BACHELOR'S DEGREE THESIS

Statistics and Economics

Title: Two-Layer Feed Forward Neural Network (TLFN) in Predicting Loan Default Probability

Author: Marcos Rusiñol de Rueda

Advisor: Esteban Vegas, Ferran Reverter, Salvador Torra

Department: University of Barcelona & Polytechnic University of Catalonia

Academic year: 2018-2019

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Two-Layer Feed Forward Neural Network (TLFN) in Predicting Loan Default Probability

Marcos Rusiñol de Rueda (Doble degree in Economics and Statistics)
Esteban Vegas, Ferran Reverter, Salvador Torra
University of Barcelona & Polytechnic University of Catalonia

22 of June, 2019

Abstract

The main objective of the thesis is to apply a Neural Network (NN) approach in the PD used to assess whether a credit operation is granted or not. That is, given an operation, the NN model should predict whether it is granted [0], or not granted [1]. Credit Risk Models and Deep Learning concepts are also explained.

Keywords: Credit Risk Models, Deep Learning, Neural Networks, TLFN

Contents

1 Introduction

1.1 Goals and Motivation

The use of Artificial Intelligence techniques in the banking world is a major challenge today. There are several regulations (European Banking Authority, EBA 2018) that oblige entities to explain all the procedures they do, such as their methods of analysis in the calculation of default probabilities. It is commonly known that AI techniques are called "black boxes" (Daily Wisdom 2018), that is, you know the input and the output, but you are not capable to very understand the inset. So, the question is: How do you defend that an algorithm based on AI techniques gives good results? Indeed, in most cases, these results may be even better than traditional techniques. This thesis aims to take a further step towards the implementation of these techniques.

The idea of the thesis comes from the importance that the AI techniques are currently having and, above all, from their repercussion in the business world. These techniques are revolutionizing the world: more optimized processes, and better quality of life; which results in higer business profits and social benefits (M. West, Darrell and R. Allen, John 2018). In this broad sector, in particular, I am interested in the world of credit risk. The financial sector is a very traditional sector, which means that these modern techniques cannot be implemented overnight. However, at the same time, it is a very important sector in the Spanish economy (Asociación Española de Banca, 2018). Therefore, if we get to optimize this sector by implementing new and better techniques, it is very likely that the spanish economy will also improve.

Briefly, the Probability of Default (PD) is the estimation of the probability that a client defaults on its credit obligations. There are two ways to use the PD. A PD that is calculated to be incorporated into the capital or provisions of the entity, and another PD that is used in the credit rating for the creation of the scorecard. In this thesis, I will roughly explain the first type, but focus on explaining deeply the second. In other words, this second approach of the PD, its whether or not an entity grants a loan to a new client.

Following this line, the main objective of the thesis is to apply a Neural Network approach in the PD used to assess whether a credit operation is granted or not. That is, given an operation, the NN model should predict whether it is granted [0], or not granted [1]. Moreover, this thesis is intended to provide an explanation of the concepts of Credit Risk, Machine Learning and Deep Learning. As well as to carry out a simple practical implementation. Following this approach, the less error there is in an entity's PD predictions, the better the entity will be able to control and manage its risks and, consequently, have a better image of reality.

1.2 Structure

The present thesis has been divided into a theoretical part and a practical part. In the theoretical part there is an explanation of credit risks concepts, Machine Learning techniques and a deeper analysis of the Deep Learning field. This part corresponds to 65% of the thesis. The remaining 35% is dedicated to the practical part. I will implement a Two-Layer Feed Forward Neural Network (TLFN) to predict credit default.

More specifically, the theoretical part contains an explanation of the importance of credit risk management in institutions, as well as Machine Learning techniques, and their repercussion in the economy. Next, and as the most extensive point (representing 35% of the thesis), an in-depth analysis of Deep Learning techniques is carried out. Along these lines, and following the scheme

presented, the thesis ends with an implementation of a TLFN to predict credit default. Measures to improve this network and its implementation are also proposed.

1.3 American Mathematical Society classification

The thematic classification of this thesis according to the American Mathematical Society corresponds to the section of Computer Science (68-XX), concretely:

- Artificial intelligence (68Txx) - Algorithms (68Wxx)

2 Statistics and Economy: Credit Risk Models

2.1 Credit Risk Models

A model is a quantitative method that applies statistical, economic, financial, or mathematical theories, techniques, or assumptions to process data, and produces quantitative estimations.

A good model must contain and guarantee representative estimations of reality and a good governance of itself. Following this line, a good credit risk model:

1. **It has to be integrated in the management.** It must include and integrate the estimation policies into the different credit risk management processes.
2. **It must be simple.** This means that we must avoid elements that introduce complexity without seeing improvements. Actually, this is an interesting point in this thesis, since it is intended to propose a more complex model measure than the one currently used.
3. **It has to be robust.** It must comply with the mathematical hypotheses in order to obtain consistent estimations.

Indeed, the robustness and suitability of the model are very legal controlled aspects. In Spain, the framework for validating any institution's models must be based on six areas set by the Bank of Spain in its documents (Banco España 2016b). These areas are: methodologies, documentation, data used, quantitative aspects, qualitative aspects and their technological environment.

Credir risk models are an essential and very important aspect for any bank. Not only because they are forecasting and provisioning methods, but also because they are required by regulations for the application of certain standards and procedures (e.g. the European Central Bank stress test (European Central Bank, 2019)).

2.1.1 What is a Financial Risk?

A financial risk may be understood as any negative impact on the expected results of any entity due to a series of variables or factors that are little or not controllable. The main risks to which an entity is subject are detailed in the following table:

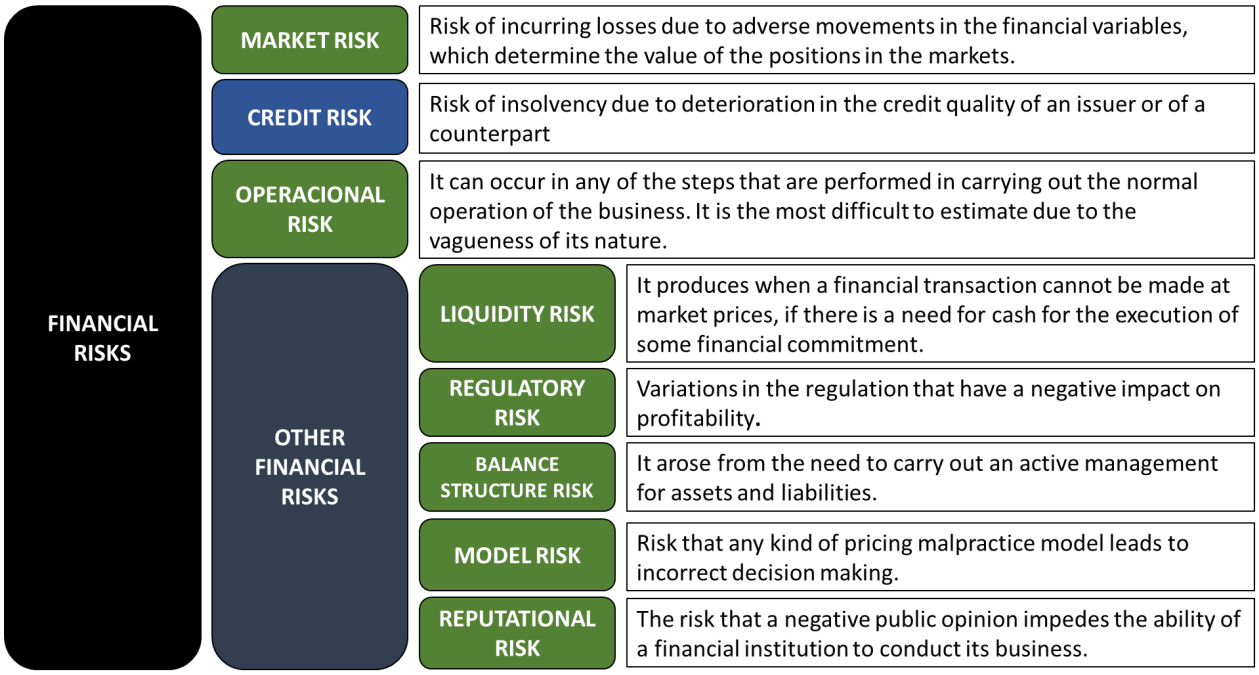


Figure 1: Financial Risks. Source: Own Elaboration (based on Deloitte presentations)

This thesis focuses on credit risk; that is, the risk that arises from the insolvency of an issuer. Credit risk is the risk of the probability that a counterparty will default on its credit obligations. In other words, it refers to the non-payment of interest or principal on a loan by the counterparty to the borrower. The period to consider default varies to the type of product, it might be 90 days, 180 days, etc (Comité de Supervisión Bancaria de Basilea 2014), actually, each country has their own regulations.

2.1.2 Credit risk: Why is its management important?

As I defined before, a risk is a negative impact. It is an event that, if it occurs, will produce a threat, not only to the entity's results, but may also affect the entire world (Senior Supervisors Group 2009).

There are two types of credit risk:

The systemic risk, defined as the risk inherent in the market, i.e. the risk that affects the entire market regardless of the characteristics of the borrower and its products. The price of oil, a country's GDP, and its risk premium are examples of systemic risks.

The idiosyncratic risk, defined as the risk that encompasses the set of factors specific to a company or industry, which affects the profitability of its shares or bonds. For example, the leverage ratio is an idiosyncratic risk.

Following this line, credit institutions have the obligation to cover their expected and unexpected risk losses. **For expected losses**, BdE Circular 4/2016 (Banco España 2016a), IAS 39, and IASB document IFRS 9 specifies how an entity should classify and measure financial assets, financial liabilities and make sure that are reflected in the **entity's provisions.** On the other hand, **unexpected losses** are regulated by Directive 2013/736/EU (Comisión Europea 2013) and subject to

periodic Stress Test exercises of the EBA and must be reflected in the **capital of the entity.**

The 3 main points of credit risk management are:

1. The assessment of all the risks faced by the company
2. The communication of identified risks to the managers, in order to make decisions.
3. The control and monitoring of these risks.

On the other hand, the risk management process has the following 5 steps:

1. Risk identification.
2. Quantification of the exposure.
3. Effects of such exposure.
4. Development of a strategy to mitigate this risk.
5. Evaluation of the performance of the measures taken.

Once the entity has identified the risk, they must estimate the expected and unexpected losses of such risk. These estimations will be proved by the regulator through some tests, such as the stress test. Below there is a scheme of the main important points in the risk management process:

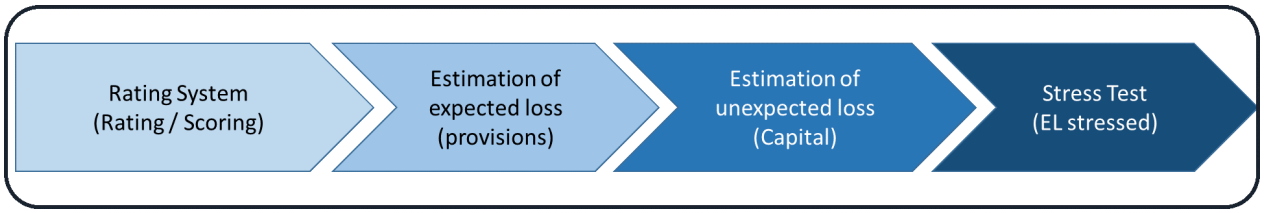


Figure 2: Risk Management. Source: Own Elaboration (based on Deloitte presentations)

2.1.3 Stress test

There are many ways to regulate risk management by institutions, both national and international. One of the measures adopted by some of these institutions is the so-called stress test.

The stress test has the duty to regulate and check how an entity manage their risks. Specifically, they should check if the sensitivity of the risk to specific economic situations is good enough.

The regulator measures the impact of economic scenarios on financial exposures and collaterals. They consider shocks in the parameters, such as the Probability of Default (PD); as well as shocks in macroeconomic factors, such as inflation, house prices, unemployment rate, and so on.

The stress test exercise has the following scheme:

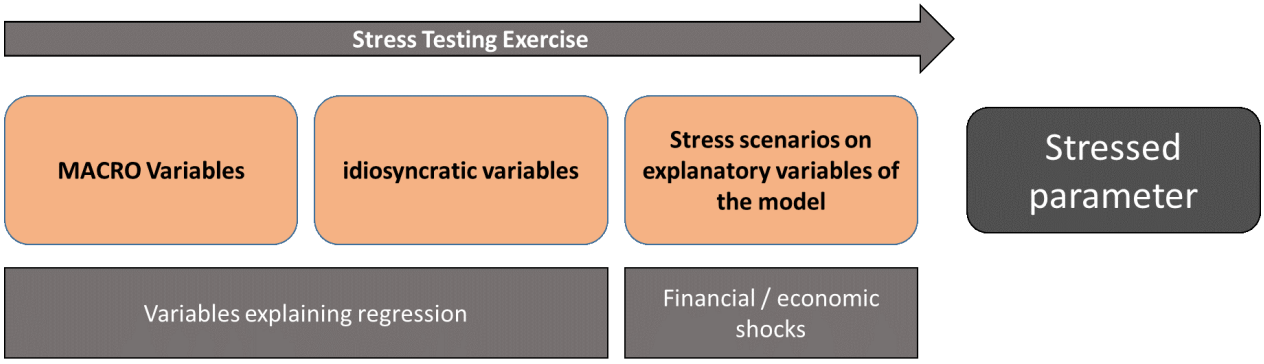


Figure 3: Stress Test. Source: Own Elaboration (based on Deloitte presentations)

2.1.4 Parameters

The measurement and management of the aforementioned risk is based on the estimation of the following parameters:

1. **Credit Rating:**

In order to evaluate credit risk, it is necessary to have a note or mark that identifies how good it is a product or a counterparty. There are two models dedicated to this objective: rating, which conveys the creditworthiness of a business or a government, for example the S&P models; and scoring, which is the numerical expression of creditworthiness used for business or individuals.

2. **Probability of Default (PD):**

Before I begin to explain this concept I would like to emphasize an important concept to make things clearer to the reader. There are two ways to use the PD. On one hand, there is the PD that is calculated to be incorporated, through the calculation of the EL and UL, in the provisions or capital of the entity. For the calculation of this first PD some procedures are used that are not the object of this thesis. On the other hand, there is the PD used in credit classification (scorecard). In simple words and in short, whether or not we grant a loan to a new client or operation. I will roughly explain the first, and I will deeply focus on the one used to calculate the scorecard.

Default is the fundamental event to model in the measurement of Credit Risk. What is, therefore, default? A situation will be considered default if a client is in a situation of delinquency management for more than some specific days, as well as those individuals (customers) who are unlikely to be able to meet their payments. According with the article 178 (b) of the EU regulations (Comisión Europea 2013):

1. A default shall be considered to have occurred with regard to a particular obligor when either or both of the following have taken place:

(a) the institution considers that the obligor is unlikely to pay its credit obligations to the institution, the parent undertaking or any of its subsidiaries in full, without recourse by the institution to actions such as realising security.

(b) the obligor is past due more than 90 days on any material credit obligation to the institution, the parent undertaking or any of its subsidiaries. Competent authorities may replace the 90 days with 180 days for exposures secured by residential or SME commercial original estate in the retail exposure class, as well as exposures to public sector entities)[…].

As seen in the previous points, the PD has an impact on various aspects of risk management; such as the calculation of accounting provisions, the calculation of regulatory capital, the management, admission and monitoring of customers and stress test exercises. Therefore, the PD is a time-dependent parameter. It is important that the procedure chosen to model the PD parameter shows both the systemic risk of each temporal moment and the idiosyncratic risk of each of the counterparties in the segment.

Therefore, the Probabiliry of Default (PD) is the estimation of the probability that a transaction or client will default on its credit obligations. Default probability models use a combination of data on default behaviour, such as the trend in its distribution; financial information, such as ratios in the case of companies; and subjective information, i.e. entity´s judgement.

There are three steps in modelling the PD:

1. **Model construction:** Develop a statistical model by combining the data obtained in order to obtain a credit rating for a client or operation. This is the one used to create the scorecard. This is the step I will focus on this thesis.
2. **Calibration of the model:** Once we have created the scorecard model, we relate this credit rating developed in the construction of the model to the probability of default.
3. **Validation of the model:** Finally we validate the model and conclude whether or not it is suitable for management, as well as the calculation of capital using statistical techniques.

- In the calibration step, we can diferentiate from 3 different ways to calibrate the PD:

Firstly, it has to said that, according to EBA regulations, in the estimation of PD and LGD, article 55 states: *A PD model may contain several different methods for ordering debtors or exposures, as well as several calibration segments.* The most common are:

(a) *PD PIT (Point in Time):* Defined as the probability of observing an event of default in the following 12 months. PD PiT tries to capture the variations in the economic cycle and, thus move along with them. High capital requirement when the PD estimations are high.

(b) *PD TTC (Through the Cycle):* PD adjusted to the economic cycle, this PD is based on a Central Trend (CT). This CT is an average default rate observed in the period of an economic cycle. It is usually required to obtain the time series of the default rates, to be able to obtain the Central Trend. The PD TTC predicts the average default rate performance for a particular customer over an economic cycle, and ignores short run changes to a customer PD. (Actuaries Institute 2012).

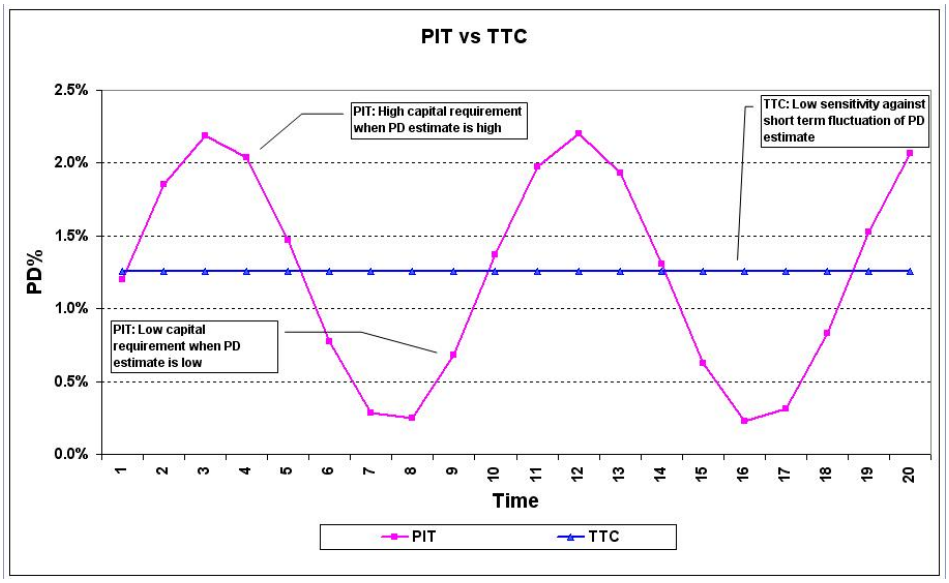


Figure 4: TTC VS PIT PD. Source: Riskopedia Blog

(c) *PD Lifetime:* Defined as the probability of observing an event of default throughout the life of the operation.

3. **Loss Given Default (LGD):**

Evaluates the loss actually incurred after the default event occurs. This represents the percentage of the risk exposure that is not expected to be recovered in the event of default. It is usually calculated using the discount of flows observed throughout the process of recovering contracts that have fallen into default at some time. For those portfolios with few defaults (low default portfolio, LDP), external sources of information are used, as they do not have sufficient historical information. This estimate includes direct credit risk mitigators such as guarantees. Therefore, LGD estimates vary depending on the underlying transaction, the collateral committed, and the security and type of loan claim.

$$LGD = 1 - RR = 1 - \frac{PV_{recoveries} - PV_{expenses}}{EAD}$$

Where:

PV = Present Value (Discounted Cash Flow).

RR = Recovery Rate.

Expenses = Direct and Indirect expenses.

An example: If the client defaults with an outstanding debt (EAD, explained later) of 80,000 dolars and the bank is able to sell the security for a net price of 60,000 (including costs related to the repurchase), then the LGD is 25% (= 20,000 / 80,000)

The LGD may be calculated under the **foundation approach** or under the **advanced approach**.

The foundation approach is for corporates, sovereign and bank exposures. Under this approach, the Bank for International Settlements (BIS) prescribes fixed LGD ratios for certain classes of unsecured

exposures. For example, the article 87 from the Consultative Document of the Internal Ratings-Based Approach (Basel Committee on Banking Supervision, 2001) states this about the treatment of unsecured exposures and non-recognised collaterals:

> *Senior claims on corporates without specifically recognised collateral will be assigned a 50% LGD. Subordinated claims on corporates without specifically recognised collateral will be assigned a 75% LGD.*

On the other hand, in the advanced approach is the bank itself that determines the appropiate LGD to be applied to each exposure. Most banks want to apply this approach, so they may control better the LGD and play strategically, and, for example, differenciate LGD based on transaction characteristics (Wikipedia). However, their LGD estimations should be validated by the supervisor.

4. **Exposure at Default (EAD):**

It is the amount of debt pending payment at the time of default of the customer. In the exposure of transactions (balances) in most products, the EAD is the estimate of the exposure that would occur when the default event occurred. The exposure of the contract usually coincides with the balance of the contract, but for products that have limits (such as cards or credit lines) the exposure incorporates the potential increase in the balance that could occur from a reference date to the time of default. The EAD is multiplied by a CCF. The Credit Conversion Factor (CCF) is the percentage of the undrawn balance that is expected to be used before default occurs (BBVA 2013).

These parameters are used to estimate the Expected Loss (EL) and Unexpected Loss (UL) of a credit institution. This will then be necessary for the calculation of the institution's provisions and capital. Specifically, they are related as follows:

$$Expected\ Loss\ (EL) : PD \cdot LGD \cdot EAD$$

They are multiplicative factors among them. The PD is a probability between [0, 1]; the LGD, a percentage; and the EAD a value (in monetary units). For an individual loan, PD is, by definition, independent of EAD and LGD.

$$Unexpected\ Loss\ (UL) : LGD \cdot EAD \cdot f(PD)$$

The above equation can be rewritten as:

$$UL = EAD \cdot \sqrt{[(PD^2 \cdot \sigma^2_{LGD}) + (LGD^2 \cdot \sigma^2_{PD})]}$$

Where:

$$\sigma^2_{LGD} = \frac{(LGD)(1 - LGD)}{4}$$
$$\sigma^2_{PD} = (PD)(1 - PD)$$

2.1.5 Portfolio segmentation: Construction of the Scorecard

The objective of the portfolio segmentation is to collect all transactions scored as default (*scoring*), in the last 12 months, and segment them into scoring brackets through their characteristics. This is done to infer the pattern of default in credits.

Portfolio segmentation can be done using parametric and non-parametric methods.

- **Parametric methods:** These are the ones commonly used in entities. They have, in general, fewer numerical requirements and are well interpretable. These include among others: linear regression, logistic regression and discriminant analysis.
- **Non-parametric methods:** They are not usually implemented in entities due to the fact that some require a high level of knowledge, which is still being studied today. However, these techniques can be much better classifiers. These techniques include decision trees, neural networks, which are discussed further ahead of this method; and Support Vector Machine (SVM) methods. The latter try to identify the unique hyperplane that generates the maximum separation between the elements of each class.

It is precisely this point the main object of the thesis. Currently, the portfolio segmentation (the selection of variables and their analysis) is done using the parametric method of logistic regression (logit method). This is because the response variable of a logistic regression is a probability (limited to the interval [0.1]) and is, therefore, very useful and easy to interpretate. The current thesis aims to do it through the non-parametric approach using neural networks.

2.1.5.1 Example

In order to make clear the procedure for the creation of the *scorecard*, I will explain it by means of a simple example (Baesens, Roesch, and Scheule 2016).

Let suppose we have a database with information on customer operations for the last 5 years. In the rows we have the operations, and in the columns we have the variable AGE and SALARY and if that operation has incurred or not in DEFAULT.

It must be said that some portfolios are treated by client. In some portfolios, even if a client has more than one transaction, it is valued as one. Therefore, in this thesis I will consider that we work by operation (one client may have more than one transaction).

Before starting, the data would be cleaned, eliminating those variables that have a high percentage of errors, among other things.

The first step we would have to do is a preprocessing of the data and a descriptive univariate analysis of the variables. In this way, we get an idea of the data we have and we can detect anomalies, such as outliers, missings, etc. and we can treat them.

The next step is to split each variable into segments. That is to divide each variable that we have in buckets. Currently, although other techniques could be used, it is done in a balanced way. They create a histogram and the variable is divided according to the groups that the analyst wants to obtain (this point is quite subjective). There are other advanced techniques, such as decision trees, which are used to set those sections or buckets.(Cañete 2018). Therefore, our variable AGE would be divided in: 18-22; 23-26; 27-29; … ; 44+

Once we have divided each variable into buckets, we calculate the Weights of Evidence (WOE) in the following way (the detailed explanation for the calculation of the WOE is done below). For the

variable AGE:

Table 1: WOE for AGE

AGE	COUNT	DISTRIBUTION.OF.COUNTS	GOODS
Missing	50	2.5 %	42
18-22	200	10.00 %	152
23-26	300	15.00 %	246
27-29	450	15.00 %	405
30-35	500	25.00 %	475
36-43	350	17.50 %	339
44+	150	7.50 %	147
	2000		1806

Table 2: WOE for AGE (II)

DISTRIBUTION.OF.GOODS	BADS	DISTRIBUTION.OF.BADS	WOE	IV
2.33 %	8	4.12 %	-57.28 %	-0.0103
8.42 %	48	24.74 %	-107.83 %	0.1760
13.62 %	54	27.84 %	-71.47 %	0.1016
22.43 %	45	23.20 %	-3.38 %	0.0003
26.30 %	25	12.89 %	71.34 %	0.0957
18.77 %	11	5.67 %	119.71 %	0.1568
8.14 %	3	1.55 %	166.08 %	0.1094
	194			0.6295

Source: Credit Risk Analytics: Measurement Techniques, Applications, and Examples in SAS

The good ones are defined as the clients who returned the loan and the bad ones as the clients who defaulted.

The WOE are calculated doing data transformation. Normally, a logarithmic transformation. Therefore, in this case, the WOE would be calculated as:

$$WOE = ln\left(\frac{Distribution\ of\ Goods}{Distribution\ of\ Bad}\right) \cdot 100$$

Thanks to this logarithmic transformation, a positive WOE means that the distribution of good operations is greater than the distribution of bad operations. The WOE indicate the predictive power of each bucket. For example: For the bucket 18-22, there are 200 clients. From these 200 clients, 152 are good (8.42 %), and 48 are bad (24.74 %). Therefore, the WOE of this bucket would be:

$$WOE = ln\left(\frac{8.42}{24.74}\right) \cdot 100 = -107.83\ \%$$

At this point there are two ways to proceed:

The first approach (not used in this thesis) is to choose those buckets which have a positive WOE. We would want to choose those with a positive WOE because we want our model to best predict those customers or transactions that will repay the loan. Therefore, the positive WOE are the ones that are going to contribute more information to the logistic model that will be done later. In the example, these would be the ages 30-35, 36-43 and 44+. What we would do is select (filtering) the operations of the ages between 30 and 44+ of our AGE variable. We would do the same procedure with the SALARY variable. Therefore, the number of rows in our dataset will decrease from the original dataset. Indeed, we might even have some problems if there are no intersactions among the variables from our dataset. I mean, suppose that for the variable SALARY we just have one bucket which its WOE is positive, lets say this is the bucket from 25.000-35.000 dolars. From the AGE variable, lets suppose we also just have one bucket with a positive WOE, lets say is the bucket from 20-30 years old. Imagine that there are no rows with a SALARY between 25.000-35.000 dolars, and also with an AGE between 20-30 years old. Then the filtered dataset will be reduced to zero rows.

The second approach is to replace all values contained in the bucket with their corresponding WOE. In the example, all individuals between the ages of 18 and 22 would be replaced by -1.0783. That is, an individual who is 21 years old, in that field, would now have a value of -1.0783. We would do the same procedure with all the buckets created in the variable AGE and the same for the variable SALARY. Thus, we would totally transform the whole database and get the variables WOEAGE and WOESALARY. Then, in this second approach, a variable selection measure (called **Information Value (IV)**) is also calculated. This measure is very popular in the scorecard construction. The idea of implementing the IV as the selection value of variables to be introduced in the model is to collect those variables that really have a minimum predictive power.

It is calculated as follows:

$$IV = \sum_{i=1}^{k} (Distribution\ of\ Goods_i - Distribution\ of\ Bad_i) \cdot WOE_i$$

Where k = Number of buckets in each variable.

Normally, the criteria to follow is the following (Brotherton, David 2013)

Table 3: IV

Items	Features
< 0.02	Non-predictive variable
$0.02 < \text{IV} < 0.1$	Weakly predictive variable
$0.1 < \text{IV} < 0.3$	Medium predictive variable
> 0.3	Strongly predictive variable
> 0.5	Variable that could be too good

Source: Own Elaboration

In the example given the IV = 0.6295. Therefore, the AGE variable is strongly predictive (in fact,

it could even be over-predicting [1]) and we would introduce it in the model. We would do the same calculation for the SALARY variable and decide whether or not to introduce it depending on its value IV.

This second approach has the advantage of being simple to implement, it also allows comparability of results, since WOE are standardized values; and permits to group variables of very dispersed discrete values into categories, since the WOE expresses information about the entire category.

It must be said that this second method also has its disadvantages. First, we lose information about the variables, since we totally change their values; second, we cannot consider a correlation between these variables; and finally, it is quite probable that the model tends to over-fit.

There are several reasons of why it is good to use the WOEs in the logit model. Reviewing the literature, we see how in a study (Majer 2006) in which the same logit model is compared using WOE variables (using the second approach previously defined), and continuous variables, results in that the model estimated from the WOEs values has a higher percentage of hits (bad as bad and good as good), than the model with continuous variables. In addition, it obtains higher values in the validation.

Once the transformed independent variables are obtained (regardless of the approach chosen to transform them) we make a logistic regression, usually with a stepwise selection method. In the example, a logistic regression with the variables WOEAGE, the WOESALARY, and if it is or not DEFAULT.

$$\pi = \ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_i \cdot x_i$$

where:

$$p = P(Default = 1|X)$$

The calculated WOEs will be used as independent variables.

Therefore, we get a PD as follows:

$$PD = \frac{1}{1 + exp - (\beta_0 + \beta_1 \cdot WOEVar_1 + ... + \beta_i \cdot WOEVar_i)}$$

Up to this point we have created a probability model for each new operation. Then, lets assume that the outputs of our logistic regression are as follows:

[1] this is due to the fact that, in this example, we just have two variables

Table 4: Output from Logistic Regression

Variable	Beta	pvalue
Intercept	-1.73	<.0001
WOEAGE	-0.654	<.0001
WOESALARY	-0.706	<.0001

Source: Own Elaboration

We therefore observe a negative relationship between WOEs and the logit function. It is logical, since when the WOE increases, the quotient between the probability of default and the probability of non default decreases.

The next step is to obtain the scorecard. At this point you can use any algorithm that is allowed by law. One way to do this would be by estimating the factor and offset (Nieto 2010). The scorecard is mainly the result of a rescalation and a translation of the logistic model.

Solving this system of equations:

$$\begin{cases} \text{Score} = \text{Offset} + \text{Factor} \ \ \ln(\text{odds}) \\ \\ \text{Score} + \text{Po} = \text{Offset} + \text{Factor} \ \ \ln(2 \ \ \text{odds}) \end{cases} \tag{1}$$

We get:

$Factor = \frac{Po}{ln(2)}$

$Offset = Score - Factor \cdot ln(odds)$

$Score = Offset + Factor \cdot ln(odds)$

$Score_{i,j} = Offset + Factor \cdot \left(\hat{\beta}_0 + \hat{\beta}_i \cdot WOEVar_{i,j} \right)$

$Score_{i,j} = Offset + Factor \cdot \hat{\beta}_0 + Factor \cdot \left(\hat{\beta}_i \cdot WOEVar_{i,j} \right)$

Rewriting:

$$Score_{i,j} = \frac{Offset}{n} + \frac{Factor}{ln(2)} \cdot \left(\frac{\beta_0}{n} - \beta_i \cdot WOEVar_{i,j} \right)$$

Where:

$Po = Increase\ in\ the\ score \ \ i = Variable \ \ j = Bucket\ of\ each\ variable\ i \ \ n = Number\ of\ variables$
$Offset = value\ imposed\ by\ the\ entity \ \ Factor = value\ imposed\ by\ the\ entity$

Example

I am going to calculate the score of the variable (i) WOEAGE of the bucket (j) 18-22. I suppose we have an offset of 500 and a factor of 20 (I will explain later in this thesis from where these values come from). Moreover, suppose we already have done the logistic regression, and we have got a β_0 = -1.73, and a B_1 = -0.654, which corresponds to the WOEAGE variable. Also, notice that the

WOE from this bucket (18-22) of this variable (WOEAGE) is -107.83 %. Finally, supose we have n = 2 variables in the logistic model.

With all this information, the score would be:

$$Score_{i,j} = \frac{500}{2} + \frac{20}{ln(2)} \cdot \left(\frac{-1.73}{2} - (-0.654 \cdot (-1.0783)) \right)$$

$$Score_{i,j} = 204.69 \approx 205$$

To better explain the score calculation I am going to base myself on the thesis *Crédito al consumo: La estadística aplicada a un problema de riesgo crediticio.* (Nieto 2010).

The values of the scorecard are calculated by doing a transformation of the parameters obtained in the logistic regression model. This transformation is done to obtain integer values for each bucket of each variable. It is a good practice to calibrate the scorecard in such a way that, for some increase in the score (Po), it doubles the good/bad ratio (Odds).

Therefore, the scores of a scorecard depend on the parameters of rescalation (factor); and translation (offset), so that the different scores can be compared. These parameters are decided by the entity. When we define the offset and the rescalation factor, what we are doing is defining an output maximum value (offset) and some decreasing values for the slope that we define (factor) until we reach to cover all the range of odds. That is, if we define a relatively low score offset and an exaggerated slope, what will happen is that we will obtain negative values in our scorecard.

The first step is to define the top of our scorecard. Lets say we establish that for an odds of 1:1 we will have a score of 500 (score = 500), this means that, if we have a score of 500, the ratio of good and bad is 1:1. This is a 500 offset.

The second step is to define a slope in the calculation. That is how many points are necessary to double the proportion of good / bad (odds). Suppose we decide that the odds are doubled every 13.86 points (Po = 13.86). This means that the proportion of good / bad (odds) should double every 13.86 points of the scorecard. So, 513.86 points are an odds of 2:1. From here we get a factor of 20 (13.86/ln(2) = 20). So, as the reader may notice, the factor and offset parameters are decided by the entity or analyst, so they are not fixed. Each entity will choose different factor and offset parameters depending on how they want their scorecard to look like.

Once we have all this information, we get the following scorecard:

Table 5: Scorecard Example

Variable	Buckets	WOE	Score	Beta
WOEAGE	18-22	-107.83 %	205	-0.654
WOEAGE	23-26	-71.47 %	212	-0.654
WOEAGE	27-29	-3.38 %	224	-0.654
WOEAGE	30-35	71.34 %	239	-0.654
WOEAGE	36-43	119.71 %	248	-0.654
WOEAGE	44+	166.08 %	256	-0.654
WOESALARY	0-9000	-23.76 %	230	-0.706
WOESALARY	10.000-20.000	-45.3 %	234	-0.706
WOESALARY	21.000+	103.5 %	246	-0.706

Source: Own Elaboration

The values of this score are very high because I am working only with two variables. You may also notice that the higher the WOE, the higher the score. This makes sense, since I have defined the WOE as the logarithmic transformation between a non default event and a default event. The higher the score, the greater the probability the loan will be granted to that client.

Finally, the model would be validated by studying its errors, its predictive power index, the Kolmogorov-Smirnov distance and the ROC curve, among other statistics. As it is not the main object of this thesis, I will not go deeper into these concepts.

This thesis aims to study the probability of default (PD) for the valuation in the granting of credit through advanced techniques of prediction algorithms. The objective is to implement Neural Network techniques to improve the PD prediction. As explained in the previous sections, risk management is one of the most important aspects of an entity. Therefore, and following this line, the less error an entity's PD predictions have, the better the entity will be able to control and manage its risks and, consequently, have a more adequate image of reality. As an introduction to the Deep Learning world, the following section explains some of the Machine Learning techniques most used in economics.

2.2 Machine Learning in Economics

This point aims to explain two main concepts: the importance of the AI in Economics, and an alternative way, through Machine Learning, to create Credit Risk Models to predict default.

Machine Learning (ML) techniques have an incredible impact on many sectors today. One of them is the economic field. Unlike the traditional field of economics, in which the researcher chooses a model based on mathematical principles and estimates it, ML techniques see the empirical analysis as an algorithms that estimate and compare models. Moreover, ML algorithms regenerate themselves, that is, these techniques learn from data. Therefore, with new data available, they improve their analysis. ML techniques improve analysis, performance and therefore optimizes processes. The combination of these techniques with the increasing availability of data is leading to a transformation of the global economic sector.

ML is a discipline within the field of AI that aims to create algorithms or systems that learn from themselves, i.e. automatically. A view of ML techniques more oriented to the field of economics would be that of a field that develops algorithms to be applied to databases in order to make

decisions in a given situation. It focuses mainly on fields such as prediction, through regression or classification.

There are many Machine Learning techniques that can be applied to the economy. The next point aims to explain the importance that these techniques are currently having in the economic field.

2.2.1 Importance of AI in Economics

This point it is intended to give greater emphasis to the role played in the economy by the current growing implementation of Artificial Intelligence techniques. I will talk about the contributions that Artificial Intelligence techniques have in the economy and, more specifically, I will talk about the repercussions that they have in the labor market.

2.2.1.1 AI and labor force

Artificial Intelligence (AI) techniques have an enormous potential to increase productivity growth in an economy. However, as a drawback, AI can lead to problems with the labor market, especially in the short term.

The main problem with AI applications is that they can give better results than humans and, therefore, they might end up replacing them. For example, error rates for image recognition are so low that they have even exceeded human performance. Therefore, these applications end up being a threat to the labor market; since these automations would kill Jobs and generate irreversible damage to the labor market (Furman and Seamans 2019).

On the other hand, there is clear evidence that automation techniques accelerate a country's productivity growth. For example, the article *Explaining a Productive Decade* by Oliner (Oliner, Sichel, and Stiroh 2007) shows that better management of IT explains part of the difference in productivity between US and UK firms.

The following graph shows the number of patents authorized in the US, as a concept of AI. This growing trend shows the importance that these techniques are currently having and their application in the economy. In other words, the number of patents ends up being a reflection of the number of innovations that are intended to be implemented in the market, in order to make a profit. These innovations tend to replace tasks that currently are done by human and, therefore, end up affecting the labor market.

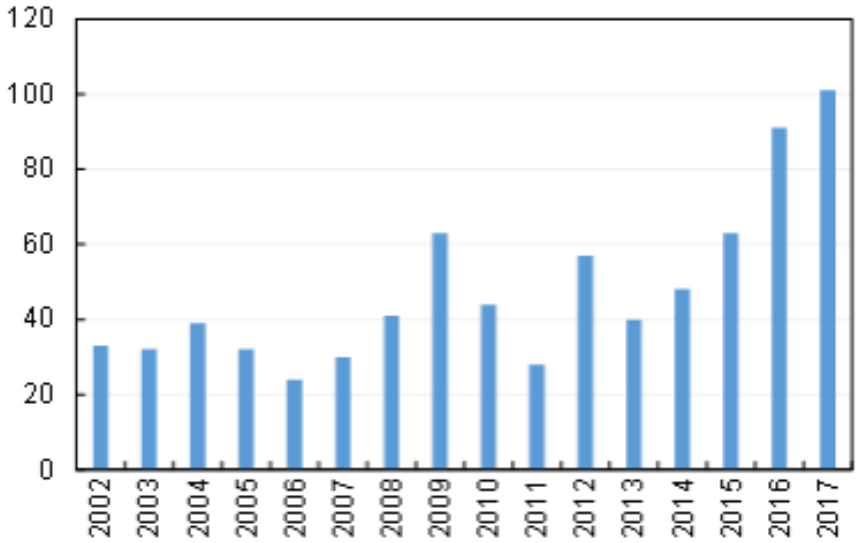


Figure 5: U.S. Patent Applications for AI. Source: United States Patent and Trademark office

Process automation therefore leads to increases in productivity and growth of economies, and to changes in the labor market. One of the most relevant concerns within these changes is that they occur so quickly that workers are unable to adapt and learn new techniques; so there end up being periods of time when long segments of the population do not work (Furman and Seamans 2019).

According with the article *AI and Economy* by Furman (Furman and Seamans 2019), there are three different perspectives that can offer insight into the effects of AI on labor market: a theoretical perspective, an empirical/historical perspective, and attempts to make granular predictions about nascent technologies. Following the argumentation of the article, these three perspectives show that automation techniques will not drastically displace the work force, but that they will have worrying effects on it.

Theorical perspective

From a theoretical perspective, AI and automation techniques have four effects on the labor market:

- Automation can directly displace labor in the affected sector.
- Job creation in new areas. In most business innovations, new jobs are created. In addition, in some cases, the construction of jobs is greater to the destruction of these.
- Technological innovation is often followed by wage increases in the sector. An increase in wages leads to an increase in the economy's demand for labor. In other words, as workers have more monetary resources, consumption increases and, therefore, jobs are created in other sectors, such as leisure.
- Technological innovations sometimes replace specific tasks, rather than entire jobs themselves. That is to say, the worker has new and better tools to carry out the same task. The problem could come from that because of these better tools not so many workers are needed anymore.

Studies by other authors confirm these four effects that AI and automation techniques have on the job market. For example, as Furman's article states (Furman and Seamans 2019); the authors Dauth, Findeisen, Sudekum, and Wobner did a study in which they combined German labor market

data with IFR robot shipment data. They found out that while each additional industrial robot leads to the loss of two manufacturing jobs, enough new jobs are created in the service industry to offset and in some cases over-compensate for the negative employment effect in manufacturing (Dauth et al. 2017).

Empirical perspective

From an empirical perspective, the authors define two types of evidence of the impact of technology on the labor market: cross-sectional y time series.

The cross-sectional evidence states that there is no correlation between the level of productivity and the rate of employment in an economy. The following chart prepared by the organization for economic co-operation and development confirms such evidence.

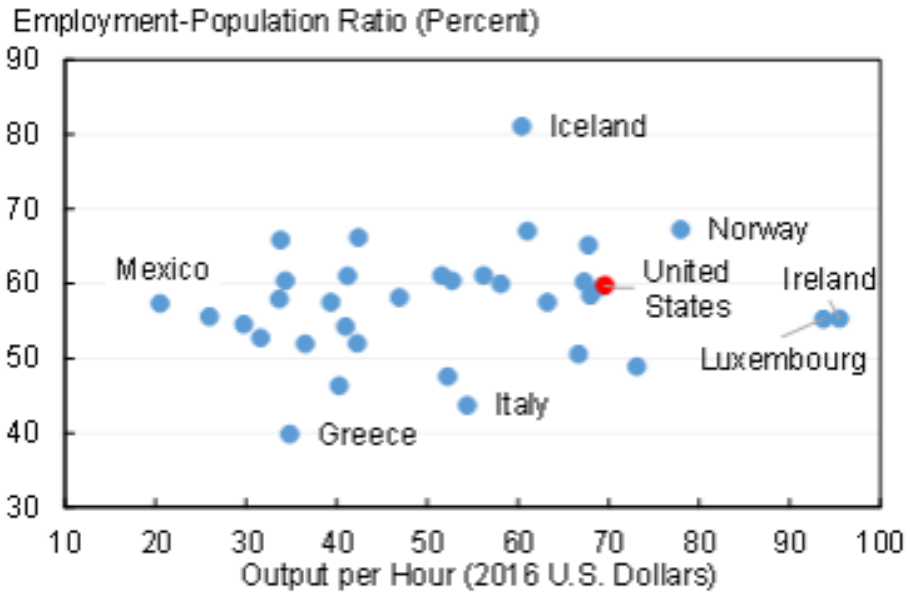


Figure 6: Employment - Population Ratio vs. Labor Productivity in the OECD, 2016. Source: Organization for Economic Co-operation and Development

In the previous graph it can be seen how, for example, Ireland produces much more output per hour than Mexico, but this does not mean that there are differences in hours per person between these two countries, but that Ireland produces more output than Mexico. That is, it is more productive. But it does not mean that it has a low rate of employment.

Granular predictions about nascent technologies

From a more granular perspective, an OECD report used data to predict the impact of IA techniques on the labor market. The results were that 9% of jobs in the USA, and the OECD countries, will be very susceptible to automation.

Following this more granular approach, it can be said that there is an important correlation between the skills that are more likely to be automated, and the level of income or education received by the worker.

The following two graphs show how jobs with lower pay are more likely to be automated. In addition,

it also shows how the level of education is distributed with the areas that have the most odds of being automated.

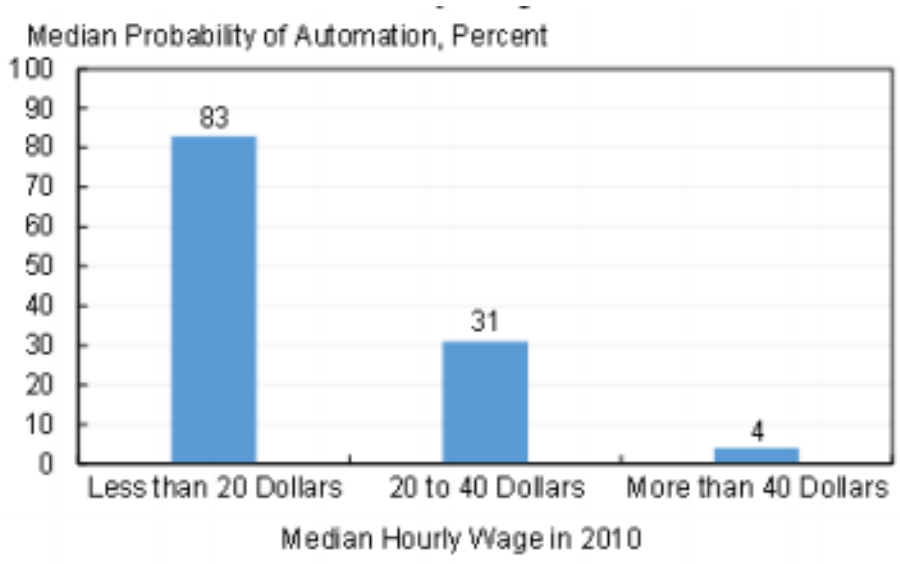


Figure 7: Probability of Automation by an Ocupation´s Median Hourly Wage. Source: Council of Economic Advisers (2016)

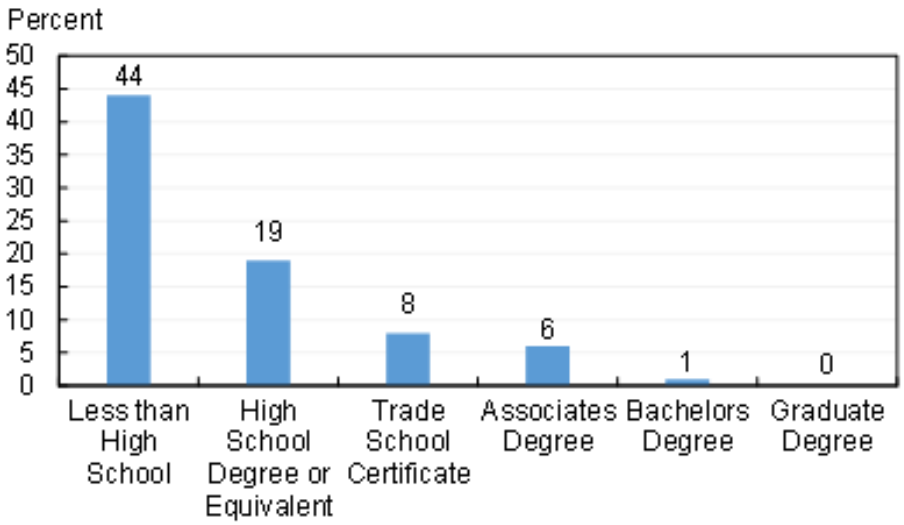


Figure 8: Share of Jobs with Highly Automatable Skills by Education. Source: Amtz, Gregory, and Zierahn (2016) calculations based on the Survey of Adult Skills (PIAAC) 2012

As a summary of the above, Artificial Intelligence has the power to drastically change the economy of any country. On the one hand, by greatly increasing productivity growth and, on the other hand, by unbalancing the labor market. These two statements are confirmed by recent studies on the subject, which have been cited throughout the point. The following points focus on an explanation of Machine Learning techniques. The idea is to introduce the basic concepts, and then make a good practical implementation.

Before starting, there is a relational schema of the steps in creating a model in ML:

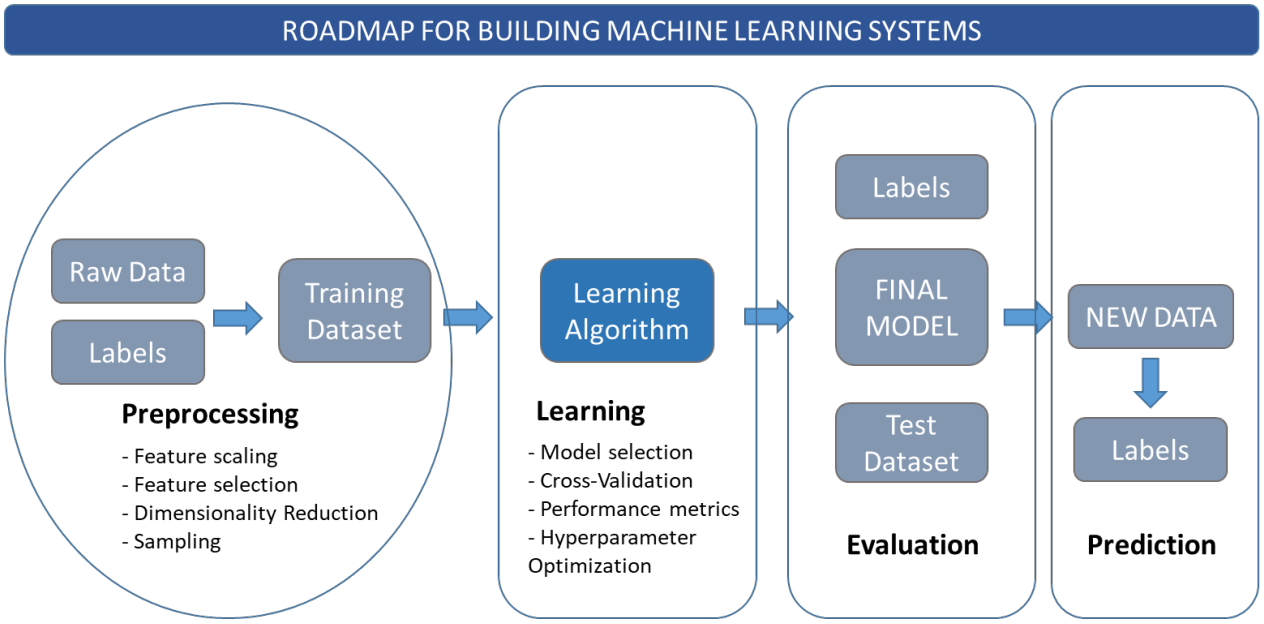


Figure 9: Relational Schema ML. Source: Own elaboration (based on the book Python Machine Learning)

2.2.2 Basic Concepts

- **Preprocessing:** This is one of the most crucial steps in any ML algorithm application. It is about treating the obtained data in order to adapt them to the problem you are trying to solve. Therefore, in many cases it will be necessary to change the scale of the variables and/or reduce the size of the data in order to obtain a lower computational cost and to solve possible correlation problems between variables, as well as to dispense with those variables that do not contribute anything to the model and only introduce noise.
- **Training Set:** Defined as the set of data used to create a predictive model. In other words, the model is created from this data set. The idea is that the parameters of the model are adapted to this data set trying to minimize the error function chosen. Usually 60% of the data is dedicated to this set.
- **Cross Validation Set:** Defined as the set of data used to fine-tune the hyperparameters of the model created in the training set and, in fact, choose the one with the least error. If, for example, in a logistic regression we have models with different polynomial degrees, this step aims to find that polynomial degree with the least error. In the field of neural networks, hyperparameters are, for example, the number of hidden layers (discussed in point 3). It is therefore a question of using data that have not been used in the creation of the model with the training set in order to find the model with the least error. However, this can lead to overfitting the model, and for that, we use the test set. Usually 20% of the data is dedicated to this set.
- **Test Set:** Defined as the set of data chosen to check how good a predictor the model created from the training is. At this point, the precision, sensitivity and specificity of the model

are calculated. This step is very important for the detection of overfitting and, therefore, to regularize the parameters (discussed in point 3). 20% of the data is usually dedicated to this set.

- **Supervised Methods:** These are defined as a ML techniques that consist of deducing a function from training data (training set) and response data (test set). The main objective is to predict the value of a new entry through this created function. In this field two different approaches are distinguished: the regression problem approach, where the answer variable will be numerical; and a classification problem approach, where the answer will be a label.
- **Unsupervised Methods:** These are defined as a ML techniques that consist of deducing a model from some data. Unlike supervised methods, there is no knowledge of the answers to these data. Therefore, unsupervised methods treat data as a random variable. One of the most relevant applications of unsupervised methods is its application in clustering. In this field, the idea is to organize a set of data into meaningful groups. It is also quite common to apply it in dimension reduction problems and in Principal Component Analysis (PCA).
- **Reinforcement Learning:** The objective in this type of ML is to develop a system (agent) that improves its performance through interactions with the outside. That is, they are goal-oriented algorithms that learn to reach a complex objective through an interaction process; that is, which actions must be chosen in order to reach the objective, which can be to maximize or minimize a model.

2.2.3 Logistic Regression: Binary Classification

As I mentioned before, logistic regression is currently the method most used in the calculation of the Probability of Default (PD) and, therefore, for the construction of a portfolio scorecard.

Following this line, logistic regression is a model that, by assigning certain weights to independent explanatory variables, aims to obtain the most accurate prediction possible of the dependent or endogenous variable, which, in our case, is default.

Lets call Y the dependent variable that reflects whether or not an operation is default.

$Y = 1$, the event occurs (default); $Y = 0$, the event does not occur (non default).

We have a sequence of variables $X_1, ..., X_p$ and we want to study the influence of these variables in the Y event. This influence can be modelled using the following probability:

$$P(Y = 1|X_1, ..., X_p)$$

Each of these variables is weighted by a weight, which will be the determinant of its influence on the variable Y.

Logistic regression will give us a probability between 0 and 1. Based on this result, a threshold or decision threshold is defined. And so, we will classify each probability into 1 or 0.

The classification algorithms are a subcategory of the supervised ML algorithms. The objective is to predict the categorical labels of new data. The functioning of these algorithms is as follows: many examples (training set) are received with their respective label. These are sorted and trained (through changes in coefficients and parameters) so that when new inputs are introduced without a label, the algorithm tells us which label it belongs to. This classification can be made for binary data

(for example, if a credit has been paid, or not paid); or, it can also be implemented for multi-class data (for example, classify number images).

Actually, prediction by classification is currently the central block of ML techniques. Some profesionals says that once the classification structure is understood, it becomes less complex to implement other techniques more complex, such as ranking, anomaly detection, etc.

The following plot represents the concept of **binary classification:** We assume that we have two groups (0 and 1), which represent whether an operation has been paid or if, on the contrary, it has not been paid. x1 and x2 are two variables, which represent characteristics of the client (AGE and SALARY, for example).

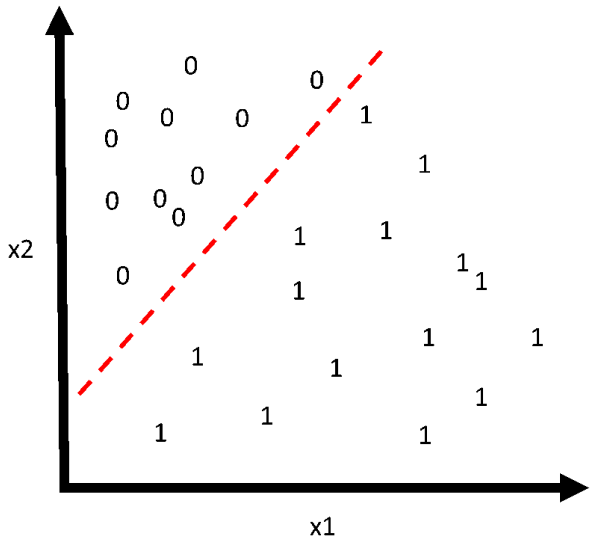


Figure 10: Binary Classification. Source: Own elaboration

2.2.4 Neural Networks: Binary Classification

Neural networks have become one of the most popular ML structures in the last years. However, the neural network concept has existed since the last century. Thanks to the advancement of technologies and the large amount of data currently available, neural networks have begun to be implemented in a more serious way, and have resulted in practical applications that optimize processes and improve results. Applications such as voice recognition, image recognition or text recognition, make possible the creation of autonomous cars, virtual assistants, more accurate translators, fraud prediction, genetic analysis, and so on (Rouse 2016). Neural networks play an indispensable role in these applications.

Therefore, neural networks are a set of very powerful algorithms with which we can model independent processes. A neural network is formed by the union of several simple parts, called neurons or nodes. As a very simple definition, a neuron is the most basic unit of processing that we are going to find in a neural network. It has input connections through which it receives external inputs (input values), which perform an internal calculation with these inputs to generate an output value.

Following the concept of linear regression (see 2.2.1), where, given some inputs, some weights were assigned to the parameters of the model (minimizing the error); a neuron acts in a very similar way.

The neural network receives some inputs and through linear regression transforms them into logit.

As it may be seen in the figure below, a neuron uses all the inputs that we introduce, and assign a weight to each of these inputs. Through some calculations, it ends up displaying an output. These calculations will be explained with more detail in point three of the present thesis.

In addition to the inputs, another connection is added to the neuron to control bias, this is the term "b". This variable "b" is always assigned to 1 and can be manipulated to change the results in an activation function, which I will discuss later in this thesis.

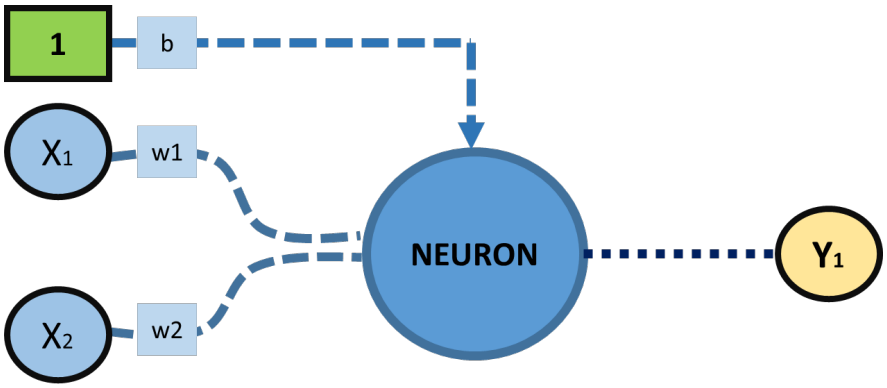


Figure 11: Simple Neural Network. Source: Own elaboration

As I said, the neuron does a linear regression calculation. Therefore, the answer will be a continuous variable and not a binary one, which is the one we are looking for. To solve this problem, the concept of **threshold or step activation function** is introduced.

The threshold is a system of making the continuous variable binary. It is to pass our continuous results through an activation function that converts them to 1 or 0 depending on the threshold we define.

That is to say, given a threshold previously defined as 0, the staggered activation function would be:

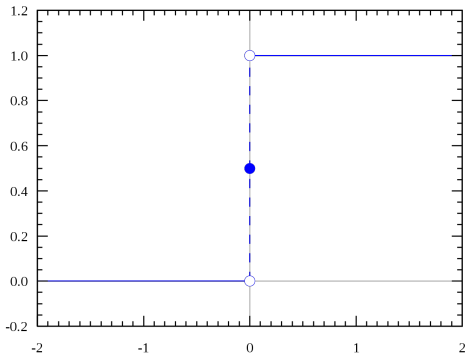


Figure 12: Step Activation Function. Source: Medium

$$\begin{cases} \text{W X} < \text{threshold} \rightarrow \text{Y} = 0 \\ \text{W X} > \text{threshold} \rightarrow \text{Y} = 1 \end{cases} \tag{2}$$

Rewriting the equation:

$$\begin{cases} \text{W X} + \text{b} < 0 \rightarrow \text{Y} = 0 \\ \text{W X - b} > 0 \rightarrow \text{Y} = 1 \end{cases} \tag{3}$$

In this way, we can finally solve a binary classification problem.

To better understand the concept of a neural network, I will introduce a continuation of the example I started in the previous chapter. We suppose that we have two variables AGE and SALARY; for now, to make it simpler, we will assume that they are not divided into buckets. This time, we want to predict whether or not a customer will default on a credit.

In this line, we have a simple two-variable model:

$X1$ = Age of the individual $X2$ = Annual salary of the individual Y = Default / no Default (1 or 0)

First, let's suppose we plot these data:

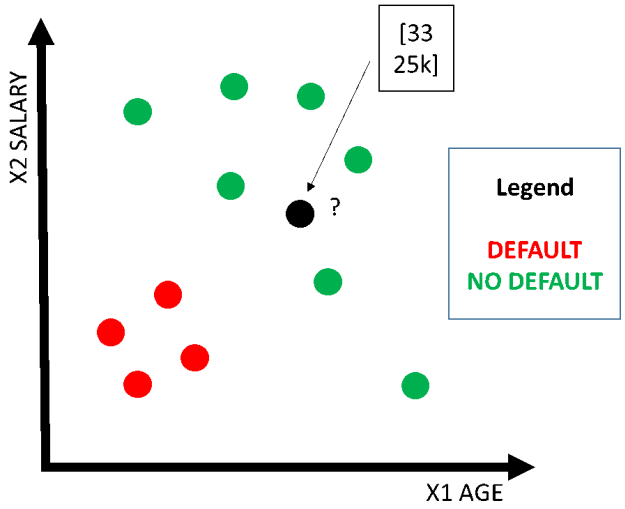


Figure 13: Example representation. Source: Own Elaboration

What we want to know is whether or not the marked individual will default. For this purpose we create a neural network. The first step -after preprocessing- is to train the network. The training, as defined at the beginning, is to fine-tune the parameters and reduce the cost function and get the prediction right with a higher probability (Malik, Farhad 2018). In this training step, the model will also apply a backpropagation algorithm to minimize the errors of the model. This procedure is explained in more detail in the next point three. The objective of the example was to make clear what i was solving.

Following the line of the previous example, we see that the set of points was separable linearly, however, there are problems that can not be separated linearly. At this point the structures AND, OR, XOR appear.

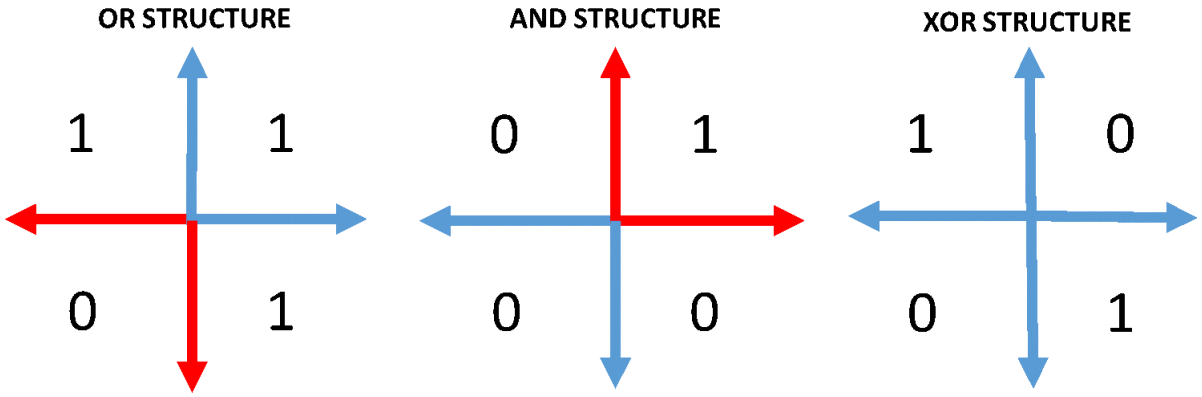


Figure 14: XOR Problem. Source: Own Elaboration

We see that the type of XOR structure problems have no solution if we use only one neuron. Minsky and Rapert showed that this was a big problem for the Neural Network architecture of the 1960 (Minsky and Papert 1969). This problem was the main reason why the investment in neural network projects decreased a lot in the 70s and 80s. Simple neural networks (perceptron simple), were not capable of solving non-linearly separable problems. In 1986, Rumelhart and McClelland designed the **multilayer perceptron (MLP, Multilayer perceptron)** allowing to solve problems that are not separable linearly (Rumelhart, Hinton, and Williams 1985). This Multilayer Perceptron will be discussed further in section 3 of this thesis.

Therefore, the solution to nonlinear problems is adding another neuron. With two neurons we would have two straight lines that can be used to separate the groups. This is an example that shows how the combination of several neurons allows us to solve more complex problems. The union of two or more neurons is what is known as a neural network.

When we place the neurons sequentially, one of them receives the information processed by the previous neuron. This concept is that of hierarchical knowledge. This means that each neuron will process a different type of information and then join together to gather the information and make the output decision (Mavrovouniotis and Chang 1992).

The more layers we add to the network, the more complex the knowledge we obtain can be. This complexity of layers is what is called Deep Learning, which I will talk about more deeply in the next point 3.

3 Deep Learning

3.1 Rol Deep Learning

The first concept I want to make clear is the difference between Machine Learning (ML) and Deep Learning (DL) techniques, because they are usually confused. In fact, the DL is a particular case of the ML. This means that all Deep Learning techniques are also Machine Learning, but not vice versa. In this line, Deep Learning arises from the complexity of some non-linear problems. It was proved (Rumelhart, Hinton, and Williams 1985) that the implementation of several neuronal layers in a neuronal network structure could solve very complex problems, such as the recognition of images, the translation of documents or the decomposition of sounds by tonalities, among others. The architecture of DL algorithms is based on each layer receiving information from the previous layer, implementing the knowledge it contains and passing it to the next layer. In this way, each layer brings knowledge about the output and, that is how the algorithm learns.

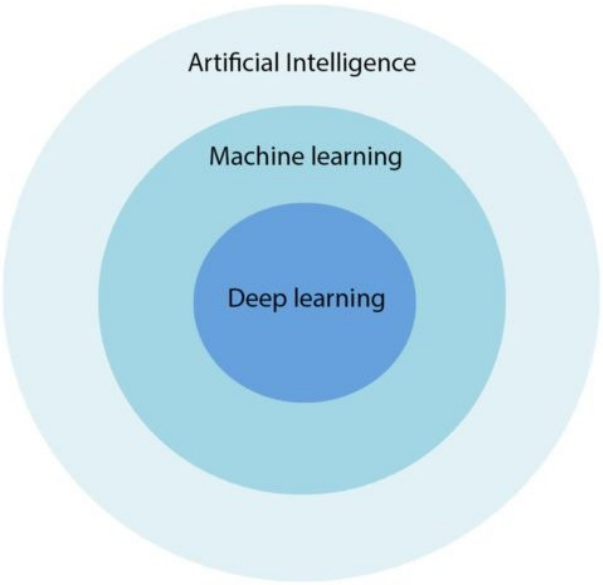


Figure 15: AI, ML, and DL. Source: Web Design Ledger

Example of how Deep Learning neural networks work

As I have defined, Deep Learning carries out the Machine Learning process using neural networks with different hierarchical levels. The learning process is as follows: At the initial level (or first node) the network learns a simple feature and sends the information to the next level. The second level collects the information sent to it by the first level and combines it making the information a little more complex.

Lets give an example so that it can be better interpreted: Suppose we want to identify the sound of a piano in an orchestra. The initial level of a Deep learning network could use the frequencies of sound to identify what is background noise and what comes from the orchestra. The initial level passes this information to the second level, which introduces new information; for example, differences in tonality and passes the information to the third level, and so on until the network learns to identify the sound of a piano in an orchestra.

Therefore, the central idea of DL is that we assume that data is generated by the composition of factors or characteristics, potentially at multiple levels of a hierarchy. Many other similar generic assumptions can further enhance deep learning algorithms.

At this point I will present the main approaches to DL. It should be noted that each approach has its strengths and weaknesses. One approach or the other will be used depending on the scope, context and data used in addressing a problem.

Why now?

The foundations of deep neural network algorithms have been uncovered for decades. So the question is: Why do they start to be applied and improved now? There are three essential points that answer this question:

1. **Big Data**, fast and efficient ways of processing and accessing large amounts of data have been discovered.
2. Better and faster **hardware**, graphics and processors (GPUs).
3. **Software**, improved techniques, new models and tools to deal with them (Tensorflow, for example).

Therefore, these three characteristics have made Deep Learning techniques resurface and give results applicable and scalable to large companies, adding value.

3.2 Multilayer Perceptron (MLP)

I would like to return to the concepts defined in the section on neural networks in ML (see 2.2.2). The big problem with XOR structures was that we couldn't define groups with only one node. However, by adding two nodes, we could solve them, since we introduced a new line in the network. This multilayer concept in a neural network is called Multilayer Perceptron (MLP).

Architecture

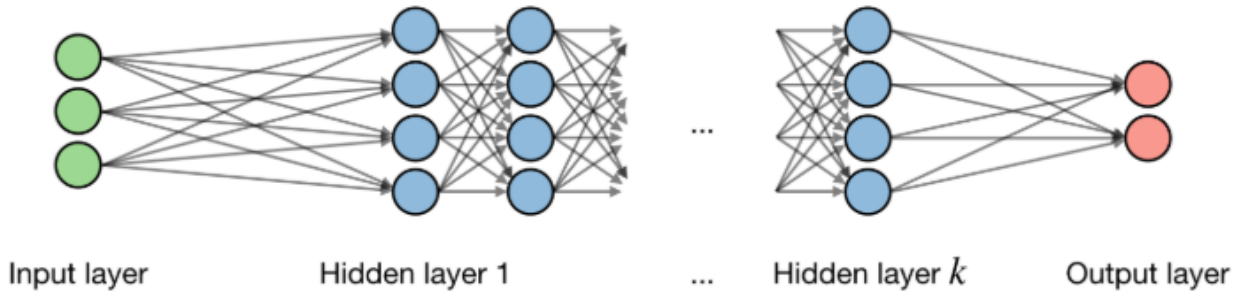


Figure 16: MLP. Source: Deep Learning cheatsheet by Afshine Amidi

By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]^T} \cdot x + b_j^{[i]}$$

where noting w,b,z the weight, bias and output respectively.

The vocabulary on neural network architecture is described below:

- **Input Layer** is the initial layer of the neural network. It is the one that receives the initial data with which we will work and learn the network.
- **Hidden layer** is in the middle of the neural network. They perform calculations by aggregating and gathering information about the defined problem. They are called "hidden" because it represents that the engineer does not design the nodes directly himself.
- **Output layer** it is the layer that displays the required information. That is, if we are facing a binary classification problem, this layer would contain two nodes [1 and 0] and would choose one of the two according to the information learned and applied by the neural network.

3.3 Activation Functions

By sequentially adding nodes we build a series of layers that, the more they are added, the more complex the problems may be to be solved. The whole of this series of layers is called Deep Learning. Therefore, in short, what we are doing is concatenating different operations of linear regression. The problem is that we mathematically verify that the result of adding many linear regression operations (i.e. straight lines) is equivalent to having done a single operation. That is, the result is another straight line. This means that, of all the number of neurons we wanted to build, we are left with only one.

The solution to this problem is that each of the linear regressions of each neuron suffers a non-linear manipulation that distorts them. This is the concept of an activation function. Therefore, the linear regression that we calculate in each neuron it is passed through the activation function, so the activation function will transform them into some values. These values depend on the kind of activation function we are using. Later in this point, I will describe some of the most important activation functions.

An activation function is simply the transformation of the output value to solve non-linear problems. Therefore, the effect of the activation functions is to distort the planes of a neuron.

Activation functions are therefore necessary to deal with non-linear problems. In other words, if we do not implement an activation function, we will obtain linear outputs and, therefore, all those XOR problems would have no solution. The activation function may also be seen as a decision tree when using the if-else procedure. In general, the activation function is applied to the sum of input products (X) and their corresponding Weights (W) to get the output from that layer and feed it as input to the next layer.

One of the main features of the activation functions is that they have to be differentiable. This is because the neural network is just passing an optimization function called Backpropagation. Although I will explain this technique in more detail later in this thesis, what does this function is to propagate backwards in order to calculate the error gradients with respect to the weights. This is interesting because it will allow to optimize the network weights with any optimization technique (like for example the gradient descent) and, in this way, to reduce its error.

There are many activation functions and each one has its advantages and disadvantages depending on the problem we are facing. It can be said that the activation functions of the hidden layers will be different than those of the output layer. This is because in each layer we look for different objectives. For this reason each activation function will be more appropriate depending on the type of layer and problem we face. In this thesis I will focus on the activation functions Sigmoid, Tanh

and ReLU, with their improvements. At the end of the point I will present a summary table of each of them.

3.3.1 Sigmoid

$$F(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Graphically represented as:

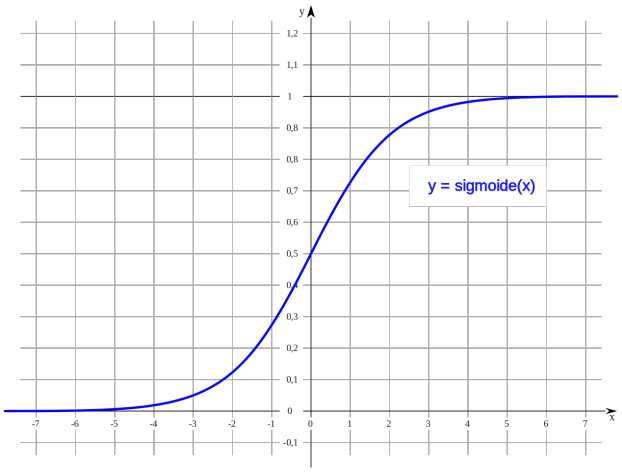


Figure 17: Sigmoide Function

Its range of values goes from 0 to 1. It is simple to understand and implement, but it has a series of disadvantages that makes it no longer a good strategy to implement in hidden layers. In this line, the main disadvantages of this function are:

1. **Vanishing gradient problem.** This is the main problem.

What is the problem with Vanishing Gradient?

The Vanishing Gradient Problem appears when we try to train a Deep Learning model using Gradient Descent using Backpropagation. It was found by Sepp Hochreiter in 1991 (Hochreiter 1998)

The problem arises when applying the Backpropagation technique. That is to say, when the error gradients are calculated with respect to the weights of each neuron. In this context, the aim is to find the local minimum of the error function and subtract it from its respective weight matrix so that these values give a more precise result. During this Backpropagation procedure, the magnitude of the gradients becomes exponentially smaller. If the gradient is very small, it no longer makes sense to subtract it from its original matrix. This causes the first neurons in the network to "learn" very slowly. And these first neurons are, in fact, the most important. The importance of the first layers of the network lies in that they are the ones that have to deal with the most basic information of the problem we are dealing with, therefore, if they are poorly trained, the model will have serious prediction flaws. When applying the sigmoid function in hidden layers, those derived from all hidden layers multiply each other, which means that the gradient decreases exponentially as we spread to the initial layers. In other words, the gradient becomes so small that it is not possible to train the first layers effectively (Singh Walia, Anish 2017).

To explain the mathematical approach to the Vanishing Gradient Problem I am going to base myself on the article written by Manik Soni: "Exploding And Vanishing Gradient Problem: Math Behind the Truth" (Soni, Manik 2017).

Consider a neural network with 4 hidden layers as the following:

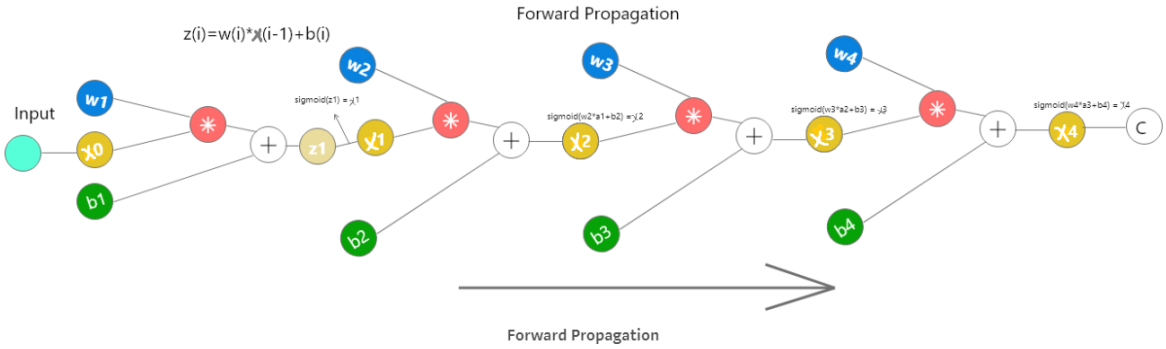


Figure 18: Forward Propagation. Source: Hackernoon

Where:
w = weight matrix
b = bias
z = w(i)*x(i-1) + b(i)
x = sigmoid (z)
In Forward Propagation we multiply the inputs with the weight matrix and add the bias. Next, the activation function is applied (sigmoid, in this case).

Once the Forward Propagation algorithm is applied, the error has to be calculated, in order to improve the weights of the matrices and give a better precision. For this, the backpropagation algorithm is used (see section 3.5.3)

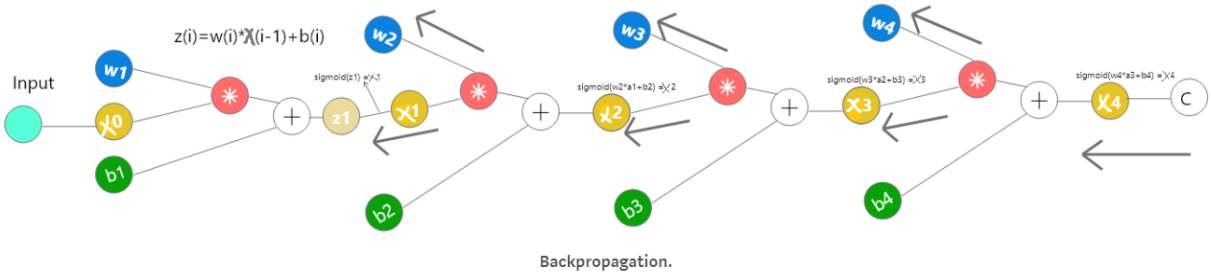


Figure 19: Backpropagation. Source: Hackernoon

In backpropagation what it does is to look for the derivative of the output with respect to the weights of the matrices. Suppose we want to find the derivative of C (cost function) with respect to the matrix of weights (b_1).

We would have this:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \cdot w_2 \cdot \sigma'(z_2) \cdot w_3 \cdot \sigma'(z_3) \cdot w_4 \cdot \sigma'(z_4) \cdot \frac{\partial C}{\partial x_4}$$

Figure 20: Derivative of the output w.r.t the weight (b1). Source: Hackernoon

Where σ' = Sigmoid function

Problem = $\sigma'(z_1)$, $\sigma'(z_2)$, $\sigma'(z_3)$, $\sigma'(z_4) < \frac{1}{4}$. This is because the maximum derivative of the sigmoid function is $\frac{1}{4}$

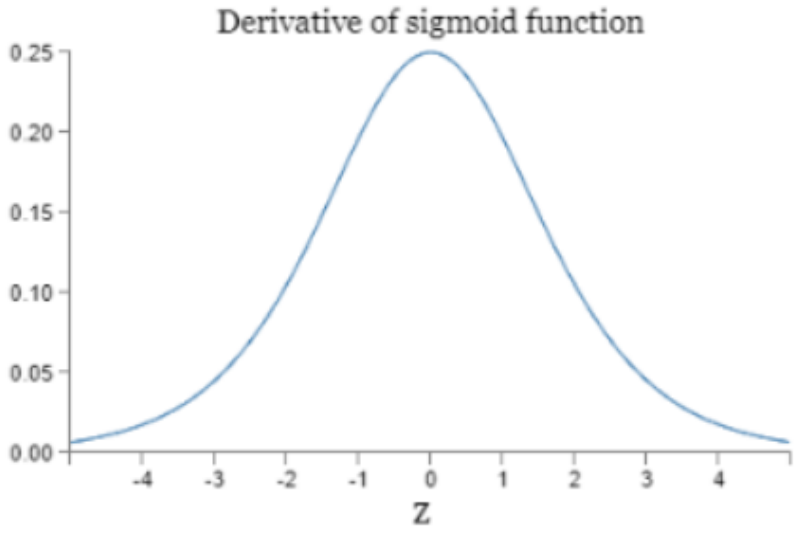


Figure 21: Derivative of sigmoid function. Source: Hackernoon

The weight matrices w_1, w_2, w_3 and w_4 are started using the Gaussian method with mean 0 and standard deviation 1. Therefore, $||w_i|| < 1$.

By putting these observation together, we observe that the terms $w_j\sigma'(z_j)$ will usually satisfy $|w_j \cdot \sigma'(z_j)| < \frac{1}{4}$. When we calculate the product of those terms, that product will tend exponentially to decrease: the more terms, the smaller the product will be. This causes the change in the weight matrices to be null; therefore, the first layers are not trained.

It is for these reasons that Sigmoid and Tanh functions are not usually used.

Later on in this point we will see what change occurs in the weight matrix and what role the derivative of the function plays. This explanation will reinforce the understanding of the problem of gradient descent.

This is a representation of the derivative of the sigmoid function (in red). We see how it tends to zero as it accumulates

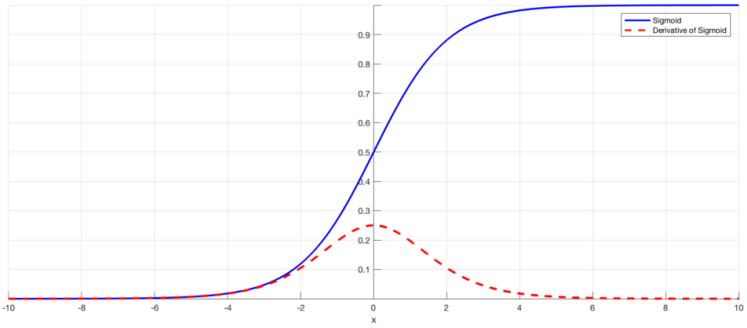


Figure 22: Vanishing Gradient Problem. Source: Towards Data Science

This problem can be solved with the batch normalization technique. However, it is more common to just use another activation function like ReLU.

Batch normalization normalizes the input so that |x| does not reach the outer edges of the sigmoid function. If we look at the image below, it normalizes the input so that most of it falls into the green region, where the derivative doesn't get too small.

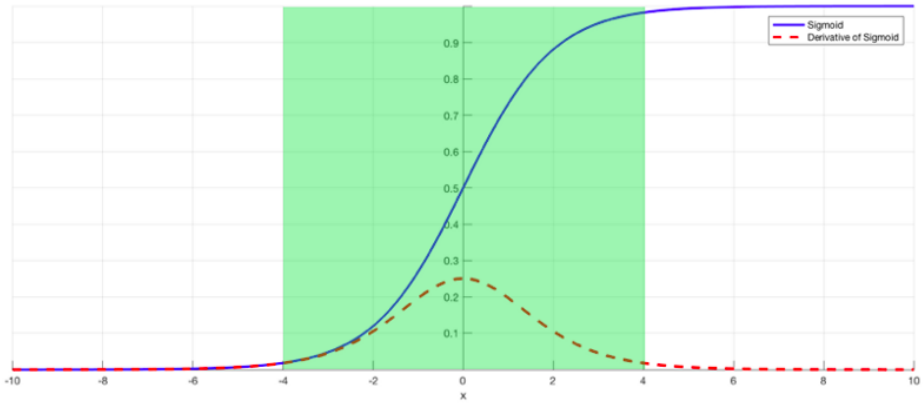


Figure 23: Vanishing Gradient Problem. Source: Towards Data Science

It also has other **secondary problems** such as:

2. The output is not centered on 0. This causes gradient updates to go too far in different directions. 0 < output< 1, and makes optimization more difficult.
3. They saturate and remove gradients.
4. Slow convergence

To solve these secondary problems the Hyperbolic Tangent function is introduced.

3.3.2 TANH

$$F(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Graphically represented as:

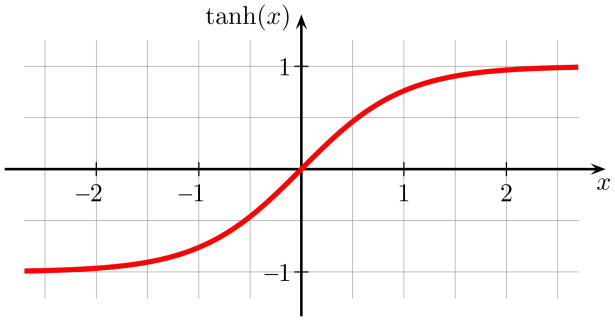


Figure 24: TANH function

This function solves the problem we had with the sigmoid function of the value centered on 0. With this function, the output is centered on 0, as the range of values goes from -1 to 1. However, with this implementation we still have the problem of the Vanishing gradient that I have already defined in the Sigmoid function. We can also solve it by means of the normalization by batches. However, another solution to solve the Vanishing gradient problem is to use of the ReLU function.

3.3.3 ReLU

$$\begin{cases} \text{f(x) = 0 if x<0} \\ \\ \text{f(x) = x if x 0} \end{cases} \tag{4}$$

Graphically represented as:

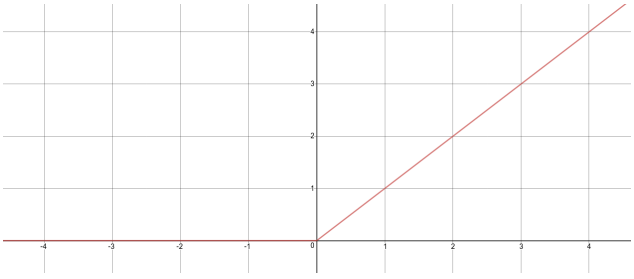


Figure 25: ReLU function

This feature has become very popular in the last two years.

The ReLU function is one of the best features discovered so far. It does not require floating point calculations of any kind. In fact, in addition to solving the vanishing gradient problem, it has been

shown (Krizhevsky, Sutskever, and Hinton 2012) that improves 6+ times the convergence speed compared to the Tanh function. In addition, as we see, the mathematical equation of this function is quite simple and effective.

The ReLU function solves the vanishing gradient problem because it does not have a small derivative.

I base myself on the article *Deep Sparse Rectifier Neural Networks* (Glorot, Bordes, and Bengio 2011) to really explain how the ReLU function works

The two main advantages of this activation function are:

1. Sparse activation: Means that, for example, in a randomly initializated network, only 50 % of the hidden units are activated.
2. Better gradient propagation: Solves the Vanishing Gradient Problem, because it does not have a small derivative.

Why is sparse activation an advantage?

1. If a network is sparse, at small changes in inputs, the set of non-zero characteristics is almost always preserved approximately.
2. If a network is sparse, it is more likely to be easier to separate it linearly; therefore, less computational cost.

In this line, a ReLU activation function allows a network to easily obtain sparse representations. To see this point more clearly I introduce the following representation. It can be seen that the only non-linearity in the whole network comes from the selection of the path associated with the individual neurons. That is, for a given input only a subset of neurons is activated. The computation in this subset is linear. As the article concludes, because of this linearity, gradients flow well in the active pathways of neurons. In other words, there is no effect of gradient disappearance due to the nonlinearity of activation, as was the case with sigmoid functions and tanh. Moreover, mathematical research is easier: there is no need to calculate the exponential function in activations.

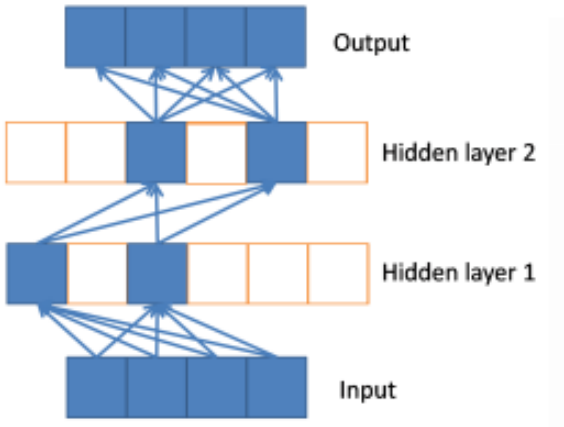


Figure 26: Deep Sparse Rectifier Neural Networks. Source: Glorot, Xavier

It must be said that too much dispersion can impair predictive performance by reducing the effective

capacity of the model.

However, the ReLU activation function has two drawbacks. The first is that it can only be applied to the hidden layers of a neural network, it cannot be applied to the output layer.

The second, and perhaps the main problem with the ReLu algorithm is that some neurons can die during training (i.e. it may end up being rendered unusable). This function can cause an update of the parameters that makes that a neuron is never activated again in any data point. In other words, the ReLu algorithm can cause neurons in the neural network to die and therefore be useless.

With the aim of solving this possible problem of neuron death, this function is introduced a small slope that makes the updates of the neurons do not die. This new implementation of the function is called Leaky ReLU, and this is generalizated by the Parametric ReLU, which is also explained.

3.3.4 Leaky ReLU

The Leaky ReLU is a variation of the ReLU activation function. It basically allows a small, positive gradient to be carried out on non-active units. Normally multiplies the input of the neuron, x, by 0.01.

$$\begin{cases} \text{f(x) = x if x>0} \\ \\ \text{f(x) = 0.01x otherwise} \end{cases} \tag{5}$$

3.3.5 Parametric ReLU (PReLU)

This is the variation that I will use in the practical part of this thesis. It is a generalization of the idea of the Leaky ReLU. That is, multiply the input by a constant a.

$$\begin{cases} \text{f(x) = x if x>0} \\ \\ \text{f(x) = ax otherwise} \end{cases} \tag{6}$$

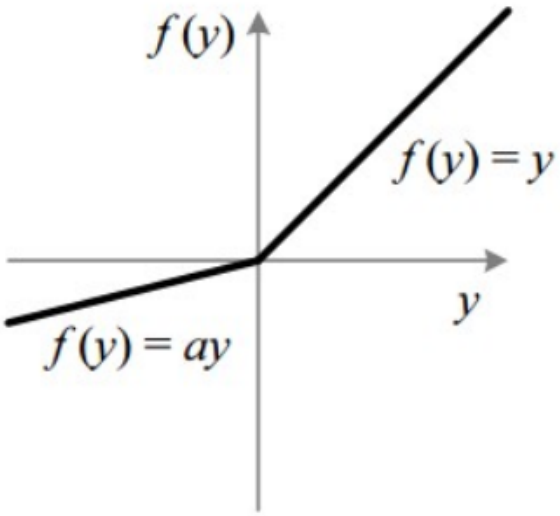


Figure 27: PReLU Function. Source: Towards Data Science

Finally, I would like to comment that there is also the ELU variation (Exponential Linear Unit), which tries to make the average activations closer to zero, in order to accelerate learning.

$$\begin{cases} \text{f(x) = x if x>0} \\ \\ \text{f(x) = a (exp (x)-1) otherwise} \end{cases} \tag{7}$$

In addition, there is also another type of activation function that is worth mentioning given its recent importance and implementation in many problems. It is a general version of the ReLU functions: The Maxout function.

Summary table of the main activation functions and their features:

In conclusion, normally the ReLU activation function is used for hidden layers, as it avoids the vanishing gradient problem. If we see that when applying the ReLU function, in the process of updating the parameters, neurons die, then we will apply the function PReLU or Maxout. However, for the output layer, if we are facing a classification problem (as is the case in this thesis) is better the sigmoid activation function, since, with this, we get the probabilities for each output given.

These activation functions make it possible to chain several neurons and create a neural network. For each neuron we can change its parameters and this will change the orientation of the activation function. The sum of several neurons with different parameters will create a non-linear structure that will allow us to create non-linear boundaries and therefore classify non-linear problems. That is, a number eight, for example. This is a very important advance within the classification problems.

Table 6: Summary table of the main activation functions and their features. Source: Own Elaboration

Activation Function	Pros	Cons
Sigmoide	Easy to implement Good for network output Output Layer	Vanishing gradient Output is not centered on 0 They saturate and eliminate gradients. Slow convergence
TANH	The output is centered on 0	Vanishing Gradient
ReLU	Eliminates the problem of Vanishing gradient Hidden Layers	Neurons may die in training Only usable in the hidden layer
PReLU	Eliminates problem of neuron death Allows the negative slope to be learned Hidden Layers	May perform differently for different problems

3.4 Loss Function ($J(\theta)$)

The loss function of our neural network basically tells the algorithm what it did wrong in predicting. That is, the loss function measures the incurred cost of incorrect predictions. It could be seen as the distance between the output we predict and the original output. Therefore, if the network predicts something that is very close to the original value, the loss function will be very small; otherwise we will get a big loss function. The objective of the neural network is to minimize the loss function in order to pr edict as best as possible.

The equation of the **loss function** is:

$$J(\theta) = \frac{1}{n} \cdot \sum_{i=1}^{n} L(f(x^{(i)}; \theta), y^{(i)})$$

Where:

$f(x^{(i)}; \theta)$ = predicted value $y^{(i)}$ = original value

Binary Cross Entropy Loss is the loss function for binary values. Cross-entropy loss increases as the predicted probability diverges from the actual label. In short, we are just multiplying the log of the actual predicted probability for the ground truth class. An important aspect of this is that cross entropy loss penalizes heavily the predictions that are confident but wrong (Parmar, Ravindra 2018).

$$J(\theta) = \frac{1}{n} \cdot \sum_{i=1}^{n} y^{(i)} \cdot log(f(x^{(i)}; \theta)) + (1 - y^{(i)}) \cdot log(1 - f(x^{(i)}; \theta))$$

Mean Squared Error Loss is the loss function for continuous values.

$$J(\theta) = \frac{1}{n} \cdot \sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}; \theta))^2$$

A summary table of the above would be:

Table 7: Summary table of the Loss Functions. Source: Deep Learning Book

Problem Type	Last-Layer activation	Loss Function
Binary Classification	Sigmoid	Binary_crossentropy
Multiclass, single-label classification	Softmax	Categorical_crossentropy
Multiclass, multilabel classification	Sigmoid	Binary_crossentropy
Regression to arbitrary values	None	MSE
Regression to values between 0 and 1	Sigmoid	Binary_crossentropy or MSE

3.4.1 Loss Optimization: How can we train a neural network by quantifying its loss?

As defined before, the loss is a function of the parameters of the neural network. Therefore, the objective of the neural network is to find those parameters that minimize the loss throughout the training process.

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(f(x^{(i)}; \theta), y^{(i)})$$

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ J(\theta)$$

Remember:

$$\theta = \{\theta^{(0)}, \theta^{(1)}, ...\}$$

The goal is to minimize the loss function.

Example

Suppose we have two parameters and what we want is to find their minimum. The following figure represents the distribution of the cost function of the two parameters (θ_0 and θ_1).

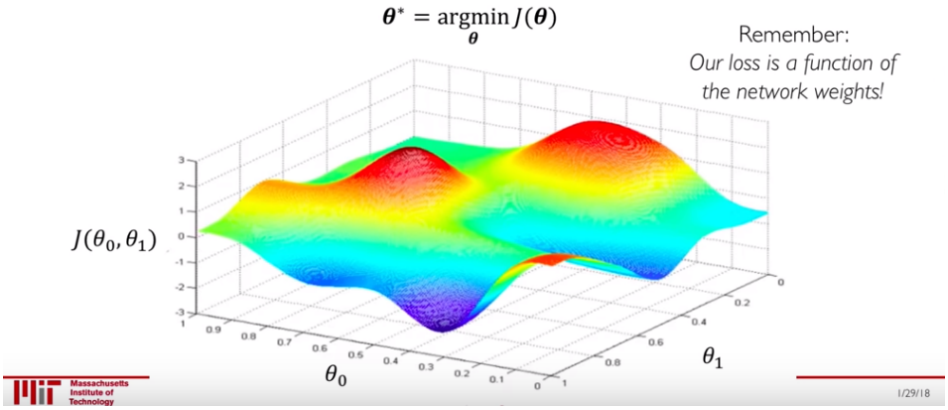


Figure 28: Loss Optimization. Source: MIT

If we find the minimum of the previous figure, we manage to find those parameters that will optimize the best predictions.

How do we minimize the cost function?

3.5 Fundamental algorithms DL

3.5.1 Gradient Descent

- Randomly pick an initial (θ_0 and θ_1).
- We compute the gradient $\frac{dJ\theta}{d\theta}$. This indicates how the loss changes with respect to each of the weights. This gradient gives us the direction of the largest ascent (not the smallest, which is what we would like, since we want to minimize the loss). Therefore, we take a step in the opposite direction of the gradient. For such a fact, we put the gradient in negative and adjust the parameters so that we take the step in the opposite direction.
- We proceed with the same technique until we converge to a local minimum.

Algorithm Gradient Descent

1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\theta)}{\partial(\theta)}$
4. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial(\theta)}$
5. Return weights

Another of the problems derived from applying neural networks to complex problems is the computational cost of calculating the gradient. One solution to this is the algorithm Stochastic Gradient Descent (SGD).

3.5.2 Stochastic Gradient Descent (SGD)

Algorithm Stochastic Gradient Descent

1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\theta)}{\partial(\theta)}$
5. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial(\theta)}$
6. Return weights

The idea of applying this algorithm is to compute the gradient using only one training example. This makes it very easy to calculate the gradient. However, being really stochastic creates a lot of noise; that is to say, it can make us take steps for the cost function that do not represent the gradient of our data and, therefore, are not useful to minimize the parameters.

Therefore, in order to improve this problem, mini-batches are introduced.

Algorithm Mini Batches

1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\theta)}{\partial(\theta)} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(\theta)}{\partial(\theta)}$
5. Update weights, $\theta \leftarrow \theta - \eta\frac{\partial J(\theta)}{\partial(\theta)}$
6. Return weights

This solution makes it faster to calculate than the descent gradient and also better estimates the gradients than the simple stochastic algorithm.

Advantages provided by Mini-batches:

1. More accurate estimation of gradient (smoother convergence and allows for larger learning rates)
2. Lead to fast training (can parallelize computation (i.e. we can divide the sample into batches and each group is trained by a different system, finally we put them all together) + achieve significant speed increases on GPUs))

3.5.3 Computing gradients: Backpropagation

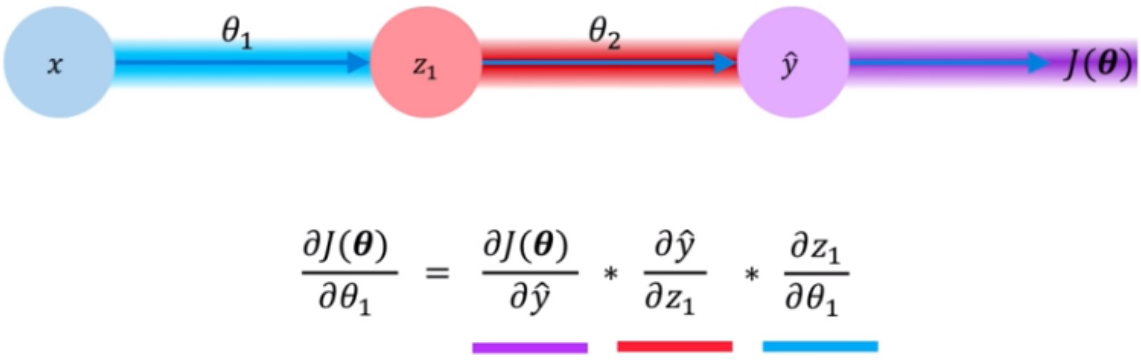


Figure 29: Backpropagation. Source: Youtube

The Backpropagation process is to repeat the above procedure for each network parameter using the gradients of the subsequent layers.

All of the above, as I have emphasized, is for a simple case. However, normally training a neural network is much more difficult given the distribution of loss function. A complex loss function can cause us to reach a local minimum, but not a global one.

Solution for complex problems:

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial(\theta)}$$

$$\eta = Learning\ Rate$$

This term is called learning rate and it is the one that determines how long is the step that we take when updating the parameters by means of the method of gradient descent.

How to reach a suitable learning rate?

The question we try to answer with the election of the learning rate is: How much the step outcome will affect our weights and biases?

We have to keep in mind that the lower the learning rate, the slower we update our parameters. However, if the learning rate is too high, we might ending diverging and, thus, not finding the local minima. The next figure shows the different scenarios we might fall into when configuring the learning rate (Zulkifli, Hafidz 2018).

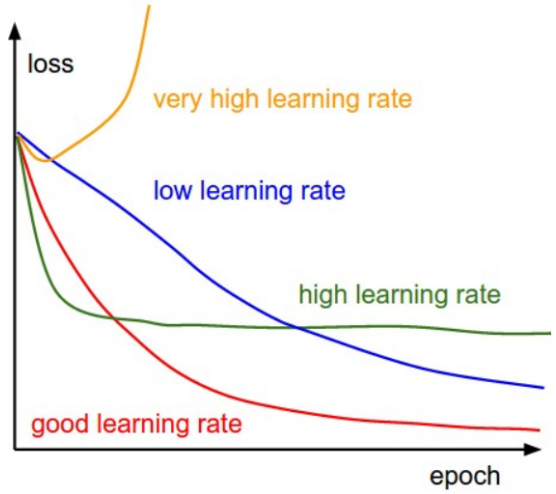


Figure 30: Different scenarios learning rate. Source: Towards Data Science

There are several ways to reach a learning rate that serves us for our neural network. One way is to try until we find the best, or one good enough.

Another more efficient technique would be to design a learning rate that adapts to the cost distribution that we have. The idea of this second approach is that the learning rate changes according to how the training is working. That is to say, depending on the speed with which the training is being executed, or the grandaria of the gradient, the sizes of the parameters, etc.

There are, among others, the following algorithms: Momentum, Adagrad, Adadelta, Adam, RMSProp.

Chain Rule

This section aims to briefly explain the math needed for Deep Learning calculations. The chain rule is a formula used to calculate the derivatives of composite functions. Composite functions are functions composed of functions inside other function(s). (Wikipedia)

$$f' = f(x, w_f)$$
$$g' = g(f')$$
$$y' = k(g')$$
$$cost = criterion(y, y')$$

So now, in order to calculate $\frac{d(cost)}{d(w_f)}$, we do:

$$\frac{d(cost)}{d(w_f)} = \frac{d(f')}{d(w_f)} x \frac{d(g')}{d(f')} x \frac{d(y')}{d(g')} x \frac{d(cost)}{d(y')}$$

The equation of above represents the chain rule in calculus. When doing backpropagation, it is used this chain rule calculus.

3.6 Main DL architectures

The three main architectures in deep learning are: Feed Forward Neural Network, Convultional Neural Networks and the Recurrent Neural Networks. At this point I intend to comment briefly on the last two and more in depth the architecture of Feed Forward, since it is the one I am going to use in the practical part.

Convolutional NN (CNNs)

CNNs learn through hierarchies of patterns. This is, for example, the first convolution layer learns a specific small pattern from the image; a second convolution layer learns larger patterns from what was learned in the first convolution layer, the third convolution layer, and so on. In this way, CNNs learn complex patterns, such as image (Francois 2017).

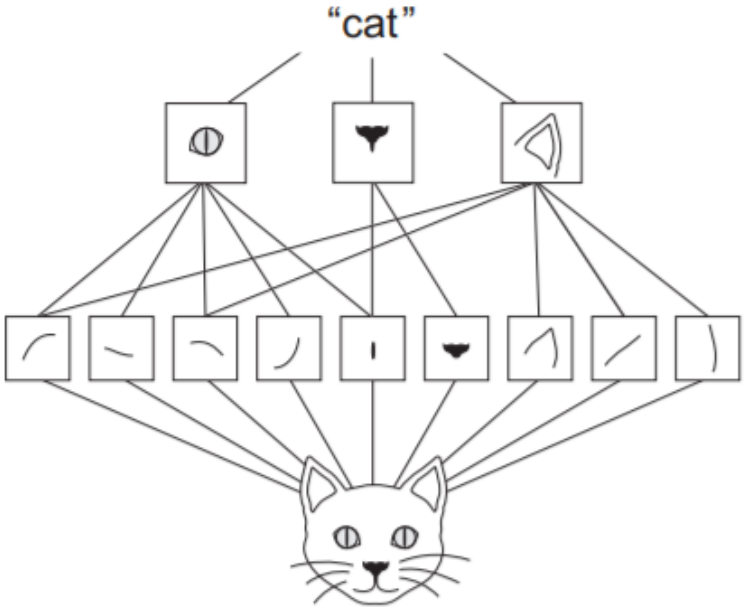


Figure 31: Example CNN. Combination of edges into local objects such as eyes or ears, which combine into high level concepts such as cat. Source: Deep Learning Book

CNNs are the most used networks in Computer Vision tasks (for example, Image Recognition). Their main disadvantage is that they are very susceptible to data quality. In other words, in an image recognition process, if the training image is of poor quality, the algorithm will learn poorly. The same goes for whether the images in the test, if they are of poor quality, will not predict them well.

Recurrent NN (RNNs)

RNNs contain loops that make information always present; that is, they consider that what happens in the past has an impact on the future. Therefore, **neurons in a RNN have memory**, i.e. they can remember important aspects of what has happened and, in fact, use this information to predict the output. Precisely because of this characteristic is why they are usually used in those problems in which the sequence of information is important (speech synthesis, machine translation, etc). For example, in a phrase (word sequence), the network must learn the words in context (using previous words).

Citing the Deep Learning for Python book (Francois 2017):

> *A recurrent neural network (RNN) […] processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far*

I am not going to go into the subject in depth, but the following scheme may help in understanding the functioning of a neural network:

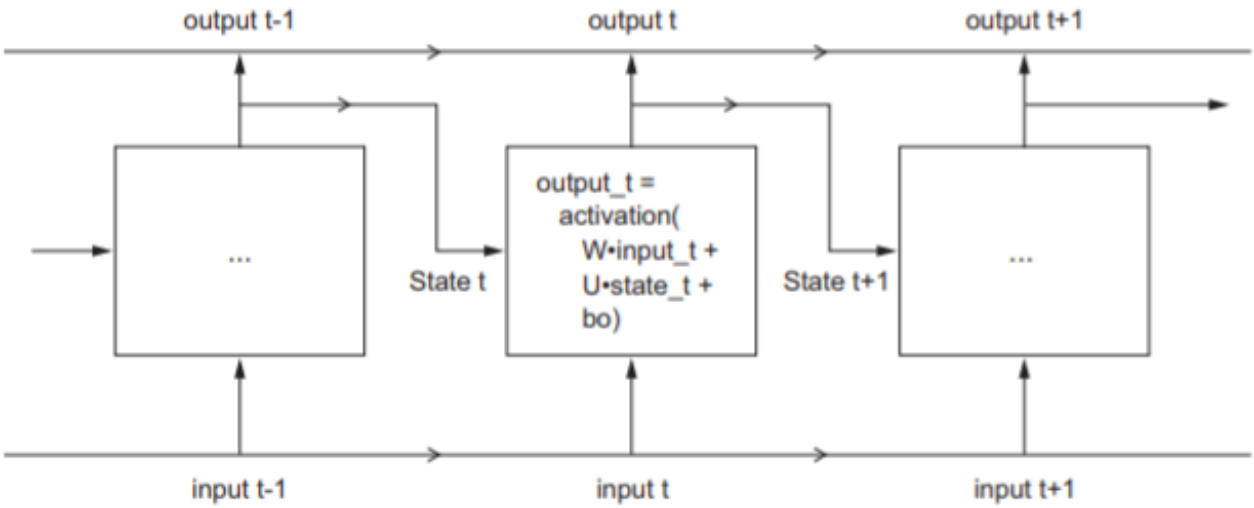


Figure 32: A simple RNN, unrolled over time. Source: Deep Learning Book

So in the scheme above we can see that the information is always present. At the time t, the architecture gathers information from the time t-1 (state_t) multiplied by a parameter "U", which is calculated teh same way it is done with the weights (W). Therefore, the output t (output_t) has information from it is own inputs and the previous one.

Probably,the main disadvantage of RNNs is that, for the moment, it is difficult to learn those data with a very large dependency sequence (e.g., a phrase with dependent words that is very long) (Varangaonkar, Amey 2018).

3.6.1 Feedforward NN

To begin with, it has to be said that the Feed Forward architecture is simply a set of perceptrons in different layers (multilayer perceptron).

Currently, this is the most widely applied form of neural network architecture in practice. The architecture is very simple: the first layer is the input and the last layer the output. When we put layers in between (hidden layers), it becomes a deep neural network (Le, James 2018).

These networks are called Feed Forward because the information only goes forward in the neural network. The main objective of this type of architecture is to approximate a function f. In the case of a classification problem, y = f(x). A feed forward neural network defines a mapping y = f(x; θ) and the idea is it learns the parameters θ that best adjust the function. (Goodfellow, Bengio, and Courville 2016).

This is where all the concepts I've been explaining so far come together: from the activation functions, to the backpropagation algorithm, including the cost function and gradients.

It is not necessary to be a math expert to understand and apply Deep Learning algorithms. In fact, it is possible to become an expert in Deep Learning with a minimum level of math. The latter is due to the fact that the libraries that are used for the application of solutions to problems through Deep Learning already have the necessary calculations incorporated for the purpose.

Despite what I have said in the previous paragraph, I think it is important to understand the

mathematics behind the Deep Learning algorithms. Since in this thesis I am going to use the Feed Forward algorithm, in this point I will explain in a summarized way the mathematics behind the architecture of the Feed Forward algorithm. For this purpose, I base myself on the article *The Matrix Calculus You Need For Deep Learning* (Parr and Howard 2018).

How to train a Feed Forward NN?

Training a neural network means reaching the weights, w, and a bias, b, so that we get the output we want from all inputs. The procedure to do this is to minimize the loss function that compares the output of the neural network with the desired target (the output that should have come out). The process of minimizing the loss may be done, for example, with the Stochastic Gradient Descent (SGD) algorithm (see section 3.5.2).

The entire training process described in the previous paragraph requires partial output derivatives with respect to parameters w and b. Therefore, the objective is to modify the parameters w and b so that the loss function decreases and the output gets closer and closer to the desired one. The repetition of this procedure throughout the neural network is known as the backpropagation process (see section 3.5.3).

Finally, what will be done is to multiply the partial derivative of the cost produced by the neural network by a parameter called learning rate. This parameter will determine how fast or slow the neural network learns (see section 3.5.3). This is called gradient ($\frac{\partial(C)}{\partial(w)}$)

The gradient indicates the direction of the highest cost. Therefore, as we want to reduce the cost, we will update the parameters towards the other direction. That is to say, we will subtract the gradient from the current weights of the nodes and thus the parameter will be updated.

$$w_{t+1} \leftarrow w_t - \eta \frac{\partial(C)}{\partial(w)}$$

The same procedure is done for the bias:

$$b_{t+1} \leftarrow b_t - \eta \frac{\partial(C)}{\partial(b)}$$

3.7 Overfitting and Underfitting problems and solutions

This is one of the most fundamental problems of neural networks.

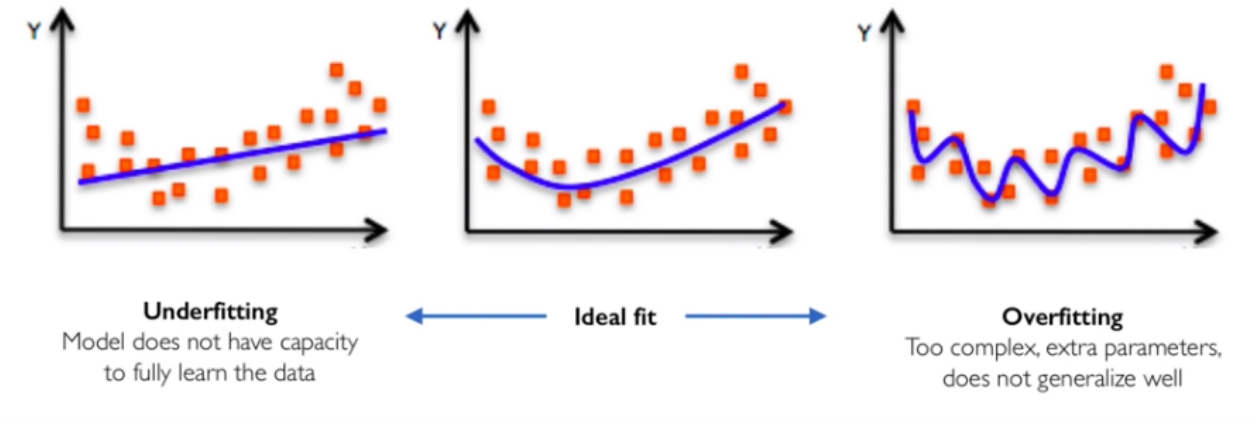


Figure 33: Underfitting and Overfitting. Source: Youtube

In a neural network, the derivative of each parameter gives information on how this has to change to reduce the final error function, taking into account what each neural network unit/node does. That is to say, the units will change their parameters always trying to solve the problems of the other units. This causes the network to end up having complex relationships with each unit, but only with the training data. When we apply the same network for the test (data that the network has not yet seen), these complex relationships do not exist and therefore do not predict well. When this situation occurs, in which the neural network has a very low error in testing, but a very high one in the test, we are in front of an overfitting situation.

We don't want the model we create to do very well in our training data and very badly in the test data. In the end, what we want is to generalize the model to new data. Therefore, it doesn't help that the error in the training set is very small if, later, in the test set it has a big error.

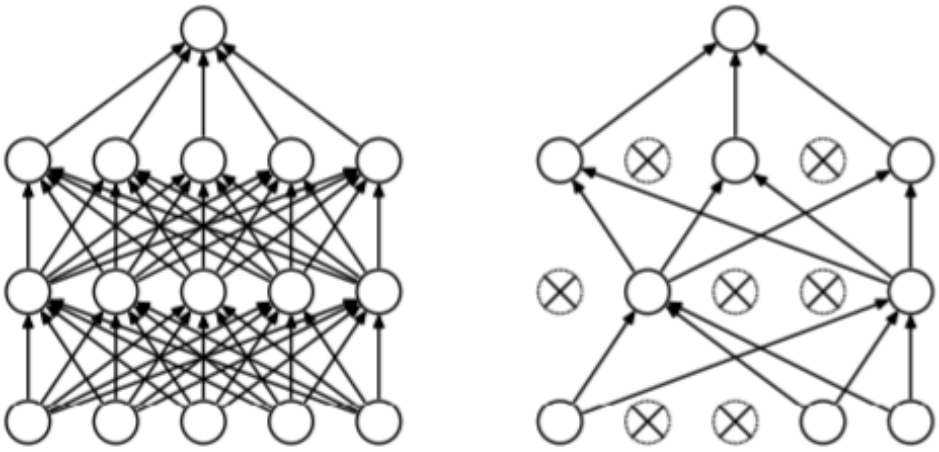
Therefore, the regularization technique serves to put a constraint on our optimization problem, so that it is not over-adjusted.

3.7.1 Dropout

To explain this whole section I will summarize the article *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, because I think that article makes it very clear.

The dropout method is used to eliminate the problem of overfitting on a neural network. It basically consists of eliminating units (hidden and visible) temporarily from a neural network, with a probability of 1- p. That is, each unit of the network is retained with a probability of p.

Visually:



(a) Standard Neural Net (b) After applying dropout.

Figure 34: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. Source: Article Reference

In this way, it has been verified that the neural network in training does not establish very close relations with other units and that, therefore, overfitting does not take place.

Feed Forward operation of a standard neural network:

$$z_i^{(l+1)} = w_i^{(l+1)} y^l + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f(z_i^{(l+1)})$$

Where: L = Number Hidden layers
l ∈ {1,..,L}
$Z^{(l)}$ = vector of inputs into layer l
$Y^{(l)}$ = vector of outputs from layer l (y(0) = X = input)
F(x) = Activation function

Feed Forward operation (with dropout):

$$r_j^{(l)} \sim Bernouilli(p)$$
$$\tilde{y}^{(l)} = r^{(l)} \cdot y^{(l)}$$
$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f(z_i^{(l+1)})$$

Where: $r^{(l)}$ = is a vector of independent Bernoulli random variables each of which has probability p of being 1.

Graphically, the difference between a standard neural network and one in which the dropout method is applied is:

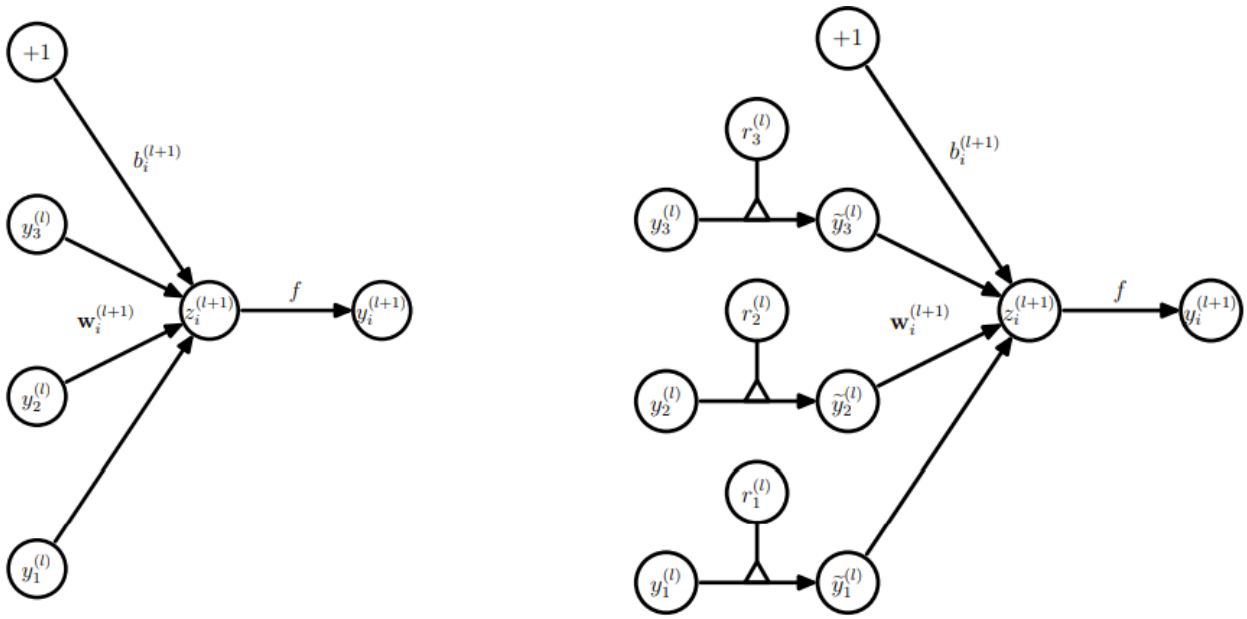


Figure 35: Comparision of the basic operations of a standard (left) and dropout network (right). Source: Article Reference

It must be said that applying these techniques also has side effects, such as that dropout automatically leads to sparse representations.

The article ends with some advice on how to apply this technique.

1. It says that a good dropout should have at least n/p units in the neural network in which it is being applied. Where n is the number of hidden units and p is the probability of retaining a unit.
2. It says that it should also have a learning rate 10-100 times higher than the optimum that would be used in a standard network.
3. It is not the object of study in this thesis, but the author claims that, if the speed of learning is very fast, it can cause the weights of the net to grow too fast. That's why he proposes to use a technique called max-norm regularization. It is basically a restriction to the norm of the weight vector, by means of a constant c (3, 4). $||w|| \leq c$.
4. He ends up recommending a dropout rate of 0.5-0.8 for hidden units and 0.8 for inputs units.

As a summary, the drop-out regularization technique is to assign to 0, in a random way, some nodes of the network. Normally it does it with 50 % (p = 0.5) of the nodes of the network. This means that the network does not have to depend on a single node.

3.7.2 Other regularization techniques

Early Stopping = Stop training before we have the chance to overfit

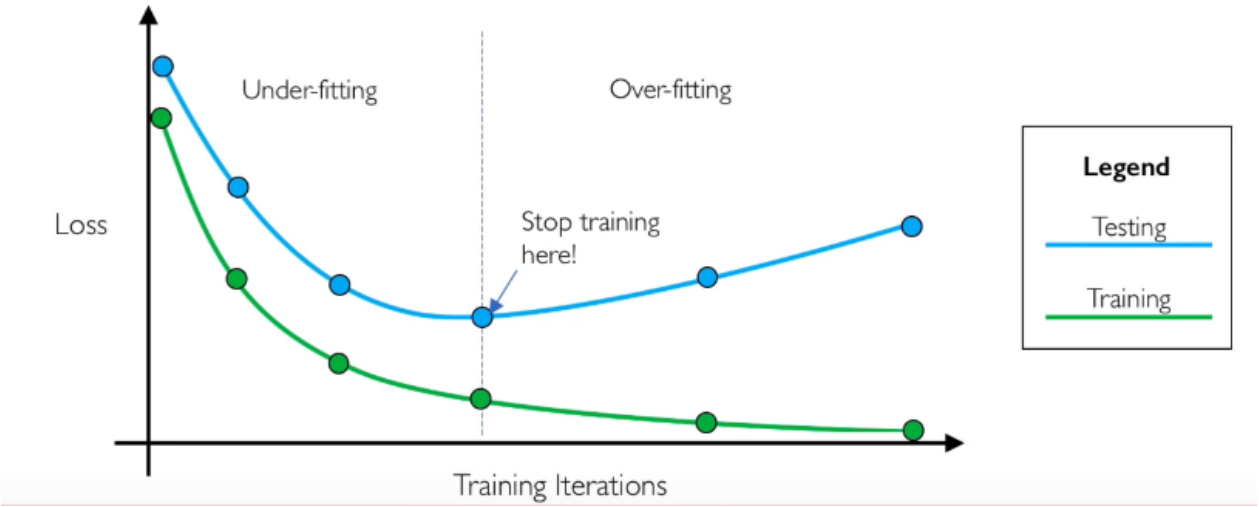


Figure 36: Regularization technique: Early Stopping

4 Real Case Approach

(Github account: https://github.com/marquitosrdr/Final-Thesis-Project)

The main objective of this practical part is to predict the probability of default in credits, regardless of time. That is, to assess whether a credit operation is granted or not. It is therefore intended that, given an operation, the DL model predicts whether it is granted [0] or not [1]. I will start with a simple network, and then make improvements.

The methodology to be followed in this part will be as follows:

First of all, I will create a functional document explaining the database that I will be using to create such a model. Moreover, I will explain the characteristics of the dataset, the source, and the variables it contains. In the same way, I will explain all the changes and transformations that I will do to the original dataset.

Then, I will carry out a descriptive analysis of the entire database, also I will explain the criteria for cleaning the database: treatment of missing data, outliers, variables that are believed not to add value, etc.

Third, I will create the WOE values of my final variables, as well as their respective Information Values (IV).

Afterwards, I will do a logistic regression with the WOE values and original values in order to study and observe the performance of this traditional approach. Then, I will implement a Two-Layer Feed Forward Neural Network (TLFN), and I will decide which one has the best performance.

Finally, I will propose some measures to improve the TLFN.

4.1 Dataset Specifications

4.1.1 Functional document

The database is extracted from the United States Small Business Administration (Small Business Administration, 2019). This database has been used in the article *"Should This Loan Be Approved or Denied"* (Li, Mickel, and Taylor 2018) a work used to teach statistics to American students. Specifically, the authors use the DB to illustrate how logistic regression can be used to classify a loan application as "lower risk" or "higher risk". For the purposes of this thesis, I will use it to create a predictive credit default model.

4.1.2 Dataset background

The U.S. SBA was founded in 1953 with the objective of promoting and assisting small businesses in the U.S. credit market. Small businesses have been a major source of job creation in the U.S.; therefore, in order to encourage their formation and growth. One of the ways in which the U.S. SBA helps these businesses is through a loan guarantee program designed to encourage banks to lend to small businesses. The U.S. SBA acts as an insurance provider to reduce a bank's risk by assuming part of the risk by guaranteeing part of the loan. That is, in the event that a small business grants a loan and the loan becomes delinquent, the U.S. SBA would cover the guaranteed amount.

Since the U.S. SBA only covers a portion of the loss in the event of default, banks would still incur some losses if the company defaulted. It is therefore logical that banks continue to have difficulty in lending to small businesses. One way they can make better decisions is by analyzing relevant

historical data. Specifically, the database focuses on loans in the original Estate, Rental and Leasing industry in California (Li, Mickel, and Taylor 2018).

4.1.3 Dataset technical features

The database contains 899,164 observations of data from 1987 to 2014.

I have changed the original database as follows:

1. To create the "Default" variable. In the variable MIS_Status. If MIS_Status: 'GHGOFF' then Default = 1; else Default = 0.

Explanation of the change: Facilitates interpretation; since it is more intuitive to analyze ordinal variables than specific text names.

2. For the creation of the "New" variable. In the NewExist variable. If NewExist = 2 then New = 1; else New = 0.

Explanation of the change: Facilitates interpretation; it is more intuitive to analyze ordinal variables than specific text names. In this case, if the company is more than 2 years old, it is assigned a 1.

3. I create an artificial variable called "Recession". It will be = 1 if the loan was active during the economic recession 2007-2009; otherwise it will be = 0.

Explanation of the change: This is to identify those loans that were active during the period of economic recession. I have defined this period between the years 2007-12-01 to 2009-06-30.

The code in R to make these transformations is:

```
colnames(dataset)[colnames(dataset) == 'LoanNr_ChkDgt'] <- "ID"
dataset$NewExist[dataset$NewExist == 2] <- 1
dataset$NewExist[dataset$NewExist == 1] <- 0
colnames(dataset)[colnames(dataset) == 'NewExist'] <- "New"
dataset$MIS_Status[dataset$MIS_Status == 'CHGOFF'] <- 1
dataset$MIS_Status[dataset$MIS_Status == 'P I F'] <- 0
colnames(dataset)[colnames(dataset) == 'MIS_Status'] <- "Default"
Daysterm <- (dataset$Term)*30
temp_var <- (as.Date(dataset$DisbursementDate))+Daysterm

dataset["recession"] <- temp_var

dataset <- within(dataset,recession <-
                    ifelse((dataset$recession >= ("2007-12-01")
                    & dataset$recession <= ("2009-06-30")),1,0))
```

Therefore, a summary table of the 20 (there are two more, MIS_Status and NewExist, which I transform and then delete) variables contained in the dataset is as follows:

Table 8: Description Raw Data Set. Source: Own Elaboration

Variable name	Data_type	Description
MIS_Status	Dichotomous qualitative Ordinal	(Variable that I transform). This variable indicates the status of the loan: delinquent (CHGOFF) or has been paid in full (PIF).
NewExist	Dichotomous qualitative Ordinal	(Variable that I transform). Values 1 or 2. Represents 1 if the company has more than two years, and 2, otherwise
ID	Qualitative Nominal	Identifier - Primary key
City	Qualitative Nominal	Customer's city (in the U.S.)
State	Qualitative Nominal	Debtor State (in the U.S.)
Zip	Qualitative Nominal	Debtor's Zip Code (in the U.S.)
Bank	Qualitative Nominal	Bank Name (in the U.S.)
BankState	Qualitative Nominal	Bank State (in the U.S.)
NAICS	Qualitative Nominal	North American Industry Classification System. Values representing the economic sector of the company. Go to the Appendix to see the correspondence of each code with its sector
NoEmp	Discrete Quantitative	Number of employees of the company
New	Dichotomous qualitative Ordinal	(Variable created from the NewExist variable). Values 0 or 1. Represents 0 if the company has more than two years, and 1, otherwise
CreateJob	Discrete Quantitative	Number of jobs created by the company
FranchiseCode	Dichotomous qualitative Ordinal	Whether the company has franchises and the franchise code. In case of not having the code is 00000 or 00001
UrbanRural	Polychotomous qualitative Ordinal	If the company is urban, rural or not specified. 1= Urban, 2= Rural, 0 = Undefined
RevLineCr	Dichotomous qualitative Ordinal	Renewable Loan (Y = Yes or N = No)
LowDoc	Dichotomous qualitative Ordinal	Values Y (Yes) and N (No). Variable that shows how efficient it is to give a loan. A system was created in which loans of less than $150,000 could be processed using less than one page. In this line, Y indicates loans that only need one one-page application and N indicates that more information is needed
DisbursementGross	Continuous Quantitative	Amount of loan approved by the bank. (Total Loan Amount)
BalanceGross	Continuous Quantitative	Gross loan amount outstanding. (What's left for the company to pay)
Default	Dichotomous qualitative Ordinal	Variable Response. Variable created from the variable MIS_Status. If the variable MIS_Status = CHGOFF, then the variable Default = 1, otherwise (PIF), variable Default = 0
ChgOffPrinGr	Continuous Quantitative	Amount of the cancellation
GrAppv	Continuous Quantitative	Amount of loan approved by the bank. (Total Loan Amount)
Recession	Dichotomous qualitative Ordinal	(Artificial variable created from the data). Values 1 and 0. Corresponds to 1 if the loan was active during the Great Recession (2007 to 2009)

Once the database has been defined, I proceed to a descriptive analysis of their variables.

4.1.4 Managing the data

4.1.4.1 Dataset modifications / Cleaning data

(For a deep knowledge of this part, refer to my Github account (https://github.com/marquitosrdr/Final-Thesis-Project/blob/master/Cleaning_dataset)).

A summary of what I have done in this part is:

- First, as described above, I create three more variables (New, Default and Recession).
- Second, I deal with the outliers and missing values. In this part I finally chose to use **two numeric variables** (DisbursementGross and NoEmp); and **8 categorical variables** (State, BankState, New, UrbanRural, RevLineCr, LowDoc and Recession).

- Third, I decide to remove all the missing values and unuseful variables. That makes my dataset pass from having 899164 rows and 20 variables to having 343467 rows and 11 variables.
- Four, I group the states by: midwest, north.east, south, and west.

These are the 11 variables I will finally work with:

Table 9: Description Data Set. Source: Own Elaboration

Variable name	Data_type	Description
ID	Qualitative Nominal	Identifier - Primary key
State	Qualitative Nominal	Debtor State (in the U.S.)
BankState	Qualitative Nominal	Bank State (in the U.S.)
NoEmp	Discrete Quantitative	Number of employees of the company
New	Dichotomous qualitative Ordinal	(Variable created from the NewExist variable). Values 0 or 1. Represents 0 if the company has more than two years, and 1, otherwise
CreateJob	Discrete Quantitative	Number of jobs created by the company
UrbanRural	Polychotomous qualitative Ordinal	If the company is urban, rural or not specified. 1= Urban, 2= Rural, 0 = Undefined
RevLineCr	Dichotomous qualitative Ordinal	Renewable Loan (Y = Yes or N = No)
LowDoc	Dichotomous qualitative Ordinal	Values Y (Yes) and N (No). Variable that shows how efficient it is to give a loan. A system was created in which loans of less than $150,000 could be processed using less than one page. In this line, Y indicates loans that only need one one-page application and N indicates that more information is needed
DisbursementGross	Continuous Quantitative	Amount of loan approved by the bank. (Total Loan Amount)
Default	Dichotomous qualitative Ordinal	Variable Response. Variable created from the variable MIS_Status. If the variable MIS_Status = CHGOFF, then the variable Default = 1, otherwise (PIF), variable Default = 0
Recession	Dichotomous qualitative Ordinal	(Artificial variable created from the data). Values 1 and 0. Corresponds to 1 if the loan was active during the Great Recession (2007 to 2009)

4.1.4.2 Descriptive Analysis

(For a deep knowledge of this part, refer to my Github account (https://github.com/marquitosrdr/Final-Thesis-Project/blob/master/Descriptive_Analysis)).

Once I have cleaned the dataset, it is time to see how the variables are distributed. The objective is to check that all make sense, and change the variables formats (factor, numeric, etc) to make them easier to use.

I start by doing a histogram and a boxplot for the numeric variables. Then, I do a Pie plot and Barplots for some of the categorical variables. Refer to the Annex to find some summaries and statistics from the most important variables of the dataset.

Once I have examined the numerical and categorical variables of my dataset, it is time to see how these variables are corralated with my response variable, Default.

Therefore, this part aims to study and examine the data in a wider way. I pretend to get an intuition of how the default variable is distributed among the other variables in the dataset.

As an interesting example, I do a graphical analysis from the State and Default variables. The objective is to see which states have more defaults. In order to do so, I show a graph by mapping the states with their associate default.

Heatmap State vs Default

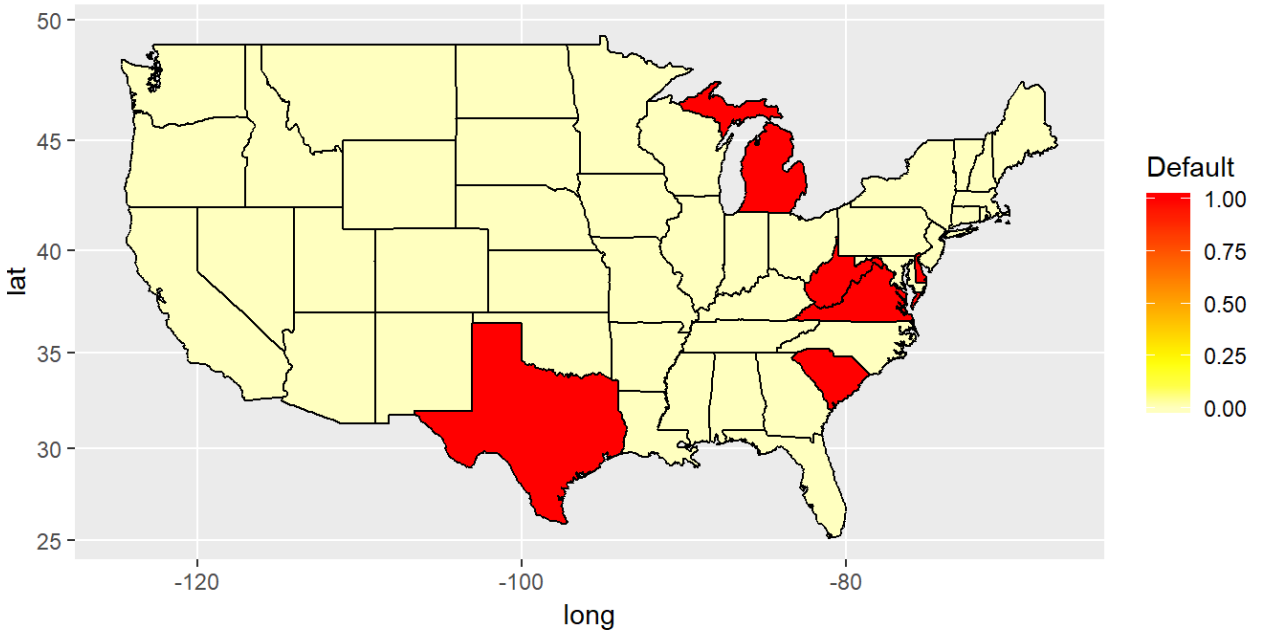


Figure 37: Heatmap State vs Default. Source: Own Elaboration

Thanks to this plot I can see the states which seem to have more default cases. However, it has to be said that due to the lack of memory of my computer, I just can run 30000 samples (chosen randomly). That is why I do corroborate this results by doing a frequency table, as follows:

Table 10: State vs Default. Source: Own Elaboration

States	Freq
California	44602
New York	27787
Texas	22012
Florida	17782
Pennsylvania	15045
Ohio	13673
Illinois	12444
Massachusetts	11914
New Jersey	9431
Washington	9086

So, the State that has the most default rates is California. (It might not match with the graphic due of what I explained before about the lack of memory of my computer).

4.1.4.3 IV and WOE

This whole part of the thesis is extracted from the work published by AMAZON (Amazon AWS, 2018).

In R you can make these groupings using the package called scorecard::woebin(). It automatically

creates the optimal bins and calculates the WOE values. The details of this package can be found in the Annex [Package Scorecard].

Woebin generates optimal binning for numerical, factor and categorical variables using methods including tree-like segmentation or chi-square merge. woebin can also customizing breakpoints if the breaks_list was provided. The default woe is defined as $ln(Bad_i/Good_i)$. In my analysis I prefer the equation $ln(Good_i/Bad_i)$, so I set the argument positive as negative value. If there is a zero-frequency class when calculating woe, the zero will replaced by 0.99 to make the woe calculable (CRAN R, 2019).

Rationale of the use of WOE in the DL algorithm

This point is intended to give an explanation that justifies the use of WOE as predictive values of the default event in a credit. I explain why it is better to use the WOE values of the variables; instead of using their original values.

Following the context described in point 2 of this paper, WOEs are a transformation of the original data that are usually used to discretize continuous variables (normally a logarithmic transformation of events that have not been default among those that have been default).

WOE represent the predictive power of each category, several studies cited above (Majer 2006) show that the use of WOE leads to a higher percentage of hits.

The analysis of the WOE allows:

- Consideration of the independent contribution of each variable to the final model.
- Detect linear and non-linear relationships.
- Classify the variables in terms of "univariate" predictive power.
- Visualize the correlations between the predictive variables and the binary variable.
- Compare the predictive power of continuous and categorical variables without creating dummy variables.
- Evaluate the predictive power of lost values.

It must be said that, although the use of WOE is particularly suitable for logistic regression modeling, I will also use them in the practical implementation of the TLFN.

For a deep knowledge of this part, refer to my Github account (https://github.com/marquitosrdr/Final-Thesis-Project/blob/master/WOE_variables)

Information Values

Table 11: IV from my variables. Source: Own Elaboration

variable	info_value
DisbursementGross	0.635
UrbanRural	0.516
BankState	0.364
State	0.171
LowDoc	0.135
Recession	0.122
NoEmp	0.037
RevLineCr	0.032
New	0.004
ID	0.000

Weights of Evidence

I will select those variables with an IV > 0.02

Therefore, the final variables -with their WOE- I will use are:

Table 12: WOE from my variables. Source: Own Elaboration

State_woe	BankState_woe	NAICS_woe	NoEmp_woe	New_woe	UrbanRural_woe
0.1541	0.1130	-0.1707	-0.0772	0.1086	1.7684
0.1541	0.1130	-0.0292	-0.0772	0.1086	1.7684
0.1541	0.1130	0.8907	-0.0772	-0.0378	1.7684
-0.0859	-0.2066	-0.0292	-0.0772	0.1086	1.7684
-0.0859	-0.2066	-0.0292	-0.0772	0.1086	1.7684
-0.0859	-0.2066	-0.0292	-0.0772	0.1086	1.7684
-0.0859	0.1130	-0.1707	-0.0772	0.1086	-0.3584
-0.0859	-0.2066	-0.1707	-0.0772	0.1086	1.7684
-0.0859	-0.2066	0.8907	-0.0772	-0.0378	1.7684
-0.0859	-0.2066	-0.0292	-0.0772	-0.0378	-0.3584

Table 13: WOE from my variables (II). Source: Own Elaboration

RevLineCr_woe	LowDoc_woe	DisbursementGross_woe	Recession_woe
0.1732	1.7622	-0.1325	0.1178
0.1732	1.7622	-0.1325	0.1178
0.1732	-0.0774	0.8472	0.1178
0.1732	1.7622	-0.1325	0.1178
0.1732	-0.0774	0.8472	0.1178
0.1732	1.7622	-0.1325	0.1178
0.1732	-0.0774	0.8472	0.1178
0.1732	-0.0774	0.8472	0.1178

0.1732	-0.0774	-0.1325	0.1178
0.1732	-0.0774	-0.1325	0.1178

WOE variable table description

As i mentioned above, the scorecard::woebin() returns a list with one element for each variable. There also is a plotting function that we can use to make meaningful plots and check the binning scorecard::woebin_plot().

Note: The value transformation have followed this equation $ln(Good_i/Bad_i)$

The bins are:

State

Table 14: State. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
State	midwest%,%north.east	0.1541	0.0107	0.0202
State	south	-0.0859	0.0020	0.0202
State	west	-0.1637	0.0076	0.0202

BankState

Table 15: BankState. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
BankState	midwest	0.1130	0.0039	0.0385
BankState	north.east	0.3140	0.0176	0.0385
BankState	south	-0.2066	0.0151	0.0385
BankState	west	-0.1062	0.0018	0.0385

NoEmp

Table 16: NoEmp. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
NoEmp	[-Inf,8)	-0.0772	0.0049	0.0313
NoEmp	[8,13)	0.2879	0.0089	0.0313
NoEmp	[13, Inf)	0.5288	0.0175	0.0313

UrbanRural

Table 17: UrbanRural. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
UrbanRural	0	1.7684	0.4277	0.5162
UrbanRural	1	-0.3584	0.0884	0.5162
UrbanRural	2	0.0351	0.0002	0.5162

RevLineCr

Table 18: RevLineCr. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
RevLineCr	N	0.1732	0.0155	0.0321
RevLineCr	Y	-0.1857	0.0166	0.0321

LowDoc

Table 19: LowDoc. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
LowDoc	N	-0.0774	0.0057	0.1349
LowDoc	Y	1.7622	0.1292	0.1349

DisbursementGross

Table 20: DisbursementGross. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
DisbursementGross	[-Inf,150000)	-0.1325	0.0143	0.0881
DisbursementGross	[150000,250000)	0.4371	0.0190	0.0881
DisbursementGross	[250000, Inf)	0.8472	0.0548	0.0881

Recession

Table 21: Regression. Source: Own Elaboration

variable	bin	woe	bin_iv	total_iv
Recession	0	0.1178	0.0124	0.1221
Recession	1	-1.0465	0.1097	0.1221

4.2 Sensitivity and Specificity in Credit Risk

This part of the thesis is a summary from the web article published by Statinfer (Statinfer, 2018).

I especially like how these authors explain this point. Therefore, I am just going to cite them:

"By changing the threshold, the good and bad customers classification will be changed hence the sensitivity and specificity will be changed. The question then is which one of these two we should maximize? Ideally we want to maximize both Sensitivity and Specificity. But this is not possible always. There is always a tradeoff. Sometimes we want to be 100% sure on Predicted negatives, sometimes we want to be 100% sure on Predicted positives. Sometimes we simply don't want to compromise on sensitivity sometimes we don't want to compromise on specificity. The threshold is set based on business problem."

This next table explains very well what sensitivity and specificity mean and why we want to have more sensitivity in some cases and more specificity in some others.

Table 22: Specificity and Sensitivity in Credit Default. Source: statinfer.com

	1(Default)	**0(Non-Default)**	
1(Default)	**TRUE POSITIVE (TP)** **Customer is BAD** **Model predicts as BAD**	**FALSE NEGATIVE(FN)** **Customer is BAD** **Model predicts as GOOD**	**Sensitivity** **TP/(TP+FN)**
0(Non-Default)	**FALSE POSITIVE (FP)** **Customer is GOOD** **Model predicts as BAD**	**TRUE NEGATIVE (TN)** **Customer is GOOD** **Model predicts as GOOD**	**Specificity** **TN/(TN+FP)**

If we wrongly reject a good customer, our loss is very less compared to giving a loan to a bad customer. Hence, in that problem we would like to have the highest sensitivity possible.

I will refer to this point when doing the practical analysis.

Before applying the TLFN algorithm, I will do the analysis through a Logit model regression. The aim of which is later comparation. My initial hypothesis is that the Neural Network will perform better than the Logit model. In order to corroborate this I will calculate the ROC curve, the AUC value and the specificity and sensibility values. At the end of this point I will insert a table with the comparations between the logit and neural network predictions.

4.3 Implementation Logit Model with WOE values

Since it is not the main objective of this thesis, I am not explaining the theory behind a logit model. However, I will add the code and some graphics and statistics, and comments to make this point more consistent and robust.

As to the best possible logistic regression results, I will apply a balancing to my dataset. The number of non-default cases is 79768 and the number of default cases is 263699. Therefore, I will randomly choose 79768 non-default cases and do the regression with them. In addition, in order to minimize the error, I will repeat this procedure several times and see if the results are consistent.

The results are as follows:

Logit model result

I implement a logistic model in R with the stepwise algorithm (direction = both) to select the most relevant variables.

$P(Default = 1) = \frac{1}{1+exp(0.005-0.704 \cdot WOEstate-...-0.888 \cdot WOErecession)}$

The whole coefficients from the model are:

Table 23: Logit Model WOE values - Coefficients

Logit Model WOE value - Coefficients	
(Intercept)	0.005128
State_woe	-0.704622
BankState_woe	-0.737444
NoEmp_woe	-0.496589
UrbanRural_woe	-0.945094
RevLineCr_woe	0.769201
LowDoc_woe	-0.255808
DisbursementGross_woe	-0.716863
Recession_woe	-0.888542

Unlike linear regression with ordinary least squares estimation, there is no R2 statistic which explains the proportion of variance in the dependent variable that is explained by the predictors (Mathew, 2015).

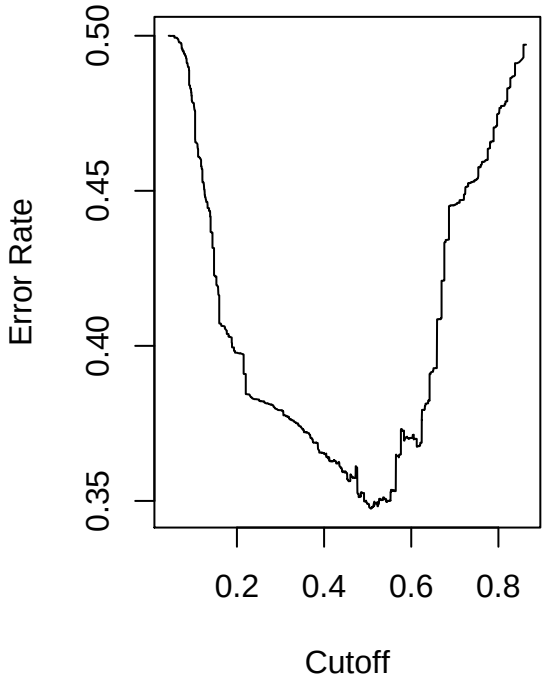
However, there are a number of pseudo R2 metrics that could be of value. I will use the McFadden's pseudo R2. McFadden's pseudo R-squared ranging from 0.2 to 0.4 indicates very good model fit. My model has a McFadden's pseudo R-squared of 0.1057 is likely not a terrible model, at least by this metric, but it isn't particularly strong either (McFadden and others 1973).

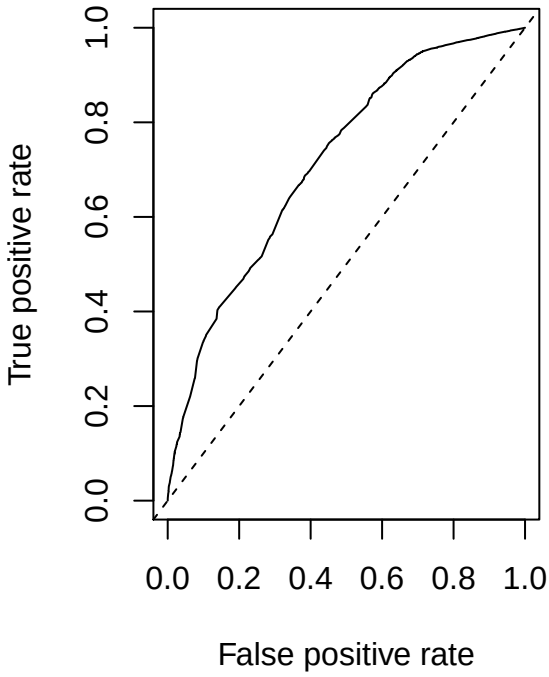
This is a summary performance table of the implementation of a logistic model with the WOE values:

Table 24: Logit model WOE

Logit model WOE	
Sensitivity	0.533
Specificity	0.763
AUC	0.717
McFadden R^2	0.118

The ROC and error curves are:





The confusion matrix is (with a treshold of 0.5)

Table 25: Confusion Matrix WOE values (threshold 0.5).
Source: Own Elaboration

	0	1
0	43019	18943
1	36749	60825

Now, I a going to do the same analysis with the original values of the dataset (instead of the WOE values)

4.4 Implementation Logit Model with original values

I implement a logistic model in R with the stepwise algorithm (direction = both) to select the most relevant variables.

Table 25: Logit Model original values - Coefficients

Logit Model original values - Coefficients	
(Intercept)	-4.620e-01
Statenorth.east	7.252e-02
Statesouth	1.407e-01
Statewest	2.028e-01
BankStatenorth.east	-3.041e-01
BankStatesouth	2.137e-01
BankStatewest	1.841e-01
NoEmp	-2.034e-02
UrbanRural	5.856e-01
LowDocY	-1.329e+00
DisbursementGross	-2.658e-06
Recession	1.100e+00

The whole coefficients from the model are:

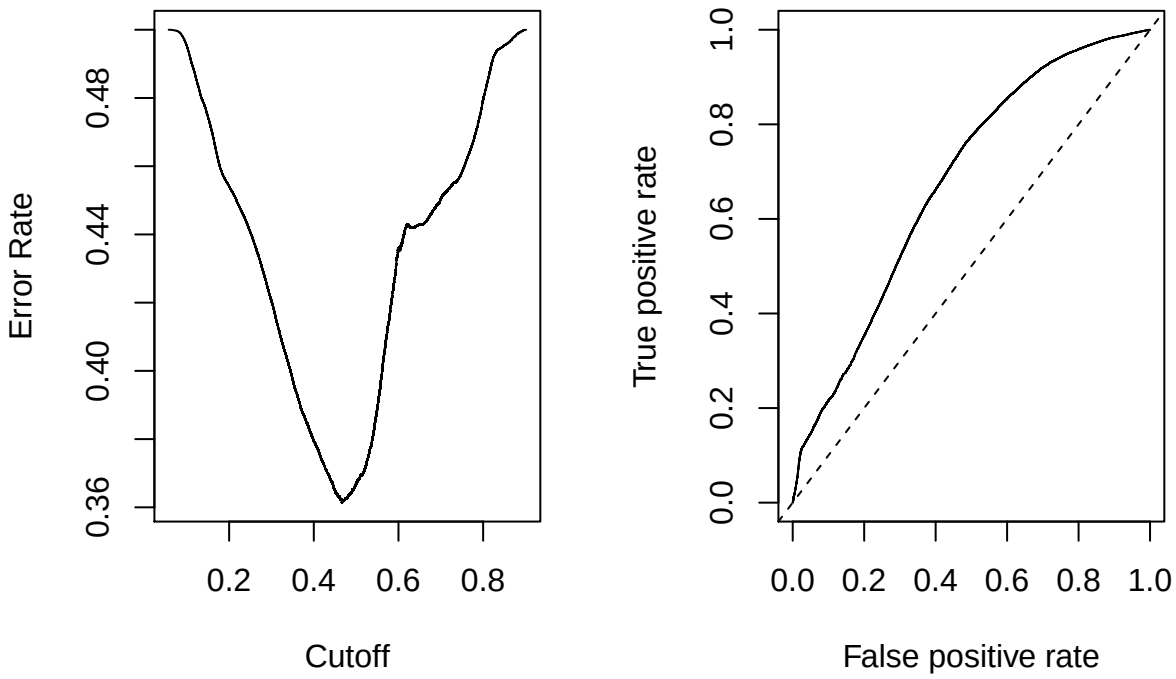
The model with the original values has a McFadden's pseudo R-squared of 0.07198, so it is a weak model.

This is a summary performance table of the implementation of a logistic model with the original values:

Table 26: Logit model original Values

Logit model original Values	
Sensitivity	0.536
Specificity	0.762
AUC	0.678
McFadden R^2	0.081

The ROC and error curves are:



The confusion matrix is (with a treshold of 0.5)

Table 27: Confusion Matrix original values (threshold 0.4).
Source: Own Elaboration

	0	1
0	45895	24652
1	33873	55116

Conclusions

As i expected, the logistic model when using the WOE variables has better performance statistics. Now, I am going to implement a Feed Forward Neural Network to both the original and WOE values, in order to compare the logistic and NN model performance.

4.5 Benchmark: Two-Layer Feed Forward Neural Network (TLFN) to predict loan default

The whole implementation has been done with PYTHON. Refer to the Annex or to my Github account to see the code. (https://github.com/marquitosrdr/Final-Thesis-Project).

The first thing to say is that the application of a Feed Forward Neural Network is not the only method used in classification problems. There are other algorithms that may, or may not, be more optimal for this problem, such as Random Forest or Support Vector Machine (SVM). However, this

thesis aims to study the implementation of default prediction through a TLFN. In the discussion section, it is proposed to make this prediction using algorithms such as Random Forest or Support Vector Machine (SVM) and a comparison to study which technique would be the most appropriate.

Therefore, the choice of a Feed Forward Neural Network has been due to the topic chosen in the present thesis, and in the curiosity of the results.

This point intends to give a very detailed explanation of the whole process of construction of the network; as well as a deep explanation of all the parameters chosen and decisions taken.

In the annex, you can find the most relevant Python code parts, as well as the installation processes of the libraries necessary for the application of Deep Learning techniques, such as the Keras library.

Train, Test, Validation sets

In Machine Learning, we always divide our data into training and testing part meaning that we train our model on training data and then we check the accuracy of a model on testing data. Testing your model on testing data will only help you evaluate the efficiency of model.

It is usually divided into 80:20, 75:25 or 60:40. In this case I will do 75 % training and 25 testing.

Also, in order to evaluate the performance of the neural network I will create a validation set. I will do this by extracting 10,000 data from 75% of the training set.

Table 27: Train, Test and Validation sets

Train set	75 % - 10000	247.600 data
Test set	25 %	85.867 data
Validation set		10.000 data
Total		343.467 data

Balanced dataset

Because the sample is not balanced, I will balance it. The number of non-default events are 263.699 and the default events are 79.768 Therefore, I will randomly choose 79.768 non-default events and apply the TLFN with them. In addition, in order to minimize the error, I will repeat this procedure several times, and see if the results are consistent.

Therefore, I will do the whole analysis with both the total dataset, and the balanced one. This is because I want also to show the importance of balancing the data when doing analysis.

Software

For the implementation of the neural network, I have used Python and the Keras library (Tensorflow), along with all the required additional libraries, such as Numpy, sckitit, etc. In the annex, you can find the most relevant code parts, together with the installation process of these libraries.

I have decided to use Pyhton for several reasons. First, because there is a lot of information about Deep Learning applications in Python. Second, because it is one of the most used software in this industry and, therefore, it is the one I will probably use in the future, if I dedicate myself in this

sector; and third, because I personally find it very consistent and with many kind of packages that allow a wider application of concepts than other softwares.

Activation function

A detailed explanation of some of the most relevant activation functions currently in the Deep Learning field can be found in section 3.3 of this thesis. At the end of this point, I present a table with the characteristics of each one and concluding which is better to use in each type of problem. Based on this point, I will use the ReLU activation function for the hidden layers and the sigmoid activation function for the output Layer.

Optimizer

As discussed in point 3.5, there are several optimization algorithms: From the famous Stochastic Gradient Descent (SGD) to variations and improvements such as Momentum, RMSProp, Adam.

RMSProp has a very good feature: it incorporates adaptive learning-rate methods. Therefore, they provide a fast convergence.

RMSprop divides the learning speed by an exponential decay average of square gradients. Hinton (Hinton, Srivastava, and Swersky 2012) suggests that γ must be set to 0.9, while a good default value for the learning rate η is 0.001.

As an example, putting Hinton ideas in consideration, the RMSProp optimizer would be:

$$E[f^2]_t = 0.9 \cdot E[f^2]_{t-1} + 0.1 \cdot f_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[f^2]_t + \epsilon}} \cdot f_t$$

Therefore, I will use the RMSProp optimizer. I will implement Hinton ideas. thus, the learning rate will be 0.001, and the γ wil be 0.9.

Moreover, it has to be said that the learning rate decay over each update will be as default: 0.0.

Loss Function

Following the same rationale as the choice of activation functions, in point 3.4 I make a theoretical study of the possible loss functions in the neural network. In this point, I also end up putting a summary table saying which function is more suitable for each case. In the case of binary classification, the most appropriate activation function is binary_crossentropy.

Batches and Epochs

The batch size is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated. The batch size is a number of samples processed before the model is updated. The number of batches should be between 1 and the number of training samples.

The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset. The number of epochs is the number of complete passes through the training dataset. We can run the algorithm as long as we want. The number of batches should be between 1 and infinite.

I will try with some batches and some epochs, and I will display only the best results.

Classification Threshold

What I will do is a comparison of different cut-off points and I will display the one with better results. In this way, I will present a consistent analysis of the sensitivity of my network.

Number of layers and nodes

One of the most important questions when creating a neural network is to choose the number of neurons and layers that will be used.

This problem has been a matter of discussion for a long time, and still is today. There is no 100% accurate method for calculating the number of neurons and layers in a neural network. Each choice will depend on the problem we face. However, there are studies and different approaches that can be useful.

The panacea of the calculation of the number of neurons probably comes from the studies of Kolmogorov´s applied to neuronal computation (Hecht-Nielsen 1987). This determines and proves that any continuous function can be represented by a neural network with a single layer and exactly 2n+1 nodes (n being the number of inputs). However, this rule established by Hetch-Nielsen applies only to a more complicated type of activation functions. Therefore, it is not applicable in this thesis. However, other studies (Kurkova 1992) demonstrates that for neural networks with sigmoidal activation functions it is sufficient to incorporate two hidden layers. However, it says that the number of neurons in each layer can be as large as the number of training samples.

Therefore, based on these studies, in my application I will use two hidden layers. The second layer will be to reduce the total number of nodes required. Therefore, I will apply a Two-hidden-Layer Feedforward Neural-Network (TLFN).

Additionally, since a TLFN could generate doubts about whether it is a Deep Learning algorithm or not, and with the intention of applying the Deep Learning concepts that I have been explaining throughout this thesis, I will implement a multi-layered Feed Forward. As my own criteria, I believe that 20 layers would be enough to name a network as Deep Learning. Although it is probably unnecessary to apply so many layers, I will test it by doing the experimental implementation.

Now the most complex question: How many neurons should I add to each hidden layer?

The article *Learning Capability and Storage Capacity of Two-Hidden-Layer Feedforward Networks* by Guang-Bin Huang (Huang 2003) demonstrates that a TLFN with $2 \cdot \sqrt{(m+n) \cdot N}$ ($<<N$) neurons can learn N distinct samples (x_i,t_i) with a very small error; being m the number of output neurons and N the number of distinct samples. However, because, in this thesis, the database has an N of approximately 350.000 data, the number of neurons needed would be enormous and therefore does not computationally compensate.

To determine the number of neurons there are four basic methods (How many hidden layers and nodes? By Stathakis):

1. Trial and error: Go testing. You will probably end up with a very suboptimal result.
2. Heuristic Search: The objective is to extract a formula that estimates the number of nodes in the hidden layers according to the nodes we have in the input and output layers.
3. Exhaustive search: Evaluate all the possible alternatives and studies that talk about the number of neurons to choose. This is not usually an option in its entirety, as it requires a lot of time.

4. Pruning and constructive algorithms: These algorithms extract the number of optimal nodes by adding and removing weights from the neural network.

Due to the ease of the Heuristic search, I will use this approach to determine the number of nodes. I will then apply the K-fold cross validation statistical technique to determine which is the combination of nodes that minimizes error in both layers.

For the input layer: Reviewing the literature previously mentioned in the present point, in which it is deduced that the number of neurons that comprise this layer is equal to the number of columns/variables of my dataset (adding one for the bias). As my dataset comprises nine explanatory variables, the input layer will be 9+1(bias) = 9 neurons (+1 internal).

For the output layer: I will apply a simple neuron, as it is a classification problem.

For hidden layers:

Deciding the number of neurons in the hidden layers is a very important decision. Although these layers do not interact with the "outside", they have a very high influence on the outcome of the process. It is for this reason that the choice of the number of neurons has to be properly studied.

The issue is that using too many neurons can cause overfitting or insufficient data in training. On the other hand, using too few neurons can lead to underfitting. The concepts of overfitting and underfitting are explained in this thesis (point 3.7).

Some rule-of-thumb for determining the number of neurons are:

- the optimal size of the hidden layer is usually between the size of the input and size of the output layers.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

Despite these rule-of-thumbs, I would like to bring a more consistent approach. That is why I will base myself in the article *Review on Methods to Fix Number of Hidden Neurons in Neural Networks* written by K. Gana Sheela and S.N. Deepa to finally determine the number of neurons I will apply to my TLFN (Sheela and Deepa 2013). This article does a reviewing of the methods to fix hidden neurons in neural networks for the past 20 years.

Once I determine three possible nodes in the neural network through the heuristic research, I will apply a k-fold cross validation algorithm to finally decide the number of nodes I will use in each layer.

These are the nodes which could fit well in the TLFN:

1. Tamura and Tateishi method (Tamura and Tateishi 1997)

$$N_h = n - 1 = 8$$

   Where: N_h = Number of nodes in the hidden layer. n = Number of input nodes; n = 9, in my case.

2. Zhang et al. method (Zhang, Ma, and Yang 2003)

$$N_h = \frac{2^n}{n} + 1 = 57.8$$

Where: N_h = Number of nodes in the hidden layer. n = Number of input nodes; n = 9, in my case.

3. Hunter et al. method (Hunter et al. 2012)

$$N_h = 2^n - 1 = 511$$

Where: N_h = Number of nodes in the hidden layer. n = Number of input nodes; n = 9, in my case.

Moreover, I have the intuition that might be some optimal node between the 8 and 58. Therefore, I will also validate the possibility of having 30 nodes.

Cross-validation procedure: K-Fold

The K-Fold cross validation procedure will be used to choose the combination of nodes that minimizes the loss of my neural network. The idea is to split the dataset into equal k segments and, for each iteration, a segment is retained as data to validate and the training is done with the other k-1 pieces. This procedure carries out with N proposals of nodes that you have. Finally, we choose the combination of nodes that minimizes the loss of the network.

The following scheme explains very well the concept of K-fold cross validation:

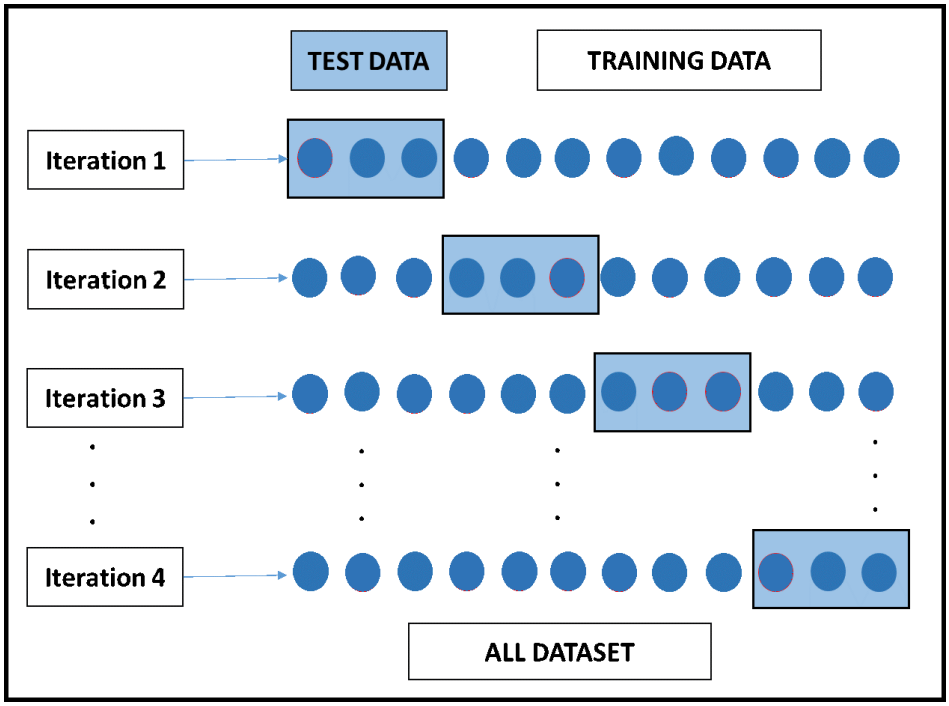


Figure 38: Cross Validation (k=4). Source: Own Elaboration

Therefore, in my case, because I have about 300.000 data and 9 variables, I will apply a partition of k = 10. Therefore, I will split my dataset in 10 equal segments. In addition, the "candidates" to be the final nodes, which have been extracted from the heuristic study done previously are: 8, 30, 58, 511 nodes.

Moreover it has to be said that I have done this cross validation with balanced and unbalanced data.

This is because the number of non-default cases is 79768 and the number of default cases is 263.699. Therefore, I will randomly choose 79768 non-default cases and apply the TLFN with them. In addition, in order to minimize the error, I will repeat this procedure several times and see if the results are consistent.

4.5.1 FeedForward NN with WOE values:

Cross-validation procedure: 10-Fold

The resulting graph of 10-Fold cross validation with my WOE dataset is:

WOE: Balanced data

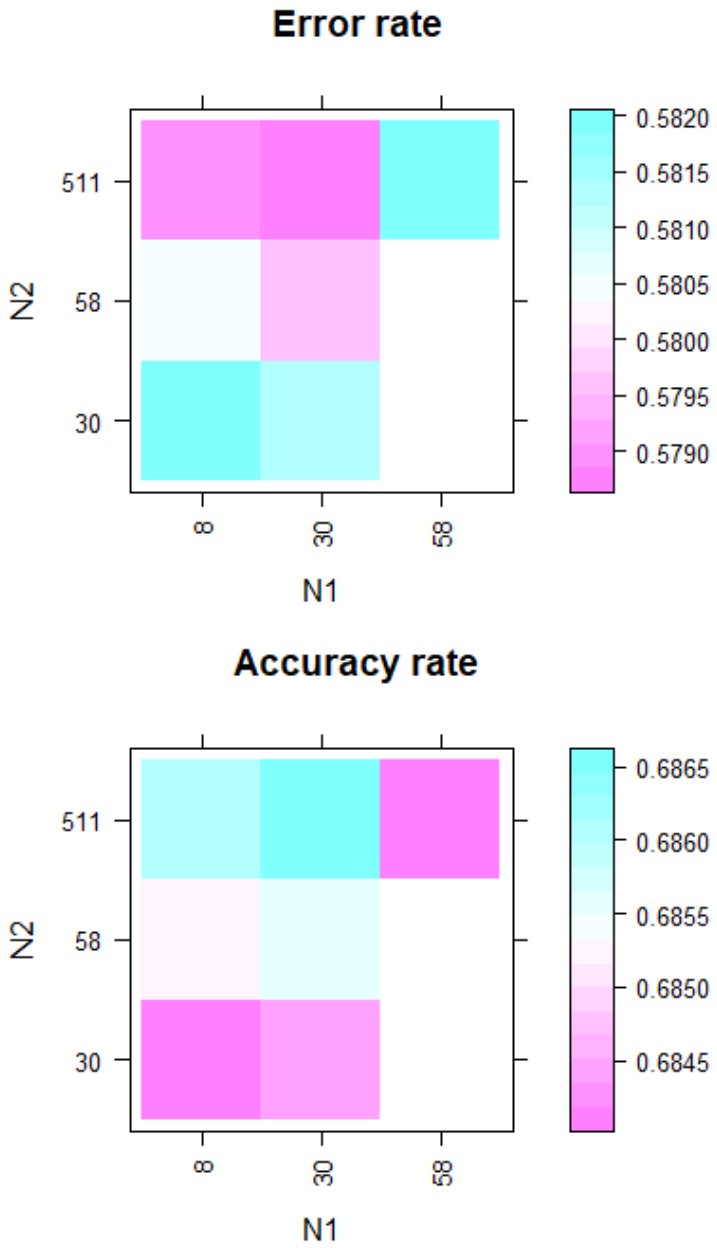


Figure 39: Accuracy TLFN. Cross Validation (k=10). Balanced dataset. Source: Own Elaboration

WOE: Unbalanced data

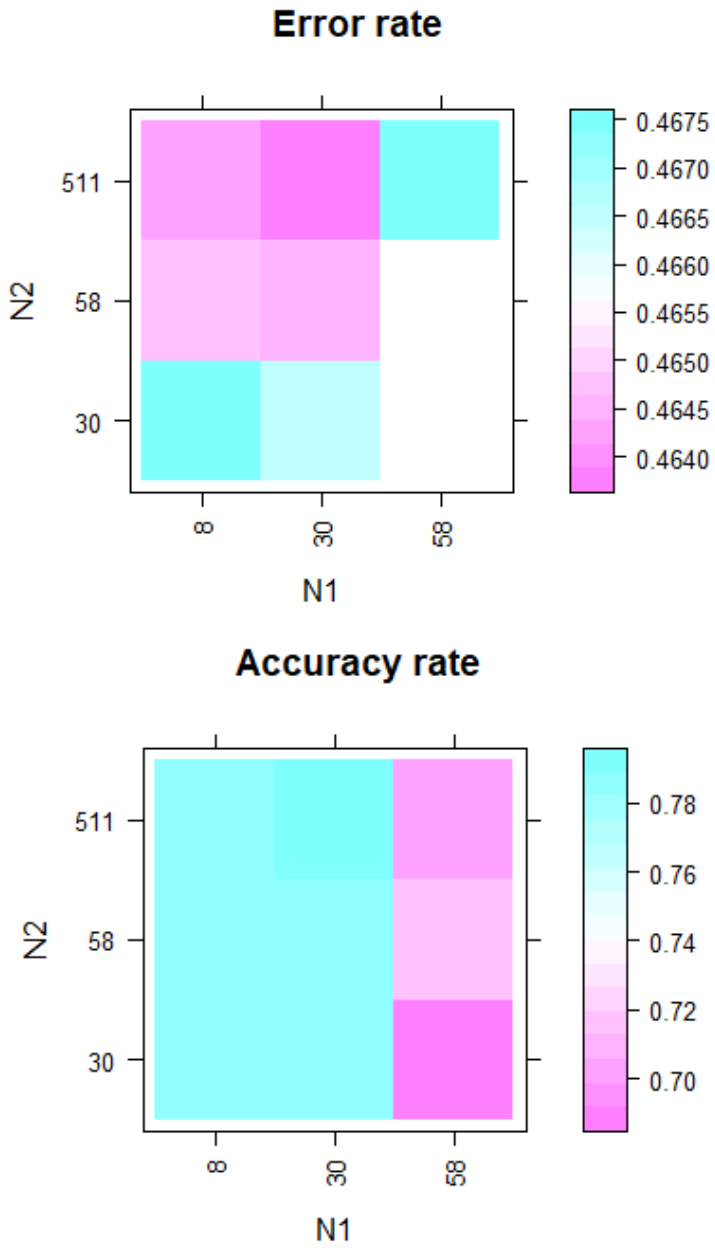


Figure 40: Accuracy TLFN. Cross Validation (k=10). No balanced dataset. Source: Own Elaboration

In conclusion, I will create the TLFN with 30 nodes in the first layer and 511 nodes in the second layer.

A summary table is as follows:

Table 28: Parameters WOE NN

Output Layer	Sigmoid
Epochs	30
Loss	Binary_crossentropy
Batch Size	100
init	uniform
Hidden Layers	ReLU
Nodes	(30,511)

4.5.1.1 Results

The results for the implementation of the Two-Layer Feedforward NN with the WOE values are:

Balanced Data

Table 29: Results - Balanced Data - WOE values

Performance	Values
AUC	0.75296
Accuracy	0.67746
Sensitivity	0.72111
Specificity	0.63382

Unalanced Data

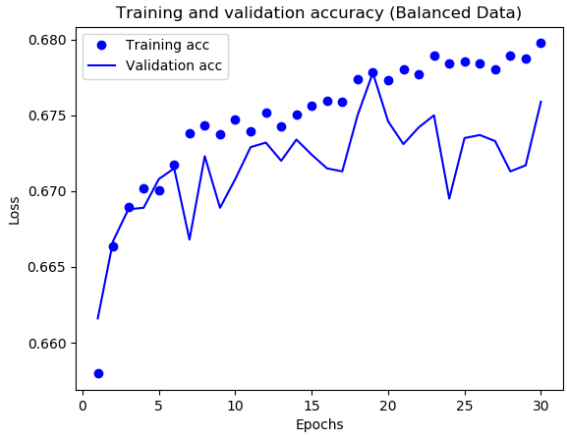
Table 30: Results - Unbalanced Data - WOE values

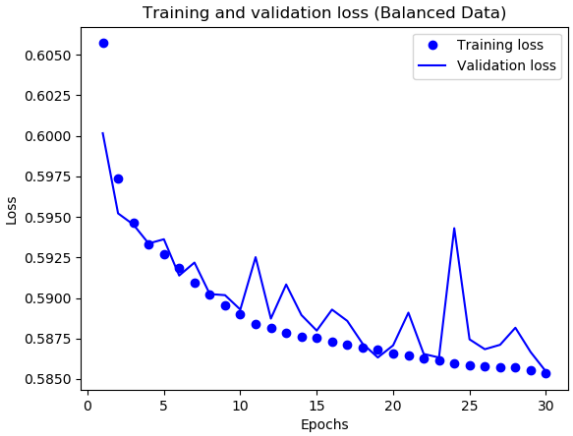
Performance	Values
AUC	0.75484
Accuracy	0.78779
Sensitivity	0.96260
Specificity	0.21123

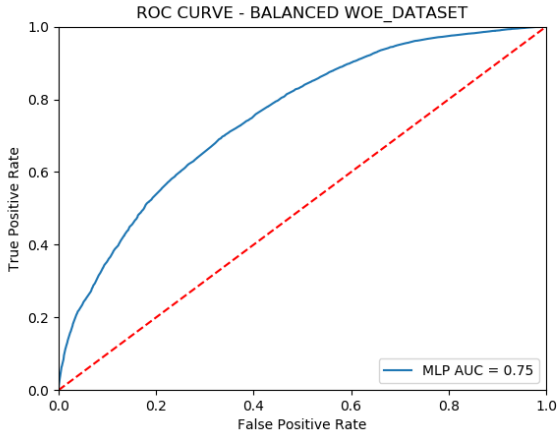
4.5.1.2 Performance

To study the performance of the NN I have separated 10.000 data from the training set and I have made the respective graphs of the loss function, the accuracy and the ROC curve for both the unbalanced and balanced data. The results are:

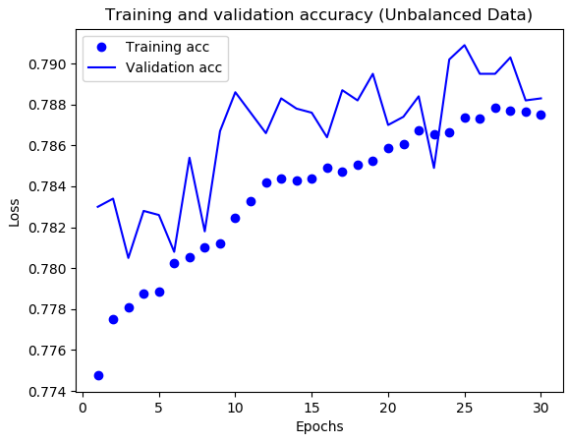
Balanced Data

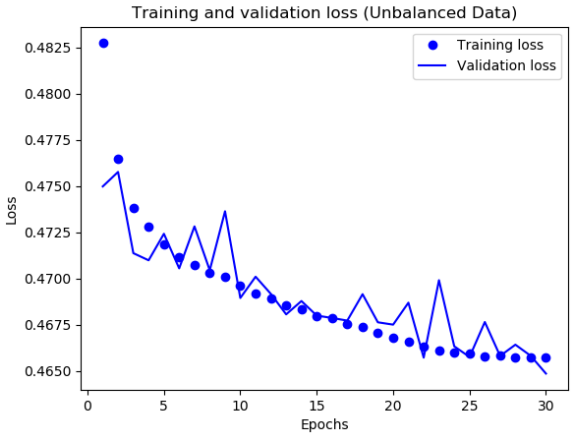


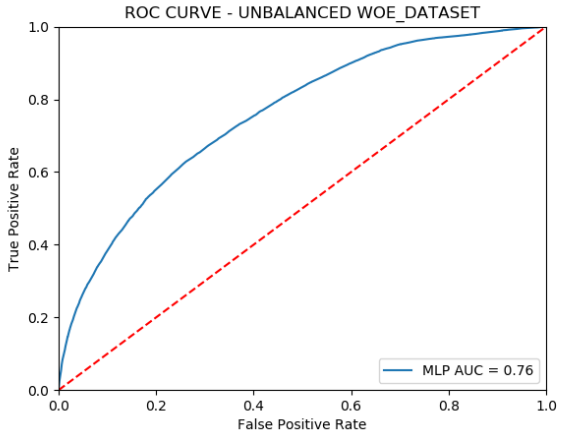




Unbalanced data







Interpretation

When running a NN with gradient descent optimization the quantity you are trying to minimize should be less with every iteration.

I can observe that the training loss decreases with every epoch, and so does the validation loss. Moreover, the training accuracy increases with every epoch, and so does the validation accuracy. That means my model performs well in the training and validation sets. Hence, it is a good model.

4.5.2 FeedForward NN with original values

In this part I will also implement a Two-Layer Feedforward NN but with the original values of my dataset. Therefore, I will have to create dummy variables, for all those variables which have more than two categories.

In the implementation of the dataset with the original values, I will also do a 10-fold cross validation. this time I will test 4 possible nodes, which I have the intuition that could fit well. These are: 8, 15, 30, 58. I got the intuition that these nodes could fit well, because I have tested the Feed Forward Neural Network several times with different nodes.

Dummy Variables

- State
- Bankstate
- UrbanRural

To create the dummy variables I will use the LabelEncoder and OneHotEncoder function from the ScikitLearn library of python. In the next lines I explain what I have done.

The idea is to make the analysis simpler by encoding these string variables. State and BankState has string labels such as “west, midwest, north.east and south”, while RevLineCr and LowDoc have “Y, N”. I will encode these strings into numeric, with the ScikitLearn which is strongest machine learning library in python.

I will use LabelEncoder. It works like whenever we pass a variable to this function, this function will automatically encode different labels in that column with values between 0 to n_classes-1.

So the values will be replaces such as: west with 0, midwest 1, north.east 2 and south 3 But the problem is that south is not higher than west, so we need to create a dummy variable for State.

The OneHotEncoder function creates automatically the dummies.

I actually like how Jason Brownlee defines the importance of using OneHotEncoding in his publication: *A one hot encoding allows the representation of categorical data to be more expressive.* (Brownlee, Jason 2017). I am just going to cite him:

> *Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical.*
>
> *We could use an integer encoding directly, rescaled where needed. This may work for problems where there is a natural ordinal relationship between the categories, and in turn the integer values, such as labels for temperature 'cold', warm', and 'hot'.*
>
> *There may be problems when there is no ordinal relationship and allowing the representation to lean on any such relationship might be damaging to learning to solve the problem. An example might be the labels 'dog' and 'cat'*
>
> *In these cases, we would like to give the network more expressive power to learn a probability-like number for each possible label value. This can help in both making the problem easier for the network to model. When a one hot encoding is used for the output variable, it may offer a more nuanced set of predictions than a single label.*

The resulting graph of 10-Fold cross validation with my original dataset is:

original VALUES: Balanced data

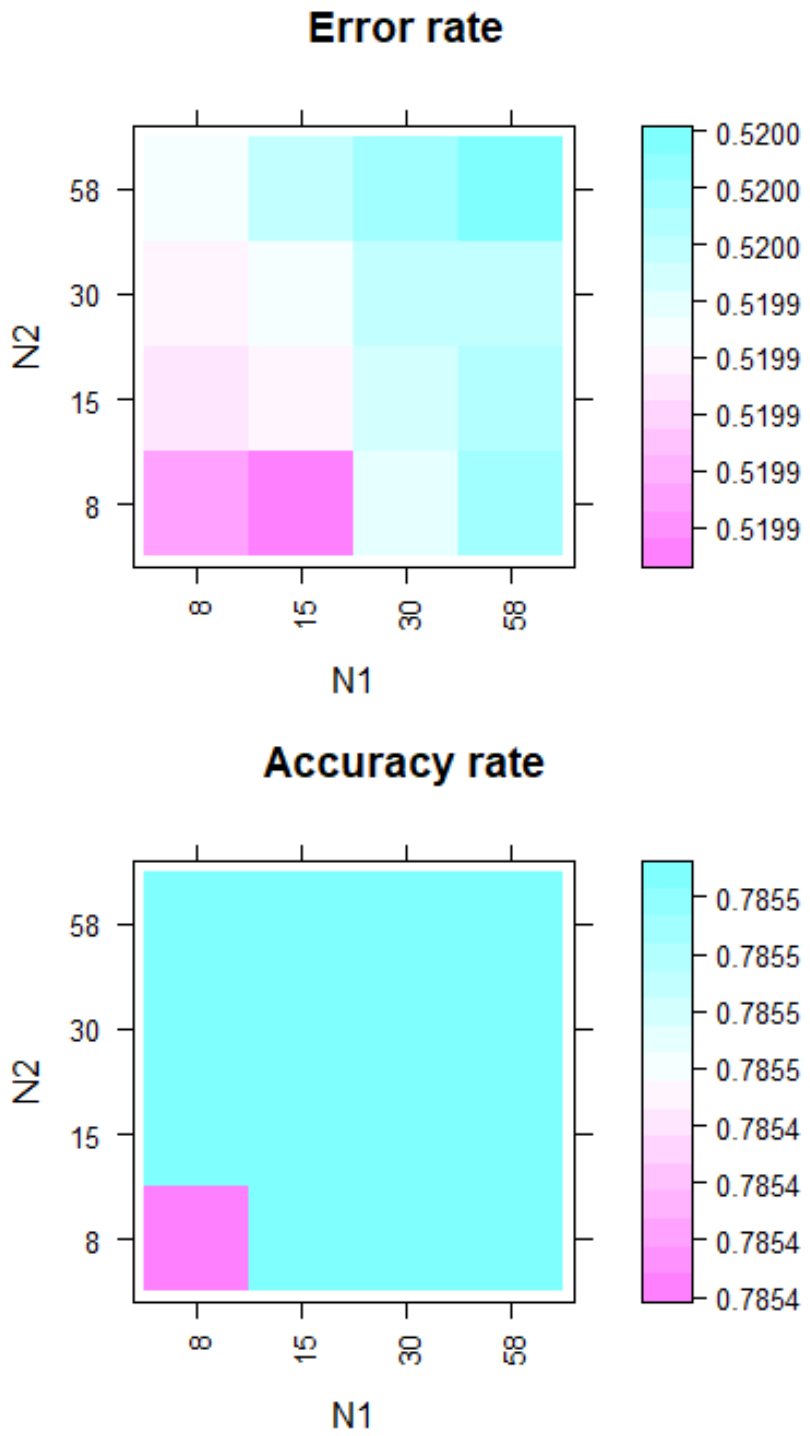


Figure 41: Accuracy TLFN. Cross Validation (k=10). Balanced dataset. Source: Own Elaboration

original VALUES: Unbalanced data

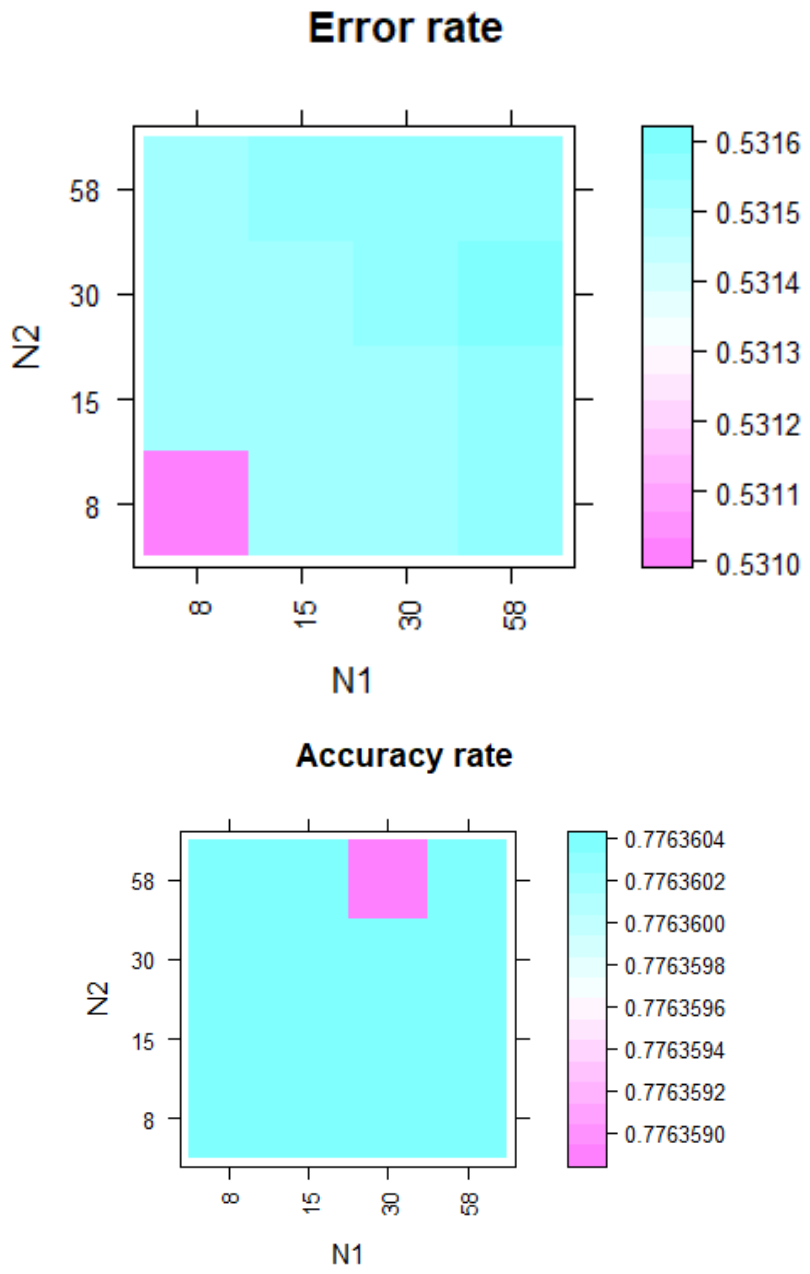


Figure 42: Accuracy TLFN. Cross Validation (k=10). No balanced dataset. Source: Own Elaboration

As it can be observed, the pair of layers which have the lowest error rate are 8 and 8. Moreover, the accuracy of all the combinations is barely the same. Therefore, in this case, I will just use the error rate criteria. In conclusion, I will create the TLFN with 8 nodes in the first layer and 8 nodes in the second layer.

A summary table is as follows:

Table 31: Parameters original NN

Output Layer	Sigmoid
Epochs	30
Loss	Binary_crossentropy
Batch Size	100
init	uniform
Hidden Layers	ReLU
Nodes	(8,8)

4.5.2.1 Results

The results for the implementation of the Two-Layer Feedforward NN with the original values are:

Balanced Data

Table 32: Results - Balanced Data - original values

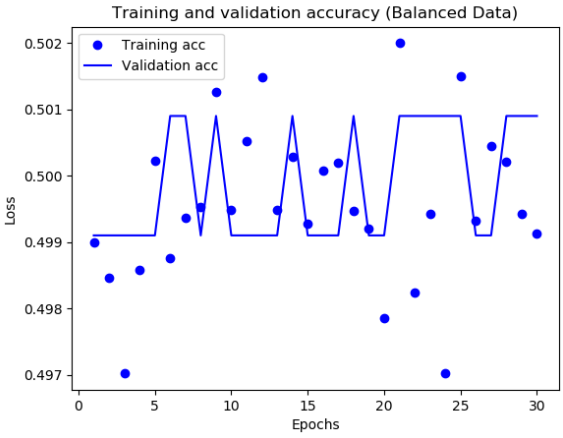
Performance	Values
AUC	0.5
Accuracy	0.50260
Sensitivity	0
Specificity	1

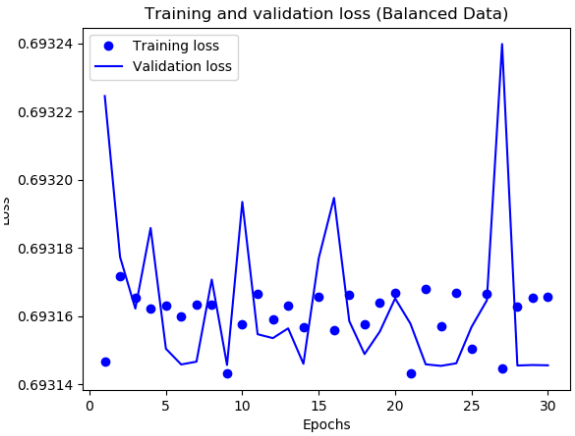
The results of the **Unbalanced Data** are kind of the same bad as the balanced.

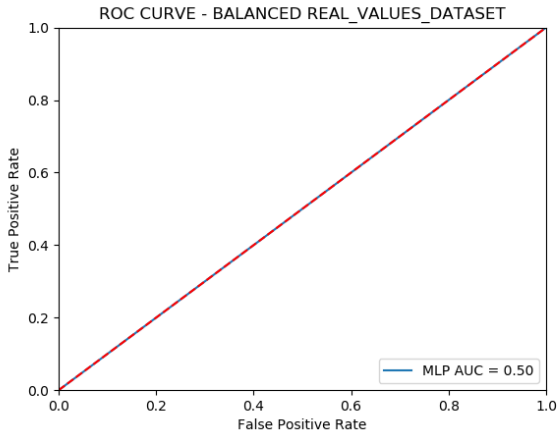
4.5.2.2 Performance

To study the performance of the NN I have separated 10.000 data from the training set and I have made the respective graphs of the loss function, the accuracy and the ROC curve for both the unbalanced and balanced data. I have also created dummies variables (refer to the anex to see the pyhton code). The results are:

Balanced Data







Same happens with the **Unbalanced data.**

Interpretation

There is a clear problem in the Two-Layer Feed Forward Neural Network with the original Values. We can see that in the training and validation plot of the loss of the balanced data, the loss of the training data decreases until the epoch number ten. At the epoch number ten, the loss starts to increase again. Therefore, a good measure will be stop the training process at epoch 10. This is actually one way to control the overfitting of a neural network.

4.5.3 20-Hidden Layer Feedforward Implementation

My whole thesis has been about deep learning. Some people would say that a Two-Layer Feedforward Neural Network it is not exactly a deep learning algorithm. That is why I have implemented a 20-Hidden Layer Feedforward Neural Network with the WOE values.

I will use the same nodes that I applied in the implementation of the TLFN: 30 and 511. Concretly, the first hidden layer will have 30, and the rest 511.

The results are as follows:

4.5.3.1 Results

The results for the implementation of the 20-Hidden Layer Feedforward Neural Network with the WOE values are:

Balanced Data

Table 33: Results - 20 Hidden - Balanced Data - WOE values

Performance	Values
AUC	0.75487
Accuracy	0.67721
Sensitivity	0.79619
Specificity	0.55836

Unalanced Data

Table 34: Results - 20 Hidden - Unbalanced Data - WOE values

Performance	Values
AUC	0.75198
Accuracy	0.78873
Sensitivity	0.96488
Specificity	0.20887

Interpretation

As we can see, the results are quite the same as the TLFN. Therefore, as expected, it was not necessary to add that much hidden layers.

4.6 Improvements of the original dataset: Early stopping and dropout

In this point I will apply some techniques to improve the TLFN implemented before. I will apply these two techinques to the balanced dataset. I will only apply them to the balanced dataset, because the aim of this point is just to demonstrate that those measures work. The dropout rate will be set to 10%, meaning one in 1 inputs will be randomly excluded from each update cycle.

A summary table of the implementation is as follows:

Early stopping in Balanced original Data

Table 35: Early stopping in Balanced original Data

Output Layer	Sigmoid
Epochs	10
Loss	Binary_crossentropy
Batch Size	100
init	uniform
Hidden Layers	ReLU
Nodes	(8,8)

Dropout in Balanced original Data

Table 36: Dropout in Balanced original Data

Output Layer	Sigmoid
Epochs	30
Loss	Binary_crossentropy
Batch Size	100
init	uniform
Hidden Layers	ReLU + dropout (p = 0.1)
Nodes	(8,8)

4.6.0.1 Results

The results for the implementation of the Two-Layer Feedforward NN with the original values are:

Early stopping in Balanced original Data

Table 37: Results Early Stopping (epoch 11)

Performance	Values
AUC	0.56763
Accuracy	0.55342
Sensitivity	0.45122
Specificity	0.65581

Dropout in Balanced original Data

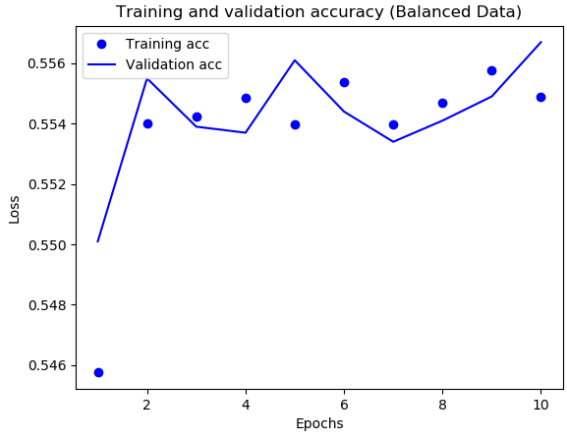
Table 38: Results Dropout

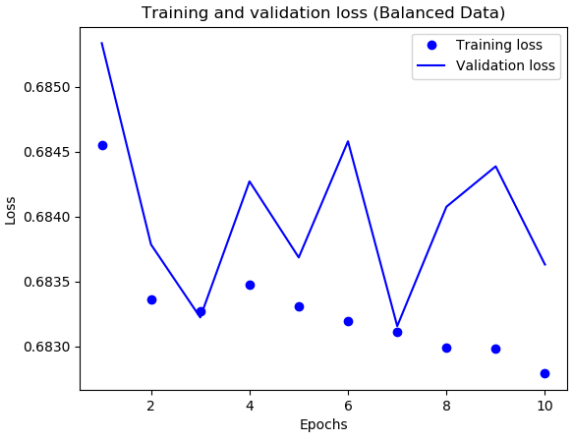
Performance	Values
AUC	0.56919
Accuracy	0.55529
Sensitivity	0.30027
Specificity	0.80523

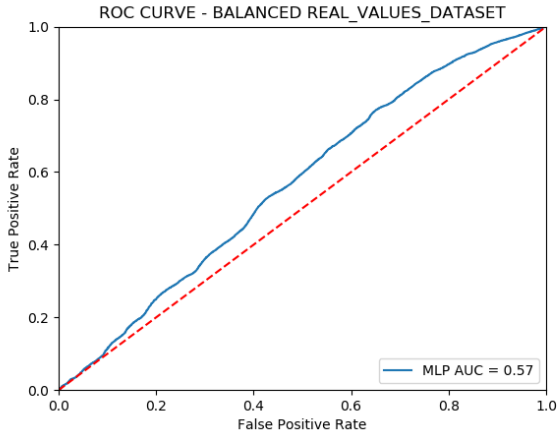
4.6.0.2 Performance (Graphics)

I have implemented the same structure of the neural network, but applying two measures to control the problems I found before. These measures are: early stopping and drop out. These both are explained in the chapter three of the present thesis.

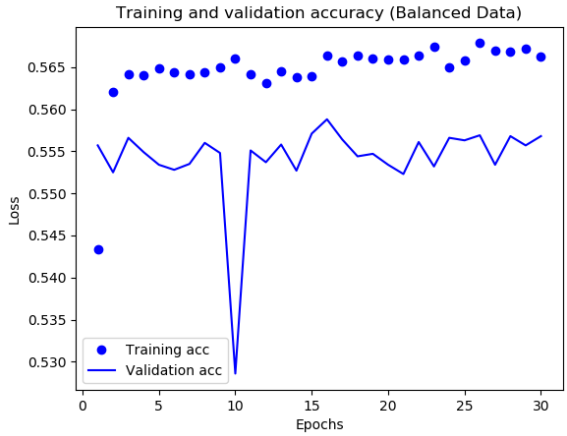
Early stopping in Balanced original Data

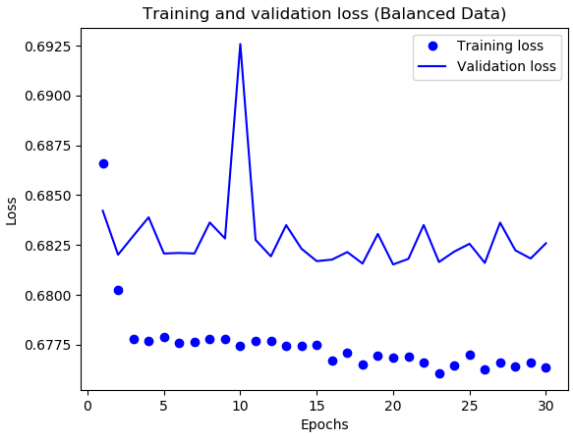


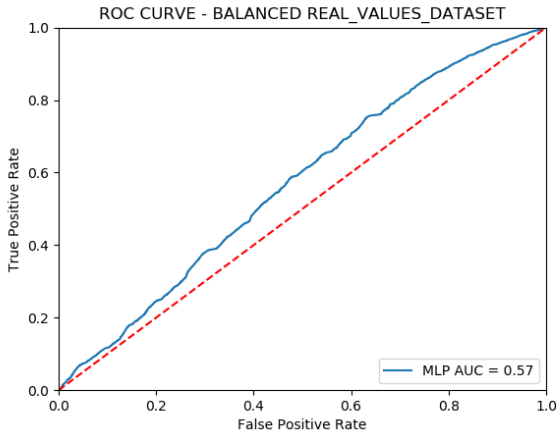




Dropout in Balanced original Data







Interpretation

Once applied the early stopping and drop out measures, we can see that the loss is descreasing with each epoch, the accuracy increasing with each epoch, and that the sensitivity and specificity measures are more equilibrated. Moreover, the accuracy and AUC have improved.

5 Conclusions

This section aims to detail the most relevant aspects of the thesis, as well as to give greater emphasis to the most important points. In addition, interesting topics for future research will be proposed.

Due to the many different aspects that have been studied in the thesis, this point will be divided into several themes: the importance of credit risk management; a comparison of the results obtained through logistic regression and the neural network; the general complexity in the preparation of the thesis, the time spent, the software and materials used, and other aspects that are considered important to emphasize; and, finally, the fulfillment of expectations and the knowledge acquired. In addition, some points for future research will be presented.

Credit risk management is one of the most crucial and important issues that banks face today. Banking regulation requires institutions to have a minimum of capital and reserves, depending on their level of risk. The idea of regulation is that, in the event of an economic crisis, it should be supported by the capital of the institution and not by the depositors. This leads these institutions to develop internal models to better manage their risks and play strategically. Therefore, this "struggle" between regulation and banking strategies makes a good risk management a differentiating element in the market. In short, good risk management is a key differentiating element in a bank's strategy.

At the same time, institutions also use internal models for the decision to grant credit to a client. Most entities use traditional statistical models, such as logistic regression. From this regression they construct a scorecard with the optimal characteristics to approve or deny a credit.

It is precisely at this point that most of this thesis comes. Due to the great growth that artificial intelligence techniques are having, I have decided to study the implementation of a neural network with the aim of predicting whether a credit will be or will not be default. I have compared a linear regression model with a Two-Layer Feed Forward Neural Network (TLFN) model.

To put it in context with the economic reality, it must be said that these more advanced forecasting techniques have not yet been incorporated into the credit risk management of banks. This is because entities are very traditional in nature, and these techniques cannot be explained very well. For example, to be more precise at this point, it is not currently known with certainty what weights the neural network assigns to each variable to predict the output. Therefore, we do not know the importance, nor the interpretation of the variables of the model. However, it must be said that this information is not indispensable to know whether or not to grant credit.

The following table shows the comparison between logistic regression and the TLFN for the WOE values, and the original values of the database used. An implementation of a 20-layer neural network is also done. For a complete comparison, I use the AUC, Accuracy, Sensitivity and Specificity performance statistics.

Table 39: Logit Model vs TLFN

	Logit Model		**TLFN**		**20LFN**
Statistics	**WOE**	**original Values**	**WOE**	**original Values**	**WOE**
AUC	0.7170	0.6782	0.7529	0.5	0.7548
Accuracy	0.6495	0.6338	0.6774	0.5	0.6772
Sensitivity	0.5330	0.5360	0.7211	0	0.7961
Specificity	0.7630	0.7620	0.6338	1	0.5584

An interesting first observation is that the use of WOE values, both in logistic regression and in the TLFN, performs better than the use of the original values. These results confirm that the decision of using the WOE values in the default prediction is a success.

Likewise, it is observed how the use of the TLFN in the WOE values has better results than the implementation of these values in the logistic model. Although these better results are not very significant, these go in accordance with the growing importance that these more advanced techniques are having nowadays.

Another observation is the poor results of the application of the TLFN in the original values of the database. In their respective point, I comment that it is due to the overfitting problem. In this line, the results that come out after applying an improvement, through regularization techniques: *early stopping* and *dropout*, are the following:

Table 40: TLFN original Values Improvements

TLFN original Values Improvements		
Statistics	Dropout (p = 0.1)	Early Stopping
AUC	0.5692	0.5676
Accuracy	0.5553	0.5534
Sensitivity	0.3002	0.4512
Specificity	0.8052	0.6559

The results are indeed improving. However, in this case, they are no better than those implemented with the logistic regression.

Therefore, I can conclude that the TLFN is better than the logistic regression only if we use WOE values. Otherwise, if we use original values, it is better to go for a logistic model.

With the idea of being consistent with the thesis, as I have dedicated most of it to the explanation of Deep Learning techniques and, because there are professionals who believe that two layers are not enough to call a neural network a Deep Learning network, I have implemented a 20-Layer Feed Forward Neural Network with WOE values. The results are very similar to those of the TLFN. Therefore, in this case, it would not be necessary to implement more than two layers, since the additional computational cost would be unnecessary.

As a third conclusion I would like to emphasize the general complexity in the elaboration of the thesis, the time spent, the software and the materials used.

The idea for the thesis arose from the growing importance of artificial intelligence techniques today. However, during the double degree in economics and statistics these techniques have not been studied. Therefore, in the elaboration of this thesis, I have been able to learn new advanced statistical concepts, which are actually very important for the reality we live today.

These include all the theoretical concepts learned in Deep Learning and, specifically, neural networks. Along these lines, I would also like to highlight the software I have used for the practical application. I have used the software Python. Pyhton is a software widely used in the field of artificial intelligence. I have been able to learn this software and apply the *Tensorflow* and *Keras* libraries, mainly. At

the same time, I've also learned a lot about R markdown, since all the thesis is done on this R platform.

Obviously, the whole process of learning these subjects is not an easy task and it has taken me a lot of time, effort and dedication.

Finally, it is necessary to comment that the objectives and expectations defined at the beginning of the thesis have been fulfilled.

5.1 Thesis Extension

These are some of the proposals that could be of interest in future research:

Firstly, it is proposed to do the same analysis, but using other prediction techniques, such as Random Forest or Support Vector Machine (SVM). The idea is to make a comparison between these techniques and the TLFN. Therefore, the question is: Which is the best algorithm in predicting credit default?

Secondly, it is proposed to use a database focused on the individual client. The idea would be to study the differences in the results from a database with characteristics from individuals, rather than from SMEs.

Finally, and as a more ambitious proposal, it is proposed to find a method to extract the coefficients (weights) of the variables used by the Neural Network, in order to elaborate a scorecard.

6 Conclusiones

El presente punto pretende detallar los aspectos más relevantes de la tesis, así como dar un mayor énfasis a los puntos más importantes. Además, se propondrán temas de investigación futura que se consideren interesantes.

Dado que en la tesis se han estudiado muchos aspectos distintos, este punto estará dividido en varios temas: la importancia de la gestión del riesgo de crédito; una comparativa de los resultados obtenidos mediante una regresión logística y la red neuronal; la complejidad general en la elaboración de la tesis, el tiempo dedicado, el software y los materiales utilizados y otros aspectos que se consideren importantes de recalcar; y, por último, el cumplimiento de las expectativas y objetivos propuestos antes de empezar, así como el conocimiento adquirido. Adicionalmente, se expondrán algunos puntos de futura investigación.

La gestión del riesgo de crédito es uno de los aspectos más cruciales e importantes que afrontan las entidades bancarias hoy en día. La regulación bancaria exige que las entidades tengan un mínimo de capital y reservas, según su nivel de riesgo. La idea de la regulación es que, en el supuesto caso de una crisis económica, ésta sea soportada por el capital de la entidad y no por los depositantes. Esto hace que dichas entidades elaboren modelos internos para gestionar mejor sus riesgos y jugar estratégicamente. Por tanto, esta "lucha" entre la regulación y las estrategias bancarias lleva a que una buena gestión del riesgo acabe siendo un elemento diferenciador en el mercado.

En definitiva, una buena gestión del riesgo es un elemento clave y diferenciador en la estrategia de una entidad bancaria. Paralelamente, las entidades también utilizan modelos internos para la decisión en la concesión de un crédito a un cliente. La mayoría de las entidades utilizan modelos estadísticos tradicionales, tales como la regresión logística. A partir de esta regresión construyen una *scorecard* con las características óptimas para aprobar o denegar un crédito. Es precisamente en este punto donde recae la mayor parte de la presente tesis. Debido al gran crecimiento que están teniendo las técnicas de inteligencia artificial, he decidido estudiar la implementación de una red neuronal con el objetivo de predecir si un crédito será, o no será default. He comparado un modelo de regresión lineal, con un modelo de red neuronal (Two-Layer Feed Forward Neural Network, TLFN).

Para ponerlo en contexto con la realidad económica, cabe decir, que estas técnicas de predicción más avanzadas, no están -todavía- incorporadas en la gestión del riesgo de crédito de las entidades bancarias. Esto es debido a que las entidades son de carácter muy tradicional, y dichas técnicas no acaban de poderse explicar muy bien. Por ejemplo, para ser más preciso en este punto, actualmente no se sabe con certeza qué pesos asigna la red neuronal a cada variable para predecir el output. Por tanto, no se sabe la importancia, ni la interpretación de las variables del modelo. Sin embargo, cabe decir que esta información no es indispensable para saber si otorgar o no otorgar un crédito.

La siguiente tabla muestra la comparativa entre la regresión logística y la TLFN para los valores WOE y los valores reales de la base de datos utilizada. Así mismo, también se muestra una implementación de una red neuronal con 20 capas. Para una comparación completa, utilizo los estadísticos de rendimiento AUC, *Accuracy*, *Sensitivity* y *Specificity*.

Table 41: Logit Model vs TLFN

	Logit Model		**TLFN**		**20LFN**
Statistics	**WOE**	**original Values**	**WOE**	**original Values**	**WOE**
AUC	0.7170	0.6782	0.7529	0.5	0.7548
Accuracy	0.6495	0.6338	0.6774	0.5	0.6772
Sensitivity	0.5330	0.5360	0.7211	0	0.7961
Specificity	0.7630	0.7620	0.6338	1	0.5584

Una primera observación interesante recae en que la utilización de valores WOE, tanto en la regresión logística como en la TLFN, tiene mejor rendimiento que la utilización de los valores reales. Esto confirma que la utilización de los valores WOE en la predicción de default, en la base de datos que he utilizado, es un acierto.

Así mismo, se observa como la utilización de la TLFN en valores WOE tiene mejores resultados que la implementación de estos en un modelo logístico. Aunque la mejora no es muy significante, cabe decir que es un gran resultado muy interesante, que es coherente con la creciente importancia que estas técnicas más avanzadas están teniendo.

Otra de las observaciones son los malos resultados de la aplicación de la TLFN en los valores reales de la base de datos. En su respectivo punto comento que es debido al *overfitting*. En esta línea los resultados que salen después de aplicar una mejora, a través de técnicas de regularización: *early stopping* y *dropout*, son los siguientes:

Table 42: TLFN original Values Improvements

TLFN original Values Improvements		
Statistics	Dropout (p = 0.1)	Early Stopping
AUC	0.5692	0.5676
Accuracy	0.5553	0.5534
Sensitivity	0.3002	0.4512
Specificity	0.8052	0.6559

Los resultados efectivamente mejoran. Sin embargo, en este caso, no son mejores que los implementados con una regresión logística.

Por tanto, a modo resumen, puedo concluir que la TLFN es mejor si se aplican valores WOE; y, por otro lado, en caso de que se apliquen valores reales, es mejor una regresión logística.

Con la idea de ser consistente con la tesis, como he dedicado la mayor parte de la misma a la explicación de las técnicas de Deep Learning y, debido a que hay profesionales que opinan que dos capas no son suficientes para denominar una red neuronal como una red de Deep Learning; he implementado una 20-Layer Feed Forward Neural Network con los valores WOE. Los resultados son muy similares a los que salían con las TLFN. Por tanto, en este caso, no sería necesario implementar más de dos capas, ya que el coste computacional adicional sería innecesario.

Como tercera conclusión me gustaría recalcar la complejidad general en la elaboración de la tesis, el tiempo dedicado, el software y los materiales utilizados.

La idea de la tesis surgió a raíz de la creciente importancia que están teniendo las técnicas de inteligencia artificial hoy en día. Sin embargo, durante el doble grado en economía y estadística no se han estudiado dichas técnicas. Por tanto, en la elaboración de esta tesis he podido aprender conceptos estadísticos avanzados nuevos muy importantes para la realidad que vivimos actualmente.

Entre ellos, cabe destacar todos los conceptos teóricos aprendidos en materia de Deep Learning y, concretamente, de redes neuronales. En esta línea, también recalcar el software que he utilizado para la aplicación práctica, Python. Pyhton es un software muy utilizado en el campo de la inteligencia artificial. He podido aprender este software y aplicar las librerías *Tensorflow* y *Keras*, principalmente. Paralelamente, también he aprendido mucho en materia de R markdown, ya que toda la tesis está hecha sobre esta plataforma de R.

Evidentemente, todo el proceso de aprendizaje de estos temas no es una tarea fácil y me ha conllevado mucho tiempo, esfuerzo y dedicación.

Por último, es preciso comentar que se han cumplido los objetivos y expectativas definidos al inicio de la tesis.

6.1 Extensión de la tesis

Este punto pretende motivar futuras investigaciones sobre el campo estudiado. En concreto, se proponen varias medidas para ampliar dicha tesis.

Primeramente, se propone hacer el mismo análisis, pero utilizando otras técnicas de predicción como Random Forest o Support Vector Machine (SVM). La idea es hacer una comparativa con la TLFN y estudiar los resultados.

En segundo lugar, se propone hacer el mismo análisis con una base de datos enfocada al cliente particular. La idea sería ver las diferencias en los resultados que ésta tiene, con la utilizada en la actual tesis, sobre Pymes.

Por último, y como propuesta más ambiciosa, se propone encontrar un método para extraer los coeficientes (pesos) de las variables que otorga la TLFN, con tal de elaborar una scorecard.

7 Index of Figures and Tables

7.1 Figures

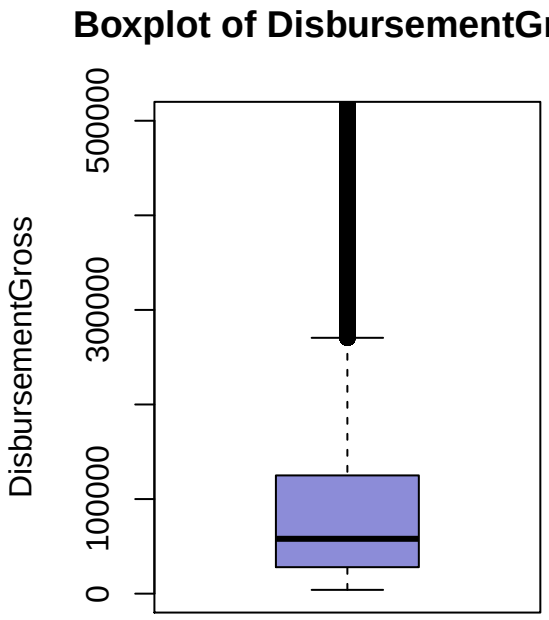
7.2 Tables

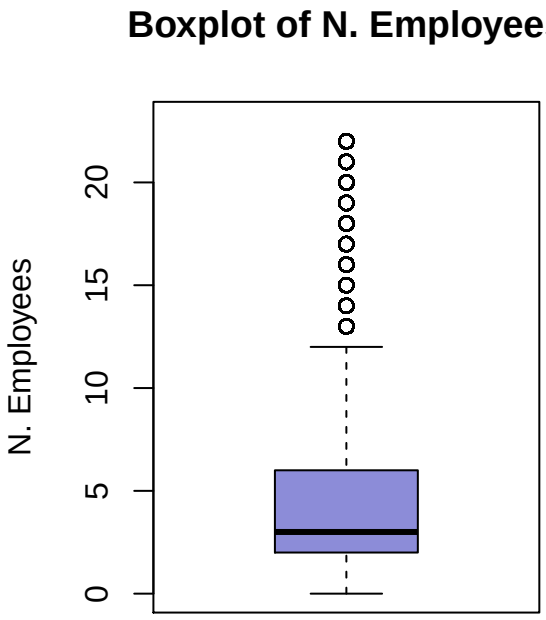
8 Annexes

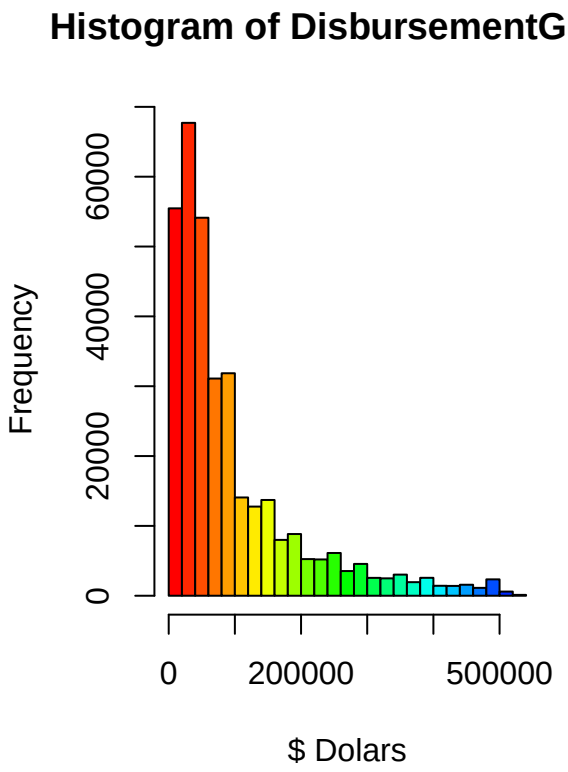
8.1 Plots

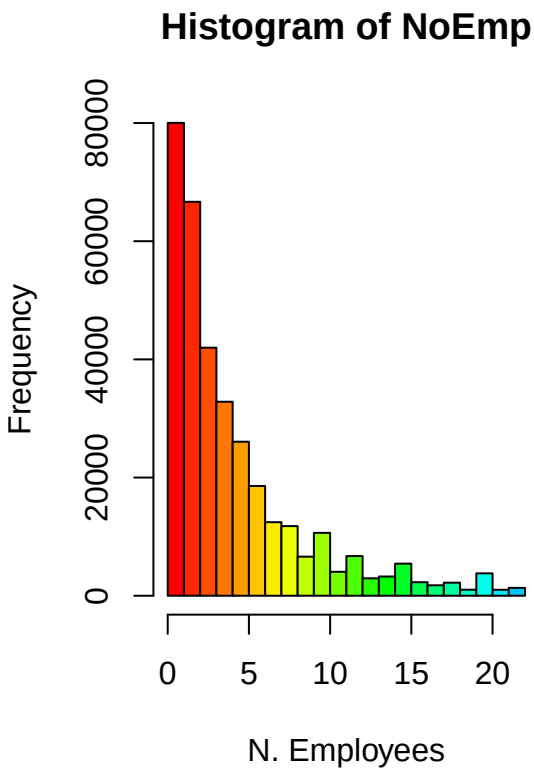
Descriptive Analysis Dataset

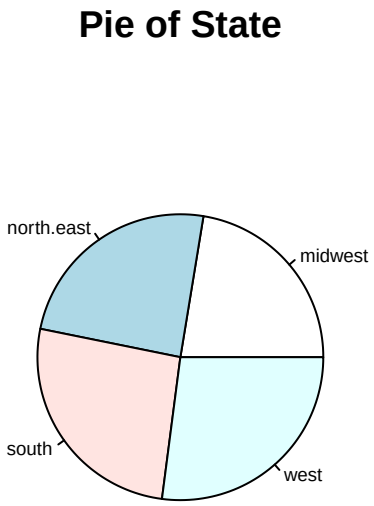
```
##        ID                    State            BankState
##  Min.   :1000014003   midwest   :76986   midwest   :107125
##  1st Qu.:2129120002   north.east:83808   north.east: 67111
##  Median :3488703004   south     :89741   south     :115543
##  Mean   :4219252838   west      :92932   west      : 53688
##  3rd Qu.:6136084003
##  Max.   :9995603000
##      NAICS           NoEmp             New          UrbanRural  RevLineCr
##  Min.   :11.0   Min.   : 0.00   Min.   :0.000   Min.   :0.0   N:185906
##  1st Qu.:42.0   1st Qu.: 2.00   1st Qu.:0.000   1st Qu.:1.0   Y:157561
##  Median :48.0   Median : 3.00   Median :0.000   Median :1.0
##  Mean   :49.9   Mean   : 4.65   Mean   :0.266   Mean   :0.9
##  3rd Qu.:62.0   3rd Qu.: 6.00   3rd Qu.:1.000   3rd Qu.:1.0
##  Max.   :92.0   Max.   :22.00   Max.   :1.000   Max.   :2.0
##  LowDoc     DisbursementGross    Default        Recession
##  N:318916   Min.   :  4000    Min.   :0.000   Min.   :0.0000
##  Y: 24551   1st Qu.: 28000    1st Qu.:0.000   1st Qu.:0.0000
##             Median : 58000    Median :0.000   Median :0.0000
##             Mean   : 97683    Mean   :0.232   Mean   :0.0811
##             3rd Qu.:125000    3rd Qu.:0.000   3rd Qu.:0.0000
##             Max.   :535590    Max.   :1.000   Max.   :1.0000
```

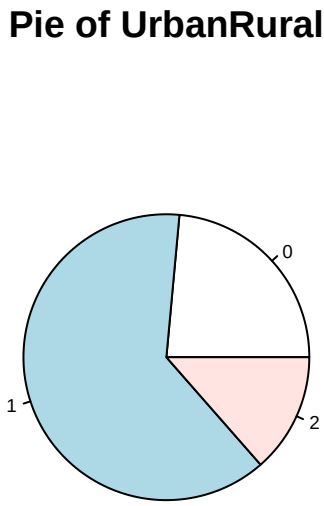

Boxplot of DisbursementGross
DisbursementGross
0
100000
300000
500000

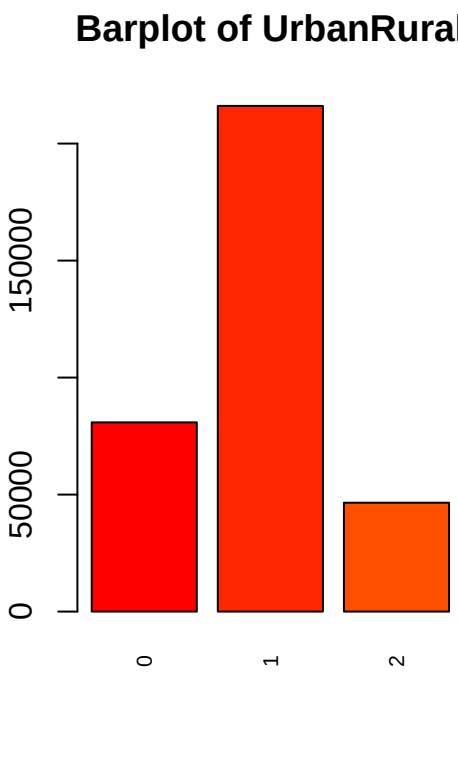

Boxplot of N. Employees
N. Employees
0
5
10
15
20

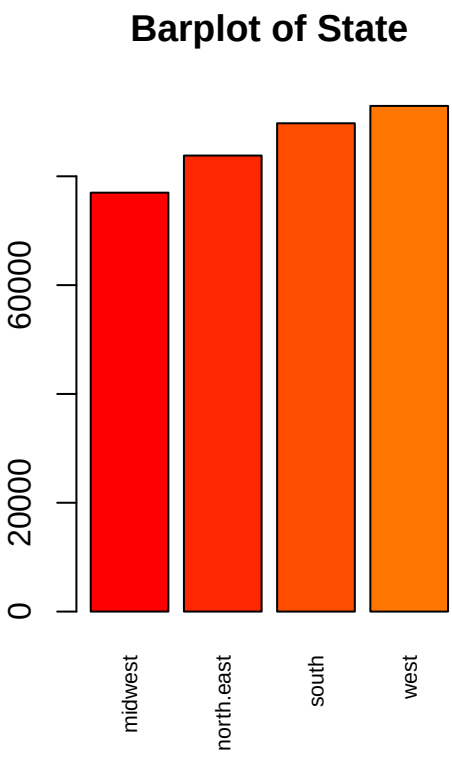

Histogram of DisbursementGros
Frequency
0
20000
40000
60000
0
200000
500000
$ Dolars

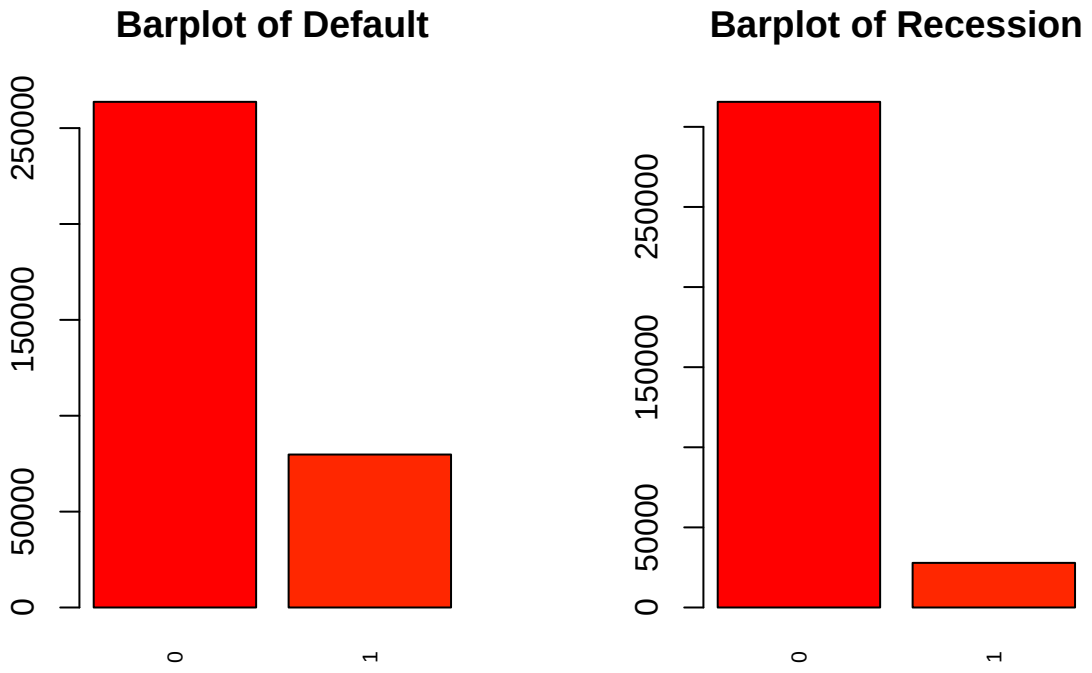

Histogram of NoEmp
Frequency
0
20000
40000
60000
80000
0
5
10
15
20
N. Employees


Pie of State
north.east
midwest
south
west


Pie of UrbanRural
0
1
2


Barplot of UrbanRural
0
50000
150000
0
1
2


Barplot of State
0
20000
60000
midwest
north.east
south
west




```
## [1] "Summary of Defaults in my dataset"
```

```
##      0      1
## 263699  79768
```

Package Scorecard

The scorecard package makes the development of credit risk scorecard easier and efficient by providing functions for some common tasks, such as data partition, variable selection, woe binning, scorecard scaling, performance evaluation and report generation. These functions can also used in the development of machine learning models.

8.2 Instalation Python and Keras

As mentioned in the present thesis I have used Keras for Python as the library to create and implement the neural network.

Pyhton is an object-oriented programming software. This object-oriented concept refers to the fact that the language is designed so that programmers can work with fields, or attributes, and it is easier for them to implement them in problem solving.

Python has not been used in the career at any time; however, I had already researched on my own how to use it, and had even applied it in original cases. Installing Pyhton is relatively simple.

First, download the software directly from the website (https://www.python.org/). At the time of writing this thesis, the most recent version is 3.7.3.

Once the software is installed, you need an environment to start using it. For begginers, I recommend installing Thonny (https://www.thonny.org/). In fact, with the installation of Thonny, the pyhton software is also installed. Thonny is a very simple and intuitive environment. For the more experienced, I recommend to download PyCharm. This interpreter is much more useful and has more functionalities. This is the one I used in this thesis.

There is a lot of information on the internet about libraries and basic packages to start using Pyhton (like numpy, pandas, etc). I invite the reader to look for them and be informed.

Once the basic packages are installed, tensorflow can be installed. Tensorflow is a Pyhton library widely used in Machine Learning problems, as it makes numerical computation much faster and easier to apply. I also invite the reader to look for more information on the Internet on the subject.

To install tensorflow, one of the ways is to type the following command into the system CPU: *pip install tensorflow*.

The next step is to install Keras. Keras is built from Tensorflow. Keras is very useful when you want to apply simple neural networks quickly and simply. Keras has two modules, Model and Sequential, which allow many implementations. For example, a quick model in Keras would be:

```
model = Sequential
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy',metrics=['accuracy'])
# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))
#Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

As the network that I implement in this thesis is very simple I use Keras. However, if I had implemented a more complex network, maybe I would use Tensforflow directly.

To install Keras, one of the ways is to type the following command in the system CPU: pip install keras

8.3 Code: Python

Cross Validation WOE values - Code

```
import os
os.chdir("C:\\Users\\Marcos Rusinol\\Desktop\\dataset")
import pandas as pd
from keras import models
from keras import layers

# Importing the dataset
dataset = pd.read_csv('WOE_datasetpy.csv',sep=';')
print(dataset.shape)
dataset.iloc[:,1:10] = dataset.stack().str.replace(',','.').unstack()
pd.to_numeric(dataset.iloc[:,0], errors='coerce')
```

```
pd.to_numeric(dataset["State_woe"], errors='coerce')
pd.to_numeric(dataset.iloc[:,2], errors='coerce')
pd.to_numeric(dataset.iloc[:,3], errors='coerce')
pd.to_numeric(dataset.iloc[:,4], errors='coerce')
pd.to_numeric(dataset.iloc[:,5], errors='coerce')
pd.to_numeric(dataset.iloc[:,6], errors='coerce')
pd.to_numeric(dataset.iloc[:,7], errors='coerce')
pd.to_numeric(dataset.iloc[:,8], errors='coerce')
pd.to_numeric(dataset.iloc[:,9], errors='coerce')

#BALANCING DATASET
datasetdef = dataset.loc[dataset.Default == 1]
datasetnodef = dataset.loc[dataset.Default == 0]
datasetnodef = datasetnodef.sample(79768)
dataset = pd.concat([datasetdef, datasetnodef])

dataset_part_1 = dataset.iloc[1:15953,:]
dataset_part_2 = dataset.iloc[15954:31906,:]
dataset_part_3 = dataset.iloc[31907:47859,:]
dataset_part_4 = dataset.iloc[47860:63812,:]
dataset_part_5 = dataset.iloc[63813:79765,:]
dataset_part_6 = dataset.iloc[79766:95718,:]
dataset_part_7 = dataset.iloc[95719:111671,:]
dataset_part_8 = dataset.iloc[111672:127624,:]
dataset_part_9 = dataset.iloc[127625:143577,:]
dataset_part_10 = dataset.iloc[143578:159530,:]

k1_train = pd.concat([dataset_part_2,dataset_part_3,dataset_part_4
,dataset_part_5,dataset_part_6,dataset_part_7,dataset_part_8
,dataset_part_9,dataset_part_10])
X1_train = k1_train.iloc[:, 1:10].values
y1_train = k1_train.iloc[:, 0].values
k1_test = dataset_part_1
X1_test = k1_test.iloc[:, 1:10].values
y1_test = k1_test.iloc[:, 0].values

k2_train = pd.concat([dataset_part_1,dataset_part_3,dataset_part_4
,dataset_part_5,dataset_part_6,dataset_part_7,dataset_part_8
,dataset_part_9,dataset_part_10])
X2_train = k2_train.iloc[:, 1:10].values
y2_train = k2_train.iloc[:, 0].values
k2_test = dataset_part_2
X2_test = k2_test.iloc[:, 1:10].values
y2_test = k2_test.iloc[:, 0].values

k3_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_4
,dataset_part_5,dataset_part_6,dataset_part_7,dataset_part_8
,dataset_part_9,dataset_part_10])
```

```
X3_train = k3_train.iloc[:, 1:10].values
y3_train = k3_train.iloc[:, 0].values
k3_test = dataset_part_3
X3_test = k3_test.iloc[:, 1:10].values
y3_test = k3_test.iloc[:, 0].values

k4_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_3,
dataset_part_5,dataset_part_6,dataset_part_7,dataset_part_8,
dataset_part_9,dataset_part_10])
X4_train = k4_train.iloc[:, 1:10].values
y4_train = k4_train.iloc[:, 0].values
k4_test = dataset_part_4
X4_test = k4_test.iloc[:, 1:10].values
y4_test = k4_test.iloc[:, 0].values

k5_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_3
,dataset_part_4,dataset_part_6,dataset_part_7,dataset_part_8
,dataset_part_9,dataset_part_10])
X5_train = k5_train.iloc[:, 1:10].values
y5_train = k5_train.iloc[:, 0].values
k5_test = dataset_part_5
X5_test = k5_test.iloc[:, 1:10].values
y5_test = k5_test.iloc[:, 0].values

k6_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_3
,dataset_part_4,dataset_part_5,dataset_part_7,dataset_part_8
,dataset_part_9,dataset_part_10])
X6_train = k6_train.iloc[:, 1:10].values
y6_train = k6_train.iloc[:, 0].values
k6_test = dataset_part_6
X6_test = k6_test.iloc[:, 1:10].values
y6_test = k6_test.iloc[:, 0].values

k7_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_3
,dataset_part_4,dataset_part_5,dataset_part_6,dataset_part_8
,dataset_part_9,dataset_part_10])
X7_train = k7_train.iloc[:, 1:10].values
y7_train = k7_train.iloc[:, 0].values
k7_test = dataset_part_7
X7_test = k7_test.iloc[:, 1:10].values
y7_test = k7_test.iloc[:, 0].values

k8_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_3
,dataset_part_4,dataset_part_5,dataset_part_6,
dataset_part_7,dataset_part_9,dataset_part_10])
X8_train = k8_train.iloc[:, 1:10].values
y8_train = k8_train.iloc[:, 0].values
k8_test = dataset_part_8
```

```
X8_test = k8_test.iloc[:, 1:10].values
y8_test = k8_test.iloc[:, 0].values

k9_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_3
,dataset_part_4,dataset_part_5,dataset_part_6,
dataset_part_7,dataset_part_8,dataset_part_10])
X9_train = k9_train.iloc[:, 1:10].values
y9_train = k9_train.iloc[:, 0].values
k9_test = dataset_part_9
X9_test = k9_test.iloc[:, 1:10].values
y9_test = k9_test.iloc[:, 0].values

k10_train = pd.concat([dataset_part_1,dataset_part_2,dataset_part_3
,dataset_part_4,dataset_part_5,dataset_part_6,
dataset_part_7,dataset_part_8,dataset_part_9])
X10_train = k10_train.iloc[:, 1:10].values
y10_train = k10_train.iloc[:, 0].values
k10_test = dataset_part_10
X10_test = k10_test.iloc[:, 1:10].values
y10_test = k10_test.iloc[:, 0].values

X_train_cross = [X1_train,X2_train,X3_train,X4_train
,X5_train,X6_train,X7_train,X8_train,X9_train,X10_train]
y_train_cross = [y1_train,y2_train,y3_train,y4_train
,y5_train,y6_train,y7_train,y8_train,y9_train,y10_train]
X_test_cross = [X1_test,X2_test,X3_test,X4_test
,X5_test,X6_test,X7_test,X8_test,X9_test,X10_test]
y_test_cross = [y1_test,y2_test,y3_test,y4_test
,y5_test,y6_test,y7_test,y8_test,y9_test,y10_test]

hidden1 = [8,30,58]
hidden2 = [30,58,511]
nhidden1 = len(hidden1)
nhidden2 = len(hidden2)
ncross = len(X_train_cross)
err = list()
acc = list()

for c in range(ncross):
    set_train_X = X_train_cross[c]
    set_train_y = y_train_cross [c]
    set_test_X = X_test_cross [c]
    set_test_y = y_test_cross[c]
    for i in range(nhidden1):
        N1 = hidden1[i]
        print("N1",N1)
        for j in range(nhidden2):
            N2 = hidden2[j]
```

```
            classifier = models.Sequential()
            classifier.add(layers.Dense(output_dim = 10
            , init = 'uniform', input_dim = 9, activation = 'relu'))
            classifier.add(layers.Dense(output_dim = N1
            , init = 'uniform',activation = 'relu'))
            classifier.add(layers.Dense(output_dim = N2
            , init='uniform', activation = 'relu'))
            classifier.add(layers.Dense(output_dim = 1
            , init = 'uniform', activation = 'sigmoid'))
            classifier.compile(optimizer = 'rmsprop',
            Loss = 'binary_crossentropy', metrics = ['accuracy'])
            history = classifier.fit(set_train_X,
            set_train_y, batch_size = 100, nb_epoch = 30)
            err.append(history.history['loss'][29])
            acc.append(history.history['acc'][29])
print(err)
print(acc)
```

Cross Validation original values - Code

```
import os
os.chdir("C:\\Users\\Marcos Rusinol\\Desktop\\dataset")
import numpy as np
import pandas as pd
from keras import models
from keras import layers

# Importing the dataset
dataset = pd.read_csv('datasetcompletepy.csv',sep=';')

#BALANCING DATASET
datasetdef = dataset.loc[dataset.Default == 1]
datasetnodef = dataset.loc[dataset.Default == 0]
datasetnodef = datasetnodef.sample(79768)
dataset = pd.concat([datasetdef, datasetnodef])

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
dataset['State'] = labelencoder_X_1.fit_transform(dataset['State'])
labelencoder_X_2 = LabelEncoder()
dataset['BankState'] = labelencoder_X_2.fit_transform(dataset['BankState'])
labelencoder_X_3 = LabelEncoder()
dataset['UrbanRural'] = labelencoder_X_3.fit_transform(dataset['UrbanRural'])
labelencoder_X_4 = LabelEncoder()
dataset['RevLineCr'] = labelencoder_X_4.fit_transform(dataset['RevLineCr'])
labelencoder_X_5 = LabelEncoder()
dataset['LowDoc'] = labelencoder_X_5.fit_transform(dataset['LowDoc'])

onehotencoder_0 = OneHotEncoder(categorical_features = [1,2,4])
```

```
dataset = onehotencoder_0.fit_transform(dataset).toarray()

dataset_part_1 = dataset[1:15953,:]
dataset_part_2 = dataset[15954:31906,:]
dataset_part_3 = dataset[31907:47859,:]
dataset_part_4 = dataset[47860:63812,:]
dataset_part_5 = dataset[63813:79765,:]
dataset_part_6 = dataset[79766:95718,:]
dataset_part_7 = dataset[95719:111671,:]
dataset_part_8 = dataset[111672:127624,:]
dataset_part_9 = dataset[127625:143577,:]
dataset_part_10 = dataset[143578:159530,:]

k1_train = np.concatenate([dataset_part_2,dataset_part_3
,dataset_part_4,dataset_part_5,dataset_part_6,dataset_part_7,
dataset_part_8,dataset_part_9,dataset_part_10])
X1_train = k1_train[:, 1:16]
y1_train = k1_train[:, 0]
k1_test = dataset_part_1
X1_test = k1_test[:, 1:16]
y1_test = k1_test[:, 0]

k2_train = np.concatenate([dataset_part_1,dataset_part_3,
dataset_part_4,dataset_part_5,dataset_part_6,dataset_part_7,
dataset_part_8,dataset_part_9,dataset_part_10])
X2_train = k2_train[:, 1:16]
y2_train = k2_train[:, 0]
k2_test = dataset_part_2
X2_test = k2_test[:, 1:16]
y2_test = k2_test[:, 0]

k3_train = np.concatenate([dataset_part_1,dataset_part_2
,dataset_part_4,dataset_part_5,dataset_part_6,dataset_part_7,
dataset_part_8,dataset_part_9,dataset_part_10])
X3_train = k3_train[:, 1:16]
y3_train = k3_train[:, 0]
k3_test = dataset_part_3
X3_test = k3_test[:, 1:16]
y3_test = k3_test[:, 0]

k4_train = np.concatenate([dataset_part_1,dataset_part_2,
dataset_part_3,dataset_part_5,dataset_part_6,dataset_part_7,
dataset_part_8,dataset_part_9,dataset_part_10])
X4_train = k4_train[:, 1:16]
y4_train = k4_train[:, 0]
k4_test = dataset_part_4
X4_test = k4_test[:, 1:16]
y4_test = k4_test[:, 0]
```

```
k5_train = np.concatenate([dataset_part_1,dataset_part_2
,dataset_part_3,dataset_part_4,dataset_part_6,dataset_part_7,
dataset_part_8,dataset_part_9,dataset_part_10])
X5_train = k5_train[:, 1:16]
y5_train = k5_train[:, 0]
k5_test = dataset_part_5
X5_test = k5_test[:, 1:16]
y5_test = k5_test[:, 0]

k6_train = np.concatenate([dataset_part_1,dataset_part_2,
dataset_part_3,dataset_part_4,dataset_part_5,dataset_part_7,
dataset_part_8,dataset_part_9,dataset_part_10])
X6_train = k6_train[:, 1:16]
y6_train = k6_train[:, 0]
k6_test = dataset_part_6
X6_test = k6_test[:, 1:16]
y6_test = k6_test[:, 0]

k7_train = np.concatenate([dataset_part_1,dataset_part_2,
dataset_part_3,dataset_part_4,dataset_part_5,dataset_part_6,
dataset_part_8,dataset_part_9,dataset_part_10])
X7_train = k7_train[:, 1:16]
y7_train = k7_train[:, 0]
k7_test = dataset_part_7
X7_test = k7_test[:, 1:16]
y7_test = k7_test[:, 0]

k8_train = np.concatenate([dataset_part_1,dataset_part_2,
dataset_part_3,dataset_part_4,dataset_part_5,dataset_part_6,
dataset_part_7,dataset_part_9,dataset_part_10])
X8_train = k8_train[:, 1:16]
y8_train = k8_train[:, 0]
k8_test = dataset_part_8
X8_test = k8_test[:, 1:16]
y8_test = k8_test[:, 0]

k9_train = np.concatenate([dataset_part_1,dataset_part_2,
dataset_part_3,dataset_part_4,dataset_part_5,dataset_part_6,
dataset_part_7,dataset_part_8,dataset_part_10])
X9_train = k9_train[:, 1:16]
y9_train = k9_train[:, 0]
k9_test = dataset_part_9
X9_test = k9_test[:, 1:16]
y9_test = k9_test[:, 0]

k10_train = np.concatenate([dataset_part_1,dataset_part_2,
dataset_part_3,dataset_part_4,dataset_part_5,dataset_part_6,
```

```
dataset_part_7,dataset_part_8,dataset_part_9])
X10_train = k10_train[:, 1:16]
y10_train = k10_train[:, 0]
k10_test = dataset_part_10
X10_test = k10_test[:, 1:16]
y10_test = k10_test[:, 0]

X_train_cross = [X1_train,X2_train,X3_train,X4_train,
X5_train,X6_train,X7_train,X8_train,X9_train,X10_train]
y_train_cross = [y1_train,y2_train,y3_train,y4_train,
y5_train,y6_train,y7_train,y8_train,y9_train,y10_train]
X_test_cross = [X1_test,X2_test,X3_test,X4_test,X5_test,
X6_test,X7_test,X8_test,X9_test,X10_test]
y_test_cross = [y1_test,y2_test,y3_test,y4_test,y5_test,
y6_test,y7_test,y8_test,y9_test,y10_test]

hidden1 = [8,15,30,58]
hidden2 = [8,15,30,58]
nhidden1 = len(hidden1)
nhidden2 = len(hidden2)
ncross = len(X_train_cross)
err = list()
acc = list()

for c in range(ncross):
    print("dataset (1-10)", c)
    set_train_X = X_train_cross[c]
    set_train_y = y_train_cross [c]
    set_test_X = X_test_cross [c]
    set_test_y = y_test_cross[c]
    for i in range(nhidden1):
        N1 = hidden1[i]
        print("N1",N1)
        for j in range(nhidden2):
            N2 = hidden2[j]
            classifier = models.Sequential()
            classifier.add(layers.Dense(output_dim = 15,
            init = 'uniform', input_dim = 15, activation = 'relu'))
            classifier.add(layers.Dense(output_dim = N1,
            init = 'uniform',activation = 'relu'))
            classifier.add(layers.Dense(output_dim = N2,
            init='uniform', activation = 'relu'))
            classifier.add(layers.Dense(output_dim = 1,
            init = 'uniform', activation = 'sigmoid'))
            classifier.compile(optimizer = 'rmsprop',
            loss = 'binary_crossentropy', metrics = ['accuracy'])
            history = classifier.fit(set_train_X, set_train_y,
            batch_size = 100, nb_epoch = 11)
```

```
            err.append(history.history['loss'][10])
            acc.append(history.history['acc'][10])
print(err)
print(acc)
```

Two Layer Neural Network with WOE values - Code

```
import os
os.chdir("C:\\Users\\Marcos Rusinol\\Desktop\\dataset")
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from keras import models
from keras import layers
from sklearn.metrics import confusion_matrix

# Importing the dataset
dataset = pd.read_csv('WOE_datasetpy.csv',sep=';')
print(dataset.shape)
dataset.iloc[:,1:10] = dataset.stack().str.replace(',','.').unstack()
pd.to_numeric(dataset.iloc[:,0], errors='coerce')
pd.to_numeric(dataset["State_woe"], errors='coerce')
pd.to_numeric(dataset.iloc[:,2], errors='coerce')
pd.to_numeric(dataset.iloc[:,3], errors='coerce')
pd.to_numeric(dataset.iloc[:,4], errors='coerce')
pd.to_numeric(dataset.iloc[:,5], errors='coerce')
pd.to_numeric(dataset.iloc[:,6], errors='coerce')
pd.to_numeric(dataset.iloc[:,7], errors='coerce')
pd.to_numeric(dataset.iloc[:,8], errors='coerce')
pd.to_numeric(dataset.iloc[:,9], errors='coerce')

#BALANCING DATASET
#datasetdef = dataset.loc[dataset.Default == 1]
#datasetnodef = dataset.loc[dataset.Default == 0]
#datasetnodef = datasetnodef.sample(79768)
#dataset = pd.concat([datasetdef, datasetnodef])

X = dataset.iloc[:, 1:10].values
y = dataset.iloc[:, 0].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

classifier = models.Sequential()
classifier.add(layers.Dense(output_dim=10, init='uniform',
input_dim=9, activation='relu'))
classifier.add(layers.Dense(output_dim=30, init='uniform',
activation='relu'))
```

```
classifier.add(layers.Dense(output_dim=511, init='uniform',
activation='relu'))
classifier.add(layers.Dense(output_dim=1, init='uniform',
activation='sigmoid'))
classifier.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])
x_val = X_train[:10000]
partial_x_train = X_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
history = classifier.fit(partial_x_train, partial_y_train,
batch_size = 100, nb_epoch = 30, validation_data=(x_val, y_val))

from sklearn.metrics import roc_curve, auc
fpr2, tpr2, threshold = roc_curve(y_test,
classifier.predict_proba(X_test)[:,0])
roc_auc = auc(fpr2, tpr2)
print('roc_auc : ', roc_auc)

plt.figure()
plt.title('ROC CURVE - UNBALANCED WOE_DATASET')
plt.plot(fpr2, tpr2, label = 'MLP AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

predictions_NN_prob = classifier.predict(X_test)
predictions_NN_prob = (predictions_NN_prob > 0.5)

# Creating the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predictions_NN_prob)

# Getting Accuracy, Sensitivity, Specificity and AUC
accuracy = (cm[0,0]+cm[1,1])/(cm[0,0]+cm[0,1]+cm[1,0]+cm[1,1])
sensitivity = cm[0,0]/(cm[0,0]+cm[0,1])
specificity = cm[1,1]/(cm[1,0]+cm[1,1])
```

Two Layer Neural Network with original values - Code

```
import os
os.chdir("C:\\Users\\Marcos Rusinol\\Desktop\\dataset")
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from keras import models
from keras import layers
from sklearn.metrics import confusion_matrix

# Importing the dataset
dataset = pd.read_csv('datasetcompletepy.csv',sep=';')
print(dataset.shape)

#BALANCING DATASET
datasetdef = dataset.loc[dataset.Default == 1]
datasetnodef = dataset.loc[dataset.Default == 0]
datasetnodef = datasetnodef.sample(79768)
dataset = pd.concat([datasetdef, datasetnodef])

#pd.to_numeric(dataset["Default"], errors='coerce')
#pd.to_numeric(dataset["Recession"], errors='coerce')
pd.to_numeric(dataset["NoEmp"], errors='coerce')
pd.to_numeric(dataset["DisbursementGross"], errors='coerce')

X = dataset.loc[:,'State':]
y = dataset.loc[:,'Default']

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X['State'] = labelencoder_X_1.fit_transform(X['State'])
labelencoder_X_2 = LabelEncoder()
X['BankState'] = labelencoder_X_2.fit_transform(X['BankState'])
labelencoder_X_3 = LabelEncoder()
X['UrbanRural'] = labelencoder_X_3.fit_transform(X['UrbanRural'])
labelencoder_X_4 = LabelEncoder()
X['RevLineCr'] = labelencoder_X_4.fit_transform(X['RevLineCr'])
labelencoder_X_5 = LabelEncoder()
X['LowDoc'] = labelencoder_X_5.fit_transform(X['LowDoc'])

onehotencoder_0 = OneHotEncoder(categorical_features = [0,1,3])
X = onehotencoder_0.fit_transform(X).toarray()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

classifier = models.Sequential()
classifier.add(layers.Dense(output_dim=15, init='uniform',
input_dim=16, activation='relu'))
classifier.add(layers.Dense(output_dim=8, init='uniform',
activation='relu'))
classifier.add(layers.Dense(output_dim=8, init='uniform',
```

```
activation='relu'))
classifier.add(layers.Dense(output_dim=1, init='uniform',
activation='sigmoid'))
classifier.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

x_val = X_train[:10000]
partial_x_train = X_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
history = classifier.fit(partial_x_train, partial_y_train,
batch_size = 100, nb_epoch = 30, validation_data=(x_val, y_val))

from sklearn.metrics import roc_curve, auc
fpr2, tpr2, threshold = roc_curve(y_test,
classifier.predict_proba(X_test)[:,0])
roc_auc = auc(fpr2, tpr2)
print('roc_auc : ', roc_auc)

plt.figure()
plt.title('ROC CURVE - BALANCED REAL_VALUES_DATASET')
plt.plot(fpr2, tpr2, label = 'MLP AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

predictions_NN_prob = classifier.predict(X_test)
predictions_NN_prob = (predictions_NN_prob > 0.5)

# Creating the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, predictions_NN_prob)

# Getting Accuracy, Sensitivity, Specificity and AUC
accuracy = (cm[0,0]+cm[1,1])/(cm[0,0]+cm[0,1]+cm[1,0]+cm[1,1])
sensitivity = cm[0,0]/(cm[0,0]+cm[0,1])
specificity = cm[1,1]/(cm[1,0]+cm[1,1])
```

Works Cited

Actuaries Institute. 2012. "Basel 2 Retail Modelling Approaches Pd Models." Present.

Amazon AWS,. 2018. "WOE, IV and Scorecards in Credit Risk Modelling." https://rstudio-pubs-static.s3.amazonaws.com.

Asociación Española de Banca,. 2018. "Importancia y estabilidad bancaria." https://www.aebanca.es/importancia-y-estabilidad-bancaria/.

Baesens, Bart, Daniel Roesch, and Harald Scheule. 2016. *Credit Risk Analytics: Measurement Techniques, Applications, and Examples in Sas*. John Wiley & Sons.

Banco España. 2016a. "Circular 4/2016." *Boletín Oficial Del Estado (BOE) Núm. 110*. Boletín Oficial del Estado (BOE) núm. 110.

———. 2016b. "Implementation and Validation of Basel Ii Advanced Approaches in Spain." *Banco de España Papers*. Banco de España.

Basel Committee on Banking Supervision,. 2001. "The Internal Ratings-Based Approach." https://www.bis.org/publ/bcbsca05.pdf/.

BBVA. 2013. "Exposición en el momento del incumplimiento (EAD)." https://accionistaseinversores.bbva.com.

Brotherton, David. 2013. "Information Value Statistic." https://www.mwsug.org/proceedings/2013/AA/MWSUG-2013-AA14.pdf.

Brownlee, Jason. 2017. "How to One Hot Encode Sequence Data in Python." https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/.

Cañete, Iria. 2018. "Scoring No Clientes (Modelización Con Información Big Data)." *Trabajo Fin de Máster*. Trabajo Fin de Máster.

Comisión Europea. 2013. "Reglamento Delegado (Ue) N 736/2013 de La Comisión de 17 de Mayo de 2013." *Diario Oficial de La Unión Europea*. Diario Oficial de la Unión Europea.

Comité de Supervisión Bancaria de Basilea. 2014. "Revisión Del Método Estándar Para El Riesgo Del Crédito." *Banco de Pagos Internacionales*. Banco de Pagos Internacionales.

CRAN R,. 2019. "Package Scorecard." https://cran.r-project.org/web/packages/scorecard/scorecard.pdf.

Daily Wisdom. 2018. "The Black Box Problem of AI." https://medium.com.

Dauth, Wolfgang, Sebastian Findeisen, Jens Südekum, and Nicole Woessner. 2017. "German Robots-the Impact of Industrial Robots on Workers." CEPR discussion paper no. DP12306.

European Banking Authority, EBA. 2018. "Technical Standards, Guidelines & Recommendations." https://eba.europa.eu/regulation-and-policy/credit-risk.

European Central Bank,. 2019. "ECB 2018 stress test analysis shows improved capital basis of significant euro area banks." https://www.bankingsupervision.europa.eu.

Francois, Chollet. 2017. "Deep Learning with Python." Manning Publications Company.

Furman, Jason, and Robert Seamans. 2019. "AI and the Economy." *Innovation Policy and the*

Economy 19 (1). University of Chicago Press Chicago, IL: 161–91.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. 2011. "Deep Sparse Rectifier Neural Networks." In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 315–23.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Hecht-Nielsen, Robert. 1987. "Kolmogorov's Mapping Neural Network Existence Theorem." In *Proceedings of the International Conference on Neural Networks*, 3:11–14. IEEE Press New York.

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky. 2012. "Neural Networks for Machine Learning Lecture 6a Overview of Mini-Batch Gradient Descent." *Cited on* 14.

Hochreiter, Sepp. 1998. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (02). World Scientific: 107–16.

Huang, Guang-Bin. 2003. "Learning Capability and Storage Capacity of Two-Hidden-Layer Feedforward Networks." *IEEE Transactions on Neural Networks* 14 (2). IEEE: 274–81.

Hunter, David, Hao Yu, Michael S Pukish III, Janusz Kolbusz, and Bogdan M Wilamowski. 2012. "Selection of Proper Neural Network Sizes and Architectures—A Comparative Study." *IEEE Transactions on Industrial Informatics* 8 (2). IEEE: 228–40.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "Imagenet Classification with Deep Convolutional Neural Networks." In *Advances in Neural Information Processing Systems*, 1097–1105.

Kurkova, Vera. 1992. "Kolmogorov's Theorem and Multilayer Neural Networks." *Neural Networks* 5 (3). Elsevier: 501–6.

Le, James. 2018. "The 8 Neural Network Architectures Machine Learning Researchers Need to Learn." https://www.kdnuggets.com.

Li, Min, Amy Mickel, and Stanley Taylor. 2018. "Should This Loan Be Approved or Denied?: A Large Dataset with Class Assignment Guidelines." *Journal of Statistics Education* 26 (1). Taylor & Francis: 55–66.

M. West, Darrell and R. Allen, John. 2018. "How Artificial Intelligence Is Transforming the World." *Brookings*. Brookings.

Majer, I. 2006. "Application Scoring: Logit Model Approach and the Divergence Method Compared)." *Http://Kolegia.sgh.waw.pl*. Warsaw School of Economics.

Malik, Farhad. 2018. "How to Fine Tune Your Machine Learning Models to Improve Forecasting Accuracy?" https://medium.com.

Mathew,. 2015. "Evaluating Logistic Regression Models." https://www.r-bloggers.com/evaluating-logistic-regression-models/.

Mavrovouniotis, ML, and S Chang. 1992. "Hierarchical Neural Networks." *Computers & Chemical Engineering* 16 (4). Elsevier: 347–69.

McFadden, Daniel, and others. 1973. "Conditional Logit Analysis of Qualitative Choice Behavior."

Institute of Urban; Regional Development, University of California ….

Minsky, Marvin, and Seymour Papert. 1969. "Perceptron: An Introduction to Computational Geometry." *The MIT Press, Cambridge, Expanded Edition* 19 (88): 2.

Nieto, S. 2010. "Crédito Al Consume: La Estadística Aplicada a Un Problema de Riesgo Crediticio." *Http://Www.academia.edu*. Universidad Autónoma Metropolitana.

Oliner, Stephen, Daniel E Sichel, and Kevin J Stiroh. 2007. "Explaining a Productive Decade." *Brookings Papers on Economic Activity* 2007 (1). Brookings Institution Press: 81–152.

Parmar, Ravindra. 2018. "Common Loss functions in machine learning." https://towardsdatascience.com/.

Parr, Terence, and Jeremy Howard. 2018. "The Matrix Calculus You Need for Deep Learning." *arXiv Preprint arXiv:1802.01528*.

Rouse, Margaret. 2016. "Artificial Neural Network (Ann)." *Https://Searchenterpriseai.techtarget.com*. Search Enterprise AI.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1985. "Learning Internal Representations by Error Propagation." California Univ San Diego La Jolla Inst for Cognitive Science.

Senior Supervisors Group, SSG. 2009. "Risk Management Lessons from the Global Banking Crisis of 2008." *Senior Supervisors Group (SSG)*. Senior Supervisors Group (SSG).

Sheela, K Gnana, and Subramaniam N Deepa. 2013. "Review on Methods to Fix Number of Hidden Neurons in Neural Networks." *Mathematical Problems in Engineering* 2013. Hindawi.

Singh Walia, Anish. 2017. "The Vanishing Gradient Problem." https://medium.com.

Small Business Administration,. 2019. "U.S. Small Business Administration." https://www.sba.gov/.

Soni, Manik. 2017. "Exploding and Vanishing Gradient Problem: Math Behind the Truth." https://hackernoon.com.

Statinfer,. 2018. "Calculating Sensitivity and Specificity." https://statinfer.com.

Tamura, Shin'ichi, and Masahiko Tateishi. 1997. "Capabilities of a Four-Layered Feedforward Neural Network: Four Layers Versus Three." *IEEE Transactions on Neural Networks* 8 (2). IEEE: 251–55.

Varangaonkar, Amey. 2018. "Top 5 Deep Learning Architectures." https://hub.packtpub.com/top-5-deep-learning-architectures/.

Zhang, Zhaozhi, Xiaomin Ma, and Yixian Yang. 2003. "Bounds on the Number of Hidden Neurons in Three-Layer Binary Neural Networks." *Neural Networks* 16 (7). Elsevier: 995–1002.

Zulkifli, Hafidz. 2018. "Understanding Learning Rates and How It Improves Performance in Deep Learning." https://towardsdatascience.com.