Chaos Theory and Neural Networks

Stochastic Volatility and Chaos Theory

Exchange rates are volatile and erratic, not smooth, stable and well behaved and hence cannot be determined using a deterministic equation, instead stochastic models are used in which we represent exchange rates by including stochastic error terms with white noise properties.

Hence,

$$S_t = a*S_{t-1} + e_t$$

Note that e_t is a stochastic term.

The stochastic equivalent of the model containing an explanatory variable is represented by:-

$$S_t = a*S_{t-1} + b*X_t + c*X_{t-1} + e_t$$

$$X_t = \beta*X_{t-1} + \xi_t$$

To understand nonlinearity in exchange rate movements we must understand the concept of feedback effects. According to the stabilising speculation hypothesis, when the exchange rate gets too high the underlying currency is sold, pulling the exchange rate down, and when it is low the currency is bought, leading to a reversion in the direction of the exchange rate. This is an example of feedback effect which can be linear or nonlinear.

Linear feedback implies adjustments, which are proportional to the amount by which the exchange rate is above or below its equilibrium value. By a sequence of such adjustments, the exchange rate is forced back in line. The problem is that a linear feedback effect cannot generate the random (or random-looking) exchange rate movements that we observe in reality.

On the other hand, nonlinear feedback effects mean that corrections are not proportional. Savit (1988) argues that one source of nonlinear reaction is market psychology that causes overreaction to bad news.

Non-Linear feedback can be explained in the terms of equation below.

$$S_t = k*S_{t-1} – k*S^2_{t-1}$$

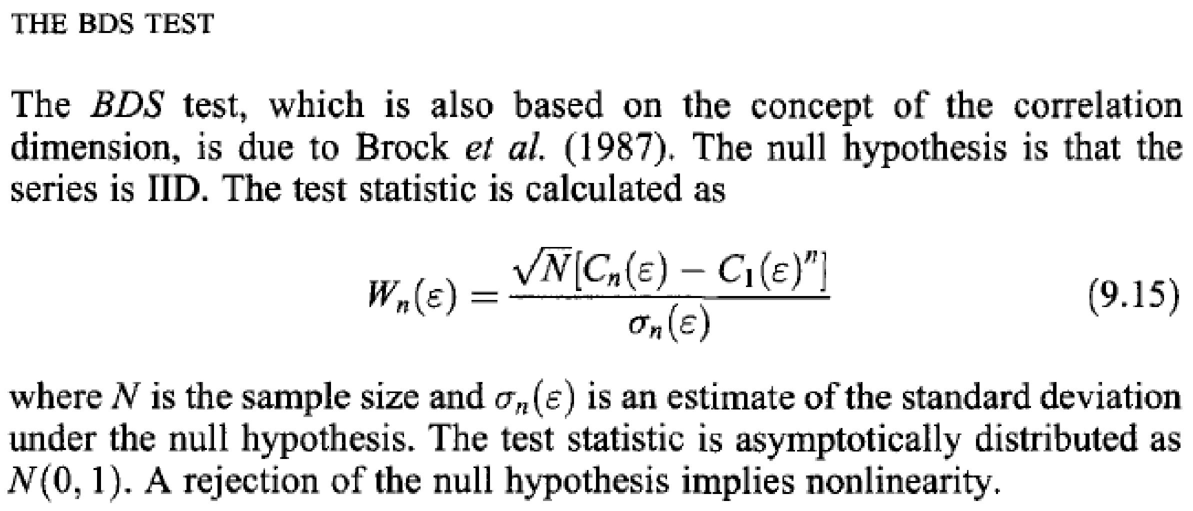
The nonlinear term $k*S^2_{t-1}$, generates negative feedback that competes with the linear term $k*S_{t-1}$, and under some conditions helps to stabilise exchange rate fluctuations. For example, if $k > 1$ but not too large and $0 < So < 1$, then $0 < St < 1$ for any value of t. On the other hand if $St = k*S_{t-1}$ then $S_t = K^t$, which means that the process becomes explosive.

The bulk of exchange rate (univariate or multivariate) models assume linearity. A requirement for the validity of conventional 'Autoregressive' model as a representation of the exchange rate generating process is that the residuals of the model (the difference between the actual values and the values estimated by the model) are not serially correlated. This is because serial correlation simply implies the presence of information in the residuals, which can be utilised to improve the model.

The absence of serial correlation is a property of independent, identically distributed (IID) residuals that economists look for and sometimes assume. However, if the residuals are not serially correlated, this may not necessarily imply the absence of dependence. It could be the case that they are linearly independent, but that they are dependent in a nonlinear manner.

Economists have for some time now dropped the assumption of IID residuals by using the autoregressive conditional heteroscedasticity (ARCH) model due to Engle (1982), or its other variants, such as the GARCH model or the EGARCH model. These models seem to fit the behaviour of financial variables very well, including the property of volatility clustering (periods of extreme turbulence followed by periods of extreme calm). If the ARCH models do not fit well then we need to test for general nonlinearity.

What is needed here is a test which is capable of detecting nonlinear dependence, if it is present, without specifying the nature of this dependence. Three tests are used in the literature: the correlation dimension, the *BDS* test and the Lyapunov exponent test.



THE BDS TEST

The *BDS* test, which is also based on the concept of the correlation dimension, is due to Brock *et al.* (1987). The null hypothesis is that the series is IID. The test statistic is calculated as

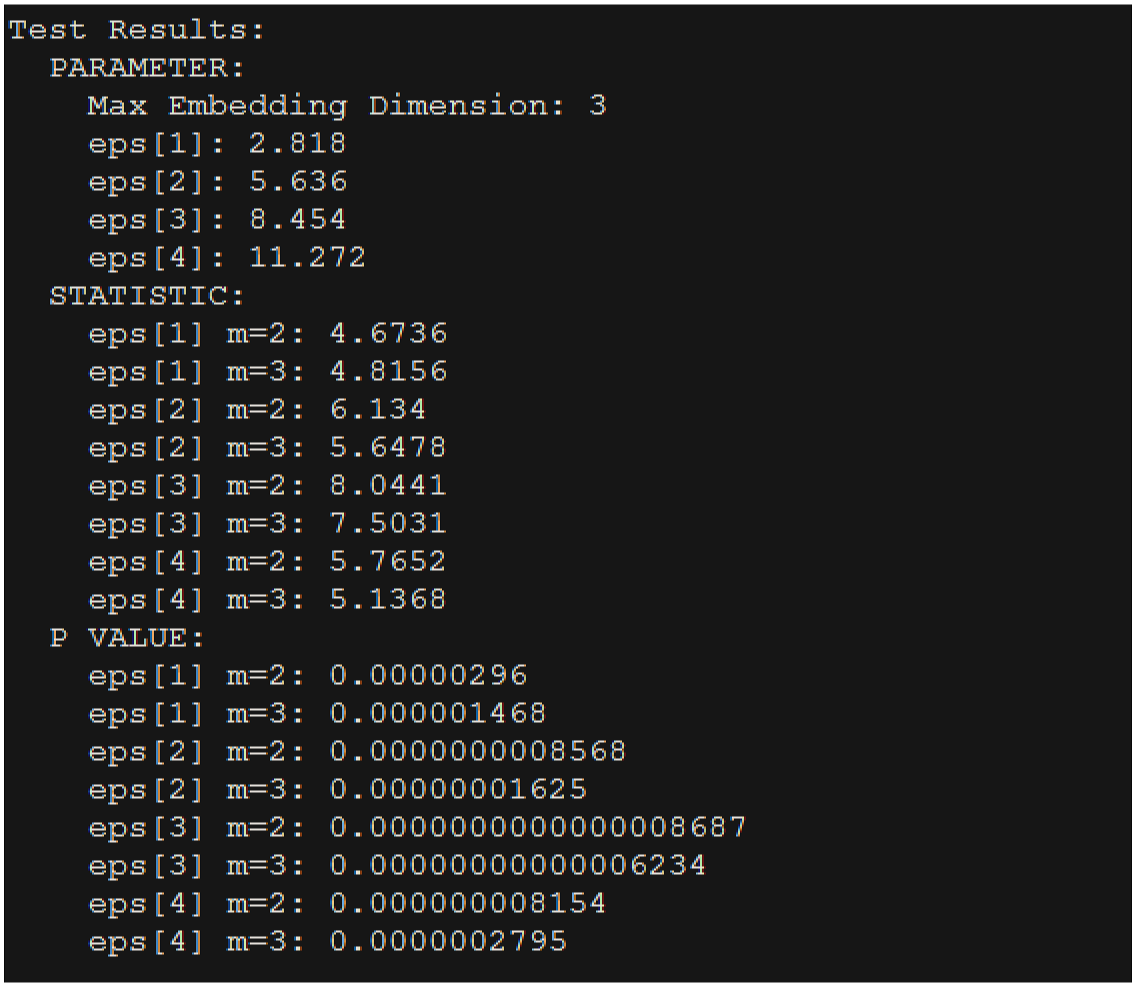
$$W_n(\varepsilon) = \frac{\sqrt{N}[C_n(\varepsilon) - C_1(\varepsilon)^n]}{\sigma_n(\varepsilon)} \tag{9.15}$$

where N is the sample size and $\sigma_n(\varepsilon)$ is an estimate of the standard deviation under the null hypothesis. The test statistic is asymptotically distributed as $N(0,1)$. A rejection of the null hypothesis implies nonlinearity.

This will be done using ‘fNonlinear’ package in R, developed by Diethelm Wuertz. Below is the R code

- bdsTest(final.df[,1], m = 3, eps = NULL, title = NULL, description = NULL)

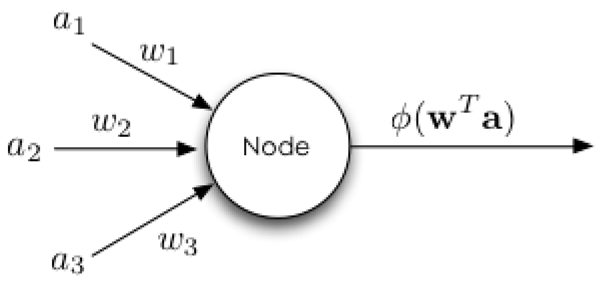
The BDS test is performed at various embedded dimensions (m) like 2,4,8 etc. at various distances (e) like 0.75, 1.0, 1.25 etc. or default values which is seq(0.5*sd(x), 2*sd(x), length = 4). The BDS tests the NULL hypothesis that the series is i.i.d. The rejection of the NULL hypothesis that the random walk hypothesis does not hold good. The NULL of i.i.d is rejected for the FOREX. **Rejection represents presence of non-linear structure.**



```
Test Results:
  PARAMETER:
    Max Embedding Dimension: 3
    eps[1]: 2.818
    eps[2]: 5.636
    eps[3]: 8.454
    eps[4]: 11.272
  STATISTIC:
    eps[1] m=2: 4.6736
    eps[1] m=3: 4.8156
    eps[2] m=2: 6.134
    eps[2] m=3: 5.6478
    eps[3] m=2: 8.0441
    eps[3] m=3: 7.5031
    eps[4] m=2: 5.7652
    eps[4] m=3: 5.1368
  P VALUE:
    eps[1] m=2: 0.00000296
    eps[1] m=3: 0.000001468
    eps[2] m=2: 0.0000000008568
    eps[2] m=3: 0.00000001625
    eps[3] m=2: 0.00000000000000008687
    eps[3] m=3: 0.00000000000006234
    eps[4] m=2: 0.000000008154
    eps[4] m=3: 0.0000002795
```

Artificial Neural Networks and FOREX Forecasting

Below is a typical structure of a Neural Network:-



There are several canonical activation functions. For instance, we can use a *linear* activation function:

$$\phi(\mathbf{w}^T\mathbf{a}) = \mathbf{w}^T\mathbf{a}$$

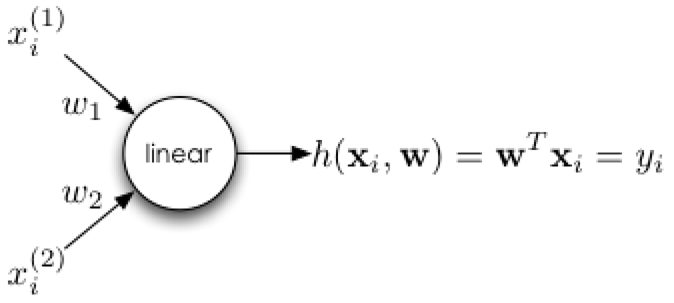
This is also called the *identity* activation function. Another example is the *sigmoid* activation function:

$$\phi(\mathbf{w}^T\mathbf{a}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{a})}$$

We can then form a network by chaining these nodes together. Usually this is done in layers - one node layer's outputs are connected to the next layer's inputs.

We perform regression using Neural Networks and then draw a comparison between the neural model performance and linear model performance.

So what does this have to do with neural networks? In fact, the simplest neural network performs least squares regression. Consider the following single-layer neural network, with a single node that uses a linear activation function:


$x_i^{(1)}$
w_1
linear
$h(\mathbf{x}_i, \mathbf{w}) = \mathbf{w}^T\mathbf{x}_i = y_i$
w_2
$x_i^{(2)}$

This network takes as input a data point with two features $x_i^{(1)}$,$x_i^{(2)}$ weights the features with w1,w2,w3 and sums them, and outputs a prediction . We could define a network that takes data with more features, but we would have to keep track of more weights, e.g. w1,w2 ,…,wj if there are j features.

In our case we have GDP, Bank Call Rate and WPI(Inflation index) as our features which help us in **forecasting** and help us prove why **Neural Nets are a better alternative to linear regression.**

We will then use gradient descent on the loss's gradient $\nabla_w L(w)$ in order to minimize the overall error on the training data. We first derive the gradient of the loss with respect to a particular weight $w_{j\to k}$(which is just the weight of the edge connecting node j to node k [note that we treat inputs as "nodes," so there is a weight $w_{j\to k}$ for each connection from the input to a first-layer node]) in the general case:

$$
\begin{aligned}
\frac{\partial}{\partial w_{j\to k}} L(\mathbf{w}) &= \frac{\partial}{\partial w_{j\to k}} \sum_i \left(h(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2 \\
&= \sum_i \frac{\partial}{\partial w_{j\to k}} \left(h(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2 \\
&= \sum_i 2\left(h(\mathbf{x}_i, \mathbf{w}) - y_i\right) \frac{\partial}{\partial w_{j\to k}} h(\mathbf{x}_i, \mathbf{w})
\end{aligned}
$$

The network function is $h(\mathbf{x}_i, \mathbf{w}) = w_1 x_i^{(1)} + w_2 x_i^{(2)}$. The gradient with respect to w1 is just x1, and the gradient with respect to w2 is just x2. We usually store all the weights of our network in a vector or a matrix, so the full gradient is:

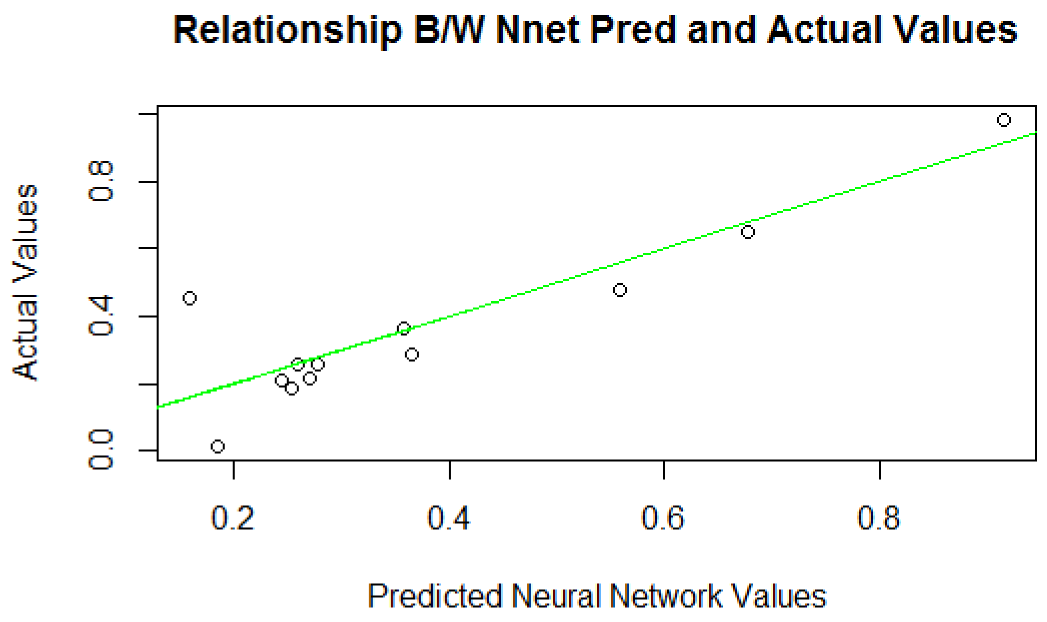
$$
\nabla_{\mathbf{w}} L(\mathbf{w}) = \left(\frac{\partial L(\mathbf{w})}{\partial w_1}, \frac{\partial L(\mathbf{w})}{\partial w_2}\right) = \left(\sum_i 2x_i^{(1)} h(\mathbf{x}_i, \mathbf{w}), \sum_i 2x_i^{(2)} h(\mathbf{x}_i, \mathbf{w})\right)
$$

Using this, we then update our weights using standard gradient descent:

$$
\mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} L(\mathbf{w})
$$

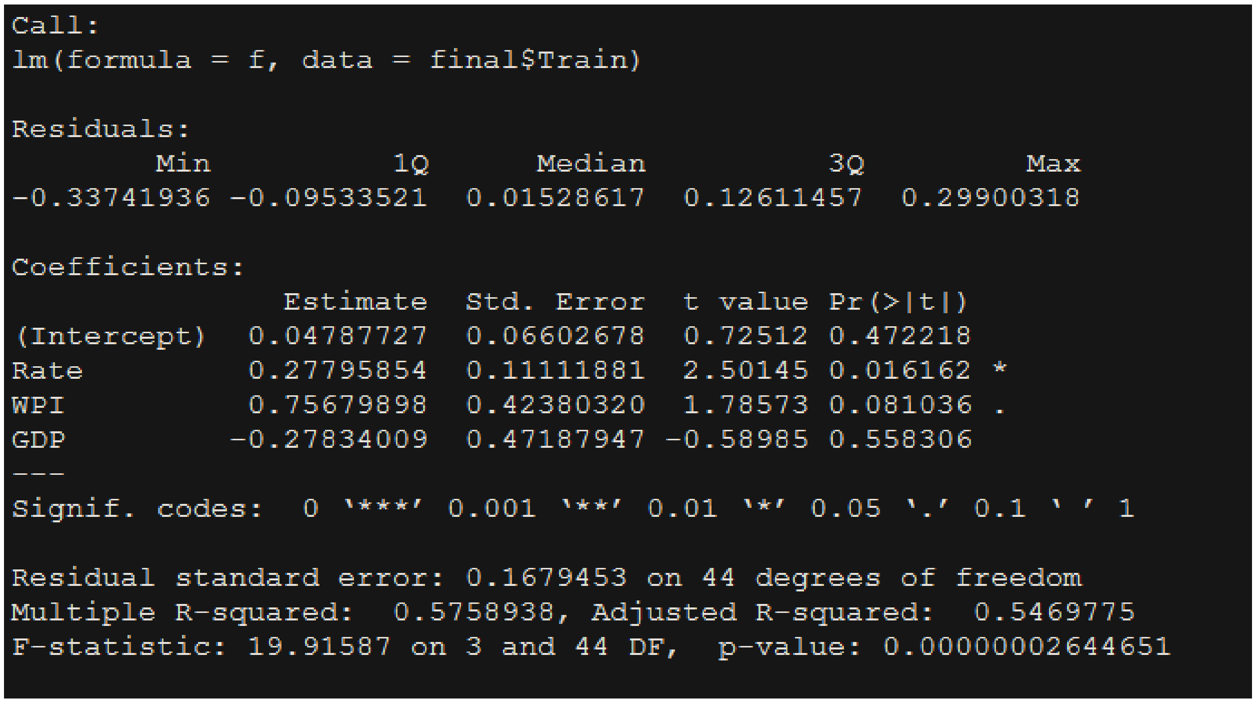
The test error is computed with the quadratic loss, exactly as in training:

$$
L(\mathbf{w}) = \sum_i \left(h(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2 = \sum_i \left(\hat{y}_i - y_i\right)^2
$$



Comparison with Linear Model

It turns out that when linear modeling was done on the same data set R^2 came out to be 0.575.



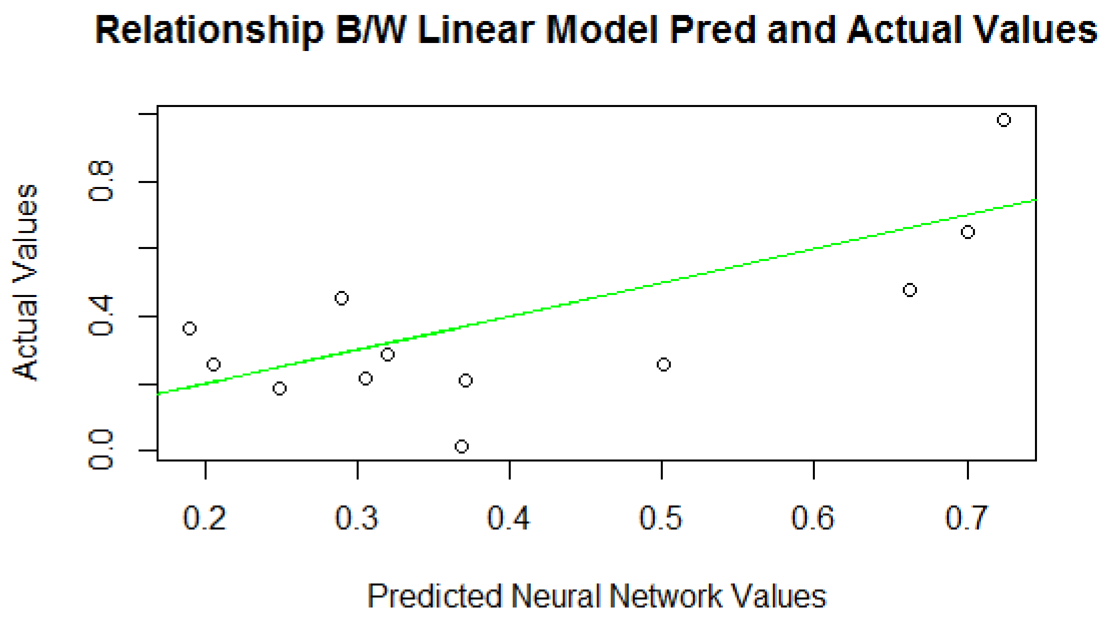
```
Call:
lm(formula = f, data = final$Train)

Residuals:
       Min          1Q      Median          3Q         Max
-0.33741936 -0.09533521  0.01528617  0.12611457  0.29900318

Coefficients:
              Estimate  Std. Error  t value Pr(>|t|)
(Intercept)  0.04787727  0.06602678  0.72512 0.472218
Rate         0.27795854  0.11111881  2.50145 0.016162 *
WPI          0.75679898  0.42380320  1.78573 0.081036 .
GDP         -0.27834009  0.47187947 -0.58985 0.558306
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1679453 on 44 degrees of freedom
Multiple R-squared:  0.5758938, Adjusted R-squared:  0.5469775
F-statistic: 19.91587 on 3 and 44 DF,  p-value: 0.000000002644651
```

Also The Mean Square error of Linear Model was approximately 10 time the error in Neural Network.



We can clearly see in the figure above that how points are distributed for linear model while they should ideally lie along the line; On the other hand the points predicted by a neural network does lie along the ideal line.

References :- 1) Kindly enter book name here

2) Wikipedia.com, R-bloggers.com, briandolhansky.com/