A Comparison of Recurrent Neural Network Architectures

Sean R. Aubin

October 26, 2016

1 Introduction

Artificial neural networks (ANNs) are distributed representations modelled after neurons. They have seen great success in various classification tasks. Recurrent Neural Networks (RNNs) are any ANN that connects back on itself, as opposed to just being feed-forward. The most common paradigm for training ANNs for classification is Deep Learning (DL), wherein the error between the network output and the desired is propagated amongst the layers of neurons using a method called Backpropogation [13]. This error is then used to adjust the weights of the respective neurons. The simplest DL RNN is the Vanilla Recurrent Neural Network (vRNN).

Spiking Neural Networks (SNNs) are ANNs that use spikes for inter-neuron communication and are trained with methods other than back-propagation. The reliance on spikes for communication make these neurons more biologically plausible and motivate their use in cognitive models [6]. In terms of computation, SNNs also have a number of advantages over vRNNs. Their spiking communications allows for the creation low-power neuromorphic hardware which responds with lower latency than typical digital hardware [11] and they are able to learn while processing data [2].

This report examines whether the spiking nature of SNNs also helps with performance given a limited data-set and robustness to noise, when compared to vRNNs.

2 Methods

2.1 The Neural Engineering Framework

The Neural Engineering Framework [5] may be understood as a "neural compiler" for mathematical functions using vector spaces. The classification networks presented in this report rely on two key principles of the NEF as a method for constructing networks using spiking neurons and connection weights. The first principle describes how a vector can be mapped onto a distributed representation over neurons. The second principle characterizes how connections between neurons can transform these vectors.

A vector $\vec{x}(t)$ is represented by encoding it into the spiking activity of a population of neurons. Each neuron i has an encoding vector $\vec{e_i}$ (which can

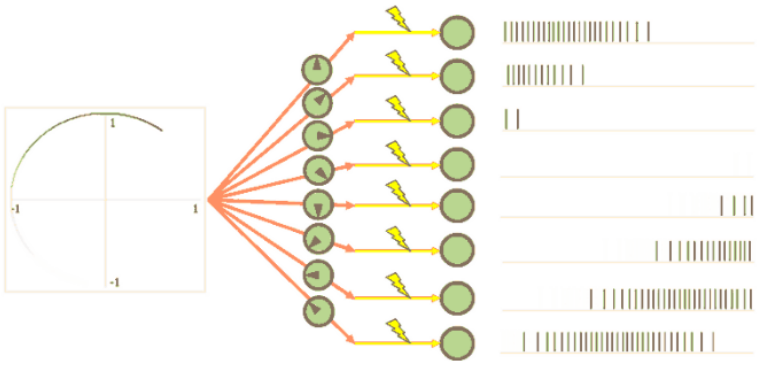

Figure 1: Illustration showing how neurons transform a 2-dimensional signal into spikes depending on their preferred direction. The signal is the mark moving in a circular fashion and the preferred direction is indicated by the compass-like circles overlaid on the connections. An animated version of this figure can be found at `https://goo.gl/m59Pbf`.

be understood as a preferred direction in the vector space), a gain α_i, and a background current J_i^{bias}. These parameters determine how the input vector is translated into the input current $J_i(t)$ to a neural nonlinearity $G_i[\cdot]$. In this report, two neural neural nonlinearities are compared in Section 2.1.1, leaky integrate-and-fire (LIF) and the adaptive LIF (ALIF) model. Both convert the input current into a neural spike train $a_i(t)$. This process from input to spike train is described in Equation 1 and shown in Figure 1.

$$a_i(t) = G_i\left[J_i(t)\right], \quad J_i(t) = \alpha_i \vec{e}_i \cdot \vec{x}(t) + J_i^{bias} \tag{1}$$

To decode an approximation of the vector back from spike trains, spike trains are first convolved with a low-pass filter $h(t)$ (a decaying exponential modelled after the postsynaptic current) and then multiplied by a decoding vector $\vec{d}_i$:

$$\hat{\vec{x}}(t) = \sum_i \vec{d}_i \, (a_i * h)(t) \tag{2}$$

The decoders $\vec{d}_i$ are found using regularized least squares optimization to minimize the error over the range of inputs $\vec{x}$:

$$\int (\vec{x} - \hat{\vec{x}})^2 \, d\vec{x}. \tag{3}$$

To describe how two neural ensembles are connected, we define a weight matrix as the outer product of the encoders and decoders $\omega_{ij} = \vec{e}_i \bigotimes \vec{d}_j$. The second principle then shows that the input current to neuron i may be rewritten as a weighted summation of its postsynaptic potentials, allowing for the connection of multiple populations of neurons:

$$J_i(t) = \sum_j \alpha_i \, \omega_{ij} \, (a_j * h)(t) + J_i^{bias}. \tag{4}$$

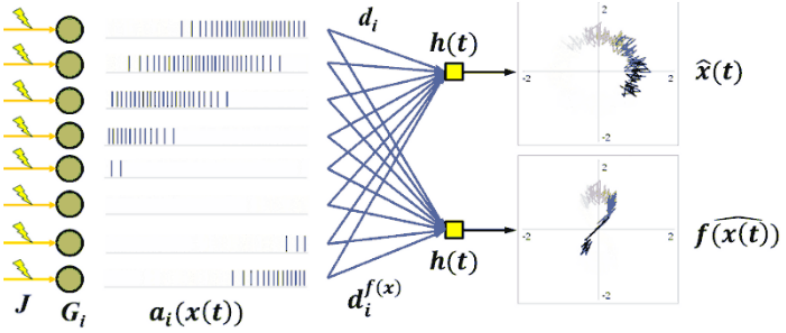

Figure 2: Illustration of decoding process where the spikes from the encoding figure are now transformed to a 2D output signal via decoding weights. An animated version of this figure can be found at `https://goo.gl/m59Pbf`.

To apply arbitrary transformations to the vector using these connections, we can minimize the decoding error for the desired function:

$$\int (f(\vec{x}) - \hat{f}(\vec{x}))^2 \, d\vec{x}. \tag{5}$$

This process from spike trains to desired function approximation is shown in Figure 2. In the case of classification, which this report is focused on, the function approximated is a mapping is from the input signal to the desired output class, wherein the output class is a one-hot encoding.

2.1.1 Neuron Types

The NEF is functions with any sort of spiking neuron model. In this report, LIF (Equation 6) and ALIF (Equation 7) neurons are compared. The LIF neuron and ALIF neuron voltage update equations are nearly identical. The voltage equation dictates how much the membrane voltage $V(t)$ rises in response to an input $J(t)$. When $V(t) > 1$ the neuron spikes, which resets $V(t) = 0$ and makes the neuron unresponsive for the refractory period τ_{ref}. In addition to τ_{ref}, the response of neuron is dictated by the constant τ_{RC}, which is named as such due to the underlying model of this neuron being an RC-circuit.

$$V(t) = \frac{1}{\tau_{ref} - \tau_{RC} \ln(1 - \frac{1}{J(t)})} \tag{6}$$

The difference of between the ALIF and LIF neurons is the response to stimuli over time. The ALIF has an n term which reduces the spiking response to a stimuli over time in proportion to the constants τ_n and inc_n, thus the name "Adaptive".

$$V(t) = \frac{1}{\tau_{ref} - \tau_{RC} \ln(1 - \frac{1}{J(t) - n})}, \quad n = n + \frac{\text{inc}_n \dot{V}(t) - n}{\tau_n} \tag{7}$$

In this report, the two neuron models were compared in the hopes of identifying whether ALIF neurons would be better at classifying temporal signals given their greater temporal sensitivity.

2.2 Spiking Classification Networks

Two architectures of SNNs for classification are compared, a Reservoir Computing (RC) based and an Frequency Component Support Vector Machine (FC-SVM) design.

2.2.1 Reservoir Computing

Although RC is an approach usually considered distinct from the NEF, it will be described in this section as an extension. In RC, a reservoir is constructed by connecting a neuron to all neurons in a group using random connection weights. This generates a complex system of dynamics out of which, in a manner identical to the NEF method of finding decoders described in 2.1. So that the dynamic system remains stable, the magnitude of the weights are normalized by the largest eigenvalue. It should be noted, that in the context of reservoir computing, networks that use spiking networks are called Liquid State Machines [14], while those that use rate-based neurons are called Echo State Machines [10], however only spiking reservoirs will be used in this case and will be referred to as Reservoir Computing to avoid confusion.

RC has been previously used for classification. The use cases were static pattern recognition [7], applications using hard-coded non-neural feature extraction [1] or instances where hierarchical stacking of reservoirs were required for them to be competitive [15], but eventually surpassed by vRNNs [12]. This report investigates a novel experiment wherein only a single reservoir of spiking neurons is applied to dynamic time-dependent signal use case.

2.2.2 Frequency Component Support Vector Machine

The Frequency Component Support Vector Machine (FC-SVM) SNN design is adapted from [8] where it is used to classify textures. FC-SVM has two stages, the frequency feature extraction stage and the SVM classification stage. The frequency feature extraction is accomplished by adding a 2^{nd} order band-pass filter to the spikes of each neuron as a synapse. This allows for features 0-500Hz to be extracted from the input signal. The features extracted from the filtered spiking activity are then used to train an SVM. The SVM is implemented in neuronal basis functions using the NEF. A SVM is a high-dimensional vector operation where coefficients vectors applied to selected features and modified by biases are used to classify a data point. This use of vectors operations (as opposed to hard logic) allows SVM to be phrased in terms of the NEF, as shown in Figure 3. Specifically, the decoder vector from the feature population to the score output is the coefficients of the SVM. An added bias connection to the score output provides the final classification output.

2.3 Deep Learning with "Vanilla" RNNs

As mentioned previously, all of Deep Learning works by propagating the output error of the network to each component/layer and then performing gradient

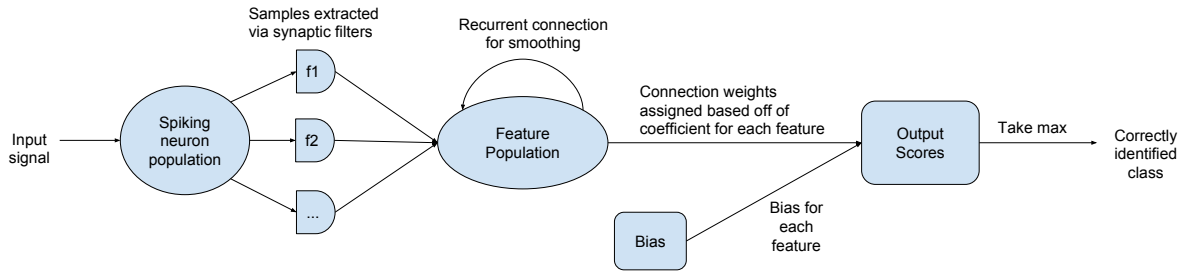

Figure 3: High level view of FC-SVM system.

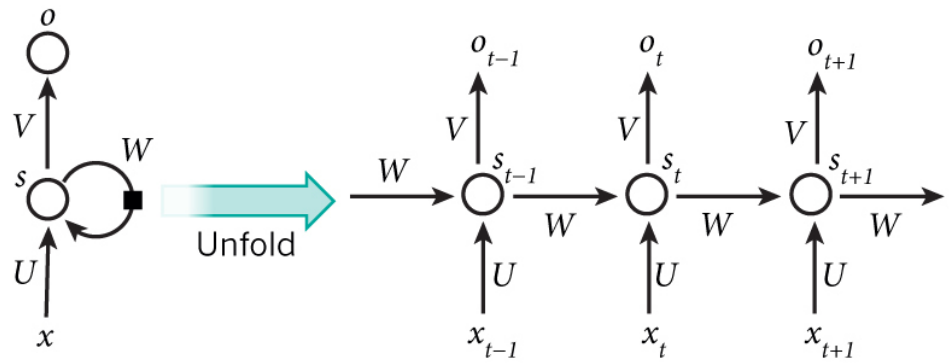

Figure 4: An example of back-propagation through time applied to a vanilla Deep Learning Recurrent Neural Network taken from [13].

descent on each component/layer to reduce the error. Due to the architecture of the "Vanilla" RNNs (vRNNs) the layers need to be unwrapped and the error needs to be back-propagated through each time-step of the network for a single input signal. This is typically called back-propagation through time and is shown in Figure 4. Given that the topic of back-propagation was already covered in this course, this section will not delve into further detail about the method.

2.4 Synthetic Datasets

A continuous dataset with an adjustable length, dimensionality and different features to fully characterize the RNNs was impossible to acquire, so instead a synthetic dataset was created using the following methods signal generation methods, as displayed in Figure 5:

1. Continuous White Noise
2. Discretized White Noise
3. Continuous Signals Generated from interpolation between values of Orthogonal Vectors

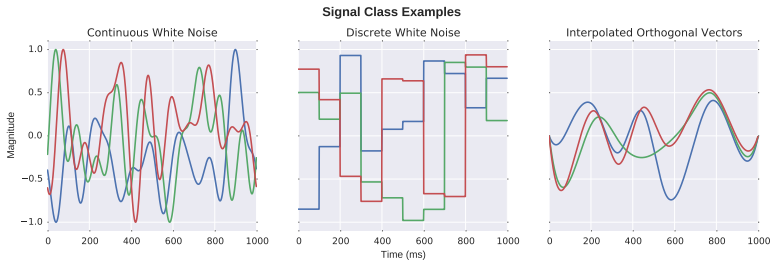

Figure 5: Example of three different classes of single-dimensional signals of 1s length and a frequency of 10Hz. For the Orthogonal Vectors, the frequency is interpreted as the length of the vector who's values will be interpolated to generate the signal.

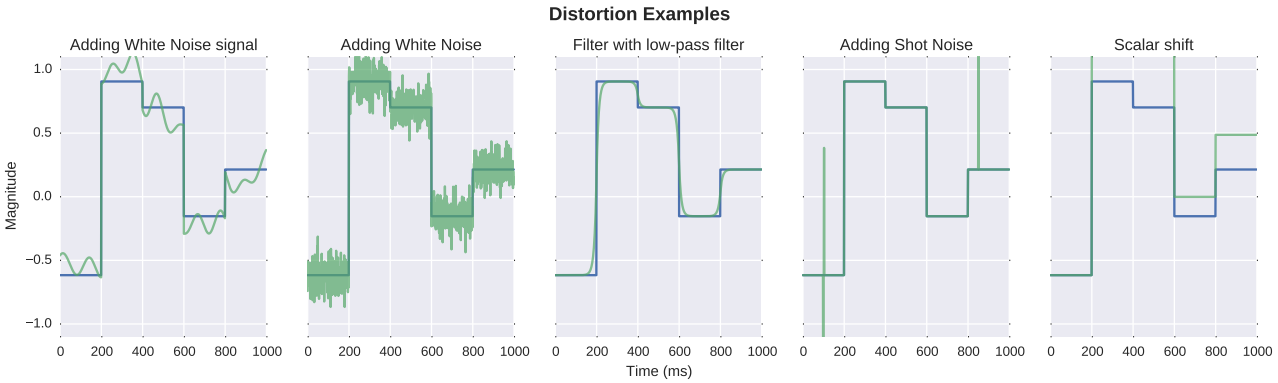

Figure 6: Examples of corruption classes applied to a discrete signal.

To corrupt the dataset, the following noise augmentations were used, as displayed in Figure 6:

1. Additive White Noise
2. Adding a White Noise signal
3. Filtering with a low-pass filter
4. Impulse like "shot" noise
5. Shifting the whole signal by a scalar value

3 Results

All simulations were run using Nengo [3] with Lasagne [4] being used for the DL components. All code is available available at `https://github.com/Seanny123/rnn-comparison`. Classification output was given as a one-hot encoded vector. The final classification decision used to measure the accuracy was the output dimension with the maximum confidence over the period of time the signal

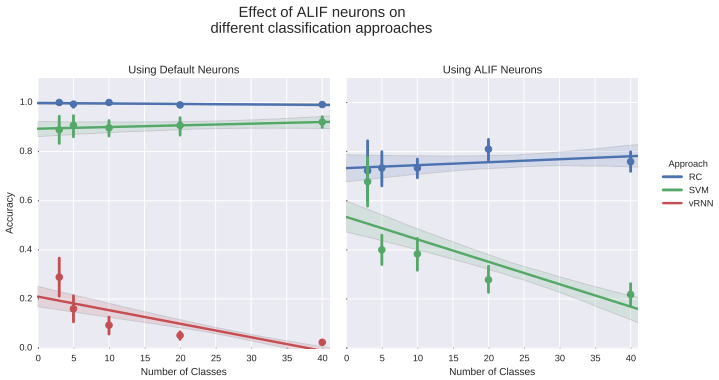

Figure 7: Effect ALIF neurons on accuracy depending on number of classes.

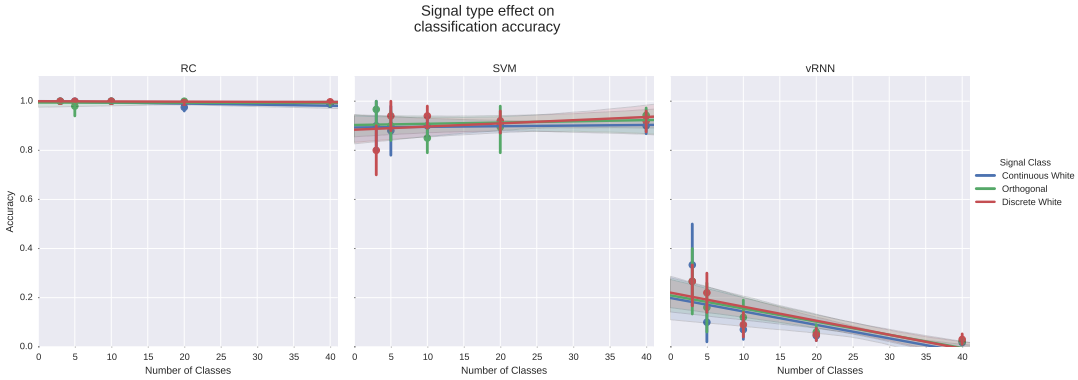

Figure 8: Effect ALIF neurons on accuracy depending on number of classes.

was presented. Note that all error-bars in the result plots are boot-strapped confidence intervals and were plotted using Seaborn [16].

3.1 Effect of ALIF neurons and number of classes

All approaches decreased in accuracy with the increase of a number of classes. ALIFs offered no improvement over SNNs when it comes to classification. The success of RC over FC-SVM is most likely due to the over-specialization of FC-SVM to frequency data, whereas RC could optimize for any given feature during the regression step. There were no noticeable differences between different signal classes and the classification performances of the different approaches, as shown in Figure 8. This is probably because the signals did not vary much in information density and the features were equally easily encoded in spiking neurons.

The inability for ALIFs to improve the classification error is unsurprising given previous work in this area, which was found after a subsequent literature search, showing that exploiting complex neuron models is an open systems identification problem [9]. Essentially, the regression method described in Section 2.1 implicitly assumes a linear neuron model, which the ALIF is specifically

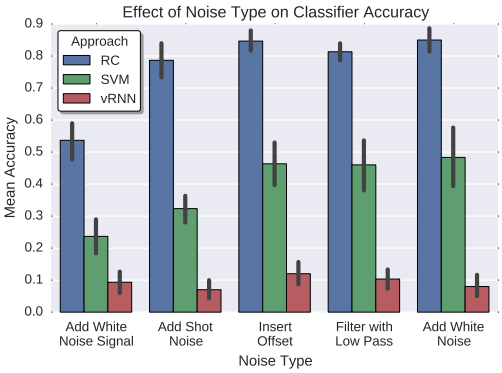

Figure 9: Effect of being trained and tested with signals corrupted by different noise classes on accuracy of classifying 10 different signals.

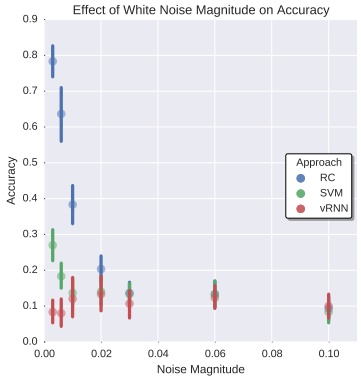

Figure 10: Effect of being trained and tested with signals corrupted by different White Noise magnitudes.

not.

3.2 Effect of noise magnitude and type

As seen in Figure 10, adding a White Noise signal caused the largest global change in classification accuracy. The FC-SVM responded poorly to any sort of noise being added due it's inability to distinguish from useful frequency features and noise.

As shown in figure 10, increase of magnitude of noise globally reduced accuracy across all classification approaches. An interesting exception to this trend is vRNNs which got increased accuracy with low-level noise, probably reducing their tendency to get caught in local minima when doing gradient descent.

3.3 Compensating for vRNN deficiencies

DL is dependent on large data-sets to perform well. Consequently, an attempt was made to augment the data-set provided, by increasing the number of exam-

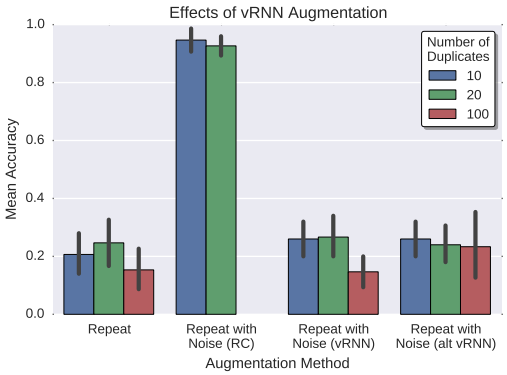

Figure 11: Although there may appear fluctuations in accuracy given the number of duplicates, all confidence intervals are overlapping and thus the fluctuations should be dismissed.

ples. This was accomplished by duplicating the signals to classify and corrupting them with White Noise with a magnitude of 0.01. However, as shown in Figure 11, the effect of this augmentation was minimal. In addition to augmenting the dataset, an alternative architecture (denoted as "alt vRNN" in the Figure) using rectified linear nonlinearities and initializing the recurrent weights as an identity matrix [12], also failed to improve results.

The abysmal performance of the vRNNs is not conclusive since not all options for hyper-parameters (learning rates, number of hidden units... etc.), training regimes, objective functions and optimizers were explored. However, it is still indicative of the challenge of training DL models when compared to SNNs.

4 Discussion

This report demonstrated that, in addition to being biologically plausible and computationally efficient, when given short signals and a limited training set, SNNs can outperform DL RNNs at classification. Future work in this domain, should focus on more varied signals and benchmarks, other forms of signal corruption, utilizing external memories, online learning and learning hierarchical structure in neurons.

In terms of benchmarks, more real-world benchmarks should be examined, such as the Japanese Phoneme database previously used by RC [15]. If such benchmarks were too difficult to obtain or were not helpful in identifying limitations of SNNs, one could still generate artificial signals with more sparse information and longer-term dependencies to reveal the limitations of SNNs. Alternatively signal corruption, such as random signal lag or missing time-steps could be implemented to artificially increase the sparsity of information in the signal.

Once these more challenging benchmarks were established, further augmentations to SNNs for classification could be explored. One of the significant advantages of vRNN and LSTM are the ability to maintain long term memories. It is possible to create memory-like storage in spiking neurons, however it

is unclear how the SNN should learn to exploit them.

Finally, this experiment only considered both vRNN and RC as single modules and performed no online learning. However, one of the advantages to both of these architectures is the ability to stack them hierarchically and a significant advantage of SNNs is their ability to learn online [2]. Hierarchical online learning isn't possible NEF networks, because back-propagation isn't biologically plausible, as it requires passing information backwards through unidirectional synapses. Future research should focus on finding a way to do error minimization across multiple layers SNNs.

References

[1] Abdulrahman Alalshekmubarak and Leslie S Smith. On improving the classification capability of reservoir computing for arabic speech recognition. In *International Conference on Artificial Neural Networks*, pages 225–232. Springer, 2014.

[2] Trevor Bekolay. Learning in large-scale spiking neural networks. Master's thesis, University of Waterloo, Waterloo, ON, 09/2011 2011.

[3] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7, 2013.

[4] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, Diogo Moitinho de Almeida, Brian McFee, Hendrik Weideman, Gábor Takács, Peter de Rivaz, Jon Crall, Gregory Sanders, Kashif Rasul, Cong Liu, Geoffrey French, and Jonas Degrave. Lasagne: First release., August 2015.

[5] Chris Eliasmith and Charles H. Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, Cambridge, MA, 2003.

[6] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

[7] Mark J Embrechts, Luís A Alexandre, and Jonathan D Linton. Reservoir computing for static pattern recognition. In *ESANN*, 2009.

[8] Ken Elmar Friedl, Aaron Russell Voelker, Angelika Peer, and Chris Eliasmith. Human-inspired neurorobotic system for classifying surface textures by touch. *Robotics and Automation Letters*, 1(1):516–523, 01 2016.

[9] Eric Hunsberger. System identification of adapting neurons. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON, 09 2016.

[10] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German*

National Research Center for Information Technology GMD Technical Report, 148:34, 2001.

[11] James Knight, Aaron Russell Voelker, Andrew Mundy, Chris Eliasmith, and Steve Furber. Efficient spinnaker simulation of a heteroassociative memory using the neural engineering framework. In *The 2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 07 2016.

[12] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[14] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[15] Fabian Triefenbach, Azarakhsh Jalalvand, Benjamin Schrauwen, and Jean-Pierre Martens. Phoneme recognition with large hierarchical reservoirs. In *Advances in neural information processing systems*, pages 2307–2315, 2010.

[16] Michael Waskom, Olga Botvinnik, drewokane, Paul Hobson, Yaroslav Halchenko, Saulius Lukauskas, Jordi Warmenhoven, John B. Cole, Stephan Hoyer, Jake Vanderplas, gkunter, Santi Villalba, Eric Quintero, Marcel Martin, Alistair Miles, Kyle Meyer, Tom Augspurger, Tal Yarkoni, Pete Bachant, Constantine Evans, Clark Fitzgerald, Tamas Nagy, Erik Ziegler, Tobias Megies, Daniel Wehner, Samuel St-Jean, Luis Pedro Coelho, Gregory Hitz, Antony Lee, and Luc Rocher. seaborn: v0.7.0 (january 2016), January 2016.