Journal of Machine Learning Research * (2021) * Submitted *; Published *

Empirical Evaluation: Variational Inference Reinforcement Learning in Pyro

Jianlin Li jianlin.li@uwaterloo.ca
University of Waterloo
Canada

Editor: *

Abstract

The main focus of this project is to empirical evaluate mainly **Merlin** (Levine, 2018), **Virel** (Fellows et al., 2019) and the **Pyro**[1] (Bingham et al., 2019) probabilistic programming language (PPL). **Merlin** and **Virel** are two frameworks aimed to solve the problem of maximum entropy reinforcement learning (MERL) via variational inference (VI). Levine (2018), and Fellows et al. (2019) have already shown that reinforcement learning (RL) tasks could be reformed into probabilistic inference tasks in general, which in principle allows us to bring to bear a wide array of approximate inference tools. Deep universal Probabilistic programming languages, the inference engine of which has stochastic variational inference (SVI) algorithms (Wingate and Weber, 2013) builtin, could solve such inference problems; however, was not applied.

I developed **rl.Pyro**[2] as a proof-of-concept, which implements two of the most classic classic *policy-based* reinforcement learning algorithms, namely **Reinforce** (Thomas and Brunskill, 2017; Shi et al., 2019) and **Actor-Critic** (Mnih et al., 2016; Haarnoja et al., 2018), in **Pyro**, a deep universal PPL implemented in Python and supported by PyTorch (Paszke et al., 2019) on the backend, and empirically evaluated these algorithms on the CartPole environment from OpenAI Gym benchmark (Brockman et al., 2016).

The experimental results [3] validated the theoretical connection between MERL and VI. More importantly, The experimental results confirmed that (1) **Pyro** is express enough to implement *policy-based* RL algorithms, (2) the performance of **Pyro** version of the algorithm is satisfying, and (3) modeling and training are better decoupled using **Pyro**.

Keywords: Reinforcement Learning, Variational Inference, Probabilistic Programming Languages, **Pyro**

1. Introduction

In this report, I address the problem of variational inference reinforcement learning (VIRL), i.e. achieving the same object as maximum entropy reinforcement learning via **Pyro**'s built-in stochastic variational inference (SVI) algorithm, as a black-box tool, instead of performing gradient descent in **PyTorch** manually.

This project is not only a empirical evaluation of **Merlin** (Levine, 2018) and **Virel** (Fellows et al., 2019) , but also of the **Pyro** (Bingham et al., 2019) PPL. To address the problem of VIRL, we propose the following research questions:

1. `https://pyro.ai/`
2. Code available on Github: `https://github.com/ljlin/rl.Pyro`
3. Available on Colab: `https://drive.google.com/drive/folders/1PE5lqtHGvrnrFMDpe3vxbYdCWJAJhlh6`

- **RQ1.1:** Is the expressiveness of these probabilistic programming languages sufficient engouh to express reinforcement learning algorithms?
- **RQ1.2:** Can modeling and training be better decoupled using probabilistic programming languages?
- **RQ2:** Whether the performance of stochastic variational inference algorithm in the inference engine of a probabilistic programming language is sufficient enough to support the training in reinforcement learning?

RQ1 is the more conceptual question, which will be discussed in Section 2, while **RQ2** it the more empirical one, which would answered in Section 4.

Contributions.

- A **Pyro** module **rl.Pyro**, which implements the **Pyro** version of **Reinforce** and **Actor-Critic** algorithm, is developed as a proof-of-concept.
- Experiments conducted in CartPole environment from the OpenAI Gym benchmark.
- Contribute a tutorial[4] to **Pyro**'s documentation[5].

2. Techniques to tackle the problem of variational inference RL

I will briefly review previous work concerning this problem, describe the technique that I developed, and a brief description of the existing techniques that I will compare to.

2.1 Reinforcement Learning

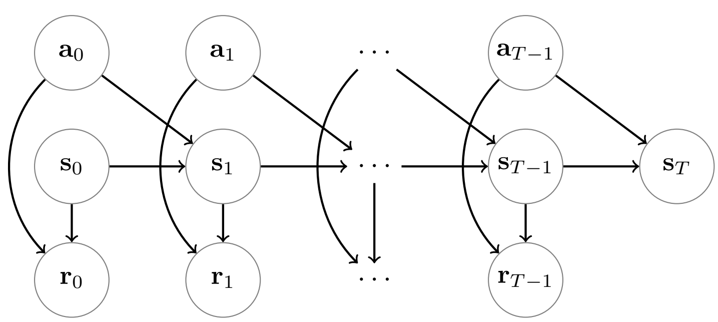


Figure 1: The probabilistic graphical model for the reinforcement learning problem: A Bayesian network w.r.t. the factorization of the *trajectory distribution* $p(\tau)$ for a finite trajectory $\tau = \mathbf{s}_0, \mathbf{a}_0, \mathbf{r}_0, \mathbf{s}_1, \mathbf{a}_1, \mathbf{r}_1, \ldots, \mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{r}_{T-1}, \mathbf{s}_T$.

First, we present the formalization of reinforcement learning. But we only simply list the symbols and notations used in this report due to the lack of space.

$\mathbf{s}_t \in \mathcal{S}$, $\mathbf{a}_T \in \mathcal{A}$ and $\mathbf{r}_t$ to denote the state, action, and immediate reward at time stamp t. $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ stands for the stochastic transition dynamics, which are in general unknown. T stands for the horizon, α is the temperature, and γ is the discount factors. $\tau = \mathbf{s}_0, \mathbf{a}_0, \mathbf{r}_0, \mathbf{s}_1, \mathbf{a}_1, \mathbf{r}_1, \ldots, \mathbf{s}_i, \mathbf{a}_i, \mathbf{r}_i, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{r}_{T-1}, \mathbf{s}_T$ is a finite trajectory.

A standard reinforcement learning policy search problem is then given by the following maximization: $\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p(\mathbf{s}_t, \mathbf{a}_t|\theta)}[\gamma^t \cdot \mathbf{r}_t]$. This optimization problem aims to find a vector of policy parameters θ that maximize the discounted total expected reward

4. I will create a pull request to submit this report to the **pyro**'s official Github repository.
5. `https://pyro.ai/examples/`

$\sum_t \gamma^t \cdot \mathbf{a}_t$ of the policy. The expectation is taken under the policy's *trajectory* distribution $p(\tau)$, given by $p(\tau) = p(\mathbf{s}_1, \mathbf{a}_t, \ldots, \mathbf{s}_T, \mathbf{a}_T|\theta) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{a}_t|\mathbf{s}_t, \theta)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. $p(\mathbf{a}_t|\mathbf{s}_t, \theta)$ can often be written as $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$. The *probabilistic graphical model* (PGM) is shown in Fig. 1.

2.2 Probabilistic programming language: Pyro 101

Pyro (Bingham et al., 2019) enables flexible and expressive deep probabilistic modeling, unifying the best of modern deep learning and Bayesian modeling.

`pyro.factor`[6] could add arbitrary log probability factor to a probabilisitic model, which allow use define a computational stochastic model with normalized or normalized likelihood density.

Pyro's built-in stochastic variational inference engine, called `pyro.infer.SVI`, could solve the inference problem by maximizing the ELBO objective.

2.3 Reinforcement learning as probabilistic inference

Levine (2018) designed the **Merlin** framework, and claimed that soft REINFORCE (soft policy gradient) could be reformed into a probabilistic inference problem. Fellows et al. (2019) designed the **Virel** framework, and claimed that soft actor-critic algorithm could also be reformed into an inference procedure.

Levine (2018) established the connection between reinforcement learning and inference in probabilistic models by introducing the notion of "optimality variables" $\mathcal{O}$. Levine (2018) have also discussed how maximum entropy reinforcement learning (Haarnoja et al., 2017, 2018) is equivalent to exact probabilistic inference in the case of deterministic dynamics and variational inference in the case of stochastic dynamics.

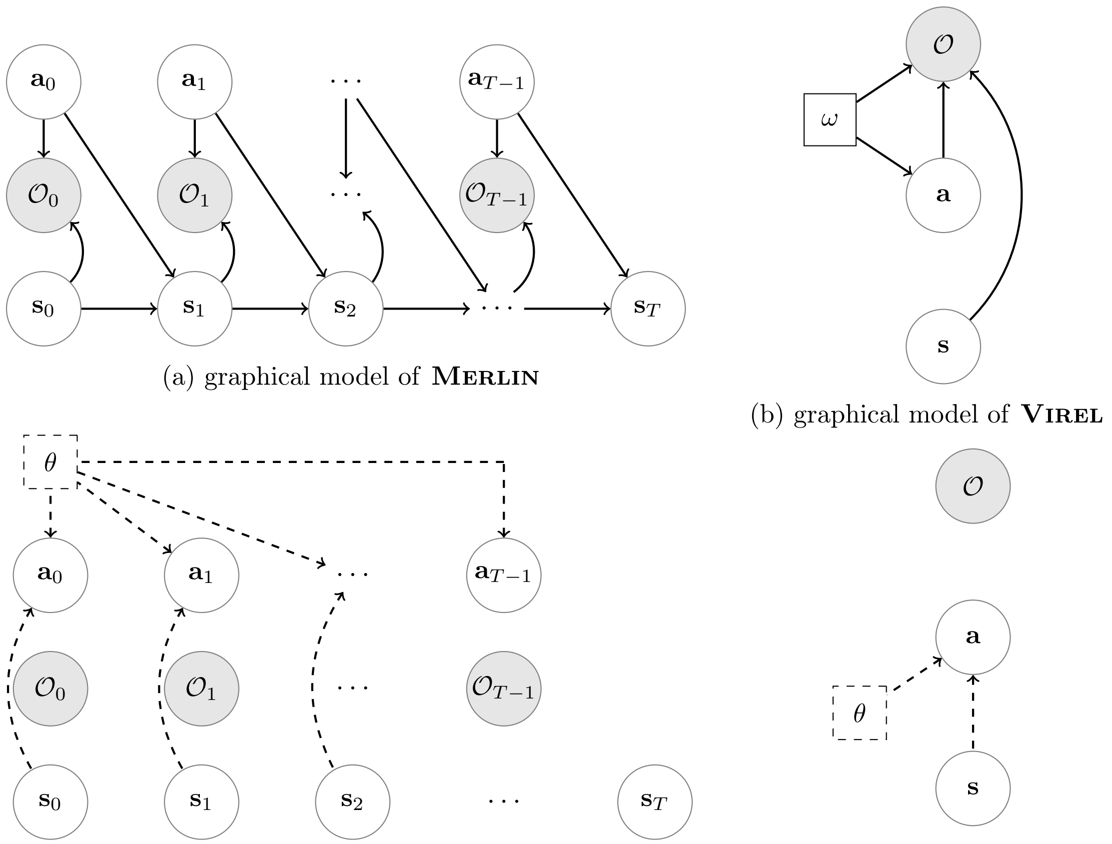
The probabilistic graphic models of **Merlin** and **Virel** are shown in Fig. 2. These "optimality variables" $\mathcal{O}_t$ depends on the reward $\mathbf{r}_t$, but it's okay that we omit $\mathbf{r}_t$ in this Bayesian network if we regard $\mathbf{r}_t$ is a function of $\mathbf{s}_t$ and $\mathbf{a}_t$. Then we condition on the optimality variables being true and then infer the most probable action (distributions). Fig. 2a and Fig. 2b are two Bayesian networks stand for two conditional distributions. And we could apply SVI to these two Bayesian networks to approximate the poster distribution, which induces optimal policies.

The **Merlin** models exponential cumulative rewards over entire trajectories. In contrast, **Virel**'s variational policy models a single step, and a function approximator (the Q-network) is used to model future expected rewards. The resulting KL divergence minimization for **Merlin** is therefore much more sensitive to the value of temperature. (Fellows et al., 2019). Thus **Merlin** results in a trajectory-sampling-based Monte Carlo policy gradient algorithm (i.e. **Reinforce**), which is on-policy, while **Virel** leading to an experience-sampling-based **Actor-Critic** algorithm, which is off-policy, uses temporal difference update and utilizes a memory reply buffer to improve sample efficiency.

2.4 Summary: messages for experts.

Here are messages for experts who are familiar with one but not all of VI, PPL, and RL:

6. `https://docs.pyro.ai/en/stable/primitives.html#pyro.primitives.factor`



(a) graphical model of **Merlin**

(b) graphical model of **Virel**

(c) graphical model of **Merlin**

(d) graphical model of **Virel**

Figure 2: Probabilistic graphical models for **Merlin** (Levine, 2018) and **Virel** (Fellows et al., 2019) (variational approximations are dashed), where θ is the policy parameters, and ω is the parameters of the Q-network.

```python
def guide(env=None, trajectory=None):
    pyro.module("policy_net", policy_net)
    S, A, R, D, step = [], [], [], [], 0

    obs = env.reset()
    done = False
    while not done:
        S.append(obs)
        D.append(done)
        action = pyro.sample(
            f"action_{step}",
            pyro.distributions.Categorical(
                policy_net(obs)
            )
        ).item()
        obs, reward, done, _ = env.step(action)
        A.append(action)
        R.append(reward)
        step += 1
    S.append(obs)
    D.append(done)

    # send the trajectory to the model program
    trajectory["S"] = S
    trajectory["A"] = A
    trajectory["R"] = R
    trajectory["D"] = D
```

```python
def model(env=None, trajectory=None):
    S, R = trajectory["S"], trajectory["R"]
    for step in pyro.plate("trajectory", len(R)):
        action = pyro.sample(
            f"action_{step}",
            pyro.distributions.Categorical(
                torch.ones(ACT_N) / ACT_N
            )
        )
        pyro.factor(
            f"discount_{step}",
            torch.log(GAMMA)
        )
        pyro.factor(
            f"reward_{step}",
            R[step] / TEMPERATURE
        )

def train():
    adma = pyro.optim.Adam({"lr": LEARNING_RATE})
    svi  = pyro.infer.SVI(
        model, guide, adma,
        loss = pyro.infer.Trace_ELBO()
    )
    pyro.clear_param_store()
    for epi in range(EPISODES):
        svi.step(env, trajectory={})
```

Figure 3: The REINFORCE(PYRO) pseudocode*: the agent model conditioned on optimality (`model`), approximate posterior (`guide`), and trace implementation of ELBO-based (`loss = pyro.infer.Trace_ELBO()`) stochastic variational inference `svi`.
`policy_net` is a `torch.nn.Module` object. `pyro.module` calls `pyro.param` on every parameter of a `torch.nn.Module`.
* `https://github.com/ljlin/rl.pyro/blob/main/REINFORCE.py`

- There's two main methods to tackle the Bayesian inference problem: Markov Chain Monte Carlo (MCMC), a *sampling-based* approach, and variational inference (VI), an *optimization-based* approach.
- Counterintuitively, probabilistic programming is not about writing software that behaves probabilistically. Instead, probabilistic programming is a tool for statistical modeling. (Sampson, 2013)
- Stochastic structural variational inference (SVI) uses probabilistic graphical models, mainly Bayesian networks, but variational inference reinforcement learning (VIRL) is not Bayesian RL. VIRL is maximum entropy reinforcement learning (MERL).

3. RL.PYRO implementation

We discuss the **RQ1** in this section. **MERLIN** is implemented as REINFORCE(PYRO) (shown in Fig. 3). **VIREL** is implemented as ACTOR-CRITIC(PYRO) (shown in Fig. 4).

#SLOC & Workload # lines of code (#LOC): 1,121. Github Pulse: excluding merges, 1 author has pushed 37 commits, **7,616 ++** additions and **6,495 --** deletions (#LOC).

RQ1.1: Is PYRO expressive enough to implement RL algorithms? Yes, but only for policy-based RL algorithms. PYRO can only be applied if there is a policy distribution, and SVI can only be applied if the objective of the RL problem can be reformed into a KL-divergence.

```python
def model_unif(S):
    with pyro.plate("batch", S.shape[0]):
        A = pyro.sample(
            "action",
            pyro.distributions.Categorical(
                torch.ones(ACT_N) / ACT_N
            )
        )
        qvalues = Qt(S).gather(
            index=A.view(-1, 1), dim=-1
        ).squeeze()
        pyro.factor(
            "reward", qvalues / TEMPERATURE
        )
```

(a) Conditional model

```python
def model_softmaxQ(S):
    with pyro.plate("batch", S.shape[0]):
        probs = torch.nn.functional.softmax(
            Qt(S) / TEMPERATURE,
            dim=-1
        )
        softmax_greedy_policy =
            pyro.distributions.Categorical(
                probs
            )
        A = pyro.sample(
            "action",
            softmax_greedy_policy
        )
```

(b) Unconditional model

```python
def guide(S):
    pyro.module("policy_net", policy_net)
    with pyro.plate("batch", S.shape[0]):
        A = pyro.sample(
            "action",
            pyro.distributions.Categorical(
                policy_net(S)
            )
        )
```

(c) Approximate posterior

```python
def update_networks():
    # Sample a minibatch (s, a, r, t, d)
    S, A, R, T, D, N = buf.sample(MINIBATCH_SIZE)
    # Maximisation (M-)step: policy evaluation
    # Temporal Difference loss (MSE) for Q
    loss = torch.nn.functional.mse_loss(..., ...)
    OPT_Q.step()
    # Expectation  (E-)step: policy improvement
    svi.step(S)
```

(d) Expectation-Maximisation (EM) algorithm

Figure 4: The Actor-Critic(pyro) pseudocode*: the probabilistic model of the softmax greedy policy w.r.t `Q` (`model_softmaxQ` in Fig. 4b) is semantically equivalent to the conditional model `model_softmaxQ` in Fig. 4b (uniform distribution as prior conditioned on rewards). `policy_net`, `Q` and `Qt` are `torch.nn.Module` objects. (`Q` and `Qt` are the twin Q-networks.)

* `https://github.com/ljlin/rl.pyro/blob/main/AC.py`

RQ1.2: Is modeling and training better decoupled ? Yes, modeling and training are better decoupled to a satisfying extent, although not completely decoupled. `guide` in Fig. 3 play an episode with the environment, but have to send the sampled trajectory to `model` via shared memory, it would be more elegant if **PYRO** impose message-passing diagram here.

4. Empirical evaluation

We empirically compare techniques and answer **RQ2** in this section.

Machine. All experiments are performed on a Google Colab Pro+ GPU runtime (GPU: Tesla P100-PCIE-16GB, CPU: Intel(R) Xeon(R) CPU @ 2.30GHz 2 Cores, RAM: 12G).

Software versions. Python 3.7.12, **PYRO** 1.8.0, **PYTORCH** 1.10.0 and CUDA 11.2.

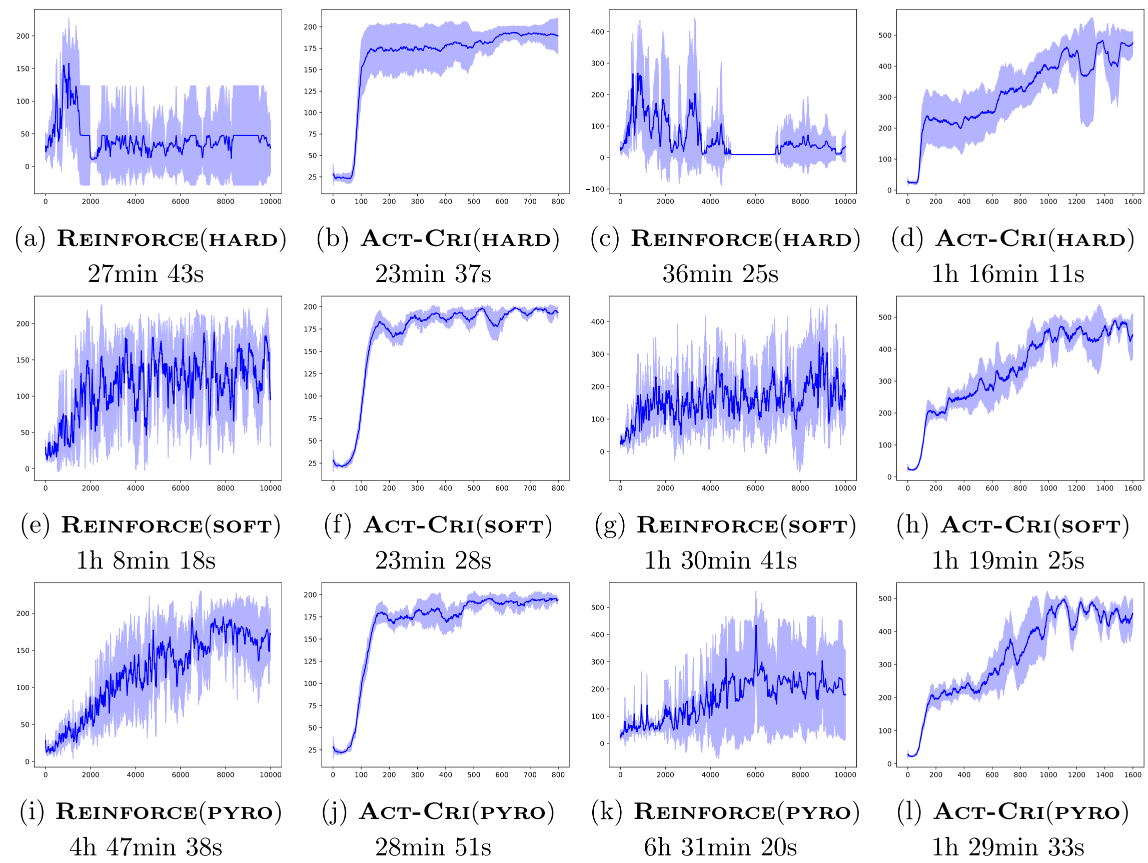
Environments. CartPole-v0 (total reward limit 200) and CartPole-v1 (total reward limit 500) environment from OpenAI Gym benchmark[7] (Brockman et al., 2016).

Implementation. The **RL.PYRO**[8] library implemented the **REINFORCE** (Monte Carlo Policy Gradient) and the **ACTOR-CRITIC** algorithm. There are 3 versions, namely **HARD**, **SOFT** and **PYRO**, of both these two algorithms: **HARD** versions are vanilla versions (vanilla policy gradient and Advantage Actor-Critic), **SOFT** versions refer to maximum entropy versions (soft REINFORCE and soft actor-critic), and **PYRO** versions are also maximum entropy versions but implemented in **PYRO** by me. **SOFT** and **HARD** versions are from CS885 slides and textbooks. **PYRO** versions based on the theoretical foundation from **MERLIN** (Levine, 2018) and **VIREL** (Fellows et al., 2019).

Default hyper parameters. learning rate is $5 * 10^-4$, mini batch size is 64, temperature $\alpha = 1$ and discount factor $\gamma = 0.99$. All agents are trained with a discount factor $\gamma \neq 1$, but tested with no discount (i.e. $\gamma = 1$).

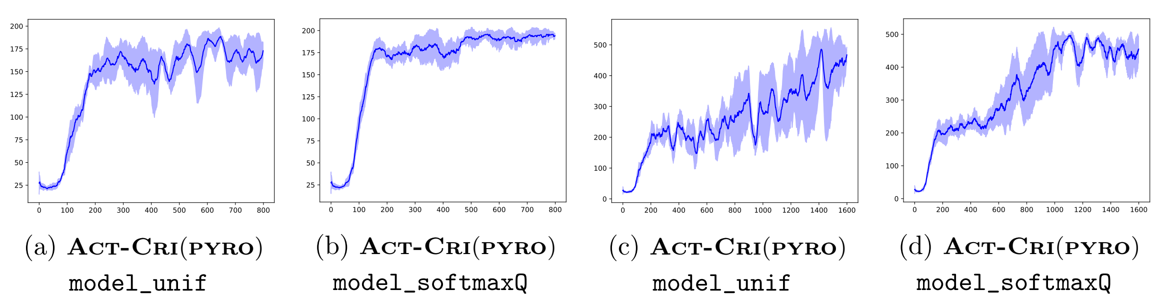
Training rewards & time consumption. Shown in Fig. 5. average rewards (running average of 10) The curve should show the averaged (undiscounted) cumulative total reward (y-axis, averaged across 5 trials with fixed random seeds $1, 2, 3, 4, 5$) against the number of episodes (x-axis), where averaged (undiscounted) cumulative total reward is the averaged episodic reward of last 25 episodes. **Choice of the model program.** `model_unif` was used for **ACTOR-CRITIC**(**PYRO**) in Fig. 6a and Fig. 6c, while `model_softmaxQ` was used for **ACTOR-CRITIC**(**PYRO**) in Fig. 6b and Fig. 6d. `model_unif` should be semantically equivalent to `model_softmaxQ`, but `model_softmaxQ` performs better. This might due to `model_softmaxQ` is more numerical robust. **Temperature.** Fig. 7 shows how temperature α effect **REINFORCE**(**PYRO**) and **ACTOR-CRITIC**(**PYRO**), and it confirms what was claimed by Fellows et al. (2019)–**MERLIN** is much more sensitive to the value of temperature, as it could be seen from the figure that **REINFORCE**(**PYRO**) behaves worse with both $\alpha = 0.1$ and $\alpha = 10$. Lower temperature will push the agent to become more deterministic, thus reducing exploration, and $\alpha = 0.1$ is small enough to let the algorithm converge to a sub-optimal policy. Higher temperature will increase the randomness of the policy distribution, and $\alpha = 100$ could already force this algorithm to produce an almost uniformly random policy. **REINFORCE**(**PYRO**) is better that **REINFORCE**(**SOFT**) since Fig. 7d fails while Fig. 7c still improves.

7. `https://gym.openai.com/`
8. Code available on Github: `https://github.com/ljlin/rl.Pyro`



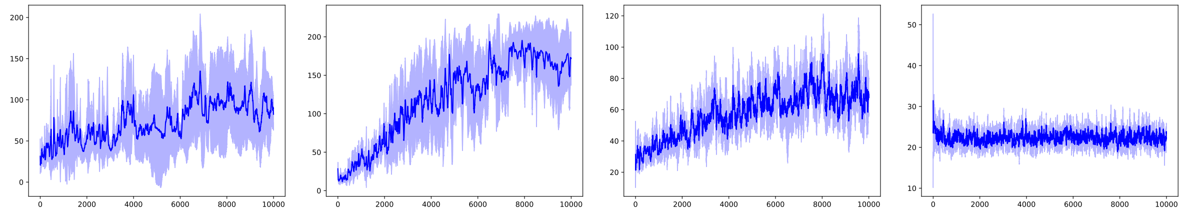
(a) **Reinforce(hard)** 27min 43s
(b) **Act-Cri(hard)** 23min 37s
(c) **Reinforce(hard)** 36min 25s
(d) **Act-Cri(hard)** 1h 16min 11s
(e) **Reinforce(soft)** 1h 8min 18s
(f) **Act-Cri(soft)** 23min 28s
(g) **Reinforce(soft)** 1h 30min 41s
(h) **Act-Cri(soft)** 1h 19min 25s
(i) **Reinforce(pyro)** 4h 47min 38s
(j) **Act-Cri(pyro)** 28min 51s
(k) **Reinforce(pyro)** 6h 31min 20s
(l) **Act-Cri(pyro)** 1h 29min 33s

Figure 5: Experimental results on CartPole-v0 (left two columns) and CartPole-v1 (right two columns).`model_softmaxQ` was used in **Actor-Critic(pyro)**.

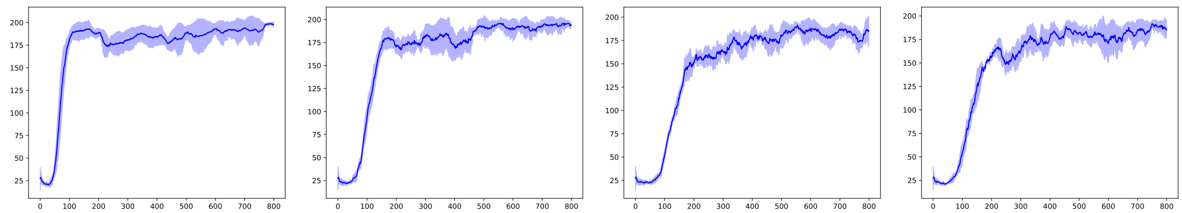


(a) **Act-Cri(pyro)** `model_unif`
(b) **Act-Cri(pyro)** `model_softmaxQ`
(c) **Act-Cri(pyro)** `model_unif`
(d) **Act-Cri(pyro)** `model_softmaxQ`

Figure 6: Experimental results on CartPole-v0 (Fig. 6a, Fig. 6b) and CartPole-v1 (Fig. 6c, Fig. 6d).



(a) **Reinforce(pyro)** $\alpha = 0.1$ (b) **Reinforce(pyro)** $\alpha = 1$ (c) **Reinforce(pyro)** $\alpha = 10$ (d) **Reinforce(soft)** $\alpha = 10$



(e) **Act-Cri(pyro)** $\alpha = 0.1$ (f) **Act-Cri(pyro)** $\alpha = 1$ (g) **Act-Cri(pyro)** $\alpha = 10$ (h) **Act-Cri(soft)** $\alpha = 10$

Figure 7: Experimental results on CartPole-v0 with different temperature α.

5. Conclusion & Discussion

RQ2: Whether the performance of `pyro.infer.SVI` is sufficient enough to support RL tasks? Yes, the performance of **Reinforce(pyro)** and **Actor-Critic(pyro)**, but they are expensive to perform–the time consumption is roughly 1 5 times than **Reinforce(soft)** and **Actor-Critic(soft)**, respectively. But we use `pyro.infer.SVI` as a black-box tool, this problem might be able to solve by opening the black box.

PL perspective: constructive v.s. existential random variables. There are two types of randomness / random variables (RVs): the constrictive one (randomness in **Pyro**, i.e. `pyro.sample`), and the existential one (i.e. randoms outside **Pyro**, e.g. from gym `env.step()`, or from robot play around in the real world). For example, the dynamics is RL is a kind of existential randomness: we only assume that there *exists* a transition distribution $p(s_{t+1}|s_t, a)t$ for the dynamics, we don't assume a prior or any information about it. And such existential RVs should be transparent to the **Pyro**'s SVI engine.

It seems that the RL (**Reinforce** and **Actor-Critic**) and MDP solving (policy iteration and value iteration) could be analog to the constructivism-existentialism dichotomy. The gap between RL and MDP solving lies in whether all random variables (RVs) are constructive. Curry-Howard correspondence claims that writing a program is equivalent to writing a proof term in a constructive logic system. We do not have existential RVs as primitives in PPLs since we do constructivism mathematics when dealing with programs. All RVs in PPLs are constructive.

Suppose we have a constructive RV that stands for the environment. In this case, we always have the implementation or assumptions we need about the environment to do optimal control. Thus we are doing model-based RL or solving MDPs. In contrast, model-free RL does not apply such strong assumptions. RL aims to do optimal control without assuming knowledge of the environment; thus, it must estimate via interaction. It is reasonable to not assume an implementation or a prior distribution for the dynamics since those RVs themself are even just an empirical approximation of the real world, without analytic ground truth.

In conclusion, MDP solving via dynamic programming corresponds to exact inference on constructive models in PPL. While we have to introduce existential RVs when performing model-free RL. More specifically, samples of existential RVs should be drawn from outside of **Pyro** (e.g., from **NumPy**, **PyTorch**, Gym rather than from **Pyro**). In such cases, these existential RVs would be transparent to the **pyro** and it's SVI engine.

Bayesian RL, distributional RL, and inverse RL. Although it would be a bit beyond the workload and the scope of this project, it would also be interesting to explore the theoretical connection between Bayesian inference in PPL and Bayesian RL (Duff, 2002), distributional RL (Bellemare et al., 2017), as well as inverse RL (Bingham et al., 2019).

References

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 2017. URL http://proceedings.mlr.press/v70/bellemare17a.html.

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019. URL http://jmlr.org/papers/v20/18-403.html.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Michael O'Gordon Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. University of Massachusetts Amherst, 2002.

Matthew Fellows, Anuj Mahajan, Tim G. J. Rudner, and Shimon Whiteson. VIREL: A variational inference framework for reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7120–7134, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/582967e09f1b30ca2539968da0a174fa-Abstract.html.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1352–1361. PMLR, 2017. URL http://proceedings.mlr.press/v70/haarnoja17a.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *CoRR*, abs/1805.00909, 2018. URL http://arxiv.org/abs/1805.00909.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html`.

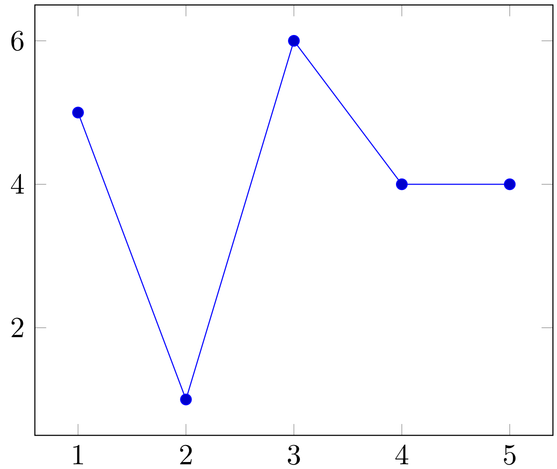
Adrian Sampson. Probabilistic programming, 2013. URL `http://adriansampson.net/doc/ppl.html`.

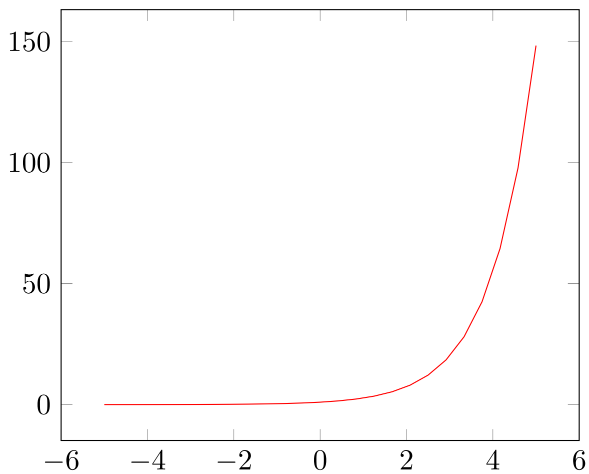
Wenjie Shi, Shiji Song, and Cheng Wu. Soft policy gradient method for maximum entropy deep reinforcement learning. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 3425–3431. ijcai.org, 2019. doi: 10.24963/ijcai.2019/475. URL `https://doi.org/10.24963/ijcai.2019/475`.

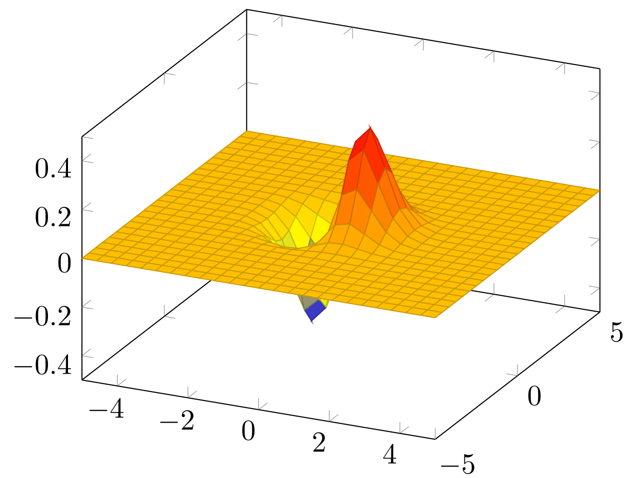
Philip S. Thomas and Emma Brunskill. Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines. *CoRR*, abs/1706.06643, 2017. URL `http://arxiv.org/abs/1706.06643`.

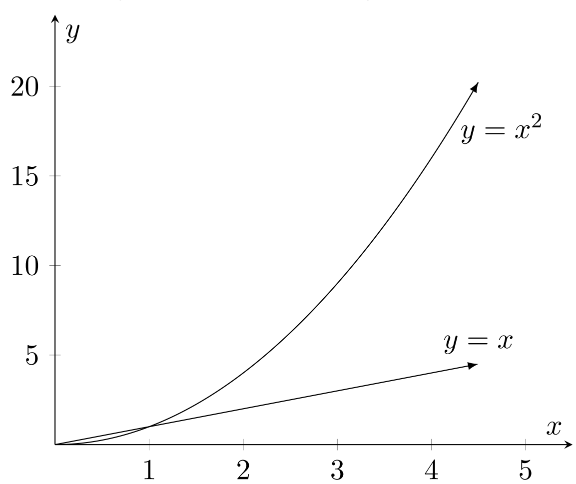
David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *CoRR*, abs/1301.1299, 2013. URL `http://arxiv.org/abs/1301.1299`.

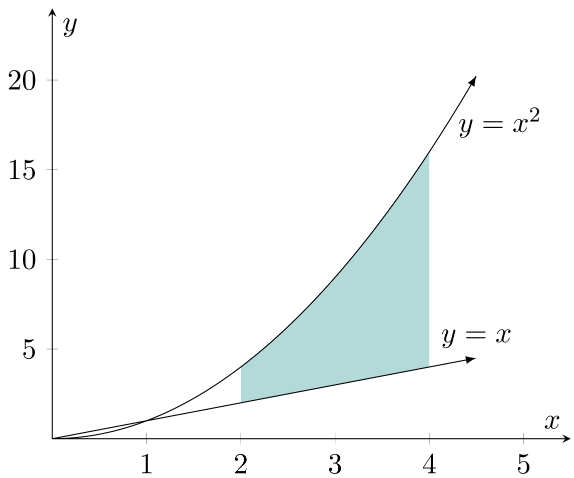
Appendix


6
4
2
1
2
3
4
5


150
100
50
0
−6
−4
−2
0
2
4
6


0.4
0.2
0
−0.2
−0.4
−4
−2
0
2
4
−5
0
5


y
20
15
10
5
y = x²
y = x
x
1
2
3
4
5



y
20
15
10
5
$y = x^2$
$y = x$
x
1
2
3
4
5