State Space Decomposition in Deep Reinforcement Learning

Saurabh Kumar[1]

Undergraduate Thesis

[1] A paper based on this work was accepted to the 2017 Multidisciplinary Conference on Reinforcement Learning and Decision Making. Himanshu Sahni, Farhan Tejani, Yannick Schroecker, and Charles Isbell were co-authors on that paper.

Acknowledgements

I would like Himanshu Sahni and my supervisor, Professor Charles Isbell, for extensive guidance on this project. A special thanks to Himanshu Sahni, Yannick Schroecker, and Farhan Tejani for the great discussions that we have had.

Abstract

Typical reinforcement learning (RL) agents learn to complete tasks specified by reward functions tailored to their domain. As such, the policies they learn do not generalize even to similar domains. To address this issue, we develop a framework through which a deep RL agent learns to generalize policies from smaller, simpler domains to more complex ones using a recurrent attention mechanism. The task is presented to the agent as an image and an instruction specifying the goal. This meta-controller guides the agent towards its goal by designing a sequence of smaller subtasks on the part of the state space within the attention, effectively decomposing it. As a baseline, we consider a setup without attention as well. Our experiments show that the meta-controller learns to create sub-goals within the attention. These results have implications for human-robot interactive applications, in which a robot can transfer skills it has learned in one task to another one and be robust to variability in its environment.

Introduction

Reinforcement learning (RL) is a subfield of artificial intelligence that addresses the problem of choosing behavior which maximizes some notion of long term cumulative reward. Within the RL framework, an artificial agent navigates an environment and receives rewards and punishments that it must use to determine what decisions to make. RL has successfully been applied to various domains, from learning to play Atari games from raw image pixels as input to beating world champions of backgammon and the ancient game of Go.

A reinforcement learning problem is typically formulated as a Markov decision process (MDP). An MDP is characterized by the tuple $< S, A, T, R, \gamma >$. S is the set of states an agent can be in. $A(S)$ is the set of actions the agent has available to it in each state. Typically, the agent chooses an action to execute from A, which may lead it into a new state according to the transition function $T : S \times A \rightarrow P(S)$. $R : S \times A \rightarrow R$ is a scalar value received upon executing an action in a state. Finally, $0 \leq \gamma \leq 1$ is the discount factor that determines how much future rewards are valued with respect to the agent's current state. A policy, $\pi : S \rightarrow P(A)$, informs the agent on which action to execute in each state. The goal of a reinforcement learning agent is to find $\pi*$, the optimal policy, which maximizes the long term expected reward, or utility, in each state.

Usually, RL agents cannot generalize policies learned in small domains to larger, more complicated ones. Yet this ability can allow them to immediately apply skills learned in simple settings without having to explore a large state space. In the deep learning setting, reduction in the size of the state also means smaller neural networks are required, which are easier to train. A neural network is a set of layers of neurons, where the first layer takes input data, applies non-linear activation functions to those inputs, and passes linear combinations of those inputs to subsequent layers. In this work, we design an approach to decompose complicated environments into simpler ones and provide sub-goals within them that ultimately solve the larger task. The agent can be pre-trained on the smaller environment to solve each sub-goal independently, or in conjunction with the sub-goal creation algorithm.

Our work most closely matches that of Kulkarni et al. [1]. They present a hierarchical framework where an agent learns from intrinsic rewards provided by a higher level agent setting sub-goals and operating on a longer time frame. Rewards for the higher level agent are provided by the environment for completing tasks. The sub-goals in turn are provided through functions over entities and relations in an object oriented framework.

In our work, we describe a meta-controller that learns to decompose the state space and provide sub-goals solvable within the smaller space. The meta-controller is solving a delayed reward problem as it only gets positive reinforcement when the underlying agent solves the original task. It has to come up with a sequence of sub-goals which maximize the expectation of this reinforcement. In addition to creating sub-goals, the meta-controller also fragments the state space such that the underlying agent is presented with a smaller state on which it can easily learn an optimal policy for the sub-goal. It does this by using an attention mechanism, similar to the Recurrent Attention Model [5]. The meta-controller learns to control its attention and only passes the part of the state within it to the agent. The meta-controller's MDP formulation is:

- States, S, are summaries of its past and current attentions.
- Actions, A, are the locations of the attention L^{attn}, and sub-goal distribution, g.
- Rewards, r, are +1 if the underlying agent solves the task and -0.01 otherwise.

• Transitions: The underlying agent executes its policy according to the state and sub-goal provided to it. Since this policy is unknown to the meta-controller this is a source of stochasticity in its environment.

The meta-controller selects a value for L^{attn} and a distribution *P(g).* The state space under that location and a sub-goal, *g*, are passed to the underlying agent. The agent then chooses an atomic action that moves it towards achieving *g*. The new agent location L^{agent} changes the meta-controller's environment, which picks a new attention and sub-goal. In this work, we make a few simplifying assumptions. First, we assume that the underlying agent has access to the optimal policy for each sub-goal. Such a goal-dependent policy can be learned by a technique such as universal value function approximators (UVFAs) [6]. UVFAs learn to approximate *V(S, g)*, or the value function with respect to a goal, using a function approximator such as deep neural nets. The learned value function $V(S,g;\theta)$ can be used to construct a policy that achieves the goal *g*. This value function can be trained independent of or in conjunction with the meta-controller by providing intrinsic rewards for achieving sub-goals [1]. Secondly, we assume that the agent remains still unless both its location and the sub-goal are present within the state provided to it by the meta-controller. In general, the meta-controller will automatically be incentivized to focus its attention and provide sub-goals such that the underlying agent is able to solve the given task because of its reward structure. In this case, that means keeping both the agent location and the sub-goal within the attention. For example, in the game of Pacman, if the sub-goal is to eat the closest pill, the underlying agent should have both the Pacman and at least one pill within the state provided to it. Otherwise, the agent may move randomly, and will be unable to achieve its overall goal of getting a high score. The above assumptions simplify training of the meta-controller, but the methodology we provide should be applicable in the general setting where the policy of the underlying agent is learned as well.

We extend upon the hierarchical reinforcement learning approach developed by Kulkarni et al. by decomposing the state space such that the base agent has to only see a small portion of it at any time. This allows for better computational efficiency, as the base agent can now use smaller networks, and may allow for transfer of learnt policies to different parts of the state space that are similar without having to explicitly explore them. To achieve this, the higher level agent, or meta-controller, must learn to integrate information over the states it has observed so far. Therefore, we use a recurrent model to represent the meta-controller through a Long Short Term Memory (LSTM) network [7]. In order to train the attention mechanism of our meta-controller, we employ a technique similar to that of Mnih et al. [5]. They use policy gradients to train an attention mechanism for classification and simple control tasks. In our approach, we do not employ a complex glimpse sensor, but instead simply use a 5 x 5 crop of our input image. This can be incorporated into our setup easily. Further, instead of specifying L^{attn} directly using a continuous output, we use discrete actions, *up, down, no-op*, to move the attention. Finally, Schaul et al. [6] describe how goal specific function approximators may be constructed for deep RL agents. Such a function can be constructed for our base learner by independently learning sub-goals on a 5x5 image.

Literature Review

This study explores how to transfer a reinforcement learning agent's knowledge from one task to another. The ability for an agent to reuse skills it has learned in a simple domain in a more complicated environment would save computation time, significantly speeding up the training process. This method would also allow agents to be able to solve various types of tasks using a single knowledge base, which would result in more robust artificial intelligence algorithms. Below, recent work in task decomposition, sub-goal discovery, and recurrent architectures is discussed.

Transfer in reinforcement learning is a difficult problem that has not been well explored in the literature. One proposed approach to skill generalization by Kulkarni et al. is the use of a meta-controller, which selects sub-goals for an underlying agent [1]. The presence of this meta-controller results in more efficient exploration of the environment, since the meta-controller can guide the movement of the underlying agent. However, when presented with a new domain, the agent once again has to learn the task from scratch. Another approach by Bacon et. al is to learn a policy over options [2]. Options provide a way for reinforcement learning agents to pick sequences of actions rather than primitive actions at each time step. This approach removes the need for a meta-controller to provide sub-goals, since the agent learns action combinations that represent those sub-goals. One key advantage of this approach is that an agent can learn options which do not need to be hand-designed by someone familiar with the domain. At the same time, no decomposition of the state space takes place, and this decomposition could increase computational efficiency. Additionally, a deliberation cost is needed to prevent the options from becoming degenerate to length one.

Recent work by Vezhnevets et al. builds upon the hierarchical structure developed by

Kulkarni et al. by introducing a mechanism involving a Manager and a Worker, known as feudal reinforcement learning [3]. The Manager interacts with the environment and selects sub-goals for the Worker, which then learns to complete those sub-goals using intrinsic rewards. The Manager is trained to predict directions that the Worker should follow, while the Worker is trained to follow these directions. This work also developed a way to jointly train the Manager and Worker while accounting for the different time scales at which these two components operate. One drawback of this mechanism, however, is that it does not effectively decompose the state space into manageable chunks that the agent already knows how to solve. By ignoring certain components of the state space that may not be relevant at a particular instance to the overall task, an agent can generalize its knowledge of how to do simple tasks to more complex ones. In our study, we adopt the feudal reinforcement learning framework and also address this generalization problem.

Another method developed for transfer learning by Parisotto et al. trains a mimic network to do various pre-defined tasks and then uses this network to initialize a deep Q-learning (DQN) agent on a novel task [4]. Multiple DQNs are trained on multiple source tasks; the DQNs are then converted to a policy, and then the probabilities are used as target labels for a supervised learning problem. The mimic network is therefore trained using supervised learning rather than reinforcement learning. While this approach efficiently uses data generated by the various trained DQNs, it does not scale well to tasks where large numbers of DQNs are required.

An orthogonal approach to hierarchical reinforcement learning is a method of state space decomposition developed by Mnih et al. This work introduces a recurrent model of visual attention, which moves an attention mechanism around the state space before outputting an action [5]. The advantages to this method is that computational time is significantly reduced, since a smaller region of the state space needs to be processed at each time step. Additionally, through its recurrent framework, an agent can learn to synthesize its knowledge of what it has experienced in the past to obtain a more complete understanding of its environment. One drawback is that unnecessary repetitive saccading of the attention window may occur at each time step. Because the environment marginally changes at each time step, it is advantageous to alter the time scale of the attention framework when applying it to hierarchical reinforcement learning tasks. We employ this concept of an attention mechanism in our work, since this is an effective way to decompose the state space; however, we allow the attention window to move only once per iteration in each episode so that repetitive movement is eliminated. Specifically, our meta-controller outputs an attention action at each time step, which then changes the subsection of environment experienced by the agent at the next time step.

The works discussed above address transfer learning and state space decomposition where no human input is included. The work of Griffith et al. approaches the problem of speeding up learning by including human feedback and creating an algorithm that incorporates that feedback in an optimal manner [8]. This approach is another way to address the problem of computational efficiency in reinforcement learning. In summary, state-of-the-art reinforcement learning algorithms do not allow an agent to easily transfer its knowledge of how to do one task to another. Our work addresses this gap in the literature by combining components of previous approaches to the problem.

Materials and Methods

Environment

For our experiments, we use an environment consisting of a 10x5 grid. The grid consists of four "rooms," where each room is a horizontal *k* x 5 strip for some k not exceeding 4. The rooms are stacked on top of each other and are each a different color from the set *{red, green, blue, yellow}*. The environment also generates an instruction as a one-hot vector of length 4, specifying the target room. An episode terminates when either the agent reaches the target room, receiving a positive reward of +1, or it times out without reaching it. The step cost is −0.01.

Approach

We construct three frameworks for the meta-controller that is tasked with providing sub-goals to the underlying agent such that it navigates successfully to the target room. In all experiments, the meta-controller uses the Adam Optimizer with a learning rate of $1 \times e^{-5}$. The agent and attention (if used) always start at the top of left corner of the grid. We train the networks using policy gradients, as these methods demonstrate low bias in the training of reinforcement learning algorithms as opposed to off-policy methods such as DQN. This is extremely important in the context of long-term dependencies introduced by the recurrent attention mechanism.

Meta-Controller with No Attention Mechanism

First, we simplify the problem by providing the entire state space as input to the meta-controller at each time step, therefore not employing any attention mechanism. Specifically, the meta-controller receives as input the 10x5 image of the grid which contains the rooms as well as the location of the agent, L^{agent}. The output of the meta-controller is a distribution *P(g)* over the rooms. A room is sampled from *P(g)* and is provided to the base agent as an instruction. This is the sub-goal it must achieve. It is assumed that the underlying agent can move optimally on the entire *10x5* grid given an instruction. In this setup, the optimal policy for the meta-controller is to always output the target room directly.

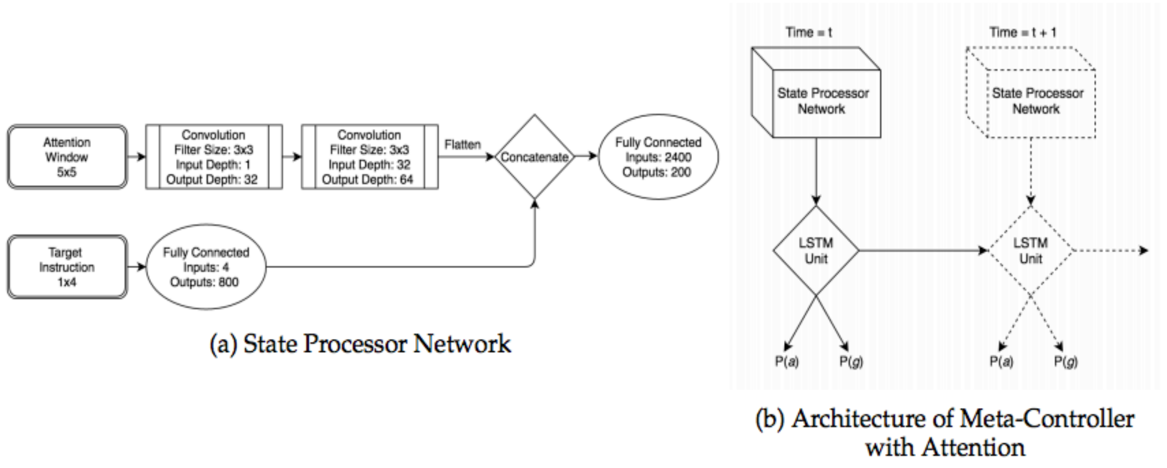

Attention Window 5x5
Convolution Filter Size: 3x3 Input Depth: 1 Output Depth: 32
Convolution Filter Size: 3x3 Input Depth: 32 Output Depth: 64
Flatten
Concatenate
Fully Connected Inputs: 2400 Outputs: 200
Target Instruction 1x4
Fully Connected Inputs: 4 Outputs: 800
(a) State Processor Network
Time = t
Time = t + 1
State Processor Network
State Processor Network
LSTM Unit
LSTM Unit
P(a)
P(g)
P(a)
P(g)
(b) Architecture of Meta-Controller with Attention

Figure 1. Network architecture for the partial state decomposition and constrained attention mechanism experiments. The 5x5 attention window and target instruction are inputted at each time step into the meta-controller, which then outputs probability distributions over attention actions and sub-goal instructions.

Meta-Controller with Partial State Decomposition

In this setup, we have a meta-controller with an attention mechanism, which consists of a 5x5 window into the 10x5 grid. In addition to the sub-goal instruction, it must now also output an action to control the attention. Here, the attention mechanism only partially decomposes the state space, meaning that the agent may move optimally to a provided sub-goal, even if the agent is not within the current attention. The sub-goal, however, must be located inside the attention. The goal of the meta-controller is to find the location of the target room using its attention mechanism and then instruct the agent to go to that room color at every time step. The architecture of this meta-controller consists of a state processor network, which takes the 5x5 attention window and target instruction provided by the environment as input at each time step. It processes these inputs using a feedforward convolutional network and uses an LSTM unit to output *P(g)* and *P(a)*, where *P(a)* is the probability distribution over attention actions. The convolutional layers of the network use rectified linear unit activation functions. The attention action affects the next attention location, L^{attn}, while the sub-goal instruction affects the next agent location, L^{agent}. Thus, the LSTM's hidden state contains knowledge gained from taking a sequence of both instruction and attention actions in an episode. In order to train this network effectively with Policy Gradients, we assume that the attention and instruction actions probabilities are independent of each other.

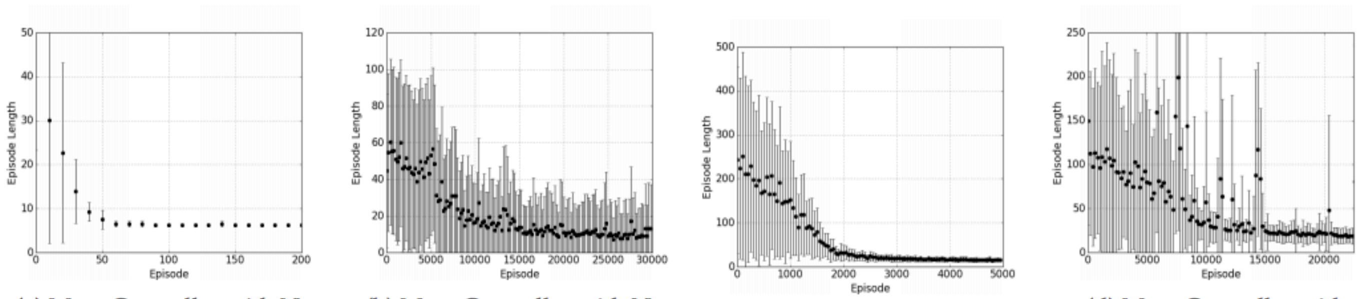
Meta-Controller with Constrained Attention Mechanism

In this setup the agent will not move unless it is within the attention. This means that the meta-controller must instruct the agent to move to a room within the view of the attention before moving downwards and repeating the process until the agent has reached the target room. Thus, if the agent or the sub-goal do not appear within the decomposed state space, the target task will not be achieved. In both the partial state decomposition and constrained attention mechanism experiments, the LSTM unit allows the meta-controller to use its memory of the locations of the agent and target room to guide its action selection when either the agent or target room is not present within the attention window at a particular time step. The constrained attention mechanism framework adds the additional step of constructing an optimal sequence of sub-goals for the agent to reach the target room, and this is the overall goal of this thesis.

Results

Meta-Controller with No Attention Mechanism

We ran two experiments using the meta-controller with no attention mechanism. In the first experiment, the environment is fixed, i.e. the room arrangement is fixed between episodes, and the target room is always at the very bottom. The optimal policy of the meta-controller is simply to output the instruction corresponding to the target room, since the underlying agent is an optimal agent on the entire grid. In the second experiment, the environment is dynamic, which means that the room arrangement is randomly generated between episodes, but the target room is always located at the very bottom. Here, it must learn a mapping between the color of the bottom-most room and the optimal instruction. These experiments serve as a baseline for the experiments that we run using the meta-controller with the attention mechanism. For both cases, the meta-controller converges to the optimal policy. This policy allows the higher level agent to guide the agent to the correct room.



(a) Meta-Controller with No Attention Mechanism on a Fixed Environment

(b) Meta-Controller with No Attention Mechanism on a Dynamic Environment

(c) Meta-Controller with Partial State Decomposition

(d) Meta-Controller with Constrained Attention Mechanism

Figure 2. Episode lengths over training episodes. The dots represent means over a variable number of episodes dependent on the total number of episodes displayed and the whiskers represent the variance. In the case of the meta-controller with no attention (a and b), it converges quickly on a fixed environment, supplying the agent with the target room and hence completing episodes in the minimum time. It takes longer to converge in the dynamic case. The meta-controller with attention plots (c and d) show the results on the effect of using attention to guide sub-goal creation. For these experiments, the environment was kept fixed.

Meta-Controller with Attention Mechanism

Here, we show the results for the meta-controller with both the partial state decomposition and the constrained attention mechanism. In these experiments, we keep the environment fixed between episodes. Compared to Figure 2a, in both cases, it takes longer to train the meta-controller to output sub-goals leading to the optimal policy. One reason for this is that the meta-controller now has to control its attention in addition to creating sub-goals. It is also operating in a partially observed setting and has to integrate information gleaned in past attentions into its hidden state. Note that the state space of the meta-controller is the set of combinations of attention window, target instruction, and hidden state of the LSTM unit. But since this is possible to learn, the underlying agent does not have to be trained on the entire 10x5 image, but only on the 5x5 attention size. Our approach may scale to even larger sized domains, where directly learning a policy on the original input image may be infeasible.

Discussion

Our overall contribution is a framework that allows an agent to complete a task in a large environment given knowledge of how to do so in a smaller environment. Through the use of an attention mechanism, smaller networks are required, which are easier to train. This approach addresses the computational efficiency problem that would occur when retraining an agent from scratch on the larger environment.

With all three frameworks developed, the meta-controller learns the representation of the room colors and how that representation transfers to sub-instructions that lead the agent to the desired goal. Our results show that it is possible to scale a policy learned on a smaller environment by decomposing a large state space using an attention mechanism, which addresses the problem of transferring skills learned on one task to another.

This research fits into the larger mission of creating AI agents that can generalize knowledge that they learn on specific domains. HRL can be combined with other machine learning approaches in areas such as Computer Vision and Natural Language Processing (NLP) to develop cognitive models that can use contextual knowledge of their environments to exhibit broad intelligence. Recent work in MIT's NLP group [9] uses language to performing conceptual grounding in order to achieve skill transfer across tasks. In the computer vision domain, [10] and [11] learn reinforcement learning policies in simulations that can then be used by robots to complete tasks in the real world, which is an example of transferring the input data that the agent sees. Lastly, learning invariant feature spaces to transfer skills, such as the work of [12], is a research topic of recent prominence. Synthesizing our approach of decomposing the state space of a reinforcement learning agent with these contributions offers a promising path to creating AI with broad competence.

Future Work

Our eventual goal is to train the underlying agent in conjunction with the meta-controller and apply this framework to dynamic and complex environments. The environment used in the experiments in this work are static and simplistic, and it is essential to verify that this attention-based training procedure can generalize to other environments. Another limitation of this work is that the attention mechanism size is fixed and pre-determined, whereas the ability for the reinforcement learning agent to control its own view of the environment would be more optimal.

References

1. Kulkarni, T. D., Narasimhan, K. R., Saeedi, A., & Tenenbaum, J. B. (2016). Hierarchical Deep

   Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. *arXiv preprint.*

2. Bacon, P., Precup, D., and Harb, J. The option-critic architecture. In AAAI , 2017.

3. Vezhnevets, A. et al. FeUdal Networks for Hierarchical Reinforcement Learning. *arXiv preprint.*

4. Parisotto, E., Jimmy, L., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint*. arXiv:1511.06342 (2015).

5. Mnih, V., Hees, N., Graves, A., and Kavukcuoglu, K. Recurrent models of visual attention. In *NIPS,* 2014.

6. Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning – Volume 37,* ICML'15, pages 1312-1320. JMLR.org, 2015.

7. Sepp Hochreiter and Jurden Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735-1780, November 1997.

8. Griffith, S., Subramanian, K., Scholz, J., Isbell, C., and Thomaz, A. L. (2013). Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems,* pages 2625–2633.

9. Narasimhan, K., Barzilay, R., & Jaakkola, T. (2017). Deep Transfer in Reinforcement Learning by Language Grounding. *arXiv preprint arXiv:1708.00133*.

10. Devin, C., Gupta, A., Darrell, T., Abbeel, P., & Levine, S. (2017, May). Learning modular neural network policies for multi-task and multi-robot transfer. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on* (pp. 2169-2176). IEEE.

11. Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, *17*(39), 1-40.

12. Gupta, Abhishek, et al. "Learning invariant feature spaces to transfer skills with reinforcement learning." *arXiv preprint arXiv:1703.02949* (2017).