DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Robotics

Preference-based Inverse Reinforcement Learning Approach for Robot Locomotion under Suboptimal Learning Conditions

Andreas Josef Binder

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Robotics

Preference-based Inverse Reinforcement Learning Approach for Robot Locomotion under Suboptimal Learning Conditions

Präferenzbasierter inverser Verstärkungs-Lernansatz für die Roboterfortbewegung unter suboptimalen Lernbedingungen

Author:	Andreas Josef Binder
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Dr. rer.nat. Zhenshan Bing
First External Advisor:	M. Sc. Marcel Kurovski, inovex GmbH
Second External Advisor:	Dr. rer.nat. Florian Wilhelm, inovex GmbH
Submission Date:	October 26, 2020

I confirm that this bachelor's thesis in robotics is my own work and I have documented all sources and material used.

Munich, October 26, 2020 Andreas Josef Binder

Acknowledgments

First and foremost, I want to thank the inovex GmbH for its generous support over the last few months. The friendly and productive atmosphere in the office helped me tremendously to work concentrated on this thesis. Especially, I want to express my highest gratitude for my dedicated advisor Marcel Kurovski who guided me patiently throughout this project with his valuable experience and expertise as data scientist. His calm and decisive calls have had a huge impact on this thesis. I also want to thank Dr. Florian Wilhelm who always took the time for my problems despite his full schedule. In general, I am glad I had the possibility to meet all the wonderful people at inovex GmbH where it always was a joy to be around.

Of course, I also want to thank my TUM supervisor, Prof. Dr. Alois Knoll for making this work possible. Additionally, I appreciate the help of my TUM advisor Dr. Zhenshan Bing who was actually the first to introduce me to reinforcement learning. The interesting discussions with him narrowed down the scope of this thesis and developed a deep interest in the scientific process in me.

Abstract

Recent advances in Reinforcement Learning have gained huge attention by both the industry and academia. Additionally, with Neural Networks (NN) as universal funcion approximators, it is possible to model complex value functions which used to be intractable. Especially policy-gradient algorithms for policy optimization are popular among researchers because those can be applied on continuous, high-dimensional tasks.
Also, those methods have better convergence properties than traditional RL algorithms. In addition, they can model the true distribution of actions quite well when provided with a proper reward function. That is where the concept of Inverse Reinforcement Learning (IRL) comes into play. Given some demonstrations generated by a policy, IRL tries to infer the underlying reward function. The main advantage of IRL is that there is no need to construct a reward function, a process that can take long and be error prone. Unfortunately, most existing IRL approaches suffer from the problem of not being able to extrapolate beyond the provided demonstrations. This work aims to tackle this problem by using trajectories generated by a suboptimal policy which is used to deliver better results compared to other state-of-the art approaches like Generative Adversarial Imitation Learning (GAIL). I build on the work of Brown et al. which uses pairwise trajectory ranking to capture the intent of the agent and thus obtains good results on standard Mujoco tasks and Atari games. Additionally, I investigate the impact of the loss function. Specifically, I analyze whether there are performance gains when choosing a triplet ranking loss that operates on triplets of trajectories. In detail, I measure the traveled distance and the ranking quality of the respective approaches.
The motivation to use suboptimal demonstrations reflects the fact that generally those are easier to obtain and can be derived for most tasks relatively cheap. In addition, it can be argued that the reward function obtained by IRL from preferences mirrors the intention of the agent instead of plainly trying to explain the presented trajectories.
I aim to implement those ideas for a snake-like robot simulated on the Mujoco physics engine. Those autonomous robots have a broad range of application, most notably for rescue missions on terrain that is barely accessible to humans. For instance, snake-like robots are used to search and detect human beings buried by collapsed houses in case of an earthquake.

List of Abbreviations

BCE	Binary Cross-Entropy
DOF	Degrees of freedom
DL	Deep Learning
D-REX	Disturbance-based Reward Extrapolation
GAIL	Generative Adversarial Imitation Learning
GPU	Graphical Processing Unit
IL	Imitation Learning
IRL	Inverse Reinforcement Learning
LTR	Learning-to-Rank
MDP	Markov Decision Process
ML	Machine Learning
MuJoCo	Multi-Joint dynamics with Contact
NN	Neuronal Network
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
SL	Supervised Learning
TD	Temporal Difference
T-REX	Trajectory-ranked Reward Extrapolation
TRPO	Trust Region Policy Optimization
UL	Unsupervised Learning
VFA	Value Function Approximator

Contents

1. Introduction

1.1. Motivation

The number of applications utilizing Machine Learning (ML) has skyrocketed for roughly 10 years with no end in sight. Arguably, the great breakthrough happened in 2012 when the so called AlexNet [KSH12] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Rus+15], where the goal is to correctly classify and detect objects and scenes. The proposed network architecture achieved a top-5 error performance that almost cut the error of the previous winner in half.
Inspired by that success, also people outside the ML research community realized the significance of the event which caused broader attraction to and investments in ML as a field. There were major developments ongoing which all fuel Deep Learning (DL) as one branch of ML in particular.
The creators of AlexNet describe in their paper two pillars as of great importance for there success. First, they used a Deep NN (DNN) with around 60 Million parameters combining multiple layers, all of them serving a different purpose such as feature selection or feature extraction. They also note that the depth of the network is critical for its performance.
The issue of having a deep and therefore computationally more expensive network is nowadays approached by running multiple Graphical Processing Units (GPUs) in parallel. Those technological advances at hardware level are another reason for the recent success in DL.
Additionally, with so many people having access to the internet, connecting all of our devices, the amount of data being publicly available grew exponentially over the past few years. Very often the success of DL methods strongly correlates positively with the number of training samples at hand. Thus, another problem, that limited the application of DL, was mitigated.
One research area strongly influenced by DL is robotics. Robots for plain and repetitive tasks already exist for a long time since they are cheaper and less error-prone than humans. DL can be especially useful for processing and interpreting the input a robotic agent receives via its sensors. An example of the former mentioned task is image recognition where the robot predicts objects captured by its camera. On the other hand, the latter problem can be seen as on how to react to this environment information. In

this thesis, I will focus on the second aspect, meaning I am interested in optimizing my decision, that I made along the learning process.
I opted to evaluate my approach on a simulated snake-like robot. Those kinds of robots have several use cases such as exploration, inspection, monitoring, as well as search and rescue missions. It is noteworthy that $ACM-R5$ [Yam+05] is the only robot, that is able to perform a cleanup mission at the devastated power plant of Fukushima, Japan.

Reinforcement Learning (RL) is a collection of learning techniques and paradigms. Its goal is to teach an agent on how to behave optimally in an environment; in this thesis, the snake-like robot is an instantiation of the agent within the RL framework. The agent achieves its objective by continuously interacting with the environment and learning about its actions through feedback from the environment. The snake-like robot does so by moving its body and receives information about its current state through sensors. It also obtains a reward indicating the quality of the decision. This process is modeled by a reward function that depends on the current state and returns its immediate value.

It can be very cumbersome to handcraft such a function. Furthermore, I might end up with an agent who learnt unintended behaviour and still maximizes the cumulative reward. Therefore, IRL was created with the idea of infering the reward function through demonstrations. This thesis deals with IRL from preferences which takes ranked inputs. I do this in order to capture the intent of the agent and by learning to distinguish good and bad behaviour with respect to the preference label. This work can achieve good results when using suboptimal training data, which makes this approach cheap and scalable. A training set is considered suboptimal when at least a large part of the data was generated by a novice with regards to the performed task.

1.2. Research Questions and Procedure

I aim at providing a solution for learning a reward function given suboptimal demonstrations. Daniel Brown et al. [Bro+19] approached this problem by using a dataset of pairwise ranked trajectories where a label indicates whether the first or second one is to be preferred. They feed this information into a NN which learns features by examining why one trajectory is worse or better than the other one. This way they can reconstruct the ground truth reward function that can extrapolate beyond the performances seen in the training set, nevertheless one missing point in their evaluation is the impact of the ranking approach. Especially, whether performance is improved or deteriorates

when ranking more than two trajectories at once. This leads to the following research questions:

1. Is it possible to teach a snake-like agent how to move forward using ranked, suboptimal trajectories?
2. How does the kind of used ranking mechanism impact the performance?

I define performance on two metrics: first, how well the learnt reward function approximates the original one. To do this I compare the covered distance of the agent, that was trained on the synthetic reward function, and of the trajectories contained in the training set. Second, the ranking quality will be examined. Unseen data will be used to check, which network has better learnt to rank the data according to the timestep as label.

1.3. Outline

This thesis consists of six chapters. In the first one I laid out a rough overview of the paradigms involved in this work and the research question I aim to answer.
In the following chapter I will talk about the theoretical background. In particular, I will talk about the theory of RL and its underlying framework, Markov Decision Process. I will clarify how IRL differs from RL and we will contrast IRL to Imitation Learning (IL). I also describe the simulation environment and the agent in more detail.
In chapter three I will shed light on the related work that this thesis builds upon. It will contain information about the current state of research on snake-like robots as well as focuses. on preference-based IRL.
Chapter four explains the approach I chose to tackle the problem. I will define the task, give details on the data, its processing and explain how preference learning works.
Chapter five presents the results I achieved with my procedure and answer my research questions.
Eventually in chapter six, I draw my conclusion and also discuss ideas that could increase the performance.

2. Theoretical Background

This chapter covers the theoretical basic knowledge. We will discuss Markov Decision Processes and RL. Additionally, we give an overview on Learning to Rank (LTR). Lastly, we explain IL and IRL and lay out there differences.

2.1. Reinforcement Learning

I want to mention, that "Reinforcement Learning: An Introduction" by Sutton and Barto[SB18] is used as reference through this section.

2.1.1. Markov Decision Process

Markov Decision Processes (MDPs) are the main framework RL utilizes to model an agent who is interacting with an environment. The components of a MDP and its variations will be explained.

The simplest version of a MDP is a Markov Process (MP), also called Markov Chain: all one has here is a (finite) set of states S and a state transition probability matrix $P_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s)$. A state $s \in S$ is a snapshot of the current location of the agent with t indicating the timestep. $P_{ss'}$, on the other hand, tells you the probability, that you end up in specific state s' afterwards.

Adding rewards for reaching a state, the MP evolves into a Markov Reward Process (MRP). I use $R(s)$ to denote a reward function. Usually, there also is a discount factor $\gamma \in [0, 1]$. It is important to note that the given reward function is constructed to guide towards a goal. However, in many cases there is no guarantee that the given reward function is the optimal one to achieve the goal.
Also there exists a distinction between continuous and episodic problems. The former one describes tasks with no defined ending. On the other side, the latter one has an end. The termination can happen either upon reach a final state or after a certain timestep. In this work, episodic tasks are considered. In this setting, the total discounted cumulative return G_t at timestep t is defined as follows:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \tag{2.1a}$$
$$= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1b}$$

Depending on the choice of γ, one gets a "myopic" or "far-sighted" evaluation. There are multiple reasons to discount, one of which might be that getting rewarded immediately is preferred over getting the same reward later on. Further, by calculating the expected return starting from state s, $v(s)$ represents the state-value function. $v(s)$ is used by the agent to determine the value of a state and thus assists deciding which state it should visit.

The value function can be composed into two parts, the immediate reward r_{t+1} and the discounted value function of the successor state $\gamma v(S_{t+1})$. This recursive deconstruction is called Bellman Expectation Equation.

$$v(s) = \mathbb{E}[G_t | S_t = s] \tag{2.2a}$$
$$= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | S_t = s] \tag{2.2b}$$
$$= \mathbb{E}[r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \dots) | S_t = s] \tag{2.2c}$$
$$= \mathbb{E}[r_{t+1} + \gamma G_{t+1} | S_t = s] \tag{2.2d}$$
$$= \mathbb{E}[r_{t+1} + \gamma v(S_{t+1}) | S_t = s] \tag{2.2e}$$

Finally, a MDP is a MRP that includes decision making. Thus, one allows the agent to make decisions which are called actions. For every state s there exist actions $A(s)$ from which the agent can choose. $A(s)$ can be set for discrete spaces and a distribution for continuous ones.
Figure 2.1 gives a system level view of a MDP. There, the two main entities of a MDP are depicted: the agent and the environment. Tha agent interacts with the environment by choosing actions. In return, the environment emits information about the current state and a reward signal. Eventually, the agent aims to optimize the taken actions given the states by taking into account the feedback from the environment. This thesis works with this kind of framework. However, it is noteworthy, that there exist extensions to MDPs, such as Multi-Agent Environments or Partially Observable MDPs (POMDPs).

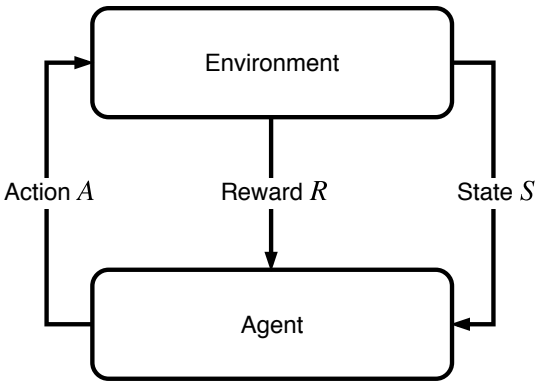


Figure 2.1.: The agent interacting with the environment [SB18]

In a MDP, a policy $\pi : A \times S \rightarrow [0,1], \pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$ describes the behaviour of the agent. Mathematically, it returns that probability that you choose to take the action a when being in state s. The definition above depicts a stochastic policy. There are also deterministic policies $\pi : S \rightarrow A$, which represent functions mapping from a given state s to an action. This means that for deterministic policies no uncertainty is involved.

MDP policies π only depend on the current state s because MDPs, as well as MPs and MRPs, satisfy the Markov Property. It states that the agent's current state is a sufficient statistic of its history, thus the future is independent of the past given the present.

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t] \tag{2.3a}$$

The eventual goal of the agent and thus RL is to maximize the G_t by adjusting the policy π in a way, that higher returns a more likely. One receives the highest possible return if $v^*(s) = \max_\pi v_\pi(s)$ is known which is the maximum value function over all policies. A MDP is considerd being solved when one knows $v^*(s)$.
The following can be implied with regards to an optimal policy π^* for any MDP:

- There exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall \pi$
- All optimal policies achieve the optimal value function, $v_{\pi^*}(s) = v^*(s)$

So far, the focus has been on state value functions $v(s)$ depending on the state s. It is also possible to have an action value function $q_\pi(s, a)$ which takes both state s and action $a$into account. All the described approaches and formulas also apply for $q_\pi(s, a)$. The value functions $q_\pi(s, a)$ and $v_\pi(s)$ differ in when the start following the policy π. $v_\pi(s)$ starts in state s and is guided by π from then on. $q_\pi(s, a)$, however, first takes action a in state s and then starts following the policy π.

2.1.2. Characteristics of Reinforcement Learning

Before discussing the different branches of Machine Learning(ML), a definition of ML is given: "Machine Learning is the study of computer algorithms that improve automatically through experience"[Mit97].
The three approaches in ML are Supervised Learning(SL), Unsupervised Learning (UL) and RL. This thesis focuses on RL. RL has certain characteristics that are set in contrast with SL and UL.

RL, SL and UL all learn from experiences which resemble the input data. Using that knowledge, those techniques aim to be able to generalize, meaning that its predictions are very accurate on unseen data if it comes from the same distribution. However, in RL the distribution is not i.i.d. and changes with each new taken action.

RL also includes exploration. Specifically, one is not bound to the initial, fixed dataset which might be a couple of images and labels as in the case of image classification. Instead, depending on its decisions the agent can find itself in new states, that it has never seen before. Stochastic policies encourage exploration. In RL, one wants to trade off both exploitation of the current knowledge and exploitation of the state space. Exploitation refers to following the policy, that yields the highest return obtained so far. The incentive for exploration stems from the possibility, that there might be a superior policy configuration with an even higher return.

Additionally, reward delay and reward sparsity occur frequently in RL. There is no supervisor giving immediate feedback but only a reward signal which the agent might only receive at the end of the task. The agent then has to figure out which part of its actions is responsible for the low or high cumulative reward respectively; this issue is called credit assignment problem.

In RL, there exist two approaches on how to solve a MDP: value-based and policy-based methods. With the former method, the agent tries to learn a value function and

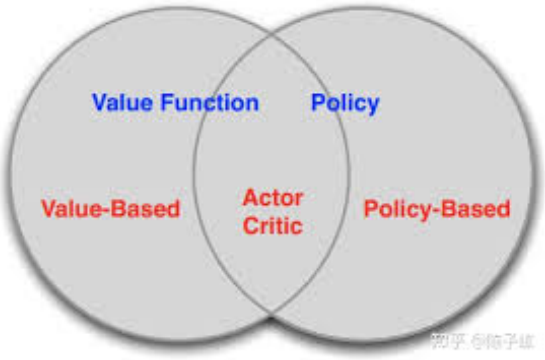


Figure 2.2.: Policy-based and Value-based Methods[Sil15]

makes a decision based on that function. The policy is then derived implicitly from that value function. $\epsilon - greedy$, $\epsilon \in [0, 1]$, is a common choice here. The agent will take the action, that leads to a state with the highest value, with probability 1 - ϵ and a random action with probability ϵ, to encourage exploration.
On the other hand, policy-based methods do not learn a value function. Instead, they try to learn the policy directly, without constructing a value function first.
In the intersection of those two approaches lies the Actor-Critic framework which learns both value function as well as an explicit policy. Figure 2.2 depicts the described relationship.

So far there has been no assumption made, how the π or v are modeled. The simplest case would be to choose a look-up table, in which the respective information for each state is stored. For each state that means the policy of the agent contains the action distribution. The same holds true for value function which stores the value for each state.
This approach has several drawbacks. While tabular approaches might work well in small, discrete domains domains, they are not scalable to huge, continuous spaces because you cannot store an infinite amount of state information. Also, the agent can only retrieve information for known states but cannot make predictions for unseen states.

Hence, the concept of function approximation was introduced to RL, solving both of the aforementioned problems. The idea is to learn a parametrized, approximated function. Hence, there is no need to store values explicitly and the approximated function can return predictions.
The following two sections look into how value-based methods and policy-based

methods make use of that concept.

2.1.3. Value-based Function Approximation

Valued-based methods scale up to higher dimensions by approximating the value function using a parameter vector $\mathbf{w}$.

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s) \tag{2.4a}$$

There exists a variety of value function approximators (VFA) such as Decision Trees, Nearest Neighbours, linear combination of features and NN. The focus will be on differentiable VFA, which can be optimized using gradient-based methods. Combining NN, as DL approach, with RL yields Deep RL (DRL).

I will now consider incremental methods that uses stochastic gradient descent(SGD) to minimize the error between the approximated value function $\hat{v}(s, \mathbf{w})$ and the ground-truth value function $v_\pi(s)$. For the moment it is assumed, that an oracle provides $v_\pi(s)$. The Means Squared Error is used as objective function to quantify the difference between $v_\pi(s)$ and $\hat{v}(s, \mathbf{w})$. In 2.5b, $\mu(s)$ defines a state distribution with $\mu(s) \geq 0, \sum_{s \in S} \mu(s) = 1$. The state space is usually much bigger than the number of weights and thus making the value approximation for one state more accurate, worsens the predictions on other states. $\mu(s)$ captures the visitation pattern or frequency of the policy π. One common measure is the time spent in one state s, hence making its value more precise if visited more often. Intuitively, it is desirable to have the approximation for states to be better if visited a lot. Also, this implies that states, which are ignored for most of the time by the policy π, are of less importance. Thus, if a uniform distribution is chosen for $\mu(s)$ by $\frac{1}{s}$, it states that each state is equally cared about.

$$J(\mathbf{w}) = \mathbb{E}_\pi[(v_\pi(s) - \hat{v}(\mathbf{w}, s))^2] \tag{2.5a}$$
$$= \sum_{s \in S} \mu(s)[(v_\pi(s) - \hat{v}(\mathbf{w}, s))^2] \tag{2.5b}$$

So $J(\mathbf{w})$ represents the error that should be minimized. Thus, this implies to go into the opposite direction of $J(\mathbf{w})$. Hence, one wants to update the parameter vector $\mathbf{w}$ with the negative derivative $-\nabla\mathbf{w}$. Defining the gradient of $J(\mathbf{w})$ with respect to $\mathbf{w}$ as $\nabla_\mathbf{w} J(\mathbf{w})$, $\nabla\mathbf{w} = -\nabla_\mathbf{w} J(\mathbf{w})$ follows. Additionally, adding the step-size parameter α, which intuitively is responsible for the speed of learning, and $\frac{1}{2}$ to take cancel out the square of $J(\mathbf{w})$ when deriving, gives

$$\nabla \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \tag{2.6a}$$

as update for the weights.
Finally, the full update via Gradient Descent is

$$\nabla \mathbf{w} = \alpha E_{\pi}[(v_{\pi}(s) - \hat{v}(\mathbf{w}, s))\nabla_{\mathbf{w}} \hat{v}(\mathbf{w}, s)] \tag{2.7a}$$

Since SGD is used, the formula simplifies to:

$$\nabla \mathbf{w} = \alpha (v_{\pi}(s) - \hat{v}(\mathbf{w}, s))\nabla_{\mathbf{w}} \hat{v}(\mathbf{w}, s) \tag{2.8a}$$

So far it was assumed to get $v_{\pi}(s)$ from an oracle. In the following it will be described how Monte Carlo(MC) policy evaluation, a standard prediction algorithm, handles $v_{\pi}(s)$. MC learns directly from episodes of experience and uses the empirical mean return since it samples from the environment.
Also, MC substitutes $v_{\pi}(s)$ with the the ground-truth return G_t:

$$\nabla \mathbf{w} = \alpha (G_t - \hat{v}(\mathbf{w}, s_t))\nabla_{\mathbf{w}} \hat{v}(\mathbf{w}, s_t) \tag{2.9a}$$

This way, MC updates towards the acual return. Therefore, the training data consists of the tuples of the states and their respective returns: $\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, \ldots, \langle S_{T-1}, G_{T-1} \rangle$.

2.1.4. Policy-based Methods

This thesis follows the convention of [SB18], thus policy-based methods are parametrized θ: $\pi(a|s, \theta)$. There are several merits to the policy-based approach. For example, it is possible to give theoretical convergence guarantees for those kinds of methods, meaning that they will definitely converge to at least a local optima. Additionally, policy-based methods are quite effective in high-dimensional or continuous action spaces and can be simple to represent while it might be more complex to approximate the state value function.

$$\pi_{\theta}(a|s) = \mathbb{P}[A_t = a | S_t = s, \theta_t = \theta] \tag{2.10a}$$

To recap, the policy π_θ is represented by a probability distribution $\mathbb{P}$ over action a depending on state s and parameter vector θ. Intuitively, the agent adjusts P in a way that actions with high rewards are more probable and actions yielding low rewards get less likely. Basically, the agent incorporates knowledge, that helps it getting closer to maximize the return, and discards information that does not add value.

The goal is to optimize for the policy objective function $J(\theta)$ that returns some scalar value $r \in \mathbb{R}$. Different to VFA, where the error is minimized, the cumulative return is maximized directly, meaning gradient ascent is performed instead of gradient descent for the parameter θ. As previously, α is a hyperparameter, that is used to set the learning speed.

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} J(\theta_t) \tag{2.11a}$$

I now consider the one-step case for Monte Carlo Policy Gradient. This one step means, that the agent starts in state s and takes action a, with $R(s,a)$ being the reward function for the respective state and action. $d(s)$ represents the initial state distribution. Since equation 2.16a cannot be sampled , its identity 2.12f is used:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}[R(s,a)] && \text{(2.12a)}\\
&= \nabla_\theta \sum_s d(s) \sum_a \pi_\theta(a|s) R(s,a) && \text{(2.12b)}\\
&= \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(a|s) R(s,a) && \text{(2.12c)}\\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} R(s,a) && \text{(2.12d)}\\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) R(s,a) && \text{(2.12e)}\\
&= \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) R(s,a)] && \text{(2.12f)}
\end{aligned}$$

Since single samples are used, the policy-gradient update is stochastic. Nevertheless, in expectation it equals the actual policy gradient.

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_\theta(a|s) \tag{2.13a}$$

The Policy Gradient Theorem generalizes this approach, replacing the immediate reward R with the long-term value $q_\pi(s,a)$.

Theorem 1 (Policy Gradient Theorem) *For any differentiable policy $\pi_\theta(s,a)$, for any of the policy objective functions $J(\theta)$, the policy gradient is $\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) q_\pi(S,A)]$. [SB18]*

Since Monte Carlo Policy Gradient is considered, the return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$ substitutes $q_{\pi_\theta}(S_t, A_t)$ as an unbiased sample of it.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi_\theta(a|s) G_t \tag{2.14a}$$

The full algorithm for Monte Carlo Policy Gradient looks as follows for episodic tasks and is also called REINFORCE algorithm.

Algorithm 1: REINFORCE

Result: θ
Initialize θ arbitrarily ;
for *each episode* $S_1, A_1, R_2, \dots, S_{T1}, A_{T1}, R_T \sim \pi$ **do**
  for *t=1 to T-1* **do**
    $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta \log \pi_\theta(a|s) G_t$;
  end
end

Although the result is unbiased, it can have high variance because the return is the sum of many random variables $R_t, \dots, \gamma^{T-t} R_T$. Thus, variance reduction methods will be discussed now.
One simple possibility is the introduction of a baseline function $b(s)$. This way one can reduce the variance, without changing the expectation:

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) b(s)] = \sum_{s \in S} d_{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(a,s) b(s) \tag{2.15a}$$

$$= \sum_{s \in S} d_{\pi_\theta}(s) b(s) \nabla_\theta \sum_a \pi_\theta(a,s) \tag{2.15b}$$

$$= 0 \tag{2.15c}$$

Thus, one can write:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)(q_{\pi_\theta}(s,a) - b(s))] \tag{2.16a}$$

There are further improvements possible, for example by using the actor-critic framework[SB18]. To put it into a nutshell, the actor is usually a policy-gradient algorithm and the critic is instantiated by a value-based method, giving some immediate feedback, hence critic, which the actor can use as base of direct improvement. There exist a lot of different and interesting architectures built on top of it, such as the A3C[Mni+16], but I will not go into more detail here.

2.1.5. Case Study: Proximal Policy Optimization

Using the knowledge about policy-gradient methods, this section explains the main algorithm used in this thesis. To be precise, a clipped version of Proximal Policy Optimization algorithms(PPO)[Sch+17] was chosen. PPO actually actually contains a group of different variants of policy optimization algorithms. Those were introduced by Schulman et al.[Sch+17] and at the time of writing this thesis considered state-of-the-art.

There exists a broad range of policy optimization algorithms but PPO stands out for its simplicity, lower sample complexity and ease of tuning.
Using PPO, $log\pi_\theta$ returns the log probabilities from the output of the policy network, while $\hat{A}_t$ is difference between the predicted and actual value of the selected action. $\hat{A}_t$ basically tells whether the taken action was better or worse than expected. This implies that a positive advantage A_t makes those actions more likely by increasing the expected value for the given combination and vice versa for a negative advantage estimate. Mathematically, the advantage is defined as: $A(s,a) = q_{\pi_\theta}(s,a) - v_{\pi_\theta}(s)$.

$$J(\theta) = \mathbb{E}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{2.17a}$$

Instead of taking the log, the authors of the PPO paper use ideas from Trust Region Methods(TRPO)[Sch+15] where they divide the current policy π_θ by the old policy $\pi_{\theta_{old}}$. Here, old means having the parameter vector θ before the update.

$$J(\theta) = \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{2.18a}$$

$$= \mathbb{E}_t[r_t(\theta)\hat{A}_t] \tag{2.18b}$$

The reason for dividing π_θ by $\pi_{\theta_{old}}$ is that you want to avoid making too large policy updates, which might otherwise slow down training or could even cause diverging.

Obviously, if both the old and the current policy of 2.18a are very similiar, $r_t(\theta)$ will be close to 1. Using 2.18b, the authors of the PPO paper add a constraint that the policy update should lay within a certain interval, therefore trust region, yielding the following objective formalization:

$$J^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.19a}$$

with ϵ being the clipping parameter that is usually set to 0.2. Intuitively, this mean that PPO does not allow the new policy to be further away than 20 percent from the old policy in the current update. The authors recommended this exact value of 0.2 because of their empirical results.

Eventually, the min operator sets a lower, pessimistic bound for the expected outcome.

2.2. Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL), also known as Inverse Optimal Control or Inverse Optimal Planning, is a well known concept that gained a lot of attention since the breakthroughs achieved by RL. I aim to explain, formalize as well as differentiate IRL from approaches as RL and IRL.

2.2.1. Intuition

So far, access to a reward function $R(s)$ has been assumed, either handcrafted or given by an oracle. In this section, I want to explain how the reward function is created with the help of demonstrations. The behaviour of the agent is the result of the combination of the interaction with the environment as well as the rewards attained by doing so. This procedure is depicted on the left side of 2.3. In contrast, IRL inverses that process by trying to recover the reward function provided some behaviour inside the environment.

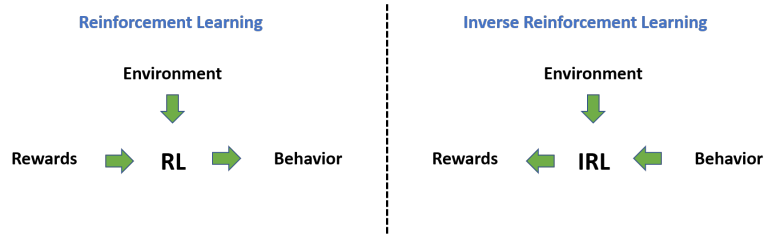


Figure 2.3.: Depiction of the flow in RL and IRL [1]

2.2.2. Formal Definition

I defined the MDP of a RL problem in the last section. Here, I want to clarify how IRL is different from RL. A MDP for RL consists of a set of states S, a set of actions A, a state transition probability matrix P and a reward function R, represented as tuple $\langle S, A, P, R \rangle$ with the goal to learn an optimal policy $\pi^*(a|s)$.
In the case of IRL, the agent do not have access to the reward function R, implying the representation MDP R. Instead, a set of m trajectories $\tau_0, \ldots, \tau_{m-1}$ sampled from a policy π is available. In this work, the agent learns from observations meaning that a trajectory a list of consecutive states of length n $s_0, \ldots, s_{n-1}$. It is not necessary to assume that π is the optimal policy π^* although often it is the case.
Using the above information, one could try to recover a good approximation $\hat{R}_w$ of the the reward function R. One could run RL on top of $\hat{R}_w$ in order to derive a better policy than the original one. To be more specific, one looks for an optimal reward function $\hat{R}^*_w$ that satisfies:

$$\mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R^*_w(s_t)|\pi^*] \geq \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R^*_w(s_t)|\pi] \forall \pi \tag{2.20a}$$

I will describe the example of a feature based reward function:

$$\mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_w(s_t)|\pi] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k w^T \phi(s)|\pi] \tag{2.21a}$$

$$= w^T \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k \phi(s)|\pi] \tag{2.21b}$$

$$= w^T \mu(\pi) \tag{2.21c}$$

where $R(s) = w^T\phi(s), w \in \mathbb{R}^n$ and $\phi : S \to \mathbb{R}^n$. μ is the expected cumulative discounted sum of feature values or "feature expectations". Substituting equation 2.21c

into equation 2.20a, one gets 2.22a as objective:
Find w^* such that

$$w^{*T}\mu(\pi^*) \geq w^{*T}\mu(\pi)\forall\pi \tag{2.22a}$$

I will focus on feature-based IRL. Also, the number of demonstrations provided by an expert required scales with the number of features in the reward function. Additionally, the amount of expert demonstrations needed does not depend on the complexity of the expert's optimal policy π^* nor the size of the state space S.

2.2.3. Role of Imitation Learning

I also briefly want to touch on the distinction between IL and IRL. While IL tries to directly learn a policy from observations, and optionally actions, IRL intends to infer a reward function. Note that IRL on its own does not imply finding an optimal behaviour but instead returns $\hat{R}$ as result. As mentioned above, solving the MDP can be achieved by using a RL algorithm that learns using $\hat{R}$.

One of the most famous and also simplest applications of IL is Behavioural Cloning (BC). It uses supervised learning on state-action pairs (s, a) as training data where a represents the action that was taken in state s. The goal is to learn a mapping from state to action. Hence, the state is input data and the respective action the label. Despite its simplicity, BC can perform well on certain tasks. An important example application is ALVINN [Pom88] where its authors taught a vehicle to drive autonomously. It was trained using a NN and after training it utilized its sensors to process the environment information in order to move forward.

The main problem BC faces is that once it encounters samples not given in the training dataset and therefore decides to take an unseen action, the errors that follow might compound, making the agent to completely fail the task. Figure 2.4 illustrates BC quite well.

As said, the basic BC procedure is appealing because of its simplicity. It consists of three major steps [Pom88]:

1. Collect demonstrations τ^* from an expert policy π^*
2. Treat the demonstrations as i.i.d. state-action pairs $(s_0^*, a_0^*), (s_1^*, a_1^*, \ldots)$
3. Learn π_θ using supervised learning by minimizing the loss function $J(a^*, \pi_\theta(s))$

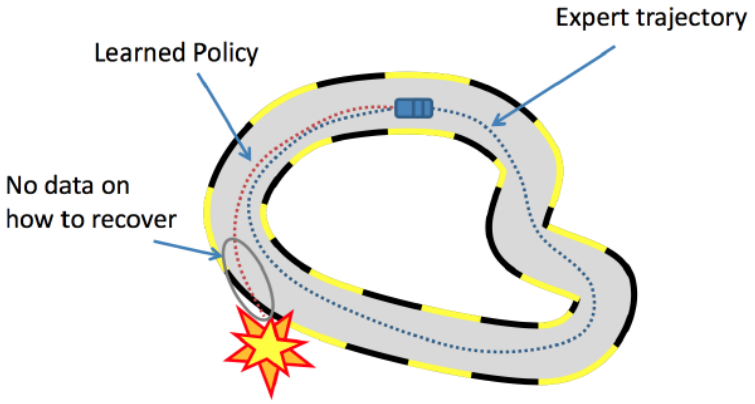


Figure 2.4.: Components of BC [2]

IL is still a field with a lot of active research going on. Some scientists extended BC to Behavioural Cloning from Observations [TWS18] which delivers competitive results while not requiring access to action labels. Ho et al. applied the GAN learning technique [Goo+14] to IL, yielding Generative Adversarial Imitation Learning [HE16].

2.3. Learning to Rank

Learning to Rank aims at solving ranking tasks by using ML techniques. Ranking is critical for information retrieval problems like document retrieval. It also has been applied successfully to other research areas such as Natural Language Processing [DK08] or Recommender-Systems[Lv+11].

In general, there exist three categories of ranking: pointwise, pairwise and listwise which were proposed by Tie-Yan Liu in his paper "Learning to Rank for Information Retrieval"[Liu09]. There exist numerous applications of pointwise[LBW08] [CGD92], pairwise [Che+10] [OAD18] and listwise approaches [Sta+19]. In this thesis, pairwise and tripletwise approaches are considered.

Specifically, I will now introduce the rankings and the corresponding objective functions that I use for the learning process.

2.3.1. Pairwise-Ranking

Intuitively, there are two entities with each having a distinct utility. That utility is used to determine which entity is superior. Within one pair, the entity with the higher utility receives the label one, while the other one gets the label zero. Hence, opting for the Binary Crossentropy (BCE) loss is a natural choice. The training goal can be described as identifying as many higher ranked items as possible.
In my implementation I use the Binary Crossentropy function provided by PyTorch which is defined in 2.23a. For a single pair, y_i represents the true label, also called target, and $\hat{y}_i$ is the prediction of the NN with respect to its input. Let the whole input data be x and the NN be a function $f : x \to \hat{y}$, then $\hat{y}$ is the set of predictions $\hat{y_1}, \hat{y_2}, \ldots$ which are compared to true labels y.

$$J^{BCE}(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i) \tag{2.23a}$$

2.3.2. Tripletwise-Ranking

The kind of ranking I explain here is inspired by the Tripletloss, first introduced by Schroff et al. in their paper "FaceNet: A Unified Embedding for Face Recognition and Clustering" [SKP15].
Figure 19 gives an intuitive overview of the Triplet Loss for image verification and recognition. As the name suggests, this loss function takes three inputs: an anchor a, a positive p and a negative n. In the context of images as domain, the anchor and positive are the same entity but in different variations, e.g. postures and angles. The negative refers to another entity. The Tripletloss is defined as follows:

$$J^{Triplet}(a, p, n) = \max(d(a, p) - d(a, n) + margin, 0) \tag{2.24a}$$

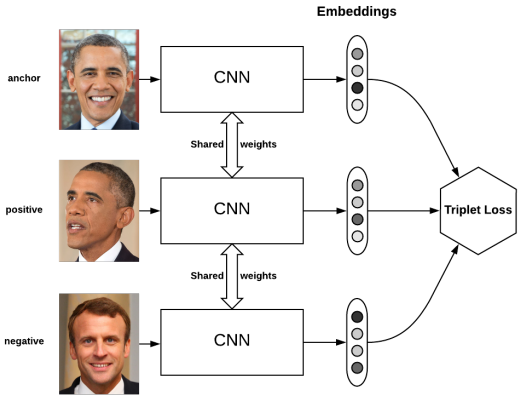


Figure 2.5.: Triplet loss on two positive faces (Obama) and one negative face (Macron) [3]

2.3.3. Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) is a ranking metric that operates on a list of entities and their respective relevance. There exist other common metrics such as Mean Reciprocal Rank and Mean Average Precision (MAP). NDCG was chosen because it can work on lists and allows fine-grained relevance values while MAP only can make binary distinction between relevant and non-relevant.

NDCG is based on the Cumulative Gain (CG) which just adds up all the relevance scores rel_i of all entities in the list 2.25a. By itself, CG does contain information about the position of the entities that the relevance scores are derived from. The parameter p tells how many entities are taken into consideration for NDCG. In this work, p is also the length of the whole list.

$$CG_p = \sum_{i=1}^{p} rel_i \tag{2.25a}$$

To incorporate the possibility to evaluate with regards to the positions of the entities, discounting was introduced. Thus, if an entity with a higher relevance appears early in the list, it is higher valued than if it appeared later on. The discount factor depends on the position i.

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} \tag{2.26a}$$

One major problem so far is that DCG is not normalized and thus lacks comparability. Therefore, the result of the DCG score of the relevances is divided by the optimal DCG score DCG_p^*, meaning the highest possible score for the list of entities. The result, the Normalized Discounted Cumulative Discount (NDCG) score, is bound between 0 and 1 and therefore can be used for comparisons.

$$NDCG_p = DCG_p / DCG_p^* \tag{2.27a}$$

2.4. Simulation

In this work I use the physics engine Mujoco [TET12], which is an abbreviation for Multi-Joint dynamics with Contact. It provides the possibility to assemble and test agents in a simulated environment. In the field of RL, Mujoco is commonly used to simulate standard agents to compare various RL algorithms and thus serves as benchmark. The snake-like robot agent is a custom model and will be described now in more detail.

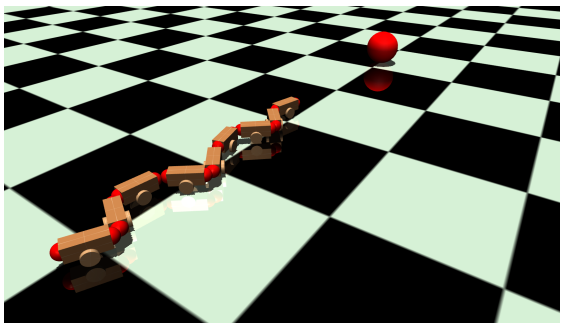


Figure 2.6.: The snake in the simulation environment)

Specifically, I will explain how my setup relates to the MDP framework introduced in Section 2.1.1.

2.4.1. Observation Space

The agent consists of nine building blocks which are connected via eight hinge joints. Each joint connects two blocks, thus obtaining the snake-like robot depicted in figure 2.6.
The observation space corresponds to the state information in a MDP. It is crucial to have an understanding of its components since it represents the input for the reward function approximator.
At each timestep the environment returns an array of length 27 with 26 degrees of freedom (DOF) with the last element of that array being the target velocity, which is actually a hyperparameter I introduce.
During the initial training procedure on the ground truth reward this value increases from 0.05 to 0.25 m/s. As training progresses, it will be incremented by 0.05 m/s until it reaches the maximum velocity of 0.25 m/s. Especially during the first 100 episodes the velocity is kept at 0.1 m/s to allow easier learning by keeping the target constant.
The red sphere is always moving if the agents moves. Nonetheless, it is necessary for the

agent to know the direction of where to move to. One also receives the current position for each joint, resulting in eight additional values. The same applies to the speed for each joint, making it 16 values in total. Since I also track the energy consumption, eight values for the sensor actuator frictions are obtained. In addition, one gets the sensor velocity and the angle between the desired direction and the direction the snake is moving towards.
In total, there are 27 values for one observation per timestep.

2.4.2. Action Space

Similarly to the observations space, the actions per timestep are also related to the joints. Each of the 8 values which are returned every step by the environment corresponds to the actuator angle positions of the servo motors. To keep the movement between -90 and 90 degree when facing the red sphere, the actions are clipped. This In Mujoco, this translates to a range of [-1.5, 1.5].

2.4.3. Reward Function

In this subsection I clarify how the introduced components are put together to calculate the reward for velocity and power efficiency. The formulas were developed by Bing et al. [Bin+20].
The first component is the velocity reward r_v. minimizing the distance between those two metrics is the optimal behaviour.
As can be seen in 2.28a, the velocity reward r_v increases if the current velocity v_1 is closer to the target velocity v_t. The maximum attainable reward is 1, for the case that both velocities are identical. In this work as well as the thesis built upon, the hyperparameters a_1 and a_2 were both set to 0.2. Please refer to Lemke et al. [LB18] for a detailed explanation of this choice.

$$r_v = (1 - \frac{|v_t - v_1|}{a_1})^{\frac{1}{a_2}} \tag{2.28a}$$

The second term of the main equation depends on the power consumption, yielding the reward r_p. The power consumption is normalized and depends on the hyperparameter b_1 and is constrained by r_{max}. b_1 equals 0.6. This choice reflects the empirical results by Lemke et al. [LB18].

$$r_p = r_{max}|1 - \hat{P}|^{b_1^{-2}} \tag{2.29a}$$

Combining equation 2.28a and equation 2.29a, one obtains the composed reward r which returns a scalar value $r \in \mathbb{R}$ for every timestep.

$$r = (1 - \frac{|v_t - v_1|}{a_1})^{\frac{1}{a_2}} |1 - \hat{P}|^{b_1^{-2}} \tag{2.30a}$$

3. Related Work

This chapter describes the previous problems and approaches with regards to the snake environment. Also, contributions and methods of the papers, that inspired to utilize IRL from preferences, are also presented.

3.1. Snake-like Robots

3.1.1. Reinforcement Learning for Autonomous Locomotion Control of Snake-Like Robots

In his master thesis, Christian Lemke [LB18] focused on tackling locomotion tasks inside the custom snake environment. The emphasis of this subsection is to explain his setup and results as well as the differences to the approaches and targets of my work.

Christian Lemke has made two major contributions: first, he trained a snake-like robot on forward locomotion using RL techniques and compared the outcome against a traditional equation controller. As the name of the latter method suggests, it is based on a mathematical model that takes several input parameters such as angular frequency, amplitude and bending radius.
In order to find an optimal solution for the equation controller, a grid search is performed on a range of parameters. For the RL controller, Lemke chooses the PPO algorithm to optimize the policy π. The policy is trained with respect to the ground-truth reward r, that was described in 2.30a. Thus, his evaluation is two-dimensional, depending on the target velocity and the power consumption. He trains the PPO controller in a way, that it starts with a slow target velocity which keeps increasing with a certain number of timesteps. This way he obtains different gaits with a wide range of velocities. The target velocities are chosen empirically.
His results indicate that the PPO controller outperforms the traditional controller by achieving a lower energy consumption for a broad range of velocities.
The second task requires the snake to follow a red sphere which moves in a straight line. The eventual goal is to keep a certain distance between itself and the red sphere. Thus, the snake should stop, when the sphere stops, and continue to move forward otherwise. In order to locate the target, the snake is equipped with a head camera,

$$R \rightarrow_{ppo} \pi \rightarrow_{\sim} (\tau_0, \tau_1, \ldots) \in D \rightarrow_{\theta} \hat{R} \rightarrow_{ppo} \hat{\pi}$$

Figure 3.1.: Approach by Lemke (black) and what was added by this work (red)

enabling it to have vision of its front. Lemke defines the target distance to be around four meters. Thus, if the agent is exactly four meters away, it obtains the maximum reward for that timestep. Additionally, he constrains the deviation to be around 2 meters, thus the distance $d \in [2, 6]$ meters is clipped. Formula 3.1a is used by Lemke to calculate the attained distance reward, with t_d being the target distance, d_r the range from the target distance towards the maximum and minimum distance respectively. And finally d_{before}/d_{after} is the actual distance before/after the taken action.

$$r^d = (1 - \frac{|t_d - d_{after}|}{d_r}) - (1 - \frac{|t_d - d_{before}|}{d_r}) \tag{3.1a}$$

$$= \frac{|t_d - d_{before}| - |t_d - d_{after}|}{d_r} \tag{3.1b}$$

His findings suggest that the snake is capable of keeping a certain distance without too much oscillation, in particular the standard deviation is less the 0.25 meters for all the predefined tracks that he used. A track is a route the red sphere takes and thus the path the agent also follows.

After presenting his experiments and outcomes, a brief analysis of the differences and the components, that have been re-purposed for this work, is given. Figure 3.1 illustrates this contrast.

The following breaks down figure 3.1: First a policy π is obtained using PPO on the ground-truth reward function R. But while Christian Lemke stops here, in this work strictly ranked trajectories τ are created as dataset D by π. Next, a synthetic reward function $\hat{R}$ is constructed using a NN, parametrized by θ. If $\hat{R}$ managed to grasp the intended goal, the synthetic policy $\hat{\pi}$ moves forward efficiently. Again, PPO is utilized for the policy optimization.

Christian Lemke worked quite empirically to find a reward function that shows the desired behaviour. This includes the construction of the ground-truth reward function as well as parameters directly related to the policy optimization. Those parameters are justified by his empirical results. Thus, he heavily engages in reward shaping. My

work aims to get rid of this explicit reward signal construction and instead use a NN as black box.

3.2. Inverse Reinforcement Learning

This section is split into two parts: the first paper, T-REX [Bro+19], the authors present their approach that uses IRL from preferences. In their follow-up paper, D-REX [BGN19], they give more background theory on preference-based IRL.

3.2.1. T-REX

T-REX is an abbreviation for **T**rajectory-ranked **R**eward **EX**trapolation. The name was coined in the paper titled "Extrapolating Beyong Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations" by Daniel Brown et al. [Bro+19].

Their underlying consideration is that, unlike most other IRL or IL approaches, it is more sound to understand the intentions of the demonstrator instead of just learning a reward function, that explains the behaviour of the policy, given in the form of demonstrations.
Brown et al. focus on learning from suboptimal demonstrations. Suboptimal demonstrations are demonstrations generated by a policy that returns a lower cumulative reward than other available policies which are not used. This is different to existing IRL approaches. Also, often a suboptimal reward function is learnt, when trained on suboptimal observations [RA07]. Similarly, the performance of IL also deteriorates if you do not use optimal trajectories as training input. As training data, Brown et al. used pairs of subtrajectories with a preference label indicating the superior subtrajectory. Those subtrajectories were taken from longer trajectories, which were generated by a RL agent. In their case, the PPO algorithm [Sch+17] was optimizing the policy of the agent, which was checkpointed every 5000th timestep to have policies of varying quality which then interacted with the environment for 1000 timesteps.
Thus, the subtrajectory with the higher timestep and the one with the lower timestep were labeled respectively. One subtrajectory consists of 50 up to 100 observations, depending on the environment and environment type.

Given the training data described above, they perform a forward pass through a NN, which returns a prediction of each state contained in the subtrajectory. Considering the Mujoco environment as an example, they obtain 50 predictions per subtrajectory, which are then summed and can be viewed as predicted cumulative return. It is worth

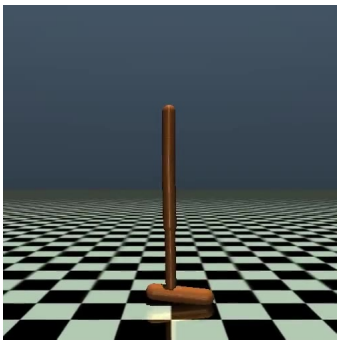
noting that Brown et al. do not use an output function, meaning those values are the raw logits.

As T-REX works on pairs, it applies this procedure to both subtrajectories. Thus, it is a pairwise LTR problem. Those two unnormalized scalars are then fed into the loss function, which compares whether the subtrajectory, that is estimated higher by the network, is actually the superior subtrajectory by using the preference label as target. They used the cross-entropy loss as objective function to minimize the cost. Hence, this way they transform the task into a supervised classification problem.

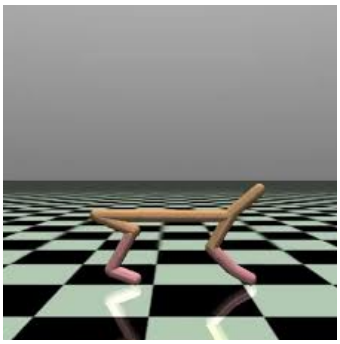
For benchmarking, they used several high-dimensional control tasks for forward locomotion: they tackled HalfCheetah, Hopper and Ant as well as Beam Rider, Breakout, Enduro, Hero, Pong, Q*bert, Seaquest, and Space Invaders from the Mujoco and Atari respectively.
They compared their results against Generative Adversarial Imitation Learning [HE16] and Behavioral Cloning from Observation [TWS18]. Both are state-of-the-art IL algorithms. Brown et al. figured that T-REX performs at least on par for most environments and also manages to outperform the benchmark algorithms in some environments as HalfCheetah for example. For more details, it is recommended to go through the results and appendix of the T-REX paper [Bro+19].
In conclusion, the advantages of the T-REX framework are as follows: it is not required to solve any inner MDP, which makes the approach less computationally expensive. There is no need for action labels, plus it works with relatively few subtrajectories as training data and is applicable to high-dimensional tasks.

Despite those contributions, the authors give no clarification why they just analyzed the pairwise approach. Therefore, there is no information how the performance might increase or deteriorate, depending on the ranking method. I want to tackle this shortcoming in this thesis, and also motivate further research in learning to rank methods in the field of IRL.

Some screenshots of the robot-related environments are provided in figure 3.2.



(a) Hopper Environment



(b) HalfCheetah Environment

(b) Ant Environment

Figure 3.2.: The Hopper, HalfCheetah and Ant Mujoco Environment

3.2.2. D-REX

D-REX [BGN19] is a follow-up paper on T-REX by Brown et al., which consists of two major contributions: they propose **D**isturbance-based **R**eward **EX**trapolation, a ranking-based Imitation Learning method, and they present theory on how using preferences can mitigate reward ambiguity in IRL and a condition when better-than-demonstrator performance is possible.

D-REX is an IL approach, that uses Behavioural Cloning as well as IRL from Preferences. D-REX, in comparison to T-REX, is a broader framework.

D-REX consists of four steps: First, the authors generate unlabeled trajectories and run BC on top to learn an imitation policy.
Second, they inject different levels of noise into the policy, which they then use to create trajectories. The trajectories of a noisier policy are ranked worse than the ones with less noise. Noisy means the learnt policy is masked by an ϵ-greedy strategy where $\epsilon \in [0, 1]$ and the action proposed by π_{BC} is taken with probability $1 - \epsilon$ and a random action with probability ϵ. The higher ϵ, the higher the noise.
Third, one learns a preference-based reward function; here they do this by using the T-REX framework.
And finally, one can choose a suitable policy optimization algorithm to tweak the policy π. They opted for the PPO algorithm.

The rest of this section deals with the theory on extrapolation they presented. Intuitively, there are some requirements, that need to be fulfilled in order to be able to get a good approximated reward function $\hat{R}$.
First, the error between the approximated reward function and the ground-truth reward function needs to be as small as possible. Also one wants $\hat{\pi}$ to be very similar to π when solving the MDP. Additionally, when aiming to extrapolate beyond the performance of the initial policy π, then π needs to be sufficiently suboptimal.

Equation 3.2b shows, that in a MDP, where one can find an optimal policy, one might still not be able to obtain a reasonable synthetic reward function. It is even proven, that a recovery is impossible without additional side-information or human supervision [AM19].
One such example is the all-zero reward function, which as a consequence lets every policy appear optimal.
In equation 3.3a, one can see how the approach by Daniel Brown et al. helps to mitigate that problem. The usage of ranked trajectories avoids reward functions like the one returning zero for every step. Last but not least, ranking not only gives feedback on

what to do but also on what to avoid.

Proposition 2 *There exist MDPs with true reward function R, expert policy π_E, approximate reward function $\hat{R}$, and non-expert policies π_1 and π_2, such that*

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|R) and J(\pi_1|R) \ll J(\pi_2|R) \tag{3.2a}$$

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) and J(\pi_1|\hat{R}) = J(\pi_2|\hat{R}) \tag{3.2b}$$

However, enforcing a preference ranking over trajectories, $\tau^* \succ \tau_2 \succ \tau_1$, where $\tau^* \sim \pi^*$, $\tau_2 \sim \pi_2$ and $\tau_1 \sim \pi_1$, results in a learned reward function $\hat{R}$, such that

$$\pi_E = \arg\max_{\pi \in \Pi} J(\pi|\hat{R}) and J(\pi_1|\hat{R}) < J(\pi_2|\hat{R}) \tag{3.3a}$$

4. Approach

The goal of this thesis is to learn forward locomotion for a simulated snake-like robot. The taken approach consists of three steps:

1. The policy of an RL agent gets optimized using PPO [Sch+17]. During this learning process, the policy is checkpointed after some timesteps. For each of those policies of varying qualilty, demonstrations are generated.
2. Those demonstrations are labeled and combined to tuples of different length, depending on the ranking approach. With this typcial supervised learning setup, a NN is trained on those demonstrations and labels. This NN represents a synthetic reward function, mapping states to rewards.
3. Lastly, a policy is trained on this learnt reward function, again using PPO. Thus a comparison to the ground-truth can be drawn. Additionally, the forward movement of the agent can be examined visually.

Each following section contains a figure displaying the procedure, accompanied by an in-depth explanation.

4.1. Data Generation

Figure 4.1 shows the data generation procedure.

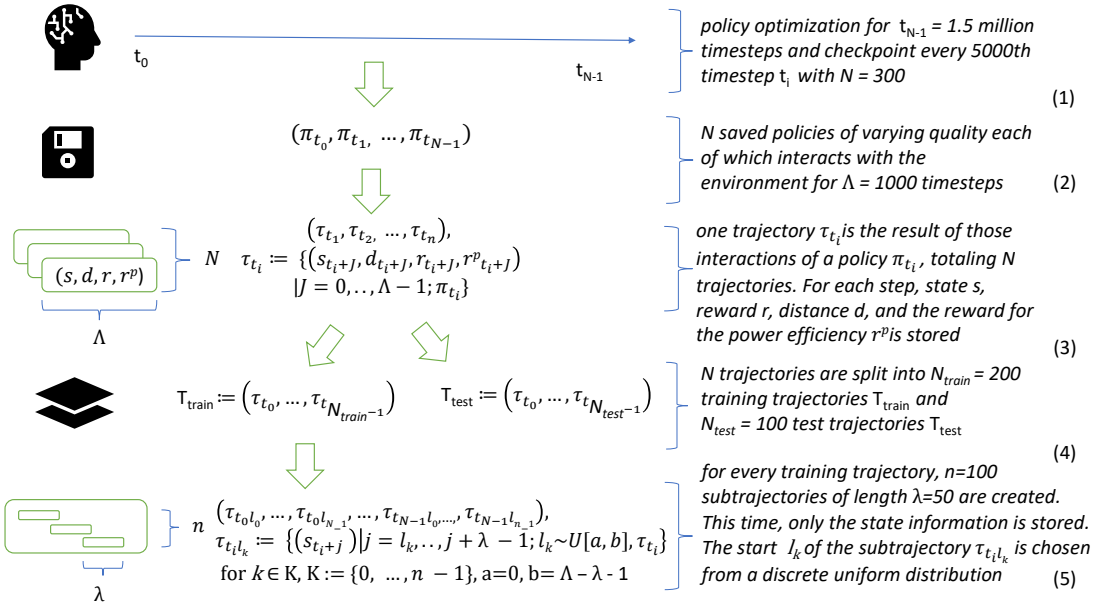


Figure 4.1.: Data Generation

First, the policy π is trained on the ground-truth reward R for 1.5 million timesteps (1). During the training, π is checkpointed every 5000th timestep t_i. Thus, N=300 different policies $\pi_{t_0}, \ldots, \pi_{t_{N-1}}$ of varying quality are obtained, $i \in \{0, \ldots, N-1\}$.

Subsequently, each of those policies is allowed to interact with the environment for one episode which lasts for Λ=1000 timesteps (2). For every timestep, the observation s (2.4.1), the traveled distance d in meters, the total reward r (2.30a) and the reward for the power consumption (PC) r^p (2.29a) are stored. Thus, four lists of length Λ are obtained (3).

Those 300 trajectories $(\tau_{t_0}, \tau_{t_1}, \ldots, \tau_{t_{N-1}})$ are split into training T_{train} and test data T_{test}. The first N_{train}=200 trajectories, $(\tau_{t_0}, \tau_{t_1}, \ldots, \tau_{t_{N_{train}-1}})$, are used for training. Hence, the other N_{test}=100 trajectories, $(\tau_{t_{200}}, \tau_{t_{201}}, \ldots, \tau_{t_{N_{train}+N_{test}-1}})$, are used for the evaluation (4). The terms test and extrapolation data will be used synonymously.

Name	Descriptions	Shape
s	Observations per Step	(1000, 27)
d	Distance per Step	(1000, 1)
r	Total Reward per step	(1000, 1)
r^p	Reward for PC per step	(1000, 1)

Table 4.1.: Information contained in one Trajectory τ_{t_i}

Next, partial trajectories $\tau_{t_i l_k}$, alias subtrajectories, are taken from each trajectory τ_{t_i} of the training set T_{train}. From each trajectory τ_{t_i}, n=100 subtrajectories $\tau_{t_i l_k}$ of length λ=50 are sampled (5). l_k is an integer, represent the starting point of $\tau_{t_i l_k}$ inside τ_{t_i}. It is drawn from a discrete uniform distribution $U[a, b]$ with $a = 0, b = \Lambda - \lambda - 1$. b is the latest point of time a subtrajectory $\tau_{t_i l_k}$ can start within one trajectory τ_{t_i}. $k \in K, K := \{0, n-1\}$ indicates the subtrajectory τ_{t_i, l_k} it belongs to.
For the subtrajectories, only the s will be stored. The other information, like d, r and r^p will be discarded. That is because only the state information is needed for the preference training.
The information like the total reward r and distance d will be used for the evaluation of the experiments. Table 4.2 lists the information in one subtrajectory τ_{t_i, l_k}.

Name	Descriptions	Shape
s	Observations per Step	(50, 27)

Table 4.2.: Information contained in one Subtrajectory τ_{t_i, l_k}

The total number of subtrajectories τ_{t_i, l_k} can be computed by $|K| \times |T_{train}|$, thus 20000.

4.2. Data Labeling

The data creation so far has been the same for both the pairwise and tripletwise approach. This section explains the different data labeling steps. Especially, $label_{triplet} : \omega \times \Omega \rightarrow \{0,1\}$ takes a set Ω of timesteps and ω, the timestep for which the label is requested. $label_{pair} : \omega \times \Omega \rightarrow \{0,1\}$ shows the function working on pairs

$$label_{pair}(\omega, \Omega) = \begin{cases} 1 & if(\omega = max(\Omega)) \\ 0 & else \end{cases} \tag{4.1a}$$

while $label_{triplet}$ is applied on triplets.

$$label_{triplet}(\omega, \Omega) = \begin{cases} 1 & if(\omega = max(\Omega)) \vee \\ & |\omega - max(\Omega)| < |\omega - min(\Omega)|) \\ 0 & if(\omega = min(\Omega)) \vee \\ & (|\omega - max(\Omega)| > |\omega - min(\Omega)|) \end{cases} \tag{4.2a}$$

Figure 4.2 visualizes this process.

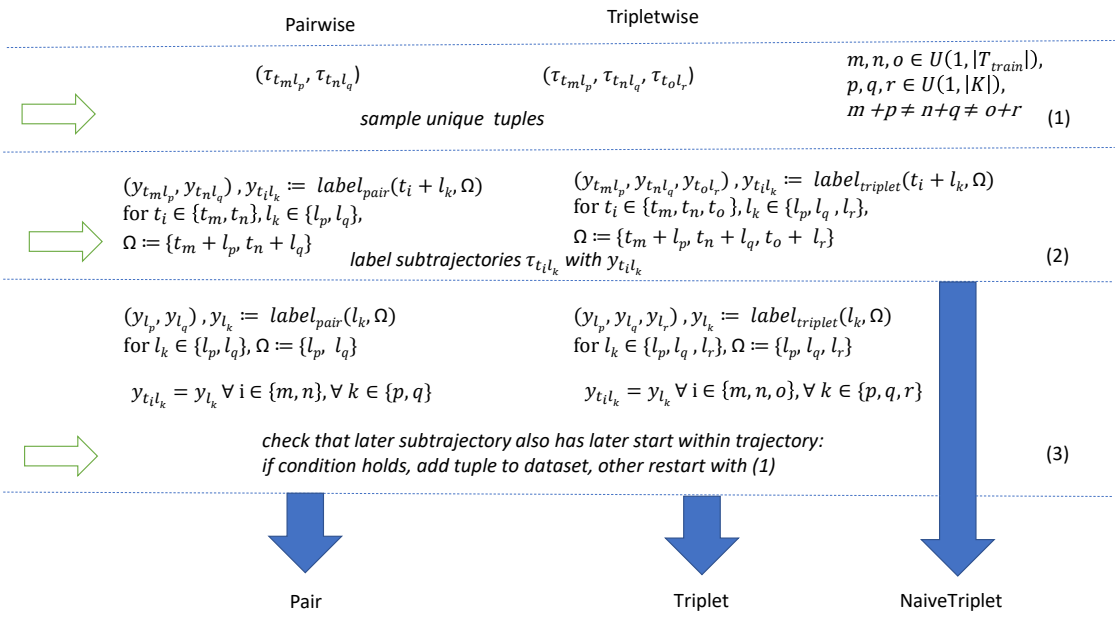


Figure 4.2.: Data Labeling

All phases will be described for the tripletwise approach. The pairwise method follows analogously for two subtrajectories instead of three. First, some subtrajectories $\tau_{t_m,l_p}, \tau_{t_n,l_q}, \tau_{t_o,l_r}$ are taken from the training dataset (1). Thus, three indices $m, n, o \in U[1, |T_{train}|]$, representing the timestep t when the policy π_t was saved. Also three further indices $p, q, r \in U[1, |K|]$, being the start of τ_{t_i,l_k} inside τ_{t_i}, are sampled from U, a discrete uniform distribution. This procedure is repeated until the subtrajectories are unique, identified by its total timestep which is received by adding both indices t_i and l_k.

Next, the subtrajectories get labeled via $label_{triplet}$. The goal is to label the later subtrajectory as one, and the one the the lowest total timestep with zero. This is decided by the left clause of the $\vee$ operator. If the subtrajectory is neither the one with the maximum nor the minimum total timestep, then the absolute distance regarding the total timestep is calculated toward the maximum and minimum total timestep. If the remaining subtrajectory is closer to the one with the highest total timestep, it obtains the label one, and vice versa.

The third phase checks that the labeling of t_i, l_k matches the labeling l_k (3). This way it is prevented that a subtrajectory τ_{t_i,l_k} with higher total timestep $t_i + l_k$ but lower episode start l_k gets included in the final dataset. This is necessary since the beginning of a trajectory might be worse than the end of the trajectory, even if the latter one is created by a worse policy.

The approach Pair and Triplet have all three phases while NaiveTriplet skips this check.

The number of training tuples in the final dataset is the total number of subtrajectories, 20000, divided by the tuple length. In the pairwise case one thus obtains 10000 tuples and 6667 in the triplewise.

4.3. Preference-Learning

Again, the examples are given for the triplet case: the final training data set is a list of tuples of subtrajectories $(\tau_{t_m,l_p}, \tau_{t_n,l_q}, \tau_{t_o,l_r})$ and labels $(y_{t_m,l_p}, y_{t_n,l_q}, y_{t_o,l_r})$ Further, this list is split into a training and validation set; the chosen ratio is 0.9/0.1. Figure 4.3 visualizes the forward pass of one tuple.

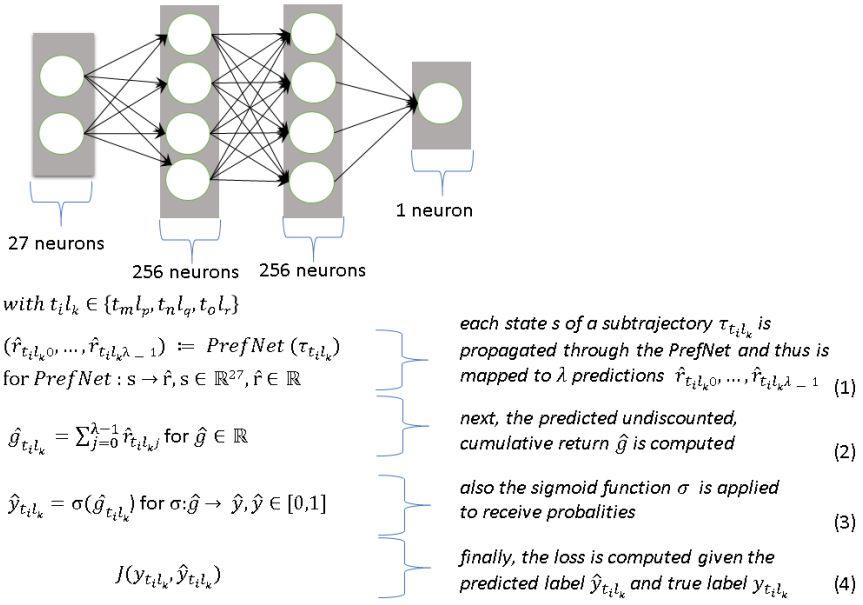


Figure 4.3.: Preference Learning for one triplet $(\tau_{t_m,l_p}, \tau_{t_n,l_q}, \tau_{t_o,l_r})$

First, the trajectories are fed forward through a NN, called PrefNet $\hat{R}$ (1). PrefNet has two hidden layers with 256 neurons each and ReLU non-linearities in between. The network has 27 input neurons which matches the dimension for one state s.
It outputs one single, unnormalized scalar $\hat{r} \in \mathbb{R}$, meaning no output function is applied. Thus, every subtrajecory gets mapped to a vector of raw logits of length $\lambda = 50$.

Next, those values in one vector are summed up, representing the predicted cumulative return $\hat{g}$ for that subtrajectory (2). This is done for every subtrajectory in the tuple $(\tau_{t_m,l_p}, \tau_{t_n,l_q}, \tau_{t_o,l_r})$.

Next, the sigmoid function $\sigma : \hat{g} \rightarrow \hat{y}$ is applied on the cumulative returns each to receive probabilities (3).

Eventually, the loss between the predictions and the labels is calculated (4). The intuition by making this task a multilabel-classification problem is that the PrefNet learns which features are important for the quality of a state s.

All hyperparameters are listed in appendix A.2. Those are the ones that worked best empirically. It might be possible that there are even better hyperparameter configurations.

4.4. RL on learnt Reward Function

This section closes the cycle of how the learnt reward function $\hat{R}$ is actually used to learn a policy $\hat{\pi}$. The setting itself is quite similar to 4.1.
Primarily, one has to substitute the ground-truth reward function R with $\hat{R}$ which is explained in the following.

In the illustration 4.4, the first step is that the policy $\hat{\pi}$ acts in the environment (1). The environment then replies with returning the ground-truth reward r and the observation s (2). This information gets intercepted by a proxy, called reward wrapper. Implementation-wise, it lets one manipulate incoming information. In particular, the state observation s can be given to the synthetic reward function $\hat{R}$ to obtain a predicted reward for that state (3). Upon receiving the prediction, the reward wrapper substitutes the ground-truth reward with the predicted reward and forwards it to the policy $\hat{\pi}$, together with the unchanged, original state information s (4). This procedure is in contrast to the normal policy optimization procedure where the ground-truth reward is used, depicted in figure 2.1 in section 2.1.1.

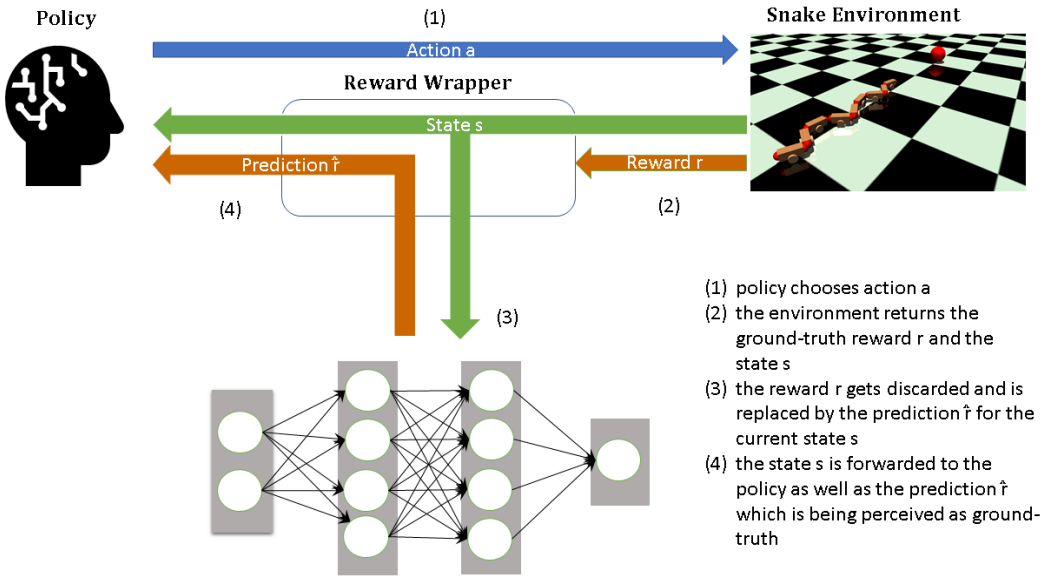


Figure 4.4.: The reward masking

5. Experiments and Results

In this chapter, three experiments are conducted. Each will compare the Pair, Triplet and NaiveTriplet approach. The first experiment measures how well the network was able to learn to rank. The second investigates the performance of a policy that is optimized on the learnt reward function, and third, how strongly the ground-truth reward correlates with the the network predictions. Note that only network are considered in the evaluation on where policy could be learnt that made the agent move for at least one meter when run for 1000 timesteps. This way, one can rule out those reward functions that make the agent block itself. The best policy, trained on the ground-truth reward, moves forward for around 6 meters.

5.1. Network Ranking

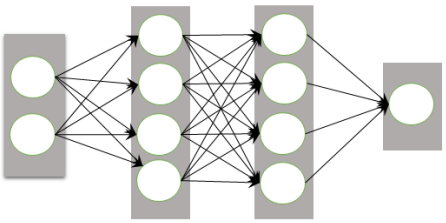
The networks trained with Pair, Triplet and NaiveTriplet approach are compared here and will be refered to as Pairnet, Tripletnet and NaiveTripletnet. This experiment presupposes that steps (1,2) are finished, thus a trained network is available. Step (3) is not necessary for this experiment.

NDCG (2.27a)is used as metric.

The optimal ranking can be derived from the indices $i \in 0, ..., N-1$ of each timestep t_i of the respective trajectory τ_{t_i}. The trajectory generated by the earliest checkpointed and thus worst policy π_{t_0} is τ_{t_0}. Hence, the index 0 indicates the trajectory for which one would also expected the worst reward prediction. Vice versa, the index $N-1$ of $\tau_{t_{N-1}}$ represents the trajectory that should receive the best ranking. 5.1a shows the list with the optimal ranking Φ^* of the indices $i \in \Phi^*$ of the timesteps $t_0, \ldots, t_{N-1}$ of the trajectories $\tau_{t_0}, \ldots, \tau_{t_{N-1}}$ as list (1).

$$\Phi^* = [0, 1, \ldots, N-1] \tag{5.1a}$$

The predicted ranking $\hat{\Phi}$ will be derived from the cumulative reward predictions



$\tau_{t_i} \in \{T_{train} \cup T_{test}\}, i \in \Phi, \Phi \coloneqq (0, \dots, N-1),$
$\tau^s_{t_i} = \{(s_{t_i+J})|J = 0, \dots, \Lambda - 1; \tau_{t_i}\}$

For every trajectory τ_{t_i} from the whole set of trajectories, state-only trajectories $\tau^s_{t_i}$ are created (1)

$(\hat{r}_{t_i 0}, \dots, \hat{r}_{t_i \Lambda\_1}) \coloneqq PrefNet(\tau_{s_{t_i}})$
for $PrefNet : s \to \hat{r}, s \in \mathbb{R}^{27}, \hat{r} \in \mathbb{R}$

Each of the Λ states of the state-only trajectory $\tau_{s_{t_i}}$ is forwarded through PrefNet (2)

$\hat{g}_{t_i} = \sum_{j=0}^{\Lambda-1} \hat{r}_{t_i j}$ for $\hat{g} \in \mathbb{R}$

Next, the predicted undiscounted, cumulative return $\hat{g}$ is computed (3)

$\hat{\imath}_{t_i} = \chi(\hat{g}_{t_i})$ for $\chi: \hat{g} \to \hat{\imath}$,
$\hat{\imath} \in \hat{\Phi}, \hat{\Phi} \coloneqq (0, 1, \dots, N-1)$

χ maps the lowest $\hat{g}$ to 0, the second lowest to 1 etc (4)

$\phi_{NDCG}(i, \hat{\imath}_{t_i})$, $\phi_{NDCG}: \hat{\Phi}\, X\, \Phi \to [0,1]$

Finally, the NDCG score is computed given the predicted indices $\hat{\imath}_{t_i}$ and the actual indices i (5)

Figure 5.1.: Network Ranking Explanation

$\hat{g}_{t_0}, \hat{g}_{t_1}, \dots, \hat{g}_{t_{N-1}}$ for the trajectories $\tau_{t_0}, \tau_{t_1}, \dots, \tau_{t_{N-1}}$ (2). $\hat{\imath}_{t_i} \in \hat{\Phi}$ represents $\hat{g}_{t_i}$ at timestep t_i in after ϕ as been applied. As defined in 4.1, each trajectory is a sequence of states s, the distance d, the total reward r and the reward for the power consumption r^p. For this experiment, only the states are retrieved. $\tau^{(s)}_{t_i}$ denotes a trajectory at timestep t_i that only contains the sequence of states: $\tau^{(s)}_{t_i} = (s_{t_i}, s_{t_i+1}, \dots, s_{t_i+\Lambda-1})$.
Next, a forward pass through the trained network for every $\tau^{(s)}_{t_i}$, starting from $i = 0$ to $i = N-1$, is performed. The intermediate result is an array of two dimensions because for every $\tau^{(s)}_{t_i}$, the network returns a prediction $\hat{r} \in \mathbb{R}$ for each of the 1000 states 5.2a.

$$[[\hat{r}_{t_0 0}, \dots, \hat{r}_{t_0 \Lambda-1}], \dots, [\hat{r}_{t_{N-1} \Lambda-1}, \dots, \hat{r}_{t_{N-1} \Lambda-1}]] = PrefNet([\tau^{(s)}_{t_0}, \dots, \tau^{(s)}_{t_{N-1}}]) \tag{5.2a}$$

For each t_i, those λ single predictions are summed up to obtain N predicted undiscounted cumulative returns $\hat{g}_{t_i}$ are raw logits $\hat{g}_{t_i} \in \mathbb{R}$. Undiscounted means the discount factor γ is set to zero.

$$\hat{g}_{t_i} = \sum_t [\hat{r}_{t_i 0}, \dots, \hat{r}_{t_i \Lambda-1}] \tag{5.3a}$$

Next a function $\phi : \mathbb{R} \to \{0, \ldots, N-1\}$ is introduced (3). It maps the unnormalized cumulative network outputs $\hat{g}_{t_i}$ to a discrete range. ϕ assigns the lowest $\hat{g}_{t_i}$ the label 0, the second-lowest the value 1, etc. Thus the highest cumulative prediction receives the label $N-1$.

$$\hat{\Phi} = [\phi(\hat{g}_{t_0}), ..., \phi(\hat{g}_{t_{N-1}})] \tag{5.4a}$$

Now, the list with the predicted ranking $\hat{\Phi}$ and the optimal ranking Φ^* each represent a tuple of length N with the unique elements $(0, 1, ..., N-1)$.
The NDCG (2.27a) metric $\phi_{NDCG} : \hat{\Phi} \times \Phi^* \to [0, 1]$ then computes a score with 1 being the best possible score (4). A score close to 1 means that both rankings are very similar. Thus, scores further away from one indicate less accurate ranking.
Since this work uses preference-based learning methods, a high score of ϕ_{NDCG} is desirable. This would validate that the network learns preferences.

The experiment was conducted for both the training set $\tau_{t_0}, \ldots, \tau_{t_{199}}$ and the extrapolation set $\tau_{t_{200}}, \ldots, \tau_{t_{299}}$.

For each of those two datasets, Pairnet, Tripletnet and NaiveTripletnet are compared.

As expected, the Pairnet and Tripletnet perform well on the training data 5.1 with both close to perfect ranking scores. On the other hand, NaiveTriplet also performs relatively well but has a much larger span. This is likely due to the fact that the NaiveTriplet approach does not check for monotony at step 3 in figure 4.2.
On the extrapolation data, seen at table 5.2, the Tripletnet is the only network able to significantly perform better than random. That makes this approach even more promising since it achieves a good score on the training data and still manages to rank unseen data better than the other approaches.

	Number of τ_{t_i}	**Best**	**Worst**	**Random**	**Pair**	**Triplet**	**Naive Triplet**
Mean	200	1.0	0.7210	0.8715	**0.9980**	0.9978	0.9442
Span	200	1.0	0.7210	0.8715	0.0022	**0.0007**	0.1151

Table 5.1.: NDCG scores on the training data over random 3 seeds

There are more ranking evaluations on different seeds in appendix A.3.1.

	Number of τ_{t_i}	Best	Worst	Random	Pair	Triplet	Naive Triplet
Mean	100	1.0	0.6862	0.8475	0.8329	**0.9128**	0.7933
Span	100	1.0	0.6862	0.8475	0.0422	**0.0169**	0.0422

Table 5.2.: NDCG scores on the test data over 3 seeds

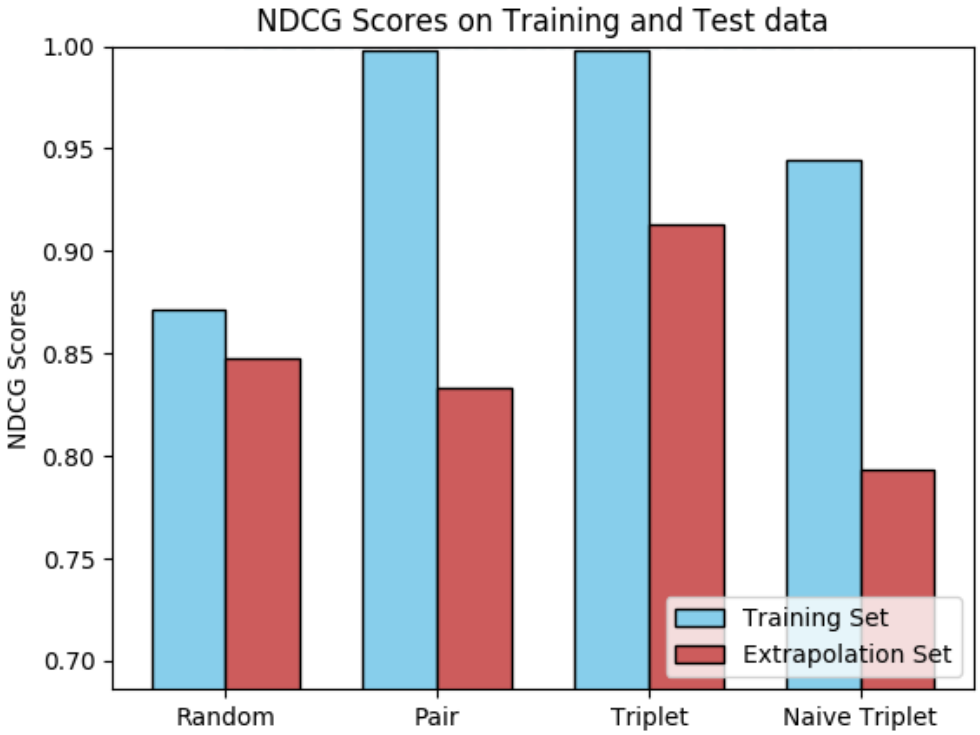


Figure 5.2.: Ranking results on training and test set over 3 random seeds

5.2. Evaluation of RL on learnt Reward Function

In previous chapters it was clarified that IRL does not include optimizing a policy. Instead, the result of running IRL is a synthetic reward function $\hat{R}$. In this experiment, the goal is to learn an optimal policy $\hat{\pi}$, trained on the learnt reward function. Two metrics for forward locomotion are considered: the total distance d the agent traveled and the energy efficiency r^p over Λ timesteps.
Three network training procedures are compared: Pair, Triplet and NaiveTriplet. The policy optimization is the same for all three approaches. Thus, any difference in performance stems from the process itself. Every variable that is created by using $\hat{R}$ as reward function carries the hat sign.

The following explains how the distance d is evaluated. The approach is analogously for the power efficiency r^p. Figure 5.3 shows the process visually
For each approach, five different policies $\hat{\pi}_0, \ldots, \hat{\pi}_4$ are inferred, each is the result of one million training timesteps (1). Each policy $\hat{\pi}_a$ is indexed by an agent $a \in \{0, \ldots, 4\}$ (2). It replaces the timestep schema because there are no checkpoints except for when the policy stops training. Upon finishing to optimize $\hat{\pi}_a$, one distance-only trajectory $\hat{\tau}_a^d$ of length Λ is created (3). Distance-only means it discards all other information.
Next, all the distances d in $\hat{\tau}_a^d$ are summed, resulting in the cumulative distance $\hat{g}_a^d$ (4).

Those cumulative distances $\hat{g}_a^d$ are compared to the cumulative distances $g_{t_i}^d$ of the trajectories $\tau_{t_i}^d, t_i \in T_{train}$ of the initial training set (5,6). $\hat{g}_a^d$ for each agent a is shown in 5.4. Table 5.3 does the same for the power efficiency. Higher values are better, both for the distances as well as the power efficiency.

Regarding the reward for the power efficiency, it turns out that the mean cumulative power efficiency $g_{t_i}^{r^p}$ of the demonstrations is relatively high and the maximum is almost the highest possible value which is 1000. This is due to the fact that in the early episodes the agent barely moved, if at all, and thus gets high values for the power efficiency generally. As the snake learns to move forward, its power efficiency goes down but is still relatively high.
The learnt networks achieve half of the maximum possible power efficiency with the exception of the Tripletnet which obtains around 700. It is interesting to see that NaiveTripletnet manages to obtain a power efficiency quite close to the Pairnet.

Concerning the cumulative distances $\hat{g}_a^d$, both, Pairnet and NaiveTripletnet outperform the demonstrations by a fair margin. On the other hand, the Tripletnet covers a lot less cumulative distance. It seems as if the Tripletnet values moving forward efficiently

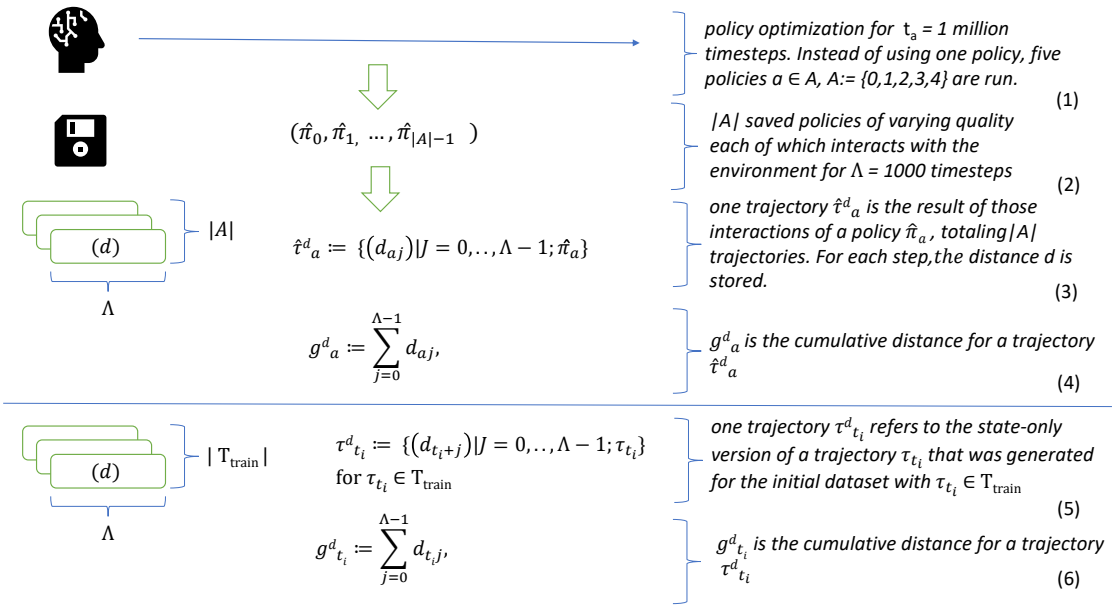


Figure 5.3.: Origin of $\hat{\tau}^d_a$ and $\tau^d_{t_i}$

Table 5.3.: Power Consumption for Pairnet, NaiveTripletnet and Tripletnet

	$\tau^{r_p}_{t_i}$	**Pairnet** $\hat{\tau}^{r_p}_a$	**NaiveTripletnet** $\hat{\tau}^{r_p}_a$	**Tripletnet** $\hat{\tau}^{r_p}_a$
$\hat{s}^{r_p}_0$	-	**559.37**	529.65	725.65
$\hat{s}^{r_p}_1$	-	546.30	527.77	701.15
$\hat{s}^{r_p}_2$	-	516.87	525.98	694.83
$\hat{s}^{r_p}_3$	-	523.69	525.81	**742.19**
$\hat{s}^{r_p}_4$	-	528.85	**531.20**	706.06
Mean	845.57	535.02	528.08	713.98
Max	999.97	559.37	531.20	742.19
Std	-	15.60	2.09	17.47

more than the covered distance.

Figure 3.1a shows my results plotted for the the different approaches regarding power efficiency and distance. The dotted lines represent the mean of the given demonstrations respectively.

An overview of policy parameters can be found in appendix A.1.2.

Table 5.4.: Distances for Pairnet, NaiveTripletnet and Tripletnet

	$\tau_{t_i}^d$	**Pairnet** $\hat{\tau}_a^d$	**NaiveTripletnet** $\hat{\tau}_a^d$	**Tripletnet** $\hat{\tau}_a^d$
$\hat{g}_0^d$	-	5.7042	7.7602	1.0024
$\hat{g}_1^d$	-	6.5585	7.7804	**1.4544**
$\hat{g}_2^d$	-	**7.4335**	**8.0863**	1.0224
$\hat{g}_3^d$	-	7.1280	7.9003	1.2065
$\hat{g}_4^d$	-	6.9760	7.9408	1.3862
Mean	4.49	6.7600	7.8936	1.2144
Max	6.30	7.4335	8.0863	1.4544
Std	-	0.5985	0.1183	0.1838

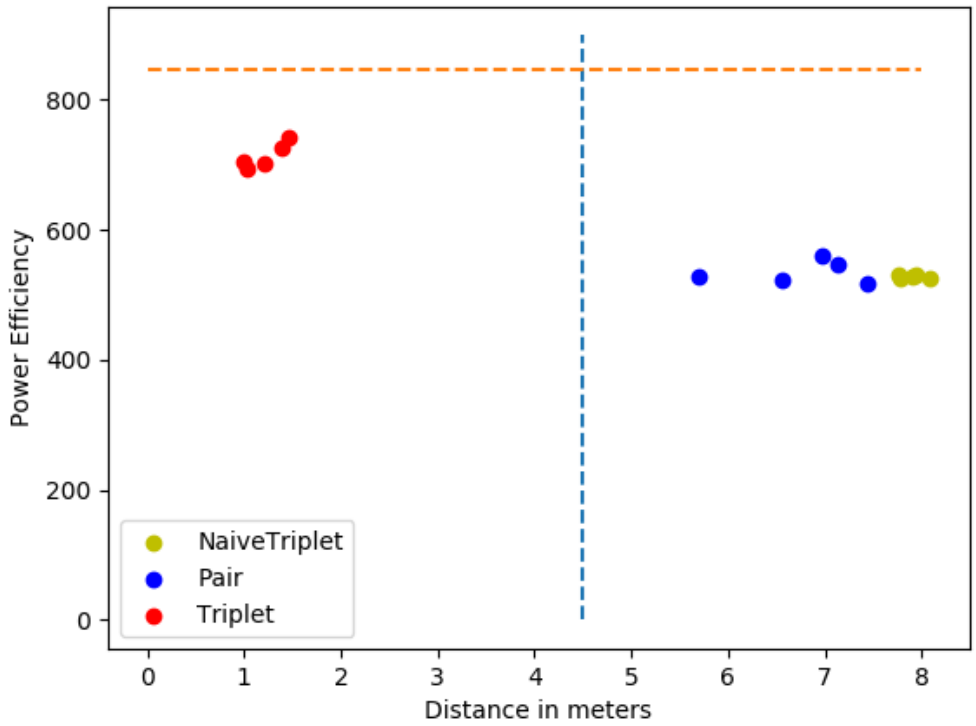


Figure 5.4.: Comparison of different approaches regarding distance and energy efficiency

5.3. Correlation

This experiment serves to find out the correlation between ground-truth reward and network predictions. As previously, multiple network training approaches are compared.
The Pearson Correlation Coefficient $\rho_{Pearson}$ and the Spearman Rank Correlation Coefficient $\rho_{Spearman}$ will be used as metric.
What can be seen for both coefficients is that Tripletnet is performing best.

	Number of τ_{t_i}	**Pairnet**	**Tripletnet**	**NaiveTripletnet**
Mean	200	0.8912	**0.9557**	0.7235
Span	200	0.0669	**0.0034**	0.5255

Table 5.5.: $\rho_{Pearson}$ scores for the network predictions and the ground truth reward

	Number of τ_{t_i}	**Pairnet**	**Tripletnet**	**NaiveTripletnet**
Mean	200	0.9718	**0.9790**	0.8061
Span	200	0.0213	**0.0007**	0.2814

Table 5.6.: $\rho_{Spearman}$ scores for the network predictions and the ground truth reward

Figure 5.5 contrasts the predictions and the ground-truth reward. One single is represents a trajectory. The vertical axes depict the cumulative ground-truth reward and the cumulative prediced reward. The horizontal axis tells the timestep of the policy by which the trajectory was generated.

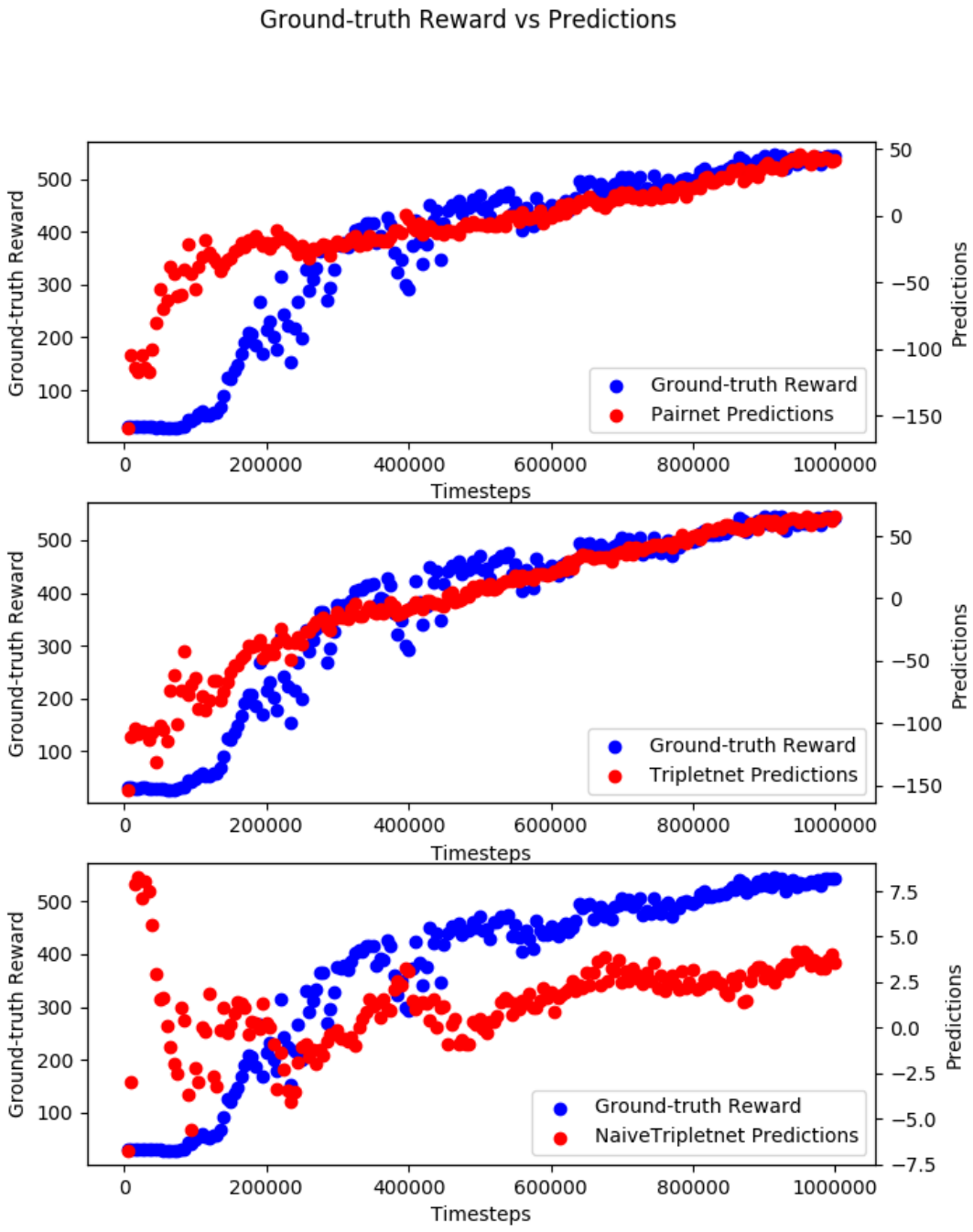


Figure 5.5.: Ground-truth Reward vs Predictions

6. Conclusion and Outlook

The question of what is the best way to learn is as old as mankind. Thus, it is even more exciting to see how different methods are used to make artificial systems more intelligent. In the last few years huge progress has been made in hardware and algorithmic terms, making it easier to teach machines. Especially non-stationary robots have to find their way in a high-dimensional space and have to learn to react to new experiences. This is accompanied by safety and efficiency considerations. In this thesis it was investigated how deep IRL can be combined with LTR to obtain good results for snake-like agents on forward locomotion.

A short overview of the content of the individual chapters is given. In the first chapter, the motivation of the thesis and especially, which technical achievements make the realization of this thesis possible, are explained. On the basis of those aforementioned changes, the research questions of whether and how the agent can learn, were introduced. Chapter two contains knowledge from the areas and techniques used in this thesis. In particular, the concept of RL and its relation to DL is explained. Among the policy-based methods, PPO is highlighted in particular, which is the policy optimization algorithm used in this thesis. Also, a comparison between RL and other techniques like IRL and IL is drawn. Additionally, LTR losses and metrics are discussed. In addition, information about the simulation environment Mujoco and especially, how actions, observations and reward are modeled, is provided. In the following chapter, the related work is laid out. This includes the initial experiments and results on the snake-like environment that Christian Lemke achieves as well as the T-REX approach and theory by Daniel Brown et al.. Chapter four presents the three steps this thesis takes to perform preference-based IRL. First, the trajectories and subtrajectories are created. Second, the data is preprocessed and preferences are learnt. And lastly, a policy can be optimized on top of that synthetic reward function.

Three experiments were performed in chapter five to evaluate the approach and to answer the research questions. The first experiment measures the ranking quality of the network. Since this work learns from preferences, it is important to find out whether the network manages to put all the predictions in order. The results confirm the assumption that a ranking is learnt because high NDCG scores are obtained. The

proposed Triplet approach even manages to perform well on unseen data.
The second experiment measures the performance of the policy optimization $\hat{\pi}$ on the learnt reward function $\hat{R}$. The evaluation metrics are the distance d and the power efficiency r^p. This way both components of efficient forward locomotion are incorporated. It turns out that both the Pair and the NaiveTriplet approach learn to move forward with a moderate power efficiency but actually learn to move forward further than the baseline. The Triplet approach on the other hand puts a much larger priority in the power efficiency, thus it does not move forward very far. Lastly, the correlation to the ground-truth reward is measured with the Pearson Correlation Coefficient and Spearman Rank Correlation Coefficient. As it turns out, there exists a strong correlation between the ground-truth and the predictions in each ranking approach. Again, the triplet approach looks a bit more promising than the other approaches.

Considering all the experiments, the research questions can be answered as follows. Regarding the first research question, all the presented approaches achieve to move forward with an at least moderate power efficiency. Second, concerning the impact of the ranking approach, the Triplet and Pair approach both achieve a high correlation to the ground-truth and learn to rank the training data well while the Triplet approach yields also stable result on unseen data. On the other hand, the RL evaluation on top of the learnt reward function shows very different behaviours. While the Pair and NaiveTriplet approach very fast, the Triplet approach puts more weight onto the power efficiency than the others.
Thus, the first research question can be answered positively, meaning the snake-like robot can be taught efficient forward locomotion with learning from demonstrations. Also, my second research question, the impact of the ranking approaches, gets sufficiently investigated. There is a lot potential in the Triplet approach and, since this work only uses a primitive version, there is still a lot of room for improvements. It is fair to say that more sophisticated ranking methods can definitely improve performance even further. I find that it is too hard for the current setup to optimize both goals, distance and power efficiency simultaneously, especially since those are conflicting. Thus, when one goal is identified by the agent, it merely tries to optimize this single metric.

Despite those achievements, there are some problems concerning the generalizability of this work. First, only the hyperparameters regarding data preparation and network training, such as for example trajectory length Λ and learning rate α, were studied. The configurations of the PPO algorithm were adopted from Christian Lemke. In general, there are a lot of hyperparameters, each of which can have a huge impact on how the performance is perceived and which configurations are being optimized for.
Additionally, the approach was only tested for the custom snake environment. It is

definitely needed to extend the research to more standard environments. One related problem here is the used reward function. First of all, the two metrics, velocity and power efficiency, are combined in a multiplicative way. This is problematic for two reasons: first, both are contrary goals, and second that those are combined by multiplication. It is far harder to create a synthetic reward function that weights both goals optimally, especially since those are contrary. Meaning that whatever increases the performance of one metric, automatically decreases the other one. Additionally, this reward function is more difficult to reconstruct than one where the features are combined linearly.
Also, the snake-like robot is trained in an high dimensional space with lots of room for error. For example, if the front part is in a horizontal position towards the direction it should move, then the rest of the snake is blocked. Daniel Brown et al. also reported they had more problems tackling higher dimensional tasks like Ant. Maybe partly because of that, it was necessary to train the snake on more subtrajectories than Brown et al. did.
Furthermore, while representing an issue and an opportunity, the used loss does not directly optimize the ranking. Also, there are a lot more sophisticated ranking approaches that should be tried. Lastly, having a differentiable ranking loss might be an interesting approach to optimize the ranking directly.

Altogether, this thesis covers a lot of interesting research areas like DL, LTR and RL. So any improvement in one of those fields open the door to mitigate flaws or increase the performance of this approach. Many of the possibilities for further improvement were already mentioned. Developing new or trying out existing ranking approaches is probably the most intuitive one. It was also mentioned that a lot of hyperparameter had to be fixed because of their sheer number. It might be especially promising to try out different network architectures for the synthetic reward function. In specific, a kind of recurrent approach can be quite suitable since T-REX works on timeseries.

List of Figures

List of Tables

Bibliography

[AM19] S. Armstrong and S. Mindermann. *Occam's razor is insufficient to infer the preferences of irrational agents*. 2019. arXiv: 1712.05812 [cs.AI].

[BGN19] D. S. Brown, W. Goo, and S. Niekum. "Better-than-Demonstrator Imitation Learning via Automatically-Ranked Demonstrations." In: *Proceedings of the 3rd Conference on Robot Learning*. 2019.

[Bin+20] Z. Bing, C. Lemke, L. Cheng, K. Huang, and A. Knoll. "Energy-efficient and damage-recovery slithering gait design for a snake-like robot based on reinforcement learning and inverse reinforcement learning." In: *Neural Networks* (2020). ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.05.029.

[Bro+19] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum. *Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations*. 2019. arXiv: 1904.06387 [cs.LG].

[CGD92] W. S. Cooper, F. C. Gey, and D. P. Dabney. "Probabilistic Retrieval Based on Staged Logistic Regression." In: *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '92. Copenhagen, Denmark: Association for Computing Machinery, 1992, pp. 198–210. ISBN: 0897915232. DOI: 10.1145/133160.133199.

[Che+10] J. Chen, W. Chu, Z. Kou, and Z. Zheng. *Learning to Blend by Relevance*. 2010. arXiv: 1001.4597 [cs.IR].

[DK08] K. Duh and K. Kirchhoff. "Learning to Rank with Partially-Labeled Data." In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '08. Singapore, Singapore: Association for Computing Machinery, 2008, pp. 251–258. ISBN: 9781605581644. DOI: 10.1145/1390334.1390379.

[Goo+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680.

[HE16] J. Ho and S. Ermon. *Generative Adversarial Imitation Learning*. 2016. arXiv: 1606.03476 [cs.LG].

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

[LB18] C. Lemke and Z. Bing. *Reinforcement Learning for Autonomous Locomotion Control of Snake-Like Robots*. https://github.com/christianlemke/masterthesis. 2018.

[LBW08] P. Li, C. J. Burges, and Q. Wu. *Learning to Rank Using Classification and Gradient Boosting*. Tech. rep. MSR-TR-2007-74. Advances in Neural Information Processing Systems 20. Jan. 2008.

[Liu09] T.-Y. Liu. "Learning to Rank for Information Retrieval." In: *Foundations and Trends® in Information Retrieval* 3.3 (2009), pp. 225–331. ISSN: 1554-0669. DOI: 10.1561/1500000016.

[Lv+11] Y. Lv, T. Moon, P. Kolari, Z. Zheng, X. Wang, and Y. Chang. "Learning to Model Relatedness for News Recommendation." In: *Proceedings of the 20th International Conference on World Wide Web*. WWW '11. Hyderabad, India: Association for Computing Machinery, 2011, pp. 57–66. ISBN: 9781450306324. DOI: 10.1145/1963405.1963417.

[Mit97] T. M. Mitchell. *Machine Learning, volume 1 of 1*. 1997.

[Mni+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[OAD18] I. F. D. Oliveira, N. Ailon, and O. Davidov. "A New and Flexible Approach to the Analysis of Paired Comparison Data." In: *Journal of Machine Learning Research* 19.60 (2018), pp. 1–29.

[Pom88] D. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network." In: *NIPS*. 1988.

[RA07] D. Ramachandran and E. Amir. "Bayesian Inverse Reinforcement Learning." In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2586–2591.

[Rus+15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[Sch+15] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. *Trust Region Policy Optimization*. 2015. arXiv: 1502.05477 [cs.LG].

[Sch+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[Sil15] D. Silver. *Policy-gradient Lecture*. 2015.

[SKP15] F. Schroff, D. Kalenichenko, and J. Philbin. "FaceNet: A unified embedding for face recognition and clustering." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015). DOI: 10.1109/cvpr.2015.7298682.

[Sta+19] A. Stanton, A. Ananthram, C. Su, and L. Hong. *Revenue, Relevance, Arbitrage and More: Joint Optimization Framework for Search Experiences in Two-Sided Marketplaces*. 2019. arXiv: 1905.06452 [cs.IR].

[TET12] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033.

[TWS18] F. Torabi, G. Warnell, and P. Stone. *Behavioral Cloning from Observation*. 2018. arXiv: 1805.01954 [cs.AI].

[Yam+05] H. Yamada, S. Chigisaki, M. Mori, K. Takita, K. Ogami, and S. Hirose. "Development of Amphibious Snake-like Robot ACM-R5, 36th, International symposium on robotics." In: *International symposium on robotics, INTERNATIONAL SYMPOSIUM ON ROBOTICS, 36th, International symposium on robotics*. Vol. 36. Tokyo: Japan Robot Association; 2005, p. 133. ISBN: 499027170X.

A. Appendix

A.1. PPO Hyperparameter

A.1.1. Trajectory Generation

Hyperparameter Name	Descriptions	Values
max_timesteps	time constraint	1.5 million
timesteps_per_actorbatch	timesteps per actor per update	2048
clip_param	clipping parameter epsilon	0.2
entcoeff	entropy coeff	0.0
optim_epochs	optimization hypers	10
optim_stepsize	optimization hypers	3e-4
optim_batchsize	optimization hypers	64
gamma	advantage estimation	0.99
lam	advantage estimation	0.95
schedule	annealing for stepsize parameters (epsilon and adam)	linear

Table A.1.: PPO hyperparameter configuration

A.1.2. RL on learnt Reward Function

Hyperparameter Name	Descriptions	Values
max_timesteps	time constraint	1 million
timesteps_per_actorbatch	timesteps per actor per update	2048
clip_param	clipping parameter epsilon	0.2
entcoeff	entropy coeff	0.0
optim_epochs	optimization hypers	10
optim_stepsize	optimization hypers	3e-4
optim_batchsize	optimization hypers	64
gamma	advantage estimation	0.99
lam	advantage estimation	0.95
schedule	annealing for stepsize parameters (epsilon and adam)	linear

Table A.2.: PPO hyperparameter configuration

A.2. Reward Function Hyperparameter

Hyperparameter Name	Descriptions	Values
split_ratio	the ratio train/validation set	0.9
epochs	number of epochs	15
lr	learning rate	0.00005
adam	optimizer	-

Table A.3.: Reward Function hyperparameter configuration

A.3. Additional Experiments

A.3.1. Network Ranking

Seed	Number of τ_{t_i}	Best	Worst	Random	Pair	Triplet	Naive Triplet
0	200	1.0	0.7210	0.8715	0.9966	**0.9979**	0.8732
1	200	1.0	0.7210	0.8715	**0.9988**	0.9981	**0.9883**
2	200	1.0	0.7210	0.8715	0.9986	0.9974	0.9712

Table A.4.: NDCG scores on the training data

Seed	Number of τ_{t_i}	Best	Worst	Random	Pair	Triplet	Naive Triplet
0	100	1.0	0.6862	0.8475	0.8257	**0.9209**	0.7695
1	100	1.0	0.6862	0.8475	0.8154	0.9135	0.7987
2	100	1.0	0.6862	0.8475	**0.8576**	0.9040	**0.8117**

Table A.5.: NDCG scores on the extrapolation data

A.3.2. Correlation

Seed	Number of τ_{t_i}	Pairnet	Tripletnet	NaiveTripletnet
0	200	0.8548	**0.9578**	0.3765
1	200	**0.9216**	0.9550	**0.9020**
2	200	0.8970	0.9544	0.0.8921

Table A.6.: $\rho_{Pearson}$ scores for the network predictions and the ground truth reward

Seed	Number of τ_{t_i}	Pairnet	Tripletnet	NaiveTripletnet
0	200	0.9578	**0.9789**	0.6206
1	200	**0.9791**	0.9794	**0.9020**
2	200	0.9784	0.9787	0.8956

Table A.7.: $\rho_{Spearman}$ scores for the network predictions and the ground truth reward