Méthodes avancées en exploitation de donnée
(MATH80619)

Estefan Apablaza-Arancibia 11271806
Adrien Hernandez 11269225

March 17, 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

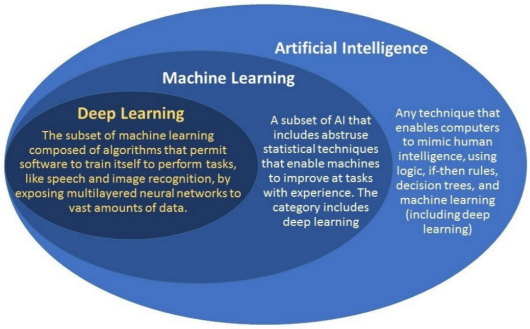


Figure 1: Artificial Intelligence vs Machine Learning vs Deep Learning

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**]

Feed-Forward Neural Network (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. 4) 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.

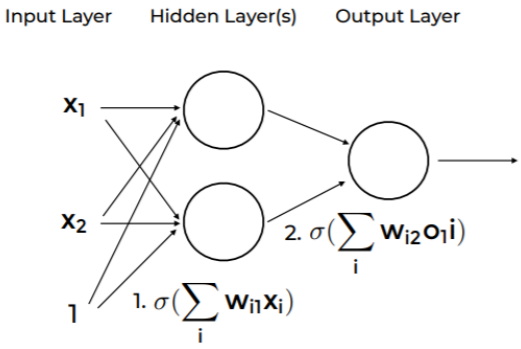


Figure 2: feedforward neural networks

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

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:

A detailed explanation of the theory of feedforward neural network can be found in [**Goodfellow-et-al-2016**]

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

2.1.2 Convulutional Neural Network (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.

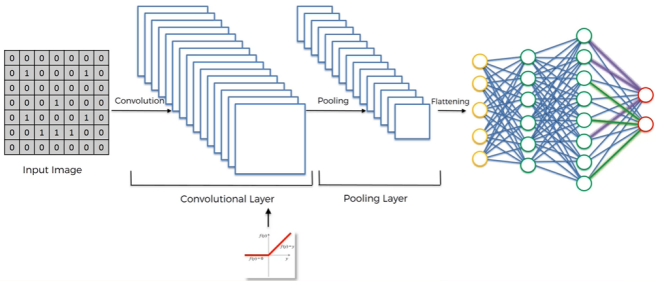


Figure 4: CNN, Deep Learning A-Z by SuperDataScience

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.

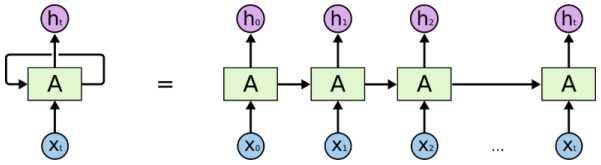


Figure 5: RNN, Understanding LSTM Networks by Olah

To get a detailed explanation of the theory of recurrent neural networks we recommand to read [**Goodfellow-et-al-201**
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 Application Programming Interface (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

4.1 Dataset

4.1.1 FNN

4.1.2 CNN

4.1.3 RNN

4.2 Benchmark

4.2.1 Accuracy

4.2.2 Time elapsed

5 Available resources

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

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

http://www.parallelr.com/r-deep-neural-network-from-scratch/ https://edu.kpfu.ru/pluginfile.php/419285/mod_resource/content/2/neural-networks-r.pdf

5.1.1 nnet package

Keyword: FNN

Description This package comes from the CRAN platform and allows to build and fit "FNN with a single hidden layer and multinomial log-linear models". This package uses C/C++ in backend.

```
nnet(x, y, weights, size, Wts, mask,
     linout = FALSE, entropy = FALSE, softmax = FALSE,
     censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
     maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
     abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Figure 6: Package nnet, CRAN

Pros

- One of the easiest R package to build a quick FFN model.

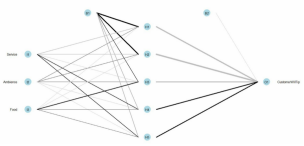
Cons

- It does not offer a lot of flexibility, and can only apply logistic sifmoid function for the hidden layer activation and cannot use tanH and ReLu.
- 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.
- Cannot use classical backpropagation algorithm to train the network. It only uses the BFGS (Broyden-Fletcher-Goldfarb-Shanno) which is a Quasi-Newton method and therefore increases the number of computation to reach a local optima. However, applied on a small dataset with a model having only one hidden layer does not seem to be a problem.
- Does not have any function to plot the model.

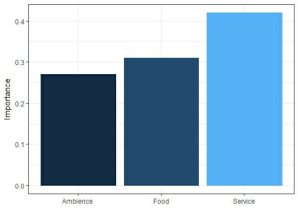
5.1.2 NeuralNetTools package

Keywords: Visualization; Variable importance; Sensitivity

Description This package is a complement to R's neural networks packages as it allows to visualize and perform analysis on neural network models. "Functions are available for plooting, quantifying variable importance, conducting a sensitivity analysis, and obtaining a simple list of model weights."



(a) NeuralNetTools, plotnet function



(b) NeuralNetToolsm Input variables importance

Figure 7: Embedding dimension hyper parameter

Pros

- Very easy to plot your neural network models with the function **plotnet(model)**.
- Visualize the input variables importance to the output prediction witht the Garson and olden algorithm.
- Perform sensitivity analysis on your neural networks model using the Lek profile method.
- Works with the models built from different packages: **caret, neuralnet, nnet, RSNSS**
- Can plot neural networks with pruned connections from the RSNSS package.

Cons

- Does not provide any function for neural network model development.
- Is not optimized to visualize large neural networks models.

5.1.3 Neuralnet package

Keyword: FNN; Deep FNN

Description According to its CRAN description, the package allows the "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$/Cinbackend$.

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 8: Package neuralnet, CRAN

Pros

- Very easy to use and to build a quick FFN and deep FNN model.
- Allows to use several types of backpropagation algorithms. 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. We can however decided to change the rprop+ for a standard backpropagation by changing for **algorithm = "backprop"**.
- Several hidden layers and neurons per layer can be added. By default, the algorithm uses only 1 hidden layer with 1 neuron, but it can be increased by addind the command line: **hidden = c(5,2)** to get 2 hidden layers of 5 neurons.
- Allows to easily plot and visualize the model and its parameters with the command line: **plot(model).**

Cons

- 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.

- The package doesn't provide built-in normalization functions. Therefore, it is recommended to manually normalize the data before using them as input in the neural networks in order to reduce the number of iretation of the algorithm.

5.1.4 Deepnet package

Keywords: FNN; RBM; DBN; Autoencoder

Description This package is available on the CRAN platform and has been written specifcally written for R. It allows "implement som deep learning architectures and neural network algorithms, including backpropagation, Restricted Boltzmann Machine (RBM), Deep Belief Network (DBN) and Deep autoencoder".

```
nn.train(x, y, initW = NULL, initB = NULL, hidden = c(10), activationfun = "sigm",
    learningrate = 0.8, momentum = 0.5, learningrate_scale = 1, output = "sigm",
    numepochs = 3, batchsize = 100, hidden_dropout = 0, visible_dropout = 0)
```

Figure 9: Package deepnet, CRAN

Pros

- Allows to load benchmark datasets such as MNIST with the function load.mnist(dir)
- Allows to initialize the weights of a FNN with the Deep Belief Network algorithm (dbn.dnn.train()) or Stacked AutoEncoder algorithm (sae.dnn.train()).
- Allows to have an estimation of the probabilistic distribution of a dataset trough the use of the Restricted Boltzmann machine algorithm.
- Have more activation functions than the other R packages. Hidden Layers activation function can be linear, sigmoid or tanh. The output activation function can be linar, sigmoid or softmax.

Cons

- Does not support the ReLu activation function for hidden layers.
- Not the fastest package due to its implementation in R.
- Does not provide a lot of hyperparameters tuning options, and is not user-friendly.

5.1.5 brnn package

Keywords: Bayesian Regularization; FNN

Description This package available on the platform CRAN, allows to perform "Baeysian regularized neural networks inclusing both additive and dominance effects, and allows to takes advantage of multicore architectures via a parallel computing approach using openMP for the computations". (Pérez-Rodriguez, ...)

```
nnet(x, y, weights, size, Wts, mask,
     linout = FALSE, entropy = FALSE, softmax = FALSE,
     censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
     maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
     abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Figure 10: Package nnet, CRAN

Pros

- Allows to add Additive and Dominance effects in the neural networks models.
- Take advantage of multicore processors (only for UNIX-like systems).
- Allows to use a Gauss-Newton algorithm for optimization.
- The package is able to assign initial weights by using the Nguyen and Widrow algorithm.
- Include an algorithm to deal with ordinal data by using the function **brnn_ordinal(x, ...)**
- It has a function to normalize and unnormalized the data.

Cons

- Cannot use the classic backpropagation algorithm for optimization.
- The package fits a two layers neural networks and it is not possible to increase the number of hidden layers.

5.1.6 rnn package

Keywords: RNN Installation: You will need to install the "diges" package first through the function **install.packages("digest")** in order to download the rnn package.

Description Avaible from the CRAN platform, this package allows the "implementation of a Recurrent Neural Network architectures in native R, including Long-Short Term Memory (LSTM), Gated Recurrent Unit (GRU) and vanilla RNN.

```
trainr(Y, X, model = NULL, learningrate, learningrate_decay = 1,
  momentum = 0, hidden_dim = c(10), network_type = "rnn",
  numepochs = 1, sigmoid = c("logistic", "Gompertz", "tanh"),
  use_bias = F, batch_size = 1, seq_to_seq_unsync = F,
  update_rule = "sgd", epoch_function = c(epoch_print,
  epoch_annealing), loss_function = loss_L1, ...)
```

Figure 11: Package rnn, CRAN

Pros

- Allows to run a demonstration of how RNN models work by using predefined values and allowing the user to see the impact of a changes in the model's hyperparameters by using the function run.rnn_demo()
- Allows to plot the model's errors through all epochs.
- Allows to use LSTM and GRU models.

Cons

- Uses R native, therefore it might not be the best package in terms of time computation.

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

5.2 API packages

5.2.1 Keras

Description

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 :

```
install.packages("devtools")
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

Pros

Cons

5.2.2 Tensorflow

Description

Pros

Cons `https://srdas.github.io/DLBook/DeepLearningWithR.html#using-tensorflow`

5.2.3 MXNET

Description

Pros

- Allows the use of multiple CPU and multiple GPU, but it requires to download the package Rtools and a c++ compiler.
- Very easy to use and has a flexible implementation of diffenent neural networks architectures and models.
- The models can be built layer per layer.
- Provide details and information about the learning progress during the training phase.

Cons `https://srdas.github.io/DLBook/DeepLearningWithR.html#using-mxnet`

5.2.4 rTorch

5.2.5 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.

```
dl_fit3 <- h2o.deeplearning(x = x,
                            y = y,
                            training_frame = train,
                            model_id = "dl_fit3",
                            epochs = 50,
                            hidden = c(20,20),
                            nfolds = 3,                            #used for early stopping
                            score_interval = 1,                    #used for early stopping
                            stopping_rounds = 5,                   #used for early stopping
                            stopping_metric = "misclassification", #used for early stopping
                            stopping_tolerance = 1e-3,             #used for early stopping
                            seed = 1)
```

Figure 12: H2o package

```
install.packages("h2o")
library(h2o)
```

How to install it **Prerequisites to launch H2o**, 64 bit Java 6+ if you want to open a h2o model that s more than 1 GB.

Pros

- Very easy to use and includes a cross-validation feature and functions for grid search in order to optimize the model's hyperparameters.
-
- Allows the use of multiple CPU.
- Provide an adaptive learning rate (ADADELTA) which improves the optimization process by having a different learning rate for each neuron.
- Provide details and information about the learning progress during the training phase.

Cons

- Requires the latest version of Java.

6 Examples

Parler des packages quon décider de tester et comparer.

6.1 Installation

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

6.1.1 Keras

6.2 FNN

6.2.1 Ressources

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

6.2.2 Compare

6.3 CNN

6.3.1 Ressources

6.3.2 Compare

6.4 Recurent Neural Network (RNN)

6.4.1 Ressources

6.4.2 Compare

6.4.2. Find a way to remove page number from references