Neural Network Δ-Forecast for El Nino 3.4 Anomalies

dara@lossofgenerality.com

Data

1950 1 24.84 26.26 -1.42
1950 2 25.22 26.53 -1.31
1950 3 26.04 27.09 -1.04
...
2013 10 26.65 26.79 -0.14
2013 11 26.54 26.74 -0.20
2013 12 26.20 26.69 -0.49

Δ

$\Delta_t = X_t - X_{t-1}$

Example: let t be 1950,2 then

$\Delta_{t=1950,2} = -1.31 - (-1.42) = 0.11$

2013, 12:

$\Delta_{t=2013,12} = -0.49 - (-0.20) = -0.29$

0.11
0.27
-0.08

...
0.14
-0.06
-0.29

Forecast Model

$X_{t+1} = X_t + \text{Forecast}(\Delta_t)$

$\Delta_{t+1} \approx \text{Forecast}(\Delta_t)$ since:

$\Delta_{t+1} = X_{t+1} - X_t \implies X_{t+1} = X_t + \Delta_{t+1}$

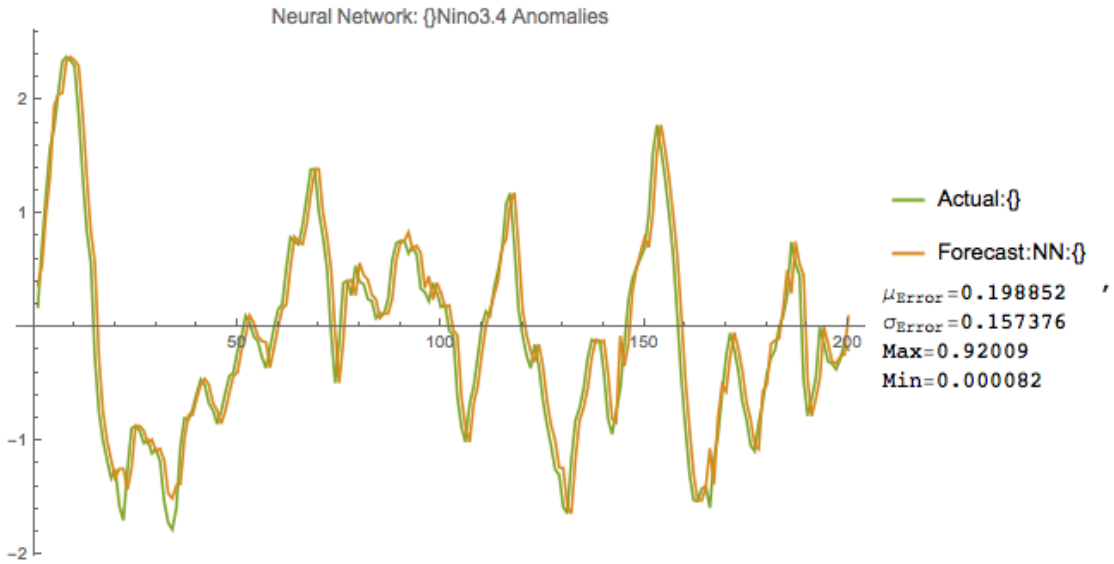
This model avoids wobbling and oscillation and reduces the over-fitting but prone to noise. For noisy data Wavelet Transforms which remove the noise create near ideal forecast for Trend.

Neural Network Forecast

Remark 1. *{} stands for raw data untransformed with no filters*

Average error of about 7.9% with maximum deviation of 38%.

Remark 2. *For normalization divide* μ_{Error} *by 2.38 signal maximum.*



Network Configuration

3 Layers, Input, Hidden and Output.
37 Training samples i.e. Input and Hidden layer each of length 37.
6 length of the Output layer.

300 sweep of the entire data, each step 30 repetition (like in memorization), all together 9000 learning sessions. Error reported for the very last session.

The Backpropagation Adaptive Learning

Because we have more weights in our network than in perceptrons, we firstly need to introduce the notation: wij to specify the weight between unit i and unit j. As with perceptrons, we will calculate a value Δij to add on to each weight in the network after an example has been tried. To calculate the weight changes for a particular example, E, we first start with the information about how the network should perform for E. That is, we write down the target values ti(E) that each output unit Oi should produce for E. Note that, for categorisation problems, ti(E) will be zero for all the output units except one, which is the unit associated with the correct categorisation for E. For that unit, ti(E) will be 1.

Next, example E is propagated through the network so that we can record all the observed values oi(E) for the output nodes Oi. At the same time, we record all the observed values hi(E) for the hidden nodes. Then, for each output unit Ok, we calculate its error term as follows:

$$\delta_{O_k} = o_k(E)(1 - o_k(E))(t_k(E) - o_k(E))$$

The error terms from the output units are used to calculate error terms for the hidden units. In fact, this method gets its name because we propagate this information backwards through the network. For each hidden unit Hk, we calculate the error term as follows:

$$\delta_{H_k} = h_k(E)(1 - h_k(E)) \sum_{i \in outputs} w_{ki}\delta_{O_i}$$

In English, this means that we take the error term for every output unit and multiply it by the weight from hidden unit Hk to the output unit. We then add all these together and multiply the sum by hk(E)*(1 - hk(E)).

Having calculated all the error values associated with each unit (hidden and output), we can now trans-

fer this information into the weight changes Δij between units i and j. The calculation is as follows: for weights wij between input unit Ii and hidden unit Hj, we add on:

$$\Delta_{ij} = \eta \delta_{H_j} x_i$$

[Remembering that xi is the input to the i-th input node for example E; that η is a small value known as the learning rate and that δHj is the error value we calculated for hidden node Hj using the formula above].

For weights wij between hidden unit Hi and output unit Oj, we add on:

$$\Delta_{ij} = \eta \delta_{O_j} h_i(E)$$

[Remembering that hi(E) is the output from hidden node Hi when example E is propagated through the network, and that δOj is the error value we calculated for output node Oj using the formula above].

Each alteration Δ is added to the weights and this concludes the calculation for example E. The next example is then used to tweak the weights further. As with perceptrons, the learning rate is used to ensure that the weights are only moved a short distance for each example, so that the training for previous examples is not lost. Note that the mathematical derivation for the above calculations is based on derivative of σ that we saw above. For a full description of this, see chapter 4 of Tom Mitchell's book "Machine Learning".