Request confirmation networks for neuro-symbolic script execution

Joscha Bach[*]
Program for Evolutionary Dynamics
Harvard University
Cambridge, MA 02138
joscha@mit.edu

Priska Herger
micropsi industries GmbH
Berlin, Germany
priska@micropsi-industries.com

Abstract

We propose Request Confirmation Networks (ReCoNs) to combine neural learning with the sequential, hierarchical detection of sensory features, and to facilitate planning and script execution. ReCoNs are spreading activation networks with units that contain an activation and a state, and are connected with 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 an application of ReCoNs in the cognitive architecture MicroPsi2 for an active perception task in a virtual environment.

1 Introduction

MicroPsi is a cognitive architecture that combines neuro-symbolic representations [1] with situated perception and a motivational system [2], [3]. MicroPsi agents live in an open environment that they have to explore and navigate; currently we are using the game environment Minecraft [4]. To this end, agents require mechanisms for bottom-up/top-down perception, reinforcement learning, motivation, decision making and action execution. We are using MicroPsi to study how to combine perceptual and conceptual representations, and to facilitate autonomous learning with full perceptual grounding.

Cognitive architectures with perceptual grounding require a way to combine symbolic and subsymbolic operations: planning, communication and reasoning rely on discrete, symbolic representations, while fine-grained visual and motor interaction requires distributed representations. The standard solution consists in a hybrid architecture that combines a neural network layer with a carefully crafted model that learns to compress the perceptual input, and a layer that facilitates deliberation and control using symbolic operations and uses the extracted regularities [5]. However, 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, i.e. that both should use the same set of representational mechanisms, and primarily differ in the operations that are performed upon them. In our view, symbolic/localist representations are best understood as a special case of subsymbolic/distributed representations, for instance ones 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 KBANN [6]).

MicroPsi's representations combine neural network principles and symbolic operations via recurrent spreading activation networks with directed links. The individual nodes in the network process

[*] http://micropsi.com

information by summing up the activation that enters the nodes through slots, and calculating a function for each of their gates, which are the origin of links to other nodes. Node types differ by the number of gates and slots they have and by the functions and parameters of their gates. Typed links can be expressed by the type of their gate of origin.

The most common node type in earlier MicroPsi implementations is called a concept node. Concept nodes possess seven gate types (with approximate semantics in parentheses): *gen* (associated), *por* (successor), *ret* (predecessor), *sur* (part-of), *sub* (has-part), *exp* (is-exemplar-of), *cat* (is-a). Concept nodes can be used to express hierarchical scripts [7] by linking sequences of events and actions using *por*/*ret*, and subsuming these sequences into hierarchies using *sub*/*sur*.

In MicroPsi2, a perceptual representation amounts to a hierarchical script to test for the presence of the object in the environment. At each level of the hierarchy, the script contains disjunctions and subjunctions of substeps, which bottom out in distributed substeps 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 past, this required a central executive that used a combination of explicit backtracking and activation propagation. We have replaced this mechanism with a completely distributed mode of execution called request confirmation network that only requires the propagation of activation along the links of connected nodes.

2 Request confirmation networks

Modeling perception in a cognitive architecture requires an implementation of top-down/bottom-up processing, in which sensors delivers cues (bottom-up) to activate higher-level features, while perceptual hypotheses produce predictions of features that have to be verified by active sensing (top-down). When using a neural network paradigm, the combination of bottom-up and top-down processing requires recurrency. Request confirmation networks are a solution to combine constrained recurrency with the execution of action sequences. They provide a neural solution for the implementation of sensorimotor scripts.

Request Confirmation Networks (ReCoN) are implemented within MicroPsi2's formalism but are in fact a more general paradigm for auto-executable networks of stateful nodes with typed links. We therefore provide a general definition that is independent of our implementation.

A request confirmation network can be defined as a set of units $\mathbb{U}$ and edges $\mathbb{E}$ with

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

A script node has a state s where

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

and an activation $a \in \mathbb{R}^n$, which can be used to store additional state. (In MicroPsi2, the activation is used to store the results of combining feature activations along the *sub* links). A terminal node performs a measurement and has a state $s \in \{\text{inactive, active, confirmed}\}$, and an activation $a \in \mathbb{R}^n$, which represents the value obtained from the measurement. An edge is defined by $\langle u_s, u_t, type \in \{\text{por, ret, sub, sur}\}, w \in \mathbb{R}^n \rangle$, whereby u_s and u_t denote the source and target unit, *por* connects to a successor node, *ret* to a predecessor node, *sub* to a parent node, and *sub* connects 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_s, u_t) is either unconnected or has exactly one pair of links of the types *por*/*ret*, or *sub*/*sur*. 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 source 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*.

ReCoNs solve the problem of constraining the flow of control in a hierarchical script on the level of individual units without using topological information, i.e. without a centralized interpreter or informing individual units about their neighboring elements. To do this, they need to store the execution state within the script elements in a distributed manner; each of these units is a state machine with one of eight states (as defined above). Each unit implements connections to its neighbors using

directional links. Initially, all units are inactive. The execution of a script starts with sending the signal 'request' to its root node. Semantically, the root node represents a hypothesis that is spelled out in the script, and that we want to test; we request the validation of the root node. After the script is executed, it will either be in the state "confirmed" or "failed" (until we turn off the 'request' signal, and the unit becomes inactive again).

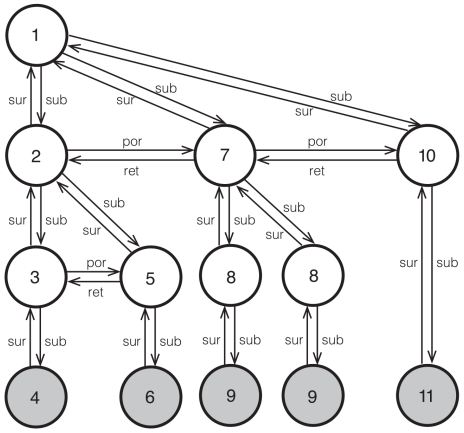


Figure 1: Example of script execution.

During the validation, it will need to validate all of its children, by sending a 'request' signal along its *sub* links. These may either form sequences or alternatives, cf. Figure 1. The former are validated in succession, the latter in parallel. Successive execution requires that units prevent their successors from becoming "active" until it is their turn; they do this by sending an 'inhibit request' signal along their *por* links. If a requested unit receives an 'inhibit request' signal, it becomes "suppressed" until its predecessor becomes "true" and turns off the signal. Active elements tell their parent that it should wait for their validation to finish by sending them a 'wait' message. Each active element will remain in the "waiting" state until either one of its children sends a 'confirm' message (in which case it turns into the state "true", or no more children ask it to wait (in which case the element will turn into the state "failed"). In sequences, we also need to ensure that only the last element of a sequence can confirm the parent request, so each unit sends an 'inhibit confirm' signal via *ret* to its predecessors. The last unit in a sequence will not receive an 'inhibit confirm' signal, and can turn into the state "confirmed" to send the 'confirm' signal to the parent. The execution of the script can be reset or interrupted at any time, by removing the 'request' from its root node.

The ReCoN can be used to execute a script with discrete activations, but it can also perform additional operations along the way. This is done by calculating additional activation values during the request and confirmation steps. During the confirmation step (a node turns into the state "confirmed" or "true"), the activation of that node is calculated based on the activations of its children, and the weights of the *sur* links from these children. During the requesting 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.

2.1 A message passing definition of a ReCoN unit

The units of a ReCoN implement a finite state machine – as drawn out in Figure 2. This can be realized by defining the eight discrete states explicitly and using a set of messages that are distributed along the *por*, *ret*, *sub* and *sur* links, depending on the current state of each unit. These messages can be defined as {inhibit_request, inhibit_confirm, wait, confirm, fail}. They are passed as specified in Table 1.

In each state, incoming messages are evaluated and nodes either stay in the current state or switch to the next, if a condition is fulfilled. These conditions are given in Figure 2.

Unit state	POR	RET	SUB	SUR
inactive (ø)	–	–	–	–
requested (R)	inhibit request	inhibit confirm	–	wait
active (A)	inhibit request	inhibit confirm	request	wait
suppressed (S)	inhibit request	inhibit confirm	–	–
waiting (W)	inhibit request	inhibit confirm	request	wait
true (T)	–	inhibit confirm	–	–
confirmed (C)	–	inhibit confirm	–	confirm
failed (F)	inhibit request	inhibit confirm	–	–

Table 1: Message passing in ReCoNs.

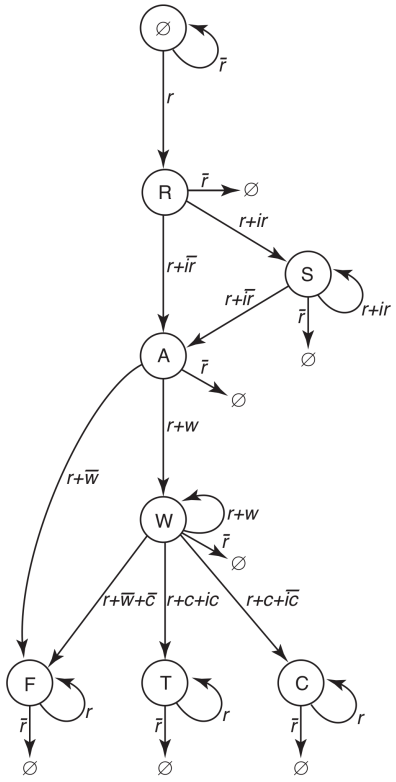


Figure 2: State transitions.

2.2 A neural definition of a ReCoN unit

It is possible to realize a ReCoN unit with an ensemble of artificial neurons. Figure 3 shows an arrangement of ten simple threshold elements with only excitatory and inhibitory links that satisfies these conditions. Let the activation of a neuron be defined as

$$\alpha_j = \begin{cases} \sum(w_{ij} \cdot \alpha_i) & \text{if } w_{ij} \cdot \alpha_i \geq 0 \text{ for all } w_{ij}, \alpha_i \\ 0 & \text{otherwise} \end{cases}$$

i.e. any incoming activation on a link with a negative weight is sufficient to entirely inhibit the neuron. A message is any activation value sent over a link that crosses the boundary between units. Units may send the message 'request' to their children (top-down), 'wait' and 'confirm' to their parents (bottom-up), 'inhibit request' to their successors, and 'inhibit confirm' to their predecessors (lateral inhibition).

We want all activity in a unit to cease as soon as the unit is no longer requested. Thus, all activation in the unit is derived from the activation of the incoming request message. The request activation is directly sent to the neurons IC, IR, and W. IC will inhibit confirm signals by predecessor units, and IR will inhibit requests to child nodes by successor units, before the current unit is confirmed. W informs the parent node that it has active (non-failed) children, by sending the 'wait' message. IR also prevents the unit from confirming prematurely; i.e. before it has received a 'confirm' message by one of its children.

IC then passes on its activation to R, and R sends out a request to the unit's children, unless it is inhibited by the predecessor node. To give predecessor nodes enough time to send the inhibition signal, the flow of activation between IC and R is delayed by DR. F becomes active as soon as the unit has no more active children, and represents the failure state of the unit. Like all states, it must receive its activation from the initial request. It cannot be allowed to fail before the unit has sent a request to its children, so F is inhibited by the predecessors 'inhibit request' message at the neuron R′. F can also not fail before the children cease sending their 'wait' message, so it is inhibited by that message. F must also allow enough time for requested children to respond with this message, so its activation is delayed by the helper neuron DF. Once F becomes active (i.e. the unit fails), it stops the 'request' to the children by inhibiting R, and the 'wait' to the parent, by inhibiting W. (F cannot get its activation directly from R, because F will later inhibit R and would thus remove its own activation; therefore it is activated through the helper neuron R′).

The neuron T represents that the unit has been confirmed (true). It receives its activation from the original request, but is not allowed to become active through the inhibition by IC. Only if IC is turned off by the 'confirm' message of one of its children, T becomes active and will turn off W (the 'wait' message to the parent), R (the request of the children) and IR (the inhibition of the successors, and itself). If the unit has no successors (i.e. receives no 'inhibit confirm' message), it will signal T's value via C as a 'confirm' message to the unit's parent. This is just one of many possible ways in which a ReCoN unit could be realized with artificial neurons.

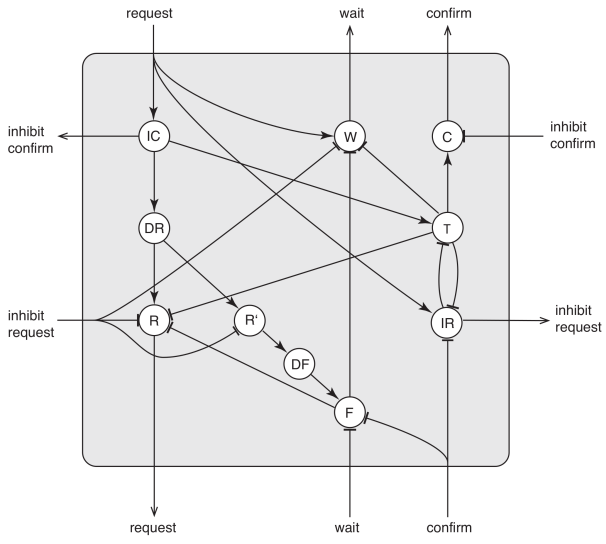


Figure 3: Schematic of a neural ReCoN specification.

In MicroPsi2, the functionality of this circuitry is embedded in a single node, which requires a only one calculation step and reduces memory usage; cf. section 3.1.

3 Implementation

3.1 A compact implementation of ReCoNs

Rather than explicitly modeling the state machine of a ReCoN unit, this implementation makes use of the fact that the state of a ReCoN unit is fully determined by the activation of connected nodes in the previous time step. States are implemented as a set of simple arithmetic rules operating on incoming activation from connected nodes and the node itself. Nodes can link to themselves using "*gen* loops" to store the states "true", "confirmed", and "failed". Activation as well as all link weights are numbers enabling real-value information processing and neural learning.

In this implementation, *por*/*ret* activation $\in \{-1, 0, 1\}$ is used to control the flow of *sub* requests ($\{0, 1\}$) and *sur* confirmation. The real-valued *sur* activations can be interpreted as probabilistic in the range $[0, 1]$ or take on the value -1 to indicate a solid fail. They control *por*/*ret* activation in the next higher layer.

Activation is spread in discrete steps. Each step consists of two phases: Propagation and calculation. Propagation is simply: $\mathbf{z} = \mathbf{W} \cdot \mathbf{a}$ where $\mathbf{W}$ is a matrix of link weights and $\mathbf{a}$ is the activation vector. Calculation is $f_{gate}(f_{node}(\mathbf{z}))$, where f_{gate} is an activation function specified per link type and f_{node} implements the ReCoN behavior. The following definitions describe the behavior of the node functions f_{node} per link type.

$$f^{gen}_{node} = \begin{cases} z^{sur} & \text{if } (z^{gen} \cdot z^{sub} = 0) \vee (\exists \,\text{link}^{por} \wedge z^{por} = 0) \\ z^{gen} \cdot z^{sub} & \text{otherwise} \end{cases}$$

$$f^{por}_{node} = \begin{cases} 0 & \text{if } z^{sub} \leq 0 \vee (\exists \,\text{link}^{por} \wedge z^{por} \leq 0) \\ z^{sur} + z^{gen} & \text{otherwise} \end{cases}$$

$$f^{ret}_{node} = \begin{cases} 1 & \text{if } z^{por} < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$f^{sub}_{node} = \begin{cases} 0 & \text{if } z^{gen} \neq 0 \vee (\exists \,\text{link}^{por} \wedge z^{por} \leq 0) \\ z^{sub} & \text{otherwise} \end{cases}$$

$$f^{sur}_{node} = \begin{cases} 0 & \text{if } z^{sub} \leq 0 \vee (\exists \,\text{link}^{por} \wedge z^{por} \leq 0) \\ (z^{sur} + z^{gen}) \cdot z^{ret} & \text{if } \exists \,\text{link}^{ret} \\ z^{sur} + z^{gen} & \text{otherwise} \end{cases}$$

An undesirable property of this implementation is that new activation is not solely based on incoming activation but also depends on local knowledge about the existence of *por*/*ret* links[1].

3.2 Using ReCoNs for an active perception task

Active perception with ReCoNs is based on learned representations that are encoded in the structure and weights of links. Whenever the system encounters a new situation (for instance, after a locomotion action) it will form a model of its environment as a combination of verifiable hypotheses. At higher levels, these hypotheses can contain object representations and geometric relations between objects, at the lower levels features and eventually input patches that make up the visual structure of these objects. In the basic variant implemented so far, hypotheses are based on a single autoencoder that captures features of the scene as a whole.

Hypotheses are verified by activating their top-most node with a request (i.e. sending activation to its *sub* slot). From there, activation spreads downwards through the sub-hypotheses defined by the node's children, which are verified sequentially. If all existing hypotheses fail, the agent constructs a new one, scanning its visual field and connecting the localized feature nodes into a hierarchy. We

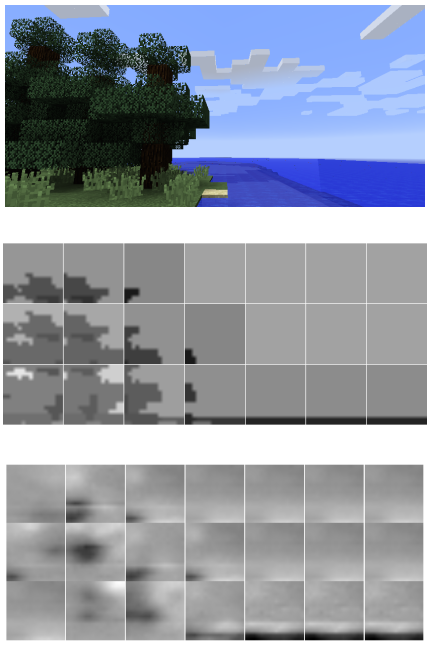
[1] The source code is available from https://github.com/joschabach/micropsi2. Formulas found in the code are slightly more complex as they include additional features like time-based failing and searchability which are not relevant in this context.

built a MicroPsi agent that moves about in a Minecraft world along a transition graph, samples its visual environment and learns to recognize scenes on the graph with a request confirmation network.

The agent does not process the entire visual field at once, but uses a rectangular fovea with a 16×16 sensor matrix. Using actuators, it can move this fovea to fixate a point (and thus a 16×16 patch) in its field of vision. The possible fixation points are constrained to allow for a 50 percent overlap between neighboring patches.

These patches are learned using a denoising autoencoder [8] with zero-masking Bernoulli noise and a corruption level of 0.3. A 2-dimensional perspective projection of the Minecraft block world at a given position serves as input to the autoencoder. Figure 4b shows, for one such position, the visual field as the standard Minecraft client would present it to a human player (top), the projection provided as visual input to the agent (center), and a visualization of the features at that position as learned by the autoencoder (bottom).

(a) Transition graph of the agent.

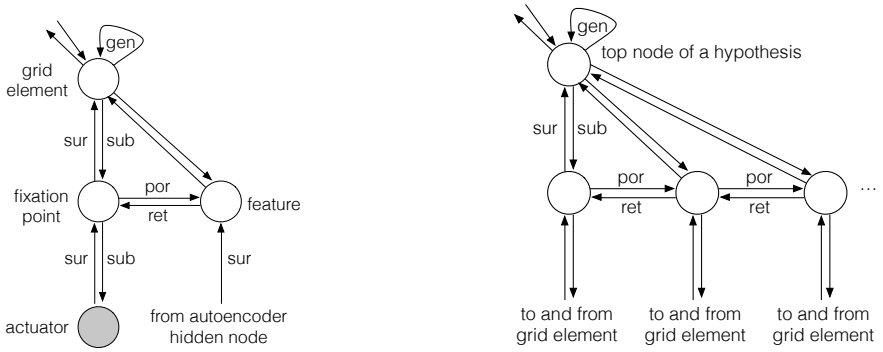


(b) Training data – Minecraft vanilla client (top), visual input (center), learnt features (bottom).

Figure 4: Action space and input sample of a MicroPsi2 agent in Minecraft

The autoencoder hidden nodes play the role of terminal nodes to the ReCoN units. Features detected by the hidden nodes are fed into a localized feature grid, where each combination of feature and fixation point is expressed as a pair of *por*/*ret* connected ReCoN units. Each pair is *sur*/*sub* connected to a parent unit. The first node of the *por*/*ret* sequence is *sub*/*sur* connected to the actuator node that was active when the feature was detected, the second has an incoming *sur* link from one of the hidden nodes of the autoencoder (see Figure 5a). The grid as a whole associates expected features with all possible fixation points. Each grid element represents a feature at a given location in the agent's visual field, together with the action required to reach that location. Since hypotheses are constructed as sequences of grid nodes, which contain actuators for moving the fovea, they are simple sensorimotor scripts. New hypotheses are formed based on those features that are deemed relevant to the current situation. Here, relevance is simply defined by an activation value exceeding a threshold of 0.8. Figure 5b shows the structure and connectivity of such a hypothesis.

To evaluate the functionality of this implementation, we let the agent move around in the Minecraft world using a number of fixed locations. As expected the agent learned hypotheses for all locations, subsequently stopped forming new hypotheses, and successfully recognized all the locations.



(a) An element of the feature grid.

(b) Structure and connectivity of a hypothesis.

Figure 5: Connectivity schemas used for scene recognition.

4 Summary and current work

Request confirmation networks (ReCoNs) provide an elegant way to integrate hierarchical scripts into neuro-symbolic architectures. They combine top-down/bottom-up processing with sequences to form sensorimotor scripts. We gave a definition of a ReCoN unit as a message passing state machine and a second, neural definition using simple threshold elements. The implementation in our cognitive architecture MicroPsi2 uses dedicated nodes as ReCoN units. We tested our model using an autoencoder over visual input as input layer, and let our agent learn and recognize scenes in the game Minecraft. The agent also uses these representations for symbolic operations such as protocol formation and basic policy learning. ReCoNs do not imply a novel way of learning, but a representational formalism – which also limits the ways in which we can quantitatively evaluate them. We are currently working towards combining ReCoN units with long short-term memory [9] to learn the temporal/causal dynamics in Minecraft, for instance, to predict which hypothesis is the most likely candidate at any given time. We are also working on making hypotheses more robust against different types of variance: alternative views of a scene or object, presence /absence of features, and forming hierarchies of hypotheses.

Acknowledgments

We would like to thank Dominik Welland and Ronnie Vuine (micropsi industries) who are vitally involved in our recent and ongoing research and development of request confirmation networks.

References

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

[2] Joscha Bach. *Principles of synthetic intelligence PSI: an architecture of motivated cognition*, volume 4. Oxford University Press, 2009.

[3] Joscha Bach. Modeling motivation in micropsi 2. In *Artificial General Intelligence*, pages 3–13. Springer, 2015.

[4] Daniel Short. Teaching scientific concepts using a virtual world – minecraft. *Teaching Science - Journal of the Australian Science Teachers Association*, 58(3):55, 2012.

[5] Stefan Wermter and Ron Sun, editors. *Hybrid neural systems*. Springer Science & Business Media, 2000.

[6] Geoffrey G Towell and Jude W Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1):119–165, 1994.

[7] Roger C Schank and Robert P Abelson. *Scripts, plans, and knowledge*. Yale University, 1975.

[8] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[11] Ronald J Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.