A REINFORCEMENT MODEL OF SEQUENTIAL ROUTINE ACTION

Nicolas Ruh[1], *Richard P. Cooper*[2] *and Denis Mareschal*[3]

School of Psychology, Birkbeck, University of London
Malet Street, London, WC1E 7HX, UK.
[1] n.ruh@psychology.bbk.ac.uk
[2] r.cooper@bbk.ac.uk
[3] d.mareschal@bbk.ac.uk

ABSTRACT

In a recent paper, Botvinick & Plaut [1] claim to capture the relevant empirical data on a sequential routine action task in a single embedded SRN (simple recurrent network). In this type of model, task representations are most appropriately described as an emergent property, that is, in terms of the continuous attractor dynamics in the network's hidden layer. Two major shortcomings of this model are identified: (a) the implausibility of the learning regime and (b) its inability to account for the goal directedness in human behavior.

In this paper we suggest that reinforcement learning can overcome both of these problems. We present a first attempt to implement hierarchical routine action in a distributed reinforcement model. Employing two connectionist networks in an actor/critic architecture with TD(temporal difference)-learning it is shown that the model learns to negotiate the high-dimensional state space of a simplified routine action task. This is possible because of the delicate interplay of several principles, each of which simplifies the learning of the task at hand in its own fashion. The implications of the emergent representations in this type of model are discussed.

1. INTRODUCTION

How is the knowledge of hierarchical sequences represented in the brain? This question has reemerged recently in a debate concerning the computational mechanisms that govern routine behavior. Given a broad enough definition, most researchers might acknowledge that some kind of schema is at the core of the brain's ability to deal flexibly with sequences of actions [2, 3]. At this point, however, the general agreement ends. Concepts vary from conceiving of schemas as discrete, hierarchically organized and explicitly represented knowledge structures [4] to regarding them as an emergent property of a system, inseparably tied to its domain and the way it has learned [5]. The latter view is advocated in a simulation of routine action selection by Botvinick & Plaut [1, 6]. They present an embedded SRN-model that learns to make a cup of tea (2 versions) or coffee (4 versions) by performing the correct sequence of actions. It is claimed that the model produces the kind of errors that can be observed in human slips of action when injected with a moderate amount of noise. At higher levels of noise, the model's behavior resembles the performance of neurological patients with ADS (action disorder syndrome). The authors conclude that a single connectionist system is sufficient to capture the empirical data and that the underlying representations can most appropriately be conceptualized in terms of the distribution of activation in the network's hidden layer or, since it is a recurrent network dealing with sequences, the trajectory in the activation space over time. It is emphasized that the so conceptualized task representations "overlap structurally, sharing graded, multidimensional similarity relations" [6]. This overcomes the problem of more traditional approaches that require one discrete representation/schema for each version of the task.

Closer inspection, however, reveals that not only is the supervised learning regime implausible, it is furthermore heavily dependent on the exact composition of the training set. Only an equal distribution of choices at every branching point enables the network to perform each version of a task independently. This is because the distributed representations of the SRN function as continuous attractors in the state space of the hidden layer. In fact, the net develops one independent attractor for each task version. In the absence of any other non-deterministic influences, the choice of which attractor's basin to enter is taken by the random initialization of the context layer. A replication of the model shows that less frequent task versions that share the first few steps with the dominant one are not accessible by this mechanism [7]. One solution to this problem is to associate so-called "instruction units" with the first input in order to force the initial state into the influence of a certain attractor's basin (see [1], Simulation 2). If this logic is followed to the end, it leads to a discrete representation (instruction unit) for every version of the task and thus to the very kind of discrete and independent representation that the connectionist approach attempts to challenge.

Both of the above-mentioned weaknesses could be overcome by reinforcement learning. While the distributed representations developed in the actor network would share many of the appealing features of the simple SRN approach, a reinforcement model would acquire them in a more plausible way by actively exploring it's options in a given environment and evaluating it's performance by estimating future reward. This implies as well that the model discovers different possible ways to reach the rewarded state. Because the model essentially tries to approch the rewarded state(s) in the most efficient way, it can be seen as implementing goal directed behavior. Hence, the representations in the reinforcement model resemble the ones in a supervised SRN in terms of their overlapping distributed nature, but they develop by a different mechanism. We consider the implications of this in the discussion.

In the remainder of this paper, we present a first attempt to implement sequential routine action in a connectionist reinforcement model. This task is not trivial since (a) there is no proof of convergence when using distributed neural networks to approximate policy and value function [8], and (b) to our knowledge it is the first time an SRN has been used as an actor in this kind of architecture.

2. METHOD

2.1. Task

The Nutella task is a simplified version of the Coffee task employed by earlier computational accounts of routine action [1, 4, 6]. Two task sequences must be learned:

<u>Nutellatoast:</u> fixate knife – pick knife – fixate nutella – pick nutella – fixate toast – use knife – pick nutella-toast – eat nutellatoast (8 steps)

<u>Butternutellatoast:</u> fixate knife – pick knife – fixate butter – pick butter – fixate toast – use knife - fixate nutella – pick nutella – fixate buttertoast – use knife – pick butternutellatoast – eat butternutellatoast (12 steps)

2.2. Architecture

An actor/critic architecture was employed with both components implemented as neural networks.

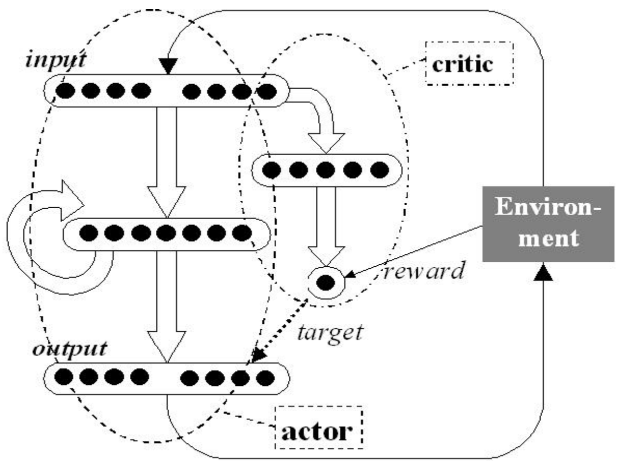

Figure 1; Architecture of the reinforcement model.

The actor was implemented as an SRN with eight units in the input and the output layer, seven units in the hidden and context layer.

<u>Input; 8 units, representing perceived objects:</u>

[toast – butter - Nutella – knife] as fixated (units 1-4) or held (units 5-8).

<u>Output; 8 units, representing the actions:</u>

pick – put – use – eat – fixate toast – fixate butter – fixate Nutella – fixate knife

The critic was implemented as a multi-layer feed forward network with the same 8 input units as the actor, 5 hidden units and one output unit representing the value of the perceived state. A sigmoidal activation function was employed in all layers of both networks.

The actor net was embedded in an environment that maps the chosen actions to the perceived changes in the environment, that is, the new input. If the actor activated the "fixate toast" unit, for example, the unit representing fixation on toast would be turned on in the next input. If an ingredient has already been added to the toast, subsequent fixating on toast would lead to perceiving the toast and the ingredient.

The environment, which is defined as everything outside the actual model, also supplied the reward signal. The reward thus may be interpreted as provided by some other part of the cognitive system and not directly by the outside world.

2.3. Learning algorithm

The critic learned to predict the value of the state determined by the chosen action, that is, it approached either the reward it received in the terminal case, or its own prediction at step t+1 (= temporal difference (TD-) error). By this mechanism, the anticipation of a reward at the end of a sequence is propagated backwards in time. The difference between the prediction and the actual next value is used as an error signal in order to change the weights of the critic via backpropagation.

The actor, on the other hand, learned to adjust the activation of the unit that represents the action chosen towards the value predicted by the critic. Only the weights that contributed to the activation of this output unit are changed by calculating the difference and propagating this error back through the net.

Both nets learn at the same time and online, the actor thus "chasing a moving target".

2.4. Learning regime

Both nets were initialised with small random weights (± 0.5) and none of the input units active. Activation was then propagated through both nets. A random number (uniform distribution) between 0 and 0.5 (T) was added to the activation of each of the actor's output units so as to implement a certain amount of randomness in the network's behavior. This enables the model to explore the state space. The unit with the highest activation was chosen as the action executed, the new input was obtained via the environmental loop and the prediction of the critic calculated. At this point, the TD-error can be calculated and used to adjust the weights of both nets, as described above. The next iteration was then started.

The model starts out with random behavior, obtaining reinforcement only when it produces a valid (sub) sequence by chance. As the expectation of reward is propagated backwards in time and the actor learns to choose promising actions more often, the critic sees more and more examples of valid sequences and thus is able to establish an ever more accurate estimate of the value of an action, which in turn enables the actor to perform better, and so on.

The model learned on a continuous stream of self produced actions, mediated by the environmental loop. If the actor chose an action that was physically impossible to accomplish (e.g. picking up an object that was already held) the target value for this unit was set to 0, the error propagated back (see "negative evidence") and the choice was repeated. After completion of a task, the context layer was reinitialized; eligibility traces were set to zero.

3. RESULTS

3.1. Learning

The model typically produced between 700 and 1100 correct nutellatoast sequences and between 80 and 350 butternutellatoast sequences in 100000 iterations. The proportion of correct sequences increased steadily until a point of saturation was reached. Similarly, the frequency of negative rewards decreased steadily until it plateaued.

In the trained model, variations occured in the order of picking up objects, eg. Nutella before knife. This is because the reward only depends on the final state (e.g. eat a nutellatoast). If there are several ways to reach this state, all get rewarded and thus learned. Similarly, the model discovered that in most cases it is more efficient not to put down the knife between task sequences. We return to this important point in the discussion.

3.2. Performance

The performance of the model was tested by running 100 iterations using the final set of weights and different levels of noise in the output layer.

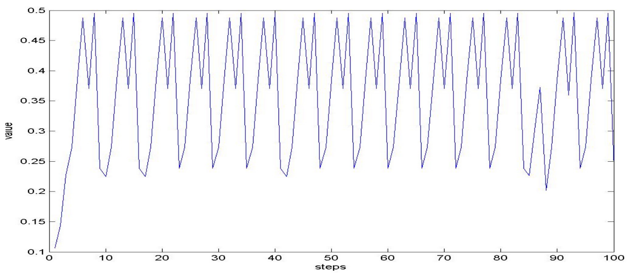

Figure 2; The value function of the critic when tested without noise (T=0). The regular patterns with two peaks correspond to the nutellatoast sequence. The three-peak pattern around step 90 is an example of the butternutellatoast sequence.

Without any noise, the model typically produced between 12 and 16 correct sequences per 100 iterations. Minor deviations were observed, usually due to superfluous actions like fixating something else before fixating the required object (e.g. step 10 in fig. 2). Since it is the only non-deterministic feature in this mode of testing, the occurrence of these disturbances must be attributed to the influence of the context layer, which is reinitialized after completion of a task. The fact that the model always finds its way back to the correct behavior shows its ability not only to produce one fixed sequence, but also to recover quite flexibly from disturbances. Because of the way it is trained, the model has learned more than just one task sequence. In fact, it has acquired knowledge on many different ways that lead to the final aim of receiving a reward, and these different ways include not only variations in the order of actions, but occasional wrong choices as well.

This is even more evident when testing with a higher level of noise. Up to T~=0.3, structured goal approaching behavior was produced, although it took increasingly longer for the model to find its way back to completing a certain sequence. For even higher levels of noise, the model only occasionally found its way into a sequence attractor's basin.

It is interesting to note that disturbances occur more often at or before the beginning of a sequence than towards the end. This is because values of states early in a sequence are low, and the corresponding action units thus cannot gain a large advantage over the other output units. Secondly, the influence of the randomly initialised context layer is bigger shortly after resetting. Furthermore, the choices of actions that lead to a certain outcome are more tightly constrained at later stages. One might start in different ways to make nutellatoast, but picking it up and eating it will always be the unique possible end. This characteristic of the model's behavior leads to testable predictions on human action sequencing. Specifically it suggests that humans are more likely to commit errors near the beginning rather than the end of a sequence and action selection will get faster towards the end of a task sequence.

3.3. Representations

Visualization of the trajectory in activation space of the actor by means of MDS (multi dimensional scaling) shows the "shadowing" typical for recurrent networks.

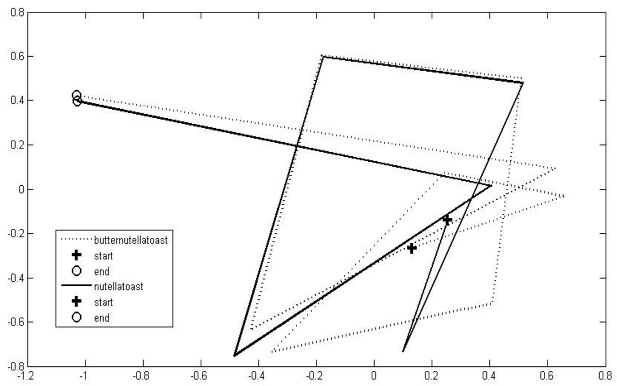

Figure 3; MDS-plot of two task sequences performed by the model when tested without noise.

The last six steps of the two sequences in figure 3 resemble each other closely, despite the fact that the Nutella is added to the plain toast in one case, but to the toast with butter already on it in the other case (i.e. different input). Also, the adding-butter subsequence (steps 1-4 in the dotted trajectory) is similar to adding Nutella (steps 5–8). This picture exemplifies the ability of the network to grasp structural similarity in sub sequences and to share information between them.

3.4. Parameters

In order to understand the robustness of the model's behavior, parameters were explored in over 80 simulations with different settings. The exploration cannot be seen as exhaustive due to the many degrees of freedom resulting from the interaction of the parameters.

3.4.1. Reward

A full reward of 1 was given at the end of a completed task sequence (nutellatoast or butternutellatoast). An intermediate reward of 0.5 was provided every time the knife was used correctly to apply butter or nutella on the toast. Adding something twice or adding butter to toast with Nutella already on it was not rewarded in the initial simulations. If these constraints were loosened, the respective task version appeared in the performance of the model.

Without the intermediate reward, the model did not produce a sufficient number of full task sequences (by chance) to start the learning process.

3.4.2. Recurrence (X)

The factor *X* is multiplied with the connections from hidden to context layer of the actor. A value of 0 indicates a pure feedforeward net, *X*=1 is a standard SRN. Neither of these options worked in the present simulation since on the one hand, recurrence is needed to cover the non-markovian features of the task, but the full influence of the context layer, on the other hand, hampered the net in initially learning to differentiate between inputs. Hence, the value of *X* was linearly changed from 0 to 1 during learning. This could be interpreted as a simple implementation of the starting small principle [9].

3.4.3. Learning rate (lr)

The learning rate for both nets (actor and critic) was decreased linearly from 0.8 to 0.0 during learning. The extremely high value at the start is needed for two reasons. Firstly, the nets must make the most out of the rare positive feedback while operating on a very imperfect policy (random behaviour at the start). Secondly, the moving target provided by the critic will be very low, initially, so that a large step towards this value is not a big change in the actor's weight matrix. The learning rate was decreased to allow for fine-grained adjustments towards the end of the learning process.

3.4.4. Gamma

Gamma is a discount parameter that influences, how high a future reward is valued. It ranges between 0 (future values/rewards are not taken into account and thus no TD-learning) and 1 (every state that eventually leads to a reward has the same value, thus obscuring which action makes the reward available quickest).

Good results were obtained for *gamma* between 0.8 and 0.9, the difference being that the model is more likely to stay on its path in the former case, but has more difficulties in finding the starting point (lower values at the beginning of the sequence, steeper ascent of the value function towards the end).

3.4.5. Eligibility trace (Lambda)

Lamda, in the theory of reinforcement learning, has the function of restricting the update of transition probabilities to the states recently visited, which in this case is done indirectly through adjustment of the weights. The lower the value of *lambda* (0 < *lambda* < 1), the fewer of these visited states are taken into account. For the present model, high values of *lambda* worked best. For low values the model, if learning reasonably well, rarely found a solution that worked without noise.

3.4.6. Initialising weights

No influence of the values with which weights were randomly initialised was observed. This presumably is because all output activations in both nets tended to go back to zero in the first few hundred steps of learning. Initial differences were thus nullified.

3.4.7. Level of noise (T)

A certain level of noise is needed in every model that uses reinforcement learning in order to guarantee that valid states are visited, even if the policy currently used is wrong and thus would not lead to these states. Only if positive examples are provided can the model learn to produce them more often. Adding noise can be interpreted as enforcing a certain amount of exploration, as opposed to strictly following the current policy (exploitation). There is a trade-off between exploration and exploitation since exploration is needed to improve the policy but, on the other hand, it leads to unnecessary mistakes in cases where behavior already is adequate.

In the present model, the level of noise was controlled by adding an equal distribution of values between 0 and *T* to the output layer of the actor. In combination with a winner-takes-all choice, this implementation is equivalent to the temperature parameter *T* in a probabilistic softmax function. Its role is to create a blur around the output activation such that the unit with the highest activation has the highest chance to be chosen, but that other units will occasionally get a chance, too. The choice is more random if there are many equally highly activated units, not random at all if only one unit is maximally excited but none of the others are. With *T* held constant, this still leads to an increasingly structured behavior of the actor as its output activations get more differentiated.

Decreasing *T* during learning leads to more correct sequences (up to 10 times more) in this phase, but not to better performance when tested without noise. In fact, these nets were easier to divert, presumably because they lacked experience in recovering from wrong choices.

Note that in the standard case with T=0.5, the model's perfect behavior when tested without noise arose out of an overall rather poor performance during learning. This so-called off-policy learning shows the relative independence of critic and actor and has interesting implications for a psychological view of learning as a whole. Specifically it suggests that there is much to be learned from doing things wrongly.

3.4.8. Number of steps

100000 iterations proved to be sufficient, no advantages were observed for more extensively trained models. Due to the two linearly changing parameters X and lr, not much can be stated about the minimum number of iterations needed for convergence.

3.4.9. Number of units

Several simulations were carried out in order to determine the minimum size of the hidden layer required. Some models with 4 hidden/context units for the actor and 3 hidden units for the critic obtained good results, but learning seemed to get unstable. Using 7 and 5 hidden units, respectively, appears to be on the safe side.

3.4.10. Negative evidence

The initial idea was to speed up learning by providing a negative reward for the actor if a physical impossible action is chosen (e.g. pick-up when nothing is fixated or too many objects are already held). This information is available for an embedded system and, if used efficiently, should lead to a massive restriction of the state space the model has to explore. The possible actions in the present task, e.g., could be reduced by about 40%. In the present simulation this restriction didn't seem to work, though, because the values and activations off all actions go back to close to zero during the first few hundred iterations and impossible actions therefore keep on getting chosen. Models trained without the negative reward term, on the other hand, only occasionally learned the full task.

Only a closer look reveals the benefit of providing the model with negative evidence. When, at the beginning of learning, a reward is encountered, the weights in the actor that led to the chosen action are changed to a large extend (very high learning rate). This inevitably leads to the situation that different inputs inappropriately will also elicit a higher activation of that same action. This ability to generalize can be desirable if applied to "similar" situations, but because of the net's initial inability to correctly categorize situations and the extremely high learning rate, it will apply it much too widely. In the present model, valid experience is a valuable thing, because it is dependent on the current behavior of the actor. Furthermore, experience is mainly based on positive evidence (reward given or expected). In this learning regime it is very hard to find out that an overly valued state really is not so good after all. By introducing the negative evidence term, the actor is given a new, more direct and more frequent source of experience that helps to correct the initial overgeneralization. Owing to this mechanism the net can take more information out of the occasional positive reward signal and at the same time is able to exhibit better behavior and thus to provide more positive evidence.

4. DISCUSSION

Most of the literature on reinforcement learning is concerned with the abstract mathematical properties of models, proofs of convergence and different algorithms. Therefore, the tasks learned are usually illustrative and easy to analyze (e.g. navigation in gridworlds, games). This approach reveals important principles in terms of what can be learned by which means, but the relationship to real world tasks stays somewhat obscured. Neither is it possible to determine in a principled way, how powerful combinations of several techniques are (e.g. function approximation with neural networks and reinforcement learning). Nor is it clear how complicated natural tasks will prove to be if different kinds of available information are taken into account.

The present routine task, by being naturalistic, involves many complexities beyond the standard theoretical tasks frequently considered in the reinforcement learning literature: a large state space (~ 8^8 states); many possible actions/output states; non-Markovian states (e.g. if butter and toast are held, is the butter on the toast or were they picked up separately?); and relatively long sequences and thus long distance dependencies. The simulations reported here show that it is possible to learn this structurally quite complex task within a relatively simple model with few *a priori* assumptions in the comparatively short time of 100,000 iterations. The power of this model arises from the interplay of different principles, each of which simplifies the learning of the task at hand in its own fashion:

- Because the model is embedded in an environment, some of the possible states in state spaces are not accessible and thus do not have to be taken into account.
- Because of the recurrent connections in the actor, information of earlier states can be preserved and therefore can turn the decision at a later state into a Markov Decision Problem.
- The actor can use the knowledge acquired by another part of the model, the critic, as positive evidence that helps it to improve performance and thus leads to more correct examples to learn from for the critic. The relative independence of the two nets guarantees good learning even for imperfect policies.
- The use of negative evidence for one-step sequences can furthermore improve the actor's behaviour.
- Reward for shorter sequences can bootstrap the model to the acquisition of longer sequences. Generalization, for valid, as well as for invalid actions, helps the model to learn different sequences with partial overlap more efficiently.

The task representations the actor develops are comparable in many respects to the ones in an SRN that uses supervised learning: they overlap structurally (generalization), they allow for information sharing between similar (sub) sequences, they are graded and context sensitive etc. However, in contrast to a standard SRN, a reinforcement model explores the whole state space and thus is able to recover flexibly from wrong choices along the way. Rather than forming attractors for the valid examples and organizing the reminder of the state space in terms of proximity to them, a reinforcement model experiments with many different sequences and discovers which choices to make in order to get rewarded in the end. Hence, the possible variations in a task are discovered by trial and error and the model is not dependent on a carefully balanced training set that provides it with an equal amount of experience for every task version it is to produce. Even if the resulting representations resemble each other in many ways, in a reinforcement model they develop because it tries to reach rewarded states (= goals), not because it adapts to the examples it is taught with. As an additional advantage of this mechanism one might expect such a model to be less prone to catastrophic interference, because a certain amount of experience with all physically possible sequences is implicit in the final pattern of connection weights.

Evidently, more work needs to be done before the present model can give a full-scale account of hierarchical routine action. Several adjustments could help the model to learn more efficiently, such as bootstrapping the learning process with a few valid examples of task sequences, or adjusting the learning rate and/or the exploration/exploitation ratio dynamically according to some measure of current performance. Botvinick et al. [10] propose a measure of conflict that seems suitable to indicate how well the model is doing at a certain point in time. High conflict indicates inefficient behavior and therefore might trigger an increased learning rate while low conflict would indicate that the policy is adequate and should not be changed.

Another issue seems to be more important though: so far, all the rewards given simply are numerical values along one dimension. If one sees the value maximizing behavior of the nets as corresponding to goal directed behavior in humans as suggested by the reinforcement literature, then, so far, there is only one single goal. The critic is unable to discriminate between the rewarded sequences, because they all influence its single output unit. The aim of our ongoing work is to implement multiple goals, as well as a way to dynamically switch between them. In a recent model of the Wisconsin card sorting task [11], the adaptive critic (AC) component plays this role by influencing the sorting rule the network applies via lateral connections to the hidden layer. Transferred to our model this means a way to tell the net which one of several value functions, each representing a different goal, is to be maximized at a certain point in time. Importantly, this would result in one goal per (sub) task, not one ‘instruction’ per task version.

In summary we have shown that a reinforcement approach to modeling hierarchical routine action is promising. The presented model shares the advantages of the emergent representations in supervised PDP models, but has additional benefits in terms of plausibility of the process of learning, flexibility of representations and the ability to account for goal directed behavior.

5. REFERENCES

[1] M.M. Botvinick and D.C. Plaut, Doing without schema hierarchies: a recurrent connectionist account to normal and impaired routine sequential action. *Psychological Review*. 111(2). 2004.

[2] D.E. Rumelhart, P. Smolensky, J.L. McClelland, and G.E. Hinton. *Schemata and sequential thought processes in PDP models*. Parallel distributed processing: explorations in the microstructure of cognition, Vol. 2: Psychological and biological models, ed. D.E. Rumelhart and J.L. McClelland. Vol. 2, MIT Press: Cambridge MA. 1986.

[3] R.A. Schmidt, A schema theory of discrete motor skill learning. *Psychological Review*. 82(4): p. 225 - 260. 1975.

[4] R. Cooper and T. Shallice, Contention Scheduling and the control of routine activities. *Cognitive Neuropsychology*. 17(4): p. 297 - 338. 2000.

[5] J. Elman, Finding structure in time. *Cognitive Science*. 14: p. 179 - 211. 1990.

[6] M.M. Botvinick and D.C. Plaut, Representation task context: proposal based on a connectionist model of action. *Psychological Research*. 66: p. 298 - 311. 2002.

[7] N. Ruh and R. Cooper. *Modelling Routine Action*. Poster presented at the IC2004. March 2004. Guenne, Germany.

[8] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*. 4. ed, Cambridg, MA: MIT Press. 2002.

[9] J. Elman, Learning and development in neural networks: The importance of starting small. *Cognition*. 48: p. 71-99. 1993.

[10] M.M. Botvinick, T.S. Braver, D.M. Barch, C.S. Carter, and J.D. Cohen, Conflict Monitoring and Cognitive Control. *Psychological Review*. 108(3): p. 624 - 652. 2001.

[11] N.P. Rougier and R.C. O'Reilly, Learning representations in a gated prefrontal cortex model of dynamic task switching. *Cognitive Science*. 26: p. 503-520. 2002.