Restricted Echo State Networks

Aaron Stockdill(✉) and Kourosh Neshatian

Department of Computer Science and Software Engineering,
University of Canterbury, Christchurch, New Zealand
aas75@uclive.ac.nz

Abstract. Echo state networks are a powerful type of reservoir neural network, but the reservoir is essentially unrestricted in its original formulation. Motivated by limitations in neuromorphic hardware, we remove combinations of the four sources of memory—leaking, loops, cycles, and discrete time—to determine how these influence the suitability of the reservoir. We show that loops and cycles can replicate each other, while discrete time is a necessity. The potential limitation of energy conservation is equivalent to limiting the spectral radius.

1 Introduction

Feed-forward neural networks are limited to learning "pure" functions. This excludes an entire class of problems which are time-dependent. Extending neural networks to handle this involves adding recurrency, and thus we have *recurrent neural networks*. These networks are capable of holding state, but it comes at the cost of them being difficult to train. Approaches to managing this complexity vary from unfolding the network sufficiently many steps to create a feed-forward network so that back-propagation can be used (this is called back-propagation through time [6]), or more advanced methods such as Kalman filtering [7].

A type of recurrent neural network is the *reservoir neural network* is becoming a common approach to learning. They are simple and fast to train, and as capable as other approaches [2]. A prominent model of reservoir neural network is the *echo state network* (ESN) described by Jaeger in 2001 [3]. Jaeger defines the ESN by a reservoir of neurons connected by synapses with weights that remain unaltered, in contrast to trained artificial neural networks, and restricts all the learning to a readout layer. Section 2 provides a description of ESNs.

An important component of a reservoir neural network is the eponymous reservoir. In the original definition, there are no restrictions on how the neurons connect—neurons can connect to any other, including themselves, and may form cycles. This gives the network powerful memory abilities, to the point that the past inputs entirely define the current state, after sufficient inputs. This is the *echo property* in ESNs. Because the structure of the reservoir has a strong influence on the expressive power of the ESN, this paper explores the effect of making structural changes: removing loops (wherein a neuron can output directly back into itself), cycles (as a sequence of nodes such that the output of a node eventually becomes input to itself), and the discrete time steps, and forcing a reservoir to be "conservative", where the outgoing weights for each neuron sum to one.

Although software allows arbitrary reservoirs, this is not the case in hardware. Thus this work aims to detail how the structure of a reservoir can impact on learning potential, and thus inform and guide neuromorphic hardware development. Work from Sillin et al. and other researchers has shown great potential for reservoir-style machine learning on novel hardware [5]. Because physical laws govern the structure of the network when it must exist in a physical system, these restrictions we impose become relevant—even when they are not a problem encountered in a purely software environment.

2 Echo State Networks

An echo state network (ESN) is a recurrent neural network first described by Jaeger in 2001 [3] that uses a reservoir of neurons that do not need training. Instead, the training happens in a readout layer. A more complete summary of ESNs is available from Lukoševičius [4], but can be fully defined by

$$\begin{aligned} \mathbf{y}(t) &= \mathbf{W}^{\text{out}}[1;\mathbf{u}(t);\mathbf{x}(t)] \\ \mathbf{x}(t) &= (1-\alpha)\times\mathbf{x}(t-1)+\alpha\times\tanh\left(\mathbf{W}^{\text{in}}[1;\mathbf{u}(t)]+\mathbf{W}\mathbf{x}(t-1)\right). \end{aligned} \tag{1}$$

The readout layer $\mathbf{W}^{\text{out}}$ is the map from the input vectors $\mathbf{u}(t)$ to the output vectors $\mathbf{y}(t)$, based on the internal state of the network $\mathbf{x}(t)$. The operator $[\cdot;\cdot]$ denotes vertical vector concatenation. The constant α is the leaking rate, referring to the mixing between current and previous output.

The ESN's readout layer is typically trained using a linear model, leading to a simpler recurrent neural network than otherwise possible with techniques such as back-propagation through time, or extended Kalman filters. The ESN also retains all the expressiveness in the recurrent neural network, performing comparably to these other training methods [2]. The linear model is for the ESN author to choose, but a common and recommended choice we use here is ridge regression [4]. Thus our learning is solving the matrix equation

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{\top}\left(\mathbf{X}\mathbf{X}^{\top}+\beta\mathbf{I}\right)^{-1}. \tag{2}$$

The matrix $\mathbf{Y}$ is the sequence of input vectors arranged horizontally, while the matrix $\mathbf{X}$ is the sequence of $[1;\mathbf{u}(t);\mathbf{x}(t)]$ vectors arranged horizontally.

The reservoir used by an ESN is the source of, and restriction on, its memory capacity. Normally, the only restriction placed on an ESN is the spectral radius, a measure of scaling performed by the reservoir. If the input could potentially contain a zero vector, then the spectral radius must be less than one [4]. Sometimes, the sparsity of the reservoir is also restricted, but this is for performance reasons. By placing further restrictions, we can reduce the 'power' of the ESN. Čerňanský and Makula demonstrated this in their work exploring the feed-forward ESN [1]. By removing both cycles and loops in the reservoir, they show an ESN becomes equivalent to a feed-forward neural network with inputs representing up to n steps back in the input history, where n is the number of neurons in the reservoir.

3 Exploring Reservoir Variations

We have identified four sources of memory in an ESN: leaking, cycles, loops, and the discrete time steps. Because leaking is inherently 'outside' the reservoir, this is a simple addition to any network. Any state read from the network can be stored to mix in with the next state before being sent to the readout layer. Thus we will not consider leaking any further.

Cycles are when there is a sequence of neurons $n_1 n_2 \dots n_k n_1 n_2 \dots$. Loops are an edge that connects neuron n_i back to itself. The discrete time steps are when the state of a neuron at time t receives information from its neighbours from time $t-1$. This final property works in tandem with the first two to exploit the state of the network and provide the memory so vital in its power.

By removing these features, and combinations thereof, we potentially weaken the ESN, but in doing so make it more closely resemble the hardware implementations available. Two distinctions remain between the "fully-weakened" ESN and a hardware network. First, the hardware network is updating the connection weights while the network is running. Modern neuromorphic hardware is not composed of static resistors, but often some variation of memristive hardware, which has a dynamic resistance depending on the history of voltage or current. To explore the effect this might have, we introduce a *wobbling weights* matrix, which changes its weights based on the previous input across it, much like memristors. Second, the software reservoir is able to amplify and suppress energy arbitrarily, whereas hardware must conserve electricity. To remove this, we force the output weights of each neuron to sum to one.

3.1 Loops

Loops are a source of memory for the network. Because the neuron now has explicit access to its own state at time $t-1$, it creates a type of weighted average, effectively giving each neuron total memory of past inputs. Cycles give the same effect, but the tighter effect of the loop is more easily emulated in hardware solutions by sensors and external voltage sources.

As shown by Čerňanský and Makula, removing both cycles and loops reduces an ESN to a feed-forward network with delayed-time inputs [1]. The memory of the network is limited by the longest chain. The network was still capable of solving the typical sorts of problems such as Mackey-Glass because the memory requirement is by convention set at 17 steps, and the reservoirs are trivially made larger than this. Removing just one of loops or cycles will not cause the same reduction in expressive power for temporal datasets. By removing only loops and not cycles, there is no immediate loss of power—any memory a loop supported is replicated with a cycle, but with a k-step delay, where k is the length of the shortest cycle through a neuron. Thus learning may slow, but not stop.

Consider a simple network of two neurons connected by a directed edge in both directions. If no loops are available, it is not immediately possible to mix the input to neuron i at time t, denoted $u_i(t)$, with $u_i(t-1)$. But we can mix $u_i(t)$ with $u_i(t-2)$. Thus the length of the cycle through neuron i is two, so there is a two-step delay in the network. In the meantime, neuron j is mixing $u_i(t-1), u_i(t-3), \ldots$. The readout layer can mix both streams, thus mixing $u_i(t)$ for all t. This scales appropriately for cycles of length k.

3.2 Cycles

Cycles provide the network with 'infinite' memory. Removing cycles is an important research question, because hardware implementations are unable to recreate cycles. Kirchhoff's Voltage Law limits the amount of energy in a circuit, and forces conservation. That is, a junction is unable to amplify a signal, and so there cannot be cycles in the network. Having cycles would imply an infinite sequence of groups where the potential difference drops forever, leading to an impossible infinitely-descending structure:

$$V_1 > V_2 > \cdots > V_k > V_1 > \cdots \implies V_1 > V_1 \lightning \tag{3}$$

If cycles were to form, energy would cycle forever and become infinite, something not possible in a physical circuit.

Now, having removed cycles, infinite mixing of inputs is not available to every neuron, but infinite mixing of input at neuron n for input to neurons $m < n$ is available because we have not yet excluded loops. By modifying input to be repeated (i.e. $\mathbf{u}(t) \mapsto [\mathbf{u}(t); \mathbf{u}(t)]$), the inputs to neurons $m < n$ are also available at neurons $o > n$. Thus having loops can be made equivalent to having cycles, although particular mixes may not be available within the same number of time steps. This is important as a device which mixes the previous voltage across a memristor with the present voltage is conceivable.

As an illustrative example, consider a network that once contained the cycle of two neurons m and n such that $m \to n \to m$. Normally we could infinitely mix $u_m(t)$ with $u_n(t-2k-1)$ and $u_m(t-2k)$ for any natural number k, and vice versa, by allowing the inputs to cycle around each other. By removing cycles, such a structure is unavailable. Instead, we can simulate it with $m \to n \to m' \to n'$ such that $u_m(t) = u_{m'}(t)$ and $u_n(t) = u_{n'}(t)$, and every neuron also loops back into itself. It is now possible to mix $u_m(t)$ with $u_n(t-k)$ and $u_m(t-k-1)$ for any natural number k, as it now occurs further back in the network, and the cycle acts as an infinite internal delay mechanism for the input. This is a stronger guarantee than necessary, but does ensure the desired effect of cycles.

3.3 Conservation of Energy

The restriction of conservation of energy is not a restriction at all. It limits a network in the same manner as the spectral radius, the spectral radius being the largest absolute eigenvalue of the weights matrix. By ensuring that a neuron's outputs sum to one, we have effectively forced each column in the weights matrix to sum to one. The eigenvalues of matrix $\mathbf{W}$ are the same for $\mathbf{W}^\top$, so we can consider the matrix $\mathbf{W}^\top$ with row sums equal to 1. For some $\mathbf{v}$, we have

$$\mathbf{W}^\top \mathbf{v} = (\mathbf{w}_1^\top \mathbf{v}, \mathbf{w}_2^\top \mathbf{v}, \ldots)^\top = (\mathbf{w}_1 \cdot \mathbf{v}, \mathbf{w}_2 \cdot \mathbf{v}, \ldots)^\top \tag{4}$$

Algorithm 1 Propagate the input $\mathbf{u}(t)$ over the reservoir defined by $\mathbf{W}$

```
1: procedure PROPAGATE($\mathbf{W}$, $\mathbf{W}^{\text{in}}, \mathbf{u}(t)$)
2:     $\mathbf{v} \leftarrow \mathbf{W}^{\text{in}}\mathbf{u}(t)$
3:     $\mathbf{o} \leftarrow (0, 0, \ldots, 0)^\top$
4:     for all $n \in$ toposort($\mathbf{W}$) do
5:         $s \leftarrow v_n$
6:         for all $m \in$ predecessors($n$) do          ▷ finds all nodes with edges into $n$
7:             $s \leftarrow s + o_m \mathbf{W}_{n,m}$          ▷ $\mathbf{W}_{n,m}$ is the weight from $m$ to $n$
8:         end for
9:         $o_n \leftarrow \tanh(s)$
10:    end for
11:    return $\mathbf{o}$
12: end procedure
```

Given that $\mathbf{w}_i \cdot \mathbf{v} = \|\mathbf{w}_i\|\|\mathbf{v}\| \cos\theta$, $\|\mathbf{w}_i\| \leq 1$, and $-1 \leq \cos\theta \leq 1$, the largest absolute scaling possible by $\mathbf{W}^\top$ (and thus also by $\mathbf{W}$) is 1. Hence conservation of energy is equivalent to specifying a spectral radius of at most one. This is not an issue: ESNs are only guaranteed to work for spectral radii below one [3].

3.4 Discrete time steps

The discrete time nature of an ESN is the fundamental feature of its memory. This is also a difficult feature to replicate in hardware. Because a circuit will have the electricity pass through at significant fractions of the speed of light, no matter how rapidly we switch the input voltage, we are essentially saturating the network with the same signal millions of times before switching. Because of this speed disparity, a hardware network will essentially not contain discrete time steps, instead it will function more like a traditional feed-forward neural network, which we will call the one-hop reservoir, where the input $\mathbf{u}(t)$ is influencing the entire network at time t, but inputs $\mathbf{u}(s)$ from times $s < t$ are not in the network. The difference is now, there is no new information written to the network before the propagation is complete. Because of this distinct termination, the network is not allowed to have cycles or loops. Algorithm 1 outlines the propagation.

Because there is now no state in the network beyond the leaking rate, the network will be unable to learn any function requiring knowledge of previous time steps. Essentially, we remove the echo property. The state now depends solely on the random initial weights, not the history of previous inputs as required by an ESN. This network is now an untrained feed-forward neural network. Hence this reservoir is incapable of learning any of the time-series problems it was designed to solve. While there are potential applications for traditional machine learning, by training an equivalent neural network in software and 'burning in' the weights to hardware, this is not a suitable use for memristors—they will update their weight, and move away from their desired weight.

This comparison is not fair, because a network of memristors *does* maintain a state, because the weights *do* get updated. The question then becomes does the memristor's state act as a suitable substitute for the ESN's discrete time steps? The answer would seem to be no. By making the ESN have a 'wobbling' weights matrix to simulate the updating conductances of the memristors and switches, we handicap the readout layer by removing the underlying assumption of regression—for a given input x, there is a function $f(x)$ that we attempt to find. Because f is a function, each x uniquely maps to some y. By changing the weights matrix, we change the function we are trying to fit, and so prevent the linear regression from successfully fitting the training data.

4 Conclusion

We have presented four important sources of memory in an ESN: leaking, loops, cycles, and discrete time. While removing both loops and cycles is known to undermine the infinite memory capability in a reservoir, removing just one leaves the reservoir sufficiently powerful for all learning, assuming some modifications to inputs. The single most important feature of a reservoir neural network is the discrete time step nature of input propagation. Should this be unavailable, as is likely in hardware implementations, the reservoir will cease to function as an effective learner. The potential for self-updating hardware such as memristors does not provide the necessary memory to overcome the lack of discrete time propagation. A primary concern in hardware implementations of reservoir neural networks is the strict requirement of energy conservation. However we have shown this to be no more severe than spectral radius scaling.

References

1. Čerňanský, M., Makula, M.: Feed-forward echo state networks. In: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., vol. 3, pp. 1479–1482 (2005).
2. Čerňanský, M., Tiňo, P.: Comparison of Echo State Networks with Simple Recurrent Networks and Variable-Length Markov Models on Symbolic Sequences. In: Sá, J.M., Alexandre, L.A., Duch, W., Mandic, D. (eds.) ICANN 2007. LNCS, vol. 4668, pp. 618–627. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74690-4_63
3. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. GMD Report (2001)
4. Lukoševičius, M.: A practical guide to applying echo state networks. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 7700, 2nd edn, pp. 659–686. Springer, Heidelberg (2012). doi:10.1007/978-3-642-35289-8_36
5. O Sillin, H., Aguilera, R., Shieh, H.-H., Avizienis, A.V., Aono, M., Stieg, A.Z., Gimzewski, J.K.: A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing. Nanotechnology **24**(38), p. 384004 (2013)
6. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proc. IEEE **78**(10), 1550–1560 (1990)
7. Williams, R.J.: Training recurrent networks using the extended kalman filter. In: International Joint Conference on Neural Networks 1992. IJCNN, vol. 4, pp. 241–246 (1992)