# Shaping Model-Free Reinforcement Learning with Model-Based Pseudorewards


## Abstract

Model-free and model-based reinforcement learning have provided a successful framework for understanding human behavior and brain data. These two systems are usually thought to compete for control of behavior. However, models exist which integrate the two in a cooperative manner. Dyna, and its variant, prioritized sweeping, use model-based replay of past experience to train the model-free system. Here we introduce a new method which links the two systems via the reward function. Dynamic programming is used to iteratively estimate state values that monotonically converge to state values under the optimal policy. *Pseudorewards* are calculated from these values and used to shape the reward function in a way that is guaranteed not to change the optimal policy. In two experiments we show that this method offers computational advantages over Dyna and prioritized sweeping. It also offers a new way to think about integrating model-free and model-based reinforcement learning. One interesting hypothesis is that emotions may in certain conditions function as model-based pseudorewards that tune a model-free system.


## Introduction

Researchers in both psychology and artificial intelligence taken the relationship between model-free and model-based reinforcement learning, and whether they can be used synergistically. Model-free learning relies on direct trial-and-error interaction with the environmentc̃itesutton1992reinforcement, while model-based learning occurs through simulation of knowledge about the causal structure of the environment [Barto et al., 1995]. Historically, animal psychologists viewed these two systems as distinct and competing hypotheses, with behaviorists arguing in favor of reflexive, model-free learning based on stimulus-response associations [Thorndike, 1933], and Tolman and others positing an internal representation of the environment, referred to as the "cognitive map" [Tolman, 1948]. Nowadays, while both behavioral and neural data indicate that human learning relies on both systems (e.g. [Daw et al., 2005], [Gläscher et al., 2010], [Dayan and Berridge, 2014]), it is typically assumed that they only compete for control of behavior. Yet it is also possible for both systems to cooperate. The Dyna architecture achieves such cooperation by integrating model-free learning with model-based planning [Sutton, 1991]. In Dyna, as model-free learning occurs, state-action transitions and reward contingencies are cached into a model, which simultaneously replays these past experiences, using them to further train model-free state-action values.

Here we introduce a new method for cooperative interaction between model-free and model-based learning. The model-based system generates pseudorewards that are used to shape the model-free reward function. The idea of modifying the reward function to improve behavior is central to *gamification*, in which additional rewards or incentives—*pseudorewards*—are used to influence behavior [McGonigal, 2011]. Moreover, according to the *shaping theorem*, conditions exist under which the optimal policy for a Markov Decision Process will remain invariant to such modifications of the reward function, opening the possibility that pseudorewards can be used to guide agents toward optimal behavior [Ng et al., 1999]. That is, if the optimal policy can be guaranteed to remain unchanged with the introduction of pseudorewards, then pseudorewards can potentially be used to guide the agent to the optimal policy. Using these principles, we show that pseudorewards can provide a link between model-free and model-based learning through modification of the reward function.

This method of cooperation between learning systems offers an appealing alternative to Dyna both conceptually and practically. Dyna uses model-based replay of past experience to refine state-action values, and can be conceptualized in a cognitive framework as using memory replay or planning to train model-free learning. Recent behavioral data with humans performing a retrospective revaluation task is consistent with a cooperative archetecture like Dyna. Our method introduced here links the two systems cooperatively by shaping the reward function. Cognitively, one way that such shaping might occur is with emotion. Emotions could be produced based on a model of the environment and used as pseudorewards to tune the habitual model-free system. To give a concrete example, one might have knowledge of a normative model of ethics, and when one reactively behaves in a way that violates the model, the feeling of remorse might be generated. This feeling could function as a pseudoreward by reshaping the reward function, making the person less likely to misbehave in the future. Our method also offers a practical advantage to Dyna by requiring less computation time and learning more quickly.

In this paper we begin by reviewing the Dyna architecture for integrated model-free and model-based learning. We then introduce our method and the theoretical background on which it is based. We present two experiments which show the effectiveness of our model, and how it compares to Dyna. The first experiment involves learning in a maze environment, and the second experiment uses the standard mountain car problem. We end by discussing additional variants of Dyna and our method, and consider how this integrated approach might provide a useful model for understanding human cognition and emotion.

## Cooperative Reinforcement Learning

Dyna uses a model-based system to replay past experiences, which are used to train the model-free system (Figure **??**). After each action taken in the environment, the model stores the state-action pair and reward received. It then randomly selects $n$ past state-action pairs and replays them, using them to update the model-free system as if they were real actions. In Dyna-Q, the model-free system uses one-step tabular Q-learning (which is what we use in our experiments).The number of simulated planning steps, $n$, is a parameter that can be set to any positive integer value. Dyna typically begins with no knowledge about the causal structure of the environment (i.e. transitions between states and reward contingencies), but builds this knowledge based on experience. However, Dyna can also inherit a model of the environment, and we discuss this possibility later.

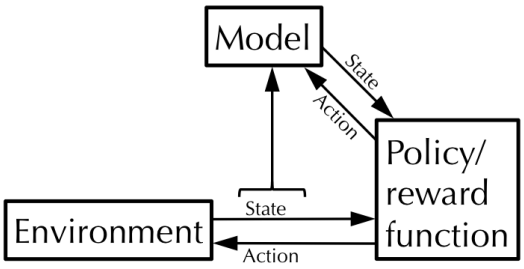


Figure 1: Schematic of the Dyna archetecture

In addition to being a useful artificial intelligence agent for integrating direct learning with indirect replay, it has begun to provide a useful model of human cognition. [Gershman et al., 2014] found behavioral evidence in humans cositent with a Dyna archetecture. Participants performed a sequential decision task with separate phases that tested behavioral revaluatation. When given either more time between phases of learning or a smaller cognitive load, the magnitude of revaluation was larger, consistent with model-based replay of past experience. There are also neurophysiological data that suggest Dyna-like cooperation between the two systems. [Lansink et al., 2009] identified cells in the hippocampus of rats encoding spatial location and cells in the striatum of the same rats that encoded reward. During sleep, the activation of hippocampal cells correlated with and proceeded activation of the same striatal cells that encoded the value of those locations.

## Model-based pseudoreward approximation

Our method uses dynamic programming to approximate state values. These values are used to calculate pseudorewards according to the shaping theorem. By shaping the reward function, pseudorewards provide a link between model-based planning and model-free learning.

### Pseudorewards

Pseudorewards are an intelligible way of conferring extra information to an agent about the reward landscape. Essentially, a small reward is given to the model-free agent (a Q-learner in our experiments) whenever it takes an action that helps the agent move towards the goal. Instead of the agent receiving actual reward $R(s, s')$ when moving from state $s \rightarrow s'$, the agent receives an augmented reward $R'(s, s')$ where

$$R'(s, s') = R(s, s') + F(s, s') \tag{1}$$

### Shaping theorem

Pseudorewards are defined by *shaping functions* $F$. In [Ng et al., 1999], conditions for which the optimal policy $\pi^*$ remains invariant under a *shaping function* are developed. If the shaping function does not possess this invariance policy, it is possible that Q-learning will converge to a suboptimal solution. The simplest example of an invariant shaping function uses the difference in optimal values between the agent's current state and next state:

$$F(s, s') = \gamma V_{\pi^*}(s') - V_{\pi^*}(s) \tag{2}$$

$$V_{\pi^*}(s) = \max_a R(s, s') + \gamma V_{\pi^*}(s') \tag{3}$$

This method is called the *optimal policy pseudoreward*–it encourages the agent to always move down the optimal path from its current state. If $\epsilon = 0$, the agent would move directly to the goal along the shortest path with an $\epsilon$-greedy decision policy.

With *optimal policy pseudorewards* the agent can maximze long-term reward simply by taking the most reward action at each step. In real-world scenarios, it's often unrealistic for a human to have such a complete information set. Computing the optimal policy usually entails solving a linear program.

### Bouded-RTDP

Bounded Real-Time Dynamic Programming [McMahan et al., 2005] is a planning algorithm that attains certain performance guarantees if its lower and upper bounded estimates of state values converge monotonically toward state values under the optimal policy. Importantly, this monotonic convergence toward optimal values can be proved to occur if the lower and upper bounds are initialized properly. Here, we take advantage of this monotone property to calculate approximate (but progressively better) state values using dynamic programming. These values are then used to approximate pseudorewards according to the shaping theorem.

Figure 2 provides a schematic illustration of how dynamic programming is used to approximate pseudorewards, which in turn shape the reward function and policy of the model-free agent.

## Experiment 1

### Methods

**Maze learning** The agent (a simple Q-learner), began each episode in the upper-left corner of a maze (Figure 3), and was rewarded one point for reaching the lower-right corner (Figure 3). The state space consisted of 121 locations in the grid shown in Figure 3, and actions consisted of each of the four cardinal directions. The agent was trained for fifty episodes, with each episode ending when the goal was

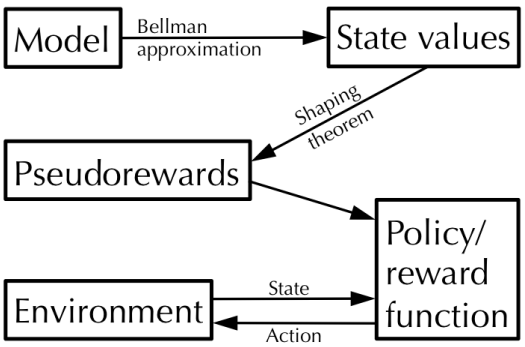


Figure 2: Schematic of the approximate pseudoreward shaping method

reached, or 2,000 steps were taken (whichever came first). An $\epsilon$-greedy decision policy was used with $\epsilon = 0.25$. The colors in Figure 3 correspond to state values under the optimal policy. Since rewards were discounted with $\gamma = 0.95$, the value of each state is $0.95^{\min stepsToGoal}$. All simulations were run ten times and averaged to produce plots.

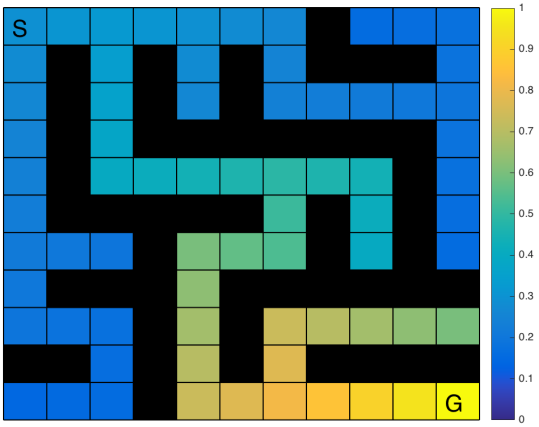


Figure 3: Maze environment. Colors correspond to state values under the optimal policy. (S=start state, G=goal state)

**Approximate pseudorewards** Dynamic programing was used to approximate state values by iterating over the Bellman equation. In [McMahan et al., 2005] the authors detail conditions under which initial state values will provably converge monotonically toward optimal values, but they note that in practice most reasonable initial values will achieve this monotonic convergence. Here, all states were initialized with a lower bound of zero and an upper bound of one, which, in our simple environment, is known to bound state values. Figure 4 shows that the approximate state values do indeed converge monotonically. The point at which each state reaches its optimal value is exactly equal to the minimum number of steps that state is away from the goal.

At each state, the pseudoreward for each action was calculated according to the shaping theorem as the difference between the value of the current state and the value of the of the next state given that action (which was deterministic in this environment). Pseudorewards are adding onto actual rewards to shape the reward function, thereby guiding the Q-learning algorithm.

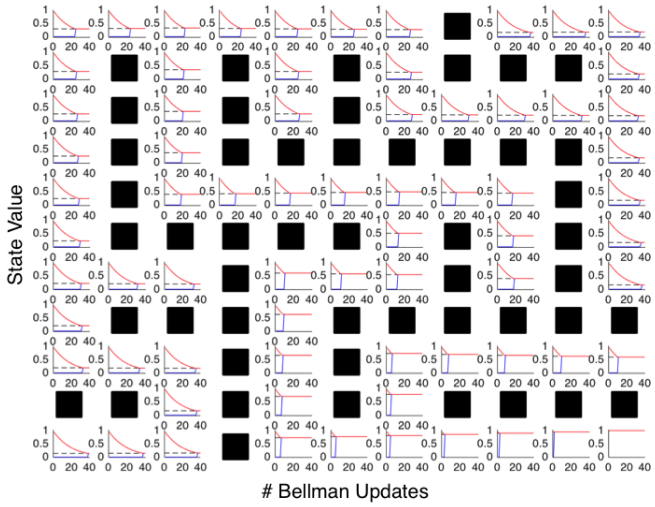


Figure 4: Monotonic convergence of estimated state values. Each subplot corresponds to a state in the maze. (Red lines=upper bound estimate, blue lines=lower bound estimate, dashed lines=optimal state values)

**Tradeoff between model-free and model-based computation** The closer pseudorewards are to their optimal values, the easier the learning for the model-free agent (at least to some precision). However, whereas Q-learning is simple and quick, the model-based method of approximating state values is relatively slow and computationally costly. Therefore, we sought to understand the most efficient tradeoff between model-based pseudoreward approximation and model-free learning. This was done by computing the amount of CPU time (in seconds) required for each algorithm (while CPU time is a variable factor, the relative time across algorithms is generally invariant).

## Results

Either the lower bound of state values or the upper bound (or the mean of the two) after $n$ iterations of Bellman updates can be used to approximate pseudorewards for Q-learning, according to the shaping theorem. Figure 5 shows the number of steps per episode needed to reach the goal, averaged across 50 episodes, as a function of the the number of Bellman updates used to approximate pseudorewards. The red line shows learning when approximate pseudorewards are based on upper bound state values, the blue line is for lower bound values, and the black line is for pseudorewards that are the mean of the two. As expected, learning is quicker when pseudorewards are closer to their optimal values. For comparison, we also show performance of the Dyna and prioritized sweeping algorithms, as a function of the number of

planning steps taken after each real step (separate green x-axis). While approximate pseudorewards are calculated just once using $n$ iterations, the planning steps used by Dyna and prioritized sweeping are taken *after every single step of every episode*. The dashed green lines show the number of real steps and the number of planning steps taken using Dyna.

Because Dyna and prioritized sweeping must learn a model of the environment through experience, the first episodes require more than the minimal number of steps to learn the goal, which is why the average across episodes is higher than the shortest path (34 steps when $\epsilon = 0.25$). With sufficiently precise pseudorewards, on the other hand, the agent can learn the shortest path on the very first episode. Specifically, 24 Bellman updates are required for this, because the start state is 24 steps away from the goal state; after 24 iterations of the Bellman equation, optimal state values have propogated back from the goal state to the start state, along the shortest path.

Also shown are the number of steps taken by a simple Q-learning agent when states values are initialized to 0 (blue asterisk), 1 (red asterisk), or 0.5 (black asterisk).

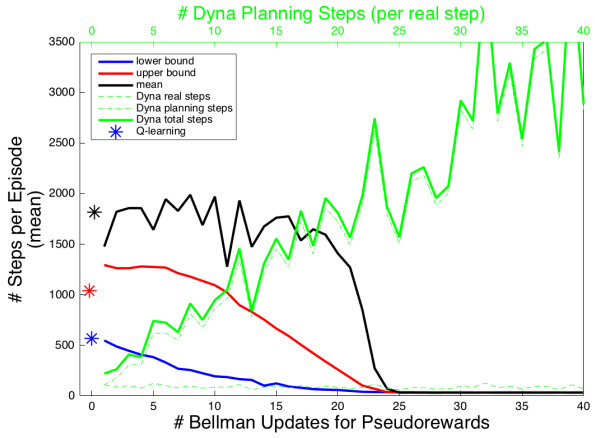


Figure 5: Pseudoreward approximation requires fewer steps to reach the goal than Dyna or prioritized sweeping. The number of steps taken to reach the goal, averaged across 50 episodes, is displayed as a function of the number of iterations of pseudoreward approximation, or the number of planning steps used by Dyna. With a sufficient number of iterations of pseudoreward approximation, the agent will reach the goal in the minimum number of steps on the very first episode. Dyna will always require some training episodes before achieving this, which is why the average is higher.

Next, we calculated the actual time (in CPU seconds) required to learn the shortest path. While the pseudoreward method may take fewer steps to reach the goal than either Dyna or prioritized sweeping, it does not necessarily mean its faster. Planning steps are about two orders of magnitude quicker than Bellman updates, so it's possible that Dyna is still faster. However, figure 6 shows that pseudoreward approximation is still faster than Dyna. The fastest learning occurs when 24 iterations of the Bellman equation are used; any more than this is unnecessary and the CPU time gradually increases linearly. Prioritized sweeping is not shown because it takes at least an order of magnitude longer to run, due to maintaining a sorted list of the most informative states.

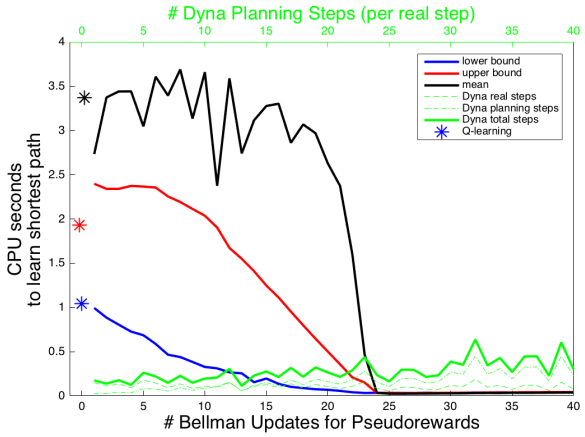


Figure 6: Pseudoreward approximation learns the shortest path more quickly than Dyna.

## Experiment 2

### Methods

**Mountain cart problem** Experiment 2 entailed learning in a classic mountain car environment. The agent begins in a valley between two mountains with the goal of reaching the star at the top of the mountain on the right (figure 7). The agent must learn to apply force such that it oscillates between the slopes of each mountain, building enough momentum to eventually reach the top. States consisted of discretized locations along the 2-D mountainsides and discretized velocities. Actions consisted of discretized forces applied tangentially to the direction of movement. The agent used Q-learning during 200 learning episodes, where each episode ended when the car reached the goal state (or if 1,000 steps were taken). For every state-action not leading to the goal, the agent was punished one point, and was awarded 100 points for reaching the goal. An $\epsilon$-greedy decision policy was used where $\epsilon = 0.01 \times 0.99^{episodeNumber-1}$. As before, all simulations were run ten times and averaged together.

**Comparison of learning methods** As before, pseudorewards were approximated using bounded dynamic programming and the shaping theorem. Performance using this algorithm was compared with Dyna.

### Results

Figure 8 shows the number of steps per episodes required to reach the goal, averaged across 200 episodes. Although not shown, the upper-bound and lower-bound estimates of state

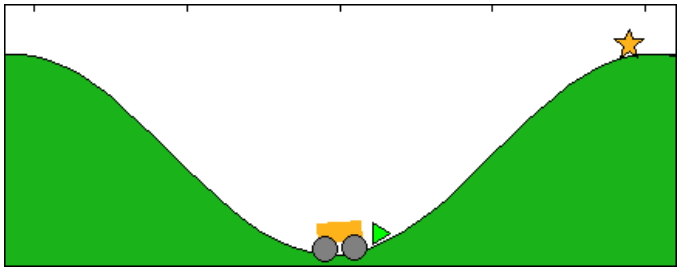

Figure 7: The mountain car problem.

values all converged to optimal values after 73 iterations of the Bellman equation. 73 is the number of steps required to reach the goal from the starting state under the optimal policy, and thus after 73 Bellman updates, values from the goal state had propagated back to the start state (in this task the start state is the furthest possible state away from the goal). The x-axis shows the number of Bellman updates used to estimate pseudorewards. As before, the blue line indicates pseudorewards based on the lower-bound estimation of state values the red line is for the upper-bound approximation of state values, and the black line is using pseudorewards based on the mean of the upper bound and lower bound state values. The green lines show the performance of Dyna. The total number of steps (real steps plus simulated steps) far exceeds the number of steps using our method, and even the number of actual steps alone does not converge as low as the number of steps taken using our method. The blue asterisk indicates performance of a simple Q-learning agent.

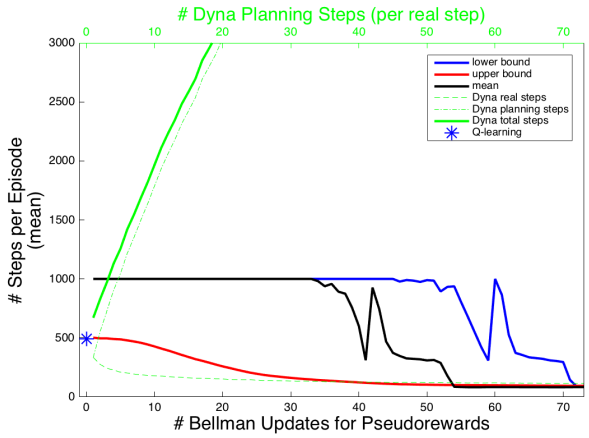


Figure 8: Performance of the pseudoreward approximation method during the mountain car problem. The average number of steps taken across episodes is plotted against the number of model-based iterations of pseudoreward approximation. Also shown (green lines) is performance of Dyna.

Figure 9 shows the amount of time in CPU seconds required to learn the shortest path. Although Dyna requires many more steps to learn, because its computations are scalar-based Q-learning approximations, it is relatively quick, whereas Bellman approximation requires more costly matrix multiplication. Still, the pseudoreward approximation method learns more quickly.

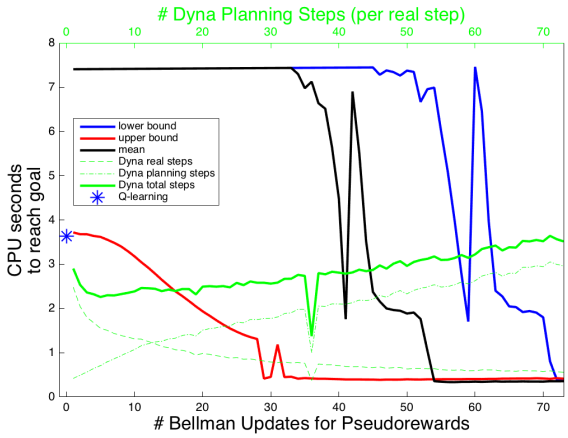


Figure 9: The amount of CPU time required to learn the shortest path.

## Discussion

We have introduced a new method for cooperatively integrating model-free and model-based reinforcement learning. This method relies on bounded dynamic programming as a model-based method to iteratively estimate state values that converge monotonically to values under the optimal policy. These approximate values are used to calculate pseudorewards according to the shaping theorem, such that the reward function is altered but the optimal policy is invariant. This reward function is used for model-free learning. Our experiments demonstrate that this method performs comparably and even better than the Dyna algorithm, which is another cooperative reinforcement learning method.

One important difference between our method and Dyna is that Dyna learns the model of the environment, whereas our model is omniscient (that is, it is given the full state-action transition matrix and reward-action pairs). This may at first make any comparison between the two methods in terms of learning or computation time unfair. In actuality, however, if Dyna is given a full model so that it is equally omniscient, learning and and computation performance goes a tiny bit down in the maze environment, and improves marginally in the mountain car task. An omniscience Dyna agent will only learning more quickly during the first episode, before it has discovered the location of reward. Once it knows how to reach the goal, a full model is unnecessary, and would even slow learning through planning, because unvisited states that do not help the agent reach the goal will be replayed. One modification that would save computation time for Dyna would be to modulate the number of planning steps in proportion to the change in

variance of estimated Q-values. When the temporal difference errors on average are larger, more planning steps are needed, but as they converge, the number of planning steps would go down. Another improvement to Dyna, which has been studied, is known as prioritized sweeping [**?**]. Here, the replay of state-action pairs is selected from the top of a que, where the que is sorted by the temporal difference error from that state action pair. This allows states with the most uncertainty about their value to be replayed the most, and the effect of this is that learning of state values propagates backwards from rewarding states. If a prioritized sweeping algorithm were omniscient in the sense described above, then the number of planning steps needed to compute Q-values that would allow the agent to follow an optimal policy would simply be the distance from the start state to the goal state. While prioritized sweeping would require fewer steps (real and simulated) than a Dyna to learn the shortest path, the sorting the que after every simulated step, as described above, is extremely costly in terms of CPU time, and therefore takes much longer to compute in practice.

While our method is omniscient with respect to state-action transitions and rewards, this need not be the case. Our method can also be initialized with a naive model that has a uniform prior for all transition probabilities and reward outcomes (that is, any action is assumed to transition to any other state with equal probability and the expected reward for any action is $R/(numberofstates)$, where $R$ is the expected reward for (optimally) performing the task). This model can be used for state value approximation with bounded RTDP, just as before, and as experience is acquired, the model can be updated (with either probabilistic transition estimates or deterministic ones if the environment is assumed to be such). When our method is run this way, however, the change in performance in our environments is negligible. When the model is initialized with a uniform prior on rewards, the expected reward for any given action is relatively small. As a result, as soon as the agent discovers (relatively large) actual reward, this has a much bigger impact on estimated state values.

By providing a new way to link model-free and model-based reinforcement learning, our method offers a new way to think about human cognition, and potentially test experimental. While experimental findings on cooperative reinforcement learning in humans is nascent, there is substantial interest in understanding the interactions between them [**?**]. As discussed earlier, Dyna is readily likened to memory replay (including planning) in human cognition as a means to train a model-free system. What might be an analogue of pseudoreward approximation in human cognition? For any given task or goal-directed behavior, emotions quite often have the effect of altering the reward landscape, and it is reasonable to think of them as pseudorewards when the goal itself is something external or independent of the emotion. If certain emotions represent the values of states that are stored in a model, and these emotions are used to train model-free learning by adding bonuses (positive emotions) or punishments (negative emotions) to certain actions, this would be quite akin to our method. The accuracy with which the emotion represents the value of a state would depend on the accuracy of the model, and could be implemented using something similar to Bellman approximation. In the introduction, we used the example of the feeling of remorse based on a model of ethics as a means to train the habitual learning system to avoid such future actions. It is worth pursuing these questions experimentally to test the utility of our method for understanding human cognition.

## References


[Barto et al., 1995] Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138.

[Daw et al., 2005] Daw, N. D., Niv, Y., and Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8(12):1704–1711.

[Dayan and Berridge, 2014] Dayan, P. and Berridge, K. C. (2014). Model-based and model-free pavlovian reward learning: revaluation, revision, and revelation. *Cognitive, Affective, & Behavioral Neuroscience*, 14(2):473–492.

[Gershman et al., 2014] Gershman, S. J., Markman, A. B., and Otto, A. R. (2014). Retrospective revaluation in sequential decision making: A tale of two systems. *Journal of Experimental Psychology: General*, 143(1):182.

[Gläscher et al., 2010] Gläscher, J., Daw, N., Dayan, P., and O'Doherty, J. P. (2010). States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 66(4):585–595.

[Lansink et al., 2009] Lansink, C. S., Goltstein, P. M., Lankelma, J. V., McNaughton, B. L., and Pennartz, C. M. (2009). Hippocampus leads ventral striatum in replay of place-reward information. *PLoS Biol*, 7(8):e1000173.

[McGonigal, 2011] McGonigal, J. (2011). *Reality is broken: Why games make us better and how they can change the world*. Penguin.

[McMahan et al., 2005] McMahan, H. B., Likhachev, M., and Gordon, G. J. (2005). Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd international conference on Machine learning*, pages 569–576. ACM.

[Ng et al., 1999] Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

[Sutton, 1991] Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163.

[Thorndike, 1933] Thorndike, E. L. (1933). A proof of the law of effect. *Science*.

[Tolman, 1948] Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological review*, 55(4):189.