Project I

CS5102: Deep Learning

Due date: 12th Oct 2018

Problem 1:

The objective of this part of the project is to implement our first draft of a Multilayer Perceptron trained using the back-propagation gradient descent, and test it on a working example.

Parts (a - e) deal with step by step procedure to develop this implementation.

(a): Write MATLAB code for a Neural Network with n_i inputs, n_o outputs, one input and output layer with N_h hidden layers and n_h nodes in each hidden layer.

> 1: Start by writing code for sigmoid activation(Sigm.m), derivative of sigmoid(DSigm.m), δ for output layer(DeltaO.m), δ for hidden layer(DeltaH.m)
>
> 2: Perform a feedforward pass, computing the activations for each layer until you calculate the output.
>
> 3: Calculate δ for output layer using $\delta = o(1-o)(y-o)$ and its weight updates.
>
> 4: Back-propagate δ to calculate δ for hidden units using $\delta_j = o_j(1-o_j)\sum \delta_k w_{kj}$, then compute corresponding weight updates.
>
> You can apply this algorithm for a fixed number of epochs, or until the error function converges.

(b): Apply your MATLAB code developed in (a) to the example solved in class.

This example was taken from and is available in step-by-step, solved form here.

Compute weight updates for each layer and confirm that your code gives the correct values.

(c): Apply your code to the Housing regression data set This data relates house prices with a set of variables. More detail about the data can be found here.

(d): Begin by having only 1 hidden layer with 1 node, and try to approximate the function. Indicate your training-test split and report your RMSE on each. Make any changes to the architecture of NN (e.g add more units in hidden layer) to construct a model suitable for data. You can also record the effect of adding momentum or changing optimization strategy on error convergence.

(e) Change the number of hidden units/layers in such a way that your model underfits and overfits the data. Describe how you can prove using training/test error plots that your model underfits/overfits in each case.

Best of luck!

Implementation Notes

1: You can implement this project in MATLAB. If you have no background in MATLAB, you can start with this tutorial. You can also use any other programming language to code, as long as you do not use built-in functions for back-propagation or other algorithms.

2: You can seek help from each other, use MATLAB code developed by other people(Mathworks maintains a huge File Exchange repository of user contributed code) to correct and modify yours or even use parts of their code in yours as long as you understand it. However, you MAY NOT copy the entire code from else where and you MUST give credit to the original developer of the code if you are using it in parts.

3: Search online material if you run into any issues. If you can't make your neural network(or any other system) to work at all, applying a trivial input is sometimes a good strategy.

4: Keep your code flexible. Use varargin and varargout where needed. Use MATLAB's debugging to your advantage. Visualize your neural network (there are many such user-contributed files available on MATLAB file exchange that draw your NN for you) to aid your understanding.
[1]

[1]