Pontifícia Universidade Católica
do Rio de Janeiro

Gustavo Madeira Krieger

Reducing Teacher-Student Interactions Between Two Neural Networks

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Sérgio Colcher

Rio de Janeiro
April 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Gustavo Madeira Krieger

Reducing Teacher-Student Interactions Between Two Neural Networks

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the undersigned Examination Committee.

Prof. Sérgio Colcher
Advisor
Departamento de Informática – PUC-Rio

Prof. Bruno Feijó
Departamento de Informática – PUC-Rio

Prof. Ruy Luiz Milidiú
Departamento de Informática – PUC-Rio

Rio de Janeiro, April 30th, 2019

Gustavo Madeira Krieger

Graduaded in computer engineering by the Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). His research is focused in Machine Learning and Knowledge Distillation.

Bibliographic data

Krieger, Gustavo Madeira

Reducing Teacher-Student Interactions Between Two Neural Networks / Gustavo Madeira Krieger; advisor: Sérgio Colcher. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 57 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Aprendizado de Máquina;. 3. Destilação de Conhecimento;. 4. Perceptron Multicamada;. 5. Classificação Multiclasse.. I. Colcher, Sérgio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

First and foremost, thanks to my advisor Prof. Ruy Luiz Milidíu for his guidance, wisdom and support for this work. Thanks to the Informatics Department at PUC-Rio and its wonderful professors. Thanks to my friends at LEARN, Luis and Rafael, whose help and knowledge were key to making this a reality. Finally, thanks to my family, who were by my side cheering me on and their endless support.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Krieger, Gustavo Madeira; Colcher, Sérgio (Advisor). **Reducing Teacher-Student Interactions Between Two Neural Networks**. Rio de Janeiro, 2019. 57p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Propagation of knowledge is one of the pillars of human evolution. Our discoveries are all based on preexisting knowledge, built upon them and then become the foundation for the next generation of learning. In the field of artificial intelligence, there's an interest in replicating this aspect of human nature on machines. By creating a first model and training it on the original data, another model can be created and learn from it instead of having to learn everything from scratch. If this method is proven to be reliable, it will allow many changes in the way that we approach machine learning, specially allowing different models to work together. This relation between models is nicknamed the "Teacher-Student" relation. This work describes the development of two separate models and their ability to learn using incomplete data and each other. The experiments presented here show the results of this training and the different methods used in the pursuit of an optimal scenario where such learning process is viable for future use.

Keywords

Machine Learning; Knowledge Distillation; Multilayer Perceptron; Multiclass Classification.

Resumo

Krieger, Gustavo Madeira; Colcher, Sérgio. **Reduzindo as interações professor-aluno entre duas redes neurais**. Rio de Janeiro, 2019. 57p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Propagação de conhecimento é um dos pilares da evolução humana. Nossas descobertas são baseadas em conhecimentos já existentes, construídas em cima deles e então se tornam a fundação para a próxima geração de aprendizado. No ramo de Inteligência Artificial, existe o interesse em replicar esse aspecto da natureza humana em máquinas. Criando um primeiro modelo e treinando ele nos dados originais, outro modelo pode ser criado e aprender a partir dele ao invés de ter que começar todo o processo do zero. Se for comprovado que esse método é confiável, ele vai permitir várias mudanças na forma que nós abordamos machine learning, em que cada inteligência não será um microcosmo independente. Essa relação entre modelos é batizada de relação "Professor-Aluno". Esse trabalho descreve o desenvolvimento de dois modelos distintos e suas capacidades de aprender usando a informação dada em um ao outro. Os experimentos apresentados aqui mostram os resultados desse treino e as diferentes metodologias usadas em busca do cenário ótimo em que esse processo de aprendizado é viável para replicação futura.

Palavras-chave

Aprendizado de Máquina; Destilação de Conhecimento; Perceptron Multicamada; Classificação Multiclasse.

Table of contents

List of figures

List of tables

List of Abreviations

AI – Artificial Intelligence
ML – Machine Learning
NN – Neural Networks
ANN – Artificial Neural Networks
MLP – Multi Layer Perceptron
TPR – True Positive Rate
TNR – True Negative Rate

1
Introduction

1.1
Overview

Propagation of knowledge is one of the pillars of human evolution. Our discoveries are all based on preexisting knowledge, built upon them and then become the foundation for the next generation of learning. In the field of Artificial Intelligence (AI), there is an interest in replicating this aspect of human nature on machines. The idea is to create an AI, train it and finally use its knowledge to teach another AI, similar to a teacher and a student.

This is a very recent field of study, with one of its pioneer works being Model Compression (1), written by Rich Caruana, Cristian Bucila, and Alexandru Niculescu-Mizil in 2006. In this work, the authors state that more accurate models require more space and more processing power, which reduces their viability in day to day activities. However, through their studies they show that less robust AI models can be trained based on the bigger models and achieve similar results. This new, less demanding model is then used instead of its cumbersome counterpart.

Although the purpose of this work is not to create a lighter version of an existing model, the ideas discussed in the previously mentioned paper are the base from which the Knowledge Distillation area of research has stemmed from. By creating a first model and training it on the original data, another model can be created and learn from it. If this method is proven to be reliable, it will allow many changes in the way that we approach machine learning, increasing overall accuracy and diminishing the differences between models learning the same data. This relation between models is nicknamed the "Teacher-Student" relation.

This work describes the development of two separate AI models and their ability to learn using each other. The experiments presented here show the results of this training and the different methods used in the pursuit of an optimal scenario where such learning process is viable for future use.

1.2 Related Work

The Model Compression (1) paper is the first that comes to mind when talking about recent Knowledge Distillation discussion. It is considered the first modern paper that discusses this subject and the idea of two AI working together to improve each other. It talks specifically about usability and the problem of big AI models not being applicable in day to day activities.

After it, there is a drought in this area, with no other research of note being published until 2015 with Distilling the Knowledge in a Neural Network by Hinton et al. (2). In it, they explore the needs and differences between models used to learn huge datasets and models used in smaller systems.

This reignited the discussion on Knowledge Distillation and its applications. The success of the previous researches raised the question of their capacity, a problem approached by Urban et al. (3) in Do Deep Convolutional Nets Really Need to be Deep and Convolutional. They conclude that yes, they need to be deep and convolutinal. Shallow networks can not replicate the results on more complex problems and are left wanting. However, they do not justify why additional layers are necessary to achieve better results.

The doors opened and several other papers started working with knowledge distillation. Two of note were Gupta et al. (4) and Zagoruyko et al. (5) with Cross Modal Distillation for Supervision Transfer and Paying More Attention To Attention: Improving The Performance Of Convolutional Neural Networks Via Attention Transfer, respectively. They have similar ideas of using an intelligence to filter data and guide another intelligence in the right direction, achieving better results than a stand-alone model.

Finally, Lopes et al. (6) studied the workings of an AI with unverified information and another AI to help it. It reversed the results obtained on the full information AI to reconstruct the original data and fed it to the second AI. It achieved acceptable results and showed that such an approach to AI is possible, although with some limitations.

1.3 Contribution and Outline

In this dissertation we develop models for learning several different datasets. In particular, we develop classification models that use an existing model as a teacher to help the model achieve better results. The objective is to show that this approach is viable and has future applications.

In Chapter 2, we describe how the world is changing due to the current possibilities for storing and analyzing data. Then we present details of Machine

Learning with regard to the overall process of building an intelligent application. This is followed by a historical description of Neural Networks up to the one of the most recent breakthrough in the field, Deep Learning. Finally, we give background info on knowledge distillation and its problems.

Chapter 3 contains an explanation of the main problem tackled in this work, which is the knowledge distillation between two neural networks, and propose an approach to the Teacher-Student relation that reduces the call to the teacher. The main goal is to study and analyze the results achieved with different networks under the influence of a teacher. The teacher model will be the same in every experiment.

Chapter 4 presents the classification problem as a whole. It explains its uses and applications, as well as methods to approach it with optimal results.

Chapter 5 presents the methodology used to build the classification models and the preprocessing of the datasets used in this work. The data is processed to remove unnecessary information and better streamline the input for the neural networks.

Chapter 6 contains the experimental evaluation of the models and methods. As baseline, we used the teacher model result and later analyzed its performance against the apprentice networks, with and without distillation.

Finally, Chapter 7 presents the achievements of this dissertation and identifies remaining challenges and potential avenues of evaluation in future work.

2
Background

In recent years, AI has grown to become one of the most interesting and sought after research areas in Computer Science. Its origins data back to 1956, when it was formally named by John McCarthy in his proposal for a project about machines that reason intelligently (7). However, AI would face many rocky roads until it reached the status it has today.

Achieving an artificially intelligent being wasn't so simple back then as it is now. After several reports criticizing progress in AI, government funding and interest in the field waned and began a period from 1974–80 that became known as the "AI winter". The field later revived in the 1980s when the British government started funding it again in part to compete with efforts by the Japanese.

The field experienced another major winter from 1987 to 1993, coinciding with the collapse of the market for some of the early general-purpose computers and, once again, reduced government funding. But research began to pick up again after that, and in 1997, IBM's Deep Blue became the first computer to beat a chess champion when it defeated Russian grandmaster Garry Kasparov.

In modern days, it has fanned out in many subfields, from general-purpose utilities such as logical reasoning and natural language processing to specific areas such as playing the famous Jeopardy game show, beating the best human players in a match. Its growth is related to the application of AI in almost all fields of research, even outside of Computer Science. Scientists often find tools to computationally express intellectual tasks, and automate and improve their work (8). Its spread and acceptance show that it is truly a universal field.

Much of the revival in interest in AI is due to Machine Learning. Machine Learning is the part of AI responsible for developing computational theories focused on data pattern extraction in order to make predictions or decisions. For a long time, it was deemed impossible to work on this field due to the high costs involved. The cost of keeping the high amount of data and the computational power required to process in an acceptable time was immense. As prices dropped, the field grew and more works started incorporating it. The key characteristic of Machine Learning is the ability to learn from previous

accumulated knowledge and making decisions based on that. Given some context, one can take a history of information, learn from this record, and then model these activities to improve our understanding of this context going forward.

2.1 Machine Learning

Machine Learning uses data in a direct and productive way, filtering only relevant information and disconsidering outliers and data points that do not aid in the problem at hand. It is designed to *train* machines to extract patterns from previous data, sometimes acting in an frightening human-like fashion. Watson (9) comes to mind when talking about this. It was trained in natural language processing and context analysis to play the 'Jeopardy!' television game and managed to defeat the best human players in the world. While this 'human-like' approach might not be completely correct all of the time, the solutions are often considered good enough to solve everyday problems. The truth of the matter is that this kind of solution requires data; lots of data.

It is natural to conclude then that, with the advent of Big Data, the possibilities of creating such models are significantly enhanced and thus the correctness of their predictions too. Clearly, with more available data, it is probable that the predictions will be better. With the addition of real time information sources and real time decision making continuously creating even more data, there is a rapidly growing amount of data in our daily lives that influences our choices and behavior. But how can this data and information be easily understood and translated to improve the efficiency or quality of our business processes and our lives?

2.1.1 General Concepts of Machine Learning

Machine Learning has many concepts and theories, so many it's unviable to discuss all of them in this work. Summarizing, there are three main steps for the creation of a successful Machine Learning algorithm: Representation, Evaluation and Optimization (10). Following is an attempt at describing the whole process in an abstract but effective way.

1. **Representation:** The first step is to work on the data available to create a *hypothesis space*, i.e. a collection of data that can be interpreted and learned by the computer. This step starts with the selection of the data to be used, it's cleaning, sorting and analysis of viability as a dataset. After this is done, we need to select the features. A feature is a measurable

property of a dataset, for instance, to a computer, an image is just a bunch of pixel values. However, features are not limited to only the data received, as there are many features that can be extracted from it. In the case of the pixels, density of pixels per quadrant can be useful and is easily derived from the raw data. After the data has been worked to satisfaction, it's separated into two groups: training data - the examples the application will learn from - and testing data - the examples that will be used to verify the accuracy of the model.

2. **Model Selection:** At this point it's time to define what model one should be working on. It's good to start by developing simple models and increase their complexity, only if necessary. The model is part of a past reality where the occurrence of what took place is known. This allows the proposed algorithm learn until it's able to predict the outcome with the required quality level.

3. **Evaluation:** The next step is selection the method of evaluation of the model, i.e. the metrics and functions that will determine the success of the model. This can be as simple as the accuracy of prediction on percentage points or even a mathematical function such as the *log loss* function. This is used to verify if the model chosen or the parameters set are good or bad.

4. **Learning:** Now it's time to start the most important part: the training. Here we must configure and tune the algorithm parameters in order to minimize the errors.

5. **Application:** This step is where the model is applied to new, unknown data and analyzed to see if the results are consistent with reality. If the overall results are not acceptable, it's necessary to return back steps to find the error. In the worst case scenario, one might even have to rethink his features and start the whole process from scratch.

6. **Production:** At this point, the model has already been successfully validated and the time has come to put everything into production. Of course, even in production, the model should be continuously tuned and evolved in concert with changes in the processes being modelled.

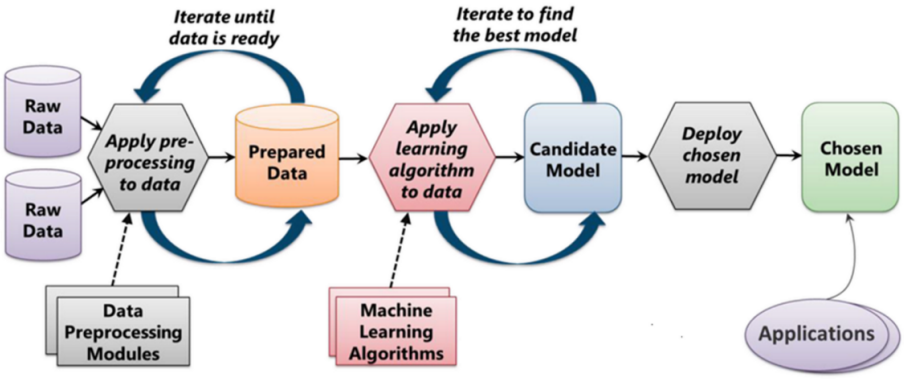

Figure 2.1: Machine Learning Steps

2.1.1.1 Different Types of Learning

When creating the machine learning model, we need to determine what kind of learning is going to be used. Broadly speaking, there are three types of learning: Supervised, Semi-Supervised and Unsupervised. The choice of which one depends on the data, the results expected and several other factors, so sometimes the choice has already been made by what kind of problem it's trying to solve.

Supervised learning is the simplest to approach. The data all has a result attached to it, e.g. a class in a classification problem, that helps the model to learn what's right and what's wrong. The problem with it is the cost. Finding results for a expansive dataset means a lot of human work was necessary. This can be extremely expensive and impossible with limited resources. The models in this work use this kind of learning.

Unsupervised learning is the extreme opposite of supervised learning, with absolutely no results. It is mostly used for clustering problems, where exact results aren't as important as finding similarities in the data and organizing it into different groups. The upside is that this kind of data is extremely cheap, so lots of it can be made, much more than the other types of learning for the same price.

Semi-supervised leaning strikes a balance between the other two. Some of the data has results, but most doesn't. In this kind of learning, we need to mix the methods applied on the other learning types, clustering the unknown data around known data and finding their class using an approximation model, such as the K-Means.

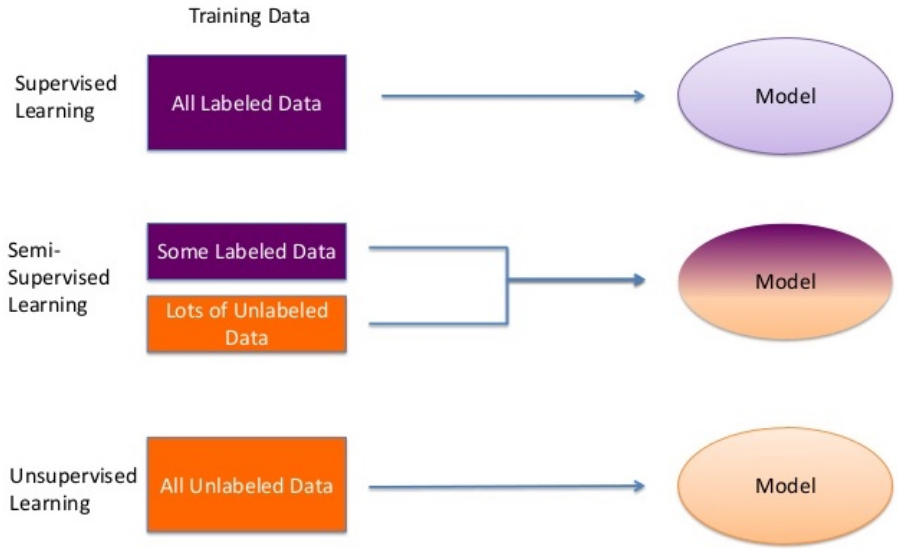

Figure 2.2: Different Types of Learning

2.1.2 Neural Networks

Artificial Neural Networks are mathematical models that try to represent the operation of Biological Neural Networks. They are based on research into the structure and functions of the human brain, and try to emulate intelligence with regard to information processing. Some studies of neurophysiology consider that the computational power and efficiency of the human brain is associated with its large number of neurons, interconnected by a complex network of synapses. It is estimated that the number of neurons of the human brain is of the order of billions. However, the processing speed of these components is relatively slow as compared with traditional computers. This lack of speed is overcome by the huge quantity of neurons operating in parallel. Such characteristics allow the human brain to execute some functions, such as sound and image recognition, in a way that conventional computers aren't able to compute with similar performance (11).

2.1.2.1 The Perceptron

The first Neural Network model implementation was the perceptron, by Frank Rosenblatt in 1958 (12). The perceptron is based on the McCulloch and Pitts (13) neuron with an input and output layer. For each entry, there exists a related weight wherein the output value is the sum of the products of each entry with their respective weight. Rosenblatt describes a topology for Neural Networks and proposes an algorithm to train the network and execute certain types of functions. The perceptron behaves as a pattern classifier, dividing the

entry space into distinct regions that represent each existing class.

Despite the impact the perceptron had on the Artificial Intelligence community, this model was heavily criticized by Minsky and Papert (14). In their book, the authors cite an example of the perceptron with a single layer that cannot simulate, for instance, the behavior of a simple XOR function (exclusive-or), since it can only solve linearly separable[1] problems. However, after the publication of this work, the period known as the dark years of the neural networks started, as discouraging new research, a lack of funding and deeper studies turned interest away from this field.

The deficiency of the perceptron with non-linear patterns was eliminated by Rumelhart, Hinton and Willians (15). The solution was the generalization of the Delta Rule, known as the Backpropagation algorithm. The perceptron had proved its validity, enabling the implementation of the third layer required to learn the XOR. Using a network of neurons as those used in the perceptron, the backpropagation performs a back-propagation of the output error for prior layers. The error is the result of the comparison between the desired output (pre-determined) and the actual output of the network. With this back-propagation, in conjuction with one threshold function of fractional values (leaving out the all-or-nothing), representation of non-linear functions was made possible, allowing a multi layer perceptron (MLP) Neural Network be trained to learn the XOR function.

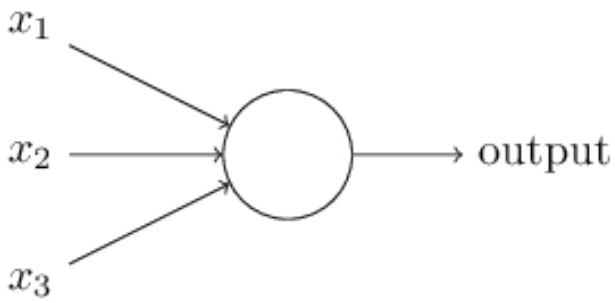

Figure 2.3: A Perceptron

2.1.2.2 Neural Network Characteristics

The operation model of a processing unit (Neuron) proposed by McCulloch and Pitts can be summarized as follows:

[1] A dataset is said to be linearly separable if there exists a hyperplane in the input space that separates data from different classes. In geometry, a hyperplane can be a vector space, affine transformation or n-1 dimensional subspace. For example, for a tridimensional dataset, the hyperplane is a normal 2D plane, for a bidimensional, a line and for a unidimensional, a point.

– Signals are provided as input;

– Each signal is multiplied by a number, or weight, which indicates or reflects their influence on the output;

– The weighted sum of the signals produces a level of activity;

– If this activity level exceeds a certain threshold, the unit outputs 1, otherwise, 0.

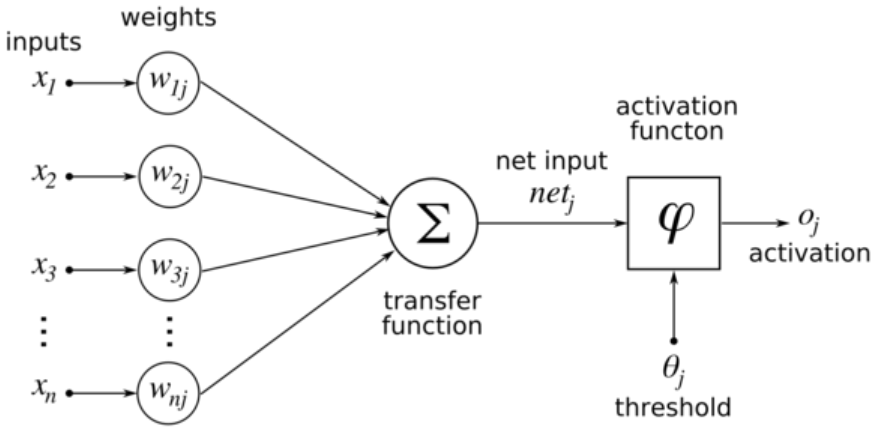

Figure 2.4: Artificial Neuron Structure

A perceptron takes several binary inputs, x_1, x_2, ..., and produces a single binary output. The basic mathematical model can be described as follows:

$$output = \begin{cases} 0 & if \sum_j w_j x_j \leq threshold \\ 1 & if \sum_j w_j x_j > threshold \end{cases}$$

To simplify, some notational changes can be made. One can write $\sum_j w_j x_j$ as a dot product, $w \cdot x \equiv \sum_j w_j x_j$, where w and x are vectors whose components are the weights and inputs, respectively, and move the threshold to the other side of the inequality, replacing it by what's known as the perceptron's bias, $b \equiv -threshold$. Hence, the perceptron rule can be redefined as:

$$output = \begin{cases} 0 & if w \cdot x + b \leq 0 \\ 1 & if w \cdot x + b > 0 \end{cases}$$

Perceptrons can be used to represent basic elementary logical functions such as AND, OR, and NAND. For example, if we create a perceptron with two inputs in the input layer, each with weight -2, and an overall bias of 3. In a simple way, Figure 2.5 describes this perceptron.

Analyzing the possible outcomes, the input 00 produces output 1, since $(-2)\times 0+(-2)\times 0+3 = 3$ is positive. Similarly, inputs 01 and 10 produce output 1. But the input 11 produces output 0, since $(-2) \times 0 + (-2) \times 0 + 3 = -1$ is

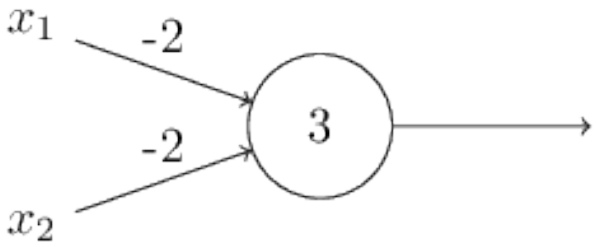

Figure 2.5: Simple perceptron with weigths and bias

negative. This lets us conclude that the perceptron above implements a NAND gate. This example shows that we can use perceptrons to compute simple logical functions. In fact, we can use networks of perceptrons to compute any logical function since the NAND gate is universal for computation, that is, we can implement any computation out of NAND gates.

The computational universality of perceptrons mean that that networks of perceptrons can be as powerful and flexible as any other computing device. But it is also disappointing, because it makes it seem as though perceptrons are merely a new type of NAND gate. However, it turns out that we can use learning algorithms which can automatically tune the weights and biases of a network of artificial neurons without the direct intervention of a programmer. These learning algorithms enable us to use artificial neurons in a way which is radically different to conventional logic gates. Instead of explicitly laying out a circuit of NAND and other gates, our neural networks can simply learn to solve problems, sometimes problems where it would be extremely difficult to directly design a conventional circuit.

Most Neural Network models have some form of training step, where the weights of connections are adjusted in accordance with the provided patterns. In other words, they learn through examples. Neural architectures are typically organized into layers that usually are classified as shown below:

- **Input layer:** Where inputs are presented to the network;
- **Intermediate or Hidden Layers:** Where most of the processing is done through weighted connections; Can be considered as a feature extractor;
- **Output layer:** Where the output is produced.

In Figure 2.6, a Neural Network with four layers and two hidden layers is presented. The term "hidden" refers to the layers between the input and output ones. Designing input and output layers is often simple. For instance, suppose we want a Neural Network to recognize the digit "9". A good way to design the input layer is to encode image pixels intensities into the input neurons. The

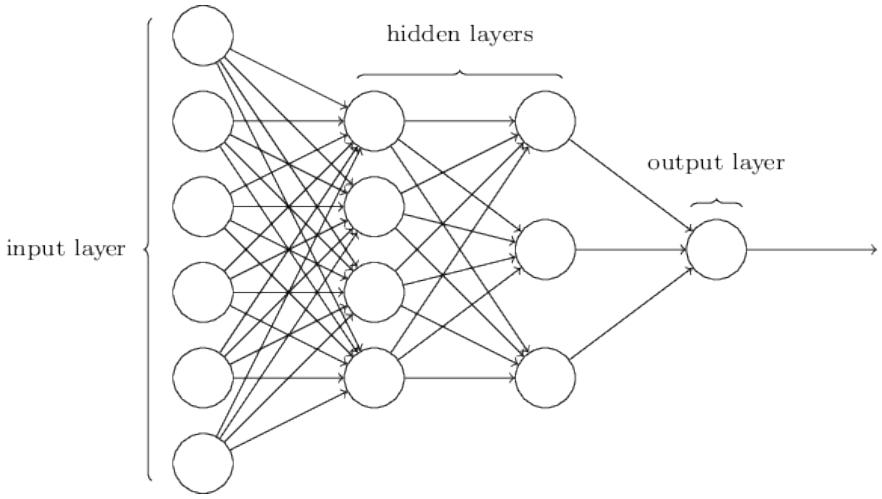

Figure 2.6: A Multi-layered Neural Network

output layer will contain just a single neuron, with output value indicating if the image is a "9" or not. However, designing the hidden layers has been a major interest of Neural Network research. While there aren't a few simple rules of thumb, one of the most crucial trade offs is related to the number of hidden layers against the time required to train the network.

2.1.2.3 Deep Learning

A subarea of Machine Learning, known as deep learning, accounts for much of the renewed excitement surrounding AI. The depth of an architecture is related to the number of levels of non-linear operations in the learned function. In a Neural Network, this corresponds to the number of layers. Functions that can be represented by an architecture of depth k may require an exponential number of computational elements to be represented by an architecture of level $k-1$. This means that some real world functions cannot be represented by shallow architectures.

Deep Learning methods focus on learning features of higher levels through the composition of lower level features and is inspired by the human brain. For decades, researchers tried, without success, to train Neural Networks with multiple deep layers. But in 2006, Hinton discovered that the results of a deep Neural Network could be improved when pre-trained with an unsupervised learning algorithm layer by layer, starting with the first one (16). This work started the area known today as Deep Learning.

2.2 Model Compression

Working with machine learning requires a balance between efficiency and accuracy. One could use a giant ensemble of several deep learning algorithms to solve trivial problems, but this approach would be too costly. There are also limitations on which machine this model will be running. On the other end of the spectrum, a model too shallow will have low accuracy and not achieve desirable results. So the question arose on how to create a small model with the accuracy of a large model. One of the answers was model compression.

Model compression (1) is the first widespread application of knowledge distillation. When dealing with increasingly complex problems, it is natural to use increasingly bigger models. This bigger models learn the problem better, at the cost of requiring more time for training and processing and more space to save it. However, using knowledge distillation, Caruana et al. showed that it is possible to make a smaller model copy the process of larger models. Using this method, he could create an ensemble and train it to achieve the best possible result and then create a smaller model to replicate it.

The problem with model compression is that it does not care about the smaller model findings. The smaller model serves only to copy the bigger model. Even if the smaller model manages to find the correct answer to cases that the bigger model gets wrong, it is dismissed in favor of the bigger model overall better results. While this streamlines the distillation process, there may be critical information lost. There's also the problem that we need to consult the cumbersome model at every case to learn from it. This makes the learning much slower.

2.2.1 Dark Knowledge

The most obvious way to have the student imitate or mimic the teacher would be to train it directly on the output of the teacher. However, it turns out that doing just that will result in a severe drop in predictive performance, because the student model is not capable of generalizing to unseen data. The reason for this is something that Hinton et al. (19) calls *Dark Knowledge*. Dark knowledge is their analogy to dark matter in physics, suggesting that a good part of the knowledge obtained by the model is somehow hidden. They argue that most of the knowledge learned by the teacher is hidden in the relative probabilities of the wrong answers.

Hinton gives the following example in his presentation. Consider we give our model the image of a dog, i.e. the truth reads:

$$(P_{cow} \quad P_{dog} \quad P_{cat} \quad P_{car}) = (0 \quad 1 \quad 0 \quad 0)$$

And our model output would be:

$$y = (10^{-6} \quad 0.9 \quad 0.1 \quad 10^{-9})$$

We can see that the label *cow* is relatively unlikely but it is still by a factor of 10^3 more likely than the label *car*. This tells us a lot about the models ability to generalize, since somehow he appears to have learned to distinguish between animals and objects like cars, maybe because the teacher found out that dogs, cats and cows have four legs and a car is something completely different. This kind of knowledge that does not immediately contribute to the decision is something that we can not neglect if we want our student to be able to generalize well. Nevertheless, when training the student with a cross–entropy loss function on the teacher's label directly would ultimately result in the student almost ignoring everything about cats and cows and their relation to each other, since really small probabilities do not really contribute.

To solve this, they propose a way to soften the probability distribution and smooth the target, allowing for distillation to take into consideration this values. This process is the crux of our work, and will be explained in full in the next chapter.

3
The Teacher-Student Relation

It is an obvious logical expectation that the function used for training should reflect the objective of the user as closely as possible. Despite this, models are normally trained to optimize performance on the training data when the real objective is to generalize well to new data. It would clearly be better to train models to generalize well, but this requires information about the correct way to generalize. This is the information we are trying to achieve with the training in the first place. (2)

When we are distilling the knowledge between two models, however, we can train the second model to generalize in a similar way as the first model. If the first model, the "Teacher", already generalizes well because of its training, the second model, the "Apprentice", trained to generalize in a similar way will typically do much better on test data than a model that is trained without a Teacher, since it has the additional information already accrued by the Teacher.

Consider a typical scenario of a teacher and an student through a learning process. The student is not made to copy the teacher in every aspect and replicate what he does. He approaches the teacher only when he is mistaken or otherwise stuck and the teacher helps him through the difficulty. The teacher, for his part, allows the student to learn on his own and realise his potential. We apply this same logic to our models.

The student model learns from the dataset as normal. But whenever he makes a wrong classification, there is distillation of the teacher's knowledge through logit transfer, which will be explained later in the chapter. This allows the student model to retain the correct information and not be as influenced by the teacher to the point the two models are the same.

The Teacher-Apprentice relation is a new method of knowledge distillation that we developed and used on this work. We distanced ourselves from the idea of copying the larger models into the smaller models and allowed the learning to happen. Our main goal is to showcase that this method improves training time and possibly model accuracy, especially when dealing with similarly sized models.

Going forward, we are going to explain how the teacher-student distillation is executed and its methods and steps.

3.1 The Distillation Process

When dealing with knowledge distillation, the main issue we are trying to solve is how to train the student model. Taking the step from normal learning to learning with distillation raises several questions, such as how much of the original data do you use and what approach to distillation you take.

There are many studies in these topics that focus solely on optimizing or solving them, but this is not the final objective of this work. In this work, we use methods based on Hinton et al. (2) researches to develop a new distillation process, the Teacher-Apprentice relation.

3.1.1 Overview of Hinton's Method

Hinton et al. (2) created a framework for knowledge distillation that achieved great results. His efforts were to think knowledge distillation in a different light. Knowledge distillation, in it's first thinking, treated the cumbersome model as a black box. By feeding this model data, it is possible to learn from its outputs and eventually replicate its inner workings. Hinton worked on a method using the what he called the "logits", the pre-normalized values of the last layer of the neural network.

The process proposed by him is as follows: we start with a singular model and a dataset D, for classification, with features x and labels y.

$$D = \{(x, y)\}$$

This model can have any configuration desired and is trained normally. In this work we will use a similar network composition than the one used by Hinton et al. After this model is trained to satisfaction, it is then saved for later use. Its logits will be added to y to create a new target to learn. This new $\hat{y}$ will be used to create a new dataset, $\hat{D}$.

$$\hat{D} = \{(x, \hat{y})\}$$

This dataset is named the "transfer set". We now create a new model to which we feed the transfer set. This new model can be anything and has no relation to the previous model, but usually it is a lighter model and will be the one where information is distilled to. This distillation happens at the calculation of the loss function. The entry data is unchanged and working of the neural suffer no changes prior to the loss calculation. The loss calculation and minimization still occurs, but before this calculation there is a loss calculation

between this models findings and the previous models findings. This calculation is named "logits transfer".

This summarizes the whole process, and each step will be explained further in detail.

3.1.2 Our approach

When studying Hinton's method, a simple question came up: is it necessary to distill knowledge on every case? When the smaller model classifies it correctly, the distillation is unnecessary since it already has the right answer. There's also the problem that the cumbersome model doesn't have a 100% correct rate, so it is possible that it is teaching wrong information. Although this change does not guarantee this problem is solved, it at least guarantees that there will be no cases of a right classification being distilled to a wrong classification. There is also a possible gain in performance since there is no need to consult the cumbersome model every time.

This simple change is the focus of our work, and we record the changes it makes to the process.

3.1.3 Learning

The learning is done "online". This means that the error calculation and loss function is applied after each single input during training, with the new values being applied to the next input. This differs from the "offline" method, which waits the model reach the end of an epoch to apply the loss function. The distillation is also done online, being applied at each iteration of learning.

The dataset input order is randomised before being fed to the model since randomized arrangement of the observations according to the response variable is strongly preferred versus ordered arrangement, e.g. if we have 150 observations comprising our training set, we do not want the network to see all 50 observations in class one, then all 50 in class two, then all 50 in class three. Randomizing the entries, however, presented a problem. We need to know which set of logits, the main source of distillation, were related to each entry so we can instruct the student correctly. Our solution was creating an array of equal size to the number of distinct entries on the dataset, with each value on the array representing the logits of the specific entry numerically, in order, i.e. the logits of the 500th entry is in the 500th array position. We shuffle the feeding order while remembering their order number and then fetched that

value. Figure 3.1 shows a diagram with a macro representation of the learning process. The pseudocodes 1 and 2 show the first step of the learning process.

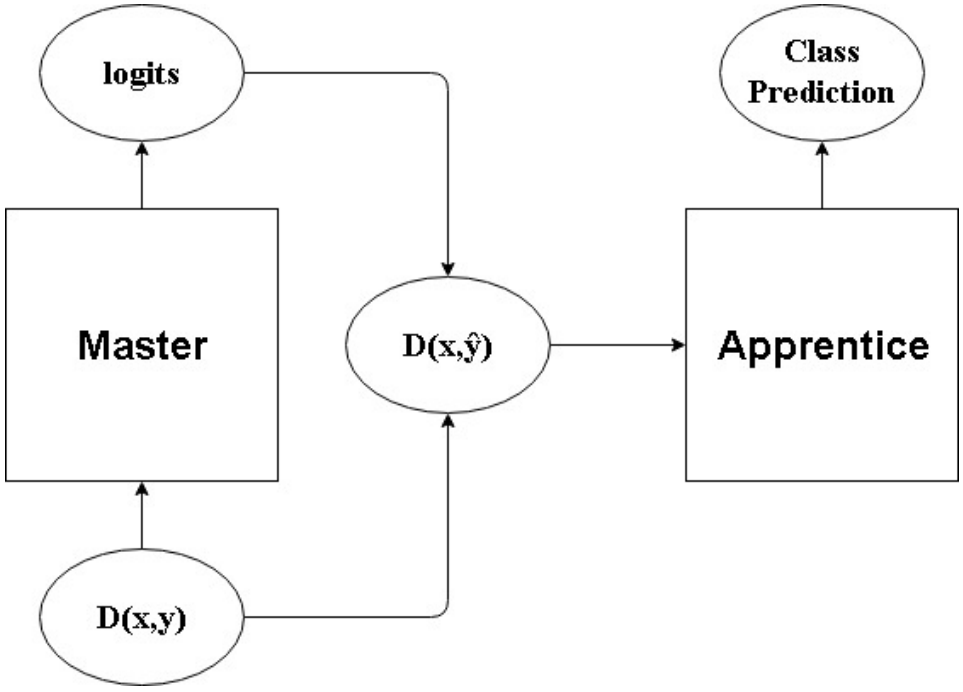

Figure 3.1: Teacher-Apprentice Relation Diagram

Algorithm 1 Teacher Learning

```
X ← Input Layer
W1 ← First Hidden Layer Starting Values
W2 ← Second Hidden Layer Starting Values
Y ← Output Layer
model ← X + W1 + W2 + Y
dataset ← Dataset
true-class ← Classes
for entry in dataset do:
    i ← Entry Order
    model ← TRAINING(model, dataset[i], true-class[i])
```

Algorithm 2 Apprentice Learning

1: $X \leftarrow$ *Input Layer*
2: $W1 \leftarrow$ *First Hidden Layer Starting Values*
3: $W2 \leftarrow$ *Second Hidden Layer Starting Values*
4: $Y \leftarrow$ *Output Layer*
5: $model \leftarrow X + W1 + W2 + Y$
6: *teacher-logits* $\leftarrow$ *Teacher logits*
7: *dataset* $\leftarrow$ *Dataset*
8: *true-class* $\leftarrow$ *Classes*
9: **for** entry in dataset **do**:
10: $i \leftarrow$ *Entry Order*
11: $model \leftarrow$ Training w/ Distillation$(model, dataset[i], teacher - logits[i], true\text{-}class[i])$

3.1.4 Distillation Targeting

When we do distillation, we must decide what is the target of our distillation, i.e. what value are we trying to pass down between generations. This target can be a feature or a result. In our case, our target is the label of the class we are trying to achieve.

A way to transfer the generalization ability of the teacher to the student is to use the class probabilities produced by the teacher as "soft targets" for training the student. This works well especially if we add a small term to the objective function that encourages the student model to predict the true targets as well as matching the soft targets provided by the teacher model. Typically, the student model cannot exactly match the soft targets and erring in the direction of the correct answer turns out to be helpful (2).

In the simplest form of distillation, knowledge is transferred to the student model by training it on the dataset and using a soft target distribution for each case in the dataset that is produced by using the teacher model with a high temperature in its softmax. The temperature is a variable that can be applied to the softmax function that smooths the probability distribution of the softmax. This is the parameter that allows the student model to learn the small details that could have been hidden if no treatment was done to the result.

A better way is to simply use a weighted average of two different objective functions. The first objective function is the cross entropy with the soft targets and this cross entropy is computed using the same high temperature in the softmax of the student model as was used for generating the soft targets from

the teacher model. The second objective function is the cross entropy with the correct labels. The sum of this two functions is what defines the "transfer set".

3.1.4.1 Temperature

The temperature is the parameter that Hinton added to his method that allowed him to solve the Dark Knowledge issue he raised. By dividing the logits by the temperature in the softmax, he changed the formula to look like this:

$$q_i = \frac{exp(z_i/T)}{\Sigma_j exp(z_j/T)}$$

Where z_i are the logits and q_i the probability output. The temperature softens the probability values returned by the softmax, so that the values being fed to the student have better representation. He presents the change like this (19), showing the end result in a macro scale:

$$\hat{y} = (10^{-6} \quad 0.9 \quad 0.1 \quad 10^{-9})$$

$$\hat{y} = (0.05 \quad 0.3 \quad 0.2 \quad 0.005)$$

With the first distribution being done without temperature softening and the second one with temperature softening. Now, values that were too small to influence the student are meaningful and can be used for distillation. The higher the value of the temperature, the closer the probability of each possible outcome.

However, it is not so simple to define temperature as simply raising its value. If the temperature is too high, it will render the information being distilled to the student model useless, as the probability distribution will all be the same. On the other hand, using a value of temperature value that is too low will give the student a very hard time to learn the function, as the values will be too small to make a significant impact on the student's learning. For this reason, in all practical cases, temperature has to be considered a hyperparameter that needs to be tuned as a part of the training process.

The temperature choice was stated by Hinton as: "When the distilled net had 300 or more units in each of its two hidden layers, all temperatures above 8 gave fairly similar results. But when this was radically reduced to 30 units per layer, temperatures in the range 2.5 to 4 worked significantly better than higher or lower temperatures" (2). This work uses similar temperatures for analysis.

3.1.5 Transfer Set

The transfer set is the intermediate set used to train the student. This set can, in theory, be anything that helps the student to train for the problem it wants to solve. Realistically, it is some sort of derivative of the original dataset. Which changes one does to the data to transform it into the transfer set depends on the problem being tackled and what the final objective is.

These changes can be as drastic or as minute as needed, such as removing all labels of the dataset until it consists entirely of unlabeled data (2) or use a reduced number of features. That being said, it is important to point out that no change has to be done. It is possible, and sometimes preferable, to use the exact same dataset. If the main objective is to achieve the least amount of errors, it is unsurprising that research has found that using the original dataset yield better results.

In this work, as was in Hinton's, we made no changes to the entry features dataset being fed to the student, but we changed the targets by adding the logits as soft labels. This helps with the learning using the labels as true targets, as we will explain in the next section. So in our final version of the dataset being fed to the student, we have a new target, $\hat{y}$. This $\hat{y}$ is composed of the correct labels as hard targets and the logits as soft targets. The soft targets are used in knowledge distillation while the hard targets are used for training as normal after distillation is finished.

3.1.6 Cross-Entropy Loss

Cross-entropy is one of many possible loss functions. Cross-entropy is commonly used to quantify the difference between two probability distributions. Usually the true distribution, in this case the correct class, is expressed in terms of a one-hot distribution. Cross-entropy loss in represented by the following equation:

$$H(p, q) = \text{-}\Sigma_x p(x) ln(q(x))$$

Where *p(x)* is the actual probability, *q(x)* the wanted probability and x is each class.

Consider the following example: There are three possible classes, A, B and C. The correct class for the current instance is B, so the distribution is an array with values [0, 1, 0]. The model predicts the chance of being each class

in a softmax distribution with values [0.228, 0.619, 0.153], which represent A, B and C, respectively. The cross-entropy loss then is:

$$H = \text{-}(0.0 * ln(0.228) + 1.0 * ln(0.619) + 0.0 * ln(0.153)) = 0.479$$

We use this between both the real targets and soft targets, although the soft targets have a different treatment, explained next. The sum of this efforts is the distillation.

3.1.7 Logits Transfer

The logits are the information of the last layer of the neural network. Neural networks typically produce class probabilities by using a softmax output layer. A softmax function normalizes a K-dimensional vector of arbitrary real values into comparable values between (0,1) such that all values add up to 1. This means that given logits z_i, we compute for each class into a probability, q_i, by comparing z_i with the other logits.

$$q_i = \frac{exp(z_i/T)}{\Sigma_j exp(z_j/T)}$$

Where T is the temperature that is normally set to 1. Using a higher value for T produces a softer probability distribution over classes (2).

Each case in the transfer set contributes a cross-entropy gradient, $\frac{\partial C}{\partial z_i}$, with respect to each logit, z_i, of the student model. The cross-entropy gradient $\frac{\partial C}{\partial z_i}$ is the loss function we are trying to minimize, as shown in the pseudocode. $\frac{\partial C}{\partial z_i}$ represents the result of the cross-entropy between the teacher and the student, based on the input logits z_i. The teacher model has logits v_i which produce soft target probabilities p_i, p_i being a distribution similar to q_i presented before. We define that the transfer training is done at a temperature of T. This gradient is given by:

$$q_i = \frac{exp(z_i/T)}{\Sigma_j exp(z_j/T)}$$

$$p_i = \frac{exp(v_i/T)}{\Sigma_j exp(v_j/T)}$$

$$\frac{\partial C}{\partial z_i} = \frac{1}{T}(q_i\text{-}p_i) = \frac{1}{T}(\frac{exp(z_i/T)}{\Sigma_j exp(z_j/T)}\text{-}\frac{exp(v_i/T)}{\Sigma_j exp(v_j/T)})$$

If the temperature is high compared to the magnitude of the logits, we can approximate the above equation to this:

$$\frac{\partial C}{\partial z_i} \approx \frac{1}{T}(\frac{1 + z_i/T}{N + \Sigma_j z_j/T} - \frac{1 + v_i/T}{N + \Sigma_j v_j/T})$$

Where N is the number of possible classes.

We can then change the logits vector in such a way that it is zero-meaned by subtracting the mean from every element, thus giving $\Sigma_j z_j = \Sigma_j v_j = 0$. If we do that, the above equation can be simplified to:

$$\frac{\partial C}{\partial z_i} \approx \frac{1}{NT^2}(z_i - v_i)$$

So in the high temperature limit, as stated and discussed above, distillation is equivalent to minimizing $1/2(z_i - v_i)^2$. The pseudocodes 3 and 4 show the training process for both the teacher and the student, the student with distillation.

Algorithm 3 Teacher Training

```
1:  procedure TRAINING(model, features, true-class)
2:      T ← Temperature
3:      L ← Loss Function between model and true-label
4:      N ← Number of Classes
5:      X ← Features
6:      Z1 ← X * W1
7:      Z2 ← Z1 * W2
8:      Y ← Z2 * Y
9:      output ← softmax(Y)
10:     L ← cross-entropy(Y, true-class)
11:     minimize(L)
12:     return model
```

Algorithm 4 Apprentice Training

procedure TRAINING W/ DISTILLATION(model, features, teacher-logits, true-class)
    $T \leftarrow$ *Temperature*
    $C \leftarrow$ *Loss Function between Teacher and Apprentice*
    $L \leftarrow$ *Loss Function between model and true-label*
    $N \leftarrow$ *Number of Classes*
    $X \leftarrow$ *Features*
    $Z1 \leftarrow X * W1$
    $Z2 \leftarrow Z1 * W2$
    $Y \leftarrow Z2 * Y$
    $output \leftarrow softmax(Y)$
    if $output \neq true\text{-}class(i)$ **then**
        $C \leftarrow (teacher\text{-}logits(i)\text{-}Y)/(N * T^2)$.
        close;
    $L \leftarrow cross\text{-}entropy(C, true\text{-}class)$
    minimize(L)
    return model

4
The Classification Problem

This chapter describes and analyzes the classification problem, its needs and uses in modern day. Generally, classification can be defined as identifying to which of a set of categories a new observation belongs. Classification is a prime example in pattern recognition problems alongside others such as Clustering, which shares many similarities but are fundamentally different.

4.1
Pattern Recognition

Pattern recognition has its origins in engineering, whereas machine learning grew out of computer science. However, these activities can be viewed as two facets of the same field. The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories (17).

Pattern recognition was one of the first areas of machine learning and continues being developed to this day for its many uses. Face recognition software is one ubiquitous modern example, as a program trying to identify the typical facial features on an image is a form of pattern recognition. Other uses, such as trend prediction on time series analysis, are also common as pattern recognition is what allows forecasts to happen.

However, pattern recognition is a very broad area with an uncountable amount of problems and just as many methods to solve them. Hitherto, we've mentioned forecasting and facial recognition, but there's also classification. The ability to pick a new item and assigning a previously known class is very desirable. It can be used to identify new fauna and flora or extended to something similar to a forecast, where each possible outcome is a distinct class.

4.2
Binary Classification

Binary classifications are the simplest form of classification, as it only needs to check if a given object is part of a single group, and if not it defaults

to the other one. Taking a 'pass or fail' approach to classification makes models easy to make and highly efficient. One should always try to make their problem binary, if possible.

When dealing with continuous values, an arbitrary cutoff point must be chosen to split the possible results into two separate classes. The choice of this cutoff point is important, as it will greatly influence the models prediction.

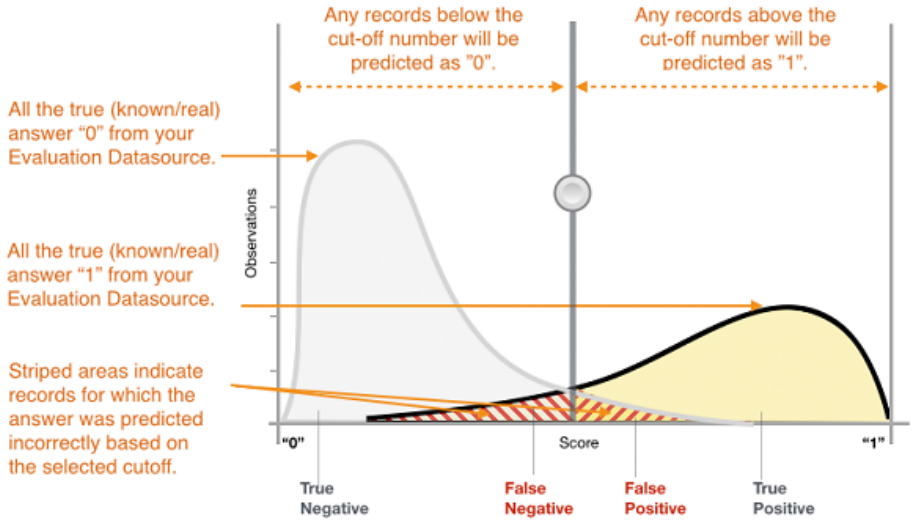

Figure 4.1: Binary Classification Distribution

4.2.1 Binary Classification Metrics

Binary classifiers have four possible outcomes: True positive, false positive, true negative and false negative, i.e. the result was accurate in it's positive classification, it gave a negative result for a positive class, the result was accurate in it's negative classification, it gave a positive response for a negative class, respectively. This is measured by two scopes, Sensitivity, or True Positive Rate (TPR), and Specificity, or True Negative Rate (TNR).

In theory, Sensitivity and Specificity are independent in the sense that it's possible to achieve 100% in both. In more practical instances, however, there's usually a trade-off, such that they are inversely proportional to one another to some extent. This is because we rarely measure the actual thing we would like to classify; rather, we generally measure an indicator of the thing we would like to classify.

It's also important to note that different problems may have different a different subset of interest of these metrics. There may be problems that strive for maximum certainty of the positive results, allowing for some misclassifica-

tion of positives as negatives if it keeps the accuracy of positive classifications high. Another problem might want to minimize false classifications altogether, not minding if the mistake is a false positive or false negative.

4.3 Multiclass Classification

Multiclass classification occurs when there are 3 or more possible classes that an instance can belong to. Not to be confused with multilabel classification, in which a single instance can have multiple classes. Multiclass and multilabel classifications can happen in a single problem, although it's quite rare.

Multiclass classification is used when a classification problem would lose information in its simplification as a binary classification or when it's impossible to make such reduction. Multiclass problems are much harder for a machine to learn and its complexity is related to the number of possible classes.

In this work, we will use multiclass classification in all datasets, as the method proposed by Hinton et al. for knowledge distillation was designed for multiclass problems.

4.3.1 Multiclass Classification Metrics

Different from binary classification, multiclass classification isn't so simple to separate into TPR and TNR. When dealing with multiclass, there are several ways that the classification can be wrong and only one to be right, so we lack the certainty on the result we have in binary. The way this uncertainty manifests itself is in the results, and we represent them in probability distributions.

A multiclass classification output presents the probability of the result being of each class. This probability distribution is then transformed into a class, with the label with highest probability being the chosen class. Multiclass also suffers a lot from imbalanced data. Since there are more possible classes, there are less representation of each one proportional to the number of classes. So even a little imbalance may cause great repercussions on the results.

4.4 Classification Using Neural Networks

Neural networks versatility make them great for classification. Defining the scope of the neural network is easy: the input layer has as many neurons

as there are features to analyze; The amount of hidden layers and the size of each depends of the problem; The output layer should represent the different classes.

In binary classifiers, the output layer should be a single neuron, which can represent positive and negative responses with its activation. In multiclass classifiers, multiple neurons are used on the output layer, each representing a single class. In this instance, the result vector is a softmax distribution, with the highest value being the most likely class.

It's worth mentioning that it's possible to create a binary classifier with 2 neurons on the output layer. However, this increases the complexity of the neural network and lowers the average success.

4.5 Classification as Forecasting

It's possible to use classification models as forecasting models. A normal approach for forecasting is to try to predict a value of a desired feature given it's previous values and/or other features. This is normally used in time series forecasting, with models such as Holts-Winter and SARIMA.

However, this same forecasting with continuous values can be made discrete. These discrete values can then be turned into different classes, turning the problem into a classification problem. In this work, the Flight Delay dataset is a forecasting problem, but it can be made into a discrete classification problem. This is made possible by the final objective of the data to predict delays. Since there's no need to give the real-valued output of the delay, only tell its window of delay, this discrete classification approach is feasible.

5
Methodology

In this chapter we present the methodology used to build the constructed dataset and classification algorithms.

5.1
Datasets

To test the viability of our method, we applied it in 3 datasets: 'MNIST', 'Fashion MNIST' and '2015 Flight Delays and Cancellations'. MNIST was also used by Hinton, so will serve as good baseline for comparison; Fashion MNIST is a new dataset very similar to MNIST, but using clothing to increase difficulty; and the Flight Delay dataset, which is a real dataset that we can use to verify its application in real life scenarios.

5.1.1
MNIST

The Modified National Institute of Standards and Technology, or MNIST[1], dataset is one of the most diffused dataset in modern machine learning. Members of the AI/ML/Data Science community love this dataset and use it as a benchmark to validate their algorithms. In fact, MNIST is often the first dataset researchers try. It is commonly said among the community that "If it doesn't work on MNIST, it won't work at all" and "Well, if it does work on MNIST, it may still fail on others."

The dataset is composed of 70000 handwritten digits, 60000 for training and 10000 for testing. All images are size normalized, centered to the best of the creators ability, gray-scaled and 28x28 pixels. The digits are numbers from 0 to 9, as shown in Figure 5.1, with 10 classes total.

5.1.1.1
Preprocessing

To transform the image into entry features, the image was flattened into a 1-dimension array with size 784. Each value on the array represented the value of a single pixel, which originally was an integer between 0 and 255.

[1]http://yann.lecun.com/exdb/mnist/

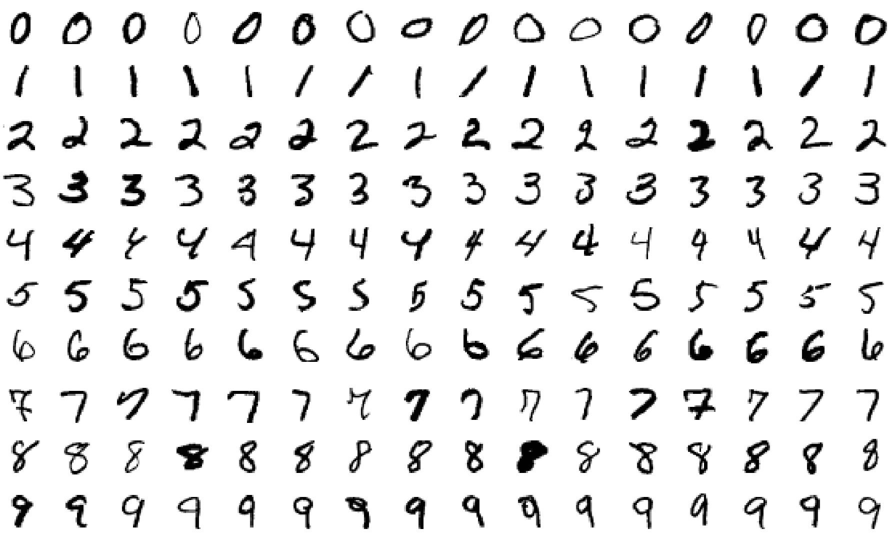

Figure 5.1: MNIST Examples

However, to better work with the neural network, the value was normalized between 0 and 1 by dividing each one by 255. A visual representation of this array of the number 5 before normalization can be seen on Figure 5.2.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0
0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0
0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0	0	0
0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	35	241	225	160	108	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5.2: MNIST Array Representing Number 5

The label of each entry is the digit represented on the image. We change this label into a one-hot array with the active bit being the correct answer, so we create an array with size 10, every value 0 except the correct value as 1, e.g. if the label is 3, the array would be [0,0,0,1,0,0,0,0,0,0]. The distribution of each label is perfectly balanced, with 6000 training examples and 1000 test examples for each, as shown in Figure 5.3.

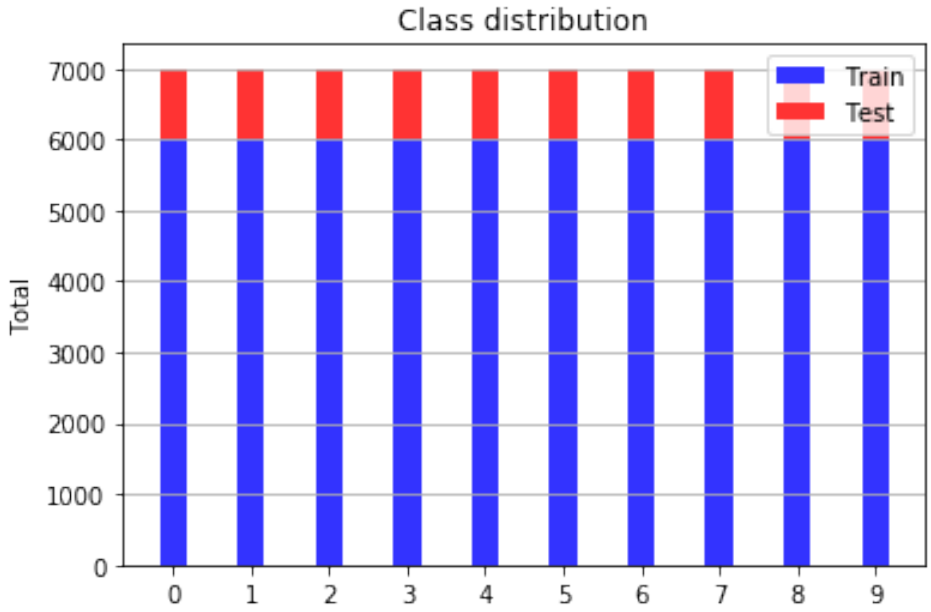

Figure 5.3: MNIST Class Distribution

5.1.2 Fashion MNIST

Fashion MNIST is a dataset of Zalando's[2] article images created by Xiao et al. (20). This dataset was made with the sole intention of substituting the original MNIST as the go-to dataset for benchmarking a ML model. It was created with increased difficulty to compensate the growing accuracy of newer models.

The researchers cite that the MNIST dataset has become outdated, being too easy to solve and overused over the 20 years of its existence. In their efforts to push this new dataset, they made it very similar in composition to the original MNIST, consisting of a training set of 60000 examples and a test set of 10000 examples of 28x28 pixel gray-scaled images. These images show 10 different articles of clothing, each one with its label, totaling 10 different labels.

5.1.2.1 Preprocessing

Since the two datasets are so similar, the preprocessing was identical. We flattened the image into an array and normalized each value to be between 0 and 1. A visual representation of this array of an ankle boot before normalization can be seen on Figure 5.5.

The labels received a similar treatment, being transformed into a one-hot array. The labels before transformation associated with each type of apparel

[2]https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/

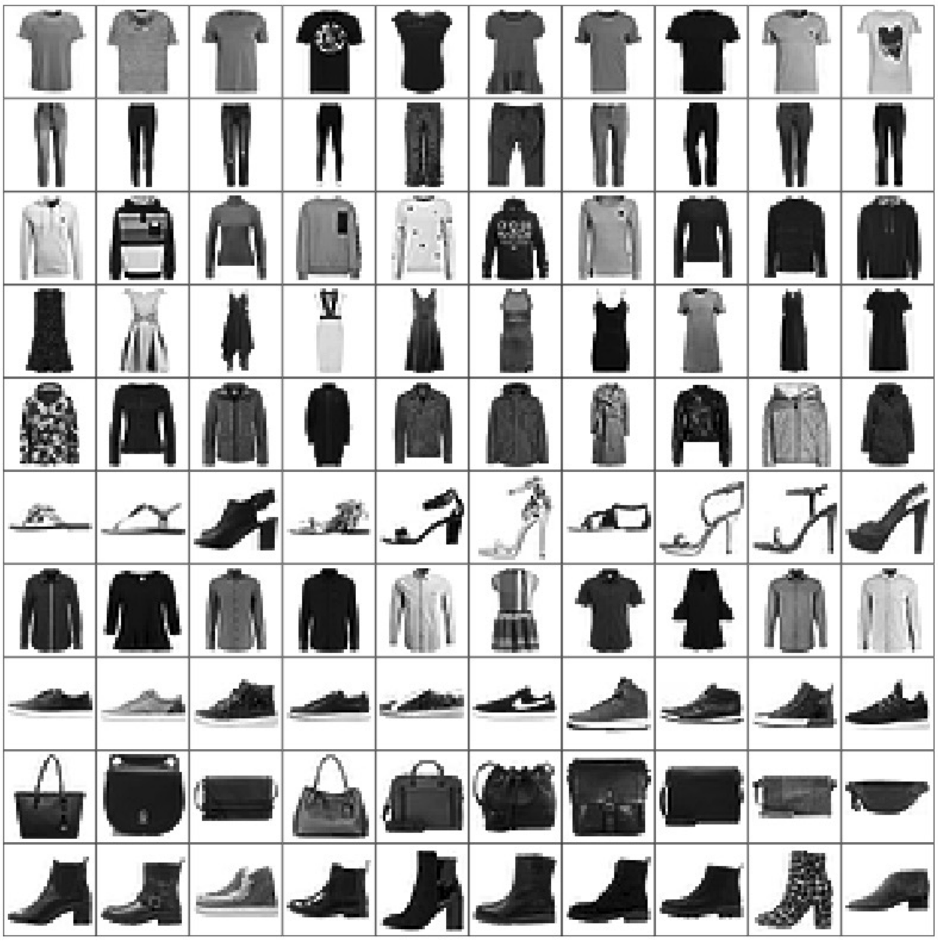

Figure 5.4: MNIST Fashion Examples

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	13	73	0	0	1	4	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	3	0	36	136	127	62	54	0	0	0	1	3	4	0	0	3
0	0	0	0	0	0	0	0	0	0	0	0	6	0	102	204	176	134	144	123	23	0	0	0	0	12	10	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	155	236	207	178	107	156	161	109	64	23	77	130	72	15
0	0	0	0	0	0	0	0	0	0	0	1	0	69	207	223	218	216	216	163	127	121	122	146	141	88	172	66
0	0	0	0	0	0	0	0	0	1	1	1	0	200	232	232	233	229	223	223	215	213	164	127	123	196	229	0
0	0	0	0	0	0	0	0	0	0	0	0	0	183	225	216	223	228	235	227	224	222	224	221	223	245	173	0
0	0	0	0	0	0	0	0	0	0	0	0	0	193	228	218	213	198	180	212	210	211	213	223	220	243	202	0
0	0	0	0	0	0	0	0	0	1	3	0	12	219	220	212	218	192	169	227	208	218	224	212	226	197	209	52
0	0	0	0	0	0	0	0	0	0	6	0	99	244	222	220	218	203	198	221	215	213	222	220	245	119	167	56
0	0	0	0	0	0	0	0	0	4	0	0	55	236	228	230	228	240	232	213	218	223	234	217	217	209	92	0
0	0	1	4	6	7	2	0	0	0	0	0	237	226	217	223	222	219	222	221	216	223	229	215	218	255	77	0
0	3	0	0	0	0	0	0	0	62	145	204	228	207	213	221	218	208	211	218	224	223	219	215	224	244	159	0
0	0	0	0	18	44	82	107	189	228	220	222	217	226	200	205	211	230	224	234	176	188	250	248	233	238	215	0
0	57	187	208	224	221	224	208	204	214	208	209	200	159	245	193	206	223	255	255	221	234	221	211	220	232	246	0
3	202	228	224	221	211	211	214	205	205	205	220	240	80	150	255	229	221	188	154	191	210	204	209	222	228	225	0
98	233	198	210	222	229	229	234	249	220	194	215	217	241	65	73	106	117	168	219	221	215	217	223	223	224	229	29
75	204	212	204	193	205	211	225	216	185	197	206	198	213	240	195	227	245	239	223	218	212	209	222	220	221	230	67
48	203	183	194	213	197	185	190	194	192	202	214	219	221	220	236	225	216	199	206	186	181	177	172	181	205	206	115
0	122	219	193	179	171	183	196	204	210	213	207	211	210	200	196	194	191	195	191	198	192	176	156	167	177	210	92
0	0	74	189	212	191	175	172	175	181	185	188	189	188	193	198	204	209	210	210	211	188	188	194	192	216	170	0
2	0	0	0	66	200	222	237	239	242	246	243	244	221	220	193	191	179	182	182	181	176	166	168	99	58	0	0
0	0	0	0	0	0	0	40	61	44	72	41	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5.5: Fasion MNIST Array Representing an Ankle Boot

are shown in Table 5.1. The distribution of each label is, once again, perfectly balanced, with 6000 training examples and 1000 test examples for each, as shown in Figure 5.6.

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Table 5.1: Fashion MNIST labels

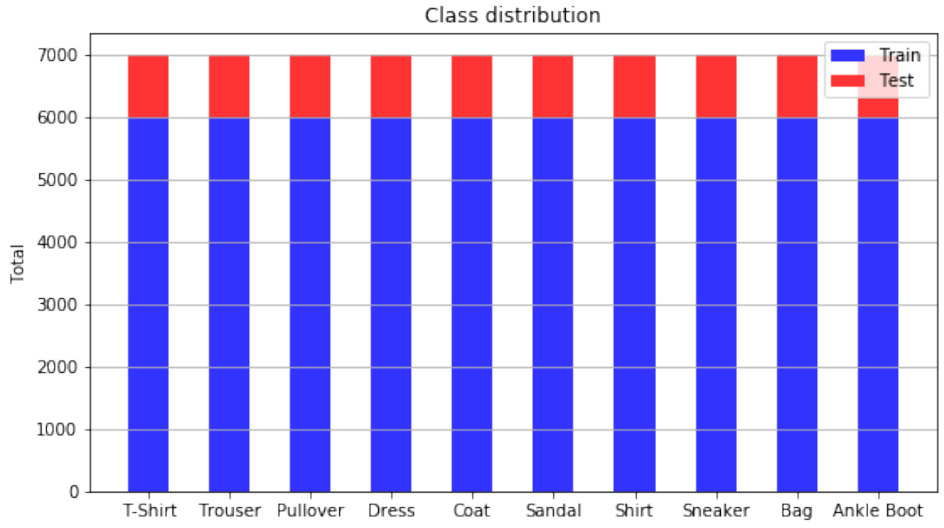

Figure 5.6: Fashion MNIST Class Distribution

5.1.3 2015 Flight Delays and Cancellations

The final dataset used in this work is the kaggle dataset '2015 Flight Delays and Cancellations'[3] worked on to better be analyzed by the neural network. As per the dataset's description: "The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights is published in DOT's monthly Air Travel Consumer Report and in this dataset of 2015 flight delays and cancellations."

There are 3 *csv* files provided in the dataset: *airlines.csv*, that relates the company's IATA code to its name; *airports.csv*, that has data related to the airports; *flights.csv*, that has data related to the flights itself and composes

[3]https://www.kaggle.com/usdot/flight-delays

the bulk of the dataset. The metadata of each file and its components are presented on tables Table 5.2, Table 5.3, Table 5.4, respectively.

Feature	**Information**	**Type**
`IATA_CODE`	Airline Identifier	String
AIRLINE	Airline Identifier	String

Table 5.2: *airlines.csv* Metadata

Feature	**Information**	**Type**
`IATA_CODE`	Location Identifier	String
AIRPORT	Airport's Name	String
CITY	City Name of the Airport	String
STATE	State Name of the Airport	String
COUNTRY	Country Name of the Airport	String
LATITUDE	Latitude of the Airport	Numeric
LONGITUDE	Longitude of the Airport	Numeric

Table 5.3: *airports.csv* Metadata

Feature	Information	Type
YEAR	Year of the Flight Trip	Numeric
MONTH	Month of the Flight Trip	Numeric
DAY	Day of the Flight Trip	Numeric
DAY_OF_WEEK	Day of Week of the Flight Trip	Numeric
AIRLINE	Airline Identifier	String
FLIGHT_NUMBER	Flight Identifier	Numeric
TAIL_NUMBER	Aircraft Identifier	String
ORIGIN_AIRPORT	Starting Airport	String
DESTINATION_AIRPORT	Destination Airport	String
SCHEDULED_DEPARTURE	Planned Departure Time	Numeric
DEPARTURE_TIME	WHEELS_OFF - TAXI_OUT	Numeric
DEPARTURE_DELAY	Total Delay on Departure	Numeric
TAXI_OUT	The time duration elapsed between departure from the origin airport gate and wheels off	Numeric
WHEELS_OFF	The time point that the aircraft's wheels leave the ground	Numeric
SCHEDULED_TIME	Planned time amount needed for the flight trip	Numeric
ELAPSED_TIME	AIR_TIME + TAXI_IN + TAXI_OFF	Numeric
AIR_TIME	The time duration between wheels_off and wheels_on time	Numeric
DISTANCE	Distance between two airports	Numeric
WHEELS_ON	The time point that the aircraft's wheels touch on the ground	Numeric
TAXI_IN	The time duration elapsed between wheels-on and gate arrival at the destination airport	Numeric
SCHEDULED_ARRIVAL	Planned arrival time	Numeric
ARRIVAL_TIME	WHEELS_ON + TAXI_IN	Numeric
ARRIVAL_DELAY	ARRIVAL_TIME - SCHEDULED_ARRIVAL	Numeric
DIVERTED	Aircraft landed on airport that was out of schedule	Numeric
CANCELLED	Flight Cancelled (1 = cancelled)	Numeric
CANCELLATION_REASON	Reason for Cancellation of flight	String
AIR_SYSTEM_DELAY	Delay caused by air system	String
SECURITY_DELAY	Delay caused by security	String
AIRLINE_DELAY	Delay caused by the airline	String
LATE_AIRCRAFT_DELAY	Delay caused by the aircraft	String
WEATHER_DELAY	Delay caused by the weather	String

Table 5.4: *flights.csv* Metadata

5.1.3.1 Preprocessing

The *airlines.csv* file contains only redundant data and has been discarded in its entirety. The only information of interest for us on *airports.csv* is the latitude and longitude, so it was joined with the information on the *flights.csv* using the airport code as a key.

Finally, we worked on *flights.csv* until the features were prime for learning. First, every flight that was cancelled or redirected did not contain information on the delay, and so they were removed. Every feature with identification information, such as flight number, were removed since they had no relevant information for learning. Features with categorical information, such as destination airport, were transformed into **n** binary features, where **n** is the number of categories of the original feature. The remaining numeric features were normalized with zero mean and unit variance, also called standardization.

The final dataset is composed of a total of 72 features, of which 54 are binary categorization and 18 are numeric. Also, for each timestep, the features represent a window of 15 minutes.

The target variable of the problem at hand is the *ARRIVAL_DELAY* of each flight, from which we derive the label for each entry of the dataset transformed into a one-hot array, as the previous examples, i.e., the label representing the first time window of extreme earliness would be an array with values [1,0,0,0,0]. The cutoff was -16, -9, -1 and 15 minutes for the multiclass classifier, separating into 5 distinct classes with no overlay.

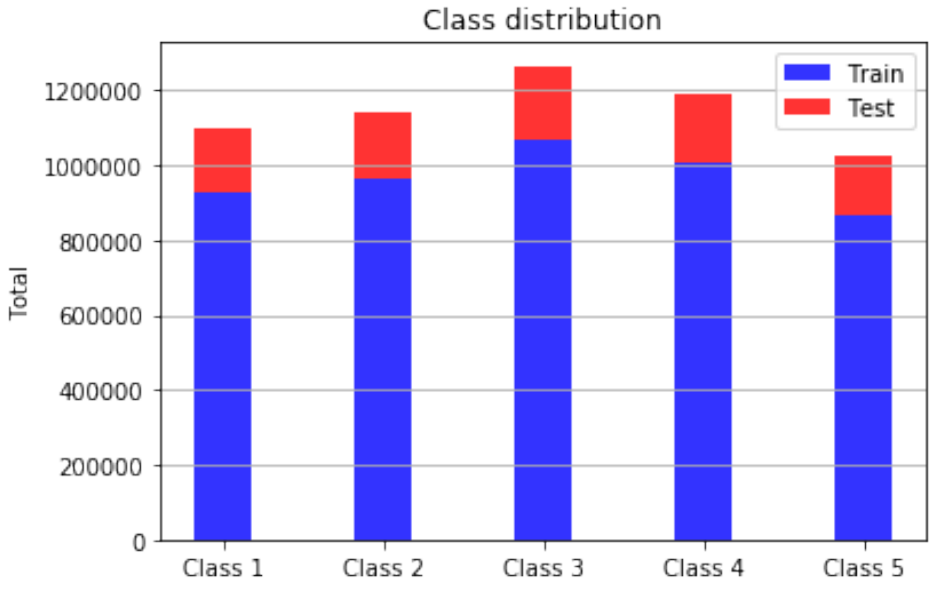

Figure 5.7: Flight Delay Class Distribution

The multiclass partition of the dataset was designed to be balanced among all 5 classes. The numbers were chosen arbitrarily and have no real significance besides being the integers that best divide the dataset. This was done because we're more interested in testing the capabilities of knowledge distillation in several scenarios rather than use a completely real example.

The multiclass class distribution shown on Figure 5.7 was designed to have a more balanced distribution for better learning.

5.2 Classifier Construction

The teacher model is a neural network with two hidden layers of 1200 neurons each, an input layer with the right size for all features and an output layer. All student models are also neural networks with two hidden layers, but they have varying sizes in the experimentation. We tackle the problem of delay prediction as a classification problem by predicting the value of the delay and attributing a class pertaining to that value.

The chosen machine learning algorithm was a Neural Network (NN). This algorithm was developed using TensorFlow (18).

6
Experimental Evaluation

In this chapter we present the results of the proposed models and how the accuracy of the models were represented by the student models versus the same models without a teacher.

The experiments were evaluated on different sizes of neural networks. All of them have the same basis: an input layer, followed by two hidden layers and, finally, an output layer. The teacher model had 1200 neurons on each of its hidden layers. The student models were as following: 100 per hidden layer; 200 per hidden layer; 400 per hidden layer; 800 per hidden layer; and 1200 per hidden layer. We used the same machine to train all models, which was a Intel(R) Core(TM) i7-5960X CPU @ 3.00GHz, 64GB RAM and a Nvidia Tesla K40c GPU.

6.1
MNIST Results

Below, in Table 6.1, we show the achieved accuracy in both train and test sets, with and without distillation. The best results between the same model with and without distillation are highlighted with bold face. The training times were similar to the binary classifier. In Table 6.1, we show the amount of time required to train each model with distillation, comparing our method with Hinton's method.

Neurons	**Hinton's Method**	**Teacher-Student Relation**
100	0:40:47 hours	0:27:20 hours
200	1:07:30 hours	0:45:06 hours
400	1:55:50 hours	1:19:22 hours
800	5:02:04 hours	4:30:59 hours
1200	9:47:15 hours	9:10:47 hours

Table 6.1: MNIST Performance Results

It is possible to observe the principle of knowledge distillation in the smaller models, as their result improves heavily when under the influence of the teacher. This happens because the smaller models makes so many mistakes, the information is distilled constantly, much like what happens in model compression proposed by Hinton. Notwithstanding, the MNIST dataset

Status	Neurons		Train Acc.	Test Acc.
Teacher	1200		99.82%	99.80%
Student	100	Without Teacher	85.77%	84.50%
		With Teacher	97.29%	**95.72%**
	200	Without Teacher	94.11%	93.87%
		With Teacher	98.63%	**98.39%**
	400	Without Teacher	99.76%	99.75%
		With Teacher	99.77%	**99.77%**
	800	Without Teacher	99.78%	**99.74%**
		With Teacher	99.79%	99.73%
	1200	Without Teacher	99.83%	99.80%
		With Teacher	99.82%	**99.82%**

Table 6.2: MNIST Accuracy Results

is so simple that our model accuracy reached a ceiling on 400 neurons. The increase in accuracy is dubious after that and we cannot observe knowledge distillation working. The gain from the teacher is negligible on the bigger models and don't expose any new information.

We can also see that the performance follows a growth proportional to the size of the student. We can see the gain easily since the student model already achieves a result similar to the teacher so early, our method is seldom called. This means that the time of training with our method is basically the time to train the model by itself. This shows that calling it only on incorrect labeling already displays a considerable performance gain.

6.2 Fashion MNIST Results

Below, in Table 6.2, we show the achieved accuracy in both train and test sets, with and without distillation. The best results between the same model with and without distillation are highlighted with bold face. The training times were similar to the binary classifier. In Table 6.2, we show the amount of time required to train each model with distillation, comparing our method with Hinton's method.

Neurons	Hinton's Method	Teacher-Student Relation
100	0:42:20 hours	0:37:20 hours
200	1:09:11 hours	0:55:33 hours
400	2:01:14 hours	1:39:02 hours
800	5:00:53 hours	4:36:59 hours
1200	9:51:29 hours	9:25:08 hours

Table 6.3: Fashion MNIST Performance Results

Status	Neurons		Train Acc.	Test Acc.
Teacher	1200		86.51%	84.23%
Student	100	Without Teacher	64.54%	62.54%
		With Teacher	80.40%	**79.00%**
	200	Without Teacher	74.44%	73.29%
		With Teacher	81.93%	**80.49%**
	400	Without Teacher	80.03%	78.96%
		With Teacher	83.55%	**82.37%**
	800	Without Teacher	83.84%	82.61%
		With Teacher	85.72%	**84.42%**
	1200	Without Teacher	86.49%	84.28%
		With Teacher	86.80%	**85.10%**

Table 6.4: Fashion MNIST Accuracy Results

The results on the Fashion MNIST data were worse than on the original MNIST, as was expected. Rather, now it is possible to see knowledge distillation in action in the results. Once again, the smaller models had great gain when trained with a teacher, but now we can observe the bigger models in action. The bigger models had better results without teacher with growth related to the size of the model. When trained with a teacher, it is easy to see the improvements until we reach two models of the same size. When dealing with models of the same size, there is a small gain on the student model over the teacher model. So with a more difficult problem, the principle holds true.

The performance was almost identical to the original MNIST for Hinton's method, which makes sense since the datasets are basically the same, with the same amount of input and information. However, now we can analyze better the scaling of our solution. It is clear that on smaller models, the gain in performance from Hinton's method is minimal, since the teacher is being consulted just as frequently. However, as the student model grows, our method's performance gets better as the teacher's presence becomes more scarce. The gain is smaller than the gain on the original MNIST since the teacher is being called more frequently, but it is still considerable.

6.3 Flight Delays Results

Below, in Table 6.3, we show the achieved accuracy in both train and test sets, with and without distillation. The best results between the same model with and without distillation are highlighted with bold face. The training times were similar to the binary classifier. In Table 6.3, we show the amount of time required to train each model with distillation, comparing our method with Hinton's method.

Neurons	Hinton's Method	Teacher-Student Relation
100	0:53:03 hours	0:51:08 hours
200	1:37:14 hours	1:32:42 hours
400	2:57:50 hours	2:48:22 hours
800	5:53:15 hours	5:38:28 hours
1200	10:38:15 hours	10:20:47 hours

Table 6.5: Flight Delays Performance Results

Status	Neurons		Train Acc.	Test Acc.
Teacher	1200		83.02%	74.17%
Student	100	Without Teacher	52.54%	43.33%
		With Teacher	75.40%	**64.72%**
	200	Without Teacher	63.12%	53.99%
		With Teacher	75.63%	**64.95%**
	400	Without Teacher	70.56%	60.87%
		With Teacher	77.55%	**67.76%**
	800	Without Teacher	78.19%	69.51%
		With Teacher	80.60%	**71.24%**
	1200	Without Teacher	83.12%	74.18%
		With Teacher	83.98%	**75.14%**

Table 6.6: Flight Delays Accuracy Results

This dataset was the hardest one for the models to learn. The dropout between learn and test is the biggest of the 3 datasets, but the knowledge distillation is still visible. We can also see the limitation of distillation on the student models. On the 100 and 200 neurons models, the learning without teacher produce growing results, but the learning with teacher are very similar. This highlights the models limitations of assimilating the teacher's information. After that, it continues growing as the more robust models have the capability to learn more from the distillation. When the teacher and student model are the same size, there is once again a small gain in accuracy.

The performance this time was closer than with the MNIST sets, which correlates to the total success of the student set. The lower the overall accuracy of the student, the smaller the gap in performance since it will call the teacher for assistance more. Nevertheless, there is still performance gain.

6.4 Overall Results

Under distillation, every single model but one performed better, with this one being with MNIST that had already achieved peak result. While this bodes well for the theory behind the teacher-student relation and knowledge distillation as a whole, it does not speak of the cost behind this increase.

Two separate models were trained, using resources for a marginal increase in accuracy. For model compression, this makes sense since the gain is great. Hinton's idea was to approach this subject for model compression, so the increased cost was acceptable. When the models grow to the same size, however, the gain is minimal and the cost versus gain equation does not maintains itself.

In summary, our results are consistent with the suggestion that the teacher-student relation improves the results using information obtained by a model and applying logits transfer to another model. Although we had performance gains over Hinton's original method, this performance gain was small with smaller student models. With same sized models where the performance is considerably better, the cost of this operation is still high and the improvement in accuracy too little to be considered a extensively viable application to improve stagnated models. However, the theory is solid and future works can improve on it.

7
Conclusions

In this final chapter, we conclude by describing the progress made towards knowledge distillation and how to approach this problem. Also, we suggest some future research directions that could provide the next steps along the path to better develop this technique.

The initial objective of this work was to showcase another approach to knowledge distillation that doesn't have model compression as its end goal. With the discussion to make machine learning models more human-like or with better human-like behaviour, trying to replicate human learning on machines is an obvious step to take and explore.

Our results are consistent with the suggestion that using an already existing, robust model to acquire more data for training other models is viable. Using the end result data from a teacher model, we observe a slightly increase in accuracy of our classification models. Our results suggest that these results were achieved with the distillation of logits between the neural network models.

The quality of the proposed methodology has been tested on several datasets and it has been shown that the results that we have obtained are satisfactory, thus giving some validity of our approach to the teacher-student relation.

Even though we tried our best to provide a complete analysis of the teacher-student relation proposed at the beginning, it will be of sure interest to delve into many aspects of the methodology. The first thing would be to apply the same methodology to different types of problems besides classification. Another aspect that may deserve attention could be that of its efficacy in recurrent neural network models, such as LSTMs, since they already have a recurrence of data planned on its design.

We strongly believe that the idea of using machine learning models in conjunction with each other in this format can increase the performance of models even though it requires remarkable efforts. We conclude that these results further illustrate the exciting possibilities offered by new big datasets to further push our approach to learning data.

This work has opened lots of branches for future research. The first obvious one is to use this method recursively to see if the results continue to

improve and, if so, how much until it reach a ceiling. Also, the use of different datasets, such as time series, remain to be studied.

The experiments were focused on verifying the theory behind knowledge distillation, and not to reach the best possible result on the dataset. It's possible that some approach was overlooked on this work or the features chosen were not optimal, even after extensive research and testing. This optimizations may change the results or reveal new information, which may provide crucial improvements in the method.

Finally, a teacher model with perfect knowledge would be ideal, as we could guarantee that it's training the students correctly. A teacher model that makes a mistake will also instruct the student to make the same mistake when the distillation happens. However, this would be too expensive to train in terms of time and computer processing and will be left for future projects.

Bibliography

[1] CARUANA, R.; BUCILA, C. ; NICULESCU-MIZIL, A.. **Model compression**. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, p. 535–541, 2006.

[2] HINTON, G.; VINYALS, O. ; DEAN, J.. **Distilling the knowledge in a neural network**. CoRR, abs/1503.02531, 2015.

[3] URBAN, G.; GERAS, K. J.; KAHOU, S. E.; ASLAN, O.; WANG, S.; CARUANA, R.; MOHAMED, A.; PHILIPOSE, M. ; RICHARDSON, M.. **Do deep convolutional nets really need to be deep and convolutional**. CoRR, abs/1603.05691, 2016.

[4] GUPTA, S.; HOFFMAN, J. ; MALIK, J.. **Cross modal distillation for supervision transfer**. CoRR, abs/1507.00448, 2016.

[5] ZAGORUYKO, S.; KOMODAKIS, N.. **Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer**. CoRR, abs/1612.03928, 2017.

[6] LOPES, R. G.; FENU, S. ; STARNER, T.. **Data-free knowledge distillation for deep neural networks**. CoRR, abs/1710.07535, 2017.

[7] **A proposal for the dartmouth summer research project on artificial intelligence**. `http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html`. Último acesso em 05/01/2017.

[8] RUSSEL, S. J.; NORVIG, P.. **Artificial Intelligence A Modern Approach**. Prentice Hall, first edition, 1995.

[9] FERRUCCI, D.; BROWN, E.; CHU-CARROLL, J.; FAN, J.; GONDEK, D.; KALYANPUR, A. A.; LALLY, A.; MURDOCK, J. W.; NYBERG, E.; PRAGER, J.; SCHLAEFER, N. ; WELTY, C.. **Building watson: An overview of the deepqa project**. AI Magazine, 31(3):59–79, 2010.

[10] DOMINGOS, P.. **A few useful things to know about machine learning**. Commun. ACM, 55(10):78–87, Oct. 2012.

[11] SIMPSON, P. K.. **Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations**. Pergamon Pr, 1990.

[12] ROSENBLATT, F.. **The Perceptron: A Probabilistic Model For Information Storage and Organization in the Brain**. Psychological Review, 1958.

[13] MCCULLOCH, W. S.; PITTS, W.. **A Logical Calculus of the Ideas Immanent in Nervous Activity**. Bulletin of Mathematical Biology, 1943.

[14] MINSKY, M.; PAPERT, S.. **Perceptrons: An Introduction to Computational Geometry**. MIT Press, Cambridge, MA, USA, 1969.

[15] RUMELHART, D. E.; HINTON, G. E. ; WILLIAMS, R. J.. **Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1**. chapter Learning Internal Representations by Error Propagation. MIT Press, Cambridge, MA, USA, 1986.

[16] HINTON, G. E.; OSINDERO, S. ; TEH, Y.-W.. **A fast learning algorithm for deep belief nets**. Neural Computation, 2006.

[17] BISHOP, C. M.. **Pattern Recognition and Machine Learning**. Springer, Cambridge CB3 0FB, U.K, 2006.

[18] ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIÉGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y. ; ZHENG, X.. **TensorFlow: Large-scale machine learning on heterogeneous systems**, 2015. Software available from tensorflow.org.

[19] HINTON, G.; VINYALS, O. ; DEAN, J.. **Dark knowledge**. Talk given at TTIC Distinguished Lecture Series, 2014.

[20] XIAO, H.; RASUL, K. ; VOLLGRAF, R.. **Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms**, 2017.