Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Learning Competitive Ensemble of Information-Constrained Primitives

présenté par:

Shagun Sodhani

a été évalué par un jury composé des personnes suivantes:

Abdelhakim Hafid,	président-rapporteur
Yoshua Bengio,	directeur de recherche
Pascal Vincent,	membre du jury

Mémoire accepté le: 23 juillet 2019

Résumé

Nous voulons développer des algorithmes d'apprentissage par renforcement qui permettent à l'agent apprenant d'obtenir une décomposition structurée de son comportement. L'apprentissage par renforcement hiérarchique fournit un mécanisme permettant de le faire en modularisant explicitement la politique en deux composants: un ensemble de sous-politiques de bas niveau (ou primitives) et une politique principale de haut niveau permettant de coordonner les primitives. Alors que les primitives ne doivent se spécialiser que dans une partie de l'espace d'états, la stratégie principale doit se spécialiser dans tout l'espace d'états, car elle décide du moment d'activer les primitives. Cela introduit un "goulot d'étranglement" dans lequel le succès de l'agent dépend du succès de la stratégie principale, ce qui en fait un point d'échec unique.

Nous proposons de supprimer cette limitation en utilisant un nouveau mécanisme selon lequel les sous-politiques peuvent décider elles-mêmes dans quelle partie de l'état elles souhaitent agir. Cette prise de décision décentralisée supprime la nécessité d'une politique principale paramétrée. Nous utilisons ce mécanisme pour former une politique composée d'un ensemble de primitives, mais ne nécessitant pas de stratégie principale pour choisir entre les primitives. Nous démontrons de manière expérimentale que cette architecture de politique améliore les politiques à la fois plates et hiérarchiques en termes de généralisation. Ce travail a été soumis à la conférence NeurIPS 2019 sous la forme d'un document intitulé **Apprentissage d'un ensemble concurrentiel de primitives à contraintes d'informations**.

Dans le premier chapitre, j'introduis des informations de base sur l'apprentissage par renforcement, l'apprentissage par renforcement hiérarchique, les goulots d'étranglement d'information, la compositionnalité et les réseaux de modules neuronaux, et explique en quoi le travail proposé au chapitre deux est lié à ces idées. Le chapitre deux décrit l'idée de former un ensemble de primitives. Je conclus ma thèse en discutant de quelques axes de recherche futurs pour les travaux décrits au chapitre deux.

Keywords: apprentissage par renforcement, apprentissage par renforcement hiérarchique, goulot d'étranglement de l'information, compositionnalité, réseaux modulaires

Summary

We want to develop reinforcement learning algorithms that enable the learning agent to obtain a structured decomposition of its behavior. Hierarchical Reinforcement Learning provides a mechanism for doing this by explicitly modularising the policy into two components — a set of low-level sub-policies (or primitives) and a high-level master policy to coordinate between the primitives. While the primitives have to specialize to only a part of the state space, the master policy has to specialize to the entire state space as it decides when to activate which primitives. This introduces a "bottleneck" where the success of the agent depends on the success of the master policy, thereby making it a single point of failure.

We propose to do away with this limitation by using a new mechanism where the sub-policies can decide for themselves in which part of the state they want to act. This decentralized decision making does away with the need for a parameterized master policy. We use this mechanism to train a policy that is composed of an ensemble of primitives but one that does not require a master policy to choose between the primitives. We experimentally demonstrate that this policy architecture improves over both flat and hierarchical policies in the terms of generalization. This work is under review at the NeurIPS 2019 Conference as a paper titled **Learning Competitive Ensemble of Information-Constrained Primitives**.

In Chapter One, I provide a background to Reinforcement Learning, Hierarchical Reinforcement Learning, Information Bottleneck, Compositionality, and Neural Module Networks and discuss how the proposed work in Chapter Two relates to these ideas. Chapter Two describes the idea of training an ensemble of primitives. I conclude the thesis by discussing some future research directions for the work described in Chapter Two.

Keywords: reinforcement learning, hierarchical reinforcement learning, information bottleneck, compositionality, modular networks

Table des matières

Table des figures

Liste des tableaux

List of Abbreviations

HRL	Hierarchical Reinforcement Learning
LSTM	Long-Short Term Memory
MDP	Markov Decision Process
NMN	Neural Module Network
PVF	Proto-Value Functions
RL	Reinforcement Learning
RNN	Recurrent Neural Networks

To my family!

Acknowledgments

I thank the almighty god for giving me a wonderful family, friends, mentors and collaborators who make me who I am.

I am grateful to Prof Yoshua Bengio and Prof Jian Tang who supervised me and supported me throughout the two years of the Masters program. They enabled my introduction to this amazing research community and gave me the freedom to pursue my research interests. Working with them, I learnt new things every day.

I am thankful to Sarath Chandar and Anirudh Goyal who supported me a lot during the last two years.

I acknowledge my friends and collaborators(in alphabetical order):

Bhairav Mehta, Carlos Eduardo Lassance, Chinnadurai Shankar, Disha Srivastava, Gunshi Gupta, Jayakumar Subramanian, Jonathon Binas, Kanika Madan, Khimya Khetarpal, Koustuv Sinha, Kushal Arora, Meng Qu, Nishanth, Olexa Bilaniuk, Philippe Lacaille, Prasanna Parthasarathi, Rim Assouel, Ritesh Kumar, Sai Krishna, Sai Rajeshwar, Simon Blackburn, Sébastien Lachapelle, Tristan Deleu, Vardaan Pahuja, Vikas Verma, Vikram Voleti, Vishal jain, Weiping Song, Zhaocheng Zhu

I thank Linda Pinthiere, Celine Begin, Julie Mongeau and all the other staff members at Mila who provided administrative help throughout my studies.

I am thankful to Balaji Krishnamurthy who always believed in me. I learned a lot of things from him - and not just about research.

I thank my undergraduate research advisor Dr Dhaval Patel for accepting me as his student and always encouraging me to pursue research.

1 Introduction

We want to develop reinforcement learning algorithms that can quickly adapt to new tasks by obtaining a structured decomposition of the learning agent's behavior. This problem is well studied under the paradigm of Hierarchical Reinforcement Learning. We review this body of literature in section 1.3 and describe the central idea here: Explicitly modularise the policy (learning agent's behavior) into two components — a set of low-level primitives (or sub-policies) and a high-level master policy to coordinate between the primitives. The rationale behind this decomposition is as follows: When we have a standard policy, the policy needs to specialize in the entire state space. In the case of Hierarchical Reinforcement Learning, the primitives have to specialize in only a part of the state space, thus easing the learning process. But this decomposition of policy into primitives and master policy may not be sufficient. Since the master policy decides when to activate which primitive, the master policy needs to specialize in the entire state space. This introduces a "bottleneck" where the success of the agent depends on the success of the master policy, making it a single point of failure.

In this work, we propose a new mechanism for training an ensemble of primitives without requiring a parameterized master policy. We introduce an information-theoretic objective that enables a competition mechanism. This mechanism leads to specialization of individual primitives to different parts of the state space and is used to select primitives in a decentralized manner. We review the information theory literature in section 1.4.

The general idea of training modular networks goes beyond Hierarchical Reinforcement Learning and is studied under the paradigm of Neural Modular Networks, which we review in section 1.9.

Our proposed approach enables addition, removal, and recombination of primitives across tasks. One manifestation of this idea is to train two separate set of primitives on two distinct tasks and then combine them for transferring on a third task. We demonstrate the superior transfer performance of our model to new tasks.

This idea relates to the broader concept of compositionality which we review in section 1.5.

1.1 Learning System

Thrun and Mitchell [1993] define a *learning* system as follows: “Given a task, training experience and a measure of performance (of the learning system), a system is said to learn if its performance on the task improves with training experience”. The definition of learning does not put any restrictions on the nature of the task, on the way training experience is provided (or leveraged for learning) or on the metrics used to measure the performance. A learning system could be provided with the training experience in the form of examples of a mapping between the inputs and the outputs and the task could be to learn this unknown mapping function. This is the well-known *supervised learning* paradigm. Another form of learning could be where the system interacts with its environment to collect training experience and instead of being given explicit supervision in terms of what input maps to what output, the system has to learn based on the “rewards” it gets from the environment. This is the well known *reinforcement learning paradigm* [Sutton and Barto] (summarized in section 1.2).

Deep Learning and Reinforcement Learning based techniques have transformed many application domains - image classification, image generation, machine translation, question answering, natural language inference, visual question answering, audio generation - to name a few. But the learning process is still unsatisfactory as the models trained using these techniques can not generalize as well as humans do. In this work, we look at this discrepancy from the lens of *compositionality* (section 1.5) and look at architecture designs like hierarchical reinforcement learning (section 1.3) and neural module networks (section 1.9) that can incorporate the notion of compositionality.

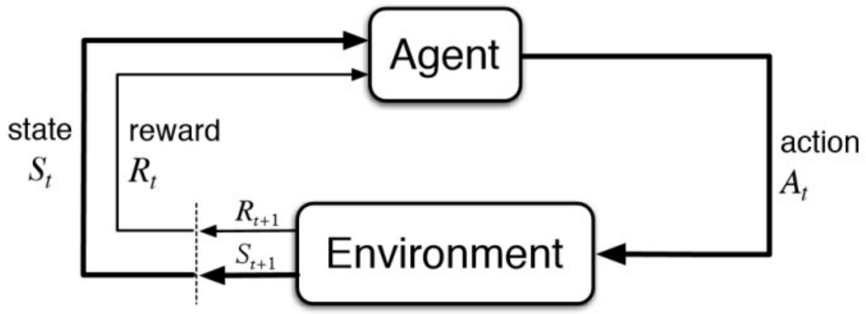
1.2 Reinforcement Learning

Consider a *learning agent* that is acting in an *environment* E. At time t, the agent is in some environmental *state* s_t and performs an *action* a_t in the *environment*. This leads to a change in the environmental *state* from s_t to s_{t+1}. This change can be experienced by the agent in two ways: the agent observes the *environment* and gets an *observation* o_t and the agent may receive a *reward* r_t from the environment. This sequence of interactions continues for the subsequent timesteps (Figure 1.1). The goal for the agent is to take *actions* which maximize the expected sum of *reward* that it will receive by interacting with the *environment*. Reinforcement learning is the learning paradigm that focuses on developing policy that the agent can follow to achieve this goal.

We formalize some of these terms:

1. *Learning agent* refers to the agent that interacts with the environment and for which we want to find (or learn) a good policy or strategy.
2. *Environment* E refers to everything that is outside of the learning agent.
3. *State* s_t refers to the "true" state of the environment at time t.
4. *State Space* $\mathcal{S}$ refers to the set of all the possible states that the agent can be in.
5. *Action* a_t refers to the action that the agent takes at time t.
6. *Action Space* $\mathcal{A}$ refers to the set of all the possible actions that the agent can take.
7. *Observation* o_t refers to the state of the environment that the agent can

Figure 1.1 – Elements of a reinforcement learning problem. Taken from Sutton and Barto



perceive at time t. In some special cases, where the agent can access the complete state of the environment, the observation is the same as the true state (denoted as *state*). This setup is referred to as the *fully-observed* setup. The other setup, where the agent can access only a part of the full state is far more common and is referred to as the *partially observed setup*.

8. *Reward* r_t refers to the signal that the agent gets from the *environment* at time t which indicates how good (or bad) its *actions* are.

In order to simplify the notation, we assume the *fully-observed* setup for the rest of this section so that $\forall t, s_t = o_t$

1.2.1 Model of the Environment

In the context of RL, *model* refers to the underlying *true* mechanisms of the environment. It is generally composed of two components:

1. **State Transition Operator** $\mathcal{P}$ which describes the probability that the environment transitions from state s_t to s_{t+1} when the agent takes an action a_t. The transition operator could be stochastic i.e. $s_{t+1} \sim \mathcal{P}(s_t, a_t)$ or deterministic i.e. $s_{t+1} = \mathcal{P}(s_t, a_t)$
2. **Reward Function** $\mathcal{R}$ describes the reward that the agent would get when it performs the action a_t in the state s_t. Like the transition operator, the reward function could be stochastic i.e. $r_t \sim \mathcal{R}(s_t, a_t)$ or deterministic i.e. $r_t = \mathcal{R}(s_t, a_t)$

1.2.2 Model-Based RL vs Model-Free RL

Consider the two scenarios:

1. The agent may have access to a model of the environment and use that model for deciding how to act via *planning*. This family of approach is called the *model-based* approach. Generally, the model of the environment is not available to the agent in which case the model can be learned using supervised or unsupervised learning approaches.
2. The agent could try to learn a good policy without having to build an explicit model of the environment. This family of approach is called the *model-free* approach.

Both approaches have their pros and cons. Recently, the model-free methods have shown many successes, such as learning to play Atari games with pixel observations [Mnih et al., 2015b, Mnih et al., 2016] and learning complex motion skills from high dimensional inputs [Schulman et al., 2015a,b]. However model-free approaches require huge amount of data to train and their high sample complexity remains a major criticism. On the other hand, model-based reinforcement learning approaches are known to be more sample efficient but have several caveats. For example, if the policy takes the learner to an unexplored state in the environment, the learner's model could make errors in estimating the environment dynamics, leading to sub-optimal behavior. This problem is referred to as the *model-bias problem* [Deisenroth and Rasmussen, 2011]. Hence the agent is only as good as the model.

1.2.3 Discount Factor and Expected Return

Consider an agent that starts in state s_1, takes an action a_1, receives a reward r_1 and transitions to the state s_2. The agent's interaction with the environment forms an *episode*: $s_0, a_0, r_0 ..., r_{T-1}, s_T$ where T is called the horizon. Recall that the agent wants to maximize the expected sum of rewards that it gets from the environment i.e. $\mathbb{E}[\sum r_t]$ but this formulation has some limitations.

All the rewards are given equal importance irrespective of whether they are obtained early in the episode or much later. It can be argued that if the agent can obtain the same reward at say the first timestep and the hundredth timestep, the reward obtained at first timestep should be more valuable than the reward obtained later as future rewards can be uncertain and hence do not provide the same value as the immediate rewards. Consider a scenario where we are given a choice — to take 100$ right now or take the 100$ one year later. Even though the face-value of the rewards remains the same, the two offers are not equal and we are more likely to take up the first offer. It is common to exponentially "discount" the future rewards with a factor called the *discount factor* γ. Then the effective value of the reward obtained at time t is given as $\gamma^t r_t$ and is referred to as the *discounted reward.*

Another benefit of using γ is that now we do not have to worry about the possibility of infinite length episodes. In the earlier formulation, the expected sum of reward could diverge if the episode length is not finite. In this new formulation, γ makes the expected sum of discounted rewards finite even when the episode length is infinite. Generally, $\gamma \in [0, 1]$. The lower is γ, the more important are the short-term rewards and more myopic (short-sighted) is the agent.

We denote the sum of discounted rewards as *returns* (equation 1.1) and the goal of the learning agent is to maximize the *returns*.

$$G_t = r_1 + \gamma r_2 + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \tag{1.1}$$

Here G_t denotes the returns at time t or the expected sum of discounted rewards from time t onwards.

1.2.4 Policy

Policy π refers to the strategy that the agent follows to decide which actions to take in the different states of the environment. It can be thought of as a mapping between a *state* and the *action* that should be executed in that *state*. Hence the policy π is generally a function of the state s and can be implemented using neural-network based function approximators. We can consider two cases:

1. **Stochastic Policy** where the action is sampled from the policy i.e. $a_t \sim \pi(s_t)$.
2. **Deterministic Policy** where the action is deterministically obtained from the policy i.e. $a_t = \pi(s_t)$.

1.2.5 Value Function

Value functions estimate how good it is for the agent to be in a state (or to take an action in a state). There are two types of value functions:

1. **state value function** $V(s)$ which estimates the *goodness* of a state s.
2. **action value function** $Q(s, a)$ which estimates the *goodness* of a state-action pair is or how good is it to take an action a in the state s.

The value of a state s, under the policy π can be defined as the expected return when the agent starts in state s and follows the policy π from that time onwards [Sutton and Barto]. Mathematically, the state value function can be written as follows:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{1.2}$$

Similarly, the action value (value of taking action a in a state s) under the policy π can be defined as the expected return when the agent starts in state s, takes action a and follows the policy π from that time onwards. Mathematically, the action value function can be written as follows:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{1.3}$$

The state value function and the action value function are related as follows:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s,a)\pi(a|s) \tag{1.4}$$

The difference between the action-value function and the state-value function is known as the **action-advantage function** (or just the advantage function). It is denoted by $A_\pi(s,a)$ and can be defined as:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{1.5}$$

There exists a class of techniques called *Value based methods* that aim to learn either the state value function V or the action value functions Q and use those value functions to obtain the policy π. If we have the optimal value functions (represented as V^* and Q^* respectively) then the optimal policy (represented as π^*) can be obtained through them as shown below:

$$\pi^* = argmax_\pi(V^*_\pi) \tag{1.6}$$

$$\pi^* = argmax_\pi(Q^*_\pi) \tag{1.7}$$

1.2.6 Markov Decision Process

Most of the RL problems can be formulated as a Markov Decision Process or MDP [Puterman, 1994].

A finite time Markov Decision Process $\mathcal{M}$ is generally defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. Here, $\mathcal{S}$ is the set of states, $\mathcal{A}$ the action space, $\mathcal{P}(s_{t+1}|s_t, a_t)$ the transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow R$ is the reward function and γ the discount factor. The *return* is defined to be the sum of discounted rewards $r(s_t, a_t)$ along an episode $\tau := (s_0, a_0, ..., s_{T-1}, a_{T-1}, s_T)$, where T refers to the effective horizon of the process. We note that this terminology is very similar to terminology when defining the different components of an RL task (section 1.2). In this MDP formulation, the goal of the learner is to find a policy π that maximizes the expected return. In practice, we parameterize the policy and represent it as π_θ where θ denotes the parameters of the policy π.

1.2.7 Policy Gradient Methods

Policy Gradient methods refer to the class of algorithms that aim to learn the policy π_θ directly. An objective function $J(\theta)$ is defined as follows:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \pi(a|s) \tag{1.8}$$

where $d^\pi(s)$ is the stationary distribution of Markov chain for the policy π.

It is not straightforward to compute $\nabla_\theta J(\theta)$ directly because $J(\theta)$ depends on π both directly (through the action sampling step) and indirectly through $d^\pi(s)$. Hence the Policy Gradient Theorem [Sutton and Barto] is used to compute $\nabla_\theta J(\theta)$ using the derivative of *log* function as:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[Q^\pi(s, a) \nabla_\theta ln(\pi_\theta(a|s))] \tag{1.9}$$

Since $\mathbb{E}_\pi[G_t|s_t, a_t] = Q^\pi(s_t, a_t)$, we can rewrite equation 1.9 as:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[G_t(s_t, a_t) \nabla_\theta ln(\pi_\theta(a|s))] \tag{1.10}$$

Thus the policy gradient theorem provides means to estimate the gradient for the objective function that can be used to update the policy π.

REINFORCE [Williams, 1992] is a Monte-Carlo policy gradient method that estimates the gradient for the policy by unrolling episodes in the environment. The expectation of the sampled gradient is an unbiased estimator of the actual gradient but often the variance of the sampled gradient is quite high. Subtracting a *baseline* value from the returns helps to reduce the variance while keeping the sampled gradient unbiased. One common and straightforward choice for the baseline is the running mean of rewards. The gradient of $J(\theta)$ can then be computed as:

$$\nabla J(\theta) = \mathbb{E}_{\pi}[\big(G(s_t, a_t) - b(s_t)\big)\nabla_{\theta} ln(\pi_{\theta}(a|s))] \tag{1.11}$$

where $b(s)$ is the value of the baseline in state s.

Alternatively, one could try to predict the value function and use that as the baseline for reducing the variance. This approach is know as the **Actor-Critic** methods [Konda and Tsitsiklis, 2000] where along with the policy (called *actor*), a value function (called as *critic*) is learned. The *actor* and the *critic* networks may share some parameters. Note that in general, the *critic* network could learn either the state-value or the action-value function though it is more common to learn the state-value function. In that case, we would replace the baseline $b(s)$ with the state value function $V(s)$.

One of the most prominent policy gradient algorithm is the **Asynchronous Advantage Actor-Critic** (or A3C algorithm) and its synchronous variant **Advantage Actor-Critic** (or A2C algorithm) [Mnih et al., 2016]. A3C algorithm uses one *critic* network and multiple *actor* networks which synchronize their weights regularly. In contrast, A2C uses one *critic* network and one *actor* network. The *actor* network interacts, in parallel, with different "copies" of the environment (for n steps) to collect a batch of trajectories. Then the *actor* and the *critic* networks are trained using these trajectories. In practice, A2C works faster than A3C though both the approaches are widely used [Jaderberg et al., 2016, Mirowski et al., 2016, Wang et al., 2016, Pathak et al., 2017].

Another popular idea, when training policy gradient algorithms, is to ensure that the update step cannot change the policy too much as this could de-stabilize the training of the policy. Consider the case where the agent takes an action in a state such that the reward function corresponding to that state-action pair is a Gaussian distribution with a low reward as the mean. But since the environment is stochastic, the agent got a high reward (an event which is not likely to happen again). In this case, the gradient could strongly bias the model to always perform the sub-optimal action in that state. This is similar to the case of supervised learning where the loss gradient is often clipped to make sure the weights do not change too much because of a single update. Schulman et al. [2015a] proposed the **Trust Region Policy Optimization** (TRPO) approach that implements this constraint by regularizing the KL divergence corresponding to the change in the policy to ensure that the new policy does not change too much compared to the old policy.

Specifically, a *trust region constraint* is applied as follows:

$$\mathbb{E}_{\pi_{\theta_{old}}}\Big[D_{KL}(\pi_{\theta_{old}}||\pi_\theta)\Big] \leq \delta \tag{1.12}$$

where $\pi_{\theta_{old}}$ is the old policy (before update), π_θ is the new policy (after update) and δ is a parameter controlling how much is the policy allowed to change.

One downside of TRPO is that it is quite complex to implement and use in practice. Schulman et al. [2017] proposed the **Proximal Policy Optimization** algorithm (PPO) which uses a simpler mechanism for regularizing the change in the policy by using a clipped surrogate objective. Specifically, consider the ratio of the old policy and the new policy as

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}(a|s)}} \tag{1.13}$$

The ratio $r(\theta)$ is clipped to lie in the region $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter. PPO achieves performance similar to TRPO but is much easier to implement in practice.

1.3 Hierarchical Reinforcement Learning

Consider a routine task like making coffee. The instructions for making coffee can be written down as (i) Pour the coffee from the machine (ii) Add sugar (iii) Add milk (iv) Stir and serve. Alternatively, one could break down each instruction into smaller sub-instructions: (i) Check if the coffee machine has coffee beans or not. (ii) Turn on the machine and press the button to brew coffee. (iii) Let the coffee pour into the cup and so on. An extreme case would be where the action sequence is described in terms of muscle movements: (i) Contract your hand muscle to lift the hand. (ii) Extend the arm muscle to reach the coffee machine and so on.

The idea is that when performing any task, there are multiple levels of temporal abstractions (or granularity) at which an agent can operate. Standard RL approaches operate only at one level of abstractions which leads to issues like limited scalability, low sample efficiency, lack of generalization, difficulty in performing long-term credit assignment and poor transfer to new tasks. Since these approaches look at the problem from just one level of abstraction, they are also called “flat” approaches.

On the other hand, humans complete complex tasks by first breaking them into subtasks and then completing each of those sub-tasks i.e. they operate at multiple levels of abstractions or hierarchy. This enables them to quickly adapt to new tasks and scenarios. For example, once a person knows how to make coffee, they can quickly learn how to make tea because most of the “high-level” instructions (like add sugar, add milk, etc) remains the same and they need to learn fewer new things (like how to prepare tea water). Contrast this situation with the scenario where the instructions were defined in terms of low-level muscle movements and not high-level sub-goals. In that case, the learner would not be able to transfer much knowledge as it would not have the notion of grouping sequence of actions into a temporal abstraction.

Another way to think about the notion of temporal abstractions is in the terms of “skills” where a skill represents a sequence of actions that can perform some subtask. When the learner is faced with a new task, it just needs to learn how to compose skills and does not need to learn the skills from scratch. The transfer of skills from one task to another reduces the amount of information that the learner needs to acquire thus improving generalization and reducing the sample complexity. For the transfer to a new task to be effective, the learner stills need to ensure that it learns the optimal behavioral hierarchy [Solway et al., 2014] so that it can learn to compose them and solve more complex tasks. The notion of compositionality is discussed further in section 1.5.

Hierarchical Reinforcement Learning (HRL) is the learning paradigm that attempts to address the limitations of flat RL approaches by learning to operate at the different levels of temporal abstractions at once. HRL provides benefits like better sample efficiency, structured exploration, easier long-term credit assignment, more effective generalization, and better transfer performance.

1.3.1 Options

One of the most well-known HRL framework is the *Options* framework that was introduced by Sutton et al. [1998b, 1999b], Precup [2000] and extended by [Stolle and Precup, 2002, Bacon et al., 2017, Machado et al., 2017a, Fox et al., 2017a, Machado et al., 2017a, Liu et al., 2017] among others.

An option ω is a triple of the form $<I_\omega, \pi_\omega, \beta_\omega>$ where:

1. I_ω refers to the initiation set of the option i.e. the states in which the option can be initiated. $I_\omega \subset \mathcal{S}$.
2. π_ω refers to the policy of the option. It is also called the *intra-option* policy. $\pi_\omega : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$
3. β_ω refers to the termination condition of the option. This gives the probability of the option terminating in the current state. $\beta_\omega : \mathcal{S} \rightarrow [0, 1]$

The more recent descriptions of options tend to drop the reference to the *initiation set* [Sutton and Barto]. An option can be thought of as an action that extends over multiple time-steps. At the time $t = 0$, when the agent starts interacting in the environment, it chooses the option that it wants to use. Once an option has been selected, the policy corresponding to that option is followed until the option has been terminated. At each timestep, the agent decides if it wants to terminate the option using the termination probability β_ω corresponding to the option. If an option is terminated, the agent chooses a new option at the next timestep using the policy π_Ω where Ω represents the set of all the options i.e. $\forall \omega, \omega \in \Omega$

One major strength of the options framework is that it is defined in a way that makes options interchangeable with the one-time step action. In fact, actions can be thought of as options that are active for just one timestep. Sutton et al. [1999b] and Precup [2000] show that defining options over a MDP makes it a Semi-Markov Decision Process (SMDP) [Puterman, 1994]. Hence the notion of the reward function, transition function, state value function, action, value function, advantage function, etc can be easily extended in the options framework.

What is not so straightforward is coming up with the options, as unlike actions, options are not defined as part of the RL problem. Bacon et al. [2017] proposes a *call-and-return option* execution model called the **option-critic** framework where the different components are parameterized using function approximators. π_Ω is denoted as $\pi_{\Omega,\phi}$, the intra-option policy π_ω is denoted as $\pi_{\omega,\theta}$ and the termination function β_ω is denoted as $\beta_{\omega,\nu}$. Here ϕ, θ, ν represents the parameters of the policy (between options), intra-option policy and the termination function respectively. Once an option ω has been chosen, it is used until it has been terminated in which case a new option is selected using the policy π_Ω. Figure 1.2 depicts the diagram of the option-critic architecture.

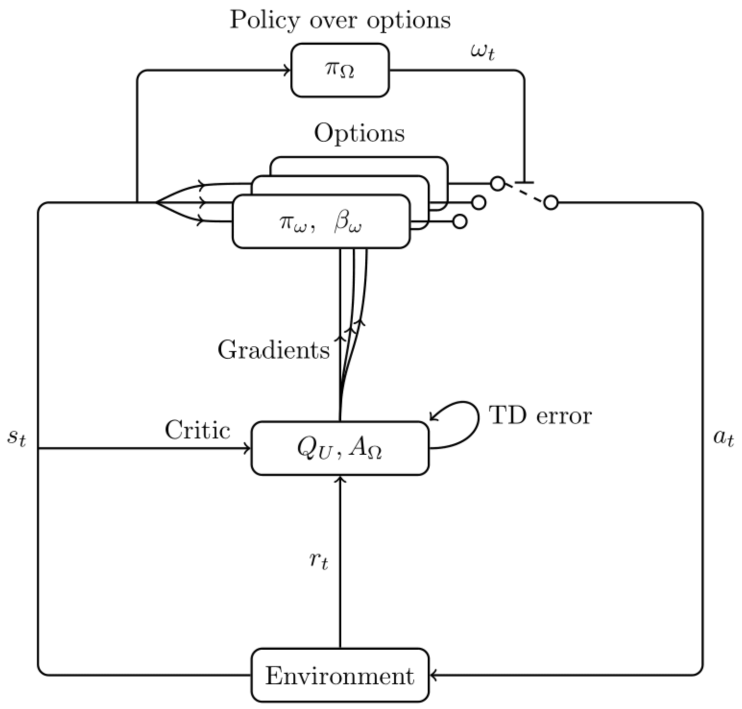


Figure 1.2 – The option-critic architecture consists of a set of options, a policy over them and a critic. Gradients can be derived from the critic for both the intra-option policies and termination functions.The execution model is suggested pictorially by a *switch* $\perp$ over the *contacts* $-\!\circ$. Switching can only take place when a termination event is encountered. Q_Ω denotes the option-value function while A_Ω denotes the option-advantage function. The image is taken from Bacon et al. [2017]

The policy over the options π_Ω is trained to maximize the expected returns $\mathbb{E}\Big[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)|s_0, w_0\Big]$

Bacon et al. [2017] prove two important theorems that enable them to train the options in an end-to-end manner. The first is the **intra-option policy gradient theorem** which enables them to compute the gradient of the expected discounted returns with respect to the parameters θ of the intra-option policy $\pi_{\omega,\theta}$ for any option ω. The second theorem is the **termination gradient theorem** that enables them to compute the gradient of the expected discounted returns with respect to the parameters ν of the termination condition $\beta_{\omega,\nu}$ for any option ω. They test the option-critic architecture for the Four-rooms task, Pinball domain and the Arcade Learning Environment [Bellemare et al., 2013] and verify that the approach of learning the options in an end-to-end manner is useful in both simple and complex tasks.

Given an MDP with adjacency matrix A and diagonal matrix D, the graph Laplacian L can be defined as $D - A$. Proto-value functions (PVFs) [Mahadevan, 2005, Mahadevan and Maggioni, 2007] are the eigenvectors obtained after the eigen-decomposition of L and are commonly used to learn representations in RL. Machado et al. [2017a] proposed an approach to discover task-independent options by showing that the proto-value functions implicitly defined options. They introduced the concept of *eigenpurposes* which are the intrinsic reward functions of the PVFs. Each eigenpurpose can be used to obtain an *eigenbehavior* which is the optimal policy that the agent should follow to maximize the reward corresponding to the given eigenpurpose. An option is associated with each eigenpurpose and the corresponding eigenbehvaior becomes the policy of that option. These options are referred to as the *eigenoptions*.

Machado et al. [2017b] builds on the idea of eigenoptions and extends Machado et al. [2017a]'s work for (i) environments where the state space cannot be enumerated as such and hence needs function approximation to learn the state representation and (ii) environments with stochastic transitions. The idea is to use the fact that the PVFs and the successor representations [Dayan, 1993] are equivalent, up to a constant [Stachenfeld et al., 2017]. First, a neural network is trained to approximate the successor representation from the raw pixels. Then this learned successor representation is used in place of the PVFs.

Fox et al. [2017b] proposed a policy gradient algorithm called *Discovery of Deep Options* (DDO) that can discover parameterized options from a given set of trajectories (or policy rollouts). The trajectories need not come from an expert policy but they should not be completely random either. The idea is to apply imitation learning techniques to the trajectory data to learn policies at multiple levels of hierarchy. This approach is called the Hierarchical Behaviour Cloning (HBC). The approach is recursive in nature and can be applied multiple times to discover multiple levels of hierarchy. The discovered options are helpful in accelerating the training of DQN agents on Atari tasks.

1.3.2 Feudal Learning

Several other HRL architectures follow a common pattern:

1. There are a number of low-level policies (called the *primitives* or *primitive-*

policies) whose action space is the same as the action space of the task i.e. these policies directly act in the environment.

2. There is a high-level policy (called the *meta-policy* or the *master policy* or the *controller*) whose task is to select which primitive acts in the environment in the current state thereby coordinating between the primitives. The action space for the master policy is the set of primitives. The idea is that the master policy would break down a given task into sub-tasks that the individual primitive policies can then execute.

In this particular case, we considered only two levels of hierarchy though in general there could be multiple levels of hierarchy. This kind of architecture is quite similar to the modular neural network architecture as well (section 1.9). We now discuss some prominent approaches which follow this architecture design.

Dayan and Hinton [1993] introduced the concept of *Feudal Reinforcement Learning* for solving a RL task. The idea is the following: Consider a hierarchy where there is a super-manager which controls several managers. The super-manager has a task to accomplish and it creates some subtasks which are assigned to its managers. Each manager controls many sub-managers and is controlled only by its super-manager. The manager receives a task from the super-manager, creates various subtasks to solve it and assigns them to its sub-managers. Each sub-manager controls another level of sub-sub managers and will be controlled only by its own manager. At the lowest level of the hierarchy are the workers who can act in the environment. In such a setup, each manager in the intermediate layers gets a "task" from the manager directly above it and it delegates a new "task" to the managers directly below it. The manager on the top of the hierarchy gets observations from the environment while the workers in the last layer act in the environment. This is what a feudal architecture looks like - agents are arranged in a hierarchy, with each agent getting some task information from the agent immediately above it in the hierarchy and each agent passing on some task information to the agents directly below it. Note that only the task information flows from the manager to the sub-manager i.e. an agent is told what it needs to do but not how to do it. These agents together make up the RL system and are not separate RL systems. This organization schema is similar to how the medieval feudal society was organized hence the name *feudal*.

There are two important aspects of Feudal RL:

1. **Reward Hiding**: Each agent's reward depends on how well it completes the task assigned to it by the agent directly above it - irrespective of whether fulfilling that sub-goal helps to complete the actual task or not.
2. **Information Hiding**: Each agent has access to information at a particular level of temporal abstraction only. For example, the agent does not have access to the task that was assigned to the agent directly above it. It does not even have access to the sub-goal that the agent directly below it assigns to other agents.

Sasha Vezhnevets et al. [2017] built on the idea of feudal RL and proposed FeUdal Networks which is a fully-differentiable neural network architecture with two levels of hierarchy - *Manager* and *Worker*. The manager receives an observation x_t from the environment and encodes it in as a latent representation z_t to produce a goal representation g_t. The goal is a direction in which the worker should go in the latent space. The worker takes as input an intermediate representation of the observation z_t (that it shares with the master), the goal g_t and its own internal hidden state. Both the worker and the manager use RNNs to maintain the history of the state though the worker uses an LSTM while the manager uses a dilated LSTM. The manager's goals g_t are trained using *transition policy gradient* [Sasha Vezhnevets et al., 2017] to ensure that these goals correspond to meaningful semantics (or meaningful action sequences) and are not just latent variables in the model. The worker is trained using an intrinsic reward that encourages it to follow the goal direction provided by the manager.

Figure 1.3 shows a schematic illustration of the FeUdal Network and how the gradients flow through the network. The input to the system is the observation x_t which is encoded into z_t using a convolutional network called $f_{percept}$. z_t is shared between the master and the worker. The master encodes z_t into its latent state space to obtain s_t which is fed as input to its dilated LSTM to produce the goal g_t. The worker projects both the goal g_t and the encoding of the observation z_t into a k dimensional vector space and use the product of the two representations to sample the actions. The worker is trained with the A3C algorithm [Mnih et al., 2016] using the intrinsic reward it gets for following the goal direction from the master and the master is trained using the transition policy gradient. Note that no gradient flows between the master and the worker through the goal.

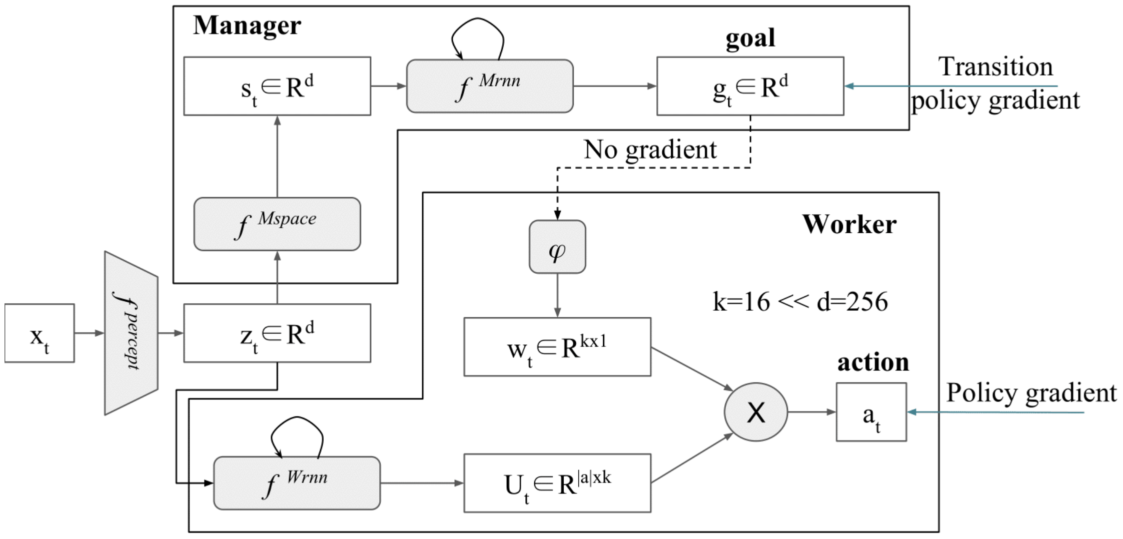


Figure 1.3 – The schematic illustration of FeUdal Network (taken from Sasha Vezhnevets et al. [2017])

1.3.3 Meta-Learning Shared Hierarchies

Frans et al. [2017] considers the problem of training an RL agent for a given distribution of tasks $P(M)$ such that the agent can quickly adapt to the new tasks. While it is intuitive that the agent's policy (or parts of the policy) should be shared across tasks, using a single policy to solve all the tasks could be ineffective. They propose to break down the policy into two components - (i) a task-agnostic component that is shared across all the tasks and denoted by the parameters ϕ (ii) a task specific component that is learnt separately for all the tasks and denoted by the parameters θ.

During training, a new task is sampled from the given distribution over tasks. For each task, a new set of task-specific parameters θ are initialized and the previously-trained task-agnostic parameters ϕ are reused.

The meta-learning objective for the task-agnostic components is designed to maximize the agent's expected returns over the distribution of the tasks ie:

$$maximize_{\phi}\mathbb{E}_{M\sim P(M)}[G_M] \tag{1.14}$$

where G_M denotes the returns from task M.

The setup is cast as a HRL problem with the task-agnostic component ϕ implemented as a set of K sub-policies $\phi_{i=1}^{K}$ and the task-specific component θ implemented as a master policy whose task is to switch between the K sub-policies. The master policy makes a selection every N timesteps i.e. once a subpolicy ϕ_i is selected, it is executed for N steps. Such a restriction does not exist in the options framework and an option can terminate at any point in time.

When training on a new task, a new master policy θ is randomly initialised. A two phase training process follows:

1. **Warmup Phase**: In this phase, only the master policy is trained to ensure that it can learn to use the primitives learnt so far. The master policy selects a sub-policy which then executes for N timesteps. From the master policy's perspective, the effect of unrolling a sub-policy for N timesteps is part of the dynamics of the environment and these N-step transitions (for the sub-policy) are just 1 transition for the master policy. The master policy can be thought of as operating in a modified MDP with a horizon that is $1/N$ times as long as the horizon of the actual MDP (in which the sub-policies operate). Any RL algorithm can be used to train the master policy in this modified MDP
2. **Joint Training Phase**: In this phase, both the master policy and the sub-policies are updated together. The updates of the master policy are performed as described in the warmup phase. From the sub-policy's perspective, the master policy is a part of the environment. Given a sequence of transitions in the environment, a sub-policy is updated using only that part of the trajectory for which it was active.

Figure 1.4 shows how the updates happen for the master policy and the sub-policies (assuming $N = 3$). The idea behind the two phase learning process is to learn sub-policies that can easily generalize across multiple tasks so that it is easy to learn a master policy for a given set of pre-trained sub-policies (as done in the warmup phase). The authors show that they are able to discover meaningful motor primitives (like walking in a particular direction) for robotic locomotion tasks and also achieve superior transfer performance for sparse-reward setups where the flat policies completely fail.

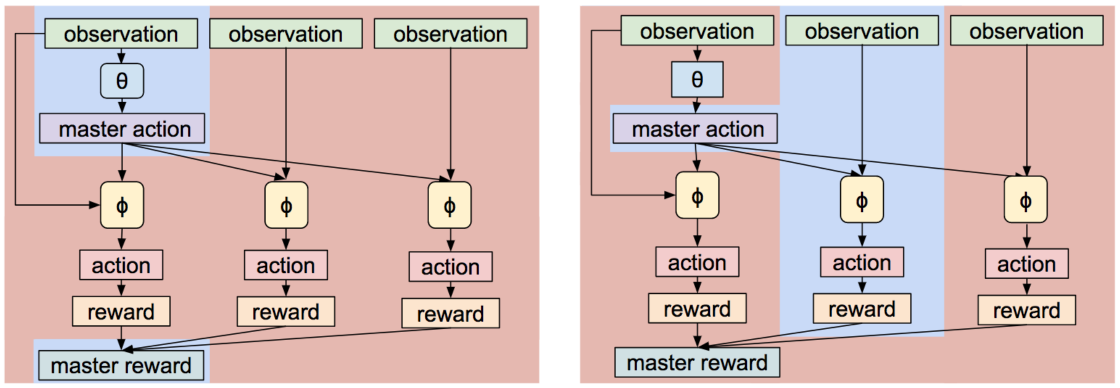


Figure 1.4 – Unrolled structure for a master policy action lasting for $N = 3$ timesteps. Left: When training the master policy, the update only depends on the master policy's action and total reward (blue region), treating the individual actions and rewards as part of the environment transition (red region). Right: When training sub-policies, the update considers the master policy's action as part of the observation (blue region), ignoring actions in other timesteps (red region). Figure taken from Frans et al. [2017].

1.4 Information Theory

In this section, we review some of the preliminary terms and concepts in information theory. We use the concept of information bottleneck to design an information-theoretic objective which leads to the specialization of individual primitives (to distinct regions in the state space) and enables a competition mechanism to select the active primitives in a decentralized manner.

1.4.1 Preliminary Terms

Entropy The entropy of a random variable X, with the probability distribution p, is a measure of uncertainty in the value of X. It is denoted by $H(X)$. We consider two cases:

Discrete Case: X can take one of the k discrete values $\{x_1, \cdots x_k\}$

$$H(X) = -\sum_{i=1}^{k} p(x) log_2 p(x). \tag{1.15}$$

Continuous Case: If X is a continuous variable, the entropy is also refereed to as the *differential entropy* and given as

$$H(X) = -\int_X p(x)log_2 p(x)dx. \tag{1.16}$$

From this point onward, we assume all the random variables to be continuous variables to simplify the notations.

Conditional Entropy The entropy of a random variable Y, conditioned on another random variable X is referred to as the conditional entropy. It is a measure of uncertainty in the value of Y given that the value of X is known. It is denoted by $H(Y|X)$ and can be expressed as follows:

$$H(Y|X) = -\int_{X,Y} p(x,y)log_2 p(y|x)dx. \tag{1.17}$$

Kullback–Leibler Divergence The Kullback–Leibler divergence (KL divergence) is a metric to measure the dissimilarity between two probability distributions p and q. It is denoted as $D_{KL}(p||q)$, is also referred to as the *relative entropy* and can be expressed as follows:

$$D_{KL}(p||q) = \int_X p(x)log_2\Big(\frac{p(x)}{q(x)}\Big)dx. \tag{1.18}$$

KL divergence $D_{KL}(p||q)$ is convex in the pair (p, q). Using Jensen's inequality, it can be shown that $D_{KL}(p||q) \geq 0$ and $= 0$ iff $p = q$. This property is called the *information inequality*.

Mutual Information The Mutual Information (MI) is a metric to measure the dissimilarity between the joint distribution $p(X, Y)$ and the factored distributions $p(X)p(Y)$. It is denoted as $I(X, Y)$ and can be expressed as follows:

$$I(X,Y) = \int_{X,Y} p(X,Y)log_2\Big(\frac{p(X,Y)}{p(X)p(Y)}\Big)dx. \tag{1.19}$$

The mutual information can also be written as the KL divergence between the joint distribution and the factored distributions:

$$I(X,Y) = D_{KL}(p(X,Y)||p(X)p(Y)). \tag{1.20}$$

Using the definition of conditional entropy (equation 1.17), the mutual information can be re-written as:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{1.21}$$

Hence, the mutual information between X and Y can be interpreted as a measure of reduction in the uncertainty about X after observing Y or as a measure of how much information does Y have about X (and vice-versa).

1.4.2 Information Bottleneck

Tishby et al. [2000] introduced the concept of *Information Bottleneck* as an *information-theoretic* way of extracting the relevant information that an input variable X contains about an output variable Y. They define the *relevant* information as being the mutual information between X and Y. They consider an intermediate "bottleneck" variable $\widetilde{X}$ which retains only that information in X that is useful to produce Y (or is relevant to Y), thus creating a bottleneck for the information in X. The value of $\widetilde{X}$ can be learnt by maximizing $I(\widetilde{X}, Y)$ such that $I(\widetilde{X}, X) < I_c$ where I_c is the information constraint. In the absence of the information constraint, $\widetilde{X}$ could encode all the information in X, leading to a trivial solution. The resulting constrained optimization equation can be solved by minimizing the following Lagrangian:

$$\mathcal{L}[p(\widetilde{x}|x)] = I(\widetilde{X}, X) - \beta I(\widetilde{X}, Y) \tag{1.22}$$

where β is the Lagrangian multiplier and can be seen as the parameter that balances between retaining information *relevant* to the output Y vs encoding more information from the input X.

Tishby and Zaslavsky [2015] applied the information bottleneck principle to deep neural networks and showed that it can be used to obtain the information-theoretic limits in the deep networks.

Consider an intermediate layer of a deep neural network (parameterized by θ) which takes as input X and produces a stochastic variable Z. We want to learn the parameters θ such that the mutual information between Z and and the output Y is maximized while that between Z and X is minimized. The objective function, to be minimized, can then be written as:

$$J_{IB}(\theta) = I(Z, X; \theta) - \beta I(Z, Y; \theta) \tag{1.23}$$

Here β plays the role of a hyperparameter.

In general, computing the mutual information is difficult, thus limiting the applicability of this approach.

Alemi et al. [2016] proposed to get around this problem by using a variational approximation to obtain a lower bound on the information bottleneck objective. This approach is called as the *Deep Variational Information Bottleneck* or Deep VIB. Using variational approximation for $p(y|z)$ and $p(z)$, they obtain an approximation to equation 1.23 as shown below:

$$J_{IB} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[-\log q(y_n|f(x_n,\epsilon))\right] + \beta \mathrm{KL}\left[p(Z|x_n), r(Z)\right]. \tag{1.24}$$

$p(x, y)$ is approximated using the empirical distribution $p(x, y) = \frac{1}{N}\sum_{n=1}^{N} \delta_{x_n}(x)\delta_{y_n}(y)$, $q(y|z)$ is the variational approximation for $p(y|z)$, $r(Z)$ is the variational approximation for $p(z)$, and f is the feed-forward network which encodes x to produce z ie $p(z|x) = \mathcal{N}(z|f^{\mu}(x), f^{\Sigma}(x))$.

1.5 Compositionality

Our proposed model uses many ideas and concepts from the literature on *compositionality.* In this section, we review the notion of compositionality, why would compositionality be a desirable property and look at some related concepts and prior work.

1.5.1 What is compositionality

Compositionality is defined as the property that "the meaning of a whole can be expressed as a function of the meaning of its parts" [Keenan and Faltz, 1987] or that the "meaning of a complex expression can be determined by its structure and the meaning of its constituents" [Szabó, 2004]. In the terms of language and semantics, compositionality can be defined as the existence of a homomorphism between the expressions of a language and the meanings of those expressions [Montague, 1970]. For example, the rules of propositional logic are compositional as the truth value of any propositional statement is derived in terms of the value of different sub-statements.

1.5.2 Why do we want compositional models

Most of the recent advancements and success in the field of deep learning and reinforcement learning are targeted at solving well-defined tasks under somewhat controlled scenarios. Though the learning systems learn to solve these tasks, sometimes surpassing human performance, the learning process itself is not very satisfactory. For example, these algorithms are known to be extremely data hungry and sample inefficient and their performance often deteriorates as the task or the data distribution changes. In contrast, human beings are far superior learners. They can quickly adapt to new tasks (or data distributions) and can easily transfer knowledge from remotely related but essentially distinct experiences. Thus it is safe to conclude that our current learning systems are very different from human beings in terms of what they learn and how they learn it.

Fodor and Pylyshyn [1988] proposes a theory to account for this discrepancy. The theory uses three features of cognition — productivity, compositionality, and inferential coherence — to compare the cognition models in humans with the connectionist models in the learning systems. They invoke these three cognitive features to argue that connectionist models lack some important features of human cognition and that combinatorial structure is useful in any rich representation system (which are important constituents of a learning system). The arguments are as follows:

1. **Productivity Argument** - Providing the learning system with a combinatorial structure would enable the system to (in-principle) encode infinitely-many combinations of the input representations. While the input representations themselves could be simplistic, they could be composed together in complex ways to learn more powerful representations. Thus compositionality can be seen as a way for increasing the expressivity or "productivity" of the learning system [Chomsky, 1964, 1968]. We can also think of the productivity argument in the following way: Assume that we have a learner that can solve a novel task. Since the learner has not encountered this task before, the only way it could have understood (and hence solved) the task

is if the knowledge to solve the task can be obtained by just knowing what components make up the task and how these components are related [Frege, 1914].

2. **Compositionality Argument** - It is based on the hypothesis that the world is inherently compositional, that is the dynamics in the world are governed by some low-level independent mechanisms [Parascandolo et al., 2017a]. Hence the new tasks (or experiences) are actually compositions of parts of previously seen tasks (or experiences). For example, spoken words are basically combinations of phonemes and sentences are the composition of the words. If a learning agent has already learned these low-level mechanisms then it can perform a new task without training (or with much fewer training examples) provided that it can learn to compose these learned mechanisms. The process of learning can be seen as a form of inductive inference over this compositional representation system [Piantadosi et al., 2016].
3. **Inferential Coherence** - The idea is that similar tasks could use similar compositions or similar representations so reusing representations and transformations aid in the learning process. This manifests in machine learning systems in the form of pretraining and multi-task training approaches where the entire network or parts of the network are shared across tasks.

The compositionality property has been proposed as an important principle underlying cognitive control in humans [Cole et al., 2013, Sakai, 2008]. Lake et al. [2017] propose to use compositionality for learning generalizable models that can quickly adapt to new tasks and environments by effectively transferring knowledge across tasks. Diuk et al. [2008] and Cobo et al. [2013] also find compositionality to be a useful property when training RL systems. In section 1.3 we discussed that HRL systems can be seen as composition of policies operating at different levels of temporal abstractions. Hence the ability to compose representations seems to be a useful inductive bias.

1.6 Systematicity

Fodor and Pylyshyn [1988] defined *systematicity* in the following way: “the ability to learn to solve parts of one task is related to the ability to learn to solve parts of another task". For example, if a system can understand the sentence "John loves the girl", the system should also be able to understand sentences like "John does not love the girl" or "The girl loves John". How strongly can the learning agent generalize from one task to another depends on many factors including how similar the different parts of the distinct tasks are. It can be argued that compositionality is closely related to the concept of systematicity. Recall that the compositionality argument says that the world is composed of some kind of low-level mechanisms that are shared across different tasks. Hence systematicity depends on compositionality - if the world is not compositional and all the tasks are completely independent, it may as well not be systematic. In other words, if all the tasks are independent of each other in the true sense, learning to solve one task should not provide any useful information about the other task. Alternatively, one could say that a systematic learning system should also be compositional so that it is able to exploit the underlying compositional mechanisms and learn to solve novel tasks that are combinations of the tasks that the system has learned to solve.

1.7 Models of Cognition

In this section, we discuss some models that are used to describe human cognition and see if these models could provide useful cues for designing neural network architectures.

It is possible to study human cognitive capabilities at various levels. One such hierarchy was proposed by Marr et al. [1982] where three levels are considered:

1. **Computational level**: What is the goal of the high-level computation and how does the computation achieve that goal.
2. **Algorithmic level**: How can the computation be executed as an algorithm ?
3. **Hardware level**: How can the algorithm be realized in practice.

There are two broad approaches for developing models of cognition - the *bottom-up* approach and the *top-down* approach.

1.7.1 Bottom-up approach

In the *bottom-up* approach (or the "mechanism-first" approach), one first identifies the mechanisms that could be responsible for explaining the cognitive behavior and then comes up with an explanation for the behavior. For example, the *direct perception theory* [Gibson, 2002] proposes that the world provides sufficient information to our visual system to directly perceive the objects and we do not have to rely on some high order cognitive process or some external information ie we do not need to be told that we are in a garden before we perceive a flower. Cibson [1979] proposed the notion of *affordances* - "aspects (or features) of environments that allow an individual to perform an action". In a broad sense, affordances can be used to refer to all the interactions that are possible between an individual and an environment. The idea is that affordances can drive actions in a bottom-up manner without any external input. In our proposed approach (Chapter 2), we use a bottom-up approach to learn an ensemble of primitive policies which can be composed to solve the given task in a decentralized fashion.

1.7.2 Top-down approach

In the *top-down* approach (or the "function-first" approach), we start by identifying the "functions" (or the high-level constructs) that are needed to solve a problem and then express these high-level constructs in terms of low-level mechanisms. For example, the information from the environment could be ambiguous and some high-level cognitive perspective is needed to resolve the ambiguity. This high-level information could either come from a higher-level cognitive module (which might be keeping track of the past experience) or from some kind of external knowledge. Griffiths et al. [2010] argue that while both top-down and bottom-up approaches are useful, the top-down approach seems to be more useful for learning representations and inductive biases that seem to support the cognitive system in humans. Top-down approach is used in modular neural networks (section 1.9) and hierarchical reinforcement learning systems (section 1.3) where a high level *controller* (or meta-policy) is used to choose between *modules* (or low-level primitives). The controller corresponds to the "function" and while the modules correspond to the low level mechanisms.

1.8 How to learn Compositional Models

In this section, we review some work from cognitive science and neuroscience to understand what inductive biases might be useful for learning compositional models.

1.8.1 Learning Compositional Task Representation

Human studies [Cole et al., 2011, Reverberi et al., 2012] hint that the task representations learnt in the prefrontal cortex are compositional. For example, Reverberi et al. [2012] consider experiments where humans are shown images and provided with rules like "if there is a house, press left". Participants are first tested on tasks where only one rule needs to be invoked and then tested on tasks where multiple rules are to be used at once (compound rules). The authors report that they were able to decode the compound rules by training classifiers only on the simple rules using prefrontal neural activity patterns as the input data. This suggests that the inferior lateral frontal cortex uses the composition of the encoding of the constituent rules. While the nature of these compositions (permutation-invariant, sequential, hierarchical, etc) is not clear, these studies do provide a very useful insight - **a learning agent could learn a distribution of tasks by first learning the elementary skills that are useful across all the tasks and then learning to compose these skills to solve the different tasks**.

Following up on these works, Yang et al. [2019] considers an important question - if we train a network to perform many tasks, should its units be clustered and should its representation be compositional ? They train an RNN for solving 20 cognitive tasks simultaneously and study the emerging task representations. The tasks include variants of memory-guided response [Funahashi et al., 1989], simple perceptual decision making [Gold and Shadlen, 2007], context-dependent decision making [Siegel et al., 2015, Mante et al., 2013], multi-sensory integration [Raposo et al., 2014], parametric working memory [Romo et al., 1999] etc. They introduce a per-task, per-neuron measure called as the *task variance* which computes the average variance of a single neuron's noise-free response across the different stimulus conditions in that task. It can be seen as representing the amount of information encoded by the neuron unit for that task and can be used as a proxy for how "selective" a neuron is for a given task. These *task-variances* are normalized and used to cluster the neuron units. Units in the same cluster are found to be selective for the same subset of tasks. This kind of analysis can lead to even more interesting insights. For example, one cluster of units was found to be selective for both parametric working memory tasks and perceptual decision-making tasks. This indicates that there is some common substrate shared between the two tasks. **An inductive bias which tells the agent how different tasks are related to each other is quite useful for learning and multi-task training seems to provide this kind of bias to the learning system.**

1.8.2 Using models which are composed of other models

An alternate strategy would be to learn a set of *modules* (or networks) and learn to compose those networks for different tasks. This is the approach taken by the *Modular Neural Networks* and is described in detail in section 1.9.

1.8.3 Sequential Training vs MultiTask Training

Yang et al. [2019] considers a scenario where the tasks are learned sequentially, one after the other as this learning paradigm is closer to how animals learn as well. To avoid the problem of catastrophic forgetting [McCloskey and Cohen, 1989], they use the *intelligent synapses* based approach proposed by Zenke et al. [2017]. In this case, the network develops mixed task selectivity, that is, neurons within a cluster become selective towards different tasks. This is in contrast to the task specialization as exhibited in the case of multi-task training. Both these forms of task selectivity have their benefits. **If several tasks share a common substrate or a subtask, it is will be useful to develop a specialized mechanism for handling that substrate. On the other hand, if the system needs to continually learn new tasks while retaining performance on the previous tasks, a mixed-specialization based approach would be useful**. The focus of our work (chapter 2) is on the former setting where we learn specialized mechanisms and transfer them (via means of composition) on the new task.

1.9 Modular Neural Networks

Modular Neural Networks refers to the class of neural networks that are composed of *modules* which can be combined in different ways (depending on the task) to solve a task or a family of tasks [Happel and Murre, 1994, SHARKEY, 1997, Auda and Kamel, 1999, Andreas et al., 2016a,b, Johnson et al., 2017a, Santoro et al., 2017, Yu et al., 2018, Alet et al., 2018a,b]. Even though the modules are independent of each other, they are functionally integrated together (or put together like lego blocks) to solve a given task.

We adopt the following terminology for the rest of this section: "neural network" refers to the standard (monolithic) neural network while the term "modular neural network" refers to the modular neural network architecture described here. A standard neural network is composed of layers which can be thought of as modules but the key difference is that the topology of the network (or the nature of connections between the layers) does not change. In a modular network, the topology is decided dynamically depending on the current task and input. This "flexibility" of organization of modules allows for the same module to be reused as many times as needed while this is not possible with standard neural networks. The modules in modular neural networks are still neural networks which are to be trained jointly with other modules using the same backpropagation mechanism [LeCun et al., 1989] that is used for training standard neural networks.

1.9.1 Neural Module Networks

Andreas et al. [2016c] proposed the *Neural Module Network* (NMN) architecture in the context of a VQA task. The model comprises of a collection of neural modules which are dynamically composed to form a network to answer a given natural language question about an image. These modules are shared across the datapoints, jointly-trained and are expected to specialize as different mechanisms.

We first introduce a generic terminology for describing the Neural Module Networks.

The Neural Module Networks are comprised of:

1. a collection of modules m where each module $m_i \in m$ is itself a neural network with the parameters θ_i.

2. a *controller* whose task is to produce the network layout specifying how the modules are to be connected.

The controller is any function that takes in the input and maps it to a network layout z. Once this layout has been determined, the modules are arranged to form a neural network f which is used to generate the output y as shown in equation 1.26 below. The process of learning the modules (that make up f) is the same as the process of learning the parameters of different layers in a standard neural network. In general, we parameterize the controller to be a neural network (with parameters ϕ) that is trained jointly with the modules. Since the structure z is a discrete variable, we could use either the REINFORCE method [Williams, 1992] or the gumble-softmax trick [Maddison et al., 2016, Jang et al., 2016]. Even though the Neural Module Network was introduced in the context of VQA tasks, the framework is quite general.

$$z = P(input) \tag{1.25}$$

$$y = f_z(y|input, m) \tag{1.26}$$

Model

Now we describe the architecture of the Neural Module Network as introduced by Andreas et al. [2016c]. The input to the model is a tuple of the form (w, x) where x is the input image and w is the natural language question about the image. The model has to predict the correct answer y. The model comprises of a collection of pre-identified modules m and a controller which is referred to as the *network layout predictor* P but instead of learning the network layout predictor, the linguistic structure of the question is used to determine how the modules should be organized.

The input question is parsed with the Stanford Dependency Parser [Klein and Manning, 2003] to obtain a dependency parse representation. The set of dependencies consisting of the *w-words*("what", "who", "where", "how") are filtered to obtain a modular representation of the question. This representation can be directly mapped to a network layout and the resulting network can be used to answer the given question. Figure 1.5 (taken from Andreas et al. [2016a]) shows an example of this approach.

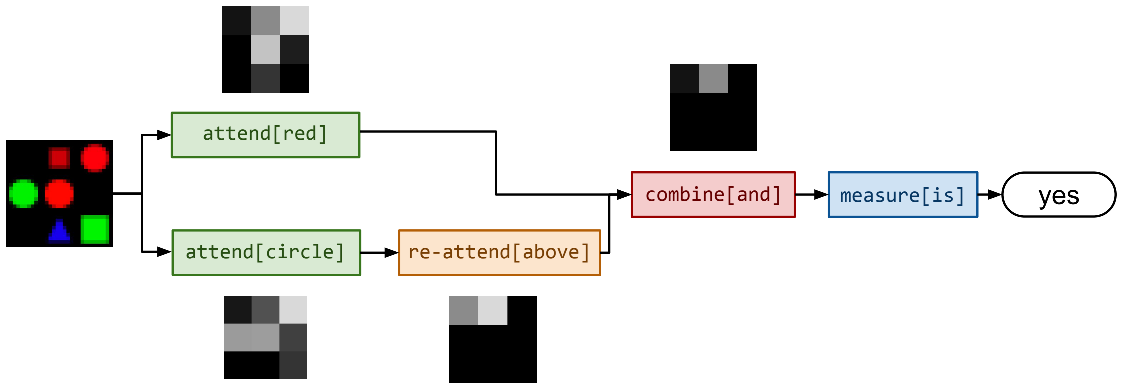


Figure 1.5 – Neural Module Network for answering the question *Is there a red shape above a circle ?* The two *attend* modules locate the red shapes and circles, the *re-attend[above]* shifts the attention above the circles, the *combine* module computes their intersection, and the *measure[is]* module inspects the final attention and determines that it is non-empty. Taken from Andreas et al. [2016a]

Using such a heuristically-designed network layout predictor makes the training of modules quite easy as the layout predictor does not need to be trained at all. The downside is that since the layout-predictor gets no training signal at all, the model might fail even if the modules are learned perfectly, because of incorrect network layout. Furthermore, in this case, the network layout depends only on the input question and any useful signal from the input image is completely disregarded.

Despite these limitations, the proposed approach obtains state-of-the-art results on two VQA datasets ([Antol et al., 2015, Andreas et al., 2016a]) showing that the approach is highly useful even in its limited form.

Andreas et al. [2016b] builds on the idea of Neural Module Network and proposes the *Dynamic Neural Module Network* architecture. The new architecture has two improvements over the previous one:

1. The set of five modules (m), that was originally introduced in the context of visual question answering, is now extended to be able to reason over any kind of structured representations (text or images). For this purpose, the following new modules are added: *Lookup*, *Relate*, *And*, and *Exists*.
2. Instead of completely relying on a dependency parser for generating the network layout, the new approach learns to select from the list of automatically generated candidate layouts.

Johnson et al. [2017b] proposed a variant of neural module networks and evaluate it for a visual reasoning task. Like before, the input to the system is an image-question pair (x, q) and the output is the answer to the question. The task is formulated as that of inducing a program to describe how the modules should be connected for the given input. Their proposed system has two components:

1. **Program Generator**: This component performs the role of the controller and produces a layout which describes how the modules should be connected. The key difference from the previous approaches is that the layout is generated, step by step, and not selected from a list of heuristically generated layouts. The generator takes as input the question q and produces the structure z for the network.
2. **Execution Engine**: The execution engine takes as input the given image x and the predicted structure z and executes the program (defined by z) on the image to obtain the answer y. Like the previous works, the set of modules (m) is predefined.

Since the program generator generates a discrete object. it is trained using the REINFORCE algorithm [Williams, 1992] while the execution engine is trained with the standard supervised learning techniques using the backpropagation algorithm.

2 Learning Competitive Ensembles of Information-Constrained Primitives

Learning Competitive Ensembles of Information-Constrained Primitives: Anirudh Goyal, Shagun Sodhani, Jonathan Binas, Xue Bin Peng, Yoshua Bengio, Sergey Levine

This chapter presents joint work with Anirudh Goyal, Jonathan Binas, Xue Bin Peng, Prof. Yoshua Bengio, and Prof. Sergey Levine. It is under review at Advances in Neural Information Processing Systems 32 (NeurIPS 2019) (Conference Track).

Contribution: Anirudh had the idea of training primitives without a master policy and using information bottleneck mechanism. I helped him develop the idea. I wrote much of the code for implementing and evaluating the approach, running the model in the batch setting, and for modifying the models to test on the new environments. I conducted experiments on four environments and helped with the draft of the paper. Anirudh did the motion imitation experiments using the code provided by Xue Bin Peng, helped me in finetuning the parameters for other experiments and wrote the first draft. Jonathan helped us with the writing and brainstorming the idea. Prof Yoshua and Prof Sergey helped us to develop the idea and provided feedback on the writing.

Affiliation

— Anirudh Goyal, MILA, University of Montreal

— Shagun Sodhani, MILA, University of Montreal

— Jonathan Binas, MILA, University of Montreal

— Xue Bin Peng, University of California, Berkeley

— Yoshua Bengio, MILA, University of Montreal

— Sergey Levine, University of California, Berkeley

2.1 Abstract

Reinforcement learning agents that operate in diverse and complex environments can benefit from the structured decomposition of their behavior. Often, this is addressed in the context of hierarchical reinforcement learning, where the aim is to decompose a policy into lower-level primitives or options, and a higher-level meta-policy that triggers the appropriate behaviors for a given situation. However, the meta-policy must still produce appropriate decisions in all states. In this work, we propose a policy design that decomposes into primitives, similarly to hierarchical reinforcement learning, but without a high-level meta-policy. Instead, each primitive can decide for themselves whether they wish to act in the current state. We use an information-theoretic mechanism for enabling this decentralized decision: each primitive chooses how much information it needs about the current state to make a decision and the primitive that requests the most information about the current state acts in the world. The primitives are regularized to use as little information as possible, which leads to natural competition and specialization. We experimentally demonstrate that this policy architecture improves over both flat and hierarchical policies in terms of generalization.

2.2 Introduction

Learning policies that generalize to new environments or tasks is a fundamental challenge in reinforcement learning. While deep reinforcement learning has enabled training powerful policies, which outperform humans on specific, well-defined tasks [Mnih et al., 2015a], their performance often diminishes even if only some properties of the environment or the task changes. This is in stark contrast to how humans learn, plan, and act: humans can seamlessly switch between different aspects of a task, transfer knowledge to new tasks from remotely related but essentially distinct prior experience, and combine primitives (or skills) used for distinct aspects of different tasks in meaningful ways to solve new problems. A hypothesis hinting at the reasons for this discrepancy is that the world is inherently compositional, such that its features can be described by compositions of small sets of primitive mechanisms [Parascandolo et al., 2017b]. Since humans seem to benefit from learning skills and learning to combine skills, it might be a useful inductive bias for the learning models as well.

This is addressed to some extent by the hierarchical reinforcement learning (HRL) methods, which focus on learning representations at multiple spatial and temporal scales, thus enabling better exploration strategies and improved generalization performance [Dayan and Hinton, 1993, Sutton et al., 1999b, Dietterich, 2000, Kulkarni et al., 2016]. However, hierarchical approaches rely on some form of learned high-level controller, which decides when to activate different components in the hierarchy. While low-level subpolicies can specialize to smaller portions of the state space, the top-level controller (or master policy) needs to know how to deal with any given state. That is, it should provide optimal behavior for the entire accessible state space. As the master policy is trained on a particular state distribution, learning it in a way that generalizes to new environments effectively can, therefore, become the bottleneck for such approaches [Sasha Vezhnevets et al., 2017, Andreas et al., 2017].

We argue, and empirically show, that in order to achieve better generalization, the interaction between the low-level primitives and the selection thereof should itself be performed without requiring a single centralized network that understands the entire state space. We, therefore, propose a fully decentralized approach as an alternative to the standard HRL techniques, where we only learn a set of low-level primitives without learning a high-level controller. We construct a factorized representation of the policy by learning simple "primitive" policies, which focus on distinct regions of the state space. Rather than being gated by a single meta-policy, the primitives directly compete with one another to determine which one should be active at any given time, based on the degree to which their state encoders "recognize" the current state input.

We frame the problem as one of information transfer between the current state and a dynamically selected primitive policy. Each policy can by itself decide to request information about the current state, and the amount

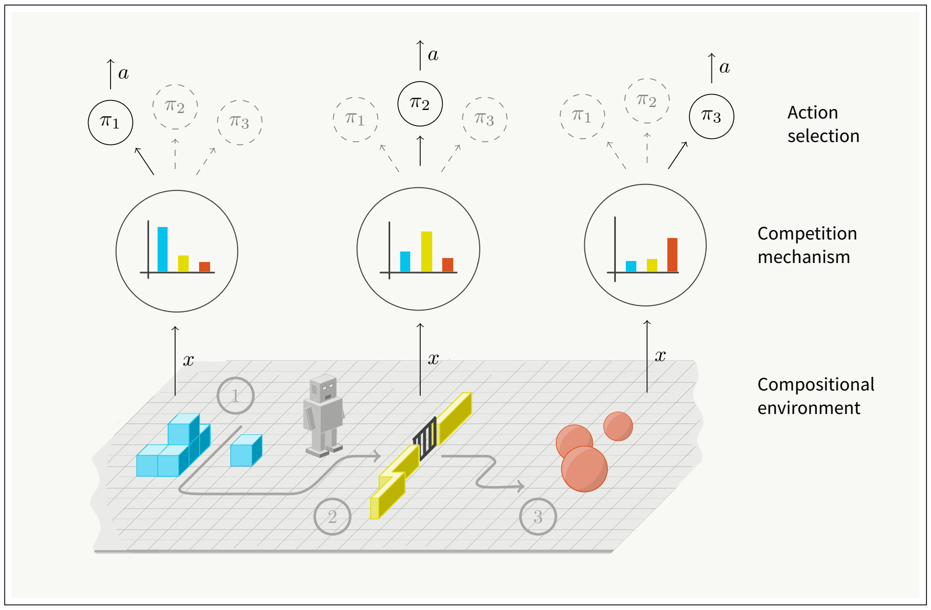


Figure 2.1 – Illustration of our model. An intrinsic competition mechanism, based on the amount of information each primitive provides, is used to select a primitive to be active for a given input. Each primitive focuses on distinct features of the environment ; in this case, one policy focuses on boxes, a second one on gates, and the third one on spheres.

of information requested is used to determine which primitive acts in the current state. Since the amount of state information that a single primitive can access is limited, each primitive is encouraged to use its resources wisely. Constraining the amount of accessible information in this way naturally leads to a decentralized competition and decision mechanism, where individual primitives specialize in smaller regions of the state space. We formalize this information-driven objective based on the variational information bottleneck. The resulting set of competing primitives achieves both a meaningful factorization of the policy and an effective decision mechanism for which primitives to use. Importantly, not relying on a centralized meta-policy means that individual primitive mechanisms can be recombined in a *"plug-and-play"* fashion, and can be transferred seamlessly to new environments.

Contributions: In summary, the contributions of our work are as follows:

1. We propose a method for learning and operating a set of functional primitives in a fully decentralized way, without requiring a high-level

meta-controller to select active primitives (see Figure 2.1 for illustration).

2. We introduce an information-theoretic objective, the effects of which are twofold: a) it leads to the specialization of individual primitives to distinct regions in the state space, and b) it enables a competition mechanism, which is used to select active primitives in a decentralized manner.
3. We demonstrate the superior transfer learning performance of our model, which is due to the flexibility of the proposed framework regarding the dynamic addition, removal, and recombination of primitives. Decentralized primitives can be successfully transferred to larger or previously unseen environments, and outperform models with an explicit meta-controller for primitive selection.

2.3 Preliminaries

We consider a Markov decision process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where the state space $\mathcal{S}$ and the action space $\mathcal{A}$ may be discrete or continuous. The environment emits a bounded reward $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$ on each transition and $\gamma \in [0, 1)$ is the discount factor. $\pi(.|s)$ denotes a policy over the actions given the current state s. $R(\pi) = \mathbb{E}_\pi[\sum_t \gamma^t r(s_t)]$ denotes the expected total return when the policy π is followed. The standard objective in reinforcement learning is to maximize the expected total return $R(\pi)$. We use the concept of the information bottleneck [Tishby et al., 2000] to learn compressed representations. The information bottleneck objective is formalized as minimizing the mutual information of a *bottleneck* representation layer with the input while maximizing its mutual information with the corresponding output. This type of input compression has been shown to improve generalization [Tishby et al., 2000, Achille and Soatto, 2016, Alemi et al., 2016].

Computing the mutual information is generally intractable, but can be approximated using variational inference [Alemi et al., 2016].

2.4 Related Work

There are a wide variety of hierarchical reinforcement learning approaches [Sutton et al., 1998a, Dayan and Hinton, 1993, Dietterich, 2000]. One of the most widely applied HRL framework is the *Options* framework ([Sutton et al., 1999b]). An option can be thought of as an action that extends over multiple timesteps thus providing the notion of temporal abstraction or subroutines in an MDP. Each option has its own policy (which is followed if the option is selected) and the termination condition (to stop the execution of that option). Many strategies are proposed for discovering options using task-specific hierarchies, such as pre-defined sub-goals [Heess et al., 2017], hand-designed features [Florensa et al., 2017], or diversity-promoting priors [Daniel et al., 2012, Eysenbach et al., 2018]. These approaches do not generalize well to new tasks. Bacon et al. [2017] proposed an approach to learn options in an end-to-end manner by parameterizing the intra-option policy as well as the policy and termination condition for all the options. Eigen-options [Machado et al., 2017a] use the eigenvalues of the Laplacian (for the transition graph induced by the MDP) to derive an intrinsic reward for discovering options as well as learning an intra-option policy.

In this work, we consider a sparse reward setup with high dimensional action spaces. In such a scenario, performing unsupervised pretraining or using auxiliary rewards leads to much better performance [Frans et al., 2017, Florensa et al., 2017, Heess et al., 2017]. Auxiliary tasks such as motion imitation have been applied to learn motor primitives that are capable of performing a variety of sophisticated skills [Liu and Hodgins, 2017, Peng et al., 2017, Merel et al., 2019b,a].

Another common HRL architecture is as follows: there are a number of low-level primitives whose action-space is the action space of the task and there is a high-level policy (called the controller or the master policy) whose job is to switch or coordinate between these low-level policies. These approaches aim to learn the hierarchical spatiotemporal decompositions from the rewards obtained while interacting with the environment [Sasha Vezhnevets et al., 2017]. Unlike these approaches, which focus on the single-task setting only, Frans et al. [2017] proposed a meta-learning algorithm to learn

an ensemble of primitives that can be designed to easily generalize across a distribution of tasks. Since our focus is on highlighting the superior performance of our method in multi-task setup and transfer learning setup, *MLSH* is an important baseline.
Our work is also related to the *Neural Module Network* family of architectures [Andreas et al., 2017, Johnson et al., 2017b, Rosenbaum et al., 2019] where the idea is to learn *modules* that can perform some useful computation like solving a subtask and a *controller* that can learn to combine these modules for solving novel tasks.
The key difference between our approach and all the works mentioned above is that we learn functional primitives in a fully decentralized way without requiring any high-level meta-controller or master policy.

2.5 Information-Theoretic Decentralized Learning of Distinct Primitives

Our goal is to learn a policy, composed of multiple primitive sub-policies, to maximize average returns over T-step interactions for a distribution of tasks. Simple primitives which focus on solving a part of the given task (and not the complete task) should generalize more effectively, as they can be applied to similar aspects of different tasks (i.e subtasks) even if the overall objective of the tasks are drastically different. Learning primitives in this way can also be viewed as learning a factorized representation of a policy, which is composed of several *independent* policies.
Our proposed approach consists of three components: 1) a mechanism for restricting a particular primitive to a subset of the state space ; 2) a competition mechanism between primitives to select the most effective primitive for a given state ; 3) a regularization mechanism to improve the generalization performance of the policy as a whole. We consider experiments with both fixed and variable sets of primitives and show that our method allows for primitives to be added or removed during training, or recombined in

new ways. Each primitive is represented by a differentiable, parameterized function approximator, such as a neural network.

2.5.1 Primitives with an Information Bottleneck

To encourage each primitive to encode information from a particular part of state space only, we limit the amount of information each primitive can access from the state. In particular, each primitive has an information bottleneck with respect to the input state, preventing it from using all the information from the state.

To implement an information bottleneck, we design each of the K primitives to be composed of an encoder $p_{\mathrm{enc}}(Z_k \mid S)$ and a decoder $p_{\mathrm{dec}}(A \mid Z_k)$, together forming the primitive policy,

$$\pi_\theta^k(A \mid S) = \int_z p_{\mathrm{enc}}(z_k \mid S)\, p_{\mathrm{dec}}(A \mid z_k)\, \mathrm{d}z_k \,.$$ [1]

The encoder output Z is meant to represent the information about the current state S that an individual primitive believes is important to access in order to perform well. The decoder takes this encoded information and produces a distribution over the actions A. Following the variational information bottleneck objective [Alemi et al., 2016], we penalize the KL divergence of Z and the prior,

$$\mathcal{L}_k = \mathrm{D}_{\mathrm{KL}}(p_{\mathrm{enc}}(Z_k|S)||\mathcal{N}(0,1))\,. \tag{2.1}$$

In other words, a primitive pays an "information cost" proportional to $\mathcal{L}_k$ for accessing the information about the current state. In the experiments below, we fix the prior to be a unit Gaussian. In the general case, we can learn the prior as well and include its parameters in θ. The information bottleneck encourages each primitive to limit its knowledge about the current state, but it will not prevent multiple primitives from specializing to similar parts of the state space. To mitigate this redundancy, and to make individual primitives focus on different regions in the state space, we introduce an information-

1. In practice, we estimate the marginalization over Z using a single sample throughout our experiments.

based competition mechanism to encourage diversity among the primitives, as described in the next section.

2.5.2 Competing Information-Constrained Primitives

We can use the information measure from equation 2.1 to define a selection mechanism for the primitives without having to learn a centralized meta-policy. The idea is that the information content of an individual primitive encodes its effectiveness in a given state s such that the primitive with the highest value $\mathcal{L}_k$
should be activated in that particular state. We compute normalized weights α_k for each of the $k = 1, \ldots, K$ primitives by applying the softmax operator,

$$\alpha_k = \exp(\mathcal{L}_k) / \textstyle\sum_j \exp(\mathcal{L}_j)\,. \tag{2.2}$$

The resulting weights α_k can be treated as a probability distribution that can be used in different ways: form a mixture of primitives, sample a primitive from the distribution, or simply select the primitive with the maximum weight. The selected primitive is then allowed to act in the environment.

$$\begin{tikzcd}
(\mathcal{L}_1, \ldots, \mathcal{L}_K) \arrow[r, "\text{softmax}"] & (\alpha_1, \ldots, \alpha_K) \arrow[d] \\
(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K) \arrow[u, "D_{\mathrm{KL}}(\cdot\|\mathcal{N})"'] \arrow[r, "\text{decoder}"'] & \mathbb{E}_\alpha \pi_k \arrow[d] \\
\text{state } s \arrow[u, "\text{encoder}"'] & \text{action } a
\end{tikzcd}$$

Figure 2.2 – The primitive-selection mechanism of our model. The primitive with most information acts in the environment, and gets the reward.

Trading Reward and Information: To make the different primitives compete for competence in the different regions of the state space, the environment reward is distributed according to their participation in the global decision, i.e. the reward r_k given to the k^{th} primitive is weighted by its selection coefficient, such that $r_k = \alpha_k r$, with $r = \sum_k r_k$. Hence, a primitive that access more information about the current state has a higher scaling factor which could result in higher returns (even though the returns from the environment remain the same, the primitive experiences a scaled version of the return). At the same time, accessing more information requires the primitive to pay a higher price (equal to information cost i.e. $\mathcal{L}_k$) as well. Hence, a primitive that does not access any state information, will not have to pay any information cost. This means that the primitives need to balance between the cost ($\mathcal{L}_k$) and benefit (α_k) of encoding the information. The information bottleneck and the competition mechanism, when combined with the overall reward maximization objective, should lead to specialization of individual primitives to distinct regions in the state space. That is, each primitive should specialize in a part of the state space that it can reliably associate rewards with. Since the ensemble together still needs to understand all of the state space for the given task, different primitives need to encode and focus on different parts of the state space. Figure 2.2 depicts the various components of our proposed model.

2.5.3 Regularization of the Combined Representation

To encourage a diverse set of primitive configurations and to make sure that the model does not collapse to a single primitive (which remains active at all times), we introduce an additional regularization term,

$$\mathcal{L}_{\text{reg}} = \sum_k \alpha_k \mathcal{L}_k \, . \tag{2.3}$$

where

$$\alpha_k = e^{\mathcal{L}_k} / \sum_j e^{\mathcal{L}_j} \, ,$$

and thus

$$\log \alpha_k = \mathcal{L}_k - \log \sum_j e^{\mathcal{L}_j} \, ,$$

or

$$\mathcal{L}_k = \log \alpha_k + \text{LSE}(\mathcal{L}_1, \ldots, \mathcal{L}_K) \, ,$$

where LSE is the LogSumExp function, $\mathrm{LSE}(x) = \log(\sum_j e^{x_j})$. Note that $\mathrm{LSE}(\mathcal{L}_1, \ldots, \mathcal{L}_K) = \log \sum_j e^{\mathcal{L}_j}$ is independent of k.

Plugging this in, and using $\sum \alpha_k = 1$, we get

$$\mathcal{L}_{reg} = \sum_k \alpha_k \log \alpha_k + \mathrm{LSE}(\mathcal{L}_1, \ldots, \mathcal{L}_K) = -H(\alpha) + \mathrm{LSE}(\mathcal{L}_1, \ldots, \mathcal{L}_K)\,. \tag{2.4}$$

where $H(\alpha)$ is the entropy of the α distribution. The desired behavior is achieved by minimizing $\mathcal{L}_{\mathrm{reg}}$. Increasing the entropy of α leads to a diverse set of primitive selections, ensuring that different combinations of the primitives are used. On the other hand, LSE approximates the maximum of its arguments, $\mathrm{LSE}(x) \approx \max_j x_j$, and, therefore, penalizes the dominating $\mathcal{L}_k$ terms, thus equalizing their magnitudes.

Information-theoretic interpretation Notably, $\mathcal{L}_{\mathrm{reg}}$ also represents an upper bound to the KL-divergence of a mixture of the currently active primitives and a prior,

$$\mathcal{L}_{\mathrm{reg}} \geq \mathrm{D}_{\mathrm{KL}}(\sum_k \alpha_k p_{\mathrm{enc}}(Z_k|S) || \mathcal{N}(0,1))\,,$$

and thus can be regarded as a term limiting the information content of the mixture of all active primitives. This arises from the convexity properties of the KL divergence, which directly lead to

$$\mathrm{D}_{\mathrm{KL}}(\sum_k \alpha_k f_k || g) \leq \sum_k \alpha_k \mathrm{D}_{\mathrm{KL}}(f_k || g)\,.$$

2.5.4 Objective and Algorithm Summary

Our overall objective function consists of 3 terms,

1. The expected return from the standard RL objective, $R(\pi)$ which is distributed among the primitives according to their participation,
2. The individual bottleneck terms leading the individual primitives to focus on specific parts of the state space, $\mathcal{L}_k$ for $k = 1, \ldots, K$,
3. The regularization term applied to the combined model, $\mathcal{L}_{\mathrm{reg}}$.

The overall objective for the k^{th} primitive thus takes the form:

$$J_k(\theta) \equiv \mathbb{E}_{\pi_\theta}[r_k] - \beta_{\text{ind}}\mathcal{L}_k - \beta_{\text{reg}}\mathcal{L}_{\text{reg}}\,, \tag{2.5}$$

where $\mathbb{E}_{\pi_\theta}$ denotes an expectation over the state trajectories generated by the agent's policy, $r_k = \alpha_k r$ is the reward given to the kth primitive, and β_{ind}, β_{reg} are the hyper-parameters controlling the impact of the respective terms.

Implementation: In our experiments, the encoders $p_{\text{enc}}(z_k|S)$ and decoders $p_{\text{dec}}(A|z_k)$ are represented by neural networks, the parameters of which we denote by θ. The encoder $p_{\text{enc}}(z_k|S)$ produce the mean and the log-variance of a Gaussian distribution from which the z_k values can be sampled. z_k and actions are sampled for each primitive at every step. While our approach is compatible with any RL method, we maximize $J(\theta)$ computed on-policy from the sampled trajectories using a score function estimator [Williams, 1992, Sutton et al., 1999a], specifically A2C [Mnih et al., 2016] (unless otherwise noted). Every experimental result reported has been averaged over 5 random seeds. Our model introduces 2 extra hyper-parameters β_{ind}, β_{reg}.

2.6 Environments

2.6.1 2D Bandits Environment

We use the 2D moving bandits task, introduced in Frans et al. [2017]. In this task, the agent is placed in a 2D world and is shown the position of two randomly placed points. One of these points is the goal point but the agent does not know which. We use the sparse reward setup where the agent receives the reward of 1 if it is within a certain distance of the goal point and 0 at all other times. Every episode lasts for 50 steps and to get the reward, the learning agent must reach near the goal point in those 50 steps. The agent's action space consists of 5 actions - moving in one of the four cardinal directions (top, down, left, right) and staying still.

Observation Space The 2D bandits task provides a 6-dimensional flat observation. The first two dimensions correspond to the (x, y) coordinates of the current position of the agent and the remaining four dimensions correspond to the (x, y) coordinates of the two randomly chosen points.

2.6.2 Four-rooms Environment

We consider the Four-rooms grid environment [Sutton et al., 1999c] where the agent has to navigate its way through a grid of four interconnected rooms to reach a goal position within the grid. The agent can perform one of the following four actions: *move up*, *move down*, *move left*, *move right*. The environment is stochastic and with 1/3 probability, the agent's chosen action is ignored and a new action (randomly selected from the remaining 3 actions) is executed ie the agent's selected action is executed with a probability of only 2/3 and the agent takes any of the 3 remaining actions with a probability of 1/9 each.

The World In the Four-rooms setup, the world (environment for the learning agent) is a square grid 11×11. The grid is divided into 4 rooms such that each room is connected with two other rooms via hallways. The layout of the rooms is shown in figure 2.3. The agent spawns at a random position and has to navigate to a goal position within 500 steps.

Reward Function We consider the sparse reward setup where the agent gets a reward (of 1) only if it completes the task (and reaches the goal position) and 0 at all other time steps. We also apply a time limit of 300 steps on all the tasks.

Observation Space The environment has a total of 104 states (or cells) that can be occupied. These states are mapped to integer identifiers. At any time t, the environment observation is a one-hot representation of the identifier corresponding to the state (or the cell) the agent is in right now and does not return any information about the goal state.

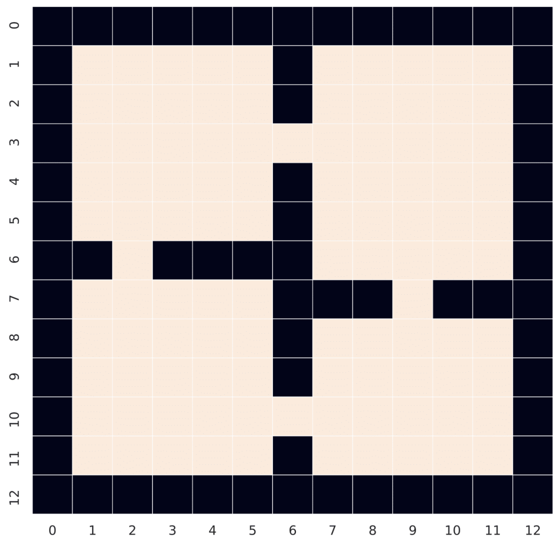


Figure 2.3 – View of the four-room environment

Task distribution In the Four-room environment, the agent has to navigate to a goal position which is randomly selected from a set of goal positions. We can use the size of this set of goal positions to define a curriculum of task distributions. Since the environment does not provide any information about the goal state, the larger the goal set, the harder is the task. Specifically, we consider three tasks - *Fourroom-v0*, *Fourroom-v1* and *Fourroom-v2* with the set of 2, 4 and 8 goal positions respectively. The set of goal positions for each task is fixed but not known to the learning agent.

2.6.3 Ant Maze Environment

We use the Mujoco-based quadruple ant [Todorov et al., 2012] to evaluate the transfer performance of our approach on the cross maze environment [Haarnoja et al., 2018]. The training happens in two phases. In the first phase, we train the ant to walk on a surface using a motion reward and using just 1 primitive. In the second phase, we make 4 copies of this trained policy and train the agent to navigate to a goal position in a maze (Figure 2.4). The goal position is chosen from a set of 3 (or 10) goals. The environment is a continuous control environment and the agent can directly manipulate the movement of joints and limbs.

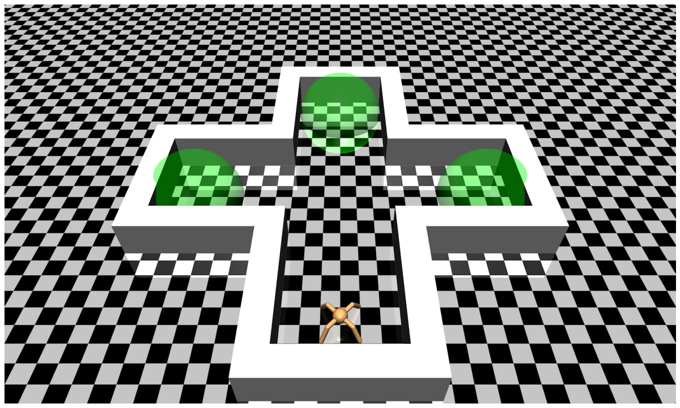


Figure 2.4 – View of the Ant Maze environment with 3 goals

Observation Space In the first phase (training the ant to walk), the observations from the environment correspond to the state-space representation ie a real-valued vector that describes the state of the ant in mechanical terms - position, velocity, acceleration, angle, etc of the joints and limbs. In the second phase (training the ant to navigate the maze), the observation from the environment also contains the location of the goal position along with the mechanical state of the ant.

2.6.4 MiniGrid Environment

We use the MiniGrid environment [Chevalier-Boisvert et al., 2018] which is an open-source, grid-world environment package[2]. It provides a family of customizable reinforcement learning environments that are compatible with the OpenAI Gym framework [Brockman et al., 2016]. Since the environments can be easily extended and modified, it is straightforward to control the complexity of the task (eg controlling the size of the grid, the number of rooms or the number of objects in the grid, etc). Such flexibility is very useful when experimenting with curriculum learning or testing for generalization.

2. https://github.com/maximecb/gym-minigrid

The World In MiniGrid, the world (environment for the learning agent) is a rectangular grid of size say $M \times N$. Each tile in the grid contains either zero or one object. The possible object types are *wall*, *floor*, *lava*, *door*, *key*, *ball*, *box* and *goal*. Each object has an associated string (which denotes the object type) and an associated discrete color (could be red, green, blue, purple, yellow and grey). By default, walls are always grey and goal squares are always green. Certain objects have special effects. For example, a key can unlock a door of the same color.

Reward Function We consider the sparse reward setup where the agent gets a reward (of 1) only if it completes the task and 0 at all other time steps. We also apply a time limit of 500 steps on all the tasks.

Action Space The agent can perform one of the following seven actions per timestep: *turn left*, *turn right*, *move forward*, *pick up an object*, *drop the object being carried*, *toggle*, *done* (optional action).
The agent can use the *turn left* and *turn right* actions to rotate around and face one of the 4 possible directions (north, south, east, west). The *move forward* action makes the agent move from its current tile onto the tile in the direction it is currently facing, provided there is nothing on that tile, or that the tile contains an open door. The *toggle* actions enable the agent to interact with other objects in the world. For example, the agent can use the *toggle* action to open the door if they are right in front of it and have the key of matching color.

Observation Space The MiniGrid environment provides partial and egocentric observations. For all our experiments, the agent sees the view of a square of 4×4 tiles in the direction it is facing. The observations are provided as a tensor of shape $4 \times 4 \times 3$. Additionally, the environment also provides a natural language description of the goal. An example of the goal description is: "Unlock the door and pick up the red ball".

Tasks in MiniGrid Environment We consider the following tasks in the MiniGrid environment:

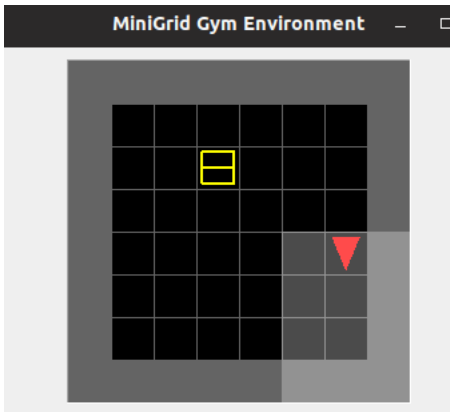


Figure 2.5 – RGB view of the *Fetch* environment.

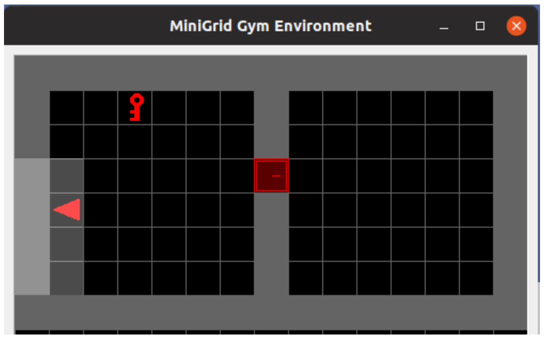


Figure 2.6 – RGB view of the *Unlock* environment.

1. **Fetch**: In the *Fetch* task, the agent spawns at an arbitrary position in a 8×8 grid (figure 2.5). It is provided with a natural language goal description of the form "go fetch a yellow box". The agent has to navigate to the object being referred to in the goal description and pick it up.
2. **Unlock**: In the *Unlock* task, the agent spawns at an arbitrary position in a two-room grid environment. Each room is 8×8 square (figure 2.6). It is provided with a natural language goal description of the form "open the door". The agent has to find the key that corresponds to the color of the door, navigate to that key and use that key to open the door.
3. **UnlockPickup**: This task is basically a union of the *Unlock* and the *Fetch* tasks. The agent spawns at an arbitrary position in a two-room grid environment. Each room is 8×8 square (figure 2.7). It is provided with a natural language goal description of the form "open the door and

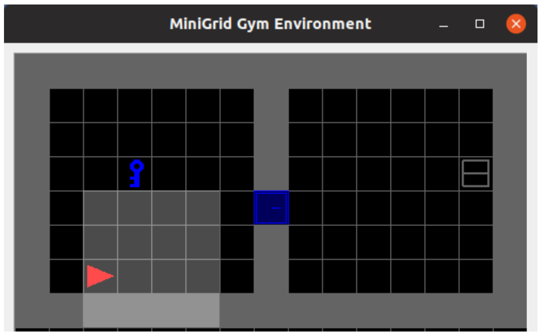


Figure 2.7 – RGB view of the *UnlockPickup* environment.

pick up the yellow box". The agent has to find the key that corresponds to the color of the door, navigate to that key, use that key to open the door, enter the other room and pick up the object mentioned in the goal description.

2.7 Implementation Details

In this section, we describe the implementation details which are common for all the models. Other task-specific details are covered in the respective task sections.

1. All the models (proposed as well as the baselines) are implemented in Pytorch 1.1 unless stated otherwise. [Paszke et al., 2017].
2. For Meta-Learning Shared Hierarchies [Frans et al., 2017] and Option-Critic [Bacon et al., 2017], we adapted the author's implementations [3]for our environments.
3. During the evaluation, we use 10 processes in parallel to run 500 episodes and compute the percentage of times the agent solves the task within the prescribed time limit. This metric is referred to as the "success rate".
4. The default time limit is 500 steps for all the tasks unless specified otherwise.

3. https://github.com/openai/mlsh, https://github.com/jeanharb/option_critic

5. All the feedforward networks are initialized with the *orthogonal* initialization where the input tensor is filled with a (semi) orthogonal matrix.
6. For all the embedding layers, the weights are initialized using the unit-Gaussian distribution.
7. The weights and biases for all the GRU model are initialized using the uniform distribution from $U(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{hidden\_size}$.
8. During training, we perform 64 rollouts in parallel to collect 5-step trajectories.
9. The β_{ind} and β_{reg} parameters are both selected from the set $\{0.001, 0.005, 0.009\}$ by performing cross validation.

In section 2.7.2, we explain all the components of the model architecture along with the implementation details in the context of the MiniGrid Environment. For the subsequent environments, we describe only those components and implementation details which are different than their counterpart in the MiniGrid setup and do not describe the components which work identically or the hyperparameters which do not change.

2.7.1 Model Architecture for MiniGrid

Encoder Architecture The agent's *encoder* network consists of two models - a CNN+GRU based *observation encoder* and a GRU [Cho et al., 2014] based *goal encoder*

Observation Encoder It is a three layer CNN with the output channel sizes set to 16, 16 and 32 respectively (with ReLU layers in between) and kernel size set to 2×2 for all the layers. The output of the CNN is flattened and fed to a GRU model (referred to as the *observation-rnn*) with 128-dimensional hidden state. The output from the *observation-rnn* represents the encoding of the observation.

Goal Encoder It comprises of an embedding layer followed by a unidirectional GRU model. The dimension of the embedding layer and the hidden and the output layer of the GRU model are all set to 128.

The concatenated output of the *observation encoder* and the *goal encoder* represents the output of the *encoder*.

Decoder The decoder network comprises the *action network* and the *critic network* - both of which are implemented as two layer feedforward networks.

2.7.2 Components specific to the proposed model

The components that we described so far are used by both the baselines as well as our proposed model. We now describe the components that are specific to our proposed model. Our proposed model consists of an ensemble of primitives and the components we describe apply to each of those primitives.

Hyperparameters Table 2.1 lists the different hyperparameters for the MiniGrid tasks.

Table 2.1 – Hyperparameters for the Minigrid tasks

Parameter	Value
Learning Algorithm	A2C
Opitimizer	RMSProp[Tieleman and Hinton, 2012]
learning rate	$7 \cdot 10^{-4}$
batch size	64
discount	0.99
lambda (for GAE [Schulman et al., 2015])	0.95
entropy coefficient	10^{-2}
loss coefficient	0.5
Maximum gradient norm	0.5

Information Bottleneck Given that we want to control and regularize the amount of information that the *encoder* encodes, we compute the KL divergence between the output of the *action-feature encoder network* and a diagonal unit Gaussian distribution. More is the KL divergence, more is the information that is being encoded with respect to the Gaussian prior and vice-versa. Thus we regularize the primitives to minimize the KL divergence.

2.7.3 Model Architecture for 2D Bandits

Encoder Architecture

The agent's *encoder* network consists of a GRU-based recurrent model (referred as the *observation-rnn*) with a hidden state size of 128. The 6-dimensional observation from the environment is the input to the GRU model. The output from the *observation-rnn* represents the encoding of the observation.

2.7.4 Hyperparameters

Table 2.2 lists the different hyperparameters for the Bandit tasks.

Table 2.2 – Hyperparameters for the 2D Bandits task

Parameter	Value
Learning Algorithm	PPO [Schulman et al., 2017]
epochs per update (PPO)	10
Optimizer	Adam[Kingma and Ba, 2014]
learning rate	$3 \cdot 10^{-5}$
β_1	0.9
β_2	0.999
batch size	64
discount	0.99
entropy coefficient	0
loss coefficient	1.0
Maximum gradient norm	0.5

2.7.5 Model Architecture for Ant Maze Environment

Encoder Architecture

The agent's *encoder* network consists of a GRU-based recurrent model (referred as the *observation-rnn* with a hidden state size of 128. The real-valued state vector from the environment is fed to the GRU model. The output from the *observation-rnn* represents the encoding of the observation. Note that in the case of phase 1 vs phase 2, only the size of the input to the *observation-rnn* changes and the encoder architecture remains the same.

Decoder

The decoder network comprises the *action network* and the *critic network*. All these networks are implemented as feedforward networks. The design of these networks is very similar to that of the *decoder* model for the Mini-Grid environment as described in section 2.7.1 with just one difference. In this case, the action space is continuous so the *action-feature decoder network* produces the mean and log-standard-deviation for a diagonal Gaussian policy. This is used to sample a real-valued action to execute in the environment.

2.8 Experimental Results

We designed experiments to address the following questions:

1. **Learning primitives**: Can an ensemble of primitives be learned over a distribution of tasks ?
2. **Transfer Learning using primitives**: Can the learned primitives be transferred to unseen/unsolvable sparse environments ?
3. **Comparison to centralized methods**: How does our method compare to approaches where the primitives are trained using an explicit meta-controller, in a centralized way ?

Baselines. We compare our proposed method to the following baselines:

1. **Option Critic** [Bacon et al., 2017]: We extended the author's implementation[4] of the Option Critic architecture and experimented with multiple variations in the terms of hyperparameters and state/goal encoding. None of them yielded reasonable performance in partially observed tasks, so we omit it from the results.

4. https://github.com/jeanharb/option_critic

2. **MLSH**(Meta-Learning Shared Hierarchy [Frans et al., 2017]): This method uses meta-learning to learn sub-policies that are shared across tasks along with learning a task-specific high-level master. It also requires a phase-wise training schedule between the master and the sub-policies to stabilize training. We use the MLSH implementation provided by the authors[5].
3. **Transfer A2C**: In this method, we first learn a single policy on one task and then transfer the policy to another task, followed by fine-tuning in the second task.

2.8.1 Multi-Task Training

We evaluate our model on the Minigrid environment. We consider three tasks here: the *Pickup* task (A), where the agent is required to pick up an object specified by the goal string, the *Unlock* task (B) where the agent needs to unlock the door (there could be multiple keys in the environment and the agent needs to use the key which matches the color of the door) and the *UnlockPickup* task (C), where the agent first needs to unlock a door that leads to another room and in the next room, the agent needs to find and pick up the object specified by the goal string.

We train agents with varying numbers of primitives on various tasks – concurrently, as well as in transfer settings. The different experiments are summarized in Figs. 2.8 and 2.9. An advantage of the multi-task setting is that it allows for quantitative interpretability as to when and which primitives are being used. The results indicate that a system composed of multiple primitives generalizes more easily to a new task, as compared to a single policy. We further demonstrate that several primitives can be combined dynamically and that the individual primitives respond to stimuli from new environments when trained on the related environments.

5. https://github.com/openai/mlsh

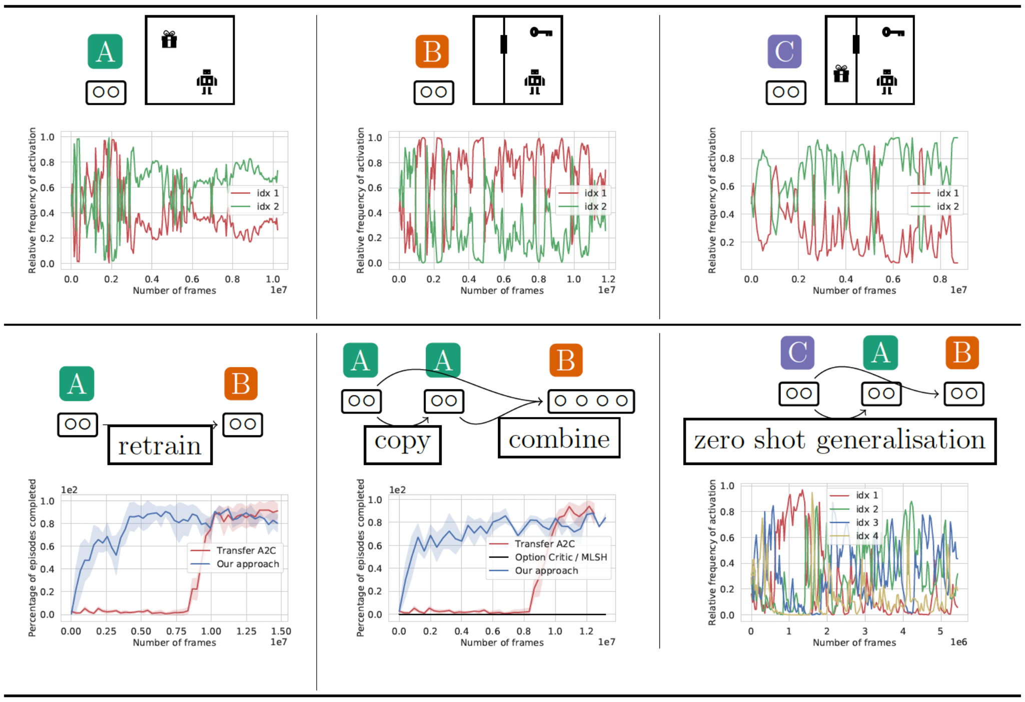


Figure 2.8 – **Multitask training**. Each panel corresponds to a different training setup, where different tasks are denoted A, B, C, ..., and a rectangle with n circles corresponds to an agent composed of n primitives trained on the respective tasks. Top row: activation of primitives for agents trained on single tasks. Bottom row: **Retrain:** two primitives are trained on A and transferred to B. The results (success rates) indicate that the multi-primitive model is substantially more sample efficient than the baseline (transfer A2C). **Copy and Combine:** more primitives are added to the model over time in a plug-and-play fashion (2 primitives are trained on A ; the model is extended with a copy of itself ; the resulting four-primitive model is trained on B.) This is more sample efficient than other strong baselines, such as [Frans et al., 2017, Bacon et al., 2017]. **Zero-Shot Generalization:** A set of primitives is trained on C, and zero-shot generalization to A and B is evaluated. The primitives learn a form of spatial decomposition which allows them to be active in both target tasks, A and B.

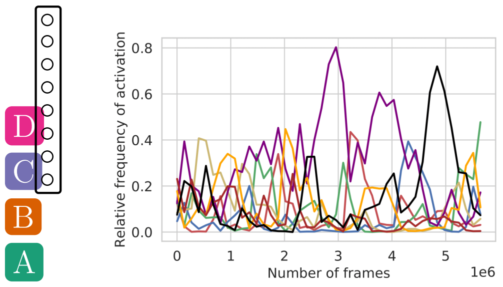
2.8.2 Do Learned Primitives Help in Transfer Learning ?

We now evaluate our approach in the settings where the adaptation to the changes in the task is vital. The argument in the favor of modularity is that it enables better knowledge transfer between related tasks. This transfer is more effective when the tasks are closely related as the model would only have to learn how to compose the already learned primitives. In general, it is difficult to determine how "closely" related two tasks are and the inductive bias of modularity could be harmful if the two tasks are quite different. We investigate here the transfer properties of a primitive trained in one environment and transferred to a different one. We want to answer the following questions:

1. Can our proposed approach learn primitives that remain active when training the agent over a sequence of tasks ?
2. Can our proposed approach be used to improve the sample efficiency of the agent over a sequence of tasks ?

To answer these questions, we consider two setups. In the baseline setup, we train a flat A2C policy on *Fourrooms-v0* till it achieves a 100 % success rate during evaluation. Then we transfer this policy to *Fourrooms-v1* and continue to train till it achieves a 100 % success rate during the evaluation on *Fourrooms-v1*. We transfer the policy one more time to *Fourrooms-v2* and continue to train the policy until it reaches a 60% success rate. In the last task(*Fourrooms-v2*), we do not use 100% as the threshold as the models do not achieve 100% for training even after training for 10M frames. We use 60% as the baseline models generally converge around this value.
In the second setup, we repeat this exercise of training on one task and transferring to the next task with our proposed model. Note that even though our proposed model converges to a higher value than 60% in the last task(*Fourrooms-v2*), we compare the number of samples required to reach 60% success rate to provide a fair comparison with the baseline.
Figure 2.10 shows that our proposed setup needs much fewer samples as compared to the A2C baseline which is trained in the same manner. The primitives aslo remain active over the distribution of the tasks.

Continuous control for ant maze We evaluate the transfer performance of pretrained primitives on the cross maze environment [Haarnoja et al., 2018]. Here, a quadrupedal robot must walk to the different goals along the different paths (see Section 2.6.3 for details). The goal is randomly chosen from a set of available goals at the start of each episode. We pretrain a policy with a motion reward in an environment which does not have any walls (similar to [Haarnoja et al., 2018]), and then transfer the policy to the second task where the ant has to navigate to a random goal chosen from one



Method	3 goals	10 goals
Flat Policy (PPO)	11 ± 5 %	4 ± 2 %
Option critic	18 ± 10 %	7 ± 3 %
MLSH	32 ± 3 %	5 ± 3 %
Explicit high level policy	21 ± 5 %	11 ± 2 %
Proposed method	68 ± 3%	40 ± 3%

Figure 2.9 – Left: Multitask setup, where we show that we are able to train 8 primitives when training on a mixture of 4 tasks. Right: Success rates on the different Ant Maze tasks. Success rate is measured as the number of times the ant is able to reach the goal (based on 500 sampled trajectories).

of the 3 (or 10) available goal options. For our model, we make four copies of the pretrained policies and then finetune the model using the pretrained policies as primitives. We compare to both MLSH [Frans et al., 2017] and option-critic [Bacon et al., 2017]. All these baselines have been pretrained in the same manner. As evident from Figure 2.9, our method outperforms the other approaches. The fact that the initial policies successfully adapt to the transfer environment underlines the flexibility of our approach.

2.8.3 Learning Ensembles of Functional Primitives

We evaluate our approach on a number of RL environments to show that we can indeed learn sets of primitive policies focusing on different aspects of a task and collectively solving it.

Motion Imitation To test the scalability of the proposed method, we present a series of tasks from the motion imitation domain. In these tasks, we train a simulated 2D biped character to perform a variety of highly dynamic skills by imitating motion capture clips recorded from human actors. Each mocap clip is represented by a target state trajectory $\tau^* = \{s_0^*, s_1^*, ..., s_T^*\}$, where s_t^* denotes the target state at timestep t. The input to the policy is augmented with a goal $g_t = \{s_{t+1}^*, s_{t+2}^*\}$, which specifies the the target states for the next two timesteps. Both the state s_t and goal g_t are then

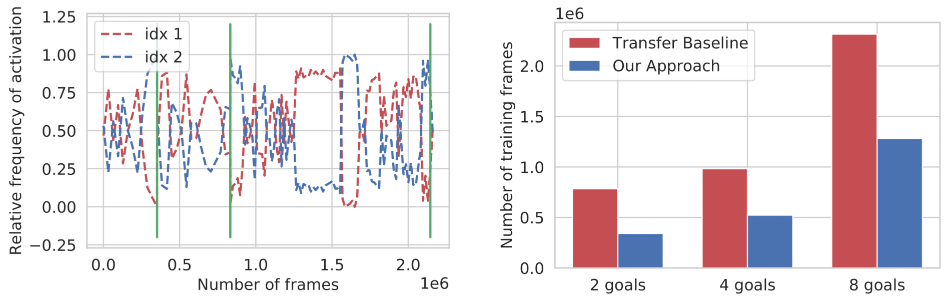


Figure 2.10 – Continual Learning Scenario: We consider a continual learning scenario where we train 2 primitives for 2 goal positions, then transfer (and finetune) on 4 goal positions then transfer (and finetune) on 8 goals positions. The plot on the left shows the primitives remain activated. The solid green line shows the boundary between the tasks, The plot on the right shows the number of samples taken by our model and the transfer baseline model across different tasks. We observe that the proposed model takes fewer steps than the baseline (an A2C policy trained in a similar way) and the gap in terms of the number of samples keeps increasing as tasks become harder.

processed by the encoder $p_{\text{enc}}(z_t|s_t, g_t)$. The repertoire of skills consists of 8 clips depicting different types of walks, runs, jumps, and flips. The motion imitation approach closely follows Peng et al. [2018].
Snapshots of some of the learned motions are shown in Figure 2.11.[6] To analyze the specialization of the various primitives, we computed 2D embeddings of states and goals which each primitive is active in, and the actions proposed by the primitives. Figure 2.12 illustrates the embeddings computed with t-SNE van der Maaten and Hinton [2008]. The embeddings show distinct clusters for the primitives, suggesting a degree of specialization of each primitive to certain states, goals, and actions.

2D Bandits We train two primitives on the 2D Bandits tasks and evaluate the relative frequency of activation of the primitives throughout the training. It is important that both the primitives remain active. If only 1 primitive is acting most of the time, its effect would be the same as training a flat policy. We evaluate the effectiveness of our model by comparing the success rate with a flat A2C baseline. Figure 2.13 shows that not only do both the primitives remain active throughout training, our approach also outperforms the baseline approach.

6. See supplementary information for video material.

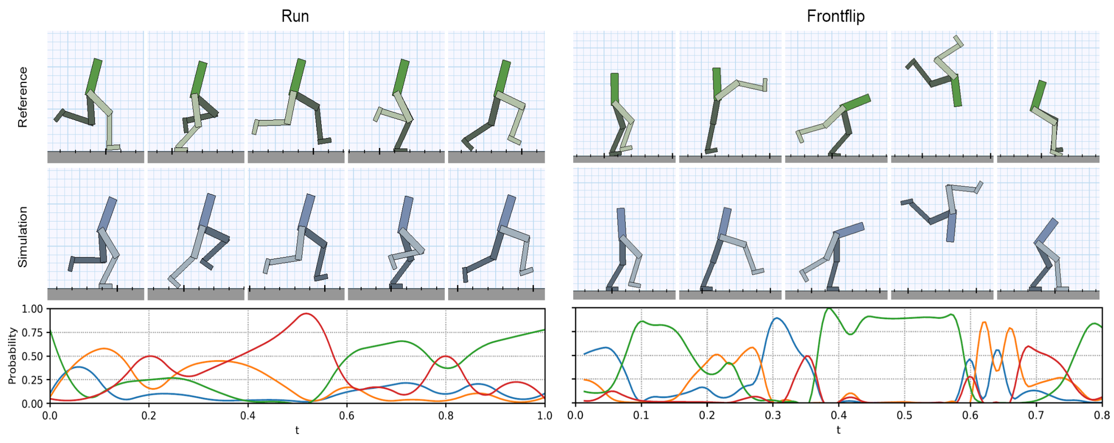


Figure 2.11 – Snapshots of motions learned by the policy. **Top:** Reference motion clip. **Middle:** Simulated character imitating the reference motion. **Bottom:** Probability of selecting each primitive.

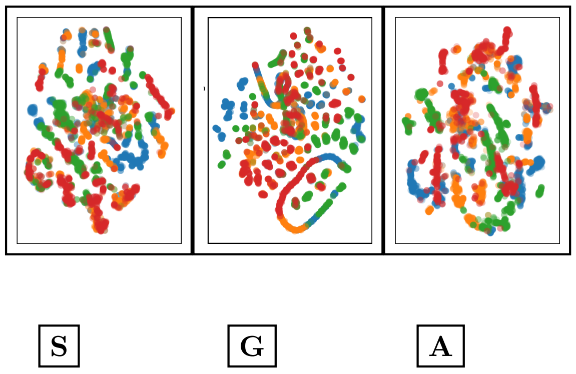
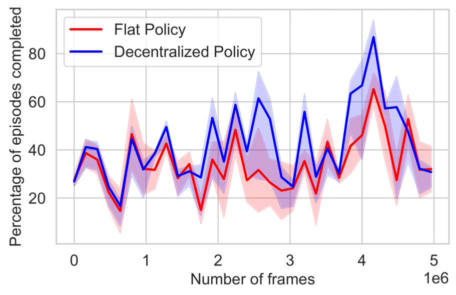


Figure 2.12 – Embeddings visualizing the states (S) and goals (G) which each primitive is active in, and the actions (A) proposed by the primitives for the motion imitation tasks. A total of four primitives are trained. The primitives produce distinct clusters.

2.9 Summary and Discussion

We present a framework for learning an ensemble of primitive policies which can collectively solve the tasks in a decentralized fashion. Rather than relying on a centralized, learned meta-controller, the selection of active primitives is implemented through an information-theoretic mechanism. The learned primitives can be flexibly recombined to solve more complex tasks. Our experiments show that, on a partially observed "Minigrid" task and a continuous control "ant maze" walking task, our method can enable better transfer than flat policies and hierarchical RL baselines, including the Meta-learning Shared Hierarchies model and the Option-Critic framework.



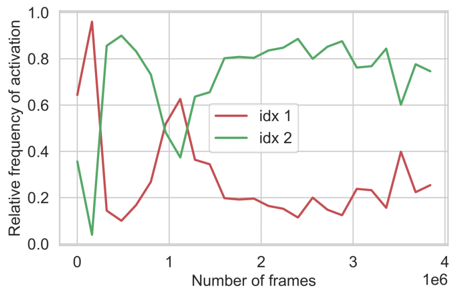


Figure 2.13 – Performance on the 2D bandits task. Left: The comparison of our model (blue curve - decentralized policy) with the baseline (red curve - flat policy) in terms of success rate shows the effectiveness of our proposed approach. Right: Relative frequency of activation of the primitives (normalized to sum up to 1). Both primitives are utilized throughout the training.

On Minigrid, we show how primitives trained with our method can transfer much more successfully to new tasks and on the ant maze, we show that primitives initialized from a pretrained walking control can learn to walk to different goals in a stochastic, multi-modal environment with nearly double the success rate of a more conventional hierarchical RL approach, which uses the same pretraining but a centralized high-level policy even though the architecture of the primitives remains the same in the two cases.

3 Conclusion

The work described in this thesis explores the problem of training an ensemble of low-level functional primitives which can be composed together to solve complex tasks. Our work is related to the Hierarchical Reinforcement Learning paradigm where several approaches exist for training a set of low-level policies using a high-level meta-policy to achieve structural decomposition of the behavior. The centralized decision-making approach (where the meta-policy decides which primitive will act in which state) makes the meta-policy a single point of failure. Our proposed approach improves over these approaches by introducing a mechanism for training the primitives in a decentralized manner. We introduce an information-theoretic objective which encourages competition between the primitives to specialize in different parts of the state space and provides a mechanism to select which primitive should be active in what state. We show that our approach achieves superior transfer learning performance as compared to both flat and hierarchical reinforcement learning approaches by enabling the dynamic addition, removal, and recombination of primitives.

In general, for a given arbitrary task (or distribution of tasks), it is very difficult to come up with the exact number of primitives that should be trained to obtain the most optimal composition. Given that the decision making (selecting the primitives) is decentralized, our approach enables adding and removing primitives both during training and during transfer. This gives us a greater degree of freedom to experiment with different number of primitives. For instance, we observe that if we introduce too many primitives, some of the primitives do not specialize at all and we can remove them from training. This allows for the possibility of dynamically changing the capacity of the network, while it is training. In the future work, we would like to explore this direction further to see if we can come up with an approach where the existing primitives may “vote” to add/remove new primitives to enable learning better compositions.

Another related application of our approach would be in the context of continual learning where we add primitives as the model trains over newer tasks. The two fundamental challenges in continual learning are i) catastrophic forgetting - as the model switches training from one task to another, it forgets the knowledge it acquired on the first task. ii) Capacity Saturation - as the model trains through a sequence of tasks, it runs out of effective capacity to learn new tasks without forgetting the knowledge from previous tasks. Adding new primitives provides a way for the old primitives to retain knowledge about the previous tasks while allowing the new primitives to adapt to the newer tasks where ever required. Note that existing HRL approaches cannot be extended for the continual learning use case in such a straightforward manner because of the meta-policy which would now have to adapt itself to all the new tasks.

There could be several possible mechanisms for enabling the training of primitives without an explicit meta-policy, In this work, we use the competition mechanism which works quite well in practice. Another way of formulating the objective for training the primitives would be in terms of how fast they adapt to new tasks. We could use a meta-learning approach to learn a factorization of the primitives that leads to faster adaptation to the transfer task. In order to have a good transfer performance, the system would have to learn an optimal factorization of the primitives.

Bibliographie

Alessandro Achille and Stefano Soatto. Information dropout: learning optimal representations through noise. *CoRR*, abs/1611.01353, 2016. URL http://arxiv.org/abs/1611.01353.

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. *CoRR*, abs/1612.00410, 2016. URL http://arxiv.org/abs/1612.00410.

Ferran Alet, Maria Bauza, Alberto Rodriguez, Tomas Lozano-Perez, and Leslie P Kaelbling. Modular meta-learning in abstract graph networks for combinatorial generalization. *arXiv preprint arXiv:1812.07768*, 2018a.

Ferran Alet, Tomás Lozano-Pérez, and Leslie P Kaelbling. Modular meta-learning. *arXiv preprint arXiv:1806.10166*, 2018b.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016a.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*, 2016b.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016c.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 166–175. JMLR. org, 2017.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

Gasser Auda and Mohamed Kamel. Modular neural networks: a survey. *International Journal of Neural Systems*, 9(02):129–151, 1999.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Noam Chomsky. Aspects of the theory of syntax. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE RESEARCH LAB OF ELECTRONICS, 1964.

Noam Chomsky. Language and mind. 1968.

JJ Cibson. The ecological approach to visual perceptions. 1979.

Luis C Cobo, Charles L Isbell, and Andrea L Thomaz. Object focused q-learning for autonomous agents. In *Proceedings of the 2013 international* *conference on Autonomous agents and multi-agent systems*, pages 1061–1068. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

Michael W Cole, Joset A Etzel, Jeffrey M Zacks, Walter Schneider, and Todd S Braver. Rapid transfer of abstract rules to novel contexts in human lateral prefrontal cortex. *Frontiers in human neuroscience*, 5:142, 2011.

Michael W Cole, Patryk Laurent, and Andrea Stocco. Rapid instructed task learning: A new window into the human brain's unique capacity for flexible cognitive control. *Cognitive, Affective, & Behavioral Neuroscience*, 13(1): 1–22, 2013.

Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281, 2012.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017a.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017b.

K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta Learning Shared Hierarchies. *arXiv e-prints*, October 2017.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

Gottlob Frege. *Uber Logik in der Mathematik: Fruhling, 1914*. Institut fur Mathematische Logik und Grundlagenforschung, Universitat Munster iW, 1914.

Shintaro Funahashi, Charles J Bruce, and Patricia S Goldman-Rakic. Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex. *Journal of neurophysiology*, 61(2):331–349, 1989.

James J Gibson. A theory of direct visual perception. *Vision and Mind: selected readings in the philosophy of perception*, pages 77–90, 2002.

Joshua I Gold and Michael N Shadlen. The neural basis of decision making. *Annu. Rev. Neurosci.*, 30:535–574, 2007.

Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, 14(8):357–364, 2010.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018.

Bart LM Happel and Jacob MJ Murre. Design and evolution of modular neural network architectures. *Neural networks*, 7(6-7):985–1004, 1994.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017a.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2989–2998, 2017b.

Edward L Keenan and Leonard M Faltz. Boolean semantics for natural language. 1987.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Libin Liu and Jessica Hodgins. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics*, 36(3), 2017.

Miao Liu, Marlos C Machado, Gerald Tesauro, and Murray Campbell. The eigenoption-critic framework. *arXiv preprint arXiv:1712.04065*, 2017.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. *arXiv preprint arXiv:1703.00956*, 2017a.

Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*, 2017b.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 553–560. ACM, 2005.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(Oct):2169–2231, 2007.

Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78, 2013.

David Marr et al. Vision: A computational investigation into the human representation and processing of visual information, 1982.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=BJfYvo09Y7.

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=BJl6TjRcY7.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *ArXiv e-prints*, February 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015a.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015b.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Richard Montague. Universal grammar. *Theoria*, 36(3):373–398, 1970.

Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. *arXiv preprint arXiv:1712.00961*, 2017a.

Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. *arXiv preprint arXiv:1712.00961*, 2017b.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 36(4):41:1–41:13, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073602. URL http://doi.acm.org/10.1145/3072959.3073602.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201311. URL http://doi.acm.org/10.1145/3197517.3201311.

Steven T Piantadosi, Joshua B Tenenbaum, and Noah D Goodman. The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological review*, 123(4):392, 2016.

Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

David Raposo, Matthew T Kaufman, and Anne K Churchland. A category-free neural population supports evolving demands during decision-making. *Nature neuroscience*, 17(12):1784, 2014.

Carlo Reverberi, Kai Görgen, and John-Dylan Haynes. Compositionality of rule representations in human prefrontal cortex. *Cerebral cortex*, 22(6): 1237–1246, 2012.

Ranulfo Romo, Carlos D Brody, Adrián Hernández, and Luis Lemus. Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature*, 399(6735):470, 1999.

Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

Katsuyuki Sakai. Task set and prefrontal cortex. *Annu. Rev. Neurosci.*, 31: 219–245, 2008.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust Region Policy Optimization. *ArXiv e-prints*, February 2015a.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *ArXiv e-prints*, June 2015b.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

AMANDA J C SHARKEY. Modularity, combining and artificial neural nets. *Connection Science*, 9(1):3–10, 1997.

Markus Siegel, Timothy J Buschman, and Earl K Miller. Cortical information flow during flexible sensorimotor decisions. *Science*, 348(6241): 1352–1355, 2015.

Alec Solway, Carlos Diuk, Natalia Córdova, Debbie Yee, Andrew G Barto, Yael Niv, and Matthew M Botvinick. Optimal behavioral hierarchy. *PLoS computational biology*, 10(8):e1003779, 2014.

Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643, 2017.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998a.

Richard S Sutton, Doina Precup, and Satinder P Singh. Intra-option learning about temporally abstract actions. In *ICML*, volume 98, pages 556–564, 1998b.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999a. MIT Press. URL http://dl.acm.org/citation.cfm?id=3009657.3009806.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999b.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999c.

Zoltán Gendler Szabó. Compositionality. 2004.

Sebastian B. Thrun and Tom M. Mitchell. Integrating inductive neural network learning and explanation-based learning. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 2*, IJCAI'93, pages 930–936, San Francisco, CA, USA, 1993. Morgan Kaufmann

Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1624140.1624154.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

Naftali Tishby, Fernando C. N. Pereira, and William Bialek. The information bottleneck method. *CoRR*, physics/0004057, 2000. URL http://arxiv.org/abs/physics/0004057.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Guangyu Robert Yang, Madhura R Joglekar, H Francis Song, William T Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature neuroscience*, 22(2):297, 2019.

Licheng Yu, Zhe Lin, Xiaohui Shen, Jimei Yang, Xin Lu, Mohit Bansal, and Tamara L Berg. Mattnet: Modular attention network for referring expression comprehension. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1307–1315, 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.