Learning Flows By Parts

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Flow-based models have exhibited promising results in image synthesis due to their exact likelihood computation, efficient sampling, and invertible transformation. However, state-of-the-art flows introduce significant model complexity in deep neural networks which induces long training times even on high-end GPUs. Yet, unlike standard objective functions, the normalizing flow objective function has a simple summation structure across layers because the log determinant terms merely sum across layers. Thus, inspired by recent work on optimizing large modules independently rather than full end-to-end backpropagation [Löwe et al., 2019], we propose to optimize normalizing flow models in parts rather than full end-to-end backpropagation to see if we can achieve similar performance even without end-to-end backpropagation. In particular, each model part is gradient-isolated such that each part does not exchange gradient information with other parts. This approach could enable modularized training of large flow models, reduced communication between model parts (i.e., forward-only communication), and a possible new perspective on training large flow models. We present initial results along this direction for training the state-of-the-art Glow model [Kingma and Dhariwal, 2018] via parts showing that in some cases we can achieve comparable accuracy in a shorter time.

1 Introduction

Inspired by the recent success of local modular optimization rather than end-to-end optimization in Löwe et al. [2019], we ask a similar question: Can state-of-the-art normalizing flow models be trained without full end-to-end backpropagation but still attain reasonable performance? An answer to this question could support new ways of thinking about training generative modeling and also have other benefits such as reduced computational burden and reduced communication between model parts. In particular, if learning algorithms are ever to deploy across multiple devices which may have intermittent communication and reduced computational capacity, localized learning that does not require synchronization with future parts may be important. We expect localized training of flows to be possible because of their particular objective structure that is different from standard loss functions as we describe next.

Normalizing flow models [Rezende and Mohamed, 2016, Dinh et al., 2017, Kingma and Dhariwal, 2018] have seen promising success in generating realistic images but also include tractable log likelihood computations. Normalizing flows leverage the multivariate change of variables formula to specify a probabilistic model by only two objects: 1) a latent but known prior distribution $p_z(z)$ (usually standard Gaussian or uniform distribution) and 2) an invertible function f that transforms between the original space and the latent representation. More formally, this can be written as:

$$\log p_x(x) = \log p_z(f(x)) + \log \left| \det \frac{\partial f(x)}{\partial x} \right| , \tag{1}$$

where $\frac{\partial f(x)}{\partial x}$ is the Jacobian of f. Generally, the function f is composed of a composition of invertible functions, i.e., $f = f_L \circ f_{L-1} \circ f_{L-2} \circ \cdots \circ f_1$, where L denotes the number of layers in the deep invertible network. Thus, to perform maximum likelihood estimation, we can minimize the following equation:

$$\min_{f_1, \cdots, f_L} -\log p_z(f(x)) - \sum_{\ell=1}^{L} \log \left| \det \frac{\partial f_\ell(z^{(\ell-1)})}{\partial z^{(\ell-1)}} \right| , \tag{2}$$

where $z^{(\ell-1)} = f_{\ell-1} \circ \cdots \circ f_1(x)$ is the intermediate latent representation. This reveals the summation structure of the log likelihood in terms of the layers f_ℓ. While the layers are interdependent because of the latent representations, we could consider the following *local* "truncated" optimization problem for the k-th layer, where we assume other layers are held fixed:

$$\min_{f_k} -\log p_z(f^{(1:k)}(x)) - \sum_{\ell=1}^{k} \log \left| \det \frac{\partial f_\ell(z^{(\ell-1)})}{\partial z^{(\ell-1)}} \right| = \min_{f_k} -\log p_z(f^{(1:k)}(x)) - \log \left| \det \frac{\partial f_k(z^{(k-1)})}{\partial z^{(k-1)}} \right| , \tag{3}$$

where $f^{(1:k)} = f_k \circ \cdots \circ f_1$ and the summation terms drop out because they are constant for f_k. Notice that this allows a local reduction in each of the log determinant terms even though it may not be globally optimal. Each of these terms are dependent on previous layers but only through the latent representation $z^{(k-1)}$ and thus assuming prior layers are fixed, we can optimize for each f_k locally. We explore the above local optimization problems for optimizing the joint model without full end-to-end backpropagation. In particular, we consider splitting the state-of-the-art Glow [Kingma and Dhariwal, 2018] model into 2 or 3 large parts and training each part without gradients from the other parts using the local optimization defined above. Our initial results show that we can get reasonable performance compared to end-to-end training but our experiments also highlight new challenges for this approach.

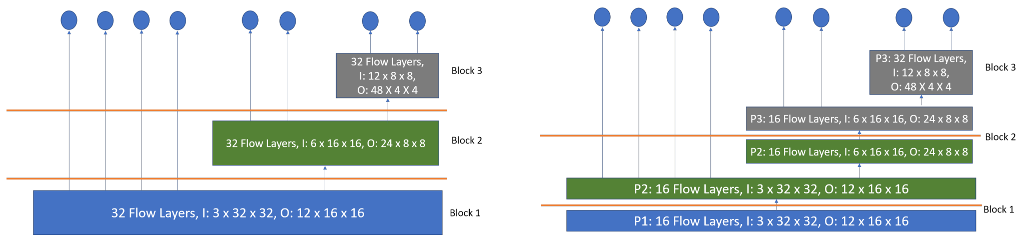
2 Glow Model

We briefly describe the state-of-the-art Glow [Kingma and Dhariwal, 2018] model which has stood out among other generative flow-based models due to its effective density estimation, simple architecture, and image synthesis. There are 32 or 48 flow layers each of which consists of a single layer of Actnorm, Invertible 1x1 Convolution, and affine coupling. Actnorm is initialized via the training data to normalize the post-norm activations to be zero mean and unit variance but is not updated for every batch like BatchNorm. The invertible 1x1 convolutions can be seen as generalizing a permutation of the channels to an arbitrary invertible mixing of the channels. The affine coupling layers originally introduced by [Dinh et al., 2017] enable complex invertible functions by holding some channels fixed and altering the other channels conditioned on the first:

$$\begin{aligned} x_1, x_2 &= \text{SplitChannelDim}(x) \\ y_2 &= g(x_1) + \exp(h(x_1)) \odot x_2 \\ y &= \text{ConcatenateChannelDim}(y_1, y_2)\,, \end{aligned}$$

where g and h can be arbitrary neural networks. This affine coupling structure is easy to invert, easy to compute the log determinant while the 1x1 convolutions ensure that all channels can be affected. Finally, to build up the final model, Glow uses the squeeze operation and hierarchical model architecture from Dinh et al. [2017]. Glow is split into 3 high-level blocks that first apply a squeeze operation (which quadruples the channels but halves the height and width) and then apply 32 flow layers. After each block, only half of the channels are propagated to the next block as in Dinh et al. [2017]. Intuitively, with end-to-end training, higher-level information is passed to the higher blocks since it can be processed more by future layers. The high level diagram can be seen in Figure 1. Derivative works on flow based models (e.g., Flow++ [Ho et al., 2019], Macow [Ma et al., 2019]) are expensive to train and generally require time in order of weeks[1] to converge to the specified results. In the next section, we describe our parts-based training approach while attempting to maintain comparable performance in terms of bits per dimension.

[1]Training time also depends on the number of GPUs used. Glow is trained on 8 GPUs over 7 days for CIFAR-10.


32 Flow Layers, I: 12 x 8 x 8, O: 48 X 4 X 4
Block 3
32 Flow Layers, I: 6 x 16 x 16, O: 24 x 8 x 8
Block 2
32 Flow Layers, I: 3 x 32 x 32, O: 12 x 16 x 16
Block 1
P3: 32 Flow Layers, I: 12 x 8 x 8, O: 48 X 4 X 4
Block 3
P3: 16 Flow Layers, I: 6 x 16 x 16, O: 24 x 8 x 8
Block 2
P2: 16 Flow Layers, I: 6 x 16 x 16, O: 24 x 8 x 8
P2: 16 Flow Layers, I: 3 x 32 x 32, O: 12 x 16 x 16
Block 1
P1: 16 Flow Layers, I: 3 x 32 x 32, O: 12 x 16 x 16

(a) Split-by-Block Training

(b) Split-across-Block Training

Figure 1: The Glow model (left) for CIFAR-10 dataset includes 3 blocks, where each block consists of a squeeze layer (to increase the number of channels), 32 Flow Layers (Actnorm, Invertible 1x1 Convolution, and Affine Coupling), and a split layer (that halves the number of channels to pass on to the next block). We consider two ways to split the Glow model into parts (orange lines): (left) split the model by block where each part is a single block or (right) split the model across the blocks so that the split layer is included within the part. Each part is denoted as P1(Blue), P2(Green), and P3(Grey). For two-part models, we combine P2 and P3. "I" and "O" represent the input and output dimensions for individual block respectively.

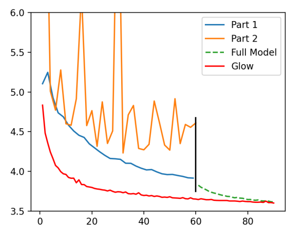
3 Splitting Flows

Owing to their depth, flow-based models require enormous GPU cycles to train. An example of Glow model used for training on CIFAR-10 Krizhevsky et al. is shown in figure 1a. Inspired from the training mechanism behind Löwe et al. [2019], we attempt to split the Glow model in parts (hereby referred to as Glow Part) and optimize each part locally via Equation 3 with an objective of avoiding end-to-end backpropagation. We propose two methods for splitting up the model and two training schemes for comparison with the goal of understanding the gap between global training and parts-based training.

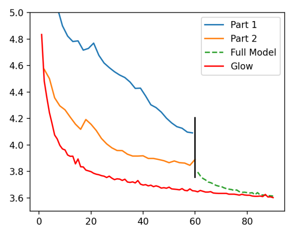
We propose two ways to split the model. First, we propose to split the model by blocks (see Figure 1a) which means that each part consists of one or more whole blocks. This seems to be the most obvious way to split. Since half of the channels are lost in splitting between the Glow parts, the local optimization does not pass higher-level information into half of the channels that will continue being processed. Thus, this approach has a major disadvantage, and our experiments show that this leads to poor performance. Second, we propose to split the model across blocks (Figure 1b) where one or more block is split by the number of flow layers. Details are shown in figure 1b. We aim to make our model customizable with respect to the number of Glow parts, the type of block splits, and the number of flow layers for each Glow part. The Glow parts are individually trained while keeping the parameters of all the previous Glow parts in a frozen state, i.e. stopping the flow of gradients for Glow parts P_{i-1} from propagating backward during the training of Glow part P_i. Note that, even though P_n is an aggregation of all previous parts, the training time is much lower due to zero backward propagation between $P_0, P_1 \dots P_{n-1}$.

Sequential Training Because the training of a part is not affected by training future parts in our local optimization setup, it seems we could merely train each part greedily where each Glow part is trained for a fixed number of epochs followed by the next Glow part and so on. In our initial experiments, we found that this sequential training led to significant instability in training the second or third part. In fact, after training the first part, the optimization for the second part diverged quickly. We believe that this behavior is observed due to the instability in the overall model where one part has converged reasonably whereas the other is completely untrained.

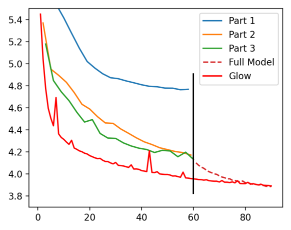
Alternating Epoch Training To rectify the problem of instability, we introduce an alternate Glow part training approach. As the name suggests, the Glow parts are alternatively trained using weights of the previous epoch which proved to be more resistant and maintains model training stability. In both cases (Sequential and Alternative training), after the initial training phase, we train the entire model with end-to-end backpropagation to double-check if our model has converged to a bad local minimum that would be hard to escape.



(a) Glow (ID=1) with 2-part split-by-blocks (ID=2)



(b) Glow (ID=1) with 2-part split-across-blocks (ID=3)



(c) Glow (ID=4) with 3-part split-across-blocks (ID=5) with a lower learning rate

Figure 2: Our parts-based alternating training method (60 epochs) followed by training the full model for 30 epochs achieves nearly equal performance to training the original Glow model end-to-end but requires less wall-clock time and less communication between parts. Model IDs from Table 1. The vertical line represents the epoch that we train all parts together (i.e., end-to-end) to see if we can close the remaining performance gap.

Table 1: Comparison between Glow and our alternating training style in terms of test Bits Per Dimension and total training time for 90 epochs.

ID	Approach	No. Parts	Parts	LR	BPD	Time(mins)
1	Glow	1	1,2,3	1e-4	3.60	1026
2	Ours(By)	2	$1 \rightarrow 2, 3$	1e-4	3.62	735
3	Ours(Across)	2	$1_{1/2} \rightarrow 1_{1/2}, 2, 3$	1e-4	3.61	720
4	Glow	1	1,2,3	1e-5	3.89	1020
5	Ours(Across)	3	$1_{1/2} \rightarrow 1_{1/2}, 2_{1/2} \rightarrow 2_{1/2}, 3$	1e-5	3.89	647

4 Experiments

We compare both of our splitting approaches with two different learning rates (1e-4 and 1e-5) using alternating training with Glow [Kingma and Dhariwal, 2018] on CIFAR10. As Glow, we use bits per dimension(BPD) as the comparison metric and the model consists of 3 blocks each containing 32 flow layers. We report the training times 1 for our results to show the efficiency of using our method. In all our experiments, we use 2 NVIDIA V100 GPUs with a batch size of 256.

Our alternating training approach can be best described in Figure 2 where Glow parts are separately trained for the initial 60 epochs. As described in Section 3, gradients do not backward propagate through part 1 while training part 2. For another 30 epochs, we train a full model where we see our results in BPD differ in a negligible amount compared to Glow while using less time to train as seen in Table 1. We notice that the split-by-block training procedure (Figure 2a) can be behave more erratically with this learning rate while the split-across-block (Figure 2b) is more stable as might be expected because the split channel operation is included inside of the part. We suspect that careful tuning of the learning rates for each part could alleviate this issue. Finally, we give initial results for a 3-part training via split-across-blocks in Figure 2c using a smaller learning rate to help optimization stability. From our experiments, we observe that alternative training strategy on split across block saves at least 30% of the time required for training without using full end-to-end backpropagation. Additionally, we emphasize that our approach also reduces the backward communication needed between the part optimizations, which could be useful in a distributed setting.

Our initial results give evidence that it may be possible to split the training of large flow models into parts that can be optimized separately. This could provide a new perspective to training large flow models that reduces the training time and the communication complexity between parts (e.g., in a distributed computing environment). Yet, many challenges and questions remain. For example, how many parts can we split the model into? Is the last end-to-end training required to get comparable performance or could this last stage be removed altogether as in Löwe et al. [2019]? Is there a natural way to set the learning rates for each part to ensure the stability of the optimization? We hope this work stirs discussion on these possibilities and suggests practical alternatives to end-to-end backpropagation.

References

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. 2017. URL `https://arxiv.org/abs/1605.08803`.

J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. volume 97 of *Proceedings of Machine Learning Research*, pages 2722–2730, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/ho19a.html`.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10215–10224. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf`.

A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

S. Löwe, P. O'Connor, and B. Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, pages 3039–3051, 2019.

X. Ma, X. Kong, S. Zhang, and E. Hovy. Macow: Masked convolutional generative flow. In *Advances in Neural Information Processing Systems 33, (NeurIPS-2019)*. Curran Associates, Inc., 2019.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows, 2016.