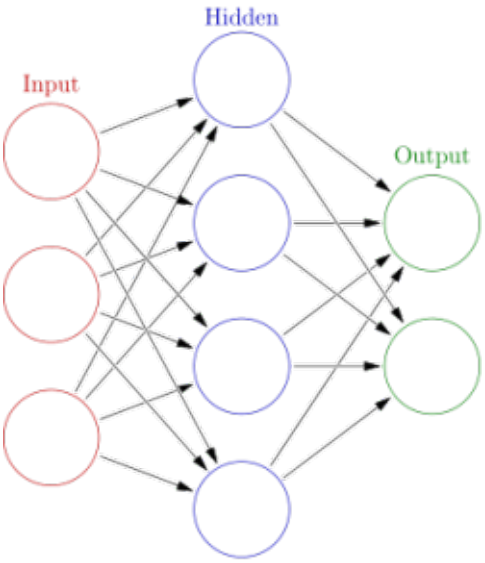
Neural networks Exercises (Part-1)



Source: Wikipedia

Neural network have become a corner stone of machine learning in the last decade. Created in the late 1940s with the intention to create computer programs who mimics the way neurons process information, those kinds of algorithm have long been believe to be only an academic curiosity, deprived of practical use since they require a lot of processing power and other machine learning algorithm outperform them. However since the mid 2000s, the creation of new neural network types and techniques, couple with the increase availability of fast computers made the neural network a powerful tool that every data analysts or programmer must know.

In this series of articles, we'll see how to fit a neural network with R, we'll learn the core concepts we need to know to well apply those algorithms and how to evaluate if our model is appropriate to use in production. This set of exercises is an introduction to neural networks where we'll use them to create two simple regression and clustering model. Doing so, we'll use a lot of basic concepts we'll explore

further in future sets. If you want more informations about neural network, your can see this page.

Answers to the exercises are available here.

Exercise 1
We'll start by creating the data set on which we want to do a simple regression. Set the seed to 42, generate 200 random points between -10 and 10 and store them in a vector named X. Then, create a vector named Y containing the value of sin(x). Neural network are a lot more flexible than most regression algorithms and can fit complex function with ease. The biggest challenge is to find the appropriate network function appropriate to the situation.

Exercise 2
A network function is made of three components: the network of neurons, the weight of each connection between neuron and the activation function of each neuron. For this example, we'll use a feed-forward neural network and the logistic activation which are the defaults for the package nnet. We take one number as input of our neural network and we want one number as the output so the size of the input and output layer are both of one. For the hidden layer, we'll start with three neurons. It's good practice to randomize the initial weights, so create a vector of 10 random values, picked in the interval [-1,1].

Exercise 3
Neural networks have a strong tendency of overfitting your data, meaning they become really good at describing the relationship between the values in your data set, but are not effective with data that wasn't used to train your model. As a consequence, we need to cross-validate our model. Set the seed to 42, then create a training set containing 75% of the values in your initial data set and a test set containing the rest of your data.

Exercise 4

Load the nnet package and use the function of the same name to create your model. Pass your weights via the Wts argument and set the maxit argument to 50. We want to fit a function which can have for output multiple possible values. To do so, set the linout argument to true. Finally, take the time to look at the structure of your model.

Learn more about neural networks in the online course Machine Learning A-Z™: Hands-On Python & R In Data Science. In this course you will learn how to:

- Work with Deep Learning networks and related packagse in R
- Create Natural Language Processing models
- And much more

Exercise 5

Predict the output for the test set and compute the RMSE of your predictions. Plot the function sin(x) and then plot your predictions.

Exercise 6

The number of neurons in the hidden layer, as well as the number of hidden layer used, has a great influence on the effectiveness of your model. Repeat the exercises three to five, but this time use a hidden layer with seven neurons and initiate randomly 22 weights.

Exercise 7

Now let us use neural networks to solve a classification problem, so let’s load the iris data set! It is good practice to normalize your input data to uniformize the behavior of your model over different range of value and have a faster training. Normalize each factor so that they have a mean of zero and a standard deviation of 1, then create your train and test set.

Exercise 8
Use the nnet() and use a hidden layer of ten neurons to create your model. We want to fit a function which have a finite amount of value as output. To do so, set the linout argument to true. Look at the structure of your model. With classification problem, the output is usually a factor that is coded as multiple dummy variables, instead of a single numeric value. As a consequence, the output layer have as one less neuron than the number of levels of the output factor.

Exercise 9
Make prediction with the values of the test set.

Exercise 10
Create the confusion table of your prediction and compute the accuracy of the model.