Policy Agnostic Successor Features

Norman Tasfi[†]
ntasfi@uwo.ca

Eder Santana[*]
eder.santana@twitch.tv

Miriam Capretz[†]
mcapretz@uwo.ca

† University Of Western Ontario, Canada * Twitch Inc., California, USA

Abstract

Current Reinforcement Learning algorithms cannot efficiently transfer between related tasks within the same environment. The successor feature framework aims to solve this by decomposing the policy into two components: one learning the discounted future expected state feature, called the successor features, of each state feature and one modelling the environment reward structure. In this paper, we make the connection that the successor feature component learned by the agent is an amortized latent state-transition model that conditions on the agent's policy. Instead of learning this coupled function, we show that it is possible to use a state-transition model to compute the successor features dynamically. This decoupling lets us leverage methods from unsupervised and semi-supervised learning to learn the state-transition model. We believe the state-transition model's resulting features will be more generic to the particular policy, possibly helping task transfer. Furthermore, now that the features are decoupled from the policy, we gain immense freedom in learning the reward structure. We plan to evaluate this change in a continuous gridworld environment and robotic reacher, both with several tasks and reward structures.

1 Introduction

Reinforcement learning's (RL) performance has seen leaps and bounds in improvement over the past few years, outperforming human experts on tasks such as Go [1], Poker [2], Atari [3], and chip placement [4]. Deep neural networks, a powerful class of function approximators [5], have primarily fueled by being coupled with RL algorithms producing deep RL algorithms.

However, RL algorithms still lack the ability to transfer their learned policy between related tasks; an ability that deep neural network based image recognition models possess to some extent [6]. RL algorithms often require full retraining on the policy to achieve good performance on a related task. The ability to transfer between tasks improves sample efficiency as the model can reuse previously learned knowledge.

The successor feature (SF) framework [7][8][9] improves transfer between tasks by decomposing the policy into two separate components: one capturing future expected environment dynamics under a given policy, the successor features, and another modelling the task reward. The state-action value is computed as the dot product between the SFs, one per action, and the reward component. During transfer, the successor features are held frozen, and only the reward component's parameters are trained, which require fewer samples by virtue of being small.

Mathematically, we show that the SFs can be dynamically computed using a latent state-transition model given the reward component for a specific task. Solving the RL problem now requires only

Pre-registration workshop NeurIPS (2020), Vancouver, Canada.

learning a state-transition and a small vector that parameterizes the reward component. By decoupling the policy from the SFs, we gain flexibility in the methods we can use to learn our model, allowing us to borrow from the unsupervised and semi-supervised learning literature – with potential for a significant increase in model quality. As a result of more expressive models, we believe that the learned state features will be more general, leading to better transfer across tasks in the same environment than the standard SF framework.

The proposed approach can be easily related to model-based reinforcement learning, which has the goal of learning the world dynamics for enhanced sample efficiency [10][11][12]. Furthermore, model-based RL also has the advantage of being able to leverage offline data or expert demonstrations during learning [13][14]; further increasing sample efficiency. However, possible drawbacks can exist with model-based methods as errors in the latent state representation quality can cause accumulation of errors during the rollout. Models of the environment also add the ability to perform guided exploration using state reconstruction quality as a proxy to novelty and state visitation probabilities [15]. Therefore, we believe state-transition models will expand what was previously possible within the Successor Feature framework.

The contributions we wish to make in this paper are as follows:

- Provide evidence that a state-transition model can dynamically produce successor features that are equivalent in quality to the amortized version.
- Show that dynamic successor features provide many benefits, such as increased generality.
- Demonstrate it is possible to leverage the unsupervised and semi-supervised learning literature to strengthen the learned state-transition model.

We organize our paper as follows: Section 2 provides an overview of background material, Section 3 describes the proposed changes to the successor framework and how it is training, and Section 4 discusses the experiments we hope to perform to validate our changes.

2 Background

In this section, we formally define the RL problem, then provide a derivation of the Successor Features, and finally introduce our modification to the SF framework.

We assume an agent interacts with an environment according to a Markov Decision Process (MDP) [16]. At each timestep t the agent observes a state $s_t \in \mathcal{S}$, where $\mathcal{S}$ are the set of all available states. From state s_t, the agent selects an action a_t from the set of all available actions $\mathcal{A}$ and executes it in the environment. The environment transitions to a new state s_{t+1} and emits a reward r_{t+1} according to the environment transition dynamics $P(s_{t+1}, r_{t+1}|s_t, a_t)$. In this paper, we focus on tasks and environments where only the rewards are statistically independent of the previous state, being entirely defined by the resulting state only, thus focusing on $P(s_{t+1}, r_{t+1})$. Ma *et al.*[9] show several tasks where this holds true. The objective of the agent is to find a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ which maximizes the expected value of the discounted sum of rewards in the environment G_t, defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

where $\gamma \in [0, 1]$ is the discount factor that trades off the importance of immediate and future rewards. A policy π can be derived from the state-action value function, the Q-Learning function, which is defined as:

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}\left[G_t | s_t = s, a_t = a\right], \tag{1}$$

which provides the future expected reward of selecting any action $a \in \mathcal{A}$ and then following the policy π afterwards; such that $\pi(s) \in \text{argmax}_a Q^{\pi}(s, a)$.

The Successor Feature framework decomposes the Q-Learning function into two components. The decomposition is accomplished by first assuming that the reward r_t is computed as the dot product between state features $\phi_t \in \mathbb{R}^z$, extracted from s_t using an encoder $\text{E} : \mathcal{S} \rightarrow \Phi$, and a vector $w \in \mathbb{R}^z$.

Where z is a hyperparameter that controls the embedding dimension of the vectors. The vector w models the reward of the environment, and in this work, we assume, once decoupled, that it can induce policy π through $\psi(s, a)$.

Therefore, following from the assumption that $r_t = \phi_t^\top \cdot w$ and the definition of the Q-Learning function in Equation 1, we can see the following derivation:

$$Q^\pi(s,a) = \mathbb{E}^\pi \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t = s, a_t = a \right] \tag{2a}$$

$$= \mathbb{E}^\pi \left[\phi_t^\top \cdot w + \gamma \phi_{t+1}^\top \cdot w + \dots | s_t = s, a_t = a \right] \tag{2b}$$

$$= \psi^\pi(s,a)^\top \cdot w, \tag{2c}$$

where the vector $\psi^\pi(s, a) \in \mathbb{R}^z$ is referred to as the successor features under policy π, with one vector defined *per* action. The i-th component of $\psi^\pi(s, a)^{(i)}$ predicts the future expected discounted sum of the feature $\phi_t^{(i)}$ that the policy π visits. As mentioned above, because we are focusing our research on tasks where the reward r_t is a function of only the current state s_t and not the state-transition: $\mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S}'$. Therefore, the proposed method follows the reward formulation proposed by [9]. Likewise, the proposed method assumes that the extracted features ϕ_t are a function of the state s_t only and do not rely on the transition tuple.

Under this decomposition, transfer between new tasks requires retraining of just the smaller reward vector w. While keeping the state features ϕ and successor features ψ^π frozen, reusing previously learned knowledge. Unfortunately, the successor features learn an amortized function for a specific policy π. We believe that the successor features ψ^π and encoder producing ϕ are less general as a result – possibly hurting transfer between related tasks.

3 Policy Agnostic Successor Features

We believe that by decoupling the successor features from the policy, we will gain a myriad of benefits, such as more robust generalization capabilities to both related tasks and policies. More exactly, we propose learning a latent state-transition model that can dynamically bootstrap successor features irrespective of the policy given. Within this work, we make the assumption that ϕ_t, and therefore the reward r_t, are predicted solely from the current state s_t. We begin by revisiting the definition of the successor features ψ:

$$\psi^\pi(s,a) = \mathbb{E}^\pi[\phi_t + \gamma \phi_{t+1} + \gamma^2 \phi_{t+2} + \dots] \tag{3}$$

where we observe that the visited latent states within the expectation depends on the particular policy π and that Equation 3 can therefore be rewritten in terms of a chain of state-transitions dictated by said policy. Defining Equation 3 in terms of state transitions requires access to a model of the environment that can predict the next latent state ϕ_{t+1} given the current latent state ϕ_t and an action a. In this work we refer to this model $\mathcal{M}$ as the latent-state transition model and define it as:

$$\mathcal{M} : (\phi_t \in \Phi) \times (a_t \in \mathcal{A}) \mapsto (\phi_{t+1} \in \Phi) \tag{4}$$

where Φ is the set of all encoded valid states $\mathcal{S}$. Following from this definition, Equation 3 can now be rewritten with the latent state-transition model $\mathcal{M}$ as follows:

$$\psi(s, a, \pi, \gamma) = \phi_t + \gamma \mathcal{M}(\phi_t, a) + \dots \tag{5}$$

where ψ is now accepts additional arguments such as the policy π and discount factor γ. Rewriting Equation 5 with a summation and k length rollout:

$$\psi(s, a, \pi, \gamma) = \phi_t + \gamma \mathcal{M}(\phi_t, a) + \sum_{j=1}^{k-1} \gamma^{j+1} \mathcal{M}(\hat{\phi}_{t+j}, \pi(\hat{\phi}_{t+j})) \tag{6}$$

where we define $\hat{\phi}_{t+i+1} = \mathcal{M}(\hat{\phi}_{t+j}, \pi(\hat{\phi}_{t+j}))$ and $\hat{\phi}_{t+1} = \mathcal{M}(\phi_t, a)$. Equation 6 implies that we can compute the successor features dynamically given a latent state-transition model, a discount

factor γ, and a policy π induced by w with a previous estimate of ψ from the last rollout step. The policy π is bootstrapped before each step in the environment by gradually unrolling the latent space with $\mathcal{M}$ to construct ψ, as shown in Equation 6. Comparing Equation 3 & 6, we can see that the original function formulation of ψ is learning an amortized version of this rollout under a specific policy and discount factor.

The final Q-Learning function in Equation 2, can now be dynamically inferred based on the specific policy and discount factor by unrolling $\mathcal{M}$ for k steps:

$$Q(s, a, \pi, \gamma) = \left\{\phi_t + \gamma\mathcal{M}(\phi_t, a) + \sum_{j=1}^{k-1} \gamma^{j+1}\mathcal{M}(\hat{\phi}_{t+j}, \pi(\hat{\phi}_{t+j}))\right\}^\top \cdot w \tag{7}$$

Again, we note that this contrasts with the traditional phrasing of the SF framework, where the successor features learn an amortized inference of the Q-function in (2c), which we believe couples the learned features and policy together – limiting flexibility. Further, we see that RL can now be accomplished without explicitly using temporal difference learning.

3.1 Training

We expect to train the proposed model under the same training regime as used in previous works in the successor feature framework. That is, we will train our model following the Deep Q-Learning methodology [3]. Our experiments will have a single ϵ-greedy policy interacting with an environment and updating the models with transitions sampled from a replay memory at fixed intervals. We intend to have at least three broadly defined components in our architecture: a reward component, a state encoder $\mathcal{E}$ that produces the latent state ϕ_t from the raw state space s_t, and state-transition model $\mathcal{M}$.

The reward component will simply be the vector $w \in \mathbb{R}^z$ which we train to minimize the following loss:

$$\mathcal{L}_r = (r_t - \phi_t^\top \cdot w)^2 \tag{8}$$

At this time, there is uncertainty in what is the best way to train and structure the state encoder $\mathcal{E}$ and state-transition model $\mathcal{M}$ which we hope to resolve through experimentation. The state encoder $\mathcal{E}$ needs to produce a suitable representation of the state which is useful for downstream tasks. However, we can say with certainty that the state encoder will either be a fully connected network or a convolutional network depending on the task. The state-transition model structure will have much greater variability in its structure. We will evaluate different models such as those using a purely feed forward network [15], a residual network [17, 18], recurrent networks [15, 19, 20], or even generative models [21].

Auxiliary tasks are often included when learning the encoder $\mathcal{E}$, as done several related works within the SF framework, such as a encode-decode task [8, 9, 22]. This task pairs the encoder model $\mathcal{E}$ with a decoder $\mathcal{D} : (\phi_t \in \Phi) \rightarrow (s_t \in S)$ which takes the latent representation ϕ_t produced by the encoder and predicts the current state s_t. The encode-decode task learns to minimize the following loss:

$$\mathcal{L}_{ed} = (s_t - \mathcal{D}(\phi_t))^2 \tag{9}$$

where $\phi_t = \mathcal{E}(s_t)$. As learning a well formed latent representation ϕ is important we might consider investigating other auxiliary tasks such as contrastive learning [23, 24, 25] in our experiments. The state-transition model $\mathcal{M}$ will be trained to predict the next latent state ϕ_{t+1}, and is trained to minimize the loss:

$$\mathcal{L}_{\mathcal{M}} = (\phi_{t+1} - \mathcal{M}(\phi_t, a_t))^2 \tag{10}$$

where $\phi_t = \mathcal{E}(s_t)$, $\phi_{t+1} = \mathcal{E}(s_{t+1})$, and a_t is the action taken at time t. Gradients will be back-propagated through ϕ_t but not the target variable ϕ_{t+1}. We also plan to investigate additional auxiliary tasks that can be used to train a stronger state-transition model $\mathcal{M}$, such as predicting the reward r_{t+1} from the predicted state ϕ_{t+1}.

The manner in which the losses are combined and trained is also an area we will need to verify experimentally. Typically in the SF framework, the loss used to train successor features ψ are trained in alternating steps from the reward and encode-decode losses. This is primarily done to improve stability of the induced policy during learning and environment interactions. As our proposed method decouples the policy from the successor features and no longer relies on learning an amortized

approximation of the ψ quantity we believe it might be possible to train all our components together end-to-end at once.

Therefore, during the model update step we tentatively plan to minimize the following loss:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_{ed} + \mathcal{L}_{\mathcal{M}} \tag{11}$$

Additionally, as our model now learns in an unsupervised/semi-supervised manner, from randomly sampled batches of data, it might be possible use the DQN methodology without having a frozen set of weights to stabilize learning.

4 Proposed Experiments

This section discusses the experiments we wish to use to evaluate our dynamic successor feature model. We identify roughly three areas that our experiments must cover: equivalency, benefits, and ablations. Our first goal is to prove that our method is equivalent to the original formulation with the amortized function. However, multiple axes must be considered, such as overall performance, training time, sample efficiency, and ability to transfer.

The second area we will examine is what benefits this formulation provides. We wish to examine and understand the state features learned by the model, if greater performance can be achieved by using cutting edge supervised learning methods such as contrastive learning, and if our method leads to greater generalization between tasks.

The final area of experimentation will focus on the ablations of this methodology. This area will focus on understanding what effects our model's hyperparameters have on its performance and if we can gain insight into its performance.

4.1 Environments

Below we provide a general description of environments where we will conduct the investigations discussed above. A consideration in the selection of the environments was made such that the reward r_t can be predicted from the current state s_t.

Axes: This environment will feature an agent traversing around a 2d type gridworld where it must reach a goal location specified by a marker. Different tasks are described by varied goal locations where we will split tasks between the train and test sets. The raw state space will include the distance between the agent and every goal available. Therefore, the optimal reward vector for a particular task would be a 1-hot encoding. Two termination conditions will exist in this environment, either the agent reaches the marked goal or goes over a max number of steps. The agent will be able to traverse left, up, right, and down. The reward will be the negative distance between the agent and the reward. In this environment, the state and reward are continuous. Additionally, as this environment is quite simple, we should have a ground-truth state-transition model available for experimentation. We will compare both hard coded and learned state features in this environment.

Reacher: This environment is the reacher task from the Deepmind Control suite [26] based on the MuJuco software package [27]. In this environment, the agent must control a two-joint torque-controlled robotic arm to a specific goal location. We will define separate train and test tasks as [22]. The reward will be defined as the distance between the robotic arm's head and the current goal in the environment. We define two termination conditions: if the arm's end is within a fixed distance of the goal or if the agent has used too many steps in the environment. The state features are learned in this experiment.

4.2 Experimental Details

We will compare our proposed model to the traditional successor feature framework as a baseline model in all experiments. Both versions will follow a similar training procedure, where we train the models as described by the Q-Learning method in Barreto *et al.*[22]. Each experiment will have at least five runs with varied seeds and report the average performance with confidence intervals on all experiments.

We will aim to restrict the number of parameters in our new model such that they are roughly equivalent to the previous formulation.

4.3 Equivalency

We wish to prove that the dynamic successor features are equivalent to the amortized version in this set of experiments and done in varying stages.

We begin by seeing if, given a ground-truth state-transition model and hand-designed reward vector w, the resulting policy is optimal. The next experiment to evaluate is if we can, again given a ground-truth state-transition model, learn the optimal w and, therefore, optimal policy from environment interactions. The final experiments will compare learning both the state-transition model and reward vector w from environment interactions.

Throughout each of these stages of experiments, we will compare each learned component to that of one found in the SF framework's original formulation. Specifically, we will compare the mean absolute percent error between the ground-truth state-transition model and both the dynamic and amortized versions. This will provide information on how strong the compounding rollout error is and if it is small enough to find good policies. We will also examine how well the agent can predict reward throughout the state space from the learned feature vectors ϕ.

To accomplish the aforementioned experiments, we will primarily use the gridworld environment to access a ground-truth state-transition model. Another benefit of the gridworld environment is that it is simple enough to easily visualize the state space in conjunction with another value, such as the reward error.

4.4 Benefits

If the dynamic and amortized versions are equivalent, we then move to examine the benefits of our proposed model. In particular, we are interested in understanding if our proposed variant:

- Does it allow quicker task transfer due to the more general state features?
- Can $\mathcal{M}$ and γ be used guide exploration in the SF framework?
- How well exotic modelling techniques work for the latent state-transition model? (recurrent networks, contrastive approaches, etc.)
- Instead of using a fixed discount factor γ can we use either a dynamic or annealed version?

4.5 Ablations

This area will mostly be carried out in the gridworld environment as we will require access to a ground-truth state-transition model. We are most interested in understanding the following questions with regards to our propose model:

- What value of k for our rollout length, in Equation 6, works best? Is $k = 1$ enough?
- How does the size of the encoder and state-transition model impact performance? Generalization?
- Should $\mathcal{M}$ be trained to optimize $\mathcal{L}_r$ in Equation (8) and help w differentiate reward states or just minimizing $\mathcal{L}_\phi$ is enough?
- Can we train all components together at once?

References

[1] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–503.

[2] Matej Moravčık et al. "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker". In: *Science* 356.6337 (2017), pp. 508–513. ISSN: 0036-8075. DOI: 10.1126/science.aam6960.

[3] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836.

[4] Azalia Mirhoseini et al. *Chip Placement with Deep Reinforcement Learning*. 2020. arXiv: 2004.10746 [cs.LG].

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.

[6] Ali Sharif Razavian et al. "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition". In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. CVPRW '14. USA: IEEE Computer Society, 2014, pp. 512–519. ISBN: 9781479943081. DOI: 10.1109/CVPRW.2014.131.

[7] Peter Dayan. "Improving Generalization for Temporal Difference Learning: The Successor Representation". In: *Neural Comput.* 5.4 (July 1993), pp. 613–624. ISSN: 0899-7667. DOI: 10.1162/neco.1993.5.4.613.

[8] Tejas D. Kulkarni et al. "Deep Successor Reinforcement Learning". In: *ArXiv* abs/1606.02396 (2016).

[9] Chen Ma et al. *Universal Successor Features for Transfer Reinforcement Learning*. 2020. arXiv: 2001.04025 [cs.LG].

[10] E. Todorov and Weiwei Li. "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems". In: vol. 1. July 2005, 300–306 vol. 1. ISBN: 0-7803-9098-9. DOI: 10.1109/ACC.2005.1469949.

[11] C. B. Browne et al. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43.

[12] Nicholas Watters et al. *COBRA: Data-Efficient Model-Based RL through Unsupervised Object Discovery and Curiosity-Driven Exploration*. 2019. arXiv: 1905.09275 [cs.LG].

[13] Eder Santana and George Hotz. *Learning a Driving Simulator*. 2016. arXiv: 1608.01230 [cs.LG].

[14] Rahul Kidambi et al. *MOReL : Model-Based Offline Reinforcement Learning*. 2020. arXiv: 2005.05951 [cs.LG].

[15] Junhyuk Oh et al. "Action-Conditional Video Prediction using Deep Networks in Atari Games". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2863–2871.

[16] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley amp; Sons, Inc., 1994. ISBN: 0471619779.

[17] Gregory Farquhar et al. "Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning". In: *arXiv preprint arXiv:1710.11417* (2017).

[18] Norman Tasfi and Miriam Capretz. "Dynamic planning networks". In: *arXiv preprint arXiv:1812.11240* (2018).

[19] Brandon Amos et al. "Learning awareness models". In: *arXiv preprint arXiv:1804.06318* (2018).

[20] Danijar Hafner et al. "Learning latent dynamics for planning from pixels". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2555–2565.

[21] David Ha and Jürgen Schmidhuber. "Recurrent World Models Facilitate Policy Evolution". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 2450–2462.

[22] 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.

[23] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *arXiv preprint arXiv:2002.05709* (2020).

[24] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. “Curl: Contrastive unsupervised representations for reinforcement learning”. In: *In the proceedings of the International Conference on Machine Learning (ICML)*. 2020.

[25] Jean-Bastien Grill et al. “Bootstrap your own latent-a new approach to self-supervised learning”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[26] Yuval Tassa et al. *dm_control: Software and Tasks for Continuous Control*. 2020. arXiv: 2006.12983 [cs.RO].

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5026–5033. URL: https://ieeexplore.ieee.org/abstract/document/6386109/.