Time Series Analysis
MSc Fundamentals of Data Science

Ignasi Cos
Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Reservoir Computing Networks

1. Introduction

Reservoir computing (RC) is a family of Recurrent Neural Networks models which achieved excellent performance in time series forecasting and process modelling in the last few years. In machine learning, RC techniques were originally introduced under the name Echo State Networks (ESN).

ESNs were designed to efficiently process and model temporal data. A key feature that differentiates them from other RNNs is that the reservoir (layer) is not to be updated during training. In other words, the weights defining the connections between the neurons in the reservoir layer will remain the same during all the training process. The reservoir layer serves as a dynamic memory thus allowing the network to be able to process temporal context information in the following layer.

Therefore, the operation with ESNs may be summarized in the following three steps:

• First, the RCN is created and remains unchanged during training. This layer is called the reservoir. It is passively excited by the input signal and maintains in its state a nonlinear transformation of the input history.

• Secondly, the weights obtained from the reservoir layer are trained, usually by direct methods, in order to map them to the desired output. This layer is called the readout layer and it is, usually, composed by linear units.

• Finally, the output weights acquired in step two are used with new input data in order to compute new final outputs. In this project, we will be focusing in one particular Reservoir Computing model, named Echo State Network. This will, actually, be the model used later in the practical part as it has been shown that it has clear superiority over standard RC models in problems requiring effective propagation of input information over multiple time-steps, which is the particular case of the data we will be dealing with.

1.1. Reservoir Computing Networks

Reservoir computing (RC) is a learning technique used to deduce the underlying dynamics given a set of sequential data points. For instance, it may be able to predict future data by learning its dynamics or it may produce a related output sequence by learning the dynamics of the input sequence. The neural network is composed by a single hidden layer along with an input and output layers.

Reservoir computing is a recurrent neural network approach where all connections within the network are fixed except for the output ones, which are the only connections to be learnt. All connections except the output will remain unchanged at all times. As the output connections are the only ones to be adjusted, Reservoir Computing requires a lower number of parameters to be trained, making it computationally cheaper than other RNN approaches. This is thanks to its sparse structure and to the so-called echo states, which is the key concept behind reservoir computing.

2.2.1. Structure of the neural network

The structure of the RC network we are going to be working with can be divided as follows:

- K-dimensional input denoted by u.

- N-dimensional hidden layer denoted by x.

- L-dimensional output denoted by y.

The values of the different elements in the network will be updated at each time step n, which will define the dynamics of the network. Each of the elements can be defined as a multidimensional time series:

- *u(n)T = (u1(n), u2(n), ..., uK(n)))* is a K-dimensional vector giving the input units at time n.

- *x(n)T = (x1(n), x2(n), ..., xN(n)))* is an N-dimensional vector giving the internal units at time n.

- *y(n)T = (y1(n), y2(n), ..., yL(n)))* is an L-dimensional vector giving the output units at time n.

In order to determine the connections between the layers we have the connection matrices:

- Input connections, Win∈ M(ℝ)N×K, matrix with the connection weights from the input units to the internal units. Wini,j gives the weight of the connection from the *j*-th input unit to the *i*-th internal unit.

- Internal connections, W ∈ M(ℝ)N×N, matrix with the connection weights between the different internal units. Wi,j gives the weight of the connection from the j-th internal unit to the i-th internal unit.

- Feedback connections, Wback∈ M(ℝ)N×L, matrix with the connection weights from the output units to the internal units. Wbacki,j gives the weight of the connection from the j-th output unit to the i-th internal unit. In our case we have not used feedback connections as they didn’t improve the accuracy on the tests.

- Output connections, Wout∈ M(R)L×N, matrix with the connection weights from the internal units to the output units. Wouti,j gives the connection weight from the j-th internal unit to the i-th output unit.

Once the structure of the network is determined, we can introduce how the dynamics manifest in the network.
Internal states *X* are going to be updated at each timestep *n* as follows:

$$\boldsymbol{x}(\boldsymbol{n}+\mathbf{1}) = \boldsymbol{f}(\boldsymbol{W}\,\boldsymbol{x}(\boldsymbol{n}) + \boldsymbol{W}_{\boldsymbol{in}}\boldsymbol{u}(\boldsymbol{n}+\mathbf{1}) + \boldsymbol{W}_{\boldsymbol{back}}\boldsymbol{u}(\boldsymbol{n}+\mathbf{1}))$$

where *f* is the activation function.

It can bee seen from the equation that the hidden state is computed from the past one. The hidden State, therefore, is an echo of the past. This allows us to deterministically define the connection matrices **W***in* and **W**, in such a way that the only weights to be learned are those associated with the output (**W***out*).

On the other hand, the equation that computes the output *y* is as follows:

$$\boldsymbol{y}(\boldsymbol{n}+\mathbf{1}) = \boldsymbol{f}_{\boldsymbol{out}}(\boldsymbol{W}_{\boldsymbol{out}}\boldsymbol{x}(\boldsymbol{n}+\mathbf{1}))$$

where *f out* is the activation function.

The most commonly used activation function in RNNs is the hyperbolic tangent (*tanh)*, which we also used in the tests that we performed.

1.2. Echo State Networks

An Echo State Network (ESN) is an instance of the more general concept of Reservoir Computing. The basic idea behind the ESN is to benefit from the RNN structure, but without the training problems of a traditional RNN, such as vanishing/exploding gradient problems --- see sections next.
This is achieved with a relatively large reservoir of sparsely connected neurons. These connections are randomly assigned at the beginning and remain unchanged. In other words, the reservoir weights are not trained. Like reservoir connections, input connections are also randomly assigned upon initialization, and its weights are not trained. Input neurons feed the activations into the reservoir. On the other hand, the 7

output layer is fully connected to the reservoir, and its weights are the only ones that will be trained.
In training, once all the inputs have been fed into the reservoir, a simple application of linear regression to fit the relationship between the captured reservoir states and the target output targets.
The main idea is that the some of the sparse random connections in the reservoir will create loops between the neurons, allowing previous states to "echo" even after they have passed, so in case the network receives a novel input that is similar to one it has been trained with, the dynamics of the reservoir will start to follow the appropriated activation trajectory for this input, so it can provide a matching signal to the one it was trained with.

One of the main advantages of ESN is that only the output weights need to be trained, while still being able to capture complex temporal dynamics and to model properties of dynamical systems.
An important notion to keep in mind when working with ESN is that the **W** matrix needs to have a spectral radius |λmax| < 1, where λmax is the largest eigenvalue of **W** in module. Then, the network will not have any asymptotically unstable null states.

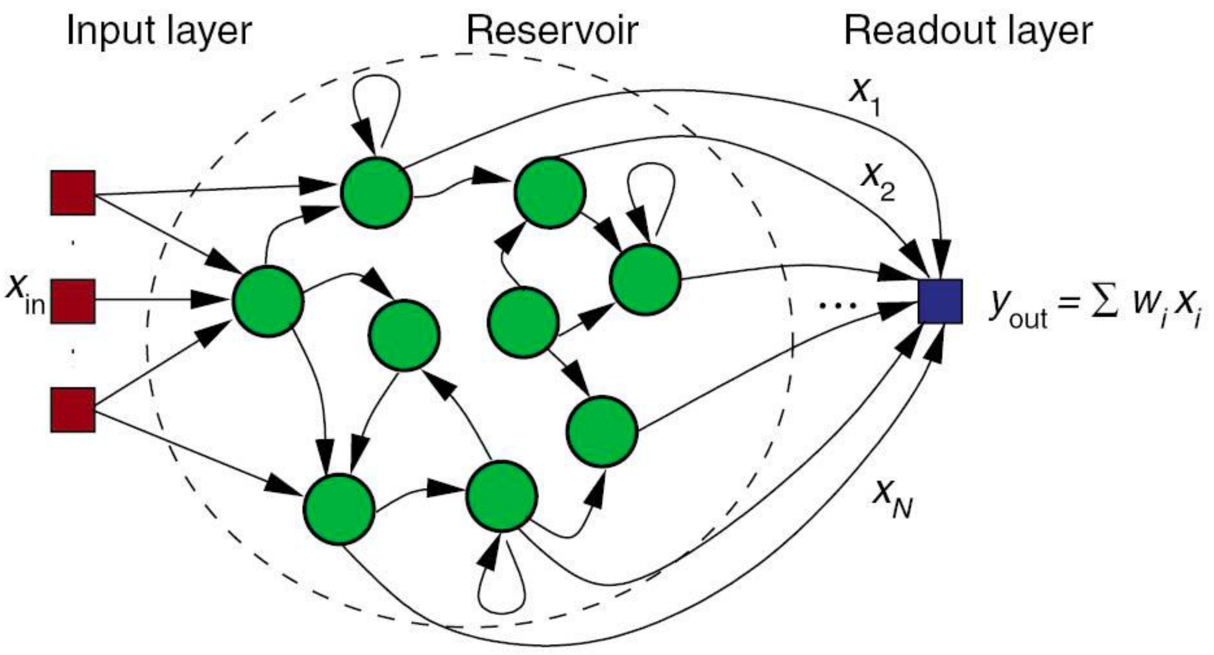

Figure 1. Echo State Network. Connections between the input layer and the reservoir as well as the connections between the reservoir units are sparse (as opposed to fully connected). Also, some of the reservoir nodes have recurrent loops that will act as a short-term memory. By contrast, the reservoir layer is fully connected to the output layer.

3.3. Our reservoir computing network

Extending from the network presented in section 2.2.1, we will demonstrate how we build the different layers and weight matrices. This process is similar for both tests.

Firstly, the input matrix (**W***in*) will be of size N×K, that is, the number of internal units and the number of input units. The data consists of 128 input channels. This means the value of K will be 128 as well. As a starting value, we will use 50 internal units. This is a hyperparameter that we will adjust later for better performance. So, to start, **W***in* will be a matrix of size 50x128.

The connections between the units will consist of a 10% probability of connection and their weights will be defined by a uniform distribution between [-1,1).

Secondly, the internal matrix (**W**) will be of size 50x50 (NxN) as we have defined that we will be using 50 internal units.

The connections of the internal units will also have a 10% probability of connection, but it's weights will be defined using a Gaussian distribution with a mean of 0 and standard deviation of 1.

Thirdly, in our problems do not use feedback connections to improve the performance, so **W***back* will not need to be defined.

Finally, the output matrix (**W***out*) will be fully connected to the inner matrix, and its weights will be defined through the regression method of our choice. The size of **W***out* is of L×N, where L is the number of output units. If we have 5 possible states, we need 5 output units, so the final size of **W***out* will be 5x50.

Since we do not define **W***back*, the state update equation will be defined as:

$$\boldsymbol{x(n+1) = f(W\,x(n) + W_{in}u(n+1)}$$

We will also check that the largest eigenvalue (|λmax|) is |λmax| < 1, to ensure we have Echo States.

2. Experiment and Data Description.

These neural data were collected during a reward retrieval task involving reaching and grasping.

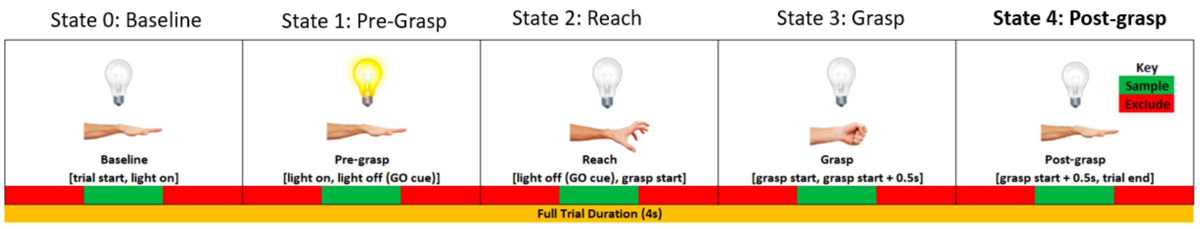

Figure 4: Timeline of a single trial.

The data set is divided into trials. Figure 4 shows the structure of a trial, which starts in a baseline condition, once the visual cue was active, it would initiate the movement towards the handle, afterwards if would reach for it, to grasp it. Finally, the primate would release the handle expecting a reward.

The data we will be using has been separated by motor state (5 states in total, as seen in Figure 4) and it includes data in which the handle to reach for was positioned in different orientations (0, 45, 90 and 135 degrees). The dataset consists of the LFP collected from 128 electrodes, sometimes referred to as channels, and it is separated by trial. Each trial has a duration of roughly 0.25 seconds (508 samples). Moreover, there were 140 trials in total resulting in an input matrix of size [128,508,140,5].

The data used for the Left Vs Right problem has 10 states, 5 for the left arm and 5 for the right arm, so it will be a matrix of size [128,508,140,10].

3.2. Data Pre-processing

We will be using two different datasets, one for each test. As mentioned above, the dataset for the Five-State Classification will be of shape [128,508,140,5], whereas the dataset for the Left vs Right test will be of shape [128,508,140,10]. For the latter, the first 5 trials are for the left arm and the next 5 for the right arm. Basically, the first 5 states are the corresponding ones for the Five-State Classification test.

To normalize the data, we first computed the absolute value for all the trials to get rid of the negative values.

One extra step with the data will be performed after is has gone through the network, and that will be to use only the mean of all 128 channels for each trial and state. In other words, a full trial consists of 128 channels, 508 readings and 5 states, by taking the mean of the readings we are removing the temporal component of the data. This yielded a significant improvement in the performance of the model. The data that will use for the regressor will be of shape [128 x 5 x n_trials].

3.2.1. Frequency Bands

To analyze performance of different brain functions, we will be filtering the data in different frequency bands, known to encode different brain processes, so we will be filtering by the following frequency bands:

Frequency bands	Hz
Baseline	4-500
Theta	4-7
Alpha	7-15
Beta	15-30
Low gamma	30-70
High gamma	70-100
Low ripple	100-150
High ripple	150-200
Low multi-unit	200-500

Table 1. List of frequency bands.

Each band will be band-passed and analysed in a separate fashion.

3. Your Task.

You are given a fully operational RCN that captures the variability of the task. Read the tutorial and analyse the code.

The code is distributed across three python classes:

1. Reservoir.py

This is the main class, managing the main body structure.

2. Network.py

This class composes the reservoir network, with all bells and whizzles.

3. Data.py

This class pre-processes the data (normalization, bandpass filtering, etc.) prior to training/testing.

Your ToDo:

1. What is the minimal N to reach top accuracy for this problem.

2. How do they probabilities of connection within the network condition and influence accuracy?

References:

[1] Maass W, Natschläger T, Markram H.(2002) Real-time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. Neural Computation 14(11), 2531–2560

[2] Jaeger, H. (2001) The "Echo State" Approach To Analysing And Training Recurrent Neural Networks. Technical Report GMD Report 148, German National Research Center for Information Technology.

[3] Rigotti M, Rubin DBD, Wang XJ and Fusi S(2010) Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses. Frontiers in computational neuroscience.

[4] Naval A, Cos I. (2020)

[5] Enel P, Procyk E, Quilodran R, Dominey PF. (2016) Reservoir Computing Properties of Neural Dynamics in Prefrontal Cortex. PLOS Computational Biology 12(6): e1004967.

Ignasi Cos
Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Tutorial on ESN

1. Revealing brain states with Reservoir Computing

The purpose of the project is to build a Reservoir Computing model capable of identifying different movement related states from a neural data set recorded from a non-human primate. To this end, we will perform the following analyses:

* Five-State Classification: where we will try to distinguish all five states at the same time, giving it a full trial to classify.

* Left vs Right: where we will try to distinguish when the data is from the primate using the left arm or the right arm.

Not only it is interesting to see if RC is capable of capturing the features recorded from specific brain regions, we will also see if the representation of the data after going through the model helps classify the states into their belonging classes better than without using the neural network.

2. Files description

* data.py : This file contains all the code regarding the data structures used and data processing

* network.py : This file contains the structure of the network and all the processes applied to it

* reservoir.py : This file runs the project, it builds the data and network using both the mentioned files above.

Note: the data files need to be in the same directory as the python files mentioned above.

3. How to run the code

There are 2 tests to perform:

* Five-State Classification: that uses either Logistic Classifier (log) or Linear Classifier (lin). Command -> 5s

* Left vs Right: that uses either Logistic Classifier (log) or 1NN Classifier (1nn). Command -> lvr

Both tests can apply a range of frequency filters:

Name - command
* Baseline - baseline
* Theta - theta
* Alpha - alpha
* Beta - beta
* Low gamma - lowgamma
* High gamma - highgamma
* Low ripple - lowripple
* High ripple - highripple
* Low multi-unit - lowmultiunit

In order to run the console we will use the following command:

```
python3 reservoir.py {test_command (string)} {filter_command (string)} {classifier_command (string)} {number of nodes (int)} {input probability connection (float)} {reservoir probability connection (float)}
```

An example for 5 state classification would be:

```
python3 reservoir.py 5s lowmultiunit lin 100 0.5 0.2
```

An example for Left vs Right classification would be:

```
python3 reservoir.py lvr theta 1nn 50 0.2 0.4
```

The following combinations are not supported:

* Using 1NN Classifier (1nn) with the 5 state problem

* Using Linear Classifier (lin) with the Left vs Right problem