Finding Action-State Similarities in Tabular Reinforcement Learning Using Low-Dimensional Embeddings

Gabriel Rubin

PUCRS

gdearrud@gmail.com

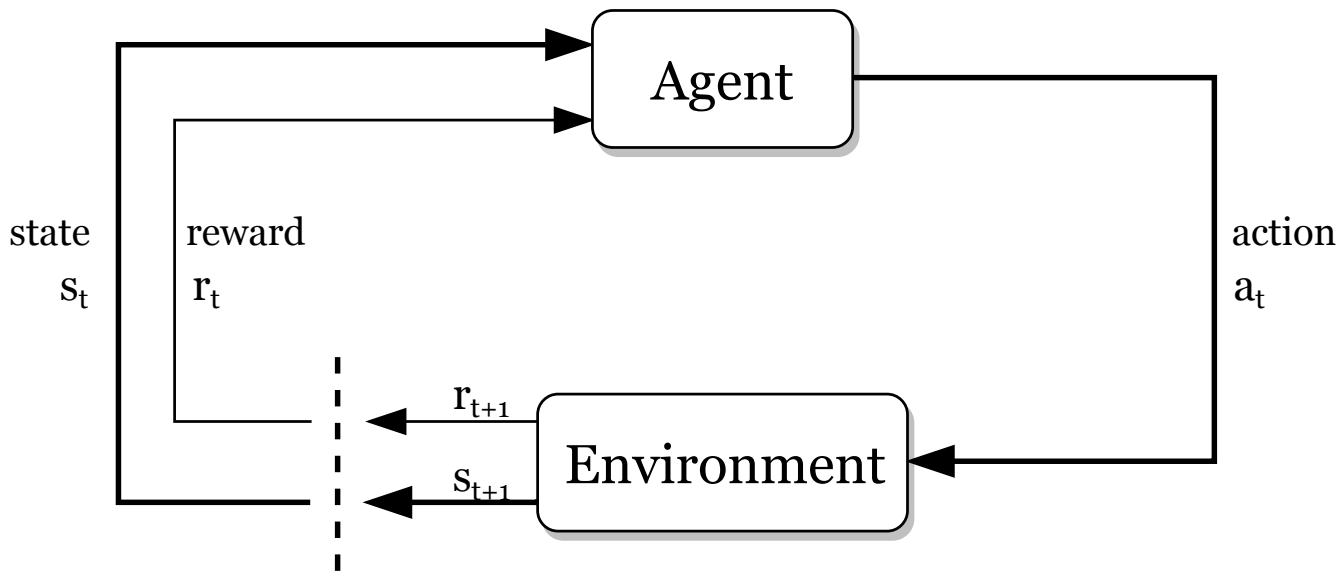
November 29, 2018

Overview

Background

Reinforcement Learning [4]

- **Represented through a Markov Decision Process (MDP)**
- **Agent learns a policy by interacting with an environment**
- **State transitions can be expressed as an experience tuple** $\langle s, a, r, s' \rangle$



Background

Reinforcement Learning Training Process

- During training, an RL agent interacts with the environment for several time-steps in order to learn a good policy
- This training process can often be expensive resource-wise and time consuming

Background

Q-Learning [5]

- **Model-free RL approach**
- **Calculates the estimated reward, or Q-Value, of an action-state pair $\langle a, s \rangle$**
- **Q-learning updates the Q-value estimation according to the temporal difference update rule**

Temporal difference update rule

$Q(s,a) = Q(s,a) + \alpha \cdot \delta$, **where** $\delta = r + max_{a'} Q(s^{'}, a^{'}) - Q(s,a)$

Previous Work

SASS Agent

Previous work introduces the State Action Similarity Solutions (SASS) approach that accelerate the training process by *spreading* the Q-Values of an action-state pair to similar action-state pairs [3].

- Similar action-states $\langle \tilde{a}, \tilde{s} \rangle$ are updated according to their similarity to action-state $\langle a, s \rangle$
- Similarities are calculated through a human-designed similarity function $\sigma : S \times A \times S \times A \mapsto [0, 1]$
- Each $\langle \tilde{a}, \tilde{s} \rangle$ runs a modified *temporal difference update rule*

$\langle \tilde{a}, \tilde{s} \rangle$ update rule

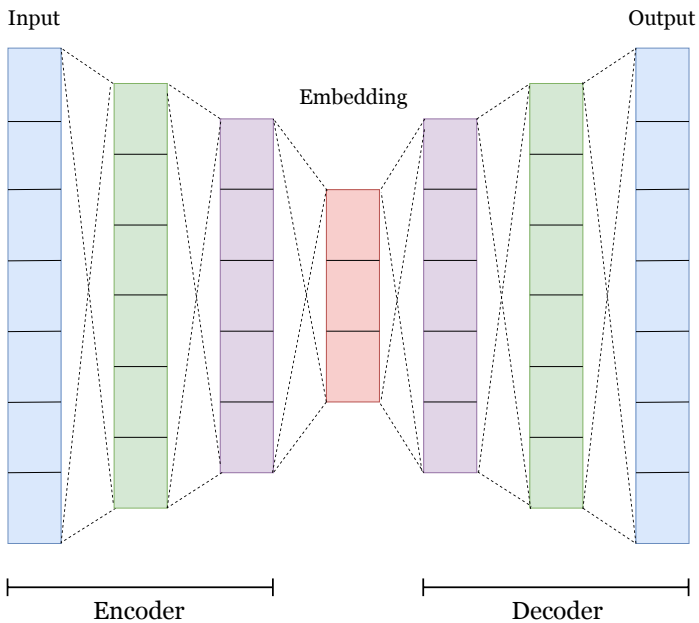
$$Q(\tilde{s}, \tilde{a}) = Q(\tilde{s}, \tilde{a}) + \alpha\sigma(s, a, \tilde{s}, \tilde{a})\delta$$

Motivation

- We find that SASS is a interesting approach for accelerating RL training, but its not a generic solution yet
- Its performance is dependent on the quality of the human-designed similarity function
- Because of this human input necessity, this approach is not domain-independent
- We implemented a solution to automatically detect action-state similarities that is domain-independent
- Then, we tested our solution in a Super Mario Bros domain that was also used to test the SASS agent and analysed our results

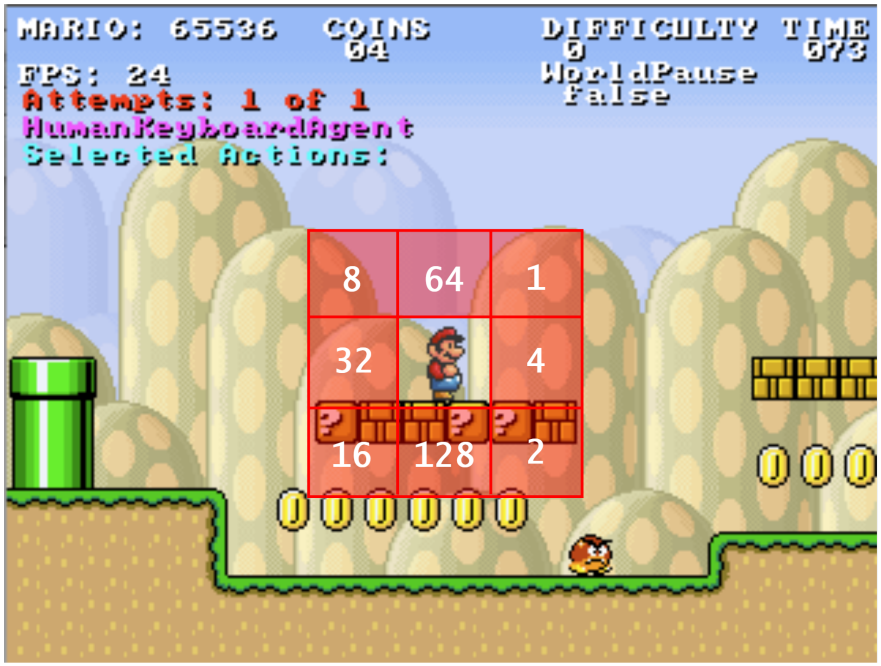
Our Method

- We propose a novel method for automatically finding similarities between action-state pairs
- Our similarity function $\sigma_A : S \times A \times S \times A \mapsto [0, 1]$ works by comparing the distances between the embedding of $\langle a, s \rangle$ [2].
- Embedding $E(a, s)$ is generated via an autoencoder [1], trained with $\langle a, s \rangle$ from the environment



Implementation

Super Mario Bros Domain



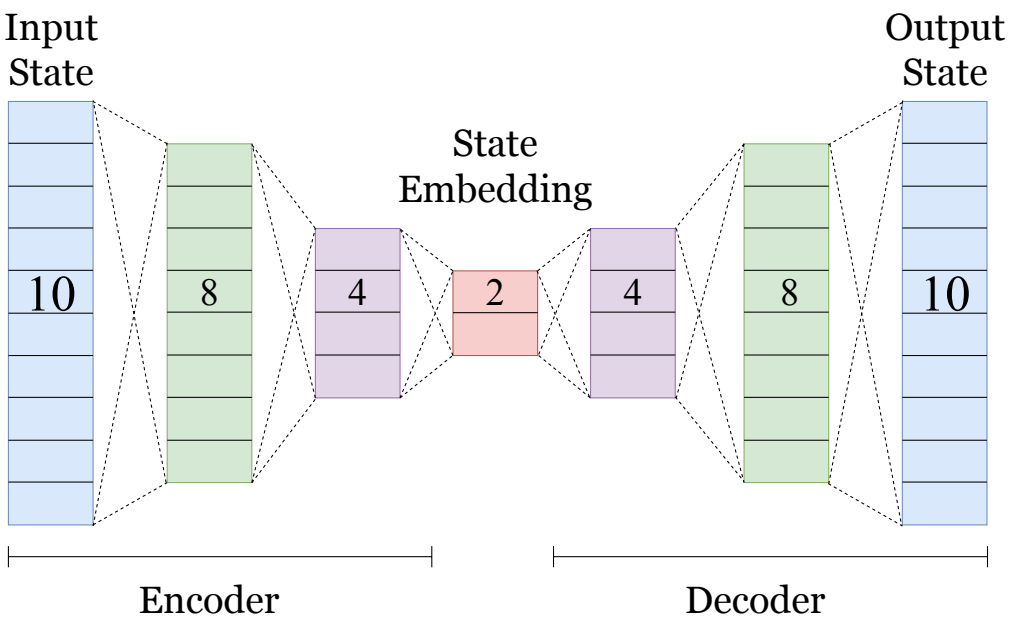
- Custom version of Super Mario Bros, implemented in Java for the Mario AI Competition
- Randomly generated levels
- Game state with 10 features
- Features contain information on Mario's state and surroundings

Implementation

Autoencoder

In order to compare the states from the Super Mario Bros domain, we implemented an autoencoder on Python using Keras.

- We trained it for 400 epochs using 2 million recorded state data from a regular Q-Learning agent
- Loss value close to 0.1



Implementation

Similarity Function σ_A

Our similarity method works by comparing the agent's last $\langle a, s \rangle$ embedding $E(a, s)$ with other $\langle \tilde{a}, \tilde{s} \rangle$ embedding $E(\tilde{a}, \tilde{s})$.

- Embeddings are calculated by calling our trained autoencoder model using the Java deep learning library DL4J.
- We considered 3 distance measures: Euclidian distance, Manhattan distance, and Cosine Similarity
- Distances are scaled by a threshold ϵ

Similarity Function

$$\sigma_A = 1 - \textsc{Min}(\textsc{Distance}(E(a, s), E(\tilde{a}, \tilde{s})) \cdot \tfrac{1}{\epsilon}, 1)$$

Implementation

Agent Implementation

Our agent utilizes an action-state history buffer in order to select $\langle \tilde{a}, \tilde{s} \rangle$ pairs for comparison with the current $\langle a, s \rangle$ pair.

- Each time Q(a, s) is updated, $\langle a, s \rangle$ is stored in the history buffer
- Extra updates are performed for each state in the history buffer
- The buffer's storing capacity C_b is configurable

Experiments

Setup

We tested different agent configurations in an easy setting of the Super Mario Bros domain. In each test, we measured the mean reward obtained by the agent for a period of 200.000 episodes. We tested the following agent configuration:

- 3 distance measures for σ_A: Euclidean distance, Manhattan distance, and Cosine Similarity
- With different history buffer capacities C_b
- Distance threshold ϵ was fixated at 1

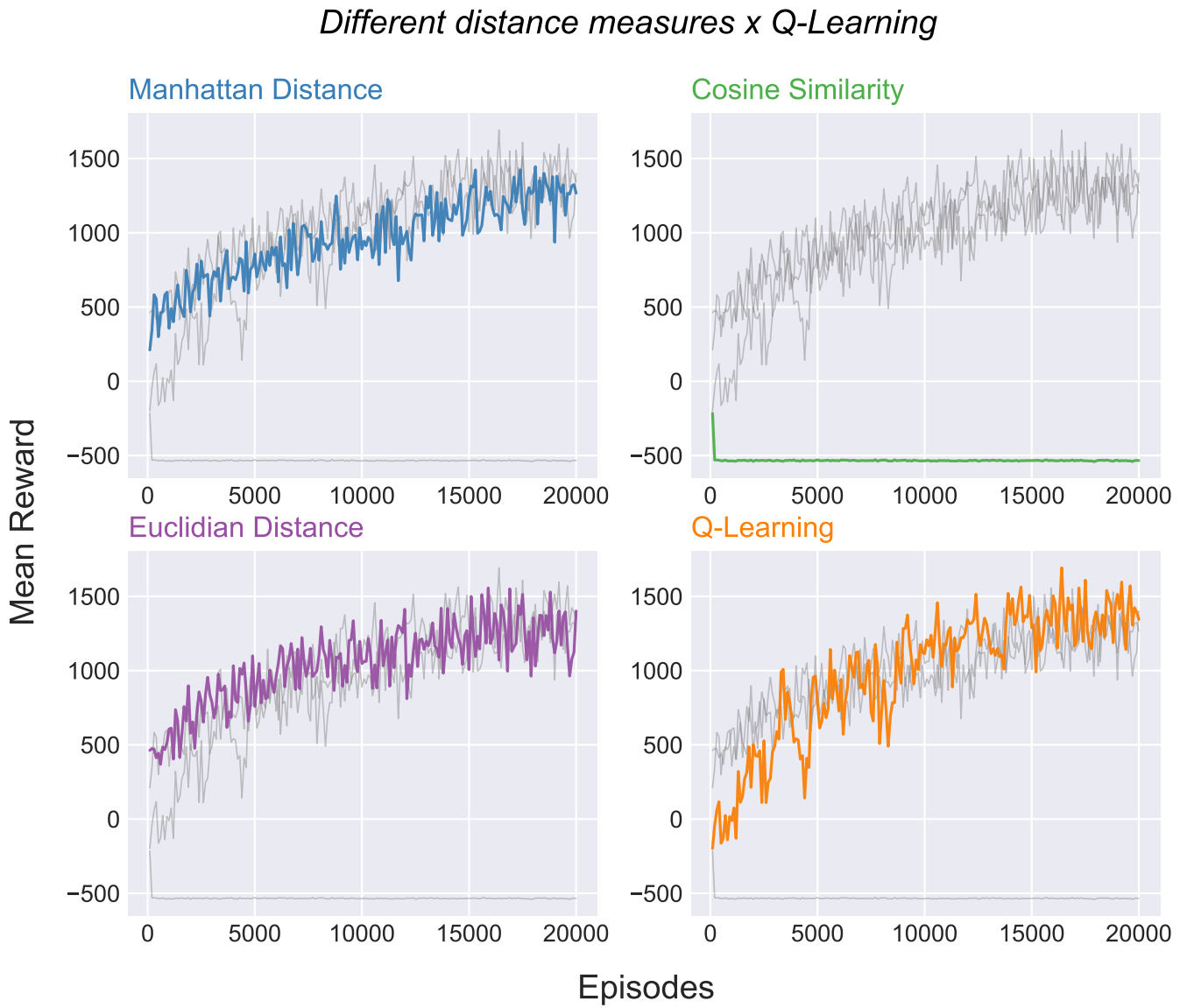
Experiments

Criteria

We compared our agents results with those obtained by a Q-Learning agent. Results were analyzed using the following criteria:

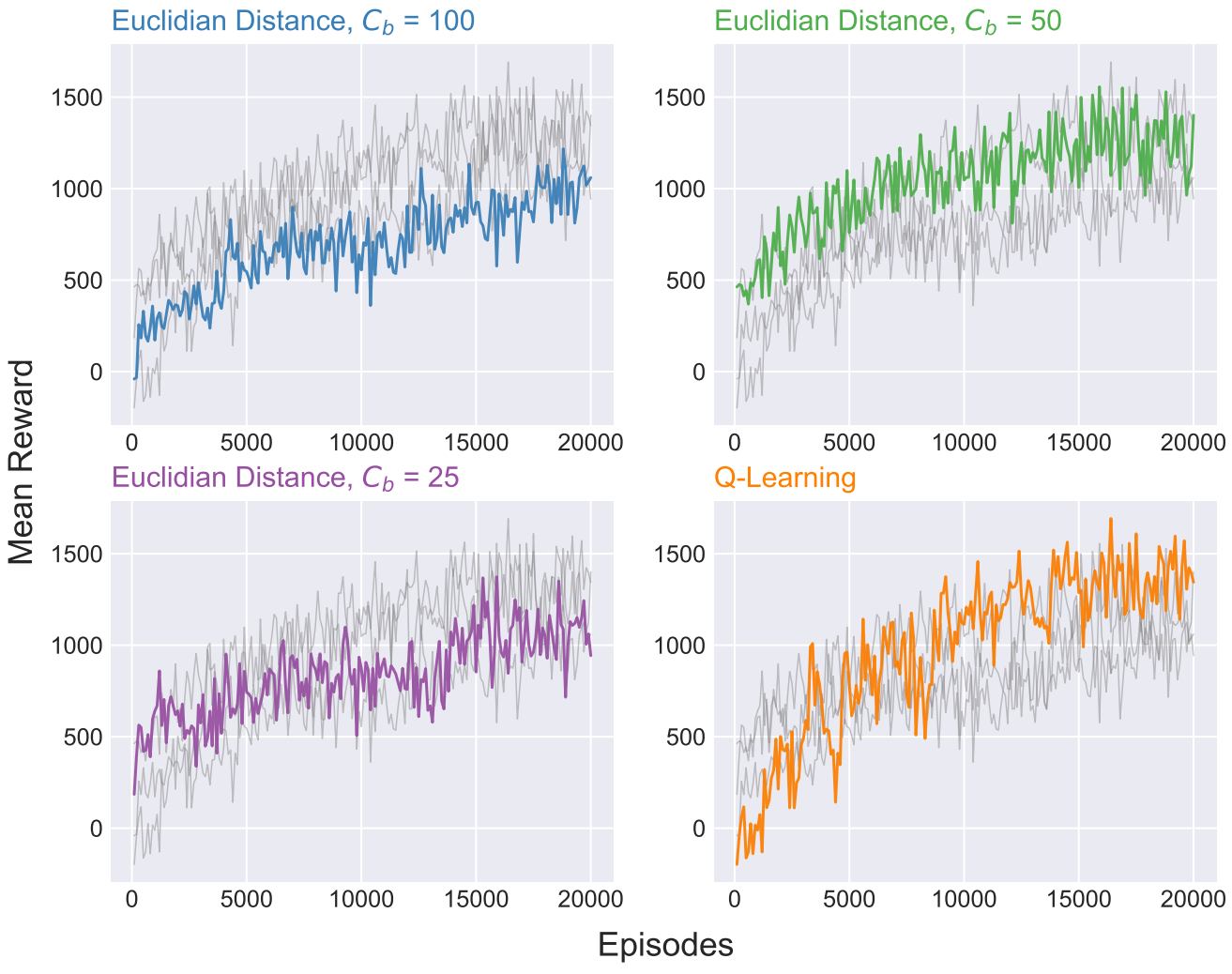
- Training stability: how steadily the mean rewards grow.
- Training speed: how fast the mean rewards grow.
- Final mean reward value: mean reward obtained at episode 200.000.

Results


Different distance measures x Q-Learning
Manhattan Distance
Cosine Similarity
Euclidian Distance
Q-Learning
Mean Reward
Episodes
1500
1000
500
0
−500
0
5000
10000
15000
20000

Results

Euclidian distance agent with various C_b values x Q-Learning



Analyzing Similarity Results

We concluded that our similarity function is not well suited for the Super Mario Bros domain. Let's demonstrate the problem with 2 state comparisons:

Mirrored states


Measures:

- Euclidian: 18.0
- Manhattan: 24.8
- Cosine Similarity: 0.1

On ground / Off ground


Measures:

- Euclidian: 0.75
- Manhattan: 0.9
- Cosine Similarity: 0.0

Conclusion

- We implemented a novel reinforcement agent that combines the training acceleration of the SASS agent with a method for finding action-state similarities automatically
- With the right configuration, our agent has a performance comparable to that of the Q-Learning agent in the challenging Super Mario Bros domain
- Our agent yields higher rewards at the first training episodes
- Nevertheless, we also found that our similarity function is not optimal for this domain
- We believe that this is due to the domain's state representation

Future work

- Test our approach in other domains
- Investigate our solution's performance in a domain that represent states via images, thus being less prune to design limitations
- Experiment with other buffer configurations and strategies
- Try a generational autoencoder for similar state generation

References

[1] Hinton, G. E., and Salakhutdinov, R. R.
Reducing the dimensionality of data with neural networks.
science 313, 5786 (2006), 504–507.

[2] Nikol'skii, S. M.
Approximation of functions of several variables and imbedding theorems, vol. 205.
Springer Science & Business Media, 2012.

[3] Rosenfeld, A., Taylor, M. E., and Kraus, S.
Speeding up tabular reinforcement learning using state-action similarities.
In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (2017), International Foundation for Autonomous Agents and Multiagent Systems, pp. 1722–1724.

[4] Sutton, R. S., Barto, A. G., Bach, F., et al.
Reinforcement learning: An introduction.
MIT press, 1998.

[5] Watkins, C. J., and Dayan, P.
Q-learning.
Machine learning 8, 3-4 (1992), 279–292.

Thank You!

Questions?