maRLios Final Report

Josh Mazen mazen@usc.edu
Carlos Osorio osoriova@usc.edu
Bozhena Pokorny pokorny@usc.edu
Cameron Witz witz@usc.edu

Abstract

Traditional Deep Q-Learning agents often suffer from long training times and large state spaces. To combat this issue, we introduce maRLios, an agent that generalizes to different action sets to play Super Mario Bros. Using both CNN and vanilla RNN layers, we train this agent by generating a subset of actions alongside the current game state to select the best expected reward. We find that RNN + positional encoding provides the best reward for this type of architecture and conclude with recommendations for future improvements.

1 Introduction

Recent growth in the development of video game AI is being pushed further by the rising availability of machine learning methods. However, because most games require combining sequences of complex patterns to achieve a goal, this task presents unique challenges. A strong agent should have domain-specific knowledge of the game's operations while still having the capability to adapt to some unforeseen scenarios. This is ultimately the goal of our proposed deep learning model, maRLios, shorthand for Missing Action Reinforcement Learning in Obstacle Systems, a Super Mario Bros. playing agent.

2 Background

Since their inception, AI and gaming have grown in synergy, both benefiting from each other's progress. Various AI methods, such as minimax and Q-learning have been used to create artificial agents capable of playing among human players. In turn, the expansion and popularity of video games have driven further development of different learning algorithms by providing new benchmark environments. Reinforcement learning is one such approach that has found its home in video games.

In typical reinforcement learning, agents are trained using all of the available actions they can take for a given task and environment. A detriment of this style of training agents, however, is the inability to effectively scale to different and actions and environments which were not explicitly introduced during training time. Indeed generalization is one of the core research areas looking to be expounded upon in the field of Reinforcement Learning.

Existing methods to create agents capable of generalization have mainly focused on learning specific tasks, as in the case of (Hessel et al., 2019) where multitask models are carefully trained to create agents capable of generalizing to multiple environments. While this is certainly useful, we believe that there is still much work to be done in the space of creating agents capable of generalizing to new actions, which has been under explored. Creating a benchmark agent for OpenAI's gym could open up the possibility to combine multitask learning agents with action generalization, enabling developers and experimenters to reuse reinforcement learning agents across games without limiting the mechanics available to the player or agent.

2.1 Generalization in RL

Reinforcement Learning is a unique machine learning paradigm that requires sequential decision making to solve complex tasks. It can require millions of episodes of repetition to learn the impact of such a series of decisions, and to make progress towards achieving the goal. The ability to generalize to new situations is desired for employing RL in real-life applications, as the possible decisions an agent is able to make may change over time. Early work explored improving generalization in RL by applying techniques commonly used in Supervised Learning: including regularization (Farebrother et al., 2018), data augmentation (Lee et al., 2019), and alternative neural network architecture (Cobbe et al., 2018). However, such attempts made modest improvements when an agent was presented with an unseen task and still struggled with overfitting when presented with small or medium training sets. Agarwal et al. have hypothesized that these techniques were insufficient because they ignored the sequential nature of the agent's decision-making space, or otherwise failed to capture the semantics of an agent's interactions in a high-dimensional representation (Jain et al.,

2020)(Agarwal et al., 2021).

Recent progress has been made by attempting to embed greater contextual information into the environment-, state-, or action-space. A project by Agarwal et al. attempts to learn similarity between states, despite varied environments, based on shared sequences of actions. They achieve this by applying contrastive learning to learn embeddings of behaviorally similar states based on a state-similarity metric. Act2Vec (Tennenholtz and Mannor, 2019) borrows the notion of context in language from NLP to learn embeddings for actions representative of their likelihood to occur in a given environment (context). They successfully generate embeddings that capture similarity between actions, which can improve generalization by injecting prior knowledge into the problem-space before solving an RL task.

An approach taken by Chandak et al. extends this idea to a set of actions: they propose learning a representation of action-embeddings to create a space in which an agent can learn an internal policy. Their key innovation was that the *policy* can learn during RL training to improve upon the provided action representation. This increased the trained agent's ability to generalize when presented with unknown actions in training subsets. Further work by (Jain et al., 2020) and (Chandak et al., 2020) combined the above approaches into two steps to train an RL agent optimized for generalization across action sets. Both first acquire descriptive action representations, then they apply RL with a policy flexible to varying action sets.

2.2 Deep Q-learning

Double Deep Q-learning (DDQL) is a popular RL algorithm used to train game agents. It is an off-policy value-based model that uses two deep neural networks to estimate each state-action pair's expected cumulative reward, or Q-value. One network, known as the online network, is used to select the best action while the other, the target network, evaluates the selected action. Decoupling selection and evaluation this way helps reduce overestimation of Q-values and stabilizes the learning process.

While DDQN has shown success in projects in the video game space, it still suffers from challenges common to RL - including the extensive training time required to create competent game-playing agents and difficulties with generalization.

Transfer of knowledge between agents in different environments typically requires re-training agents for distinct tasks, making integration of this technology across multiple games impractical. One approach that addresses the challenge of agent generalization across similar, but distinct, action-spaces uses action embeddings to train agents to associate action representations with an optimal policy (Jain et al., 2020). This was shown to enhance an agent's ability to use unseen actions to achieve its objectives.

3 Problem Statement

In this project, we seek to create a benchmark for off-policy learning with action generalization in a complex gaming environment. Super Mario Bros. is a game that requires complex sequential problem-solving, suitable for this task. The agent is trained using a modified DDQL procedure, in which the agent will be trained to estimate an optimal Q-value for its set of available actions to achieve its goal: overcome obstacles, avoid enemies and death, and proceed towards the finish line as efficiently as possible.

The novelty introduced by this benchmark comes in both the complexity of the environment to which the agent must adapt, which sequentially introduces new challenges as the agent proceeds, as well as in the application of the DDQN algorithm to achieve this goal.

4 Approach

We approach generalization to new actions by following procedures similar to those outlined by Jain et al.. This required two steps to establish an RL-training procedure that would enable generalization to unseen actions. First, create an action representation that would allow our model to identify similarities across different actions. Second, adopt a flexible model architecture that could take any subset of the actions as batched input during training time, or subset of unseen actions during testing time.

While the approach taken by (Jain et al., 2020) used a policy-based training objective, we train a deep neural network to estimate Q-values. We began with a framework for DDQN and initial model architecture based on a blog post by Grebenisan (2020). We similarly used used a series of convolutional layers to process the image data representing the environment at each time step and output estimated Q-value distribution for the available actions.

However, since our model is not presented with the full action-set, we had to adjust the output to give a batched matrix of Q-values for the subset of actions available at each time step. Additionally, we adjust the model architecture to consider the choice of available actions given the context of its environment by feeding both a learned vector-representation for the environment (the output of the CNN), and a vector representing the available action-set into the final Neural Network to learn the Q-values.

5 Implementation Details

5.1 Environment Details

We are using Open AI's gym environment, specifically gym-super-mario-bros environment (Kauten, 2018). This environment has been widely used to train RL agents, which allows our team to focus on the model and training as opposed to configuring the environment. Mario is divided into worlds and stages, and the agent we are training sees input from the complete game level in World 1, Stage 1.

5.2 Rewards and Actions

Rewards and actions are a crucial part of defining any RL agent. The rewards used to train maRLios are defined below, per (Kauten, 2018):

$$r = \delta x - \delta c - d \tag{1}$$

Here, δx and δc represent the changes in x-position and game clock between time steps, respectively, whereas $d = 15$ if the agent dies and 0 otherwise. This reward function incentives our agent to complete the level (by moving to the right, maximizing δx) without wasting time (minimizing δc).

Additionally, the action space of the gym environment mirrors the original Nintendo Entertainment System (NES) action space of 256 unique button press combinations. Popular subsets of these actions exist in the gym environment as "wrappers", labelled as *Right Only* and *Simple Movement*, and *Complex Movement* (Kauten, 2018), which drastically reduce the action space to 5, 7, and 12 unique actions, respectively.

5.3 Baseline Model

The baseline model we compare maRLios against uses a double deep-Q learning scheme. The model takes a frame of pixels as its state and processes them using three convolutional layers, as seen in Figure 1a. This is then fed through several linear layers and outputs a single index to the action that is expected to maximize reward.

5.4 Action Encodings

We define an action as a combination of simultaneous button presses during a 4-frame duration. As there are only 8 buttons available as input in our environment, we can use a multi-hot vector of size 8 to encode each possible combination of button presses comprising 256 total actions.

Initially our team sought to use a simple representation of an action as a simple multi-hot encoded vector of size 5 for the 5 possible valid button presses that are available on the NES ('A', 'B', 'down', 'left' and 'right'). The agent may also choose to do nothing, which is represented by a vector of all 0's ('NOOP'). For the purposes of training a generalizable agent however, the resulting set of possible actions we determined to be too small, so instead of taking one action at a time, our agent takes *two*. The result is an action encoding of size 10 where each group of 5 values represents the buttons being pressed during a single action.

During run time, our agent selects two actions from it's action set, and we play each of the two actions it chose over a 2 frame period.

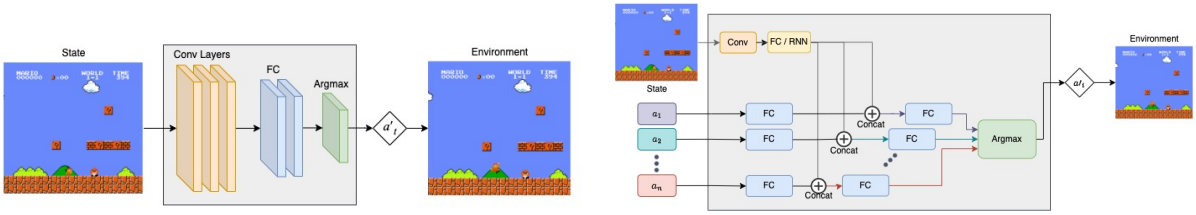
5.5 Positional Bias

We chose to experiment with including a positional bias to our *two action* vector representations, by multiplying the second 5 values for the second action by 2. Our hope is that the model will learn that the larger values are actions which play after the smaller values.

5.6 Sufficient Action Sets

During training and testing, in order to ensure that our model always had enough actions to complete or at the very least progress in the game, we defined what we refer to as sufficient action sets. These sufficient action sets were split into 'jump' sets, and 'right' sets, where the sets include only action representations that include 'jump' and 'right' in both of the two actions represented by our encoding. One note about our sufficient right sets is that we made sure not to include 'left' in any of the available actions for those sets.

For this, we split our entire action space into training, validation, and a final testing holdout set. Additionally we created sufficient sets for each of these splits, and ensured mutual exclusivity.



(a) Baseline DQN

(b) DQN with Action Generalization

Figure 1: (a) Baseline agent trained using Deep Q-learning. The model estimates the Q-value of each action from the current game state using convolutional and linear layers. (b) Agent with action generalization. The model uses separate action representations concatenated to the encoded state to estimate each action's Q-value. FC/RNN denotes that some agents were trained with an fully connected layer here, and others substituted this with an RNN layer

5.7 Model Architecture

To achieve generalization to unseen actions, we consider the method of accepting and evaluating actions in our model's design. We experimented with several different architectures over the duration of this project, however the common thread amongst them all was the concatenation of action representations to our latent state representation. Our final architecture consists of 3 convolutional layers to process the current game state into a latent representation as shown in Figure 1b. This latent representation will then be passed through a single layer RNN. The output of the RNN is then repeated n times for each of the n *actions* we plan to evaluate. Our action encodings are passed through two small fully connected layers to create a latent action represenation. We then concatenate each latent action representation to our latent state encoding, before finally feeding this batch of action-state vectors through several linear layers. The final output is a n dimensional tensor representing the Q *value* estimate for each action, from which our model will select the action corresponding to the highest value.

We experimented with other architectures along the way, the most notable differences being that these early models did not contain any RNN layers, as well as in some cases we did not use a latent action representation, but rather simply concatenated the raw 10 dimensional action encoding to our state representation directly. We abandoned these earlier architectures in favor of the RNN based approach which was achieving better results and better generalization.

5.8 Training procedure

To train our model, we used a combination of epsilon-greedy action selection and random action space sampling. At the start of each training episode, our agent samples random actions from our training set. In addition to this, we always sample 3 more actions. One from the sufficient jump training set, another from the sufficient right training set, and lastly, we include 'NOOP'. The agent progresses through the entire level until it reaches a terminal state, at which point we save the relevant statistical information, and re-sample its available actions in the aforementioned manner.

Every 10 episodes, we also evaluate our agent on the validation set and compare it's progress to see if it is learning to generalize to unseen actions. The agent is never trained on the data from this validation set however. During this process, the agent is not exposed to our test set, which we reserve for our final comparison.

5.9 Experiments

Our experimentation followed three phases to establish our problem formulation and validate a baseline model, then to optimize our training procedure for generalization.

5.9.1 Establish valid action representation

We first needed to build an action representation that could be compatible with the gym environment as well as represent a complex action-space. For comparison, we trained a benchmark DDQN RL-agent on action-sets native to the gym environment to serve as our baseline. The baseline model did not consider the action as input, but estimates a distribution of Q-values over the static action space (Fig. 1b). The **Baseline DQN** model, trained on the static 7-action set *simple movement* served as

Name	**Avg. Total Rewards**	**Pipe 2%**	**Pipe 3%**	**Pipe 4%**	**Completion %**
Baseline	3080	100	100	100	100
maRLios	223.6 ± 214.7	19	7	1	0
maRLios + RNN	716.9 ± 197.5	97	80	6	0

Figure 2: Summary of each model run using test set for 100 episodes. Generalized models are tested on an unseen holdout set, while the baseline model was tested on the same simple actions used for training. Pipe X columns denote percentage of time an agent made it past pipe X (we start with pipe 2 as pipe 1 is too easy to use as a benchmark).

an example of successful training. We tested different formulations of our action encoding to ensure they had a comparable abstraction in the gym environment.

5.9.2 Demonstrate feasibility

Here, we established a process for sub-sampling the actions such that the agent must learn their utility based on their embedding. To determine an appropriate sub-sampling process, we experimented with: sub-sampling different sizes of randomly-chosen action subsets, sub-sampling a new set every action, sub-sampling every episode, and including sufficient action sets (Section 5.6). In this phase we build **maRLios** – our first action-generalizable model out of a CNN and a series of feed-forward networks to generate latent action-embedding space and output estimated Q-values.

5.9.3 Train for action generalization

Finally, we established a training procedure to optimize for generalization and avoid over-fitting to the training set. We split our actions into approximately 70:15:15 (training:validation:testing) each had independent sufficient action as described in Section 5.8 sets for sampling. Hyper-parameter tuning of learning rate, exploration rate, and neural network size was chosen by tracking the performance on this unseen validation set. In this phase, we introduced alternative architectures that include RNN components, to help train for the sequential nature of the game's tasks. To further account for sequentiality, we experimented with including positional bias sampling for our vector representations, as described in the above section 5.5. Our final, best-performing action-generalizing model from this phase we refer to as **maRLios + RNN**.

5.10 Evaluation Metrics

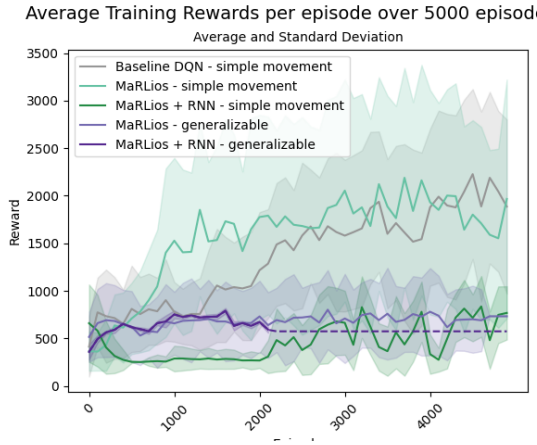
We compare the baseline, maRLios, and maRLios + RNN with a number of useful metrics. The total reward for each run, calculated in Section 5.2, is our primary success metric as it allows us to approximate the horizontal position of the agent alongside the time remaining. X-position and time elapsed for each particular run are also tracked separately in the training process to determine if the rewards are more heavily weighted by one metric versus the other. Tracking each separately also allows us to understand how far the model reaches in the level before either dying or running out of time.

To track the progress of each model over time, we compute both average total reward and standard deviation of reward. Within the training loop, we compute each metric after each episode to achieve fine-grained precision for the total training process. Every 10 episodes within the training process, we track these statistics using the validation set. For our test set, we run each model for 100 episodes using a different set of actions with potential for overlap for simple models and completely new actions for general models. These test sets are evaluated using all of the above metrics. Additionally, we track a number of benchmarks within the level to measure potential obstacles that the model has overcome. These benchmarks include moving past 3 tall pipes in the beginning of the level as well as if the model completed the level for a given run.

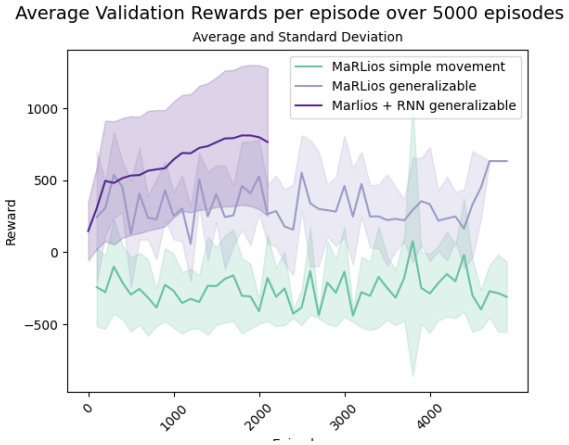
6 Results

The results of the models yielded mixed results, especially compared to the baseline. Though the baseline had been trained for a significantly longer period of time, when removing all randomness and given the *Simple Movements* action set, it was able to complete the entire level with 100% success rate and an average reward of 3080. While the generalized maRLios models did not complete the level at all during the testing phase, we believe that, given further modifications to the architecture, the model would compare favorably with the baseline.

The generalizing MaRLios without an RNN had conflicting results. While during the first stages



(a) Average and standard deviations of training rewards for baseline and maRLios variations, with predicted future performance of *MaRLios + RNN - generalizable* projected.



(b) Average and standard deviations of validation rewards for baseline and maRLios variations. The *MaRLios + RNN - generalizable* validation data is over 2000 episodes.

of training it seemed to be learning and its validation rewards had a positive slope (Fig. 3b), the model soon began overfitting to the training data, and the average validation rewards started decreasing. This decay in performance was evident during the testing phase, where after 100 episodes it obtained and average total reward of 223.6 ± 214.7 (Fig. 2). This indicates it frequently died to the first enemy and often failed to pass the beginning of the stage. This is likely due to some imbalance in the architecture, where the action set or the convolutional net overly influenced the decision making of the agent. The RNN model, however, yielded far more encouraging results. With an average reward of 716.9 ± 197.5, the model almost always passed the second pipe and usually the third. Given the fact that this model is tested on a set of data it had never seen and is trained for significantly less time than the baseline, we can surmise that there is improvement easily available.

The positional bias made a significant improvement to the model. In two comparable models after 2,000 episodes of training, the average validation reward is 516 when including positional bias and 344 without positional bias.

7 Discussion

We succeeded in establishing a new benchmark agent in this environment for testing action-generalization, and showed plausibility for our goal of achieving performance on unseen actions comparable with the baseline.

However, this project did face some challenges in its scope. Due to the experimental nature of generalization in reinforcement learning, and the challenge of training an RL agent to complete a full level of game-play, we set some simpler checkpoints to evaluate success. These checkpoints evaluate the agent's ability to progress beyond tall pipes, which we observed to be major obstacles to overcome, even in the baseline model. Additionally, our final best-performing model architecture and hyperparameters were determined to be Marlios + RNN. Although we planned to train under optimal conditions for a full 5000 episodes to compare longer-term training performance, we experienced time and resource constraints as the model took over 48 hours to train 2000 episodes.

An additional design decision aimed at enhancing the agent's generalization and validation results involved subsampling the available actions at the onset of each run, as opposed to each step. Although the latter method yielded superior average rewards for the training set, the validation agent's performance was suboptimal, and generalization was insufficient. Furthermore, we discovered that, when operating these trained agents on the training set without subsampling at each step, the agent would frequently become trapped behind an obstacle. The agent persistently executed the same action, typically involving consecutive jump presses, which failed to register as a second jump.

This behavior manifested due to the agent's acclimation to the continual alteration of available actions at each time step. As a result, the agent converged to select the "statistically best action" from the current subsample, with minimal regard for the state itself. Conversely, when the agent subsampled its actions at the inception of each episode, it demonstrated improved learning of action uti-

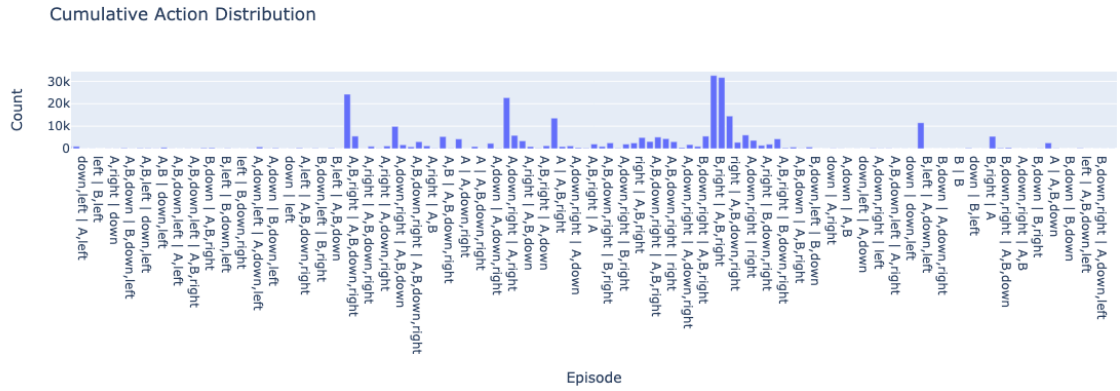


Figure 4: Distribution of most used actions by the maRLios-RNN model during the training period.

lization within its state, thereby achieving better generalization.

7.1 Major Challenges

We faced some challenges common to the field of Reinforcement Learning, including the massive amount of time required to train an intelligent agent, the problem of sample efficiency, and balancing exploration versus exploitation to find convergence to a suitable Q-value without over-fitting action selection (Shao et al., 2019) (Cao, 2020). Considering the time-frame restrictions of a semester-long project, we were able to undergo significant iterations of training, but increasing the number of episodes may be necessary to further improve performance.

We were able overcome the limited number of action (5 buttons) to define a tractable problem-statement by including the sufficient action-sampling and incorporating some redundancy in the dataset via the two-action combination. This way, we ensured the new test actions were within the distribution of the trained actions.

7.1.1 Action-space Representation

A primary challenge encountered in the training of maRLios pertained to defining the action-space representation and establishing suitable training, testing, and validation sets. This proved to be a fundamental aspect in achieving generalization in our model. We had to carefully consider the balance between the size and variability of the action set and the complexity of the task assigned to the Mario agent.

To facilitate the learning of action representations and their effective utilization in conjunction with the provided environment state, a sufficiently large training action set was required. The Mario Bros. game encompasses merely five types of button presses, necessitating the definition of actions as distinct combinations of button presses at a time. This resulted in an action space comprising 32 actions (including no button press). Nevertheless, certain button combinations (e.g., left and right) were contradictory, canceling each other out and reducing the original action set to a mere 24 actions.

To address this limitation, we opted to define each action as two consecutive "single" actions, as delineated in Section 5.6, thus augmenting our total action set size to 576. Although this approach expanded the action space, it simultaneously increased the complexity of the task the agent needed to learn. Consequently, the agent was now tasked with determining the subsequent two actions based on its current state, rather than a single action.

7.1.2 Sufficient Action Sets

Another critical facet of the action space involved ensuring that the maRLios agent had access to a sufficient set of actions during each run to accomplish its objectives. Accordingly, we established distinct sufficient action sets for training, validation, and testing, as described in Section 5.6.

During each run, we ensured that a minimum of one action from each of the sufficient action categories was available to the agent when subsampling the action sets. This approach, however, introduced a tradeoff: the sufficient action set exhibited a considerably higher probability of being sampled during each training run. Moreover, as these sets were more likely to prove useful in a diverse range of states, they were increasingly susceptible to repeated selection by the agent, resulting in a higher chance for overfitting and the inadequate learning

of actual action representations.

As evidenced by Fig. 4, the preponderance of actions selected by the agent at convergence belonged to these sufficient sets. We postulate that, while the majority of trained agents experienced a consistent and steep increase in average total rewards in validation for the initial 500 episodes, this pattern was followed by an evident degeneration of the agent on validation, as it adapted to identify a limited number of "good" actions.

7.2 Future Work

In the future, we propose including additional regularization techniques to encourage diversification of action selection and fight over-fitting. One approach is to include entropy maximization regularization, which has been shown to improve performance and stability in off-policy RL algorithms (Haarnoja et al., 2018). Further improvements to data efficiency and stability could be investigated by including Kullback-Leibler Divergence Regularization (Li et al., 2022).

Building upon the maRLios approach, future research could also investigate the use of a richer action representation space to train the agent. Several avenues for exploration include:

1. Extending the action definition to consist of 3 or 4 consecutive single actions: While this approach could increase the complexity of the task, it would also substantially expand the size of the training set and its respective sufficient sets. This, in turn, could help mitigate overfitting and improve the agent's performance.

2. Investigating more sophisticated auto-encoders for action representation: During our experiments, we observed that introducing a latent action space before incorporating it into the state representation led to increased validation rewards. We believe that this latent representation encoded useful information and relationships between button presses and consecutive actions, contributing to better generalization in the downstream task. Consequently, employing more advanced auto-encoders, such as Variational Autoencoders (VAEs) (Jain et al., 2020), could result in a more representative latent action space with additional encoded relationships.

By exploring these potential improvements, future work can seek to enhance the action representation space for maRLios, leading to better generalization and performance in reinforcement learning tasks.

8 Conclusion

In conclusion, the approach presented within this paper demonstrated promising results in policy learning reinforcement learning agents in video game play, by introducing action generalization in a complex and dynamic environment such as Mario Bros. Through our experiments, we observed that latent action representations could effectively capture relevant information about the agent's interactions with the environment, and that, in conjunction with state representations, led to increased validation rewards, indicating better generalization in the downstream task.

However, the results also revealed that overfitting can occur as the agent adapts to identify a limited number of "good" actions in the sets provided. To address this issue, future work should explore incorporating additional regularization techniques such as entropy maximization regularization and Kullback-Leibler Divergence Regularization. Furthermore, expanding the action representation space by extending action definitions or employing more sophisticated autoencoders, such as Variational Autoencoders (VAEs), could lead to even better performance and generalization.

By building upon the maRLios approach and addressing its limitations, future research can contribute to the development generalization in RL for gaming leading to more effective and efficient reinforcement learning agents that are capable of solving complex tasks and adapting to diverse environments.

References

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. 2021. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*.

Tianyue Cao. 2020. Study of sample efficiency improvements for reinforcement learning algorithms. In *2020 IEEE Integrated STEM Education Conference (ISEC)*, pages 1–1.

Yash Chandak, Georgios Theocharous, James Kostas,

Scott Jordan, and Philip S. Thomas. 2019. Learning action representations for reinforcement learning.

Yash Chandak, Georgios Theocharous, Chris Nota, and Philip Thomas. 2020. Lifelong learning with a changing action set. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:3373–3380.

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

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

Andrew Grebenisan. 2020. Blog. [link].

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. 2019. Multi-task deep reinforcement learning with popart. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3796–3803.

Ayush Jain, Andrew Szot, and Joseph Lim. 2020. Generalization to new actions in reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4661–4672. PMLR.

Christian Kauten. 2018. Super Mario Bros for OpenAI Gym. GitHub.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. 2019. A simple randomization technique for generalization in deep reinforcement learning. *CoRR*, abs/1910.05396.

Renxing Li, Zhiwei Shang, Chunhua Zheng, Huiyun Li, Qing Liang, and Yunduan Cui. 2022. Efficient Distributional Reinforcement Learning with Kullback-Leibler Divergence Regularization.

Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. 2019. A survey of deep reinforcement learning in video games.

Guy Tennenholtz and Shie Mannor. 2019. The natural language of actions. *CoRR*, abs/1902.01119.

Code Repository

https://github.com/joshmazen14/maRLios

Team Contributions

- **Cameron Witz**
  - Final Report
    - New Sections 5.4 to 5.8 (included)
    - Editing
    - RNN model architecture
    - RNN model training
    - Action sampling
    - Statistics gathering set up
    - Hyper Parameter Tuning
  - Midterm Report
    - Description of Solution
    - Environment Details
    - Set-up and training right only agent
  - Survey Report
    - Introduction
    - Existing work
    - Limitations
  - Project Proposal
    - Project Impact
    - Feasability and Data Sources
    - Approach
- **Carlos Osorio**
  - Final Report
    - Baseline
    - MaRLios Simple Movement Architecture & Training
    - MaRLios Generalizable Architecture & Training
    - Hyper Parameter Tuning
    - Validation Statistics
    - Discussion, Conclusion, Plots, Diagrams
  - Midterm Report
    - Modify environment to run on CARC with command line arguments
    - Set-up GCP training environment
    - Problem Statement, Architecture Diagrams
  - Survey Report
    - Introduction
    - Conclusion
    - Editing
  - Project Proposal
    - Project Objectives and Overview
    - Editing
- **Bozhie Pokorny**
  - Final Report
    - MaRLios model training variations on CARC
    - Rewards Training/Validation Plots
    - General Editing, Experiments, Discussion, Future Work
  - Midterm Report
    - Deployment of Baseline models on CARC
    - Results and Discussion and General Editing
  - Survey Report
    - Existing work
    - Editing
  - Project Proposal
    - Costs
    - Risks
    - Editing/Style
- **Josh Mazen**
  - Final Report
    - Baseline Model
    - Hyper Parameter Tuning
    - Collecting Test Data
    - Action Sampling
    - Results, Discussion, Final Editing
  - Midterm Report
    - Introduction
    - Evaluation and Metrics
    - Setup of Gym Environment
  - Survey Report
    - Existing Work
    - Editing
  - Project Proposal
    - Expected Timeline
    - Divison of Labor
    - Editing