Analog Stable Simulation of Discrete Neural Networks

Jiří Šíma *
Department of Theoretical Informatics
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8
Czech Republic
e-mail: sima@uivt.cas.cz

Abstract

The finite discrete-time recurrent neural networks are also exploited for potentially infinite computations (e.g. finite automata) where the input is being gradually presented from an external environment via input neurons. Because of gradient learning heuristics or analog hardware implementation reasons the usage of some continuous activation function is sometimes preferred rather than the discrete hard limiter (threshold function). However, in such cases the approximate representation of finite automaton states by analog network states can lead to an unstable behavior for long input sequences and consequently, to an incorrect resulting computation. Therefore, a stable simulation of any discrete neural network by an analog network of the same size is proposed. The simulation works in real time ('step per step') for any real activation function with different finite limits in improper points. In fact, only the weight parameters of the analog neural network are adjusted to achieve sufficient state precision. The same result holds for symmetric networks.

Keywords: simulation, analog neural network, discrete neural network, activation function, finite automaton

1 Introduction

The discrete-time recurrent neural network models are sometimes exploited as formal language acceptors in order to investigate their computational power.

**Acknowledgement:* I am grateful to Jiří Wiedermann for discussions which inspired the issues in this paper and I thank Jiří Červenka who found some formal mistakes in the manuscript. This research was supported by GA ČR Grant No. 201/95/0976.

The input word, being recognized by a neural network, can potentially be of an arbitrary length. There are two main input protocols introduced. Either the input is gradually read symbol after symbol during the computation by means of input neurons within a finite network or the infinite family of neural networks, each for one input length, is considered.

It has been known for a long time that finite discrete neural networks are, in fact, finite automata and thus, exactly recognize regular languages [8]. Introducing the continuous activation function (e.g. saturated-linear function, standard sigmoid, etc.) into finite, so-called analog neural networks, the computational power, depending on the Kolmogorov complexity of weight parameters [2], increases. For rational weights (i.e. rational network states) the (even real-time) simulation of the universal Turing machine can be achieved with a reasonable constant number of neurons by using a saturated-linear function [14, 6]. For the same activation function, except with real weights, the finite analog neural networks have super-Turing computational capabilities, e.g. polynomial time computations correspond to the complexity class $P/poly$ or even all languages can be recognized within the exponential time [13]. A similar simulation of the universal Turing machine is known for the standard sigmoid activation function [7]. Concerning the alternative input protocol, the (non-uniform) infinite families of discrete neural networks of the polynomial size correspond to the complexity class $PSPACE/poly$ or $P/poly$ when the symmetric polynomial weights are considered [12].

We will confine ourselves to finite automaton simulations by (finite) neural networks. For this case finer descriptive complexity measures are considered. The size of the discrete neural network (i.e. the number of neurons) is measured with respect to the number of (deterministic) finite automaton states [1]. It has been shown that a neural network with $O(\sqrt{m})$ neurons which simulates a given finite automaton with m states can be constructed [5, 6] and in the worst case, this network size cannot be decreased assuming either at most $O(\log m)$-time simulation delay [5] or polynomial weights [6]. An alternative approach compares the network size with the length of regular expressions. The construction of the discrete neural network of the optimal size $\Theta(n)$ corresponding to a regular expression of length n has been described [16, 15].

The above-mentioned constructions of the finite neural network which is equivalent with the given deterministic finite automaton or regular expression, employ the discrete hard limiter (threshold) activation function which makes encoding of automaton states into the network more direct and easier. However, the analog neural networks which exploit continuous (e.g. sigmoidal) activation functions are used in practical applications as well. 'They offer other advantages besides their use in gradient-based training algorithms; they also permit analog VLSI implementation, the foundations necessary for the universal approximation theories of neural networks, the interpretation of neural network outputs as posteriori probability estimates, etc.'[11]. Several experiments to train the analog neural networks from examples to behave like deterministic finite automata either for a practical exploitation or for a further rule extraction have been done [3, 10]. Typically, the so-called second-order analog neural networks, which, besides the continuous activation function, compute a polynomial of the second-order instead of linear weighted sum, are employed in this context [4, 17, 18]. However, the trained analog neural networks exhibit unstable behavior for longer input strings because the dynamic nature of recurrent networks together with the continuity of an activation function can cause the internal representation of finite automaton to deteriorate for longer computations [18].

Sometimes partial information (i.e. several transition rules) about the finite automaton which is being learned, is known. This prior knowledge can be used for the construction of the neural network which simulates the corresponding partial finite automaton. This network either serves as an initialization of the learning process to speed-up the training time or it is integrated with the network which was created by learning from examples [3]. That is why the construction of the analog second-order neural network (with the standard sigmoid activation function), which is proved to simulate the finite automaton in a stable way, has been proposed [11].

In our contribution we will show a similar construction but for simpler first-order analog neural networks which consist of neurons computing the activation function applied to the linear function of inputs. In addition, our simulation is more general. Given a finite discrete neural network with the hard limiter (threshold) activation function we are able to adjust its weight parameters in such a way that the corresponding analog network (of the same size) simulates the original network 'step per step' for arbitrarily long computations. This result can be achieved for any real activation function with different finite limits in improper points (including e.g. saturated-linear function, standard sigmoid, etc.) and works for symmetric networks as well. Moreover, the size of the analog network weights can be estimated and depends on the state representation precision which is (within specific bounds) an optional parameter of simulation. This means that the simulation is robust with respect to the specific rounding error. Applying the known results concerning the construction of the discrete neural network from the given deterministic finite automaton or regular expression, similar constructions for stable analog neural networks are obtained.

Recently, the proof technique used in our simulation has been independently discovered within the framework of noisy analog neural networks [9].

2 One-Neuron Simulation

We consider a *neuron* (*unit*) which computes its *state* (*output*) as a (generally) real function $y : X_n \longrightarrow \Re$ of its *input* $(x_1, \ldots, x_n) \in X_n \subseteq \Re^n$ in the following way:

$$y = \sigma\left(w_0 + \sum_{i=1}^{n} w_i x_i\right) \tag{1}$$

where $w_0, w_1, \ldots, w_n \in \Re$ are real weights and $\sigma : \Re \longrightarrow \Re$ is the so-called *activation function*. Further, we assume that this function has different finite limits a, b in improper points:

$$\lim_{x \to -\infty} \sigma(x) = a < \infty, \quad \lim_{x \to \infty} \sigma(x) = b < \infty, \quad a \neq b\,. \tag{2}$$

Particularly, we call this unit the *discrete neuron* if its input is from binary domain $X_n = \{0,1\}^n$ and the so-called *hard limiter* (*threshold*) activation function $\sigma_H : \Re \longrightarrow \{0,1\}$ is used:

$$\sigma_H(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0\,. \end{cases} \tag{3}$$

On the other hand, a typical example of the *analog neuron* is the unit which employs the so-called *standard sigmoid* (*logistic*) activation function $\sigma_S : \Re \longrightarrow (0,1)$:

$$\sigma_S(x) = \frac{1}{1+e^{-x}}\,. \tag{4}$$

In our simulation the discrete neuron states 0, 1 are represented by the analog ones using the limits a, b (2), respectively, within the precision $\delta > 0$ assuming at least $(a-\delta, a+\delta) \cap (b-\delta, b+\delta) = \emptyset$. We denote this representation and its inverse (with $\varepsilon > 0$ precision satisfying the same condition) as follows:

$$r_\delta[x] \in \begin{cases} (a-\delta, a+\delta) & x = 0 \\ (b-\delta, b+\delta) & x = 1 \end{cases} \qquad r_\varepsilon^{-1}[y] = \begin{cases} 0 & y \in (a-\varepsilon, a+\varepsilon) \\ 1 & y \in (b-\varepsilon, b+\varepsilon)\,. \end{cases} \tag{5}$$

This means, that $r_\delta[x]$ stands for any number within the relevant interval.

Now, we formulate the main result stating that the weights of the discrete neuron can be adjusted so that the corresponding analog neuron with the activation function satisfying (2) computes the same function with respect to the representation (5) while the required representation precision is preserved.

Theorem 1 *For every discrete neuron with weights $w_0, \ldots, w_n \in \Re$, for any activation function σ with the different finite limits a, b in improper points (2), i.e.*

$$\forall \varepsilon > 0 \ \exists \alpha_1 \ \forall x < \alpha_1 \qquad |\sigma(x) - a| < \varepsilon \tag{6}$$

$$\forall \varepsilon > 0 \ \exists \alpha_2 \ \forall x > \alpha_2 \qquad |\sigma(x) - b| < \varepsilon\,, \tag{7}$$

for any input precision $\delta > 0$ which equals

$$\delta = k_\delta \frac{\xi|b-a|}{2nw} \qquad \textit{where } 0 < k_\delta < 1 \tag{8}$$

$$\xi = \min\left\{\left|w_0 + \sum_{i=1}^{n} w_i x_i\right| \neq 0\,;\, x_1,\ldots,x_n \in \{0,1\}\right\}\,,\quad w = \max_{1\leq i\leq n} |w_i|\,, \tag{9}$$

and for any output precision $0 < \varepsilon \leq \delta$ there exists an analog neuron with the activation function σ and weights $v_0,\ldots,v_n \in \Re$ as follows:

$$v_0 = K\left(w_0 - \sum_{i=1}^{n} \frac{a w_i}{b-a} + \frac{\xi}{2}\right) \qquad v_i = \frac{K w_i}{b-a} \quad i = 1,\ldots,n \tag{10}$$

where $K > 0$ is any positive number satisfying

$$K > \frac{2\max(-\alpha_1, \alpha_2)}{\xi(1-k_\delta)} \tag{11}$$

and α_1, α_2 are taken from (6), (7) for the output precision $\varepsilon > 0$, such that

$$\sigma_H\left(w_0 + \sum_{i=1}^{n} w_i x_i\right) = r_\varepsilon^{-1}\left[\sigma\left(v_0 + \sum_{i=1}^{n} v_i r_\delta[x_i]\right)\right]. \tag{12}$$

Proof: First, the bias $w_0' = w_0 + \xi/2$ where $\xi > 0$ is defined by (9), is adjusted to achieve the inequality $w_0' + \sum_{i=1}^{n} w_i x_i \neq 0$ for all binary inputs $x_1,\ldots,x_n \in \{0,1\}$. Moreover, from (9) it follows

$$w_0 + \sum_{i=1}^{n} w_i x_i \geq 0 \quad\longrightarrow\quad w_0' + \sum_{i=1}^{n} w_i x_i \geq \frac{\xi}{2} \tag{13}$$

$$w_0 + \sum_{i=1}^{n} w_i x_i < 0 \quad\longrightarrow\quad w_0' + \sum_{i=1}^{n} w_i x_i \leq -\frac{\xi}{2}\,. \tag{14}$$

Further, the linear transformation $\Phi : \Re \longrightarrow \Re$ of the input domain is defined in order to map the limits a, b to the discrete binary inputs 0, 1, respectively, i.e. $\Phi(a) = 0$, $\Phi(b) = 1$:

$$\Phi(y) = \frac{y-a}{b-a}\,. \tag{15}$$

This transformation is taken into account when defining new weights $v_0',\ldots,v_n'$:

$$\begin{aligned} w_0' + \sum_{i=1}^{n} w_i \Phi(y_i) &= w_0' + \sum_{i=1}^{n} w_i \frac{y_i - a}{b-a} \quad = \\ &= \underbrace{w_0' - \sum_{i=1}^{n} \frac{w_i a}{b-a}}_{v_0'} + \sum_{i=1}^{n} \underbrace{\frac{w_i}{b-a}}_{v_i'} y_i \quad = \quad v_0' + \sum_{i=1}^{n} v_i' y_i \end{aligned} \tag{16}$$

and when the input precision is considered:

$$|y - y'| < \delta \quad\longleftrightarrow\quad |y - a - (y' - a)| < \delta \quad\longleftrightarrow$$

$$\longleftrightarrow \quad \left| \frac{y-a}{b-a} - \frac{y'-a}{b-a} \right| < \frac{\delta}{|b-a|} \quad \longleftrightarrow \quad |\Phi(y) - \Phi(y')| < \frac{\delta}{|b-a|} \,. \tag{17}$$

Finally, define $v_0 = Kv'_0$ and $v_i = Kv'_i$ $(i = 1, \ldots, n)$ for a sufficiently large positive $K > 0$. Consider the weighted sum $v_0 + \sum_{i=1}^n v_i r_\delta[x_i]$ of the analog neuron from (12) corresponding to some binary inputs $x_i \in \{0,1\}$ $(i = 1, \ldots, n)$. Denote by $y_i = \Phi^{-1}(x_i) \in \{a, b\}$ the precise analog inputs and by $y'_i = r_\delta[x_i] \in (a-\delta, a+\delta) \cup (b-\delta, b+\delta)$ the actual analog inputs within the required input precision. Similarly, $x'_i = \Phi(y'_i)$ denote the approximate values of binary inputs. By using (16) we can compute:

$$v_0 + \sum_{i=1}^n v_i r_\delta[x_i] = K\left(v'_0 + \sum_{i=1}^n v'_i y'_i\right) = K\left(w'_0 + \sum_{i=1}^n w_i \Phi(y'_i)\right) =$$

$$= K\left(w'_0 + \sum_{i=1}^n w_i x'_i\right) = K\left(w'_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n w_i (x'_i - x_i)\right) \,. \tag{18}$$

It is clear that $|y'_i - y_i| < \delta$ which is equivalent with $|\Phi(y'_i) - \Phi(y_i)| < \delta/|b-a|$ due to (17). Then the absolute value of the last term in (18) can be estimated as follows:

$$\left| \sum_{i=1}^n w_i (x'_i - x_i) \right| \leq \sum_{i=1}^n |w_i| \cdot |\Phi(y'_i) - \Phi(y_i)| < nw \frac{\delta}{|b-a|} = k_\delta \frac{\xi}{2} \,. \tag{19}$$

Now, we will prove (12). First, suppose that $\sigma_H(w_0 + \sum_{i=1}^n w_i x_i) = 1$, i.e. $w_0 + \sum_{i=1}^n w_i x_i \geq 0$ from (3) which implies

$$w'_0 + \sum_{i=1}^n w_i x_i \geq \frac{\xi}{2} \tag{20}$$

due to (13). We want to prove $r_\varepsilon^{-1}[\sigma(v_0 + \sum_{i=1}^n v_i r_\delta[x_i])] = 1$ which is equivalent with

$$\left| \sigma\left(v_0 + \sum_{i=1}^n v_i r_\delta[x_i]\right) - b \right| < \varepsilon \tag{21}$$

because of (5). From (7) it follows that α_2 exists such that for

$$v_0 + \sum_{i=1}^n v_i r_\delta[x_i] > \alpha_2 \tag{22}$$

the inequality (21) holds. Therefore, it is sufficient to prove (22) which can be rewritten according to (18) as follows:

$$K\left(w'_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n w_i (x'_i - x_i)\right) > \alpha_2 \,. \tag{23}$$

Furthermore, from (20), (19) and the fact that $K > 0$ is positive it suffices to prove

$$K\left(\frac{\xi}{2} - k_\delta \frac{\xi}{2}\right) > \alpha_2\,. \tag{24}$$

The assumption (8) implies $(1 - k_\delta)\xi/2 > 0$ and thus, any $K > 0$ satisfying

$$K > \frac{2\alpha_2}{\xi(1 - k_\delta)} \tag{25}$$

according to (11) implies (24).

The case $\sigma_H(w_0 + \sum_{i=1}^{n} w_i x_i) = 0$ is analogous. Instead of (13), (7) the formulae (14), (6), respectively, are applied. □

For a particular activation function the weights can be expressed more explicitly. For example, concerning the standard sigmoid σ_S (4) the limits (2) are $a = 0$, $b = 1$. Furthermore, α_1, α_2 from (6), (7) can be determined for any $\varepsilon > 0$ as follows:

$$\alpha_1 = \sigma_S^{-1}(\varepsilon) = \ln \frac{\varepsilon}{1 - \varepsilon} \qquad \alpha_2 = \sigma_S^{-1}(1 - \varepsilon) = \ln \frac{1 - \varepsilon}{\varepsilon}\,. \tag{26}$$

Then $K > 0$ can be chosen according to (11) so that the size of the new weights, which compare to the old ones, is estimated by (10) as follows:

$$|v_i| = O\left(\frac{\ln \frac{1-\varepsilon}{\varepsilon}}{\xi(1 - k_\delta)}|w_i|\right) \qquad i = 0, \ldots, n\,. \tag{27}$$

This means that the weight size increases when the input is more inaccurate (δ increases) or the output is required to be more precise (ε decreases).

3 Neural Network Simulation

First, we briefly recall some basic notions from neural network terminology. The *recurrent neural network* consists of neurons which are connected in a generally cyclic oriented graph (the so-called *architecture*) with edges labelled with relevant weights. The *symmetric* (*Hopfield*) neural networks in which the architecture is an undirected graph with symmetric weights, are considered as well. The discrete-time computation starts in the so-called *initial state*. In a general step, the input neurons are possibly set to the next bits of an external input and some (typically, either one or all) non-input neurons compute their new states according to formula (1) where the states of neighbour neurons represent their inputs. Possibly, the so-called *output neurons* exhibit a result of computation. Further, the *discrete neural network* consists of discrete neurons and a typical *analog neural network* employs, for example, the units with the standard sigmoid activation function (4).

Theorem 1 is used for a stable analog simulation of discrete neural networks. The representation of a discrete network state by the analog one is a vector of the single neuron state representations (5). In our real-time ('step per step') simulation, the analog computation proceeds the same way as in the discrete case but for the analog state representations. For this purpose, the weights of the discrete network are adjusted for analog computation using Theorem 1. The output precision $0 < \varepsilon \leq \delta$ is less or equal than the input precision $\delta > 0$ in order to preserve the representation precision which ensures the stable simulation. It is also clear that the simulation can be made robust with respect to a rounding error within specific bounds. For the symmetric networks, the parameter $K > 0$ is chosen to be the same value for all neurons in the network (e.g. maximum of K's in (11) over all neurons) to maintain the symmetric weights. These considerations are summarized in the following theorem.

Theorem 2 *For every finite discrete recurrent (symmetric) neural network, for any activation function σ satisfying (2), and any representation precision $\delta > 0$ satisfying (8) for all neurons, a (symmetric) analog network of the same size exists, with the activation function σ, which simulates the original discrete neural network in a real time and a stable way for arbitrarily long computations.*

By applying theorem 2 to the known results [5, 16, 15], the constructions of the analog (even symmetric, for the so-called Hopfield languages [15]) neural network acceptor of the size $O(\sqrt{m})$ which is equivalent to the given deterministic finite automaton with m states, or with $O(n)$ neurons for the regular language described by the regular expression of the length n, are obtained.

References

[1] Alon, N., Dewdney, A. K., Ott, T. J.: Efficient simulation of finite automata by neural nets. *Journal of the ACM*, **38**, 495–514, 1991.

[2] Balcázar, J. L., Gavaldà, R., Siegelmann, H. T.: Computational power of neural networks: A Kolmogorov-like complexity characterization. In *IEEE Transactions of Information Theory*, 1996.

[3] Frasconi, P., Gori, M., Maggini, M., Soda, G.: A unified approach for integrating explicit knowledge and learning by example in recurrent networks. In *Proceedings of the International Joint Conference on Neural Networks*, vol. **1**. IEEE, New York, 881–816, 1991.

[4] Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., Lee, Y. C.: Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, **4**, 393–405, 1992.

[5] Horne, B. G., Hush, D. R.: Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks*, **9**, 243–252, 1996.

[6] Indyk, P.: Optimal simulation of automata by neural nets. In Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, vol. **900** of LNCS, 337-348. Springer-Verlag, Berlin, 1995.

[7] Kilian, J., Siegelmann, H. T.: Computability with the classical sigmoid. In *Proceedings of the 6th ACM Conference on Computational Learning Theory*, 200-208. ACM Press, Las Vegas, 1995.

[8] Kleene, S. C.: Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, vol. **34** of *Annals of Mathematics Studies*, 3–41. Princeton University Press, Princeton NJ, 1956.

[9] Maass, W., Orponen, P.: On the Effect of Analog Noise in Discrete-Time Analog Computations. Manuscript, May 1997.

[10] Manolios, P., Fanelli, R.: First-order recurrent neural networks and deterministic finite state automata. *Neural Computation*, **6**, 1155–1173, 1994.

[11] Omlin, C. W., Giles, C. L.: Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, **43**, 937–972, 1996.

[12] Orponen, P.: The computational power of discrete Hopfield nets with hidden units. *Neural Computation*, **8**, 403-415. 1996.

[13] Siegelmann, H. T., Sontag, E. D.: Analog computation via neural networks. *Theoretical Computer Science*, **131**, 331–360, 1994.

[14] Siegelmann, H. T., Sontag, E. D.: Computational power of neural networks. *Journal of Computer System Science*, **50**, 132–150. 1995.

[15] Šíma, J.: Hopfield languages. In M. Bartošek, J. Staudek, and J. Wiedermann, editors, *Proceedings of the SOFSEM Seminar on Current Trends in Theory and Practice of Informatics*, vol. **1012** of LNCS, 461–468, Milovy, 1995. Springer-Verlag, Berlin, 1995.

[16] Šíma, J., Wiedermann, J.: Neural language acceptors. In *Proceedings of the 2nd International Conference Developments in Language Theory*, 430–439, Magdeburg, 1995. World Scientific, Singapore, 1996.

[17] Tiňo, P., Šajda, J.: Learning and extracting initial mealy automata with a modular neural network model. *Neural Computation*, **7**, 822–844, 1995.

[18] Zeng, Z., Smyth, P.: Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, **5**, 976–990, 1993.