Multi-Scale Continuous Normalizing Flows

Anonymous Authors

Abstract

We introduce a multi-scale variant of Continuous Normalizing Flows, and explore the computation of likelihood values. We also introduce a Wavelet-version of the model. However, we find that this formulation is flawed in the computation of BPD, and explore ways to alleviate this problem.

Introduction

Reversible generative models derived through the use of the change of variables technique (Dinh et al., 2017; Kingma & Dhariwal, 2018; Ho et al., 2019; Yu et al., 2020) are growing in interest as alternatives to generative models based on Generative Adversarial Networks (GANs) (Goodfellow et al., 2016) and Variational Autoencoders (VAEs) (Kingma & Welling, 2013). A change of variables approach facilitates the transformation of a simple known probability distribution such as Gaussian noise into a more complex model distribution, such as images. Reversible generative models using this technique are attractive because they enable efficient density estimation, efficient sampling, and admit exact likelihoods to be computed. A promising variation of the change-of-variable approach is based on the use of a continuous time variant of normalizing flows (Chen et al., 2018; Grathwohl et al., 2019), which uses an integral over continuous time dynamics to transform a base distribution into the model distribution. This approach uses ordinary differential equations (ODEs) specified by a neural network, or Neural ODEs.

In this work, we consider a direct multi-resolution approach to continuous normalizing flows. While state-of-the-art GANs and VAEs exploit the multi-resolution properties of images, and recently top performing methods also inject noise at each resolution (Brock et al., 2019; Shaham et al., 2019; Karras et al., 2020; Vahdat & Kautz, 2020), only recently have normalizing flows exploited the multi-resolution properties of images, using wavelets (Yu et al., 2020). We use Continuous Normalizing Flows (CNF) in a multi-resolution fashion to generate an image at finer resolutions conditioned on the immediate coarser resolution image. A high-level view of our approach is shown in Figure 1.

2. Our method

Since images naturally exhibit structure in resolution, images can be decomposed into representations at multiple resolutions. We take advantage of this property by first decomposing an image in resolution space i.e. into a series of images at coarser resolutions : \( (x_0, x_1, \ldots, x_S) = x_{s \leq S} \).

In this work, we consider a direct multi-resolution approach to continuous normalizing flows. While state-of-the-art GANs and VAEs exploit the multi-resolution properties of images, and recently top performing methods also inject noise at each resolution (Brock et al., 2019; Shaham et al., 2019; Karras et al., 2020; Vahdat & Kautz, 2020), only recently have normalizing flows exploited the multi-resolution properties of images, using wavelets (Yu et al., 2020).

2.1. Normalizing Flows

We wish to train a generative model on a multi-resolution set of true images, i.e. find a probability distribution \( p(x_{s \leq S}) = p(x_0, x_1, \ldots, x_S) \) that matches the true data distribution. Normalizing flows (Tabak & Turner, 2013; Jimenez Rezende & Mohamed, 2015; Dinh et al., 2017; Papamakarios et al., 2019; Kobyzev et al., 2020) are good candidates for such a model, as they are probabilistic generative models that perform exact likelihood estimates, and can be run in reverse to generate novel data from the model’s distribution. This allows model comparison and measuring generalization to unseen data. Normalizing flows are trained...
Multi-Scale Continuous Normalizing Flows

We now wish to map the joint distribution of multi-resolution images $x$ to an input image $z$, the change-of-variables formula provides the likelihood of the image under this transformation as:

$$
\log p(x) = \log p(z) + \log \left| \det \frac{dz}{dx} \right| \quad (1)
$$

$\log p(z)$ is computed as the log probability of $z$ under the noise distribution (typically standard Gaussian).

### 2.2. Joint multi-resolution representation

We depart from the typical usage of the multi-resolution representation of images by observing that had we not known that the images at different resolutions could be derived from one fine image, we now have a joint distribution over all possible images at all resolutions. Suppose $x$ and $y$ are two different images. Under this joint multi-resolution distribution, $(x_{S-2}, x_{S-1}, x_S)$ and $(y_{S-2}, y_{S-1}, y_S)$ are valid multi-resolution images, but so are $(y_{S-2}, x_{S-1}, y_S)$ and $(x_{S-2}, y_{S-1}, x_S)$. It so happens that our real data distribution of multi-resolution images are those multi-resolution data points that are correlated in resolution space. This is equivalent to the fact that among all possible single-resolution images, only those that have correlated pixels in width and height are real/natural images, as opposed to noise images without any correlation among pixels.

### 2.3. Multi-Resolution Normalizing Flows

We now wish to map the joint distribution of multi-resolution images $x_{s \leq S}$ to “joint” multi-resolution noise $z_{s \leq S}$. In this case, the multi-resolution change-of-variables formula is:

$$
\log p(x_{s \leq S}) = \log p(z_{s \leq S}) + \log \left| \det \frac{\partial z_{s \leq S}}{\partial x_{s \leq S}} \right| \quad (2)
$$

The multi-resolution structure of the data results in a simplification of the calculation of the Jacobian determinant.

To illustrate this, choose a non-redundant basis of multi-resolution variables such that $z_s$ at any resolution is linearly independent of $x_{s+j}, j > 0$ at finer resolutions. This leads to the following block lower triangular structure in the variables:

$$
\log p(x_{s \leq S}) = \sum_{s=0}^{S} \log p(z_s) + \log \left| \begin{array}{ccc}
\frac{\partial z_s}{\partial x_0} & \cdots & 0 \\
\frac{\partial z_s}{\partial x_1} & \ddots & \vdots \\
\vdots & \ddots & \ddots \\
\frac{\partial z_s}{\partial x_{s-1}} & \cdots & \frac{\partial z_s}{\partial x_S}
\end{array} \right| 
$$

$$
= \sum_{s=0}^{S} \left( \log p(z_s) + \log \left| \frac{\partial z_s}{\partial x_s} \right| \right) \quad (3)
$$

We train a normalizing flow at each resolution to compute the likelihood of the image up to that resolution using Equation 3. This allows us to learn normalizing flows at each resolution independently, and in parallel.

Since the Jacobian determinant is a (block) lower triangular matrix, the non-zero off-diagonal elements don’t contribute to the final log probability. Hence, we can freely condition each normalizing flow on the coarser images, by treating the coarser images as independent variables. This allows us to learn only the higher-level information at each resolution. We use this to our advantage, and train each normalizing flow $g_s$ between $x_s$ and $z_s$ conditioned on the immediate coarser $x_{s-1}$ making a Markov assumption:

$$
z_0 = g_0(x_0); \quad z_s = g_s(x_s \mid x_{s-1}) \quad \forall s > 0 \quad (4)
$$

Hence, equation 3 can be rewritten as:

$$
\log p(x_{s \leq S}) = \log \mathcal{N}(g_0(x_0); 0, I) + \log \left| \det \frac{\partial g_0}{\partial x_0} \right|
$$

$$
+ \sum_{s=1}^{S} \left( \log \mathcal{N}(g_s(x_s \mid x_{s-1}); 0, I) + \log \left| \det \frac{\partial g_s}{\partial x_s} \right| \right) \quad (5)
$$

### 2.4. Multi-Resolution Continuous Normalizing Flows

We choose to use Continuous Normalizing Flows at each resolution (CNF) (Chen et al., 2018; Grathwohl et al., 2019), since they have recently been shown to effectively model image distributions using a fraction of the number of parameters typically used in normalizing flows (and non-flow-based approaches). At each resolution, each CNF $g_s$ transforms its state (say $v(t)$) using a Neural ODE (Chen et al., 2018) with neural network $f_s$:

$$
v(t_1) = g_s(v(t_0) \mid c) = v(t_0) + \int_{t_0}^{t_1} f_s(v(t), t, c) \, dt \quad (6)
$$

Chen et al. (2018); Grathwohl et al. (2019) proposed an instantaneous variant of the change-of-variables formula CNFs, which expresses the change in log-probability of the state of the Neural ODE i.e. $\Delta \log p_v$ as a differential equation:

$$
\Delta \log p_v(t_0 \rightarrow v(t_1)) = - \int_{t_0}^{t_1} \text{Tr} \left( \frac{\partial f_s}{\partial \nabla v(t)} \right) \, dt \quad (7)
$$

Hence, the ODE solver solves for the augmented state with the above differential, to obtain both the final state as well as the change in log probability simultaneously. Thus, the log probability at each resolution in eqs. (3) and (5) can be computed as:

$$
\log p(x_{s \leq S}) = \sum_{s=0}^{S} \left( \log p(z_s) + \Delta \log p_{x_{s-1} \rightarrow z_s} \right) \quad (8)
$$
We call this model MSFlow-Image.

In general, at each resolution (except the coarsest), the image \( x_s \) could first be converted to another representation \( y_s \) using a suitable orthogonal bijective transformation \( T \) from \( x_s \) to \( y_s \) so that \( \Delta \log p_{x_s \rightarrow y_s} = 0 \):

\[
\log p(x_s \leq S) = \sum_{s=0}^{S} \left( \log p(z_s) + \Delta \log p_{y_s \rightarrow z_s} + \Delta \log p_{x_s \rightarrow y_s} \right)
\]

In the simplest case, \( y_s = x_s \), which is MSFlow-Image.

A more complex orthogonal transform to use is the Haar wavelet transform, we call this model Multi-Scale Continuous Normalizing Flow - Wavelet (MSFlow-Wavelet). At each resolution, \( x_s \) is transformed into a composition of the 3 wavelet coefficients \( w_s \), and the coarser version \( x_{s-1} \), i.e. \( y_s = (w_s, x_{s-1}) \). In this case, the conditioning becomes more obvious: each CNF maps the wavelet coefficients \( w_s \) to a noise sample \( z_s \) conditioned on \( x_{s-1} \) (see Figure 2), similar to WaveletFlow(Yu et al., 2020) which builds on Glow (Kingma & Dhariwal, 2018).

Figure 2. Architecture of MSFlow-Wavelet.

Training: The overall model is trained to maximize the log-probability of the joint multi-resolution image, given by Equation 8 as the sum of the likelihoods of the images at each resolution. Equivalently, our model is trained to minimize the Bits-per-dimension (BPD) of the image at finest resolution \( S \) with \( D_S \) pixels:

\[
\text{bpd}(x_s \leq S) = \frac{-\log p(x_s \leq S)}{D_S \log 2} = \frac{-1}{D_S \log 2} \left[ \sum_{s=0}^{S} (\log p(z_s) + \Delta \log p_{x_s \rightarrow z_s}) \right]
\]

Since each CNF \( g_s \) independently models the conditional distribution of the image at that resolution, we train each \( g_s \) to minimize each \( \text{bpd}(x_s \leq S) \) step by step from the coarsest resolution (\( s = 0 \)) to the finest resolution (\( s = S \), having frozen \( g_j : j \neq s \)).

We use FFJORD (Grathwohl et al., 2019) as the baseline model for our CNFs. In addition, to speed up the training of FFJORD models by stabilizing the learnt dynamics, FFJORD RNODE (Finlay et al., 2020) introduced two regularization terms: the kinetic energy of the flow and the Jacobian norm. STEER (Ghosh et al., 2020) introduced temporal regularization by making the final time of integration stochastic.

Generation: Assuming each \( g_s \) is invertible (which CNFs are), we may then generate images using ancestral sampling: we first sample \( z_s \)'s from a latent noise distribution, and transform them backwards into image space progressively from coarser to finer resolutions through the CNFs:

\[
\begin{align*}
&x_0 = g_0^{-1}(z_0) \\
&x_s = T^{-1}(y_s) = T^{-1}(g_s^{-1}(z_s \mid x_{s-1})) \quad \forall \: s > 0
\end{align*}
\]

3. Related work

Several prior works on normalizing flows (Kingma & Dhariwal, 2018; Song et al., 2019; Ma et al., 2019; Yu et al., 2020) build on RealNVP (Dinh et al., 2017). Although they achieve great results in terms of BPD and image quality, they nonetheless report results from significantly higher parameters and several GPU hours for training.

Our MSFlow-Wavelet model is quite similar to the recently published WaveletFlow(Yu et al., 2020). However, WaveletFlow builds on the Glow (Kingma & Dhariwal, 2018) architecture, while ours builds on CNFs (Grathwohl et al., 2019; Finlay et al., 2020). Moreover, WaveletFlow applies certain techniques to obtain better samples from its model. We have so far not used such techniques for generation, but they can potentially help generate better samples from our models.

4. Experimental results

We train MSFlow-Image and MSFlow-Wavelet models on the CIFAR10 (Krizhevsky et al., 2009) dataset at finest resolution of 32x32, and the ImageNet (Deng et al., 2009) dataset at 32x32, 64x64, 128x128. We build on top of the code provided in (Finlay et al., 2020). In all cases, we train using only one NVIDIA V100 GPU with 16GB.

Ablation study on regularizers: We perform an ablation study using the two regularizations mentioned above: with/without RNODE (Finlay et al., 2020), with/without STEER (Ghosh et al., 2020). We find that consistently in all cases, FFJORD RNODE achieves superior BPD in lesser time. In some cases, FFJORD fails to train.

Ablation study on resolutions: We train models with varying number of total resolutions. Increasing the number of

\(^1\)https://github.com/cfinlay/ffjord-mode
Multi-Scale Continuous Normalizing Flows

Table 1. Unconditional image generation metrics (lower is better in all cases): number of parameters in the model, bits-per-dimension, time (in hours). Most previous models use multiple GPUs for training, all our models were trained on only one NVIDIA V100 GPU. † As reported in (Ghosh et al., 2020). * FFJORD RNODE (Finlay et al., 2020) used 4 GPUs to train on ImageNet64. ’x’: Fails to train. Blank spaces indicate unreported values.

<table>
<thead>
<tr>
<th></th>
<th>CIFAR10</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARAM</td>
<td>BPD</td>
<td>TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-scale Continuous Normalizing Flow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFJORD (Grathwohl et al., 2019)</td>
<td>3.40</td>
<td>&gt;5days†</td>
<td>x</td>
</tr>
<tr>
<td>FFJORD RNODE (Finlay et al., 2020)</td>
<td>3.38</td>
<td>31.84</td>
<td>x</td>
</tr>
<tr>
<td>FFJORD + STEER (Ghosh et al., 2020)</td>
<td>1.36M</td>
<td>3.40</td>
<td>86.34</td>
</tr>
<tr>
<td>FFJORD RNODE + STEER (Ghosh et al., 2020)</td>
<td>3.397</td>
<td>22.24</td>
<td>2.35</td>
</tr>
<tr>
<td>2-scale FFJORD</td>
<td>1.69</td>
<td>16.36</td>
<td>x</td>
</tr>
<tr>
<td>2-scale FFJORD RNODE</td>
<td>0.48M</td>
<td>2.04</td>
<td>18.76</td>
</tr>
<tr>
<td>2-scale FFJORD + STEER</td>
<td>1.74</td>
<td>18.43</td>
<td>x</td>
</tr>
<tr>
<td>2-scale FFJORD RNODE + STEER</td>
<td>1.54</td>
<td>21.51</td>
<td>x</td>
</tr>
<tr>
<td>3-scale FFJORD</td>
<td>1.42</td>
<td>19.95</td>
<td>x</td>
</tr>
<tr>
<td>3-scale FFJORD RNODE</td>
<td>0.48M</td>
<td>1.21</td>
<td>21.09</td>
</tr>
<tr>
<td>3-scale FFJORD + STEER</td>
<td>1.44</td>
<td>23.44</td>
<td>x</td>
</tr>
<tr>
<td>3-scale FFJORD RNODE + STEER</td>
<td>1.54</td>
<td>21.51</td>
<td>x</td>
</tr>
<tr>
<td>4-scale FFJORD</td>
<td>1.18</td>
<td>17.73</td>
<td>x</td>
</tr>
<tr>
<td>4-scale FFJORD RNODE</td>
<td>0.64M</td>
<td>1.73</td>
<td>17.08</td>
</tr>
<tr>
<td>4-scale FFJORD + STEER</td>
<td>1.44</td>
<td>17.60</td>
<td>x</td>
</tr>
<tr>
<td>4-scale FFJORD RNODE + STEER</td>
<td>1.18</td>
<td>65.6</td>
<td>65.6</td>
</tr>
<tr>
<td>5-scale FFJORD</td>
<td>1.55</td>
<td>19.42</td>
<td>x</td>
</tr>
<tr>
<td>5-scale FFJORD RNODE</td>
<td>0.81M</td>
<td>1.24</td>
<td>20.52</td>
</tr>
<tr>
<td>5-scale FFJORD + STEER</td>
<td>1.29M</td>
<td>3.87</td>
<td>10.73</td>
</tr>
</tbody>
</table>

Hence, to find the likelihood of \(x_S\) under our MSFlow-Image model, the likelihood of \(x_{s \leq S}\) needs to be marginalized over the entire subspace of lower resolution images. This is intractable. To make it tractable, we could approximate this marginal using Monte Carlo integration, by sampling multiple lower-resolution images and summing over the respective joint likelihoods. Inevitably, this leads to much greater BPD values than the ones reported in Table 1. Hence, Table 1 is not a fair comparison to make, except for the MSFlow-Wavelet rows.

6. Conclusion

We have presented a Multi-Resolution approach to Continuous Normalizing Flows, and performed exact likelihood calculations on several benchmark datasets of images by training on a single GPU in lesser time with a fraction of the number of parameters of other competitive models. However, we found that our formulation is fundamentally flawed in the computation of BPD for a single image. We explored ideas over how to fix this issue. We found that formulations similar to the Wavelet formulation which ensure the consistency of coarsened images with respect to the finest image can help alleviate this problem.

5. Fundamental flaw

However, we note that there is a fundamental flow to this calculation of BPD: we calculated the BPD of \(x_{s \leq S}\), while prior works report the BPD of \(x_S\). This implies that our model maps the joint distribution of images to joint noise, meaning our model includes images whose coarser versions do not correspond to the finest image. This does not apply to our MSFlow-Wavelet models since the wavelet formulation ensures the consistency of coarser images with respect to the fine image.

total resolutions consistently improves BPD across models with the same number of parameters per resolution, except in the case of MSFlow-Wavelet where we see the opposite case.

Progressive training: Since each resolution can be trained independently, we can train an MSFlow-Image model on ImageNet128 by training only the finest resolution (128x128) conditioned on 64x64 images for 1 epoch, and then attach that to a 4-resolution model trained on ImageNet64 from scratch. This 5-resolution ImageNet128 model gives a BPD of 1.13.
Multi-Scale Continuous Normalizing Flows

References


