CROSS-STATE SELF-CONSTRAINT FOR FEATURE GENERALIZATION IN DEEP REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Representation learning on visualized input is an important yet challenging task for deep reinforcement learning (RL). The feature space learned from visualized input not only dominates the agent's generalization ability in new environments but also affect the data efficiency during training. To help the RL agent learn more general and discriminative representation among various states, we present cross-state self-constraint(CSSC), a novel technique that regularizes the representation feature space by comparing representation similarity across different pairs of state. Based on the implicit feedback between state and action from the agent's experience, this constraint helps reinforce the general feature recognition during the learning process and thus enhance the generalization to unseen environment. We test our proposed method on the OpenAI ProcGen benchmark and see significant improvement on generalization performance across most of Procgen games.

1 INTRODUCTION

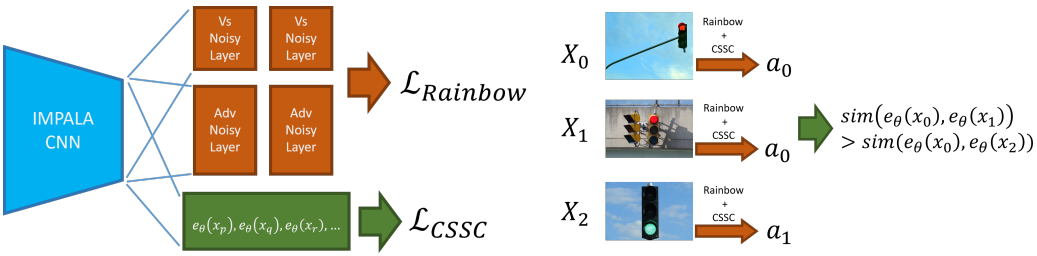
Deep Reinforcement learning has achieved tremendous success on mastering video games(Mnih et al., 2015) and the game of GO(Silver et al., 2017). While training agent by using deep reinforcement learning algorithms, we usually assume that the agent could extract appropriate and effective features from different states and take actions accordingly. However, as more and more research works(Zhang et al. (2018), Song et al. (2019), Dabney et al. (2020)) have pointed out, even well-trained RL agents that learns from visualized input tend to memorizing spurious pattern rather than understanding the essential generic features of a given state. For example, an agent might pay more attention to the change of irrelevant background rather than noticing the obstacles or enemies(Song et al., 2019).

To improve generalization in the new environment, various kinds of regularization method like dropout(Farebrother et al., 2018) and data augmentation(Laskin et al., 2020) has been proposed and tested in combination with reinforcement learning. Conventional methods like dropout and batch-norm has been proven to be effective in supervised-learning, and for self-supervised learning like RL we see multiple related applications across various environments. Data augmentation like random crop(Laskin et al., 2020) or random convolution(Lee et al., 2019) have also been proposed recently and provide considerable generalization enhancement to the unseen levels of various tested environment(Tassa et al. (2018), Cobbe et al. (2018), Cobbe et al. (2020)). The agent is acting on multiple augmented views of the same input and learn from these prior injected data. However, modifying state information(injecting prior to the data) may be risky or even detrimental for representation learning because vital features may be altered or lost (ex: flipping state image might change the corresponding behavioral meaning, cropping the input image might lose critical features like the enemy position in the game).

To avoid losing informative features of the visualized input, we choose a different approach. As a human learner, we rarely depend on multiple augmented views of the same input to discriminate important or fictitious features. Instead, human learners try to recognize general patterns across multiple states and act accordingly. In other words, if the same action(or behavior) has been conducted by a well-trained agent in two different states, we would infer that the agent has conceive similar feature patterns in these states. For example, if one car stops for ninety seconds at two different intersections, we would guess that the car might be stopped by the red light at both places(Figure 1b).

From here we get the intuition about the relation between action(or behavior) and representation feature space learned by the agent: for a RL agent acting rationally across sequence of states, its behavior would be similar if it perceives similar critical patterns while acting differently if it perceives different patterns. Based on this intuition, we designed a novel constraint that perform regularization directly in the representation feature space of the learning agent. Our hypothesis for this constraint is simple: states in which the rational agent behave identically should share more representation resemblance than states the agent behave differently.

We test this novel constraint in combination with Rainbow(Hessel et al., 2017), the state-of-the-art Q-learning method that combines numerous improvements with the original DQN model(Mnih et al., 2015). Inspired by the pair-wise structure used in BPR-opt(Rendle et al., 2012), we design the self-constraint based on our hypothesis and utilize implicit feedback between positive and negative state pairs in the replay buffer. One thing worth noticing is that no change is needed for the underlying RL algorithm or model to adapt the proposed method, and it can be easily applied to other models with minimal effort(Figure 1a).



(a) Rainbow + CSSC model structure

(b) Intuition for cross-state feature matching

Figure 1: Model structure and CSSC concept

To measure the improvement on generalization to unseen environment levels, We test CSSC across 16 games on the OpenAI ProcGen benchmark(Cobbe et al., 2020). We see significant improvement on generalization capability for most ProcGen games in comparison with base Rainbow model.

We highlight the main contributions of CSSC as follows:

- Directly optimize feature generalization across various input states
- Requires no additional modification on the input data and the base model
- Enhance generalization and data efficiency on existing RL methods

2 Related Work

2.1 Auxiliary Task for Representation Learning in RL

Representation Learning has been a vital part for deep RL algorithms. Even though the main goal of RL is to find the optimal value function, Mccallum & Ballard (1996) and Li et al. (2006) show that a representation specialized to this function may not be suitable for the sequence of value functions leading to it. On the other hand, Dabney et al. (2020) argue that the path toward optimal value function might be hindered by overfitting the representation to any intermediate value function during training. One popular and effective method to address this challenge is by using *auxiliary tasks*(Jaderberg et al. (2016), Bellemare et al. (2019)). Dabney et al. (2020) also propose new auxiliary tasks by considering the value-improvement path holistically.

2.2 Regularization on RL

2.2.1 Adding stochasticity to RL

Conventional practices like stochastic policy(Hausknecht & Stone, 2015), random starts(Mnih et al., 2015), sticky actions(Hausknecht & Stone, 2015) and frame skipping are wildly used for popular

tasks like Atari(Machado et al., 2017). By adding stochasticity to the environments during training and testing, we would prevent simple algorithms like *trajectory tree*(Kearns et al., 1999) and *brute*(Machado et al., 2017) from optimizing over open-loop sequences of actions without even considering the input states. However, as Zhang et al. (2018) has pointed out, injecting stochasticity to the maze-like environments does not necessarily prevent the deep RL agents from overfitting.

2.2.2 CONVENTIONAL REGULATIZATION ON RL

For deep supervised learning, regularizations has long been a major issue as generalization from training to test set is the major concern. Common regularization practices like L2 regularization, dropout, batch-normalization from supervised learning has also been tested in the reinforcement learning scenario. Cobbe et al. (2018) and Farebrother et al. (2018) show that dropout and L2 regularization improve generalization to test environments in Coinrun(Cobbe et al., 2018) and Atari 2600(Bellemare et al., 2013) respectively.

2.2.3 DATA AUGMENTATION ON RL

For supervised learning related to images or computer vision, data augmentation is a popular way to enhance performance by injecting useful priors on the training data. In recent years, we see numerous data augmentation methods for RL that improve generalization or data efficiency. Lee et al. (2019) introduce a randomized convolution network that randomly perturbs input features. Srinivas et al. (2020) utilize contrastive learning and show considerable performance gains in terms of sample-efficiency. Kostrikov et al. (2020) uses augmented data and weighted Q-functions to achieve state-of-the-art data-efficiency on the DMControl(Tassa et al., 2018). Laskin et al. (2020) investigate ten different data augmentation on RL and point out that random crop is the most effect on DMControl(Tassa et al., 2018)

3 BACKGROUND

CSSC is a general framework of cross-state representation regularization for RL. In principle, one can apply CSSC to other variation of DQN or policy-based models for discrete action-space environments. In this work we pick Rainbow(Hessel et al., 2017) as our base model to show that CSSC is compatible with the original Nature DQN(Mnih et al., 2015) and its multiple improvements. In the following subsections we will review Rainbow DQN and introduce the concept of implicit feedback in Bayesian Personalized Ranking (BPR) proposed by Rendle et al. (2012).

3.1 RAINBOW

In combination with a convolutional neural network as visualized input encoder, Deep Q network(Mnih et al., 2015) demonstartes that it is possible to use neural network as a function approximator that maps raw pixels to Q values. Since then, multiple improvements such as Double Q Learning(van Hasselt et al., 2015), Dueling Network Architecture(Wang et al., 2015), Prioritized Experience Replay(Schaul et al., 2015) and Noisy Networks(Fortunato et al., 2017) have been proposed. In addition, Bellemare et al. (2017) proposed the method of predicting a distribution over possible value support through the C51 algorithm. By combining all the above techniques into a single off-policy algorithm, Rainbow DQN showcases the state-of-the-art sample-efficiency on Atari Benchmarks. The resulting loss function for Rainbow DQN is as follows:

$$\mathcal{L}_{Rainbow} = D_{KL}(\Phi_z d_t^{(n)} || d_t) \tag{1}$$

where Φ_z is the projection onto fixed support z. $d_t^{(n)}$ and d_t are the n-step target return distribution and the model-predicted return distribution respectively.

3.2 BPR-OPT

As opposed to explicit feedback like user ratings, implicit feedback in recommendation systems focuses on interactions like click or view between users and items. In implicit feedback system only positive observations are available, and non-observed user-item pairs -e.g. a user has not view an

item yet - are considered as negative observations. Instead of rating prediction, BPR-opt(Rendle et al., 2012) is designed for direct ranking optimization based on the implicit feedback between users and items. BPR extends the user preference from observed interaction pairs to non-observed data by ranking the preference of positive observation and negatived observation across the training data. The formulation of the maximum posterior estimator for the personalized ranking optimization (BPR-opt) can be written as follows:

$$\ln p(\Theta| >_u) = \sum_{(u,i,j)\in D_s} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2 \tag{2}$$

where Θ represents the parameter vector of the base model. D_s represent the batch samples from training data. $p(\Theta| >_u)$ is the posterior probability conditioned on latent preference structure $>_u$ for user u, λ_Θ is the model specific regularization parameter, σ is the sigmoid function, and $\hat{x}_{uij}(\Theta)$ is the real-valued function model Θ which captures the preference relationship between user u, item i and item j. Rendle et al. (2012) use $\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj}$ to indicate that the user u prefers item i over item j.

4 METHOD

The main idea of CSSC is to extend the similarity ranking across various state representation based on our hypothesis: representations motivating identical behavior should share more similarity than those motivating different behavior. We name this method as "self-constraint" to emphasize that the state pairing is decided by the agent's behavior rather than pre-defined fixed labels. In the following subsection below, we introduce the following concepts to provide further explanation and insight: (i) definition of the behavior of an agent at a given state (ii) implicit feedback between state and behavior (iii) cross-state self-constraint as an auxiliary loss for representation regularization

4.1 BEHAVIOR DEFINITION

We describe a typical Markov Decision Process(MDP) as (X, A, R, P, γ) with state space as X, action space as A, reward function as R, state transition function as P and discount factor as γ. The agent would take an action a_i at state x_i and then being transited to x_{i+1} by the environment with a given step reward r_i. Here we define the behavior set $B_i^n = \{(a_i, a_{i+1}, ..., a_{i+n-1})|a \in A\}$ as a set of action series of length n taken by the agent since state x_i. For $b_i^n \in B_i^n$ and $b_j^n \in B_j^n$, $b_i^n = b_j^n$ only if $a_{i+p} = a_{j+p}$ for $0 \leq p \leq n-1$. With this definition in mind, we can infer that if the agent conduct the behavior $b_i^2 = (left, fire)$ at state x_i, it would move left in state x_i and fire in the next state x_{i+1}. In the following paragraph we coin names like unigram, bigram and trigram for behaviors of length one, two and three respectively.

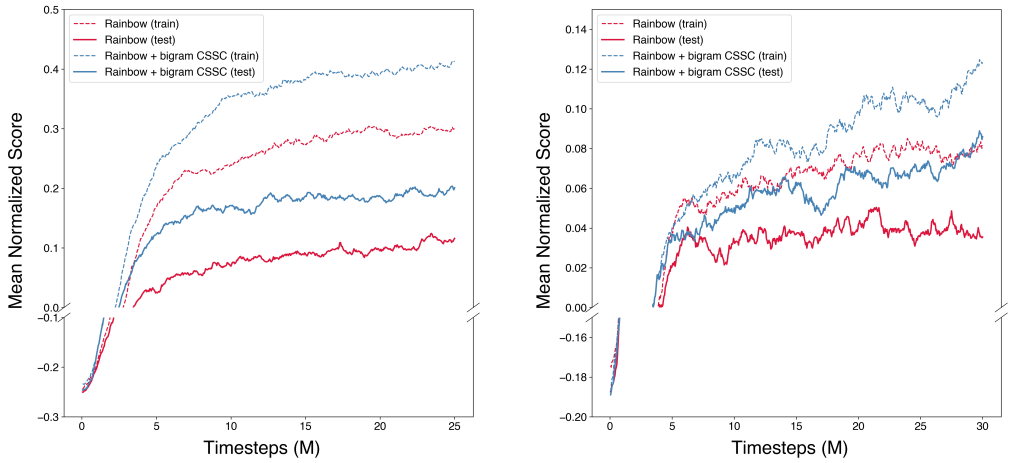
4.2 IMPLICIT FEEDBACK IN REINFORCEMENT LEARNING

In the traditional deep Q-learning scenario, each state representation is highly correlated to neighboring states by the Bellman equation during the training process. However, observational overfitting could still happen as long as the representation of neighboring states can fit to specific sub-optimal version of Q-function indefinitely as mentioned in Dabney et al. (2020). Besides the base RL algorithm, we can view the interaction between state and behavior as an implicit feedback system by regarding states sharing same action as positive observation. The general feature among these positive observation would be reinforced if we compare their similarity with negative observation -e.g. states sharing different action. Through this "ranking" procedure the pair-wise relationship would be extended across non-neighboring states and therefore preventing observational overfitting to specific Q-function.

4.3 IMPLEMENTATION OF CROSS-STATE SELF-CONSTRAINT IN COMBINATION WITH RAINBOW

For each sample of state triples$(x_p, x_q, x_r) \in X$ with $(b_p^n, b_q^n, b_r^n) \in B^n$ and $b_p^n = b_q^n \neq b_r^n$, we decompose the estimator $\hat{x}_{pqr}$ and define it as:

$$\hat{x}_{pqr} := \hat{x}_{pq} - \hat{x}_{pr} = e_\theta(x_p) \cdot e_\theta(x_q) - e_\theta(x_p) \cdot e_\theta(x_j) \tag{3}$$



(a) Easy mode

(b) Hard mode

Figure 2: Mean normalized score curve on OpenAI ProcGen environment. We normalized the episodic return of each game by the constants from Cobbe et al. (2020) and report the mean score. Every curve is smoothed with an exponential moving average of 0.95 to improve readability.

where e_θ is the encoder function with weight θ that maps the pixel input to 1-dimension feature vector as the state representation. We directly perform inner-product on $e_\theta(x_p)$ and $e_\theta(x_q)$ to calculate the representation similarity $\hat{x}_{pq}$ between x_p and x_q. Please note that the encoder function is the same part of the original base model used for feature extraction on visualized input(Figure 1a). To sample these state triplets without modifying the base Rainbow algorithm, we collect these state triples in the same batch of transitions used to calculate distributional Bellman Loss $\mathcal{L}_{Rainbow-loss}$ defined in (1). We pair up the state triples for each transition in the sample and calculate the CSSC loss as follows:

$$\mathcal{L}_{CSSC} = -\frac{1}{n} \sum_{(p,q,r)\in D_s} \ln \sigma(\hat{x}_{pqr}) \tag{4}$$

where D_s represents the sample batch of size n from the replay buffer and σ is the sigmoid function. Finally, we train the base Rainbow algorithm in combination with the auxiliary CSSC loss as follows:

$$\mathcal{L}_{total} = \mathcal{L}_{Rainbow} + \beta_{cssc} \cdot \mathcal{L}_{CSSC} \tag{5}$$

where β_{cssc} is the hyperparameter to control the contribution of CSSC during training. We find that setting β_{cssc} to 0.01 works quite well in most cases, so we stick with this setting for all the experiment conducted in the next section.

5 Experiment

Our primary goal for CSSC is to enhance the generalization capability of RL algorithm to unseen levels that share similar mechanism. Fortunately, OpenAI ProcGen(Cobbe et al., 2020) presents a collection of game-like environments where the training and testing environments differs in visual appearance and structure. Therefore, we evaluate CSSC in three different ways:(i) generalization improvement on 12 OpenAI ProcGen games in easy mode, (ii) generalization improvement on 8 OpenAI Procgen games in hard mode and (iii) visualization on the representation space learned with CSSC.

5.1 Environment and Model Setting

For the following testing in ProGen, we use the IMPALA CNN architecture recommended by Cobbe et al. (2020) for the base Rainbow model, and all the experiments in this paper are conducted in

Table 1: Mean normalized score for OpenAI ProcGen Easy Mode

Mode	Rainbow	Rainbow+unigram CSSC	Rainbow+bigram CSSC
easy(train)@25M (12 Games)	0.2862	0.3832	0.4030
easy(test)@25M (12 Games)	0.1018	0.1670	0.1892
Improve. for train(%)	0.0	33.89	40.81
Improve. for test(%)	0.0	64.05	85.85

Table 2: Mean normalized scores of bigram-CSSC

Mode	Rainbow	Rainbow + bigram CSSC	Improvement(%)
easy(train)@25M (12 Games)	0.2862	0.4030	40.81
easy(test)@25M (12 Games)	0.1018	0.1892	85.85
hard(train)@30M (8 Games)	0.1085	0.1699	56.63
hard(test)@30M (8 Games)	0.0512	0.1199	133.94

single-agent single-environment setting. For all hyperparameter used in ProcGen benchmark testing, please refer to Table 3 in the Appendix. We use "unigram CSSC" or "bigram CSSC" to indicate the behavior length used to pair up the state triples for CSSC.

5.2 Generalization on Procgen in Easy mode

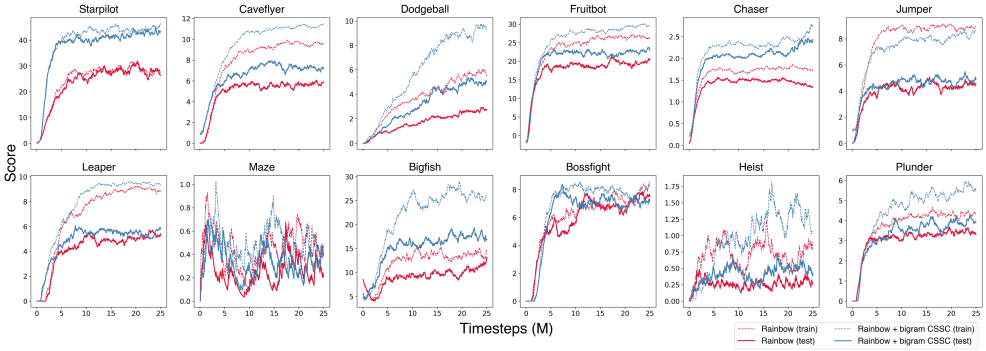


Figure 3: Learning Curve of Rainbow with bigram CSSC in 12 ProcGen Games. We report the raw episodic return for both training and testing. All final scores are listed in Table 4. Every curve is smoothed with an exponential moving average of 0.95 to improve readability.

As Cobbe et al. (2020) have suggested, we conduct the training of 25 million timesteps on 200 training levels and evaluate the generalization improvement on the full distribution of testing levels across 12 ProcGen games. We test both the unigram and bigram CSSC in comparison with vanilla Rainbow and see significant improvement on testing performance. To evaluate the mean normalized score for bigram and unigram CSSC across 12 games, we follow the normalization method used in Cobbe et al. (2020) and show the result in Figure 2a and Figure 9a respectively. We also display the learning curve of bigram-CSSC in Figure 3 and unigram-CSSC in Figure 10. In the following we summarize the main findings below:

- In supervised learning, posing regularization or constraint to the model would usually improve its testing performance on unseen data at the expense of hurting training performance. However, we see that the proposed self-constraint improves both training and testing performance across most of the 12 game tested.
- For games like Starpilot, Chaser and Bigfish, we see that the testing performance is even better than the training performance of vanilla Rainbow.
- For the mean normalized score in Table 1, Bigram-CSSC and Unigram-CSSC bring 85% and 64% improvement respectively on the testing performance in comparison with base Rainbwo model.

5.3 GENERALIZATION ON PROCGEN IN HARD MODE

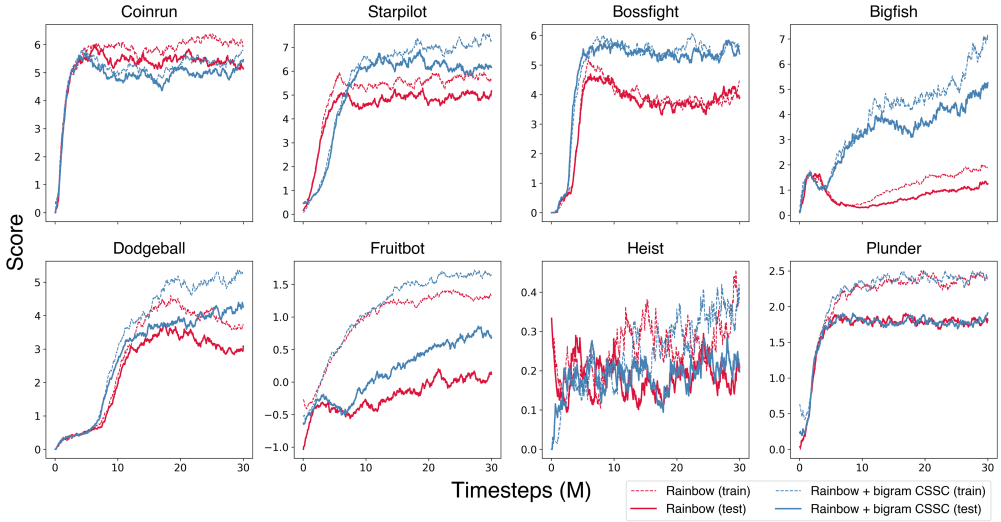


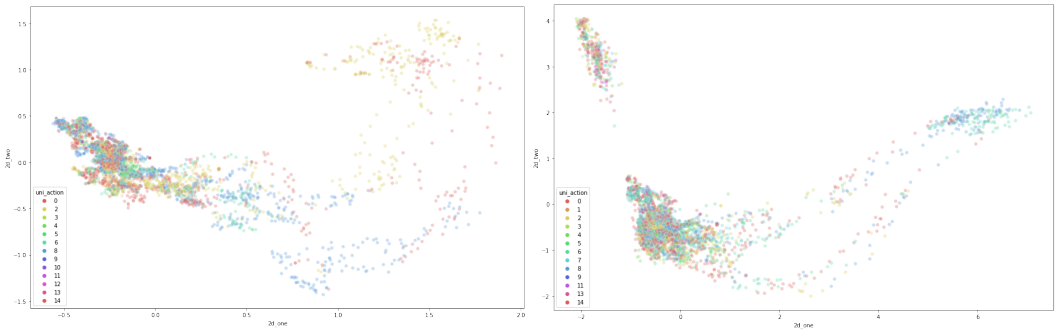
Figure 4: Learning Curve of Rainbow with bigram CSSC in 8 ProcGen Games. We report the mean raw episodic return for training and testing. Every mean return is shown across 3 seeds and all final scores are listed in Table 5. Every curve is smoothed with an exponential moving average of 0.95 to improve readability.

To further explore the generalization capability in more complicated and challenging tasks, we examine the generalization improvement brought by CSSC on 8 ProcGen games.We use 500 training levels for 30 million timesteps of training and evaluate generalization on the full distribution of testing levels. We test the bigram CSSC in comparison with the normal version of Rainbow across three different seeds and see significant improvement on testing performance. The learning curve of bigram CSSC is summarized in Figure 4. To evaluate the mean normalized score for bigram CSSC across 8 games in hard mode, we also follow the normalization method used in Cobbe et al. (2020) and show the result in Figure 2b. In the following we summarize the main findings below:

- As shown in Figure4, CSSC substantially improves both training and testing performance in Bigfish, Dodgeball, Starpilot, Fruitbot and Bossfight.
- In particular, we see nearly 4x performance jump in Bigfish at 30M timestep in comparison with the base Rainbow model. The mean episodic return at 30M timestep of bigram CSSC is even higher than that of PPO at 200M timestep as shown in Figure 4 of Cobbe et al. (2020).
- In the case of Heist (a puzzle-solving task in maze-like layour), coinrun(a 2-D scroll platform game) and Plunder(a challenging shooting game), the gain from CSSC is not as obvious because all these games require careful manipulation and planning. Add constraint that modifies state representation directly would be risky or even detrimental for the learning process of the base RL algorithm.
- For the mean normalized score in Table 2, Bigram-CSSC brings about 130% improvement on the testing performance.

5.4 VISUALIZATION OF REPRESENTATION EMBEDDING

To give a more tangible explanation about the effectiveness of CSSC, we plot the representation distribution in the embedding feature space across 4096 states from the replay buffer. We first perform dimension reduction on these state embeddings using principal decomposition analysis and label each embedding point with the index of corresponding action. Here we show the embedding space of both Rainbow and bigram-CSSC models trained to play the Bigfish game. In figure 5 we can see that states of the same action are more clustered in bigram-CSSC than that of the vanilla Rainbow



(a) Bigfish: bigram CSSC

(b) Bigfish: Rainbow

Figure 5: Representation embedding projected to 2D space by PCA. We use the same 4096 state frames to extract representation from both models and display the distribution across all state representation.

model. We believe this is the possible reason behind the significant improvement on generalization as shown in Figure 4. For representation visualization in other ProcGen games, please refer to A.3.

6 Conclusion

In this paper we propose a novel regularization on state representation learning based on the connection between agent behavior and the visual input. Our hypothesis is derived from the observation that the behavior of a rational agent would have certain relationship with general cross-state features or patterns. We see significant improvement on generalization brought by the proposed cross-state self-constraint(CSSC) on most of the games in OpenAI ProcGen benchmark. The connection between behavior and conceived visual input can be considered as some kind of "motivation" that acts as a decisive factor behind the learned policy. It worth further study to better understand the derivation and transformation of agent's "motivation" during the Markov Decision Process, and we believe the concept proposed in this work would facilitate more research in this direction.

References

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, Jun 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL `http://dx.doi.org/10.1613/jair.3912`.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning, 2017.

Marc G. Bellemare, Will Dabney, Robert Dadashi, Adrien Ali Taiga, Pablo Samuel Castro, Nicolas Le Roux, Dale Schuurmans, Tor Lattimore, and Clare Lyle. A geometric perspective on optimal representations for reinforcement learning, 2019.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning, 2018.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2020.

Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G. Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning, 2020.

Jesse Farebrother, Marlos C. Machado, and Michael Bowling. Generalization and regularization in dqn, 2018.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration, 2017.

Matthew Hausknecht and Peter Stone. The impact of determinism on learning atari 2600 games. In *AAAI Workshop on Learning for General Competency in Video Games*, Austin, Texas, USA, January 2015. URL http://www.cs.utexas.edu/users/ai-lab?hausknecht:aaai15.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.

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

Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pp. 1001–1007, Cambridge, MA, USA, 1999. MIT Press.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels, 2020.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data, 2020.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning, 2019.

L. Li, T. Walsh, and M. Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents, 2017.

Andrew Kachites Mccallum and Dana Ballard. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, 1996. AAI9618237.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529—533, February 2015. ISSN 0028-0836. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback, 2012.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354—359, October 2017. ISSN 0028-0836. doi: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning, 2019.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015.

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning, 2018.

A APPENDIX

A.1 HYPERPARAMTERS AND MODEL SETTING FOR OPENAI PROCGEN BENCHMARK

Table 3: Hyperparameters of Rainbow for OpenAI ProcGen

Env.Distribution Mode	**Easy**	**Hard**
Use stacked Frames	False	False
Image size(Channel, Height, Width)	(3, 64, 64)	(3, 64, 64)
Training levels	200	500
Training start Level	123	123
Testing levels	0	0
Testing start level	0	0
Batch size	128	128
CSSC type	unigram, bigram	bigram
CSSC constant β_{CSSC}	0.01	0.01
Multi step return	3	3
Learning rate	0.00025	0.00025
Replay buffer size	500000	500000
Minimum replay size for sampling	20000	20000
Replay frequency	16	16
Target network update period	8000	8000
Total timesteps	25M	30M

A.2 SCOREBOARDS ON OPENAI PROCGEN

Table 4: Easy mode scores evaluated after 25M timesteps of training.

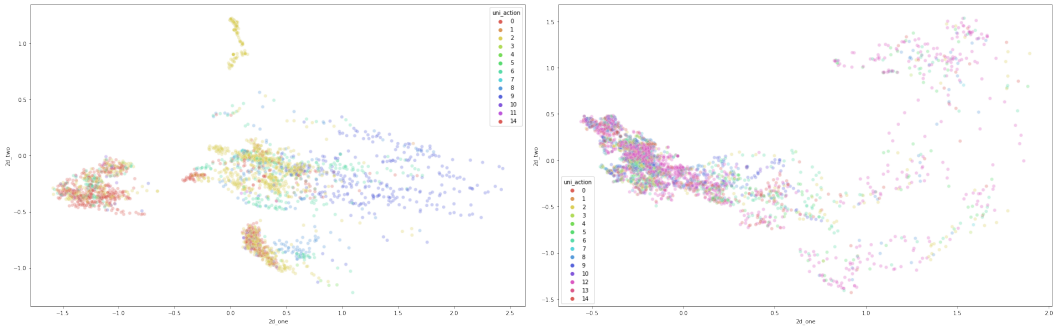
	Train			**Test**		
Game	Rainbow	Unigram	Bigram	Rainbow	Unigram	Bigram
Starpilot	27.79	43.75	**46.13**	26.37	39.72	**43.40**
Caveflyer	9.70	10.63	**11.43**	5.83	7.22	**7.23**
Dodgeball	5.46	9.08	**9.48**	2.74	4.25	**5.04**
Fruitbot	26.25	29.27	**29.68**	20.75	**23.81**	22.91
Chaser	1.71	2.33	**2.74**	1.34	2.07	**2.43**
Jumper	**8.96**	8.73	8.62	4.54	**4.99**	4.97
Leaper	8.81	**9.58**	9.39	5.42	5.83	**5.88**
Maze	0.41	**0.71**	0.45	0.21	**0.50**	0.38
Bigfish	14.80	**26.60**	26.04	13.16	14.87	**16.91**
Bossfight	8.26	8.16	**8.57**	**7.52**	6.90	7.32
Heist	0.77	0.77	**1.06**	0.26	0.34	**0.42**
Plunder	4.34	4.51	**5.58**	3.32	2.75	**3.95**

A.3 VISUALIZATION OF REPRESENTATION EMBEDDING ON OPENAI PROCGEN GAMES

A.4 GENERALIZATION ON PROCGEN IN EASY MODE FOR UNIGRAM-CSSC

Table 5: Hard mode scores evaluated after 30M timesteps of training.

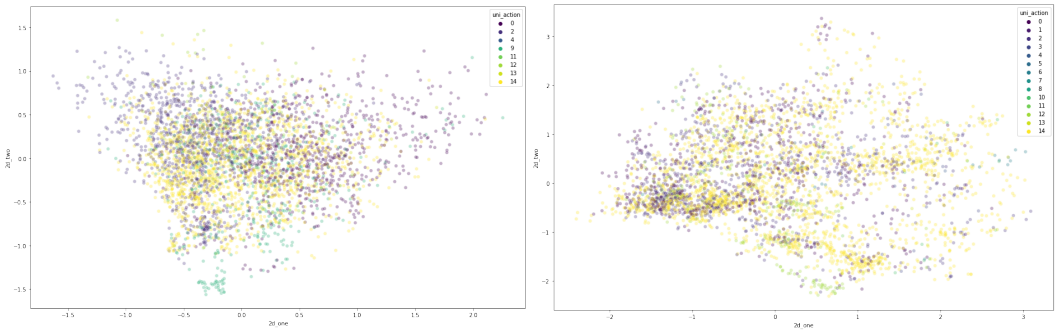
	Train		**Test**	
Game	Rainbow	Bigram	Rainbow	Bigram
Coinrun	**5.99**	5.78	5.13	**5.46**
Starpilot	5.63	**7.30**	5.17	**6.16**
Bossfight	4.42	**5.64**	3.89	**5.38**
Bigfish	1.88	**7.22**	1.24	**5.26**
Dodgeball	3.74	**5.23**	3.09	**4.26**
Fruitbot	1.33	**1.65**	0.12	**0.69**
Heist	0.38	**0.39**	**0.21**	0.20
Plunder	**2.43**	2.38	1.81	**1.91**



(a) Dodgeball: bigram CSSC

(b) Dodgeball: Rainbow

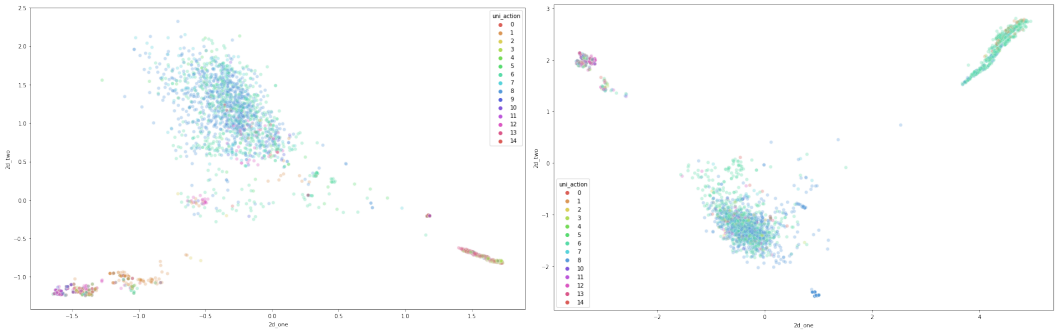
Figure 6: Dodgeball: representation embedding projected to 2D space by PCA



(a) Starpilot: bigram CSSC

(b) Starpilot: Rainbow

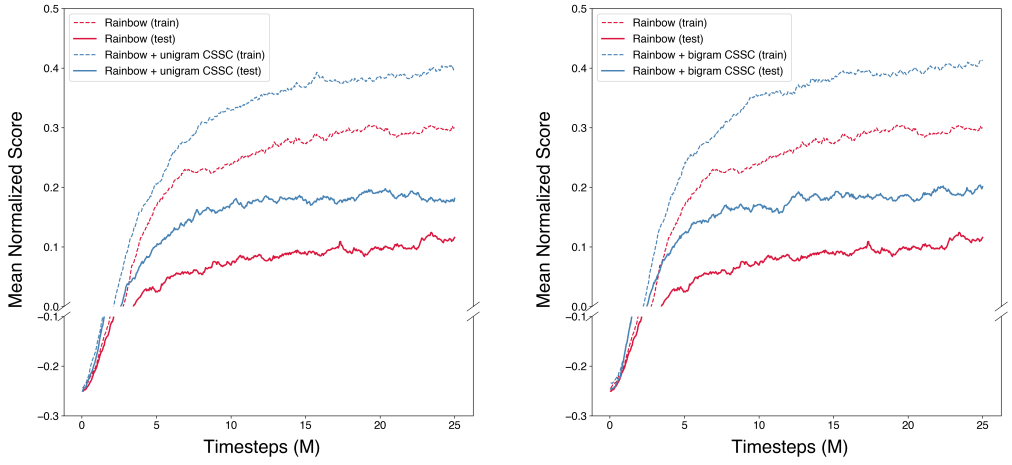
Figure 7: Starpilot: representation embedding projected to 2D space by PCA



(a) Coinrun: bigram CSSC

(b) Coinrun: Rainbow

Figure 8: Coinrun: representation embedding projected to 2D space by PCA



(a) unigram-CSSC

(b) bigram-CSSC

Figure 9: Mean normalized score on OpenAI ProcGen environment in easy mode. Every curve is smoothed with an exponential moving average of 0.95 to improve readability.

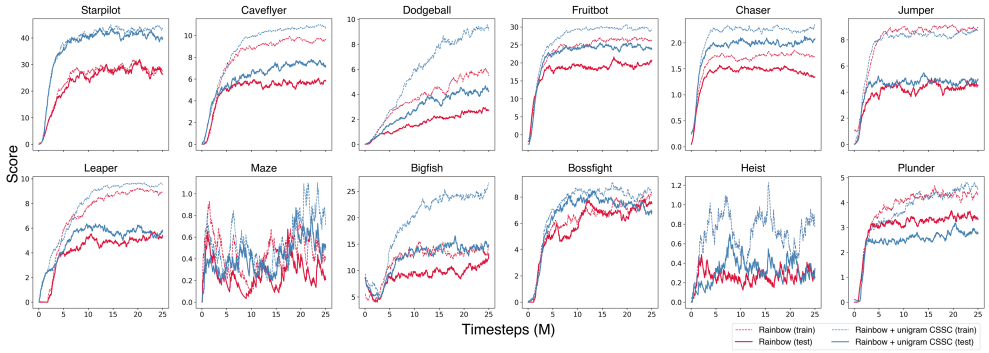


Figure 10: Learning Curve of Rainbow with unigram CSSC in 12 ProcGen Games. We report the raw episodic return for both training and testing. Every curve is smoothed with an exponential moving average of 0.95 to improve readability.