DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Predicting Solar Radiation using a Deep Neural Network

SICS - Swedish Institute of Computer Science

ADAM ALPIRE

TRITA TRITA-ICT-EX-2017:105

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Predicting Solar Radiation using a Deep Neural Network

SICS - Swedish Institute of Computer Science

Adam Alpire Rivero

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

8 Jun 2017

Examiner: Jim Dowling

Abstract

Simulating the global climate in fine granularity is essential in climate science research. Current algorithms for computing climate models are based on mathematical models that are computationally expensive. Climate simulation runs can take days or months to execute on High Performance Computing (HPC) platforms. As such, the amount of computational resources determines the level of resolution for the simulations. If simulation time could be reduced without compromising model fidelity, higher resolution simulations would be possible leading to potentially new insights in climate science research. In this project, broadband radiative transfer modeling is examined, as this is an important part in climate simulators that takes around 30% to 50% time of a typical general circulation model. This thesis project presents a convolutional neural network (CNN) to model this most time consuming component. As a result, swift radiation prediction through the trained deep neural network achieves a 7x speedup compared to the calculation time of the original function. The average prediction error (MSE) is around 0.004 with 98.71% of accuracy.

Keywords
Deep Learning; Climate Science Prediction; Regression; Convolutional Neural Network; Solar Radiation; Tensorflow.

Sammanfattning

Högupplösta globala klimatsimuleringar är oumbärliga för klimatforskningen. De algoritmer som i dag används för att beräkna klimatmodeller baserar sig på matematiska modeller som är beräkningsmässigt tunga. Klimatsimuleringar kan ta dagar eller månader att utföra på superdator (HPC). På så vis begränsas detaljnivån av vilka datorresurser som finns tillgängliga. Om simuleringstiden kunde minskas utan att kompromissa på modellens riktighet skulle detaljrikedomen kunna ökas och nya insikter göras möjliga. Detta projekt undersöker Bredband Solstrålning modellering eftersom det är en betydande del av dagens klimatsimulationer och upptar mellan 30-50% av beräkningstiden i en typisk generell cirkulationsmodell (GCM). Denna uppsats presenterar ett neuralt faltningsnätverk som ersätter denna beräkningsintensiva del. Resultatet är en sju gångers uppsnabbning jämfört med den ursprungliga metoden. Genomsnittliga uppskattningsfelet är 0.004 med 98.71 procents noggrannhet.

Nyckelord
Djupinlärning; klimatprediktion; regression; neurala faltningsnätverk; solstrålning; Tensorflow.

Acknowledgements

I would like to express my deepest gratitude to my master's thesis advisors Prof. Jim Dowling and Dr. Ying Liu. I have learned many things since I became Prof. Jim Dowling's student, such as finding my passion in machine learning. I am also grateful to Mazen M. Aly for spending time reading this thesis and providing useful suggestions about it. They are all hard-working people, and I believe their academic achievements will continue to increase.

Special thanks are given to the Swedish Institute of Computer Science where I worked in, in this thesis project, as a part of KTH and SU. I also wanted to thank KTH and UPM, as the two universities I have studied in during these two years, as exit and entry universities (respectively) within the EIT Digital Master's School program.

During the period of two years, many friends have colored my life. I have to acknowledge all my colleagues that studied with me for their assistance in many aspects. They are Adrian Ramirez, Alejandro Vera, Braulio Grana, Ignacio Amaya, Carlos Garcia, Filip Stojanovski, and Philipp Eisen, among many others.

My life friends have been always there, even in the distance. This is why I want to acknowledge them for all the good times that were converted into energy for overcoming the hard times.

Last but not the least important, I owe more than thanks to my family members which includes my parents, siblings, and my life partner, for their support and encouragement throughout my life. Without their support, it would have been impossible for me to become the person I am.

Mañana más y mejor,

Adam Alpire

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

This document requires readers to be familiar with terms and concepts related to: machine learning modeling; deep learning; and some acronyms internally defined in this thesis report.

SR	Solar Radiation
RT	Radiative Transfer
GPU	Nvidia GeForce GTX 1080 8GB DDR5 RAM
CPU	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 16GB DDR3 RAM
DNN	Deep Neural Network
AI	Artificial Intelligence
CNN	Convolutional Neural Net
ConvNet	Convolutional Neural Net
DL	Deep Learning
BN	Batch Normalization
CO_2	Dioxide Carbon
CRISP-DM	Cross Industry Standard Process for Data Mining
ST	Surface Temperature in Celsius
H	Humidity
T	Temperature in Celsius

P	Pressure
IM	Initial Model, represents the previous work to this project done in SR modeling
BM	Basic Model, includes the work done in the first part of the project
FM	Final Model, represents the set of architectures developed in the second part of the project
MSE	Mean Squared Error
Climate Science	Is the field that studies the Earth's climate system.
ANN	Artificial Neural Net
ReLU	Rectified Linear Unit
PReLU	Parametric ReLU
RRTMG	Rapid Radiative Transfer Model for GCM
SMAPE	Symmetric Mean Absolute Percentage Error

Chapter 1

Introduction

Chapter 1 serves as an introduction to the degree project where the background to the problem, and overall aim of the thesis report, are presented. It also discusses delimitations and choice of methodology to accomplish the project goal.

1.1 Problem description

Climate science is facing a number of challenges related to Big Data [2]. Firstly, (1) increasing volumes of data are increasing the turnover time for analysis and hypothesis testing. Secondly, (2) researchers are throwing away valuable data that could provide useful insights because of the perception that data storage volumes are limited by cost and/or availability. Thirdly, (3) simulation times are increasing as the amount of input features and observations increases inexorably.

Historically, the main limitation for climate science has been the computational power [3], requiring the usage of expensive alternatives such as supercomputers. The large-scale project leveraged in these supercomputers are mostly executing long-running simulations over enormous models. Such simulations try to study the climate ranging from seasons to centuries. Alongside the simulation computations, it is also needed to evaluate their outputs of up to terabytes of data, and in many cases, against other simulation outputs.

Deep learning offers the potential to help solve each of these challenges. To date, deep learning has been used in climate science for tackling climate pattern detection problems - such as extreme weather events [4].

1.2 Thesis objective

In this degree project, data-driven approaches to using deep learning for regression in the field of climate science will be investigated. Concretely, this thesis will look in detail at the problem of accurately modeling solar radiation. Currently, modeling solar radiation is an important part of climate simulators, where they expend a majority of their computation cycles estimating solar radiation with complex and costly calculations. Therefore, the approach to be taken instead of the classical and complex mathematical models, is to use deep learning networks, such as convolutional neural networks (Section 2.2.2).

The solar radiation (SR), better explained in Section 2.1.3, takes between 30% and 50% percent of the computation in specific climate simulations (Section 2.4), where each solar radiation computation takes around 0.1 to 0.03 seconds in a regular CPU (detailed in Section 3.1.2). The aim of this thesis work is to reduce the time per solar-radiation calculation through deep learning by ”learning” patterns in the SR calculations functions explained in Chapter 2.

The fact that around 30% to 50% of the computation is spent in SR calculations, implies that any little improvement in the calculation speed will be translated into a huge reduction of computing time and hence, a great reduction in simulation costs. Such improvement would enable more ambitious simulations in terms of temporal time, granularity, and size.

Notwithstanding, an application of a faster model (deep learning) instead of the regular model (classic model) will not have any impact as long as the predictions are of a high fidelity. An exhausting evaluation of the model to ensure that the error compared to the ground-truth radiation is very little, will be required.

In summary, deep neural networks learn high-level representations from data directly. We will investigate the potential of functions used in climate science simulations to be replaced by deep neural networks.

1.3 Methodology

The conclusions and recommendations of this quantitative thesis project will be based on a scientific study using an empirical and experimental approach to achieve scientific validity. The implementation process of the neural net will be an iterative process that would allow continuous improvements. The iterations are intended to be working with easier scenarios (simpler data, less granularity) at the beginning and as the results improve, use more complex data with more spectrum

and more granularity. At the first stage of the project a state-of-the-art evaluation is performed to be able to take advantage of current research in the field of deep learning techniques, and to better justify in this thesis the work being done. The resulting model after each iteration is then assessed using error metrics and data visualization tools.

In parallel with the model training, a library for integrating the model to the simulator needs to be implemented. Special effort in the production performance is also required, and a previous literature study is also needed to understand how the deep learning model deployments are being done and what would be the best approach for this concrete scenario.

A final evaluation of the improvements in speed and the drawbacks in accuracy will be done at the end of the work.

1.4 Delimitations

Deep learning is a technology that is setting the state-of-the-art in innumerable fields, and a huge community is growing every day. Then, it is normal that every week new advances are published and new techniques are described to be the next key element of deep learning. Hence, it is normal that the hardware industry is trying to accommodate to the market, and produce new technology accordingly.

For the academic field, the stated before is a limitation as the very latest techniques that are achieving the state-of-the-art performance, require recent technology or really powerful equipment that are not within everyone's reach. On the one hand, despite having a powerful GPU, for some techniques like residual nets [5], a single GPU is far from being enough; therefore, in some occasions some design decisions may be inclined to a simpler solution than to the best solution. On the other hand, deep learning frameworks are aware of the limitations in vertical scaling, and are working on simplifying the parallelization of the training process in many distributed environments. However, the lack of documentation and stability of this libraries at the moment of defining the thesis, and the uncertainty if horizontal scaling was really needed for solving this problem, left this alternative for further iterations of the big project. Therefore, using parallel deep learning training was out of the scope for this master thesis project.

Finally, the innovative nature of the project, deep learning application in climate science, is a new approach that has not been done yet, and hence the risk of not succeeding in achieving the best within this master thesis project is possible. Nonetheless, this master thesis project is in the boundaries of a bigger project that will keep working after the master thesis project finishes.

1.5 Structure of this thesis

Chapter 1 describes the problem and its context. Chapter 2 provides the background necessary to understand the problem and the specific knowledge that the reader needs to have in order to understand the rest of this thesis. Following, Chapter 3 the implementation of the DNN and the deployment process. Then, the solution is analyzed and evaluated in Chapter 4. Finally, Chapter 5 offers some conclusions and suggests future work.

Chapter 2

Background

Chapter 2 intends to lay the foundation of the theory that is essential in order to understand the problem that this degree project aims to explore. This includes basic theory about the climate science part, the deep learning part, and how other work has used machine learning in climate science.

2.1 Climate Science

2.1.1 Climate Science vs Weather Forecasting

The first important thing to understand before starting to read this report is: **what is climate science?**.

It is very frequent to see a huge misunderstanding and confusion regarding this question. In summary, climate science is not weather forecasting.

Weather forecasting is related to short-term predictions that need accurate knowledge of the current state of the weather system. Weather varies tremendously from day to day, week to week, season to season. Climate, on the other hand, regards to really long-term predictions to many years or even centuries that averages the weather over periods of time. A climate prediction/forecast is the result of an attempt to reproduce an estimate of the actual evolution of the climate in the future [6]. Therefore, a change of 7 °C from one day to the next is barely noting when discussing weather. Seven degrees, however, make a dramatic difference when talking about climate.

A great analogy to understand the nature of both, weather forecast and climate

forecast, is the balloon analogy [7] explained in the blog of a researcher in climate informatics. It is worth explaining this analogy in this report.

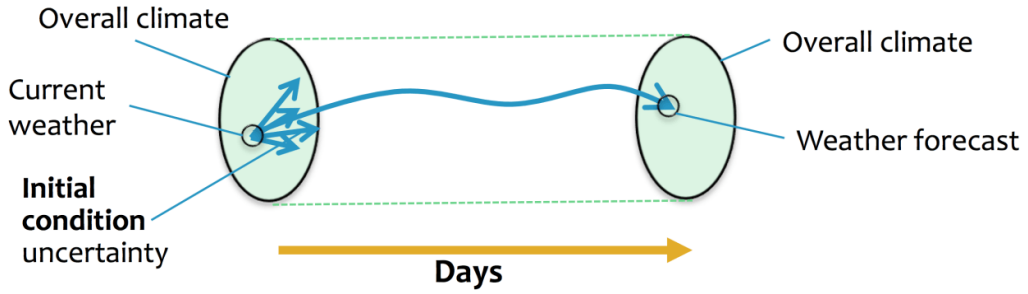


Figure 2.1: Weather forecasting illustration, short-term prediction.

It is possible to determine where a water ballon will go after throwing it if the physics are understood and precise measurement about the angle and the force of throw are available. The more errors in the measurements and the longer the throw, the less accurate will be the falling prediction. This is how weather works and this is why the weather models tend to not be accurate within more than a week predictions. Figure 2.1 gives a graphic idea of weather prediction.

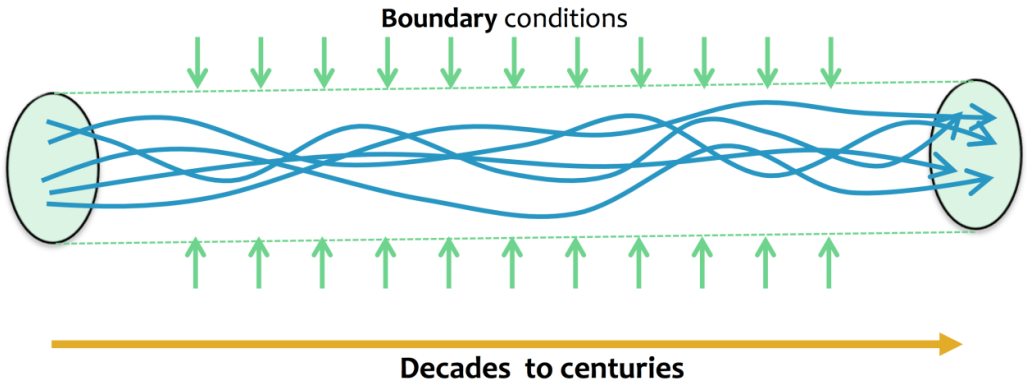


Figure 2.2: Climate science boundaries, long-term prediction.

Now, imagine an air balloon tied with a string to the case of a working fan. The balloon will keep on bobbing within the same boundaries of the fan's air, for hours and hours as long as the fan is behaving the same way. This is a boundary problem, it will not be feasible to predict exactly where the balloon will be at any moment, but it will be possible to tell, fairly precisely, the boundaries of the space in which it will be bobbing. Figure 2.2 illustrates this idea. Then, if someone suddenly changes the direction of the fan, the balloon will move with the air flow; and, if the physics are well modeled, it would be possible to determine the new

boundaries for the balloon's bobbing. Figure 2.3 shows this case. This is how climate prediction works: one cannot predict what the weather will do on any given day far into the future. But if one understands the boundary conditions and how they are altered, he can predict fairly accurately, how the range of possible weather patterns will be affected. Climate change is a change in the boundary conditions on the weather systems.

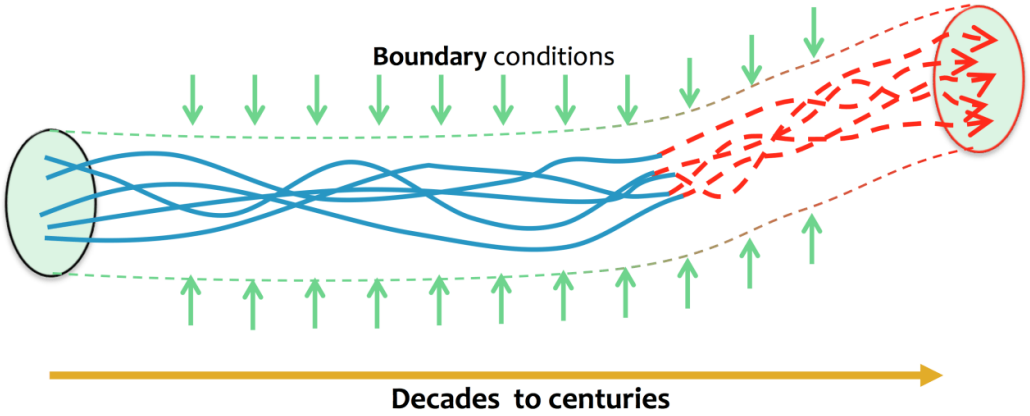


Figure 2.3: Climate science change of boundaries, long-term prediction.

For purposes of the Climate Science Laboratory (CSL), the Earth's climate system is defined as the coupled atmosphere, oceans, land, cryosphere, and associated bio-geochemistry and ecology, studied on time scales ranging from seasons to centuries. Climate science is the field that studies the Earth's climate system as defined before.

2.1.2 Climate Science - intensive computation

Climate science is an active research field that is taking advantage of the improvements in computing speed and memory [8]. The *climate simulators*, model 3D boxes over the surface of the Earth and within each box those models describe physical, chemical and biological processes. The smaller the box, the more precision but the more expensive in terms of computation.

Important institutions such as the National Center for Atmospheric Research (NCAR) or the University Corporation for Atmospheric Research (UCAR), allocate calls [9] for high-performance computing and data storage systems to support extremely demanding, high-profile climate simulations. The motivation of such calls is because climate simulations require: high resolution, span many centuries of simulated time, encompass large numbers of ensembles, integrate

new physics or models, or address national and international scientific priorities.

Therefore, improving such simulations not only allow to have better resolution (smaller boxes) but also to leverage the development of more sophisticated models that can simulate the climate more precisely.

2.1.3 Solar Radiation

Predict solar radiation (SR) is the final objective of this project; thus, even though it is not critical to understand what SR is, understanding it will enrich the learning beyond the pure machine learning modelling. However, understanding the context of SR in order to understand the data is essential to correctly approach the problem and train the NN model.

Solar radiation is energy irradiated by the Sun in form of a wide spectrum of light waves. Light waves, very briefly, are vibrations of electromagnetic fields. The solar radiation that hits the atmosphere is filtered by the organic components of it such us O_3 (ozone) , CO_2 (carbon dioxide), H_2O (water vapour), and N_2O (nitrous oxide). [10] These compounds absorb radiation at different spectrums of the wave and directly influences in the **Heating Rate**. Heating rate is the energy, the gaseous absorption of solar radiation and is essential in studies of radiative balance and global circulation models of the atmosphere [11].

The atmosphere filters the energy received from the Sun and from the Earth. Radiative transfer describes the interaction between radiation and matter (gases, aerosols, cloud droplets). The three key processes to be taken into account are: emission; absorption of an incident radiation by the atmospheric matter (which corresponds to a decrease of the radiative energy in the incident direction); scattering of an incident radiation by the atmospheric matter (which corresponds to a redistribution of the radiative energy in all the directions). All these interactions include a highly degree of complexity; however, Figure 2.4 tries to illustrate all these processes.

What the model developed in this thesis project is trying to predict, is the energy of the solar radiation transferred to the atmosphere, called **Radiative Transfer**. For simplicity of reading, throughout this whole report, radiative transfer is referred as solar radiation.

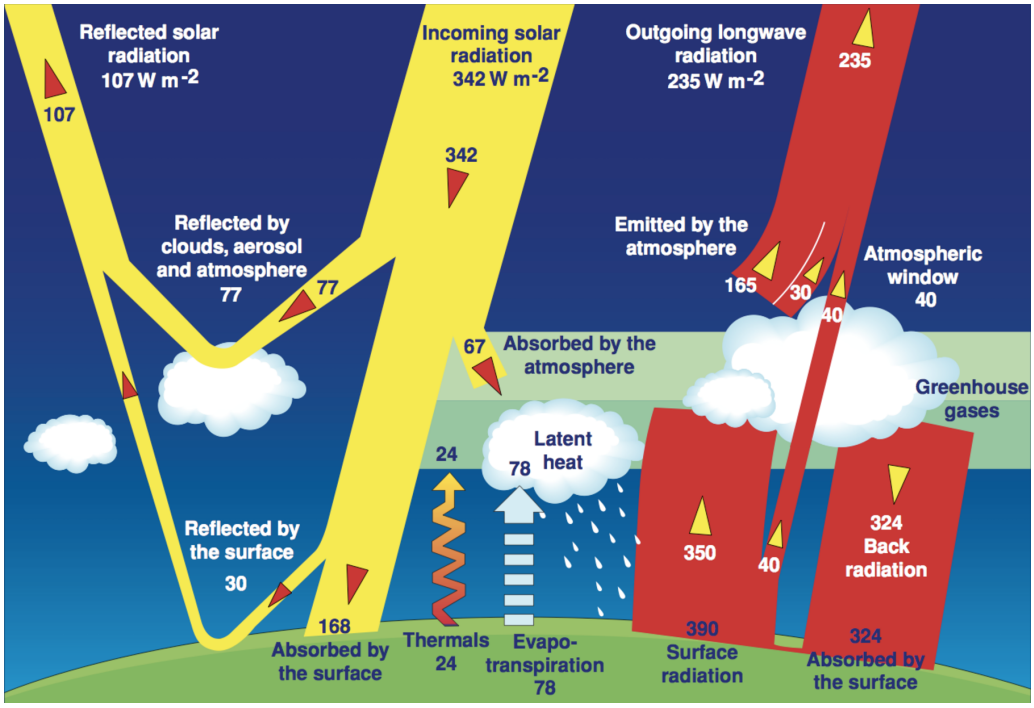


Figure 2.4: Illustration of natural interaction between solar radiation and matter in the atmosphere.

2.1.4 Simulators

The climate science simulators are basically software that responds what-if questions regarding the atmosphere. i.e, *what if we stop burning fossil sources today?*, *what if we burn all of a sudden all fossil sources?*, *what if we keep on burning fossil sources on the same pace for another 50 years?*. Such simulators are based in models that have been developed and improved for years, and include software considered to be about the same level of quality as the Space Shuttle flight software [12].

What makes these models work so good is that the way climate scientists work is not the common software development. Once in a while, climate scientists all around the world gather together and bring the models they have been working on in their research labs, and test them against a big battery of tests. And not only that, they also agree to release all the results of those test publicly, so that anyone who wanted to use any of that software can pore over all the data and find out how well each version did, and decide which version they want to use for their own purposes [13].

On the other hand, climate scientists are aware that modelling the climate

system is not easy, and they are aware that their models are not perfect. They regularly quote the statistician, George Box, who said *"All models are wrong, but some are useful"*. In fact, scientists do not do the model to try to predict the future, but to understand the past.

2.2 Deep Learning

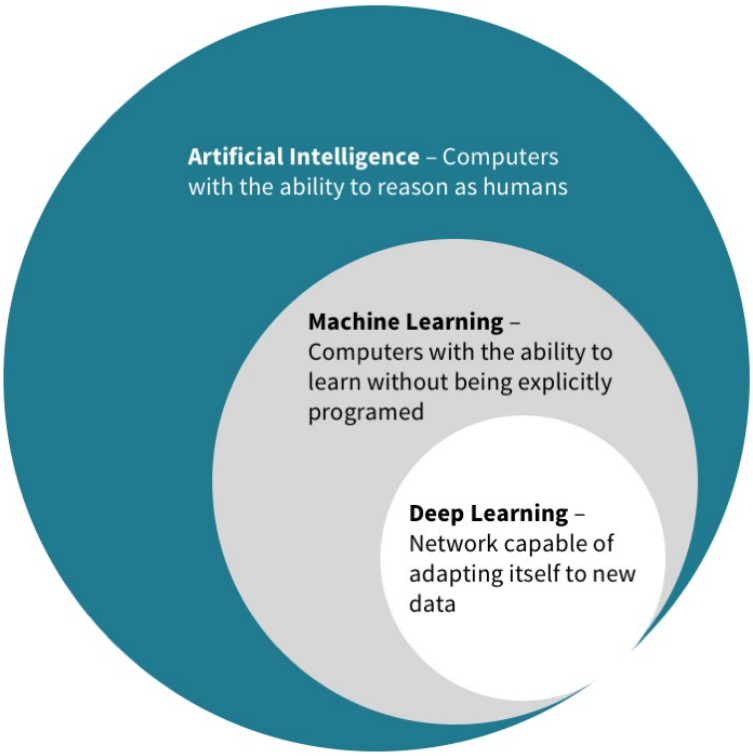


Figure 2.5: Graphic definition of where is deep learning located in the current research of AI.

Today, artificial intelligence (AI) is a thriving field with many practical applications and active research topics. Intelligent software is used to automate routine labor, understand speech or images, make diagnoses in medicine and support basic scientific research. Several artificial intelligence projects have sought to hard-code knowledge about the world in form of techniques such as formal language using logical inference rules. In general, people straggle devising normal rules with enough complexity to accurately describe the world. The difficulties faced by systems relying on hard-coded knowledge suggest that AI

systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This capability is known as *machine learning*.

Deep learning [14] is a machine learning algorithm that can be seen as a graph of operations with trainable weights that introduces non-linear operations that allows the graph to solve non-linear problems. The graph forms a net of operations that have different layers where the data flows, and where each node of the graph is a *neuron*. At each layer a set of nodes operate with portions of the data from previous layers (or also further layers in the case of recurrent networks). The number of layers represent the *depth* of the net and the number of nodes per layer are the *breadth* of the net. The net has two external layers, the input layer and the output layer, and all the intermediate layers are the hidden layers. The input layer receives the data that then suffers several transformations until it reaches the output layer, presenting the final output. Therefore, the deep neural net ”learns” a representation of the problem expressed in terms of other representations. Deep learning allows the computer to build complex concepts out of simpler concepts. Figure 2.5 illustrates the idea of deep learning within its context.

2.2.1 Brief history

Deep learning is an old research field that dates back to the 1940s, however, *deep learning* is a recent name. Broadly speaking, deep learning is in its third boom of development and this is because the actual situation of the technology. Nowadays, the amount of available training data has increased, and computer hardware and software infrastructure has improved. Therefore, deep learning models have grown in size and have become more useful than in older eras.

The first DL models were basic trainable linear models such as the perceptron [15], yet, linear models are very limited and cannot learn simple problems such as the XOR function. This was the first major dip in the popularity of neural networks.

Despite earlier algorithms were intended to be computational models that try to imitate how learning happen in the brain, biological neural networks are only a motivation for deep learning, and this is why they are also known as *Artificial Neural Nets* (ANN or NN). The modern deep learning goes beyond the neuroscientific perspective, on the current breed of machine learning models.

In the 1980s, the second wave of neural network research emerged in great part via a movement called connectionism. The central idea in connectionism is that a large number of simple computational units can achieve intelligent behavior when networked together. This insight applies equally to neurons in biological nervous

systems and to hidden units in computational models. This way, introducing non-linear hidden functions in between, solving non-linear problems became possible. Another major improvement in this time was the development of the back-propagation algorithm to train deep learning models fast and automatically. After these improvements, deep learning raised huge expectations that AI researchers did not fulfill. Simultaneously, other fields of machine learning made advances. These two factors led to a decline in the popularity of neural networks.

In 2006 neural networks were thought to be hard to train, but, a research in this year showed that a good initialization of the weights could considerably improve the training time and thus, allow the training of more complex and deeper nets. After this, deep learning showed to be successfully working in many other fields and the third wave of NN research began with breakthrough in this year. The ability to train workable deep neural nets let to the so-called name of *deep learning* for this algorithm.

A very important factor of the popularity of this algorithm after 2006 until now is the trend of Big Data and the disposition of large datasets that can be processed easier and faster with the improvements in hardware and software.

Nowadays, deep neural networks outperformed competing AI systems based on other machine learning technologies as well as hand-designed functionalities. Academia and industry are currently investing efforts in research and development of neural networks due that the deep learning is successfully becoming the state-of-the-art solution for many problems. Accordingly, using deep learning is being easier everyday as more information and powerful frameworks are being published everyday. Such frameworks, give abstractions at many levels for using deep learning, and for managing the Big Data; making the life of developers much easier.

In order to understand the next sections of this chapter it is recommendable to have some machine learning background and some general understanding about deep learning as it will be discussed concrete techniques used for improving the training process, and to develop a more sophisticated model based in current research of state-of-the-art techniques.

2.2.2 Convolutional Neural Nets

Convolutional neural networks (CNNs or ConvNets) are a type of feed-forward neural nets in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. These networks have been popularized because they scale better than the classical fully-connected networks,

where neurons of further layers are not connected to all the previous layers of the net. In computer vision problems, the need to process images where each image conform a sample, and each pixel represents a feature, scaling the algorithm is crucial in order to process large sets of images.

What makes ConvNets the de-facto DL technique is that the convolution operations that applies to the input allows to encode important properties for images, such as importance of proximity of pixels, into the architecture of these nets. The neurons of the net have a restricted region of space, known as receptive field, that depends on the filter of the convolution. The receptive fields of different neurons partially overlap such that they tile the visual field in further layers that have representations of smaller representations. This representation of more basic representations conceptually allow to identify very basic element for example in photos of faces, and then use this representations for identifying bigger representations like eyes, mouth, nose. And then, finally recognize a human head. Figure 2.6 shows this idea.

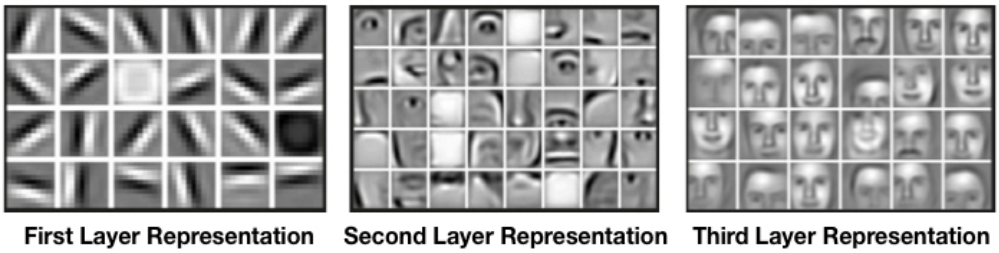


Figure 2.6: Learning representation of ConvNets in different layers.

Many popular configurations of this architecture are realised every year with improvements in the yearly event ILSVRC [16]. But, a very basic configuration for the classic CIFAR-10 [17] dataset would be:

INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.

CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if it was decided to use 12 filters.

RELU layer will apply an element-wise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).

POOL layer will perform a down-sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

FC fully-connected layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Every year, the winner of the ILSVRC competition comes up with a deeper net that requires the latest hardware, and patience for training, with the benefit of improving the results of the last year. The net used in this project, better explained in Chapter 3, is a ConvNet motivated in the ILSVRC2014 winner, VGGnet [18], but takes important elements of earlier and older contestant winners such as AlexNet [19] or GoogleNet [20].

2.2.3 Terms and Techniques of Deep Learning

With the purpose of helping the reader to understand some of the key ideas of the final solution, this section will describe in some detail, some critical concepts in deep learning that will be later on referenced in further Chapters.

2.2.3.1 Hyperparameters

In machine learning, the algorithms have several settings that one can use to control the behavior of the learning algorithm. The values of hyperparameters are not adapted by the learning algorithm itself (though one can design a nested learning procedure where one learning algorithm learns the best hyperparameters for another learning algorithm). Sometimes a setting is chosen to be a hyperparameter that the learning algorithm does not learn because it is difficult to optimize. More frequently, one does not learn the hyperparameter because it is not appropriate to learn that hyperparameter on the training set.

In general, a long part of the project's time is spent in tuning hyperparameters and this is what has happened in this project too. Many techniques have been developed for finding the best hyperparameters such as grid search or random search. However, when the dimensions of the problem is too big and trying new hyperparameters is costly, a more precise understanding of the problem and more experience is crucial. In Chapter 3 the hyperparameter tuning of the project is detailed.

2.2.3.2 Batch Normalization

Batch Normalization (BN) [21] is a recent technique that has been popularized a lot in the last year, and nowadays is absolutely recommendable, almost mandatory, for all neural nets.

Why does this work? Well, it is known that normalization (shifting inputs to zero-mean and unit variance) is often used as a pre-processing step to make the data comparable across features. As the data flows through a deep network, the weights and parameters adjust those values, sometimes making the data too big or too small again - a problem the authors of BN refer to as "internal covariate shift". By normalizing the data in each mini-batch, this problem is largely avoided. Hence, rather than just performing normalization once in the beginning, one is doing it all over place after each trainable layer.

The problem is the weights, even though they are initialized empirically, they tend to change a lot compared to the trained weights, and also the DNN by itself is ill-posed, i.e. a small perturbation in the initial layers, leads to a large change in the later layers. During back propagation, these phenomena causes distraction to gradients, meaning the gradients have to compensate the outliers, before learning the weights to produce required outputs. This leads to the requirement of extra epochs to converge.

Therefore, batch normalization potentially helps in two ways: faster learning and higher overall accuracy. The improved method also allows to use a higher learning rate, potentially providing another boost in speed.

This technique then, is implemented as a hidden layer and the original paper recommends its insertion after the activation function; however, it is still a discussion area in the research field.

2.2.3.3 Parametric ReLU

Parametric ReLUs or PReLUs [22] are activation functions based in the ReLU activation function [23], where the slopes of negative part are learned from data rather than establishing predefined constants. The best advantage of using a non-linear function like ReLU for introducing the non-linearity to the NN is that is it very easy to calculate. Nonetheless, the main disadvantage is that NN tend to suffer from dead neurons whom weight is zero and is not likely to recover from that state because the function gradient at 0 is also 0, so gradient descent learning will not alter the weights. "Leaky" ReLUs with a small positive gradient for negative inputs ($y = 0.01x$ when $x < 0$ say) are one attempt to address this issue

and give a chance to recover. In fact, Leaky ReLUs are one a case of a PReLU where the learned factor of the negative part is very close to 0. Figure 2.7 shows the curve of PReLU.

Overall, PReLUs improve the accuracy of the models compared to ReLUs and Leaky ReLUs [24] in both, train and test datasets. However, the improvement is relatively bigger on training set, it indicates that PReLU may suffer from severe overfitting issue in small scale dataset. This is not a problem at all in big data problems with large datasets as it also offers an improvement over the test dataset against other activation functions.

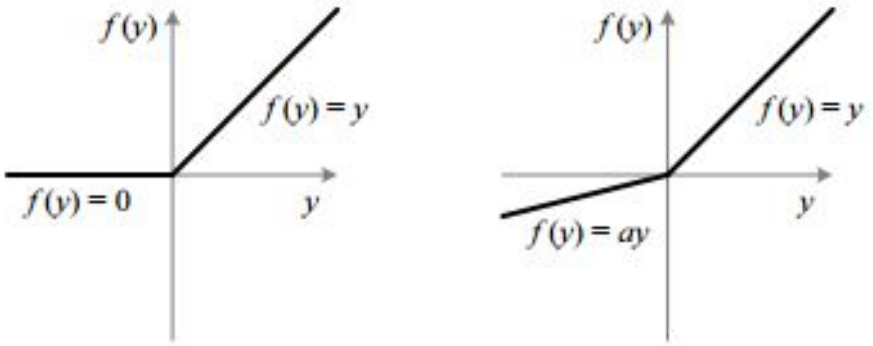


Figure 2.7: ReLU vs PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

2.2.3.4 Xavier Initialization

Xavier initialization [25] makes sure the weights are ”just right”, keeping the signal in a reasonable range of values through many layers. Hence, this initializer is basically designed to keep the scale of the gradients roughly the same in all layers. The main purpose of this initialization is helping signals to reach deep into the network avoiding vanishing and exploding of gradients. Other great benefit is that with this initialization the weights are within a reasonable range before starting to train the network, this speeds up the training in the initial steps as the error is not so big as with random initialization.

In summary, what Xavier initialization does is:

- If the weights in a network start too small, then the signal shrinks as it passes through each layer until it is too tiny to be useful.

- If the weights in a network start too large, then the signal grows as it passes through each layer until it is too massive to be useful.

2.2.3.5 Capacity

In general, the capacity of the net refers to the ability of the network to learn a function to explain the data. One can control whether a model is more likely to overfit or underfit by altering its capacity. Informally, a model's capacity is its ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.

2.2.3.6 Cost function

Cost function is the function that one is trying to minimize when training a model. In the case of this project, the purpose is to predict solar radiation, so in fact, the problem is to do a regression. In the case of regressions a very typical cost function is the mean squared error (MSE). However, MSE is not a good cost function when outliers are present. The outliers may extremely increase the gradient and end up in a gradient explosion making the net to lose the learning all of a sudden.

$$MSE(y_i, h(x_i)) = (h(x_i) - y_i)^2$$

Another popular cost function is the mean absolute error (MAE). While this function is robust against outliers, is not a convex function and is not differentiable, something which is a real problem when applying a learning algorithm as the transition of gradient in the non-differentiable point is not smooth, and the gradient would not get close to the minimum.

$$MAE(y_i, h(x_i)) = |h(x_i) - y_i|$$

Huber loss (HL) [26] is a function that basically smoothens the MAE acquiring the advantages from both, the MSE and MAE, that is: it is differentiable, and robust to outliers. Despite the other functions, Huber loss is a composed function that depends on a variable δ (hyperparameter) not very easy to set. Another alternative that is differentiable and robust is log-cosh, that is similar to Huber loss, but twice differentiable everywhere.

$$HL(y_i, h(x_i)) = \begin{cases} \frac{1}{2}(h(x_i) - y_i)^2 & \text{if } |h(x_i) - y_i| < \delta \\ \delta(|h(x_i) - y_i| - \frac{\delta}{2})) & \text{otherwise} \end{cases}$$

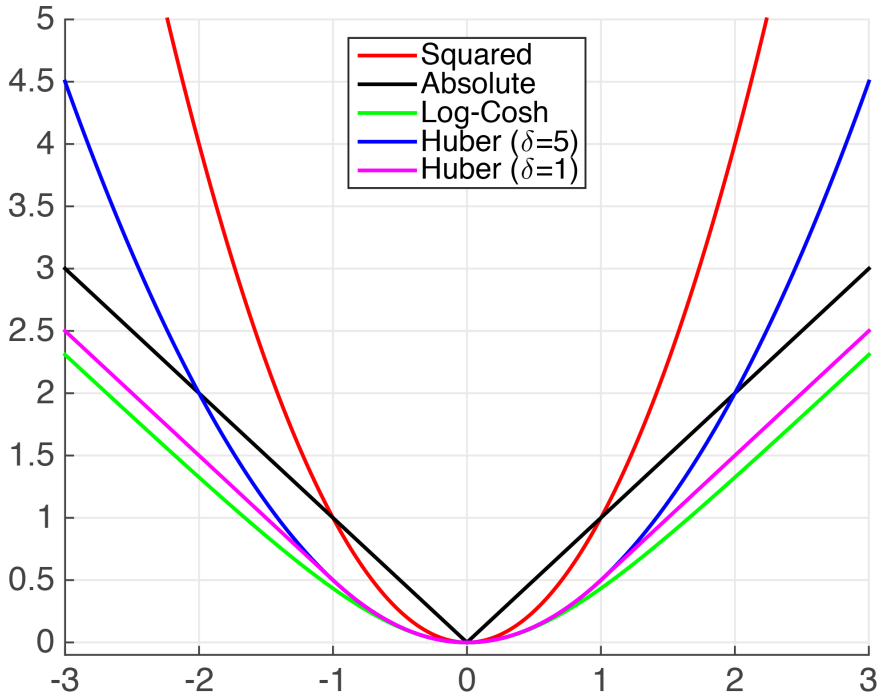


Figure 2.8: Plots of Common Regression Loss Functions - x-axis: "error" of prediction; y-axis: loss value

In the progress of this thesis project, MSE was the cost function being used. However, in the latest model the use of Huber loss was required over MSE because of the complexity of the problem. This progression is explained in further sections in Chapter 3.

2.3 Tensorflow

Tensorflow [26] is one of the many recent frameworks released for developing deep learning. This will be the tool used for training, and later, inferencing the neural net.

Deep learning is a very complex field that gathers together complex and very deep mathematical knowledge; requires the computation of long pipelines of operations, and thus, efficiency is a must have; requires the usage of complex programmatic tools such as using threading or using GPU calculations; has repetitive operations; and requires expertise in debugging and putting in production. All this reasons and many others that it may be being forgotten, are a huge motivation for many organizations to create frameworks that would add some level of abstraction over the art of developing deep learning. Many frameworks have been released in the latest years, each of them giving a different level of abstraction that would give more or less flexibility to the user, to ”play” over the neural nets.

Tensorflow is an open-source package under the Apache 2.0 license. This framework is of property of the giant Google, and nowadays has become a critical tool not only for people at Google but also worldwide. it was considered out of the scope of this thesis to do a study of the different deep learning frameworks. The reasons this frameworks has been chosen are:

- the potential of this tool is extremely high and is a good skill to have in the CV,
- the flexibility to implement things is enormous,
- has a big company supporting, maintaining, and improving it,
- has a growing community, with growing documentation,
- and despite it was a young project (still $<$ 1 year nowadays), it is an incredible active project.

2.4 Related work

Deep learning has proved to be a powerful tool by setting the state-of-the-art of several problems and various fields such as a human-like reader (WaveNet), Atari bot players, Alpha-go players, self-driving cars, and so on.

The climate science research is very active field, and many workshops are promoted annually where researchers can share their work. The workshop of the Climate Informatics group holds the top workshop for informatics applied to climate science. In the 2016s workshop [27] 34 papers were accepted, and 5 out of them included work related to neural networks. However, none of them are related to improving the climate simulators. Instead, three of them focus on

extreme weather event (extreme rainfalls, Tropical Cyclone, etc) prediction using satellital images [4], and two of them focus on spatio-temporal predictions to detect recurrent events [28]. So far, using deep learning for substituting classical functions in the climate science models has not been done yet, and hence, the research in this project is very innovative.

Nowadays, the state-of-the-art work on calculating the solar radiation uses RRTMG, which is based on the single-column correlated k-distribution reference model, RRTM [29]. It is the fastest implementation to calculate SR with a considerable level of accuracy that has been tested against the line-by-line SR model, LBLRTM. For the purposes of creating the ground-truth labels for the dataset, it has been used this state-of-the-art modeling of broadband radiative transfer as our baseline. The Python based interface to RRTMG can be found at [30]. The inferencing speed of RRTMG, which is the to-beat speed with the deep learning model developed in this project is 30 milliseconds/sample.

Regarding the deep learning field, the neural net developed, is based in classical architectures that use recent techniques mostly in the computer vision field. However, the usage of ConvNets for processing a vector of concatenated different parameters, as done in this project, has not been done. This approach is detailed in Chapter 3.

Chapter 3

Implementation

Chapter 3 accounts for the implementation phase of the degree project. The input data is described in detail; and the development process, the prototype models, the final model, and the put in production are explained.

3.1 Programming

Overall, working with deep learning means operating with data intensive computations. As described in Section 2.3, deep learning is a complex field, and for ambitious neural nets that would be big enough for learning a big data problem, having the software for implementing it, is not sufficient. The industry is trying to accommodate to the current needs and the big suppliers such as NVIDIA or Intel, that are producing modern hardware (GPUs, CPUs, RAM) that is more powerful and more oriented to the use of training deep learning networks.

In this section, the reader may find the tools used for designing, managing, implementing, debugging, testing, and putting into production the project.

3.1.1 Software

The first decision was to decide which framework to use for developing deep learning. The chosen framework was **Tensorflow**, several reasons are behind this decision explained in a previous Section 2.3. Then, the loyal programming language was **Python**, which was an easy decision, mainly because it is the main language for developing in Tensorflow; yet, this framework has API's for Java and

C++.

Debugging neural nets is a bit tricky, but Tensorflow proposes several alternatives such as checkpoints, visualization tools (Tensorboard), and a big community.

The programming has been done locally, and then pushed to GitHub free-of-bugs. The training was done remotely to a cluster with GPU's.

Some scripting in both Bash and Python, was required. The first one for automatizing repetitive routines. The second one was used for testing the nets using Jupyter Notebooks, and for doing some data analysis. Some of the libraries used are SciPy, NumPy, Pandas, and Matplotlib.

Finally, all the environment was configured using Anaconda.

3.1.2 Hardware

For working locally I had a laptop; a MacBook pro 2016 with Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 16GB DDR3 RAM.

Working in a research lab such as SICS was an advantage as it was possible to use modern hardware, with CPU Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, 25GB DDR3 RAM. And GPU Nvidia GeForce GTX 1080 8GB DDR5 RAM.

When referring the usage of a **CPU**, the CPU is from the MacBook pro. When referring the usage of a **GPU**, the GPU is the one from the cluster.

3.2 Methodology

The methodology of the thesis is detailed in Section 1.3. However, the development has been done using a very popular methodology for data science, and mentioning this methodology to the reader will help him to better understand the development process of the final solution.

This popular methodology is the called CRISP-DM (Cross Industry Standard Process for Data Mining) [12]. CRISP-DM was conceived 20 years ago, and still in 2009, a survey of the methodologies [31], described CRISP-DM as the *de-facto standard for developing data mining and knowledge discovery projects*. Today CRISP-DM is a popular choice despite its limitations: big data involves organizing big interdisciplinary human teams and CRISP-DM does not contemplate this; and the deployment phase is barely contemplated. In 2015, IBM released a new methodology called ASUM-DM (Analytics Solutions Unified

Method for Data Mining/Predictive Analytics) which is basically an extended version of CRISP-DM that refines the explained problems.

Therefore, for the sake of this project, it was decided to work using CRISP-DM as for the purposes of this project, it covers all the needs. With Figure 3.1 the reader can get a quick idea of the methodology.

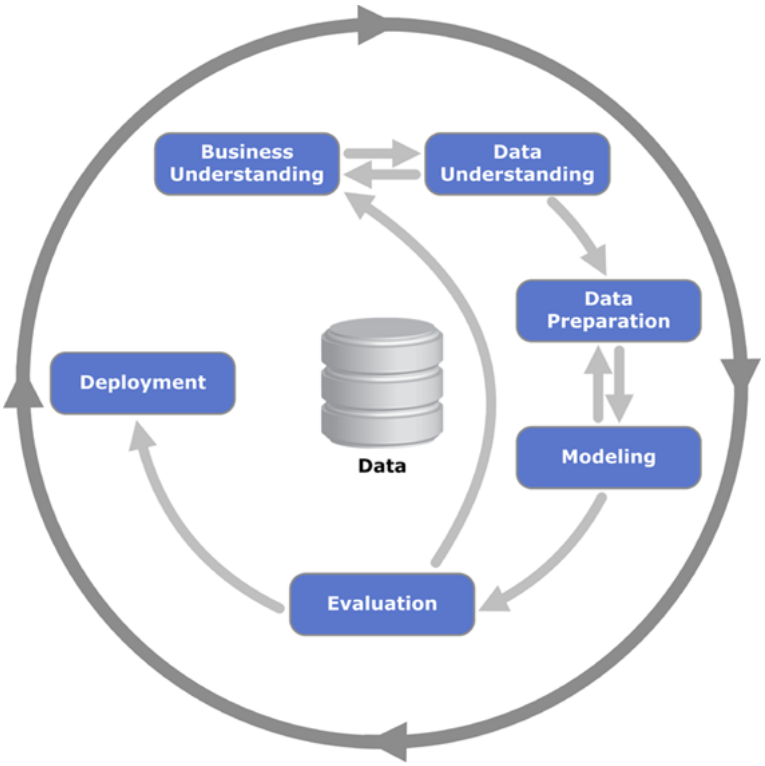


Figure 3.1: CRISP-DM methodology.

In this project, and in machine learning in general, iterating over the data is very common because a main point of failure is the quality of the data, being difficult to detect a bad quality data early, because of this and in general the complexity of climate science and deep learning, many iterations were needed. In the next sections of this chapter, the work performed at each phase of the CRISP-DM process is detailed. For simplicity, the data related phases (Business Understanding, Data Understanding, and Data Preparation) are grouped and explained in one single section, Section 3.3.

3.2.1 Overall picture

In the development of the neural net of this project three big cycles are distinguishable:

Initial model (IM): before starting this master thesis, some first exploration work [1] of few weeks had been done. This work was using a small dataset that was used for training a very simple model, rather small, that gave a first glance that it was possible to learn that data but was still having bad predictions. The code was a single script with basic Tensorflow (using an old version of it), and was missing the implementation of many features such as multi-threaded input queues or check-pointing. The number of levels were 26.

Basic model (BM): after taking the code of the previous work and refactoring it into a project-like structure, many improvements were done. The data changed, and the neural net was improved in several ways. The number of levels over the surface of the Earth were 26.

Final model (FM): once the BM got modeled to a point that the accuracy was practically 100%, the real work started. More parameters for the input, more granularity, and bigger data spectrum. The number of levels over the surface of the Earth were 16, 32 and 96.

The models will be explained in more detail in further sections and will be referencing these 3 cycles. It is important to understand that each cycle is not a model, but a set of iterations that gave as a result different models.

3.3 Data

The dataset has changed several times giving a total of 11 different versions. Table 3.1 summarizes the versions including the changes and their characteristics. Many versions of the data have been required because once jumped to the FM, it was realized that the problem was more complex than it was thought to be, and this is why, most of the versions belong to FM.

My role in this big project was to purely work in the modeling part while another member of the team worked in preparing the data; however, I still needed to understand the data and iterate together in the design of the new datasets. My opinion in this case was important as I was the one who observed how the data was performing with the neural net.

3.3.1 Features

Samples in the table, and in further sections, means a single observation that has all its set of features and includes the ground-truth value for solar radiation.

Version (data_v*)	Features					Size		Files	Format	Precision (decimals)	Levels
	P	CO_2	ST	T	H	Samples (millions)	Space (GB)				
1	V	-	-	V	V	0.4	1.5	4	CSV	4	26
2	V	-	-	V	V	1.6	6.1	16	CSV	10	26
3	V	V	V	V	V	1	6.1	10	JSON	15	96
4	V	F	F	V	V	2	11	20	JSON	15	96
5	V	F	F	V	V	2	2	20	JSON	15	16
6	V	V	V	V	V	4	4.2	40	JSON	15	16
7	V	V	V	V	V	10	57	100	JSON	15	96
8	V	F	F	V	V	4	22	40	JSON	15	96
9	V	F	F	V	V	4	7.6	40	JSON	15	32
10	V	V	V	V	V	∼19.1	104	4	JSON	15	96
11	V	V	V	V	V	∼19.1	105	19049	JSON	15	96

Table 3.1: All version of dataset used for working in the project. In features: P=pressure; CO_2=dioxide carbon; ST=surface temperature; T=air temperature; H=humidity; - = does not exist; V=variable values; F=fixed or static values.

This observation represents the exact number of features (measurements) in one specific point of the Earth, and has vector of measurements for as many levels that corresponds to the dataset. So, if the dataset has measurements for 6 levels, the vectors of features will have 6 measurements corresponding to each level. In Figure 3.2, this idea is represented.

In the features section of Table 3.1, *static or fixed* means that the data is static for all the samples, variable is the opposite. Pressure is always static over all the samples of each dataset. This is because the measurements of each level are measures corresponding to a specific height over the surface of the Earth. The measurements are taken over equidistant intervals within a range; those intervals are always the same for all the samples of each dataset, and the pressure at each of these heights remains always the same. In Section 3.3.4, the model's input is explained. As it can be seen, pressure is not contemplated; this is because the pressure is static for all the samples of the dataset, it is a constant, and therefore, the predictions will not depend in the pressure. It can be removed from the training.

As observed in Table 3.1, the features for IM and BM are exactly the same. The different between these two datasets, besides the number of samples, is the

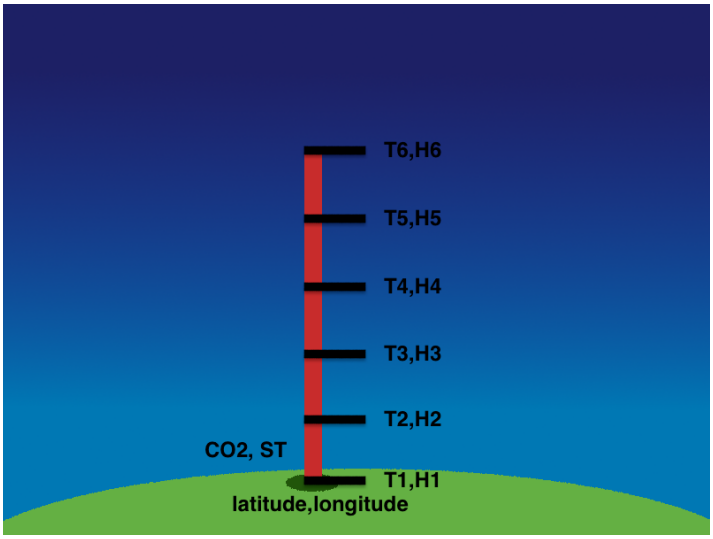


Figure 3.2: Illustration of the features of one sample with 6 levels.

precision. *data v1* had values with 4 decimal numbers, and *data v2* had 10 decimals of numerical precision. In both, IM and BM, the parameters were only pressure, air temperature, and humidity.

All the datasets for FM include CO_2 and surface temperature. However, in some cases, it was decided to go slower, and maintain these parameters static until it was managed to make the net learn the rest of the data. Once it was learned we were able to move to variable CO_2 and surface temperature.

Finally, the features showed in Table 3.1, for a sample of λ levels are the following ones*:
*Note**: after explaining each feature, some statistical values for each variable is included. These statistics are obtained from the latest dataset used, *data v11*. These statistics are the result of the data augmentation; therefore, if more data is created, these values may change a bit. However, due to the big amount of data generated, the new values should be near the actual statistical values.

P Vector of length λ. Pressure measured in Pascals (Pa).

statistics not calculated, this was an static value for all the samples

CO_2 Scalar. Dioxide carbon.

$min = 0.0$ $max = \sim 0.00999$ $mean = \sim 0.00173$ $std = \sim 0.002386$

ST Scalar. Surface Temperature measured in °C.

$min = 100.0$ $max = \sim 336.45$ $mean = \sim 280.73$ $std = \sim 17.71$

T Vector of length λ. Temperature at each level, measured in °C.

$$min = 100.0 \qquad max =\sim 337.81 \qquad mean =\sim 242.09 \qquad std =\sim 29.77$$

H Vector of length λ. Humidity at each level.

$$min = 0.0 \qquad max =\sim 32.483 \qquad mean =\sim 0.858 \qquad std =\sim 1.318$$

3.3.2 Ground-truth

The ground-truth for training the NN is the solar radiation calculated using the classical function for it. This process is illustrated in Figure 3.3. The range of the solar radiation is based in the input samples. For the final dataset, *data v11*, the statistical values are:

$$min = -61.514 \qquad max =\sim 43.820 \qquad mean =\sim -0.846 \qquad std =\sim 3.030$$

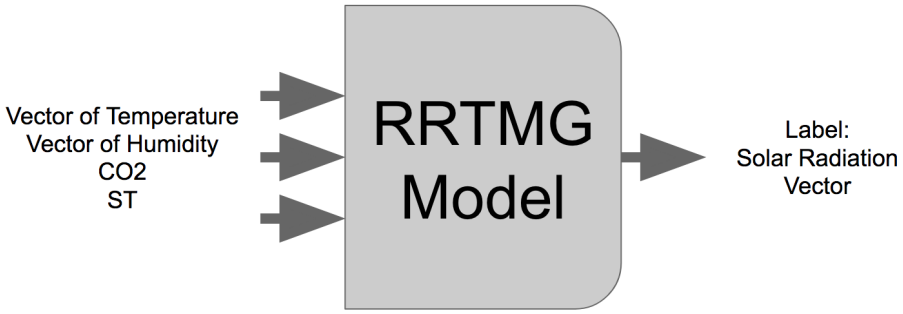


Figure 3.3: Illustration of ground-truth generation.

3.3.3 Data generation

Despite I did not prepare the datasets, I participated in their design, and it is of value to briefly describe how the data was prepared.

The origin of the data for the IM and BM models was a data augmentation over data outputted by the climate simulator. This data was totally biased to the basic data with which the data augmentation was done, but allowed to develop a first model faster as the training time was not really high. Working with a simpler

problem helped also to smooth the learning curve of deep learning and its related tools.

For the FM, the various datasets created were based in historical data [32] collected from 1979 at a rate of 1 measurement every 6 hours in different coordinates, and contain around 3.3 billion records. These measurements have been taken by the European Centre for Medium-Range Weather Forecasts (ECMWF): the data set is the ERA Interim [32] historical data. In total, all that data takes around 5TB. Having such amount of data was one motivation for this project; however, the measurements have only 16 levels, and for many simulators these granularity is not enough. As the aim was to train the model with fined granularity, using this data required some data augmentation, interpolating the levels in between.

For the data generated for FM there are two categories, (1) data that is purely randomly generated using a Gaussian distribution within the standard deviation and mean of the historical data at each level. And (2) data that is sampled from the historical data, and then receives some interpolation between levels. In the case of the interpolation, if there are 3 levels and it is wanted to move to 9 levels, in between of levels 1 and 2 of the base data, will be inserted three new values.

In the interpolation case (category 2) for the data generation it was decided to insert samples with smoothed transitions in the interpolation; and other more spiky samples with more random transition (within some standard deviation, using the Gaussian distribution) that would help to better generalize the problem, and hence not create a bias to the historical data.

3.3.4 Model Input

IM had an 8x8 mattrix as an input for the net per sample. This was an interesting approach because, as explained in 3.3.2 the solar radiation of one level is dependent on the variables of that level but is also influenced by the solar radiation in neighboring levels. Thus, there is a relation that allows to use convolutional nets. However, while reshaping (as explained in next paragraphs) the vectors into a NxN matrix, that reshaping may add a non existent relation between that layers that are contiguous in different columns; e.g., in Figure 3.2 (a) the first level is suddenly directly connected as neighbor of the third level. The guess is that, the net learns that in order to get low error, it has to recognize that in fact, there is not a direct relation from these virtual contiguity.

The final net trained in BM was using the same idea as IM, it had as an input an 8x8 matrix. In both cases, IM and BM, use 8x8 matrices, because there were

26 levels, and the parameters used to train were air temperature and humidity as represented in Figure 3.2 (a) for 6 levels. Therefore, 26 levels for temperature + 26 levels of humidity are 52, and then another 14 slots filled with 0 are required to fulfill the 8x8 matrix (64 slots). Pressure is not contemplated because as explained before, is a constant for all the samples in the dataset and does not influence the result of the model.

In the case of FN, the mixture of vectors and scalars parameters was a peculiar problem for NN. The approach that worked better was also to reshape the input into an NxN matrix concatenating the scalars on the first to positions and then concatenate pairs of temperature-humidity per each level. Finally, fill the empty spaces with zeros until filling all the spots of the NxN matrix. Therefore, each matrix constitutes a single sample for the NN. The pressure remains static for all the samples so again it is not included in the matrix. The desired model needs to be fined grained and hence, is trained for 96 levels; however, a simplified version for 6 levels over the surface of the earth is illustrated in table 3.2 (b).

(a) Input for IM and BM

H_1	T_1	H_2	T_2
H_3	T_3	H_4	T_4
H_5	T_5	H_6	T_6
0.0	0.0	0.0	0.0

(b) Input for FM

CO_2	ST	H_1	T_1
H_2	T_2	H_3	T_3
H_4	T_4	H_5	T_5
H_6	T_6	0.0	0.0

Table 3.2: Simplified input sample with only 6 levels over the surface of the Earth.

The performance of the NxN input was a case of study and other alternatives were considered. The first alternative was to avoid the virtual relation of not contiguous levels by training a net with input 4x96x1 (this last is the number of channels), where each row is a level of the sample with CO_2, ST, T, H. The second approach was to use 1D convolutions with vector and a channel per feature 1x94x4 (this last is the number of channels), where CO_2 and ST were replicated into a vector with length equal the number of levels. In summary, the input of the model was very unstable with a very broad spectrum that did not allow the net to learn. i.e., the net learned to some point where outliers ocationally generated a gradient explosion that practically lost all the learning; this happened with a lot of frequency. The NxN input did not give this problem and seemed more robust for the problem.

3.3.5 Input miscellaneous

Threading Deep learning is normally a data intensive problem, fortunately Tensorflow offers ways to optimize the training time as much as possible. One of the great alternatives it has, although a bit tricky to implement, is using Queue Inputs. This consists on instead of feeding directly the data, launch as many threads as desired, that will be reading the data in parallel with the main program that is managing the Tensorflow's connection with the GPU. The data is read into an asynchronous queue where Tensorflow knows will find the data for training.

Mini Batching For training the net, it was used mini batches of many sizes. 64 seemed to be give a good balance of improvement and training speed. The variation of the input was so big that using a bigger size of mini-batches would have lost detail when averaging, so 64 was a perfect. An smaller number would have delayed the convergence as going through the data would have taken more time.

Normalization At the beginning it was used normalization for the input and for the ground-truth. Then, it was decided to stop using a normalized output because the deployment of the net would have required denormalizing the output (solar radiation). Because of this, and because the data was starting to be so big for normalizing the input, the decision to stop normalizing the input was also taken. The results were exactly the same, the guess is that having the batch normalization (as explained in Section 2.2.3.2) layers throughout the net was having the same effects as the input normalization. All the normalizations were zero-mean normalizations.

Randomization Randomization of the input was used by default in the software. The idea was to collect all the data files in one array and then randomize the files to read at every epoch. Randomize at every epoch means that after going through all the data, the software starts another iteration (epoch) over the data again, the order of the data then is changed with regards to the last epoch. Therefore every training of the NN is different.

3.4 Modelling

In Section 3.2.1 the different cycles of training were mentioned: IM, BM and FM. In the following sections the architecture of the most significant models that were used within each cycle are explained in more detail.

3.4.1 Initial Model

As explained before this cycle was the first exploration in the data, and was the starting point for working in BM and FM. In Table 3.3 the architecture of the net is represented.

input	conv2-32	conv2-64	maxpool	conv2-128	maxpool	fc-512	fc-256	out-96

Table 3.3: SalmanNet [1]: net used for predicting solar radiation. conv layers are with stride and padding 1. maxpool layers are 2x2 filters with stride 2. conv2 means use of 2x2 filters for the convolution.

The characteristics of this nets are:

1. The NN represented in the image above was using Leaky ReLUs as activation function.
2. Those layers were put before maxpool layer.
3. Learning rate was 0.001.
4. Dropout 0.95.
5. Mini batch size 3.
6. Weight and bias random initialization with small values.

Even before the SalmanNet's, a simple implementation with a fully connected network with only 20 neurons was done. In Section 3.5.2 the results of these two nets is showed.

3.4.2 Basic Model

Based in MI, its code was totally re-implemented for simplifying the training process, and once it was refactored into a project-like software the improvement of the model started. Some of the features worth mentioning of this new software are:

- Set program parameters to speed up the hyperparameter tuning without necessarily touching the code.
- Have check-pointing controls every x (parameter option) steps and save checkpoint before manually finishing program.
- Use of Tensorboard for visualizing the training.

A good part of the project was spent here, first in re-implementing the code and second in improving the model.

When refining the net, after working with *data v1*, it was decided to move into another dataset with more precision and more samples, *data v2*. Once moved to *data v2*, the capacity of the model was not enough and adding more layers was required. In Table 3.4 some of the architectures tried are detailed.

ConvNet Configuration		
5 weight layers	6 weight layers	7 weight layers
A	**B**	**C**
input 8x8x1		
conv2-32 conv2-64	conv2-32 conv2-64	conv1-32 conv2-64 conv2-128
maxpool		
conv2-128	conv2-128 conv2-256	conv2-256 conv2-512
maxpool		
fc-512	fc-1024	
fc-256		
out-26		

*Table 3.4: **Radnet v1**. Set of more representative nets for BM. conv layers are with stride and padding 1. maxpool layers are 2x2 filters with stride 2. conv2 means use of 2x2 filters for convolutions in that layer.*

Each of the models came along with some other changes that are worth mentioning for further discussion in the Section 3.5:

Model A This model is exactly the SalmanNet model used in IM but without dropout, and trained without the use mini-batches. The first objective was to overfit the net, so the dropout was removed.

Model B In this model, the biggest implementation was adding BN layers (Section 2.2.3.2) alongside with the increase in capacity. The increase on capacity was using increasing neurons in the first fully-connected layer to 1024; and adding a 4th convolutional layer with 256 neurons.

Model C Finally, the biggest model, inserts a 5th convolutional layer of 512 neurons. Also, the first layer was converted into a convolutional layer with filter 1x1 instead of 2x2. This change is a layer inspired in the Google-net [20], that uses convolutions of filter 1x1 for reducing dimensionality. However, in this case it was used for augmenting dimensionality, known as a data augmentation technique. The last big change in this model was using Xavier initialization 2.2.3.4 in the fully-connected layers, the result discussed in 3.5 is surprisingly good.

The training time spent in these models was relatively short, 0.031 seconds per step in GPU in the heavier one, Model C. In general, it was rapid to see results when tuning hyperparameters.

3.4.3 Final Model

In this cycle, FM, the majority of the time was spent. After the very great results (explained later) obtained with *data v2* in the previous section, the important decision to jump to the real problem was done. The final aim of the project was to substitute really complex formulas with a deep learning model; yet, for training such model, being able to predict real solar radiation, with all its parameters and all the data spectrum, was needed. However, as showed in Table 3.1, many versions were needed mainly for two reasons. Some problems were encountered in designing a good dataset that would properly cover the spectrum of possible situations, and the data was so complex that making the NN learn it, was more than a challenging experience.

In Table 3.5, the most representative architectures for this second part of the thesis are detailed.

In the next lines of this section, the justification of the models are detailed in chronological order:

Model D The data used for training this model was randomly generated as explained in Section 3.3.3. When treating with this data (*data v3*), the team realized this was a really complex data to model, and that this problem was another level compared to the one in BM. To discard that the new parameters were

ConvNet Configuration						
data v3, v4	data v3	data v4	data v5, v6	data v7, v8	data v9, v7, v10	data v11
7 weight layers	8 weight layers	9 weight layers	10 weight layers	10 weight layers	8 weight layers	12 weight layers
D	**E**	**F**	**G**	**H**	**I**	**J**
input 14x14x1	input 20x20x1	input 14x14x1	input 6x6x1	input14x14x1	input 9x9x1 input 14x14x1	input 14x14x1
conv1-32 conv2-64	conv1-32 conv2-64	conv1-64 conv3-128 conv3-256	conv1-32 conv3-64 conv3-128 conv3-128	conv1-128 conv3-256 conv3-512 conv3-512	conv1-64 conv3-128 conv3-256 conv3-256	conv1-64 conv3-64 conv3-128 conv3-128 conv3-256 conv3-256
maxpool						
conv2-128	conv2-128	conv3-512 conv3-256	conv1-256 conv3-256 conv3-384 conv3-512	conv3-1024 conv3-1024	conv3-512	conv3-512 conv3-512
maxpool						
conv2-256 conv2-256	conv2-256 conv2-256	conv3-128 conv3-64		conv3-2048 conv3-4096	conv3-512	conv3-512 conv3-512
maxpool				maxpool		
	conv2-512					
	maxpool					
fc-1024	fc-2048	fc-2048	fc-1024	fc-2048	fc-2048	fc-2048
fc-256	fc-512	fc-512	fc-256	fc-512	fc-256	fc-256
out-96	out-96	out-96	out-16	out-96	out-32 / out-96	out-96

*Table 3.5: **Radnet v2**. Set of more representative nets for FM. conv layers are with stride and padding 1. maxpool layers are 2x2 filters with stride 2.*

the problem, a new dataset, *data v4*, was created having static CO_2 and surface temperature. However, the net was still unable to learn it.

Anyway, working with 96 levels was a real problem as the training time incremented significantly, and for observing if there was some improvement, letting the model run for more than 6 hours was required. After trying to tune different hyperparameters, and having continuously negative results, another approaches were starting to be taken.

Model E This model represents one of the approaches. Instead of putting the CO2 and surface temperature as scalars in the first two positions of the input matrix (as explained in Section 3.3.4). These two parameters should be replicated for all the 96 levels, so the input would be an array of $H_1, T_1, ST, CO2, H_2, T_2, ST, CO2$ ... and so on, giving a total of $96 * 4 = 384$

elements, that would fit in a 20x20 matrix.

Other approaches that are not represented in Table 3.5, but were tested at this moment, were: Trying an input of 4x96 instead of 20x20, to avoid non-existent relations as explained in Section 3.3.4. Or another alternative, also explained in the same section, that is the usage of 1D ConvNet with input 1x94x4, were the last 4 represents the channels, each of one corresponding to each of the features of the dataset.

Many other trials were done, such as considerably increasing the capacity with deeper nets of up to 12 trainable layers; still, the error remained the same. The problem was that the net was converging to a point were the radiation is an average, but did not train further than that. The guess is that the input was too broad that the net was being pulled to average value of each step (the gradient was too big because the error was too big), and in the next step was pulled to the other side. With the result that the net was not learning anything but sticking in the mean. Many times there were gradient explosions that also reseted the learning.

Model F After so many failures with the datasets *data v3* and *data v4*, the creation of a simpler dataset (*data v5*) was agreed. But in the meantime model F represents more trials, such as changing filter sizes and channels of the convolutional layers. Filters of 5x5 and 3x3 were tried and the extended use of 3x3 filters in many nets like VGG net, motivated to stay convolutional layers with filters of 3x3 instead of the 2x2 filters used in previous models.

Model G Once with access to the new dataset, *data v5*, the problem remained the same, no training at all. However, having a smaller data helped to speed up the training time, and thus to increment the feasibility of trying new things. The guess of the error explained few paragraphs before, motivated to understand the cost function being used, and the alternatives to it. Finally, the decision to substitute the MSE function with the Huber loss (Section 2.2.3.6) as cost function, resulted to be the key to advance in the project and learn *data v5* with model G. In order to test the model with variable CO_2 and ST, the *data v6* was created, and tested against this same model that successfully learned it.

Model H Unfortunately, the happiness did not last too much when moving to *data v7*, that had 96 levels with variable CO_2 and ST. Therefore, *data v8* was created, with fixed CO_2 and ST, to simplify a bit the problem, but the NN was still failing to learn data with 96 levels. With the cost function problem, that would solve the high spectrum problem, the network was still having

problems for learning. The guess here was that the net had not enough capacity and the neurons were missing the key patterns. Model H is one example of architecture with enormous capacity tried; however, the big drawback was that training such a big net was extremely slow even for our powerful GTX1080.

Model I The next logical move here was to test an intermediate dataset with 32 levels, *data v9*. The guess now was that the previous guess, discussed in the last paragraph, was not right because the model was not learning anything, just averages. Then, the idea that the neurons were dying came. This idea had not been contemplated before because Leaky ReLUs were supposed to mitigate the dying neuron's problem. Probably the complexity of the input was too high that Leaky ReLUs were not enough. Then, the task to implement a more sophisticated activation function, PReLU 2.2.3.3, started. Happily, dead neurons happened to be the disease , and PReLUs happened to be the cure. The *data v9* was learned and then *data v7* was also learned.

With *data v7* learned, some test with the real simulator started but failed. The problem was that *data v7* was data randomly generated, within some range based on statistics of the real historical data, but still random that did not have the natural smoothness between levels as the real data. For example, in all the cases of the data randomly generated, the temperature was one value at one level and suddenly in the next level the temperature was a totally different temperate with jumps like 30 °C of difference. The data was clearly bias to the random generator. To solve this problem, a new dataset, *data v10*, was created. This dataset consisted in sampling with rate of 0.05 over 4 years of the historical data, and then interpolated to 96 levels (remember the historical data had only 16 levels) using different techniques, creating samples with more and less smoothness. Model I had the problem that its net was too small that barely learned *data v10*, and the error was too high. Another problem was that the data was contained in 8 really big files and randomizing the samples at each epoch was not really possible.

Model J Finally, the solution was to split the data into chunk files of 1000 samples creating the dataset *data v11*, and to increase the capacity of the model with more layers (avoiding increase in breadth so the NN does not get too slow to train). Model J, represent the architecture of the ultimate NN used in this minor thesis.

3.5 Evaluation

In this section an overview of the different results achieved is showed.

3.5.1 Tools

With the purpose to debug and evaluate the different neural net architectures, the procedure was the following:

```
train: step 498600 - loss = 0.002709246706, (0.079 sec/step)
test: step 498600 - loss = 0.002673355862, (0.079 sec/step)
train: step 498700 - loss = 0.002994635142, (0.082 sec/step)
test: step 498700 - loss = 0.002420817968, (0.082 sec/step)
train: step 498800 - loss = 0.002272743033, (0.087 sec/step)
test: step 498800 - loss = 0.002087666886, (0.087 sec/step)
train: step 498900 - loss = 0.001370104961, (0.091 sec/step)
test: step 498900 - loss = 0.001980746863, (0.091 sec/step)
```

Figure 3.4: Output of the main program for the Model J architecture after 5 million steps.

1. Observe output of the program. The output was of the form: $< stepnumber >$ $, < error >, < time/step >$. This was printed every 100 steps, and was really useful to see quick results and to observe how the net was performing without any extra effort. Figure 3.4 is an example of output.

2. Use Tensorboard. This tool is a very powerful tool of Tensorflow that helps in debugging and visualizing the performance of the net. In this project it was used for comparing the performance among different trainings and evaluate the long-term learning of one specific NN.

3. Use of a prediction generator and several scripts for the evaluation of the predictions. These consists on generating *csv* files (one file per sample) with useful information of each prediction, such as error measurements, features, ground-truth, prediction, and sample identifier. The generator can rebuild a checkpoint in Tensorflow with some specific training. This was very versatile, as it could be used independently of the training, or even take a checkpoint of a running training and evaluate it.

   Then, with the generated *csv* files, some scripts in Python for evaluating

the predictions and the samples was possible. The scripts included things such as line plots of predictions or histogram of errors.

4. The last step, if everything before went as wished, was evaluating the model using the production library, explained in the next section 3.6.

3.5.2 Initial Model

(a) *Fully-connected NN*

(b) *SalmanNet*

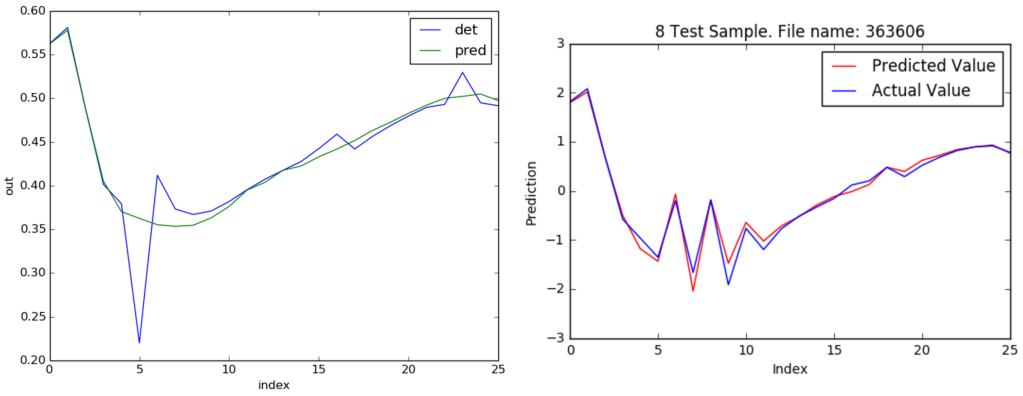


Figure 3.5: Results for the first cycle, previous to this project thesis.

As explained before in Section 3.4.1, the work in this first cycle was done before starting this thesis. Figure 3.5 shows two ancient plots of the results of both nets; (a) the fully-connected net with one hidden layer of 20 neurons; and (b) the SalmanNet. The plot (a) is clearly understanding the sample and averaging the value but does not have enough capacity for learning the spikes. With the introduction of ConvNets in (b), the results improved considerably; however, the prediction were not perfect and was still very distant of the objective of practically perfect predictions.

The next sections show the results achieved in BM and FM.

3.5.3 Basic Model

The second cycle, BM, included several small changes to the previous net. Figure 3.6 is a summary of important training losses over the net of this cycle. Line 1 is Model A but with the low precision dataset. Line 2 is Model A with the extended

dataset with 10 decimals. Line 3 is Model A trained with mini-batches. Line 4 is Model A with batch normalization layers. Line 5 is the model with another layer, bad placed, it is the previous version to Model B. Line 6 is the actual Model B. Line 7 is Model C that included Xavier initialization.

The starting of the training can be perfectly visualized in this figure (Figure 3.6), the lines for 1-4 are close from each other because they belong to the model with 3 conv layers. When jumped to lines 5 and 6, that belong to a net with 4 conv layers, the error falls down slower as more time to train the net is needed. However, something very impressive happens with line 7, the line that belongs to Model C with an added 5th conv layer, here the initial error curve starts very small and gets close to 0 very fast. This effect is explained in Section 2.2.3.4; and it is a clear proof that this works, and boosts the training time in the first steps incredibly saving more than 30k steps of training.

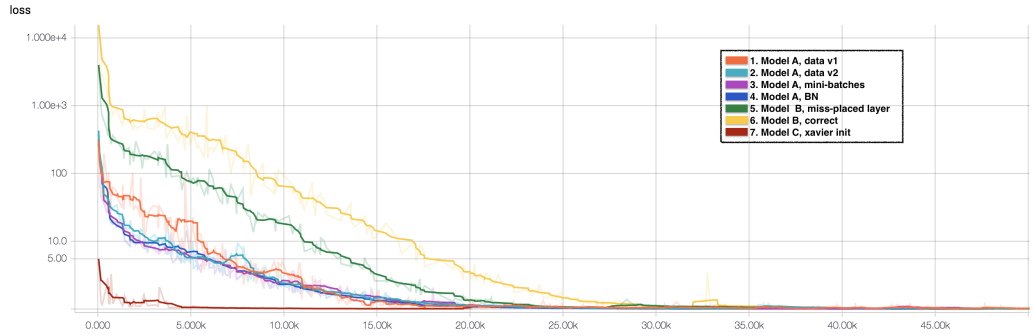


Figure 3.6: MSE error for train datasets. This plot shows the training line until 50,000 steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

Figure 3.7 is a zoomed image of Figure 3.6 in the interval 40k-50k of trained steps. The lines are smoothed and for example in line 1, the peaks of the error can be observed behind in a lighter orange color. In this figure, it is easy to observe how Model A improved the performance only by changing the data. The difference between line 1 and line 2 is enormous, specially in the peaks.

Figure 3.8 is a zoom in the y-axis over the Figure 3.7. Here the differences of the lines are better observed. One can see how the line 7 remains the best one with a very small error. Another important thing to notice is how line 4 with BN performs much better than the previous lines (1-3), this is because, as explained in Section 2.2.3.2, BN helps to improve the convergence and slightly gives better overall training.

In Figure 3.9 however, which shows the training between 80k to 95k, it can be observed that lines 3 and 4 got closer. This is because BN mainly improves the speed convergence, and that happens at the beginning of the training. Nonetheless,

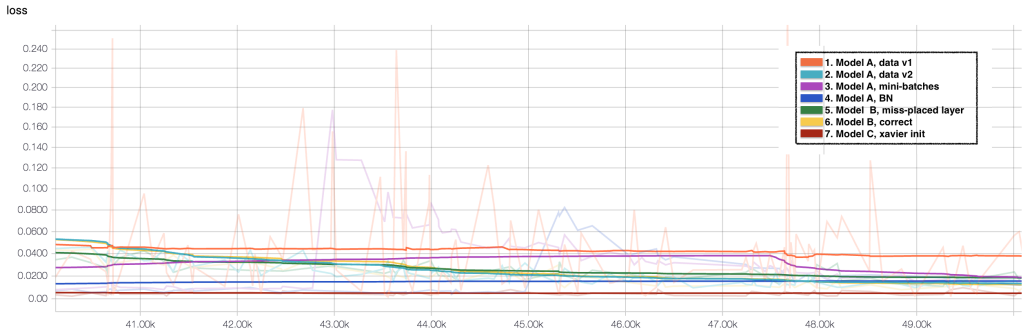


Figure 3.7: MSE error for train datasets. This plot shows the training line from 40k to 50k steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

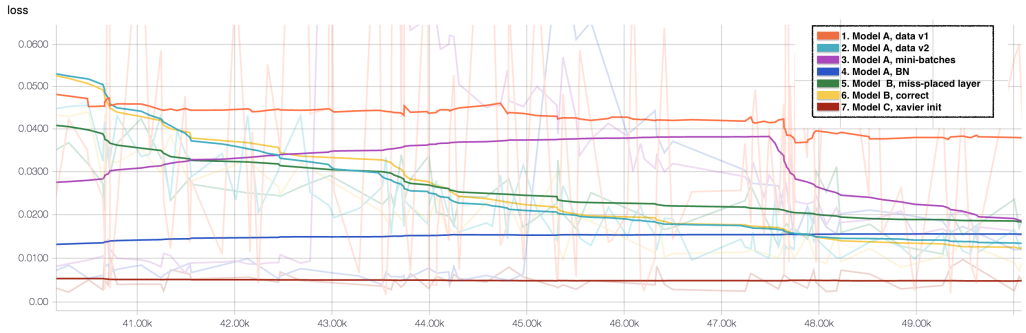


Figure 3.8: MSE error for train datasets. This plot shows the training line from 40k to 50k steps. x-axis number of steps ($K = 1000steps$), y-axis training mse error.

the "overall better training" is also observed, as line 4 remains having a slower error than line 3. Another interesting thing to observe in this figure, is the enormous difference between line 2 and 3, only by adding mini-batch training. This happens because the NN "sees" the data more times and faster, thus trains in less steps. Mini-batches also increases the training time; yet, the boost in learning pays off the extra time. At step 90k, line 2 took 2:30hrs and line 3 took 5:10hrs of training time in CPU. This is double the time, but with mini-batches of 64, line 3 was seeing 64 times more data.

Figure 3.10 shows the training in more advanced steps, between 110k and 140k steps. Here line 7, unarguably, remains being the best model. On the other hand, the non-mentioned line 5 is in a very bad shape compared to the rest. This line was the result of a miss-placed 4th layer with few neurons that was doing okay at the beginning but late step was not reducing the error as desired.

The lines 4 and 6 showed in Figure 3.11 were very close, and although the line 6 was getting slightly better, it was not enough. The model B belonging to line 6,

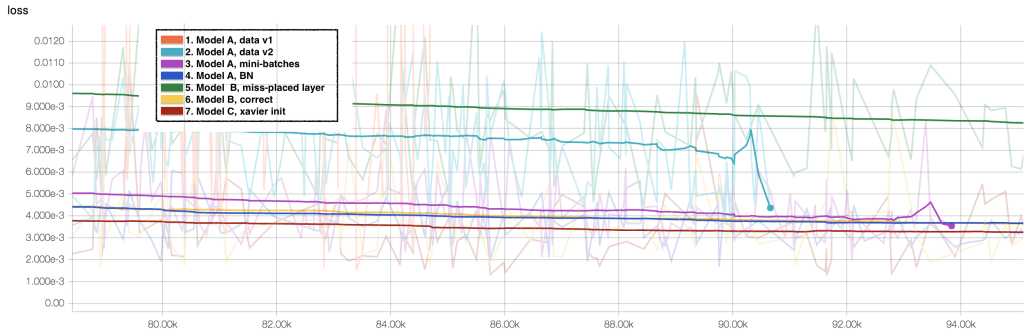


Figure 3.9: MSE error for train datasets. This plot shows the training line from 80k to 95k steps. x-axis number of steps ($K = 1000steps$), y-axis training mse error.

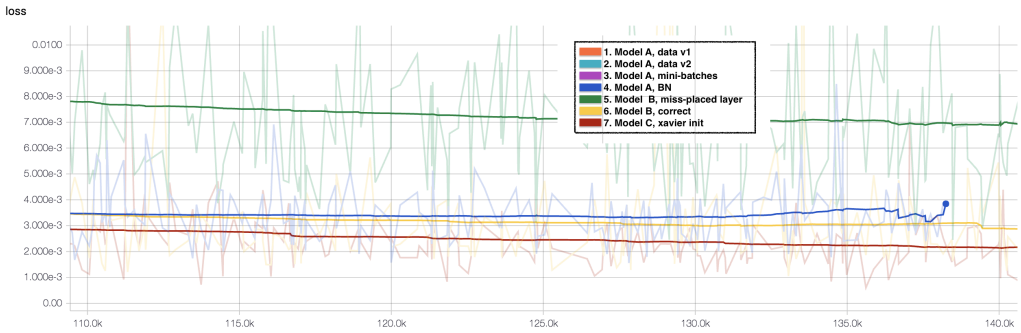


Figure 3.10: MSE error for train datasets. This plot shows the training lines from 110k to 140k steps. x-axis number of steps ($K = 1000steps$), y-axis training mse error.

despite having one more layer did not have enough capacity yet. The gap in this image among line 7 and line 4,6 was clearer.

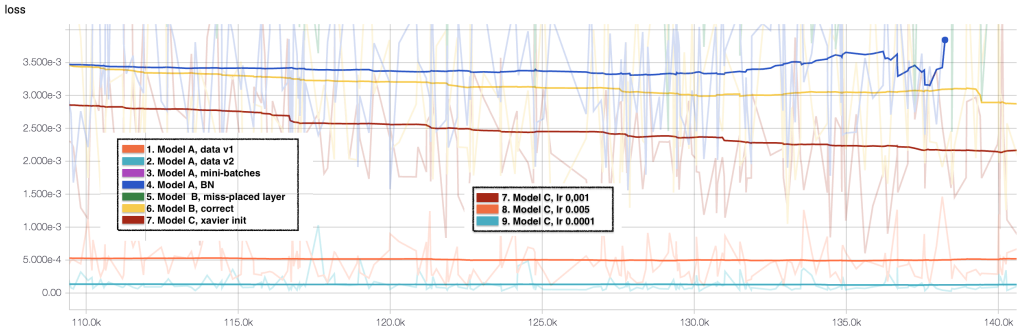


Figure 3.11: MSE error for train datasets. This plot shows the training lines from 110k to 140k steps. x-axis number of steps ($K = 1000steps$), y-axis training mse error.

This Figure 3.11, also introduces two new lines, lines 8 and 9. Together with line 7 they conform a one full training. This means, after the 150k of line 7,

with $learningrate = 0.001$, a new training using the checkpoint of this training was started with smaller $learningrate = 0.0005$. This new training is line 8, and after 150k steps of line 8, a last training with that checkpoint was done with $learningrate = 0.0001$ resulting in line 9. In Figure 3.12, the training until 150k is showed. This training was starting to be time consuming so it was run in GPU taking a sum of ∼8 hrs. The justification of the improvement when reducing the learning rate (lr) is because the learning algorithm can be more precise and not over jump the ”convexity” of the problem, so the minimization is maximized.

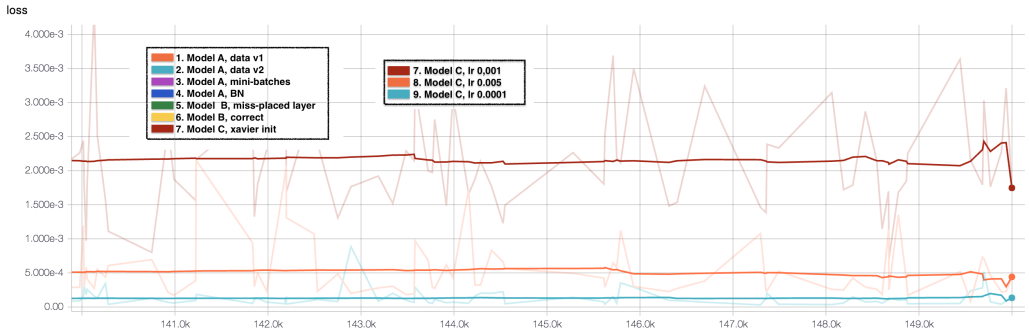


Figure 3.12: MSE error for train datasets. This plot shows the training lines 140k to 150k steps. x-axis number of steps ($K = 1000steps$), y-axis training mse error.

In y-axis of Figure 3.12, it can be observed that the average MSE error of line 7 is around 0.002. In the next image (Figure 3.13, with extra zooming for lines 8 and 9, it can be observed that the error for line 8 is ∼0.0005. The last best result for cycle 2 is the result for line 9 with an MSE error of ∼0.0001.

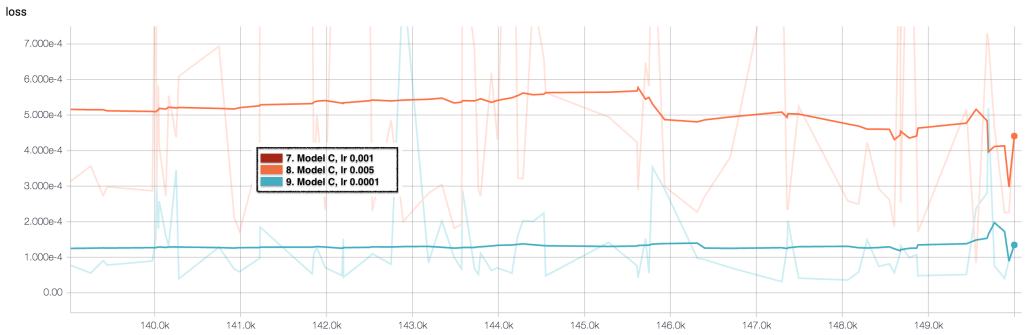
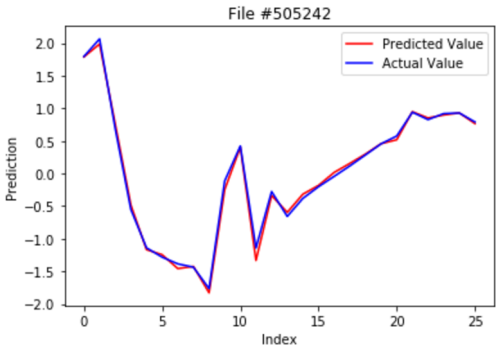


Figure 3.13: MSE error for train datasets. This plot shows the training lines 140k to 150k steps. x-axis number of steps ($K = 1000steps$), y-axis training mse error.

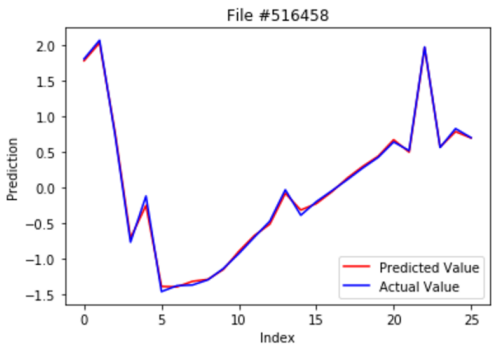
A perfect summary of the evolution in this second cycle is showed in Figure 3.14. These results are obtained using only the test dataset. The images selected

are specially selected to show the overall progression but of course, in all the cases, there were images with better predictions and with worse predictions.

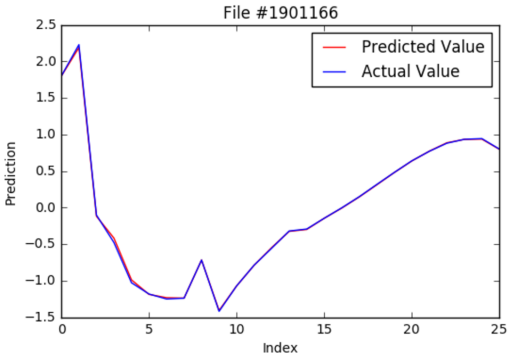
(a) *SR prediction line 2*



(b) *SR prediction line 4*



(c) *SR prediction line 7*



(d) *SR prediction line 9*

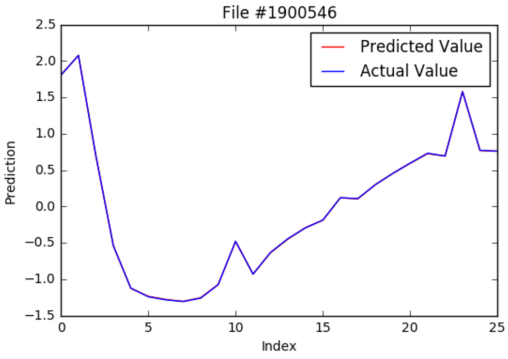


Figure 3.14: Results for the second cycle, previous to jumping to the really complex data.

3.5.4 Final Model

This is the third and final cycle where the focus was to concentrate in modeling for more realistic data.

Figure 3.15 shows a messy set of lines that are the most representative training of this cycle. Lines 1 and 2 are the logs of training Model D and F; the new data was very unstable as explained in Section 3.4.3, and the gradient explosions can be observed in form of high peaks. Line 3 is the hidden short line in the left-bottom corner of the figure, that short training was enough to know that Huber loss was helping to train, and then move forward to more complex data as the data being used was the data with only 16 levels. This line can be better observed in Figure 3.16.

Line 4, in Figure 3.15, is one of the many trials to make the NN learn, so the architecture used in this line is Model H that was very heavy with a really big number of neurons, to only train 200k steps more than 15 hours was needed.

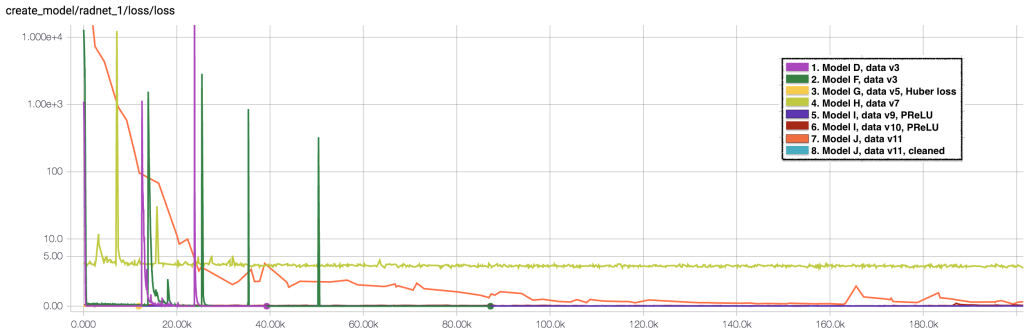


Figure 3.15: MSE error for train datasets. This plot shows the training lines until 190k steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

Then line 5, better observable in Figure 3.16, is the line that used ”panacea” PReLU as activation function, and finally succeed in learning the *data v9* with 32 levels. Then, with the same model but with *data v7*, with 96 levels (not represented in the figure), the result was also satisfactory.

At this point there was a problem of learning the behavior of the historical data because the data was being generated randomly only basing the generation in the statistical values of the historical data. So, as explained in 3.3.3 *data v10* was created, and line 6 shows the difficulties for learning the data v10. The NN was never falling to a level of low error (< 0.01), and the learning was being reseted once in a while.

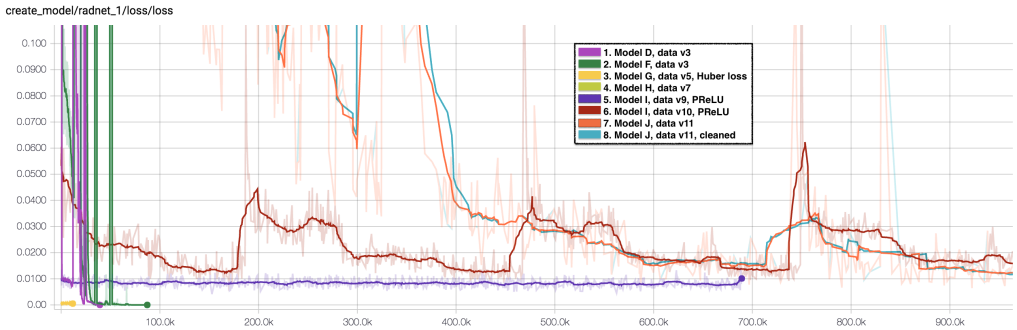


Figure 3.16: MSE error for train datasets. This plot shows the training lines until 1 million steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

Then, lines 7 and 8 are pretty much the same model, but line 8 is the actual architecture of Model J. Line 8 shows how the learning was difficult at the

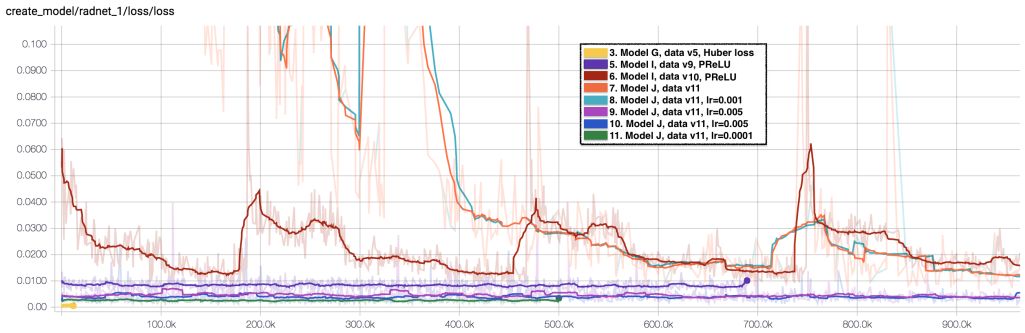


Figure 3.17: MSE error for train datasets. This plot shows the training lines until 1 million steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

beginning but then eventually the error was getting lower and lower, despite some remaining peaks that were not really delaying the learning, they were just outlier's peaks. The progression of the learning of line 8 can be better visualized in Figure 3.18. In fact, the learning of line 8 is prolonged to 2.5 million steps taking 1day, 18hrs, 11min to train.

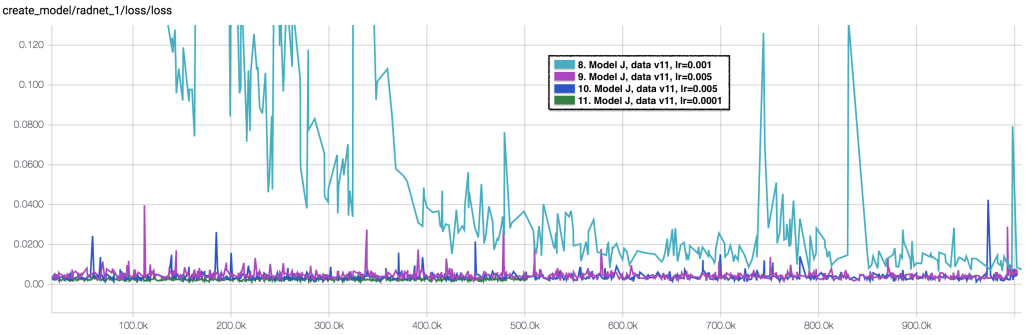


Figure 3.18: MSE error for train datasets. This plot shows the training lines until 1 million steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

In Figure 3.17, three new lines are introduced, lines 9, 10 and 11. Those lines, together with line 8, are a full training with a total of 86 hours of training over 5 million steps. Figure 3.19, with the smoothed lines, gives a better sight of the training in lines 9-11. Line 8 was trained using $learningrate = 0.001$ for 2.5 million steps; line 9 was trained using $learningrate = 0.005$ for 1 million steps; then, 1 million was not enough, so line 10 is the continuation of line 9 with other 1 million steps with $learningrate = 0.005$. Finally, line 11 is the continuation of line 10 with $learningrate = 0.0001$ over 0.5 million steps.

Figure 3.20 is a zoomed version of Figure 3.19. In this figure it is easier to observe the learning of lines 9-11. Line 9 had an average error of 0.0045, line

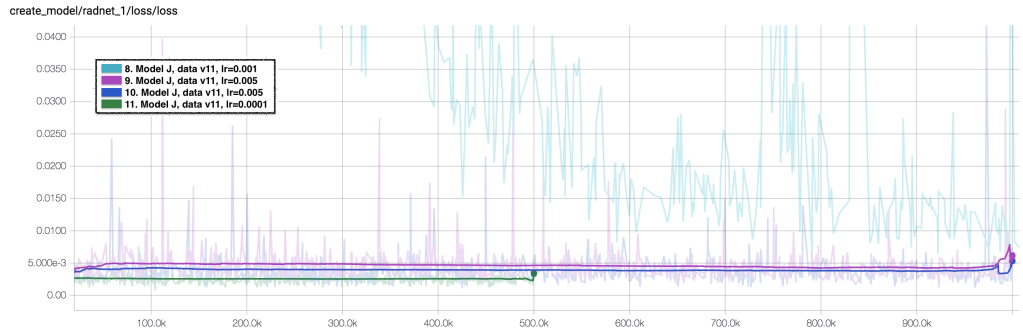


Figure 3.19: MSE error for train datasets. This plot shows the training lines until 1 million steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

10 had 0.004, and line 11 had around 0.0025 MSE error. These errors may not be as small as in BM, but the problem was wider and more complex. Probably, increasing the size of the net would improve the learning, but the time is crucial and the error in this final model is acceptable.

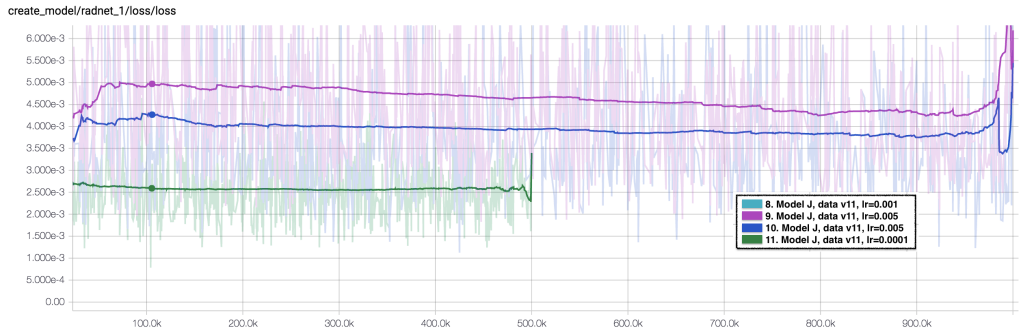


Figure 3.20: MSE error for train datasets. This plot shows the training lines until 1 million steps. x-axis number of steps ($K = 1000steps$), y-axis training MSE error.

Image 3.21 shows the problem once moved to *data v3*, where the NN was not learning anything. This image is the result of one of the many nets trained before using Huber loss.

In Figure 3.22 the result of introducing PReLU and Huber loss is showed. This sample is a sample of *data v7*. In the temperature and humidity plots, it can be observed that the transition between levels is not smooth at all. This is because, as explained before, this dataset was randomly generated, and did not have any data based in the historical dataset. When tested against real measurement, the model was not working adequately as showed in Figure 3.23. In this figure, the model is predicted to 96 levels and down-sampled to 32 levels. As observed, the result is not positive. This is the reason it was decided to create a new dataset with both,

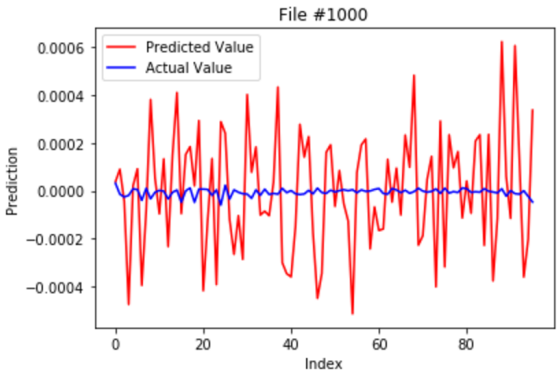
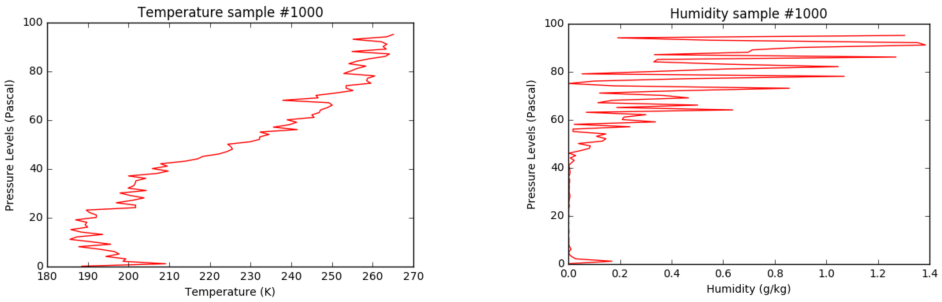


Figure 3.21: MSE error for train datasets. Predictions before Huber loss.

(a) *Temperature* **(b)** *Humidity*



(c) *SR prediction*

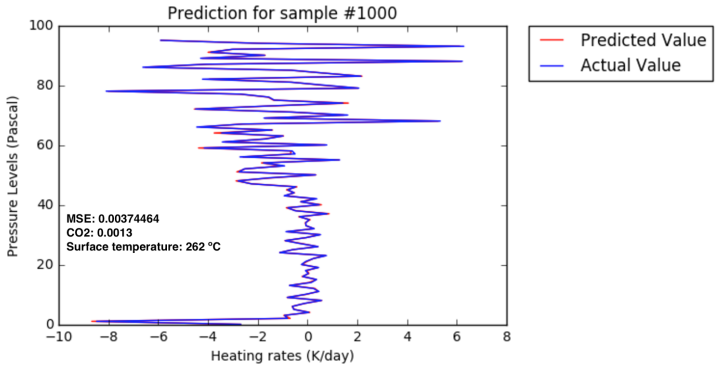


Figure 3.22: Results for the third cycle. This is a sample of the final dataset, data v7, and the neural net architecture, Model I. This is a result when PReLU was included.

random data and historical based data.

After retraining the Model I with new data, the results were some approximated predictions, but the errors were really high. In Figure 3.24 it can be seen how that prediction was good, but was not perfect. That was actually one of the best

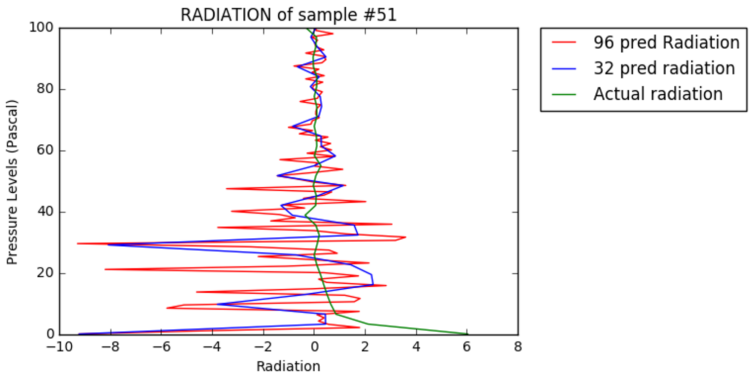
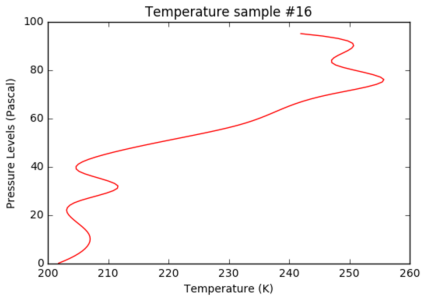


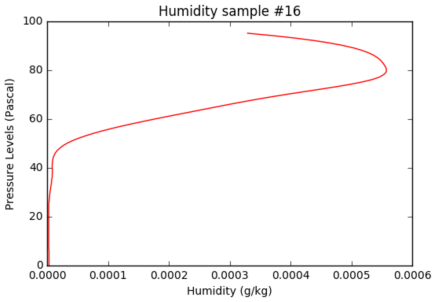
Figure 3.23: MSE error for train datasets. Wrong predictions for real data after training a dataset of randomly generated samples.

prediction examples.

(a) *Temperature*



(b) *Humidity*



(c) *SR prediction*

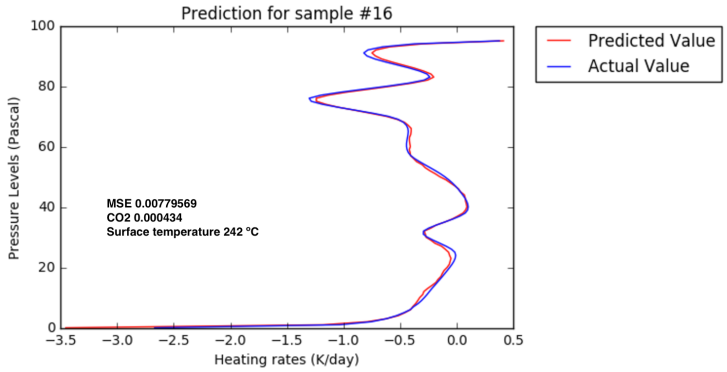
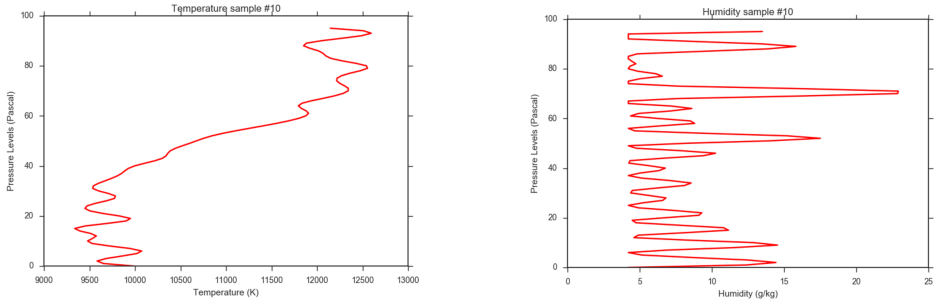


Figure 3.24: Results for the third cycle. This is a sample of the final dataset, data v10, and the neural net architecture, Model I. This is a result when PReLU was included. However, the new data was too complex.

Finally, when moved to Model J, the predictions improved over the wished

threshold. Figure 3.25 illustrates an example of a prediction for the best model; having a prediction of 97.65% of accuracy. Figure 3.26 summarizes 4 predictions of different samples of *data v11*. The plots in the top are two examples of samples with noisy and spiky transitions. The plots in the bottom are two samples with smooth transitions between levels.

(a) *Temperature* **(b)** *Humidity*



(c) *SR prediction line 11*

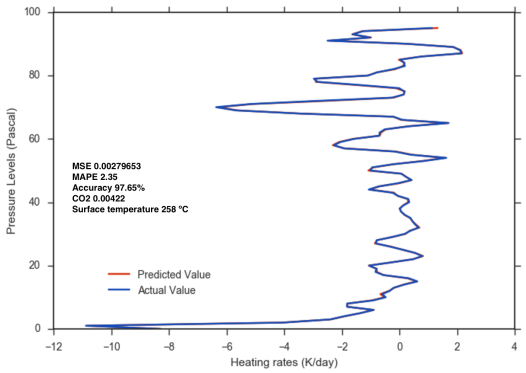


Figure 3.25: Results for the third cycle. This is a sample of the final dataset, data v11, and the final neural net architecture, Model J.

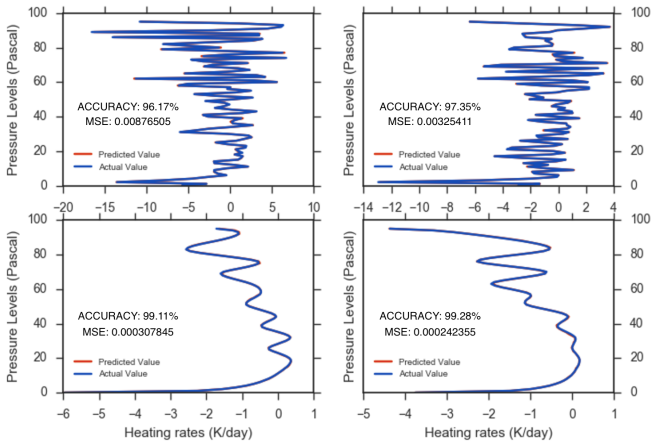


Figure 3.26: MSE error for train datasets. Summary of 4 predictions with the final model and the final dataset.

3.6 Deployment

In parallel of training the model, in between the long hours of waiting, the development of a library to inference the Tensorflow model was needed. The clear requirement was to **inference as fast as possible**. However, the common solution for Tensorflow is to use the inferencing module that launches an gRPC service for serving the module. This service is implemented to have high availability and is optimized to serve as much as 100,000 petitions per second.

The main problem of using this serving solution was the gRPC overhead that is built over HTTP/2, and thus, has the transfer delay. In the case of climate science, every millisecond counts so it was decided to implement the graph serialization and deserialization directly in a library, and inference locally instead of using the gRPC service. For such purpose, the Tensorflow API was of great help as it had method for "freezing" the execution graph into a file after training. This graph contains the value of the trained variables and the architecture. Then, this graph can be reloaded in the inferencing library into memory using Tensorflow again. The model is then ready to be asked to generate predictions.

The latest model is training to be fine grained, and gives predictions for 96 levels samples. In many cases, the applications that use this model do not use 96 levels. Therefore, a pre-processing and post-processing was included to interpolate to 96 levels in order to prepare the input matrix with the required levels. Finally, after the prediction has been done, a downsampling of the output (solar radiation) is done to the initial number of levels. The interpolation performed is a cubic point-to-point interpolation called cubic spline interpolation [33]; an

(a) *Temperature* **(b)** *Humidity*

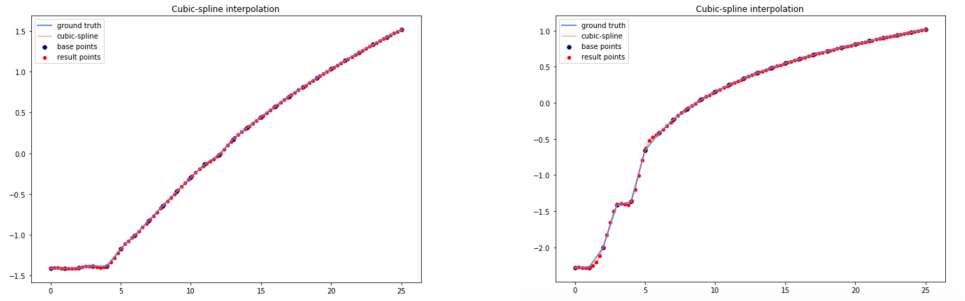


Figure 3.27: Results for the third cycle. This is a sample of the final dataset, data v11, and the final neural net architecture, Model J.

example of this interpolation can be seen in Figure 3.27. This interpolation is not the same one performed for data generation, in this case it is a cubic interpolation, and in the data generation case it was an stochastic process within some standard deviation over a Gaussian distribution.

Chapter 4

Analysis

Chapter 4 includes the analysis and discussion of the overall project and final results. The analysis will be done over the two main branches, speed of making one prediction and fiability of the prediction.

4.1 Fiability

The objective is to speedup the execution of the climate simulators that require solar radiation calculations by using deep learning for predicting the solar radiation. However, even more important than speeding up the execution is to accurately predict the solar radiation. This is, have the minimum error as possible compared to the ground-truth solar radiation.

4.1.1 Metrics

In order to measure the results of the model three main metrics have been used:

1. **Mean Squared Error (MSE).** This is a very common measure for continuous variables that measures the average of the squares of the errors. When the error is the difference between the actual value and the predicted value. In the formula for MSE below, $\hat{Y}$ is the prediction and Y is the actual value.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

This error has been showed throughout all the evaluation in Section 3.5. And this formula is the way it was calculated. At the beginning of the project MSE was used as a cost function too, but then it was showed that MSE was not a good function for training and the Huber loss was used.

2. **Symmetric Mean Absolute Percentage Error (SMAPE).** SMAPE is a modified MAPE (Mean Absolute Percentage Error) in which the divisor is half of the sum of the actual and forecast values. MAPE is the classical alternative, but has several drawbacks when the error is small below 1, and even it is not defined at zero, where the formula breaks. SMAPE has been redefined many times in the past, and currently there are many versions with slight changes being used. The used one in this work is a SMAPE formula that ranges from 0 to 1 (then multiplied by 100 to make it percentage). This formula has symmetric in its name because some versions can give the direction of the overall prediction. In this version however, the direction is lost when applying the absolute formula to both the nominator and the denominator. The formula is:

$$SMAPE = 100 * \frac{\sum_{i=1}^{n} \left|\hat{Y}_i - Y_i\right|}{\sum_{i=1}^{n} \left|\hat{Y}_i\right| + |Y_i|}$$

It is important to understand that this error is relative to the sample. This means that two predictions that may have a similar MSE, can have a different SMAPE error. One scenario where this can happen is when the range of solar radiation is in one case -20 to 5 and in the second one is from -7 to 2. If both cases have a similar MSE, the first case that has more amplitude is having a better prediction so the SMAPE may be smaller for the first case than for the second case.

3. **Accuracy**. In order to obtain a representative accuracy for a regression prediction it was needed to calculate SMAPE, that would give a percentage error. Then, the accuracy is calculated:

$$Accuracy = 100 - SMAPE$$

4.1.2 Discussion

The best model trained, belonging to the architecture Model J and explained in Chapter 3, is considered to be the final solution for this thesis project. This model has taken 86 hours of training over 5 million steps learning that learned over 15 million samples doing a total of $\sim$20 epoch over all the dataset.

Figure 4.1 is a summary of the study of 100,000 samples taken randomly from the test dataset, that contains 4 million samples. This means that the model has not been trained with this samples. In this figure, two histograms are showed, the histogram on the left is a summary of the SMAPE measure for this 100,000 samples that follow a Gaussian-like distribution but rather leptokurtic with mean (1.29). The plot in the right represents the histogram with a summary of the MSE measurements. In this second plot, 2 mountains are observable, the big mountain near 0 and the second mountain near 0.003. This might because the dataset contains a mixture of two types of data: data with smoothed transitions between levels; and other data with spiky and noisy transitions. The distribution of the MSE curve has a big positive skewing approximating the curve to a power law distribution. This means that most of the samples have good predictions, but some of the are having less good prediction; nonetheless, this predictions are still pretty decent, being below 0.02 MSE.

Overall the MSE error shows a better picture of the data, because the SMAPE error is an error relative to each sample. Therefore, MSE was the main error for comparing models. Nonetheless, SMAPE gives a good idea of ”how far are we from the perfect prediction”, and therefore, after finding the best model, SMAPE was of great importance to study how good is the model against the ground-truth.

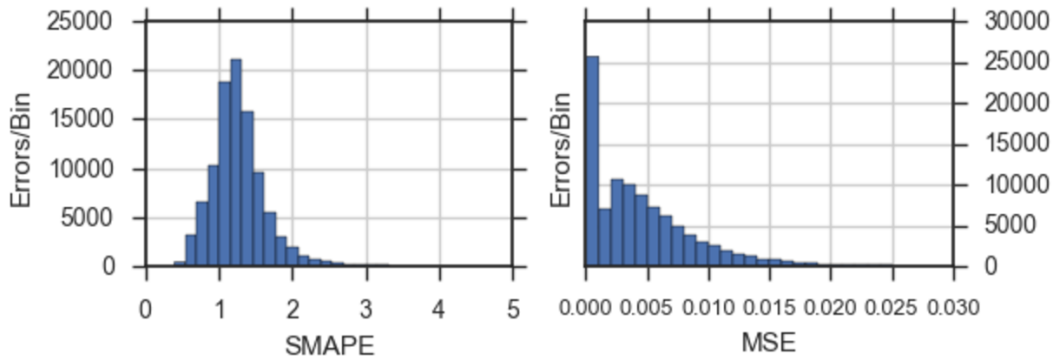


Figure 4.1: Histogram of the SMAPE and MSE measurements for 100,000 samples of the test dataset. The y-axis is the number of samples that had the error belonging to x-axis's bins.

With an average MSE of 0.0049, and an average Accuracy of 98.71% we consider that the results are above expectations and are applicable to real simulations. It is important to understand that climate models are already assuming errors, so such a small variation on these predictions over the ground-truth values given by the state-of-the-art model is considered to be more than enough for accepting the deep learning model as a good alternative.

4.2 Speed

When the requirement of having a good prediction was accomplished, the time of training is the key factor. The inferencing library, explained in 3.6 was developed to minimize the fetch operation, and to predict results as fast as possible.

4.2.1 Metrics

The only metric used for measure the inferencing speed is **time/prediction**. If one thinks in the problem itself, and how the inferencing library works, the inferencing library has an overhead for loading the model into memory. This overhead is negligible when millions of fetches over the mode are performed. Some the important part is to minimize as much as possible the fetching operation including the pre-processing and post-processing.

4.2.2 Discussion

In total, the time required for loading the model into memory is $\sim$0.85 seconds. As explained before this is done only once so does not represent an issue.

The loading time of the model and the inferencing time per sample are totally dependent on the size of the model. The state-of-the-art climate model for calculating the solar radiation, as explained in Section 2.4, takes around 30 milliseconds per sample. In order to minimize the inferencing time, the size of the NN was always a thing to bear in mind. Unfortunately, the large spectrum of the input made it impossible to read the desired error with Model I, and then it was required to increment the capacity of the model by almost doubling the size of the net from 4032 to 6624 neurons.

In model I the average inferencing time was $\sim$10 ms in CPU and $\sim$3.2 ms in GPU. The average time for inferencing the model trained with the architecture Model J is an average of $\sim$17 ms in CPU and $\sim$4.6 ms in GPU. Despite the fetch time increased with the last model, the speedup against the fastest model is of almost 2 times in CPU, and 6.5 times in GPU.

The speedup in GPU is, in summary, an amazing result that would speedup the climate science simulators enormously.

Chapter 5

Conclusions

Chapter 5 is the last chapter and concludes the degree project with the final results obtained throughout the course of the study. In this chapter recommendations for future work are also given.

5.1 Conclusions

This thesis project empirically proves that deep learning is applicable to the field of climate science. Figure 3.26, shows a summary of the final performance of the NN with architecture Model J explained in Section 3.4.3. This NN has been evaluated against the solar radiation calculated using the state-of-the-art algorithm, RRTMG algorithm [30]. The tests have been performed using a machine with a *GPU Nvidia GeForce GTX 1080 8GB* and an *Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz*. The prediction speed using our trained NN model is $\sim$4.6 ms per sample on the GPU or $\sim$17 ms per sample on the CPU. Therefore, they are fastest alternatives against the fastest state-of-the-art alternative, RRTMG, that takes $\sim$30 ms. These speedups leverage an improvement of 7x speedup in GPU or 2x speedup in CPU in the calculation of the solar radiation.

The most important part is that the speedup achieved is using a model that predicts the solar radiation with an accuracy of 98.71% and a MSE of 0.0049 over 100,000 samples randomly chosen from the test dataset. If this solution is validated and accepted by the climate community, the research performed in this thesis would enable more ambitious simulations thanks to the enhancement on the speed of the algorithm.

If the deep learning solution was not fully accepted by the community, the

deep learning model could be used to evaluate many climate scenarios faster, and then validate the results of the most interesting ones with simulators that use the RRTMG algorithm.

The task of obtaining such predictions, was not simple. The main problems that were needed to overcome in this project, are the following:

1. *Obtain really high accuracy.* To comply with this requirement, the NN used was refined with the latest techniques in deep learning.

2. *Generalize to noisy cases.* In solar radiation computation, clouds influence the SR output in the specific levels they are in. The clouds basically increase the humidity and produce peaks of solar radiation in those levels. They are basically random, so it was required to augment the data with noisy samples, resulting in a considerably more complex problem to modelize.

3. *Limit the size of the net.* In order to minimize the time cost for doing a prediction with the deep learning model, it was needed to keep the size of the NN as small as possible. More layers mean more calculations, and hence, more time to fetch.

4. *Manage big data.* Having such a big amount of data to learn was a real problem as the number of historical records is around 3.3 billion samples. The final model was trained over a dataset of 19 million samples. This dataset was the result of sampling 4 years of the total data with a rate of 0.05. But still, the size of the sampled dataset was 105GB. The best NN, for doing 20 epochs over that data, took 86 hours, with a total of 5 million steps.

To conclude, some of the lessons learned in this project are:

1. Simplifying the initial problem and working with a ”toy” case at the beginning (approach proposed by the supervisors) has proved to be such an effective approach that without it, the outcome of this thesis may not have been as successful as it has been.

2. Training big data with deep learning is a complex field that takes time. The experience and the deep understanding of deep learning plays an important factor when deciding what hyperparameters to use. Each trial is expensive and needs to be as precise as possible.

3. Using a powerful GPU is everything for such problems where enormous amounts of data are available. While at the beginning minimizing the size

of the NN was important for minimizing the inference time. At the end, the main motivation to keep the NN small, was to not increment the training time as a very long training was needed for ”learning” the dataset.

4. Having a good dataset is not easy. After training a dataset with randomly generated samples, the net was not generalizing to the real data. This was because in fact, all the data was so spiky that the real data was not like that; therefore, the model was not seeing real-type data and hence, was missing the predictions when launched with a test with real data. It sounds trivial, but it took so much time to realize about this, that this problem was not detected until the model was already trained and tested.

Ultimately, the work done in this project led to the publication of a paper that can be found in Appendix A. The code developed in this thesis project can be publicly found in the GitHub repository of the author [34].

5.2 Future work

Overall, all the objectives of the thesis have been covered. But, as this thesis project is placed within the boundaries of a bigger project, some work is still yet to be done. Future work beyond the thesis' objectives, that are recommended to be done, are:

- Validate the model in a real simulation.
- Improve the results by training with more data and with more time.
- Generate a train dataset not only with 4 years but for the whole dataset of 37 years of historical data.
- Reduce the inferencing time with a deeper study of the alternatives. Maybe tighten the net even more, making it smaller and thus, lighter to fetch.
- Implement a distributed environment with more GPU's with the Spark integration with Tensorflow. It is still very recent and probably unstable, but the speedup in training may be worth it.
- Think on new problems where deep learning is applicable in the climate science field, and optimize them.

Bibliography

[1] Salman Niazi and Shadi Issa, "Id2223 project," 2016. [Online]. Available: https://github.com/smkniazi/id2223-proj

[2] J. H. Faghmous and V. Kumar, "A big data guide to understanding climate change: The case for theory-guided data science," *Big data*, vol. 2, no. 3, pp. 155–163, 2014.

[3] A. Lupo and W. Kininmonth, "Global climate models and their limitations."

[4] Y. Liu, E. Racah, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, W. Collins *et al.*, "Application of deep convolutional neural networks for detecting extreme weather in climate datasets," *arXiv preprint arXiv:1605.01156*, 2016.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[6] S. Easterbrook, "climate science vs weather forecasting," 2010. [Online]. Available: http://www.easterbrook.ca/steve/2010/01/initial-value-vs-boundary-value-problems/

[7] ——, "Balloon analogy, climate vs weather prediction," 2014. [Online]. Available: http://www.easterbrook.ca/steve/2014/02/weather-balloons-vs-climate-balloons/

[8] "Call for high-performance computing and data storage systems for climate science." 2017. [Online]. Available: https://www2.cisl.ucar.edu/user-support/allocations/climate-simulation-laboratory-csl

[9] T. Stocker, *Climate change 2013: the physical science basis: Working Group I contribution to the Fifth assessment report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, 2014.

[10] R. Caballero, *Physics of the Atmosphere*. IOP Publishing, 2014, vol. 150. [Online]. Available: http://iopscience.iop.org/chapter/978-0-7503-1052-9/bk978-0-7503-1052-9ch5.pdf

[11] T. Shimazaki and L. C. Helmle, "A simplified method for calculating the atmospheric heating rate by absorption of solar radiation in the stratosphere and mesosphere." *NASA*, Jan. 1979. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19790008322.pdf

[12] C. Shearer, "The crisp-dm model: the new blueprint for data mining," *Journal of data warehousing*, vol. 5, no. 4, pp. 13–22, 2000.

[13] Steve Easterbrook, "Computing the Climate: How Can a Computer Model Forecast the Future?" March 2014.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. doi: 10.1007/s11263-015-0816-y

[17] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, J. Furnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 807–814. [Online]. Available: http://www.icml2010.org/papers/432.pdf

[24] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[25] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks." in *Aistats*, vol. 9, 2010, pp. 249–256.

[26] P. J. Huber *et al.*, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[27] A. R. E. I. E.-U. C. M. D. N. S. E. A. Banerjee, W. Ding; J. Dy; V. Lyubchich, "Proceedings of the 6th International Workshop on Climate Informatics: CI 2016." *NCAR*, 2016.

[28] L. D. Arthur Pajot, Ali Ziat and P. Gallinari, "Incorporating prior knowledge in spatio-temporal neural network for climatic data," *NCAR*, 2016.

[29] S. A. Clough, M. W. Shephard, E. J. Mlawer, J. S. Delamere, M. J. Iacono, K. Cady-Pereira, S. Boukabara, and P. D. Brown, "Atmospheric radiative transfer modeling: a summary of the AER codes," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 91, no. 2, pp. 233–244, Mar. 2005. doi: 10.1016/j.jqsrt.2004.05.058

[30] Joy Merwin Monteiro and Rodrigo Caballero, "The Climate Modelling Toolkit," in *Proceedings of the 15th Python in Science Conference*, Sebastian Benthall and Scott Rostrup, Eds., 2016, pp. 69 – 74.

[31] Ó. Marbán, G. Mariscal, and J. Segovia, "A data mining & knowledge discovery process model," *Data Mining and Knowledge Discovery in Real Life Applications*, vol. 2009, p. 8, 2009.

[32] "Climate science data measure from 1979 to 2017," 2017. [Online]. Available: http://apps.ecmwf.int/datasets/data/interim-full-daily/levtype=sfc/

[33] T. Lyche, "Discrete cubic spline interpolation," *BIT Numerical Mathematics*, vol. 16, no. 3, pp. 281–290, 1976.

[34] A. Alpire, "RadNet: Deep Learning for Climate Science. Solar radiation prediction," 2017. [Online]. Available: https://github.com/capiman13/radnet

Appendix A

Predicting Radiative Transfers using a DeepNeural Network

Predicting Radiative Transfers using a Deep Neural Network

Adam Alpire*, Ying Liu†, Jim Dowling*, Joy Monteiro† and Rodrigo Caballero†
*Royal Institute of Technology, Stockholm, Sweden
Email: {alpire, jdowling}@kth.se
†Stockholm University, Stockholm, Sweden
Email: {liu.ying, joy.monteiro, rodrigo}@misu.su.se

Abstract—Simulating the global climate in fine granularity is essential in climate science research. Current algorithms for computing climate models are based on mathematical models that are computationally expensive. Climate simulation runs can take days or months to execute on High Performance Computing (HPC) platforms. As such, the amount of computational resources determines the level of resolution for the simulations. If simulation time could be reduced without compromising model fidelity, higher resolution simulations would be possible leading to potentially new insights in climate science research. In this paper, we examine an important part of climate modeling involving the calculation of the broadband radiative transfer that takes around 30% to 50% time of a typical general circulation model. Our study presents a convolutional neural networs (CNN) to model this most time consuming component. As a result, swift radiation prediction through our deep neural network achieves a 7x speedup compared to the calculation time of the original function. The average prediction error (MSE) is around 0.004.

I. Introduction

Long simulations using high resolution climate models are needed to explore key questions in climate research, particularly changes in weather extremes such as extreme windstorms and precipitation events. In this paper, we describe how to reduce simulation time for climate models by replacing a computationally expensive simulation component called Radiative Transfer (RT) with a convolutional neural network (CNN). Our profiling of the EC-Earth climate model (http://www.ec-earth.org) shows that ∼30% of the total time is spent on RT.

Many state-of-the-art climate models, for example the EC-Earth model, use the Rapid Radiative Transfer Model for GCMs (RRTMG). RRTMG is based on the single-column correlated k-distribution reference model RRTM [1]. It is among the fastest implementations to calculate broadband RT with a considerable level of accuracy by testing it against RRTM. We use this state-of-the-art modeling of broadband RT as our baseline. We reduce its calculation time by generating predictions from a trained neural network model that retain a high level of fidelity to the original RRTMG model. In this exploratory study, we limit our scope to the longwave radiative component.

Recent advances in neural networks (NNs) have improved state-of-the-art results for pattern recognition tasks. In particular, CNNs have achieved impressive results for image classification [2], while recurrent neural networks (RNNs) have made breakthroughs in sequence-to-sequence learning, such as world leading machine-translation [3]. In our work, we demonstrate that we are able approximate RT and RRTMG functions using a CNN with 12 trainable hidden layers with 6284 neurons. Our CNN is able to predict the RT with less than 0.004 MSE while achieving 10x speedup comparing to the original calculation.

Our contributions are the following.

- We model the calculation of radiative transfers in climate models as a prediction problem in machine learning.
- We employ state-of-the-art CNN techniques to achieve accurate predictions of broadband RT with a 10x speedup to state-of-the-art methods.
- We shed light on the possibility of transforming mathematical approximation problems into function approximation with deep neural networks.

II. Neural Network for Predicting RT

In climate models, the earth is modeled in three dimensions: longitude, latitude, and height. RT is calculated in 1-dimensional atmospheric columns at each longitude and latitude grid point. Fixing a pair of longitude and latitude, RTs can be represented by a vector, whose length is the number of vertical levels into which the column is discretized. The calculation of RT is based on a number of inputs, including a vector of atmosphere pressures at each level, a vector of air temperature at each level, a vector of humidity at each level, a scalar of surface temperature, and a scalar of a specific carbon dioxide mixing ratio.

Preparing the inputs: the mixture of vector and scalar input parameters is a challenging feature extraction problem. Using domain knowledge, we decided to reshape the input to an NxN matrix filling the empty spaces with 0s. The granularity of the RT calculation is determined by the number of atmospheric levels that we model. We decided to train it with 96 levels, a relatively high level of resolution.

High level of resolution: we trained our NN using input data with a high level of resolution from the ERA Interim [4] historical dataset. The dataset provides historical data of the previously mentioned

input
conv1-64
conv3-64
conv3-128
conv3-128
conv3-256
conv3-256
maxpool
conv3-512
conv3-512
maxpool
conv3-512
conv3-512
maxpool
fc-1024
fc-256
out-96

TABLE I
RadNet: net used for predicting RTs. Convs have stride=1 and padding=1. Maxpool use 2x2 filters with stride=2.

input parameters from 1979. There are around 3.3 billion records. However, the records in the dataset use a 16-level separation of the atmosphere. In order to extend the historical dataset to 96 levels, we insert 5 data points (levels) evenly in between 2 historical data points (levels) based on the following rules.

1) A random number of these 5 data points are generated using a Gaussian process, whose mean is the mean of the 2 historical data points and standard deviation is the statistical standard deviation of the entire dataset with respect to a specific latitude, longitude, level and day.
2) After step 1, the input could be 16 times α levels, where α can be an integer from 1 to 6. Then, the input is interpolated to 96 levels.

This data augmentation strategy covers both spiky and smooth cases of the 96 level inputs. After the inputs are prepared, longwave RT is calculated using a Python based interface to RRTMG [5], which serve as the labels of the training samples.

Designing the neural network: Our neural network is a CNN implementation based on the VGG Net [6]. We have made significant modifications and additions, including the use of 1x1 convolutions for cheaply increasing the depth of the net, Parametric Rectified Linear Units (PReLUs) that resolved a vanishing gradient during training, batch normalization, and the Huber loss function, which provides robustness to the net. Table I illustrates the structure of the neural network. The neural net has been trained using 19.1 million samples, a total of 105GB. The training lasted 5 days doing 4.5 million steps (∼0.1 sec/step). The number of parameters of the net is ∼4.8 million.

III. Evaluation

We have evaluated our NN against the RRTMG algorithm. The tests have been performed using a machine with a GPU Nvidia GTX 1080 8GB and a Intel Xeon CPU E5-2620 v3 @ 2.40GHz. The prediction speed using our trained NN model is ∼4.6 ms per sample on the GPU or ∼17 ms per sample on the CPU against ∼30ms using RRTMG.

Figure 1 illustrates the RTs of four samples. The blue lines represent the calculated results using the RRTMG algorithm, and the red lines are the predicted results using our NN. The two plots on the top are RTs from spiky input profiles while the two plots on the bottom are RTs from smooth input profiles. All of the plots show the MSE and the accuracy, which is calculated using $Accuracy = 100 - SMAPE$ [7]. SMAPE is a modified MAPE, in which the divisor is

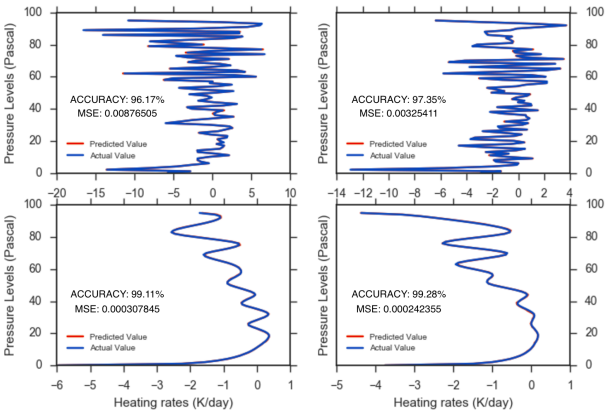


Fig. 1. Radiative Transfer: prediction vs actual value.

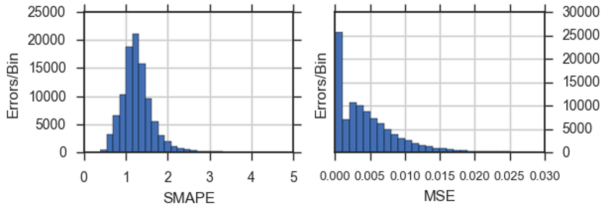


Fig. 2. Histogram of the SMAPE and MSE for 100,000 samples.

half of the sum of the actual and forecast values.

The accuracy and the MSE of the NN model is 98.71% and 0.004 respectively over the entire test dataset, which is not used during training. In figure 2, two histograms of errors are presented.

IV. Conclusions

We show that we can predict radiative transfers using NN efficiently, enabling significant reductions in computation time for high resolution climate simulations. For future work, we will extend our NN for more inputs and improve its prediction accuracy.

References

[1] S. A. Clough, M. W. Shephard, E. J. Mlawer, J. S. Delamere, M. J. Iacono, K. Cady-Pereira, S. Boukabara, and P. D. Brown, "Atmospheric radiative transfer modeling: a summary of the AER codes," Journal of Quantitative Spectroscopy and Radiative Transfer, vol. 91, no. 2, pp. 233–244, Mar. 2005.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.

[3] Y. Wu, M. Schuster, and et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," CoRR, vol. abs/1609.08144, 2016.

[4] D. P. Dee et al., "The ERA-Interim reanalysis: configuration and performance of the data assimilation system," Quarterly Journal of the Royal Meteorological Society, vol. 137, no. 656, pp. 553–597, Apr. 2011.

[5] Joy Merwin Monteiro and Rodrigo Caballero, "The Climate Modelling Toolkit," in Proceedings of the 15th Python in Science Conference, Sebastian Benthall and Scott Rostrup, Eds., 2016, pp. 69 – 74.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[7] S. Makridakis, "Accuracy measures: theoretical and practical concerns," International Journal of Forecasting, vol. 9, no. 4, pp. 527 – 529, 1993.