Deep Amortized Inference for Probabilistic Programs using Adversarial Compilation

Mahdi Azarafrooz m.azarafrooz@gmail.com

We propose an amortized inference strategy for probabilistic programs, one that learns from past inferences to speed up the future inferences. Our proposed inference strategy is to train neural guidance programs via a minimax game, with the probabilistic program as a *correlation device*. From a game-theoretical vantage point, the role of a correlation device is to enforce better outcomes by sharing information between players. The shared information, in our case, is the execution trace, which gets used for computation of payoffs in the minimax game.

1 Probabilistic Programming Setup

One approach to Probabilistic programming languages (PPL) is through "many world" execution trace tree. Upon execution, a probabilistic program encounters a series elementary random primitives (ERP). From the many world view, different execution traces will be taken depending on the return value of each call to these ERP. Let's consider a generative model $p(\mathbf{x}, \mathbf{y})$ with latent variables \mathbf{x} and observation data \mathbf{y} . Prior distribution $p(\mathbf{x})$ can be interpreted as distribution of the execution traces of unconditioned programs $\mathcal{F}(.)$. Various execution of the program yield in various samples from the prior. Similarly, the posterior distribution $p(\mathbf{x}|\mathbf{y})$ can be considered as $\mathcal{F}(\mathbf{y})$. By tracking down the execution trace, $p(\mathbf{x})$ can be decomposed as a product of conditionals $p(x_i|\mathbf{x}_{\leq i})$, one for each ERP on the execution trace to x_i . $\mathbf{x}_{\leq i}$ indicates that it could potentially be depending on any or all previous ERPs. Therefore PPL expresses the generative model as:

$$p(\mathbf{y}, \mathbf{x}) = p(\mathbf{y}|\mathbf{x}) \prod_{i} p(x_i|\mathbf{x}_{< i})$$
(1)

2 Amortized Inference using Neural Guidance Programs

Any arbitrary programming languages can be turned into probabilistic programming languages with various Markov Chain (MC) inference strategies [1]. However, MC inference strategies are computationally expensive. More importantly, with the arrival of a new observation \mathbf{y}' , the PPL's inference engine has to recompute $p(\mathbf{x}|\mathbf{y}')$ from scratch. Amortized inference addresses this problem by learning from past inferences to speed up the future inferences. The amortized inference strategy in [2] is to learn a neural guidance program q upfront so that at inference time, sampling from q is both fast and accurate. It is an approximation of $p(\mathbf{x}|\mathbf{y})$ using side neural computations:

$$q(\mathbf{x}|\mathbf{y};\phi) = \prod_{i} q(x_i; D_i(\mathbf{y}, \mathbf{x}_{< i}; \phi))$$
(2)

where D_i is a neural network. Their proposed guidance program q extends the mean field family but is still limited by its *fully-factored representational* format. In our proposed setup, the guidance program $G(\theta)$ is a neural network itself that gets trained by playing a minimax game against $D(\phi)$. Moreover, the proposed amortized inference in [2] affects the learning dynamic of the guidance program via the generated data, in a passive way. In our proposed approach, PPL's inference engine can interact with the guidance program $G(\theta)$ in an interactive way, as is explained in the next section. Aside from the benefits of amortized inference, it can lead to more efficient MC inferences.

3 Amortized inference using Deep Adversarial Compilation

Generative adversarial networks (GAN) [3] trade complexities of sampling algorithms with the complexities of finding Nash equilibrium in minimax games. The players of the game are deep neural networks known as generator ¹ and adversary. The adversary tries to maximize its payoff defined as negative Jensen-Shannon (JD) distance, whereas generator objects to it. The strategies are parameters of these neural networks that get updated with stochastic gradient descent, at every stage of the game. Similarly, we cast the learning problem of the guidance program as a game theoretical problem. As a result, instead of solving variational

¹Not to be confused with the generative model of PPL

objective using Eq. 2, as in done in [2], we formulate a minimax search for guidance program. Guidance program $G(\mathbf{y}, \mathbf{x}_{\langle i}; \theta)$ plays the role of the generator that generates x_i given the execution trace $\mathbf{x}_{\langle i}$. The neural architecture of G can be selected from Recursive Neural Networks (RNN) variants such as LSTM [6], GRU [7], etc. The adversary D then evaluates the execution trace choices made by G. D can be a simple feedforward neural network.

In our proposed game theoretical framework, the probabilistic program engine recommends an execution trace $\mathbf{x}_{\langle i}$ that gets used by G, D to compute their payoff. As the result of probabilistic program mediation, the achieved equilibrium turns out to be a *correlated* equilibrium. This is unlike other GAN formulations, where the harder problem of reaching Nash equilibrium is of interest.

The adversary 's goal is to maximize its payoff:

Generator's payoff is defined as expected end payoff formulated in Eq. 3, similar to [5].

$$V_{G_{\theta}} = -\sum_{i} G(\mathbf{y}, \mathbf{x}_{\langle i \rangle}; \theta) Q_{D_{\phi}}^{\mathcal{F}^{G_{\theta}}}(s = \mathbf{x}_{\langle i \rangle}, a = x_{i}) \quad (3)$$

where $Q_{D_{\phi}}^{\mathcal{F}^{G_{\theta}}}(\mathbf{x}_{\langle i}, a)$ is the reward function for taking compilation action a, and then following the inferred execution trace $\mathcal{F}(\mathbf{y})$ with G_{θ} as its proposal distribution:

$$Q_{D_{\phi}}^{\mathcal{F}^{G_{\theta}}}(s = \mathbf{x}_{\langle i, a} = x_{i}) =$$

$$\begin{cases} D(\mathbf{y}, \mathbf{x}; \phi), \mathbf{x} \sim \mathcal{F}^{G_{\theta}} & if \ i < N \\ D(\mathbf{y}, \mathbf{x}; \phi) & else \end{cases}$$
(4)

In other words, the generator G takes into account the future outcome of branching off to a new execution trace. However, the adversary D only provides a payoff value for the finished execution trace. This introduces subjectivity ² to the game. We refer to this inference strategy as adversarial compilation since it transforms the form of a probabilistic program into an adversarial neural network specification language.

Proposition 1 [4] There may exist mutually advantageous equilibrium points for a 2-person minimax game, if we permit both correlation and subjectivity.

As a result of proposition 1, the amortized inference can generate samples that come even closer than minimum JD to the inferred execution trace by probabilistic program.

Theoretical comparison between amortized inference strategy of [2] and ours is not easy. This is because [2] minimizes the KL distance, whereas we formulate a minimax search for minimum JD distance. However, we would like to reiterate that, unlike [2], our proposed amortized inference strategy is not limited by fullyfactored representational formats.

Future works include extending the game theoretical framework to the case, where guidance programs not only interact with each other through a minimax game, but also can affect the random choices made by the inference engine of PPL. This could improve the efficiency of MC strategies.

References

[1] Wingate, D. & Stuhlmüller, A and Goodman, N.D. . (2011) Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AIS-TATS

[2] D. Ritchie, P. Horsfall, Noah D Goodman (2016) Deep Amortized Inference for Probabilistic Programs *arXiv preprint arXiv:1610.05735*

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. (2017) Generative Adversarial Nets Advances in Neural Information Processing Systems 27

[4] Robert J. AUMANN (1974) subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*.

[5] L. Yu, W. Zhang, J. Wang, Y. Yu (2017) SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

[6] S. Hochreiter and J. Schmidhuber. (1997). Long Short-Term Memory. *Neural Comput. 9*.

[7] K. Cho, B. v. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. (2014) Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation.

²Subjectivity originates from either disagreement of players over probabilities of outcomes or the recommendation of the correlation device.