Méthodes avancées en exploitation de donnée
(MATH80619)

Estefan Apablaza-Arancibia	11271806
Adrien Hernandez	11269225

March 3, 2020

List of Acronyms

API Application Programming Interface

CNN Convulutional 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 Litterature Review

2.1 What is Deep Learning and why use it?

Deep learning is a subset field of articial 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

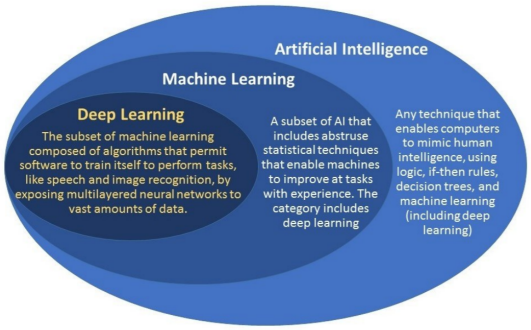


Figure 1: Artificial Intelligence vs Machine Learning vs Deep Learning

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 ([**Goodfellow-et-al-2016**]).

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." [**Goodfellow-et-al-2016**]

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

output layer. It is important to know that in a feedfoward 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

$$a(x) = b + \sum_i w_i x_i \tag{1}$$

and the output

$$h(x) = g(a(x)) \tag{2}$$

where:

x = the input data
b = the bias term
w = the weight or parameter
$g(...)$ = the activation function

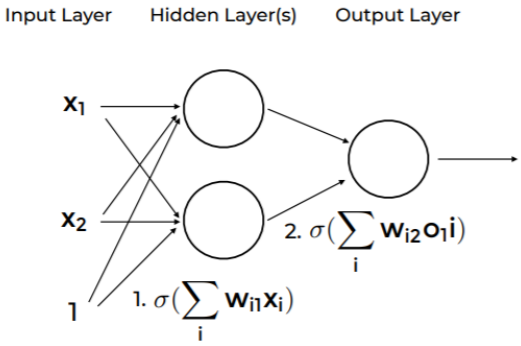


Figure 2: feedforward neural networks

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(...)**, 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)	$\textbf{sigmoid}(z) = \frac{1}{1+\exp(-z)}$	Bernoulli
Categorical (0,1,2,3,k)	$\textbf{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{i'} \exp(z_{i'})}$	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 [**Goodfellow-et-al-2016**]

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 perfom 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 fonction 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.

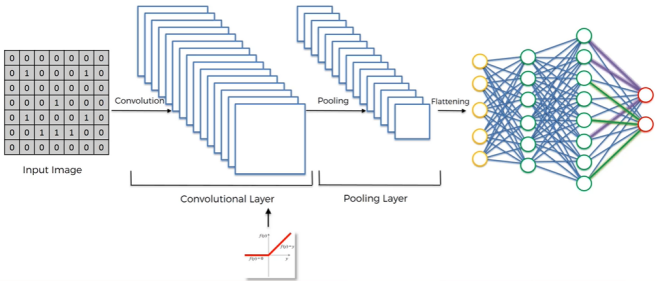
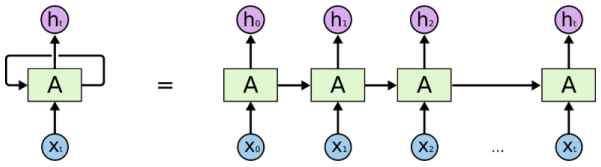


Figure 4: CNN, Deep Learning A-Z by SuperDataScience

To get a detailed explanation of convolutional neural networks we recommand to read [**Goodfellow-et-al-2016**].

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 feedfoward 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 sequantial data as well as what they have learned over time. One of their other advantage is to be able to handle input and ouput 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 recommand to read [**Goodfellow-et-al-2016**]. We also recommand reading this blog post `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

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 restraing in terms deep learning architecture.

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

3 Description of methods

3.0.0. I think this section is more to explain, for example, the dataset with their end objective so when we are doing examples we just say we are going to use the CIFAR10 and the readers know we are doing a classification problem.

4 Methodology

4.0.0. Give example of deep learning project pocedure

5 Available resources

5.0.0. Très intéressant:`https://edu.kpfu.ru/pluginfile.php/419285/mod_resource/content/2/neural-networks-r.pdf`

5.0.0. Faire une recherche exhaustive des ressources disponibles en R pour faire des analyses liées à ce sujet. Par exemple: fonctions R de base, packages sur les différentes plate-formes (CRAN,)

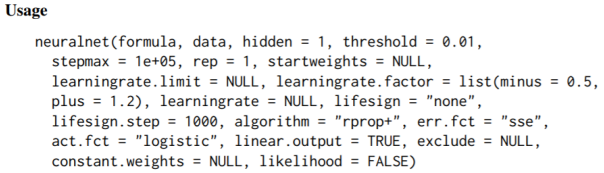
According to the CRAN-R project website, we give a list of the different packages known in R allowing to use simple and deep neural network algorithms.
`https://cran.r-project.org/web/views/MachineLearning.html`

5.1 R packages

5.1.1 Neuralnet

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 C/C++ in backend.

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 therm 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"



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

`https://jamesmccaffrey.wordpress.com/2017/08/01/neural-network-resilient-back-propagation/`. 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 benefitial 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 iretation of the algorithm. part of CRAN, `https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf`

5.1.2 nnet

This package uses c/C++ in backend.

5.1.3 NeuralNetTools

for the garson algorithm.

5.1.4 RSNNS

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

5.1.4. à creuser ce package

5.1.5 Deepnet

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. (

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

)

Package	Pro	Con	Actif	URL
tensorflow R				Link
Kera R				
MxNet				

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.

5.1.5. Faire une recherche exhaustive des ressources disponibles en R pour faire des analyses liées à ce sujet. Par exemple: fonctions R de base, packages sur les différentes plate-formes (CRAN,)

According to the CRAN-R project website, we give a list of the different packages known in R allowing to use simple and deep neural network algorithms.
`https://cran.r-project.org/web/views/MachineLearning.html`

5.1.6 Nnet

(Ripley and Venables, 2016) allows to quickly train and fit a feed-forward neural networks with one hidden layer. 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.`https://cran.r-project.org/web/packages/nnet/nnet.pdf`

5.1.7 brnn

5.1.8 deepnlp

5.1.8. Ce mec parle de la gestion de la memoire en R avec keras, qui me semble utilise des numpy array en back-end ... à creuser mais peut etre interessant d'en parler car le gros probleme de R pour le deep learning c'est sa gestion de la memoire

`https://community.rstudio.com/t/deep-learning-in-r-memory-allocation/17541`

5.1.8. Bonne ressource pour videos `https://www.youtube.com/user/westlandindia/videos`

5.1.8. Ici on pourrait faire un tableau avec tous les packages et donner les pour et les contres, ADRIEN: Le tableau je suis pas trop fan sachant qu on doit ecrire 20 pages sur ces librairies

5.2 API! packages

5.2.1 Keras

How to install it

5.2.1. Très très intéressant:`https://r2018-rennes.sciencesconf.org/data/pages/Deep_with_R_1.pdf`

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 :

```
library(keras)
install_keras()
```

```
##
## Installation complete.
```

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")
```

Tensorflow backend

CNTK backend

Theano backend

5.2.2 Tensorlow

5.2.3 rTorch

5.2.4 H2o

(LeDell, 2020) video qui explique h2o 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". Source: $a(x) = b + \sum_i w_i x_i$

6 Examples

6.0.0. Encore à voiron pourrait créer des tutoriels pour peut-être comparer

6.0.0. Il faut trouver des datasets intéressantes à utiliser avec les differents packages

6.0.1 neuralnet package

```
#Download and install the package
install.packages("neuralnet")
```

```
## Installing package into 'C:/Users/adrie/OneDrive/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## Error in contrib.url(repos, "source"):  trying to use CRAN without setting a mirror
```

```
library(neuralnet)
```

Let's practice with the Fisher's Iris dataset. We will start by splitting the data into a training test and a test set.

```
# Split data into training and test set
train_idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris_train <- iris[train_idx, ]
iris_test <- iris[-train_idx, ]
```

Let's build our model for a multiclass classification and predict.

```
# Multiclass classification
nn <- neuralnet((Species == "setosa") + (Species == "versicolor") + (Species == "virginica")
                ~ Petal.Length + Petal.Width, iris_train, hidden = c(3,2), linear.output =FALSE)
pred <- predict(nn, iris_test)
table(iris_test$Species, apply(pred, 1, which.max))
```

```
##
##                1  2  3
##    setosa     15  0  0
##    versicolor  0 15  1
##    virginica   0  2 17
```

Let's plot our model:

```
# Multiclass classification
plot(nn)
```

6.0.1. Find a way to remove page number from references