Due: Mon April 15, 2022

Dubbler ANN Regression Model

ANN models can be of two types: Classification or Regression. We covered the Regression-style ANN models in class -- models that could simulate another function via a regression/fitting process. This assignment requires you to build a *regression-style* -- not a classification-style -- ANN model.

Build a single-input regression-style ANN models capable of approximating the following function:

1. $f(x) = 2x$ $\quad\quad\quad\quad$ $x \in [-5 \ldots +5]$

In the programming language of your choice, implement classes that will allow you to create functional neural network models with non-linear activation functions and a *single* hidden-layer.

You may choose the number of hidden nodes in your hidden (middle) layer for your model.

For each model:

- generate two disjoint sets of valid inputs (say 500 for each of *training* and *testing* sets) from the continuous input-range provided. Also generate a _*target* set of correct matching outputs for each item in your training set

- Use just your raw input/output values at first. **Once you get your model working with raw numbers,** independently *normalize* your training_input and your training_output raw_values to normalized values between [0 … +1]. Also normalize your testing_input set (and the matching testing_output set)

  Functions: $$Norm(x_i) = \frac{x_i - MIN_i}{MAX_i - MIN_i}$$

  $$UnNorm(x_o) = x_o * (MAX_o - MIN_o) + MIN_o$$

  Explain in your report what difference (if any!) normalizing your inputs made, such as in terms of how much more quickly the model trained, for instance.

- Build your network from a set of classes (e.g. *InputNode, HiddenNode, OutputNode, BiasNode*)

- I suggest using the sigmoid act_fcts (shown in class) in all hidden nodes ~~and also output nodes~~. You can leave linear activation functions for the output nodes.
    - (Alternately, you could use ***tanh()*** as activation functions instead, but for now let's just stick with the traditional sigmoid**)**.
- Use bias nodes in each non-output layer

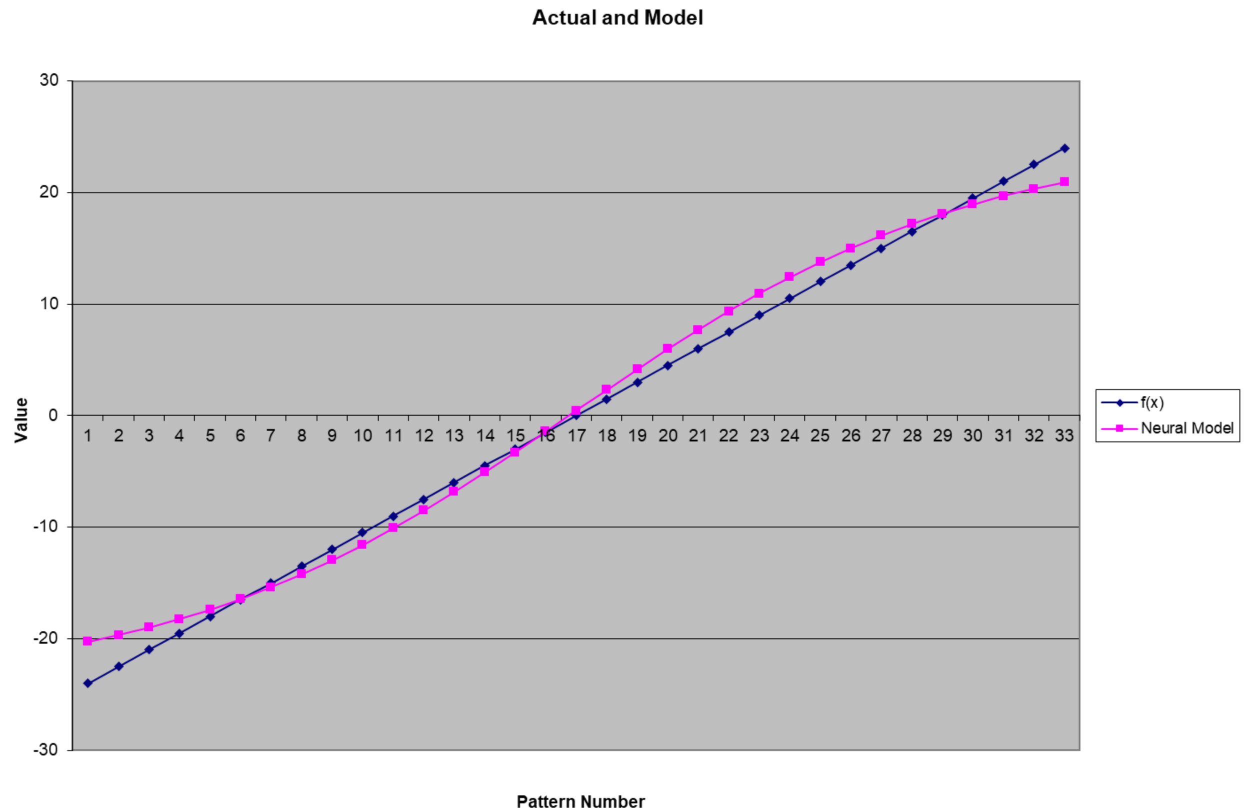
- Train your network
    - by: presenting an input to the network, feed it forward, determine the error in the output, determine the delta-value of each output/hidden node, and then determine and make the change to be made to each weight leading *in* to each node;
    - and repeat for each input

    - until either: a) the RMSE of the set of outputs on the training set is less than 10^{-1}, or b) you have made ~10,000 or so passes (i.e. "epochs") through your training set

- Determine the RMSE of your model on the testing set (if you have normalized your inputs/outputs, remember to de-normalize your test-outputs to get real-world RMSEs!)
    - build me a graph of your RMSE values at the end of every 10-100 epochs or so (i.e. a pass through all training-set instances equals one epoch)

    (Tip: Your RMSE should drop by some amount with each epoch of training. If your RMSE thrashes repeatedly between ever-increasing negative-error and positive-error, reducing your Learning-Rate by an order of magnitude (or more) might help)

- Try to determine the fewest hidden nodes required to still achieve acceptable RMSE accuracy
- Generate a graph of your results. In the same graph, display: the target function, and the approximated function *(example follows)*



- Draw a picture of the final, trained models, including the weights assigned to the connections (to three significant digits)

You may choose any programming language you like for this implementation.
You may *not* use WEKA or Excel worksheets for this implementation.

Deliverable:

A written report (~2-5 pages) explaining your approach and your process, as well as demonstrating your results, to be turned in by the due date.

Include snippets of your source code in your report; and the entirety of your ANN code as an Appendix to your report. Turn in your report to the D2L assignment folder.