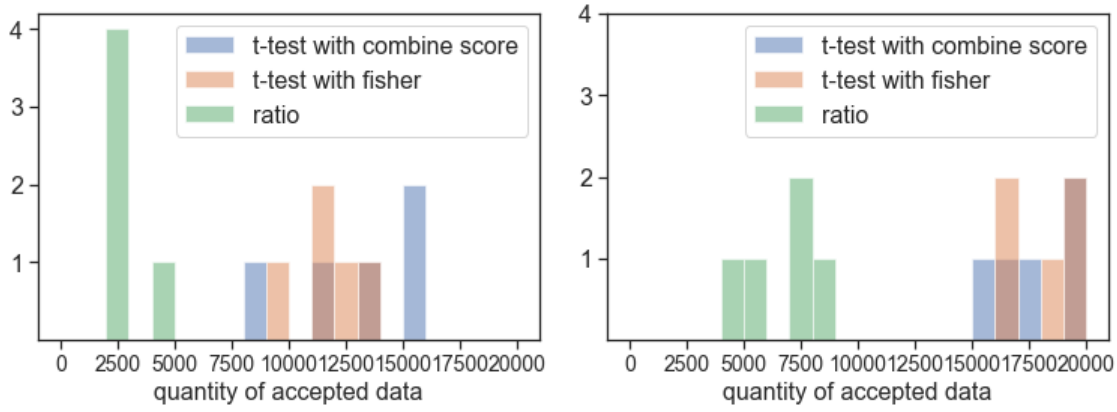
(a) Configuration 1

(b) Configuration 2

(c) Configuration 3

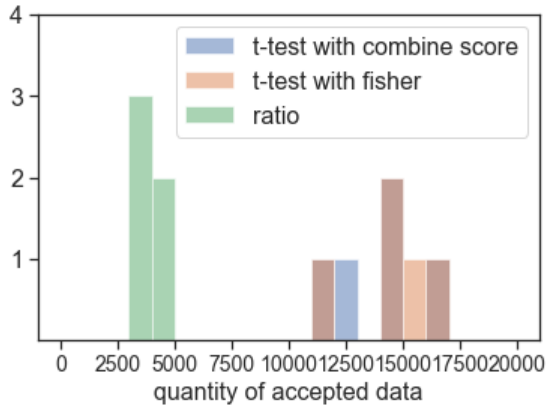
(d) Configuration 4

Figure 3.7: Distribution of the validation loss at the end of the coalition for different MEP methods and configurations

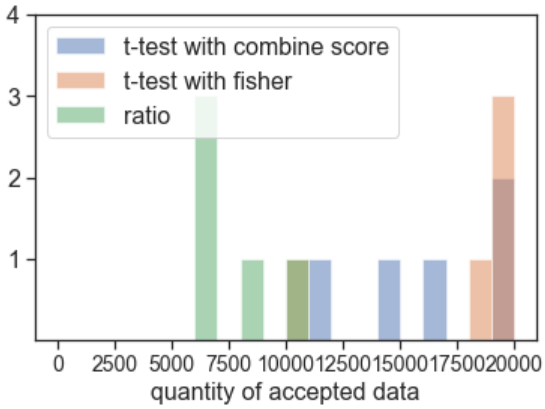


(a) Configuration 1

(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 3.8: Distribution of the quantity of accepted data at the end of the coalition for different MEP methods and configurations

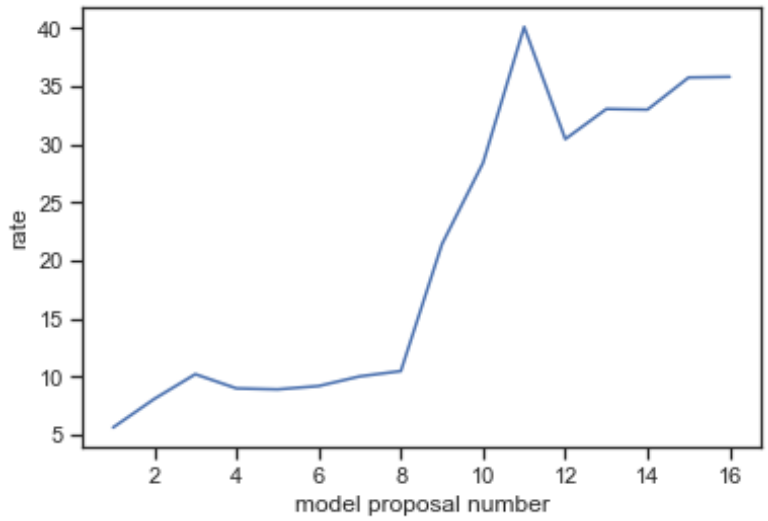


Figure 3.9: Growing rates of the model proposals with bad data using ratio method in the configuration 2

3.4 Simulation of training on wrong data

Thanks to the previous tests, we know the performance of the different methods and configurations. But we do not know yet if they are effectively resilient to training on bad data. In order to measure this resilience, we simulate multiple coalitions where a member training set is wrong. More precisely, we shuffle 5% of the image's labels in the member's training set. The member with the bad data is the member number two in the coaltion. These simulations have been done on the three MEP methods (the two t-test methods and the ratio method) with the configuration 2. We chose this configuration because it is associate to the highest threshold and so is the more permissive one. Unfortunately, this modification of a member's training set is too important for testing the coalition resilience mechanism. Indeed, as shown on Fig. 3.9, the growing rates of the model proposals with bad data are all much higher than 1. And so, as we may expect, the t-test methods always associated a p-value of 1 to these model proposals (which implies that no bad training have been accepted).

Chapter 4

Conclusion

4.1 Summary

The work that has been done in this thesis follows the work of the TCLearn paper. After going through the key concepts of machine learning, hypothesis testing and blockchain technologies, we explained the foundations of the TCLearn architecture. This allowed us to introduce the MEP that has been the main focus of the thesis. The objective of this protocol is to replace the method presented in the TCLearn paper for evaluating the quality of a machine learning training increment. The method presented in the paper is a naive method which is based on the growing rate between the evaluated loss of two models. We tried to improve the method by using statistical tools. More precisely, we investigate about the use of hypothesis testing methods for evaluating the probability that a model proposal will not improve the current model.

The first part of this investigation was to look at the challenges behind the evaluation of a model improvement. We choose to study the training of a CNN on the MNIST database. We observed that training a machine model does not implies a monotone decrease of the loss evaluated on a test set. We sometimes observed an increase of the loss. Moreover, we define a vector γ where each element corresponds to the difference between the loss of the current model and the loss of the model proposal evaluated on an image of a test set. We observed that the distribution of γ is far from normal. This is an obstacle for using the most common hypothesis testing method: the t-test.

Due to these observations, we selected four hypothesis testing methods , three of which are designed for non normality. The last one is the t-test. Despite the non normality obstacle, the t-test still suits for solving the problem thanks to

the large number of samples. We also shown the necessity of choosing a threshold behind 0.5 for the hypothesis testing methods and bellow 1 for the ratio method.

Finally, we ran coalition simulations with different configurations and methods in order to compare them and measure there performance. In the best configuration, we observed that the t-test associated with the Fisher's method allows to obtain 91.25% of accepted data. With the same training size (250) and with a threshold of 0.9, the ratio method leads to 32.5% of accepted data. Because we mainly focus on the comparison between the different methods and because of the large amount of time required to run simulations, we did not had time to perform enough tests concerning the resilience against wrong behaviours.

4.2 Further work

The next important step is to perform other tests on the MEP concerning the resilience to wrong behaviours. We already observed that a training on a data set with 5% of shuffled labels can be easily detected by the MEP because of the important growing rates that results from this training. Further work could be to apply the MEP on training done with smaller percentages of shuffled data and see when it becomes hard for the protocol to detect this altered data. Some additional features could be tested in order to improve the protocol such as :

- Instead of using a fixed test set, the validator randomly draw a fixed quantity of unused data from his training set each time he is selected for the MEP. This implies that the members should keep a defined quantity of unused training during the TCLearn protocol. The use of test sets that change each time may help to decrease the difference between validation loss and the coalition loss.

- Some federated learning solution (such as the Substra framework for example [28]) use averaging[1] of several model proposals to fasten the training. Indeed, it allows the members to train the current model at the same time. A solution for the TCLearn architecture could be to select multiple members for the role of partner at each turn and only average the accepted models. A secure aggregation method would be required [29].

In a more practical way, it could be helpful for further simulations to run them on distributed processes. This could be done by using a library such as pyspark, the python version Apache Spark [30] combined with a cloud computing service. We could also test the protocol with other data bases and other neural networks (it

[1] or more advanced aggregation techniques

seems better to stay focus on neural networks at this stage) in order to see if we obtain similar results.

Beside the MEP, the blockchain architecture has already been implemented on the TCLearn GitHub but the coalition and blockchain protocol still need to be implemented. Moreover in the simulations of this thesis, we did not use a differential private SGD algorithm. We also need to implement such an algorithm in TCLearn and decide of its parameters.

Bibliography

[1] Louis Columbus. Roundup of machine learning forecasts and market estimates — Forbes, 2020. [Online; accessed 21-May-2020]. URL: https://www.forbes.com/sites/louiscolumbus/2020/01/19/roundup-of-machine-learning-forecasts-and-market-estimates-2020/.

[2] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1310–1315. Ieee, 2016.

[3] Kiran Bhageshpur. Data is the new oil and that's a good thing — Forbes, 2020. [Online; accessed 21-May-2020]. URL: https://www.forbes.com/sites/forbestechcouncil/2019/11/15/data-is-the-new-oil-and-thats-a-good-thing.

[4] Regulation (eu) 2016/679 of the european parliament and of the council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). *Official Journal of the European Union*, L 119, 2016.

[5] Preciosa Coloma. Mining electronic healthcare record databases to augment drug safety surveillance. *PhD Manuscript, University Medical Center, Rotterdam*, 2012.

[6] June-Goo Lee, Sanghoon Jun, Young-Won Cho, Hyunna Lee, Guk Bae Kim, Joon Beom Seo, and Namkug Kim. Deep learning in medical imaging: general overview. *Korean journal of radiology*, 18(4):570–584, 2017.

[7] Wildiers Hans, Stordeur Sabine, Vlayen Joan, Scholten Rob, Van de Wetering Fleur, Bourgain Claire, Carly Birgit, Christiaens Marie-Rose, Cocquyt Véronique, Lifrange Eric, Schobbens Jean-Christophe, Van Goethem Mireille, Villeirs Geert, Van Limbergen Erik, and Neven Patrick. Breast cancer in

women: diagnosis, treatment and follow-up. Technical report, Brussels, 2013. Good Clinical Practice (GCP). URL: https://kce.fgov.be/report/143ER-.

[8] Maciej A Mazurowski, Mateusz Buda, Ashirbani Saha, and Mustafa R Bashir. Deep learning in radiology: an overview of the concepts and a survey of the state of the art. *arXiv preprint arXiv:1802.08717*, 2018.

[9] Website of mammoscreen (a therapixel company). [Online; accessed 12-August-2020]. URL: https://www.mammoscreen.com/.

[10] S. Lugan, P. Desbordes, E. Brion, L. X. Ramos Tormo, A. Legay, and B. Macq. Secure architectures implementing trusted coalitions for blockchained distributed learning (tclearn). *IEEE Access*, 7:181789–181799, 2019.

[11] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[12] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[13] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[14] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[15] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[17] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 120–130, 1999.

[18] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $o(1)$ rounds, expected $o(n^2)$ communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[19] Jiasi Weng, Jian Weng, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[20] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33(3):499–516, 1986.

[21] Thomas Lumley, Paula Diehr, Scott Emerson, and Lu Chen. The importance of the normality assumption in large public health data sets. *Annual review of public health*, 23(1):151–169, 2002.

[22] Alan F. Karr. *Classical Limit Theorems*, pages 183–216. Springer New York, New York, NY, 1993.

[23] Ralph B. D'Agostino. An omnibus test of normality for moderate and large size samples. *Biometrika*, 58(2):341–348, 1971.

[24] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[25] Frank Konietschke and Markus Pauly. Bootstrapping and permuting paired t-test type statistics. *Statistics and Computing*, 24(3):283–296, 2014.

[26] Lili Su and Nitin H Vaidya. Defending non-bayesian learning against adversarial attacks. *Distributed Computing*, 32(4):277–289, 2019.

[27] RC Elston. On fisher's method of combining p-values. *Biometrical journal*, 33(3):339–345, 1991.

[28] Mathieu N Galtier and Camille Marini. Substra: a framework for privacy-preserving, traceable and collaborative machine learning. *arXiv preprint arXiv:1910.11567*, 2019.

[29] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[30] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

Appendix A

The CNN used in the tests

The model has been build using Keras version 2.2.4 and TensorFlow version 1.15.0 on Python. The GPU version of TensorFlow has been used with CUDA to hasten the computations. The neural network layers has been implemented in the following order :

1. The input layer : the input is a 28×28 matrix of float numbers between 0 and 1 that corresponds to an handwritten digit image.
2. 2D convolution : 32 filters of size 3×3 with a bias. The activation function is the ReLU function.
3. 2D convolution : 64 filters with the same configuration than the previous layer.
4. 2D max-pooling: the pool is of size 2×2
5. Dropout: each input has a probability of 0.25 to be set at 0 during the training phase. This is a commonly used regularization method to prevent overfitting in neural networks.
6. Flatten
7. Dense layer of neurons : Each output of the flattening layer is connected to the 128 neurons of this layer. The action function is the ReLU function.
8. Dropout: each input has a probability of 0.5 to be set at 0 during the training phase.
9. Dense layer of neurons : Each output of the previous layer is connected to the 10 neurons of this layer. The activation function is the softmax function. The 10 outputs of this layers are the outputs of the neural network and correspond to the probability of the 10 classes (the digit from 0 to 9).

Layer(type)	Output Shape	Output Shape
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 128	1179776
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290
Total params: 1,199,882 Trainable params: 1,199,882 Non-trainable params: 0		

Table A.1: CNN resume from Keras

The model is trained using batches of 64 images and 20 epochs. The optimizer is Adadelta which is provided by Keras.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**