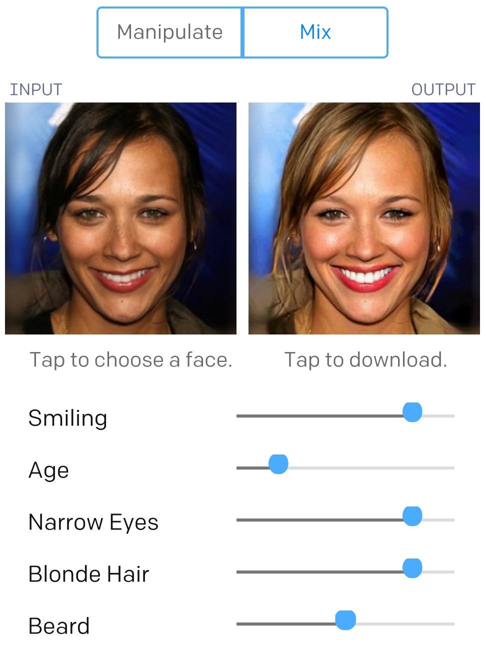
Glow: Better Reversible Generative Models

JULY 9, 2018

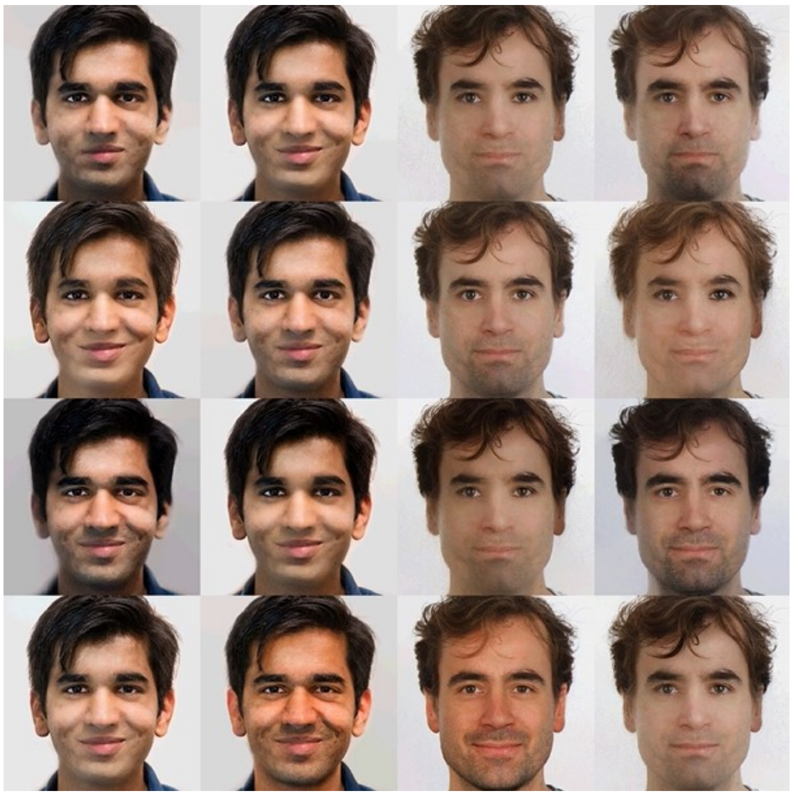
We introduce *Glow*, a reversible generative model which uses invertible 1x1 convolutions. It extends previous work on reversible generative models and simplifies the architecture. Our model can generate realistic high resolution images, supports efficient sampling, and discovers features that can be used to manipulate attributes of data. We're releasing code for the model and an online visualization tool so people can explore and build on these results.

READ PAPER



An interactive demo of our model to manipulate attributes of your face, and blend with other faces

Motivation



Manipulating attributes of images of researchers Prafulla Dhariwal and Durk Kingma. The model isn't given attribute labels at training time, yet it learns a latent space where certain directions correspond to changes in attributes like beard density, age, hair color, and so on.

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. Accurate generative models have broad applications, including speech synthesis, text analysis and synthesis, semi-supervised learning and model-based control. The technique we propose can be applied to those problems as well.

Glow is a type of reversible generative model, also called *flow-based generative model*, and is an extension of the NICE and RealNVP techniques. Flow-based generative models have so far gained little attention in the research community compared to GANs and VAEs.

Some of the merits of flow-based generative models include:

- Exact latent-variable inference and log-likelihood evaluation. In VAEs, one is able to infer only approximately the value of the latent variables that correspond to a datapoint. GAN's have no encoder at all to infer the latents. In reversible generative models, this can be done exactly without approximation. Not only does this lead to accurate inference, it also enables optimization of the exact log-likelihood of the data, instead of a lower bound of it.

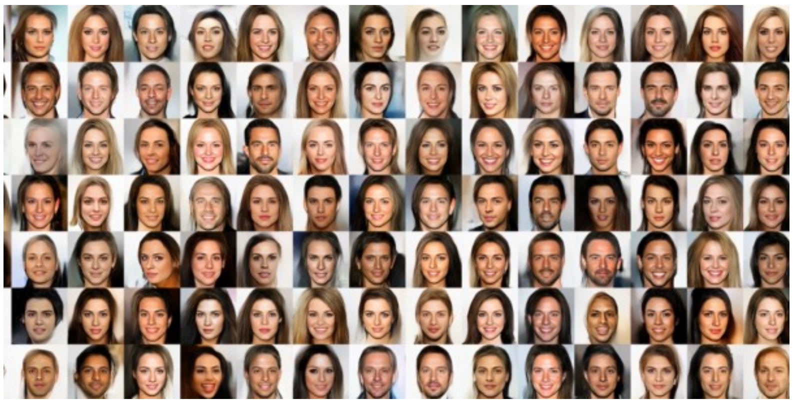
- Efficient inference and efficient synthesis. Autoregressive models, such the PixelCNN, are also reversible, however synthesis from such models is difficult to parallelize, and typically inefficient on parallel hardware. Flow-based generative models like Glow (and RealNVP) are efficient to parallelize for both inference and synthesis.
- Useful latent space for downstream tasks. The hidden layers of autoregressive models have unknown marginal distributions, making it much more difficult to perform valid manipulation of data. In GANs, datapoints can usually not be directly represented in a latent space, as they have no encoder and might not have full support over the data distribution. This is not the case for reversible generative models and VAEs, which allow for various applications such as interpolations between datapoints and meaningful modifications of existing datapoints.
- Significant potential for memory savings. Computing gradients in reversible neural networks requires an amount of memory that is constant instead of linear in their depth, as explained in the RevNet paper.

Results

Using our techniques we achieve significant improvements on standard benchmarks compared to RealNVP, the previous best published result with flow-based generative models.

Dataset	RealNVP	Glow
CIFAR-10	3.49	**3.35**
Imagenet 32x32	4.28	**4.09**
Imagenet 64x64	3.98	**3.81**
LSUN (bedroom)	2.72	**2.38**
LSUN (tower)	2.81	**2.46**
LSUN (church outdoor)	3.08	**2.67**

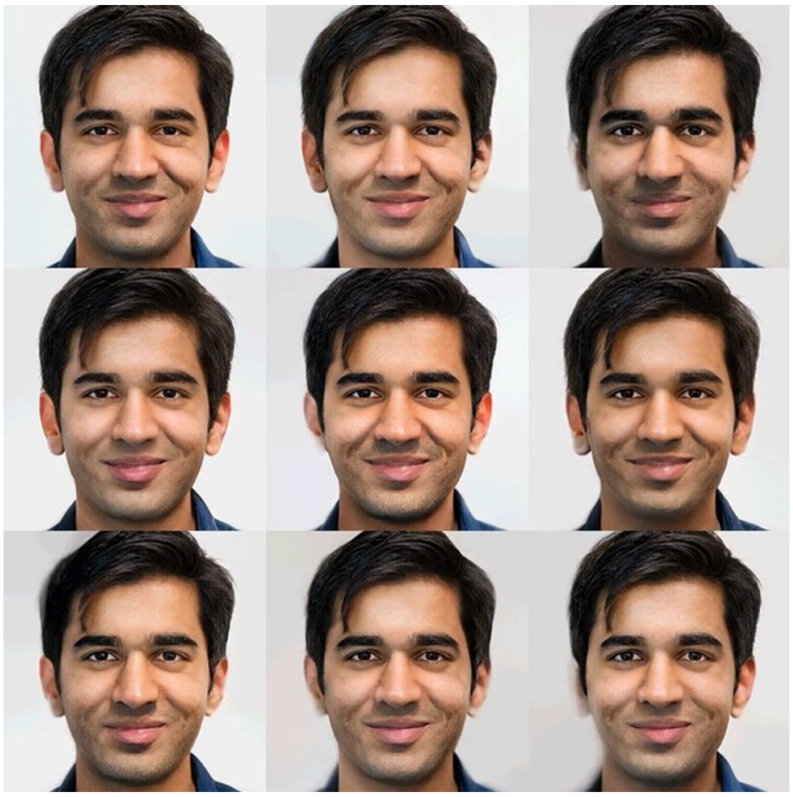
Quantitative performance in terms of bits per dimension evaluated on the test set of various datasets, for the RealNVP model versus our Glow model.


Samples from our model after training on a dataset of 30,000 high resolution faces

Glow models can generate realistic-looking high-resolution images, and can do so efficiently. Our model takes about 130ms to generate a 256 x 256 sample on a NVIDIA 1080 Ti GPU. Like previous work, we found that sampling from a reduced-temperature model often results in higher-quality samples. The samples above were obtained by scaling the standard deviation of the latents by a temperature of 0.7.

Interpolation in latent space

We can also interpolate between arbitrary faces, by using the encoder to encode the two images and sample from intermediate points. Note that the inputs are arbitrary faces and not samples from the model, thus providing evidence that the model has support over the full target distribution.


Interpolating between Prafulla's face and celebrity faces.

Manipulation in latent space

We can train a flow-based model, without labels, and then use the learned latent reprentation for downstream tasks like manipulating attributes of your input. These semantic attributes could be the color of hair in a face, the style of an image, the pitch of a musical sound, or the emotion of a text sentence. Since flow-based models have a perfect encoder, you can encode inputs and compute the average latent vector of inputs with and without the attribute. The vector direction between the two can then be used to manipulate an arbitrary input towards that attribute.

The above process requires a relatively small amount of labeled data, and can be done after the model has been trained (no labels are needed while training). Previous work using GAN's requires training an encoder separately. Approaches using VAE's only guarantee that the decoder and encoder are compatible for in-distribution data. Other approaches involve directly learning the function representing the transformation, like Cycle-GAN's, however they require retraining for every transformation.

```
# Train flow model on large, unlabelled dataset X
m = train(X_unlabelled)

# Split labelled dataset based on attribute, say blonde hair
X_positive, X_negative = split(X_labelled)

# Obtain average encodings of positive and negative inputs
z_positive = average([m.encode(x) for x in X_positive])
z_negative = average([m.encode(x) for x in X_negative])

# Get manipulation vector by taking difference
z_manipulate = z_positive - z_negative

# Manipulate new x_input along z_manipulate, by a scalar alpha \in [-1,1]
z_input = m.encode(x_input)
x_manipulated = m.decode(z_input + alpha * z_manipulate)
```

Simple code snippet for using a flow-based model for manipulating attributes

Contribution

Our main contribution and also our departure from the earlier RealNVP work is the addition of a reversible 1x1 convolution, as well as removing other components, simplifying the architecture overall.

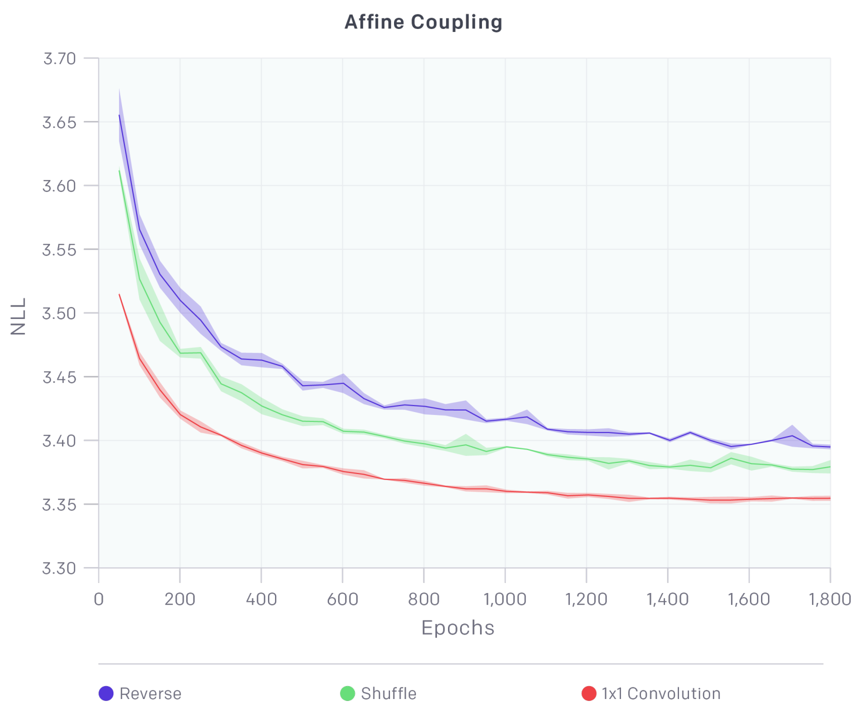
The RealNVP architecture consists of sequences of two types of layers: layers with checkboard masking, and layers with channel-wise masking. We remove the layers with checkerboard masking, simplifying the architecture. The layers with channel-wise masking perform the equivalent of a repetition of the following steps:

1. Permute the inputs by reversing their ordering across the channel dimension.
2. Split the input into two parts, A and B, down the middle of the feature dimension.

3. Feed A into a shallow convolutional neural network. Linearly transform B according to the output of the neural network.
4. Concatenate A and B.

By chaining these layers, A updates B, then B updates A, then A updates B, etc. This bipartite flow of information is clearly quite rigid. We found that model performance improves by changing the reverse permutation of step (1) to a (fixed) *shuffling* permutation.

Taking this a step further, we can also *learn* the optimal permutation. Learning a permutation matrix is a discrete optimization that is not amendable to gradient ascent. But because the permutation operation is just a special case of a linear transformation with a square matrix, we can make this work with convolutional neural networks, as permuting the channels is equivalent to a 1x1 convolution operation with an equal number of input and output channels. So we replace the fixed permutation with learned 1x1 convolution operations. The weights of the 1x1 convolution are initialized as a random rotation matrix. As we show in the figure below, this operation leads to significant modeling improvements. We've also shown that the computations involved in optimizing the objective function can be done efficiently through a LU decomposition of the weights.



Our main contribution, invertible 1x1 convolutions, leads to significant modeling improvements.

In addition, we remove batch normalization and replace it with an activation normalization layer. This layer simply shifts and scales the activations, with data-dependent initialization that normalizes the activations given an initial minibatch of data. This allows scaling down the minibatch size to 1 (for large images) and scaling up the size of the model.

Scale

Our architecture combined with various optimizations, such as gradient checkpointing, allows us to train flow-based generative models on a larger scale than usual. We used Horovod to easily train our model on a cluster of multiple machines; the model used in our demo was trained on 5 machines with each 8 GPUs. Using this setup we train models with over a hundred million parameters.

Directions for research

Our work suggests that it's possible to train flow-based models to generate realistic high-resolution images, and learned latent representations that can be easily used for downstream tasks like manipulation of data. We suggest a few directions for future work:

1. **Be competitive with other model classes on likelihood.** Autoregressive models and VAE's perform better than flow-based models on log-likelihood, however they have the drawbacks of inefficient sampling and inexact inference respectively. One can combine flow-based models, VAEs and autoregresive models to trade off their strengths; this would be an interesting direction for future work.
2. **Improve architecture to be more compute and parameter efficient.** To generate realistic high-resolution images, the face generation model uses ~200M parameters and ~600 convolution layers, which makes it expensive to train. Models with smaller depth performed worse on learning long-range dependencies. Using self attention architectures, or performing progressive training to scale to high resolutions could make it computationally cheaper to train glow models.

Finally, if you'd like use Glow in your research, we encourage you to check out our paper for more details, or look at our code on this Github repo.

Special thanks to Nicholas Benson for helping us build the demo.

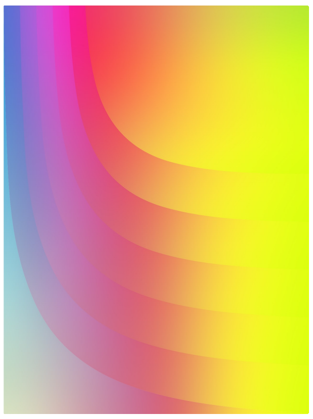
By PRAFULLA DHARIWAL & DURK KINGMA.

TWEET SHARE

Keep reading

MORE

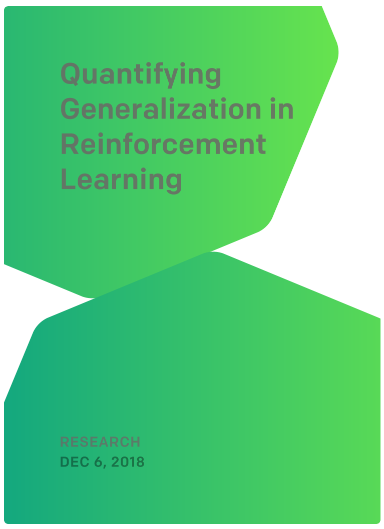
How AI
Training Scales



NO. 16
DEC 14
2018

OpenAI Fellows
Summer Class of
’18: Final Projects

UPDATES
DEC 19, 2018



Quantifying
Generalization in
Reinforcement
Learning

RESEARCH
DEC 6, 2018