University of BRISTOL

Massively Parallel Bayesian Inference

Importance weighting and more...

Sam Bowyer

Mini-Project Presentation

Supervisor: Laurence Aitchison

17th May 2023

Table of Contents

1. Bayesian Inference
2. Traditional Importance Weighting
3. The Massively Parallel Framework
4. Massively Parallel Importance Weighting
5. Other Massively Parallel Applications

Bayesian Inference

Bayesian inference relies on computing a posterior distribution,

$$P(z'|x) = \frac{P(x|z')P(z')}{\int_{\mathcal{Z}} P(x, z'')dz''} \tag{1}$$

given a prior $P(z')$ over latent variables $z' \in \mathcal{Z}$ and a likelihood $P(x|z')$ for data x.

Bayesian Inference

Bayesian inference relies on computing a posterior distribution,

$$P(z'|x) = \frac{P(x|z')P(z')}{\int_{\mathcal{Z}} P(x, z'')dz''} \tag{1}$$

given a prior $P(z')$ over latent variables $z' \in \mathcal{Z}$ and a likelihood $P(x|z')$ for data x.

Problem: Computation of the normalising constant, (the marginal likelihood)

$$P(x) = \int_{\mathcal{Z}} P(x, z'')dz'', \tag{2}$$

is often intractable (especially if you have a large number of latent variables).

Bayesian Inference

Problem: Computing $P(x) = \int_{\mathcal{Z}} P(x, z'')dz''$.

The full joint latent space $\mathcal{Z}$ is too big.

Bayesian Inference

Problem: Computing $P(x) = \int_{\mathcal{Z}} P(x, z'')dz''$.

The full joint latent space $\mathcal{Z}$ is too big.

(One) solution: Importance Weighting—instead of trying to evaluate $P(x, z'')dz''$ for *every possible* latent setting $z'' \in \mathcal{Z}$, sample $K \in \mathbb{N}$ latent settings $z^1, ..., z^K \in \mathcal{Z}$ and attempt to reason about this collection.

Importance Weighting

In particular, draw K IID samples of your latents from the full joint space, $\mathcal{Z}$,

$$z = (z^1, ..., z^K) \in \mathcal{Z}^K, \tag{3}$$

$$z^k \in \mathcal{Z}, \ \ \forall k \in \mathcal{K} := \{1, \ldots, K\}, \tag{4}$$

Importance Weighting

In particular, draw K IID samples of your latents from the full joint space, $\mathcal{Z}$,

$$z = (z^1, ..., z^K) \in \mathcal{Z}^K, \tag{3}$$

$$z^k \in \mathcal{Z}, \ \ \forall k \in \mathcal{K} := \{1, \dots, K\}, \tag{4}$$

via some proposal distribution Q,

$$Q(z) = \prod_{k \in \mathcal{K}} Q(z^k). \tag{5}$$

(We can choose Q to be easy to sample from and have generally 'nice' qualities.)

Importance Weighting

Define $r_k(z)$ as the likelihood ratio of the latent sample z^k,

$$r_k(z) = \frac{P(x, z^k)}{Q(z^k)}. \tag{6}$$

Importance Weighting

Define $r_k(z)$ as the likelihood ratio of the latent sample z^k,

$$r_k(z) = \frac{P(x, z^k)}{Q(z^k)}. \tag{6}$$

Then we can define a 'global' estimator, $\mathcal{P}_{\text{global}}(z)$, of the marginal likelihood as follows:

$$\mathcal{P}_{\text{global}}(z) = \tfrac{1}{K} \sum_{k \in \mathcal{K}} r_k(z). \tag{7}$$

Importance Weighting

Define $r_k(z)$ as the likelihood ratio of the latent sample z^k,

$$r_k(z) = \frac{P(x, z^k)}{Q(z^k)}. \tag{6}$$

Then we can define a ‘global’ estimator, $\mathcal{P}_{\text{global}}(z)$, of the marginal likelihood as follows:

$$\mathcal{P}_{\text{global}}(z) = \frac{1}{K} \sum_{k \in \mathcal{K}} r_k(z). \tag{7}$$

It is fairly easy to prove that this is unbiased:

$$\mathbb{E}_{Q(z)}[\mathcal{P}_{\text{global}}(z)] = P(x). \tag{8}$$

Importance Weighting: An Issue

Suppose each $z^k \in \mathcal{Z}$ is comprised of n individual latent variables:

$$z^k = (z_1^k, z_2^k, ..., z_n^k) \in \mathcal{Z} \tag{9}$$

(Note: z^k is a tuple, not a vector—each latent variable z_i^k might not be scalar-valued.)

Importance Weighting: An Issue

Suppose each $z^k \in \mathcal{Z}$ is comprised of n individual latent variables:

$$z^k = (z_1^k, z_2^k, ..., z_n^k) \in \mathcal{Z} \tag{9}$$

(Note: z^k is a tuple, not a vector—each latent variable z_i^k might not be scalar-valued.)

Problem: As n increases, the number of samples, K, required to obtain accurate importance weighted estimates through $\mathcal{P}_{\text{global}}$ grows with $\mathcal{O}(e^n)$ [Chatterjee and Diaconis, 2018].

Importance Weighting: An Issue

Suppose each $z^k \in \mathcal{Z}$ is comprised of n individual latent variables:

$$z^k = (z_1^k, z_2^k, ..., z_n^k) \in \mathcal{Z} \tag{9}$$

(Note: z^k is a tuple, not a vector—each latent variable z_i^k might not be scalar-valued.)

Problem: As n increases, the number of samples, K, required to obtain accurate importance weighted estimates through $\mathcal{P}_{\text{global}}$ grows with $\mathcal{O}(e^n)$ [Chatterjee and Diaconis, 2018].

Massively Parallel Solution: Consider all possible K^n combinations of the latent samples within our collection $z = (z^1, ..., z^K) \in \mathcal{Z}^K$.

The Massively Parallel Framework

Let z_i^k be the value of the ith latent variable within the kth sample.

The Massively Parallel Framework

Let z_i^k be the value of the ith latent variable within the kth sample.

Let $\mathbf{k} = (k_1, ..., k_n) \in \mathcal{K}^n$ be a vector of indices.

The Massively Parallel Framework

Let z_i^k be the value of the ith latent variable within the kth sample.

Let $\mathbf{k} = (k_1, ..., k_n) \in \mathcal{K}^n$ be a vector of indices.

Then we can write a combination of the samples with these indices as

$$z^{\mathbf{k}} = \left(z_1^{k_1}, z_2^{k_2}, \dots, z_n^{k_n}\right) \in \mathcal{Z}. \tag{10}$$

For the ith latent variable, we use the k_jth sample.

The Massively Parallel Framework

Let z_i^k be the value of the ith latent variable within the kth sample.

Let $\mathbf{k} = (k_1, ..., k_n) \in \mathcal{K}^n$ be a vector of indices.

Then we can write a combination of the samples with these indices as

$$z^{\mathbf{k}} = \left(z_1^{k_1}, z_2^{k_2}, \ldots, z_n^{k_n}\right) \in \mathcal{Z}. \tag{10}$$

For the ith latent variable, we use the k_jth sample.

Previously: we were essentially only using $\mathbf{k} \in \mathcal{K}^n$ where $k_1 = k_2 = \cdots = k_n$.
E.g. for the third sample drawn from Q,

$$z^3 = \left(z_1^3, z_2^3, \ldots, z_n^3\right) \in \mathcal{Z}.$$

The Massively Parallel Framework

Let z_i^k be the value of the ith latent variable within the kth sample.

Let $\mathbf{k} = (k_1, ..., k_n) \in \mathcal{K}^n$ be a vector of indices.

Then we can write a combination of the samples with these indices as

$$z^{\mathbf{k}} = \left(z_1^{k_1}, z_2^{k_2}, \dots, z_n^{k_n}\right) \in \mathcal{Z}. \tag{10}$$

For the ith latent variable, we use the k_jth sample.

Previously: we were essentially only using $\mathbf{k} \in \mathcal{K}^n$ where $k_1 = k_2 = \cdots = k_n$.
E.g. for the third sample drawn from Q,

$$z^3 = \left(z_1^3, z_2^3, \dots, z_n^3\right) \in \mathcal{Z}.$$

The difference is that *now*, we don't require $k_1 = k_2 = \cdots = k_n$.

The Massively Parallel Framework

Before: We can index $|\mathcal{K}| = K$ samples.

$$k \in \mathcal{K}$$
$$z^k = \left(z_1^k, z_2^k, \ldots, z_n^k\right) \in \mathcal{Z}$$

The Massively Parallel Framework

Before: We can index $|\mathcal{K}| = K$ samples.

$$k \in \mathcal{K}$$

$$z^k = \left(z_1^k, z_2^k, \ldots, z_n^k\right) \in \mathcal{Z}$$

Now: We can index $|\mathcal{K}^n| = K^n$ samples.

$$\mathbf{k} = (k_1, ..., k_n) \in \mathcal{K}^n$$

$$z^{\mathbf{k}} = \left(z_1^{k_1}, z_2^{k_2}, \ldots, z_n^{k_n}\right) \in \mathcal{Z}$$

The Massively Parallel Framework

What does a useful proposal Q_{MP} look like in this setting?

The Massively Parallel Framework

What does a useful proposal Q_{MP} look like in this setting?

One choice is to have a graphical structure:

$$Q_{\text{MP}}(z) = \prod_{i=1}^{n} Q_{\text{MP}}\left(z_i | z_j \ \forall j \in \text{qa}(i)\right), \tag{11}$$

where $\text{qa}(i)$ is the set of indices of parents of z_i under the proposal's graphical model.

The Massively Parallel Framework

What does a useful proposal Q_{MP} look like in this setting?

One choice is to have a graphical structure:

$$Q_{\text{MP}}(z) = \prod_{i=1}^{n} Q_{\text{MP}}\left(z_i | z_j \; \forall j \in \text{qa}(i)\right), \tag{11}$$

where $\text{qa}(i)$ is the set of indices of parents of z_i under the proposal's graphical model. So the value of the ith latent variable depends only on the value of its parents.

The Massively Parallel Framework

What does a useful proposal Q_{MP} look like in this setting?

One choice is to have a graphical structure:

$$Q_{\text{MP}}(z) = \prod_{i=1}^{n} Q_{\text{MP}}\left(z_i | z_j \; \forall j \in \text{qa}(i)\right), \tag{11}$$

where $\text{qa}(i)$ is the set of indices of parents of z_i under the proposal's graphical model. So the value of the ith latent variable depends only on the value of its parents.

(But *which* parents? We have K sets of parent samples to choose from...
Easy answer: for the kth child, only use the kth parents.)

The Massively Parallel Framework

Similarly, we work with a graphical structure on the generative distribution:

$$P(x, z^{\mathbf{k}}) = P\left(x | z_j^{k_j} \ \forall j \in \mathsf{pa}(x)\right) \prod_{i=1}^{n} P\left(z_i^{k_i} | z_j^{k_j} \ \forall j \in \mathsf{pa}(i)\right) \tag{12}$$

where $\mathsf{pa}(i)$ is the set of indices of parents of z_i under the generative graphical model (and $\mathsf{pa}(x)$ is the set of indices of parents for x)..

Massively Parallel Importance Weighting

Now we can define a new, *massively parallel* marginal likelihood estimator $\mathcal{P}_{\text{MP}}$,

$$\mathcal{P}_{\text{MP}}(z) = \tfrac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \tag{13}$$

$$r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^{n} Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{14}$$

Massively Parallel Importance Weighting

Now we can define a new, *massively parallel* marginal likelihood estimator $\mathcal{P}_{\text{MP}}$,

$$\mathcal{P}_{\text{MP}}(z) = \frac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \tag{13}$$

$$r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^{n} Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{14}$$

This is also an unbiased marginal likelihood estimator [Heap and Aitchison, 2023]:

$$\mathbb{E}_{Q_{\text{MP}}}(z)[\mathcal{P}_{\text{MP}}(z)] = P(x). \tag{15}$$

What makes this 'massively parallel'?

Problem: We've got a sum over K^n terms. How do we compute this?

$$\mathcal{P}_{\text{MP}}(z) = \tfrac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \qquad r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^{n} Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{16}$$

What makes this ‘massively parallel’?

Problem: We’ve got a sum over K^n terms. How do we compute this?

$$\mathcal{P}_{\text{MP}}(z) = \frac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \qquad r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^{n} Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{16}$$

Solution:

1. Write $r_{\mathbf{k}}(z)$ as a big tensor product [Aitchison, 2019, Heap and Aitchison, 2023]:

$$r_{\mathbf{k}}(z) = f^x_{\mathbf{k}_{\text{pa}(x)}}(z) \prod_i f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z) \tag{17}$$

What makes this 'massively parallel'?

Problem: We've got a sum over K^n terms. How do we compute this?

$$\mathcal{P}_{\text{MP}}(z) = \frac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \qquad r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^n Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{16}$$

Solution:

1. Write $r_{\mathbf{k}}(z)$ as a big tensor product [Aitchison, 2019, Heap and Aitchison, 2023]:

$$r_{\mathbf{k}}(z) = f^x_{\mathbf{k}_{\text{pa}(x)}}(z) \prod_i f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z) \tag{17}$$

   - $f^x_{\mathbf{k}_{\text{pa}(x)}}(z)$ is a tensor of rank $|\text{pa}(x)|$

What makes this ‘massively parallel’?

Problem: We’ve got a sum over K^n terms. How do we compute this?

$$\mathcal{P}_{\text{MP}}(z) = \frac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \qquad r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^n Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{16}$$

Solution:

1. Write $r_{\mathbf{k}}(z)$ as a big tensor product [Aitchison, 2019, Heap and Aitchison, 2023]:

$$r_{\mathbf{k}}(z) = f^x_{\mathbf{k}_{\text{pa}(x)}}(z) \prod_i f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z) \tag{17}$$

- $f^x_{\mathbf{k}_{\text{pa}(x)}}(z)$ is a tensor of rank $|\text{pa}(x)|$
- $f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z)$ are tensors of rank $1 + |\text{pa}(i)|$

What makes this ‘massively parallel’?

Problem: We’ve got a sum over K^n terms. How do we compute this?

$$\mathcal{P}_{\text{MP}}(z) = \frac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \qquad r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^{n} Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{16}$$

Solution:

1. Write $r_{\mathbf{k}}(z)$ as a big tensor product [Aitchison, 2019, Heap and Aitchison, 2023]:

$$r_{\mathbf{k}}(z) = f^x_{\mathbf{k}_{\text{pa}(x)}}(z) \prod_i f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z) \tag{17}$$

   - $f^x_{\mathbf{k}_{\text{pa}(x)}}(z)$ is a tensor of rank $|\text{pa}(x)|$
   - $f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z)$ are tensors of rank $1 + |\text{pa}(i)|$

2. Use efficient tensor-product implementations (e.g. opt-einsum [a. Smith and Gray, 2018]) which can be performed in parallel on GPUs.

Other Massively Parallel Applications

- Posterior moment estimation

$$m_{\mathsf{MP}}(z) = \tfrac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} \frac{r_{\mathbf{k}}(z)}{\mathcal{P}_{\mathsf{MP}}(z)} m(z^{\mathbf{k}}). \tag{18}$$

Other Massively Parallel Applications

- Posterior moment estimation

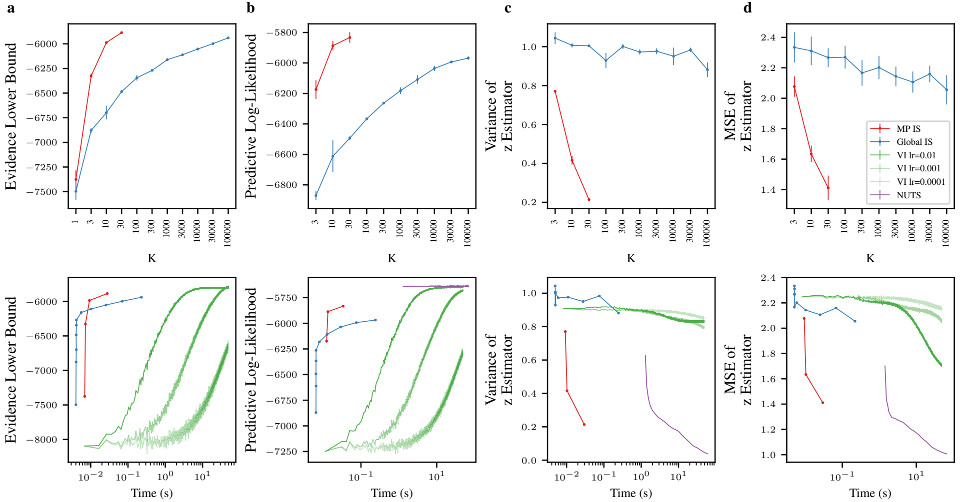
$$m_{\text{MP}}(z) = \tfrac{1}{K^n} \sum_{\mathbf{k}\in\mathcal{K}^n} \frac{r_{\mathbf{k}}(z)}{\mathcal{P}_{\text{MP}}(z)} m(z^{\mathbf{k}}). \tag{18}$$

- Importance sampling

$$m_{\text{MP}}(z) = \sum_{\mathbf{k}\in\mathcal{K}} P(\mathbf{k}) m(z^{\mathbf{k}}) \qquad P(\mathbf{k}) = \frac{1}{K_n} \frac{1}{\mathcal{P}_{\text{MP}}(z)} r_{\mathbf{k}}(z). \tag{19}$$

Sampling $\mathbf{k}$ from $P(\mathbf{k})$ will give us $z^{\mathbf{k}}$ that are approximate samples from the true posterior $P(z|x)$.

Performance (on MovieLens 100K Dataset)



Conclusion

- A general approach for parallelising a variety of inference methods:

Conclusion

- A general approach for parallelising a variety of inference methods:
  - Massively Parallel MCMC

Conclusion

- A general approach for parallelising a variety of inference methods:
  - Massively Parallel MCMC
  - Massively parallel Reweighted Wake-Sleep (RWS) (Bornschein and Bengio [2015]) and Importance Weighted Autoencoders (IWAE) Burda et al. [2016] developed in Heap and Aitchison [2023].

Conclusion

- A general approach for parallelising a variety of inference methods:
  - Massively Parallel MCMC
  - Massively parallel Reweighted Wake-Sleep (RWS) (Bornschein and Bengio [2015]) and Importance Weighted Autoencoders (IWAE) Burda et al. [2016] developed in Heap and Aitchison [2023].
- Future work:

Conclusion

- A general approach for parallelising a variety of inference methods:
  - Massively Parallel MCMC
  - Massively parallel Reweighted Wake-Sleep (RWS) (Bornschein and Bengio [2015]) and Importance Weighted Autoencoders (IWAE) Burda et al. [2016] developed in Heap and Aitchison [2023].
- Future work:
  - Extend MCMC past $K = 2$.

Conclusion

- A general approach for parallelising a variety of inference methods:
  - Massively Parallel MCMC
  - Massively parallel Reweighted Wake-Sleep (RWS) (Bornschein and Bengio [2015]) and Importance Weighted Autoencoders (IWAE) Burda et al. [2016] developed in Heap and Aitchison [2023].
- Future work:
  - Extend MCMC past $K = 2$.
  - Probabilistic programming language (e.g. Stan [Carpenter et al., 2017]).

References I

Daniel G. a. Smith and Johnnie Gray. opt_einsum - a python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software*, 3 (26):753, 2018. doi: 10.21105/joss.00753. URL https://doi.org/10.21105/joss.00753.

Laurence Aitchison. Tensor Monte Carlo: particle methods for the GPU era, January 2019. URL http://arxiv.org/abs/1806.08593. arXiv:1806.08593 [cs, stat].

Jörg Bornschein and Yoshua Bengio. Reweighted Wake-Sleep, April 2015. URL http://arxiv.org/abs/1406.2751. arXiv:1406.2751 [cs].

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance Weighted Autoencoders, November 2016. URL http://arxiv.org/abs/1509.00519. arXiv:1509.00519 [cs, stat].

References II

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017. Publisher: Columbia Univ., New York, NY (United States); Harvard Univ., Cambridge, MA (United States).

Sourav Chatterjee and Persi Diaconis. The sample size required in importance sampling. *The Annals of Applied Probability*, 28(2), April 2018. ISSN 1050-5164. doi: 10.1214/17-AAP1326. URL https://projecteuclid.org/journals/annals-of-applied-probability/volume-28/issue-2/The-sample-size-required-in-importance-sampling/10.1214/17-AAP1326.full.

Thomas Heap and Laurence Aitchison. Massively parallel reweighted wake-sleep. *UAI*, 2023.

References III

Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey, and Radford M. Neal. The "Wake-Sleep" Algorithm for Unsupervised Neural Networks. *Science*, 268(5214): 1158–1161, May 1995. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.7761831. URL https://www.science.org/doi/10.1126/science.7761831.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, 2014. URL http://arxiv.org/abs/1312.6114. arXiv:1312.6114 [cs, stat].

Thank you

Any questions?

Importance Weighting: Two Examples

1. Variational Autoencoders [Kingma and Welling, 2014]
   $\longrightarrow$ Importance Weighted Autoencoders [Burda et al., 2016].
   Update (θ, ϕ) by maximising $\mathcal{L}_{\text{global}}(\theta, \phi)$:

$$\log P_\theta(x) \geq \mathcal{L}_{\text{global}}(\theta, \phi) = \mathbb{E}_{Q(z|x)}[\log \mathcal{P}_{\text{global}}(z)] \tag{20}$$

Importance Weighting: Two Examples

1. Variational Autoencoders [Kingma and Welling, 2014]
   $\longrightarrow$ Importance Weighted Autoencoders [Burda et al., 2016].
   Update (θ, ϕ) by maximising $\mathcal{L}_{\text{global}}(\theta, \phi)$:

$$\log P_\theta(x) \geq \mathcal{L}_{\text{global}}(\theta, \phi) = \mathbb{E}_{Q(z|x)}[\log \mathcal{P}_{\text{global}}(z)] \tag{20}$$

2. Wake-Sleep [Hinton et al., 1995]
   $\longrightarrow$ Reweighted Wake-Sleep [Bornschein and Bengio, 2015].
   Update (θ, ϕ) with:

$$\Delta\theta = \mathbb{E}_{Q(z|x)}[\nabla_\theta \log \mathcal{P}_{\text{global}}(z)], \tag{21}$$

$$\Delta\phi = \mathbb{E}_{Q(z|x)}[\nabla_\phi(-\log \mathcal{P}_{\text{global}}(z))]. \tag{22}$$

Importance Weighting: Two Examples

1. Variational Autoencoders [Kingma and Welling, 2014]
   $\longrightarrow$ Importance Weighted Autoencoders [Burda et al., 2016].
   Update (θ, ϕ) by maximising $\mathcal{L}_{\text{global}}(\theta, \phi)$:

$$\log P_\theta(x) \geq \mathcal{L}_{\text{global}}(\theta, \phi) = \mathbb{E}_{Q(z|x)}[\log \mathcal{P}_{\text{global}}(z)] \tag{20}$$

2. Wake-Sleep [Hinton et al., 1995]
   $\longrightarrow$ Reweighted Wake-Sleep [Bornschein and Bengio, 2015].
   Update (θ, ϕ) with:

$$\Delta\theta = \mathbb{E}_{Q(z|x)}[\nabla_\theta \log \mathcal{P}_{\text{global}}(z)], \tag{21}$$

$$\Delta\phi = \mathbb{E}_{Q(z|x)}[\nabla_\phi(-\log \mathcal{P}_{\text{global}}(z))]. \tag{22}$$

Importance Weighting: Two Examples

1. Variational Autoencoders [Kingma and Welling, 2014]
   $\longrightarrow$ Importance Weighted Autoencoders [Burda et al., 2016].
   Update (θ, ϕ) by maximising $\mathcal{L}_{\text{global}}(\theta, \phi)$:

$$\log P_\theta(x) \geq \mathcal{L}_{\text{global}}(\theta, \phi) = \mathbb{E}_{Q(z|x)}[\log \mathcal{P}_{\text{global}}(z)] \tag{20}$$

2. Wake-Sleep [Hinton et al., 1995]
   $\longrightarrow$ Reweighted Wake-Sleep [Bornschein and Bengio, 2015].
   Update (θ, ϕ) with:

$$\Delta\theta = \mathbb{E}_{Q(z|x)}[\nabla_\theta \log \mathcal{P}_{\text{global}}(z)], \tag{21}$$

$$\Delta\phi = \mathbb{E}_{Q(z|x)}[\nabla_\phi(-\log \mathcal{P}_{\text{global}}(z))]. \tag{22}$$

(Recent work [Heap and Aitchison, 2023] has ‘massively parallelised’ IWAE and RWS using the same framework introduced in this talk.)

Exploiting Conditional Independencies

$$\mathcal{P}_{\text{MP}}(z) = \frac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) \qquad r_{\mathbf{k}}(z) = \frac{P(x, z^{\mathbf{k}})}{\prod_{i=1}^n Q_{\text{MP}}(z_i^k | z_j^k \text{ for } j \in \text{qa}(i))}. \tag{23}$$

Expressing $r_{\mathbf{k}}(z)$ as a product of many (low-rank) tensors,

$$r_{\mathbf{k}}(z) = f^x_{\mathbf{k}_{\text{pa}(x)}}(z) \prod_i f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z) \tag{24}$$

$$f^x_{\mathbf{k}_{\text{pa}x}}(z) = P(x | z_j^{k_j} \text{ for all } j \in \text{pa}(x)), \tag{25}$$

$$f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z) = \frac{P(z_i^{k_i} | z_j^{k_j} \text{ for all } j \in \text{pa}(i))}{Q_{\text{MP}}(z_i^{k_i} | z_j \text{ for all } j \in \text{qa}(i))}. \tag{26}$$

- $f^x_{\mathbf{k}_{\text{pa}(x)}}(z)$ is a tensor of rank $|\text{pa}(x)|$
- $f^i_{k_i, \mathbf{k}_{\text{pa}(i)}}(z)$ are tensors of rank $1 + |\text{pa}(i)|$

Modified Marginal Likelihood Estimator

We can calculate posterior moments nicely by differentiating through a modified version of our marginal liklihood estimator. Specifically, we introduce a 'source term' $e^{Jm(z^{\mathbf{k}})}$ for $J \in \mathbb{R}$:

$$\mathcal{P}_{\mathsf{MP}}^{\mathsf{exp}}(z, J) = \tfrac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) e^{Jm(z^{\mathbf{k}})}. \tag{27}$$

Note that $\mathcal{P}_{\mathsf{MP}}^{\mathsf{exp}}(z, J = 0) = \mathcal{P}_{\mathsf{MP}}(z)$, and also that the source term, like the factors that make up $r_{\mathbf{k}}(z)$, can be thought of as a tensor indexed by some subset of $\mathbf{k}$ (depending on m), hence efficient computation of this sum is still possible.

$$\frac{\partial}{\partial J}\bigg|_{J=0} \log \mathcal{P}_{\mathsf{MP}}^{\mathsf{exp}}(z, J) = \frac{\frac{\partial}{\partial J}\big|_{J=0} \mathcal{P}_{\mathsf{MP}}^{\mathsf{exp}}(z, J)}{\mathcal{P}_{\mathsf{MP}}(z)} = \frac{\frac{1}{K^n} \sum_{\mathbf{k} \in \mathcal{K}^n} r_{\mathbf{k}}(z) m(z^{\mathbf{k}})}{\mathcal{P}_{\mathsf{MP}}(z)} = m_{\mathsf{MP}}(z). \tag{28}$$