Learning to Sample Using Stein Discrepancy

Dilin Wang  Yihao Feng  Qiang Liu
Department of Computer Science
Dartmouth College
Hanover, NH 03755
{dilin.wang.gr, yihao.feng.gr, qiang.liu}@dartmouth.edu

Abstract

We propose a simple algorithm to train stochastic neural networks to draw samples from given target distributions for probabilistic inference. Our method is based on iteratively adjusting the neural network parameters so that the output changes along a Stein variational gradient [1] that maximally decreases the KL divergence with the target distribution. Our method works for any target distribution specified by their unnormalized density function, and can train any black-box architectures that are differentiable in terms of the parameters we want to adapt. By allowing to “learn to draw samples”, our method opens a host of applications. We present two examples in this paper: 1) we propose an amortized MLE method for training deep energy model, where a neural sampler is adaptively trained to approximate the likelihood function. Our method mimics an adversarial game between the deep energy model and the neural sampler, and obtains realistic-looking images competitive with the state-of-the-art results. 2) by treating stochastic gradient Langevin dynamics as a black-box sampler, we train it to automatically adjust its learning rate to maximize its convergence speed, and get better performances than the hand-designed learning rate schemes.

1 Introduction

Modern machine learning increasingly relies on highly complex probabilistic models to reason about uncertainty. A key computational challenge is to develop efficient inference techniques to approximate, or draw samples from complex distributions. Currently, most inference methods, including MCMC and variational inference, are hand-designed by researchers or domain experts. This makes it difficult to fully optimize the choice of different methods and their parameters, and exploit the structures in the problems of interest in an automatic way. The hand-designed algorithm can also be inefficient when it requires to make fast inference repeatedly on a large number of different distributions with similar structures. This happens, for example, when we need to reason about a number of observed datasets in settings like online learning, or need fast inference as inner loops for other algorithms such as maximum likelihood training. Therefore, it is highly desirable to develop more intelligent probabilistic inference systems that can adaptively improve its own performance to fully optimize computational efficiency, and generalize to new tasks with similar structures. Specifically, we study the following problem:

Problem 1. Given a distribution with density \( p(x) \) specified up to the normalization constant, and a function \( f_\eta(\xi) \) with parameter \( \eta \) and random input \( \xi \), for which we only have access to draws of the random input \( \xi \) (without knowing its true distribution \( q_\xi \)), and the output values of \( f_\eta(\xi) \) and its derivative \( \partial_\eta f_\eta(\xi) \) given \( \eta \) and \( \xi \). We want to find an optimal parameter \( \eta \) so that the density of the random output variable \( x = f_\eta(\xi) \) with \( \xi \sim q_\xi \) matches closely with the target density \( p(x) \).

Because we have no assumption on the structure of \( f_\eta \) and the distribution of random input, we can not directly calculate actual distribution of the output random variable \( x = f_\eta(\xi) \); this makes
We approach Problem 1 by iteratively adjusting the network parameter \( \tilde{\eta} \) where \( \epsilon \) is a small step size, and \( \Delta x_i \) is an optimal perturbation direction chosen to maximize the decrease in the KL divergence between the distribution of the particles and the target distribution. The derivation in Liu and Wang [1] shows that \( \Delta x_i \) can be chosen to be

\[
\Delta x_i = \mathbb{E}_{x \sim \{x_i\}_{i=1}^n} [\nabla \log p(x, x_i) + \nabla_x k(x, x_i)],
\]  

(1)

where \( \mathbb{E}_{x \sim \{x_i\}_{i=1}^n} \) denotes the empirical averaging on the current particles \( \{x_i\}_{i=1}^n \). The two terms in \( \Delta x_i \) play two different roles: the term with the gradient \( \nabla_x \log p(x) \) drives the particles towards the high probability regions of \( p(x) \), while the term with \( \nabla_x k(x, x_i) \) serves as a repulsive force to make the network outputs indexed by parameter \( \eta \) be more diverse. The two terms

\[
\sum_i f(x_i)/n \approx \mathbb{E}_p f \quad \text{for general test functions } f.
\]

Each iteration of SVGD updates the current particles \( \{x_i\}_{i=1}^n \) with a gradient-like update

\[
x_i' = x_i + \epsilon \Delta x_i,
\]

where \( \epsilon \) is a small step size, and \( \Delta x_i \) is an optimal perturbation direction chosen to maximize the decrease in the KL divergence between the distribution of the particles and the target distribution.
We propose to solve Problem 1 by “amortizing” SVGD to make the output of network \( f_\eta(\xi) \) mimic the SVGD dynamics. This is done by iteratively adjusting the network parameter \( \eta \) such that the network outputs \( x_i = f_\eta(\xi_i) \), \( i = 1, \ldots, n \), change along with the optimal direction \( \Delta x_i \) given by SVGD. To be specific, we should update \( \eta \) via

\[
\eta \leftarrow \arg \min_\eta \sum_{i=1}^n ||f_\eta(\xi_i) - x_i - \epsilon \Delta x_i||_2^2.
\]

If we approximately solve this optimization with a single step of gradient update, we get a simpler update of \( \eta \):

\[
\eta \leftarrow \eta + \epsilon \sum_i \partial_{\eta} f_\eta(\xi_i) \Delta x_i, \tag{2}
\]

which can be intuitively interpreted as a form of chain rule that back-propagates the SVGD gradient to the network parameter \( \eta \). In fact, when we have only one particle, (2) reduces to the standard gradient ascent for maximizing \( \log p(x) \) (i.e., maximum a posteriori (MAP)).

We propose to solve Problem 1 by “amortizing” SVGD to make the output of network \( f_\eta(\xi) \) mimic the SVGD dynamics. This is done by iteratively adjusting the network parameter \( \eta \) such that the network outputs \( x_i = f_\eta(\xi_i) \), \( i = 1, \ldots, n \), change along with the optimal direction \( \Delta x_i \) given by SVGD. To be specific, we should update \( \eta \) via

\[
\eta \leftarrow \arg \min_\eta \sum_{i=1}^n ||f_\eta(\xi_i) - x_i - \epsilon \Delta x_i||_2^2.
\]

If we approximately solve this optimization with a single step of gradient update, we get a simpler update of \( \eta \):

\[
\eta \leftarrow \eta + \epsilon \sum_i \partial_{\eta} f_\eta(\xi_i) \Delta x_i, \tag{2}
\]

which can be intuitively interpreted as a form of chain rule that back-propagates the SVGD gradient to the network parameter \( \eta \). In fact, when we have only one particle, (2) reduces to the standard gradient ascent for maximizing \( \log p(x) \) (i.e., maximum a posteriori (MAP)).

We propose to solve Problem 1 by “amortizing” SVGD to make the output of network \( f_\eta(\xi) \) mimic the SVGD dynamics. This is done by iteratively adjusting the network parameter \( \eta \) such that the network outputs \( x_i = f_\eta(\xi_i) \), \( i = 1, \ldots, n \), change along with the optimal direction \( \Delta x_i \) given by SVGD. To be specific, we should update \( \eta \) via

\[
\eta \leftarrow \arg \min_\eta \sum_{i=1}^n ||f_\eta(\xi_i) - x_i - \epsilon \Delta x_i||_2^2.
\]

If we approximately solve this optimization with a single step of gradient update, we get a simpler update of \( \eta \):

\[
\eta \leftarrow \eta + \epsilon \sum_i \partial_{\eta} f_\eta(\xi_i) \Delta x_i, \tag{2}
\]

which can be intuitively interpreted as a form of chain rule that back-propagates the SVGD gradient to the network parameter \( \eta \). In fact, when we have only one particle, (2) reduces to the standard gradient ascent for maximizing \( \log p(x) \) (i.e., maximum a posteriori (MAP)).

We propose to solve Problem 1 by “amortizing” SVGD to make the output of network \( f_\eta(\xi) \) mimic the SVGD dynamics. This is done by iteratively adjusting the network parameter \( \eta \) such that the network outputs \( x_i = f_\eta(\xi_i) \), \( i = 1, \ldots, n \), change along with the optimal direction \( \Delta x_i \) given by SVGD. To be specific, we should update \( \eta \) via

\[
\eta \leftarrow \arg \min_\eta \sum_{i=1}^n ||f_\eta(\xi_i) - x_i - \epsilon \Delta x_i||_2^2.
\]

If we approximately solve this optimization with a single step of gradient update, we get a simpler update of \( \eta \):

\[
\eta \leftarrow \eta + \epsilon \sum_i \partial_{\eta} f_\eta(\xi_i) \Delta x_i, \tag{2}
\]

which can be intuitively interpreted as a form of chain rule that back-propagates the SVGD gradient to the network parameter \( \eta \). In fact, when we have only one particle, (2) reduces to the standard gradient ascent for maximizing \( \log p(x) \) (i.e., maximum a posteriori (MAP)).
Algorithm 2 Amortized MLE as Generative Adversarial Learning

Goal: MLE training for energy model \( p(x|\theta) = \exp(-\phi(x, \theta) - \Phi(\theta)). \)

Initialize \( \eta \) and \( \theta \).

for iteration \( t \) do

Updating \( \eta \): Draw \( \xi_i \sim q_\xi \), \( x_i = f_\eta(\xi_i) \); update \( \eta \) using (2) with \( p(x) = p(x|\theta) \). Repeat several times when needed.

Updating \( \theta \): Draw a mini-batch of observed data \( \{x_{i,obs}\} \), and simulated data \( x_i = f_\eta(\xi_i) \), update \( \theta \) by

\[
\theta \leftarrow \theta - \hat{E}_{obs}[\nabla_\theta \phi(x, \theta)] + \hat{E}_q[\nabla_\theta \phi(x, \theta)].
\]

end for

This allows us to solve Problem [1] for wild variational inference by directly minimizing \( \eta \) with standard (stochastic) gradient descent.

However, (3) does not have the nice property of reducing to “learning to optimize” like (2). (3) is also less natural because (3) involves the Hessian matrix \( \nabla^2_x \log p(x) \). This makes it less convenient to implement (3), although it can be made easy with automatic differentiation tools. We also find in practice that (3) turns to be unstable sometimes, possibly because \( \mathbb{D}(q \parallel p) \) forms a weaker discrepancy measure than KL divergence. More studies are needed to understand and improve amortized KSD.

4 Applications

Our method allows us to design efficient approximate sampling methods adaptively and automatically, and enables a host of novel applications. In this paper, we exploit two particular examples: (1) amortized MLE for training deep generative models, and (2) automatic hyper-parameter tuning for Bayesian inference.

4.1 Amortized MLE for Generative Adversarial Training

Maximum likelihood estimator (MLE) provides a fundamental approach for learning probabilistic models from data, but can be computationally prohibitive on distributions for which drawing samples or computing likelihood is intractable due to the normalization constant. Traditional methods such as MCMC-MLE uses hand-designed methods (e.g., MCMC) to approximate the intractable term but do not work efficiently in practice. We propose to adaptively train a generative neural network to draw samples from the distribution, which not only provides computational advantage, and also allow us to generate realistic-looking images competitive with, or better than the state-of-the-art generative adversarial networks (GAN) [19, 20] (see Figure 2-6).

To be specific, denote by \( \{x_{i,obs}\} \) a set of observed data. We consider the maximum likelihood training of energy-based models of form

\[
p(x|\theta) \propto \frac{1}{Z(\theta)} \exp(-\phi(x, \theta)), \quad Z(\theta) = \int \exp(-\phi(x, \theta)) dx,
\]

where \( \phi(x; \theta) \) is an energy function for \( x \) indexed by parameter \( \theta \) and \( Z(\theta) \) is the normalization constant. The maximum likelihood estimator of \( \theta \) is based on maximizing the log likelihood function,

\[
L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \log p(x_{i,obs}|\theta),
\]

whose gradient is

\[
\nabla_\theta L(\theta) = -\hat{E}_{obs}[\partial_\theta \phi(x; \theta)] + \hat{E}_q[\partial_\theta \phi(x; \theta)],
\]

where \( \hat{E}_{obs}[] \) and \( \hat{E}_q[] \) denote the empirical averaging on the observed data \( \{x_{i,obs}\} \) and the expectation under model \( p(x|\theta) \). The key computational difficulty is to approximate the model averaging \( \hat{E}_q[] \). To address this problem, we use a generative neural network \( x = f_\eta(\xi) \) trained by Algorithm [1] to approximately sample from \( p(x|\theta) \),

\[
\nabla_\theta L(\theta) = -\hat{E}_{obs}[\partial_\theta \phi(x; \theta)] + \hat{E}_q[\partial_\theta \phi(x; \theta)],
\]
Figure 2: Results on CelebA. Upper: Images generated by DCGAN and our SteinGAN. Lower: images generated by SteinGAN when performing a random walk $\xi \leftarrow \xi + 0.01 \times \text{Uniform}([-1, 1])$ on the random input $\xi$.

where $\hat{E}_n$ denotes the empirical averaging on $\{x_i\}$ where $x_i = f_\eta(\xi_i), \{\xi_i\} \sim q_\xi$. As $\theta$ is updated by gradient ascent, $\eta$ is successively updated via Algorithm 1 to follow $p(x|\theta)$. See Algorithm 2.

We call our method SteinGAN, because it can be intuitively interpreted as an adversarial game between the generative network $f_\eta$ and the energy model $p(x|\theta)$ which serves as a discriminator: The MLE gradient update of $p_\eta$ effectively decreases the energy of the training data and increases the energy of the simulated data from $f_\eta$, while the SVGD update of $f_\eta$ decreases the energy of simulated data to fit better with $p(x|\theta)$. Meanwhile, our procedure is still a principled maximum likelihood procedure and can be more stable than the original GAN [19] that attends to find a Nash equilibrium. Compared with the traditional MCMC-MLE methods, we amortize the sampler as we train, which gives much faster speed and provides a high quality generator to generate realistic images simultaneously.

We tested our SteinGAN on four datasets, MNIST, CIFAR10, CelebA [21], and Large-scale Scene Understanding (LSUN) [22], on which we find our method tends to generate realistic-looking images competitive with DCGAN [20] (see Figure 2-Figure 4). In particular, we find we generate better images than DCGAN on CelebA (Figure 2), and our simulated CIFAR10 images achieves better testing classification accuracy when used as training data (see Figure 4). See Appendix A for more information. Our code is available at [https://github.com/DartML/SteinGAN](https://github.com/DartML/SteinGAN).

### 4.2 Hyper-parameter Optimization for Bayesian Inference

By treating the existing MCMC or variational methods as black-box procedures, we can apply our method to adaptively tune the hyper-parameters in these methods. This allows us to fully optimize the potential of existing Bayesian inference methods and also decreases the need of hyper-parameter tuning by human experts. As an example, we applied our method to adaptively learn the optimal learning rate for stochastic gradient Langevin dynamics (SGLD) [23], and find that significantly
outperforms hand-designed learning rates such as Adagrad [24] and RMSprop (See Figure 5). See Appendix B for more information.

5 Conclusion

We provide efficient algorithms for training neural samplers, together with a new SteinGAN method for generative adversarial training, and a hyperparameter optimization method for stochastic gradient Langevin dynamics. Future directions involve more applications and theoretical understandings for training neural samplers.

References


A Empirical Results of SteinGAN

In order to generate realistic-looking images, we define our energy model based on an autoencoder:

\[ p(x|\theta) \propto \exp(-||x - D(E(x))||), \]

(4)

where \( x \) denotes the image. This choice is motivated by Energy-based GAN [25] in which the autoencoder loss is used as a discriminator but without a probabilistic interpretation. We assume \( f_\eta(\xi) \) to be neural networks whose input \( \xi \) is a 100-dimensional random vector drawn by Uniform([-1, 1]). The positive definite kernel is defined by the RBF kernel on hidden layer of the autoencoder, that is,

\[ k(x, x') = \exp\left(-\frac{1}{h^2}||E(x) - E(x')||^2\right). \]

The kernel can act as a repulsive force to enforce diversity on the generated samples via the term \( \nabla_x k(x, x') \) in (1). This is similar to the heuristic “repelling regularizer” in [25], but derived in a principled way. We take the bandwidth to be \( h = 0.5 \times \text{med} \), where \( \text{med} \) is the median of distances between \( E(x) \) on the observed images \( \{x_{i,\text{obs}}\}_i \).

For MNIST and CIFAR-10, each image \( x \) also has a discrete label \( y \), and we train a joint model on \((x, y)\):

\[ p(x, y|\theta) \propto \exp\{-||x - D(E(x))|| - \max[m, \sigma(y, E(x))]\}, \]

(5)

where \( \sigma(\cdot, \cdot) \) is the cross entropy loss function of a fully connected output layer. In this case, the generative network will first draw a \( y \) randomly according to the empirical counts in the dataset, and pass it into a neural network together with a 100 dimensional random vector to generate image \( x \). This model allows us to generate images for each category.

We refer to Wang and Liu [26] for more details and discussions about SteinGAN.

B Empirical Results for Hyper-parameter Optimization of Bayesian Inference

We applied our method to adaptively learn the optimal learning rate for stochastic gradient Langevin dynamics (SGLD) [23]. Denote by \( D = \{z_j\}_{j=1}^N \) an observed dataset drawn i.i.d. by \( p(z|x) \), and \( x \) is a random parameter with prior \( p_0(x) \). The posterior distribution of \( x \) is

\[ p(x|D) \propto p_0(x) \prod_{j=1}^N p(z_j|x). \]

SGLD draw approximate sample from \( p(x|D) \) via iterative update of form

\[ x^{t+1} \leftarrow x^t + \eta^t \cdot [\log p_0(x^t) + \frac{N}{|M^t|} \sum_{j \in M^t} \nabla_x \log p(D_j|x^t)] + \sqrt{2\eta^t} \cdot \xi^t, \]

8
Figure 4: Results on CIFAR-10. “500 Duplicate” denotes 500 images randomly subsampled from the training set, each duplicated 100 times. Upper: images simulated by DCGAN and SteinGAN (based on joint model (5)) conditional on each category. Middle: inception scores for samples generated by various methods (all with 50,000 images) on inception models trained on ImageNet and CIFAR-10, respectively. Lower: testing accuracy on real testing set when using 50,000 simulated images to train ResNets for classification. SteinGAN achieves higher testing accuracy than DCGAN.

where $\xi^t$ is a standard Gaussian random vector of the same size as $x$, and $M^t$ is a random mini-batch selected at $t$-th iteration (we use a mini-batch size of 100), and $\eta^t$ denotes a (vector) stepsize at $t$-th iteration. Consider running SGLD for $T = 100$ iterations, we can treat $x^T$ as the output of a $T$-layer neural network parametrized the collection of stepsizes $\eta = \{\eta^t\}_{t=1}^T$, with $x_0$ and $\{M^t, \xi^t\}_{t=1}^T$ as the random inputs. This allows us to apply amortized SVGD or KSD to adaptively estimate the optimal stepsize $\eta$.

We test our method with Bayesian logistic regression on the Covertype dataset. To demonstrate that our estimated learning rate can work well on new datasets never seen by the algorithm. We partition the dataset into mini-datasets of size 50,000, and use 80% of them for training and 20% for testing. We adapt our amortized SVGD/KSD to train on the whole population of training mini-datasets by randomly selecting a mini-dataset at each iteration of Algorithm 1. Figure 5 reports the test accuracy when we apply the estimated step sizes on the 20% mini-datasets held for testing. We find that our method outperforms all the hand-designed learning rates, including Adagrad [24] and Rmsprop. We find that amortized KSD does not work as well as amortized SVGD, likely because KSD is a weaker discrepancy measure compared with KL divergence.

We refer to Liu and Feng [27] for more details and discussions about wild variational inference using Stein discrepancy.
Figure 5: (a) The testing accuracy of the first $T = 100$ iterations with learned and hand-designed step sizes. We find that our method outperforms all the hand-designed learning rates, as well as amortized KSD. (b)-(c) Examples of step sizes learned by amortized SVGD (for two different dimensions of $x$).

Figure 6: More images generated by SteinGAN on CelebA.