Exploration in Reinforcement Learning

John Canny

Spring 2019

Lecture 22 of CS182/282A: Designing, Visualizing and Understanding Deep Neural Networks

Some slides borrowed from S. Levine et al. "Deep Reinforcement Learning"

Last Time: Q Functions

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)}_{\text{“reward to go” } \hat{Q}_{i,t}}$$

Define an expected value estimator Q:

$$Q(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[\gamma^{t'-t} r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t]$$

True expected reward-to-go

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\, Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

Allows us to reward individual ***actions***. REINFORCE only improves entire trajectories.

Last Time: Advantage functions

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[\gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t]$$

True expected reward to go
Total reward from taking $\mathbf{a}_t$ in $\mathbf{s}_t$

Value function $V^{\pi}(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\,[Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)]$ = total reward from $\mathbf{s}_t$

Advantage Function $A^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) - V^{\pi}(\mathbf{s}_t)$ = how much better $\mathbf{a}_t$ is.

Advantage functions are typically ***sparse*** functions of state.

i.e. for many environments, most states have action advantages = 0. Addresses the temporal credit assignment problem.

Last Time: Actor-critic algorithms

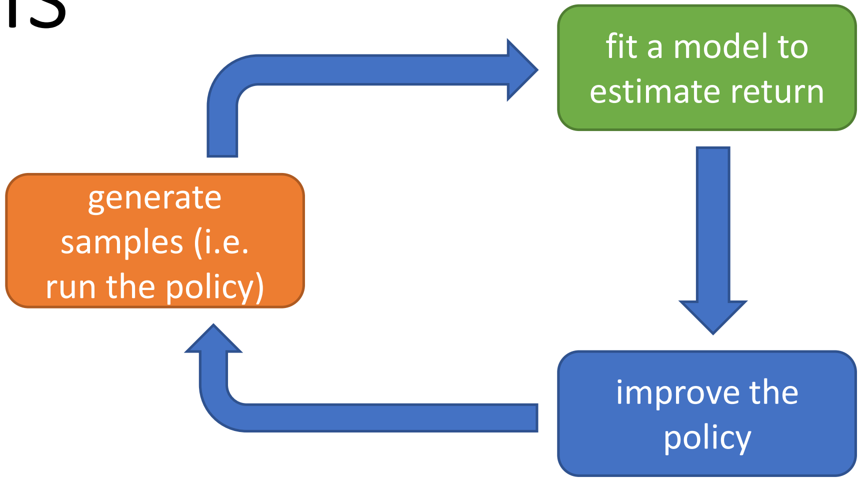
batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

online actor-critic algorithm:

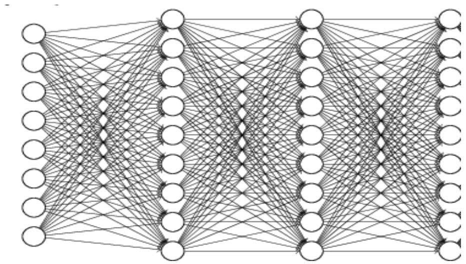
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

fit $\hat{V}^\pi$



$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$\mathbf{s}$



$\hat{V}^\pi(\mathbf{s})$

parameters ϕ

Last Time: "Classic" deep Q-learning algorithm (DQN)

Save to and sample from a replay buffer

"classic" deep Q-learning algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ': copy ϕ every N steps

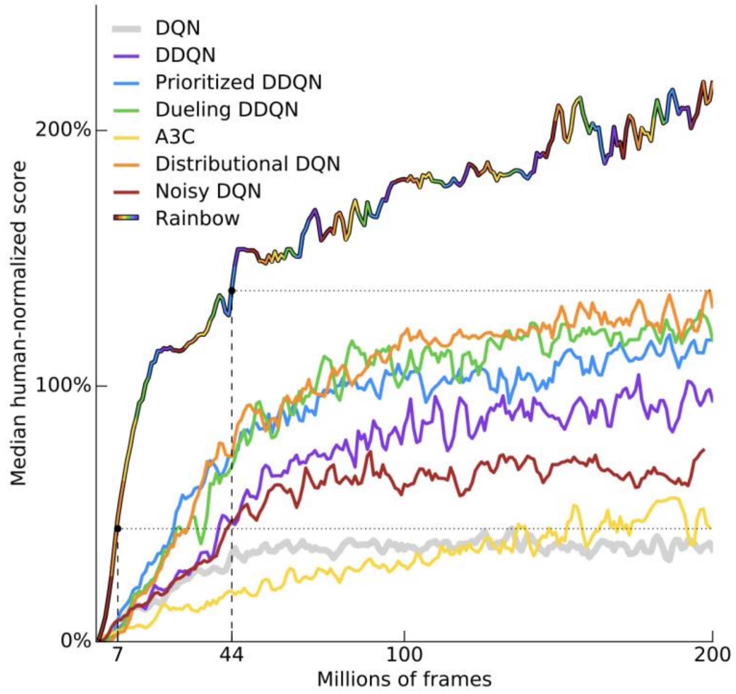
$K = 1$

Use a target network, updated periodically

Mnih et al. '13

Last Time: Upgrades to (D)DQN

- Prioritized Experience Replay [1]
  - Rather than sample uniformly at random, sample according to TD error!
- Dueling Architectures [2]
  - Separate state value and advantage value streams.
- Rainbow [3]
  - Combine a bunch of improvements together!



[1]: Schaul et al., *Prioritized Experience Replay*, ICLR 2016.
[2]: Wang et al., *Dueling Network Architectures for Deep Reinforcement Learning*, ICML 2016.
[3]: Hessel et al., *Rainbow: Combining Improvements in Deep Reinforcement Learning*, AAAI 2018.

Course Logistics

- HW4 is due Friday.
- Make sure you completed (2^{nd}) project checkin.
- Final Project Poster due Saturday 5/4
- Poster Session is Tuesday May 7^{th}, 2-4pm.
- Course survey is open now, please fill it out.

This Time: Exploration

- Exploration vs. Exploitation
- Exploration Methods:
  - Optimistic exploration
  - Posterior sampling
  - Curiosity-driven exploration

- Information Bottleneck Methods
  - General approach
  - InfoBots for Reinforcement Learning
  - Mapping and exploration

Exploration: What's the problem?

Policy Gradients: sample trajectories ***using $\boldsymbol{\pi_\theta}$***, then compute gradients weighted by reward or advantage.

Value-Based methods: update Q- or Value-function estimates ***using states visited by $\boldsymbol{\pi_\theta}$***, then compute gradients weighted by reward or advantage.

So unless π_θ already visits a state $\mathbf{s}$, optimization is unlikely to explore it.

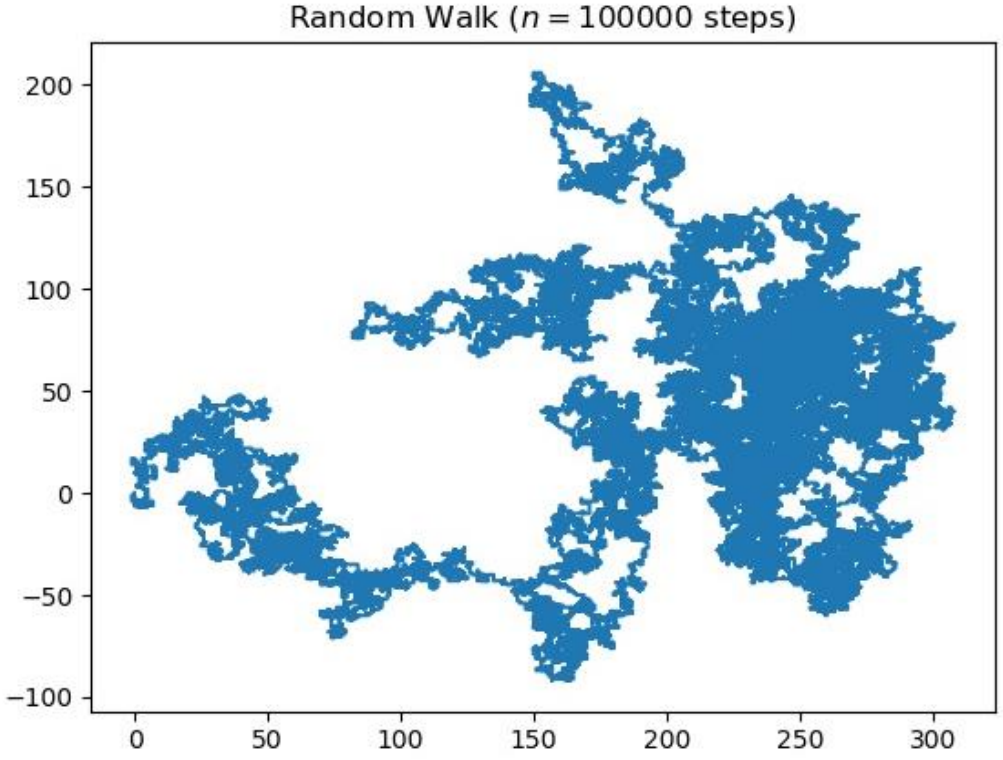
For Value-based methods, we defined **epsilon-greedy** methods as:

$$\pi'_\theta(\mathbf{a}|\mathbf{s}) = (1-\epsilon)\pi_\theta(\mathbf{a}|\mathbf{s}) + \epsilon\ Uniform(\mathbf{a})$$

With $\epsilon = 1$, this is a random walk strategy.

Exploration: What’s the problem?

Random walks are a very inefficient way to explore.
The maximum radius explored in time T is proportional to $\sqrt{T}$.



Better exploration strategies require us to remember what we did…

What's the problem?

this is easy (mostly) with epsilon-greedy

Why?

this is impossible

Montezuma's revenge

- Getting key = reward
- Opening door = reward
- Getting killed by skull = nothing (is it good? bad?)
- Finishing the game only weakly correlates with rewarding events
- We know what to do because we **understand** what these sprites mean!

Exploration and exploitation examples

- Restaurant selection
  - Exploitation: go to your favorite restaurant
  - Exploration: try a new restaurant
- Online ad placement
  - Exploitation: show the most successful advertisement
  - Exploration: show a different random advertisement
- Oil drilling
  - Exploitation: drill at the best known location
  - Exploration: drill at a new location

Examples from D. Silver lecture notes: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/XX.pdf

Classes of exploration methods in deep RL

- Optimistic exploration:
  - new state = good state
  - requires estimating state visitation frequencies or novelty
  - typically realized by means of exploration bonuses

- Thompson sampling style algorithms:
  - learn distribution over Q-functions or policies
  - sample and act according to sample

- Information gain style algorithms
  - reason about information gain from visiting new states

Optimistic exploration in RL

UCB (Upper Confidence Bound) methods for MDPs

Define $N(s)$ as the number of times we have visited state s,
or $N(s, a)$ as the number of times we performed action a in state s.

Add an intrinsic reward or bonus for visiting rarely-visited states:
Use $r^+(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \mathcal{B}\big(N(\mathbf{s})\big)$

Bonus that decreases with $N(\mathbf{s})$

Use $r^+(\mathbf{s}, \mathbf{a})$ instead of $r(\mathbf{s}, \mathbf{a})$ with any model-free algorithm, e.g. $\mathcal{B}\big(N(s)\big) = \sqrt{\frac{2 \ln T}{N(s)}}$

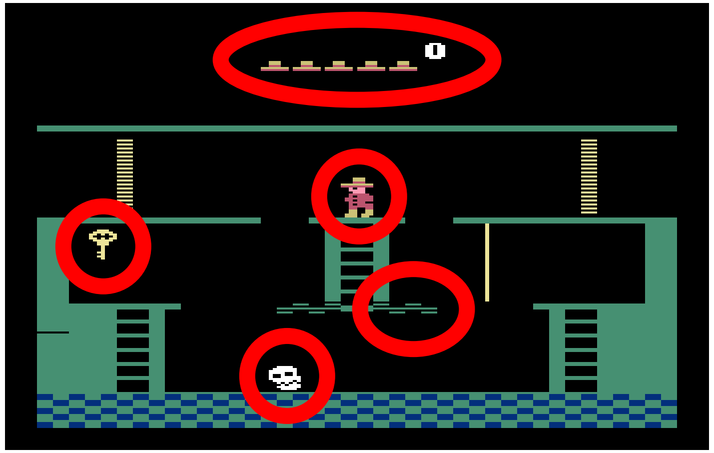
+ simple addition to any RL algorithm

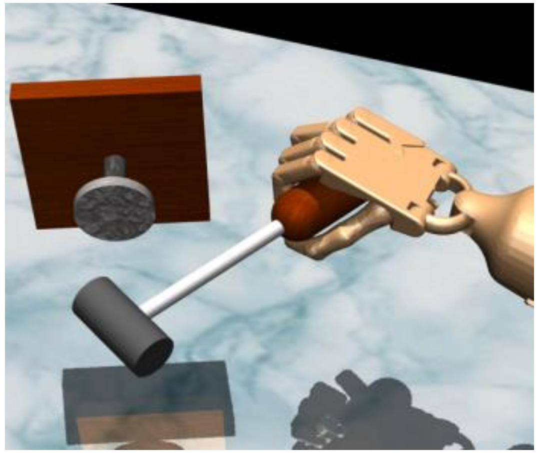
- need to tune bonus weight

The trouble with counts

Use $r^+(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \mathcal{B}(N(\mathbf{s}))$

But wait… what's a count?

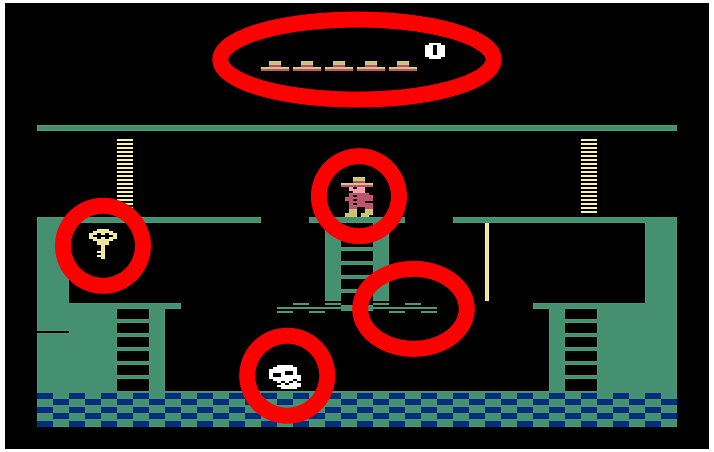


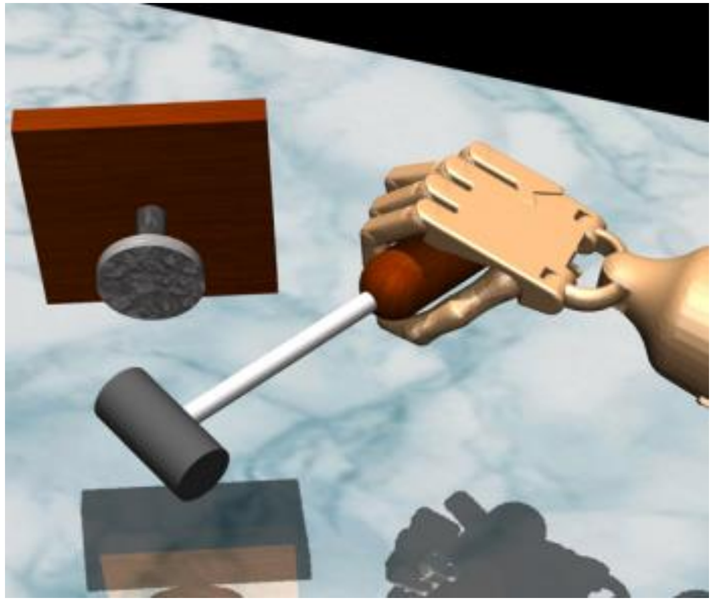


Uh oh… we never see the same thing twice!

But some states are more similar than others

Fitting generative models





Idea: fit a density model $p_\theta(\mathbf{s})$ or $p_\theta(\mathbf{s},\mathbf{a})$.

$p_\theta(\mathbf{s})$ might be high even for a new $\mathbf{s}$ if $\mathbf{s}$ is similar to previously seen states.

Can we use $p_\theta(\mathbf{s})$ to get a “pseudo-count” ?

For small MDPs, the true probability is:

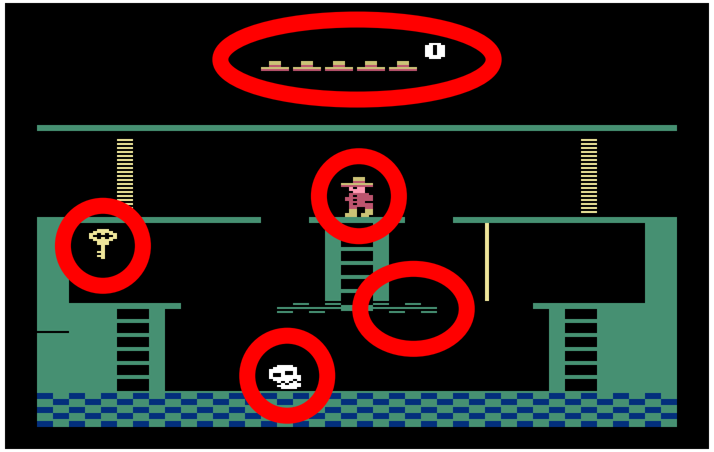
$$\underbrace{p(\mathbf{s})}_{\text{probability/density}} = \frac{\overbrace{N(\mathbf{s})}^{\text{count}}}{\underbrace{n}_{\text{total states visited}}}$$

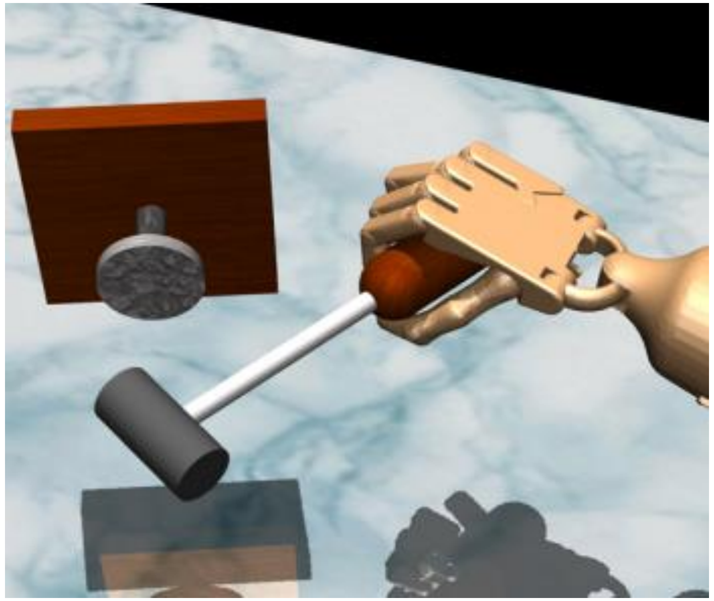
After we see $\mathbf{s}$, we have:

$$p'(\mathbf{s}) = \frac{N(\mathbf{s})+1}{n+1}$$

Can we get $p_\theta(\mathbf{s})$ and $p_{\theta'}(\mathbf{s})$ to obey these equations?

Exploring with pseudo-counts





fit model $p_\theta(\mathbf{s})$ to all states $\mathcal{D}$ seen so far

take a step i and observe $\mathbf{s}_i$

fit new model $p_{\theta'}(\mathbf{s})$ to $\mathcal{D} \cup \mathbf{s}_i$

use $p_\theta(\mathbf{s}_i)$ and $p_{\theta'}(\mathbf{s}_i)$ to estimate $\hat{N}(\mathbf{s})$

set $r_i^+ = r_i + \mathcal{B}(\hat{N}(\mathbf{s}))$ ← "pseudo-count"

how to get $\hat{N}(\mathbf{s})$? use the equations

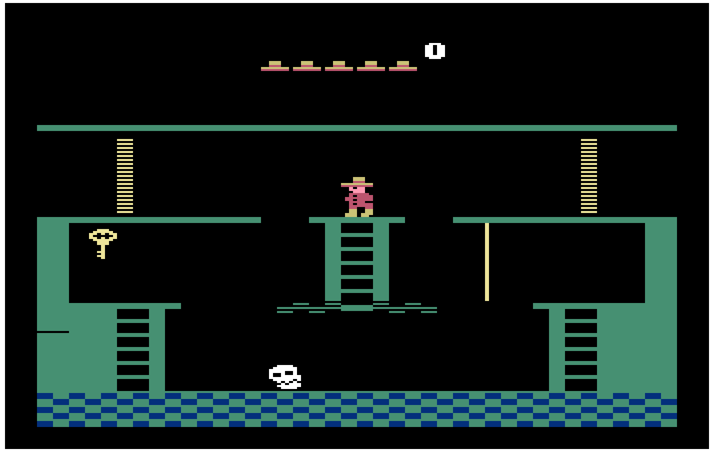
$$p_\theta(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)}{\hat{n}} \qquad p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i) + 1}{\hat{n} + 1}$$

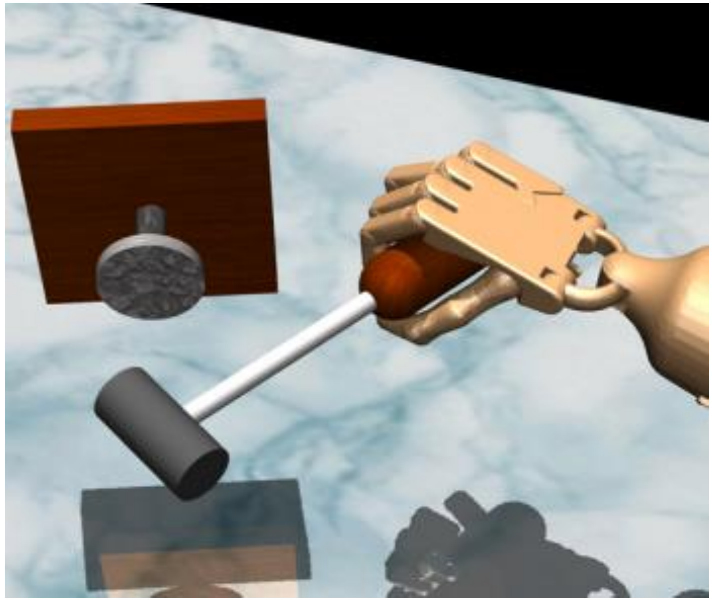
two equations and two unknowns!

$$\hat{N}(\mathbf{s}_i) = \hat{n} p_\theta(\mathbf{s}_i) \qquad \hat{n} = \frac{1 - p_{\theta'}(\mathbf{s}_i)}{p_{\theta'}(\mathbf{s}_i) - p_\theta(\mathbf{s}_i)} p_\theta(\mathbf{s}_i)$$

Bellemare et al. "Unifying Count-Based Exploration…"

What kind of model to use?



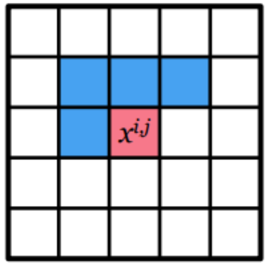


$$p_\theta(\mathbf{s})$$

need to be able to output densities, but doesn't necessarily need to produce great samples

opposite considerations from many popular generative models in the literature (e.g., GANs)

Bellemare et al.: "CTS" model:
something like Pixel CNN but condition each pixel on its top-left neighborhood



Other models: stochastic neural networks, compression length, EX2

What kind of bonus to use?

Lots of functions in the literature, inspired by optimal methods for bandits or small MDPs

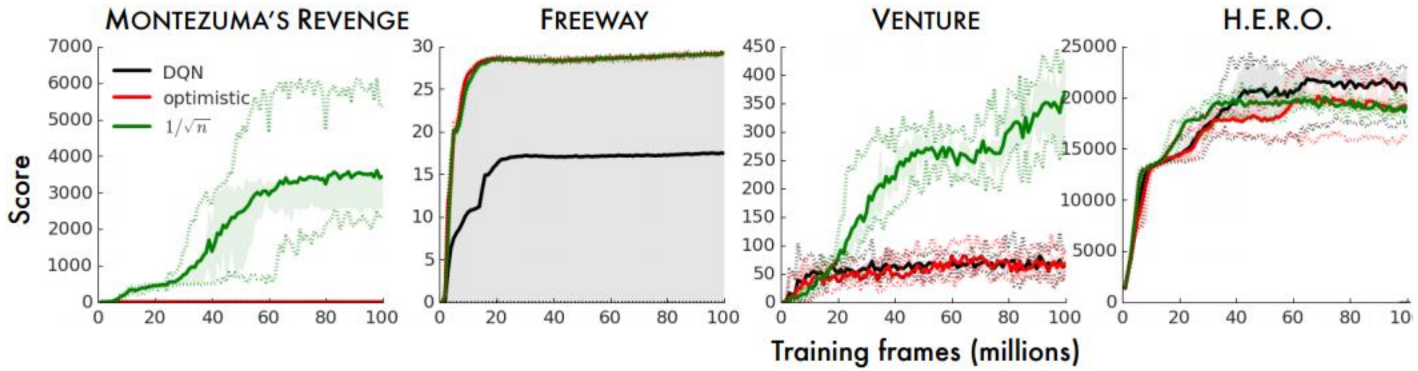
UCB: $\mathcal{B}(N(s)) = \sqrt{\frac{2 \ln n}{N(s)}}$

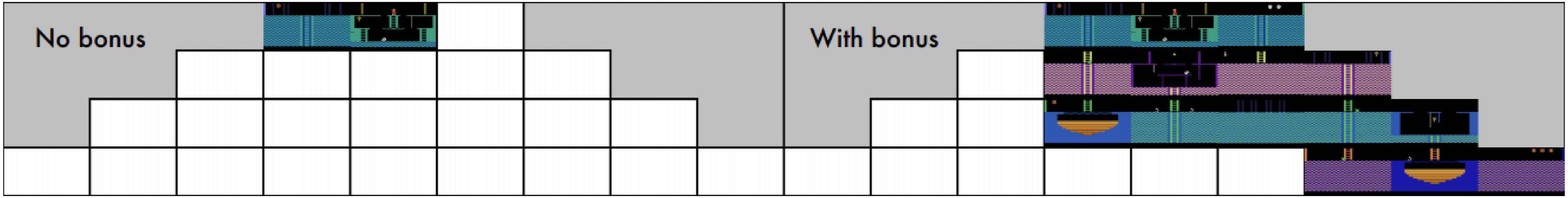
MBIE-EB (Strehl & Littman, 2008): $\mathcal{B}(N(s)) = \sqrt{\frac{1}{N(s)}}$

this is the one used by Bellemare et al. '16

BEB (Kolter & Ng, 2009): $\mathcal{B}(N(s)) = \frac{1}{N(s)}$

Does it work?





Bellemare et al. "Unifying Count-Based Exploration…"

Posterior sampling in deep RL

Represent explicitly our uncertainty in the model parameters θ

Then sample from it (Thompson sampling):

$\theta_1, \ldots, \theta_n \sim \hat{p}(\theta_1, \ldots, \theta_n)$ How do we represent the distribution?

$a = \arg\max_a E_{\theta_a}[r(a)]$

A simple and very general approach is to compute an ensemble of models, and then sample from it:

1. sample Q-function Q from $p(Q)$
2. act according to Q for one episode
3. update $p(Q)$

since Q-learning is off-policy, we don't care which Q-function was used to collect data

Osband et al. "Deep Exploration via Bootstrapped DQN"

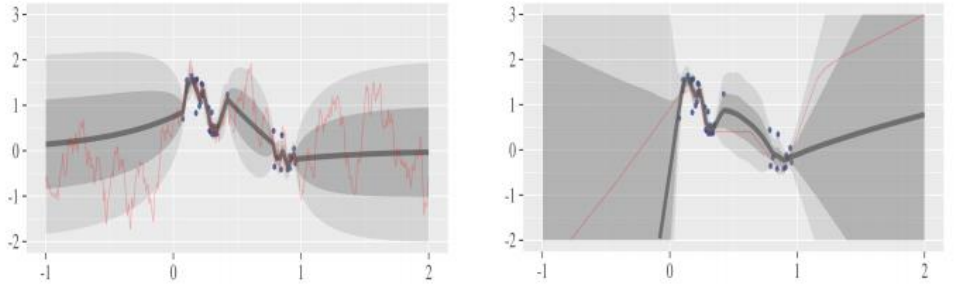
Bootstrap

Given a dataset $\mathcal{D}$, resample with replacement N times to get bootstrap datasets $\mathcal{D}_1, \ldots, \mathcal{D}_N$.

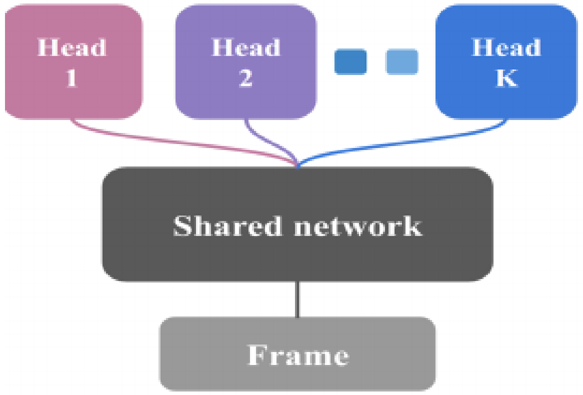
Train a model f_{θ_i} on $\mathcal{D}_i$.

To sample from $p(\theta)$, sample $i \in [1, \ldots, N]$, use f_{θ_i}.

But training N large neural networks is expensive, can we avoid it?



(b) Gaussian process posterior (c) Bootstrapped neural nets

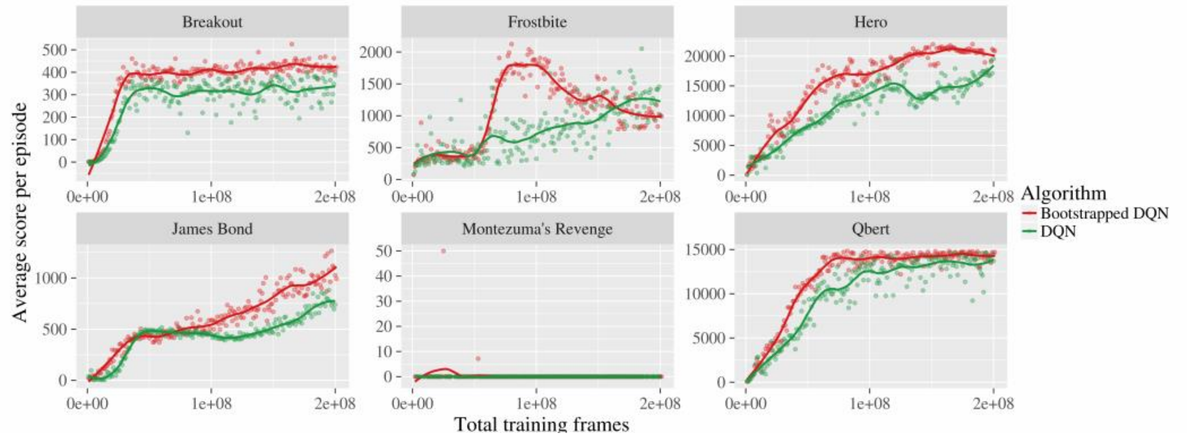


Osband et al. “Deep Exploration via Bootstrapped DQN”

Why does this work?

Exploring with random actions (e.g., epsilon-greedy): random walk pattern, in general $\Omega(N^2)$ steps to visit N states.

Exploring with random Q-functions: commit to a randomized but internally consistent strategy ***for an entire episode***

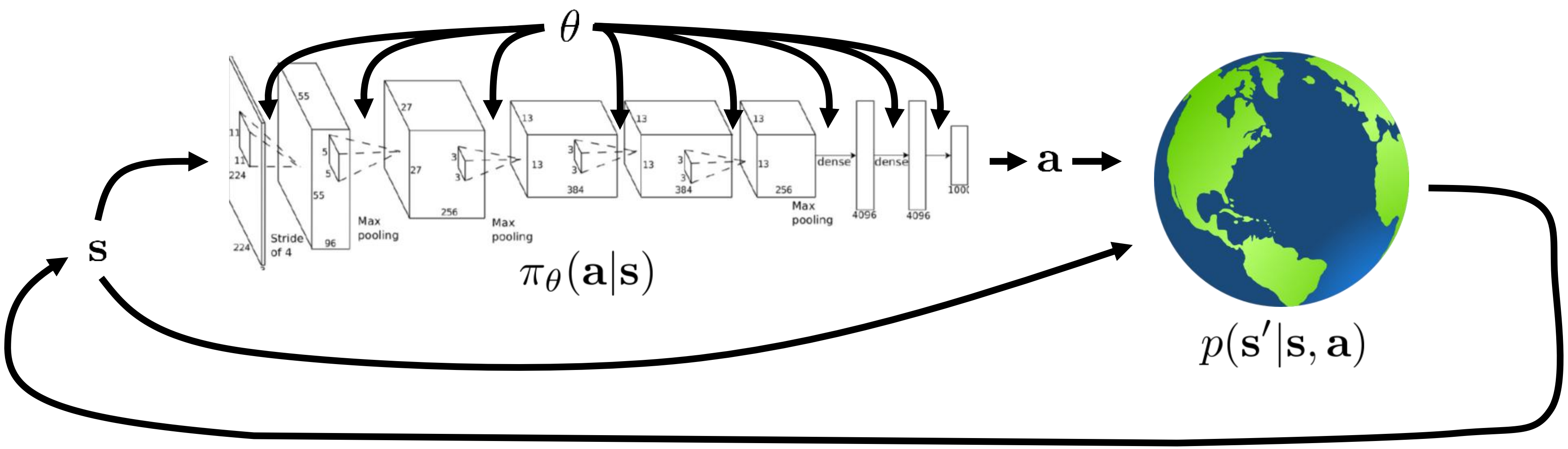


+ no change to original reward function

- very good bonuses often do better

Osband et al. "Deep Exploration via Bootstrapped DQN"

Recap: model-free reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}$$

assume this is unknown
don’t even attempt to learn it

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

What if we knew the transition dynamics?

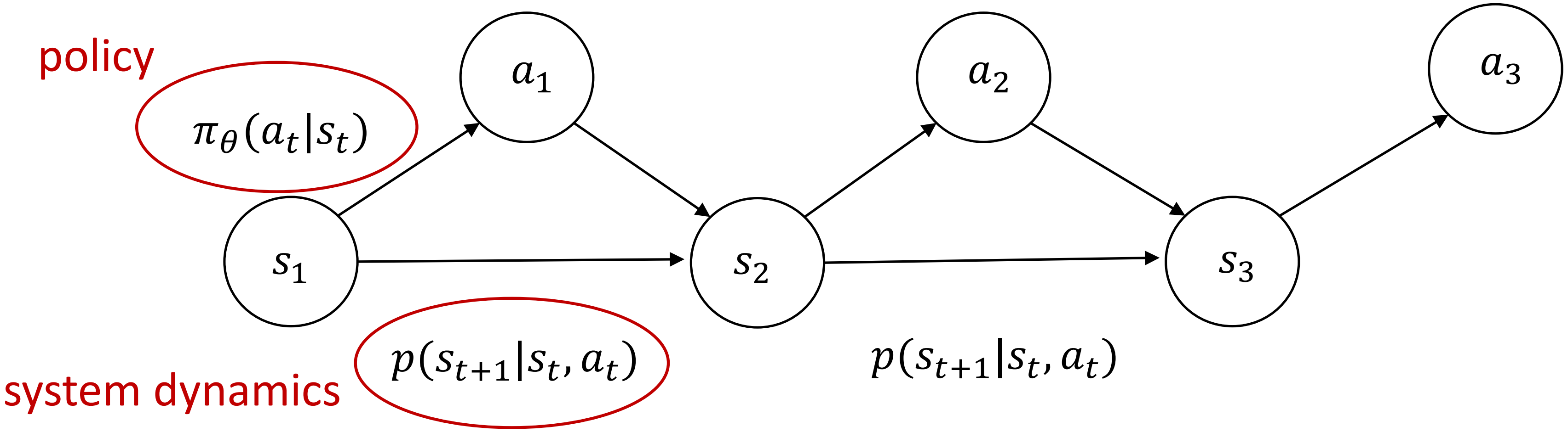
- Often we do know the dynamics
  1. Games (e.g., Go)
  2. Easily modeled systems (e.g., navigating a car)
  3. Simulated environments (e.g., simulated robots, video games)
- Often we can learn the dynamics
  1. System identification – fit unknown parameters of a known model
  2. Learning – fit a general-purpose model to observed transition data

Does knowing the dynamics make things easier?

Often, yes!

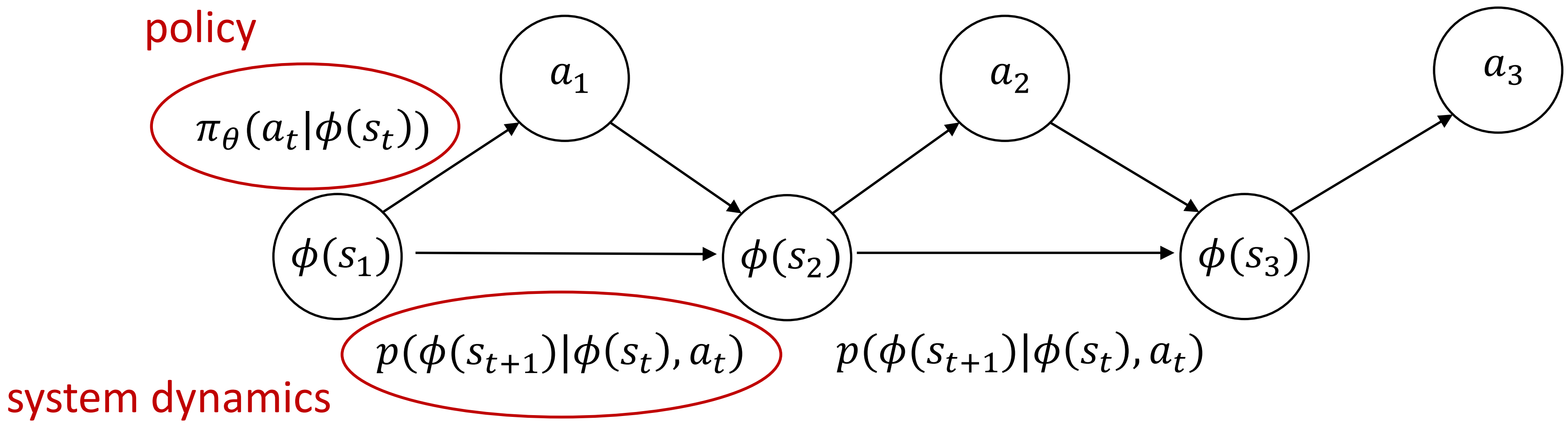
Model-based reinforcement learning

1. Model-based reinforcement learning: learn the transition dynamics, then figure out how to choose actions
2. Advantages:
   1. Avoid making expensive real-world or simulator actions
   2. Possibly differentiate through the dynamics to optimize action choice
3. We then have an optimal control problem



Simplifying Model-based reinforcement learning

1. Computing a complete environment model can be very expensive (has to generate images for vision-based policies)
2. Do we really need the full state?
3. Is there a simplified function of the state $\phi(s)$ that's sufficient for RL?
4. Should be sufficient to predict next state and for policy to choose next action.



Curiosity-driven Exploration by Self-supervised Prediction

Pathak et al. 2017

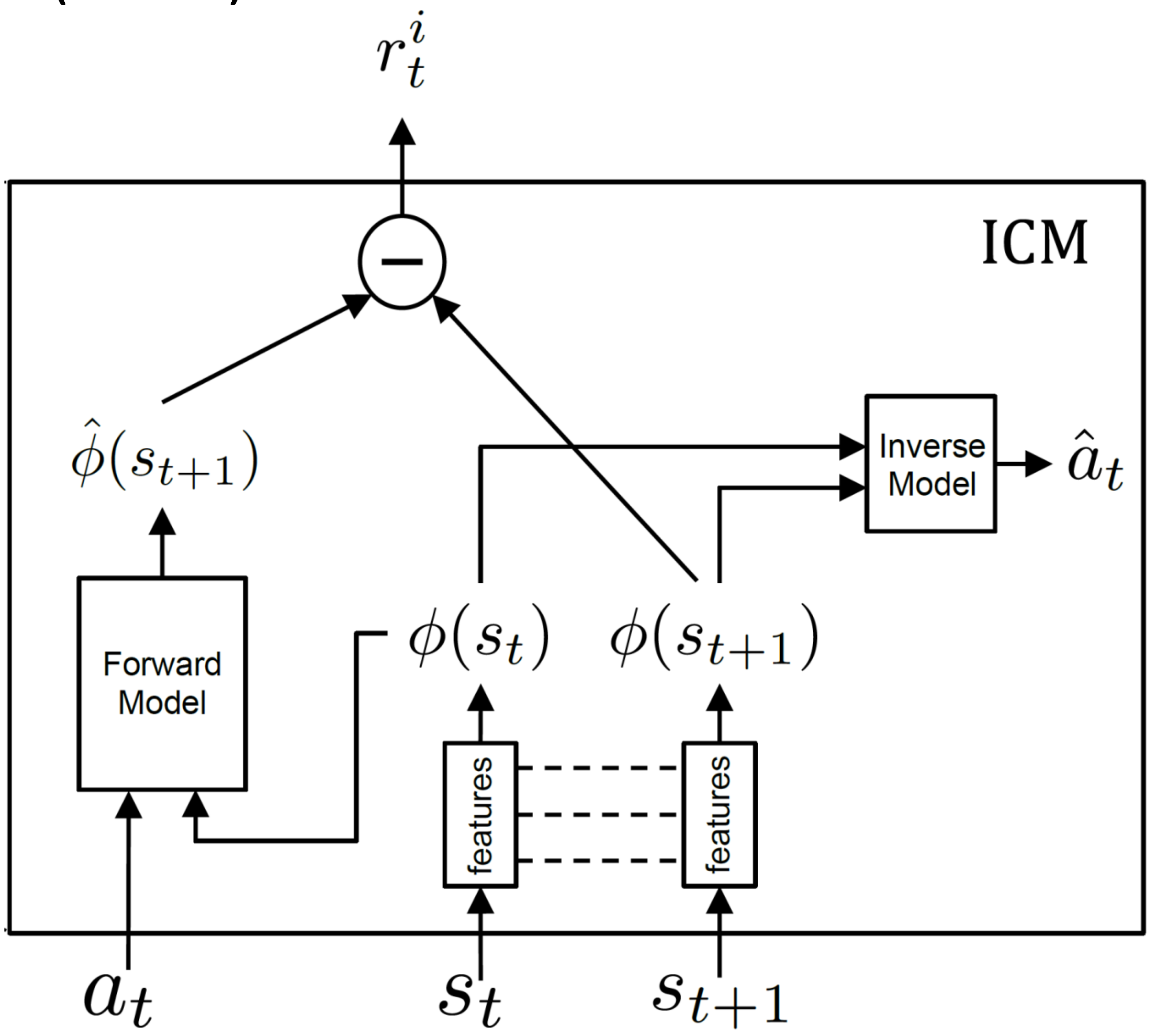
1. Computes a simplified environment model $\phi(s_t)$

2. Uses error in the model's prediction to highlight states that need further exploration.

Intrinsic Curiosity Model (ICM)

Pathak et al. 2017

1. Uses an inverse model to ensure $\phi(s_t)$ is sufficient for action selection.
2. Estimates a forward model for next state.
3. The forward model error is the curiosity signal.

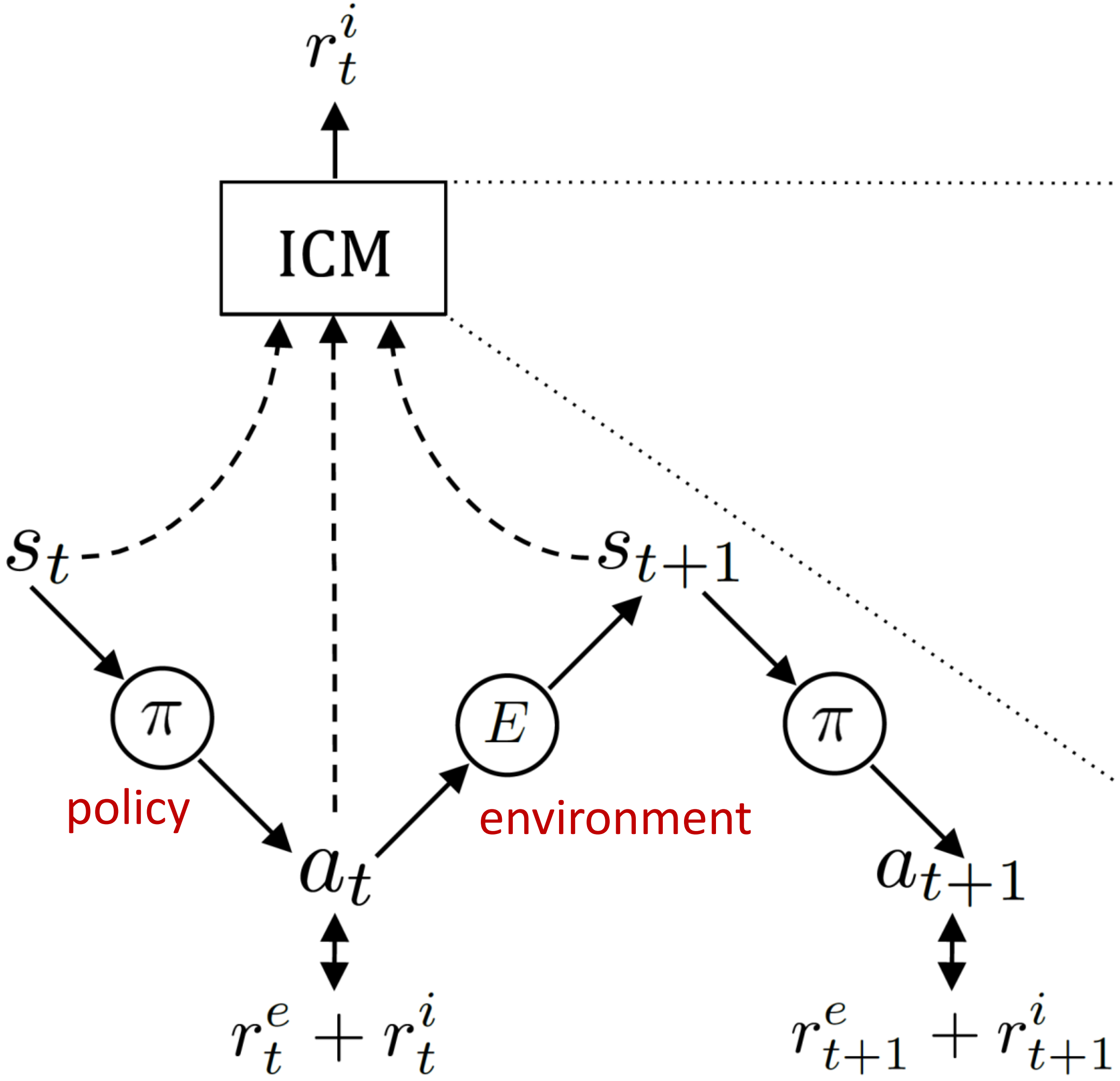


Curiosity-Driven Exploration

Pathak et al. 2017

1. The ICM provides an intrinsic reward signal r_t^i.

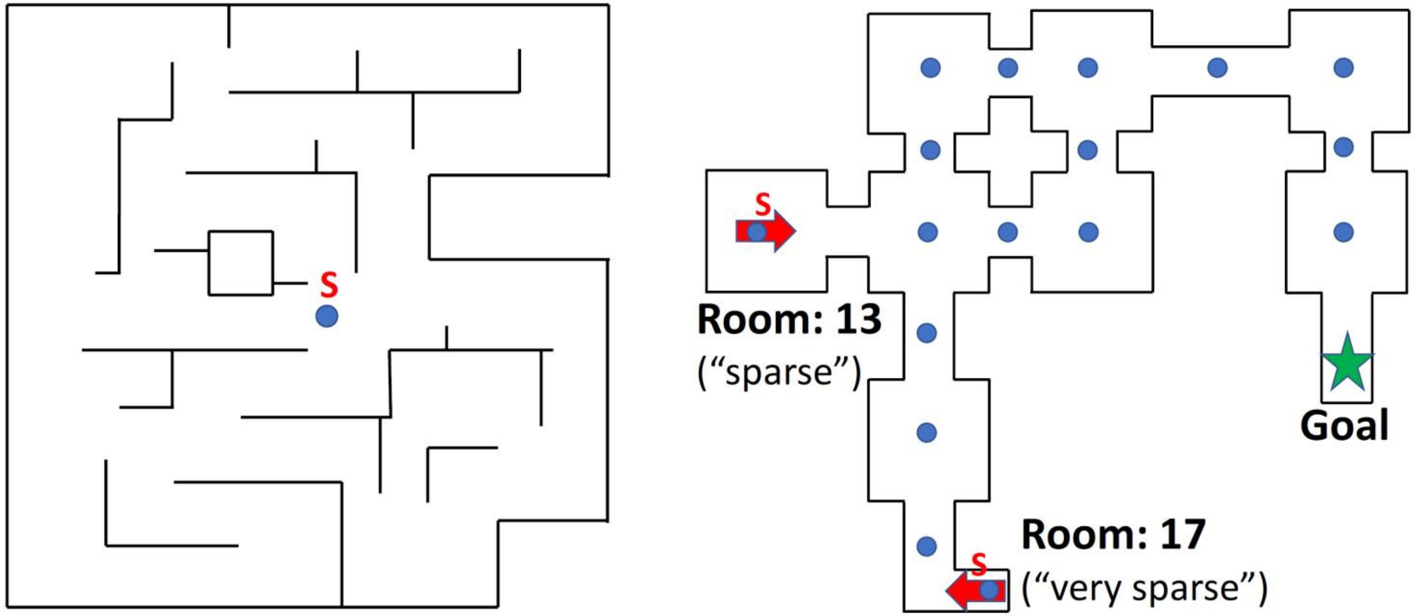
2. The policy can be trained by any RL method using the combined reward.



Curiosity-Driven Exploration

Pathak et al. 2017

Environments: Vizdoom



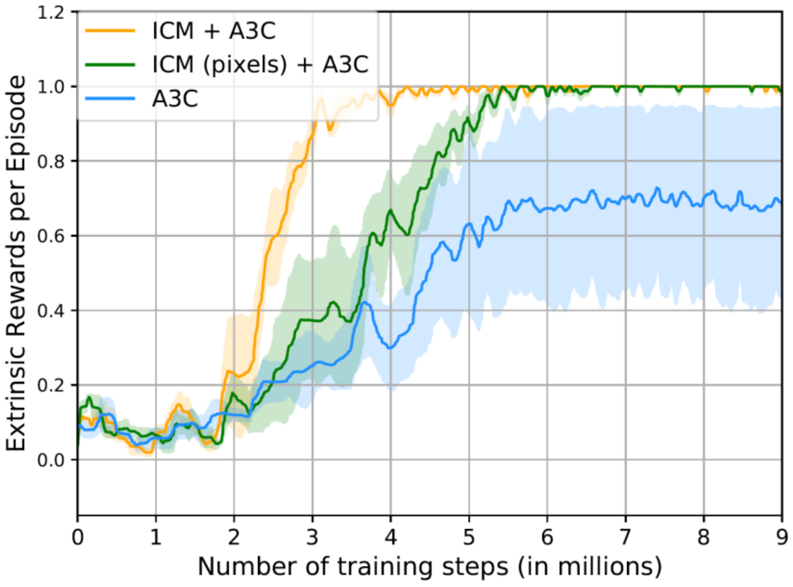
(a) Train Map Scenario (b) Test Map Scenario

Model is trained initially without goal reward (left). Agent is randomly started on one of the blue dots in the "dense" case (right).

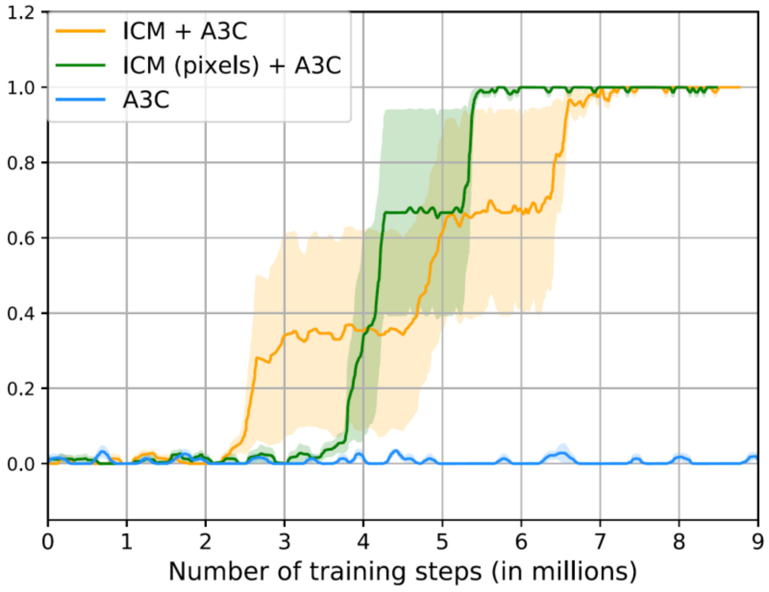
Pathak et al. 2017

Curiosity-Driven Exploration

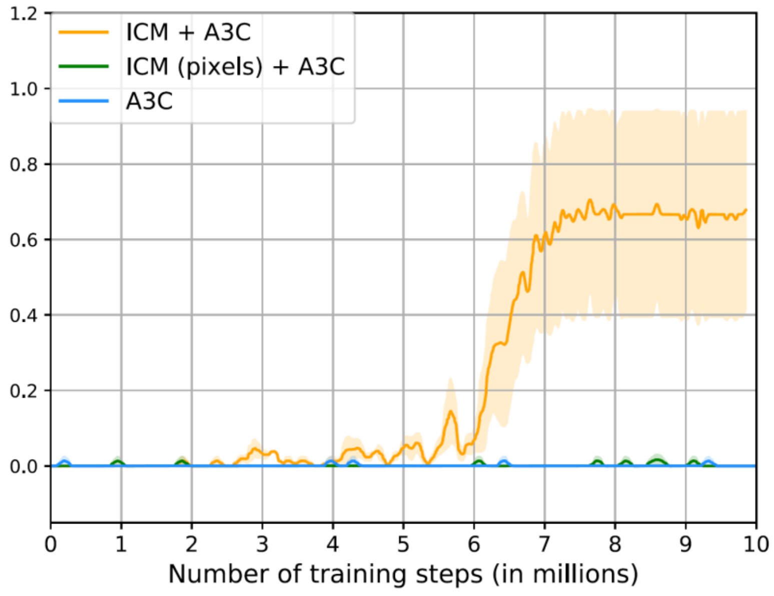
Relative performance improves as reward sparseness increases.



(a) “dense reward” setting



(b) “sparse reward” setting



(c) “very sparse reward” setting

Information Bottleneck Approaches

Information bottleneck* approaches are a powerful new approach in machine learning that improves robustness and generalization.

As we will see in a moment, they have remarkable implications for Reinforcement Learning when used as an auxiliary reward.

The idea is to train a machine learning model to perform a task, or set of tasks, while minimizing spurious knowledge about the input.

* *Tishby, Naftali; Pereira, Fernando C.; Bialek, William (September 1999). The Information Bottleneck Method*

Mutual Information

To describe the information bottleneck method, we need to define mutual information:

The ***mutual information*** between two random variables X and Y with joint distribution $p(X, Y)$ is defined as:

$$I(X; Y) = \int_y \int_x p(x, y) \log \left(\frac{p(x, y)}{p_X(x) p_Y(y)} \right) dx\, dy$$

Where $p_X(X)$ and $p_Y(Y)$ are the marginal distributions of X and Y.

So it can be written as:

$$I(X; Y) = E_{(x,y)\sim p(x,y)} \left[\log \frac{p(x, y)}{p_X(x) p_Y(y)} \right]$$

and we notice that the mutual information is zero if the variables X and Y are independent, i.e. if $p(x, y) = p_X(x) p_Y(y)$ for all x, y.

Also $I(X; Y) = I(Y; X)$ and by Jensen's inequality $I(X; Y) \geq 0$.

Mutual Information

So mutual information is a measure of how dependent the two variables X and Y are.

If you use $\log_2$ in the formulas, MI measures how many bits of information you gain about Y by knowing X's value.

Example: Let X, Y be random variables over $\{0,1,2,3\}$.

Let $p(X,Y)$ be defined by the following table:

X\Y	0	1	2	3
0	$\frac{1}{8}$	$\frac{1}{8}$	0	0
1	$\frac{1}{8}$	$\frac{1}{8}$	0	0
2	0	0	$\frac{1}{8}$	$\frac{1}{8}$
3	0	0	$\frac{1}{8}$	$\frac{1}{8}$

i.e. X tells you the most significant bit of Y.

Then the marginals $p_X(X)$ and $p_Y(Y)$ are $= \frac{1}{4}$, and

$$I(X;Y) = E_{(x,y)\sim p(x,y)}\left[\log_2 \frac{p(x,y)}{p_X(x)p_Y(y)}\right] = E_{(x,y)\sim p(x,y)}\left[\log_2 \frac{\frac{1}{8}}{\frac{1}{4}*\frac{1}{4}}\right] = 1 \text{ bit}$$

Mutual Information and KL-Divergence

The Mutual Information between X and Y can also be defined as a KL-divergence:

$$I(X;Y) = D_{KL}\left(p(X,Y)||p_X(X)p_Y(Y)\right)$$

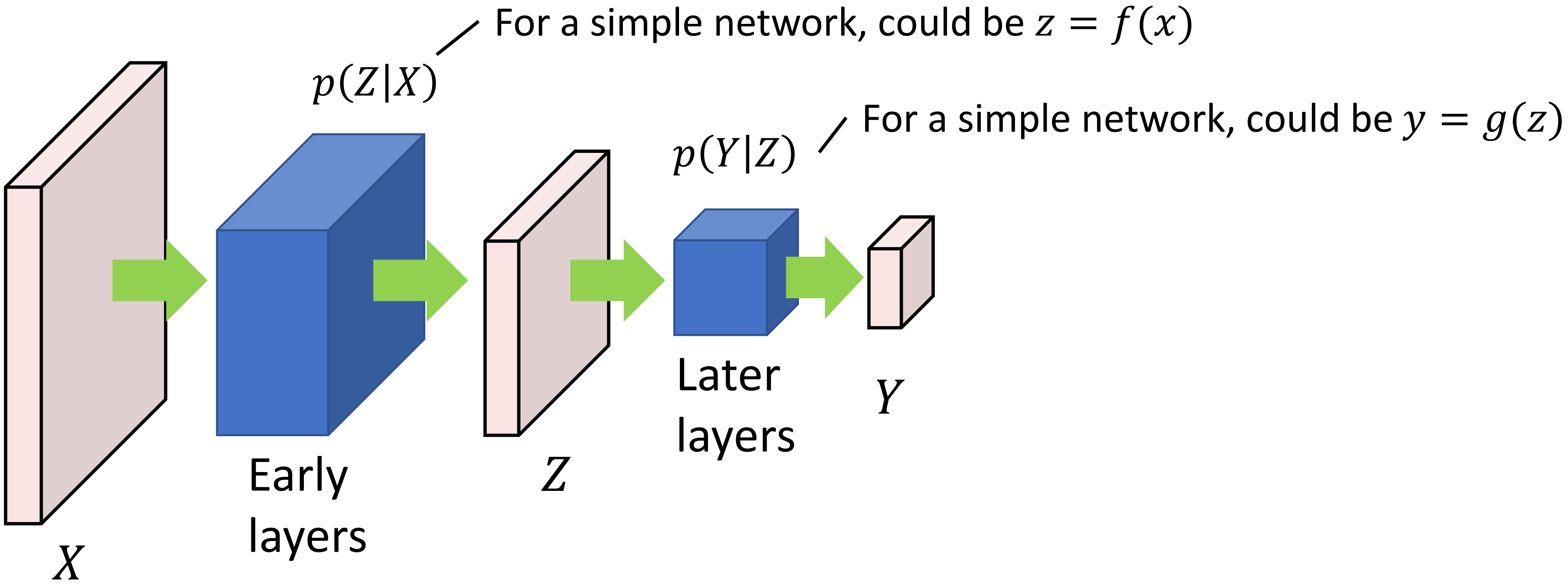
i.e. how much the joint distribution of X and Y diverges from the factored distribution $p_X(X)p_Y(Y)$.

The divergence is zero if and only if X and Y are independent, and otherwise ≥ 0.

Information Bottleneck Method

We assume we have some data X, Y where X is an input and Y is a desired output or label.

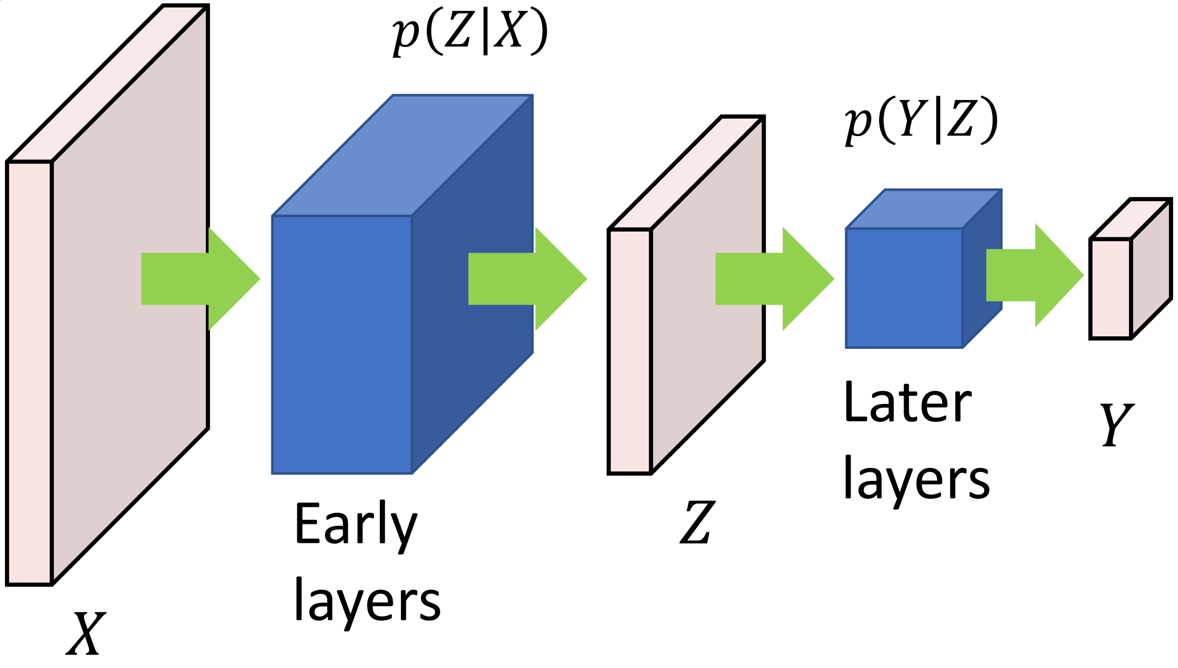
Let Z be a latent or hidden variable "between" X and Y, typically the activations of some layer of a deep network.



Information Bottleneck

We need to be able to predict Y from Z, so we want to maximize $I(Y;Z)$. This should be at least as good as predicting Y from X, i.e. we want $I(Y;Z) = I(Y;X)$.

Information Bottleneck: But we also want to minimize extraneous information in Z, i.e. minimize $I(X;Z)$.

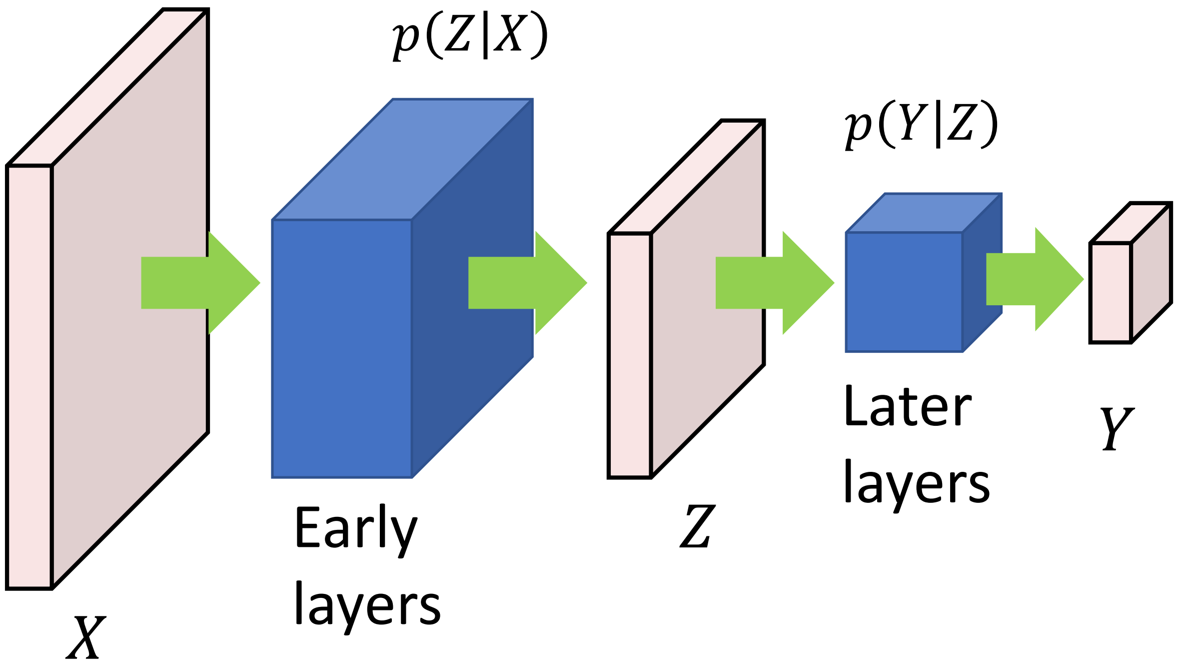


Information Bottleneck Optimization

Information Bottleneck: The combined objective is

$$\max_{p(Z|X)} \; I\,(Z;Y) - \beta\, I(Z;X)$$

Minimizing $I(Z;X)$ makes the model less sensitive to the particular dataset. It improves robustness (sensitivity to single inputs) and generalization.

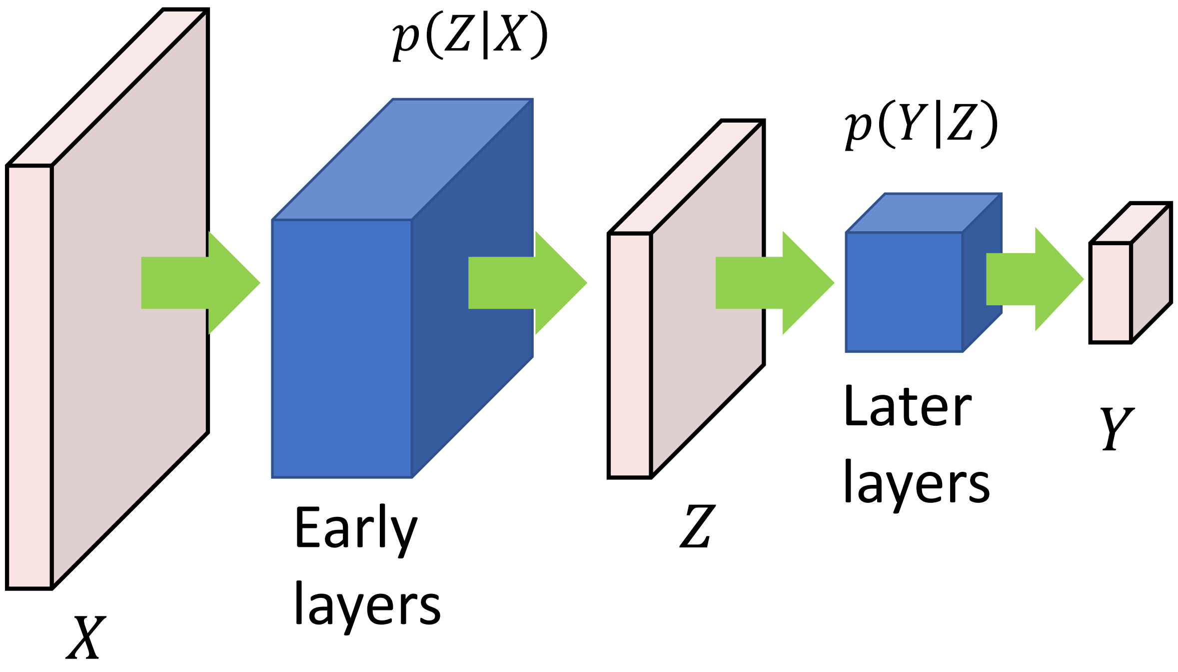


Information Bottleneck Example

Information Bottleneck: Suppose $c(X)$ is a label of X we would like to ignore: we minimize

$$\max_{p(Z|X)} I(Z;Y) - \beta\, I(Z;c(X))$$

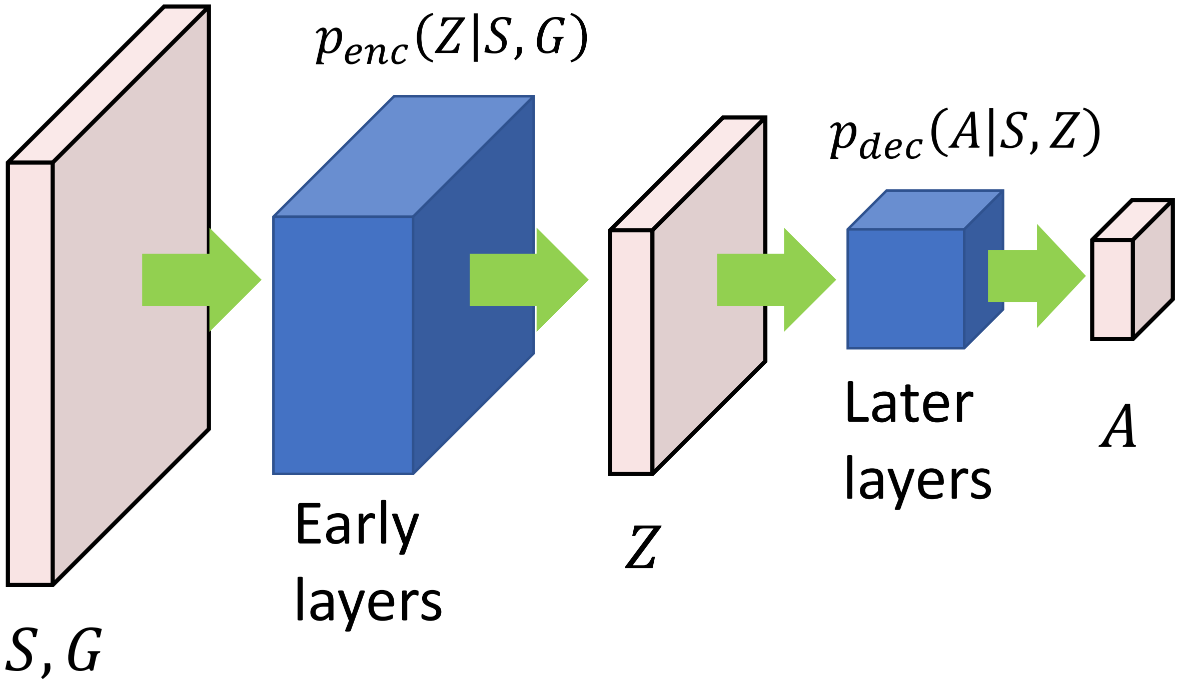
e.g. $c(X) = 1$ means X is a simulator image. $c(X) = 0$ means X is a real image. This forces the model not to represent c in Z, i.e. to use features that are domain-independent.

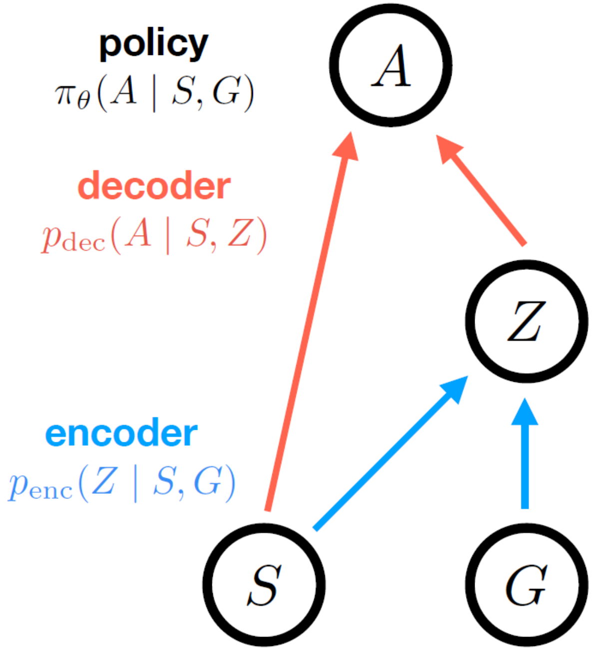


InfoBot: Information Bottleneck for Reinforcement Learning

Minimize the Conditional Goal/Action Mutual Information, i.e. minimize the divergence between $\pi_\theta(A|G,S)$ and a default policy $\pi_0(A|S)$ that doesn't depend on G.
The policy has a bottleneck Z and satisfies:

$$\pi_\theta(A|G,S) = \sum_z p_{enc}(z|S,G)p_{dec}(A|S,z)$$

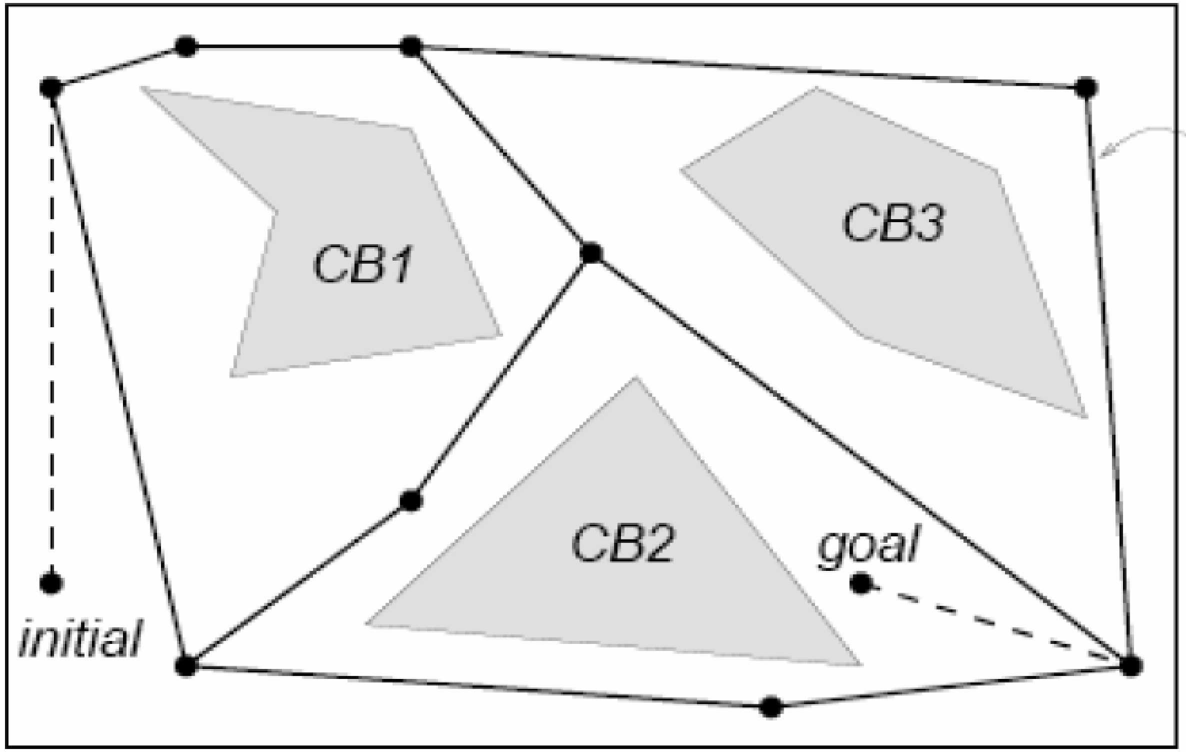




InfoBot: Minimizing Goal-Conditioned Actions

Since the actions are "downstream" from Z, it follows that $I(Z; G|S) \geq I(A; G|S)$. Our original objective was $J(\theta) = E_{\pi_\theta}[r] - \beta I(A; G|S) \geq E_{\pi_\theta}[r] - \beta I(Z; G|S)$, and we indirectly optimize $J(\theta)$ by maximizing the lower bound:

$$E_{\pi_\theta}[r] - \beta\, I(Z; G|S)$$



Goal-conditioned actions only happen at vertices

Adversarial Information Bottlenecks

Q. Xie, et al. Controllable invariance through adversarial feature learning. NIPS 2016

We can minimize mutual information with adversarial methods.

To minimize $I(Z;G|S) = \sum_{z,s,g} p(z,s,g) \log \frac{p(g,z|s)}{p(g|s)p(z|s)} = \sum_{z,s,g} p(z,s,g) \log \frac{p(g|s,z)}{p(g|s)}$

We can ***train goal predictors*** $q(g|s,z)$ and $q_0(g|s)$ using cross-entropy loss. i.e. Maximize:

$$\sum_{z,s,g} p(z,s,g) \log q(g|s,z) \quad \text{and} \quad \sum_{s,g} p(s,g) \log q_0(g|s)$$

1. Train predictors: we compute trajectories and maximize over parameters of q and q_0:

$$\frac{1}{N}\sum_{t=1}^{N} \log q\left(g|s_t,z_t\right) + \cancel{\log q_0\left(g|s_t\right)}$$

2. Maximize using RL over model parameters θ the objective:

$$\frac{1}{N}\sum_{t=1}^{N} \gamma^t r_t - \beta\left(\log q\left(g|s_t,z_t\right) - \cancel{\log q_0\left(g|s_t\right)}\right)$$

Doesn't depend on z

InfoBot: Variational Optimization

Now $E_{\pi_\theta}[r] - \beta I(Z;G|S)$ can be written as:

$$E_{\pi_\theta}\left[r \ - \beta D_{KL}\left(p_{enc}(Z|S,G)||p(Z|S)\right)\right]$$

Where $p(Z|S) = \int_g p_{enc}(Z|S,g)dg$.

Now $I(Z;G|S) = \sum_{z,g,s} p(z,s,g) \log \frac{p(z|s,g)}{p(z|s)}$

$$= \sum_{z,g,s} p(z,s,g) \log p(z|s,g) - \sum_{z,s} p(s)p(z|s) \log p(z|s)$$

$$\geq \sum_{z,g,s} p(z,s,g) \log p(z|s,g) - \sum_{z,s} p(s)p(z|s) \log q(z|s)$$

Where $q(z|s)$ is a variational approximation to $p(z|s)$.

InfoBot: Variational Optimization

So the actual maximization is (using a lower bound on $J(\theta)$):

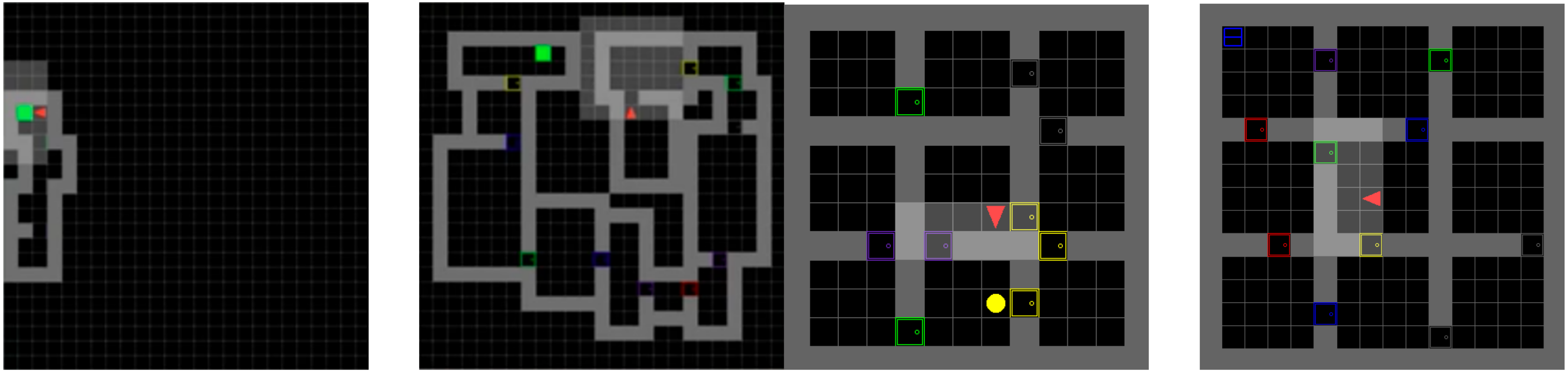
$$\max_{\pi_\theta} E_{\pi_\theta}\left[r \ - \beta D_{KL}\left(p_{enc}(Z|S,G)||q(Z|S)\right)\right]$$

And we can use the reparameterization trick for both $p_{enc}(Z|S,G)$ and $q(Z|S)$.

This seems underdetermined since we have to learn both $p_{enc}()$ and $q()$ but notice that $q()$ will be optimized to minimize the average divergence to $p_{enc}()$.

Note that both densities must be easily computable. E.g. we need to use neural networks that output mean and variance of Z.

InfoBot: Generalization



(a) MultiRoomN4S4 (b) MultiRoomN12S10 (c) FindObjS5 (d) FindObjS6

Figure 2: **MultiRoomNXSY and FindObjSY MiniGrid environments.** See text for details.

Method	FindObjS7	FindObjS10
Goal-conditioned A2C	56%	36%
InfoBot with $\beta = 0$	44%	24%
InfoBot	**81%**	**61%**

InfoBot: Finding Bottleneck States

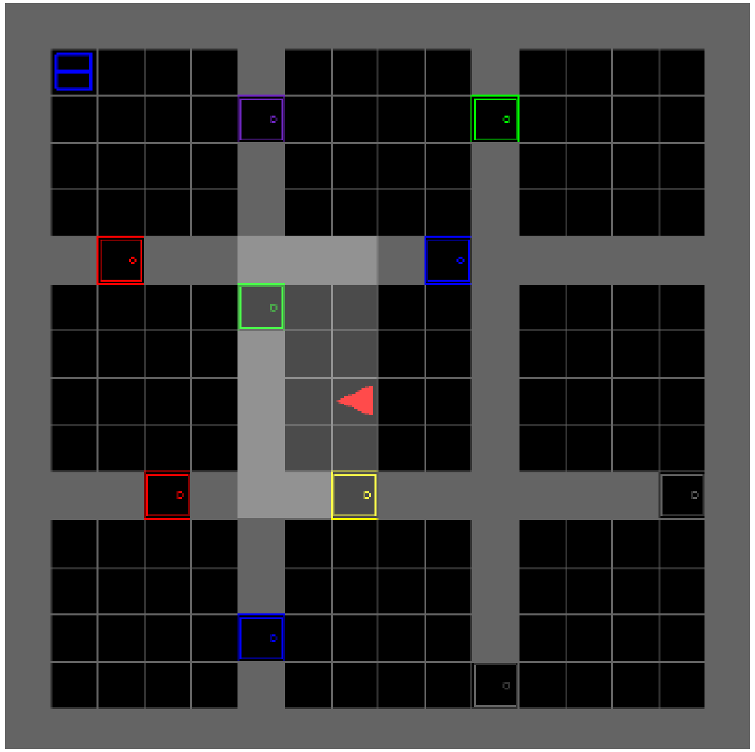
The states s with high $I(G; A|s)$ (or high $I(G; Z|s)$) give additional rewards:

Adversarial model: these states have high $D_{KL}\big(q(G|s_t, z_t)||q_0(G|s_t)\big)$

Variational model: these states have high $D_{KL}\big(p_{enc}(Z|s_t, g)||q(Z|s_t)\big)$

There is also a discount for repeated visits:

$$r_t = r_e(t) + \frac{\beta}{\sqrt{c(s_t)}} D_{\mathrm{KL}}\left[p_{\mathrm{enc}}(Z \mid s_t, g_t) \mid q(Z \mid s_t)\right]$$



They often correspond to doorways or similar locations where the agent must make a goal-conditioned decision.

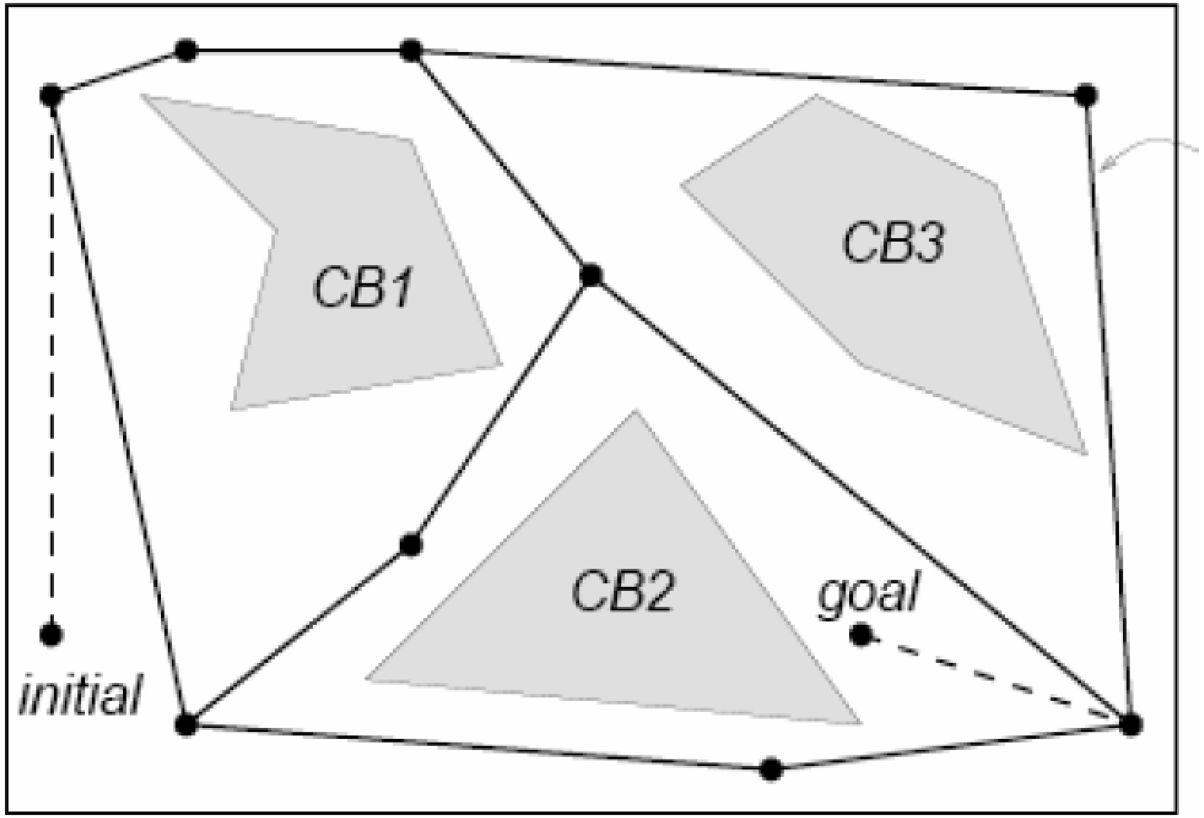
By regularly revisiting these states, the agent is more likely to explore all the open regions in the environment.

InfoBot: Finding Bottleneck States

The states s with high $I(G; A|s)$ (or high $I(G; Z|s)$) provide additional rewards.

So, the model is trained first to find policies which ***minimize*** $I(G; A|s)$. This leads to sparse environment maps with few vertices.

But then the model is run with an exploration bonus to ***seek out states with high*** $I(G; A|s)$. These lead to rapid exploration of the sparse map.



InfoBot: Exploration Performance

Method	MultiRoomN3S4	MultiRoomN5S4
Goal-conditioned A2C	0%	0%
TRPO + VIME	54%	0%
Count based exploration	**95%**	0%
Curiosity-based exploration	**95%**	54%
InfoBot (decision state exploration bonus)	90%	**85%**

Table 2: **Transferable exploration strategies on MultiRoomNXSY.** InfoBot encoder trained on MultiRoomN2S6. All agents evaluated on MultiRoomN3S4 and N5S4. While several methods perform well with 3 rooms, InfoBot performs far better as the number of rooms increases to 5.

InfoBot: For General RL

Minimizing $I(G; A|s)$ is a very good regularizer for general RL problems.

States where $I(G; A|s)$ is high are potential points of failure (by making a bad decision).

By avoiding these states (where it doesn't have to move), the policy can accomplish the goal with fewer decisions.

i.e. a pong agent will avoid "tracking" the ball in danger of missing it.

Take-aways

- Optimistic exploration methods favor poorly-explored states, assuming their value is as high as it could be. Use counts to estimate uncertainties.

- An elegant way to model θ uncertainty is to train an ensemble of models using bootstrap sampling. Sample from the posterior → choose a model.

- Avoid random walks at all costs! $\Omega(N^2)$ cost to explore N states. Simple methods: epsilon-greedy, entropy regularization and Thompson sampling all do random walks. Fix policy for an episode to avoid this.

- Doing better requires memory (counts, environment models etc.)

Take-aways

- Can plan and anticipate states and rewards with a simplified environment model ($\phi(s)$ instead of s).

- Dis-similarity to previous first-person views (curiosity) is a useful exploration heuristic.

- Information Bottlenecks:
  - Minimize the number of goal-conditioned states, ideally to a finite number.
  - But then seek them out and revisit them (exploration) !