Implementing an Echo State Network using Tensorflow

Taher Habib, Matrikel Nr. 975761

Abstract

The aim of this report is to describe a simple implementation of an Echo State Network (ESN) using TensorFlow, an open-source software library created by Google for dataflow and machine learning applications.

---

1. Introduction

Simply stated, an ESN is a recurrent neural network with sparsely connected hidden layer (connectivity ranging from 1% to 10% typically). They can be used to implement any complex time-series computation or realize a dynamical system [1, 2]. The main idea is to drive a random, large, fixed recurrent neural network with the input signal, thereby inducing in each neuron within this "reservoir" network a nonlinear response signal, and then combine a desired output signal by a trainable linear combination of all of these response signals [1].

ESNs consists of two conceptually different parts: a dynamical system with “rich/high dimensional” dynamics and a memoryless readout function. This idea is based on the following observation: If one excites a sufficiently complex recurrent circuit with an input signal, and looks at a later time at the current internal state of the circuit, then this state is likely to hold a substantial amount of information about recent inputs. In order to implement a specific task, it is enough that the readout function is able to extract the relevant information from the network state [3].

ESNs, together with Liquid State Machines (LSMs), are grouped under Reservoir Computing [1, 4].

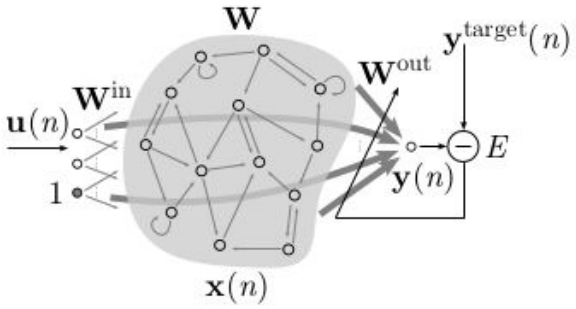
2. The Basic Model

I only discuss ESNs with regards to supervised temporal learning tasks where for a given input signal $\mathbf{u}(n) \in \mathbf{R}^{Nu}$ a desired target output signal $\mathbf{y}^{target}(n) \in \mathbf{R}^{Ny}$ is known. Here $n = 1, \ldots, T$ is the discrete time and T is the number of training data points. The task is to learn an RNN model with output $\mathbf{y}(n) \in \mathbf{R}^{Ny}$ which matches $\mathbf{y}^{target}(n)$ as well as possible, hence minimizing an error measure $E(\mathbf{y}, \mathbf{y}^{target})$. However, more important for the model is to generalize well to unseen data. E is typically a Mean-Squared Error (MSE), like [4]:

$$E(\mathbf{y}, \mathbf{y}^{\text{target}}) = \frac{1}{N_\text{y}} \sum_{i=1}^{N_\text{y}} \sqrt{\frac{1}{T} \sum_{n=1}^{T} \left(y_i(n) - y_i^{\text{target}}(n)\right)^2}$$

Let's consider a discrete-time neural network with N_u input units, N_x hidden (reservoir) units and N_y output units. Real-valued connection weights are collected in a N_u x N_x matrix $\mathbf{W}^{\text{in}}$ for the input units, in a N_x x N_x matrix $\mathbf{W}^{\text{res}}$ for the reservoir units, and in a N_y x N_x matrix $\mathbf{W}^{\text{out}}$ for the output units. There may be additional weight matrices for controlling the feedback from the output units to the input units. I shall refrain from discussing this aspect of the implementation since I will only describe the supervised learning method of training the ESN in this report.

The basic model of an echo state network is shown below[4]:



The learning task involves[1]:

2.1 Construct a random ESN.

(i) Create a random *reservoir*, using any neuron model. Here, leaky-integrated discrete-time continuous-value neurons are used, with update equation [4]:

$$\tilde{\mathbf{x}}(n) = \tanh\left(\mathbf{W}^{\text{in}}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)\right),$$
$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n),$$

where $\alpha \in (0, 1]$ is the leaking rate of the reservoir neurons. I use a leaking rate of 0.5 in my model.

The reservoir size N is task-dependent. In this model, I use N=100.

(ii) Connect the units in the reservoir either densely or sparsely. In ESNs, the connections in the reservoir are mostly sparse. I use a sparsity of 5%, which means that each neuron in the reservoir is connected to 50 other neurons only. Usually, the sparsity is even smaller for tasks that involve very many complex computations in order to save time, but this is an issue only in cases involving online training.

(iii) Connect input units to the reservoir by creating random all-to-all connections. I use (N_u=) 1 input node in my network.

(iv) Create output units. I use (N_y=) 1 output unit in my model. If the task requires output feedback, install randomly generated output-to-reservoir connections (all-to-all). If the task does not require output feedback, do not create any connections to/from the output units in this step.

2.2 Harvest reservoir states.

Drive the dynamical reservoir with the training data for times $n = 1,\ldots,T$ using a suitable activation. The activation function used in my model is tf.tanh. In tasks without output feedback like the current model, the reservoir is driven by the input $u(n)$ only. This results in a sequence $\mathbf{x}(n)$ of N-dimensional reservoir states which are nonlinear transforms of the driving input.

2.3 Compute output weights.

Since readouts from an ESN are typically linear and feed-forward:

$$\mathbf{Y} = \mathbf{W}^{\text{out}}\mathbf{X},$$

the output weights can be computed using linear regression, with respect to the teacher outputs $\mathbf{y}(n)$. In this model, I use the popular and universal method of *ridge regression* to carry out this operation, also known as regression with Tikhonov regularization:

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{\mathbf{T}}(\mathbf{X}\mathbf{X}^{\mathbf{T}} + \beta\mathbf{I})^{-1}$$

The implemented model uses a β value of 0.1. The term $\beta\mathbf{I}$ acts as a regularization term for preventing extremely large values of $\mathbf{W}^{\text{out}}$ [4]. Use these output weights to create reservoir-to-output connections. The training is now completed and the ESN is ready for use.

2.4 Other Important Remarks

As indicated in [4], one of the three main aspects of optimizing an ESN is input weight ($\mathbf{W}^{\text{in}}$) scaling. But this is generally done in the case where there are feedback connections coming in from output of the reservoir to the input. Since the current model does not have these feedback connections, I have not performed scaling of $\mathbf{W}^{\text{in}}$.

One of the most central parameters of an ESN is spectral radius of the reservoir matrix $\mathbf{W}^{\text{res}}$, i.e., the maximal absolute eigenvalue of this matrix. It scales the matrix $\mathbf{W}^{\text{res}}$. In other words, it scales the width of the distribution of its nonzero elements. For the ESN approach to work, the reservoir should satisfy the so-called echo state property: the state of the reservoir $\mathbf{x}(n)$ should be uniquely defined by the history of the input $\mathbf{u}(n)$ [2]. In order to ensure the echo state property of the network, it is desirable to have the spectral radius of the reservoir matrix near or less than 1 [4]. Large values of spectral radius can drive the reservoir into multiple fixed point, periodic, or even chaotic attractor modes spontaneously, thus violating the echo state property.

Here, I normalize the weight matrix to ensure this property. However, the echo state property often holds for spectral radius greater than 1 if the the dynamics of the reservoir are driven by non-zero inputs [4].

Another important aspect of training are the initial transients in the states of the network. Since $\mathbf{x}(n)$ data from the beginning of the training are contaminated by initial transients, they are discarded (not used for learning $\mathbf{W}^{\text{out}}$). The initial transient is a result of an arbitrary setting of $\mathbf{x}(0)$, which is typically $\mathbf{x}(0) = \mathbf{0}$. This introduces

an unnatural starting state. The amount of time steps to discard depends on the memory of the network (which in turn mostly depends on the reservoir parameters, α in the current model), and typically are in the order of tens or hundreds. The length of initial transients considered in the current ESN model is 400 data points (or 4 time-units).

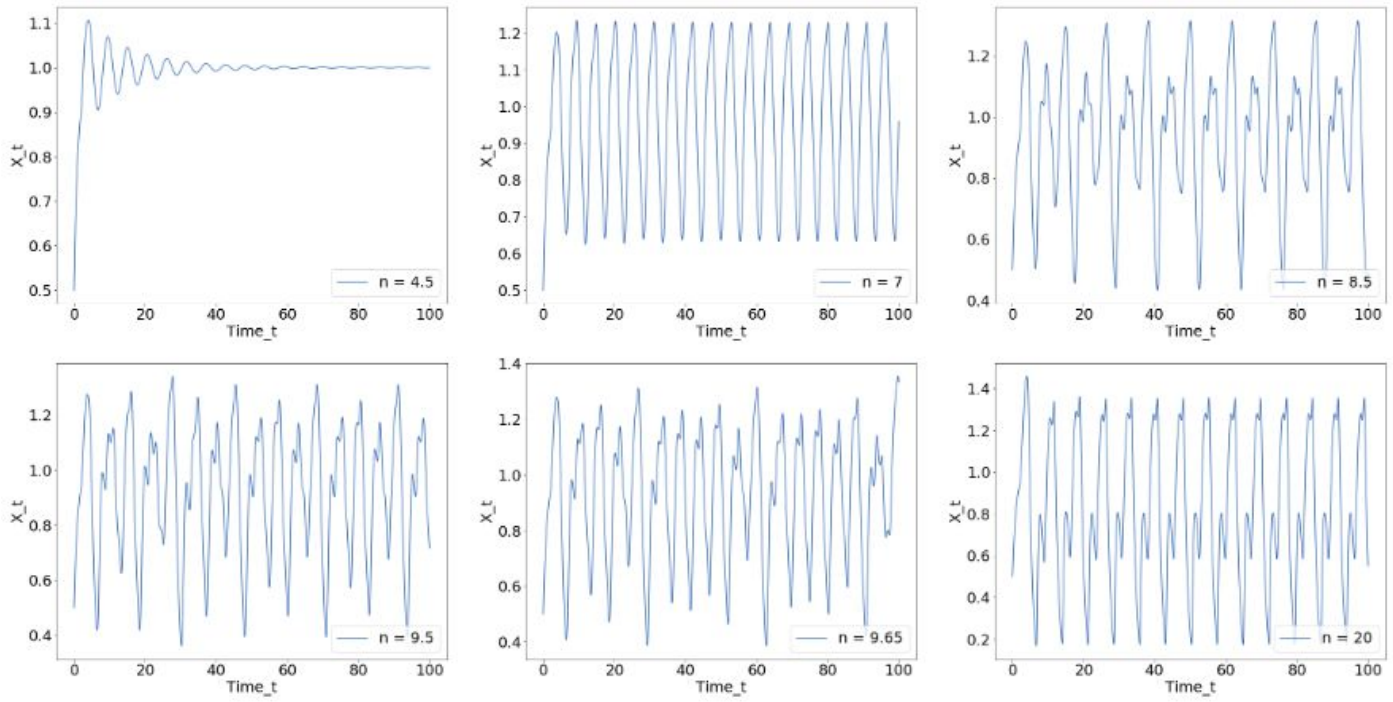
3. Description of the Implemented ESN Model in Tensorflow

I use the data from the Mackey-Glass Equation (MGE), which is a nonlinear time-delayed differential equation and a standard learning dataset for an RNN, to train my ESN on [5]. The MGE is given as:

$$\frac{dx}{dt} = \beta \frac{x_\tau}{1 + {x_\tau}^n} - \gamma x, \quad \gamma, \beta, n > 0,$$

where β , γ , τ , n are real numbers, and x_τ represents the value of the variable x at time $(t - \tau)$. Depending on the values of the parameters, this equation displays a range of periodic and chaotic dynamics. I have used the following values of parameters in my model: $\beta = 2$, $\boldsymbol{\tau} = 2$ and $\gamma = 1$ and the length of the generated time signal is 10000 time units, divided between 0 to 100 time-units.

The variable n of the differential equation acts as a bifurcation parameter — shown below in the figures are a few samples of the dynamics displayed by the MGE:
(The initial condition that I use in order to generate the series is: $x = 0.5$ for $t < 0$. Also, I use the library '*ddeint'* https://github.com/Zulko/ddeint for plotting these time-delayed differential equations).



I use these 6 different dynamics of the MGE to train my ESN. The six values of n used in the implementation are, respectively starting from top left in the figure above: 4.5 (fixed point orbit) , 7 (periodic orbit with a lower period), 8.5 (periodic orbit with a higher period), 9.5 (orbit at the imminence of chaos), 9.65 (chaotic orbit), 20 (periodic orbit).

In this model, the python class ESN is used to construct an instance of the ESN model or an ESN cell. Details of the class methods and instances can be found in ESN_Cell.py. Built-in tensorflow API of tf.nn.dynamic_rnn is used to compute the states of the ESN.

4. Results & Discussion

It must be noted that higher the value of α, lower will be the memory of the ESN reservoir neurons since the state-update equation, shown before, will have a greater contribution from the current value of the state activations **x**(n) (which gets scaled with α) as compared to the previous state value **x**(n-1) (which gets scaled with 1-α).

This, in turn, results in making the learning of the chaotic dynamics of MGE easier with higher values of α. This is indicated by the fact that the MSE values for n = 9.5 and 9.65 are lower for α = 0.75 as compared to the respective values for α = 0.1. This can be seen in the tables below:

	n	spectralradius	mse	alpha
1	4.50	1.0	8.156314e-08	0.1
2	7.00	1.0	3.969153e-05	0.1
3	8.50	1.0	6.248921e-05	0.1
4	9.50	1.0	7.860733e-05	0.1
5	9.65	1.0	8.214551e-05	0.1
6	20.00	1.0	2.120292e-04	0.1

	n	spectralradius	mse	alpha
1	4.50	1.000001	9.086773e-09	0.5
2	7.00	1.000001	3.400267e-05	0.5
3	8.50	1.000001	3.786021e-05	0.5
4	9.50	1.000001	4.387336e-05	0.5
5	9.65	1.000001	4.175199e-05	0.5
6	20.00	1.000001	6.073016e-05	0.5

	n	spectralradius	mse	alpha
1	4.50	1.000002	7.552657e-09	0.75
2	7.00	1.000002	1.852335e-05	0.75
3	8.50	1.000002	2.270342e-05	0.75
4	9.50	1.000002	2.719306e-05	0.75
5	9.65	1.000002	2.652599e-05	0.75
6	20.00	1.000002	5.267700e-05	0.75

The above observation is in agreement with the results observed in [2]. Further details of the activity of the reservoir neurons with time and figures depicting the fit of the network output with the original Mackey-Glass Equation time-series can be found by running the TensorFlow implementation.

References

[1] Herbert Jaeger. Echo state network. Scholarpedia, 2(9):2330, 2007.
[2] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.
[3] Bertschinger, Natschläger (2004). Real-Time Computation at the Edge of Chaos in Recurrent Neural Networks, Neural Computation 2004 16:7, 1413-1436.
[4] Lukoševičius M. (2012) A Practical Guide to Applying Echo State Networks. In: Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg
[5] Leon Glass and Michael Mackey (2010), Scholarpedia, 5(3):6908