Generative Models (other than GANs)

AI602: Recent Advances in Deep Learning

Lecture 9

Slide made by

Sungsoo Ahn and Sejun Park

KAIST EE

Table of Contents

Table of Contents

Explicit vs. Implicit Generative Models

- From now on, we study generative models with explicit data distribution:

$$\max_{\theta} \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}^{(n)}), \qquad \boldsymbol{x}^{(n)} : \text{ training samples}$$

model parameters

  - Autoregressive models use conditional distributions to express the target distribution sequentially.
  - Flow-based models warp a simple distribution via invertible transformations to match the target distribution.
  - Variational auto-encoders model the target distribution by maximizing its lower bound for the training dataset.

- Remember generative adversarial network (GAN) models implicit distribution.

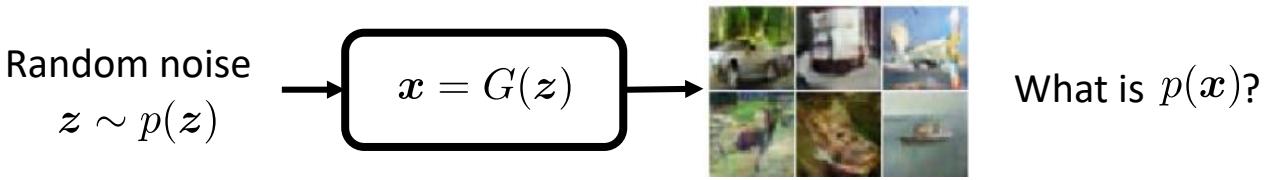
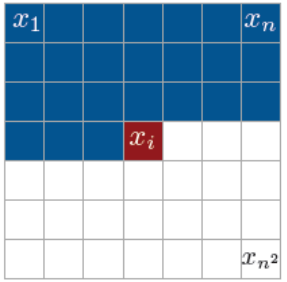


Table of Contents

- Autoregressive generation (e.g., pixel-by-pixel for images) [Oord et al., 2016]:

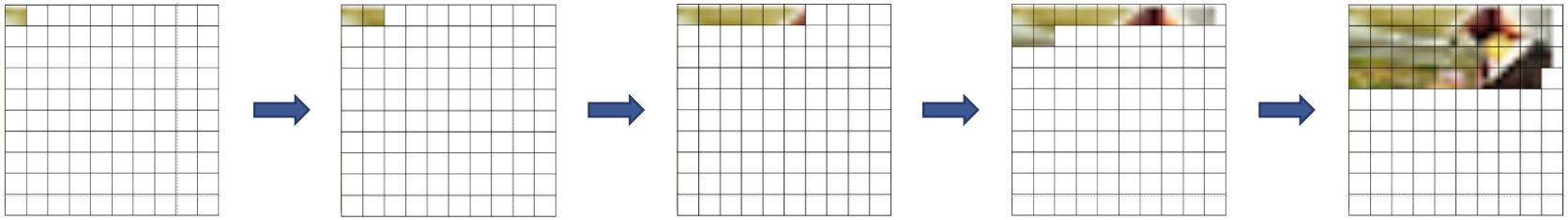
$$p(\boldsymbol{x}) = \prod_{k=1}^{K^2} p(x_k | x_1, \cdots, x_{k-1})$$
$$= \prod_{k=1}^{K^2} p(x_k | \boldsymbol{x}_{<k})$$



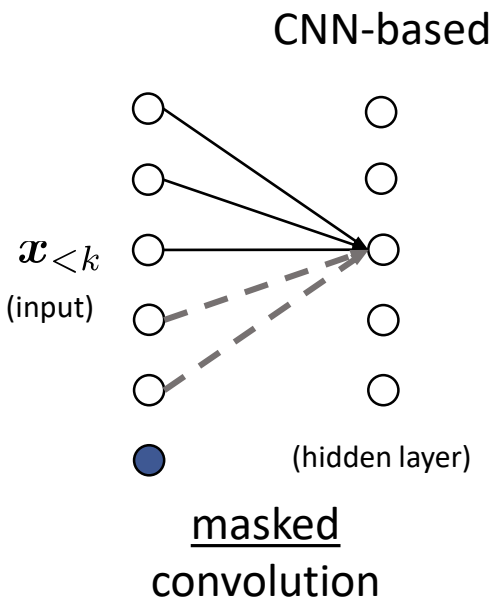
- For example, each RBG pixel is generated autoregressively:

$$p(x_k | \boldsymbol{x}_{<k}) = p(x_{k,R}, x_{k,B}, x_{k,G} | \boldsymbol{x}_{<k})$$
$$= p(x_{k,R} | \boldsymbol{x}_{<k}) p(x_{k,B} | \boldsymbol{x}_{<k}, x_{k,R}) p(x_{k,G} | \boldsymbol{x}_{<k}, x_{k,R}, x_{k,B})$$

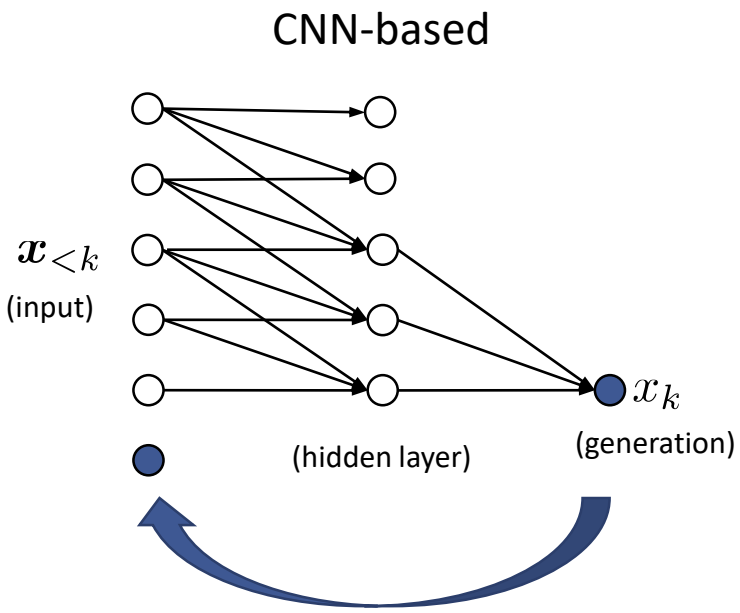
- Each pixel is treated as discrete variables, sampled from softmax distributions:



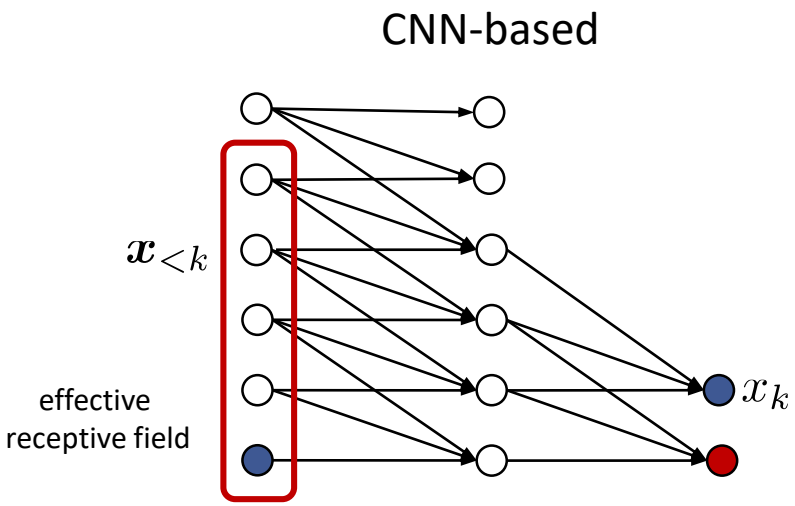
- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:



- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:

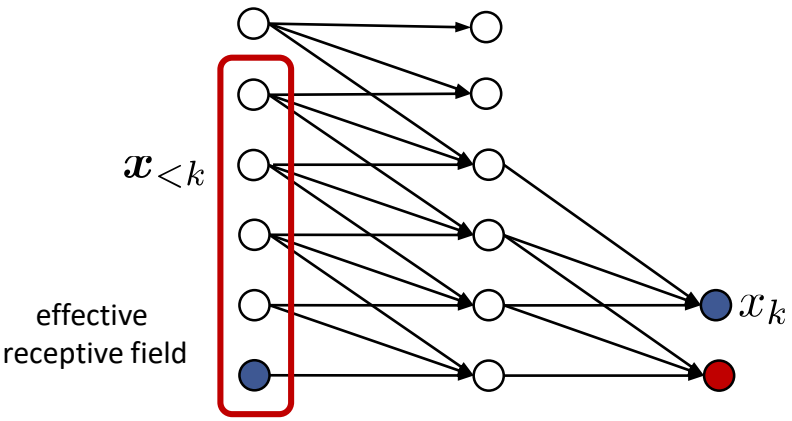


- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:

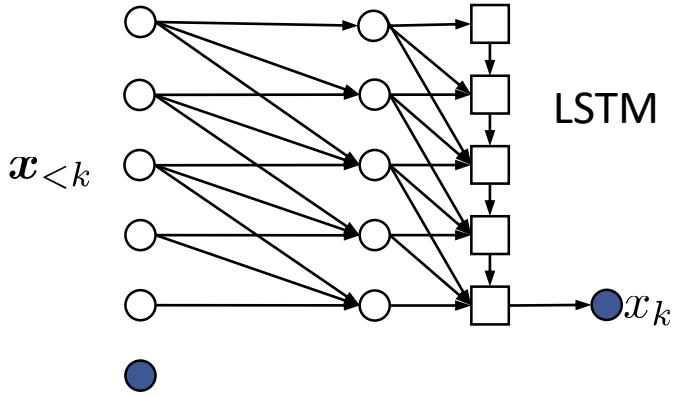


- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:

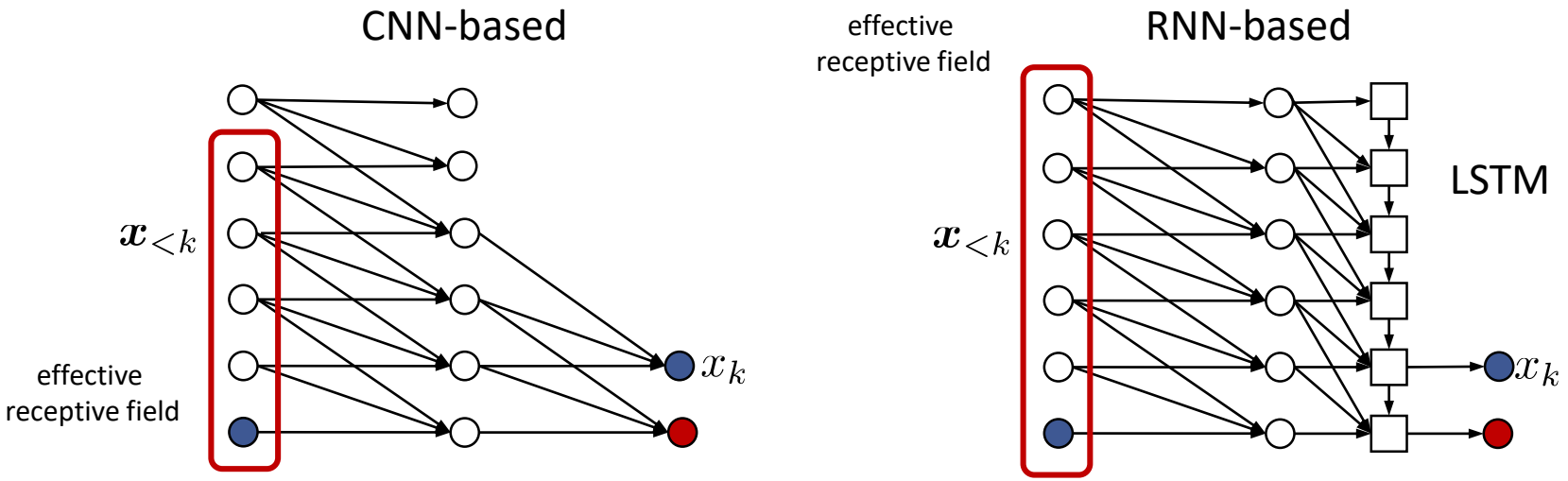
CNN-based



RNN-based



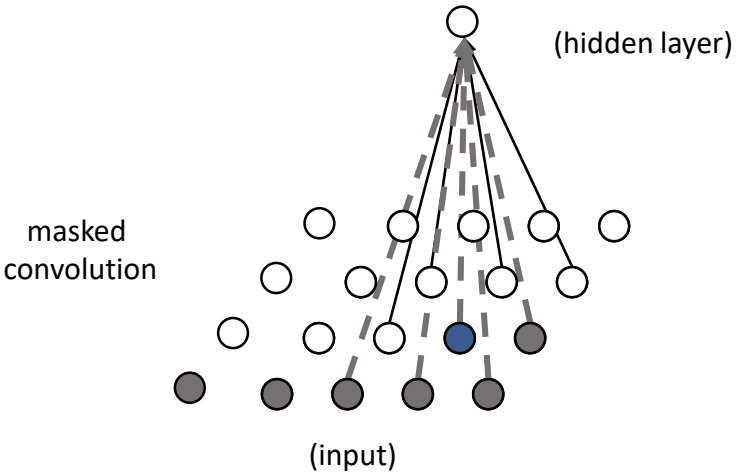
- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Simply treating $\boldsymbol{x}_{<k}$ as one-dimensional (instead of two-dimensional) vector:



  - Image generation requires multiple forward passes for each pixel.
  - Evaluating $p(\boldsymbol{x})$ (at training time) requires single forward pass for CNN, but multiple (sequential, non-parallelizable) LSTM pass are required for RNN (slow).
  - Effective receptive field (context of pixel generation) is unbounded for RNN, but bounded for CNN (constrained).

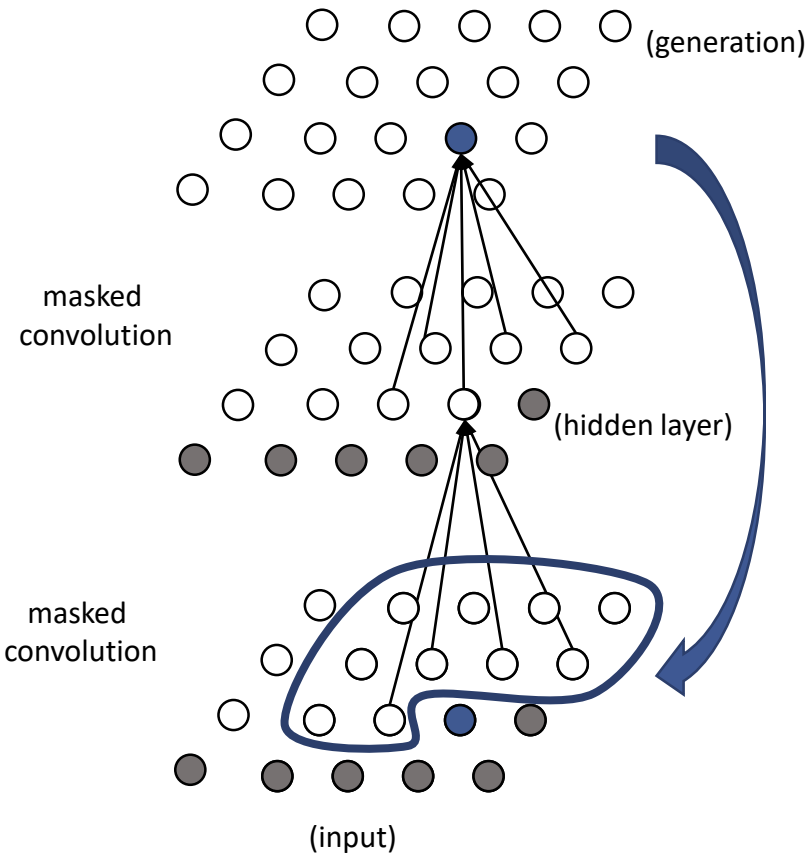
Next, extending to two-dimensional data

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)



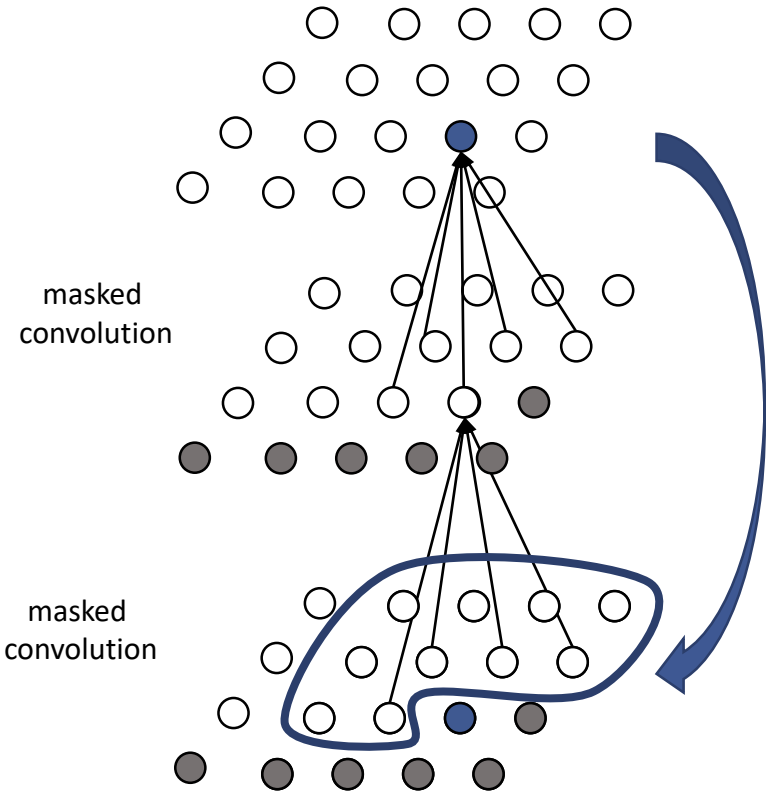
Pixel CNN

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)

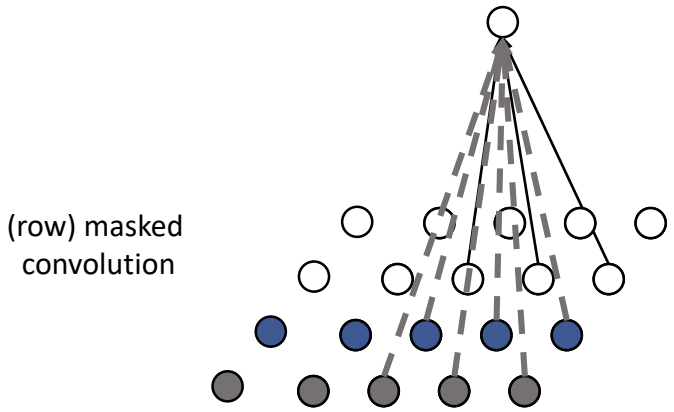


Pixel CNN

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).

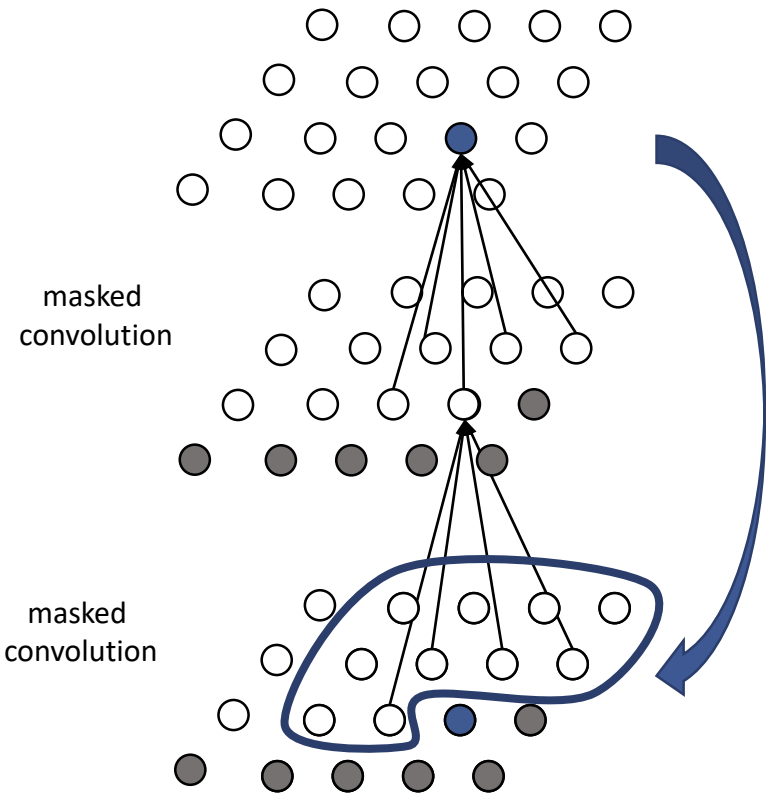


Pixel CNN

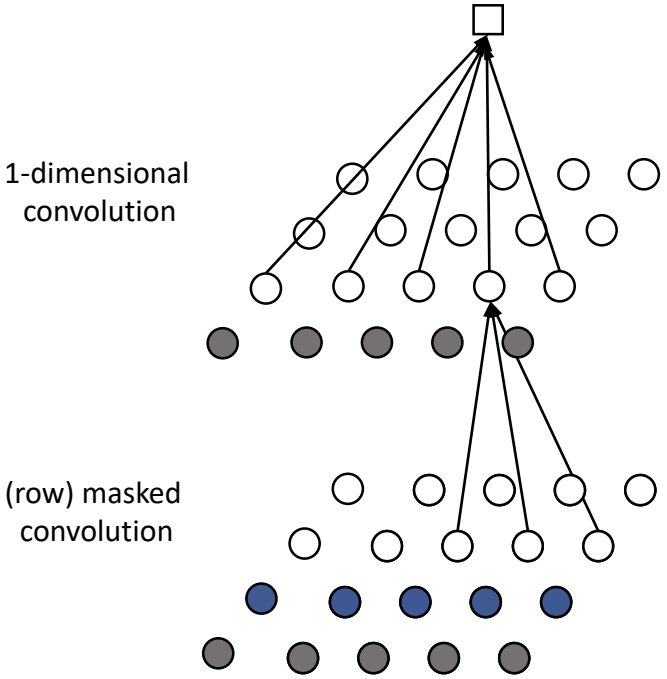


Row LSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).

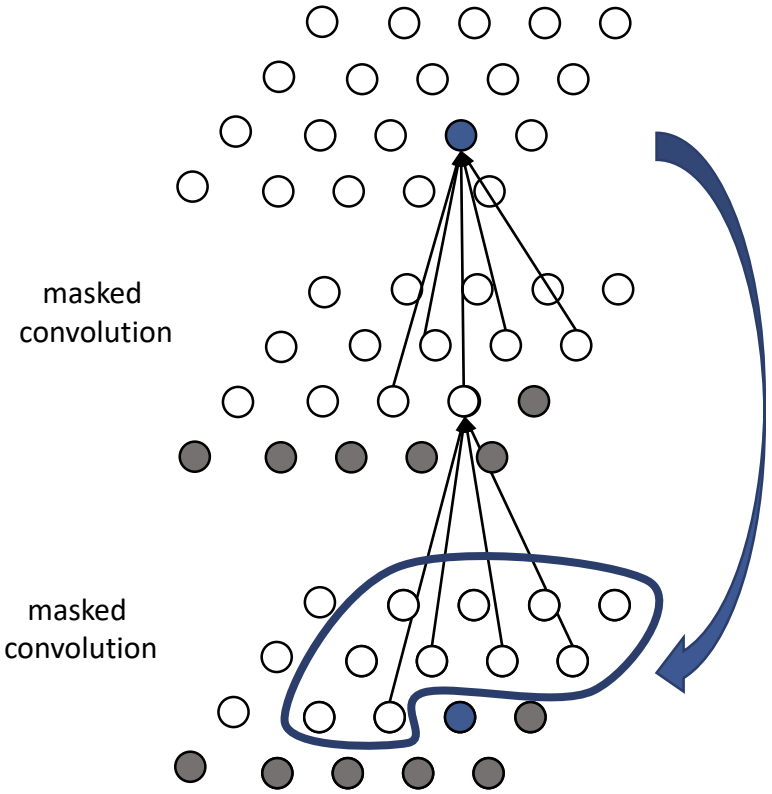


Pixel CNN

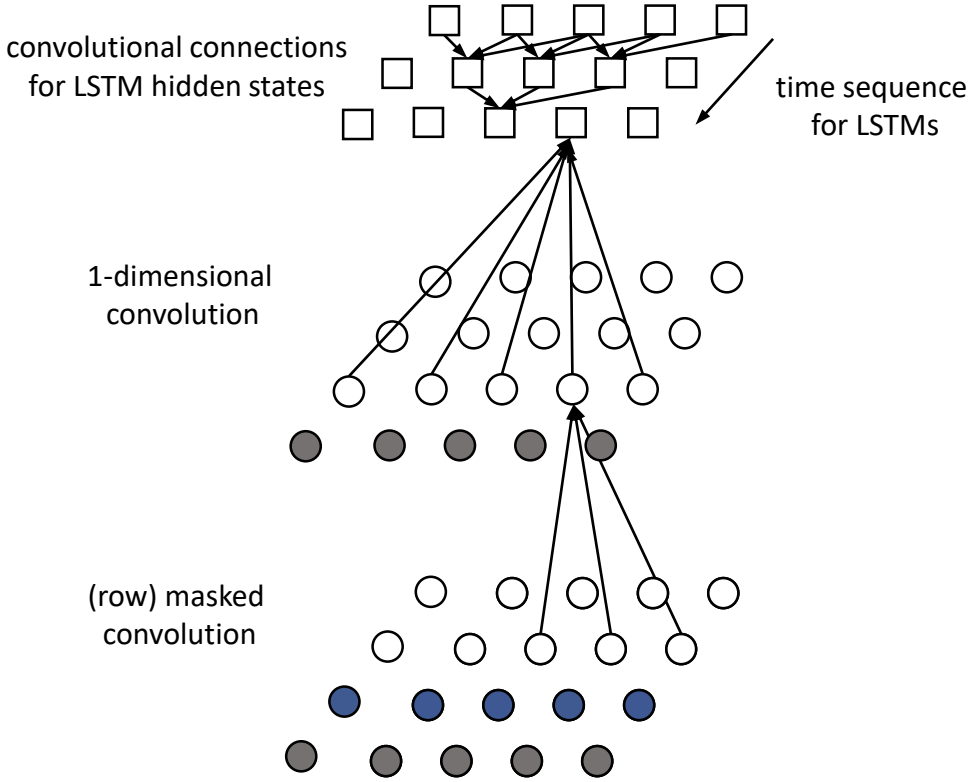


Row LSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).

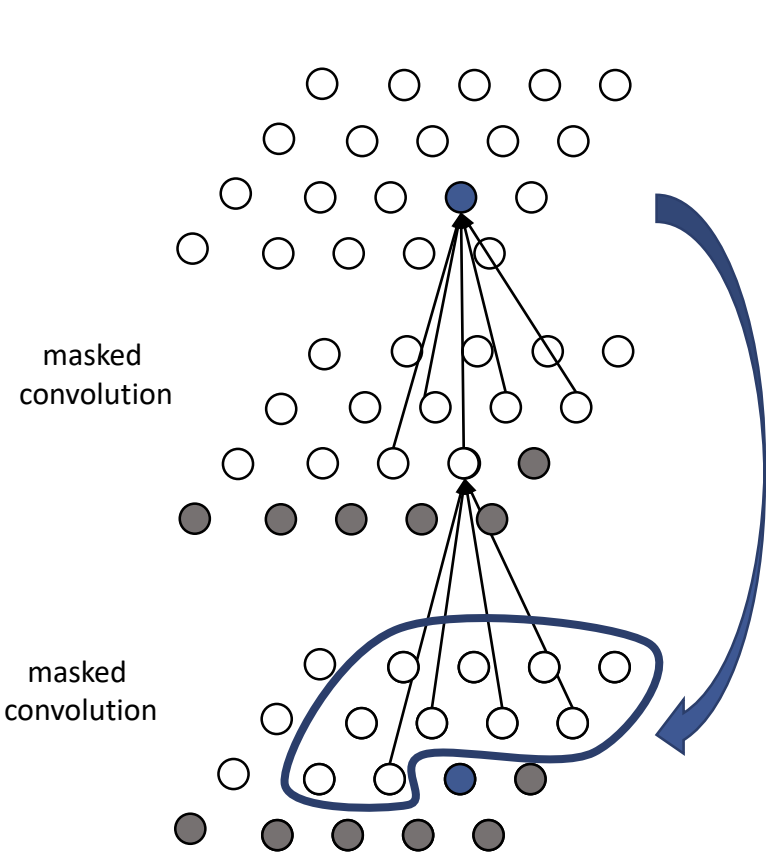


Pixel CNN

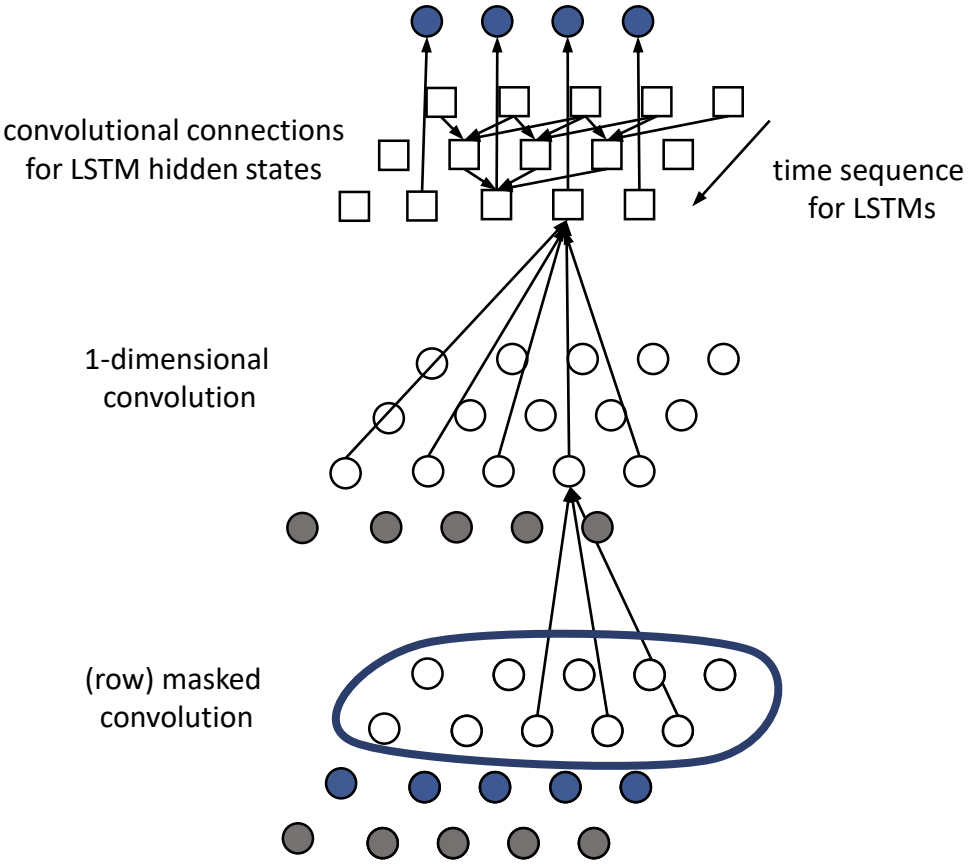


Row LSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).



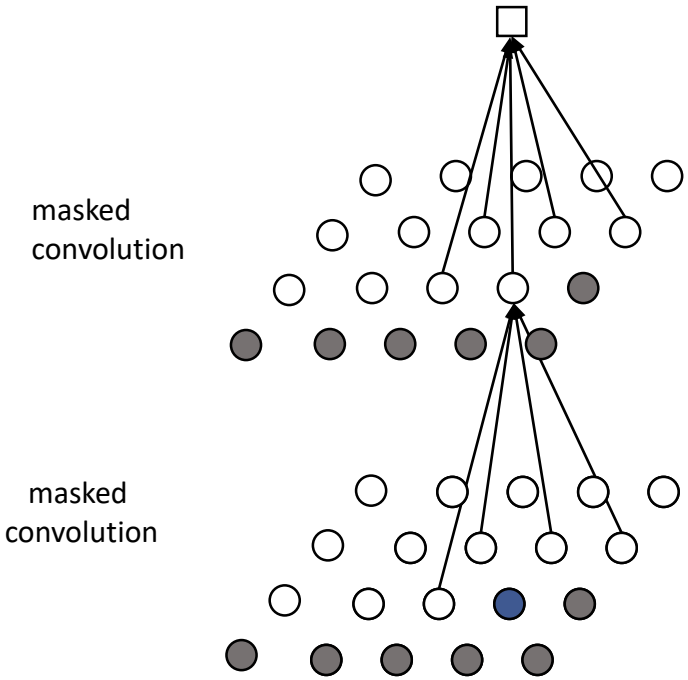
Pixel CNN



Row LSTM

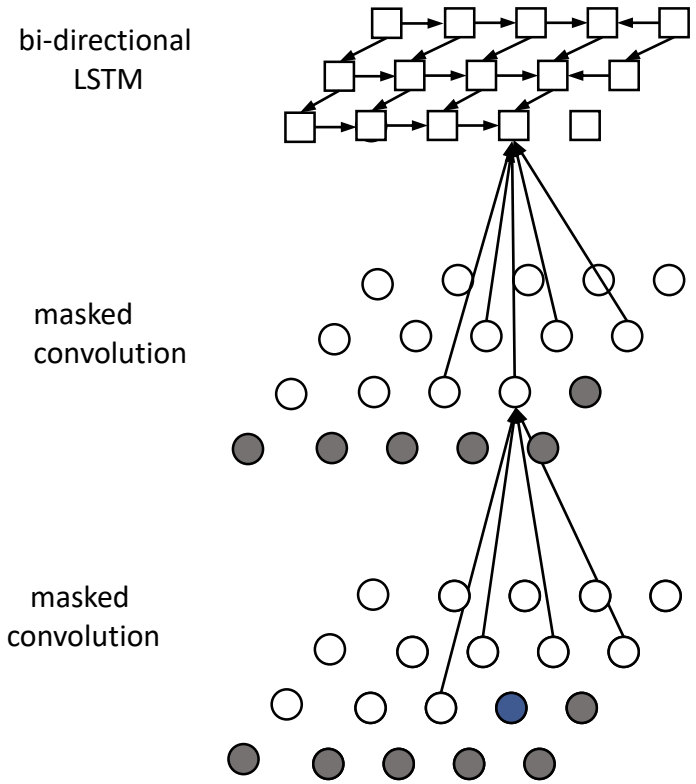
Next, introducing column-wise dependencies using LSTMs

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel.



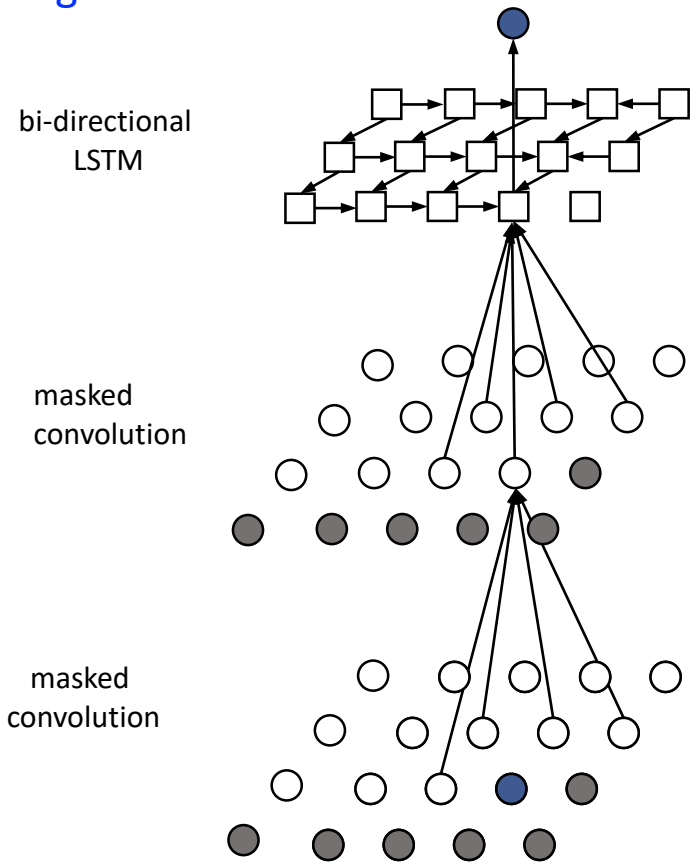
Diagonal BiLSTM

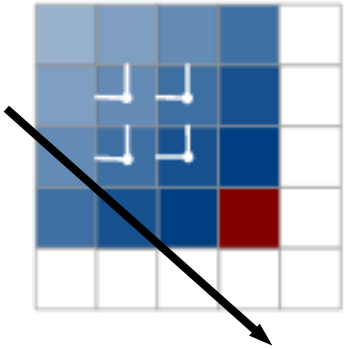
- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel.



Diagonal BiLSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel.

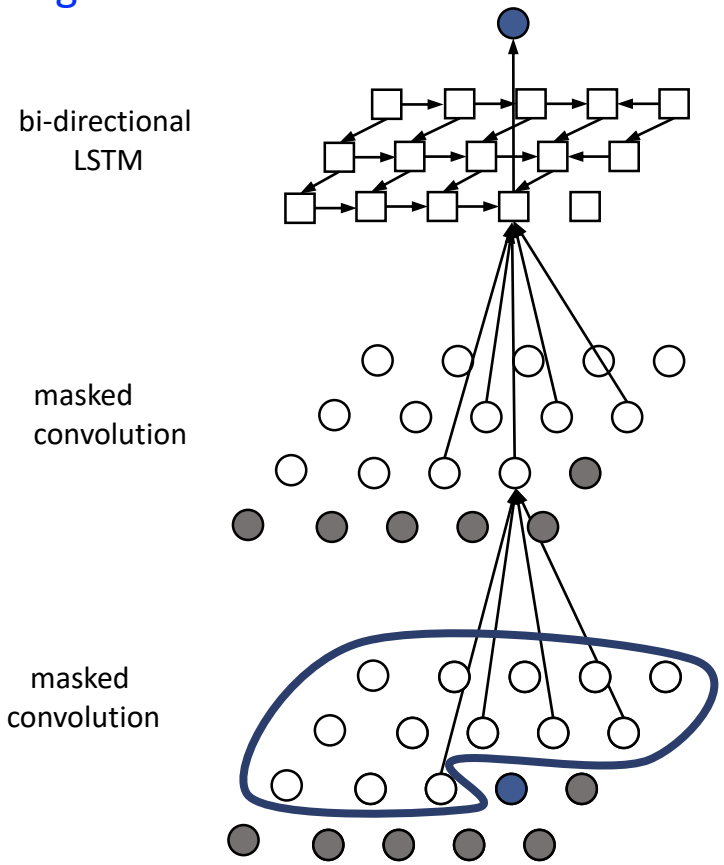


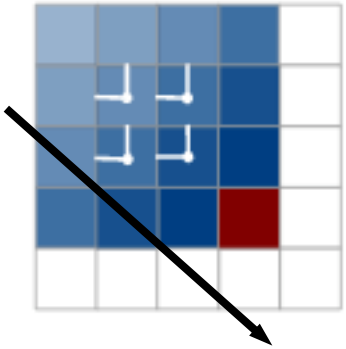


"diagonal"
time sequence
for LSTMs

Diagonal BiLSTM

- Using CNN and RNN for modeling $p(x_k|\boldsymbol{x}_{<k})$
  - Pixel CNN use masked convolutional layer (for $\boldsymbol{x}_{>k}$)
  - Row LSTM use LSTMs, generating image row-by-row (not pixel-by-pixel).
  - Diagonal BiLSTM use bi-directional LSTMs, to generate image pixel-by-pixel.





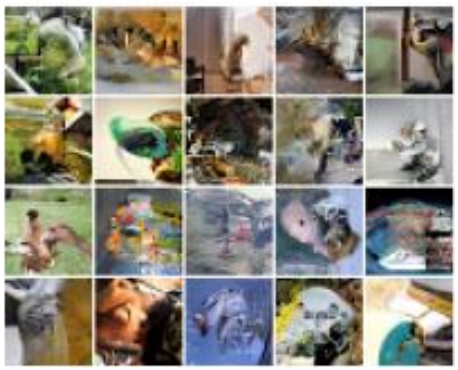
"diagonal"
time sequence
for LSTMs

- Receptive field now covers every pixels generated previously.

Diagonal BiLSTM

- Image generation results from CIFAR-10 and ImageNet:

CIFAR-10



ImageNet

- Evaluation of negative log-likelihood (NLL) on MNIST and CIFAR-10 dataset:

Only explicit models (not GAN) can compute NLL.

Model	NLL Test
PixelCNN:	81.30
Row LSTM:	80.54
Diagonal BiLSTM (1 layer, $h = 32$):	**80.75**
Diagonal BiLSTM (7 layers, $h = 16$):	**79.20**

MNIST

Model	NLL Test (Train)
PixelCNN:	3.14 (3.08)
Row LSTM:	3.07 (3.00)
Diagonal BiLSTM:	**3.00** (2.93)

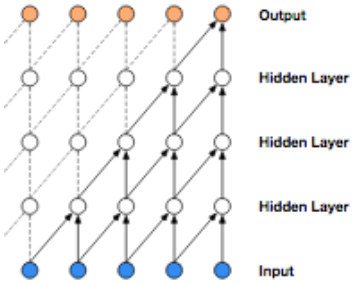
CIFAR-10

- In general, pixel CNN is easiest to train and diagonal BiLSTM performs best.

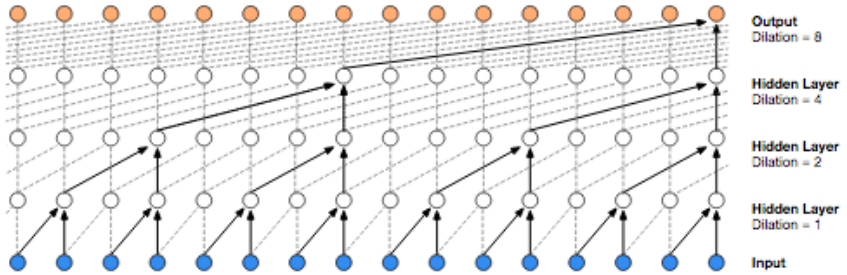
Other Auto-Regressive Models

- Pixel CNN++ [Salimans et al., 2017] improves pixel CNN by replacement of the softmax distribution with discretized logistic mixture likelihood.

- WaveNet [Oord et al., 2017] applies pixel CNN to speech data by introducing dilated convolutional layer for scalability.



convolutional layer



dilated convolutional layer

- Applications of pixelCNN to video [Kalchbrenner et al., 2016a] and machine translation [Kalchbrenner et al., 2016b] have been investigated.

Table of Contents

- Modifying data distribution by flow (sequence) of invertible transformations [Rezende et al., 2015]:

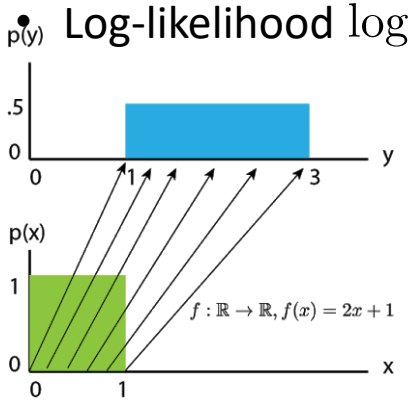
$$\boldsymbol{x} = \boldsymbol{z}_0 \rightarrow \boldsymbol{z}_T = f_T \circ f_{T-1} \circ \cdots f_1(\boldsymbol{z}_0) \qquad \boldsymbol{z}_t \in \mathbb{R}^K$$

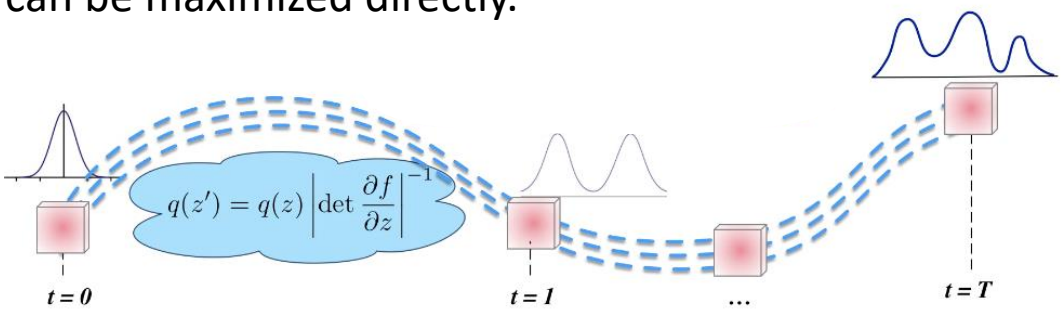
  - Final variable follows some specified prior $p_T(\boldsymbol{z}_T)$.

- Data distribution is explicitly modeled by change-of-variables formula:

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_0) = \log p_T(\boldsymbol{z}_T) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|$$

  - Log-likelihood $\log p(\boldsymbol{x})$ can be maximized directly.





* source: Jang, https://blog.evjang.com/2018/01/nf1.html,
Mohamed et al., https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf

- Modifying data distribution by flow (sequence) of invertible transformations [Rezende et al., 2015]:

$$\boldsymbol{x} = \boldsymbol{z}_0 \rightarrow \boldsymbol{z}_T = f_T \circ f_{T-1} \circ \cdots f_1(\boldsymbol{z}_0) \qquad \boldsymbol{z}_t \in \mathbb{R}^K$$

  - Final variable follows some specified prior $p_T(\boldsymbol{z}_T)$.

- Data distribution is explicitly modeled by change-of-variables formula:

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_0) = \log p_T(\boldsymbol{z}_T) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|$$

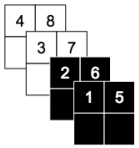
  - Log-likelihood $\log p(\boldsymbol{x})$ can be maximized directly.
  - For training, important to design transformations with tractable computation of $\log \left| \det \left(\partial f_t(\boldsymbol{z}_{t-1}) / \partial \boldsymbol{z}_{t-1} \right) \right|$, which takes $\mathcal{O}(K^3)$ times.

Next, introducing a flexible and tractable form of invertible transformation

- Coupling layer $\boldsymbol{z}_t = f_t(\boldsymbol{z}_{t-1})$ for flow with tractable inference [Dinh et al., 2017]:
  1. Partition the variable into two parts:

$$\boldsymbol{z}_{t-1} \rightarrow [\boldsymbol{z}_{t-1,1:d}, \boldsymbol{z}_{t-1,d+1:K}]$$

spatial-partition

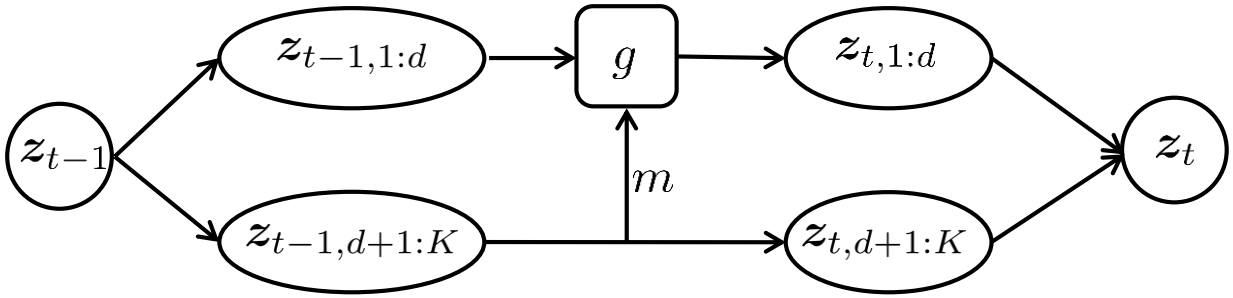


channel-partition

  2. Coupling law defines a simple invertible transformation of the first partition given the second partition (g and m are described later).

$$\boldsymbol{z}_{t,d+1:K} = g(\boldsymbol{z}_{t-1,d+1:K}; m(\boldsymbol{z}_{t-1,1:d}))$$

  3. Second partition is left invariant ($\boldsymbol{z}_{t,1:d} = \boldsymbol{z}_{t-1,1:d}$).



Next, specifying a particular effective form of the coupling layer

- Affine coupling layer was shown to be effective in practice:

$$\begin{aligned} \boldsymbol{z}_{t,d+1:K} &= g(\boldsymbol{z}_{t-1,d+1:K}; m(\boldsymbol{z}_{t-1,1:d})) \\ &= \boldsymbol{z}_{t-1,d+1:K} \odot \exp(m_1(\boldsymbol{z}_{t-1,1:d})) + m_2(\boldsymbol{z}_{t-1,1:d}) \end{aligned}$$

element-wise product

neural networks

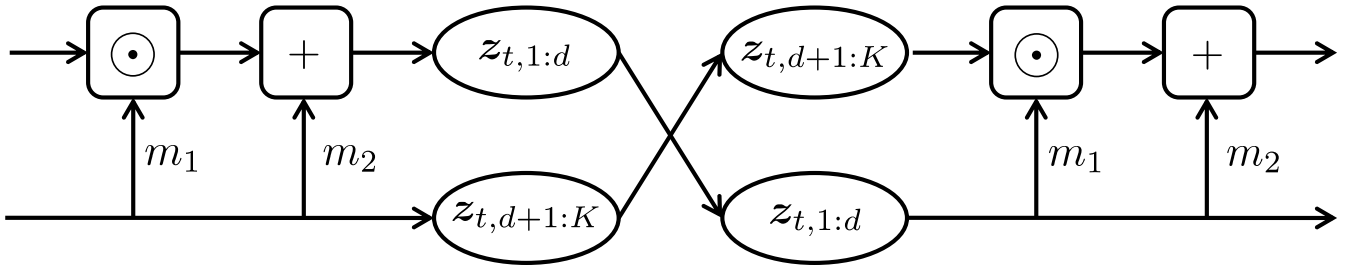
- Jacobian of each transformation becomes a lower triangular matrix:

$$\frac{\partial f_{t-1}(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} = \begin{bmatrix} \mathrm{diag}(\exp(m_2(\boldsymbol{z}_{t-1,1:d}))) & \mathbf{0} \\ \frac{\partial f_{t-1}(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} & \mathbb{I} \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ & a_{22} & 0 & \vdots \\ & & \ddots & 0 \\ & & & a_{KK} \end{bmatrix}$$

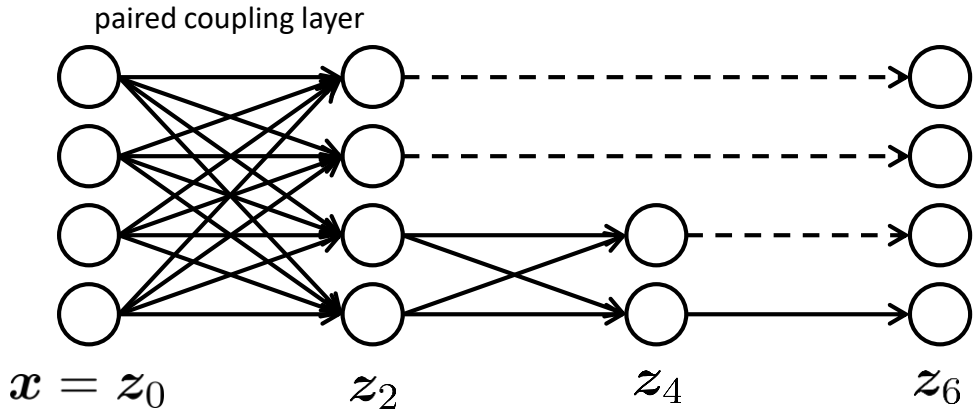
- Inference for such transformations can be done in tractable time.
  - Determinant of lower triangular matrix can be computed in $\mathcal{O}(K)$ time.

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_0) = \log p_T(\boldsymbol{z}_T) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|$$

- For each coupling layer, there exists asymmetry since the first partition $z_{t-1,1:d}$ is left invariant.
  - Two coupling layers are paired alternatively to overcome this issue.



- Multi-scale architectures are used.
  - Half variables follow Gaussian distribution at each scale.



$$p(z_3) = \mathcal{N}(z_3; \mathbf{0}, \mathbf{1})$$

- Test results for log-likelihood of test images:

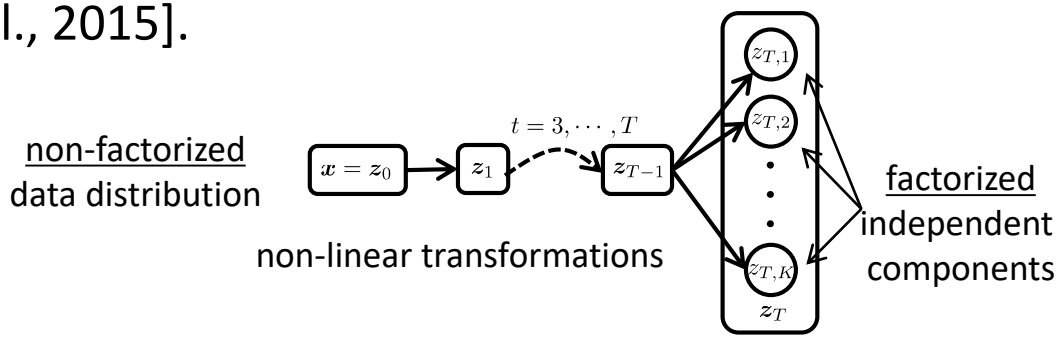
Done by forward pass of functions $\boldsymbol{x} = \boldsymbol{z}_0 \rightarrow \boldsymbol{z}_T$.

Dataset	PixelRNN [46]	Real NVP	Conv DRAW [22]	IAF-VAE [34]
CIFAR-10	3.00	3.49	< 3.59	< 3.28
Imagenet (32×32)	3.86 (3.83)	4.28 (4.26)	< 4.40 (4.35)	
Imagenet (64×64)	3.63 (3.57)	3.98 (3.75)	< 4.10 (4.04)	
LSUN (bedroom)		2.72 (2.70)		
LSUN (tower)		2.81 (2.78)		
LSUN (church outdoor)		3.08 (2.94)		
CelebA		3.02 (2.97)		

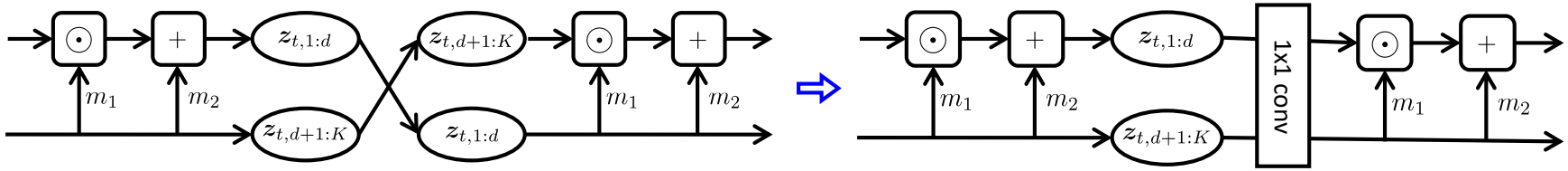
- Synthetic image generation results:

Done by backward pass of functions $\boldsymbol{z}_T \rightarrow \boldsymbol{z}_0 = \boldsymbol{x}$.

- If one uses fully factorized prior distribution $p_T(\boldsymbol{z}_T)$, then flow based models can be interpreted as non-linear independent components estimation (NICE) [Dinh et al., 2015].



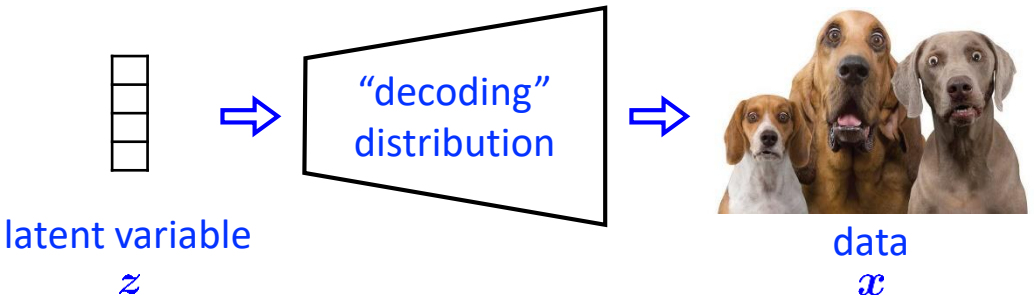
- Alternative pairing of coupling layers have been generalized to linear transformations [Kingma et al., 2018].



- More advanced flows specified for density estimation (not sampling) have been proposed by using autoregressive transformations [Papamakarios et al., 2017].

Table of Contents

- Consider the following generative model:



  - Fixed prior on random latent variable.
    - e.g., standard Normal distribution

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbb{I})$$

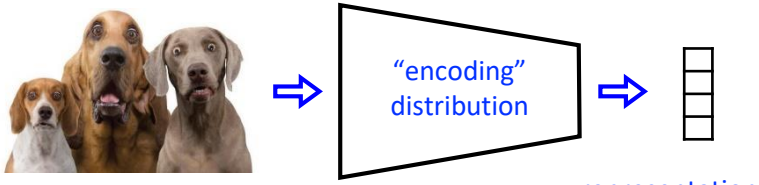
  - Parameterized likelihood (decoder) for generation:
    - e.g., Normal distribution parameterized by neural network

$$p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; f_{\mathsf{dec}}(\boldsymbol{z}), \mathbb{I})$$

  - Resulting generative distribution (to optimize):

$$\log p_\theta(\boldsymbol{x}) = \log \int_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} = \log \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[p(\boldsymbol{x}|\boldsymbol{z})]$$

- Variational autoencoder (VAE) introduce an auxiliary distribution (encoder) [Kingma et al., 2013].



$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; f_{\texttt{enc},\mu}(\boldsymbol{x}), f_{\texttt{enc},\sigma}(\boldsymbol{x}))$$

- Each $\log p_\theta(\boldsymbol{x})$ term is replaced by its lower bound:

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &\geq \log p_\theta(\boldsymbol{x}) - \min_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) \\
&= \log p_\theta(\boldsymbol{x}) + \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{z}|\boldsymbol{x}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\
&= \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}) + \log p_\theta(\boldsymbol{z}|\boldsymbol{x}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\
&= \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))
\end{aligned}$$

- Bound becomes equality when $q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$.

- The training objective becomes:

tractable between two Gaussian distributions

$$\max_\theta \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}^{(n)}) \geq \max_\theta \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))$$

$$\approx \max_\theta \max_\phi \sum_{n=1}^{N}\sum_{k=1}^{N} \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))$$

where latent variables are sampled by $\boldsymbol{z}^{(n,k)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})$.

- However, non-trivial to train with back propagation due to sampling procedure:

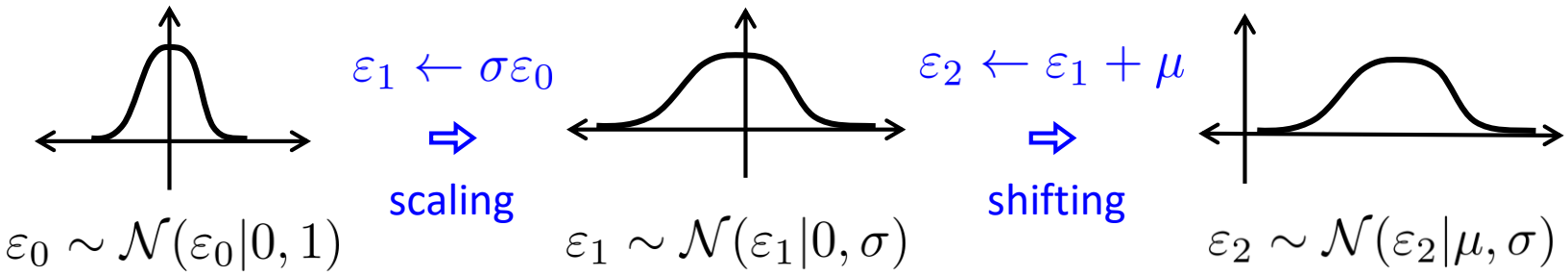
$$\nabla_\phi \mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{N} - \nabla_\phi \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) + \nabla_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))$$

⇩

Since $\boldsymbol{z}^{(n,k)}$ is fixed after being sampled, $\nabla_\phi \log p(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) = 0$?

- Reparameterization trick is based on the change-of-variables formula:

$$\varepsilon_2 \sim \mathcal{N}(\varepsilon_2|\mu,\sigma) \;\Leftrightarrow\; \varepsilon_2 = \mu + \sigma\varepsilon_0, \qquad \varepsilon_0 \sim \mathcal{N}(\varepsilon_0|0,1)$$



- Latent variable $\boldsymbol{z}^{(n,k)}$ can be similarly parameterized by encoder network:

$$\boldsymbol{z}^{(n,k)} \sim \mathcal{N}(\boldsymbol{z}; f_{\text{enc},\mu}(\boldsymbol{x}^{(n)}), f_{\text{enc},\sigma}(\boldsymbol{x}^{(n)}))$$

$$\Updownarrow$$

$$\boldsymbol{z}^{(n,k)} = f_{\text{enc},\mu}(\boldsymbol{x}^{(n)}) + f_{\text{enc},\sigma}(\boldsymbol{x}^{(n)}) \odot \boldsymbol{\varepsilon}^{(n,k)}, \qquad \boldsymbol{\varepsilon}^{(n,k)} \sim \mathcal{N}(\boldsymbol{\varepsilon}|\mathbf{0},\mathbf{1})$$

- Total loss of variational autoencoder:

$$\nabla_\phi \mathcal{L} = \sum_{n=1}^{N} \sum_{k=1}^{N} - \underbrace{\nabla_\phi \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)})}_{\nabla_\phi \mathcal{L}_1} + \underbrace{\nabla_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))}_{\nabla_\phi \mathcal{L}_2}$$

- Recall that $f_{\mathtt{dec}}, f_{\mathtt{enc},\mu}, f_{\mathtt{enc},\sigma}$ are parameterized by ϕ .

- Derivative of first part:

$$\nabla_\phi \mathcal{L}_1 = \nabla_\phi \log \mathcal{N}(\boldsymbol{x}^{(n)}; f_{\mathtt{dec}}(\boldsymbol{z}^{(n,k)}), \mathbf{1})$$

- Total loss of variational autoencoder:

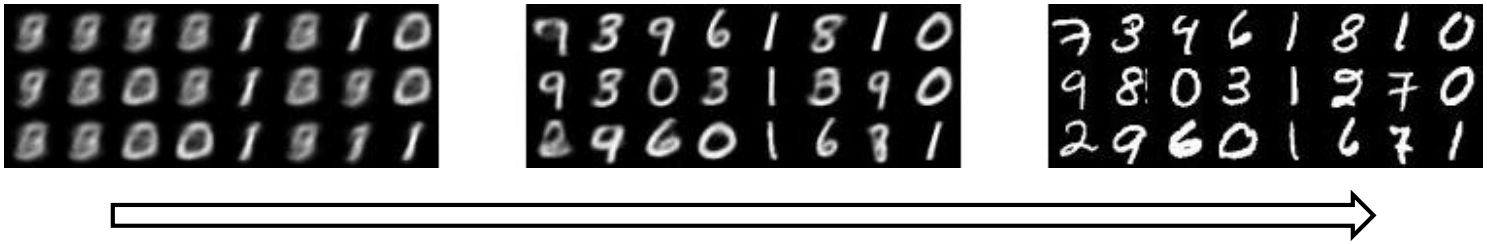
$$\nabla_\phi \mathcal{L} = \sum_{n=1}^{N} \sum_{k=1}^{N} - \underbrace{\nabla_\phi \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)})}_{\nabla_\phi \mathcal{L}_1} + \underbrace{\nabla_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))}_{\nabla_\phi \mathcal{L}_2}$$

- Recall that $f_{\mathtt{dec}}, f_{\mathtt{enc},\mu}, f_{\mathtt{enc},\sigma}$ are parameterized by ϕ .

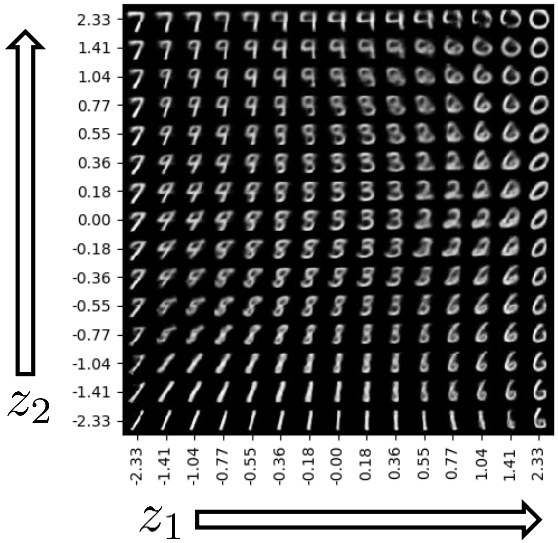
- Derivative of second part:

$$\nabla_\phi \mathcal{L}_1 = \nabla_\phi \mathrm{KL}(\mathcal{N}(\boldsymbol{z}; f_{\mathtt{enc},\mu}(\boldsymbol{x}^{(n)}), f_{\mathtt{enc},\sigma}(\boldsymbol{x}^{(n)}))||\mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{1}))$$

- Based on the proposed scheme, variational autoencoder successfully generates images:



- Interpolation of latent variables induce transitions in generated images:



Towards Improving Variational Inference

- Recall: evidence lower bound (ELBO, also variational lower bound).

$$\log p_\theta(\boldsymbol{x}) \geq \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))$$

- Approximating the log-likelihood is also called variational inference.
  - Considered to be a important topic in theory.
  - However, gap in bound does not decrease no matter how many samples we use.

- Two ways of improving variational inference:
  - Replace ELBO with a tighter bound: importance weighted samples
  - Use more powerful parameterization of q_ϕ: inverse auto-regressive flow

- Observe that ELBO can also be proved by the Jensen's inequality:

$$\log p(\boldsymbol{x}) = \log \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] \geq \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]$$

  - Based on convexity, interchange order of logarithm and summation.

- Importance weighted AE (IWAE) relax the inequality [Burda et al., 2018]:

$$\log p(\boldsymbol{x}) = \log \mathbb{E}_{\boldsymbol{z}^{(1)},\cdots,\boldsymbol{z}^{(K)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\sum_{k=1}^{K}\frac{p(\boldsymbol{x},\boldsymbol{z}^{(k)})}{q_\phi(\boldsymbol{z}^{(k)}|\boldsymbol{x})}\right]$$

$$\geq \mathbb{E}_{\boldsymbol{z}^{(1)},\cdots,\boldsymbol{z}^{(K)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{1}{K}\sum_{k=1}^{K}\boxed{\frac{p(\boldsymbol{x},\boldsymbol{z}^{(k)})}{q_\phi(\boldsymbol{z}^{(k)}|\boldsymbol{x})}}\right]$$

also called importance weights

  - Becomes original ELBO when $K = 1$ and becomes exact bound when $K = \infty$.

$$\downarrow$$

$$\mathbb{E}_{\boldsymbol{z}^{(1)},\cdots,\boldsymbol{z}^{(K)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{p(\boldsymbol{x},\boldsymbol{z}^{(k)})}{q_\phi(\boldsymbol{z}^{(k)}|\boldsymbol{x})}\right] \approx p(\boldsymbol{x})$$

- IWAE improves the ELBO, i.e., lower negative log-likelihood (NLL).

- It also alleviates the so-called posterior collapse problem: hidden units become inactive (=invariant for all input).

	MNIST				OMNIGLOT			
	VAE		IWAE		VAE		IWAE	
k	NLL	active units	NLL	active units	NLL	active units	NLL	active units
1	86.76	19	86.76	19	108.11	28	108.11	28
5	86.47	20	85.54	22	107.62	28	106.12	34
50	86.35	20	84.78	25	107.80	28	104.67	41

best performance

Improving Variational Inference with Inverse Auto-Regressive Flow

- Fully factorized Gaussian $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ often leads to poor approximation of the true distribution $p_\theta(\boldsymbol{z}|\boldsymbol{x})$.

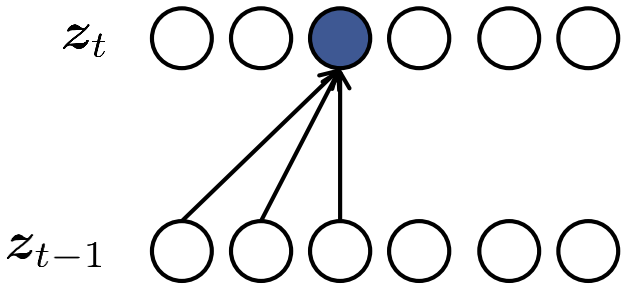
- This can be improved by introducing flow-based distribution as the flexible parameterization of $q_\phi(\boldsymbol{z}|\boldsymbol{x})$.
  - Recall: functions are invertible in order to allow tractable density estimation.

$$\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}) \quad \Longleftrightarrow \quad \begin{aligned} \boldsymbol{z}_0 &\sim \mathcal{N}(\boldsymbol{z}_0; \boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\sigma}_\phi(\boldsymbol{x})), \\ \boldsymbol{z} &= f_T \circ f_{T-1} \circ \cdots, \circ f_1(\boldsymbol{z}_0) \end{aligned}$$

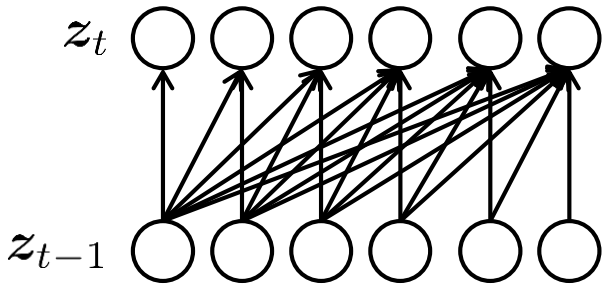
- Inverse autoregressive flow (IAF) for improving variational auto-encoder was proposed [Kingma et al., 2016] (details in the next slide).
  - IAF is specialized for variational inference since its forward pass $\boldsymbol{z}_0 \rightarrow \boldsymbol{z}_T$ is fast, but its backward pass $\boldsymbol{z}_T \rightarrow \boldsymbol{z}_0$ is slow (not needed for variational inference).

- Inverse autoregressive flow (IAF) modifies each dimension of variable in auto-regressive manner [Kingma et al., 2016]:
  - For each $d = 1, \cdots, K$:

$$z_{t,d} = \mu_{t,d}(\boldsymbol{z}_{t-1,1:d-1}) + \sigma_{t,d}(\boldsymbol{z}_{t-1,1:d-1}) z_{t-1,d}$$



case of $d = 3$



updates done in parallel

  - Inference for corresponding normalizing flow is efficient:

$$\log q(\boldsymbol{z}|\boldsymbol{x}) = \log q_0(\boldsymbol{z}_0|\boldsymbol{x}) + \sum_{t=1}^{T} \log \left| \det \left(\frac{\partial f_t(\boldsymbol{z}_{t-1})}{\partial \boldsymbol{z}_{t-1}} \right) \right|$$

$$\begin{bmatrix} \sigma_{t,1} & 0 & \cdots & 0 \\ & \sigma_{t,2} & 0 & \vdots \\ & & \ddots & 0 \\ & & & \sigma_{t,K} \end{bmatrix}$$

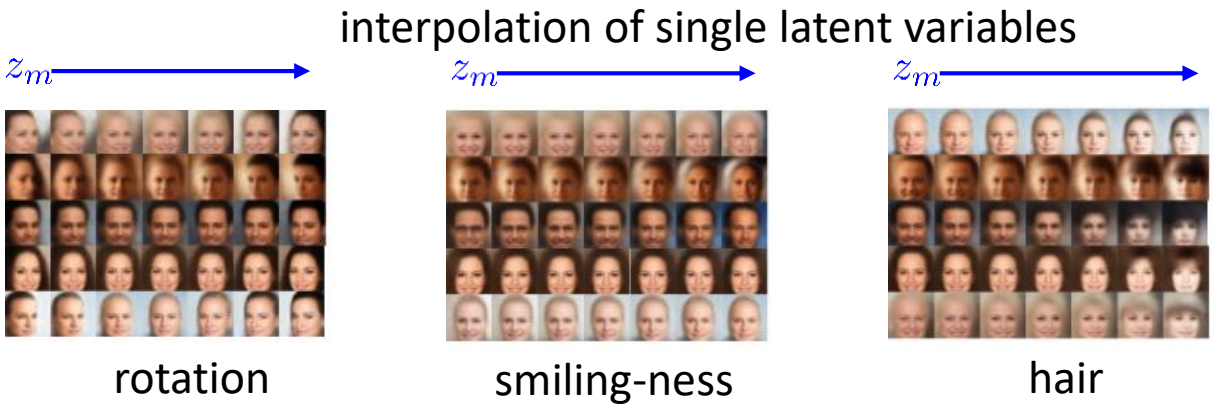
- Variational lower bound and estimated log likelihood for test dataset (MNIST):

Model	VLB	$\log p(\mathbf{x}) \approx$
Convolutional VAE + HVI [1]	-83.49	-81.94
DLGM 2hl + IWAE [2]		-82.90
LVAE [3]		-81.74
DRAW + VGP [4]	-79.88	
Diagonal covariance	-84.08 ($\pm$ 0.10)	-81.08 ($\pm$ 0.08)
IAF (Depth = 2, Width = 320)	-82.02 ($\pm$ 0.08)	-79.77 ($\pm$ 0.06)
IAF (Depth = 2, Width = 1920)	-81.17 ($\pm$ 0.08)	-79.30 ($\pm$ 0.08)
IAF (Depth = 4, Width = 1920)	-80.93 ($\pm$ 0.09)	-79.17 ($\pm$ 0.08)
IAF (Depth = 8, Width = 1920)	-80.80 ($\pm$ 0.07)	**-79.10** ($\pm$ 0.07)

- Average negative log likelihood for test dataset (CIFAR-10):

Method	bits/dim $\leq$
Results with tractable likelihood models:	
Uniform distribution (van den Oord et al., 2016b)	8.00
Multivariate Gaussian (van den Oord et al., 2016b)	4.70
NICE (Dinh et al., 2014)	4.48
Deep GMMs (van den Oord and Schrauwen, 2014)	4.00
Real NVP (Dinh et al., 2016)	3.49
PixelRNN (van den Oord et al., 2016b)	**3.00**
Gated PixelCNN (van den Oord et al., 2016c)	**3.03**
Results with variationally trained latent-variable models:	
Deep Diffusion (Sohl-Dickstein et al., 2015)	5.40
Convolutional DRAW (Gregor et al., 2016)	3.58
ResNet VAE with IAF (Ours)	**3.11**

- Overcoming the posterior collapse problem (hidden units becoming inactive) is an active topic of research [He et al., 2019].
- Latent variables can be disentangled, i.e., trained to be sensitive to only a single generative factor of dataset [Higgins et al., 2017]:



- Recent variational autoencoders were successfully applied to generating high-fidelity images [Razavi et al., 2019] :

References

[Kingma et al., 2013] Auto-Encoding Variational Bayes, ICLR 2013
link: https://arxiv.org/abs/1802.06455

[Dinh et al., 2015] NICE: Non-Linear Independent Components Estimation, ICLR 2015
link: https://arxiv.org/abs/1410.8516

[Rezende et al., 2015] Variational Inference with Normalizing Flows, ICML 2015
link: https://arxiv.org/abs/1705.08665

[Oord et al., 2016] Pixel Recurrent Neural Networks, ICML 2016
link: https://arxiv.org/pdf/1601.06759

[Burda et al., 2016] Importance Weighted Autoencoders, ICLR 2016
link: https://arxiv.org/abs/1509.00519

[Kingma et al., 2016] Improving Inference with Inverse Autoregressive Flows, NIPS 2016
link: https://arxiv.org/abs/1710.10628

[Oord et al., 2017] WaveNet: A Generative Model for Raw Audio, SSW 2017
link: https://arxiv.org/abs/1703.01961

[Dinh et al., 2017] Density Estimation using Real NVP, ICLR 2017
link: https://arxiv.org/abs/1605.08803

[Higgins et al., 2017] beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, ICLR 2017
link: https://openreview.net/pdf?id=Sy2fzU9gl

[Papamakarios et al., 2017] Masked Autoregressive Flow for Density Estimation, NIPS 2017
link: https://arxiv.org/abs/1710.10628

[Kingma et al., 2018] Generative Flow with Invertible 1x1 Convolutions, NIPS 2018
link: https://arxiv.org/abs/1807.03039

[He et al., 2019] Lagging Inference Networks and Posterior Collapse in Variational Autoencoders, ICLR 2019
link: https://openreview.net/forum?id=rylDfnCqF7