

On the Practical Computational Power of Finite Precision RNNs for Language Recognition

Gail Weiss
Technion, Israel

Yoav Goldberg
Bar-Ilan University, Israel

Eran Yahav
Technion, Israel

{sgailw, yahave}@cs.technion.ac.il
yogo@cs.biu.ac.il

Abstract

While Recurrent Neural Networks (RNNs) are famously known to be Turing complete, this relies on infinite precision in the states and unbounded computation time. We consider the case of RNNs with finite precision whose computation time is linear in the input length. Under these limitations, we show that different RNN variants have different computational power. In particular, we show that the LSTM and the Elman-RNN with ReLU activation are strictly stronger than the RNN with a squashing activation and the GRU. This is achieved because LSTMs and ReLU-RNNs can easily implement counting behavior. We show empirically that the LSTM does indeed learn to effectively use the counting mechanism.

1 Introduction

Recurrent Neural Network (RNNs) emerge as very strong learners of sequential data. A famous result by Siegelmann and Sontag (1992; 1994), and its extension in (Siegelmann, 1999), demonstrates that an Elman-RNN (Elman, 1990) with a sigmoid activation function, rational weights and infinite precision states can simulate a Turing-machine in real-time, making RNNs Turing-complete. Recently, Chen et al (2017) extended the result to the ReLU activation function. However, these constructions (a) assume reading the entire input into the RNN state and only then performing the computation, using *unbounded time*; and (b) rely on having *infinite precision* in the network states. As argued by Chen et al (2017), this is not the model of RNN computation used in NLP applications. Instead, RNNs are often used by feeding an input sequence into the RNN one

item at a time, each immediately returning a state-vector that corresponds to a prefix of the sequence and which can be passed as input for a subsequent feed-forward prediction network operating in constant time. The amount of tape used by a Turing machine under this restriction is linear in the input length, reducing its power to recognition of context-sensitive language. More importantly, computation is often performed on GPUs with 32bit floating point computation, and there is increasing evidence that competitive performance can be achieved also for quantized networks with 4-bit weights or fixed-point arithmetics (Hubara et al., 2016). The construction of (Siegelmann, 1999) implements pushing 0 into a binary stack by the operation $g \leftarrow g/4 + 1/4$. This allows pushing roughly 15 zeros before reaching the limit of the 32bit floating point precision. Finally, RNN solutions that rely on carefully orchestrated mathematical constructions are unlikely to be found using backpropagation-based training.

In this work we restrict ourselves to *input-bound recurrent neural networks with finite-precision states (IBFP-RNN)*, trained using backpropagation. This class of networks is likely to coincide with the networks one can expect to obtain when training RNNs for NLP applications. An IBFP Elman-RNN is finite state. But what about other RNN variants? In particular, we consider the Elman RNN (SRNN) (Elman, 1990) with squashing and with ReLU activations, the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and the Gated Recurrent Unit (GRU) (Cho et al., 2014; Chung et al., 2014).

The common wisdom is that the LSTM and GRU introduce additional *gating components* that handle the vanishing gradients problem of training SRNNs, thus stabilizing training and making it more robust. The LSTM and GRU are often considered as almost equivalent variants of each other.

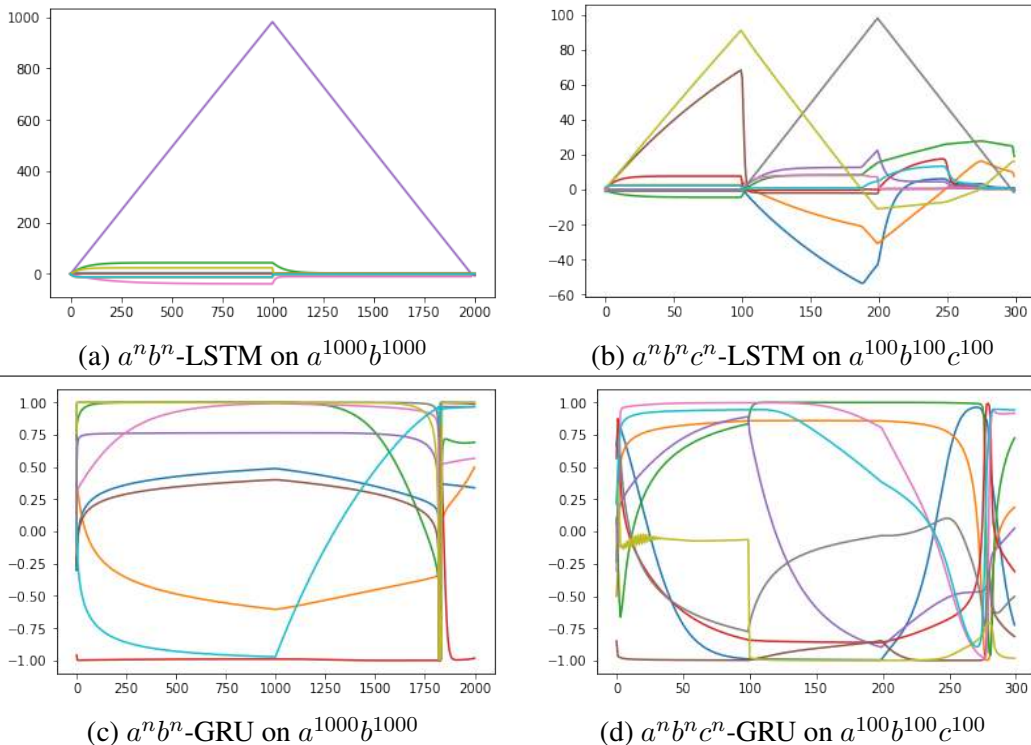


Figure 1: Activations — c for LSTM and h for GRU — for networks trained on $a^n b^n$ and $a^n b^n c^n$. The LSTM has clearly learned to use an explicit counting mechanism, in contrast with the GRU.

We show that in the input-bound, finite-precision case, there is a real difference between the computational capacities of the LSTM and the GRU: the LSTM can easily perform unbounded counting, while the GRU (and the SRNN) cannot. This makes the LSTM a variant of a k -counter machine (Fischer et al., 1968), while the GRU remains finite-state. Interestingly, the SRNN with ReLU activation followed by an MLP classifier also has power similar to a k -counter machine.

These results suggest there is a class of formal languages that can be recognized by LSTMs but not by GRUs. In section 5, we demonstrate that for at least two such languages, the LSTM manages to learn the desired concept classes using back-propagation, while using the hypothesized control structure. Figure 1 shows the activations of 10-d LSTM and GRU trained to recognize the languages $a^n b^n$ and $a^n b^n c^n$. It is clear that the LSTM learned to dedicate specific dimensions for counting, in contrast to the GRU.¹

¹Is the ability to perform unbounded counting relevant to “real world” NLP tasks? In some cases it might be. For example, processing linearized parse trees (Vinyals et al., 2015; Choe and Charniak, 2016; Aharoni and Goldberg, 2017) requires counting brackets and nesting levels. Indeed, previous works that process linearized parse trees report using LSTMs

2 The RNN Models

An RNN is a parameterized function R that takes as input an input vector x_t and a state vector h_{t-1} and returns a state vector h_t :

$$h_t = R(x_t, h_{t-1}) \quad (1)$$

The RNN is applied to a sequence x_1, \dots, x_n by starting with an initial vector h_0 (often the 0 vector) and applying R repeatedly according to equation (1). Let Σ be an input vocabulary (alphabet), and assume a mapping E from every vocabulary item to a vector x (achieved through a 1-hot encoding, an embedding layer, or some other means). Let $RNN(x_1, \dots, x_n)$ denote the state vector h resulting from the application of R to the sequence $E(x_1), \dots, E(x_n)$. An RNN recognizer (or RNN acceptor) has an additional function f mapping states h to 0, 1. Typically, f is a log-linear classifier or multi-layer perceptron. We say that an RNN recognizes a language $L \subseteq \Sigma^*$ if $f(RNN(w))$ returns 1 for all and only words $w = x_1, \dots, x_n \in L$.

Elman-RNN (SRNN) In the Elman-RNN (Elman, 1990), also called the Simple RNN (SRNN), and not GRUs for this purpose. Our work here suggests that this may not be a coincidence.

the function R takes the form of an affine transform followed by a tanh nonlinearity:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b) \quad (2)$$

Elman-RNNs are known to be at-least finite-state. Siegelmann (1996) proved that the tanh can be replaced by any other squashing function without sacrificing computational power.

IRNN The IRNN model, explored by (Le et al., 2015), replaces the tanh activation with a non-squashing ReLU:

$$h_t = \max(0, (Wx_t + Uh_{t-1} + b)) \quad (3)$$

The computational power of such RNNs (given infinite precision) is explored in (Chen et al., 2017).

Gated Recurrent Unit (GRU) In the GRU (Cho et al., 2014), the function R incorporates a *gating mechanism*, taking the form:

$$z_t = \sigma(W^z x_t + U^z h_{t-1} + b^z) \quad (4)$$

$$r_t = \sigma(W^r x_t + U^r h_{t-1} + b^r) \quad (5)$$

$$\tilde{h}_t = \tanh(W^h x_t + U^h (r_t \circ h_{t-1}) + b^h) \quad (6)$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t \quad (7)$$

Where σ is the sigmoid function and \circ is the Hadamard product (element-wise product).

Long Short Term Memory (LSTM) In the LSTM (Hochreiter and Schmidhuber, 1997), R uses a different gating component configuration:

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f) \quad (8)$$

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i) \quad (9)$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o) \quad (10)$$

$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1} + b^c) \quad (11)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \quad (12)$$

$$h_t = o_t \circ g(c_t) \quad (13)$$

where g can be either tanh or the identity.

Equivalences The GRU and LSTM are at least as strong as the SRNN: by setting the gates of the GRU to $z_t = 0$ and $r_t = 1$ we obtain the SRNN computation. Similarly by setting the LSTM gates to $i_t = 1, o_t = 1$, and $f_t = 0$. This is easily achieved by setting the matrices W and U to 0, and the biases b to the (constant) desired gate values.

Thus, all the above RNNs can recognize finite-state languages.

3 Power of Counting

Power beyond finite state can be obtained by introducing counters. Counting languages and k-counter machines are discussed in depth in (Fischer et al., 1968). When unbounded computation is allowed, a 2-counter machine has Turing power. However, for computation bound by input length (real-time) there is a more interesting hierarchy. In particular, real-time counting languages cut across the traditional Chomsky hierarchy: real-time k-counter machines can recognize at least one context-free language ($a^n b^n$), and at least one context-sensitive one ($a^n b^n c^n$). However, they *cannot* recognize the context free language given by the grammar $S \rightarrow x|aSa|bSb$ (palindromes).

SKCM For our purposes, we consider a simplified variant of k-counter machines (SKCM). A counter is a device which can be incremented by a fixed amount (INC), decremented by a fixed amount (DEC) or compared to 0 (COMP0). Informally,² an SKCM is a finite-state automaton extended with k counters, where at each step of the computation each counter can be incremented, decremented or ignored in an input-dependent way, and state-transitions and accept/reject decisions can inspect the counters' states using COMP0. The results for the three languages discussed above hold for the SKCM variant as well, with proofs provided in the supplementary material.

4 RNNs as SKCMs

In what follows, we consider the effect on the state-update equations on a single dimension, $h_t[j]$. We omit the index $[j]$ for readability.

LSTM The LSTM acts as an SKCM by designating k dimensions of the memory cell c_t as counters. In non-counting steps, set $i_t = 0, f_t = 1$ through equations (8-9). In counting steps, the counter direction (+1 or -1) is set in \tilde{c}_t (equation 11) based on the input x_t and state h_{t-1} . The counting itself is performed in equation (12), after setting $i_t = f_t = 1$. The counter can be reset to 0 by setting $i_t = f_t = 0$.

Finally, the counter values are exposed through $h_t = o_t g(c_t)$, making it trivial to compare the

²Formal definition is given in the supplementary material.

counter’s value to 0.³

We note that this implementation of the SKCM operations is achieved by saturating the activations to their boundaries, making it relatively easy to reach and maintain in practice.

SRNN The finite-precision SRNN cannot designate unbounded counting dimensions.

The SRNN update equation is:

$$h_t = \tanh(Wx + Uh_{t-1} + b)$$

$$h_t[i] = \tanh\left(\sum_{j=1}^{d_x} W_{ij}x[j] + \sum_{j=1}^{d_h} U_{ij}h_{t-1}[j] + b[i]\right)$$

By properly setting U and W, one can get certain dimensions of h to update according to the value of x , by $h_t[i] = \tanh(h_{t-1}[i] + w_ix + b[i])$. However, this counting behavior is within a tanh activation. Theoretically, this means unbounded counting cannot be achieved without infinite precision. Practically, this makes the counting behavior inherently unstable, and bounded to a relatively narrow region. While the network could adapt to set w to be small enough such that counting works for the needed range seen in training without overflowing the tanh, attempting to count to larger n will quickly leave this safe region and diverge.

IRNN Finite-precision IRNNs can perform unbounded counting conditioned on input symbols. This requires representing each counter as two dimensions, and implementing INC as incrementing one dimension, DEC as incrementing the other, and COMP0 as comparing their difference to 0. Indeed, Appendix A in (Chen et al., 2017) provides concrete IRNNs for recognizing the languages $a^n b^n$ and $a^n b^n c^n$. This makes IBFP-RNN with

³Some further remarks on the LSTM: LSTM supports both increment and decrement in a single dimension. The counting dimensions in c_t are exposed through a function g . For both $g(x) = x$ and $g(x) = \tanh(x)$, it is trivial to do compare 0. Another operation of interest is comparing two counters (for example, checking the difference between them). This cannot be reliably achieved with $g(x) = \tanh(x)$, due to the non-linearity and saturation properties of the tanh function, but is possible in the $g(x) = x$ case. LSTM can also easily set the value of a counter to 0 in one step. The ability to set the counter to 0 gives slightly more power for real-time recognition, as discussed by Fischer et al. (1968).

Relation to known architectural variants: Adding peephole connections (Gers and Schmidhuber, 2000) essentially sets $g(x) = x$ and allows comparing counters in a stable way. Coupling the input and the forget gates ($i_t = 1 - f_t$) (Greff et al., 2017) removes the single-dimension unbounded counting ability, as discussed for the GRU.

ReLU activation more powerful than IBFP-RNN with a squashing activation. Practically, ReLU-activated RNNs are known to be notoriously hard to train because of the exploding gradient problem.

GRU Finite-precision GRUs cannot implement unbounded counting on a given dimension. The tanh in equation (6) combined with the interpolation (tying z_t and $1 - z_t$) in equation (7) restricts the range of values in h to between -1 and 1, precluding unbounded counting with finite precision. Practically, the GRU can learn to count up to some bound m seen in training, but will not generalize well beyond that.⁴ Moreover, simulating forms of counting behavior in equation (7) require consistently setting the gates z_t , r_t and the proposal \tilde{h}_t to precise, non-saturated values, making it much harder to find and maintain stable solutions.

Summary We show that LSTM and IRNN can implement unbounded counting in dedicated counting dimensions, while the GRU and SRNN cannot. This makes the LSTM and IRNN at least as strong as SKCMs, and strictly stronger than the SRNN and the GRU.⁵

5 Experimental Results

Can the LSTM indeed learn to behave as a k-counter machine when trained using backpropagation? We show empirically that:

1. LSTMs can be trained to recognize $a^n b^n$ and $a^n b^n c^n$.
2. These LSTMs generalize to much higher n than seen in the training set (though not infinitely so).
3. The trained LSTM learn to use the per-dimension counting mechanism.
4. The GRU can also be trained to recognize $a^n b^n$ and $a^n b^n c^n$, but they do not have clear

⁴One such mechanism could be to divide a given dimension by $k > 1$ at each symbol encounter, by setting $z_t = 1/k$ and $\tilde{h}_t = 0$. Note that the inverse operation would not be implementable, and counting down would have to be realized with a second counter.

⁵One can argue that other counting mechanisms— involving several dimensions—are also possible. Intuitively, such mechanisms cannot be trained to perform unbounded counting based on a finite sample as the model has no means of generalizing the counting behavior to dimensions beyond those seen in training. We discuss this more in depth in the supplementary material, where we also prove that an SRNN cannot represent a binary counter.

counting dimensions, and they generalize to much smaller n than the LSTMs, often failing to generalize correctly even for n within their training domain.

5. Trained LSTM networks outperform trained GRU networks on random test sets for the languages $a^n b^n$ and $a^n b^n c^n$.

Similar empirical observations regarding the ability of the LSTM to learn to recognize $a^n b^n$ and $a^n b^n c^n$ are described also in (Gers and Schmidhuber, 2001).

We train 10-dimension, 1-layer LSTM and GRU networks to recognize $a^n b^n$ and $a^n b^n c^n$. For $a^n b^n$ the training samples went up to $n = 100$ and for $a^n b^n c^n$ up to $n = 50$.⁶

Results On $a^n b^n$, the LSTM generalizes well up to $n = 256$, after which it accumulates a deviation making it reject $a^n b^n$ but recognize $a^n b^{n+1}$ for a while, until the deviation grows.⁷ The GRU does not capture the desired concept even within its training domain: accepting $a^n b^{n+1}$ for $n > 38$, and also accepting $a^n b^{n+2}$ for $n > 97$. It stops accepting $a^n b^n$ for $n > 198$.

On $a^n b^n c^n$ the LSTM recognizes well until $n = 100$. It then starts accepting also $a^n b^{n+1} c^n$. At $n > 120$ it stops accepting $a^n b^n c^n$ and switches to accepting $a^n b^{n+1} c^n$, until at some point the deviation grows. The GRU accepts already $a^9 b^{10} c^{12}$, and stops accepting $a^n b^n c^n$ for $n > 63$.

Figure 1a plots the activations of the 10 dimensions of the $a^n b^n$ -LSTM for the input $a^{1000} b^{1000}$. While the LSTM misclassifies this example, the use of the counting mechanism is clear. Figure 1b plots the activation for the $a^n b^n c^n$ LSTM on $a^{100} b^{100} c^{100}$. Here, again, the two counting dimensions are clearly identified—indicating the LSTM learned the canonical 2-counter solution—although the slightly-imprecise counting also starts to show. In contrast, Figures 1c and 1d

⁶Implementation in DyNet, using the SGD Optimizer. Positive examples are generated by sampling n in the desired range. For negative examples we sample 2 or 3 n values independently, and ensuring at least one of them differs from the others. We dedicate a portion of the examples as the dev set, and train up to 100% dev set accuracy.

⁷These fluctuations occur as the networks do not fully saturate their gates, meaning the LSTM implements an imperfect counter that accumulates small deviations during computation, e.g.: increasing the counting dimension by 0.99 but decreasing only by 0.98. Despite this, we see that the its solution remains much more robust than that found by the GRU—the LSTM has learned the essence of the counting based solution, but its implementation is imprecise.

show the state values of the GRU-networks. The GRU behavior is much less interpretable than the LSTM. In the $a^n b^n$ case, some dimensions may be performing counting within a bounded range, but move to erratic behavior at around $t = 1750$ (the network starts to misclassify on sequences much shorter than that). The $a^n b^n c^n$ state dynamics are even less interpretable.

Finally, we created 1000-sample test sets for each of the languages. For $a^n b^n$ we used words with the form $a^{n+i} b^{n+j}$ where $n \in \text{rand}(0, 200)$ and $i, j \in \text{rand}(-2, 2)$, and for $a^n b^n c^n$ we use words of the form $a^{n+i} b^{n+j} c^{n+k}$ where $n \in \text{rand}(0, 150)$ and $i, j, k \in \text{rand}(-2, 2)$. The LSTM’s accuracy was 100% and 98.6% on $a^n b^n$ and $a^n b^n c^n$ respectively, as opposed to the GRU’s 87.0% and 86.9%, also respectively.

All of this empirically supports our result, showing that IBFP-LSTMs can not only theoretically implement “unbounded” counters, but also learn to do so in practice (although not perfectly), while IBFP-GRUs do not manage to learn proper counting behavior, even when allowing floating point computations.

6 Conclusions

We show that the IBFP-LSTM can model a real-time SKCM, both in theory and in practice. This makes it more powerful than the IBFP-SRNN and the IBFP-GRU, which cannot implement unbounded counting and are hence restricted to recognizing regular languages. The IBFP-IRNN can also perform input-dependent counting, and is thus more powerful than the IBFP-SRNN.

We note that in addition to theoretical distinctions between architectures, it is important to consider also the practicality of different solutions: how easy it is for a given architecture to discover and maintain a stable behavior in practice. We leave further exploration of this question for future work.

Acknowledgments

The research leading to the results presented in this paper is supported by the European Union’s Seventh Framework Programme (FP7) under grant agreement no. 615688 (PRIME), The Israeli Science Foundation (grant number 1555/15), and The Allen Institute for Artificial Intelligence.

References

- Roei Aharoni and Yoav Goldberg. 2017. [Towards string-to-tree neural machine translation](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 132–140, Vancouver, Canada. Association for Computational Linguistics.
- Yining Chen, Sorcha Gilroy, Kevin Knight, and Jonathan May. 2017. [Recurrent neural networks as weighted language recognizers](#). *CoRR*, abs/1711.05408.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Do Kook Choe and Eugene Charniak. 2016. [Parsing as language modeling](#). In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas. Association for Computational Linguistics.
- Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs]*.
- Jeffrey L. Elman. 1990. [Finding Structure in Time](#). *Cognitive Science*, 14(2):179–211.
- Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. 1968. [Counter machines and counter languages](#). *Mathematical systems theory*, 2(3):265–283.
- F. A. Gers and E. Schmidhuber. 2001. [Lstm recurrent networks learn simple context-free and context-sensitive languages](#). *Transactions on Neural Networks*, 12(6):1333–1340.
- F. A. Gers and J. Schmidhuber. 2000. [Recurrent nets that time and count](#). In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194 vol.3.
- K. Greff, R. K. Srivastava, J. Koutnk, B. R. Steunebrink, and J. Schmidhuber. 2017. [Lstm: A search space odyssey](#). *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. [Binarized neural networks](#). In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc.
- Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *arXiv:1504.00941 [cs]*.
- Hava Siegelmann. 1999. *Neural Networks and Analog Computation: Beyond the Turing Limit*, 1 edition. Birkhäuser Basel.
- Hava T. Siegelmann. 1996. [Recurrent neural networks and finite automata](#). *Computational Intelligence*, 12:567–574.
- Hava T. Siegelmann and Eduardo D. Sontag. 1992. [On the computational power of neural nets](#). In *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992.*, pages 440–449.
- Hava T. Siegelmann and Eduardo D. Sontag. 1994. [Analog computation via neural networks](#). *Theor. Comput. Sci.*, 131(2):331–360.
- Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. [Grammar as a foreign language](#). In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, NIPS’15*, pages 2773–2781, Cambridge, MA, USA. MIT Press.