FlowGrad: Controlling the Output of Generative ODEs with Gradients

Xingchao Liu¹, Lemeng Wu¹, Shujian Zhang¹, Chengyue Gong¹, Wei Ping², Qiang Liu¹
¹University of Texas at Austin  ²NVIDIA
{xcliu,lmwu,szhang19,cygong}@utexas.edu, wping@nvidia.com, lqiang@cs.utexas.edu

Abstract

Generative modeling with ordinary differential equations (ODEs) has achieved fantastic results on a variety of applications. Yet, few works have focused on controlling the generated content of a pre-trained ODE-based generative model. In this paper, we propose to optimize the output of ODE models according to a guidance function to achieve controllable generation. We point out that, the gradients can be efficiently back-propagated from the output to any intermediate time steps on the ODE trajectory, by decomposing the back-propagation and computing vector-Jacobian products. To further accelerate the computation of the back-propagation, we propose to use a non-uniform discretization to approximate the ODE trajectory, where we measure how straight the trajectory is and gather the straight parts into one discretization step. This allows us to save $\sim 90\%$ of the back-propagation time with ignorable error. Our framework, named FlowGrad, outperforms the state-of-the-art baselines on text-guided image manipulation. Moreover, FlowGrad enables us to find global semantic directions in frozen ODE-based generative models that can be used to manipulate new images without extra optimization.

1. Introduction

Controllable generation is very important for image editing [2,11,43], text-guided image manipulation [19,30,33], etc.. Traditionally, we use GAN and optimize the latent embedding with the desirable objective functions [2,7,11,29,31,38,49]. But the disadvantage is that, it is difficult to embed the image into the GAN space honestly and the performance is limited by the pre-trained GANs, which suffer from training instability and mode collapse.

Recently, diffusion models (or stochastic differential equation (SDE)-based generative models) has been popular [9,12,30,39,40,42,45], and there has been a number of works on controlled generation based on diffusion models, such as [9,28]. But due to the diffusion noise, it is hard to accurately control the output, especially when it comes to optimizing a complex loss function such as CLIP [35]. To achieve controlled generation, existing methods either requires training a noised version of the guidance [9,30,40], or fine-tuning the whole diffusion model [13,19].

In contrast, ordinary differential equation (ODE)-based generative models represent a simpler alternative than diffusion without involving the diffusion noise and the Ito calculus machinery. Recently, it has been shown that 1) ODEs can be trained directly without resorting to SDEs, and 2) ODE can perform comparable or even better than SDE [16,21,36,44].

Due to the deterministic nature of ODEs, they form an ideal model for controlled generation, as they enjoy both the rich latent space as SDEs and the explicit optimization framework as GANs. The goal of this work is to fully explore its potential in terms of controlled generation, with unconditioned pre-trained ODEs. Technically, 1) We present a simple way to control the output of ODE-based deep generative models with gradients; 2) We present a novel strategy to speedup the gradient computation by explore the straightness of ODE trajectories. By measuring the straightness at each time step during the simulation with Euler discretization, we can approximate the ODE trajectory with a few-step non-uniform discretization, and consequently reduce a great amount of time in back-propagation.

Our fast gradient computation scheme, named FlowGrad, allows us to efficiently control the generated contents of ODEs with any differentiable loss functions. In particular, we test FlowGrad on a challenging objective function, the CLIP loss, to manipulate user-provided images with text prompts. Moreover, by optimizing a set of training images together, FlowGrad can find semantically meaningful global directions in pre-trained ODE models, which allow manipulating new images for free. Equipped with advanced ODE-based generative models, FlowGrad outperforms state-of-the-art CLIP-guided diffusion models and GANs.

2. Background

In this section, we introduce background knowledge of ODE-based generative models.
2.1. Probability Flows

Given i.i.d. samples $D = \{x^{(i)}\}_{i=1}^N$ from a data distribution $\pi_{\text{data}}$, a probability flow can be effectively learned by training a velocity field $v_\theta(x, t)$ of an ordinary differential equation, which is indexed by a continuous time variable $t \in [0, T]$, to map a simple distribution $\pi_0$ to the complex target distribution $\pi_{\text{data}}$. Without loss of generality, we assume $T = 1$. Probability flows shift the initial distribution $\pi_0$ in a deterministic way,

$$dx_t = v_\theta(x_t, t)dt, \quad x_0 \sim \pi_0,$$

Traditionally, $v_\theta$ can be trained by the adjoint method [10] with the extra cost of solving an ODE during training. Later, researchers propose to extract probability flows from learned diffusion models as a way to save computation [39, 40]. Recently, a series of methods focus on directly designing the target velocity field and learning $v_\theta$ by regressing the target field [3, 21, 44].

Once $v_\theta$ is learned, one can simply sample from the ODE to get $x_1$ as draws from the target distribution $\pi_1$ with $N$-step Euler solver,

$$x_{(k+1)/N} = x_{k/N} + \frac{1}{N} v_\theta(x_{k/N}, k/N),$$

where $k \in \{0, 1, \ldots, N - 1\}$, $x_0 = x_{0/N}$ is a random sample from $\pi_0$ and $x_1 = x_{N/N}$ is the generated data. The number of discretization steps, $N$, determines the closeness of the simulated trajectory (2) and the continuous ODE trajectory (1). When $N \to \infty$, the simulated trajectory (2) has approaches the same end point as the continuous one. In practice, it is often found that an appropriate choice of $N$ for existing probability flow ODEs ranges from 20 to 200.
Representative ODE-based Generative Models  Existing ODE-based generative models can be characterized by the same framework \[23, 40, 42\]. Let \( X_t \) be any differentiable interpolation of \( x_0 \sim \pi_0 \) and \( x_1 \sim \pi_{\text{data}} \), then, \( v \) can be learned by minimizing

\[
\mathbb{E}_{x_t \sim \text{Uni}(0,1)} \left[ \left\| \dot{X}_t - v_{\theta}(X_t, t) \right\|^2 \right],
\]

where \( \dot{X}_t \) denotes the time derivative of \( X_t \). Rectified flow \[21\] is the special case when \( X_t = t x_1 + (1 - t) x_0 \), and DDIM \[39, 40\] corresponds to \( X_t = \alpha_t x_1 + \beta_t x_0 \) with \( x_0 \) a Gaussian distribution, \( \alpha_t = \exp(-5\sigma(1 - t)^2 - 0.05(1 - t)) \) and \( \beta_t = \sqrt{1 - \alpha_t^2} \).

**Straightness and Fast Simulation** The \( N \) needed to accurately discretize and simulate the ODE depends on how straight the trajectories are. In particular, if the trajectories of the ODE is straight, that is straight the trajectories are. In particular, if the trajectories accurately discretize and simulate the ODE depends on how straight trajectories. Although \[21\] proposes a reflow procedure to straighten the ODE trajectory and allow generation with very small \( N \), it still needs a curved trajectory to generate high-quality images. For these typical non-straight ODE trajectories, a small \( N \) leads to non-realistic blurry generation.

### 2.2. Encoding and Latent Space

The ODE model can be viewed as an auto-encoder, in which each data point \( x_1 \) can be encoded to \( x_0 \), and the decoding \((x_0 \rightarrow x_1)\) and encoding \((x_1 \rightarrow x_0)\) can be simply realized by solving the ODE forwardly, and backwardly. Specifically, with \( N \)-step Euler solver, we can encode \( x_1 \) by

\[
x_{k/N} = x_{(k+1)/N} - \frac{1}{N} v_{\theta}(x_{(k+1)/N}, (k + 1)/N),
\]

Encoding/Decoding with ODEs is extremely simple with nearly perfect reconstruction. This avoids the complicated encoding strategy and imperfect reconstruction in GAN inversion \[1, 2, 41\], making ODEs favorable for real image editing.

### 3. FlowGrad

In this section, we introduce how to efficiently back-propagate the gradient through the trajectory of an ODE, and provide acceleration schemes to effectively decrease the computational cost.

#### 3.1. The Optimal Control Framework and Challenges

Beyond unconditioned generation, we would like to directly control the generated contents given by the ODE from a starting point \( x_0 \). To achieve this, we can define a differentiable loss function \( L \) to control the end point \( x_1 \) towards our expectation by minimizing the following optimal control problem,

\[
\min_{u} \mathcal{L}(x_1) + \lambda \int_{0}^{1} \| u(t) \|^2 dt,
\]

s.t. \( x_1 = x_0 + \int_{0}^{1} (v_{\theta}(x(t), t) + u(t)) dt \),

where \( u(t) \) is a control function. This continuous optimal control problem is intractable, since the velocity field \( v_{\theta} \), which is usually represented by a neural network, is complicated. Instead, we turn to the discretized version of this problem,

\[
\min_{u} \mathcal{J} := \mathcal{L}(x_1) + \lambda \sum_{k=0}^{N-1} \| u_{k/N} \|^2
\]

s.t. \( x_{(k+1)/N} = x_{k/N} + \frac{1}{N} (v_{\theta}(x_{k/N}, k/N) + u_{k/N}) \),

where \( k \in \{0, 1, \ldots, N - 1\} \) and we adopt \( N \)-step Euler discretization.

#### Challenges

A straightforward optimization method for problem \(7\) is to run the simulation with Euler discretization, compute the gradient for each \( u_{k/N} \) and perform gradient descent. However, compute the gradient \( \nabla u_{k/N} \mathcal{J} \) needs to back-propagate through a chain with \( N - k \) nested \( v_{\theta} \). Since \( v_{\theta} \) is a neural network, computing the gradient for the nested structure requires overwhelming GPU memory in current auto-differentiation framework, like PyTorch \[32\], when the depth is large. For example, with a Nvidia TITAN XP GPU with 12GB memories, we can only compute the gradients for \( k < 4 \) with direct back-propagation.

#### 3.2. Decomposition of the Back-propagation

To allow gradient computation for all the \( u_{k/N} \), we propose to decompose the gradient computation by leveraging the following fact,

\[
\nabla_{x_{k/N}} \mathcal{J} = (\nabla x_{(k+1)/N} \mathcal{J}) \cdot J_{\phi_{k}}(x_{k/N}),
\]

where \( \phi_{k}(x) = x + \frac{1}{N} (v_{\theta}(x, k/N) + u_{k/N}) \) and \( J_{\phi_{k}}(x_{k/N}) \) is the Jacobian of \( \phi_{k} \) at point \( x_{k/N} \). Typically, the Jacobian is costly to obtain in the auto-differentiation framework. Fortunately, Eq. \(8\) is a vector-Jacobian product, which is much easier to get than the whole Jacobian. For
example, in PyTorch, the vector-Jacobian product can be efficiently computed with the double-backwards trick [32], at the cost of two normal backwards. After the gradient at the end point, \( \nabla x_j J(x_1) \), is computed, we can get the gradient \( \nabla u_{k/N} J = \frac{1}{N} \nabla x_{(k+1)/N} J \) by iteratively applying Eq. (8). Then we can then minimize \( J \) with gradient descent,
\[
    u_{k/N} \leftarrow u_{k/N} - \alpha \nabla u_{k/N} J,
\]
where \( \alpha \) is the step size set by the user.

3.3. Acceleration: Using non-uniform discretization

Although in Section 3.2, we are able to solve the optimization problem (7) through decomposing the back-propagation, its time consumption is still unacceptable. To compute the gradients for all the \( u(k/N) \) and perform one gradient descent, it takes \( \sim 18s \) on TITAN XP with a pre-trained Rectified Flow ODE model on 256 \( \times \) 256 images when \( N = 100 \). In comparison, it only takes \( 7s \) to simulate the ODE trajectory and generate the image \( x_1 \) with Euler discretization.

For better efficiency in back-propagation, we turn to non-uniform discretization rather than naive Euler discretization. Non-uniform discretization simulates the ODE with the following equation,
\[
    x_{t_{j+1}} = x_{t_j} + (t_{j+1} - t_j)v_\theta(x_{t_j}, t_j),
\]
where \( G := \{ t_j \}_{j=0}^{N'} \) is a set of grid points that satisfies \( t_0 = 0 \) and \( t_{N'} = 1 \). If we can find \( G \) that contains only a few elements but small discretization error, then we can significantly reduce the number of applying Eq. (8) and hereby save time.

Inspired by the properties of straight ODEs, we propose to detect the straight parts on the ODE trajectory and approximate the trajectory with the non-uniform discretization for fast back-propagation. Formally, our method consists of three steps:

- **Step 1: ODE Simulation**: We run Euler discretization for \( N \) steps to generate the ODE trajectory as in Section 3.1 and save the velocities \( \{ v_k := v_\theta(x_{k/N}, k/N) \}_{k=0}^{N-1} \).

- **Step 2: Approximation with Non-uniform Discretization**: We define the straightness at time \( k/N \) as,
\[
    S(k/N) = \max(d(v_{k-1}, v_k), d(v_{k+1}, v_k)),
\]
where \( d(v_1, v_2) = ||v_1 - v_2||^2 / ||v_2||^2 \) measures the relative change in velocity. If \( S(k/N) \) is small, the velocity is almost invariant at time \( k/N \) and we can consider \( t \in [(k-1)/N, (k+1)/N] \) is ‘straight’. Using the straightness measurement, \( G \) can be constructed by,
\[
    t_0 = 0, \quad N(t_{j+1} - t_j) = \arg \min_m \left\{ \sum_{i=1}^{m} S(t_j + \frac{i}{N}) \geq \xi \right\},
\]
where \( 1 \leq m \leq N(1 - t_j) \) and stop when \( t_j = 1 \).

Here, \( \xi \) is a hyper-parameter to control the tolerance of straightness. As \( \xi \) increases, the minimal \( m \) becomes larger and the number of elements in \( G \) decreases. Thus, a large \( \xi \) can reduce the computational time at the cost of introducing more error. Note that all the element \( t_j \) in \( G \) also belongs to the grid points of Euler discretization \( \{ k/N \}_{k=0}^{N} \).

- **Step 3: Fast Back-propagation**: Now that the ODE trajectory can be represented by the new non-uniform discretization with only \( N' << N \) steps, we can compute the gradient for \( u_{t_j} \) with only \( N' \) times of vector-Jacobian product computation. Specifically, we have,
\[
    \nabla x_{t_j} J = (\nabla x_{t_{j+1}} J) \cdot J\phi'(x_{t_j}),
\]
where \( \phi'(x) = x + (t_{j+1} - t_j)(v_\theta(x, t_j) + u_{t_j}) \). And correspondingly, \( \nabla u_{t_j} J = (t_{j+1} - t_j) \nabla x_{t_{j+1}} J \).

Re-assignment After the computation of \( \{ u_{t_j} \}_{j=1}^{N'} \), we re-assign the values to each \( u_{k/N} \) to allow Euler discretization in the next iteration. Specifically, for \( k \in [N t_j, N t_{j+1}) \), we assign \( u_{k/N} \leftarrow u_{t_j} \), so that the result still holds when simulated with Euler discretization.

---

**Table 1. Average running time for computing the gradients for all the variables on the trajectory with a pre-trained Rectified Flow ODE and the same TITAN XP GPU.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Full BP</th>
<th>( \xi = 5e-3 )</th>
<th>( \xi = 1e-2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full BP</td>
<td>17.7</td>
<td>1.4</td>
<td>1.1</td>
</tr>
</tbody>
</table>

---

**Figure 2.** To accelerate the back-propagation, we approximate the trajectory generated by Euler discretization using non-uniform discretization. We can significantly decrease the number of applying Eq. (8) and hereby save time.
Algorithm 1 FlowGrad: An efficient algorithm for backpropagation and optimization with ODEs

Input: A pre-trained ODE velocity \( v_\theta \), a differentiable guidance for clean images \( \mathcal{L} \), number of Euler discretization steps \( N \), the number of optimization iterations \( M \), penalty coefficient \( \lambda \), threshold \( \xi \) and step size \( \alpha \).

Procedure:
1. Initialize all the variables \( \{u(k/N)\}_{k=0}^{N-1} \) to zero.
2. for \( i \) in 1 : \( M \) do
   // Simulate the PF ODE trajectory
   for \( k \) in 0 : (\( N-1 \)) do
     \( x_{(k+1)/N} = x_{k/N} + \frac{1}{N} \left( v_\theta(x_{k/N}, k/N) + u(k/N) \right) \)
   end for
   Save \( \{v_k := v_\theta(x_{k/N}, k/N)\}_{k=0}^{N-1} \).

3. // Approximate with Non-uniform Discretization
   for \( k \) in 0 : (\( N-1 \)) do
     \( S(k/N) = \max(d(v_{k-1}, v_k), d(v_{k+1}, v_k)) \)
   end for
   Construct \( G \) with Eq. (12).
   Compute gradient with Eq. (13).

4. // Update the variables
   Compute \( \{u_j\}_{j=0}^{N} \) and re-assign the values to \( u_{k/N} \).
   end for

4. Related Works

**GAN Inversion** Using the latent space of pre-trained GANs for image editing has attracted much attention and achieved impressive results [43]. The rich latent spaces learned by modern GANs allow high-quality modification on images of human faces, animals, natural scenes, etc. [2,7,11,29,31,38,49], even generate out-of-distribution images under language guidance [22,33]. However, due to the highly non-convex landscape of the latent space of pre-trained GANs, how to satisfactorily embed the images into the latent spaces is a difficult problem [1,15,43,48], which affects the later performance on editing the images.

**SDE and ODE-based Generative Models** Recently, diffusion models gains popularity because of their astonishing power in learning generative models on various domains, e.g., images, molecules and point clouds [9,12,20,23,25,37,40,42]. Unlike GANs, diffusion models do not have a one-to-one mapping between images and noises in the latent space. On the contrary, ODE-based models, either derived from pre-trained diffusion models [39,40] or directly learned from scratch [16,21,44], can map a latent code to a deterministic image, while enjoying the high generation quality like diffusion models. Moreover, by running the ODE solver in the reverse way, it is straightforward to find the corresponding latent code for a given image, with nearly perfect reconstruction, which makes it favorable over GAN inversion [40]. However, even equipped with acceleration methods like [4,26], both SDE and ODE-based models require more than 20 iterations to generate a high-quality image, making it impossible to naively migrate the optimization techniques for image editing in GAN inversion to these novel models.

**Controlled Generation with ODE and SDE-based Models** Because of the inherent difference between GANs and SDE/ODE-based models, a variety of new methods have been proposed for controlled generation with diffusion models [5,6,18,19,27,28,30,34,40,47]. In [9,30,40], a guidance model for noisy images is trained to guide the model in the intermediate time steps of SDE. Training the noised guidance model is time-consuming and expensive. [5,28] propose to edit or generate realistic images from coarse strokes or reference images, but their methods can only find images that are close in the Euclidean space thanks to the noise in SDEs. DiffusionCLIP [19] can manipulate images according to CLIP guidance with SDEs, but it needs to fine-tune the whole score function model for each text prompt. This causes problems in storage and time consumption. Our method can directly use guidance models pre-trained on clean images, and manipulate the images following text instructions with pre-trained frozen ODE models.

5. Experiments

We conduct a series of experiments on text-guided image manipulation to demonstrate the effectiveness of our FlowGrad. We apply FlowGrad on state-of-the-art ODE-based generative models, including Rectified Flow (RF), and Latent Diffusion Model (LDM) with DDIM. For all the experiments, we use the accelerated FlowGrad with \( \xi = 5e - 3 \) without further notification. Code is available at https://github.com/gnobitab/FlowGrad.

5.1. Text-guided Image Manipulation

Zero-shot image manipulation with text guidance is a desirable application for generative models after the emergence of Contrastive Language-Image Pretraining (CLIP) [35]. In this task, algorithms need to modify the provided images according to the text instruction without affecting the unrelated components on the image. Typical methods for this task, e.g., StyleCLIP [33], rely on GAN inversion. Recently, diffusion models are adopted for this task for their superiority in reconstruction [19,34], at the cost of re-training or fine-tuning existing diffusion models.

Here, we apply FlowGrad to this task. Given an image
Figure 3. We provide qualitative comparison between different text-guided image manipulation algorithms, including DiffusionClip [19], StyleCLIP [33] with e4e encoder [41], FlowGrad + Rectified Flow (RF) [21] and FlowGrad + Latent Diffusion Model (LDM) [36]. The input images are sampled from CelebA-HQ dataset. The unconditional RF is pre-trained on CelebA-HQ and the unconditional LDM is pre-trained on FFHQ. We observe: (1) DiffusionCLIP causes unexpected changes in the style and color; (2) even with the state-of-the-art e4e encoder and the strong StyleGAN2 generator, StyleCLIP generates images with noticeable change in identity and background; (3) ODEs can smoothly encode the input images and then edit the images by FlowGrad, without changing the unrelated elements. Note that FlowGrad uses frozen pre-trained ODEs.
Method & LPIPS (↓) & ID (↑) & CLIP (↑) \\
CG + RF [9] & 0.346 & 0.643 & 0.292 \\
CG + LDM [9] & 0.383 & 0.513 & 0.298 \\
DiffusionCLIP [19] & 0.398 & 0.659 & 0.285 \\
StyleCLIP [33] + e4e [41] & 0.359 & 0.704 & 0.267 \\
FlowGrad + RF & 0.302 & 0.737 & 0.299 \\
FlowGrad + LDM & 0.298 & 0.743 & 0.294 \\

Table 2. Quantitative comparison between different algorithms for editing images from CelebA-HQ. ‘CG’ refers to Classifier Guidance. We measure the LPIPS similarity and the identity similarity between the original image and the manipulated images to reflect faithfulness to the given image. Besides, we measure the Augmented CLIP score of the manipulated images to reflect closeness between the manipulated image and the text prompt. FlowGrad yields the highest faithfulness to both the provided image and the text prompt.

Table of interest $x_g$, our loss function is defined as,

$$L(x_1) = \eta s(x_1, T) + (1 - \eta)||x_1 - x_g||,$$

where $s(\cdot, \cdot)$ is the similarity score between image $x_1$ and text prompt $T$ given by the CLIP model, $||x_1 - x_g||$ penalizes large variations from the original image $x_g$, and $\eta = 0.7$ for all the experiments. To avoid adversarial generation, we adopt the Augmented CLIP score [22] as $s(\cdot, \cdot)$. The hyperparameter $\lambda$ is set to $1e^{-3}$. We exploit the official pre-trained models including Latent Diffusion Model (LDM) [36] on FFHQ [17] and LSUN Church [46], and Rectified Flow (RF) [21] on CelebA-HQ [24]. We generate images with DDIM for LDM. For LDM, the Euler discretization step $N = 200$; for RF, $N = 100$. We set the step size $\alpha = 10.0$ and the number of optimization iterations $M = 10$. We compare with state-of-the-art text-guided image manipulation baselines, StyleCLIP [33] and Diffusion-CLIP [19]. The details of the baseline configurations can be found in Appendix A. The qualitative results are shown in Figure 3, 4.

For quantitative comparison, we use the CelebA dataset, randomly sampled 1,000 images, and manipulate them with text guidance. The text guidance contains {old, sad, smiling, angry, curly hair}. As in [19], we measure the face identity similarity, LPIPS similarity and augmented CLIP score, to respectively show the closeness to the original face, the original image and the text prompt. The results are shown in Table 2. Comparison of the running time between different methods is reported in Table 4.

5.2. Identify Global Semantic Direction in Pre-trained ODEs

Beyond editing individual images, we expect to find global semantic direction $u$ that generalizes across for a wide range of images so that we can directly manipulate a new image without solving the optimization problem again. Global semantic directions can be found by optimizing the following loss function,

$$L = \sum_{i=1}^{n} \eta s(x_1^{(i)}, T) + (1 - \eta)||x_1^{(i)} - x_g^{(i)}||,$$

where $x_g^{(i)}$ are the training images. The global directions found in frozen pre-trained LDM-FFHQ and RF-CelebA are shown in Figure 6.
Figure 6. We show global directions found in frozen pre-trained ODEs with FlowGrad. We randomly generate 10 images as the training set, and demonstrate the effectiveness of the obtained global directions on other synthesized images.

5.3. Ablation Study

In this section, we perform ablation studies to justify the design choices of FlowGrad. The ablations studies are conducted on text-guided image manipulation tasks.

Threshold $\xi$ We examine the influence of the threshold $\xi$. We set $\xi = 0, 5e^{-3}, 1e^{-2}$, where $\xi = 0$ results in full back-propagation. Besides, we set $\xi = \sum_{i=1}^{N-1} S(i/N)$, which corresponds to the following one-step discretization,

$$x_1 = x_0 + \nu(x_0, 0).$$

We adopt RF-CelebA for this study. The results are shown in Figure 5b. We use the text prompt curly hair. We observe that one-step discretization brings large error, and $\xi < 1e^{-2}$ results in images with similar visual quality and loss.

Sampling step $N$ We examine the influence of reducing the number of sampling steps $N$. We use RF-CelebA. The results are shown in Figure 7. Smaller $N$ generates oversmoothed images and fails the manipulation task.

6. Conclusions

We propose FlowGrad, an efficient framework for controlled generation with ODE-based generative models using gradients. Our FlowGrad decomposes the back-propagation of the ODE trajectory, and compute the gradient with vector-Jacobian products. Moreover, by re-approximating the ODE trajectory with non-uniform discretization, FlowGrad can save 90% of the back-propagation time with small computation error. Experiments on text-guided image manipulation and global semantic direction detection shows the superiority of FlowGrad.

Acknowledgements

This research is supported by NSF CAREER1846421, SenSE2037267, EAGER-2041327, Office of Navy Research, and NSF AI Institute for Foundations of Machine Learning (IFML).
References


conference on computer vision and pattern recognition. pp. 2437–2445 (2020) 1, 5


[40] Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. In: International Conference on Learning Representations (2020) 1, 2, 3, 5, 11


A. More Details on Experiment Configuration

Here, we provide more details for reproducing our experiments.

Pre-trained Rectified Flow We acquire the pre-trained models and corresponding sampling scripts from the authors of [21]. The model is the same U-Net architecture as NCSN++ [40], which contains \( \sim 65 \) million parameters.

Pre-trained Latent Diffusion Models We acquire the pre-trained models of Latent Diffusion Models from their official open-sourced repository.\(^1\) The encoder and decoder are associated with the pre-trained velocity network. The size of the latent code for FFHQ is \( 3 \times 64 \times 64 \), and the size of the latent code for LSUN church is \( 3 \times 32 \times 32 \). To manipulate an image, we first use the encoder to encode an image into the latent space of LDM to get \( x_0 \), then use DDIM to generate the result \( x_1 \) in the latent space. \( x_1 \) is then decoded by the decoder to get the generated image. Since the decoder is differentiable, the gradient can be back-propagated through the decoder.

Baselines For DiffusionCLIP\(^2\) [19] and StyleClip\(^3\) [33], we adopted their official implementation from github without modification and followed their instructions to manipulate images with text prompts.

For CG, we used the same loss as FlowGrad, grid searched guidance magnitude over \([1, 10, 25, 50, 100]\), and reported the result with the lowest average loss. We compute the gradient of CG w.r.t. to the predicted \( x_1 \) in every discretization step. In RF, \( x_1 \) can be predicted by \( \hat{x}_1 = x_1 + (1 - t)v(x_1, t) \); In LDM-DDIM, \( \hat{x}_1 \) can be predicted by Eq. (12) in [39]. CG needs a guidance function \( \mathcal{L} \) that can take noisy images as input. In our implementation, we use the same \( \mathcal{L} \) that only works on clean images for classifier guidance as other methods for fair comparison. The update rule for classifier guidance is,

\[
x_{k+1} = x_k + \frac{1}{N}(v_\theta(x_k, k/N) - c\nabla_x \frac{k}{N}(\mathcal{L}(\hat{x}_1^{k/N}))),
\]

where \( c \) is a hyper-parameter that controls the magnitude of the gradient guidance.

Identity Similarity For the identity similarity metric in Table 2, we follow the previous works [19, 33] and use the pre-trained ArcFace network [8].

\(^1\)https://github.com/CompVis/latent-diffusion
\(^2\)https://github.com/gwang-kim/DiffusionCLIP
\(^3\)https://github.com/orpatashnik/StyleCLIP

Figure 8. Image manipulation with FlowGrad+RF-CelebA. The text prompt is \textit{curly hair}. Images directly generated from the non-uniform discretization are over-smooth and unreal, while the image generated with the re-assigned Euler discretization yields high-quality generation with curly hair.

Additional Details for Identifying Global Semantic Directions The number of training images \( n = 10 \) for the experiment. The hyper-parameter \( \lambda = 1e - 3 \) as in image manipulation. We use \( \alpha = 10.0 \) and \( M = 20 \) since we found the convergence is slower than manipulating one image. The training images are synthesized by the generative ODEs.

Re-assignment We found that, although a few-step non-uniform discretization accelerates back-propagation, images directly generated following that non-uniform discretization have low-quality, as shown in Figure 8.

The reasoning of re-assignment is as follows: assuming the ODE trajectory is nearly straight and velocity is nearly invariant in \([t_j, t_{j+1}]\), we have

\[
x_{t_{j+1}} = x_{t_j} + (t_{j+1} - t_j)(v_\theta(x_{t_j}, t_j) + u_{t_j}).
\]

Let us denote \( k_1 = Nt_j, k_2 = Nt_{j+1} \), we know

\[
x_{k_2/N} = x_{k_1/N} + (k_2/N - k_1/N)(v_\theta(x_{k_1/N}, k_1/N) + u_{t_j}).
\]

Using re-assignment, for \( k \in [k_1, k_2] \), we assign \( u_{k/N} = u_{t_j} \), so that if we simulate with Euler discretization, we have,

\[
x_{k_2/N} = x_{k_1/N} + \frac{1}{N} \sum_{k=k_1}^{k_2} (v_\theta(x_{k_1/N}, k_1/N) + u_{k/N}).
\]

Therefore, with re-assignment, the terminal \( x_{k_2/N} \) is not changed. With the complete \( N \)-step Euler discretization and re-assignment, we can generate high-quality images according to the guidance \( \mathcal{L} \).

B. Additional Quantitative Results

Quantitative Results on Image Manipulation with LSUN Church For quantitative comparison, we use
Table 3. Quantitative comparison between different algorithms for editing images from LSUN Church. ‘CG’ refers to Classifier Guidance. We measure the LPIPS similarity and the identity similarity between the original image and the manipulated images to reflect faithfulness to the given image. Besides, we measure the Augmented CLIP score of the manipulated images to reflect closeness between the manipulated image and the text prompt. FlowGrad yields the highest faithfulness to both the provided image and the text prompt.

<table>
<thead>
<tr>
<th>Method</th>
<th>LPIPS (↓)</th>
<th>CLIP (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG + LDM [9]</td>
<td>0.435</td>
<td>0.305</td>
</tr>
<tr>
<td>DiffusionCLIP [19]</td>
<td>0.382</td>
<td>0.274</td>
</tr>
<tr>
<td>StyleCLIP [33]+e4e [41]</td>
<td>0.502</td>
<td>0.287</td>
</tr>
<tr>
<td>FlowGrad + LDM</td>
<td>0.374</td>
<td>0.312</td>
</tr>
</tbody>
</table>

the LSUN Church dataset, randomly sampled 1,000 images, and manipulate them with text guidance. The text guidance contains \{wooden, colorful, stone, Byzantine\}. We measure the LPIPS similarity and augmented CLIP score, to respectively show the closeness to the original image and the text prompt. The results are shown in Table 3.

Comparison in Running Time We provide the total running time of different algorithms on the same TITAN XP GPU for image manipulation in Table 4.

<table>
<thead>
<tr>
<th>Method</th>
<th>Training New Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier Guidance [9]</td>
<td>-</td>
</tr>
<tr>
<td>DiffusionCLIP [19]</td>
<td>∼ 500s</td>
</tr>
<tr>
<td>StyleCLIP [33]+e4e [41]</td>
<td>-</td>
</tr>
<tr>
<td>FlowGrad + LDM</td>
<td>-</td>
</tr>
<tr>
<td>FlowGrad + RF</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4. The total running time of different algorithms on the same TITAN XP GPU for image manipulation on CelebA-HQ.

C. Additional Qualitative Results

We provide more qualitative results in the following figures, including multi-attribute manipulation and style manipulation.

D. Editing Molecules

Additionally, we applied FlowGrad to Equivariant Diffusion for Molecules (EDM)\(^4\) [14] to show its impact in molecule generation. We trained EDM on QM9 for 1000 epochs, drew 100 stable base molecules, then edited their atom positions to achieve lower size (the average distance

\(^4\)https://github.com/ehoogeboom/e3_diffusion_for_molecules

Figure 9. The global directions found by FlowGrad can be directly added together for multi-attribute image manipulation. In this experiment, we use apply the global directions \(u_{\text{eye-shadow}}\), \(u_{\text{angry}}\) and \(u_{\text{eye-shadow}} + u_{\text{angry}}\) to the same input image with Rectified Flow.

Figure 10. Image manipulation with FlowGrad for multi-attribute editing. In this experiment, we use LDM-FFHQ, and set our loss function as \(\mathcal{L}(x_1) = \eta_1 s(x_1, T_1) + (1 - \eta_1) s(x_1, T_2) + (1 - \eta_1)||x_1 - x_g||\), where \(\eta = 0.7\), \(T_1 = \text{smiling}\) and \(T_2 = \text{old}\). We vary the magnitude of \(\eta_1\) and find FlowGrad can interpolate between the two attributes.

Figure 11. We provide generated images from classifier guidance with LDM-FFHQ and LDM-Church. With a clean guidance function, classifier guidance cannot minimize the loss effectively, resulting in unnatural images. The text prompt for the first row is angry. The text prompt for the second row is stone.
between the atoms in the molecule), HOMO (the Highest Occupied Molecular Orbital energy), and LUMO (Lowest Unoccupied Molecular Orbital energy) using FlowGrad/CG. Like LDM, we use DDIM to transfer the diffusion process into an generative ODE. The Euler discretization step $N = 1000$. For editing HOMO and LUMO, we use the neural network predictors trained on the real molecules in QM9 as guidance. To ensure stability, we punished the deviation of the atoms from its original position in the base molecule.

The loss function for molecule editing is,

$$
\mathcal{L} := \mathcal{L}_{\text{property}}(M) + \frac{1}{N} \sum_{i=1}^{N} || p_i - \hat{p}_i ||,
$$

where $M$ is the molecule, $N$ is the number of atoms, $p_i$ is the position of the $i$-th atom and $\hat{p}_i$ is the original position of the $i$-th atom in the base molecule. The classes of the atoms are kept unchanged. In this experiment, we set the step size $\alpha = 5e^{-4}$ for FlowGrad. We keep the number of optimization iterations to $M = 10$ as before. For CG baseline, we grid searched the guidance magnitude over $c \in [1e^{-2}, 1e^{-3}, 1e^{-4}, 1e^{-5}, 1e^{-6}]$. We found large guidance magnitude ($c \geq 1e^{-3}$) sometimes causes numerical issues in the sampling process with CG and the loss underflows, so we pick from $c \leq 1e^{-4}$ with the lowest average loss.

The results are shown in Table 5. Compared to classifier guidance, molecules edited with FlowGrad had lower values on all the three properties.

<table>
<thead>
<tr>
<th>Method</th>
<th>size</th>
<th>HOMO</th>
<th>LUMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDM</td>
<td>3.34</td>
<td>-6.31</td>
<td>-0.204</td>
</tr>
<tr>
<td>CG+EDM</td>
<td>3.11</td>
<td>-6.50</td>
<td>-0.232</td>
</tr>
<tr>
<td>FlowGrad+EDM</td>
<td><strong>3.08</strong></td>
<td><strong>-6.58</strong></td>
<td><strong>-0.742</strong></td>
</tr>
</tbody>
</table>

Table 5. Molecule editing with EDM.