Universidade do Minho
Escola de Engenharia
Departamento de Informática

Gonçalo José Azevedo Esteves

Functional Programming for Explainable AI

February 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Gonçalo José Azevedo Esteves

Functional Programming for Explainable AI

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
José N. Oliveira (U.Minho)

February 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Neural Networks (NN), which are increasingly used in Artificial Intelligence, are computational representations inspired by the existing biological neural systems, seeking to be capable of learning how to perform tasks and recognize complex patterns. Internally, NN programs are made up of several small structures, called neurons (by analogy with Biology) which are responsible for handling input values in order to determine what the output values should be. The fact that these programs are organized hierarchically makes it plausible applying compositional patterns, often associated with functional programming, in order to obtain more refined neuronal networks, and whose understanding would be easier.

This document is presented as a pre-dissertation report for a master's degree thesis that intends to focus more on the possibility of using the high compositionality presented in functional languages, namely the Haskell, in order to make Neural Networks programming better strucctured and elegant, facilitating not only the creation but also the understanding of what goes on inside these systems, which are sometimes seen as black boxes, due to the great lack of knowledge about how they work.

KEYWORDS Artificial Intelligence, Neural Networks, Compositionality, Functional Programming, Haskell

RESUMO

No estudo da Inteligência Artificial é comum serem várias vezes referenciadas um conjunto de estruturas designadas de Redes Neuronais. Estas nada mais são do que representações computacionais inspiradas nos sistemas nervosos biológicos existentes, e que procuram tornar-se capazes de aprender não só a executar tarefas como também a reconhecer diferentes padrões, por exemplo. Internamente, estes programas são constituídos por diversas pequenas estruturas, chamadas de neurónios, tendo em conta a analogia estabelecida com a Natureza, e que são responsáveis por tratar valores de entrada de modo a determinar quais os valores de saída. O facto destes programas possuírem uma organização tão hierarquizada torna plausível a possibilidade de aplicar padrões composicionais, muitas vezes associados a programação funcional, de modo a obter Redes Neuronais mais refinadas, e cuja compreensão seja mais fácil.

Apresenta-se este documento como relatório de pré-dissertação de uma tese de mestrado que pretende debruçar-se mais sobre a possibilidade de utilização da elevada composicionalidade presente em linguagens funcionais, nomeadamente o Haskell, de forma a tornar a programação de Redes Neuronais num processo mais simples e elegante, facilitando não só a criação mas também o entendimento daquilo que se passa dentro destes sistemas, que por vezes são vistos como caixas negras, devido ao grande desconhecimento sobre a forma como funcionam.

PALAVRAS-CHAVE Dissertação de Mestrado, Programação Funcional, Composicionalidade, Inteligência Artificial, Redes Neuronais, Haskell

CONTENTS

LIST OF FIGURES

ACRONYMS

- **AI** - Artificial Intelligence
- **DL** - Deep Learning
- **FP** - Functional Programming
- **ML** - Machine Learning
- **NN** - Neural Networks
- **RL** - Reinforcement Learning
- **SL** - Supervised Learning
- **UL** - Unsupervised Learning
- **XAI** - Explainable Artificial Intelligence

1

INTRODUCTION

"(...) ML systems have been highly resistant to compositional description and analysis. However, there is a major push in current research to achieve this, in order to have explainable, verifiable and accountable AI."

Samson Abramsky (2022)

Despite all recent advances regarding Machine Learning (ML) and Neural Network (NN), the truth is that there is still a long path to follow before these disciplines are scientifically explainable. Concerns about this state of affairs have already reached public opinion, as newspaper articles such as the one published very recently by Spinney (2022) in The Guardian, show. People are concerned about programs that are used in public services and seem to work, but no one knows exactly how.

ML techniques are more and more used in critical applications, whereby software development shifts from traditional coding to example-based training. This raises the need for achieving confidence in ML modeling through new forms of verification and validation and through explainable AI (XAI) techniques.

Several safety standards (e.g., ISO 26262, IEC 62304, EN50128, DO-178C) and security standards (e.g., Common Criteria, ISO/IEC 15408, NIST Publication 800-53), which have matured over the years, are thus facing new challenges. The work to be carried out in this project is triggered by the following research questions:

1. What "is" Machine Learning?
2. How explainable is it?
3. How compositional is it?

In particular, this project proposal arises from an interesting conjecture by Christopher Olah (2015) in his research blog:

> *Strongly typed functional programming (FP) can play an important role in XAI.*

According to Olah, at present three narratives are competing with each other to become *the easy way* for one to understand deep learning (DL):

- the neuroscience narrative,

- the probabilistic narrative and
- the representations narrative.

Data transformation is formalizable in a rigorous and calculational way (Oliveira, 2008) and Olah's conjecture suggests that the representation narrative can be easily associated to functional programming's type theory. Besides this, many models in DL, if not all of them, can be depicted as optimization problems related to function composition.

Considering that both type theory and the idea of function composition are some of the founding stones of FP (Bird and de Moor, 1997) and that such notions can be intuitively associated to other DL notions, we are led to believe that these two areas can be effectively help each other, in particular leading to better understanding neural networks (NN).

1.1 MAIN GOALS

The lack of explainability in ML techniques is somewhat in contradiction with the heavy use of maths (linear algebra, in particular), which should shed some light on how ML programs work.

If we take into consideration the fact that this sort of technology is being increasingly more used in new applications, and many of them require high assurance, like those that are used in banks or medical necessities, we can easily understand why this sort of problems must be overcome, since they can put human lives at risk.

Considering our starting research questions, we propose ourselves to focus on compositionality of neural networks, in order to prove the third question, that ML can actually be compositional (in part, at least). With this, we also hope to help in the understanding of the second one (how explainable can ML be). The first one we understand that is something that one can not explain easily, since after all these years no one was capable of answering it.

The overall goal of this study is to understand how far DL and FP can go together, in order to (possibly) surpass some of the problems that NN present to us, namely lack of confidence, using the techiques and calculi that FP provides us with (Bird and de Moor, 1997).

Validating Olah's conjecture is the main aim of this dissertation. The experimental part of the project will be carried out in Haskell, including the use of a Haskell library for typed linear algebra developed in a previous MSc project (Santos and Oliveira, 2020).

2

STATE OF THE ART

This chapter is devoted to a state-of-the-art review of the literature on ML, while analyzing what is being currently researched in its intersection with FP.

The discussion will start by reviewing *supervised*, *unsupervised* and *reinforcement learning*, indicating the pros and cons of each approach and their applications. This is followed by dissecting why XAI is a hot topic and the reasons that make us regard it as an interesting subject for research in applied functional programming. Last but not least, we will present some previous work on how ML and FP can be combined. But, before all that, let us just make a quick review of the three narratives that were referred to previously.

- **Neuroscience Narrative** - draws analogies with Biology, comparing the functioning of networks with how biological neurons work;
- **Probabilistic Narrative** - relates neural networks to finding latent variables;
- **Representations Narrative** - is based on data transformations and the manifold hypothesis.

As stated above, we will center our work in the third narrative, since it is the one that offers more correlation to FP (Olah, 2015).

2.1 SUPERVISED LEARNING

Supervised Learning (SL) is an aproach to ML that is characterized by its use of labeled training datasets (training sets that have inputs associated to their expected outputs) in order to train algorithms to better classify data or determine the desired outputs (IBM Cloud Education, 2020a). With this, the model is able to adjust its weights so it can make correct predicitions, using a loss function to determine if the error has been sufficiently minimized or if there are more changes needed in order to raise its accuracy.

The problems addressed by this strategy can be separated into two different types: *classification* problems and *regression* problems. The former involve the recognition of entries, trying to decide how they should be labeled or classified, using for that algorithms that assign such entries to specific categories. This can be applied, for example, to the automatic association of messages received in an inbox to their respective folders (important, spam, ...). The latter are related to understanding how dependent and independent variables are connected. Projections, such as sales revenue for a given business, are some of the possible problems that can be aproached.

Despite of all of the advantages that SL offers, like deep data insights and improved automation, challenges arise in order to build sustainable models. These can require high levels of expertise and knowledge to structure accurately, since their complex nature and sometimes difficult understanding can limit the capabilities of the programmer to build extensive and efficient models. Not only this, but the training of the models can be very time consuming, since the training sets are pretty large in order to make the programs as precise as possible. Lastly, it is possible for the datasets to contain errors, since they are sometimes man-made, which can lead to algorithms learning incorrectly.

2.2 UNSUPERVISED LEARNING

Unsupervised Learning (UL) is a strategy that implies the use of specific algorithms that are capable of interpret unlabeled datasets in order to find common characteristics or patterns between the data, so the creation of clusters of information is possible (IBM Cloud Education, 2020b). Thanks to this, the program is able to group information into different sets without human intervention. This approach is ideal to some types of tasks, like clustering, which is the process of grouping unlabeled data according to a given characteristic (in which data can be similar or differ), structure or pattern found in the information; association, that allows to find and establish relationships between variables in a dataset using for that a rule-based technique, being example of this the market basket analysis that allows companies to observe connections between produts; and dimensionality reduction, which is a method applied when the dataset as a high level of features, or dimensions, and makes possible to reduce the number of data inputs, in order to optimize the performance of the algorithm and facilitate the interpretation of datasets without risking their integrity.

Although all of the pros regarding the use of UL, there are some barriers that occur when models are allowed to run without human intervention. For example, thanks to the high volumes of the training sets that are used, two distinct phenomena occur, being the first one a stronger necessity to use more computationally complex models, since the datasets are harder to interpret, and the second one long training times, even longer than those observed with other methods. There is also a higher probability of obtaining inaccurate and unwanted results, since there are no comparable outputs in the training data and, despite the search of this results being completly autonomous, there is still a necessity for an human analysis at the end, in order to validate ouputs. Another problem that sometimes happens is the lack of transparency regarding the reason why data is clustered the way it is, with the programs being unable to explain the reasons behind the distribution that is made.

2.3 REINFORCEMENT LEARNING

Reinforcement Learning (RL) approaches other side of ML, where one tries to create an autonomous agent that is able to observe and analyze the environment in which it is inserted in order to make the optimal decisions that allow it to achieve its goals (Mitchell, 1997). The main focus of the agent is to achieve the maximum reward possible on a sequence of actions, being this reward a sum of the numerical values that are given by a reward function, as an evaluation of each action performed (the reward function can, for example, assign a positive value

for a good action taken by the agent and a negative value for a bad action). With this, it has to be capable of analyzing the results of performing different sequences of actions, in order to learn a control policy that enables it to maximize the final reward, independently of the initial state. This strategy can be applied in many different areas, such as robot learning, *bots* (software programs that are able to perform a series of automated tasks) and sequential schedulling problems.

Learning a control policy can be similar to a function approximation problem, but they are not the same, since there are many differences that distinguish them. For starters, the final reward value we want to maximize is not previously known, because we will only determine it after performing all of the required actions (and even the reward value of each action is only known after performing it); this opposes to what happens in SL, where we already know the ouput for every input when teaching the program with a training set. Next, instead of having fixed training sets, RL agents influence the distribution of them, thanks to the different actions they can take (which arises to the learner the question of what experimentation strategy is the best in a determined case: to keep on exploiting the known states and actions or to explore new ones). Besides this, it is possible for the agent to not have full knowledge on the actual state of the environment at a given time, which limitates the decision making capability and may trigger the necessity of using previous observations too, in order to find a pattern. Finally, in many RL applications, the agent is obliged to learn a variety of different tasks related to the same environment in which it is inserted, enabling the possiblity of using the information gattered about another task so it is possible to facilitate the learning process of a new task.

2.4 EXPLAINABLE ARTIFICIAL INTELLIGENCE

Thanks to the constant increase of accumulated data and the consequent necessity of programs capable of interpreting them, ML techniques have been in the vanguard of application development, with some even arguing that the new programs that are developed using these concepts are now better then humans when it comes to find relationships with in data, exposing man-made theories as oversimplifications of reality (Spinney, 2022). Individuals even hypothesised that it could be the end of the scientific method, with theories being put into a corner and people embracing the simple correlations that were given to them.

There is some truth associated to this, since the huge amount of information and complexity that are currently provided to us make it impossible for one to be able of writting theories usefull for describing the presented problems, since experts do not even know how this theories could be formalized. Despite all this, theory refuses to die and there may be some reasons that lead to this. First of all, there is still the possibility of existing many "traditional theories" that are yet to be exposed. Second, AI is incredible yes, but it is fallible. Last but not least, humans themselves do not feel comfortable with something when they do not know how it works. And this is where Explainable Artificial Intelligence (XAI) enters the scene.

XAI intends to create systems capable of providing more insight to humans on how they effectively work, giving for that more explanations not only on their capabilities and understandings, but also on what happened until a certain point, what will happen at that moment and what will happen in the future; they should also allow observers to know the important information that they use in order to obtain the results (Gunning et al., 2019).

Despite all this, it is relevant to refer that all explanations are context dependent, since the intended result may vary regarding who asks for them and for what they are using them. For starters, models can be fully interpretable (if the given explanations are transparent and provide a complete view of the system) or partially interpretable (if the given explanations only provide information on important pieces of their reasoning process). The determination of what is the best technique to use and what kind of information will be given always depends on the final use of the given explanation, and this is many times related to the type of user that seeks that information. Some users may search for explanations regarding the reasons why given results are the ones observed, while others may want to know how the results were obtained; there is even the possibility of users wanting answers for both of this aspects. Being the ultimate goal of XAI to provide the intended explanations to humans, it should always take into consideration the target user group, in order to better suit the given answers to their background and necessities.

When all this is taken into consideration, there are challenges and issues that arise, some of which we will present up next. First, it is hard to decide if programs should take into account the knowledge that users have or do not have, as well the characteristics of the users that may obtain the explanation; also, can we get to a point where humans provide feedback of the given explanations, in order to generate a interactive system? It can also be difficult to find a "sweet spot" between accuracy (all the precise information that allowed the program to generate the result) and interpretability (what users can effectively understand), so one can obtain a understandable yet useful explanation. Besides this, there is always the challenge of finding abstractions that allow some easier explaining of the phenomena that occur, like high-level patterns that allow a simpler explanation of big plans in big steps. As an last example, we can refer the necessity of distinguishing the decisions that were done from the competencies that the system has, so that end users can understand if some result was not the intended one because of the path that was took to find the solution or if the program is not capable of solving the problem.

2.5 MACHINE LEARNING AND FUNCTIONAL PROGRAMMING

As some researchers have already shown and hypothesized, ML and FP can effectively become two closer worlds then one initially may think of. In this section, we will expose some of the previous work done in order to connect both areas.

2.5.1 *Christopher Olah's View*

THE STATE OF DEEP LEARNING AND PREDICTIONS ABOUT THE FUTURE Olah (2015) defends that DL is still in it's "ad-hoc state", which means that because it is a relatively young field, it is still built in a very necessity based way, originating solutions that work and can solve problems without being completely understood by humans. Also, he argues that all of the DL concept is stick together with the help of a tool that enables one being capable of creating solutions, despite having no way to create a general agreement on how

DL should be seen and in what foundations should it be supported. With all of this, he theorises that DL will certainly be very different in the future.

Starting from this point, he leads us to speculate in how will one see DL in the future, despite knowing it is impossible to predict what is going to happen.

As we previously stated, there are three narratives used in order to better understand how someone can see DL, and despiting not being mutually exclusive, they certainly represent different views: the neuroscience narrative, the probabilistic narrative and the representations narrative. Christopher leans over the last one in this case, so one new answer can be formulated:

Deep learning studies a connection between optimization and functional programming.

The idea is to formulize a theory that establish the paralelism between the representations narrative in DL and the type theory in FP, seeing DL as the junction of two enormously diverse areas.

The parallels between Deep Learning/Function Composition and Representations/Types

Anyone can effectively affirm that one of the characteristics of DL is embracing the study of deep neural networks, which are multilayered NN that transform data as it crosses through each layer in order to solve the given task. Despite the details of the layers being constantly evolving, the notion of "sequence of layers" stays the same, with each layer being a function that transforms the output of a previous layer. One can undoubtedly see this as an chain of composed functions that must be optimized so it can solve the problem that was presented, and it is on this statement that Olah believes that the heart of what we are studying relies.

Every layer of the NN transforms data so it can better fit the purpose of the task that is being performed. These transformed versions of data can be seen as "representations", that correspond to types in computer science, which are ways of embedding information in an arbitrary amount of bits. We can establish the parallel between types and representations if we look into the last ones as a way of embedding information in an arbitrary number of dimensions. The similarities continue if we take into account that the same way two functions can only be composed if their types agree, two layers can only be composed when their representations agree.

In basic NN architechtures with linear sequences of layers this becomes easy since there is only the necessity of making the ouput of a layer match the input of the next one, but the same can not be said when more complex architechtures are built. When there are multiple inputs layers converging into one representation and multiple outputs diverging from the same representation, things get trickier. Despite this, it is plausible to think that one can formulate things in such a way that becomes possible for all of the pieces to stick together in a harmonious way.

Considering that representations and types are the building blocks for DL and FP respectively, that one of the narratives of DL (representations narrative) is based on the fact that NN transform data into new representations, and that there are many known relations between maths and FP, the author argues that the connections between representations and types are maybe very relevant.

Deep Learning and Functional Programing

When it comes to programming, function abstraction is an amazing technique that allows for one to reduce not only the amount of code written (since a piece of code is written once and used whenever it is needed, instead of repeating it) but also the probability of introducing bugs in programs, while making it easier to detect errors. Once again, we can start to establish a parallel between this and another DL technique, weight tying, which enables the use of many copies of the same neuron in a NN, making so that the models learn more quickly, since there is less to learn. The reuse of a neuron must be done with criteria tho and one must understand where it could be placed, taking advantage of the structure in data so it will not harm the correct functioning of the model.

There are a bunch of patterns based on this, such as recurrent and convolutional layers, that are widely used in NN and work as higher order functions, which are functions that take other functions as arguments and are immensely studied in functional programming. Many of these patterns work similarly to well known functions, being the main difference the fact that instead of receiving normal functions as arguments, they receive chunks of a neural network. Some examples are:

- **Encoding Recurrent Neural Networks** - work as *folds* and are often used in order to enable the NN to receive lists with variable lengths, like strings.

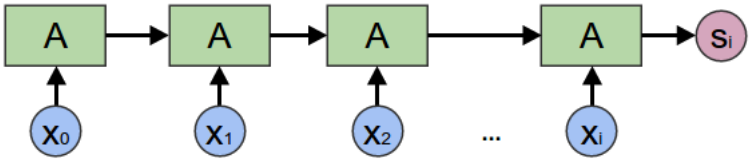


Figure 1: Representation of an Encoding Recurrent Neural Network

- **Generating Recurrent Neural Networks** - resemble *unfolds* and their intent is to allow the NN to create a list of outputs, such as words in a sentence.

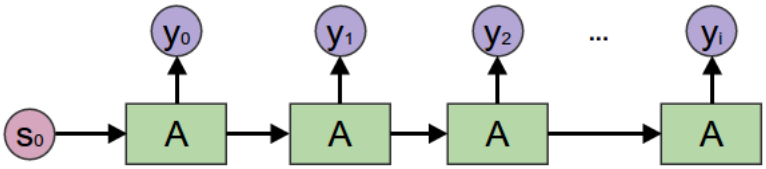


Figure 2: Representation of a Generating Recurrent Neural Network

- **General Recurrent Neural Networks** - are *accumulating maps* and they are often used so systems can try to make predictions in a sequence. An example can be the prediction of a phenome in every time step in an audio segment, based on past context, when developing a voice recognition program.

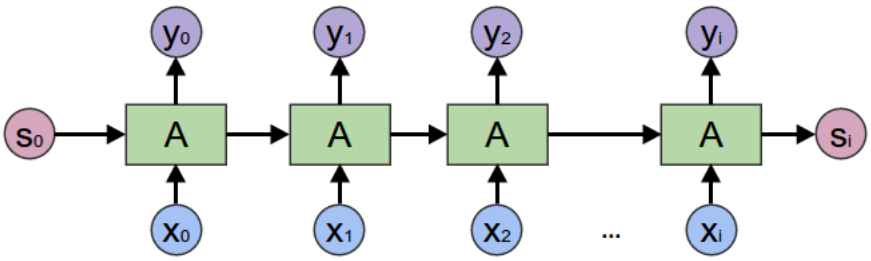


Figure 3: Representation of a General Recurrent Neural Network

- **Bidirectional Recursive Neural Networks** - can be seen as a *left and right accumulating map zipped* together and it is used so predictions in a sequence are possible using both past and future context.

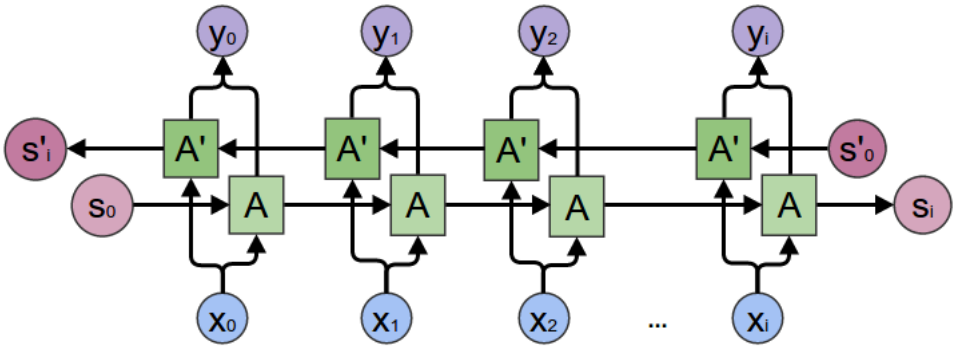


Figure 4: Representation of a Bidirectional Recursive Neural Network

- **Convolutional Neural Networks** - similar to a *map* but not the same, since maps apply a function to every element while convolutional NN apply it also at neighboring elements.

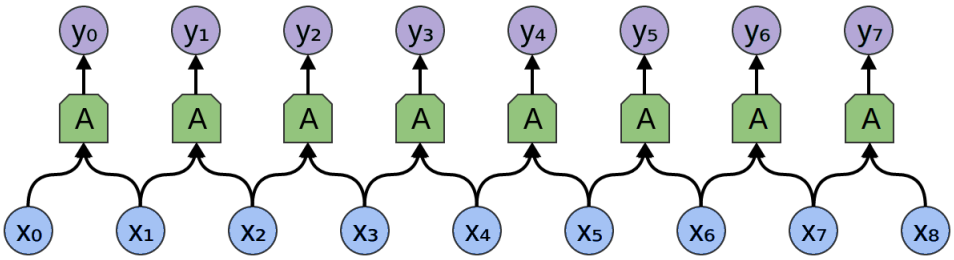


Figure 5: Representation of a Convolutional Neural Network

- **Recursive Neural Networks** - also known as "TreeNets", they are a generalization of folds - *catamorphisms* - that consume a data structure in a bottom-up pattern and are mainly used in natural language processing, so NN can operate on parse trees.

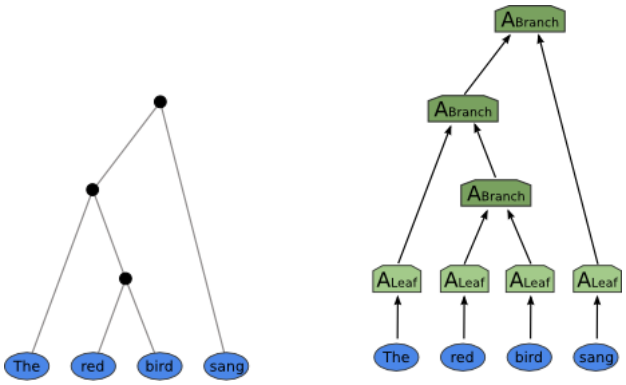


Figure 6: Representation of a Recursive Neural Network

One can effectively see this large network of different patterns combined together as large functional programs with chunks of neural network throughout. Being a functional program, it provides a high level structure, while still allowing for the chunks to be flexible pieces that actually learn to do the requested task inside the framework provided by the program. This is the intersection that Olah idealizes in order to create a somehow new kind of programming, capable of constructing programs that are able of doing things no one can create directly.

2.5.2 *Bogdan Penkovsky's View*

Learning Neural Networks The Hard Way

After a brief introduction on how Penkovsky (2020) was presented to ML, more specifically NN, and it's very basics, he starts by referring some concepts about Haskell and FP and why he believes these are two tools one can use in order to further develop NN. First of all, he believes that Haskell is a particularly good language for NN since these are very "function objets": a network can be seen as a large composition of functions, which is a very common concept in any functional language. He also enumerates some advantages of Haskell as a language, like the ease on reasoning about what a program is doing or how one can refactor it's base code, the ability to improve someone's capability on thinking towards a solution for a problem on a more pragmatic way and the fact that Haskell based programs are fast. One can easily understand why all of these features are very appealing for anyone who is developing NN, since the systems made in the subject are very hard to interpret (in what regards on comprehending what is happening), need to be altered many times before obtaining the final version and must run as fast as possible.

The author then explains what is gradient descent but more importantly it's importance to a NN, making one understand that is by using this method that the error on a network, given by the error function, can be systematically decreased until the minimum error value is achieved. He then continues by giving a demonstration on how this can be implemented in Haskell, and how the written code works.

With this presented, he leans over a concrete example of a network, in this case a program to classify a flower into it's specific classes, using for that some distinctive characteristics that distinguish each other. For starters, this example is useful for him to explain why natural numbers must not be used when referring to a class, but

vectors with a number of dimensions equal to the number of existing classes, so that the distances between all of them are the same. After this, and considering what was previously spoken about gradient descent, he puts the pieces together in order to create the intended network, explaining step-by-step how it must be done, demonstrating not only the calculus that has to be made, but also how to convert it to a Haskell program and how to interpret the given results.

What Do Hidden Layers Do?

In the second part of his blog, Penkovsky begins with a brief explanation on why multi-layer NN can be so useful, sheding some light in their ability to capacitate a system with the possibility to learn non-linear relations between inputs and outputs, providing a way to dodge the linear classifier trap.

With this in mind, he creates an example of a program that receives patterns made with dots and tries to associate a class to any new dot inserted based on it's coordinates. This example is continuously used in order to better illustrate the observed phenomena, since only multi-layer architectures are capable of solving this tasks (single-layer architectures are only capable of drawing straight lines in the input space).

Using this example as a basis, there are taken into account many important aspects of the system's modelying, namely the necessity of a random initialization of the NN weights in order to create neurons capable of learning distinct features; the need of using not only an adequate NN architecture for the task that we are solving, but also an effective training method so one can prevent the program to get stuck with a suboptimal setting; being careful with all activation functions, ensuring all remain nonlinear in order to avoid that a multi-layer NN "collapses" into a single-layer one; and that the ideal number of neurons and layers in a NN may depend on the problem that is being solved (in many cases, more neurons on a single layer don't represent a faster capacity to learn thanks to the fact that the composition of simple transformations made by a fewer number of neurons into more complex ones - what happens in multi-layer architectures - may benefit the resolution of non-linear tasks, since an architecture with more layers can better represent more complex relationships). A curious fact is that, in order to better justify this last explanation, Penkovsky cites another post from Olah's blog (Olah, 2014), regarding the topological transformations that a NN does in the input space.

The writer then proceeds demonstrating how all this can be applied in Haskell, starting by recalling backpropagation, used on the previous demonstration, and how recursion can be used in order to make it more intuitive, while also constructing the basic data structures and functions that work as the founding stones of the program. He also creates the bridge between the *for* loops used in imperative programming languages, that are very useful for NN development, and Haskell's higher order functions, like *map* and *zip*. With all this, one is capable of starting the construction of the model itself, implementing the gradient descent function and the backprop algorithm. Then both the training and the model validation datasets are built, the architecture is defined and the weights are properly initialized. He also explains how an alternative to the gradient descent can be implemented, in order to avoid getting stuck in suboptimal settings in more complex tasks. In the end, a series of tests are done, making evident the benefits of using the techniques referred previously.

Haskell Guide to Neural Networks

In order to continue to make one better understand the doors that Haskell opens on matters of DL studies, Penkovsky writes a new post on the importance of automatic differentiation and how this technique can be applied using this programming language.

He begins by explaining how random search works, giving for that an example that later also helps one to comprehend why it is not the most effective way to maximize the sum of two values, for example, since the inputs are changed in random directions, something that sometimes neither allows to optimize the previous value nor takes into account all of the work done until that point. It is here that automatic differentiation makes a step in, being capable of changing the input in such a way that the output is always improved. This is possible because the algorithm defines specific gradients only for elementary operators, combining them according to the chain rule so that the necessary gradients will be infered by the strategy itself. All of this can be implemented in Haskell, as the author shows, and be adapted so it fits the necessities made up by the NN. In addition to this, one is also presented to a Haskell library that facilitates the construction of differentiable programs, composed by many tools that prove to be useful when programming based on this principles.

The Importance of Batch Normalization

As Penkovsky states, there is one main challenge relative to NN that can be divided in two parts. First, one must know how to train the millions of parameters that compose the net and second, there is a need to know how to interpret them. In the recent years, a new technique as appeared that makes training more efficient, reducing in a very significant way the number of training epochs and making possible the training of certain architectures. This method is batch normalization and consists on the division of a huge dataset into smaller mini-batches that will be processed one at a time during a forward or a backward pass. The precise way this is done varies depending on the phase of the learning process that is running at the moment, existing differences between the application of the procedure at the training and the inference phases.

But how efficient can this method really be? When developing a program that is capable of recognizing human-written digits, tests show that a NN with batch normalization can reach higher accuracy values with way less epochs that a NN without batchnorm. However, and despite all of the good results that are presented, this tecnique presents some pitfalls that can sometimes hinder the program's construction. The fact that the methods used during training phase are different from those used in the inference phase makes the implementation more complicated, while there is also a need to guarantee that all the training data has the same origin, which means one must ensure that every batch represent the whole dataset, having data that comes from the same distribution as the ML task being performed. Also, the truth is that there is still not an explanation with which everyone agrees on why batchnorm is effective (similar to what happens in many other matters related to ML). Initially, it was hypothesized that it reduced the internal covariate shift, while recent studies show that may not be necessarily the case and that the true explanation may reside in the fact that it makes landscape smoother, providing a more efficient way to train the gradient descent and allowing to use higher learning rates.

The author then procedes to show how to modify the previously written Haskell code, so the program becomes capable of supporting batch normalization, explaning how to alter the data structures and the functions that were already made and how to initialize all the needed parameters.

Convolutional Neural Networks Tutorial

In this post, the writter talks about Convolutional NN, labeling them as the "master algorithm" in computer vision and explaining how they work and can be applied. This type of architecture also has a neuron as its building block and is differentiable, allowing a proper backpropagation training, but it's distinctive feature resides in the connection topology that results in sparsely connected convolutional layers, where neurons share their weights, defined by a kernel, resulting in a reduced number of trainable parameters when compared to fully-connected layers. They were specially design in order to address two specifical issues, translation simmetry and image locality, and their capacity to solve them is associated to the convolutional application of an image kernel.

There are many different convolution types, namely Computer Vision-Style Convolution, dedicated to low-level computer vision that operate on one, three or four channels, in which a individual kernel is applied to each channel; LeNet-like Convolution, where a single convolution operates simultaneously on all input channels, producing a single channel, while also allowing for any number of output channels to be obtained depending on the number of kernels; and Depthwise Separable Convolution, where there are not only individual kernels applied to each individual channel but also another convolution in-between channels after that, with an optional activation before that.

Following up on all this, the author builds up a Haskell example based on a LeNet-5, but first gives an explanation on how one can obtain the convolutional layer gradients algebraically. After that, he assembles all the needed pieces so the NN can be made, constructing the data structures and the base functions. The use of stencils and how to implement them is also covered, since these are very important tools in the creation of a convolutional NN. In the end, tests show that an error twice as low as the one obtained with a simple fully-connected can be achieved, while also requiring a drastically lower number of learnable parameters.

Saving Energy with Binarized Neural Networks

Lastely, Penkovsky briefly explains what are binarized NN, which are networks that differ from the "normal" ones since their weights and activations are limited to a single bit precision, enabling that those only have the values of either +1 or -1. This allows to drastically reduce the hardware requirements needed to operate those systems, since most of the needed operations can be replaced by simpler versions of themselves that work bitwise; a speed up on conventional hardware can be obtained; there is a reduction of the area of dedicated ASICs, which means a smaller manufacturing cost; and a there is a reduced energy consumption. In this specific type of NN, the usage of batch normalization is mandatory, so that neurons do not enter a useless state.

2.6 SUMMARY

One can effectively claim that, in what regards ML, distinct strategies can be better suited for distinct problems. SL algorithms require the existence of labeled data, which they use in order to determine final outcomes or to distribute inputs throughout their possible categories. Despite tending to be more accurate, they require human intervention in the creation of training sets with correctly labeled information. However, labelled datasets enable the programs to avoid computational complexity, since they will not need training sets as big in order to produce the intended outcomes (IBM Cloud Education, 2020a). On the other hand, UL algorithms only require unlabeled data, from which they found patterns that enable the creation of groups and allow solving association or clustering problems. This can be useful in matters in which experts are not sure about the characteristics associated to a data set. If we want to extend even further our research, we can also start to talk about semi-supervised learning, which is an aproach that involves giving only a partially labeled training set (IBM Cloud Education, 2020b), mixing this two previous ideas together! By adding RL algorithms to the mixture, even more different problems can be covered, since this strategy allows the creation of programs that are able to interpret information relative to its surroundings in order to perform a series of actions that enable it to achieve its goal. With this, one can make autonomous agents, that can learn from previous decisions and actions that it took, in order to continually evolve and make better decisions (Mitchell, 1997).

Despite the existence of many different types and applications of ML, most of them still lack on one very important thing, explanations, and this is where XAI kicks in. With it, one can hope of beggining to build programs that can be understandable by humans, surpassing the "black-box phase" in which many of them are, in order to shed some light in all of the magic that happens behind the curtains. This technology can potentially be limitless and mark the start of a new era, where XAI systems can interact with humans in many intricate ways, in order to sky-rocket our society to the next level (Gunning et al., 2019).

We start to see some studies and predictions about how functional programming and formal methods can help in the development of ML techniques, like the ones presented in the blogs of Olah (2015) and Penkovsky (2020), but despite that we still lack on proves that all of these areas can be effectively combined. Olah theorises on how one is capable of creating NN using as building blocks the similarities between DL and FP, giving for that examples on how these two areas converge, while Penkovsky shows that it is possible to create many different NN using functional programing, namely Haskell, but there is still a long way to go until someone can assert with certainty that these areas can cooperate and the intent of this project is to help carving that path.

3

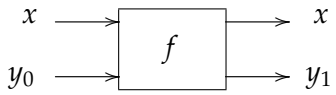
PROSPECT OF FUTURE RESEARCH

The main goal of this dissertation is to prove that patterns like the *accumulating* map and others presented by Christopher Olah (2015) can be effectively used to structure NN design, in a compositional and therefore more understandable way. Compositionality enables one to understand the whole in terms of its parts, meaning that large NN systems could be divided into smaller ones.

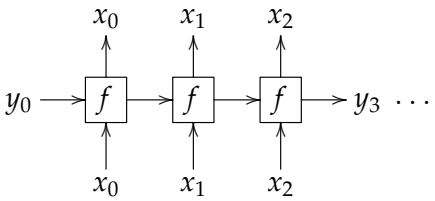
Despite being used with a different aim, Neri et al. (2021) show how the *accumulating map* pattern, which in Haskell has the following signature (for the *right* variant)

$$mapAccumR :: Traversable\ t \Rightarrow (a \to b \to (a, c)) \to a \to t\ b \to (a, t\ c)$$

can be depicted as a chain of blocks, with each block being a function that alters the two inputs it receives in different ways.



When such blocks are combined as shown below,



one can interpret $[x_0, x_1, x_2, ...]$ as a control sequence, that is passed along to the output, while the input y_0 can be used as an accumulator of transformations, one for each x_i, being y_i the final value.

This matches Olah's RNN pattern, recall Fig. 3. Having this in mind, the idea is to build a number of NN examples in Haskell using a library for typed linear algebra developed in a previous MSc project (Santos and Oliveira, 2020) to describe the f blocks above. Besides $mapAccumR$, other structuring combinators will be considered, as suggested by Olah.

Hopefully, this will provide an answer to the second question of the three put forward in the introduction, bringing some order to the inscrutable complexity that ML can sometimes have. Showing that NNs can be built in a more meticulous way, thanks to the compositionality of functional programming, would be a significant step towards revealing what may be hidden from the eyes of the majority of programmers until now.

BIBLIOGRAPHY

Samson Abramsky. Nothing will come of everything: Software towers and quantum towers, 2022. Department of Computer Science, UCL.

R.S. Bird and O. de Moor. *Algebra of programming*. Prentice Hall International series in computer science. Prentice Hall, 1997. ISBN 978-0-13-507245-5.

David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. XAI - explainable artificial intelligence. *Sci. Robotics*, 4(37), 2019. doi: 10.1126/scirobotics.aay7120. URL https://doi.org/10.1126/scirobotics.aay7120.

IBM Cloud Education. What is supervised learning?, 2020a. Site: https://www.ibm.com/cloud/learn/supervised-learning, last read: February 4, 2022.

IBM Cloud Education. What is unsupervised learning?, 2020b. Site: https://www.ibm.com/cloud/learn/unsupervised-learning, last read: February 4, 2022.

Tom Mitchell. *Machine Learning*. McGraw Hill, 1997. ISBN 0-07-042807-7. OCLC 36417892.

A. Neri, R.S. Barbosa, and J.N. Oliveira. Compiling quantamorphisms for the IBM Q-Experience. *IEEE Trans. Soft. Eng.*, 2021. DOI: 10.1109/TSE.2021.3117515 (In the press).

C. Olah. Neural networks, manifolds, and topology, 2014. Blog: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/, last read: February 4, 2022.

C. Olah. Neural networks, types, and functional programming, 2015. Blog: http://colah.github.io/posts/2015-09-NN-Types-FP/, last read: February 4, 2022.

J.N. Oliveira. Transforming Data by Calculation. In *GTTSE'07*, volume 5235 of *LNCS*, pages 134–195. Springer-Verlag, 2008.

B. Penkovsky. 10 days of grad: Deep learning from the first principles, 2020. Blog: https://penkovsky.com/neural-networks/, last read: February 4, 2022.

A. Santos and J.N. Oliveira. Type your matrices for great good: a haskell library of typed matrices and applications (functional pearl). In Tom Schrijvers, editor, *Proceedings of the 13th ACM SIGPLAN International Symposium on Haskell, Haskell@ICFP 2020, Virtual Event, USA, August 7, 2020*, pages 54–66. ACM, 2020. doi: 10.1145/3406088.3409019. URL https://doi.org/10.1145/3406088.3409019.

Laura Spinney. Are we witnessing the dawn of post-theory science?, 2022. The Guardian, Sun 9 Jan 2022 09.00 GMT. Last read:February 4, 2022.