CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

CSDS 600: Deep Generative Models

Autoregressive Models

Yu Yin (yu.yin@case.edu)

Case Western Reserve University

https://yin-yu.github.io/

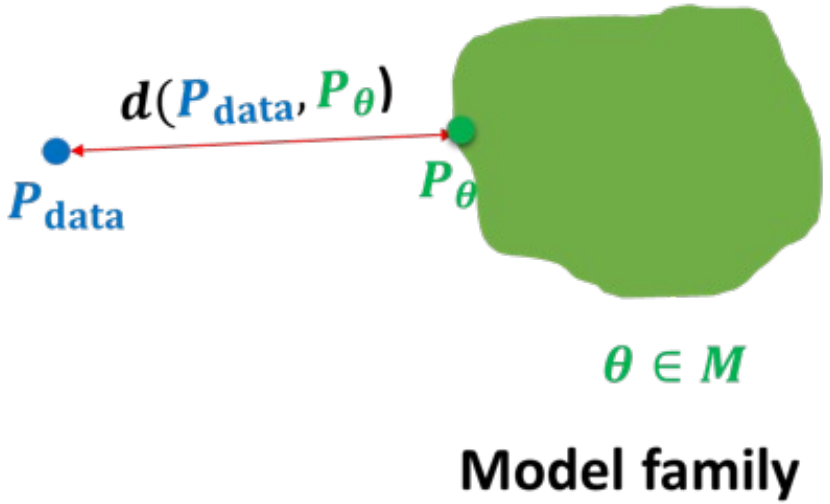
Outline

- Basic Representation
- Definition of Autoregressive Models
- Learning and Inference of Autoregressive Models
- Examples
  - FVSBN
  - NADE
  - MADE
  - Pixel RNN
  - Pixel CNN

Learning a generative model

- Given a training set of examples, e.g., images of dogs

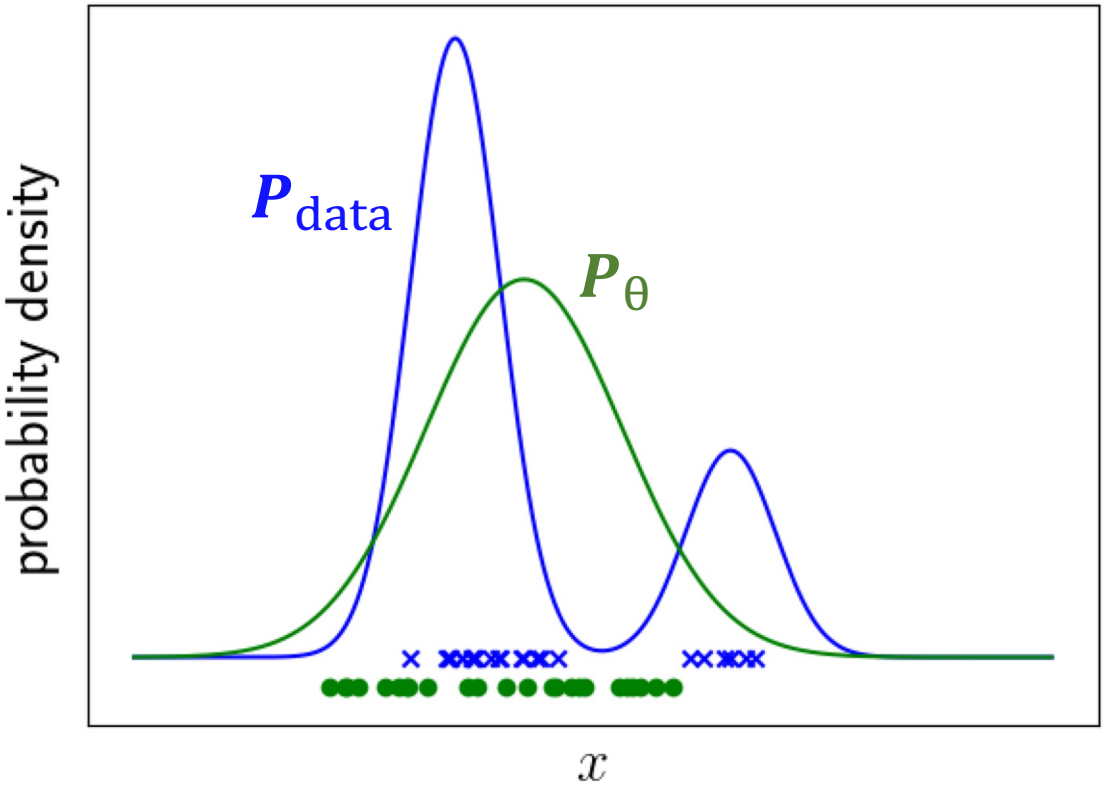
$$x_i \sim P_{\text{data}}$$
$$i = 1, 2, \ldots, n$$



- Goal: Learn a probability distribution $p(x)$ over images x

Credit to Stefano Ermon, Yang Song

Example



P_{data} : data distribution (unknown)

$\times$: data samples (known)

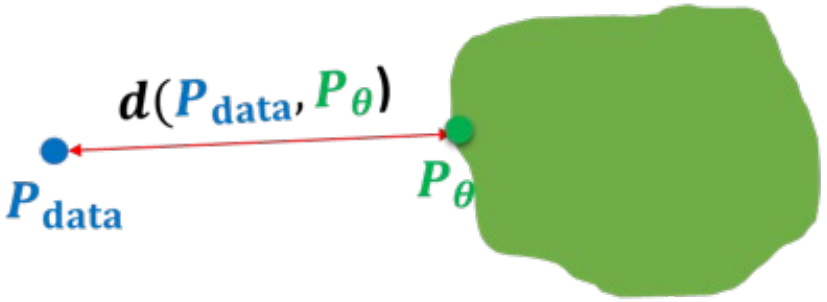
M : mixture of 1d Gaussians

P_θ : generative model

● : generated samples

Challenge of High Dimensionality

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



- P_{data} can contain many modes
- Training samples cover very small region of true support

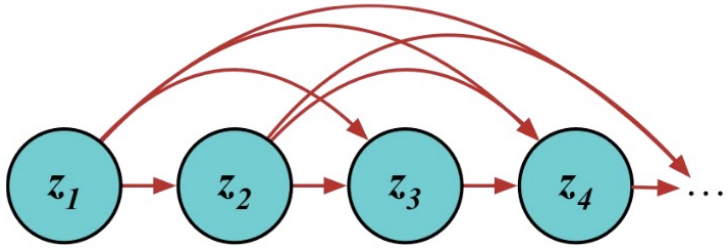
Learning a generative model

- Three Objectives:
  - **Generation**: If we sample $x_{new} \sim p(x)$, x_{new} should look like a dog (sampling)
  - **Density estimation**: $p(x)$ should be high if x looks like a dog, and low otherwise (anomaly detection)
  - **Unsupervised representation learning**: We should be able to learn what these images have in common, e.g., ears, tail, etc. (features)
- Question: how to **represent** and **learn** $p(x)$?

Types of Generative Models

- **Fully-observed models**

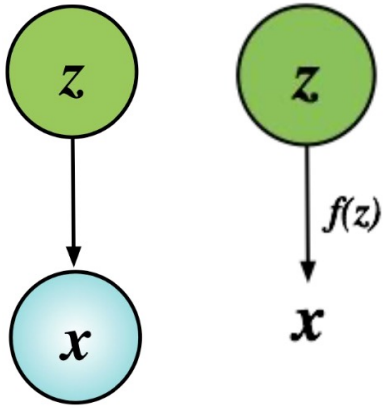
  Model observed data directly without introducing any new unobserved local variables.



- **Latent Variable Models**

  Introduce an unobserved random variable for every observed data point to explain hidden causes.

  - **Prescribed models**: Use observer likelihoods and assume observation noise.
  - **Implicit models**: Likelihood-free models (*e.g.,* GANs).



Motivating Example: MNIST

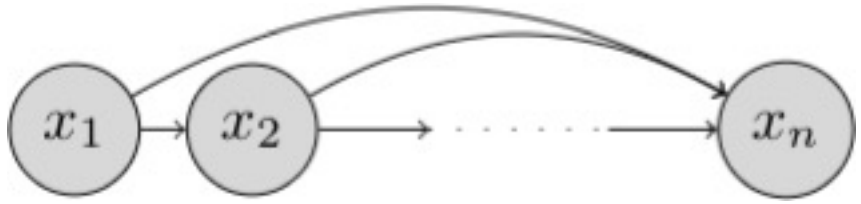
- **Given**: a dataset D of handwritten digits (binarized MNIST)

Each image has n = 28 × 28 = 784 pixels. Each pixel can either be black (0) or white (1).

- **Goal**: Learn a probability distribution $p(x) = p(x_1, \ldots, x_{784})$ over $x \in \{0, 1\}^{784}$ such that when $x \sim p(x)$, x looks like a digit

Autoregressive Models

- Directed, fully-observed graphical models



Key idea: Decompose the joint distribution as a **product of tractable conditionals**

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i|x_1, x_2, \ldots, x_{i-1}) = \prod_{i=1}^{n} p_\theta(x_i|x_{<i})$$

Autoregressive Models

- We can pick an ordering of all the random variables, i.e., raster scan ordering of pixels from top-left (X_1) to bottom-right ($X_{n=784}$)

- Without loss of generality, we can use chain rule for factorization

$$p(x_1, \cdots, x_{784}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$$

Too complex!

Chain Rule: $p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2)p(x_4 \mid x_1, x_2, x_3)$

Autoregressive Models

- To simplify, we assume

$$p(x_1, \cdots, x_{784}) = p_{\text{CPT}}(x_1; \alpha^1) p_{\text{logit}}(x_2 \mid x_1; \boldsymbol{\alpha}^2) p_{\text{logit}}(x_3 \mid x_1, x_2; \boldsymbol{\alpha}^3) \cdots p_{\text{logit}}(x_n \mid x_1, \cdots, x_{n-1}; \boldsymbol{\alpha}^n)$$

conditional Bernoulli

$$p_{\text{CPT}}(X_1 = 1; \alpha^1) = \alpha^1, \; p(X_1 = 0) = 1 - \alpha^1$$
$$p_{\text{logit}}(X_2 = 1 \mid x_1; \boldsymbol{\alpha}^2) = \sigma(\boldsymbol{\alpha}_0^2 + \boldsymbol{\alpha}_1^2 x_1)$$
$$p_{\text{logit}}(X_3 = 1 \mid x_1, x_2; \boldsymbol{\alpha}^3) = \sigma(\boldsymbol{\alpha}_0^3 + \boldsymbol{\alpha}_1^3 x_1 + \boldsymbol{\alpha}_2^3 x_2)$$

- Parameterized functions (e.g., logistic regression) is used to predict next pixel given all the previous ones.

Learning and Inference

- **Learning** maximizes the model log-likelihood over the dataset !

$$\max_{\theta} \log p_\theta(\mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \log p_\theta(\mathbf{x}) = \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i=1}^{n} \log p_\theta(x_i | x_{<i}))$$

- Tractable conditionals allow for **exact likelihood evaluation**
  - Conditional evaluated in parallel during training
- Directed model permits **ancestral sampling**, one variable at a time

$$x_1 \sim p_\theta(x_1), x_2 \sim p_\theta(x_2|x_1), \ldots, x_n \sim p_\theta(x_n|x_{<n})$$

Learning and inference

- **Inference** samples each variable of one data from estimated conditional distributions step by step, until the whole data is generated.
- Ancestral sampling:

  A process of producing samples from a probabilistic model.

  - **First** sample variables which has no conditional constraints using their prior distribution. $x_1 \sim p_\theta(x_1)$
  - **Then** sample child variables using conditional distribution based on their parents and repeat so on. $x_2 \sim p_\theta(x_2|x_1)$
- The attribute of Autoregressive Models that directly model and output distributions allows for ancestral sampling.

Learning and inference

Differences between Autoregressive models (AR), VAE and GAN:

- **GAN** model doesn't define any distribution, it adapts discriminator to learn the data distribution implicitly. $P(X, Z) = P(X|Z)P(Z)$
- **VAE** model believes the data distribution is too complex to model directly, thus it tries to learn the distribution by defining an intermediate distribution and learning the map between the defined simple distribution to the complex data distribution. $P(X, Z) = P(X|Z)P(Z)$
- **AR** model on the one hand assumes that the data distribution can be learned directly (tractable), then it define its outputs as conditional distributions to solve the generation problem by directly modeling each conditional distribution.

Learning and inference

Conclusion:

1. Using complex networks, each step Autoregressive Models output an approximated complex conditional distribution $\widehat{x_i} = p_\theta(x_i|x_1, x_2, \dots, x_{i-1})$
2. Taking in the previous inputs $x_1, x_2, \dots, x_{i-1}$ and the next input x_i by sampling previous estimated conditional distribution $\widehat{x_i}$, Autoregressive Model is able to generate all conditional distribution iteratively

$$x_1 \sim P_\theta(x_1),\ x_2 \sim P_\theta(x_2|x_1),\ x_3 \sim P_\theta(x_3|x_1, x_2), \dots\ x_n \sim P_\theta(x_n|x_1, \dots, x_{n-1})$$

3. Product rule makes sure the generated data that made up of sampled result x_i from each step follows the data distribution.

$$(x_1, x_2, \dots, x_n) \sim \prod_{i=1}^{n} p_\theta(x_i|x_{<i})$$

Parameterizing Autoregressive Models

Linear/multi-layer NN based parameterizations

- Fully-visible sigmoid belief networks (FVSBN)
- Neural autoregressive density estimator (NADE): 1 hidden layer NN
- Masked autoencoder distribution estimation (MADE): multilayer NN

Neal, 1992, Larochelle & Murray, 2011, Germain et al., 2015

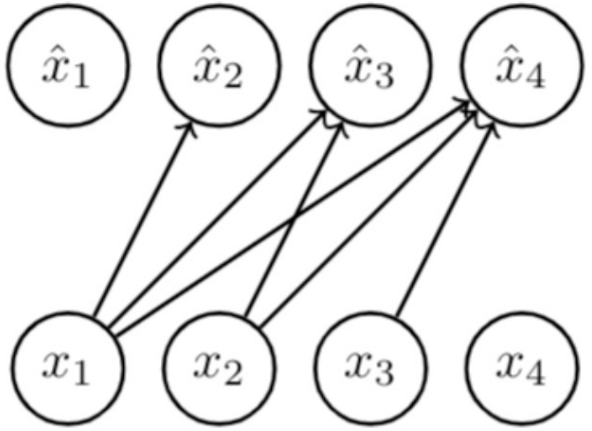
Fully Visible Sigmoid Belief Network (FVSBN)

- The fully visible sigmoid belief network **without any hidden** units is denoted FVSBN.
- The conditional variables $x_i|x_{<i}$ in FVSBN are Bernoulli with parameters.
- Some conditionals are too complex. So FVSBN assume:

$$\hat{x}_i = p(x_i = 1|x_1, x_2, \dots x_{i-1}) = f_i(x_1, x_2, \dots, x_{i-1}; \alpha^i)$$

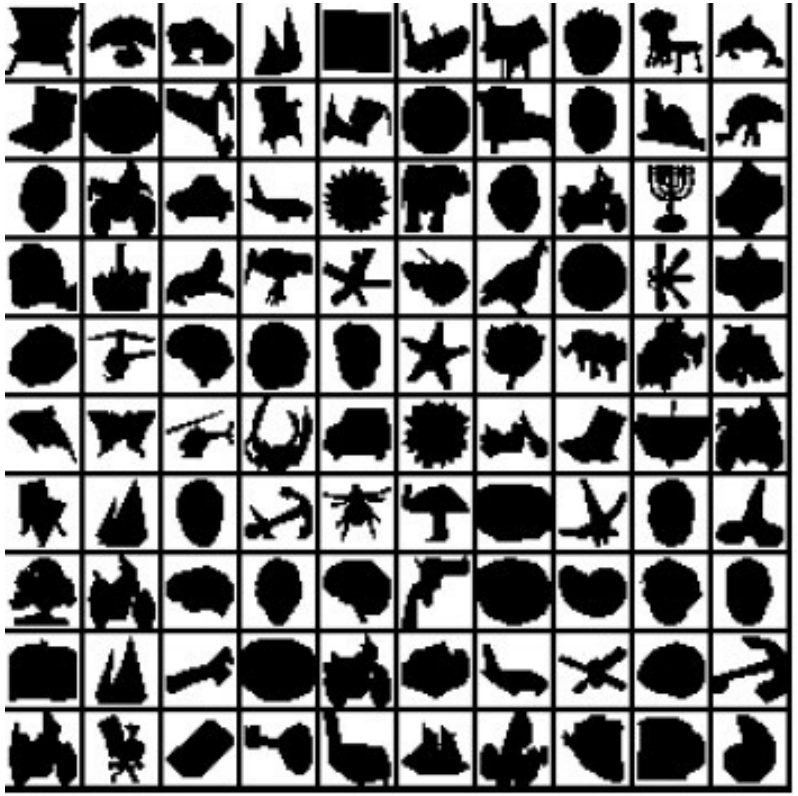
$$= \sigma(\alpha_0^{(i)} + \alpha_1^{(i)} x_1 + \cdots + \alpha_{i-1}^{(i)} x_{i-1})$$

- # of parameters: $1 + 2 + 3 + \cdots + n \approx n^2/2$

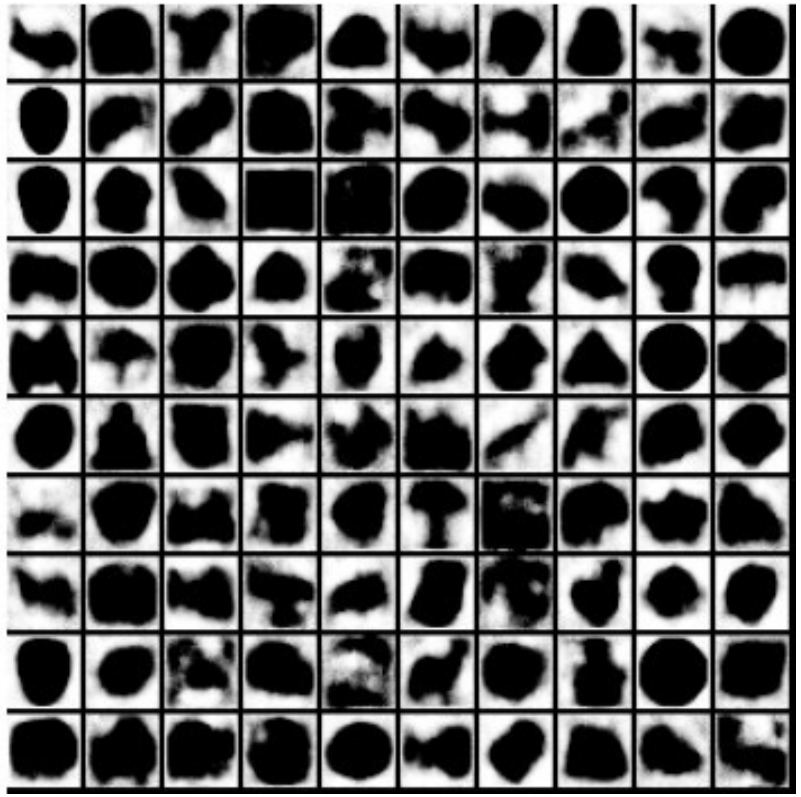


FVSBN

FVSBN Results



Training data from Caltech 101 Silhouettes



Samples from the model

Learning Deep Sigmoid Belief Networks with Data Augmentation, Gan et al. 2015.

Neural autoregressive density estimator (NADE)

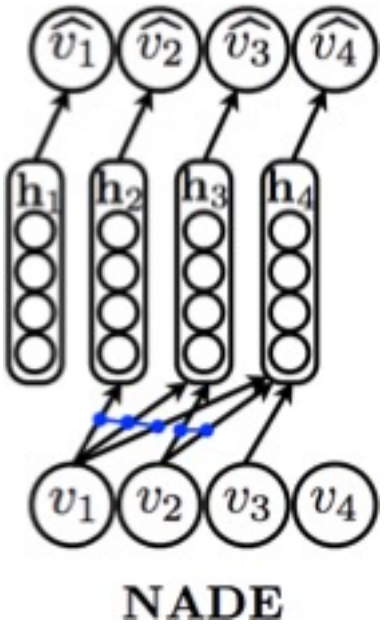
Improve FVSBN: use one hidden layer neural network instead of logistic regression

$$\boldsymbol{h_i} = \sigma(A_i \boldsymbol{x}_{<i} + \boldsymbol{c_i})$$

$$\hat{\boldsymbol{x}}_i = p(x_i | x_1, x_2 \ldots x_{i-1}; \underbrace{A_i, \boldsymbol{c_i}, \boldsymbol{\alpha_i}, b_i}_{\text{parameters}}) = \sigma(\boldsymbol{\alpha_i h_i} + b_i)$$

$x_i = v_i$

$$\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \\ w_1 \\ \vdots \end{pmatrix}}_{A_2} x_1\right) \quad \mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ w_1 & w_2 \\ \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) \quad \mathbf{h}_4 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$$



NADE

Tie weights are shared to *reduce the number of parameters* and *speedup computation*
(see blue dots in the figure)

Neal, 1992, Larochelle & Murray, 2011

NADE

$$\boldsymbol{h_i} = \sigma(A_i \boldsymbol{x_{<i}} + \boldsymbol{c_i})$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{i}} = p(x_i | x_1, x_2 \dots x_{i-1}; A_i, \boldsymbol{c_i}, \boldsymbol{\alpha_i}, b_i)$$

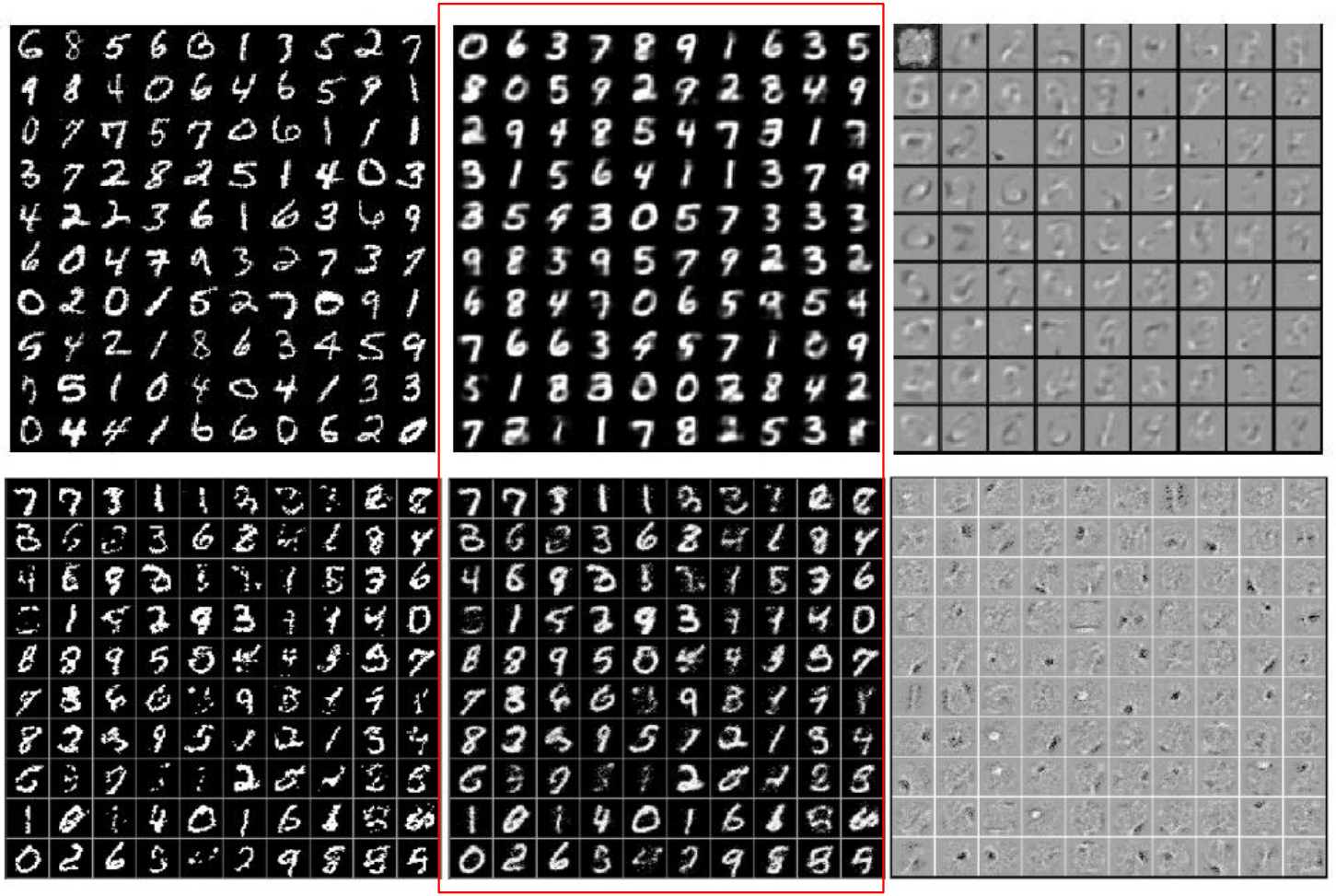
$$= f_i(x_1, x_2, \dots, x_{i-1}) = \sigma(\boldsymbol{\alpha_i h_i} + b_i)$$

- Parameters: $\theta_i = \{A_i \in R^{d \times (i-1)}, \boldsymbol{c_i} \in R^d, \boldsymbol{\alpha_i} \in R^d, bi \in R\}$
- The total number of parameters in this model is dominated by the matrices $\{\boldsymbol{A1}, \boldsymbol{A2}, \dots, \boldsymbol{An}\}$ given by O(n^2 * d).
- Sharing parameters: Tie weights are shared to reduce the number of parameters and speed up computation. → O(n*d).

Results on MNIST

FVSBN

NADE



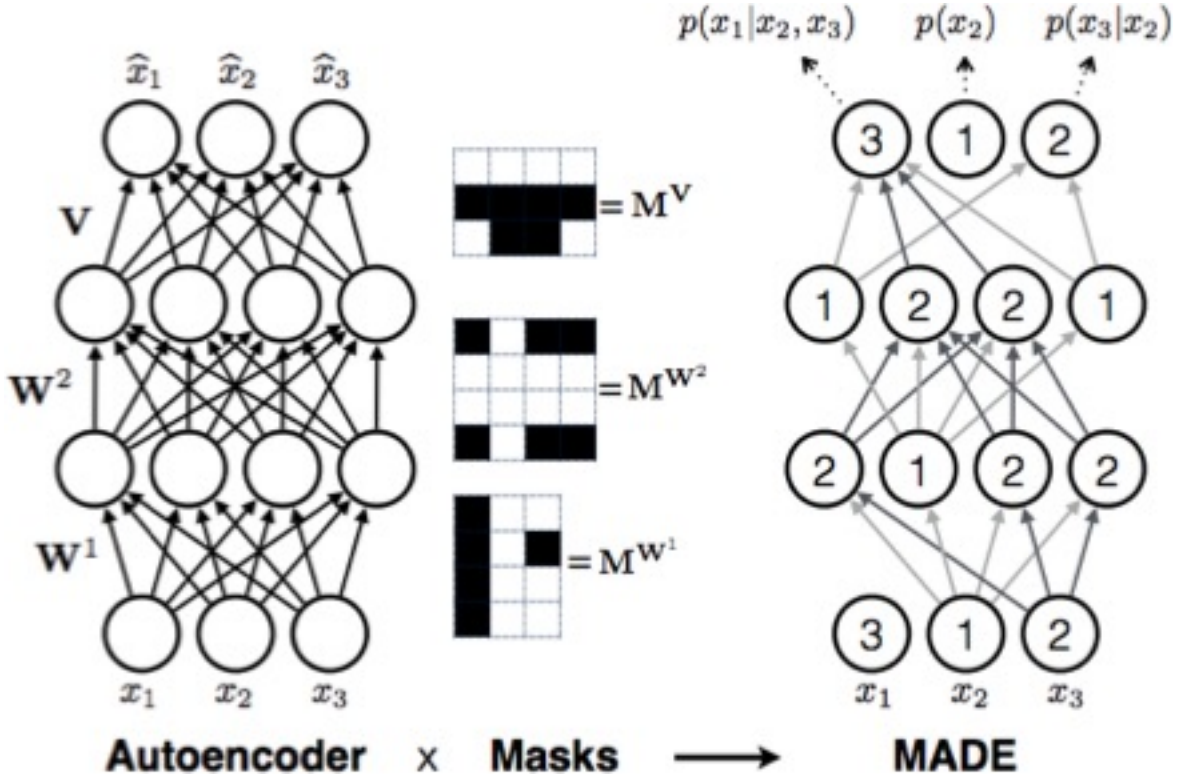
Training data

Generated results

Learned Features of the bottom layer

Masked autoencoder distribution estimation (MADE)

- Extension to multilayer NN

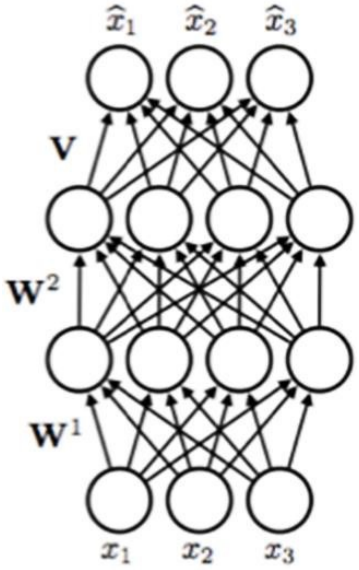


Germain et al., 2015

Autoregressive model vs. Autoencoders

- FVSBN and NADE look similar to an **autoencoder.**
- An **encoder** $e(\cdot)$. E.g., $e(x) = \sigma(W^2(W^1 x + b^1) + b^2)$
- A **decoder** such that $d(e(x)) \approx x$. E.g., $d(h) = \sigma(Vh + c)$.
- Loss function: (if continuous)

$$\min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in D} \sum_{i} (x_i - \hat{x}_i)^2$$

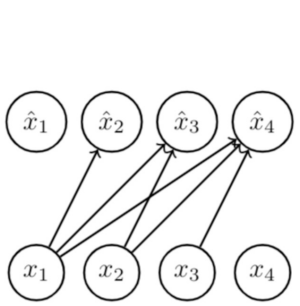


Autoencoder

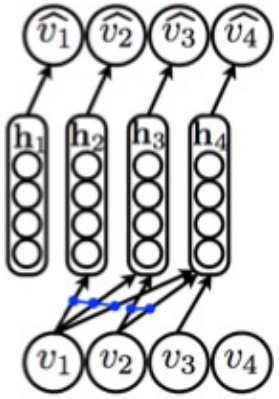
- Encoder: feature learning
- A vanilla autoencoder is not a generative model: it does not define a distribution over x we can sample from to generate new data points.

Autoregressive model vs. Autoencoders

- FVSBN and NADE look similar to an autoencoder.
- Can we get a generative model from an Autoencoder?

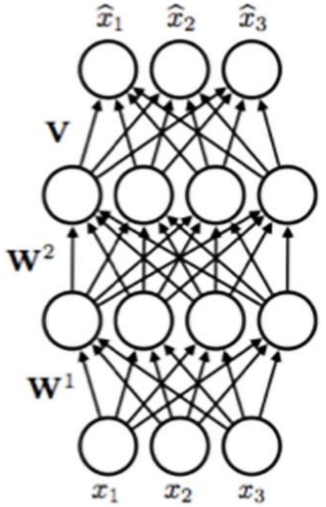


FVSBN



NADE

VS



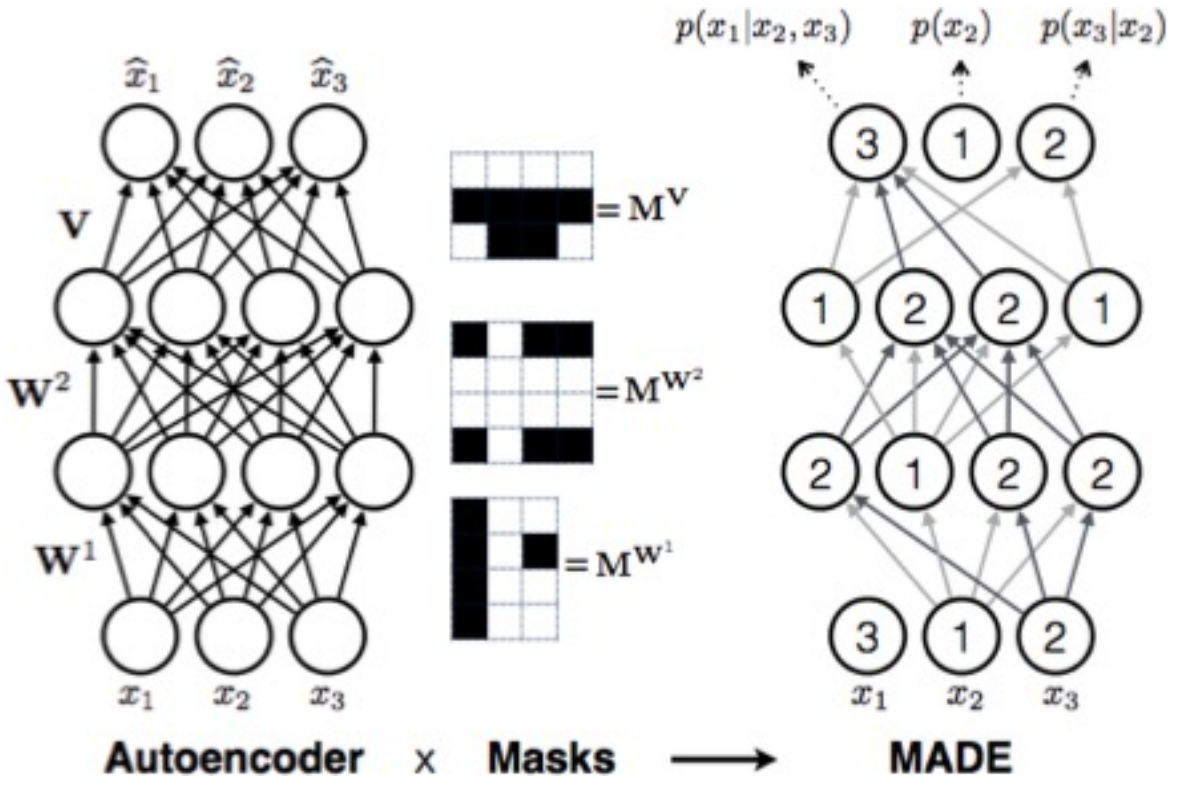
Autoencoder

➢A dependency order constraint is required for Autoencoder to make it a Bayesian Network.

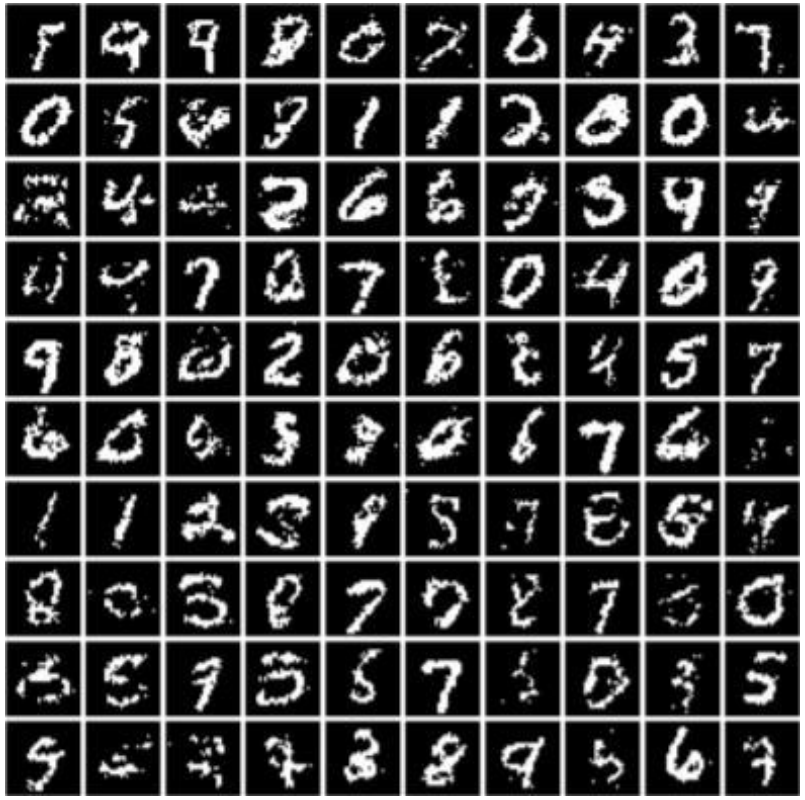
Masked autoencoder distribution estimation (MADE)

- Use Masks to constraint dependency paths!
- Each output unit is an estimated distribution, it only depends on the inputs with orderings that before its chosen ordering

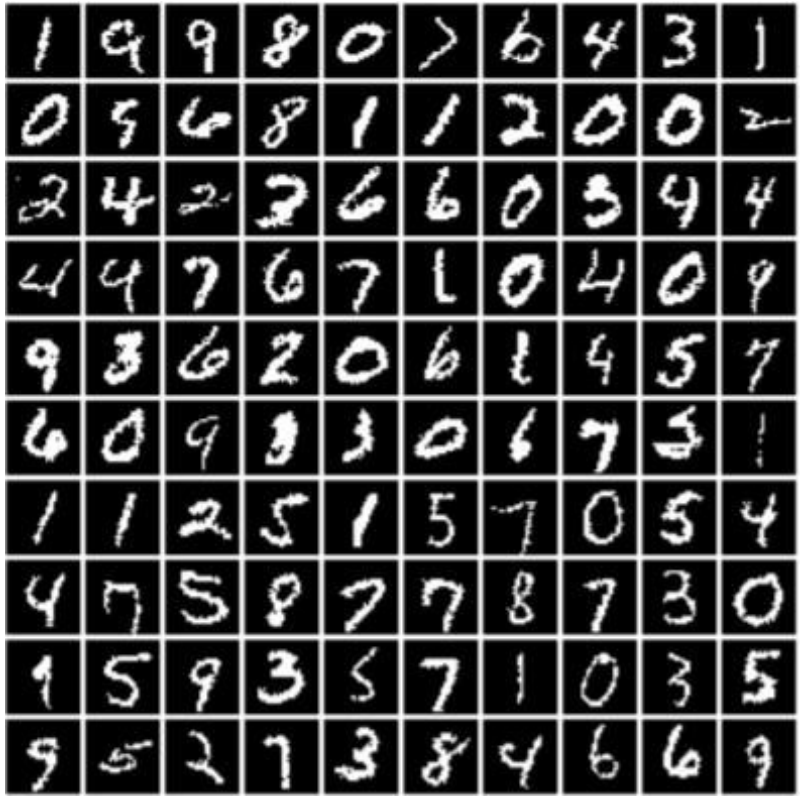
- With the order x_2, x_3, x_1:
  1. $p(x_2)$ doesn't depends on any input
  2. $p(x_3|x_2)$ depends on input x_2
  3. $p(x_1|x_2, x_3)$ depends on input x_2, x_3



MADE Results on MNIST



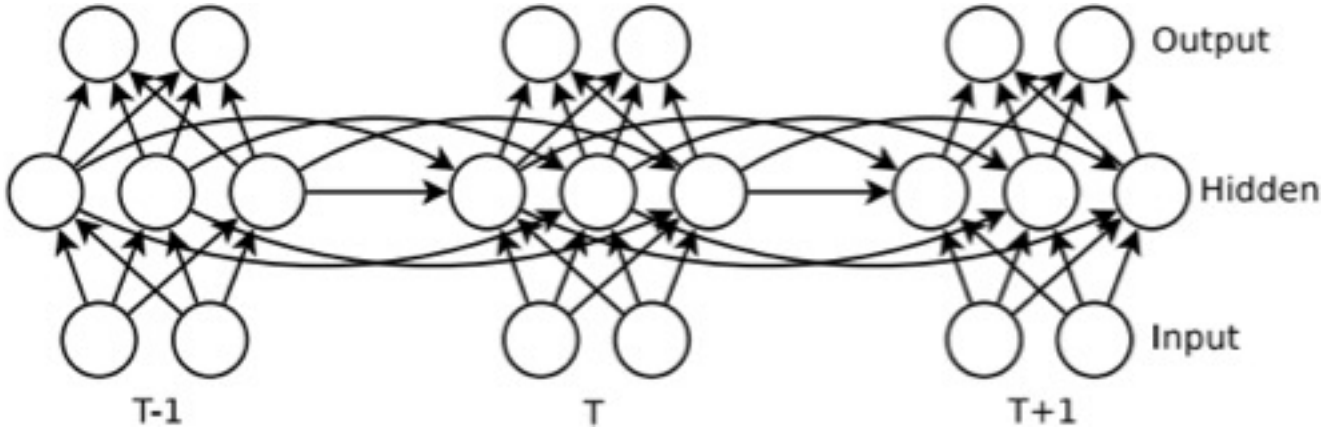
Samples from a 2 hidden layer MADE



Nearest neighbor in dataset

RNN-based parameterizations

Effective architecture for learning long-range sequential dependencies

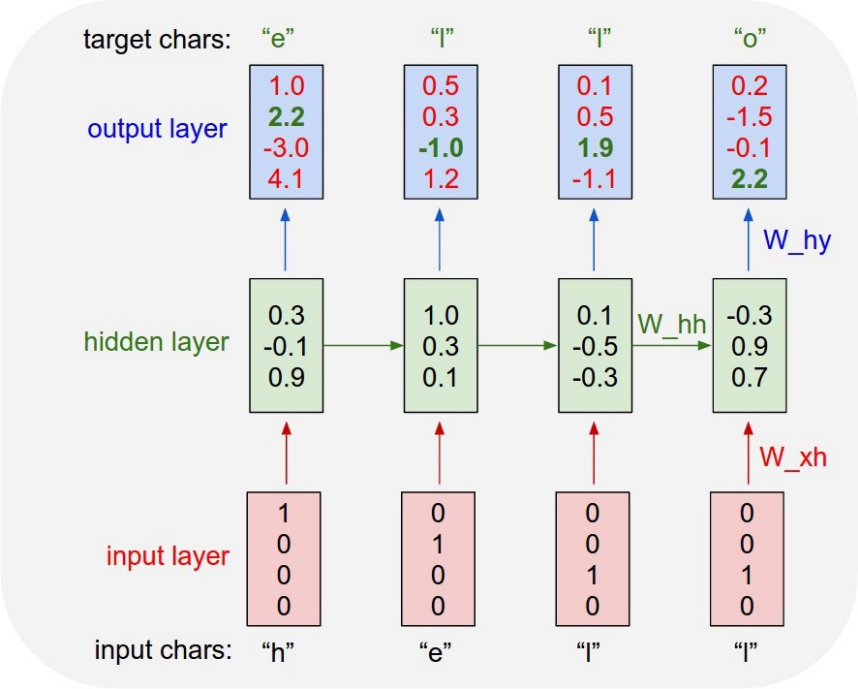


Hidden layer $h_\theta(t)$ is a summary of the inputs seen till $t-1$

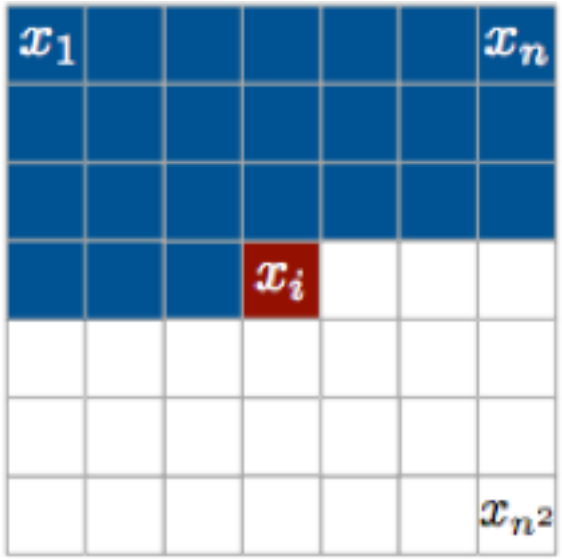
Output layer $o_\theta(t)$ specifies parameters for conditionals $p_\theta(x_t \mid x_{<t})$

Sutskevar et al., 2011

RNN-based parameterizations



Char-RNN



Pixel-RNN

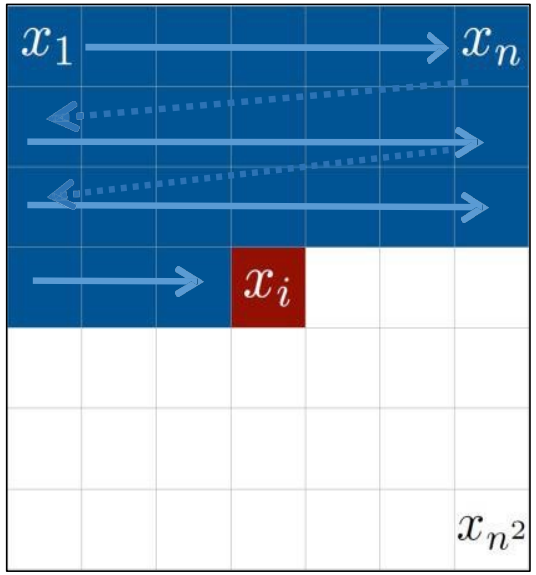
Sutskevar et al., 2011, Karpathy, 2015, Theis & Bethge, 2015, van den Oord, 2016a

Pixel RNN (Oord et al., 2016)

- Model images pixel by pixel using raster scan order

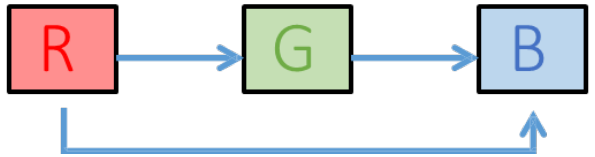
$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1}).$$

- Sequentially predict pixels rather than predicting the whole image at once (like as GAN, VAE)



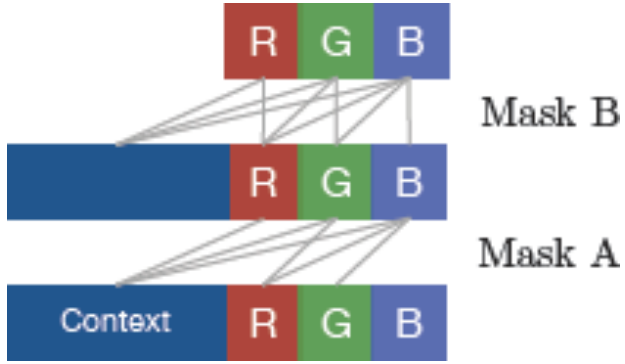
Pixel RNN (Oord et al., 2016)

- For color image, 3 channels are generated successive conditioning, blue given red and green, green given red, and red given only the pixels above and to the left of all channels



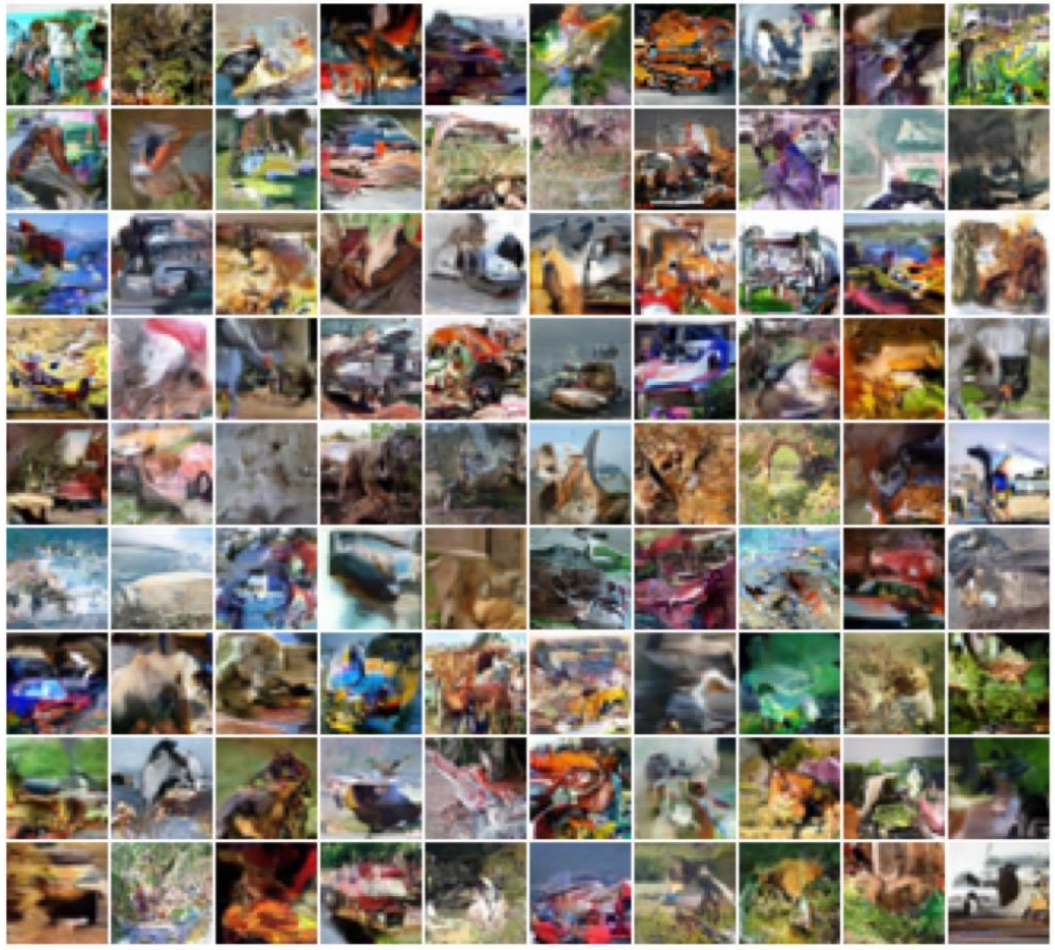
$$p(x_t \mid x_{1:t-1}) = p(x_t^{red} \mid x_{1:t-1})p(x_t^{green} \mid x_{1:t-1}, x_t^{red})p(x_t^{blue} \mid x_{1:t-1}, x_t^{red}, x_t^{green})$$

- Conditionals modeled using RNN variants. LSTMs + masking (like MADE)

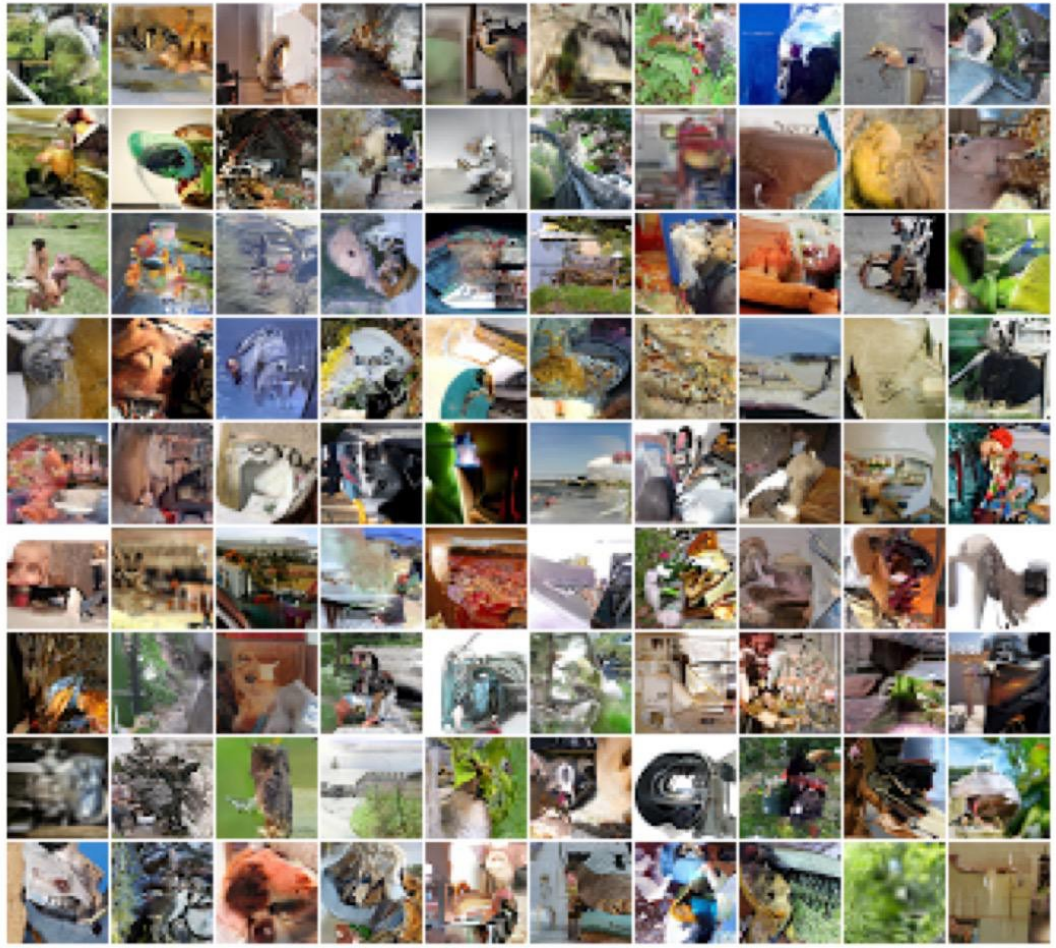


Results of Pixel RNN

CIFAR-10



ImageNet 32x32



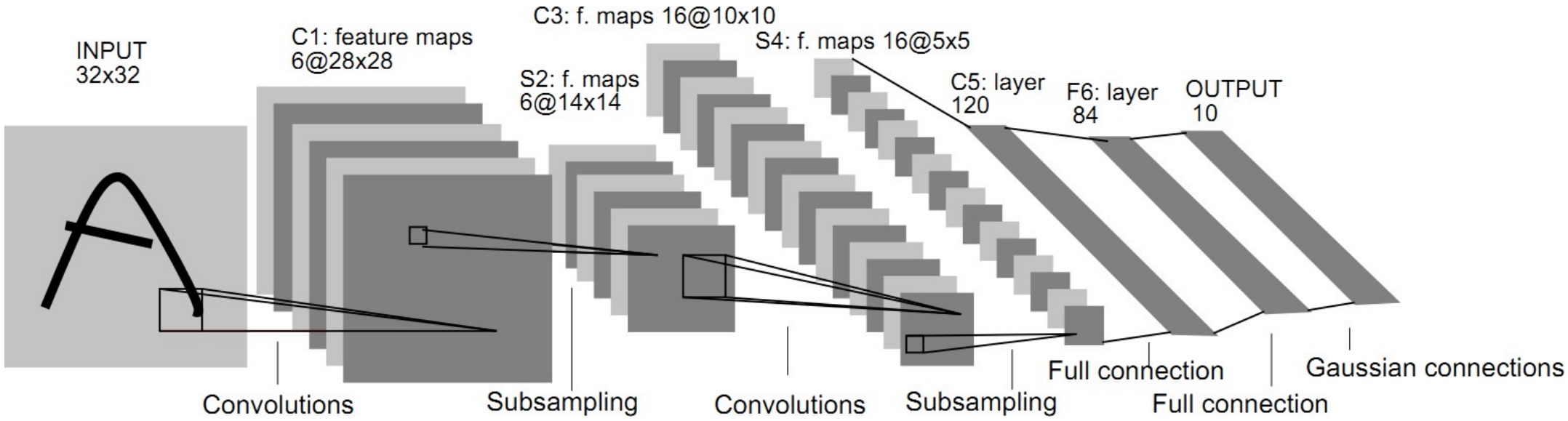
Results of Pixel RNN

Results on downsampled ImageNet

Very slow! (sequential likelihood evaluation)

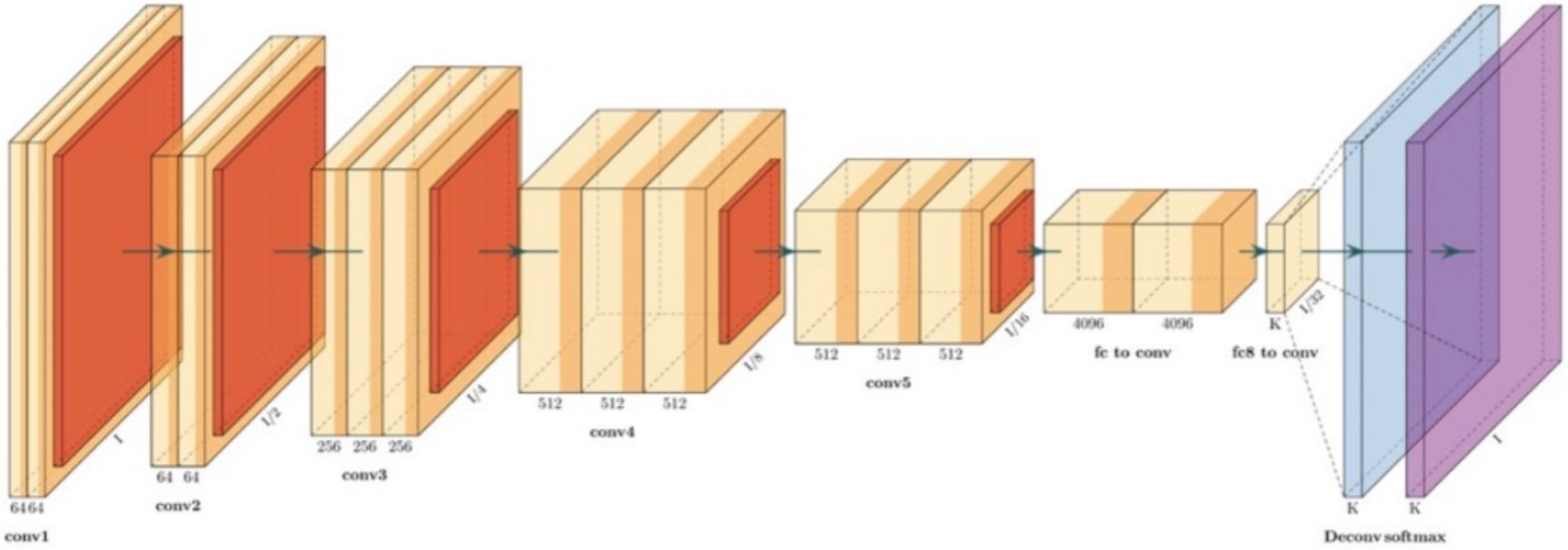
CNN-based parameterizations

Convolutions are amenable for parallelization on modern hardware



Lecunn et al., 1998

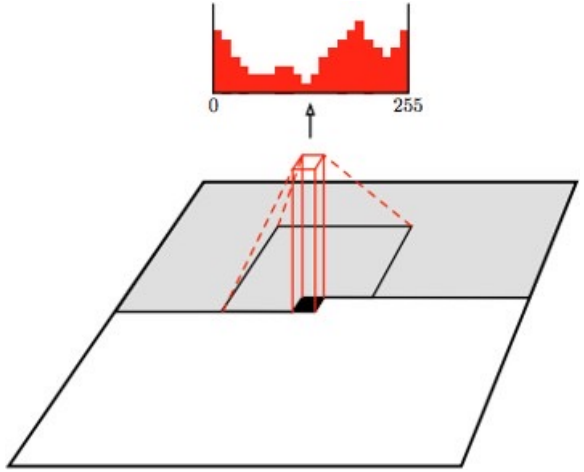
Convolutional Architectures

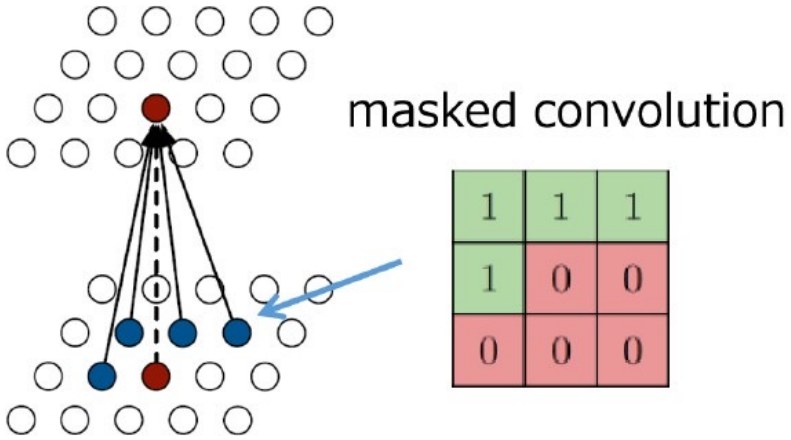


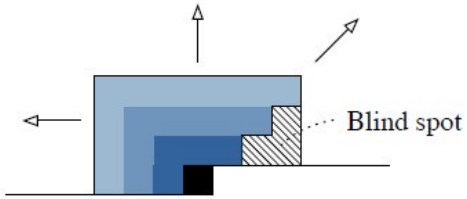
Convolutions are natural for image data and easy to parallelize on modern hardware.

PixelCNN (Oord et al., 2016)

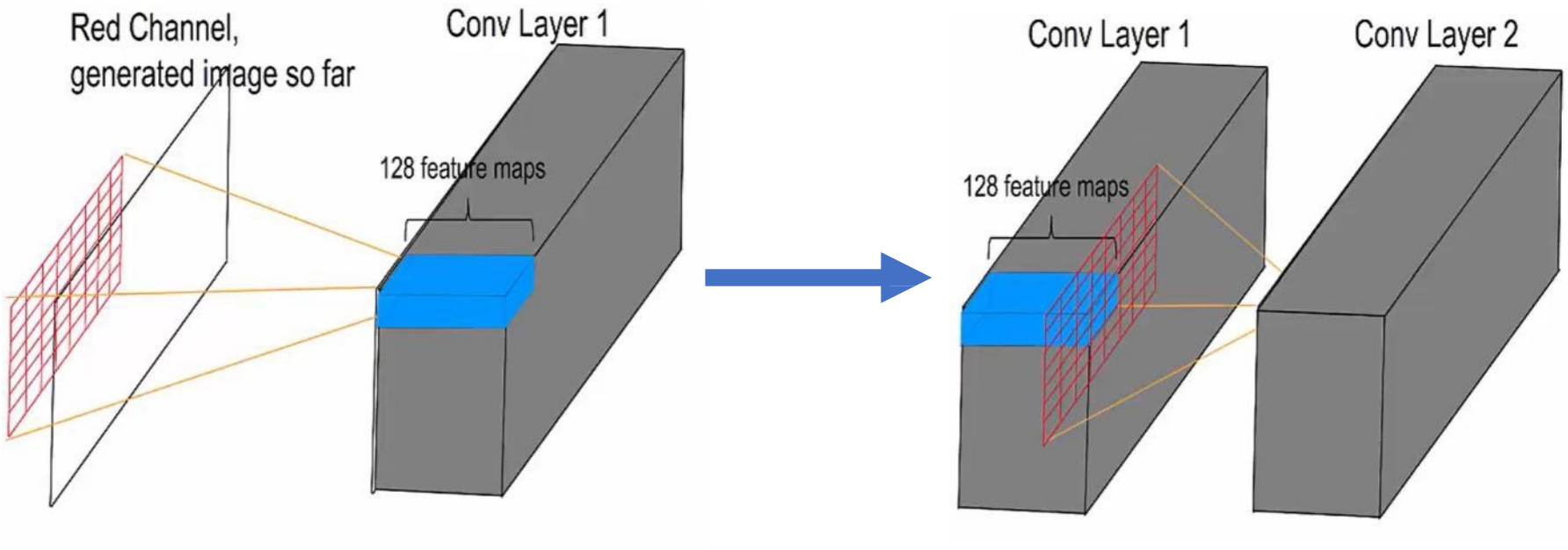
- **Idea**: Use convolutional architecture to predict next pixel given context (a neighborhood of pixels).
- **Challenge**: Has to be autoregressive. Masked convolutions preserve raster scan order. Additional masking for colors order.

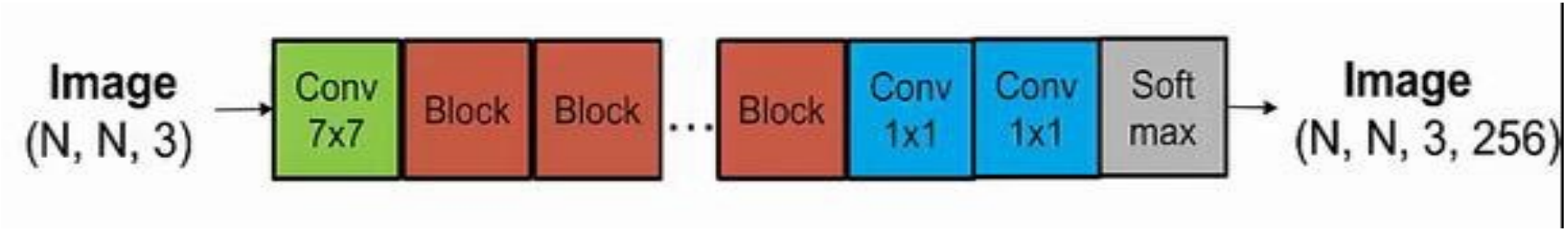




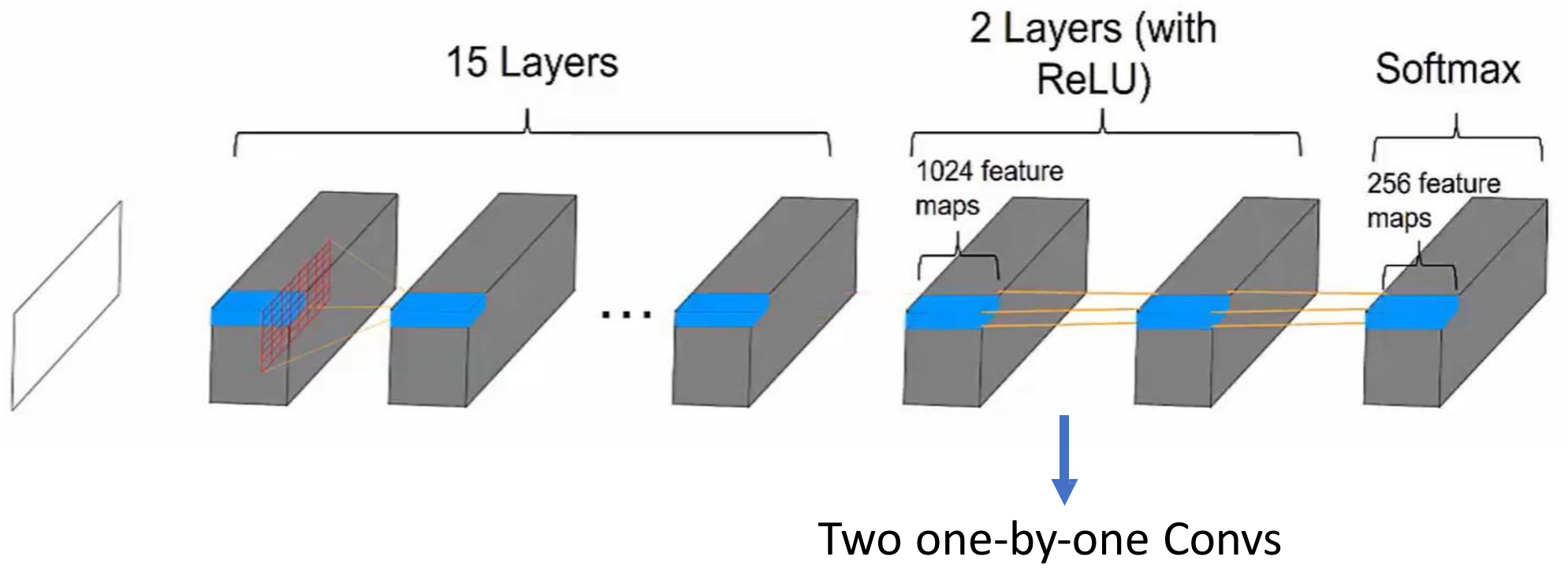


PixelCNN Network Structure

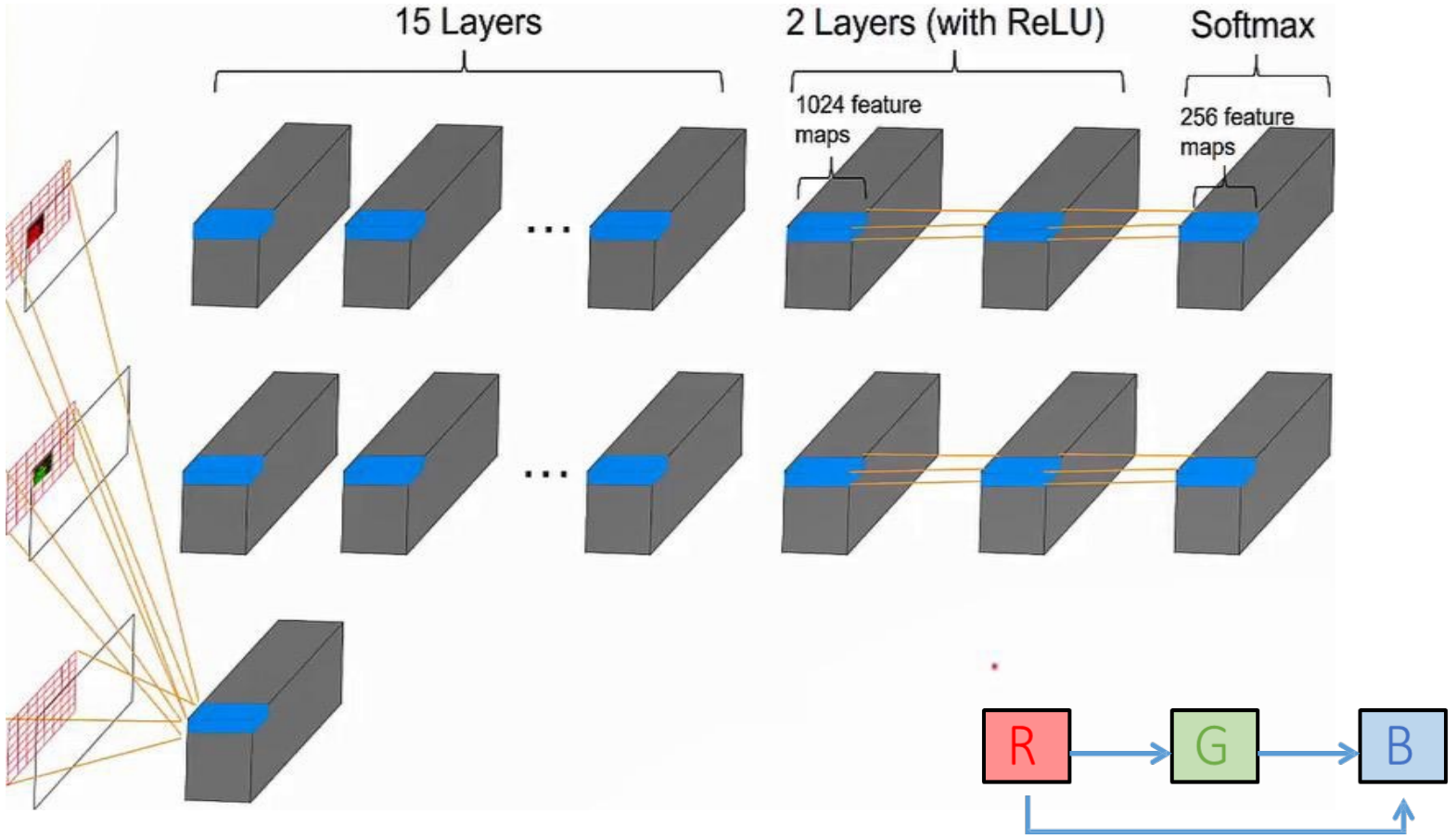




PixelCNN Network Structure

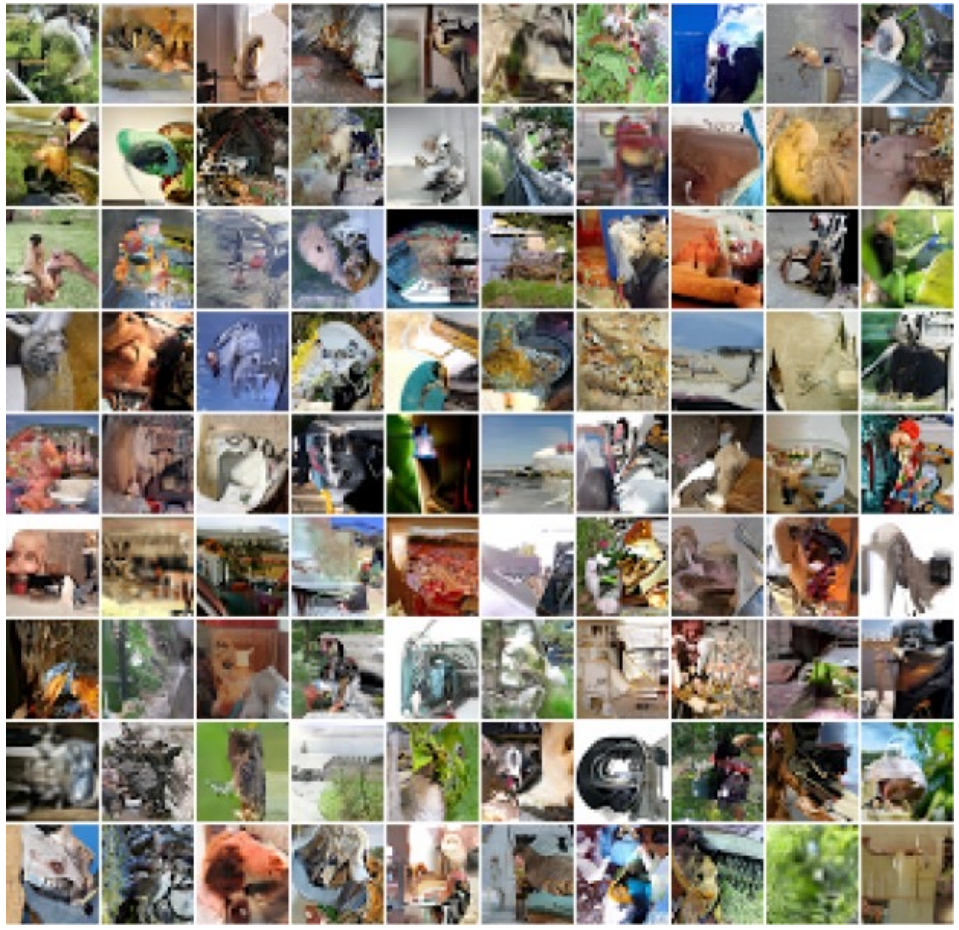


PixelCNN Network Structure



PixelCNN

Samples from the model trained on Imagenet (32 × 32 pixels). Similar performance to PixelRNN, but much faster.



Summary

- Autoregressive models:
  - Chain rule based factorization is fully general
  - Compact representation via conditional independence and/or neural parameterizations
- Autoregressive models Pros:
  - Easy to evaluate likelihoods
  - Easy to train
- Autoregressive models Cons:
  - Requires an ordering
  - Generation is sequential
  - Cannot learn features in an unsupervised way

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Thank You

- Questions?

- Email: yu.yin@case.edu