Glow: Generative Flow with Invertible 1×1 Convolutions

(but mainly just an introduction to normalizing flows)

Jakub Arnold

2019

Generative Models

- Suppose we want to model $p(x)$, which has a complicated and unknown form (an image of a digit).
- We simplify the problem by introducing a latent variable z which explains the context of x, that is $p(x|z)$ (what digit is in the image).
- We can still reconstruct $p(x)$ by marginalizing out z while picking a prior $p(z)$

$$p(x) = \int p(x|z)p(z)\ dz.$$

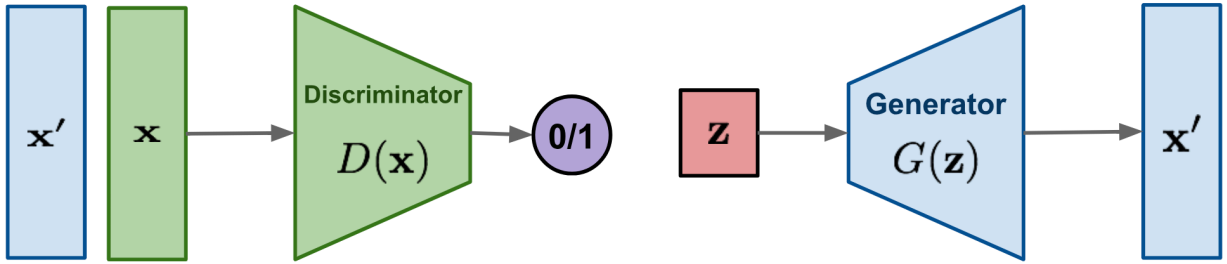
- With this in mind, we might want to ask what the latent variable z is, given that we have an observation x, that is the posterior distribution $p(z|x)$.
- But unfortunately, this is intractable to compute directly, because it requires us to compute $p(x)$.

Generative Models

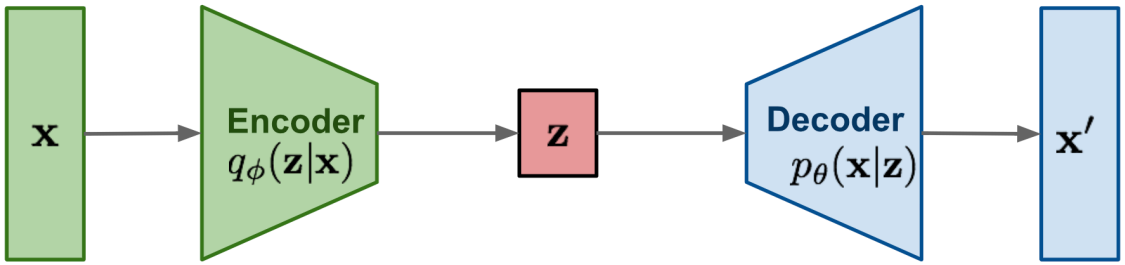
- So far we've only encountered two types of generative models. GANs and VAEs.
- Neither of these learn the real data distribution $p(x)$, because $p(x) = \int p(x|z)p(z)\ dz$ is intractable to compute directly.
- **GAN** plays minimax and learns the distribution of data implicitly as a side effect.
- **VAE** indirectly optimizes the log-likelihood by maximizing the ELBO. Only learns an approximation of $p(z|x)$.
- **Flow-based generative models** are constructed from a sequence of invertible continuous transformations (with easy Jacobian determinant – more on this later), and allows for exact latent variable inference.

Three Types of Generative Models

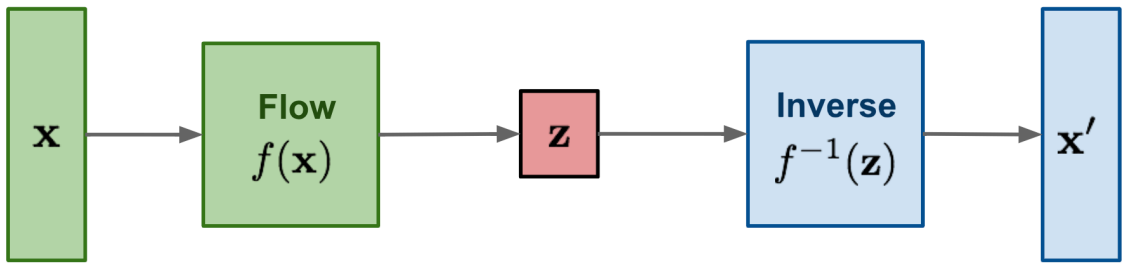
GAN: minimax the classification error loss.



VAE: maximize ELBO.



Flow-based generative models: minimize the negative log-likelihood



Change of Variable Formula

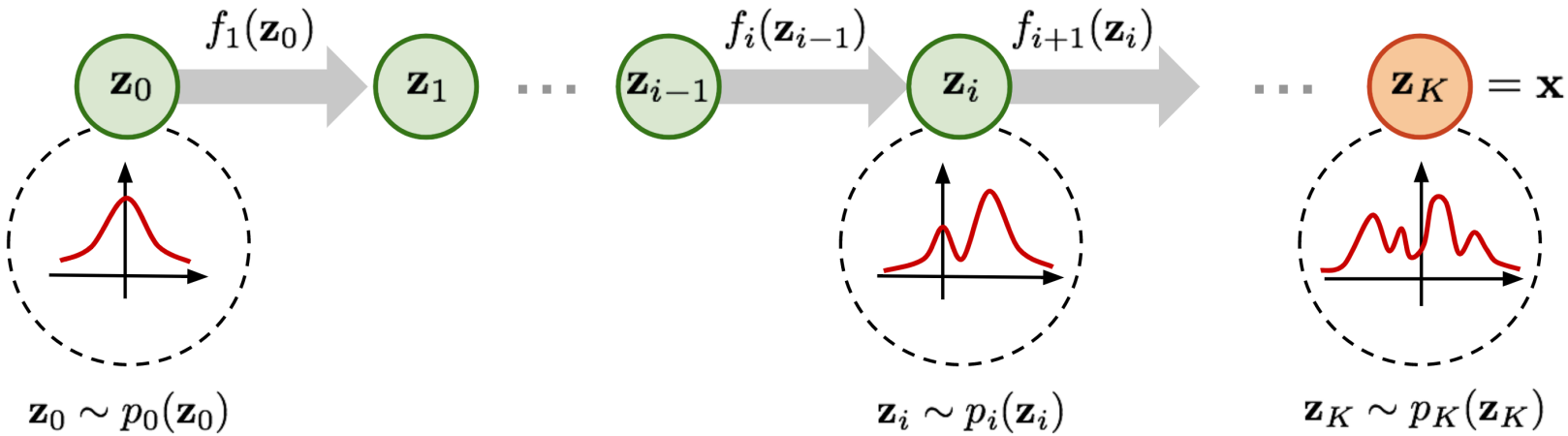
Given a random variable z with a known density $\pi(z)$, and an invertible function $x = f(z)$ (meaning we can write $z = f^{-1}(x)$). **The question is, what is $p(x)$?**

$$p(\mathbf{x}) = p(f(\mathbf{z})) \tag{1}$$
$$= \pi(f^{-1}(\mathbf{x})) \cdot |\det J(f^{-1})| \tag{2}$$

Proof (not really 😱): Think of $\int \pi(z)\, dz$ as a sum of tiny rectangles of width Δz. Their height is the density $\pi(z)$. Substituting $z = f^{-1}(x)$ yields $\frac{\Delta z}{\Delta x} = (f^{-1}(x))'$ and $\Delta z = (f^{-1}(x))' \Delta x$, where $(f^{-1}(x))'$ indicates the volume change between the two variables. **This extends to the multivariate version with the Jacobian determinant of f.**

Normalizing Flows

Start with a simple distribution, then apply a sequence of invertible transformations, applying the change of variables theorem.
Because we can use the change of variables theorem in each step, multiplying by $|\det J(f_i)|$, we can recover the exact density at the end.



The full chain is called a **normalizling flow**.

Normalizing Flows - Training

Because the exact log-likelihood $p(x)$ is tractable, we can use it to train the flow-based generative model by simply minimizing the negative log-likelihood on the training data $\mathcal{D}$:

$$\mathcal{L}(\mathcal{D}) = -\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log p(x).$$

This works because we can compute $p(x)$ using the change of variables and because we can choose the prior $\pi(z)$ to be a simple tractable density, such as a Gaussian.

$$p(x) = \pi(f^{-1}(x)) |\det J(f^{-1})| \tag{3}$$

$$\log p(x) = \log \pi(f^{-1}(x)) - \sum \log |\det J(f^{-1})| \tag{4}$$

Normalizing Flows

To actually compute this, we need two things:

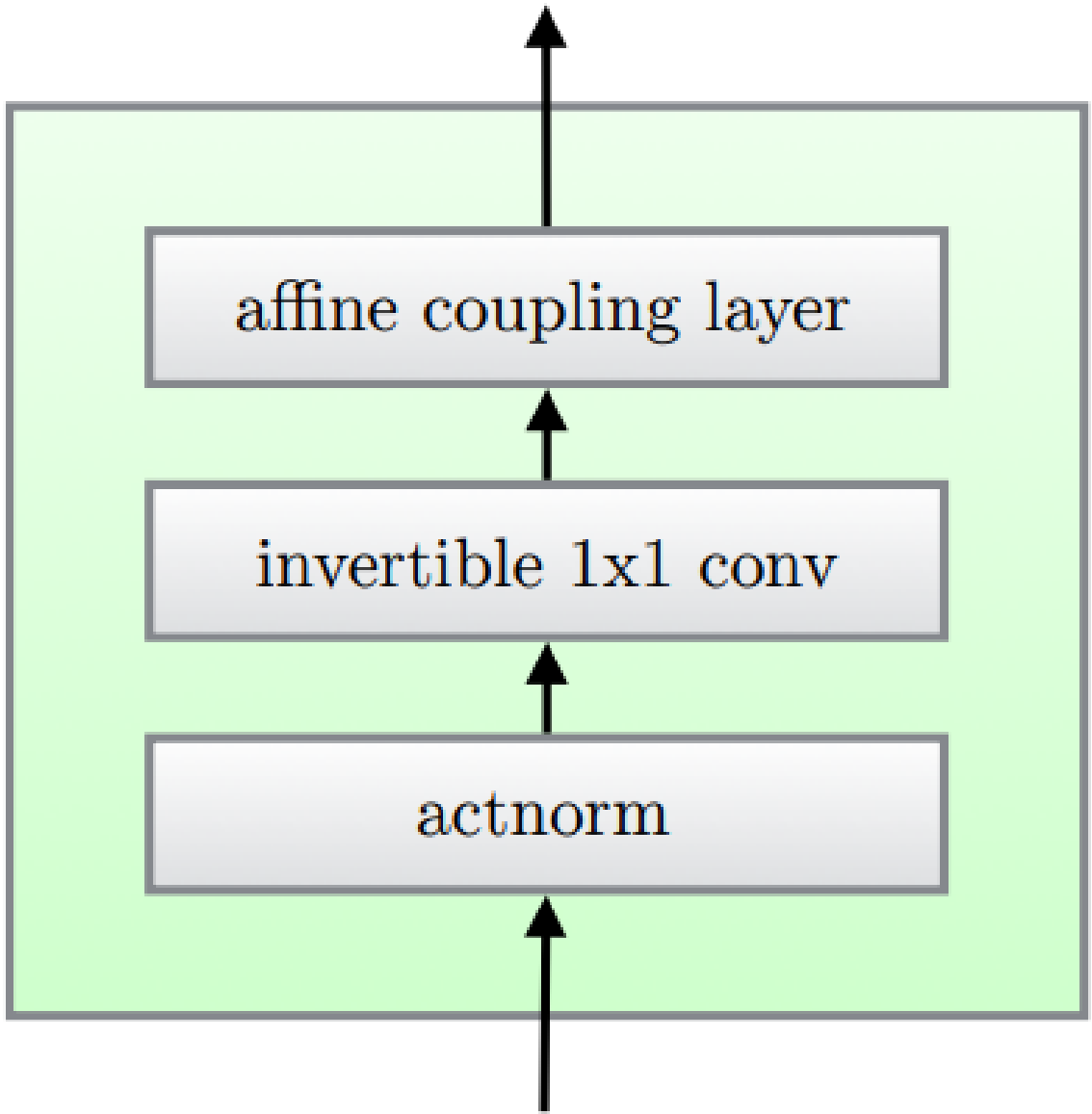
- Easily computible inverse of f.
- Diagonal/triangular Jacobian of f, because $det(A)$ is $O(n^3)$.

If we choose f such that the Jacobian is a diagonal or a triangular matrix L, we simply get $\det(L) = prod(diag(L))$. In practice for numerical reasons we'll deal with log-likelihood and a log determinant. This is even nicer, because $\log(\det(L)) = sum(\log(diag(L)))$.

Why do we care about any of this?

- We can encode an image to z and get a perfect reconstruction.
- In comparison, GANs don't have an encoder at all to infer $p(z|x)$. Unless doing something weird like VAE-GAN, which still does not optimize the likelihood $p(x)$.
- Even with a VAE, we only get an approximation of $p(z|x)$ because we're optimizing the lower bound on the log-likelihood. As a result, $decoder(encoder(x)) \neq x$.

Glow Architecture



Glow Layers

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$

Actnorm

The authors of RealNVP (previous paper, Dinh et al. 2016) suggest the use of batch normalization. But because Glow is too large and can only be trained with batch size of 1 to fit in memory, the authors of Glow propose a *actnorm* layer (activation normalization).

$$\mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$$

- Performs an affine transformation using a trainable scale and bias per channel.
- Initialized such that activations post-actnorm have zero mean and unit variance.
- After initialization, scale and bias are trained as regular trainable parameters.

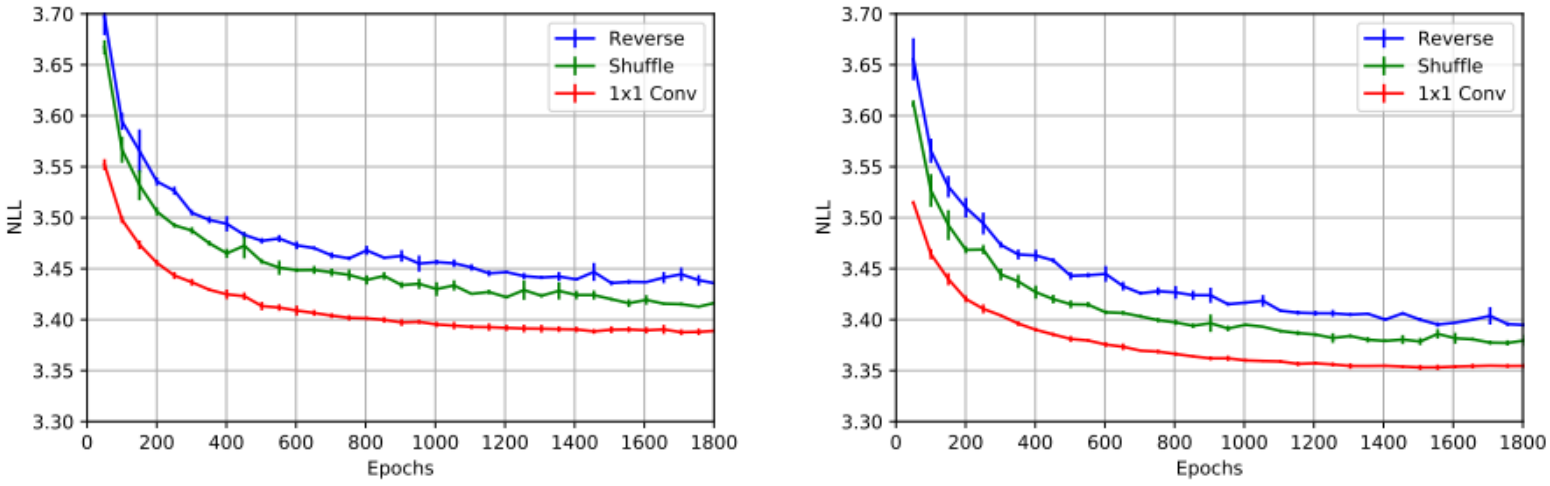
Invertible 1×1 convolution

(Dinh et al., 2014, 2016) used a fixed permutation and reversed the order of channels. Glow uses an invertible 1×1 convolution instead. This is done by having the same number of filters as there are input channels.

$$\mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$$

The cost of computing $det(\mathbf{W})$ is $O(c^3)$, which is comparable to the cost of $conv2D(\mathbf{h}; \mathbf{W})$, which is $O(h \cdot w \cdot c^2)$. They introduce a way of reducing this to $O(c)$ by using LU decomposition on $\mathbf{W}$, but in the experiments they don't measure a significant difference in computation time.

Invertible 1×1 convolution



(a) Additive coupling.

(b) Affine coupling.

Figure 3: Comparison of the three variants - a reversing operation as described in the RealNVP, a fixed random permutation, and our proposed invertible 1×1 convolution, with additive (left) versus affine (right) coupling layers. We plot the mean and standard deviation across three runs with different random seeds.

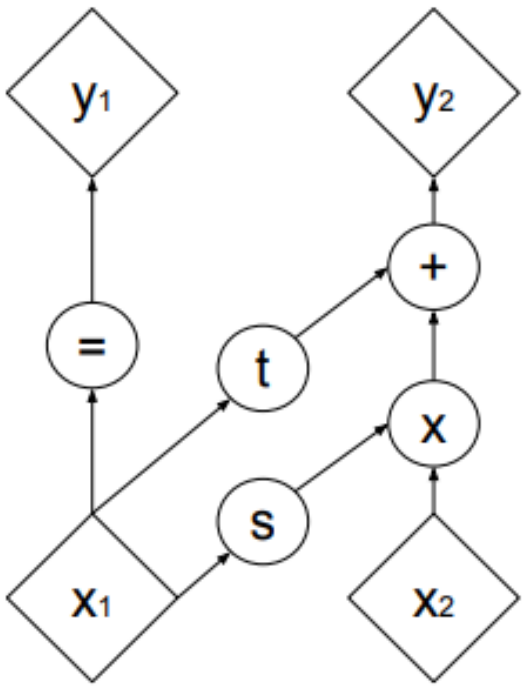
Affine coupling layer

Since both of the previous operations are affine transformation, the affine coupling layer is introduced to add a non-linear (but invertible) transformation. Splitting x into two parts along the channel dimension (in the code they split it in half) as $x = (x_1\ x_2)^T$ we get

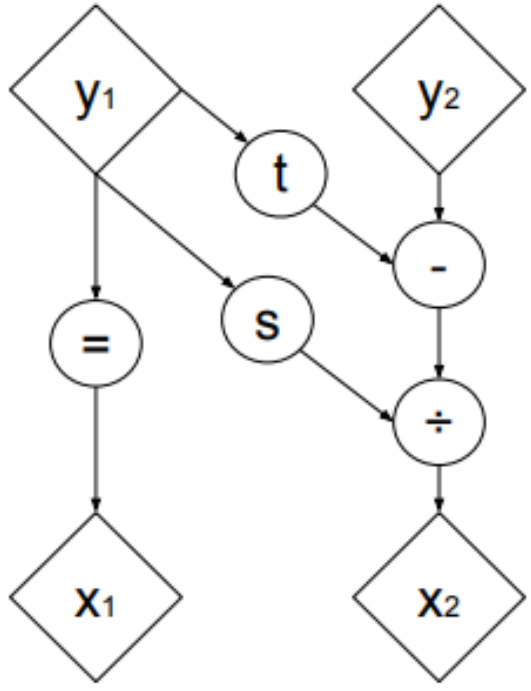
$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \\ s(x_1)x_2 + b(x_1) \end{pmatrix}.$$

where s is an invertible network. Since an activation function is an element-wise transformation its Jacobian is a diagonal matrix, so we just have to make sure it is invertible (LeakyReLU instead of ReLU, etc.).

Affine coupling layer



(a) Forward propagation



(b) Inverse propagation

Affine coupling layer — Jacobian

Now we just make sure we can compute the Jacobian of the affine coupling layer

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \\ x_2 s(x_1) + b(x_1) \end{pmatrix}.$$

It turns out, almost as if by magic, that this is easy!

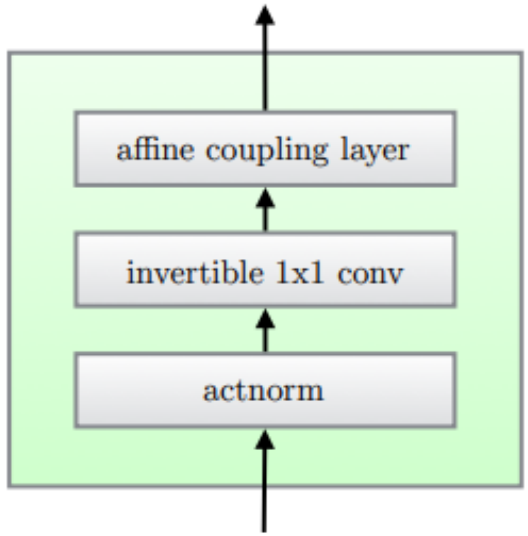
$$J = \begin{pmatrix} \mathbf{I} & 0 \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & 0 \\ \frac{\partial y_2}{\partial x_1} & diag(s(x_1)) \end{pmatrix}$$

We don't actually have to compute $\frac{\partial y_2}{\partial x_1}$ to get $\det J$. Note that the $\frac{\partial y_2}{\partial x_2}$ is just $diag(s(x_1))$, because the Jacobian of an affine transform is a diagonal matrix.

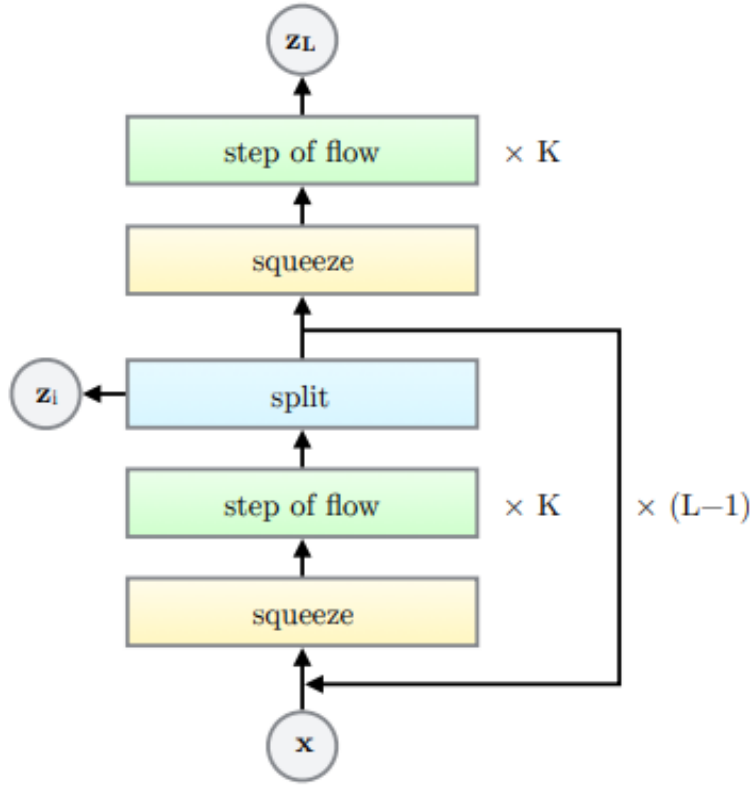
▸ Affine Coupling Layer source code

Glow Architecture

Each block is repeated K times in a bigger block, which is repeated $L-1$ times, wich another K repeats. Set $K = 32, L = 6$.



(a) One step of our flow.

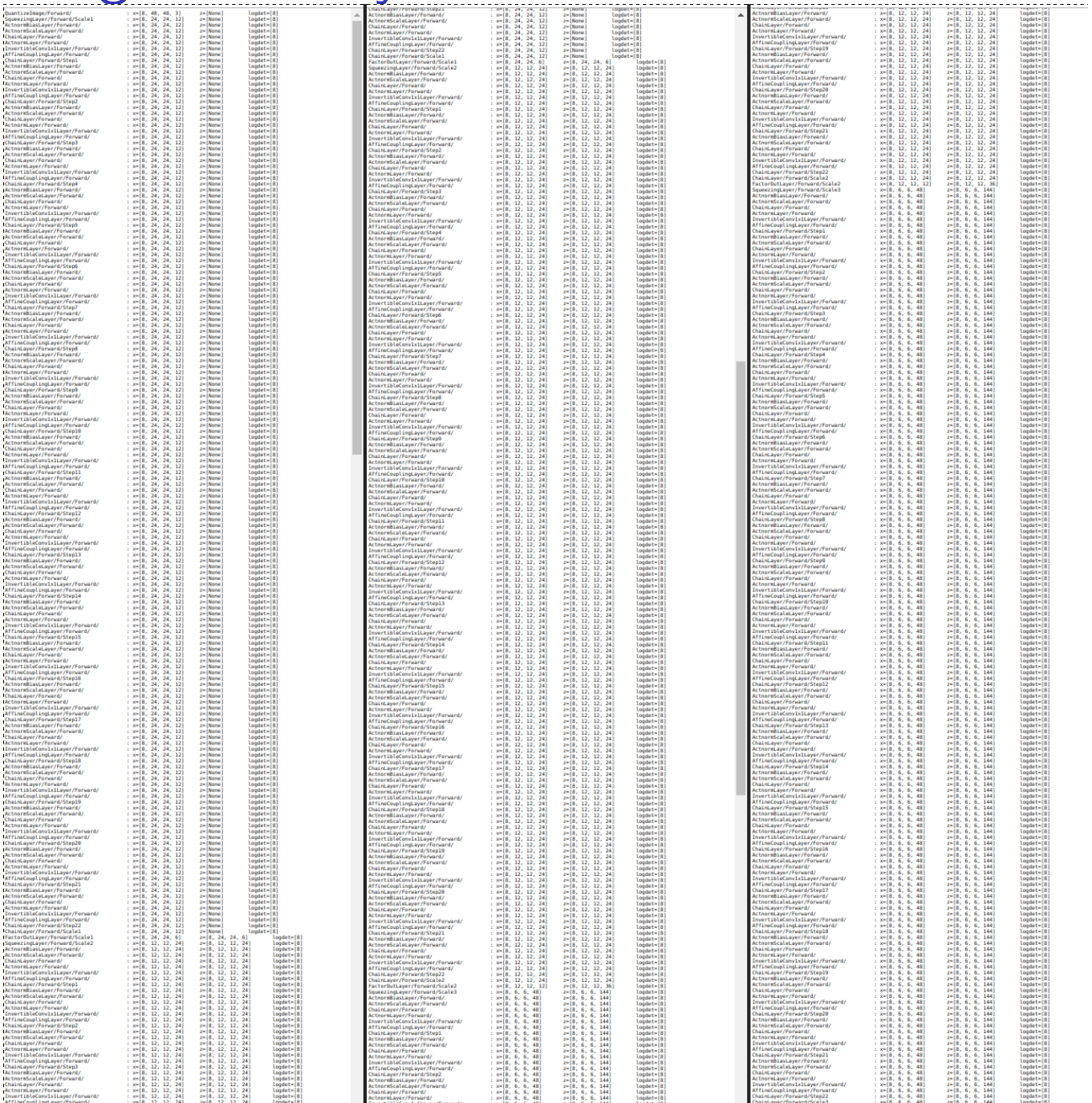


(b) Multi-scale architecture (Dinh et al., 2016).

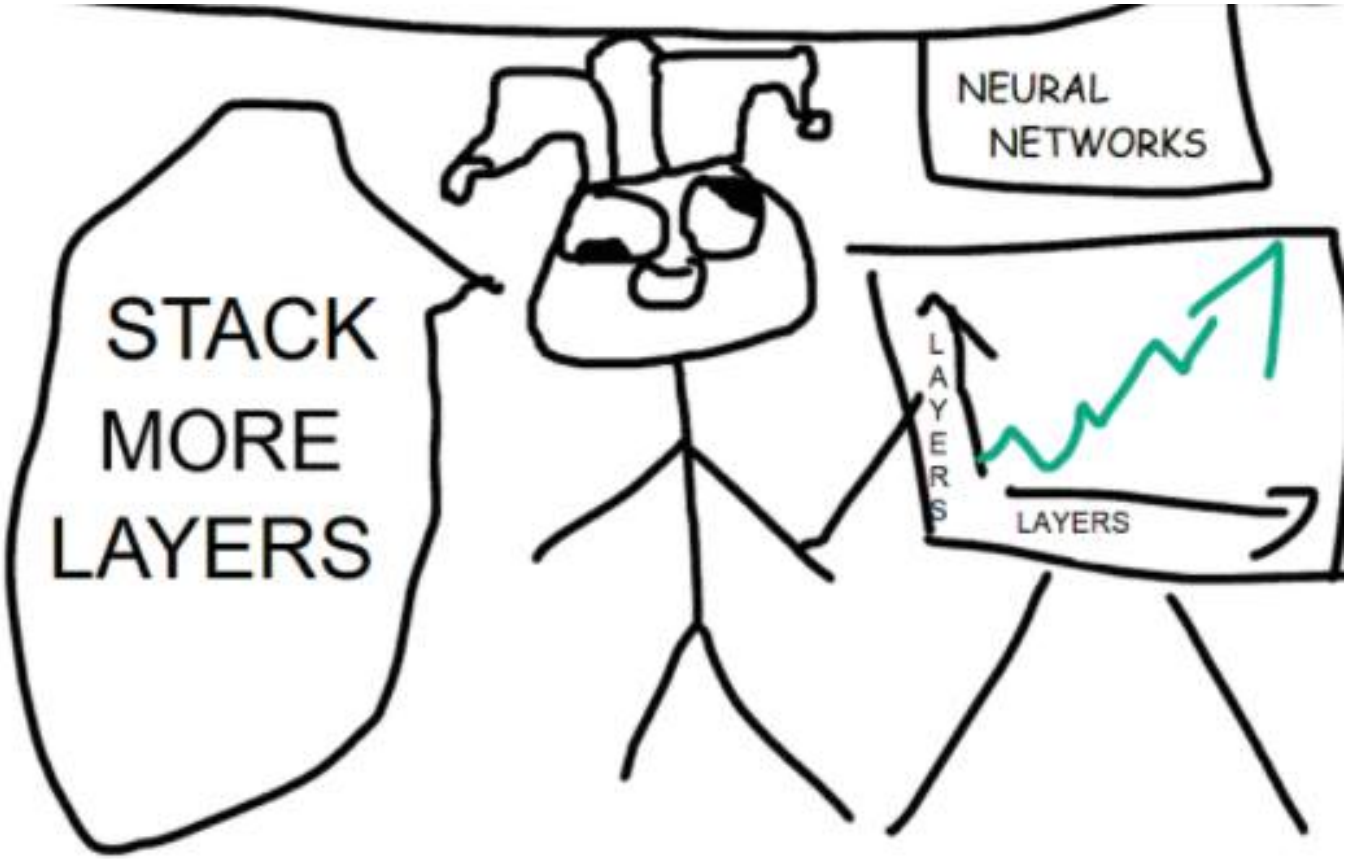
How big is it really?

- Trained on 5 machines, each with 8 GPUs.
- **But the model is big.**
- They only managed to fit a minibatch of size 1 with gradient checkpointing (makes up to 10x bigger models fit into memory at the cost of increased computation).
- Used 5-bit images.
- One 256×256 sample on a 1080ti takes 130ms to generate.

How big is it really?



How big is it really?



Glow Samples trained on CelebA HQ

Glow Interpolations in z

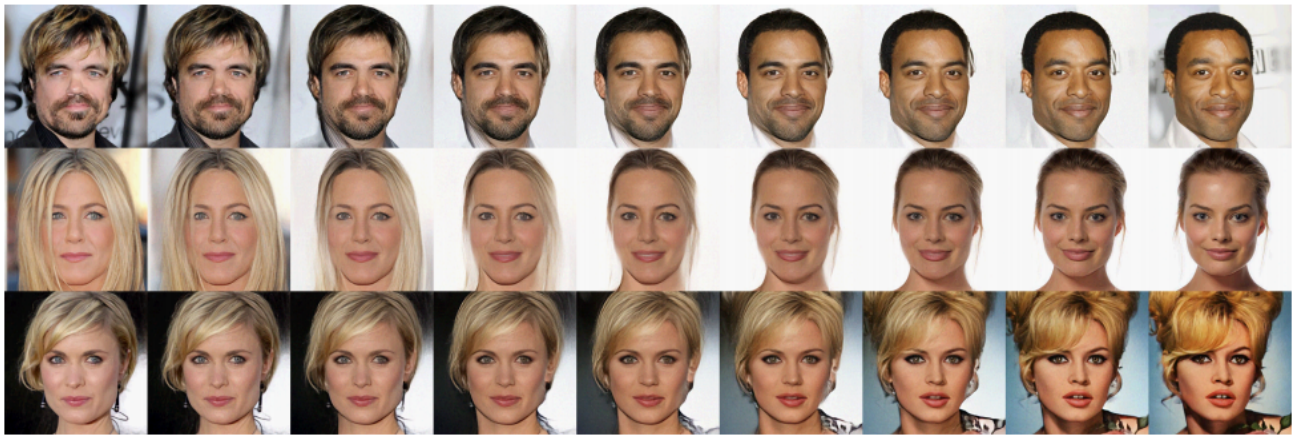


Figure 5: Linear interpolation in latent space between real images

(a) Smiling

(b) Pale Skin

(c) Blond Hair

(d) Narrow Eyes

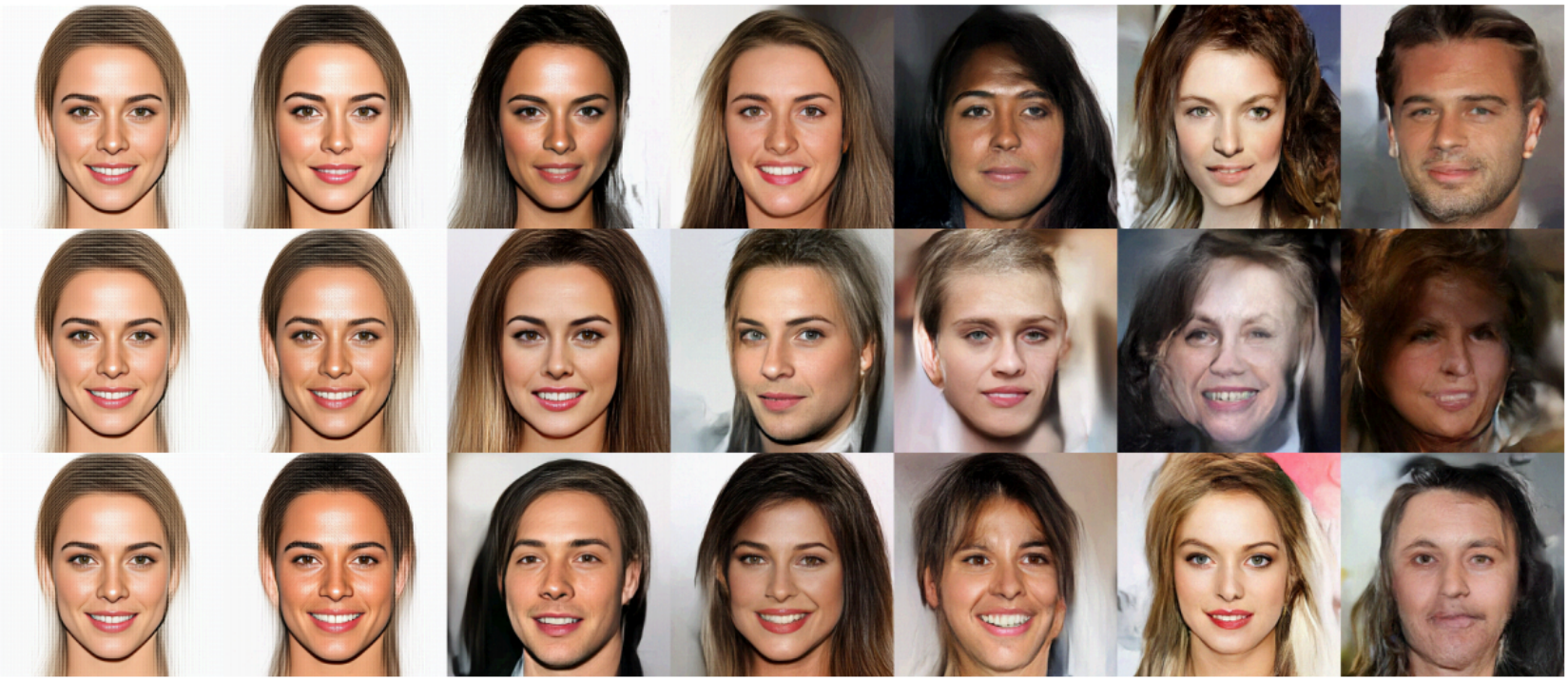
(e) Young

(f) Male

Demo

Effect of change in temeperature

Samples obtained at $0, 0.25, 0.6, 0.7, 0.8, 0.9, 1.0$ temperature.
Ended up choosing 0.7.



Effect of depth

Samples from shallow model on the left ($L = 4$) and deep model on the right $L = 6$.

Progress in normalizling flows

Even though the model is huge and seems impractical, the progress in normalizing flows is quite visible.

NICE → RealNVP → Glow

(b) Model trained on TFD

Figure 8: Samples from a model trained on *CelebA*.

Conclusion

Our model is, to the best of our knowledge, the first likelihood-based model in the literature that can efficiently synthesize high-resolution natural images.

authors of the paper

References

- Density estimation using Real NVP, Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio.
- NICE: Non-linear Independent Components Estimation Laurent Dinh, David Krueger, and Yoshua Bengio.
- Glow: Generative Flow with Invertible 1x1 Convolutions Diederik P. Kingma, and Prafulla Dhariwal.
- Variational Inference with Normalizing Flows Danilo Jimenez Rezende, Shakir Mohamed.
- Flow-based Deep Generative Models, Lilian Weng
- Normalizing Flows Tutorial, Part 1: Distributions and Determinants, Eric Jang.
- Normalizing Flows Tutorial, Part 2: Modern Normalizing Flows, Eric Jang.
- Normalizing Flow, Adam Kosiorek.
- Introduction to Normalizing Flows, Krzysztof Kolasinski
- Normalizing Flows: From NICE, RealNVP to GLOW, Krzysztof Kolasinski