ISARA-Lyon
23, rue Jean Baldassini
69364 LYON CEDEX 07

Wageningen University
6708 PB Wageningen

Antoine Drouillard
Engineer Student ISARA-Lyon
Promotion 45 (2012- 2017)

Vincent Payet
vpayet@isara.fr

Supervisor: Jos Hageman

Using deep learning for molecular structure prediction from mass spectra*

Antoine Drouillard

August 18, 2017

*This report was written by an engineer student of ISARA-Lyon accordingly to an official agreement with Wageningen University. All citation, diffusion or communication have to refer ISARA-Lyon as one of the main institution at the origin of this document.

Abstract

Deep learning is a new branch of neural network that is subject to an hype from computer scientist. Model creating from deep neural networks showed outstanding results for a model created from scratches that can even overcome human accuracy in some cases. Current researches orientated in image analysis, natural language processing and smart cars are mainly done using deep learning. Besides, deep learning can be applied to a wide range of application such as gene prediction, protein classification, biomedical imaging and other various biologically related field. In this study, the focus was made on the possible efficiency of *Deep Neural Network* (DNN) for analyzing *mass spectra*. The analysis of mass spectra currently is time consuming and requires human analysis and large databases. Furthermore, mass spectra are subjected to high dimensionality. Thus deep learning was used to generated model in this study. However, it was not possible in that case to study the overall molecular structure. So two study cases have been studied: molecular weigh *predictions* and *molecular descriptors* identification. In the end, even if models were not fully optimized, the results showed that deep learning can be relevant in this type of analysis and could perform quite well.

Keywords: Deep learning, Deep Neural Network, mass spectra, molecular descriptors, prediction.

Contents

1 Introduction

Deep learning is a domain of computer sciences belonging to machine learning and can be considered as a new way of using neural networks. It appeared after the first neural network based systems. However the calculation power and data availability was limited and made the training non-viable for large models[18;22]. According to Moore's law and its generalization, the hardware has evolved enough to support heavy loads generated by the calculation needed for deep learning. In addition, deep learning requires extremely large data sets for the training. It was only in the last few years that all the obstacles were overcome with better memory storage, GPU based calculation and Big Data introduction. In 2012, Hinton's lab obtain an error of 15% in an image labeling competition instead of the usual 25% obtain by machine learning application[20]. Since then studies and technologies based on Deep Learning structures are becoming more and more common[28] such as smart car[12], image processing[3;20;21], natural language processing softwares[22;16], bio-medical image processing[11;23;24;25;35;37] , or other biological applications.[29;40]

Furthermore, deep learning based analysis can be a way to deal with large data sets characterized by high-dimensionality. Thus, it can be used as an other way of analyzing complex datasets without reducing dimensionality. Those two properties can easily be found in biological data sets. But deep learning can not be applied to every kind of analysis. In order to successfully generate models based on deep learning, the study case data have to be large, with no empty values. Besides, the output of the models needs to be either a predictions or a classifications. The study of DNA sequences and protein structure predictions are suitable dataset for deep learning. But regarding the time constrain and hardware limitation, it was unrealistic to run analysis on those problematics. After reducing choices, the analysis of mass spectra seemed to be a compatible and logical choice for a application based on deep neural network for a thesis. The data can be stored and used as images or peak values allowing a broader selection of possible methods.

The output of this analysis is called a mass spectrum. It is usually presented as a graph of peaks composed of relative intensity for a given mass-to-charge value (m/z). However, raw outputs are not directly usable partly because of background noises. So a pre-processing step is applied to the raw output form the detector[7].

Besides, mass spectra is an important tool for metabolomics analysis.[10;1]. Mass spectrometry is commonly use to identify organic compounds at trace levels. It provides structural

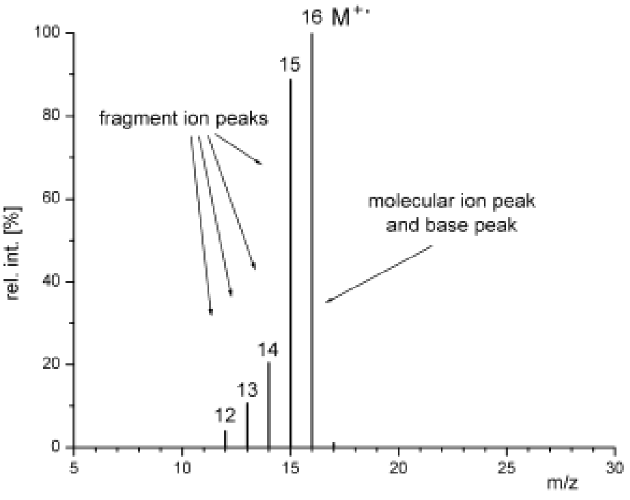


Figure 1: Basic example of mass spectrum[1]

information of the molecule with a high sensitivity[8]. Thus metabolites and other non-organic molecules can be studied and potentially identified with a relatively high accuracy. However, mass spectra analysis is problematic especially for complex molecules such as long chain polymers[1]. High dimensionality and the nature of the peaks are two of the main reasons that cause this complexity.

One possible analysis could be done by using a database search algorithm to compare the output of one molecule against a full database[39;32;36]. The main drawbacks of this technique is that the molecule has to be inside the database to obtain an accurate results. There is also a risk of misclassification if two compound have similar spectra profiles.

A second possibility can be to use an algorithm or trained neural network to perform a classification on the spectrum[8;9;34]. The aim of all those method is to identify the molecule and its structure. In case of application using a trained neural network, the training set and the choice of classifiers highly impact the results. The generalization of those application is usually limited.

Regarding the high dimensionality and availability of data in mass spectra, deep learning based applications could perform even better than the current packages available. Those two characteristics are usually seen as limitations in common machine learning applications, but it is not the case in models based on deep learning if there is enough data available. However, only a very few studied were conducted using deep neural architectures[4;39;40]. In the end none of these studies managed to obtain promising results. However this field of artificial intelligence applied to biological data remains relatively recent and have room for improvements.

Finally, deep learning represent an opportunity to create new tools to analyze biological data. During this project, the main objective was to learn how to use deep learning by analyzing mass spectra. However, due to various limitations presented later, it was impossible to recreate or follow the systems implemented in the previous studies[4;39;40]. So a different logic was applied. Instead of trying to classify and predict accurately the molecule and its structure, the objective became to finding the molecular weight and molecular descriptors. Molecular descriptors can be contained inside a structure called fingerprints or bitstring. Those descriptors gives partial information about the nature and the structural aspect of a given molecule. But it losses the informations about the overall structure. The molecular weight is relatively simple to determine with classical analysis or with experience. Thus it is used as a simple test to implement and optimize a first model. Then the molecular descriptors are harder to predict but also contain more useful information of the molecule. A trained model that can predict with an acceptable accuracy is the end goal of this work.

2 Methods

As presented before, deep learning is a part of artificial intelligent (AI) like machine learning. To be more accurate deep learning can also be seen as a special case of deep machine learning process. Deep learning models can possess hundred of thousands parameters instead of hundred of them in usual machine learning models. Besides, setting values for every single parameters requires more data every time that th number of parameters grows. That is the reason why deep learning require more data and calculation power than optimized machine learning models.

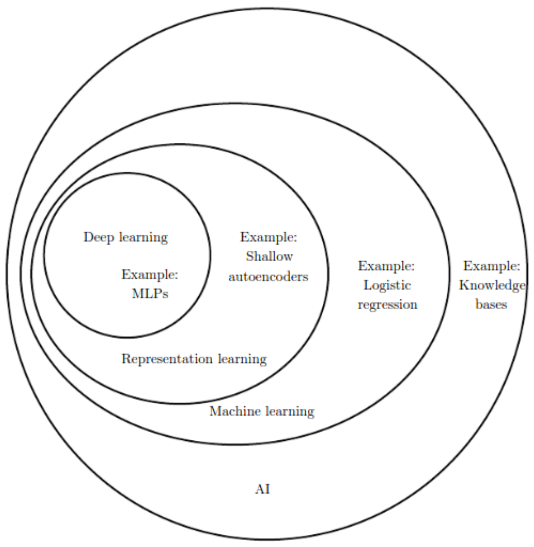


Figure 2: Quick representation of AI domains[13]

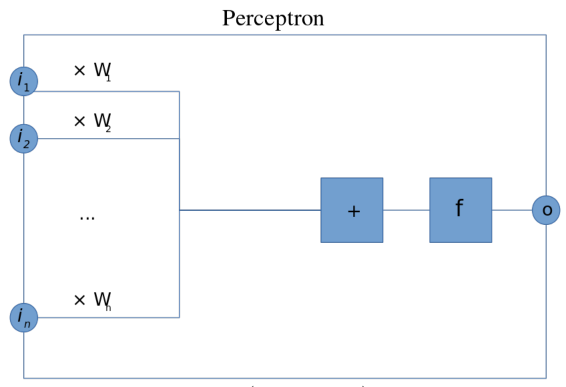
Currently deep learning could be seen as a miracle solution to a lot of modeling problems. For some example deep learning applications have showed outstanding results when applied to smart cars, image or natural language processing. However it is not always suitable to every type of analysis and few theoretical concepts have to be understood in order to implement a deep learning based analysis.

In this section some theoretical aspects about deep learning are presented. This is the only basic requirements needed to fully reproduce the simulations and analyze obtained results. In addition, the information provided in this section are the basic requirements to successfully implement, train and validate a model generated by a deep neural network.

2.1 Neural Network

A neural networks is a conceptual construction made of neurons. It was inspired by the cellular brain structure. The idea was to use artificial neurons to create an artificial structure mimicking a brain. This structure was first introduce in the 1940s[38] and at that time they expected to create an advanced AI capable of having a conversation with humans in a few decades. However this objective is far from being accomplished. AI progress has been less important than every computer scientist could have expected. In the last several years, few hypothesis have been emitted on potential improvement possibility for the AI. The first one is to is to stop using structure inspired for brain and create a new structural network optimized for computers. However, even if in theory this hypothesis is valid[28], there is currently no study published that shows a new system implementing a network using this principle. The second one is to develop neural networks even further by making them even larger and deeper. These deeper models have became black-box models after becoming too complex.

In a biological organism that has a developed central nervous system, the brain is the central part of a nervous system. A brain is built from the association of many neurones connected. From this concept, neurones were adapted to an artificial state called perceptron. The structure is simple at the smallest scale such as shown in figure 3. The perceptron is composed of 3 parts. First it receives values as input. Then these inputs are summed and a bias is applied. Finally an activation function f is used to produce the output. Furthermore, a perceptron can have many inputs and produce only one output value.



$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

Figure 3: Structure of a Perceptron, `https://en.wikipedia.org/wiki/Perceptron`

The activation function is a crucial point in a perceptron. This function determines and formats the output. There are a lot of different options and the activation function was chosen according to the data and expected results. In addition only a few are commonly used as activation function such as relu or sigmoid. For a practical example, the package *MXNET* presented later in this report allows the use of:

- **Rectified linear unit:** Also called relu, this function became one of the most commonly known activation function in deep learning. It is used in various *Convolutional Neural Networks* models in order to analyze famous datasets such as the *MNIST* dataset[21] . This function exist in a few forms but the basic formula is $f(x) = max(x, 0)$
- **SoftPlus:** Also called soft rectified linear unit, has been used in neural networks for non-normalized data containing exponential values. This function is more suitable for a dataset with broad range input data. The formula is $f(x) = log(1 + exp(x))$
- **Logistic sigmoid:** This function allows the introduction of the non-linearity to activate a neuron. Also this function is often used because it is fast to compute compared to other non-linear activation function[17]. The formula is $f(x) = \frac{1}{1+exp(-x)}$
- **Hyperbolic tangent:** This function is also used in image processing as relu. A model was develop to analysze the *MNIST* and shows outstanding result and obtain an error of 0.35% in 2010[6]. The formula is $f(x) = \frac{exp(x)-exp(-x)}{exp(x)+exp(-x)}$

Even if a neuron can be used alone, the main interest in this technology is found when neurons are connected together. All neurons connected together creates a network called artificial neural network. This network can work various analysis by going through a training based on a labeled dataset.

The figure 2.1 shows a possible architecture for an artificial neural network. This network is composed of three distinct blocks. Each block have their neurons fully connected to the other. The terminology fully connected is important in this type of network. That means that every neuron in one layer possesses a weighted connection to every single neurons in the following layer. The first block is called the input layer, second is a hidden layer and finally the output layer. Those units are commonly found in sophisticated artificial neural networks. In addition, a network with this structure needs to have one hidden layer to be classified as machine learning structure.

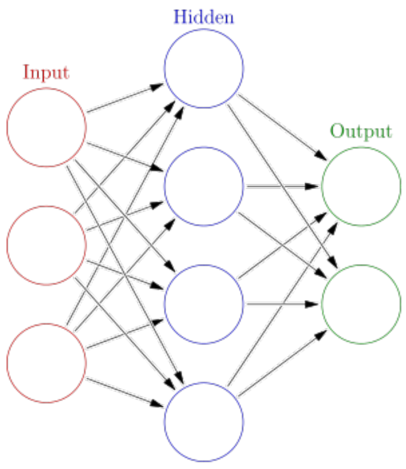


Figure 4: Scheme of an artificial neural network, `https://en.wikipedia.org/wiki/Artificial_neural_network`

Artificial neural networks can be used to find patterns inside a dataset using a training set. However, in order to do so, some assumptions and requirements have to be fulfilled. First the number of hidden layer have to be limited to one in order to be called machine learning application. Second, the training set should be representative of the population in order to generalize the application of the model produced. Third, cross-validation test has to be performed on the generated models because over-fitting occurs easily during the training. Finally, a test set have to be identify and isolated with the same proprieties as the train set and never be used for the training.

Finally, the definition and uses of machine learning applications have evolved drastically since their introduction. Furthermore, the apparition of efficient deep learning applications had changed artificial neural network even more than any techniques created before. Artificial neural networks were previously limited to one hidden layer because of technical limitations. So optimization processes have been developed and improved the accuracy of a single layer model a lot. However,models based on deep learning use multiples hidden layers thanks to better computer and large datasets that allow acceptable training time and over-fitting[13;22].

2.2 Deep Learning overview

As presented previously, deep learning is a cluster of machine learning. One simple way to approach deep learning for the first time is to consider it as a deeper neural network using more layers. Models based on deep leaning can stack millions of parameters instead of hundreds possible using machine learning networks.

However this definition is too restrictive to define deep learning systems. Like machine learning, deep learning applications can be implemented in different ways.[26] So other comparisons are needed in order to tackle have a broad overview on deep learning. On one hand, machine learning requires to have a hand-design model. Then the training is used to balance wights and values. On the other hand, the programmer only chooses and codes an architecture for deep learning. Then the model finds patterns and features inside the given dataset. Figure 5 summarizes the situation about how every structures works. Only two branch in representation learning can perform prediction for scratch with no pre-setting parameters. The last chart on the right side represents a more advanced and most likely complex architecture and deep learning split features into block to be able to find out main characteristics first and adapt even more with the additional features generated.

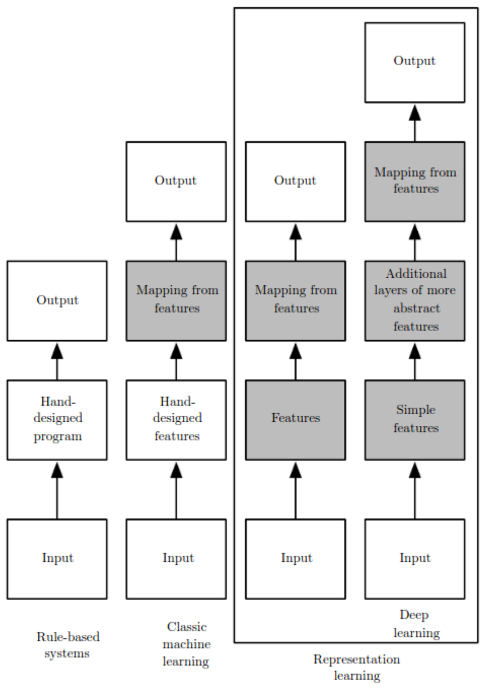


Figure 5: Schematic representation of AI domains[13]

To summarize this quick introduction so far, deep learning is a recent branch of machine learning. In addition, regarding recent surprising success of deep learning applications against machine learning models, deep learning is currently growing in importance and variability. Finally, to conclude the comparison between those levels of artificial neural networks, deep learning is about choosing an architecture and optimizing parameters in order to allow the network to find features of a dataset itself. Thus, deep learning networks have the ability of creating models from scratch without any human intervention. For example humans identify number by looking t the shape, but a computer uses the pixel values and uses more comparison points or features.

Computer vision programs using a model generated by deep learning can end up by being more accurate and faster that humans.

2.2.1 Principle

Deep artificial neural networks are able to determine features and patters from scratch. In order to obtain this ability, the network requires a specific structure and various parameters.

2.2.1.1 Hyper-parameters

Hyper-parameters are elements that code for the overall topology and specifications of a network.[13]. In this part the main hyper-parameters are presented. Only several hyper-parameters are common in every deep learning networks. The other are used to optimized specific structures that determines the nature of the network.

First, hyper-parameters coding for the overall structure of the network are required and extremely important. The number of layer is the first aspect to take into consideration. Since deep artificial neural network can accept more than one layer, the number of layers used have a central part in a model[30] . Adding more layers inside a network can be interpreted as allowing the model to grasp more lower features and adapt more to a dataset. In addition, a number of neurons per layer need to be assigned. Adding more neurons per layer make the network more flexible to recognize patterns. The figure 6 shows the effect of adding more parameter, supported per neuron, on a model accuracy.

However, adding more layers and neurons shows limitations that have to be taken into account. Having more neurons and layers increases the time and space required to train a model. Furthermore, it also increases the risks of over-fitting . Thus, adding more layers and neurons to a network can allow better predictions but only if the over-fitting is controlled and the dataset big enough to make it possible.

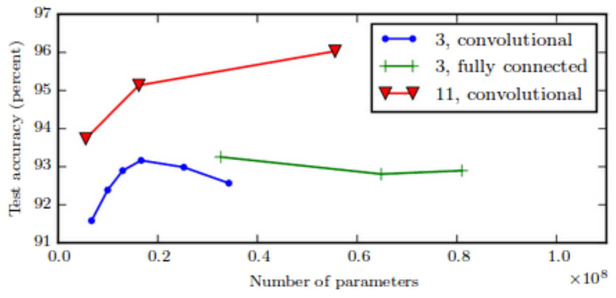


Figure 6: Graph showing effects on increasing parameters on a model[13]

Then an activation function has to be applied on neurons. Several choices can be made from that point. The first one is to apply an activation function per neurons. from a theoretical point of view, this choice is the best one to have a more accurate result. However, selecting and implement one function per neurons is in practice time consuming and make the optimization close to impossible. So the most logical choice is to set one activation function per layer. In that case the ratio between optimization complexity and accuracy can be balanced (MIT online course `http://selfdrivingcars.mit.edu/`). An hidden layer that has an activation function applied can called activated hidden layer.

Finally, one other essential hyper-parameter related to topology and size is called batch size. Training a deep artificial neural network refers to the phase when a network is initialized. Then the train set is used in order to set the weights of edges and neurons using many iterations. Besides, the matrix used to train a network is large, often too large to load everything into RAM. So only pieces of this dataset are loaded and use for the training. Batch size is the length of this chunk of dataset. After one batch is used a new one is loaded. In the end, after all entries are used, another iteration begins with new batches. In deep learning, the epoch is a parameter that refers to an iteration done by using all batch generated from the train matrix.[28]

The batch size importance is more about its influence on the memory requirement than the effect on the results. Reducing the batch size allows programmers to overcome memory limitations. Besides, using a small batch size compared to the size of the dataset is called mini-batch size. Mini-batch size increases training time and a study shows that bigger batch-size are more accurate.[19] Thus, batch size has to be optimized in order to make the training time of various models fit into the project limitations.

2.2.1.2 Training parameters

After the topology, the second important aspect about deep learning is the training. Training the network is the phase that influences the final result and accuracy of different models. On one hand, an topology can be used for different analysis. For example MNIST and iris dataset analysis can be done using the same nuber of layers and neurons. But on the other hand a training is unique to every application and objective. And that is why every models ends up being unique.

The epoch, as presented before, corresponds to iterations. This parameter is easy to optimize using an empirical method. The best number of epoch can be found by looking at a learning curve. Figure 7 is an example of one training curve that has been generated during this work. The index are the number of epochs. The objective in that training is to minimize the root mean square error. Regarding the curve, the number of epochs needed is between 200 and 250 epochs to have a stabilization of the error. Finding the number of epochs can be tricky is there is a temporary stabilization in the training. So it is safer to start with a large number of epochs and reduce afterward to a reasonable number that allow a complete training.

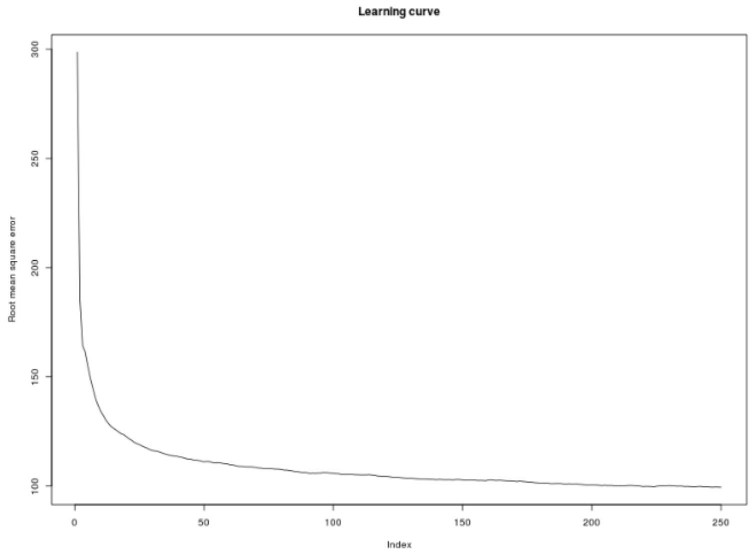


Figure 7: Training curve using root mean square error as criterion to minimize

Besides, a method has to be applied in order to set and optimized weights inside neurons. In our case, all models have been trained by using the stochastic gradient descent (SGD) algorithm (see figure 25 in appendix) coupled to a mini-batch sized. The SGD is one of the most commonly method used to adjusts the values inside a neural network.[5;38]. At the appearance of deep learning SGD was replaced by Hessian-Free optimization[33] because SGB requires the use of several parameters to work in deep learning networks.

To be more accurate two parameters are crucial when SGB is used in deep learning: momentum and initialization. The momentum is a parameter that is implemented directly inside the SGD algorithm (appendix figure 26). The momentum is named from the variable used in physics. It is used as a corrective value during the calculation a the new weight value at every neuron.

The initializer is a parameter that determines the possible values randomly assigned to each neurons before the training phase. The fact that a random initialization and batch compositions are applied make a deep learning network randomized. Even with the same parameters and data, every model produced is unique and it is nearly impossible to obtain an identical result. But even is the combination values of every parameters inside a deep learning network is unique, the results and function of every layers remain identical and allow a potential reproducibility.

Then the learning rate is also a crucial point to optimize. This parameter is also used in common machine learning application.[17] Learning rate correspond to the step used in balancing the weight at every neurons. On one hand, choosing a small learning rate could improve the accuracy because the weight has more chance to get close to an optimum, on the other hand, taking a bigger learning rate reduces the training time. Usually the learning rate should be small enough to be able to train a network efficiently and big enough to have acceptable training time.

2.2.2 Architecture

Network structure has been introduced earlier. There is a lot of different architectures nowadays and new ones are continuously created. Only three main architectures are presented in this section: Deep Neural Network (DNN), Convolutional Neural Networks (CNN) and Recurrent Neural Network (RNN). Only those three architectures are developed in this document because other newly created structure often come from those three.

2.2.2.1 Deep Neural Network or Deep Feed-forward Network (DNN)

Deep neural network is the easiest structure to understand because it is composed of several fully-connected layers. Each neuron from layer n is linked to all neurons from the next layer $n+1$. The network is represented in figure 8 below. This architecture is called multi-hidden layer artificial neural network. In addition, this network is also called feed-forward neural network. That is because batch are inserted from the input and get processed to the output. Then second phase, called back-propagation, uses a methods to adapt the weights inside neurons to optimize the network accordingly to the results. The model is trained using those two phases at every epoch or batch regarding the structure chosen for this parameter.

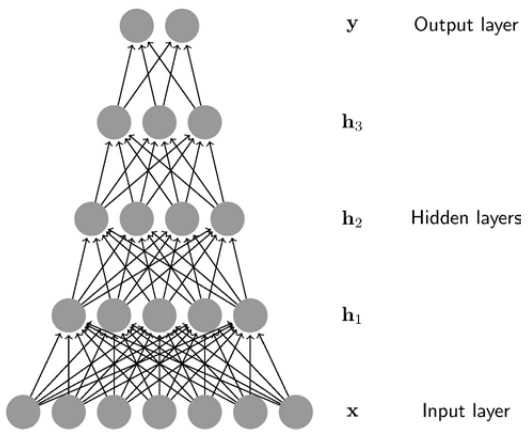


Figure 8: Deep Neural Network representation[13]

The result of the model will be generated on the last layer called the output layer. This layer takes values from the previous layer $n-1$ and uses a method to generate a chosen output. One example of possible output can be a classification method. The softmax classifier showed in figure 9 is commonly used in image classification. It can return probabilities for n classes, when it is coupled with a threshold it can return a set of most probable class[4;12;22]. A logistic regression classification can be used in binary prediction instead of a softmax. So changing the method applied on the output layer of a network determines its results.

$$\text{softmax}(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

Figure 9: Softmax function, from MXNET documentation (`http://mxnet.io/api/python/symbol.html`)

DNN is the easiest deep learning architecture to understand since it used only fully-connected layers. In addition, this architecture can process various analysis from classification to continuous variable prediction with good accuracy without requiring preprocessing. However, this structure is not the most efficient regarding memory complexity because data are processed in a way close to a brute force algorithm[13]. Details of pseudo-code for implementing a DNN is describe ***Deep Learning*** book[13].

2.2.2.2 Convolutional Neural Network (CNN)

Convolutional neural networks could be considered as a filtering methods used before a DNN structure. DNN takes the inputs and do not do any operation to reduce dimensionality. For small images for example it is not an issue. But for example an image in shade of gray of a size of 32x32 pixels is used, 1,024 input neurons are needed. This number become even larger with RGB information and become 3,072 input neuron. So in order to analyze real life sized images faster and with a lower memory complexity CNN where created. Lex Fridman describes convulotional layer in his course (`http://selfdrivingcars.mit.edu/`) as "cheating preprocessing" in oder to reduce dimensionality of the input.

Convolutional layers can be considered as filter used in order to find main features inside the input data. Many filters can be applied in one layer. Then a pooling step is added in order to reduce the data even more according to the features previously identified. A more detailed example can be found in the appendix 27 and a possible structure in figure 10.

Finally, convolutional layers characterize CNN structures. However, comvolutional layer are often coupled to other structures such as DNN in order to predict classes[21;22] or a recurrent

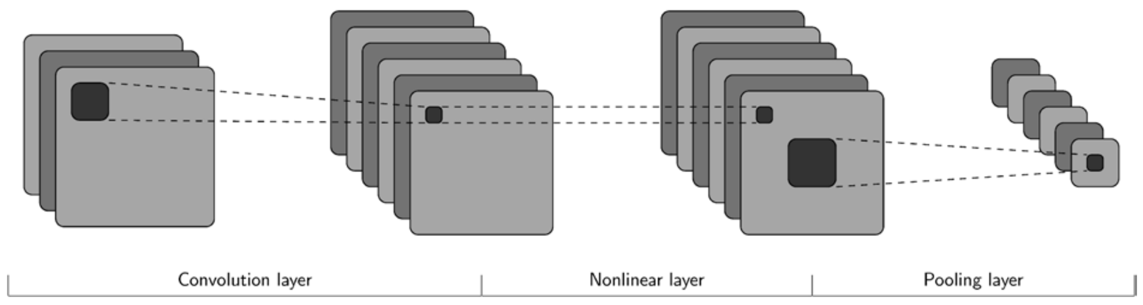


Figure 10: Convolutional Neural Network representation[13]

neural network to produce a descriptive sentence about an image.[26]

2.2.2.3 Recurrent neural network (RNN)

The last architecture introduced in this paper is called recurrent neural network. This architecture shows outstanding results when it analyses sequential data such as natural language processing and even DNA sequences.[13;25]. The basic idea is that at some nodes a loop is applied and the values is calculated many times at this node before going through the network again. The value in the looping node is computed everytime using the same weight, bias and activation function. A simplified representation in showed in figure 11 and a more developed on in appendix 28.

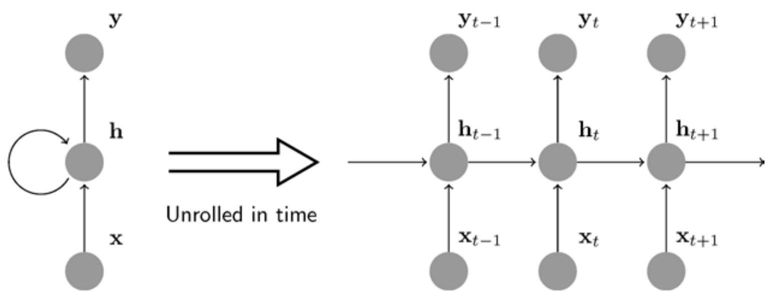


Figure 11: Recurrent Neural Network representation[13]

If the network is represented unfold it can be one of the deeper architecture used. Furthermore, using this architecture is more difficult to implement and use, In addition, it requires even larger dataset that the previous architectures. But finding labeled sequenced data is easier than labeled images. Because languages can be used either as a values or labels (depend on which sentence to translate from one language to another).

Finally, RNN is the only architecture that allows a different number of output. Thus, RNN is also the only structure that allow the flexibility and adaptability to change the topology of the output layer. In application, a translation from one languages to another can have a different number of words and order of words.

2.3 Over-fitting and validation

One of the biggest challenges in deep learning is to check the accuracy and to validate the model. Deep architecture of neural networks gives it the ability of finding patterns and features inside a dataset. However, at least thousands of parameters are used in order to do so. This systems create two main issues.

The first drawback of deep neural networks is caused by over-fitting. Machine learning models tend to over-fit easily, this trend is even more important in deep learning application.[13] The use thousands or millions of parameters greatly increases the opportunity for over-fitting to occur. Two solutions are used to limit the overall over-fitting.

The first one is to use a large databases of hundred of thousands of units to have a number of parameters lower than the number of units in the dataset. Unfortunately, models are continuously growing deeper and the amount of labeled data needed also grows.

Then the second system is called **dropout**. The dropout value is between 0 and 1. It represents the percentage of neurons that had been randomly inactive inside a model at every batch size[31]. On one hand, this system allow the network to fully use all its neurons in the end of the training. On the other hand, non-used neurons in a network is preventing it to adapt weights of extreme values. Thus, the model is able to perform the prediction using all neurons but the inactivation during a training reduces the over-fitting.[31]

The second drawback is that the model is considered as a black-box one because of the overall complexity of the trained network[28]. Currently the only way to check if a model is correct is to measure the accuracy by calculating a percentage of error or calculating an error such as *Root Mean Square Error (RMSE)* or *Root Mean Squared Logarithmic Error (RMSLE).*

$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ $RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(x_i) - \log(y_i))^2}$

Then the over-fitting and generalization of the model should be checked. So a cross-validation validation is often needed. In addition other tests can be done. In this work, only two of them had been used: a Q^2 test and a AUROC test.

A Q^2 test test, $Q^2 = 1 - \frac{\sum(y_i - \hat{y}_i)}{\sum(y_i - \bar{y}_i)}$ [14],can be performed in order to validate a model that predict a continuous value. For example a model that uses a SquaredLoss (figure 12) method in the output layer can be used for this type of prediction. A Q^2 value is in an interval $]-\infty;1]$ and during the optimization this score is maximized. However, in case of over-fitting, the difference between a ${Q^2}_{train}$ and a ${Q^2}_{test}$ is increasing. So in practical use, the programmer has to optimize a Q^2 score while keeping the balance $\frac{{Q^2}_{test}}{{Q^2}_{train}}$ as close as possible of 1.

$$\text{SquaredLoss}(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} \left(y_i - \hat{y}_i\right)^2$$

Figure 12: Square Loss formula, MXNET documentation (`http://mxnet.io/api/python/symbol.html`)

A second class of output possible for deep learning is a classification probability. In that case it could be a multi-class classification or a binary classification. A logistic regression binary classification layer can be used in order to perform binary classification. In that case a Q^2 is not useful. An Area under Receiver Operating Characteristic or AUROC is more suitable in that case. This test compares a *true positive rate* against a *false positive rate* at different threshold (ROC curve, appendix . In neural network the *true positive rate* can also be called sensitivity[15]. Finally, during the learning phase the AUROC value is maximize to a maximum of 1 for a perfect model. In addition, the difference of results between the train set and test set should be minimized.

2.4 MACCS bitstring and molecular weight

A molecule can be defined by many descriptors. It could be defined by a structure, a name or a formula. Even a structure can be written in different ways such as using an InChi code or Smiles string. However, using a 2D representation of a 3D structure is complex. So molecular descriptors were created to describe the structures inside a molecule.

Molecular descriptors can be generated in many ways. The Tanimoto index is relatively new and seems to be able to take into account the complexity of a 3D structure efficiently.[2] One other possibility is to use a bitstring. Those descriptors can be easily used in computations. The idea is to find out some molecular characteristics inside a molecule. Then this list of characteristic is created and merged to end up in a string of 0 and 1. Finally, this string is called fingerprint of a molecule.

It exist many fingerprints composition, all describing different combinations of descriptors and different ways to generate them. Packages on R such as *rcdk* can be used in order to generate fingerprints. The default fingerprints length used in rcdk is composed of 1,024 bit. Shorter solutions are also available and describe on the *rcdk* documentation (`https://cran.r-project.org/web/packages/rcdk/rcdk.pdf`)

The shorter bitstring generalizable by the *rcdk* package is the *166 MACCS fingerprints*. The MACCS or Molecular ACCess System fingerprints can have a length form 166 to 332 bits. All details about the MACCS keys descriptions are available and summarize on the following website: `http://www.mayachemtools.org/docs/modules/html/MACCSKeys.html`. See appendix 35 for more detailed information about the 166 MACCS bitstring.

3 Implementation and Results

The project begins with the choice of a suitable package to implement the structure of the neural network on R. Then the dataset was created and stored in order to be used efficiently. In the end, two models have been created and partially optimized. the partial optimization refers to a possibility that the model corresponds to a local optimum value. However, it was impossible to check deeper structures due to data and memory limitations. All those steps are presented bellow.

3.1 Packages

Choosing a package to implement a deep learning analysis is an important step. Packages often allow a limited choice of architecture and parameter twinkling.[27] TensorFlow had became the clear leader package on python (`https://www.tensorflow.org/`). This tool is developed and supported by Google. It possesses a graphic interface called TensorBoard that allow the user to check every parameter of a network in real time during training. The package selection was based on a comparison between available ones in R that are close enough to TensorFlow. In addition, the integration of a link to perform calculations on a *GPU* is becoming a basic requirement for deep learning packages currently.

Currently there are two main packages implementing deep learning in R: **MXNet** and **h2o**. A benchmarking comparison between the two packages has been done in order to select one. This comparison was realized on the famous **MNIST** dataset.[23]

The overall accuracy and training times were evaluate for the models produced. The advantages of doing this test was to find out how a model can be generated with those packages. Furthermore, there is an exhaustive documentation and code available on the *MNIST* dataset online.

LAYERS	Two topologies used: two or three hidden layers. In both cases the model performs a Softmax classification with 10 classes. Using a *relu* activation function at each fully connected layer
NEURONS	For the case of two hidden layers models, neurons were assigned either at a 500-300 neurons or 750-450. With three layers a third length coding for the new layer is added. The final compositions were: 500-300-150 and 750-450-300
EPOCHS	Packages perform at different speed and the SGD algorithm implementation creates a difference in epochs needed to complete a training. The epoch values tested are: 24, 50, 100

Figure 13 shows scores on the training speed of the two packages. The testing shows that the accuracy will increase rapidly and after 500 seconds the curve stabilize. From the results of this test, the two packages are similar and score nearly the same.

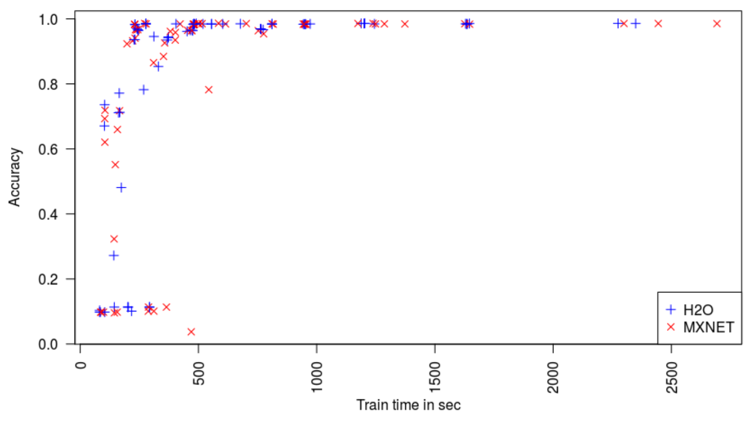


Figure 13: Training time results

Thus, it was impossible to choose one packages only on the training time. So other parameters were observed such as the overall accuracy and the gain in accuracy per second. Figure 14 is an overview of this second comparison. The results were clear, MXNET is slower than H2O. However in the overall results MXNET is more repeatable than H2O. The package H2O seems to have failures sometimes and requires to be restarted. This lack of stability from H2O is a severe limitation. Finally MXNET has a GPU link and GPU calculation functionality already implemented.

Regarding the scoring results and the GPU option, MXNET is the clear leader for DNN implementation in R. It is important to note that a multi-classifier model was produced to compare packages. However, those results could be different on a binary classifier or prediction of continuous trends.

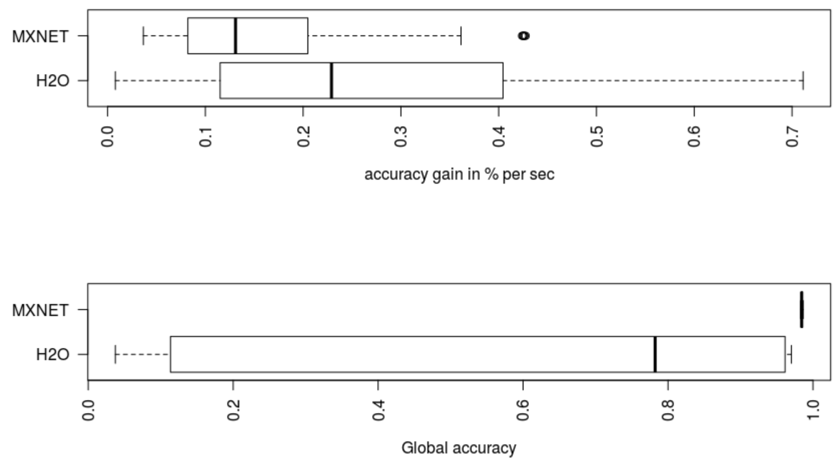


Figure 14: Accuracy results

3.2 Data

The dataset is taken from MassBank (`http://www.massbank.jp/`). It consists of 45,000 different molecules with a *Creative Commons* license. Thus, mass spectra from MassBank are freely usable and publishable. Another advantages of MassBank is that all mass spectra are stored a unique text file following a fixed layout. An example can be found at the appendix 31. So parsing the data and selected necessary information is relatively easy to do.

The parsing script was written in python. Parsing and handling files is relatively easy and quick in python. Parsed entries were stored into a SQL database. This database is composed of two tables.

The first table is called *molecule*. Information about molecules are stored inside. The bitstring wasn't inside the MassBank informations and were generated. The generation followed the process showed by the appendix 35 by using the R package *rcdk*.

The second table stores all available peak information for every molecule. Every peak corresponds to one and only one spectra. Using SQL statements to obtain a list of peaks for a given molecule is relatively easy since the link between the two tables is direct and based on the unique access number.

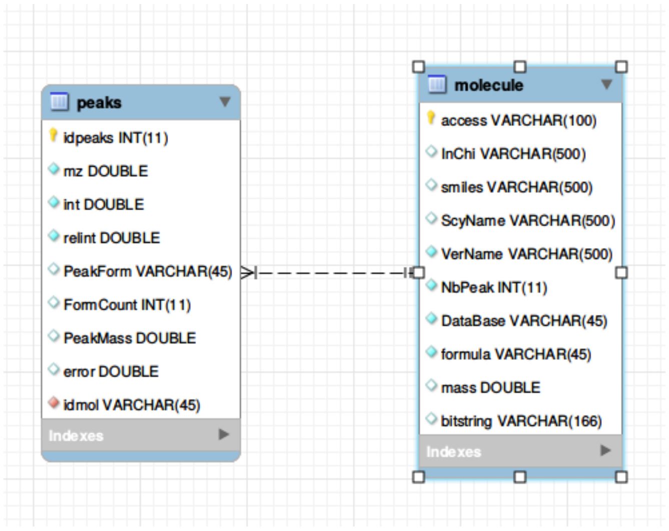


Figure 15: Mass spectra database model

Using a MySQL database with a large dataset is useful for several reasons. First, SQL statements can be used to check and to visualize or to handle data. Second, the package *RMySQL* makes the interaction with a MySQL database easy to use and all SQL statements can be called via the package. Finally, the database can be exported without compatibility issues. In addition, the access via Internet makes this database a safe back up source for datasets easily accessible.

Making this database took a long time. Using a mechanical hard drive ended up to be a a bottleneck during writing and mad the addition of new data slow. However, the compatibility and stability of data has been a tremendous asset during the export of all code and data from a regular computer to a computer equipped with a GPU.

3.3 Molecular weight prediction

The molecular weight is a value that can be found with good accuracy directly from a mass spectrum. Predicting molecular weight is a good test case for starting to gain some experience with MXNET package and optimization process.

3.3.1 Method

The first step was to create the different matrices. The test set was obtained by randomly selecting 10% of the *access code*, primary key for the molecules, in order to build a list of keys (*access list*). Then all peaks in the database were merged per molecules into a matrix. After a matrix is imported in the workspace and load in the RAM, it is split in two sets based on the access list. All rows with an access inside the list are merged to build a test set and the remaining one to build the train set.

Since the molecular weight study is used to gain experience before starting more complex analysis, a suitable number of digits for the *m/z values* has to be chosen. A binning is made by summing every peak intensity inside a given interval. Five matrices were generated, each one of them corresponding to a peculiar bin size. Five different *m/z* steps values were used: $[5, 2, 1, 0.5, 0.1]$. After the creation of the different matrices,it can be noticed that smaller bin sizes lead to larger matrices.

After testing trials, molecules with m/z values over 1,000 weren't predicted accurately by all models. After a deep observation of the database distribution, around 85% of molecules have a maximum peak value under 300. In the end a second generation of matrices was created using only molecules with *m/z* value smaller than 300.

Table 1: Dimensions of the matrices with different batch sizes

Bin size	5	2	1	0.5	0.1
Number of columns	60	150	300	600	3,000

The topology used for to predicting molecular weight was a DNN with fully connected layer. The output layer was set as a continuous prediction output using a squared loss function. The package put it inside the label *Linear Regression Output.* According to the nature of the output, a RMSE was used as learning criterion. Finally two Q^2 values are calculated for training and test sets.

The optimization process was split in 4 phases. The first phase consisted of using different activation functions and combinations in order to set this hyper-parameter. To do so models with two and three layers were implemented. The topology was identical to the one used to perform the benchmarking between R packages. Except that a new layer of one neuron, without activation was used to sum all weights form the previous activated layer $n - 1$.

After testing the relu activation function, the RMSE never change throughout the training . One hypothesis could be that the function sets nearly all neurons to a value close to 0 and losing predictive abilities. Out of the remaining three functions, only the logistic sigmoid allowed the creation of running model. According to those observations, the logistic sigmoid is the activation function used for every hidden layer of the network except the last one.

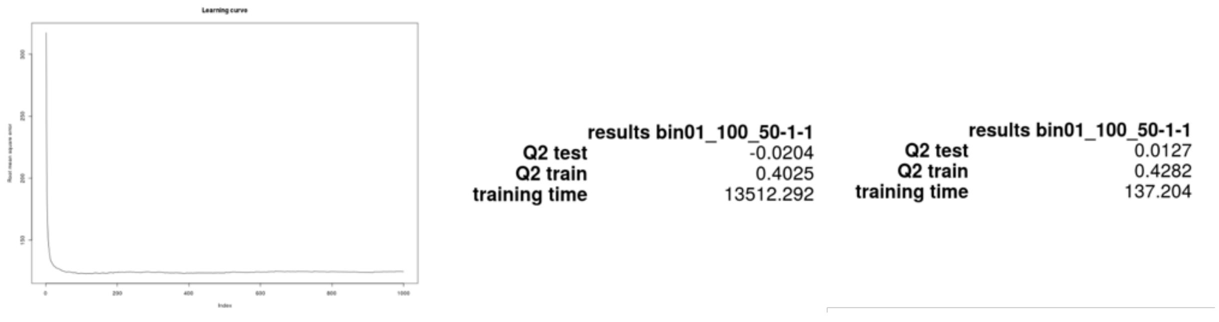
The second and third phase are for the optimization. It was impossible to optimize all parameters at once by using an experimental random design and stay on schedule. Eventually, the parameters optimization had been split in two parts.

The second phase was used to set the parameters related to the learning process. A random trial and error design was used to set all possible combinations between: learning rate, momentum, initializer. Here again the same previous topologies were used in order to set other parameter to equal values. In the end an initializer of 0.07 coupled with a momentum of 0.9 were the most effective choices. The learning rate was set as it's maximum value of $5.5e^{-6}$ to allow a smooth learning. But it is important to note the these values were only local optima obtain out of 4 seeds $\in [0, 20, 50, 100]$. To do a meaningful comparison, more models and repetitions were needed.
The hyper-parameters coding for the topology were also optimized by using a complete random design to locate an optima with those parameters. All other values for the remaining variables were taken from the previous optimization phase.

Layers	Three possible dimensions, every model needed one hidden layer with one neuron without activation function to use a regression output layer. All following counts of hidden layers do not take into account this last hidden layer. The possible values for the number of layer was 1, 2 or 3 hidden layers plus the last layer.
Neurons	The number of neurons is a bit more complicated to handle. The idea was to create a topology with a pyramidal structure. So every layer h_{n+1} had to have less neuron than the layer h_n. The number of neurons tested were $[20, 40, 100, 300, 500]$
seeds	4 seeds $\in [0, 10, 50, 100]$ are used to replicate runs to avoid unlucky runs.
Epoch	After running 30 random models using all possible combinations of all previous values using an epoch of 1,000, results were computed and showed that using an epoch of 250 is more than enough to have a stable model.

Finally, after running models using all combinations of parameters, the possible values are centered around a local optima in order to obtain a partially optimized model. In addition, the epoch value is changed to produced train models as fast as possible.

However, the system using CPU based calculations never reach this phase. The running time is relatively long regarding the number of models that had to be produced. The same optimization process was done on a GPU. Figure 16 shows a short overview of the possible decreasing was obtained with GPU based calculations.



results bin01_100_50-1-1	
Q2 test	-0.0204
Q2 train	0.4025
training time	13512.292

results bin01_100_50-1-1	
Q2 test	0.0127
Q2 train	0.4282
training time	137.204

Figure 16: Comparison between simple analysis done with CPU and GPU

This switch to GPU is useful to produce more models in a shorter time. But memory shortage became even more apparent in this case. The previous system possesses 16 GB of RAM, allowing the user to use large matrices and models. On the opposite side, the *GTX 1050 Ti* is a consumer grade graphic card with 4 GB of dedicated RAM. This DDR5 RAM can not be extended, so to decrease the running time by around 10 fold, the bin 0.5 and 0.1 where dropped. In addition, the batch size went down from 600 to 200.

3.3.2 Results

As introduced before, the memory limitations limited the usable bin size to $[1, 2, 5]$ and also limited the number of hidden layer to 3(+1) hidden layers. So the optimization process stopped at those limits. In addition, a strong over-fitting was observed from the beginning. To reduce this issue, a *dropout* was set in place at every layer with a value of 0.5. That ratio means that at every batch, the model is only training half of the possible neurons. The dropout showed sufficient results but all generated models still present over-fit.

3.3.2.1 Bin size comparison

The optimization process has been done for the three possible bin sizes. The first observation was that the best model was exactly the same between the three bin sizes. In addition, the global accuracy ranking between all models is nearly identical in every bin size. So the binning does not influence the topology of a network. Besides, it also shows that the accuracy of the input does not interact with the model structure. However, if the bin size is reduced, the training time would increase.

Figure 17 shows the results from the best model out of the randomized design. This model is composed of 4 hidden layers, in order 500, 300, 50 and 1 neurons for a total of 7,500,000

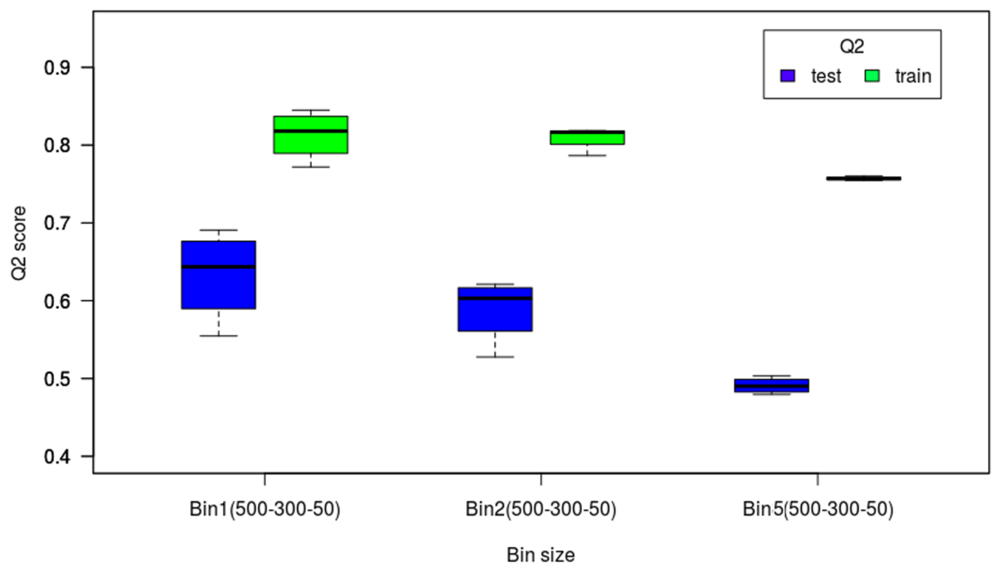


Figure 17: Comparison between Q^2 scores for the 3 bin sizes

connections and 851 weights.

First, as showed by every model, a clear difference is observed between the training and testing scores. This gap in results had already been corrected using a dropout of 0.5. So to limit this difference even more, the data set has to be extended. Second, the scores decrease every time that the bin size is growing bigger. So using bigger steps in binning causes a loss of sensitivity. Finally, reducing the bin size means increasing the number of column of the input matrix. This increase affects the training by making it longer and fill up the dedicated RAM even faster.

In conclusion, using small bins can be used to ease the optimization process. Since using a bigger bin size is independent from the topology and quicker, a rough optimization can be based on a large bin size before reducing it to increase the accuracy. Furthermore, a suitable bin size is a balance between sensitivity and space complexity. Long training times are not problematic to produce optimized models. But if the memory is too limited to store the matrix and training would not be possible. Unfortunately this balance is unique for every dataset and topology and can only be found after a *trial and error process*.

3.3.2.2 Optimization

After observing the effect of the bin size on sensitivity, all further optimization trials were done with a bin size of 1. So the matrix length was of 301 columns by 40,000 rows. Besides, different topologies have been tested in order to identify the best model that fit our minimal requirements. Around 200 models had been produced during this process.

All these models can be used to understand the relationship of the number of layers on the sensitivity. Results of this benchmarking are summarized by the figure 32 in the appendix. However, the simplest models with only one layer perform poorly in comparison with models with multiple hidden layers. Figure 18 shows partial results for easier visualization.

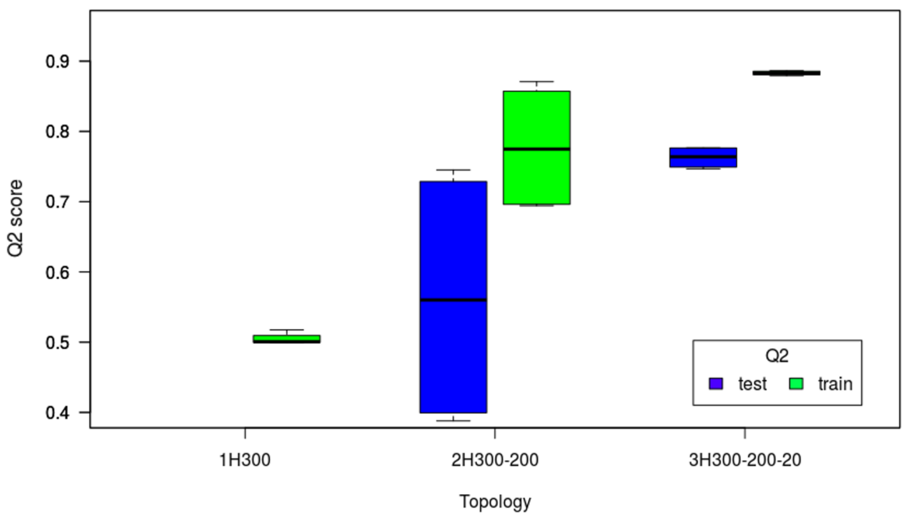


Figure 18: Partial comparison of Q^2 score in molecular weight prediction between the 3 topologies

On one hand the addition of a second active hidden layer allows the model to grasp more complexity of the dataset and greatly increases the sensitivity. On the other hand, models accuracies are contained in a wide interval of possible values. Furthermore, the over-fitting can be either important or acceptable following the random nature of neural networks.

The addition of the third layer allows to set and narrow down greatly the variability intra-models. One hypothesis can be that the addition of the third layer allow models to adapt even more to every batch that can be selected and avoid extreme values during the training. The new layer could summarize the results from previous layers and corrects them to reduce the variation at each batch.

Besides, the training profiles changes with the addition of the third layer. Figure 19 show the two possible training profiles. The left graph represent the training curve observed with one and two active hidden layers. The curve is smooth and decreases directly to a lower value before stabilization. The right graph represents the profile of models with 3 hidden layers. In that case the curve temporary stops for few epochs before the model starts to learn again. My hypothesis is that the correction of weights introduced with the third layer is done during the pause inside the learning. When the correction is done the learning starts again and the overall variations between repetitions can be limited.

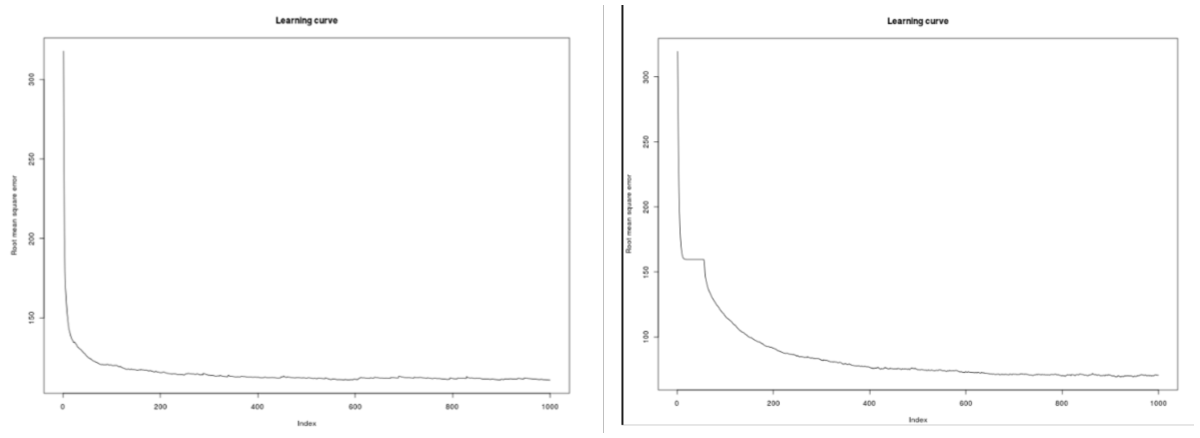


Figure 19: Possible temporary stabilization in the learning process

In conclusion, the final optimized model chosen for the molecular weight possesses 3 active hidden layer with neural composition of $[300, 200, 20]$. This model wasn't the best one in sensitivity, but it was the best one with an acceptable difference between the testing and training scores close to 0.1. The model summary is presented in appendix 34. In addition, this model performs well compared to other methods as showed in figure 20. However results between random forest and the deep learning models are close. The figure 33 in appendix show the $\bar{Q}^2$ scores of MXNET model, random forest and a linear regression. In that case random forest is the best methods. So one optimized deep learning model can outperform random forest, but if repetitions are taken in account random forest is more constant and performs best.

3.4 Bitstring completion

The bin size of one was used in the bit prediction models . This choice was driven by observations made in the first test case. Furthermore, the batch size and the maximum of three hidden layer were also kept. The optimization method used in this set is nearly identical to the one used in the previous set . Besides, the code was simply reused and only the label selection function, for the train dataset, has been modified. However, the results are different and surprising.

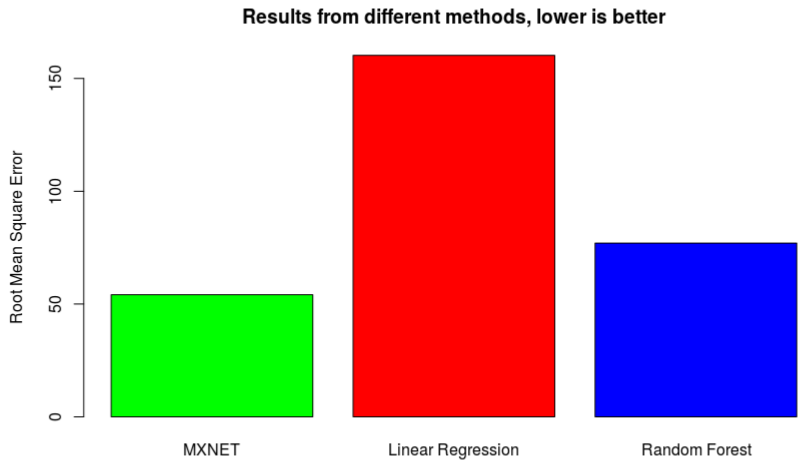


Figure 20: Molecular weight prediction error comparison

3.4.1 Bitstring generation and characterization

This set requires some preprocessing. In fact, the data remains unchanged but a new label system need to be set. Previously the molecular weights were used as label, in this case bits constituted training labels. All molecules inside the database possesses a *Smiles* string. The package R *rcdk* can translate *smiles* strings into fingerprints. The MACCS standard was selected because it was short enough to implement 166 chained models to predict bit by bit the overall bitstring.

The fingerprint generation process is showed in appendix 35. First all access and corresponding Smiles are extracted from the database. Then the package generates bitsring from all valid *Smiles* strings. If *Smiles* strings are not valid the bitstring is replace by "NA". Finally the database is updated. The bitstring values for every molecule is updated with the MACCS string for the corresponding primary key in the database.

In the end 40,000 fingerprints had been generated. But before starting the training, the newly obtained strings have to been characterized to predict the hard ones and maybe detect errors. Three classes of bits are contained inside the database:

Constant bits	Bits at positions $[1, 2, 3, 4, 5, 6, 7, 10, 166]$ are constant inside the dataset. They can not be used by a model since classification requires at least two classes
Complex bits	23 bits have a distribution between 40 to 60% for either 0 or 1. The bits 132 is the closest to a 50% value and had been selected for the optimization. The list of complex bit is $[99, 112, 118, 123, 126, 127, 131, 132, 137, 139, 140, 143, 146, 147, 148, 149, 150, 151, 152, 153, 156, 158, 162]$
Simple bits	All other bits. Those bits have a distribution centered around either 1 or 0

3.4.2 Method

The method about the optimization is exactly the same as the one applied in the molecular weight prediction with a few exceptions. First, this time the objective is to perform a binary classification based on an Logistic regression classification output. In order to characterize the model, a RMSLE (Root Mean Squared Logarithmic Error) is used as learning criterion to reduce as much as possible. Besides the scoring system was based on AUROC method.

Second, after generating a couple of dozen of models, it appears that an hyperbolic tangent is the most suited activation function. Also, larger layers also showed better results, so the value 750 was added in order to study that hypothesis. 750 neurons was the highest value supported by the RAM capacity while still allowing 3 layers. However, the training was done on a CPU with 16 GB of RAM, so way slower than all other topologies. Finally, the initial number of epoch was 500 but it was too small and this value was set at 5,000 epochs. From this observations presented before, the learning parameters were optimized first. This time the learning rate was $5.5e^{-4}$, the momentum was set at 0.1 and the initializer at 0.01.

Then all needed parameters were set and the topology optimization is done by only varying the number of layers and neurons. Note that all models produced with a first hidden layer of 750 neurons had to be done on CPU. The results of the first benchmarking are summarized in figure 38.

At one layer, 750 neurons is the best choice. But even if the train accuracy is high, the test score clearly shows an important over-fitting. On contrary, this time with 3 active layer the best model performs actually quite poorly compared to the other ones. Thus, the overall best topology is the one using 2 hidden layers. Again there is a variation in the learning curve

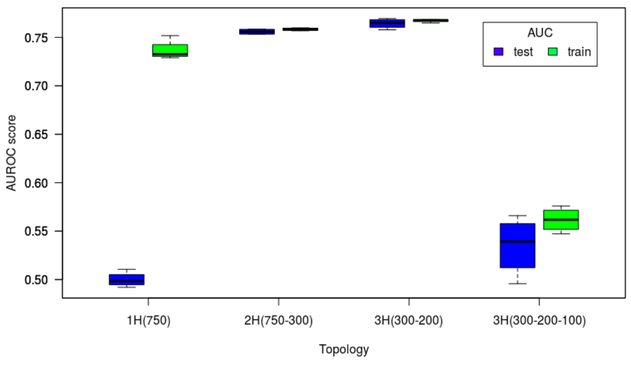


Figure 21: Overview of AUROC score at bit 132

that appears only for two layers. A temporary stop is visible just before a new intense error reduction. The shape can be found in appendix 36.

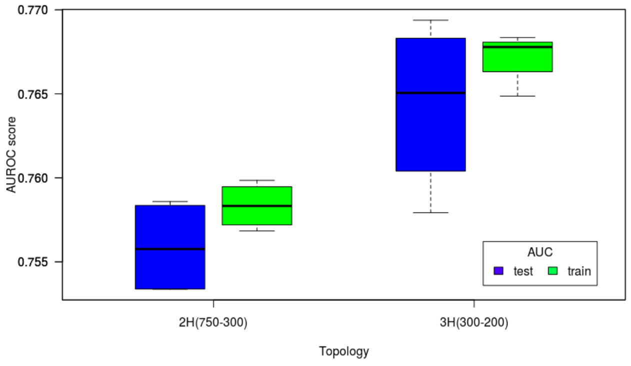


Figure 22: Overview of AUROC score at bit 132 with 2 hidden layers

A closer look at the two active hidden layers was made. Figure 37 shows the second run made for the optimization. This set of model where only focused on a topology using 2 layers but with a larger number of neurons $\in [750, 500, 300, 200, 100, 50]$. After running all simulations, one architecture was clearly better than the others. This topology is made with 2 layers of 300 and 200 neurons respectively. The second one, with 750 and 300, was clearly less sensitive than the first one.

There is a local maximum around the topology 300-200 at the bit 132. Furthermore, the effect of dropout was also tested on this model. The result was quite surprising. The dropout showed no effect on the over-fitting, but it had a clear effect on the learning curve. With a dropout of 0.5 the curve would stabilize in the end and always move between a median constant value. Thus, in case of a model that does not over-fit, using a dropout could be a mistake that can lead to a decrease in sensitivity.

Finally, adding more neurons from a certain threshold value decreases the accuracy. One possible reason could be that splitting the data too much ends up by creating neurons that contradict each other for one input value. In order to check this hypothesis, a study in depth of the different values for the weight and connections is needed. Currently, only *TensoFlow* in Python have the tool called *TensorBoard* to check in real time and create graphs containing all possible values during the training phase. So it could be interesting to adapt the code and implement those two models with *TensorFlow* in order to have a more precise overview of the model.

3.4.3 Bitstring prediction

The model using two layers with 300 and 200 neurons is tested on a simple bit. The results was really close to the one obtained for the more complex ones. So this topology was set and used for all bits. 157 models have been generated using this topology and previously optimized parameters. Besides 4 seeds were applied to overcome the randomization issue. Unfortunately the model generation showed many limitations on our methodology.

First, as showed for the result for the bit at position 8 in appendix 37, the test score is higher than the train score. The test set is too simple in comparison to the training set for those bits. But for other bits, the test set is representative of the all dataset. This complexity is a drawback in my validation strategy. Fixing a test set for all models would most likely not be able to be representative for the overall dataset for all bits. This issue is often seen during generalization of models out of one fixed dataset. Creating a new representative test set for every bit could be a solution. But this strategy would cause an increase of memory usage and running time.

Second, for most of the bits values of the different scoring is higher than 0.7. However, the "mean" structure end up, at several positions, in models with small over-fitting and average scores around 0.5. So using only one model quickly shows limitations. This decision was driven by the time limitation of the thesis. With 6 more months, a custom optimized model at every position could have been created. Optimized models per bit could perform way better than the current generalized model used.

In addition, figure 39-41 in the appendix shows the models and accuracy comparisons between deep learning, random forest. The comparison was done on only three different bits because of time limitations. Unfortunately, it was not possible within the time available to create new models for every bits. In the end of the comparison, random forest predict more accurately at every tested bit. However, even if the accuracy of both method are high, optimizing random forest is easier and quicker than an optimization process in deep learning. The incomplete optimization in deep learning in this study can be the cause of the difference observed with random forest. So, at this point deep learning might be able to overcome random forest in future optimized models.

Third, an epochs of 50 was set for the easy bits and 6,000 for the complex ones. Since the training can not be stopped automatically, due to possible temporary stabilization, a fixed number of epoch is set. But even this value can vary between bits and sometimes ends up to useless epochs or a non-complete learning too. However, a safety margin was taken to avoid as much as possible incomplete training. Besides, the training time is not increasing significantly since most of the bits can be predicted in minutes with only 50 epochs.

To conclude, 157 models have been generated with 4 different seeds. So in total 628 models were created only for an optimized model. The fingerprints prediction is more complex than expected. Training 4 times extensive models with 6,000 epochs takes a relatively long time. So even if the results are relatively good, the investment in time is substantial.

Furthermore, the figure 23 set a focus on the top of the bars of the figure 42 obtained during the accuracy testing. The bit prediction accuracy drops form the bit 75, the lowest accuracy is obtain at the bit 164 but remains over 50%. So models created using deep learning all perform better than a random distribution but it requires optimized models per bits to have possibly increasing the accuracy. Besides, the constant bit are making the generalization of those models impossible because it is impossible to predict these values. Eventually, this work requires more optimization, validation and a larger dataset. Though, it is important to note that in order to accomplish these points, an access to a more complete database and an hardware upgrade are mandatory.

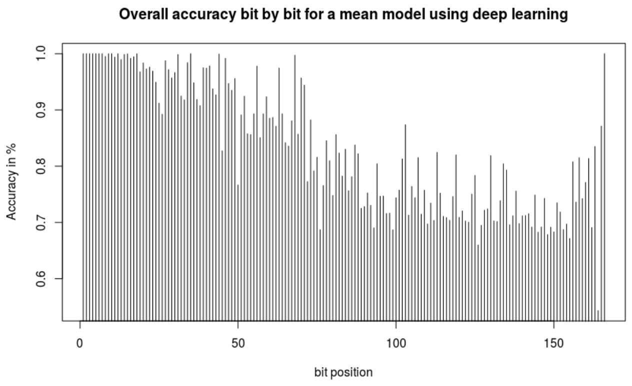


Figure 23: Accuracy results obtained for each bit in the MACCS standard of molecular descriptors

4 Discussion

This work has been done within 26 weeks using open sources programs and database. In addition, the two computers do not possess the best components to train neural networks. Those information are important in order to explain some choices that had to be made. Furthermore, the main focus was centered around methods to apply and implement deep leaning rather than results themselves.

First, using two different sets is a great help during the implementation of deep learning. It allows to the programmer to test various analysis and logfiles produced during training are valuable informations to understand how deep learning works. Furthermore, one code for a specific architecture can be used to train different topologies and models.

Unfortunately, the user has to spend a lot of time observing hundreds of models and topologies to understand the logfiles and the optimization. *MXNET* does not print in real time results and it makes checking or interpretation way harder than it would have been on *TensorFlow*. It is probable that MXNET is not complete enough compared to TensorFlow. Thus, if a user has to choose between those two package and programming languages, it would be better to use TensorFlow. This choice remains valid even if the user has no experience with Python. In fact, Python is similar to R. So learning one from the other is relatively easy and a quick task. Finally, there is way more available documentation and examples on TensorFlow than MXNET. So one way to improve this work is to generate the same models using TensorFlow. Then the comparison in results of those two packages can be valuable for future improvement of packages.

Second, nearly all models were generated using a consumer grade graphic card, *NVIDIA GTX 105Ti*. The memory limitation was the main issue during every single training. But the gain in speed is significant between CPU and GPU. In this study, the running time using a GPU is at least around fifty times faster than a using the CPU. Figure 43 in the appendix shows a detailed comparison between CPU and GPU-based training. Besides Figure 44 in the appendix shows that GPU based models are more sensitive than the CPU based ones.

The available CPU was a higher-end consumer grade 8 cores i7-6700 CPU against a consumer grade GPU sold at less than half the price of the processor installed. For example, by buying two compatible GPU without an SLI connexion for the same prince as the CPU, it is possible to reduce the running time by at least 50 times while stacking 10 GB of RAM. Although, the CPU have to be powerful enough to handle multiple graphic cards to avoid bottlenecks. This pricing example is not to show the market trends, but it is more to show the crucial importance of choosing the right components and pieces accordingly to the analysis. Not all calculations use the same type of operations and equipments have to be set accordingly.

Third, the bitstring application is incomplete and not fully optimized yet. It was a choice to only set a test set from the beginning and spend more time to understand models than validate models. Completing everything within 24 weeks was not doable. On one hand, the 157 models perform better than a randomly assigned value assigned for bits. On the other hand, those models are outperformed by random forest in the test for 3 bits. Thus, in the current state of those models, it is not really usable and even less generalizable. In order to achieve that goal, optimized topologies have to be set for every bit and to use a more complete training set.

Then, one final use of this application can be a tool to perform preprocessing on mass spectra directly. Using 157 models together took only a couple of seconds to give a possible bitstring with a probability of accuracy for each bit. This result can be used as a filter on a database search algorithm to narrow down possible candidates. In that case it can also increase the accuracy of the classification in the database by avoiding wrong associations.

To adapt this algorithm, one way could be to only select possible molecules and family and fingerprints over a certain threshold for an acceptable accuracy. For example all molecule with 80% of identical bits between the prediction an bitstrings found inside the database form a reduced list of possible matches.

4.1 Final thought about deep learning

Deep learning is certainly subject of an hype of the computer scientist community. Systems used by Facebook, Google, Tesla and others had showed crushing results. No other application or models can even reach the degree of precision of deep learning based models on some application such as image processing or speech recognition.

The benefits of deep learning are interesting and can end up solving problems than humans can not even grasp. It is the first time that a program possesses the ability to create a model from scratch and manage to perform better than humans. So where a human looks for a shape to identify an animal, an artificial neural network creates many more selection criteria. This is exactly why deep artificial neural network or all other architectures can handle problems humans usually struggle with.

However, deep learning has also many disadvantages. First, the investment in time to learn theory and gain experience in training is important. In a 6 month thesis, only basics about deep learning optimization has been acquired. Besides, the lack of documentation in analysis other than image classifications or natural language processing is very limited. This science is relatively new and the lack of standard methods slows down learning a lot.

Second, deep learning application need huge datasets to perform the training. Also the hardware investment is extremely important. Using GPU became mandatory in these sciences. However many of them are required to increase the dedicated RAM to compute larger models. Besides, GPUs are hardly used in calculation except in GIS rendering and neural network training. Even the predictions are done on CPU. So investing on GPU for deep learning can end up investing in a very specialized equipment nearly usable only for that purpose.

Finally, from my point of view, not all analysis can be made using deep learning. For non-complex analysis, a regression or man-made models perform at least as good as a deep learning model would. But in that case it could take months to product a model instead hours to use an optimized machine learning tool.

However, for problems impossible to solve or optimize further, deep learning models can worth investments. For example, deep learning could map genome quicker and more accurately than the currents pipelines of algorithms. A models could be more accurate by using artificially generated patterns and not stacking errors form different algorithm.

If a deep learning model is needed, it is crucial to first invest in suitable equipment such as memory, datasets and GPUs. Then it is central to anticipate the analysis and expected output in order to normalize and generate the input matrix, topology and validation trials.

To conclude, I believe that models using deep learning can allow scientist to create new tools to grasp a complexity level impossible for humans. Scientist could build new models and algorithms based on those results to make faster and more accurate programs. So even if the hype is probably over, the overall benefit of those models, this new science, will most likely be the source of new breakthroughs.

License

Acknowledgement

I would also like to thanks Dr. Vincent Payet for supervising this project too and pushing me to go beyond my initial training.

I would like to greatly thanks Dr. Jos Hageman, assistant professor at Wageningen University, for supervising this work.

Jos Hageman was formally trained as a medicinal chemist (Vrije Universiteit Amsterdam, 1998). During his studies he discovered his interest in quantitative matters and obtained a PhD in chemometrics on the topic of global optimisation (Nijmegen, 2004). After a few postdoctoral positions he became an assistant professor at the mathematics and statistical methods department at Wageningen University in 2007. He has years of experience in developing and applying novel algorithms, heuristics and statistical methodology for modelling a broad range of complex data structures. Among his research interests are statistical method development in life sciences, multi-omics data integration, multivariate power calculations, and many more.

References

[1] Mass Spectrometry in Metabolomics. URL http://link.springer.com/10.1007/978-1-4939-1258-2.

[2] Dávid Bajusz, Anita Rácz, and Károly Héberger. Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations? 7(1). ISSN 1758-2946. doi: 10.1186/s13321-015-0069-3. URL http://www.jcheminf.com/content/7/1/20.

[3] Essa Basaeed, Harish Bhaskar, and Mohammed Al-Mualla. Supervised remote sensing image segmentation using boosted convolutional neural networks. 99:19–27. ISSN 09507051. doi: 10.1016/j.knosys.2016.01.028. URL http://linkinghub.elsevier.com/retrieve/pii/S0950705116000484.

[4] Jens Behrmann, Christian Etmann, Tobias Boskamp, Rita Casadonte, Jörg Kriegsmann, and Peter Maass. Deep Learning for Tumor Classification in Imaging Mass Spectrometry. URL https://arxiv.org/abs/1705.01015.

[5] Paweł Cichosz. *Data Mining Algorithms: Explained Using R*. John Wiley & Sons, Ltd. ISBN 978-1-118-95095-1 978-1-118-33258-0. URL http://doi.wiley.com/10.1002/9781118950951.

[6] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. 22(12):3207–3220. ISSN 0899-7667, 1530-888X. doi: 10.1162/NECO_a_00052. URL http://www.mitpressjournals.org/doi/10.1162/NECO_a_00052.

[7] Kevin R. Coombes, Keith A. Baggerly, and Jeffrey S. Morris. Pre-Processing Mass Spectrometry Data. In Werner Dubitzky, Martin Granzow, and Daniel Berrar, editors, *Fundamentals of Data Mining in Genomics and Proteomics*, pages 79–102. Springer US. ISBN 978-0-387-47508-0. doi: 10.1007/978-0-387-47509-7_4. URL http://link.springer.com/10.1007/978-0-387-47509-7_4.

[8] Bo Curry and David E. Rumelhart. MSnet: A Neural Network which Classifies Mass Spectra. 3:213–237. ISSN 08985529. doi: 10.1016/0898-5529(90)90053-B. URL http://linkinghub.elsevier.com/retrieve/pii/089855299090053B.

[9] Susmita Datta. Feature Selection and Machine Learning with Mass Spectrometry Data. In Rune Matthiesen, editor, *Mass Spectrometry Data Analysis in Proteomics*, volume 1007, pages 237–262. Humana Press. ISBN 978-1-62703-391-6 978-1-62703-392-3. doi: 10.1007/

978-1-62703-392-3_10. URL http://link.springer.com/10.1007/978-1-62703-392-3_10.

[10] Katja Dettmer, Pavel A. Aronov, and Bruce D. Hammock. Mass spectrometry-based metabolomics. 26(1):51–78. ISSN 02777037, 10982787. doi: 10.1002/mas.20108. URL http://doi.wiley.com/10.1002/mas.20108.

[11] Santa Di Cataldo and Elisa Ficarra. Mining textural knowledge in biological images: Applications, methods and trends. 15:56–67. ISSN 20010370. doi: 10.1016/j.csbj.2016.11.002. URL http://linkinghub.elsevier.com/retrieve/pii/S2001037016300605.

[12] Seyedshams Feyzabadi. Joint Deep Learning for Car Detection. URL http://arxiv.org/abs/1412.7854.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press. \url{http://www.deeplearningbook.org}.

[14] J. A. Hageman, B. Engel, Ric C. H. de Vos, Roland Mumm, Robert D. Hall, H. Jwanro, D. Crouzillat, J. C. Spadone, and F. A. van Eeuwijk. Robust and Confident Predictor Selection in Metabolomics. In Susmita Datta and Bart J. A. Mertens, editors, *Statistical Analysis of Proteomics, Metabolomics, and Lipidomics Data Using Mass Spectrometry*, pages 239–257. Springer International Publishing. ISBN 978-3-319-45807-6 978-3-319-45809-0. doi: 10.1007/978-3-319-45809-0_13. URL http://link.springer.com/10.1007/978-3-319-45809-0_13.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York. ISBN 978-0-387-84857-0 978-0-387-84858-7. URL http://link.springer.com/10.1007/978-0-387-84858-7.

[16] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. 18(7):1527–1554. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.2006.18.7.1527. URL http://www.mitpressjournals.org/doi/10.1162/neco.2006.18.7.1527.

[17] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York. ISBN 978-1-4614-7137-0 978-1-4614-7138-7. URL http://link.springer.com/10.1007/978-1-4614-7138-7.

[18] Nicola Jones. Computer science: The learning machines. 505(7482):146–148. ISSN 0028-0836, 1476-4687. doi: 10.1038/505146a. URL http://www.nature.com/doifinder/10.1038/505146a.

[19] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. URL `https://arxiv.org/abs/1609.04836`.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. pages 1106–1114. URL `http://machinelearning.wustl.edu/mlpapers/papers/NIPS2012_0534`.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 86(11):2278–2324. ISSN 0018-9219. doi: 10.1109/5.726791.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. 521(7553):436–444. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14539. URL `http://www.nature.com/doifinder/10.1038/nature14539`.

[23] Polina Mamoshina, Armando Vieira, Evgeny Putin, and Alex Zhavoronkov. Applications of Deep Learning in Biomedicine. 13(5):1445–1454. ISSN 1543-8384, 1543-8392. doi: 10.1021/acs.molpharmaceut.5b00982. URL `http://pubs.acs.org/doi/abs/10.1021/acs.molpharmaceut.5b00982`.

[24] Min Meng, Yiting Jacqueline Chua, Erwin Wouterson, and Chin Peng Kelvin Ong. Ultrasonic signal classification and imaging system for composite materials via deep convolutional neural networks. ISSN 0925-2312. doi: 10.1016/j.neucom.2016.11.066. URL `http://www.sciencedirect.com/science/article/pii/S0925231217301522`.

[25] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. page bbw068. ISSN 1467-5463, 1477-4054. doi: 10.1093/bib/bbw068. URL `https://academic.oup.com/bib/article-lookup/doi/10.1093/bib/bbw068`.

[26] Michael A. Nielsen. Neural Networks and Deep Learning. URL `http://neuralnetworksanddeeplearning.com`.

[27] Kutkina Oksana and Stefan Feuerriegel. Deep Learning in R – R Blog. URL `http://www.rblog.uni-freiburg.de/2017/02/07/deep-learning-in-r/`.

[28] Nicole Rusk. Deep learning. 13(1):35–35. ISSN 1548-7091. URL `http://dx.doi.org/10.1038/nmeth.3707`.

[29] Sara Sheehan and Yun S. Song. Deep Learning for Population Genetic Inference. 12(3): e1004845. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1004845. URL `http://dx.plos.org/10.1371/journal.pcbi.1004845`.

[30] Sargur N. Srihari. Architecture Design for Deep Learning.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

[32] Masahiro Sugimoto, Masato Kawakami, Martin Robert, Tomoyoshi Soga, and Masaru Tomita. Bioinformatics Tools for Mass Spectroscopy-Based Metabolomic Data Processing and Analysis. 7(1):96–108. ISSN 15748936. doi: 10.2174/157489312799304431. URL `http://www.eurekaselect.com/openurl/content.php?genre=article&issn=1574-8936&volume=7&issue=1&spage=96`.

[33] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *PMLR*, pages 1139–1147. URL `http://proceedings.mlr.press/v28/sutskever13.html`.

[34] Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. 43(6):1947–1958. ISSN 0095-2338. doi: 10.1021/ci034160g. URL `http://pubs.acs.org/doi/abs/10.1021/ci034160g`.

[35] Li Kuo Tan, Yih Miin Liew, Einly Lim, and Robert A. McLaughlin. Convolutional neural network regression for short-axis left ventricle segmentation in cardiac cine MR sequences. 39:78–86. ISSN 13618415. doi: 10.1016/j.media.2017.04.002. URL `http://linkinghub.elsevier.com/retrieve/pii/S1361841517300543`.

[36] Yingfeng Wang, Guruprasad Kora, Benjamin P. Bowen, and Chongle Pan. MIDAS: A Database-Searching Algorithm for Metabolite Identification in Metabolomics. 86(19):9496–9503. ISSN 0003-2700, 1520-6882. doi: 10.1021/ac5014783. URL `http://pubs.acs.org/doi/10.1021/ac5014783`.

[37] Yan Xu, Tao Mo, Qiwei Feng, Peilin Zhong, Maode Lai, and Eric I-Chao Chang. Deep learning of feature representation with multiple instance learning for medical image analysis. pages 1626–1630. IEEE. ISBN 978-1-4799-2893-4. doi: 10.1109/ICASSP.2014.6853873. URL `http://ieeexplore.ieee.org/document/6853873/`.

[38] Cha Zhang and Yunqian Ma, editors. *Ensemble Machine Learning*. Springer US. ISBN 978-1-4419-9325-0 978-1-4419-9326-7. URL `http://link.springer.com/10.1007/978-1-4419-9326-7`.

[39] Zhi-Shui Zhang, Li-Li Cao, Jun Zhang, Peng Chen, and Chun-hou Zheng. Prediction of Molecular Substructure Using Mass Spectral Data Based on Deep Learning. In De-Shuang

Huang, Kang-Hyun Jo, and Abir Hussain, editors, *Intelligent Computing Theories and Methodologies*, volume 9226, pages 520–529. Springer International Publishing. ISBN 978-3-319-22185-4 978-3-319-22186-1. doi: 10.1007/978-3-319-22186-1_52. URL `http://link.springer.com/10.1007/978-3-319-22186-1_52`.

[40] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. 12(10):931–934. ISSN 1548-7091, 1548-7105. doi: 10.1038/nmeth.3547. URL `http://www.nature.com/doifinder/10.1038/nmeth.3547`.

Appendix

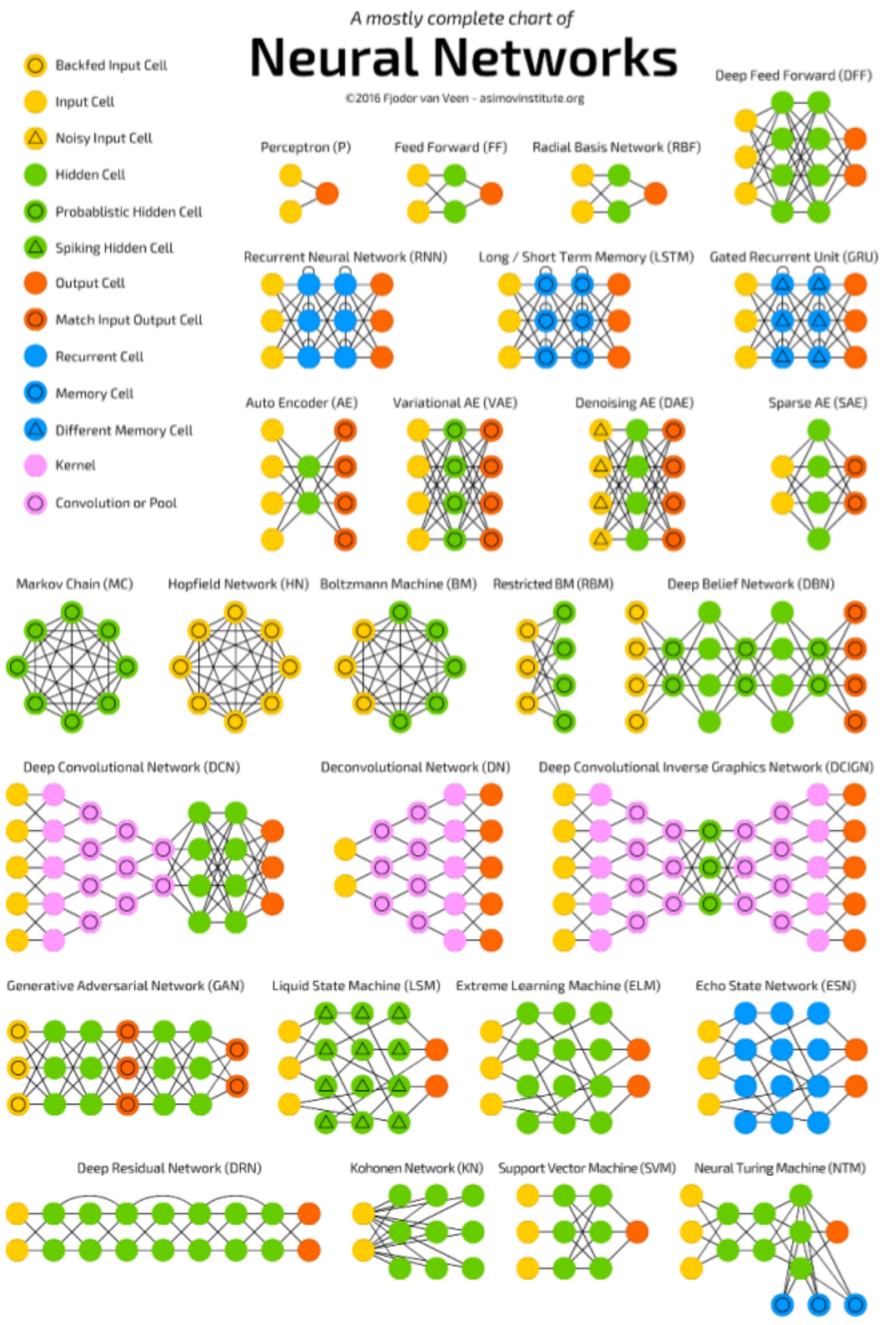
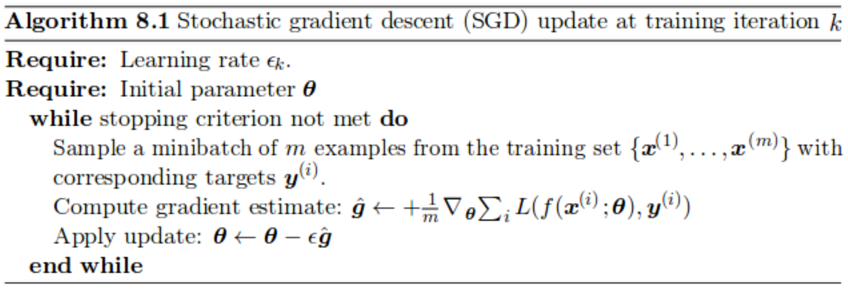
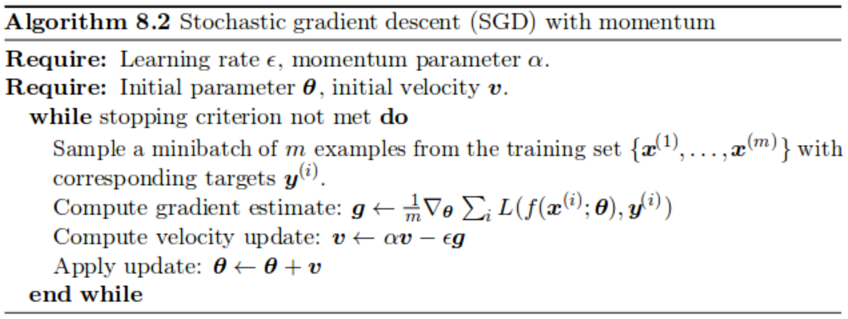

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Deep Feed Forward (DFF)
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Figure 24: Examples of architecture design[30]



```
Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration $k$
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
  end while
```

Figure 25: Pseudo-code Stochastic Gradient algorithm[13]



```
Algorithm 8.2 Stochastic gradient descent (SGD) with momentum
Require: Learning rate $\epsilon$, momentum parameter $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

Figure 26: Pseudo-code Stochastic Gradient algorithm with momentum[13]

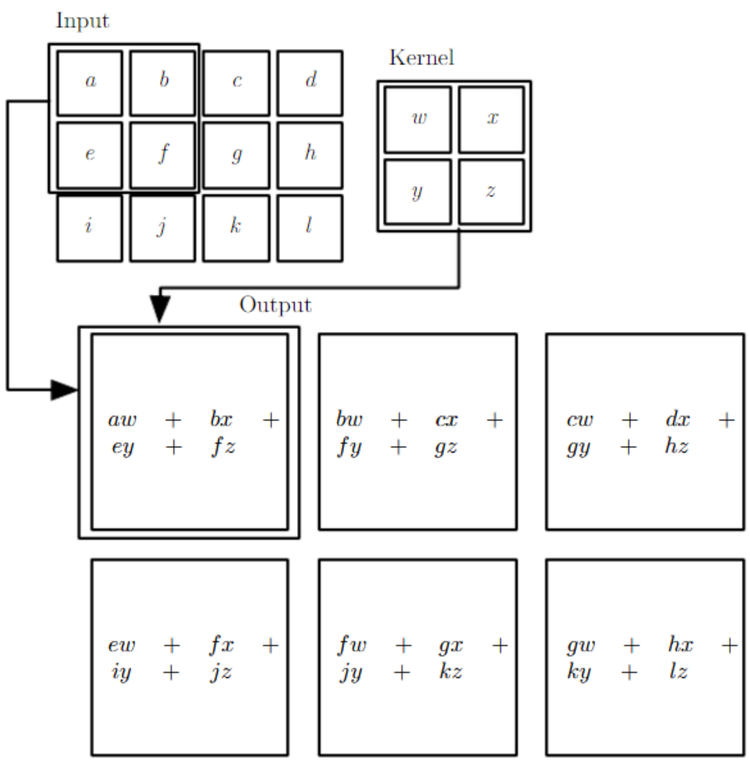


Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Figure 27: CNN developed example of a convolutional layer functions[13]

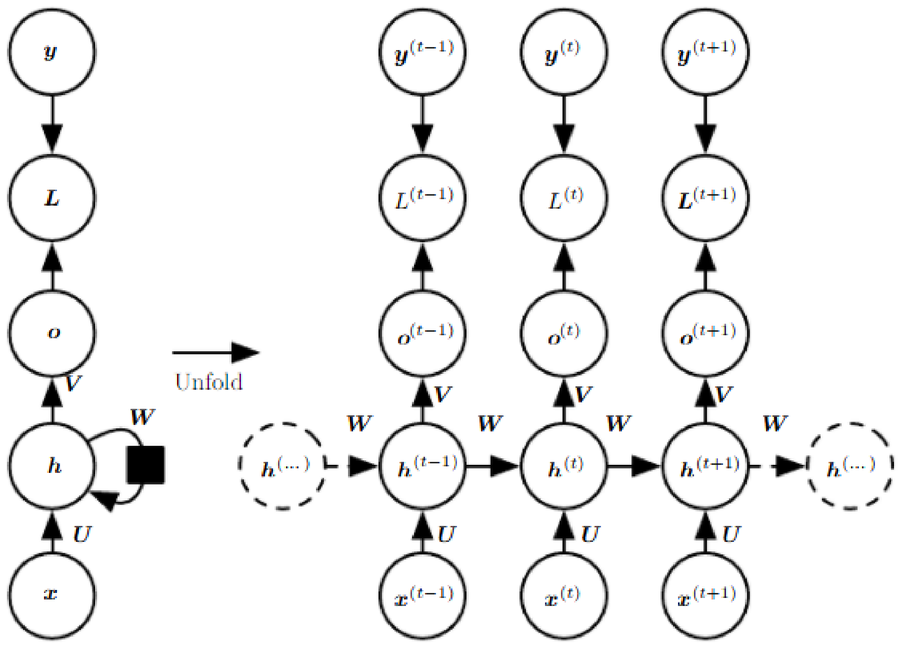
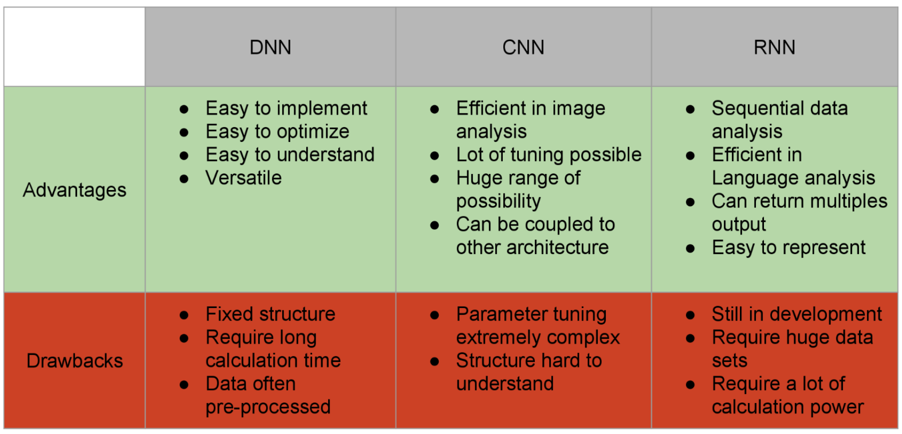


Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation 10.8 defines forward propagation in this model. *(Left)*The RNN and its loss drawn with recurrent connections. *(Right)*The same seen as an time-unfolded computational graph, where each node is now associated with one particular time instance.

Figure 28: RNN developed example and quick explanation[13]



	DNN	CNN	RNN
Advantages	• Easy to implement • Easy to optimize • Easy to understand • Versatile	• Efficient in image analysis • Lot of tuning possible • Huge range of possibility • Can be coupled to other architecture	• Sequential data analysis • Efficient in Language analysis • Can return multiples output • Easy to represent
Drawbacks	• Fixed structure • Require long calculation time • Data often pre-processed	• Parameter tuning extremely complex • Structure hard to understand	• Still in development • Require huge data sets • Require a lot of calculation power

Figure 29: Table summarizing the advantages and drawbacks of the three main architecture

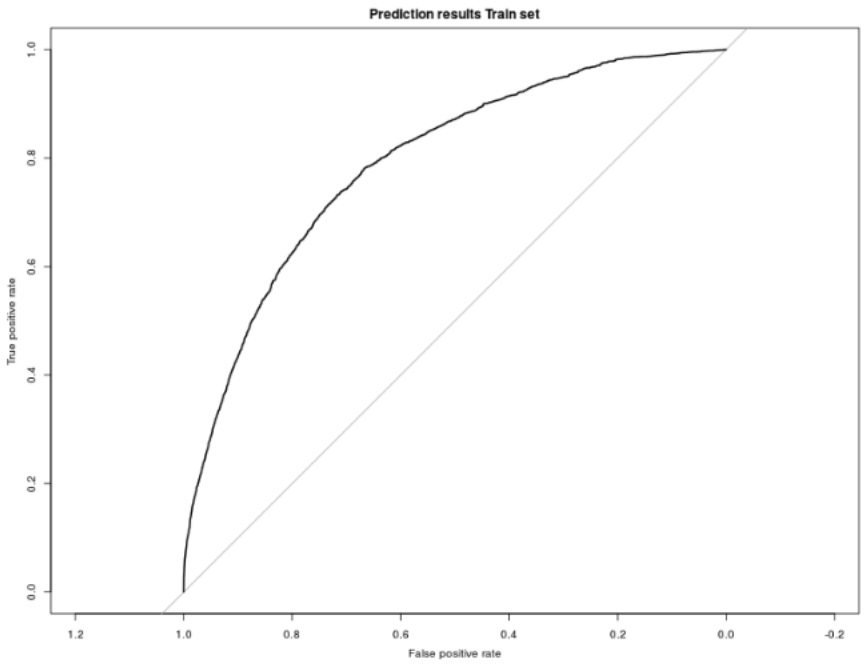
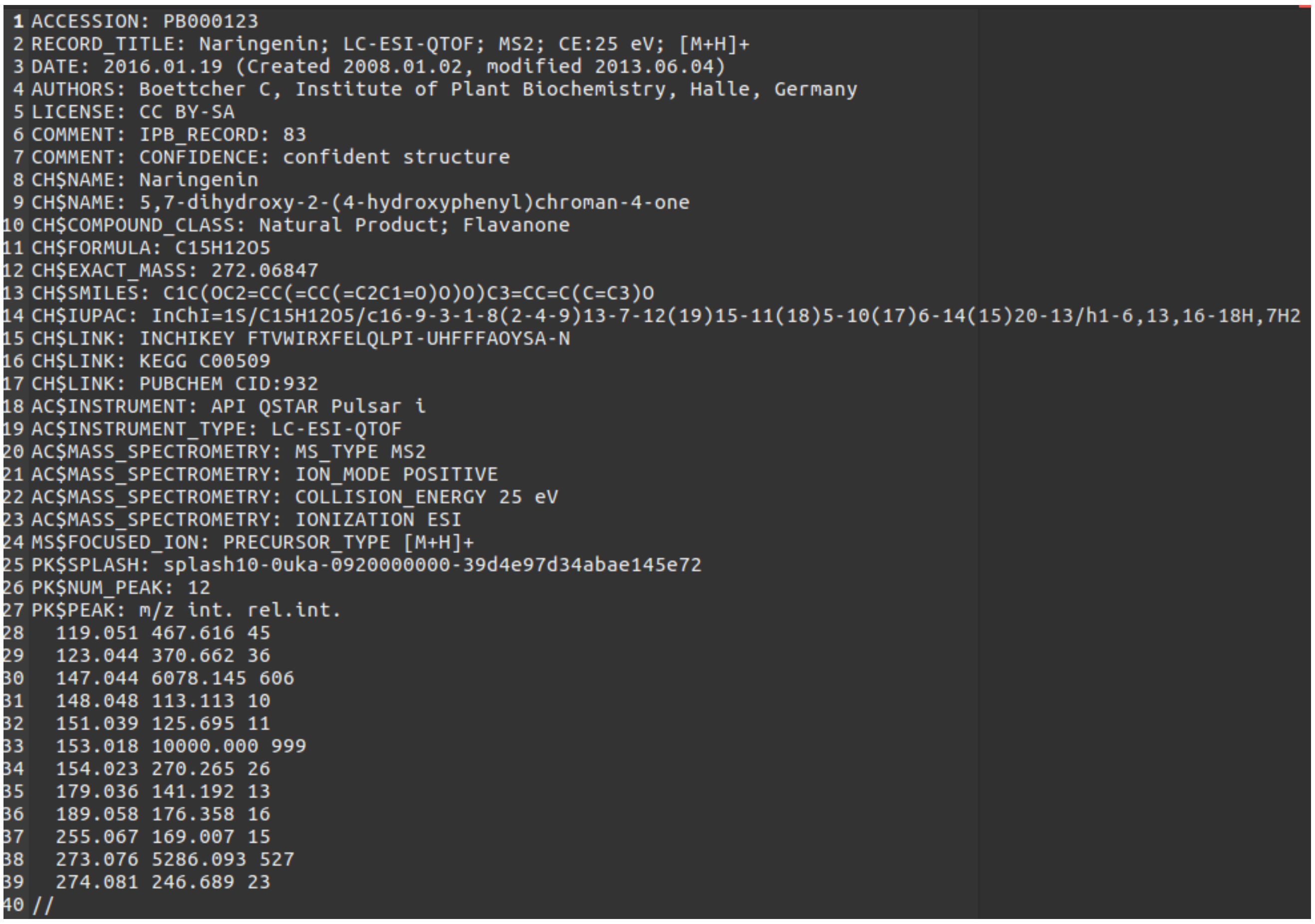


Figure 30: ROC curve output from a binary classifier model



```
ACCESSION: PB000123
RECORD_TITLE: Naringenin; LC-ESI-QTOF; MS2; CE:25 eV; [M+H]+
DATE: 2016.01.19 (Created 2008.01.02, modified 2013.06.04)
AUTHORS: Boettcher C, Institute of Plant Biochemistry, Halle, Germany
LICENSE: CC BY-SA
COMMENT: IPB_RECORD: 83
COMMENT: CONFIDENCE: confident structure
CH$NAME: Naringenin
CH$NAME: 5,7-dihydroxy-2-(4-hydroxyphenyl)chroman-4-one
CH$COMPOUND_CLASS: Natural Product; Flavanone
CH$FORMULA: C15H12O5
CH$EXACT_MASS: 272.06847
CH$SMILES: C1C(OC2=CC(=CC(=C2C1=O)O)O)C3=CC=C(C=C3)O
CH$IUPAC: InChI=1S/C15H12O5/c16-9-3-1-8(2-4-9)13-7-12(19)15-11(18)5-10(17)6-14(15)20-13/h1-6,13,16-18H,7H2
CH$LINK: INCHIKEY FTVWIRXFELQLPI-UHFFFAOYSA-N
CH$LINK: KEGG C00509
CH$LINK: PUBCHEM CID:932
AC$INSTRUMENT: API QSTAR Pulsar i
AC$INSTRUMENT_TYPE: LC-ESI-QTOF
AC$MASS_SPECTROMETRY: MS_TYPE MS2
AC$MASS_SPECTROMETRY: ION_MODE POSITIVE
AC$MASS_SPECTROMETRY: COLLISION_ENERGY 25 eV
AC$MASS_SPECTROMETRY: IONIZATION ESI
MS$FOCUSED_ION: PRECURSOR_TYPE [M+H]+
PK$SPLASH: splash10-0uka-0920000000-39d4e97d34abae145e72
PK$NUM_PEAK: 12
PK$PEAK: m/z int. rel.int.
  119.051 467.616 45
  123.044 370.662 36
  147.044 6078.145 606
  148.048 113.113 10
  151.039 125.695 11
  153.018 10000.000 999
  154.023 270.265 26
  179.036 141.192 13
  189.058 176.358 16
  255.067 169.007 15
  273.076 5286.093 527
  274.081 246.689 23
//
```

Figure 31: Example of an entry in MassBank database

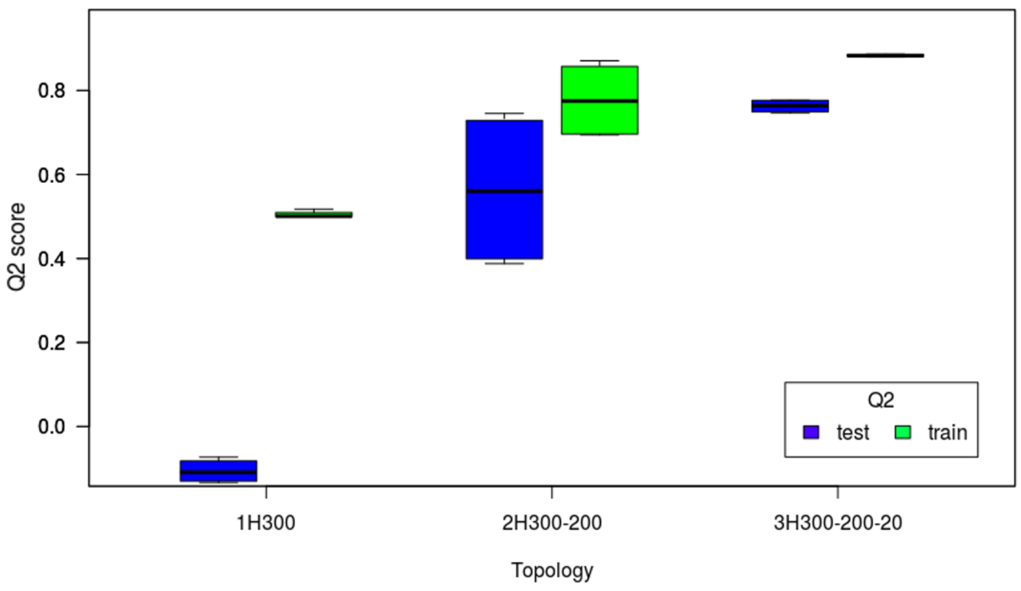


Figure 32: Complete comparison of Q^2 score in molecular weight prediction between 3 topologies

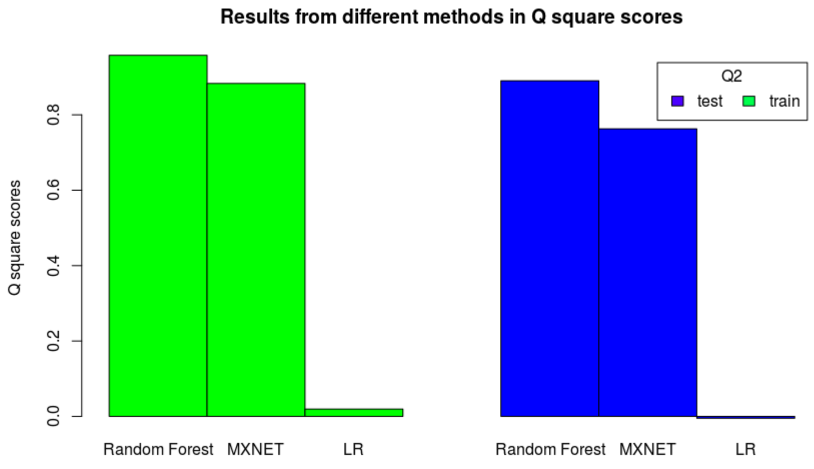
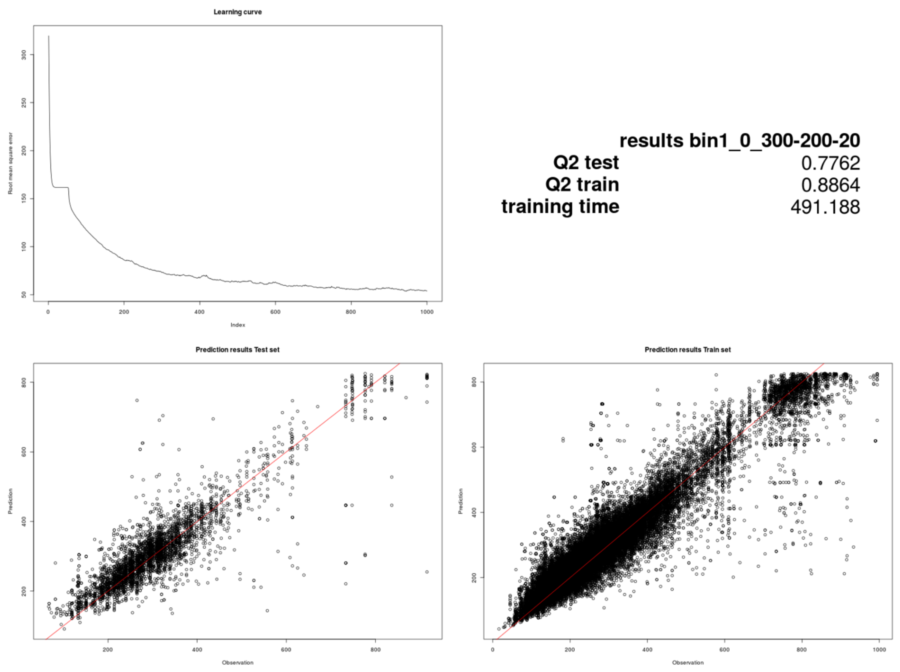


Figure 33: Molecular weight prediction Q^2 scores comparison



	results bin1_0_300-200-20
Q2 test	0.7762
Q2 train	0.8864
training time	491.188

Figure 34: Overview of main characteristics for the chosen model in molecular weight prediction

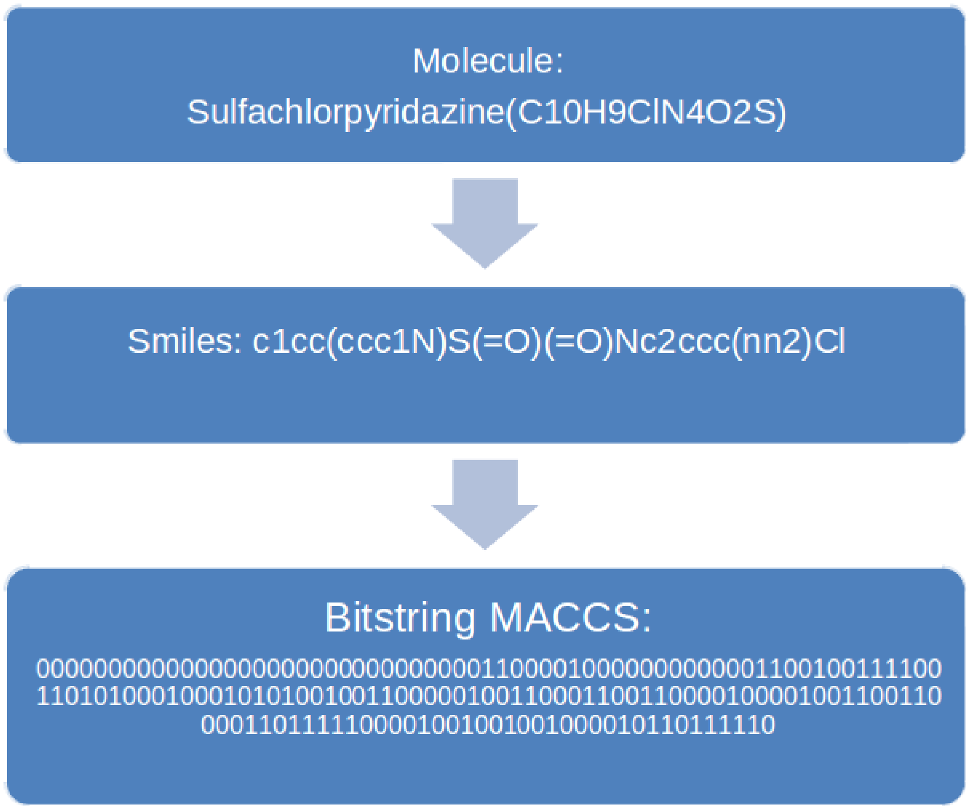
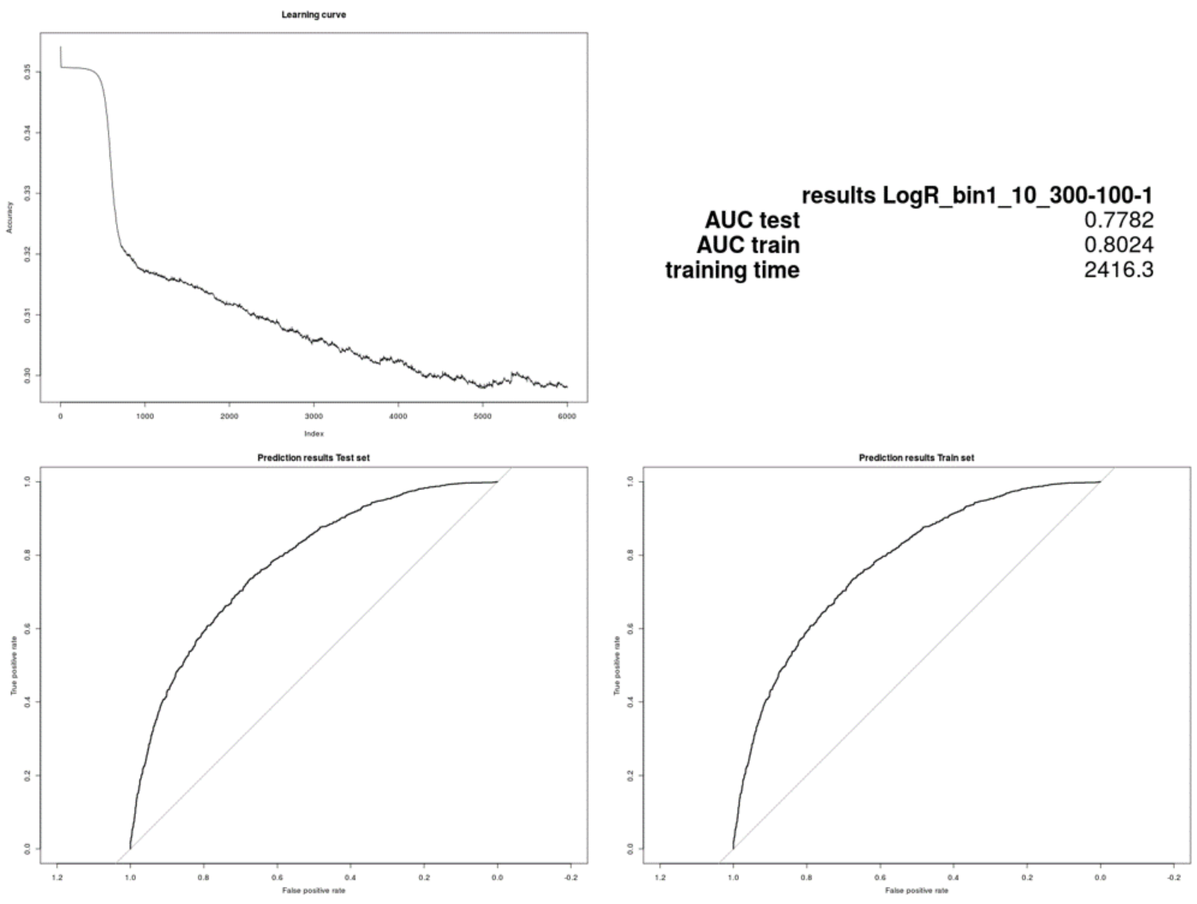
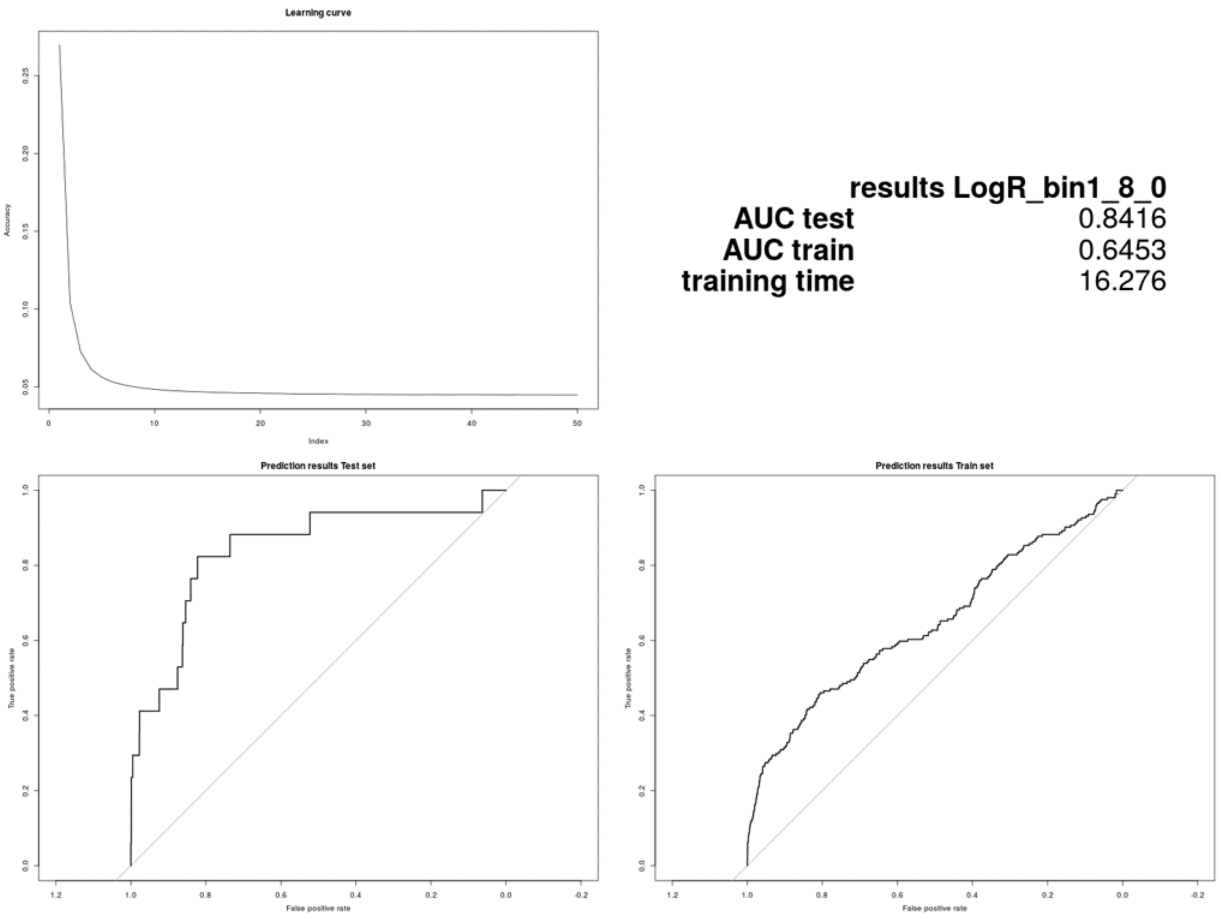


Figure 35: Flow chart of the bitstring translation process



results	LogR_bin1_10_300-100-1
AUC test	0.7782
AUC train	0.8024
training time	2416.3

Figure 36: Optimized model for a complex bit, position 132



results LogR_bin1_8_0	
AUC test	0.8416
AUC train	0.6453
training time	16.276

Figure 37: Optimized model for a, easy bit, position 8

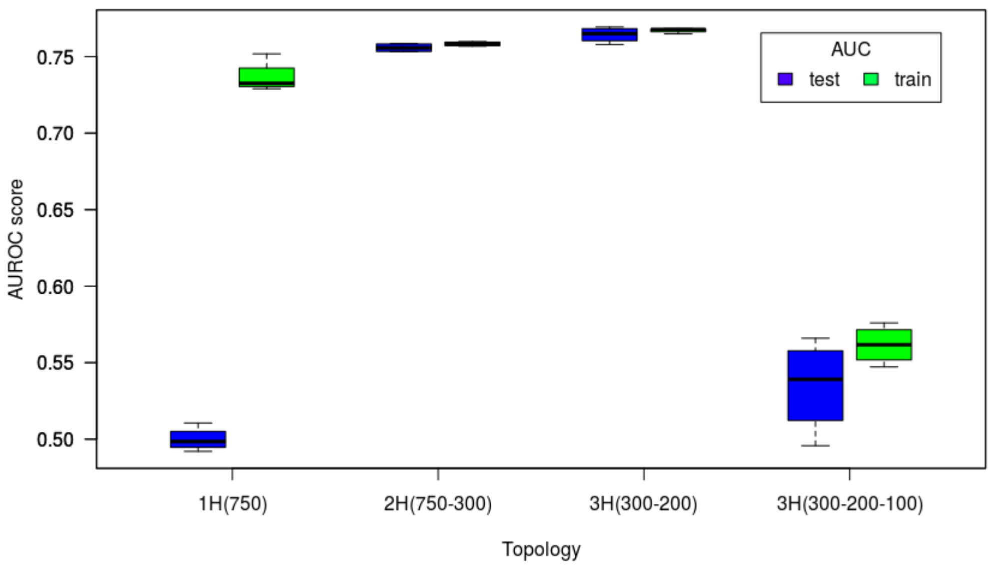


Figure 38: Overview of AUROC score at bit 132

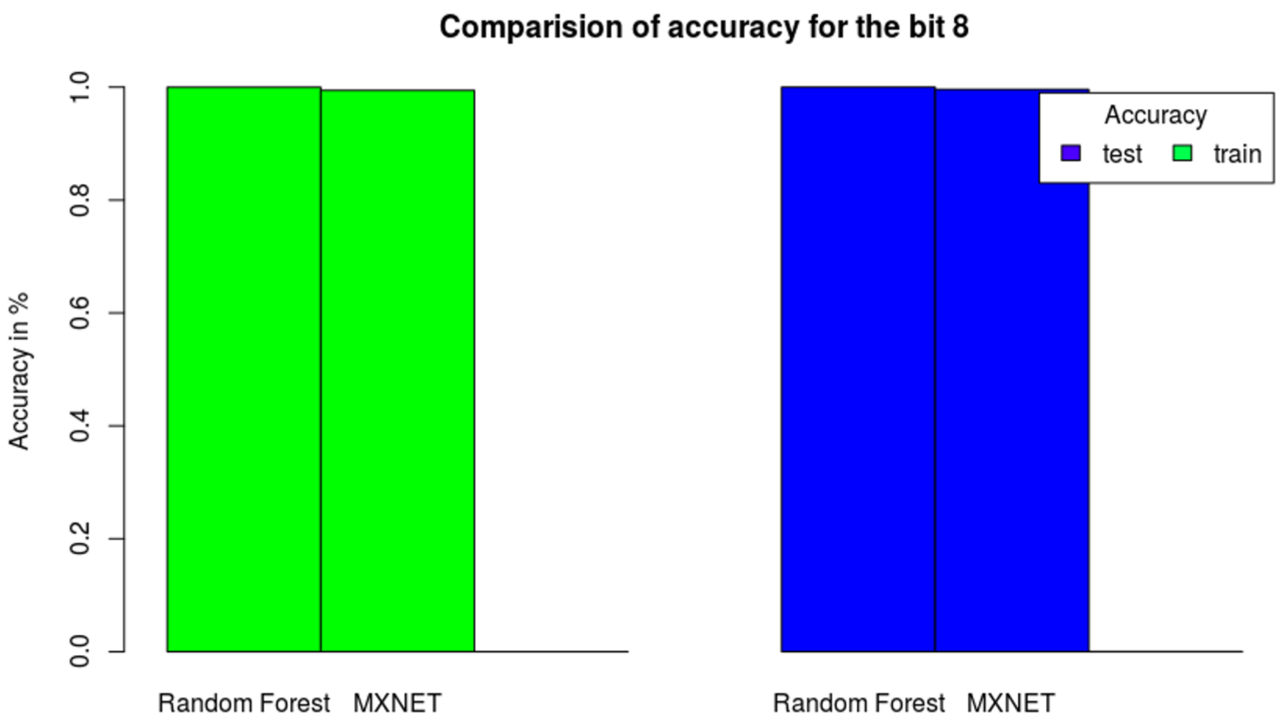


Figure 39: Comparison of accuracy between various methods at bit 8

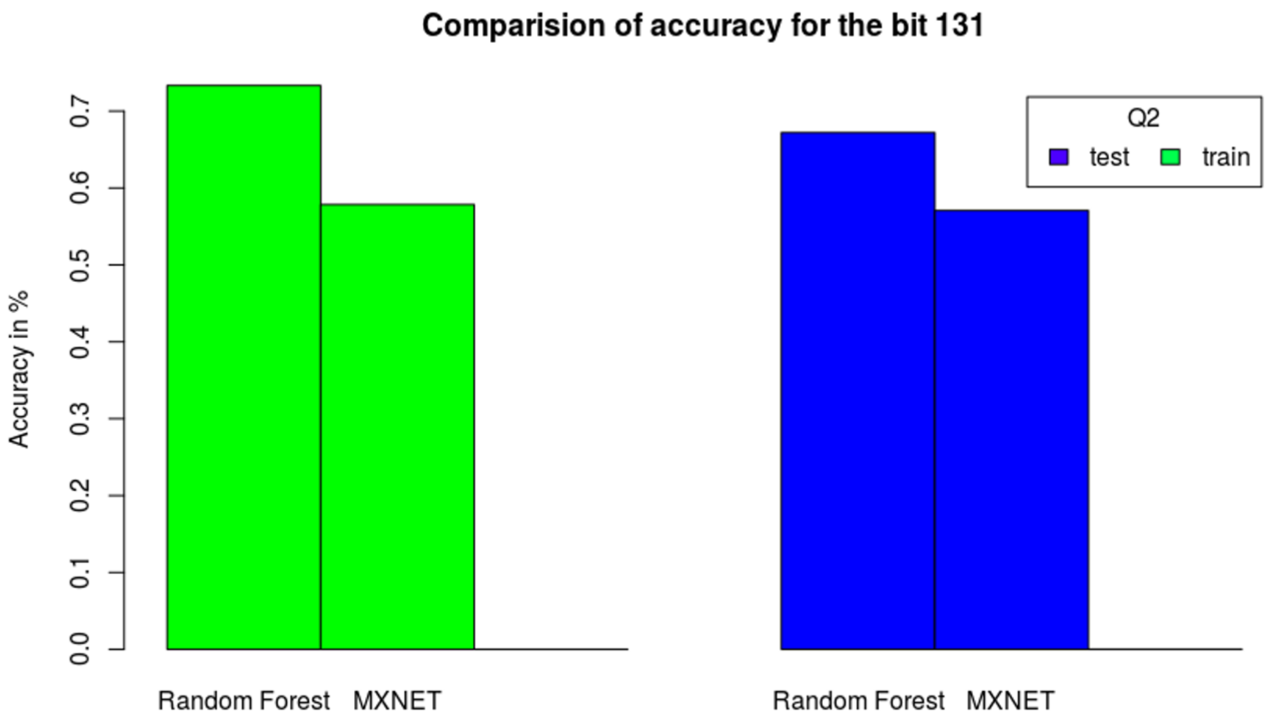


Figure 40: Comparison of accuracy between various methods at bit 131

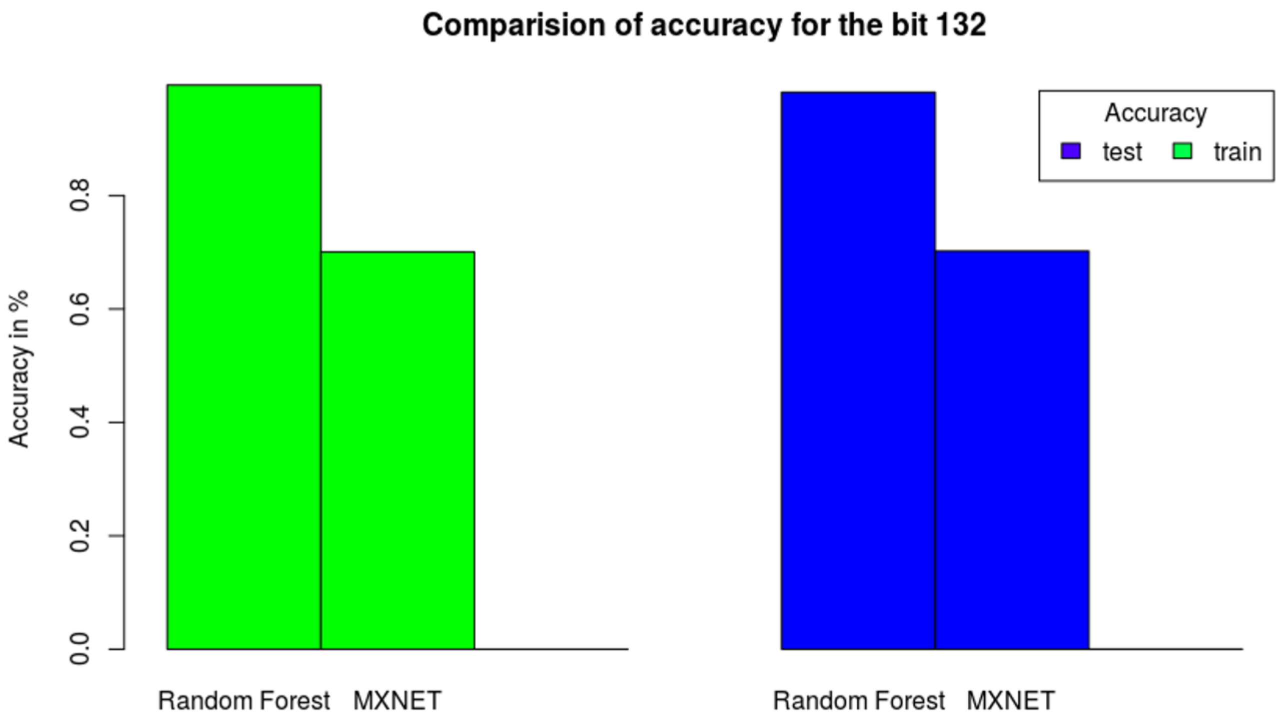


Figure 41: Comparison of accuracy between various methods at bit 132

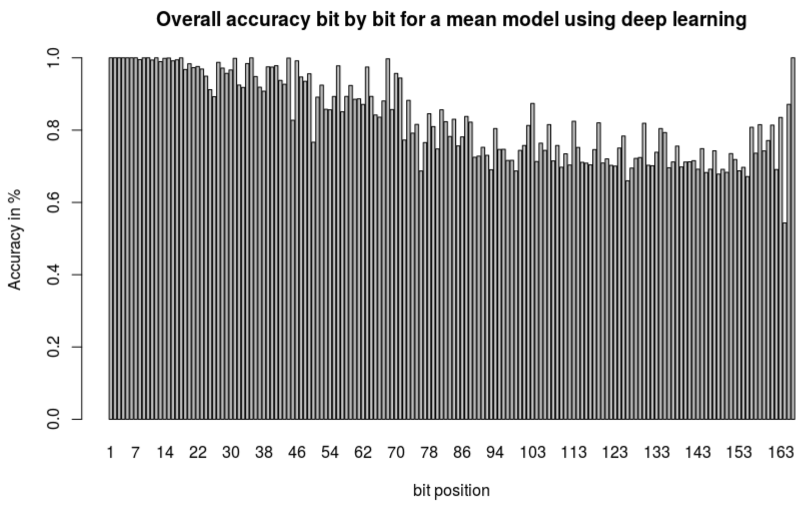


Figure 42: Overall accuracy obtained per bit

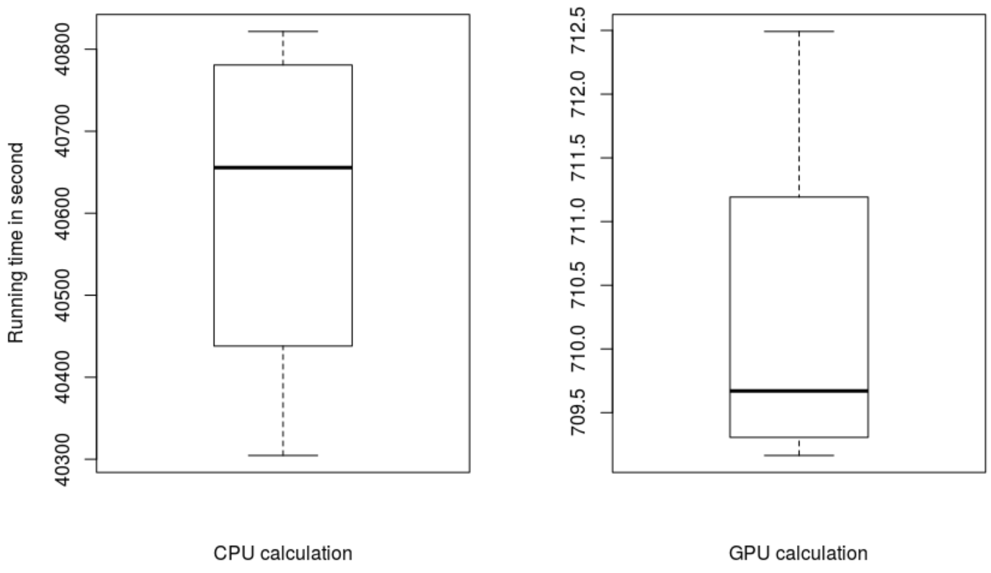


Figure 43: Running time comparison between CPU and GPU based leaning

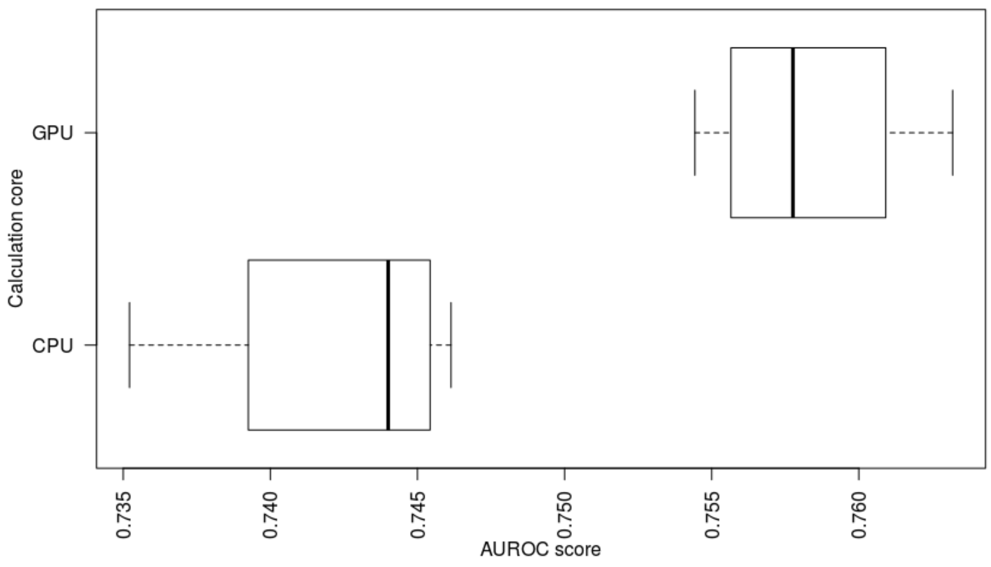


Figure 44: Accuracy comparison between CPU and GPU based leaning

Author: Antoine Drouillard	Year: 2016-2017	Confidential: No

THEME : Recherche et développement

Title: Using deep learning for molecular structure prediction from mass spectra

Key- words: Deep learning, Deep Neural Network, mass spectra, molecular descriptors, prediction models

Summary:
Deep learning is a part of machine learning that is subjected of a hype and considered as a breakthrough in artificial intelligence. It can be considered as a deeper and more versatile network than single layers neural network. Those models sometimes outperform humans on some tasks such as image classification. This study was conducted to evaluate the utility of deep learning based models for various prediction purposes. To do so, the analysis of mass spectra has been used. Two applications have been created, the first one is a model that predict a molecular weight from the spectra. The second one is a combination of models that forms a molecular descriptor following the MACCS bitstring standard. Every model produced are using a Deep Neural Network (DNN) architecture and go through the same optimization process. Eventually, all needed models were produced but not fully optimized. But the results are still promising, it is possible to increase the accuracy considerably by completing the optimization process. Finally, the performance for both applications showed that deep learning based models can also be meaningful to analyse biological data.

Titre : Utilisation du deep learning pour la prédiction de structures moléculaires à partir de l'analyse de spectre de masse.

Mots-clés : Deep learning, Deep Neural Network, spectrométrie de masse, descripteurs moléculaires, modèle de prédiction

Résumé :
Deep learning, qui peut être traduit par apprentissage profond, est une branche des systèmes neuronaux artificiels. Cette nouvelle technologie est une avancée majeure dans le domaine de l'intelligence artificielle et de l'analyse de données. Certains modèles et applications créés utilisant cette technologie ont montré des résultats impressionnants qui sont parfois plus précis que l'être humain. Cette étude a pour objectif d'évaluer la potentielle efficacité l'apprentissage profond sur des domaines autres que l'analyse d'images ou de séquences. Pour ce faire, deux applications ont été créés pour analyser des spectres de masses. La première a pour but de déterminer le poids moléculaire. La deuxième est un ensemble de modèle de prédiction binaire, qui permet à l'utilisateur de prédire le descripteur moléculaire au format MACCS de la molécule analysée. Chaque modèle utilise une architecture du type Deep Neural Network (DNN) et a subi plusieurs étapes d'optimisation. Finalement, l'optimisation effectuée est partielle et peut être encore plus poussée. Cependant, les résultats obtenus sont encourageants. En effet, il serait possible d'augmenter significativement la précision de ces applications en poussant plus loin l'optimisation. Pour conclure, les performances pour les deux applications montrent que l'apprentissage profond peut aussi avoir un réel potentiel dans l'analyse de données biologique.

Total number of volumes: 1
Total number of pages for the main document: 36 pages

Employer: Dr Jos Hageman, Assistant professor of Statistics at Wageningen University.