Imperial College London

Normalising flows

Deep Learning Course

Kevin Webster, Pierre Harvey Richemond

Course outline

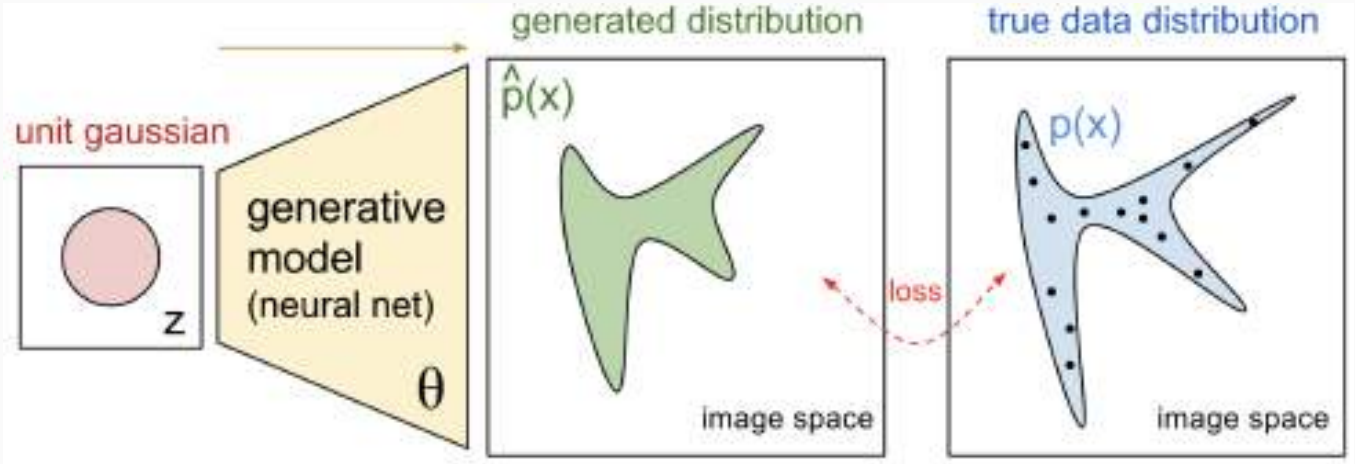
1. Introduction to deep learning
2. Neural networks optimisation
3. Convolutional neural networks
4. Introduction to reinforcement learning - part 1
5. Introduction to reinforcement learning - part 2
6. Sequence models
7. Generative adversarial networks (GANs)
8. Variational autoencoders (VAEs)
9. Normalising flows
10. Theories of deep learning

Outline

Outline

Generative modelling

- Consider again the generative modelling problem of fitting a parametric distribution $p(x, \theta)$ to the data
- Allows us to generate new (synthetic) data examples
- In many applications, generating likely samples may not be enough: we also want to know how likely given data examples are
- GANs and VAEs do not enable exact computation of data likelihood
- Problem may be complicated by conditional dependencies



Normalising flows

- Normalising flows learn invertible transformations of (known) distributions
- They can be used to model rich, multi-modal distributions
- Data likelihood can be directly evaluated
- No reconstruction loss
- Also used for providing flexible, rich distributions for variational inference

source: [Dinh et al., 2016]

Probabilistic change of variables

- A normalising flow is a series of invertible transformations of an initial distribution

$$\mathbf{z}_i = f_i(\mathbf{z}_{i-1}), \qquad i = 1, \ldots, K$$

  where each f_i is invertible, each $\mathbf{z}_i \in \mathbb{R}^N$, and $p(\mathbf{z}_0)$ is known.

- Then $p(\mathbf{z}_K)$ is given by

$$p(\mathbf{z}_K) = p(\mathbf{z}_0) \prod_{i=1}^{K} \left| \det \frac{\partial \mathbf{z}_i}{\partial \mathbf{z}_{i-1}} \right|^{-1}$$

  as density volumes are successively transported by $\left| \det \frac{\partial \mathbf{z}_i}{\partial \mathbf{z}_{i-1}} \right|^{-1}$. Taking logs, we obtain

$$\log p(\mathbf{z}_K) = \log p(\mathbf{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{\partial \mathbf{z}_i}{\partial \mathbf{z}_{i-1}} \right|$$

Probabilistic change of variables

$$
\begin{aligned}
z_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1(z_0) \\
\log p(\mathbf{z}_K) &= \log p(\mathbf{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{\partial \mathbf{z}_i}{\partial \mathbf{z}_{i-1}} \right|
\end{aligned}
$$

- The functions f_i are implemented by a neural network
- Recall these functions need to be invertible
- In practice, the determinant computation could be prohibitively costly
- Normalising flow networks are designed to enable fast computation of this determinant

Outline

Autoregressive models as normalising flows

- Recall that autoregressive networks work by modelling the conditional distributions of the data $\mathbf{x} \in \mathbb{R}^N$

$$p(\mathbf{x}) = \prod_{i=1}^{N} p(x_i | x_{1:i-1})$$

- An example is a model whose conditional density is a Gaussian,

$$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | \mu_i, \sigma_i), \qquad i = 1, \ldots, N$$

where the mean and standard deviation are provided by the network

$$\left.\begin{array}{rcl} \mu_i & = & f_{\mu_i}(x_{1:i-1}) \\ \sigma_i & = & f_{\sigma_i}(x_{1:i-1}) \end{array}\right\} \quad i = 1, \ldots, N$$

Autoregressive models as normalising flows

- To sample from such a model, we sample from a noise vector

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}), \qquad \boldsymbol{\epsilon} \in \mathbb{R}^N$$

  and recursively compute

$$x_i = f_{\sigma_i}(x_{1:i-1})\epsilon_i + f_{\mu_i}(x_{1:i-1}), \qquad i = 1, \ldots, N$$

- This procedure is a deterministic transformation of the underlying noise variable $\boldsymbol{\epsilon}$ and can be interpreted as a normalising flow

$$p(\boldsymbol{\epsilon}) \quad \Rightarrow \quad p(\mathbf{x})$$

- Note that we need to enforce $f_{\sigma_i} > 0$ to ensure invertibility

Inverse autoregressive flow (IAF)

- The x_i are sampled sequentially

$$x_i = f_{\sigma_i}(x_{1:i-1})\epsilon_i + f_{\mu_i}(x_{1:i-1}), \qquad i = 1, \ldots, N$$

- As long as $f_{\sigma_i} > 0$, this transformation can be inverted:

$$\epsilon_i = \frac{x_i - f_{\mu_i}(x_{1:i-1})}{f_{\sigma_i}(x_{1:i-1})}, \qquad i = 1, \ldots, N$$

- Note that the calculation of individual ϵ_i do not depend on each other. Therefore the inverse transformation can be parallelised:

$$\boldsymbol{\epsilon} = (\mathbf{x} - \mathbf{f}_{\boldsymbol{\mu}}(\mathbf{x}))/\mathbf{f}_{\boldsymbol{\sigma}}(\mathbf{x}),$$

  where we denote $\mathbf{f}_{\boldsymbol{\mu}}(\mathbf{x}) := (f_{\mu_i}(x_{1:i-1}))_{i\in\{1,\ldots,N\}}$ and $\mathbf{f}_{\boldsymbol{\sigma}}(\mathbf{x}) := (f_{\sigma_i}(x_{1:i-1}))_{i\in\{1,\ldots,N\}}$

Computing probability densities

- Note that we have $\frac{\partial f_{\mu_i}}{\partial x_j} = \frac{\partial f_{\sigma_i}}{\partial x_j} = 0$ for $j \geq i$ by design, due to the autoregressive property
- This gives us

$$\frac{\partial \epsilon_i}{\partial x_j} = \begin{cases} \frac{1}{f_{\sigma_i}(x_{1:i-1})}, & j = i \\ 0, & j > i \end{cases}$$

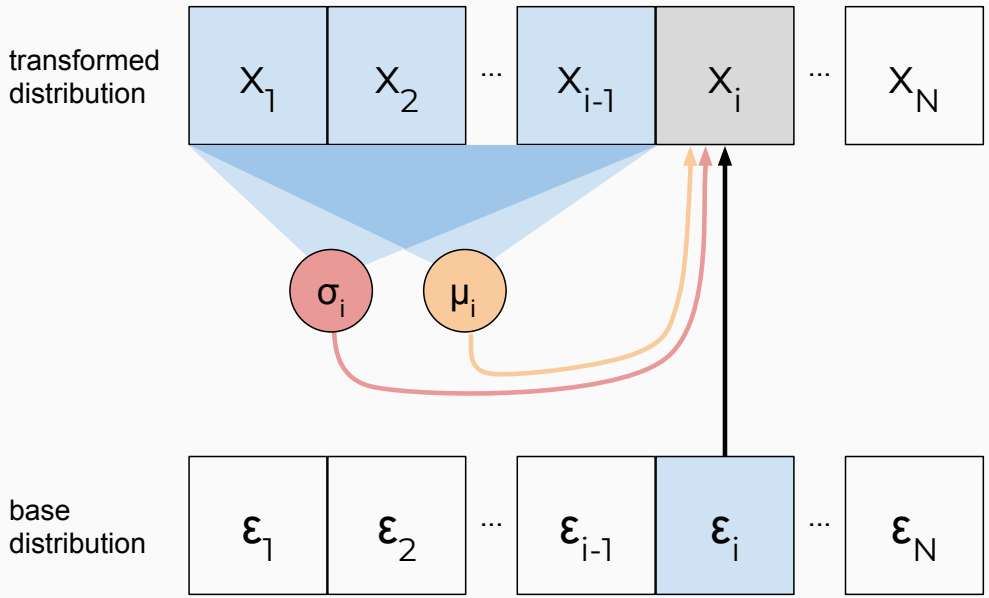
  and so the Jacobian $\partial \boldsymbol{\epsilon} / \partial \mathbf{x}$ is lower triangular, and the determinant can be easily calculated
- The transformed densities can then be computed as follows

$$\begin{aligned} \log p(\mathbf{x}) &= \log p(\boldsymbol{\epsilon}) - \log \left| \det \frac{\partial \boldsymbol{\epsilon}}{\partial \mathbf{x}} \right|^{-1} \\ &= \log p(\boldsymbol{\epsilon}) - \sum_{i=1}^{N} \log f_{\sigma_i}(x_{1:i-1}) \end{aligned}$$

Autoregressive flow - sampling

In summary, sampling data generations is slow (sequential)

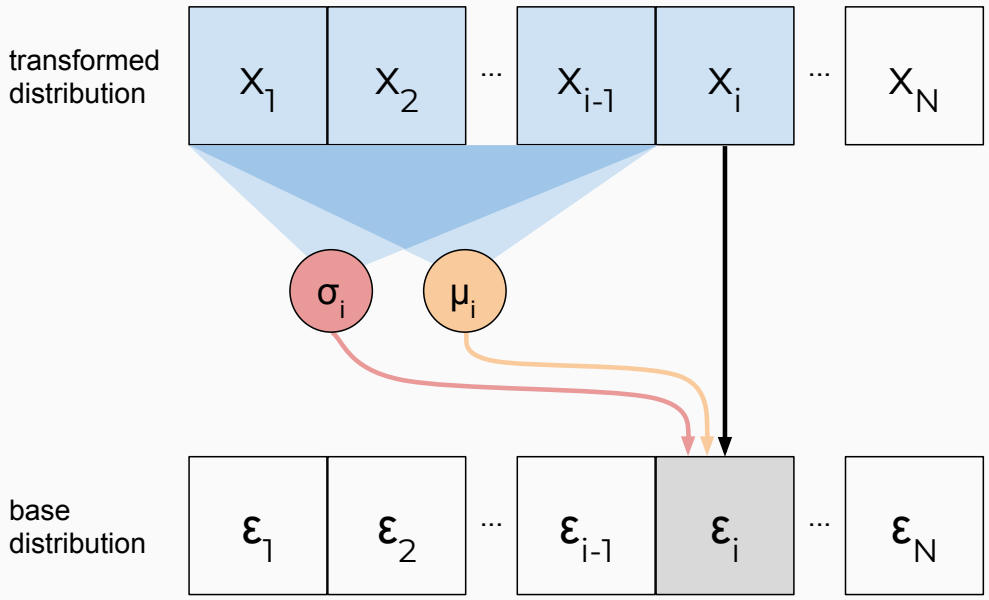
$$
\begin{aligned}
\epsilon &\sim \mathcal{N}(0, \mathbf{I}) \\
x_i &= f_{\sigma_i}(x_{1:i-1})\epsilon_i + f_{\mu_i}(x_{1:i-1}), \qquad i = 1, \ldots, N
\end{aligned}
$$



Inverse autoregressive flow - inference

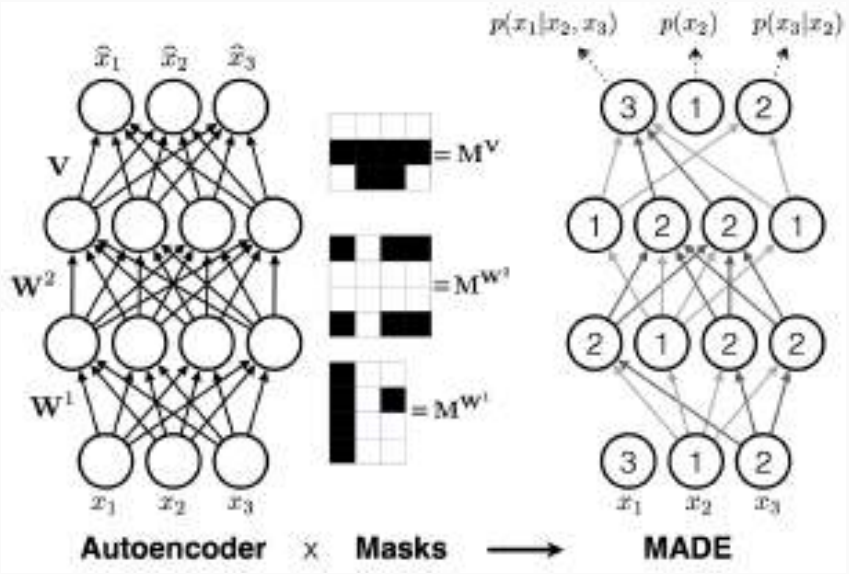
Computing data likelihood is fast (parallelisable)

$$\epsilon_i = \frac{x_i - f_{\mu_i}(x_{1:i-1})}{f_{\sigma_i}(x_{1:i-1})}, \qquad i = 1, \ldots, N$$



Masked autoregressive flow (MAF)

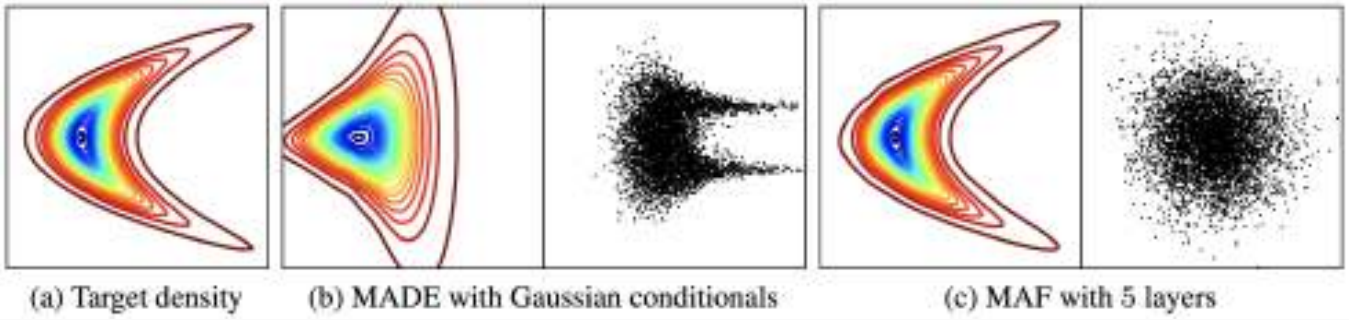
- The masked autoregressive flow (MAF) [Papamakarios et al., 2017] implements such a conditional-Gaussian autoregressive model
- The autoregressive functions $f_{\mu_i}(x_{1:i-1})$ and $f_{\sigma_i}(x_{1:i-1})$ are implemented by a masked autoencoder (MADE)



source: [Germain et al., 2015]

Masked autoregressive flow (MAF)

- Deeper normalising flows can be constructed by stacking several MADE blocks
- Complex, multi-modal distributions can be learned with stacked blocks (note that the conditional distributions of each block is unimodal)

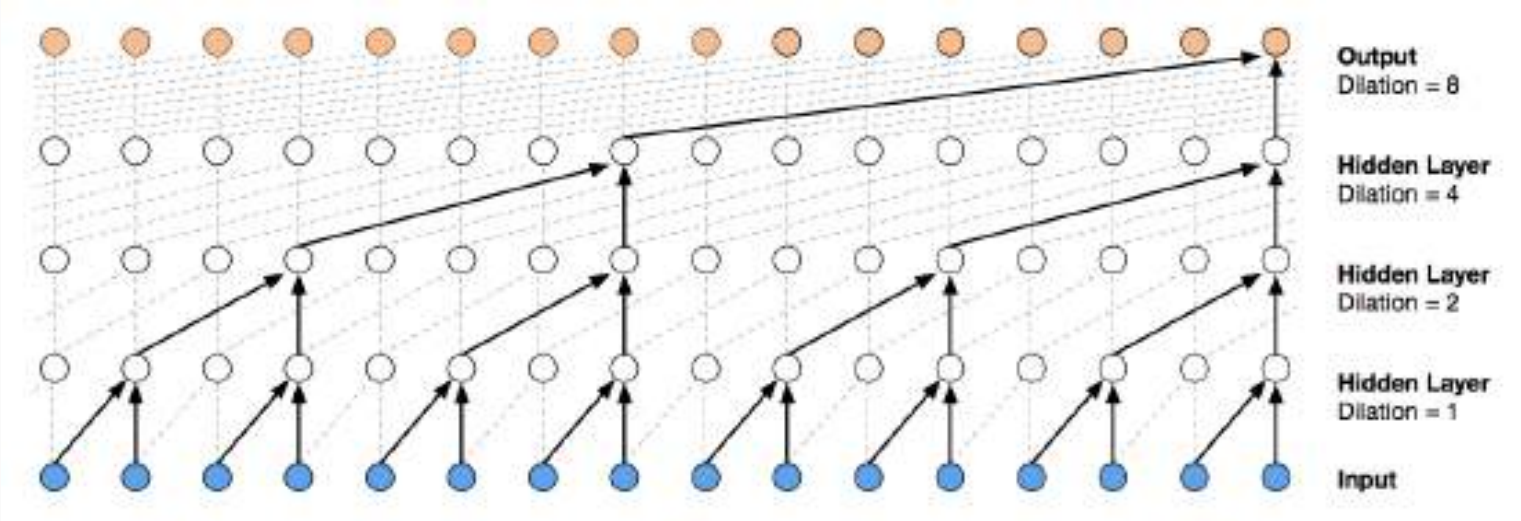


source: [Papamakarios et al., 2017]

Outline

Parallel WaveNet

- Recall the WaveNet architecture for modelling audio data



- It is an autoregressive model that generates audio samples one sample at a time
- Training is fast as it can be done in parallel
- Prohibitively slow sampling for real-world applications

Parallel WaveNet

- Parallel WaveNet aims to treat the slow sampling problem by training a second network that generates samples using IAF
- The base distribution $\mathbf{z} \sim \text{Logistic}(0, \mathbf{I})$ is transformed with IAF to model the distribution $p_X(\mathbf{x})$
- Mean and scale parameters are output by the network at each time step to compute the sample

$$x_t = z_t \cdot s(\mathbf{z}_{<t}, \boldsymbol{\theta}) + \mu(\mathbf{z}_{<t}, \boldsymbol{\theta})$$

- Therefore $p(x_t|\mathbf{z}_{\leq t}, \boldsymbol{\theta})$ follows the logistic distribution:

$$p(x_t|\mathbf{z}_{\leq t}, \boldsymbol{\theta}) = \mathbb{L}(x_t|\mu(\mathbf{z}_{<t}, \boldsymbol{\theta}), s(\mathbf{z}_{<t}, \boldsymbol{\theta}))$$

- The IAF uses the same architecture as the original WaveNet for predicting $\mu(\mathbf{z}_{<t}, \boldsymbol{\theta})$ and $s(\mathbf{z}_{<t}, \boldsymbol{\theta})$.
- Sampling is done in parallel! $> 1000\times$ faster than original WaveNet

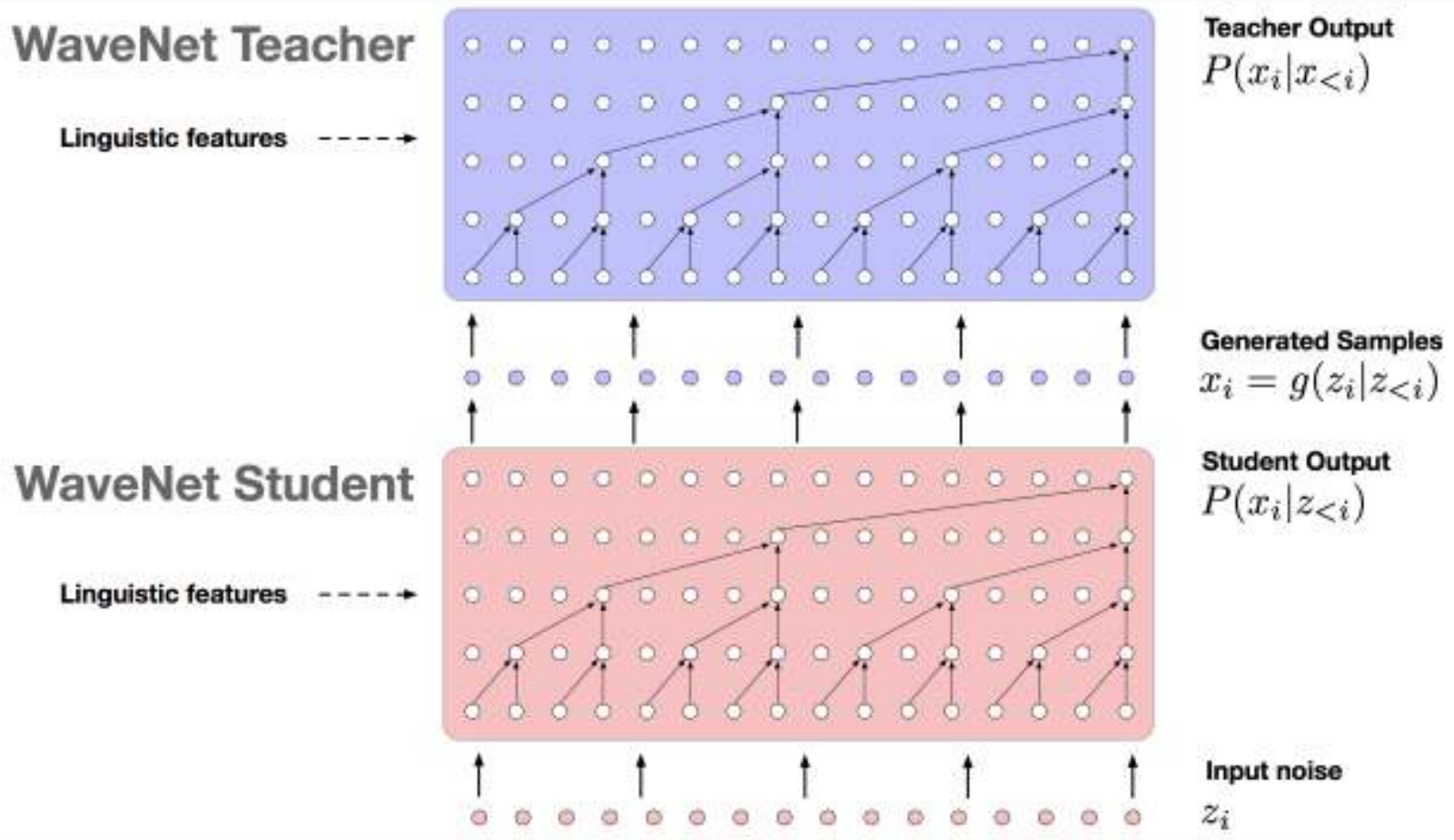
Parallel WaveNet

- The 'teacher' is the pre-trained WaveNet with learned distribution $p_T(\mathbf{x})$
- The 'student' is the IAF model to be trained with distribution $p_S(\mathbf{x})$
- The aim is to distill the knowledge of the teacher into the student
- Samples are drawn from the student whose likelihood are then evaluated by the teacher
- Note that this sampling and evaluation can all be done in parallel
- The student is trained to minimize the divergence

$$D_{KL}(p_S||p_T) = H(p_S, p_T) - H(p_S),$$

where $H(p_S, p_T)$ is the cross entropy between p_S and p_T, and $H(p_S)$ is the entropy of p_S

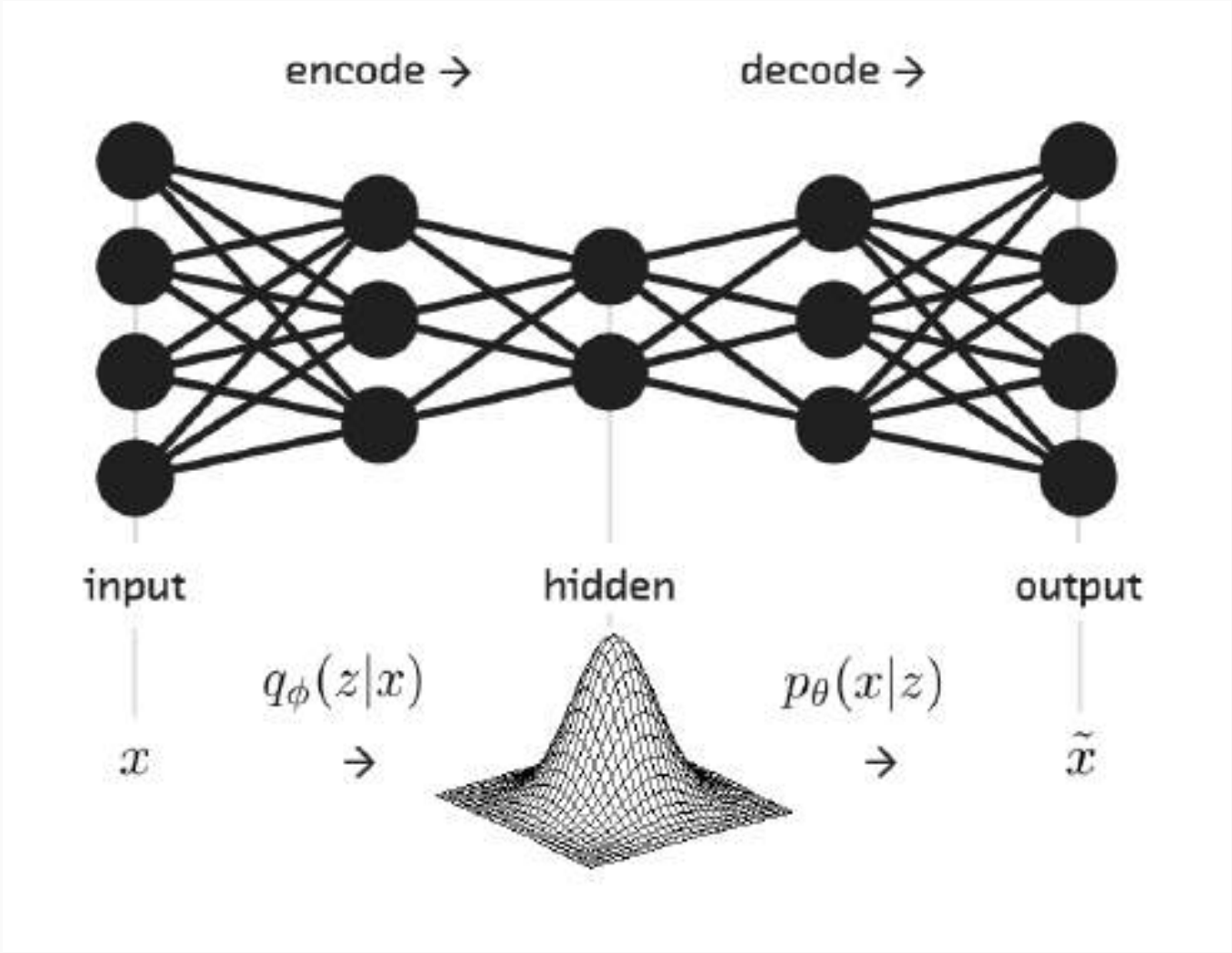
Parallel WaveNet



source:[van den Oord et al., 2018]

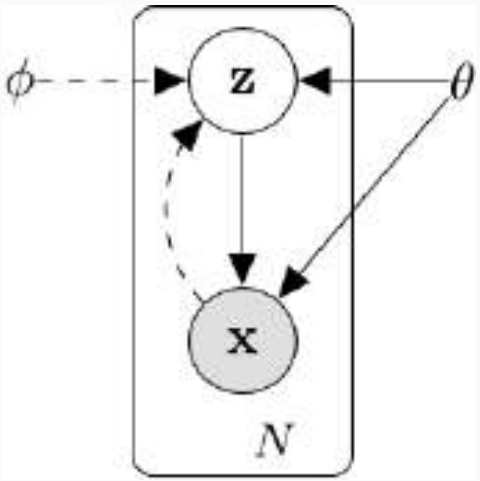
IAF for improved variational inference

Recall the general VAE framework:



Variational autoencoder (VAE) recap

- Let $\mathbf{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$ be observed variables, $\mathbf{z} \in \mathbb{R}^N$ are latent variables
- Choose a prior distribution $\hat{p}_\theta(\mathbf{z})$ over latent variables (often isotropic Gaussian). We will use the hat notation to identify the prior
- Generative model is $p_\theta(\mathbf{x}, \mathbf{z}) = \hat{p}_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z})$
- We want to maximise data log-likelihood



$$\log p(\mathbf{X}) = \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)})$$

- In general this marginal likelihood is intractable to compute or differentiate

Variational autoencoder (VAE) recap

- A solution is to introduce a parametric inference model $q_\phi(\mathbf{z}|\mathbf{x})$ as an approximation to $p_\theta(\mathbf{z}|\mathbf{x})$
- Optimise the variational lower bound for each observation:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] =: \mathcal{L}(\mathbf{x}; \theta)$$

- In fact, we have

$$\mathcal{L}(\mathbf{x}; \theta) = \log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})),$$

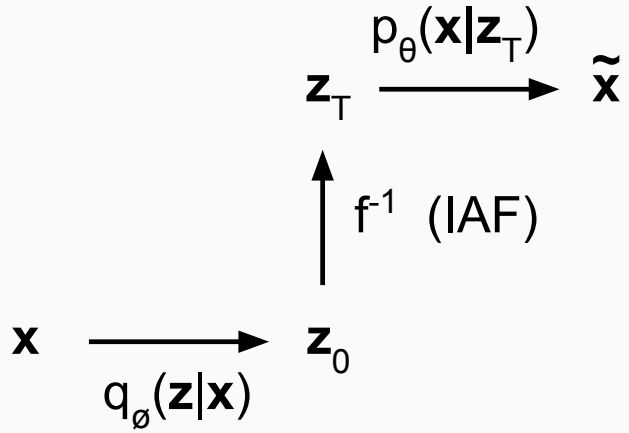
  from which it is clear that maximising $\mathcal{L}(\mathbf{x}; \theta)$ w.r.t. (θ, ϕ) both maximises $\log p_\theta(\mathbf{x})$ and minimises $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$
- Therefore we would like our inference model $q_\phi(\mathbf{z}|\mathbf{x})$ to match the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ as closely as possible

IAF for improved variational inference

- The approximate posterior is often modelled as a diagonal Gaussian: the encoder network outputs parameters $(\boldsymbol{\mu}_0(\mathbf{x}), \boldsymbol{\sigma}_0(\mathbf{x}))$ and we have

$$q_\phi(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_0, \text{diag}(\boldsymbol{\sigma}_0))$$

- This severely restricts the flexibility of the posterior, making it difficult to effectively minimize $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$
- IAF can be used to make the posterior approximation more flexible [Kingma et al., 2016]



- Note that the prior is defined for the variable $\mathbf{z}_T \sim \hat{p}_\theta(\mathbf{z}_T)$

IAF for improved variational inference

- We use several steps of IAF to increase the flexibility of the posterior. The chain is initialised with

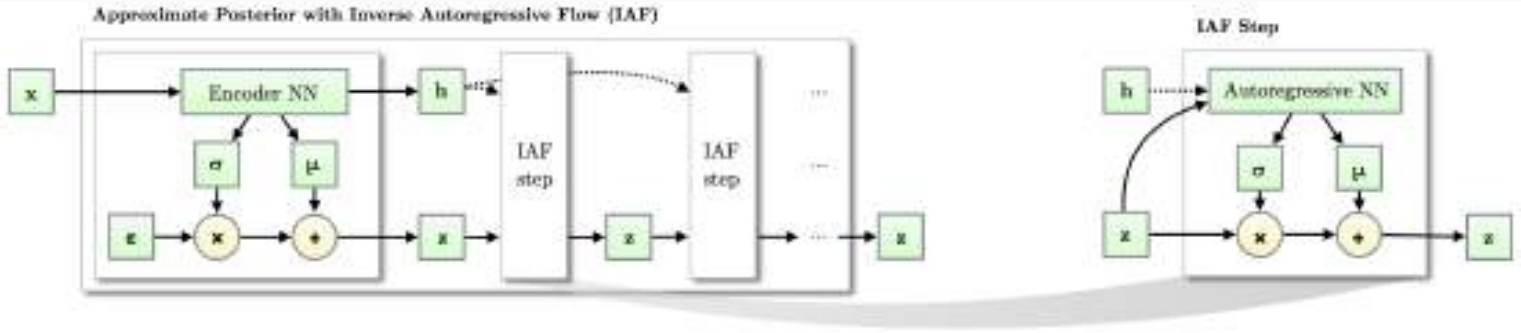
$$\mathbf{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0 \odot \boldsymbol{\epsilon}$$

where $\boldsymbol{\mu}_0$ and $\boldsymbol{\sigma}_0$ come from the encoder network and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$

- The flow then consists of a chain of T transformations

$$\mathbf{z}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1}, \qquad t = 1, \ldots, T,$$

where an IAF flow generates $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$ from $\mathbf{z}_{t-1}$



IAF for improved variational inference

- As before, we have

$$\log q_\phi(\mathbf{z}_T|\mathbf{x}) = \log q_\phi(\mathbf{z}_0|\mathbf{x}) - \sum_{t=1}^{T} \log \left| \det \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_{t-1}} \right|$$

- With the above transformations this becomes

$$\begin{aligned} \log q_\phi(\mathbf{z}_T|\mathbf{x}) &= \log q_\phi(\mathbf{z}_0|\mathbf{x}) - \sum_{t=1}^{T} \sum_{i=1}^{N} \log \sigma_{t,i} \\ &= -\sum_{i=1}^{N} \left(\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T} \log \sigma_{t,i} \right) \end{aligned}$$

IAF for improved variational inference

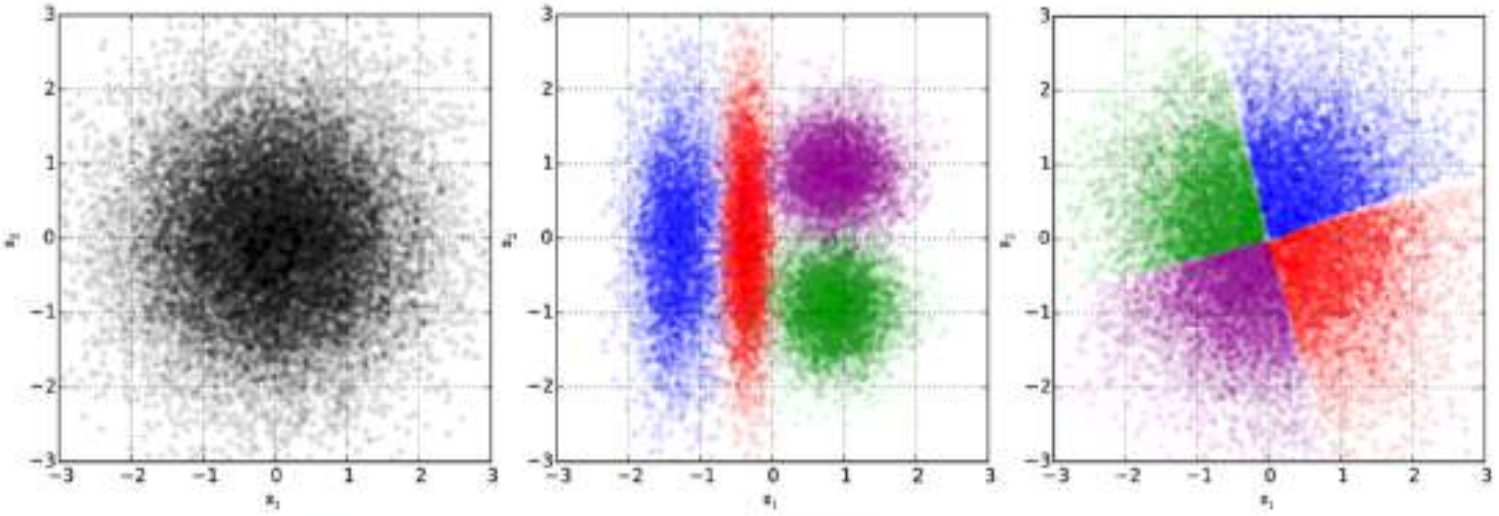
- Finally, recall the VAE objective to be maximised, updated with the extra IAF steps:

$$\mathcal{L}(\mathbf{x};\theta) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}_0|\mathbf{x}),\, \mathbf{z}_T = f^{-1}\mathbf{z}}\left[\log p_\theta(\mathbf{x}|\mathbf{z}_T) + \log \hat{p}_\theta(\mathbf{z}_T) - \log q_\phi(\mathbf{z}_T|\mathbf{x})\right]$$

- Substituting the above expressions for $\log q_\phi(\mathbf{z}_T|\mathbf{x})$ gives

$$\begin{aligned}
\mathcal{L}(\mathbf{x};\theta) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}_0|\mathbf{x}),\, \mathbf{z}_T = f^{-1}\mathbf{z}}\left[\log p_\theta(\mathbf{x}|\mathbf{z}_T) + \log \hat{p}_\theta(\mathbf{z}_T)\right. \\
&\qquad \left. - \left(\log q_\phi(\mathbf{z}_0|\mathbf{x}) - \sum_{t=1}^{T} \log \left|\det \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_{t-1}}\right|\right)\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}_0|\mathbf{x}),\, \mathbf{z}_T = f^{-1}\mathbf{z}}\left[\log p_\theta(\mathbf{x}|\mathbf{z}_T) + \log \hat{p}_\theta(\mathbf{z}_T)\right. \\
&\qquad \left. + \sum_{i=1}^{N}\left(\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T} \log \sigma_{t,i}\right)\right]
\end{aligned}$$

IAF for improved variational inference



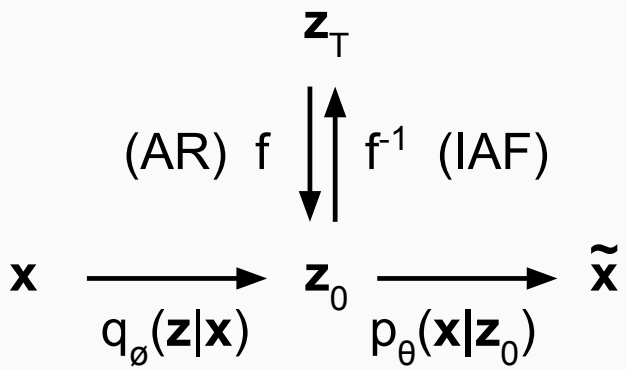
(a) Prior distribution (b) Posteriors in standard VAE (c) Posteriors in VAE with IAF

Figure 1: Best viewed in color. We fitted a variational auto-encoder (VAE) with a spherical Gaussian prior, and with factorized Gaussian posteriors **(b)** or inverse autoregressive flow (IAF) posteriors **(c)** to a toy dataset with four datapoints. Each colored cluster corresponds to the posterior distribution of one datapoint. IAF greatly improves the flexibility of the posterior distributions, and allows for a much better fit between the posteriors and the prior.

source:[Kingma et al., 2016]

Autoregressive prior

- An alternative structure is to specify an autoregressive prior with factorised posterior, instead of IAF posterior with factorised prior:



- We again have $\hat{p}_\theta(\mathbf{z}_T)$ as our factorised prior (e.g. isotropic Gaussian)
- Such an arrangement has no computational training overhead, and allows a more flexible generative model
- Note the trade-off is (slow) sequential sampling of the prior at test time

Autoregressive prior

- Now the prior is transformed according to

$$\log p_\theta(\mathbf{z}_0) = \log \hat{p}_\theta(\mathbf{z}_T) + \sum_{t=1}^{T} \log \left| \det \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_{t-1}} \right|$$

- The VAE objective ends up being very similar:

$$\begin{aligned}
\mathcal{L}(\mathbf{x};\theta) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}_0|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}_0|\mathbf{x}),\, \mathbf{z}_T = f^{-1}\mathbf{z}} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log \hat{p}_\theta(\mathbf{z}_T) \right. \\
&\qquad \left. + \sum_{t=1}^{T} \log \left| \det \frac{\partial \mathbf{z}_t}{\partial \mathbf{z}_{t-1}} \right| - \log q_\phi(\mathbf{z}_0|\mathbf{x}) \right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}_0|\mathbf{x}),\, \mathbf{z}_T = f^{-1}\mathbf{z}_0} \left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log \hat{p}_\theta(\mathbf{z}_T) \right. \\
&\qquad \left. + \sum_{i=1}^{N} \left(\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log(2\pi) + \sum_{t=0}^{T} \log \sigma_{t,i} \right) \right]
\end{aligned}$$

Outline

Nonlinear independent components estimation (NICE)

- Originally motivated by the idea that *a good representation is one in which the data has a distribution that is easy to model*
- The NICE [Dinh et al., 2014] structure has lead to development of Real NVP and Glow architectures for normalising flows
- Look for an invertible nonlinear transformation $\mathbf{z} = f(\mathbf{x})$ such that the latent space components are independent:

$$p_Z(\mathbf{z}) = \prod_d p_{Z_d}(z_d)$$

- As before, we have

$$p_X(\mathbf{z}) = p_Z(f(\mathbf{x})) \left| \det \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|$$

Nonlinear independent components estimation (NICE)

- The core idea is to split $\mathbf{x} \in \mathbb{R}^D$ into two blocks $(\mathbf{x}_{1:d}, \mathbf{x}_{d+1:D})$ and apply a building block transformation from $(\mathbf{x}_{1:d}, \mathbf{x}_{d+1:D})$ to $(\mathbf{y}_{1:d}, \mathbf{y}_{d+1:D})$ of the form

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} + m(\mathbf{x}_{1:d}) \end{aligned}$$

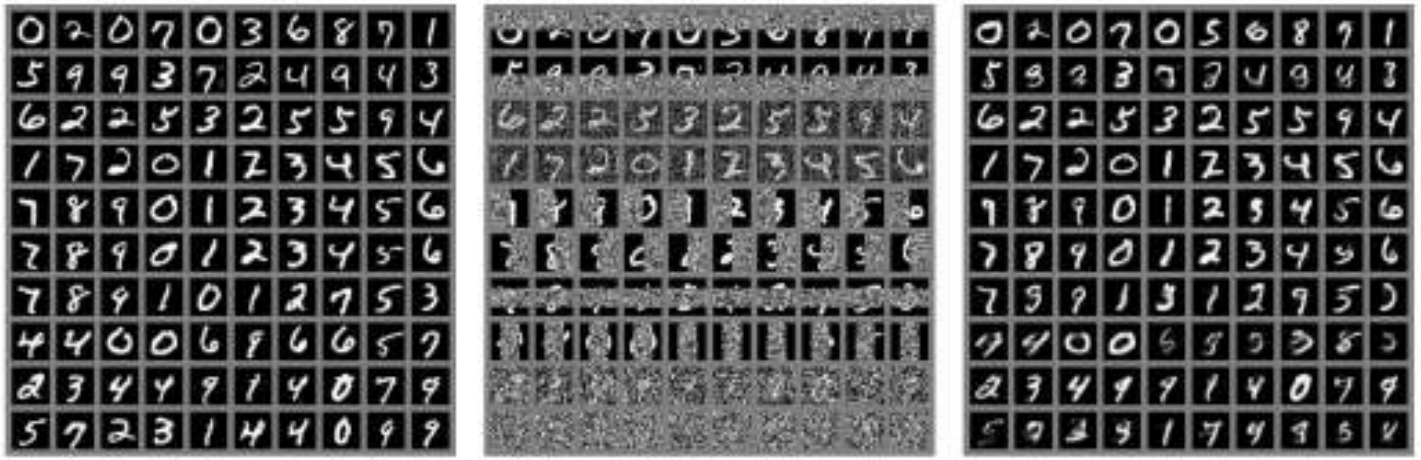
  where m is an arbtrarily complex function from $\mathbb{R}^d \to \mathbb{R}^{D-d}$ (e.g. a neural network)

- This building block coupling layer has a unit Jacobian determinant and is trivially invertible:

$$\begin{aligned} \mathbf{x}_{1:d} &= \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} &= \mathbf{y}_{d+1:D} - m(\mathbf{y}_{1:d}) \end{aligned}$$

Nonlinear independent components estimation (NICE)

- Several coupling layers are composed to obtain a more complex layered transformation
- Note that a coupling layer leaves part of its input unchanged
- The roles of the two subsets are interchanged in alternating layers
- If we examine the Jacobian, we can see that at least three coupling layers are needed to allow all dimensions to influence one another. In the paper, networks were composed with four coupling layers
- The architecture was also used for inpainting, by clamping unmasked pixels and performing projected gradient ascent on the masked pixels



(a) MNIST test examples (b) Initial state (c) MAP inference of the state

Real NVP

- Real NVP stands for real-valued, non-volume preserving transformations [Dinh et al., 2016]
- Builds on the NICE framework
- The affine coupling layer is defined as

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

  where s and t stand for scale and translation, and are functions from $\mathbb{R}^d \to \mathbb{R}^{D-d}$

- The Jacobian is given by

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \operatorname{diag}(\exp[s(\mathbf{x}_{1:d})]) \end{bmatrix}$$

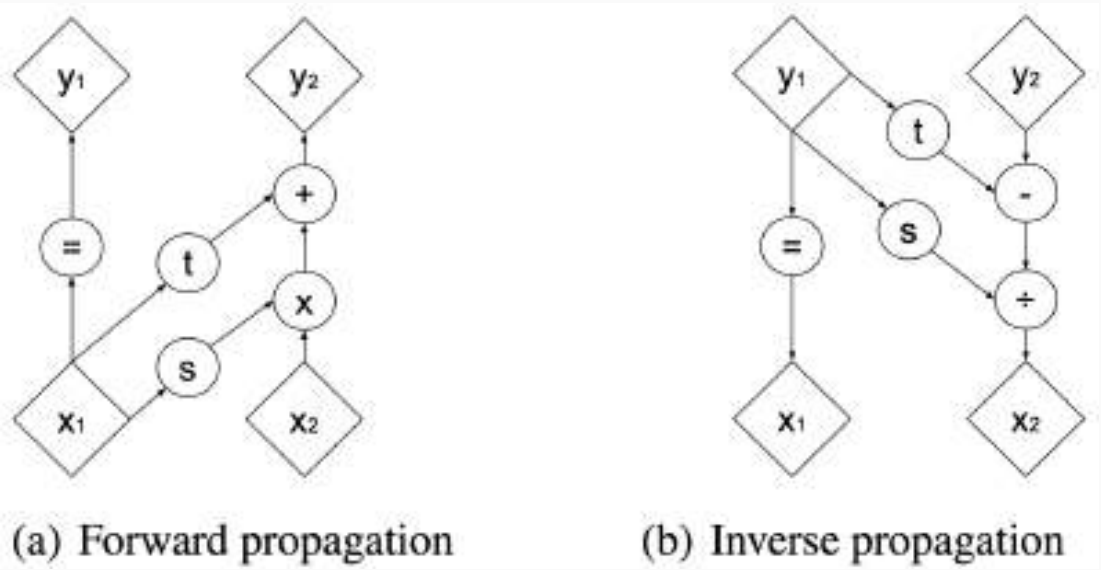
  with determinant equal to $\exp\left[\sum_j s(\mathbf{x}_{1:d})_j\right]$

Real NVP

- As before, computing the inverse transformation is no more complex than the forward propagation:

$$\text{forward} \quad \begin{cases} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \end{cases}$$

$$\text{inverse} \quad \begin{cases} \mathbf{x}_{1:d} &= \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} &= (\mathbf{y}_{d+1:D} - t(\mathbf{y}_{1:d})) \odot \exp(-s(\mathbf{y}_{1:d})) \end{cases}$$



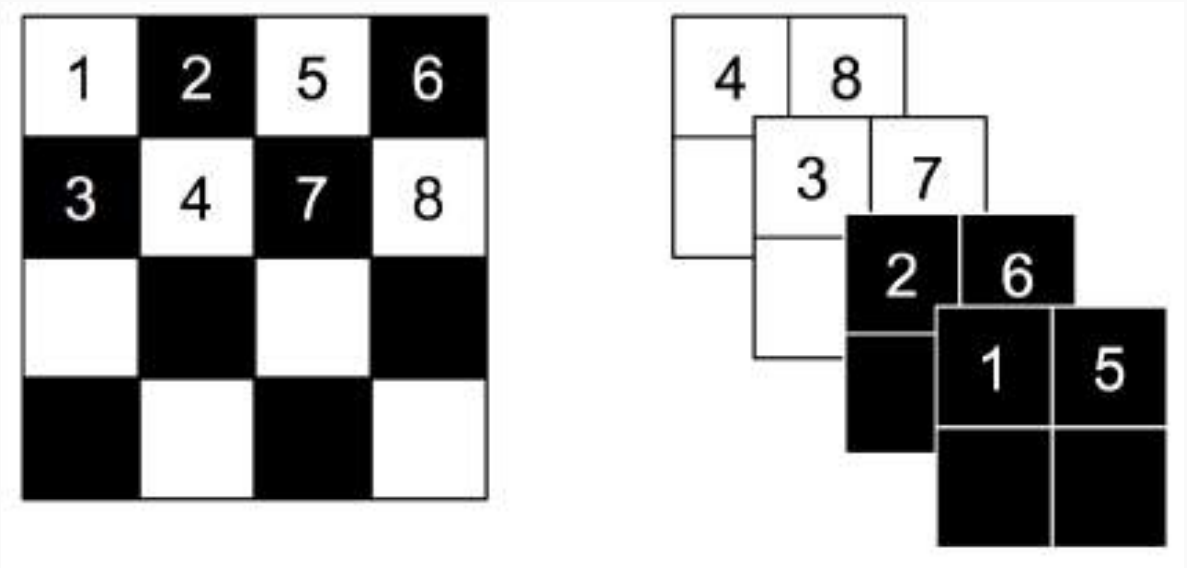
(a) Forward propagation (b) Inverse propagation

Real NVP: spatial and channel-wise masking

- Partitioning can be implemented using a binary mask b:

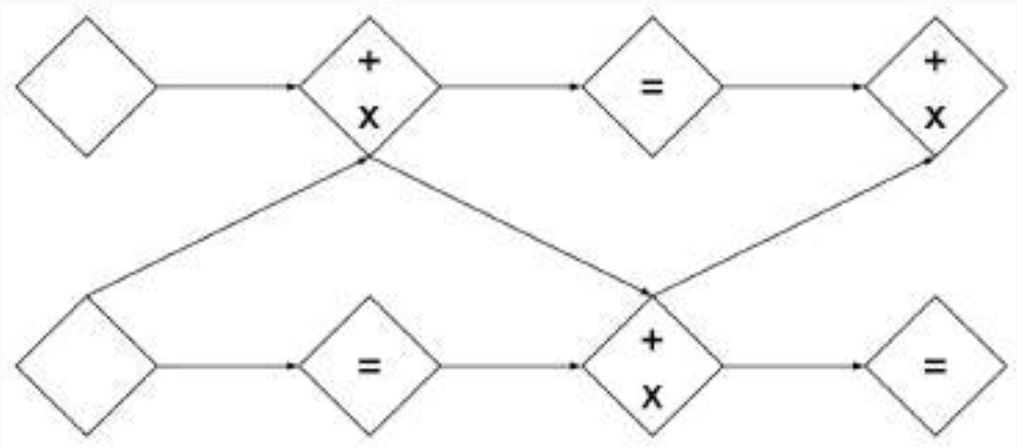
$$\mathbf{y} = b \odot \mathbf{x} + (1 - b) \odot (\mathbf{x} \odot \exp\left[s(b \odot \mathbf{x})\right] + t(b \odot \mathbf{x}))$$

- Real NVP implements two types of masking: spatial checkerboard and channel-wise masking



Real NVP: alternating masks

- As in the NICE framework, we want to ensure that all dimensions are able to interact with each other
- The Real NVP architecture first consists of three layers of alternating checkerboard masks, where the partitions are permuted
- This is followed by a squeeze operation, where the image is divided into $2 \times 2 \times c$ subsquares and reshaped into $1 \times 1 \times 4c$
- Finally, there are three more alternating channel-wise masking layers, again permuting the partition

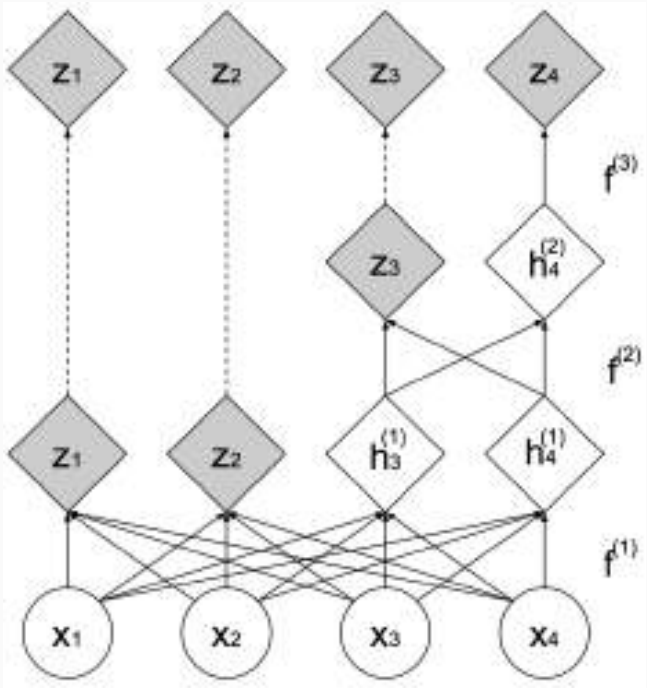


Real NVP: multiscale architecture

- In order to reduce the layer sizes in the deeper layers, dimensions are factored out as latent variables at regular intervals

$$
\begin{aligned}
\mathbf{h}^{(0)} &= \mathbf{x} \\
(\mathbf{z}^{(i+1)}, \mathbf{h}^{(i+1)}) &= f^{(i+1)}(\mathbf{h}^{(i)}), \\
\mathbf{z}^{(L)} &= f^{(L)}(\mathbf{h}^{(L-1)}) \\
\mathbf{z} &= (\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)})
\end{aligned}
$$

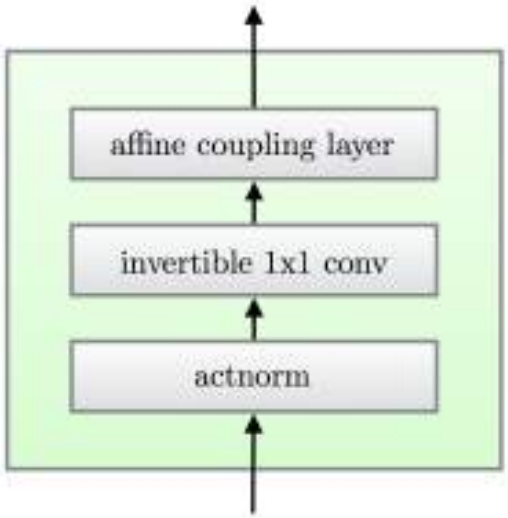
with $i = 0, \ldots, L-2$.



- Batch normalisation is also used in the network. The Jacobian determinant is also easily computed for a batch norm layer as $\left(\prod_i (\tilde{\sigma}_i^2 + \epsilon)\right)^{-\frac{1}{2}}$

Glow

- The Glow [Kingma and Dhariwal, 2018] architecture builds directly on NICE and Real NVP



- One step of the flow consists of an actnorm layer, a 1×1 convolution, and an affine coupling layer
- It disposes of the spatial checkerboard masks
- The 1×1 convolutions are used as an alternative to permuting the channel partition as done in Real NVP

Glow: flow components

- The affine coupling layer is as in Real NVP, where the split (or masking) is along the channel dimension

$$\mathbf{y} = b \odot \mathbf{x} + (1 - b) \odot (\mathbf{x} \odot \exp\left[s(b \odot \mathbf{x})\right] + t(b \odot \mathbf{x}))$$

- The invertible 1×1 convolution is a generalisation of permuting the partition of the channel dimension at each layer.

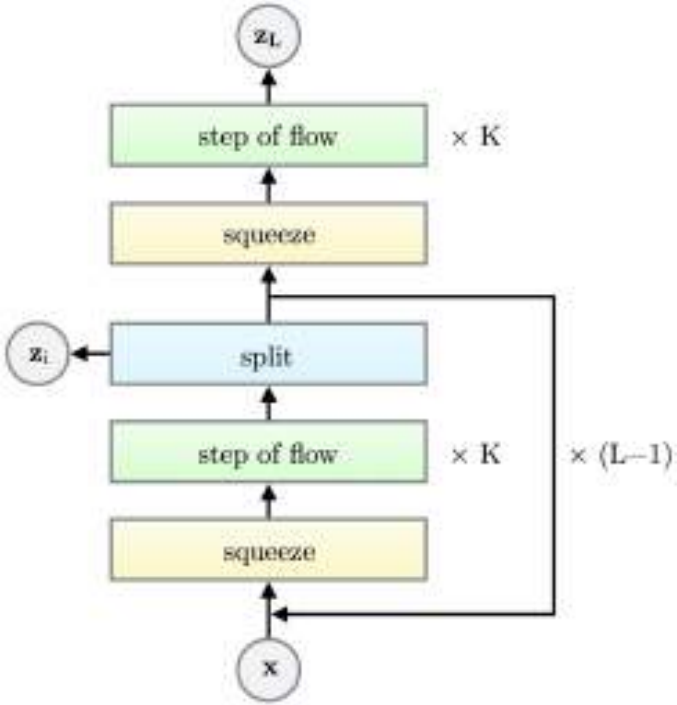
$$\mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j} \qquad \mathbf{W} \in \mathbb{R}^{c \times c}, \forall i, j$$

- The *actnorm* replaces batch norm to mitigate the effect of small minibatches on the sample variance. Learned parameters γ and β are initialised on a minibatch such that activations have zero mean and unit variance

$$\mathbf{y}_{i,j} = \gamma \odot \mathbf{x}_{i,j} + \beta \qquad \forall i, j$$

Glow: multiscale architecture

- Glow also uses a multiscale architecture similar to Real NVP, where the splits are performed along the channel dimension, with the same squeeze operation
- The network has depth of flow K and number of levels L



Glow: flow components

- For each of the layers in the flow, we need to compute the log determinant:

$$\log p_\theta(\mathbf{h}^{(i+1)}) = \log p_\theta(\mathbf{h}^{(i)}) - \left|\log \det \frac{\partial \mathbf{h}^{(i+1)}}{\partial \mathbf{h}^{(i)}}\right|$$

- The affine coupling layer has log determinant equal to $\sum_j \log |s(b \odot \mathbf{x})_j|$ as in Real NVP
- The invertible 1×1 convolution has log determinant $h \cdot w \cdot \log |\det(\mathbf{W})|$, where h and w are the height and width of the image at the given layer
- The actnorm layer has log determinant $h \cdot w \cdot \sum_j \log |\gamma_j|$, again h and w are the height and width of the image at the given layer

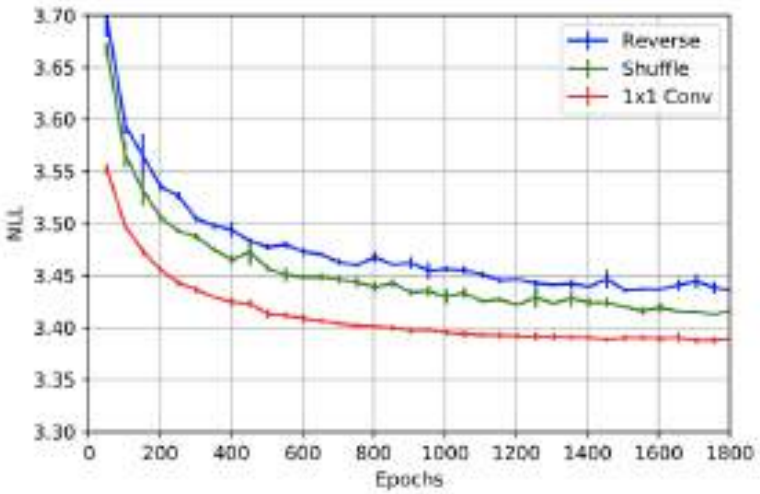
Glow: 1×1 convolution parameterisation

- We need to compute $\det(\mathbf{W})$, which is $\mathcal{O}(c^3)$
- In practice, parameterise $\mathbf{W}$ directly in its LU decomposition

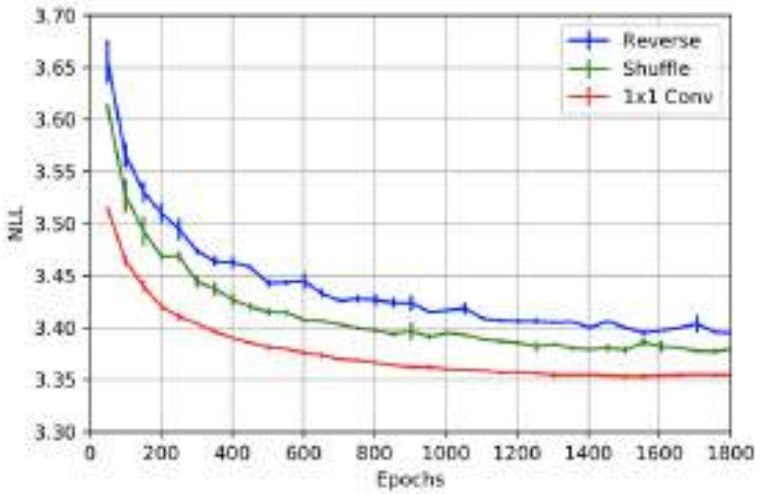
$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \text{diag}(\boldsymbol{\alpha}))$$

  with permutation matrix $\mathbf{P}$, lower triangular matrix $\mathbf{L}$ with ones on the diagonal, upper triangular matrix $\mathbf{U}$ with zeros on the diagonal, and $\alpha \in \mathbb{R}^c$
- Computing $\det(\mathbf{W})$ is then $\mathcal{O}(c)$
- $\mathbf{W}$ is initialised to a random rotation matrix with log determinant equal to zero
- $\mathbf{P}$ is fixed, $\mathbf{L}$, $\mathbf{U}$ and $\boldsymbol{\alpha}$ are optimised

Glow: 1×1 convolution comparison



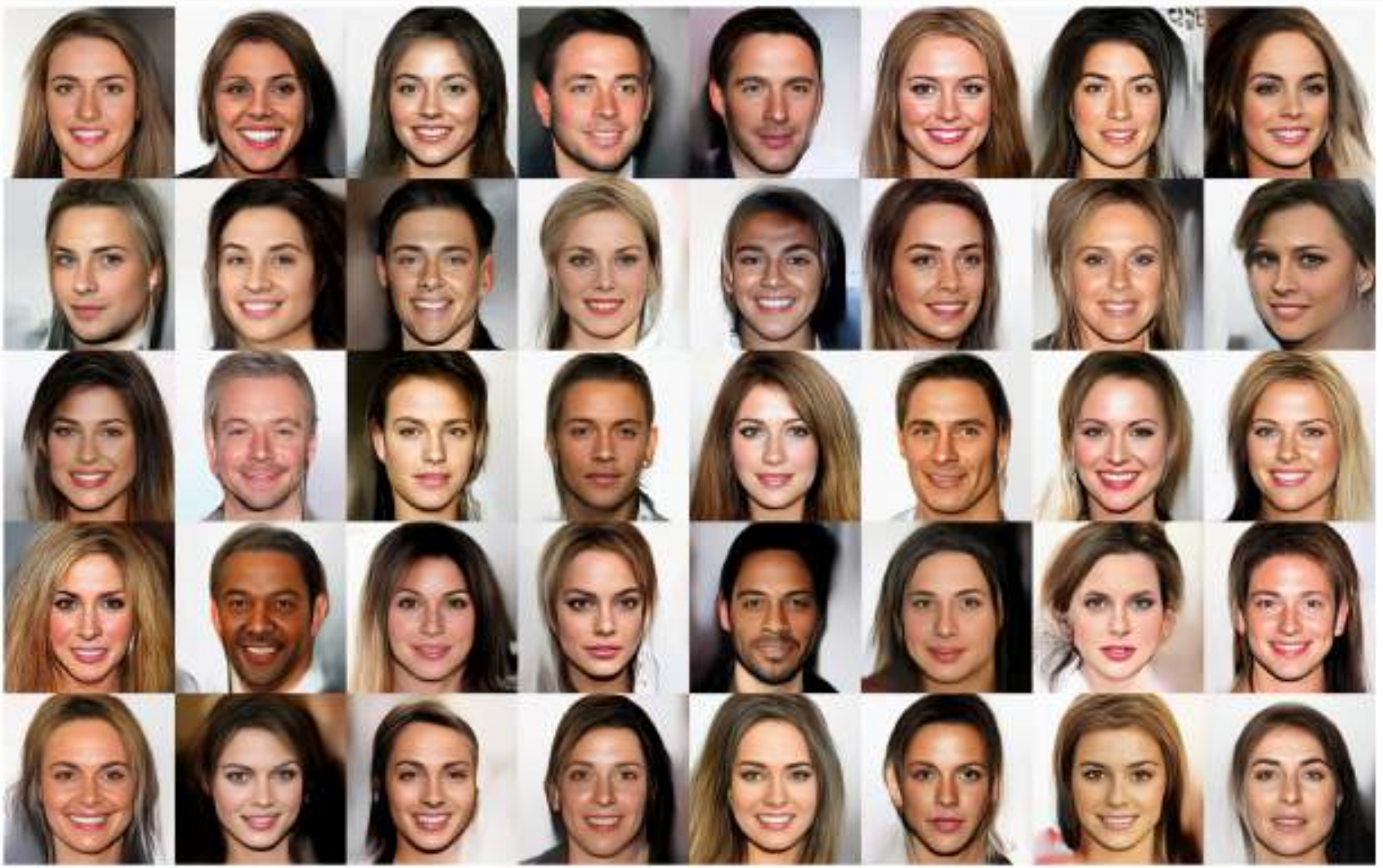
(a) Additive coupling.



(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.

Glow: samples



Glow: latent space interpolation

(a) Smiling

(b) Pale Skin

(c) Blond Hair

(d) Narrow Eyes

(e) Young

(f) Male

GitHub code repositories

Tensorflow

glow waveglow real_nvp iaf maf_tf

Keras

keras-iaf-mnist

PyTorch

real-nvp-pytorch bdir_vae waveglow FloWaveNet pytorch-flows

pytorch-made

References i


Dinh, L., Krueger, D., and Bengio, Y. (2014).
NICE: non-linear independent components estimation.
CoRR, abs/1410.8516.


Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016).
Density estimation using real nvp.
CoRR, abs/1605.08803.


Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015).
Made: Masked autoencoder for distribution estimation.
In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France. PMLR.

References ii

Kingma, D. P. and Dhariwal, P. (2018).
Glow: Generative flow with invertible 1x1 convolutions.
CoRR, abs/1807.03039.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016).
Improved variational inference with inverse autoregressive flow.

In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4743–4751. Curran Associates, Inc.

References iii


Papamakarios, G., Murray, I., and Pavlakou, T. (2017).
Masked autoregressive flow for density estimation.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 2335–2344. Curran Associates, Inc.


Rezende, D. J. and Mohamed, S. (2015).
Variational inference with normalizing flows.
In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1530–1538. JMLR.org.

References iv


van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., van den Driessche, G., Lockhart, E., Cobo, L., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. (2018).
Parallel WaveNet: Fast high-fidelity speech synthesis.
In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3918–3926, Stockholmsmässan, Stockholm Sweden. PMLR.