A PROOFS FOR SECTION 4

Theorem 1. *If Φ^* is a feature matrix minimizing $\mathcal{L}_{MCSM}(\Phi)$, then its column space spans the top d left singular vectors of the (infinite-dimensional) successor measure matrix Ψ with respect to the inner product $(x, y)_\Xi = y^\top \Xi x$, for all $x, y \in \mathbb{R}^n$. Here, $\Xi \in \mathbb{R}^{\mathcal{X}\times\mathcal{X}}$ is the diagonal matrix whose entries are given by ξ.*

Proof. We consider the SVD of the successor measure ψ with respect to the weighted inner product Ξ. In matrix form, we write

$$\Psi = F\Sigma B^\mathsf{T}$$

where $F \in \mathbb{R}^{n\times d}, \Sigma \in \mathbb{R}^{d\times d}$ and $B \in \mathbb{R}^{n\times d}$ satisfy

$$F^\mathsf{T}F = I, B^\mathsf{T}\Xi B = I, \Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_d)$$

and σ_i are the singular values of Ψ sorted in decreasing order.

$$\begin{aligned}
\arg\min_{\Phi\in\mathbb{R}^{n\times d}} \mathcal{L}_{MCSM}(\Phi) &= \arg\min_{\Phi\in\mathbb{R}^{n\times d}} \min_{w_{S,a}\in\mathbb{R}^d} \mathop{\mathbb{E}}_{S\sim\xi}\Big[\Big(\sum_{x\in\mathcal{X},a\in\mathcal{A}} (\phi(x)^\top w_{S,a} - \psi(x,a,S)\Big)^2\Big] \\
&= \arg\min_{\Phi\in\mathbb{R}^{n\times d}} \min_{W} \|(\Phi W - \Psi)\|_\Xi^2 \\
&= \arg\min_{\Phi\in\mathbb{R}^{n\times d}} \|\Pi_\Phi^\perp \Psi\|_\Xi^2
\end{aligned}$$

where Π_Φ is the orthogonal projection onto $\mathrm{span}(\Phi)$. The above is equivalent to saying that Φ must span the top d singular vectors of Ψ. □

B IMPLEMENTATION DETAILS

B.1 UNIVERSAL HASH FUNCTIONS

We define the set of multiply-shift universal hash functions (Carter & Wegman, 1979) as:

$$h_i(x) = \left(a_0^{(i)} + \sum_{j=1}^{n} a_j^{(i)} \cdot x_j \bmod p\right) \bmod m\,,$$

where $x \in \mathbb{R}^n$ is a flattened vector of the environment's observation, $a^{(i)} \in \mathbb{R}^n$ is a randomly initialized vector that that parameterizes the hash function, p is a prime, which in our case is the Mersenne prime $p = 2^{13} - 1$, and m allows us to control the activation proportion of the indicator function. We can now define the indicator function as follows:

$$f_i(x) = \mathbf{1}\{h_i(x) = 0\}\,.$$

B.2 QUANTILE REGRESSION

We use quantile regression to tune the proportion of states that trigger our random network indicator functions. To do so, we use a tunable bias that we update with gradient descent. First, recall that the random network indicators are computed using a random neural network $f : x \mapsto \mathbb{R}$. If we naively apply the SIGN function to the network output, the proportion of states that map to 1 is unlikely to match the target proportion p. Therefore, we first add a bias term to the output $r' = f(x) + b$, and then tune the bias to minimize the quantile regression loss

$$\mathcal{L}_{\text{QR}}(b) = \mathbb{E}_{x\in\mathcal{X}} r'(x) \cdot ((1-p) - \textsc{Sign}(r'(x))) \tag{2}$$

Once the bias has been tuned, the output of the random network indicator is $r = \textsc{sign}(f(x) + b)$.

B.3 ALGORITHM

Algorithm 1 gives pseudo-code for the method as implemented with a fixed replay memory.

Algorithm 1 Proto-Value Networks

Require: Transition dataset $\mathcal{D}$, Function approximator $\hat{\Psi}_\theta : \mathcal{X} \to \mathbb{R}^{m \times |\mathcal{A}|}$, m RNI networks $f_i : \mathcal{X} \to \mathbb{R}$, m RNI threshold bias vectors b_i, Polyak coefficient τ, reward proportion p

for step $= 1, \ldots$ **do**
  Sample mini-batch of n transitions $\{(x, a, x')\}_{i=1}^{n} \subset \mathcal{D}$

  # Calculate random network indicators
  $r'_j(x) \leftarrow f_j(x) + b_j \quad \forall j = 1, \ldots, m$
  $r_j(x) \leftarrow \textsc{Sign}(r'_j(x))$

  # Calculate PVN loss
  $\mathcal{L}_{\text{PVN}}(\theta) \leftarrow \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} \left(r_j(x_i) + \gamma \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \hat{\Psi}_\theta^{(j)}(x'_i, a') - \hat{\Psi}_{\theta^-}^{(j)}(x_i, a_i) \right)^2$

  # Calculate quantile regression loss
  $\mathcal{L}_{\text{QR}}(b_j) \leftarrow \frac{1}{n} \sum_{i=1}^{n} r'_j(x) \cdot ((1-p) - \textsc{Sign}(r'_j(x)))$

  # Perform gradient step
  Update $\theta \leftarrow \theta - \eta_1 \frac{\partial}{\partial \theta} \mathcal{L}_{\text{PVN}}(\theta)$
  Update $b_j \leftarrow b_j - \eta_2 \frac{d}{db_j} \mathcal{L}_{\text{QR}}(b_j) \quad \forall\, j = 1, \ldots, m$

  # Polyak average target network parameters
  $\theta^- \leftarrow \tau \theta^- + (1 - \tau)\, \theta$
end for

B.4 HYPERPAREMETERS

In the tables below we report all relevant hyperparameter choices for both our offline pre-training phase, and online learning phase.

We selected most of our hyperparameters based on best practices from previous work. We chose p based on the estimated reward proportion from actual Atari games. We tuned our online hyperparameters using 5 tuning games, ASTERIX, BEAM RIDER, PONG, QBERT, and SPACE INVADERS.

Table 1: Offline Hyperparameters

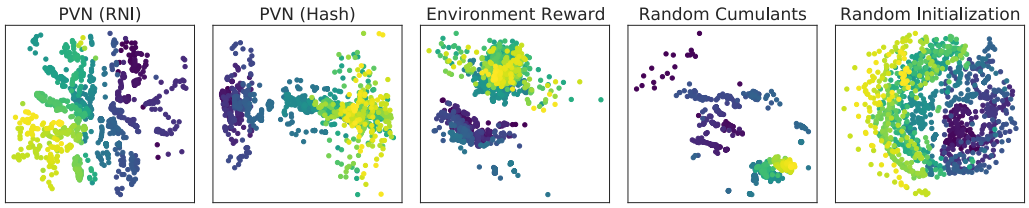
Hyperparameter	Value
Number of auxiliary tasks	100
Batch size	256
Number of gradient steps	1,562,500
Discount factor γ	0.99
Polyak average coefficient τ	0.99
Reward proportion p	0.01
Quantile regression burn-in steps	62,500
Tasks per module	10
Optimizer	Adam
Learning rate	1e-4
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	1.5e-4

Table 2: Online Hyperparameters

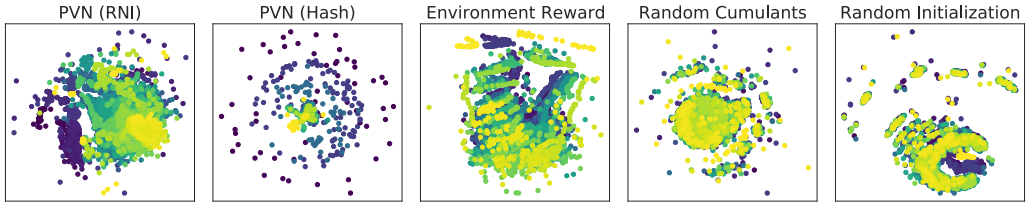
Hyperparameter	Value
Update rule	DQN
Num. layers	1 (linear)
Num. agent steps	3.75M
Train ϵ	0.01
Evaluation ϵ	0.001
n step	1
Discount factor γ	0.99
Optimizer	Adam
Learning rate	6.25e-5
Adam ϵ	1.5e-4
Maximum gradient norm	10
Batch size	32
Minimum replay size	2,000
Maximum replay size	1,000,000
Gradient steps per env step	1

C MDS PLOTS

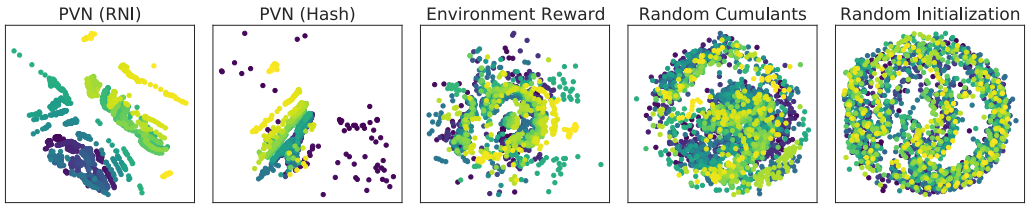
Below are a selection of MDS plots for the methods discussed in the paper for each of the 5 tuning games. These plots are generated using the representations learned during the pre-training phase, and one expert trajectory is presented in each plot. Darker points correspond to states at the beginning of the trajectory, and lighter points correspond to states at the end of the trajectory. These plots demonstrate that the representations learned by each method are clearly different, and therefore have different properties.



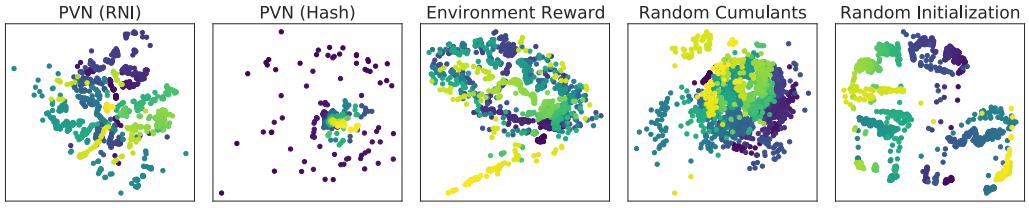
(a) Asterix



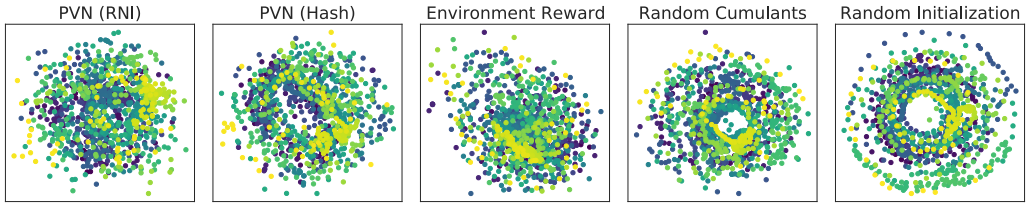
(b) BeamRider



(c) Pong



(d) Qbert



(e) SpaceInvaders

Figure 5: MDS plot for a single trajectory.

D PER-GAME RESULTS

Below, we report the per-game results for the methods discussed in the paper. In addition, we include DQN and the Environment Reward method, which trains an encoder using the environment reward during the pre-training phase, and then uses the fixed representation to train a linear head in the same manner as the compared methods. Note that Environment Reward acts as a kind of oracle in our setting – it is the only method that has access to the environment reward during the pre-training phase. The results reported here use 1 offline seed and 3 online seeds, and evaluation scores (averaged over 100 evaluation runs) are reported after 3.75M agent steps. DQN's performance is reported after 50M agent steps.

Table 3: Per-Game Results on RLU Games

	Random Init	DQN	Env Reward	RCs	PVN (Hash)	PVN (RNI + Opt. Policy)	PVN (RNI)
Alien	404	2484	2634	223	804	67	1034
Amidar	44	1208	214	74	96	4	96
Assault	140	1525	792	617	433	1464	1077
Asterix	523	2711	17412	266	17293	413	17067
Atlantis	14270	853640	399021	21981	5302	14485	12318
BankHeist	2	602	431	937	5	6	193
BattleZone	1683	17785	46471	5164	5516	12937	13313
BeamRider	384	5852	11325	3933	1550	12402	7744
Boxing	-37	78	99	-5	-3	-47	94
Breakout	7	96	275	207	150	252	8
Carnival	362	4785	4702	977	1144	665	1054
Centipede	3534	2583	1559	1067	2535	2422	3186
ChopperCommand	474	2691	2631	882	649	960	3238
CrazyClimber	4	104569	117285	136788	3861	1	25509
DemonAttack	119	6362	121236	2466	2168	40692	61531
DoubleDunk	-18	-7	-15	-20	-14	-15	-20
Enduro	0	629	1391	0	16	2	405
FishingDerby	-98	1	37	-99	-95	-96	-77
Freeway	7	26	34	28	18	31	24
Frostbite	28	367	2826	102	204	29	643
Gopher	208	5480	3554	3299	467	105	4384
Gravitar	43	330	1217	26	2	64	97
Hero	23	17325	10148	896	2904	2	1952
IceHockey	-13	-6	21	-14	-14	-5	-7
Jamesbond	24	573	1085	198	9	263	398
Kangaroo	199	11486	10826	1671	394	819	2007
Krull	1188	6098	9193	79	827	2561	4551
KungFuMaster	2807	23435	32442	14496	2269	30	16761
MsPacman	836	3402	5460	1255	1018	475	1523
NameThisGame	848	7279	16307	2977	2052	8568	6178
Phoenix	415	4997	18791	12817	5007	5896	8969
Pong	-21	17	21	15	-9	19	19
Pooyan	748	3212	1826	1844	1244	1383	1688
Qbert	172	10118	13679	3202	1807	9748	5226
Riverraid	1221	11639	11623	316	2430	319	8348
RoadRunner	2543	36925	49522	332	4609	531	13327
Robotank	2	60	76	18	10	5	4
Seaquest	44	1601	260	75	122	-	951
SpaceInvaders	131	1794	23138	1148	1345	-	908
StarGunner	643	42165	844	719	899	801	9670
TimePilot	2244	3654	13231	3956	4006	1375	1447
UpNDown	1174	8488	25181	12142	7089	11564	9136
VideoPinball	3877	63406	280340	3314	21555	6372	12319
WizardOfWor	465	2066	5079	1289	435	275	1399
YarsRevenge	3089	23909	49298	2360	6591	2068	23687