Tensorised probabilistic programming

laurence.aitchison

June 2020

Abstract

Tensor Monte Carlo introduced the possibility of dramatically improving the tightness of the evidence lower bound objective by sampling each latent variable n times and considering all K^n combinations as importance samples. However, their particular approach to augmenting the state-space was inspired by particle filters, and was poorly suited to the full probabilistic programming setting. In particular, In particular ...

Examples:

- Can we compare speed to Justin's differentiable particle filter thingy?
- Simple Heirarchical model.
- MNIST?

1 Methods

We have an underlying model with data, x, and n latent variables denoted $(z'_1, \ldots, z'_n)$. Each latent variable, z'_i lives in a set, $z'_i \in \mathcal{Z}_i$. The full collection of latent variables is denoted z',

$$z' = (z'_1, \ldots, z'_n) \in \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n = \mathcal{Z}. \tag{1}$$

The full probability density includes one factor for the probability density of the data, conditioned on the latents, $\pi_x(x; z'_{\mathrm{pa}(x)})$ and one factor for each latent variable, $\pi_i(z'_i; z'_{\mathrm{pa}(i)})$,

$$\pi(x, z') = \pi_x(x; z'_{\mathrm{pa}(x)}) \prod_{i=1}^{n} \pi_i(z'_i; z'_{\mathrm{pa}(i)}). \tag{2}$$

where $\mathrm{pa}\,(i)$ is the indicies of all parents of z_i, and $z'_{\mathrm{pa}(i)}$ is the collection of all parent latent variables.

We embed this model in an extended state space. The full extended model consists of data, x, indices, k, and latent variables, z, and factorises as,

$$\mathrm{P}\,(x, k, z) = \mathrm{P}\,(x, z|k)\,\mathrm{P}\,(k)\,. \tag{3}$$

There are n latent variables, as in the underlying model, and each latent variable is copied K times to form an extended state space. We denote a single copy of a single latent variable as z_i^κ, where $i \in \{1, \ldots, n\}$ indexes the latent variable and $\kappa \in \{1, \ldots, K\}$ indexes the copy. That single latent variable, z_i^κ, is a random variable with support $\mathcal{Z}_i$ (analogous to z_i'). The tuple of all K copies of a single latent variable is denoted z_i,

$$z_i = (z_i^1, \ldots, z_i^K) \in \mathcal{Z}_i^K \tag{4}$$

and the tuple of all copies of all latent variables is denoted z,

$$z = (z_1, \ldots, z_n) \in \mathcal{Z}_1^K \times \cdots \times \mathcal{Z}_n^K = \mathcal{Z}^K. \tag{5}$$

We also augment the state-space with a set of uniformly distributed random indices, $k_i \in \mathcal{K} = \{1, \ldots, K\}$, one for each latent variable,

$$\mathrm{P}\left(k_i\right) = \mathrm{UniformDiscrete}\left(1, K\right). \tag{6}$$

These pick out one copy for each latent variable, and the tuple containing all random indices is denoted k,

$$k = (k_1, \ldots, k_n) \in \mathcal{K}^n. \tag{7}$$

Now, we can write the probability density,

$$\mathrm{P}\left(x, z|k\right) = \pi\left(x, z^k\right) \prod_i q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k}). \tag{8}$$

There's alot of notation here, so lets unpack it. First, z^k is a tuple containing a single copy of each latent variable, indexed by k,

$$z^k = (z_1^{k_1}, \ldots, z_n^{k_n}) \in \mathcal{Z}. \tag{9}$$

Next, $z_i^{/k}$ is the other, non-indexed copies,

$$z_i^{/k_i} = (z_i^1, \ldots, z_i^{k_i - 1}, z_i^{k_i + 1}, \ldots, z_i^K) \in \mathcal{Z}_i^{K-1}. \tag{10}$$

and a tuple containing all non-indexed copies of all latents is,

$$z^{/k} = (z_1^{/k_1}, \ldots, z_n^{/k_n}) \in \mathcal{Z}^{K-1}, \tag{11}$$

i.e. all copies of the ith latent variable except $z_i^{k_i}$. Now we can understand the implied generative process:

- First, generate the indices, k_i from uniform discrete distributions.
- Next, generate the indexed latent variables, z^k, and data, x, from the underlying model with density π.
- Next, generate the other non-indexed latent variables, $z^{/k}$.

Now comes the question of what is the most general distribution over the non-indexed latent variables, $q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k})$. We generate z^k and x first, so clearly $z_i^{/k_i}$ can depend on them. But we can also introduce conditional dependencies between the non-indexed latent variables, $z_i^{/k_i}$. The structure of these conditional dependencies is expressed by $z_{\mathrm{qa}(i)}^{/k}$, which selects out those latent variables that are direct parents of $z_i^{k_i}$ under q. Importantly, this conditional dependency structure can be different from that in the underlying model, which is why the parents for the distribution over the non-indexed latent variables is denoted $\mathrm{qa}\,(i)$, rather than $\mathrm{pa}\,(i)$ for the underlying model.

1.1 MCMC

In MCMC, we typically make only a single proposal, then compare the probability of the original sample and proposal under the posterior. This is very problematic because in larger state-spaces, it becomes increasingly difficult to make a proposal for all latent variables jointly that has high probability under the posterior. While we can make multiple proposals (e.g. 20 proposals) [1], this doesn't make a material difference as we move to high-dimensions. Here, we introduce a tensorised MCMC method that makes exponentially proposals, n^K, and is therefore able to mix quickly and efficiently even in high-dimensional state-spaces.

In tensorised MCMC, the idea is to use the extended state-space construction described in the first Section. One iteration then involves:

- Start with random variables, k, z^k and $z^{/k}$ from the previous iteration.
- Gibbs sample for $z^{/k}$ by sampling from the density, $\prod_i q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k})$, which acts as a proposal.
- Gibbs sample k, with fixed z.

This is an extension of the parallel Metropolis Hastings [1], where we introduce structured conditional independencies in the latent variables, and consider one index for each latent variable.

Resampling $z^{/k}$ is straightforward. The only difficult part is in sampling k.

$$\mathrm{P}\,(k|z, x) \propto \pi\left(x, z^k\right) \prod_i q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k}). \tag{12}$$

To obtain an efficient MCMC algorithm, we need a proposal that simplifies this conditional. One approach would be to generalise the notion of a symmetric proposal to multiple variables, following [1]. In particular, we consider a proposal which is independent of other latent variables,

$$q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k}) = q(z_i^{/k_i}; z_i^{k_i}) \tag{13}$$

We restrict this proposal to be symmetric in the sense that its probability density should be the same for any choice of k_i and $k_i' \neq k_i$,

$$q(z_i^{/k_i}; z_i^{k_i}) = q(z_i^{/k_i'}; z_i^{k_i'}). \tag{14}$$

Thus, the conditional distribution for Gibbs sampling becomes proportional to the probability density for the underlying model,

$$\mathrm{P}\left(k|z,x\right) \propto \pi\left(x, z^k\right). \tag{15}$$

What does this distribution look like? Well, remember that the underlying model factorises,

$$\pi(x, z^k) = \pi_x(x; z^k) \prod_{i=1}^{n} \pi_i(z_i^k; z_{\mathrm{pa}(i)}^k). \tag{16}$$

and that z and x are fixed, and we are interested only in the k dependence. This is a product of factors, $\pi_x(x; z^k)$ and $\pi_i(z_i^k; z_{\mathrm{pa}(i)}^k)$. Each factor depends only on a subset of the k_i's. We could thus write each factor as a tensor,

$$f_i^{k_i, k_{\mathrm{pa}(i)}} = \pi_i(z_i^k; z_{\mathrm{pa}(i)}^k) \qquad f_i \in \mathbb{R}^{K^{1+|\mathrm{pa}(i)|}} \tag{17}$$

$$f_x^{k_{\mathrm{pa}(x)}} = \pi_x(z_{\mathrm{pa}(x)}^k) \qquad f_x \in \mathbb{R}^{K^{|\mathrm{pa}(x)|}} \tag{18}$$

where i and x is a label indicating which tensor we are referring to and $|\mathrm{pa}\left(i\right)|$ is the number of parents of the ith latent variable. Thus, $k_i, k_{\mathrm{pa}(i)}$ and $k_{\mathrm{pa}(x)}$ are a list of elements of k, which act as indices into f_i and f_x respectively. The conditional distribution for Gibbs sampling can thus be written as a product of $n+1$ factors,

$$\mathrm{P}\left(k|z,x\right) \propto \pi(x, z^k) = f_x^{k_{\mathrm{pa}(x)}} \prod_{i=1}^{n} f_i^{k_i, k_{\mathrm{pa}(i)}}. \tag{19}$$

And this is a discrete factor-graph, which can be treated using standard message-passing techniques.

1.1.1 Symmetric MCMC proposals with $K > 2$

Draft: not ready for consumption!

Unsurprisingly, it is non-trivial to build symmetric proposals for $K > 2$. Calderhead [1] had complex proposals involving auxiliary variables that involved are difficult to build into a general probabilistic programming framework. Instead, we consider other approaches to building symmetric proposals. In particular, we would like $\mathrm{P}\left(Z^{-K}|Z^K\right)$ to be independent of K. This is straightforward if we set $\kappa = 2$, and use a factorised, symmetric proposal,

$$\mathrm{P}\left(Z^{-K}|Z^K\right) = \prod_i \mathrm{P}\left(Z_i^{-K_i}|Z_i^{K_i}\right) \tag{20}$$

$$\mathrm{P}\left(Z_i^{-K_i} = z_i'|Z_i^{K_i} = z_i, X\right) = \mathrm{P}\left(Z_i^{-K_i} = z_i|Z^{K_i} = z_i', X\right) \tag{21}$$

It should be possible to obtain analogues of symmetric proposals for higher values of K by considering spatial structure. For instance, if the individual proposals here are symmetric, and we generate the proposals as a chain,

$$Z_i^1 \leftarrow \cdots \leftarrow Z_i^{K_i-1} \leftarrow Z_i^{K_i} \rightarrow Z_i^{K_i+1} \rightarrow \cdots \rightarrow Z_i^n. \tag{22}$$

As this requires many tensor operations, it may be possible to put samples on a hyper-cube.

With a symmetric proposal, we have,

$$\mathrm{P}\left(K|Z^K, Z^{-K}\right) \propto \mathrm{P}\left(K\right)\mathrm{P}\left(Z^K\right)\mathrm{P}\left(X|Z_{\mathrm{pa}(X)}^{K_{\mathrm{pa}(X)}}\right) \tag{23}$$

$$= \mathrm{P}\left(K\right)\mathrm{P}\left(X|Z_{\mathrm{pa}(X)}^{K_{\mathrm{pa}(X)}}\right)\prod_i \mathrm{P}\left(Z_i|Z_{\mathrm{pa}(i)}^{K_{\mathrm{pa}(i)}}\right) \tag{24}$$

Importantly, the right-hand-side can be understood as a factor graph for K.

1.2 VI

Just as the effectiveness of MCMC is determined by the quality of the proposal, the effectiveness of VI is determined by the quality of the approximate posterior. Training a very accurate approximate posterior is difficult, and not always possible (e.g. due to the presence of complex conditional independencies). An alternative approach to improving the variational bound is to draw multiple samples from the approximate posterior and do importance sampling. However, past work on this topic (IWAE) drew a small number of samples (e.g. 20) from the full joint state-space. Here, we seek an approach that is equivalent to drawing exponentially many samples, K^n, which should dramatically improve the quality of the variational bound.

We construct a variational approximate posterior just over z from the extended state-space in the first section. To achieve this, we marginalise k out of the generative model, then compute the usual ELBO (albeit in an extended state-space),

$$\mathcal{L} = \mathrm{E}_{\mathrm{Q}(z|x)}\left[\log \frac{\sum_k \mathrm{P}\left(z,x,k\right)}{\mathrm{Q}\left(z|x\right)}\right] = \mathrm{E}_{\mathrm{Q}(z|x)}\left[\log \frac{\sum_{k_1,\ldots,k_n} \mathrm{P}\left(z,x,k_1,\ldots,k_n\right)}{\mathrm{Q}\left(z|x\right)}\right]. \tag{25}$$

[WHY DOES THIS LOOK LIKE IWAE] where the sums over the k_i's run from 1 to K. The original TMC paper showed that such a marginalisation is closely related to using an importance weighted bound (e.g. as in IWAE), but with an exponential number of importance samples. This exponential number of importance samples enables a much tighter variational bound, and much more accurate posteriors.

We consider an approximate posterior which independently samples each "copy" of the full latent space, indexed κ,

$$\mathrm{Q}\left(z^{\kappa}|x\right) = \prod_{i=1}^{n} \mathrm{Q}\left(z_i^{\kappa}|z_{\mathrm{qa}(i)}^{\kappa}, x\right) \tag{26}$$

where z^{κ} is the κth copy of each latent variable,

$$z^{\kappa} = (z_1^{\kappa}, \ldots, z_n^{\kappa}) \in \mathcal{Z}, \tag{27}$$

and where $\mathrm{qa}\left(i\right)$ denotes the parents of variable i under the approximate posterior. Overall, we have,

$$\mathrm{Q}\left(z|x\right) = \prod_{\kappa \in \mathcal{K}} \mathrm{Q}\left(z^{\kappa}|x\right) = \prod_{\kappa \in \mathcal{K}} \prod_{i=1}^{n} \mathrm{Q}\left(z_i^{\kappa}|z_{\mathrm{qa}(i)}^{\kappa}, x\right) \tag{28}$$

where, remember, $\mathcal{K} = \{1, \ldots, K\}$. Note: it is this choice of dependency structure in the approximate posterior that distinguishes our approach from past work in tensor Monte-Carlo and particle filtering. In particular, in all of these approaches, the z_i^{κ} depends on all copies of the previous latent variables, $z_{\mathrm{qa}(i)}$ For instance, the proposal might be a mixture, with one component corresponding to each copy of the previous latent variables.

Now, we need to pick $q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k})$ in the generative model (Eq. 8) to ensure as much cancellation as possible in the ELBO (Eq. 25). We choose,

$$q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k}) = \prod_{\kappa \in \mathcal{K}/k_i} \mathrm{Q}\left(z_i^{\kappa}|z_{\mathrm{qa}(i)}^{\kappa}, x\right). \tag{29}$$

and where $\mathcal{K}/k_i$ is all indicies except k_i,

$$\mathcal{K}/k_i = \{1, \ldots, k_i - 1, k_i + 1, \ldots, K\}. \tag{30}$$

Importantly, $z_{\mathrm{qa}(i)}^{\kappa}$ is always available in $q(z_i^{/k_i}; x, z^k, z_{\mathrm{qa}(i)}^{/k})$. Consider one parent, $j \in \mathrm{qa}\left(i\right)$. If $\kappa = k_j$, then z_j^{κ} is included in z^k. Alternatively, if $\kappa \neq k_j$, then z_j^{κ} is included in $z_{\mathrm{qa}(i)}^{/k} = z_j^{/k}$.

Substituting this choice of generative model for the extended state space, and approximate posterior into the ELBO, we get,

$$\sum_k \frac{\mathrm{P}\left(z, k, x\right)}{\mathrm{Q}\left(z|x\right)} = \sum_k \mathrm{P}\left(k\right) \frac{\pi(x, z^k) \prod_{i=1}^{n} \prod_{\kappa \in \mathcal{K}/k_i} \mathrm{Q}\left(z_i^{\kappa}|z_{\mathrm{qa}(i)}^{\kappa}, x\right)}{\prod_{i=1}^{n} \prod_{\kappa \in \mathcal{K}} \mathrm{Q}\left(z_i^{\kappa}|z_{\mathrm{qa}(i)}^{\kappa}, x\right)} \tag{31}$$

Now, the $\mathrm{Q}\left(z_i^{\kappa}|z_{\mathrm{qa}(i)}^{\kappa}, x\right)$ terms for all $\kappa \in \mathcal{K}/k_i$ cancel,

$$\sum_k \frac{\mathrm{P}\left(z, k, x\right)}{\mathrm{Q}\left(z|x\right)} = \sum_k \mathrm{P}\left(k\right) \frac{\pi(x, z^k)}{\prod_{i=1}^{n} \mathrm{Q}\left(z_i^{k_i}|z_{\mathrm{qa}(i)}^{k_i}, x\right)}. \tag{32}$$

Remembering that $\pi(x, z^k)$ could be written as a factor graph, we can write our estimator as,

$$\sum_{k \in \mathcal{K}^n} \frac{\mathrm{P}\left(z, k, x\right)}{\mathrm{Q}\left(z|x\right)} = \sum_{k_1, \ldots, k_n} \mathrm{P}\left(k\right) f_x^{k_{\mathrm{pa}(x)}} \prod_{i=1}^{n} \frac{f_i^{k_i, k_{\mathrm{pa}(i)}}}{\mathrm{Q}\left(z_i^{k_i} | z_{\mathrm{qa}(i)}^{k_i}, x\right)}. \tag{33}$$

this factor graph has the same structure as the underlying graphical model.

1.3 Advantages of this construction over TMC/particle filtering

This approach has several advantages over TMC/particle filtering.

First, this approach has minimal asymptotic complexity. In particular, the factors, which dominate asymptotic compute and memory costs have size $K^{1+|\mathrm{pa}(i)|}$. Our particular choice of approximate posterior ensures that the asymptotic costs depend only on the structure of the prior, through $\mathrm{pa}\left(i\right)$; they do not depend on the structure of the approximate posterior, expressed in $\mathrm{qa}\left(i\right)$. We can therefore have arbitrarily complex approximate posteriors, without increasing the computational complexity. This only arises under our particular choice of approximate posterior. Other methods such as particle filtering and the original TMC paper have an approximate posterior that depends on all particles of all previous latent variables, and the usual choice is to use a mixture over all parent particles. This is fine if there is only one parent variable, in which case this mixture has K components. However, the number of components grows exponentially with the number of parents, $K^{|\mathrm{qa}(i)|}$. This exponential growth can be prohibitive in many settings. For instance, consider doing inference over latent variables, $z = (\theta, y_1, \ldots, y_N)$ (there is one θ for all datapoints, and one y_i associated with each datapoint), with the following generative model and approximate posterior.

$$\mathrm{P}\left(x_1, \ldots, x_N, y_1, \ldots, y_N, \theta\right) = \mathrm{P}\left(\theta\right) \prod_{i=1}^{n-1} \mathrm{P}\left(y_i|\theta\right) \mathrm{P}\left(x_i|y_i\right) \tag{34}$$

$$\mathrm{Q}\left(y_1, \ldots, y_N, \theta | x_1, \ldots, x_N\right) = \mathrm{Q}\left(\theta | y_1, \ldots, y_{n-1}\right) \prod_{i=1}^{n-1} \mathrm{Q}\left(y_i|x\right). \tag{35}$$

Note that each term in the generative model has only one or two latent variables ($\mathrm{P}\left(\theta\right)$ and $\mathrm{P}\left(x_i|y_i\right)$ have one latent variable, and $\mathrm{P}\left(y_i|\theta\right)$ has two). As such, if we implemented this model in our framework, the largest factor would be K^2, which is eminently tractable. But problems arise if we use a mixture proposal for $\mathrm{Q}\left(\theta|y_1, \ldots, y_{n-1}\right)$, as in past work from TMC and particle filtering. The issue is that θ depends on N latent variables, so there are K^N mixture components, which is intractable in all but the smallest models.

Second, our approach offers increased parallelisation as, when sampling from the approximate posterior we simply sample K independent copies from the full

joint latent space. In contrast, sampling is much more complicated under the mixture proposals in TMC/particle filtering, as z_i^κ might depend on any copy of the previous latent variables. These complex dependencies impede effective GPU parallelisation.

Third, our method is straightforwardly differentiable using autodiff build in to e.g. PyTorch. In contrast, high dimensional mixture proposals are difficult to differentiate (as they cannot easily be reparameterised; see Justin's paper).

1.4 Marginal inference

TPP not only provides a tighter lower bound for estimating the likelihood, it can also benefit the inference over the marginal distribution of latent variables, especially those with multiple modes.

2 Implementation notes: don't look at this! Its not relevant to the underlying algorithm

2.1 Summing and sampling in factor graphs

Take the case of MCMC. Here, we start with the full joint, written as a product of factors. We progressively sum out each variable, (omitting conditioning on Z),

$$\mathrm{P}\left(K_1, K_2, K_3, K_4\right) = \sum_{K_5} \mathrm{P}\left(K_1, K_2, K_3, K_4, K_5\right) \tag{36}$$

$$\mathrm{P}\left(K_1, K_2, K_3\right) = \sum_{K_4} \mathrm{P}\left(K_1, K_2, K_3, K_4\right) \tag{37}$$

$$\mathrm{P}\left(K_1, K_2\right) = \sum_{K_3} \mathrm{P}\left(K_1, K_2, K_3\right) \tag{38}$$

$$\mathrm{P}\left(K_1\right) = \sum_{K_2} \mathrm{P}\left(K_1, K_2\right) \tag{39}$$

Each joint distribution in this list is represented as a product of factors. When we sum out a factor, we:

- Remove all tensors/factors from the list that depend on that variable.
- Take the product of all factors, to make a single big factor.
- Sum out k_i for that variable.
- Put the single resulting tensor back in the list.

When we do a sum, we multiply all factors in the list that depend on that variable, sum it out, and put it back in the list.

Sampling is now straightforward. We begin by,

$$K_1 \sim \mathrm{P}\left(K_1\right) \tag{40}$$
$$K_2 \sim \mathrm{P}\left(K_2|K_1\right) \propto \mathrm{P}\left(K_1, K_2\right) \tag{41}$$
$$K_3 \sim \mathrm{P}\left(K_3|K_1, K_2\right) \propto \mathrm{P}\left(K_1, K_2, K_3\right) \tag{42}$$
$$K_4 \sim \mathrm{P}\left(K_4|K_1, K_2, K_3\right) \propto \mathrm{P}\left(K_1, K_2, K_3, K_4\right) \tag{43}$$
$$K_5 \sim \mathrm{P}\left(K_4|K_1, K_2, K_3, K_4\right) \propto \mathrm{P}\left(K_1, K_2, K_3, K_4, K_5\right) \tag{44}$$

Importantly, sampling these distributions is easy, because we have all the joints on the right-hand-side, represented as lists of factors. Even if that list is long, we can still sample easily because we have a fixed sample for all the other K_i's in that list.

2.2 IID sampling in plates

We keep the list of factors as a flat list. When we combine factors, we can only do so if:

- They have the same plates
- There are no lower-layer plates

When there is only one factor left under a plate, we can sum across those dimensions.

This can be achieved, and should be efficient if we just sum out plates in reverse order.

Algorithm:

- Extract plate order, by looking for the first tensor with that plate.
- Sum out plates in reverse order of definition.
  - Find all tensors in plate, remove them from underlying list of tensors, and add them to a new list.
  - Sum out all sample indexes within the plate.
  - Sum out the plate
  - Put the resulting tensor back in the original list.

References

[1] Ben Calderhead. A general construction for parallelizing metropolis-hastings algorithms. *Proceedings of the National Academy of Sciences*, 111(49):17408–17413, 2014.