Abstract

We introduce the Adaptive Skills, Adaptive Partitions (ASAP) algorithm that (1) learns skills (i.e., temporally extended actions or options) as well as (2) where to apply them to solve a Markov decision process. ASAP is initially provided with a misspecified hierarchical model and is able to correct this model and learn a near-optimal set of skills to solve a given task. We believe that (1) and (2) are the core components necessary for a truly general skill learning framework, which is a key building block needed to scale up to lifelong learning agents. ASAP is also able to solve related new tasks simply by adapting where it applies its existing learned skills. We prove that ASAP converges to a local optimum under natural conditions. Finally, our extensive experimental results, which include a RoboCup domain, demonstrate the ability of ASAP to learn where to reuse skills as well as solve multiple tasks with considerably less experience than solving each task from scratch.

1. Introduction

Many problems are naturally modelled using hierarchical abstractions. Consider a soccer game where the objective is for one team to score more goals than their opponent and win the game. Each player has a given set of primitive actions such as pass, dribble and shoot, but in order to win, need to combine these actions with other teammates and devise hierarchical strategies such as 

\textit{Attack} and \textit{Defend} in order to beat their opponent (Bai et al., 2012; Hausknecht & Stone, 2015). Robotics, video games, navigation tasks and maintenance schedules in smart grids are some examples of applications that contain inherent hierarchical abstractions (Peters & Schaal, 2008; Mann et al., 2015; Bu et al., 2014).

Modelling real-world problems with \textit{useful} hierarchical abstractions is a non-trivial, time-consuming endeavor. Domain experts are responsible for developing these models, and often generate incorrect hierarchical abstractions, especially in complex, real-world domains, causing degenerate, sub-optimal performance. We refer to modelling a problem with an incorrect hierarchical abstraction as \textit{model misspecification} (Mankowitz et al., 2014).

One way to build hierarchical abstractions in Reinforcement Learning (RL) is to build a policy that composes together lower level policies. A policy is a solution to a Markov Decision Process (MDP) and is defined as a mapping from states to a probability distribution over actions. That is, it tells the RL agent which action to perform given the agent’s current state. For MDPs with large state spaces, it is infeasible to store which action to perform for every possible state. Thus, the agent’s policy must be \textit{generalizable} (Bertsekas et al., 1995; Sutton, 1996) such that the agent will take similar actions in nearby states. The popular technique that we use in this paper for achieving generalization is \textit{Linear Function Approximation} (LFA) (Sutton, 1996). A \textit{compact} policy is defined as a policy parameterized using LFA.

Generating a set of compact policies, also referred to as options, skills, macro-actions (Hauskrecht et al., 1998; He et al., 2011; Sutton et al., 1999; Konidaris & Barto, 2009), and combining them into a \textit{hierarchical policy} (that chooses which skills to execute and where to execute them) enables an agent to solve a task whilst taking advantage of the hierarchical nature of the task. From here on in, we will use the term \textit{skill} when referring to a compact policy.
Hierarchical abstraction in RL has become popular in many domains including RoboCup soccer (Bai et al., 2012; Hausknecht & Stone, 2015), video games (Mann et al., 2015) and Robotics (Fu et al., 2015). Here, taking advantage of the hierarchical nature of the domains (strategies in RoboCup, strategic move combinations in video games and skill controllers in Robotics for example) has generated impressive solutions. In addition, it has been shown both experimentally (Precup & Sutton, 1997; Sutton et al., 1999; Silver & Ciosek, 2012) and theoretically (Mann & Mannor, 2014) that making use of hierarchical abstraction (also referred to as temporal abstraction) speeds up the convergence rates of RL planning algorithms.

It is however, non-trivial and time-consuming to design a good set of skills and combine these skills into a hierarchical policy that solves a given task. It is not always clear what skill is good for a particular domain. In addition, learning a hierarchical policy is non-trivial as the agent needs to know which skill to execute at any given state. This is particularly difficult in continuous, high dimensional domains where the state space is large. Sub-optimal skills and a sub-optimal hierarchical policy lead to model misspecification.

A truly general skill learning framework must (1) learn skills as well as (2) determine where they should be executed. This framework should also determine (3) when skills can be reused in different parts of the state space and (4) adapt to changes in the task itself. A number of works have addressed some of these issues separately, but no work, to the best of our knowledge, has developed this truly general skill-learning framework. (Eaton & Ruvolo, 2013; Ammar et al., 2014; 2015) have developed related optimization frameworks for learning across multiple tasks in continuous domains, but only learns a single policy to solve each domain and does not take advantage of temporal abstraction. (Brunskill & Li, 2014) provide a technique for improving the sample complexity of option discovery in lifelong RL for discrete domains. (Mankowitz et al., 2014; Comanici & Precup, 2010) learn the termination conditions of options. (Bacon & Precup, 2015) provide a new option-critic architecture for learning option policies and termination conditions simultaneously.

Our framework entitled ‘Adaptive Skills, Adaptive Partitions (ASAP)’ is the first of its kind to incorporate all of the above-mentioned elements into a single framework and solve continuous state MDPs. It receives as input a misspecified model (a sub-optimal set of skills and hierarchical policy). These skills are incorporated in a Bayesian-like manner into a hierarchical policy, which we refer to as the ASAP policy, and a near optimal set of skills as well as the ASAP policy are learned simultaneously.

Main Contributions: (1) The Adaptive Skills, Adaptive Partitions (ASAP) algorithm which automatically corrects a misspecified model. It learns a set of near-optimal skills, skill execution regions and a hierarchical policy to solve a given task. (2) Learning skills over multiple different tasks by automatically adapting both the hierarchical policy, skill execution regions and the skill set. (3) ASAP can determine where skills should be reused in the state space. (4) The ability of ASAP to learn using offline data.

2. Background

2.1. Reinforcement Learning Problem

A Markov Decision Process is defined by a 5-tuple \( \langle X, A, R, \gamma, P \rangle \) where \( X \) is the state space, \( A \) is the action space, \( R \in [-b, b] \) is a bounded reward function, \( \gamma \in [0, 1] \) is the discount factor and \( P : X \times A \rightarrow [0, 1] \) is the transition probability function for the MDP. The transition probability function maps the current state \( x \) and action \( a \) to a probability distribution over next states. The solution to a MDP is a policy \( \pi : X \rightarrow \Delta_A \) which is a function mapping states to a probability distribution over actions. The optimal policy \( \pi^* : X \rightarrow \Delta_A \) determines the best actions to take so as to maximize the expected reward. The value function in Equation 1 defines the expected reward for following a policy \( \pi \) from state \( x \).

\[
V^\pi(x) = \mathbb{E}_{a \sim \pi(\cdot|x)} R(x, a) + \gamma \mathbb{E}_{x' \sim P(\cdot|x, a)} V^\pi(x')
\]  

(1)

The optimal expected reward \( V^\pi^*(x) \) is the expected value obtained for following the optimal policy from state \( x \).
2.2. Policy Gradient

Policy Gradient (PG) methods have enjoyed success in recent years especially in the fields of robotics (Peters & Schaal, 2006; 2008). The goal in PG is to learn a policy $\pi_\theta$ that maximizes the expected return:

$$ J(\pi_\theta) = \int_\tau P(\tau) R(\tau) d\tau, $$

where $\tau$ is a set of trajectories, $P(\tau)$ is the probability of a trajectory and $R(\tau)$ is the reward obtained for a particular trajectory. $P(\tau)$ is defined as follows:

$$ P(\tau) = P(x_0) \prod_{k=0}^{T} P(x_{k+1}|x_k, a_k) \pi_\theta(a_k|x_k). $$

Here, $x_k \in X$ is the state at the $k^{th}$ timestep of the trajectory; $a_k \in A$ is the action at the $k^{th}$ timestep; $T$ is the trajectory length. Only the policy, in the general formulation of policy gradient, is parameterized with parameters $\theta$. The idea is then to update the policy parameters using stochastic gradient descent leading to the update rule:

$$ \theta_{t+1} = \theta_t + \eta \nabla J(\pi_\theta), $$

where $\theta_t$ are the policy parameters at timestep $t$, $\nabla J(\pi_\theta)$ is the gradient of the objective function with respect to the parameters and $\eta$ is the step size.

3. Skills

A skill is a Temporally Extended Action (TEA) (Sutton et al., 1999) that is defined over a compact policy - a policy parameterized using LFA. The power of a skill is that it incorporates both generalization and temporal abstraction. Skills are a special case of Options and therefore inherit many of their useful theoretical properties (Sutton et al., 1999; Precup et al., 1998).

**Definition 1.** Skill: A skill $\zeta$ is a TEA that consists of the two-tuple $\zeta = (\sigma_\theta, p)$ where $\sigma_\theta : X \rightarrow \Delta_A$ is a compact, intra-skill policy (see Section 4.2) with parameters $\theta \in \mathbb{R}^d$ and $p : X \rightarrow [0, 1]$ is the termination probability distribution of the skill.

4. Skill Partitions and Intra-Skill Policy

A skill, by definition, performs a specialized task on a sub-region of a state space. We refer to these sub-regions as Skill Partitions (SPs) which are necessary for skills to specialize during the learning process. These partitions are unknown a-priori. In addition, once a skill is being executed, the agent needs to select actions from the skill’s intra-skill policy $\sigma_\theta$. We now introduce skill partitions enabling us to derive the probability of executing a skill, when presented with the current state. We also define an intra-skill policy which the agent executes once the relevant skill has been selected. These two elements are used to construct the hierarchical policy, defined in Section 6.1.

4.1. Skill Partitions

A general hyperplane in $\mathbb{R}^d$ is defined as $H = \{ y | a^T y = b \}$ where $a, y \in \mathbb{R}^d$, $a \neq 0$ and $b \in \mathbb{R}$. We now formally define a skill hyperplane.

**Definition 2.** Skill Hyperplane (SH): Let $\psi_{x,m} \in \mathbb{R}^d$ be a vector of features that depend on a state $x \in X$ and an MDP environment $m$. Let $\beta_i \in \mathbb{R}^d$ be a vector of hyperplane parameters. A skill hyperplane is defined as $\psi_{x,m}^T \beta_i = L$, where $L$ is a constant.

In this work, we interpret hyperplanes to mean that the intersection of skill hyperplane half spaces form sub-regions in the state space called Skill Partitions (SPs), defining where each skill is executed. Figure 1 contains two example skill hyperplanes $h_1, h_2$. Skill $\zeta_1$ is executed in the SP defined by the intersection of the negative half-space of $h_1$ and the positive half-space of $h_2$. The same argument applies for $\zeta_0, \zeta_2, \zeta_3$. From here on in, we will refer to skill $\zeta_i$ interchangeably with its index $i$.

Skill hyperplanes define SPs which enable us to derive the probability of executing a skill, given a state $x$ and MDP $m$. First, we need to be able to uniquely identify a skill. We define a binary vector $B = [b_1, b_2, \cdots, b_K] \in \{0, 1\}^K$ where $b_k$
Figure 1. Skill hyperplanes: The intersection of hyperplanes \{h_1, h_2\} form four skill partitions, each of which defines a skill’s execution region.

is a Bernoulli random variable and \(K\) is the number of skill hyperplanes. We define the skill index \(i\) as a sum of Bernoulli random variables \(b_k\) as seen in 5.

\[
i = \sum_{k=1}^{K} 2^{k-1} b_k.
\] (5)

In principle this setup defines \(2^K\) skills, but in practice, far fewer skills are typically used. Furthermore, the complexity of the SP is governed by the VC-dimension. We can now define the probability of executing skill \(i\) as a Bernoulli likelihood in Equation 6.

\[
p(i|x, m) = P \left[ i = \sum_{k=1}^{K} 2^{k-1} b_k \right] = \prod_{k=1}^{K} p_k(b_k = i_k|x, m)
\] (6)

Here, \(i_k \in \{0, 1\}\) is the value of the \(k^{th}\) bit of \(B\), \(x\) is the current state and \(m\) is a description of the MDP. The probability \(p_k(b_k = 1|x, m)\) and \(p_k(b_k = 0|x, m)\) are defined in Equations 7 and 8, respectively.

\[
p_k(b_k = 1|x, m) = \frac{1}{1 + \exp(-\alpha \psi^T_{x,m} \beta_k)} ,
\] (7)

\[
p_k(b_k = 0|x, m) = 1 - p_k(b_k = 1|x, m) .
\] (8)

We have made use of the logistic sigmoid function to ensure valid probabilities where \(\psi^T_{x,m} \beta_k\) is a skill hyperplane and \(\alpha > 0\) is a temperature parameter. The intuition here is that the \(k^{th}\) bit of a skill, \(b_k = 1\), if the skill hyperplane \(\psi^T_{x,m} \beta_k > 0\) meaning that the skill’s partition is in the positive half-space of the hyperplane. Similarly, \(b_k = 0\) if \(\psi^T_{x,m} \beta_k < 0\) corresponding to the negative half-space. Using skill 3 as an example with \(K = 2\) hyperplanes in Figure 1, we would define the Bernoulli likelihood of executing \(\zeta_3\) as \(p(i = 3|x, m) = p_1(b_1 = 1|x, m) \cdot p_2(b_2 = 1|x, m)\).

4.2. Intra-Skill Policy

Now that we have defined the probability of executing a skill based on its SP, we define the intra-skill policy \(\sigma_\theta\) for each skill. The Gibb’s distribution is a commonly used function to define policies in RL (Sutton et al., 1999). Therefore we define the intra-skill policy for skill \(i\), parameterized by \(\theta_i \in \mathbb{R}^d\) as

\[
\sigma_{\theta_i}(a|s) = \frac{\exp(\alpha \phi^T_{x,a} \theta_i)}{\sum_{b \in A} \exp(\alpha \phi^T_{x,b} \theta_i)}.
\] (9)
5. Generalized Trajectory

A generalized trajectory is necessary to derive policy gradient update rules with respect to the parameters $\Theta, \beta$ as will be shown in Section 6.3. A typical trajectory is usually defined as $\tau = (x_t, a_t, r_t, x_{t+1})^T_{t=0}$. For a generalized trajectory, our algorithm emits a class $i_t$ at each timestep $t \geq 1$, which denotes the skill that was executed. The generalized trajectory is defined as $g = (x_t, a_t, i_t, r_t, x_{t+1})^T_{t=0}$. The probability of a generalized trajectory, as an extension to Equation 3, is now:

$$P_{\Theta, \beta}(g) = P(x_0) \prod_{t=0}^{T} P(x_{t+1}|x_t, a_t) \cdot P_{\beta}(i_t|x_t, m) \sigma_{\theta_i}(a_t|x_t)$$

where $P_{\beta}(i_t|x_t, m)$ is the probability of a skill being executed, given the state $x_t \in X$ and environment $m$ at time $t \geq 1$; $\sigma_{\theta_i}(a_t|x_t)$ is the probability of executing action $a_t \in A$ at time $t \geq 1$ given that we are executing skill $i$. The generalized trajectory is now a function of two parameter vectors $\theta$ and $\beta$.


We have previously defined two important distributions $P_{\beta}(i_t|x_t, m)$ and $\sigma_{\theta_i}(a_t|x_t)$ respectively. These distributions have been incorporated into a generalized trajectory and we now have the necessary tools to develop the Adaptive Skills, Adaptive Partitions (ASAP) framework. This is the first framework that learns skills from scratch and simultaneously learns skill hyperplanes which define SPs. This framework is not necessarily task specific as it incorporates the environment $m$ into its hyperplane feature set, allowing for a multi-task setting if required. We first define the hierarchical policy, which we refer to as the ASAP policy, which inherently chooses the skill and intra-skill policy to execute.

6.1. ASAP Policy

Assume that we are given a probability distribution $\mu$ over MDPs with a $d$-dimensional state-action space and a $z$-dimensional vector describing each MDP. We define $\beta$ as a $(d + z) \times K$ matrix where each column $\beta_i$ represents a skill hyperplane, and $\Theta$ is a $(d \times 2^K)$ matrix where each column $\theta_j$ parameterizes an intra-skill policy. Using the previously defined distributions, we now define the ASAP policy.

**Definition 3. (ASAP Policy).** Given $K$ skill hyperplanes, a set of $2^K$ skills $\Sigma = \{\zeta_i|i = 1, \cdots 2^K\}$, a state space $x \in X$, a set of actions $a \in A$ and an MDP $m$ from a hypothesis space of MDPs, the ASAP policy is defined as,

$$\pi_{\Theta, \beta}(a|x, m) = \sum_{i=1}^{2^K} p_{\beta}(i|x, m) \sigma_{\theta_i}(a|x)$$

where $P_{\beta}(i|x, m)$ and $\sigma_{\theta_i}(a|x)$ are the distributions as defined in Equations 6 and 9 respectively.

This is a powerful description for a policy, which resembles a Bayesian approach, as the policy takes into account the uncertainty of the skills that are executing as well as the actions that each skill’s intra-skill policy chooses. We now define the ASAP objective with respect to the ASAP policy.

6.2. The ASAP Objective

We defined the policy with respect to a hypothesis space of $m$ MDPs. We now need to define an objective function which takes this hypothesis space into account. Since we assume that we are provided with a distribution $\mu : M \rightarrow [0, 1]$ over possible MDP models $m \in M$, with a $d$-dimensional state-action space, we can incorporate this into the ASAP objective function as follows:

$$\rho(\pi_{\Theta, \beta}) = \int \mu(m) J^{(m)}(\pi_{\Theta, \beta}) dm$$

where $J^{(m)}(\pi_{\Theta, \beta})$ is a function of the policy $\pi_{\Theta, \beta}$ evaluated on the MDP $m$. This integral takes into account the uncertainty in the MDP space and provides a principled way to optimize the policy.
where \( \pi_{\theta, \beta} \) is the ASAP policy and \( J^{(m)}(\pi_{\theta, \beta}) \) is the expected return for MDP \( m \) with respect to the ASAP policy. To simplify the notation, we group all of the parameters into a single parameter vector \( \Omega = [\text{vec}(\Theta), \text{vec}(\beta)] \). We define the expected reward for generalized trajectories as

\[
J(\pi_{\Omega}) = \int \mu(m) J^{(m)}(\pi_{\Omega}) \, dm \quad ,
\]

where \( R(g) \) is the reward obtained for a particular trajectory \( g \). This is a slight variation of the original policy gradient objective shown in Equation 2. We then insert Equation 12 into 11 and we get the ASAP objective function

\[
\rho(\pi_{\Omega}) = \int \mu(m) J^{(m)}(\pi_{\Omega}) \, dm \quad ,
\]

where \( J^{(m)}(\pi_{\Omega}) \) is the expected return for MDP \( m \) when following policy \( \pi_{\Omega} \). Next, we need to derive gradient update rules to learn the parameters of the optimal policy \( \pi_{\Omega}^* \) that maximizes this objective.

### 6.3. ASAP Gradients

To learn both intra-skill policy parameters matrix \( \Theta \) as well as the hyperplane parameters matrix \( \beta \) (and therefore implicitly the SPs), we derive an update rule for the policy gradient framework with generalized trajectories. The derivation is in the supplementary material. The first step involves calculating the gradient of the ASAP objective function yielding the ASAP gradient (Theorem 1).

**Theorem 1. (ASAP Gradient Theorem).** Suppose that the ASAP objective function is \( \rho(\pi_{\Omega}) = \int \mu(m) J^{(m)}(\pi_{\Omega}) \, dm \) where \( \mu(m) \) is a distribution over MDPs \( m \) and \( J^{(m)}(\pi_{\Omega}) \) is the expected return for MDP \( m \) whilst following policy \( \pi_{\Omega} \), then the gradient of this objective is:

\[
\nabla_{\Omega} \rho(\pi_{\Omega}) = \mathbb{E}_{\mu(m)} \left[ \mathbb{E} \sum_{i=0}^{H^{(m)}} \nabla_{\Omega} Z_{\Omega}^{(m)}(x_t, i_t, a_t) R^{(m)}(m) \right] \quad ,
\]

\[
Z_{\Omega}^{(m)}(x_t, i_t, a_t) = \log P_{\beta}(i_t|\pi_t, m) \sigma_{\theta_t}(a_t|\pi_t) \quad ,
\]

where \( H^{(m)} \) is the length of a trajectory for MDP \( m \); \( R^{(m)} = \sum_{i=0}^{H^{(m)}} \gamma^i r_i \) is the discounted cumulative reward for trajectory \( H^{(m)} \) \(^3\).

If we are able to derive \( \nabla_{\Omega} Z_{\Omega}^{(m)}(x_t, i_t, a_t) \), then we can estimate the gradient \( \nabla_{\Omega} \rho(\pi_{\Omega}) \). We will refer to \( Z_{\Omega}^{(m)} = Z_{\Omega}^{(m)}(x_t, i_t, a_t) \) where it is clear from context. It turns out that it is possible to derive this term as a result of the generalized trajectory. This yields the gradients \( \nabla_{\theta_t} Z_{\Omega}^{(m)} \) and \( \nabla_{\beta} Z_{\Omega}^{(m)} \) in Theorems 2 and 3 respectively. The derivations can be found in the supplementary material.

**Theorem 2. (\( \Theta \) Gradient Theorem).** Suppose that \( \Theta \) is a \((d \times 2^K)\) matrix where each column \( \theta_j \) parameterizes an intra-skill policy. Then the gradient \( \nabla_{\theta_t} Z_{\Omega}^{(m)} \) corresponding to the intra-skill parameters of the \( i \)th skill at time \( t \) is:

\[
\nabla_{\theta_t} Z_{\Omega}^{(m)} = \alpha \phi_{x_t, a_t} - \frac{\alpha \left( \sum_{b \in A} \phi_{x_t, b} \exp(\alpha \phi_{x_t, b}^T \Theta_i) \right)}{\left( \sum_{b \in A} \exp(\alpha \phi_{x_t, b}^T \Theta_i) \right)} \quad ,
\]

where \( \alpha > 0 \) is the temperature parameter and \( \phi_{x_t, a_t} \in \mathbb{R}^{d \times 2^K} \) is a feature vector of the current state \( x_t \in X_t \) and the current action \( a_t \in A_t \).

**Theorem 3. (\( \beta \) Gradient Theorem).** Suppose that \( \beta \) is a \((d + z) \times K\) matrix where each column \( \beta_k \) represents a hyperplane. Then the gradient \( \nabla_{\beta_k} Z_{\Omega}^{(m)} \) corresponding to parameters of the \( k \)th hyperplane is:

\(^3\)These expectations can easily be sampled (see supplementary material).
Using these gradient updates, we can then order all of the gradients into a vector \( \nabla \) \( \Omega \) to correspond to locations in the binary vector equal to 1 in Algorithm 1. The skills (\( \Theta \)) and SPs (\( \beta \)) are initially arbitrary and therefore form a misspecified model. Line 2 combines the skill and hyperplane parameters into a single parameter vector \( \Omega \). Lines 3-8 learns the skill and hyperplane parameters (and therefore implicitly the skill partitions). In line 4 a generalized trajectory is generated using the current ASAP policy. The gradient \( \nabla \Omega \rho(\pi_\Omega) \) is then estimated in line 5-6 from this trajectory and updates the parameters in line 7. This is repeated until the skill and hyperplane parameters have converged, thus correcting the misspecified model. Theorem 4 provides a convergence guarantee of ASAP to a local optimum (see supplementary material for the proof). Our proof is based on a slight variation of the original policy gradient convergence proof (Sutton et al., 2000).

\[ \nabla_{\beta_{k,b_k=1}} Z_{\Omega}^{(m)} = \frac{\alpha \psi_{(x,m)} \exp(-\alpha \psi_{(x,m)}^{T} \beta_k)}{1 + \exp(-\alpha \psi_{(x,m)}^{T} \beta_k)} \]

\[ \nabla_{\beta_{k,b_k=0}} Z_{\Omega}^{(m)} = -\alpha \psi_{(x,m)} + \alpha K(x,m,\beta_k) \]

\[ K(x,m,\beta_k) = \frac{\psi_{(x,m)} \exp(-\alpha \psi_{(x,m)}^{T} \beta_k)}{1 + \exp(-\alpha \psi_{(x,m)}^{T} \beta_k)} \]

where \( \alpha > 0 \) is the hyperplane temperature parameter, \( \psi_{(x,m)}^{T} \beta_k \) is the \( k^{th} \) skill hyperplane for MDP \( m \), \( \beta_{k,b_k=1} \) corresponds to locations in the binary vector equal to 1 and \( \beta_{k,b_k=0} \) corresponds to locations in the binary vector equal to 0.

Using these gradient updates, we can then order all of the gradients into a vector \( \nabla \Omega Z_{\Omega}^{(m)} = (\nabla_{\theta_\psi} Z_{\Omega}^{(m)} \ldots \nabla_{\theta_\psi} Z_{\Omega}^{(m)} \ldots \nabla_{\beta} Z_{\Omega}^{(m)} \ldots \nabla_{\beta} Z_{\Omega}^{(m)}) \) and update both the intra-skill policy parameters and hyperplane parameters for the given task (learning a skill set and SPs). Note that the updates occur on a single time scale. This is formally stated in the ASAP Algorithm.

7. ASAP Algorithm (ASAP)

We present the first algorithm for dynamically learning skills and SPs simultaneously. A schema for the algorithm is found in Algorithm 1. The skills (\( \Theta \) matrix) and SPs (\( \beta \) matrix) are initially arbitrary and therefore form a misspecified model. Line 2 combines the skill and hyperplane parameters into a single parameter vector \( \Omega \). Lines 3-8 learns the skill and hyperplane parameters (and therefore implicitly the skill partitions). In line 4 a generalized trajectory is generated using the current ASAP policy. The gradient \( \nabla \Omega \rho(\pi_\Omega) \) is then estimated in line 5-6 from this trajectory and updates the parameters in line 7. This is repeated until the skill and hyperplane parameters have converged, thus correcting the misspecified model. Theorem 4 provides a convergence guarantee of ASAP to a local optimum (see supplementary material for the proof). Our proof is based on a slight variation of the original policy gradient convergence proof (Sutton et al., 2000).

Algorithm 1 ASAP

Require: \( \phi_{x,a} \in \mathbb{R}^d \) \{state-action feature vector\}, \( \psi_{x,m} \in \mathbb{R}^{(d+z)} \) \{skill hyperplane feature vector\}, \( K \) \{The number of hyperplanes\}, \( \Theta \in \mathbb{R}^{d \times 2^K} \) \{An arbitrary skill matrix\}, \( \beta \in \mathbb{R}^{(d+z) \times K} \) \{An arbitrary skill hyperplane matrix\}, \( \mu(m) \) \{A distribution over MDP tasks\}

1: \( Z = ([d]2^K + (d+z)K) \) \{Define the number of parameters\}
2: \( \Omega = [vec(\Theta), vec(\beta)] \in \mathbb{R}^Z \)
3: repeat
4: Perform a trial (which may consist of multiple MDP tasks) and obtain \( x_{0:H}, i_{0:H}, a_{0:H}, r_{0:H}, m_{0:H} \) \{states, skills, actions, rewards, task-specific information respectively\}
5: \( \nabla \Omega \rho(\pi_\Omega) = \left\{ \sum_{m} \left( \sum_{i=0}^{T(m)} \nabla \Omega Z_{\Omega}^{(m)}(\Omega) R_{\Omega}^{(m)} \right) \right\} \) \{T is the task episode length\}
6: \( \Omega \rightarrow \Omega + \eta \nabla \Omega \rho(\pi_\Omega) \)
7: until parameters \( \Omega \) have converged
8: return \( \Omega \)

Theorem 4. Convergence of ASAP: Given an ASAP policy \( \pi(\Omega) \), an ASAP objective over MDP models \( \rho(\pi_\Omega) \) as well as the ASAP gradient update rules. If (1) the step-size \( \eta_k \) satisfies \( \lim_{k \rightarrow \infty} \eta_k = 0 \) and \( \sum_k \eta_k = \infty \); (2) The second derivative of the policy is bounded and we have bounded rewards. Then, the sequence \( \{\rho(\pi_\Omega,k)\}_{k=0}^{\infty} \) converges such that \( \lim_{k \rightarrow \infty} \frac{\partial \rho(\pi_\Omega,k)}{\partial \Omega} = 0 \).

8. Experiments

The experiments have been performed on five different domains: the Two Rooms (2R) domain, the Three rooms (3R) domain, RoboCup domains that include a one-on-one scenario between a striker and a goalkeeper (R1), a two-on-one
 Adaptive Skills Adaptive Partitions (ASAP)

Figure 2. Domains: (a) The Two Room domain, (b) Flipped Two-Room domain, (c) the Three Room domain (d) The RoboCup domain (with a varying number of defenders for R1, R2, R3)

scenario of a striker against a goalkeeper and a defender (R2), and a striker against two defenders and a goalkeeper (R3) (see supplementary material). In each experiment, ASAP is provided with a misspecified model; that is, a set of skills, hyperplane parameters (corresponding to SPs) and hierarchical policy that achieve degenerate, sub-optimal performance. ASAP corrects this misspecified model in each case to learn a set of near-optimal skills and SPs. For each experiment we implement ASAP using Actor-Critic Policy Gradient (AC-PG) as the learning algorithm.

In each of the experiments, our aim is to show some interesting and somewhat unexpected results of the ASAP framework. This framework has been tested on five continuous domains with the aim to provide intuition to the readers through some useful visualizations of the learned SPs and skills. We first show results of ASAP in learning SPs and skills simultaneously on the 2R, 3R and RoboCup domains. We also show, in the 2R domain, the potential of the algorithm to perform multi-task learning. In the 3R domain, we show a surprising result where ASAP is able to automatically discover reusable skills and defines the SPs accordingly. In the RoboCup R1 domain, we show that different local optimums can be attained resulting in different SP configurations. Each configuration solves the task in a near-optimal fashion. In the R2 domain, we show the learned SPs and how the defender affects the final shape of the SPs.

The Two-Room and Flipped Room Domains (2R): The 2R and Flipped 2R domains are shown in Figure 2(a) and (b) respectively. The agent (red ball) needs to reach the goal location (blue square) in the shortest amount of time. When the agent reaches the goal, it receives a large reward. There is a wall dividing the environment which creates two rooms. The state space is a 4-tuple consisting of the continuous $\langle x_{agent}, y_{agent} \rangle$ location of the agent and the $\langle x_{goal}, y_{goal} \rangle$ location of the center of the goal. The agent can move in each of the four cardinal directions. For each experiment involving the two room domains, a single hyperplane is learned (resulting in two SPs) with a linear feature vector representation for $\psi_{x,m}$.

Automated Hyperplane and Skill Learning: We applied ASAP to each of these domains and the agent learned intuitive SPs and skills as seen in Figure 3a and b. Each colored region corresponds to a SP. The white arrows have been superimposed onto the figures to indicate the skills learned for each SP. Since each intra-skill policy is a probability distribution over actions, each skill is unable to solve the entire task on its own. ASAP has taken this into account and has positioned the hyperplane accordingly such that the given skill representation can solve the task. Figure 3d shows the average reward obtained by ASAP on the 2R domain. This is compared to executing ASAP on the fixed initial misspecified partitioning as well as on a fixed approximately optimal partitioning for the domain. As seen in the figure, ASAP improves upon the initial misspecified partitioning and attains near-optimal performance.

Multitask Learning: A natural extension of ASAP is to that of multitask learning. That is, can the agent learn SPs and a skill set for a single task and then transfer some of that knowledge to solving a new task? We tested the multitask learning capabilities of ASAP on the 2R domain and Flipped 2R domain respectively. The optimal SP for the 2R domain (Figure 3a) is not suitable to solving the flipped 2R domain (Figure 3b) since the wall has inverted and the goal has changed its location effectively creating a completely different task. We first applied ASAP to the 2R domain and attained a near optimal average reward as seen in Figure 4a. It took approximately 35000 episodes to get near-optimal performance and resulted in the SPs and skill set shown in Figure 3f (top). Using the learned SPs and skills, ASAP was then able to adapt and learn a new set of SPs and skills to solve a different task in only 5000 episodes as seen in Figure 4a indicating that the parameters learned from the old task provided a good initialization for the new task. The knowledge transfer can be seen
Figure 3. The skills and hyperplanes learned using ASAP for the (a), (b) Two room and (c) Three room domains. The green and red regions indicate the learned SPs. The superimposed white arrows indicate the direction of the learned skills. (d) Average reward of the ASAP learned SPs and skill set compared to the approximately optimal SPs and skill set as well as the initial misspecified model. (e) The average reward for the Three Room domain. (f) The SPs and skills learned for the multi-task scenario.

Figure 4. Multitask Learning: (a) The average reward for learning two different tasks using ASAP. This is compared to an initial fixed SP and the approximately optimal SP. (b) Average reward for flipping the SPs and solving a new task without any additional learning. In Figure 3f as the SPs do not significantly change between tasks, however the skills are completely relearned.

We also wanted to see whether we could flip the SPs; that is, switch the sign of the hyperplane parameters learned in the 2R domain and see whether ASAP can solve the Flipped 2R domain without any additional learning. Due to the symmetry of the domains, ASAP was able to solve the new domain without any learning and attained near-optimal performance as seen in Figure 4b. This is an exciting result as many problems, especially navigation tasks, often possess symmetrical characteristics. This insight could dramatically reduce the sample complexity of these problems.

The Three-Room Domain (3R): The 3R domain, shown in Figure 2c is similar to the two-room domain regarding the goal, state-space, available actions and rewards. However, in this case, there are two walls, dividing the state space into three rooms. For this experiment, the hyperplane feature vector $\psi_{x,m}$ consists of a single fourier feature. The intra-skill policy is again represented as a probability distribution over actions. Since a single Fourier feature is utilized for the hyperplane representation, a wave-like partitioning was expected. The resulting learned hyperplane partitioning and skill
set are shown in Figure 3c. Using this partitioning ASAP achieved near optimal performance as seen in Figure 3e. This experiment shows an insightful and unexpected result.

**Reusable Skills**: By using a Fourier feature for the hyperplane representation, ASAP was able to not only learn the intra-skill policies and SPs, but it was also able to determine that the skill ‘move up and to the right’ needed to be reused in two different parts of the state space as seen in Figure 3c. ASAP therefore provides a natural framework for automatically creating reusable skills, which is an important characteristic if we wish to develop a truly general skill learning framework.

**RoboCup Domain**: The RoboCup 2D soccer simulation domain is a soccer platform used to advance Artificial Intelligence (AI) without the hassle of dealing with hardware challenges on real robots (Akiyama & Nakashima, 2014). It consists of a 2D soccer field as shown in Figure 2d with two opposing teams. The objective is to score more goals than your opponent and win the game. The code-base used for this experiment is that of HFO-Robocup4. This is based on the agent2D framework, which is frequently used by RoboCup teams. We utilized three RoboCup sub-domains R1, R2 and R3 as mentioned previously. The approximately optimal controller for each domain is the samplePlayer scoring controller provided by the agent2D framework. The goalkeeper and defender controllers used in these domains are from the standard agent2D framework. In these sub-domains, a striker (the agent) needs to learn to dribble the ball and try and score goals past the goalkeeper. For the R1 domain, the state space consists of the continuous locations of the striker \( \langle x_{\text{str}} \rangle \), the ball \( \langle x_{\text{ball}}, y_{\text{ball}} \rangle \), the goalkeeper \( \langle x_{\text{goalkeeper}}, y_{\text{goalkeeper}} \rangle \) and the constant goal location \( \langle x_{\text{goal}}, y_{\text{goal}} \rangle \). In the R2 domain, we have the addition of the defender’s location \( \langle x_{\text{defender}}, y_{\text{defender}} \rangle \) to the state space. In the R3 domain, we add the locations of two defenders. For the R1 domain, we tested both a linear and polynomial feature representation for the hyperplanes. For the R2 and R3 domains, we utilized a polynomial hyperplane feature representation. The striker has three actions which are (1) move to the ball (M), (2) move to the ball and dribble towards the goal (D) (3) move to the ball and shoot towards the goal (S). The striker gets small negative rewards for shooting from outside the box, or dribbling too close to the goalkeeper. The striker gets small positive rewards for dribbling outside the box and shooting when near the goal. This reward setup is consistent with logical football strategies. In addition, the agent gets a large positive reward for scoring and large negative rewards for kicking the ball out of bounds or losing it to an opponent. Previous works on RoboCup involve heavy engineering of the reward as RoboCup is a non-trivial domain and requires a highly customized reward function (Hausknecht & Stone, 2015; Bai et al., 2012).

**Offline Learning**: Due to the slow speed of the RoboCup simulator, trajectories were gathered offline for training ASAP using the approximately optimal samplePlayer controller. Policy gradient algorithms are usually designed for the online learning case, but ASAP nevertheless managed to learn optimal SPs and skill sets for the RoboCup domains (See SPs for R1 in Figure 5a). These results were consistently attained over five datasets. In each case, the agent learned that it should dribble (D) in the yellow SP and should shoot (S) in the semi circular SP near the goal.

**Different SP Optimas**: Since ASAP attains a locally optimal solution, it may sometimes learn different SPs. For the polynomial hyperplane feature representation, ASAP attained two different solutions as shown in Figures 5b(i) and b(ii) respectively. Both achieve near optimal performance compared to the approximately optimal hand-coded controller (see supplementary material). For the linear feature representation, the hyperplane and skill set in Figure 5b(iii) is obtained. This achieved near-optimal performance as seen in Figure 5d and performed better on average compared to the polynomial representation. The average ratio of goals scored over 1000 episodes is 79% for ASAP compared to 91% for the approximately optimal controller and 18% for ASAP evaluated on the misspecified model.

**SP Sensitivity**: In the R2 domain, an additional player (the defender) is added to the game. It is expected that the presence of the defender will affect the shape of the learned SPs. ASAP again learns intuitive SPs which consists of dribbling when outside the box and shooting when inside the box. However, the shape of the learned SPs change based on the predefined hyperplane feature vector \( \psi_{m,x} \). Figure 5c(i) shows the learned SPs when the location of the defender is not used as a hyperplane feature. When the \( x \) location of the defender is utilized, the ‘flatter’ SPs are learned in Figure 5c(ii). Using the \( y \) location of the defender as a hyperplane feature causes the hyperplane offset shown in Figure 5c(iii). This is due to the striker learning to dribble around the defender in order to score a goal as seen in Figure 5e. Finally, taking the \( \langle x, y \rangle \) location of the defender into account results in the ‘squashed’ SPs shown in Figure 5c(iv) clearly showing the sensitivity and adaptability of ASAP to dynamic factors in the environment.

In the case of R2, another surprising result is that ASAP had a better goal scoring ratio, on average, of 67% compared to

---

4https://github.com/mhauskn/HFO.git
Figure 5. The RoboCup Domains: (a) The learned SPs and skill set for the R1 domain. Note D refers to Dribble towards goal and S refers to shoot towards goal. (b) The learned SPs using a polynomial hyperplane (i),(ii) and linear hyperplane (iii) representation. (c) The learned SPs using a polynomial hyperplane representation without the defender’s location as a feature (i) and with the defender’s x location (ii), y location (iii), and ⟨x, y⟩ location as a feature (iv). (d) The average reward for the R1 domain compared to initial misspecified model and the approximately optimal controller for linear SPs. (e) The dribbling behavior of the striker when taking the defender’s y location into account.

the approximately optimal controller’s ratio of 62%. In the R3 domain, the average goal ratio over 1000 trials achieved with ASAP is 53% which outperforms the approximately optimal controller’s average goal ratio of 51%. A Video of ASAP’s performance in the RoboCup domains can be found in the supplementary material.

9. Discussion

We have presented the Adaptive Skills, Adaptive Partitions (ASAP) framework which is able to learn a near-optimal skill set, hierarchical policy and Skill Partition (SPs) simultaneously from an initially misspecified model. We derived the gradient update rules for both skill and skill hyperplane parameters and incorporated them into a policy gradient framework. This is possible due to our definition of a generalized trajectory. The algorithm provides the foundation for a truly general skill learning framework as it learns skills and skill partitions, has shown the potential to learn across multiple tasks as well as automatically determines the regions in state space where skills should be reused. These are also the core requirements for lifelong learning (Ammar et al., 2015; Thrun & Mitchell, 1995) in that an agent is able to learn skills, determine where the skills should be utilized and ultimately reuse the skills as required. An exciting extension of this work is to incorporate it into a Deep Reinforcement Learning framework, where both the skills and ASAP policy can be represented as Deep Q Networks (Mnih et al., 2015).

References


Mann, Timothy Arthur, Mankowitz, Daniel J, and Mannor, Shie. Learning when to switch between skills in a high dimensional domain. In Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.


