Reservoir Computing: a Rich Area for Undergraduate Research

Dr. Thomas E. Gibbons
CS/CIS Department
College of St. Scholastica
Duluth, MN 55811
tgibbons@css.edu

Abstract

Reservoir computing is a neural network model that has been developing over the last ten years. With its origins in two separate models, Echo State Networks and Liquid State Machines, reservoir models have become more unified in the last three years, particularly since the 2007 special issue of Neural Networks dedicated to this field.

Reservoir computing is based on relatively simple principles and can be modeled with software researchers share with the community, which makes it an excellent candidate for undergraduate research. It is relatively new and has a rich area of questions that have not been answered about the model and its application. Many of these questions can be easily understood by an undergraduate student. Finally, the model appeals to a wide variety of students, since its theoretical aspects involve areas of linear algebra which often appeal to mathematically inclined CS students, while its applications are often related to areas of cognitive science, economics and robotics which often appeal to applied CS students.

This paper will follow the following framework: Sections 1 and 2 will review the traditional neural network model and provide an overview of reservoir computing models. Sections 3 and 4 will describe current research areas and avenues for possible undergraduate research. The final section will describe simulation software available to anyone interested in research reservoir systems.

1 Traditional Neural Network Model

Neural networks have been an active field of research in artificial intelligence for the past few decades. Researchers have appreciated the model's basis in biological systems while providing the symbolic formalism needed for mathematical analysis. Neural networks have been used to address a wide range of problems and applications from abstract problem solving to modeling specific biological functions. While there have been a number of different neural models proposed, a majority of research has focused on a simple three layer, feed-forward network trained with a version of the back propagation learning algorithm.

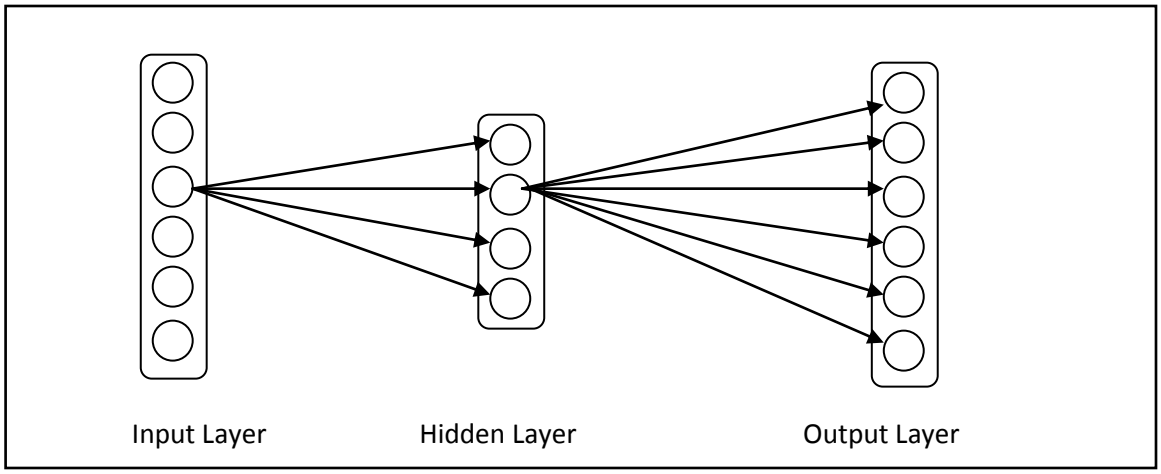

Figure 1: Standard three layer, feed-forward, neural network.

In this model, each node in the input layer is connected to every node in the hidden layer, and each node in the hidden layer is connected to every node in the output layer. This is a feed-forward network, so there are no recurrent connections between models in the same layer. There are a number of different node types used by these models, but generally a node receives weighted connections from a number of other nodes. Each node calculates a weighted sum of the inputs and determines its own activation using a non-linear function of this sum.

Neural networks are often used as classifiers where the task is to assign input patterns to the corresponding output. Generally the inputs are divided into groups, and the goal is to assign new inputs to the correct group. For example, the inputs could be a bitmap image of a single digit, and the bitmaps would be assigned to the groups associated with each digit. In this classification task, an output node is assigned to each group, and the network is trained to activate the correct output node for each input pattern. Figure 2 shows an example of images of digits being classified into groups corresponding to the number the images represent.

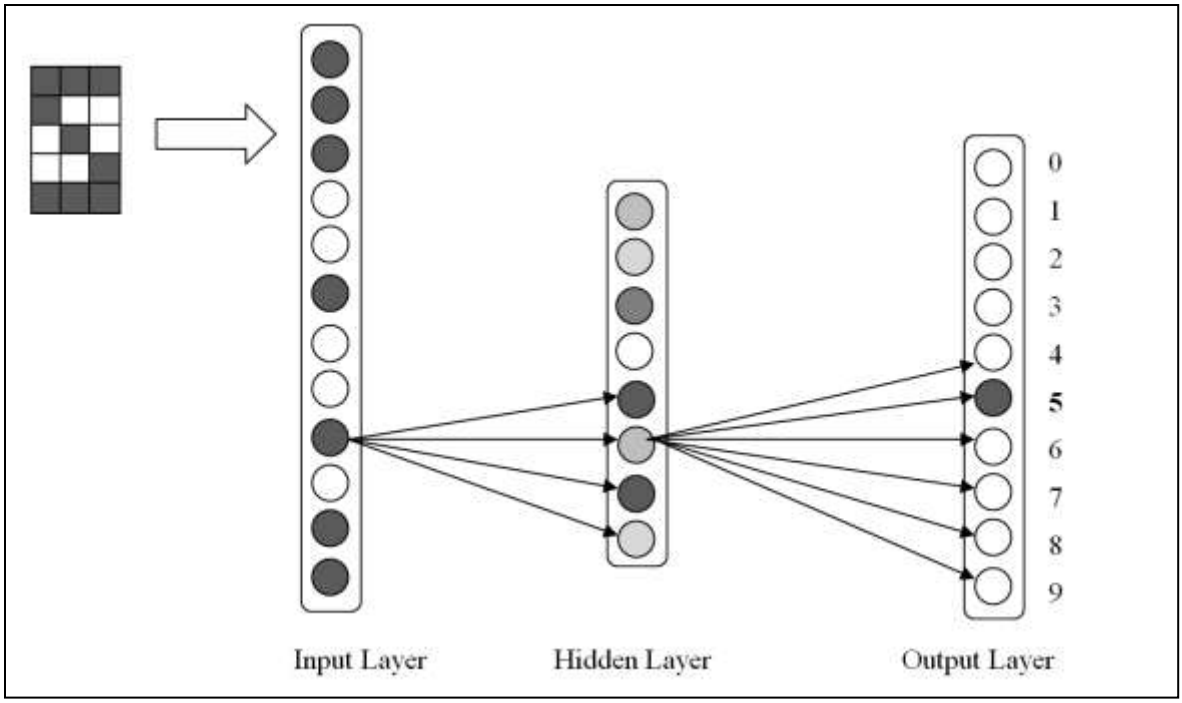

Figure 2: Categorization example of digit bitmap example

Training is supervised, meaning that a set of training inputs are provided with their corresponding group assignments. For each training input, the input layer is set, and then the activations of the nodes in the hidden layer are calculated. Once the hidden layer activations are determined, the activations of the output layer can also be calculated. Since the weights assigned to the connections between nodes determine the nodes' activations, the back propagation training algorithm adjusts the weights. During training, weights on all connections are adjusted to increase the activation of the correct output node and decrease the activation of all the other nodes. The training patterns are repeated over and over until errors in the output activations drop below a threshold level.

Some of the challenges faced by this three layer, feed-forward network model are a result of the structure. Since the model requires full connectivity between layers, the number of connections increases with the square of the number of nodes. Since a different output node is generally assigned to each group of patterns, the size of the output layer can grow large in certain tasks. While this works fine for digit recognition, many more output nodes are required for full ASCII character recognition, and the model cannot handle word recognition because the number of output nodes would be too large.

Adding additional layers is also difficult because, although the back propagation algorithm can work with multiple layers, its efficiency decreases as the number of layers increase. Generally this efficiency loss limits models to three layers. These constraints on the number of nodes in each layer and the number of layers limit the applications of these models.

Another challenge faced by this standard neural network model relates to how inputs are processed. Because this model is strictly feed-forward, the update process is simplified. Once the input nodes' activations are set, the hidden layer activations only have to be calculated once and similarly with the output activations. When a new input pattern is presented to the network, the past activations of the hidden and output layers do not affect the calculations of the new activations. This makes each input pattern independent of the other input patterns. While this is beneficial in some cases, many real world patterns are of a temporal nature. Vision and speech recognition systems process sequences of related inputs. While it is possible to map these temporal sequences of patterns onto larger non-temporal patterns, this increases the size of the input patterns significantly and often makes the number of connections unmanageable.

2 Reservoir Computing Model

In 2001, a new network model was developed independently by two different researchers. Wolfgang Maass proposed Liquid State Machines (Maass, Natschläger, & Markram, 2002) and Herbert Jaeger proposed Echo State Networks (Herbert Jaeger, 2001). While there were differences in these models, they shared the same fundamental structure. Today these two models have been incorporated under a unified model, Reservoir Computing. Details on the development of these two models and a summary of the current state of reservoir computing can be found in (Lukosevicius & Jaeger, 2009) and (Verstraeten, Schrauwen, D'Haene, & Stroobandt, 2007).

The reservoir computing model introduces two significant changes to the traditional three layer feed-forward network. First, full connectivity between layers is no longer required. Each node in the hidden layer is connected to a random set of nodes in the input layer, likewise for the output layer. Second, recurrent connections are included. Each node in the middle or hidden layer is connected to a random number of other nodes in the same layer, besides the connections to the input layer. These recurrent connections change the dynamics of this middle layer. The current activations of this middle layer now affect the updated activations. When a new pattern is presented in the input layer, the activations of the middle layer now combine the previous state middle layer with the new input pattern. Because of this, the middle layer is now renamed the reservoir because its activations encode parts of all the recent input patterns.

Some reservoir models also have recurrent connections from the output layer back to the middle or reservoir layer, but these recurrent connections are not required in this model. Reservoir models also incorporate a number of different node types similar to the variety of node types used in the traditional model.

The recurrent connections change the dynamics of the network, allowing it to process temporal patterns and also requiring multiple updates to the activations for each input pattern. For non-temporal patterns, the activations of the reservoir and output nodes must be calculated over multiple iterations, since values of the activations now affect these calculations. More commonly, reservoir models are used to process temporal patterns where the activations of the reservoir and output nodes are repeatedly calculated and the input pattern changes over time.

The reservoir is formally defined by the following: Given a reservoir and an input pattern $U = \{u_1, u_2, \dots, u_N\}$ defined over discrete time $t \in \mathbb{Z}$ where *u(t)* is the input vector at time *t*, the reservoir state, x_i , derived from $u_i(t)$ is defined as: Given the sparse matrices defining the weights between the input vector and the reservoir, W_{inp}^{res}, and within the reservoir, W_{res}^{res}, and given the vectors $U = \{u_1, u_2, \dots, u_N\}$ containing the input activations and $X = \{x_1, x_2, \dots, x_N\}$ containing the reservoir activations defined over discrete time $t \in \mathbb{Z}$ where *u(t)* is the input vector at time *t*, the reservoir state is defined as:

$$X(t+1) = (1-r)X(t) + rf\left(k_{inp} W_{inp}^{res} U(t+1) + k_{res} W_{res}^{res} X(t)\right) \quad (1)$$

Here *r* is a constant defining the decay rate and k_{inp} and k_{res} are constants controlling the relative strengths of the input and reservoir weights. The function *f()* is generally a non-linear sigmoid function, such as *tanh()*, or a binary threshold function returning 1 when the activation is above a given threshold and 0 otherwise.

Given the sparse matrices defining the weights between the reservoir and the output layer, W_{res}^{out} , and given the vectors $O = \{o_1, o_2, \dots, o_N\}$ containing the output activations

$$O(t+1) = f\left(k_{out} W_{res}^{out} X(t+1)\right) \quad (2)$$

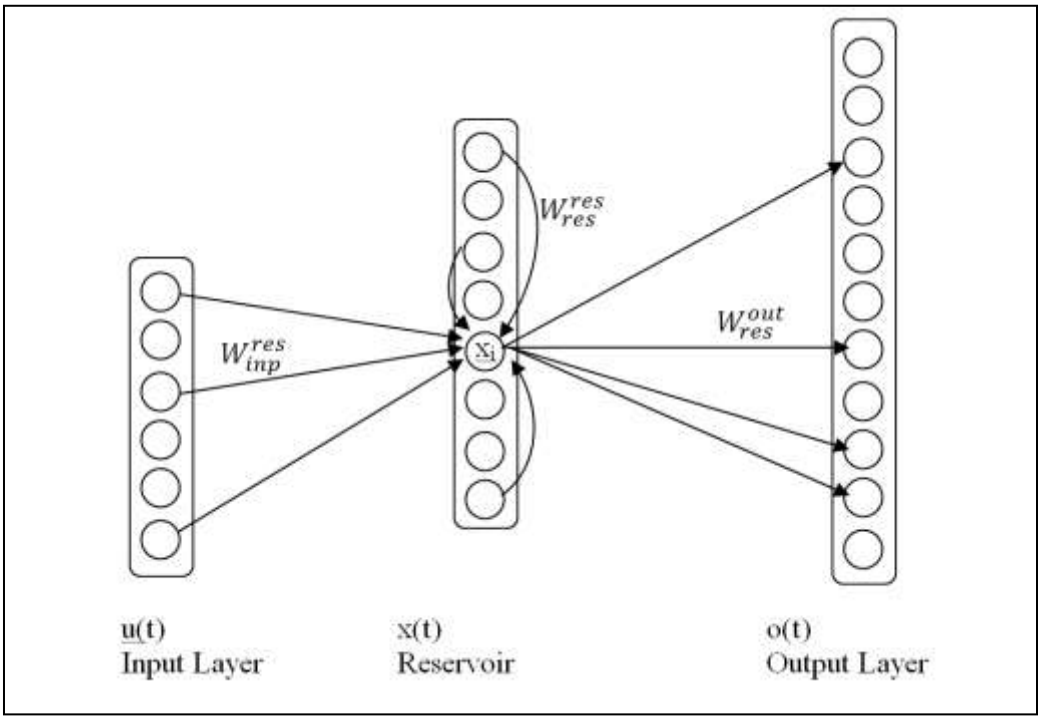

Figure 3: Reservoir Model

Training in the reservoir model is also modified from the traditional back propagation model. Training is only allowed to modify the weights in W_{res}^{out} which define the weights between the reservoir and the output layer. The weights in W_{inp}^{res} and W_{res}^{res} are set initially to random values and not changed during the simulation and training. This simplifies the training, since it is limited to a single layer.

Reservoir computing addresses a number of challenges faced by the traditional neural network model. First, reservoir models scale better, since each node is not connected to all the nodes in the other layers, but only a random set of nodes. Second, reservoir models can process temporal patterns directly, avoiding the need to map temporal patterns onto large input patterns.

3 Reservoir Computing and Undergraduate Research

While some might question working with undergraduate students, reservoir computing offers a number of advantages for undergraduate research. First, since it is a relatively new field, there are many open questions about the reservoir model that remain unanswered. Over the last 25 years, the traditional three layer, feed-forward model trained with back propagation has been studied extensively, leaving only the most esoteric question unanswered. While the original reservoir models were proposed back in 2001, significant research work has only been done over the last five years, leaving many open questions. One area of open questions concerns how reservoir computing models can be applied to different applications, from robot navigation to speech recognition. The following section of the paper will review current work in these applied areas and suggest open research questions.

There are a variety of open questions concerning the dynamics of reservoir computing that can also be addressed by undergraduate research. Researchers have focused on the echo state property of reservoirs as one way of understanding their dynamics. When the values of W_{res}^{res} are low enough that the reservoir state falls to the zero state without any external input, the reservoir is said to have the echo state property. Just like the sound waves caused by a large noise quiet down to nothing as the sound echoes down a stone well, the reservoir activations cause by a strong input signal will quiet down to nothing as the reservoir is updated repeatedly—if the recurrent waits are small enough. If the recurrent weights are large, the input is amplified like feedback from a microphone and amplifier.

Work by (Herbert Jaeger, 2002a) and (Herbert Jaeger, 2002b) has shown that the echo state property requires that the spectral radius, the largest eigenvalue of W_{res}^{res}, should be less than one. Limits to $w_{\max}$, the largest singular value in W_{res}^{res}, can also guarantee the echo state property (Herbert Jaeger, Lukosevicius, Popovici, & Siewert, 2007). While these upper bounds exist, many researchers struggle with finding good values for W_{res}^{res} that result in rich reservoir dynamics, leaving a number of open questions:

- How sparse should W_{res}^{res} be? What percent of nodes should each node in the reservoir be connected to? How does this scarcity affect the reservoir dynamics?

- The reservoir weights are generally divided into two groups, positive excitatory weights and negative inhibitory weights. How does the balance between positive and negative weights affect reservoir performance and the echo state property? What probability distribution should be used to generate the random weights in W_{res}^{res}?
- The nodes in the reservoir can be placed in a two-dimensional or three-dimensional geometry. The Liquid State Machines model places nodes in a three dimensional matrix, but other models use a two-dimensional geometry or no geometry at all. When a geometry is used, the probability of two nodes being connected can be based on their distance from each other. Nodes might inhibit the nodes near them or inhibit nodes further away. How the geometry affects the reservoir dynamics is an open question.
- Much of the research on reservoir dynamics has been tied to the spectral radius or the largest eigenvalue of W_{res}^{res}. While any undergraduate who has completed Linear Algebra knows how to calculate eigenvalues for a small matrix, this gets more challenging for a matrix with 10,000 elements. Researchers are looking for guidelines for how the random values for the weights can used to predict the largest eigenvalue.

Answering the above questions requires measuring when one reservoir is a higher quality than another; unfortunately the best way to measure reservoir quality is also an open question. The standard method for determining reservoir quality requires that the output layer be trained on a categorization task based on the reservoir. A reservoir should create a multi-dimensional representation of the input, which facilitates training. Reservoir quality is then measured directly from the ability of a system to learn the given supervised learning task. This method requires the time and computational intensive task of repeatedly training the output layer before the quality of the reservoir can be measured. Researchers are looking for ways to measure the quality of the reservoir more directly, avoiding the long training process.

One method of determining reservoir or kernel quality is separation, defined by (Goodman & Ventura, 2006) and expanded by (Norton, 2008). Separation is a measure of the average distance between the reservoir states resulting from different classes of input vectors. It is based on the assumption that input vectors and their corresponding reservoir states are divided into discrete classes. Since all reservoir states in the same class have the same target output, the distance between reservoir states in the same class are assumed small. Correspondingly, the distance between reservoir states in different classes should be large to facilitate the correct classification by the readout layer. Separation is defined as the ratio of the C_d, the average distance between the center of mass of each category, to C_v, the average distance from each vector to the center of mass. A high separation means the reservoir creates a high dimensional representation of the input where each category of inputs is distinct. Thus, the higher the separation, the higher the quality of the reservoir.

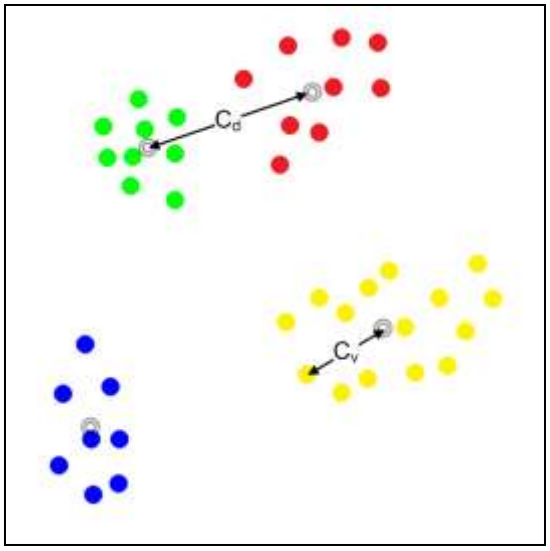

Figure 4: Distance between center of mass of two classes

This concept of separation has recently been expanded using a visualization technique called Separation Ratio Graphs (Gibbons, 2010). In a separation graph, the distance between pairs of input vectors and their corresponding reservoir states are shown in a scatter plot. The linear trend line is determined and a high quality reservoir has a trend line whose slope is near one and whose y-intercept is near zero. Figure 5 shows a sample Separation Ratio Graph.

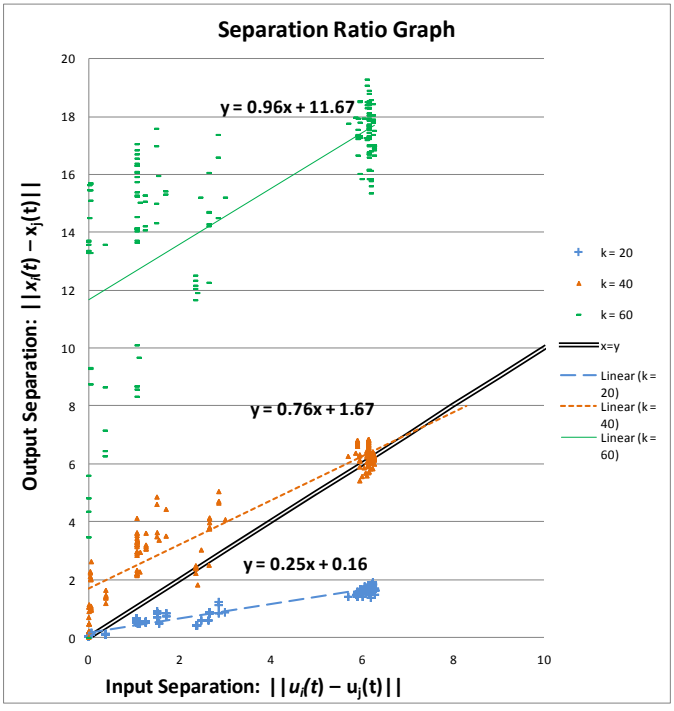

Figure 5: Sample Separation Ratio Graph showing reservoirs generated with three different parameter sets.

There are still many open questions concerning the measurement of reservoir quality. These include:

- Since each reservoir is created with a large number of random values, including which nodes are connected and the weight of each connection, how much do these random values affect reservoir quality? When measuring the quality of a reservoir generated with a set of parameters, how many reservoirs need to be generated to get an accurate measurement of quality?
- How are the different quality measures related? There have been basic comparisons of the different measures, but do these hold on a wide variety of reservoirs? The author has been working with an undergraduate student on this particular topic.
- What is the best distance measure to use in the quality measurements? What are the important characteristics of this distance measure and do any of the standard metrics satisfy these characteristics?

4 Application of Reservoir Computing

A goal of reservoir computing, like most artificial intelligence research, is to develop techniques that can solve real world problems. Since reservoir systems can process temporal data directly, many of the applications studied so far involve temporal data. This section provides an overview of some of the applications studied.

Biological:

Since neural networks are modeled after biological systems, one of the obvious applications is to see how they mimic biological systems. Initial research has been done modeling the cerebellum (Yamazaki & Tanaka, 2007) and the hippocampus (Eric Aislan Antonelo, Benjamin, & Dirk, 2009) using reservoirs. On a more primitive level, researchers have also tried to model the E. Coli bacteria as a Liquid State Machine (Jones, Stekel, Rowe, & Fernando, 2007). While these initial studies have been done, there is much more research that can be done using reservoir systems to model biological systems.

Robotics:

The temporal nature of reservoir models also makes them an obvious choice for robotic systems which process sensor data over time. Eric Antonelo has been leading a team using reservoir computing to build internal maps and provide two-dimensional robot navigation. (Eric Aislan Antonelo, et al., 2009), (E. A. Antonelo, Schrauwen, & Stroobandt, 2008) and (Eric Aislan Antonelo, Benjamin, & Dirk, 2008). Work has also been done using reservoir computing to model the classic double pole test (Jiang, Berry, & Schoenauer, 2008) and control a simulated robot arm (Joshi & Maass, 2005).

Speech Recognition:

Speech recognition is an obvious application of reservoir systems. A good place to start in this area is the comparison of an Echo State Network with a Simple Recurrent Network (Tong, Bickett, Christiansen, & Cottrell, 2007) where they look at a grammar problem first defined by Elman (Elman, 1990). Echo State Networks have also been used in a system that out-performs one of today's best models, the Hidden Markov Model (Skowronski & Harris, 2007).

Researchers have also looked at how reservoir systems can process isolated spoken digits (Verstraeten, et al., 2007) and also learn grammatical structure (Matthew, Adam, Eric, & Garrison, 2007). Given the breadth of this research, one might think that there is little left for undergraduate work, but the opposite is really true. While there have been one or two papers in each area of speech recognition, there are still many open questions. Also, the limited research makes it possible for undergraduate students to get up to speed on the current state of research. While there are hundreds of papers on the use of Hidden Markov Models is speech recognition, there are only a dozen papers on reservoir models applied to speech recognition, and only a few of these pertain to a specific research question. Thus an undergraduate can read all the papers related to the research in a week.

Music:

The ability of reservoirs to process temporal patterns has also allowed them to be applied to a number of applications related to music. Reservoirs have been trained to hum a melody and re-create a cyclical temporal pattern (H. Jaeger & Eck, 2008) and to repeat patterns from a drum machine (Tidemann & Demiris, 2008). There has also been undergraduate research on the use of Echo State Networks to model musical improvisation (Grychtol, 2006).

Other Applications:

There are many other areas where there has been limited or no study yet of the application of reservoir models. Possible applications include modeling:

- Short term memory, dyslexia, or other areas from psychology and cognitive science
- Speech generation, animal calls, or other auditory signal generation tasks
- Image recognition, especially tasks related to video images, such as motion detection or object recognition.

5 Simulation Tools

While reservoir computing is a relatively new field of research, there have been several open source tools developed for running simulations of different reservoir models. Given the mathematical nature of reservoir models and how equations 1 and 2 describe simple vector and matrix operations, it is not surprising that many of these simulation tools use Matlab, a common numerical computing environment that many undergraduates are

familiar with, for their base. Open source libraries in C++ have also been developed and released for all to use. A good listing of current simulation software is maintained by the Reservoir Computing website at www.reservoir-computing.org/software.

The author has also developed a simulator in VB.NET which he is willing to share with interested researchers, though it is a simple research tool without all the bells and whistles of a full application.

6 Summary

Some undergraduate students and their advisors find newly developed research areas intimidating, because they are not covered in the traditional AI course or textbook, but these recently developed models can be a rich area for undergraduate research. Reservoir computing is a model that most undergraduate students can understand and visualize. An understanding of linear algebra is sufficient mathematical background to understand most of the research in this area. The limited number of papers written so far allows students and advisors to quickly understand the state of the research and identify open questions. Open source simulation software allows students to run actual simulations using the same tools other researchers use. To demonstrate this, the author was able to work with an undergraduate student in a single semester to identify and research an open question related to reservoir computing. A good place to start research in this area is www.reservoir-computing.org, which maintains a current list of publications in reservoir computing and software simulators available.

References

Antonelo, E. A., Benjamin, S., & Dirk, S. (2008). *Mobile Robot Control in the Road Sign Problem using Reservoir Computing Networks*. Paper presented at the IEEE Int. Conf. on Robotics and Automation (ICRA).

Antonelo, E. A., Benjamin, S., & Dirk, S. (2009). *Unsupervised Learning in Reservoir Computing: Modeling Hippocampal Place Cells for Small Mobile Robots*. Paper presented at the International Conference on Artificial Neural Networks (ICANN).

Antonelo, E. A., Schrauwen, B., & Stroobandt, D. (2008). Event detection and localization for small mobile robots using reservoir computing. *Neural Networks, 21*(6), 862-871.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*(2), 179-211.

Gibbons, T. (2010). *Unifying Quality Metrics for Reservoir Networks*. Paper presented at the Proceedings of the International Joint Conference on Neural Networks (IJCNN).

Goodman, E., & Ventura, D. (2006). Spatiotemporal pattern recognition via liquid state machines. *Proceedings of the International Joint Conference on Neural Networks*, 3848–3853.

Grychtol, B. (2006). *Using Echo State Networks for Modeling Musical Improvisation.* Jacobs University.

Jaeger, H. (2001). *The ``echo state'' approach to analysing and training recurrent neural networks*: German National Research Center for Information Technology.

Jaeger, H. (2002a, 2003). *Adaptive nonlinear system identification with echo state networks.* Paper presented at the In: Advances in Neural Information Processing Systems.

Jaeger, H. (2002b). *Tutorial on training recurrent neural networks, covering {BPTT}, {RTRL}, {EKF} and the ``echo state network'' approach*: German National Research Center for Information Technology.

Jaeger, H., & Eck, D. (2008). Can't Get You Out of My Head: A Connectionist Model of Cyclic Rehearsal *Modeling Communication with Robots and Virtual Humans* (Vol. 4930/2008, pp. 310-335): Springer Berlin / Heidelberg.

Jaeger, H., Lukosevicius, M., Popovici, D., & Siewert, U. (2007). Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks, 20*(3), 335-352.

Jiang, F., Berry, H., & Schoenauer, M. (2008). *Unsupervised learning of echo state networks: balancing the double pole*. Paper presented at the Proceedings of the 10th annual conference on Genetic and evolutionary computation.

Jones, B., Stekel, D., Rowe, J., & Fernando, C. (2007). Is there a Liquid State Machine in the Bacterium Escherichia Coli? *IEEE Symposium on Artificial Life, 2007 (ALIFE '07)*, 187-191.

Joshi, P., & Maass, W. (2005). Movement generation with circuits of spiking neurons. *Neural Computation, 17*, 1715-1738.

Lukosevicius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review, 3*(3), 127-149.

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*, 2531â€“2560.

Matthew, H. T., Adam, D. B., Eric, M. C., & Garrison, W. C. (2007). Learning grammatical structure with Echo State Networks. *Neural Networks, 20*, 424â€“432.

Norton, R. D. (2008). Improving Liquid State Machines Through Iterative Refinement of the Reservoir. [Thesis].

Skowronski, M. D., & Harris, J. G. (2007). Automatic speech recognition using a predictive echo state network classifier. *Neural Networks, 20*(3), 414-423.

Tidemann, A., & Demiris, Y. (2008). *Groovy Neural Networks*. Paper presented at the Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence.

Tong, M. H., Bickett, A. D., Christiansen, E. M., & Cottrell, G. W. (2007). Learning grammatical structure with Echo State Networks. *Neural Networks, 20*(3), 424-432.

Verstraeten, D., Schrauwen, B., D'Haene, M., & Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks, 20*(3), 391-403.

Yamazaki, T., & Tanaka, S. (2007). The cerebellum as a liquid state machine. *Neural Networks, 20*(3), 290-297.