MLP Coursework 4: Modeling Chaotic Time Series Data with Deep Echo State Networks

s1455790, s1408726

Abstract

In this report we explore time-series modeling of the Mackey-Glass system using echo state networks (ESNs), a unique deep network that is only partially trained. ESNs are a simple yet surprisingly effective tool for modeling temporal data, and the only training they require is the learning of linear regression weights, making them quick and easy to train and test. The ESN has been rigorously experimented since its induction. Alternatively, augmenting the ESN model to new architectures is relatively new research and we are interested in how these new models compare to the original ESN. Specifically, we explore three modern network architectures based around the echo state network: the deep echo state network (DESN), the ensemble echo state network (EESN), and the deep multi-encoding echo state network (DMESN). We evaluated the models using the normalized root mean square error (NRMSE), calculated by having the trained models autoregressively predict 1,000 time steps of Mackey-Glass data. By this metric, we found that the modern architectures easily outperformed the original ESN. Between the deep models, themselves, the DMESN performed the best, with the DESN coming in second. We conclude with an analysis on the internal structure of the DMESN - and some hypotheses as to why it performed so well - in addition to a discussion on some of the problems with deep-ESN architectures that remain as open questions for future work.

1 Introduction

1.1 Review of Last Report

In our interim report, we compared the performance of feed-forward neural networks (FNNs), recurrent-LSTM neural networks (RNNs), and echo state networks (ESNs) when modeling the chaotic Mackey-Glass dynamical system. The best FFNN was unable to effectively model the system, instead predicting a recurrent pattern. The best RNN (an LSTM with 100 units) was better than the FNN due to its ability to store memory, but it required significant computational resources (hundreds of epochs and about two hours to train on a GPU). The RNN still performed worse than the ESN when trying to generate a sequence of 1000 test data points as shown in table 1. We believe that the RNN results could be improved, but their high computational cost and lengthy training time made it challenging to find optimal hyperparameters. Nonetheless, the ESN outperformed all the other models by +70% and was trainable in a single epoch (as only the output weights are trained via linear regression - see Section 1.4). We believe that much of the ESN's success was due to its extremely quick training time, which allowed us to grid search suitable hyperparameters and run many more tests than with the LSTM.

Model	Test NRMSE	% Improvement with ESN
FFNN	1.0559	87%
RNN (LSTM)	0.9660	70%
ESN	**0.5666 ± 0.054**	N/A

Table 1: Autoregressive prediction results on a test set of 1000 time steps for the best FFNN, RNN, and ESN models.

1.2 Overview of This Report

In this report, we expand upon our interim report by combining the echo state network (ESN) with deep-learning style architectures to attempt to improve upon the baseline ESN performance. Specifically, we explore three modern multi-ESN neural network architectures: the ensemble ESN (Yao et al., 2013), the deep ESN (Gallicchio & Micheli, 2017), and the deep multi-encoding ESN (Ma et al., 2017). Before presenting the results on these models, we provide some preliminary background. In Section 1.3, we outline our research questions and objectives, and in Section 1.4, we offer a refresher on the 'vanilla' echo state network. We then introduce the three new ESN architectures in Section 2 and the data we use to train the models in Section 3. We then outline our experimental methodology in Section 4, go over the results in Section 5, then attempt to offer a qualitative comparison of the three deep ESN-based models in Section 6.

1.3 Research Questions and Objectives

There is an abundance of literature on many unique ESN-based architectures (Wang & Li, 2016) (Triefenbach et al., 2010) (Ma et al., 2017) (Gallicchio & Micheli, 2016), but little research formally comparing them. Our research objective is to rigorously analyze three of these newer models and provide some comparative analysis of their performance on well-known time-series benchmarks. Models similar to the ones we will be exploring have shown promising results (Gallicchio & Micheli, 2016). To inform our comparative analysis, we also explore some of the basic properties that make ESNs work and discuss how they translate into the deeper models. Many of these properties have already been

explored in shallow ESN models (Liebald & Jaeger, 2004), and it is a compelling question whether these results are consistent with more complicated architectures.

The contributions we made are listed below:

- Built a NumPy framework for building and testing these three deep-ESN architectures.
- Successfully trained three deep-ESN architectures and provided comparative results.
- Coalesced the deep-ESN research into a standardized system for testing and benchmarking results.

1.4 Review of Echo State Networks

Echo state networks (ESNs) (Jaeger, 2001) are designed to model temporal data represented as: $\boldsymbol{u}(1), \boldsymbol{u}(2), ..., \boldsymbol{u}(t)$. Keeping with the notation used by H. Jaeger (the inventor of the ESN) (Jaeger, 2007), we denote the dimensionality of $\boldsymbol{u}$ as K. The focal point of the echo state network is the *reservoir*, an untrained, randomly initialized recurrent neural network. The number of units in the reservoir is N, where $N \gg K$. The reservoir is characterized by an $N \times K$ input weight matrix $\boldsymbol{W}^{in}$, an internal recurrent $N \times N$ weight matrix $\boldsymbol{W}^{res}$, a scalar echo parameter $\beta \in (0, 1)$ and an activation function f (typically *tanh*). $\boldsymbol{W}^{res}$ and $\boldsymbol{W}^{in}$ are initialized once and then left untrained. We initialize each element of $\boldsymbol{W}^{res}$ from a uniform distribution over $[-0.5, 0.5]$, and each element of $\boldsymbol{W}^{in}$ by sampling from the set $\{-0.5, 0.5\}$ with equal probability. Some papers (Ma et al., 2017) have also seen success with a sparse reservoir weight matrix, where each element (i, j) in $\boldsymbol{W}^{res}$ is non-zero with probability ρ:

$$\boldsymbol{W}^{res}_{ij} = \begin{cases} w & if \quad \omega \leq \rho \\ 0 & otherwise, \end{cases} \tag{1}$$

where w and ω are samples from the uniform distribution over $[-0.5, 0.5]$ and $[0, 1]$, respectively.

As the data $\boldsymbol{u}(1), \boldsymbol{u}(2), ..., \boldsymbol{u}(t)$ is fed into the reservoir, the states of its recurrent units (which represent be an N-dimensional vector $\boldsymbol{x}(t) \in \mathbb{R}^N$) are updated as follows:

$$\boldsymbol{x}(t) = \beta f(\boldsymbol{W}^{res}\boldsymbol{x}(t-1) + \boldsymbol{W}^{in}\boldsymbol{u}(t)) + (1-\beta)\boldsymbol{x}(t-1) \tag{2}$$

Intuitively, the echo parameter, β, controls the rate of turnover in the network's memory: setting it to a low value causes the reservoir states to change to new inputs slowly, while setting it to a high value causes them to change quickly with new inputs, thereby washing away information about the past input history of $\boldsymbol{u}$ at a quicker rate.

With the reservoir states, we have effectively non-linearly transformed the K-dimensional input sequence to a much larger N-dimensional representation $\boldsymbol{x}(t)$. In this sense, "the reservoir of an ESN can be viewed as a nonlinear temporal kernel, which can map sequences of inputs into a high-dimensional space" (Ma et al., 2017).

After inputs $\boldsymbol{u}(1), ..., \boldsymbol{u}(t)$ have been fed into the reservoir, prediction of $\boldsymbol{u}(t+1)$ can be done by a simple linear transformation of the reservoir state by the trained output weights:

$$\hat{\boldsymbol{u}}(t+1) = g(\boldsymbol{W}^{out}\boldsymbol{z}(t)), \tag{3}$$

where $\hat{\boldsymbol{u}}(t+1)$ is the predicted value of $\boldsymbol{u}(t+1)$, $\boldsymbol{W}^{out}$ is a $K \times (N+K)$ weight matrix, $\boldsymbol{z}(t) = [\boldsymbol{x}(t); \boldsymbol{u}(t)]$ is the extended system state, and g is an optional output activation function (we always use the identity function). A simple diagram of the ESN architecture can be found in Appendix A.

1.4.1 Training Echo State Networks

With ESNs, only the output weights $\boldsymbol{W}^{out}$ require training. Before training them, however, we must feed t_{echo} time-steps of input data into the reservoir sequentially to 'warm-up' the reservoir (t_{echo} can be chosen arbitrarily, but should not be too low, for example ≥ 100). This 'warm-up' stage is essential because the reservoir states $\boldsymbol{x}$ are initialized arbitrarily and do not initially store memory of the past input history (Jaeger, 2007). To avoid training the output weights on these arbitrary starting states, data must be fed into the reservoir before training so that the states $\boldsymbol{x}$ reflect the most recent input history of $\boldsymbol{u}$.

After the 'warm-up' stage, we can train the output weights using the ensuing T-time-steps of training data, which we denote as a $T \times K$ matrix $\boldsymbol{D}$. Training is done by feeding each data point in the training sequence into the reservoir, harvesting the extended system states $\boldsymbol{z}(1), ..., \boldsymbol{z}(T)$ as we go and constructing a $T \times (N+K)$ "state collection" matrix $\boldsymbol{S}$, such that $\boldsymbol{S}_{i,:} = \boldsymbol{z}(i)$. The output weights are then calculated by solving the following linear system:

$$\boldsymbol{W}^{out} = (\boldsymbol{S}^{\dagger}\boldsymbol{D})^T, \tag{4}$$

where $\dagger$ denotes the pseudo-inverse. As the above system is highly over-determined ($T \gg N + K$), adding Tikhonov regularization to the output weights speeds up calculation of the pseudo-inverse and tempers overfitting during training:

$$\boldsymbol{W}^{out} = (\boldsymbol{S}^T\boldsymbol{S} + \lambda\boldsymbol{I})^{-1}\boldsymbol{S}^T\boldsymbol{D}, \tag{5}$$

where λ is a hyperparameter and $\boldsymbol{I}$ is the identity matrix.

The solving of this system is all the training an ESN requires. By directly inverting the matrix, this can be done in $O(T(N+K)^2 + (N+K)^3)time$. When a direct inversion is too expensive, other methods (such as the method of conjugate gradients (Hestenes & Stiefel, 1952)) can be used to solve the system much quicker. During our experiments, even relatively large models (e.g. 2,000 total reservoir units) were able to train about 8 seconds (on a sequence of 14,000 one-dimensional pieces of data), and using direct matrix inversion, no less.

The simplicity of training also makes echo state networks very capable for online learning via "recursive least squares algorithms" (Jaeger (2002), Section 4). Due to time constraints, we do not explore this capability in this report.

1.4.2 The Echo State Property

In the last section, we stressed the importance of 'warming-up' the reservoir with some amount of initial data, so as to avoid training $\boldsymbol{W}^{out}$ on the arbitrary initial states. This relates to a crucial property of a well-working ESN: the 'echo state property' (ESP), which states that "the reservoir [should] asymptotically wash out any information from the initial conditions" (Jaeger, 2007) as inputs $\boldsymbol{u}$ are fed into it. It has been empirically observed that this property is almost guaranteed to hold for all inputs when the spectral radius of the reservoir weights $\boldsymbol{W}^{res}$ is less than 1 (Jaeger et al., 2007). We discuss this heuristic in a bit more detail in Section 5.3.1). For inputs with large amplitudes, the spectral radius can be raised above 1 while still obtaining the ESP. To ensure we obtain the ESP during our experiments, we divide the randomly initialized $\boldsymbol{W}^{res}$ element-wise by its largest singular value, γ_{eig}, then multiply element-wise by a hyperparameter α so that the spectral radius of $\boldsymbol{W}^{res}$ equals α:

$$W^{res}_{ij} = \frac{\alpha}{\gamma_{eig}} W^{res}_{ij}. \tag{6}$$

In this report, we also try varying the scale of the input weights $\boldsymbol{W}^{in}$ with a parameter γ:

$$W^{in}_{ij} = \gamma \cdot W^{in}_{ij}. \tag{7}$$

1.4.3 Randomness in Echo State Networks

Compared to fully-trained recurrent networks, echo state networks deal with a lot of randomness, as both the reservoir weights and the input weights are initialized randomly and remain untrained. As a result of this, networks trained using the same hyperparameters but different (randomly initialized) $\boldsymbol{W}^{res}$ and $\boldsymbol{W}^{in}$ can have significantly different performances. Even ESNs with optimal hyperparameters (found through a thorough grid search) can yield normalized root mean square errors (NRMSEs, see Section 4) ranging from 0.1 to 1.2 (and rarely, even higher).

Sections 5.1 through 5.3.2 give examples of the NRMSE distributions these models produce. As these distributions of NRMSEs are not always symmetric or even unimodal, we find that reporting mean NRMSEs is not a sufficient method for quantifying network performance (at least not on its own). As such, we attempt to evaluate the networks both qualitatively (with these error histograms), and quantitatively (e.g. with mean NRMSE over a number of independent runs).

2 Overview of the Deep ESN-Based Networks

2.1 Ensemble Echo State Network

The ensemble echo state network (EESN) is a natural generalization of the ESN. In our experience, the biggest issue with ESNs we have faced is that their performance often depends heavily on the random weight initializations, and also on their hyperparameter settings (i.e. echo parameter, reservoir size, ...). EESNs aim to reduce the impact of this issue by using R independent reservoirs (Yao et al., 2013). A simple diagram showing the EESN architecture can be found in Appendix A.

With R different reservoirs, a different echo parameter, spectral scale, sparsity and/or input scale can be used for each reservoir, which theoretically should provide a more informative representation of $\boldsymbol{u}$. For example, using a low echo parameter β in one reservoir and a high β in another could offer both the low-β reservoir's "knowledge" of time further back, plus the high-β reservoir's stronger "knowledge" of the recent input history.

For some input $\boldsymbol{u}(t)$, we propagate it through each of the EESN's reservoirs independently, and return the concatenated reservoir states as the extended system state: $\boldsymbol{z}(t) = [\boldsymbol{x}^{(0)}(t); \boldsymbol{x}^{(1)}(t); ...; \boldsymbol{x}^{(R)}(t); \boldsymbol{u}(t)]$. Training of the output weights is the same as in a vanilla ESN, as outlined in section 1.4.

In addition to the advantages offered by the ability to use multiple hyperparameter settings over multiple reservoirs, EESNs also have a lower number of total recurrent weights than an ESN with the same number of total units. For instance, an EESN with 5 reservoirs with 200 units each has $5 \cdot 200^2 = 200,000$ units, compared to an ESN with 1000 recurrent units, which has $1,000,000$ recurrent weights.

2.2 Deep Echo State Network

In a DESN, the reservoirs are stacked like a multilayer perceptron: the first reservoir takes the input signal then feeds its output into the second, the second feeds its output into the third, and so on. The states of the first $r_1, .., r_{R-1}$ reservoirs of an R-deep DESN are directly connected to the input weights of the following $r_2, .., r_R$ reservoirs. The first reservoir works the same as in a vanilla ESN, taking the raw input signal $\boldsymbol{u}(t)$ as input and updating its states as in Equation 2. The states of the first reservoir $\boldsymbol{x}^{(1)}(t)$ are then fed as inputs into the second reservoir, the states of the second $\boldsymbol{x}^{(2)}(t)$ as inputs to the third and so on:

$$\begin{aligned}\boldsymbol{x}^{(i+1)}(t) = \beta f(\boldsymbol{W}^{in}_{(i+1)} \boldsymbol{x}^{(i)}(t) + \boldsymbol{W}^{res}_{(i+1)} \boldsymbol{x}^{(i+1)}(t-1)) \\ +(1-\beta)\boldsymbol{x}^{(i+1)}(t-1),\end{aligned}$$

for $i = 2, ..., R$.

Prediction of future data $\hat{\boldsymbol{u}}(t+1)$ is done via Equation 3, but with the extended system state $\boldsymbol{z}(t)$ containing the states of all R reservoirs:

$$\boldsymbol{z}(t) = [\boldsymbol{x}^{(1)}(t); ...; \boldsymbol{x}^{(R)}(t); \boldsymbol{u}(t)]. \tag{8}$$

For a DESN with R reservoirs, each with $\frac{N}{R}$ units, the total number of weights is $2\frac{N^2}{R} + 2N$. Therefore the DESN has $\frac{2}{R}$ times less weights than a vanilla ESN. Intuitively, this makes sense as the DESN is theoretically equivalent to a vanilla ESN with a recurrent weight matrix $\boldsymbol{W}^{res}$ constrained to act like segregated sub-reservoirs. A simple diagram of the DESN architecture can be found in Appendix A.

The DESN (as well as the DMESN, discussed next) require a variation of the 'warm-up' procedure described

in Section 1.4.1. Because a reservoir at depth $i+1$ is 'down stream' from the data signal of a reservoir at depth i, it is important that the data *leaving* reservoir i is from a reservoir that is 'warmed-up' (otherwise reservoir $i+1$ will be fed data that is not from the real data distribution). This is even more of a concern when intermediate training (as with the DMESN) occurs between reservoirs. To 'warm-up' all reservoirs correctly, we must set aside $R \cdot t_{echo}$ time steps of the training data for 'warming up'. First, all $\boldsymbol{u}(0), ..., \boldsymbol{u}(R \cdot t_{echo})$ initial inputs are fed into the first reservoir to obtain the outputs $\boldsymbol{u'}^1(0), ..., \boldsymbol{u'}^1(R \cdot t_{echo})$ and of which we immediately *discard* the first t_{echo} outputs: $\boldsymbol{u'}^1(0), ..., \boldsymbol{u'}^1(t_{echo})$. The second reservoir is then fed the remaining inputs (which have t_{echo} less data than the first reservoir) and we discard t_{echo} more data from its output (we now are left with $\boldsymbol{u'}^2(2 \cdot t_{echo}), ..., \boldsymbol{u'}^2(R \cdot t_{echo})$). This process repeats for all R reservoirs, by which then the initial data input will have been exhausted.

2.3 Deep Multi-Encoding Echo State Network

The deep multi-encoding ESN (DMESN) (Ma et al., 2017), as the name might suggest, is structured similarly to the DESN, where each reservoir's state acts as inputs to the next reservoir. This deep ESN-based network, however, also uses unsupervised learning to extract salient features from the reservoirs' states. As usual, the first reservoir takes the raw input signal $\boldsymbol{u}(t)$ as input. The N_1-dimensional states $\boldsymbol{x}^{(1)}(t)$ of that reservoir are then transformed into $\hat{\boldsymbol{x}}(t)$ using some dimensionality reduction technique τ:

$$\hat{\boldsymbol{x}}^{(1)}(t) = \tau(\boldsymbol{x}^{(1)}(t)). \tag{9}$$

This is done for all reservoirs except the last, whose states $\boldsymbol{x}^{(R)}$ are left un-encoded. For this technique to work properly, each encoded $\hat{\boldsymbol{x}}^{(i)}$ should have a dimensionality $\hat{N}_i$ which is much smaller than the number of states in the corresponding reservoir, N_i (Ma et al., 2017).

Prediction of future data $\hat{\boldsymbol{u}}(t+1)$ is again done via Equation 3, but with the states of reservoirs 1 through $R-1$ replaced by their encoded versions in $\boldsymbol{z}(t)$:

$$\boldsymbol{z}(t) = [\hat{\boldsymbol{x}}^{(1)}(t); ...; \hat{\boldsymbol{x}}^{(R-1)}(t); \boldsymbol{x}^{(R)}(t); \boldsymbol{u}(t)]. \tag{10}$$

There are many potential choices for τ. The original authors tried DMESNs with three types of encoders: principal component analysis (PCA), random projection (RP), and an 'extreme learning machine' (ELM). In this paper, we experiment with the best-performing encoder from the paper, PCA, and with an original architecture that uses variational autoencoders (VAEs) as the encoder (Kingma & Welling, 2013). The justification for using VAEs is that they provide non-linear encoding that is more robust than traditional autoencoders (Doersch, 2016). Because PCA is a linear encoding, replacing it with a non-linear encoding could help extract more useful latent features from the reservoir. We provide a brief review of VAEs in Appendix A.

A diagram showing the DMESN architecture can be found in Appendix A.

The DMESN, as a hierarchical network, must be warmed up reservoir-by-reservoir as with the DESN (see Section 2 for a thorough explanation of the 'warm-up' process). The encoders are also trained one by one: after reservoir 1 is warmed up, T data points are driven into it and encoder 1 is trained on the harvested states. After training, the first t_{echo} time steps from the data used to train encoder 1 are passed through the encoder to warm up reservoir 2, and so on for each reservoir and encoder.

As these encoders require zero-mean data (and the VAE also requires data with a variance of 1), we calculate the means and variances of each encoder's inputs during training and normalize the training data. Future data is normalized using these stored mean and variance values.

3 The Data

To evaluate all model architectures, we use low-dimensional, easily obtainable time series data that are very challenging to model. In particular, we chose to use the Mackey-Glass chaotic system. We planned on also testing the models on Lorenz system and Henon map data, but weren't able to due to time constraints. A chaotic dynamical system is roughly defined as a time series which is highly sensitive to its starting conditions, i.e. changing the initial parameters by a small amount could cause the system to look wildly different, say, one hundred time steps later. This phenomenon is commonly referred to as 'the butterfly effect'. These systems "are always deterministic and may be very simple, yet they produce completely unpredictable and divergent behaviour." (Boeing, 2016).

The Mackey-Glass system is defined by a continuous time-delay differential equation:

$$\frac{dx}{dt} = \beta \frac{x_\tau}{1 + x_\tau^n} - \gamma x, \tag{11}$$

where β, γ, and τ are positive constants.

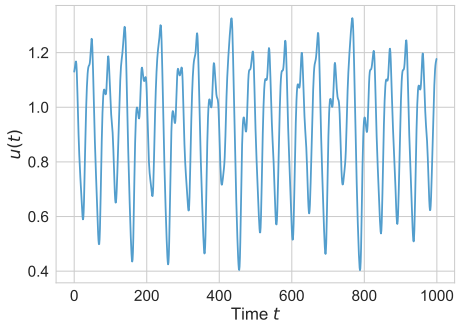


Figure 1: The first 1,000 steps of Mackey Glass data generated with parameters $x_0 = 1$, $\beta = 0.2$, $\tau = 17$, $\gamma = 0.1$.

4 Experimental Methodology

4.1 Implementation

All models were implemented by us in Python using only NumPy. The DHESN also required the use of PyTorch to implement the variational auto-encoder, and scikit-learn's

PCA library for the PCA version. The culmination of this report was a fast and efficient library for deep-ESNs which will be made publicly available.

4.2 Error Metric

Our primary method for quantifying network performance is the normalized root mean square error (NRMSE):

$$NRMSE(\boldsymbol{\theta}) = \frac{\sum_{i=1}^{T-1}(f(x_{t-1};\boldsymbol{\theta}) - x_t)^2}{\sum_{i=1}^{T-1}(x_t - \hat{x})^2}, \qquad (12)$$

where x_t is a data point at discrete-time t, $f(x_t;\boldsymbol{\theta})$ is some model's prediction for x_t given the previous piece of data x_{t-1} with parameters $\boldsymbol{\theta}$, and $\hat{x}$ is the true mean of the data across the considered time period. The advantage of using NRMSE over traditional mean squared error (MSE) is that the evaluation of the model is independent of the units of the data (Forman-Gornall, 2013). We can therefore compare the model performance across multiple dynamical systems datasets of different scales (Farzad et al.). A NRMSE of zero indicates perfect precision; a NRMSE of one indicated the model is no better than predicting the mean value of the data, $\hat{x}$.

4.3 Experiment Details

For all experiments, we generate a Mackey-Glass sequence of length 16,000. The first 14,000 data points are used as training, the next 1,000 as validation data, and the final 1,000 as test data. The Mackey-Glass data used during grid searching for hyperparameters was initialized with parameters:

$$x_0 = 1, \quad \beta = 0.2, \quad \tau = 17, \quad \gamma = 0.1.$$

All models were trained and tested on a single CPU. To test the model's performance after training, we generate a sequence of 1,000 data points by feeding the first validation data point into the model, then recursively feeding the output of the model at time step t as the input of the model at time step $t+1$. This justifies why the validation and testing datasets are so small relative to the training dataset: beyond 1,000 data points, the 'butterfly effect' (see section 3) causes the model to inevitably deviate from the correct data sequence and cause the error metric to be very high, even when prediction is very accurate for the first 600-700 time steps.

As a review, the hyperparameters of each of the deep ESN-based models are listed below:

- Number of reservoirs R.
- Reservoir sizes $N_1, ..., N_R$.
- Echo parameters $\beta_1, ..., \beta_R$.
- Spectral scales $\alpha_1, ..., \alpha_R$.
- Input weight scales $\gamma_1, ..., \gamma_R$.
- Sparsities $\rho_1, ..., \rho_R$.
- Regularization strength λ.
- For the DMESN only:
  - Choice of encoder τ.
  - Encoder dimensionalities $\hat{N}_1, ..., \hat{N}_{R-1}$.

All of this amounts to a huge hyperparameter space to search, but with the extremely quick training times of these models, it can still be done. We optimized these hyperparameters with a massive grid search, totaling to about 1,000 combinations of settings for each model. We also attempted to use genetic algorithms to find optimal hyperparameters (as done in Ma et al. (2017)). While *Ma et al.* only optimized the hyperparameters for three reservoirs and then repeated those settings in the remaining reservoirs, we attempted to optimize them over the whole space, which proved too difficult.

During our experiments, we found that larger and more numerous reservoirs tended to yield better prediction power, regardless of model. As such, we try to keep these quantities similar across models for the sake of comparison. Due to time constraints, we also were not able to explore varying the sparsities across reservoirs, instead exploring one ρ which holds for them all.

5 Experiment Results

The optimal hyperparameters found for each model are listed below:

Model	R	$N_1, ..., , N_R$	$\beta_1, ..., \beta_R$	$\alpha_1, ..., \alpha_R$
ESN	NA	1000	0.85	1.25
EESN	10	200	0.5,...,0.85	1.25
DESN	10	200	0.85, ..., 0.5	1.0
DMESN_{PCA}	8	200,...,400	0.5, ..., 0.1	0.4, ..., 1.2
DMESN_{VAE}	4	200, ..., 400	0.5, ..., 0.1	1, ..., 0.7

Model	$\gamma_1, , ..., \gamma_R$	ρ	λ	$\hat{N}_1, ..., \hat{N}_{R-1}$
ESN	1.0	1.0	1e-5	NA
EESN	0.5	1.0	1e-4	NA
DESN	0.2	1.0	1e-5	NA
DMESN_{PCA}	0.5	1.0	1e-6	30,...,80
DMESN-VAE	0.5	0.1	1e-2	30, ..., 80

Entries with two values and an ellipsis denote a linear increase (or decrease) in the parameter through the reservoirs, e.g. $0.1, ..., 1.0$ with ten reservoirs denotes $0.1, 0.2, ..., 0.9, 1.0$. Whenever a single value is listed, this is the value every reservoir uses.

We tried basic ESNs with more than 1,000 recurrent units, but found the improvement was marginal and the training time significantly increased.

5.1 Ensemble Echo State Network Results

The training time for one EESN with 2,000 total reservoir units was almost always between 7 and 8 seconds. Intuitively, we would expect an EESN to outperform a vanilla ESN because the use of multiple reservoirs dampens stochasticity in the output evaluation. Pleasingly, the best EESN had 4.04×10^5 weights making a significantly smaller model than the vanilla ESN, which required 1.002×10^6

weights.

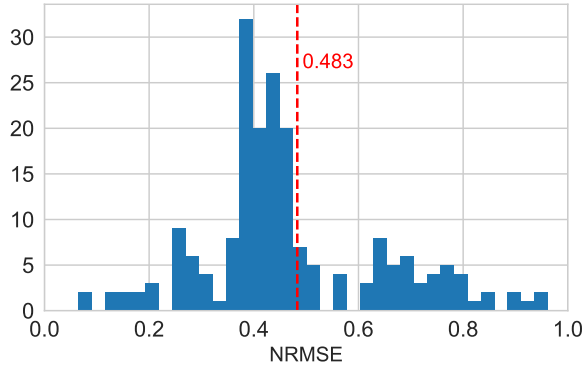


Figure 2: Distribution of NRMSEs from our best EESN's autoregressive predictions $\hat{\boldsymbol{u}}$.

5.2 Deep Echo State Networks Results

The training time for one DESN with 2,000 total reservoir units was almost always between 8 and 9 seconds. The best DESN model contained 7.622×10^5 weights, which was double the size of the EESN. From this result, if one is planning to use a vanilla ESN for a problem, we would recommend always using an DESN instead; the reduction in model size coupled with the gain in performance makes DESNs a competitive choice for simple architectures.

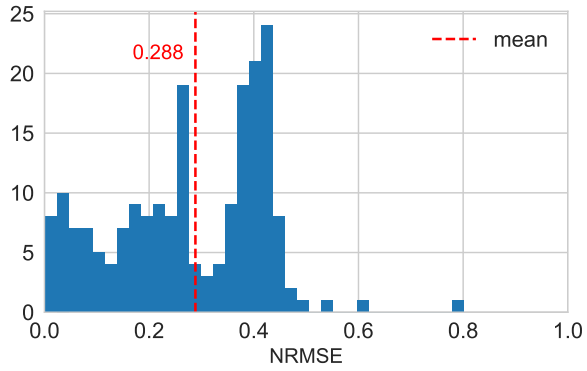


Figure 3: DESN histogram validation.

5.3 Deep Multi-Encoding Echo State Network Results

We tried two types of encoding techniques in the DMESN: principal component analysis (PCA) and variational autoencoders (VAEs). The use of PCA was proposed by Ma et al. (2017), while the use of VAEs was our idea. We denote a DMESN using PCA as a "DMESN-PCA", and one using VAEs as a "DMESN-VAE".

5.3.1 DMESN-PCA Results

The mean training time for the DMESN-PCA was about 8.5 seconds, with most times falling in the [7.5, 9.5] interval. (THIS IS UNORDERED CURRENTLY Interestingly, we find that the sparsity parameter has only a minor impact on the distribution of NRMSE on the validation data set. In Figure 4 we show the NRMSE distribution of two models with varying sparsity parameters, $\alpha = 0.1$ and $\alpha = 1.0$. Both have a uniform distribution in the range $\approx [0.0, 0.45]$; the model with $\alpha = 1.0$ has an improvement of ≈ 0.02 on the NRMSE..

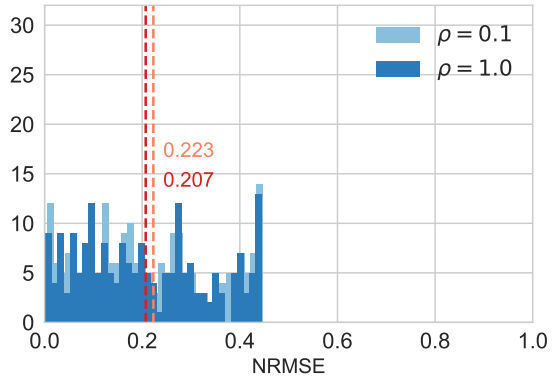


Figure 4: Histogram of NRMSEs achieved by the best DMESN-PCA model. α denotes reservoir sparsity.

Our experiments with various hyperparameters for the DMESN-PCA posit some interesting hypotheses. We found that regardless of the other hyperparameter settings, the echo parameter setting $0.5, ..., 0.1$ was found in *all* the top models, implying it was important for success. Additionally, all the best-performing models had reservoir sizes and weight-input scales that ascended or remain uniform - that is, for all reservoirs r_j, $j = 1, ..., R$, $r_j \leq r_{j+1}$ and $\gamma_j \leq \gamma_{j+1}$. Models with descending reservoir sizes and/or input weight scales generally exploded (i.e. huge autoregressive predictions, e.g. $|\hat{\boldsymbol{u}}(\cdot)| \geq 10^{10}$). We hypothesize this is because the ESP is dependent on the size of the reservoir and the scales of the signals. As shown in Figure 14 , the magnitude of the signals passing through the reservoirs tends to grow as they progress through the network. We believe a smaller reservoir is unable to handle these larger scales, as longer paths in a reservoir with weights s.t. $|W_{ij}^{res}| < 1$ help scale the data down: $|\hat{\boldsymbol{x}}^{(j)}(t)| \propto w^P \cdot |\hat{\boldsymbol{x}}^{(j-1)}(t)|$ where w is an arbitrary weight s.t. $|w| < 1$, and P is the length of a connected 'path' through reservoir. As P grows, the magnitude of a signal propagating through such a reservoir should lower at an exponential pace, thus helping with the 'explosion' problem we have experienced.

To understand the latent representations of the data that the PCA-encoders are learning, we plot the encoded signals transformed by the encoders from the output of each reservoir over 400 time steps, as shown in Figure 15. The amplitudes of the encoded signals grows proportionally with those of signals of the reservoirs in Figure 14. Compared to the DESN signals 12, the oscillations are less chaotic and contain patterns similar to those found in the vanilla ESN. This implies the encoding between reservoirs helps to remove noise from the signal, only supplying the minimum-encoding signal information.

5.3.2 DMESN-VAE Results

The performance of the DMESN was underwhelming. The mean training time for the DMESN-PCA was about 15 seconds, with most times falling in the [13, 18] interval. The extended train time is due to the multiple epochs required to train the VAEs. In Figure 7 we show the distribution of

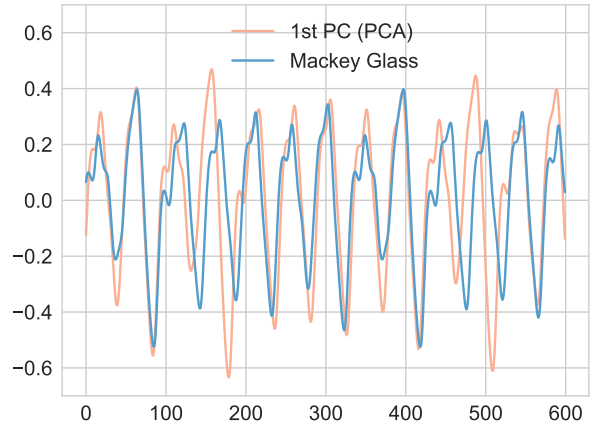


Figure 5: First principal component of the first encoder in the DHESN-PCA. Interestingly, it closely resembles the Mackey-Glass time-series data.

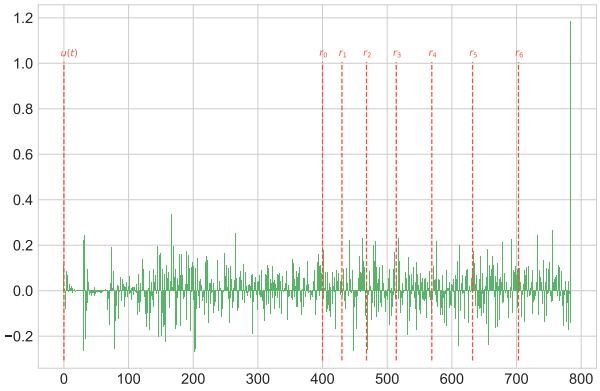


Figure 6: The output weights $\mathbf{W}^{out}$ learned by a DMESN-PCA.

the best DMESN-VAE model with varying training epochs of 2, 4, and 8. This plot shows well the trouble we had with getting the VAE model to work: there is only minor improvement when we increase the number of epochs, depsite the fact that training of the VAE converges at 15+ epochs. The fact that the VAE produces worse training results than the other deep models implies the VAE is harming the model rather than enhancing it. We hypothesize that the VAE is such an effective non-linear encoder that it actual 'undoes' the non-linearity of the reservoir prior to this. This is evident by the output weights which are large for the first VAE in the model and ≈ 0 for deeper VAEs. It is possible to make the weights uniform by either introducing heavy regularisation or training for less epochs, but both methods produce unsatisfactory results. Additionally, the VAEs in the model should have different latent variable encoding sizes; if the VAEs have the same encoding size they will usually learn the same latent representations. We conclude the 'stacked' architecture for VAEs is not useful as the best performance was with only one or two VAEs. Models with 8 layers (7 VAEs) performed the same as the models with 1-2 VAEs, once again showing the redundancy of the architecture.

5.3.3 General Comments

Reading through the DMESN paper (Ma et al., 2017), we noticed they bound the optimisation of the spectral-scale

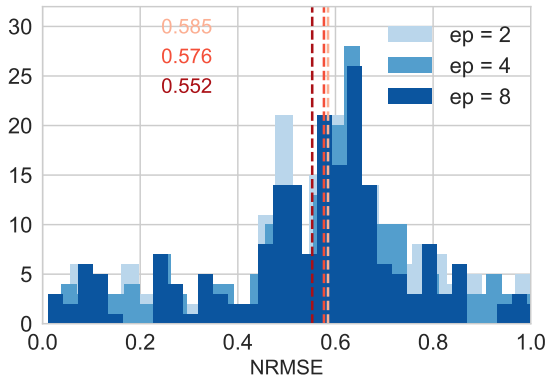


Figure 7: DMESN-VAE histogram validation. 'ep' is the number of epochs the VAE was trained on.

parameter to $[0, 1]$. Recent research (Yildiz et al., 2012) has shown that the spectral-scale doesn't need to be less than 1 to obtain the echo state property, and in fact in many cases setting it that low can wash away the input information in a similar vein to vanishing gradients. Additionally, the DMESN paper constrained the encoders to have equivalent-size encodings. For both PCA and VAE encoder architectures, we found that architectures where the size of the encodings varied performed better than architectures where the size of the encodings were all the same. Varying encodings protects against the potential problem of 'undoing' the non-linearity that the reservoir invokes between encoders (especially for non-linear encoders like VAE).

We found that the hyperparameters provided in the paper perform poorly, but this could be a result of unavoidable subtleties such as the differences between datasets.

5.4 Test Set Results

Model	Test NRMSE ± Std. Dev.
ESN	0.785 ± 0.254
EESN	0.647 ± 0.196
DESN	0.461 ± 0.273
DMESN-VAE	0.823 ± 0.205
DMESN-PCA	**0.344 ± 0.207**

Table 2: Autoregressive prediction results on a test set of 1000 time steps for the best ESN, EESN, LCESN, DMESN-PCA, and DMESN-VAE models. Errors and standard deviations were averaged over 30 runs.

As we said earlier, 1000 time steps of validation/testing data is quite a lot for a network to predict well fully autoregressively. To illustrate a fuller picture of these models' performances, we show a plot below containing the cumulative NRMSEs over time (e.g. the error shown at $t = 100$ is between the first 100 predictions and first 100 test data points, and the error at $t = 800$ is over the first 800 predictions). We also include some error bars.

On the next page, we show a plot of the best model, DMESN-PCA, autoregressively predicting the validation series (time steps 15,000 to 16,000):

These predictions yielded an NRMSE of 0.261.

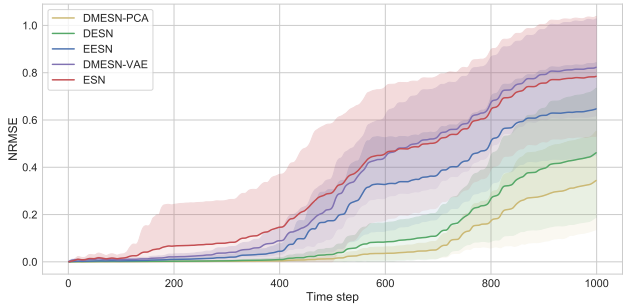


Figure 8: Test-set NRMSEs achieved by the best version of each model over time.

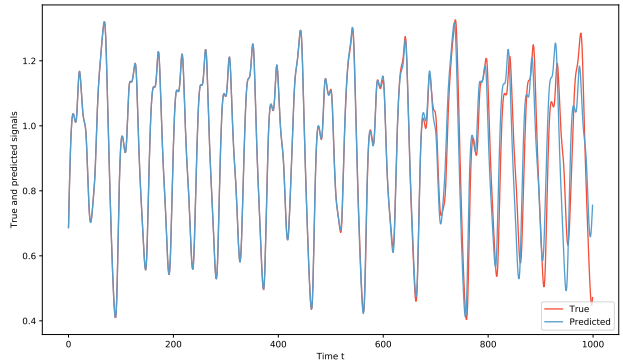


Figure 9: Autoregressive predictions of the test data by the best DMESN-PCA, with the true test data shown in red.

6 Conclusions

The goal of this report was to rigorously analyze these three modern ESN-based models and provide some comparative analysis of their performance on the Mackey-Glass time-series benchmark. Additionally, we also explored a fourth model that was a DHESN with VAE encoders. We conclude with some final general remarks on the comparative analyses we have performed through this report. We discuss the success of the models based on their performance on the test data as well as the complexity of the model. Firstly, from the test-set results it is clear that DMESN-PCA outperformed the other models. In terms of model complexity, DHESN-PCA also required the lowest number of output weights for training, making it the fastest model to train in terms of linear regression complexity. Interestingly, the trend seems to be as more constraints are added to the model architecture (e.g. EESN adds more reservoirs, DESN constrains interactions between reservoirs, DMESN de-noises the signal and reduces the reservoir dimensionality), the model improves. Models with more free parameters (such as a vanilla ESN) struggle with the stochastic influences of the weight randomization. This leads us to the conclusion that further success in ESN models will find better ways to remove noise between reservoir communications. Keeping randomization in the reservoir is important for 'edge-of-chaos' information processing, but communicating this noise to different models seems to be the salient issue for deep ESN models.

Originally, we hypothesized the DMENS with variational autoencoders would improve upon the DMESN with PCA encoders because VAEs have the capacity to perform non-linear encoding of the data with richer and more informative latent representations (Doersch, 2016). Contrary to our prediction, the DMESN-VAE performed poorly - even worse than DESN.

6.1 Future Work

It is discouraging that the more powerful VAE failed. As explained, we think this is due to the 'stacked' architecture of the deep ESN model. An alternative model would be to place a encoder after each reservoir in an EESN model such that encoders cannot disrupt one-another. This would allow for reservoirs that are uncorrelated to be de-noised by the encoders and potential provide even better prediction accuracy.

Another interesting enhancement would be to 'pre-train' each reservoir before feeding it into the reservoir in the next layer, as done in (Nichele & Molund, 2017). This would treat future reservoirs as 'error-correcting' reservoirs rather than the whole model having to delicately propagate a single signal.

References

Boeing, Geoff. Visual analysis of nonlinear dynamical systems: Chaos, fractals, self-similarity and the limits of prediction. 2016. doi: 10.3390/systems4040037.

Doersch, Carl. Tutorial on variational autoencoders, 2016. URL https://arxiv.org/pdf/1606.05908.pdf#cite.Kingma14a.

Farzad, M., Tahersima, H., and Khaloozadeh, H. Predicting the mackey glass chaotic time series using genetic algorithms. *Department of Electrical Engineering, Malek Ashtar of Tech University, Tehran, Iran*. URL http://saba.kntu.ac.ir/eecd/people/Tahersima/Predicting%20the%20Mackey%20Glass%20Chaotic%20Time%20Series/Predicting%20the%20Mackey%20Glass%20Chaotic%20Time%20Series.pdf.

Forman-Gornall, J. Exploring the potential of a novel time series predication algorithm. *Imperial College London, Department of Computing*, pp. 16, 6 2013. URL https://www.doc.ic.ac.uk/teaching/distinguished-projects/2013/j.forman-gornall.pdf.

Gallicchio, C. and Micheli, A. Deep reservoir computing: A critical analysis. *Department of Computer Science, University of Pisa Largo Bruno Pontecorvo 3 - 56127 Pisa, Italy*, 4 2016. URL https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2016-175.pdf.

Gallicchio, Claudio and Micheli, Alessio. Deep echo state network (deepesn): A brief survey. 12 2017. URL https://arxiv.org/pdf/1712.04323.pdf.

Hestenes, Magnus and Stiefel, Eduard. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 1952.

Jaeger, Herbert. The "echo state" approach to analysing and training recurrent neural networks. 148, 2001.

Jaeger, Herbert. Adaptive nonlinear system identification with echo state networks. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02. MIT Press, 2002.

Jaeger, Herbert. Echo state network. *Scholarpedia*, 2(9), 2007.

Jaeger, Herbert, LukoÅąeviÄĲius, Mantas, Popovici, Dan, and Siewert, Udo. Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks*, 20(3), 2007.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes, 2013. URL https://arxiv.org/abs/1312.6114.

Liebald, B. and Jaeger, H. Exploration of effects of different network topologies on the esn signal cross-correlation matrix spectrum. *School of Engineering and Science at International University of Bremen*, 2004. URL http://www.eecs.jacobs-university.de/archive/bsc-2004/liebald.pdf.

Ma, Qianli, Shen, Lifeng, and Cottrell, Garrison. Deep-esn: A multiple projection-encoding hierarchical reservoir computing framework. 11 2017.

Nichele, Stefano and Molund, Andreas. Deep reservoir computing using cellular automata, 2017. URL https://arxiv.org/abs/1703.02806.

Triefenbach, Fabian, Jalalvand, Azarakhsh, Schrauwen, Benjamin, and pierre Martens, Jean. Phoneme recognition with large hierarchical reservoirs. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 23*, pp. 2307–2315. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/4056-phoneme-recognition-with-large-hierarchical-reservoirs.pdf.

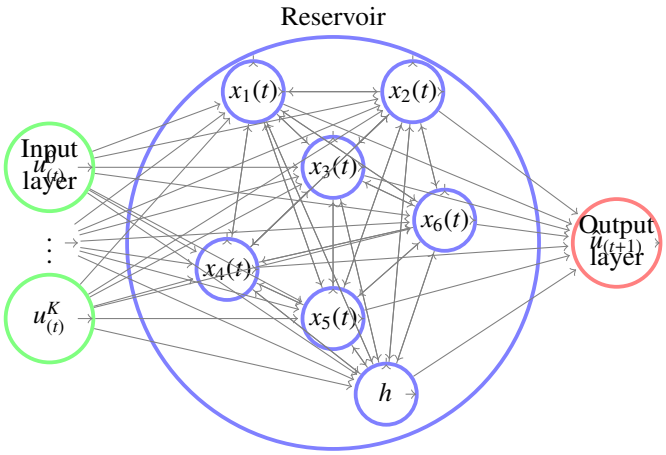
Wang, Q. and Li, P. D-lsm: Deep liquid state machine with unsupervised recurrent reservoir tuning. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2652–2657, Dec 2016. doi: 10.1109/ICPR.2016.7900035. URL http://ieeexplore.ieee.org/document/7900035/.

Yao, W., Zeng, Z., Lian, C., and Tang, H. Ensembles of echo state networks for time series prediction. In *2013 Sixth International Conference on Advanced Computational Intelligence (ICACI)*, pp. 299–304, Oct 2013. URL http://ieeexplore.ieee.org/document/6748520/.

Yildiz, I., Jaeger, H., and Kiebel, S. Re-visiting the echo state property. *Neural Networks*, 35:1–9, 2012.

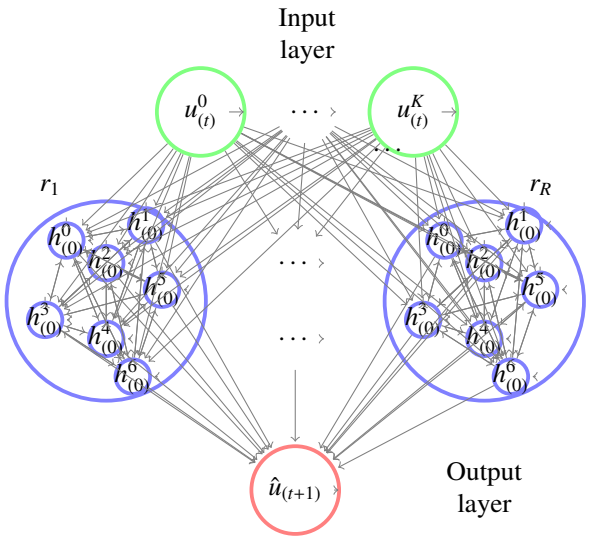
A Appendix A: ESN Zoo

ESN



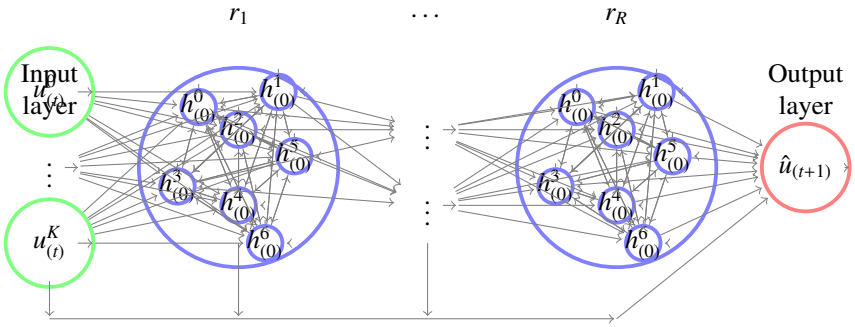
The 'basic' echo state network with one reservoir.

EESN



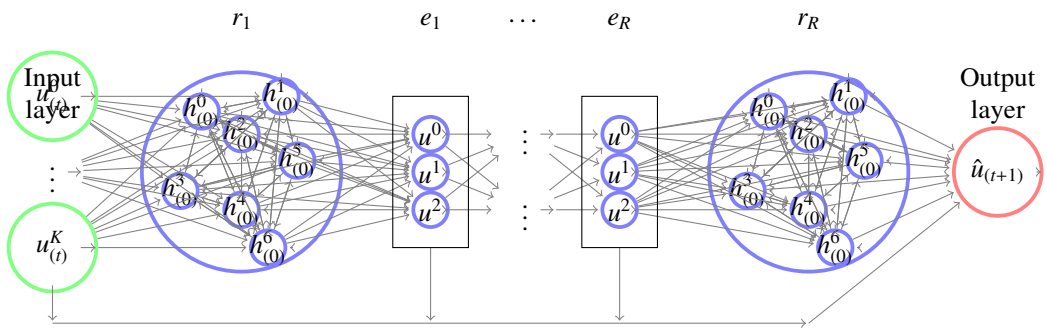
The ensemble echo state network: each reservoir takes the raw input signal $\boldsymbol{u}(t)$ as input, and linear regression is done on their combined states.

DESN



The deep echo state network: reservoir states are passed as inputs to the next reservoir, and the states of each reservoir are passed to the output layer for use in linear regression.

DMESN



The deep multi-encoding echo state network: reservoir states are passed through some encoder (e.g. PCA or VAE), and the encoded states are passed as inputs to the next reservoir.

Appendix B: Variational Autoencoders

Variational Autoencoders (VAEs) are considered a general enhancement over vanilla autoencoders (or denoising autoencoders). Unlike vanilla autoencoders, VAEs try to encode relationships between the latent variables rather than arbitrarily forcing a latent representation via bottlenecking. The loss function for a VAE is composed of two parts: the usually loss function used for penalising a vanilla autoencoder, which is some way to compare the true input to the generated output, and a second loss function which is a 'regularising' term which minimises the KL-divergence between a unit-norm Gaussian and latent variables:

$$L_1 = \frac{1}{N}\sum_{i=1}^{N}(\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i)^2 \tag{13}$$

$$L_2 = -\frac{1}{2}\sum_{i=1}^{N}(1 + log(\sigma_i) - \mu_i^2) Loss_{total} = L_1 + L_2 \tag{14}$$

The middle layer of the VAE is represented by two separated layers such that one layer encodes the vector of values $\boldsymbol{\mu} \in \mathbb{R}^N$ and the other layer encodes the vector of values $\boldsymbol{log(\sigma^2)} \in \mathbb{R}^N$, where N is the size of the layer (the number of latent variables) and μ_i and σ_i^2 parameterize a normal distribution: $\mathbb{N}(\mu_i, \sigma_i^2)$. When training the VAE, we sample from this normal distribution during the feed-forward process - this encourages the VAE to have small σ_i so that the μ_i encoded value is most likely; but the KL-divergence loss in the regularisation is adversarially preventing this by 'pulling' the σ_i to unit value. This forces a VAE to produce useful representations not just on the value μ_i but also *near* the value $\mu_i + \epsilon$, for $\epsilon << 1$. For all experiments we use a VAE with 3-hidden layers where the 2nd hidden layer is the latent representation. The 1st and 2nd layer use ReLU activation functions and the 1st layer uses linear activation functions. The final layer uses a linear activation function which is required to attempt to decode to the chaotic values of the Mackey-Glass time-series data.

Appendix C: Reservoir Signals

EESN

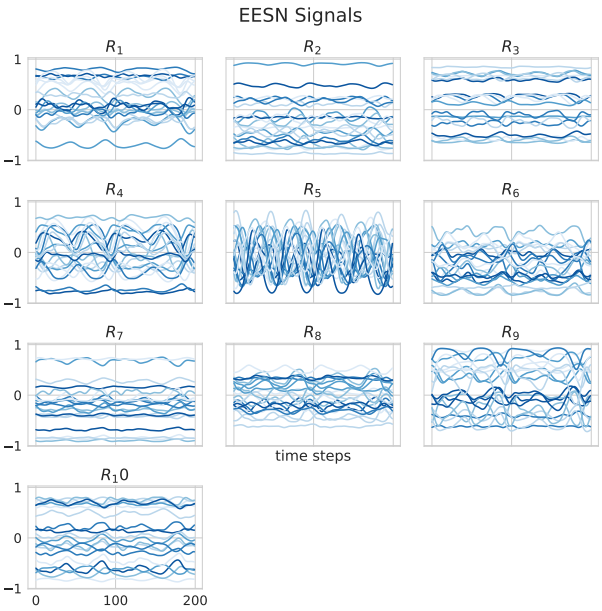


Figure 10: Reservoir signals $\boldsymbol{x}(t)$ over time in a DESN being fed MG validation data. The diversity of signals between reservoirs allows the model to incorporate both long and short term memories of reservoirs with varying β.

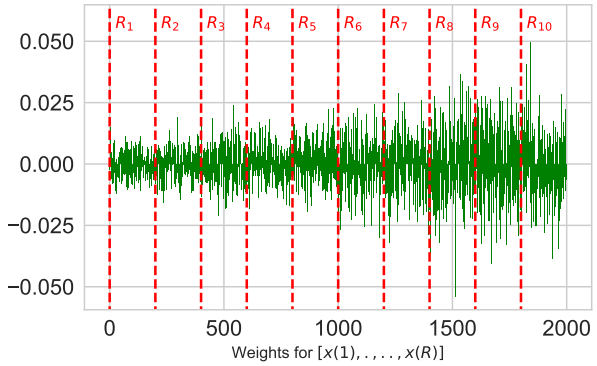


Figure 11: Output weights $\boldsymbol{W}^{out}$ learned by an EESN.

DESN

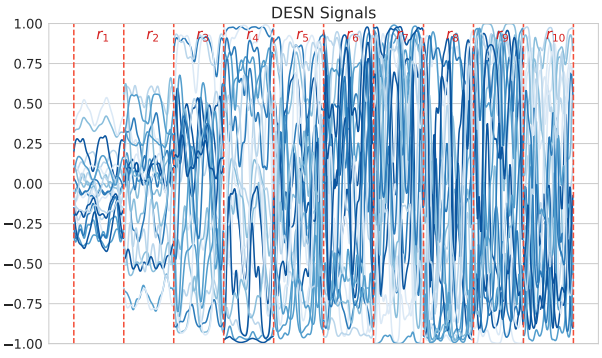


Figure 12: Reservoir signals $\boldsymbol{x}(t)$ over time in a DESN being fed MG validation data. It is interesting how the signal amplitude increases as it moves to deeper reservoirs.

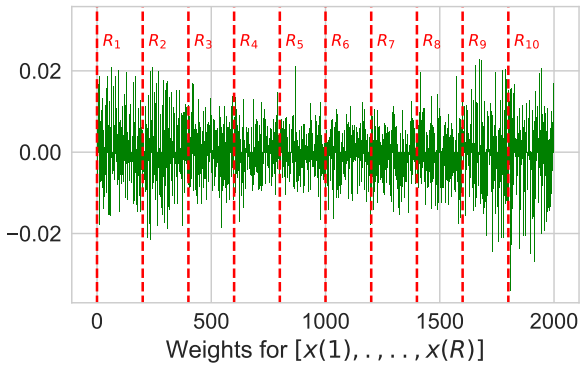


Figure 13: $\boldsymbol{W}^{out}$ weights for a DESN (excluding the bias and the weight for $u(t)$). Interestingly, it seems all of the signals are being used, even though those of the deeper reservoirs appear to be completely chaotic.

DMESN-PCA

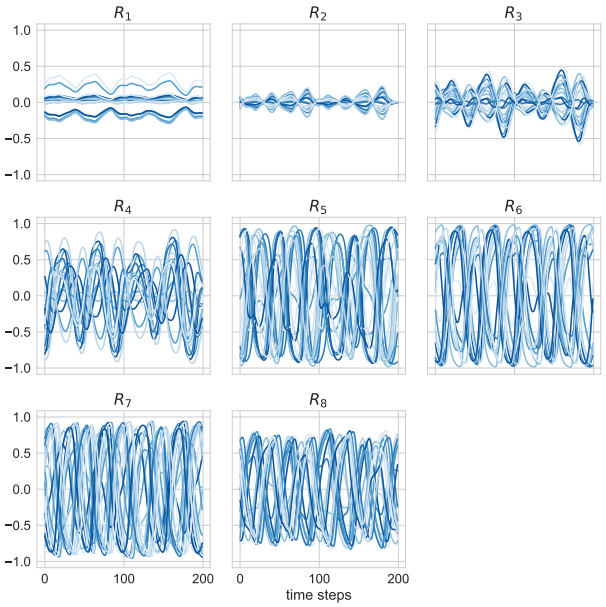


Figure 14: Reservoir signals $\boldsymbol{x}(t)$ over time in a DMESN-PCA being fed MG validation data. Notice how the magnitude of the signals tends to increase with deeper reservoirs.

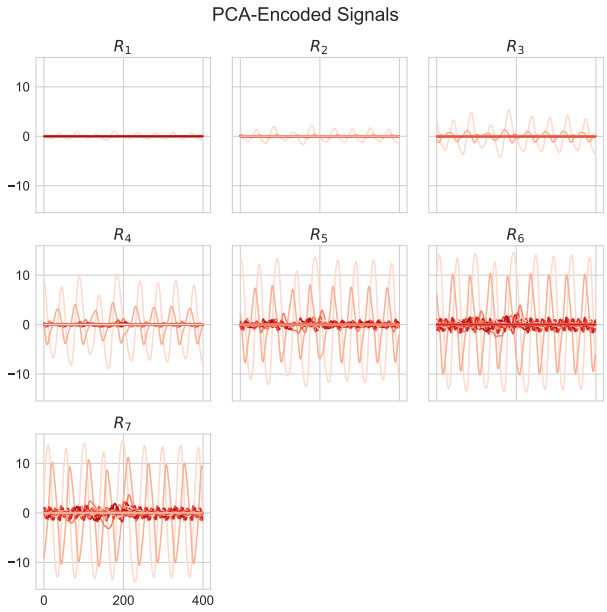


Figure 15: Encoded reservoir signals $\hat{\mathbf{x}}(t)$ over time in a DMESN-PCA being fed MG validation data. Note how the encoders turn the rather chaotic reservoir signals into a number of neat sinusoidal waves.