Natural Language Processing with Recurrent Neural Networks as Finite-State Automata with External Parallel-Access Memory

Roman Pozarlik

Wroclaw University of Technology, Institute of Engineering Cybernetics
Janiszewskiego Str. 11-17, PL 50-370 Wroclaw, Poland
E-mail:rpoz@pwr.wroc.pl, http://www.pwr.wroc.pl/~rpoz/

Abstract

In this paper a model of finite-state Parallel-Access Memory Automaton (PAMA) is presented. PAMA has a finite memory on which it operates in a massively-parallel manner. Therefore it is useful as a formal model of computations in binary recurrent neural networks. PAMA introduces a new type of finite computations, which are *non-structural*. They are based not on manipulating symbolic structured representations but on associations between *casually* constructed binary vectors (causal representations). A method of constructing such vectors has been proposed in a definition of PAMA called Neural-Finite State Transducer (NFST).

NFST computations are very suitable for NLP problems although they are finite. It is argued that the traditional notion of computational power, which favors models of infinite-state computations, is misleading because such models are not available in practice. Moreover it does not take into account the problem of designing such models. NFST represents a different view, which favors finite-state models of computations, including neural networks, as more appropriate in the NLP domain.

1. Introduction

In this paper we would like to present an idea that neural networks can perform a new type of computations which cannot be carried out by the traditional models. We call them *causal computations* because the only formal requirement to carry out such computations is the principle of causality. Computations performed by classical models, such as Turing Machine (TM), are also causal but in addition to that they must be structural – they are carried out by manipulating structured representations. We use the term *causal* to say that computations do not have to be structural.

The idea of causal computations was informally introduced in [3, 4] by showing that massively-parallel computations, such as neural computations, do not have to be based on manipulating structured representations (e.g. lists representing graphs), assuming that such representations are constructed causally. This conclusion is drawn from the observation that in such a case all the data taking part in computations are available for processing simultaneously and can be modified at a single step of computations. Therefore they do not have to encode any constituent symbolic structures because they can be defined arbitrarily. Also processing of such arbitrary representations does not have to be structure-sensitive because it can be defined in a look-up table manner using, for instance, associative neural networks. Massively-parallel style of computations may also lead to a different than our conclusion that symbolic structures are in such a case processed holistically. However, we do not find any *formal* necessity for such an assumption.

The idea of causal computations is very simple and extremely useful in symbol processing tasks such as Natural Language Processing (NLP). At the same time, it is not very effective for ordinary, that is numerical, computations. The parallel style of processing and the distributed form of representations imply that problems, which are composed from numerous but simple steps of processing, become difficult. On the other side, problems, in which many constraints determine a solution, will be easier to implement because all such constraints can be taken into account at the same time (i.e. at a single step of computations). NLP is a problem of the second type.

We present the idea of causal computations by defining Neural-Finite State Transducer (NFST), which is a specific type of Parallel-Access Memory Automaton (PAMA). PAMA is combined from Finite-State Automaton (FSA) and external memory, thus resembling Push-Down Automaton (PDA) or Turing Machine (TM). However, PAMA has a finite memory with a massively-parallel access but not a potentially infinite stack or tape with a serial access. This modification significantly changes the way, how computations are carried out. In order to explain this new idea of computations PAMA will be compared with finite versions of PDA and TM, called Finite PDA (FPDA) and Finite TM (FTM). Moreover, it will be shown that PAMA is naturally implemented using Binary Recurrent Neural Network (BRNN). This result is especially interesting

because there is a sharp boundary between TM computations and finite-state computations with the second ones considered as less powerful.

This paper is organized as follows. In Section 2 we introduce formally PAMA and describe its relationship with FPDA, FTM, multi-tape FTM, and BRNN. Section 3 defines NFST and shortly depicts its basic properties. In Section 4 we discuss if and how NFST can be useful in the NLP domain. Also in this section we analyze the notion of computational power and a new status of finite-state computations that should be defined by reason of the discrepancy between formally postulated capabilities of infinite-state models and practically available finite-state models. Section 5 is the summary of the paper.

2. PAMA

In this section we present a model of FSA with an external finite parallel-access memory. We call this model Parallel-Access Memory Automaton (PAMA). Formally PAMA can be defined as follows.

Definition 1 Parallel-Access Memory Automaton (PAMA) is a 9-tuple $(Q, U, Y, X, \delta, g, q_0, x_0, F)$, where:

Q - is a finite set of states,
U - is a finite set of input symbols,
Y - is a finite set of output symbols,
$X \subseteq \{0, 1\}^N$ - is a finite set of memory configurations,
δ: $Q \times (U \cup \{\varepsilon\}) \times X \to Q \times X$ - is the state transition function,
g: $Q \times X \to Y$ - is the output function,
$q_0 \in Q$ - is the initial state,
$x_0 \in X$ - is the initial memory configuration,
$F \subset Q$ - is the set of accepting states.

We may represent δ by a list of 5-tuples, with $(q_i, u_i, x_i, q_j, x_j)$ in the list just in case $\delta(q_i, u_i, x_i) = (q_j, x_j)$. PAMA has an N-bit memory, denoted by X. We assume that X is a parallel-access memory, which means that its whole contents (configuration) can be modified during a single write operation and can determine state transition throughout a single read operation. We denote also by X the set of possible configurations. PAMA starts computations from the initial state q_0 and initial memory configuration x_0. It reads an input symbol $u_i \in U$, which can be the empty ε symbol, and makes a transition to the next state and next memory configuration. It accepts a sequence of input symbols if after reading the last one it achieves some state $q \in F$. For transition depends on state and memory configurations it is not very useful to analyse PAMA using an ordinary transition graph.

2.1 Finite-State Automata, Push-Down Automata, Turing Machines and PAMA

From the definition of PAMA it follows that an ordinary FSA is PAMA with $X = \varnothing$. There is also an FSA computationally equivalent for any PAMA because PAMA is a finite-state model of computations.

In the case of infinite-state automata such as Push-Down Automaton (PDA) and Turing-Machine (TM), PAMA has lower computational power in the sense of its powerlessness to perform infinite computations. However no practically available model of computations can perform infinite computations. We will discuss this problem in Section 4. Let us now compare PAMA with finite versions of PDA or TM, called Finite PDA (FPDA) and Finite TM (FTM).

In PAMA transition depends on the whole memory configuration and, at the same time, this configuration can be modified without any restrictions during a single transition. Hence it can be shown that FPDA and FTM are special cases of PAMA.

In the first case X is a finite stack. It means that there is a serial-access to X defined in such a way that only symbol on the stack's top can be changed or read during a single transition. Only this symbol can be meaningful for transition. FPDA may be defined as follows.

Definition 2 Finite Push-Down Automaton (FPDA) is PAMA with δ defined as follows:

δ: $Q \times (U \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma$, where Γ represents finite stack.

FPDA can be viewed as PAMA with a very restricted serial-access to X, denoted here by Γ. Such a restriction provides the simplest solution to the problem of using X. In other words it is easy to perform reading and writing operations with stack.

In the case of FTM X is a tape. During computations only one symbol in X is significant for δ at a time step. FTM can be defined as follows.

Definition 3 Finite Turing-Machine (FTM) is PAMA with δ defined as follows:

δ: $Q \times T \to Q \times T \times \{L, N, R\}$, where T is a finite set of tape symbols and $\{L, N, R\}$ indicates possible moves of the read/write head (L - left, R - right, and N - no move); we assume that X is a finite tape.

Here X stores tape symbols, which can be accessed only one per time step. FTM offers a bit less re-

stricted access to X than FPDA, but nonetheless it is still serial.

As we know, it is postulated that any computation, which can be formally defined, can be performed using this type of serial access to X, assuming that the tape is infinite.[1] It can lead us to a conclusion that there are not more powerful models of computations. In other words, a less restricted access to X, for instance a parallel-access, gives us nothing because all such models' computations can be reduced to TM computations. Formally speaking, it is true in the case of PAMA because it performs finite-state computations. Yet, we must take into account the translation problem. It may be practically, although not formally, possible to translate one style of computations into another one. Therefore, problems that can be easily solved by some model can be very difficult for another one, even if they have the same computational power in the present sense of this term. For instance, we would face such a situation if we were trying to use FSA to perform computations of the present-day computers, which actually perform finite-state computations. We assert that finite-state models are not computationally equivalent if we take into account not only their capabilities of performing infinite computations but in the first place the problem of programming such models. It means that a less restricted non-serial access to X may reveal a new style of computations.

One approach to a non-serial access to X is FTM with more than one tape. Such a model can be defined as follows.

Definition 4 Multi-tape Finite Turing-Machine (MFTM) with K tapes is PAMA with δ defined as follows:

δ: $Q \times T^K \rightarrow Q \times T^K \times \{L, N, R\}^K$, where T and $\{L, N, R\}$ are defined as in Definition 3; we assume that X is divided on K parts representing tapes.

In order to carry out computations with MFTM we must use the same technique as in the FTM case. Thus this model does not offer any new style of computations. In this case we can talk about parallel access to X but not about massively-parallel access because usually $K << N$, where $N = |X|$. We can talk about massively-parallel access if $K = N$. Such an access can provide neural networks.

2.2 Recurrent Neural Networks and PAMA

Before we compare Recurrent Neural Networks and PAMA let us simplify a bit its definition. Since PAMA has a finite number of states, they can be encoded into X using additional S bits. In this way PAMA with 2^S states and N-bit memory can be equivalently described as PAMA with no states and $S+N$-bit memory.

Let us now consider a Binary RNN (BRNN) build from M inputs, L outputs, and N hidden neurons. The input and output neurons may have linear or sigmoid activation functions. The hidden layer is composed from threshold elements. Its activation is denoted by binary vector x_n, that is $x_n \in \{0, 1\}^N$, where n indicates a time step. We assume that the network is fully connected and some neurons are designated as input and some as output ones. A particular layout of connections can be achieved by assigning zero weights for some of the connections. Such a network can be defined as PAMA in the following way.

Definition 5 Binary Recurrent Neural Network (BRNN) is PAMA with δ and g defined as follows:

δ: $(U \cup \{\varepsilon\}) \times X \rightarrow X$: $\delta(x_{n-1}, u_n) \mapsto x_n$

$x_n = f(\ W^H x_{n-1} + W^I u_n + \theta\ )$,

g: $X \rightarrow Y$: $g(x_n) \mapsto y_n$

$y_n = f(\ W^O x_n + \vartheta\ )$, where W^H, W^I and W^O indicate weight matrices and θ and ϑ are bias vectors. $f(.)$ is a threshold function ($f(x) = 1$ if $x > 0$ and $f(x) = 0$ for $x \leq 0$).

This interpretation of BRNN as PAMA is straightforward because binary neural networks operate on N-dimensional binary vectors in a massively-parallel manner. It is important in this case that there are no formal limitations imposed on the type of memory modifications that can be performed in a single step of computations, assuming that in general W^H, W^I and W^O can represent mappings of multilayer perceptrons. X can be modified without any of the restrictions that are present in the FPDA, FTM and MFTM case. What does this fact imply?

If we compare this PAMA with that representing FTM and a problem of computations such as transduction, then FTM computations are reduced in the case of BRNN to a few steps of computations directly connected with the sequences of input and output symbols (see [3, 6] for more details). It can mean that either BRNN performs the same computations as FTM by manipulating structured (symbolic) representations encoded into x_n, or it does not manipulate such representations. If the first answer were correct, then we would be able to extract from x_n such structures using some data analysis tools. Actually, we can often find in X a hierarchical structure of similarities between x_n using, for instance, cluster analysis. However, and this is our crucial argument, X can be formally any set of arbitrarily defined vec-

[1] In practice, computers prove that this postulate is true even though formally they are not TMs but FTMs.

tors. Hence we can talk about non-structural processing as the fundamental paradigm. Moreover, the systematic processes of associative mappings, engaged in BRNN computations, most probably will not produce completely unrelated vectors x_n. There should emerge structures, as such which are revealed by cluster analysis, but these are not the constituent symbolic structures generated by structure-sensitive processes in the models of serial computations.

3. Neural Finite-State Transducers

In Section 2 we have shown that BRNN can provide an adequate implementation for causal computations, which can be performed by PAMA. However, Definitions 1 and 5 do not specify how to carry out causal computations. It follows from our discussion in the previous section that in order to perform such computations we must explain how to construct x_n causally without going back to the structural solution. An equation of causal construction of x_n has been introduced by analysing BRNN computations during sequential symbol processing [4, 5]. PAMA equipped with such an equation of causal construction of memory configurations becomes a formal model of massively-parallel computations. BRNN is only an implementation of computations in such model (probably the most appropriate one). We express this idea in the following definition of Neural Finite-State Transducer (NFST).

Definition 6 Neural Finite-State Transducer (NFST) is PAMA where δ is defined as follows:

$$\delta: U \times X \to X: \delta(u_n, x_{n-1}) \mapsto x_n$$

$$x_n = \alpha(u_n) \vee \beta(u_n) \wedge (x_{n-1} \vee \gamma(x_{n-1})), \qquad (1)$$

where $\alpha: U \to \{0, 1\}^N$, $\beta: U \to \{0, 1\}^N$, and $\gamma: X \to \{0, 1\}^N$ are some auxiliary functions.

Equation (1) describes how memory configuration at time step n, denoted x_n, is determined by memory configuration at time step n-1, denoted x_{n-1}, and input symbol u_n. We have three auxiliary functions in this equation: $\alpha(.)$, $\beta(.)$ and $\gamma(.)$. As we can observe they represent mappings typical for multilayer feed-forward networks. Formally, they can be defined by finite sets of pairs of vectors (look-up tables). $\alpha(.)$ and $\beta(.)$ can be easily interpreted in this equation: they encode inputs as vectors which are then used in the construction of x_n. $\gamma(.)$ is a bit more troublesome because it is defined by a finite set of pairs $\gamma_i = (a_i, b_i) \in \{0, 1\}^N \times \{0, 1\}^N$. In order to find $\gamma(x_{n-1})$ we must check which a_i fits x_{n-1}. We could use the simplest criterion $a_i = x_{n-1}$, but it is better to use the following criterion of partial fitting, denoted $a_i \approx x_{n-1}$,

$$\prod_{k=1}^{N} (x_{n-1,k} \vee \bar{a}_{i,k}) = 1, \qquad (2)$$

where $x_{n-1,k}$ denotes k-th coordinate of x_{n-1} and $\bar{a}_{i,k}$ is a complement of the k-th coordinate of a_i. Equation 2 says that a_i does not fit x_{n-1} if $x_{n-1,k} = 0$ and $a_{i,k} = 1$ for some k. If we interpret x_n as a state vector of some BRNN, then Equation (2) expresses a simple idea that only active neurons determine the network's transition from time step n-1 to n. We also assume that usually only some of the active neurons suffice to determine x_n and thus, according to Equation 2, many pairs γ_i may fit to x_{n-1} to different degrees. Therefore, it is necessary to define a criterion of best fitting. We say that a_i fits to x_{n-1} best, which will be denoted by writing $a_i \cong x_{n-1}$, if the following criterion is satisfied for all $a_j \approx x_{n-1}$

$$\sum_{k=1}^{N} a_{i,k} \geq \sum_{k=1}^{N} a_{j,k}. \qquad (3)$$

According to this criterion we can determine $\gamma(x_{n-1})$ as follows,

$$\gamma(x_{n-1}) = \begin{cases} b_i & \text{if } a_i \cong x_{n-1}, \\ \bar{0} & \text{otherwise.} \end{cases} \qquad (4)$$

We will find $y_n = g(x_n)$ in a similar way assuming that $g: X \to Y$. The components of Equation 1 can be easily located in the Elman-style RNN shown in Figure 1 (for more details on this interpretation see [5, 6]).

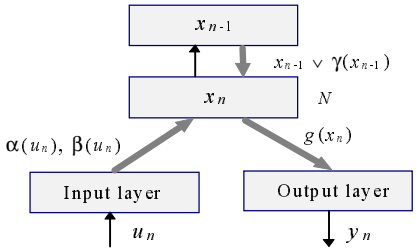

Figure 1: An Elman-style RNN and NFST's components.

The input to hidden layer mapping is described by two functions: $\alpha(.)$ and $\beta(.)$. The hidden to output layer map is described by the output function, $g(.)$. A relation between x_{n-1} and x_n is defined as $x_{n-1} \vee \gamma(x_{n-1})$. Taken together u_n and x_{n-1} form a new state x_n in a manner described by Equation (1). In NFST $\alpha(.)$, $\beta(.)$, and $\gamma(.)$ are defined in a look-up table manner. Therefore, they can be naturally implemented using multilayer feed-forward networks.

The NFST's operation is described by a sequence

x_0, $x_1 = \delta(u_1, x_0)$, $x_2 = \delta(u_2, \delta(u_1, x_0))$, ..., x_n, where some sequence of inputs $u_1\, u_2 \ldots u_n$ undergoes transduction. If the input sequence leads to a sequence $x_0\, x_1 \ldots x_n$, where $x_n \in F$, then the sequence is accepted by the NFST. We have shown in [4] that Equation (1) with appropriately defined functions $\alpha(.)$, $\beta(.)$, and $\gamma(.)$ will provide unique representations for all memory configurations during such computations in the class of regular languages exact to iteration. It means that we can define appropriately the output function for any transduction task in this class of languages.

4. NFST and NLP

In this section we would like to discuss such issues as: i) can NFST be useful in solving NLP problems; ii) how can we use it in this domain; iii) what type of advantages over classical models can it provide.

We have presented in other papers [4, 6] exemplary definitions of NFST for simple NLP problems. The difficulties that we have encountered in designing these NFSTs seem to be as hard as we face in the classical approach. Fortunately, they have quite a different nature than in this approach and we think that they will be successfully overcome using some learning techniques. In the classical symbolic approach we face the problem of designing complex structured representations and rules of manipulation of such representations. It is hard to design such a learning procedure that would produce the same structured representations. In our approach, representations are non-structured and thus can be modified in a simple procedure of adjusting $\alpha(.)$, $\beta(.)$, $\gamma(.)$, and $g(.)$. At present, these functions can be defined only after the complete set of training data is analysed. We do not have at our disposal any method of gradual adjusting these functions, which method would successfully solve the problem for huge data sets encountered in real NLP problems.

In its most basic definition NLP is the problem of transduction but in the domain of symbolic computations it is viewed more specifically as a problem of designing and manipulating some formal representations of the language syntax and semantics. This situation is caused by the fact that computations are separated from the expected transduction of input to output sequences [4]. A classical model of serial computations cannot operate in real-time and the sequential nature of its input/output data and the transduction process loose their significance in favour of internal computational processes. In the case of NFST we have a quite different situation. NFST carry out computations by reading inputs and generating outputs in real-time. There is no need for any *internal computations* (additional state transitions) because there are no restrictions imposed on transitions between adjacent in time states of computations. Therefore our approach to NLP is completely different from the classical one and does not comprise such traditionally stated problems as, for instance, parsing. We concentrate on the problem of transduction.

In the definition of NFST, Equation (1) is called state transition function because it determines state (memory configuration) at the next time step. In spite of that, it does not decide what transductions are allowed. Transitions are forced by inputs, which may depend on the output function. In the real-time computations NFST always reads an input symbol or produces an output symbol. In the first case transitions are forced by the sequence of inputs. In the second case transitions are defined by output symbols which are fed back as input symbols at the next time step (see [5, 6] for more details).

Besides trying to design NFSTs we can conclude about neural implementations of causal computations. Our interpretation of computations in BRNNs, which is expressed in the definition of NFST, implies how such networks should be built. According to the paradigm of causal computations BRNN must maintain a causal relationship between consecutive patterns of activity. It means that it must store some information about inputs and patterns at time steps n-i, n-i+1, ..., n-1 in order to make appropriate transition at time step n. This formal requirement implies that such a network should have some recurrent connections, maybe should use more localised representations to constraint possible interference between patterns, and probably should have a modular architecture to keep track of previous event at different time scales. We think that such conclusions guide us into right direction because we do not make a-priori any unnecessary assumptions, such as that of processing symbolic structured representations.

An important problem that must be discussed in this section is computational power of NFST. Finite-state models of computations are postulated to have too lower computational power to be used in NLP problems. For instance, they cannot process recursively-nested clauses. We want to state that this postulate is a deep misunderstanding that stems from mixing up theoretical and actual capabilities of the automata. It is true that all present-day computers are finite-state models of computations and thus are formally equivalent to some FSAs. At the same time natural language, as normally performed, does not require any type of infinite-state computations. The recursively-nested clauses are actually very limited [7]. It means that the term *computational power,* which we intuitively interpret – easy to design and powerful to solve difficult intelligent information

processing problems –, does not depend on the model's capability to perform infinite computations but on the type of memory it uses. We hope that NFST will support this statement very well.

Recently a successful effort has been taken to use neural networks to simulate TM computation [8]. Many scientists have felt relief that neural networks are enough powerful and research in this domain will not be stopped as it was thirty years ago by the XOR problem. Of course, there are also results, which say that such infinite-state computations cannot be performed in practice because they would require computations with infinite precision or infinite size of a network [1, 2]. It does not trouble computer scientists despite finite resources of the present-day computing devices. It will probably not trouble also neural network scientists. But neural networks do not seem to be scalable in such a way as classical models of computations. Adding neurons to a trained network will usually not increase its capability to solve bigger problems of the same type that it has just learned to solve. Therefore this problem is more serious and trying to overcome it in favour of classical models is, in our view, a wrong direction that neural network research should not follow.

5. Conclusions

In this paper we have shown that finite versions of classical models of computations, called FPDA and FTM, can be defined as specific instances of Parallel-Access Memory Automaton (PAMA). Moreover we have shown that PAMA can be naturally implemented in Binary Recurrent Neural Network (BRNN). It does not mean, however, that BRNN implements either FPDA or FTM computations. The classical models and neural networks satisfy the same principle of causality but the classical models require also computations to be structural. Such a requirement is implied by the serial access to an external memory and is not necessary in the case of BRNN. Therefore we have introduced the term *causal* for computations, which are causal but *non-structural*. In such computations only a causal relation between consecutive states of computations has to be maintained. In FPDA and FTM it is achieved by means of manipulating structured representations. In the case of non-structural processing it is achieved using some equation of causal construction of states. Such an equation has been proposed in the definition of Neural Finite-State Transducer (NFST).

NFST represents a new type of finite-state computations. This fact does not depreciate it as a valuable model of computations in the NLP domain. The capability of PDA or TM to perform infinite-state computations cannot be conclusive why they are preferred to FSA because in practice all the models are finite and still FPDA and FTM are preferred to FSA. We can conclude that what is crucial is the type of external memory that is used in computations. In this context PAMA comprises all the classical finite-state models and provides formal basis for massively-parallel computation models, including neural networks.

References

[1] Casey, M. *The Dynamics of Discrete-Time Computations, with Application to Recurrent Neural Networks and Finite State Machine Extraction*. Neural Computation, 8, pp. 1135--1178, 1996.

[2] Hadley, R. *Cognition and the computational power of connectionist networks*, 1999, available at: http://www.cs.sfu.ca/~hadley/online.html

[3] Pozarlik, R. *Connectionist Symbol Processing with Causal Representations*. In Proc. of the 4th Neural Computation and Psychology Workshop: Connectionist Representations, Bullinaria, J. A., Glasspool, G. W. & Houghton, D. (eds.), Springer Verlag: London, pp. 331--342, 1997.

[4] Pozarlik, R. *Integrating Parallel Computations and Sequential Symbol Processing in Neural Networks*. In Proc. of the World Congress on Computational Intelligence, IEEE Press, pp. 1444--1449, 1998.

[5] Pozarlik, R. *What Type of Finite Computations Do Recurrent Neural Networks Perform?* In Proc. of the Inter. Conf. on Artificial Neural Networks, Springer Verlag, 2, pp. 1065--170, 1998.

[6] Pozarlik, R. *Neural Finite-State Transducers: A Bottom-Up Approach to Natural Language Processing*. In Proc. of the Inter. Joint Conf. On Neural Networks, IEEE Press, 10-16 July, Washington, 1999.

[7] Reich, P. A. *The Finiteness of Natural Language*. Language, 45, pp. 831--843, 1969.

[8] Siegelmann, H. T. and Sontag, E. D. *On the Computational Power of Neural Nets*. Journal of Computer and System Sciences, 50, 1, pp. 132--150, 1992.