Request Confirmation Networks in MicroPsi 2

Abstract. To combine neural learning with the sequential detection of hierarchies of sensory features, and to facilitate planning and script execution, we propose Request Confirmation Networks (ReCoNs). ReCoNs are spreading activation networks with units that contain an activation and a state, and are connected by typed directed links that indicate partonomic relations and spatial or temporal succession. By passing activation along the links, ReCoNs can perform both neural computations and controlled script execution. We demonstrate the application of ReCoNs in the context of performing simple arithmetic, based on camera images of mathematical expressions.

Keywords: Request Confirmation network, MicroPsi, ReCoN, neurosymbolic representation

1 Introduction

MicroPsi2 (…) is a cognitive architecture that permits the implementation of situated agents that use neuro-symbolic representations (Hatzilygeroudis and Prentzas 2004) in combination with a motivational system (…). We are using MicroPsi2 to study how to combine conceptual and perceptual representations, and facilitate autonomous learning with full perceptual grounding. To this end, agents require mechanisms for bottom-up/top-down perception, reinforcement learning, motivation, decision making and action execution.

Cognitive architectures with perceptual grounding require a way to combine symbolic and sub-symbolic operations: planning, communication and reasoning usually rely on discrete, symbolic representations, while fine-grained visual and motor interaction require distributed representations.

A common solution is a hybrid architecture combining a neural network layer that deals with perceptual input with a symbolic layer that facilitates deliberation and control using symbolic operations. While such a dual architecture appears to be a straightforward solution from an engineering point of view, we believe that there is a continuum between perceptual and conceptual representations, and that both should use the same set of representational mechanisms. In our view, symbolic/localist representations are best understood as a special case of subsymbolic/distributed representations, for instance where the weights of the connecting links are close to

discrete values. Highly localist features often emerge in neural learning, and rules expressed as discrete valued links can be used to initialize a network for capturing more detailed, distributed features (see, for instance, Towell and Shavlik 1994).

A representational unit in MicroPsi is called a *node* and is made up of a vector of input *slots*, a *node function*, an *activation state*, and a vector of output *gates*. Weighted *links* connect the gates of a node with the slots of other nodes. Slots sum the weighted incoming activation and pass it to the node function, which updates the states of all gates by calling a function for each. The gates in turn are the origin of links to other nodes. *Node types* differ by the number of their gates and slots, and by the functions and parameters of their gates. The type of a link is given by the type of its gate of origin (…).

The most common node type in earlier MicroPsi implementations is called a *concept node*. Concept nodes possess nine gate types (with approximate semantics in parentheses): *gen* (associated), *por* (successor), *ret* (predecessor), *sur* (part-of), *sub* (has-part), *exp* (is-exemplar-of), and *cat* (is-a). Concept nodes can be used to express hierarchical scripts, by linking sequences of events and actions using *por/ret*, and subsuming these sequences into hierarchies using *sub/sur*. Specific *sensor* and *actuator* nodes provide connection to the agent's environment, and native *script* nodes may encapsulate complex functionality to provide backpropagation learning and a variety of other algorithms triggered by activating the respective node.

MicroPsi2 also provides nodes that implement interaction with external sensors, actuators, or that represent more complex neural logic, such as LSTMs (Hochreiter and Schmidhuber 1997), which we combined with denoising autoencoders (Vincent et al. 2008) to learn visual models of the virtual world that our agents inhabit (…).

In MicroPsi, a perceptual representation amounts to a hierarchical script that tests top-down for the presence of the object in the environment. At each level of the hierarchy, the script contains disjunctions and subjunctions of sub-steps, which bottom out in distributed sub-steps and eventually in sensor nodes that reflect measurements in the environment, and actuator nodes that will move the agent or its sensors. Recognizing an object requires the execution of this hierarchical script. In the earlier implementations of MicroPsi, this required a central executive that used a combination of explicit backtracking and propagation of activation. We have replaced this mechanism with a completely distributed mode of execution that only requires the propagation of activation along the links of connected nodes.

2 Request Confirmation Networks

The deliberate top-down initiation of a script, as in the intentional moving of an arm or imagining of an object, has been attributed to activity in the prefrontal cortex (Deiber et al., 1991; Frith, Friston, et al. 1991), an area associated with goal-directed behavioral planning and task management (Koechlin, et al. 1999; Tanji and Hoshi, 2001). To execute a cognitive process or an action, activation flows from its initial stimulation in the prefrontal cortex through the relevant schematic components, continuing either until the objective has been successfully achieved, or until the

sequence is interrupted or fails. ReCoNs offer a possible model for how these schemas and sensorimotor scripts are represented and executed in the cortex.

Request Confirmation Networks (ReCoNs) are auto-executable networks of stateful units that are connected with typed edges. A ReCoN can be defined as a set of units $\mathbb{U}$ and edges $\mathbb{E}$ with

$$\mathbb{U} = \{script\ nodes\ \cup terminal\ nodes\}$$
$$\mathbb{E} = \{por, ret, sub, sur\}$$

A *script node* has a state

$s \in$ {*inactive, requested, active, suppressed, waiting, true, confirmed, failed*}

and an activation $a \in \mathbb{R}^n$, which can be used to store additional state.

A *terminal node* performs a measurement or executes an action, and has a state of {*inactive, active, confirmed*}, and an activation $a \in \mathbb{R}^n$, which represents the value obtained through the measurement, or the return value of the action. A link is defined by $\langle u_1, u_2, type \in \{por, ret, sub, sur\},\ w \in \mathbb{R}^n \rangle$, whereby u_1 and u_2 denote the origin and target unit, *por* links to a successor node, *ret* links to a predecessor node, *sur* links to a parent node, and *sub* links to a child node. w is a link weight with n dimensions that can be used to perform additional computations. Each pair of nodes (u_1, u_2) is either unconnected, or has exactly one pair of links of the types *por/ret*, or *sub/sur*.

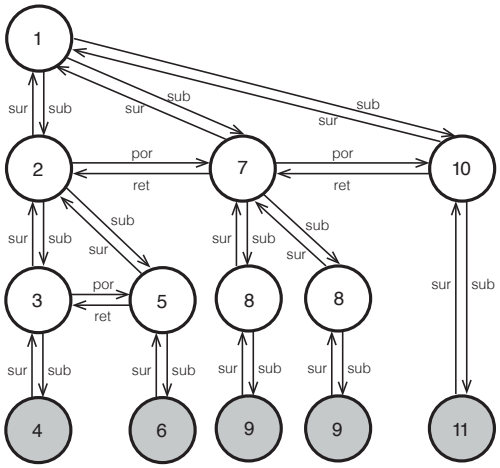

Figure 1: Script execution example

Each *script node* must have at least one link of type *sub* (i.e. at least one child that is either a *script node* or a *terminal node*). *Script nodes* can be the origin and target of links of all types, whereas *terminal nodes* can only be targeted by links of type *sub*, and be the origin of links of type *sur*. Note that not all children of a node need to have successor or predecessor relations. If they do, they will be requested and confirmed in succession. If they do not, then they are interpreted as disjunctions, and execution of the ReCoN happens in parallel.

ReCoNs form a hierarchical script without centralized access to the topology of the network. To achieve this, each individual unit implements a state machine that transitions in response to messages from directly adjacent units.

Initially, all units are in the state *inactive*. If the state of one of its nodes is set to *requested*, this triggers the evaluation of the portion of the script connected via this node's *sub*-link. The evaluation is propagated by successively and recursively requesting the children of the originally requested node. Whenever the request reaches a terminal node, confirmation or failure of the evaluation is determined and propagated back to the requesting unit. Figure 1 illustrates the order of execution of a hierarchical script containing sequences (2, 7, 10; 3, 5) and alternatives (8, 8). The script is started by sending a continuous request signal to its root node (1). Sequences are executed successively, while alternatives are executed concurrently. A failure of a step in a sequence (i.e. in one of the actions 4, 6, 11) or of all alternatives (9, 9) will result in the failure of the whole script. At any time, the script execution can be aborted by ending the request signal to its root node.

The functionality of ReCoN nodes can best be understood by using an explicit state machine with message passing. In each step, the nodes evaluate the messages they receive from their neighbors based on their current state, and change their state accordingly. The required messages are *request* (*r*), *inhibit request* (*ir*), *inhibit confirm* (*ic*), *wait* (*w*) and *confirm* (*c*): *request* will attempt to activate a child node, *inhibit request* prevents a node to become active before its predecessor has successfully finished execution, *confirm* informs a parent node that its child has successfully executed, *inhibit confirm* prevents a node to send a *confirm* message before its successor has executed successfully, and *wait* informs a parent node that it has child nodes that are still active. If a parent node receives neither a *wait* nor a *confirm* message, the execution of its child nodes is assumed to have failed.

Unit state	***por***	***ret***	***sub***	***sur***
⌀	–	–	–	–
R	*ir*	*ic*	–	*w*
A	*ir*	*ic*	*r*	*w*
S	*ir*	*ic*	–	–
W	*ir*	*ic*	*r*	*w*
T	–	*ic*	–	*c*
C	–	*ic*	–	*c*
F	*ir*	*ic*	–	–

Table 1: message passed along each gate, based on node state

The corresponding states are *inactive* (⌀): the node has not been requested; *requested* (R): the node has received a request; *active* (A): the requested node is sending a request to its children; suppressed (S): the requested node cannot yet send a request to its children; *waiting* (W): the requested node continues to request to its children and

waits for their response; *true* (T): the requested node has received a confirmation from its children; *confirmed* (C): the requested node is sending a confirmation message to its parent; *failed* (F): the requested node is no longer receiving a wait message from its children.

Table 1 details which messages are being sent in which state, and figure 2 illustrates how the state machine in every node transitions in response to the messages.

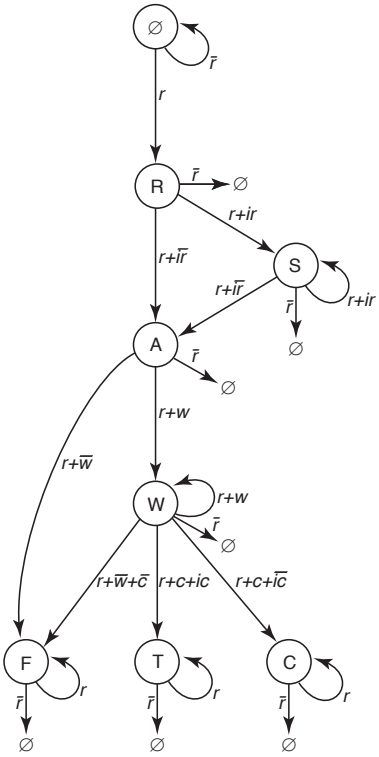

Figure 2: State transitions (for each node) in response to messages

3 Implementation and experiment

Request confirmation networks can be implemented in various ways, such as individual artificial neurons (…), or by using suitably initialized LSTMs. For the purposes of this project, we have implemented them in MicroPsi2 as a specific node type with five gates and slots (*gen, por, ret, sub, sur*) and a single real valued activation α to store the node state: *failure* corresponds to a $\alpha < 0$, *inactive* to $0 \leq \alpha < 0.01$, *requested* to $0.01 \leq \alpha < 0.3$, *suppressed* to $0.01 \leq \alpha < 0.3$, *active* to $0.3 \leq \alpha < 0.5$, *waiting* to $0.5 \leq \alpha < 0.7$, *true* to $0.7 \leq \alpha < 1$ and *confirmed* to $\alpha \geq 1$.

The ReCoN can be used to execute a script with discrete activations, but it can also perform additional operations along the way. This may done by calculating additional activation values during the request and confirmation steps.

During the confirmation step (a node turns into the state *true* or *confirmed*), the activation of that node may be calculated based on the activations of its children, and the weights of the *sur* links from these children. During the *waiting* step, children may receive parameters from their parents which are calculated using the parent activation and the weights of the *sub* links from their parents. This mechanism can be used to adapt ReCoNs to a variety of associative classification and learning tasks. In a previous experiment, we combined a ReCoN with autoencoders for learning a perceptual task in a virtual environment (…).

Here, we demonstrate the use of a ReCoN in conjunction with a neural network to extract handwritten arithmetic expressions from a scanned image, and use the *terminal nodes* of the ReCoN to perform the corresponding arithmetic operations by connecting them directly to a stack machine. The execution consists of three phases:

1. A camera image containing an arithmetic expression is parsed into separate images of its digits and mathematical operators, then individually fed into a pre-trained multilayer perceptron.

2. The array of predicted symbols is used to construct a ReCoN that represents the arithmetic expression in its topography.

3. The ReCoN is requested, performs the calculation using a stack machine, and the result is obtained.

Implementing a multi layer perceptron classifier in MicroPsi

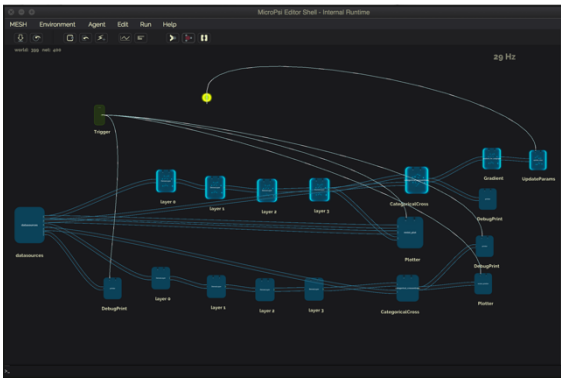

Figure 3: MLP classifier in MicroPsi's MESH editor

For the initial image recognition task, the input image is converted into a black and white image and segmented into individual symbols using the Python image processing library scikit-image. We implemented the multilayer perceptron (MLP) using an input layer with 784 nodes, 14 output nodes (for the ten digits and the arithmetic operators $+, -, \times$, and $\div$), and two hidden layers with 240 and 60 nodes, respectively. We chose linear rectifiers (ReLu) as activation functions and a

softmax classifier to pick the symbol receiving the highest activation in the output layer. The MLP was trained using MNIST for the digits and a Kaggle dataset for the operators (Nano 2016).

Generating the Request Confirmation Network

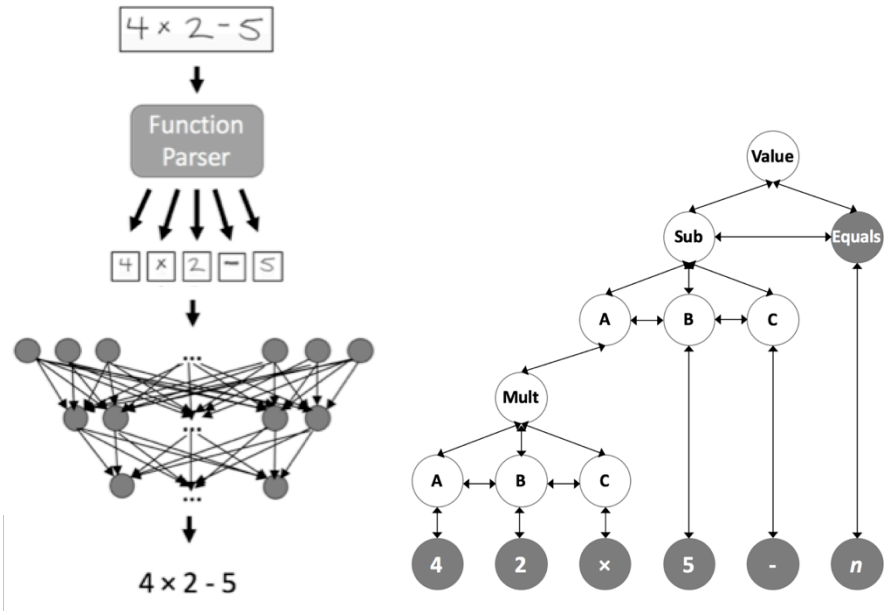

Figure 4a: function parsing and classification; 4b: constructed ReCoN

After the image segmentation and recognition stages, predicted symbols are combined into an arithmetic expression (figure 4a), and the corresponding ReCoN is generated (figure 4b). Each operation is mapped to a corresponding arrangement of nodes: multiplication is translated into the step "Mult", which consists of a *sub/sur* linked three step sequence "A" *por/ret* "B" *por/ret* "C". Each of these steps is *sub/sur* linked to its computational realization. Here, each symbol is translated into a *terminal node* that performs an operation on a stacked (Reverse Polish) calculator:

- If the symbol is a digit (0..9), pull the previous element from the stack. If the element is a number, multiply it by ten and add the new digit. Otherwise, push the previous element back to the stack, and push the new digit on the stack as well.
- If the symbol is an arithmetic operator, pull the last two elements from the stack, perform the operation, and push the result to the stack.
- If the symbol is "equals", pull the last element from the stack and print it.

Executing the Request Confirmation Network

After the setup phase, the ReCoN is executed by sending a request message to its root node. The network will spread activation through its nodes until the terminal nodes are reached, and perform the stack calculations which are implemented as node functions of the respective terminal nodes.

The successful execution of one of the elementary stack operations will result in a confirm message to its parent node, which will remove the suppression signal from its successor, which will in turn pass a request to the next stack operation, until the script is fully confirmed (figure 5). Conversely, the failure of one of the stack operations (as a result of an invalid sequence of input characters) will lead to a failure of the entire script.

Figure 5: ReCoN activation spreading in the MicroPsi MESH editor

4 Conclusion and future work

This contribution presents an early stage of Request Confirmation Networks, which are a paradigm that strives to combine the straightforward execution of symbolic scripts (especially for perceptual and motor hierarchies and planning) with

distributed representations. ReCoN nodes are state machines that can implement sequences, conjunctions, disjunctions and conditional loops without reliance on a central executive, solely by passing messages to their immediate neighbors.

The implementation discussed here shows the application of ReCoNs for a demonstrator task that combines a neural network classifier for visual input with executable hierarchical scripts and the control of a stack machine for performing arithmetic operations. While this may serve as an illustration of the basic concept, it is far from being an exhaustive treatment. Concepts not discussed here include learning strategies (which involve states and messages for the distribution of rewards), the self-assembly of ReCoNs depending on a previously encountered task context (which introduces states and messages for anticipated rewards), the use of individual sub-graphs in multiple positions of the script (which introduces semaphore states) and the translation and interoperation with existing network architectures with ReCoNs. These areas constitute our ongoing work with ReCoNs.

Acknowledgements: …

References

Deiber, M. P., Passingham, R. E., Colebatch, J. G., Friston, K. J., Nixon, P. D., Frackowiak, R. S. J. (1991): Cortical areas and the selection of movement: a study with positron emission tomography. *Experimental brain research*, *84*(2), 393-402

Frith, C. D., Friston, K. J., Liddle, P. F., Frackowiak, R. S. (1991): Willed action and the prefrontal cortex in man: a study with PET. In *Proc. R. Soc. Lond. B* (Vol. 244, No. 1311, pp. 241-246). The Royal Society

Gallagher, K. (2018). Request Confirmation Networks: A cortically inspired approach to neuro-symbolic script execution. MA Thesis, Harvard University May 2018

Hatzilygeroudis, I., Prentzas, J. (2004): Neuro-symbolic approaches for knowledge representation in expert systems. *International Journal of Hybrid Intelligent Systems*, 1(3-4):111–126, 2004.

Hochreiter S., Schmidhuber, J. (1997): Long short-term memory. Neural Computation. 9 (8): 1735–1780

Koechlin, E., Basso, G., Pietrini, P., Panzer, S., Grafman, J. (1999). The role of the anterior prefrontal cortex in human cognition. *Nature*, *399*(6732), 148

LeCun, Y. (1998): The MNIST database of handwritten digits [Data file]. Available from http://yann. lecun. com/exdb/mnist/

Nano, X. (2016): Handwritten math symbols dataset [Data file]. Available from Kaggle: https://www.kaggle.com/xainano/handwrittenmathsymbols

Tanji, J., Hoshi, E. (2001): Behavioral planning in the prefrontal cortex. *Current opinion in neurobiology*, *11*(2), 164-170

Towell, G. Shavlik, J. (1994): Knowledge-based artificial neural networks. Artificial Intelligence, 70 (p. 119-165)

Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.-A. (2008): Extractingandcomposing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103