Term Project Paper
Predictive representations to link model-based and model-free learning

Daniel Bindemann
Daniel.Bindemann@gmx.de

Computational Neuroscience
Albert Ludwig University Freiburg
Summer Semester 2018

Abstract. When faced with sequential decision tasks, such as navigating through a maze to find food, animals have shown to exhibit distinct behaviour following different changes in the environment. In certain scenarios, the test subject is able to successfully adapt to the change and reach the goal without retraining. In other scenarios, the subject seems to be unable to fully adapt as it chooses a previously learned path which is no longer the shortest to the goal. There is a need for algorithms modelling reinforcement learning in brain, which can unify these observations while staying in line with neural evidence about their neural implementation. This paper reviews three algorithms proposed for this purpose which all build upon the successor representation, an internal model of the transition structure of the environment which enables incorporation of model-based capabilities on a basis of model-free temporal-difference learning. The algorithms incorporate model-based features to a different extent, which will be demonstrated on small spatial navigation tasks in grid world environments.

1 Introduction

Many problems in the real world can be represented as sequential decision tasks. An agent has to interact with the environment in a series of steps, where performing an action is associated with some kind of reward. The goal of the agent is to choose its action in each step such that the overall reward in the long run is maximized. The challenge is, that the agent usually does not fully know the environment and cannot know in advance, which sequence of actions would lead to the desired results. Furthermore, the environment might be stochastic, which adds uncertainty to the outcome of actions. As a consequence, the agent has to figure out the best action in each state based only on the relationship between previously executed actions and their observed outcome.

This problem is addressed by reinforcement learning algorithms, which can be roughly divided into model-based and model-free approaches, each with their advantages and downsides. It turns out that observed real-world behaviour doesn't fit strictly into either category and shows properties of both approaches. The goal of this paper is to discuss hybrid algorithms which have been devised to bridge the gap between the two kinds of approaches and provide mechanisms which fit more closely to the kind of behaviour observed in experiments. First, the characteristics and differences between model-based and model-free reinforcement learning will be discussed. Then, the successor representation will be presented, which lays the basis of the algorithms considered here. A few simple spatial navigation tasks will be introduced, which serve as exemplary reinforcement learning tasks to demonstrate the differing capabilities of the algorithms. Finally, three algorithms will be discussed, which successively add more model-based properties on top of model-free core mechanisms.

2 Model-based vs model-free learning

Model-based algorithms use an internal model of the world to compute the expected cumulative reward of an action $a \in A$ by considering all possible following sequences of state transitions until the goal has been reached. If the environment (i.e. reward function $R(s, a, s')$ and transition

function $P(s'|s,a)$) is known, this can be computed using dynamic programming based on the Bellman equation:

$$\sum_{a \in A} \pi(a|s) \left[R(s,a) + \sum s' P(s'|s,a) \gamma V^{\pi}(s') \right] \tag{1}$$

The equation defines the value of a state recursively depending on the value of successor states. To ensure finite values and add a bias for early rewards, successor state values are multiplied by a discount factor γ to ensure finite numbers and favour shorter paths. Turning the equation into an update rule yields an iterative procedure which, by Bellman's principle of optimality, eventually converges to the optimal value for each state. This enables computation of the optimal policy by picking the action maximizing the expected value of successor states. Complete recomputation of the value function based on a model allows flexible adaptation upon any changes in the environment without any retraining. On the downside, apart from requiring that the environment must be known, it can be computationally expensive, especially so for large environments.

Model-free approaches on the other hand cache estimated values of each state, which get updated online (during interaction with the environment) or offline (during resting phases, using previously gathered experience) and can be immediately used to choose an action at decision time without further computation. While this has the advantage of being computationally cheaper than model-based learning, these approaches generally cannot adapt to changes in the environment without retraining. Sufficient replay of experience can actually lead to similarly good results as found in model-based learning, although it requires accordingly more learning time as well as a large collection of data for a good approximation.

One of the most important mechanisms in the model-free category is temporal difference (TD) learning, which after each action updates the value function estimate V proportionally to an error δ calculated based on the acquired reward and value of the successor state:

$$\delta = R(s,a,s') + \gamma V(s') - V(s) \tag{2}$$

$$V(s) \leftarrow V(s) + \alpha_{TD} \delta \tag{3}$$

In standard neural models, dopaminergic learning implements some kind of TD-learning. By that theory, the value function is encoded by the firing patterns of medium spiny neurons in the striatum, which get input from the frontal cortex that is thought to represent states or state-action pairs. Learning can then be made possible by regulating strength of the cortico-striatal synapses according to a TD prediction error coming from the substantia nigra pars compacta and ventral tegmental area.

However, lesion studies suggest, that some parts of the striatum also play part in model-based reinforcement learning. Since the areas involved in model-free and model-based behavior are both physiologically similar, it is likely that both parts actually perform similar computations, but on different input state representations coming from different cortical regions. A possible candidate for this could be the successor representation, explained in the following.

3 Successor Representation

All algorithms presented in the paper are built upon the successor representation (SR). The SR of a state s is a vector containing the expected number of future visits (also called "occupancies") of each successor state under a certain policy π, starting from state s. As with the value function in the Bellman equation, occupancies are discounted with a factor γ. The formal definition of the successor matrix M is as follows:

$$M^{\pi}(s,s') = E\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(s_t = s') \mid s_0 = s \right] \tag{4}$$

The Bellman equation can then be rewritten using the successor matrix M:

$$V^{\pi}(s) = \sum_{s' \in S} M^{\pi}(s,s') \sum_{a \in A} \pi(a|s') R(s',a) \tag{5}$$

The immediate rewards weighted by the probability of choosing the actions triggering them can be summarized as weights w which can be learned via TD learning in an online or offline stage:

$$V^{\pi}(s) = \sum_{s' \in S} M^{\pi}(s, s') w(s') \tag{6}$$

$$w(s') \leftarrow w(s') + \alpha_{TD} \delta M^{\pi}(s, s') \tag{7}$$

The algorithms differ in how they compute the successor matrix M, which can be done by TD learning or model-based by recursively defining the successor matrix using a one-step transition function T:

$$M^{\pi}(s, :) = 1_s + \gamma \sum_{s' \in S} T^{\pi}(s, s') M^{\pi}(s', :) \tag{8}$$

The successor states following from a certain starting state depend on the used policy, which makes the successor matrix M as defined above an on-policy construct. One can also define a version H of the matrix which maps state-action pairs to occupancies of future state-action pairs:

$$H(sa, s'a') = E\left[\sum_{t=0}^{\infty} \mathbb{I}(sa_t = s'a') \mid sa_0 = sa\right] \tag{9}$$

This can be used for off-policy strategies including offline updates of the matrix, and is also necessary if the environment is not known in the first place. The matrix can then be used to rewrite the value function V to the state-action value function Q:

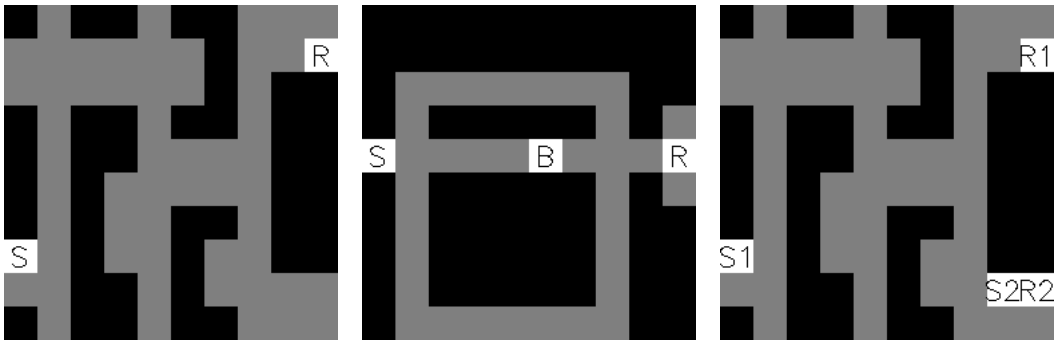
$$Q^{\pi}(sa) = \sum_{s'a'} H(sa, s'a') R(s'a') \tag{10}$$

4 Spatial Navigation Tasks

Experiments such as the ones on rodent spatial navigation by Tolman can be used reveal, whether a form of model-based or model-free learning has been used for the task. These tasks involve changes to the environment, which can be divided into two types: changes to the immediate rewards gained during transitions, defining *revaluation tasks*, and changes to the transition probabilities, also called *contingency changes*. After extensively training a subject on an initially static environment, such a change is introduced. The subject is informed only locally about the change (without any retraining), and the following behavior is observed. It is checked whether the policy of the agent correctly reflects the new optimal policy by observing its path to the goal. If it chooses the new shortest path, it is clear, that at least partly a form of model-based learning must have been applied. If the agent sticks to a previously learned path, which is no longer optimal, it is a sign that the agent is relying on change-insensitive cached values which have been computed previously by a model-free learning algorithm. The following three types of tasks are used:

- **Latent learning**: represents a simple revaluation task. Starting from position S, the agent explores an environment randomly without receiving any rewards. Then, a reward is placed at some field R in the environment, and the agent is informed about this change by (repeatedly) showing it the state R containing the new reward, without showing it any transitions leading there from the starting location. The grid world used for this task is shown in figure 1(a).
- **Detour task**: represents a simple test for adaptation to contingency changes: as in the latent learning task, the agent first explores the environment randomly starting from a location S. A reward is then set up at location R and the agent is trained to find this reward starting from S. Afterwards, a blockade is set up in a location B, which blocks the previously optimal path leading to the reward at R, and requires the agent to make a detour. The agent is informed about the change by putting it directly next to the blockade B. The grid world used for this task is shown in figure 1(b).

– **"Policy revaluation"** (named as in the original paper, where it is first introduced): this is also a type of revaluation task, however it is more difficult to solve. The goal is to test whether an agent can divert from a previously learned policy. After an initial exploration phase, an agent is trained to seek reward at location R1, starting randomly from two positions S1 and S2. A new reward, larger than the one at R1, is introduced at a position R2, which lies in a different direction away from the first reward R1. The agent is informed (again only locally) about the new reward, and one can check whether the agent can find the new reward R2 from the original starting location S1. The grid world used for this task is shown in figure 1(c).



(a) Latent learning task (b) Detour task (c) Policy revaluation task

Fig. 1. Grid worlds for the spatial navigation tasks

5 Algorithms

As mentioned previously, the algorithms implement different ways of computing the successor matrix, but the successor matrix is always used in the same way to compute the value function. Given M^π, the weights of each state are updated using equation 7, which applies TD-learning with a learning rate α_{TD}. The value of a state can then be computed with equation 6 as the dot product of its SR and the state weights.

5.1 The original successor representation (SR-TD)

The simplest algorithm based on a successor representation computes the successor matrix M^π using TD learning with learning rate α_{SR} , i.e. by applying the following update rule after each transition:

$$M^\pi(s,:) \leftarrow M^\pi(s,:) + \alpha_{SR}\left[\mathbf{1}_s + \gamma M^\pi(s',:) - M^\pi(s,:)\right] \tag{11}$$

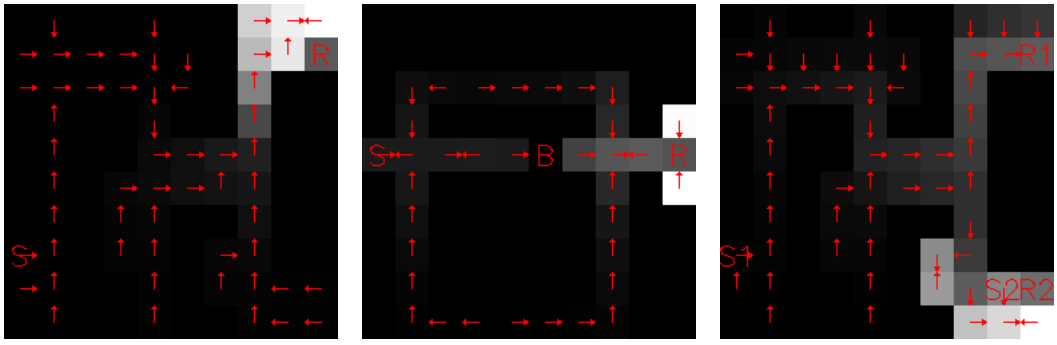
The simulation results for SR-TD are shown in figure 2. Using this algorithm, the agent is able to find the optimal path in the latent learning task. Since the transition structure stays the same, only the weights w need to be updated. By reapplying equation 7, which makes use of the matrix M^π, the weights get propagated to all states along the trajectory between the starting state and the goal, without any need for further retraining.

However, tasks involving transition changes, such as the detour task, cannot be solved. In order to update the successor matrix M using TD-learning, whole trajectories from the starting state to the goal state would be required.

5.2 Dynamic recomputation of successor representation (SR-MB)

The second algorithm computes the successor matrix in a model-based fashion in each step, from a transition matrix T^π which is defined by the current policy π as follows:

$$T^\pi(s,s') \coloneqq \sum_{a \in A} \pi(a|s) P(s'|s,a) \tag{12}$$



(a) Latent learning task (b) Detour task (c) Policy revaluation task

Fig. 2. Simulation results of the first algorithm SR-TD

The agent updates the policy using TD-learning with learning rate α_{pi}, and uses that to update the transition matrix:

$$\pi(:|s) \leftarrow \alpha_\pi \mathbf{1_a} + (1 - \alpha_{pi})\pi(:|s) \tag{13}$$

$$T^\pi(s,:) = \frac{\sum_{a \in A} \pi(a|s)\mathbf{1}_{s'}}{\sum_{a \in A} \pi(a|s)} \tag{14}$$

Just like the first algorithm, SR-MB can solve the latent learning task. Additionally, it can also solve the detour task: local information about a change in transition structure is used to update the transition matrix T^π, which can subsequently be used to recompute the whole successor matrix M^π to reflect this change in the successor representation.

SR-MB still cannot solve the policy revaluation task. It is an on-policy algorithm, because the transition matrix T^π and thus also the future state occupancies M^π constructed from it depend on the actions chosen in each state. The problem arises if the first policy, which was used for construction of T^π (and consequently also M^π), leads to a location in a completely different direction (to R1) away from the new reward R2. If the state occupancies of the paths leading to R2 become too low during the training towards reward at R1, adjusting the weights according to the newly introduced reward at R2 won't be able to sufficiently increase the value function (see equation 6) to change the trajectory of the policy towards R2.

5.3 Off-policy experience resampling (SR-Dyna)

SR-Dyna stores transitions (s, a, s') experienced online and randomly samples k of these with replacement (with more recent transitions used with a higher probability) to update the successor matrix offline. The update rule used here is similar to the one used for the SR-TD approach (equation 11), but instead of computing the error with relation to the next action chosen by the current policy, it chooses the action which is optimal according to the current data (i.e. the action which yields the maximum Q-value among all available actions in the current state):

$$H(sa,:) \leftarrow H(sa,:) + \alpha_{SR}\left[\mathbf{1_{sa}} + \gamma H(s'a',:) - H(sa,:)\right] \tag{15}$$
$$\text{where} \quad a'^* = \text{argmax}_{a'} \sum_{s''a''} H(s'a', s''a'')w(s''a'')$$

This implements an off-policy strategy, which converges towards the optimal Q-Values corresponding to the optimal policy for the given task. The Q-value of a state is computed using the learned successor matrix H and weights w, analogously to the computation of the value function V in equation 6:

$$Q(sa) = \sum_{s'a'} H(sa, s'a')w(s'a') \tag{16}$$

Like the previous two algorithms, SR-Dyna can solve the latent learning and detour tasks. Given sufficient sampling, it can also solve the policy revaluation task due to its off-policy approach. However, if the number of samples for replay is chosen too low, the performance of the algorithm degrades to that of SR-TD.

6 Implementation

The details of the spatial navigation tasks can be found in the directory `/tasks`, where each task is composed by its initial environment in a `.grid` file and a human-readable sequence of commented instructions in a `.task` file. Dots in the grid file are accessible terrain and numbers are used for reference in task instructions (to set starting points and goals). The reinforcement learning agents are implemented as different classes in the file `code/rl.py`.

Unfortunately, the implementation of the agents is not fully functional. SR-TD gives the expected results for the most part, although there are strange artefacts, especially in the cells directly around the terminal state, where the policy points away from the goal state due to its low value. The basic functionality for SR-MB is practically fully implemented, although it does not work as expected. SR-Dyna is only partly implemented.

7 Discussion

Three algorithms have been presented which all are not biologically implausible and could to some extent explain experimental findings in the literature. In particular, all algorithms learn their weights using temporal difference learning, which fits to the standard model of striatal dopaminergic learning. The difference to the standard TD learning approach is that a different input representation, namely the successor representation, is used to bestow the algorithms with model-based capabilities to differing degrees depending on how this state representation is learned. This shows how model-based behavior could result from striatal dopaminergic learning, which is suggested by findings such as the rodent lesion studies.

This also implies that other representations could be additionally passed as input, e.g. using different discount factors to build up a hierarchical model of the world at multiple temporal abstraction levels. None of the presented algorithms is intended to be a final answer to RL in the brain, but each might possibly play part in a larger system and lay the foundation for basic behaviours observed in the highly simplistic planning problems on which they have been evaluated on in the paper. E.g. when evaluating novel situations, traditional model-based approaches such as value iteration are still required for solving, as such behaviour cannot be reproduced by the algorithms presented here alone.

References

1. Russek, E.M., Momennejad, I., Botvinick, M.M., Gershman, S.J., Daw, N.D.: Predictive representations can link model-based reinforcement learning to model-free mechanisms. PLOS Computational Biology (2017)