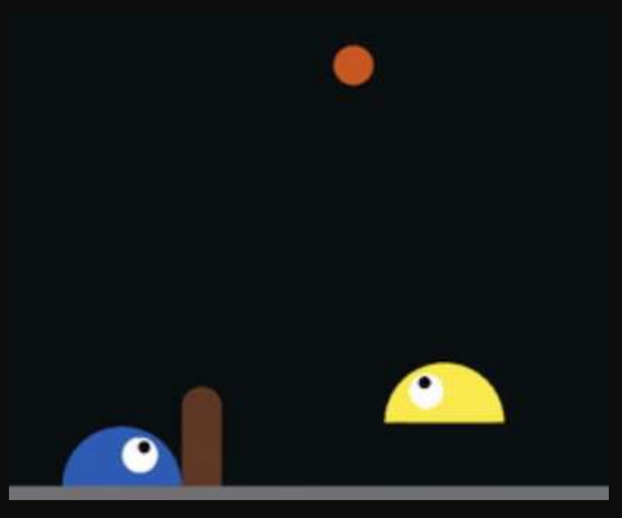
CS3244 - 08

You Play Ball, I Play Ball

Bayesian Multi-Agent Reinforcement Learning for Slime Volleyball

New Jun Jie, Stephen Tan Hin Khai, Jun Hup Lim, Aakanksha Rai, Hema Chandrasekaran, Raivat Shah
{ e0389098, e0389094, e0325362, e0313710, e0273869, e0315854}@u.nus.edu



Abstract

In Slime Volleyball[1], a two-player competitive game, we investigate how modelling uncertainty improves AI players' learning in 3 ways: against an expert, against itself and against each other, in the domain of multi-agent reinforcement learning (MARL). We show that by modelling uncertainty, Bayesian methods improve MARL training in 4 ways: performance, training stability, uncertainty and generalisability, and through experiments using TensorFlow Probability and Stable Baselines, we present interesting differences in agent behaviour. We contribute 3 code functionalities: Bayesian methods using Flipout integrated into Stable Baselines, a multi-agent versioned learning framework for Stable Baselines and uncertainty visualisation using agent clones for Slime Volleyball Gym.

Background

Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) addresses sequential decision making problems involving more than one agent, where each agent optimizes its own value function

$$V^{i}_{\pi^{i},\pi^{-i}} = E\left[\sum \gamma^{t} R^{i}(s_t, a_t, s_{t+1}) \middle| a^{i}_{t} \sim \pi(.|s_t), s_0 = s\right] \text{[2].}$$

The competitive setting is modelled as a zero-sum Markov game, where $\sum_{i\in N} R^{i}(s,a,s') = 0$ for any (s,a,s') where N is the number of agents.

Proximal Policy Optimization

Proximal policy optimization (PPO) is a policy gradient method, which adjusts weights to optimise the policy of an agent for the best reward at each state with

$$\nabla_{\theta} V^{\Pi}(s) = \sum_{a\in A}[V_{\theta}\pi_{\theta}(a|s)Q^{\pi}(s,a) + \pi_{\theta}(a|s)\sum_{s'} P(s'|s,a)\nabla_{\theta}\mathrm{V}^{\pi}(s')]\text{[3],}$$

and is effective for benchmarking results. PPO adopts the cost function:

$$J^{PPO}(\theta) = E[r(\theta)A_{\theta_{old}}(s,a),$$

where $r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)}$ is a probability ratio of new to old policies.

Flipout for Bayesian Neural Networks

Bayesian neural networks (BNN), in contrast to deep neural networks (DNN), determines neural network weights using maximum likelihood estimation to compute the maximum a posteriori estimate of weights using variational inference by minimizing the Kullback-Leibler divergence between two probability distributions, deriving the evidence lower bound:

$$F(D|\theta) \approx \frac{1}{N}\sum_{i\to N}[\log q(w^{i}|\theta) - \log p(w^{i}) - \log p(D|w^{i})],$$

serving a regularization effect and reducing tendency to overfit. Bayesian neural networks can be trained with variational inference by perturbing the weights. Flipout implements the Bayesian neural network by applying a base perturbation $\widehat{\Delta W}$ and multiplying it by a sign matrix $\Delta W_n = \widehat{\Delta W} \circ r_n s_n^{T}$[4].

Methodology

Experiment Formulation

To evaluate how Bayesian methods influence reinforcement learning training, we run the Expert experiment, where the PPO agent trains against a non-adaptive pre-trained expert baseline. The Expert experiment serves as a comparable baseline to other models trained on the Slime Volleyball gym environment. To evaluate how Bayesian methods influence multi-agent training, we run the Self and Multi experiments, where the agent trains by self-training with the previous version of itself in the former and against the previous version of its adaptive opponent in the latter. The Self experiment provides an opponent-agnostic environment so that agents can be better evaluated for generalisability. The Multi experiment serve to understand how Bayesian methods influence multi-agent reinforcement learning.

Expert, Self and Multi Experiments

In the Expert experiment, the agent trains against an expert for 5M timesteps across 10 trials on different random seeds. In the Self experiment, the agent trains against the previous version of itself (e.g. DNN-v1 vs DNN-v0) for 50M timesteps, updating its version when the current agent scores ≥ 0.5 against its previous version. In the Multi experiment, the agent trains against the previous version of its opponent (e.g. DNN-v1 vs BNN-v0 and BNN-v1 vs DNN-v0 concurrently) for 50M timesteps, differing from the Self experiment as versions are updated every 100K timesteps.

Results

PPO-BNN is superior to PPO-DNN in performance.

Performance is quantified by the agent's score against the expert baseline. In the Self experiment, performance of PPO-BNN=0.38 > PPO-DNN=0.32. In the Expert and Multi experiments, PPO-BNN and PPO-DNN scored similarly (within 0.03 difference).

PPO-BNN is superior to PPO-DNN in training stability.

Training stability is defined by the variance in performance in the last 100K training timesteps. Performance variance at the end of training of PPO-BNN=2.862 < PPO-DNN=3.996 in the Expert experiment, and PPO-BNN=0.728 < PPO-DNN=0.848 in the Self experiment. Performance variance in the Multi experiment is identical as expected.

PPO-BNN is superior to PPO-DNN in uncertainty.

Uncertainty is quantified by the entropy across action probabilities, $H(X) = -\sum P(x_i)\log_2 P(x_i)$ where $n = |A| = 3$, given observations at uncertain regions of the state space. We define uncertain regions as "speedy balls" when magnitude of ball velocity exceeds the 99th percentile and "bouncy balls" when the ball is near the net ($|x| < 0.5, y < 1$) as the angle of the ball bouncing off the net can be difficult to expect.

	PPO-DNN (Expert)	PPO-BNN (Expert)	PPO-DNN (Self)	PPO-BNN (Self)
Speedy Balls	0.623	**0.446**	0.565	**0.434**
Bouncy Balls	0.774	**0.417**	0.623	**0.446**

Across Expert and Self experiments, PPO-BNN greatly outperforms PPO-DNN with lower uncertainty.

PPO-BNN is superior to PPO-DNN in generalisability.

Generalisability is compared between two agents by evaluating against an opponent not seen during training. The range of scores possible is [-5, 5] as 5 balls are played per game.

	PPO-DNN (Expert)	PPO-BNN (Expert)
Expert Baseline	0.02 ± 0.99	0.05 ± 0.81
PPO-DNN (Expert)	-	0.14 ± 0.76
PPO-BNN (Expert)	-0.14 ± 0.76	-

	PPO-DNN (Self)	PPO-BNN (Self)
Expert Baseline	0.32 ± 0.69	0.38 ± 0.90
PPO-DNN (Self)	-	0.00 ± 0.58
PPO-BNN (Self)	0.00 ± 0.58	-
PPO-DNN (Expert)	3.47 ± 1.33	4.71 ± 0.67
PPO-BNN (Expert)	4.50 ± 0.87	4.65 ± 0.74

	PPO-DNN (Multi)	PPO-BNN (Multi)
Expert Baseline	0.07 ± 0.94	0.04 ± 0.93
PPO-DNN (Self)	1.19 ± 1.32	2.45 ± 1.56
PPO-BNN (Self)	2.03 ± 1.50	3.96 ± 1.30
PPO-DNN (Expert)	-0.34 ± 0.74	-0.23 ± 0.81
PPO-BNN (Expert)	-0.28 ± 0.85	-0.50 ± 0.96

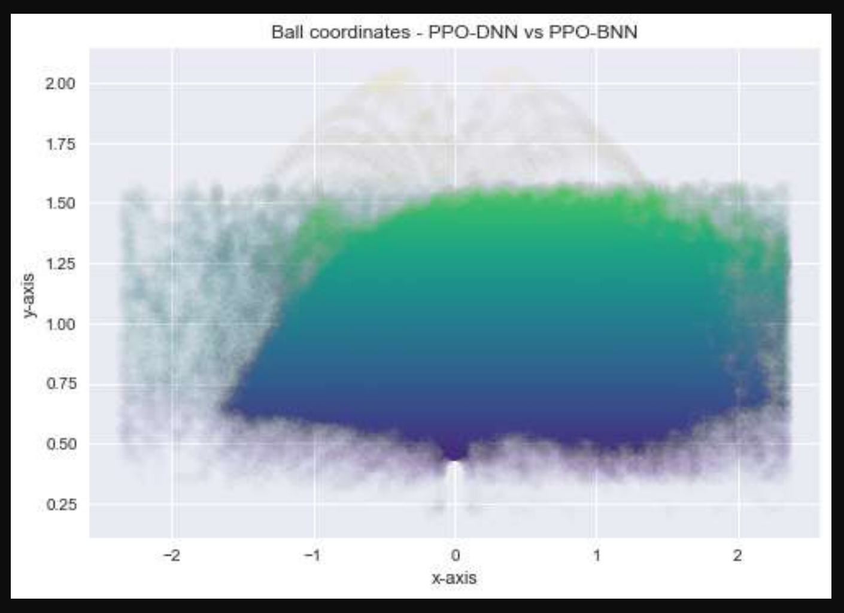
Across all experiments, PPO-BNN greatly outperforms PPO-DNN in games against opponents not seen during training.

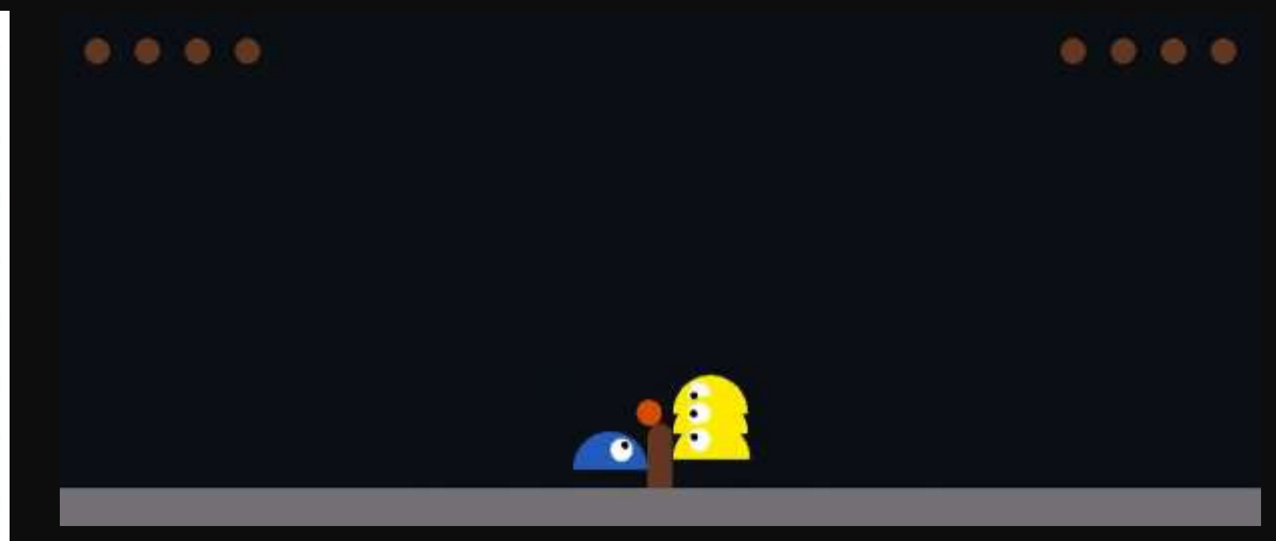
PPO-BNN is inferior to PPO-DNN in sample efficiency.

Sample efficiency is determined by the timesteps required to reach the maximum episode length. In the Self experiment, timesteps required to reach the maximum episode length for PPO-BNN=30M > PPO-DNN=20M.

Discussion

Agent Behaviour of PPO-BNN (Self) vs PPO-DNN (Self)





The left visualisation shows the ball's xy-coordinates in 100 games between PPO-DNN (right court) and PPO-BNN (left court) trained by self-training. PPO-BNN frequently bounces the ball at a sharp angle, returning the ball to PPO-DNN in one bounce. As a result, PPO-DNN usually requires more than one bounce to return the ball. The right visualisation is a clone visualisation of agent uncertainty. At the "bouncy ball" region, the variance of agent actions is higher for PPO-DNN (right) than PPO-BNN (left), indicating lower uncertainty in PPO-BNN agents.

Generalisability from Bayesian Methods

The introduction of the Flipout layer as a feature extractor in the implementation of the PPO-BNN agent that significantly improved generalisability against previously unseen opponents can be explained by the reduction of variance in the estimation of gradients in the PPO algorithm. By applying a Bayesian treatment to existing reinforcement learning algorithms, agents can hope to improve generalisability in uncertain tasks.

Experiment Limitations

Hyperparameter optimisation, while attempted, is decidedly infeasible due to the highly stochastic nature of reinforcement learning, exacerbated by time and compute limitations. Multi-agent training is implemented in batched versioning of opponents instead of live after each game, due to the highly restrictive single-agent framework of Stable Baselines. While the Expert experiments are performed across 10 trials, despite our best intentions, the Self and Multi experiments are performed only once, to the undesirable but likely possibility of variance across trials.

Future Work

We hope to extend our work in the future by implementing various agent exploration strategies such as neuroevolution and partial observability in the environment by withdrawing frames, investigating the influence of Bayesian methods on multi-agent reinforcement learning training and agent behaviour in various contexts.

Conclusion

We have shown that by modelling uncertainty, Bayesian methods improve MARL training in 4 ways: performance, training stability, uncertainty and generalisability. We contribute 3 code functionalities: Bayesian methods using Flipout integrated into Stable Baselines, multi-agent versioned learning framework for Stable Baselines (previously with only single-agent support) and uncertainty visualisation using agent clones for Slime Volleyball Gym. From our work, our team is excited to extend the applications of Bayesian methods into multi-agent reinforcement learning, improving generalisability for AI agents of the future to excel in spite of the complexities of the real world.

References

[1] David Ha (2020) Slime Volleyball Gym Environment - [Online]. Available at: https://github.com/hardmaru/slimevolleygym (Accessed 1 Oct 2020)

[2] Zhang, K., Yang, Z., & Başar, T. (2019). Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*.

[3] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[4] Wen, Y., Vicol, P., Ba, J., Tran, D., & Grosse, R. (2018). Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*.