Inferring Adaptive Goal-Directed Behavior Within Recurrent Neural Networks

Sebastian Otte[1(✉)], Theresa Schmitt[1], Karl Friston[2], and Martin V. Butz[1]

[1] Cognitive Modeling Group, University of Tübingen, Sand 14, 72076 Tübingen, Germany
sebastian.otte@uni-tuebingen.de

[2] The Wellcome Trust Centre for Neuroimaging, UCL, 12 Queen Square, London, UK

Abstract. This paper shows that active-inference-based, flexible, adaptive goal-directed behavior can be generated by utilizing temporal gradients in a recurrent neural network (RNN). The RNN learns a dynamical sensorimotor forward model of a partially observable environment. It then uses this model to execute goal-directed policy inference online. The internal neural activities encode the predictive state of the controlled entity. The active inference process projects these activities into the future via the RNN's recurrences, following a tentative sequence of motor commands. This sequence is adapted by back-projecting error between the forward-projected hypothetical states and the desired goal states onto the motor commands. As an example, we show that a trained RNN model can be used to precisely control a multi-copter-like system. Moreover, we show that the RNN can plan hundreds of time steps ahead, unfolding non-linear imaginary paths around obstacles.

Keywords: Recurrent neural networks · Long short-term memory · Neurorobotics · Robot control · Active inference · Goal-directed behavior

1 Introduction

Recently, it was shown that recurrent neural forward models can be used to compute the inverse kinematics of many-joint robot arms [12]. LSTM-like RNNs [6,10] were trained to estimate end-effector poses based on specified arm configurations. *back-propagation through time* (BPTT) was used to iteratively optimize a goal-oriented inverse mapping, that underwrites the goal-directed movement of the robot arm; essentially enacting the unfolding goal-directed optimization process. In this paper we extend this mechanism into a dynamic scenario, planning control trajectories dynamically through time.

In the motor control literature, direct inverse models [7] learn direct mappings from goal states to actions, which works only when the typically redundant inverse mappings are convex. Multiple forward-inverse models [15] as well as constraint Jacobians have been proposed for inverse redundancy resolution.

© Springer International Publishing AG 2017
A. Lintas et al. (Eds.): ICANN 2017, Part I, LNCS 10613, pp. 227–235, 2017.
https://doi.org/10.1007/978-3-319-68600-4_27

However, the results are always direct mappings from desired states to motor control commands, precluding online adaptations. Models that encode redundancies and resolve them on the fly are related to human motor control [1], but the requisite population-encoded spaces do not scale up to larger control spaces.

We show that dynamic active-inference-based processes, which are corollaries of the free-energy-based inference principle [3,5], implemented within RNNs, can generate suitable control commands when trajectory redundancies need to be resolved on the fly and even when more complex trajectories need to be found. The RNN learns a recurrent forward model, developing internal predictive encodings that reflect unobservable but inferable system dynamics. The resulting online active inference process projects neural activities into the future via the network recurrences, following a tentative sequence of motor commands. This sequence is adapted by back-projecting the error between the hypothetical and desired (goal-like) future system states onto anticipated motor commands, effectively inferring on the fly an action sequence that leads to the goal. Thus, the mechanism solves a sequential policy optimization problem. In exemplary multicopter-like simulations, we show that the approach can not only control dynamical systems with high precision, but even infer longer-term action sequences to circumnavigate obstacles.

2 Recurrent Anticipatory Neural Control Model

The RNN model consists of two main processing components. First, training the forward model, that is, learning a neural approximation of the dynamical system of interest. Second, inferring dynamic action sequences in order to generate adaptive goal-directed behavior in a continuous, dynamic control scenario.

2.1 Learning the Forward Model

Let us consider a simplified formulation of a discrete-time dynamical system. At a certain time step t the system is in a specific system state. Because we assume a dynamic, *partially observable Markov decision process* (POMDP) [13], the next system state is typically not deducible exactly from the current observables. Thus, we separate the system state into the perceivable state components $\mathbf{s}^t \in \mathbb{R}^n$ and the hidden state components $\boldsymbol{\sigma}^t \in \mathbb{R}^m$ Additionally, the system can be influenced via k control commands denoted by $\mathbf{x}^t \in \mathbb{R}^k$. The next system state $(\mathbf{s}^{t+1}, \boldsymbol{\sigma}^{t+1})$ is determined by

$$(\mathbf{s}^t, \boldsymbol{\sigma}^t, \mathbf{x}^t) \xmapsto{\varPhi} (\mathbf{s}^{t+1}, \boldsymbol{\sigma}^{t+1}), \tag{1}$$

where the mapping $\varPhi$ models the forward dynamics of the system. Thus, the next system state depends on the current control inputs as well as, in principle, on the entire state history somehow encoded in the (hidden) system state components. The task of the forward model learner is to approximate the forward model $\varPhi$ given the current state $\mathbf{s}^t$ and current control commands $\mathbf{x}^t$, as well as an internal memory encoding derived from the previous state information

$\{\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^{t-1}\}$ and motor commands $\{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{t-1}\}$. Learning proceeds by dynamically processing current state information and predicting the next state information. We propose to use LSTM-like RNNs [6] for this task because LSTMs can predict accurately and can associate even temporally dispersed input events, which is essential when learning forward models in POMDPs. Moreover, LSTMs provide stable gradients over long time periods, which is critically important when complex control trajectories are required.

2.2 Action Sequence Inference

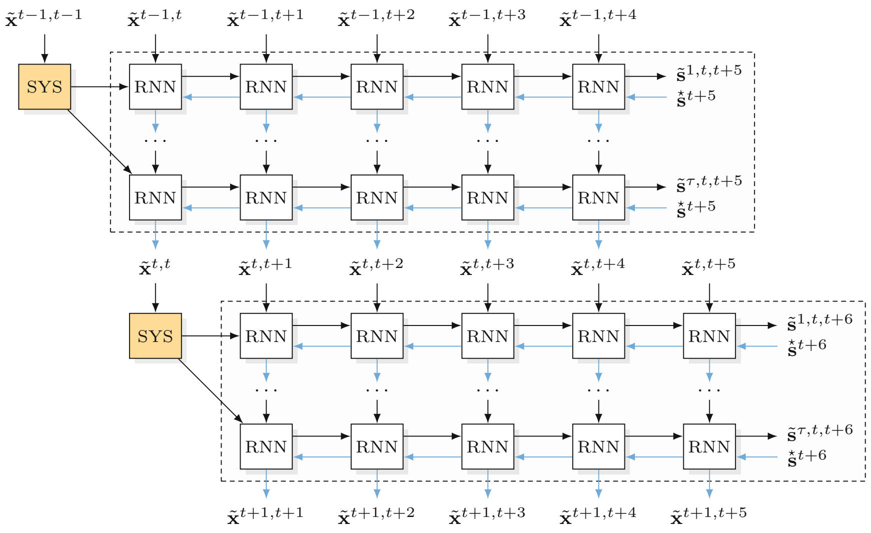
Given a certain action sequence and an initial state, the RNN can predict a state progression that is expected when executing the imagined action sequence by means of the learned recurrent forward model. To effectively control the system, however, the inverse mapping is required, that is, an action sequence needs to be inferred to approach a desired goal-state (or follow a sequence of goal-states) from an initial state. In a stationary scenario, this can be accomplished by means of BPTT in combination with gradient descent, as shown, for example, in [12]. However, this principle cannot be directly projected into the temporal dynamic domain, because motor commands need to be executed sequentially and the commands are dynamically interdependent over time.

Our active inference procedure, sketched-out in Fig. 1, unfolds within the RNN system. At any point in "world" time t, the RNN has a certain internal neural activity reflecting the dynamical system's state, which the RNN recurrently updates given the current state and motor command signals. Additionally, the RNN maintains a preinitialized anticipated action sequence, that is, an *action policy*, and anticipated gradient statistics. At each world time step, the policy is (further) refined taking the actually encountered experiences into account. Currently, the anticipated action sequence has a fixed length, which corresponds to the actual temporal planning horizon T.

Based on the previous system state as well as the previous forward RNN hidden activations, that is, the recurrent context, the action policy is adapted by generating a sequence of error gradients. This is accomplished by projecting the predicted state progression into the future given the current action policy and back-projecting the discrepancy (quantified by the loss $\mathcal{L}$) between the predicted state and the desired goal-state (or sequence of goal states) in time using back-propagation through time (BPTT) [14]. As a result, the $\mathcal{L}$ is projected onto the action policy, thus updating it. During the forward-projection, the RNN is fed with its own state prediction. After back-projecting the error, the input gradient is computed via

$$\frac{\partial \mathcal{L}}{\partial x_i^{t'}} = \sum_{h=1}^{H} \left[\frac{\partial net_h^{t'}}{\partial x_i^{t'}} \frac{\partial \mathcal{L}}{\partial net_h^{t'}} \right] = \sum_{h=1}^{H} w_{ih} \delta_h^{t'}, \tag{2}$$

where h indexes the hidden units after the input layer, $net_h^{t'}$ denotes the weighted sum of inputs (or *net input*) into unit h at time t', and t' with $t \leq t' \leq t+T$ is the running time index over the projected future sequence. Note that each gradient signal $\delta_h^{t'}$ recursively depends on the future $\delta_{h'}^{t'+1}$, $\delta_{h'}^{t'+2}$, ..., $\delta_{h'}^{t+T}$ signals,



$\tilde{\mathbf{x}}^{t-1,t-1}$
$\tilde{\mathbf{x}}^{t-1,t}$
$\tilde{\mathbf{x}}^{t-1,t+1}$
$\tilde{\mathbf{x}}^{t-1,t+2}$
$\tilde{\mathbf{x}}^{t-1,t+3}$
$\tilde{\mathbf{x}}^{t-1,t+4}$
SYS
RNN
$\tilde{\mathbf{s}}^{1,t,t+5}$
$\overset{\star}{\mathbf{s}}^{t+5}$
$\tilde{\mathbf{s}}^{\tau,t,t+5}$
$\overset{\star}{\mathbf{s}}^{t+5}$
$\tilde{\mathbf{x}}^{t,t}$
$\tilde{\mathbf{x}}^{t,t+1}$
$\tilde{\mathbf{x}}^{t,t+2}$
$\tilde{\mathbf{x}}^{t,t+3}$
$\tilde{\mathbf{x}}^{t,t+4}$
$\tilde{\mathbf{x}}^{t,t+5}$
$\tilde{\mathbf{s}}^{1,t,t+6}$
$\overset{\star}{\mathbf{s}}^{t+6}$
$\tilde{\mathbf{s}}^{\tau,t,t+6}$
$\overset{\star}{\mathbf{s}}^{t+6}$
$\tilde{\mathbf{x}}^{t+1,t+1}$
$\tilde{\mathbf{x}}^{t+1,t+2}$
$\tilde{\mathbf{x}}^{t+1,t+3}$
$\tilde{\mathbf{x}}^{t+1,t+4}$
$\tilde{\mathbf{x}}^{t+1,t+5}$

Fig. 1. The continuous action inference procedure. In each iteration, an optimization cycle (gray boxes) is performed, refining the anticipated actions. Within an optimization cycle the progression of the dynamical system is projected into the future (here $T = 5$ time steps ahead) and the discrepancy between the predicted state and the desired goal-state is back-projected via BPTT onto the individual motor commands of the policy. The black lines indicate context and information forward flow. The blue lines indicate gradient flow. $\tilde{\mathbf{x}}^{t',t''}$ refers to the action vector for time step t'' based on the context of world time step t'. Within the optimization cycle, $\tilde{\mathbf{s}}^{\tau,t',t''}$ refers to the state prediction for time step t'' based on the context of world time step t' in the τ-th optimization iteration, whereas $\overset{\star}{\mathbf{s}}^{t''}$ refers to the desired system state for time step t''. (Color figure online)

which carry the gradient information back through time. For each world time step, the above procedure is repeated several times and embedded in a higher level optimization framework, for which we use Adam [8], which is more robust and usually faster than vanilla gradient descent with momentum. To move on to the next world time step, the first action of the refined sequence is executed (including system and RNN update) and the anticipated action sequence begins with its successor. The new last action is initialized with the previous last or a neutral action. The accumulated gradient statistics of Adam (first two moments) are retained to provide an uninterrupted continuation of the optimizer dynamics. As a result, the active inference mechanism can be applied seamlessly and a continuous, active inference process unfolds, generating goal-directed interactions with the environment while continuing to plan the next steps.

2.3 On the Relation to Active Inference and Free Energy

Conceptually, the current scheme inherits the important aspect of active inference; namely, casting the control problem in terms of planning as inference.

The crucial aspect of our scheme – that renders it a form of active inference – is that the control variables $x_i^{t'}$ are not deterministic states but random variables that are inferred online. This is necessarily the case because we consider control in the future – before it is realized. Effectively, this means that fictive policies are inferred under prior beliefs about outcomes. Crucially, this inference is updated online by assimilating outcomes to date, performing sequential policy optimization. In other words, instead of a fixed state-action policy, we infer policies online with the help of the learned forward model. On this view, our scheme is effectively minimizing a path integral of (free) energy, where the energy corresponds to the cost function $\mathcal{L}$. The goal can be regarded as a prior expectation about the path's endpoint, while the sum of squared errors measures the discrepancy between predicted and realized state-space trajectories. Formally speaking, this measure corresponds to the expected free energy under the policy in play – and the path integral approximates an Hamiltonian action.

3 Experiments

Our experiments are based on a simple dynamical system simulation of a multicopter-like object, which we call *rocket ball*. The rocket ball is positioned in a rectangular environment with gravity. It has two propulsion units spread at a 45° angle from the vertical axis to both sides, inducing thrust forces in the respective direction. Each unit can be throttled within the interval $[0, 1]$.

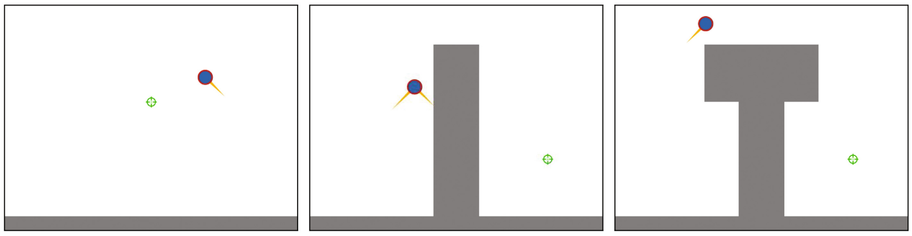


Fig. 2. Illustration of three rocket ball scenarios. The rocket ball (blue) has to reach the target (green). The red circle indicates the current position estimate predicted by the RNN. In free space (left), the rocket ball can approach the target directly. With a convex obstacle (center) a short detour is necessary. The concave obstacle case (right) requires a further ranging detour and, in addition, the local minima are more dominant. (Color figure online)

We considered three different scenarios, which are depicted in Fig. 2. In the free scenario, each target point can be approached directly somewhat linearly, while with increasingly complex obstacle constellations, reaching the goal state requires increasingly complex anticipatory behavior. For each scenario we trained an RNN with one hidden layer consisting of 16 LSTM blocks, which additionally provide block-wise gate-to-gate connections [10,11]. We also tried standard

RNNs, which, however, yielded significantly less accurate predictions and which could not be used for precise control.

At each time step a network is fed with the current position of the rocket ball as well as with the current action command (left and right thrust). The network output is the prediction of the position, the velocity, and the acceleration of the rocket ball for the next time step. In preliminary experiments it appeared that predicting the velocity and acceleration, while only signaling position information as input, forces the RNN to make more use of the recurrences (partially observable state). For all scenarios, we trained an RNN with 2 000 sequences each consisting of 200 time steps using stochastic sampling (no mini-batches) and Adam with default parameters (learning rate $\eta = 0.001$, first and second moment smoothing factors $\beta_1 = 0.9$, $\beta_2 = 0.999$). The training sequences were generated based on continuously randomly adjusted thrust values. All experiments were implemented using the JANNLab framework [9]. Note that the approach can in principle be implemented in an online-learning fashion, learning and exploiting the dynamics of the observed system concurrently.

3.1 Short-Term Inference

In free space, only short-term planning is required. The goal is to control the rocket ball such that it reaches the target as quickly and accurately as possible. We obtained the best performing controller by presenting the goal-state at each time-step during back-projection. Moreover, zero velocity and – to a lesser extent – zero acceleration in the goal-state is useful to suppress oscillations around the target. The best trade-off between robust goal state maintenance control and prospective behavior while approaching the target yielded a planning horizon of about 6–8 time steps. Shorter planning horizons increase overshooting, while longer ones decrease the control precision. To infer the action policy, we used Adam with $\eta = 0.01$ and $\beta_1, \beta_2 = 0.9$ and 30 iterations per active inference cycle. In all situations, the target was reached with a precision (RMSE) below 10^{-3} (the environment has a height of 2 units). Figure 3 depicts a short exemplary image sequence of a target approach. While the goal is reached accurately, the thrust is throttled down in anticipation of avoiding overshooting.

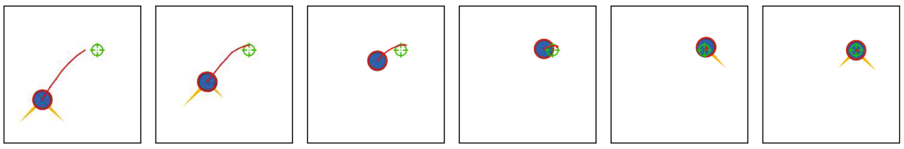


Fig. 3. Image sequence of the rocket ball approaching the target. The red line represents the actual imagined future state progression unfolded by the RNN. The length of the thrust expulsion indicates the respective force. (Color figure online)

3.2 Long-Term Inference

In order to handle the obstacle scenarios (cf. Fig. 2), it was necessary (i) to extend the planning horizon enabling the formation of more complex, indirect imaginary pathways towards the goal state and (ii) to present the goal-state only late in the future projection (here always only at the end of the planning horizon), which enables the RNN to anticipate sequences in which the state temporarily moves away from the target. We used Adam with a more aggressive learning rate of $\eta = 0.1$ but only one error projection per world time step.

Since long-term planning is fairly imprecise, we simultaneously used short-term planning, which assumes control once the distance to the target state is below a certain threshold. As a result, the rocket ball can reach the target with high precision when facing a convex and even a partially concave obstacle. Note, however, the goal reaching trajectories that evolve are only temporarily stable. They may collapse after some time as the inference process permanently searches for solutions that fulfill only the final state objective. At the moment no further constraints are added to the loss function.

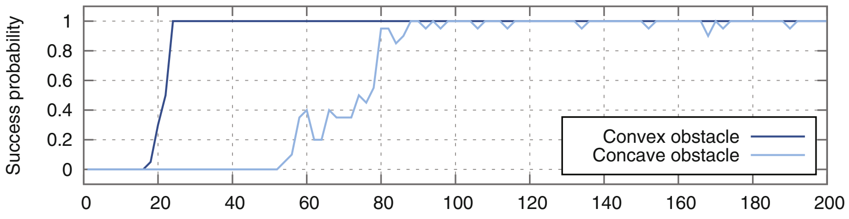


Fig. 4. Reaching the target (i.e. success probability) in the complex scenario depends on the scenario's complexity as well as on the planning horizon (x-Axis).

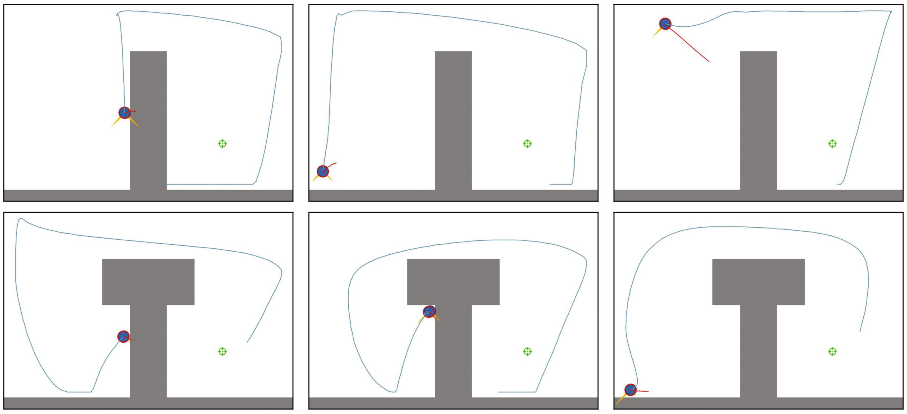


Fig. 5. Exemplar trajectories imagined by the RNN: while long term inference (120 steps) leads around the obstacle, short term inference (red) points towards the target. (Color figure online)

We systematically evaluated the required planning length by means of 20 random trials (the rocket ball is initially placed to the left of the obstacle and the target to the right of the obstacle) for several planning horizons. Figure 4 shows the fraction of trials in which the target was reached after 1 000 world time steps dependent on the set planning horizon. To provide an impression of the system's behavior, Fig. 5 shows six imaginary flight trajectories anticipated by the RNN.

4 Summary and Conclusion

We have shown that LSTMs, which learn a sensorimotor forward model in a dynamic control scenario, can generate active-inference-based, goal-directed action policies online. Active inference is accomplished by projecting neural activities into the future and back-projecting the error between the forward projections and currently desired goal states onto motor commands. The mechanism was even applicable for planning around obstacles more than hundred time steps into the future.

In should be noted that we could not achieve similar results with standard RNNs. Additionally, the usage of the Adam-based BPTT played an essential role in our system, particularly when unfolding long-term policies. Adam inherently stabilizes the gradient by smoothing it and by suppressing gradient fluctuations, dividing the gradient by its current variance estimate. As a result, the emerging policy tends to avoid uncertain regions of the gradient landscape, unfolding and adapting trajectory plans in a dynamically changing but stable manner. In other words, the mechanism attempts to achieve the extrinsic goal state while maintaining a stable trajectory, which is intrinsically motivated. Interestingly, expected free energy can also be decomposed into intrinsic and extrinsic values [4]. The extrinsic value corresponds to the degree to which goals or prior preferences are realized. In contrast, the intrinsic value scores the reduction in uncertainty about hidden or unobservable states afforded by a particular course of action. It should be explored further to which extent these two notions are indeed complementary or even identical.

Future work concerns adding suitable gradient optimization terms to the loss function, such as a term to minimize motor control effort to foster the generation of fully optimal paths. Moreover, additional scenarios involving more complex environments need to be evaluated to confirm the robustness of the approach. At the moment the system essentially learns control in only one dynamic system scenario. Thus, the grand challenge remains to extend this approach to hierarchical control scenarios, in which distinct sensorimotor controllers are necessary depending on the current event-based circumstances [2] and mutual interactions with other dynamical objects, possibly blending control responsibilities over event boundaries incorporating multiple temporal resolutions.

References

1. Butz, M.V., Herbort, O., Hoffmann, J.: Exploiting redundancy for flexible behavior: unsupervised learning in a modular sensorimotor control architecture. Psychol. Rev. **114**, 1015–1046 (2007)
2. Butz, M.V.: Towards a unified sub-symbolic computational theory of cognition. Fronti. Psychol. **7**(925) (2016)
3. Friston, K.: The free-energy principle: a rough guide to the brain? Trends Cogn. Sci. **13**(7), 293–301 (2009)
4. Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., Pezzulo, G.: Active inference: a process theory. Neural Comput. **29**(1), 1–49 (2016)
5. Friston, K.J., Daunizeau, J., Kilner, J., Kiebel, S.J.: Action and behavior: a free-energy formulation. Biol. Cybern. **102**(3), 227–260 (2010)
6. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
7. Jordan, M.I., Rumelhart, D.E.: Forward models: supervised learning with a distal teacher. Cogn. Sci. **16**, 307–354 (1992)
8. Kingma, D.P., Ba, J.L.: Adam: a method for stochastic optimization. In: 3rd International Conference for Learning Representations abs/1412.6980 (2015)
9. Otte, S., Krechel, D., Liwicki, M.: JANNLab neural network framework for Java. In: Poster Proceedings MLDM 2013, pp. 39–46. ibai-publishing, New York (2013)
10. Otte, S., Liwicki, M., Zell, A.: Dynamic cortex memory: enhancing recurrent neural networks for gradient-based sequence learning. In: Wermter, S., Weber, C., Duch, W., Honkela, T., Koprinkova-Hristova, P., Magg, S., Palm, G., Villa, A.E.P. (eds.) ICANN 2014. LNCS, vol. 8681, pp. 1–8. Springer, Cham (2014). doi:10.1007/978-3-319-11179-7_1
11. Otte, S., Liwicki, M., Zell, A.: An analysis of dynamic cortex memory networks. In: International Joint Conference on Neural Networks (IJCNN), pp. 3338–3345. Killarney, Ireland, Jul 2015
12. Otte, S., Zwiener, A., Hanten, R., Zell, A.: Inverse recurrent models – an application scenario for many-joint robot arm control. In: Villa, A.E.P., Masulli, P., Pons Rivero, A.J. (eds.) ICANN 2016. LNCS, vol. 9886, pp. 149–157. Springer, Cham (2016). doi:10.1007/978-3-319-44778-0_18
13. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (1998)
14. Werbos, P.: Backpropagation through time: what it does and how to do it. Proc. IEEE **78**(10), 1550–1560 (1990)
15. Wolpert, D.M., Kawato, M.: Multiple paired forward and inverse models for motor control. Neural Netw. **11**, 1317–1329 (1998)