Effective Monte Carlo Variational Inference for Binary-Variable Probabilistic Programs

Geng Ji   Erik B. Sudderth

UC Irvine

Monte Carlo Variational Inference

Binary Probabilistic Models: $p(z, x)$ in which the latent variables z are binary; no constraints on observed vars x.

Evidence Lower Bound (ELBO): mean-field variational bound

$$\mathcal{L}(q) = \mathbb{E}_{q(z)}\left[\log p(z, x) - \log q(z)\right] \leq \log p(x)$$

Co-ordinate Ascent Variational Inference [1]: naïve mean-field update for logits requires computing difficult expectations:

$$\tau_i \triangleq \log \frac{q(z_i = 1)}{q(z_i = 0)} = \mathbb{E}_{q(z_{-i})}\left[\log \frac{p(z_i = 1 \mid z_{-i}, x)}{p(z_i = 0 \mid z_{-i}, x)}\right]$$

Our contribution: Monte Carlo estimate using samples drawn from $q(z_i) = \text{Bernoulli}\left(\frac{1}{1+e^{-\tau_i}}\right)$:

$$\tau_i \approx \frac{1}{M}\sum_{m=1}^{M} \log \frac{p(z_i=1 | z_{-i}^{(m)}, x)}{p(z_i=0 | z_{-i}^{(m)}, x)}$$

Advantages: 1) Linear in the number of samples M, even for models with high-order dependencies.

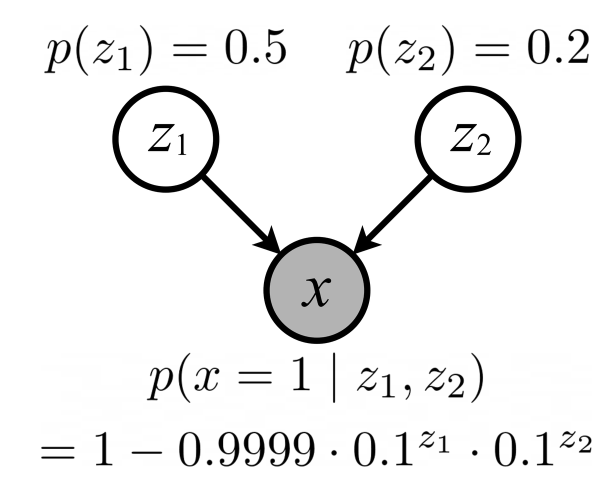
2) Applicable to all probabilistic models with binary latent variables. No model-specific treatment is required.

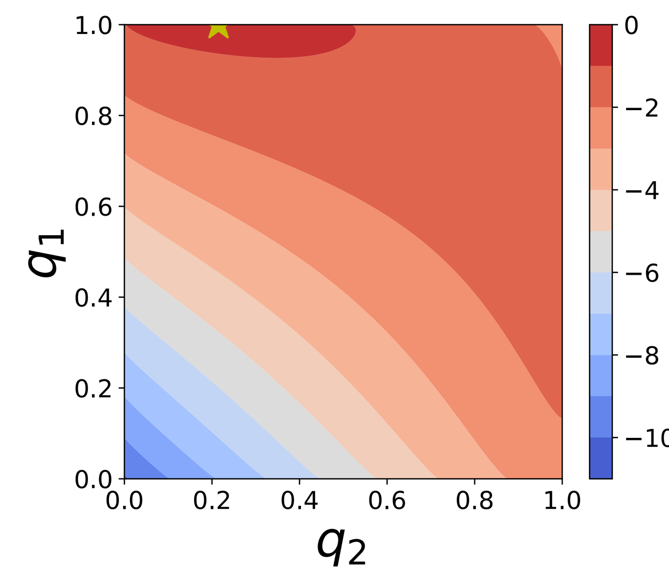
Comparisons with REINFROCE

REINFORCE variational gradient [2]: unbiased stochastic gradient, also known as black-box variational inference (BBVI)

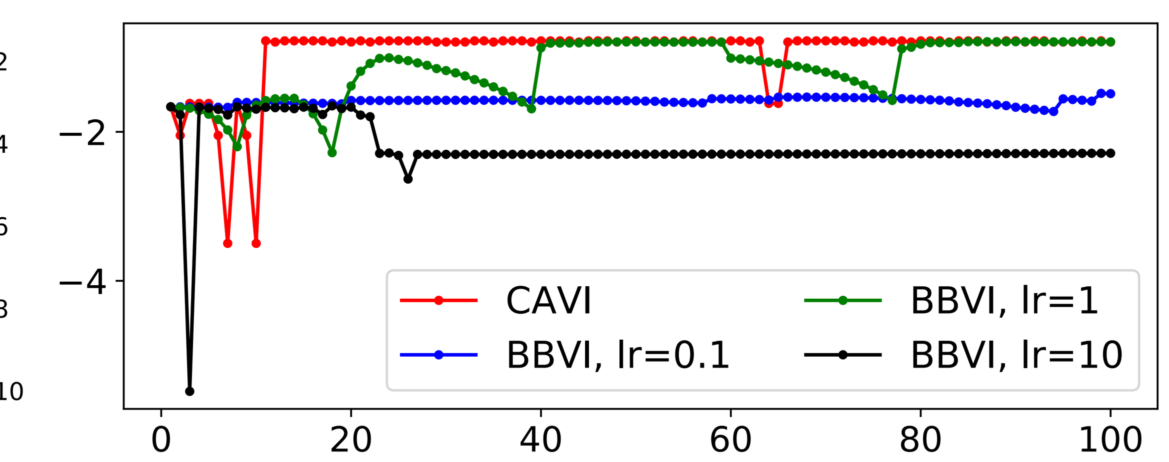
$$\frac{\partial \mathcal{L}}{\partial \tau_i} \approx \frac{1}{M}\sum_{m=1}^{M} \frac{\partial \log q(z_i)}{\partial \tau_i}\bigg|_{z_i^{(m)}} \cdot \left(\log p(z_i^{(m)} \mid z_{-i}^{(m)}, x) - \log q(z_i^{(m)})\right)$$

Drawbacks: very high variance, even under variance-reduction tricks such as Rao Blackwellization and control variates.





A toy noisy-OR model with two latent nodes (left). The ELBO contour plot (right) shows it has a single global optimum at the yellow star.

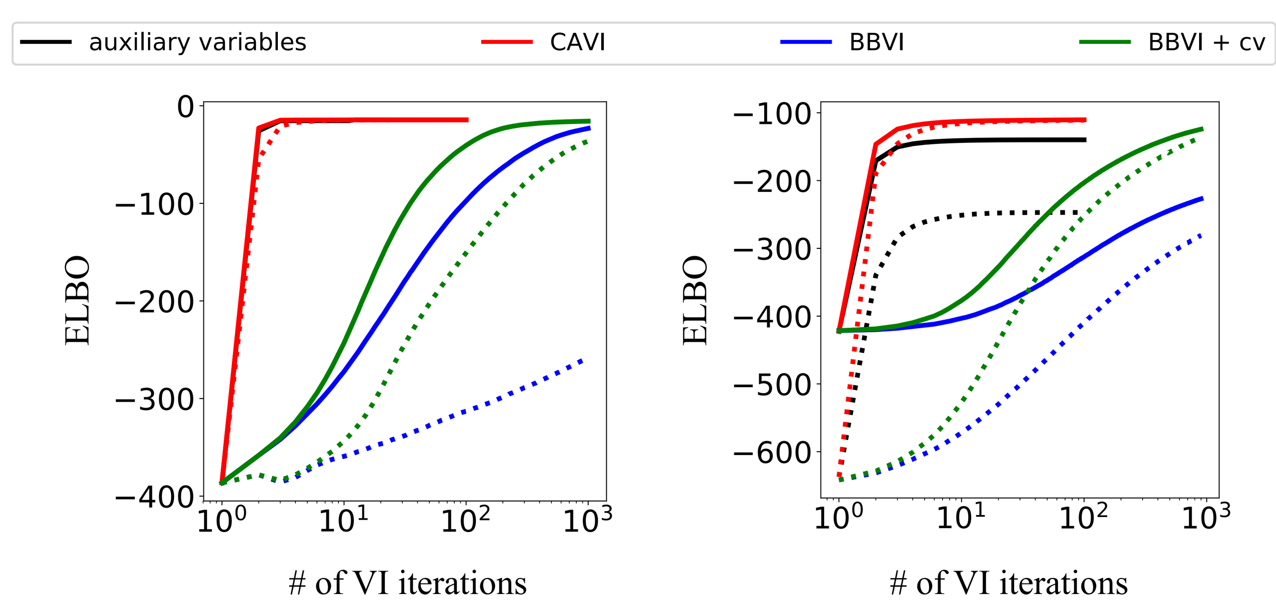


BBVI is sensitive to learning rate and needs many iterations to converge. The starting point is intentionally selected to be bad for CAVI. But it still works quite well.

Experiments: Binary Bayesian Networks

Pyro [3] implementation: specifying three-layer Noisy-OR and Sigmoid belief networks in just 30+ lines of code.

```
import torch
from pyro import plate, sample
from pyro.distributions import Bernoulli

class BN(torch.nn.Module):
  def __init__(self, params):
    super(BN, self).__init__()
    self.b, self.W1, self.c1, self.W2, self.c2 = params
    self.D_H2, self.D_H1 = self.W2.shape

  @abstractmethod
  def squash_fun(self, x):
    raise NotImplementedError

  def model(self, data):
    dat_axis = plate('dat_axis', data.shape[0], dim=-2)
    top_axis = plate('top_axis', self.D_H2, dim=-1)
    mid_axis = plate('mid_axis', self.D_H1, dim=-1)
    bot_axis = plate('bot_axis', data.shape[1], dim=-1)
    with dat_axis, top_axis:
      z_top = sample('z_top', Bernoulli(
                      probs=self.squash_fun(self.b)))
      wz_top = torch.matmul(z_top, self.W2) + self.c2
    with dat_axis, mid_axis:
      z_bot = sample('z_bot', Bernoulli(
                      probs=self.squash_fun(wz_top)))
      wz_bot = torch.matmul(z_bot, self.W1) + self.c1
    with dat_axis, bot_axis:
      sample('x', Bernoulli(
              probs=self.squash_fun(wz_bot)), obs=data)

class NoisyOrBN(BN):
  def squash_fun(self, x):
    return torch.ones([]) - torch.exp(-x)

class SigmoidBN(BN):
  def squash_fun(self, x):
    return torch.sigmoid(x)
```

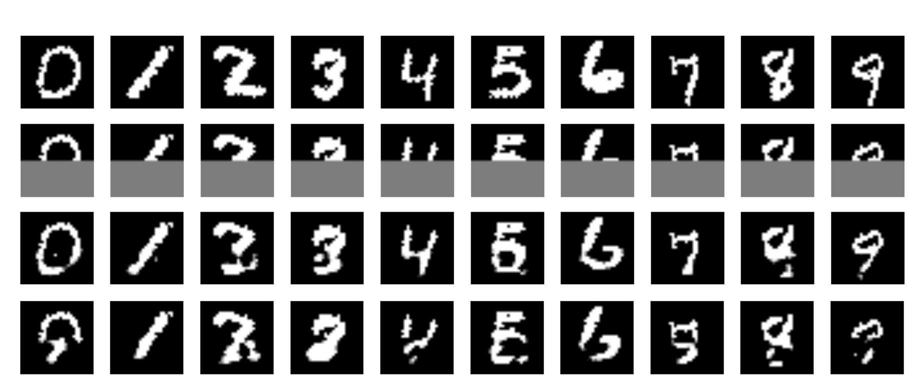


Noisy-OR topic graph [4]

- CAVI converges much faster than best tuned BBVI both w/ and wo/ control variate (cv).
- CAVI needs less # of Monte Carlo samples than BBVI: **solid** lines 10, **dotted** lines 2.

Sigmoid MNIST image model [5]

- CAVI is less sensitive to initial values of q: **solid** lines use prior values of z, **dotted** line use 0.5.
- CAVI is even better than the model-specific auxiliary-variable method in [5].



Original

Occluded

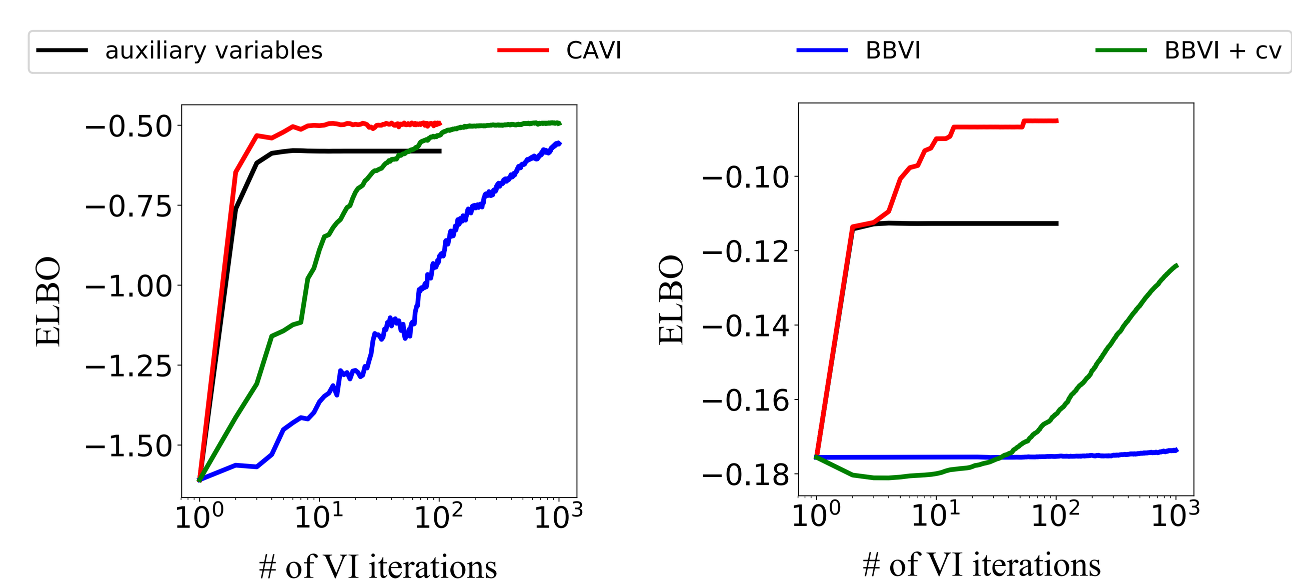
CAVI

BBVI

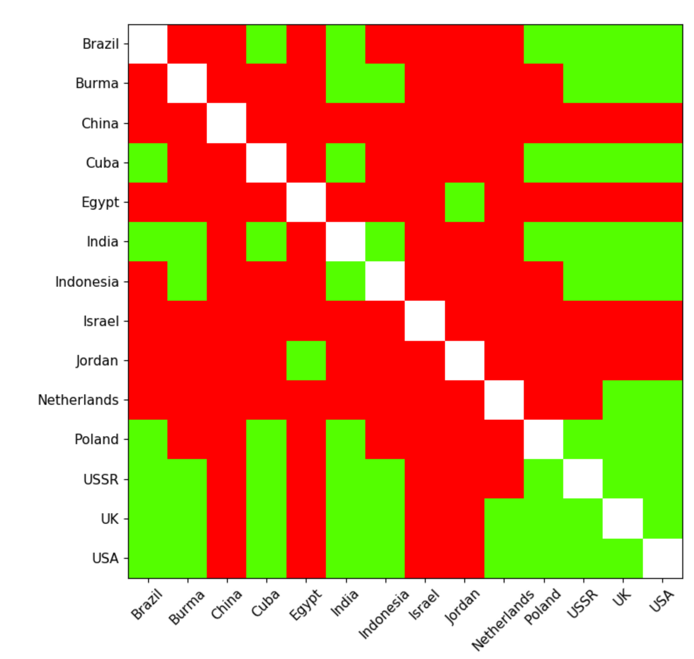
Experiments: Binary Relational Models

Pyro implementation: latent feature relational model [6] with logit (Gaussian cdf) squashing function, without the nonparametric prior.

```
import torch
from pyro import plate, sample
from pyro.distributions import Bernoulli

class LFRM(torch.nn.Module):
    def __init__(self, params):
        super(LFRM, self).__init__()
        self.W, self.W0, self.z_prior = params
        self.D = len(self.W)
        self.squash_fun = torch.distributions.normal.Normal(loc=0, scale=1).cdf

    def model(self, links):
        entity_axis = plate("entity_axis", len(links), dim=-2)
        feature_axis = plate("feature_axis", self.D, dim=-1)
        with entity_axis, feature_axis:
            features = sample("features", Bernoulli(probs=self.z_prior))
        idx = torch.triu(torch.ones_like(links), diagonal=1).nonzero().split(1, dim=1)
        with plate("link_axis"):
            wzz = self.W0 + torch.einsum('id,jd->ij', self.W, features**2)[idx]
            sample("links", Bernoulli(probs=self.squash_fun(wzz)), obs=links[idx])
```



14 Countries model (small)   **NeurIPS authors model (big)**



14 Countries dataset in [6]: presence/absence of each pair of countries co-participating international conferences during 1950 to 1965 or not.

References

[1] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. Machine Learning.

[2] Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In AISTATS.

[3] Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. JMLR.

[4] Ji, G., Cheng, D., Ning, H., Yuan, C., Zhou, H., Xiong, L., and Sudderth, E. B. (2019). Variational training for large-scale noisy-OR Bayesian networks. In UAI.

[5] Gan, Z., Henao, R., Carlson, D., and Carin, L. (2015). Learning deep sigmoid belief networks with data augmentation. In AISTATS.

[6] Miller, K., Jordan, M. I., and Griffiths, T. L. (2009). Nonparametric latent feature models for link prediction. In NeurIPS.