Exploring Flow-based Generative Models with GLOW

Anish Madan
2016223
anish16223@iiitd.ac.in

Pranav Jain
2016255
pranav16255@iiitd.ac.in

Pulkit Madaan
2016257
pulkit16257@iiitd.ac.in

***Abstract*—Generative modeling is about observing data, like a set of pictures of faces, then learning a model of how this data was generated. Learning to approximate the data-generating process requires learning all structure present in the data, and successful models should be able to synthesize outputs that look similar to the data. Glow is a type of reversible generative model, also called flow-based generative model, and is an extension of the NICE and RealNVP techniques. In this report we try to compare between shallow and deep GLOW models by studying its impact on the generated samples. We also use the GLOW architecture for speech synthesis(namely Waveglow). We extend GLOW by modifying its architecture to make the generation process class-conditional. The report is also accompanied with a demo to demonstrate the performance of this model.**

I. Introduction

Generative modelling or density estimation is a very important task which helps us in estimating the data distribution. Modelling the data distribution includes capturing all dependencies in it and has various applications including generating realistic looking samples never seen before, introduce variations in given samples, inferring latent variables, etc.

Flow based generative models are used to learn the data distribution explicitly, unlike other pevious models like VAEs [1] which approximate the likelihood of data or like GANs [2] which aim to learn the data distribution by playing a minimax game. Flow based models use the concept of normalizing flows which work by transforming a simple probability distribution into a complex one via a series of invertible transformations.

The invertible functions transformations can be as complex as possible, given that their inverse and determinant of the transformation matrix is easily computable. NICE [3] used only scaling transformation, while RealNVP [4] used both translation and scale transformations have simple inverses and easy determinant computations. GLOW [5] on the other hand, along with the scale and translate transformations, also uses 1x1 Convolutions, though these are more complex than the former, their inverse and determinant is still computable and the increase in performance trumps the increase in complexity.

We use the architecture of GLOW to perform unconditional generation of samples with performance better than DCGANs [6]. We then demonstrate the use normalizing flows on a different modality like speech which basically shows the versatility of using such an architecture. Finally, we try to implement class conditional generation of images by modifying the architecture of GLOW. Note that because a flow based architecture supports exact latent variable inference, we demonstrate semantic manipulations of attributes in images and a few other experiments using latent variable inference.

II. Method

In cases where we want to perform inference on probabilistic models, we are given the graphical models. But how do we acquire these models? For complex problems, it requires many domain experts who spend a lot of time constructing the graphical models for these problems and even then, they might not be scalable as the knowledge of the domain for which the graph is being constructed increases. One solution is to automate this process and save the time and resources invested in this task. We assume that the information (or data we have obtained) comes from an underlying distribution which we wish to approximate. One of the goals of density estimation is to construct a model M, such that $\tilde{P}$ (the approximate distribution) is close to the the generating distribution P (the underlying distribution of data). We restrict the possible approximated distributions to a family, where the hidden representations follow a spherical Gaussian distribution.
Here we use normalising flows, composition of bijective maps, to estimate $p(x)$ (where $x \in D$). Relying on the fact that a good representation is one in which the distribution is easy to model (e.g: Gaussian). Using normalising flows leads to exact inference and better synthesis; hence better performance in downstream tasks.

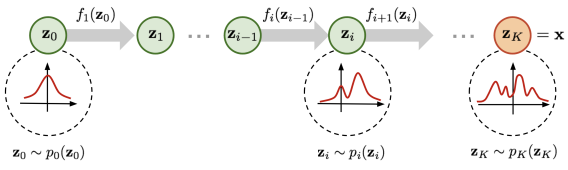


Fig. 1: Illustration of a normalizing flow model, transforming a simple distribution $p_0(z_0)$ to a complex one $p_K(z_K)$ step by step.

A normalizing flow transforms a simple distribution into a complex one by applying a sequence of invertible transformation functions. Flowing through a chain of transformations, we repeatedly substitute the variable for the new one according to the change of variables theorem and eventually obtain a probability distribution of the final target variable. Under the

change of variables theorem, the probability density function of the model given a datapoint (x) can be written as:

$$z_0 = z \quad ; \quad z_K = x$$
$$z \sim p_\theta(z) \quad ; \quad z = g_\theta(x)$$
$$f_\theta(z) = g_\theta^{-1}(z) = x$$
$$p_\theta(z) = N(z; 0; I)$$
$$f = f_1 \circ f_2 \circ \ldots \circ f_K$$

$$\log p_\theta(x) = \log p_\theta(z) + \log |det(dz/dx)| \quad (1)$$
$$\log p_\theta(x) = \log p_\theta(z) + \sum_{i=1}^{K} \log |det(df_i^{-1}/dz_i)| \quad (2)$$

The path traversed by the random variables $z_i = f_i(z_{i-1})$ (refer Fig. 1) is the flow and the full chain formed by the successive distributions $p(z_i)$ is called a normalizing flow. A transformation function f_i should satisfy two properties: It is easily invertible; Its Jacobian determinant is easy to compute. A standard idea is to construct triangular Jacobian matrices as the determinant is simply the product of the diagonal elements of that matrix. We use the following methods in our experiments:

1) Learning a generative(probabilistic graphical) model which helps us learn the probability distribution as it is difficult to construct one manually for such high dimensional data.
2) Latent Variable Inference and manipulation as that helps understand the relation between the latent variables and the generated samples.
3) Unconditional and conditional sampling(or marginal inference) from the generative models with varying temperatures to show a tradeoff between the diversity of model and quality of samples.
4) Using conditional prowess with generative modelling ability to convert between modalities. One example of this is WaveGlow [7], where the generation of raw audio is conditioned on it's mel spectrograms. Enabling one to essentially crate a neural vocoder: finding the raw audio corresponding to a mel spectrogram. One more example of this is solving the problem of image captioning, by conditioning the generation of sentences on their images. Allowing one to generate captions corresponding to the image, the sampling is conditioned on. We only explore the usability of the former, and don't go about the latter as it isn't possible in the limited time period and scope of this course project.

III. Experiments and Results

A. Experimental Setup

We conduct our experiments on the CelebA dataset [8] consisting of more than 200k images each with 40 attribute annotations. We take a 64x64 crop of the images for training. We train 2 GLOW networks with levels, depth and width as (3, 24, 256) and (3,32,512) respectively. We keep the batch size as 32 and name the models as Model1 and Model2 respectively. We use the Adam Optimizer with initial learning rate as 0.001.

Experiments on WaveGlow used the following settings. The backbone text to mel model is trained on the LJSpeech dataset [9] in both Vanilla and Emotional demo. While the Vanilla uses Tacotron2 [10] as the backbone, Emotional demo's backone Tacotron [11] model is fine-tuned on the EmoV-DB dataset [12]. Both use WaveGlow [7] as the mel spectrogram to audio converter, the vocoder.

For class conditional training, we use the MNIST dataset with classes ranging from 0-9. We condition the images on the labels. We use the code from this Github repo[1], and further modify its architecture and implement other experiments to generate the results below. Note that the training was done from scratch and no pretrained model was available.

B. Synthesis of Images

We generate random samples from the model at some specific temperatures from the trained model by sampling from the base distribution(and interpolating) and going in the inverse direction of training of the model(for inference in normalizing flows). We display the results for Model1 and Model2. See figure 2

C. Manipulation of attributes

We now consider modifying the attributes of images in the dataset. We have been provided with labels(binary) for 40 attributes(e.g smiling, black hair, etc) for each image. We compute average latent vector z_{pos} for images containing the attribute and z_{neg} for images not containing it. We then use the difference $z_{pos} - z_{neg}$ as the direction for manipulating that attribute. We display the results obtained for various attributes at temperature=0.7 in figure 3

D. Interpolation between Images

We try to visualize the intermediate space between encoded images by trying to linearly interpolate between them and reconstructing the intermediate encodings. See Figure 4

E. Visualizing the Manifold

To ensure that the latent variables are arranged smoothly and semantically in the latent space, we try to define manifold using 4 images and try to visualize it. Consider z_1, z_2, z_3, z_4 as our 4 encoded examples from the validation set. We parameterize these by 2 parameters θ, ϕ which take the form

$$z = cos(\theta)(cos(\phi)z_1 + sin(\phi)z_2) + \quad (3)$$
$$sin(\theta)(cos(\phi)z_3 + sin(\phi)z_4) \quad (4)$$

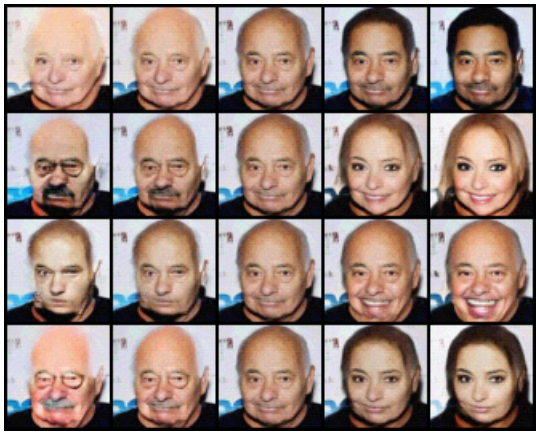
We reconstruct the above z as $\hat{x} = g(z)$, where g is the inverse flow for different values of the parameters. See Figure 5

[1] https://github.com/kamenbliznashki/normalizing_flows

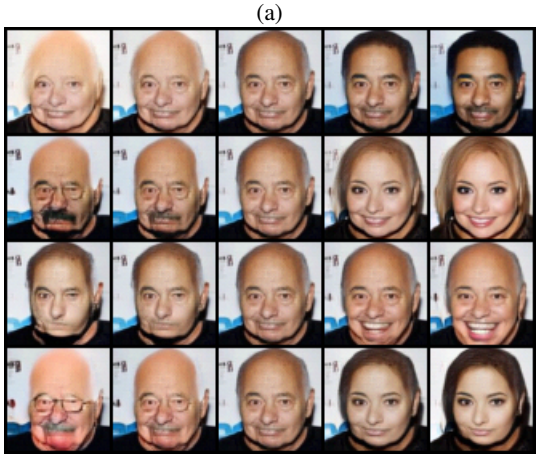
(a) Random samples from Model1, with temperature 0.5

(b) Random samples from Model2, at temperature 0.6

Fig. 2: Random samples generated from the model



(a)



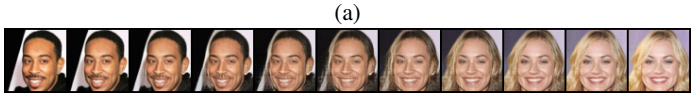
(b)

Fig. 3: We consider the following attributes(top-to-bottom): Black Hair, Heavy makeup, Mouth Slightly Open, Young. The left image is trained using Model1 and the one on the right using Model2. Each row in the image is made by interpolating the latent code of an image along a vector corresponding to the attribute, with the middle image being the original image

F. WaveGlow

We tried to use WaveGlow [7] to create a text-to-speech demo[2]. For the first demo ,Vanilla demo, we used Tacotron 2 [10][3] which produces mel spectrograms for a given text which is then fed to WaveGlow used as the nueral vocoder. For the second demo, Emotional demo, we tried to see if WaveGlow works on out of domain data. We used Tacotron [11] model trained on emotional data[4], developed in our SRU course project. Extracted mel spectrograms from the intermediate layer feeding input to the mel to linear converter module. These mel spectrograms were fed to WaveGlow[5]. Vanilla demo generates highly realistic speech, but the Emotional demo either produces silence or static noise, indicating WaveGlow works best if the mel spectrograms it gets are generated by the same architecture that fed it data during training.

(a)



(b)

Fig. 4: Select 2 images randomly and encode them into $z1$ and $z2$. Now linearly interpolate between them with varying α according to $\hat{z} = \alpha \cdot z1 + (1 - \alpha) \cdot z2$. Now pass $\hat{z}$ to the inverse flow and generate intermediate corresponding output in the data space.

[2]https://colab.research.google.com/drive/16yM9Pze9T$_H$3$QnrR$ – $pyEYwwFPHfjP75Z$

[3]https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/SpeechSynthesis/Tacotron2

[4]https://github.com/Emotional-Text-to-Speech/dl-for-emo-tts

[5]https://github.com/NVIDIA/waveglow

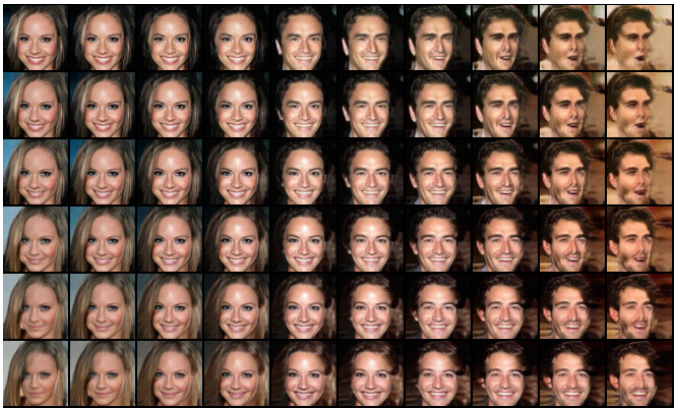


Fig. 5: Visualizing the manifold

G. Class Conditional Generation

Taking inspiration from WaveGlow [7] architecture, we tried to implement a class conditional generation model. We choose MNIST as our dataset and aim to condition the latent space on class labels ranging from $0-9$. We modify the GLOW architecture by introducing our class label, along with image, as a one hot encoded vector in the Affine Coupling Layer of the last level of GLOW [5] network. We add this class label, after passing it through a small neural network and reshaping it, to our image information and then apply the convolution operations. This ensures that our model encodes class label as well as image information. We add this class label to each Affine Couling Layer in the last level and not just in one, as the former seems to work better. Since, flow models are invertible, the class information is provided to the model at generation as well as inference. The class conditional generation is computationally very heavy to give meaningful results, so we could train our model over 70 epochs only. We started to see some results but not of very high quality. This can simply be achieved with more resources and compute time.

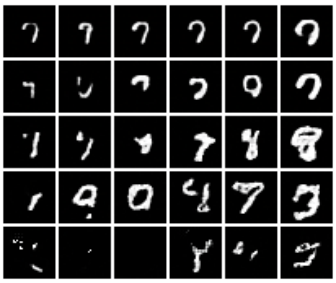


Fig. 6: Generated random sample with labels 7,9,0,4,7,8. As we can see the generation quality is not too good due to lesser amount of training, but nevertheless we get decent results and can make out the characters in the 4th row which gives more diversity as compared to above rows due to higher sampling temperature. We can see that 7 is misrepresented although it is pretty close to one in shape. Similarly 8 is generated in the 3rd row but 3 in the 4th row. This is also understandable as 3 and 8 are quite close due to structural similarities.

IV. Observations

We observe that some of the samples generated by our model in Figure 2 are quite realistic looking. These results can be improved by training for longer and using a high quality dataset like CelebA-HQ, but it would require a lot of compute power. We contrast the results we get from Model1 and Model2. Though it is harder to distinguish Model2 gives a better quality(diversity) of samples in the figure.

In Figure 3, we observe quite a smooth interpolation between the varying attributes which indicates the image manifold of the generating distribution is smooth and most intermediate samples look realistic enough. We again try to compare the results on a shallow vs deeper model (i.e Model1 vs Model2). On observing carefully we see that the interpolations are smoother and more realistic for Model2, especially at the extremities of the attribute in consideration.

In Figure 4, we linearly interpolate between two images and observe that the intermediate results are also of good quality and resemble a face as well. There is minimum blurring and the smoothness of transition reflects the strengths of normalizing flows and the quality of training.

We observe in Figure 5 that our model has organized its latent space with respect to some semantics which are possibly based on the attributes provided during training and not simply by interpolating in the pixel space. This leads us to believe that conditioning the model on some kind of supervision would help in organizing the latent space on the basis of this external signal. We follow this hypothesis up by training a class conditional model on MNIST datasets (i.e by conditioning on class labels)

On training the class conditional model, we see that while the generation quality is not very high, we still manage to get coherent and class specific samples. The inference and observations are reported in the description of Figure 6

V. Individual Contribution

The work was done by each of us contributing equally in getting the results given in this document. No specific sub-tasks were divided amongst us.

References

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[3] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation," *arXiv preprint arXiv:1410.8516*, 2014.

[4] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[5] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in Neural Information Processing Systems*, pp. 10215–10224, 2018.

[6] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015.

[7] R. Prenger, R. Valle, and B. Catanzaro, "Waveglow: A flow-based generative network for speech synthesis," *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3617–3621, 2019.

[8] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[9] K. Ito, "The lj speech dataset." https://keithito.com/LJ-Speech-Dataset/, 2017.

[10] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. J. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions," *CoRR*, vol. abs/1712.05884, 2017.

[11] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. V. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, "Tacotron: A fully end-to-end text-to-speech synthesis model," *CoRR*, vol. abs/1703.10135, 2017.

[12] A. Adigwe, N. Tits, K. E. Haddad, S. Ostadabbas, and T. Dutoit, "The emotional voices database: Towards controlling the emotion dimension in voice generation systems," *arXiv preprint arXiv:1806.09514*, 2018.