

# Proximal Policy Optimization with Explicit Intrinsic Motivation

**Proximal Policy Optimization mit expliziter intrinsischer Motivation**

Bachelor thesis by Fabian Scharf

Date of submission: February 20, 2020

1. Review: Prof. Dr. Johannes Fürnkranz
  2. Review: Prof. Jan Peters, PhD
  3. Review: MSc. Tobias Joppen
  4. Review: MSc. Svenja Stark
- Darmstadt



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fachbereich Informatik  
Knowledge Engineering Group

---

---

## **Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt**

---

Hiermit versichere ich, Fabian Scharf, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 20. Februar 2020

---

F. Scharf

---

---

## Abstract

---

As the field of reinforcement learning still suffers from a lack of sufficient exploration in sparse rewards environments and missing control over the trade-off between exploration and exploitation, we propose a new method based on intrinsic motivation and the Proximal Policy Optimization algorithm. With the intrinsic motivation serving as an exploration method, we train an agent to optimize its exploratory behavior as well as the behavior solving the predefined task of the environment. Instead of optimizing a single policy following a combination of the intrinsic motivation and the goal of the environment, we simultaneously train two separate policies, one for each of the mentioned tasks. For the interaction with the environment during the training, we use an entropy-based approach to determine the policy that chooses the next action for each step. Besides the possibility of controlling the level of exploration, the proposed method entails a low-noise policy and an extensive exploration of the state space.

---

---

## Zusammenfassung

---

Mit der Absicht, das noch immer präsente Problem einer umfangreichen Erkundung der Umgebung im Bereich des bestärkenden Lernens und die oftmals fehlende Möglichkeit, das Ausmaß der Erkundung festzulegen, zu relativieren, wird eine neue Lernmethode vorgestellt, die auf intrinsischer Motivation und dem Proximal Policy Optimization Algorithmus basiert. Mit Hilfe besagter intrinsischer Motivation, die als Grundlage für die Erkundung dient, wird ein Agent trainiert, indem sowohl sein Erkundungsverhalten als auch sein Verhalten zum Lösen des von der Umgebung festgelegten Ziels optimiert wird. Anstatt eine einzelne Strategie zu lernen, die eine Kombination aus intrinsischer Motivation und Aufgabe der Umgebung optimiert, werden gleichzeitig zwei unabhängige Strategien gelernt, um jeweils eines der beiden Ziele zu verfolgen. Bei der Interaktion zwischen Agent und Umgebung wird hierbei ein auf Entropie basierender Ansatz verwendet, um für jeden Schritt die Strategie zu wählen, die die nächste Aktion bestimmt. Neben der Möglichkeit, den Grad der Erkundung zu bestimmen, bringt die vorgestellte Methode eine Strategie mit wenig Rauschen und eine umfangreiche Erkundung des Zustandsraumes mit sich.

---



---

## Acknowledgments

---

I would like to thank my advisors Tobias Joppen and Svenja Stark for the numerous helpful conversations and for being available for questions and advice throughout all stages of the bachelor-thesis. Also, I want to thank Samuele Tosatto for the valuable discussions on off-policy reinforcement learning, for sharing the idea for the on-policy EIM approach, and his great helpfulness regarding all kind of questions.

---

---


# Contents

---

<b>1. Introduction</b>	<b>2</b>
<b>2. Related Work</b>	<b>5</b>
2.1. Intrinsic Motivation . . . . .	5
2.2. Off-Policy Policy Gradient . . . . .	5
2.3. Explicit Intrinsic Motivation . . . . .	6
<b>3. Foundations</b>	<b>7</b>
3.1. Machine Learning . . . . .	7
3.2. Unsupervised Learning . . . . .	9
3.3. Supervised Learning . . . . .	9
3.4. Deep Learning . . . . .	10
3.5. Reinforcement Learning . . . . .	17
3.6. Proximal Policy Optimization . . . . .	28
3.7. Intrinsic Motivation . . . . .	30
<b>4. Explicit Intrinsic Motivation</b>	<b>36</b>
4.1. When to explore . . . . .	38
4.2. Off-Policy Approach . . . . .	38
4.3. On-Policy Approach . . . . .	41
<b>5. Experiments</b>	<b>45</b>
5.1. General Settings . . . . .	45
5.2. Grid World Environment . . . . .	47
5.3. Intrinsic Rewards only . . . . .	49
5.4. Parameter Tuning . . . . .	49
5.5. Implicit vs Explicit Intrinsic Motivation . . . . .	51

---

---



---

<b>6. Results</b>	<b>52</b>
6.1. Parameter Tuning . . . . .	52
6.2. Exploration Behavior and Convergence of the IM Methods . . . . .	54
6.3. Implicit vs Explicit Intrinsic Motivation . . . . .	58
<b>7. Conclusion and Future Work</b>	<b>71</b>
7.1. Future Work . . . . .	73
<b>A. Parameter h</b>	<b>82</b>
<b>B. Extensive Exploration</b>	<b>84</b>

---

---

# Abbreviations

---

---

## List of Abbreviations

---

<b>Notation</b>	<b>Description</b>
ACER	Actor-Critic with Experience Replay
AI	Artificial Intelligence
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
EIM	Explicit Intrinsic Motivation
ICM	Intrinsic Curiosity Module
IIM	Implicit Intrinsic Motivation
IM	Intrinsic Motivation
KL	Kullback-Leibler

---



---

MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
Off-PAC	Off-Policy Actor-Critic
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
TD	Temporal Difference
TRPO	Trust Region Policy Optimization

---

# 1. Introduction

---

In recent years, the entire field of Machine Learning (ML) became more and more popular, conversations about and interest in the topic of Artificial Intelligence (AI) not staying exclusively in the area of research anymore. The idea of "intelligent" systems has reached publicity and is applied by many companies in some way. One big factor that helped ML to popularity is the revival of Neural Networks (NNs), especially in the form of Deep Neural Networks, and accompanying successes in a variety of different complex tasks, including the field of computer vision [1] and machine translation [2].

Another big field of ML that has been taken to a new level of success is the field of Reinforcement Learning (RL). RL fulfills multiple criteria intuitively connected to the idea of AI, such as learning by experience and the ability to make decisions. For this purpose, the algorithm (or *agent*) learns a function called *policy*, which determines an action the agent takes in a specific situation. By learning with the principle of trial and error, RL algorithms are able to solve a variety of different complex problems, internally evolving (near-)optimal sequences of decisions that require a high level of planning. The introduction of NNs into this area enabled the development of algorithms that are able to reach human-level performance in video and board games [3, 4, 5] and outreaching performance on robotic tasks [6, 7, 8, 9].

However, by no means all problems of RL have been solved yet. A successful application of an RL algorithm often highly depends on the design of the environment it interacts with, to some extent requiring prior engineering of the reward used by the agent to learn what to do. The performance of standard RL algorithms in environments with reward functions that are kept very simple, e.g., only providing a single reward when the task is fulfilled, is disillusioning.

There is, however, research dealing with this issue, part of it tackling the problem with the application of Intrinsic Motivation (IM). Using different methods of IM, the agent learns to explore the environment by learning about it, reflecting behavior observed in animals such as curiosity. This approach, for example, results in an exploratory behavior leading

---

the agent to areas of the environment it has rarely or never seen before by rewarding the agent for reaching respective states. These rewards are called intrinsic rewards, as opposed to the extrinsic rewards that are provided by the environment for solving the predefined task or a subtask. In order to explore the environment and simultaneously learn how to solve the true goal, the agent optimizes the extrinsic reward together with the intrinsic reward, whereby the intrinsic rewards should be small enough to ensure convergence to the task of the environment.

While making an important step in the direction of solving exploration issues, IM methods again come with new questions and problems that are to be solved. The main issues we want to address comprise

1. the inability of controlling the level of exploration throughout the learning process,
2. the possibility of premature convergence due to a low intrinsic-extrinsic rewards ratio, and
3. the possibility of a noisy final policy due to a high intrinsic-extrinsic rewards ratio and the possible non-convergence of the IM method.

The circumstance these problems could result from is the fact that, in most cases, the intrinsic reward is simply added to the extrinsic reward, only multiplied by a scaling factor. By jointly optimizing both goals, the one to explore and the one to fulfill the predefined task of the environment, the agent cannot distinguish them. In the case of being in need of the *greedy* behavior, i.e., only seeking the true goal of the environment, the agent has no option to immediately adjust its behavior, leading to the above-mentioned shortcoming of a noisy policy. Similarly, the exploratory behavior cannot be increased manually during the training, which could cause premature convergence.

Therefore, we aim to address this very cause by making the learning based on the intrinsic rewards explicit, that is, training a separate policy that only considers intrinsic rewards simultaneously to the training of the actual greedy policy, which only takes the extrinsic rewards into account. The agent makes decisions based on a probability distribution that is a combination of both policies, whereby the impact of each policy can be adjusted through a parameter. With this approach, we have better, state-dependent control of the level of exploration, which makes it possible to avoid a premature end of exploration. Moreover, we can turn the exploration off if a greedy behavior is desired and, in theory, the resulting greedy policy should not suffer from IM-induced noise as the intrinsic rewards should have no direct effect on it.

---

In this thesis, after presenting related work in Chapter 2 and providing the required basic knowledge in Chapter 3, the proposed approach is explained in more detail in Chapter 4. The mentioned issues of conventional intrinsically motivated RL approaches are examined in experiments that are explained in Chapter 5. In Chapter 6, we investigate the experimental results, especially with regard to the ability of the proposed approach to solve the said issues. These results are summarized and brought into a broader context in Chapter 7, in addition to a brief suggestion on when the use of the proposed method could be appropriate and when it might be disadvantageous.

---

## 2. Related Work

---

As the method we propose combines several aspects of RL, namely IM, off-policy policy gradients, and the distinct consideration of intrinsic and extrinsic rewards, we consider these aspects individually.

---

### 2.1. Intrinsic Motivation

---

The research on IM is wide-spread, including the development of a variety of different prediction error methods [10, 11, 12] as well as approaches using the idea of state novelty [13, 14, 15, 16]. While we tackle several issues of prediction error based IM, Savinov et al. [17] already showed that this type of IM can come with problems. More specifically, the problem of an intrinsically motivated agent tending to get stuck in situations from which it can induce stochasticity in the successor state has been addressed. This behavior has been observed on the example of the noisy-TV problem [12] describing the situation of giving an agent the opportunity to switch the program of an artificial television to a randomly chosen different program. The problem has been tackled by giving the agent a memory in order to make states it already knows and those that are close to such it has already experienced less attractive.

---

### 2.2. Off-Policy Policy Gradient

---

A commonly used method of training policy gradient methods in an off-policy manner is the off-policy policy gradient theorem [18]. It makes use of importance sampling in order to correct the gradient error that is caused by the sampling from a different policy than the one being optimized. Instead of using the product of importance weights of a trajectory, only a single importance weight is taken into account, which yields a lower variance but

---

introduces bias. This theorem has been well investigated by integrating it into several algorithms. Besides the initial Off-Policy Actor-Critic (Off-PAC) algorithm [18], other algorithms have been developed based on the theorem, such as Deep Deterministic Policy Gradient (DDPG) [19] and Actor-Critic with Experience Replay (ACER) [20]. However, both algorithms use the off-policy approach to reuse previously sampled transitions for updates in later iterations and, therefore, improve sample efficiency. In contrast, we use the approach to learn from transitions collected by a distinct policy. The problem of the off-policy policy gradient theorem of coming with bias due to sampling from the state visitation distribution induced by the behavior policy instead of the target policy has been addressed by adding a state distribution correction factor that is learned with an additional NN [21]. Instead of correcting the state distribution, we experiment with an on-policy approach that, similarly to the off-policy approach, makes use of a behavior policy in order to explore states according to the same.

---

### 2.3. Explicit Intrinsic Motivation

---

While the rewards produced by IMs usually only serve as a bonus reward in addition to the extrinsic rewards provided by the environment, Burda et al. [12] examined the behavior in the case of not having access to any extrinsic reward, learning a policy based on intrinsic rewards only. Successes could be achieved on a variety of different computer games, in which the exploration of new areas is advantageous. However, this approach disregards the potential of combining the information provided by intrinsic and extrinsic rewards. The combination of the ideas of intrinsically motivated and off-policy exploration was suggested by Szita et al. [22] and realized by Morere et al. [23] and Parisi et al. [24]. Hereby, the separation of intrinsic and extrinsic motivation was made on the value level, training two different value functions based on the respective type of rewards and combining them to build the behavior policy. Since their approach is value-based, i.e., the policy is made up of the learned value functions, the distinction between intrinsic and extrinsic motivation is implicitly made on the policy level, which we make explicit by learning approximators for the policies using policy gradients. Burda et al. [25] and Kim et al. [26] applied the approach of separate value functions to the field of policy gradients. However, there was still a single policy being learned to optimize a combination of intrinsic and extrinsic rewards.

---

## 3. Foundations

---

In this chapter, the basic knowledge necessary for the understanding of this thesis is provided. As it deals with IM in RL, we start with the most general super-category ML, followed by the ML fields of unsupervised and supervised learning. After an explanation of the functionality of NNs, RL and, in this context, IM is described in detail.

---

### 3.1. Machine Learning

---

In contrast to classical software engineering, in which domain-specific knowledge is required or has to be acquired at first to build an algorithm that meets the desired requirements, in the field of ML, we aim to eliminate this kind of dependency to a large extent. ML methods are instead designed to solve across-the-board problems, the algorithms themselves taking over the developer's task of acquiring problem-specific knowledge [27]. However, to train a *model* (the function approximating the true function, see Section 3.3) showing the desired behavior, the algorithm needs a set of training data (called the training set), which has to be provided (see Section 3.3) or gathered through experience (see Section 3.5) [28]. The algorithm is then being trained by optimizing a given performance measure representing the quality of the current performance of the algorithm with respect to the given data [27].

ML algorithms not only aim to learn the correct handling of data they have already seen in the training set but also to generalize in a way that they know how to treat similar data in the future. Considering the ability to learn only by example and to adapt to the circumstances, ML is especially beneficial assuming problems for which human expertise is missing or difficult to explain and ones that may change over time [28].

While there is a variety of subareas in the field of ML, this thesis focuses on the area of RL (see Section 3.5). In order to fully understand RL, a basic understanding of unsupervised learning and supervised learning is provided in Sections 3.2 and 3.3.

---

Before looking at the sub-categories of ML, general challenges that can occur when applying ML methods are discussed next.

### 3.1.1. Challenges in Machine Learning

While opening up new opportunities like the detection of patterns a human could not grasp in the amount of today's available data, it is not always easy to apply ML to any problem. There are some difficulties and requirements most ML approaches have in common, of which the most common ones are explained in this section.

**Data Availability:** The availability of sufficient data is one of the big issues when training an ML model is desired. In the case of supervised learning (see Section 3.3), this can refer to the amount of training data that is available to optimize the model. The absence of sufficient training data can lead to a bad generalization due to the lack of provided information. Regarding RL (see Section 3.5), where the data is not provided but collected by the algorithm itself, data availability can refer to the ease of collecting the data or the expensiveness of the same. For example, when training an algorithm to control a real robot, the communication between the device the algorithm runs on and the robotic components can be expensive in terms of communication time. Additionally, the use of the robot can be financially expensive, which can also restrict the availability of data.

**Expressive Power:** The expressive power of an ML model describes its function approximation ability [29]. An underlying principle of a problem can be too complex to be learned by a model. A linear function approximator, for example, will not be able to learn an accurate approximation of a function of higher complexity than linear, e.g., a quadratic one. Similarly, a linear classifier is not able to make non-linear classifications, i.e., separate data points that are not separable by a straight line.

**Overfitting:** An issue that might especially occur in the context of NNs is overfitting on the training data [30]. This expression refers to a poor generalization of the model on other input data than the training data that are of the same domain. In other words, the model learns the desired outcome of the inputs contained in the test set by heart instead of understanding the underlying problem. This problem can occur, for example, if there is not enough data available to provide sufficient knowledge of the connection between input features and output.

**Computational Resources:** Modern NNs can consist of a huge amount of parameters [31, 32]. Performing predictions and, more importantly, optimizing these parameters requires



---

tremendous computational resources. Depending on the complexity of the model, it might not be possible to train it on a conventional computer.

**Hyperparameter Tuning:** A hyperparameter is a parameter that is set before the training begins and kept constant instead of being adjusted throughout the training process. Hyperparameters can have a significant impact on the success and duration of the training. Finding appropriate hyperparameter settings gets a difficult challenge when increasing the amount of hyperparameters used (known as the curse of dimensionality [33]).

---

## 3.2. Unsupervised Learning

---

Unsupervised learning is a type of ML that, as the name indicates, does not need a supervisor providing the correct output to a given input. Instead of learning about the mapping of inputs to the output space by examples, unsupervised learning aims to find regularities in a dataset [28]. Thus, the algorithm can provide information about new data by comparing its attributes to the ones of the already available data.

A popular example of this kind of ML algorithms is *clustering*. In this case, the algorithm aims to group data by means of their attributes, i.e., data with similar attribute values are grouped. This method enables, for example, a company to group its customers by their attributes (e.g., age) to provide fitted offers to specific types of customers [28]. Unsupervised learning is being applied to a wide range of areas, including astronomy, social sciences, and medical sciences [34].

---

## 3.3. Supervised Learning

---

The problems tackled by supervised learning are characterized by the presence of a labeled training set. That is, in addition to samples of input features  $\mathbf{x} \in \mathcal{X}$  with  $\mathcal{X}$  being the set of all possible input feature vectors, the set also contains the associated desired outputs (or *targets*)  $\mathbf{y} \in \mathcal{Y}$  with  $\mathcal{Y}$  denoting the set of all possible outputs [35, 27]. The goal of supervised learning is to learn an approximation of the true function  $f: \mathcal{X} \rightarrow \mathcal{Y}$  mapping each input  $\mathbf{x}$  to the correct output  $\mathbf{y}$ . The function  $\hat{f}$  representing the approximating function learned by the ML approach is called *model*, and the output  $\hat{f}(\mathbf{x})$  produced by a model for a given  $\mathbf{x}$  is referred to as the *prediction*.

---

When using supervised learning, there are two kinds of problems that are mainly tried to be solved: classification and regression. The former represents the problem of ordering samples into a class based on their features, similarly classifying samples with similar features [36]. In contrast to mapping the input to a discrete space of output classes, the field of regression deals with the prediction of continuous values given input features.

---

## 3.4. Deep Learning

---

A popular tool that allows for addressing both classification and regression problems is the *Neural Network (NN)*, which is also known as *Connectionist Network*. While NNs can also be used to address unsupervised learning tasks [37], for our purpose of eventually using them in the context of RL, it is useful to regard their use as a supervised learning tool since the underlying methods can then be transferred to the use case of RL relatively easily (see Section 3.5.4). NNs consist of a variable amount of *hidden layers*, each of which made up of an arbitrary amount of *neurons*. The hidden layers are arranged hierarchically between two additional layers: the *input layer* and the *output layer* [38]. As the names suppose, the input layer is represented by the input  $x$ , and the output layer either directly contains the prediction  $\hat{f}(x)$  or a representation that is being converted to the final output using a fixed function, i.e., one that is not modified throughout the learning process. In the case of a classification problem, for example, the output of an NN does usually not consist of a single value representing the predicted class, but of a value for each class of the classification problem. The prediction is then determined using the argmax operator, choosing the class holding the highest value. For a better understanding, the general architecture and the functionality of the individual components of an NN are explained in this section.

### 3.4.1. Neuron

Neurons (also referred to as *units*) are the key component of NNs and, therefore, build the foundation for understanding the networks' functionality. Every neuron has  $N$  inputs  $x_i$  with  $i \in \{1, 2, \dots, N\}$  and an output  $y$ . A neuron holds a *weight*  $w_i$  for each input. The weights, together with the input signals, are united as a linear combination, which might be optionally added with a bias  $b$  [38].

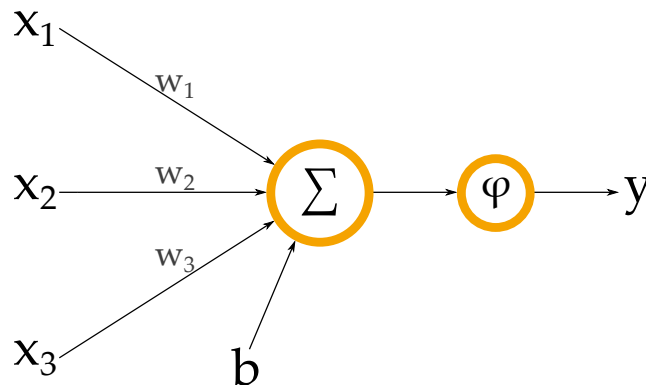


Figure 3.1.: The architecture of a neuron with three input signals  $x_i$  and corresponding weights  $w_i$ , activation function  $\varphi$ , and output  $y$ . Here, the bias  $b$  is represented as an additional input, which is added to the other weighted input signals.

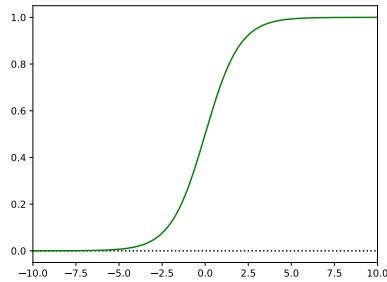
The result is mapped to the output space by applying a usually nonlinear activation function  $\varphi$ , leading to the equation

$$y = \varphi\left(b + \sum_{i=1}^N w_i x_i\right).$$

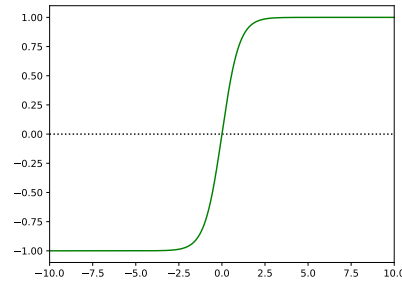
The output signal is either one of the final outputs of the network if the neuron is part of the output layer, or is used as the input signal of one or more neurons of the successor layer. A visual depiction of an artificial neuron can be seen in Figure 3.1.

Although the choice of the activation function is arbitrary to some extent, yet it can have a big impact on the result of the NN and its ability to learn a certain task. E.g., the activation function has to be nonlinear to allow the network for learning nonlinear functions [39]. However, there is no consent on which activation function should be used by neurons inside the hidden layers. A selection of commonly used activation functions can be seen in Fig. 3.2, namely

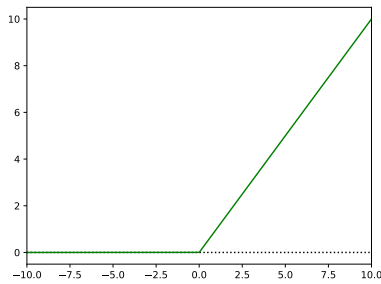
- Sigmoid:  $f(x) = \frac{1}{1+e^{-x}}$ ,
- Tangens hyperbolicus (Tanh):  $f(x) = \tanh x$ ,
- Rectified Linear Unit (ReLU):  $f(x) = \max(0, x)$ , and



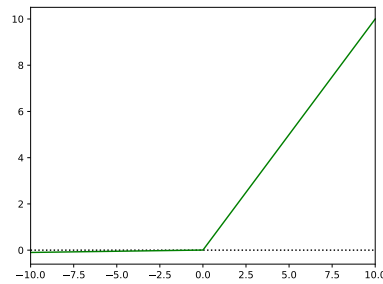
(a) Sigmoid



(b) Tangens hyperbolicus (Tanh)



(c) Rectified Linear Unit (ReLU)



(d) Leaky ReLU

Figure 3.2.: The graphs of commonly used activation functions, where the black dotted line marks the constant  $y = 0$  for a clearer presentation.

- Leaky ReLU:  $f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$

In the output layer, on the other hand, it is important to consider the task that is to be solved. As an example, assume a classification task with  $n \in \mathbb{N}^+$  possible classes. The output is usually represented by  $n$  neurons, each outputting a value for the associated class. In order to obtain a probability-like distribution of the classes, the softmax function can be used [38]. Said function assigns each output a value between 0 and 1 and is denoted as

$$\sigma(y_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

for each output  $y_i$  with  $i = \{1, 2, \dots, n\}$ . In contrast, networks designed for regression tasks might not make use of an activation function in the output layer at all.

---

### 3.4.2. Layer

The architecture of NNs is structured by the use of *layers*, each consisting of  $M \in \mathbb{N}^+$  neurons. The layers are hierarchically ordered, whereby the outputs produced by neurons of one layer are fed as inputs to the neurons of the next layer. In the case of a *fully connected* layer, the input of each neuron consists of the outputs of all neurons of the previous layer [38]. Since neurons of the same layer do not depend on the outputs of each other, the computation inside one layer can be expressed by a matrix multiplication and can be performed in parallel. With  $\mathbf{W}$  being a matrix built up by the weight vectors  $\mathbf{w}^T$  of each neuron of the layer as its rows,  $\mathbf{b}$  denoting the vectors of the biases of the neurons, and  $\mathbf{x}$  being the vector of all input signals, the computation of the output vector  $\mathbf{y}$  of a layer can be denoted as

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \text{ with } \mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & \cdots & \cdots & w_{MN} \end{pmatrix}.$$

For the first layer (input layer), there is no computation taking place since it is not an actual layer of neurons. In fact, the input layer is represented only by the features being fed into the NN. The last layer (output layer) provides the final results of the network as its output. All layers between the input and output layers are called hidden layers. Networks with one or more hidden layers are also called *Multilayer Perceptrons* with perceptron referring to a special type of neuron, which is activated according to a threshold, only being able to output 0 or 1 [40]. When viewing the number of hidden layers as the *depth* of the network, NNs with multiple hidden layers are often referred to as *deep neural networks*, which explains why the use of NNs as an ML method is called *deep learning*. Figure 3.3 depicts a feedforward NN with fully connected layers.

It was shown that every continuous function can be approximated by an NN with only one hidden layer and a sigmoidal activation function [41]. Later, NNs were proven to be universal function approximators using other activation functions as well [42]. The few restrictions that were pointed out are the need for a sufficient amount of neurons inside the hidden layer and the activation function to be continuous, bounded, and nonconstant. Nonetheless, deep neural networks with more than only one hidden layer became famous as they were shown to be advantageous [43] and thanks to empirical results suggesting that deep architectures might be beneficial learning complex tasks [1, 3].

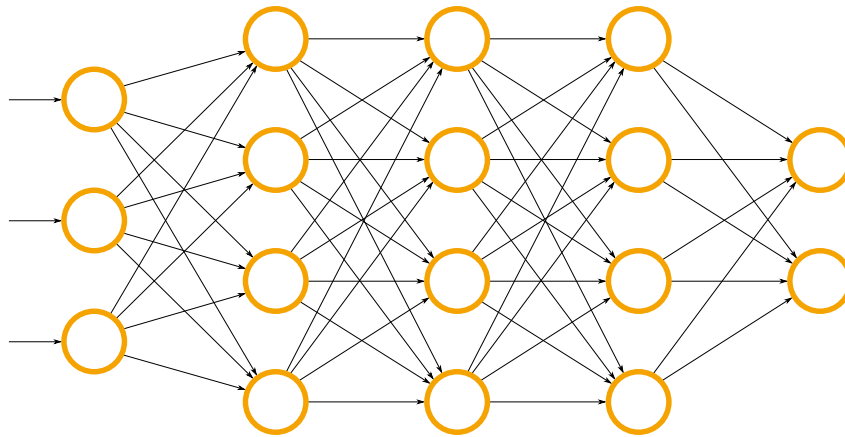


Figure 3.3.: A feedforward NN with an input layer of three neurons (left), an output layer of size two (right), and three hidden layers of size four.

### 3.4.3. Types of NNs

NNs with the so far explained architecture are called feedforward neural networks, referring to the direction the input data passes through the network. In a feedforward network, the input signal passes all layers in a straight row, whereas each layer is usually fully connected to its successor layer, i.e., each neuron of a layer is connected to each neuron of the next layer. However, there are two more commonly used architectures that are appropriate under certain circumstances, briefly explained next.

**Recurrent Neural Networks** are NNs that have at least one recurrent layer, which does not only feed its output to the next layer, but also to itself, combined with the next input that comes from the previous layer. In this way, the network possesses a kind of memory, incorporating previous samples into the prediction [38]. This approach is especially appropriate in the case of the data being of a sequential nature. For example, assuming a handwriting recognition task, uncertainty in the classification of a single character might be waived by incorporating information of previous characters since certain characters are more likely to follow certain sequences of other characters. Since the use of recurrent networks, the idea developed and entailed architectures that are more efficient at storing long-term information, such as the *Long Short-Term Memory* [44].

**Convolutional Neural Networks** consist of one or more convolutional layers. Instead of being fully connected to the previous layer and learning a weight for each connection,

---

a *kernel* is learned. The kernel consists of a fixed set of  $n$  parameters connecting each combination of  $n$  adjacent predecessor outputs to a single input [45, 46]. Hence, the same parameters are applied to all neighboring outputs of the previous layer, enabling the extraction of a specific feature characterizing the relationship between the respective values. Analogously, the same method can be applied to two-dimensional data like images. In this context, a convolutional layer can be regarded as a filter, for example, recognizing edges. Therefore, especially in computer vision tasks, convolutional neural networks helped to achieve great successes [1].

### 3.4.4. Training

While the previous paragraphs dealt with the architecture of a neural network, the most important part, which makes the approach belong to the area of ML, is its ability to change some of its components, the weights and biases, over time in order to learn problem solutions. To optimize an NN, we need a performance measure or *loss*  $l_{\mathbf{W}}$  indicating how good a prediction  $\hat{y}$  by a network holding the parameters  $\mathbf{W}$  was, with respect to the true target  $y$ . For example, we can use the squared error loss

$$l_{\mathbf{W}}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

to determine the error of a single prediction [47].

In order to measure the performance of the model regarding the entire dataset  $D$ , we take the sum of the losses of all data points in  $D$  with respect to the model  $\hat{f}$  and the true function  $f$

$$L_{\mathbf{W}}(\hat{Y}, Y) = \sum_{d \in D} l_{\mathbf{W}}(\hat{f}(x), f(x)),$$

with  $x$  being the input of data point  $d$ ,  $\hat{Y}$  denoting the predictions, and  $Y$  the targets of all data points in  $D$  [38]. The loss can optionally be divided by the number of data points in order to build the mean loss. Our optimization aim is to minimize  $L$  in order to train a model that produces predictions with a small error on the training data.

The method used to do so is called *gradient descent*. To apply this method, the first derivative  $\nabla_{\mathbf{W}} L_{\mathbf{W}}$  (the gradient) of the loss  $L$  is formed with respect to the weights of the network. Since the gradient points into the direction that maximizes the loss, we want to

---

follow the opposite direction, which is why the method is called gradient *descent*. Hence, a gradient descent step can be denoted as

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \alpha \nabla_{\mathbf{w}_t} L_{\mathbf{w}_t}(\hat{Y}, Y),$$

where  $\alpha \in (0, 1]$  denotes the *learning rate*, and  $t$  is the number of update steps performed, i.e.,  $\mathbf{W}_t$  is the old set of weights and  $\mathbf{W}_{t+1}$  the new one after the update step [38]. The learning rate is an important parameter, which can have a great impact on the learning process. While  $\alpha$  prevents the update step from becoming too big, which could cause an optimum being missed by skipping over it, a very small learning rate can lead to slow learning. However, the learning rate does not have to be constant. In fact, in practical application, it is common to use an optimizer (such as Adam [48]) that automatically adapts the learning rate over time by taking the gradients of previous steps into account.

In practice, instead of calculating the gradient based on the predictions of the whole dataset, the training data is usually separated into subsets of equal size, called *mini-batches*, whereas the whole dataset is referred to as the *batch*. Hereby, an estimation of the true loss  $L_{\mathbf{W}}(\hat{Y}, Y)$  is being made. Batch optimization enables the parameters to be updated more frequently since fewer data has to be processed for each update step. Despite increasing the variance in the loss, this method can lead to faster convergence. The network is often trained using each data point of the dataset multiple times. One training iteration over the entire dataset is called an *epoch*.

When trying to optimize NNs with multiple hidden layers, a method called *backpropagation* is usually being used, which makes use of the chain rule of differentiation [38, 37, 47]. After a forward pass, which produces the predictions, with the help of which the loss is calculated, a backward pass or backward propagation is performed, updating the weights of each layer one after the other, starting by the last one, which gives the algorithm its name.

### 3.4.5. Hyperparameters

Using NNs, we have to deal with the tuning of several hyperparameters. Besides the (initial) learning rate, we have to decide on a fixed number of layers and the number of neurons that should be placed in each layer. While it seems that deeper networks, i.e., networks with multiple hidden layers, perform better regarding complex tasks, there is no consensus on a certain number that should be used. Another hyperparameter that has to be decided upon before beginning the training is the mini-batch size, i.e., the number



---

of data points processed before each update, reaching from a size of 1 (called stochastic learning or stochastic gradient descent) to  $|D|$  (called batch learning) [49].

---

## 3.5. Reinforcement Learning

---

Reinforcement Learning (RL) is a separate field of ML and lies somewhere between unsupervised and supervised learning. Instead of relying on a given labeled training set, RL algorithms collect data on their own.

In order to do so, one or more so-called *agents* interact with an *environment*. The environment can be regarded as the world an agent is located in. It follows predefined rules and reacts to the behavior of the agent. An RL agent performs actions within the environment and is being rewarded for choosing actions leading to a learning goal and, thus, is reinforced to choose these actions again. A new agent usually starts by trying random actions until the collected rewards indicate other actions to be beneficial. Hence, the agent learns a desired behavior by trial and error, without any prior knowledge of the environment. The RL approach is especially appropriate for solving robotic and gaming tasks as can be seen by means of the numerous successes in these areas in the last years [3, 4, 6, 7].

### 3.5.1. Markov Decision Process

Many applications of RL regard the case of Markov Decision Processes (MDPs). Here, the agent can interact with a given environment  $\mathcal{E}$  by performing actions  $a$ . The environment provides a finite or infinite amount of states  $s$  in which the agent can be located. A state space that can be represented by a finite amount of numbers is called discrete. Otherwise, the state space is continuous. The time in which the agent interacts with the environment is divided into timesteps  $t$ . Each timestep, the agent chooses an action  $a_t$  to perform and passes its decision to the environment. This decision of which action to take is made by an ML model held by the agent, which is called the *policy* and is denoted  $\pi(s)$  in the case of a deterministic policy always choosing a specific action given a state  $s$  or  $\pi(a | s) = Pr(a_t = a | s_t = s)$  for a stochastic policy. The environment responds with the new state  $s_{t+1}$  the agent has moved to, resulting from action  $a_t$  performed at state  $s_t$  and depending on a transition probability distribution in the case of a stochastic environment or a transition function in the deterministic case. In point of fact, the agent often does not see the "true" state but receives an *observation* representing the features of the respective

---

state. As opposed to fully observable MDPs, partially observable MDPs (POMDPs) do not encode all features necessary to understand the entire state in an observation [50, 51]. Since we only consider fully observable MDPs, we use the term *state* as a synonym for *observation*.

In addition to the new state, a reward  $r_t$  is returned by the environment, indicating the short-term quality of choosing  $a_t$  at  $s_t$ . The compound sample of state, action, and new state  $(s_t, a_t, s_{t+1})$  is also referred to as a *transition* and can additionally contain the reward  $r_t$ . Sometimes, the notation  $(s, a, s')$  is used instead. The agent may interact with the environment as long as the *episode* does not end. Hence, *episode* describes the time period the agent can sample transitions before the environment is reset. There are various reasons why an episode ends, e.g., the exceeding of a timestep limit or the agent entering a terminal state such as the goal field in a game. A sequence of transitions is also referred to as a *trajectory*; the process of sampling the trajectory is called *rollout*. However, the overall goal of RL is not to maximize the short-term reward for each timestep, but to maximize the long-term reward. Therefore, the discounted future return is usually being optimized, with *return* referring to the cumulative rewards of an entire episode or of a subset of transitions of an episode, starting from a specific timestep. The discounted future return is defined by

$$G_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'},$$

with episode length  $T$  and a discount factor  $\gamma \in [0, 1]$  starting from timestep  $t$ . In some cases,  $T$  can also be the *horizon*, which describes the number of future steps that are considered in the calculation.

The discount factor determines how much of an impact the rewards that are collected in successor timesteps of  $t$  should have on the decision making of the agent. That is, with the discount factor we decide whether to maximize the short-term (low  $\gamma$ ) or the long-term (high  $\gamma$ ) quality of the agent's decision. Usually, this parameter is set to a value near to 1 [19, 52].

In general, an MDP can be described by a tuple  $(S, A, P, R, d_0, \gamma)$ . Hereby,  $S$  is the set of all possible states and  $A$  the set of all possible actions. Moreover,  $P(s' | s, a)$  denotes the transition probability distribution or transition function, and  $R(s, a)$  is the reward function. The start state of an episode is sampled from the initial state distribution  $d_0$  and  $\gamma$  is the discount factor mentioned before [53].

The general process of RL considering an MDP is visualized in Figure 3.4.

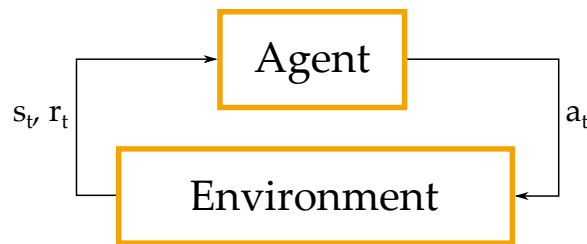


Figure 3.4.: General flow of RL considering an MDP. The agent performs actions inside an environment. The decision on which action to perform is made based on the observation or state, which is provided by the environment together with a reward, based on which the decision making is updated.

### 3.5.2. Value Function, Policy Gradient, and Actor-Critic

There are many ways to approach RL problems, which becomes apparent when considering the various algorithms that are successfully used inside the RL domain, despite many of them following different underlying techniques. The RL algorithm classes commonly used are value-based algorithms such as Deep Q-Network (DQN) [3], policy gradient methods like REINFORCE [54], and actor-critic algorithms like Proximal Policy Optimization (PPO) [52]. The latter is a combination of both, value-based and policy gradient methods, referring to the policy part as actor and to the value part as critic. The functionality and properties of these three methods are clarified in the following.

#### Value-based

In RL, we can express the value of the agent, which uses a policy  $\pi$ , for being in state  $s$  with the help of the (state) value function  $V^\pi(s)$ . It is defined by the expected discounted future return from state  $s$  when following  $\pi$ . Additionally, there is another value function: the state-action value or Q-function  $Q^\pi(s, a)$ , which is the expected discounted future return when performing action  $a$  from state  $s$  and following  $\pi$  afterward.

The objective function we want to maximize in the domain of RL is the expected discounted

---

future return when following a policy  $\pi$ , which can be expressed by

$$\begin{aligned}
J(\pi) &= \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) \\
&= \mathbb{E}_{s \sim d^\pi(\cdot)} [V^\pi(s)] \\
&= \mathbb{E}_{s \sim d^\pi(\cdot)} \left[ \sum_{a \in \mathcal{A}} \pi(a | s) Q^\pi(s, a) \right] \\
&= \mathbb{E}_{s \sim d^\pi(\cdot), a \sim \pi(\cdot | s)} [Q^\pi(s, a)].
\end{aligned} \tag{3.1}$$

Hereby,

$$d^\pi(s) = \sum_{s_0 \in \mathcal{S}} \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s | s_0, \pi) d_0(s_0)$$

is the state visitation distribution under  $\pi$ , where  $Pr(s_t = s | s_0, \pi)$  denotes the probability that the agent is in state  $s$  at timestep  $t$  when starting from the initial state  $s_0$  and executing  $\pi$  [55].

Value-based RL methods usually try to learn an approximation of the state-action value function in order to choose the action that maximizes the discounted future return. The policy resulting from such a method is deterministic, built up, for instance, by always choosing the action with the highest approximated state-action value for each state [53, 56]. For a better understanding of how to learn the state-action value function, we will have a look at the Q-Learning algorithm [56].

Q-Learning is a Temporal Difference (TD) method. In contrast to Monte Carlo (MC) methods, which use the actual return of entire rollouts of episodes to update their estimates, TD methods only need the immediate reward  $r_t$  for an action  $a_t$  performed in a state  $s_t$  to update the approximate value  $\hat{V}^\pi(s_t)$  for this state. Hence, TD methods can perform updates every timestep and do not have to wait for the episode to end. They do so by learning from the approximate values they have learned so far instead of relying on the actual returns, which gets clearer when looking at the update step

$$\hat{V}^\pi(s_t) \leftarrow \hat{V}^\pi(s_t) + \alpha(r_t + \gamma \hat{V}^\pi(s_{t+1}) - \hat{V}^\pi(s_t))$$

of a basic TD approach, with a positive learning rate  $\alpha$  [53, 57]. The learning rate determines the extent of the update step, causing a small update in case of a small  $\alpha$  and the full adoption of the approximation based on the value of the next state if  $\alpha$  is 1. As we can see, TD methods use their own predictions as a part of their update target

---

---

$(r_t + \gamma \hat{\mathbf{V}}^\pi(\mathbf{s}_{t+1}))$ ). Similarly, Q-Learning updates a Q-value using the highest approximated Q-value of the next state of the sampled transition  $(s_t, a_t, s_{t+1})$ , resulting in the update

$$\hat{Q}^\pi(s_t, a_t) \leftarrow \hat{Q}^\pi(s_t, a_t) + \alpha(r_t + \gamma \max_a \hat{Q}^\pi(s_{t+1}, a) - \hat{Q}^\pi(s_t, a_t)),$$

with which an approximation  $\hat{Q}$  of the true Q-function is learned. While the fact that the estimates of the value function are based on other estimates (bootstrapping) can lead to a relatively fast convergence, it also comes with bias [53]. It has been shown that some value-based methods are guaranteed to converge under certain circumstances, which include the requirements of the function approximator being linear and discrete spaces [56, 54]. However, there is no general guarantee of value-based methods converging to an optimum. Under certain circumstances, some algorithms were even shown to diverge [58]. Moreover, while it is possible to handle continuous state spaces using value approximations only [3], the action space has to be discrete. This shortcoming comes with the methodology of using the max operator over all state-action values in order to build the policy, which is computationally expensive for very large action spaces and not possible without further effort for continuous ones. In the case of a continuous action space, a prior discretization is necessary, i.e., the continuous actions have to be represented in a discrete way. Finding an appropriate discretization approach can take a long time and might lead to an insufficient representation of the action space. Furthermore, value-based methods usually require an additional exploration strategy, leading to the issue of finding an appropriate trade-off between exploration and exploitation (when to explore new states and when to follow the currently learned *greedy* policy), which is known as the *exploration-exploitation dilemma*. Nonetheless, value-based RL algorithms achieved big successes in recent years, especially with the introduction of NNs. An example of successful value approximation algorithms is the DQN, which managed to play 49 different *Atari 2600* games on a level comparable to the level of a professional game tester [3]. This algorithm works very similarly to classical Q-Learning, with the difference of using an NN as its Q-value function approximator and learning from transition samples that were gathered in the past in order to reuse them multiple times.

## Policy Gradient

A different but also very popular method of approaching the domain of RL is the use of policy gradients. In contrast to value-based methods, this approach aims to directly approximate the policy itself instead of deriving it from a value function. To do so, a parameterized policy  $\pi_\theta(a | s)$  is used with  $\theta$  being the set of parameters. The parameters

---

are adjusted throughout the learning process to maximize a performance measure or objective function  $J(\theta)$ . This performance measure often is the expected discounted return, as described by Equation (3.1), now with the parameterized policy  $\pi_\theta$ . The policy gradient theorem, which was introduced in its early form as the algorithm REINFORCE [54], provides an approximation of the gradient of the objective function with respect to  $\theta$  [59], which is

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d^{\pi_\theta(\cdot)}, a \sim \pi_\theta(\cdot|s)} [\nabla_\theta \log \pi_\theta(a | s) (Q^{\pi_\theta}(s, a) - b(s))],$$

with  $b(s)$  being an arbitrary function called reinforcement baseline [54] with the only restriction of being unbiased, i.e., not to use bootstrapping [53]. The policy is updated by performing gradient ascend steps with respect to the objective function. Hence, an update step has the shape

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta)|_{\theta=\theta_t},$$

with a learning rate  $\alpha$  [59]. In contrast to value-based methods, policy gradient algorithms are unbiased by nature because they fully rely on MC rollouts for their performance measure. However, while this kind of algorithms does not have the problem of bias, they suffer from high variance, which is also a consequence of the MC rollouts. Because of the policy being of stochastic nature, it is unlikely that the exact same trajectory is produced in each episode; on the contrary, the trajectories often look very different, which makes it difficult to estimate the true performance of the current policy. The effect of high variance compared to high bias in estimations is depicted in Figure 3.5. The baseline  $b(s)$  addresses this issue of high variance and, in fact, can reduce it effectively [53]. Besides using a baseline, it is also possible to reduce the variance by sampling more than one episode before performing a policy update. However, policy gradient methods already are prone to a high level of sample inefficiency (i.e., a high number of transitions have to be sampled before achieving successes) because of the reliance on entire rollouts before being able to perform an update step, which might get worse when sampling multiple episodes before updating the policy. Especially under circumstances where collecting samples is expensive, such as the work with real robots with a long communication time between the robot and the computer, sample inefficiency might be very undesirable.

Although the mentioned shortcomings of policy gradient methods could be enough reasons to not use them in some cases, there are various advantages over value-based methods that make the use of this kind of algorithm desirable under a large variety of circumstances.

While value-based methods usually require an additional method to provide an exploratory behavior (e.g.,  $\epsilon$ -greedy [3]), policy gradient methods come with a natural exploration because of the usually stochastic policy. Furthermore, policy gradient methods open the opportunity to not only work with continuous state spaces but also to handle continuous

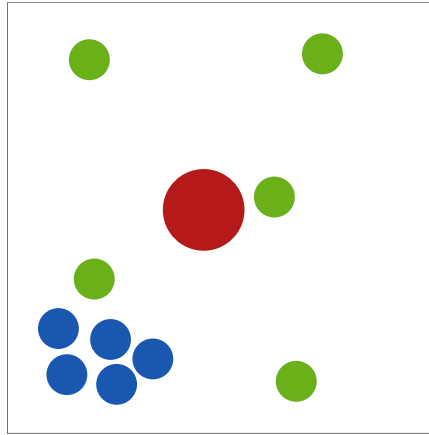


Figure 3.5.: Example estimates using a highly biased estimator with low variance and one with high variance and low bias, where the red circle is the true value that is to be estimated, the blue circles mark the estimates of the biased estimator, and the green circles mark the estimates of the high-variance estimator. The biased estimator makes predictions concentrated on a small area due to the low variance, but the predictions might be far from the true value, whereas estimates with high variance can be close to the true value in mean but are widely distributed.

action spaces. While the policy predicts the probability of choosing each possible action in the case of a discrete action space, a different approach is needed to predict the probabilities for a continuous action space. A way of doing so is to define the policy as the density function of a normal distribution [53, 54]. The density function is defined by

$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right),$$

with mean  $\mu$  and standard deviation  $\sigma$ . Note that in this specific equation,  $\pi$  is the number and not the policy. For the policy  $\pi_\theta$ , this results in

$$\pi_\theta(a | s) \doteq \frac{1}{\sigma(s, \theta)\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \theta))^2}{2\sigma(s, \theta)^2}\right).$$

Hence, what is now learned is not a model predicting the individual probabilities for the actions, but one that predicts the mean and standard deviation of a normal distribution from which a real valued action can be sampled.

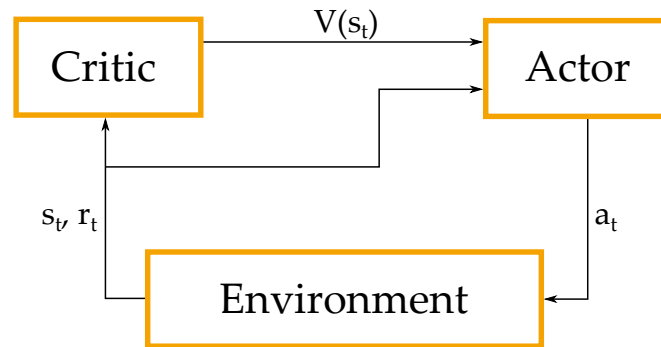


Figure 3.6.: General flow of RL when using actor-critic algorithms. The actor interacts with the environment by performing an action. The critic learns a value function in order to provide information for the actor about the quality of the chosen action.

### Actor-Critic

The third widespread class of RL methods is the class of *actor-critic* algorithms. It combines the approaches of directly learning a policy approximation using policy gradients and learning a value function approximation that is allowed to use bootstrapping. Hereby, *actor* refers to the part of the algorithm that decides what to do, i.e., the policy, whereas *critic* describes a component providing information about the quality of the chosen action, which is the task of the value function approximator. The general flow of actor-critic algorithms is shown in Figure 3.6.

In this context, unbiased policy gradient methods are often referred to as actor-only, value-based methods as critic-only algorithms. In many actor-critic algorithms, the critic replaces the unbiased reinforcement baseline [60, 52, 61]. Therefore, it usually approximates the state value function  $V^{\pi_\theta}(s)$  by estimating the value of the agent for being at a specific state. The resulting difference of state-action value and state value forms the *advantage*  $A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$  of performing action  $a$  in state  $s$  over the estimated value in state  $s$ , which leads to the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d^{\pi_\theta}(\cdot), a \sim \pi_\theta(\cdot | s)} [\nabla_\theta \log \pi_\theta(a | s) A^{\pi_\theta}(s, a)],$$

whereby we refer to  $A^{\pi_\theta}(s_t, a_t)$  as  $A_t^{\pi_\theta}$ . The goal of using actor-critic approaches is to furtherly reduce the variance, which can lead to a faster convergence than it is the case



---

for unbiased policy gradient algorithms [62]. At the same time, the abilities to handle continuous action spaces and naturally exploring the environment are adopted from actor-only methods. The downside, however, is the introduction of bias in exchange for the lower variance, entailing a trade-off between bias and variance. The newly introduced bias comes with the additional issue of the convergence guarantees of actor-only methods no longer being existent. Also, regarding non-stationary environments, i.e., the model of the environment changes (the feedback for identical given state-action pairs) over time, actor-critic approaches might not be appropriate due to the critic might not be able to adapt to the changes fast enough and, thus, might not provide meaningful information [63]. However, actor-critic methods have shown to be superior in many cases, which is, together with the many advantageous characteristics, why their use became widespread.

### 3.5.3. Off-/On-Policy Learning

The exploration an agent provides can be handled in two different ways: on-policy or off-policy. More precisely, the type of algorithm determines which policy the agent uses for sampling trajectories that are used to compute the policy update. An on-policy algorithm uses the same policy for exploration as the one that is learned, i.e., the one it assumes to be the best policy to maximize the objective at this time. Off-policy methods, on the contrary, collect samples using a different policy and, thus, make their policy updates based on a different state visitation distribution than the one induced by the policy that is updated. The *behavior policy*, which is used to sample trajectories in such a case, can be an entirely separate policy from the one that is being learned or involve the same. We call the policy we learn about the *target policy* [53]. A value-based example for an off-policy approach using a behavior policy that makes use of the target policy is DQN, which chooses a random action with a specified probability and draws an action from the target policy otherwise. This method is called  $\epsilon$ -greedy, referring to the said probability as  $\epsilon$ . The introduced Q-Learning algorithm can also be considered as an off-policy approach since it performs updates based on the Q-values of the actions the target policy would have chosen, even if the behavior policy chose different actions. The on-policy equivalent of Q-Learning is SARSA [64], which uses the update

$$\hat{Q}^\pi(s_t, a_t) \leftarrow \hat{Q}^\pi(s_t, a_t) + \alpha(r_t + \gamma \hat{Q}^\pi(s_{t+1}, a_{t+1}) - \hat{Q}^\pi(s_t, a_t)),$$

where the Q-value of the next chosen action is taken into account instead of using the action with the highest value.

---

Regarding the use of a behavior policy that is different from the target policy in the context of policy gradients, it should be considered to rather use an off-policy correction method instead of naively applying off-policy learning [65] (see Section 4.2). However, many policy gradient methods use on-policy exploration by default because of their stochastic nature.

### 3.5.4. Deep Reinforcement Learning

With the success of deep learning, it was a natural consequence to apply NNs in the context of RL. The introduction of convolutional neural networks opened up the possibility of dealing with high dimensional state spaces, e.g., images of a computer game in pixels. In the case of DQN, which reached human performance in several games [66, 3], the NN is used as a function approximator representing the state-action value function. However, as DQN is a value-based algorithm, it is still only able to handle discrete action spaces. This issue does not occur when using the policy gradient or actor-critic approach, which have also been adapted to work with NNs approximating the policy, of which DDPG is an example [19].

NNs come with the advantage of being universal function approximators. That is, in theory, they can learn functions of arbitrary complexity. While we could use an approximator of a fixed degree, e.g., quadratic, to approximate a function, the approximator could never learn a function of a higher degree. Therefore, we would restrict the potential of the approximator to learn arbitrary functions. While this restriction can be advantageous if we know the complexity of the function that is to be approximated, it causes problems if the function in question is unknown. As we usually do not know a lot about the function we want to approximate in RL (value function or policy), the use of NNs can facilitate the search for an appropriate model.

### 3.5.5. Exploration and Exploitation

We already got a minor impression of what exploration and exploitation mean in the previous sections. However, there is a variety of issues that come along with these two expressions. The most important ones are explained next.

---

## Reward Function

To fully understand the problem, it is useful to take a closer look at the reward function and the accompanying issues. The reward function is responsible for providing a reward in exchange for a state-action pair. However, this function can differ significantly throughout different environments in terms of its complexity and the amount of information it provides. A poorly defined reward function can cause the agent to take a long time to learn the desired task or hardly learn it at all. These issues that make the learning process hard could, for example, be *sparse rewards* or the presence of local optima [67, 68], to name two common ones. The former denotes environments that provide rewards very rarely, e.g., when solving a sub-task such as finding a door leading in the goal direction in a maze, or even only when reaching the ultimate goal. The latter describes a reward function that does not make it clear if a specific state is the best one the agent can reach by only exploring in the direct surrounding. An example is an agent in the real world being stranded on an island. Without any knowledge of the world, staying on the island seems to be the best option when only exploring the sea around the island with a boat. The agent has to move further away from the island that is assumed to be the optimal location so far to find even better conditions than the ones on the island.

## Lack of Exploration

A very common problem of standard RL algorithms is the lack of sufficient exploration [15]. While value-based algorithm often rely on methods like  $\epsilon$ -greedy, policy gradient methods mainly follow a probability distribution that often leads to states they already assume to be advantageous. Both are suboptimal for acquiring knowledge and understanding of the entire environment if it is not trivial. Especially regarding the case of a sparse rewards environment, it is hard to find a reward that is far away from the initial state when never finding a reward telling the agent what to do.

In the presence of local optima, the problem of premature convergence might occur. This example fits the island example above. The agent finds a reward that is better than others it has collected up to this time, causing the policy to converge to this point, assuming it to be the best option. However, there might still be an even better reward that is just harder to find, which will possibly never be found because of the convergence to the local optimum that already took place.

---

## Exploration-Exploitation Trade-off

It is a big and, to a large extent, unsolved issue to determine under which circumstances it is appropriate to keep exploring the environment and when to exploit the policy learned so far. Since an agent usually has no prior knowledge of the environment, it can never know if it has already seen every possible observation, let alone if the behavior learned so far is optimal. Common approaches are to decrease the level of exploration over time [3] or let the agent decide how much exploration is advantageous, as it is done using the stochasticity of policy gradient methods [54].

---

## 3.6. Proximal Policy Optimization

One of the most popular actor-critic algorithms in recent times is the PPO [52] algorithm. Besides its relative ease of implementation, which comes with the fact of it being a first-order algorithm (no second-order differentiation needed, in contrast to Natural Policy Gradient [69], for example), it convinces with an outstanding performance on complex tasks and high sample efficiency. As this algorithm uses NNs for the function approximations, it falls into the Deep RL category.

In order to understand PPO, it is useful to have a look at Trust Region Policy Optimization (TRPO) [61] first, which the idea of PPO is built upon.

### 3.6.1. Insight into TRPO

TRPO uses the idea of importance sampling to build its objective function. Importance sampling enables an estimation of the expected outcome of a function  $f(X)$  given a random variable  $X$  with probability distribution  $p$  by sampling from a different distribution  $q$ . This estimation is possible by using the *importance sampling ratio*  $p(X)/q(X)$  as a factor and, thus, forming the expectation

$$\mathbb{E}_{X \sim p(\cdot)} [f(X)] = \mathbb{E}_{X \sim q(\cdot)} \left[ \frac{p(X)}{q(X)} f(X) \right].$$

The intuition of importance sampling is to weight the outcome of  $f(X)$  with a low factor if the probability that a sample would have been drawn by the "true" distribution  $p$  is much

---

lower than the probability according to  $q$  and, thus, lower the impact of this sample [53]. The same principle applies to the other way around. Transferred to the field of RL, this technique can be used to apply off-policy learning, building the policy update by sampling from a different state visitation distribution than the one induced by the target policy. That is, we can modify the policy gradient so that we can update the target policy based on transitions collected by the behavior policy (see Section 4.2). TRPO makes use of this idea in a slightly different way by using the target policy to sample trajectories but using samples collected by an old policy version for updating the policy. For the sake of a clearer notation, we assume  $\pi$  to be parameterized with  $\theta$  but do not write it out. The resulting objective function is

$$J(\theta) = \mathbb{E}_{s \sim d^{\pi_{\text{old}}}(\cdot), a \sim \pi_{\text{old}}(\cdot|s)} [\rho_{\theta}(s, a) A^{\pi_{\text{old}}}(s, a)], \text{ with } \rho_{\theta}(s, a) = \frac{\pi(a | s)}{\pi_{\text{old}}(a | s)}.$$

However, the importance sampling ratio can have a high variance if the two distributions are very different, possibly leading to very big or very small update steps. TRPO aims to solve the optimization problem as a constrained one, bounding the Kullback-Leibler (KL) Divergence (a measure for the inequality of two probability distributions) of the target policy and old policy to a maximal value. This constraint creates a *trust region*, ensuring not to perform update steps that are too big. However, using the KL Divergence makes the algorithm computationally expensive.

### 3.6.2. PPO

PPO follows an idea very similar to the one of TRPO. While the algorithm also comes with the option of using the KL Divergence, we use the option that does not rely on such a measure since it appears to be the better performing approach [52]. The main difference between TRPO and PPO is how they ensure the trust region. Instead of involving the KL Divergence, PPO clips the importance sampling ratio between the updated policy and the old one. More precisely, it creates a lower bound (or pessimistic bound) on the objective [52], resulting in

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_{s \sim d^{\pi_{\text{old}}}(\cdot), a \sim \pi_{\text{old}}(\cdot|s)} [\min(\rho_{\theta}(s, a) A^{\pi_{\text{old}}}(s, a), \text{clip}(\rho_{\theta}(s, a), 1 - \epsilon, 1 + \epsilon) A^{\pi_{\text{old}}}(s, a))],$$

with clipping parameter  $\epsilon$  (which is not related to the  $\epsilon$  of the mentioned  $\epsilon$ -greedy method).

The policy function approximator, as well as the value function approximator we learn, are represented by NNs. It is possible to share a subset of the parameters of these networks

---

so that they collectively learn features for the observations. We can use a single optimizer to update the networks with respect to a collective objective

$$J(\theta) = \mathbb{E}_{s \sim d^{\pi_{\text{old}}(\cdot)}, a \sim \pi_{\text{old}}} [J^{\text{CLIP}}(\theta) - cJ^{\text{VF}}(\theta)],$$

where  $J^{\text{VF}}$  is the squared error loss of the value function, and  $c$  denotes a constant factor, e.g., 0.5 [52]. Additionally, an optional bonus for the entropy in the policy (see Section 5.4 for more information on entropy) can be included to avoid premature convergence, which we do not make use of since we want the exploration to be assured by using IM. To update the models of PPO, we use epochs and batches as it is often done using Deep Learning. An iteration of PPO can be regarded as three parts. The first one is to sample transitions for  $T$  steps according to  $\pi_{\text{old}}$ . The second part is to compute the advantage estimates  $\hat{A}_t^{\pi_{\text{old}}}$  for each timestep  $t$ . After this, the policy and value function are optimized for  $K$  epochs with  $T/B$  mini-batches each, where  $B$  is the batch size. During the iteration, the old policy  $\pi_{\text{old}}$  is held fixed and adopts the parameters of the new policy afterward.

---

### 3.7. Intrinsic Motivation

---

Starting from the sparse exploration methods we learned of so far (see Section 3.5.5), the potential for a more efficient exploration is high, especially in sparse rewards environments. Since RL is similar to the way animals learn, it could make sense to have a look at how animals decide when and what to explore. It is to be assumed that, besides following the behavior they know to be good, animals like humans have a type of motivation leading to new places and trying new things. These motivations can be diverse, e.g., curiosity or the will to see something new in order to learn if it is good or bad. Such a kind of motivation is called *Intrinsic Motivation (IM)* and is to be regarded in contrast to Extrinsic Motivation [70, 71]. IM can be described as a motivation encouraging an organism to perform actions for its own satisfaction, e.g., out of curiosity or to have fun [70]. This kind of behavior is independent of external reinforcement like punishments or rewards [70]. Instead of regarding the process of learning only as the result of direct feedback the agent receives from the environment, we can view the internal state of the agent as an independent internal environment, reflecting the motivation of the agent [72].

In recent years, such a behavior has been tried to be adopted in the area of RL. Since then, a variety of IM methods with the goal of bringing a better, more efficient way of learning an advantageous behavior has been developed. There are two [73] or three [71] classes of IM, depending on which definition considered.

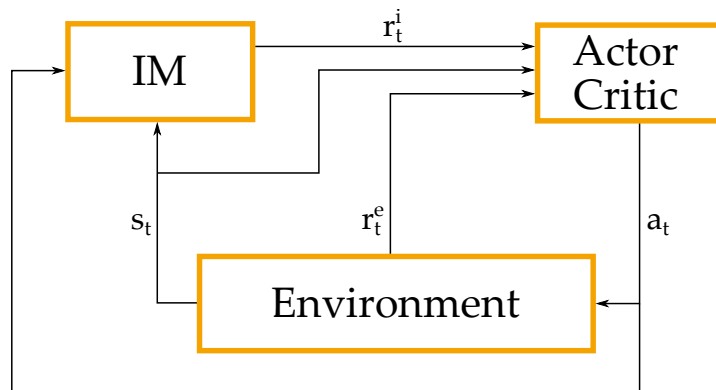


Figure 3.7.: General flow of intrinsically motivated RL using an actor-critic method. The environment provides the current state and extrinsic reward. The IM method produces an intrinsic reward based on the state and potentially on the chosen action, depending on the used IM method. Thereupon, the agent chooses an action. The action probabilities are updated based on a combination of intrinsic and extrinsic rewards.

However, we concentrate on a single one, namely *knowledge acquisition* [73] or *knowledge based models* [71] respectively. Methods of the remaining class(es) consist of, e.g., algorithms that aim to autonomously acquire task-independent skills that can be adopted to help to reach specific goals later [73]. The class of knowledge acquisition methods, on the other hand, often deals with the exact issue mentioned before: the efficient exploration of the environment. In general, this class aims to acquire knowledge about the environment, including its functionality and connections between states [73].

Therefore, the IM method produces an additional reward for each step, called the *intrinsic reward*  $r^i$ . In this context, we call the "normal" reward, which is provided by the environment for taking an action at a specific state, the *extrinsic reward*  $r^e$ . Usually, the intrinsic reward is just added to the extrinsic reward by which the objective can be optimized with respect to the weighted compound reward  $r_t = c_1 r_t^e + c_2 r_t^i$ , with constant factors  $c_1$  and  $c_2$  at timestep  $t$  [10, 74, 25, 73]. One example of how such an intrinsically motivated RL process can be viewed is depicted in Figure 3.7.

For our experiments, we use two very different IM methods in terms of their complexity and overall underlying methodology. The first one is a simple form of state novelty, and

---

the second one is the Intrinsic Curiosity Module (ICM) [10]. Said methods are explained next.

### 3.7.1. State Novelty

The basic principle of state novelty, as the name indicates, is to observe new states, i.e., encourage the agent to visit states it has never seen before or has visited very rarely [75, 76]. State novelty, hence, can be considered as a count-based method, counting the visitations of each state or observation. To achieve the goal of encouraging the visitation of states that have not been visited often yet, the reciprocal of the counted state visitation can be used as an intrinsic reward signal

$$r_t^i = \frac{1}{N(s_t)},$$

with  $N(s)$  denoting the visitation count of state  $s$  [73]. For the reason of the agent having to store a number for every individual state, this method is only feasible in case of a discrete and not too large state space. There has already been proposed a variety of novelty-based IM methods, including multiple approaches being able to handle continuous state spaces, e.g., by using density models to approximate the visitation count [13]. However, since we only consider environments with a small, discrete state space (see Section 5.2), and the above-proposed method is efficient for this kind of environment [73], there is no need for a more complex implementation for our purpose. Further mention of "novelty" in this thesis refers to the introduced basic state novelty method.

### 3.7.2. Prediction Error

The second type of IM we consider is based on learning the transitions of states of the environment, that is, a model  $\hat{f}(s, a)$  is learned approximating the true transition function by predicting the next state given a state-action tuple [73]. This prediction can be described by

$$\hat{\phi}(s_{t+1}) = \hat{f}(\phi(s_t), a_t),$$

where  $\phi$  is an optional function that maps the state to a feature space, and  $\hat{\phi}(s_{t+1})$  is the predicted next state (or predicted feature encoding of the same) [73]. Using this method, it is desired to lead the agent to perform actions of which it cannot accurately predict the outcome (the next state). The intention of this approach is that actions entailing an uncertain behavior have probably not been sufficiently explored yet, i.e., there have not



been enough according transitions collected to train the model the respective behavior. This insufficient training of a specific transition can be measured by calculating how wrong the prediction of the model is with respect to the true transition. This measure can be used as an intrinsic reward and gives the IM type its name. It can be formalized by

$$r_t^i = \left\| \phi(s_{t+1}) - \hat{f}(\phi(s_t), a_t) \right\|.$$

This model predicting the next state is also called *forward model* [73] and can be represented, for example, by an NN.

### Intrinsic Curiosity Module

Using the prediction error as an exploration bonus comes along with a problem that especially takes place in environments with a high level of stochasticity. Stochastic noise in observations does not provide additional useful information and could increase the prediction error. An example of such noise is the movement of tree leaves [10]. The ICM is a specific algorithm that makes use of the prediction error. However, it improves the standard approach by a method that aims to reduce the impact of the aforementioned noise. Therefore, it uses an *inverse model* in addition to the forward model, whereby both, the forward and inverse model, are represented by NNs. The inverse model intends to learn to predict the action  $a_t$  the agent has chosen given the state  $s_t$  at which it performed the action and the successor state  $s_{t+1}$ . As a by-product, it learns the function  $\phi(s)$ , which generates an encoding of a specific state  $s$ . A neuron layer of the inverse model, which the original state representation passes through for the action prediction, is trained to encode the input (i.e., the state) so that the action can be predicted as well as possible. Since noise does not contribute to a better prediction, it does not get encoded. The output of this particular layer is used as the feature representation  $\phi(s)$ . The predicted action between two states  $s_t$  and  $s_{t+1}$  can be described by

$$\hat{a}_t = g_{\theta_I}(s_t, s_{t+1}),$$

where  $g_{\theta_I}$  is the NN predicting the action with parameters  $\theta_I$ . The network is trained by minimizing the loss  $L_I$  arising from the discrepancy between the actually chosen action  $a_t$  and the prediction  $\hat{a}_t$  [10].

As the prediction of the chosen action between two states is a classification task in the case of a discrete action space, we use the mean Cross-Entropy loss

$$L_I(\mathbf{a}, \hat{\mathbf{a}}) = -\frac{1}{N} \sum_{i=1}^{|\mathbf{a}|} \mathbf{a}_i \log(\hat{\mathbf{a}}_i)$$

---

that is commonly used for the optimization of classifications. Hereby,  $\hat{\mathbf{a}}$  consists of the probabilities for each possible action predicted by the model and  $\mathbf{a}$  contains the corresponding true probabilities, i.e., 1 for the actually chosen action and 0 for all others.

The forward model is optimized using the squared error loss

$$L_F(\phi(s_t), \hat{\phi}(s_{t+1})) = \frac{1}{2} \left\| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right\|_2^2.$$

The loss of the forward model is then used as the intrinsic reward with the customization of multiplying it with a scaling factor  $\eta$  beforehand. This factor scales the intrinsic reward in order to keep it small enough to avoid nullifying the extrinsic reward while, at the same time, keep it high enough to ensure a sufficient exploration. The composed reward is then a simple addition of the intrinsic and extrinsic reward [10].

The forward and inverse model are optimized collectively using a compound loss

$$L_{F+I} = (1 - \beta)L_I + \beta L_F,$$

with a scalar  $0 \leq \beta \leq 1$  [10]. A visualization of the architecture of the entire ICM can be seen in Figure 3.8.

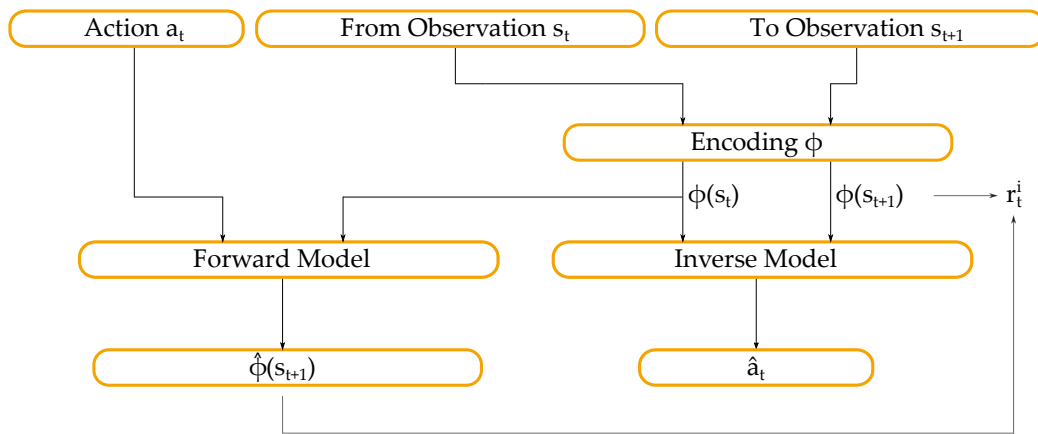


Figure 3.8.: The general structure of the ICM [10]. The observation of the states  $s_t$  and  $s_{t+1}$  are encoded, resulting in the state encodings  $\phi(s_t)$  and  $\phi(s_{t+1})$ , which are used to predict the action performed in between by the inverse model and, thus, are trained by optimizing the according loss. The action, together with the encoding  $\phi(s_t)$ , is fed into the forward model to predict the next state encoding. The error of this prediction with respect to the true next state encoding induces the intrinsic reward  $r_t^i$ .

---

## 4. Explicit Intrinsic Motivation

---

We want to experiment with different ways of incorporating IM in RL, including on-policy as well as off-policy approaches, which is why we do not use an algorithm as our baseline that is off-policy by design such as ACER or DDPG. Thus, we use PPO due to its empirically shown performance, its ability to adapt to continuous state and action spaces for potential further experiments, and its ease of implementation and, thus, modification. To use PPO for the different approaches, we apply specific modifications that are explained in this chapter.

We distinguish between two classes of algorithms that make use of some kind of IM, namely Implicit Intrinsic Motivation (IIM) and Explicit Intrinsic Motivation (EIM) algorithms. The former describes the use of IM as stated in Section 3.7, comprising an intrinsic reward  $r^i$  that is added to the extrinsic reward  $r^e$ . The latter approaches the integration of the IM by learning separate function approximators for the intrinsic and extrinsic rewards as Szita et al. [22] suggested and realized by Morere et al. [23] and Parisi et al. [24] on a value-based level. Besides learning two separate value functions, we also train two individual policies for this purpose. The resulting *greedy policy*  $\pi_{\theta_g}(a | s)$  is trained only considering the extrinsic rewards while the *exploration policy*  $\pi_{\theta_e}(a | s)$  only makes use of the intrinsic rewards. To sample trajectories during training, we make use of a behavior policy  $\pi_{\theta_b}(a | s)$  that is a combination of both policies. Figure 4.1 shows the idea of the EIM approach, as opposed to the IIM approach depicted in Figure 3.7. When using the IIM approach, we also refer to the policy we learn as the *greedy policy*.

For more clarity, we again get rid of the  $\theta$  in the notation of the policies. That is, we write  $\pi$  instead of  $\pi_\theta$ . In the case of an indexed  $\theta$ , we replace the whole expression of the parameter with the index. For example,  $\pi_{\theta_b}$  becomes  $\pi_b$  and  $\pi_{\theta_{b,\text{old}}}$  becomes  $\pi_{b,\text{old}}$ .

As we use the same methods for updating both individual policies, we use the notation  $\pi_i$  when deriving the methods and implicitly mean  $\pi_g$  or  $\pi_e$ , depending on which one the method is applied to. Analogously, we use  $\theta_i$  instead of  $\theta_g$  or  $\theta_e$ .

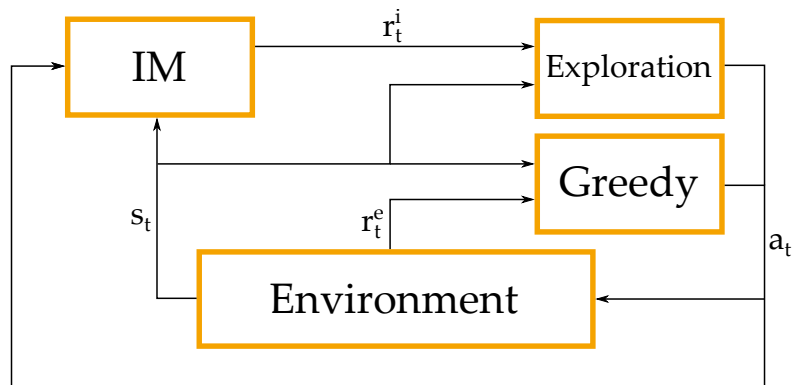


Figure 4.1.: General process flow when using the EIM approach, where "Exploration" represents the exploration part of the agent, i.e., the policy and value function considering the intrinsic rewards, and "Greedy" denotes the part only considering the extrinsic rewards.

By making the intrinsic motivation explicit, we aim to tackle multiple issues of intrinsically motivated algorithms of which, most notably, are the possibility of a noisy greedy policy in the case of a non-converging intrinsic motivation and the tendency of premature convergence due to low intrinsic rewards with respect to the extrinsic rewards.

Moreover, with the possibility of manually regulating the impact of the exploration policy throughout the learning process, we can make use of the greedy policy without direct noise induced by intrinsic rewards at any time. Moreover, we can keep the exploration high, even if the greedy policy already converged to a local optimum. Also, high intrinsic rewards no longer prevent the greedy policy from converging since they have no direct impact on its learning.

Besides the potential positive aspects, the explicit way of using intrinsic motivations also comes with some new questions that have to be addressed, such as the way of sampling the trajectories and the shape of the policy gradient. After a detailed explanation of the off-policy approach we use, the application of an alternative on-policy approach is discussed in this chapter. At last, the method of determining which policy to use for decision making is presented.

---

## 4.1. When to explore

---

The desirable property of policy gradient methods of naturally exploring the environment and, thus, not relying on a manual trade-off between exploration and exploitation is being waived by using an EIM approach. Following the idea of  $\epsilon$ -greedy [3], we use a parameter  $h \in [0, 1]$  for determining the impact of the exploration policy on the decision process while sampling a trajectory. The parameter represents the probability of drawing the action  $a_t$  from  $\pi_e$ . We can describe the behavior policy that is used to sample trajectories with

$$\pi_b(a | s) = h\pi_e(a | s) + (1 - h)\pi_g(a | s),$$

where  $\pi_b$  implicitly has the parameter set  $\theta_b = \theta_g \cup \theta_e$ . As opposed to IIM, using EIM, we have the option to adjust the impact of the IM method during the learning process and even during a single episode. This possibility of control enables to not only choose  $h$  based on the training time but also based on timestep-specific information such as the state or action probabilities (see Section 5.4). Therefore and in order to generalize the derivations of the gradient in Section 4.3 for both policies, we also define  $\alpha(i | s)$  as the probability of choosing policy  $\pi_i$  in state  $s$ . Hence, another way of describing the behavior policy is

$$\pi_b(a | s) = \sum_{i \in \{e, g\}} \alpha(i | s) \pi_i(a | s) = \alpha(e | s) \pi_e(a | s) + \alpha(g | s) \pi_g(a | s).$$

When using IIM methods, the only direct impact we have on the level of exploration is to scale the intrinsic rewards by a constant factor  $\eta$  for both IM methods.

---

## 4.2. Off-Policy Approach

---

Since PPO is not naturally designed to be used as an off-policy method with a behavior policy that can be very different from the target policy, we need an approach to correct the update steps when sampling from such a different policy. We, therefore, make use of the off-policy policy gradient theorem [18].

When using this theorem, we maximize the values of every state according to the policy  $\pi_i$  that is to be optimized, weighted by the state visitation distribution of the behavior policy  $d^{\pi_b}$  instead of  $d^{\pi_i}$  as usual, which leads to the objective function

$$J(\theta_i) = \sum_{s \in S} d^{\pi_b}(s) V^{\pi_i}(s).$$

Despite this objective bringing a bias [77, 78], which comes with the sampling from the state visitation distribution under the behavior policy, empirical results showed its potential and successes [79, 19, 20]. An approximation of the gradient of the objective is given by

$$\begin{aligned}
\nabla_{\theta_i} J(\theta_i) &\approx \sum_{s \in S} d^{\pi_b}(s) \sum_{a \in A} \nabla_{\theta_i} \pi_i(a | s) Q^{\pi_i}(s, a) \\
&= \mathbb{E}_{s \sim d^{\pi_b}(\cdot)} \left[ \sum_{a \in A} \nabla_{\theta_i} \pi_i(a | s) Q^{\pi_i}(s, a) \right] \\
&= \mathbb{E}_{s \sim d^{\pi_b}(\cdot)} \left[ \sum_{a \in A} \pi_b(a | s) \frac{\pi_i(a | s)}{\pi_b(a | s)} \frac{\nabla_{\theta_i} \pi_i(a | s)}{\pi_i(a | s)} Q^{\pi_i}(s, a) \right] \\
&= \mathbb{E}_{s \sim d^{\pi_b}(\cdot), a \sim \pi_b(\cdot | s)} \left[ \frac{\pi_i(a | s)}{\pi_b(a | s)} \frac{\nabla_{\theta_i} \pi_i(a | s)}{\pi_i(a | s)} Q^{\pi_i}(s, a) \right] \\
&= \mathbb{E}_{s \sim d^{\pi_b}(\cdot), a \sim \pi_b(\cdot | s)} \left[ \frac{\pi_i(a | s)}{\pi_b(a | s)} \nabla_{\theta_i} \log \pi_i(a | s) Q^{\pi_i}(s, a) \right] \tag{4.1}
\end{aligned}$$

for the target policy  $\pi_i$  [18]. Equation (4.1) looks similar to the policy gradient theorem with the differences of sampling from the behavior policy and weighting the gradient with an importance sampling ratio between the target policy and the behavior policy. As we use PPO, the actual policies present during the process of trajectory sampling are  $\pi_{i,\text{old}}(a | s)$  and  $\pi_{b,\text{old}}(a | s)$ . Hence, we replace the ratio accordingly. Additionally replacing the logarithm of the target policy with the ratio between the new and the old policy, as we do when using PPO, yields the objective

$$J(\theta_i) = \mathbb{E}_{s \sim d^{\pi_{b,\text{old}}}(\cdot), a \sim \pi_{b,\text{old}}(\cdot | s)} \left[ \frac{\pi_{i,\text{old}}(a | s)}{\pi_{b,\text{old}}(a | s)} \frac{\pi_i(a | s)}{\pi_{i,\text{old}}(a | s)} Q^{\pi_i}(s, a) \right] \tag{4.2}$$

$$= \mathbb{E}_{s \sim d^{\pi_{b,\text{old}}}(\cdot), a \sim \pi_{b,\text{old}}(\cdot | s)} \left[ \frac{\pi_i(a | s)}{\pi_{b,\text{old}}(a | s)} Q^{\pi_i}(s, a) \right]. \tag{4.3}$$

Equation (4.3) includes importance sampling similar to the normal use of TRPO and PPO, with the difference of using the old behavior policy instead of the old greedy policy. However, as PPO aims to limit the ratio of the new and old target policy [52], we use Equation (4.2) and apply clipping to the according ratio. To approximate the advantage  $A_t^{\pi,\lambda} = G_t^\lambda - V^{\pi_i}(s_t)$ , we use the off-policy  $\lambda$  return [18] described by

$$G_t^\lambda = r_{t+1} + (1 - \lambda) \gamma \hat{V}^{\pi_i}(s_{t+1}) + \lambda \gamma \frac{\pi_i(a | s)}{\pi_b(a | s)} G_{t+1}^\lambda,$$

---

```

for iteration=1, 2, ... do
  Collect transitions  $(s_t, a_t, s_{t+1}, r_t^e)$  using  $\pi_{b,\text{old}}$  for  $T$  timesteps
  Compute intrinsic rewards  $r_{1..T}^i$  based on the IM
  Compute greedy advantage estimates  $\hat{A}_{1..T}^{\pi_g,\text{old}}$  based on the extrinsic rewards
  Compute exploration advantage estimates  $\hat{A}_{1..T}^{\pi_e,\text{old}}$  based on the intrinsic rewards
  for epoch=1, 2, ...,  $K$  do
    for minibatch=1, 2, ...,  $T/B$  do
      Optimize objective  $J^{\text{off}}(\theta_g)$  w.r.t.  $\theta_g$  using  $\hat{A}^{\pi_g,\text{old}}$  with minibatch size  $B \leq T$ 
      Optimize objective  $J^{\text{off}}(\theta_e)$  w.r.t.  $\theta_e$  using  $\hat{A}^{\pi_e,\text{old}}$  with minibatch size  $B \leq T$ 
    end
  end
end

```

**Algorithm 1:** Off-Policy EIM PPO

with a parameter  $\lambda \in [0, 1]$  for a bias-variance trade-off. Consequently, the off-policy objective we aim to optimize is

$$J^{\text{off}}(\theta_i) = \mathbb{E}_{s \sim d^{\pi_{b,\text{old}}}(\cdot), a \sim \pi_{b,\text{old}}(\cdot|s)} \left[ \frac{\pi_{i,\text{old}}(a|s)}{\pi_{b,\text{old}}(a|s)} J^{\text{CLIP}}(\theta_i) \right],$$

$$\text{with } J^{\text{CLIP}}(\theta_i) = \min \left( \rho_{\theta_i}(s, a) A^{\pi_{i,\text{old}},\lambda}(s, a), \text{clip}(\rho_{\theta_i}(s, a), 1 - \epsilon, 1 + \epsilon) A^{\pi_{i,\text{old}},\lambda}(s, a) \right).$$

Since, in our case, the greedy policy and the exploration policy are trained separately,  $\pi_i$  is to be substituted with  $\pi_g$  or  $\pi_e$  and  $\theta_i$  with  $\theta_g$  or  $\theta_e$  respectively, depending on which one should be updated. Algorithm 1 shows the pseudo-code of PPO as an off-policy EIM approach.



---

### 4.3. On-Policy Approach

---

Using off-policy approaches as described in Section 4.2 is controversial due to the use of importance sampling and the accompanying possibility of very high policy gradients in case of a performed action being chosen much more likely by the target policy than by the behavior policy. Moreover, it was shown that off-policy algorithms could provide poor learning behavior if the greedy policy and the behavior policy are too different [80]. Therefore, we also consider an on-policy approach that still handles the exploration policy separately. We still sample trajectories according to the behavior policy in the same way as we do when using the off-policy approach. However, the policy gradient for each policy is not based on all collected transitions but only on the ones of which the corresponding action was chosen by the respective policy. For example, a transition that was collected by following policy  $\pi_g$  is not considered when forming the gradient for  $\pi_e$ . Hence, the policies do not explicitly learn from the experience of a different policy. Nonetheless, they can still benefit from each other as the chance exists to visit states that would have never or very rarely been visited by using only one of them. Performing this method, we use a modification of the rewards, which is defined by

$$\hat{R}_i(s_t, a_t) = \begin{cases} R_i(s_t, a_t) & \text{if } k_t = i, \\ 0 & \text{otherwise,} \end{cases}$$

whereby  $k_t \sim \alpha(\cdot | s_t)$  represents the specific index sampled at timestep  $t$ . In other words,  $k_t = i$  means that policy  $\pi_i$  was chosen to decide the according action  $a_t$ . The reward function  $R_i(s, a)$  is the one provided for the respective policy  $\pi_i$  that is to be updated, i.e., the extrinsic reward function if it is the greedy policy  $\pi_g$  or the intrinsic reward function in the case of the exploration policy  $\pi_e$ . In this way, a policy can only be rewarded for a decision it made itself and not for one made by the other policy. The accompanying value function, hence, is defined by

$$V^{\pi_i}(s) = \sum_{a \in A} \pi_b(a | s) \mathbb{E}_{k \sim \alpha(\cdot | s)} \left[ \hat{R}_i(s, a) + \gamma \sum_{s' \in S} V^{\pi_i}(s') P(s' | s, a) \right].$$

Analogously, the state-action value function is described by

$$Q^{\pi_i}(s, a) = \mathbb{E}_{k \sim \alpha(\cdot | s)} \left[ \hat{R}_i(s, a) + \gamma \sum_{s' \in S} V^{\pi_i}(s') P(s' | s, a) \right].$$

Moreover, we can denote the state visitation distribution of both individual target policies as

$$d^{\pi_i}(s) = \sum_{s_0 \in S} \sum_{t=0}^{\infty} \gamma^t \alpha(i | s) Pr(s_t = s | s_0, \pi_b) d_0(s_0),$$

which is similar to the state visitation distribution of the behavior policy but takes the probability of  $\pi_i$  being chosen to decide the last action of a trajectory into account.

When we observe the gradient of the value function, we get

$$\begin{aligned} \nabla_{\theta_i} V^{\pi_i}(s) &= \nabla_{\theta_i} \sum_{a \in A} \pi_b(a | s) E_{k \sim \alpha(\cdot | s)} \left[ \hat{R}_i(s, a) + \gamma \sum_{s' \in S} V^{\pi_i}(s') P(s' | s, a) \right] \\ &= \sum_{a \in A} \alpha(i | s) \nabla_{\theta_i} \pi_i(a | s) E_{k \sim \alpha(\cdot | s)} \left[ \hat{R}_i(s, a) + \gamma \sum_{s' \in S} V^{\pi_i}(s') P(s' | s, a) \right] \\ &\quad + \gamma \pi_b(a | s) \sum_{s' \in S} \nabla_{\theta_i} V^{\pi_i}(s') P(s' | s, a) \\ &= \sum_{a \in A} \alpha(i | s) \nabla_{\theta_i} \pi_i(a | s) Q^{\pi_i}(s, a) + \gamma \pi_b(a | s) \sum_{s' \in S} \nabla_{\theta_i} V^{\pi_i}(s') P(s' | s, a) \\ &= \sum_{a \in A} \alpha(i | s) \nabla_{\theta_i} \pi_i(a | s) Q^{\pi_i}(s, a) \\ &\quad + \gamma \pi_b(a | s) \sum_{s' \in S} \left( \sum_{a'} \alpha(i | s') \nabla_{\theta_i} \pi_i(a' | s') Q^{\pi_i}(s', a') \right. \\ &\quad \quad \left. + \gamma \pi_b(a' | s') \sum_{s'' \in S} \nabla_{\theta_i} V^{\pi_i}(s'') P(s'' | s', a') \right) \\ &\quad \quad P(s' | s, a) \\ &= \sum_{x \in S} \sum_{t=0}^{\infty} \gamma^t \alpha(i | x) Pr(s_t = x | s, \pi_b) \sum_{a \in A} Q^{\pi_i}(x, a) \nabla_{\theta_i} \pi_i(a | x). \end{aligned}$$

Now we define the objective each policy aims to optimize as

$$J(\theta_i) = \sum_{s \in S} d_0(s) V^{\pi_i}(s),$$

again with  $\theta_i$  being  $\theta_g$  or  $\theta_e$  depending on the current target policy. When forming the gradient of the objective function by substituting  $V^{\pi_i}(s)$  with  $\nabla_{\theta_i} V^{\pi_i}(s)$ , we receive the

---

on-policy policy gradient

$$\begin{aligned}
\nabla_{\theta_i} J(\theta_i) &= \sum_{s \in S} d_0(s) \sum_{x \in S} \sum_{t=0}^{\infty} \gamma^t \alpha(i | x) Pr(s_t = x | s, \pi_b) \sum_{a \in A} Q^{\pi_i}(x, a) \nabla_{\theta_i} \pi_i(a | x) \\
&= \sum_{x \in S} d_0(x) \sum_{s \in S} \sum_{t=0}^{\infty} \gamma^t \alpha(i | s) Pr(s_t = s | x, \pi_b) \sum_{a \in A} Q^{\pi_i}(s, a) \nabla_{\theta_i} \pi_i(a | s) \\
&= \sum_{s \in S} d^{\pi_i}(s) \sum_{a \in A} Q^{\pi_i}(s, a) \nabla_{\theta_i} \pi_i(a | s) \\
&= \sum_{s \in S} d^{\pi_i}(s) \sum_{a \in A} Q^{\pi_i}(s, a) \pi_i(a | s) \nabla_{\theta_i} \log \pi_i(a | s) \\
&= \mathbb{E}_{s \sim d^{\pi_i}(\cdot), a \sim \pi_i(\cdot | s)} [\nabla_{\theta_i} \log \pi_i(a | s) Q^{\pi_i}(s, a)]
\end{aligned}$$

of the policy  $\pi_i$  that we want to update, under consideration of the above-defined state visitation distribution.

The objective  $J^{\text{on}}(\theta_i)$  adapted to PPO that we aim to optimize is similar to  $J^{\text{off}}(\theta_i)$  with the only difference of not using the off-policy importance weight  $\pi_{i,\text{old}}(a | s) / \pi_{b,\text{old}}(a | s)$ , neither inside the off-policy  $\lambda$  return nor to weight the policy gradient. However, take notice of the fact that only the data collected using the respective policy that is to be updated is taken into account when forming the gradient.

Since each policy has fewer samples per iteration to estimate the gradient, the policies are less often updated as we still want to keep the batch size the same as for the other methods. A pseudo-code of PPO as an on-policy EIM approach can be seen in Algorithm 2.

---



---

```

for iteration=1, 2, ... do
  Collect transitions  $(s_t, a_t, s_{t+1}, r_t^e)$  using  $\pi_{b,\text{old}}$  for  $T$  timesteps
  Compute intrinsic rewards  $r_{1..T}^i$  based on the IM
  Replace all  $r_t^e$  with 0 if the exploration policy chose action  $a_t$ 
  Replace all  $r_t^i$  with 0 if the greedy policy chose action  $a_t$ 
  Compute greedy advantage estimates  $\hat{A}_{1..T}^{\pi_{g,\text{old}}}$  based on the extrinsic rewards
  Compute exploration advantage estimates  $\hat{A}_{1..T}^{\pi_{e,\text{old}}}$  based on the intrinsic rewards
  for epoch=1, 2, ...,  $K$  do
    for minibatch=1, 2, ...,  $T/B$  do
      Optimize objective  $J^{\text{on}}(\theta_g)$  w.r.t.  $\theta_g$  using  $\hat{A}^{\pi_{g,\text{old}}}$  with minibatch size  $B \leq T$ 
      Optimize objective  $J^{\text{on}}(\theta_e)$  w.r.t.  $\theta_e$  using  $\hat{A}^{\pi_{e,\text{old}}}$  with minibatch size  $B \leq T$ 
    end
  end
end

```

**Algorithm 2:** On-Policy EIM PPO

---

## 5. Experiments

---

With the intention of practically showing the accompanying advantages and disadvantages of making IM-based exploration explicit, in this chapter, the setup of several experiments is described after a brief introduction of the environment they are applied to. Primarily, the experiments aim to show the desired properties of the EIM approach mentioned in Chapter 4. In addition, we observe the behavior of agents learning only based on the intrinsic rewards. We do so to estimate the impact of the respective IMs on the behavior of an agent that is trained by a combination of both the intrinsic and extrinsic rewards.

---

### 5.1. General Settings

---

Besides the parameters determined experimentally, there are several hyperparameters set beforehand and kept constant for all experiments for the sake of computation time that would be required for the experimental determination of all possible parameters.

**Timesteps per Iteration:** To optimize the models, we sample an amount of 1024 transitions before performing the updates of an iteration. This number is a reasonable choice regarding computational resources and time needed for each iteration, and in most cases, it is enough to represent the current behavior of the agent.

**Training:** The training itself takes place in 1000 iterations. With 1024 steps per iteration, this equals an amount of 1,024,000 sampled transitions throughout the training, on which the updates are based.

**Evaluation:** After every fifth iteration, an evaluation over 10 episodes is performed. For the evaluation, only the greedy policy is used when using EIM, which allows us to see the actual performance the agent considers best regarding the extrinsic task.

**Learning Rate:** The learning rate of both, the exploration policy and the greedy policy, are set to  $5 \times 10^{-4}$  since learning rates between  $1 \times 10^{-4}$  and  $1 \times 10^{-3}$  have been empirically

---

shown to be advantageous. We jointly optimize the policy and value function by adding the loss of the value function, multiplied with a factor of 0.5, to the policy loss. The learning rate of the forward and inverse model of the ICM is set to  $5 \times 10^{-3}$ . These models are also optimized jointly, equally contributing to the loss.

**PPO Clipping Parameter  $\epsilon$ :** The parameter  $\epsilon$  used to determine the clipping range of PPO has a value of 0.2, as this is a value that showed to perform well [52] and is commonly used for many tasks.

**Epochs:** Each iteration, all involved models are optimized in five epochs. These models comprise the policies and value functions and, in the case of using the ICM, the forward and inverse model.

**Batch Size:** The batch size we use is 256 for the policies, value functions, and ICM models. Each epoch, the samples are shuffled, which leads to different losses per batch every epoch.

**Advantage  $\lambda$ :** We keep the parameter  $\lambda$ , which is responsible for the trade-off between the MC-caused variance and the bias introduced by the reliance on the learned value function when estimating the advantage, constant with a value of 1. Therefore, the policy updates fully rely on the MC rollout for the estimation of their current state-action values. The critic is updated using the advantage estimate as its target, which leads to an unbiased value function in terms of not using bootstrapping. Foregoing bootstrapping is desirable for our experiments concerning the use of IM and the accompanying non-stationary rewards. While we could make use of bootstrapping for the greedy policy in the case of using EIM methods, we want to keep the circumstances as similar as possible for all experiments to enable a fair comparison.

**Activation Function:** We use the Leaky ReLU activation function in all NNs. Thus, it is used for the hidden layers of the policies and value functions as well as for the ones of the ICM except for the encoding layer. The output of the policies is activated using the softmax function in order to receive probability-like values needed to form a stochastic policy.

**Discount Factor:** In order to optimize the long time quality of the policies, we use the commonly used discount factor 0.99.

**Gradient Clipping:** We make use of gradient clipping [30] for batch gradients with a norm of 0.5 or higher. This clipping is intended to avoid the problem of exploding gradients, i.e., very large update steps.

---

## 5.2. Grid World Environment

---

To keep the insights provided by the experiments as unaffected as possible by circumstances that were not taken into account inadvertently, as well as by unpredictable behavior produced by a stochastic environment, we use a simple deterministic environment. Additionally, the possibility of visualizing the experiments in an intuitive way is advantageous. Therefore, the environment we use is a small grid world with a discrete state and action representation. That is, the environment has a tabular shape, consisting of ten rows and ten columns per row. As each table cell acts as a field thinking of the environment like a game, we refer to them as just this: *fields*. Each field is represented by a number sorted ascending by row and column, whereby the top left table element has the representation 0 and the bottom right one is represented by 99. Furthermore, each table element can be of one of the following classes describing its functionality:

1. a *normal* field where the agent can be located without any reward being provided,
2. a terminal *goal* field ending the episode and returning a reward when entered, or
3. a *wall* field that is not accessible, causing the agent to stay at its previous position.

The observation an agent in this environment receives consists of the number representation of the field in which it is currently located in the form of a one-hot encoding. That is, the state representation is a 100-dimensional vector containing 99 zeros and a single 1 at the index of the number of the field it represents. This shape of observation ensures that the agent cannot guess the outcome of an action in a state it has not seen before since the elements of the input array are all fed into the policy NN as separate features without giving hints about positional relations.

The action space consists of the four actions *up*, *down*, *left*, and *right*, moving the agent in the respective direction except for the case of the agent trying to move on a wall field or out of the state space, i.e., the row or column of the new state would be smaller than 1 or higher than 10. In such a case, the agent stays in the same state, but it costs a timestep nonetheless.

In addition to the arrival at the goal field, exceeding a specific number of timesteps also ends an episode. This number is set to 50 for all our experiments.

The environment is set up in the form of a maze with several paths leading to dead ends and only a single way to most fields. Depending on the aim of the individual experiment, a goal field can be placed on an arbitrary field. The agent always starts on the field in the top

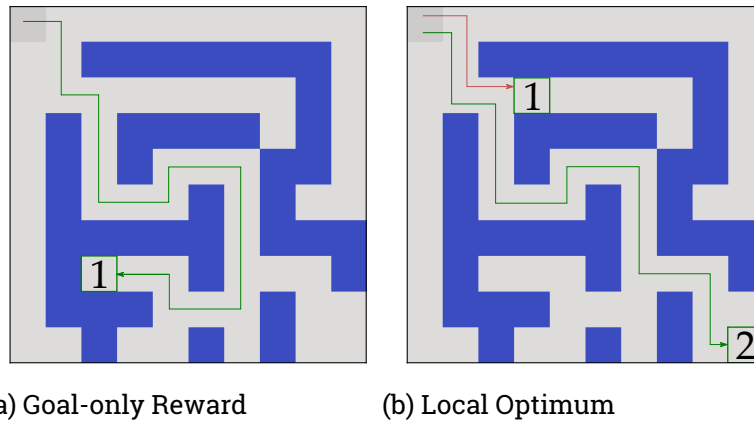


Figure 5.1.: Visualization of the environmental setups used for the experiments. The agent starts on the field in the upper left corner and can move one field per timestep. The goal fields are marked with a green border and they provide the reward denoted inside. Wall fields are represented by blue squares. Exemplary optimal paths from the start field to the optimal goal are denoted by a green line, whereas the optimal path to the suboptimal goal is represented by a red line in (b).

left corner. The field that is furthest away from the start field can be reached in 21 steps, and at least 97 steps are required to reach every field in a single episode. Overall, the agent can be located at 61 different fields, whereby the remaining 39 fields are occupied by walls. The main experiments are performed on the following environmental setups that are visualized in Figure 5.1.

**Goal-only Reward:** There is only a single reward that can be achieved by finding the way through the maze and reaching the goal field. The environmental setup is designed in a way that cannot be reliably solved by the plain PPO algorithm without any additional exploration strategy. A minimum of 21 steps is required to reach the goal.

**Local Optimum:** In this setup, two goal fields are positioned in the maze, one of which provides a reward of 1 and the other one a reward of 2, which is, at the same time, the optimal return achievable. As the suboptimal goal field is reachable from the start field in much fewer steps than the optimal goal (5 steps and 20 steps respectively), the goal providing the reward of 1 builds a local optimum.



---

### 5.3. Intrinsic Rewards only

---

With the observation of the training and especially the convergence behavior of the IM methods, we intend to get an impression of how the learning of the actual task of optimizing the policy according to the extrinsic rewards is influenced when combining intrinsic and extrinsic rewards. Most importantly, we want to observe whether the intrinsic rewards converge to 0 after sufficient exploration or continue to have high values, which might lead to a noisy greedy policy or even to non-convergence of the same. We, therefore, train a policy for each IM method with the IIM approach without providing any extrinsic reward, which causes the agent to learn its behavior only based on the intrinsic rewards produced by the respective IM. Additionally, with this experiment, we have the opportunity to see the ability of the individual IMs to explore the entire state space. As a comparison, we also observe the exploration behavior of a PPO agent that does not use any kind of additional exploration method.

These experiments are performed on the plain maze environment without a goal field in order to show the exploratory behavior of the different IM methods in an environment that is hard to explore.

---

### 5.4. Parameter Tuning

---

We run the experiments mentioned in Section 5.3 multiple times with different values for  $\eta$  to observe the impact of the parameter and explore its importance or unimportance for extensive exploration. Then, we use the insights to determine an appropriate  $\eta$  to use for the exploration policy in the case of EIM methods. The values we consider are 1, 0.1, and 0.01. Additionally, we try different sizes for the state representation learned by the inverse model of the ICM with the values of  $2^n$  with  $n \in \{2, 3, 4, 5, 6\}$ . Furthermore, we also run PPO without any IM on a simple form of the maze environment with a goal that is easy to reach in order to determine an appropriate amount of neurons we use for all hidden layers except for the encoding layer of the ICM. Hereby, we try layer sizes of  $2^n$  with  $n \in \{5, 6, 7, 8\}$ .

We have two parameters to tune that directly impact the exploration by scaling the impact of the IM on the overall behavior. The parameter  $\eta$  scales the intrinsic rewards produced by the ICM in order to keep them in a reasonable range relative to the extrinsic

---

rewards, whereas the parameter  $h$  determines the impact of the exploration policy  $\pi_e$  on the behavior when using EIM.

Regarding  $\eta$  when using IIM, we again try three different parameter values in both proposed environmental setups ("Goal-only Reward" and "Local Optimum"), namely 1, 0.1, and 0.01.

While  $\eta$  is a constant hyperparameter,  $h$  does not have the restriction of being fixed. However, even if we have the possibility to adjust its value, the first and most obvious approach is keeping the parameter constant during the entire process. Therefore, we execute the EIM approaches with an  $h$  of 0.25, 0.5, and 0.75.

Eventually, we want to use the full potential of the EIM approach by determining  $h$  dynamically. What we want is to keep exploring the environment, even if we suppose that the optimal solution has been found, which we can do because the greedy policy should not be directly affected by doing so and keep its greedy behavior. To exploit this advantage, we make  $h$  dependent on the entropy of  $\pi_g$  as well as on the entropy of  $\pi_e$ . The entropy of a policy given a state provides information about the uncertainty of its action choice. For a policy  $\pi$ , the entropy is defined as

$$H_\pi(s) = - \sum_{a \in A} \pi(a | s) \log \pi(a | s)$$

for a given state  $s$  [81]. With this definition, we can look up the certainty of  $\pi_g$  and  $\pi_e$  at each timestep. A high entropy of  $\pi_g$  in  $s$  indicates high uncertainty of which action to choose, that is, the probability of drawing each action is near  $1/|A|$ . Vice versa, a very low entropy means that the policy is very certain about its choice of the action. Interpreting such a low entropy as an indication that the policy has found a strategy for reaching a local optimum, we could use this value as our  $h$  and, therefore, reduce exploration at this point. In that case, the agent would, in theory, behave similarly to an IIM agent in terms of its tendency to converge to a local optimum. To avoid such a behavior, the entropy of  $\pi_e$  could be taken into account. With the entropy of the exploration policy indicating how certain the policy is about an action leading to a state worth exploring according to the IM method, we could create a trade-off between uncertainties. What we want is to follow the greedy policy in states, in which we are certain to know a good action choice, but explore if we know that there is something to explore starting from the particular state. We can do this by averaging the two entropies, resulting in the computation

$$h = \frac{1}{2}(1 - H'_{\pi_e}(s_t) + H'_{\pi_g}(s_t))$$

---

that is made for each timestep. In order to use  $h$  as a probability, the entropies have to lie between 0 and 1. Since the entropy does not fulfill this requirement, we normalize it through division by the maximal entropy value, which is described by

$$H'_\pi(s) = \frac{H_\pi(s)}{-\log \frac{1}{|A|}},$$

where  $-\log(1/|A|)$  is the maximal entropy for a discrete action space with  $|A|$  actions.

We compare the mentioned approaches of determining  $h$  in the "Local Optimum" environment.

---

## 5.5. Implicit vs Explicit Intrinsic Motivation

---

We want to see the impact of the IM methods performing in environments with extrinsic rewards, and, most of all, we aim to compare the performance and applicability of IIM and EIM methods. Therefore, we apply both approaches to the above-mentioned environments and make the comparison based on several properties. The experiments are designed to specifically show certain assumed advantages and shortcomings of both methods if these prove to be correct, or show their similarity otherwise.

The "Goal-only Reward" environmental setup should show the general ability of both approaches to solve classical sparse rewards environments and entail a comparison between their performances under these conditions. An extensive exploration of the environment is required to find the goal, which is why this experiment is intended to show the approaches' ability to make use of their IM in general. Moreover, we want to observe the noise in the greedy policy when the extrinsic reward was found, which is desired to be minimal but expected to be high when using the IIM approach with a high  $\eta$ .

Executing the IIM approach in the "Local Optimum" environment is expected to cause premature convergence in the case of a too-small chosen  $\eta$ . In general, the local optimum makes an extensive exploration and, therefore, the discovery of the optimal goal difficult. What we want to observe is the ability of both approaches, IIM and EIM, to keep exploring the environment after converging to an optimum. In order to do so, we look at the ability of the methods to find the optimal goal and adapt the policy for all states, especially those not lying on the optimal way. Additionally, we again have a look at the noise in the resulting greedy policies by observing the timesteps the agents need to reach the goal, which should be as constant as possible to indicate a low noise policy.

---

## 6. Results

---

This chapter provides the results of the experiments discussed in Chapter 5. After observing the findings regarding (hyper)parameters and the plain environment without extrinsic rewards, we take a detailed look at the performance of EIM methods in comparison to IIM methods in the "Goal-only Reward" and "Local Optimum" setting.

---

### 6.1. Parameter Tuning

---

In this section, we present our choice of parameters based on the results of the respective experiments. Hereby, the first part addresses the appropriate choice of the network sizes, in particular, the dimension of the state representation layer learned by the ICM and the general size of the fully connected layers that will be used for all other hidden layers. The second part provides insights into the effect of using different values for the parameter  $\eta$ . The different methods of determining  $h$  are evaluated in the last part of this section.

#### 6.1.1. Network Architectures

We experimentally determined appropriate network architectures by testing different numbers of neurons for the hidden layers. A visualization of the resulting architectures can be seen in Figure 6.1.

By observing the total state visitations after the learning process in the plain maze environment (without extrinsic rewards), we can find out to what extent the different sizes of the state encoding learned by the ICM lead to an area-covering exploration. In average, the 64-dimensional encoding resulted in the lowest standard deviation regarding the number of visitations of all normal fields, which tells us that the state space had been

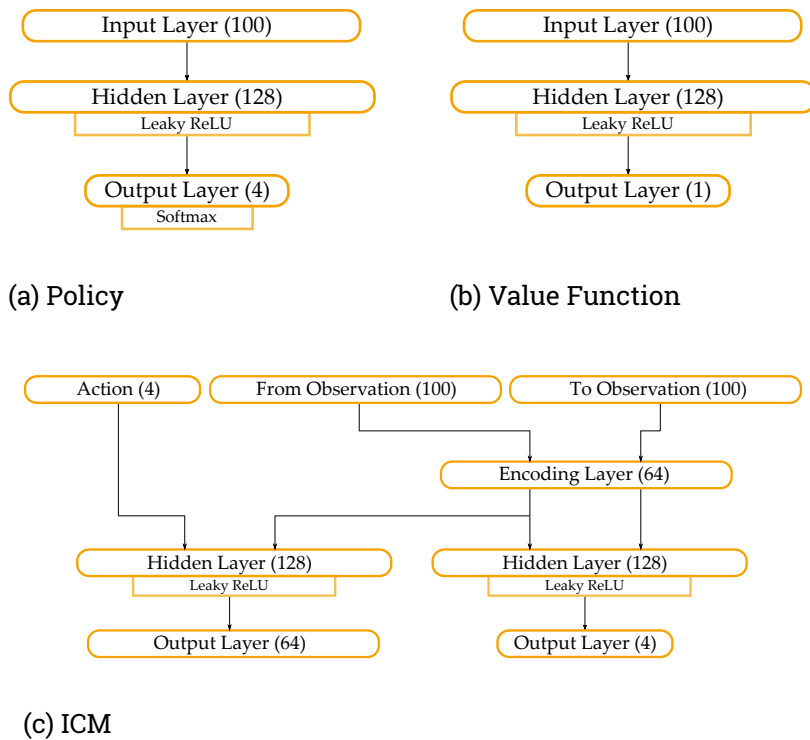


Figure 6.1.: Network architectures of the policies (a), the value functions (b), and the ICM (c). The numbers in parentheses represent the respective layer size, i.e., the number of neurons inside the layer.

explored most evenly in this case. Therefore, all further experiments are applied with an ICM state encoding size of 64.

The experiments on how many neurons to use for the other hidden layers showed a size of 256 to be superior to the others in terms of an earlier convergence to the optimal value. However, all tested settings achieved very similar results and the experiment was a very simple one. Nonetheless, each of them should serve the purpose of performing in a minimalistic grid environment such as the one we use, which is why we renounce further experiments on the layer sizes and instead work with the value of 128, which performed nearly as good as a size of 256, in order to keep the computational resources as low as possible.

---

	$\eta = 1.0$	$\eta = 0.1$	$\eta = 0.01$
ICM	4362.2220	4379.5591	4481.9204
Novelty	3144.7337	5468.5325	12634.0472

Table 6.1.: Standard deviation of the number of times all states were visited after 1000 iteration, averaged over five runs.

### 6.1.2. Parameters $\eta$ and $h$

When training an agent by optimizing the intrinsic returns only, the parameter  $\eta$  had seemingly no significant impact on the exploration behavior when using the ICM but made a big difference regarding the use of novelty. The total state visitations of each tested parameter value (1, 0.1, and 0.01) after 1000 iterations are depicted in Figure 6.2 in the form of a heat map. The observation of the standard deviation of the number of times each field was visited, which can be seen in Table 6.1, suggests an advantage of using an  $\eta$  of 1 for both types of IM, which is why we use it for the exploration policy  $\pi_e$  in all experiments. In general, the results suggest that  $\eta$  should not be chosen too small. However, the advantage of EIM is that we can hardly choose this parameter too high as this does not affect the agent in terms of not taking note of extrinsic rewards.

Regarding the parameter  $h$ , the approach of using a constant value showed to be worse than the use of an entropy-based  $h$  in most experiments and in terms of a (fast) convergence to the optimal goal in the "Local Optimum" environment (see Appendix A for more information). In order to keep the number of experiments in a reasonable range, all results described in Section 6.3 are based on the entropy-based setting. This approach performed well in most regards compared to the others and, additionally, is the most general approach in terms of an easy adoption to other experiments, free of hyperparameter tuning.

---

## 6.2. Exploration Behavior and Convergence of the IM Methods

---

By only reinforcing the behavior of the agent with the help of the IM, that is, with no extrinsic rewards provided, we are able to see how well the respective methods can help to discover all states of the environment.

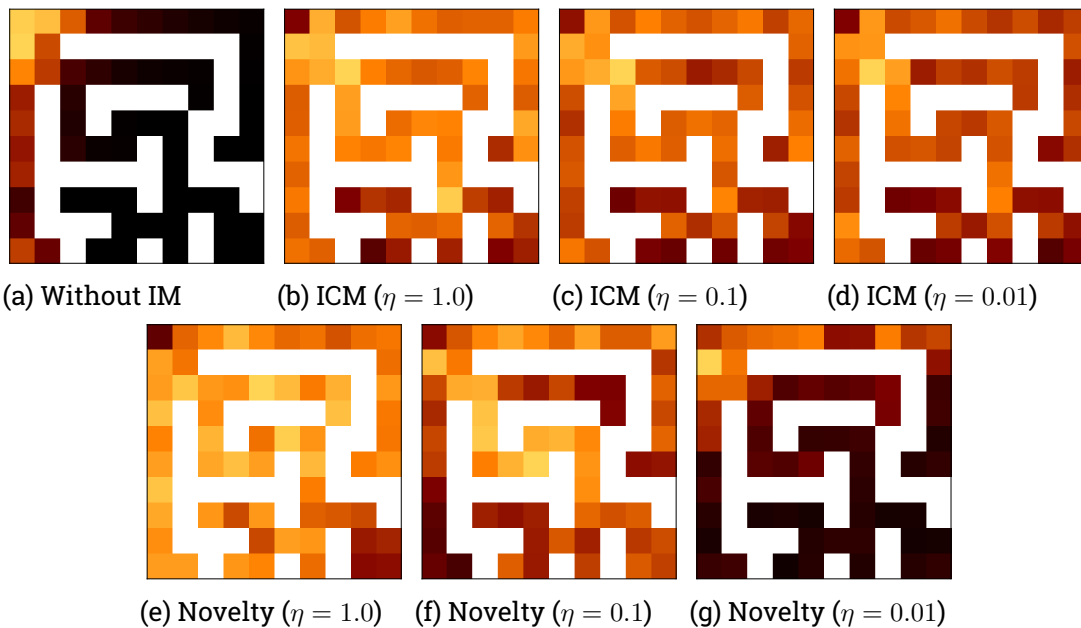
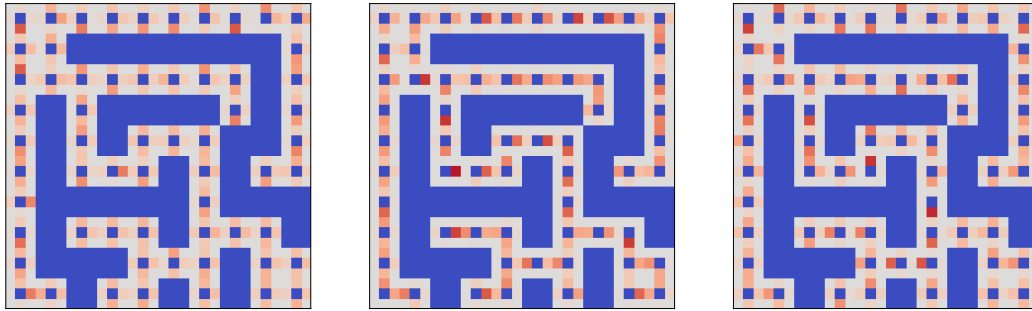


Figure 6.2.: Heat maps depicting the total state visitations of agents learned based on intrinsic rewards only (b - g) and without any rewards (a) after 1000 iterations. Hereby, (b)-(d) show the results when using the ICM with different values for  $\eta$  and (e)-(g) when using novelty. A brighter color indicates a higher number of visitations on the respective field. White squares represent wall fields.

In addition to showing the methods' ability to improve the exploration, we can also see the convergence behavior of the IMs. Additionally, the results of applying PPO without any exploration method is presented. It should be noted, however, that PPO in practice often gets a bonus reward based on the entropy, which is not done in our case.

### 6.2.1. Without Exploration Method

As expected, without using any kind of additional exploration method, the environment has been explored poorly, which gets visible when observing Figure 6.2a. The fields have been visited unevenly, and unexpectedly the visitations are not characterized by a gaussian distribution, which is to be expected when the actions are chosen completely randomly.



(a) Without IM

(b) ICM ( $\eta = 1.0$ )

(c) Novelty ( $\eta = 1.0$ )

Figure 6.3.: Visualization of the action probabilities for every state after 1000 iterations using intrinsic rewards only (b and c) and no rewards (a). The big blue squares are wall fields, whereas the small blue squares are the center of each normal field. In each direction the agent can move (up, down, left, right), the center has an additional square attached, which indicates the probability of choosing the according action, whereby a more intense red implies a higher probability.

Figure 6.3a shows the action probabilities of the policy for each field. The bias in the action probabilities could be explained by the randomly initialized value function falsely indicating a better outcome for some actions.

## 6.2.2. Novelty Behavior

The convergence to 0 of the intrinsic rewards produced by the state novelty approach gets rather obvious when regarding its calculation. Because of the denominator  $N(s_t)$ , which represents the number of times the agent visited state  $s_t$  and is monotonously increasing for every state, the fracture  $1/N(s)$ , which is the harmonic sequence for each state, goes towards 0 during the extensive exploration of the entire state space. The convergence of the reward is illustrated in Figure 6.4. However, considering an optimum that does not require the entire state space to be explored, the convergence might not be quick enough to ensure fast adaptation of the policy to the optimal behavior regarding the extrinsic task. In this case, a smaller  $\eta$  might be appropriate, which, however, presupposes knowledge of the extrinsic reward function. The method of count-based state novelty enables a largely evenly distributed exploration of all fields when choosing an  $\eta$  that is not too small, which



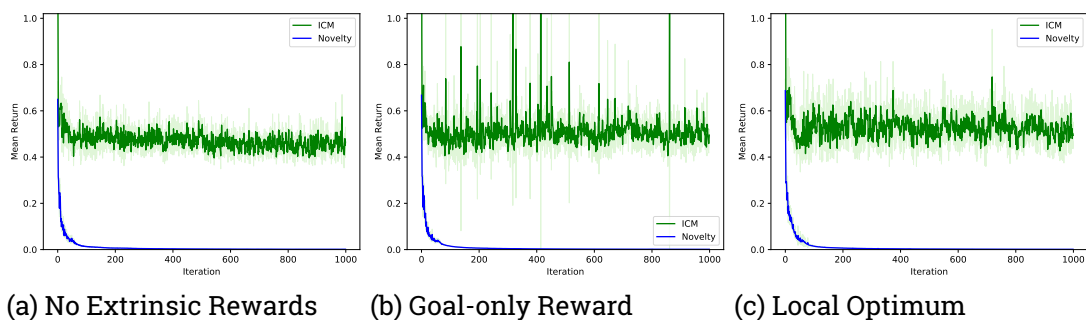


Figure 6.4.: Mean intrinsic training returns per iteration using the ICM (green line) and novelty (blue line) as IIM approaches with an  $\eta$  of 1, averaged over five runs.

can be seen in Figures 6.2e - 6.2g.

### 6.2.3. ICM Behavior

In contrast to the convergence behavior of state novelty, the one of the ICM is not as clear because of the produced rewards depending on the precision of predictions made by the forward model and implicitly the inverse model. Our experiments showed that the rewards did not converge at all within 1000 iterations (see Figure 6.4). This non-convergence might be explainable with overfitting on states recently visited, which could make the prediction in the remaining states worse again. Another explanation could be that the inverse model was not able to converge until the end of the experiment, causing the state representation to change during the entire process and, thus, entailing the necessity of adjusting the prediction of the state representation of the next state. The last possible cause of the non-convergence is a possibly too small NN architecture, resulting in the inability of the model to learn the optimal results.

However, it is interesting to see the policy still providing a reasonable exploration strategy after 1000 iterations (see Figure 6.3b). While the policy trained with novelty leads to areas that are hard to reach in some states, the action probabilities seem rather arbitrary in others (see Figure 6.3c). This outcome might be caused by the small intrinsic rewards provided by this method after some time of exploration, not providing useful information anymore. In contrast, the policy learned using the ICM confidently leads through the

---

maze with high probabilities to choose actions leading to the deepest areas that are the hardest to reach.

---

### **6.3. Implicit vs Explicit Intrinsic Motivation**

---

In order to evaluate the experimental results and make statements on the advantages and disadvantages of using EIM methods, we observe four properties, which are

1. sample and time efficiency,
2. premature convergence to a local optimum,
3. noise and stability of the greedy policy, and
4. ongoing exploration after convergence of the greedy policy.

Each of the said aspects is regarded separately from the others in order to make suggestions about appropriate fields of applications based on particular properties possible.

#### **6.3.1. Sample and Time Efficiency**

In this section, we have a look at the time and sample efficiency of the individual approaches, that is, how long the trajectory sampling and the training takes and how many samples are needed to find a near-optimal solution.

##### **Time Efficiency**

In general, the IIM approach took less time per iteration compared to the EIM approach, which is due to its less complex architecture. The use of an additional exploration policy and, in the case of the ICM, the forward and inverse model entail computationally more expensive and, therefore, more time-consuming update steps. When using the entropy of the greedy and the exploration policy during the trajectory sampling, both policies have to make a prediction in order to compute the respective entropy every step, which additionally slows down the overall learning.

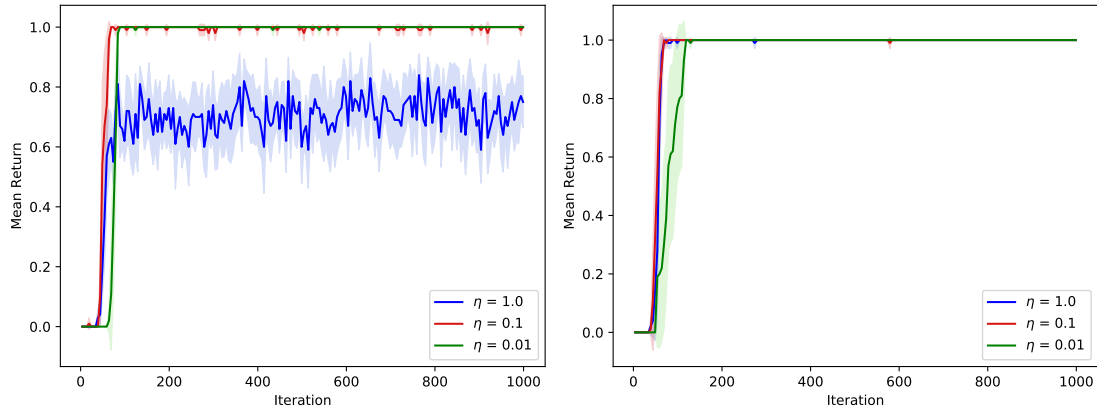
---

## Sample Efficiency

The sample efficiency, on the other hand, depends on the problem that is to be solved. The "Goal-only Reward" experiment showed the IIM methods to be more sample efficient (see Figures 6.5c and 6.5d), which we explain considering two circumstances. The first one is related to the importance weight of the off-policy approach, which makes the update steps of the greedy policy smaller if it would have been less likely that the same would have stepped into the rewarding field than it was following the behavior policy. A similar problem occurs when using the on-policy approach, namely the possibility of very rarely moving into the optimal goal when following the greedy policy and, therefore, rarely receiving the accompanying reward. The second circumstance results from the possibility of the agent performing actions according to the greedy policy instead of the exploration policy, which diminishes the effect of the IM and, thus, leads to a worse exploration and a later discovering of the reward.

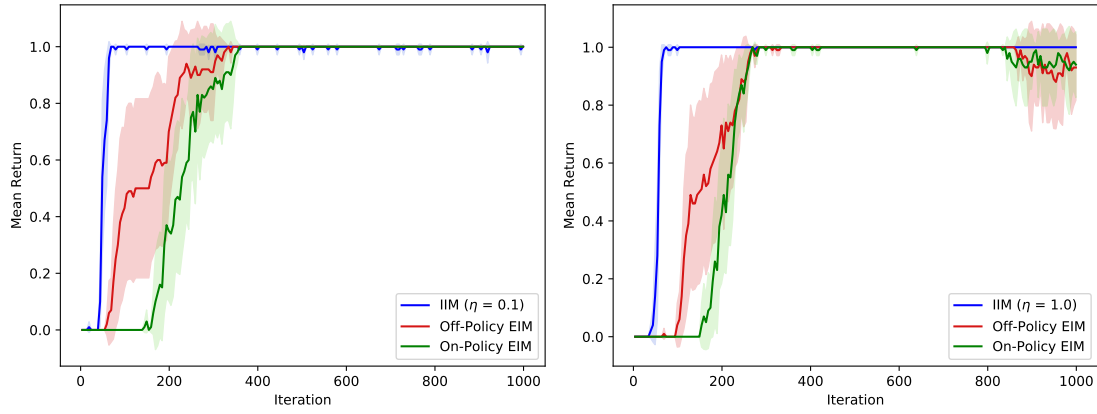
When observing the results regarding the "Local Optimum" environmental setup using IIM methods, we see that the use of different settings of  $\eta$  influences the number of samples needed to find the optimal goal. This result was to be expected since the two types of rewards, intrinsic and extrinsic, influence each other, which leads to the need for a trade-off. As can be seen in Figures 6.6a and 6.6b, the number of samples needed to find the optimal goal and adjust the policy accordingly highly depends on the used  $\eta$ . The higher the chosen  $\eta$ , the quicker the agent finds the optimal goal, which can be explained by the fact that the distraction from the exploration, caused by the extrinsic reward leading to the suboptimal goal, gets smaller. On the other hand, however, a higher  $\eta$  vice versa comes with a higher distraction from the extrinsic task (see Section 6.3.2). In the case of the "Goal-only Reward" environment, on the other hand, the choice of  $\eta$  makes nearly no difference regarding the number of samples needed to find the optimal goal, which can be seen in Figures 6.5a and 6.5b, and which is caused by the fact that there is only one extrinsic reward. Therefore, no further exploration is needed after convergence to the corresponding goal field; i.e., it does not matter that the intrinsic reward is much smaller than the extrinsic one.

Regarding the use of EIM methods, it becomes obvious that local optima pose a problem when looking at the results. The development of the returns achieved during the periodic evaluations throughout the learning process, which is depicted in Figures 6.6c and 6.6d, shows that both the on-policy and off-policy EIM approach sometimes manage to learn a near-optimal solution and sometimes they only converge to the suboptimal solution (see Section 6.3.3 for more information on premature convergence).



(a) ICM with different values for  $\eta$

(b) Novelty with different values for  $\eta$



(c) Approaches using ICM

(d) Approaches using Novelty

Figure 6.5.: Mean extrinsic evaluation returns per iteration in the "Goal Only Reward" setup averaged over ten runs. A comparison between the use of different values for  $\eta$  in the case of IIM can be seen in (a) and (b). The results of the different approaches using the ICM and novelty are depicted in (c) and (d) respectively.

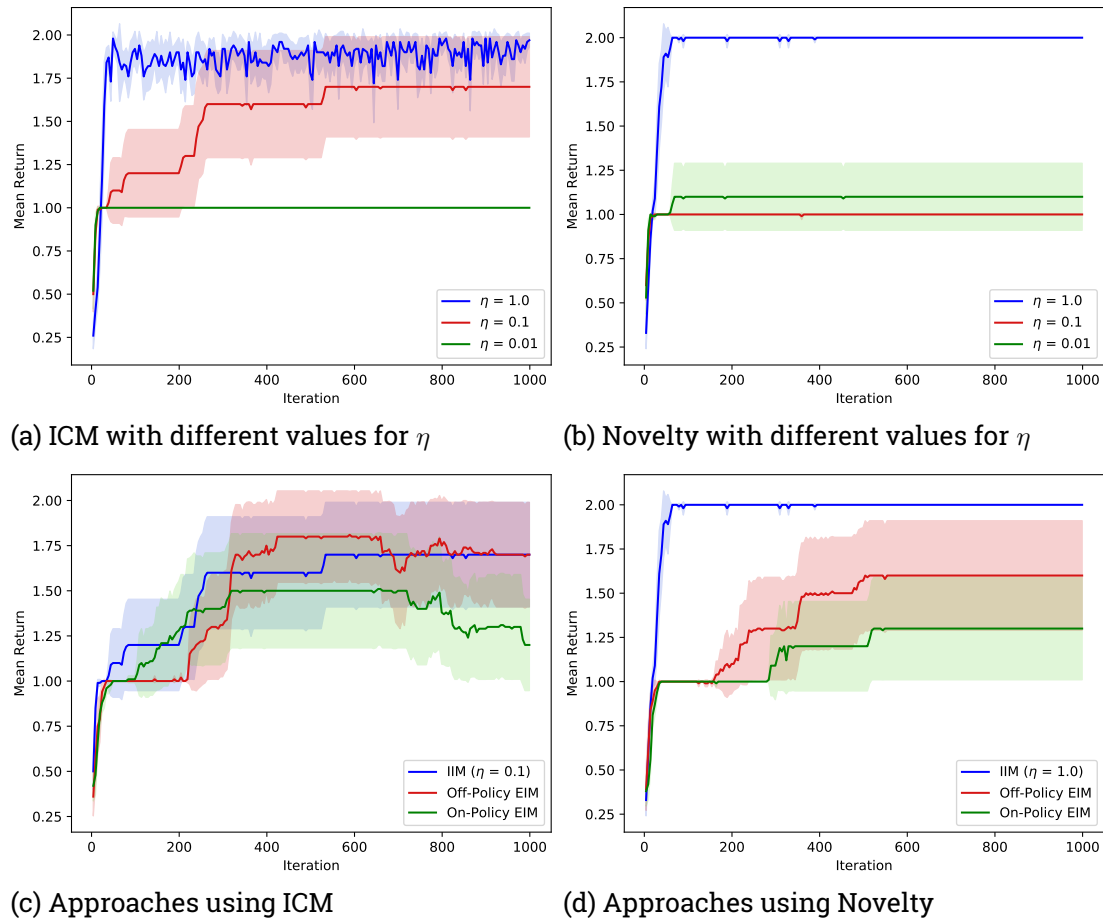


Figure 6.6.: Mean extrinsic evaluation returns per iteration in the "Local Optimum" setup averaged over ten runs. A comparison between the use of different values for  $\eta$  in the case of IIM can be seen in (a) and (b). The results of the different approaches using the ICM and novelty are depicted in (c) and (d) respectively.

---

In the cases in which the agent managed to solve the task near-optimally, the EIM approach needed a similar number of samples as the IIM approach when using the ICM. When using novelty, the IIM approach was visibly more sample efficient than the EIM approach.

Overall, the IIM approaches appear to be superior in terms of a quick and sample efficient convergence, assuming an appropriate choice of  $\eta$ .

### 6.3.2. Noise and Stability of the Greedy Policy

Now we have a look at the results of single representative executions of experiments, in which both approaches, IIM and EIM, managed to solve the task, i.e., found a near-optimal policy, and compare the stability and noise in the resulting greedy policy of the respective methods. Hereby, stability refers to the ability to keep the policy to be optimal throughout the ongoing learning process after it has already converged. The noise of a policy describes how reliably it chooses the optimal actions. Hence, more noise can lead to a worse average performance of the greedy policy since it performs some suboptimal actions from time to time. We can measure the noise by observing the number of timesteps needed until termination per episode during the evaluation. Hereby, a constant number of 21 timesteps in the "Goal-only Reward" setup and 20 in the "Local Optimum" environment is desired. A less constant number of timesteps implies a noisier policy.

Before directly comparing the IIM and EIM approaches, we look at the impact of  $\eta$  in this regard when using IIM. While Figures 6.5a and 6.6a show the high impact of the parameter on the stability of the policy when using the ICM, Figure 6.7a suggests the same in terms of noise. Regarding the impact of  $\eta$  when using novelty, the stability is only minimally affected (see Figures 6.5b and 6.6b), whereas the noise differs visibly when using different values for  $\eta$  (see Figure 6.7b). Generally said, a higher  $\eta$  provides a better exploration while, simultaneously, leads to higher noise and instability. Since an  $\eta$  of 0.1 when using the ICM and 1 using novelty provided a good trade-off between noise and performance, we compare these settings to the EIM approaches.

The results make clear that it is necessary to distinguish between intrinsic rewards that converge to 0 within a reasonable time (novelty) and those not converging (ICM).

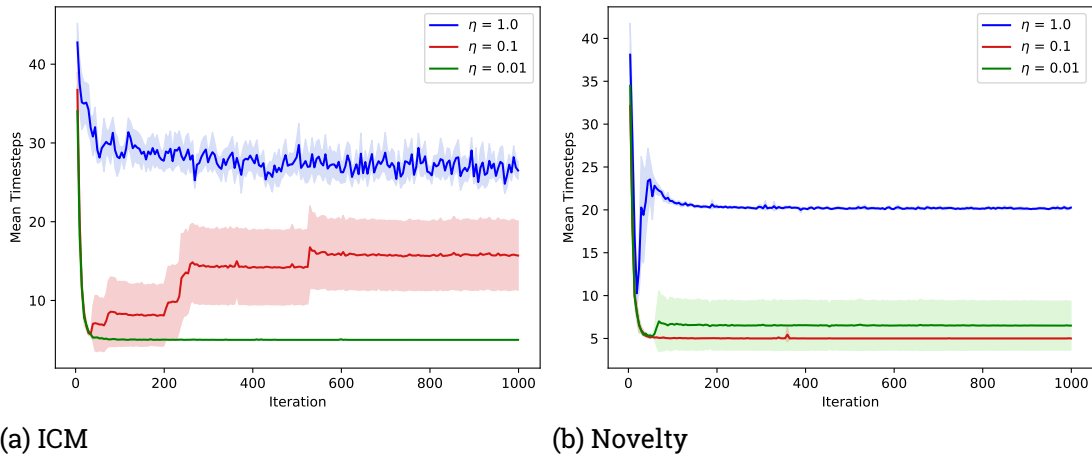


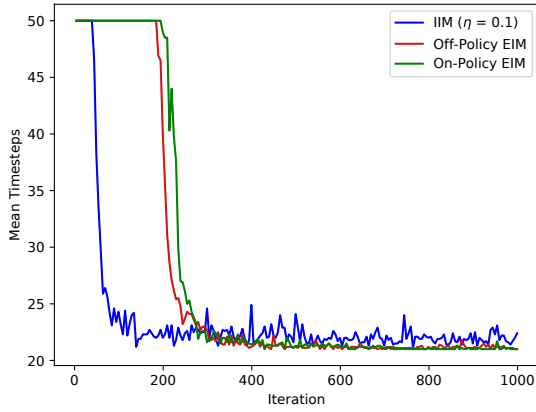
Figure 6.7.: Mean timesteps until termination during evaluation in the "Local Optimum" setup using different values for  $\eta$  in the case of IIM, averaged over ten runs.

### Noise and Stability - Novelty

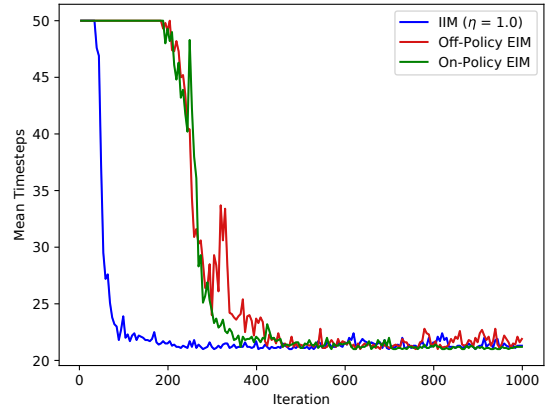
When observing Figures 6.8b and 6.9b, we discover that all used methods resulted in similarly noisy greedy policies. The only conspicuous thing to mention is that the off-policy approach took longer to provide a policy with relatively low noise. A similar conclusion can be made regarding the stability in the "Local Optimum" environment, which is nearly perfectly provided by the IIM and on-policy EIM approach. However, both EIM methods showed instabilities at the end of the training in the "Goal-only Reward" environment, whereas the policy was kept very stable a long time before this stability collapse. As the intrinsic rewards converge to 0, the IIM approach does not suffer overly from the distraction of these reward signals.

### Noise and Stability - ICM

In contrast to the use of novelty, Figures 6.8a and 6.9a show that using the ICM causes a steadily continuing distraction from the main task, which is reflected by a greedy policy that is more noisy using IIM than EIM in many iterations.

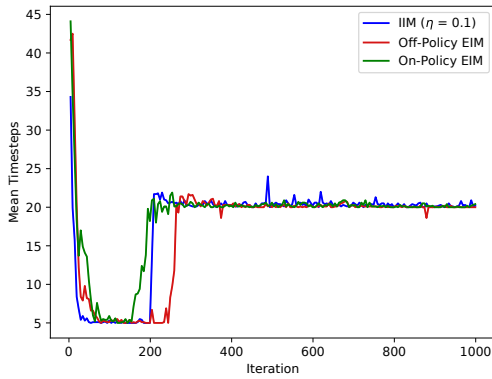


(a) Approaches using ICM

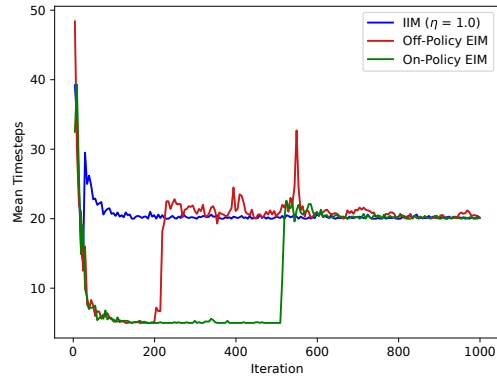


(b) Approaches using Novelty

Figure 6.8.: Mean timesteps until termination during evaluation in the "Goal-only Reward" setup.



(a) Approaches using ICM



(b) Approaches using Novelty

Figure 6.9.: Mean timesteps until termination during evaluation in the "Local Optimum" setup.



---

Especially regarding the "Goal Only Reward" setup, a high level of noise can be observed, which might be due to the fact that the agent has a lot more timesteps available than it needs to reach the goal, which it possibly uses to explore the environment, resulting in sometimes more steps being taken. The resulting policies show similar stability to the ones using novelty but, contrarily, with continuously stable policies in the "Goal-only Reward" setup and collapsing stability in the "Local Optimum" setup when using EIM (see Figures 6.5c and 6.6c).

### **Noise and Stability - Conclusion**

Considering noise, the results of the experiments reveal what we expected and what we wanted to show. When using IIM, non-converging intrinsic rewards keep disturbing the policy when the goal reward has already been found. This impact of the intrinsic rewards, however, is necessary to ensure further exploration after extrinsic rewards have been found, since the reward function is not known and, therefore, there might be more rewards to be found. In order to minimize noise, it is possible to lower the factor  $\eta$  in order to get smaller intrinsic rewards that have less impact on the update steps relative to the extrinsic rewards. However, using EIM methods, this is not necessary as the intrinsic rewards do not directly influence the greedy policy while simultaneously providing extensive exploration throughout all iterations (see Section 6.3.4).

Stability, on the other hand, can not necessarily be assured throughout the entire learning process when using EIM but was shown to be present most of the time. The instability, however, could be caused by the particular methods we used for the EIM approach and, hence, could possibly be waived when using different methods, e.g., an unbiased off-policy algorithm.

### **6.3.3. Premature Convergence**

Our experiments in the "Local Optimum" environment were designed to make an observation of the property of premature convergence to this same local optimum possible. Again, we regard the IM methods individually.

---

## Premature Convergence - Novelty

Regarding the use as an IIM approach, converging intrinsic rewards, such as those generated by the state novelty approach, regulate their impact on the overall reward by themselves by diminishing over time. Even when the initial intrinsic rewards are very high, the eventual diminishing of the same ensures convergence to an optimum according to the extrinsic rewards with minimal noise. By providing high rewards in the beginning, a well covering of the observation space can take place. In our example, the state novelty IM produces a reward of 1 for each state the agent has never visited before, which is the same value as the extrinsic reward the agent receives for entering the local optimum. However, since the agent does not optimize the immediate reward but the discounted return, visiting new states is much more attractive in the early states of the learning process. Nonetheless, without having knowledge about the reward function, it could be difficult to scale the intrinsic rewards appropriately. Intrinsic rewards that are too low compared to the extrinsic rewards can lead to premature convergence to a local optimum, as can be seen in Figure 6.6b.

Applying novelty in an EIM manner did not lead to an improvement in terms of a faster convergence; on the contrary, in all cases, it took longer to learn an optimal policy or the optimal solution could not be learned at all (see Figure 6.6d). That shows that the EIM method is more prone to premature convergence than the IIM approach with novelty when using an appropriate  $\eta$ .

## Premature Convergence - ICM

In contrast to the state novelty approach, the ICM quickly learned to approximately predict the next state, leading to low intrinsic rewards, which, on the one hand, evolve over time and, thus, lead to a well-exploring behavior before discovering any extrinsic reward but, on the other hand, come with a more difficult and influencing choice of the reward scaling, i.e., the parameter  $\eta$  when using IIM. As the intrinsic returns remain very similar over time, we have to choose  $\eta$  so that the extrinsic rewards are always much higher than the intrinsic rewards. Doing so, however, leads to a poor exploration due to premature convergence once an extrinsic reward has been found since the intrinsic reward has no big impact from this point on. Choosing a higher value for  $\eta$  prevented premature convergence but led to a noisy and unstable greedy policy due to the consistent high impact of the intrinsic rewards (see Figures 6.6a and 6.7a).

---

In the case of using the ICM, the advantage of the EIM over the IIM approach depends on the choice of  $\eta$  of the IIM. Setting the parameter to a value of 0.01 did not allow the algorithm to find the optimal solution of the "Local Optimum" environment within 1000 iterations, which is shown in Figure 6.6a. With an  $\eta$  of 0.1, the agent was able to find the optimal goal in some cases. However, a higher  $\eta$  affected the noisiness and stability of the resulting policy, which is discussed in Section 6.3.2. Using the EIM instead, this parameter is less significant, as the impact of the exploration policy is largely independent of the scaling of the reward.

A comparison between the learning process of the IIM approach and the EIM approach is depicted in Figure 6.6c. Here we can see that the EIM approaches, as well as the IIM approach, do not guarantee convergence to the optimal solution and that they performed very similarly in this regard. However, this is only a comparison using the IIM methods with an  $\eta$  of 0.1, which seemed to be the most reasonable value to use among the ones tested. As mentioned before, a higher  $\eta$  causes oscillation around the optimum but might lead to a more reliable discovering of the global optimum, whereas a lower  $\eta$  might not find it at all.

### **Premature Convergence - Conclusion**

The results suggest that IIM and EIM both suffer from premature convergence, assuming an  $\eta$  that is low enough in the case of IIM, but the reasons are different in both cases.

While the IIM approach suffers from a too low impact of the intrinsic reward relative to the extrinsic one, the EIM approach keeps exploring but is not able to adjust the already converged action probabilities, which gets clear when observing the action probabilities depicted in Figures 6.10b, 6.10c, and 6.10f. While the policy reliably directs to the global optimum if the agent manages to get a few steps beyond the suboptimal goal, the high probabilities provided for the states around the local optimum leading to the same make it very unlikely to step past said states using the greedy policy only. By additionally using the exploration policy during the training, the agent often manages to get through the fields in which the greedy policy points to the local optimum and, thus, reaches the optimal goal. The chance that the action choice of the greedy policy in the relevant states gets adjusted to lead to the global optimum is very unlikely in both cases, the on-policy and off-policy approach. The importance weight makes the update diminishing small in the off-policy case, and the correct action is unlikely to be chosen by the greedy policy and, therefore, rarely updated in the on-policy case.

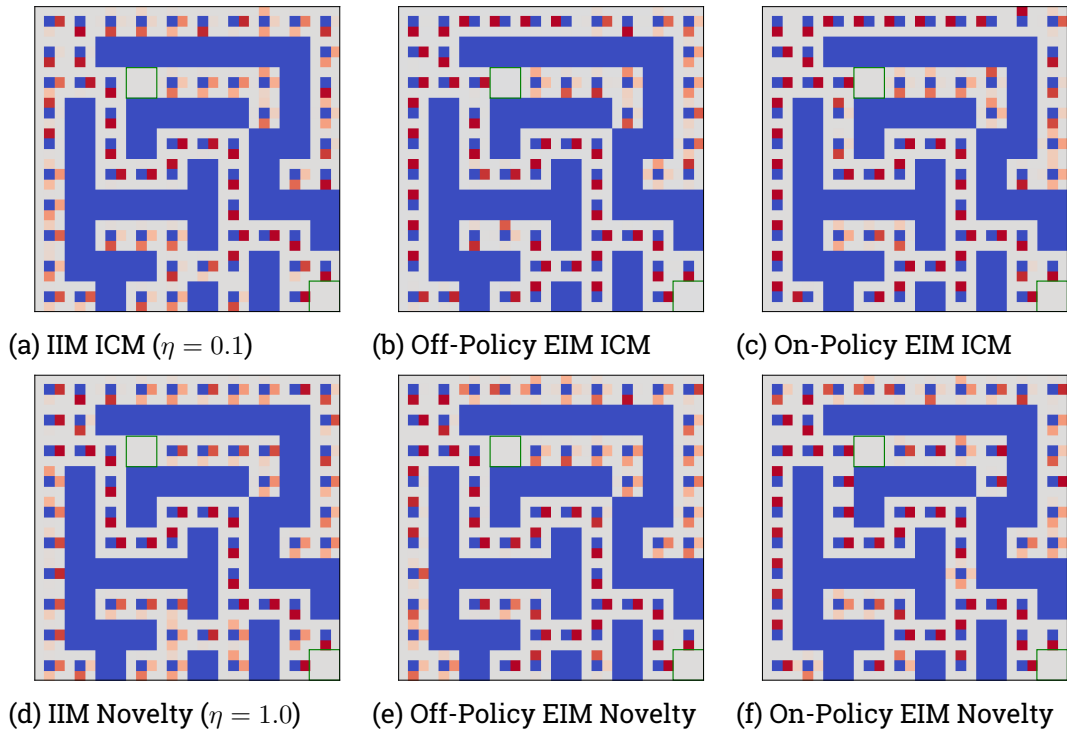


Figure 6.10.: Visualization of the action probabilities of the greedy policy for every state after 1000 iterations in the "Local Optimum" setting. The big blue squares are wall fields, whereas the small blue squares are the center of each normal field. In each direction the agent can move (up, down, left, right), the center square has an additional square attached, which indicates the probability of choosing the according action, whereby a more intense red implies a higher probability. The green bordered fields mark the goal fields with the one in the bottom right corner being the optimal goal providing a reward of 2, and the other one providing a reward of 1.

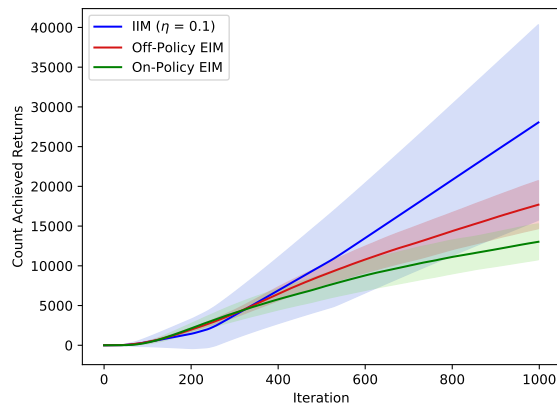



Figure 6.11.: Overall count of collected optimal returns (2.0) up to the respective iteration during training using the ICM averaged over 10 runs.

The overall problem of EIM methods of adjusting an already converged behavior is also reflected by the fact that, compared to IIM approaches, they were often able to find the global optimum with fewer samples but, nonetheless, did not perform better in terms of an earlier convergence to this optimum. The number of optimal returns (2.0) collected over time can be seen in Figure 6.11 using the example of the ICM.

### 6.3.4. Ongoing Exploration

An ongoing exploration after an extrinsic reward has been discovered could serve two goals: the discovering of more extrinsic rewards and the adjustment of the policy for all states in terms of a target-oriented behavior. Figure 6.10 depicts the action probabilities of the different approaches after 1000 iterations for all fields of the "Local Optimum" environmental setup. We can see that the EIM approaches provided wide coverage of converged action probabilities leading the agent to at least a local optimum for most of the fields. Especially the EIM approaches using the ICM provided an extensive exploration of most parts of the state space, which might be due to their non-converging and, therefore, ongoing availability of meaningful intrinsic rewards.

Regarding the IIM methods, on the other hand, we find that the policy did not converge in states that are not on the way to the goals or even confidently points in the wrong direction on many fields. These unadjusted action probabilities indicate poor exploratory



---

behavior after the discovering of the goals, leading to not visiting the according states anymore and, thus, not updating the policy with respect to them. Further indications for the more extensive exploration of EIM methods can be seen in Appendix B.

---

## 7. Conclusion and Future Work

---

We introduced two algorithms, one of them being an off-policy approach and the other one learning on-policy, which learn two separate policies that are optimized independently following different tasks, the one induced by the reward function of the environment and the one of exploring the environment. Therefore, we equipped a PPO agent with an exploration policy that was learned using intrinsic rewards produced by IMs in addition to the greedy policy following the extrinsic rewards. While our experiments on a simple maze grid world showed the conventional IIM approach using a single policy that optimizes the combined intrinsic and extrinsic return to be more sample and time efficient under most of the tested conditions, the newly introduced EIM approach turned out to be competitive or even superior under certain circumstances and regarding particular properties. Considering the requirement of tuning the hyperparameter  $\eta$ , which determines the impact of the intrinsic rewards, the approach of EIM has been shown to find solutions to the extrinsic task more efficiently if  $\eta$  is not well chosen for the IIM. It became clear that the EIM approach does not rely on an extensive search for an optimal  $\eta$ .

The results regarding the three main issues that we aimed to address are the following:

1. **Control over the level of exploration:** By treating the greedy policy individually from the exploration policy, we were allowed to control the level of IM-based exploration throughout the training. We, therefore, dynamically set the parameter  $h$ , which represents the probability of choosing an action according to the exploration policy, by means of the entropy of both policies. This method appeared to be superior to using a fixed value for  $h$ , and it enabled the possibility of keeping the exploration high even when the extrinsic task already seemed to be solved. When the evaluation took place, the parameter could be set to 0 manually in order to evaluate the performance of the greedy policy on its own.
2. **Premature convergence:** We showed that in the case of IIM methods, intrinsic rewards that are very small relative to the extrinsic rewards can lead to premature convergence to a suboptimal solution. Compared to such a setting, the EIM

---

approaches were more likely to find a near-optimal solution. Nonetheless, they appeared to be prone to premature convergence as well, performing worse than the IIM approach when the intrinsic rewards were scaled reasonably. This problem, however, was restricted to states that were near to the path leading to the suboptimal goal, which got apparent when regarding the behavior of the policy in other states. A wide covering of near-optimally converged action probabilities had been obtained for most of the states. The IIM methods, on the other hand, showed a poor behavior for many states that were not the ones leading from the initial state to the goal.

3. **Noisy policy:** Very high intrinsic rewards were shown to lead to a noisy and unstable policy when using them in an IIM approach. When using an appropriate  $\eta$  for the IIM methods, both EIM approaches, off-policy and on-policy, still resulted in a policy with a similar or lower level of noise in most of the experiments. Regarding the stability, the EIM methods suffered few collapses but were similarly or more stable than the IIM approach most of the time.

There are several scenarios in which we suggest considering the use of the EIM approach. These scenarios include one or more of the following assumptions:

- the absence of knowledge about the reward function since the intrinsic reward does not have to be scaled considering the extrinsic rewards,
- an extensive exploration of the state space entailing a policy that leads to a (local) optimum for most of the states, e.g., a robot that has to be trained always starting from a certain state, which might change in the practical application later on, or
- the need for a greedy policy with low noise and high stability without a time-consuming hyperparameter search.

If the effort of the search for an appropriate  $\eta$  can be applied, the use of the IIM approach might be especially beneficial when fast convergence is desired.

It is important to mention that the use of a converging IM method minimizes some of the problems of IIM methods. The noise in the policy gets very low with the intrinsic rewards converging to 0. Moreover, a higher  $\eta$  for scaling the intrinsic rewards produced by the novelty approach might still lead to a relatively low-noise policy while providing extensive exploration, which would make the hyperparameter search similarly easy as it is the case when using EIM. However, we applied the experiments on a small discrete environment, which is why we could easily use this type of count-based state novelty. This condition is not always given in practical applications and the results might look different under different circumstances.



---

## 7.1. Future Work

---

There is a lot of imaginable future work to be done. In addition to more extensive experimentation under similar conditions as the ones given in this thesis that were sparse in terms of a low execution number of the individual experiments and the extent of the hyperparameter tuning due to restrictions of computational resources, comprehensive research in the following areas might be of interest.

**Complex Environment** As we used a relatively simple environment for our experiments, the performance of EIM methods in more complex environments could be of interest. A higher complexity could refer to high-dimensional observations such as the pixels of the images of a game [67, 82] as well as more complex tasks requiring a higher level of planning [68, 67].

**EIM Algorithms** Algorithms other than PPO could be adopted to serve as EIM algorithms. More developed off-policy approaches could be investigated in this context, such as the state distribution correction used by the OPPOSD algorithm [21].

**Pre-train Exploration Policy** Another investigation of interest would be to pre-train the exploration policy, beginning to train the greedy policy when the exploration policy has developed a well-exploring behavior, similar to the experiments conducted by Pathak et al. [10]. This approach could help to solve the problem of EIM methods prematurely converging to a local optimum. A higher reward that is further away might be found more quickly due to the already exploratory behavior of the agent. Thus, the agent might not converge to the local optimum as easily as it recognizes the global optimum earlier.

**Parameter  $h$**  Regarding the determination of the parameter  $h$ , the effect of different methods could be explored. For example, starting with a value of 1 and decreasing it over time could improve the results in very sparse rewards environments as the greedy policy, which is rather meaningless at the beginning of the training, would have no impact at first, leaving the initial exploration exclusively to the exploration policy. Another approach tackling the same situation would be to keep  $h$  constantly at 1 until an extrinsic reward has been found.

---

**Different parameters for different policies** An advantage of EIM methods we did not exploit is the possibility of choosing different hyperparameters regarding the two policies. For example, the parameter  $\lambda$  used to calculate the advantage of a state-action tuple could be varied for the greedy policy, introducing bias but making use of actor-critic benefits such as a possibly faster convergence. Furthermore, different learning rates for the policies could be tested, e.g., equipping the exploration policy with a higher learning rate in order to speed up the exploration without making the training of the greedy policy unstable.

---

## Bibliography

---

- [1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.
- [2] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *CoRR*, vol. abs/1409.3215, 2014.
- [3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, 2015.
- [4] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, “Learning to navigate in complex environments,” *arXiv preprint arXiv:1611.03673*, 2016.
- [5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, 2016.
- [6] J. Peters and S. Schaal, “Policy gradient methods for robotics,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.
- [7] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, 2013.
- [8] Y. Duan, X. Chen, R. Houthoofd, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control,” in *International Conference on Machine Learning*, 2016.

- 
- 
- [9] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *The Journal of Machine Learning Research*, 2016.
- [10] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017.
- [11] B. C. Stadie, S. Levine, and P. Abbeel, “Incentivizing exploration in reinforcement learning with deep predictive models,” *arXiv preprint arXiv:1507.00814*, 2015.
- [12] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, “Large-scale study of curiosity-driven learning,” *arXiv preprint arXiv:1808.04355*, 2018.
- [13] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, “Unifying count-based exploration and intrinsic motivation,” in *Advances in Neural Information Processing Systems*, 2016.
- [14] M. C. Machado, M. G. Bellemare, and M. Bowling, “Count-based exploration with the successor representation,” *arXiv preprint arXiv:1807.11622*, 2018.
- [15] H. Tang, R. Houthoofd, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, “# exploration: A study of count-based exploration for deep reinforcement learning,” in *Advances in neural information processing systems*, 2017.
- [16] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, “Count-based exploration with neural density models,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.
- [17] N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, and S. Gelly, “Episodic curiosity through reachability,” *arXiv preprint arXiv:1810.02274*, 2018.
- [18] T. Degris, M. White, and R. S. Sutton, “Off-policy actor-critic,” *arXiv preprint arXiv:1205.4839*, 2012.
- [19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [20] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, “Sample efficient actor-critic with experience replay,” *arXiv preprint arXiv:1611.01224*, 2016.

- 
- 
- [21] Y. Liu, A. Swaminathan, A. Agarwal, and E. Brunskill, “Off-policy policy gradient with state distribution correction,” *arXiv preprint arXiv:1904.08473*, 2019.
- [22] I. Szita and A. Lőrincz, “The many faces of optimism: a unifying approach,” in *Proceedings of the 25th international conference on Machine learning*, 2008.
- [23] P. Morere and F. Ramos, “Bayesian rl for goal-only rewards,” in *Conference on Robot Learning*, 2018.
- [24] S. Parisi, D. Tateo, M. Hensel, C. D’Eramo, J. Peters, and J. Pajarinen, “Long-term visitation value for deep exploration in sparse reward reinforcement learning,” *arXiv preprint arXiv:2001.00119*, 2020.
- [25] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” *arXiv preprint arXiv:1810.12894*, 2018.
- [26] Y. Kim, W. Nam, H. Kim, J.-H. Kim, and G. Kim, “Curiosity-bottleneck: Exploration by distilling task-specific novelty,” in *International Conference on Machine Learning*, 2019.
- [27] O. Simeone, “A very brief introduction to machine learning with applications to communication systems,” *IEEE Transactions on Cognitive Communications and Networking*, 2018.
- [28] E. Alpaydin, *Introduction to machine learning*. MIT press, 2009.
- [29] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, “The expressive power of neural networks: A view from the width,” in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017.
- [30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [31] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” *arXiv preprint arXiv:1701.06538*, 2017.
- [32] A. J. Simpson, G. Roma, and M. D. Plumbley, “Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network,” in *International Conference on Latent Variable Analysis and Signal Separation*, 2015.
- [33] R. E. Bellman, *Adaptive control processes: a guided tour*. Princeton university press, 2015.
- [34] R. Xu and D. Wunsch, *Clustering*. John Wiley & Sons, 2008.

- 
- [35] J. Bell, *Machine learning: hands-on for developers and technical professionals*. John Wiley & Sons, 2014.
- [36] K. D. Bailey, *Typologies and taxonomies: an introduction to classification techniques*. Sage, 1994.
- [37] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, “Artificial neural networks,” *Cham: Springer International Publishing*, 2017.
- [38] C. C. Aggarwal, “Neural networks and deep learning,” *Cham: Springer International Publishing*, 2018.
- [39] M. W. Gardner and S. Dorling, “Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences,” *Atmospheric environment*, 1998.
- [40] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological review*, 1958.
- [41] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, 1989.
- [42] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural networks*, 1991.
- [43] Y. Bengio, “Learning deep architectures for ai,” *Foundations and trends® in Machine Learning*, 2009.
- [44] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, 1997.
- [45] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, 1998.
- [46] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological cybernetics*, 1980.
- [47] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, 1990.
- [48] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

- 
- [49] G. Montavon, G. Orr, and K.-R. Müller, *Neural networks: tricks of the trade*. springer, 2012.
- [50] A. R. Cassandra, “A survey of pomdp applications,” in *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, 1998.
- [51] R. D. Smallwood and E. J. Sondik, “The optimal control of partially observable markov processes over a finite horizon,” *Operations research*, 1973.
- [52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [53] R. S. Sutton and A. Barto, “Reinforcement learning : an introduction,” 2018.
- [54] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, 1992.
- [55] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in neural information processing systems*, 2000.
- [56] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, 1992.
- [57] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine learning*, 1988.
- [58] M. Fairbank and E. Alonso, “The divergence of reinforcement learning algorithms with value-iteration and function approximation,” in *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 2012.
- [59] J. Peters and J. A. Bagnell, “Policy gradient methods,” *Encyclopedia of Machine Learning*, 2010.
- [60] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, 2016.
- [61] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, 2015.
- [62] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in *Advances in neural information processing systems*, 2000.

- 
- [63] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2012.
- [64] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, England, 1994.
- [65] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016.
- [66] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [67] A. Juliani, A. Khalifa, V.-P. Berges, J. Harper, H. Henry, A. Crespi, J. Togelius, and D. Lange, "Obstacle tower: A generalization challenge in vision, control, and planning," *arXiv preprint arXiv:1902.01378*, 2019.
- [68] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, 2013.
- [69] S. M. Kakade, "A natural policy gradient," in *Advances in neural information processing systems*, 2002.
- [70] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, 2000.
- [71] P.-Y. Oudeyer and F. Kaplan, "How can we define intrinsic motivation," in *Proc. of the 8th Conf. on Epigenetic Robotics*, 2008.
- [72] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2005.
- [73] A. Aubret, L. Matignon, and S. Hassas, "A survey on intrinsic motivation in reinforcement learning," *arXiv preprint arXiv:1908.06976*, 2019.
- [74] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.
- [75] R. I. Brafman and M. Tennenholtz, "R-max-a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, 2002.



- 
- 
- [76] M. Kearns and S. Singh, “Near-optimal reinforcement learning in polynomial time,” *Machine learning*, 2002.
- [77] S. Zhang, W. Boehmer, and S. Whiteson, “Generalized off-policy actor-critic,” in *Advances in Neural Information Processing Systems*, 2019.
- [78] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi, “Top-k off-policy correction for a reinforce recommender system,” in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019.
- [79] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014.
- [80] S. Fujimoto, D. Meger, and D. Precup, “Off-policy deep reinforcement learning without exploration,” *arXiv preprint arXiv:1812.02900*, 2018.
- [81] C. E. Shannon, “A mathematical theory of communication,” *Bell system technical journal*, 1948.
- [82] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “Vizdoom: A doom-based ai research platform for visual reinforcement learning,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016.



---

## A. Parameter $h$

---

This chapter consists of the results of the comparison between different approaches to determine the parameter  $h$  in EIM methods. Figure A.1 shows the average returns when using the entropy-based method of determining  $h$  in contrast to the use of constant values.

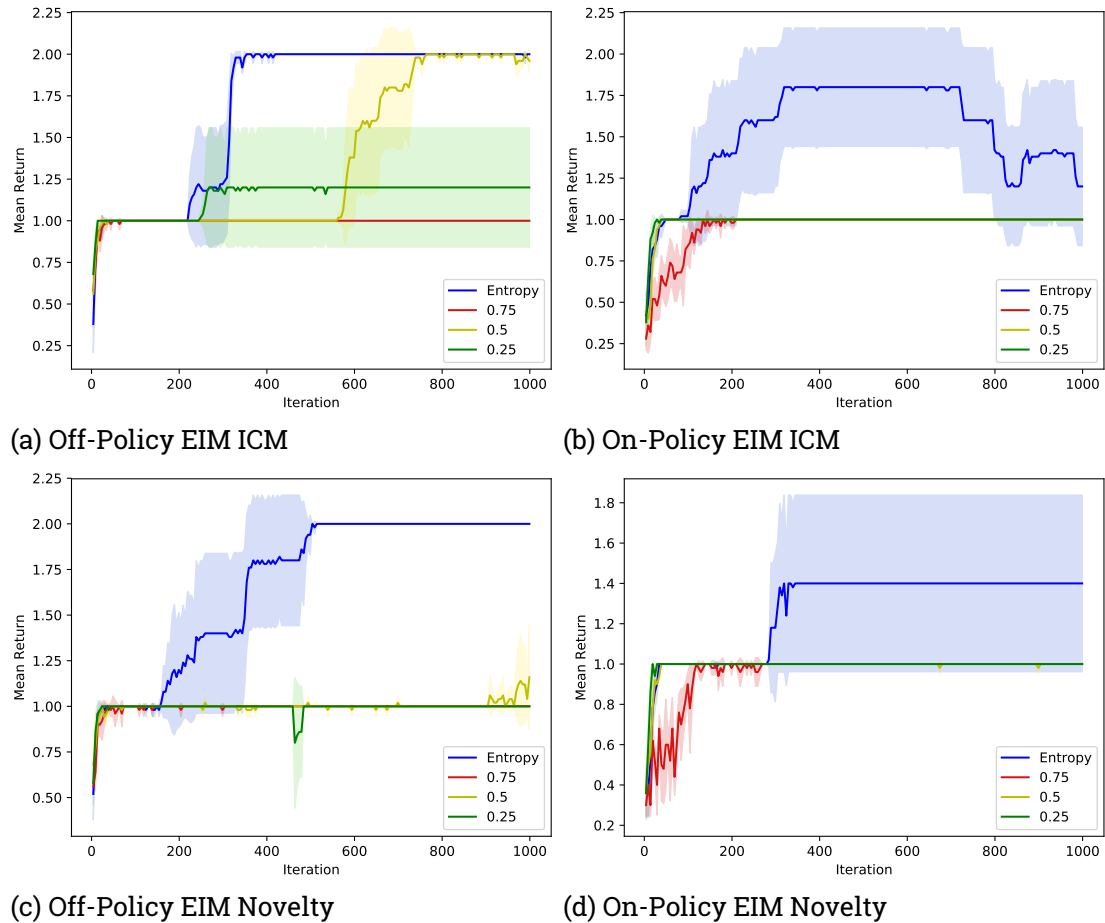


Figure A.1.: Mean extrinsic evaluation returns per iteration in the "Local Optimum" setup using different constant values for  $h$  compared to the entropy-based method averaged over five runs.

---

## B. Extensive Exploration

---

In this chapter, further experimental results highlighting the extensive exploration of EIM methods compared to IIM methods are provided. Figure B.1 depicts the total state visitations of 1000 iterations in the "Local Optimum" environment reflecting the tendency of IIM methods to follow only the optimal path to solve the extrinsic task, whereas EIM methods maintain a higher level of exploration. The environment is more evenly explored using the ICM approach regarding the EIM methods, which might be caused by the lower entropy of the exploration policy in many states (see Figure B.2) leading to a higher  $h$ . Figure B.2 shows that the exploration policy still provides an exploratory behavior after 1000 iterations.

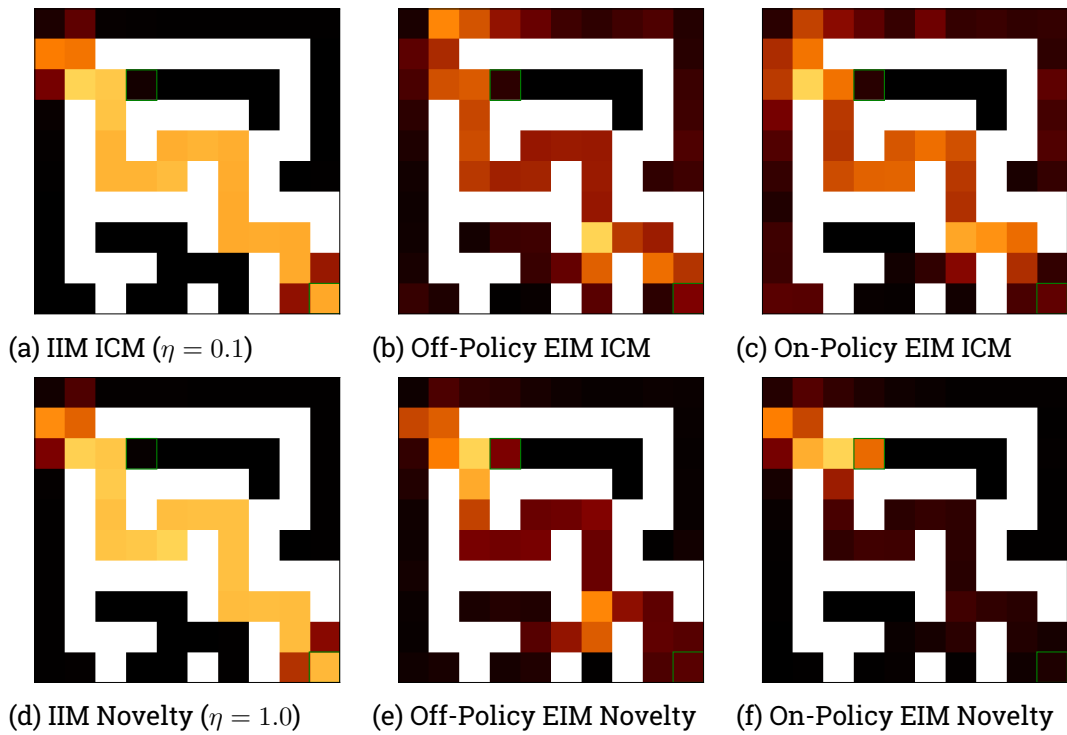
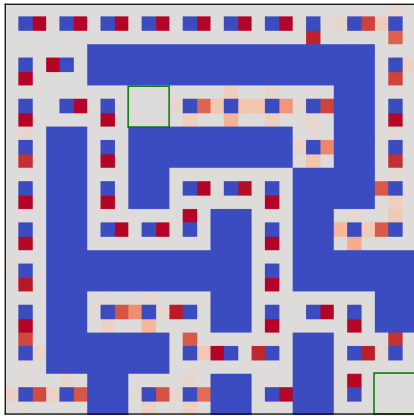
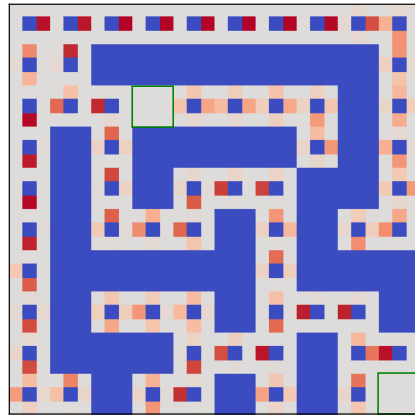


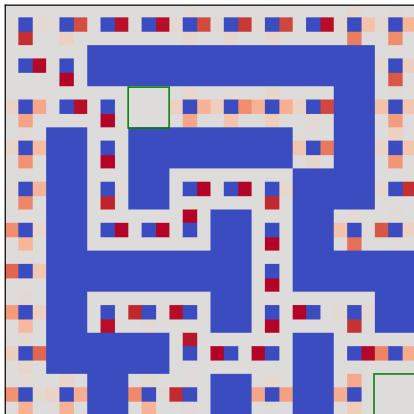
Figure B.1.: Heat maps depicting the total state visitations for every state after 1000 iterations in the "Local Optimum" setting. A brighter color indicates a higher number of visitations on the respective field. White squares represent wall fields. The green bordered fields mark the goal fields with the one in the bottom right corner being the optimal goal providing a reward of 2, and the other one providing a reward of 1.



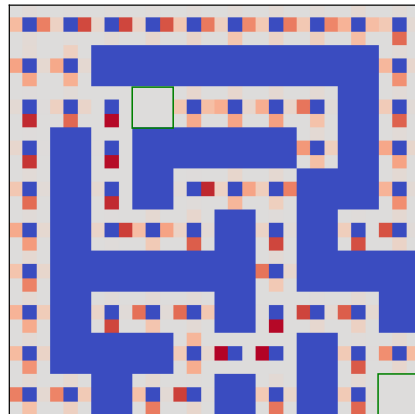
(a) Off-Policy EIM ICM



(b) On-Policy EIM ICM



(c) Off-Policy EIM Novelty



(d) On-Policy EIM Novelty

Figure B.2.: Visualization of the action probabilities of the exploration policy for every state after 1000 iterations in the "Local Optimum" setting. The big blue squares are wall fields, whereas the small blue squares are the center of each normal field. In each direction the agent can move (up, down, left, right), the center square has an additional square attached, which indicates the probability of choosing the according action, whereby a more intense red implies a higher probability. The green bordered fields mark the goal fields with the one in the bottom right corner being the optimal goal providing a reward of 2, and the other one providing a reward of 1.