**Due: Tues April 30, 2024**
**10:30am - 12:30pm**

# Fancier ANN Regression Models

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* -- not a classification -- -style ANN model.

Build regression-style ANN models capable of approximating the following functions:

1. $f(x) = 2x$ $\quad x \in [-5 \ldots +5]$
2. $f(x) = \sin(2x)$ $\quad x \in [0 \ldots 4\pi]$
3. $f(x, y) = xy$ $\quad x, y \in [-5 \ldots +5]$

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 valid input-range provided for each. Also generate a *_target* set of correct matching outputs for each item in your training set

- If training your network to raw numbers works, try the following: independently *normalize* your training_input and your training_output matrices to values between [0 … +1]. Also normalize your testing_input set (and the matching testing_output set)

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

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

- 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 and also output nodes (Alternately, you could use ***tanh()***, or the ***ReLU()*** function).
    - 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!)
    - graph your RMSE values resulting at the end of every 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*)

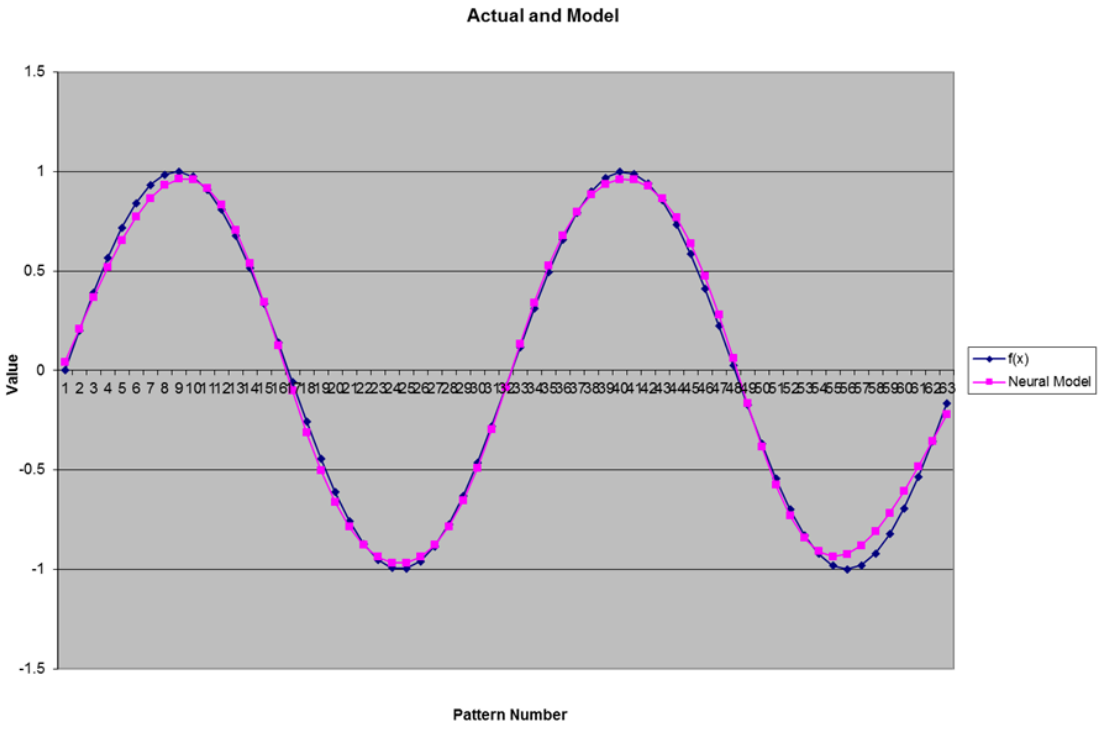
- Try to determine the fewest hidden nodes required to still achieve acceptable RMSE accuracy
- Generate a graph of your results. In the same graph, display: the target function, and the approximated function.

    How do you do this? For each instance in your combined TRAIN+TEST sets, feed the input through your trained model (*without* subsequently changing the weights!) and graph the output for that input. 2D graphs will suffice for #1-2 above; you'll need a 3-D surface graph for #3: { f(x, y) = xy }

    Alternately: for even coverage of the problem-space, use these inputs:

    ```
    for (double x=-5.0; x <= 5.0; x+=0.1) {}
    ```

*(2D example follows)*




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

You may work in teams of up to two (2) on this project.
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.

A 5-10 slide Powerpoint report detailing your best/favorite/most-effective tricks and code(s) to get your neural model to train to each of the functions above. Share your best secrets with the rest of us!!!

Include legible snippets of your source code in your report and in your Powerpoint; and the entirety of your ANN code as an Appendix to your report. Include

all team-members' names in your report and Powerpoint files. Each team-member in your group should turn in both report and Powerpoint to the D2L assignment folder.

---

**External Resources:**
You may find the linked vanilla Python ANN code helpful. It comes with no guarantees (i.e. functionality, accuracy) other than illustration.

Also: this series of Neural Networks From Scratch videos was started up during this class four years ago, and might be useful to you.