Neural Networks (2009 – 2010)

Multiple Output Echo State Networks

Matthijs Dorst - #1380982
Arjan Eising - #1782347

Table of Contents

Abstract

And Echo State Neural Network model is created and tested against multiple input signals. The resulting network is capable of emulating two distinct figures based on period-changed sinusoids. Various parameter scans are performed to optimize the result and minimize the error, producing a network with acceptable error margins and promising capabilities.

Introduction

We assume the reader has at least a basic familiarity with Echo State Networks (ESN)'s: recurrent partially connected networks with a dynamic reservoir and optionally one or more input and output nodes. Using such an ESN we attempt to create a network capable of recreating two distinct versions of a well defined figure: a sinusoid "8" or "∞".

To start we create an echo state network capable of emulating a sinusoid signal to within negligible margins of error. Using this network we attempt to add a second output node so the signal can be warped in both output dimensions. Then, with two output nodes capable of producing a sinus, we attempt to train the frequency of one node independently, producing the desired output. Finally, the trained weights are combined to create a single network capable of producing the multiple node output desired.

Methodology

Creating the Echo State

A common method to create an echo state, or to be more specific, a dynamic reservoir containing an echo state is to use a random sparse connection matrix of N by N nodes, with N being the total amount of neurons in the network. This ensures an initial state of random connections with low strengths.

Input Vector Initialization

Using a simple formula for the sinus waves of both output vectors and applying this to all steps used in the transient, training and test phase, we create a matrix of desired output values of L x T, with L output nodes and T total steps. A small amount of noise is added to this signal using a random number generator so the network also sees 'imperfect' data.

Driving the Dynamic Reservoir

To get the dynamic reservoir to recognize the input signal, we train it during a transient period. After this period we assume the signal is embedded into the reservoir and we start training the network. Using the activation of the network during this training period a connection weight vector can be obtained to use for training the output weights.

Training Output Vectors

By taking the pseudo inverse of the weight matrix and combining this with the driver signal we create output weights for the network. Over time, these should stabilize the network to the desired output patterns given an input signal.

Since the network itself produces a sinusoid output vector, no further transformation is required and we can suffice with a linear output function. Alternatively a sigmoid or hyperbolic tangent could be used but would require further adjustment to fit the model.

Result Testing

Once the network has been trained, a measurement of its accuracy can be made over both the training and test period by calculating the difference between desired output as determined during the input vector initialization phase and the actual output as produced by the output nodes. Averaging over all output nodes gives an estimate of global network error and thus its accuracy.

Using this error and an automated testing algorithm it is possible to determine the average error for a certain configuration over a predetermined number of trials. From this, it is possible to get some idea as to how the error relates to specific configurations, and what configuration might result in the smallest eventual average error.

Measures are taken to prevent a ‘wild’ network, which can produce error values in the millions and more, from disproportionally affecting the average error for a given parameter value. Instead, these results are discarded and the final average error for that parameter value is taken over the remaining valid errors. As a result, these might proof less accurate however, yet this is considered an acceptable tradeoff.

Parameter Scanning

Using the above described algorithms we attempt to find optimal values for the numerous network parameters. For each parameter we shortly describe its main function, error plot and chosen optimal value. In certain cases this optimal value is randomly chosen when no distinct best value is available – in these cases the parameter is assumed not to influence results too greatly and thus the chosen setting is as well suited as any other.

Alpha scan

The α parameter is a measure for spectral radius. Increasing this will increase reservoir activity to a point where it eventually can oscillate out of control, whereas decreasing this too much may result in a premature dampening of the output matrix. Literature indicates optimal values can generally be found around α = 0.8.

As can be seen in figure no clear trend is discernable, though it would seem both 0.78 as well as 0.9 would

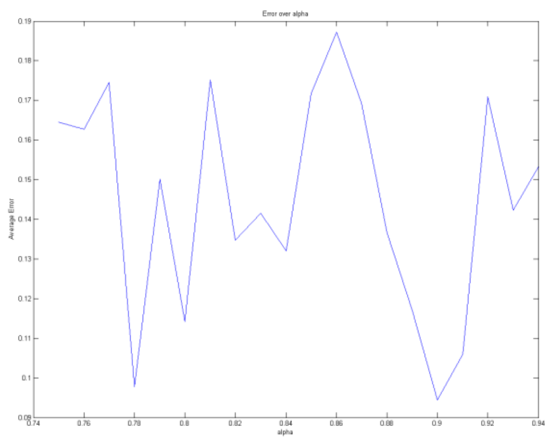


Figure 1: Error over alpha

make good candidates. Since literature suggests an alpha value of 0.8 we choose to use this.

Node count scan

An important parameter is the amount of nodes used in the dynamic reservoir. In more advanced implementations this is not a predetermined amount but instead variable, at least effectively variable, with weights between unimportant nodes and output nodes nearing zero. For a good approximation of the sinusoid we found 20 nodes suggested by literature, so for a double sinusoid we would expect to require at least 40 nodes.

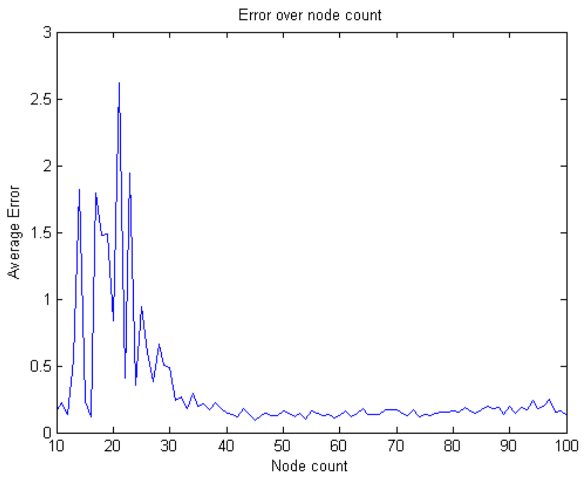


Figure 2: Error over node count

As the error plot shows this estimate is a fair approximation, as error keeps decreasing till well within forty-odd nodes. Adding more nodes has little impact on the average error, so as optimal node count we chose to use 50 nodes. The tail of this error plot suggests that beyond 90 nodes the error actually starts to increase again, indicating overfitting taking place there.

Internal Connection Probability Scan

During initialization of the sparse matrix used to create the dynamic reservoir the internal connection probability value determines how many nodes are actually connected to each other. Lowering this value means fewer connections between nodes and less 'dynamics' in the reservoir, while increasing this value can result in different signals influencing each other too much resulting in a more generic output signal – which, while good for single output networks, is undesirable in our multiple-output network.

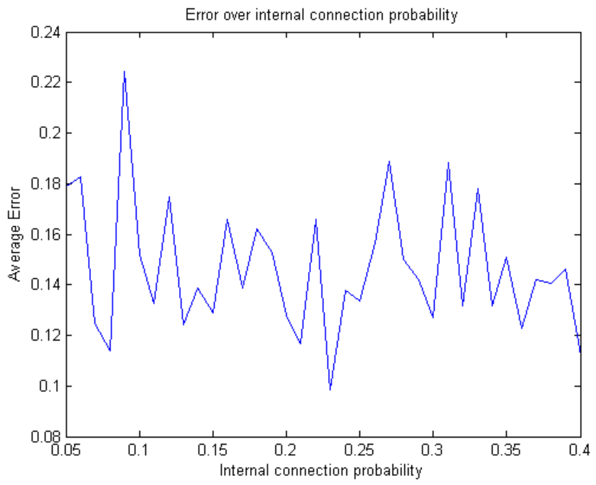


Figure 3: Error over internal connection probability

Generally this value resides around 0.15, or a 15% chance two nodes in the dynamic reservoir are connected. Our error plot in figure 3 shows the effect of this value to be chaotic at best, with slight increases at the lower and higher end of the chosen range of 5% - 40% probability. A value of 21% internal connection probability best seems to fit our current scenario.

Output Connection Probability Scan

The output connection probability determines the chance a node in the dynamic reservoir is connected to one of the output nodes. As with internal connection probability this value is of relatively little influence on network performance, with the extremes merely meaning the dependence on key-nodes increases or decreases and as such the output smoothness and ability to generalize changes accordingly.

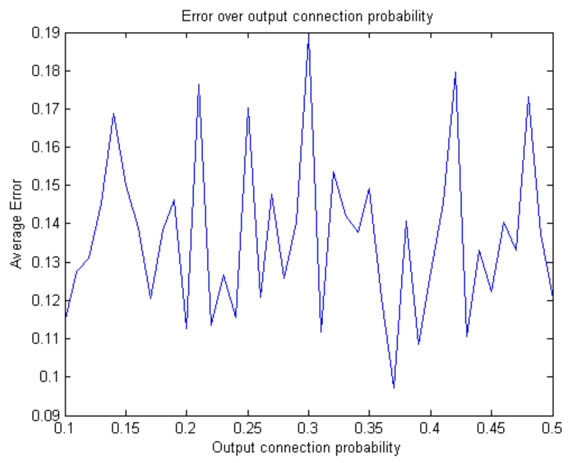


Figure 4: Error over output connection probability

Normally a value of around 30% is used here, and from our error plot it seems if anything can be said that this value is also in this scenario a good estimate. With lowest errors somewhere between 0.3 and 0.4 we choose our output connection probability to be 35%.

Input Noise Ratio Scan

One method to improve a networks ability to generalize is to add randomly distributed noise to the input vector. Of course with too much noise the network might not actually learn the correct pattern, and with too little noise the networks ability to generalize could be compromised, thus choosing the correct noise level can be difficult and varies greatly between networks and intended applications.

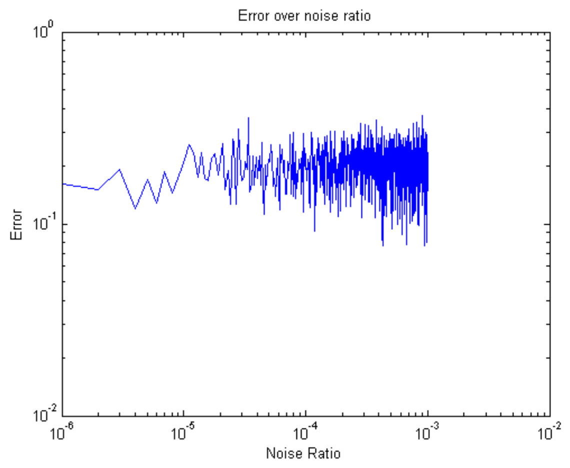


Figure 5: Error over Noise Ratio

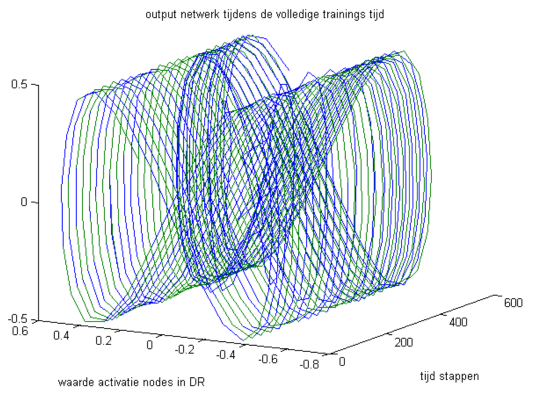
For our initial sinusoid a noise level of 10^{-5} was suggested, so we perform a scan over values a tenth that to hundred times as much. This gives a range of a factor 1000, which should clarify whether noise can be of significant influence on network performance. For clarity sake the results are printed logistically in figure 5. While infinitesimal values seem to have some negative impact on performance, the change for greater noise ratios seems negligible with an optimum around 10^{-5} and a slight increase towards 10^{-3}. We therefore keep the original noise ratio of 10^{-5}.

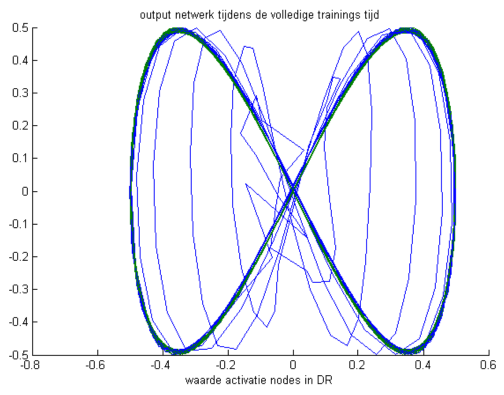
Model Results

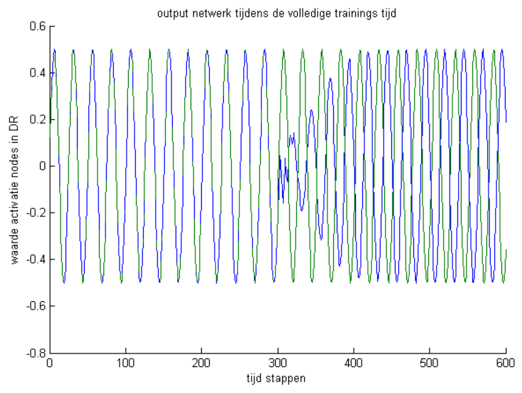
Network & Dynamic Reservoir Output

Figure: infinity

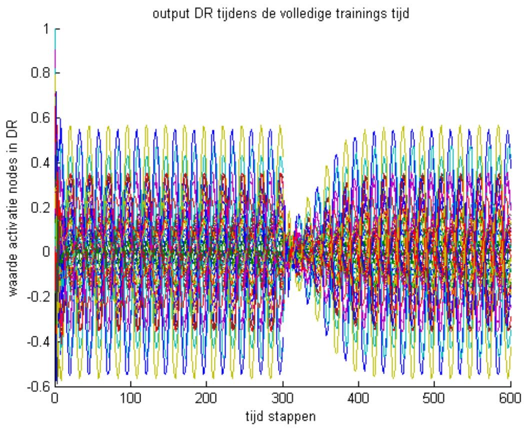
We look at the output over time for the network during a successful run for the ‘infinity’ figure:







As can be seen from the plots the network manages to approximate the desired figure quite well, though during the first test steps it does seem to require some initialization before it reaches an output equilibrium. This would indicate that, though there might be a problem with initial test values, the network learned the pattern successfully and attains it as final stable position.

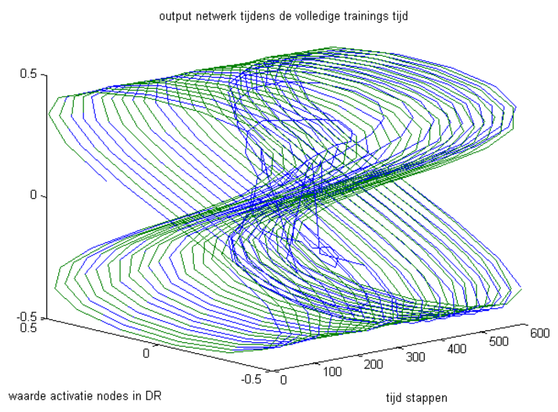


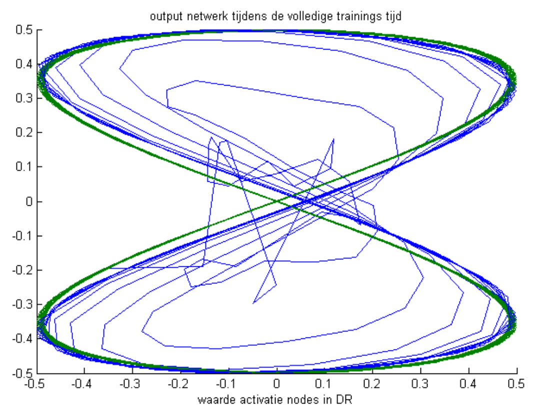
A similar result can be obtained from examining the output of the dynamic reservoir during the entire training period: it clearly shows the reservoir recovering from the transition to testing at step 300 and regaining its original momentum.

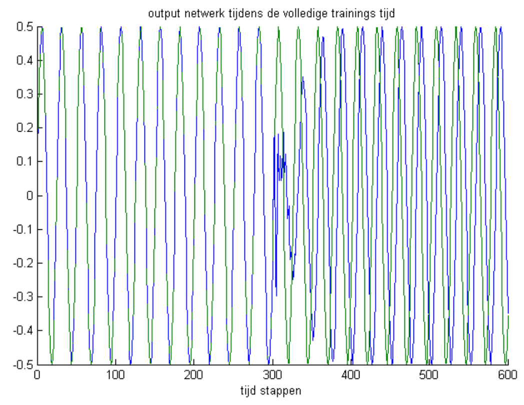
Additionally we can see the chaotic behavior of the reservoir at the very start of the transient period around step 0 to 10 – during this period the network has not actually achieved a stable echo state and would be unusable it its current form, demonstrating the necessity for this transient period.

Figure: eight

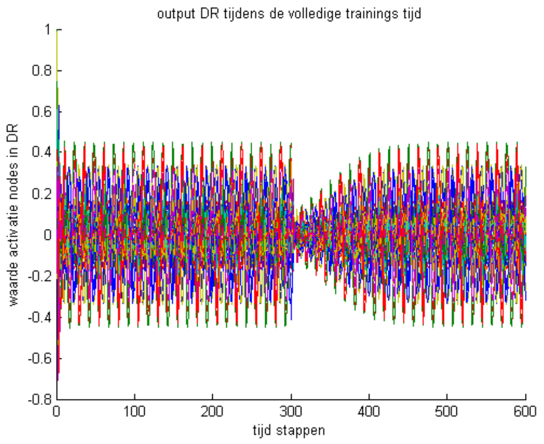
If we switch the periods of both signals the expected output rotates 90˚ to form a figure 8. We give once again the output for both the network as well as the dynamic reservoir:







While still a fair approximation, upon closer inspection one observes some discrepancies in the final network output and the intended figure. We assume this is due to interference from one pattern on another. One explanation for this might be that the order of training changes from slowest first to fastest first, and while a fast pattern could fit in a slow base pattern without affecting it too much, vice versa this might not be the case. Of course this could also simply be due to an error in our implementation, for a discussion see later on in the text.



No significant change can be seen in the dynamic reservoir output: once again we observe chaotic behavior at the start and an initial reset at the beginning of the test period (step 300) followed by retaining the previously established equilibrium state.

Conclusion

Model Performance

While some discrepancies can be observed in model output, especially for the figure 8 scenario, we would argue that overall the network performed well with relatively small errors. Additionally, in a single output scenario the network manages to achieve errors as small as 10^{-11} which in our opinion is quite satisfactory.

Some doubt exists as to the nature of the output 'reset' at the start of the testing period: this can be most easily be explained by an error in our model, yet it seems of relatively small importance due to the networks ability to regain its original dynamic state and still produce valid output.

Improvement suggestions

The combining of both output signals during the training phase is, while novel, not the best possible solution. It would be better (and probably actually required) to feed the network only one combined signal instead of two separate ones and combining the network activity into a single network capable of producing both. Both performance as well as accuracy can be improved upon by remedying this fatal flaw in our design, yet how this can be achieved is, at this point, unclear to us.

While weights are changed according to demand and training signal, one could argue that it might yield better results to dynamically add or remove nodes that perform less than average. We feel that a truly dynamic reservoir should contain a dynamic node complement as well, for which various pruning techniques are available such as genetic algorithms, a complexity penalty, etcetera.

Discussion

Model Design

Our original model was based on a single-output node sinusoid emulator. ESN are well suited to imitate a sinus, even with few internal nodes. It was a logical step to add another sinus input signal, let the network train on this, and combine the results into a single network theoretically capable of emulating both learned signals.

One can (and almost definitely will) argue that this is cheating – the network does not learn two combined signals, it learns one signal at the time and creates a combined result that coincidentally performs quite well under the circumstances. We would implore the reader to consider the validity of this approach: while unconventional, its results are undeniable and its complexity is possibly much smaller than a combined signal approach might yield. That being said, yes, we do acknowledge our mistake here but are due to various reasons unable to fix it.