Offline Reinforcement Learning

Reading

- Levine, Sergey, Aviral Kumar, George Tucker, and Justin Fu. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems." *arXiv preprint arXiv:2005.01643* (2020).
- Kaiser, Lukasz, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan et al. "Model-based reinforcement learning for atari." *arXiv preprint arXiv:1903.00374* (2019).

Overview

Rensselaer

- Only seen online RL methods so far
  - The algorithm interacts with the system and iteratively learns a better policy over time
- This approach is not very practical in many real-life applications
  - Why?
  - Data collection is expensive (think robotics, healthcare)
  - Learning can be dangerous (e.g., autonomous driving, healthcare)
- Instead, it makes sense to collect a dataset a learn a policy from that dataset
  - Similar to supervised learning on images, language, etc.

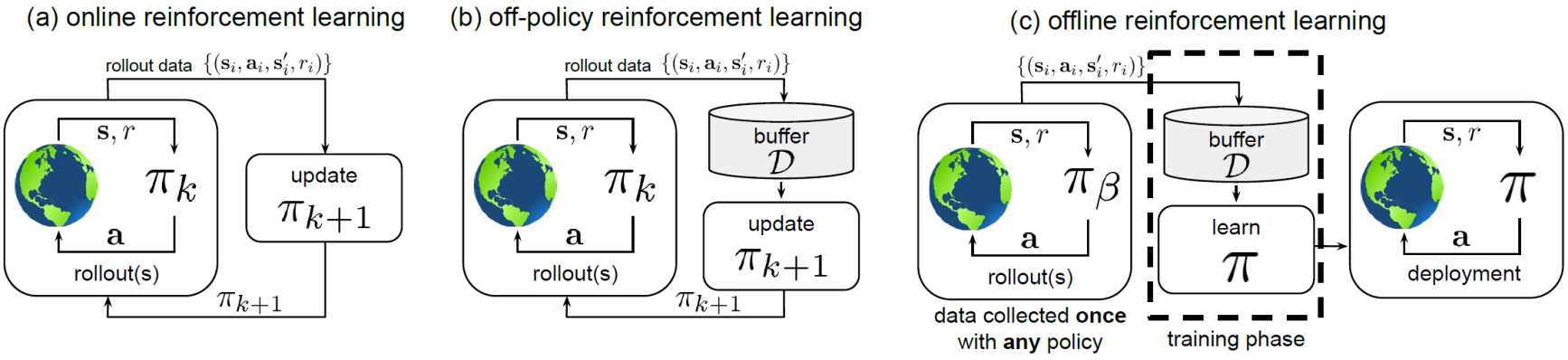
Setup

Rensselaer

- We have an MDP as before
  - May or may not be finite
  - MDP dynamics may or may not be known
- We are given a static dataset $\mathcal{D} = \{(S_t^i, A_t^i, R_{t+1}^i, S_{t+1}^i)\}$
  - where i is the trajectory index
  - and t is the time within each trajectory
  - similar to an experience replay buffer in deep RL, except it doesn't change
- Similar to a supervised learning setting
- In essence, the offline RL problem amounts to figuring out the underlying reward structure and finding the optimal policy for it

Different Shades of RL

- Ultimately, offline RL is a combination of off-policy RL and supervised learning
- Though one could envision a framework where the “training” dataset is gradually collected by carefully updating the behavior over time



Examples: Healthcare

- Model the process of diagnosis and treatment of a patient
- Collect data of interventions and corresponding outcomes
- One example from my research is automated ventilator weaning
  - Some patients cannot go back to spontaneous breathing after surgery
  - Need to develop a gradual weaning method
  - Have historic data on ventilator settings, outcomes, etc.
  - Need to learn an optimal policy
  - Problem is made extra hard since each patient is different

Examples: Learning Robotic Manipulation Skills

Rensselaer

- In controlled settings, online RL may actually be feasible
- Still hard to extend to modifications of the training setting or slightly different tasks
- With offline RL, we can collect all data from past experiments and learn a policy that generalizes to new settings
- The hope is to generalize better than online RL which effectively only works for the current environment
- Could be wishful thinking since new settings may be out of distribution

Challenges with Offline RL

Rensselaer

- Similar to off-policy RL with one major difference
  - Cannot collect additional data
- Existing RL algorithms do not work well on fixed datasets
- Why?
- No exploration
  - It is common to assume that training data is sufficient to learn an optimal policy, though it is hard to even formalize this assumption
- Once the target policy is changed, policy evaluation may be challenging if behavior policy was too different
  - Essentially asking a counterfactual question, i.e., a “what if” question

Offline Evaluation

- Suppose our current policy is π
- Data was generated using behavior policy b
- How do we compute $v_\pi(s)$?
- Off-policy evaluation with importance sampling!

$$v_\pi(s) = \mathbb{E}_b[\rho_{t:T-1} G_t | S_t = s]$$

  - where $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

- This estimate has high variance that can be improved by normalizing the weights
- What is the main issue with this approach?
  - Requires the training data to have explored most state/action pairs

Off-Policy Policy Gradients

- Offline RL could use the off-policy gradient policy approach
- Recall the off-policy deterministic policy gradient

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \int_S d_b(s) \nabla_{\boldsymbol{\theta}} \pi(s; \boldsymbol{\theta}) \nabla_a q_\pi(s, a) \Big|_{a=\pi(s)} ds$$

- What is the challenge with this approach?
  - If the behavior policy hasn't explored enough, the policy gradient estimate will be very noisy
  - Effectively, we'll assign very small weights to most actions chosen by the target policy
  - Ultimately, the same problem as importance sampling

Offline RL via Dynamic Programming

- In principle, dynamic programming is a useful approach for offline RL
- If we could approximate the q value for all state/action pairs, then we can use dynamic programming to find the optimal policy
- What are the challenges with this approach?
  - May not have enough data for each state/action pair
  - What type of approximation should we use?
    - Linear approximation may be too simplistic
    - Neural networks may overfit

Dynamic Programming with Linear Q-function Approximations

Rensselaer

- Suppose we approximate the Q-function with a linear function
- Let the features be $\boldsymbol{f}(s, a)$
  - Want to learn parameters $\boldsymbol{\phi}$ such that $\hat{q}(s, a) = \boldsymbol{\phi}^T \boldsymbol{f}(s, a)$
- What is one way to learn these parameters?
  - Least squares using the Bellman equation!
    - Requires knowing the MDP transition function
- Recall that we can write the Bellman equation in matrix form:

$$q(\boldsymbol{s}, \boldsymbol{a}) = R(\boldsymbol{s}) + \gamma \boldsymbol{P} q(\boldsymbol{s}, \boldsymbol{a})$$

- Let $\boldsymbol{F}$ be the matrix of all features. Then

$$\boldsymbol{F}\boldsymbol{\phi} = R(\boldsymbol{s}) + \gamma \boldsymbol{P}\boldsymbol{F}\boldsymbol{\phi}$$

- i.e.,

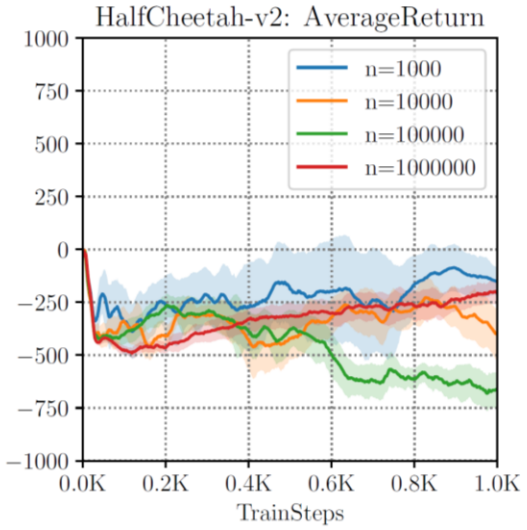
$$\boldsymbol{\phi}^* = \left((\boldsymbol{F} - \gamma \boldsymbol{P}\boldsymbol{F})^T (\boldsymbol{F} - \gamma \boldsymbol{P}\boldsymbol{F})\right)^{-1} (\boldsymbol{F} - \gamma \boldsymbol{P}\boldsymbol{F})^T R(\boldsymbol{s})$$

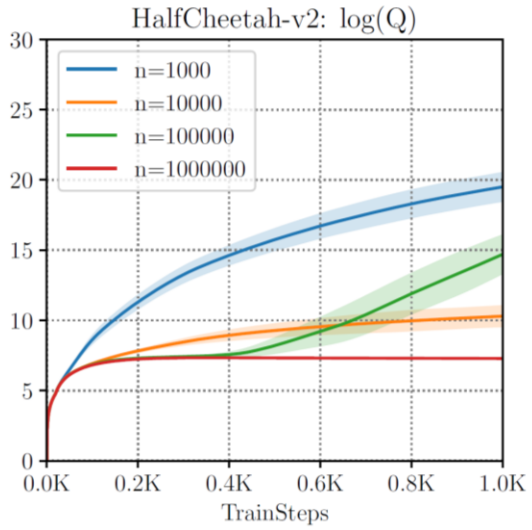
Distribution Shift with Dynamic Programming

- Offline dynamic programming suffers from distribution shift
  - Why?
    - Same reason as any other offline method really
    - Q-estimates may be very wrong for some state-action pairs
  - If behavior policy did not explore enough, dynamic programming will overfit (regardless of how much training data we have)

Soft Actor Critic applied on offline data.

Notice the number of trajectories does not help, i.e., the issue is not overfitting on the training data (rather exploration)





Mitigating Distribution Shift: Policy Constraints

Rensselaer

- The idea of policy constraints is to keep the learned policy similar to the behavior policy
  - What does this achieve?
  - The q function will not be queried on state/action pairs that weren't explored in the training set
- Of course, you want a significantly different policy since the behavior policy is likely far from optimal
- How do we achieve both?
  - Constrained learning, e.g., actor-critic!
  - Effectively learn the best policy that can be evaluated sufficiently well given the available data

Policy Constraints, cont'd

Rensselaer

- Suppose we use a standard actor-critic method
- Iteratively train a critic using (bootstrapped) least squares
  - Then obtain the optimal policy for the current critic
- How do we constrain the policy w.r.t. the behavior policy b?
  - Can add a constraint such as KL divergence
  - E.g., $D_{KL}(\pi||b) \leq \epsilon$
    - Requires knowing the probabilities of actions under b, $b(a|s)$
    - If not, can learn a prior $\pi_{\boldsymbol{\theta}_b}$ approximating behavior policy b
      - E.g., by maximizing the log-likelihood of training data

Offline Model-Based RL

Rensselaer

- The goal of model-based RL is to learn the underlying MDP
  - i.e., the transition probability matrix $P(s, a, s')$
- While still suffering from distribution shift and exploration issues in training data, this is more promising than DP
  - Why?
  - Transition probabilities do not depend on behavior policy
    - Unlike q function approximations
- Model-based RL is similar to a vast area in control theory known as system identification
  - Given training trajectories of states and controls, can you learn system dynamics $x_{k+1} = f(x_k, u_k)$

Model Exploitation and Distribution Shift

Rensselaer

- Once a model is trained, it can be used as a standard simulator in model-free online RL
  - It may still suffer from distribution shift, both through visiting states or actions that weren't explored in the training data
- During RL training, the policy may *exploit* deficiencies in the model
  - How?
  - If the model has assigned an erroneously high reward to some transition, the policy may exploit that
- This also happens in unrealistic driving simulators
  - E.g., car learns to drive through a tree and cut a corner

- Kaiser, Lukasz, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan et al. "Model-based reinforcement learning for atari." *arXiv preprint arXiv:1903.00374* (2019).

Overview

Rensselaer

- It is known that humans can learn Atari games in minutes
- RL methods typically require millions of training steps
- In this paper, the authors ask whether they can learn good policies with a budget of 100K steps
  - Roughly 2 hours of play time
  - No model-free method can get even close to this goal
- The idea is to use a model so as to minimize interaction with the Atari simulator
  - Offline learning steps are not counted towards the 100K

Goal

- The real environment (i.e., the Atari simulator) is called env
- The goal is to sample from env as little as possible
  - Collecting data from real environments is expensive in general
- The authors aim to build a model environment env' that will be used for large-scale sampling
  - Essentially run model-free RL on env'
- In some sense, the RL algorithms remain the same
  - The main difference is the use of a model so as to minimize interactions with the real environment

System Identification Aside

- System identification is quite challenging for several reasons
- Suppose training data contains trajectories $\left\{\left(x_k^t, u_k^t, x_{k+1}^t\right)\right\}$
  - where t is the trajectory index and k is the time within the trajectory
- What challenges do you see when trying to approximate the real dynamics $x_{k+1} = f(x_k, u_k)$?
- Samples within a trajectory are not independent
  - Can't just use them in a standard supervised fashion
- Data was generated using a behavior controller
  - Need sufficient exploration so as to ensure good learning

Learning an Image Model

Rensselaer

- Learning a predictive model for images is particularly hard
  - Why?
  - Output is high-dimensional
  - A single image doesn't store the full state (e.g., velocity)
    - Need to stack multiple (4) images together as input
- Specifying the loss function is also not trivial
  - Authors experiment with softmax and L_2
  - Regardless of the loss choice, what challenges do you see?
  - When you average the loss over many pixels, non-important pixels may dominate
  - Authors resolve this issue by capping the loss per pixel
    - Results in better behaved gradients across all pixels

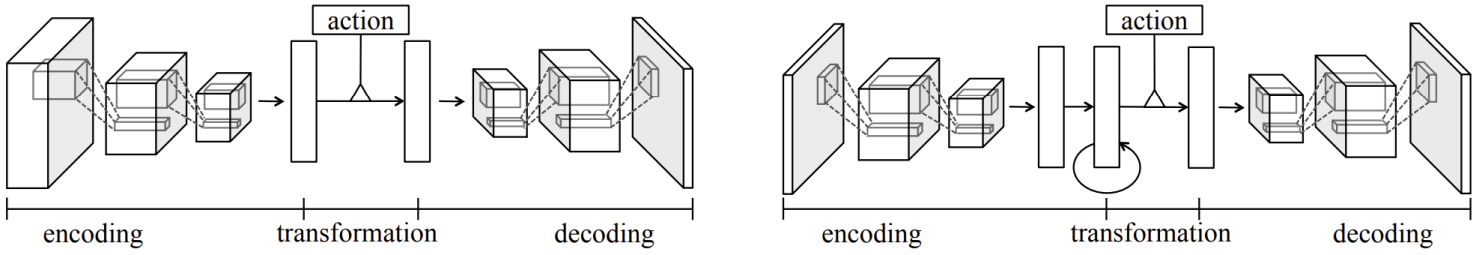
Learning an Image Model, cont'd

Rensselaer

- Neural networks that output high-dimensional objects, such as images, are called generative models
  - Examples include generative adversarial networks (GANs) and variational autoencoders (VAEs)
  - Foundation models are variants of the above
- Generative models use a number of layers that are not present in classification models
  - Deconvolutional layers
  - Attention
  - Recurrent layers

Autoencoders

- Idea is to use an encoder and decoder (autoencoder)
  - Encode the input as a low-dimensional embedding and combine with actions
  - Then decode the embedding into the predicted output
- Recurrent encoding allows one to use a single frame at a time since it's able to remember the past
  - Note that this model is deterministic
    - May need stochasticity, e.g., for random environment events

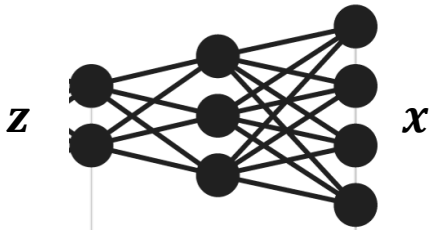


(a) Feedforward encoding (b) Recurrent encoding

Oh, J., Guo, X., Lee, H., Lewis, R. L., & Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. Advances in neural information processing systems, 28.

Variational Autoencoders

- Suppose we want to add stochasticity to the generative model
  - i.e., we want to learn the data-generating process for a dataset $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$
  - One approach is to assume there is a low-dimensional representation of the high-dimensional data
    - E.g., an MNIST digit can be described by its shape, etc.
    - Called latent variables



- Goal is to
  - Sample latent variable $\boldsymbol{z}_i \sim p(\boldsymbol{z})$
  - Sample a high-dimensional example $\boldsymbol{x}_i \sim p(\boldsymbol{x}|\boldsymbol{z}_i)$

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In 2nd International Conference on Learning Representations, ICLR 2014

Variational Autoencoders, cont'd

- Goal is to
  - Sample latent variable $\boldsymbol{z}_i \sim p(\boldsymbol{z})$
  - Sample a high-dimensional example $\boldsymbol{x}_i \sim p(\boldsymbol{x}|\boldsymbol{z}_i)$
- What is the challenge?
  - We don't observe $\boldsymbol{z}$'s
- Ideally, we would use marginalization

$$p(\boldsymbol{x}) = \int p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z}$$

- Or Bayes rule

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}$$

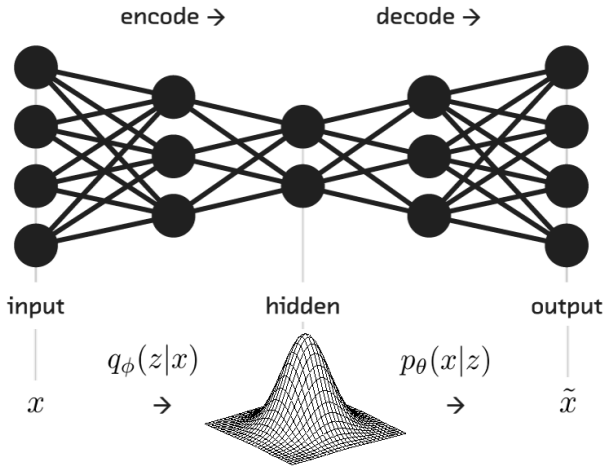
  - but hard to estimate quantities in a high-dimensional space

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In 2nd International Conference on Learning Representations, ICLR 2014

Variational Autoencoders, cont'd

- The idea is to approximate $p(\boldsymbol{z}|\boldsymbol{x})$ with a model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$
  - Similarly, approximate $p(\boldsymbol{x}|\boldsymbol{z})$ with a model $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$
    - Where models are neural nets, of course
  - And train both parts so that they become "compatible"
    - Intuitively, want $dec(enc(x)) = x$

Source:
https://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in-prose-and-code.html



Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In 2nd International Conference on Learning Representations, ICLR 2014

Variational Autoencoders, cont'd

Rensselaer

- As we are approximating $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ with $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, we minimize

$$D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_i)\right) = \mathbb{E}\left[\frac{\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)}{\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_i)}\right]$$
$$= \mathbb{E}\left[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)\right] - \mathbb{E}\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i)}\right]$$
$$= \mathbb{E}\left[\log\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] - \mathbb{E}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z})] + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

- So finally,

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_i)\right) - D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)||p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) + \mathbb{E}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z})]$$
$$= D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_i)\right) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i)$$

  - where $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i)$ is called the *variational lower bound*
    - The other term is always positive
  - maximizing $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i)$ maximizes the log-likelihood!

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In 2nd International Conference on Learning Representations, ICLR 2014

Variational Autoencoders, cont'd

- Look at the variational lower bound

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)||p(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z}))]$$

- First, we need to assume something about $p(\boldsymbol{z})$
  - Typically, we assume $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$
  - So, we need to choose $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)$ so as to minimize the distance to a Gaussian "prior"
- Similarly, we'll assume $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)$ is Gaussian also, i.e.,

$$\boldsymbol{z} \sim \mathcal{N}\left(\mu_{\boldsymbol{\phi}}(\boldsymbol{x}_i), \sigma_{\boldsymbol{\phi}}(\boldsymbol{x}_i)\right)$$

  - For each $\boldsymbol{x}_i$, sample $\epsilon_i \sim \mathcal{N}(0,1)$ and write

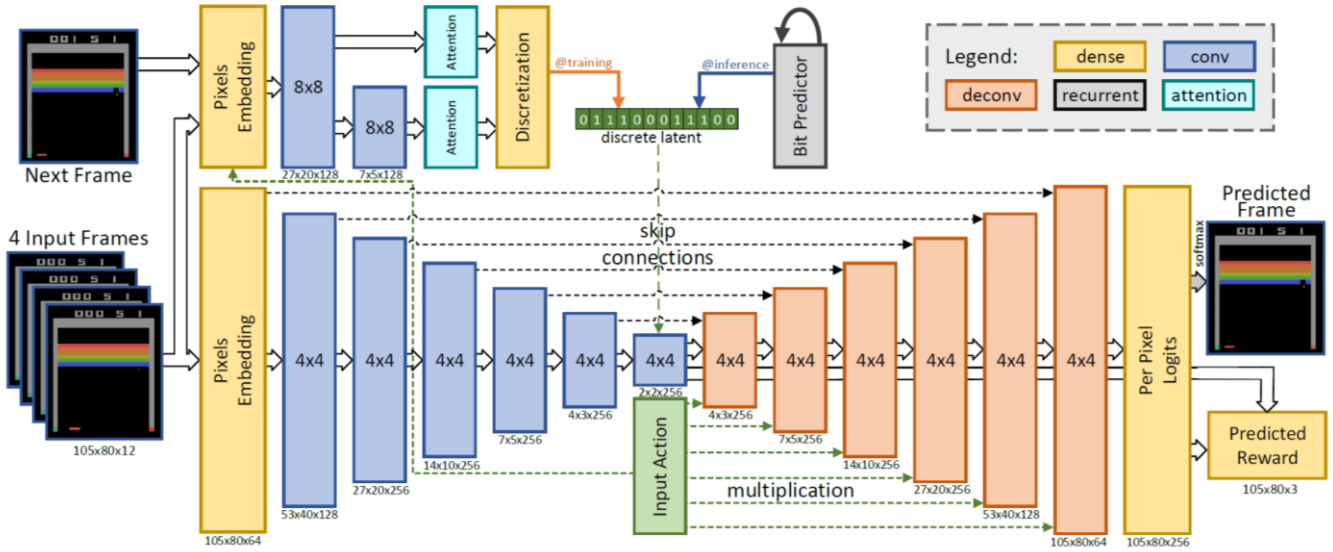
$$\boldsymbol{z}_i = \mu_{\boldsymbol{\phi}}(\boldsymbol{x}_i) + \sigma_{\boldsymbol{\phi}}(\boldsymbol{x}_i) \cdot \epsilon_i$$

  - Can now calculate gradient of $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}_i)$ for $(\boldsymbol{x}_i, \epsilon_i)$

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In 2nd International Conference on Learning Representations, ICLR 2014

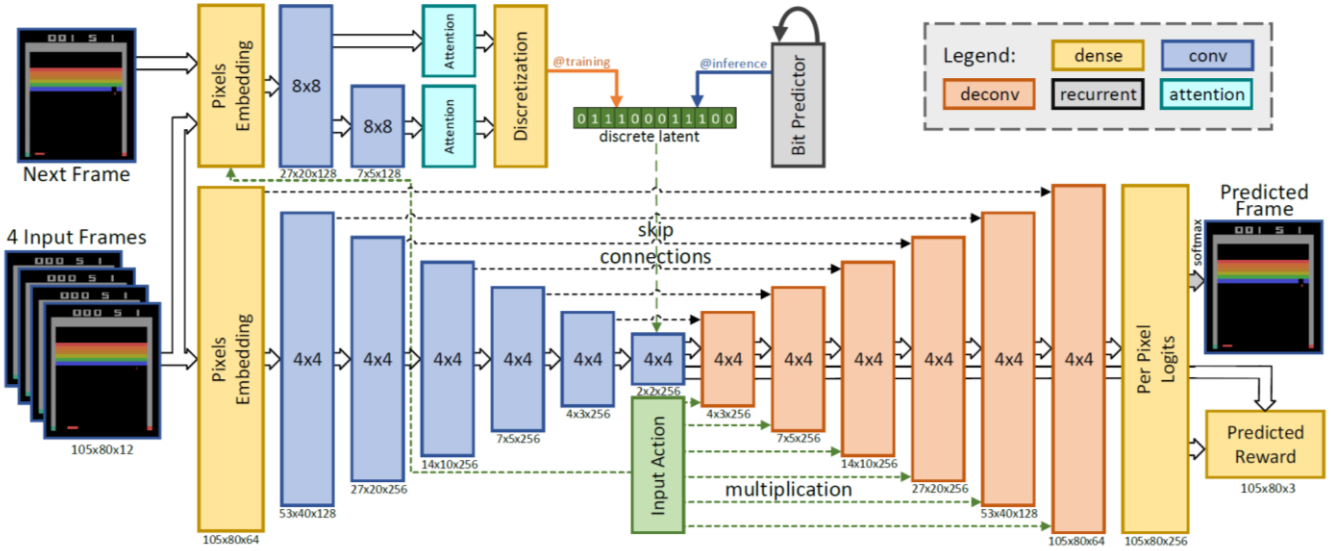
Full World Model

- Bottom part is the deterministic autoencoder
- Top left is a VAE
  - Ideally, noise is sampled randomly at test time
  - Authors claims using raw noise embeddings is unstable and varies across games



Full World Model, cont'd

- Authors claims using raw noise embeddings is unstable and varies across games
  - Instead discretize embedding into bits
  - And train a recurrent inference network to predict next bit
  - At test time, only use inference network
    - Not sure why this is stochastic anymore



Policy Training

Rensselaer

- Alternate between sampling from env and env'
- Then train using a standard model-free algorithm
  - Authors used PPO
- One challenge is that env' cannot be used for very long trajectories since the noise will compound over time
  - Cap trajectory length to 50 steps
- In first loop, train world model for 45K steps
  - Then, use 15K real world steps until reaching target of 100K
- Agents interact with env' for a total of 15.2M interactions

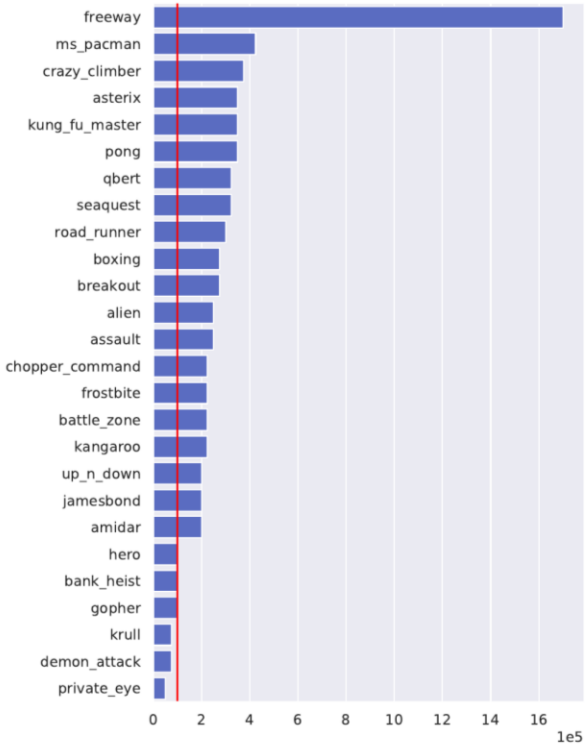
Algorithm 1: Pseudocode for SimPLe

Initialize policy π
Initialize model parameters θ of env'
Initialize empty set $\mathbf{D}$
while not done **do**
  ▷ collect observations from real env.
  $\mathbf{D} \leftarrow \mathbf{D} \cup \text{COLLECT}(env, \pi)$
  ▷ update model using collected data.
  $\theta \leftarrow \text{TRAIN\_SUPERVISED}(env', \mathbf{D})$
  ▷ update policy using world model.
  $\pi \leftarrow \text{TRAIN\_RL}(\pi, env')$
end while

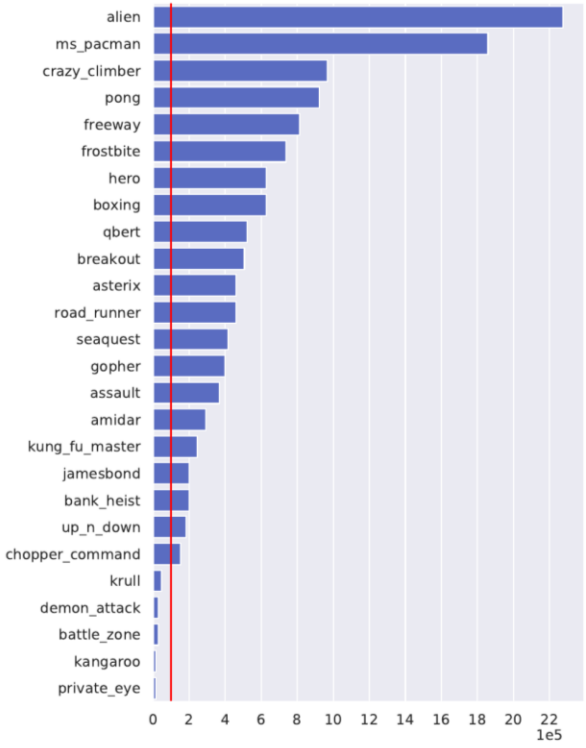
Experiments

- Proposed method drastically outperforms other approaches
  - Graph shows number of steps needed to reach SimPLe's performance

Rainbow (model-based)

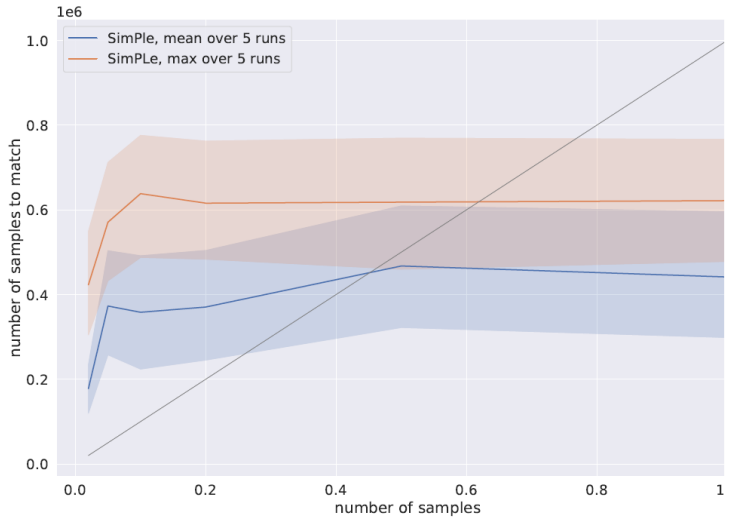


PPO (model-free)



Experiments, cont'd

- Model-based methods eventually plateau with more steps
  - Why?
  - Probably because model isn't good enough
  - But useful as initialization for model-free methods (right)



(a) Model based (SimPLe)

(b) Model based + model free (SimPLe + PPO)

Experiments, cont'd

Rensselaer

- Peak SimPLe performance still well below model-free methods given enough data

Game	SimPLe		PPO_100k		PPO_500k		PPO_1m		Rainbow_100k		Rainbow_500k		Rainbow_1m		random	human
Alien	616.9	(252.2)	291.0	(40.3)	269.0	(203.4)	362.0	(102.0)	290.6	(14.8)	828.6	(54.2)	945.0	(85.0)	184.8	7128.0
Amidar	74.3	(28.3)	56.5	(20.8)	93.2	(36.7)	123.8	(19.7)	20.8	(2.3)	194.0	(34.9)	275.8	(66.7)	11.8	1720.0
Assault	527.2	(112.3)	424.2	(55.8)	552.3	(110.4)	1134.4	(798.8)	300.3	(14.6)	1041.5	(92.1)	1581.8	(207.8)	233.7	742.0
Asterix	1128.3	(211.8)	385.0	(104.4)	1085.0	(354.8)	2185.0	(931.6)	285.7	(9.3)	1702.7	(162.8)	2151.6	(202.6)	248.8	8503.0
Asteroids	793.6	(182.2)	1134.0	(326.9)	1053.0	(433.3)	1251.0	(377.9)	912.3	(62.7)	895.9	(82.0)	1071.5	(91.7)	649.0	47389.0
Atlantis	20992.5	(11062.0)	34316.7	(5703.8)	4836416.7	(6218247.3)	-	(-)	17881.8	(617.6)	79541.0	(25393.4)	848800.0	(37533.1)	16492.0	29028.0
BankHeist	34.2	(29.2)	16.0	(12.4)	641.0	(352.8)	856.0	(376.7)	34.5	(2.0)	727.3	(198.3)	1053.3	(22.9)	15.0	753.0
BattleZone	4031.2	(1156.1)	5300.0	(3655.1)	14400.0	(6476.1)	19000.0	(4571.7)	3363.5	(523.8)	19507.1	(3193.3)	22391.4	(7708.9)	2895.0	37188.0
BeamRider	621.6	(79.8)	563.6	(189.4)	497.6	(103.5)	684.0	(168.8)	365.6	(29.8)	5890.0	(525.6)	6945.3	(1390.8)	372.1	16926.0
Bowling	30.0	(5.8)	17.7	(11.2)	28.5	(3.4)	35.8	(6.2)	24.7	(0.8)	31.0	(1.9)	30.6	(6.2)	24.2	161.0
Boxing	7.8	(10.1)	-3.9	(6.4)	3.5	(3.5)	19.6	(20.9)	0.9	(1.7)	58.2	(16.5)	80.3	(5.6)	0.3	12.0
Breakout	16.4	(6.2)	5.9	(3.3)	66.1	(114.3)	128.0	(153.3)	3.3	(0.1)	26.7	(2.4)	38.7	(3.4)	0.9	30.0
ChopperCommand	979.4	(172.7)	730.0	(199.0)	860.0	(285.3)	970.0	(201.5)	776.6	(59.0)	1765.2	(280.7)	2474.0	(504.5)	671.0	7388.0
CrazyClimber	62583.6	(16856.8)	18400.0	(5275.1)	33420.0	(3628.3)	58000.0	(16994.6)	12558.3	(674.6)	75655.1	(9439.6)	97088.1	(9975.4)	7339.5	35829.0
DemonAttack	208.1	(56.8)	192.5	(83.1)	216.5	(96.2)	241.0	(135.0)	431.6	(79.5)	3642.1	(478.2)	5478.6	(297.9)	140.0	1971.0
FishingDerby	-90.7	(5.3)	-95.6	(4.3)	-87.2	(5.3)	-88.8	(4.0)	-91.1	(2.1)	-66.7	(6.0)	-23.2	(22.3)	-93.6	-39.0
Freeway	16.7	(15.7)	8.0	(9.8)	14.0	(11.5)	20.8	(11.1)	0.1	(0.1)	12.6	(15.4)	13.0	(15.9)	0.0	30.0
Frostbite	236.9	(31.5)	174.0	(40.7)	214.0	(10.2)	229.0	(20.6)	140.1	(2.7)	1386.1	(321.7)	2972.3	(284.9)	74.0	-
Gopher	596.8	(183.5)	246.0	(103.3)	560.0	(118.8)	696.0	(279.3)	748.3	(105.4)	1640.5	(105.6)	1905.0	(211.1)	245.9	2412.0
Gravitar	173.4	(54.7)	235.0	(197.2)	235.0	(134.7)	325.0	(85.1)	231.4	(50.7)	214.9	(27.6)	260.0	(22.7)	227.2	3351.0
Hero	2656.6	(483.1)	569.0	(1100.9)	1824.0	(1461.2)	3719.0	(1306.0)	2676.3	(93.7)	10664.3	(1060.5)	13295.5	(261.2)	224.6	30826.0
IceHockey	-11.6	(2.5)	-10.0	(2.1)	-6.6	(1.6)	-5.3	(1.7)	-9.5	(0.8)	-9.7	(0.8)	-6.5	(0.5)	-9.7	1.0
Jamesbond	100.5	(36.8)	65.0	(46.4)	255.0	(101.7)	310.0	(129.0)	61.7	(8.8)	429.7	(27.9)	692.6	(316.2)	29.2	303.0
Kangaroo	51.2	(17.8)	140.0	(102.0)	340.0	(407.9)	840.0	(806.5)	38.7	(9.3)	970.9	(501.9)	4084.6	(1954.1)	42.0	3035.0
Krull	2204.8	(776.5)	3750.4	(3071.9)	3056.1	(1155.5)	5061.8	(1333.4)	2978.8	(148.4)	4139.4	(336.2)	4971.1	(360.3)	1543.3	2666.0
KungFuMaster	14862.5	(4031.6)	4820.0	(983.2)	17370.0	(10707.6)	13780.0	(3971.6)	1019.4	(149.6)	19346.1	(3274.4)	21258.6	(3210.2)	616.5	22736.0
MsPacman	1480.0	(288.2)	496.0	(379.8)	306.0	(70.2)	594.0	(247.9)	364.3	(20.4)	1558.0	(248.9)	1881.4	(112.0)	235.2	6952.0
NameThisGame	2420.7	(289.4)	2225.0	(423.7)	2106.0	(898.8)	2311.0	(547.6)	2368.2	(318.3)	4886.5	(583.1)	4454.2	(338.3)	2136.8	8049.0
Pong	12.8	(17.2)	-20.5	(0.6)	-8.6	(14.9)	14.7	(5.1)	-19.5	(0.2)	19.9	(0.4)	20.6	(0.2)	-20.4	15.0
PrivateEye	35.0	(60.2)	10.0	(20.0)	20.0	(40.0)	20.0	(40.0)	42.1	(53.8)	-6.2	(89.8)	2336.7	(4732.6)	26.6	69571.0
Qbert	1288.8	(1677.9)	362.5	(117.8)	757.5	(78.9)	2675.0	(1701.1)	235.6	(12.9)	4241.7	(193.1)	8885.2	(1690.9)	166.1	13455.0
Riverraid	1957.8	(758.1)	1398.0	(513.8)	2865.0	(327.1)	2887.0	(807.0)	1904.2	(44.2)	5068.6	(292.6)	7018.9	(334.2)	1451.0	17118.0
RoadRunner	5640.6	(3936.6)	1430.0	(760.0)	5750.0	(5259.9)	8930.0	(4304.0)	524.1	(147.5)	18415.4	(5280.0)	31379.7	(3225.8)	0.0	7845.0
Seaquest	683.3	(171.2)	370.0	(103.3)	692.0	(48.3)	882.0	(122.7)	206.3	(17.1)	1558.7	(221.2)	3279.9	(683.9)	61.1	42055.0
UpNDown	3350.3	(3540.0)	2874.0	(1105.8)	12126.0	(1389.5)	13777.0	(6766.3)	1346.3	(95.1)	6120.7	(356.8)	8010.9	(907.0)	488.4	11693.0
YarsRevenge	5664.3	(1870.5)	5182.0	(1209.3)	8064.8	(2859.8)	9495.0	(2638.3)	3649.0	(168.6)	7005.7	(394.2)	8225.1	(957.9)	3121.2	54577.0

Summary

Rensselaer

- Offline RL is a very challenging and somewhat ill-defined field
- Ideally, would like to learn a policy that is better than the behavior policy used to collect the data
  - Can't learn actions that are not present in the dataset
  - Optimal policy may require a sequence of optimal actions (e.g., in a game)
    - this sequence may not appear at all in the training data
- Unclear how to compare methods
  - Limit number of interactions with real world or fix a dataset that is somehow sufficient to learn optimal policy
- Offline RL can be particularly useful in safety-critical domains
  - Enable safe exploration in the modeled environment