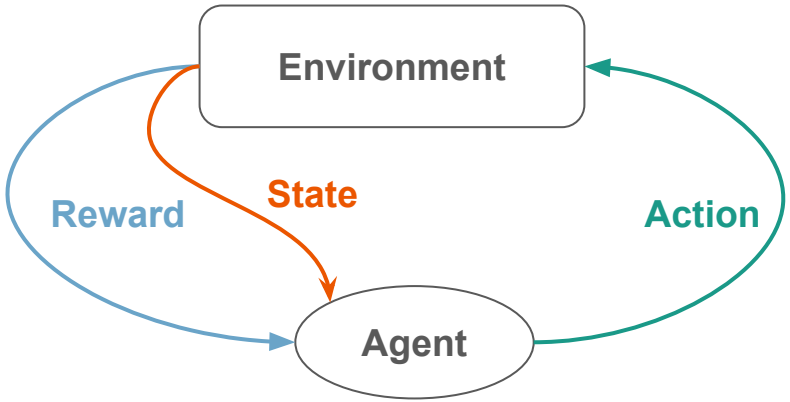
Deep Exploration via Bootstrapped DQN

Ian Osband, Charles Blundell, Alexander Pritzel, Benjamin Van Roy

Presenters : Irene Jiang, Jeremy Ma, Akshay Budhkar

Recap on Reinforcement Learning

- Multi-armed bandit problem
  - Stateless
- Contextual bandits
  - Have states, but states are independent
- Reinforcement learning
  - More complicated settings: states with specific states transitions
- MDPs(Markov Decision Processes)



Key terms: state(***s***), action(***a***), reward(***r***), policy(π), value of a state(v)

Recap on RL

- On-policy Vs Off-policy:
  - Off-policy: Independent of action taken from the agents. E.g. Q-learning
  - On-policy: Dependent of policy used. E.g. Policy gradient, SARSA
- Some major challenges in RL
  - Off-policy
  - Exploration Vs exploitation
  - Hierarchy
  - Optimization

Background on this paper

- Motivation:
  - Deep exploration is often inefficient
  - To measure uncertainty for values calculated by neural network
- Highlights:
  - DQN Vs Q-learning
  - Based on DQN(we will talk more later), bootstrap gives an efficient way to explore deeply.
  - Bootstrap method also provides a way of measuring the uncertainty for this neural network.

DQN: Deep Q network

- What is DQN?
  - A neural network to calculate the Q-value for each state-action pair
  - A off-policy technique
- Recap: Q-learning
  - A value table for each state-action pair

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \right) \quad 1$$

- DQN Loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right] \quad 2$$

[1]Mnih et al., Playing Atari with Deep Reinforcement Learning [2]https://en.wikipedia.org/wiki/Q-learning

DQN: Deep Q network(ctd)

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

→ Building dataset

→ Training the network

Mnih et al., Playing Atari with Deep Reinforcement Learning

DQN Modification

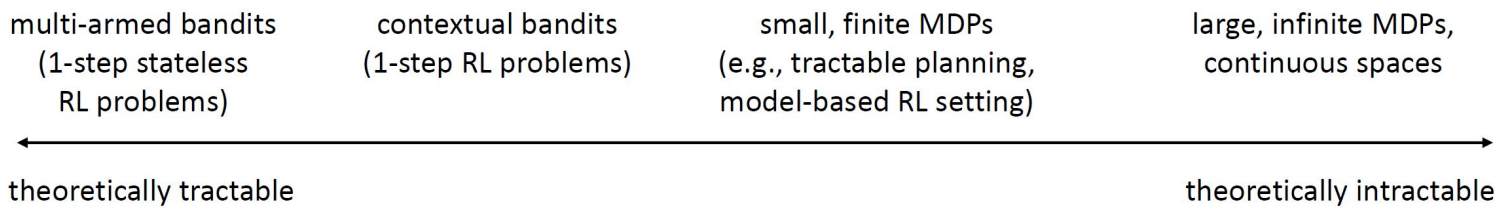
- Double DQN
  - Updated the target calculation

$$y_t = r_t + \gamma \max_a Q(\phi_{t+1}, arg\max_a Q^*(\phi_{t+1}, a_t; \theta); \theta)$$

- Bootstrap DQN
  - Approximate a distribution of Q-values
  - Natural adaption from Thompson sampling heuristic to RL (more details later)

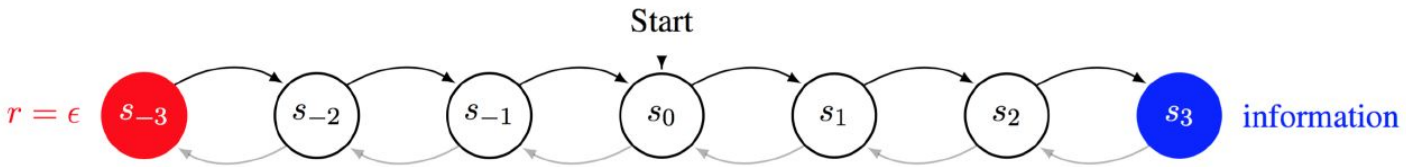
Thompson Sampling in RL

- Review: How is **Thompson sampling** used in bandits problems?
- Why is **RL exploration** different from bandit problems?
- How to apply Thompson Sampling in RL exploration?



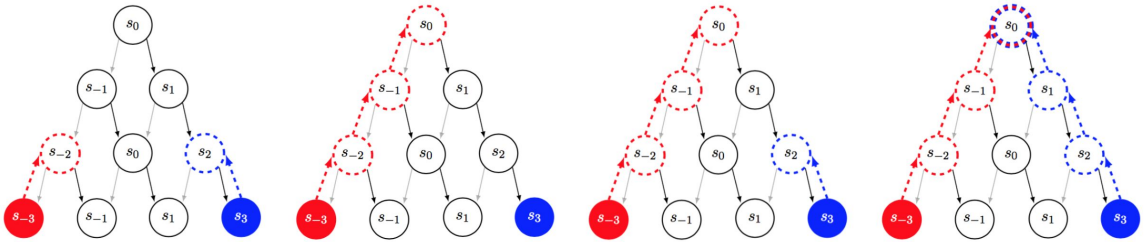
Deep Exploration - What is it?

- Difference between RL exploration and bandits: RL exploration must be **deep**
- **Deep exploration** = “planning to learn” or “farsighted exploration”
- Example: The agent has life horizon = 3. What’s the best policy?



Deep Exploration - Why is it better?

- Normal RL (DQN) agent can plan to exploit future rewards
- By contrast, RL agent with deep exploration can **plan to learn**



(a) Bandit algorithm (b) RL+dithering (c) RL+shallow explore (d) RL+deep explore

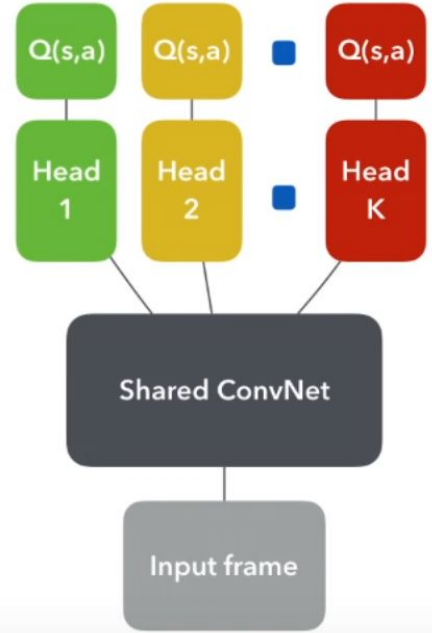
Bootstrapping - What is it?

- Bootstrapping means to **approximate the population distribution** using a **sample distribution**
- How to bootstrap?
  - Step 1: Sample population data D with replacement to get $\{D_1, D_2, \dots, D_n\}$
  - Step 2: Train n probabilistic models $\{M_1(\theta), M_2(\theta), \dots, M_n(\theta)\}$, each with training data D_i
  - Step 3: Uniform Randomly choose one model $M_k(\theta)$ from all models $\{M_1(\theta), M_2(\theta), \dots, M_n(\theta)\}$
  - Step 3: Sample from $M_k(\theta)$
- Naive implementation:
  - Train n different neural networks to realize $\{M_1(\theta), M_2(\theta), \dots, M_n(\theta)\}$
- Really expensive to train n different big neural networks.
  - What is the better solution?

Bootstrapped DQN - Implementation

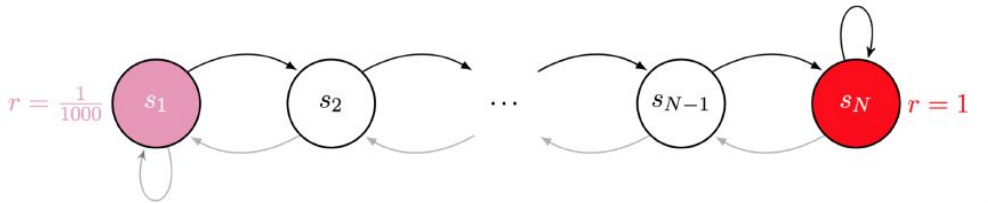
- **DQN** uses 1 Q function for value estimation
- **Bootstrapped DQN** uses **K bootstrapped heads** for value estimation

- **Training**: Each head is trained on different slice of data
- **Execution**: Bootstrapped DQN randomly selects 1 head to follow per episode



Bootstrapped DQN vs DQN

- DQN fails when there is the need for **deep exploration**
- Consider the following example:
- The agent starts at s2, has life horizon of N+9 steps. What's the best policy?



Bootstrapped DQN vs DQN - Test time

- None of the other types of DQN could explore as well as bootstrapped DQN when the chain length is really long!

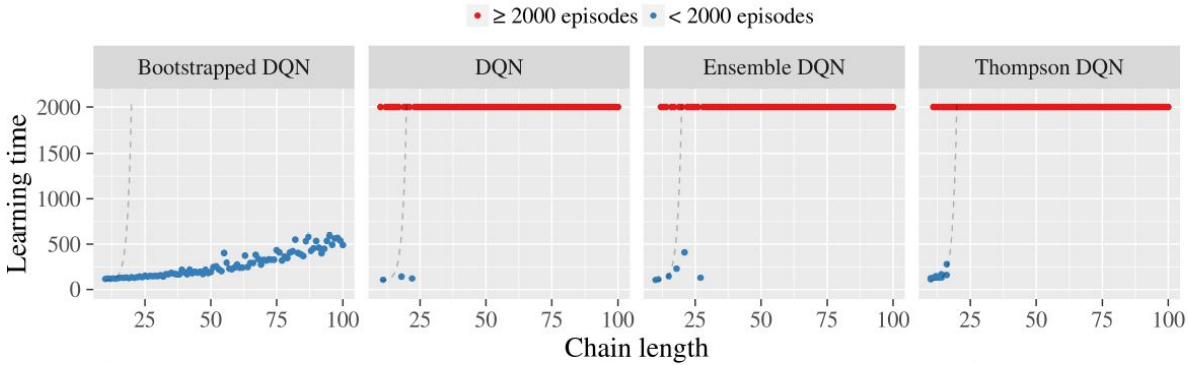


Figure 4: Only Bootstrapped DQN demonstrates deep exploration.

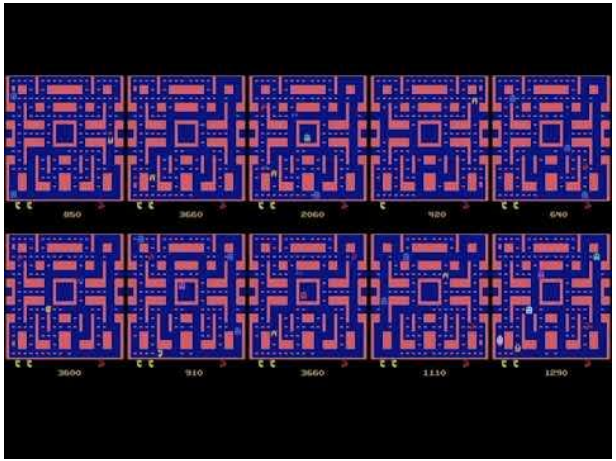
Bootstrapped DQN - Why does it work?

- Problem with epsilon-greedy:
  - **Oscillates** back and forth and not determined to go to a place
- How does bootstrapped DQN drive deep exploration?
  - It **commits** to a randomized but internally consistent strategy for an entire episode
- It is just like a team of diverse people in real life.
  - For every episode, we randomly choose a leader in the diverse group

Bootstrapped DQN - Exact Algorithm

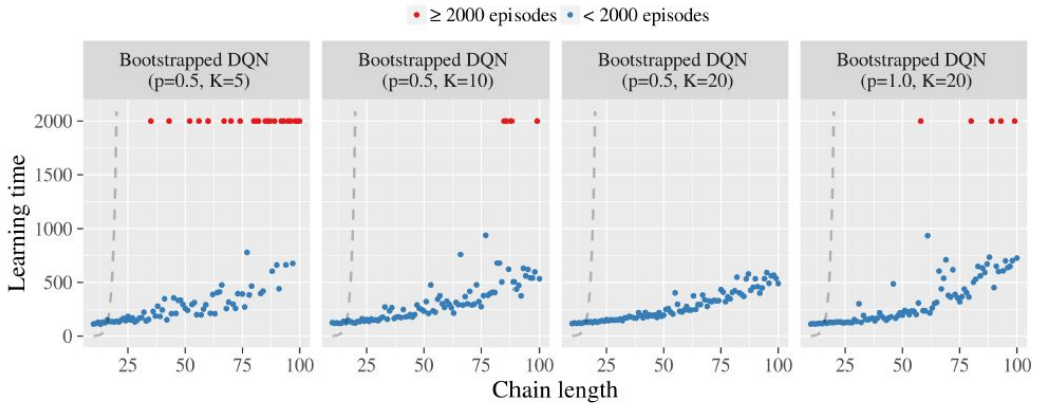
Algorithm 1 Bootstrapped DQN

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



Bootstrapped DQN - Methodology

- Hyperparameters:
  - P: How much data sharing do we want among the heads?
  - K: How many heads do we want?



Bootstrapped DQN - Test with Stochastic MDP

- Does bootstrapped DQN work well under **stochastic situations**?

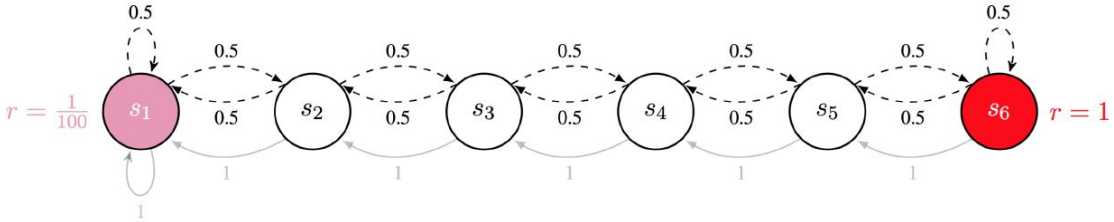
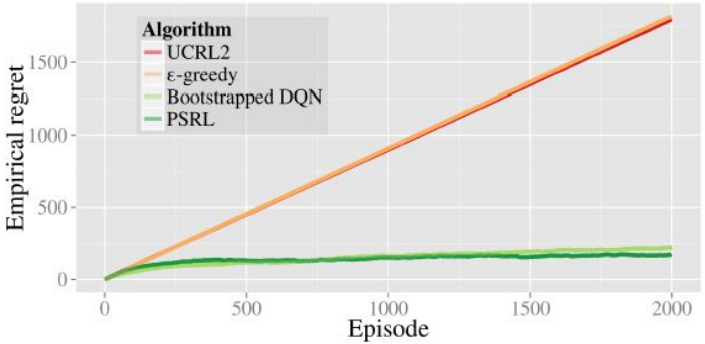
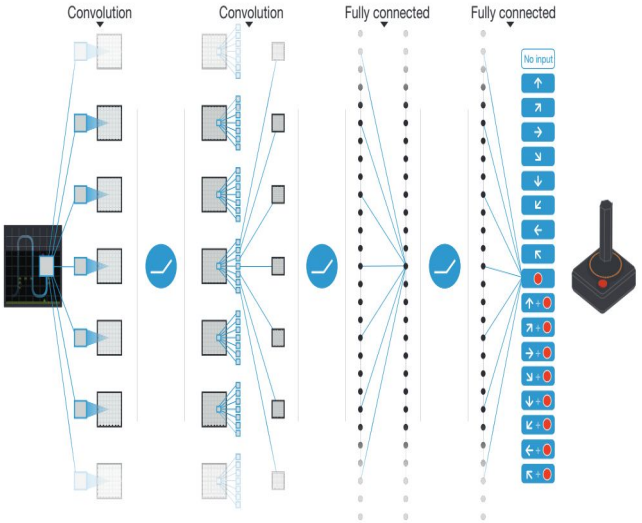


Figure 14: A stochastic MDP that requires deep exploration.



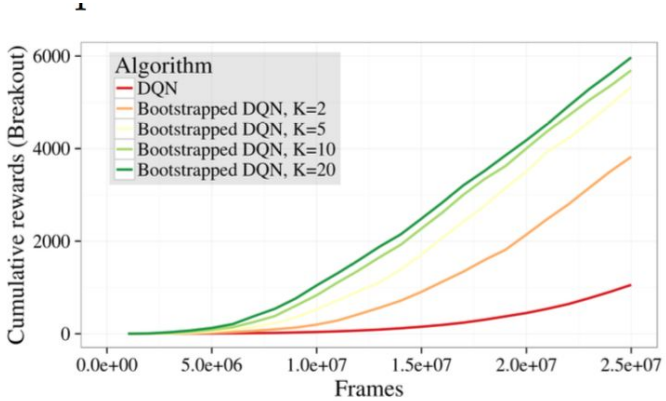
Experiments with Atari: Setup

- 49 Atari Games
- Reward Values clipped between -1 and 1
- Conv network
  - Input: 4x84x84 tensor
  - Beyond conv layer - K distinct heads with identical architectures
  - 512 units fully connected + Q-values for each action fully connected
  - ReLu activation presents nonlinearity
  - RMSProp with momentum 0.95, lr = 0.00025, discount = 0.99
- Evaluation = ensemble voting policy



Bootstrap DQN at scale

- Varying K
  - More heads = Better performance
  - Even a small value of K has better performance than DQN



Gradient Normalization in Bootstrap Heads

- Shared architecture allows training combined network through backpropagation
- Bootstrap without normalization
  - Learns faster, but prone to premature convergence
- Normalization allows bettering “best” by DQN
- Cumulative Reward vs. Best policy

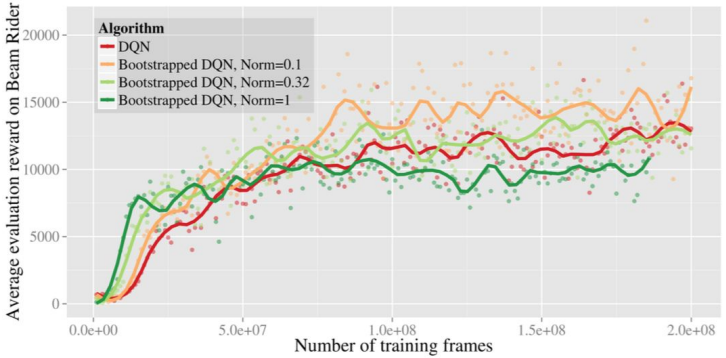
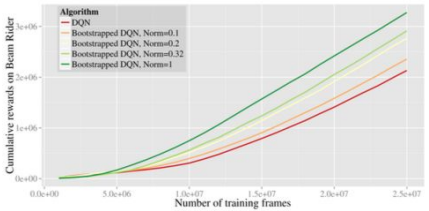


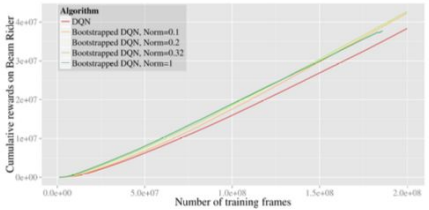
Figure 17: Normalization fights premature convergence.

Gradient Normalization in Bootstrap Heads

- Bootstrap DQN with no Normalization is better than everything else for cumulative reward
- Bootstrap DQN with normalization is still better than DQN



(a) Normalization does not help cumulative rewards.



(b) Even over 200m frames the importance of exploration dominates the effects of an inferior final policy.

Figure 18: Planning, learning and exploration in RL.

Efficient exploration in Atari

- Bootstrap DQN outperforms DQN for most Atari Games
- Montezuma's Revenge = Bootstrap DQN reaches some reward after 200m steps

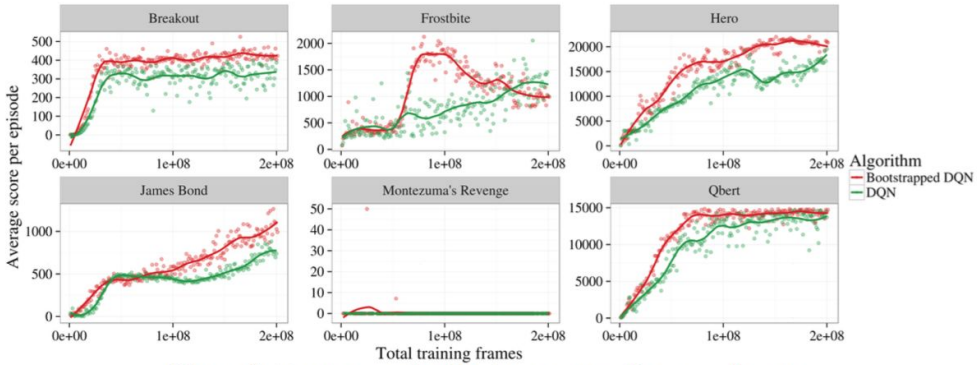


Figure 6: Bootstrapped DQN drives more efficient exploration.

Overall Performance

- Bootstrap reaches human level performance faster than DQN
- Improvement Factor = (Time taken by DQN)/(Time taken by Bootstrap DQN)

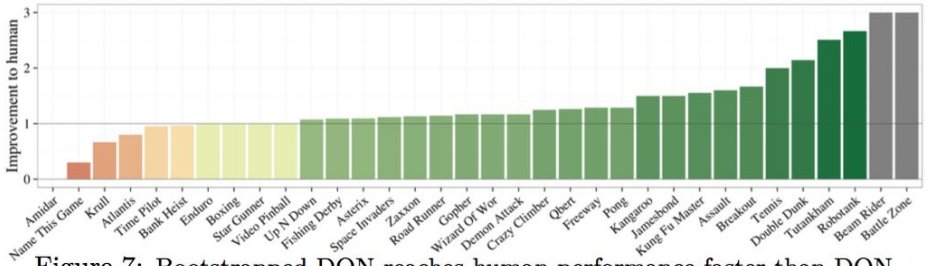


Figure 7: Bootstrapped DQN reaches human performance faster than DQN.

Overall Performance

- Bootstrap DQN's final score is *usually* more than DQN
- Bootstrap DQN cumulative rewards are orders of magnitude more than DQN

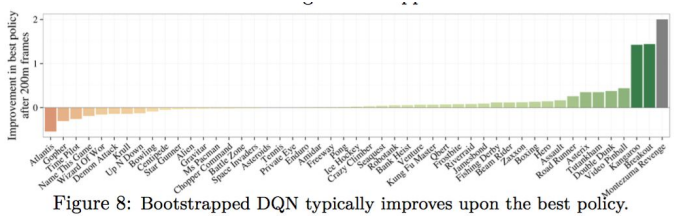


Figure 8: Bootstrapped DQN typically improves upon the best policy.

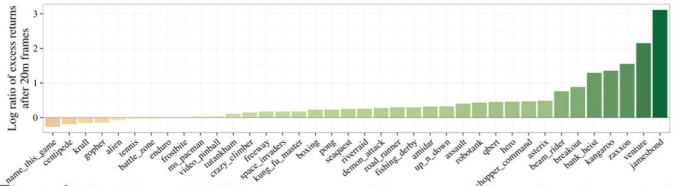
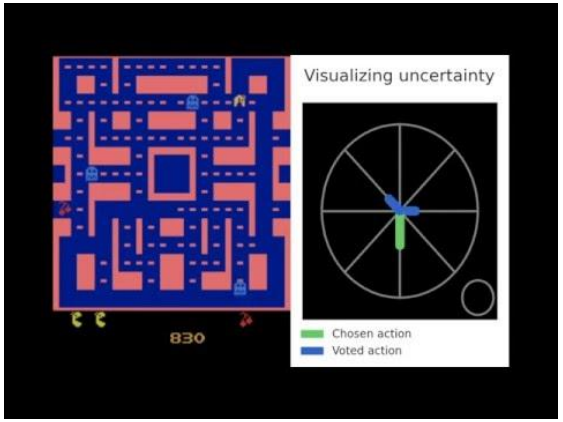


Figure 9: Bootstrapped DQN improves cumulative rewards by orders of magnitude.

Visualizing Bootstrapped DQN



Closing Remarks on Bootstrap DQN

- Efficient RL Algorithm in complex environments
- Computationally tractable and parallelizable
- Practically combines efficient generalization with exploration for nonlinear value functions

Thank you!

VIME: Variational Information Maximizing Exploration

Rein Houthooft*^#, Xi Chen*#, Yan Duan*#, John Schulman*#, Filip De Turck^, Pieter Abbeel*#

Presented by: Dan Goldberg & Kamyar Ghasemipour

* UC Berkeley, Department of Electrical Engineering and Computer Science
^ Ghent University - imec, Department of Information Technology
# OpenAI

The Reinforcement Learning Setup

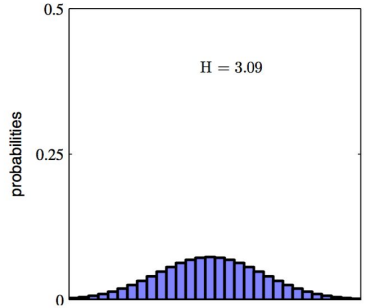
States	$\mathcal{S} \subseteq \mathbb{R}^n$	*Continuous*
Actions	$\mathcal{A} \subseteq \mathbb{R}^m$	*Continuous*
Rewards	$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$	Bounded
Transition Dynamics	$\mathcal{P} : \mathcal{S} \times \mathcal{A} \times S \to \mathbb{R}_{\geq 0}$	$s_{t+1} \sim P(s_{t+1} \vert s_t, a_t)$
Discount Rate	$\gamma \in (0, 1]$	General
Policy	$\pi_\alpha : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$	$a_t \sim \pi_\alpha(a_t \vert s_t)$

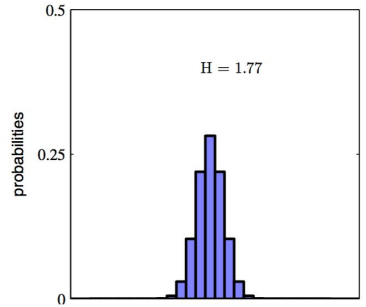
Images from OpenAI

Curiosity Driven Exploration

Model the Transition Dynamics with model parameterized by Θ (BNN):

$$p(s_{t+1}|s_t, a_t; \theta), \text{ with prior over parameters } p(\theta)$$





From: Bishop, 2006

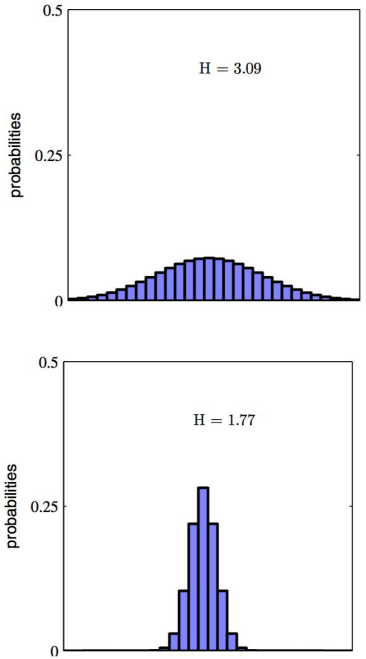
Curiosity Driven Exploration

Model the Transition Dynamics with model parameterized by Θ (BNN):

$$p(s_{t+1}|s_t, a_t; \theta), \text{ with prior over parameters } p(\theta)$$

Objective: maximize the *reduction in posterior uncertainty over the parameters:*

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



From: Bishop, 2006

Curiosity Driven Exploration

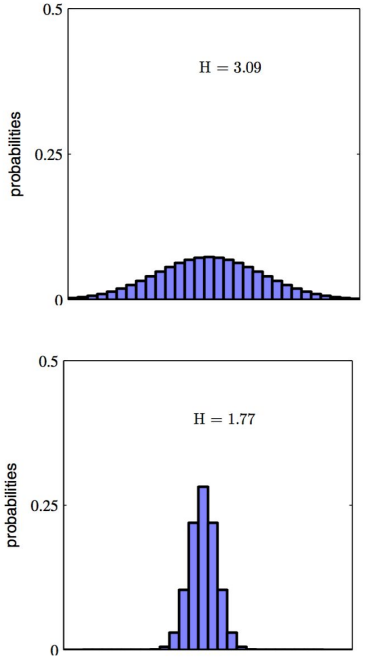
Model the Transition Dynamics with model parameterized by Θ (BNN):

$$p(s_{t+1}|s_t, a_t; \theta), \text{ with prior over parameters } p(\theta)$$

Objective: maximize the *reduction in posterior uncertainty over the parameters:*

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

The point: **encourage systematic exploration by seeking out state-action pairs that are relatively unexplored**



From: Bishop, 2006

Information Gain as Intrinsic Reward

Goal: maximize the *reduction in posterior uncertainty over the parameters*

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

Information Gain as Intrinsic Reward

Goal: maximize the *reduction in posterior uncertainty over the parameters*

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

Each term in the sum is equal to mutual information between next state random variable and parameter random variable:

$$I(S_{t+1}; \Theta|\xi_t, at) = \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[D_{\mathsf{KL}} \left[p(\theta|\xi_t, a_t, s_{t+1}) || p(\theta|\xi_t) \right] \right]$$

Information Gain as Intrinsic Reward

Goal: maximize the *reduction in posterior uncertainty over the parameters*

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

Each term in the sum is equal to mutual information between next state random variable and parameter random variable:

$$I(S_{t+1}; \Theta|\xi_t, at) = \mathbb{E}_{s_{t+1}\sim P(\cdot|s_t,a_t)} \left[D_{\mathrm{KL}} \left[p(\theta|\xi_t, a_t, s_{t+1})||p(\theta|\xi_t)\right]\right]$$

The KL Divergence term can be interpreted as ***Information Gain***.

Information Gain as Intrinsic Reward

So goal is to maximize the summed expectations of the information gain:

$$\sum_t \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[D_{\mathsf{KL}} \left[p(\theta|\xi_t, a_t, s_{t+1}) || p(\theta|\xi_t) \right] \right]$$

Information Gain as Intrinsic Reward

So goal is to maximize the summed expectations of the information gain:

$$\sum_t \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[D_{\mathsf{KL}} \left[p(\theta|\xi_t, a_t, s_{t+1}) || p(\theta|\xi_t) \right] \right]$$

To do this, add the expectation of information gain at time *t* as an intrinsic reward for the RL agent at time *t*:

Information Gain as Intrinsic Reward

So goal is to maximize the summed expectations of the information gain:

$$\sum_t \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[D_{\mathsf{KL}} \left[p(\theta|\xi_t, a_t, s_{t+1}) || p(\theta|\xi_t) \right] \right]$$

To do this, add the expectation of information gain at time *t* as an intrinsic reward for the RL agent at time *t*:

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta \mathbb{E}_{P(s_{t+1}|s_t, a_t)} \left[D_{\mathsf{KL}} \left[p(\theta|\xi_t, a_t, s_{t+1}) || p(\theta|\xi_t) \right] \right]$$

where η is a hyperparameter

Information Gain as Intrinsic Reward

Practically, sample a single action and transition (take an on-policy step) to get an estimate of the mutual information, and add that estimate as an intrinsic reward; this captures the agent's surprise at each step:

Information Gain as Intrinsic Reward

Practically, sample a single action and transition (take an on-policy step) to get an estimate of the mutual information, and add that estimate as an intrinsic reward; this captures the agent's surprise at each step:

$$a_t \sim \pi_\alpha(a_t|s_t)$$

Information Gain as Intrinsic Reward

Practically, sample a single action and transition (take an on-policy step) to get an estimate of the mutual information, and add that estimate as an intrinsic reward; this captures the agent's surprise at each step:

$$a_t \sim \pi_\alpha(a_t|s_t) \qquad s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$

Information Gain as Intrinsic Reward

Practically, sample a single action and transition (take an on-policy step) to get an estimate of the mutual information, and add that estimate as an intrinsic reward; this captures the agent's surprise at each step:

$$a_t \sim \pi_\alpha(a_t|s_t) \qquad s_{t+1} \sim P(s_{t+1}|s_t, a_t) \qquad D_{\mathrm{KL}}\left[p(\theta|\xi_t, a_t, s_{t+1})||p(\theta|\xi_t)\right]$$

Information Gain as Intrinsic Reward

Practically, sample a single action and transition (take an on-policy step) to get an estimate of the mutual information, and add that estimate as an intrinsic reward; this captures the agent's surprise at each step:

$$a_t \sim \pi_\alpha(a_t|s_t) \qquad s_{t+1} \sim P(s_{t+1}|s_t, a_t) \qquad D_{\mathsf{KL}}\left[p(\theta|\xi_t, a_t, s_{t+1})||p(\theta|\xi_t)\right]$$

So the actual reward function looks like this:

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{\mathsf{KL}}\left[p(\theta|\xi_t, a_t, s_{t+1})||p(\theta|\xi_t)\right]$$

Mackay: Information Gain

Active learning for Bayesian interpolation

- Total Information Gain objective function; the same as what we are considering here (maximizing the KL divergence of posterior from prior).

$$\Delta S = S_N - S_{N+1} \qquad G' = -\int d^k\mathbf{w}\, P(\mathbf{w} \mid \mathbf{t}) \log \frac{P(\mathbf{w})}{P(\mathbf{w} \mid \mathbf{t})}$$

MacKay, 1992. Information-based objective functions for active data selection.

Mackay: Information Gain

Active learning for Bayesian interpolation

- Total Information Gain objective function; the same as what we are considering here (maximizing the KL divergence of posterior from prior).

$$\Delta S = S_N - S_{N+1} \qquad G' = -\int d^k \mathbf{w}\, P(\mathbf{w} \mid \mathbf{t}) \log \frac{P(\mathbf{w})}{P(\mathbf{w} \mid \mathbf{t})}$$

$$S = \frac{k}{2}(1 + \log 2\pi) + \frac{1}{2}\log\left(m^2 \det \mathbf{A}^{-1}\right)$$

MacKay, 1992. Information-based objective functions for active data selection.

Mackay: Information Gain

Active learning for Bayesian interpolation

- Total Information Gain objective function; the same as what we are considering here (maximizing the KL divergence of posterior from prior).

$$\Delta S = S_N - S_{N+1} \qquad G' = -\int d^k\mathbf{w}\, P(\mathbf{w} \mid \mathbf{t}) \log \frac{P(\mathbf{w})}{P(\mathbf{w} \mid \mathbf{t})}$$

$$S = \frac{k}{2}(1 + \log 2\pi) + \frac{1}{2}\log\left(m^2 \det \mathbf{A}^{-1}\right) \qquad \text{Total information gain} = \frac{1}{2}\Delta \log\left(m^2 \det \mathbf{A}\right)$$

$$= \frac{1}{2}\log(1 + \beta \mathbf{g}^{\mathrm{T}} \mathbf{A}^{-1} \mathbf{g})$$

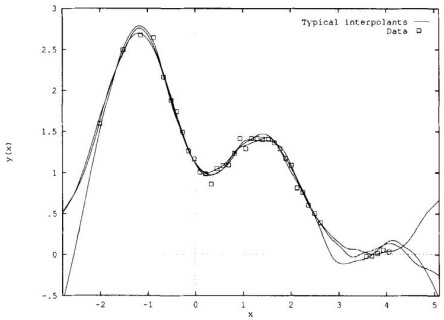
MacKay, 1992. Information-based objective functions for active data selection.

Mackay: Information Gain

Active learning for Bayesian interpolation

- Total Information Gain objective function; the same as what we are considering here (maximizing the KL divergence of posterior from prior).
- He proved a common result of this method to pick the furthest points from the data (points with maximum variance).

MacKay, 1992:
Bayesian Interpolation



$$\frac{1}{2}\log(1+\beta\mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g})$$

MacKay, 1992. Information-based objective functions for active data selection.

Mackay: Information Gain

Active learning for Bayesian interpolation

- Total Information Gain objective function; the same as what we are considering here (maximizing the KL divergence of posterior from prior).
- He proved a common result of this method to pick the furthest points from the data.
- In this case, that is desired - to encourage systematic exploration.

$$\frac{1}{2}\log(1+\beta\mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g})$$

MacKay, 1992. Information-based objective functions for active data selection.

Mackay: Information Gain

Active learning for Bayesian interpolation

- Total Information Gain objective function; the same as what we are considering here (maximizing the KL divergence of posterior from prior).
- He proved a common result of this method to pick the furthest points from the data.
- In this case, that is desired - to encourage systematic exploration.
- The main difference is this for VIME exploration is only part of the reward function - there is still exploitation!

$$\frac{1}{2}\log(1+\beta\mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g})$$

MacKay, 1992. Information-based objective functions for active data selection.

Variational Bayes

Problem: the posterior over parameters is usually intractable.

$$p(\theta|\xi_t, a_t, s_{t+1}) = \frac{p(\theta|\xi_t)p(s_{t+1}|\xi_t, a_t; \theta)}{p(s_{t+1}|\xi_t, a_t)}$$

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{\mathrm{KL}}\left[p(\theta|\xi_t, a_t, s_{t+1})||p(\theta|\xi_t)\right]$$

Variational Bayes

Problem: the posterior over parameters is usually intractable.

$$p(\theta|\xi_t, a_t, s_{t+1}) = \frac{p(\theta|\xi_t)p(s_{t+1}|\xi_t, a_t; \theta)}{p(s_{t+1}|\xi_t, a_t)}$$

Solution: use Variational Bayes to approximate the posterior!

$$q_\phi(\theta) \equiv q(\theta; \phi_t) \approx p(\theta|\xi_t, a_t, s_{t+1})$$

Variational Bayes

The best approximation minimizes KL Divergence

$$D_{\mathrm{KL}}\left[q(\theta;\phi_t)||p(\theta|\xi_t,a_t,s_{t+1})\right]$$

Variational Bayes

The best approximation minimizes KL Divergence

$$D_{\mathrm{KL}}\left[q(\theta;\phi_t)||p(\theta|\xi_t,a_t,s_{t+1})\right]$$

Maximize the **Variational Lower Bound** to Minimize KL Divergence

$$\log p(\xi_t,a_t,s_{t+1}) = D_{\mathrm{KL}}\left[q(\theta;\phi_t)||p(\theta|\xi_t,a_t,s_{t+1})\right] + \int_{\Theta} q(\theta;\phi)\log\frac{p(\theta,\xi_t,a_t,s_{t+1})}{q(\theta;\phi)}$$

Variational Bayes

Variational Lower Bound = (negative) Description Length:

$$L\left[q(\theta;\phi);\xi_t,a_t,s_{t+1}\right]=\mathbb{E}_{\theta\sim q(\cdot;\phi)}\left[\log p(s_{t+1}|\xi_t,a_t,\theta)\right]-D_{\mathsf{KL}}\left[q(\theta;\phi)||p(\theta)\right]$$

Variational Bayes

Variational Lower Bound = (negative) Description Length:

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

*Data terms abstracted away

Variational Bayes

Variational Lower Bound = (negative) Description Length:

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

(-) **data** description length (-) **model** description length

*Data terms abstracted away

Graves: Variational Complexity Gain

Curriculum Learning for BNNs

- EXP3.P algorithm for piecewise stationary adversarial bandits

Graves et al., 2017. Automated curriculum learning for neural networks.

Graves: Variational Complexity Gain

Curriculum Learning for BNNs

- EXP3.P algorithm for piecewise stationary adversarial bandits
- Policy is function of importance sampled rewards based on *learning progress v*

Graves et al., 2017. Automated curriculum learning for neural networks.

Graves: Variational Complexity Gain

Curriculum Learning for BNNs

- EXP3.P algorithm for piecewise stationary adversarial bandits
- Policy is function of importance sampled rewards based on *learning progress v*
- Focus on *model* complexity (a.k.a. *model* description length) as opposed to *total* description length

Graves et al., 2017. Automated curriculum learning for neural networks.

Graves: Variational Complexity Gain

Curriculum Learning for BNNs

- EXP3.P algorithm for piecewise stationary adversarial bandits
- Policy is function of importance sampled rewards based on *learning progress v*
- Focus on *model* complexity (a.k.a. *model* description length) as opposed to *total* description length

$$\nu_{VCG} = D_{\mathsf{KL}}\left[q(\theta;\phi_{t+1})||p(\theta)\right] - D_{\mathsf{KL}}\left[q(\theta;\phi_t)||p(\theta)\right]$$

Graves et al., 2017. Automated curriculum learning for neural networks.

Graves: Variational Complexity Gain

Curriculum Learning for BNNs

- EXP3.P algorithm for piecewise stationary adversarial bandits
- Policy is function of importance sampled rewards based on *learning progress v*
- Focus on *model* complexity (a.k.a. *model* description length) as opposed to *total* description length

$$\nu_{VCG} = D_{\mathrm{KL}}\left[(\text{posterior})_{t+1} || (\text{prior})\right] - D_{\mathrm{KL}}\left[(\text{posterior})_{t} || (\text{prior})\right]$$

Graves et al., 2017. Automated curriculum learning for neural networks.

VIME

Final Reward Function

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

Implementation

Recap

- $I(S_{t+1}; \Theta | \xi_t, at) = \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[D_{\mathrm{KL}} \left[p(\theta | \xi_t, a_t, s_{t+1}) || p(\theta | \xi_t) \right] \right]$
- $r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{\mathrm{KL}}[p(\theta | \xi_t, a_t, s_{t+1}) \,||\, p(\theta | \xi_t)]$
- $L[q(\theta; \phi), \mathcal{D}] = \mathbb{E}_{\theta \sim q(\cdot;\phi)} [\log p(\mathcal{D} | \theta)] - D_{\mathrm{KL}}[q(\theta; \phi) \,||\, p(\theta)]$
- $r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{\mathrm{KL}}[q(\theta; \phi_{t+1}) \,||\, q(\theta; \phi_t)]$

Implementation: Model

BNN weight distribution is chosen to be a fully factorized Gaussian distribution

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

$$\phi = \{\mu, \sigma\}$$

$$\sigma = \log(1 + e^{\rho})$$

Optimizing Variational Lower Bound

We have two optimization problems that we care:

1. One to compute: $r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{\mathrm{KL}}[q(\theta; \phi_{t+1}) \,\|\, q(\theta; \phi_t)]$
2. Other to fit the approximate posterior:

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

Review: Reparametrization Trick

Let z be a continuous random variable $\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$

In some cases it is then possible to write: $\mathbf{z} = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x})$

Where:

- g is a deterministic mapping parametrized by $\boldsymbol{\phi}$
- $\boldsymbol{\epsilon}$ is a random variable sampled from a simple tractable ditstribution

Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114* (2013).

Review: Reparametrization Trick

In the case of fully factorized Gaussian BNN:

- ϵ is sampled from a multivariate Gaussian with 0 mean and identity covariance
- $\theta = \mu(\phi, x) + \epsilon \odot \sigma(\phi, x)$

Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114* (2013).

Review: Local Reparametrization Trick

- Instead of sampling the weights, sample from the distribution over activations
- More computationally efficient, reduces gradient variance
- In the case of fully factorized Gaussian, this is super simple

$$q_\phi(w_{i,j}) = N(\mu_{i,j}, \sigma^2_{i,j})\ \forall w_{i,j} \in \mathbf{W} \implies q_\phi(b_{m,j}|\mathbf{A}) = N(\gamma_{m,j}, \delta_{m,j}),\ \text{with}$$

$$\gamma_{m,j} = \sum_{i=1}^{1000} a_{m,i}\mu_{i,j}, \quad \text{and} \quad \delta_{m,j} = \sum_{i=1}^{1000} a^2_{m,i}\sigma^2_{i,j}.$$

Kingma, Diederik P., Tim Salimans, and Max Welling. "Variational dropout and the local reparameterization trick." *Advances in Neural Information Processing Systems*. 2015.

Optimizing Variational Lower Bound #2

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

- q is updated periodically during training
- $\mathcal{D}$ is sampled from a FIFO history buffer containing tuples of the form (s_t, a_t, s_{t+1}) encountered during training
    - Reduces correlation between trajectory samples, making it closer to i.i.d.

Optimizing Variational Lower Bound #1

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

$$\phi' = \arg\min_{\phi} \Big[\underbrace{D_{\mathrm{KL}}[q(\theta; \phi) \,\|\, q(\theta; \phi_{t-1})]}_{\ell_{\mathrm{KL}}(q(\theta;\phi))} \overbrace{- \mathbb{E}_{\theta \sim q(\cdot;\phi)} [\log p(s_t | \xi_t, a_t; \theta)]}^{\ell(q(\theta;\phi), s_t)} \Big]$$

Optimizing Variational Lower Bound

$$\phi' = \arg\min_{\phi} \Big[\overbrace{\underbrace{D_{\mathrm{KL}}[q(\theta;\phi) \,\|\, q(\theta;\phi_{t-1})]}_{\ell_{\mathrm{KL}}(q(\theta;\phi))} - \mathbb{E}_{\theta\sim q(\cdot;\phi)} \left[\log p(s_t|\xi_t, a_t; \theta)\right]}^{\ell(q(\theta;\phi), s_t)} \Big]$$

- Optimized using reparametrization trick & local reparametrization trick:
  - Sample values for parameters from q: $\theta \sim q(\cdot;\phi)$
  - Compute likelihood: $\frac{1}{N}\sum_{i=1}^{N} \log p(\mathcal{D}|\theta_i)$

Optimizing Variational Lower Bound

$$\phi' = \arg\min_{\phi} \Big[\overbrace{\underbrace{D_{\mathrm{KL}}[q(\theta;\phi) \,\|\, q(\theta;\phi_{t-1})]}_{\ell_{\mathrm{KL}}(q(\theta;\phi))} - \mathbb{E}_{\theta\sim q(\cdot;\phi)} \left[\log p(s_t|\xi_t, a_t;\theta)\right]}^{\ell(q(\theta;\phi),s_t)} \Big]$$

Since we assumed the form to be a fully factorized Gaussian:

$$D_{\mathrm{KL}}[q(\theta;\phi) \,\|\, q(\theta;\phi')] = \frac{1}{2} \sum_{i=1}^{|\boldsymbol{\Theta}|} \left(\left(\frac{\sigma_i}{\sigma_i'}\right)^2 + 2\log\sigma_i' - 2\log\sigma_i + \frac{(\mu_i' - \mu_i)^2}{\sigma_i'^2} \right) - \frac{|\boldsymbol{\Theta}|}{2}$$

Hence we can compute the gradient and Hessian in closed form:

$$\frac{\partial^2 \ell_{\mathrm{KL}}}{\partial \mu_i^2} = \frac{1}{\log^2(1+e^{\rho_i})} \quad \text{and} \quad \frac{\partial^2 \ell_{\mathrm{KL}}}{\partial \rho_i^2} = \frac{2e^{2\rho_i}}{(1+e^{\rho_i})^2} \frac{1}{\log^2(1+e^{\rho_i})}$$

Optimizing Variational Lower Bound

$$\phi' = \arg\min_{\phi} \Big[\overbrace{\underbrace{D_{\mathrm{KL}}[q(\theta;\phi) \,\|\, q(\theta;\phi_{t-1})]}_{\ell_{\mathrm{KL}}(q(\theta;\phi))} - \mathbb{E}_{\theta\sim q(\cdot;\phi)} \left[\log p(s_t|\xi_t, a_t;\theta)\right]}^{\ell(q(\theta;\phi),s_t)} \Big]$$

In each optimization iteration, take a single second-order step:

$$\Delta\phi = H^{-1}(\ell)\nabla_{\phi}\ell(q(\theta;\phi), s_t)$$

“Because this KL divergence is approximately quadratic in its parameters and the log-likelihood term can be seen as locally linear compared to this highly curved KL term, we approximate H by only calculating it for the term KL”

Optimizing Variational Lower Bound

The value of the KL term after the optimization step can be approximated using a Taylor expansion:

$$\Delta\phi = H^{-1}(\ell)\nabla_\phi \ell(q(\theta;\phi), s_t)$$

$$\tfrac{1}{2}\Delta\phi H \Delta\phi = \tfrac{1}{2}\left(H^{-1}\nabla\right)^\top H \left(H^{-1}\nabla\right)$$

At the origin, the gradient and value of the KL term are zero, hence:

$$D_{\mathrm{KL}}[q(\theta;\phi+\lambda\Delta\phi)\,\|\,q(\theta;\phi)] \approx \tfrac{1}{2}\lambda^2 \nabla_\phi \ell^\top H^{-1}(\ell_{\mathrm{KL}})\nabla_\phi \ell$$

Intrinsic Reward

Last detail:

- Instead of using $D_{\mathrm{KL}}[q(\theta;\phi_{t+1}) \,||\, q(\theta;\phi_t)]$ as the intrinsic reward, it is divided by the median of the intrinsic reward over the previous k timesteps
- Emphasizes relative difference between KL divergence between samples

Experiments

Sparse Reward Domains

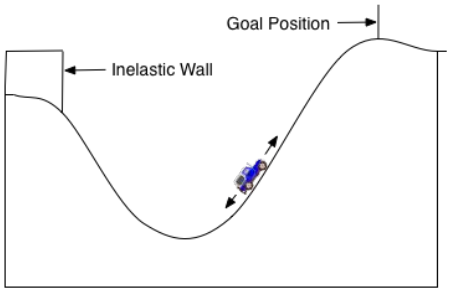
The main domains in which VIME should shine are those in which:

- Rewards are sparse and difficult to get the first rewards
- Naive exploration does not result in any feedback to improve policy

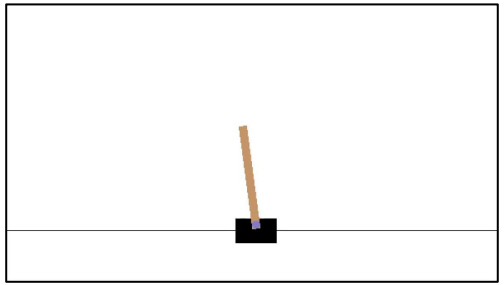
Testing on these domains allows to examine whether VIME is capable of systematic exploration

Sparse Reward Domains: Examples

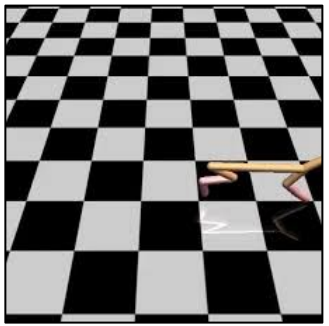
Mountain Car



Cartpole



HalfCheetah



1)http://library.rl-community.org/wiki/Mountain_Car_(Java)
2) https://www.youtube.com/watch?v=46wjA6dqxOM
3) https://gym.openai.com/evaluations/eval_qtOtPrCgS8O9U2sZG7ByQ/

Baselines

- Gaussian control noise
  - Policy model outputs the mean and covariance of a Gaussian
  - Actual action is sampled from this Gaussian
- L^2 BNN prediction error as intrinsic reward
  - A model of the environment aims to predict the next state given the current state and action to be taken (parametrized for example as a neural network)
  - Use the prediction error as an intrinsic reward
  - $e(s_t, a_t) = \|\sigma(s_{t+1}) - \mathcal{M}_\phi(\sigma(s_t), a_t)\|_2^2.$
  - $\overline{e_T} := \frac{e_T}{\max_{t \leq T}\{e_t\}} \qquad \mathcal{R}_{Bonus}(s,a) = \mathcal{R}(s,a) + \beta \left(\frac{\bar{e}_t(s_t, a_t)}{t * C} \right)$

1) Duan, Yan, et al. "Benchmarking deep reinforcement learning for continuous control." *International Conference on Machine Learning*. 2016.

2) Stadie, Bradly C., Sergey Levine, and Pieter Abbeel. "Incentivizing exploration in reinforcement learning with deep predictive models." *arXiv preprint arXiv:1507.00814* (2015).

Results

TRPO used as the RL algorithm is all experiments:

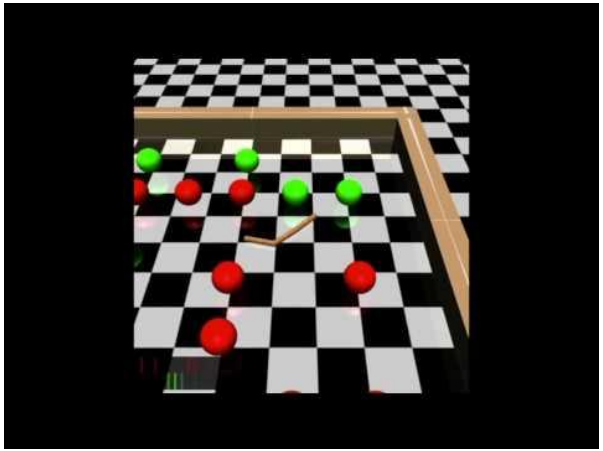
- Naive (Gaussian noise) exploration almost never reaches the goal
- L^2 does not perform well either
- VIME is very significantly more successful

Curiosity drives exploration even in the absense of any initial reward

More Difficult Task: SwimmerGather

SwimmerGather task:

- Very difficult hierarchical task
- Need to learn complex locomotion patterns before any progress can be made
- In a benchmark paper, none of the naive exploration strategies made any progress on this task



https://www.youtube.com/watch?v=w78kFy4x8ck

More Difficult Task: SwimmerGather

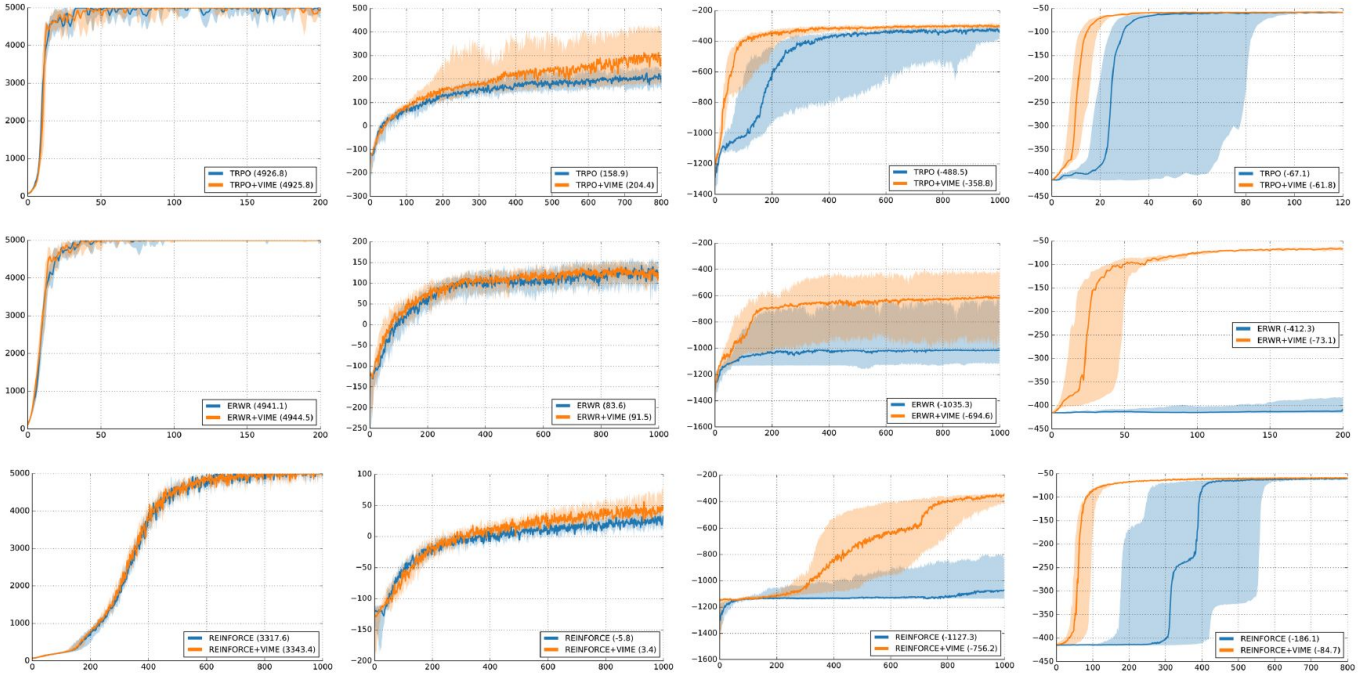
- Yet, VIME leads the agent to acquire complex motion primitives without any reward from the environment

Comparing VIME with different RL methods:

TRPO

ERWR

REINFORCE



(a) CartPole (b) CartPoleSwingup (c) DoublePendulum (d) MountainCar

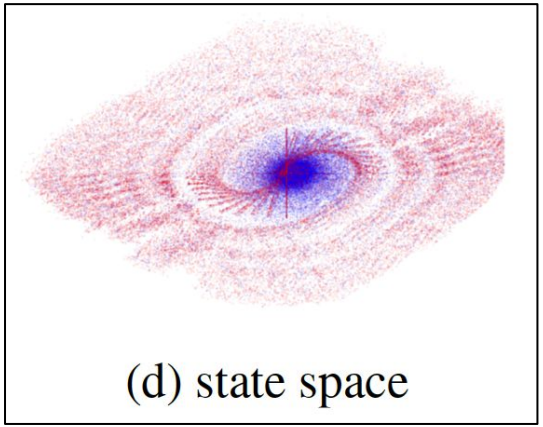
- REINFORCE & ERWR suffer from premature convergence to suboptimal policies

VIME's Exploration Behaviour:

Plot of state visitations for MountainCar Task:

- Blue: Gaussian control noise
- Red: VIME

VIME has a more diffused visitation pattern that explores more efficiently and reaches goals more quickly



(d) state space

Exploration Exploitation Trade-off

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

- By adjusting the value of n, we can tune how much emphasis we are putting on exploration:
  - Too high: will only explore and not care about the rewards
  - Too low: algorithm reduces to Gaussian control noise which does no perform well on many difficult tasks

Conclusion

- VIME represents exploration as information gain about the parameters of a dynamics model: $r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta \mathbb{E}_{P(s_{t+1}|s_t,a_t)} \left[D_{\mathrm{KL}} \left[p(\theta|\xi_t, a_t, s_{t+1}) || p(\theta|\xi_t \right.\right.$
- We do this with a good (easy) choice of approximating distribution q: $r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{\mathrm{KL}}[q(\theta; \phi_{t+1}) \,\|\, q(\theta; \phi_t)]$
- VIME is able to improve exploration for different RL algorithms, converge to better optima, and can solve very difficult tasks such as SwimmerGather

Optimizing Variational Lower Bound

Modification of variational lower bound:

- We can assume at timestep t, the approximate posterior q at step t-1 is a good prior since q is not updated very regularly (as we will see).

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

$$\phi' = \arg\min_{\phi}\Big[\overbrace{\underbrace{D_{\mathrm{KL}}[q(\theta;\phi)\,\|\,q(\theta;\phi_{t-1})]}_{\ell_{\mathrm{KL}}(q(\theta;\phi))} - \mathbb{E}_{\theta\sim q(\cdot;\phi)}\left[\log p(s_t|\xi_t,a_t;\theta)\right]}^{\ell(q(\theta;\phi),s_t)}\Big]$$

PILCO: A Model-Based and Data-Efficient Approach to Policy Search

Authors:
Marc Peter Deisenroth, Carl Edward Rasmussen

Presenters:
Shun Liao, Enes Parildi, Kavitha Mekala

Background + Dynamic Model

Presenter: Shun LIAO

Model Based RL

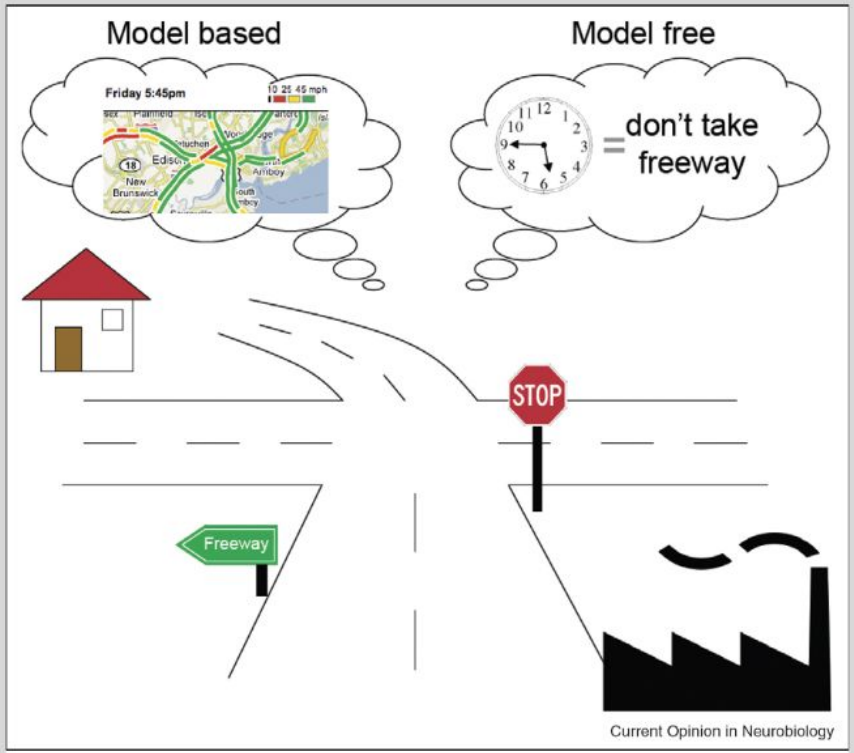
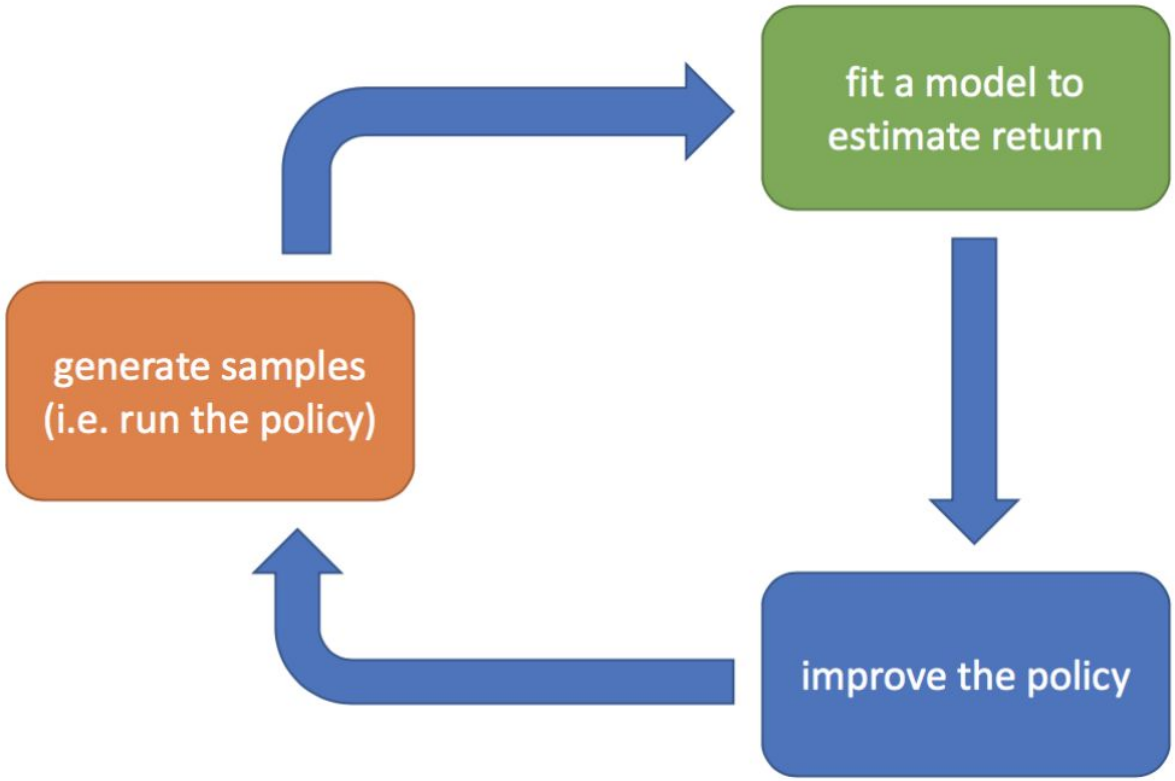


Figure 1: Two ways to choose which route to take when traveling home from work on friday evening.

Simple Definitions:

1. **S** is the state/observation space of an environment
2. **A** is the set of actions the agent can choose
3. **R(s,a)** is a function that returns the reward received for taking action **a** in state **s**
4. **π(a|s)** is the policy needed to learn for optimizing the expected rewarded
5. **T(s′|s,a)** is a transition probability function
6. Learning and using **T(s′|s,a)** explicitly for policy search is model-based

Overview of Model Based RL



Estimate **T(s′|s,a)** based on collected samples

Eg. Supervised Learning

Optimize π **(a|s)**

Eg. Backprop through model

Advantage and Disadvantages of Model-Based

+ Easy to collect data

+ Possibility to transfer across tasks

+ Typically require a smaller quantity of data

- Dynamics models don’t optimize for task performance

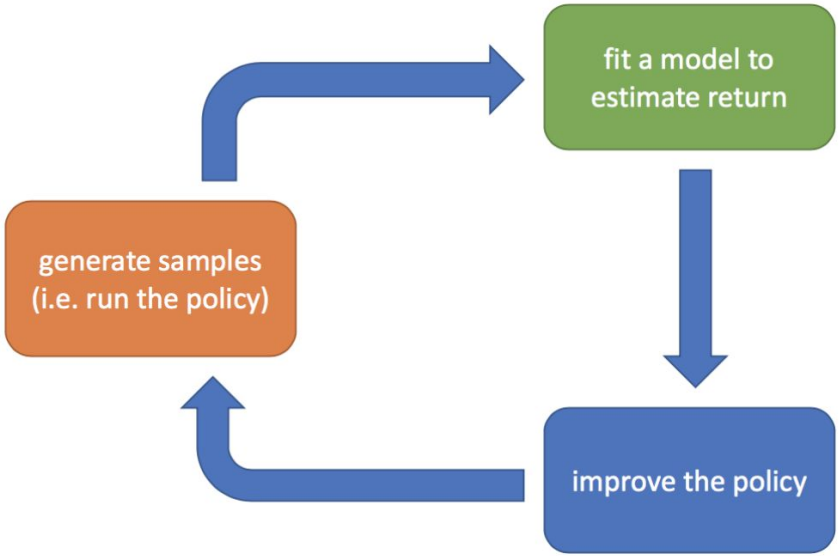
- Sometimes model is harder to learn than a policy

- Often need assumptions to learn (eg. continuity)

Main Contributions of PILCO

Sometimes model is harder to learn than a policy:

one difficulty is that the model is highly biased

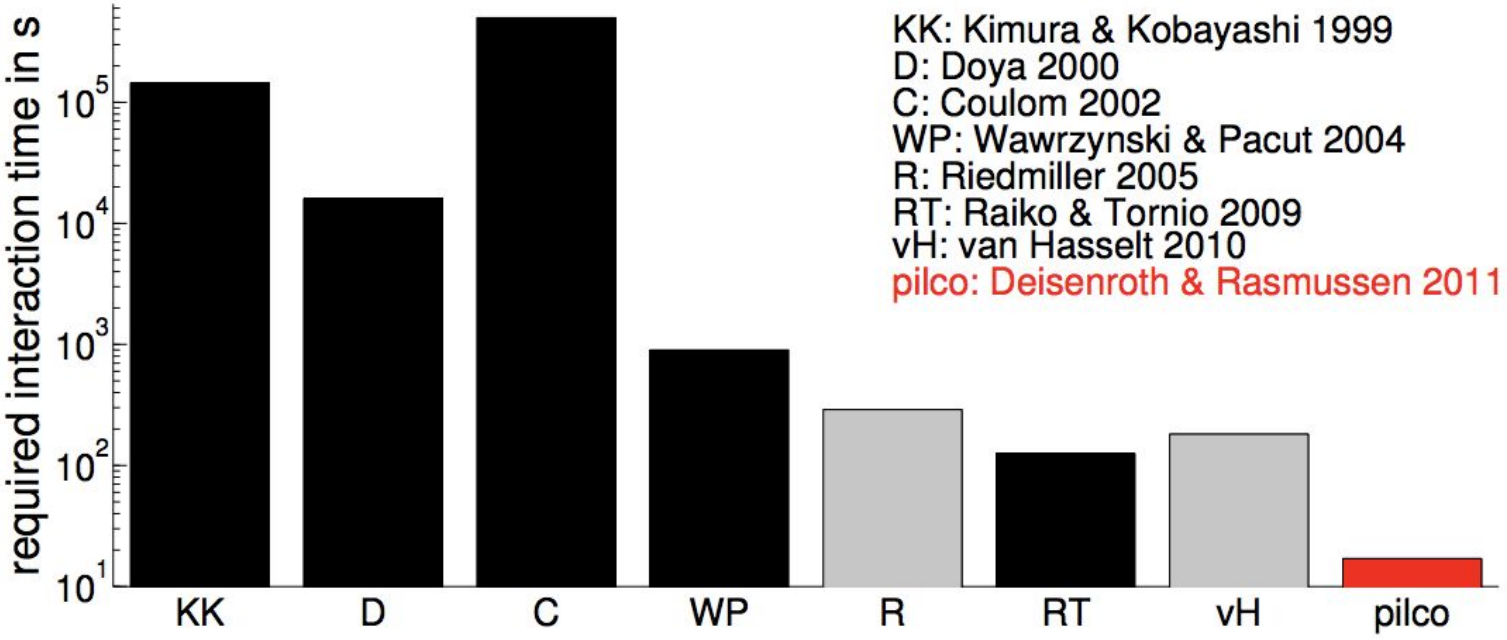


PILCO Solutions:

Reducing model bias by learning a probabilistic dynamics model and explicitly incorporating model uncertainty into long-term planning

1. **probabilistic dynamics model with model uncertainty**
2. **Incorporating uncertainty into long-term planning**

Main Contributions of PILCO



Overflow of their algorithm

Algorithm 1 PILCO

1: **init:** Sample controller parameters $\theta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Apply random control signals and record data.
2: **repeat**
3: Learn probabilistic (GP) dynamics model, see Sec. 2.1, using all data.
4: Model-based policy search, see Sec. 2.2–2.3.
5: **repeat**
6: Approximate inference for policy evaluation, see Sec. 2.2: get $J^\pi(\theta)$, Eqs. (10)–(12), (24).
7: Gradient-based policy improvement, see Sec. 2.3: get $\mathrm{d}J^\pi(\theta)/\mathrm{d}\theta$, Eqs. (26)–(30).
8: Update parameters θ (e.g., CG or L-BFGS).
9: **until** convergence; **return** θ^*
10: Set $\pi^* \leftarrow \pi(\theta^*)$.
11: Apply π^* to system (single trial/episode) and record data.
12: **until** task learned

1. Dynamics Model Learning

$$\Delta_t = \mathbf{x}_t - \mathbf{x}_{t-1} + \varepsilon$$

$$p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathcal{N}(\mathbf{x}_t \,|\, \mu_t, \mathbf{\Sigma}_t)\,, \tag{3}$$

$$\mu_t = \mathbf{x}_{t-1} + \mathbb{E}_f[\Delta_t]\,, \tag{4}$$

$$\mathbf{\Sigma}_t = \mathrm{var}_f[\Delta_t]\,. \tag{5}$$

Policy Evaluation

Presenter: Enes Parildi

2. Policy Evaluation

$$J^{\pi}(\theta) = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}\left[c(\mathbf{x}_t)\right], \quad \mathbf{x}_0 \sim \mathcal{N}(\mu_0, \mathbf{\Sigma}_0),$$

Evaluating this expected return of policy requires all distributions $p(\mathbf{x}_1), \ldots, p(\mathbf{x}_T)$

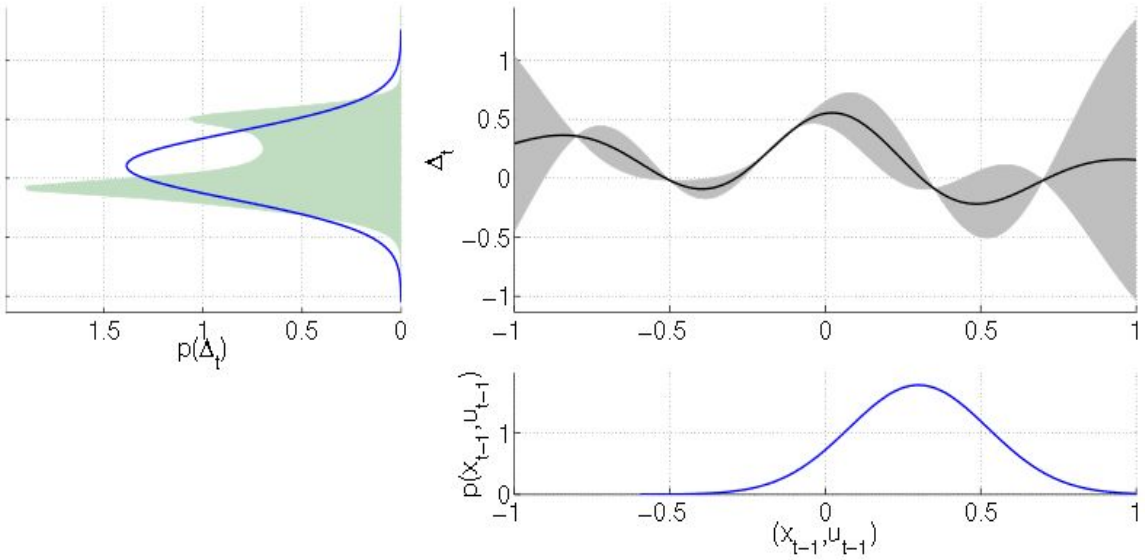
To get these distributions we should obtain first $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ and propagate this distribution through GP model

We assume that $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is gaussian and approximate its distribution using exact moment matching

After propagating $p(\tilde{\mathbf{x}}_{t-1}) = p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ through posterior GP model , the equation that gives predictive distribution of state difference is

$$p(\Delta_t) = \int p(f(\tilde{\mathbf{x}}_{t-1})|\tilde{\mathbf{x}}_{t-1}) p(\tilde{\mathbf{x}}_{t-1})\, \mathrm{d}\tilde{\mathbf{x}}_{t-1},$$

2. Policy Evaluation(cont)



Lower Right Panel: The input distribution

$$p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

Upper Right Panel: Posterior GP model

Upper Left Panel: Blue curve is approximated gaussian with exact mean and variance of green area.

2. Policy Evaluation(cont)

$$\mu_t = \mu_{t-1} + \mu_\Delta$$

$$\Sigma_t = \Sigma_{t-1} + \Sigma_\Delta + \text{cov}[\mathbf{x}_{t-1}, \Delta_t] + \text{cov}[\Delta_t, \mathbf{x}_{t-1}]$$

$$\text{cov}[\mathbf{x}_{t-1}, \Delta_t] = \text{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}]\Sigma_u^{-1}\text{cov}[\mathbf{u}_{t-1}, \Delta_t]$$

If we find μ_Δ and Σ_Δ we can define $p(\mathbf{x}_t)$ as we already know μ_{t-1} and Σ_{t-1}

2.1 Mean Prediction

Using law of iterated expectations, for target dimensions a = 1,…,D, we obtain the mean prediction dimension by dimension

$$\begin{aligned}\mu_{\Delta}^{a} &= \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}[\mathbb{E}_f[f(\tilde{\mathbf{x}}_{t-1})|\tilde{\mathbf{x}}_{t-1}]] = \mathbb{E}_{\tilde{\mathbf{x}}_{t-1}}[m_f(\tilde{\mathbf{x}}_{t-1})] \\ &= \int m_f(\tilde{\mathbf{x}}_{t-1})\mathcal{N}\left(\tilde{\mathbf{x}}_{t-1} \,|\, \tilde{\mu}_{t-1}, \tilde{\mathbf{\Sigma}}_{t-1}\right) \mathrm{d}\tilde{\mathbf{x}}_{t-1} \\ &= \beta_a^{\top} \mathbf{q}_a\end{aligned}$$

$$m_f(\tilde{\mathbf{x}}_*) = \mathbb{E}_f[\Delta_*] = \mathbf{k}_*^{\top}(\mathbf{K} + \sigma_{\varepsilon}^2\mathbf{I})^{-1}\mathbf{y} = \mathbf{k}_*^{\top}\beta$$

2.1 Mean Prediction(cont)

with $\beta_a = (\mathbf{K}_a + \sigma_{\varepsilon_a}^2)^{-1}\mathbf{y}_a$ and $\mathbf{q}_a = [q_{a_1}, \ldots, q_{a_n}]^\top$.
With m_f given in Eq. (7), the entries of $\mathbf{q}_a \in \mathbb{R}^n$ are

$$q_{a_i} = \int k_a(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_{t-1}) \mathcal{N}\left(\tilde{\mathbf{x}}_{t-1} \,|\, \tilde{\mu}_{t-1}, \tilde{\mathbf{\Sigma}}_{t-1}\right) \mathrm{d}\tilde{\mathbf{x}}_{t-1}$$

$$= \frac{\alpha_a^2}{\sqrt{|\tilde{\mathbf{\Sigma}}_{t-1}\mathbf{\Lambda}_a^{-1} + \mathbf{I}|}} \exp\left(-\frac{1}{2}\nu_i^\top (\tilde{\mathbf{\Sigma}}_{t-1} + \mathbf{\Lambda}_a)^{-1}\nu_i\right),$$

$$\nu_i := (\tilde{\mathbf{x}}_i - \tilde{\mu}_{t-1}).$$

Here, ν_i is the difference between the training input $\tilde{\mathbf{x}}_i$ and the mean of the "test" input distribution $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$.

$k_a(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_{t-1})$ is obtained from $k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}') = \alpha^2 \exp\left(-\frac{1}{2}(\tilde{\mathbf{x}} - \tilde{\mathbf{x}}')^\top \mathbf{\Lambda}^{-1}(\tilde{\mathbf{x}} - \tilde{\mathbf{x}}')\right)$

2.2 Covariance Matrix of the Prediction

From Gaussian multiplications and integration, we obtain the entries for Q_{ij} of $\mathbf{Q} \in \mathbb{R}^{n \times n}$

$$Q_{ij} = \frac{k_a(\tilde{\mathbf{x}}_i, \tilde{\mu}_{t-1}) k_b(\tilde{\mathbf{x}}_j, \tilde{\mu}_{t-1})}{\sqrt{|\mathbf{R}|}} \exp\left(\frac{1}{2} \mathbf{z}_{ij}^\top \mathbf{R}^{-1} \tilde{\mathbf{\Sigma}}_{t-1} \mathbf{z}_{ij}\right)$$

After covariance and mean prediction we can get expected return of policy $J^\pi(\theta)$ by summing the expectations calculated like this

$$\mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)] = \int c(\mathbf{x}_t) \mathcal{N}\left(\mathbf{x}_t \mid \mu_t, \mathbf{\Sigma}_t\right) \mathrm{d}\mathbf{x}_t$$

$$c(\mathbf{x}) = 1 - \exp(-\|\mathbf{x} - \mathbf{x}_{\text{target}}\|^2 / \sigma_c^2) \in [0, 1]$$

2.3 Analytic Gradients for Policy Improvement

Both μ_t and $\mathbf{\Sigma}_t$ are functionally dependent on the mean μ_u and the covariance $\mathbf{\Sigma}_u$ of the control signal (and θ) through $\tilde{\mu}_{t-1}$ and $\tilde{\mathbf{\Sigma}}_{t-1}$, respectively.

We obtain the derivative of $\mathrm{d}J^\pi / \mathrm{d}\theta$ by repeated application of the chain-rule: Swap the order or differentiating and summing with $\mathcal{E}_t := \mathbb{E}_{\mathbf{x}_t}[c(\mathbf{x}_t)]$ we obtain

$$\frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\theta} = \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}p(\mathbf{x}_t)} \frac{\mathrm{d}p(\mathbf{x}_t)}{\mathrm{d}\theta} := \frac{\partial \mathcal{E}_t}{\partial \mu_t} \frac{\mathrm{d}\mu_t}{\mathrm{d}\theta} + \frac{\partial \mathcal{E}_t}{\partial \mathbf{\Sigma}_t} \frac{\mathrm{d}\mathbf{\Sigma}_t}{\mathrm{d}\theta}$$

Applying chain rule, we obtain

$$\frac{\mathrm{d}p(\mathbf{x}_t)}{\mathrm{d}\theta} = \frac{\partial p(\mathbf{x}_t)}{\partial p(\mathbf{x}_{t-1})} \frac{\mathrm{d}p(\mathbf{x}_{t-1})}{\mathrm{d}\theta} + \frac{\partial p(\mathbf{x}_t)}{\partial \theta},$$

$$\frac{\partial p(\mathbf{x}_t)}{\partial p(\mathbf{x}_{t-1})} = \left\{ \frac{\partial \mu_t}{\partial p(\mathbf{x}_{t-1})}, \frac{\partial \mathbf{\Sigma}_t}{\partial p(\mathbf{x}_{t-1})} \right\}.$$

2.3 Analytic Gradients for Policy Improvement (cont)

- Here , we focus on $\mathrm{d}\mu_t / \mathrm{d}\theta$

$$\frac{\mathrm{d}\mu_t}{\mathrm{d}\theta} = \frac{\partial \mu_t}{\partial \mu_{t-1}} \frac{\mathrm{d}\mu_{t-1}}{\mathrm{d}\theta} + \frac{\partial \mu_t}{\partial \boldsymbol{\Sigma}_{t-1}} \frac{\mathrm{d}\boldsymbol{\Sigma}_{t-1}}{\mathrm{d}\theta} + \frac{\partial \mu_t}{\partial \theta}.$$

- Since $\mathrm{d}p(\mathbf{x}_{t-1}) / \mathrm{d}\theta$ is known from time step $t-1$ and $\partial \mu_t / \partial p(\mathbf{x}_{t-1})$ is computed from Chain-rule to the equations at the part mean prediction and we conclude with

$$\frac{\partial \mu_t}{\partial \theta} = \frac{\partial \mu_\Delta}{\partial p(\mathbf{u}_{t-1})} \frac{\partial p(\mathbf{u}_{t-1})}{\partial \theta} = \frac{\partial \mu_\Delta}{\partial \mu_u} \frac{\partial \mu_u}{\partial \theta} + \frac{\partial \mu_\Delta}{\partial \boldsymbol{\Sigma}_u} \frac{\partial \boldsymbol{\Sigma}_u}{\partial \theta}.$$

- The first derivative terms above can be obtained from equations in mean prediction part and the second ones depend on policy parametrization.
- Analytic gradient computation of J^π is much more efficient than estimating policy gradients through sampling.
- After getting $\mathrm{d}J^\pi / \mathrm{d}\theta$ using this procedure , policy parameters can be updated with CG or L-BFGS algorithm

Experiments + Result

Presenter: Kavitha Mekala

3. Experiments and Results

- PILCO's success in efficiently learning challenging control tasks, including both standard benchmark problems and high-dimensional control problems.

- PILCO learns completely from scratch by following the steps detailed in the Alg. 1.

- The results discussed in the following are typical, that is they do neither represent best nor worst cases.

3.1 Cart-Pole Swing-up (video)

PILCO was applied to learning to control a real cart-pole system, see Fig 3.

- Cart with mass 0.7 kg running on a track and a freely swinging pendulum of mass 0.325 kg attached to the cart.
- The objective was to learn a controller to swing the pendulum up and to balance it in the inverted position in the middle of the track. A linear controller is not capable of doing this.
- The learned state-feedback controller was a nonlinear RBF network that is

$$\pi(\mathbf{x}, \theta) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{x}) \,,$$
$$\phi_i(\mathbf{x}) = \exp(-\tfrac{1}{2}(\mathbf{x} - \mu_i)^\top \mathbf{\Lambda}^{-1}(\mathbf{x} - \mu_i))$$

with $n = 50$ squared exponential basis functions centered at μ_i. In our experiment, $\theta = \{w_i, \mathbf{\Lambda}, \mu_i\} \in \mathbb{R}^{305}$.

- PILCO successfully learned a sufficiently good dynamics model and controller for this standard benchmark problem fully automatically in only a handful of trials and a total of 17.5 s.

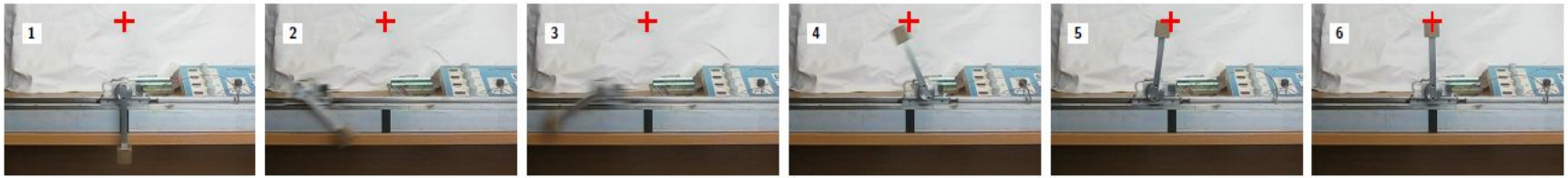


Fig 3. Real cart-pole system. Snapshots of a controlled trajectory of 20 s length after having learned the task. To solve the swing-up plus balancing, PILCO required only 17.5 s of interaction with the physical system.

3.2. Cart-Double-Pendulum Swing-up

PILCO learning a dynamics model and a controller for the cart-double-pendulum swing-up.

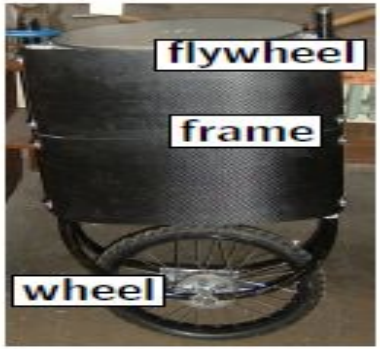
- The objective was to learn the policy π^* to swing the double pendulum up to the inverted position and to balance it with the cart at the start location x.

- A standard control approach to solve the cart-double pendulum task is to design two separate controllers, one for the swing up and one linear controller for the balancing task.

- PILCO fully automatically learned a dynamics model and single nonlinear RBF controller, with n = 200 and $\theta \in \mathbb{R}^{1816}$ to jointly solve the swing-up and balancing. It required about 20-30 trials corresponding to an interaction time of about 60s-90s.

3.3 Unicycle Riding

- They applied PILCO to riding a 5-DoF unicycle in a realistic simulation of the one shown in the Fig.4(a).
- The goal was to ride the unicycle, to prevent it from failing. To solve the balancing task, they used linear controller $\pi(\mathbf{x}, \theta) = \mathbf{A}\mathbf{x} + \mathbf{b}$ with $\theta = \{\mathbf{A}, \mathbf{b}\} \in \mathbb{R}^{28}$.
- PILCO required about 20 trials to learn the dynamics models and a controller that keeps the unicycle upright.



(a) Robotic unicycle.

(b) Histogram (after 1,000 test runs) of the distances of the flywheel from being upright.

Figure 4. Robotic unicycle system and simulation results. The state space is $\mathbb{R}^{12}$, the control space $\mathbb{R}^2$.

3.4 Data Efficiency

Tab. 1. Summarizes the results presented in the paper.

- For each task, the dimensionality of the state and parameter spaces are listed together with the required number of trials and the corresponding total interaction time.
- The table shows that PILCO can efficiently find good policies even in high dimensions that depends on both the complexity of the dynamics model and the controller to be learned.

Table 1. PILCO's data efficiency scales to high dimensions.

	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	≈ 20 s	≈ 60 s–90 s	≈ 20 s–30 s
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$

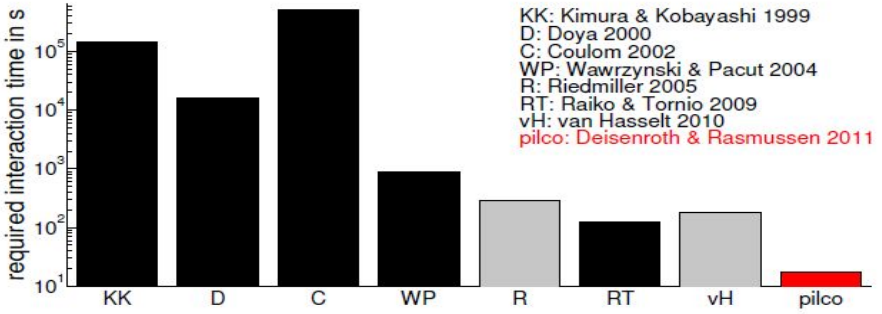


Figure 5. Data efficiency for learning the cart-pole task in the absence of expert knowledge. The horizontal axis chronologically orders the references according to their publication date. The vertical axis shows the required interaction time with the cart-pole system on a log-scale.

4. Discussion

- Trial-and-error learning leads to few limitations in the discovered policy: PILCO is not an optimal control method but it finds a solution for the task.
- PILCO exploits analytic gradients of an approximation to the expected return J^{π} for indirect policy search.
- PILCO obtains gradients with value zero and gets stuck in a local optimum, although it is relatively robust against the choice of the width of the cost in the above equations, there is no guarantee that PILCO always learns with a 0-1 cost.
- One of the PILCO's key benefits is the reduction of model bias by explicitly incorporating model uncertainty into planning and control.
- Moment matching approximation used for approximate inference is typically a conservative approximation.

4. Conclusion (cont)

- The probabilistic dynamics model was crucial to PILCO's learning success.
- Learning from scratch with this deterministic model was unsuccessful because of the missing representation of model uncertainty.
- Since the initial training set for the dynamics model did not contain states close to the target state, the predictive model was overconfident during planning.
- They introduced PILCO, a practical model-based policy search method using analytic gradients for the policy improvement.
- PILCO advances state-of-the-art RL method in terms of learning speed by at least an order of magnitude
- Results in the paper suggests using probabilistic dynamics models for planning and policy learning to account for model uncertainties in the small-sample case, even if the underlying system is deterministic.

Q & A

Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, Steffen Udluft (2016)

Presenters: Tianbao Li, Wei Yu, Yichao Lu, Yatu Zhang

Overview

- GOAL: use model-based reinforcement learning to search policy in stochastic dynamical system
- DIFFICULTY: robust learning of Bayesian Neural Networks with stochastic input variables
- METHOD: approximate by α-divergence, train by gradient-based policy search
- RESULT: better result on real-world scenarios including industrial benchmark (better than Variational Bayes)

Background & Review

Bayesian Neural Networks

- Use distributions to represent parameters
- Give prior distribution on weights P(w)
  - Usually Gaussian priors
  - One of the randomness
- Learn posterior distribution P(w | D)

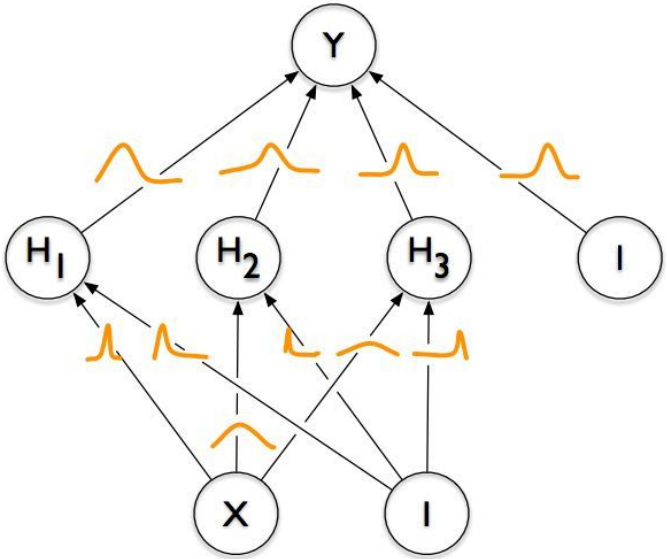


Figure source: Blundell, C. et al. Weight Uncertainty in Neural Networks. ICML 2015.

Notation Definition

- Data D = $\{x_n, y_n\}^N$
  - feature $x_n \in \mathbb{R}^D$ (N * D) target $y_n \in \mathbb{R}^K$ (N * K)
  - $y_n = f(x_n, z_n; W) + \varepsilon_n$
- Network f
  - L layers
  - V_l hidden units in layer l
- Weight W:
  - $W = \{W_l\}^L$
  - $V_l * (V_l + 1)$ matrix +1 for per-layer bias

Stochastic Dynamical System

- Originate: stochastic noise in real-world scenarios
- Randomness:
  - Stochastic input z ~ $\mathcal{N}$ (0 , γ)

$$p(\mathbf{z}) = \prod_{n=1}^{N} \mathcal{N}(z_n|0, \gamma)$$

    - Capture unobserved stochastic features that can affect the network's output
  - Noise in dynamics ε ~ $\mathcal{N}$ (0 , $\mathbf{\Sigma}$)
    - Generate a predictive distribution in form of Gaussian mixture
  - Uncertainty in weights W ~ $\mathcal{N}$ (0 , λ)

$$p(\mathcal{W}) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} \,|\, 0, \lambda)$$

    - Bring uncertainty in weights for better prediction
    - Regularization

Reinforcement Learning Model on BNN

- Likelihood

$$p(\mathbf{Y} \mid \mathcal{W}, \mathbf{z}, \mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{y}_n \mid \mathcal{W}, \mathbf{z}, \mathbf{x}_n) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(y_{n,k} \mid f(\mathbf{x}_n, z_n; \mathcal{W}), \mathbf{\Sigma})$$

- Prior

$$p(\mathcal{W}) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} \mid 0, \lambda)$$

- Posterior (given by Bayes rule)

$$p(\mathcal{W}, \mathbf{z} \mid \mathcal{D}) = \frac{p(\mathbf{Y} \mid \mathcal{W}, \mathbf{z}, \mathbf{X}) p(\mathcal{W}) p(\mathbf{z})}{p(\mathbf{Y} \mid \mathbf{X})}.$$

Reinforcement Learning Model on BNN

- Prediction

$$p(\mathbf{y}_\star \mid \mathbf{x}_\star, \mathcal{D}) = \int \left[\int \mathcal{N}(y_\star \mid f(\mathbf{x}_\star, z_\star; \mathcal{W}), \mathbf{\Sigma}) \mathcal{N}(z_\star | 0, 1)\, dz_\star \right] p(\mathcal{W}, \mathbf{z} \mid \mathcal{D})\, d\mathcal{W}\, d\mathbf{z}$$

- Intractable -> use approximations (variational method)

Variational Bayes

- A typical way
- GOAL: approximate a complex Bayesian network by a simpler network with minimum information divergence
  - Analytical approximation to posterior (Monte Carlo sampling)
  - Derive a lower bound for marginal likelihood
- TODO: select simple network q to surrogate complex network p
- TODO': select distribution q(z) to minimize dissimilarity d(q;p)
- Dissimilarity measure: Kullback Leibler divergence (KL-divergence)

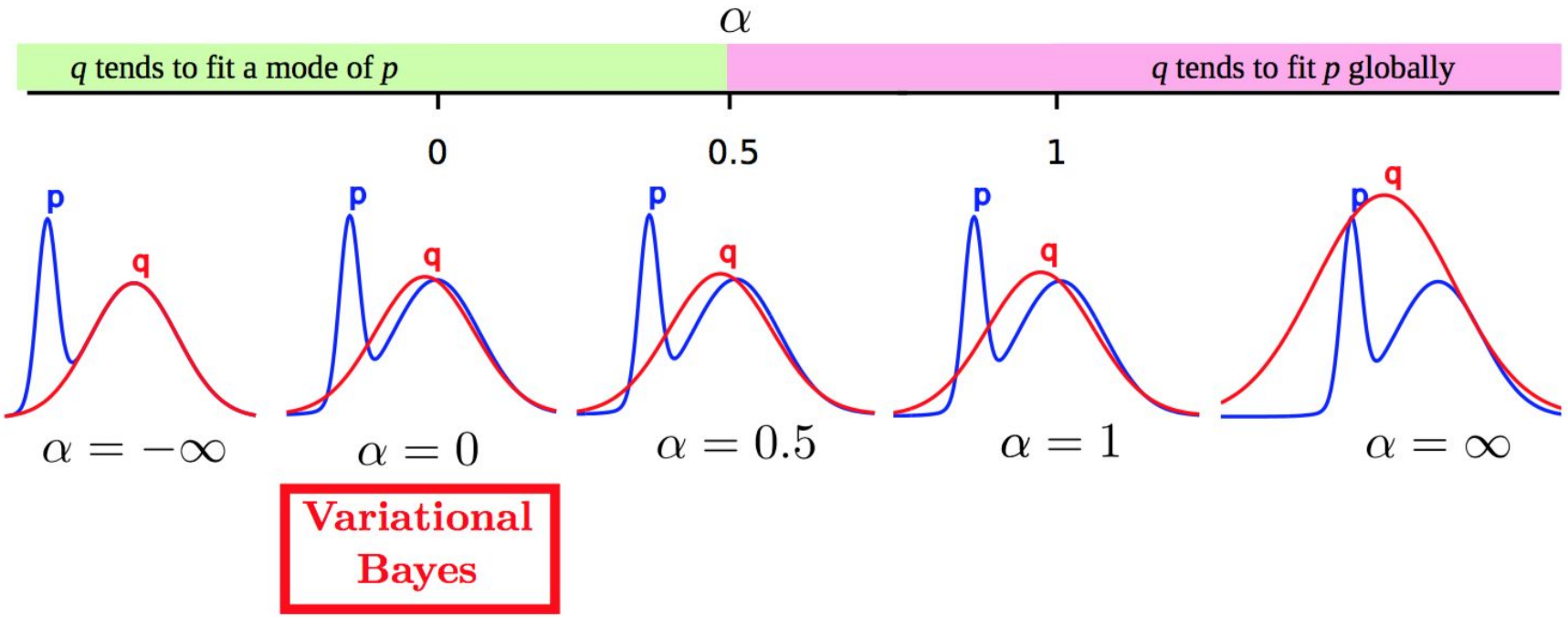
$$\mathrm{KL}(p \mid\mid q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx + \int (q(x) - p(x)) dx$$

- Generalization of KL-divergence -> α-divergence

α-Divergence

- A generalized version of KL-divergence.
  - $$D_\alpha(p||q) = \frac{\int \alpha p(x) + (1-\alpha)q(x) - [p(x)]^\alpha [q(x)]^{1-\alpha} dx}{\alpha(1-\alpha)}, \alpha \in [-\infty, +\infty],$$
- Properties
  - Convex in q for α > 1; Nonnegative;
  - = 0, when p = q, e.g. When α = 0.5, Hellinger Distance $D_{\frac{1}{2}}(p||q) = 2\int \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2 dx$
  - When α goes to 0 or 1, it is equivalent to KL-divergence
    - Interchange of limit and integral and use L'Hospital's rule

α-Divergence vs. Variational Bayes



BNN vs. Gaussian Process

- GP
  - Work extremely well with small amounts of low dimensional data;(hard to scale)
  - Handling of input uncertainty can be done analytically;
  - Sampling dynamics for approximation is infeasible;(the abundance of local optima)
  - No temporal correlation in model uncertainty between successive state transitions.(Markov process)
- BNN
  - Overfitting;
  - Express output model uncertainty (Compared with NN);
  - Sampling dynamics is good for BNN;
  - Recurrent neural network.

❏ Ref: Gal, Y., McAllister, R.T. and Rasmussen, C.E., 2016, April. Improving PILCO with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop* (Vol. 951, p. 2016).

Minimization

- Minimization
  - Similarly, we approximate the posterior distribution with the factorized Gaussian distribution

$$q(\mathcal{W}, \mathbf{z}) = \left[\prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} | m_{ij,l}^{w}, v_{ij,l}^{w}) \right] \left[\prod_{n=1}^{N} \mathcal{N}(z_n \,|\, m_n^z, v_n^z) \right] .$$

- α-Divergence in this case
  - $\mathbf{D}_\alpha[p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D}) || q(\mathcal{W}, \mathbf{z})] = \frac{1}{\alpha(\alpha-1)} \left(1 - \int p(\mathcal{W}, \mathbf{z} \,|\, \mathcal{D})^\alpha q(\mathcal{W}, \mathbf{z})^{(1-\alpha)} \right) d\mathcal{W}\, d\mathbf{z} ,$
  - Direct minimization is infeasible, instead, we optimize an energy function whose minimizer corresponds to a local minimization of α-divergences, with one α-divergence for each of the N likelihood factors.
  - We can represent q as $q(\mathcal{W}, \mathbf{z}) \propto \left[\prod_{n=1}^{N} f(\mathcal{W}) f_n(z_n) \right] p(\mathcal{W}) p(\mathbf{z}) ,$
  - *f* is Gaussian factor that approximates the likelihood factor $p(\mathbf{y}_n \,|\, \mathcal{W}, \mathbf{z}, \mathbf{x}_n)$
  - Black-Box α-Divergence Minimization

Energy function

- Energy function
  - $$E_\alpha(q) = -\log Z_q - \frac{1}{\alpha}\sum_{n=1}^{N} \log \mathbf{E}_{\mathcal{W},z_n\sim q}\left[\left(\frac{p(\mathbf{y}_n \mid \mathcal{W}, \mathbf{x}_n, z_n, \mathbf{\Sigma})}{f(\mathcal{W})f_n(z_n)}\right)^\alpha\right]$$

    $$f(\mathcal{W}) = \exp\left\{\sum_{l=1}^{L}\sum_{i=1}^{V_l}\sum_{j=1}^{V_{l-1}+1}\frac{1}{N}\left(\frac{\lambda v^w_{i,j,l}}{\lambda - v^w_{i,j,l}}w^2_{i,j,l} + \frac{m^w_{i,j,l}}{v^w_{i,j,l}}w_{i,j,l}\right)\right\} \propto \left[\frac{q(\mathcal{W})}{p(\mathcal{W})}\right]^{\frac{1}{N}}$$

    $$f_n(z_n) = \exp\left\{\frac{\gamma v^z_n}{\gamma - v^z_n}z_n^2 + \frac{m^z_n}{v^z_n}z_n\right\} \propto \frac{q(z_n)}{p(z_n)}, \qquad q(\mathcal{W},\mathbf{z}) \propto \left[\prod_{n=1}^{N} f(\mathcal{W})f_n(z_n)\right] p(\mathcal{W})p(\mathbf{z}),$$

    log Zq is the logarithm of the normalization constant of the exponential Gaussian form of q

  - Minimization of Energy function agrees with local minimization of α-Divergence.
- Training
  - The scalable optimization is done in practice by using stochastic gradient descent.

The Reparameterization Trick

- The gradient of the energy function

$$E_\alpha(q) = -\log Z_q - \frac{1}{\alpha}\sum_{n=1}^{N} \log \mathbf{E}_{\mathcal{W},z_n\sim q}\left[\left(\frac{p(y_n|\mathcal{W},x_n,z_n,\Sigma)}{f(\mathcal{W}f_n(z_n))}\right)^\alpha\right],$$

  can be obtained using the reparameterization trick[1].
- For a chosen approximate posterior $q_\phi(z|x)$, one can reparameterize the random variable $\widetilde{z} \sim q_\phi(z|x)$ using a differentiable transformation $g_\phi(\epsilon, x)$ of an auxiliary noise variable ϵ:

$$\widetilde{z} = q_\phi(z|x) \quad \textit{with} \quad \epsilon \sim p(\epsilon).$$

[1] **Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes..**

Policy Search Using BNNs with Stochastic Inputs

Model-based policy search methods include two key parts:

- Learning a dynamics model from data in the form of state transitions (s_t, a_t, s_{t+1}).
- Learning the parameters $\mathcal{W}_\pi$ of a deterministic policy function π that returns the optimal action $a_t = \pi(s_t; \mathcal{W}_\pi)$.

Assuming the dynamics to be stochastic with the true transition model:

$$s_t = f_{true}(s_{t-1}, a_{t-1}, z_t; \mathcal{W}_{true}). \tag{1}$$

The probability distribution $p(s_t|s_{t-1}, a_{t-1})$ can be approximated using a BNN with stochastic inputs:

$$p(s_t|s_{t-1}, a_{t-1}) \approx \int \mathcal{N}(s_t|f(s_{t-1}, a_{t-1}, z_t); \mathcal{W}), \Sigma) q(\mathcal{W}) \mathcal{N}(z_t|0, \lambda) d\mathcal{W} dz_t.$$

Policy Search Using BNNs with Stochastic Inputs

- The parameters $\mathcal{W}_\pi$ of a policy is optimized through minimizing the expected cost over a finite horizon T with respect to the belief $q(\mathcal{W})$.
- Given a cost function $c(s_t)$, the objective to be optimized by the policy search algorithm is

$$J(\mathcal{W}_\pi) = \mathbf{E}\left[\sum_{t=1}^{T} c(s_t)\right].$$

- Assuming that the dynamics are Markovian with respect to the current state and the current action, then

$$J(\mathcal{W}_\pi) = \int \left[\sum_{t=1}^{T} c(s_t)\right] \left[p(s_t|s_{t-1}, a_{t-1})\right] p(s_0) ds_0 \ldots ds_T. \qquad (2)$$

Policy Search Using BNNs with Stochastic Inputs

The objective function $J(\mathcal{W}_\pi)$ can be approximated by sampling methods:

$$
\begin{aligned}
J(\mathcal{W}) &= \int \left[\sum_{t=1}^{T} c(s_t)\right] \left[p(s_t|s_{t-1}, a_{t-1})\right] p(s_0) ds_0 \dots ds_T \\
&= \int \left[\sum_{t=1}^{T} c(s_t^{\mathcal{W},\{z_{1\dots t}\},\{\epsilon_{1\dots t}\},\mathcal{W}_\pi})\right] q(\mathcal{W}) d\mathcal{W} \\
&\quad \left[\prod_{t=1}^{T} \mathcal{N}(\epsilon_t|0,\Sigma)\mathcal{N}(z_t|0,\lambda) d\epsilon_t dz_t\right] p(s_0) ds_0 \\
&\approx \frac{1}{K} \sum_{k=1}^{K} \left[\sum_{t=1}^{T} c(s_t^{\mathcal{W},\{z_{1\dots t}\},\{\epsilon_{1\dots t}\},\mathcal{W}_\pi})\right],
\end{aligned}
$$

where $s_t^{\mathcal{W},\{z_{1\dots t}\},\{\epsilon_{1\dots t}\},\mathcal{W}_\pi}$ is the state obtained at time t in a roll-out generated by using a policy with parameter $\mathcal{W}_\pi$, a transition function parameterized by $\mathcal{W}$ and input noise $z_1, \dots, z_t$, with additive noise values $\epsilon_1, \dots, \epsilon_t$.

Policy Search Using BNNs with Stochastic Inputs

Algorithm 1: Model-based policy search using Bayesian neural networks with stochastic inputs.

Input: $\mathcal{D} = \{s_n, a_n, \Delta_n\}$ for $n \in 1 \dots N$
Fit $q(\mathcal{W})$ and Σ by minimizing the α-divergence
function UNFOLD (s_0)
    Sample $\{\mathcal{W}^1, \dots, \mathcal{W}^K\}$ from $q(\mathcal{W})$
    $C \leftarrow 0$
    for $k = 1 : K$ **do**
        for $t = 0 : T$ **do**
            $z_{t+1}^k \sim \mathcal{N}(0, \gamma)$
            $\Delta_t \leftarrow f(s_t, \pi(s_t; \mathcal{W}_\pi), z_{t+1}^k; \mathcal{W}^k)$
            $\epsilon_{t+1}^k \sim \mathcal{N}(0, \Sigma)$
            $s_{t+1} \leftarrow s_t + \Delta_t + \epsilon_{t+1}^k$
            $C \leftarrow C + c(s_{t+1})$
    return C/K
Fit $\mathcal{W}_\pi$ by optimizing $\frac{1}{N} \sum_{n=1}^{N}$ UNFOLD(s_n)

Experiments: Wet Chicken

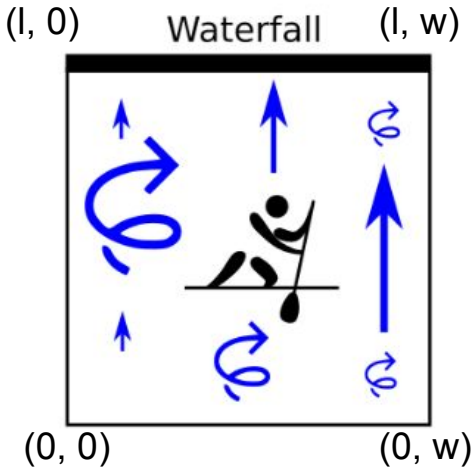
Wet Chicken Description

- A canoeist paddles in a 2D river starting at the origin (0,0), with position given by (x_t, y_t).

- The river has a waterfall at $y_t = l$. The canoeist has to start over at the origin after falling into the waterfall

- The canoeist performs an action $(a_{t,x}, a_{t,y}) \in [-1, 1]^2$ that represents the direction and magnitude of paddling at each time step **t**

- At each time step **t**, The canoeist receives a reward $r_t = -(l - y_t)$



Wet Chicken Description

- However, the system has stochastic turbulences $s_t\tau_t$ and drift v_t that is dependent on horizontal position x_t, where:

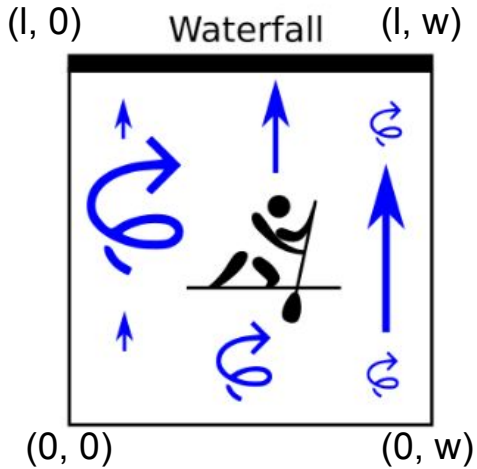
$$v_t = 3x_t w^{-1} \qquad s_t = 3.5 - v_t \qquad \tau_t \sim \text{Unif}([-1, 1])$$

- Canoeist's new position under this dynamics is:

$$x_{t+1} = \begin{cases} 0 & \text{if} \quad x_t + a_{t,x} < 0 \\ 0 & \text{if} \quad \hat{y}_{t+1} > l \\ w & \text{if} \quad x_t + a_{t,x} > w \\ x_t + a_{t,x} & \text{otherwise} \end{cases}, \qquad y_{t+1} = \begin{cases} 0 & \text{if} \quad \hat{y}_{t+1} < 0 \\ 0 & \text{if} \quad \hat{y}_{t+1} > l \\ \hat{y}_{t+1} & \text{otherwise} \end{cases},$$

where,

$$\hat{y}_{t+1} = y_t + (a_{t,y} - 1) + v_t + s_t\tau_t$$



Bi-Modality and Heteroskedasticity

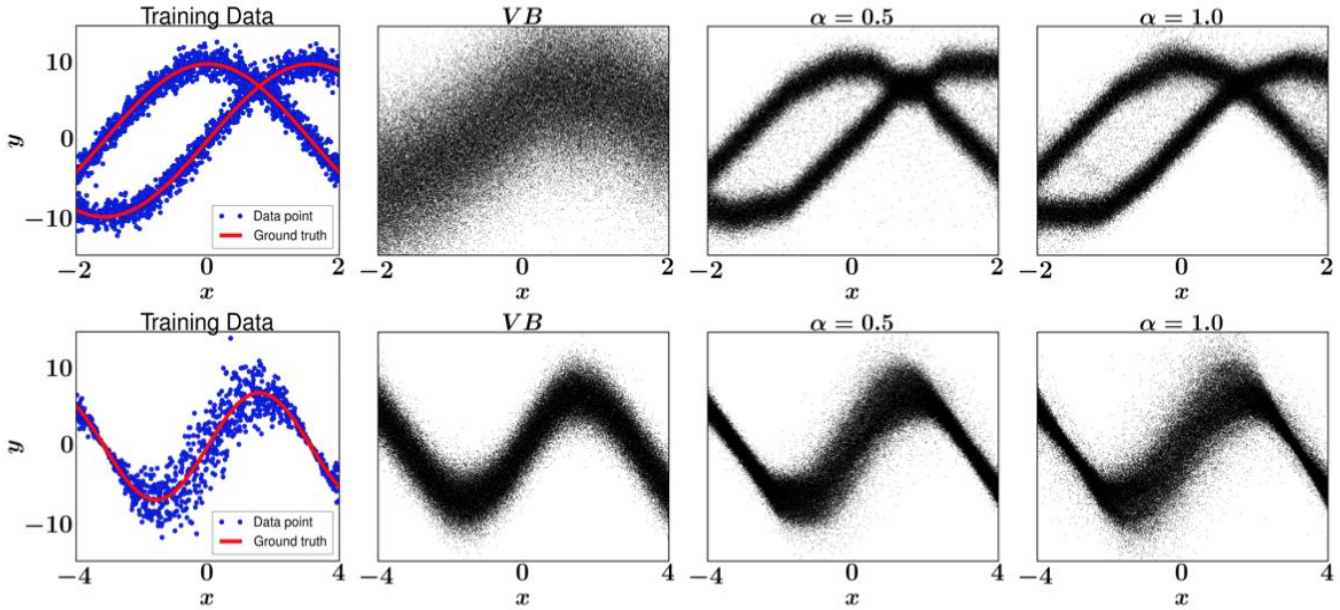
$$\hat{y}_{t+1} = y_t + (a_{t,y} - 1) + v_t + s_t \tau_t$$

- The transition dynamics of this system exhibit complex stochastic patterns
- **Bi-modality**: As canoeist moves closer to the waterfall, the predictive distribution for the next state becomes increasingly bi-modal
- **Heteroskedasticity**: noise variance is different depending on current state
- Challenging for traditional model-based reinforcement learning methods.
- Need to tackle with models that can capture both bi-modality and heteroskedasticity patterns in the predictive distribution (BNN optimized using alpha divergence).

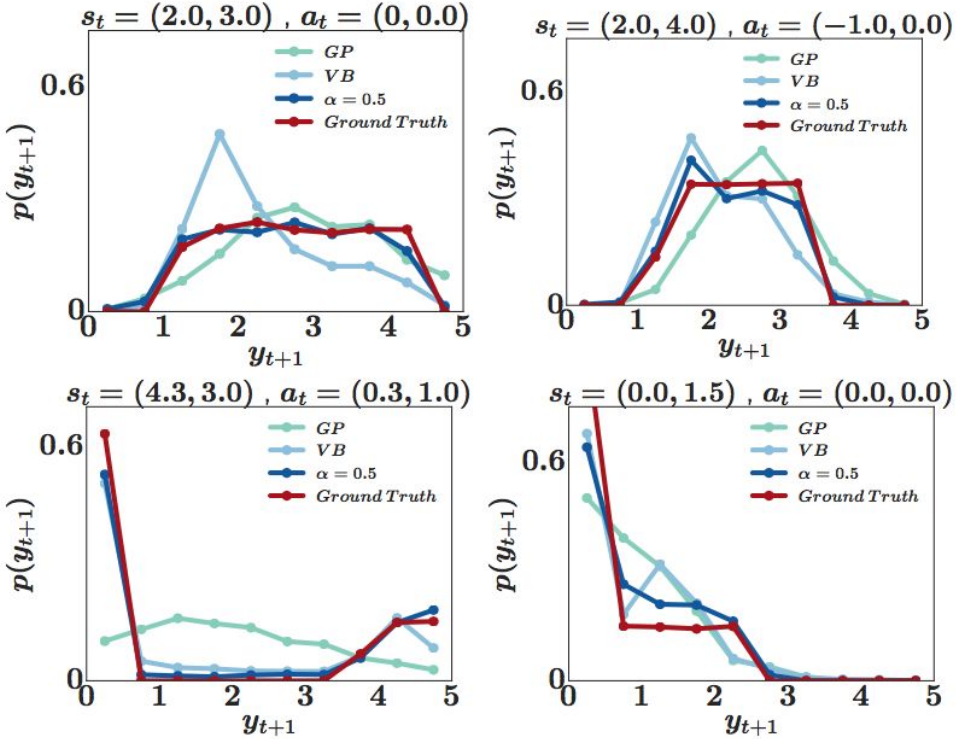
Bi-Modality and Heteroskedasticity - Toy Dataset

Bi-Modal

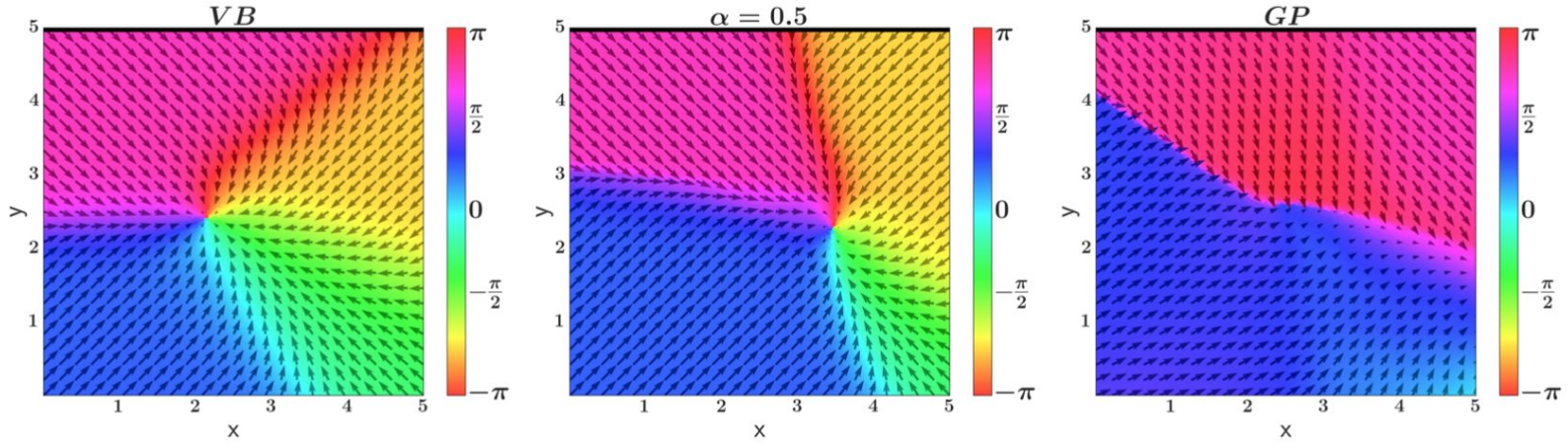
Heteroskedastic



Wet Chicken Results

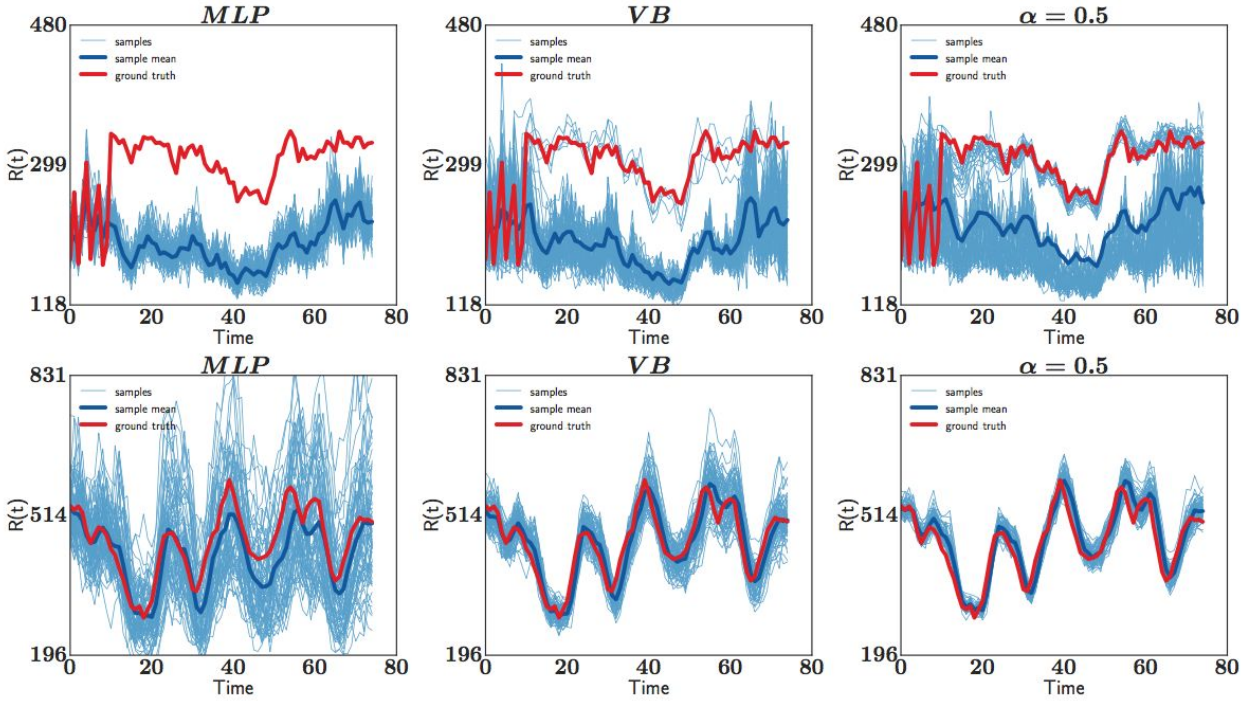


Wet Chicken Results



Experiments: Industrial Applications

Results


MLP
VB
$\alpha = 0.5$
samples
sample mean
ground truth
R(t)
Time
480
299
118
831
514
196
0
20
40
60
80

Results

Dataset	MLP	VB	α=0.5	α=1.0	GP
Log-Likelihood					
WetChicken	-1.755±0.003	-1.140±0.033	**-1.057±0.014**	-1.070±0.011	-1.722±0.011
Turbine	-0.868±0.007	-0.775±0.004	**-0.746±0.013**	-0.774±0.015	-2.663±0.131
Industrial	0.767±0.047	1.132±0.064	**1.328±0.108**	1.326±0.098	0.724±0.04
Avg. Rank	4.3±0.12	2.6±0.16	**1.3±0.15**	2.1±0.18	4.7±0.12

Table 2: Model test error and test log-likelihood for different benchmarks. Printed are average values over 5 runs with respective standard errors. Bottom row is the average rank over all 5×3 runs.

Dataset	MLP	VB	α=0.5	α=1.0	GP	PSO-P
Wetchicken	-2.71±0.09	-2.67±0.10	**-2.37±0.01**	-2.42±0.01	-3.05±0.06	-2.34
Turbine	-0.65±0.14	-0.45±0.02	**-0.41±0.03**	-0.55±0.08	-0.64±0.18	NA
Industrial	-183.5±4.1	-180.2±0.6	-174.2±1.1	**-171.1±2.1**	-285.2±20.5	-145.5
Avg. Rank	3.6±0.3	3.1±0.2	**1.5±0.2**	2.3±0.3	4.5±0.3	

Table 1: Policy performances over different benchmarks. Printed are average values over 5 runs with respective standard errors. Bottom row is the average rank over all 5×3 runs.

Conclusion

Conclusion

- GOAL: use model-based reinforcement learning to search policy in stochastic dynamical system
- DIFFICULTY: robust learning of Bayesian Neural Networks with stochastic input
- METHOD: approximate by α-divergence, train by gradient-based policy search
- RESULT: Obtained state-of-the-art policies obtained in industrial problems, with rollouts sampled from the BNN model.