Joint control variate for faster black-box variational inference

Xi Wang **Tomas Geffner** **Justin Domke**
Manning College of Information and Computer Sciences,
University of Massachusetts Amherst
{xwang3,tgeffner,domke}@cs.umass.edu

Abstract

Black-box variational inference performance is sometimes hindered by the use of gradient estimators with high variance. This variance comes from two sources of randomness: Data subsampling and Monte Carlo sampling. While existing control variates only address Monte Carlo noise, and incremental gradient methods typically only address data subsampling, we propose a new "joint" control variate that *jointly* reduces variance from both sources of noise. This significantly reduces gradient variance, leading to faster optimization in several applications.

1 INTRODUCTION

Black-box variational inference (BBVI) (Hoffman et al., 2013; Titsias and Lázaro-Gredilla, 2014; Ranganath et al., 2014; Kucukelbir et al., 2017; Blei et al., 2017) is a popular alternative to Markov Chain Monte Carlo (MCMC) methods. The idea is to posit a variational family and optimize it to be close to the posterior, using only "black-box" evaluations of the target model (either the density or gradient). This is typically done by minimizing the KL-divergence using stochastic optimization methods with unbiased gradient estimates. Often, this allows the use of data subsampling, which greatly speeds-up optimization with large datasets.

The BBVI optimization problem is often called "doubly-stochastic" (Titsias and Lázaro-Gredilla, 2014; Salimbeni and Deisenroth, 2017), as the gradient estimation has two sources of randomness: Monte Carlo sampling from the variational distribution, and data subsampling from the full dataset. Because of the doubly-stochastic nature, one common challenge for BBVI is the variance of the gradient estimates: If this is high, it forces small stepsizes, leading to slow optimization convergence (Nemirovski et al., 2009; Bottou et al., 2018).

Numerous methods exist to reduce the "Monte Carlo" noise that comes from drawing samples from the variational distribution (Miller et al., 2017; Roeder et al., 2017; Geffner and Domke, 2018, 2020; Boustati et al., 2020). These can typically be seen as creating an approximation of the objective for which the Monte Carlo noise can be integrated exactly. This approximation can then be used to define a control variate—a zero mean random variable that is negatively correlated with the original gradient estimator. These methods can sometimes be used with data subsampling, essentially by creating different approximations for each datum. However, they are only able to reduce *per-datum* Monte Carlo noise—they do not reduce subsampling noise itself. This is critical, as subsampling noise is often the dominant source of gradient variance (Sec. 3).

For (non-BBVI) optimization problems with *only* subsampling noise, there are numerous incremental gradient methods, that "recycle" previous gradient evaluations to speed up convergence (Roux et al., 2012; Shalev-Shwartz and Zhang, 2013; Johnson and Zhang, 2013; Defazio et al., 2014a,b). However, with few exceptions (Sec. 6) these methods do not address Monte Carlo noise and, due to how they rely on efficiently maintaining running averages, cannot be applied to doubly-stochastic problems.

This paper presents a method that *jointly* controls Monte Carlo and subsampling noise. The idea is to create approximations of the target for each datum, where the Monte Carlo noise can be integrated exactly. The method maintains running averages of the *approximate* gradients, with noise integrated, overcoming the challenge of applying incremental gradient ideas to doubly-stochastic problems. The method addresses both forms of noise and also *interactions* between them. Experiments with variational inference on a range of probabilistic models show that the method yields lower

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. Copyright 2024 by the author(s).

variance gradients and significantly faster convergence than existing approaches.

2 BACKGROUND: BLACK-BOX VARIATIONAL INFERENCE

Given a probabilistic model $p(x, z) = p(z)\prod_{n=1}^{N} p(x_n \mid z)$ and observed data $x_1, \ldots, x_N$, variational inference's goal is to find a tractable distribution $q_w(z)$ to approximate the (often intractable) posterior $p(z \mid x)$ over the latent variable $z \in \mathbb{R}^d$. BBVI achieves this by finding the parameters w that minimize the KL-divergence from $q_w(z)$ to $p(z \mid x)$, equivalent to minimizing the negative Evidence Lower Bound (ELBO)

$$f(w) = -\mathop{\mathbb{E}}_{\mathsf{n}} \mathop{\mathbb{E}}_{q_w(\mathsf{z})} \left[N \log p(x_{\mathsf{n}} \mid \mathsf{z}) + \log p(\mathsf{z}) \right] - \mathbb{H}(w), \tag{1}$$

where $\mathbb{H}(w)$ denotes the entropy of q_w.

The expectation with respect to z in Equation (1) is typically intractable. Thus, BBVI methods rely on stochastic optimization with unbiased gradient estimates, usually based on the score function method (Williams, 1992) or the reparameterization trick (Kingma and Welling, 2014; Rezende et al., 2014; Titsias and Lázaro-Gredilla, 2014). The latter is often the method of choice due to the fact that it often yields estimators with lower variance (Kucukelbir et al., 2017; Xu et al., 2019). The idea is to define a fixed base distribution $s(\epsilon)$ and a deterministic transformation $\mathcal{T}_w(\epsilon)$ such that for $\boldsymbol{\epsilon} \sim s$, we have $\mathcal{T}_w(\epsilon) \sim q_w$. Then, the objective in Equation (1) can be re-written as

$$f(w) = \mathop{\mathbb{E}}_{\mathsf{n}} \mathop{\mathbb{E}}_{\boldsymbol{\epsilon}} f(w; \mathsf{n}, \boldsymbol{\epsilon}), \tag{2}$$

where

$$f(w; n, \epsilon) = -N \log p(x_n \mid \mathcal{T}_w(\epsilon)) - \log p(\mathcal{T}_w(\epsilon)) - \mathbb{H}(w). \tag{3}$$

The "naive" gradient estimate is obtained by drawing a random n and ϵ, and evaluating

$$g_{\texttt{naive}}(w; n, \epsilon) = \nabla f(w; n, \epsilon). \tag{4}$$

Since this only requires point-wise evaluations of $\log p$ and its gradient, it can be applied to a diverse range of models, including those with complex and non-conjugate likelihoods. And by subsampling data, it can be used with large datasets, which may be challenging for traditional methods like MCMC (Hoffman et al., 2013; Kucukelbir et al., 2017). However, the effectiveness of this strategy depends on the gradient estimator's variance; if it is too large, then very small step sizes will be required, slowing convergence.

Task	$\mathbb{V}_{\mathsf{n},\boldsymbol{\epsilon}}[\nabla f(w; \mathsf{n}, \boldsymbol{\epsilon})]$	$\mathbb{V}_{\mathsf{n}}[\nabla f(w; \mathsf{n})]$	$\mathbb{V}_{\boldsymbol{\epsilon}}[\nabla f(w; \boldsymbol{\epsilon})]$
Sonar	4.04×10^4	2.02×10^4	1.16×10^4
Australian	9.16×10^4	8.61×10^4	2.07×10^3
MNIST	4.21×10^8	3.21×10^8	1.75×10^4
PPCA	1.69×10^{10}	1.68×10^{10}	3.73×10^7
Tennis	9.96×10^7	9.59×10^7	8.56×10^4
MovieLens	1.78×10^9	1.69×10^9	1.75×10^6

Table 1: BBVI gradient variance decomposition across tasks, computed at the optimization endpoint. With a batch size of 5, step size of 5×10^{-4} for Sonar and Australian, a batch size of 100, step size of 1×10^{-2} for others. We generally observe subsampling noise $\mathbb{V}_{\mathsf{n}}[\nabla f(w; \mathsf{n})]$ dominates MC noise $\mathbb{V}_{\boldsymbol{\epsilon}}[\nabla f(w; \boldsymbol{\epsilon})]$.

3 GRADIENT VARIANCE IN BBVI

Let $\mathbb{V}_{\mathsf{n},\boldsymbol{\epsilon}}[\nabla f(w; \mathsf{n}, \boldsymbol{\epsilon})]$ denote the variance of the naive estimator from Eq. (4).[1] The two sources of variance correspond to data subsampling (n) and Monte Carlo noise (ϵ). It is natural to ask how much variance each of these sources contributes.

Let $f(w; n) = \mathbb{E}_{\boldsymbol{\epsilon}} f(w; n, \boldsymbol{\epsilon})$ be the objective for a single datum n with Monte Carlo noise integrated out. Similarly, let $f(w; \epsilon) = \mathbb{E}_{\mathsf{n}} f(w; \mathsf{n}, \epsilon)$ be the objective for a fixed ϵ evaluated on the full dataset. In Fig. 1 and Table. 1, we do a single run of BBVI using our proposed gradient estimator (described below). Then, for each iteration on that single optimization trace, we estimate the variance of $\nabla f(w; \mathsf{n}, \boldsymbol{\epsilon})$, $\nabla f(w; \boldsymbol{\epsilon})$, and $\nabla f(w; \mathsf{n})$. We do this for multiple tasks, described in detail in Sec. 7. For later reference, we also include the joint estimator developed below.

The amount of variance contributed by each source is task-dependent. But in many tasks considered, subsampling noise is larger than Monte Carlo noise. This is problematic since computing $f(w; \boldsymbol{\epsilon})$ requires looping over the full dataset, eliminating any benefit of subsampling. These results also illustrate the limitations of any approach that only handles a single source of noise: No control variate applied to each datum can do better than $\nabla f(w; n)$, while no incremental-gradient-type method can do better than $\nabla f(w; \epsilon)$.

4 VARIANCE REDUCTION FOR STOCHASTIC OPTIMIZATION

This section introduces existing methods of variance reduction for stochastic optimization problems with a single source of gradient variance and their applicability to doubly-stochastic settings.

[1] When z is a vector, we use $\mathbb{V}[\mathsf{z}] = \operatorname{tr} \mathbb{C}[\mathsf{z}]$.

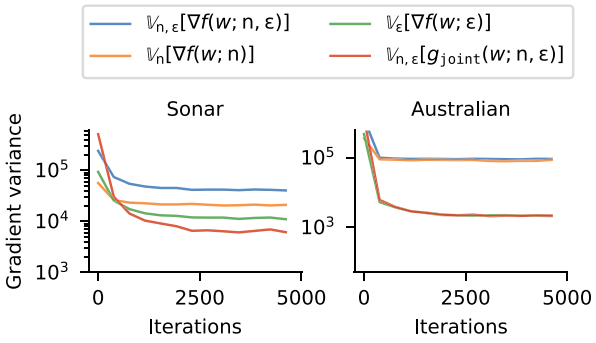


Figure 1: **The contributions of subsampling and Monte Carlo noise vary by problem. The proposed joint estimator reduces both.** Orange lines denote variance from data subsampling (n), and green lines denote Monte Carlo noise variance (ϵ). We use a batch size of 5. For the Sonar dataset, both sources show similar scales. For the Australian dataset, subsampling noise dominates. Regardless, our proposed gradient estimator $g_{\texttt{joint}}$ (red line, Eq. (15)) mitigates subsampling noise and controls MC noise, aligning closely with or below green lines (i.e. the variance without data subsampling) in both datasets.

4.1 Monte Carlo sampling and approximation-based control variates

Suppose we sum over the full dataset in each iteration. Then the objective from Eq. 3 becomes $f(w) = \mathbb{E}_{\epsilon} f(w;\epsilon)$. A gradient can easily be estimated by sampling ϵ. Previous work (Paisley et al., 2012; Tucker et al., 2017; Grathwohl et al., 2018; Boustati et al., 2020) has proposed to reduce the variance using a *control variate* (Robert et al., 1999): A (zero-mean) random variable $c(w;\epsilon)$ negatively correlated with the gradient estimator and defining the new estimator

$$g(w;\epsilon) = \nabla f(w;\epsilon) + c(w;\epsilon). \tag{5}$$

The hope is that $c(w;\epsilon) \approx \nabla f(w) - \nabla f(w;\epsilon)$ approximates the noise of the original estimator, which can lead to large reductions in variance and thus more efficient and reliable inference.

A general way to construct control variates involves using an approximation function $\tilde{f} \approx f$ for which the expectation $\mathbb{E}_{\epsilon} \tilde{f}(w,\epsilon)$ is available in closed-form (Miller et al., 2017; Geffner and Domke, 2020). Then, the control variate is defined as $c(w;\epsilon) = \mathbb{E}_{\xi} \nabla\tilde{f}(w;\xi) - \nabla\tilde{f}(w;\epsilon)$, and the estimator from Eq. (5) becomes

$$g(w;\epsilon) = \nabla f(w;\epsilon) + \mathop{\mathbb{E}}_{\xi} \nabla\tilde{f}(w;\xi) - \nabla\tilde{f}(w;\epsilon). \tag{6}$$

The better $\tilde{f}$ approximates f, the lower the variance of this estimator tends to be. (For a perfect approximation, the variance is zero.) A popular choice for $\tilde{f}$ is a quadratic function as the expectation of a quadratic under a Gaussian is tractable. The quadratic can be learned (Geffner and Domke, 2020) or obtained through a second-order Taylor expansion (Miller et al., 2017).

In doubly-stochastic problems of the form $f(w;n,\epsilon)$, data n is subsampled as well as ϵ. While the above control variate has typically been used without subsampling, it can be adapted to the doubly-stochastic setting by developing an approximation $\tilde{f}(w;n,\epsilon)$ to $f(w;n,\epsilon)$ for each datum n. This leads to the control variate $\mathbb{E}_{\xi} \nabla\tilde{f}(w;n,\xi) - \nabla\tilde{f}(w;n,\epsilon)$ and gradient estimator

$$g_{\texttt{cv}}(w;n,\epsilon) = \nabla f(w;n,\epsilon) + \underbrace{\mathop{\mathbb{E}}_{\xi} \nabla\tilde{f}(w;n,\xi) - \nabla\tilde{f}(w;n,\epsilon)}_{\text{zero mean control variate } c_{\texttt{cv}}(w;n,\epsilon)}. \tag{7}$$

Note, however, that such a control variate cannot reduce subsampling noise. Even if $\tilde{f}(w;n,\epsilon)$ were a *perfect* approximation there would still be gradient variance due to n being sampled randomly. Using the law of total variance, one can show that

$$\mathbb{V}[g_{\texttt{cv}}] = \mathop{\mathbb{E}}_{\mathsf{n}} \mathop{\mathbb{V}}_{\epsilon}[\nabla f(w;n,\epsilon) - \nabla\tilde{f}(w;\mathsf{n},\epsilon)] + \mathop{\mathbb{V}}_{\mathsf{n}}[\nabla f(w;\mathsf{n})]. \tag{8}$$

(See Appendix. C.1 for a proof.) While the first term on the right-hand side can be made arbitrarily small if $\tilde{f}$ is close to f, the second term is irreducible. Fig. 2 and Table 1 show that this subsampling variance is typically substantial, and may be orders of magnitude larger than Monte Carlo variance. When this is true, this type of control variate can only have a limited effect on overall gradient variance.

4.2 Data subsampling and incremental gradient methods

Now consider an objective $f(w) = \mathbb{E}_{\mathsf{n}} f(w;\mathsf{n})$, where n is uniformly distributed on $\{1,\dots,N\}$ and there is no Monte Carlo noise. While one can compute the exact gradient by looping over n, this is expensive when N is large. A popular alternative is to use stochastic optimization by drawing a random n and using the estimator $\nabla f(w;\mathsf{n})$. Alternatively, *incremental gradient* methods (Roux et al., 2012; Shalev-Shwartz and Zhang, 2013; Johnson and Zhang, 2013; Defazio et al., 2014b; Gower et al., 2020) can lead to faster convergence. While details vary by algorithm, the basic idea of these methods is to "recycle" previous gradient evaluations to reduce randomness. SAGA (Defazio et al., 2014a), for instance, stores w^n for the most recent iteration where $f(w;n)$ was evaluated and takes a step

$$w \leftarrow w - \lambda \left(\nabla f(w;n) + \mathop{\mathbb{E}}_{\mathsf{m}} \nabla f(w^{\mathsf{m}};\mathsf{m}) - \nabla f(w^n;n) \right), \tag{9}$$

where λ is the step size. The expectation over m is tracked efficiently using a running average, so the cost per iteration is independent of N. This update rule can be interpreted as regular stochastic gradient descent using the naive estimator $\nabla f(w; n)$ along with a control variate, i.g.

$$g(w; n) = \nabla f(w; n) + \underbrace{\mathop{\mathbb{E}}_{\mathsf{m}} \nabla f(w^{\mathsf{m}}; \mathsf{m}) - \nabla f(w^n; n)}_{\text{zero mean control variate}}. \tag{10}$$

When $w^m \approx w$, the first and last terms in Eq. (10) will approximately cancel, leading to a gradient estimator with much lower variance.

We now consider a doubly-stochastic objective $f(w; n, \epsilon)$. In principle, one might compute the estimator from Eq. (10) for each value of ϵ, i.e. use the gradient estimator

$$g_{\texttt{inc}}(w; n, \epsilon) = \nabla f_n(w; n, \epsilon) + \underbrace{\mathop{\mathbb{E}}_{\mathsf{m}} \nabla f(w^{\mathsf{m}}; \mathsf{m}, \epsilon) - \nabla f(w^n; n, \epsilon)}_{\text{zero mean control variate } c_{\texttt{inc}}(w; n, \epsilon)}. \tag{11}$$

One issue with this is that it does not address Monte Carlo noise. It can be shown that the variance is

$$\mathbb{V}[g_{\texttt{inc}}] = \mathop{\mathbb{E}}_{\epsilon} \mathop{\mathbb{V}}_{\mathsf{n}} [\nabla f(w; \mathsf{n}, \epsilon) - \nabla f(w^{\mathsf{n}}; \mathsf{n}, \epsilon)] + \mathop{\mathbb{V}}_{\epsilon} [\nabla f(w; \epsilon)]. \tag{12}$$

(See Appendix C.2 for a proof.) Since the second term above is irreducible, the variance does not go to zero even when all the stored parameters w^n are to the current parameters. Intuitively, this estimator cannot do better than evaluating the objective on the full dataset for a random ϵ.

But there is an even larger issue: $g_{\texttt{inc}}$ *cannot be implemented efficiently.* The value of $\nabla f(w^{\mathsf{n}}; n, \epsilon)$ depends on ϵ, which is resampled at each iteration. Therefore, it is not possible to efficiently maintain $\mathbb{E}_{\mathsf{m}} \nabla f(w^{\mathsf{m}}; \mathsf{m}, \epsilon)$ (as needed by Eq. (11)) as a running average. The only general strategy is to compute this by looping over the full dataset in each iteration, eliminating the computational benefit of subsampling. For some models with special structures (e.g. log-linear models), it is possible to efficiently maintain the needed running average (Wang et al., 2013; Zheng and Kwok, 2018), but this can only be done in special cases with model-specific derivations, breaking the universality of BBVI.

4.3 Ensembles of control variate

It can be valuable to ensemble multiple control variates. For example, (Geffner and Domke, 2018) combined control variates that reduced Monte Carlo noise (Miller et al., 2017) with one that reduced subsampling noise (Wang et al., 2013) (for a special case where $g_{\texttt{inc}}$ is tractable). While this approach can be better than either control variate alone, it does not reduce *joint* variance. To see this, consider a gradient estimator that uses a convex combination of the two above control variates. For any $\beta \in (0, 1)$ write

$$g_{\texttt{ens}}(w; n, \epsilon) = \nabla f(w; n, \epsilon) + \underbrace{\beta c_{\texttt{cv}}(w; n, \epsilon) + (1 - \beta) c_{\texttt{inc}}(w; n, \epsilon)}_{\text{zero mean control variate } c_{\texttt{ens}}(w; n, \epsilon)}. \tag{13}$$

Even if both $c_{\texttt{cv}}$ and $c_{\texttt{inc}}$ are "perfect" (i.e. $\tilde{f}(w; n, \epsilon) = f(w; n, \epsilon)$ and $w^n = w$ for all n), then the variance is

$$\mathbb{V}[g_{\texttt{ens}}] = \beta^2 \mathop{\mathbb{V}}_{\mathsf{n}} [\nabla f(w; \mathsf{n})] + (1 - \beta)^2 \mathop{\mathbb{V}}_{\epsilon} [\nabla f(w; \epsilon)]. \tag{14}$$

(See Appendix C.3 for a proof.) So, even in this idealized scenario, such an estimator cannot reduce variance to zero. Lastly, as $g_{\texttt{ens}}$ relies on $c_{\texttt{inc}}$, it also faces the computational efficiency issue of $g_{\texttt{inc}}$, making it impractical in general problems.

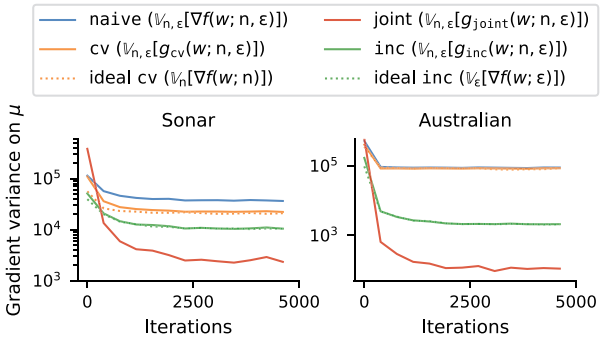


Figure 2: **In practice, `cv` and `inc` reduce variance nearly as much as theoretically possible. The `joint` estimator variance is lower than these bounds.** The `naive` gradient estimator (Eq. (4)) is the baseline, while the `cv` estimator (Eq. (7)) controls the Monte Carlo noise, the `inc` estimator (Eq. (11)) controls for subsampling noise, and the proposed `joint` estimator (Eq. (15)) controls for both. The variance of `cv` and `inc`, as is shown in Eq. (8) and Eq. (12) are lower-bounded by the dotted lines, while `joint` is capable of reducing the variance to significantly lower values, leading to better and faster convergence (first two grids in Fig. 3).

5 JOINT CONTROL VARIATE

We now introduce the *joint control variate*, a new approach for controlling the variance of gradient estimators for BBVI, which *jointly* reduces both subsampling noise from n and Monte Carlo noise from ϵ. In order to construct such control variate, we take two steps:

1. Create an approximation $\tilde{f}(w; n, \epsilon)$ of the true objective $f(w; n, \epsilon)$, designed so that the expectation $\mathbb{E}_{\epsilon} \nabla \tilde{f}(w; n, \epsilon)$ can easily be computed for any datum n (Miller et al., 2017; Geffner and Domke, 2020). A common strategy for this is a Taylor-expansion—replacing f with a low-order polynomial. If the base distribution $s(\epsilon)$ is simple, the expectation $\mathbb{E}_{\epsilon}[\nabla \tilde{f}(w; n, \epsilon)]$ may be available in closed-form.
2. Inspired by SAGA (Defazio et al., 2014a), maintain a table $W = \{w^1, \ldots, w^N\}$ with $w^n \in \mathbb{R}^D$ that stores the variational parameters at the last iteration each of the data points $x_1, \cdots, x_N$ were accessed, along with a running average of gradient estimates evaluated at the stored parameters, denoted by G. Unlike SAGA, however, this running average is for the gradients of the *approximation* $\tilde{f}$, with the Monte Carlo noise ϵ integrated out, i.e. $G = \mathbb{E}_{\mathsf{n}} \mathbb{E}_{\epsilon} \nabla \tilde{f}(w^{\mathsf{n}}; \mathsf{n}, \epsilon)$. In practice, we initialize W using a single epoch of optimization with the `naive` estimator.

Intuitively, as optimization nears the solution, the parameters w tend to change slowly, meaning the entries w^n in W will tend to become close to the current iterate w. So if $\tilde{f}$ is a good approximation of the true objective, we may expect $\nabla f(w; n, \epsilon)$ to be close to $\nabla \tilde{f}(w^n; n, \epsilon)$, meaning the two will be strongly correlated. However, thanks to the running average G, the full expectation of $\nabla \tilde{f}(w^n; n, \epsilon)$ is available in closed-form. This leads to our proposed gradient estimator

$$g_{\texttt{joint}}(w; n, \epsilon) = \nabla f(w; n, \epsilon) + \underbrace{\mathop{\mathbb{E}}_{\mathsf{m}} \mathop{\mathbb{E}}_{\xi} \nabla \tilde{f}(w^{\mathsf{m}}; \mathsf{m}, \xi) - \nabla \tilde{f}(w^n; n, \epsilon)}_{\text{zero mean control variate}}. \quad (15)$$

The running average $G = \mathbb{E}_{\mathsf{n}} \mathbb{E}_{\epsilon} \nabla \tilde{f}(w^{\mathsf{n}}; \mathsf{n}, \epsilon)$ can be cheaply maintained through optimization, since a single value w^n changes per iteration and $\mathbb{E}_{\epsilon} \nabla \tilde{f}(w; n, \epsilon)$ is known in closed form. The variance of the proposed gradient estimator is

$$\mathbb{V}[g_{\texttt{joint}}] = \mathop{\mathbb{V}}_{\epsilon, \mathsf{n}}[\nabla f(w; \mathsf{n}, \epsilon) - \nabla \tilde{f}(w^{\mathsf{n}}; \mathsf{n}, \epsilon)]. \quad (16)$$

This shows that the variance of $g_{\texttt{joint}}$ can be arbitrarily small, only limited by how close $\tilde{f}$ is to f and how close the stored values w^n are to the current parameters w. This is in contrast with the variance achieved by typical control variates or incremental gradient methods, which are unable to reduce both sources of variance jointly. In fact, as shown in Eq. (8) and Eq. (12), these methods, even in ideal scenarios, are provably unable to produce estimators with zero variance, as they can only handle a single source of gradient noise.

Alg. 1 illustrates how the joint gradient estimator can be used for black-box variational inference. The same idea could also be applied more generally to doubly-stochastic objectives in other domains. A generic version of the algorithm and an example of how it can be applied for generalized linear models with Gaussian dropout on the feature is shown in Appendix. E.

Memory overhead Like SAGA, our method requires $\Theta(ND)$ storage for the parameter table W. However, it is easy to create analogous methods based on other incremental gradient methods. In Appendix. B, we develop an analogous method based on SVRG (Johnson and Zhang, 2013) which only requires $\Theta(D)$ storage. Our empirical evaluation shows that its performance is comparable to the SAGA version. However, it has an extra hyperparameter that controls the frequency of recomputing full dataset gradient and involves additional gradient evaluations per iteration.

Advantages over existing estimators Compared with `cv` and `inc`, `joint` can reach arbitrary small gradient variance without lower bound (Eq. (16)), we empirically verify the lower bounds on two small problems: Fig. 2 shows a detailed trace of gradient variance for different estimators using the same optimization trace acquired from `joint`, where the variance of `cv` and `inc` both reach the theoretical lower bounds derived in Eq. (8) and Eq. (12), whereas `joint` shows much lower variance. A summarization of variance lower bounds can be seen in Table 3. Moreover, unlike `inc`, our proposed `joint` estimator controls subsampling noise without the efficiency issue, as `joint` only stores (approximate) gradients after integrating over the Monte Carlo variable ϵ, which makes the needed running average independent of ϵ.

6 RELATED WORK

Recently, Boustati et al. (2020) proposed to approximate the optimal per-datum control variate for BBVI using a recognition network. This takes subsampling into account. However, like $c_{\texttt{cv}}$, this control variate reduces the *conditional* variance of MC noise (conditioned on n) but does not address subsampling noise.

Also, Bietti and Mairal (2017) proposed new incremental gradient method called SMISO, designed for doubly-stochastic problems, which we will compare to below. Intuitively, this uses exponential averages to approximately marginalize out ϵ, and then runs MISO/Finito (Defazio et al., 2014b; Mairal, 2015) (a method similar to SAGA) to reduce subsampling noise. This is similar in spirit to running SGD with a kind of joint control variate. However, it is not obvious how to separate the control variate from the algorithm, meaning we cannot use the SMISO idea as a control variate to get a gradient estimator that can be used

Algorithm 1 Black-box variational inference with the joint control variate.

Input step size λ, negative ELBO estimator $f(w; n, \epsilon)$, and approximation $\tilde{f}(w; n, \epsilon)$ with closed-form over ϵ.
Initialize parameters w and parameter table $W = \{w^1, \ldots, w^N\}$ using a single epoch with `naive`.
Initialize running mean. $G \leftarrow \mathbb{E}_{\mathsf{m}} \mathbb{E}_{\xi} \nabla \tilde{f}(w; \mathsf{m}, \xi)$ ▷ Sum over m, closed-form over ξ
Repeat until convergence:
  Sample n and ϵ.
  Compute base gradient. $g \leftarrow \nabla f(w; n, \epsilon)$
  Compute control variate. $c \leftarrow \mathbb{E}_{\mathsf{m}} \mathbb{E}_{\xi} \nabla \tilde{f}(w^{\mathsf{m}}; \mathsf{m}, \xi) - \nabla \tilde{f}(w^{\mathsf{n}}; n, \epsilon)$ ▷ Use $\mathbb{E}_{\mathsf{m}} \mathbb{E}_{\xi} \nabla \tilde{f}(w^{\mathsf{m}}; \mathsf{m}, \xi) = G$
  Update the running mean. $G \leftarrow G + \frac{1}{N} \mathbb{E}_{\xi}\big(\nabla \tilde{f}(w; n, \xi) - \nabla \tilde{f}(w^{\mathsf{n}}; n, \xi)\big)$ ▷ Closed-form over ξ
  Update the parameter table $w^n \leftarrow w$
  Update parameters. $w \leftarrow w - \lambda(g + c)$ ▷ Or use $g + c$ in any stochastic optimization algorithm

Task	N	Dims	Model class
Sonar	208	60	Logistic regression
Australian	690	14	Logistic regression
MNIST	60,000	7,840	Logistic regression
PPCA	60,000	12,544	Matrix factorization
Tennis	169,405	5,525	Bradley Terry model
MovieLens	100,000	85,050	Hierarchical model

Table 2: Dataset size (N), latent dimensionality (Dims) and model class of tasks used in experiments

with other optimizers like Adam, we include a detailed discussion on this issue in Appendix. A. Nevertheless, we still include SMISO as one of our baselines.

7 EXPERIMENTS

This section evaluates the proposed `joint` estimator for BBVI on a range of linear and non-linear probabilistic models, with 208 to $170K$ samples and latent dimensionalities ranging from 14 to $85K$. Aside from two toy models (Sonar and Australian) these are large enough that a single full-batch evaluation of $\log p$ takes 15-20 times longer than subsampled valuation, even when implemented on GPU. We compare the proposed `joint` estimator against the `naive` estimator which controls for no variance, as well as estimators that control for Monte Carlo or data subsampling separately. Our experiments on GPUs show that the `joint` estimator's reduced variance leads to better solutions in fewer optimization steps and lower wallclock time. The code can be found at https://github.com/xidulu/JointCV.

7.1 Experiment setup

Tasks and datasets We evaluate our method by performing BBVI on the following tasks (the complete dataset size and latent dimensionality of each task are provided in table. 2):

- **Binary/Multi-class Bayesian logistic regression.** We consider Bayesian logistic regression with standard Gaussian prior for binary classification on the *Sonar* and *Australian* datasets, and multi-class classification on *MNIST* (LeCun et al., 1998).
- **Probabilistic principal component analysis (PPCA).** Given a centered dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$, *PPCA* (Tipping and Bishop, 1999) seeks to extract its principal axes $\boldsymbol{W} \in \mathbb{R}^{D \times K}$ assuming
$$\mathsf{W}_{ij} \sim \mathcal{N}(0, 1), 1 \le i \le D, 1 \le j \le K,$$
$$\mathbf{x}_n \sim \mathcal{N}(\mathbf{0}, \mathsf{W}\mathsf{W}^\top + \mathrm{diag}(\lambda^2)).$$
In our experiments, we use BBVI to approximate the posterior over W. We test *PPCA* on the standardized training set of MNIST with $K = 16$ and $\lambda = \mathbf{1}$.
- **Bradley Terry model for tennis players rating.** Given a set of N tennis match records among M players. Each record has format $(x_{n,1}, x_{n,2}, y_n)$, which denotes a match between players $x_{n,1}$ and $x_{n,2}$ with result $y_n \in \{0, 1\}$: $y_n = 1$ denotes player $x_{n,1}$ winning the match and vice versa. The Bradley Terry model (Bradley and Terry, 1952) assigns each player a score $\theta_m \in \mathbb{R}, m = 1, \ldots, M$, and models the match result via
$$\theta_m \sim \mathcal{N}(0, 1),$$
$$\mathsf{y}_n \sim \mathrm{Bernoulli}(\mathrm{logit}^{-1}(\theta_{x_{n,1}} - \theta_{x_{n,2}})).$$
We subsample over matches and perform inference over the score of each player. Following Giordano et al. (2024), we evaluate the model on men's *tennis* matches log starting from 1960, which contains the results of 169405 matches among 5525 players.
- **MovieLens analysis with Bayesian hierarchical model.** The dataset contains a set of N movie review records from M users, where each record from user m has a feature vector of the movie $\boldsymbol{x}_n \in \{0, 1\}^{18}$ and a user rating $y_n \in \{1, \ldots, 5\}$. Assigning each user a weight matrix $\mathsf{Z}_m \in \mathbb{R}^{18 \times 5}, m = 1, \ldots, M$, we model the review through a hierarchical model
$$\boldsymbol{\mu}_{ij} \sim \mathcal{N}(0, 1), \boldsymbol{\sigma}_{ij} \sim \mathcal{N}(0, 1), 1 \le i \le 18, 1 \le j \le 5$$
$$\mathsf{Z}_m \sim \mathcal{N}(\boldsymbol{\mu}, \exp \boldsymbol{\sigma}),$$
$$\mathsf{y}_n \sim \mathrm{Categorical}\left(\mathrm{softmax}(\boldsymbol{x}_n^\top \mathsf{Z}_m)\right).$$
We evaluate the model on MovieLens100K (Harper

and Konstan, 2015), which has 100,000 reviews from 943 users, and perform subsampling over the reviews.

Variational distribution. We focus on mean-field Gaussian BBVI, where the variational distribution follows a factorized Gaussian $q_w(z) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$, parameterized by $w = (\boldsymbol{\mu}, \log \boldsymbol{\sigma})$. The mean parameters $\boldsymbol{\mu}$ is randomly initialized using a standard Gaussian and and we initialize $\log \boldsymbol{\sigma}$ as $\mathbf{0}$.

Choice of approximation function. For `joint` and `cv`, we use a second-order Taylor expansion as the approximation function $\tilde{f}(w; n, \epsilon)$ (Miller et al., 2017), applied only for the mean parameters $\boldsymbol{\mu}$, as for mean-field Gaussian BBVI the total gradient variance is often dominated by variance from $\boldsymbol{\mu}$ (Geffner and Domke, 2020). We provide further details in Appendix. F.

Baselines. We compare the `joint` estimator ($g_{\texttt{joint}}$, Eq. (15)) with the `naive` estimator ($g_{\texttt{naive}}$, Eq. (4)) and the `cv` estimator ($g_{\texttt{cv}}$, Eq. (7)). For Sonar and Australian (small datasets) we include the `inc` estimator ($g_{\texttt{inc}}$, Eq. (11)) as an additional baseline, which requires a full pass through the dataset at each iteration. For larger-scale tasks, the `inc` estimator becomes intractable, so we use SMISO instead.

Optimization details. For the larger-scale MNIST, PPCA, Tennis, and MovieLens, we optimize using Adam (Kingma and Ba, 2014). For the small-scale Sonar and Australian datasets, we use SGD without momentum for transparency. The optimizer for SMISO is pre-determined by its algorithmic structure and cannot be changed. For all estimators, we perform a step-size search to ensure a fair comparison (see Appendix D), testing step sizes between 10^{-3} and 10^{-1} when using Adam and step sizes between 10^{-5} and 10^{-2} when using SGD.

Mini-batching. We use mini-batches B of data at each iteration (reshuffling each epoch). For SMISO and the `inc` and `joint` estimators, we update multiple entries in the parameter table in each iteration and adjust the running mean accordingly. For the Sonar and Australian datasets, due to their small sizes, we use $|\mathsf{B}| = 5$. For all other datasets we use $|\mathsf{B}| = 100$.

Evaluation metrics. We show optimization traces for the best step size chosen retrospectively for each iteration. All ELBO values reported are on the full dataset, estimated with 5000 Monte Carlo samples. We also show the final ELBO achieved after training vs. the step size used to optimize. All results reported are averages over multiple independent runs (10 runs for Sonar and Australian datasets, and 5 for the larger scale problems).

Experiment environment. We use JAX (Bradbury et al., 2018) to implement BBVI, and NumPyro (Phan et al., 2019) for the models. We conduct all experiments on single GPU machines.

7.2 Results

On Sonar and Australian, while both the `inc` and `cv` estimators display lower variance than the `naive` estimator, our proposed `joint` estimator consistently shows the lowest variance (Fig. 2). This enables the use of larger step sizes, leading to faster convergence (first row in Fig. 3). Notice that, on Austraian, the subsampling noise dominates gradient variance. Thus, `inc` shows performance on par with `joint`. Yet, it is crucial to highlight that `inc` requires a full pass over the entire dataset at each optimization step (only possible with small datasets), while `joint` does not. Lastly, we employ MCMC to obtain true posteriors for Sonar and Australian, benefiting from their small scale. The true posterior allows us to measure approximation error by comparing $q_w(z)$'s mean and variance to the true posterior. Results in Fig. G (Appendix. G) confirm that the accelerated convergence from `joint` also helps reduce the (mean) approximation error at a faster rate.

The results for large-scale models, MNIST, PPCA, Tennis, and MovieLens, are also presented in Fig. 3 (for these datasets, the `inc` estimator is intractable, so we use SMISO as a baseline instead). For MovieLens, as the parameter table required by SAGA does not fit into the GPU memory, we use the SVRG version of the `joint` estimator with an update frequency equal to the length of an epoch, introducing one additional gradient call per iteration (amortized). Broadly, we observe that `joint` leads to faster and improved optimization convergence than `naive` and `cv`. `cv` shows little or no improvement upon `naive`, which implies that most of the improvement in the `joint` estimator comes from reducing subsampling variance. SMISO, which does not adopt momentum nor adaptive step sizes, suffers from significantly slower convergence, as it requires the use of a considerably smaller step size (to prevent diverging during optimization). We provide comparisons of different estimators using SGD in Appendix. H.

7.3 Efficiency analysis

We now study the computational cost of different estimators. In terms of the number of "oracle" evaluations (i.e. evaluations of $f(w; n, \epsilon)$ and its gradient), `naive` is the most efficient, requiring a single oracle evaluation per iteration. The `cv` estimator requires one gradient and one Hessian-vector product, and the `joint` estimator requires one gradient and two Hessian-vector products (one for the control variate and one for updating the running mean G.)

Table. 3 shows measured measured runtimes on an Nvidia 2080ti GPU. All numbers are for a single opti-

Estimator	Variance lower bound	∇f evals per iteration	Wall-clock time per iteration			
			MNIST	PPCA	Tennis	MovieLens
`naive`	$\mathbb{V}_{\mathsf{n},\epsilon}[\nabla f(w;\mathsf{n},\epsilon)]$	1	10.4ms	12.8ms	10.2ms	16.3ms
`cv`	$\mathbb{V}_{\mathsf{n}}[\nabla f(w;\mathsf{n})]$	2	12.8ms	18.5ms	14.6ms	19.6ms
`inc`	$\mathbb{V}_{\epsilon}[\nabla f(w;\epsilon)]$	N+2	328ms	897ms	588ms	-
`joint`	0	3	17.6ms	31.2ms	29.6ms	24.4ms
Fullbatch-`naive`	$\mathbb{V}_{\epsilon}[\nabla f(w;\epsilon)]$	N	201ms	740ms	203ms	267ms
Fullbatch-`cv`	0	2N	360ms	1606ms	246ms	702ms

Table 3: Variance, oracle complexity, and wall-clock time for different estimators. Notice that `inc` is more expensive than Fullbatch-`naive`. We hypothesize this is because `inc` uses separate w^n for different data points, which is less efficient for parallelism. MovieLens is too large to fit the parameter table into GPU memory, so we use the SVRG version of `joint` instead, which requires 4 gradient evals per iteration (amortized).

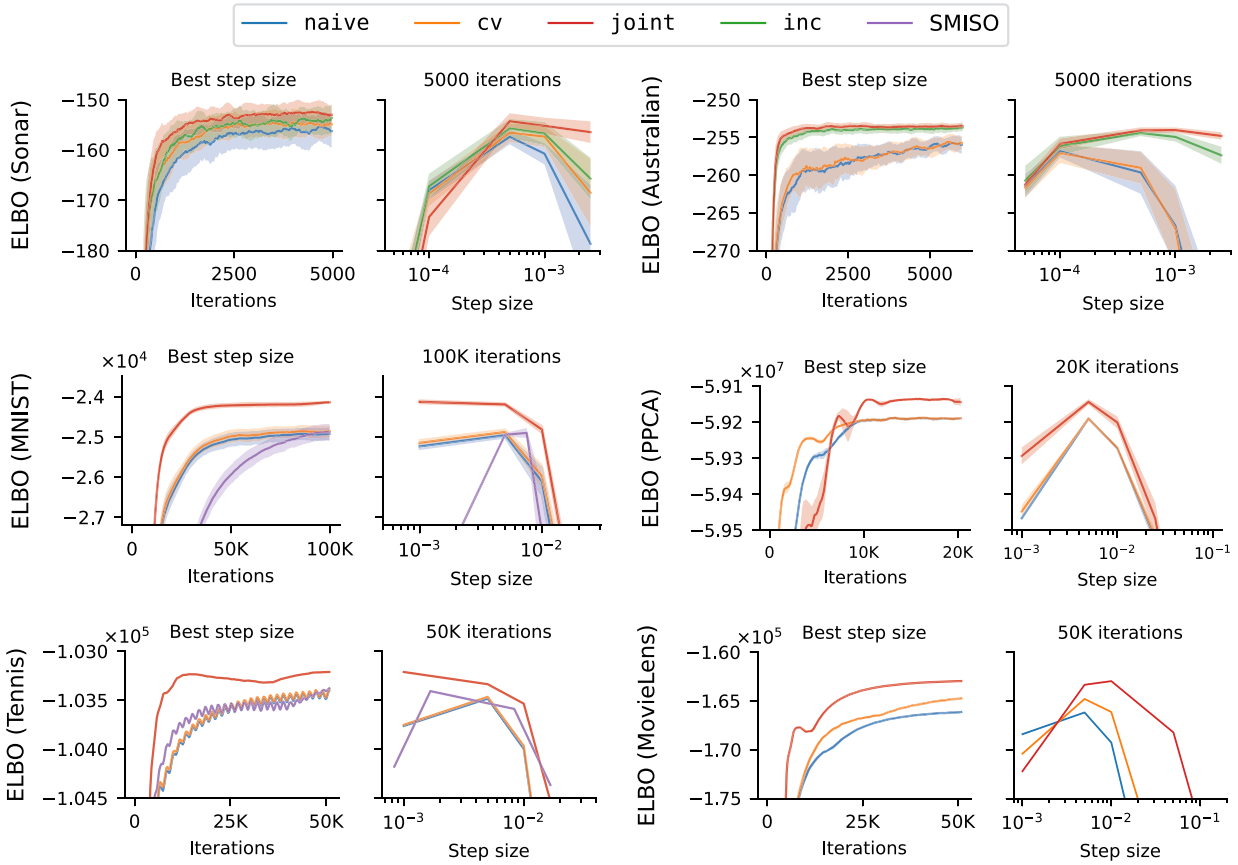


Figure 3: **On various tasks, the proposed `joint` control variate leads to faster convergence through controlling both Monte Carlo and subsampling noise.** Compared to the `naive` estimator, `cv` controls only Monte Carlo noise, while `inc` and SMISO control only subsampling noise. Our proposed `joint` estimator converges faster than `naive` and `cv` on all tasks. The step sizes for SMISO are rescaled for each model for visualization. On PPCA and MovieLens, SMISO has not converged enough to appear, see Fig. 7 in Appendix. H for full results. In Tennis, there is periodic behavior for many estimators as gradients have correlated noise that cancels out at the end of each epoch—the `joint` estimator largely cancels this. All lines presented the average of multiple trials (5 for Sonar and Australian, 10 for the rest), with shaded areas showing one standard deviation.

mization step. For estimators with $\Theta(N)$ oracle complexity (e.g. `inc`) we report average values over 5 steps. For other estimators, we average over 200 steps. Overall, computing the `joint` estimator is between 1.5 to 2.5 times slower than computing the `naive` estimator, and around 1.2 times slower than `cv`. Given that the

`joint` estimator achieves a given performance using an order of magnitude fewer iterations (Fig. 3), it leads to significantly faster optimization than the baselines considered. This can be observed in Appendix. I, where we show optimization results in terms of wall-clock time instead of iterations (i.e. ELBO vs. wall-clock time).

Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant No. 2045900.

References

Alberto Bietti and Julien Mairal. Stochastic optimization with variance reduction for infinite datasets with finite sum structure. In *NeurIPS*, 2017.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112 (518):859–877, 2017.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

Ayman Boustati, Sattar Vakili, James Hensman, and ST John. Amortized variance reduction for doubly stochastic objective. In *UAI*. PMLR, 2020.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NeurIPS*, 2014a.

Aaron Defazio, Justin Domke, et al. Finito: A faster, permutable incremental gradient method for big data problems. In *ICML*. PMLR, 2014b.

Tomas Geffner and Justin Domke. Using large ensembles of control variates for variational inference. In *NeurIPS*, 2018.

Tomas Geffner and Justin Domke. Approximation based variance reduction for reparameterization gradients. In *NeurIPS*, 2020.

Ryan Giordano, Martin Ingram, and Tamara Broderick. Black box variational inference with a deterministic objective: Faster, more accurate, and even more black box. *Journal of Machine Learning Research*, 25(18):1–39, 2024.

Robert M Gower, Mark Schmidt, Francis Bach, and Peter Richtárik. Variance-reduced methods for machine learning. *Proceedings of the IEEE*, 108(11): 1968–1983, 2020.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *ICLR*, 2018.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)*, 5(4): 1–19, 2015.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(40):1303–1347, 2013.

Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15 (1):1593–1623, 2014.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NeurIPS*, 2013.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Tech. report*, 2009.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18(1):430–474, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 25(2):829–855, 2015.

Andrew Miller, Nick Foti, Alexander D'Amour, and Ryan P Adams. Reducing reparameterization gradient variance. In *NeurIPS*, 2017.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

John William Paisley, David M. Blei, and Michael I. Jordan. Variational bayesian inference with stochastic search. In *ICML*, 2012.

Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv preprint arXiv:1912.11554*, 2019.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *AISTATS*. PMLR, 2014.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*. PMLR, 2014.

Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

Geoffrey Roeder, Yuhuai Wu, and David K Duvenaud. Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In *NeurIPS*, 2017.

Nicolas Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NeurIPS*, 2012.

Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes. In *NeurIPS*, 2017.

Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14 (2), 2013.

Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In *ICML*. PMLR, 2014.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *NeurIPS*, 2017.

Chong Wang, Xi Chen, Alexander J Smola, and Eric P Xing. Variance reduction for stochastic gradient optimization. In *NeurIPS*, 2013.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Ming Xu, Matias Quiroz, Robert Kohn, and Scott A Sisson. Variance reduction properties of the reparameterization trick. In *AISTATS*. PMLR, 2019.

Shuai Zheng and James Tin-Yau Kwok. Lightweight stochastic optimization for minimizing finite sums with infinite data. In *ICML*. PMLR, 2018.

A SMISO

In this section, we will have a brief introduction to SMISO (Bietti and Mairal, 2017). Assume we have a loss function of the form

$$\mathop{\mathbb{E}}_{\mathsf{n},\epsilon} f(w;\mathsf{n},\epsilon) \tag{17}$$

Similar to SAGA (Defazio et al., 2014a), SMISO maintains a parameter table $W = \{w^1, \dots, w^N\}$ which stores the parameter value the last time each data point was accessed. SMISO then maintains an average of the value in the parameter table $\bar{w}_k = \mathbb{E}_{\mathsf{n}}\, w_k^{\mathsf{n}}$ where k denotes the k_{th} iteration. $\bar{w}_k$ will later be used as the point for gradient evaluation. Given a randomly drawed sample n and ϵ, SMISO would first update the n_{th} entity in W using exponential average

$$w_n^{k+1} = (1-\alpha)w_n^k + \alpha(\bar{w}_k - \gamma \nabla f(\bar{w}_k;\epsilon,n)). \tag{18}$$

Then, it updates $\bar{w}_k$ using running average

$$\bar{w}_{k+1} = \bar{w}_k + \frac{1}{N}w_n^{k+1} - \frac{1}{N}w_n^k. \tag{19}$$

If we expand the equation above, we get

$$\begin{aligned}
\bar{w}_{k+1} &= \bar{w}_k + \frac{1}{N}w_n^{k+1} - \frac{1}{N}w_n^k && (20)\\
&= \bar{w}_k + \frac{1}{N}\left[(1-\alpha)w_n^k + \alpha(\bar{w}_k - \gamma\nabla f(\bar{w}_k;\epsilon,n)) - w_n^k\right] && (21)\\
&= \bar{w}_k - \frac{\alpha}{N}\left[\gamma\nabla f(\bar{w}_k;\epsilon,n) + w_n^k - \bar{w}_k\right] && (22)\\
&= \bar{w}_k - \frac{\alpha}{N}\left[\gamma\nabla f(\bar{w}_k;\epsilon,n) - (\bar{w}_k - w_n^k)\right] && (23)
\end{aligned}$$

In this case, $\alpha\gamma/N$ is the effective step size. Notice that, if we are using a mini-batch of indices/samples, denoted as $B = \{n_b\}$, in which case multiple entities in the parameter table would be updated in an iteration, then we would have

$$\begin{aligned}
\bar{w}_{k+1} &= \bar{w}_k + \sum_{n_b \in B}\left[\frac{1}{N}w_{n_b}^{k+1} - \frac{1}{N}w_{n_b}^k\right] && (24)\\
&= \bar{w}_k - \frac{\alpha|B|}{N}\left[\gamma \mathop{\mathbb{E}}_{\mathsf{n}_b}\nabla f(\bar{w}_k;\epsilon,\mathsf{n}_b) - \mathop{\mathbb{E}}_{\mathsf{n}_b}(\bar{w}_k - w_{\mathsf{n}_b}^k)\right] && (25)
\end{aligned}$$

in which case the effective step size would become $\frac{\alpha|B|\gamma}{N}$. Therefore, in order to compare SMISO with other estimators using SGD under the same step size, we can first select a range of step sizes for SMISO $\{\gamma_0, \gamma_1, \dots\}$ and test SGD with step sizes of

$$\{\frac{\alpha|B|}{N}\gamma_0, \frac{\alpha|B|}{N}\gamma_1, \dots\}. \tag{26}$$

It is also worth mentioning that, it is not clear to us how to introduce momentum or adaptive step size into SMISO, as we have to strictly follow the running mean update formula (Eq. (19)) to ensure $\mathbb{E}_{\mathsf{n}}(\bar{w}_k - w_{\mathsf{n}}^k) = 0$ for unbiasedness. Adding additional terms (e.g. momentum) or changing the scale of the updates (e.g. normalizing the update by its norm) without careful design could break the unbiasedness. However, studying such modifications is beyond the scope of our paper therefore we only compare our methods with SMISO in its original form.

Algorithm 2 Black-box variational inference with the joint control variate (SVRG version).

Input step size λ, negative ELBO estimator $f(w; n, \epsilon)$, and approximation $\tilde{f}(w; n, \epsilon)$ with closed-form over ϵ.
Input update frequency K.
Initialize the parameter $\tilde{w}$.
Repeat until convergence:
  Compute the full gradient of $\tilde{f}$ at $\tilde{w}$. $\bar{g} \leftarrow \mathop{\mathbb{E}}_{\mathsf{m}} \mathop{\mathbb{E}}_{\xi} \nabla \tilde{f}(\tilde{w}; \mathsf{m}, \xi)$
  Let $w_0 \leftarrow \tilde{w}$.
  for $k = 1, 2, \cdots, K$ do
    Sample n and ϵ.
    Compute base gradient. $g \leftarrow \nabla f(w; n, \epsilon)$
    Compute control variate. $c \leftarrow \mathop{\mathbb{E}}_{\mathsf{m}} \mathop{\mathbb{E}}_{\xi} \nabla \tilde{f}(\tilde{w}; \mathsf{m}, \xi) - \nabla \tilde{f}(\tilde{w}; n, \epsilon)$ ▷ Use $\mathop{\mathbb{E}}_{\mathsf{m}} \mathop{\mathbb{E}}_{\xi} \nabla \tilde{f}(\tilde{w}; \mathsf{m}, \xi) = \bar{g}$
    Update parameters. $w_k \leftarrow w_{k-1} - \lambda(g + c)$ ▷ Or use $g + c$ in any stochastic optimization algorithm
  Update $\tilde{w} \leftarrow w_K$,

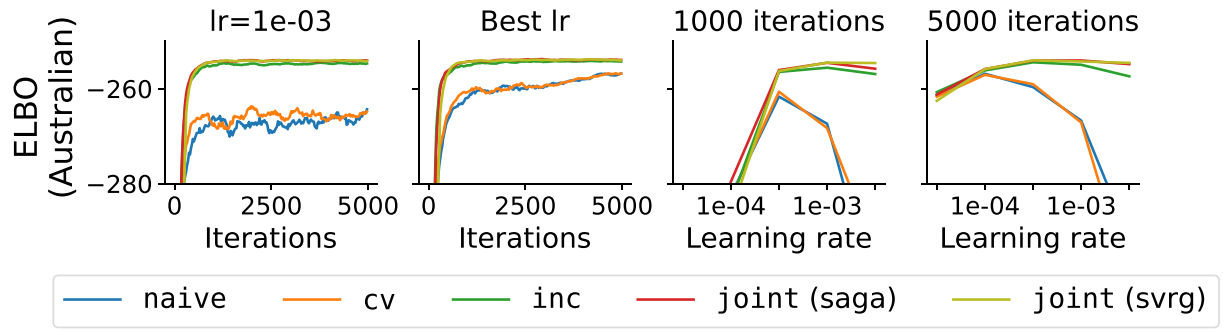


Figure 4: **The SVRG version of `joint` shows performance similar to the SAGA version on Australian.** The origin version of SAGA-based `joint` control variate requires $O(ND)$ memory cost. It is possible to alleviate the additional memory cost by using the SVRG version of `joint`, which costs no extra memory but would require extra gradient evaluation at each step. In the experiments above, we update the SVRG cache every 1 epochs, equivalent to 1 extra gradient evaluation per iteration. Overall, we observe `joint` (svrg) showing results similar to the saga version of `joint`.

B SVRG version of joint control variate

We present the end-to-end algorithm for applying SVRG version of the `joint` control variate in BBVI in Alg. 2. On Australian, we find the SVRG version and SAGA version of `joint` showing similar performance (Fig. 4).

C Derivation of variance for different estimators

In this section, we will show the full derivation for the trace of the variance of $g_{\mathtt{cv}}, g_{\mathtt{inc}}$ and $g_{\mathtt{ens}}$.

C.1 Variance of $g_{\mathtt{cv}}$

In this section, we will derive the trace for the `cv` estimator defined as

$$g_{\mathtt{cv}}(w; n, \epsilon) = \nabla f(w; n, \epsilon) + \underbrace{\mathop{\mathbb{E}}_{\xi} \nabla \tilde{f}(w; n, \xi) - \nabla \tilde{f}(w; n, \epsilon)}_{c_{\mathtt{cv}}(w; n, \epsilon)}, \tag{27}$$

where $\tilde{f}$ is an approximation function of f with closed-form expectation with respect to ϵ.

To start with, we will apply the law of total variance

$$\mathbb{V}[g_{\mathtt{cv}}] = \mathop{\mathbb{E}}_{n} \mathop{\mathbb{V}}_{\epsilon} g_{\mathtt{cv}} + \mathop{\mathbb{V}}_{n} \mathop{\mathbb{E}}_{\epsilon} g_{\mathtt{cv}}. \tag{28}$$

The first term can be computed as

$$\mathop{\mathbb{E}}_{\mathsf{n}}\mathop{\mathbb{V}}_{\epsilon} g_{\texttt{cv}} = \mathop{\mathbb{E}}_{\mathsf{n}}\mathop{\mathbb{V}}_{\epsilon}[\nabla f(w;\mathsf{n},\epsilon) + \mathop{\mathbb{E}}_{\xi}\nabla\tilde{f}(w;\mathsf{n},\xi) - \nabla\tilde{f}(w;\mathsf{n},\epsilon)] \tag{29}$$

$$= \mathop{\mathbb{E}}_{\mathsf{n}}\mathop{\mathbb{V}}_{\epsilon}[\nabla f(w;\mathsf{n},\epsilon) - \nabla\tilde{f}(w;\mathsf{n},\epsilon)], \tag{30}$$

which follows since $\mathbb{E}_{\xi}\nabla\tilde{f}(w;\mathsf{n},\xi)$ is a constant with respect to ϵ and therefore does not affect the variance.

The second term can be computed as

$$\mathop{\mathbb{V}}_{\mathsf{n}}\mathop{\mathbb{E}}_{\epsilon} g_{\texttt{cv}} = \mathop{\mathbb{V}}_{\mathsf{n}}\mathop{\mathbb{E}}_{\epsilon}[\nabla f(w;\mathsf{n},\epsilon) + \mathop{\mathbb{E}}_{\xi}\nabla\tilde{f}(w;\mathsf{n},\xi) - \nabla\tilde{f}(w;\mathsf{n},\epsilon)] \tag{31}$$

$$= \mathop{\mathbb{V}}_{\mathsf{n}}\left[\mathop{\mathbb{E}}_{\epsilon}[\nabla f(w;\mathsf{n},\epsilon)] + \mathop{\mathbb{E}}_{\epsilon}[\mathop{\mathbb{E}}_{\xi}\nabla\tilde{f}(w;\mathsf{n},\xi)] - \mathop{\mathbb{E}}_{\epsilon}[\nabla\tilde{f}(w;\mathsf{n},\epsilon)]\right] \tag{32}$$

$$= \mathop{\mathbb{V}}_{\mathsf{n}}\mathop{\mathbb{E}}_{\epsilon}[\nabla f(w;\mathsf{n},\epsilon) + \nabla\tilde{f}(w;\mathsf{n}) - \nabla\tilde{f}(w;\mathsf{n})] \tag{33}$$

$$= \mathop{\mathbb{V}}_{\mathsf{n}}\mathop{\mathbb{E}}_{\epsilon}[\nabla f(w;\mathsf{n},\epsilon)] \tag{34}$$

$$= \mathop{\mathbb{V}}_{\mathsf{n}}[\nabla f(w;\mathsf{n})]. \tag{35}$$

Then we can combine the two terms together to get

$$\mathbb{V}[g_{\texttt{cv}}] = \mathop{\mathbb{E}}_{\mathsf{n}}\mathop{\mathbb{V}}_{\epsilon}[\nabla f(w;n,\epsilon) - \nabla\tilde{f}(w;\mathsf{n},\epsilon)] + \mathop{\mathbb{V}}_{\mathsf{n}}[\nabla f(w;\mathsf{n})] \tag{36}$$

C.2 Variance of $g_{\texttt{inc}}$

Here, we will derive the trace of the variance of the `inc` estimator defined as

$$g_{\texttt{inc}}(w;n,\epsilon) = \nabla f_n(w;n,\epsilon) + \underbrace{\mathop{\mathbb{E}}_{\mathsf{m}}\nabla f(w^{\mathsf{m}};\mathsf{m},\epsilon) - \nabla f(w^n;n,\epsilon)}_{c_{\texttt{inc}}(w;n,\epsilon)}. \tag{37}$$

We can derive its variance by first applying the law of total variance

$$\mathbb{V}[g_{\texttt{inc}}] = \mathop{\mathbb{E}}_{\epsilon}\mathop{\mathbb{V}}_{\mathsf{n}} g_{\texttt{inc}} + \mathop{\mathbb{V}}_{\epsilon}\mathop{\mathbb{E}}_{\mathsf{n}} g_{\texttt{inc}}. \tag{38}$$

The first term can be computed as

$$\mathop{\mathbb{E}}_{\epsilon}\mathop{\mathbb{V}}_{\mathsf{n}} g_{\texttt{inc}} = \mathop{\mathbb{E}}_{\epsilon}\mathop{\mathbb{V}}_{\mathsf{n}}[\nabla f_n(w;\mathsf{n},\epsilon) + \mathop{\mathbb{E}}_{\mathsf{m}}\nabla f(w^{\mathsf{m}};\mathsf{m},\epsilon) - \nabla f(w^{\mathsf{n}};\mathsf{n},\epsilon)] \tag{39}$$

$$= \mathop{\mathbb{E}}_{\epsilon}\mathop{\mathbb{V}}_{\mathsf{n}}[\nabla f(w;\mathsf{n},\epsilon) - \nabla f(w^{\mathsf{n}};\mathsf{n},\epsilon)], \tag{40}$$

where the second line follows because $\mathbb{E}_{\mathsf{m}}\nabla f(w^{\mathsf{m}};\mathsf{m},\epsilon)$ is a constant with respect to n.

The second term can be computed as

$$\mathop{\mathbb{V}}_{\epsilon}\mathop{\mathbb{E}}_{\mathsf{n}} g_{\texttt{inc}} = \mathop{\mathbb{V}}_{\epsilon}\mathop{\mathbb{E}}_{\mathsf{n}}[\nabla f_n(w;\mathsf{n},\epsilon) + \mathop{\mathbb{E}}_{\mathsf{m}}\nabla f(w^{\mathsf{m}};\mathsf{m},\epsilon) - \nabla f(w^{\mathsf{n}};\mathsf{n},\epsilon)] \tag{41}$$

$$= \mathop{\mathbb{V}}_{\epsilon}\left[\mathop{\mathbb{E}}_{\mathsf{n}}\nabla f_n(w;\mathsf{n},\epsilon) + \mathop{\mathbb{E}}_{\mathsf{n}}\mathop{\mathbb{E}}_{\mathsf{m}}\nabla f(w^{\mathsf{m}};\mathsf{m},\epsilon) - \mathop{\mathbb{E}}_{\mathsf{n}}\nabla f(w^{\mathsf{n}};\mathsf{n},\epsilon)\right] \tag{42}$$

$$= \mathop{\mathbb{V}}_{\epsilon}\left[\mathop{\mathbb{E}}_{\mathsf{n}}[\nabla f_n(w;\mathsf{n},\epsilon)] + \mathop{\mathbb{E}}_{\mathsf{m}}\nabla f(w^{\mathsf{m}};\mathsf{m},\epsilon) - \mathop{\mathbb{E}}_{\mathsf{n}}\nabla f(w^{\mathsf{n}};\mathsf{n},\epsilon)\right] \tag{43}$$

$$= \mathop{\mathbb{V}}_{\epsilon}\mathop{\mathbb{E}}_{\mathsf{n}}[\nabla f_n(w;\mathsf{n},\epsilon)] \tag{44}$$

$$= \mathop{\mathbb{V}}_{\epsilon}[\nabla f(w;\epsilon)], \tag{45}$$

which then leads us to

$$\mathbb{V}[g_{\texttt{inc}}] = \mathop{\mathbb{E}}_{\epsilon}\mathop{\mathbb{V}}_{\mathsf{n}}[\nabla f(w;\mathsf{n},\epsilon) - \nabla f(w^{\mathsf{n}};\mathsf{n},\epsilon)] + \mathop{\mathbb{V}}_{\epsilon}[\nabla f(w;\epsilon)]. \tag{46}$$

C.3 Variance of $g_{\texttt{ens}}$

In this section, we will derive the variance for the estimator $g_{\texttt{ens}}$ defined as

$$g_{\texttt{ens}}(w;n,\epsilon) = \nabla f(w;n,\epsilon) + \underbrace{\beta c_{\texttt{cv}}(w;n,\epsilon) + (1-\beta)c_{\texttt{inc}}(w;n,\epsilon)}_{c_{\texttt{ens}}(w;n,\epsilon)}, \tag{47}$$

under the ideal assumption where we have $f = \tilde{f}$ and $w = w^n, \forall n$. The variance can be derived through

$$\mathbb{V}[g_{\texttt{ens}}] = \mathop{\mathbb{V}}_{\epsilon,\mathsf{n}}[\nabla f(w;\mathsf{n},\epsilon) + \beta c_{\texttt{cv}}(n,\epsilon) + (1-\beta)c_{\texttt{inc}}(\mathsf{n},\epsilon)] \tag{48}$$

$$= \mathop{\mathbb{V}}_{\epsilon,\mathsf{n}}\left[\nabla f(w;\mathsf{n},\epsilon) + \beta\left(\mathop{\mathbb{E}}_{\xi}\nabla\tilde{f}(w;\mathsf{n},\xi) - \nabla\tilde{f}(w;\mathsf{n},\epsilon)\right) + \right. \tag{49}$$

$$\left. (1-\beta)\left(\nabla\mathop{\mathbb{E}}_{\mathsf{m}} f(w^{\mathsf{m}};\mathsf{m},\epsilon) - \nabla f(w^{\mathsf{n}};\mathsf{n},\epsilon)\right)\right]$$

Then we replace $\tilde{f}$ with f and w^n with w based on our assumption,

$$\mathbb{V}[g_{\texttt{ens}}] = \mathop{\mathbb{V}}_{\epsilon,\mathsf{n}}\left[\nabla f(w;\mathsf{n},\epsilon) + \beta\left(\mathop{\mathbb{E}}_{\xi}\nabla f(w;\mathsf{n},\xi) - \nabla f(w;\mathsf{n},\epsilon)\right) + \right. \tag{50}$$

$$\left. (1-\beta)\left(\nabla\mathop{\mathbb{E}}_{\mathsf{m}} f(w;\mathsf{m},\epsilon) - \nabla f(w;\mathsf{n},\epsilon)\right)\right]$$

$$= \mathop{\mathbb{V}}_{\epsilon,\mathsf{n}}\left[\nabla f(w;\mathsf{n},\epsilon) + \beta\left(\nabla f(w;\mathsf{n}) - \nabla f(w;\mathsf{n},\epsilon)\right) + (1-\beta)\left(f(w;\epsilon) - f(w;\mathsf{n},\epsilon)\right)\right] \tag{51}$$

$$= \mathop{\mathbb{V}}_{\epsilon,\mathsf{n}}\left[\beta\nabla f(w;\mathsf{n}) + (1-\beta)\nabla f(w;\epsilon)\right] \tag{52}$$

$$= \beta^2\mathop{\mathbb{V}}_{\mathsf{n}}[\nabla f(w;\mathsf{n})] + (1-\beta)^2\mathop{\mathbb{V}}_{\epsilon}[\nabla f(w;\epsilon)]. \tag{53}$$

The last line follows because $\nabla f(w;n)$ is independent of $\nabla f(w;\epsilon)$.

D Step-size search range

For Australian and Sonar, we experiment with learning rates of

$$\{7.5\times10^{-3}, 5\times10^{-3}, 2.5\times10^{-3}, 1\times10^{-3}, 5\times10^{-4}, 1\times10^{-4}, 5\times10^{-5}, 2.5\times10^{-5}, 1\times10^{-5}\}$$

For MNIST, PPCA ,Tennis and MovieLens, we used

$$\{1\times10^{-1}, 5\times10^{-2}, 1\times10^{-2}, 5\times10^{-3}, 1\times10^{-3}\}$$

for `naive`, `cv` and `joint`, where the optimizer is Adam.

When optimizing with SMISO, we set $\alpha = 0.9$ and we perform grid search over the value of γ, for MNIST with SMISO, we experiment with γ in

$$\{5\times10^{-2}, 2.5\times10^{-2}, 1\times10^{-2}, 5\times10^{-3}, 2.5\times10^{-3}, 1\times10^{-3}, 5\times10^{-4}, 1\times10^{-4}, 5\times10^{-5}, 1\times10^{-5}\}$$

For Tennis with SMISO, we experiment with γ in

$$\{5\times10^{-2}, 2.5\times10^{-2}, 1\times10^{-2}, 5\times10^{-3}, 1\times10^{-3}, 1\times10^{-4}, 1\times10^{-5}\}$$

For PPCA with SMISO, we experiment with γ in

$$\{1\times10^{-2}, 5\times10^{-3}, 1\times10^{-3}, 1\times10^{-4}, 1\times10^{-5}, 1\times10^{-6}, 1\times10^{-7}\}$$

For MovieLens with SMISO, we experiment with γ in

$$\{2.5\times10^{-3}, 1\times10^{-3}, 5\times10^{-4}, 1\times10^{-4}\}$$

Algorithm 3 Joint control variate for generic doubly-stochastic optimization problem.

Input step size λ, doubly-stochastic objective $f(w; n, \epsilon)$, and approximation $\tilde{f}(w; n, \epsilon)$ with closed-form over ϵ.
Initialize parameters w and parameter table $W = \{w^1, \ldots, w^N\}$ using a single epoch with `naive`.
Initialize running mean. $G \leftarrow \mathbb{E}_{\mathsf{m}} \mathbb{E}_{\xi} \nabla \tilde{f}(w; \mathsf{m}, \xi)$ ▷ Sum over m, closed-form over ξ
Repeat until convergence:
  Sample n and ϵ.
  Compute base gradient. $g \leftarrow \nabla f(w; n, \epsilon)$
  Compute control variate. $c \leftarrow \mathop{\mathbb{E}}_{\mathsf{m}} \mathop{\mathbb{E}}_{\xi} \nabla \tilde{f}(w^{\mathsf{m}}; \mathsf{m}, \xi) - \nabla \tilde{f}(w^{\mathsf{n}}; n, \epsilon)$ ▷ Use $\mathop{\mathbb{E}}_{\mathsf{m}} \mathop{\mathbb{E}}_{\xi} \nabla \tilde{f}(w^{\mathsf{m}}; \mathsf{m}, \xi) = G$
  Update the running mean. $G \leftarrow G + \frac{1}{N} \mathop{\mathbb{E}}_{\xi} \left(\nabla \tilde{f}(w; n, \xi) - \nabla \tilde{f}(w^{\mathsf{n}}; n, \xi) \right)$ ▷ Closed-form over ξ
  Update the parameter table $w^n \leftarrow w$
  Update parameters. $w \leftarrow w - \lambda(g + c)$ ▷ Or use $g + c$ in any stochastic optimization algorithm

E Generic optimization algorithm

In Alg. 1, we describe the end-to-end procedure of applying joint control variate in BBVI. The joint control variate can also be applied in generic doubly-stochastic optimization problems as is shown in Alg. 3.

We evaluate the generic version on generalized linear models with Gaussian dropout, with an objective function defined as

$$f(w) = \mathop{\mathbb{E}}_{\mathsf{n}} \mathop{\mathbb{E}}_{\epsilon} f(w; \mathsf{n}, \epsilon), \tag{54}$$

$$f(w; n, \epsilon) := \mathcal{L}\left(y_n, \phi(x_n; w, \epsilon)\right) \tag{55}$$

$$\phi(x_n; w, \epsilon) = w(\epsilon \odot x_n), \tag{56}$$

where $x_n \in \mathbb{R}^D, y_n \in \mathbb{R}^K, w \in \mathbb{R}^{K \times D}$ and $\epsilon \in \mathbb{R}^D$ is a sample from $\mathcal{N}(\mathbf{1}, \sigma\mathbb{I})$, $\odot$ stands for element-wise product and $\mathcal{L}$ is a loss function such as mean-squared error.

We can find an approximation to Eq. (55) by applying second-order Taylor expansion around $\epsilon = 1$, given by

$$\tilde{f}(w; n, \epsilon) = f(w; n, 1) + (\epsilon - 1)^\top \nabla_\epsilon f(w; n, 1) + \frac{1}{2}(\epsilon - 1)^\top \nabla^2_\epsilon f(w; n, 1)(\epsilon - 1), \tag{57}$$

whose expectation with respect to ϵ can be given in closed-form as

$$\mathop{\mathbb{E}}_{\epsilon} \tilde{f}(w; n, \epsilon) = f(w; n, 1) + \frac{\sigma^2}{2} \mathrm{tr}\left(\nabla^2_\epsilon f(w; n, 1)\right). \tag{58}$$

Results We compare the performance of $g_{\texttt{naive}}$, $g_{\texttt{cv}}$, and $g_{\texttt{joint}}$ on CIFAR-10 (Krizhevsky et al., 2009) classification, where we apply dropout on features extracted from a LeNet (LeCun et al., 1998) pretrained on CIFAR-10 and then fine-tune the output layer using the cross-entropy loss with $\sigma = 0.5$. We use a batch size of 100, and optimize using standard gradient descent without momentum for a wide range of learning rates. We present the results in Figure 5 where we show the trace of objective evaluated on the full training set under different learning rates and different numbers of iterations. We can see that $g_{\texttt{joint}}$ always reaches objectives smaller than the baseline estimators, displaying significantly better convergence for large learning rates.

F Approximation function for mean-field Gaussian BBVI

Recall that, given $w = (\boldsymbol{\mu}, \log \boldsymbol{\sigma})$, the objective function for mean-field Gaussian BBVI is written as

$$f(w) = \mathop{\mathbb{E}}_{\mathsf{n}} \mathop{\mathbb{E}}_{\epsilon} f(w; \mathsf{n}, \epsilon), \tag{59}$$

$$\text{where } f(w; n, \epsilon) = -N \log p(x_n \mid \mathcal{T}_w(\epsilon)) - \log p(\mathcal{T}_w(\epsilon)) - \mathbb{H}(w), \quad \mathcal{T}_w(\epsilon) = \boldsymbol{\mu} + \epsilon \odot \boldsymbol{\sigma}, \tag{60}$$

where we use the ϵ notation here to also represent a vector. Inspired by previous work (Miller et al., 2017), we get an approximation for $f(w; n, \epsilon)$ using a second order Taylor expansion for the negative total likelihood

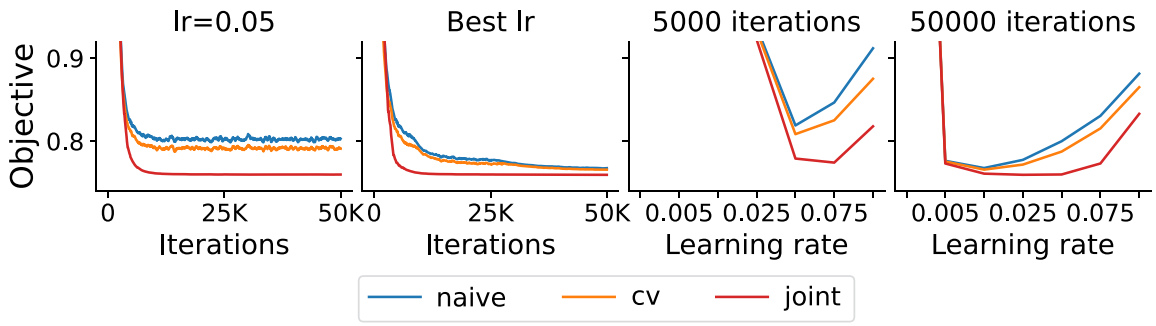


Figure 5: **The joint estimator leads to improved convergence at higher learning rates on Gaussian dropout on CIFAR-10.** (For small enough learning rates, optimization speed is limited by the learning rate itself and so all estimators perform identically.) The first column shows the trace of the objective (logistic loss) under a learning rate of 0.05. The second column shows the trace of the objective under the best learning rate chosen retrospectively at each iteration. The final two columns show the objective as a function of different learning rates at two different numbers of iterations. Note that the learning that the `joint` has its best performance at a higher learning rate than the other estimators. (the `inc` estimator is too expensive to be included here.)

$k_n(z) = N \log p(x_n \mid z) + \log p(z)$ around $z_0 = \mathcal{T}_w(0)$[2], which yields

$$\tilde{f}(w; n, \epsilon) = k_n(z_0) + (\mathcal{T}_w(\epsilon) - z_0)^\top \nabla k_n(z_0) + \frac{1}{2}(\mathcal{T}_w(\epsilon) - z_0)^\top \nabla^2 k_n(z_0)(\mathcal{T}_w(\epsilon) - z_0) + \mathbb{H}(w), \tag{61}$$

where we assume the entropy can be computed in closed form. The approximation function's gradient with respect to the variational parameter is given by:

$$\nabla_w \tilde{f}(w; n, \epsilon) = \left[\frac{\partial\left(\mathcal{T}_w(\epsilon) - z_0\right)}{\partial w}\right]^\top \left[\nabla k_n(z_0) + \nabla^2 k_n(z_0)(\boldsymbol{\mu} + \epsilon \odot \boldsymbol{\sigma} - \boldsymbol{\mu} + \mathbf{0} \odot \boldsymbol{\sigma})\right] + \nabla_w \mathbb{H}(w), \tag{62}$$

$$= \left[\frac{\partial\left(\mathcal{T}_w(\epsilon)\right)}{\partial w}\right]^\top \left[\nabla k_n(z_0) + \nabla^2 k_n(z_0)(\epsilon \odot \boldsymbol{\sigma})\right] + \nabla_w \mathbb{H}(w), \tag{63}$$

where $\frac{\partial(\mathcal{T}_w(\epsilon))}{\partial w}$ denotes Jacobian matrix. Note that, despite the gradient computation involving the Hessian, it can be computed efficiently without explicitly storing the Hessian matrix through Hessian vector product. However, the *expectation* of the gradient can only be can only be computed efficiently with respect to the mean parameter $\boldsymbol{\mu}$ but not for the scale parameter $\boldsymbol{\sigma}$. To see that, we first compute the expected gradient with respect to $\boldsymbol{\mu}$, using the fact that $\frac{\partial(\mathcal{T}_w(\epsilon) - z_0)}{\partial \boldsymbol{\mu}} = \boldsymbol{I}$ and ϵ is zero-mean:

$$\mathop{\mathbb{E}}_{\epsilon} \nabla_{\boldsymbol{\mu}} \tilde{f}(w; n, \epsilon) = \nabla k_n(z_0) + \nabla_{\boldsymbol{\mu}} \mathbb{H}(w) \tag{64}$$

The expected gradient with respect to $\boldsymbol{\sigma}$ is given by:

$$\mathop{\mathbb{E}}_{\epsilon} \nabla_{\boldsymbol{\sigma}} \tilde{f}(w; n, \epsilon) = \mathop{\mathbb{E}}_{\epsilon}\left[\operatorname{diag}(\epsilon)\left[\nabla k_n(z_0) + \nabla^2 k_n(z_0)(\epsilon \odot \boldsymbol{\sigma})\right]\right] + \nabla_{\boldsymbol{\sigma}} \mathbb{H}(w), \tag{65}$$

$$= \mathop{\mathbb{E}}_{\epsilon}\left[\epsilon \odot \nabla k_n(z_0) + \nabla^2 k_n(z_0)(\epsilon^2 \odot \boldsymbol{\sigma})\right] + \nabla_{\boldsymbol{\sigma}} \mathbb{H}(w), \tag{66}$$

$$= \operatorname{diag}\left(\nabla^2 k_n(z_0)\right) \odot \boldsymbol{\sigma} + \nabla_{\boldsymbol{\sigma}} \mathbb{H}(w), \tag{67}$$

which requires the diagonal of the Hessian, causing computing difficulty in many problems. This means $g_{\texttt{cv}}(w; n, \epsilon)$ and $g_{\texttt{joint}}(w; n, \epsilon)$ can only be efficiently used as the gradient estimator for $\boldsymbol{\mu}$. Fortunately, controlling only the gradient variance on $\boldsymbol{\mu}$ often means controlling most of the variance, as, with mean-field Gaussians, the total gradient variance is often dominated by variance from $\boldsymbol{\mu}$ (Geffner and Domke, 2020).

[2]We use $z_0 = \text{stop_gradient}\left(\mathcal{T}_w(0)\right)$ so that the gradient does not backpropagate from z_0 to w.

G Comparision with true posterior

On small problems, in particular, Sonar and Australian, we can acquire the true posterior using MCMC and compute the approximation error of the variational posterior using the true posterior. To be more specific, we run MCMC on these two problems using NUTS (Hoffman et al., 2014) with 4 chains, where we warm up for 5,000 steps and then collect 25,000 samples from each chain, giving a total of 100K samples that we use to estimate the mean and variance of the true posterior, denoted as $\boldsymbol{\mu}_{\text{mcmc}}$ and $\boldsymbol{\sigma}^2_{\text{mcmc}}$ respectively. We then compute the L2 distance between the mean and variance of the variational posterior (a diagonal Gaussian) and that of the true posterior as the approximation error. We additionally acquire a set of ground truth variational parameters using full dataset and 200 Monte Carlo samples for gradient estimation, optimized for 10,000 iterations with a learning rate of 1×10^{-4}. The approximation error based on the ground truth parameter serves as a reference value on the *smallest* error each estimator can achieve.

The results are presented in Fig. 6, the observation aligns with the ELBO traces (Fig. 3), where `joint` is capable of approaching the true posterior mean at a speed faster than with baseline estimators, eventually reach the approximation error of the ground truth variational parameters.

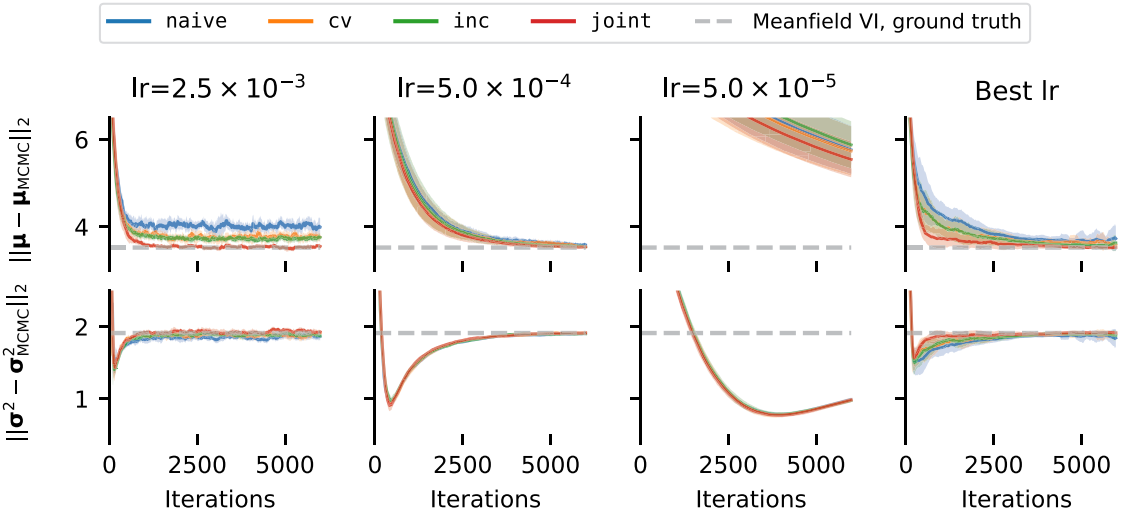
H Results under SGD

In this section, we compare `naive`, `cv`, and `joint` with SMISO using SGD. The step sizes for SMISO are the same as the values shown in Sec. D. The step sizes for other models under SGD are converted through Eq. (26) correspondingly. Additionally, we compare their performance with the optimization results acquired using Adam. The results are presented in Fig. 7 and Fig. 8. Overall, with SGD, `joint` still shows superior performance compared with baseline estimators except for MovieLens, where all estimators fail to converge under the selected step sizes (and using larger step sizes could cause divergence in optimization). In addition, all estimators show performance worse than that of Adam when optimized with SGD except for `joint` on Tennis.

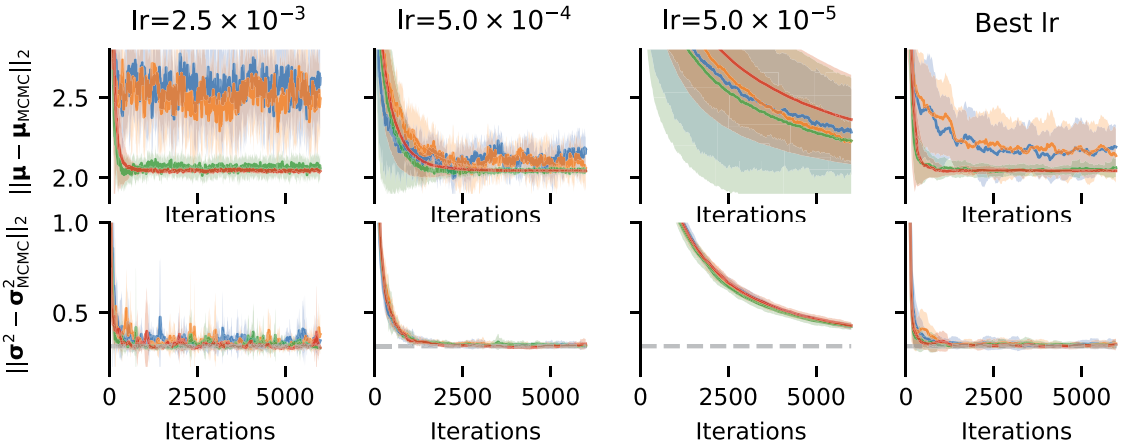
Note that, when experimenting with PPCA using `joint` and SGD, we perform updates with `naive` in the first three epochs to avoid diverging, as the `joint` shows a high gradient norm in the first few epochs when SAGA is still warming up. This modification is not required when using Adam, as Adam adaptively chooses the step size based on the gradient norm.

I Wall clock time v.s convergence

In this section, we provide the wall clock time v.s. convergence results. The results are presented in Fig. 9. The results are identical to the results in the second column in Fig. 3 with the x-axis for each estimator rescaled using the values from Table. 3.



(a) Sonar



(b) Australian

Figure 6: **Meanfield VI under the proposed `joint` estimator approaches true posterior mean faster**. On small-scale problems, we compare the mean and variance of the variational posterior (a diagonal Gaussian) to that of the true posterior estimated with MCMC using 100K samples, with approximation error measured by L2 distance. The last column shows the error trace under the best learning rate chosen retrospectively at each iteration based on the ELBO. The grey dashed lines show the error under ground truth variational parameters acquired with full batch gradient and 200 Monte Carlo samples per iteration, representing the best error each estimator can achieve. On the mean parameter, `joint` reduces the error faster than baseline estimators. For variance errors, all estimators demonstrate similar behavior, as we are only controlling the variance on the mean parameters. The results presented are based on 10 random trials, where the solid lines denote the averaged values and the shaded area represents the one standard deviation.

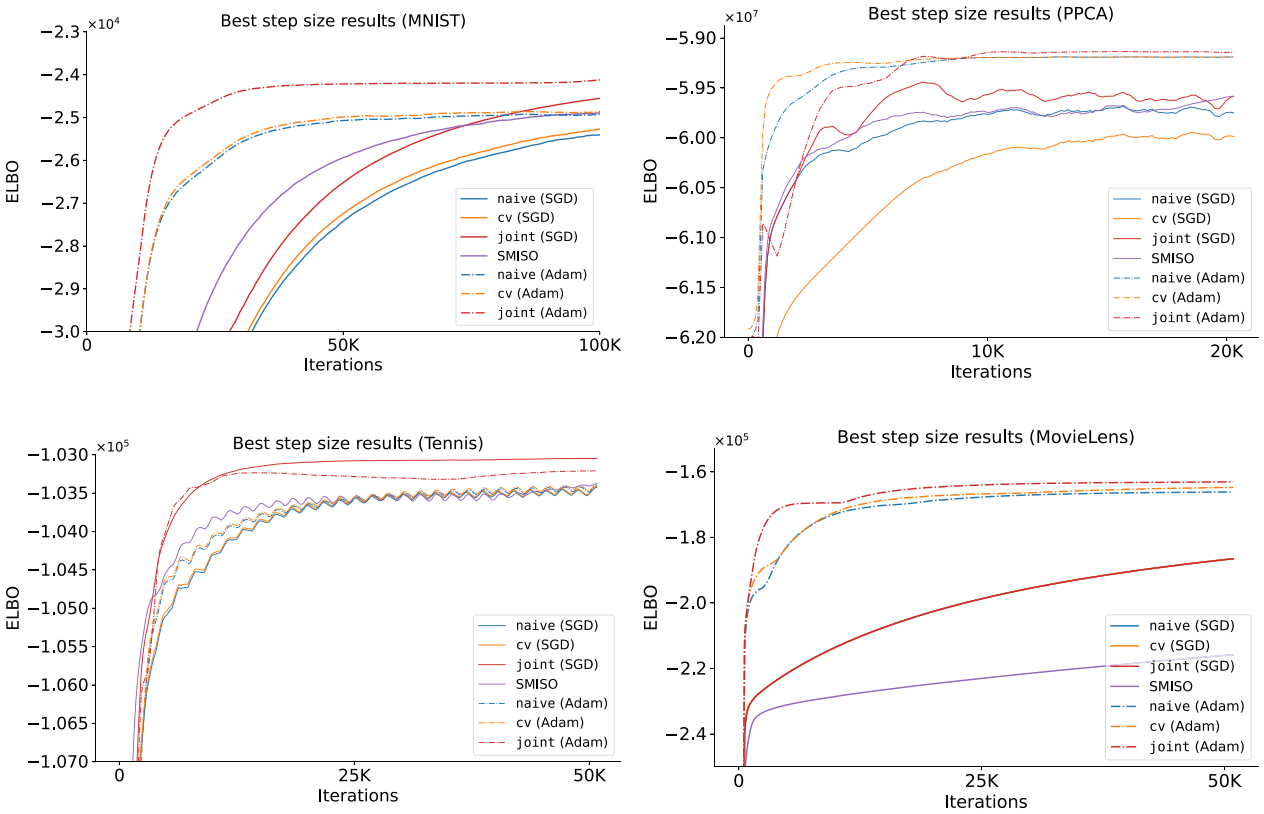
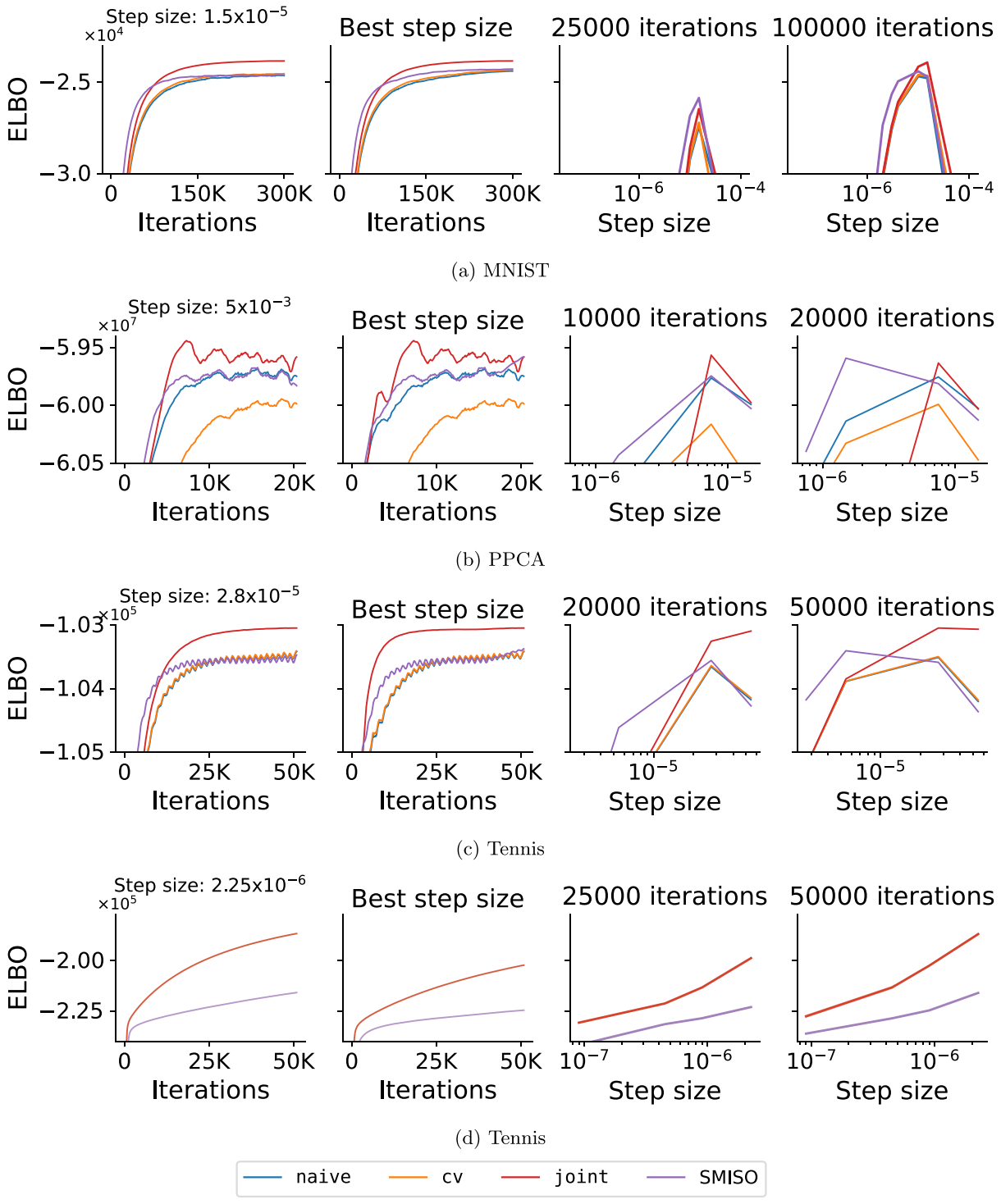


Figure 7: **Comparision of different estimators on MNIST, PPCA, and Tennis under SGD and Adam**. The proposed `joint` combined with Adam shows the best performance on all tasks except Tennis, in which `joint` with SGD demonstrates the best convergence. For other estimators, Adam leads to better and faster convergence than SGD.



(a) MNIST

(b) PPCA

(c) Tennis

(d) Tennis

Figure 8: **Optimization results on MNIST, PPCA, Tennis and MovieLens with SGD**. Using SGD does not affect the improvement of `joint` against `naive` and `cv`. In addition, we notice that `joint` still performs better than SMISO under SGD, we suspect that this is because `joint` marginalizes ϵ out explicitly while SMISO approximates the expectation using exponential averaging. All methods show slow convergence with SGD on MovieLens under the largest step size, emphasizing the importance of using adaptive optimization methods.

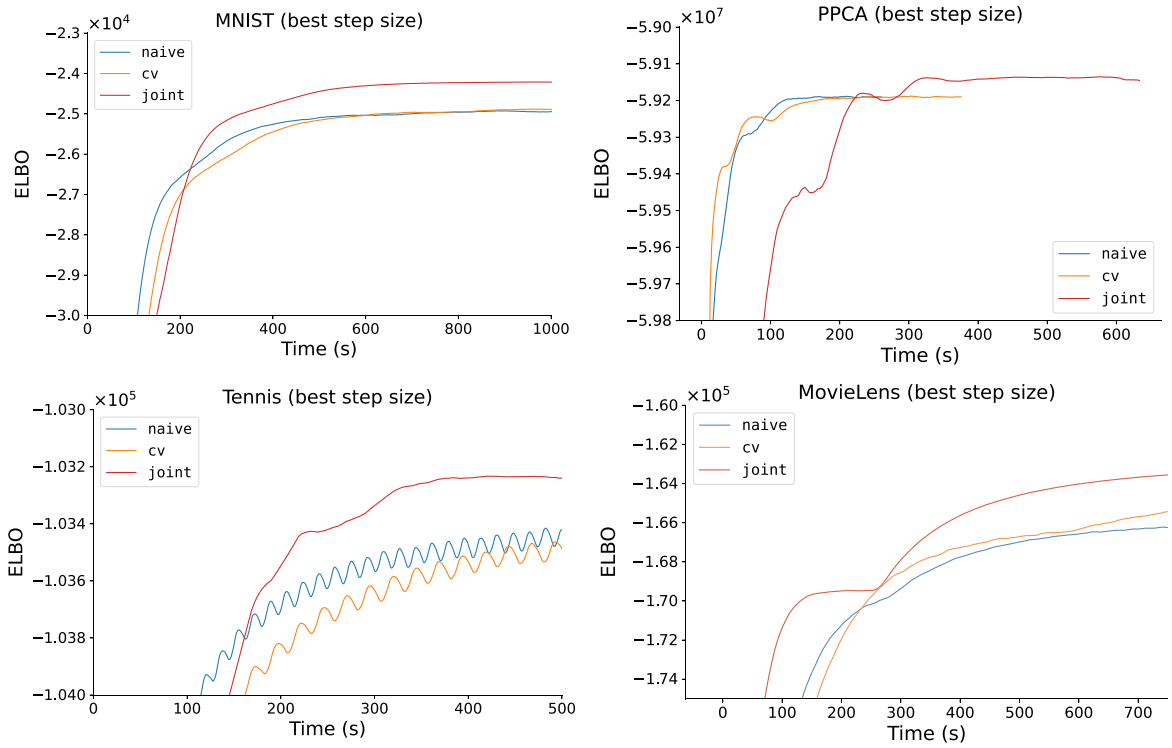


Figure 9: **On large scale problems, the joint estimator leads to faster convergence in terms of wall-clock time.** For example, on MNIST, it takes `joint` around 300 seconds to reach an ELBO of -2.5×10^4 whereas the cv and the naive estimator would take around 600 seconds. On PPCA, while the `joint` estiamtor displays slower convergence in the beginning, as SAGA is still warming up, it is capable of reaching much better results at the end of the optimization.