IFT6269 Project Final Report
Flow-based Generative Models: A Case Study of GLOW

Zhibin Lu

Université de Montreal

Yishi Xu

Université de Montréal

Tong Che

Université de Montréal

Ruixiang Zhang

Université de Montréal

Abstract

In this project final report, we review essential techniques in a family of generative models, flow-based generative models, with special focus on one state-of-the-art model, Glow(Diederik P. Kingma, 9 Jul 2018). Glow is a flow-based generative model which generates strikingly realistic looking images. In this report, we review the main ideas of flow-based generative model and how Glow particularly constructed its flow. We then reproduced the paper "Glow: Generative Flow with Invertible 1x1 Convolutions" with 3 different permutation methods. We conclude that flow-based generative model is a useful generative model and it has potential for improvement in various areas.

1. Introduction

Generative models is a hot topic in machine learning. In this report, we study flow-based generative model which is a state-of-the-art generative model that is different from GAN or VAE. We also reproduce Glow, a recent work of flow-based generative model, with experiments to show that this method can produce high quality synthetic images.

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

Flow-based generative models is first described in (Laurent Dinh, 2014). In the first part of this report, we illustrate the main ideas of this method. Then we show how Glow extended this idea and constructed its flow. For experiments we apply Glow on two small datasets, MNIST and CelebA, due to our limited computation resources. We also implemented 3 other permutation methods in the flow in addition to invertible 1x1 convolution, which is used by Glow. Glow achieves the highest generating performance among all the three variations.

2. Normalizing Flow

2.1. Prerequisite: Change of Variables

The normalizing flow uses change of variables theorem extensively. Following is a short description of change of variables formula:

$$z_i \sim p_i(z_i), z_{i-1} \sim p_{i-1}(z_{i-1})$$

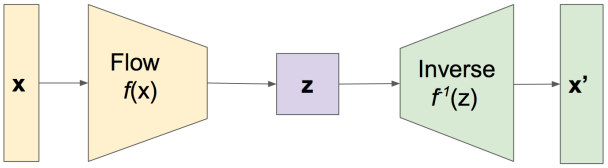
$$z_i = f_i(z_{i-1})$$

$$p_i(z_i) = p_{i-1}(f_i^{-1}(z_i))|det\frac{df_i^{-1}}{dz_i}|$$

2.2. Normalizing Flow

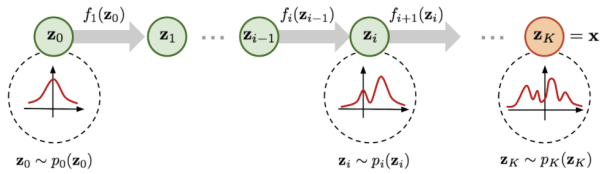
Flow-based generative models utilize invertible normalizing flows to do density estimation of $p_d(x)$ (the data-generating distribution). By using normalizing flow, the likelihood of input data become tractable and enables the optimization of the exact log-likelihood of input data. For example, let x be a high-dimensional vector with distribution $x \sim p(x)$. We can directly minimize the objective function using gradient descent. This is important for generative models because it makes the training part to simply minimize the negative log-likelihood of the input data.

A normalizing flow is a sequence of invertible functions maps a simple distribution, $p(z)$, to a more complicated distribution, $p(x)$. This sequence of transformations is called normalizing flow.



Given a latent variable $z \sim p_z(z)$ and original input $x \sim p_x(x)$, the generative process is to use a function g to transform the latent variable z to x, that is $x = g(z)$. We need g to be an invertible function so that the latent variable inference can be done by $z = g^{-1}(x)$. g is usually composed of a sequence of invertible functions f_i like below:

$$z \longleftrightarrow z_1 \longleftrightarrow z_2 \longleftrightarrow ... \longleftrightarrow z_{K-1} \longleftrightarrow x$$



(Weng, Dec. 2018)

By applying change of variables theorem, we can get log-likelihood of $p(z_i)$ as follows:

$$p_i(z_i) = p_{i-1}(f_i^{-1}(z_i))|det\frac{df_i^{-1}}{dz_i}| \quad (1)$$

$$= p_{i-1}(z_{i_1})|det\frac{df_i}{dz_{i-1}}|^{-1} \quad (2)$$

$$log\, p_i(z_i) = log\, p_{i-1}(z_{i-1}) - log\,|det\frac{df_i}{dz_{i-1}}| \quad (3)$$

If we apply the change of variable theorem to the entire sequence of functions $g = f_K \circ f_{K-1} \circ \cdots \circ f_1$, we will be able to tract the log-likelihood of $p(x)$ from $p(z)$:

$$x = f_K \circ f_{K-1} \circ \cdots \circ f_1(z_0) \quad (4)$$

$$log\, p(x) = log\, p_{K-1}(z_{K-1}) - log\,|det\frac{df_K}{dz_{K-1}}| \quad (5)$$

$$= log\, p_0(z_0) - \sum_{i=1}^{K} log\,|det\frac{df_i}{dz_{i-1}}| \quad (6)$$

According to the equations above, each transformation function f_i in the normalizing flow needs to be easily invertible and its Jacobian determinant needs to be easy to compute.

One way to choose transformation with simple Jacobian determinant is to choose transformations whose Jacobian $\frac{df_i}{dz_{i-1}}$ is a triangular matrix. The Jacobian determinants for these functions are simple:

$$log|det(\frac{df_i}{dz_{i-1}})| = sum(log|det(\frac{df_i}{dz_{i-1}})|)$$

3. Glow

Glow(Diederik P. Kingma, 9 Jul 2018) is a recently proposed flow-based generative model building on the NICE(Laurent Dinh, 2014) and RealNVP(Laurent Dinh, 2016). The basic idea is to construct a series of steps of flow, combined in a multi-scale architecture and each step of flow consists of actnorm followed by an invertible 1×1 convolution , finally followed by a coupling layer. Glow simplifies the architecture of NICE and RealNVP by replacing the reverse permutation operation on the channel ordering with invertible 1×1 convolutions.We briefly review the design details of each component in following sections.

3.1. Actnorm

Glow proposed a activation normalization layer called actnorm to perform an affine transformation of the activations using a scale and bias parameter per channel, to address the batch normalization(Ioffe & Szegedy, 2015) issues in RealNVP(Laurent Dinh, 2016). Batch normalization(Ioffe & Szegedy, 2015) is known to work poorly when the minibatch size is small. The parameters of actnorm are initialized so that the first minibatch of data have mean 0 and standard deviation 1 after actnorm. Actnorm is designed carefully with this data-dependent initialization mechanism to work well for small batch size settins, which is neccessary and common for training on large images.

3.2. Invertible 1×1 Convolution

In NICE(Laurent Dinh, 2014) and RealNVP(Laurent Dinh, 2016) the ordering of channels is reversed between layers of flow to make sure that all the data dimensions have a chance to be altered. Glow proposed to replace this fixed permutation with a learned invertible 1×1 convolution. A 1×1 convolution with equal number of input and output channels is a generalization of any permutation of the channel ordering.

Support the input of the 1×1 invertible convolution is a $h \times w \times c$ tensor and the trainable $c \times c$ weight matrix is $\mathbf{W}$, then the log-determinant of the invertible convolution is straightforward to compute:

$$\log\left|\det\left(\frac{d\,\text{conv2d}(\mathbf{h};\mathbf{W})}{d\,\mathbf{h}}\right)\right| = h \cdot w \cdot \log|\det(\mathbf{W})| \quad (7)$$

The weight matrix $\mathbf{W}$ is initialize as a random rotation matrix, having a log-determinant of 0, and the cost of computing $\det(\mathbf{W})$ is $\mathcal{O}(c^3)$ which is nearly same as the convolution layer, and this can be reduced from $\mathcal{O}(c^3)$ to $\mathcal{O}(c)$ by parameterizing $\mathbf{W}$ directly in its LU decomposition. Please refer to original Glow paper for more details on the LU decomposition parametrization.

3.3. Affine Coupling Layers

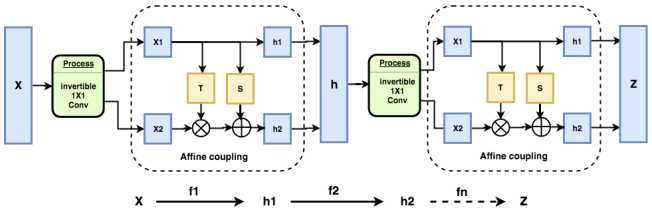
The affine coupling layer of Glow remains same design with previous works like NICE(Laurent Dinh, 2014) and RealNVP(Laurent Dinh, 2016) to construct an invertible, bijective and differentiable transformation function. The detailed construction of the transformation is shown in table 1. We briefly summarize the idea of this transformation as follows: In each bijection $f : x \rightarrow y, x, y \in R^D$, the input dimensions are split into two parts:

- The first d dimensions stay same.
- The second part, $d+1$ to D dimensions, undergo an affine transformation ("scale-and-shift") and both the scale and shift parameters are functions of the first d dimensions.

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \end{aligned} \tag{8}$$

The properties of invertible and tractable Jacobian determinant can be easily verified. By stacking a series of such transformations we can finally get a complicated enough transformation.

The following chart contains all the factors in the "flow" and is implemented in the code.



3.4. Multi-scale Architecture

The flow in Glow is combined with a multi-scale architecture. We demonstrate the details of this multi-scale architecture in figure 1. After the original input is subjected to the first flow operation(affine coupling layers), the output is the same size as the input. At this time, the input is split into two parts: $z1$ and $z2$, where $z1$ remains unchanged, and only z2 goes through next flow operation, and so on. The final output consists of $z1$, $z3$, and $z5$, the total size is the same as the input. The multi-scale operation of each step reduces the data size to half of the original value, which is obviously very effective for reducing the amount of calculation.

Description	Function	Reverse Function	Log-determinant
Actnorm.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot sum(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert \det(\mathbf{W}) \rvert$ or $h \cdot w \cdot sum(\log \lvert\mathbf{s}\rvert)$
Affine coupling layer.	$\mathbf{x}_a, \mathbf{x}_b = split(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = NN(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = concat(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = split(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = NN(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = concat(\mathbf{x}_a, \mathbf{x}_b)$	$sum(\log(\lvert\mathbf{s}\rvert))$

Table 1. The three main components of Glow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. (Diederik P. Kingma, 9 Jul 2018)

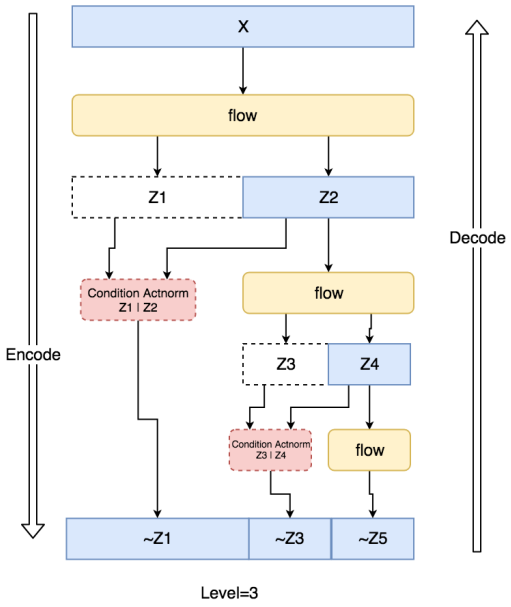


Figure 1. Multi-scale flow architecture.

Glow assumes that the three probability distributions at the top end in the chart are all Gaussian distribution, where the mean variance of $p(z1|z2)$ is calculated from $z2$, the mean variance of $p(z3|z4)$ is calculated from $z4$, and the mean variance of $p(z5)$ is directly Learn to come out. Their relationship is shown as follows:

$$p(z_1, z_3, z_5) = p(z_1|z_2)p(z_3|z_4)p(z_5)$$

$$\hat{z_1} = \frac{z_1 - \mu_{z_2}}{\sigma_{z_2}}$$

This method of joint conditional distribution can reduce

the amount of calculation during flow and also control the range of different dimensions of latent space of z, the joint conditional distribution has more flexibility than the independent distribution.

4. On permuting channels: NICE, RealNVP and Glow

In the "Flow" flow chart, it shows that if $h1$ and $h2$ do not change order in the affine coupling, $h1$ will have the unchanged $x1$ value after the coupling layer. In order to use all elements in x and to obtain the nonlinear conversion space z, three transformation methods were discussed in Glow (Diederik P. Kingma, 9 Jul 2018):

4.1. NICE: Reverse

The reverse method is proposed by NICE (Laurent Dinh, 2014). It simply reverse the order of $h1$, $h2$ before merging into H.

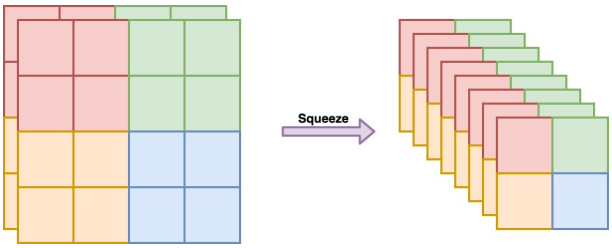
4.2. RealNVP: Shuffle

The shuffle method is proposed by RealNVP (Laurent Dinh, 2016), the H vector is shuffled randomly between the coupling blocks. By doing this, the information can be mixed more thoroughly, and thus reduce the final loss. Note that the program must remember how the affine is shuffled from normal order and shuffled order in order to produce $f^{-1}(z)$.

4.3. Glow: Invertible 1×1 convolution:

The invertible 1x1 convolution method is proposed by Glow (Diederik P. Kingma, 9 Jul 2018). Considering the spatial dimension of the images, this is where convolutional operations can achieve good performance. Glow proposed that flow-based generative model with convolutional operations may lead to a better performance and a reduction of the number of parameters to utilize the model parallelization better.

The important part is to make the convolutional operations invertible. The trick is to only shuffle the channels so that is does not destroy the local correlation of the image, and maintain the dimension of images. A more specific algorithm is to first squeeze and divide the original $h \times w \times c$ image into multiple $2 \times 2 \times c$ blocks along the dimension space, then reshape each block to $1 \times 1 \times 4c$ and finally to the $\frac{h}{2} \times \frac{w}{2} \times 4c$ dimension. After this, the data can be subjected to a invertible 1×1 convolution calculation.



Moreover, the permutation operation of vectors can be described by matrix multiplication. For example, the original vector is $[1, 2, 3, 4]$, and then the first, second, third, and fourth numbers are exchanged respectively to obtain $[2, 1, 4, 3]$. This operation can be described by matrix multiplication (permutation matrix):

$$\begin{pmatrix} 3 \\ 2 \\ 1 \\ 4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \tag{9}$$

According to the previous squeeze transformation, we can make W the permutation matrix in $h = xW$ a square matrix, so that its Jacobian matrix $\frac{\partial h}{\partial x} = W$. In order to achieve that, we need to add $-\log|\det W|$ to the loss. Additionaly, we generally use a "random orthogonal matrix" for initialization to ensure reversibility of W.

In order to reduce the computational complexity of the matrix determinant and increase the robustness, Glow (Diederik P. Kingma, 9 Jul 2018) gives a good trick to use LU decomposition and LU inverse decomposition. More specifically, as any matrix can be decomposed into $W = PLU$, where P is a permutation matrix, which is the equivalent matrix of shuffle mentioned earlier; L is a lower triangular matrix, the diagonal elements are all 1; and U is an upper triangular matrix, if we know the expression of this matrix, it is easy to find the Jacobian determinant, which is equal to:

$$\log|\det W| = \sum \log|diag(U)|$$

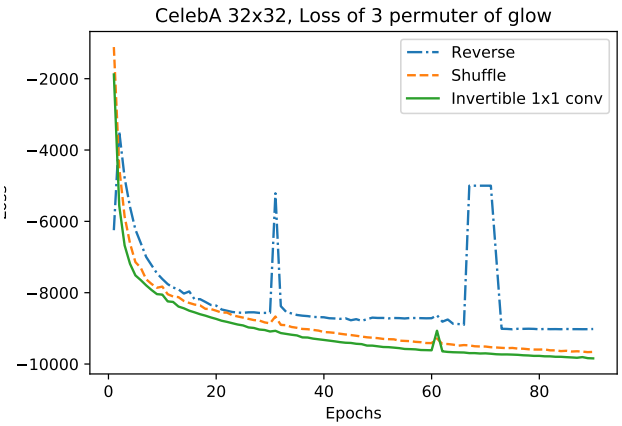
Since any matrix can be decomposed into $W = PLU$, we can directly set W to this form. By doing this, the calculation of the determinant can be greatly reduced, and the calculation is also easier.

To summarize, Glow (Diederik P. Kingma, 9 Jul 2018) propose the following procedure: first randomly generate an orthogonal matrix. Then do LU decomposition, get P, L, U. Fix P and the sign of the diagonal of U, and then constrain L to diagonal. The lower triangular array of all 1 lines, U is the upper triangular array, and optimizes the remaining parameters of L, U.

The invertible 1x1 convolution is composed of the above

trick and the convolutional operation on the image channels. It is essentially a shared, reversible, fully connected layer.

In our implementation, we test and compare these three methods and also compare our results with the results in the glow paper. Our test curve in CelebA is as follows:



Our experiments corroborate the comparative experiments of the Glow paper(Diederik P. Kingma, 9 Jul 2018). From the curve we see that "shuffle" has lower loss than direct "reverse", but the "invertible 1x1 convolution" achieves the lowest loss. In addition, we have some observation which is not mentioned in the Glow paper. For example, "reverse" is the least robust method; the loss of "reverse" stops decreasing the earliest and the loss of "invertible 1x1 convolution" is still decreasing at the end of our 90 epoch training. Moreover, in term of the sampling quality after training, we also observe the following order, reverse < shuffle < invertible 1x1 convolution.

5. Experiment Results

We refered to two versions codes of implementation on GitHub. One version is the Glow repository pushed by OpenAI and run their original code on MNIST dataset, and the other is the version implemented by Keras (Bojone, Aug. 2018). We did a MNIST training test using the version of the OpenAI, and the celebA face test was done with the Keras version.

We are not just using the existing code to run. We are more interested in the version of Keras, so we read the Keras version of the code carefully, and modified it to make the code more in line with the theoretical structure of the paper.

5.1. MNIST with OpenAI Original Code

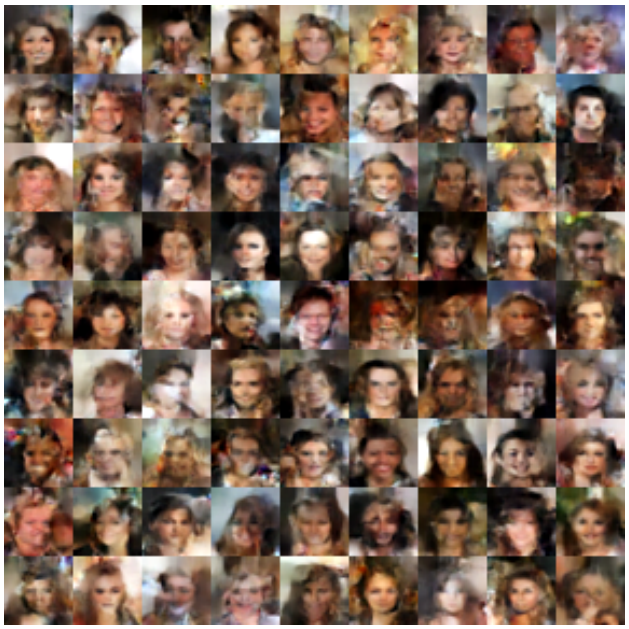
We run the experiment with learning rate=0.001, coupling_method=affine coupling, level=3, depth=32 (the number of convertible transformation functions in the flow). We only trained the model with 10 epochs and it is able to generate decent synthetic images.

5.2. CelebA with 3 permutations

After using the GPU, we used a 32x32 size of CelebA to train and get better quality than the poster. The others parameters: batch size=32, epochs=90, level of multi-scale structure=3, flow depth=10.

5.2.1. Reverse

5.2.2. SHUFFLE



5.2.3. INVERTIBLE 1X1 CONVOLUTION

5.2.4. ORIGINAL

We will not be able to reproduce the training with the full size image and wait until converge due to the limited computational power. This is the best result for us after several days of training with GPU.

6. Conclusion

In our report we show that the flow-based generative model is a capable type of generative model. We explain the probabilistic principle behind the flow-based generative model and show some synthetic images which are generated using flow-based generative model. More specifically, We study the architecture of Glow and compare it with some other variations. The experiment show that Glow performs the best among all the flow-based generative model we use in this report.

Glow produces high quality and less weird images, which is enough to illustrate his advantages over VAE and GAN, but glow also has shortcomings. The most obvious shortcoming is the computational complexity is too expensive, and the amount of calculation exceeds GAN and VAE. This is a point worth improving in future works.

References

Bojone. Implementation of glow using keras on github. Aug. 2018.

Diederik P. Kingma, Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. 9 Jul 2018.

Ioffe, Sergey and Szegedy, Christian. Batch normalization:

Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Laurent Dinh, David Krueger. Nice: Non-linear independent components estimation. 2014.

Laurent Dinh, Jascha Sohl-Dickstein. Density estimation using real nvp. 2016.

Weng, L. Flow-based deep generative models. Dec. 2018.