

# Méthodes avancées en exploitation de donnée (MATH80619)

Estefan Apablaza-Arancibia	11271806
Adrien Hernandez	11269225

March 15, 2020

## List of Acronyms

**API** Application Programming Interface

**CNN** Convolutional Neural Network

**FNN** Feed-Forward Neural Network

**RNN** Recurent Neural Network

# 1 Introduction

In the first section of this paper, a literature review covers the neural networks and deep learners algorithms, focusing on different type of neural networks architecture; the purpose of adding multiple hidden layers and, ultimately, what are the challenges regarding the increase in computing time. Furthermore, in the methodology section, a list of deep learning projects are shown in order to understand some patterns and methods. Then, an exhaustive list of the R libraries allowing to build neural networks and deep learners models. Correspondingly, the advantages and disadvantages of each libraries, their capabilities as well as what you can expect when using them. To sum up, the last section will give concrete examples on how to implement the neural networks and deep learners models with these libraries, using real data.

## 2 Literature Review

### 2.1 What is Deep Learning and why use it?

Deep learning is a subset field of artificial intelligence and can be seen as a specific way of doing machine learning. Deep learning algorithms can be seen as feature representation learning. Indeed, by applying to the data multiple non-linear transformations through the use of hidden layers, deep learning models have their own trainable feature extration capability making it possible to learn specific features from the data without needing a specific human domain expert. This means that deep learning models won't require the features extraction step that is essential for classic machine learning models. However, increasing the models capacity by adding hidden layers, requires increasingly computing power and slow down the training process of the model. The choice of hyperparameters, programming languages and memory management will therefore be important criteria to take into account while building deep learning models

Since the last decade, deep learning models have shown notable predictive power and have been revolutionizing an important number of domain such as computer vision, natural language understanding, fraud detection, health and much more.

As a first glance in the subject, it is highly recommended it to read a reference from pioneers in the field ([6, Chapter 1]).

#### 2.1.1 Feed Forward Neural Network

"Deep feedforward networks, also called feedforward neural networks, or multilayer perceptrons (MLPs), are the quintessential deep learning models." [6]

**FNN! (FNN!)** models are inspired from biology and by the way the human brain works. In a neural network, each neuron takes input from other neurons, processes the information and then transmits outputs to next neurons. Artificial neural networks follow the same process as each neuron will perform the weighted sum of inputs and will add a bias as a degree of freedom. It will then apply a non-linear transformation before outputting the information. Thus, the information goes forward in the network; neurons transmit information from the input layers towards the

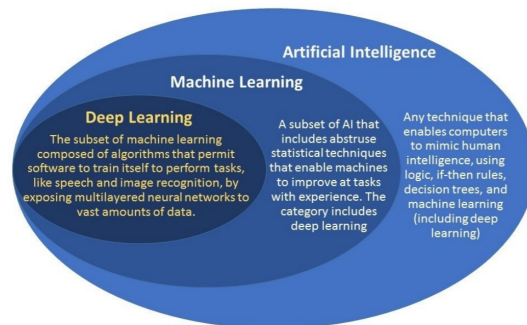


Figure 1: Artificial Intelligence vs Machine Learning vs Deep Learning

output layer. It is important to know that in a feedforward neural networks (fig. ??) the neurons of a same layer are not connected to each other; they do not allow any feedback connections within a same layer. If we want to allow this process, we will be looking at recurrent neural networks.

The equation a neuron input is

$$a(x) = b + \sum_i w_i x_i \quad (1)$$

and the output

$$h(x) = g(a(x)) \quad (2)$$

where:

$x$  = the input data

$b$  = the bias term

$w$  = the weight or parameter

$g(\dots)$  = the activation function

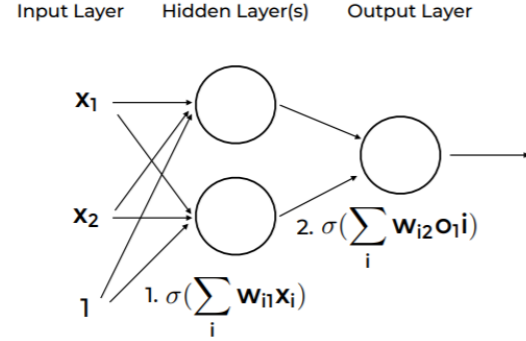


Figure 2: feedforward neural networks

Here, **the bias term  $b$**  and **the weights or parameter  $w$**  will be learned by the model in order the minimize

a cost function, through the use of the gradient descent method. Then, the network will use the backpropagation algorithm where the error is backpropagated from the output to the input layers and the bias and weighted are then updated accordingly.

Regarding the **activation function  $g(\dots)$** , the common practice is to use a ReLu (rectified linear unit) as the activation function for the neurons of the hidden layers. Regarding the neurons of the last layer, the activation function will be chosen accordingly to the task we want our model to perform:

Output type	Output Unit	Equivalent Statistical Distribution
Binary (0,1)	$\text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$	Bernoulli
Categorical (0,1,2,3,k)	$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_r \exp(z_r)}$	Multinoulli
Continuous	Identity(a) = a	Gaussian
Multi-modal	mean, (co-)variance, components	Mixture of Gaussians

Figure 3: Activation functions: output units. HEC

A detailed explanation of the theory of feedforward neural network can be found in [6, Chapter 6]

### 2.1.2 CNN! (CNN!)

The **CNN!** are a modified architecture of **FNN!** that leverage the feature engineering that used to be hand maded by domain experts. This class of deep neural network are commonly use for image recognition and classification that can serve different applications such as facial recongnition, document analysis and speech recognition. The original

**FNN!** are not suited analyzing large size images since the weights increase exponentially and, at the end, don't perform very well.

A standard architecture of a CNN model is commonly represented this way: We start with an input image to which we are going to apply several kernels, a.k.a features detectors, in order to create feature maps that will form what we call a convolutional layer. A common practice is to apply the ReLu activation function to the convolutional layer output in order to increase the non linearity of the images before using the pooling method to create a pooling layer. The next step will be to combine all the pooled images from the pooling layer into one single vector, which is called flattening and will be the input of a FNN that will perform the desired task, for instance, classification. CNN parameters are trained with the backpropagation algorithm where the gradients are propagated from the output layer of the FFN to the input of the CNN in order to minimise a cost function, most of the time a categorical cross-entropy if we are performing a multi-class classification.

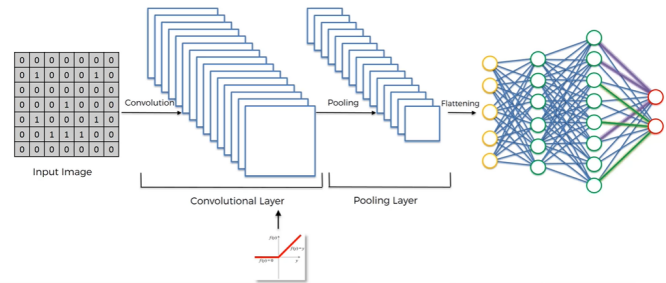
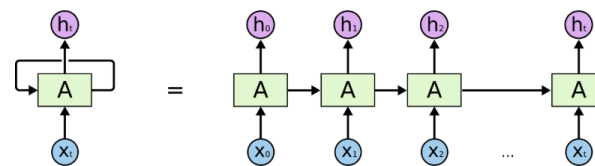


Figure 4: CNN, Deep Learning A-Z by SuperDataScience

To get a detailed explanation of convolutional neural networks we recommend to read [6, Chapter 9].

### 2.1.3 RNN

Recurrent neural networks are a more advanced type of neural networks architecture having proven state-of-art performance for solving tasks related to sequential data such as Natural Language Processing (NLP), anomaly detection and event forecasting. As a key differentiator from feedforward neural networks, is that RNNs use feedback loops connections instead of feedforward connections and get an internal memory. Indeed, that take as input each step of the sequential data as well as what they have learned over time. One of their other advantage is to be able to handle input and output sequences with different sizes. "A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor" Olah.



An unrolled recurrent neural network.

Figure 5: RNN, Understanding LSTM Networks by Olah

To get a detailed explanation of the theory of recurrent neural networks we recommend to read [6, Chapter 10]. We also recommend reading this blog post

## 2.2 How to integrate Deep Learning in R

The integration of Deep learning in R can be separate in two part ; (1) The **API!** (**API!**) integration and (2) the standalone R packages.

1. The **API!** will give the possibility to control externally through R an existing installation. In other words, a software translator for a already known software installation. This approach is not always trivial to install nor to manipulate but in long term will probably give the best flexibility in terms of Deep Learning projects.
2. The standalone R packages will be packages that require no third party software in order to create the deep learning projects. This approach is relatively fast to install but is restraining in terms deep learning architecture.

A list of the available resources with installation and examples tutorials can be found in section ?? and ?? .

## 3 Methodology

This section is divided in two; the dataset description and the benchmark type. The dataset description is mostly to describe what dataset is going to be use for the different type of deep learning algorithms. The second section will describe two indicators in order to compare results of similar algorithms. The first indicator is mostly about a simple accuracy base on the same training and validation dataset and second indicator is about the time it took to execute the same task on different resources.

### 3.1 Dataset

#### 3.1.1 FNN

For the **FNN!**, the training data will be the well known *Iris* dataset. It contains four covariables and one categorical target variable.

```
library(datasets)
data(iris)
```

#### 3.1.2 CNN

For the **CNN!**, the training data will also be a well known dataset called CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. The easiest way in R to download this package is using the Keras API. The training dataset has 50 000 images in three color (Red-Green-Blue) that are 32 pixels by 32 pixels. The testing dataset has a total of 10 000 images

```
library(keras)
cifar10 = dataset_cifar10()
```

#### 3.1.3 RNN

The dataset use to test an RNN is the IMDB Movie reviews sentiment classification. This pack

```
imdb = dataset_imdb( path = "imdb.npz", num_words = NULL, skip_top = 0L, maxlen = NULL,
                    seed = 113L, start_char = 1L, oov_char = 2L, index_from = 3L)
```

## 4 Available resources

Like explain earlier this section will be divided in two in order to really differentiate the R packages standalone of the resources that add an application interface.

## 4.1 R packages

4.1.0. à voir si on a encore de la place pour mettre un tableau récapitulatif

### 4.1.1 Neuralnet

**Description** One of the easiest R package to use which perform, according to its CRAN description, "training of neural networks using the backpropagation, resilient backpropagation with (Riedmiller, 1994) or without weight backtracking (Riedmiller, 1993) or the modified globally convergent version by Anastasiadis et al. (2005). The package allows flexible settings through custom-choice of error and activation function. Furthermore, the calculation of generalized weights (Intrator O & Intrator N, 1993) is implemented." This package uses CC in backend.

#### Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
  stepmax = 1e+05, rep = 1, startweights = NULL,
  learningrate.limit = NULL, learningrate.factor = list(minus = 0.5,
  plus = 1.2), learningrate = NULL, lifesign = "none",
  lifesign.step = 1000, algorithm = "rprop+", err.fct = "sse",
  act.fct = "logistic", linear.output = TRUE, exclude = NULL,
  constant.weights = NULL, likelihood = FALSE)
```

Figure 6: Package neuralnet, CRAN

We can see that by default, the resilient backpropagation algorithm is used "rprop+" instead of regular backpropagation which is an algorithm not very used in practice as it is more tricky to implement. However, it brings two main advantages. The first one is that it increases the training speed. The second one is that it doesn't require nor learning rate value neither optimal momentum term to train the model. Instead of having one unique learning rate, rprop+ will have one learning rate for each weight of the network where each weight has a delta value that increases when the gradient doesn't change sign or decreases when the gradient does change sign. However, it does require by default a learning rate limit which specifies the limit where the algorithm stops increasing or decreasing the learning rate, and a learning rate factor that will change the rate according to if the model jumped over a local minima or if it is moving in the right direction.

We can however decided to change the rprop+ for a standard backpropagation by changing for **algorithm = "backprop"**.

By default, the algorithm uses only 1 hidden neuron, it could be more beneficial to increase the number of neurons and hidden layers. To get 5 neurons per 2 hidden layers we could use **hidden = c(5,2)**.

One of the disadvantage of this package is that it requires some data preprocessing as it only works with numeric inputs. Therefore, factor variables will need to be transformed into dummies during the preprocessing phase. Moreover, it is recommended to normalize the data before using them as input in the neural networks in order to reduce the number of iteration of the algorithm [11].

#### Pros

#### Cons

### 4.1.2 nnet

#### Description

**Pros**

**Cons** This package uses c/C++ in backend.

**4.1.3 NeuralNetTools**

for the garson algorithm.

**Description** This package a mostly use ... [4]

**Pros****Cons****4.1.4 RSNNS**

**Description** This package a mostly use ... [12]

**Pros**

**Cons** Another package **RSNNS** (Bergmeir, 2019) allows to have access to a high view of the package SNNS (Stuttgart Neural Network Simulator), containing several neural networks algorithms

4.1.4. à creuser ce package

**4.1.5 Deepnet**

**Description** This package a mostly use ... [10]

**Pros**

**Cons** For deep learning algorithms, **Deepnet** (Rong, 2014), allows to us neural networks with several hidden layers. This package contains interesting features such as neural networks having their neurones' weight automatically initialized by a DBN (deep belief network) or a stacked autoencoder. (

4.1.5. il fait aussi Boltzmann machine, à voir ce que c'est

)

Another package, **RcppDL** (Kou and Sugomori, 2014), allows a simple use of deep neural netowkrsm including denoising autoencoders, stacked denoising autoencoder, restricted Boltzmann machines and deep belief nets.

**4.1.6 Nnet**

**Description** This package allows to quickly train and fit a feed-forward neural networks with one hidden layer [8]. This package does not allow to use more that one hidden layer, and does not have any feature to find the optimal number of neurones in the hidden layer. It is up to the analyst to build a loop to test by cross-validation, for exemple, the optimal hyperparameter values .



**Pros****Cons**

#### 4.1.7 brnn

**Description** This package a mostly use ... [9]**Pros****Cons**

## 4.2 *API!* packages

4.2.0. à voir si on a encore de la place pour mettre un tableau récapitulatif

### 4.2.1 Keras

**Description** This API is really popular in the Python world because it acts like a wrapper of more popular libraries. It might become hard to follow but Keras has multiple backends such as Tensorflow, Theano and MXNET. The *Keras* [2] is mostly use to link the wrapper tool with R. The most useful reference one can get for this workflow is written by two major players ; *Keras* creator and *RStudio* creator. It explains in detail the best practices in order to use *Keras* in R [5]. First step, is the installation and download of the Keras files from GitHub :

```
devtools::install_github("rstudio/keras")
```

Then, we need to to install the package and import in the project :

```
## Error in install_keras(): You should call install_keras() only in a fresh R session that has not
yet initialized Keras and TensorFlow (this is to avoid DLL in use errors during installation)
```

When the "*Installation complete.*" message appear you have a complete installation with CPU configure on the TensorFlow backend. If you want to take advantage of your GPU (Ensure that you have the hardware prerequisites) you will need to execute a different command as follows:

```
install_keras(tensorflow = "gpu")
```

**Pros****Cons**

### 4.2.2 Tensorflow

**Description** This package is use ...[3]**Pros****Cons**

### 4.2.3 MXNET

**Description** This package is use ...[mxnet2020]

**Pros**

**Cons**

### 4.2.4 rTorch

This package is use ...[1]

**Description**

**Pros**

**Cons**

### 4.2.5 H2o

**Description** H2o is a "scalable open source machine learning platform that offers parallelized implementations of many supervised, unsupervised and fully automatic machine learning algorithms on clusters". This package allows to run H2o via its REST API through R and offers several advantages such as the ability ot know the computation time remaining when running a model. Prerequisites to launch H2o, 64 bit Java 6+ if you want to open a h2o model that s more than 1 GB. R users: prerequisite R installed. Add R to your PATH. `install.packages("h2o")` different syntax to work with h2o. `read.csv = h2o.importFile`. `cbind = h2o.cbind`. `predict = h2o.predict`. `glm = h2o.glm`, ... According to KD Nuggets, "Recurrent Neural Networks and Convolutional Neural Networks can be constructed using H2o's deep water project throught others libraries such as Caffe and TensorFlow". "Some of the features of H2o deep learning models are: Automatic Adaptive learning rate, Model regularization, Model checkpointing, Grid Search"

This package is use ...[7]

**Pros**

**Cons**

## 5 Examples

Parler des packages quon décider de tester et comparer.

### 5.1 Installation

Cette section est surtout pour les ressources API, car les ressources R sont tout simplement

5.1.1 Keras

5.2 FNN!

5.2.1 Ressources

```
print("hello world mofo")
```

5.2.2 Compare

5.3 CNN!

5.3.1 Ressources

5.3.2 Compare

5.4 RNN! (RNN!)

5.4.1 Ressources

5.4.2 Compare

## References

- [1] JJ Allaire Alfonso R. Reyes Daniel Falbel. *rTorch: R Bindings to 'PyTorch'*. R package version 0.0.3. 2019. URL: <https://cran.r-project.org/web/packages/rTorch/rTorch.pdf>.
- [2] JJ Allaire and François Chollet. *keras: R Interface to 'Keras'*. R package version 2.2.5.0. 2019. URL: <https://cran.r-project.org/web/packages/keras/keras.pdf>.
- [3] JJ Allaire and Daniel Falbel. *tensorflow: R Interface to 'TensorFlow'*. R package version 2.0.0. 2019. URL: <https://cran.r-project.org/web/packages/tensorflow/tensorflow.pdf>.
- [4] Marcus W. Beck. *NeuralNetTools: Visualization and Analysis Tools for Neural Networks*. R package version 1.5.2. 2018. URL: <https://cran.r-project.org/web/packages/NeuralNetTools/NeuralNetTools.pdf>.
- [5] F. Chollet and J.J. Allaire. *Deep Learning with R*. Manning Publications, 2018. ISBN: 9781617295546. URL: <https://books.google.ca/books?id=xnIRtAEACAAJ>.
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [7] h2o.ai. *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.28.0.4. 2019. URL: <https://cran.r-project.org/web/packages/h2o/h2o.pdf>.
- [8] Brian Ripley and William Venables. *nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models*. R package version 7.3-13. 2020. URL: <https://cran.r-project.org/web/packages/nnet/nnet.pdf>.
- [9] Paulino Perez Rodriguez and Daniel Gianola. *brnn: Bayesian Regularization for Feed-Forward Neural Networks*. R package version 0.8. 2020. URL: <https://cran.r-project.org/web/packages/brnn/brnn.pdf>.
- [10] Xiao Rong. *deepnet: Deep learning toolkit in R*. R package version 0.2. 2015. URL: <https://cran.r-project.org/web/packages/deepnet/index.html>.
- [11] Frauke Guenther Stefan Fritsch and Marvin N. Wright. *neuralnet: Training of Neural Networks*. R package version 1.44.2. 2019. URL: <https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf>.
- [12] SNNS development team. *RSNNS: Neural Networks using the Stuttgart Neural Network Simulator (SNNS)*. R package version n 0.4-12. 2019. URL: <https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf>.