Scalable methods for computing state similarity in deterministic Markov Decision Processes

Pablo Samuel Castro
Google Brain
psc@google.com

Abstract

We present new algorithms for computing and approximating bisimulation metrics in Markov Decision Processes (MDPs). Bisimulation metrics are an elegant formalism that capture behavioral equivalence between states and provide strong theoretical guarantees on differences in optimal behaviour. Unfortunately, their computation is expensive and requires a tabular representation of the states, which has thus far rendered them impractical for large problems. In this paper we present a new version of the metric that is tied to a behavior policy in an MDP, along with an analysis of its theoretical properties. We then present two new algorithms for approximating bisimulation metrics in large, deterministic MDPs. The first does so via sampling and is guaranteed to converge to the true metric. The second is a differentiable loss which allows us to learn an approximation even for continuous state MDPs, which prior to this work had not been possible.

Introduction

A finite Markov Decision Process (MDP) is defined as a 5-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the next state transition function (where $\Delta(\mathcal{X})$ is the probability simplex over the set $\mathcal{X}$), $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function (assumed to be bounded by $\mathcal{R}_{\text{max}}$), and $\gamma \in [0, 1)$ is a discount factor. An MDP is the standard formalism for expressing sequential decision problems, typically in the context of planning or reinforcement learning (RL). The set of states $\mathcal{S}$ is one of the central components of this formalism, where each state $s \in \mathcal{S}$ is meant to encode sufficient information about the environment such that an agent can learn how to behave in a consistent manner. Figure 1 illustrates a simple MDP where each cell represents a state.

There is no canonical way of defining the set of states for a problem. Indeed, improperly designed state spaces can have drastic effects on the algorithm used. Consider the grid MDP in the bottom of Figure 1 where an agent must learn how to navigate to the green cells, and imagine we create an exact replica of the MDP such that the agent randomly transitions between the two layers for each move. By doing so we have doubled the number of states and the complexity of the problem. However, from a planning perspective the two copies of each state should be indistinguishable. A stronger notion of state identity is needed that goes beyond the labeling of states and which is able to capture behavioral indistinguishability.

In this paper we explore notions of behavioral similarity via state pseudometrics $d : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$, and in particular those which assign a distance of 0 to states that are behaviorally indistinguishable. Pseudometrics further allow us to reason about states based on what we may know about other similar states. This is a common use-case in fields such as formal verification, concurrency theory, and in safe RL, where one may want to provide (non-)reachability guarantees. In the context of planning and reinforcement learning, these can be useful for state aggregation and abstraction.

Our work builds on bisimulation metrics (Ferns, Panangaden, and Precup 2004) which provide us with theoretical guarantees such as states that are close to each other (with respect to the metric) will have similar optimal value functions. These theoretical properties render them appealing for planning and learning, and they have previously been used for state aggregation (Ferns, Panangaden, and Precup 2004, Ferns et al. 2006), policy transfer (Castro and Precup 2010), representation discovery (Ruan et al. 2015), and exploration (Santara et al. 2019).

Unfortunately, these metrics are expensive to compute (Chen, van Breugel, and Worrell 2012) and require fully enu-
merating the states, even when using sampling-based approxi-
mants (Ferns et al. 2006), on-the-fly approximants (Comanici, 
Panangaden, and Precup 2012) [Bacci et al. 2013b], or approxi-
mants which exploit structure in the state space (Bacci et al. 
2013a). The full-state enumeration requirement has thus 
far rendered bisimulation metrics impractical in problems 
with large state spaces, and completely incompatible with 
continuous state spaces. Additionally, bisimulation metrics 
can be overly pessimistic in the sense that they consider 
worst-case differences between states. Although desirable for 
certain applications, such as in guaranteeing safe behaviors, 
it can prove overly restrictive for many practical problems of 
interest.

In this paper we address these impracticalities with the 
following key contributions:
1. An on-policy variant of bisimulation which focuses only on 
the behavior of interest, rather than worst-case scenarios, 
along with an analysis of its theoretical properties.
2. A new sampling-based online algorithm for exact com-
puting the original and on-policy bisimulation metrics with 
guaranteed convergence.
3. A differentiable loss function for learning an approxima-
tion of the two bisimulation metrics using neural networks.

We provide empirical evidence of this learning algorithm 
on MDPs with large and continuous state spaces. To the 
best of the author’s knowledge, this is the first work proposing 
a mechanism for approximating bisimulation metrics 
with neural networks.

### Background

Given an MDP $M$, a policy $\pi : S \rightarrow \Delta(A)$ induces a 
corresponding state-value function $V^\pi : S \rightarrow \mathbb{R}$ (Puterman 1994): 
$V^\pi(s) = E_{a \sim \pi(s)} [R(s,a) + \gamma E_{s' \sim P(s,a)} V^\pi(s')]$. In the 
control setting, we are typically in search of the optimal 
value function (Bellman 1957):
$V^* = \max_{a \in A} [R(s,a) + \gamma E_{s' \sim P(s,a)} V^*(s')]$.

Bisimulation relations, originally introduced in the field of 
concurrency theory, were adapted for MDPs by Givan, 
Dean, and Greig (2003), capture a strong form of behavioral 
equivalence: if $s, t \in S$ are bisimilar, then $V^*(s) = V^*(t)$.

**Definition 1.** Given an MDP $M$, an equivalence relation 
$E \subseteq S \times S$ is a bisimulation relation if whenever $(s,t) \in E$ 
the following properties hold, where $S_E$ is the state space $S$ 
partitioned into equivalence classes defined by $E$:
1. $\forall a \in A, R(s,a) = R(t,a)$
2. $\forall a \in A, \forall c \in S_E, P(s,a)(c) = P(t,a)(c)$, where 
$P(s,a)(c) = \sum_{s' \in S} P(s,a)(s')$.

Two states $s, t \in S$ are bisimilar if there exists a bisimula-
tion relation $E$ such that $(s,t) \in E$. We denote the largest 
bisimulation relation as $\sim$.

Equivalence relations can be brittle: they require exact 
equivalence under probabilistic transitions. This can be es-
pecially problematic if we are estimating transition probabilities 
from data, as it is highly unlikely they will match exactly.

Extending the work of Desharnais et al. (1999) for labeled 
Markov processes, Ferns, Panangaden, and Precup (2004) 
generalized the notion of MDP bisimulation relations to met-
rics, yielding a smoother notion of similarity than equivalence 
relations. Let $M$ be the set of all pseudometrics on $S$. A pseudometric $d \in M$ induces an equivalence relation 
$E_d := \{(s,t)|d(s,t) = 0\}$. That is, any two states with 
distance 0 will be collapsed onto the same equivalence class.

**Definition 2.** (Ferns, Panangaden, and Precup 2004) A 
pseudometric $d \in M$ is a bisimulation metric if $E_d$ is $\sim$.

Bisimulation metrics use the 1-Wasserstein metric $W_1 : 
M \rightarrow \mathbb{P}$, where $\mathbb{P}$ is the set of all metrics between 
probability distributions over $S$. Given two state distributions 
$X, Y \in \Delta(S)$ and a pseudometric $d \in M$, the Wasserstein 
$W_1(d)(X,Y)$ can be expressed by the following (primal) 
linear program (LP), which “lifts” a pseudometric $d \in M$ onto one in $\mathbb{P}$ (Villani 2008):
$\max_u \in \mathbb{R}^{|S|} \sum_{s \in S} (X(s) - Y(s)) u_s$ (1) 
$\forall s, s' \in S, u - u' \leq d(s,s')$ 
$0 \leq u \leq 1$

**Theorem 1.** (Ferns, Panangaden, and Precup 2004): Define 
$F : M \rightarrow M$ by
$F(d)(s,t) = \max_{a \in A} \{|R(s,a) - R(t,a)| + \gamma W_1(d)(P(s,a), P(t,a))\}$
then $F$ has a unique fixed point, $d_\sim$, and $d_\sim$ is a bisimulation 
metric.

The operator $F$ can be used to iteratively compute a 
bisimulation metric as follows. Starting from an initial esti-
mate $d_0$, we can compute $d_{n+1} = F(d_n) = F^{n+1}(d_0)$. 
By iteratively applying $F$ $\lceil \ln \frac{1}{\delta} \rceil$ times, one can compute 
$d_\sim$ up to an accuracy $\delta$, with an overall complexity of 
$O (|A||S|^4 \log |S| \ln \frac{1}{\delta} )$.

### On-policy bisimulation

The strong theoretical guarantees of bisimulation relations 
and metrics are largely due to their inherent “pessimism”: 
they consider equivalence under all actions, even pathologi-
cally bad ones (i.e. actions that never lead to positive out-
comes for the agent). Indeed, there exist systems where 
$V^*(s) = V^*(t)$, but $d_\sim(s,t)$ can be arbitrarily large, 
providing no useful bounds on the optimal behaviour from $s$ 
and $t$ (see Figure 2). Castro and Precup (2010) also 
demonstrated that this pessimism yields poor results when using 
bisimulation metrics for policy transfer.

Another disadvantage of bisimulation relations and metrics 
is that they are computed via exact action matching between 
states; however, actions with the same label may induce very 
different behaviours from different states, resulting in 
an improper behavioral comparison when using bisimulation. In
the system in Figure 2, and $t$ have equal optimal values, but their optimal action is different ($a$ from $s$, $b$ from $t$). Taylor, Precup, and Panagaden (2009) overcame this problem by the introduction of lax-bisimulation metrics (definition and theoretical results provided in the supplemental). We note, however, that their method is still susceptible to the pessimism discussed above.

It is often the case that one is interested in behaviours relative to a particular policy $\pi$. In reinforcement learning, for example, many algorithms maintain a behaviour policy which is improved iteratively as the agent interacts with the environment. In these situations the pessimism of bisimulation can become a hindrance: if the action maximizing the environment. In these situations the pessimism of bisimulation metrics can become a hindrance: if the action maximizing the distance between two states is never chosen by $\pi$, we should not include it in the computation!

We introduce a new notion of bisimulation, on-policy bisimulation, defined relative to a policy $\pi$. This new notion also removes the requirement of matching on action labels by considering the dynamics induced by $\pi$, rather than the dynamics induced by each action. We first define:

$$R^\pi_s := \sum_a \pi(a|s)R(s,a)$$

$$\forall C \in S_{E^\pi}, P^\pi_s(C) := \sum_a \pi(a|s) \sum_{s' \in C} P(s,a)(s')$$

**Definition 3.** Given an MDP $\mathcal{M}$, an equivalence relation $E^\pi \subseteq S \times S$ is a $\pi$-bisimulation relation if whenever $(s,t) \in E^\pi$ the following properties hold:

1. $R^\pi_s = R^\pi_t$
2. $\forall C \in S_{E^\pi}$, $P^\pi_s(C) = P^\pi_t(C)$

Two states $s,t \in S$ are $\pi$-bisimilar if there exists a $\pi$-bisimulation relation $E^\pi$ such that $(s,t) \in E^\pi$. Denoting the largest bisimulation relation as $\sim_\pi$, $d \in \mathbb{M}$ is a $\pi$-bisimulation metric if $E_d$ is $\sim_\pi$.

**Theorem 2.** Define $F^\pi : \mathcal{M} \to \mathcal{M}$ by $F^\pi(d)(s,t) = |R^\pi_s - R^\pi_t| + \gamma W_1(d)(P^\pi_s, P^\pi_t)$, then $F^\pi$ has a least fixed point $d^\pi_\pi$, and $d^\pi_\pi$ is a $\pi$-bisimulation metric.

**Proof.** (Sketch) This proof mimics the proof of Theorem 4.5 from Ferns, Panangaden, and Precup (2004). All complete proofs are provided in the supplemental material.

The following result demonstrates that $\pi$-bisimulation metrics provide similar theoretical guarantees as regular bisimulation metrics, but with respect to the value function induced by $\pi$.

**Theorem 3.** Given any two states $s,t \in S$ in an MDP $\mathcal{M}$, $|V^\pi(s) - V^\pi(t)| \leq d^\pi_\pi(s,t)$.

Proof. (Sketch) This is proved by induction. The result follows by expanding $V^\pi$, taking the absolute value of each term separately, and noticing that $V^\pi$ is a feasible solution to the primal LP in Equation 1, so is upper-bounded by $\mathcal{W}_1(d^\pi_\pi)$.

Under a fixed policy $\pi$, an MDP reduces to a Markov chain. Bisimulation relations for Markov chains have previously been studied in concurrency theory (Baier et al. 2006; Katoen et al. 2007). Further, $\pi$-bisimulation can be used to define a notion of weak-$\pi$-bisimulation for MDPs (Baier et al. 2006; Ferrer Fioriti et al. 2016).

**Bisimulation metrics for deterministic MDPs**

In this section we investigate the properties of deterministic MDPs, which in concurrency theory are known as transition systems (Sangiorgi 2011).

**Definition 4.** A deterministic MDP $\mathcal{M}$ is one where for all $s \in S$, $a \in A$, and pseudometric $d \in \mathbb{M}$, $\mathcal{W}_1(d)(\mathcal{P}(s,a),\mathcal{P}(t,a)) = d(\mathcal{N}(s,a),\mathcal{N}(t,a))$.

As the next lemma shows, under a system with deterministic transitions, computing the Wasserstein metric (and approximants) is no longer necessary.

**Lemma 1.** Given a deterministic MDP $\mathcal{M}$, for any two states $s,t \in S$, action $a \in A$, and pseudometric $d \in \mathbb{M}$, $\mathcal{W}_1(d)(\mathcal{P}(s,a),\mathcal{P}(t,a)) = d(\mathcal{N}(s,a),\mathcal{N}(t,a))$.

Proof. (Sketch) The result follows by considering the dual formulation of the primal LP in Equation 1, which implies the dual variables $\lambda_{s,t}$ must all be either 1 or 0, by virtue of determinism.

By considering only deterministic policies (e.g. policies that assign probability 1 to a single action) in the on-policy case, Lemma 1 allows us to rewrite the operator $F(d)(s,t)$ in Theorem 1 and $F^\pi(d)(s,t)$ in Theorem 2 as:

$$\max_{a \in A} (|\mathcal{R}(s,a) - \mathcal{R}(t,a)| + \gamma d(\mathcal{N}(s,a),\mathcal{N}(t,a)))$$

and

$$|\mathcal{R}(s,\pi(s)) - \mathcal{R}(t,\pi(t))| + \gamma d(\mathcal{N}(s,\pi(s)),\mathcal{N}(t,\pi(t)))$$

respectively. Note the close resemblance to value functions, there is in fact a strong connection between the two: Ferns and Precup (2014) proved that $d_{\pi}$ is the optimal value function of an optimal coupling of two copies of the original MDP.

Even in the deterministic setting, the computation of bisimulation metrics can be intractable in MDPs with very large or continuous state spaces. In the next sections we will leverage the results just presented to introduce new algorithms that are able to scale to large state spaces and learn an approximant for continuous state spaces.
Computing bisimulation metrics with sampled transitions

We present the algorithm and results in this section for the original bisimulation metric, \(d_\infty\), but all the results presented here hold for the on-policy variant \(d_\infty^\pi\); the main difference is that actions in the trajectory are given by \(\pi\) and thus, may differ between states being compared.

The update operator \(\mathcal{F}\) is generally applied in a dynamic-programming fashion: all state-pairs are updated in each iteration by considering all possible actions. However, re-executing actions at all state-pairs and actions in each iteration is often not possible, especially when data is concurrently being collected by an agent interacting with an environment. In this section we present an algorithm for computing the bisimulation metric via access to transition samples. Specifically, assume we are able to sample pairs of transitions \([s, a, \mathcal{R}(s, a), \mathcal{M}(s, a)], (t, a, \mathcal{R}(t, a), \mathcal{M}(t, a))]\) from an underlying distribution \(\mathcal{D}\) (note the action is the same for both). This can be, for instance, a uniform distribution over all transitions in a replay memory [Mnih et al. 2015] or some other sampling procedure. Let \(\mathcal{T}\) be the set of all pairs of valid transitions; for legibility we will use the shorthand \(\tau_{s,t,a} \in \mathcal{T}\) to denote a pair of transitions from states \(s, t \in \mathcal{S}\) under action \(a \in \mathcal{A}\). We assume that \(\mathcal{D}(\tau) > 0\) for all \(\tau \in \mathcal{T}\).

We first define an iterative procedure for computing \(d_n\) by sampling from \(\mathcal{D}\). Let \(d_0 \equiv 0\) be the everywhere-zero metric. At step \(n\), let \(\tau_{s_n, t_n, a_n} \in \mathcal{T}\) be a sample from \(\mathcal{D}\) and define \(d_n\) as:

\[
\begin{align*}
d_n(s, t) &= d_{n-1}(s, t), \quad \forall s \neq t, \forall t, \forall n \\
d_n(s, t) &= \max \left( d_{n-1}(s, t), \right.
\end{align*}
\]

for \(d_n(s, t) := \max \left( d_{n-1}(s, t) \right)\) for \(d_n(s, t)\) and \(d_n(t, s)\).

In words, when we sample a pair of states, we only update the distance estimate for these two states if applying the \(\mathcal{F}\) operator gives us a larger estimate. Otherwise, our estimate remains unchanged.

**Theorem 4.** If \(d_n\) is updated as in Equation 3 and \(d_0 \equiv 0\), \(\lim_{n \to \infty} d_n = d_\infty\) almost surely.

**Proof.** (Sketch) We first show that since we are sampling state pairs and actions infinitely often, all state pairs will receive a non-vacuous update at least once (Maximizing action lemma); then show that \(d_n \leq d_{n-1}\) for all \(n \in \mathbb{N}\) (Monotonicity lemma). We then use these two results to show that the difference \(\|d_\infty - d_n\|_\infty\) is a contraction and the result follows by the Banach fixed-point theorem. Note that the maximizing action lemma as presented here is for the original bisimulation metric; for the on-policy variant, the equivalent result is that all states in the Markov chain induced by \(\pi\) are updated infinitely often.

**Learning an approximation**

We leverage the sampling approach from the previous section to devise a learning algorithm for approximating \(d_\infty\) and \(d_\infty^\pi\) for MDPs with large (or continuous) state spaces, using function approximators in the form of neural networks. Let \(\phi : \mathcal{S} \to \mathbb{R}^k\) be a \(k\)-dimensional representation of the state space and let \(\psi : \mathbb{R}^{2k} \to \mathbb{R}\) be a neural network parameterized by \(\theta\) that receives a concatenation of two state representations such that \(\psi_\theta(\phi(s), \phi(t)) \approx d_\infty(s, t)\) (see Figure 3).

Following the practice introduced by [Mnih et al. 2015] we make use of online parameters \(\theta\) and target parameters \(\theta^\tau\), where the online parameters are updated at each iteration while the target parameters are updated every \(C\) iterations. Given a pair of states \(s \neq t\) and action \(a \in \mathcal{A}\), at iteration \(n\) we define the target objective \(T_{\theta^\tau}(s, t, a)\) for \(d_n\) as:

\[
\max \left[ \mathcal{R}(s, a) - \mathcal{R}(t, a) + \gamma \psi_\theta \left( \phi(\mathcal{M}(s, a)), \phi(\mathcal{N}(t, a)) \right) \right]
\]

and equal to 0 whenever \(s = t\). The target objective \(T_{\theta^\tau}(s, t, a)\) for \(d_n^\pi\) is:

\[
\max \left[ \mathcal{R}(s, \pi(s)) - \mathcal{R}(t, \pi(t)) + \gamma \psi_\theta \left( \phi(\mathcal{M}(s, \pi(s))), \phi(\mathcal{N}(t, \pi(t))) \right) \right]
\]

We can then define our loss \(L_n(\pi)\) as:

\[
\mathbb{E}_D \left[ \left( T_{\theta^\tau}(s, t, a) - \psi_\theta \left( \phi(s), \phi(t) \right) \right)^2 \right]
\]

\[
L_n^\pi(\theta_i) = \mathbb{E}_D \left[ \left( \psi_\theta \left( \phi(s^2) - T \right) \right)^2 \right]
\]

\[
L_n^\pi(\theta_i) = \mathbb{E}_D \left[ \left( \psi_\theta \left( \phi(s^2) - T \right) \right)^2 \right]
\]

\(\mathbb{R}^2, \mathbb{S}^2,\) and \(\mathbb{N}^2\) are batches of rewards, states, and next-states, respectively, \(\mathbb{W}\) is a mask used to enforce action matching when approximating \(d_\infty\), and \(I\) is the identity matrix. \(\psi(X)\) indicates applying \(\psi\) to a matrix \(X\) elementwise, and \(\otimes\) stands for the Hadamard product. We multiply by \((1 - I)\) to zero out the diagonals, since those represent approximations to \(d_\infty(s, s)\) and \(d_\infty^\pi(s, s)\).

**Figure 3:** Using a neural network for learning \(\psi\) as an approximator to \(d_\infty\) or \(d_\infty^\pi\).
Training was done on a Tesla P100 GPU.

Because our methods are designed for deterministic MDPs, we only used those trained without sticky actions provided with the Dopamine library (Castro et al. 2018); (evaluated in Section 4.3 in (Castro et al. 2018)); the trained checkpoints were provided for only three games: the Arcade Learning Environment (Bellemare et al. 2013).

In this section we provide empirical evidence for the effectiveness of our bisimulation approximant. We begin with a simple 31-state GridWorld, on which we can compute the bisimulation metric exactly, and use a “noisy” representation which yields a continuous-state MDP. Having the exact metric for the 31-state MDP allows us to quantitatively measure the quality of our learned approximant. A lengthier discussion, including the derivation of these matrices, is provided in the supplemental material.

**Empirical evaluation**

In this section we provide empirical evidence for the effectiveness of our bisimulation approximant. We begin with a simple 31-state GridWorld, on which we can compute the bisimulation metric exactly, and use a “noisy” representation which yields a continuous-state MDP. Having the exact metric for the 31-state MDP allows us to quantitatively measure the quality of our learned approximant.

We then learn a $\pi$-bisimulation approximant over policies generated by reinforcement learning agents trained on Atari 2600 games. In the supplemental material we provide an extensive discussion of the hyperparameter search we performed to find the settings used for both experiments. Training was done on a Tesla P100 GPU.

**GridWorld**

We first evaluate our learning algorithms on the 31-state GridWorld environment illustrated in Figure 4. There are 4 actions (up, down, left, right) with deterministic transitions, and where an action driving the agent towards a wall keeps the agent in the same cell. There is a single reward of +1.0 received upon entering either of the green cells, and a reward of −1.0 for taking an action towards a wall. We display the bisimulation distances between all states in the supplemental, computed using the sampling approach.

We represent each state by its coordinates $(x, y) \in \mathbb{R}^2$, as illustrated in Figure 4. To estimate $d_\pi$, we use a network with an input layer of dimension 4, one fully connected hidden layer with 729 hidden units, and an output of length 1. The input is the concatenation of two state representations, normalized to be in $[-1, 1]$, while the output value is the estimate to $d_\pi$. We sampled state pairs and actions uniformly randomly, and ran our experiments with $\gamma = 0.99, C = 500, b = 256$, and increased $\beta$ from 0 to 1 by a factor of 0.9 every time the target network was updated; we used the Adam optimizer (Kingma and Ba 2015) with a learning rate of 0.01. Because of the maximization term in the target, these networks can have a tendency to overshoot (although the combination of target networks and the $\beta$ term helps stabilize this); we ran the training process for 2500 steps, which, for this problem, was long enough to converge to a reasonable solution before overshooting. The full hyperparameter settings are provided in the supplemental.

To evaluate the learning process, we measure the absolute error: $\|d_\pi - \psi\|_\infty$ using the true underlying state space for which we know the value of $d_\pi$. Note that because there is no fixed learning target, absolute errors are not guaranteed to be bounded. For this reason we also report the normalized error: $\left\|\frac{d_\pi - \psi}{\|d_\pi\|_\infty}\right\|_\infty$ as in practice one is mostly interested in relative, rather than absolute, distances. Figure 5a and Figure 5b display the results of our experiments over 10 independent runs; the shaded areas represent the 95% confidence interval.

In addition to training on the 31 state-MDP, we constructed a continuous variant by adding Gaussian noise to the state representations; this noise is centered at $(0, 0)$ with standard deviation 0.1, and clipped to be in $[-0.3, 0.3]$. The per-cell noise is illustrated by the black gradients in Figure 4. As Figure 5 shows, there is little difference between learning the metric for the 31-state MDP versus learning it for its continuous variant. Adding noise does not seem to hurt performance, and in fact seems to be helpful. We hypothesize that noise may be acting as a form of regularization, but this requires further investigation. In the supplemental material we include a figure exploring using the metric approximant for aggregating states in the continuous MDP setting with promising results.

**Atari 2600**

To evaluate the performance of our learning algorithm on an existing large problem, we take a set of reinforcement learning (RL) agents trained on three Atari 2600 games from the Arcade Learning Environment (Bellemare et al. 2013). The RL agents were obtained from the set of trained agents provided with the Dopamine library (Castro et al. 2018). Because our methods are designed for deterministic MDPs, we only used those trained without sticky actions (Machado et al. 2018) (evaluated in Section 4.3 in (Castro et al. 2018)); the trained checkpoints were provided for only three games: Space Invaders, Pong, and Asterix. We used the Rainbow

---

Code available at https://github.com/google-research/google-research/tree/master/bisimulation_aaai2020

Sticky actions add stochasticity to action outcomes.
To approximate the on-policy bisimulation metric $d_\pi^\gamma$ we loaded a trained agent and ran it in evaluation mode for each respective game, filling up the replay buffer while doing so. Once 10,000 transitions have been stored in the replay buffer, we begin sampling mini-batches and update our approximant $\psi_\pi^\gamma$ using the target and loss defined previously. (note that we still continue populating our replay buffer). We ran our experiments using a network of two hidden layers of dimension 16, with $\gamma = 0.99$, $C = 500$, $b = 128$, and increased the $\beta$ term from 0 to 1 by a factor of 0.99 every time the target network was updated. We used the Adam optimizer (Kingma and Ba 2015) with a learning rate of $7.5 \times 10^{-5}$ (except for Pong where we found 0.001 yielded better results). We trained the networks for around 600K steps, although in practice we found that about half that many steps were needed to reach a stable approximant. The configuration file specifying the full hyperparameter settings as well as the learning curves are provided in the supplemental.

After training we evaluated our approximant $\psi_\pi^\gamma([\phi(s), \phi(t)])$ by fixing $s$ to be the first state in the game and varying $t$ throughout the episode; that is, we evaluate how similar the other states of the game are to the initial state w.r.t. our approximant. In Figure 6 we display the first 500 steps of one evaluation run on Space Invaders; as can be seen, the learned metric captures more meaningful differences between frames (start of episodes, enemy alien destroyed) that go beyond simple pixel differences. Interestingly, when sorting the frames by distance, the frames furthest away from $s$ are typically those where the agent is about to be killed. It is worth noting that the way states are encoded in Dopamine is by stacking the last four frames; in our visualization we are only displaying the top frame in this stack. We observed similar results for Asterix and Pong; we include these and more extensive results, as well as videos for the three games, in the supplemental material.

Related work

There are a number of different notions of state similarity that have been studied over the years. Li, Walsh, and Littman (2006) provide a unified characterization and analysis of many of them. MDP-homomorphisms (Ravindran and Barto 2003) do not require behavioral characterization under the same action labels, and this idea was extended to a metric by Taylor, Precup, and Panagaden (2009).

Ferns et al. (2006) introduced a sampling-based approximation to $d_\pi$ which exchanges the computation of the Wasserstein with an instance of the assignment problem from network optimization; although Castro (2011) derived a PAC-bound for this approximant, the number of samples required is still prohibitive. Bacci et al. (2013a) exploit the underlying structure in $S$ to compute $d_\pi$.

Although there has been some work in concurrency theory to approximate large systems via 'polynomially accurate' simulations (Segala and Turrini 2007), they make no use of function approximators in the form of neural networks. We believe our use of neural networks may grant our approach greater generalizability.

Deterministic on-policy systems can be reduced to a graph. As such, our notion of $\pi$-bisimulation metrics bears a close relationship to graph similarity measures (Zager and Verghese 2008). However, graph similarity notions compare two full systems (graphs), as opposed to two nodes within a single graph, as we evaluate here. Nonetheless, the relationship warrants further investigation, which we leave for future work.

Perhaps most related to our sampling method is the on-the-fly methods introduced by Comanici, Panangaden, and Precup (2012). The authors replace the use of standard dynamic programming in their computation with something akin to asynchronous dynamic programming (Sutton and Barto 1998), where not all state-pairs are updated at each iteration, but rather $S$ is split into disjoint sets that are updated at different intervals. A few strategies for sampling state-pairs are discussed, of which the most similar to ours is the "uniform asynchronous update".

Gelada et al. (2019) introduced DeepMDP latent models and established a close relationship with bisimulation metrics (specifically, Theorem 4 in their paper). Although closely related, there are some important differences. Their work deals
Figure 6: Evaluating the approximant to $d^\pi_\sim$ with an eval run on Space Invaders. We plot the distance between the source state $s$ (pictured in the bottom left) and every other state, highlighting the relatively low distances in game or level starts (green shading pointing to left and right side of the plot), as well as the peaks occuring when an enemy alien is destroyed (blue shading pointing to the distance peaks). On the bottom right we display the distribution of distances and the four furthest states for this run.

with state representations, where the distance between states is their distance in the representation space; by contrast, our proposed neural networks approximate the bisimulation metric between two states, independent of their representation. Further, the authors use the DeepMDP losses as an auxiliary task without a guarantee that their representations are consistent with their theoretical results. In our work we are able to show that our approximant is valid both quantitatively (GridWorld) and qualitatively (Atari 2600). Nonetheless, a natural extension of our work is to use the bisimulation losses we introduced as a means to learn better representations.

Conclusion

We introduced new methods for computing and approximating bisimulation metrics in large deterministic MDPs. The first is an *exact* method that converges to the true metric asymptotically, and the second is a differentiable method for approximating the metric which we demonstrated can learn a good approximant even in continuous state spaces. Since their introduction, bisimulation metrics have been used for theoretical analysis or in MDPs of small-to-moderate size, but they have scarcely been used in larger problems. Our results open the doors for their application in systems with a large, and even continuous, state space.

One important avenue for research is to extend these results to stochastic systems. Computing the Wasserstein metric without access to a generative model is challenging for deep RL environments, as the next-state samples typically come from single trajectories in replay buffers. One possibility is to build a model of the transition dynamics from the transitions in the replay buffer and compute the Wasserstein metrics from this estimate.

Although the network architecture and hyperparameters used to train $d^\pi_\sim$ are by no means optimal, the results we presented for the Atari 2600 domain are very promising and suggest that bisimulation metrics can be used effectively for deep reinforcement learning. Some promising areas we are currently exploring are using bisimulation metrics as an auxiliary task for improved state representation, as a mechanism for compressing replay buffers, and as a tool for more efficient exploration.

Acknowledgements

The author would like to thank Marc G. Bellemare, Gheorghe Comanici, Marlos C. Machado, Doina Precup, Carles Gelada, as well as the rest of the Google Brain team in Montreal for helpful discussions. The author would also like to thank the anonymous reviewers for their useful feedback while reviewing this work.

References

[2015] Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems.

[2013a] Bacci, G.; Bacci, G.; Larsen, K. G.; and Mardare, R. 2013a. Computing Behavioral Distances, Compositionally. In *Mathematical Foundations of Computer Science 2013*.

[2013b] Bacci, G.; Bacci, G.; Larsen, K. G.; and Mardare, R. 2013b. On-the-Fly Exact Computation of Bisimilarity Distances. In *Proceedings of the 19th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*. 

Proofs of the theoretical results

**Theorem 2.** Define $F^\pi : M \rightarrow M$ by $F^\pi(d)(s,t) = |R^\pi_s - R^\pi_t| + \gamma W_V(d)(P^\pi_s, P^\pi_t)$, then $F^\pi$ has a least fixed point $d^\pi_\perp$, and $d^\pi_\perp$ is a $\pi$-bisimulation metric.

**Proof.** This proof mimics the proof of Theorem 4.5 from [Ferns, Panangaden, and Precup 2004] (included as Theorem 1 in this paper). Lemma 4.1 from that paper holds under Definition 3 by continuity. We make use of the same pointwise ordering on $M$: $d \leq d'$ iff $d(s,t) \leq d'(s,t)$ for all $s, t \in S$, which gives us an $\omega$-cpo with bottom $\perp$, which is the everywhere-zero metric. Since Lemma 4.4 from [Ferns, Panangaden, and Precup 2004] ($W$ is continuous) also applies in our definition, it only remains to show that $F^\pi$ is continuous:

$$F^\pi\left(\bigcup_{n \in \mathbb{N}} \{x_n\}\right)(s,t) = |R^\pi_s - R^\pi_t| + \gamma W\left(\bigcup_{n \in \mathbb{N}} \{x_n\}\right)(P^\pi_s, P^\pi_t)$$

$$= |R^\pi_s - R^\pi_t| + \gamma \sup_{n \in \mathbb{N}} W(x_n)(P^\pi_s, P^\pi_t) \quad \text{by continuity of } W$$

$$= \sup_{n \in \mathbb{N}} (|R^\pi_s - R^\pi_t| + \gamma W(x_n)(P^\pi_s, P^\pi_t))$$

$$= \sup_{n \in \mathbb{N}} (F^\pi(x_n)(s,t))$$

$$= \left(\bigcup_{n \in \mathbb{N}} \{F^\pi(x_n)\}\right)(s,t)$$

The rest of the proof follows in the same way as in [Ferns, Panangaden, and Precup 2004].

**Theorem 3.** Given any two states $s, t \in S$ in an MDP $M$, $|V^\pi(s) - V^\pi(t)| \leq d^\pi_\perp(s,t)$.

**Proof.** We will use the standard value function update: $V^\pi_n(s) = R^\pi_s + \gamma \sum_{s' \in S} P^\pi_s(s') V^\pi_{n-1}(s')$ with $V^\pi_0 \equiv 0$ and our update operator from Theorem 2 $d^\pi_n(s,t) = F^\pi(d^\pi_{n-1})(s,t)$ with $d^\pi_0 \equiv 0$, and prove this by induction, showing that for all $n \in \mathbb{N}$ and $s, t \in S$, $|V^\pi_n(s) - V^\pi_n(t)| \leq d^\pi_n(s,t)$.

The base case holds vacuously: $0 = V^\pi_0(s,t) \leq d^\pi_0(s,t) = 0$, so assume true up to $n$.

$$|V^\pi_{n+1}(s) - V^\pi_{n+1}(t)| = \left|R^\pi_s + \gamma \sum_{s' \in S} P^\pi_s(s') V^\pi_n(s') - \left(R^\pi_t + \gamma \sum_{s' \in S} P^\pi_t(s') V^\pi_n(s')\right)\right|$$

$$\leq |R^\pi_s - R^\pi_t| + |\gamma \sum_{s' \in S} V^\pi_n(s')(P^\pi_s(s') - P^\pi_t(s'))|$$

$$\leq |R^\pi_s - R^\pi_t| + |\gamma W(d^\pi_n)(P^\pi_s, P^\pi_t)|$$

$$= |R^\pi_s - R^\pi_t| + \gamma W(d^\pi_n)$$

$$= F^\pi(d^\pi_n)(s,t)$$

$$= d^\pi_{n+1}(s,t)$$

where the second inequality follows from noticing that, by induction, for all $s, t \in S$, $V^\pi_n(s) - V^\pi_n(t) \leq d_n(s,t)$, which means $V^\pi$ is a feasible solution to the primal LP objective of $W(d^\pi_n)(P^\pi_s, P^\pi_t)$ (see Equation 1).

**Lemma 1.** Given a deterministic MDP $M$, for any two states $s, t \in S$, action $a \in A$, and pseudometric $d \in \mathbb{M}$, $W_V(d)(P(s,a), P(t,a)) = d(M(s,a), M(t,a))$.

**Proof.** The primal LP defined in Equation 1 can be expressed in its dual form (which, incidentally, is a minimum-cost flow problem):

$$\min_{\lambda} \sum_{s',t' \in S} \lambda_{s',t'} d(s',t')$$

subject to

$$\forall s' \in S, \sum_{t'} \lambda_{s',t'} = P(s,a)(s')$$

$$\forall t' \in S, \sum_{s'} \lambda_{s',t'} = P(t,a)(t')$$

$$\lambda \geq 0$$
By the deterministic assumption it follows that $\lambda_{s',s} = 0$ whenever $s' \neq N(s,a)$ or $t' \neq N(t,a)$ (since otherwise one of the first two constraints will be violated). This means that only $\lambda_{N(s,a),N(t,a)}$ is positive. By the equality constraints it then follows that $\lambda_{N(s,a),N(t,a)} = 1$, resulting in $d(N(s,a),N(t,a))$ as the minimal objective value.

**Lemma 2.** (Maximizing action) If $d_0 \equiv 0$ and subsequent $d_n$ are updated as in Equation 3, then for all $s,t \in S$ and $\delta \in \mathbb{R}$ there exists an $n < \infty$ and $a \in A$ such that $\Pr(d_n(s,t) = |R(s,a) - R(t,a)| + \gamma d_{n-1}(N(s,a),N(t,a))) > 1 - \delta$.

**Proof.** Since we start at $d_0 \equiv 0$, the result will hold as long as we can guarantee that $\tau_{s,t,a}$ for some $a \in A$ will be sampled at least once. By assumption $D(\tau_{s,t,a}) > 0$, which means that the probability that $\tau_{s,t,a}$ is not sampled by step $n$ is $(1 - D(\tau_{s,t,a}))^n$. We obtain our result by taking $n > \ln(1 - D(\tau_{s,t,a}))$. 

**Lemma 3.** (Monotonicity) If $d_0 \equiv 0$ and subsequent $d_n$ are updated as in Equation 3, then $d_n \leq d_\sim$ for all $n \in \mathbb{N}$.

**Proof.** Obviously $d_{n-1} \leq d_n$ for all $n$. We will show by induction that $d_n \leq d_\sim$ for all $n$. The base case $d_0 \equiv 0 \leq d_\sim$ follows by definition, so assume true up to $n$ and consider any $s,t \in S$, where $n$ is large enough so that we have a high likelihood of sampling $\tau_{s,t,a}$ for some $a \in A$ (from Lemma 2). First note that this implies that for all $n' > n$ there exists an $a_{n'}^* \in A$ such that $d_{n'}(s,t) = |R(s,a_{n'}^*) - R(t,a_{n'}^*)| + \gamma d_{n'-1}(N(s,a_{n'}^*),N(t,a_{n'}^*))$.

Define $a_n^* = \arg \max \alpha_{n,a} ([|R(s,a_{n}^*) - R(t,a_{n}^*)| + \gamma d_{n-1}(N(s,a_{n}^*),N(t,a_{n}^*))])$. We then have:

$$d_\sim(s,t) = |R(s,a_\sim^*) - R(t,a_\sim^*)| + \gamma d_{\sim-1}(N(s,a_\sim^*),N(t,a_\sim^*))$$

where the second line follows from the fact that $a_\sim^*$ is the action that maximizes the bisimulation distance, and the third line follows from the inductive hypothesis.

**Theorem 4.** If $d_n$ is updated as in Equation 3 and $d_0 \equiv 0$, $\lim_{n \to \infty} d_n = d_\sim$ almost surely.

**Proof.** To prove this we will look at the difference $||d_\sim - d_n|| \equiv \max_{s,t \in S} |d_\sim(s,t) - d_n(s,t)|$. For any $\delta \in \mathbb{R}$ define $n^* = \max_{s,t \in S, a \in A} \left(\frac{\ln \delta}{\ln(1 - D(\tau_{s,t,a}))}\right)$. For some $n > n^*$,

Let $s,t$ be the state-pair that maximizes $||d_\sim - d_n||$ at time $n$. We have that the sequence $\{||d_\sim - d_n||\}$ is a contraction. By the Banach fixed-point theorem, the result follows.

We note that the last two results are related to Lemma 3 and Theorem 2 by (Comanici, Panangaden, and Precup 2012), but there are some important differences worth highlighting, notably in their use of the approximate update function $h_n$, which is not required in our method. Indeed, Lemma 3 in (Comanici, Panangaden, and Precup 2012) is more a statement on what’s required of $h_k$ to guarantee $h_k \leq d^*$ for all $k$; specifically, that in order for $h_k \leq d^*$, it is required that $h_k \leq d^*$ for all $k' < k$. In contrast, our Lemma 3 does not need this requirement as we make no use of an approximate update. Further, our proof of Lemma 3 relies on Lemma 2 to guarantee that we can rewrite $d_n$ as an $F$ update for a sufficiently large $n$. Note that this is rather different than the use of a similar idea with $\mathbf{1}$ by (Comanici, Panangaden, and Precup 2012), as they use it in their proof of Theorem 4, which lower-bounds $h_{km}$ with $F(m(0))$.

Although the statements of both Theorem 2 by (Comanici, Panangaden, and Precup 2012) and our Theorem 4 are similar (both methods converge to the true metric), the approach we take is quite different. (Comanici, Panangaden, and Precup 2012) construct their update via a partition of the state space into 3 sets $(A_k, B_k, E_k)$ at each step. The proper handling of these 3 sets, and in particular of $E_k$, makes the proof of Lemma 4 rather involved, which the authors require to obtain the lower-bound which is then used for the proof of Theorem 4. This extra complication is somewhat unfortunate, as the authors do not use $E_k$ sets in any
of their empirical evaluations, nor do they provide any indication as to what a good choice of the approximate update function \( \hat{h}_k \) would be.

In contrast, the proof of our Theorem 4 requires only Lemma 2 and Lemma 3. We believe our proof is much simpler to follow and makes use of fewer techniques.

**Mini-batch target and loss**

We specify a set of matrix operations for computing them on a *batch* of transitions. This allows us to efficiently train this approximant using specialized hardware like GPUs. The discussion in this section is specific to approximating \( \mathcal{d}_- \), but it is straightforward to adapt it to approximating \( \mathcal{d}^\circ \). We provide code for both approximants in the supplemental material with their implementation in TensorFlow (Abadi et al. 2015), as well as implementations of the algorithms discussed in the previous section.

At each step we assume access to a batch of \( b \) samples of states, actions, rewards, and next states:

\[
\begin{align*}
S &= \begin{bmatrix} \phi(s_1) \\ \vdots \\ \phi(s_b) \end{bmatrix}, \\
A &= \begin{bmatrix} a_1 \\ \vdots \\ a_b \end{bmatrix}, \\
R &= \begin{bmatrix} R(s_1, a_1) \\ \vdots \\ R(s_b, a_b) \end{bmatrix}, \\
N &= \begin{bmatrix} \phi(\mathcal{N}(s_1, a_1)) \\ \vdots \\ \phi(\mathcal{N}(s_b, a_b)) \end{bmatrix}
\end{align*}
\]

Letting \([X, Y]\) stand for the concatenation of two vectors \( X \) and \( Y \), from \( S \) we construct a new square matrix of dimension \( b \times b \) as follows:

\[
S^2 = \begin{bmatrix}
[\phi(s_1), \phi(s_1)], & [\phi(s_1), \phi(s_2)], & \cdots, & [\phi(s_1), \phi(s_b)] \\
[\phi(s_2), \phi(s_1)], & [\phi(s_2), \phi(s_2)], & \cdots, & [\phi(s_2), \phi(s_b)] \\
& \vdots & \ddots & \vdots \\
[\phi(s_b), \phi(s_1)], & [\phi(s_b), \phi(s_2)], & \cdots, & [\phi(s_b), \phi(s_b)]
\end{bmatrix}
\]

Each element in this matrix is a vector of dimension \( 2k \). We reshape this matrix to be a "single-column" tensor of length \( b^2 \). We can perform a similar exercise on the reward and next-state batches:

\[
R^2 = \begin{bmatrix}
|R(s_1, a_1) - R(s_1, a_1)| (= 0) \\
|R(s_1, a_1) - R(s_2, a_2)| \\
\vdots \\
|R(s_2, a_2) - R(s_1, a_1)| \\
\vdots \\
|R(s_b, a_b) - R(s_{b-1}, a_{b-1})| \\
|R(s_b, a_b) - R(s_b, a_b)| (= 0)
\end{bmatrix}
\]

\[
N^2 = \begin{bmatrix}
[\phi(\mathcal{N}(s_1, a_1)), \phi(\mathcal{N}(s_1, a_1))] \\
[\phi(\mathcal{N}(s_1, a_1)), \phi(\mathcal{N}(s_2, a_2))] \\
\vdots \\
[\phi(\mathcal{N}(s_b, a_b)), \phi(\mathcal{N}(s_{b-1}, a_{b-1}))] \\
[\phi(\mathcal{N}(s_b, a_b)), \phi(\mathcal{N}(s_b, a_b))]
\end{bmatrix}
\]

Finally, we define a mask which enforces that we only consider pairs of samples that have matching actions:

\[
W = \begin{bmatrix}
a_1 == a_1 \\
a_1 == a_2 \\
\vdots \\
a_b == a_{b-1} \\
a_b == a_b
\end{bmatrix}
\]

In batch-form, the target defined above becomes:

\[
T = (1 - I) \ast \max \left( R^2 + \gamma \beta \psi_{\theta^*} (N^2), \ \beta \psi_{\theta^*} (S^2) \right)
\] (4)

where \( \psi(X) \) indicates applying \( \psi \) to a matrix \( X \) elementwise. We multiply by \( (1 - I) \) to zero out the diagonals, since those represent approximations to \( d_-(s, s) \equiv 0 \). The parameter \( \beta \) is a stability parameter that begins at 0 and is incremented every \( C \) iterations. Its purpose is to gradually "grow" the effective horizon of the bisimulation backup and maximization. This is necessary since the approximant \( \psi \theta \) can have some variance initially, depending on how \( \theta \) is initialized. Further, (Jiang et al. 2015) demonstrate that using shorter horizons during planning can often be better than using the true horizon, especially when using a model estimated from data.

Finally, our loss \( L_i \) at iteration \( i \) is defined as \( L_i(\theta_i) = \mathbb{E}_D \left[ W \otimes (\psi_{\theta_i} (S^2) - T)^2 \right] \), where \( \otimes \) stands for the Hadamard product. Note that, in general, the approximant \( \psi \) is not a metric: it can violate the identity of indiscernibles, symmetry, and subadditivity conditions.
In Figure 7 we display the bisimulation distances from all states in the GridWorld MDP illustrated in Figure 4 in the main paper. These were computed using the sampling approach of section 5. Note how $d_\sim$ is able to capture similarities that go beyond simple physical proximity. This is most evident when examining the distance from the “hallway” state to all other states: even though it neighbours the bottom row in the top room, that row is furthest according to $d_\sim$. 

Figure 7: Bisimulation distances from all states.
### Configuration file for GridWorld

The following configuration file describes the hyperparameters used in subsection 7.1.

```python
# Configuration file for GridWorld

GridWorld.gamma = 0.99
GridWorld.batch_size = 256
GridWorld.representation_dimension = 729
GridWorld.num_iterations = 240000
GridWorld.target_update_period = 500
GridWorld.starting_learning_rate = 0.01
GridWorld.use_decayed_learning_rate = False
GridWorld.learning_rate_decay = 0.9
GridWorld.epsilon = 1e-8
GridWorld.staircase = False
GridWorld.add_noise = True
GridWorld.bisim_horizon_discount = 0.9
GridWorld.double_period_halfway = False
GridWorld.debug = False
```

### Aggregating states

In Figure 8 we explore aggregating a set of states sampled from the continuous variant of the 31-state MDP. We sampled 100 independent samples for each underlying cell, computed the distances between each pair of sampled states, and then aggregated them by incrementally growing “clusters” of states while ensuring that all states in a cluster are within a certain distance of each other. As can be seen, our learned distance is able to capture many of the symmetries present in the environment: the orange cluster tends to gather near-goal states, the dark-brown and dark-blue clusters seems to gather states further away from goals, while the bright red states properly capture the unique “hallway” cell. This experiment highlights the potential for successfully approximating bisimulation metrics in continuous state MDPs, which can render continuous environments more manageable.

![Aggregating samples](image)

**Figure 8**: Aggregating samples drawn from a continuous MDP using the learned bisimulation metric approximant.

The code for the clustering is displayed below:
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap, BoundaryNorm
import numpy as np

clusters = []
threshold = 0.86
for i in range(len(normalized_distances)):
    present = False
    include = False
    for c in clusters:
        if np.any(np.isin(i, c)):
            present = True
            break
        present = True
        for j in c[0]:
            if normalized_distances[j, i] >= threshold:
                present = False
                break
        if present:
            np.append(c, i)
            break
    if present:
        continue
    indices = np.nonzero(normalized_distances[i] < threshold)
    clusters.append(indices)

labels = np.zeros(len(normalized_distances))
label = 0
for c in clusters:
    for i in c[0]:
        labels[i] = label
    label += 1
cmap = plt.cm.jet
cmaplist = [cmap(i) for i in range(cmap.N)]
cmap = cmap.from_list('Custom cmap', cmaplist, cmap.N)
bounds = np.linspace(0, label, label+1)
norm = BoundaryNorm(bounds, cmap.N)
fig, ax = plt.subplots(figsize=(16, 16))
ax.patch.set_facecolor('white')
ax.grid(color='black')
plt.scatter(sampled_distances['samples'][0], sampled_distances['samples'][1], c=labels, cmap=cmap, norm=norm)
plt.xlim(0, 7)
plt.ylim(0, 9)
plt.colorbar()
Distance plots for all games

For each game we display four panels:

- **Bottom left:** Source frame (state $s$)
- **Bottom right:** Closest frame so far
- **Top left:** Current frame (state $t$)
- **Top right:** Plot of distances from source frame to every other frame ($\psi_\pi([\phi(s), \phi(t)])$, in black) and the difference in value function, according to the trained Rainbow network ($|\hat{V}_\pi(\phi(s)) - \hat{V}_\pi(\phi(t))|$, in blue)

To pick the hyperparameters, we did a sweep over the following values, performing 3 independent runs for each setting. We picked a setting which gave the best overall performance across all games (final values specified in the `rainbow.gin` file provided with the source code, except for Pong where we used a learning rate of 0.001, as specified in the paper):

- **Learning rates:** $[0.00005, 0.000075, 0.0001, 0.00025, 0.0005, 0.00075, 0.001]$
- **Batch size:** $[4, 8, 16, 32, 64, 128]$
- **Target update period:** $[10, 50, 100, 250, 500, 1000, 2000]$
- **Number of hidden layers:** $[1, 2, 3]$
- **Number of hidden units per layer:** $[16, 32, 64, 128]$

![Figure 9: Space Invaders: evaluation at frame 500.](image)
Training curves for $d_{\pi}^\infty$

We provide the training curves for the bisimulation metric $d_{\pi}^\infty$ learned on the trained reinforcement learning agents.

Lax bisimulation metrics

For completeness, in this section we include the main definitions and theoretical results of lax-bisimulation metrics introduced in (Taylor, Precup, and Panagaden 2009), modified to match the notation used in this paper.

Definition 5. A relation $E \subseteq S \times S$ is a lax (probabilistic) bisimulation relation if whenever $(s, t) \in E$ we have that:
1. $\forall a \in A \exists b \in A$ such that $R(s, a) = R(t, b)$
2. $\forall a \in A \exists b \in A$. $\forall c \in S_E. P(s, a)(c) = P(t, b)(c)$,

where $P(x, y)(c) = \sum_{z \in c} P(x, y)(z)$.

The lax bisimulation $\sim_{lax}$ is the union of all lax bisimulation relations.

Definition 6. Given a 1-bounded pseudometric $d \in M$, the metric $\delta(d) : S \times A \to [0, 1]$ is defined as follows:

\[
\delta(d)((s, a), (t, b)) = |R(s, a) - R(t, b)| + \gamma W(d)(P(s, a), P(t, b))
\]

Definition 7. Given a finite 1-bounded metric space $(\mathfrak{M}, d)$ let $C(\mathfrak{M})$ be the set of compact spaces (e.g. closed and bounded in $\mathbb{R}$). The Hausdorff metric $H(d) : C(\mathfrak{M}) \times C(\mathfrak{M}) \to [0, 1]$ is defined as:

\[
H(d)(X, Y) = \max \left( \sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y) \right)
\]

Definition 8. Denote $X_a = \{(s, a)|a \in A\}$. We define the operator $F : M \to M$ as:

\[
F(d)(s, t) = H(\delta(d))(X_a, X_t)
\]

Theorem 5. $F$ is monotonic and has a least fixed point $d_{lax}$ in which $d_{lax}(s, t) = 0$ iff $s \sim_{lax} t$.

Theorem 6. $|V^*(s) - V^*(t)| \leq d_{lax}(s, t) \leq d_{\sim}(s, t)$. 

Figure 10: Pong: evaluation at frame 500, peaks are when the agent scores.
Figure 11: Asterix: evaluation at frame 500. Plateaus are when the agent is not moving.

Figure 12: Training curves for Space Invaders.

Figure 13: Training curves for Asterix.
Figure 14: Training curves for Pong.