facebook
Artificial Intelligence Research

Exploration-Exploitation in Reinforcement Learning

Part 3 – Scaling up Exploration to DeepRL

Mohammad Ghavamzadeh, Alessandro Lazaric and Matteo Pirotta
Facebook AI Research

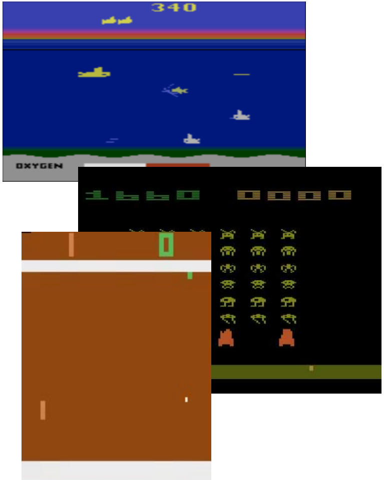
Outline

Website

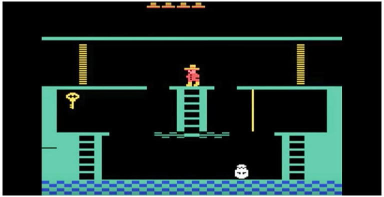
https://rlgammazero.github.io

Exploration in DeepRL

these are easy



this is hard, *almost* impossible

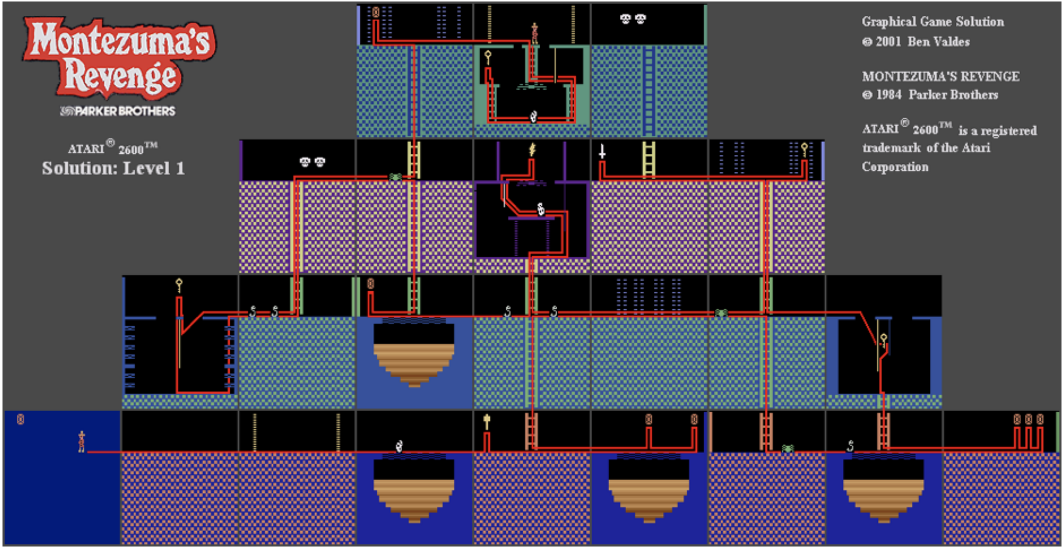


Why?

Random exploration sometimes work!
PONG GIF

Montezuma with random actions!
Link

Montezuma's Revenge: Level 1



The Four Ingredients Recipe

1. Build accurate estimators
2. Evaluate the uncertainty of the prediction
3. Define a mechanism to combine estimation and uncertainty
4. Execute the best policy

The Four Ingredients Recipe

Optimism in face of uncertainty

1. Build accurate estimators

$$\widehat{M}_k \Rightarrow V^{\pi}_{\widehat{M}_k}$$

2. Evaluate the uncertainty of the estimators

$$B^r_{hk}(s,a) := \Big[\widehat{r}_{hk}(s,a) - \beta^r_{hk}(s,a),\ \widehat{r}_{hk}(s,a) + \beta^r_{hk}(s,a)\Big]$$
$$B^p_{hk}(s,a) := \Big\{p(\cdot|s,a) \in \Delta(\mathcal{S}) :\ \|p(\cdot|s,a) - \widehat{p}_{hk}(\cdot|s,a)\|_1 \leq \beta^p_{hk}(s,a)\Big\}$$

3. Define a mechanism to combine estimation and uncertainty

$$\pi_k = \arg\max_{\pi} \max_{M \in \mathcal{M}_k} V^{\pi}_{M}$$

The Four Ingredients Recipe

Posterior Sampling

1. Build accurate estimators
2. Evaluate the uncertainty of the estimators

$$\forall \Theta, \quad \mathbb{P}(M^\star \in \Theta | H_t, \mu_1) = \mu_t(\Theta) \qquad \mu_t \text{ updated using Bayes' rule}$$

3. Define a mechanism to combine estimation and uncertainty

$$\pi_k = \arg\max_\pi V^\pi_{\widetilde{M}_k}, \qquad \widetilde{M}_k \sim \mu_k$$

"Practical" Limitations

Optimism in face of uncertainty

- Confidence intervals

$$\beta_t^r(s,a) \propto \sqrt{\frac{\log(N_t(s,a)/\delta)}{N_t(s,a)}} \qquad \beta_t^p(s,a) \propto \sqrt{\frac{S\log(N_t(s,a)/\delta)}{N_t(s,a)}}$$

- Solving

$$\pi_t = \arg\max_{\pi} \max_{M\in\mathcal{M}_t} V_M^{\pi}$$

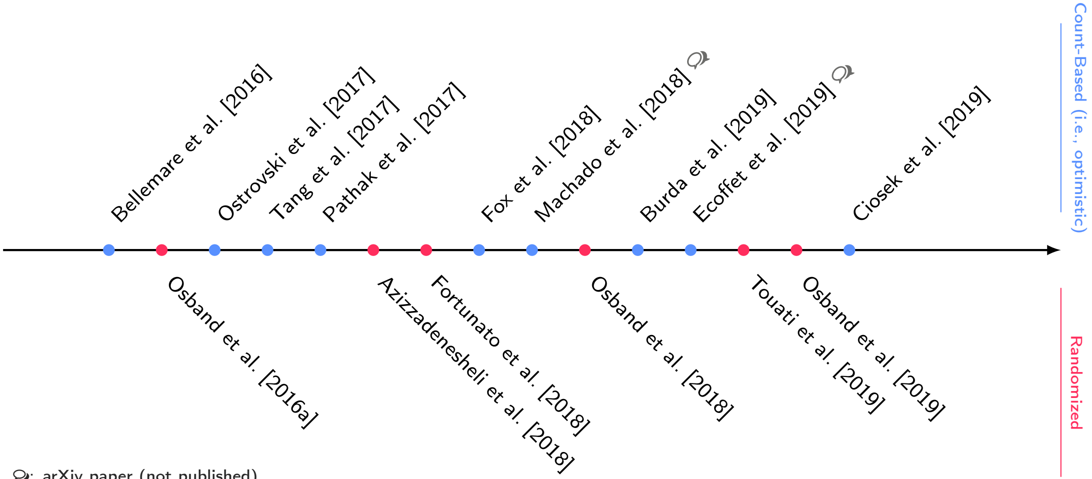
Posterior sampling

- Posterior (dynamics for any state-action pair)

$$\text{Dirichlet}\big(N_t(s_1'|s,a), N_t(s_2'|s,a), \ldots, N_t(s_S'|s,a)\big)$$

- Update/sample from a unstructured/non-conjugate posteriors

History: Exploration in DeepRL



: arXiv paper (not published)

Outline

Count-based Exploration

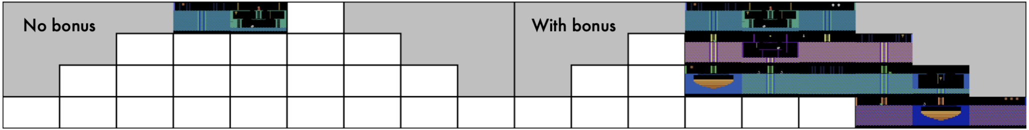
General Scheme

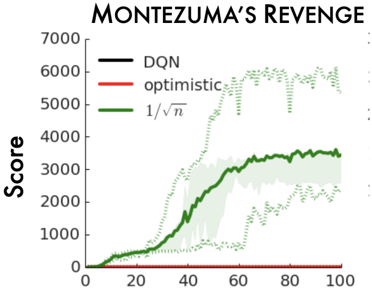
1. Estimate a “proxy” for the *number of visits* $\widetilde{N}(s_t)$
2. Add an *exploration bonus* to the rewards

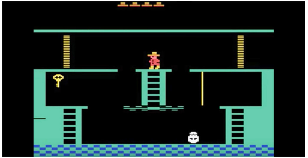
$$\widetilde{r}_t^{+} = r_t + c\sqrt{\frac{1}{\widetilde{N}(s_t)}}$$

3. Run *any DeepRL* algorithm on $\mathcal{D}_t = \left\{(s_i, a_i, \widetilde{r}_i^{+}, s_{i+1})\right\}$

Does it work?

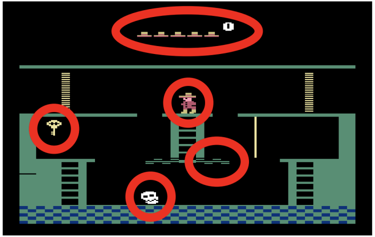






* figures from [Bellemare et al., 2016]

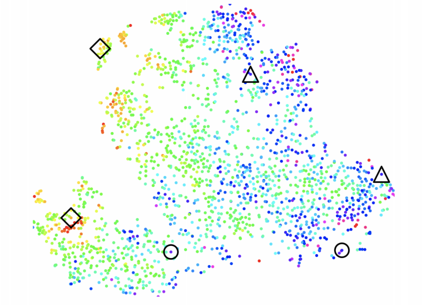
What to Count?



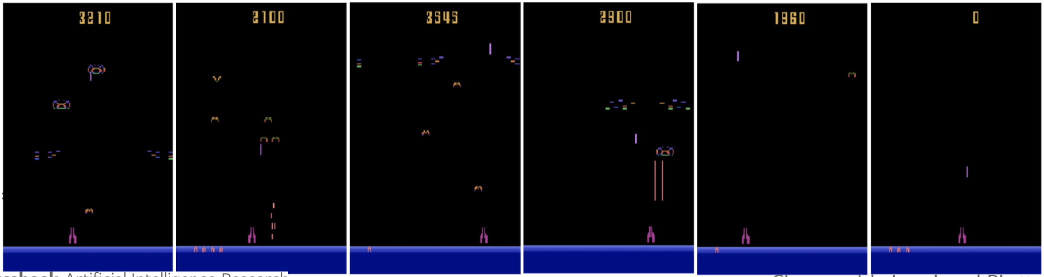
We never see the same state twice (or it is very unlikely)!

How difficult is to learn a state representation?

[Sun et al., 2019]

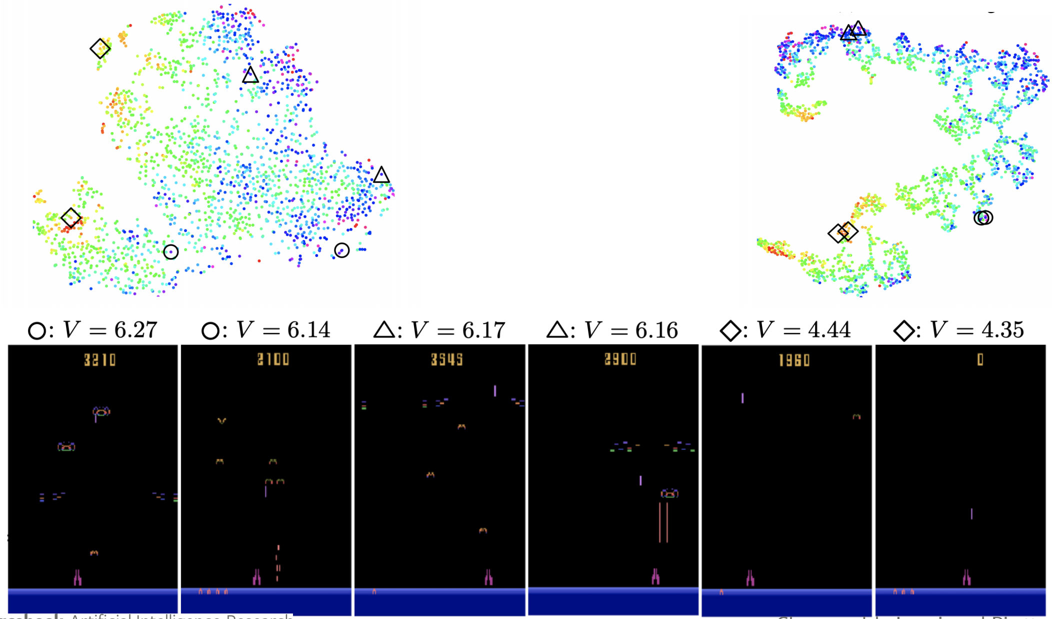


○: $V = 6.27$ ○: $V = 6.14$ △: $V = 6.17$ △: $V = 6.16$ ◇: $V = 4.44$ ◇: $V = 4.35$



How difficult is to learn a state representation?

[Sun et al., 2019]



Count-based Exploration

[Tang et al., 2017]

Algorithm 1: Count-based exploration through static hashing, using SimHash

1. Define state preprocessor $g : \mathcal{S} \rightarrow \mathbb{R}^D$
2. (In case of SimHash) Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from the standard Gaussian distribution $\mathcal{N}(0, 1)$
3. Initialize a hash table with values $n(\cdot) \equiv 0$
4. **for** each iteration j **do**
5. Collect a set of state-action samples $\{(s_m, a_m)\}_{m=0}^{M}$ with policy π
6. Compute hash codes through any LSH method, e.g., for SimHash, $\phi(s_m) = \text{sgn}(Ag(s_m))$
7. Update the hash table counts $\forall m : 0 \leq m \leq M$ as $n(\phi(s_m)) \leftarrow n(\phi(s_m)) + 1$
8. Update the policy π using rewards $\left\{ r(s_m, a_m) + \frac{\beta}{\sqrt{n(\phi(s_m))}} \right\}_{m=0}^{M}$ with any RL algorithm

- Use *locality-sensitive hashing* to discretize the input
  - Encode the state into a k-dim vector by random project
    small k = more hash collisions
  - Use the sign to discretize
    small $\phi(s) \in \{-1, 1\}^k$
- *Count on discrete hashed-states*

Count-based Exploration

[Tang et al., 2017]

Algorithm 1: Count-based exploration through static hashing, using SimHash

Define state preprocessor $g : \mathcal{S} \to \mathbb{R}^D$
(In case of SimHash) Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from the standard Gaussian distribution $\mathcal{N}(0, 1)$
Initialize a hash table with values $n(\cdot) \equiv 0$
for each iteration j **do**
  Collect a set of state-action samples $\{(s_m, a_m)\}_{m=0}^{M}$ with policy π
  Compute hash codes through any LSH method, e.g., for SimHash, $\phi(s_m) = \mathrm{sgn}(Ag(s_m))$
  Update the hash table counts $\forall m : 0 \leq m \leq M$ as $n(\phi(s_m)) \leftarrow n(\phi(s_m)) + 1$
  Update the policy π using rewards $\left\{ r(s_m, a_m) + \frac{\beta}{\sqrt{n(\phi(s_m))}} \right\}_{m=0}^{M}$ with any RL algorithm

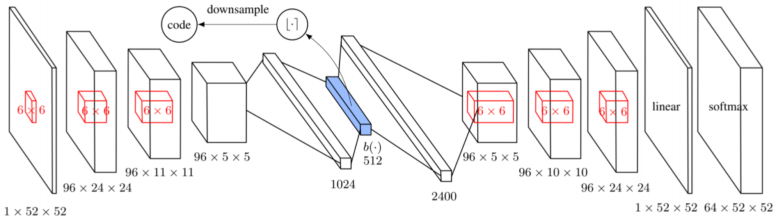
- Use *locality-sensitive hashing* to discretize the input
  - Encode the state into a k-dim vector by random project
    small k = more hash collisions
  - Use the sign to discretize
    small $\phi(s) \in \{-1, 1\}^k$
- *Count on discrete hashed-states*

Difficult to define a good hashing function

Count-based Exploration

[Tang et al., 2017]

Improve counts by learning a compression



$$L\left(\{s_n\}_{n=1}^{N}\right) = -\frac{1}{N}\sum_{n=1}^{N}\left[\log p(s_n) - \frac{\lambda}{K}\sum_{i=1}^{D}\min\left\{(1-b_i(s_n))^2, b_i(s_n)^2\right\}\right]$$

- Entropy loss for the auto-encoder
- "Binarization" loss for the "projection"

Count-based Exploration

[Tang et al., 2017]

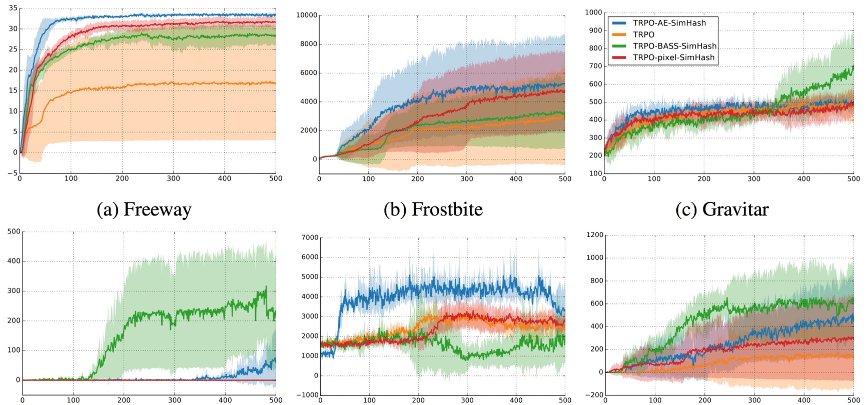
Algorithm 2: Count-based exploration using learned hash codes

Define state preprocessor $g : \mathcal{S} \to \{0,1\}^D$ as the binary code resulting from the autoencoder (AE)
Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from the standard Gaussian distribution $\mathcal{N}(0,1)$
Initialize a hash table with values $n(\cdot) \equiv 0$
for each iteration j **do**
    Collect a set of state-action samples $\{(s_m, a_m)\}_{m=0}^M$ with policy π
    Add the state samples $\{s_m\}_{m=0}^M$ to a FIFO replay pool $\mathcal{R}$
    if j mod $j_{\text{update}} = 0$ **then**
        Update the AE loss function in Eq. (3) using samples drawn from the replay pool $\{s_n\}_{n=1}^N \sim \mathcal{R}$, for example using stochastic gradient descent
    Compute $g(s_m) = \lfloor b(s_m) \rceil$, the D-dim rounded hash code for s_m learned by the AE
    Project $g(s_m)$ to a lower dimension k via SimHash as $\phi(s_m) = \operatorname{sgn}(Ag(s_m))$
    Update the hash table counts $\forall m : 0 \le m \le M$ as $n(\phi(s_m)) \leftarrow n(\phi(s_m)) + 1$
    Update the policy π using rewards $\left\{ r(s_m, a_m) + \frac{\beta}{\sqrt{n(\phi(s_m))}} \right\}_{m=0}^M$ with any RL algorithm

- Use all past history to update the AE
- AE should not be updated too often
  we need stable codes!

Count-based Exploration

[Tang et al., 2017]



(a) Freeway (b) Frostbite (c) Gravitar

(d) Montezuma's Revenge (e) Solaris (f) Venture

Count-based Exploration

[Bellemare et al., 2016, Ostrovski et al., 2017]

- Density estimation over a countable set $\mathcal{X}$

$$\rho_n(x) = \rho(x|x_1,\ldots,x_n) \approx \mathbb{P}\big[X_{n+1} = x|x_1,\ldots,x_n\big]$$

- Recording probability

$$\rho'_n(x) = \rho(x|x_1,\ldots,x_n,x) \approx \mathbb{P}\big[X_{n+2} = x|x_1,\ldots,x_n,X_{n+1} = x\big]$$

probability of x after observing a new occurrency of x

- Pseudo "local" and "total" counts $\widetilde{N}_n(x)$ and $\widetilde{N}_n(x)$ s.t.

$$\frac{\widetilde{N}_n(x)}{\widetilde{n}} = \rho_n(x); \qquad \frac{\widetilde{N}_n(x)+1}{\widetilde{n}+1} = \rho'_n(x) \Rightarrow \widetilde{N}_n(x) = \frac{\rho_n(x)(1-\rho'_n(x))}{\rho'_n(x)-\rho_n(x)} = \widetilde{n}\rho_n(x)$$

Count-based Exploration

[Bellemare et al., 2016, Ostrovski et al., 2017]

- Density estimation over a countable set $\mathcal{X}$

$$\rho_n(x) = \rho(x|x_1, \ldots, x_n) \approx \mathbb{P}\big[X_{n+1} = x|x_1, \ldots, x_n\big]$$

- Recording probability

$$\rho'_n(x) = \rho(x|x_1, \ldots, x_n, x) \approx \ \mathbb{P}\big[X_{n+2} = x|x_1, \ldots, x_n, X_{n+1} = x\big]$$

probability of x after observing a new occurrency of x

- Pseudo "local" and "total" counts $\widetilde{N}_n(x)$ and $\widetilde{N}_n(x)$ s.t.

$$\frac{\widetilde{N}_n(x)}{\widetilde{n}} = \rho_n(x); \qquad \frac{\widetilde{N}_n(x)+1}{\widetilde{n}+1} = \rho'_n(x) \ \Rightarrow \widetilde{N}_n(x) = \frac{\rho_n(x)(1-\rho'_n(x))}{\rho'_n(x)-\rho_n(x)} = \widetilde{n}\rho_n(x)$$

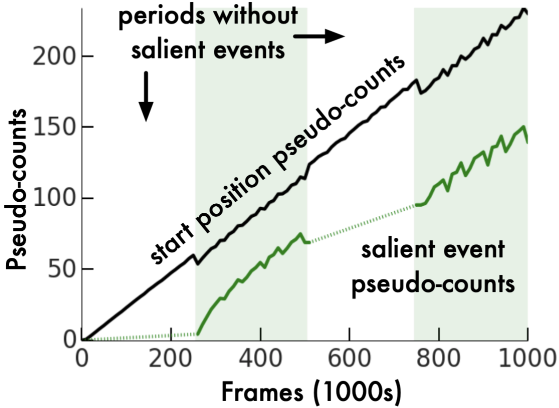
👍 Any density estimation algorithm (accurate for images)

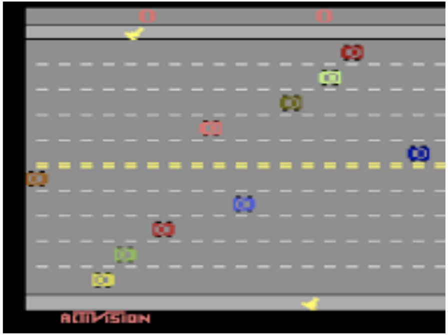
e.g., CTS [Bellemare et al., 2014] or PixelCNN [van den Oord et al., 2016]

👎 Density estimation in continuous spaces is hard

Count-based Exploration

[Bellemare et al., 2016, Ostrovski et al., 2017]





Count-based Exploration

Bellemare et al. [2016], Ostrovski et al. [2017]

Montezuma!

Prediction-based Exploration

Burda et al. [2019]

What we need is to know how accurate are our predictions.

uncertainty about the model parameters

proxy: prediction error

Sources of prediction errors

1. Amount of data 👍
2. Stochasticity (e.g., noisy-TV) 👎
3. Model misspecification 👎
4. Learning dynamics 👎

Prediction-based Exploration

[Burda et al., 2019]

- Randomly initialize two instances of the same NN (target θ_* and prediction θ_0)

$$f_{\theta_*} : \mathcal{S} \to \mathbb{R}; \qquad f_\theta : \mathcal{S} \to \mathbb{R}$$

- Train the prediction network minimizing loss w.r.t. the target network

$$\theta_n = \arg\min_\theta \sum_{t=1}^{n} \Big(f_\theta(s_t) - f_{\theta_*}(s_t) \Big)^2$$

- Build "intrinsic" reward

$$r_t^I = \Big| f_\theta(s_t) - f_{\theta_*}(s_t) \Big|$$

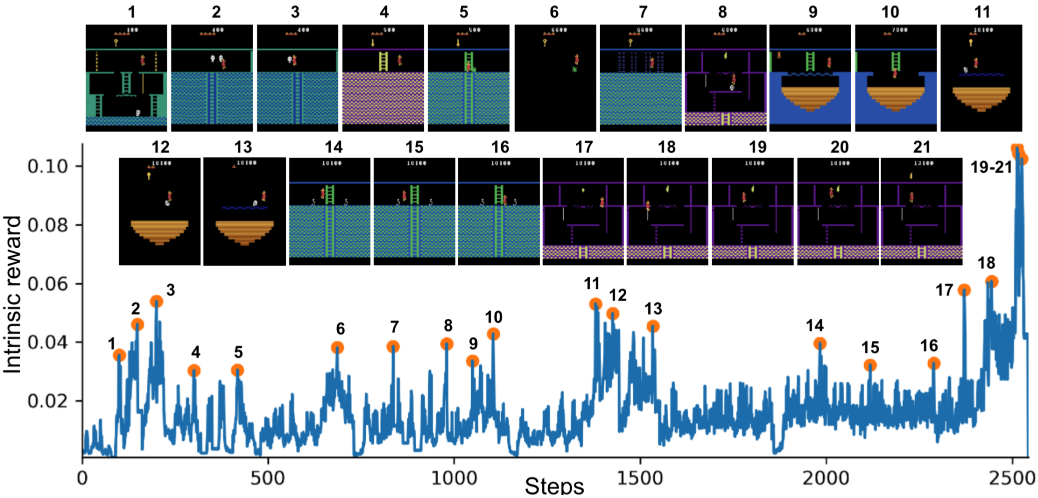
👍 No influence from stochastic transitions

👍 No model misspecification (f_θ can exactly predict f_{θ_*})

👍 Influence of learning dynamics can be reduced

Prediction-based Exploration

Burda et al. [2019]



Prediction-based Exploration

[Burda et al., 2019]

General architecture

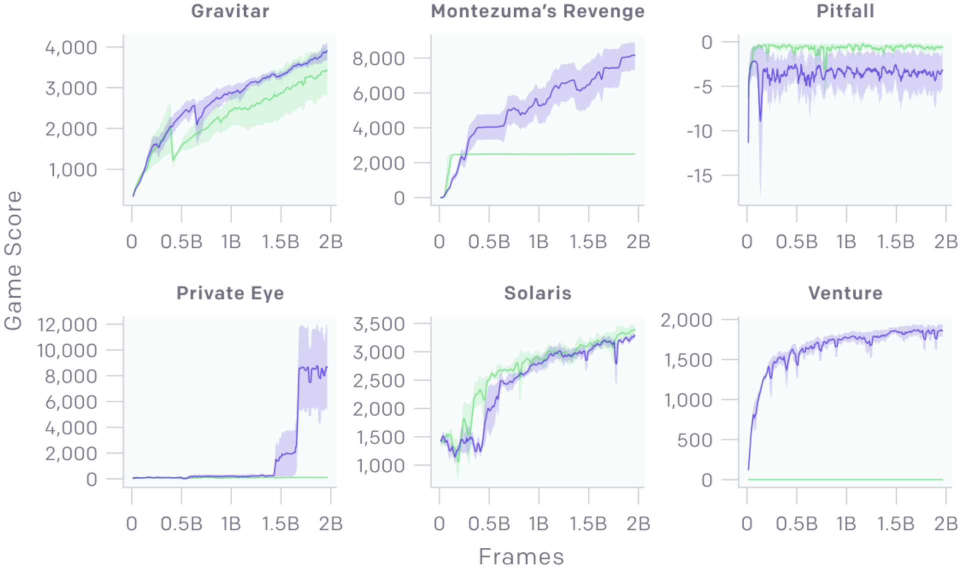
- Separate extrinsic r_t^E and intrinsic reward r_t^I
- PPO with two heads to estimate V^I and V^E
- Greedy policy w.r.t. $V^I + cV^E$

"Tricks"

- Rewards should be in the same range
- Use different discount factors for intrinsic and extrinsic rewards

Prediction-based Exploration

Burda et al. [2019]



Prediction-based Exploration

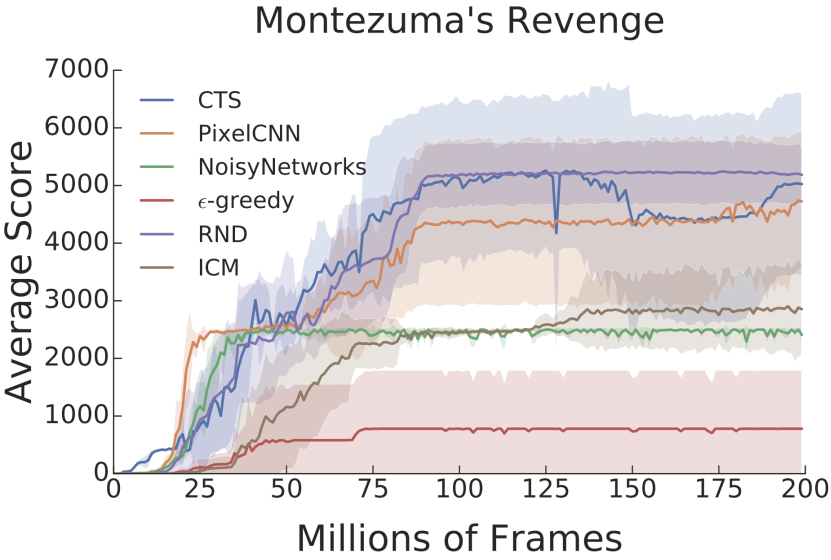
[Burda et al., 2019]

Montezuma!

finds 22 out of the 24 rooms on the first level

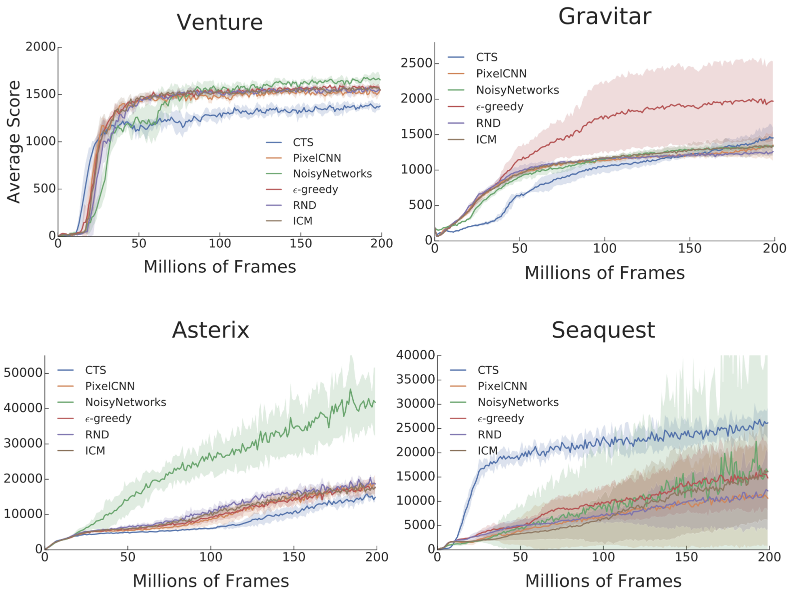
Comparison

[Taïga et al., 2019]



Comparison: not all problems require same amount of exploration

[Taïga et al., 2019]



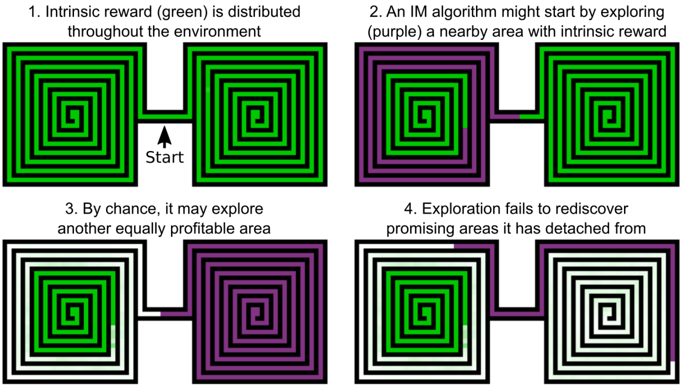
Go-Explore

[Ecoffet et al., 2019]

Issue of *intrinsic motivated* algorithms: *detachment problem*

- Forget about promising areas they have visited
- They do not return to them for further exploration

Green areas indicate intrinsic reward, white indicates areas where no intrinsic reward remains, and purple areas indicate where the algorithm is currently exploring.

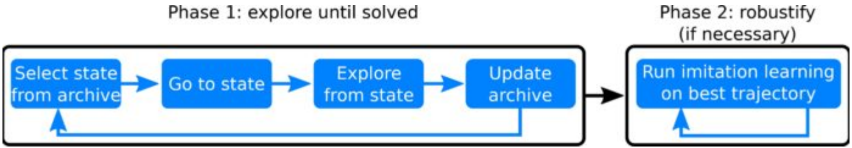


*mainly due to model-free nature

Go-Explore: Structure

[Ecoffet et al., 2019]

1. Exploration
   - Select a promising state
   - Go to a state
   - Explore locally (e.g., randomly)
   - Store observations
   - Repeat



* similar in spirit to [Lim and Auer, 2012]

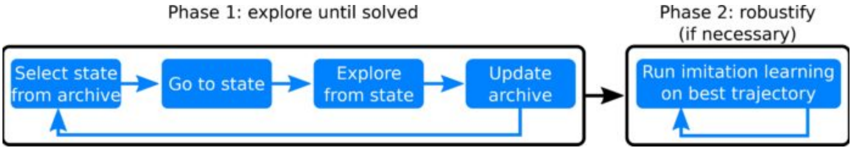
Go-Explore: Structure

[Ecoffet et al., 2019]

1 Exploration

- Select a promising state
- Go to a state
- Explore locally (e.g., randomly)
- Store observations
- Repeat

- Builds an archive of observed states in latent space
- Archive is sorted by relevance (e.g., IM, novelty)
- Knows a policy to reach an observed state (e.g., by replaying)



* similar in spirit to [Lim and Auer, 2012]

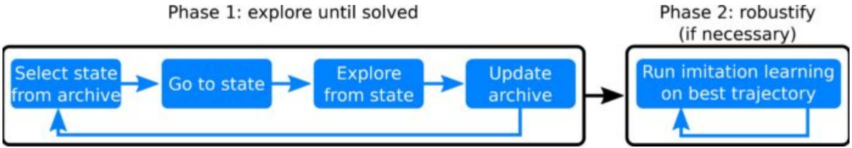
Go-Explore: Structure

[Ecoffet et al., 2019]

1 Exploration
- Select a promising state
- Go to a state
- Explore locally (e.g., randomly)
- Store observations
- Repeat

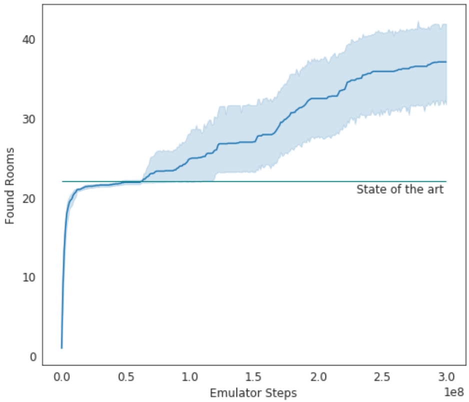
2 *Robustification*
- Against noise
- Imitation learning on best trajectories

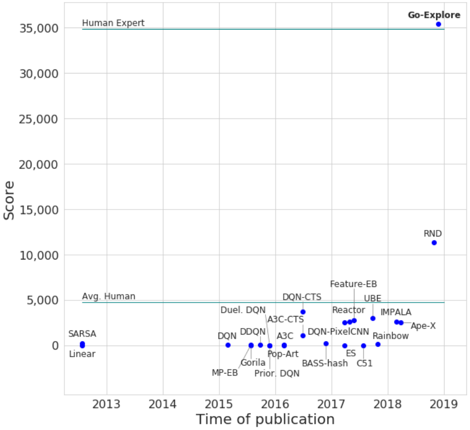
- Builds an archive of observed states in latent space
- Archive is sorted by relevance (e.g., IM, novelty)
- Knows a policy to reach an observed state (e.g., by replaying)



* similar in spirit to [Lim and Auer, 2012]

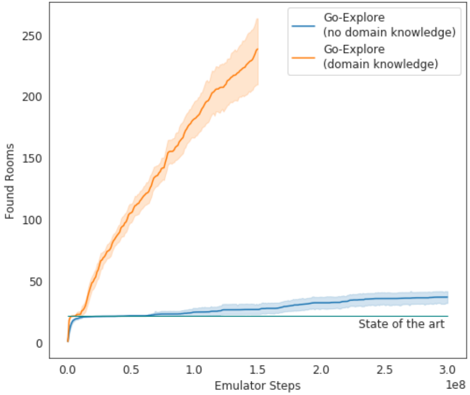
Go-Explore on Montezuma
[Ecoffet et al., 2019]

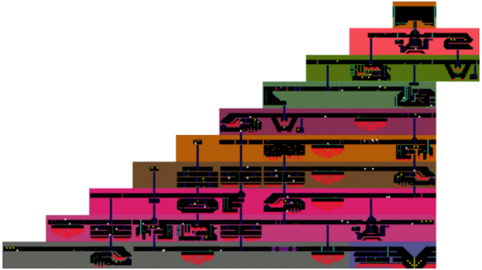




Go-Explore on Montezuma

[Ecoffet et al., 2019]





Using domain knowledge for the state representation, Phase 1 of Go-Explore finds a 238 rooms, solves over 9 levels on average

Image Credit: Wikimedia Foundation

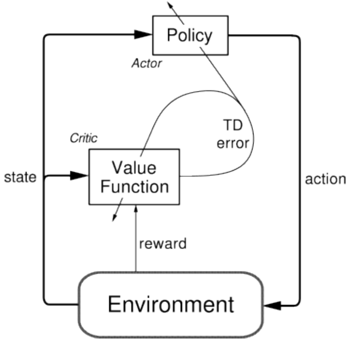
Go-Explore on Pitfall

[Ecoffet et al., 2019]

Pitfall!

Optimistic Actor-Critic

[Ciosek et al., 2019]



- Actor: decides which action to take $\implies \pi$
- Critic: estimates the goodness of an action in a state $\implies Q$

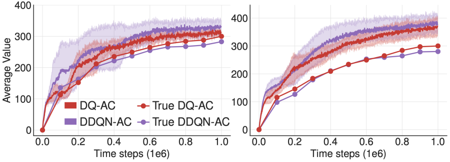
Policy performance: $J^\pi = \mathbb{E}_\pi \left[\sum_t \gamma^t r_t \right]$

Policy gradient:

$$\nabla J^\pi = \mathbb{E}_{s \sim d^\pi} \left[\sum_a \nabla \pi(s,a) Q^\pi(s,a) \right]$$

Optimistic Actor-Critic

[Ciosek et al., 2019]



(a) Hopper-v1 (b) Walker2d-v1

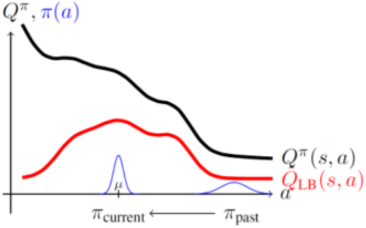
* image from [Fujimoto et al., 2018]

To limit overestimation (i.e., positive bias)

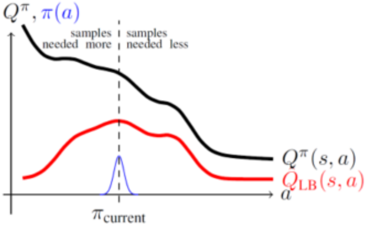
- use of two *identical* Q-functions
- train them independent

$$Q_{LB} = \min\{Q_1, Q_2\}$$

Issues with LB and greedy



(a) Pessimistic underexploration



(b) Directional uninformedness

Optimistic Actor-Critic

[Ciosek et al., 2019]

- Build an upper-bound to the true Q-value (*optimistic Q-value*)

$$\widehat{Q}(s,a) = \mu_Q(s,a) + \beta\sigma_Q(s,a)$$

with

$$\mu_Q(s,a) = \frac{1}{2}\Big(Q^1(s,a) + Q^2(s,a)\Big), \quad \sigma_Q(s,a) = \sqrt{\sum_{i\in\{1,2\}} \frac{1}{2}\left(Q^i(s,a) - \mu_Q(s,a)\right)^2}$$

- *Exploration policy* by soft-update (for stability): $\pi_E = \mathcal{N}(\mu_E, \Sigma_E)$

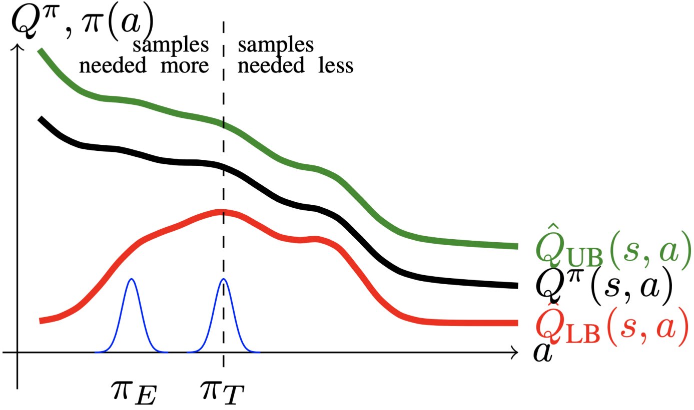
$$(\mu_E, \sigma_E) = \arg\max_{\mu,\Sigma} \mathbb{E}_{a\sim\mathcal{N}(\mu,\Sigma)}\left[\widehat{Q}(s,a)\right]$$

$$s.t. \quad KL\left(\mathcal{N}(\mu,\Sigma), \mathcal{N}(\mu_T,\Sigma_T)\right) \le \delta$$

* for computational efficiency, they fit a linear model on $\widehat{Q}$

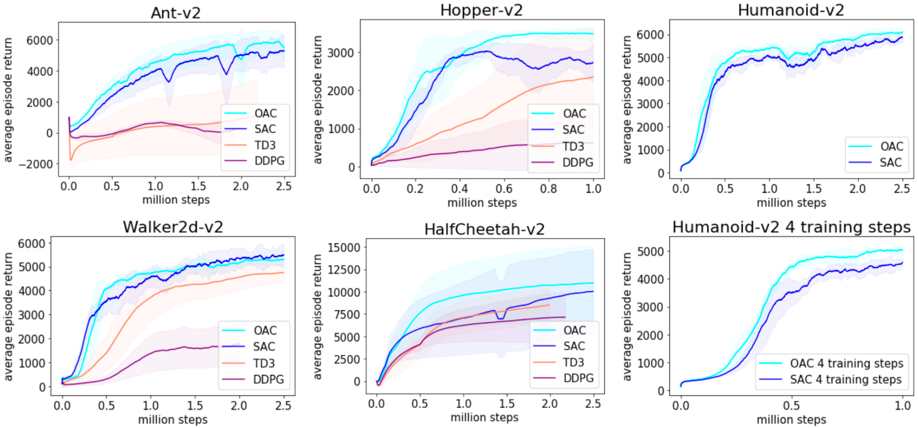
Optimistic Actor-Critic

[Ciosek et al., 2019]



Optimistic Actor-Critic: Experiments

[Ciosek et al., 2019]



Outline

Randomized Exploration

General Scheme

1. Estimate the parameters θ for either policy or value function
2. Add randomness to the parameters $\widetilde{\theta} = \theta + \text{noise}$
3. Run the corresponding (greedy) policy

Remark: changing weights induces a consistent, and potentially very complex, state-dependent change in policy over multiple time steps
$\Longrightarrow$ long-term exploration
$\Longrightarrow$ no dithering

Randomized Exploration

General Scheme

1. Estimate the parameters θ for either policy or value function
2. Add randomness to the parameters $\widetilde{\theta} = \theta + \text{noise}$
3. Run the corresponding (greedy) policy

Remark: changing weights induces a consistent, and potentially very complex, state-dependent change in policy over multiple time steps

$\Longrightarrow$ long-term exploration

$\Longrightarrow$ no dithering

⚠ The randomness needs to represent "uncertainty"

Least-Squares Value Iteration

Estimate $Q^\star$ from samples

Input: Dataset $\mathcal{D}_k = (s_{hi}, a_{hi}, r_{hi})_{h=1,i=1}^{H,k}$
Set $\widehat{Q}_{H+1}(s,a) = 0$
for $h = H, \dots, 1$ **do** `// backward induction`
Compute

$$y_{hi} = r_{hi} + \max_{a\in\mathcal{A}} \widehat{Q}_{h+1,k}(s_{h+1,i}, a) = r_{hi} + \widehat{V}_{h+1,k}(s_{h+1,i}) \ , \quad i = 1, \dots, k$$

Build regression dataset $\mathcal{D}_h^{\text{reg}} = \{\phi_h(s_{hi}, a_{hi}), y_{hi}\}_i$
Compute

$$\widehat{\theta}_{hk} = \arg\min_{\theta} \left\{ \mathcal{L}(\theta, \mathcal{D}^{\text{reg}}) := \frac{1}{k} \sum_{i=1}^{k} \left(y_{hi} - Q_{hk}(s_{hi}, a_{hi}|\theta) \right)^2 \right\}$$

end
return $\{\widehat{\theta}_{hk}\}_{h=1}^{H}$

Optimize $\mathcal{L}$ by *gradient descent*

Randomization on LSVI

How to force exploration

- Perturbe observed rewards
- Perturbe parameters (e.g., based on posterior uncertainty)

Randomized Value Function (RVF) [Osband et al., 2019, 2018, Azizzadenesheli et al., 2018, Lipton et al., 2018, Touati et al., 2019, Osband et al., 2019]

RVF: Reward Perturbation

[Osband et al., 2018, 2019]

Input: Dataset $\mathcal{D}_k = (s_{hi}, a_{hi}, r_{hi})_{h=1,i=1}^{H,k}$

Set $\widehat{Q}_{H+1}(s,a) = 0$

for $h = H, \dots, 1$ **do** `// backward induction`

Perturb rewards

$$\widetilde{r}_{hi} = r_{hi} + \omega_{hi}, \quad \omega_{hi} \sim \mathcal{N}(0, \sigma^2)$$

Compute

$$\widetilde{y}_{hi} = \widetilde{r}_{hi} + \max_{a \in \mathcal{A}} \widehat{Q}_{h+1,k}(s_{h+1,i}, a) = \widetilde{r}_{hi} + \widehat{V}_{h+1,k}(s_{h+1,i}), \ \ i = 1, \dots, k$$

Build regression dataset $\widetilde{\mathcal{D}}_h^{\text{reg}} = \{(s_{hi}, a_{hi}), \widetilde{y}_{hi}\}_i$

Sample θ^p *from prior*

Compute

$$\widehat{\theta}_{hk} = \arg\min_{\theta} \left\{ \mathcal{L}^B(\theta, \theta^p, \widetilde{\mathcal{D}}^{\text{reg}}) := \frac{1}{k} \sum_{i=1}^{k} (\widetilde{y}_{hi} - Q_{hk}(s_{hi}, a_{hi} | \theta))^2 + \mathcal{R}(\theta, \theta^p) \right\}$$

end

return $\{\widehat{\theta}_{hk}\}_{h=1}^{H}$

RVF: Perturb Parameters

[Osband et al., 2016b]

Input: Dataset $\mathcal{D}_k = (s_{hi}, a_{hi}, r_{hi})_{h=1,i=1}^{H,k}$

Set $\overline{Q}_{H+1}(s,a) = 0$

for $h = H, \ldots, 1$ **do** `// backward induction`

Compute

$$\overline{y}_{hi} = r_{hi} + \max_{a \in \mathcal{A}} \overline{Q}_{h+1,k}(s_{h+1,i}, a) = r_{hi} + \overline{V}_{h+1,k}(s_{h+1,i}) \;, \quad i = 1, \ldots, k$$

Bootstrapping randomized estimates

Build regression dataset $\overline{\mathcal{D}}_h^{\text{reg}} = \{\phi_h(s_{hi}, a_{hi}), \overline{y}_{hi}\}_i$

Sample θ^p *from prior*

Compute

$$\widehat{\theta}_{hk} = \arg\min_{\theta} \left\{ \mathcal{L}^B(\theta, \theta^p, \overline{\mathcal{D}}^{\text{reg}}) := \frac{1}{k} \sum_{i=1}^{k} \left(\overline{y}_{hi} - Q_{hk}(s_{hi}, a_{hi} | \theta) \right)^2 + \mathcal{R}(\theta, \theta^p) \right\}$$

Sample $\xi_{hk} \sim \mathcal{N}(0, \Sigma_{hk}^{-1})$

Set $\overline{\theta}_{hk} = \widehat{\theta}_{hk} + \xi_{hk}$

end

return $\{\overline{\theta}_{hk}\}_{h=1}^{H}$

$\widehat{\theta} = \mathbb{E}[\theta | \mathcal{D}^{\text{reg}}, prior], \; \Sigma^{-1} = Cov[\theta | \mathcal{D}^{\text{reg}}, prior]$

RLSVI as Regression on Perturbed Data

[Osband et al., 2018, 2019]

Bayesian Linear Regression (posterior structure)

- True parameter is $\theta^\star \in \mathbb{R}^d \Rightarrow$ we want to estimate it
- Assume *Gaussian prior* $\mathcal{N}(\overline{\theta}, \lambda I)$
- Dataset $\mathcal{D} = (x_i, y_i)_{i=1}^N$, where

$$y_i = x_i^\mathsf{T} \theta^\star + \epsilon_i \quad , \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

- Solve $\min_\theta \mathcal{L}^B(\theta, \overline{\theta}, \mathcal{D})$
- Conditional *posterior* μ_p

$$\theta^\star | \mathcal{D} \sim \mu_p = \mathcal{N}\left(\overbrace{\Sigma^{-1} \left(\frac{1}{\sigma^2} X^\mathsf{T} y + \frac{1}{\lambda} \overline{\theta} \right)}^{\widehat{\theta}}, \; \Sigma^{-1} \right)$$

$$\Sigma = \frac{1}{\sigma^2} X^\mathsf{T} X + \frac{1}{\lambda} I$$

RLSVI as Regression on Perturbed Data

[Osband et al., 2018, 2019]

Target perturbation

- Compute

Perturbation
$\omega_i \sim \mathcal{N}(0, \sigma^2)$

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{\sigma^2} \sum_{i=1}^{N} \left(y_i + \omega_i - x_i^\mathsf{T} \theta \right)^2 + \frac{1}{\lambda} \| \widetilde{\theta} - \theta \|_2^2$$

Sample from prior
$\widetilde{\theta} \sim \mathcal{N}(\overline{\theta}, \lambda I)$

$$\Rightarrow \widehat{\theta} \sim \mu_p$$

👍 Computational generation of posterior samples for linear Bayesian regression

i.e., we can sample μ_p by fitting a least-squares estimate

RLSVI as Regression on Perturbed Data

[Osband et al., 2018, 2019]

Target perturbation

- Compute

Perturbation
$\omega_i \sim \mathcal{N}(0, \sigma^2)$

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{\sigma^2} \sum_{i=1}^{N} \left(y_i + \omega_i - x_i^\mathsf{T} \theta \right)^2 + \frac{1}{\lambda} \| \widetilde{\theta} - \theta \|_2^2$$

Sample from prior
$\widetilde{\theta} \sim \mathcal{N}(\overline{\theta}, \lambda I)$

$$\Rightarrow \widehat{\theta} \sim \mu_p$$

Computational generation of posterior samples for linear Bayesian regression

i.e., we can sample μ_p by fitting a least-squares estimate

For linear models,

poster sampling = regularized least-squares on perturbed data

For tabular MDPs, $x_i = e_{s,a}$ and $\theta = Q$

backward induction on randomized rewards = RLSVI (*see Part 2*)

RVF: issues

Reward Perturbation

- Minimize least-squares problem for any reward structure
  e.g., by gradient descent
- Not so easy to define the magnitude of the reward perturbation

Posterior Sampling

- Posterior variance
  - easy for linear model
  - hard (almost impossible) for generic models
- A lot of approximate schemas for computing the posterior

Randomized Prior

[Osband et al., 2018]

Algorithm 1 Randomized prior functions for ensemble posterior.

Require: Data $\mathcal{D} \subseteq \{(x,y) | x \in \mathcal{X}, y \in \mathcal{Y}\}$, loss function $\mathcal{L}$, neural model $f_\theta : \mathcal{X} \to \mathcal{Y}$,
Ensemble size $K \in \mathbb{N}$, noise procedure `data_noise`, distribution over priors $\mathcal{P} \subseteq \{\mathbb{P}(p) | p : \mathcal{X} \to \mathcal{Y}\}$.

for $k = 1, .., K$ **do**
  initialize $\theta_k \sim$ Glorot initialization [23].
  form $\mathcal{D}_k =$ `data_noise`$(\mathcal{D})$ (e.g. Gaussian noise or bootstrap sampling [50]).
  sample prior function $p_k \sim \mathcal{P}$.
  optimize $\nabla_{\theta | \theta = \theta_k} \mathcal{L}(f_\theta + p_k; \mathcal{D}_k)$ via ADAM [28].
return ensemble $\{f_{\theta_k} + p_k\}_{k=1}^K$.

$$\mathcal{L}_\gamma(\theta; \theta^-, p, \mathcal{D}) := \sum_{t \in \mathcal{D}} \left(r_t + \gamma \max_{a' \in \mathcal{A}} \overbrace{(f_{\theta^-} + p)}^{\text{target } Q}(s'_t, a') - \overbrace{(f_\theta + p)}^{\text{online } Q}(s_t, a_t) \right)^2$$

Posterior Distribution for Deep Neural Networks

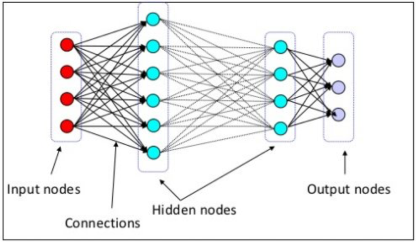
Bayesian DQN [Azizzadenesheli et al., 2018]

Same tools as in linear bandit

1. Bayesian linear regression with given feature $\phi(s) \in \mathbb{R}^d$ and given target vector for each action y_a

$$\mu_a = (\Phi_a^\mathsf{T}\Phi_a)^{-1}\Phi_a^\mathsf{T} y_a \qquad \Sigma_a = \Phi_a^\mathsf{T}\Phi_a$$

2. Draw a weight vector at random $w_a \sim \mathcal{N}(\mu_a, \Sigma_a^{-1})$
3. Run the corresponding (greedy) policy
$a_t = \arg\max_a Q(s_t, a) := \arg\max_a w_a^\mathsf{T}\phi(s_t)$
4. Train ϕ with standard NN to estimate Q



Posterior Distribution for Deep Neural Networks

Bayesian DQN [Azizzadenesheli et al., 2018]

Game	BDQN	DDQN	DDQN$^+$	Bootstrap	NoisyNet	CTS	Pixel	Reactor	Human	SC	SC^+	Step
Amidar	**5.52k**	0.99k	0.7k	1.27k	1.5k	1.03k	0.62k	1.18k	1.7k	22.9M	4.4M	100M
Alien	3k	2.9k	2.9k	2.44k	2.9k	1.9k	1.7k	**3.5k**	6.9k	-	36.27M	100M
Assault	**8.84k**	2.23k	5.02k	8.05k	3.1k	2.88k	1.25k	3.5k	1.5k	1.6M	24.3M	100M
Asteroids	**14.1k**	0.56k	0.93k	1.03k	2.1k	3.95k	0.9k	1.75k	13.1k	58.2M	9.7M	100M
Asterix	**58.4k**	11k	15.15k	19.7k	11.0	9.55k	1.4k	6.2k	8.5k	3.6M	5.7M	100M
BeamRider	8.7k	4.2k	7.6k	**23.4k**	14.7k	7.0k	3k	3.8k	5.8k	4.0M	8.1M	70M
BattleZone	**65.2k**	23.2k	24.7k	36.7k	11.9k	7.97k	10k	45k	38k	25.1M	14.9M	50M
Atlantis	3.24M	39.7k	64.76k	99.4k	7.9k	1.8M	40k	**9.5M**	29k	3.3M	5.1M	40M
DemonAttack	11.1k	3.8k	9.7k	82.6k	26.7k	**39.3k**	1.3k	7k	3.4k	2.0M	19.9M	40M
Centipede	**7.3k**	6.4k	4.1k	4.55k	3.35k	5.4k	1.8k	3.5k	12k	-	4.2M	40M
BankHeist	0.72k	0.34k	0.72k	**1.21k**	0.64k	1.3k	0.42k	1.1k	0.72k	2.1M	10.1M	40M
CrazyClimber	124k	84k	102k	**138k**	121k	112.9k	75k	119k	35.4k	0.12M	2.1M	40M
ChopperCmd	**72.5k**	0.5k	4.6k	4.1k	5.3k	5.1k	2.5k	4.8k	9.9k	4.4M	2.2M	40M
Enduro	1.12k	0.38k	0.32k	1.59k	0.91k	0.69k	0.19k	**2.49k**	0.31k	0.82M	0.8M	30M
Pong	**21**	18.82	**21**	20.9	**21**	20.8	17	20	9.3	1.2M	2.4M	5M

Posterior Distribution for Deep Neural Networks

BBQ-Networks [Lipton et al., 2018]

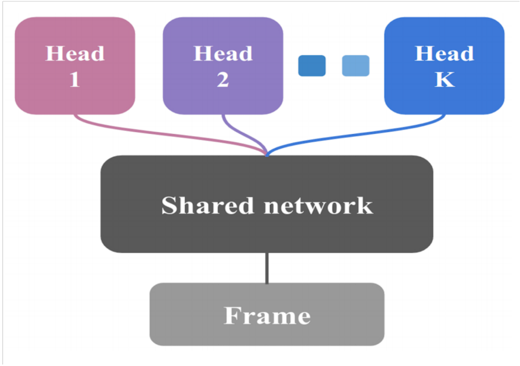
- Uses variational inference to quantify uncertainty
- Uses independent factorized Gaussians as an approximate posterior

MNF-DQN [Touati et al., 2019]

- Leverages recent advances in variational Bayesian NN
- Computationally and statistically efficient
- Uses normalizing multiplicative flows (MNF) in order to account for the uncertainty of estimates for efficient exploration

Bootstrap DQN
[Osband et al., 2016a]

- Define multiple value functions Q_k
- Update functions with different datasets
- Share part of the architecture



another way of approximating a sample from posterior

Bootstrap DQN

[Osband et al., 2016a]

Algorithm 1 Bootstrapped DQN

Input: Value function networks Q with K outputs $\{Q_k\}_{k=1}^{K}$. Masking distribution M.
Let B be a replay buffer storing experience for training.
for each episode **do**
  Obtain initial state from environment s_0
  Pick a value function to act using $k \sim \text{Uniform}\{1, \ldots, K\}$
  for step $t = 1, \ldots$ until end of episode **do**
    Pick an action according to $a_t \in \arg\max_a Q_k(s_t, a)$
    Receive state s_{t+1} and reward r_t from environment, having taking action a_t
    Sample bootstrap mask $m_t \sim M$
    Add $(s_t, a_t, r_{t+1}, s_{t+1}, m_t)$ to replay buffer B
  end for
end for

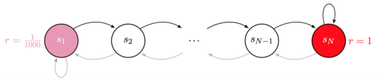
- M_t determines the type of bootstrapping strategy

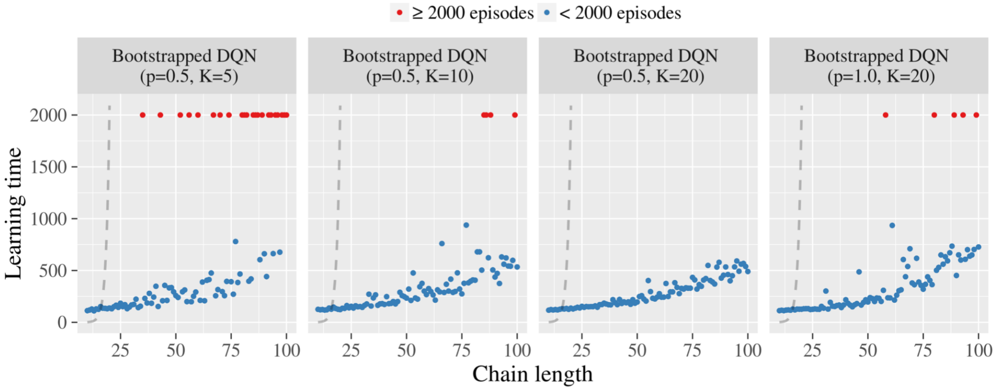
$$g_t^k = m_t^k \big(y_t^Q - Q_k(s_t, a_t; \theta)\big) \nabla_\theta Q_k(s_t, a_t, ; \theta)$$

with target $y_t = r_t + \max_a Q(s_{t+1}, a; \theta^-)$

Bootstrap DQN

[Osband et al., 2016a]

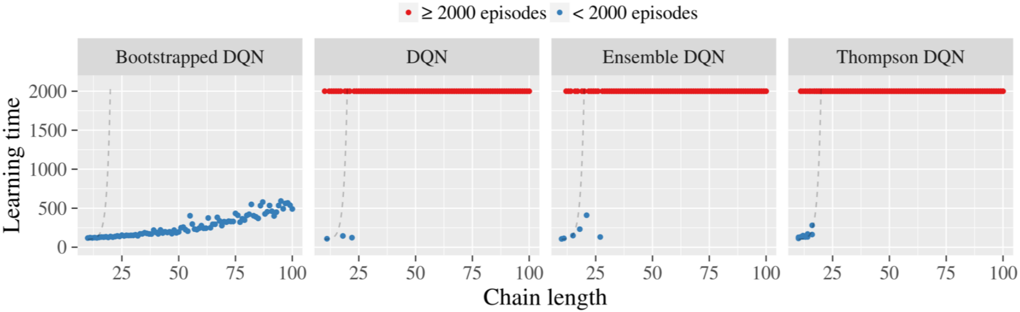




Masking rule for samples in episode k: $m_k \sim Ber(p)$

Bootstrap DQN

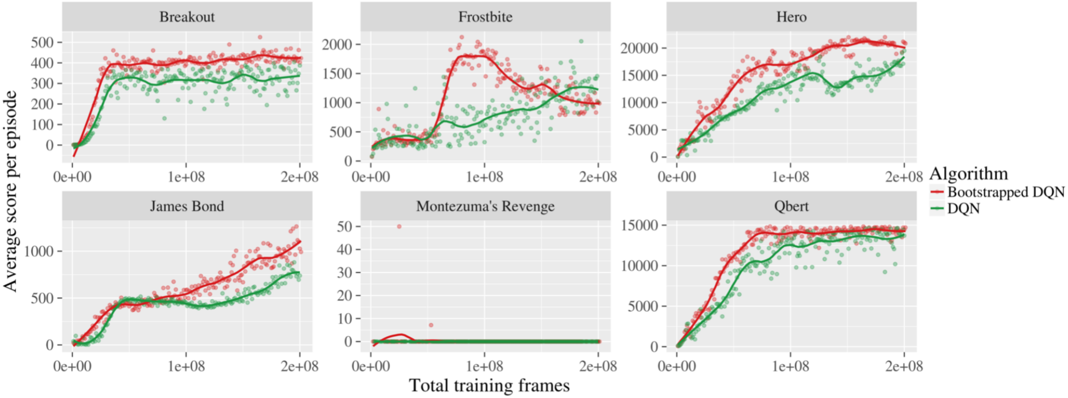
[Osband et al., 2016a]



- Ensemble DQN: ensemble policy?
- Thompson DQN: resample at each step

Bootstrap DQN

[Osband et al., 2016a]



Noisy Networks

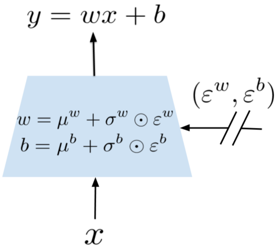
[Fortunato et al., 2018]

- Normal NN layer $y = wx + b$
- Double the parameters with mean and variance $w \to \mu^w, \sigma^w$ and $b \to \mu^b, \sigma^b$
- Whenever a layer is evaluated draw $\varepsilon^w, \varepsilon^b \sim \mathcal{D}$
- Evaluate the "random" layer as $y = (\mu^w + \sigma^w \odot \varepsilon^w) + \mu^b + \sigma^b \odot \varepsilon^b$
- Let $\zeta = (\mu^w, \sigma^w, \mu^b, \sigma^b)$, define the expected loss

$$\overline{L}(\zeta) = \mathbb{E}_\varepsilon\big[L(\zeta, \varepsilon)\big]$$

- Gradient estimation

$$\nabla_\zeta \overline{L}(\zeta) = \mathbb{E}_\varepsilon\big[\nabla_\zeta L(\zeta, \varepsilon)\big] \approx \frac{1}{n}\sum_{i=1}^{n} \nabla_\zeta L(\zeta, \varepsilon_i)$$



Noisy Networks

[Fortunato et al., 2018]

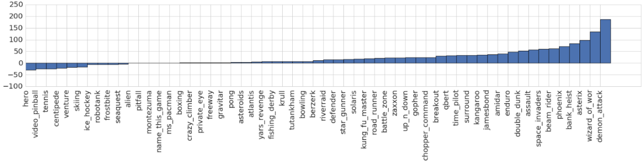
Noise models

- Independent noise $\varepsilon_{i,j}$ for each weight i at layer j
- Factorized noise $\varepsilon_{i,j} = f(\varepsilon_i)f(\varepsilon_j)$ (e.g., $f(x) = \text{sgn}(x)\sqrt{x}$)
- Independent noise for target and online networks

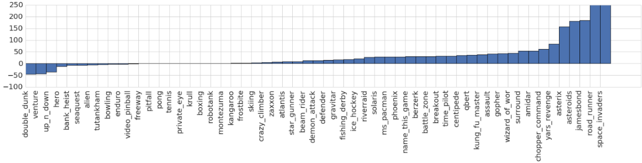
$$y_t = r_t + \max_{a'} Q(s'_t, a'; \varepsilon', \zeta^-); \qquad L_t(\zeta, \varepsilon) = \big(y_t - Q(s_t, a_t; \varepsilon, \zeta)\big)^2$$

Noisy Networks

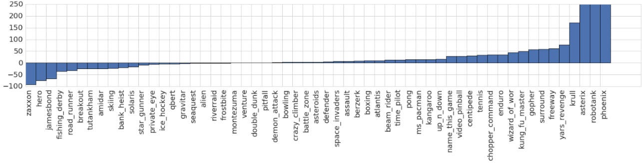
[Fortunato et al., 2018]



(a) Improvement in percentage of NoisyNet-DQN over DQN (Mnih et al., 2015)



(b) Improvement in percentage of NoisyNet-Dueling over Dueling (Wang et al., 2016)



(c) Improvement in percentage of NoisyNet-A3C over A3C (Mnih et al., 2016)

Comparison
[Touati et al., 2019]

Simple Chain domain

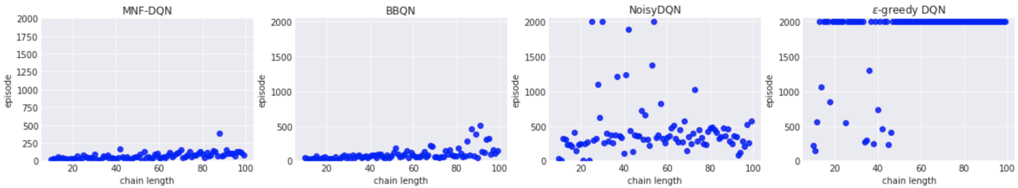
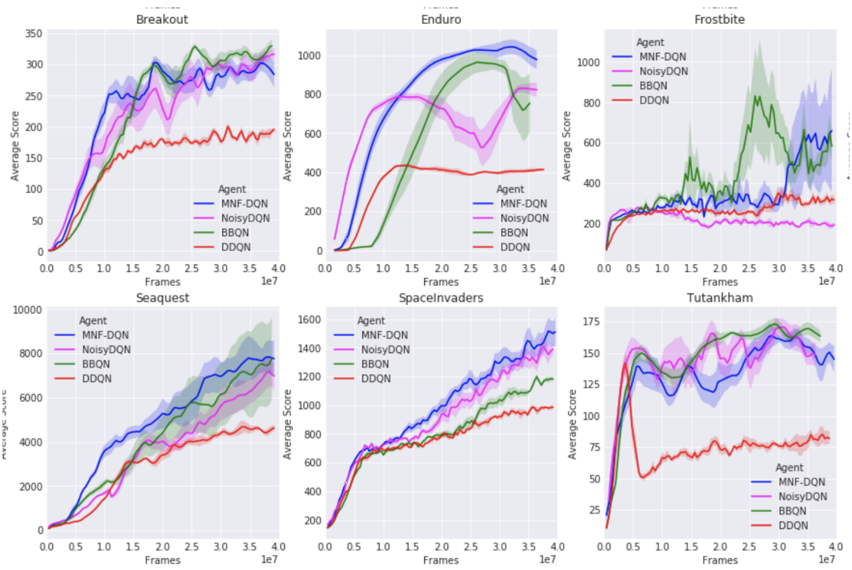


Figure 1: Median number of episodes (max 2000) required to solve the n-chain problem for (figure from left to right) MNF-DQN, BBQN, NoisyDQN and ϵ-greedy DQN. The median is obtained over 10 runs with different seeds. We see that MNF-DQN consistently performs best across different chain lengths.

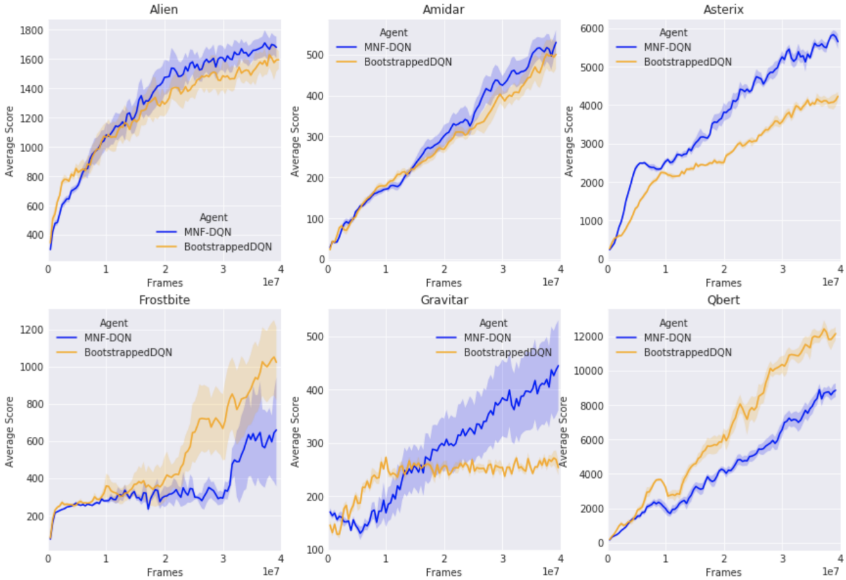
Comparison: Atari

[Touati et al., 2019]



Comparison: Atari

[Touati et al., 2019]



Remarks

- Exploration needs to account for uncertainty in the predictions
- Should account for long-term effect

Exploration at the level of (value/policy/model) parameters

Randomized explorations performs often better than optimism

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. In *ITA*, pages 1–9. IEEE, 2018.

Marc G. Bellemare, Joel Veness, and Erik Talvitie. Skip context tree switching. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1458–1466. JMLR.org, 2014.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. In *NIPS*, pages 1471–1479, 2016.

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. In *ICLR (Poster)*. OpenReview.net, 2019.

Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor critic. In *NeurIPS*, pages 1785–1796, 2019.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In *ICLR (Poster)*. OpenReview.net, 2018.

Lior Fox, Leshem Choshen, and Yonatan Loewenstein. DORA the explorer: Directed outreaching reinforcement action-selection. In *ICLR (Poster)*. OpenReview.net, 2018.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1582–1591. PMLR, 2018.

Shiau Hong Lim and Peter Auer. Autonomous exploration for navigating in mdps. In *COLT*, volume 23 of *JMLR Proceedings*, pages 40.1–40.24. JMLR.org, 2012.

Zachary C. Lipton, Xiujun Li, Jianfeng Gao, Lihong Li, Faisal Ahmed, and Li Deng. Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems. In *AAAI*, pages 5237–5244. AAAI Press, 2018.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. Count-based exploration with the successor representation. *CoRR*, abs/1807.11622, 2018.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *NIPS*, pages 4026–4034, 2016a.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2377–2386. JMLR.org, 2016b.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *NeurIPS*, pages 8626–8638, 2018.

Ian Osband, Benjamin Van Roy, Daniel J. Russo, and Zheng Wen. Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019. URL `http://jmlr.org/papers/v20/18-339.html`.

Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 2721–2730. PMLR, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 2778–2787. PMLR, 2017.

Yifan Sun, Yaqi Duan, Hao Gong, and Mengdi Wang. Learning low-dimensional state embeddings and metastable clusters from time series data. In *NeurIPS*, pages 4563–4572, 2019.

Adrien Ali Taïga, William Fedus, Marlos C. Machado, Aaron C. Courville, and Marc G. Bellemare. Benchmarking bonus-based exploration methods on the arcade learning environment. *CoRR*, abs/1908.02388, 2019.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*, pages 2753–2762, 2017.

Ahmed Touati, Harsh Satija, Joshua Romoff, Joelle Pineau, and Pascal Vincent. Randomized value functions via multiplicative normalizing flows. In *UAI*, page 156. AUAI Press, 2019.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1747–1756. JMLR.org, 2016.