Deep Learning for Computer Vision

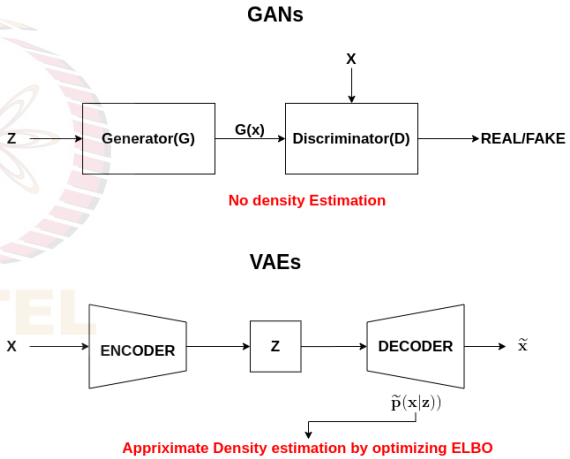
Beyond VAEs and GANs: Other Methods for Deep Generative Models

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

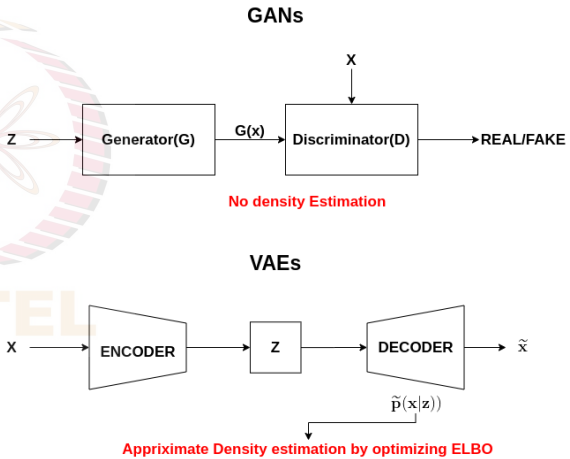
Flow-based Models

- Both GANs and VAEs do not explicitly learn probability density function of real data $p(x)$



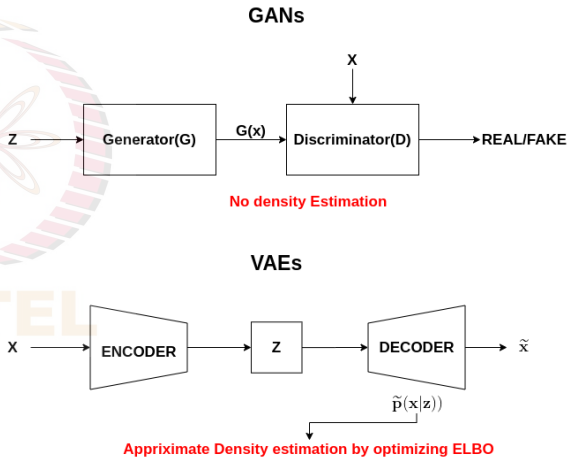
Flow-based Models

- Both GANs and VAEs do not explicitly learn probability density function of real data $p(x)$
- $p(x)$ may be useful for downstream tasks like filling incomplete data, sampling data and also identifying bias in data distributions



Flow-based Models

- Both GANs and VAEs do not explicitly learn probability density function of real data $p(x)$
- $p(x)$ may be useful for downstream tasks like filling incomplete data, sampling data and also identifying bias in data distributions
- **Flow-based models** explicitly learn $p(x)$ by optimizing the log likelihood
  - **Normalizing flows**
  - **Autoregressive flows**



Recall: Generative Models - how to learn?

Aim: To minimize some notion of distance between $p_D(x)$ and $p_M(x)$; how?

- Given a dataset $X = x_1, x_2, x_3, ..., x_N$ from an underlying distribution $p_D(x)$
- Consider an approximating distribution $p_M(x)$ coming from a family of distributions M, i.e. we need to find the best distribution in M, parametrized by θ, which minimizes distance between p_M and p_D, i.e.:

$$\theta^* = \arg\min_{\theta \in M} \text{dist}(p_\theta, p_D)$$

- If KL-divergence is the distance function, the above problem becomes one of maximum likelihood estimation!

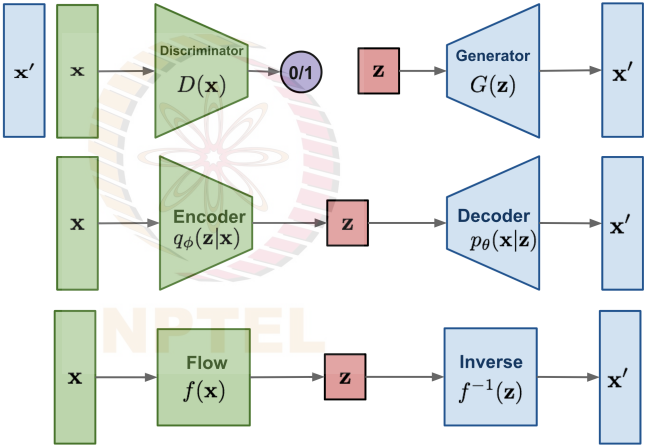
$$\theta^* = \arg\min_{\theta \in M} \mathbb{E}_{x \sim p_D}[-\log\ p_\theta(x)]$$

Flow-based Models

GAN: minimax the classification error loss.

VAE: maximize ELBO.

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

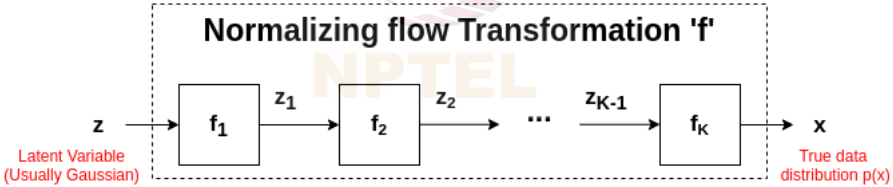


Credit: *Lilian Weng*

Normalizing Flows

- Identifies a transformation $f : \boldsymbol{Z} \rightarrow \boldsymbol{X}$ where f is a series of **differentiable bijective functions** $(f_1, f_2, ..., f_K)$.

$$\boldsymbol{x} = f(\boldsymbol{z}) = f_K \circ f_{K-1} \circ ... \circ f_2 \circ f_1(\boldsymbol{z}) \text{and}$$
$$\boldsymbol{z} = f^{-1}(\boldsymbol{x}) = f_1^{-1} \circ ..... \circ f_K^{-1}(\boldsymbol{x})$$



Normalizing Flows

- For any invertible function $f : Z \to X$, using **change of variables** for probability density functions:

$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right| \quad \text{Why?}$$

Normalizing Flows

- For any invertible function $f : Z \to X$, using **change of variables** for probability density functions:

$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right| \quad \text{Why?}$$

- If $z \sim \pi(z)$ is a random variable and $x = f(z)$ such that f is invertible (i.e. $z = f^{-1}(x)$); then:

$$\int p(x)dx = \int \pi(z)dz = 1 \text{ ; Definition of probability distribution}$$

$$p(x) = \pi(z) \left| \frac{dz}{dx} \right| = \pi(f^{-1}(x)) \left| \frac{df^{-1}}{dx} \right| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$

Normalizing Flows

- For any invertible function $f : Z \to X$, using **change of variables** for probability density functions:

$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right| \quad \text{Why?}$$

- If $z \sim \pi(z)$ is a random variable and $x = f(z)$ such that f is invertible (i.e. $z = f^{-1}(x)$); then:

$$\int p(x)dx = \int \pi(z)dz = 1 \text{ ; Definition of probability distribution}$$

$$p(x) = \pi(z) \left| \frac{dz}{dx} \right| = \pi(f^{-1}(x)) \left| \frac{df^{-1}}{dx} \right| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$

- In multiple variables: $\mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x} = f(\mathbf{z}), \mathbf{z} = f^{-1}(\mathbf{x})$

$$p(\mathbf{x}) = \pi(\mathbf{z}) \left| \det \frac{d\mathbf{z}}{d\mathbf{x}} \right| = \pi(f^{-1}(\mathbf{x})) \left| \det \frac{df^{-1}}{d\mathbf{x}} \right|$$

Credit: Lilian Weng

Normalizing Flows

- For any invertible function $f : Z \to X$, using **change of variables** for probability density functions:

$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

- Expanding this to a composite function and applying log on both sides, we get the likelihood objective:

$$\log p_X(\boldsymbol{x}) = \log p_Z(\boldsymbol{z}) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial f_i^{-1}}{\partial \boldsymbol{z_i}} \right) \right|$$

Normalizing Flows

- For any invertible function $f : Z \to X$, using **change of variables** for probability density functions:

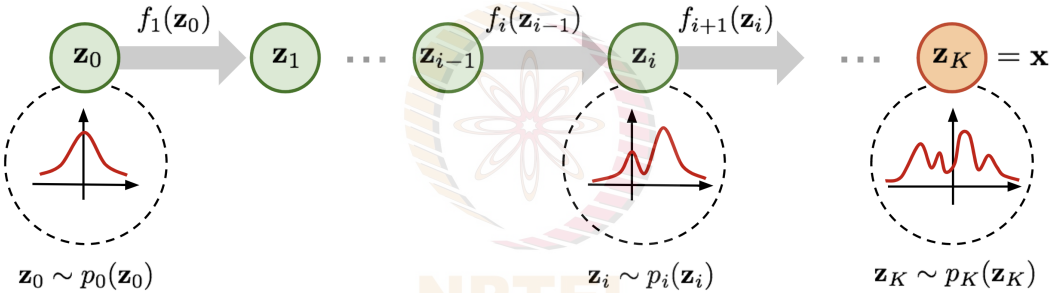
$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

- Expanding this to a composite function and applying log on both sides, we get the likelihood objective:

$$\log p_X(\boldsymbol{x}) = \log p_Z(\boldsymbol{z}) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial f_i^{-1}}{\partial \boldsymbol{z_i}} \right) \right|$$

- **Intuition:**
  - Transformation f moulds the density $p_Z(z)$ into $p_X(x)$
  - $\left| \det \left(\frac{\partial f_i^{-1}}{\partial \boldsymbol{z_i}} \right) \right|$ quantifies the relative change of volume of a small neighborhood dz around z

Normalizing Flows: Illustration



Credit: *Lilian Weng*

Normalizing Flows

Why bijective function?

- Such a bijective function is called a **diffeomorphism**
- Differomorphic functions are **composable**: given two such transformations f_1 and f_2, their composition is also invertible and differentiable
- Complex transformations can be modeled by composing multiple instances of simpler transformations

Normalizing Flows

Why bijective function?

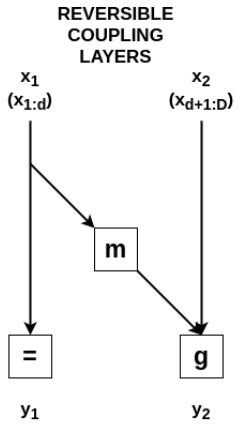
- Such a bijective function is called a **diffeomorphism**
- Differomorphic functions are **composable**: given two such transformations f_1 and f_2, their composition is also invertible and differentiable
- Complex transformations can be modeled by composing multiple instances of simpler transformations

Prerequisites for normalizing flows

- Transformation function f should be differentiable (neural networks are)
- Function should be easily invertible
- Determinant of Jacobian should be easy to compute

NICE (Non-linear Independent Components Estimation)[1]

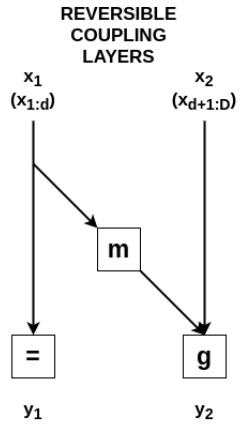
What transformation to use?



[1] Dinh et al, NICE: Non-linear Independent Components Estimation, ICLRW 2015

NICE (Non-linear Independent Components Estimation)[1]

What transformation to use?



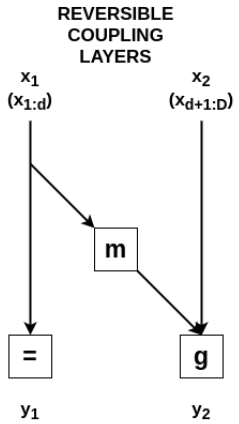
- Coupling layer operation:

$$y_1 = x_1$$
$$y_2 = g(x_2; m(x_1))$$

[1] Dinh et al, NICE: Non-linear Independent Components Estimation, ICLRW 2015

NICE (Non-linear Independent Components Estimation)[1]

What transformation to use?



- Coupling layer operation:

$$y_1 = x_1$$
$$y_2 = g(x_2; m(x_1))$$

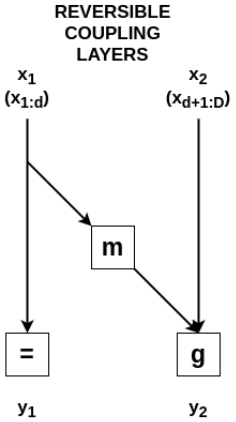
- Jacobian is a lower-triangular matrix (why?); Determinant is product of diagonal elements

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix}$$

[1] Dinh et al, NICE: Non-linear Independent Components Estimation, ICLRW 2015

NICE (Non-linear Independent Components Estimation)[1]

What transformation to use?



- Coupling layer operation:

$$y_1 = x_1$$
$$y_2 = g(x_2; m(x_1))$$

- Jacobian is a lower-triangular matrix (why?); Determinant is product of diagonal elements

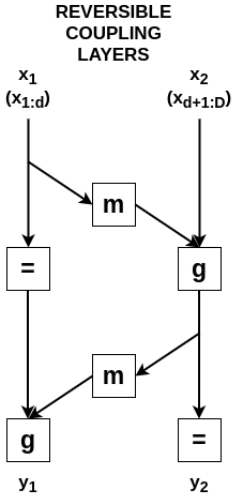
$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix}$$

- Inverse mappings are:

$$x_1 = y_1$$
$$x_2 = g^{-1}(y_2; m(y_1))$$

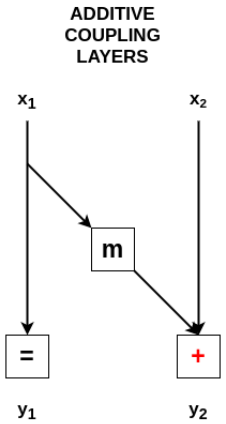
[1] Dinh et al, NICE: Non-linear Independent Components Estimation, ICLRW 2015

NICE (Non-linear Independent Components Estimation)



If all data is to incorporated, flip inputs after each layer

NICE: Additive Coupling Layer



- Additive coupling layer operations:

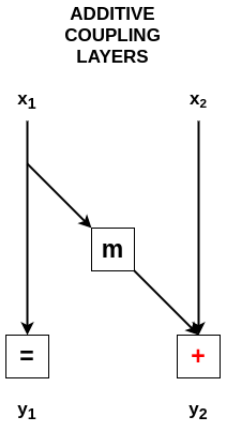
$$y_1 = x_1$$
$$y_2 = x_2 + m(x_1)$$

- Inverse operation is:

$$x_1 = y_1$$
$$x_2 = y_2 - m(y_1)$$

- Jacobian determinant is always 1, hence, volume-preserving (Why?)

NICE: Additive Coupling Layer



- Additive coupling layer operations:

$$y_1 = x_1$$
$$y_2 = x_2 + m(x_1)$$

- Inverse operation is:

$$x_1 = y_1$$
$$x_2 = y_2 - m(y_1)$$

- Jacobian determinant is always 1, hence, volume-preserving (Why?) Transformed distribution p_X will have same "volume" compared to original one p_Z
- Log-likelihood now translates to:

$$\log p_X(\boldsymbol{x}) = \log p_Y(\boldsymbol{y})$$

Real NVP (Real-valued Non Volume Preserving)[2]

- Affine Coupling operations:

$$y_1 = x_1$$
$$y_2 = x_2 \odot \exp(s(x_1)) + t(x_1)$$

- Jacobian of the transformation is:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_2}{\partial x_1} & diag(\exp[s(x1)]) \end{bmatrix}$$

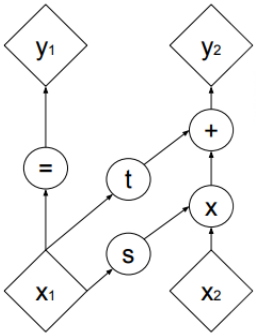
  Since Jacobian not always 1, affine coupling is not volume preserving which is the case for real-time data

- Inverse operation:

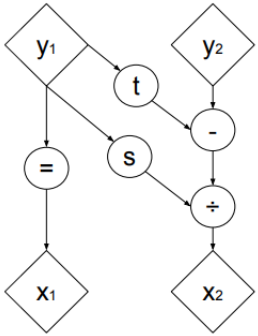
$$x_1 = y_1$$
$$x_2 = (y_2 - t(y_1)) \odot \exp(-s(y_1))$$

[2]Dinh et al, Density estimation using Real NVP, ICLR 2017

Real NVP: Affine Coupling



(a) Forward propagation



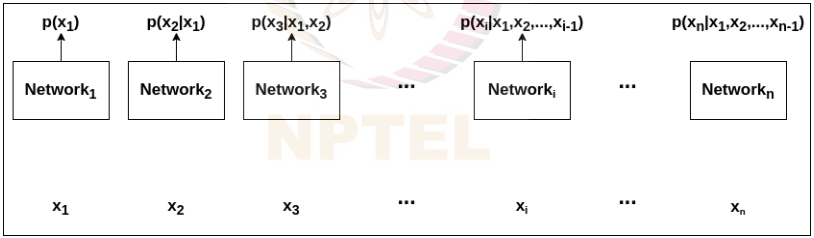
(b) Inverse propagation

Autoregressive Flows

- Decompose the task of learning the joint distribution into learning a series of conditional probabilities

$$p_X(\boldsymbol{x}) = p(x_1, x_2, ..., x_n) = p(x_1).p(x_2|x_1).p(x_3|x_2, x_1)....p(x_n|x_1, x_2, ..., x_{n-1})$$

- While calculating a conditional probability (at level i), model can only see inputs occurring prior to it $(1, 2, .., i-1)$

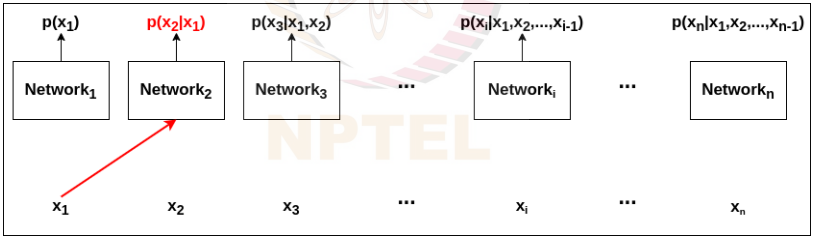


Autoregressive Flows

- Decompose the task of learning the joint distribution into learning a series of conditional probabilities

$$p_X(\boldsymbol{x}) = p(x_1, x_2, ..., x_n) = p(x_1).p(x_2|x_1).p(x_3|x_2, x_1)....p(x_n|x_1, x_2, ..., x_{n-1})$$

- While calculating a conditional probability (at level i), model can only see inputs occurring prior to it $(1, 2, .., i-1)$

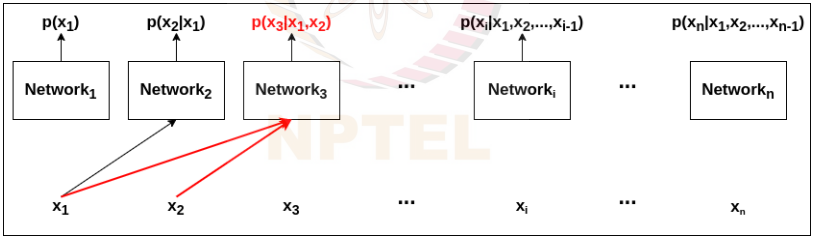


Autoregressive Flows

- Decompose the task of learning the joint distribution into learning a series of conditional probabilities

$$p_X(\boldsymbol{x}) = p(x_1, x_2, ..., x_n) = p(x_1).p(x_2|x_1).p(x_3|x_2, x_1)....p(x_n|x_1, x_2, ..., x_{n-1})$$

- While calculating a conditional probability (at level i), model can only see inputs occurring prior to it $(1, 2, .., i-1)$

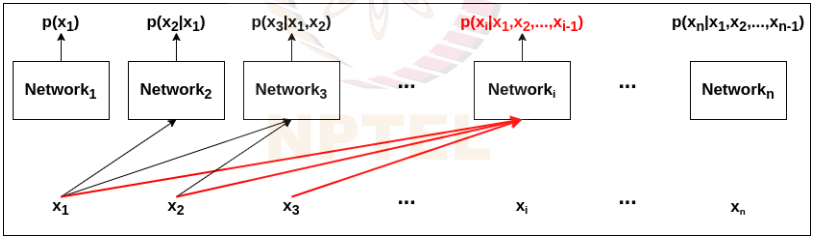


Autoregressive Flows

- Decompose the task of learning the joint distribution into learning a series of conditional probabilities

$$p_X(\boldsymbol{x}) = p(x_1, x_2, ..., x_n) = p(x_1).p(x_2|x_1).p(x_3|x_2, x_1)....p(x_n|x_1, x_2, ..., x_{n-1})$$

- While calculating a conditional probability (at level i), model can only see inputs occurring prior to it $(1, 2, .., i-1)$

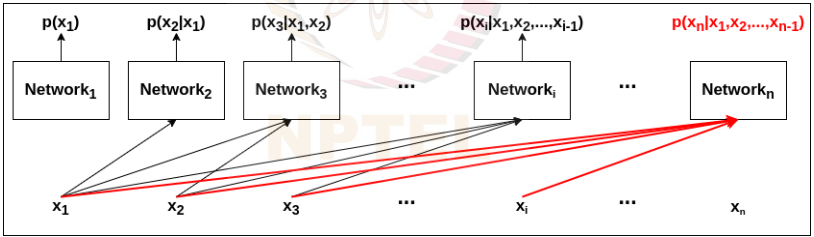


Autoregressive Flows

- Decompose the task of learning the joint distribution into learning a series of conditional probabilities

$$p_X(\boldsymbol{x}) = p(x_1, x_2, ..., x_n) = p(x_1).p(x_2|x_1).p(x_3|x_2, x_1)....p(x_n|x_1, x_2, ..., x_{n-1})$$

- While calculating a conditional probability (at level i), model can only see inputs occurring prior to it $(1, 2, .., i-1)$

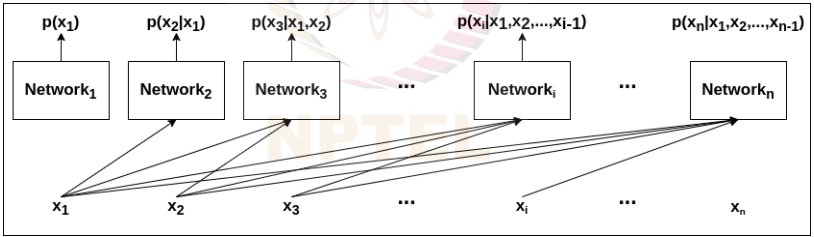


Autoregressive Flows

- Decompose the task of learning the joint distribution into learning a series of conditional probabilities

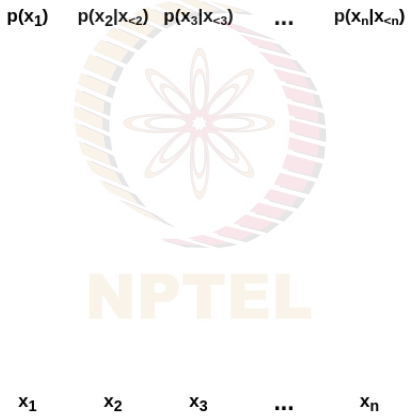
$$p_X(\boldsymbol{x}) = p(x_1, x_2, ..., x_n) = p(x_1).p(x_2|x_1).p(x_3|x_2, x_1)....p(x_n|x_1, x_2, ..., x_{n-1})$$

- While calculating a conditional probability (at level i), model can only see inputs occurring prior to it $(1, 2, .., i-1)$



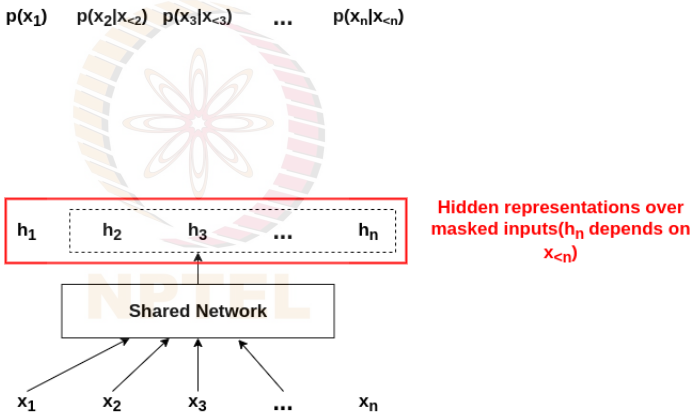
- Can be considered as a special case of Normalizing Flows where each intermediate transformation masks relevant inputs

Autoregressive Models: NADE (Neural Autoregressive Distribution Estimation)[3]



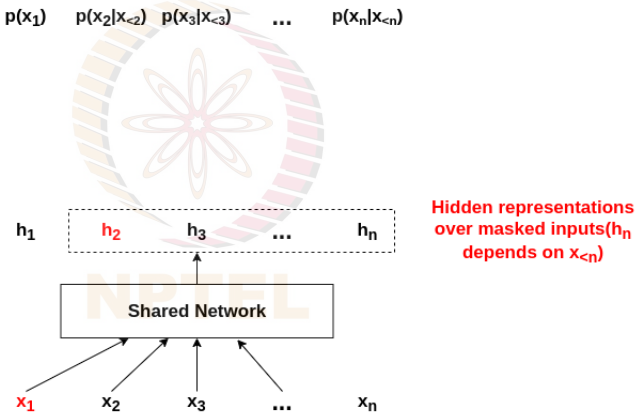
[3] Larochelle and Murray, The Neural Autoregressive Distribution Estimator, ICML 2011; Uria et al, Neural Autoregressive Distribution Estimation, JMLR 2016

Autoregressive Models: NADE (Neural Autoregressive Distribution Estimation)[3]



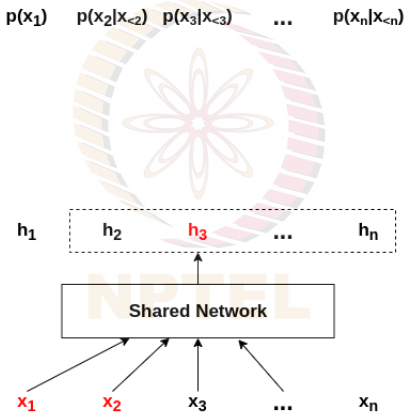
[3] Larochelle and Murray, The Neural Autoregressive Distribution Estimator, ICML 2011; Uria et al, Neural Autoregressive Distribution Estimation, JMLR 2016

Autoregressive Models: NADE (Neural Autoregressive Distribution Estimation)[3]



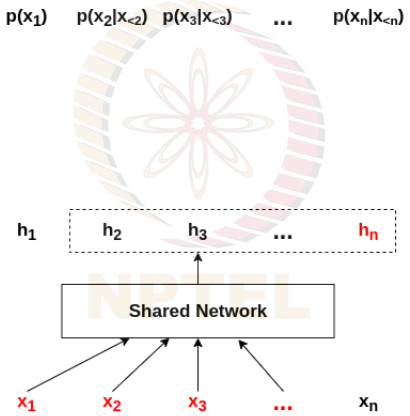
[3]Larochelle and Murray, The Neural Autoregressive Distribution Estimator, ICML 2011; Uria et al, Neural Autoregressive Distribution Estimation, JMLR 2016

Autoregressive Models: NADE (Neural Autoregressive Distribution Estimation)[3]



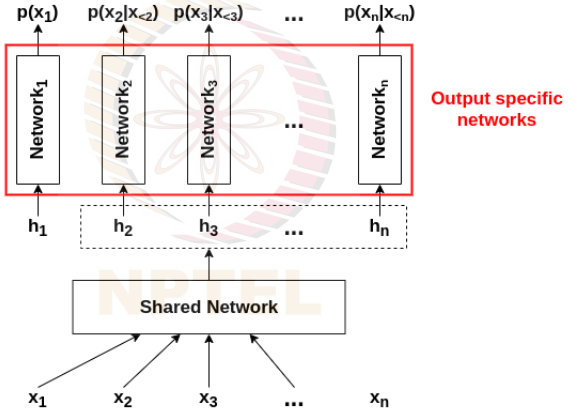
[3]Larochelle and Murray, The Neural Autoregressive Distribution Estimator, ICML 2011; Uria et al, Neural Autoregressive Distribution Estimation, JMLR 2016

Autoregressive Models: NADE (Neural Autoregressive Distribution Estimation)[3]



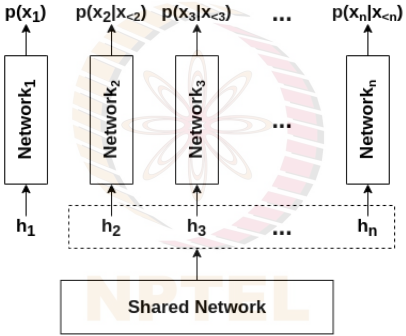
[3] Larochelle and Murray, The Neural Autoregressive Distribution Estimator, ICML 2011; Uria et al, Neural Autoregressive Distribution Estimation, JMLR 2016

Autoregressive Models: NADE (Neural Autoregressive Distribution Estimation)[3]

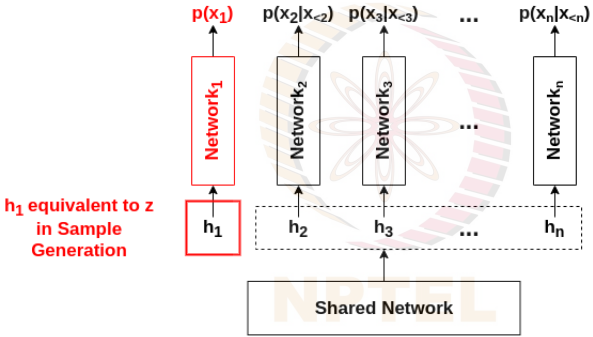


[3] Larochelle and Murray, The Neural Autoregressive Distribution Estimator, ICML 2011; Uria et al, Neural Autoregressive Distribution Estimation, JMLR 2016

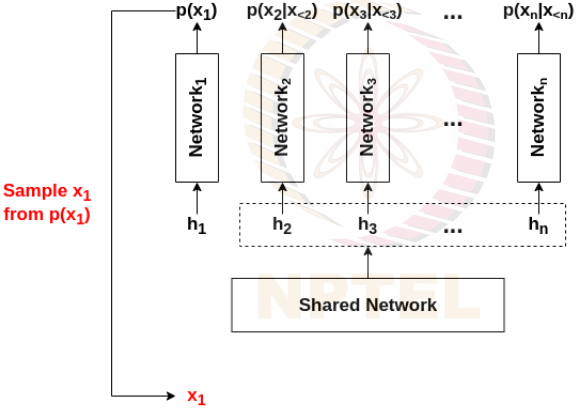
NADE: Sample Generation



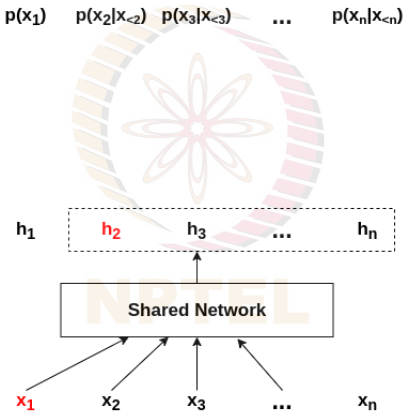
NADE: Sample Generation



NADE: Sample Generation

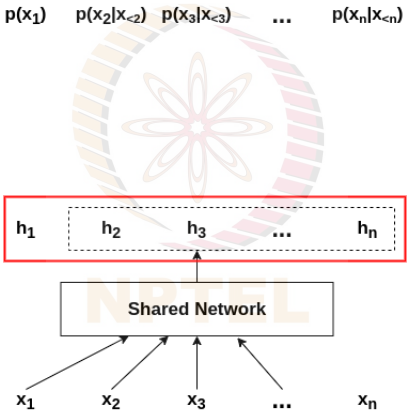


NADE: Sample Generation

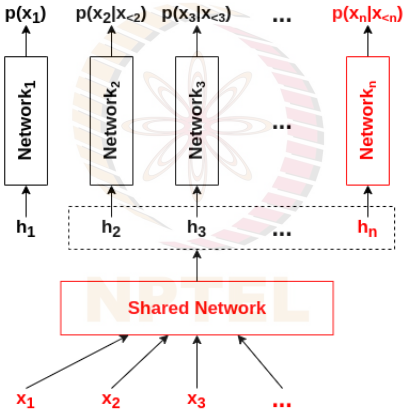


Hidden representations over masked inputs(h_n depends on $x_{<n}$)

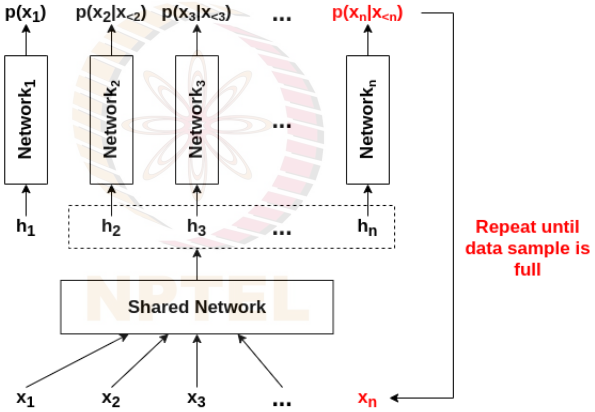
NADE: Sample Generation



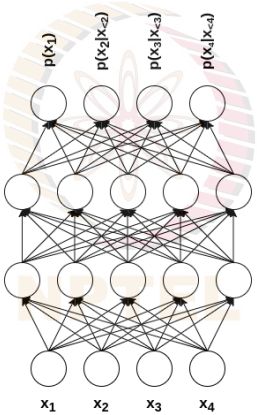
NADE: Sample Generation



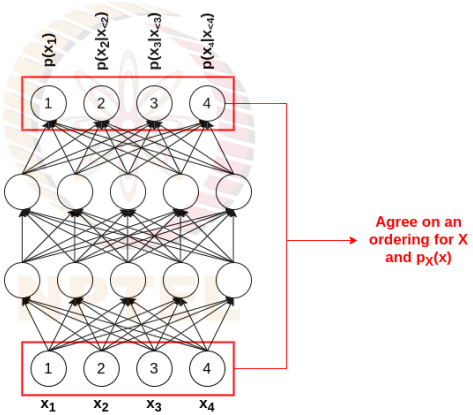
NADE: Sample Generation



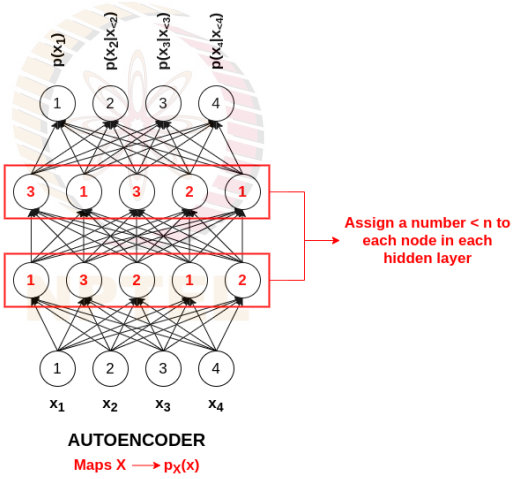
Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



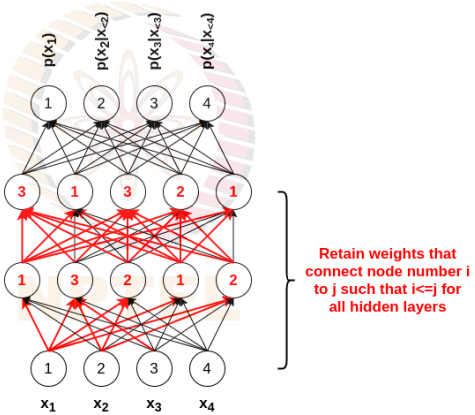
Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



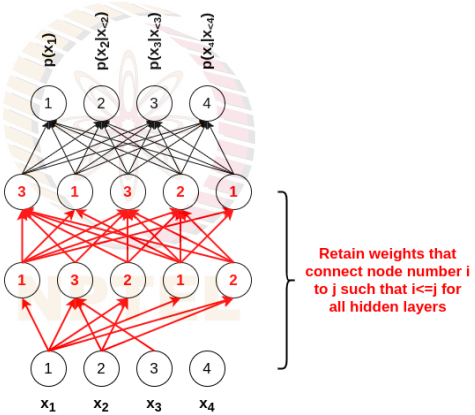
Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



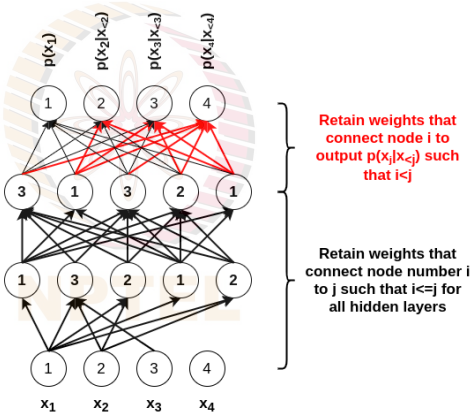
Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



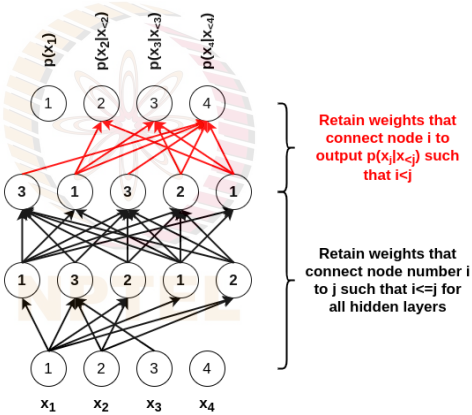
Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



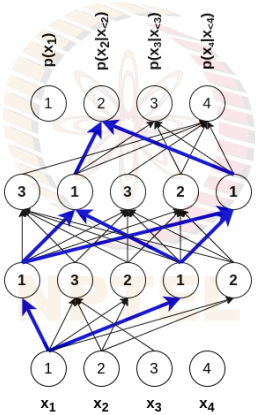
Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]

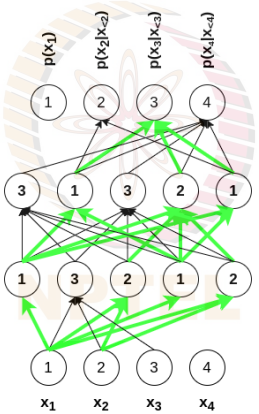


Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



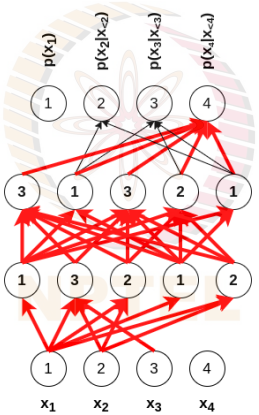
Ensures that $p(x_i|x_{<i})$ subnetwork depends only on inputs $x_{<i}$

Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]



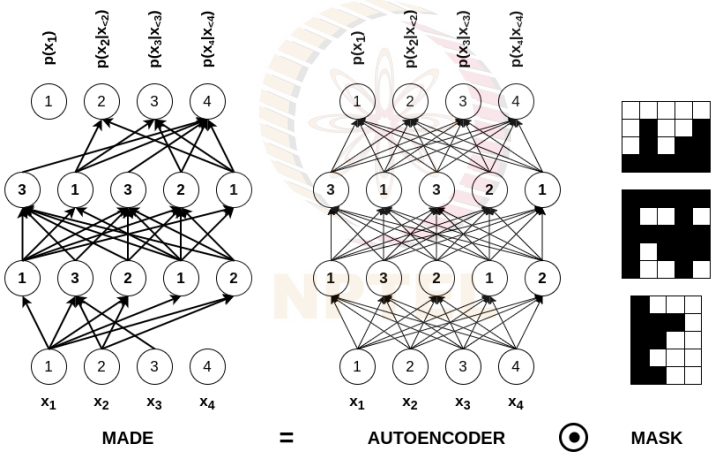
Ensures that $p(x_i|x_{<i})$ subnetwork depends only on inputs $x_{<i}$

Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]

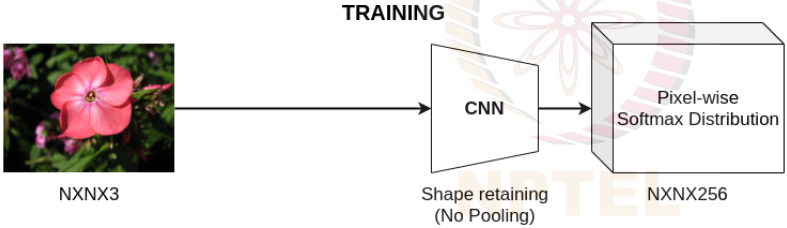


Ensures that $p(x_i|x_{<i})$ subnetwork depends only on inputs $x_{<i}$

Autoregressive Models: MADE (Masked Autoencoder for Distribution Estimation)[4]

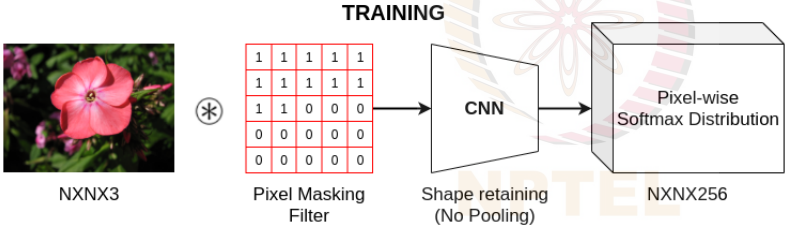


Autoregressive Models: Pixel CNNs[5]



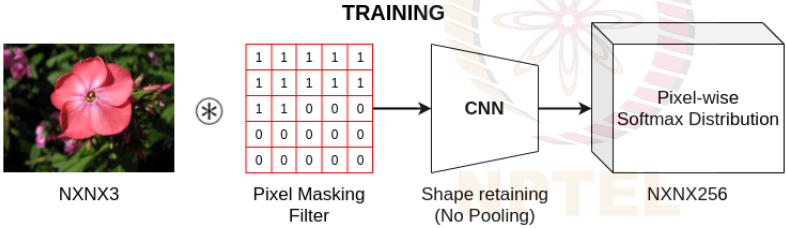
[5]van der Oord et al, Conditional Image Generation with PixelCNN Decoders, NeurIPS 2016

Autoregressive Models: Pixel CNNs[5]



[5]van der Oord et al, Conditional Image Generation with PixelCNN Decoders, NeurIPS 2016

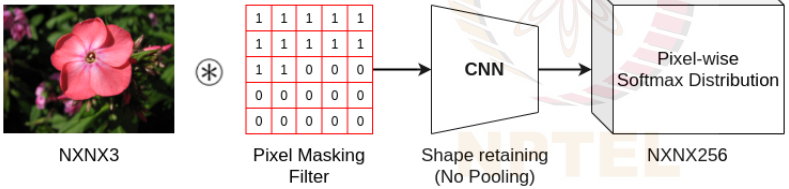
Autoregressive Models: Pixel CNNs[5]

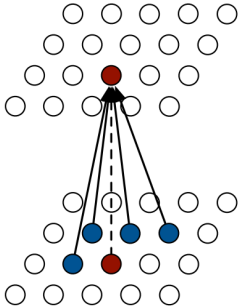


- Very fast to compute

[5]van der Oord et al, Conditional Image Generation with PixelCNN Decoders, NeurIPS 2016

Autoregressive Models: Pixel CNNs[5]





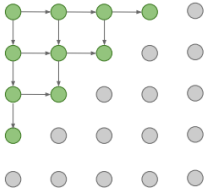
- Very fast to compute
- Does not make full use of context (local context)

[5] van der Oord et al, Conditional Image Generation with PixelCNN Decoders, NeurIPS 2016

Autoregressive Models: PixelRNN[6]

- Autoregressive model where images are generated pixel by pixel; each pixel depends on previous pixels based on a directed graph

$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, x_2, \cdots, x_{i-1})$$

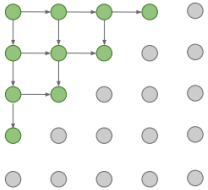


[6] van der Oord et al, Pixel Recurrent Neural Networks, ICML 2016

Autoregressive Models: PixelRNN[6]

- Autoregressive model where images are generated pixel by pixel; each pixel depends on previous pixels based on a directed graph

$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, x_2, \cdots, x_{i-1})$$



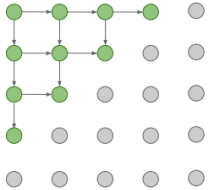
- Dependencies between pixels are modeled using Long Short Term Memory Networks (LSTMs)

[6] van der Oord et al, Pixel Recurrent Neural Networks, ICML 2016

Autoregressive Models: PixelRNN[6]

- Autoregressive model where images are generated pixel by pixel; each pixel depends on previous pixels based on a directed graph

$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, x_2, \cdots, x_{i-1})$$



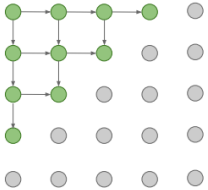
- Dependencies between pixels are modeled using Long Short Term Memory Networks (LSTMs)
- Trained to maximize likelihood using gradient descent

[6] van der Oord et al, Pixel Recurrent Neural Networks, ICML 2016

Autoregressive Models: PixelRNN[6]

- Autoregressive model where images are generated pixel by pixel; each pixel depends on previous pixels based on a directed graph

$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, x_2, \cdots, x_{i-1})$$



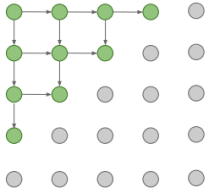
- Dependencies between pixels are modeled using Long Short Term Memory Networks (LSTMs)
- Trained to maximize likelihood using gradient descent
- **Advantage:** Likelihood is tractable

[6] van der Oord et al, Pixel Recurrent Neural Networks, ICML 2016

Autoregressive Models: PixelRNN[6]

- Autoregressive model where images are generated pixel by pixel; each pixel depends on previous pixels based on a directed graph

$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, x_2, \cdots, x_{i-1})$$



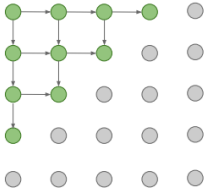
- Dependencies between pixels are modeled using Long Short Term Memory Networks (LSTMs)
- Trained to maximize likelihood using gradient descent
- **Advantage:** Likelihood is tractable
- **Limitation:** Image generation is slow (pixel by pixel)

[6] van der Oord et al, Pixel Recurrent Neural Networks, ICML 2016

Autoregressive Models: PixelRNN[6]

- Autoregressive model where images are generated pixel by pixel; each pixel depends on previous pixels based on a directed graph

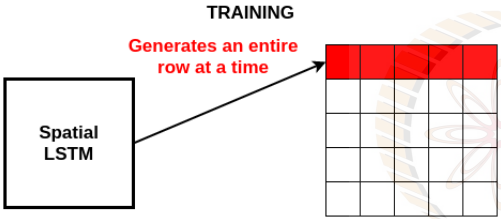
$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, x_2, \cdots, x_{i-1})$$



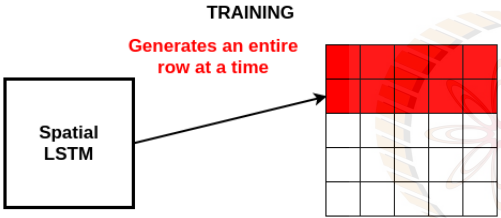
- Dependencies between pixels are modeled using Long Short Term Memory Networks (LSTMs)
- Trained to maximize likelihood using gradient descent
- **Advantage:** Likelihood is tractable
- **Limitation:** Image generation is slow (pixel by pixel)
- Example of a fully visible model

[6] van der Oord et al, Pixel Recurrent Neural Networks, ICML 2016

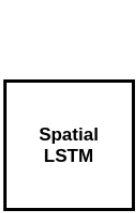
Pixel RNN: Row LSTM

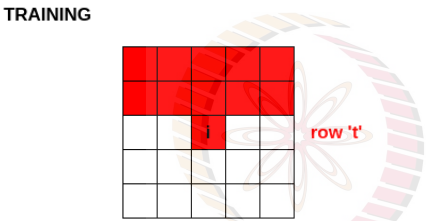


Pixel RNN: Row LSTM

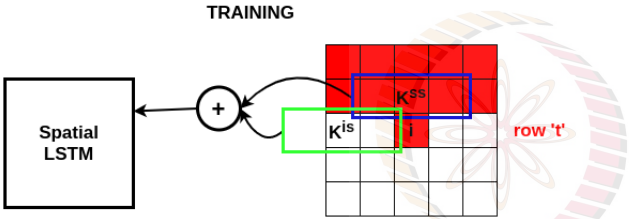


Pixel RNN: Row LSTM





Pixel RNN: Row LSTM



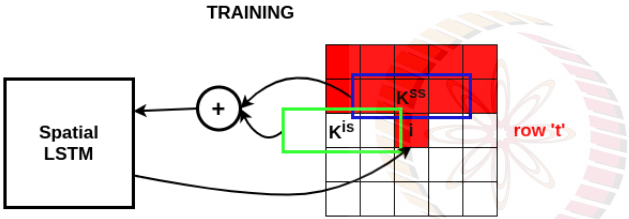
Hidden states (h_i) and cell states (c_i) given by:

$$[\boldsymbol{o_i}, \boldsymbol{f_i}, \boldsymbol{i_i}, \boldsymbol{g_i}] = \sigma(\boldsymbol{K^{ss}} \circledast \boldsymbol{h_{i-1}} + \boldsymbol{K^{is}} \circledast \boldsymbol{x_i})$$

$$\boldsymbol{c_i} = \boldsymbol{f_i} \odot \boldsymbol{c_{i-1}} + \boldsymbol{i_i} \odot \boldsymbol{g_i}$$

$$\boldsymbol{h_i} = \boldsymbol{o_i} \odot tanh(\boldsymbol{c_i})$$

Pixel RNN: Row LSTM



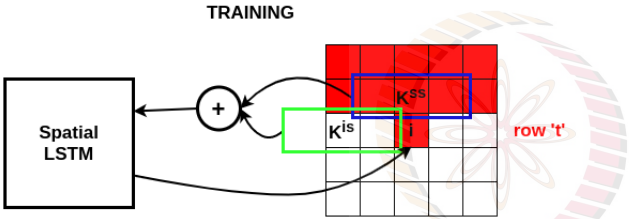
Hidden states (h_i) and cell states (c_i) given by:

$$[\boldsymbol{o_i}, \boldsymbol{f_i}, \boldsymbol{i_i}, \boldsymbol{g_i}] = \sigma(\boldsymbol{K^{ss}} \circledast \boldsymbol{h_{i-1}} + \boldsymbol{K^{is}} \circledast \boldsymbol{x_i})$$

$$\boldsymbol{c_i} = \boldsymbol{f_i} \odot \boldsymbol{c_{i-1}} + \boldsymbol{i_i} \odot \boldsymbol{g_i}$$

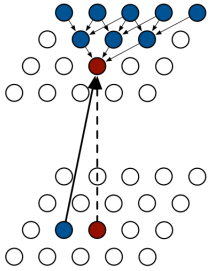
$$\boldsymbol{h_i} = \boldsymbol{o_i} \odot tanh(\boldsymbol{c_i})$$

Pixel RNN: Row LSTM



Hidden states (h_i) and cell states (c_i) given by:

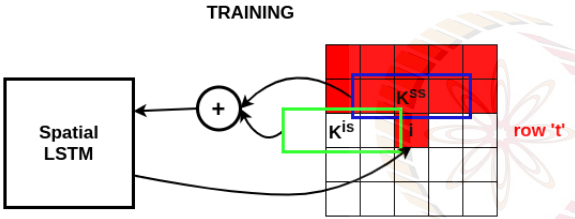
$$[\boldsymbol{o_i}, \boldsymbol{f_i}, \boldsymbol{i_i}, \boldsymbol{g_i}] = \sigma(\boldsymbol{K^{ss}} \circledast \boldsymbol{h_{i-1}} + \boldsymbol{K^{is}} \circledast \boldsymbol{x_i})$$
$$\boldsymbol{c_i} = \boldsymbol{f_i} \odot \boldsymbol{c_{i-1}} + \boldsymbol{i_i} \odot \boldsymbol{g_i}$$
$$\boldsymbol{h_i} = \boldsymbol{o_i} \odot tanh(\boldsymbol{c_i})$$



Row LSTM

- Triangular context

Pixel RNN: Row LSTM

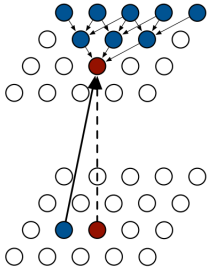


Hidden states (h_i) and cell states (c_i) given by:

$$[\boldsymbol{o_i}, \boldsymbol{f_i}, \boldsymbol{i_i}, \boldsymbol{g_i}] = \sigma(\boldsymbol{K^{ss}} \circledast \boldsymbol{h_{i-1}} + \boldsymbol{K^{is}} \circledast \boldsymbol{x_i})$$

$$\boldsymbol{c_i} = \boldsymbol{f_i} \odot \boldsymbol{c_{i-1}} + \boldsymbol{i_i} \odot \boldsymbol{g_i}$$

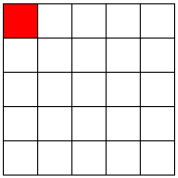
$$\boldsymbol{h_i} = \boldsymbol{o_i} \odot tanh(\boldsymbol{c_i})$$



Row LSTM

- Triangular context
- Slower than PixelCNNs

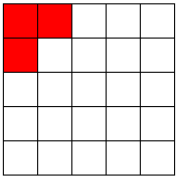
Pixel RNNs: Diagonal BiLSTM



TRAINING

Fills the image diagonal-wise

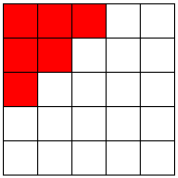
Pixel RNNs: Diagonal BiLSTM



TRAINING

Fills the image diagonal-wise

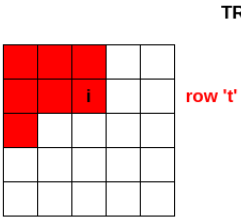
Pixel RNNs: Diagonal BiLSTM



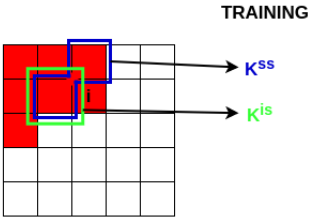
TRAINING

Fills the image diagonal-wise

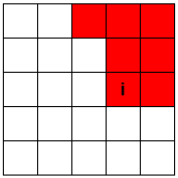
Pixel RNNs: Diagonal BiLSTM



Pixel RNNs: Diagonal BiLSTM



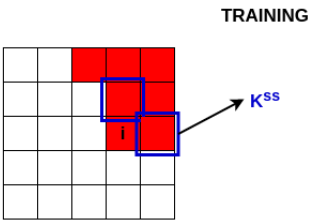
Pixel RNNs: Diagonal BiLSTM



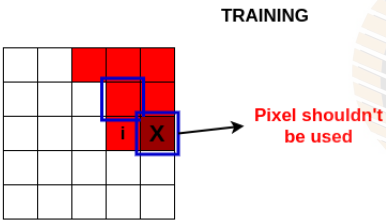
TRAINING

Repeat for the other diagonal

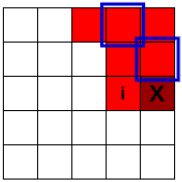
Pixel RNNs: Diagonal BiLSTM



Pixel RNNs: Diagonal BiLSTM



Pixel RNNs: Diagonal BiLSTM

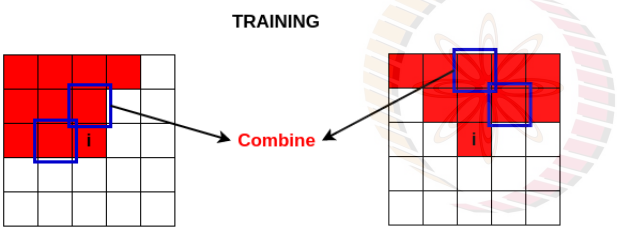


TRAINING

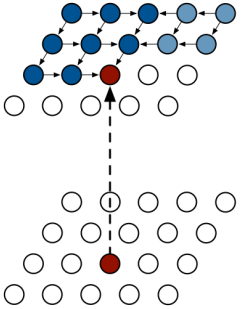
Shift the Conv window up

Pixel RNNs: Diagonal BiLSTM

Pixel RNNs: Diagonal BiLSTM

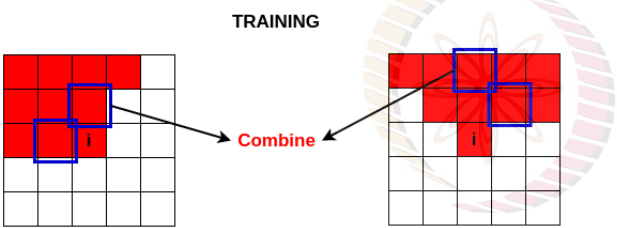


- Uses the full context

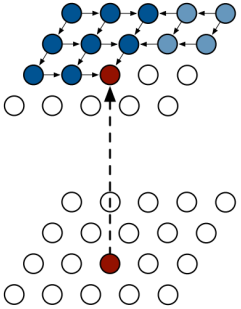


Diagonal BiLSTM

Pixel RNNs: Diagonal BiLSTM



- Uses the full context
- Slower than Pixel CNNs



Diagonal BiLSTM

Homework

Readings

- Flow-based Deep Generative Models, Lilian Weng
- Normalizing Flows:
  - Nonlinear Independent Components Estimation
  - Real-valued Non Volume Preserving Transformations(Real NVP)
- Autoregressive models:
  - Neural Autoregressive Distribution Estimation
  - Masked Autoencoder for Distribution Estimation
  - Pixel RNNs

References


Laurent Dinh, David Krueger, and Yoshua Bengio. "Nice: Non-linear independent components estimation". In: *arXiv preprint arXiv:1410.8516* (2014).


Mathieu Germain et al. "Made: Masked autoencoder for distribution estimation". In: *International Conference on Machine Learning*. 2015, pp. 881–889.


Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803* (2016).


Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *arXiv preprint arXiv:1601.06759* (2016).


Benigno Uria et al. "Neural autoregressive distribution estimation". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 7184–7220.


Phillip Isola et al. "Image-to-image translation with conditional adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.


Lilian Weng. *Flow-based Deep Generative Models*. 2018. URL: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html.