Understanding Successive Features via Random Low Rank Structure

Vasco Portilheiro and Eric Luxenberg

1 Introduction

We consider the standard reinforcement learning formulation of an agent interacting with an environment $\mathcal{E} = (\mathcal{O}, \mathcal{A}, \mathcal{P}, \rho)$, where the elements of the tuple are, respectively, the observation set, action set, transition probabilities, and distribution of initial observation. The agent acts via a policy $\pi : \overline{\mathcal{T}} \to \Delta(\mathcal{A})$, where $\overline{\mathcal{T}}$ is the set of (non-terminal) *histories*, i.e. sequences of observation-action pairs $\{(O_t, A_t)\}_{t=0}^{\tau-1}$ until termination time τ, and Δ indicates the set of all probability distributions on its input. In a reinforcement learning problem, we are also given a reward R_t at each time step, depending on the pair (O_t, A_t), which can be seen as internal to the agent, describing that agent's preferences. The learning problem itself is for the agent to learn a policy π that attempts to maximize the expected per-episode cumulative reward $\mathbb{E}\left[\sum_{t=0}^{\tau} R_t | \mathcal{E}, \pi\right]$.

2 Research Question

The overarching research program for this project is the question: *how can an agent intelligently and efficiently learn to represent world?* We can easily construct problems where an agent can take an action that will never lead to a reward later in the episode, but will reveal something critical about the environment, allowing the agent to attain rewards in later episodes. It is these settings, along with the intuition that an agent who understands the mechanics of the world they inhabit can perform more optimally, that motivate us to consider learning a representation of the world.

Adding inductive bias to an agent by having it learn to predict information about the world given a state-action pair, in addition to the state-action value function, has proven a successful way to accelerate and improve learning of the main task. One example is learning "successor features," which capture the expected state occupancy of the agent given some state [1, 4]. Another fruitful avenue has been predicting auxiliary information that is in a sense "local" to the current state-action, such as the reward observed in the next transition, the next state, or an auto-encoder-like recovery of the current state given the learned feature vector [2, 3]. Such ideas are generally referred to as "auxiliary tasks" or "generalized value functions," and have been shown to be effective ways to accelerate learning. The intuition is that for an agent to do well in these auxiliary prediction tasks, it has to develop a good way of understanding (i.e. featurization of) the environment. Note that to distinguish such

general additional predictions used to train the agent from the precise meaning of "successor features" of [4], we refer to them as *successive features*.

The formal mechanism by which, and situations in which predicting successive features helps an agent learn are not well understood. This is the crux of our research question for this project: *under which situations does predicting successive features help an agent learn?* Our hypothesis is that successive features help when the observations generated by the environment are complex, and do not in a sense "immediately reveal" the information important for solving the problem.

As an operational instantiation of this idea, we consider the case when an environment has "true features," which are necessary and sufficient for solving the reinforcement problem well using a simple approach such as DQN. For example, for the classic Cart-Pole problem, cart position-and-velocity and pole position-and-velocity can be seen as "true features." We then perturb these features by mapping them via a random matrix to a high-dimensional space (thus making the perturbed features low-rank). We note that while it is sufficient for an agent to know how invert the matrix in order to recover the true features, it is not clear that a general neural network will be able to learn this. We ask whether adding successive features helps an agent learn in this simple version of a "complex environment."

3 Relevant work: Auxiliary tasks

Auxiliary tasks articulate the idea that an agent that is able to make refined predictions about other aspects of its environment should be better suited to adapt to its ultimate reward-maximization task. An elaborate example is the HORDE architecture, introduced by Sutton et. al. in 2011 [5], which, along with multiple variations such as many agents, introduces the notion of a generalized value function. A generalized value function extends the normal notion of state-action value function, and is parameterized by a pseudo-reward function, a discount parameter, and a policy. The generalized value function formalizes the idea of an agent having different "questions" which it is trying to answer. As opposed to the main reward function introduced with the definition of the reinforcement learning problem, a generalized value function is a way to capture rewards intrinsic to the agent that help it understand its environment better, and eventually perform better on the main task. Specific examples of such auxiliary tasks are changing the color of regions of the screen in a game, moving a finite distance in a control task, and many other sub goals [3].

4 Method

4.1 Environment

The Cart-Pole environment is a standard RL problem. An agent controls the left-right acceleration of a frictionless cart with a pole attached to a hinge on its body, seeking to balance the pole. This is a common RL baseline problem a complete description of world is contained in the standard features of angular and translational position and velocity (the "true features" mentioned in §2), and a relatively simple agent can obtain near perfect performance. Additionally, it is a expressive problem via its variants. By modulating the

initial position of the pole (upright, or needing to be "swung up" from downward-hanging), and the reward structure from dense (proportional to angle from the downward positing) to sparse (only when the pole is balanced), the problem can be made more difficult and interesting.

4.2 Input perturbations

Our general perturbation procedure is as following. Given that the true features lie in $\mathbb{R}^D$, we generate a matrix $A \in \mathbb{R}^{N \times D}$ with i.i.d. Gaussian entries, where $N \geq D$. True features ϕ are then mapped to the new, higher dimensional features $A\phi$. We emphasize that the sampling of A occurs when the RL algorithm begins, and A remains constant thereafter throughout the learning process.

In the case of Cart-Pole, the true feature dimensionality is $D = 4$. We run experiments with $N = 4$, $N = 10$, and finally $N = 20$.

4.3 Agent

We compare two different agents: a vanilla DQN agent (which can be seen as a base network with a head predicting the Q function), and that same agent with additional heads added to the network, each predicting a separate successive feature — one of: next reward, next observed state, and a head that attempts to reconstruct the current input state (Figure 1). Note that the heads output one prediction per possible action, thus allowing the network to model a function from state-action pairs to each of the outputs. Note also that in this context, we use "state" to denote the input to the base network, i.e. the perturbed true features.

We use this architecture with the base outputting hidden features of dimension 50, and with each head itself having a hidden layer of size 50. We train the network by using as our loss the sum of losses for each head, scaled to be of the same order. Furthermore, each of the heads except for the Q head (i.e. all the "successive features") have and additional L2 regularization loss with scale 0.01.

5 Results

We run experiments for each setup for a number of different random seeds. Our main results are reported in Figure 2, both for each seed, and averaged across seeds. (See the caption for explanation of the columns.)

In general, we find that adding successive features seems to improve the agents' cumulative score, as well as the other desirable properties tallied in the columns. The improvement due to successive features is generally more drastic in swing-up case, which may suggest that random feature perturbations exacerbate the need to learn to understand the environment in more complex tasks. This interpretation, namely that the multi-head agents learn better, seems supported by comparing the learning curves of two kinds of agents (Fig. 3, Fig. 4).

Although our results suggest that the auxiliary predictions did help the agent learn faster, it was not the clear binary distinction that we had hoped to observe. Ideally, we

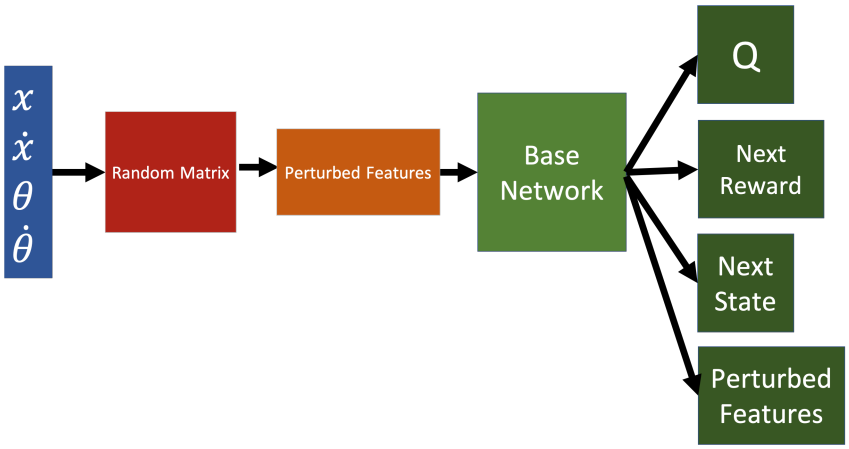

Figure 1: Multi-Headed Network Architecture. The heads displayed below the Q head can be removed, thus turning this into a vanilla DQN.

	4 dimensions				10 dimensions				20 dimension			
	score	bal	spin	>0°	score	bal	spin	>0°	score	bal	spin	>0°
All heads												
Seed 1	351.10	x	√	√	1292.52	√	√	√	-8.04	x	√	x
Seed 2	141.13	x	√	√	613.37	x	√	√	94.93	x	√	√
Seed 3	216.22	x	√	√	86.02	x	√	√	462.45	x	x	√
Seed 4	822.21	√	√	√	388.41	x	x	√	23.85	x	√	√
Seed 5	486.98	x	x	√	306.03	x	x	√	421.14	x	x	√
Avg.	**403.53**	**x**	**√**	**√**	**895.45**	**x**	**√**	**√**	198.86	x	**√**	√
Only Q head												
Seed 1	35.09	x	√	√	265.80	x	x	√	530.83	x	x	√
Seed 2	147.23	x	x	√	374.12	x	x	√	36.59	x	√	x
Seed 3	86.78	x	√	√	253.24	x	√	√	453.65	x	x	√
Seed 4	234.15	x	x	√	573.40	x	x	√	156.09	x	x	√
Seed 5	-6.02	x	√	x	14.63	x	√	x	181.05	x	x	√
Avg.	99.45	x	√	√	296.24	x	√	√	**269.64**	x	√	√

(a) Results for Swing Up Problem

	4 dimensions				10 dimensions				20 dimension			
	score	bal	spin	>0°	score	bal	spin	>0°	score	bal	spin	>0°
All heads												
Seed 1	654.27	√	√	√	1493.59	√	√	√	810.88	√	x	√
Seed 2	1480.69	√	√	√	714.77	x	x	√	358.53	x	x	√
Seed 3	119.14	x	x	√	1362.23	√	√	√	1488.38	√	√	√
Avg.	**751.37**	√	**√**	√	**1,190.20**	**√**	**√**	√	**885.93**	√	x	√
Only Q head												
Seed 1	817.19	√	√	√	1494.15	√	√	√	657.10	x	x	√
Seed 2	315.23	x	x	√	523.06	x	x	√	324.75	√	√	√
Seed 3	904.36	√	x	√	343.79	x	x	√	1294.99	√	√	√
Avg.	678.92	√	x	√	787.00	x	x	√	758.95	√	**√**	√

(b) Results for Upright Problem

Figure 2: Experimental results. The non-"score" columns are as follows: "bal" refers to the agent balancing the pole for at least ~ 0.5 seconds; "spin" has a check if the agent did *not* rapidly spin the pole; "$> 0°$" has a check if the pole was ever above zero degrees agove the horizontal.

would have been able to show that the introduction of the random matrix caused the DQN agent without auxiliary predictions to be unable to learn. However, we were not able to show this definitively. A key reason for this was the difficulty in comparing neural network architectures. It is not clear whether the correct way to compare our architecture to that of a vanilla network is to prioritize equating the total number of neurons, the maximum hidden layer width, or the number of layers. We performed some preliminary experiments with a wider base network, and found that this version of vanilla DQN performed as well as, if not better than, the multi-headed network. It is known that wider networks are better function

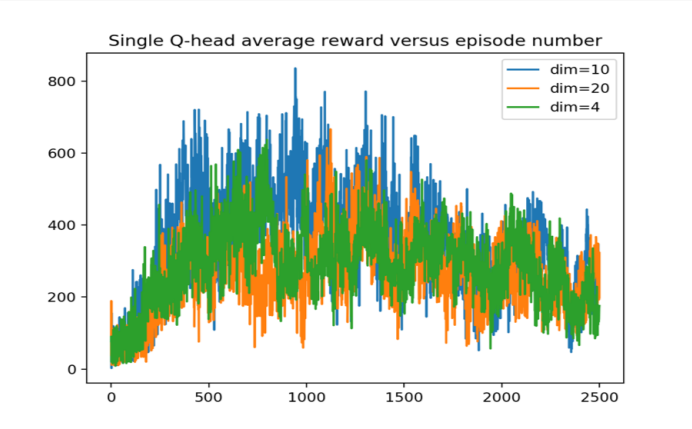

Figure 3: Single Q-Head Network Average Reward in the Swing-Up environment

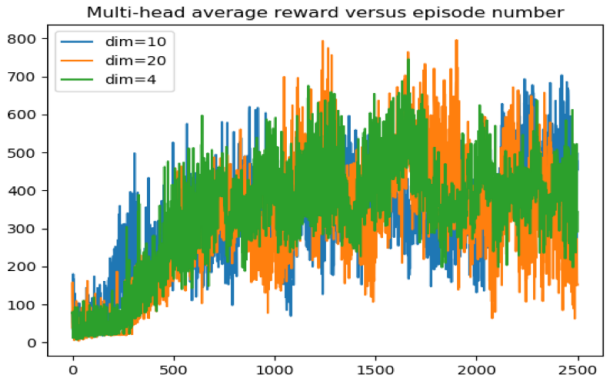

Figure 4: Multi-Head Network Average Reward in the Swing-Up environment

approximators, so if we are comparing our multi-headed network to a vanilla network with the same number of parameters but wider layers, we cannot be sure if the vanilla networks' higher performance is due to the wider layer, or because successive features don't matter in the way we hoped they would.

Despite these considerations, it is clear that our current formulation of the random matrix problem modulation does not transform a problem such that it cannot be solved without successive features. It makes sense that a simply larger network will be able to do a better job identifying the linear subspace to the features are confined. As such, while the results are encouraging, we were not able to fully address our proximal research question, which was if this particular problem augmentation would be *only* compatible with successive features and not other techniques.

6 Future Directions

One question we are interested in is understanding feature learning in combination with deep exploration. Although our random matrix feature perturbation framework above isn't

a perfect "switch" for turning on the need for representation learning, it may still prove useful as a simple way to study this question, e.g. by turning to the sparse-reward version of Cart-Pole. Another empirical direction we would like to investigate to build more intuition is understanding the relationship between the perturbed feature dimension N and the learning and performance of the agent. One way of studying this would be analyzing the rank of the possible outputs of the base network (e.g. via spectral methods). We imagine this could elucidate the effect of our transformation further, since we have observed its effect to not be binary.

A key difficulty that arises in the study of successive features is that simple representations like a tabular agent or a linear agent do not fit into the setting needed for successive features as we understand them: function approximators that contain some internal representation of the inputs. So it appears that any setting in which we study successive features experimentally, we must be using some kind of function approximator, most likely a neural network. An immediate simplification would be considering only one auxiliary head. However, there may still be a lack of interpretability due to the nature of neural networks. We would be interested in exploring whether successive features can be studied through a rigorously interpretable function approximator.

References

[1] Andre Barreto et al. "Successor Features for Transfer in Reinforcement Learning". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4055–4065. URL: http://papers.nips.cc/paper/6994-successor-features-for-transfer-in-reinforcement-learning.pdf.

[2] T. de Bruin et al. "Integrating State Representation Learning Into Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 1394–1401. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2800101.

[3] Max Jaderberg et al. "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: *CoRR* abs/1611.05397 (2016). arXiv: 1611.05397. URL: http://arxiv.org/abs/1611.05397.

[4] Tejas D. Kulkarni et al. "Deep Successor Reinforcement Learning". In: *arXiv e-prints*, arXiv:1606.02396 (June 2016), arXiv:1606.02396. arXiv: 1606.02396 [stat.ML].

[5] Richard S. Sutton et al. "Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS '11. Taipei, Taiwan: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 761–768. ISBN: 0-9826571-6-1, 978-0-9826571-6-4. URL: http://dl.acm.org/citation.cfm?id=2031678.2031726.