Balancing Exploration And Exploitation in Forward Model Learning

Alexander Dockhorn and Rudolf Kruse

Abstract Forward model learning algorithms enable the application of simulation-based search methods in environments for which the forward model is unknown. Multiple studies have shown great performance in game-related and motion control applications. In these, forward model learning agents often required less training time while achieving a similar performance than state-of-the-art reinforcement learning methods. However, several problems can emerge when replacing the environment's true model with a learned approximation. While the true forward model allows the accurate prediction of future time-steps, a learned forward model may always be inaccurate in its prediction. These inaccuracies become problematic when planning long action sequences since the confidence in predicted time-steps reduces with increasing depth of the simulation. In this work, we explore methods for balancing risk and reward in decision-making using inaccurate forward models. Therefore, we propose methods for measuring the variance of a forward model and the confidence in the predicted outcome of planned action sequences. Based on these metrics, we define methods for learning and using forward models under consideration of their current prediction accuracy. Proposed methods have been tested in various motion control tasks of the Open AI Gym framework. Results show that the information on the model's accuracy can be used to increase the efficiency of the agent's training and the agent's performance during evaluation.

Alexander Dockhorn
Queen Mary University, London, UK e-mail: a.dockhorn@qmul.ac.uk

Rudolf Kruse
Otto von Guericke University, Magdeburg, Germany e-mail: rudolf.kruse@ovgu.de

1 Introduction

Forward Model Learning describes the process of learning a model of *a priori* unknown environments. This enables an agent to anticipate the outcome of planned action sequences and use simulation-based search techniques to optimize its behavior.

This chapter builds upon our recent work on forward model learning in games [1] and directly extends our work on forward model learning for motion control tasks [12]. Previous studies have shown that reliable forward models can be learned by observation. Since no information on the environment is available, we have chosen to implement an uninformed training process in which the agent uses random actions to explore its environment. This results in many interactions being wasted due to (1) repeating a known interaction and (2) neither focusing on improving the model nor improving the agent's performance. As a result, we have observed that the model may be unreliable since large amounts of the state space remain unobserved. In return, the agent's performance during the evaluation will be limited by the prediction accuracy of the trained forward model.

Motivated by these shortcomings, we study methods for improving the efficiency of the agent's training process. Therefore, we formulate two learning goals (1) the exploration of the environment to train a reliable forward model, and (2) the exploitation of promising action sequences to become more proficient in the given task. The main contribution of this work can be summarized by:

- **Taxonomy of Learning Algorithms:** We present a unifying view on reinforcement learning, search, and forward model learning algorithms in the context of the agent-environment interface.
- **Decomposed Differential Forward Model:** We review the definitions of previously proposed model building heuristics and their generalization in form of the decomposed forward model. As a result, we propose the decomposed differential forward model which will be used throughout this study.
- **Risk Awareness:** We propose several methods for measuring the agent's confidence in the learned forward model's predictions. Furthermore, we propose several optimization goals to balance exploration and exploitation during the agent's training process.

The remainder of this chapter is structured as follows: In Section 2, we compare the concepts of reinforcement learning, search-based algorithms, and forward model learning for decision-making. The following sections (Section 3 and Section 4) focuse on the introduction of forward model learning and types of forward model representations. Section 5 exemplarily shows the usage of Gaussian process regression and ensemble regression models such as a random forest regression. For both, we propose methods for incorporating the variance of made predictions in the agent's decision-making process (Section 5.2). We evaluate proposed methods based on their resulting performance in three simple motion control tasks in Section 6. For a more detailed analysis, we compare the impact of proposed measures on the agent's training process in Section 7. We conclude our results in Section 8 and discuss opportunities for further studies.

2 Taxonomy of Learning Algorithms

The agent-environment interface represents a general description of a learning scenario. Here, the agent is in continuous interaction with an environment. Each of the agent's executed actions has the potential to update the state of the environment. In return, the agent can be offered a numerical reward that is typically associated with the given task.

In reinforcement learning, the agent focuses on picking actions to maximize its expected reward over time. Given observations of all its previous interactions, the agent tries to estimate the expected value of state-action pairs. Algorithms such as Temporal Difference Learning (TDL) [32], Q-learning [35], and the Monte Carlo (MC) method [30], update the expected value after a reward signal has been observed or after the result of an episode has been observed respectively. These techniques are called model-free since they do not try to build a model of their environment.

In contrast to these simpler reinforcement learning algorithms, which are storing the value of each state-action pair, methods in deep reinforcement learning use a neural network to approximate the value based on the input, thus drastically decreasing the storage used for the model [37]. While deep reinforcement learning algorithms have resulted in an impressive performance in the context of many game-related benchmarks, the number of required training examples to fit all the network's parameters is often high and thus require much training time.

Simulation-based search algorithms, such as minimax [23] or Monte Carlo tree search (MCTS) [5], utilize the environment's forward model to simulate the outcome of planned action sequences. Each action sequence represents a candidate solution. Its simulation is called a rollout and the result of such a simulation can be used to estimate the value of actions in the simulated sequence. In contrast to reinforcement learning algorithms, simulation-based search algorithms estimate the value of an action at run-time. Therefore, these algorithms require knowledge of the current state and the game's model, and return the action with the highest expected value regarding the current state. In return, they can be applied without prior training. Due to their capability of being applied without training, search algorithms such as MCTS and Open Loop Search [26] have performed well in general game-playing tasks [27].

Dynamic programming algorithms [21] require knowledge of the environment's forward model to calculate the true value of any state-action pairs. This optimization process can yield a perfect policy for small state and action spaces. However, the high breadth or depth of the search tree often renders this method infeasible.

Next to traditional search schemes, the rolling horizon evolutionary algorithm (RHEA) [15, 16], uses mutation and crossover to optimize the agent's action sequence. A heuristic value of simulated action sequences can be used as a fitness measure to guide evolutionary optimization. The design of such a heuristic can drastically impact, the resulting behavior.

In the following, we compare the reviewed methods based on their knowledge of the environment. Given the agent's action and the current state, the agent will observe the next state and a reward. Both updates can be encoded in a separate model, namely the forward model producing the next state and the reward model which in

turn provides the agent with a reward. However, since the accumulated reward (also known as return) is much more useful for the agent's action-selection process, the following comparison will use the agent's knowledge of the return to measure the value of a state-action pair. Therefore, we focus on the agent's knowledge of two of the environment's components, (1) the environment's forward model, and (2) the value of the environment's states and actions.

Considering the agent's knowledge of the return, reinforcement learning and simulation-based search methods represent two extremes. The eager learning process of reinforcement learning methods results in a return model. While TDL and the MC method do this without needing knowledge about the forward model, the computations done in dynamic programming need the forward model for its iterative update routine. Deep reinforcement learning replaces the need for storing the expected return for every state by learning a network that approximates the return function. The reduction in complexity can arguably be achieved through an understanding of the game's state-space.

In contrast, simulation-based search methods do not need to store the expected return of each state, since they approximate at run-time using the forward model. The rolling horizon evolutionary algorithm seems to be a slight exception to this since it is also making use of a heuristic function, which approximates the value of a state. This heuristic function is being used to rate a rollout's outcome and, in the best case, approximates the expected return.

The class of forward model learning algorithms represents a new approach that enables the application of search algorithms in case the environment's forward model is unknown to the agent. Therefore, the agent builds a model to predict upcoming states based on made observation during previous interactions. At the time, the agent can also learn to predict the reward of the environment or approximate the value of a state-action pair.

This process can be compared to recent experiments on world modeling [18] and imagination-based deep reinforcement learning [36]. In these, the agent uses a deep or recurrent neural network structure that handles the selection of the agent's actions according to predicted future states. Agents using these model-based deep reinforcement learning approaches have shown to be capable of playing games based on their visual state representation [20]. Similarly, they achieved improved performance in several visual control tasks in comparison to model-free reinforcement learning agents [19]. However, the sheer number of parameters to be tuned and the eager learning of the state's value function results in the agent requiring lots of training data, e.g. 10^8 training steps for learning to play the game Sokoban [36]. The amount of required iterations makes this process infeasible in case the model's training time is limited.

In contrast, a prediction-based search can be implemented which determines an action's value according to simulations of the trained forward model. Furthermore, reductions of the feasible model space could be achieved by assuming independencies among observed sensor values. Figure 1 presents a summary of the discussed methods, based on the two dimensions: knowledge of expected return and knowledge of the game model.

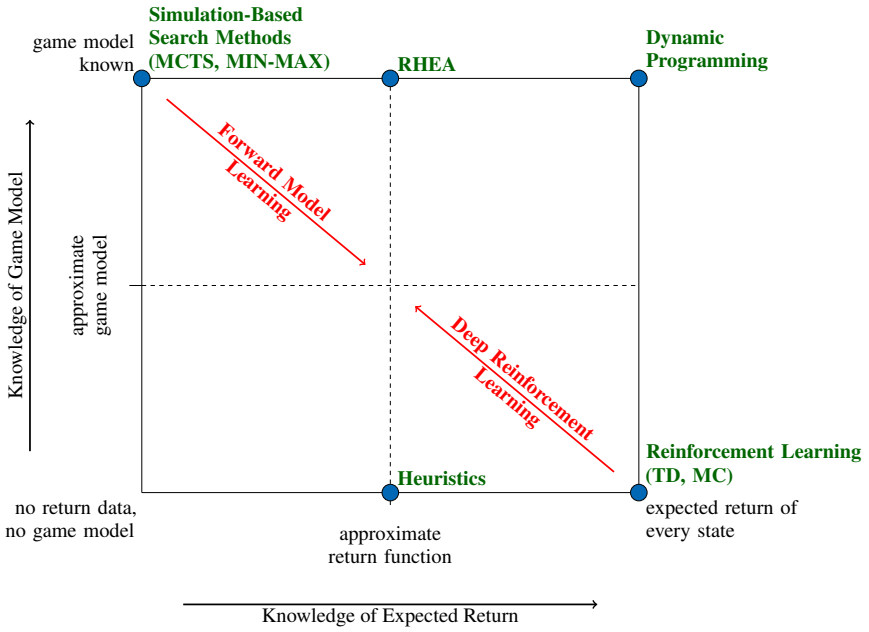


Fig. 1 Comparison of general game-playing and -learning techniques based on the agent's knowledge of the return function and the game's environment model [8].

3 Forward Model Learning

In this section, we will take a closer look at the forward model learning process. First, we will define how the environment's model can be represented. The next section will cover the forward model's representation.

First of all, we assume that the environment model can be represented as a stochastic process that maps the sequence of previous actions $a \in A$ and states $s \in S$ to a probability of observing the next state and its associated reward $r \in \mathbb{R}$.

$$p(s_t, r_t \mid s_{t-1}, a_{t-1}, \ldots, s_1, a_1) \tag{1}$$

We consider each state $s_t \in S \subseteq \mathbb{R}^n$ to be a vector of real-valued observations $s_t = (s_t^1, s_i^2, \ldots, s_t^n), s_t^i \in \mathbb{R}$ at time t. The same notation applies to the set of available actions and the action space: $a_t \in A \subseteq \mathbb{R}^n$.

For simplicity, we will assume single-value action spaces and an environment model that satisfies the Markov property. Therefore, the environment's response is independent of all but the latest interaction and can be simplified to:

$$p(s_t, r_t | s_{t-1}, a_{t-1}) \tag{2}$$

Up to now, the environment's response includes two components, the state and the reward. We further split the environment model into a state transition model and a reward model, both fulfilling the Markov property:

$$\begin{aligned} \text{State Transition Model:} \quad & p_{fm}(s_t \mid s_{t-1}, a_{t-1}) \\ \text{Reward Model:} \quad & p_r(r_t \mid s_{t-1}, a_{t-1}) \end{aligned}$$

During forward model learning, the agent will learn to approximate the state transition model based on observations of previous interactions. Given these observations, the agent can construct a training set and use supervised learning methods to create an approximation of the original model. Depending on the structure of the state-space a classification (discrete and nominal state-spaces) or a regression model (real-valued state spaces) can be used to predict upcoming states.

In recent works of model-based reinforcement learning, the state transition model has been represented by deep neural networks. The *World Models* agent learns a latent vector representation using LSTMs [18]. This allows the agent to keep track of previous events and further condense its action policy. Following up on this idea, the imagination-based *I2A* agent [28] adds a rollout phase for improving the agent's estimates of an action's expected return for long-time horizons. The *Dreamer* agent [19] has shown that latent state models can learn motion control behaviors of varying complexity based on high-dimensional sensory inputs.

In contrast to model-based reinforcement learning, forward model learning agents do not model the return of an action (accumulated reward) but estimate an action's value based on its simulated outcome. State-of-the-art planners such as Monte Carlo Tree Search [5] and Rolling Horizon Algorithms [15] have recently shown applications in game AI [15, 31] and motion control [6]. Since the accuracy of the planning process is dependent on the accuracy of the agent's prediction, the agent's goal is to improve the forward model's accuracy and not the agent's policy.

4 Forward Model Representation

Learning a model of the environment has been actively studied in model-based reinforcement learning. While many approaches rely on the generality of deep neural networks, they have not shown to be very sample efficient. Therefore, the model requires many iterations to be trained until a reliable prediction of the upcoming state can be made and the search-methods advantage to work without much training time is lost. Thus, finding a suitable state representation can drastically reduce the required training time as well as improve the model's final prediction accuracy.

Instead of predicting the environment's next state using a single model, the environment's response can often be split into multiple independent components. In return, we can decompose the forward model into independent sub-models. The reduced complexity of a sub-model's output space can result in a simpler learning task, hence, reducing the number of required model parameters and training examples.

4.1 Model Building Heuristics

Forward model decomposition has been shown to increase the sampling efficiency and resulting model performance in numerous applications. An example of a model decomposition heuristics is the local transition function that is used in local forward models ([8]). Local forward models work similar to convolution neural networks, and predict the future state based on independent predictions of each cell, whereas the future state of each cell is predicted based on its current state and the state of its neighbors. Local forward models have been successfully applied to the game of life ([22]) and Sokoban ([13]) as well as several games of the GVGAI environment ([1, 8]).

In ([14]), the authors have shown that knowledge of the environment's state representation can be used to create an object-based decomposition. An extension of this work has been presented in ([11]), in which the authors proposed the use of stochastic independence tests to find a valid decomposition of the forward model without requiring background knowledge of the environment's state representation.

The local and object-based forward model assumes independency among the input and output variables of the forward model. Such dependencies among observable sensor variables may exist in motion-control tasks, but the underlying structure may be unknown. As a result, we have previously proposed the decomposed differential forward model [12]. For the decomposition, we assume that the next state's sensor values are dependent on the previous state (and possibly its predecessors), but independent of each other. This results in a decomposability of the state transition model into several sub-models, whereas each sub-model predicts the updated sensor value:

i-th Component Model: $fm_i(s_{t-1}, a_{t-1}) \mapsto s_{i,t}$

The decomposed forward model can further be used to predict the future state by separately predicting the future state of each observable sensor-value (indicated by $\hat{s}_{i,t}$) and aggregating the result of each sub-model:

$$\begin{aligned} & fm(s_{t-1},\ a_{t-1}) \\ =& (fm_1(s_{t-1},\ a_{t-1}),\ \ldots,\ fm_n(s_{t-1},\ a_{t-1})) \\ =& (\hat{s}_{1,t},\ \ldots,\ \hat{s}_{n,t}) \end{aligned} \tag{3}$$

Alternatively, we can predict the changes of a sensor value:

i-th Differential Model: $fm_{\Delta i}(s_{t-1},\ a_{t-1}) \mapsto s_{i,t-1} - s_{i,t}$

Aggregating the predictions of all sensor values and the agent's reward, the agent can predict the outcome of an action. Similarly, a hierarchical structure can be built, in which the environment is first split into several units and the future state values of each unit are predicted independently [7, 9].

Since the result is another state observation, the agent can predict whole action sequences by repeatably applying the learned forward model. Using this, actions can be determined using forward planning methods or a policy can be learned using reinforcement learning on the simulated environment [25].

For motion control, the underlying modeling task is considered a supervised learning problem in which a regressor is trained to predict upcoming states. Several attributes of the model need to be considered upon model selection:

- **model accuracy:** the trained model needs to be accurate for previous observations and future time steps. This is especially relevant for consecutive predictions since the error will propagate over multiple predictions.
- **model speed:** the trained model needs to be applied very often during the search process. Studies on MCTS have shown that increasing the number or the quality of rollouts can improve the agent's performance [10].
- **model size:** the model's size (in terms of parameter count) can impact the training time and the number of observations required for optimizing the model's parameters. While regression models such as linear regression are the most simple to train, their applicability is quite limited. In contrast, deep neural networks are flexible but can require large amounts of training data.
- **model interpretability and reliability:** a characteristic that is often neglected in deep learning approaches is the model's interpretability. While deep reinforcement learning has shown great performance, the black-box nature of deep neural networks may not allow human interpretation of its results. This complicates to measure the reliability or risk of a trained model. In contrast, planning based approaches can transparently summarize the search path and its predicted states. This can be especially important in risk critical applications and allow the computation of confidence bounds.

5 Improving the Confidence of a Forward Model

The forward model allows us to anticipate the result of an agent's actions. However, the prediction may be inaccurate and therefore the predicted result of an action sequence may be desirable but unlikely. In the context of a simulation-based search agent which is using a learned forward model, we would like to incorporate the confidence in the model's prediction in the agent's decision-making process as well as improving it throughout the training process.

5.1 Measuring the Learned Model's Confidence

The following analysis is based on the Gaussian Process regression model [29]. In contrast to many other regression models, it does not return a single predicted value, but a full predictive distribution. Given this distribution, we can estimate the mean value and the distribution's variance to be used by the agent. Other regression models may not provide the same information but it may be inferred given their output.

In the case of ensemble regression models, we can measure the model's confidence by the agreement of all regression models included in the ensemble. Hence, the

ensemble does not need to consist of probabilistic regression models but may be built using any type of regression model. A common representative is the random forest regression [3], which consists of multiple decision tree regression models each trained on a different slice of the training data set. The ensembles response will be the mean of predicted values. Similarly, we can measure the variance of made predictions to estimate the confidence of the ensemble model.

5.2 Learning Goals Based on the Model's Confidence

In previous studies, the training process was implemented as a random exploration [13, 12]. As a result, we have observed that the forward model's accuracy quickly degrades in situations that have not previously been observed. This is due to the missing diversity in the training set. Therefore, we want to actively approach states and actions that the forward model cannot accurately predict to improve its generality.

Given the task of predicting action sequences of length m, we will use a decomposed forward model to predict the upcoming states after each action of the action sequence. Starting with a state $s_t = (s_{1,t}, \ldots, s_{n,t})$ we will predict the future states s_{t+1} to s_{t+m}. Similarly, we will learn to predict the rewards r_{t+1} to r_{t+n} the agent expects to receive, which can be modeled as an additional output of the environment's state observation. For simplicity, we will use s and r to denote the true state and reward, and $\hat{s}$ and $\hat{r}$ for the predicted state and reward respectively. The following heuristics will be used to guide the agent's action selection during training and evaluation.

First, we focus on improving the agent's performance by selecting actions that are predicted to be promising by the forward model. Naively, we can assume that we only need to take the sum of predicted rewards of an action sequence into account. Hence, the predicted discounted return Q of the agent's planned action sequence results in the weighted sum of rewards:

$$Q = \sum_{i=1}^{m} \gamma^i \hat{r}_{t+i} \tag{4}$$

for which $\gamma \in (0, 1]$ balances between favoring long-term and short-term rewards.

As we pointed out before, the regression model might be inaccurate in its prediction. Next, we want to incorporate information on the predicted reward distribution to make the agent aware of such inaccuracies. Hence, during training, we want to actively approach actions the current model cannot confidently predict. To motivate this behavior, we let the agent actively approach situations with a high variance. An optimistic return heuristic can be formulated as:

$$Q_{opt} = \sum_{i=1}^{m} \gamma^i \hat{r}_{t+i} + \sum_{i=1}^{m} \sigma^2_{\hat{r},t+i} = \sum_{i=1}^{m} \left[\gamma^i \hat{r}_{t+i} + \sigma^2_{\hat{r},t+i}\right] \tag{5}$$

During the evaluation, we want to avoid such situations to minimize the agent's risk. Taking a pessimistic (Q_{pess}) approach, we reduce the expected reward by the variance of its predictions.

$$Q_{pess} = \sum_{i=1}^{m} \gamma^i \hat{r}_{t+i} - \sum_{i=1}^{m} \sigma^2_{\hat{r},t+i} = \sum_{i=1}^{m} \left[\gamma^i \hat{r}_{t+i} - \sigma^2_{\hat{r},t+i} \right] \tag{6}$$

Thus we can assure that two action sequences with similar outcome but varying confidence in its predictions can be differentiated.

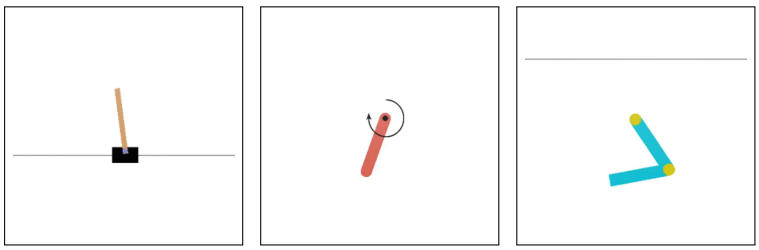
So far, both heuristics have focused on the predicted reward. To assure that the model will also become more confident in its prediction of future states, we introduce an exploration goal. For this purpose, we will use the information on the model's confidence in predicting upcoming state variables to guide the agent into states that cannot be accurately predicted yet. Once again, we measure the model's confidence by taking the variance of the prediction into account. Since the predicted states are dependent on each other, we would require additional simulations to identify the reward at each time-step of the predicted action sequence. This process would be infeasible for real-time applications because the number of forward model evaluations is limited. Instead, we will use the cumulative standard deviations to approximate the uncertainty of a whole action sequence:

$$\sum_{i=1}^{m} \sum_{j=1}^{i} \sigma^2_{\hat{s}_{t+j}}$$

Hereby, we punish action sequences in which the first actions cannot be accurately predicted more than action sequences in which the results of later actions are unsure. We further incorporate the defined optimization targets into a single heuristic for action-selection.

$$Q_{conf} = \sum_{i=1}^{m} \Big[\gamma^i \hat{r}_{t+i} + \alpha \sigma^2_{\hat{r}} + \beta \underbrace{\sum_{j=1}^{i} \sigma^2_{\hat{s}_{t+j}}}_{\substack{\text{cumulative} \\ \text{state variance}}} \Big] \tag{7}$$

for which α balances the agent's risk awareness considering the reward predictions and β balances the agent's drive to explore unknown states. This heuristic has been motivated by Bienaymé's identity but since the independency of variables cannot be assured here, we included α and β to configure the heuristic according to the user's requirements. Setting $\alpha > 0$ sets the agent to actively approach states in which the predicted reward is unsure, whereas values below 0 let the agent avoid those states. For $\beta > 0$ the agent is encouraged to explore action sequences the model cannot confidently predict and once again lets the avoid those states for values below 0. For training in a safe environment, we recommend setting both values to be positive, while during testing we recommend choosing negative values to let the agent explore its environment. Those values may be adapted throughout the training depending on the confidence of the model and the agent's success.



(a) Cart-Pole (b) Pendulum (c) Acrobot

Fig. 2 Three motion-control environments and their representation in the OpenAI Gym framework.

6 Evaluating the Agent's Performance

Our evaluation is split into two parts. First, we analyze the agent's ability to learn a forward model and using it for guiding its action-selection in three of the most common motion control tasks. During this experiment, we will compare the agent with reinforcement learning-based approaches. To evaluate the proposed measures, we designed a second experiment series, to test their effects on the agent's training and its resulting performance in the aforementioned motion control tasks. The source-code to our experiments and our results are available for download[1].

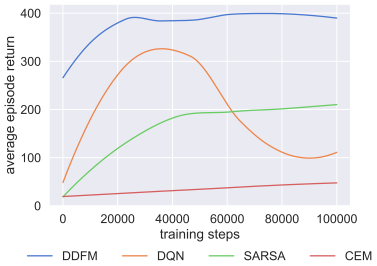
6.1 Motion Control Environments

The *cart-pole* problem [2] requires the agent to balance a pole by moving a cart back and forth. The pole is attached to a cart by an un-actuated joint, which moves along a frictionless track. The agent can apply a discrete force of $+1$ or -1 to the cart. The pole is starting in an upright position and needs to be kept in the range of $[-15, +15]$ degree from vertical. If the pole is falling below that threshold or the cart moves more than 2.4 units from the center, the episode ends. A maximal reward is achieved after balancing the pole for a maximum of 500 time-steps.

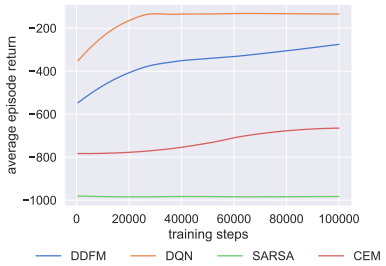
The *pendulum problem* is a classic problem in the control literature. In this version of the problem, the pendulum starts in a random position, and the goal is to swing it up so it stays upright. The reward penalizes deviations from the equilibrium and the magnitude of the agent's applied actions.

The *acrobot problem* requires the agent to swing up a pendulum with two links. Similar to the pendulum problem, the agent can apply a discrete rotational force to the joint of the first link. Initially, both links hang downwards, and the goal is to swing the end of the lower link up to a given height indicated by the target line.

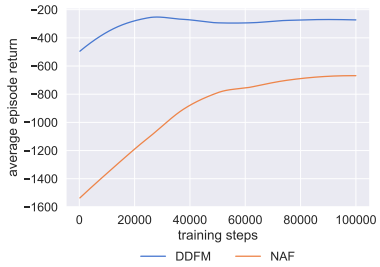
[1] https://github.com/ADockhorn/Balancing-Exploration-And-Exploitation-in-Forward-Model-Learning



(a) Cart-Pole



(b) Acrobot



(c) Pendulum

Fig. 3 Training comparison of forward model learning and deep reinforcement learning agents. Graphs are showing the average episode return per training step smoothed using the local regression (loess). Each agent has been trained 10 times for 100000 steps each.

The reward is the height of the tip of the pendulum. The implementation of these environments is provided by the Open AI Gym framework [4].

6.2 Experiment Setup - Agent Performance

The proposed decomposed differential forward model (DDFM) will be compared to several popular reinforcement learning algorithms. In case of a discrete action space (Cart-Pole and Acrobot) we chose to use Sarsa [33], DQN [24], and CEM [34]. Whereas for the Pendulum environment, which uses a continuous action space, we compared our approach with the NAF algorithm [17]. The hyper-parameters of each algorithm were tuned by a simple grid-search.

For each environment, we continuously train our agents for a total of 100000 time-steps. During training, we record the agents' reward per episode. We repeat the training process 10 times to get more stable results. The final performance of each model is measured by the average return of the last 10 training episodes.

6.3 Results

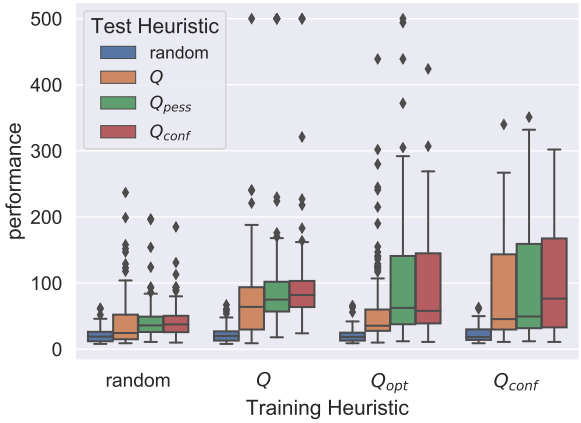
Figure 3 shows the agents' performance in each of the three training environments. The results indicate that simple problems, such as the Cart Pole and the Pendulum environment, can efficiently be solved after just a few training steps. In these, the agent outperforms deep-reinforcement learning approaches in terms of sampling efficiency. Nevertheless, the proposed model performs worse in the Acrobot environment, since solving the task requires to plan longer action sequences. Due to the simplicity of all three environments, the learning curves are steep and converge quickly. The convergence level seems to be very much dependent on the agent's parameters, for which the planning horizon and the regression model have shown to have the most impact.

6.4 Discussion

Based on this evaluation and our preceding parameter optimization we have identified two problems of current forward model learning techniques.

Planning horizon dilemma: A common problem of planning algorithms is the choice of the planning horizon. While increasing the search can improve the accuracy of a planning agent's reward estimation, it also exponentially increases the number of possible states to be analyzed. In the case of forward model learning agents, the planning horizon is further restricted by the accuracy of the learned forward model. In contrast to using the true environment dynamics, predictions of a learned forward model become less accurate with increasing search depth. For choosing a suitable rollout length for a trained model we analyzed the model's prediction error over the length of multiple predicted action sequences. This can hint at a suitable parameter combination but has shown to be too costly to repeat throughout the model learning process. Dynamic measurements of the model's confidence in its prediction might help in improving the agent's performance. Similarly, algorithms like MCTS that do not use a fixed planning horizon may be beneficial.

Exploration vs. Exploitation: During training, the agent needs to optimize the learned model as well as possible but also needs to focus on beneficial actions. Therefore, the agent needs to focus on actions that maximize our chances of succeeding, while adding new data to the forward model for improving its accuracy. In the present study, we let the agent solely decide based on its predicted reward. However, it may show beneficial to include exploring actions during training. This may not just improve the model's accuracy, but in the long run, also make the agent more robust to unlikely or new situations.



(a) Cart-Pole

Fig. 4 Testing all combinations of train and test functions. Training has been done for 500 time-steps of the environment. For each trained model we performed 10 evaluations and measured the average length the agent kept the pole balanced. The experiment has been repeated 10 times to average the results of multiple training runs.

7 Evaluating the Effects of Confidence-based Sampling

7.1 Experiment Setup - Training Efficiency

These observed problems of the previous evaluation have motivated the proposal of confidence-based sampling and action-selection during the agent's training and evaluation. To compare the effects of proposed methods we repeat the agent's training process using a (1) a random-exploration, (2) the undiscounted return (Equation 4, $\gamma = 1$), (3) the optimistic return (Equation 5), and (4) the confidence-based return (Equation 7) with weights $\alpha = \beta = 1$. Each training function is used to train 10 models for 500 time-steps each. During training, we update the agent's forward model every 10 time-steps. The forward model will be represented as a differential decomposed forward model using Gaussian process regression with a radial basis function kernel.

After 500 time-steps of training, we store the agent's forward model and keep it constant for the remaining evaluation. Furthermore, we test the learned model's for 10 episodes using (1) a random agent, (2) the discounted return (Equation 4, $\gamma = 1$), (3) the pessimistic return (Equation 5), and (4) the confidence-based return (Equation 7) with weights $\alpha = \beta = -1$. For a reliable action-selection, we simulate all action-sequences of length x and select the first action of the best-rated action sequence. In case multiple action sequences are predicted to yield the same value, we choose a random action sequence among the best action sequences.

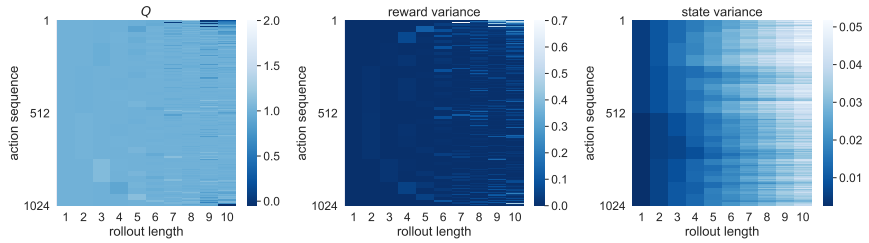


Fig. 5 Simulated action sequences an their predicted value given the proposed sampling methods.

7.2 Results

Figure 4 shows the results for each combination of train and test action-selection methods on the cart-pole environment. The box-plots show that the best performance is achieved by using the confidence-based return during the agent's training and evaluation. Even if the agent has been trained using random interactions, the agent can improve its performance using any of the proposed functions for action-selection.

The latter is especially interesting since it suggests that confidence-based sampling can benefit the action-selection of all existing applications of forward model learning without retraining the model. It allows the agent to easily balance between exploration and exploitation of the state-space by making it aware of action sequences that cannot be correctly predicted. This comes at the cost of estimating the prediction variance. Depending on the used regression method, this might result in just a small overhead, as we have demonstrated for ensemble regression models. Nevertheless, other regression models may require more work.

In our evaluation, we have only simulated action sequences of equal length. The application of confidence-based measures may allow using adaptive planning horizon by stopping the simulation as soon as the cumulative state confidence falls below a set threshold. This can be especially interesting for tree-search-based algorithms, such as MCTS, to cut off the current rollout in case the forward model cannot provide accurate predictions anymore.

To get a better picture of the effects of confidence-based action selection, we have devised a final test. In this experiment we have used a single state of the cart-pole environment, a trained model, and all simulated action sequences of length 5. We further used the (1) undiscounted return (Equation 4, $\gamma = 1$), (2) the pessimistic return (Equation 5), and (3) the confidence-based return (Equation 7) with weights $\alpha = \beta = -1$ to evaluate all the action sequences.

Figure 5 show the respective values for all steps of all simulated action sequences. Looking at the values provided by the return shows that the agent can hardly differentiate action sequences based on their value. This is caused by the uniform reward of the environment, which always awards the agent 1 point in case the pole did not tip over yet. The model easily learns this reward scheme and correctly represents the reward distribution. Since the length of simulated action-sequences does in many

cases not suffice to let the pole fall down, the reward is often not a sufficient signal to choose actions for minimizing the agent's risk. Based on the value of the pessimistic return, we see that the time point at which the reward stops is unsure, but as soon as the forward model has made a decision, the variance in its prediction returns to 0. Finally, the confidence-based return allows differentiating most action sequences based on their cumulative state variance. Returned values provide the agent with the most information on the forward model's prediction. Especially action sequences which the model does not predict with high confidence can easily be avoided.

8 Conclusion and Future Work

In this work, we have analyzed forward model learning algorithms and how they fit into the big picture of algorithms for computational intelligence in games and motion control. Based on our recent work on forward model learning, we have proposed the decomposed differential forward model. In its most general form, this model assumes that the environment's observable sensor values are independent of each other. This restriction can be loosened in case enough training data is available, such that the dependency structure can be inferred.

The model has further been tested in three classic motion control environments. Results have been compared to deep-reinforcement learning approaches. Our first experiments have shown that the performance is largely dependent on the accuracy of the learned forward model and its feasible prediction horizon. To further improve upon this, we have proposed several confidence-based sampling measures to make the agent aware of the accuracy of its predictions to judge the risk of an action sequence. A second experiment series has shown, that proposed sampling measures can improve the efficiency of the training process. Furthermore, they are capable of improving the agent's performance during evaluation even in case the model has been trained using random sampling.

Even if reinforcement-learning has shown to be able to solve many motion-control tasks, these methods still require large amounts of training data. Since the proposed planning based approaches have shown great learning performance during the first steps of acting in a new environment, the next step will be to combine these two algorithmic schemes in a single agent. Such an agent may dynamically choose to either trust the reinforcement learner's value model or to rely on a learned forward model for using a planning-based approach. Similarly, the planner could benefit from the reinforcement-learner's policy during the search phase. Since curiosity-driven learning and other intrinsic reward schemes have already shown promising results for improving reinforcement-learning agents, similar improvements could be achieved when being applied to forward model learning agents.

References

[1] Apeldoorn, D., Dockhorn, A.: Exception-Tolerant Hierarchical Knowledge Bases for Forward Model Learning. IEEE Transactions on Games pp. 1–14 (2020). DOI 10.1109/TG.2020.3008002. URL https://ieeexplore.ieee.org/document/9136897/

[2] Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. IEEE Transactions on Systems, Man and Cybernetics **SMC-13**(5), 834–846 (1983). DOI 10.1109/TSMC.1983.6313077

[3] Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001). DOI 10.1023/a:1010933404324

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)

[5] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (2012). DOI 10.1109/TCIAIG.2012.2186810. URL http://ieeexplore.ieee.org/document/6145622/

[6] Clary, P., Morais, P., Fern, A., Hurst, J.: Monte-Carlo planning for agile legged locomotion. Proceedings International Conference on Automated Planning and Scheduling, ICAPS **2018-June**(Icaps), 446–450 (2018)

[7] Dearden, A., Demiris, Y.: Learning forward models for robots. IJCAI International Joint Conference on Artificial Intelligence pp. 1440–1445 (2005)

[8] Dockhorn, A.: Prediction-based search for autonomous game-playing. Ph.D. thesis, Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik (2020). DOI 10.25673/34014. URL https://opendata.uni-halle.de//handle/1981185920/34209

[9] Dockhorn, A., Apeldoorn, D.: Forward Model Approximation for General Video Game Learning. In: Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG'18), pp. 425–432. IEEE (2018). DOI 10.1109/CIG.2018.8490411. URL https://ieeexplore.ieee.org/document/8490411/

[10] Dockhorn, A., Doell, C., Hewelt, M., Kruse, R.: A decision heuristic for Monte Carlo tree search doppelkopf agents. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8. IEEE (2017). DOI 10.1109/SSCI.2017.8285181. URL http://ieeexplore.ieee.org/document/8285181/

[11] Dockhorn, A., Kruse, R.: Detecting Sensor Dependencies for Building Complementary Model Ensembles. In: Proceedings of the 28. Workshop Computational Intelligence, Dortmund, 29.-30. November 2018, pp. 217–234 (2018)

[12] Dockhorn, A., Kruse, R.: Forward Model Learning for Motion Control Tasks. In: 2020 IEEE 10th International Conference on Intelligent Systems (IS), pp. 1–5. IEEE (2020). DOI 10.1109/IS48319.2020.9199978. URL https://ieeexplore.ieee.org/document/9199978/

[13] Dockhorn, A., Lucas, S.M., Volz, V., Bravi, I., Gaina, R.D., Perez-Liebana, D.: Learning Local Forward Models on Unforgiving Games. In: 2019 IEEE Conference on Games (CoG), pp. 1–4. IEEE, London (2019). DOI 10.1109/CIG.2019.8848044. URL https://ieeexplore.ieee.org/document/8848044/

[14] Dockhorn, A., Tippelt, T., Kruse, R.: Model Decomposition for Forward Model Approximation. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1751–1757. IEEE (2018). DOI 10.1109/SSCI.2018.8628624. URL https://ieeexplore.ieee.org/document/8628624/

[15] Gaina, R.D., Liu, J., Lucas, S.M., Pérez-Liébana, D.: Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10199 LNCS, pp. 418–434. Springer International Publishing (2017). DOI 10.1007/978-3-319-55849-3_28. URL http://link.springer.com/10.1007/978-3-319-55849-3_28

[16] Gaina, R.D., Lucas, S.M., Perez-Liebana, D.: Rolling horizon evolution enhancements in general video game playing. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG), pp. 88–95. IEEE (2017). DOI 10.1109/CIG.2017.8080420. URL http://ieeexplore.ieee.org/document/8080420/

[17] Gu, S., Lillicrap, T., Sutskever, U., Levine, S.: Continuous deep q-learning with model-based acceleration. 33rd International Conference on Machine Learning, ICML 2016 **6**, 4135–4148 (2016)

[18] Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. In: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (eds.) Advances in Neural Information Processing Systems 31, pp. 2450–2462. Curran Associates, Inc. (2018)

[19] Hafner, D., Lillicrap, T., Ba, J., Norouzi, M.: Dream to control: Learning behaviors by latent imagination (2019). URL http://arxiv.org/abs/1912.01603

[20] Henaff, M., Whitney, W.F., LeCun, Y.: Model-based planning with discrete and continuous actions (2017)

[21] Howard, R., Collection, K.M.R., of Technology, M.I.: Dynamic Programming and Markov Processes. Technology Press Research Monographs. Technology Press of Massachusetts Institute of Technology (1960)

[22] Lucas, S.M., Dockhorn, A., Volz, V., Bamford, C., Gaina, R.D., Bravi, I., Perez-Liebana, D., Mostaghim, S., Kruse, R.: A Local Approach to Forward Model Learning: Results on the Game of Life Game. In: 2019 IEEE Conference on Games (CoG), pp. 1–8. IEEE (2019). DOI 10.1109/CIG.2019.8848002

[23] Maschler, M., Solan, E., Zamir, S.: Game Theory. Cambridge University Press, Cambridge (2013). DOI 10.1017/CBO9780511794216. URL http://ebooks.cambridge.org/ref/id/CBO9780511794216

[24] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.a., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). DOI 10.1038/nature14236

[25] Nguyen-Tuong, D., Peters, J.: Model learning for robot control: a survey. Cognitive Processing **12**(4), 319–340 (2011). DOI 10.1007/s10339-011-0404-1. URL https://doi.org/10.1007/s10339-011-0404-1

[26] Perez Liebana, D., Dieskau, J., Hunermund, M., Mostaghim, S., Lucas, S.: Open Loop Search for General Video Game Playing. In: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15, pp. 337–344. ACM Press, New York, New York, USA (2015). DOI 10.1145/2739480.2754811. URL http://dl.acm.org/citation.cfm?doid=2739480.2754811

[27] Perez-Liebana, D., Lucas, S.M., Gaina, R.D., Togelius, J., Khalifa, A., Liu, J.: General Video Game Artificial Intelligence, vol. 3. Morgan & Claypool Publishers (2019). https://gaigresearch.github.io/gvgaibook/

[28] Racanière, S., Weber, T., Reichert, D.P., Buesing, L., Guez, A., Rezende, D., Badia, A.P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., Wierstra, D.: Imagination-augmented agents for deep reinforcement learning. Advances in Neural Information Processing Systems **2017-Decem**(Nips), 5691–5702 (2017)

[29] Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning -. University Press Group Limited, New York (2006)

[30] Rubinstein, R.Y., Kroese, D.P.: Simulation and the Monte Carlo Method. John Wiley & Sons, Inc. (2016). DOI 10.1002/9781118631980. URL https://doi.org/10.1002/9781118631980

[31] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). DOI 10.1038/nature16961. URL http://www.nature.com/articles/nature16961

[32] Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning **3**(1), 9–44 (1988). DOI 10.1007/BF00115009. URL http://link.springer.com/10.1007/BF00115009

[33] Sutton, R.S., Barto, A.G.: Reinforcement Learning, 2 edn. The MIT Press, Cambridge (2018)

[34] Szita, I., Lorincz, A.: Learning tetris using the noisy cross-entropy method. Neural Computation **18**(12), 2936–2941 (2006). DOI 10.1162/neco.2006.18.12.2936

[35] Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**(3-4), 279–292 (1992). DOI 10.1007/bf00992698

[36] Weber, T., Racanière, S., Reichert, D.P., Buesing, L., Guez, A., Rezende, D.J., Badia, A.P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Silver, D., Wierstra, D.: Imagination-Augmented Agents for Deep Reinforcement Learning (2017). URL http://arxiv.org/abs/1707.06203

[37] Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer International Publishing, Cham (2018). DOI 10.1007/978-3-319-63519-4. URL http://gameaibook.org/book.pdf