Evolutionary Preference-Based Reinforcement Learning for Partially Observable Environments

Anonymous Author(s)
Affiliation
Address
email

Abstract

A significant challenge in reinforcement learning is how to accurately convey our desires to the artificial agent. Preference-based reinforcement learning uses human preferences between concrete examples of the agent's behavior to model the reward or the return function the human intends. However, the existing models discard much information and can therefore be less accurate than possible, especially if the environment is only partially observable.

To overcome this limitation, the model presented in this thesis combines all available information (all observations made during one episode) through a temporal convolutional net to model the return function, instead of a reward function, from preferences. The reinforcement learning – implemented with a genetic algorithm – is then guided by this model.

We show that our method is a viable way to apply preference-based reinforcement learning in partially observable environments.

1 Introduction

To make an artificial intelligent agent learn desired behavior, one simple but powerful paradigm is reinforcement learning (RL), which uses as the only guidance a function (called reward function) that quantifies the desirability of every possible behavior. But when specifying a reward function, taking every possible behavior into consideration might be very difficult. In such cases, when applied to behavior, we did not foresee, the reward function is prone to misrepresenting our desires, which can cause the agent to learn undesired – even harmful – behavior.

Human-in-the-loop reinforcement learning (HRL) tries to solve this alignment problem with the reward specification by removing the reward function and instead try to learn from a human directly. This can be done through various mechanisms like evaluative feedback, advice, demonstration, correction, or preferences.

Prefence-based reinforcement learning (PBRL), a human-in-the-loop reinforcement learning variant, modifies the concept to use not on an abstract reward function, but only preferences between concrete examples of the agent's current behavior, from which a model of the reward function we intend is deduced. This model is continuously updated with new preferences so that errors it has can be corrected.

While PBRL has shown application in a lot of different domains, the field recently received increased attention from the research community through its application in the field of natural language processing (NLP). Most prominent here is the usage in ChatGPT (Ouyang et al. 2022) where a method similar to (Christiano et al. 2017) has been used to fine-tune a large language model. While our improvements focus on more traditional RL problem formulations (instead of NLP), we believe our methods could also show potential there.

Submitted to 16th European Workshop on Reinforcement Learning (EWRL 2023). Do not distribute.

Previously existing PBRL methods have the weakness of being infeasible to be applied in partially observable Markov decision processes (POMDPs), because the model of the reward function they train discards much useful information from the agent's observations: They model each reward as depending only on a compressed aggregation of many observations (Akrour et al. 2012) or on a single observation the agent has made of the state (Christiano et al. 2017). The latter is sufficient if each observation contains complete information about the state – but in general, the environment might only be partially observable; then, modelling the reward as a function of single observations will be inaccurate. Practically every realistic environment is partially observable, so this is a great restriction.

The variant of preference-based reinforcement learning introduced in this work features a model that is general enough to represent our reward function as accurately as possible even if the environment is only partially observable. The solution it uses is to remember and combine multiple observations: Instead of modelling each reward as depending on a single observation, the model combines (via a temporal convolutional net (Bai et al. 2018)) all observations the agent has made of the environment during one episode of interacting with it to model the sum of rewards over the episode (that is, the return). It does this by requiring only preferences between the agent's behavior during different episodes.

The fact that the model provides not reward values directly but only return values shrinks the set of reinforcement learning algorithms that can use it as guidance. One suitable class of them are genetic algorithms.

We evaluate our method on two different partially observable environments, showing that it is able to solve these kinds of tasks. Additionally, it is compared to (among other things) an ablation made to resemble some previous PBRL methods, in which its return model is replaced by a reward model. Our source code and data are available online at [to be announced].

2 Background

2.1 Reinforcement learning

The following formalization[1] of reinforcement learning is known as a partially observable Markov decision process (POMDP). It is characterized by the functions p and ω (described below) and the four sets:

- $\mathcal{S}$: the states the environment in which the agent acts can take on;
- $\mathcal{O}$: the possible observations the agent can make of the environment;
- $\mathcal{A}$: the actions the agent can take; and
- $\mathcal{R} \subset \mathbb{R}$: the reward values the agent can receive.

The interaction between the agent and the environment can be described as a series of actual states $S \in \mathcal{S}$, observations $O \in \mathcal{O}$, actions $A \in \mathcal{A}$ and rewards $R \in \mathcal{R}$ – one for each time step:[2]

$$S_0, O_0, A_0, R_1, S_1, O_1, A_1, R_2, S_2, O_2, A_2, R_3, \ldots \tag{1}$$

This is called a trajectory. It is determined iteratively, with the following happening at each time step t:

1. The agent receives some information about the environment's current state S_t. However, we do not assume that the agent learns everything about S_t; instead, it receives an observation O_t that depends on S_t, but might not uniquely identify it (hence the name *partially observable*). Precisely, O_t is drawn from the probability distribution ω conditioned on S_t, i. e. : $O_t \sim \omega(S_t)$.
2. Then, the agent chooses an action A_t by drawing it from a probability distribution conditioned on all previous observations[3]: $A_t \sim \pi(O_t, O_{t-1}, \ldots, O_0)$. π is called the agent's policy function.

[1] The notation is adapted from (Sutton and Barto 2018) and extended.

[2] Except that (following (ibid.)) there is no reward at the first step.

[3] It could also be conditioned on the previous rewards and actions, but in this work, only observations are used.

3. Finally, the environment takes on a new state S_{t+1}, and the agent receives a reward R_{t+1}. Both depend only on the environment's current state S_t and the agent's action A_t, similar to how a Markov chain evolves: $(S_{t+1}, R_{t+1}) \sim p(S_t, A_t)$. p is called the environment's transition (or dynamics) function.

Without loss of generality, we can assume that there is a single starting state S_0 and a terminating state S_T. The interaction begins with the environment being in the state S_0, and it ends when it reaches S_T, after which the state is reset to S_0 and the interaction begins anew. Each round of interaction is called an episode, and the cumulative reward of an episode is called return $G = R_1 + R_2 + \cdots + R_T$.

Now we can state the goal of the reinforcement learning agent: It should find a policy function π that maximizes the return the agent gains per episode, with its only guidance being the observations and rewards it receives. In particular, the agent is not provided a model of the environment. In other words, O_t, R_t, and of course A_t and $\mathcal{A}$ are known to the agent, while everything else ($\mathcal{S}$, $\mathcal{O}$, $\mathcal{R}$, S_t, p, and ω) is unknown. One, though by far not the only method to use for reinforcement learning is described in the next section.

2.2 Genetic algorithm

Genetic algorithms are general methods to optimize black-box functions – in particular, functions whose gradient is unknown. The task of reinforcement learning can be[4] reduced to the problem of optimizing a black-box function: We parameterize the policy function π and take the parameters as the input to the black-box function. The output, then, is the return which the policy with the given parameters brings the agent. So the policies that optimize this function are the ones that gain the highest return.

The genetic algorithm by Such et al. (2018) finds good policies by iteratively improving a set of policies called population, with each step consisting of:

1. Reproduction A random sample of policies from the population is copied.

2. Mutation The copies (also called children) are perturbed with Gaußian noise.

3. Selection Each child policy is tried out for one episode, and the best performing ones are selected as the next population.[5]

The reason why this simple algorithm works is that the black-box function which maps policies to returns is more or less continuous, so if the mutation noise is small enough, the population is updated in a fashion similar to stochastic gradient ascent.

2.3 Preference-based reinforcement learning

Reinforcement learning can be a powerful method to make an artificial intelligent agent learn to pursue a desired goal: Instead of specifying the behavior necessary to achieve the goal, we can just specify a reward function (i. e. the component of the environment's transition function p determining rewards) that rewards the agent for achieving the goal and let the agent itself figure out the necessary behavior.

While this abstract expression of our desires is convenient, it also bears a great danger: We can easily forget crucial details when specifying a reward function, which then does not represent our desires exactly, but is misaligned with them. This, in turn, can cause the agent to take us too literally and fulfill our stated goal in unintended ways, as in the following illustrative example:

If we want a robot to make us food, we might specify the obvious reward function that gives rewards exactly in those states where we have something to eat. This might lead the robot to "cook[] the cat for dinner, not realizing that its sentimental value outweighs its nutritional value" (Russell 2021). It did what we specified – but we failed to specify what we wanted.

This is known as the alignment problem of reward functions, and of AI agents in general. One attempt at a solution to this problem is reward learning (Leike et al. 2018), where we give the agent – instead

[4] but need not be, since this reduction discards much information, such as connections between observations and rewards within single episodes

[5] Here, the original algorithm and our implementation differ slightly, as explained in Appendix A.1.

of a reward function – information about the desirability of behavior that is easier to give correctly, from which the agent then infers a reward function and uses it.

Christiano et al. (2017) follow an example of this approach called preference-based reinforcement learning: They query the human overseers for binary preferences between segments of the trajectories (restricted to containing only observations and actions) from the agent's interaction with the environment. A neural net is trained with these preferences and used by the agent as its reward function.

To model reward values given only preferences, the model's predicted reward values are interpreted according to the Bradley-Terry model[6] to derive predicted preferences from them, that is, the difference between the return predictions is equated with the logit of the preference prediction. The reward model is trained with gradient descent to minimize the cross-entropy loss between the predicted and the actual preferences.

2.4 Temporal convolutional nets

Besides the reinforcement learning methods covered in the previous sections, temporal convolutional nets (Bai et al. 2018) are another tool used in this work. They are neural nets used for sequence-to-sequence processing: The input sequence is transformed into the output sequence by repeatedly applying discrete convolutions with increasingly spread kernels and interleaving a nonlinear activation function. So effectively, each value in the output is computed by applying a regular neural net (that is using parameter sharing) to a large section (called receptive field) of the input, meaning that temporal convolutional nets are capable of tasks where large contexts have to be considered.

3 Method

This section lays out a method of preference-based reinforcement learning that, unlike previously existing ones, is suited for application in partially observable MDPs. Its structure – shown in Figure 1 – is copied from the method in (Christiano et al. 2017), but the individual parts are exchanged.

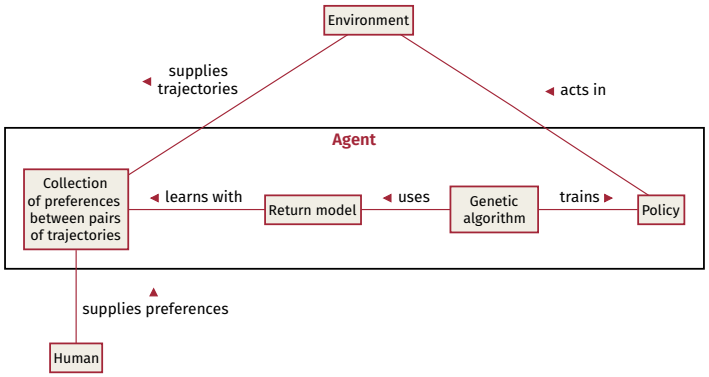

Figure 1: The examined method's components.

3.1 Description

To successfully interact with an environment that is partially observable, the agent needs a policy function that can memorize. This is implemented[7] as a neural net (with ReLUs as activation functions) whose input layer is the concatenation of the most recent observations[8], the number of which is a hyperparameter. The linear output layer directly represents the action to take.[9]

[6] which is also used in the Elo scoring system

[7] Another memory architecture tried was a recurrent net, made of gated recurrent units intertwined with feed-forward layers (similar to skip connections). In preliminary experiments, this was equal in performance, but more demanding on computing power.

[8] with zero-padding at the beginning of episodes

[9] Thus, the policy is deterministic, even though the formalism in Subsection 2.1 allowed for indeterminism.

The policy is optimized by the genetic algorithm from Subsection 2.2 with respect to return values provided not by the environment, but estimated by a model.

This return model is the temporal convolutional net from (Bai et al. 2018)[10] (see Subection 2.4). The trajectory (containing observations and actions) of one episode is the net's input, and its output sequence is summed and used as the return value by the genetic algorithm.

To train the return model, the human is queried for binary preferences between pairs of whole trajectories sampled uniformly at random from the same distribution as the ones generated in the most recent iteration of the evolutionary algorithm. The preferences (including the trajectory pairs they were given for) are stored in a buffer, and the return model is trained with random minibatches from this buffer to minimize the cross-entropy loss under the Bradley-Terry model, like in the method of Christiano et al. (2017) (see Subsection 2.3).

The three processes of querying preferences from humans, training the return model and optimizing the policy with the evolutionary algorithm are interleaved by an early-stopping schedule that tries to query just the amount of preferences the return model needs to achieve its maximal accuracy. This schedule, however, leaves room for improvement, since it produced nearly constant rates of one preference per 13–15 episodes in our experiments, which we hypothesize is more than necessary.

3.2 Related work

There are other preference-based reinforcement learning methods similar to ours. Their main differences are highlighted here.

The preference-based reinforcement learning method in (ibid.) queries preferences not between whole, but short segments of trajectories. Those have the benefit of taking less time to evaluate for humans, but also the drawback of providing less context, which can make it impossible in POMDPs to infer enough information about the environment's hidden state to accurately estimate reward values. Also, instead of a model of the returns depending on whole trajectories, a model of the reward values depending on single observations is used, which then faces the same problem as the human preference givers.

Modelling the return has been done by Akrour et al. (2012), but depending on a compressed representation of trajectories. Kupcsik et al. (2018) also model the return, but as a function of policy parameters directly. This abolishes the need to interact with the environment during policy optimization, but might be difficult if the relationship between policy parameters and behavior is complex.

An evolution strategy was already used by Busa-Fekete et al. (2014). However, they don't use the preferences to train any model – but rather to directly select the best mutants in each generation with high certainty, via a racing algorithm. This requires very many preferences per generation: between 2000 and 3000 to select the best 5 out of 10 with 95 % certainty.

4 Experiments

4.1 Setup

The method outlined in the last section is tested in two environments.

Copy Memory , inspired by the task of the same name in (Bai et al. 2018), is intended as a simple test of the agent's memory. Its observations and actions are one-dimensional numbers. The observations are uniformly random between 0 and 1 at the beginning, and afterwards 0:

$$O_t \begin{cases} \sim U(0,1) & \text{for } t = 0,\dots,3 \\ = 0 & \text{for } t = 4,\dots,7 \end{cases} \tag{2}$$

The agent has to remain silent first and then re-enter the previous observations:

$$R_t = \begin{cases} -|A_{t-1}| & \text{for } t = 1,\dots,3 \\ -|A_{t-1} - O_{t-4}| & \text{for } t = 4,\dots,7 \end{cases} \tag{3}$$

After step number 7, the episode finishes.

[10] source code available at `https://github.com/locuslab/TCN`

Pendulum-xy is an adapted version of the task Pendulum-v1 from the OpenAI Gym (Brockman et al. 2016). The original makes observable the pendulum's x and y positions as well as its angular velocity, which together is all of the state. Here, the velocity is censored to make the environment partially observable.

To save cumbersome labor and also to have a more objective evaluation, the preferences are not queried from humans, as they would be in a real application, but automatically generated to reflect the true return from the environment.

Hyperparameters are tabulated in Appendix A.2.

4.2 Ablation studies

In addition to the method as described in Section 3, three variations of it are tested to examine its individual aspects.

Reward model Subsection 3.2 stated that modelling reward values depending only on single observations is insufficient in POMDPs. This claim is assessed by using such a reward model instead of the return model.

A reward model can be implemented as a special case of the return model: by setting the kernel size of the temporal convolutional net to 1. To compensate for the loss of parameters this entails, it is given more neurons per layer.

True return To be able to separately judge the performances of the genetic algorithm optimizing the policy function and of the return model modelling the return, the latter is left out and the genetic algorithm uses the true return values.

Policy without memory To prove that the tasks are indeed partially observable and need memory to be solved, they are tried with policies that merely have access to the very last observation, that is, a history of length 1. They are only tested in combination with using the true return values, as it is expected that their performance is very low, even under otherwise good conditions.

4.3 Metrics

After each step of the genetic algorithm, the 8 policies that achieved the highest return are reevaluated.[11] Their mean return serves as the main indicator of the whole method's performance.

The following metrics about the return (or reward) modelling are reported:

- The training loss (a cross-entropy-loss).
- The model's accuracy in predicting preferences.
- The probability the model assigned to the actually preferred trajectory to be preferred, showing not only the accuracy, but also the confidence of its predictions.

4.4 Results

The mean-of-best-8 return metric over the course of training is shown in Figure 2. The other metrics were almost constant while training, so they are shown aggregated in Figures 3 and 4. Every value reported is the mean of 8 runs.

5 Discussion

5.1 Usefulness of memory

The purpose of this work is to test the new preference-based reinforcement learning method in partially observable MDPs; in other words: in environments where an agent needs to combine

[11] Also logged was the metric of reevaluating 8 random policies, which was somewhat lower, but highly correlated.

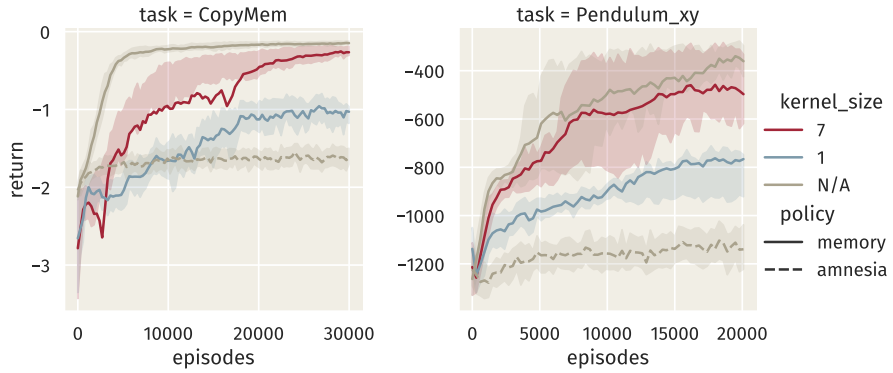

Figure 2: Return the agent is gaining after interacting with the environment for increasing numbers of episodes in the Copy Mem(ory) and the Pendulum-xy tasks. Kernel size 7 denotes using the regular return model; 1, using the reward model in the ablation study (see Subsection 4.2); and N/A, using neither but the true return instead. The agent uses a policy with (solid lines) or without (dashed lines) memory. The lines are means of the 8 runs, shaded regions are interquartile ranges.

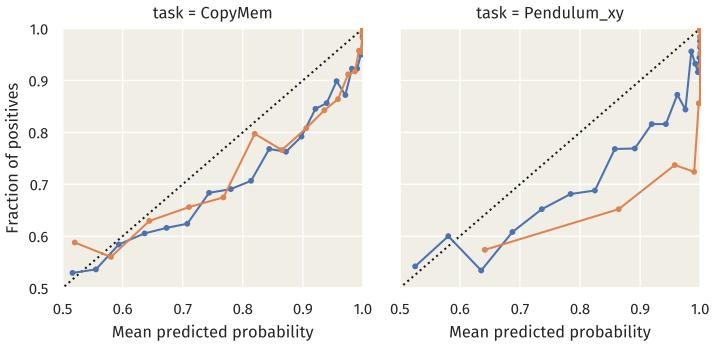

Figure 3: Calibration curves of the return model (orange) and the reward model (blue). The x-axis shows the confidence of the models' predictions, the y-axis the fraction of correct predictions. Each curve has 20 points, each of which aggregates the same number of predictions. The shown curves are accumulated across all episodes of the 8 runs.

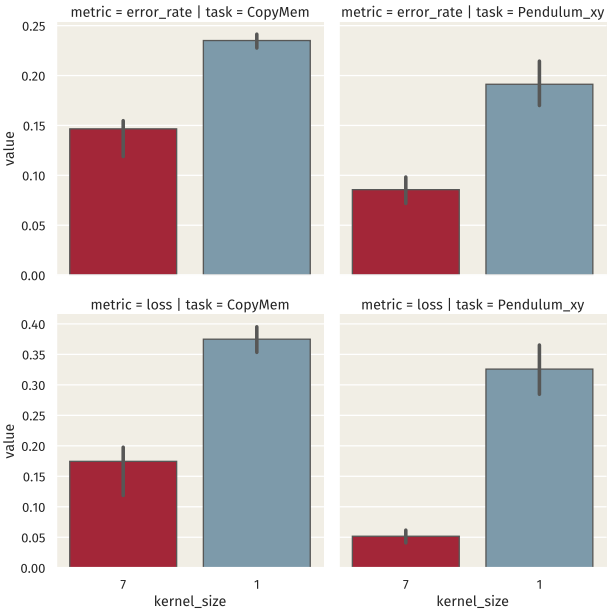

Figure 4: Metrics for the return/reward models (kernel sizes 7/1, respectively). The error rate (equal to one minus the accuracy) and the training loss are shown. Big bars denote means across all episodes of the 8 runs, small bars interquartile ranges between the runs.

multiple observations to choose good actions. Therefore, it should be verified that the environments tested fit that description.

This was successfully done by the ablation of the policy's memory, which drastically decreased the agent's performance. Even more, the memory enabled performances close to optimal[12], which shows that its architecture and sizes are sufficient for the tried tasks.

5.2 Success of return learning

The goal of the new preference-based reinforcement learning method is to be applicable in POMDPs. Judging by the two tasks tried, it largely reached this goal: Compared to reinforcement learning guided by the true returns, it is almost equally good in the Pendulum-xy task, and in the Copy Memory task, while noticeably worse, manages to close the gap after some time.

5.3 Failure of reward learning

Subsection 3.2 argued that previously existing preference-based reinforcement learning methods are not suited to POMDPs. To exemplify this, the return model was replaced with a reward model in one experimental condition, mimicking the method in (Christiano et al. 2017).

This ablation considerably worsened the results. The policy optimization seemed to get stuck in local optima.[13]

But still, training a memorizing policy with a reward model was better than training a memoriless policy with the true returns, which suggests that a memory is more important for producing good actions than for recognizing them.

There were two outliers in the Pendulum-xy task which seemingly converged to the same performance as when using the return model. It can only be speculated how this happened. On the one hand, policies of different qualities produce different trajectories that can possibly be distinguished even by a memoriless reward model. On the other hand, policies might cheat the reward model by producing poor trajectories that – to it – look indistinguishable to good ones, thus negating the evolutionary pressure towards good policies. The outliers show that truly good policies can nevertheless sometimes be favored over cheating ones.

5.4 Return/reward learning metrics

Having a metric of the return model's performance – of how well it conveys the preference giver's goals to the genetic algorithm – is important for at least two reasons:

- The schedule for adapting the rate of preference queries depends on such a metric.
- If the whole system's performance is bad, the metric gives a hint whether the bottleneck is the return model or the evolutionary algorithm.

In the experiments, three such metrics were investigated:

1. The performance the agent gains by ablating return learning, which is inversely related to the return model's performance.
2. The return model's accuracy in predicting preferences.
3. The training loss of the model.

Since predicting preferences is only a means to attain desired behavior, the first metric is what we are ultimately interested in. It, however, cannot be calculated in those tasks for which preference-based

[12] In the Copy Memory task, the optimal reward is always zero, while the agent achieved an average reward of ca. –0.03 – equal to the magnitude of the noise to which the policy is subjected by the genetic algorithm. The performance at the Pendulum-xy task is equal to the one achieved at the completely observable original in preliminary experiments, which is –400 (standard deviation 200).

[13] In the Pendulum-xy task, the agent tended to swing the pendulum around (while the task is to balance it upright). The return of –1 in the Copy Memory task can be achieved by always choosing the action ½ in the second halves of episodes.

reinforcement learning is intended: when there is no return function available. So, the question is whether the second and the third metric are a good proxy for the first.

This is affirmed by the experiments' results: The return model's accuracy was higher and its loss was lower in the Pendulum-xy task – and it was in this task where the advantage of ablating return learning (the first metric) was smaller. Also, ablating the return model's memory (turning it into a reward model) caused all three metrics to change in the same, expected ways.

But one could imagine other cases where the three metrics would not coincide: Even if the return model made few errors overall, it could still systematically underestimate and therefore not reinforce certain behavior that is highly desired but currently rarely shown. Conversely, the return model might mispredict many preferences, but if those are only weak preferences between behaviors of almost equal desirability, it might not impact the agent's performance much.

The problem in both these hypothetical cases is that the accuracy metric treats all preferences as equally important, even though they are not; so a solution might be to query not only preferences, but also how strong they are, and take this into account (for example by weighting the preferences by their strength in the calculation of the return model's accuracy).

In addition, the plots of the return and reward model's confidence distributions in Figure 3 show that those predictions given with higher confidence are indeed more likely to be correct. This information could potentially be used to query preferences not between random trajectory pairs, but ones where the model's uncertainty is high, following the idea in (Christiano et al. 2017). However, the models are far from calibrated, which could indicate that the Bradley-Terry underlying them is not a good fit for this application.

6 Conclusion

The new method of preference-based reinforcement learning based on a return function learned from preferences proved capable of performing well in POMDPs: Its performance in the experiments was comparable to reinforcement learning based on the true return function directly.

A representative of previous methods – akin to the one in (ibid.) – which models individual rewards instead of returns was shown to perform considerably worse.

The method as presented here is meant as a first proof of concept, favoring simplicity over sophistication. It has many possibilities of improvement, which might be especially worthy when scaling up the method to harder tasks, as the genetic algorithm and the temporal convolutional net individually already have been.

For example, the method should query preferences more economically by selecting the trajectories to be compared by the human not randomly, but to maximize the amount of information the return model will gain from a preference between them. In (ibid.), this is attempted by selecting trajectories for which the model's prediction is especially uncertain (measured by the variance in an ensemble of models), with mixed results.

Also, the used variants of the temporal convolutional net and the genetic algorithm are themselves rather simple and do not use many known improvements. Some of those are listed in the original papers (TCN: Bai et al. 2018; genetic algorithm: Such et al. 2018).

More generally, while preference-based reinforcement learning is enhanced by the option of applying it in partially observable environments, its promise of artificial intelligence that is aligned with human values is still blocked by many barriers: the problem of safe exploration, the risk that the agent might tamper with the process of preference giving, or the fact that even alignment of the return function does not guarantee alignment of the policy (Hubinger et al. 2019).

References

Akrour, Riad, Marc Schoenauer, and Michèle Sebag (2012). "APRIL: Active Preference Learning-Based Reinforcement Learning". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Peter A. Flach, Tijl De Bie, and Nello Cristianini. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 116–131. ISBN: 978-3-642-33486-3 (cit. on pp. 2, 5).

Busa-Fekete, Róbert, Balázs Szörényi, Paul Weng, Weiwei Cheng, and Eyke Hüllermeier (Dec. 2014). "Preference-based reinforcement learning: evolutionary direct policy search using a preference-based racing algorithm". In: *Machine Learning* 97.3, pp. 327–351. DOI: 10.1007/s10994-014-5458-8. URL: https://hal.inria.fr/hal-01079370 (cit. on p. 5).

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). *OpenAI Gym*. eprint: arXiv:1606.01540 (cit. on p. 6).

Christiano, Paul F., Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep Reinforcement Learning from Human Preferences". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS '17. Long Beach, California, USA: Curran Associates Inc., pp. 4302–4310. ISBN: 9781510860964 (cit. on pp. 1–2, 4–5, 8–9).

Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun (Apr. 2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. arXiv: 1803.01271v2 [cs.LG]. URL: http://arxiv.org/abs/1803.01271v2 (cit. on pp. 2, 4–5, 9).

Kupcsik, Andras, David Hsu, and Wee Sun Lee (2018). "Learning Dynamic Robot-to-Human Object Handover from Human Feedback". In: *Robotics Research: Volume 1*. Ed. by Antonio Bicchi and Wolfram Burgard. Cham: Springer International Publishing, pp. 161–176. ISBN: 978-3-319-51532-8. DOI: 10.1007/978-3-319-51532-8_10. URL: https://doi.org/10.1007/978-3-319-51532-8_10 (cit. on p. 5).

Leike, Jan, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg (2018). *Scalable agent alignment via reward modeling: a research direction*. DOI: 10.48550/ARXIV.1811.07871. URL: https://arxiv.org/abs/1811.07871 (cit. on p. 3).

Such, Felipe Petroski, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune (Apr. 2018). *Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning*. arXiv: 1712.06567v3 [cs.NE]. URL: http://arxiv.org/abs/1712.06567v3 (cit. on pp. 3, 9–10).

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning. An Introduction*. Second edition. Cambridge, Massachusetts: The MIT Press (cit. on p. 2).

Hubinger, Evan, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant (2019). *Risks from Learned Optimization in Advanced Machine Learning Systems*. DOI: 10.48550/ARXIV.1906.01820. URL: https://arxiv.org/abs/1906.01820 (cit. on p. 9).

Russell, Stewart (2021). "Human-Compatible Artificial Intelligence". In: *Human-Like Machine Intelligence*. Ed. by Stephen Muggleton and Nick Chater. Oxford University Press. URL: https://people.eecs.berkeley.edu/~russell/papers/mi19book-hcai.pdf (cit. on p. 3).

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, et al. (2022). "Training language models to follow instructions with human feedback". In: *ArXiv* abs/2203.02155 (cit. on p. 1).

A Appendix

A.1 Adaptation of genetic algorithm

The genetic algorithm in (Such et al. 2018) attempts to find the elite (the best policy) of each generation by taking the mean of 30 reevaluations of each of the 10 currently best policies. This elite is then retained in the population for the next generation. However, with the small population size (16 + 16) in our experiments, this would inflate the number of evaluations per generation,[14] so we do not use this approach. To nonetheless feature elitism, in each generation, we evaluate not only the children, but also reevaluate all parents (each individual only once),[15] and select the best of all of them as the next generation.

[14] from 16 to 316

[15] which makes 32 evaluations

A.2 Hyperparameters

The most important hyperparameters are listed in Table 1. Some of them were chosen at will and deemed to work satisfactorily, others (like the genetic algorithm's mutation strength) needed brief tuning.

Table 1: Hyperparameters.

	Task			
	Copy Memory		Pendulum-xy	
Policy				
History length	4		8	
Neurons in 1st layer	16		32	
Neurons in 2nd layer	16		64	
Genetic algorithm				
Mutation standard deviation	0.03		0.1	
Population size	16		16	
Number of children	16		16	
Return (reward) model				
Neurons per layer	4	(10)	12	(32)
Number of layers L	2	(2)	3	(3)
Kernel size K	7	(1)	7	(1)
Receptive field $(K-1)(2^L-1)+1$	19	(1)	43	(1)
Parameters	441	(441)	5497	(5761)