RESCIENCE C

[Re] Tensor Monte Carlo: Particle Methods for the GPU Era

Oskar Kviman[1, ID], Linus Nilsson[1, ID], and Martin Larsson[1, ID]
[1]KTH Royal Institute of Technology, Stockholm, Sweden

1 Introduction

Variational autoencoders (VAE), first introduced in the works of [1], sparked a trend in designing generative models in order to approximate the intractable posterior distribution. Many recent papers have provided ingenious schemes for improving upon VAE, among some ([2, 3, 4, 5]), by achieving tighter log-likelihood bounds on the marginal likelihood (explained in greater detail below). The original bottom-up and top-down architecture has been experimented with ([4]), as well as employing chains of transformations on an, in VAE, assumed simplistic prior distribution ([3, 5]). The importance weighted variational autoencoder (IWAE; [2]) utilized averaging over multiple samples, as opposed to VAE's single-sample objective, to tighten the mentioned bound while being able to model a richer latent space – in effect, this multi-sample scheme allows for a more complex approximate posterior. In light of IWAE, tensor Monte-Carlo ([6]; TMC) was recently proposed as an attempt to improve upon IWAE by sampling exponentially many importance samples. For each of the n latent variables in the TMC, K samples are drawn yielding K^n marginal log-likelihood evaluations. Averaging over this large number of samples might appear computationally impossible, but via clever tensor products computed in parallel, the TMC is approximately as fast as the less importance sample exhausting IWAE.

In this work, we reproduce what we believe are the most important results presented in the Tensor Monte Carlo paper ([6]), where we also provide our reimplementation code. The original results in the TMC paper were attained via a PyTorch ([7]) implementation.[1] In an attempt to ease understanding for those unfamiliar with PyTorch, we contribute with a TensorFlow[2] ([8]) implementation. Early on in our work, a connection was established with the author in order to bring our reproducibility work to their attention, as well as ensuring that we progress by clearing potential ambiguities. Due to resource and time constraints, we chose to reproduce those results that, in our meaning, appeared most informative and fundamental in the TMC paper. Additionally, as we found the TMC architecture non-trivial to understand, we aim to ease understanding for future users by complementing the textual description of the model with an algorithmic description in Alg. 1 and a depiction of the model in Fig. 4 (figure in Appendix B). Furthermore, we supplement the original paper by visualizing the TMC's reconstruction and clustering capabilities (Appendix C and D, respectively), while contrasting them to the capabilities of the baseline, IWAE.

---

[1]Code is available at: https://github.com/anonymous-78913/tmc-anon.
[2]Code is available at: https://github.com/LinuNils/TMC_reproduced.

2 Background in Variational Autoencoders

As in traditional variational inference (VI), the VAE specifies a proposal distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ for approximating an intractable posterior distribution $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. The VAE is comprised of two parts, where the first part, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, is often referred to as the representation (recognition or encoder) model ([1, 6, 2]), and it learns a mapping from the input space $\mathcal{X}$ to the parameter set $\phi = \{\mu, \sigma^2\}$. As the aim of generative modeling is to learn the joint probability distribution, using the notation aligned with this paper, $p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})$ ([1, 5, 9]), the second part, conveniently called the generative model, tries to achieve just this. In VAE, the two parameter sets, ϕ and θ, are optimized in order to tighten the lower bound for the log-likelihood

$$\log p_\theta(\boldsymbol{x}) = \log \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] \geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] = \mathcal{L}_{\text{VAE}}(\boldsymbol{x}), \qquad (1)$$

where the lower bound is achieved via Jensen's Inequality ([1]). Here, we only consider one latent sample, $\boldsymbol{z}$, per given data point $\boldsymbol{x}$, i.e. $\mathcal{L}_{\text{VAE}}$ is a single-sample objective. Since we sample from the approximate posterior (proposal distribution) q_ϕ, we want to choose this distribution such that we can operate on in a convenient manner. Therefore, we often resort to a Gaussian distribution ([2, 1, 9]. Since this is a somewhat strong assumption of the true distribution ([5, 3]) modern VAE's try to make the model more expressive, tightening the bound in Eq. (1).
One such modern VAE is IWAE ([2]), which employs a multi-sampling objective

$$\log p_\theta(\boldsymbol{x}) = \log \mathbb{E}_{q_\phi(\boldsymbol{z}^1|\boldsymbol{x})...q_\phi(\boldsymbol{z}^k|\boldsymbol{x})}\left[\frac{1}{k}\sum_i \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^i)}{q_\phi(\boldsymbol{z}^i|\boldsymbol{x})}\right] \geq \mathbb{E}_{q_\phi(\boldsymbol{z}^1|\boldsymbol{x})...q_\phi(\boldsymbol{z}^k|\boldsymbol{x})}\left[\log \frac{1}{k}\sum_i \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^i)}{q_\phi(\boldsymbol{z}^i|\boldsymbol{x})}\right]$$
$$= \mathcal{L}_{\text{IWAE}}^k(\boldsymbol{x}),$$
$$(2)$$

where k is the number of samples drawn per data point. In [2], the authors state two fundamental benefits of this approach:

- The authors prove that $\log p_\theta(\boldsymbol{x}) \geq \mathcal{L}_{k+1} \geq \mathcal{L}_k$.

- When training using gradient descent, how to update the network parameters is based on importance weighting between the k $\boldsymbol{z}$'s marginal log-likelihoods. I.e. the learning rule puts more emphasis on samples with larger marginal log-likelihoods.

3 Analysis of the Tensor Monte-Carlo Algorithm

In this section, we aim to provide a sufficient background of the TMC algorithm in order for the reader to be able to follow the reproduced results. Additionally, we outline which aspects of the algorithm and results we intend to reproduce. We do so in order to restrict what work in the TMC paper we need to explain. For completeness, we next briefly touch upon most of the novelties proposed by Aitchison.
In the TMC paper, Aitchison extends IWAE's multi-sample objective by considering exponentially many important samples, barely without increasing the computational cost compared to IWAE. Aitchison in essence proposes two different models, one where the proposal distribution is factorised, and the other one with non-factorised proposals. Furthermore, two existing variance reduction techniques, STL ([10]) and DReGS ([11]), were employed, actually worsening performance for the TMC, according to the author.

3.1 What we reproduce

Although there are many potential results to be reproduced, due to time and resource constraints we limit our work to reproducing the following

- For all tests we assume a non-factorized proposal distributions. This approach creates a recognition model in the same manner as the baseline algorithm, IWAE, was originally proposed.

- We omit the use of the variance reduction techniques STL and DReGS. The results in the TMC paper were presented with and without these techniques, why we may still compare our results with those presented in the TMC paper. Given our trade-off between time and quantitative testing, we argue that this approach is most informative if no prior knowledge of the techniques is assumed. Additionally, the best performing models in the TMC paper did not include these variance reduction techniques.

- The baseline, an IWAE model, has been reproduced from scratch in TensorFlow.

- An ablation study is provided, where we remove the intermediate layers in the recognition and generative models, effectively attaining one of the originally proposed number of layers for the IWAE ([2]). Since the effects of TMC becomes apparent only when we have intermediate layers ([6]), we expect the IWAE and TMC to produce approximately the same results.

- We provide a non-exhaustive hyper-parameter search using grid-search over the number of samples (k; particles) and the learning rate.

As stated above, we provide the complete reproducibility code publicly on GitHub.

3.2 Tensor Monte-Carlo

For reproducibility, we begin this section with providing an algorithmic description of the TMC, see Alg. 1, which we next complement with some important details.

As seen in Alg. 1, the TMC evaluates exponentially many importance samples as the IWAE ([6, 2]). Doing this, in turns, gives rise to a new objective function. In order to average over all different combinations of marginal log-likelihoods, Aitchison ([6]), defines the new marginal likelihood estimator as

$$\mathcal{P}_{\text{TMC}} = \frac{1}{\prod_{i=1}^{} K_i} \sum_{k_1, k_2, \ldots, k_{L_q}} \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z}_1^{k_1}) \prod_j p_\theta(\boldsymbol{z}_j^{k_j}|\boldsymbol{z}_{j+1}^{k_{j+1}})}{\prod_i q_\theta(\boldsymbol{z}_i^{k_i}|\boldsymbol{z}_{k-1}^{k_{i-1}})}, \tag{3}$$

which yields the following multi-sample objective

$$\log p_\theta(\boldsymbol{x}) \geq \mathbb{E}_{q_\theta}\left[\mathcal{P}_{\text{TMC}}\right] = \mathcal{L}_{TMC}(\frown). \tag{4}$$

Note that the estimator in Eq. (3) only applies when assuming a non-factorized proposal. Also note, in contrast to the notations used when describing IWAE, k_l here is a set with cardinality K and not the number of samples per latent $\boldsymbol{z}$. To compute the tensor inner-product in a numerically stable way, the author provides a method referred to as logmmexp ([6]; see Appendix A).

4 Reproducibility

Please note, that in the above section we provided a list of what we aim to reproduce. Here, we go through our experimental methodology in greater detail.

Algorithm 1: Non-factorised Tensor Monte-Carlo (TMC) algorithm. d here is the dimension of a data point, b is the batch size, and the number of layers in the recognition model is L_q.

Initialize $\boldsymbol{\theta}$
while *Not converged* **do**

    $\boldsymbol{x} \leftarrow$ Randomly draw a data minibatch, $\boldsymbol{z}_0$, of size b
    for $l \in \{1, ..., L_q\}$ **do**

        $\boldsymbol{h}_l^{k_l} \leftarrow \text{MLP}(\boldsymbol{z}_{l-1}^{k_{l-1}})$
        $\boldsymbol{\mu}_l^{k_l} \leftarrow \text{Linear}(\boldsymbol{h}_l^{k_l})$
        $\boldsymbol{\rho}_l^{k_l} \leftarrow \text{SoftPlus}(\text{Linear}(\boldsymbol{h}_l))$
        $\boldsymbol{\epsilon}_l^{k_l} \leftarrow \mathcal{N}(0,1)$ `// Reparameterization trick, ` $b \times K \times d$`-tensor of`
        `univariate gaussians`
        $\boldsymbol{z}_l^{k_l} \leftarrow \boldsymbol{\mu}_l + \boldsymbol{\rho}_l \boldsymbol{\epsilon}_l^{k_l}$
        for $i \in \{1, ..., K\}$ **do**

            for $j \in \{1, ..., K\}$ **do**
                Compute $\log q_{\boldsymbol{\theta}}(\boldsymbol{z}_l^j | \boldsymbol{z}_{l-1}^i) \leftarrow \log q_{\boldsymbol{\theta}}(\boldsymbol{z}_l^j | \boldsymbol{\mu}_l^i, \boldsymbol{\rho}_l^i)$;

        Store the log-likelihoods and the sampled latents

    Compute and store $\log p_{\boldsymbol{\theta}}(\boldsymbol{z}_{L_q}^{k_{L_q}})$
    for $\boldsymbol{z}_l^{k_l} \in \{\boldsymbol{z}_{L_q}^{k_{L_q}}, ..., \boldsymbol{z}_2^{k_2}\}$ **do**

        $\boldsymbol{h}_{l-1}^{k_{l-1}} \leftarrow \text{MLP}(\boldsymbol{z}_l^{k_l})$
        $\boldsymbol{\mu}_{l-1}^{k_{l-1}} \leftarrow \text{Linear}(\boldsymbol{h}_{l-1}^{k_{l-1}})$
        $\boldsymbol{\rho}_{l-1}^{k_{l-1}} \leftarrow \text{SoftPlus}(\text{Linear}(\boldsymbol{h}_{l-1}^{k_{l-1}}))$
        for $i \in \{1, ..., K\}$ **do**

            for $j \in \{1, ..., K\}$ **do**
                compute $\log p_{\boldsymbol{\theta}}(\boldsymbol{z}_{l+1}^j | \boldsymbol{z}_l^i) \leftarrow log \mathcal{N}(\boldsymbol{z}_{l+1}^j | \boldsymbol{\mu}_{l+1}^i | \boldsymbol{\mu}_{l+1}^i(\boldsymbol{\rho}_{l+1}^i))$

        Store the log-likelihoods

    $\boldsymbol{\mu}_0^{k_0} \leftarrow Linear(MLP(\boldsymbol{z}_1^{k_1}))$
    Compute and store $\log p_{\boldsymbol{\theta}}(\boldsymbol{x} | \boldsymbol{z}^{k_1}) \leftarrow \log \mathcal{B}(\boldsymbol{x} | \mu_0^{k_0})$
    update $\boldsymbol{\theta}$ using $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{TMC}(\boldsymbol{x})$

4.1 Dataset

The TMC paper uses the MNIST ([12]) handwritten digit database to evaluate the performance of TMC compared to IWAE. Although the author of the TMC paper also conducts an enlightening toy experiment where the true marginal likelihood is known, we restrain our work to the MNIST dataset as these results are arguably most informative when comparing with existing models in the literature. The dataset was downloaded and pre-processed via Keras ([13]), while we normalized all pixels by division with 255. No explicit scheme of how the training proceeded was presented, but the following was done, justified by discussion with the author: similar to that presented in [2], the model was exposed to all the training data, presented in randomly drawn mini-batches. Next the same was done for all the test data, concluding an epoch.

Comments on reproducibility −

- It seems crucial to know the exact training and test scheme, in order to reproduce the results. Perhaps this is a standardized approach when training VAE's, but we were unaware of such a standard. We suggest that this should be added to the paper for clarity.

4.2 Implementation details

In order to make qualitative comparisons between the proposed TMC model and the benchmark model, we implemented the algorithm together with its baselines, in TensorFlow. The instructions for running our code are located in our git repository. Each model is self-contained in a individual file.

Considering only the non-factorized proposals evaluated on the MNIST dataset, the author employed two models coined small and large. In common for both models in the TMC paper, is the number of hidden layers and dimensionality of the smallest latent space (furthest from the data), i.e. 4 units. Inspired by the works in [4], there are five stochastic layers in both the representation and generative model. In between each stochastic layer is a two-layered perceptron (deterministic layers) with varying numbers of hidden units. For the small model's recognition model, each stochastic layer, starting from the final layer, had 4, 8, 16, 32, and 64 stochastic units. The two layers in the multi-layer perceptrons (MLPs) both had twice as many units as their preceding stochastic layer. Here, the same architecture goes for the generative model.

Concerning the large recognition model, the two layers in the MLPs both had 8 times as many units as their preceding stochastic layer, i.e. 32, 64, 128, 256 and 512 units respectively. Under the large model, the generative model architecture remained the same as when using the small model. We used leaky-relu non-linearities everywhere except when we calculated the standard deviation, for which we used $0.01 + \text{softplus}(x)$ as proposed by [6].

In our ablation study, we utilized the single-stochastic layer architecture proposed in [2], i.e. the recognition and generative model have two deterministic layers each, with 200 units per layer. This should indeed regress the TMC to the IWAE model (as should be clear from Alg. 1). We kept the dimension of the latent space as 4.

For all the above experiments parameter optimization was computed via Adam ([14]) with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-4}$ and a mini-batch size of 128, all as used in the TMC paper. Apart from evaluating $K = 20$, the only choice of number of samples in the TMC paper for the MNIST dataset evaluation, we perform a hyper-parameter search over a small set of K's and learning rates. A deficit in our work is that we did not use weight normalization ([15]), in contrast to the author. In the TMC paper, weight normalization is recommended for numerical stability. We only encountered numerical overflow for some specific choices in our hyper-parameter search (discussed in 5. Results). In the TMC model, we used float32 bit precision at all points except for the tensor inner products where we used float64 precision, as it caused numerical instability. We used the TensorFlow standard weight initialization, i.e. Glorot-uniform initialization or Xavier-initialization ([16]). From what we gathered this also seems to be the default in PyTorch which is what the TMC paper used.

Comments on reproducibility −

- Despite the TMC paper being very well-written, when implementing the proposed models we experienced difficulties in grasping the flow of latent variables, $\boldsymbol{z}$, in the network. Especially that, indeed, there is no sampling step subsequent the stochastic layers in the generative model. We believe this confusion to have arisen as the notations for the latents do not clearly express them being sampled from the recognition model, i.e. when used for marginal log-likelihood evaluation in the generative model. Perhaps it was our limited prior knowledge about the IWAE algorithm that led up to this ambiguity, and this might not cause reproducibility issues for others. Nevertheless, we contribute with an algorithmic description (Alg. 1) and a transformation of the original textual description into a figure (Fig. 4).

- The plots presented in the TMC paper are informative in the context of comparison between the baseline and the presented model. For reproducibility reasons,

on the other hand, we would reason that a table presenting the final negative log-likelihood scores would ease comparison of results. Even plots with more tics on the objective value-axes might be beneficial. We acknowledge that the results presented in the TMC paper has not been averaged over multiple runs, and thus, due to the slight fluctuations in performance, the final scores might not be representative. To express the usefulness of our suggestion, we provide a table, Tab. 1, presenting scores averaged over the ultimate 50 epochs. Through contact with the author, we attained the presented results, and may thus make more precise comparisons with our results.

4.3 Reproducability cost

The MNIST dataset is a relatively small dataset, and K affects the training time for each epoch depending on if it is able to store all computations in the graphical memory. One epoch with $K = 20$, using the same number of units in the MLP units and stochastic units mentioned in implementation details, took around 50-60 seconds on the different GPU's Nvidia Geforce 1060 (6 GB ram), 1070 (8 GB ram), Tesla P4 (8 GB ram). If $K = 50$, and we use the same architecture and hardware, the time increased to 70-80 seconds. When using a Nvidia Tesla P100 GPU and $K = 20$, then one epoch using IWAE took 11 seconds and 14 seconds using TMC.

5 Results

As stated in Sec. 4.2, we ran two different models of IWAE and TMC, which we refer to as large and small models, as well as conducted a hyper-parameter search for different values of K for various learning rates. Additionally we performed a smaller ablation study. To evaluate the performance of TMC for both the small and large models, we trained an IWAE over 1200 epochs. We trained the TMC models in the same manner as the IWAE models. The results of these experiments are presented as averages over three runs in Fig. 1, where the IWAE was evaluated on the IWAE objective function and the TMC on its respective objective function, i.e. $\mathcal{L}_{TMC}$. In both cases, the TMC outperforms the IWAE with the difference being most significant for the smaller model case. Graphically displaying precise comparisons between the results we obtained to those in the TMC paper was at first, as discussed in Sec. 4.2, slightly impracticable as there are no explicit scores presented for the different models. Fortunately, we were kindly supplied with the author's results and may thus give comprehensible comparisons, as is done in Fig. 1. Concerning the small models (Fig. 1, left), our results seem to align with those presented in the TMC paper in terms of convergence rates and final scores. The author's models outperforming the reproduced models, and our speculative guess is that it might partially be explained by the use of weight normalization. It is important to emphasize that our speculative guess stands unsupported. An ablation study, with and without weight normalization, should be done in order to test the hypothesis. Apart from numerical stability, which might explain the author's less fluctuating curve, weight normalization speeds up convergence ([15]). Considering the large models (Fig. 1, right), the author's TMC clearly outperform the reproduced TMC. The gap seems too large to be caused by weight normalization. Instead we expect the gap to stem from the author not averaging over multiple runs. As such we cannot expect our results to completely coincide with those from the TMC paper due to the stochasticity when sampling, initializing weights etc. But, since, the purpose of the TMC paper was to display the TMC's superiority over the IWAE, and not state-of-the-art results, we did not investigate this discrepancy further. Nonetheless, to get a sense of the final scores in our results, we computed the mean and standard deviation of the last 50 epochs for each of our models, these results are presented in Tab. 1. Clearly the averaged scores are larger than those perceived when inspecting our curves in Fig. 1. Furthermore, we also examined

the TMC paper's claim of adding negligible time when training an IWAE for each epoch, and found that our implementation of IWAE (considering $K = 20$ importance samples) takes roughly 11 s per epoch while our TMC (considering $K^n = 20^5 = 3,200,000$ effective importance samples) implementation takes roughly 14 s per epoch, this seem to corroborate the claim by the author of TMC paper of adding negligible time to train.

For the small IWAE and TMC models mentioned above, we, in Appendix C, present their conditional reconstructions. I.e., given an MNIST test sample, we show what the reconstruction looks like. Note, these are complementary experiments to the original TMC paper. As displayed in Fig. 5, both algorithms produce almost perfect reconstructions. For future work, it would be interesting to see how well they performed on more complex datasets, such as CIFAR-10 ([17]).
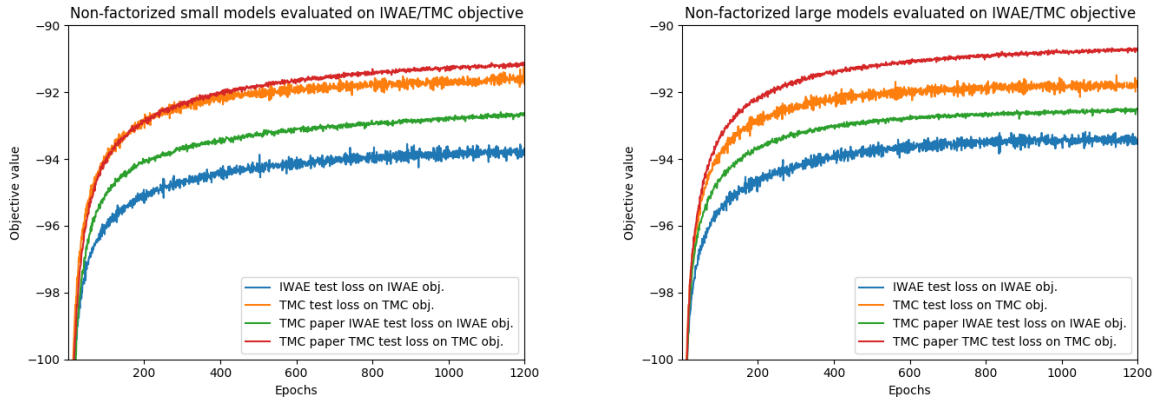

Figure 1. Results of the non-factorized IWAE and TMC models, evaluated on their respective objective functions for both small and large models over 1200 epochs. Results corresponding to legends noted with "TMC paper" was given to us by the author.

Model:	Mean objective value over last 50 epochs
IWAE large	-93.08 ± 0.19
IWAE small	-93.35 ± 0.12
TMC large	-91.94 ± 0.12
TMC small	-91.63 ± 0.16

Table 1. Reproduced results of the TMC and IWAE averaged over the last 50 epochs reported in negative log-likelihood (NLL). The values subsequent $\pm$ indicate standard deviation.

The scores presented in Fig. 3 were obtained during our ablation study. We expected the two models to give similar results, as they, indeed, did. Averaged over an increasing number of runs, the results should become indistinguishable as TMC and IWAE are in essence the same algorithms when there is only one stochastic layer, i.e. no intermediate stochastic layers in contrast to the model shown in Fig. 4. This since the parameters of the prior distribution are fixed – the model assumes a standard normal distribution, therefor there is no new information when evaluating the latent variables under the same distribution parameters. To support this claim, we refer to Alg. 1, specifically where we iterate through the distribution parameters. Mathematically, as is shown in [6], the number of evaluated importance samples grows exponentially with the layers, so if we reduce the number of layers to one, we effectively evaluate $K^1 = K$ samples in the TMC, the same as for the IWAE. Regarding the worsening of the objective value after approximately 200 epochs, we are not sure how to explain this. Probably, it is due to the insufficiently small model (100 units per deterministic layer and one stochastic layer with four units). The purpose of our ablation study was simply to show that the TMC
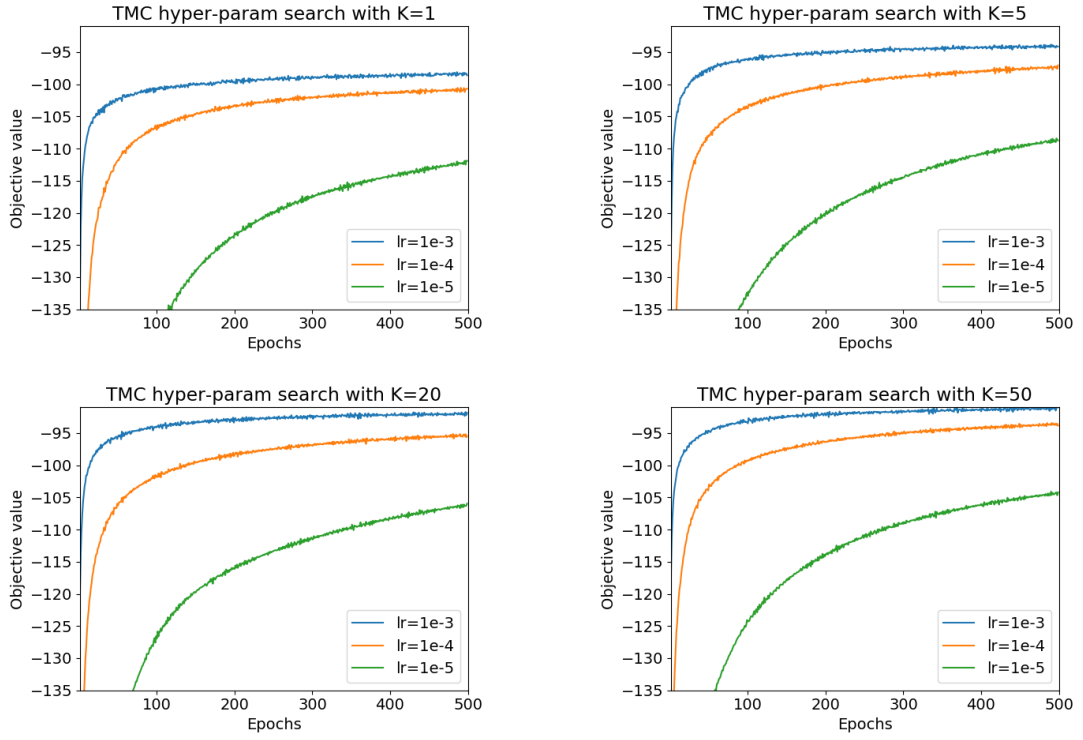
Figure 2. Results of the hyper-parameter search for the small TMC model, for different values of K and learning rates over 500 epochs.

and IWAE are the same under the mentioned conditions, and so we do not dig deeper into this phenomena.

In Appendix D, we display the TMC's clustering abilities, compared to the IWAE, via a two-dimensional latent space. By observing Fig. 6, it is clear that the TMC (right) yields less ambiguous clusters than those produced by the IWAE. Especially, note to which small degree digit nine's representation mixes with the others digit representations, as opposed to its cluster for the IWAE (left).

During our non-exhaustive hyper-parameter search, we evaluated the TMC model for $K \in \{1, 5, 20, 50\}$, and for each K we tested different learning rates lr $\in \{1e-5, 1e-4, 1e-3\}$. The results of this hyper-parameter search are presented in Fig. 2. Each of these different experiments was run over 500 epochs for the small TMC model, unsurprisingly $K = 50$ with lr $= 1e - 3$ has the best performance over these 500 epochs. This as we are effectively looking at more samples for each batch. However, setting $K = 50$ increases the running time significantly as the number of matrix calculations needed are more than doubled to that of $K = 20$ and the performance is marginally better compared to that of setting $K = 20$ and lr $= 1e-3$. The parameter settings proposed in the TMC paper seems well suited for the previous experiments based on our hyper-parameter search. We also tested with lr $= 1e - 2$, and this produced NaN results for all different values of K we tested, this might be related to the inherent numerical instability of the TMC method, especially since we have not used weight normalization in our experiments. The deployment of weight normalization might very well alleviate this problem and a larger learning rate might be able to perform better than those we have examined.
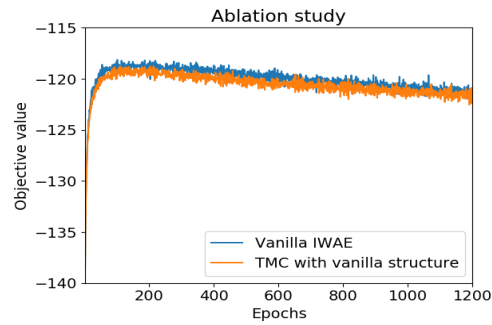
Figure 3. Results of the ablation study (more detailed description in text). The test was conducted with K=5, batch size=128 and a learning rate of 1e-3 over 500 epochs.

6 Discussion and Conclusions

Although our reproduced results and the TMC paper results differ, the general observations of the authors remain valid. I.e., the TMC clearly outperforms the IWAE model on the given dataset. This is unsurprising as the TMC, for $K = 20$ and five layers, considers a factor million more importance samples ([6]). Despite this, we see that the author's claim concerning negligible time difference between the two models is also justifiable, according to our findings. We note in our work, that the intentions of [6] were not to achieve state-of-the-art performances, but to compare the proposed the model to the baseline used in the TMC paper. Perhaps for this reason, the results are presented in a way such that they are impractical to study in detail. Although the author very skillfully and clearly describes the novelties in the TMC paper, understanding the model architecture was, for us, quite challenging. Furthermore, we argue that the specifics regarding the training and testing scheme could have been outlined in the TMC paper, at least for reproducibility. We believe, the discrepancies in test results stems from two sources, us omitting the use of weight normalization, and the presented one-shot performances, i.e. non-averaged.

To conclude, we have successfully reproduced the fundamental results of the NeurIPS 2019 paper "Tensor Monte Carlo: Particle Methods for the GPU Era", in order to support the author's claims. Additionally, we have complemented the original paper with an algorithmic description, a figure explaining the architecture and experiments displaying the TMC's reconstruction and clustering abilities. We applaud the author for an ingenious take to improve on the existing IWAE. We hope future work will be done on the tensor Monte-Carlo algorithm.

7 Acknowledgements

References

1. D. P. Kingma and M. Welling. **Auto-Encoding Variational Bayes**. 2013. arXiv: 1312.6114 [stat.ML].
2. Y. Burda, R. Grosse, and R. Salakhutdinov. **Importance weighted autoencoders**. 2015. arXiv: 1509.00519 [stat.ML].
3. D. J. Rezende and S. Mohamed. **Variational inference with normalizing flows**. 2015. arXiv: 1505.05770 [stat.ML].

4. C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. "Ladder variational autoencoders." In: **Advances in neural information processing systems**. 2016, pp. 3738–3746.
5. D. P. Kingma and P. Dhariwal. **Glow: Generative Flow with Invertible 1x1 Convolutions**. 2018. arXiv: 1807. 03039 [stat.ML].
6. L. Aitchison. "Tensor Monte Carlo: particle methods for the GPU era." In: **Advances in Neural Information Processing Systems**. 2019, pp. 7146–7155.
7. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic differentiation in PyTorch." In: **NIPS-W**. 2017.
8. Martín Abadi et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. Software available from tensorflow.org. 2015.
9. D. P. Kingma and M. Welling. **An Introduction to Variational Autoencoders**. 2019. arXiv: 1906.02691 [cs.LG].
10. G. Roeder, Y. Wu, and D. K. Duvenaud. "Sticking the landing: Simple, lower-variance gradient estimators for variational inference." In: **Advances in Neural Information Processing Systems**. 2017, pp. 6925–6934.
11. G. Tucker, D. Lawson, S. Gu, and C. J. Maddison. **Doubly reparameterized gradient estimators for Monte Carlo objectives**. 2018. arXiv: 1810.04152 [cs.LG].
12. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. "Gradient-based learning applied to document recognition." In: **Proceedings of the IEEE** 86.11 (1998), pp. 2278–2324.
13. F. Chollet et al. **Keras**. https://keras.io. 2015.
14. D. P. Kingma and J. Ba. **Adam: A Method for Stochastic Optimization**. 2014. arXiv: 1412.6980 [cs.LG].
15. T. Salimans and D. P. Kingma. **Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks**. 2016. arXiv: 1602.07868 [cs.LG].
16. X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: **Proceedings of the thirteenth international conference on artificial intelligence and statistics**. 2010, pp. 249–256.
17. A. Krizhevsky. **Learning multiple layers of features from tiny images**. Tech. rep. 2009.

Appendices

A Numerically stable tensor inner product (in log-domain)

The operation of computing ratios of probabilities is typically unstable, especially if the probabilities have small values. In order to average over a large number of importance samples in log-domain, we need a numerically stable method. If we compute

$$e^{Z_{ik}} = \sum_j e^{X_{ij}} e^{Y_{jk}},$$

in order to get

$$Z_{ik} = \log \sum_j e^{X_{ij}} e^{Y_{jk}},$$

very large or small values of X_{ij} and (or) Y_{jk} could lead to numerical instability. To circumvent this issue, we subtract the largest values along j before computing the sum. Then, to ensure the correctness of our marginal log-likelihoods, we add these values back as follows

$$Z_{ik} = \log \sum_j e^{X_{ij} - \max_j(X_{ij})} e^{Y_{jk} - \max_j(Y_{jk})} + \max_j(X_{ij}) + \max_j(Y_{jk})$$

B TMC Architecture

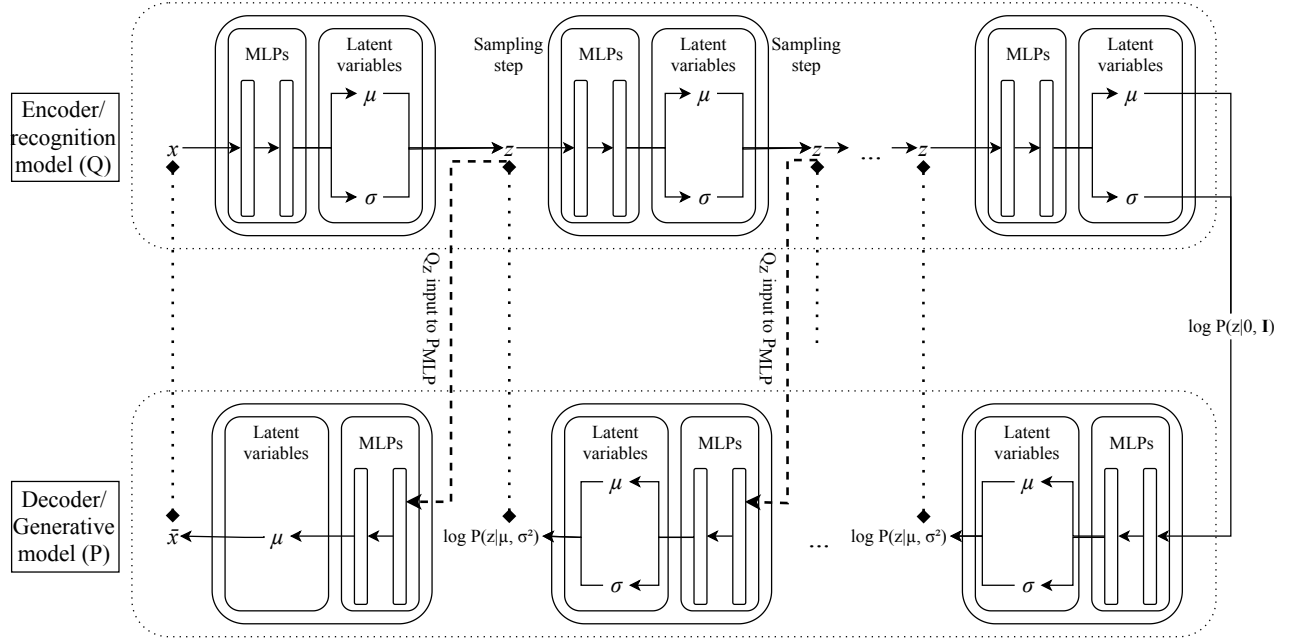Here we present a graphical overview of the model architecture we used for the TMC.

Figure 4. TMC architecture as textually presented ([6]). The final stochastic layer in the generative model assumes a Bernoulli distribution, why only the mean is computed in the corresponding MLP.

C Reconstructions

Below we display the reconstructions for the small IWAE (left) and small TMC (right) models, given an observation (sample) from the MNIST test set. In both figures, the far left column shows the input test samples (original), and the remaining columns the reconstructions.
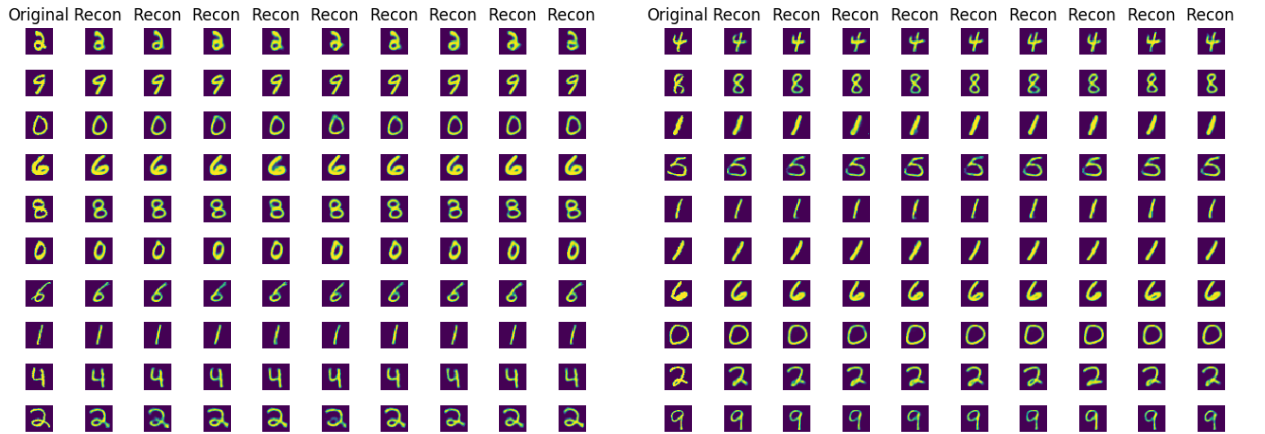

Figure 5. Reconstructions from the small IWAE (left) and TMC (right) models, after 1200 epochs of training.

D Clustering

We compared the clustering abilities for the IWAE versus the TMC algorithms, given a two-dimensional latent space (the latent space farthest from the data). In this experiment, there were two stochastic layers in the recognition model with dimensions 50 and 2 (deterministic layers [100, 100] and [100, 100] for the two MLPs). To obtain these data points, we averaged over K for every latent. It is apparent from Fig. 6 that the TMC offers a richer latent space than the IWAE, for this specific architecture. For instance, note how the digit nine's representation (view color bar) is not mixing as much, for the TMC (right), with its neighboring clusters as is the case for the IWAE (left).
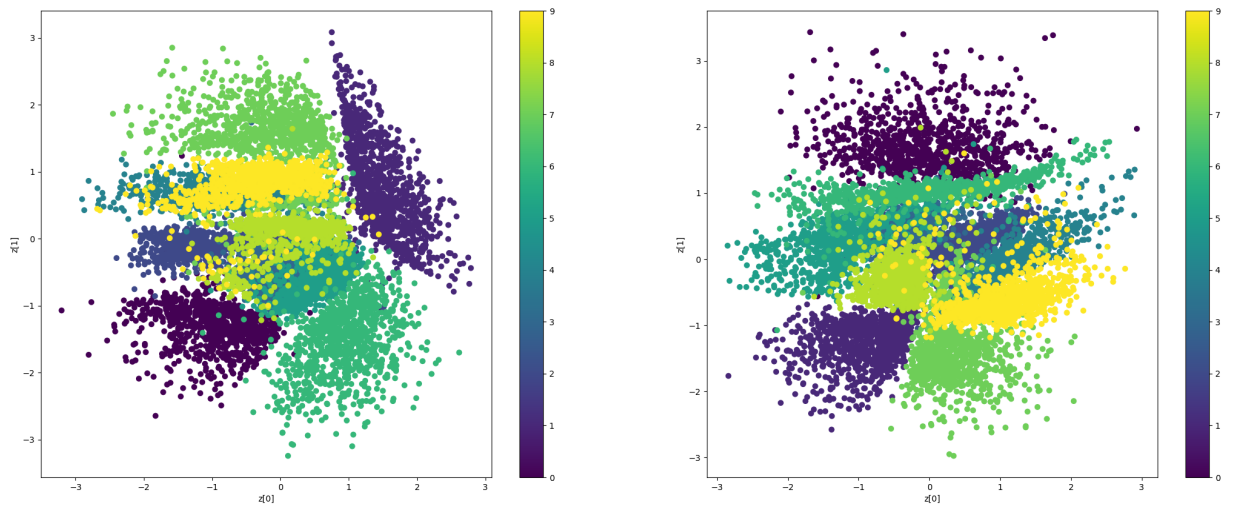

Figure 6. Clusters in a two dimensional latent space by the IWAE (left) and TMC (right), after 700 epochs of training and the special architecture mentioned in Appendix D. Best viewed in color.