

Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

# Exploration and Function Approximation

CMU 10703

Katerina Fragkiadaki



# This lecture

## Exploration in Large Continuous State Spaces

# Exploration-Exploitation

**Exploration:** trying out new things (new behaviours), with the hope of discovering higher rewards

**Exploitation:** doing what **you know** will yield the highest reward

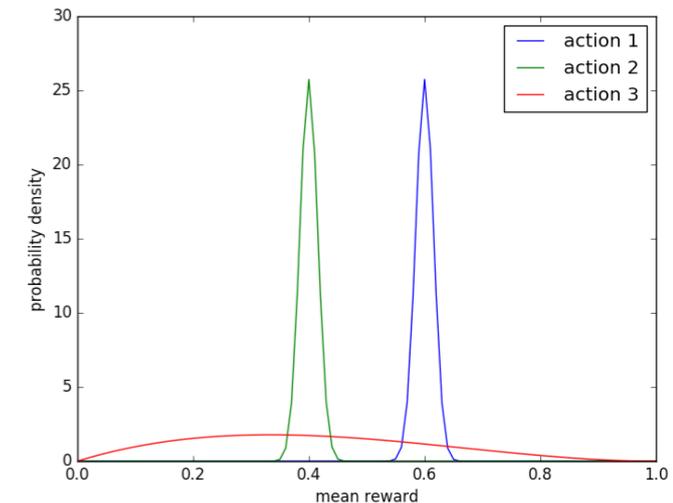
Intuitively, we explore efficiently once **we know what we do not know**, and target our exploration efforts to the unknown part of the space.

All non-naive exploration methods consider some form of **uncertainty estimation, regarding policies, Q-functions, state (or state-action) I have visited, or transition dynamics..**

# Recall: Thompson Sampling

Represent a posterior distribution of mean rewards of the bandits, as opposed to mean estimates.

1. Sample from it  $\theta_1, \theta_2, \dots, \theta_k \sim \hat{p}(\theta_1, \theta_2 \dots \theta_k)$
2. Choose action  $a = \arg \max_a \mathbb{E}_\theta[r(a)]$
3. Update the mean reward distribution  $\hat{p}(\theta_1, \theta_2 \dots \theta_k)$



The equivalent of mean expected rewards for general MDPs are Q functions

# Exploration via Posterior Sampling of Q functions

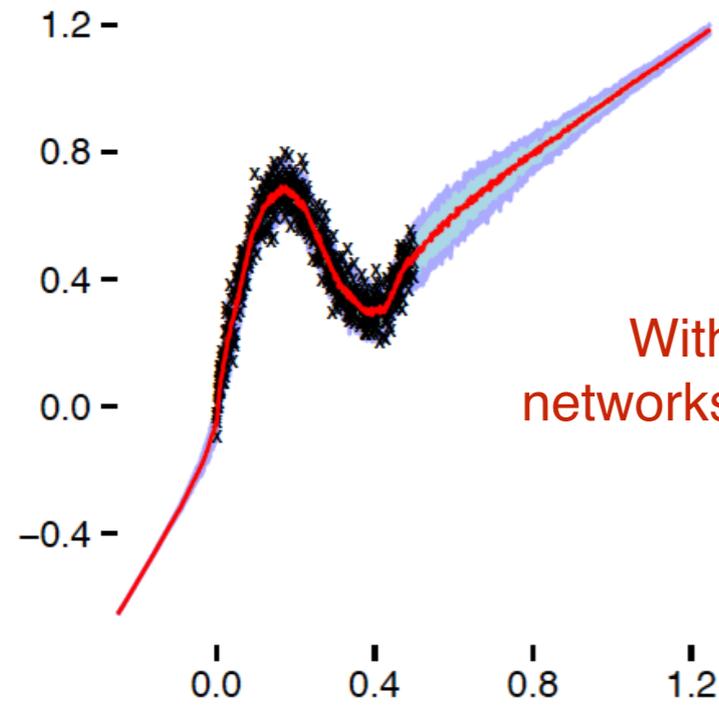
Represent a posterior distribution of Q functions, instead of a point estimate.

1. Sample from  $P(Q)$   $Q \sim P(Q)$
2. Choose actions according to this Q for one episode  $a = \arg \max Q(a, s)$
3. Update the Q distribution using the collected experience tuples

Then we do not need  $\epsilon$ -greedy for exploration! Better exploration by representing uncertainty over Q.

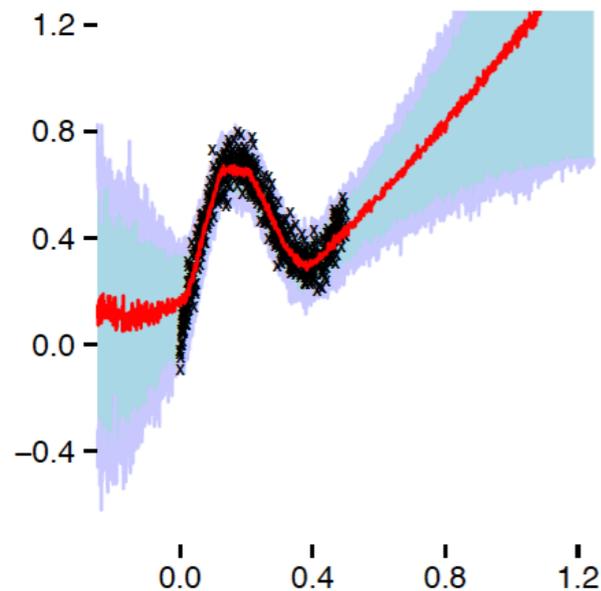
But how can we learn a distribution of Q functions  $P(Q)$  if Q function is a deep neural network?

# Representing Uncertainty in Deep Learning

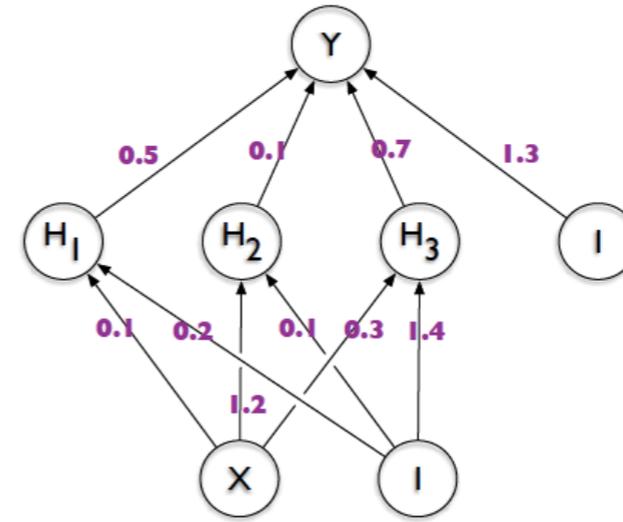


With standard regression networks we cannot represent our uncertainty

A regression network trained on  $\mathbf{X}$

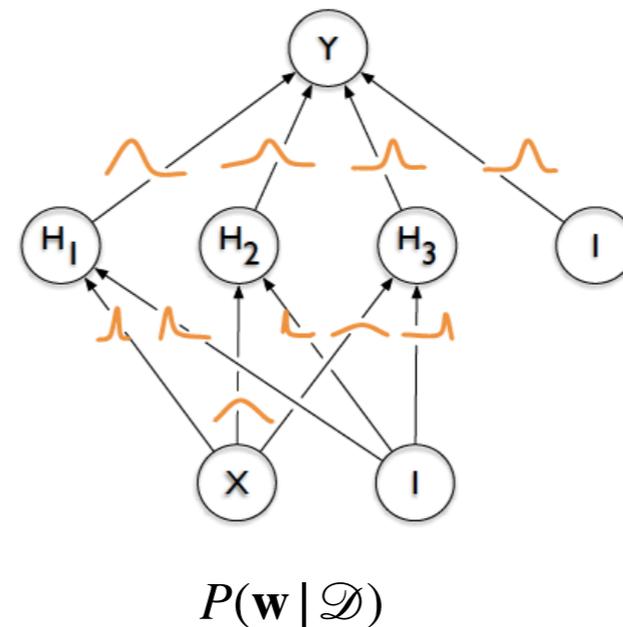


A Bayesian regression network trained on  $\mathbf{X}$



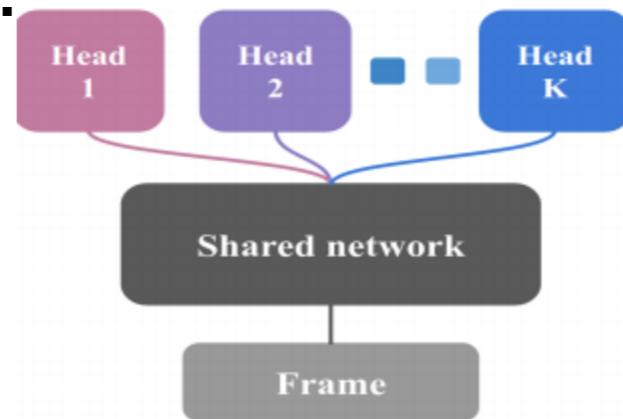
$$\mathbf{w}^{\text{MAP}} = \arg \max_{\mathbf{w}} \log P(\mathbf{w} | \mathcal{D})$$

$$= \arg \max_{\mathbf{w}} \log P(\mathcal{D} | \mathbf{w}) + \log P(\mathbf{w}).$$



# Exploration via Posterior Sampling of Q-functions

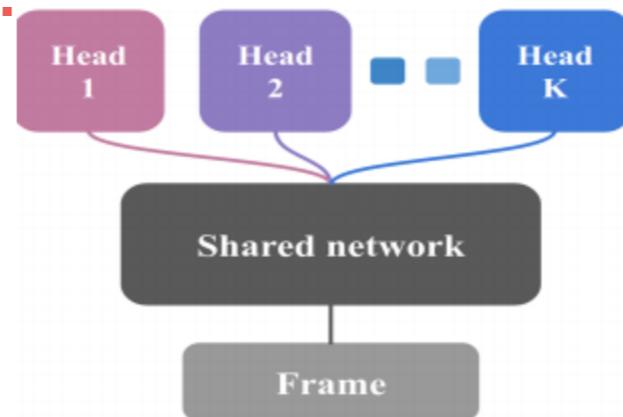
1. **Bayesian neural networks.** Estimate posteriors for the neural weights, as opposed to point estimates. We just saw that..
2. **Neural network ensembles.** Train multiple Q-function approximations each on using different subset of the data. A reasonable approximation to 1.
3. **Neural network ensembles with shared backbone.** Only the heads are trained with different subset of the data. A reasonable approximation to 2 with less computation.



4. **Ensembling by dropout.** Randomly mask-out (zero out) neural network weights, to create different neural nets, both at train and test time. reasonable approximation to 2.

# Exploration via Posterior Sampling of Q-functions

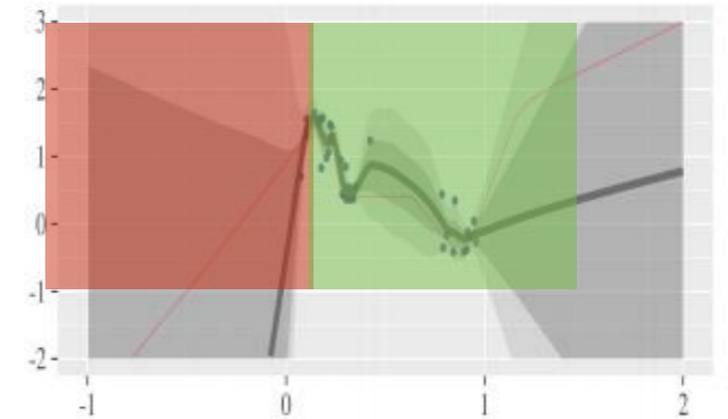
1. **Bayesian neural networks.** Estimate posteriors for the neural weights, as opposed to point estimates. We just saw that..
2. **Neural network ensembles.** Train multiple Q-function approximations each on using different subset of the data. A reasonable approximation to 1.
3. **Neural network ensembles with shared backbone.** Only the heads are trained with different subset of the data. A reasonable approximation to 2 with less computation.



4. **Ensembling by dropout.** Randomly mask-out (zero out) neural network weights, to create different neural nets, both at train and test time. reasonable approximation to 2. (but authors showed 3. worked better than 4.)

# Exploration via Posterior Sampling of Q-functions

1. Sample from  $P(Q)$   $Q \sim P(Q)$
2. Choose actions according to this  $Q$  for one episode  $a = \arg \max Q(a, s)$
3. Update the  $Q$  distribution using the collected experience tuples

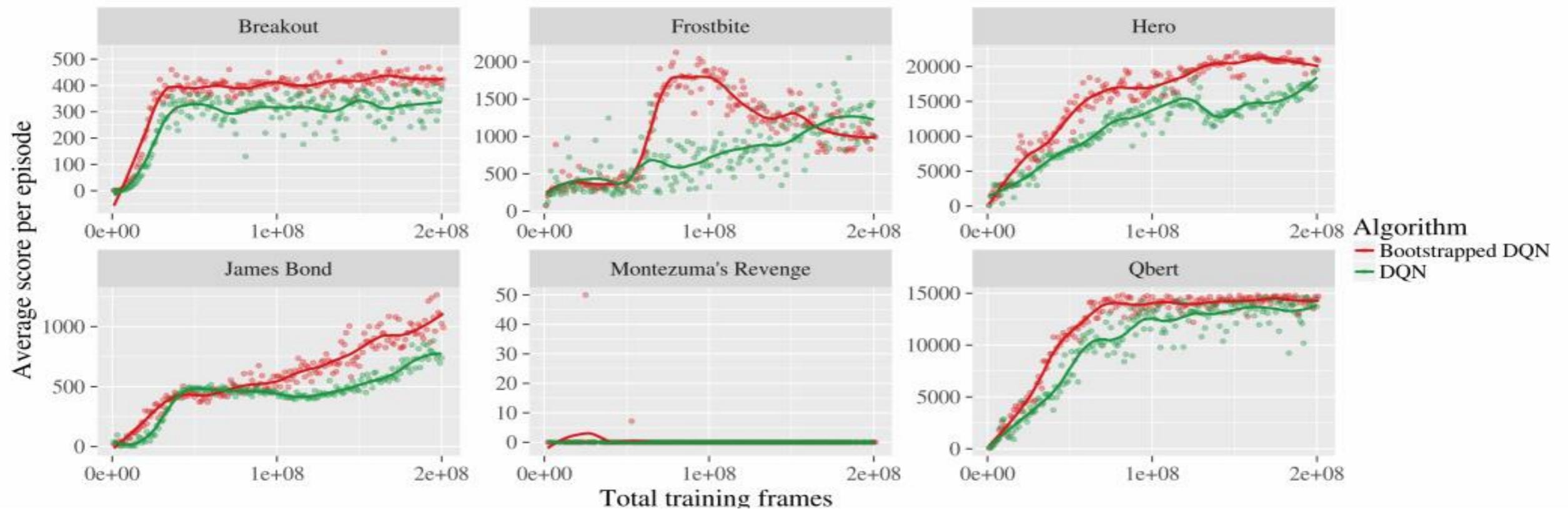


With ensembles we achieve similar things as with Bayesian nets:

- The entropy of predictions of the network (obtained by sampling different heads) is high in the no data regime. Thus, Q function values will have **high entropy** there and encourage exploration.
- When Q values have **low entropy**, i exploit, i do not explore.

No need for  $\epsilon$ -greedy, no exploration bonuses.

# Exploration via Posterior Sampling of Q-functions

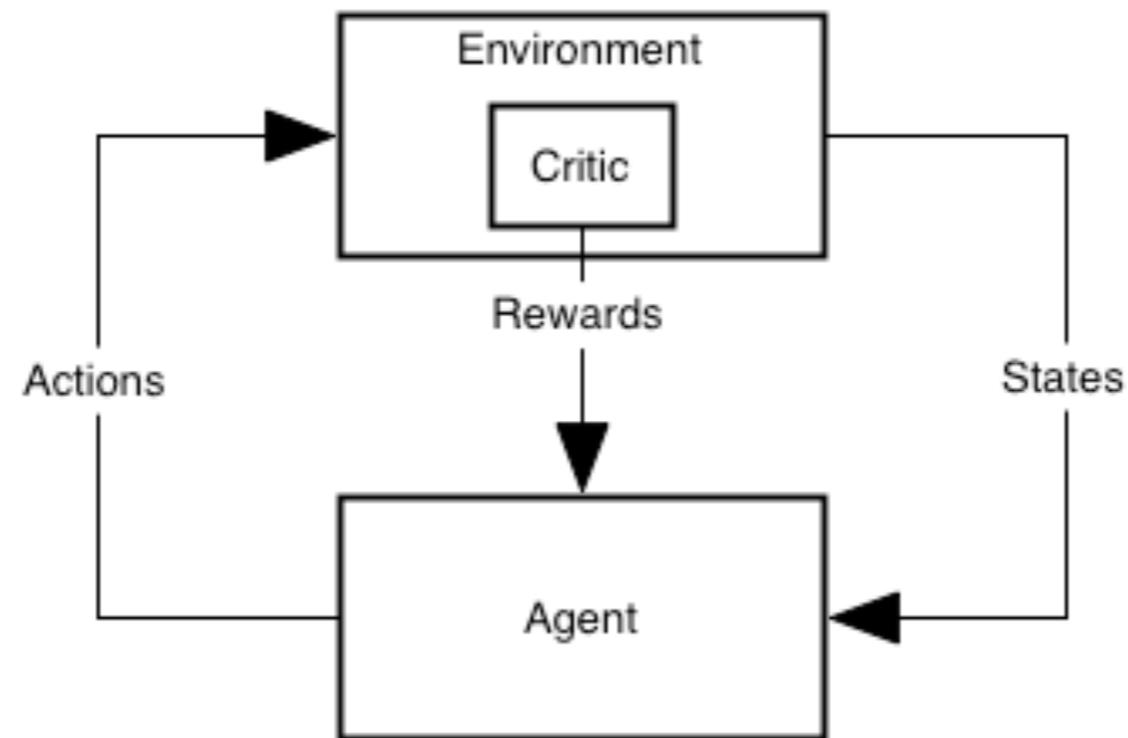


# Motivation

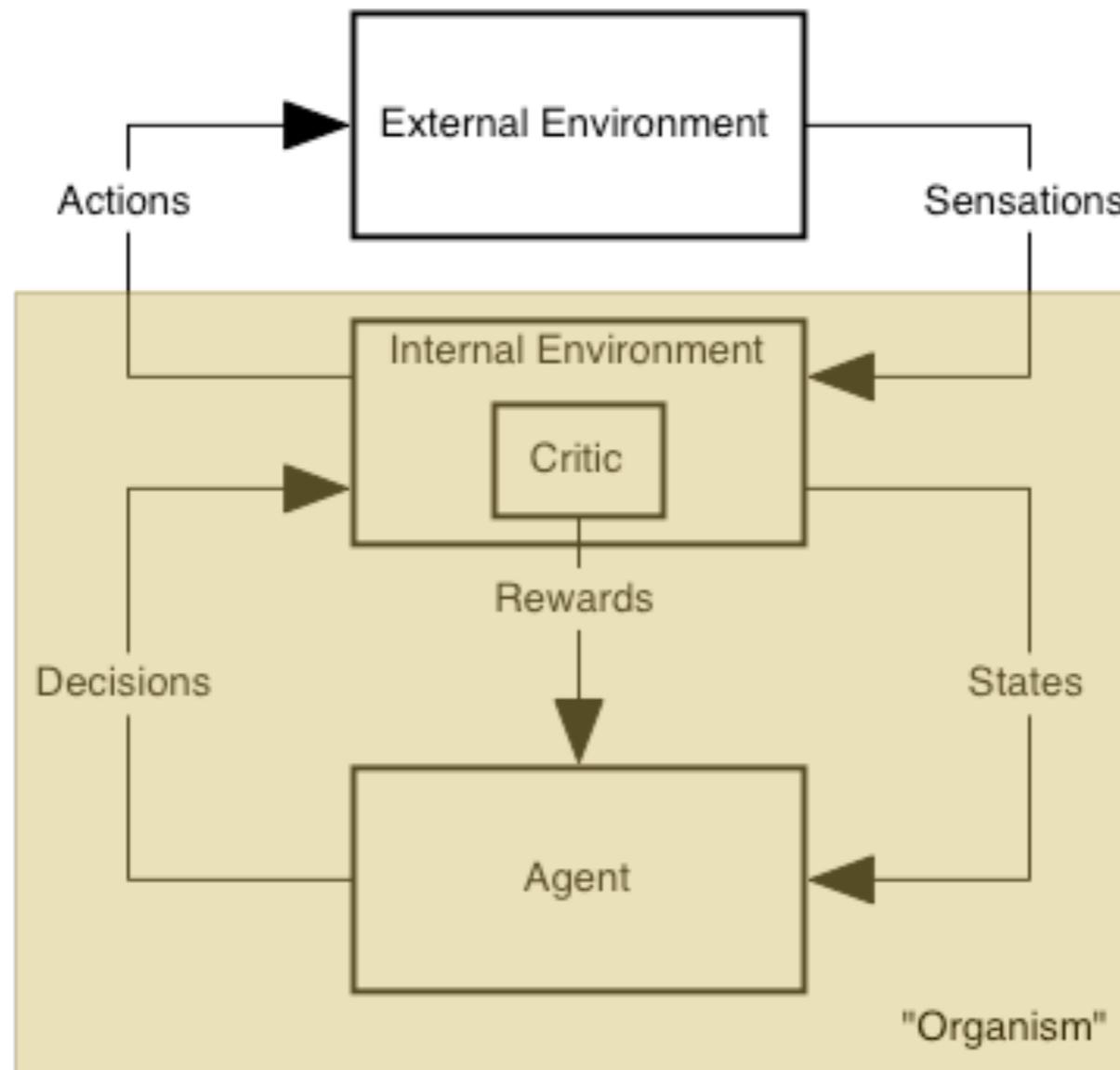
Motivation: “Forces” that energize an organism to act and that direct its activity

- **Extrinsic** Motivation: being moved to do something because of some external reward (\$\$, a prize, etc.)
- **Intrinsic** Motivation: being moved to do something because it is inherently enjoyable (curiosity, exploration, novelty, surprise, incongruity, complexity...)
- **Intrinsic** Necessity: being moved to do something because it is necessary (eat, drink, find shelter from rain...)

# Extrinsic Rewards



# Intrinsic Rewards



*All rewards are intrinsic*

# Motivation

Motivation: “Forces” that energize an organism to act and that direct its activity

- **Extrinsic** Motivation: being moved to do something because of some external reward (\$\$, a prize, etc.)
- **Intrinsic Motivation: being moved to do something because it is inherently enjoyable** (curiosity, exploration, novelty, surprise, incongruity, complexity...)
- **Intrinsic** Necessity: being moved to do something because it is necessary (eat, drink, find shelter from rain...)

# Motivation

Motivation: “Forces” that energize an organism to act and that direct its activity

- **Extrinsic** Motivation: being moved to do something because of some external reward (\$\$, a prize, etc.)-**Task dependent**
- **Intrinsic Motivation: being moved to do something because it is inherently enjoyable** (curiosity, exploration, novelty, surprise, incongruity, complexity...)-**Task independent! A general loss functions that drives learning**
- **Intrinsic** Necessity: being moved to do something because it is necessary (eat,drink, find shelter from rain...)

# Curiosity VS Survival

“As knowledge accumulated about the conditions that govern exploratory behavior and about how quickly it appears after birth, it seemed less and *less likely that this behavior could be a derivative of hunger, thirst, sexual appetite, pain, fear of pain, and the like*, or that stimuli sought through exploration are welcomed because they have previously accompanied satisfaction of these drives.”

D. E. Berlyne, *Curiosity and Exploration*, Science, 1966

# Curiosity and Never-ending Learning

Why should we care?

- Because curiosity is a general, task independent cost function, that if we successfully incorporate to our learning machines, it may result in agents that (want to) improve with experience, like people do.
- Those intelligent agents would not require supervision by coding up reward functions for every little task, they would learn (almost) autonomously
- Curiosity-driven motivation is beyond satisfaction of hunger, thirst, and other biological activities (which arguably would be harder to code up in artificial agents..)

# Curiosity-driven exploration

Seek **novelty/surprise (curiosity driven exploration)**:

- Visit *novel states*  $s$  (state visitation counts)
- Observe *novel state transitions*  $(s,a) \rightarrow s'$  (improve transition dynamics)

We would be adding **exploration reward bonuses** to the extrinsics (task-related) rewards:

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$$

Independent of the task in hand!

We would then be using rewards  $R^t(s, a, s')$  in our favorite RL method.

**Exploration reward bonuses are non stationary:** as the agent interacts with the environment, what is now new and novel, becomes old and known. Many methods consider critic networks that combine Monte Carlo returns with TD.

# State Visitation counts in Small MDPs

Book-keep state visitation counts  $N(s)$

Add **exploration reward bonuses** that encourage policies that visit states with fewer counts.

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}(N(s))}_{\text{intrinsic}}$$

UCB:

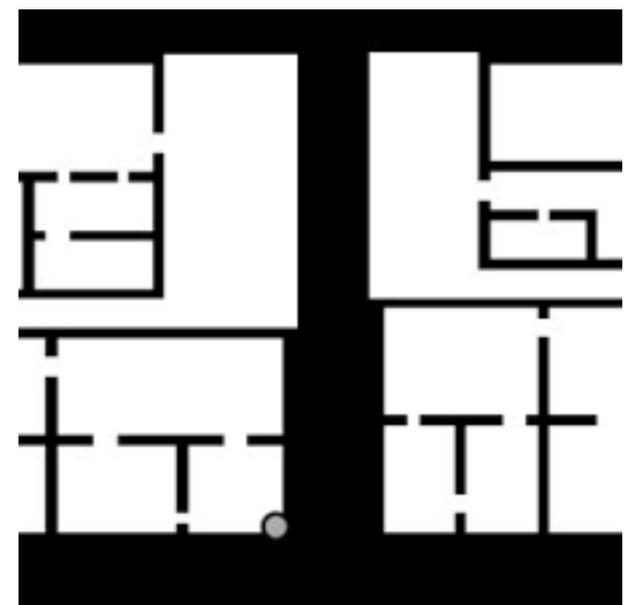
$$\mathcal{B}(N(s)) = \sqrt{\frac{2 \ln n}{N(s)}}$$

MBIE-EB (Strehl & Littman, 2008):

$$\mathcal{B}(N(s)) = \sqrt{\frac{1}{N(s)}}$$

BEB (Kolter & Ng, 2009):

$$\mathcal{B}(N(s)) = \frac{1}{N(s)}$$



# State Visitation Counts in High Dimensions

- We want to come up with something that rewards states that we have not visited often.
- **But in high dimensions, we rarely visit a state twice!**
- We need to capture a notion of state similarity, and reward states that are most *dissimilar* that what we have seen so far, as opposed to *different* (as they will always be different)

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}(N(s))}_{\text{intrinsic}}$$



the rich natural world

# Unifying Count-Based Exploration and Intrinsic Motivation

---

**Marc G. Bellemare**

bellemare@google.com

**Sriram Srinivasan**

srsrinivasan@google.com

**Georg Ostrovski**

ostrovski@google.com

**Tom Schaul**

schaul@google.com

**David Saxton**

saxton@google.com

**Rémi Munos**

munos@google.com

- We use parametrized density estimates instead of discrete counts.
- $p_{\theta}(s)$  :parametrized visitation density: how much we have visited state  $s$ .
- Even if we have not seen exactly the same state  $s$ , the probability can be high if we visited similar states.

# Exploring with Pseudocounts



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

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

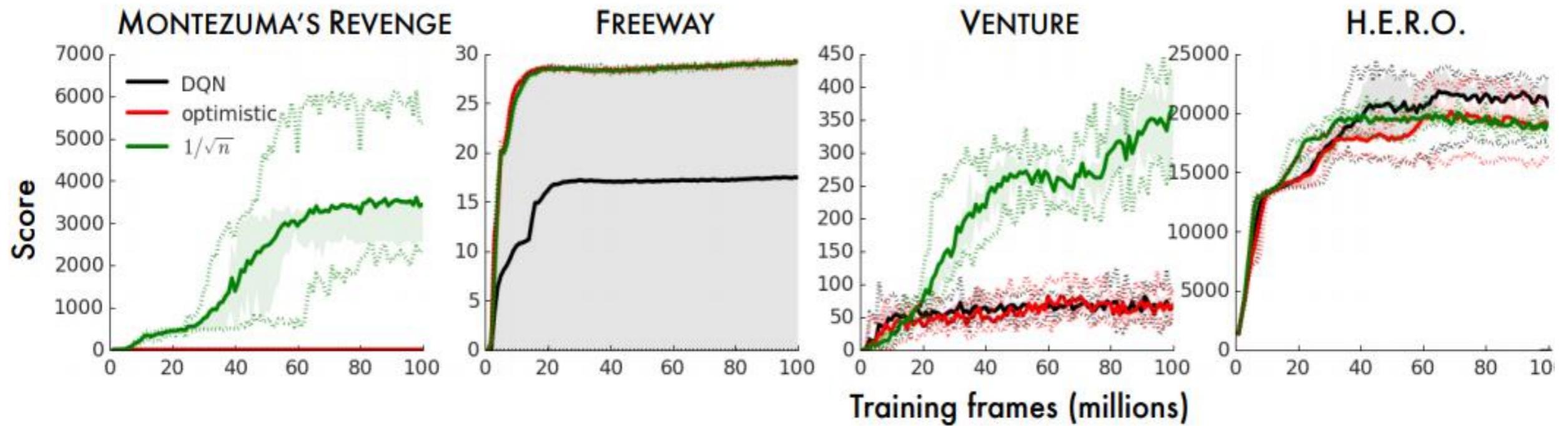
$$p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i) + 1}{\hat{n} + 1}$$

two equations and two unknowns!

$$\hat{N}(\mathbf{s}_i) = \hat{n}p_{\theta}(\mathbf{s}_i)$$

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

$$\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{1}{N(\mathbf{s})}}$$

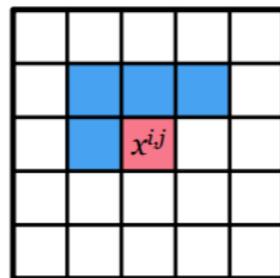


<https://www.youtube.com/watch?v=232tOUPKPoQ&feature=youtu.be>

# State visitation density

$p_{\theta}(s)$  Imagine that states  $s$  are images or short image sequences. This assigns to each a probability that it has been seen before.

CTS model: compute density by multiplying factors for each pixel, factors are location specific.

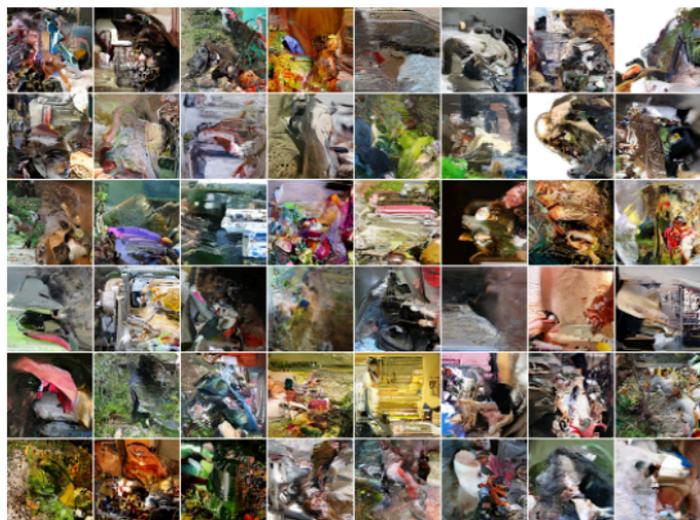


Generative model of images: given an image collection  $D$ , estimate a model that assigns a probability to a (new) image of it coming from collection  $D$ .

# Count-Based Exploration with Neural Density Models

Georg Ostrovski<sup>1</sup> Marc G. Bellemare<sup>1</sup> Aäron van den Oord<sup>1</sup> Rémi Munos<sup>1</sup>  
inpainted images

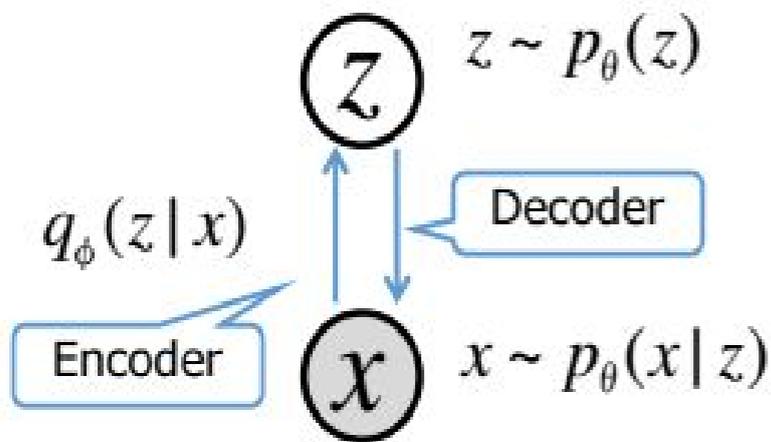
What if we use a state-of-the-art generative model of images?



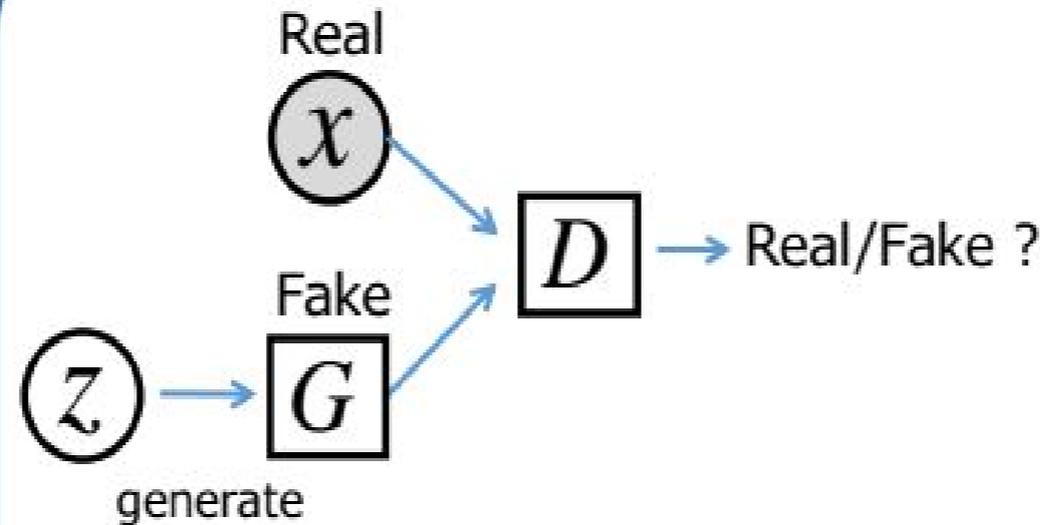
Generated images

# Generative models of Images

## Variational AutoEncoders (VAE)



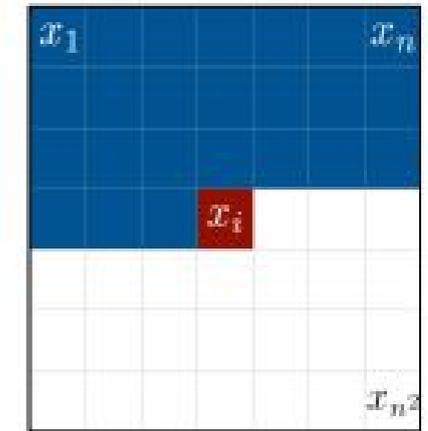
## Generative Adversarial Networks (GAN)



$$\min_G \max_D V(D, G)$$

$$= \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

## Autoregressive Models



$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \dots, x_{i-1})$$

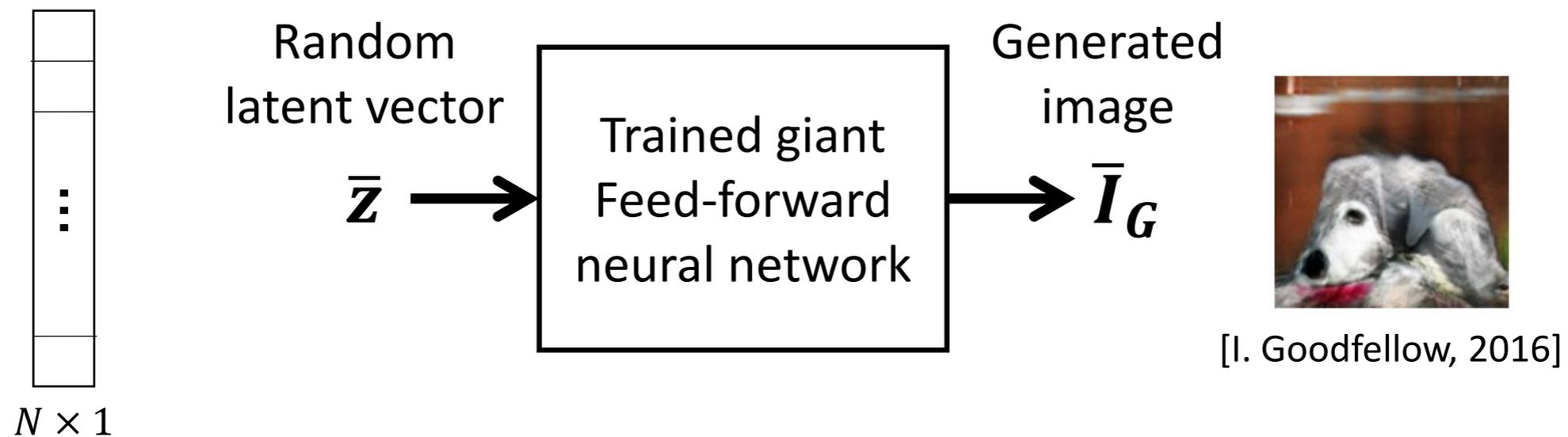
	VAE	GAN	Autoregressive Models
Pros.	- Efficient inference with approximate latent variables.	- generate sharp image. - no need for any Markov chain or approx networks during sampling.	- very simple and stable training process - currently gives the best log likelihood. - tractable likelihood
Cons.	- generated samples tend to be blurry.	- difficult to optimize due to unstable training dynamics.	- relatively inefficient during sampling

(cf. <https://openai.com/blog/generative-models/>)

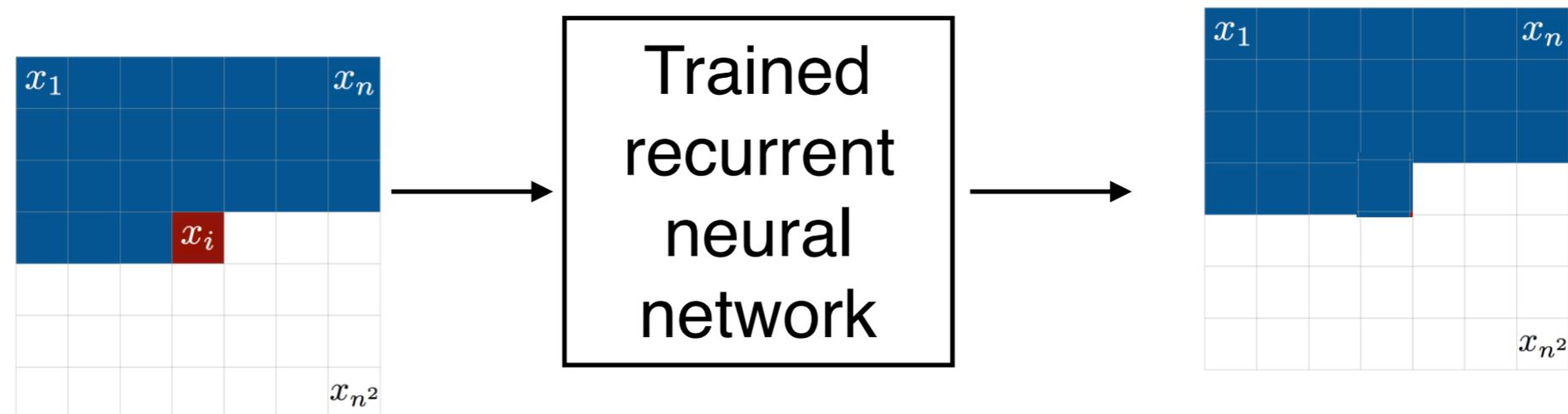
We like that! We want it to compute probabilities, not to draw beautiful samples!

# Generative models of Images

- One shot image generation (usually used in VAEs and GANs):



- Autoregressive image generation: Generate the image one pixel at a time



# Autoregressive Image generation

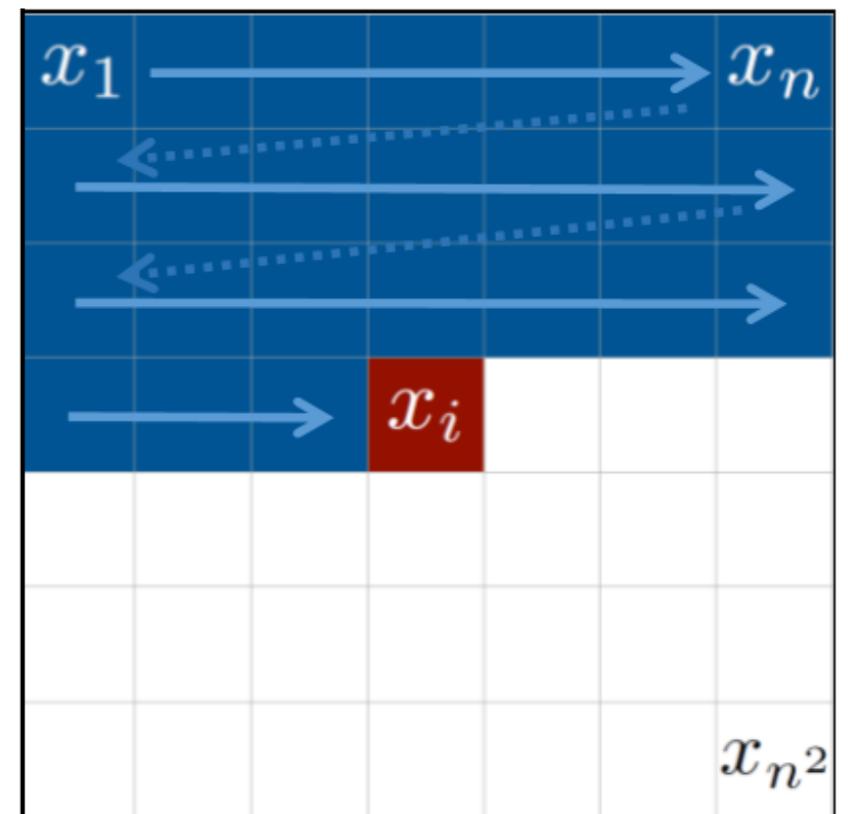
## Intuition

$$p(\mathbf{x}) = p(x_1, x_2, \dots, x_{n^2})$$

Bayes Theorem:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \dots, x_{i-1})$$

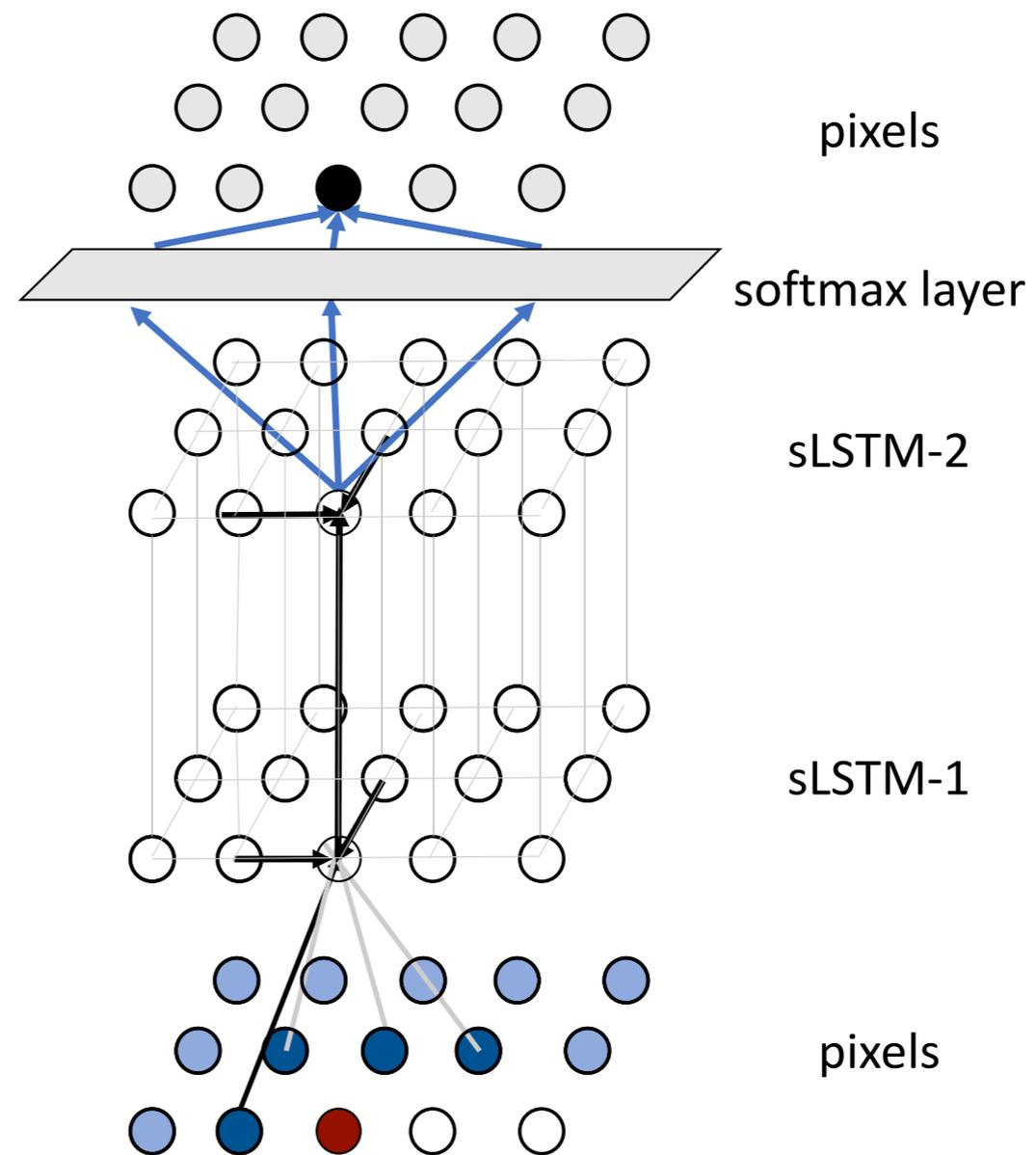
A sequential model!



[Pixel recurrent neural networks](#), ICML 2016

Pixel RNN: a neural networks that sequentially predicts the pixels in the image

# Spatial LSTM



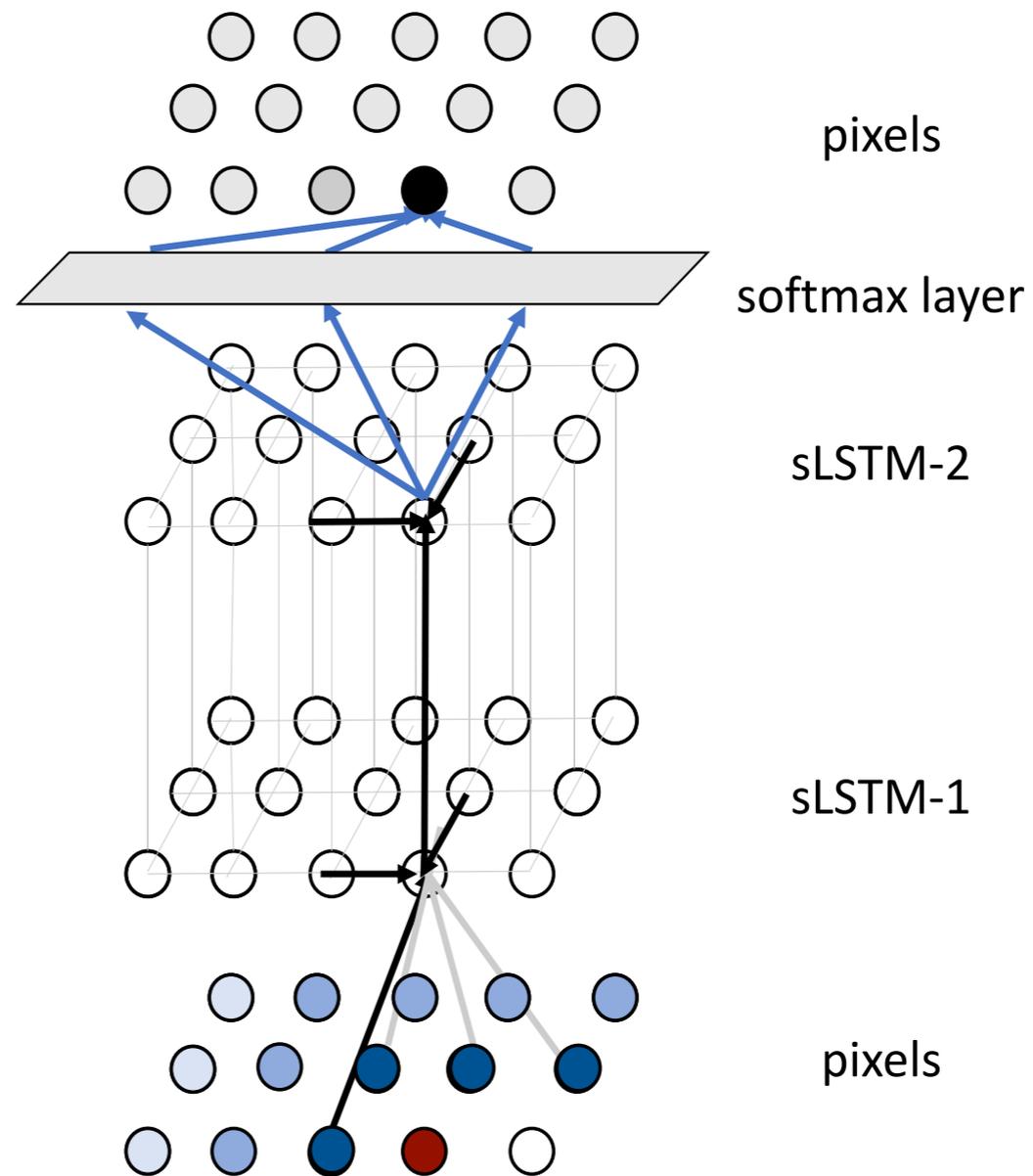
Adapted from: Generative image modeling using spatial LSTM. Theis & Bethge, 2015

  $x_{ij}$  the pixel  $i$  am estimating the value for

  $\mathbf{x}_{<ij}$  the pixel that have already been predicted, and on which our LSTM is conditioning

# Spatial LSTM

Too slow, no parallelization: I update the pixels one by one.



Adapted from: Generative image modeling using spatial LSTM. Theis & Bethge, 2015

  $x_{ij}$  the pixel  $i$  am estimating the value for

  $\mathbf{x}_{<ij}$  the pixel that have already been predicted, and on which our LSTM is conditioning

# Multinomial Distribution for Pixel Value

- Treat pixels as discrete variables:
  - To estimate a pixel value, do classification in every channel (256 classes indicating pixel values 0-255)
  - Implemented with a final softmax layer

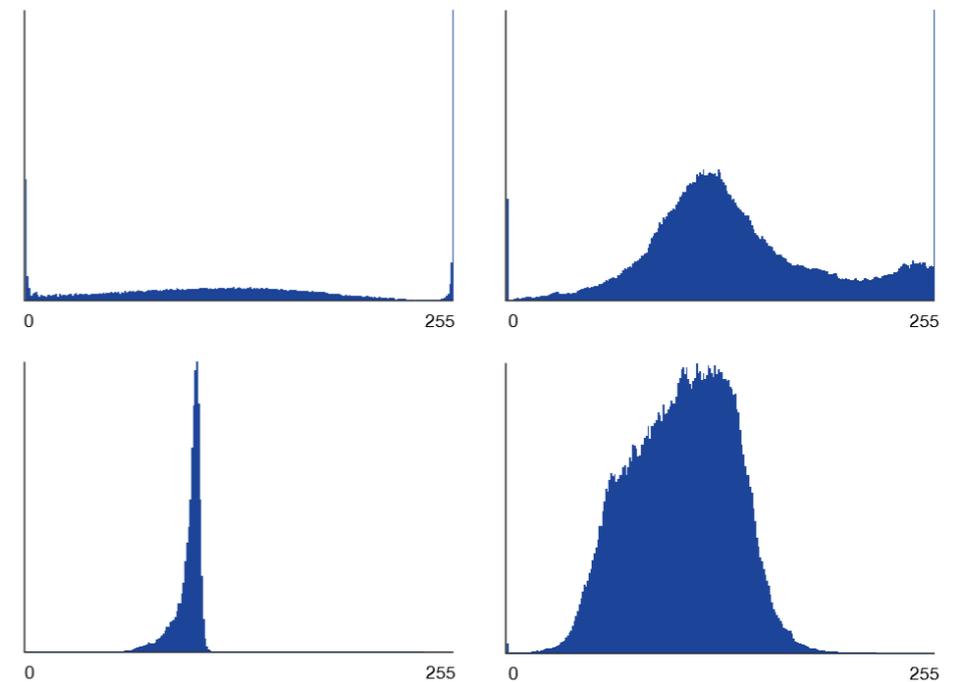
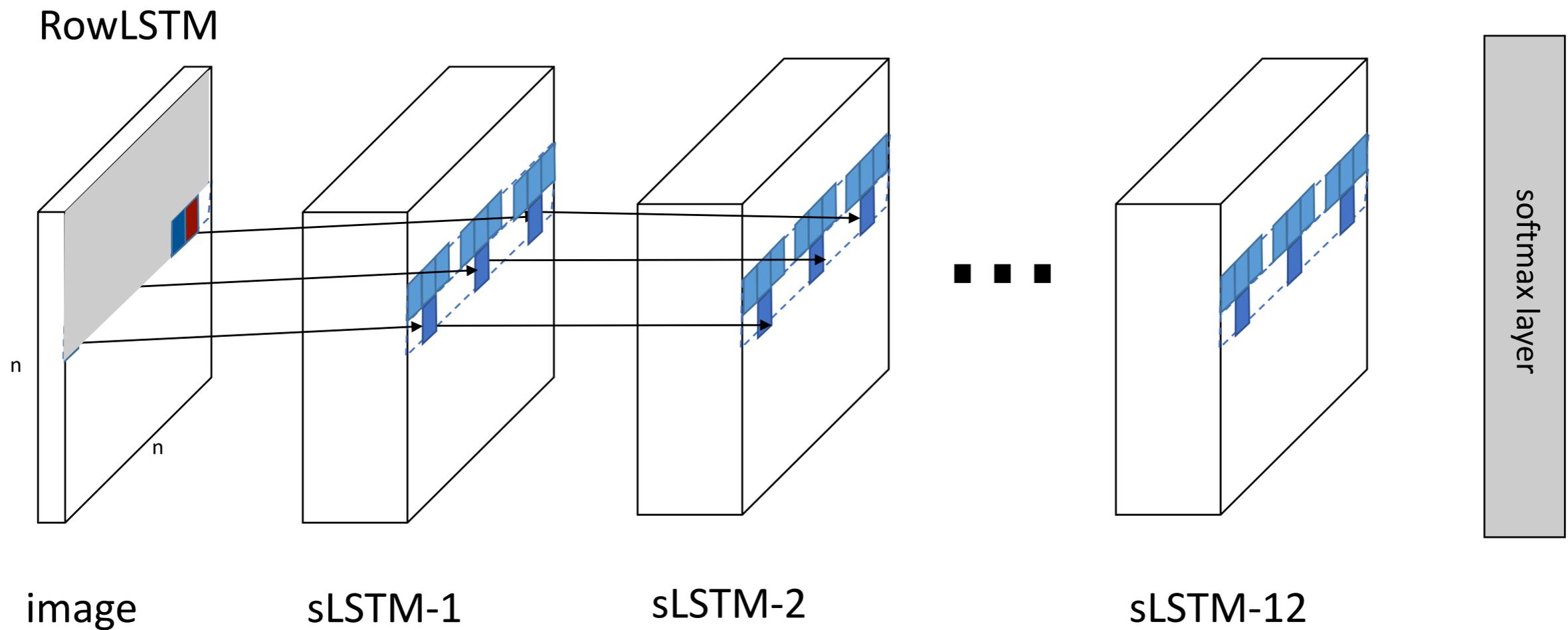


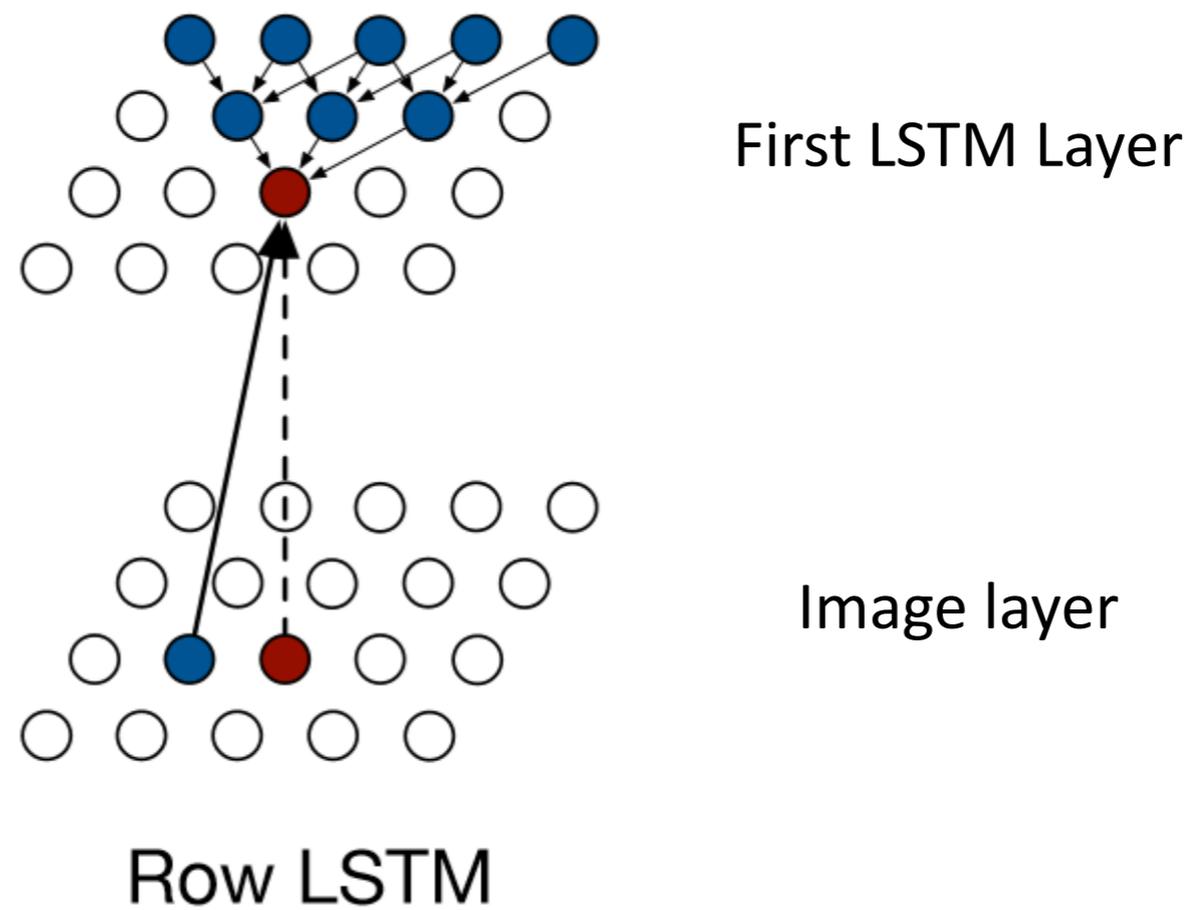
Figure: Example softmax outputs in the final layer, representing probability distribution over 256 classes.

# Pixel RNN

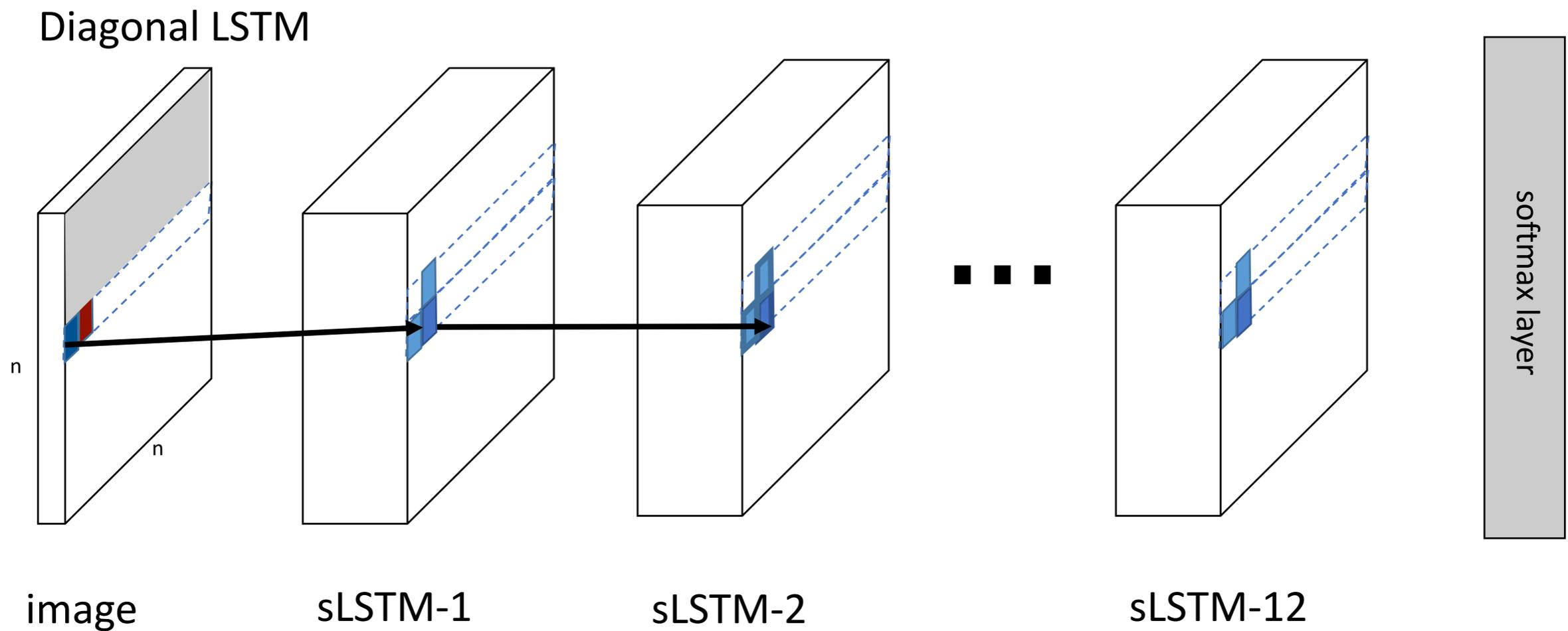


[Pixel recurrent neural networks](#), ICML 2016

# Row LSTM



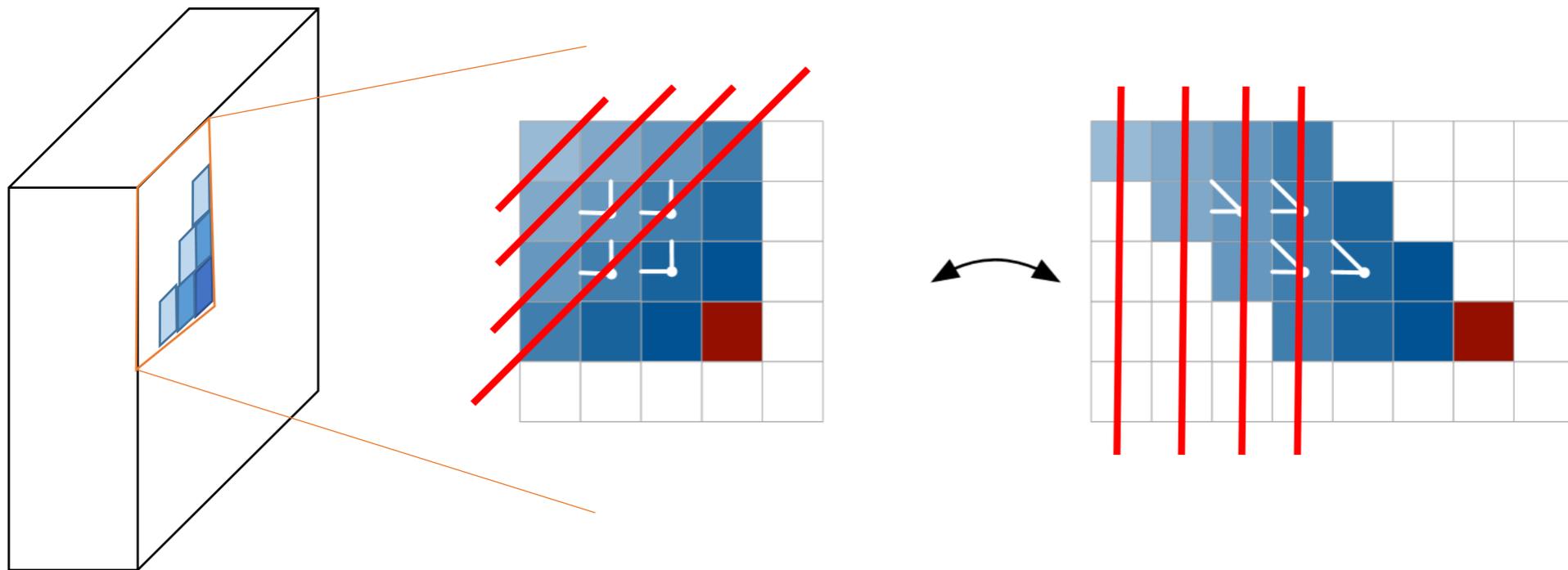
# Pixel RNN



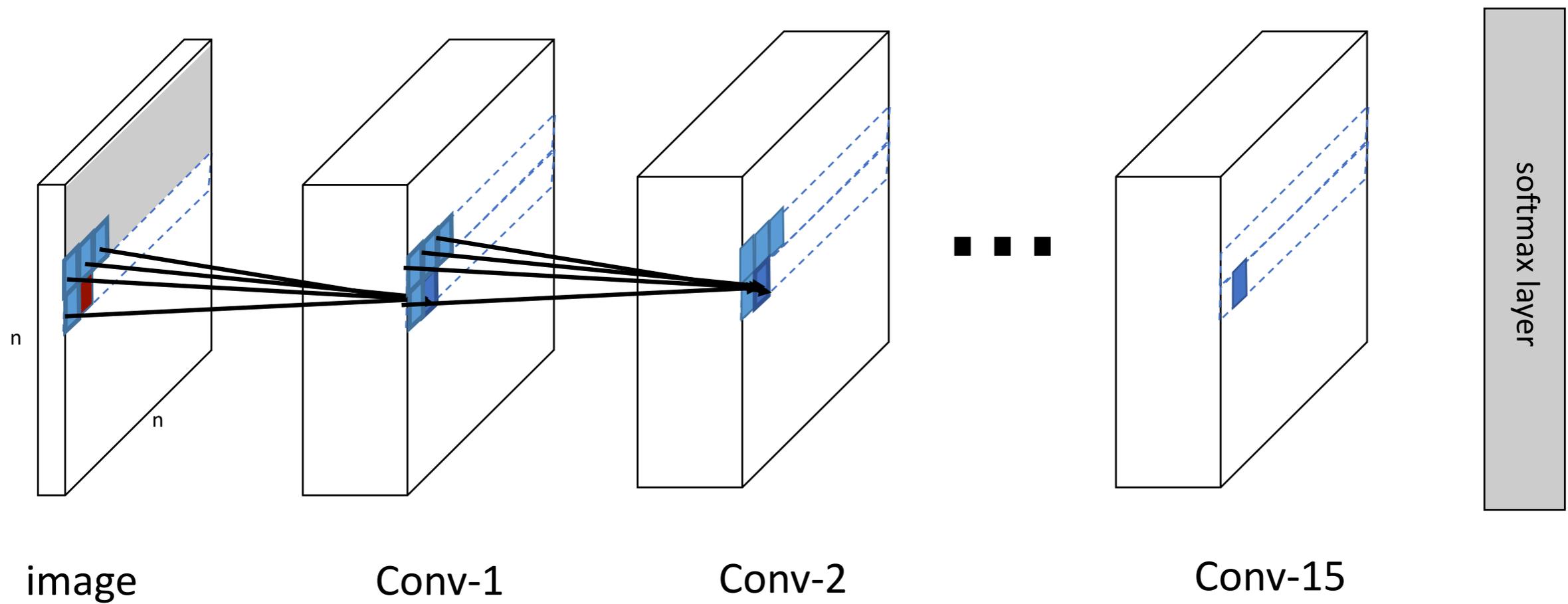
[Pixel recurrent neural networks](#), ICML 2016

# Diagonal LSTM

- To optimize, we skew the feature maps so it can be parallelized

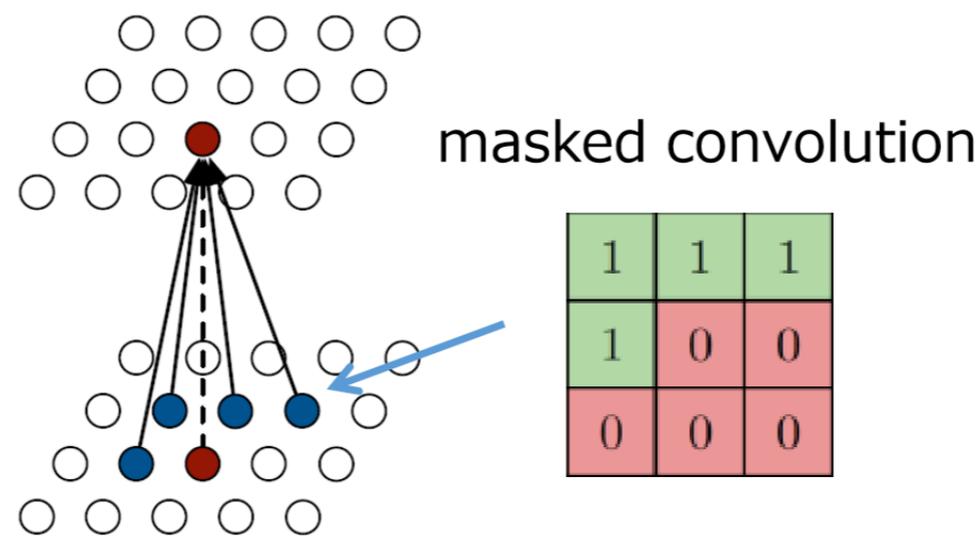


# Pixel CNN



# Pixel CNN

- 2D convolution on previous layer
- Apply masks so a pixel does not see future pixels (in sequential order)



# Comparison

PixelCNN	PixelRNN – Row LSTM	PixelRNN – Diagonal BiLSTM
Full dependency field	Triangular receptive field	Full dependency field
Fastest	Slow	Slowest
Worst log-likelihood	-	Best log-likelihood

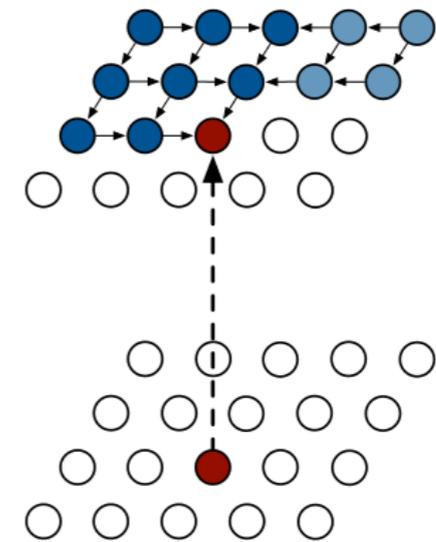
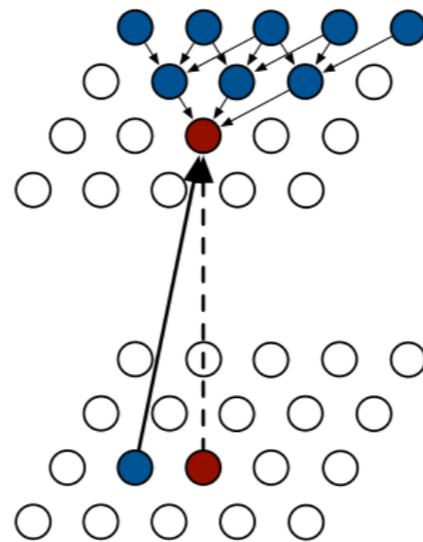
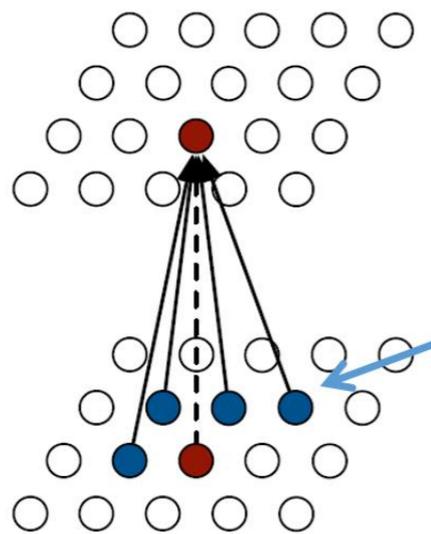
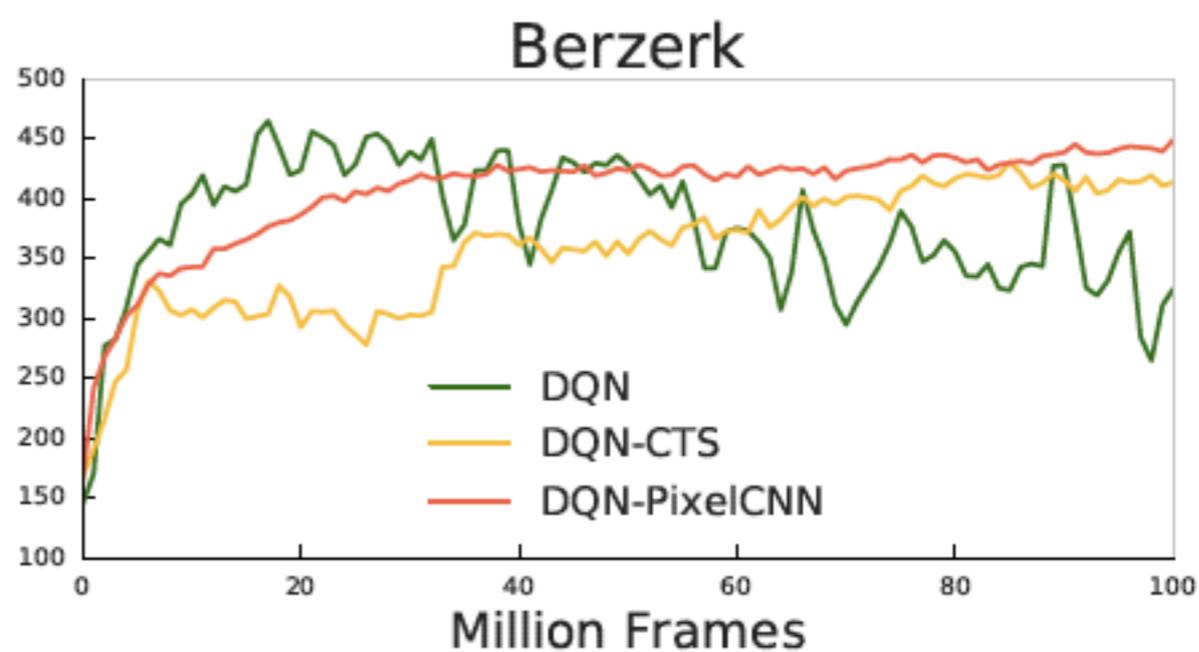
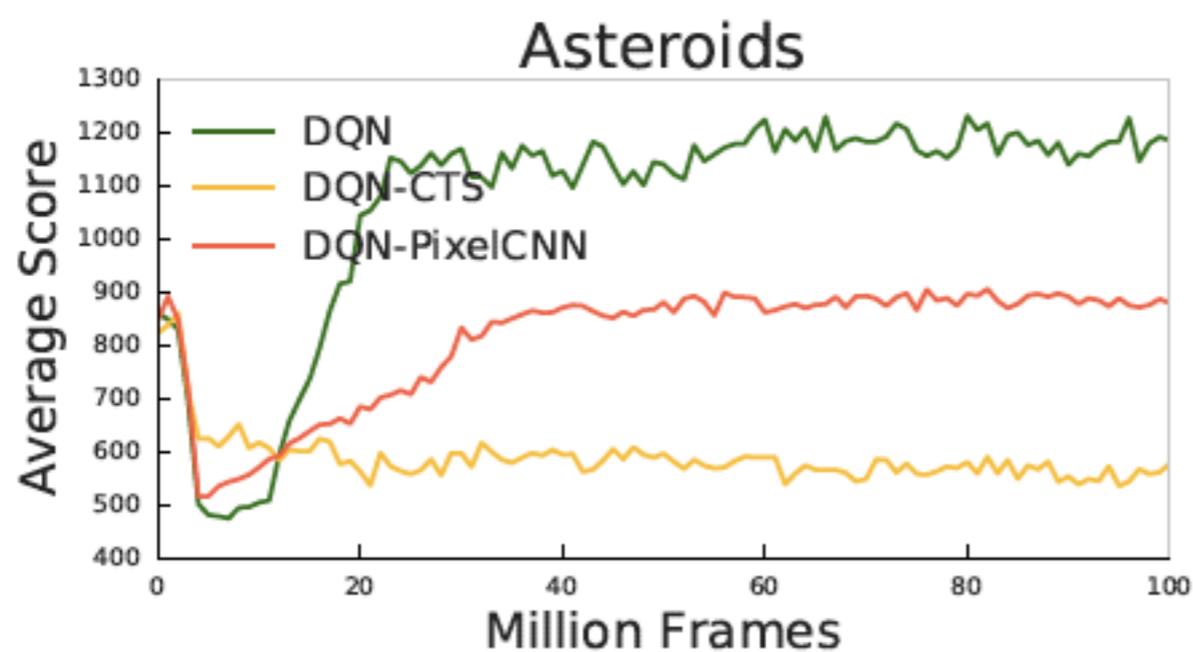
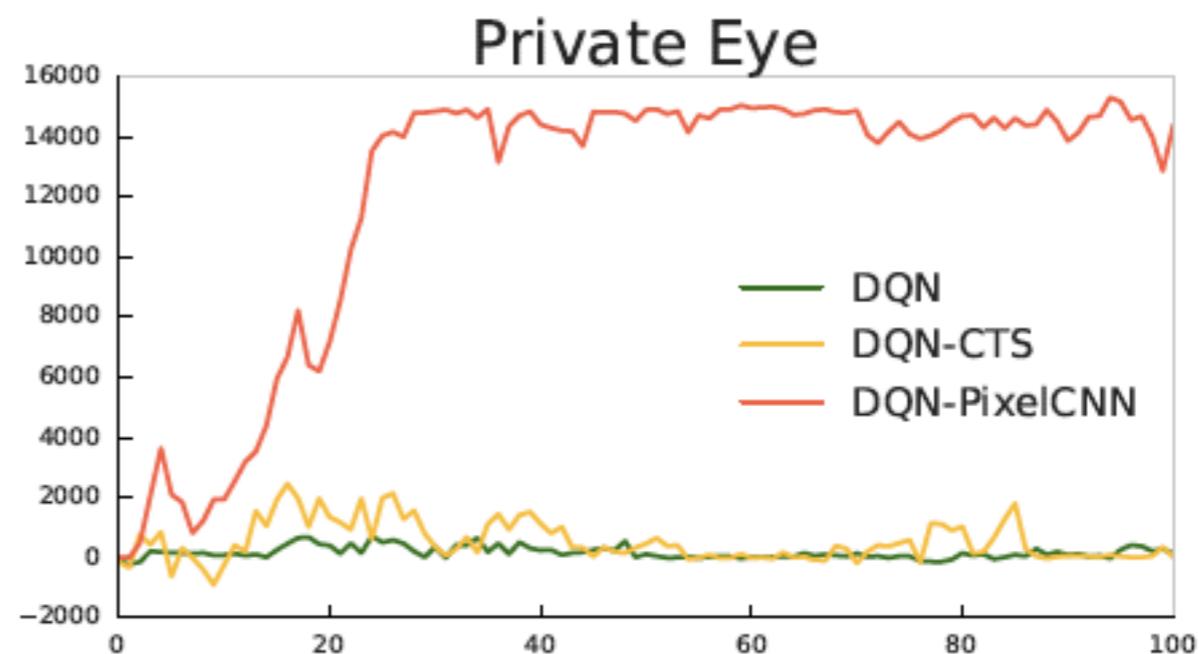
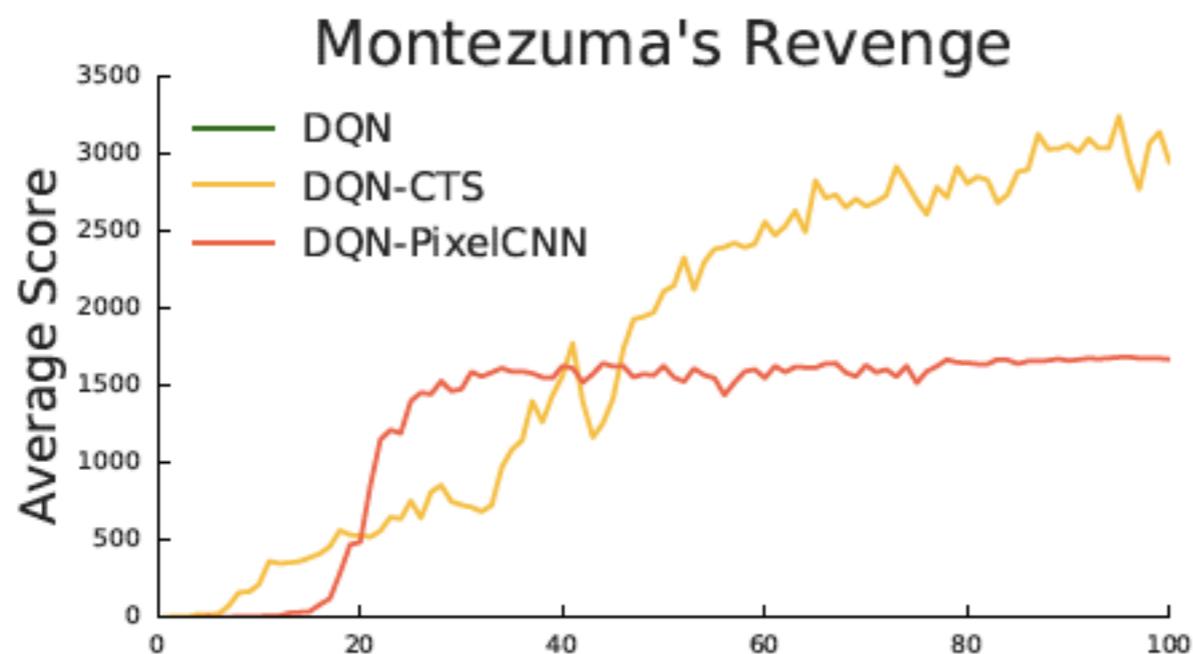


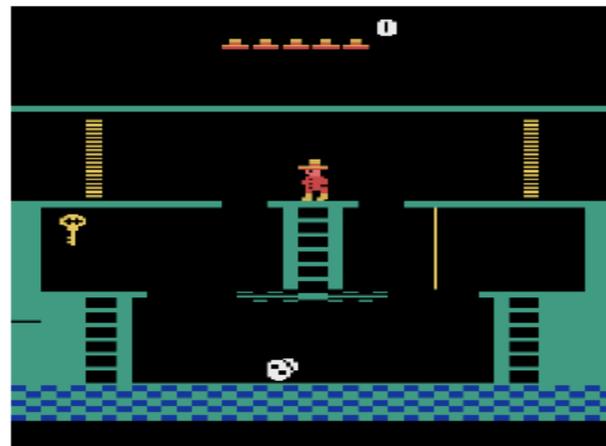
Figure from: [Oord et al.](#)

# Better density estimation usually helps

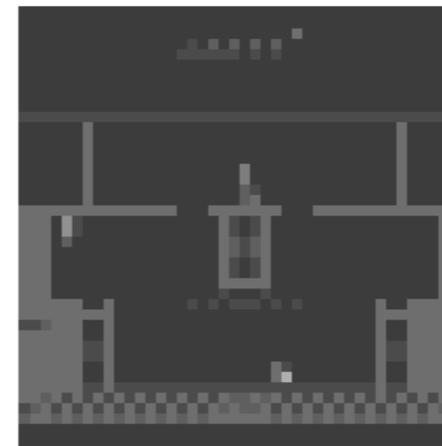


# Better density estimation helps

Frame preprocessing: shrink and convert to grayscale



Original Frame (160x210)



3-bit Greyscale (42x42)

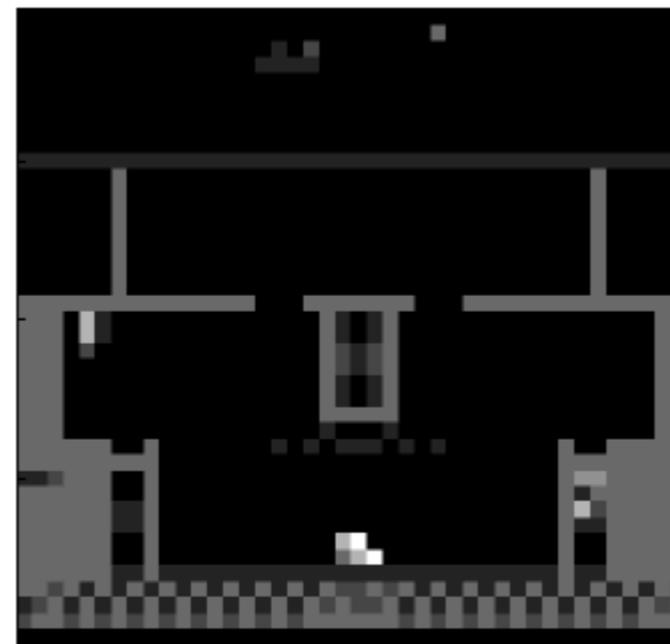
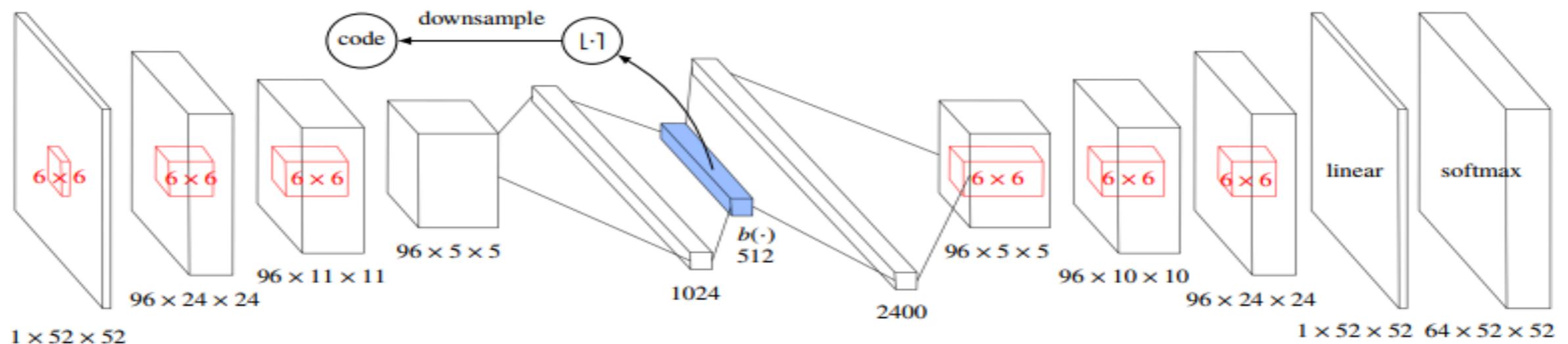


Figure 4. Samples after 25K steps. **Left:** CTS, **right:** PixelCNN.

# State Counting with DeepHashing

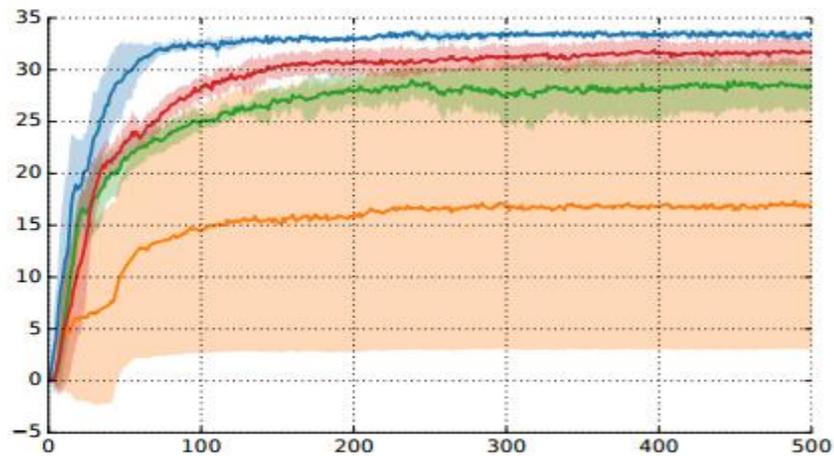
- We still count states (images) but not in pixel space, but in latent compressed space.
- Compress  $s$  into a latent code, then count occurrences of the code.
- How do we get the image encoding? E.g, using autoencoders.



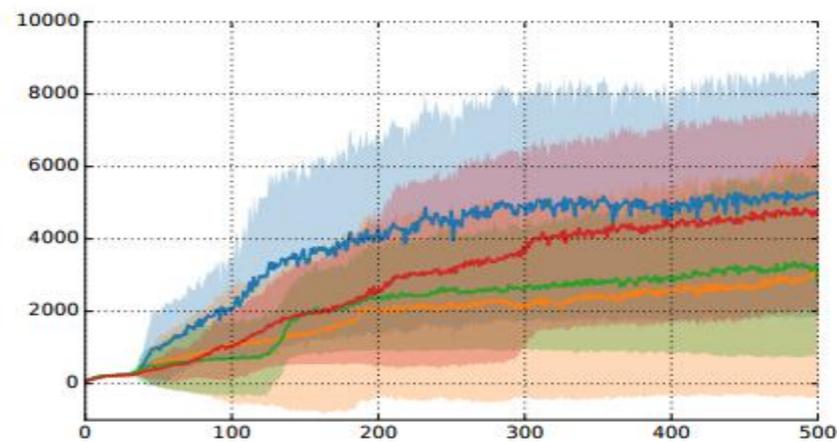
- There is no guarantee such reconstruction loss will capture the important things that make two states to be similar or not policy wise..

# State Counting with DeepHash

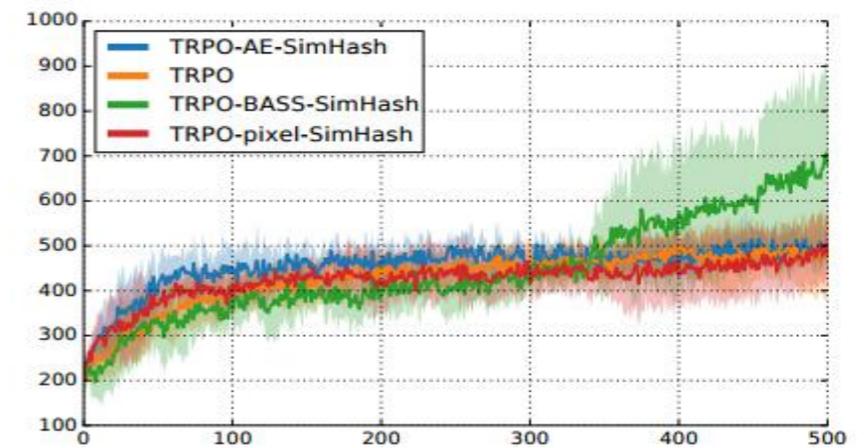
- We still count states (images) but not in pixel space, but in latent compressed space.
- Compress  $s$  into a latent code, then count occurrences of the code.
- How do we get the image encoding? E.g, using autoencoders.



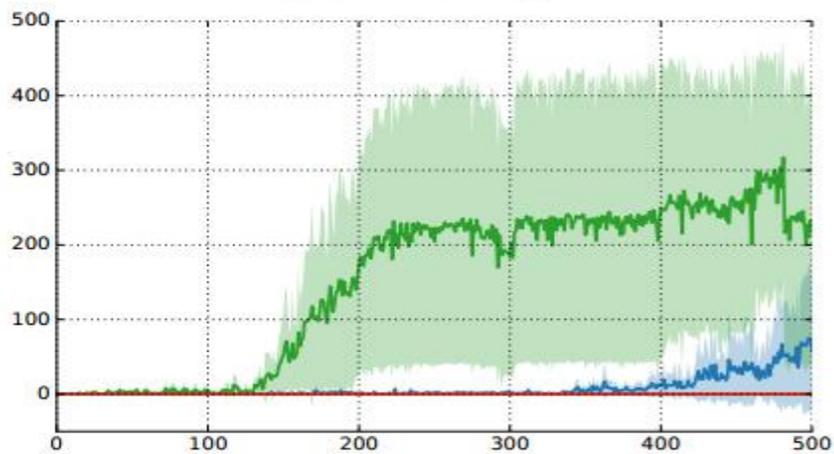
(a) Freeway



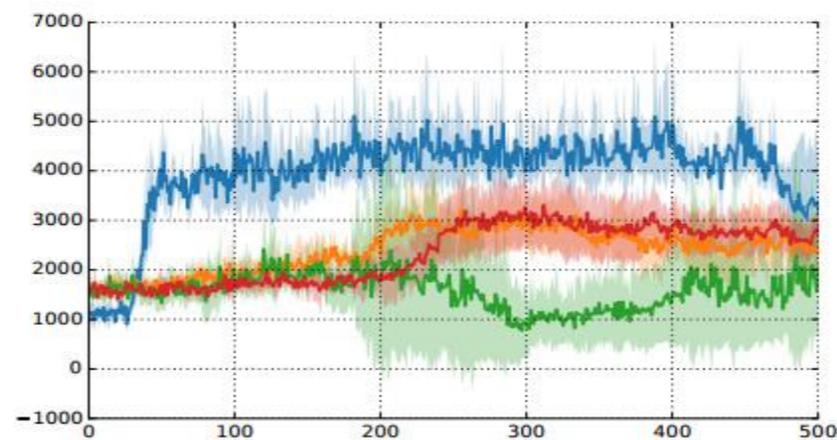
(b) Frostbite



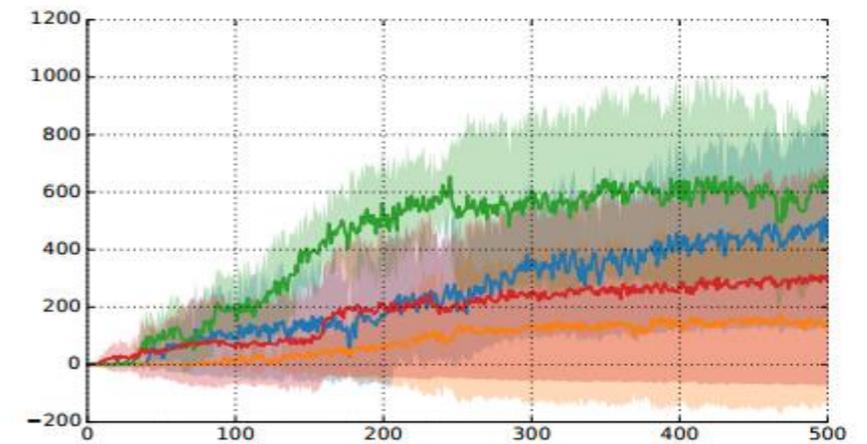
(c) Gravitar



(d) Montezuma's Revenge



(e) Solaris



(f) Venture

# Mental models



If the organism carries a `small scale model' of external reality and its own possible actions within its head, it is able try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of the past in dealing with present and the future, and in every way react in much fuller, safer and more competent manner to emergencies which face it.

-- Kenneth Craik, 1943, Chapter 5, page 61

This will come when we talk about model-baser RL, but for now we will use models for exploration!

[credit: Jitendra Malik]

# Computational Curiosity

- “The direct goal of curiosity and boredom is to improve the **world model**. The indirect goal is to ease the learning of new goal-directed action sequences.”
- “The same complex mechanism which is used for ‘normal’ goal-directed learning is used for implementing curiosity and boredom. There is no need for devising a separate system which aims at improving the world model.”
- “Curiosity Unit”: reward is a function of the **mismatch between model’s current predictions and actuality**. There is positive reinforcement whenever **the system fails to correctly predict the environment**.
- “Thus **the usual credit assignment process ... encourages certain past actions in order to repeat situations similar to the mismatch situation.**” (planning to make your (internal) world model to fail)



# Reward Prediction Error

## Seek novelty/surprise:

- Visit *novel states*  $s$  (state visitation counts)
- Observe *novel state transitions*  $(s,a) \rightarrow s'$  (improve transition dynamics)

Compute state visitation (pseudo)counts  $N(s)$

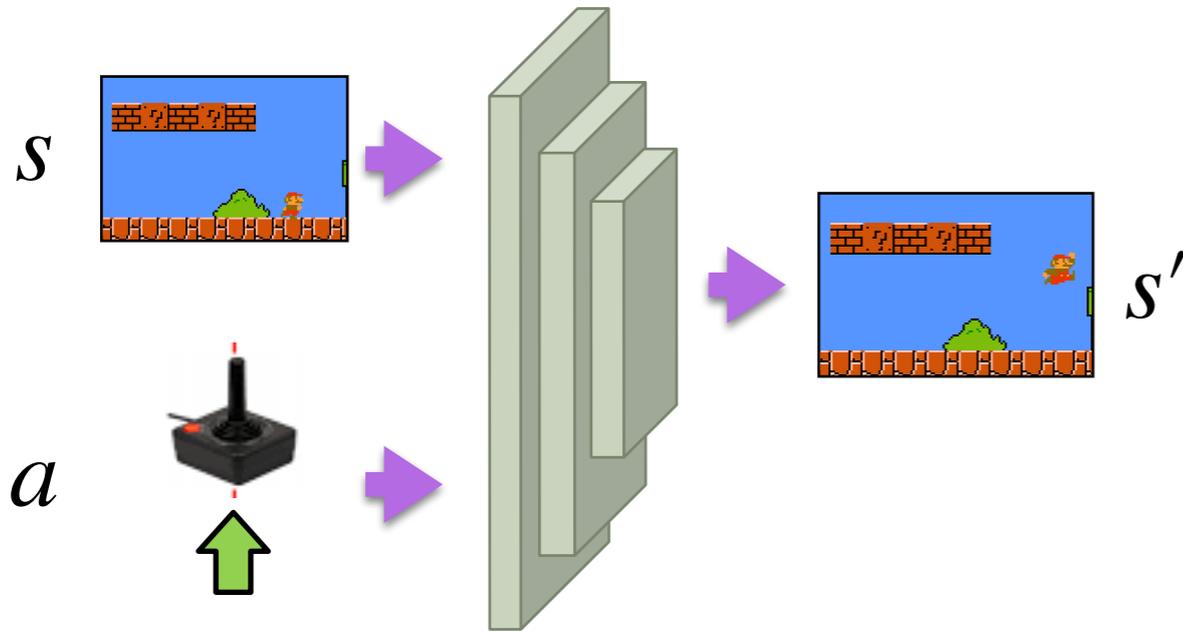
Add **exploration reward bonuses** that encourage policies to visit states that will cause the prediction model to fail.

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(\|T(s, a; \theta) - s'\|)}_{\text{intrinsic}}$$

Exploration reward bonuses **are non stationary**: as the agent interacts with the environment, what is now new and novel, becomes old and known. Many methods consider critic networks that combine Monte Carlo returns with TD.

# Learning Visual Dynamics

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(s, a; \theta) - s'\|$



$$\min_{\theta} \|T(s, a; \theta) - s'\|$$

Here we predict the visual observation!

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$$

---

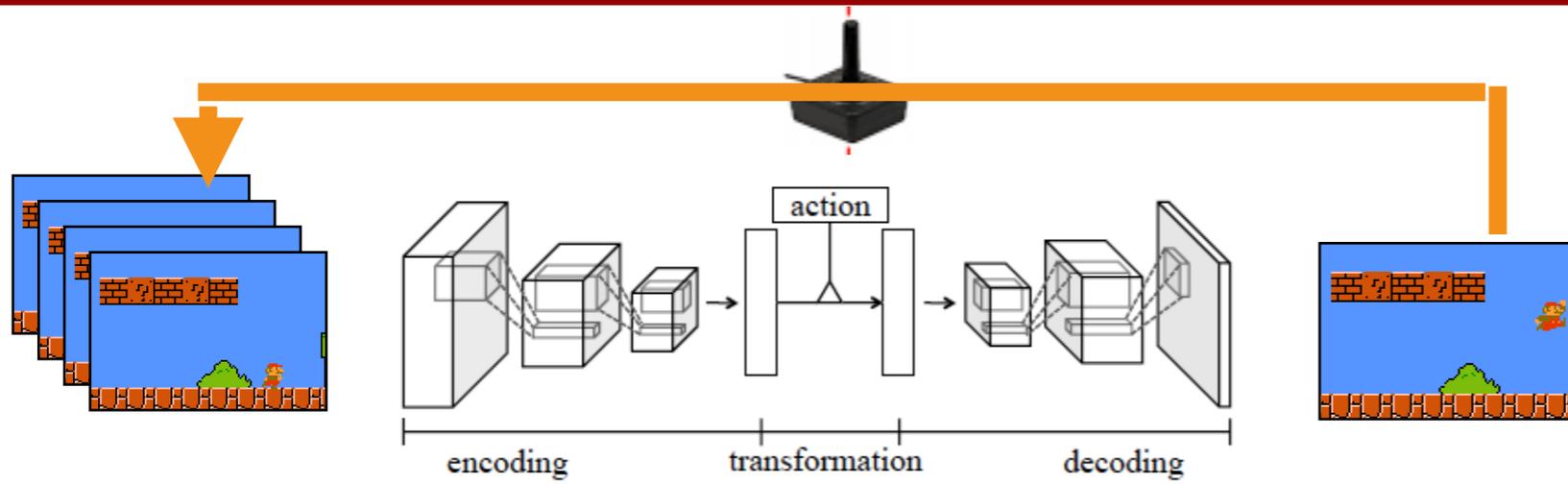
# Action-Conditional Video Prediction using Deep Networks in Atari Games

---

Junhyuk Oh   Xiaoxiao Guo   Honglak Lee   Richard Lewis   Satinder Singh

- Train a neural network that given an image (sequence) and an action, predict the pixels of the next frame
- Unroll it forward in time to predict multiple future frames
- Use this frame prediction to come up with an exploratory behavior in DQN: choose the action that leads to frames that are most dissimilar to a buffer of recent frames

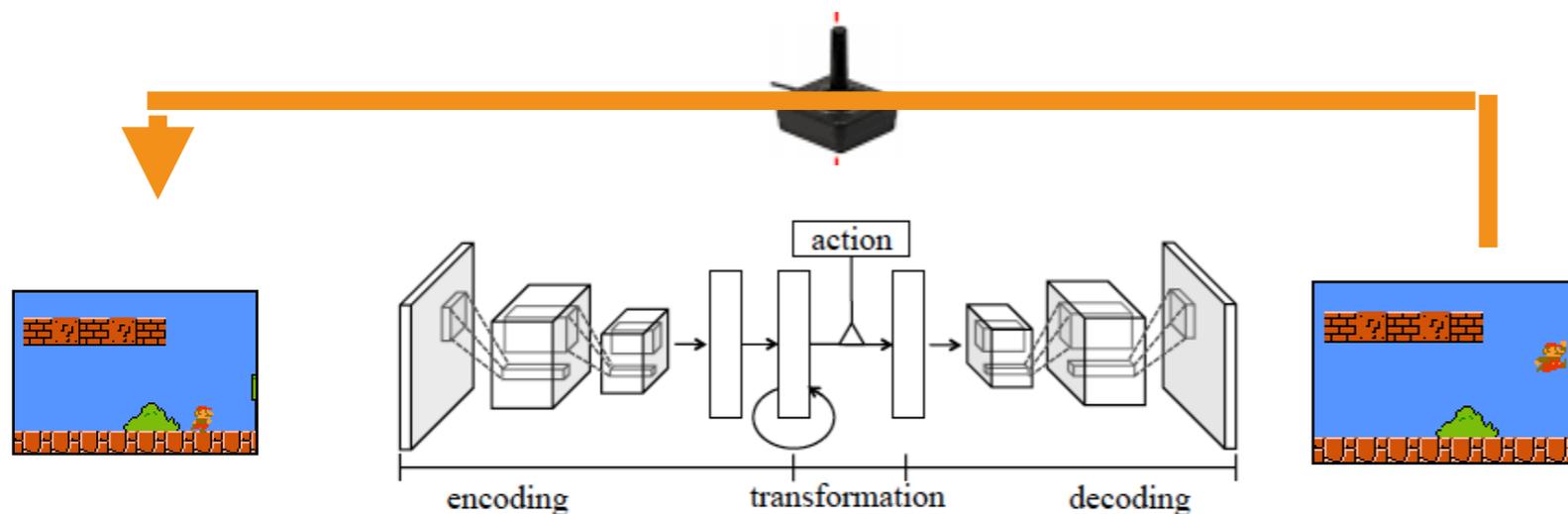
# Frame prediction



(a) Feedforward encoding

Multiplicative interactions between action and hidden state (not concatenation):

$$h_{t,i}^{dec} = \sum_{j,l} W_{ijl} h_{t,j}^{enc} a_{t,l} + b_i$$

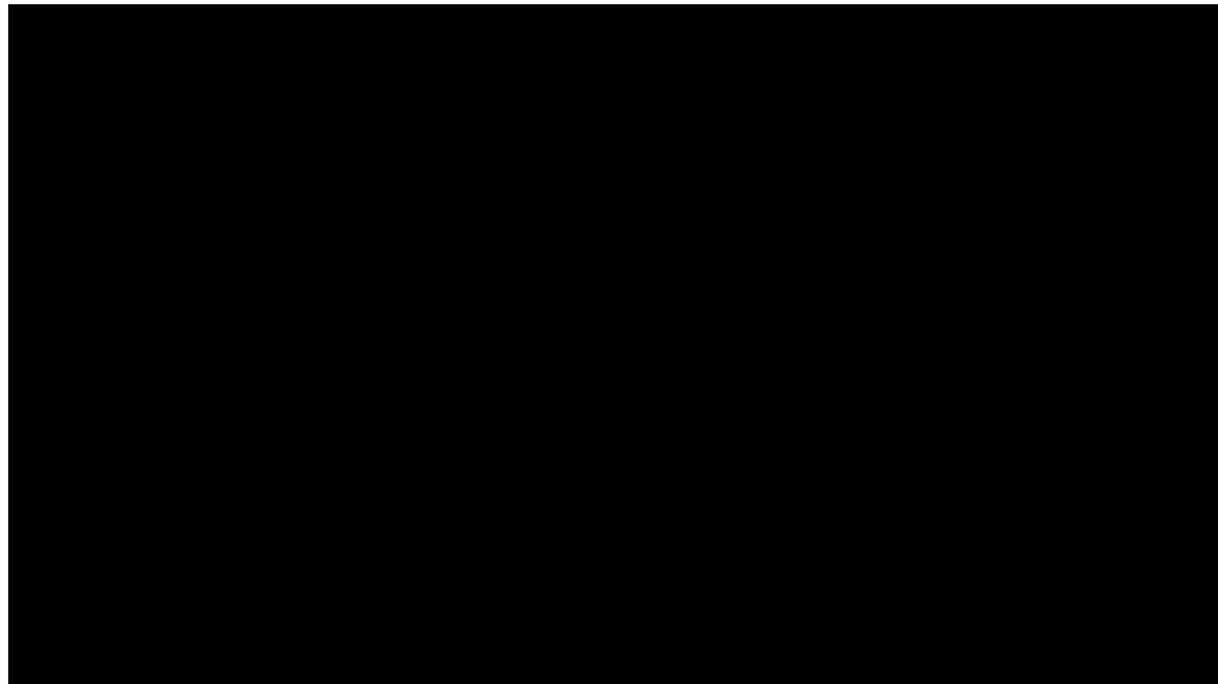


(b) Recurrent encoding

Unroll the model by feeding the prediction back as input!

Progressively increase  $k$  (the length of the conditioning history) so that we do not feed garbage predictions as input to the predictive model:

$$\mathcal{L}_K(\theta) = \frac{1}{2K} \sum_i \sum_t \sum_{k=1}^K \left\| \hat{\mathbf{x}}_{t+k}^{(i)} - \mathbf{x}_{t+k}^{(i)} \right\|^2$$



Small objects are missed, e.g., the bullets. It is because they induce a tiny mean pixel prediction loss (despite the fact they may be task-relevant)

# Frame prediction for Exploration

---

**Algorithm 1** Deep Q-learning with informed exploration

---

Allocate capacity of replay memory  $R$

**Allocate capacity of trajectory memory  $D$**

Initialize parameters  $\theta$  of DQN

**while**  $steps < M$  **do**

Reset game and observe image  $x_1$

**Store image  $x_1$  in  $D$**

Minimize similarity to a trajectory memory

**for**  $t=1$  to  $T$  **do**

Sample  $c$  from Bernoulli distribution with parameter  $\epsilon$

Set  $a_t = \begin{cases} \text{argmin}_a n_D(x_t^{(a)}) & \text{if } c = 1 \\ \text{argmax}_a Q(\phi(s_t), a; \theta) & \text{otherwise} \end{cases}$

Choose action  $a_t$ , observe reward  $r_t$  and image  $x_{t+1}$

Set  $s_{t+1} = x_{t-2:t+1}$  and preprocess images  $\phi_{t+1} = \phi(s_{t+1})$

**Store image  $x_{t+1}$  in  $D$**

Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $R$

Sample a mini-batch of transitions  $\{\phi_j, a_j, r_j, \phi_{j+1}\}$  from  $R$

Update  $\theta$  based on the mini-batch and Bellman equation

$steps = steps + 1$

**end for**

**end while**

---

Model	Seaquest	S. Invaders	Freeway	QBert	Ms Pacman
DQN - Random exploration	13119 (538)	698 (20)	30.9 (0.2)	3876 (106)	2281 (53)
DQN - Informed exploration	13265 (577)	681 (23)	32.2 (0.2)	8238 (498)	2522 (57)

---

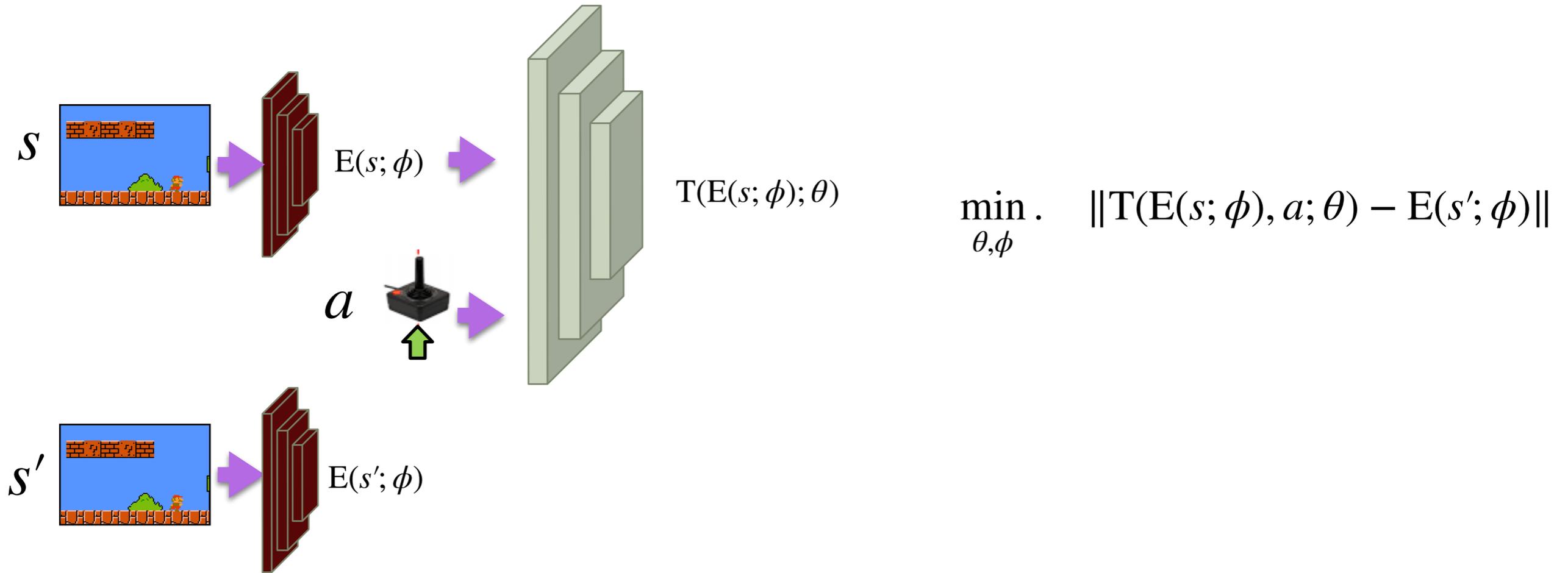
# Predicting Raw Sensory Input (Pixels)

Should our prediction model be predicting the input observations?

- **Observation prediction is difficult** especially for high dimensional observations.
- **Observation contains a lot of information unnecessary for planning**, e.g., dynamically changing backgrounds that the agent cannot control and/or are irrelevant to the reward.

# Learning Visual Dynamics

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$

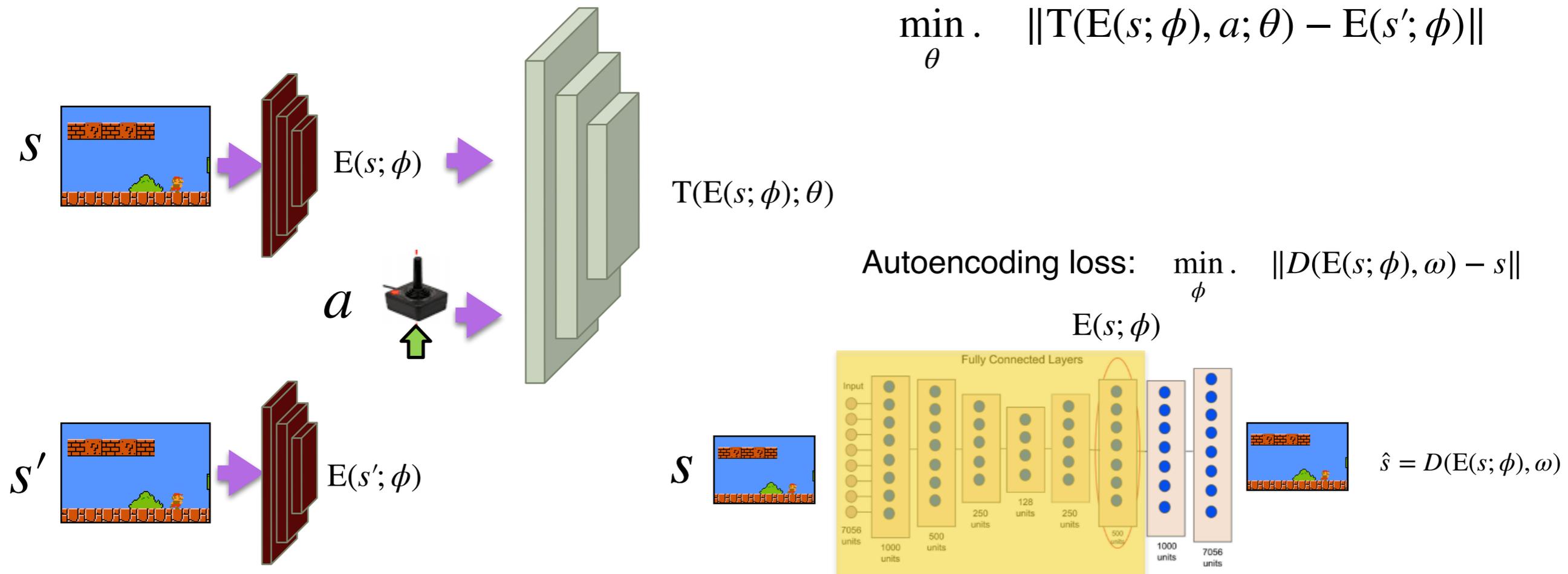


What is the problem with this optimization problem?

There is a trivial solution :-)

# Learning Visual Dynamics

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$



- Let's learn image encoding using autoencoders (to avoid the trivial solution)
- ...and suffer the problems of autoencoding reconstruction loss that has little to do with our task

# Explore guided by Novelty of Transition Dynamics

It uses the autoencoder solution.

---

**Algorithm 1** Reinforcement learning with model prediction exploration bonuses

---

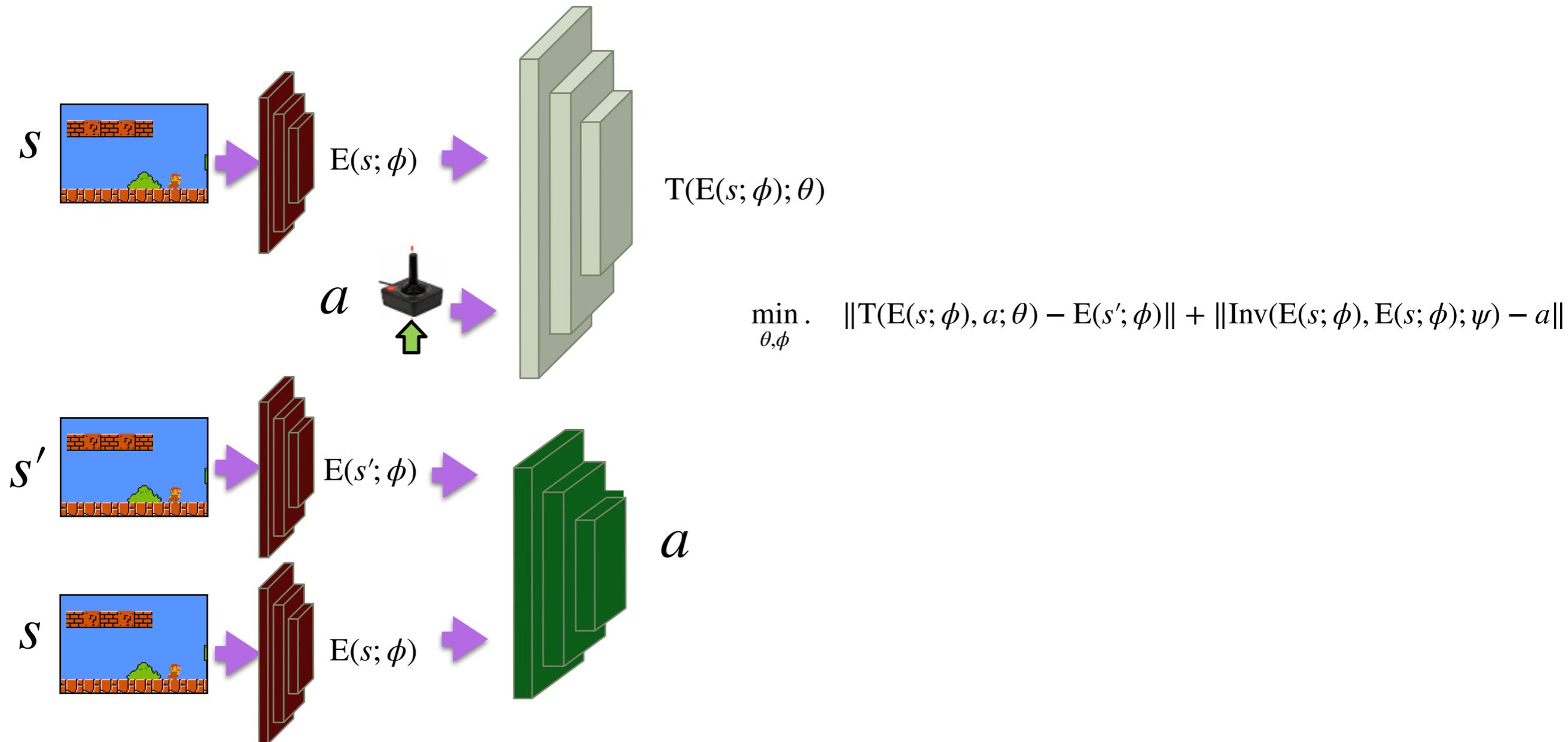
```
1: Initialize  $\max_e = 1$ , EpochLength,  $\beta$ ,  $C$ 
2: for iteration  $t$  in  $T$  do
3:   Observe  $(s_t, a_t, s_{t+1}, \mathcal{R}(s_t, a_t))$ 
4:   Encode the observations to obtain  $\sigma(s_t)$  and  $\sigma(s_{t+1})$ 
5:   Compute  $e(s_t, a_t) = \|\sigma(s_{t+1}) - \mathcal{M}_\phi(\sigma(s_t), a_t)\|_2^2$  and  $\bar{e}(s_t, a_t) = \frac{e(s_t, a_t)}{\max_e}$ .
6:   Compute  $\mathcal{R}_{\text{Bonus}}(s_t, a_t) = \mathcal{R}(s, a) + \beta \left( \frac{\bar{e}_t(s_t, a_t)}{t * C} \right)$ 
7:   if  $e(s_t, a_t) > \max_e$  then
8:      $\max_e = e(s_t, a_t)$ 
9:   end if
10:  Store  $(s_t, a_t, \mathcal{R}_{\text{bonus}})$  in a memory bank  $\Omega$ .
11:  Pass  $\Omega$  to the reinforcement learning algorithm to update  $\pi$ .
12:  if  $t \bmod \text{EpochLength} == 0$  then
13:    Use  $\Omega$  to update  $\mathcal{M}$ .
14:    Optionally, update  $\sigma$ .
15:  end if
16: end for
17: return optimized policy  $\pi$ 
```

Such reward normalization is very important!  
Because exploration rewards during training  
are non-stationary, such scale normalization  
helps accelerate learning.

The autoencoder is trained as data arrives

# Learning Visual Dynamics

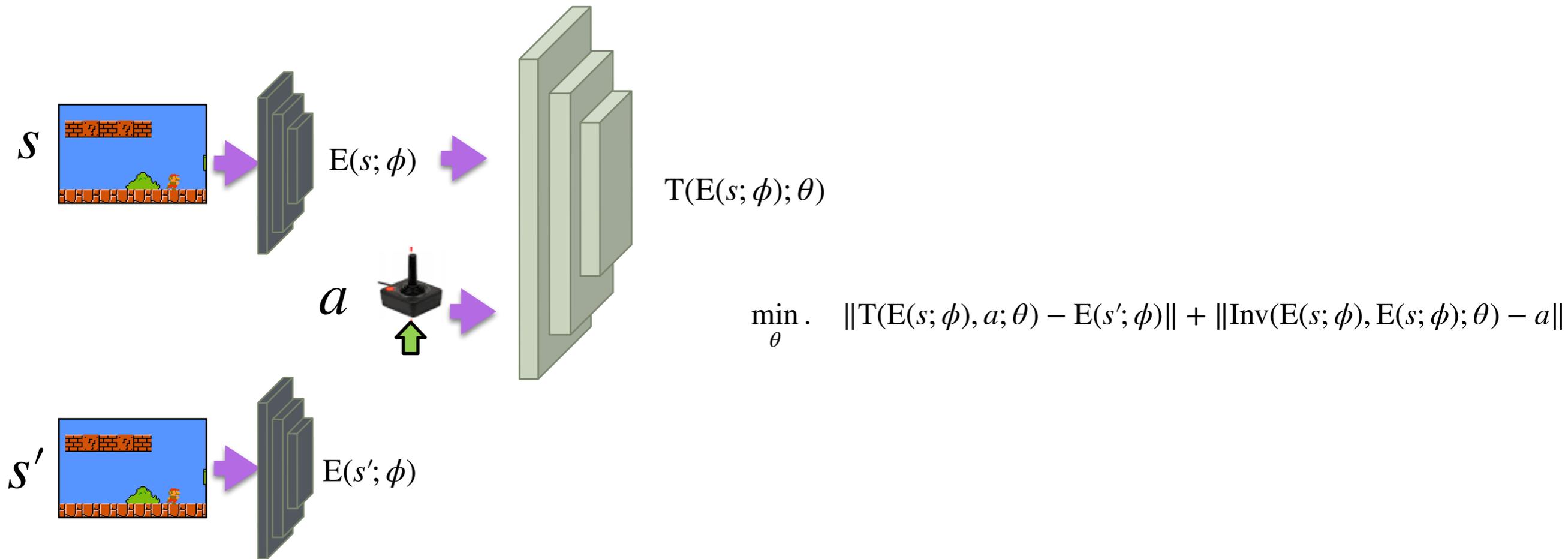
Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$



- Let's couple forward and inverse models (to avoid the trivial solution)
- ...then we will only predict things that the agent can control

# Learning Visual Dynamics

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$



- Let's use random neural networks (networks initialized randomly and frozen thereafter)
- ...and be embarrassed about how well it works on Atari games

# Task Versus Exploration rewards

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|\mathbf{T}(\mathbf{E}(s; \phi), a; \theta) - \mathbf{E}(s'; \phi)\|$

Only task reward:  $R(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}}$

Task+curiosity:  $R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$

Sparse task + curiosity:  $R^t(s, a, s') = \underbrace{r^T(s, a, s')}_{\text{extrinsic terminal}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$

Only curiosity:  $R^t(s, a, s') = \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$

# No(extrinsic)rewardRL is not new

Itti, L., Baldi, P.F.: Bayesian surprise attracts human attention. In: NIPS'05. pp. 547–554 (2006)

Schmidhuber, J.: Curious model-building control systems. In: IJCNN'91. vol. 2, pp. 1458–1463 (1991)

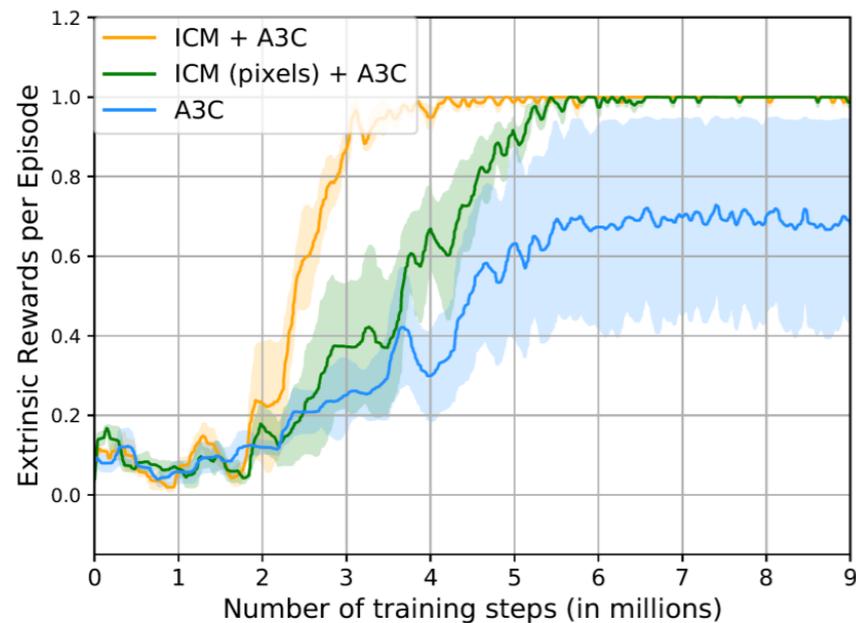
Schmidhuber, J.: Formal theory of creativity, fun, and intrinsic motivation (1990-2010). Autonomous Mental Development, IEEE Trans. on Autonomous Mental Development 2(3), 230–247 (9 2010)

Singh, S., Barto, A., Chentanez, N.: Intrinsically motivated reinforcement learning. In: NIPS'04 (2004)

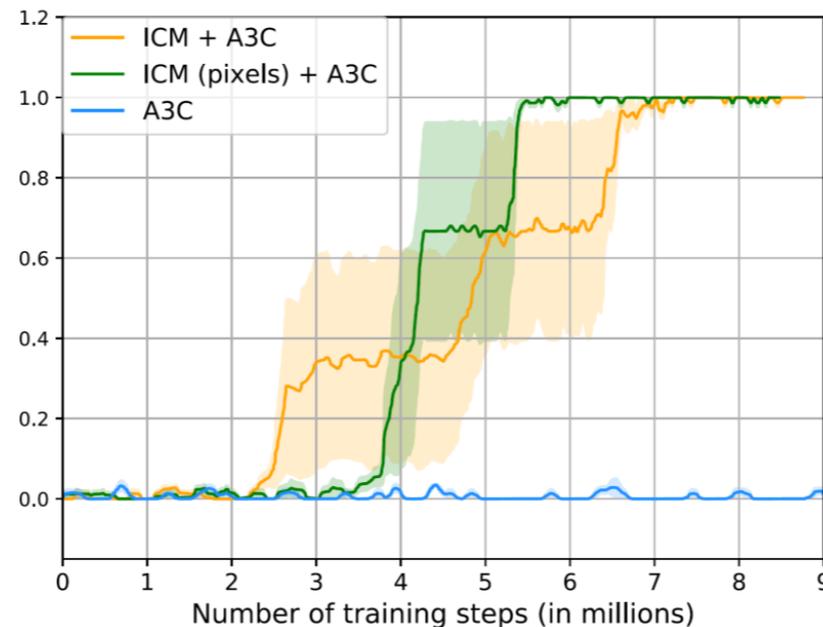
Storck, J., Hochreiter, S., Schmidhuber, J.: Reinforcement driven information acquisition in non-deterministic environments. In: ICANN'95 (1995)

Sun, Y., Gomez, F.J., Schmidhuber, J.: Planning to be surprised: Optimal bayesian exploration in dynamic environments (2011), <http://arxiv.org/abs/1103.5708>

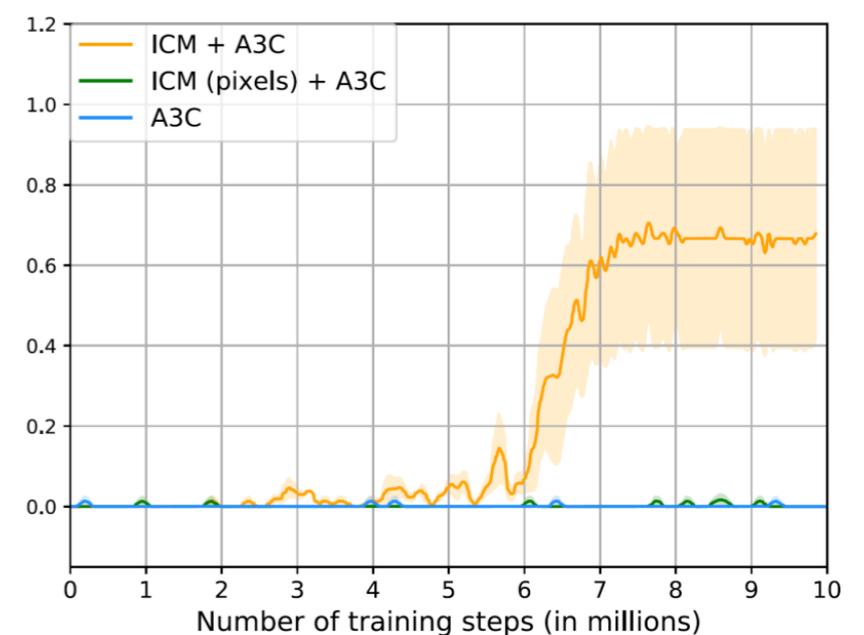
# Curiosity helps even more when rewards are sparse



(a) “dense reward” setting



(b) “sparse reward” setting



(c) “very sparse reward” setting

## Conclusions

- Using curiosity as a reward results in policies that collect much higher task rewards than policies trained under task reward alone - **so curiosity (as prediction error) a good proxy for task rewards**
- Random features do as good as learned features

# Policy Transfer

Policies trained with A3C using only curiosity rewards  
Prediction error using forward/inverse model coupling

## Trained on Level-1



## Testing on Level-2



# Limitation of Prediction Error

Agent will be rewarded even though the model cannot improve.  
So it will focus on parts of environment that are inherently unpredictable.

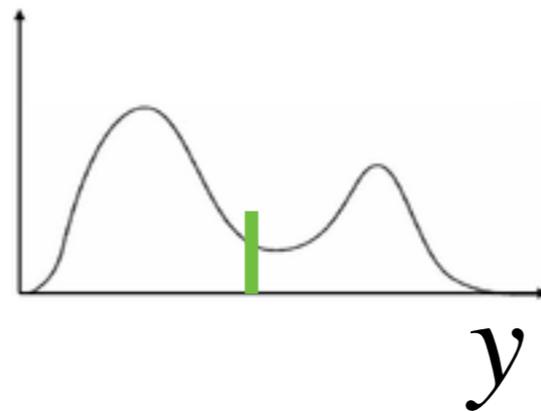
If we give the agent a TV and a remote, it becomes *a couch potato!*



The agent is attracted forever in the most noisy states, with unpredictable outcomes.

# How can we fix this?

A deterministic regression network, when faced with multimodal outputs, predicts the mean...this is the least squares solution..This will always cause our network to have **high prediction error, high surprise, high norm of the gradient, but no learning progress...**

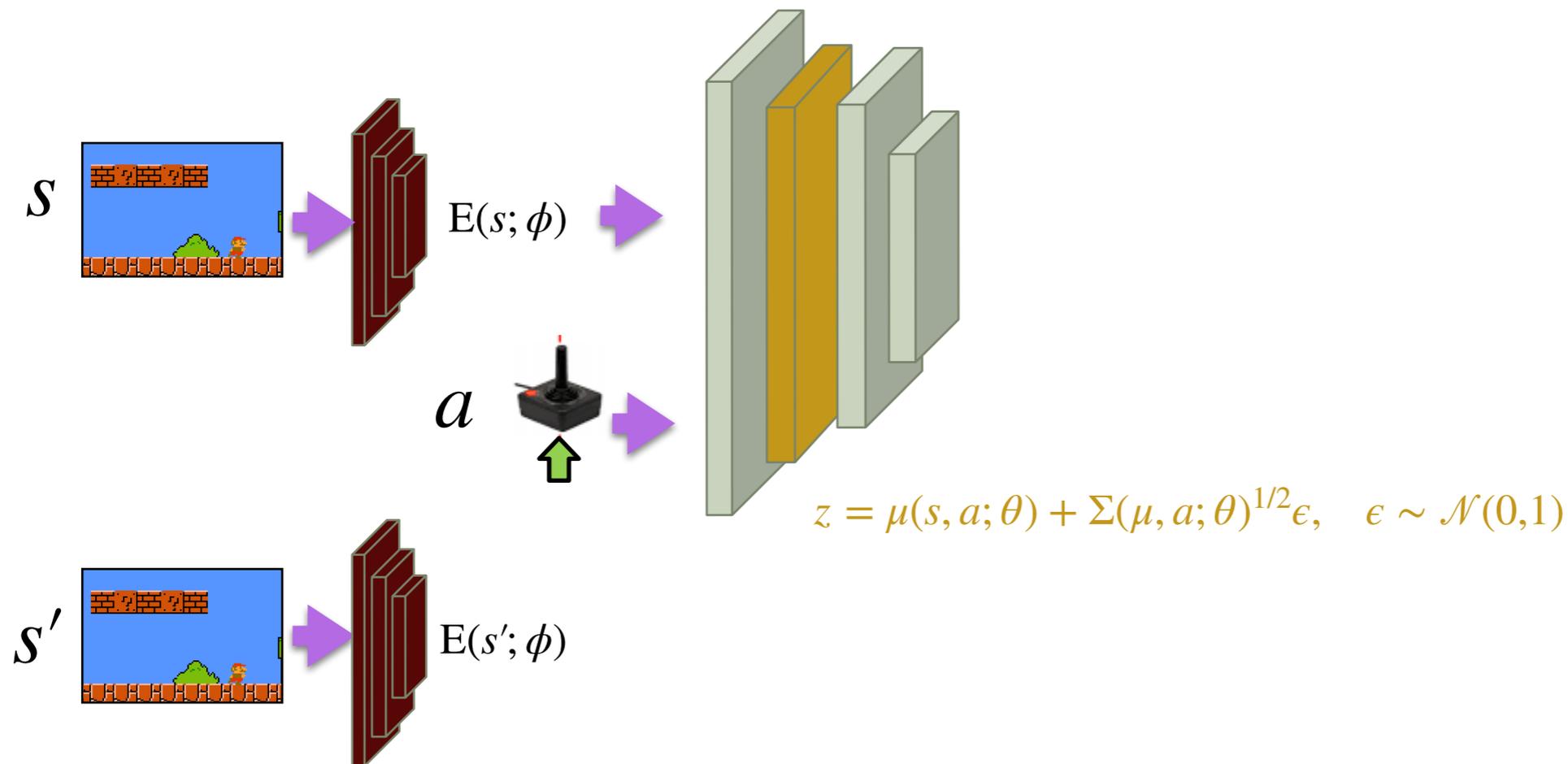


How can we handle stochasticity?

We either need to add **stochastic units in our network or stochastic weights (Bayesian deep network)**

# Learning Stochastic Visual Dynamics

We add a layer with stochastic units  $z$ , then combine those units with the rest of the network.

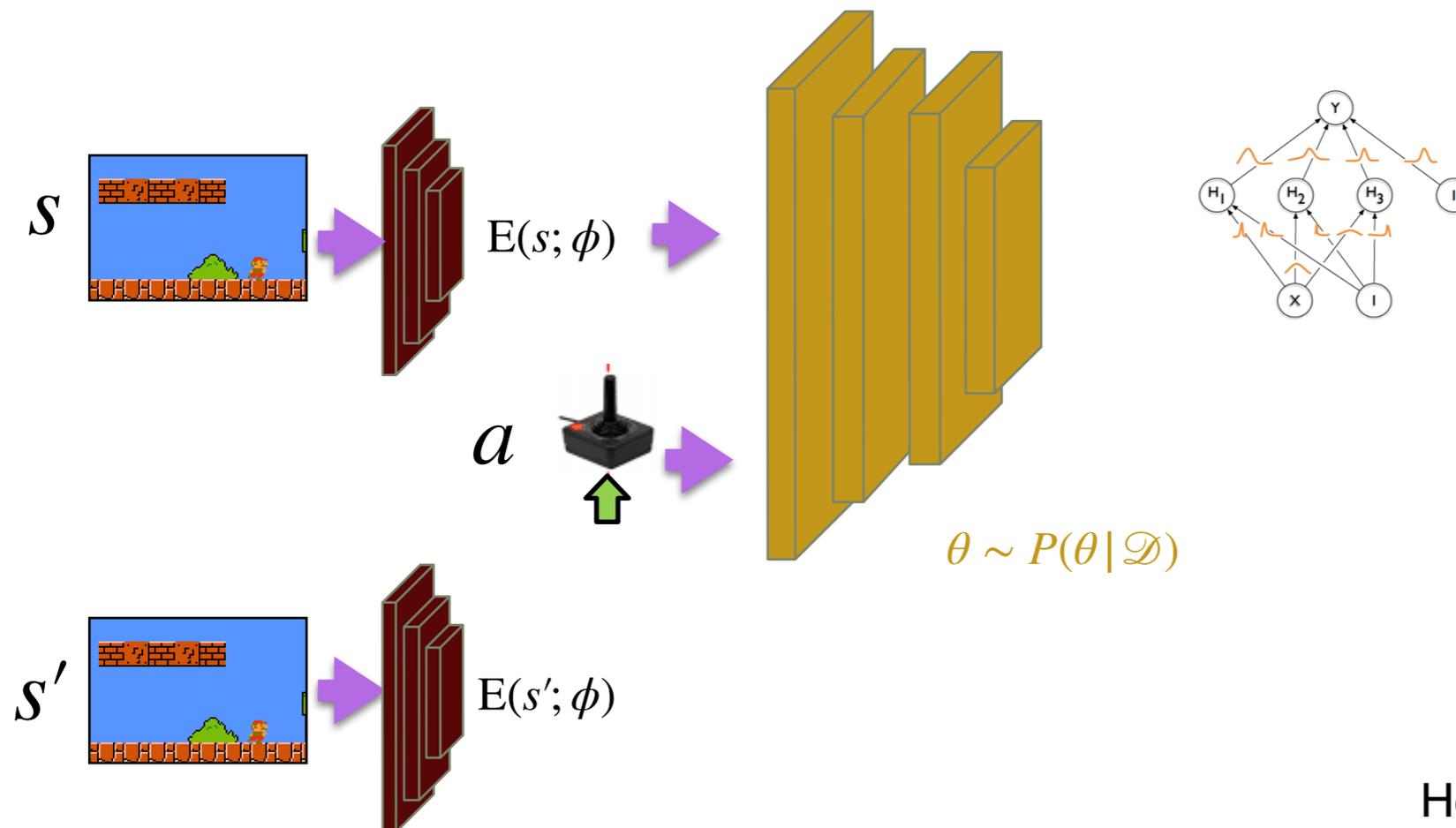


Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|\min_z T(E(s; \phi), a; \theta, z) - E(s'; \phi)\|$

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|\mathbb{E}_z T(E(s; \phi), a; \theta, z) - E(s'; \phi)\|$

# Learning Stochastic Visual Dynamics

We use stochastic weights instead of deterministic. Each weight is sampled from a distribution.

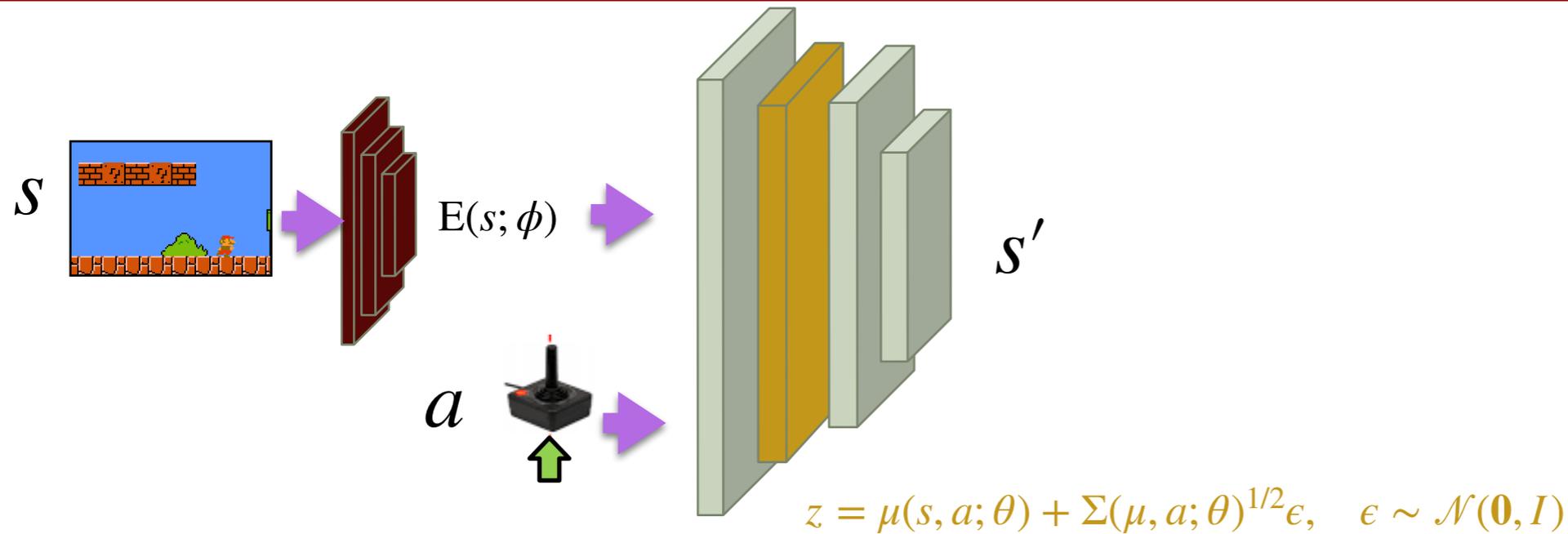


How do we train those models?

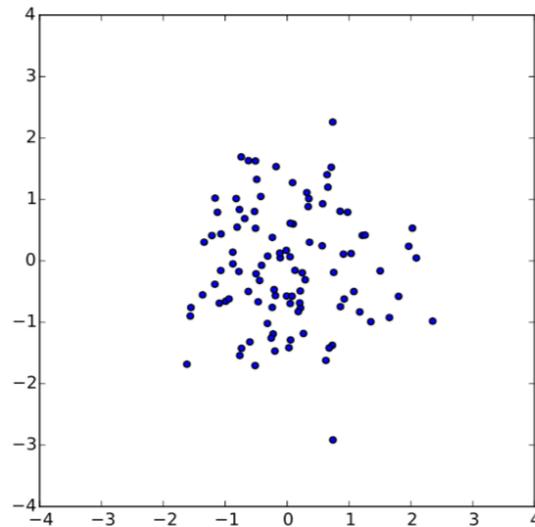
Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|\mathbb{E}_\theta T(E(s; \phi), a; \theta) - E(s'; \phi)\|$

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|\min_\theta T(E(s; \phi), a; \theta) - E(s'; \phi)\|$

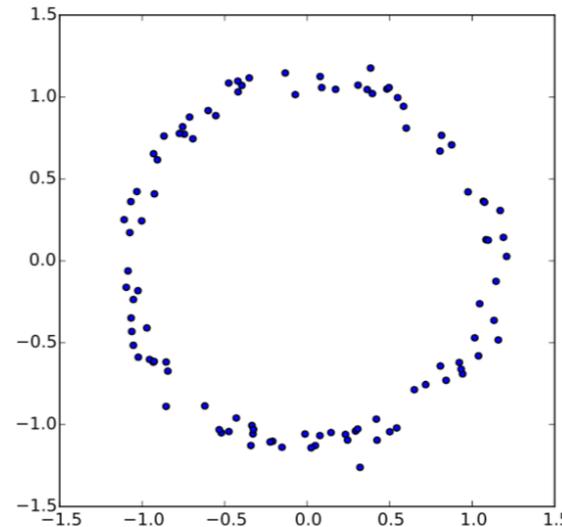
# Training Networks with Stochastic Units



Why such simple gaussian noise suffices to create complex stochastic outputs? The neural net will transform it to an arbitrarily complex distribution!

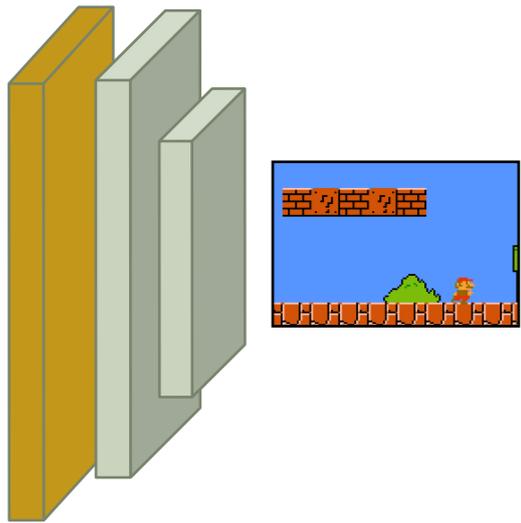


$$z \sim \mathcal{N}(\mathbf{0}, I)$$



$$f(z) = \frac{z}{10} + \frac{z}{\|z\|}$$

# Training Networks with Stochastic Units



$$z \sim \mathcal{N}(\mathbf{0}, I)$$

Let's forget for now the conditioning part and imagine we want to learn a good generative models of images.

Each sample  $z$  should give me a realistic image  $X$  once it passes through the neural network

We want to learn a mapping from  $z$  to the output  $X$ , usually we assume a Gaussian distribution to sample every pixel from:

$$P(X|z; \theta) = \mathcal{N}(X|f(z; \theta), \sigma^2 \cdot I)$$

Let's maximize data likelihood. This requires an intractable integral, too many  $z$ s..

$$\max_{\theta} . \quad P(X) = \int P(X|z; \theta)P(z)dz$$

What if we forget that it is intractable and approximate it with few samples?

$$\min_{\theta} . \quad \sum_j -\log P(X_j) = - \sum_j \sum_{z_i \sim \mathcal{N}(\mathbf{0}, I)} \log P(X_j|z; \theta) = - \sum_j \sum_{z_i \sim \mathcal{N}(\mathbf{0}, I)} \|f(z_i; \theta) - X_j\|^2$$

(we already know how to take gradients here!)

This is a bad approximation, except if we have a very large number of  $z$ s. Only few  $z$ s would produce after training reasonable  $X$ . How will we find the  $z$ s that produce good  $X$ ?

# Deep Variational Inference

Let's consider sampling  $z$ s from an alternative distribution  $Q(z)$  and try to minimize the KL between this (variational approximation) and the true posterior,  $P(z|X)$ . And because I can pick any distribution  $Q$  I like, I will also condition it on  $X$  to help inform the sampling.

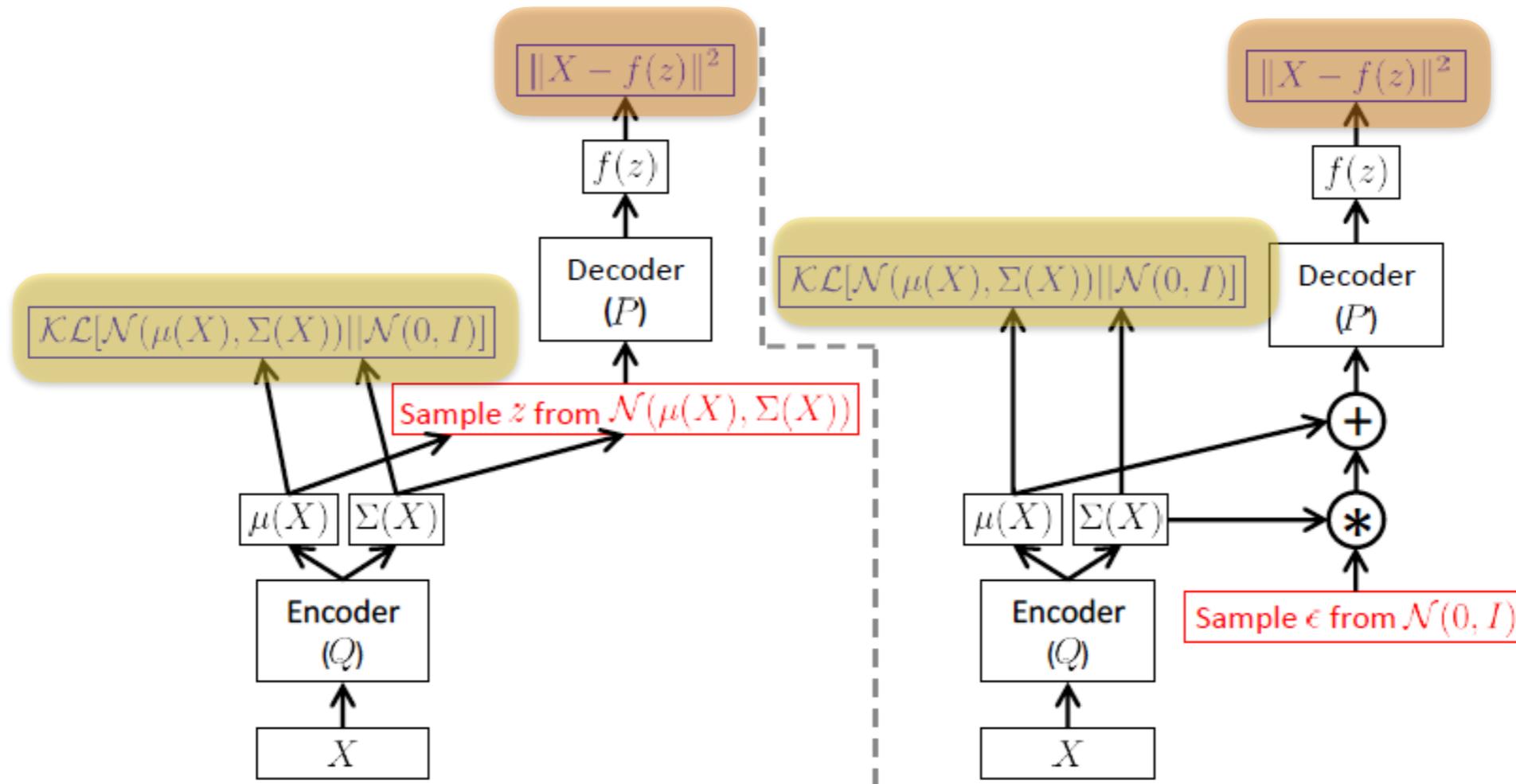
$$\begin{aligned}D_{KL}(Q(z|X) || P(z|X)) &= \int Q(z|X) \log \frac{Q(z|X)}{P(z|X)} dz \\&= \mathbb{E}_Q \log Q(z|X) - \mathbb{E}_Q \log P(z|X) \\&= \mathbb{E}_Q \log Q(z|X) - \mathbb{E}_Q \log \frac{P(X|z)P(z)}{P(X)} \\&= \mathbb{E}_Q \log Q(z|X) - \mathbb{E}_Q \log \frac{P(X|z)P(z)}{P(X)} \\&= \mathbb{E}_Q \log Q(z|X) - \mathbb{E}_Q \log P(X|z) - \mathbb{E}_Q \log P(z) + \log P(X) \\&= D_{KL}(Q(z|X) | P(z)) - \mathbb{E}_Q \log P(X|z) + \log P(X)\end{aligned}$$

$$\min_{\phi, \theta} D_{KL}(Q(z|X; \phi) || P(z)) - \mathbb{E}_Q \log P(X|z; \theta)$$

encoder decoder

# Variational Autoencoder

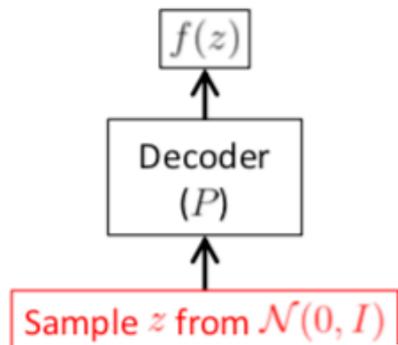
From left to right: re-parametrization trick!



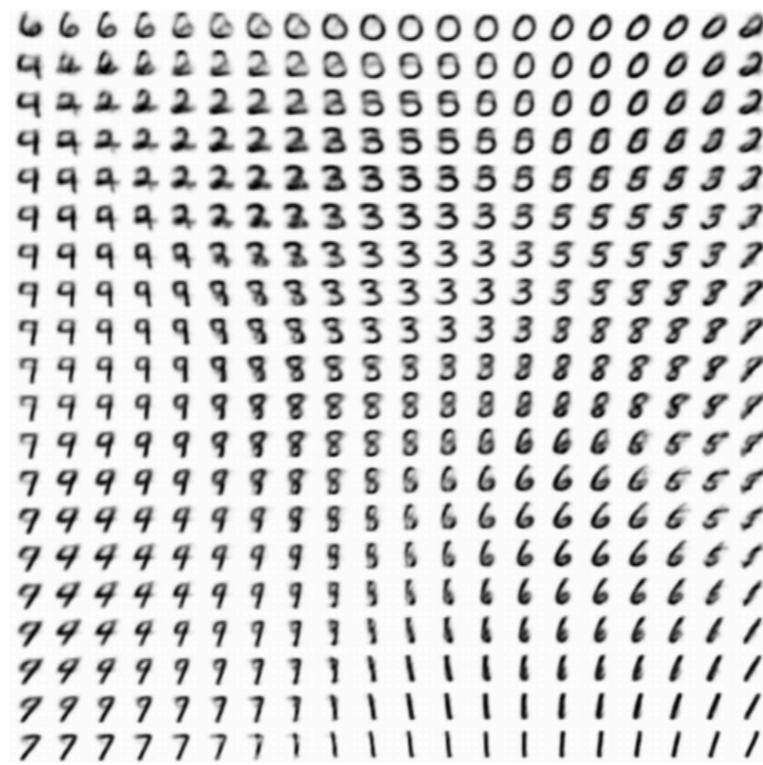
$$\min_{\phi, \theta} D_{KL}(\underbrace{Q(z | X; \phi)}_{\text{encoder}} || P(z)) - \mathbb{E}_Q \log \underbrace{P(X | z; \theta)}_{\text{decoder}}$$

# Variational Autoencoder

At test time



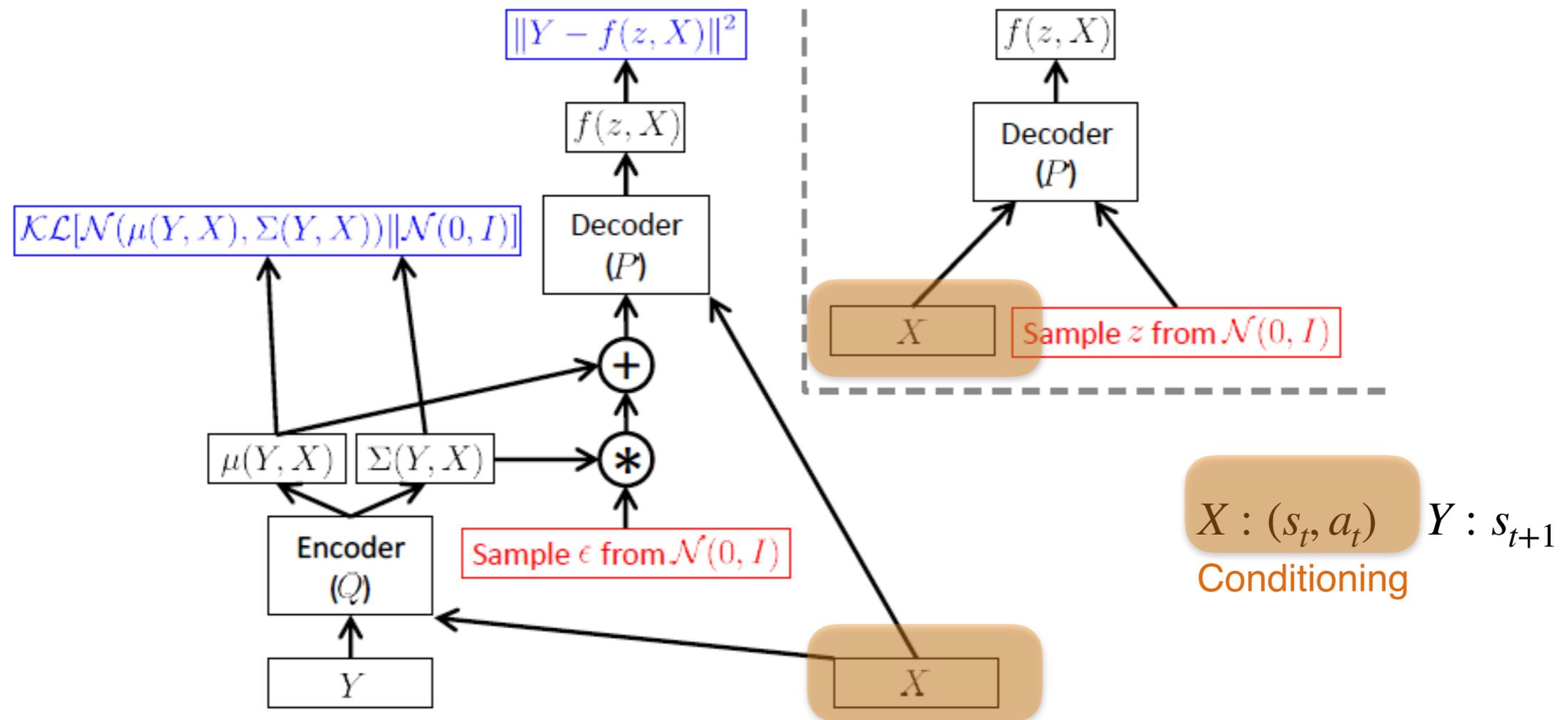
(a) Learned Frey Face manifold



(b) Learned MNIST manifold

But wait: can i use this now to sample future frames? Shouldn't my sampling be conditioned on the current state and action?

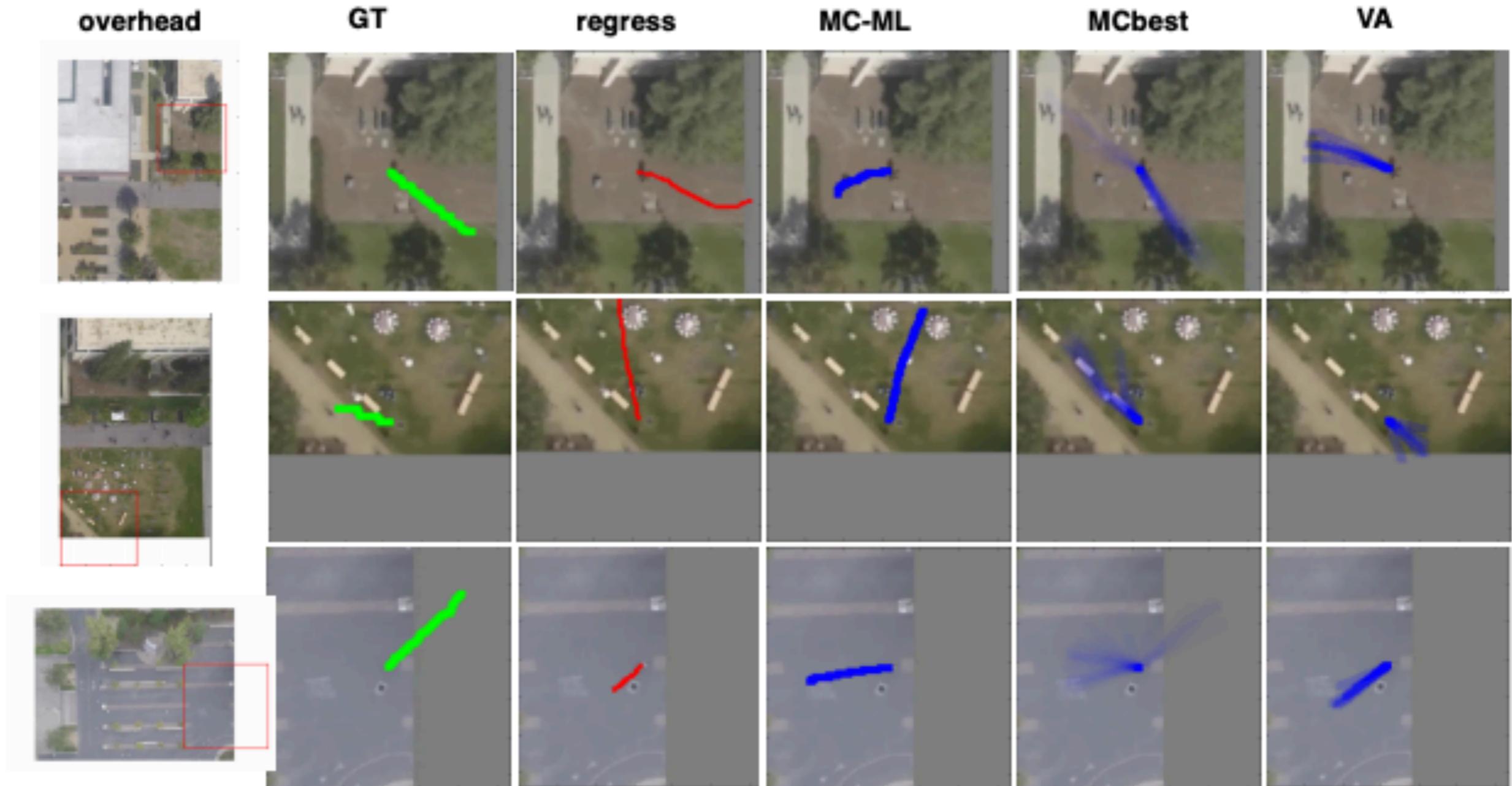
# Conditional VAE



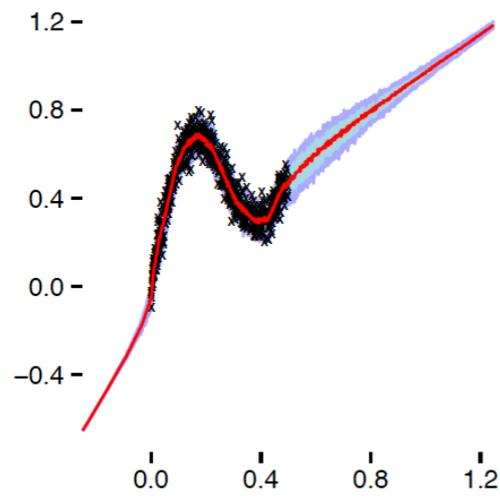
$$\min_{\phi} D_{KL}(Q(z | X, Y) \parallel P(z | \mathcal{D})) = \min_{\phi} D_{KL}(Q(z | X, Y) \parallel P(z)) - \mathbb{E}_Q \log P(\mathcal{D} | z)$$

# Conditional VAE

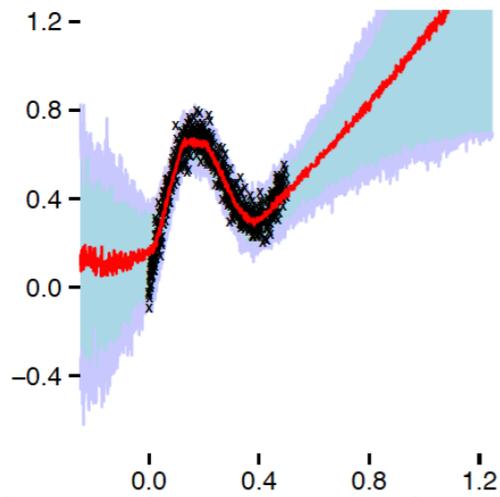
For future trajectory and frame prediction



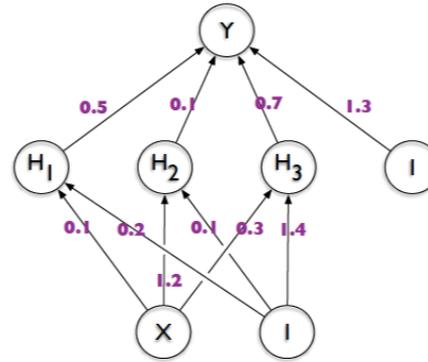
# Bayesian Deep Networks



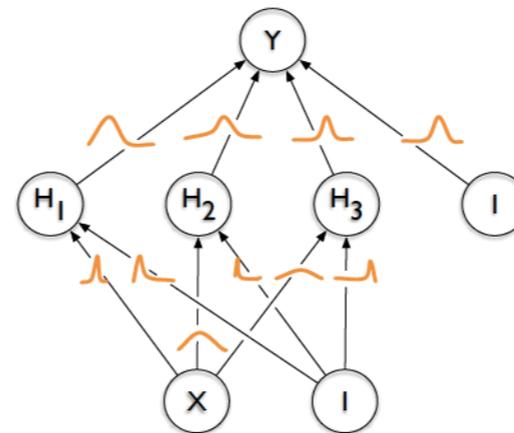
regression network



bayesian regression network



$$\begin{aligned} \mathbf{w}^{\text{MAP}} &= \arg \max_{\mathbf{w}} \log P(\mathbf{w} | \mathcal{D}) \\ &= \arg \max_{\mathbf{w}} \log P(\mathcal{D} | \mathbf{w}) + \log P(\mathbf{w}). \end{aligned}$$



$P(\mathbf{w} | \mathcal{D})$

Bayesian nets for:

- representing **uncertainty** (I have not seen this datapoint)
- representing **multimodal** outputs (I have seen this datapoint but it has had multiple different labels)

# Variational Inference for Bayesian Neural Networks

Variational approximation to the Bayesian posterior distribution of the weights.

$$\begin{aligned}D_{KL}(Q(\theta | \phi) || P(\theta | \mathcal{D})) &= \int Q(\theta | \phi) \log \frac{Q(\theta | \phi)}{P(\theta | \mathcal{D})} d\theta \\&= \mathbb{E}_Q \log Q(\theta | \phi) - \mathbb{E}_Q \log P(\theta | \mathcal{D}) \\&= \mathbb{E}_Q \log Q(\theta | \phi) - \mathbb{E}_Q \log \frac{P(\mathcal{D} | \theta) P(\theta)}{P(\mathcal{D})} \\&= \mathbb{E}_Q \log Q(\theta | \phi) - \mathbb{E}_Q \log \frac{P(\mathcal{D} | \theta) P(\theta)}{P(\mathcal{D})} \\&= \mathbb{E}_Q \log Q(\theta | \phi) - \mathbb{E}_Q \log P(\mathcal{D} | \theta) - \mathbb{E}_Q \log P(\theta) + \log P(\mathcal{D}) \\&= D_{KL}(Q(\theta | \phi) || P(\theta)) - \mathbb{E}_Q \log P(\mathcal{D} | \theta) + \log P(\mathcal{D})\end{aligned}$$

$$\min_{\phi} . \quad \underbrace{D_{KL}(Q(\theta | \phi) || P(\theta))}_{\text{weight complexity}} - \underbrace{\mathbb{E}_Q \log P(\mathcal{D} | \theta)}_{\text{data likelihood}}$$

# Variational Inference for Bayesian Neural Networks

Variational approximation to the Bayesian posterior distribution of the weights.

$$\min_{\phi} D_{KL}(Q(\theta; \phi) || P(\theta | \mathcal{D})) = \min_{\phi} \underbrace{D_{KL}(Q(\theta | \phi) || P(\theta))}_{\text{weight complexity}} - \underbrace{\mathbb{E}_Q \log P(\mathcal{D} | \theta)}_{\text{data likelihood}}$$

We will consider Q to be a diagonal gaussian distribution:  $\phi = (\mu, \sigma)$

We will consider prior P(\theta) to be a mixture of 0 mean gaussians:

$$P(\theta) = \prod_k \pi \mathcal{N}(\theta_k | 0, \sigma_1^2) + (1 - \pi) \mathcal{N}(\theta_k | 0, \sigma_2^2), \quad \pi, \sigma_1, \sigma_2 \text{ are chosen and fixed}$$

Let's try to take gradients:

$$\begin{aligned} \nabla_{\phi} \left( D_{KL}(Q(\theta | \phi) || P(\theta)) - \mathbb{E}_Q \log P(\mathcal{D} | \theta) \right) &= \nabla_{\phi} \left( \mathbb{E}_{Q(\theta | \phi)} \log \frac{Q(\theta | \phi)}{P(\theta)} - \mathbb{E}_{Q(\theta | \phi)} \log P(\mathcal{D} | \theta) \right) \\ &= \nabla_{\phi} \mathbb{E}_{Q(\theta | \phi)} \left( \log \frac{Q(\theta | \phi)}{P(\theta)} - \log P(\mathcal{D} | \theta) \right) \end{aligned}$$

The parameter is in the distribution! Reparametrization to the rescue:

$$\theta = t(\phi, \epsilon) = \mu\epsilon + \sigma, \quad \epsilon \sim \mathcal{N}(0, I)$$

# Variational Inference for Bayesian Neural Networks

We will consider  $Q$  to be a diagonal gaussian distribution:  $\phi = (\mu, \sigma)$

We will consider prior  $P(\theta)$  to be a mixture of 0 mean gaussians:

$$P(\theta) = \prod_k \pi \mathcal{N}(\theta_k | 0, \sigma_1^2) + (1 - \pi) \mathcal{N}(\theta_k | 0, \sigma_2^2), \quad \pi, \sigma_1, \sigma_2 \text{ are chosen and fixed}$$

Let's try to take gradients:

$$\begin{aligned} \nabla_{\phi} \left( D_{KL}(Q(\theta | \phi) || P(\theta)) - \mathbb{E}_Q \log P(\mathcal{D} | \theta) \right) &= \nabla_{\phi} \left( \mathbb{E}_{Q(\theta | \phi)} \log \frac{Q(\theta | \phi)}{P(\theta)} - \mathbb{E}_{Q(\theta | \phi)} \log P(\mathcal{D} | \theta) \right) \\ &= \nabla_{\phi} \mathbb{E}_{Q(\theta | \phi)} \left( \log \frac{Q(\theta | \phi)}{P(\theta)} - \log P(\mathcal{D} | \theta) \right) \end{aligned}$$

The parameter is in the distribution! Reparametrization to the rescue:

$$\theta = t(\phi, \epsilon) = \mu \epsilon + \sigma, \quad \epsilon \sim \mathcal{N}(0, I)$$

1. Sample  $\epsilon$
2. Form  $\theta$
3. Take gradients w.r.t.  $\phi$
4. Update  $\phi$

# Variational Inference for Bayesian Neural Networks

We will consider  $Q$  to be a diagonal gaussian distribution:  $\phi = (\mu, \sigma)$

We will consider prior  $P(\theta)$  to be a mixture of 0 mean gaussians:

$$P(\theta) = \prod_k \pi \mathcal{N}(\theta_k | 0, \sigma_1^2) + (1 - \pi) \mathcal{N}(\theta_k | 0, \sigma_2^2), \quad \pi, \sigma_1, \sigma_2 \text{ are chosen and fixed}$$

Let's try to take gradients:

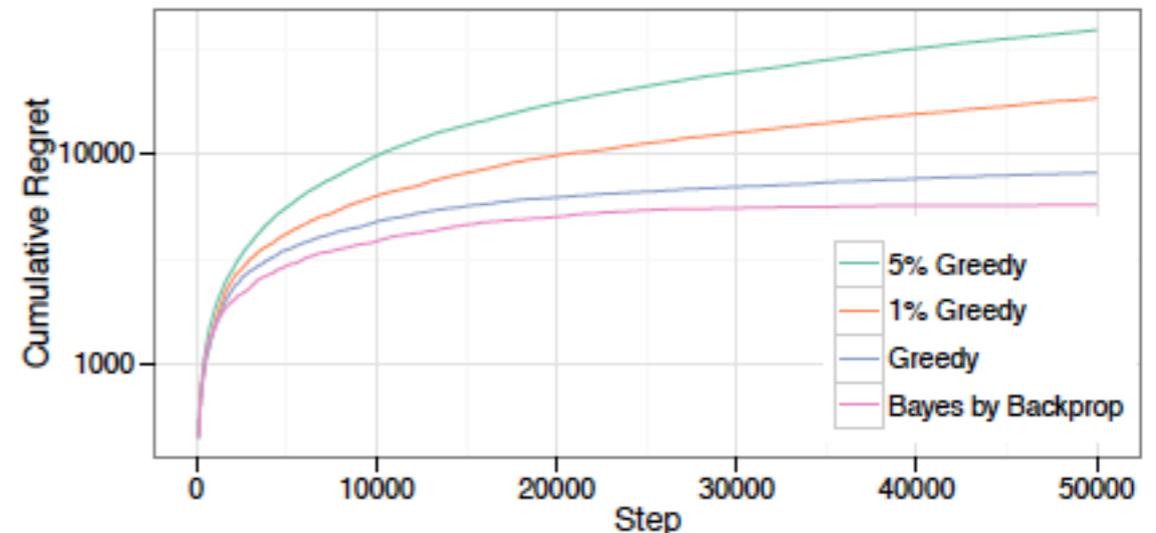
$$\begin{aligned} \nabla_{\phi} \left( D_{KL}(Q(\theta | \phi) || P(\theta)) - \mathbb{E}_Q \log P(\mathcal{D} | \theta) \right) &= \nabla_{\phi} \left( \mathbb{E}_{Q(\theta | \phi)} \log \frac{Q(\theta | \phi)}{P(\theta)} - \mathbb{E}_{Q(\theta | \phi)} \log P(\mathcal{D} | \theta) \right) \\ &= \nabla_{\phi} \mathbb{E}_{Q(\theta | \phi)} \left( \log \frac{Q(\theta | \phi)}{P(\theta)} - \log P(\mathcal{D} | \theta) \right) \end{aligned}$$

The parameter is in the distribution! Reparametrization to the rescue:

$$\theta = t(\phi, \epsilon) = \mu \epsilon + \sigma, \quad \epsilon \sim \mathcal{N}(0, I)$$

1. Sample  $\epsilon$
2. Form  $\theta$
3. Take gradients w.r.t.  $\phi$
4. Update  $\phi$

They used it for Thompson sampling!



# Reward Learning Progress

- Instead of rewarding prediction errors, **reward prediction improvements**.
- “My adaptive explorer continually wants ... to focus on those **novel things that seem easy to learn**, given current knowledge. It wants to ignore (1) previously learned, predictable things, (2) inherently unpredictable ones (such as details of white noise on the screen), and (3) things that are unexpected but not expected to be easily learned (such as the contents of an advanced math textbook beyond the explorer’s current level).”

# Learning Progress

Straightforward implementation of learning progress:

- Keep a buffer of experience tuples,
- update your model with the new transitions,
- evaluate the **reduction of your prediction error** in the buffer
- Assign reward based on such reduction

# Learning progress using Bayesian Neural dynamics

- Environment dynamics modeled as  $p(s_{t+1}|s_t, a_t; \theta)$
- Taking actions that maximize the reduction in uncertainty about the dynamics

$$\sum_t [H(\Theta|\xi_t, a_t) - H(\Theta|s_{t+1}, \xi_t, a_t)]$$

where  $\xi_t = \{s_1, a_1, \dots, s_t\}$  corresponds to the history up to time  $t$

- Interpretation using mutual information between  $s_{t+1}$  and  $\Theta$

$$I(s_{t+1}; \Theta|\xi_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot|\xi_t, a_t)} \left\{ \underbrace{D_{KL} [p(\theta|\xi_t, a_t, s_{t+1}) || p(\theta|\xi_t)]}_{\text{Information Gain}} \right\}$$

How much the weight distribution changes based on the **newly observed transition**

# Variational Approximation for the Posterior of the Weights

- Approximating  $p(\theta|D)$  with  $q(\theta; \phi)$
- The total reward is then approximated as

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{KL} [q(\theta|\phi_{t+1}) || q(\theta|\phi_t)]$$

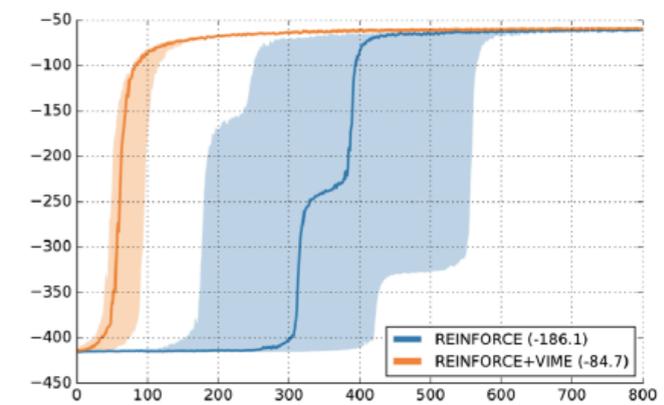
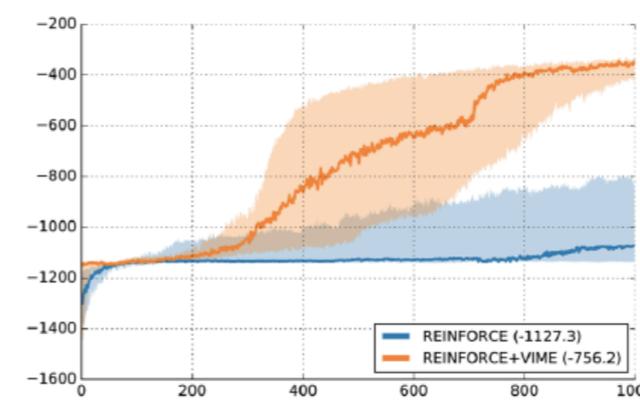
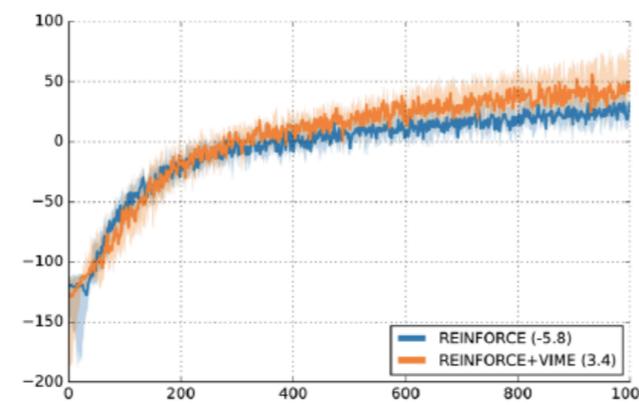
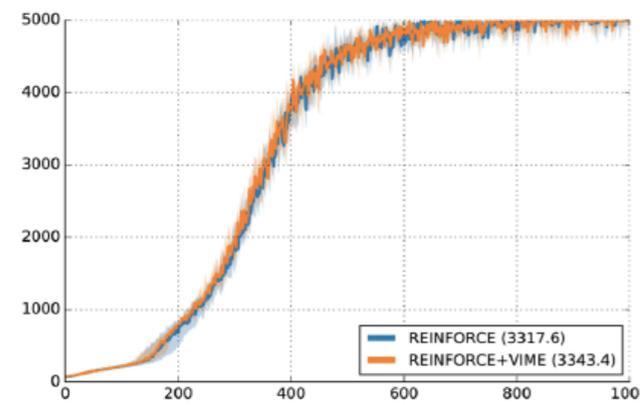
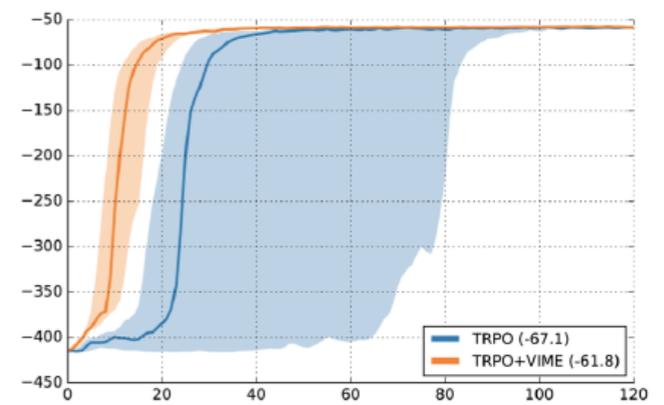
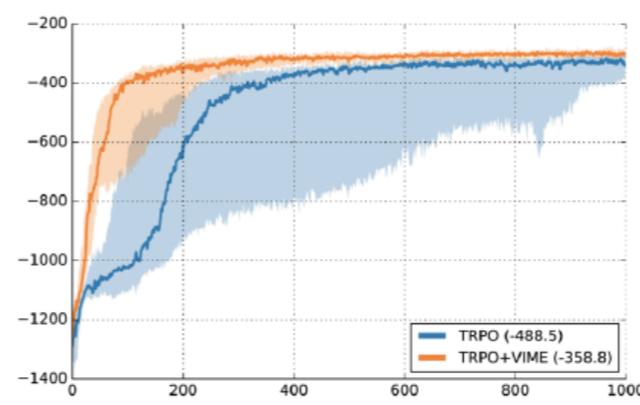
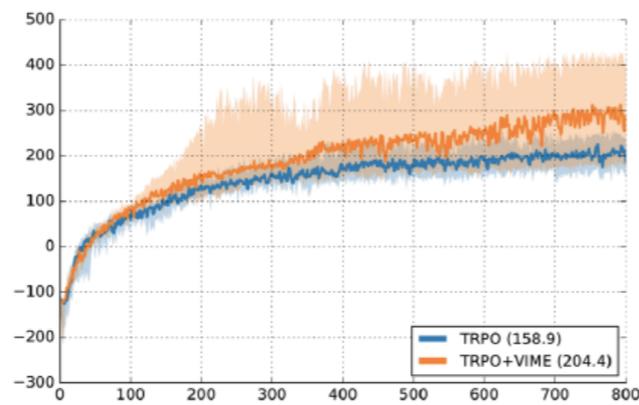
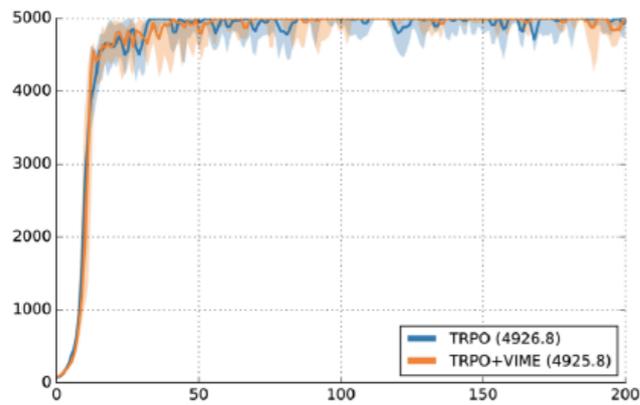
- The weight distribution parameterized by  $\phi$

$$q(\theta; \phi) = \prod_{i=1}^{|\Theta|} \mathcal{N}(\theta_i | \mu_i, \sigma_i^2)$$

- Feedforward network structure and ReLU nonlinearity between hidden layers
- Trained by maximizing the variational lower bound using Backprop

$$L[q(\theta; \phi), D] = \mathbb{E}_{\theta \sim q(\cdot; \phi)} [\log p(D|\theta)] - D_{KL}[q(\theta; \phi) || p(\theta)]$$

# Experiments



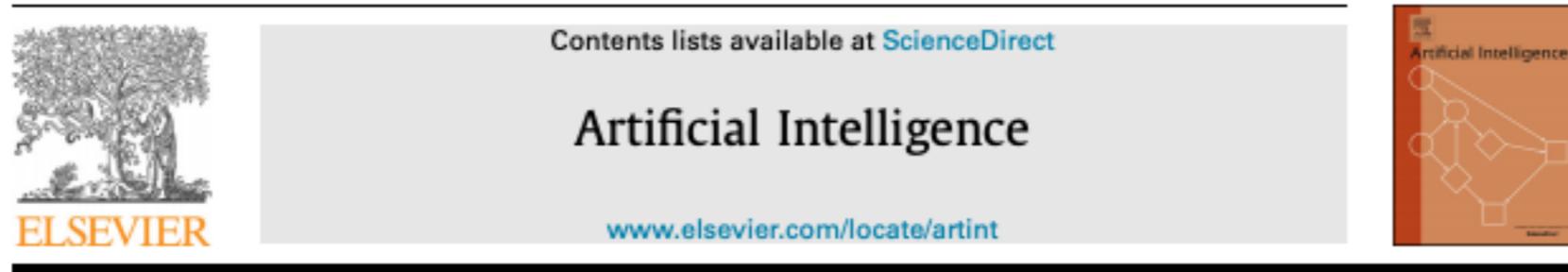
(a) CartPole

(b) CartPoleSwingup

(c) DoublePendulum

(d) MountainCar

# Curiosity only learning on real robots



Intrinsically motivated model learning for developing curious robots

Todd Hester<sup>\*,1</sup>, Peter Stone

*Department of Computer Science, The University of Texas at Austin, United States*

1. Learn a transition function (random forest)
2. Learn a reward function based on curiosity: a) state density  
2) entropy of the tree predictions.
3. Online planning (similar to MCTS) **using the learned model!**  
to pick actions that generate large rewards
4. Update the model with the collected experience

# Curiosity-guided model learning

## Algorithm 1 MODEL.

```

1: procedure INIT-MODEL( $n$ )
2:   for  $i = 1 \rightarrow n$  do
3:      $featModel_i \Rightarrow INIT()$ 
4:   end for
5:   for  $i = 1 \rightarrow nactions$  do
6:      $X_i \leftarrow \emptyset$ 
7:   end for
8: end procedure

9: procedure UPDATE-MODEL( $list$ )
10:  for all  $\langle s, a, s' \rangle \in list$  do
11:     $s^{rel} \leftarrow s' - s$ 
12:    for all  $s_i^{rel} \in s^{rel}$  do
13:       $featModel_i \Rightarrow UPDATE(\langle s, a \rangle, s_i^{rel})$ 
14:    end for
15:     $X_a \leftarrow X_a \cup s$ 
16:  end for
17: end procedure

18: procedure QUERY-MODEL( $s, a, v, n$ )
19:  for  $i = 1 \rightarrow LENGTH(s)$  do
20:     $s_i^{rel} \leftarrow featModel_i \Rightarrow QUERY(\langle s, a \rangle)$ 
21:  end for
22:   $s' \leftarrow s + \langle s_1^{rel}, \dots, s_n^{rel} \rangle$ 
23:   $D(s, a) \leftarrow \sum_{i=1}^n featModel_i \Rightarrow VARIANCE(\langle s, a \rangle)$ 
24:   $\delta(s, a) \leftarrow \min_{s_x \in X_a} \|s - s_x\|_1$ 
25:   $r_{var} \leftarrow vD(s, a)$ 
26:   $r_{nov} \leftarrow n\delta(s, a)$ 
27:   $r \leftarrow r_{var} + r_{nov}$ 
28:  return  $\langle s', r \rangle$ 
29: end procedure

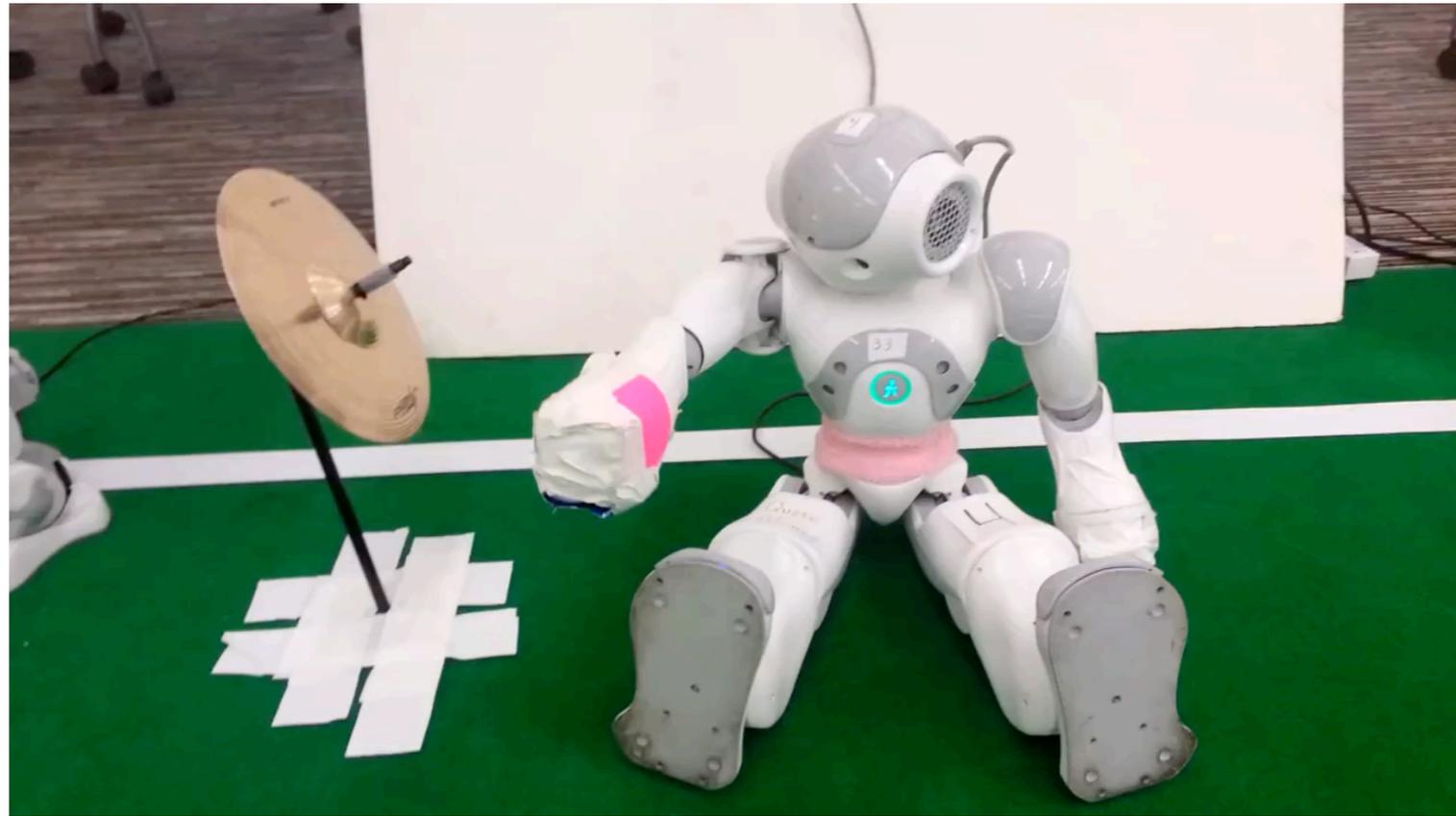
```

▷  $n$  is the number of state variables  
 ▷ Init random forest to predict feature  $i$   
 ▷ Init visited state set for action  $i$   
 ▷ Update model with  $list$  of experiences  
 ▷ Calculate relative effect  
 ▷ Train feature model  
 ▷ Add  $s$  to visited set for action  $a$   
 ▷ Get prediction of  $\langle s', r \rangle$  for  $s, a$   
 ▷ Sample prediction for feat  $i$   
 ▷ Get absolute next state  
 ▷ Calculate variance.  
 ▷ Each forest returns  $\sum_{j=1}^m \sum_{k=1}^m D_{KL}(P_j(x_i|s, a) || P_k(x_i|s, a))$   
 ▷ Calculate model novelty  
 ▷ Calculate variance reward  
 ▷ Calculate novelty reward  
 ▷ Return sampled next state and reward

This is a random forest  
 Transition novelty  
 State novelty

$$D(s, a) = \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^m D_{KL}(P_j(x_i|s, a) || P_k(x_i|s, a)),$$

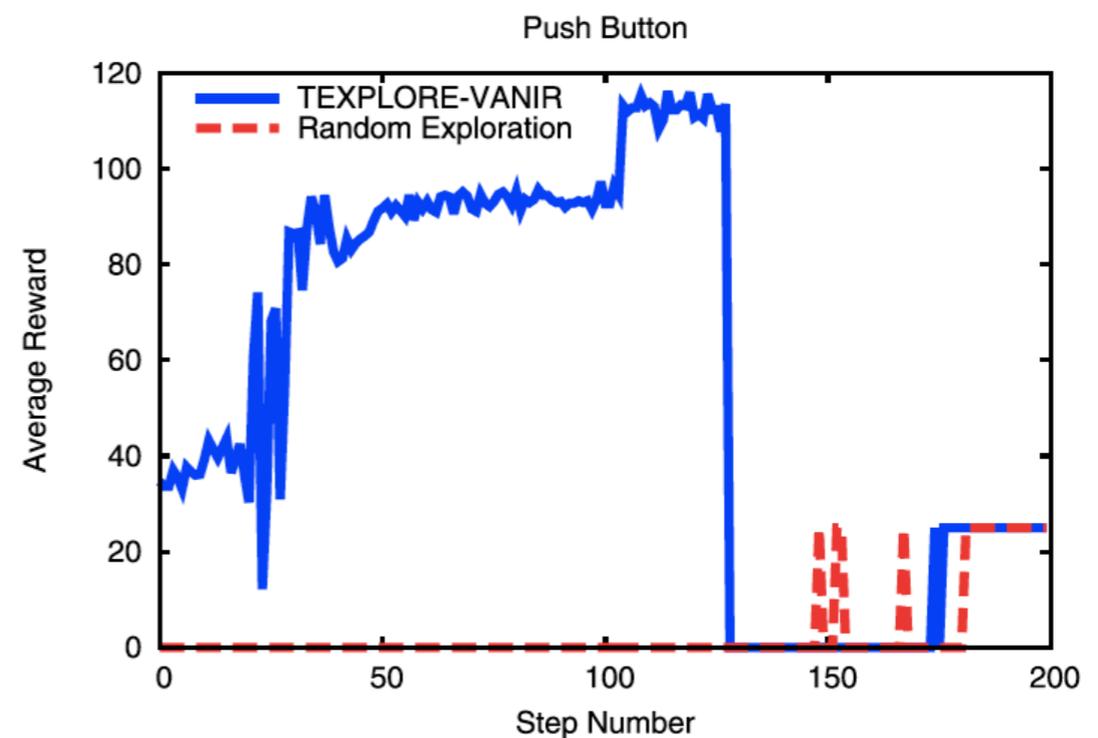
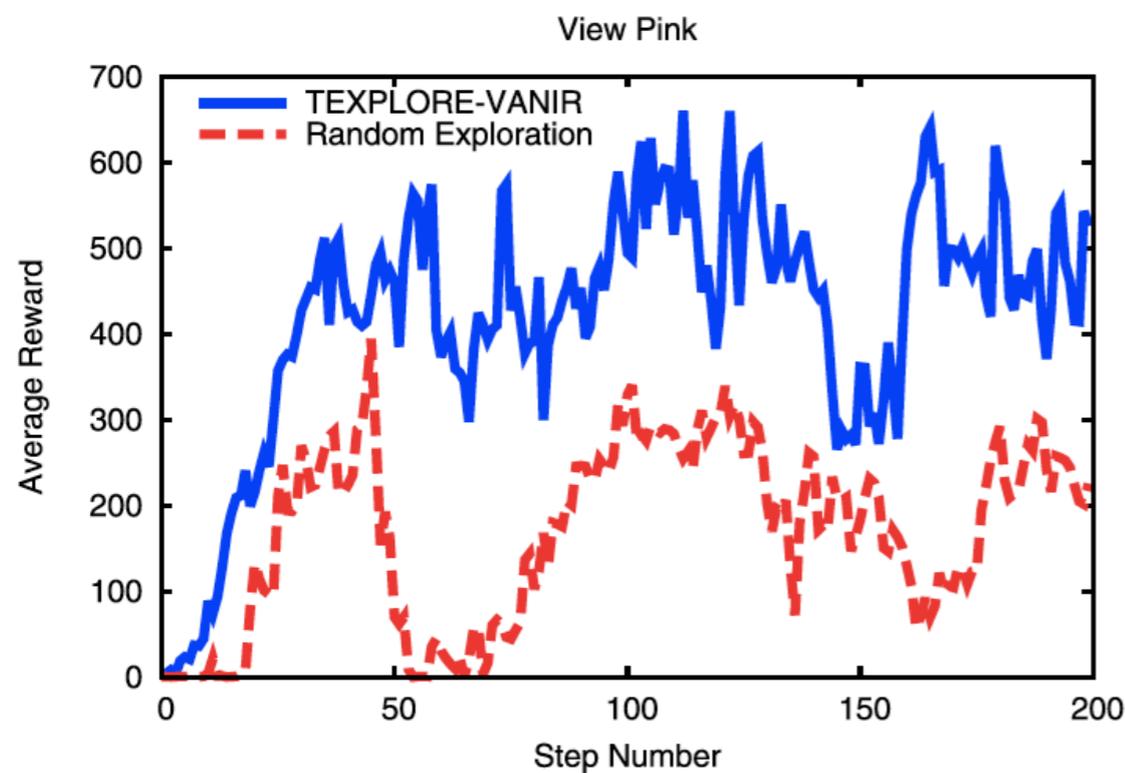
# Curiosity only learning on real robots



- **Actions:** The agent controls the robot's two right shoulder joints. It can increase the angle of either joint by 8 degrees, decrease the angle of either joint by 8 degrees, or do nothing.
- **State:** the angle of both shoulder joints, the 3-dimensional location of the robot's hand in mm relative to its chest, how many pink pixels the robot can see in its camera image, whether its right foot button is pressed, and the amount of energy it hears on its microphone

# Curiosity-guided model learning

- After the exploration stage, we have built a model and now we use it trying to maximize specific extrinsic rewards! (as opposed to curiosity rewards) using the same online planning.
- And yes, the model we built with exploration is better than what we built by trying out random actions, in that, it allows us to succeed to such new extrinsic tasks.



More on model-based RL on Wednesday