# Situation

## Through the looking glass

Previously I was able to pre-train an auto-encoder and the back-propagation fine tuning produced a network that actually made predictions. I made a seemingly benign change to the processing that has altered the training behavior. In case 1 below the trained oSdn produces predictions that are a function of the input vector. In case 2, the network rapidly converges to a state in which its predictions have a constant value independent of the network input. Perhaps I should verify that loading and storing of the oaLayers structure results in identical content?

Case 1:

* Train stack of layers, saving each one in array oaLayers.
* Pickle oaLayers
* Load oaLayers
* Create oSdn1 = SequenceDecimatingNetwork(oaLayers)
* Train oSdn1

Case 2:

* Train stack of layers, saving each one in array oaLayers.
* Create oSdn2 = SequenceDecimatingNetwork(oaLayers)
* Pickle oSdn2
* Load oSdn2
* Train oSdn2

Well this is just totally vexing. I have run both cases in parallel and confirmed that oSdn1==oSdn2. I implemented the \_\_eq\_\_ operator to verify equality of every field and subfield in both objects, and despite their equality the two objects behave entirely different during training.

It is possible that the objects are somehow entangled with the environment due to a shared reference, but I’ve experimented with deep copies and implemented explicit deep copies of seemingly every relevant structure without affecting the odd behavior.

At this point, I am going to suspend investigation of this particular phenomenon and turn my attention to improving training performance during the back-propagation phase.

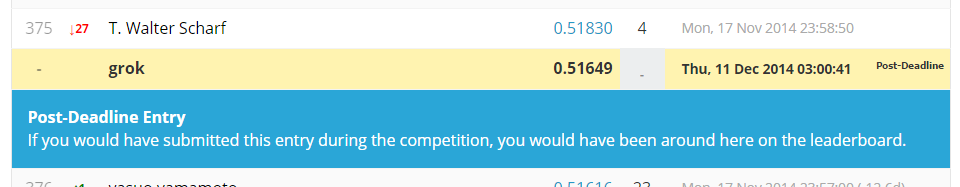
Capturing the current code with the comment: “Stranger than fiction”

## Back Propagation

Following initial training, append an additional Nx1 layer with random weights.

During supervised training, freeze all but the output layer weights during the first two training batches.

Write out train.csv in addition to test.csv.



Posted score of 0.51!