Flexible Long Sequence Modeling with Hierarchical HMMs

Artem Zholus [1] **Giovanni Belval** [1]

Abstract

Hidden Markov Models (HMMs) have been used for sequential generative modeling for decades. However, their representational capacity and computational efficiency remain limited. To address both issues, we enrich its distributional family with a hierarchical component and drop several dependencies to make the family wider and training algorithm more efficient. Our aim is to bolster the performance of long-sequence HMMs by exploring the Hierarchical Hidden Markov Model (**HHMM**) variant, promising improved capture of intricate temporal dependencies.

1. Introduction

Sequential generative modeling is a prominent task of statistical learning. It has numerous applications in supervised learning, self-supervised learning, and reinforcement learning, as well as many ML applications such as forecasting or sequence classification. One of the oldest mehtods in this realm is Hiden Markov Model (HMM), a temporal latent variable generative model that has many applications from Natural Language Processing to Reinforcement Learning. However, with the advent of modern architectures of neural networks, HMM has been put in the backyard due to its limited capacity and training efficiency.

There are two biggest challenges that prevent HMM from being competitive against modern architectures. First, HMM requires sequential recursive computation over timesteps in the data sequence. Second, the Markov assumption in HMM makes it difficult to propagate information over non-local context windows.

In this work, we aim to overcome both limitations by proposing a new architecture analogous to HMM that is capable of flexible long-sequence learning as well as of parallelizable computation in the inference algorithm.

[*]Equal contribution [1]MILA. Correspondence to: Artem Zholus <artem.zholus@mila.quebec>, Giovanni Belval <Giovanni.belval@umontreal.ca>.

IFT 6269 project report - Université de Montréal

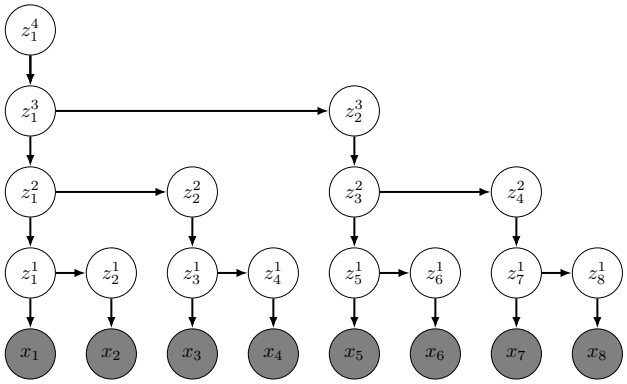

Figure 1. Our HHMM architecture, $T = 2$ and $L = 8$. The L parameter depends on the input data, but in our case the parameter T is fixed to 2.

2. Related Work

2.1. Hidden Markov Models

Hidden Markov Model (HMM) is a powerful and interpretable technique to perform generative modeling, inference, filtering, and prediction probabilistic tasks of sequential data. HMM assumes there is an observed sequence $x_{1:T}$ and also a sequence of latents variables $z_{1:T}$ each of which emits the corresponding observation. HMM assumes the following factorization

$$p(x_{1:T}, z_{1:T}) = p(z_1)p(x_1 \mid z_1) \prod_{t=2}^{T} p(z_t \mid z_{t-1})p(x_t \mid z_t) \tag{1}$$

In machine learning, this model is typically traine with the EM algorithm. The parameters of this model are matrix $A_{ij} = p(z_t = i \mid z_{t-1} = j)$, vector $\pi = p(z_1)$ and also the parameters θ for the $p(x_t \mid z_t; \theta)$ that may vary depending on the choice of the emission distribution (e.g. categorical or gaussian). This algorithm consists of two steps that are applied repeatedly until convergence:

1. E-step. We compute the posterior distribution $q(z_t) = p(z_t \mid x_{1:T})$ and $q(z_{t-1}, z_t \mid x_{1:T})$ e.g. via message-passing algorithm.
2. M-step. We solve the optimization problem

$\max_{A,\pi,\theta} \mathbb{E}_q \log p(x_{1:T}, z_{1:T}; A, \pi, \theta)$

In this work, we use HMM trained with the EM algorithm as the baseline model and modify it to meet our desiderata.

2.2. Parallel Inference on Sequences

A new prominent direction machine learning research has been devoted to building more capable recurrent models that are parallelizable over timesteps (Smith et al., 2023; Gu et al., 2021). The idea behind these methods is that we can introduce a binary operator, whose result is equivalent to doing one forward pass of the recurrent neural network. The case when such binary operator is associative means we can parallelize the inference of RNN via parallel scan (Smith et al., 2023). An example of such parallelizable RNN is linear RNN (Gu et al., 2021). Another example of parallelizable recurrent computation is the sum-product algorithm for HMM (Hassan et al., 2021). However, in (Hassan et al., 2021) the authors do not report the accuracy of the inference and our preliminary investigation showed that such computation is vulnerable to numerical underflow. Therefore, we decided to switch to the architectural enforcing of the parallelizable computation for the sum-product algorithm.

3. Hierarchical HMM

3.1. Model Architecture

The structure of a Hierarchical Hidden Markov Model (**HHMM**) is characterized by a hierarchical arrangement of hidden states that captures dependencies at multiple levels of abstraction within sequential data.

However, in our case, we decided to reduce the connectivity in order to create a tree graph. An instance of our model is illustrated in Figure 1. Within our model, complexity is adaptable and contingent upon both the entry size, denoted as L, and the number of subsequent nodes connected, marked as T. This adaptability allows us to demonstrate that the number of layers is determined by $\log_T L$.

Contrary to the typical single transition matrix structure in Hidden Markov Models (HMMs), our model works by employing distinct matrices to regulate transition behaviors. Specifically, we utilize a horizontal transition matrix, labeled as A^l, which governs transitions among nodes situated at the same level, indexed as l. Additionally, a vertical transition matrix, denoted as B^l, oversees transitions between nodes at subsequent levels, moving from level l to $l-1$. Their definitions are outlined as follows:

$$A^l_{i,j} = P(z^l_t = i | z^l_{t-1} = j) \tag{2}$$

$$B^l_{i,j} = P(z^{l-1}_{2t-1} = i | z^l_t = j) \tag{3}$$

Here, z^l_t represents the t-th node at level l. In scenarios involving multiple observations, $z^{l,(i)}_t$ denotes the t-th node at level l within the observations indexed by i.

4. Training

the advantage of using such a structure described above [1] is that we can still use **sum product** algorithm as in a vanilla HMM. To understand how the collective and distributive phase has been implemented, allow us to perform a brief examination at our HHMM framework pictured below (Figure 2) :

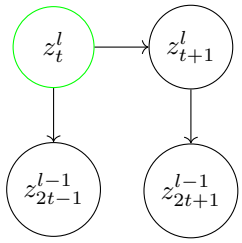

Figure 2. One cell of the HHMM, our model is composed of recursion of this cell. this cell is denoted **cell**^l_t because its root node is the node z^l_t, in green. When we are on the first layer, $l = 1$ the bottom node are the observed x_t

We begin by gathering messages $m_{x_t \to z^1_t}(z^1_t)$ using the message passing formula for those two nodes :

$$m_{x_t \to z^1_t}(z^1_t) = \sum_{x_t} \psi_{x_t, z_t}(x_t, z_t) \tag{4}$$

$$= \sum_{x_t} p(x_t | z^1_t) \delta(x_t, \bar{x}_t) \tag{5}$$

$$= p(\bar{x}_t | z^1_t) \tag{6}$$

Where $\bar{x}_t$ represents the observed data.

. Next, we calculate the message originating from the last node in the initial layer to the left, denoted as $m_{z^1_{t+1} \to z^1_t}(z^1_t)$ by once again using the message passing formula :

$$m_{z^1_{t+1} \to z^1_t}(z^1_t) = \sum_{z^l_{t+1}} \psi_{z^l_t, z^l_{t+1}}(z^l_t, z^l_{t+1}) m_{x_{t+1} \to z^1_{t+1}}(z^1_{t+1}) \tag{7}$$

$$= \sum_{z^l_{t+1}} p(z^l_{t+1} | z^l_t) m_{x_{t+1} \to z^1_{t+1}}(z^1_{t+1}) \tag{8}$$

$$= \sum_{z^l_{t+1}} p(z^l_{t+1} | z^l_t) p(\bar{x}_{t+1} | z^1_{t+1}) \tag{9}$$

$$\tag{10}$$

We can remark that all the quantities involved in this computation are known, $p(z^l_{t+1} | z^l_t)$ is our horizontal transition matrix at level l denoted as $A^{[l]}$ while $p(\bar{x}_{t+1} | z^1_{t+1})$ is our hypo-

thetical pixel conditional distribution, which is a Bernoulli in our case.

. Subsequently, we derive all messages, this can be seen in Figure 3. Once we reach this point, we employ the recursive nature of our model. We perform analogous computations one layer above, utilizing the messages from the lower cell's root (z_l^t) as if they were emitted from nodes within our current cell. This process is depicted in Figure 4.

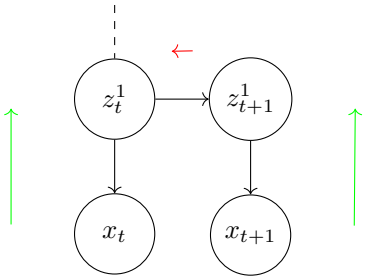

Figure 3. Collective phase in the first cell, green arrows represent the collected message during the first phase, and then once they are all collected, we compute horizontal backward message (in red)

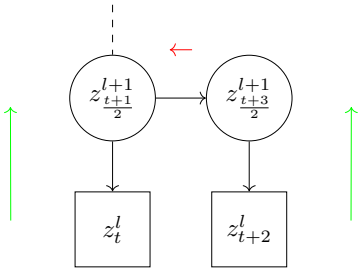

Figure 4. Model pictured as recursive cells, here the square denotes a cell for which the root node is the node depicted inside.

Once we've reached the root node, our collective phase concludes, allowing us to commence the distributive phase. This phase involves the distribution of messages originating from the top and left for each node. In a tree-like Directed Graphical Model (DGM), all the requisite quantities required for this computation would have been precomputed during the collection phase.

Denoting by χ_t^l, γ_t^l, ν_t^l, and β_t^l the messages originating from the top, left, bottom, and right nodes, respectively, with indices l and t representing the index of the node from which the message originates, our model's distinctive structure eliminates ambiguity regarding the target node for computed messages.

We compute ν's and β's during the collection phase, followed by the computation of γ's and χ's during the distributive phase. The message passing algorithm for tree-like DGM yields us the following recursive rules for our messages :

- If z_t is the root of its cell :

$$\gamma_t^l = A^{[l]} \cdot (\chi^{l+1}_{\frac{t+(T-1)}{T}} \odot \nu^{l-1}_{Tt-(T-1)}) \tag{11}$$

$$\nu_t^l = B^{[l+1]} \cdot (\nu^{l-1}_{Tt-(T-1)} \odot \beta^l_{t+1}) \tag{12}$$

$$\chi_t^l = B^{[l]} \cdot (\chi^{l+1}_{\frac{t+(T-1)}{T}} \odot \beta^l_{t+1}) \tag{13}$$

- if z_t^l is the top right node of its cell :

$$\beta_t^l = A^{[l]} \cdot \nu^{l-1}_{Tt-tT-1)} \tag{14}$$

$$\chi_t^l = B^{[l]} \cdot \gamma^l_{t-1} \tag{15}$$

- otherwise :

$$\gamma_t^l = A^{[l]} \cdot (\gamma^l_{t-1} \odot \nu^{l-1}_{Tt-(T-1)}) \tag{16}$$

$$\chi_t^l = B^{[l]} \cdot (\gamma^l_{t-1} \odot \beta^l_{t+1}) \tag{17}$$

$$\beta_t^l = A^{[l]} \cdot (\beta^l_{t+1} \odot \nu^{l-1}_{Tt-tT-1)}) \tag{18}$$

At the end of our distributive phase we have computed and stored all the messages between subsequent nodes in our model, thus we can derive node and edge marginals necessary for updating our parameters in the M-step. To compute the node marginal of a specific node z_t^l, referenced as $p(z_t^l, x_1, \ldots, x_t)$, it entails the multiplication of all incoming messages directed towards that node with the potential function. Also we need to consider the observations that are fixed.

The type of message required varies based on the specific node for which we're computing the node marginal probability. This variation stems from differences in connectivity; for instance, certain nodes lack subsequent nodes on the top and solely possess horizontal and bottom connections. As a result, the necessary message could be from the bottom, right, left, or top, contingent upon the node's specific connectivity structure within the graphical model. For instance, if this node is the root node of it's cell (Figure [5]), using the node marginal probability formula for the sum product algorithm we have :

$$p(z_t^l, \bar{x}_1, \ldots, \bar{x}_t) = \frac{1}{Z} \psi_{z_t^l}(z_t^l) \nu^{l-1}_{2t-1} \beta^l_{t+1} \chi^{l+1}_{\frac{t+1}{2}} \tag{19}$$

where Z is the normalization constant and $\psi_{z_t^l}(z_t^l)$ is our node potential for z_t^l.

Moreover, in our case all node potential $\psi_{z_t^l}(z_t^l)$ is fixed to one except for the root node of the model, then the previous equation becomes :

$$p(z_t^l, \bar{x}_1, \ldots, \bar{x}_t) = \frac{1}{Z} \nu^{l-1}_{2t-1} \beta^l_{t+1} \chi^{l+1}_{\frac{t+1}{2}} \tag{20}$$

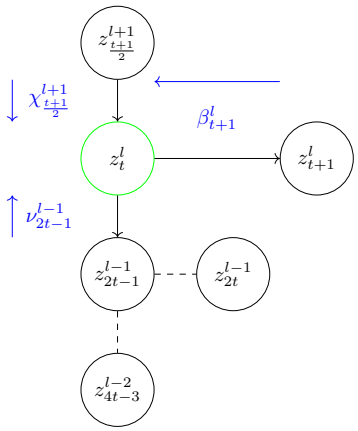

Figure 5. Computation of the node marginal probability for the node a colored in green, incoming message to that node are colored in blue.

then we can compute our E step at the iteration $s+1$ by computing :

$$q_{s+1}(z_t^l) = p(z_t^l|\bar{x}_1, \ldots, \bar{x}_t) = \frac{p(z_t^l, \bar{x}_1, \ldots, \bar{x}_t)}{p(\bar{x}_1, \ldots, \bar{x}_t)} \quad (21)$$

The process remains identical when dealing with edges instead of individual nodes. The edge marginal is computed as the normalized product of all incoming messages within the nodes comprising that particular edge, multiplied by the edge's potential. This edge, being a clique with two elements, relies on the two-clique potential, which in our case is represented by the transition probabilities between these two elements encoded as A^l and B^l at level l. For instance, computing the edge marginal for two nodes in the configuration of Figure 6 we would have :

$$p(z_t^l = i, z_{2t-1}^{l-1} = j, \bar{x}_1, \ldots, \bar{x}_t) = \frac{1}{Z} B^{[l]}_{i,j} \chi^{l+1}_{\frac{t+1}{2}} \beta^l_{t+1} \beta^{l-1}_{2t} \nu^{l-2}_{4t-3} \quad (22)$$

In our context, the computation of our q function for edge inference follows the same methodology as for nodes, utilizing the edge marginal instead of the node marginal.

This approach ensures that the estimation for the posterior distribution regarding the edge follows a parallel procedure to that used for nodes.

$$q_{s+1}(z_t^l, z_{t+1}^l) = p(z_t^l, z_{t+1}^l|\bar{x}_1, \ldots, \bar{x}_t) \quad (23)$$

$$= \frac{p(z_t^l, z_{t+1}^l, \bar{x}_1, \ldots, \bar{x}_t)}{p(\bar{x}_1, \ldots, \bar{x}_t)} \quad (24)$$

During the M step, we need to update $\log_T L - 1$ vertical transition matrix $B^{[l]}$, $\log_T L - 1$ horizontal transition matrix A^l, the prior distribution π of the root node, and the

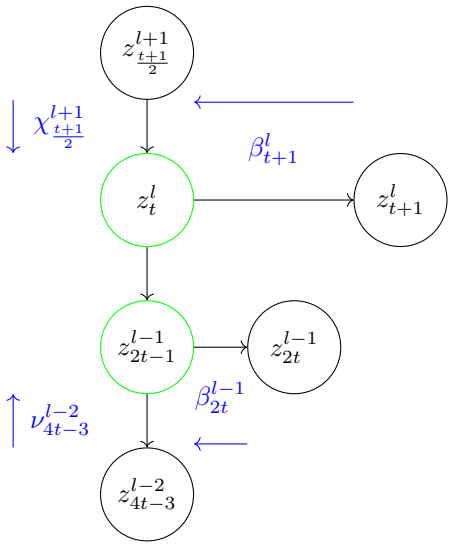

Figure 6. Example configuration for computing edge marginal for the edge containing the nodes z_t^l and z_{2t-1}^{l-1}, colored in green, and the incoming messages colored in blue.

Bernoulli coefficients for the output nodes p_j, where

$$p(x_t^i|z_t^1 = j) = p_j^{x_t^{(i)}}(1-p_j)^{1-x_t^{(i)}} \quad (25)$$

These parameters are updated as follows :

$$\hat{A}^l_{n,m} = \frac{\sum_{i=1}^N \sum_{d=1}^{L/2^l} \sum_{t=Td+2}^{Td+T} q(z_t^{l,(i)} = n, z_t^{l,(i)} = m)}{\sum_{i=1}^N \sum_{d=1}^{L/2^l} \sum_{t=Td+2}^{Td+T} \sum_{m=1}^K q(z_t^{l,(i)} = n, z_{t-1}^{l,(i)} = m)}$$

$$\hat{B}^l_{n,m} = \frac{\sum_{i=1}^N \sum_{t=1}^{L/T^{l-1}} q(z_t^{l,(i)} = n, z_{2t-1}^{l-1,(i)} = m)}{\sum_{i=1}^N \sum_{t=1}^{L/T^{l-1}} \sum_{m=1}^K q(z_t^{l,(i)} = n, z_{2t-1}^{l-1,(i)} = m)}$$

$$\hat{\pi}_j = \frac{\sum_{i=1}^N q(z_1^{root,(i)} = j)}{\sum_{i=1}^N \sum_{i=j}^K q(z_1^{root,(i)} = j)}$$

$$\hat{p_j} = \frac{\sum_{i=1}^N \sum_{t=1}^L q(z_t^{1,(i)} = j) x_t^{(i)}}{\sum_{i=1}^N \sum_{t=1}^L q(z_t^{1,(i)} = j)}$$

5. Experiments

In this section we describe the experimental validation that we performed with the proposed HHMM model. We seek to evaluate our model in two ways: 1) how well can it fit the long-sequence data compared to the standard HMM; 2) how much faster the model can be compared to standard linear recursive HMM.

Our model is implemented with the Jax library (Bradbury et al., 2018), which we chose due to it's optimizations that

it can autometically bring to any computational pipeline. On top of this, another reason for choosing jax was that it simplified the implementation of the message-passing algorithm because of its function transformations that, in our case, helped parallelize message-passing over independent parts of the graph.

5.1. EM algorithm on MNIST

We use the MNIST (LeCun & Cortes, 2010) dataset to perform the long-sequence generative modeling experiment. In this study, we tested the principled ability of the model to fit individual long sequences. In particular, we extracted one image of MNIST and upscaled it from 28×28 pixels to 32×32 pixels. This is to ensure that the sequence length is a power of 2 or (T in general, which is a maximum number of horizontally connected latents in one layer). Even though the model can work with arbitrary sequence lengths, the most efficient parallelization can be achieved when L is exactly a power of T.

After upscaling the image, we binarize it and flatten into a vector of length $L = 1024$. For HHMM, we use 6 latent layers with latent sizes of $13, 11, 7, 5, 3, 2$ and $T = 4$. That is, the first layers has 13-valued categorical latents with sequence length 1024, the second layer has 11-valued categorical distributions with sequence length 256, and so on. In addition, we train standard linear HMM on the same sequence. the hidden size of the HMM was set to 25 as in that case HMM has the same number of trainable parameters as the aforementioned HHMM.

We run E-M algorithm for our proposed model as well as for sequential HMM. The resulting sequence data loglikelihood is shown in Figure 7. Our model significantly outperforms the baseline HMM in terms of the ability to fit the data.

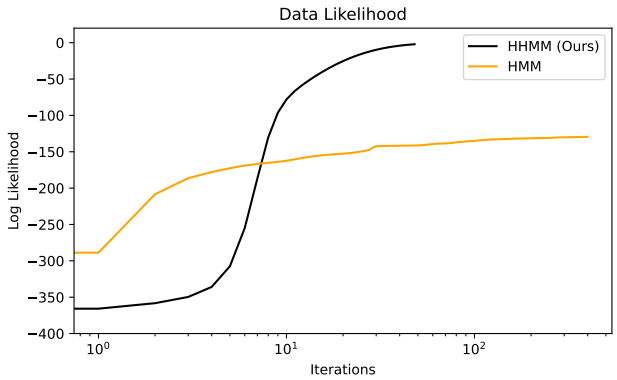

Figure 7. Training curves for HMM and HHMM on MNIST data.

5.2. Computational Performance

In addition, we performed another computational study to compare HHMM and HMM. In particular, we compare our sequential implementation of the EM algorithm for HMM in Jax with our parallel implementation of the EM algorithm for HHMM. The central idea beyond parallelization was that we can compute messages of independent parts of the graph independently. For example, we compute $m_{z^l_{t+kT} \to z^l_{t+kT+1}}(z^l_{t+kT+1})$ in parallel over k. In Figure 8, we show the comparison of how much time each of the algorithms spent doing one EM step. Note that for both algorithms, the time was reported excluding the time Jax spent on optimizing the computational graph, which happens only once, at the beginning of the training[1].

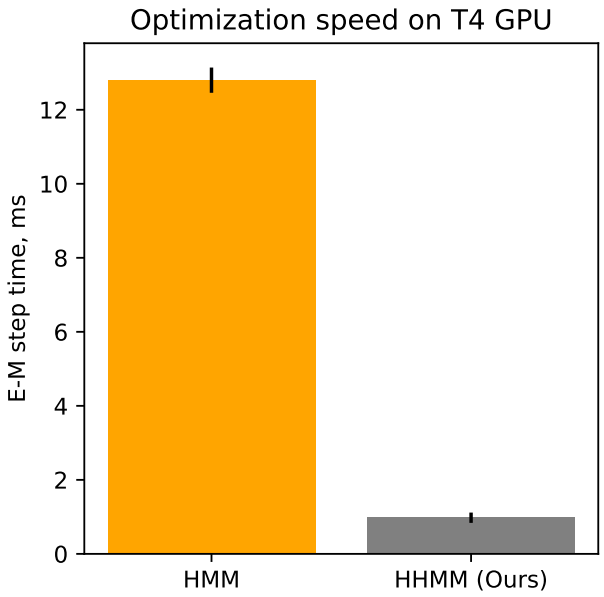

Figure 8. Comparison of the time each algorithm took to make one optimization step on GPU.

6. Conclusion

In this work we presented HHMM, a more representationally capable and computationally efficient counterpart of HMM. We derived the message-passing scheme for the EM algorithm for the proposed model as well as the maximization step (of the EM algorithm) within our graphical model. We showed that the proposed HHMM model can converge to the log-likelihood that is significantly higher than the one of HMM. At the same time, HHMM is more than $10\times$ faster computationally.

For future work in this direction, we consider doing complete training on the full dataset, sampling from the trained distribution, and analysis of the learned latent variables in terms of how they align with high-level properties of data. For, example, the root latent node can represent the over-

[1]Our Jax implementation is available at: https://gist.github.com/artemZholus/91c31d6a10bfbbbad1b7a4e2d207e0ca

all information about the data instance such as the class label. Additionally, we consider a more densely connected directed graph for which Loopy Belief Propagation could be applied, which can also be the future work in this direction.

References

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *CoRR*, abs/2111.00396, 2021. URL https://arxiv.org/abs/2111.00396.

Hassan, S. S., Särkkä, S., and García-Fernández, F. Temporal parallelization of inference in hidden markov models. *IEEE Transactions on Signal Processing*, 69:4875–4887, 2021. doi: 10.1109/TSP.2021.3103338.

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Smith, J. T., Warrington, A., and Linderman, S. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Ai8Hw3AXqks.