Will it Blend? Learning to Coexecute Subskills

Reward-Shaping zur Echtzeit-Trajektorien-Synthese im Anwendungsfall Roboter-Trommeln
Bachelor thesis by Mohammadhossein Atashak
Date of submission: November 30, 2022

1. Review: Dr.-Ing Oleg Arenz
2. Review: Prof. Dr.-Ing. Rolf Findeisen
3. Review: Prof. Jan Peters, Ph.D.
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Hiermit versichere ich, Mohammadhossein Atashak, 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, 30. November 2022

M. Atashak

Abstract

In reinforcement learning we often try to solve a composed task, for which the corresponding sub-policies are easier to learn. However, it is not clear how they can be exploited without losing optimality. Finding a way to reuse existing sub-policies can alleviate the sample complexity problem, which limits the applicability of most reinforcement learning algorithms for real world robot learning problems. By having a practical approach to use existing sub-policies we can first decompose the task into sub-tasks, and then leveraging the prior knowledge about the sub-tasks can facilitate the learning process. In this work, we study how representing the learned sub-policies as advantage functions and using them to shape the environment's reward function can accelerate the learning without any adjacency requirements on the policies. For learning sub-policies we use the maximum entropy framework, which serves as a natural way to incorporate exploration into the agent.

We first validate the theory using a simple grid-world example, then we show how it can be applied in real world robotics applications, namely robot drumming. Our experimental evaluation demonstrates that combining sub-policies can result in substantial improvement in sample complexity and make the use of model free reinforcement learning algorithms in complex robotics applications possible.

Contents

1. Introduction

A cornerstone of human and animal intelligence is the ability to learn autonomously through trial and error. To that end, reinforcement learning (RL) presents a natural framework to develop learning algorithms for embodied agents.

Unfortunately, when learning is applied to real-world environments, or other environments with inherently sparse rewards spaces, the learning algorithms are notoriously sample inefficient. The high sample complexity is due to the large number of samples required to find the reward signal for the first time. Although this problem is not considered infeasible if the state space is small enough, it can quickly become intractable for complex continuous space states. While some approaches tried to solve this problem by using an existing potentially suboptimal policy [1, 2, 3] to guide the agents exploration of the state space, they do not provide a solution to use existing policies learned to tackle each subtask that together add up to the complete problem.

Another group of approaches try to solve the sample complexity problem by choosing the most efficient reward signal. One of the early examples [4] attempted to use a potential based function to reshape the reward and it showed by doing so the optimal policy remains unchanged. This is an effective solution especially when dealing with sparse reward functions, since the choice of the right potential based shaping function can decrease and even eliminate the sparsity of the reward and make the policy convergence during training orders of magnitude faster, but choosing the right function is often not trivial in practice and requires lots of experimentation and manual handcrafting. Additionally for most of the real world robot learning problems, handcrafting a near optimal shaping function would be, due to the complexity of required steps to fulfill the objective, almost impossible.

A more recent attempt [5, 6, 7] tries to incorporate exploration directly in the objective function by adding an auxiliary objective to maximize the entropy of the policy. By also maximizing the entropy of the policy, the agent assigns non-zero probability to all actions, but still higher probability to the actions with higher expected return. As a consequence,

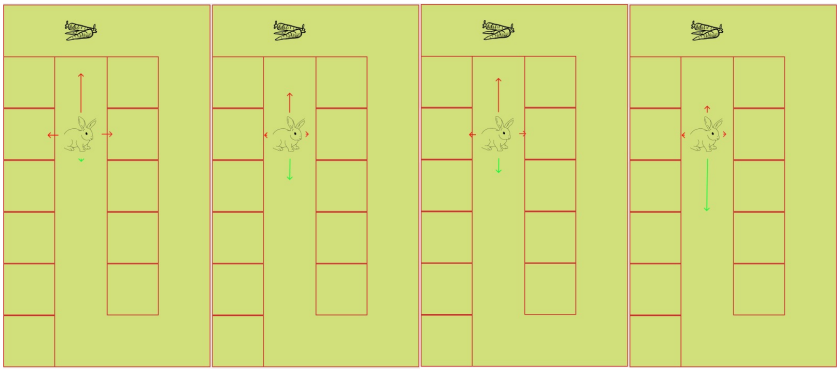

Figure 1.1.: From left to right: 1. optimal policy to reach the goal (without any penalty for collision). 2. optimal policy to just avoid the walls. 3. Naive combination of the last two policies. 4. optimal (our) policy combination for the complete task.

the resulting policy will be a combination of exploitation and exploration into the most promising regions. Maximum entropy framework provides the agent with an informed exploration strategy and leads to a substantial improvement in sample complexity and the agent can even find different ways of achieving the same goal, if the task at hand is sufficiently simple. Although maximum entropy provides a practical framework for elementary robot learning applications in the real world, it still needs an adequate amount of exploration for more complex tasks, and this can limit its range of applications in the real world. The exploration problem for complex tasks can be mitigated by breaking the task into simpler sub-tasks and using the learned policies from the sub-tasks to solve the complete task.

Haarnoja et al. [8] showed the efficiency of using soft Q-learning for learning robotic manipulation skills and tried to combine the maximum entropy policies by averaging the learned policies.
But as illustrated in Fig. 1.1 a naive combination of policies does not lead to finding a meaningful policy even when the combined policies are MaxEnt-optimal with respect to their reward functions. The naive composition of sub-policies by averaging them mostly leads to a sub-optimal resulting policy, because in most cases the objective of each sub-policy is in contrast to the others and as shown in [8] the optimality of the resulting policy is inversely correlated with the divergence of the constituent policies.

In this work we show a novel approach to combine policies from different agents by

representing them as advantage functions and using the advantage functions as potential based functions to reshape the reward, while the Maximum entropy framework with its inherent exploration strategy gives us a practical way to learn sub-policies with enough exploration. Then we demonstrate how our method can still yield near optimal result in real world robotic applications, without access to maximum entropy policies.
First we use the taxi environment to demonstrate the effectiveness of our approach and at the end, we show the practicability of it in robotics problems with robot drumming application.

2. Preliminaries

In the Reinforcement learning framework, the agent ought to take actions ($a \in \mathcal{A}$) in an environment in order to maximize the cumulative reward ($\mathcal{R}$). RL is commonly studied based on the Markov decision process (MDP) framework. An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the set of transition probabilities, $\mathcal{R} : \mathcal{S} \times \mathcal{A}$ ($\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}'$ for stochastic settings) is the reward function and $\gamma \in [0, 1]$ is the discounting factor. A policy $\pi : \mathcal{S} \times \mathcal{A}$ is defined as a probability distribution that maps actions to states and $\sum_{a \in \mathcal{A}} \pi(s, a) = 1, \forall s \in \mathcal{S}$. The goal of RL is to find an optimal policy π^* that maximizes the expected return $J(\pi)$ where:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi(\tau)}[r(\tau)] = \mathbb{E}_{\tau \sim \pi(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

and τ is the state action pairs for each episode $(s_0, a_0, s_1, a_1, ...)$. The distribution over trajectories $\pi(\tau)$ is given by:

$$\pi(\tau) = p(s_0) \prod_{t=0}^{\infty} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t).$$

The advantage function denoted as $A(s, a)$ measures the advantage of selecting each action for each state and is defined as:

$$A(s, a) = Q(s, a) - V(s) = \ln(pi^*) \tag{2.1}$$

where the state-action values function $Q(s, a)$ and the values function $V(s)$ are defined as:

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi, s' \sim P}[r(s, a) + V^{\pi}(s')], \tag{2.2}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{s' \sim P}[r(s, a) + \gamma \mathbb{E}_{a' \sim \pi}[Q^{\pi}(s', a')]] \tag{2.3}$$

with s', a' as next state and actions and $r(s, a)$ the immediate reward from taking action a in state s.

In this section we describe the main methods that were used either as the basis for the proposed method or to test it in different settings.

2.1. Maximum Entropy Reinforcement Learning

While the standard RL objective is to maximize Eq. 2, the goal of maximum entropy reinforcement learning is to maximize both its entropy and the expected return at each visited state. Therefore it changes the objective function to:

$$J(\pi_{MaxEnt}) = \arg\max_{\pi} \sum_{t=0}^{T-1} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(.|s_t))] \tag{2.4}$$

where $\mathcal{H}(\pi(.|s_t))$ represents entropy and α is a parameter that can be used to determine the relative importance of entropy and reward. The original RL objective term can be recovered by using $\alpha = 0$. Maximum entropy RL has a number of beneficial properties. First, instead of learning the best way to perform the task, the resulting policies try to learn all of the ways of performing the task. Apparently, if we can learn all of the ways that a given task might be performed, the resulting policies have better properties to get used as shaped reward as described in Chapter 4. Second, the policy is incentivized to explore more widely, while giving up on clearly unpromising avenues.

2.2. Soft Q-Learning

For optimizing the objective in 2.4 we can use the soft Q-learning algorithm [5]. Soft Q-learning optimizes a Q-funcion $Q(s, a)$ to predict the expected future return, which includes the future entropy value after taking an action a at state s and then following π. The optimal policy can be expressed in terms of the optimal Q-function as an energy based model(EBM):

$$\pi^*(a|s) \propto \exp(\frac{1}{\alpha}Q^*(s, a)). \tag{2.5}$$

For updating the Q-function, we can use the soft Bellman backup, which resembels the Bellman backup used in conventional Q-learning from Eq. 2.3:

$$Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)}[V(s')] \tag{2.6}$$

where the value function $V(s)$ is defined as the soft maximum of the Q-function over actions:

$$V(s) = \text{softmax}_a Q(s,a) = \alpha \log \int_{\mathcal{A}} \exp(\frac{1}{\alpha} Q(s,a)) da \tag{2.7}$$

Then the optimal policy for the objective function 2.4 is given by:

$$\pi^*_{\text{MaxEnt}}(a_t|s_t) = \exp(\frac{1}{\alpha}(Q^*_{\text{soft}}(s_t,a_t) - V^*_{\text{soft}}(s_t)))$$

and since the soft Bellman backup is a contraction and the optimal Q-function is the fixed point of the iteration in 2.6, we can transform any bounded function to the optimal Q-function by iteratively applying the soft Bellman backup until convergence. We will make use of the soft Bellman backup and the value function (Eq. 2.6 ,2.7) in section 5.2 to get expressive policies whose logarithm can be used to get the advantage function.

2.3. Reward shaping

One of the first papers exploring the use of reward shaping for speeding up the learning process by Andrew Y. Ng et al. [4] showed that shaping the reward function as the original reward combined with a difference of potentials shaping function:

$$R' = R + F(s,a,s') \tag{2.8}$$

with:

$$F(s,a,s') = \gamma * \Phi(s') - \Phi(s) \tag{2.9}$$

is both a necessary and sufficient condition for the optimal policy to remain unchanged. Using Eq. 2.8 and 2.9 we get as the new environments reward function:

$$R' = R + \gamma * \Phi(s') - \Phi(s) \tag{2.10}$$

($\Phi : \mathcal{S} \Rightarrow \mathcal{R}$ such that $s \in \mathcal{S} - \{s_0\}, a \in \mathcal{A}, s' \in \mathcal{S}$).
Shaping the environment's reward function as shown in Eq. 2.10 can eliminate the sparsity in the reward function, which leads to more efficient exploration and can result in

substantial improvement in sample complexity. As an intuitive example we can consider the problem of learning to drive a bicycle towards the goal. Using the shaping function in Eq. 2.10 to shape the environment's reward, results in a reward function that reinforces the agent proportional to the improvement in its potential to reach the goal at each time step and guides the agent exploration to the goal.
For the optimal value function under the shaped reward $V^*_{M'}(s)$ the author has also proved:

$$V^*_{M'}(s) = V^*_M(s) - \Phi(s)$$

and shown that choosing Φ in Eq. 2.9 and 2.10 based on domain knowledge such that in the ideal case $\Phi(s) = V^*_M(s)$ (with $\Phi(s_0) = 0$ in the undiscounted case) leads to better sample complexity in most of the standard RL gridworld environments. Recently, reward shaping has also been successfully applied in more complex problems such as Doom [9, 10] and Dota2 [11].
In Chapter 5 we show how reward shaping can be used to combine maximum entropy sub-policies (we prove that reward shaping can be extended to MaxEnt-RL in Appendix A). In Chapter 6 we demonstrate its effectiveness with an example in the real world setting, where we don't have access to optimal maximum entropy policies, by using reward shaping to combine the sub-policies from probabilistic movement primitives framework.

2.4. Probabilistic Movement Primitives

Probabilistic Movement Primitives (ProMPs) [12] have recently shown promising results in different robotics applications [13, 14]. It provides a practical framework for Movement Primitives(MPs) co-activation while encoding a time varying variance of movements to be able to capture multiple demonstrations with high variability.
In the ProMPs framework for one degree of freedom (DOF) the probability of observing a trajectory τ given the underlying weight vector ω is given as a linear basis function model

$$y_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \Phi_t^T \omega + \epsilon_y \tag{2.11}$$

$$p(\tau|\omega) = \prod_t \mathcal{N}(y_t|\Phi_t^T\omega, \Sigma_y)$$

where $\Phi_t^T = [\phi_t, \dot{\phi}_t]$ defines the $n \times 2$ dimensional basis functions for the joint positions q_t and velocities $\dot{q}_t$, n defines the number of basis functions and $\epsilon_y \sim \mathcal{N}(0, \Sigma_y)$ is zero-mean

i.i.d. Gaussian noise. By weighting the basis functions Ψ_t with the parameter vector ω, it can represent the mean of a trajectory.

For capturing the variance of the trajectories, it uses a distribution $p(\omega;\theta)$ over the weight vector ω, i.e., $p(\tau;\theta)$ can be computed by marginalizing out the weight vector ω :

$$p(\tau|\theta) = \int p(\tau|\omega)p(\omega;\theta)d\omega.$$

to obtain the probability distribution over the trajectories τ. The model's parameters are given by the observation noise variance Σ_y and the parameters θ of the weight distribution $p(\omega;\theta)$.
By defining the phase as $z_0 = 0$ at the beginning and $z_t = 1$ at the end of the movement we get the Gaussian basis functions b_i^G as:

$$b_i^G(z) = \exp(-\frac{(z_t - c_i)^2}{2h}),$$

where h defines the bandwidth and c_i the center for the ith basis function. After normalizing the basis functions, we get:

$$\phi_i(z_t) = \frac{b_i(z)}{\sum_{j=1}^{n} b_j(z)}.$$

For generalizing the approach to multiple DOFs, in order to capture the covariance of joint positions and velocities for each time step and encode a linear relationship between them, for each dimension i, we maintain a parameter vector w_i and by concatenating them we get the combined weight vector $w = [w_1^T, ..., w_n^T]^T$. The basis matrix Φ_t extends to a block-diagonal matrix containing the basis functions and their derivatives for each dimension. The observation vector y_t consists of the angles and velocities of all joints. The probability of observation y at time t is given by:

$$p(y_t|w) = \mathcal{N}\left(\begin{bmatrix} y_{1,t} \\ . \\ . \\ y_{d,t} \end{bmatrix} | \begin{bmatrix} \Phi_t & . & . & 0 \\ . & . & & . \\ . & & . & . \\ 0 & . & . & \Phi_t \end{bmatrix} w, \Sigma_y\right) = \mathcal{N}(y_t|\Psi w, \Sigma_y)$$

where $y_{i,t} = [q_{i,t}, \dot{q}_{i,t}]$ denotes the joint angle and velocity for the ith joint,
Furthermore Eq. 2.11 becomes:

$$p(\tau|w) = \prod_t \mathcal{N}(y_t|\Psi_t w, \Sigma_y).$$

In order to learn the weights w we can use expert demonstrations. For learning from demonstrations a Gaussian distribution for $p(w;\theta) = \mathcal{N}(w|\mu_w, \Sigma_w)$ over the parameters w is assumed. Consequently, the distribution of the state $p(y_t|\theta)$ for time step t is given by:

$$p(y_t;\theta) = \int \mathcal{N}(y_t|\Psi_t w, \Sigma_y)\mathcal{N}(w|\mu_w, \Sigma_w)dw = \mathcal{N}(y_t|\Psi_t\mu_w, \Psi_t\Sigma_w\Psi_t^T + \Sigma_y) \quad (2.12)$$

and, thus the mean and the variance of resulting trajectory distribution for any time point t can get evaluated. As there is a linear dependency between the parameters μ_w, Σ_w and the resulting trajectory at timestep t, we can learn the parameters $\theta = \mu_w, \Sigma_w$ from demonstrations by maximum likelihood estimation algorithm. We can first estimate the weights for each trajectory individually as:

$$w_i = (\Psi^T\Psi + \lambda I)^{-1}\Psi^T Y_i, \quad (2.13)$$

where λ denotes the ridge factor and is generaly set to a very small value ($\lambda = 10^{-12}$ in our case). Then the mean μ_w and covariance Σ_w are computed from the samples w_i:

$$\mu_w = \frac{1}{N}\sum_{i=1}^{N} w_i, \hat{\Sigma}_w = \frac{1}{N}\sum_{i=1}^{N}(w_i - \mu_w)(w_i - \mu_w)^T \quad (2.14)$$

where N is the number of demonstrations.

Using the Eq. 2.13 and 2.14 to learn weights for periodic movements, gives us the possibility to learn a trajectory distribution from as few as 10 human demonstrations and eases the data collection bottleneck in real world robotic applications.

3. Related Works

Multiobjective Reinforcement Learning (MORL) [15] can be viewed as the combination of multi-objective optimization (MOO) and RL techniques to solve the sequential decision making problems with multiple conflicting objectives. In order to approximate a solution to problems with multiple objectives, MORL algorithms try to find the policy/set of policies, which can give the best approximation of the Pareto frontier. Pareto frontier represents the set of all Pareto efficient solutions. A policy π_A is Pareto efficient when there is no other policy π_N which can achieve better results in all objectives than π_A (Fig. 6.3 illustrates the Pareto frontier for the case of two objectives).
In MORL for each objective i $(N \geqslant i \geqslant 1)$ and a stationary policy π, there is a vectored state-action Q-function $MQ^{\pi}(s,a)$ such that:

$$MQ^{\pi}(s,a) = [Q_1^{\pi}(s,a), Q_2^{\pi}(s,a), ...Q_N^{\pi}(s,a)]^T$$

with the optimal vectorized state-action Q-function defined as:

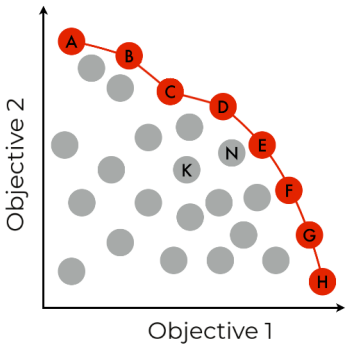

Figure 3.1.: Red line represents the Pareto frontier. No other policy(gray circles) can achieve superior results in both objectives compared to Pareto efficient policies(red circles).

$$MQ^{*}(s,a) = \max_{\pi} MQ^{\pi}(s,a) \tag{3.1}$$

where the "max" operator for a vector is defined either in the sense of Pareto optimality or in the sense of maximizing a weighted scalar of all the elements. From Eq. 3.1 the optimal policy can be easily obtained by:

$$\pi^{*}(s) = \arg\max_{a} MQ^{*}(s,a).$$

The MORL approaches can be divided in two groups based on the number of policies to be learned [16]: single policy and multiple policy approaches. single policy approaches try to find the optimal policy, which simultaneously satisfies all the objective functions. As a naive solution for the single policy case, we can design a synthetic Q-function ($TQ(s,a)$), which can suitably represent the overall preferences. The goal of multiple policy approaches is to solve MORL problem by obtaining a set of policies, which approximate the Pareto frontier. A naive solution for the case of multiple policy approaches is to synthesize a set of Q-functions, that their corresponding policies represent the Pareto frontier.
The main difference between single policy MORL and our approach is that instead of directly using sub-policies to get a new policy, we leverage the sub-policies in our new reward function and still run RL algorithms on the new reward to get the new, Pareto efficient, policy.
In the rest of this Section, we describe two prior works which attempt to approximate a Pareto efficient single policy solution for MORL problem.

One of the prior works [17] considers solving "or" combination of objectives (e.g., move to the target OR hold the glass) under the linear MDP assumption. Linear MDP considers the setting where cost is a function of state, with an action cost that is given by the KL divergence between the desired transition probabilities and the passive dynamics.

In order to solve the "and" combination of objectives (move to the target AND hold the glass) without any assumption for underlying MDP, recently Haarnoja et al [8] showed different policies trained using soft Q-learning can get combined by averaging the Q-function corresponding to each policy in the form:

$$Q^*_{\mathcal{C}}(s,a) \approx Q_{\Sigma}(s,a) = \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}} Q^*_i(s,a),$$

where $Q^*_{\mathcal{C}}(s,a)$ represents the optimal Q-function of the composed task.
Additionally it shows as an example for the case that $Q_{\Sigma}(s,a) \triangleq \frac{1}{2}(Q^*_1 + Q^*_2)$, the optimal soft Q-function of the combined reward $r_{\mathcal{C}} \triangleq \frac{1}{2}(r_1 + r_2)$ satisfies:

$$Q_{\Sigma}(s,a) \geq Q^*_{\mathcal{C}}(s,a) \geq Q_{\Sigma}(s,a) - C^*(s,a),\ \forall s \in \mathcal{S},\ \forall a \in \mathcal{A} \tag{3.2}$$

where C^* is the fixed point of :

$$C(s,a) \leftarrow \gamma \mathbb{E}_{s'\sim p(s',|s,a)}[\mathcal{D}_{\frac{1}{2}}(\pi^*_1(.|s')||\pi^*_2(.|s')) + \max_{a'\in\mathcal{A}} C(s',a')] \tag{3.3}$$

with $\mathcal{D}_{\frac{1}{2}}(.||.)$ being the Renyi divergence of order $\frac{1}{2}$.

As we can see from Eq. 3.2 and 3.3, the difference of the composite Q-function $Q_{\Sigma}(s, a)$ compared to the optimal Q-function $Q^*_{\mathcal{C}}(s, a)$ depends heavily on the divergence of the constituent policies. This property leads to highly suboptimal Q-function and as the result bad policy, if the combined policies have goals that contradict each other.

4. Proposed Method

In this Chapter we describe our approach to maximize a combination of reward functions as:

$$r_{\text{combine}} = r_1 + r_2 + r_3 + \dots \tag{4.1}$$

such that the optimal policy satisfies:

$$\pi^* = \arg\max_{\pi}(\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r_{combine}(s_t,a_t) + \alpha\mathcal{H}(\pi(.|s_t))]). \tag{4.2}$$

In order to approximate the optimal policy from Eq. 4.2 we use the combination of sub-policies, each trained separately on one of the constituent rewards from Eq. 4.1, to shape the environments reward function. More specifically, we assume that for some of the individual objectives r_i we have the optimal policy $\pi_i^* = \arg\max_\pi(\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r_i(s_t,a_t) + \alpha\mathcal{H}(\pi_i(.|s_t))])$. Since each policy has been obtained independently, they can contradict each other, therefore the optimal method for composing the policies should work equally well, independent of divergence of the constituent policies. The novel approach introduced in this work, has no adjacency requirements on sub-policies. Therefore it can work equally well, even if the goal of the constituent policies contradict each other. After computing the shaped reward, we can train a new policy on this new reward function.
In the next Section we show, how we can use the reward shaping theory, introduced in Chapter 2 to combine the n policies as a new shaped reward.

4.1. Theoretical Overview

Given Eq. 2.10 and by using $\Phi(s) = V_M(s)$ and $\Phi(s') = V_M(s')$ as potential based shaping functions, we can derive:

$$r_s(s,a,s') = \overbrace{r(s,a,s') + \gamma * v\left(s'\right)}^{Q(s,a,s')} - v\left(s\right) \tag{4.3}$$

as shaped reward. We can now use the n policies and Eq. 4.3 and 2.1 to compute the shaped reward as:

$$r_s(s,a,s') = Q_1(s,a,s') + Q_2(s,a,s') + \ldots + Q_n(s,a,s') - \\ (v_1(s) + v_2(s) + \ldots + v_n(s)) = \ln(\pi_1) + \ln(\pi_2) + \ldots + \ln(\pi_n) \quad (4.4)$$

and for the case of having n value functions we use the Eq. 4.3 in the form:

$$r_s(s,a,s') = \overbrace{(r_1(s,a,s') + r_2(s,a,s') + ... + r_n(s,a,s'))}^{r_{\text{combine}}} + \\ \gamma * \underbrace{(v_1(s') + v_2(s') + ...v_n(s'))}_{\phi(s')} - \underbrace{(v_1(s) + v_2(s) + ...v_n(s))}_{\phi(s)}. \quad (4.5)$$

to shape the reward.

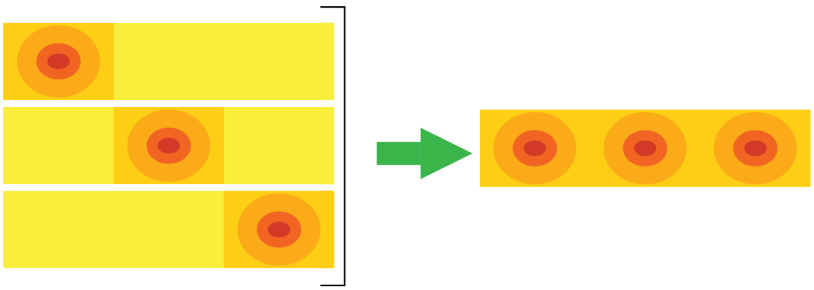

Figure 4.1.: left: three policies($\pi_n,\ for\ n \in \{1,2,3\}$), each localy concentrated at one region and uniformlly distributed in other regions, used to shape reward. right: the resulting policy trained on the shaped reward ($r_s = \sum_{n=1}^{3} ln(\pi_n)$) using the last three policies.

Using Eq. 4.5 and 4.4 to shape the environments reward, gives use two practical advantages. First it gives us an effective way to use the existing sub-policies to eliminate the sparsity in the reward function and guides the agent exploration, without the need for handcrafting the potential based function (Φ for Eq. 2.10) based on domain knowledge. We use this advantage in the next chapter to speed up the agents learning process in the taxi environment.
Additionally, in some RL applications we can exploit the locality of the constituent policies as illustrated in Fig. 4.1 and using the Eq. 4.4, we can extract the shaped reward that can be used to optimize a policy which can fulfill the goals of all composed policies. Chapter

6 represents such an real world robotics example with robot drumming. In this setting we can use the temporal locality of ProMP policies to get the shaped reward. We show how, using the temporal locality of hitting each drum table, we can optimize a policy which is able to play the drumming notes with high temporal precision, surpassing best human drummers.

5. Gridworld Experiments

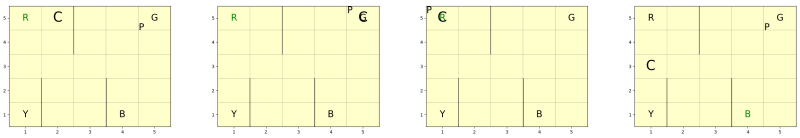

Figure 5.1.: Overview of a successful episode of the taxi environment: in this case the agent picks up the passenger at location "G" and drops it off at "R" and the environment restarts.

As our gridworld experiment, we chose for simplicity and clarity the Taxi environment. In this environment, there are four designated locations in the gridworld. When the episode starts, the taxi starts off at a random square and the passenger is at a random location (from designated locations). The taxi (agent) should drive to the passenger's location, pick up the passenger, drive to the passenger's destination (another one of the four specified locations), and then drop off the passenger without colliding to the walls. Once the passenger is dropped off, the episode ends. The reward for each step is -1, except for the step where the passenger is dropped off, which is +20, and the reward of collision which is -100.

In order to get the sub-policies we use two modified versions of the reward that sum up to the original reward. In the first version, the reward is zero for visiting every state and -100 for collision. In the second version, the reward is -1 for every state and +20 for successful passenger dropoff with no extra penalty for collision. The goal of the first reward function is to teach the agent to explore the state space without any collision, while the goal of the second reward function is to teach the agent to pick up the passenger and drop it off at the goal location, while choosing the most efficient path.

For computing the value matrix, soft value iteration or Soft-Q-Learning [5] can be used. Soft value iteration is used to demonstrate the effectiveness of the approach, when the

environment is deterministic and its transition probabilities matrix ($\mathcal{T}(s, a, s')$) is known and serves as a baseline for measuring the improvement through using the proposed methode. While the improvement becomes more apparent as we use soft Q-learning with stochastic environment and unknown transition probabilities.

5.1. Soft value iteration

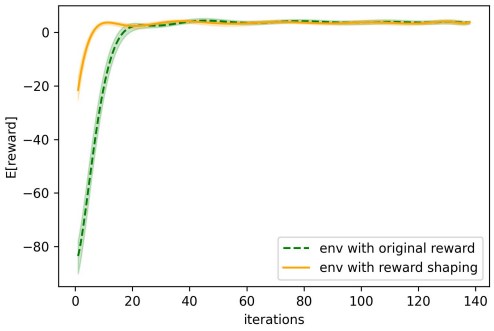

Figure 5.2.: Expected reward (on 10000 episodes rollout) of soft value Iteration in the deterministic setting from the agent trained on the shaped reward compared to the agent trained on the original reward function.

In the soft value iteration setting we use the Bellman equation (Eq. 2.6) for the deterministic setting with known transition probabilities in the form:

$$Q_{k+1} = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s')), \ for\ k \geqslant 0$$

then we can get the soft value function as in Eq. 2.7 and the corresponding policy as shown in Eq. 2.5.
Now we can use the Eq. 4.4 and sum the log of the two policies to get the shaped reward as:

$$R_s = lg(\pi_1) + lg(\pi_2)$$

and after training an agent on this reward, we evaluate its performance by comparing it to the agent trained on the original reward function.

From Fig. 5.2 we can see that the policy trained on the shaped reward converges to the optimal policy after just a few iterations, while for the case of the original environment's reward function, it takes 20 to 30 iterations to reach the optimal policy.

5.2. Soft Q-Learning

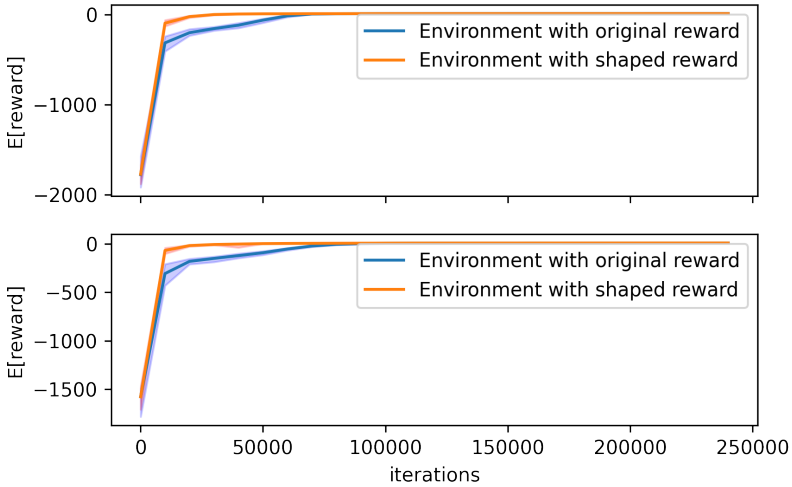

Figure 5.3.: Expected reward (on 10000 episodes rollout) of soft Q-Learning from the agent trained on the shaped reward compared to the agent trained on the original reward function. Top: deterministic setting. Down: stochastic setting with 10 % stochasticity.

In soft Q-Learning, the assumption $\pi = \exp(Q - V)$ may be violated in practice due to sample-based optimization, and $\log(\pi)$ may not be a shaped reward. Hence, we directly use the values functions obtained by Q-learning (Eq. 2.6 and 2.7) to shape the reward as shown in Eq. 4.5 in the form:

$$R_s = r_1[s,a] + r_2[s,a] + gamma * (V_1[s'] + V_2[s']) - (V_1[s] + V_2[s]).$$

After training an agent on this reward, we evaluate its performance by comparing it to the agent trained on the original reward function.

In soft Q-learning setting we can see even a bigger improvement in sample complexity and as shown in Fig. 5.3, as we increase the stochasticity of the environment the gain gets even larger. We can observe an approximate 2x improvement in sample complexity in the deterministic setting, and as the environment becomes more stochastic, the unshaped Q-learning algorithm get slower to converge, while the Q-learning algorithm with shaped reward still reaches the optimal policy after the first few thousand iterations. This suggests further big improvements in the real world robotics applications, as the complexity of the state space increases.

6. Real-World Manipulation

Figure 6.1.: To illustrate the effectiveness of reward-shaping on physical hardware, we trained six ProMPs for two drums with just a few samples for each ProMP. Their probability distributions in joint space is used to shape the reward. We showed the policy fitted on the shaped reward is able to successfully play drum.

Robot Drumming represents, due to the temporal coordination of movements, a challenging real world robotic application. To illustrate how our approach can be used to combine learned policies in the real world robotics setting, we used the wam robotic hand with 2 drum tables (as shown in Fig. 6.1) and learned 3 ProMPs per each drum table (6 in total). Although in this work we have considered the setting with 2 drum tables, our methods are generalizable to a complete drum set. In order to get the Gaussian distribution over joints from the learned ProMP weights, we use Eq. 2.12 where we consider the case $\lim(\Sigma_y) = 0$. After getting the corresponding ProMP joint distributions, we use the probability density

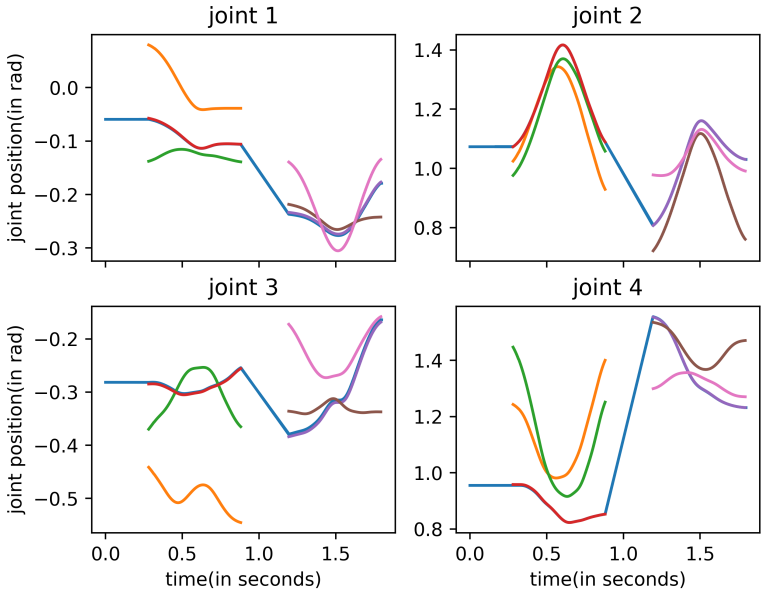

Figure 6.2.: The optimized trajectory (blue line) converges for each joint to the ProMP mean that best fits the movement.

of the corresponding ProMPs for the drum table to shape the reward as:

$$r_s = \sum_{j=0}^{1} \log(\sum_i \exp(\log p_j(i))) \tag{6.1}$$

where $p_j(i)$ represents the probability density of the trajectory section for drum table j, under the ith ProMP. Additionally, we have defined a velocity, acceleration and jerk penalty over the whole trajectory as:

$$r_{vaj} = -(\dot{tr}^2 + \ddot{tr}^2 + \dddot{tr}^2)/\alpha \tag{6.2}$$

with α as a factor that controls the effect of this penalty and tr representing the trajectory that we optimize.
Now we can simply add the reward terms from 6.2 and 6.1 to get the complete reward for the trajectory:

$$r_{trajectory} = r_{vaj} + r_s. \tag{6.3}$$

By optimizing a trajectory on reward function from 6.3 as illustrated in Fig. 6.2, the optimized trajectory converges at each note to the right ProMP in order to play the right drum.

Additional to playing the right drum, one of the major factors that indicates the drumming abilities is the correct timing. In order to evaluate the temporal error of our approach in playing the musical notes, we used the signal from the drumset, which gives us the absolute time of hitting the drum t_{hit}. By comparing t_{hit} with the absolute time of the start of movement and averaging it over the different strikes, we could show that we have achieved as shown in Fig. 6.3 more than 3x enhancement in temporal precision compared to professional drummers.

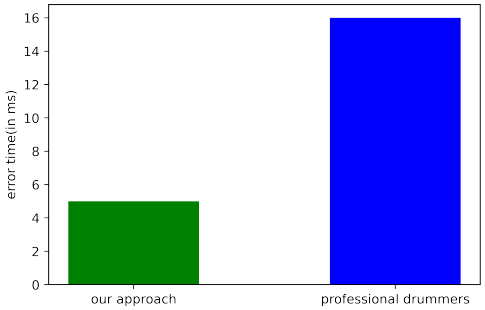

Figure 6.3.: The average temporal error comparison between our approach and professional drummers in playing the drum notes.

Our successful results with robot drumming experiment suggest our method as an effective approach to use in real world robotic tasks, where we have already access to different near optimal sub-policies.

7. Discussion

In this work, we discuss how reward shaping can be used to leverage existing policies in MaxEnt settings. We assume that the reward is composed of individual rewards and that we have access to maximum entropy optimal policies for each objective. In order to optimize a single policy which maximizes the complete reward, we show that we can still learn an optimal policy by replacing the sum of the individual reward functions with the log probability density of the corresponding policy. First we start with a gridworld example, where our experimental results suggest that using reward shaping to leverage existing policies leads to an approximate 2x improvement in sample complexity compared to using the original reward function with both soft value iteration and soft Q-learning algorithms. The improved sample complexity in the taxi environment validate our method as a practical approach to leverage existing MaxEnt policies in order to increase the training efficiency, while preserving the optimality in the resulting policy.
In order to demonstrate the effectiveness of our method in complex real world robotic problems, where due to the high cost for data collection we don't have access to optimal MaxEnt policies, we show how we can exploit temporal and/or spatial locality in policies to shape the reward. We explore the usefulness of this property in the robot drumming experiment. In this experiment, we train a few ProMPs for each drum table from human demonstrations. Afterwards, we exploit that the probability density of the optimal policy for a given note is approximately constant at timesteps far from the desired hitting time, which enables us to ignore their contribution when computing the shaped reward at distant timesteps. The method proves successful to play music notes with high temporal precision, using only a limited number of demonstrations to train ProMPs for each drum table. The success of our method to continuously optimize a trajectory, solely based on the given notes, for the right drum table with super human temporal precision suggest it as powerful approach for complex robotic tasks.

8. Outlook

In our robot drumming experiment, we show how by leveraging the locality in sub-policies to shape the reward function we can successfully play drumming notes with high temporal precision. An interesting avenue for future work would be to study how we can combine the optimization of the resulting trajectory with optimizing the sub-policies, such that they both improve simultaneously. Additionally, answering such questions would give us an insight into how we can optimize sub-policies, while using them as shaped reward to continuously adapt to changing objectives.

Bibliography

[1] I. Uchendu, T. Xiao, Y. Lu, B. Zhu, M. Yan, J. Simon, M. Bennice, C. Fu, C. Ma, J. Jiao, S. Levine, and K. Hausman, "Jump-start reinforcement learning," 2022.

[2] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *CoRR*, vol. abs/1709.10087, 2017.

[3] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," *CoRR*, vol. abs/1709.10089, 2017.

[4] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *In Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287, Morgan Kaufmann, 1999.

[5] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," 2017.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[7] M. C. Silver D, Huang A, "Mastering the game of go with deep neural networks and tree search.," *Nature*, no. 529(7587):484-489, 2016.

[8] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," *CoRR*, vol. abs/1803.06773, 2018.

[9] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, Feb. 2017.

[10] S. Song, J. Weng, H. Su, D. Yan, H. Zou, and J. Zhu, "Playing fps games with environment-aware hierarchical reinforcement learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 3475–3482, International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[11] "OpenAI-openai five."

[12] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[13] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using probabilistic movement primitives in robotics," *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, 2018.

[14] M. Ewerton, O. Arenz, G. Maeda, D. Koert, Z. Kolev, M. Takahashi, and J. Peters, "Learning trajectory distributions for assisted teleoperation and path planning," *Frontiers in Robotics and AI*, vol. 6, p. 89, 2019.

[15] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 3, pp. 385–398, 2014.

[16] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms," *Machine learning*, vol. 84, no. 1, pp. 51–80, 2011.

[17] E. Todorov, "Compositionality of optimal control laws," *Advances in neural information processing systems*, vol. 22, 2009.

A. Reward Shaping proof (of sufficiency) for maximum entropy setting

Let F be of the form given in Eq. 2.9 and if $\gamma = 1$ then replacing $\Phi(s)$ with $\Phi'(s) = \Phi(s) - k$ for any constant k would not change the shaping rewards F (which is the difference of these potentials), we may, by replacing $\Phi(s)$ with $\Phi(s) - \Phi(s_0)$ if necessary assume without loss of generality that Φ used to express F via 2.9 satisfies $\Phi(s_0) = 0$.

For the original MDP M, we know that its optimal Q-function Q^*_M satisfies the Bellman equations for maximum entropy setting (2.6,2.7) where:

$$Q^*_M(s,a) = E_{s' \sim P_{sa}(.)}[R(s,a,s') + \gamma \text{softmax}_{a' \in A} Q^*_M(s',a')]$$

with $Q^*_M(s,a)$ as the optimal Q-funcion. Now some simple algebraic manipulation gives us:

$$Q^*_M(s,a) - \Phi(s) = E_{s' \sim P_{sa}(.)}[R(s,a,s') + \gamma\Phi(s') - \Phi(s) + \gamma \text{softmax}_{a' \in A}(Q^*_M(s',a') - \Phi(s'))]. \quad \text{(A.1)}$$

Now by defining $\widehat{Q}_{M'}(s,a) \triangleq Q^*_M(s,a) - \Phi(s)$ and substituting that and $F(s,a,s') = \gamma\Phi(s') - \Phi(s)$ back into the Eq. A.1, we get:

$$\begin{aligned}\widehat{Q}_{M'}(s,a) &= E_{s' \sim P_{sa}(.)}[R(s,a,s') + F(s,a,s') + \gamma \text{softmax}_{a' \in A} \widehat{Q}_{M'}(s',a')] \\ &= E_{s' \sim P_{sa}(.)}[R'(s,a,s') + \gamma \text{softmax}_{a' \in A} \widehat{Q}_{M'}(s',a')] \quad \text{(A.2)}\end{aligned}$$

but this is exactly th Bellman equation for M'. So, $\widehat{Q}_{M'}(s,a)$ satisfies the Bellman equations for M', and must in fact be unique optimal Q-function. Thus, $Q^*_{M'}(s,a) = \widehat{Q}_{M'}(s,a) = Q^*_M(s,a) - \Phi(s)$, and the optimal policy for M' therefore satisfies from Eq. 2.5:

$$\pi^*_{M'}(s) \in \exp(\frac{1}{\alpha} Q^*_{M'}(s,a)) = \exp(\frac{1}{\alpha}(Q^*_M(s,a) - \Phi(s))) = \exp(\frac{1}{\alpha} Q^*_M(s,a))$$

and is therefore also optimal in M. To show every optimal policy in M is also optimal in M', simply apply the same proof with the roles of M and M' interchanged (and using the shaping funcion $-F$).