Implementation of a neural network in C++ programming language using Armadillo library

Michał Hamuda, Paweł Nowak, Mateusz Windak

Abstract

In this article, we present our own implementation of a simple neural network for a classification problem using open source library for matrix calculations - Armadillo. The goals of this article are the following: firstly, to gain understanding what the neural networks really are - many people think of them as some mystcal, omnipotent, brain-like black-boxes, when in fact we show that a neural network in a basic form is just a series of matrices and a series of operations defined on them. Secondly, we want to examine if Armadillo library, and C++ language in general, can be a useful tool for machine-learning related programming. Python and R had became de facto industry standard in that field, but they both have their limitations, so it is worthwhile to seek for alternatives. And, last but not least, our third goal when writing this article is to obtain 5 points needed by us to pass the famous MRO course, the last course standing between us and the engineer's degree.

Keywords: `Armadillo`, C++, neural network, MRO

1. The problem

The problem that our network attempts to solve is the famous classification problem. In this problem, we have an n-dimensional space where each point belongs to one of the k classes. We do not have full knowledge about class membership - in fact we have a training set T containing finite number of points, and only for the points from T we know which class they belong to. What this problem wants us to do is to construct a program that would take as an input x and T, where T is the training set, and x is any point from the space, and as an output determine to which class does the x belong.

Such definition might sound very dry and mathematical, like the jargon that the scientists use to be perceived as more intelligent than everyone else, but let's show it on a real world example: election statistics. Suppose we have a two-dimesional space representing people. One dimension is the age, second dimension is the income level. Our classes will be political preferences - one class will be people supporting party A and the second class will be people supporting party B, because for some reason, we believe that age and income level is correlated with political preferences, so people of certain age and certain incomes will support party A, and people of different age and income will support party B. We believe that such correlation exists, but we don't know how does it exactly look like - in fact all we know is some set of people T, that we have actually reached out and asked such indiscrete questions like how old are you, how much do you earn and who do you vote for. What we want to do is to take the information from the set T and build a program that for any combination of age and income would tell us on which side of the barricade this combination is. We want to be able to create plot as in Figure 1

2. The solution

How do we want to tackle that problem? We are going to use machine learning. First, we will come up with a function with some parameters and we will assume that this function can approximate the output we want to get, and using the training set we will try to find optimal values of the parameters

- optimal in the sence that with them function gives the best approximation.

So, what will be our parametrized function? Let us summon a mathematical abstraction called neural network. Basic building block of a neural network is a perceptron - a function that takes a vector as an input and produces a single scalar as an output. Input of one perceptron can consist of outputs of other perceptrons - therefore perceptrons can be connected forming a directed graph called *the net*. The net is usually aligned into layers - input layer, one or more hidden layers, and output layer, as shown in Figure 2. In such aligment, we can treat the whole *layer* as a function that takes a matrix as an input and produces some other matrix as an output, and output of one layer becomes input of another. How does the perceptron - and therefore the layer compute output? Perceptron has a vector of weights W a free term b, and an arbitrary *activation function* s needed to introduce nonlinearity. It takes input vector x, computes scalar product with W, adds the free term b, and passes that through the activation function. Generalizing, the *layer* has a *matrix* of weights W and a *vector* of free terms b and the activation function. So, the formula for the output is the following:

$$y = s(Wx + b)$$

In our case - the classification problem - the input of the first layer will be n-dimensional vector of the point coordinates, and the output of the last layer will be k-dimensional vector where the i-th element will be the probability that the point belongs to the i-th class. The acitivation function is chosen arbitrarily - we chose *tanh* - and W and b matrices of each layer are the parameters whose values we want to find. We find them by first assigning random values to them and then iteratively trying to improve them. What follows is a philosophical question: what does it mean to improve something? What does it mean to improve parameters? Is improvement something that can be measured or is it a matter of subjective feeling, perceived differently by every person? It turns out that in case of parameters, it can be measured. To do that, we introduce *loss function* that will take as an input a points with known classifications and our whole network and will produce a single number saying how well does our net perform on this set of points, i.e. what is the percentage of points for which our net incorrectly guesses class membership. The-

refore, when we iteratively try to improve parameters, we want to find new parameters for which the loss function of the net will be lower than for the previous ones. For that we use method called *gradient descent* - we compute gradients of the loss function with respect to the parameters ($\frac{\partial L}{\partial W_1}$, $\frac{\partial L}{\partial b_1}$ and so on) and we 'move' the parameters in direction of those gradients therefore iteratively approaching the local minimum of the loss function. That of course means that depending on how we (randomly) choose initial values of the parameters we can get stuck in the local minimum and never achieve the global minimum that can be just few steps away. But well, that's life, nobody said it will be all pink and roses.
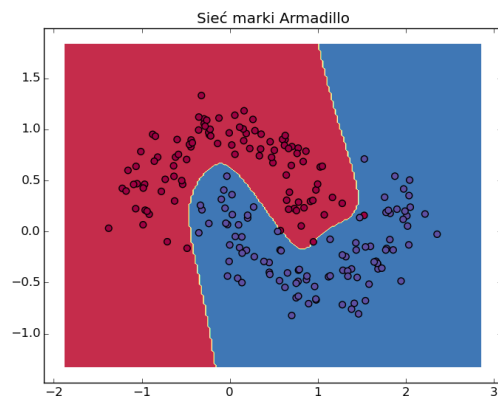
3. The implementation

One of the advantages of the C++ language is its strong type system. It makes it much easier working with legacy code or with a library previously unknown to the programmer, because types alone can give a lot of information about what a given method or class is actually doing. We wanted to follow that path and we wanted our implementation to have types reflecting the mathematical model we described above. Armadillo provided us with classes for matrix and vector - *mat* and *rowvec*, what we have to do was to implement classes representing layer of the network and the whole network. (Actually we could also model perceptron as a separated class, but we decided not to do it as it would be too detalistic and would make things more complex and not more simple). Diagram of our classes with their interfaces can be seen on Figure 1 We designed the model in such way that it can support any size of input and output vector and any number of layers, however we used it only for input and output vectors of size two and three layers. It was proven that there is no point to further increase the number of layers, because any multi-layer network can be simulated on a three-layer one. (Except for a more advanced type of networks, called *convolutional neural networks*, but that is a whole different story).
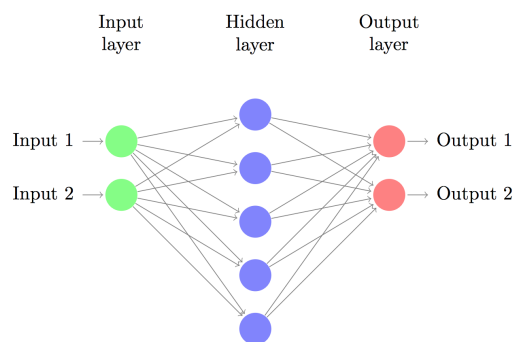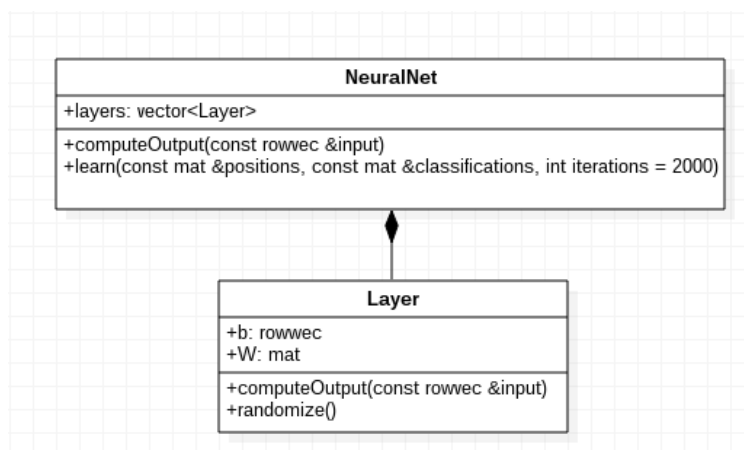
4. The results

Literatura

[1] Armadillo homepage http://arma.sourceforge.net
[2] Stack overflow

Rysunek 1: Political preferences - colors represent parties, points form the set T

Rysunek 2: Schema of the neural network

Rysunek 3: UML diagram of our implementation (library classes not shown)