Lab 1 – Optimization of a Neural Network

Krishna Kalyan

29/10/2015

1 Description

In machine learning and cognitive science, artificial neural networks (ANNs) are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. Described below is the approach and the tools used the solve this assignment.

2 Julia

Julia is a high-level dynamic programming language designed to address the requirements of high-performance numerical and scientific computing while also being effective for general-purpose programming, web use or as a specification language.
JuliaOpt brings together packages written in Julia that are related to optimization. For this assignment we will be using LsqFit package which is a small library that provides basic least-squares fitting in pure Julia. LsqFit uses Levenberg-Marquardt algorithm for non-linear fitting.
IJulia is a Julia-language backend combined with the Jupyter interactive environment (also used by IPython). This combination allows you to interact with the Julia language using Jupyter/IPython's powerful graphical notebook, which combines code, formatted text, math, and multimedia in a single document.

2.1 Forward Propagation

We will be designing a generic feed forward function that will be used to learn the weights from the optimiser as well as a function to test As shown in equation 1.1 we use a generic function forward that will be used as a

model for the least square optimiser. The first few lines define Activation function or the Sigmoid function. Sigmoid functions are always real-valued, continuous , differentiable and are also known as universal approximators. They introduce non linearity in the network.

```
function forward(X, W)
    f(x) = map(b -> act(b) ,x)
    act(x) = (1/(1+exp(-x)))
    return f(f(f(X)*reshape(W[1:8],4,2))*W[9:10])
end
```

The last few line express in the code above describe a Neural Network defined in the problem statement. In this particular network, the lower layer (with four nodes) receives four input values (x_i , $i = 1, \ldots, 4$), while the output layer, with only one neuron, gives us the result of the network.

$$\begin{aligned} F(x_1, x_2, x_3, x_4) &= O_7 = f(I_7) = f(w_{5,7}O_5 + w_{6,7}O_6) = f(w_{5,7}f(I_5) + w_{6,7}f(I_6)) = \\ &= f(w_{5,7}f(\textstyle\sum_{i=1}^{4} w_{i,5}O_i) + w_{6,7}f(\sum_{i=1}^{4} w_{i,6}O_i)) = \\ &= f(w_{5,7}f(\textstyle\sum_{i=1}^{4} w_{i,5}f(x_i)) + w_{6,7}f(\sum_{i=1}^{4} w_{i,6}f(x_i))), \end{aligned} \tag{1.1}$$

2.2 Non-Linear Optimisation

Non-linear least squares is the form of least squares analysis used to fit a set of m observations with a model that is non-linear in n unknown parameters (m > n). It is used in some forms of non-linear regression.

LsqFit.jl library solves non-linear least square problems as given by the equation 1.2 below.

$$\min_{w} \sum_{i=1}^{p} ||F(x_{e_i}; w) - x_{o_i}||^2 \tag{1.2}$$

The least square equation above is modelled using the high level function called "curve_fit(model, x, y, w)". Where our model is defined by forward function, x as input data, y as ground truth label, w as random weights between -1 and +1 applied to the network (In this case we have initialised an array with 10 random weights).
This performs a fit using a non-linear iteration to minimise the (weighted) residual between the model and the dependent variable data (y).
Finally we compute the correct rate by comparing predicted values with the ground truth. Which is basically the trace of the confusion matrix divided by total number of elements in y.