Recap.

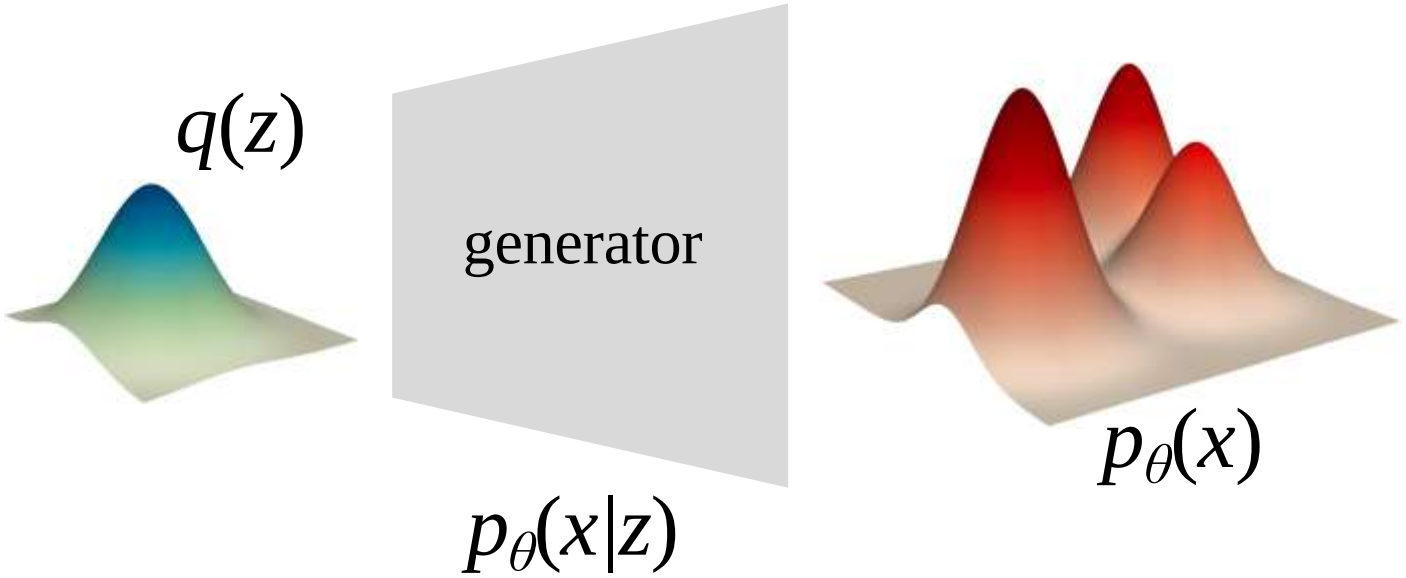
Recap: Latent Variable Models

Two sets of variables:

- q: distribution of latent
- θ: parameters of generator

VAE:

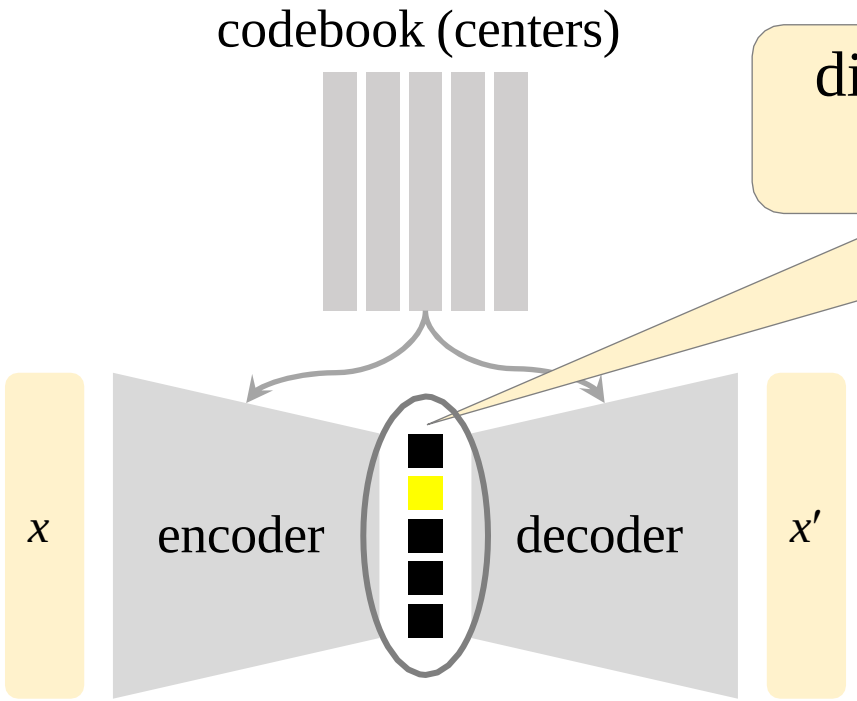
- parametrize q by a network
- stochastic gradient decent

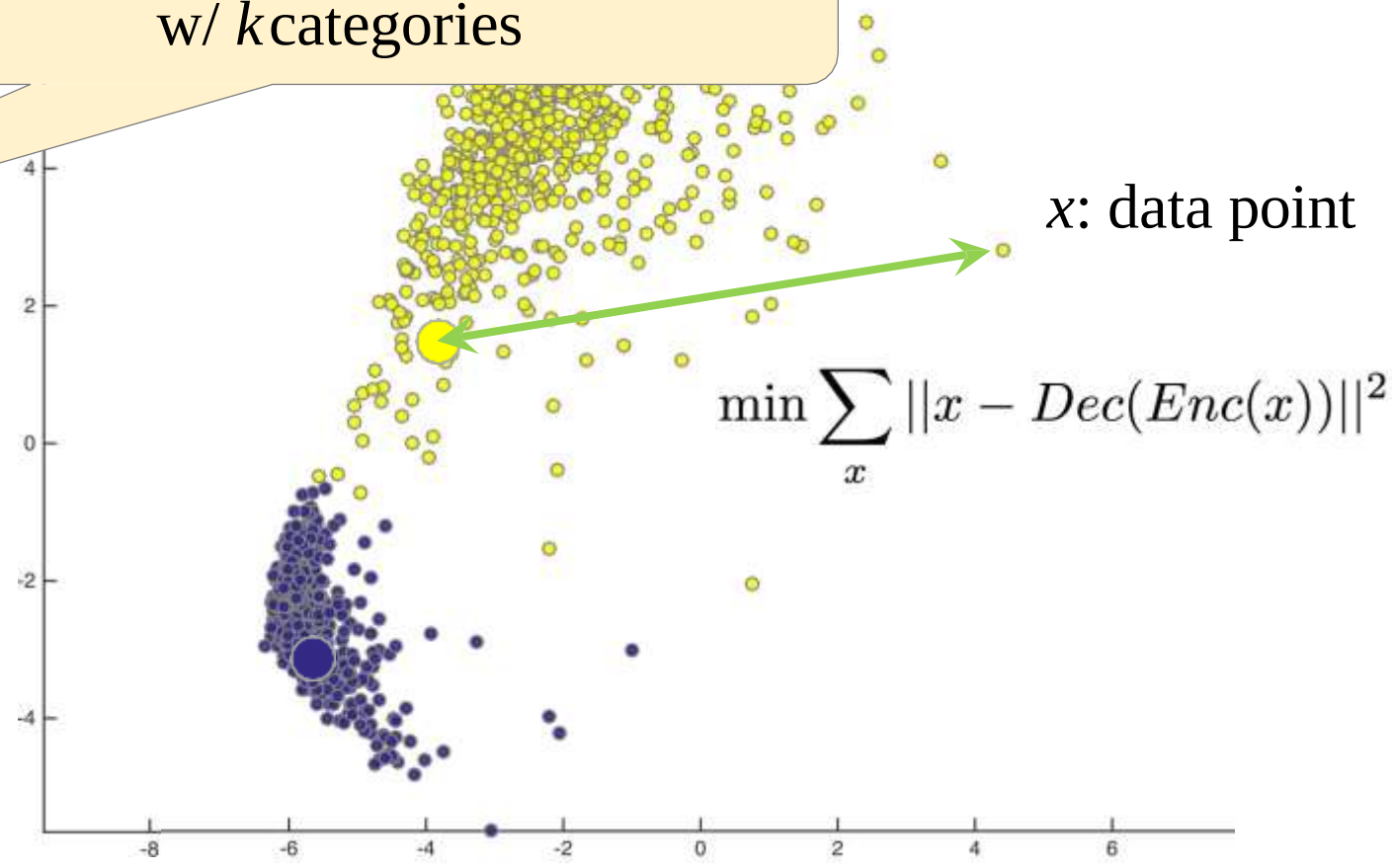


Expectation-Maximization (EM):

- often parametrize q analytically
- coordinate descent (i.e., alternating optimization)

K-means as **Autoencoder**

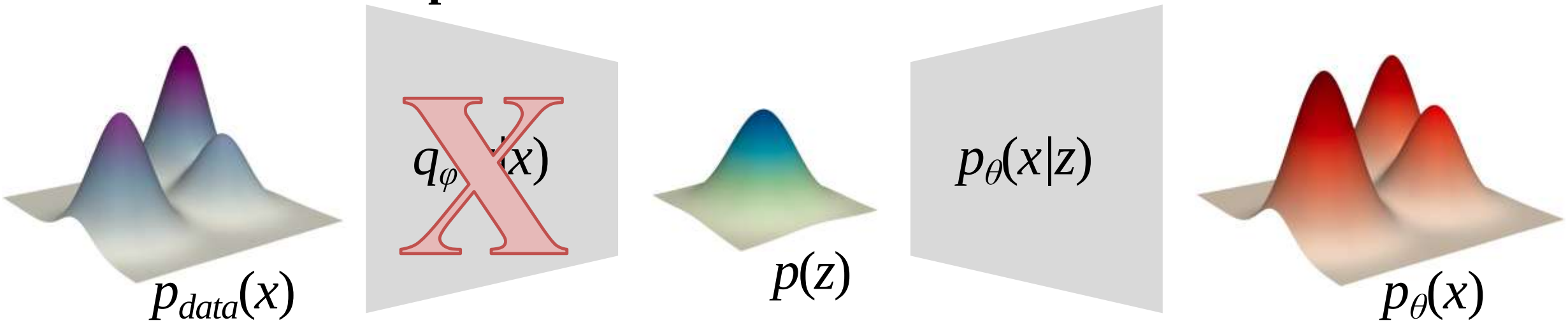




- encode: map x to one-hot
- decode: map one-hot to x'
- x' is a center

VAE

- encoder: maps data distribution to latent distribution
- decoder: maps latent distribution to data distribution
- **Posterior Collapse**



$$\mathcal{L}_{\theta,\phi}(x) = -\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \mathcal{D}_{\mathrm{KL}}\left(q_\phi(z|x)||p(z)\right)$$

Vector Quantized VAE

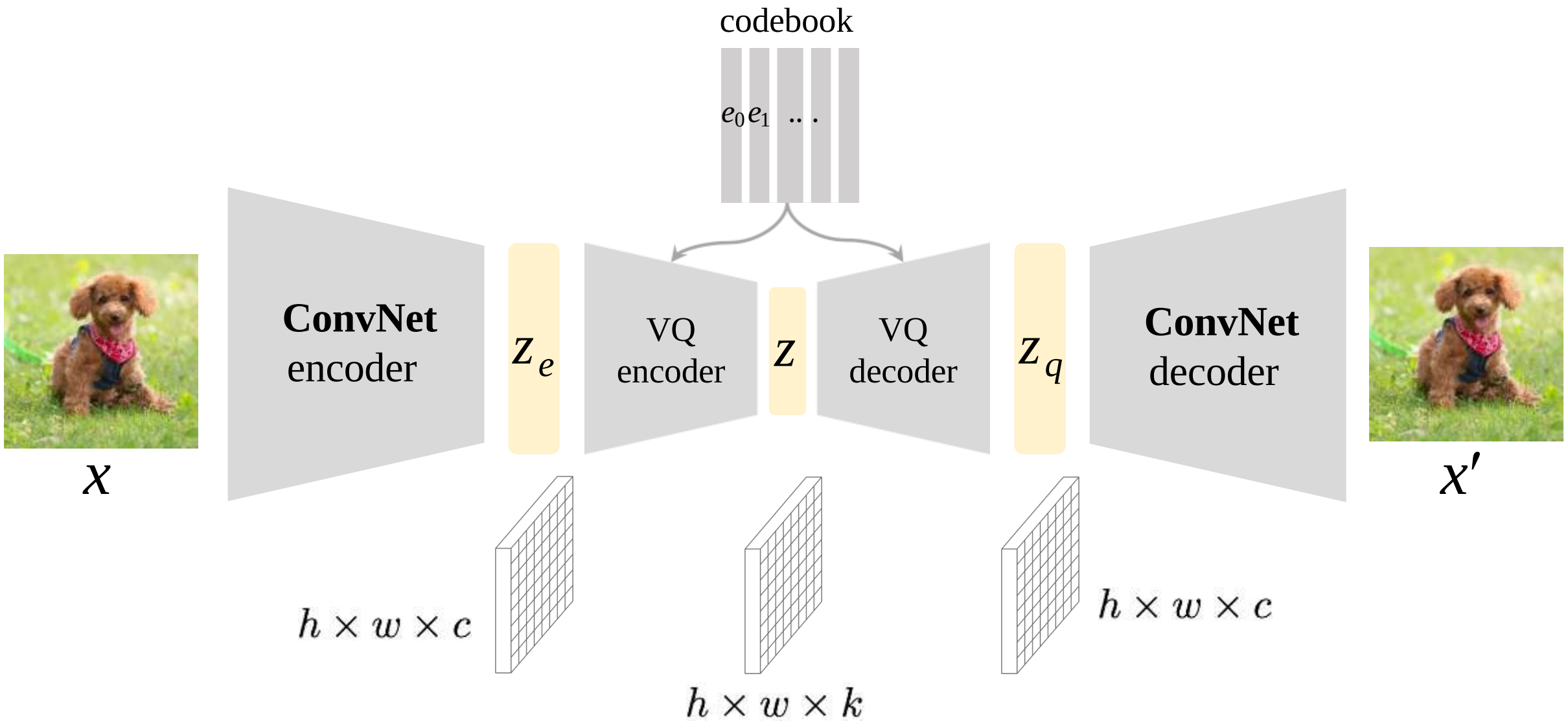
A single one-hot latent is not useful

- it's "deep K-means": with deep encoder/decoder
- a valid generative model; but not a "good" one

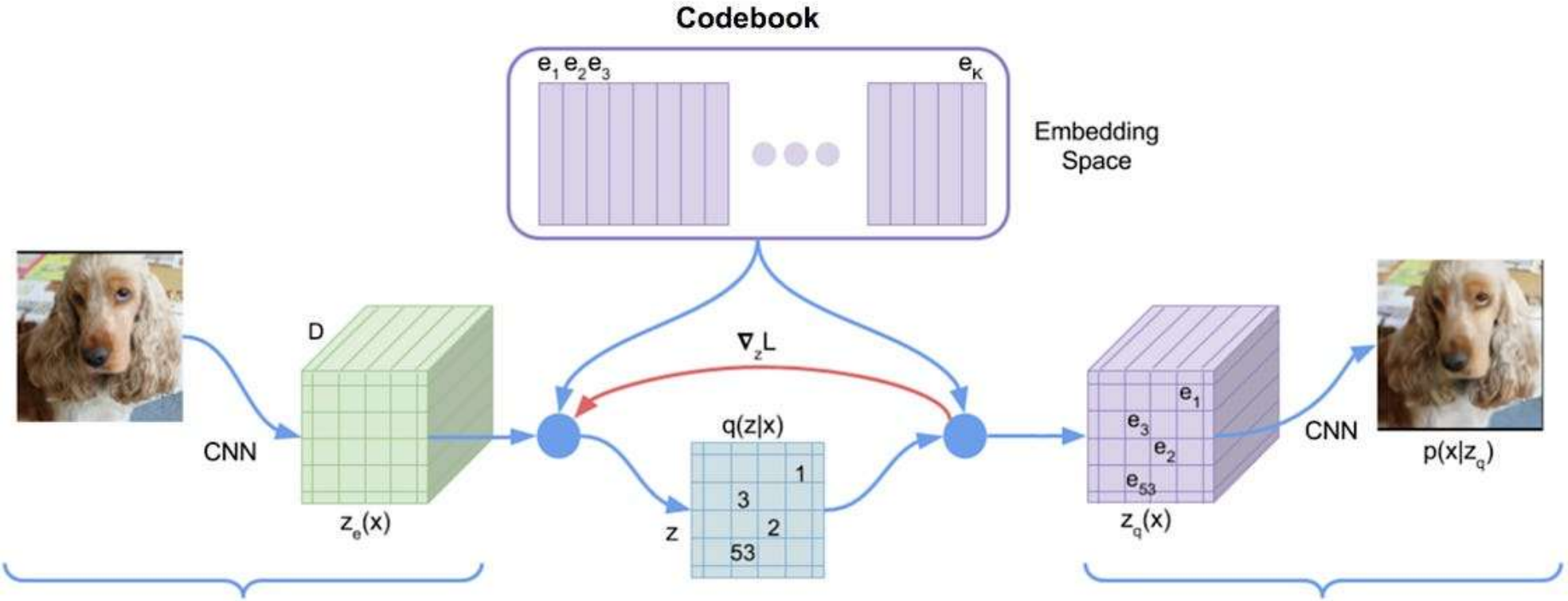
VQ-VAE: often used as "**tokenizers**"

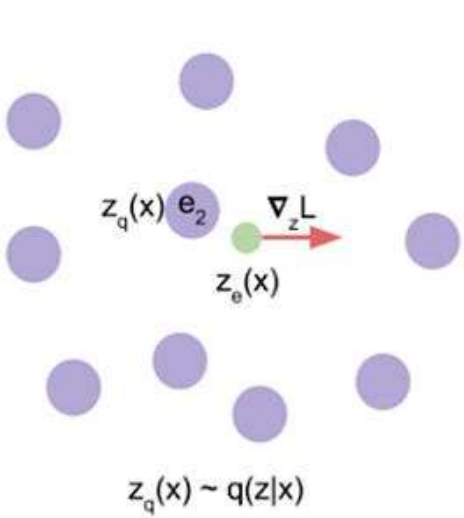
- output multiple one-hot vectors
- don't reduce latent spatial/temporal size to 1
- use ConvNet/Transformer as encoder and decoder

VQ-VAE as Tokenizers



VQ-VAE





Posterior categorical distribution:

$$q(\mathbf{z}=\mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$

Notes

- Both VAE and VQ-VAE can be “tokenizers” (produce spatial latents).

But:

- prior $p(z)$ only models per-token (per-location) distribution
- prior $p(z)$ doesn’t model **joint** distribution across tokens
- spatial tokens are not **independent**
- at inference, we can’t sample from **i.i.d.** prior $p(z)$

Next: modeling joint distribution:

$$p(z) = p(z_1)p(z_2|z_1)p(z_3|z_1, z_2)p(z_4|z_1, z_2, z_3)\dots$$

- Autoregressive models
- Masked models
- Diffusion models

Conditional Distribution Modeling

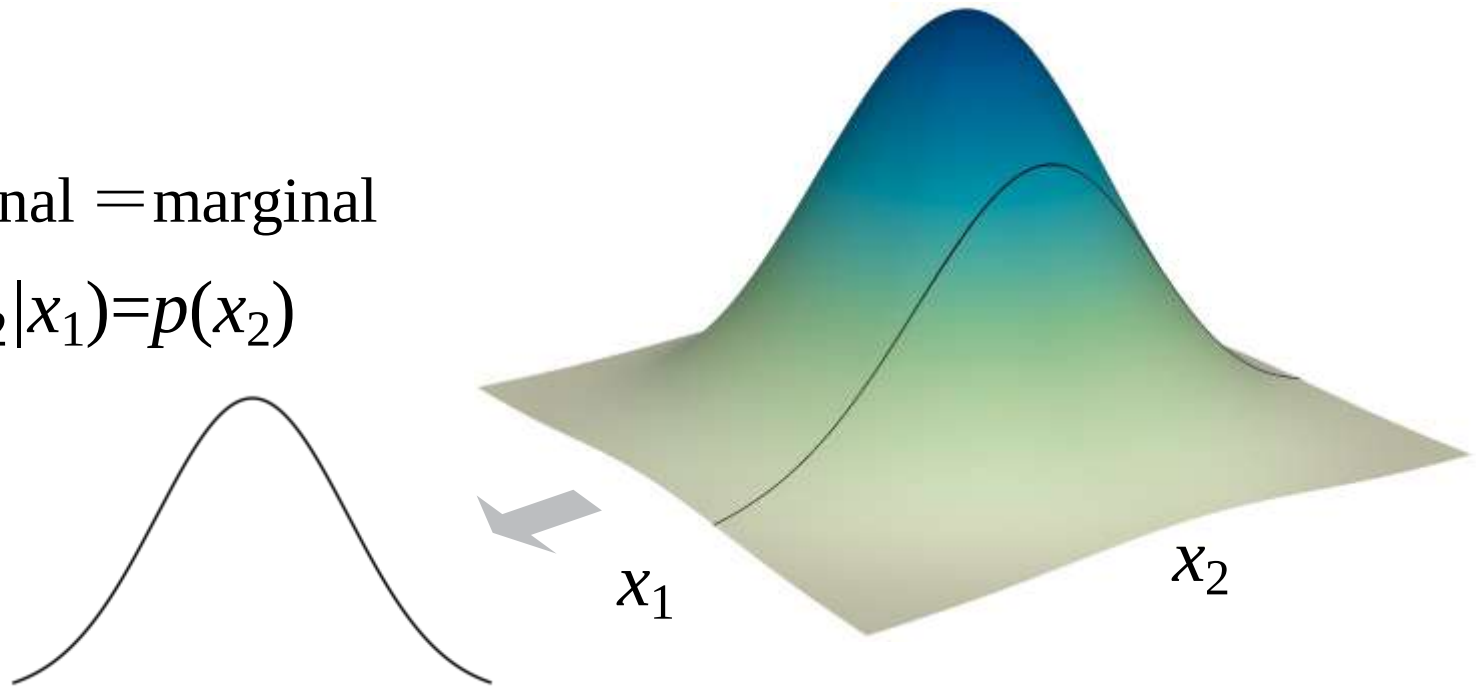
Joint Distribution

It’s convenient to model joint distributions by **independent** distributions

$$p(x_1, x_2)=p(x_1)p(x_2)$$

conditional ＝marginal

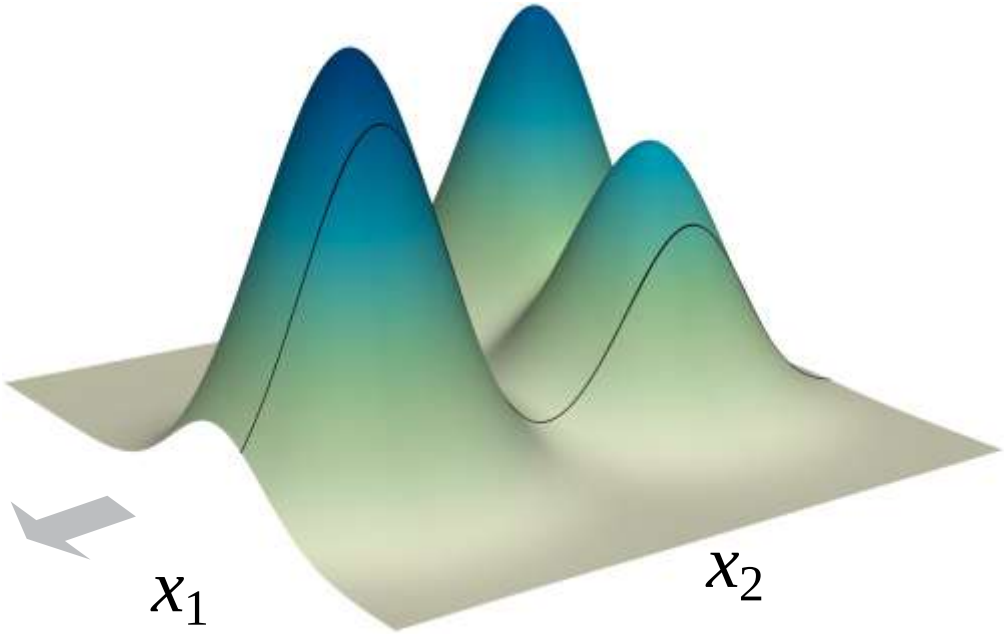
$$p(x_2|x_1)=p(x_2)$$



Joint Distribution

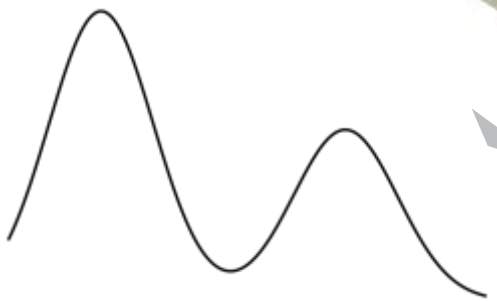
Real-word problems always involve dependent variables

$p(x_1, x_2)$



conditional ≠ marginal

$p(x_2|x_1)$

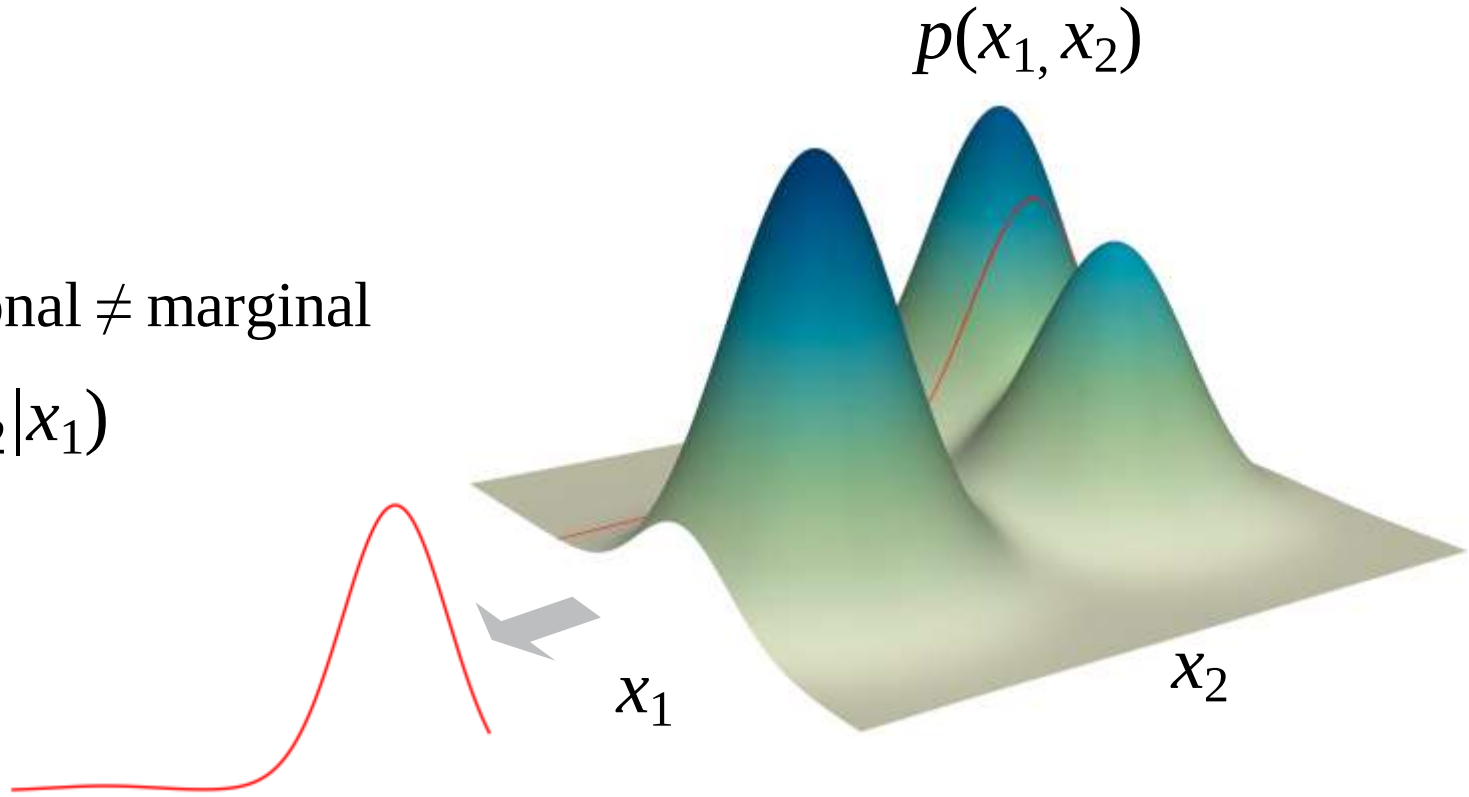


Joint Distribution

Real-word problems always involve dependent variables

conditional ≠ marginal

$p(x_2|x_1)$



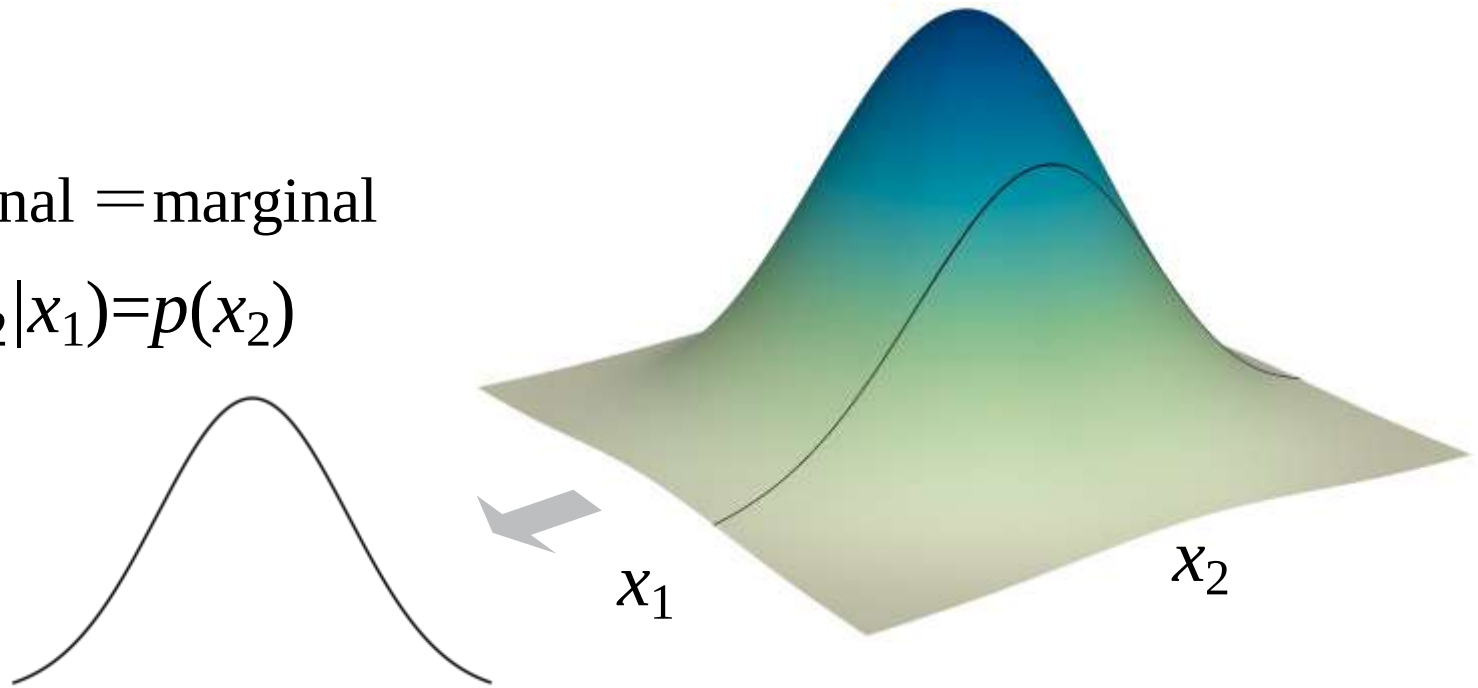
Identical Independent Distribution

VAE models latent as **identical independent distributions**

$$p(x_1, x_2)=p(x_1)p(x_2)$$

conditional ＝marginal

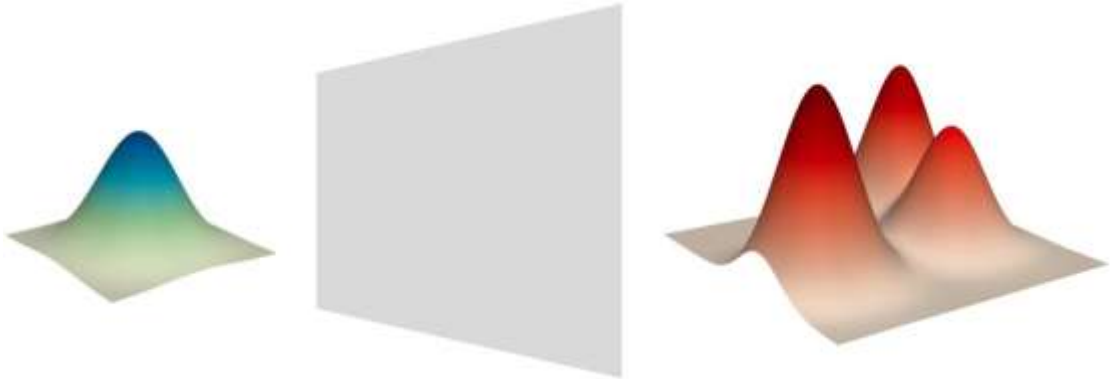
$$p(x_2|x_1)=p(x_2)$$



VAE

Modeling by **independent** latents

- mapping: independent ⇒ dependent
- strict assumption for **high-dim** data (e.g., 32x32x3 pixels)
- often with **low-dim** latents
- a good building block, but often not sufficient



independent

dependent

VAE results on 784-d MNIST data

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

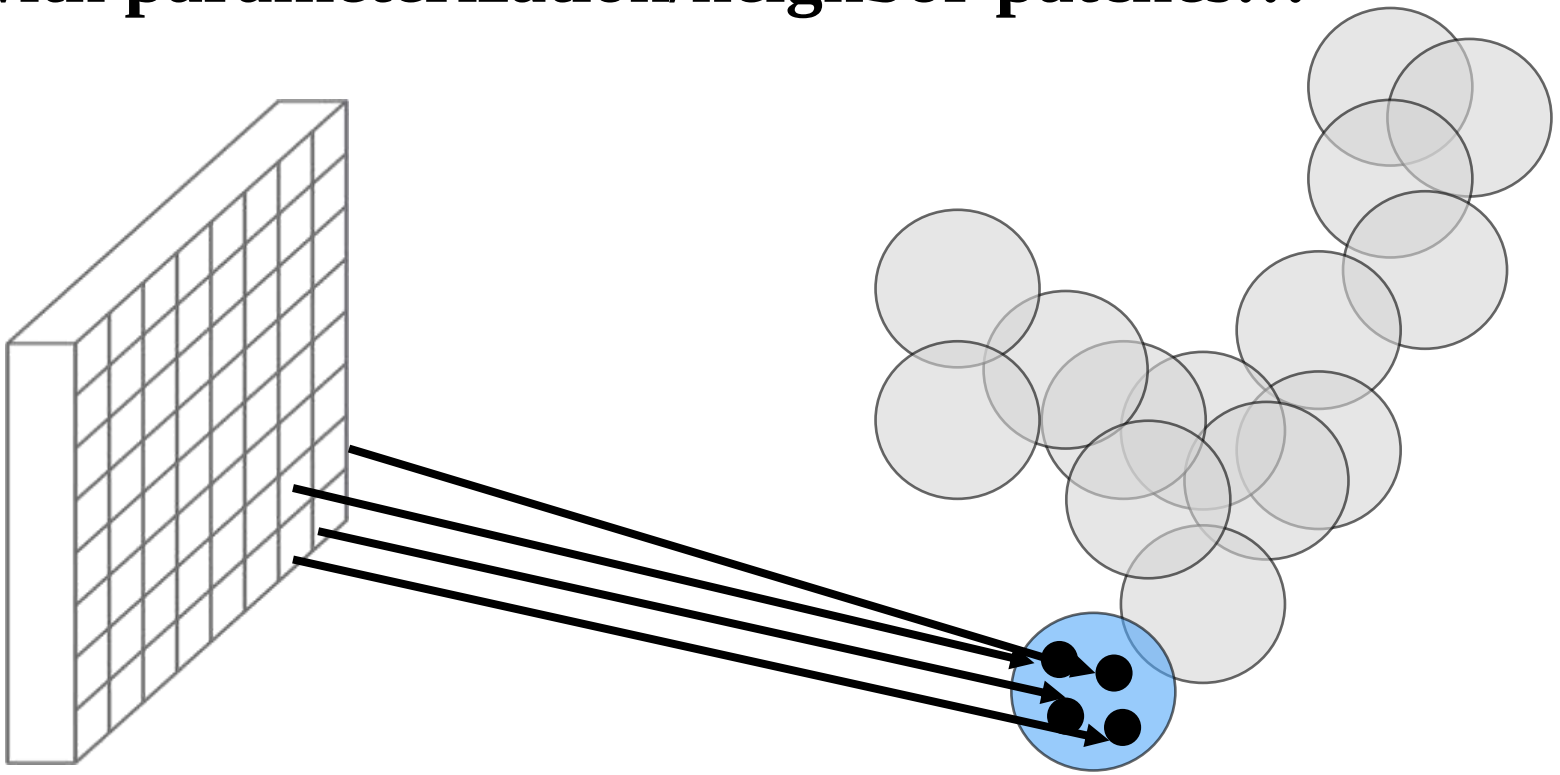
Too strict to model the 784-d (28x28) joint distribution by independent distributions

Figure from: Kingma and Welling. “Auto-Encoding Variational Bayes”, ICLR 2014

Sampling of VAE

VAE models latent as **identical independent distributions.**

But with parameterization/neighbor patches…



Dependency of Samples

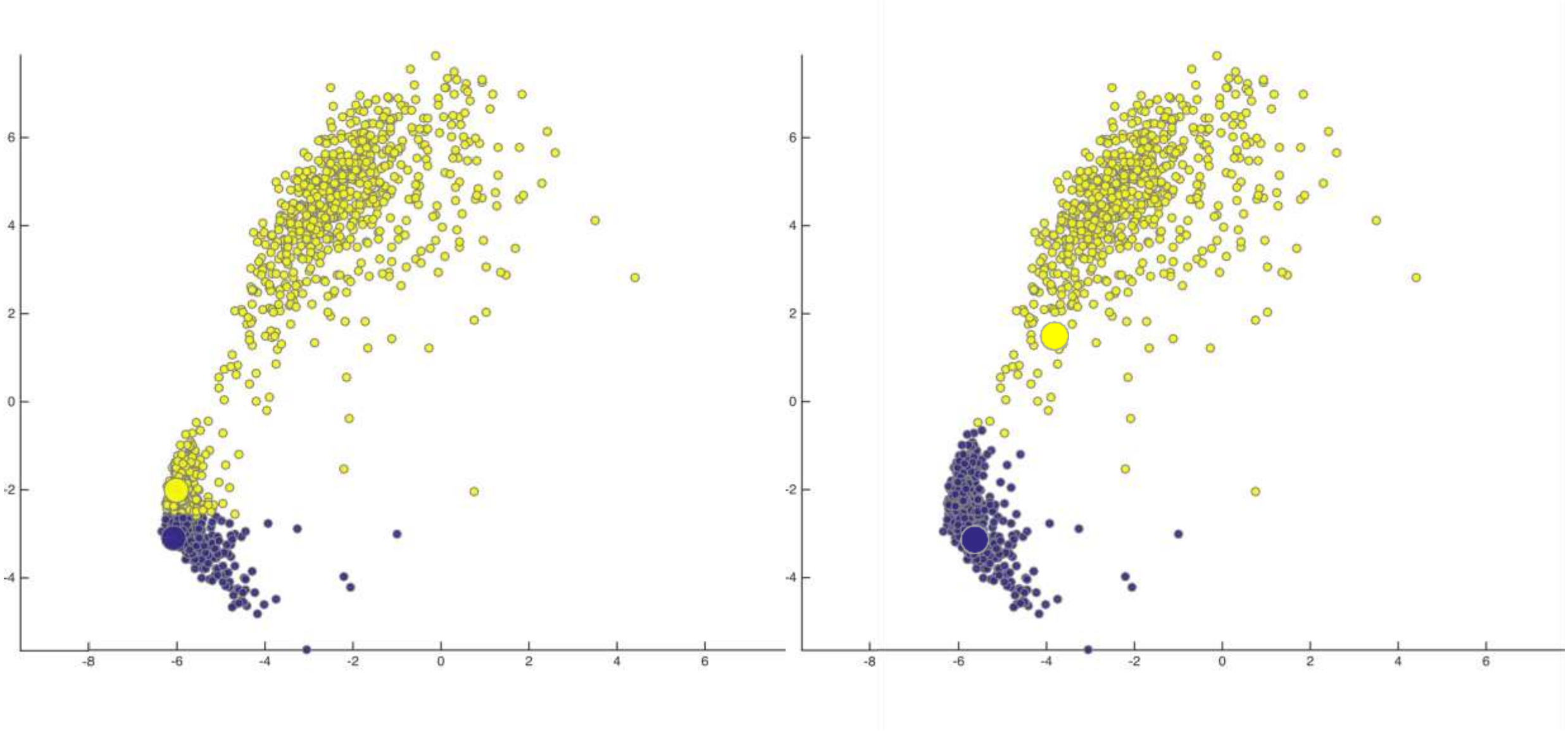
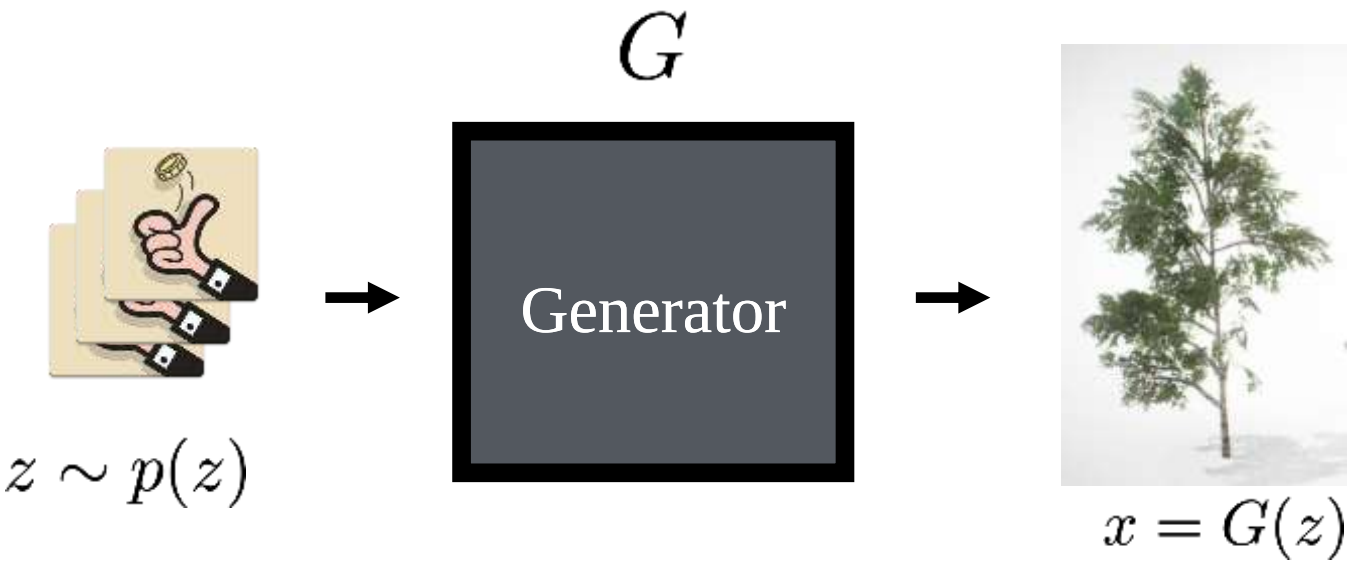


Figure adapted from: Marion Neumann, https://www.cse.wustl.edu/~m.neumann/sp2016/cse517/lecturenotes/lecturenote20.html

Recap



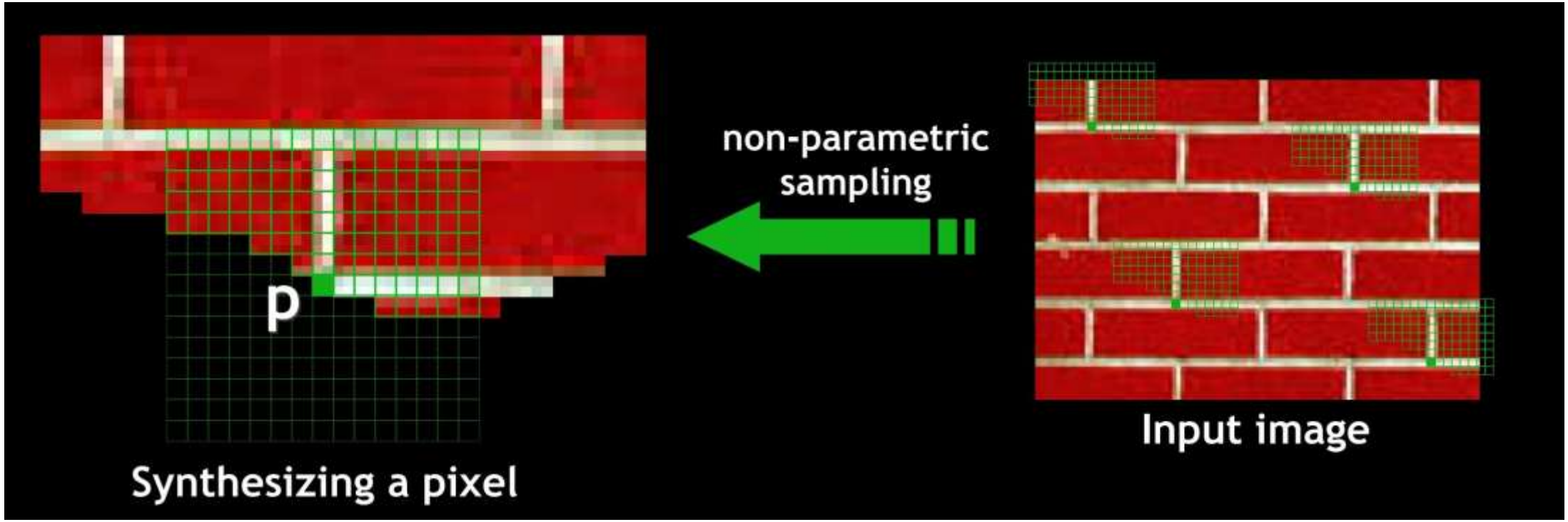
Sampler

$$G : \mathcal{Z} \to \mathcal{X}$$

$$z \sim p(z)$$

$$x = G(z)$$

Recap



Texture Synthesis by Non-parametric Sampling, ICCV 1999

Notes

- Both VAE and VQ-VAE can be “tokenizers” (produce spatial latents).

But:

- prior $p(z)$ only models per-token (per-location) distribution
- prior $p(z)$ doesn’t model **joint** distribution across tokens
- spatial tokens are not **independent**
- at inference, we can’t sample from **i.i.d.** prior $p(z)$

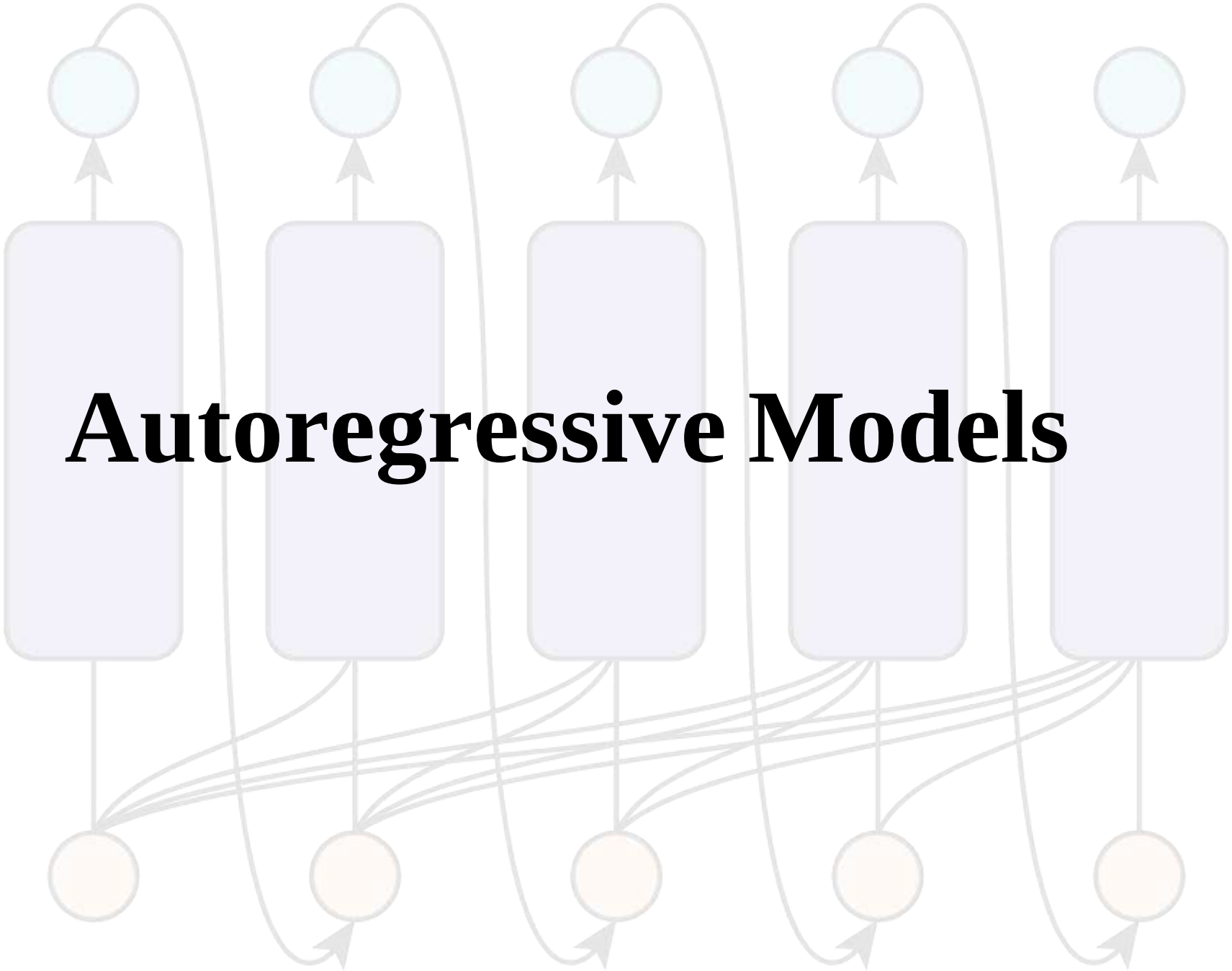
Next: modeling joint distribution:

$$p(z) = p(z_1)p(z_2|z_1)p(z_3|z_1, z_2)p(z_4|z_1, z_2, z_3)\dots$$

- Autoregressive models
- Masked models
- Diffusion models

Main References

- Kingma and Welling. “Auto-Encoding Variational Bayes”, ICLR 2014
- Neal and Hinton. “A view of the EM algorithm that justifies incremental, sparse, and other variants”, 1999
- Hastie, et al. “The Elements of Statistical Learning”, 2001
- van den Oord, et al. “Neural Discrete Representation Learning”, NeurIPS 2017



Autoregressive Models

Overview

- Conditional Distribution Modeling
- Autoregressive Models
- Network Architectures for Autoregressive Modeling

Conditional Distribution Modeling

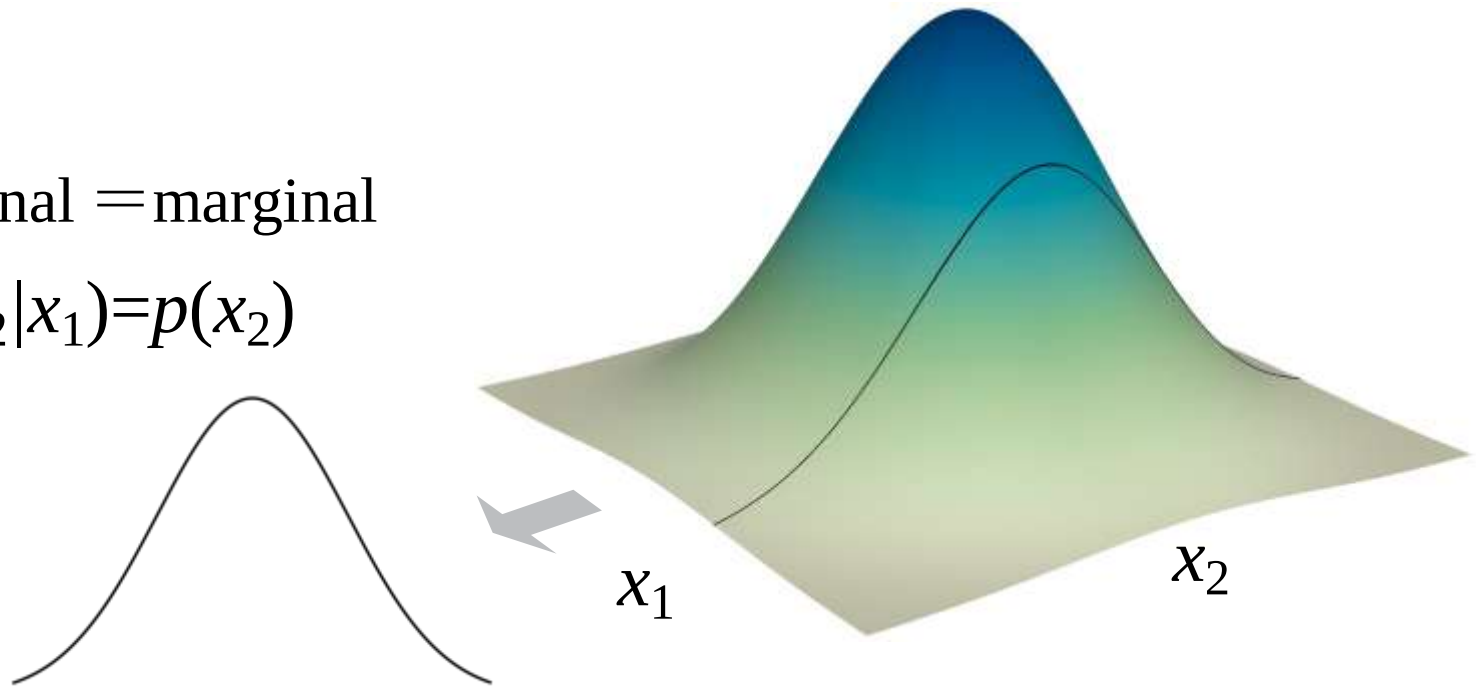
Joint Distribution

It’s convenient to model joint distributions by **independent** distributions

$$p(x_1, x_2)=p(x_1)p(x_2)$$

conditional ＝marginal

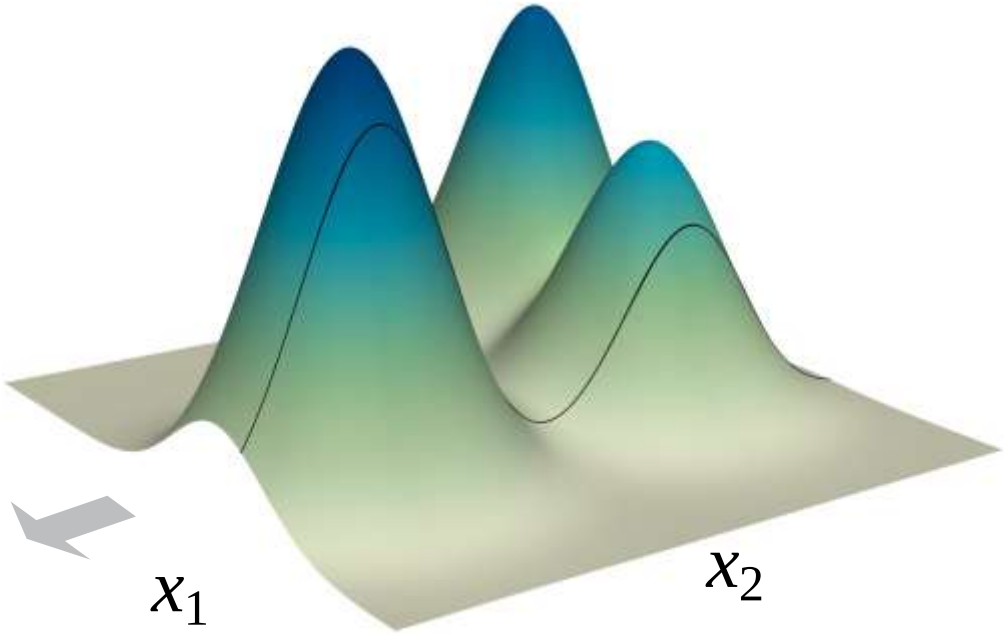
$$p(x_2|x_1)=p(x_2)$$



Joint Distribution

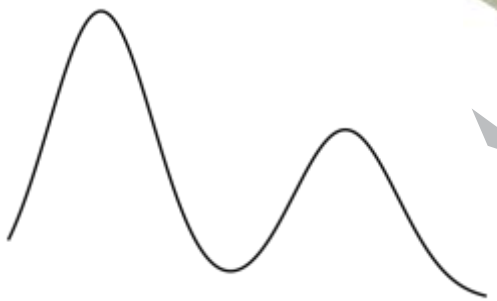
Real-word problems always involve dependent variables

$p(x_1, x_2)$



conditional ≠ marginal

$p(x_2|x_1)$

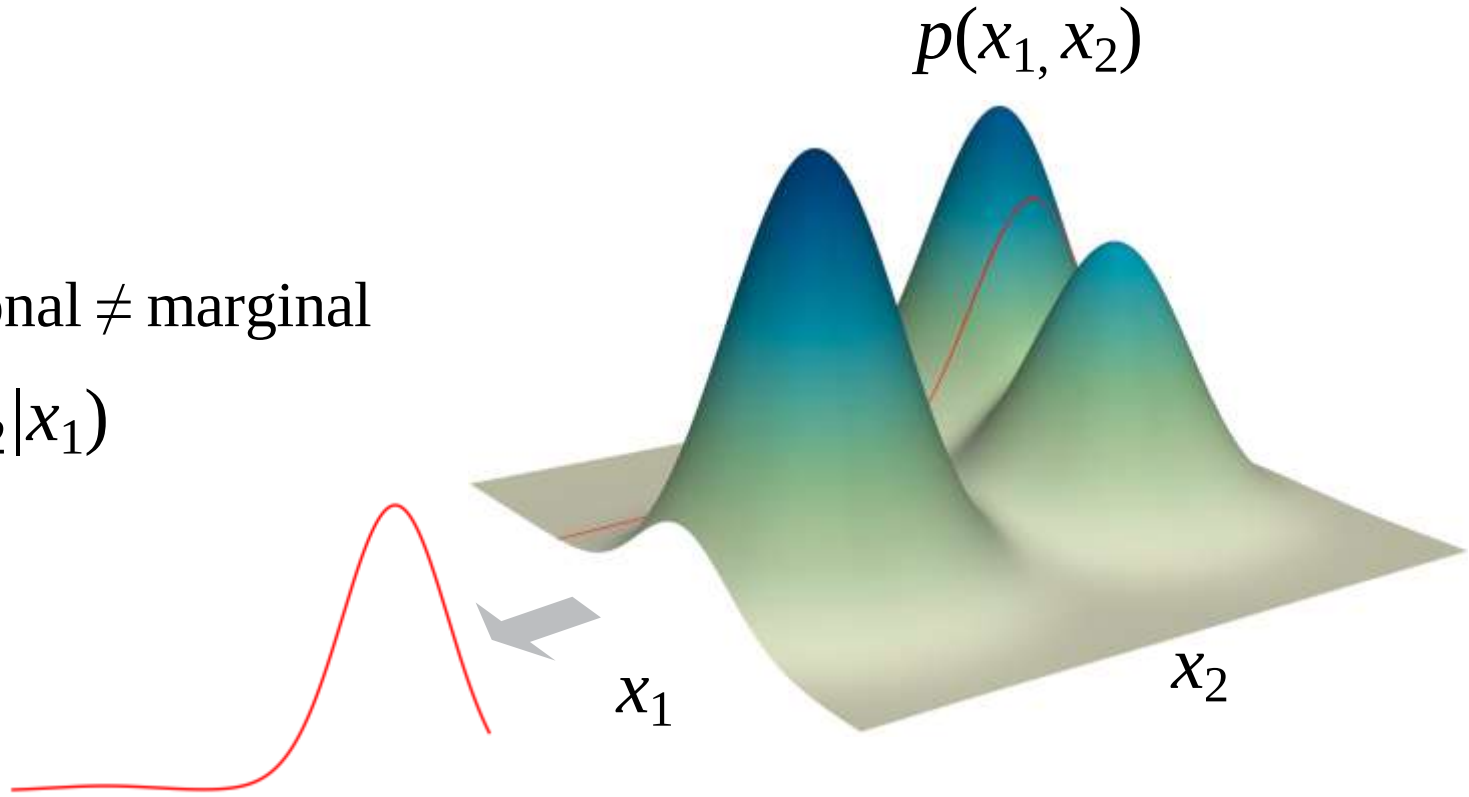


Joint Distribution

Real-word problems always involve dependent variables

conditional ≠ marginal

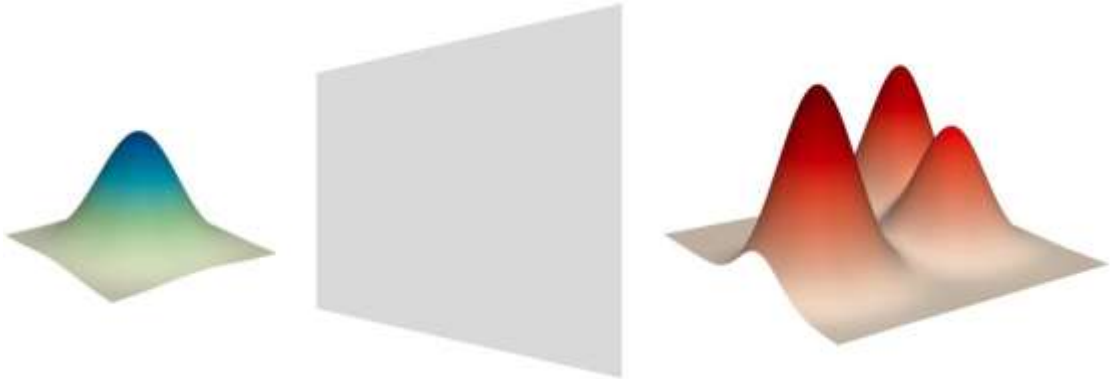
$p(x_2|x_1)$



How to Model Joint Distributions?

Solution 1: Modeling by **independent** latents (e.g., VAE)

- mapping: independent ⇒ dependent
- strict assumption for **high-dim** data (e.g., 32x32x3 pixels)
- often with **low-dim** latents
- a good building block, but often not sufficient



independent

dependent

VAE results on 784-d MNIST data

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Too strict to model the 784-d (28x28) joint distribution by independent distributions

Figure from: Kingma and Welling. “Auto-Encoding Variational Bayes”, ICLR 2014

How to Model Joint Distributions?

Solution 1: Modeling by **independent** latents

Solution 2: Modeling by **conditional** distributions

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

$$p(A, B) = p(A)p(B \mid A)$$

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

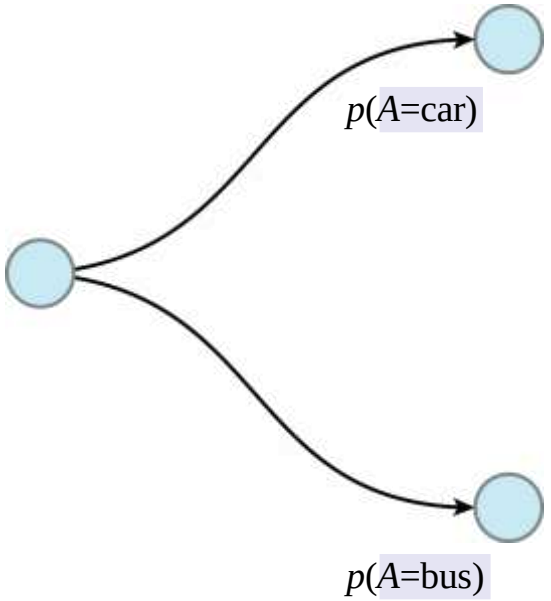
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



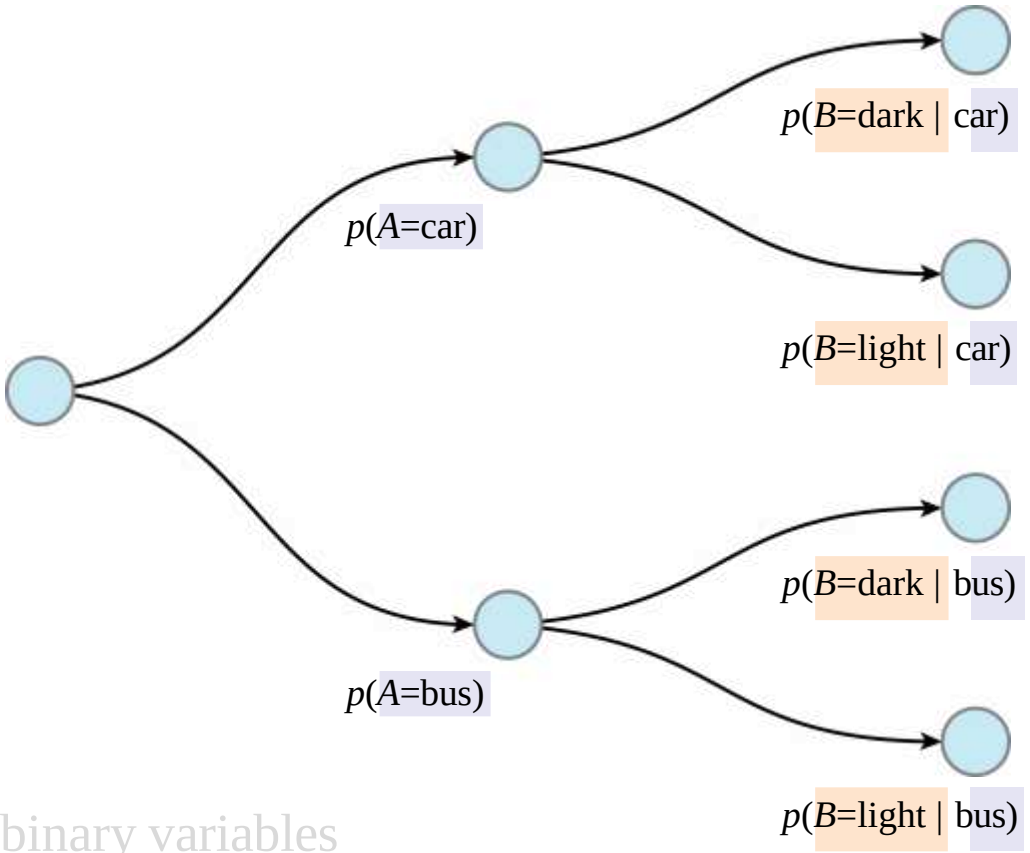
*example: binary variables

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



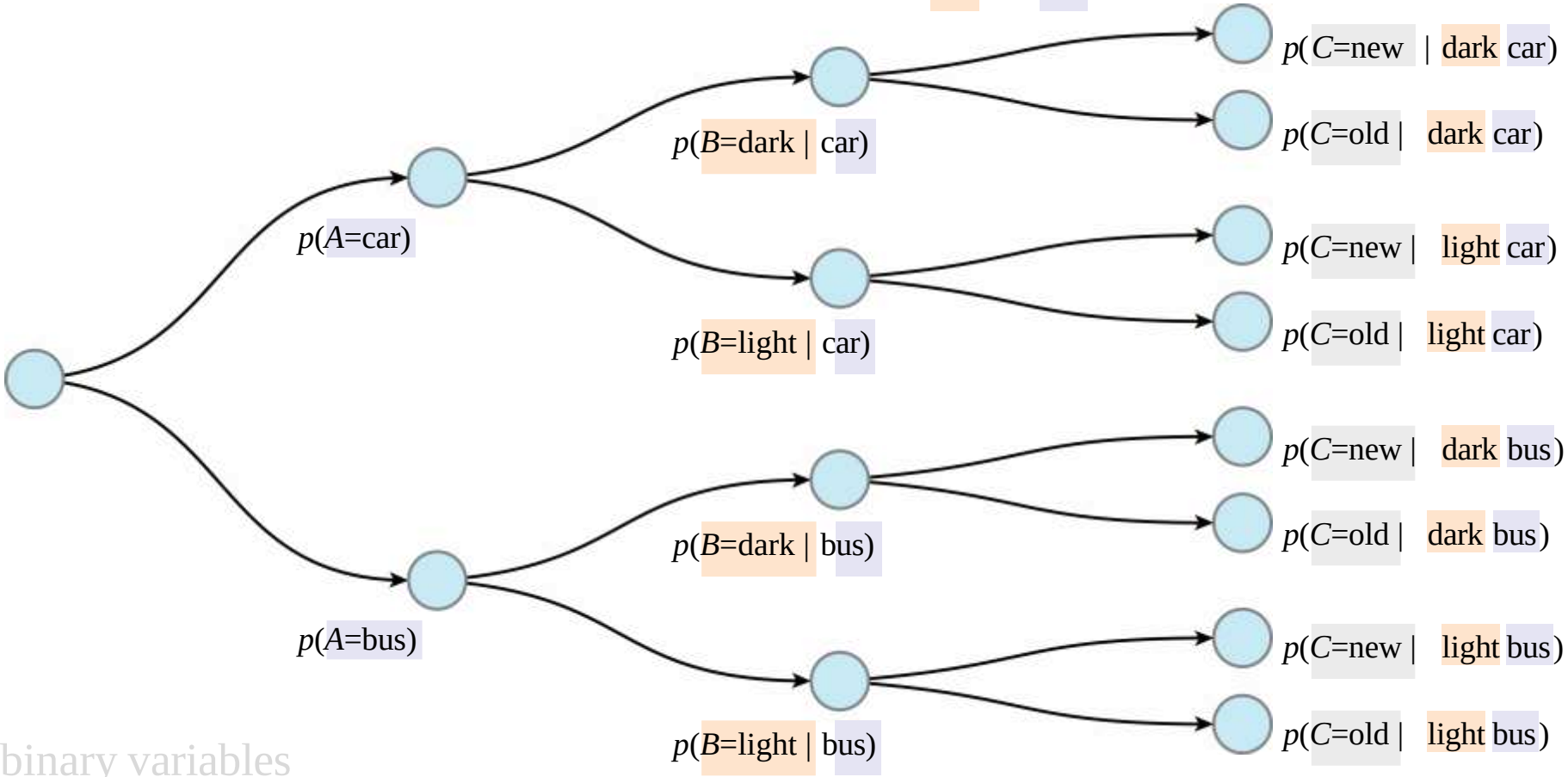
*example: binary variables

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



*example: binary variables

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

in any <u>order</u>:

$$
\begin{aligned}
p(A, B, C) &= p(A)p(B \mid A)p(C \mid A, B) \\
&= p(A)p(C \mid A)p(B \mid A, C) \\
&= p(B)p(A \mid B)p(C \mid A, B) \\
&= p(B)p(C \mid B)p(A \mid B, C) \\
&= p(C)p(A \mid C)p(B \mid A, C) \\
&= p(C)p(B \mid C)p(A \mid B, C)
\end{aligned}
$$

* This is a foundation of Masked Autoregressive (MAR) models:
Li, et al. Autoregressive Image Generation without Vector Quantization, 2024

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

in any <u>partition</u>:

$$p(A, B, C, D) = p(A, B)p(C, D \mid A, B)$$
$$= p(C, D)p(A, B \mid C, D)$$
$$= p(A, B, C)p(D \mid A, B, C)$$
$$= \ldots$$

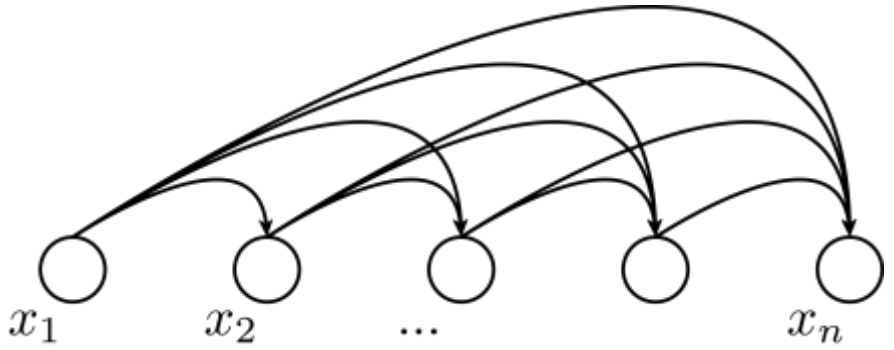
* This is a foundation of Masked Autoregressive (MAR) models:
Li, et al. Autoregressive Image Generation without Vector Quantization, 2024

Case Study: Conditional Distribution Modeling

Case 1: Partitioning the input representation space *x*

Example: Autoregressive Models on text tokens or pixels

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



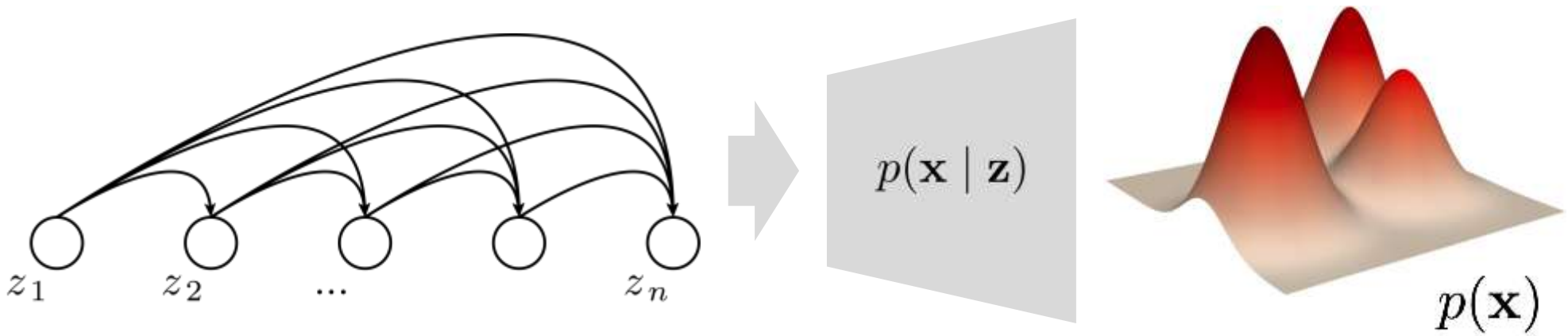
Case Study: Conditional Distribution Modeling

Case 2: Partitioning the latent representation space *z*

Example: Autoregressive Models on VQ-VAE tokens

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})$$

$$\text{with} \quad p(\mathbf{z}) = p(z_1)p(z_2 \mid z_1)...p(z_n \mid z_1, z_2, ..., z_{n-1})$$

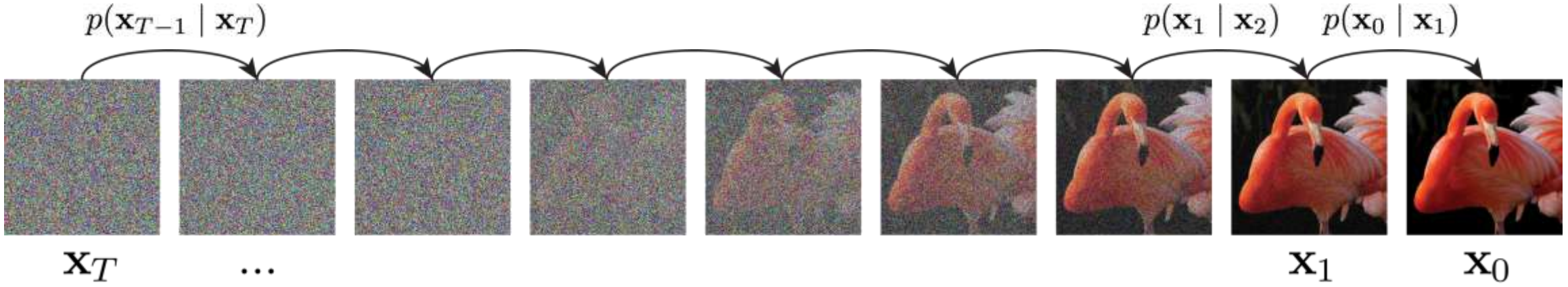


Case Study: Conditional Distribution Modeling

Case 3: Progressively transforming data distributions

Example: Diffusion Models

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)p(\mathbf{x}_{T-1} \mid \mathbf{x}_T)...p(\mathbf{x}_1 \mid \mathbf{x}_2)p(\mathbf{x}_0 \mid \mathbf{x}_1)$$



Conditional Distribution Modeling

Same spirit as Deep Learning: "**Divide-and-Conquer**"

- Chain rule of **derivatives** (backprop):

$$\frac{\partial \mathcal{E}}{\partial x_1} = \frac{\partial \mathcal{E}}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial x_1}$$

- Chain rule of **probability**:

$$p(x_1, x_2, x_3) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2)$$

Conditional Distribution Modeling

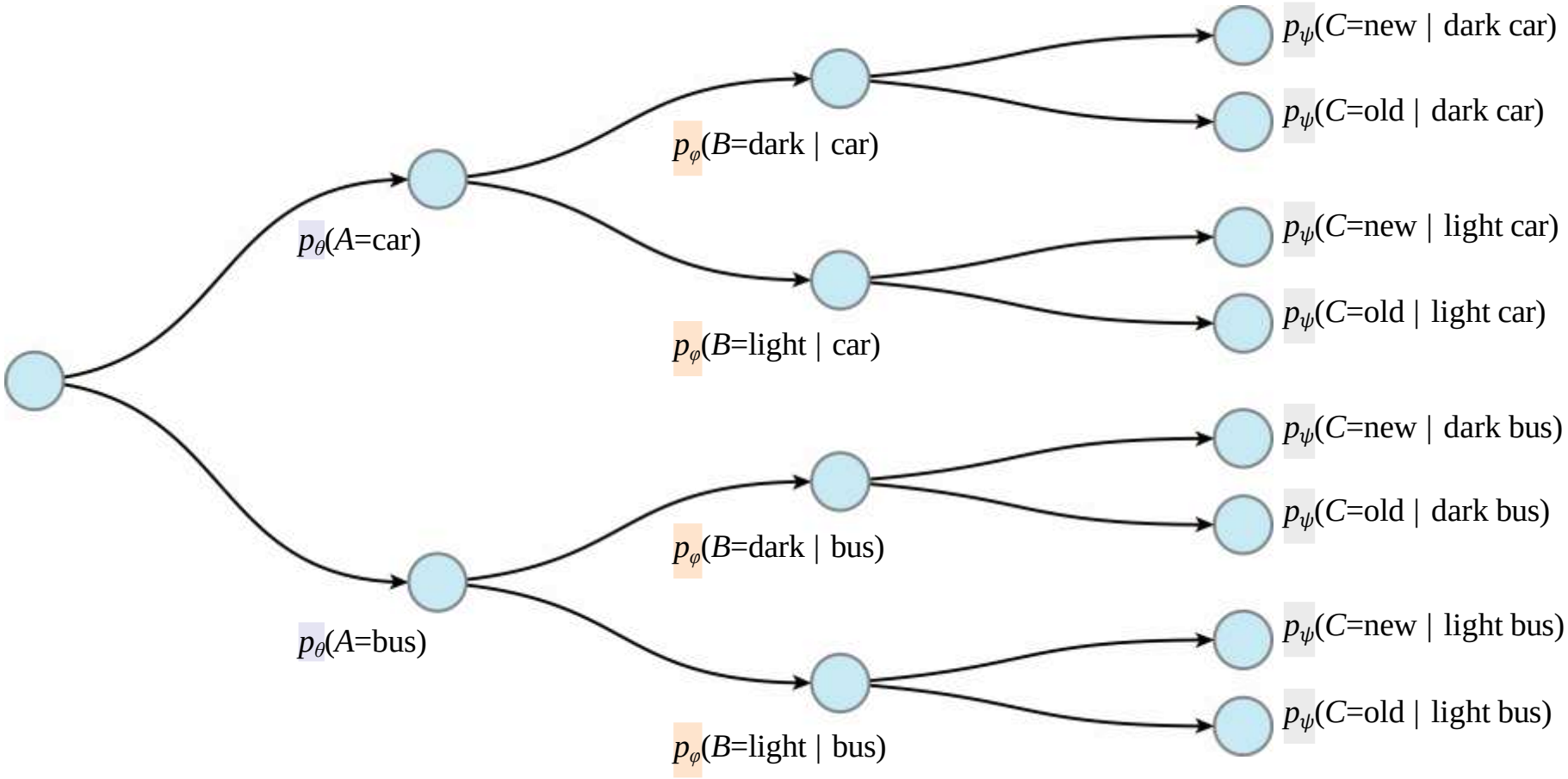
Modeling each conditional distribution with a neural network

$$p(A, B, C) = p_\theta(A)p_\varphi(B \mid A)p_\psi(C \mid A, B)$$

Conditional Distribution Modeling

Modeling each conditional distribution with a neural network

$$p(A, B, C) = p_\theta(A)p_\varphi(B \mid A)p_\psi(C \mid A, B)$$



Conditional Distribution Modeling

Modeling each conditional distribution with a neural network

$$p(A, B, C) = p_\theta(A)p_\varphi(B \mid A)p_\psi(C \mid A, B)$$

Note:

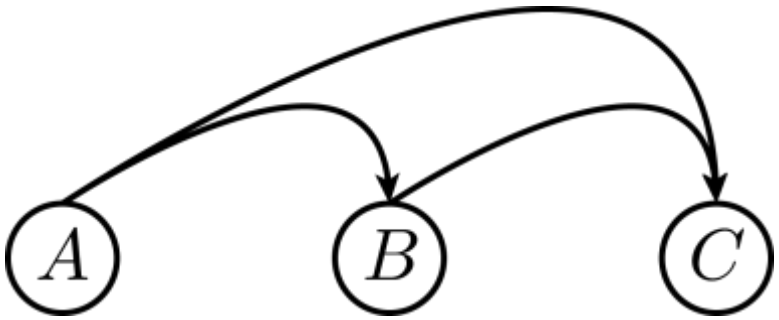
- parameterizing $p(A, B, C)$ vs. parameterizing $p(C \mid A, B)$?
  - $p(A, B, C)$ has 3 variables
  - $p(C \mid A, B)$ has 1 variable and 2 conditions (conditions are network inputs)

- weight sharing?
  - conceptually, each p has its own weights
  - weight sharing implies inductive biases (discussed later)

Dependency Graphs

- Decompose a joint distribution ⇒ induce a dependency graph
- Dependency graphs reflect prior knowledge

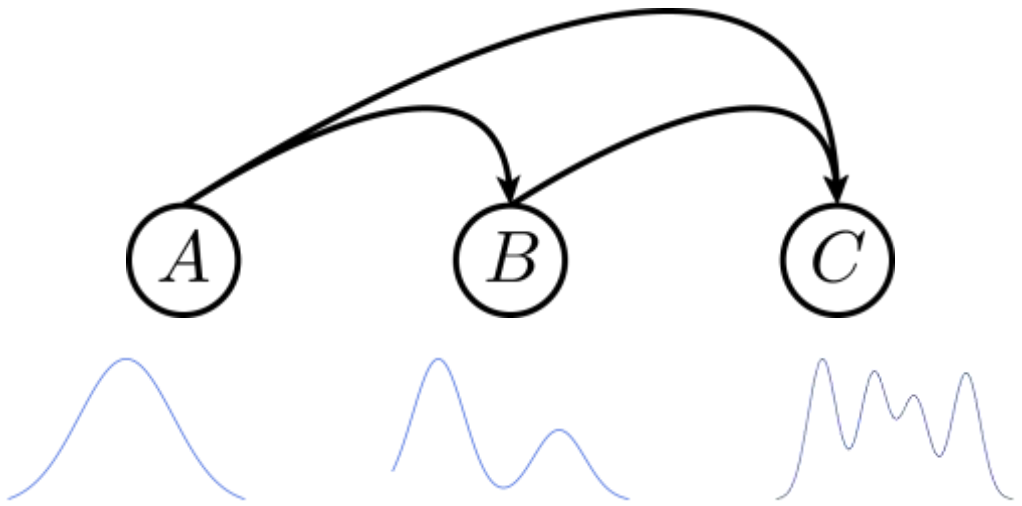
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



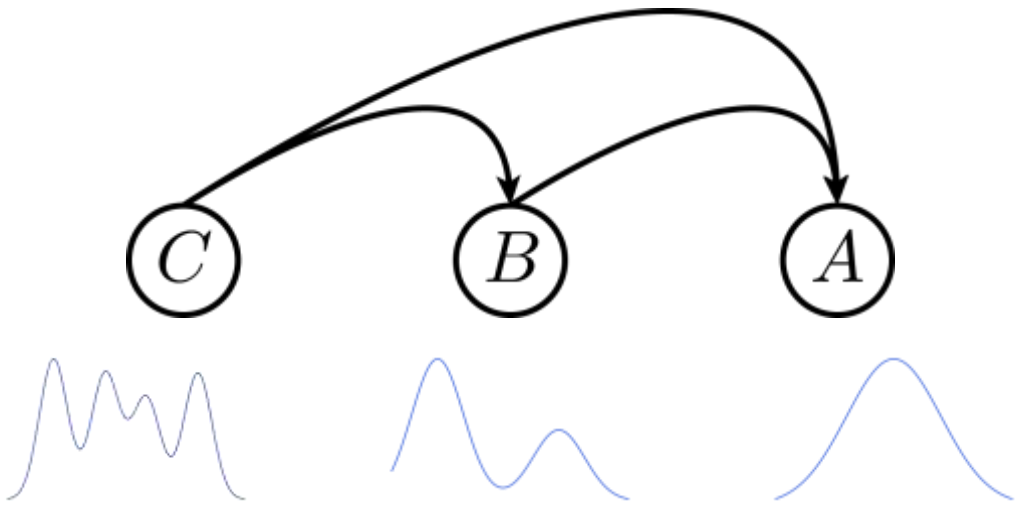
Dependency Graphs

- Some dependency graphs may induce **simpler** distributions ...

$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$



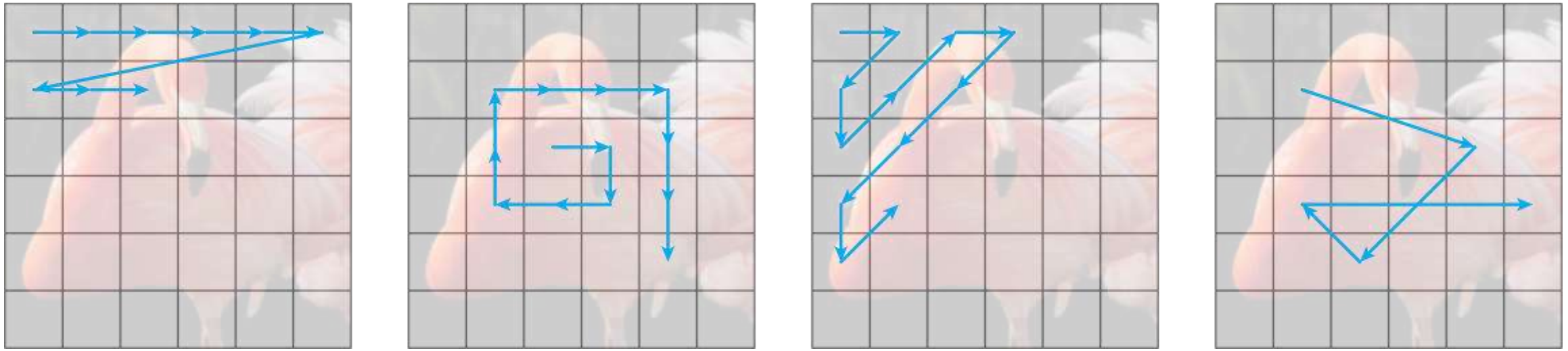
$p(A, B, C) = p(C)p(B \mid C)p(A \mid B, C)$



Both are valid formulations. But one may be simpler to learn than the other.

Dependency Graphs

- Some dependency graphs may induce **simpler** distributions ...



E.g., see: Hua, et al. “Self-supervision through Random Segments with Autoregressive Coding (RandSAC)”, ICLR 2022

Conditional Distribution Modeling

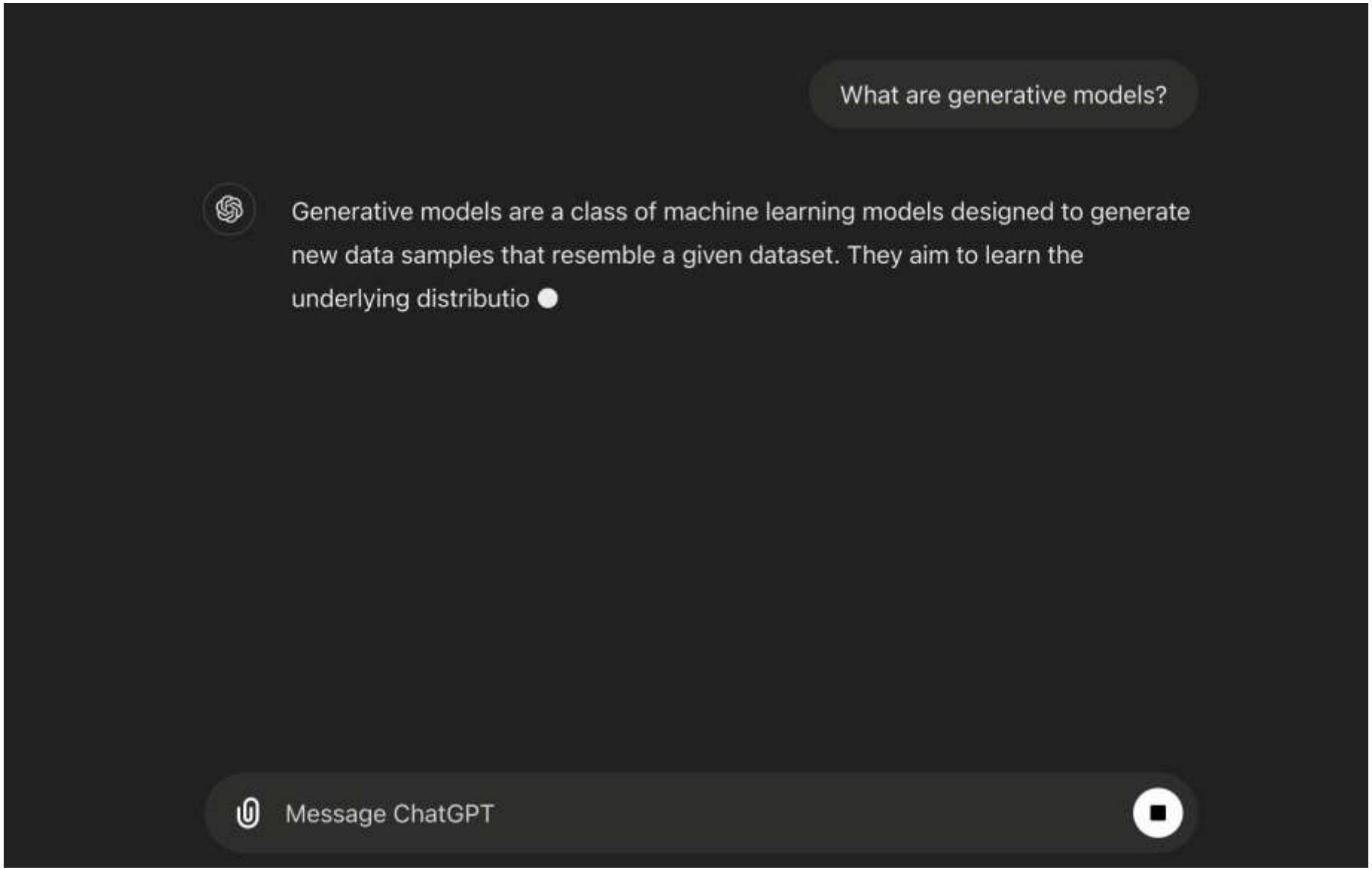
Summary:

- Joint distribution ⇒ product of conditionals
- Chain rule: divide-and-conquer
- Any order, any partition
- Dependency graphs: induce prior knowledge

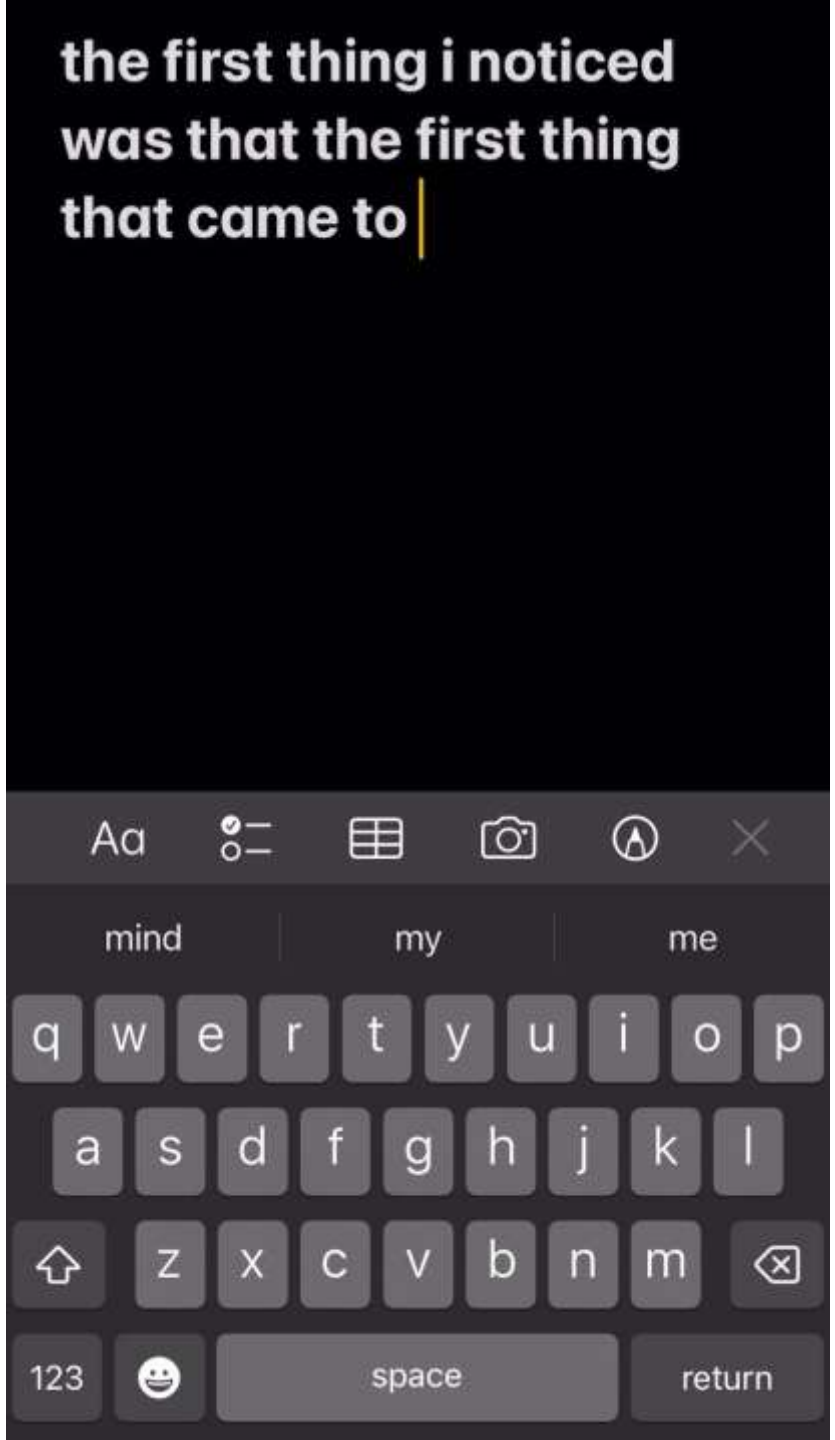
These are not specific to Autoregressive models.

Autoregressive Models

ChatGPT: Next Token Prediction



Your Keyboard



Auto + Regression

Auto: “self”

- using its “own” outputs as inputs for next perditions

Regression:

- estimating relationship between variables

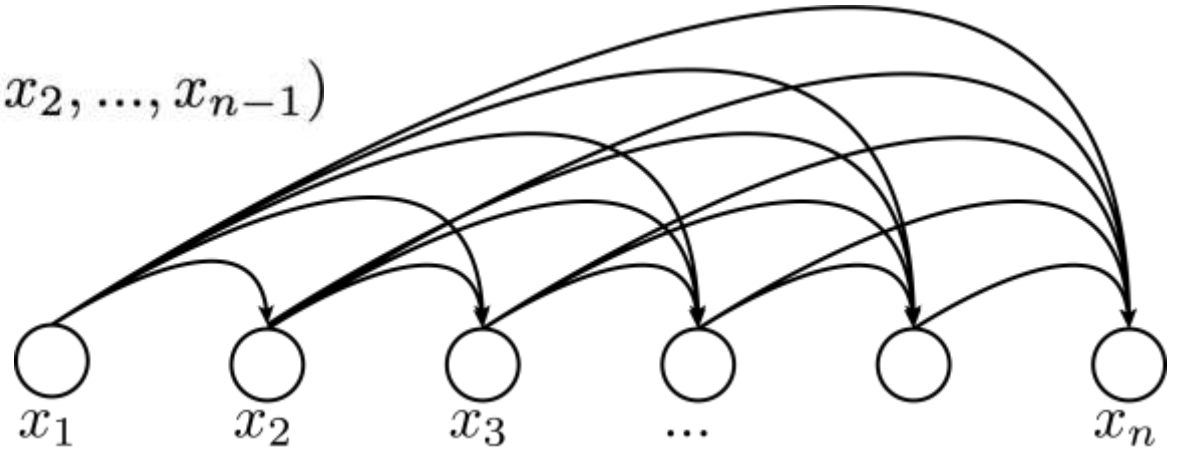
Note:

- “Autoregressive” implies an inference-time behavior
- Training-time is not necessarily autoregressive (e.g., teacher forcing)

Autoregressive Models

In general, **autoregression** is a way of modeling **joint** distribution by a product of **conditional** distributions:

$$p(x_1, x_2, ..., x_n) = p(x_1)p(x_2 \mid x_1)...p(x_n \mid x_1, x_2, ..., x_{n-1})$$
$$= \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$



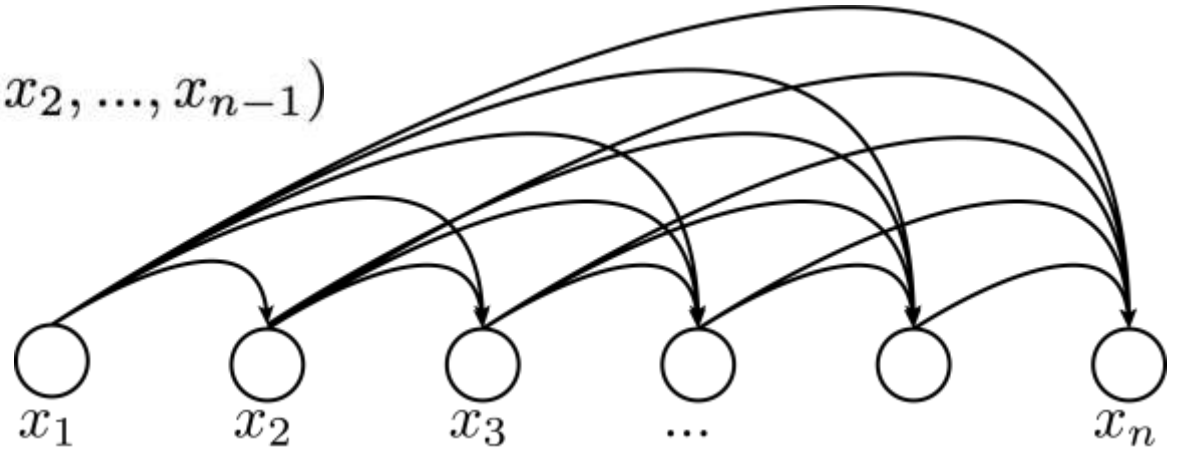
Conceptually, ...

- *x* can be **any** representation
  - not necessarily sequential/temporal
  - e.g., all dims of a vector
  - e.g., 2D, 3D, or high-dim arrays

Autoregressive Models

In general, **autoregression** is a way of modeling **joint** distribution by a product of **conditional** distributions:

$$
\begin{aligned}
p(x_1, x_2, ..., x_n) &= p(x_1)p(x_2 \mid x_1)...p(x_n \mid x_1, x_2, ..., x_{n-1}) \\
&= \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})
\end{aligned}
$$



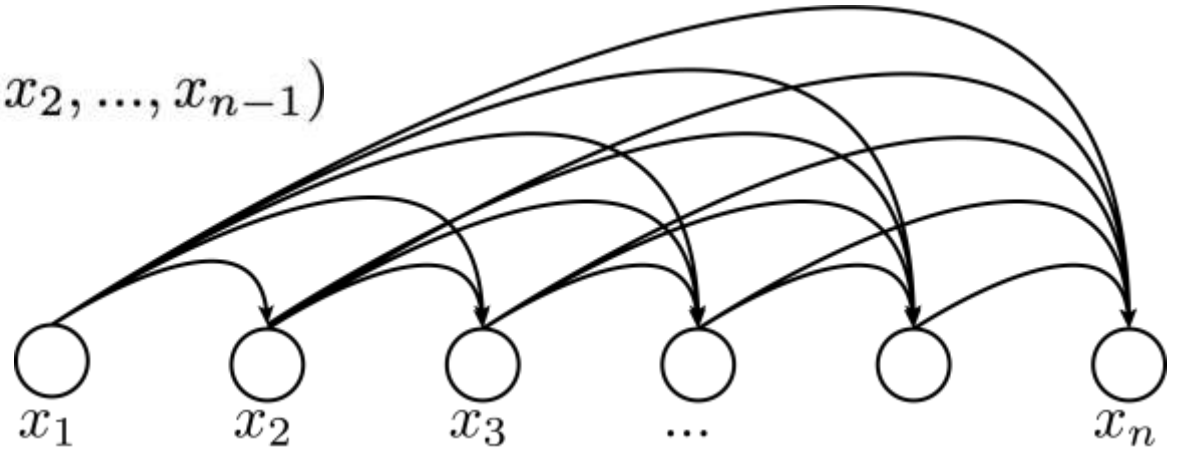
Conceptually, ...

- *x* can be **any** order and **any** partition
  - order: e.g., reverse order is valid
  - partition: e.g., each of x_i can be a scalar, vector, or tensor

Autoregressive Models

In general, **autoregression** is a way of modeling **joint** distribution by a product of **conditional** distributions:

$$
\begin{aligned}
p(x_1, x_2, ..., x_n) &= p(x_1)p(x_2 \mid x_1)...p(x_n \mid x_1, x_2, ..., x_{n-1}) \\
&= \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})
\end{aligned}
$$



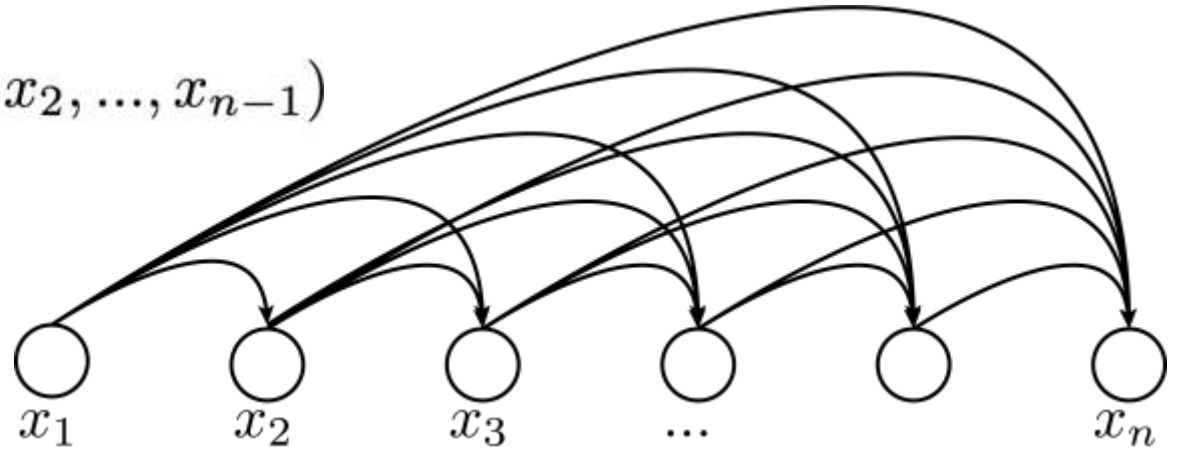
Conceptually, ...

- each $p(\cdot \mid \cdot)$ can take **any** form
  - e.g., look-up tables, trees, neural nets, or mix
  - e.g., discrete or continuous variables

Autoregressive Models

In general, **autoregression** is a way of modeling **joint** distribution by a product of **conditional** distributions:

$$
\begin{aligned}
p(x_1, x_2, ..., x_n) &= p(x_1)p(x_2 \mid x_1)...p(x_n \mid x_1, x_2, ..., x_{n-1}) \\
&= \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})
\end{aligned}
$$



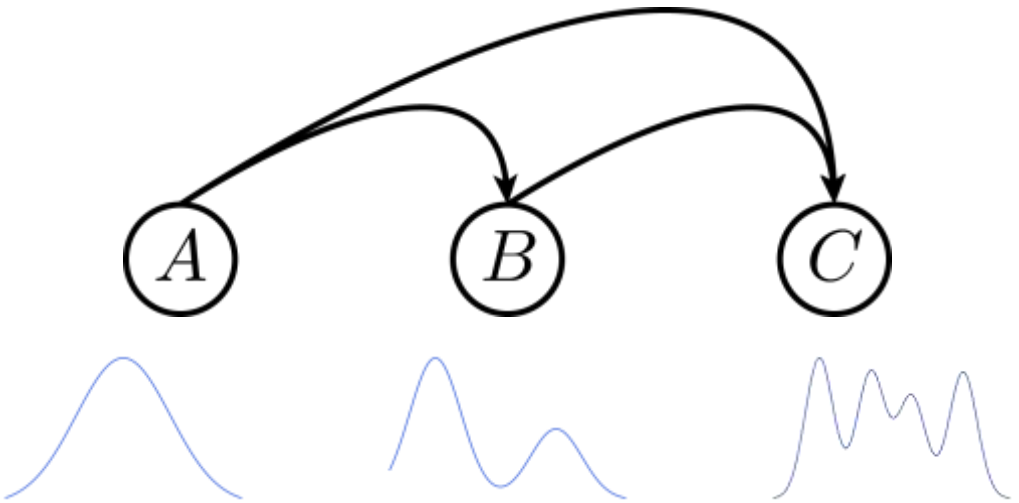
This formulation makes **no** compromise/approximation

- This decomposition is always valid (just chain rule)
- But some are easier to model: “inductive bias” ...

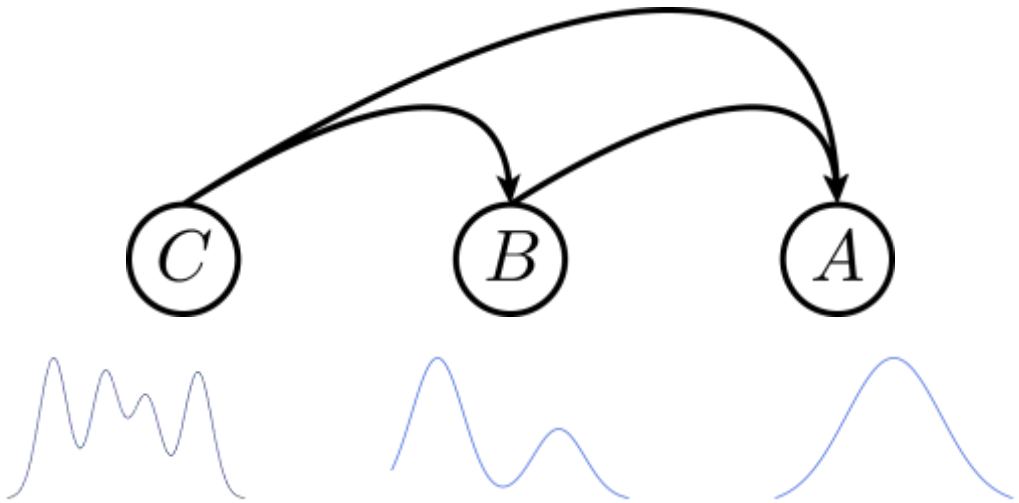
Inductive Bias

(Recap) We want the decomposition to give us **simpler** distributions ...

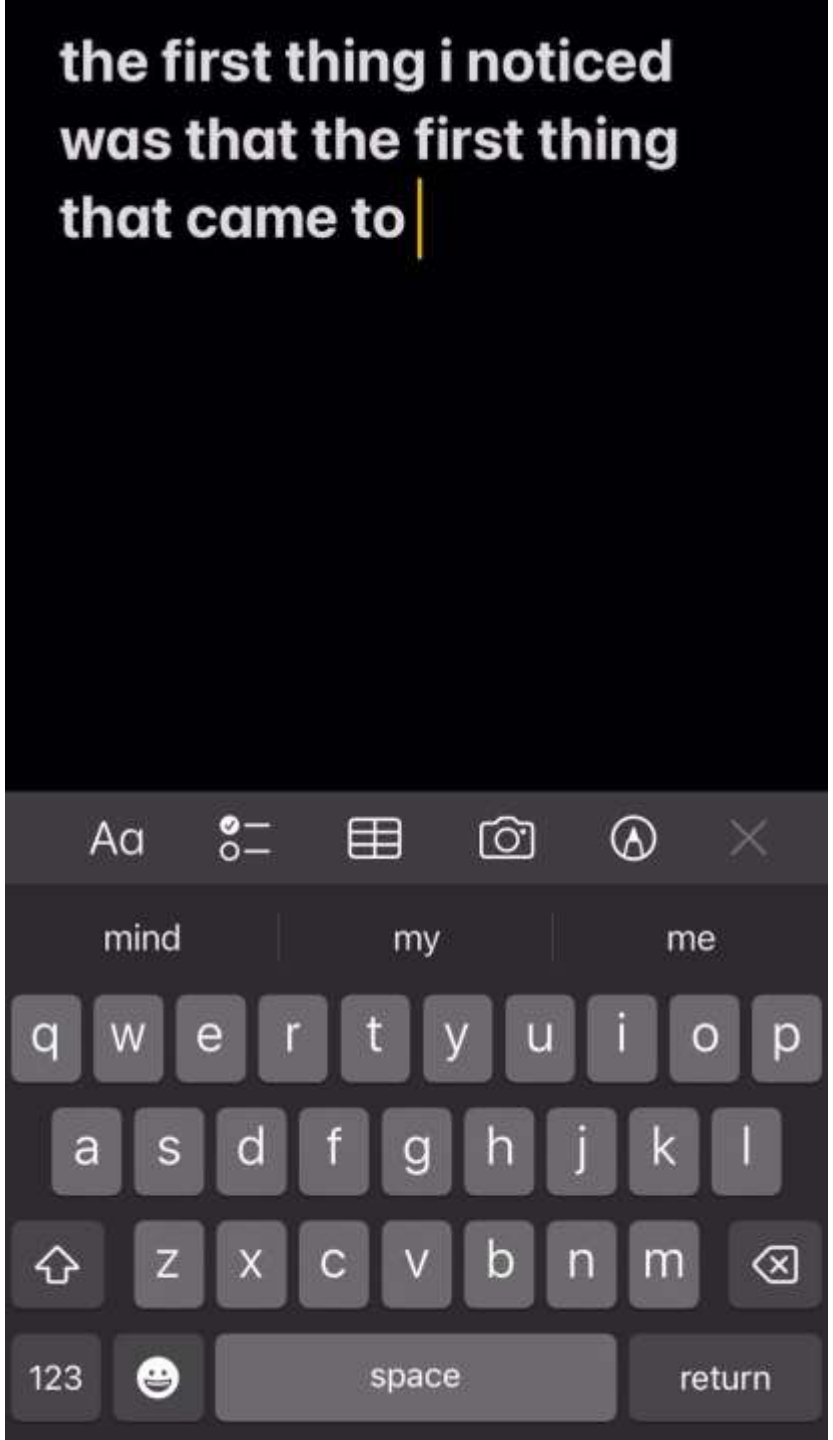
$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$



$p(A, B, C) = p(C)p(B \mid C)p(A \mid B, C)$

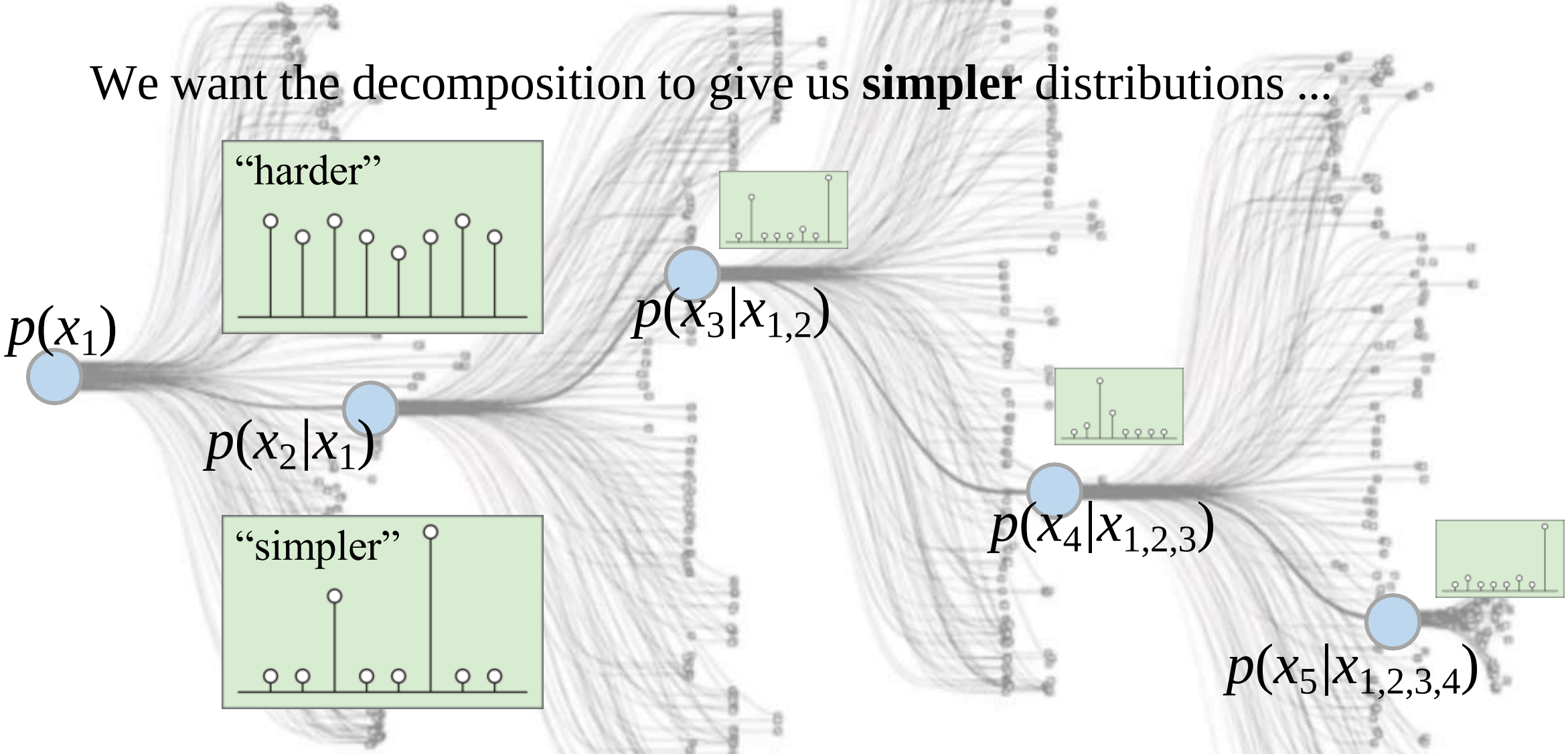


Your phone’s keyboard is
Autoregressive:
Previous outputs can largely reduce the next plausible outputs.



Inductive Bias

We want the decomposition to give us **simpler** distributions ...



Example: every p is a categorical distribution

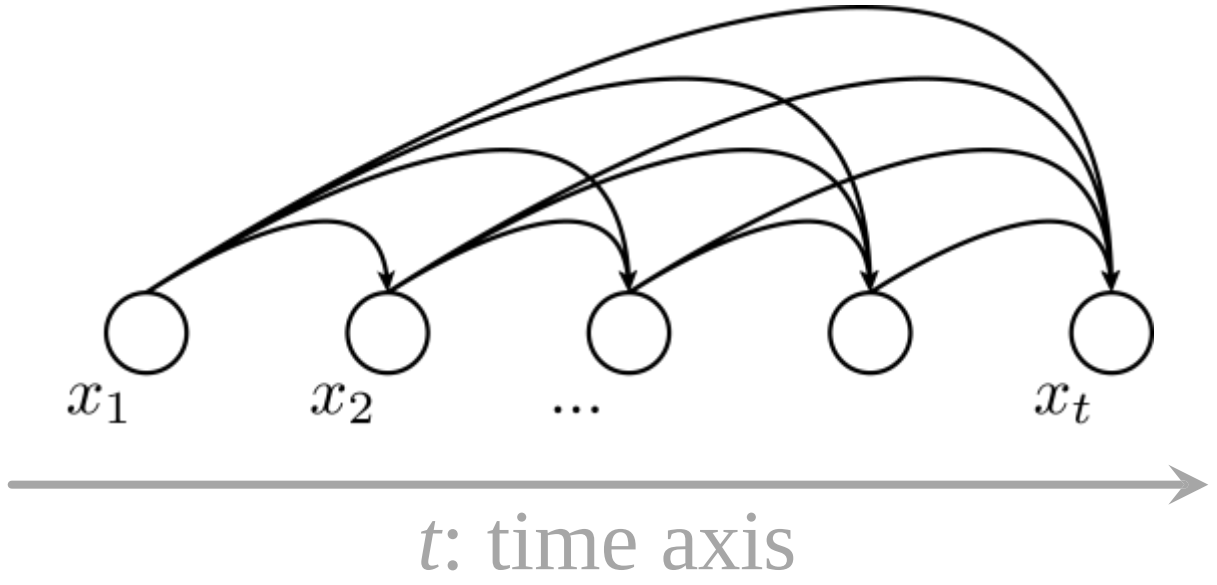
Illustration adapted from AlphaGo

Inductive Bias

We want the decomposition to give us **simpler** distributions ...

Example: “next token prediction”

- Temporal modeling implies an inductive bias



Inductive Bias

We want the decomposed distributions to be represented by "**similar**" neural networks ...

$$p(x_1, x_2, ..., x_n) = \underbrace{p(x_1)}\underbrace{p(x_2 \mid x_1)}...\underbrace{p(x_n \mid x_1, x_2, ..., x_{n-1})}$$

- Conceptually, these are different mappings
- But we model them by **shared architectures**
  (which can be RNN, CNN, Transformer, ...)

Inductive Bias

We want the decomposed distributions to be represented by "**similar**" neural networks ...

$$p(x_1, x_2, ..., x_n) = \underbrace{p_\theta(x_1)}\underbrace{p_\theta(x_2 \mid x_1)}...\underbrace{p_\theta(x_n \mid x_1, x_2, ..., x_{n-1})}$$

- Conceptually, these are different mappings
- But we model them by **shared architectures** (which can be RNN, CNN, Transformer, ...)
- and by **shared weights** θ

Inductive Bias

(Recap): The decomposition makes **no** compromise/approximation:

$$p(x_1, x_2, ..., x_n) \quad = \quad p(x_1)p(x_2 \mid x_1)...p(x_n \mid x_1, x_2, ..., x_{n-1})$$

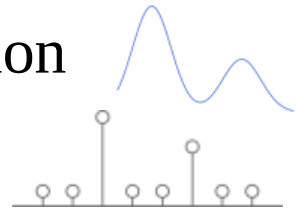
But **inductive biases** introduce approximations:

- **shared** architectures, **shared** weights, ...
- with an induced decomposition

Representing One Distribution

$$p_\theta(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

- Network inputs: $x_1, x_2, \ldots, x_{i-1}$
- Network output: a distribution of x_i
  - Continuous distribution
  - Discrete distribution



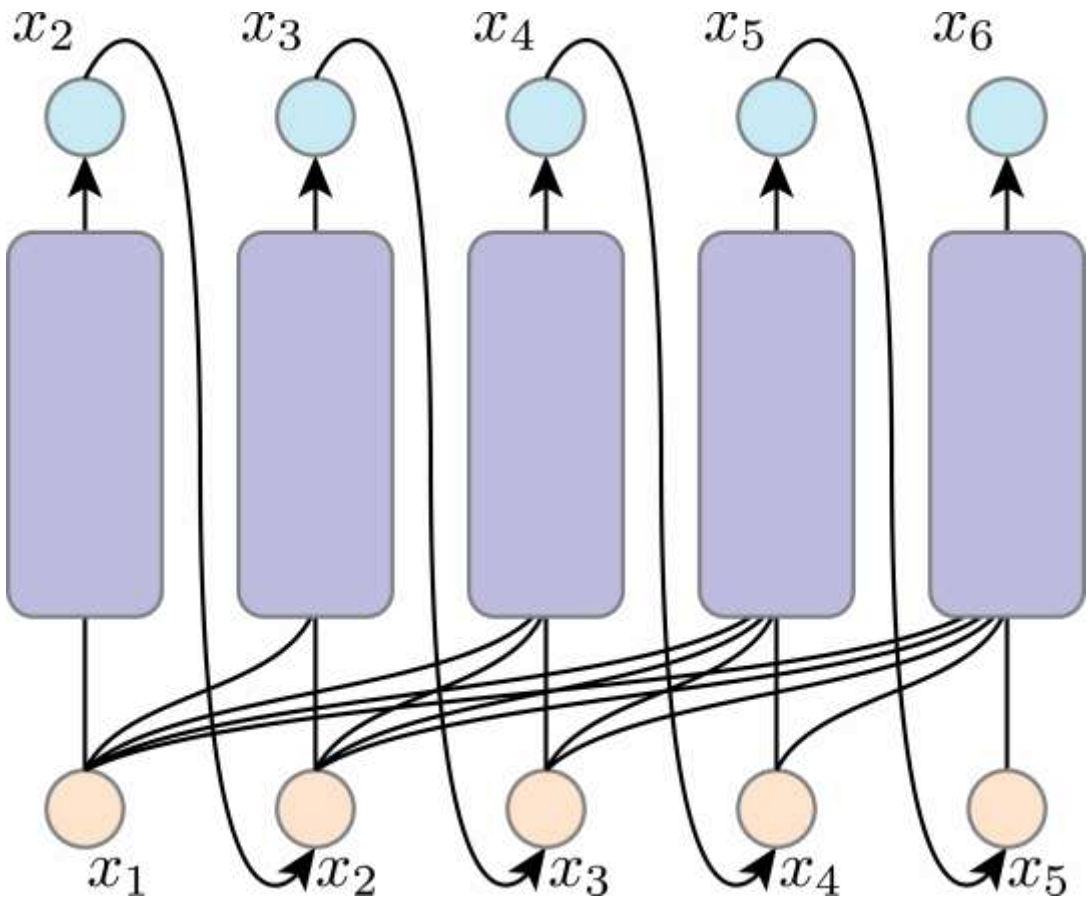
Note:

- W/ a discrete distribution, this network behaves like classification (the “regression” part of autoregression)
- Discrete distribution is popular in AR models, but not a must

Inference: Autoregressive

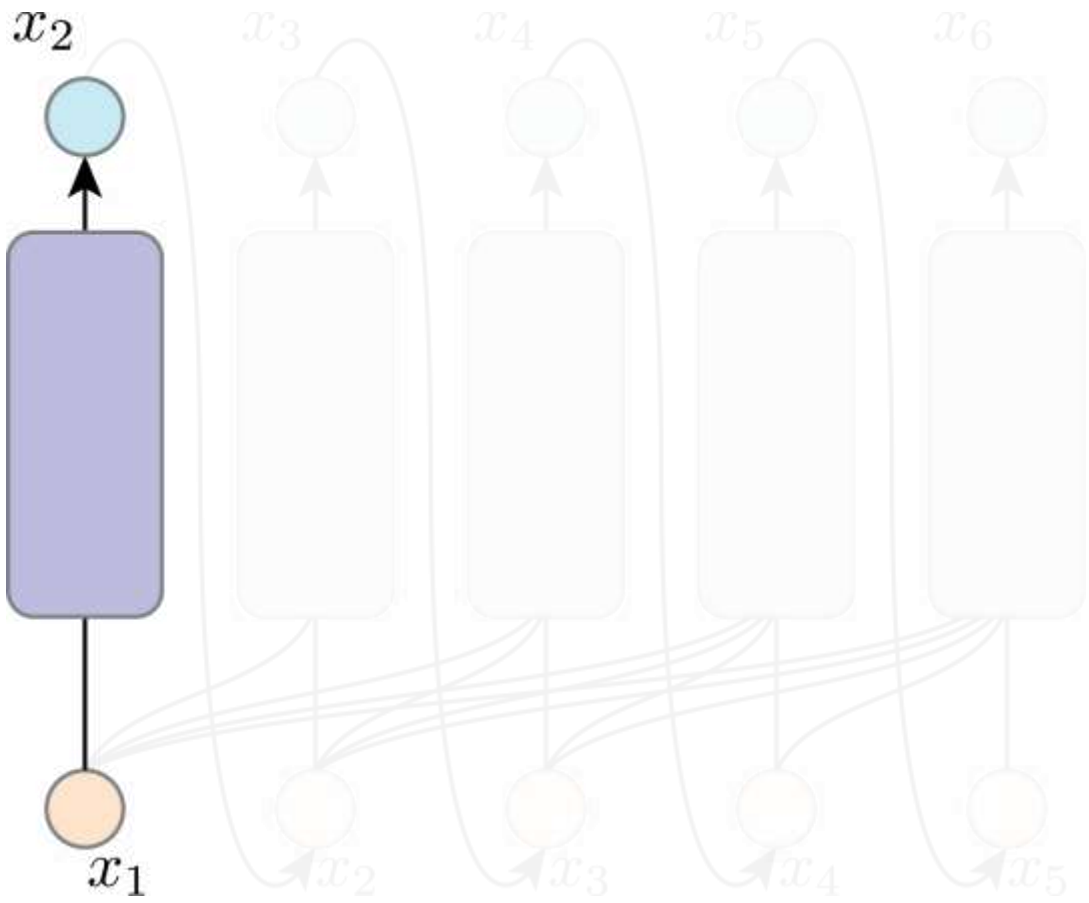
This figure implements this formulation:

$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$



Inference: Autoregressive

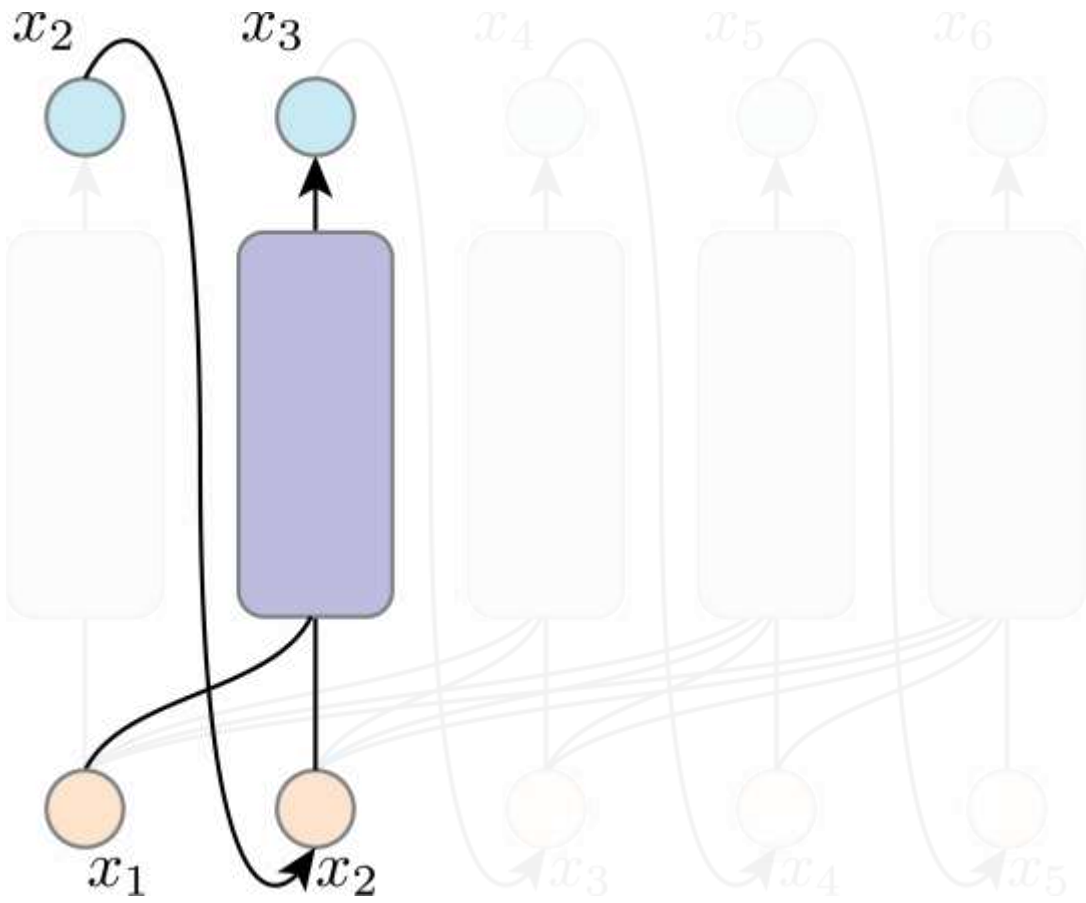
- Thi net models $p(x_2 \mid x_1)$
- 1 input
- 1 output



Inference: Autoregressive

- This net models $p(x_3 \mid x_{1,2})$
- **2 inputs**
- 1 output

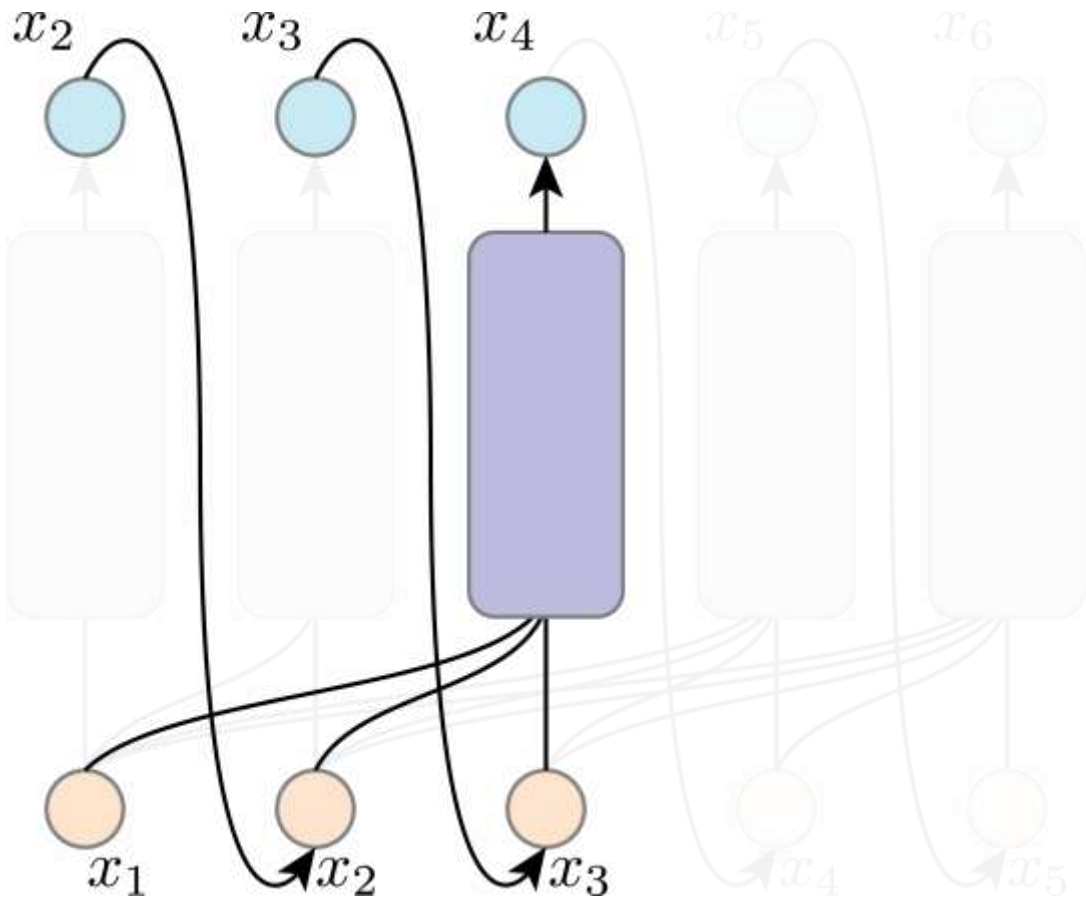
- inputs: outputs from previous steps



Inference: Autoregressive

- This net models $p(x_4 \mid x_{1,2,3})$
- **3 inputs**
- 1 output

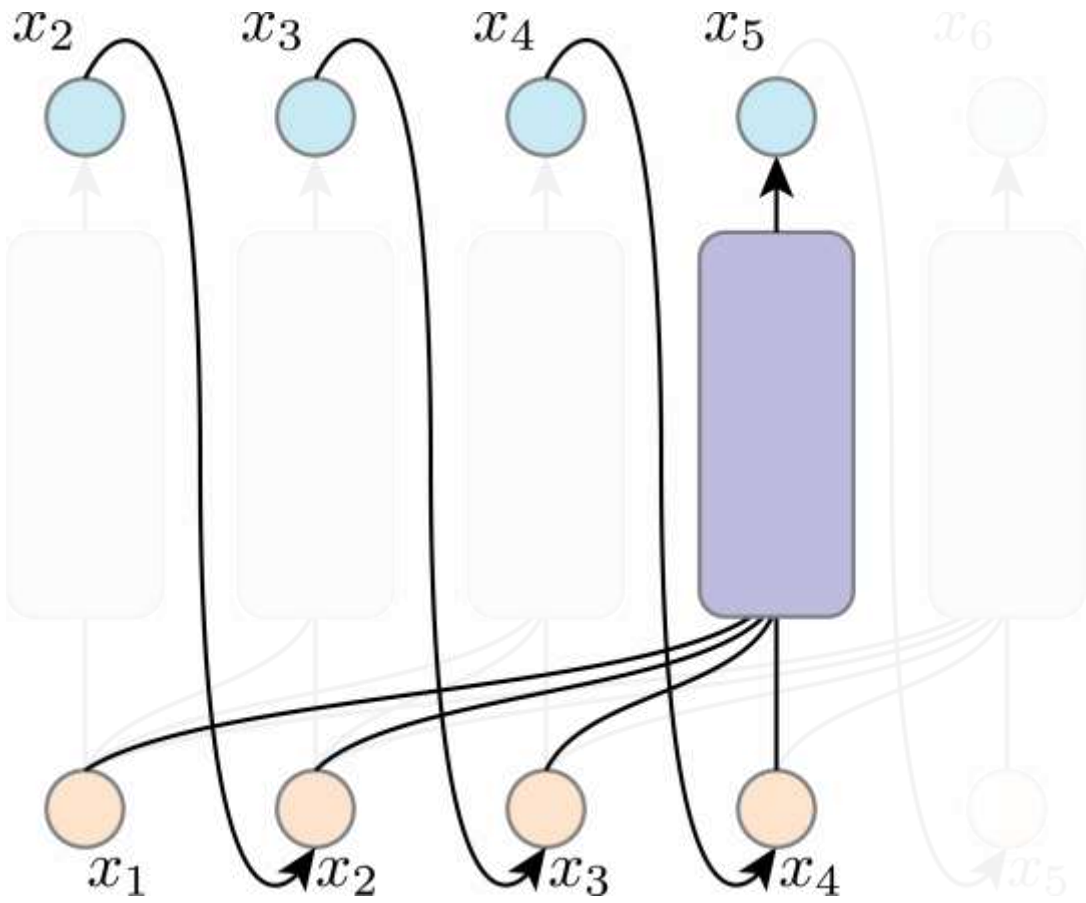
- inputs: outputs from previous steps



Inference: Autoregressive

- This net models $p(x_5 \mid x_{1,2,3,4})$
- **4 inputs**
- 1 output

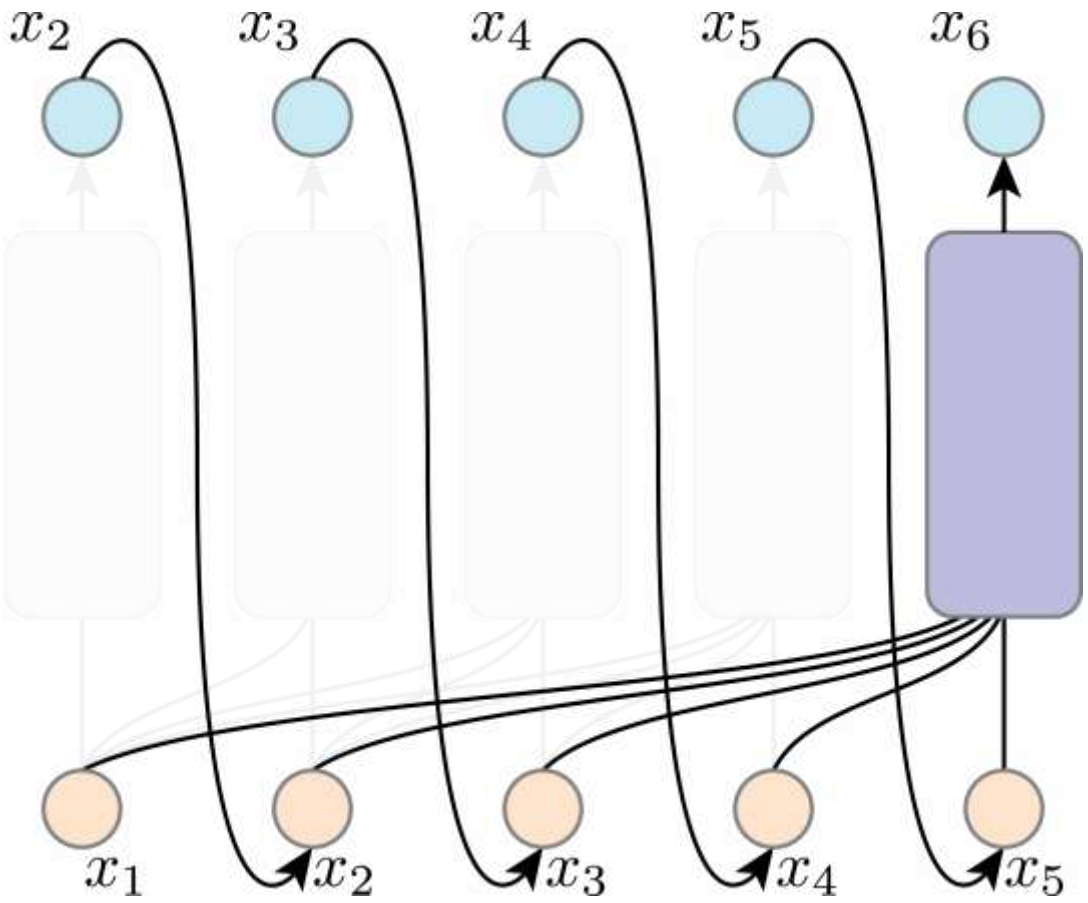
- inputs: outputs from previous steps



Inference: Autoregressive

- This net models $p(x_6 \mid x_{1,2,3,4,5})$
- **5 inputs**
- 1 output

- inputs: outputs from previous steps

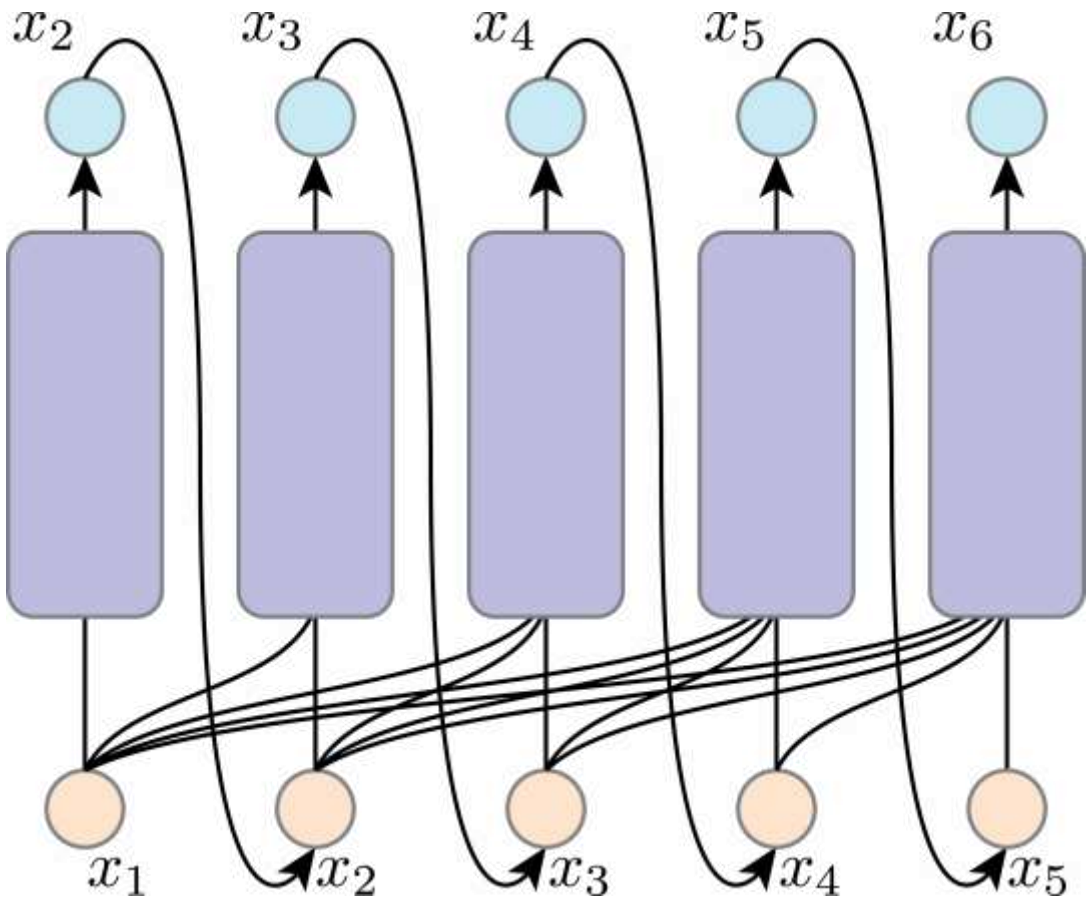


Inference: Autoregressive

Note:

- This is a **recursive** process

- but **not** necessarily done by RNN
- can be done by **any** architecture (e.g., CNN or Transformers)



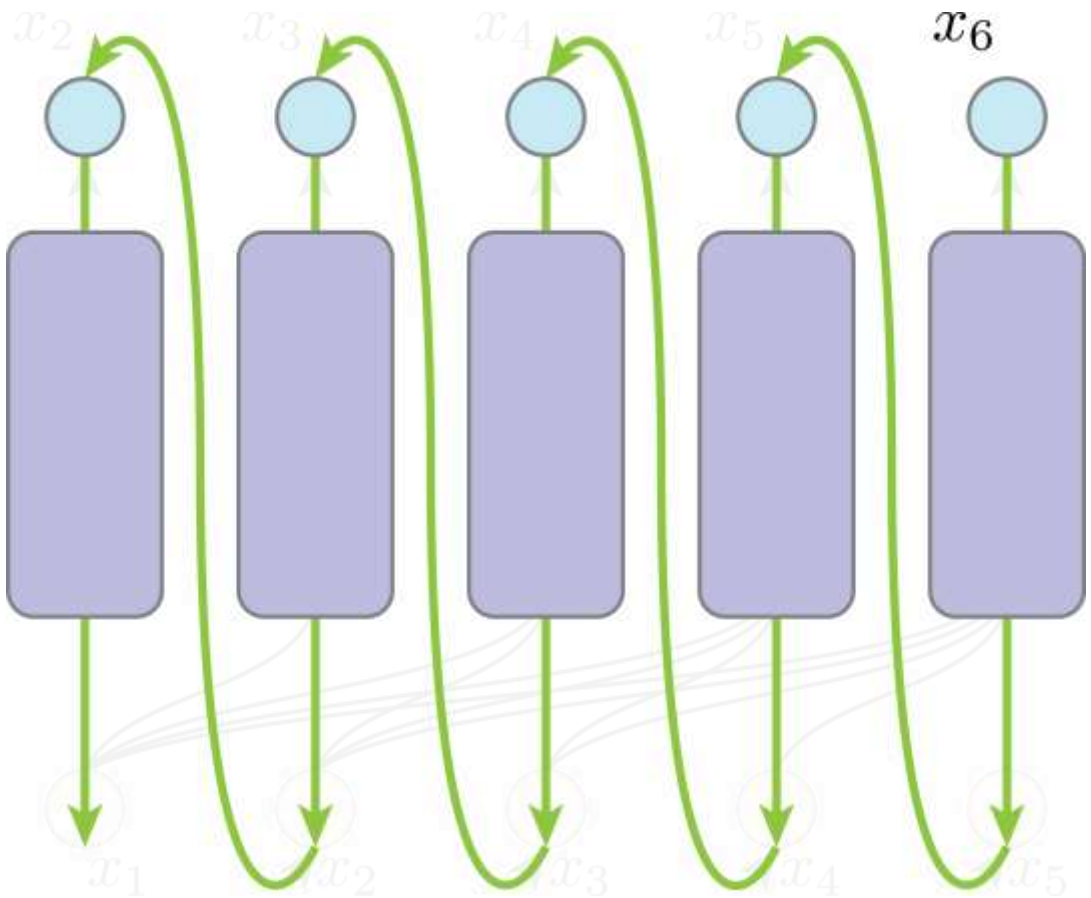
What if we backprop through this graph “as-is”?

Consider one gradient path of x_6:

- go through all previous outputs, ...
- all previous sampling ops, ...
- all previous networks

(e.g., each is a full Transformer)

It’s **infeasible** to **train** the AR model following its **inference** graph.



one gradient path

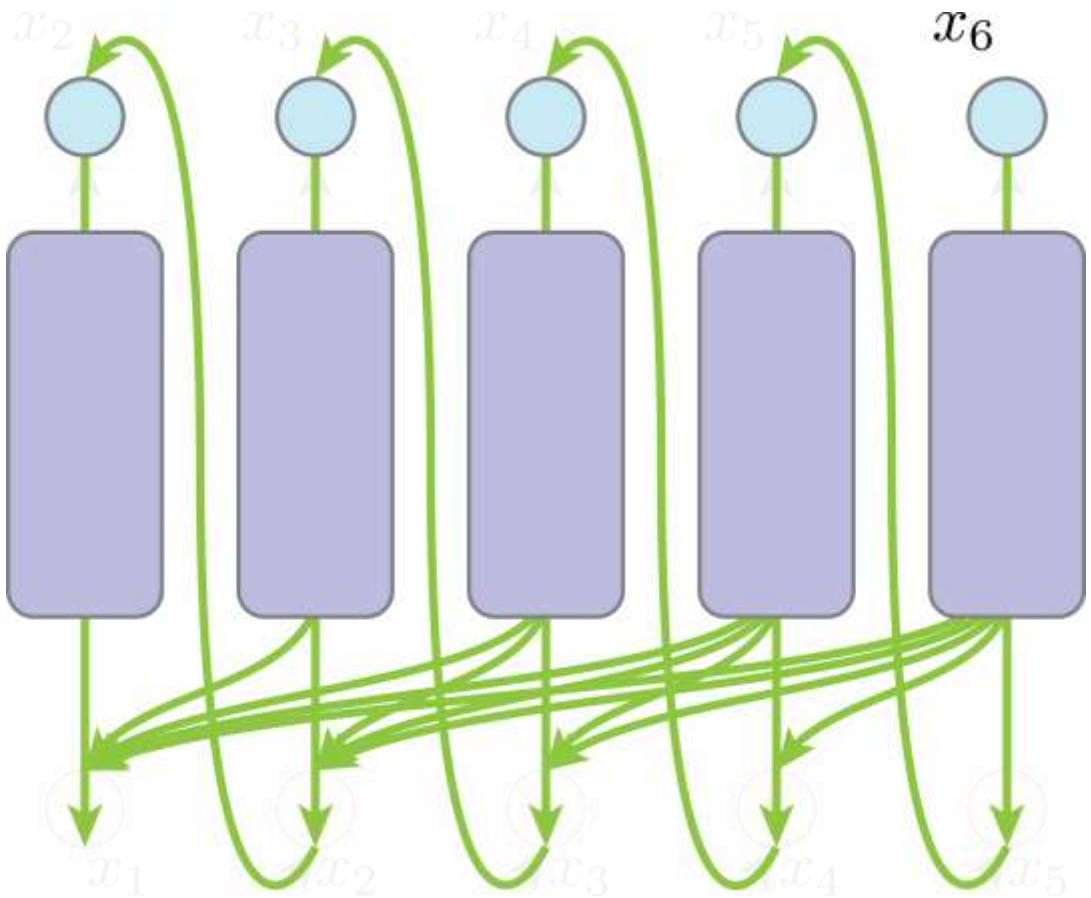
What if we backprop through this graph “as-is”?

Consider one gradient path of x_6:

- go through all previous outputs, ...
- all previous sampling ops, ...
- all previous networks

(e.g., each is a full Transformer)

It’s **infeasible** to **train** the AR model following its **inference** graph.

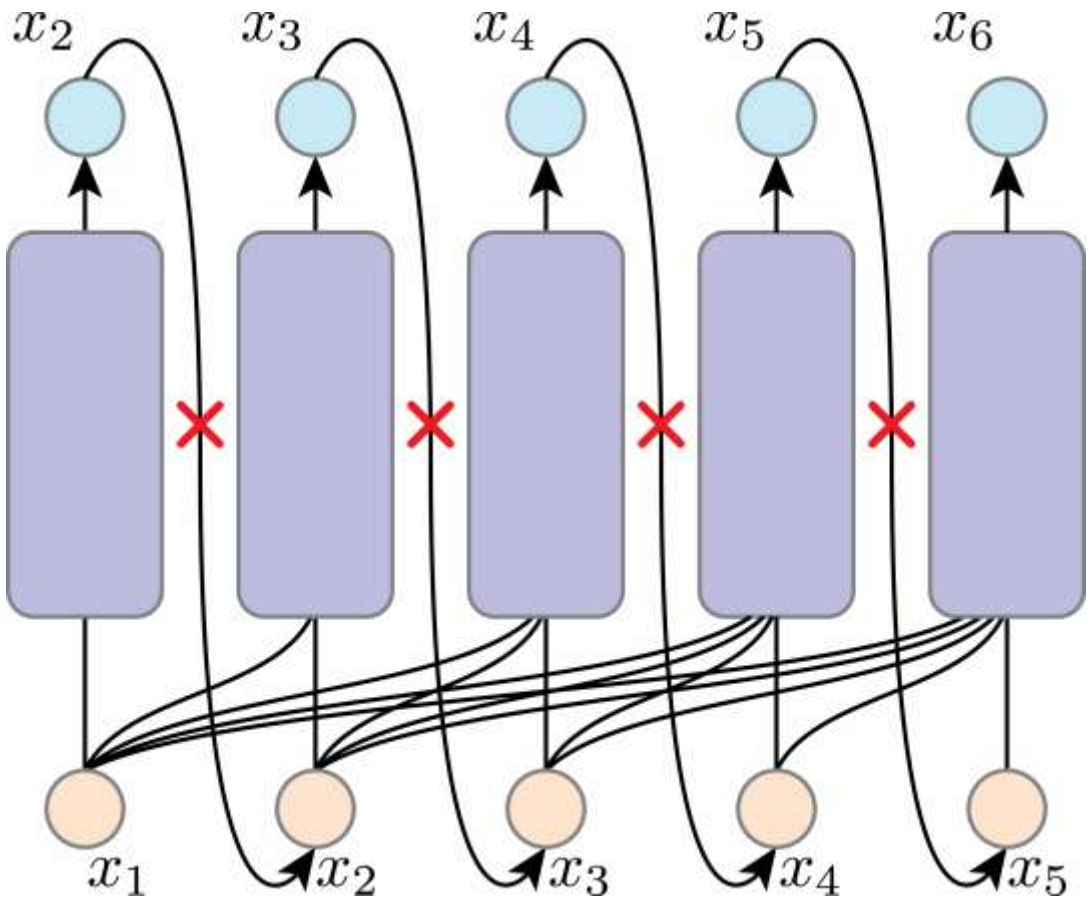


full gradients

Training: Teacher-Forcing

Teacher-forcing

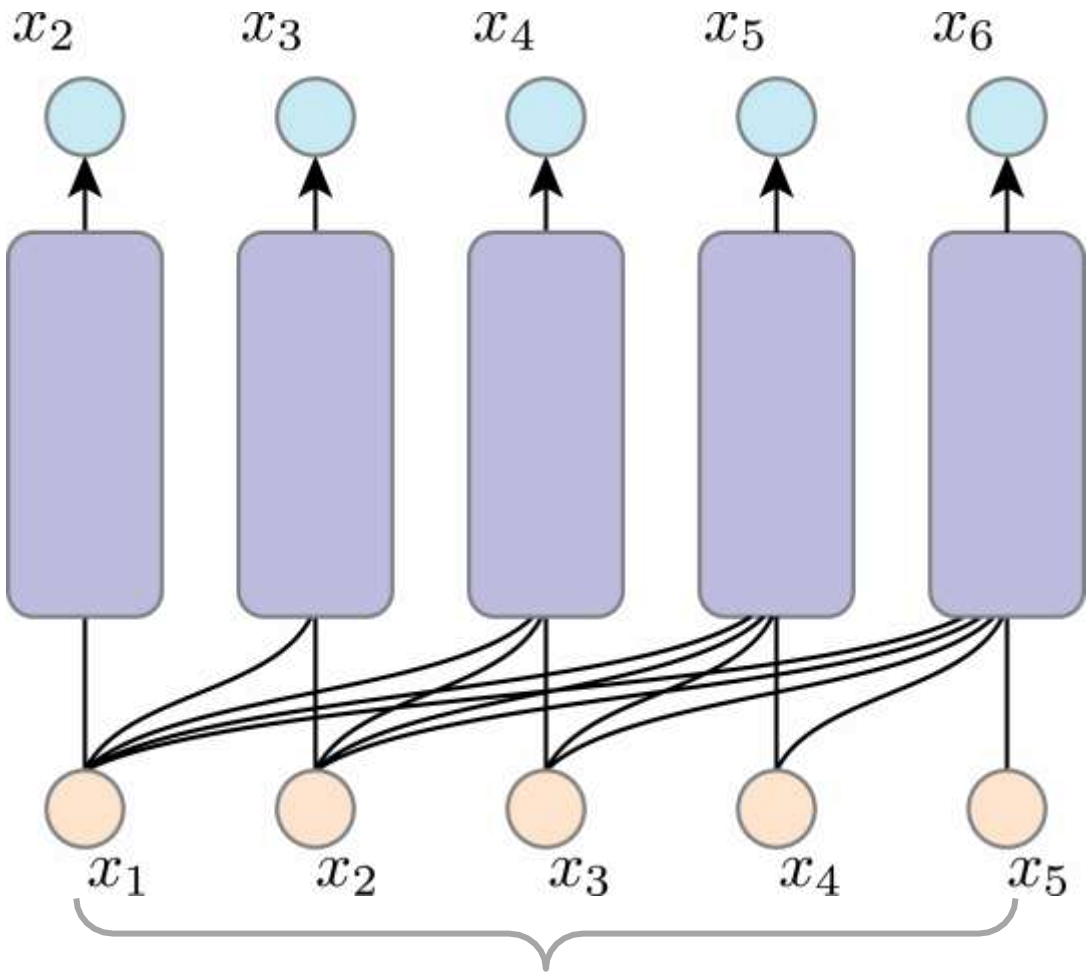
- Inputs are <u>not</u> from previous outputs
- Inputs are from ground-truth data



Training: Teacher-Forcing

Teacher-forcing

- Inputs are <u>not</u> from previous outputs
- Inputs are from ground-truth data



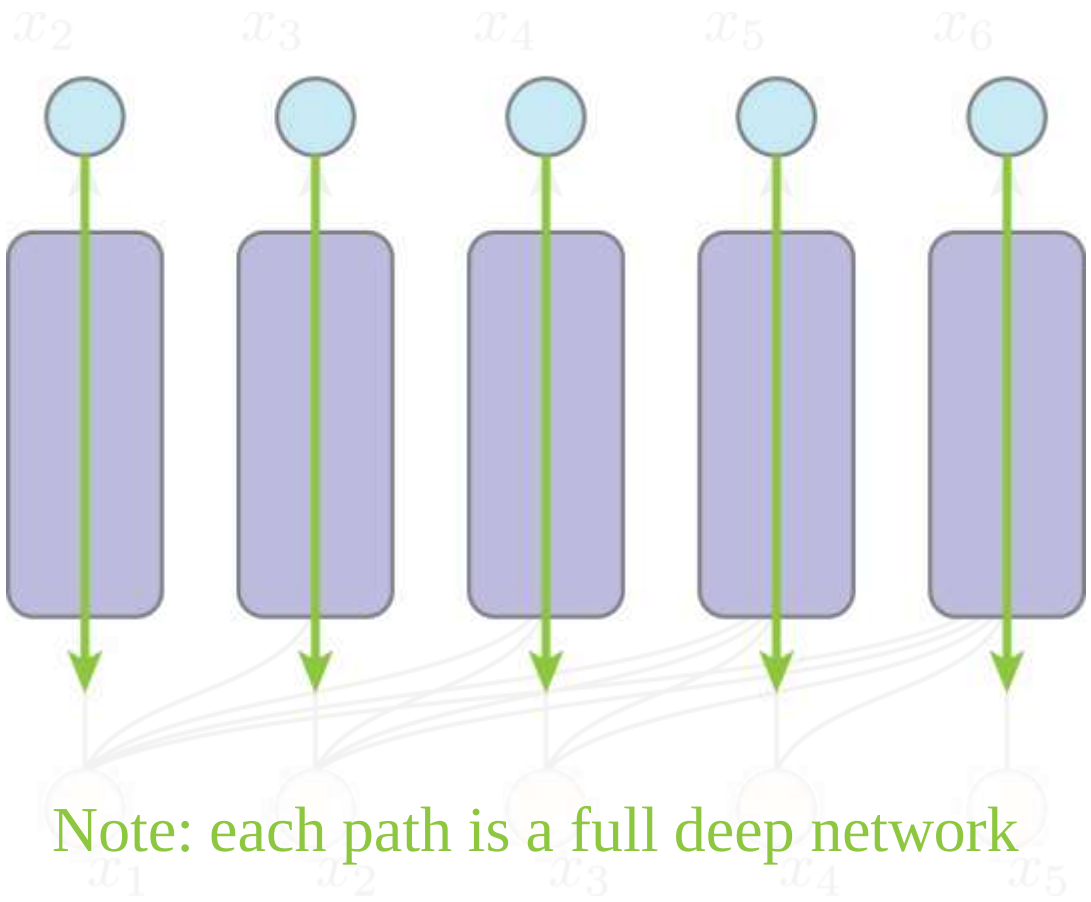
Training: Teacher-Forcing

Teacher-forcing

- Inputs are <u>not</u> from previous outputs
- Inputs are from ground-truth data

Pros:

- backprop path is much shorter



Note: each path is a full deep network

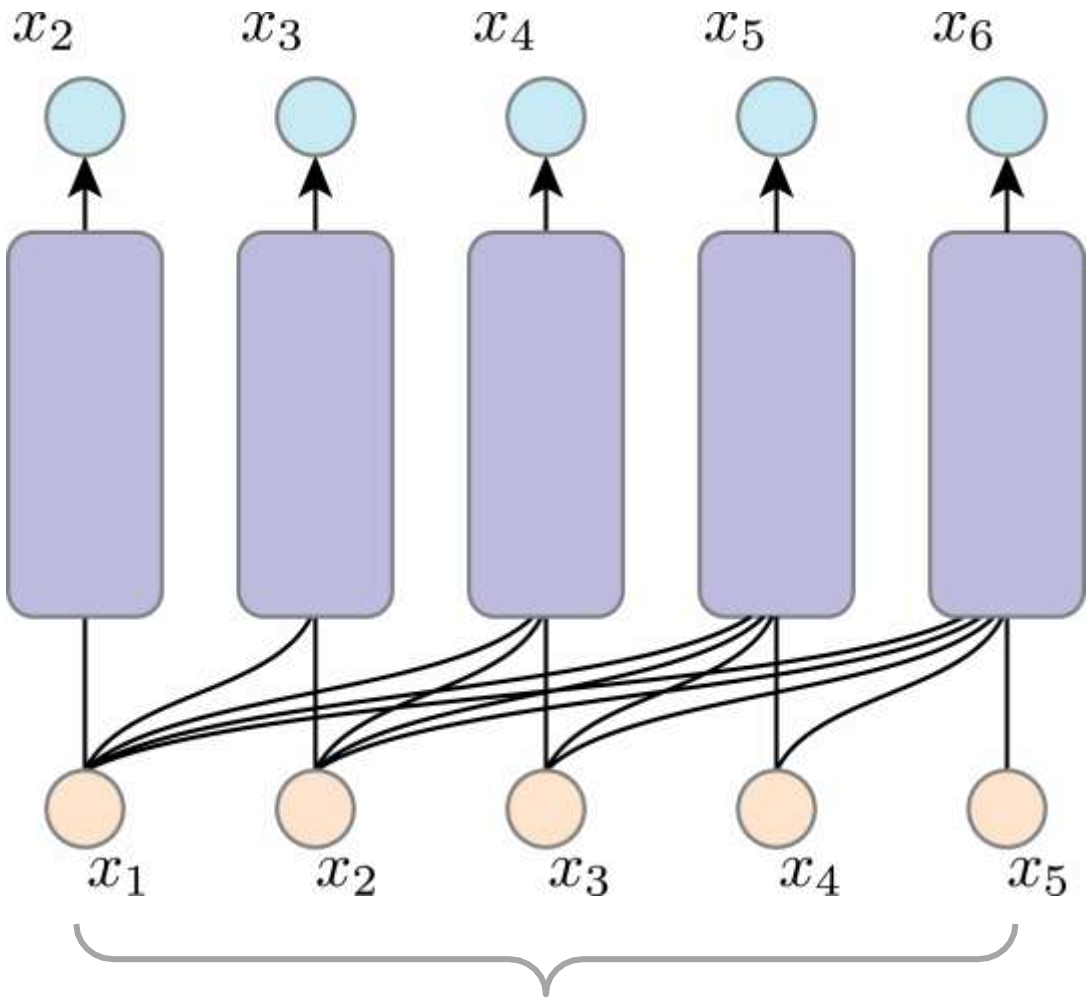
Training: Teacher-Forcing

Teacher-forcing

- Inputs are <u>not</u> from previous outputs
- Inputs are from ground-truth data

Pros:

- backprop path is much shorter
- ground-truth inputs can ease training



Training: Teacher-Forcing

Teacher-forcing

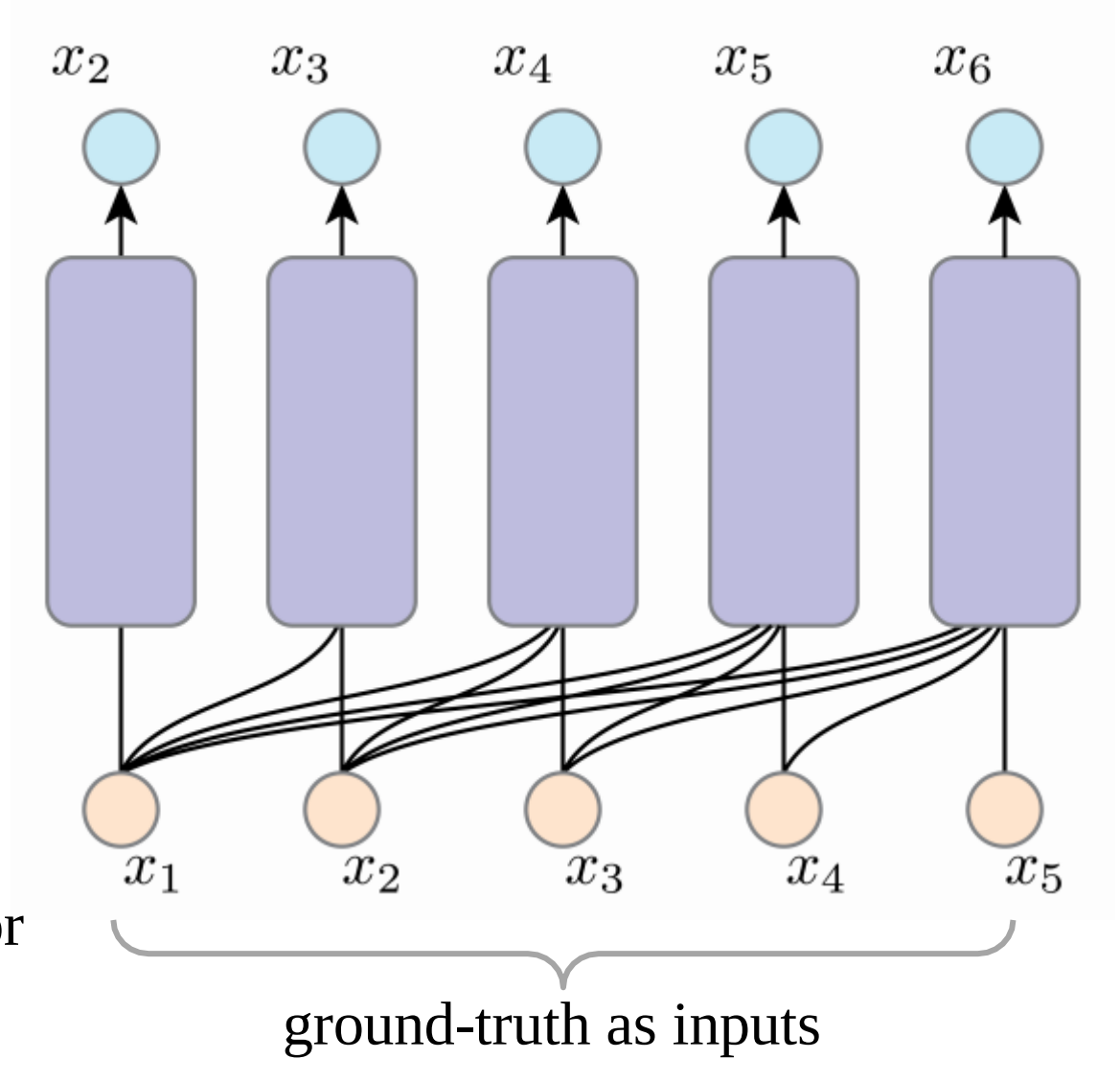
- Inputs are not from previous outputs
- Inputs are from ground-truth data

Pros:

- backprop path is much shorter
- ground-truth inputs can ease training

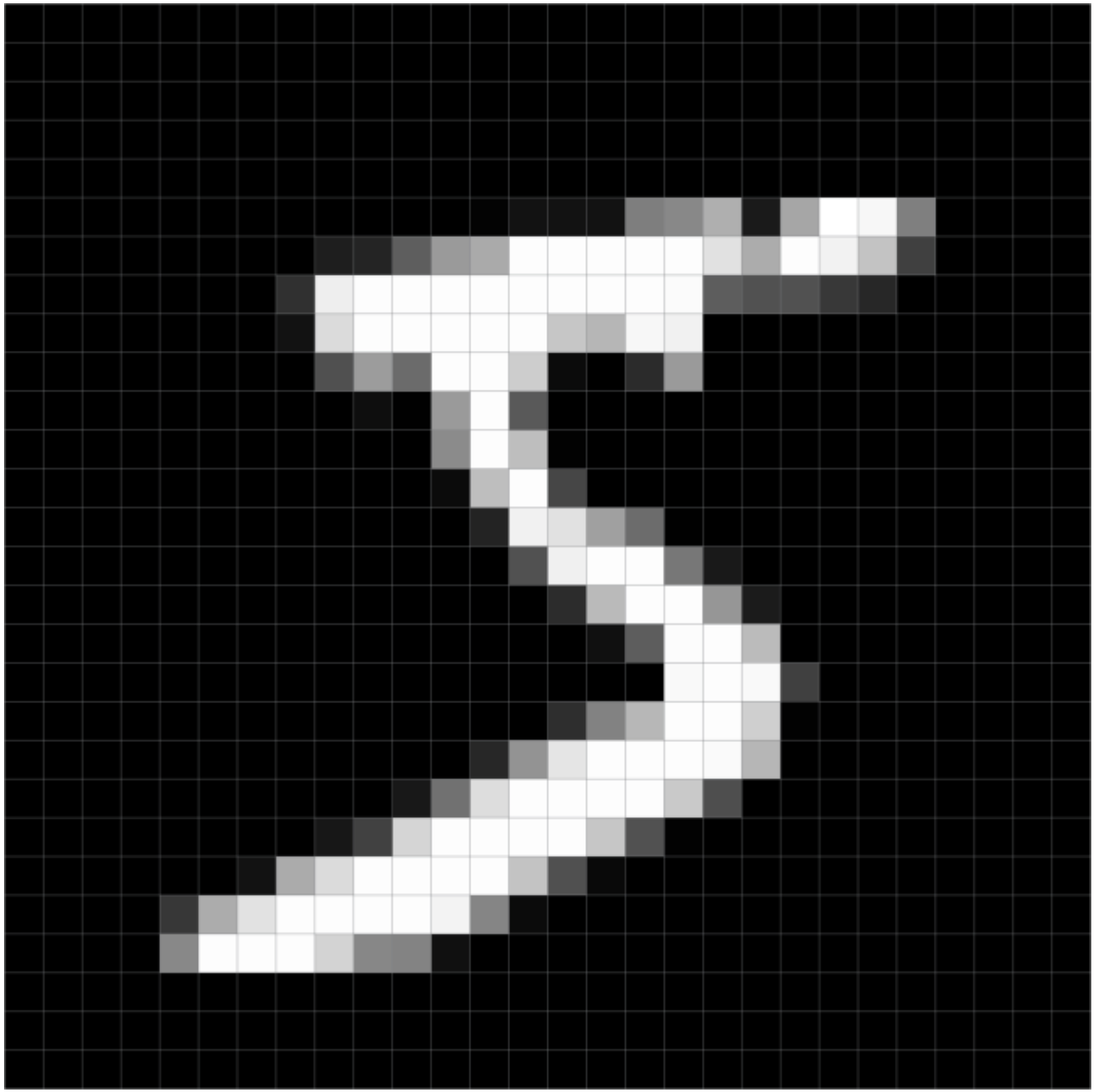
Cons:

- inconsistent training/inference
- distribution shift: can't see its own error



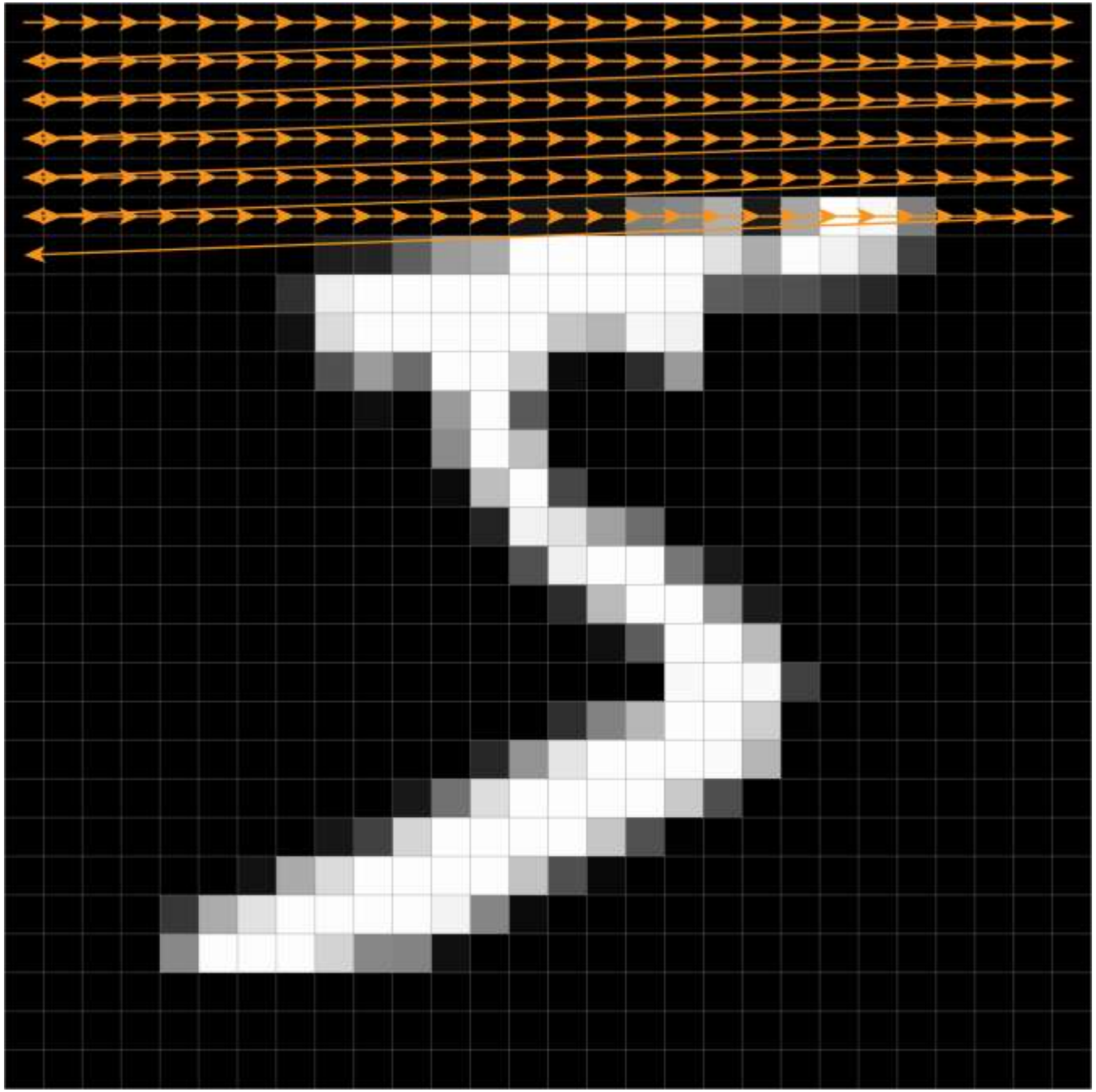
Running example: AR on MNIST

- an image as a sequence of pixels



Running example: AR on MNIST

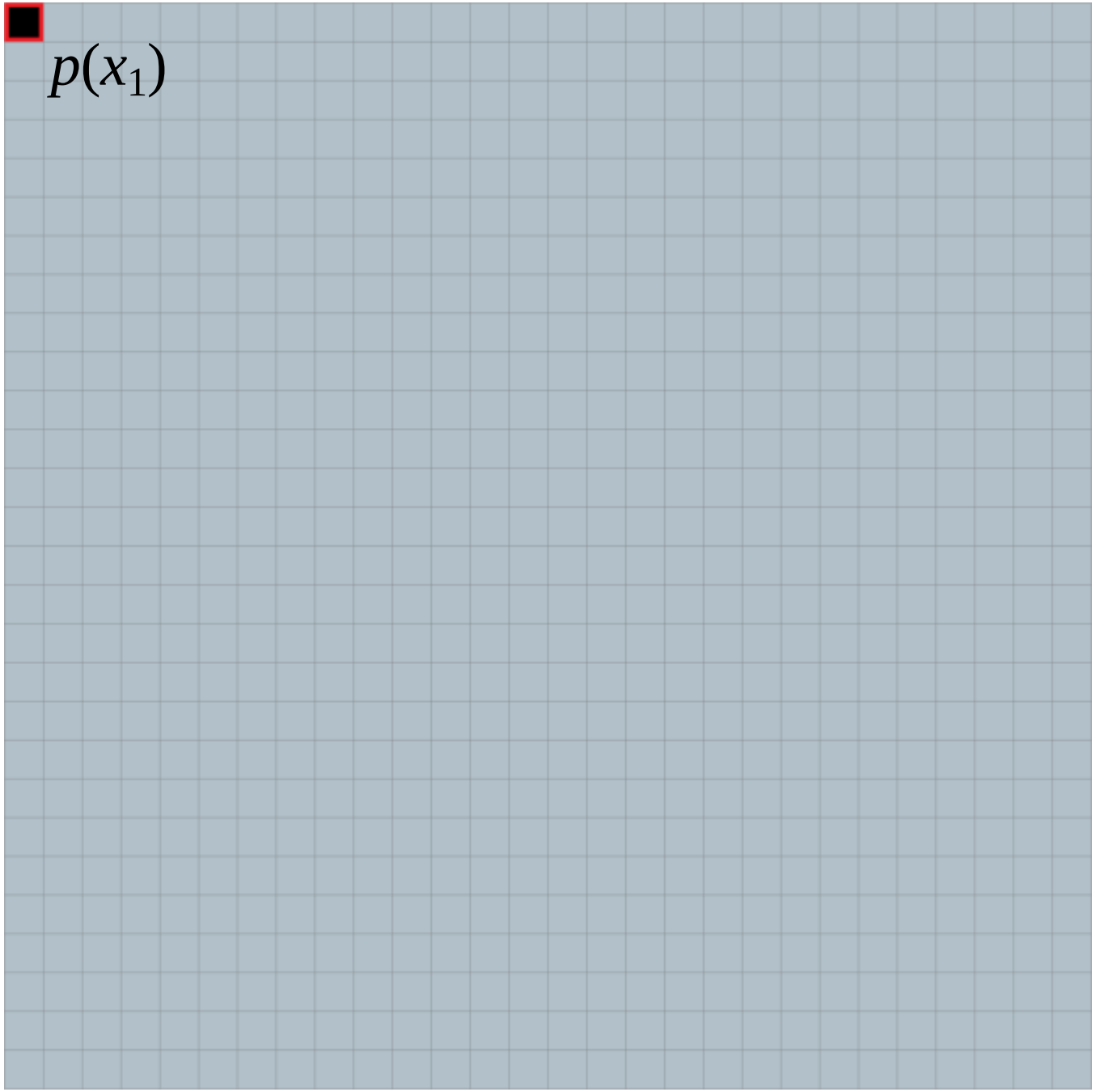
- an image as a sequence of pixels
- scan by raster order



Running example: AR on MNIST

Inference: Autoregressive

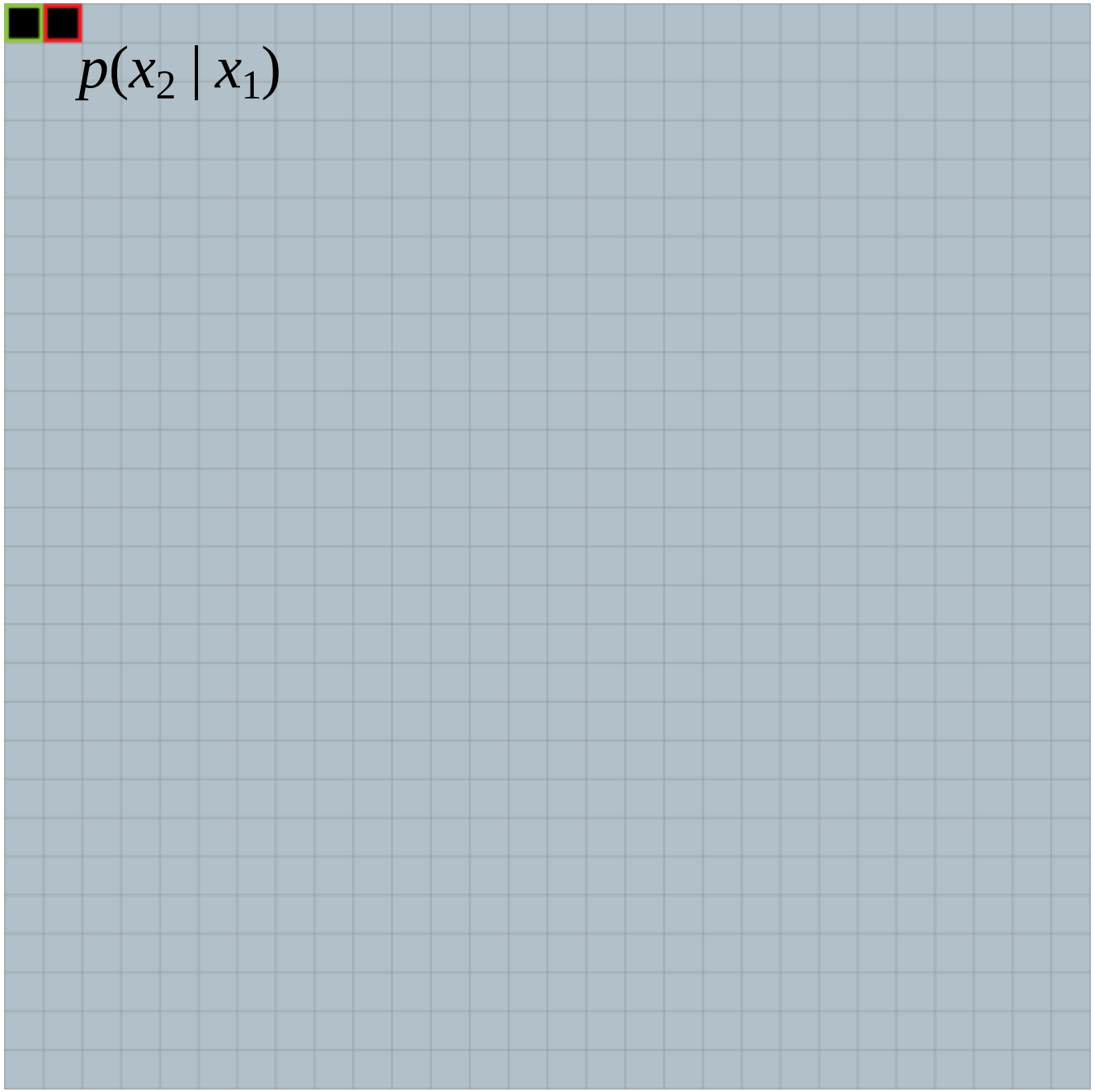
- sample this pixel from $p(x_1)$



Running example: AR on MNIST

Inference: Autoregressive

- sample this pixel from $p(x_2 \mid x_1)$
- this is output from previous step, input for current step
- network for this step:
  - 1 input
  - 1 predict



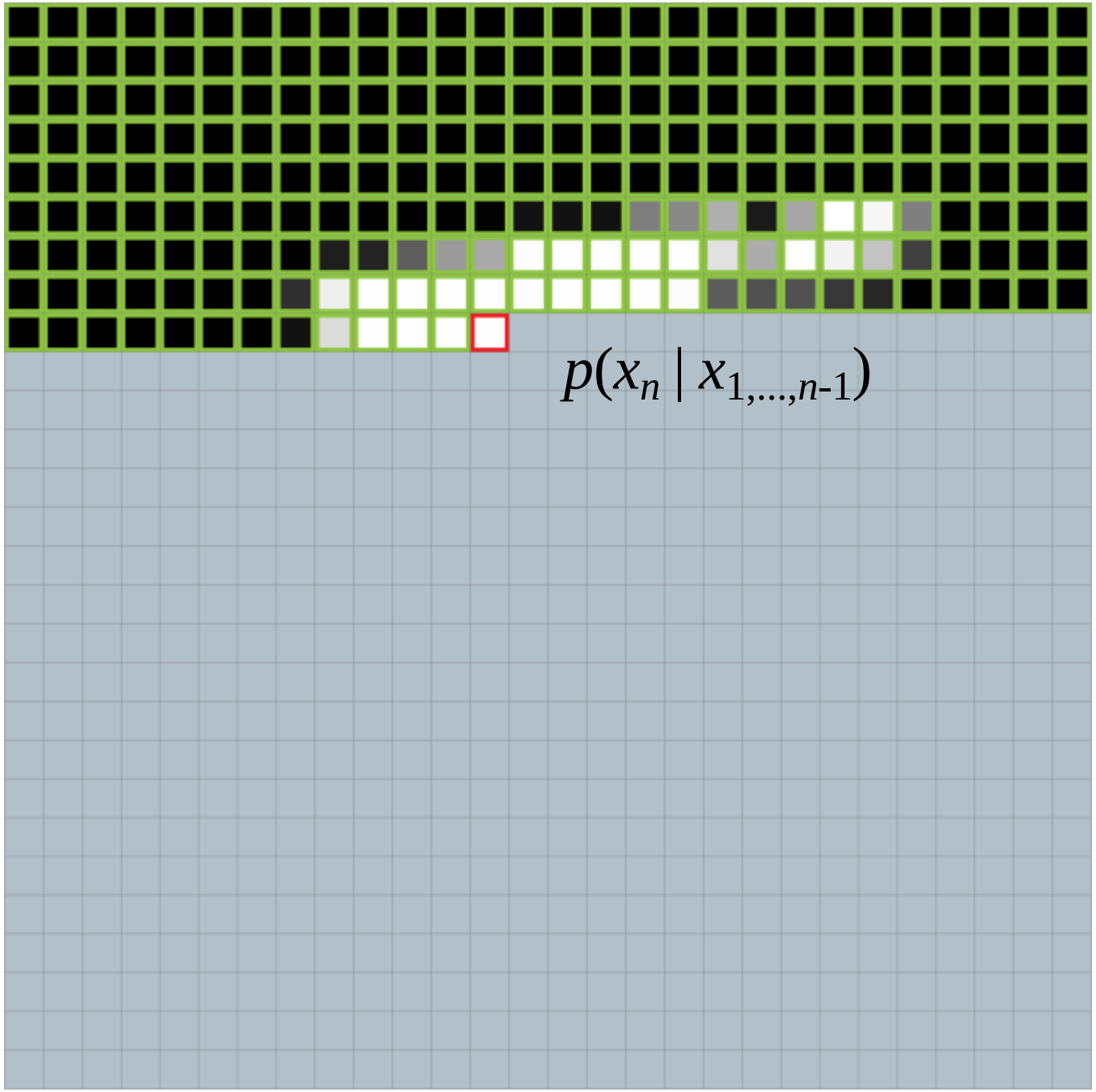
Running example: AR on MNIST

Inference: Autoregressive

- sample this pixel from $p(x_n \mid x_{1,\ldots,n\text{-}1})$

- these are outputs from previous steps, inputs for current step

- network for this step:
  - $(n - 1)$ inputs
  - 1 predict



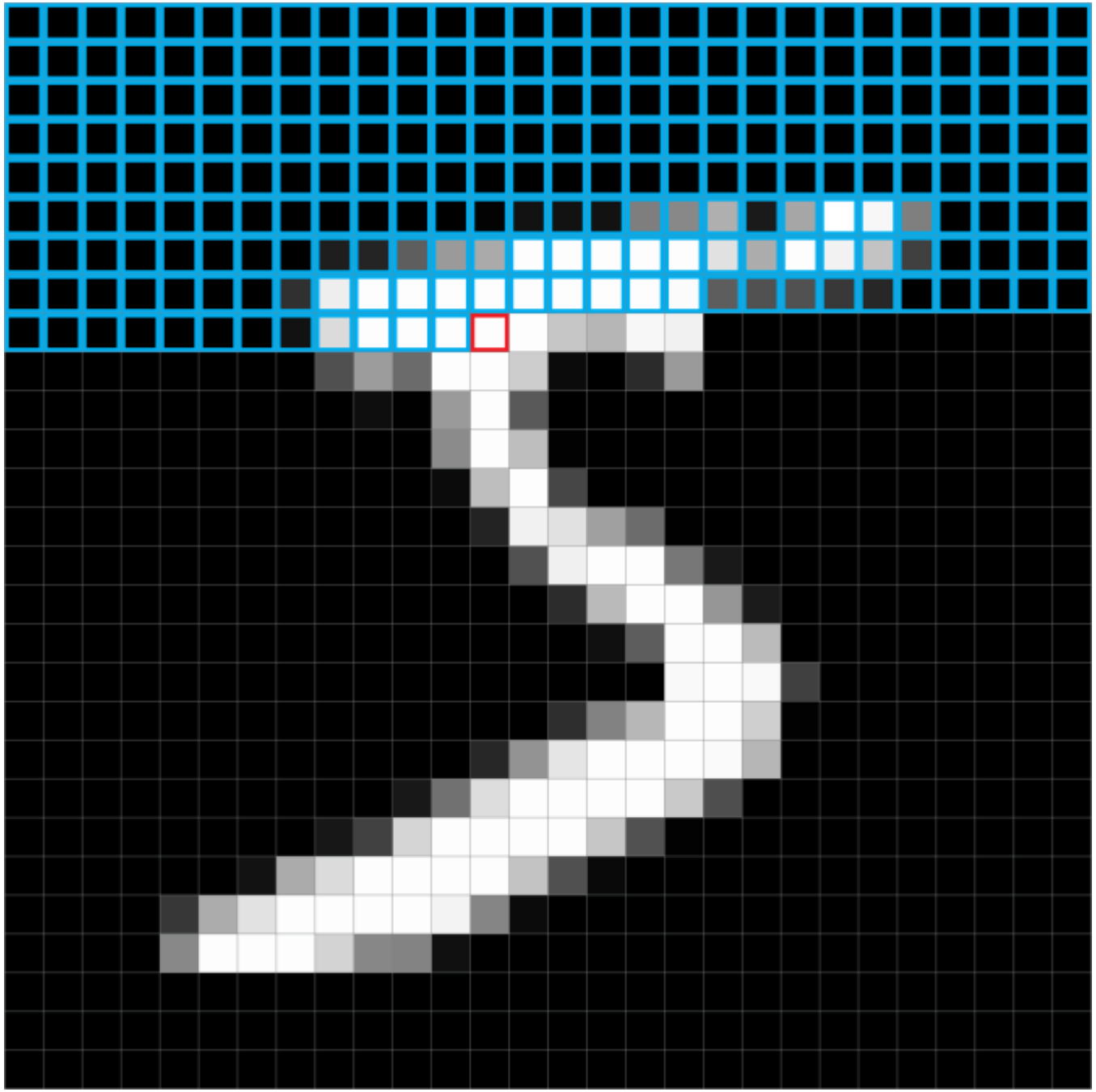
Running example: AR on MNIST

Training: Teacher-Forcing

- model this pixel by: $p(x_n \mid x_{1,\ldots,n\text{-}1})$

- these are outputs from ground-truth, inputs for current step

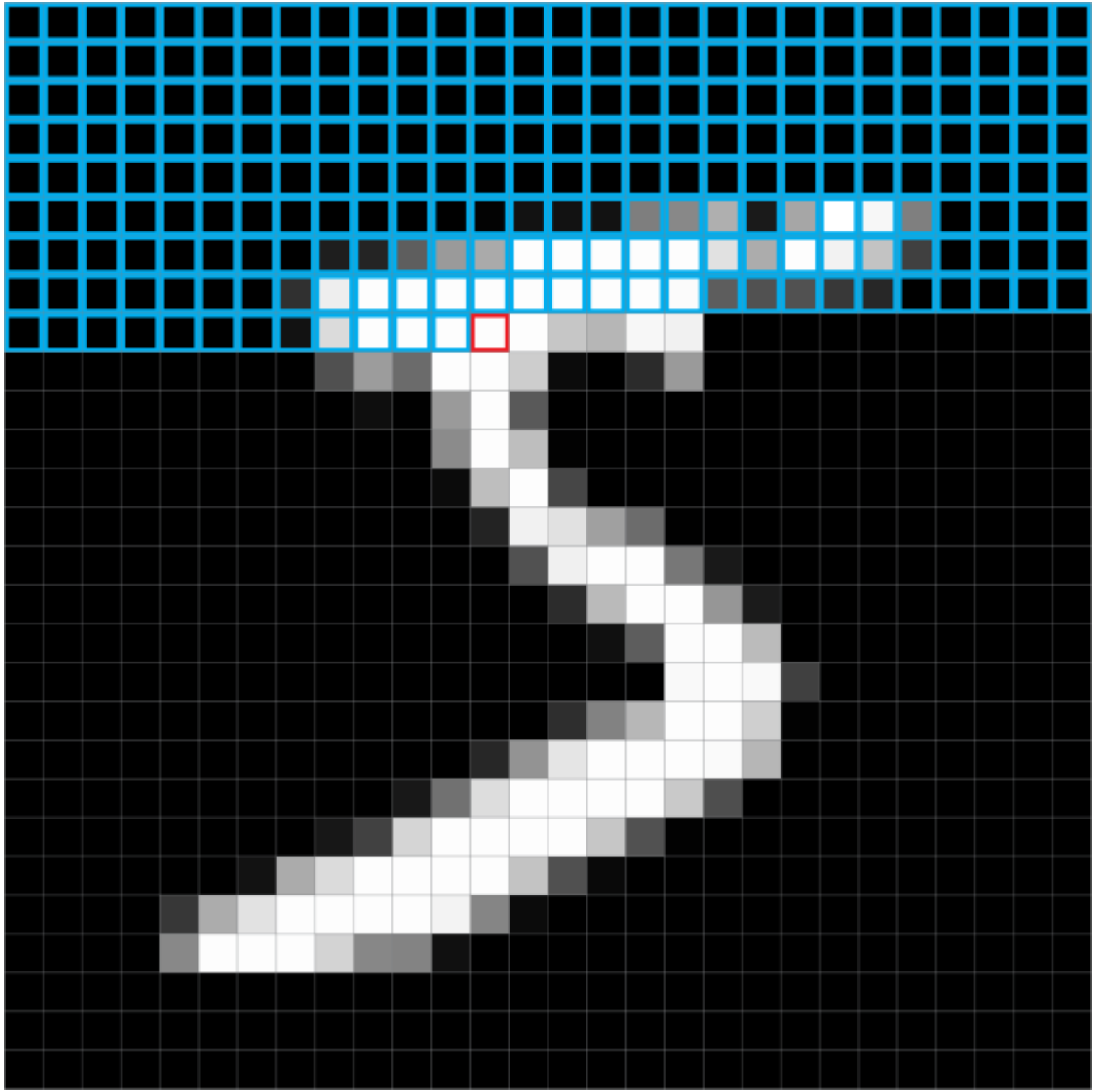
- network for this step:
  - $(n - 1)$ inputs
  - 1 predict



Running example: AR on MNIST

Note:

- This says nothing about architectures
- It’s valid for:
  RNN, CNN, Transformer, ...



Autoregressive Models

Summary:

- Joint distribution ⇒ product of conditionals
- Inductive bias:
  - shared architecture, shared weight
  - induced order
- Inference: autoregressive
- Training: teacher-forcing

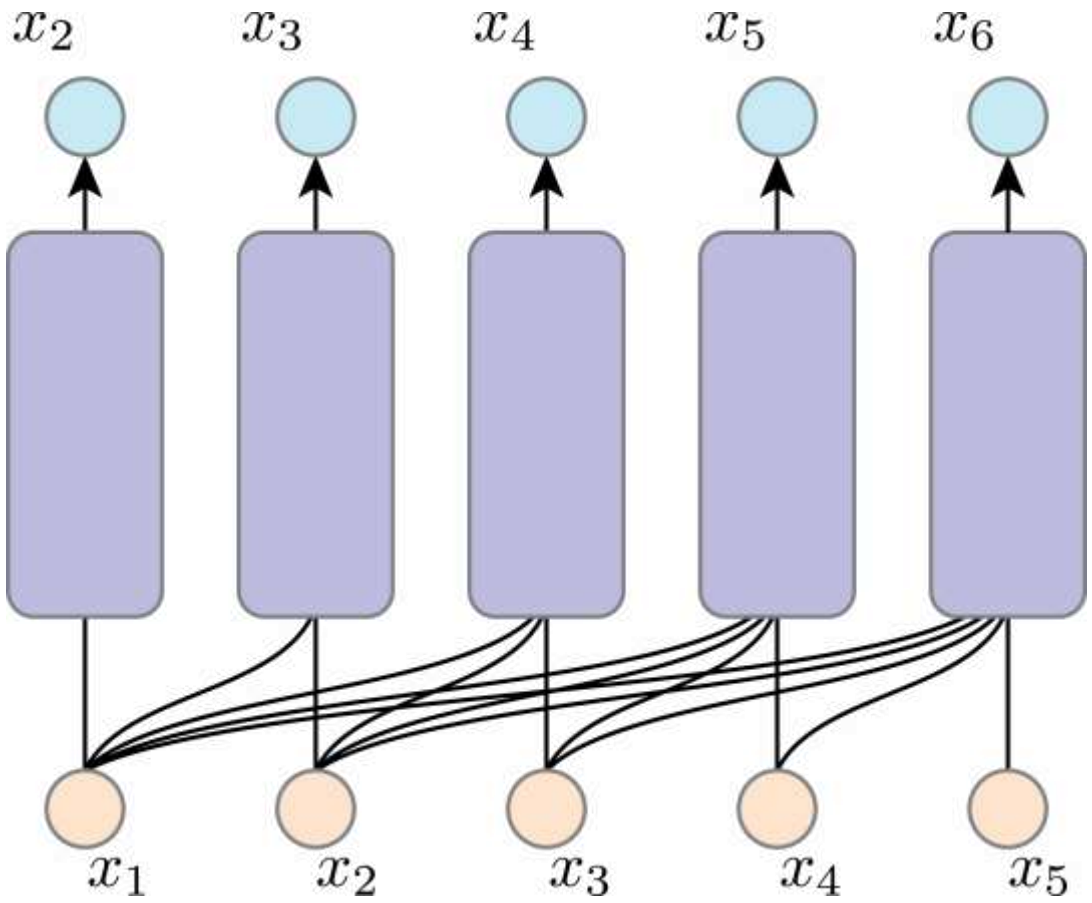
These are not specific to a certain type of network architectures.

Network Architectures for Autoregressive Modeling

Autoregression is not architecture-specific

This figure implements this formulation:

$$p(x_1, x_2, ..., x_n) =$$
$$\prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$



(showing training case for

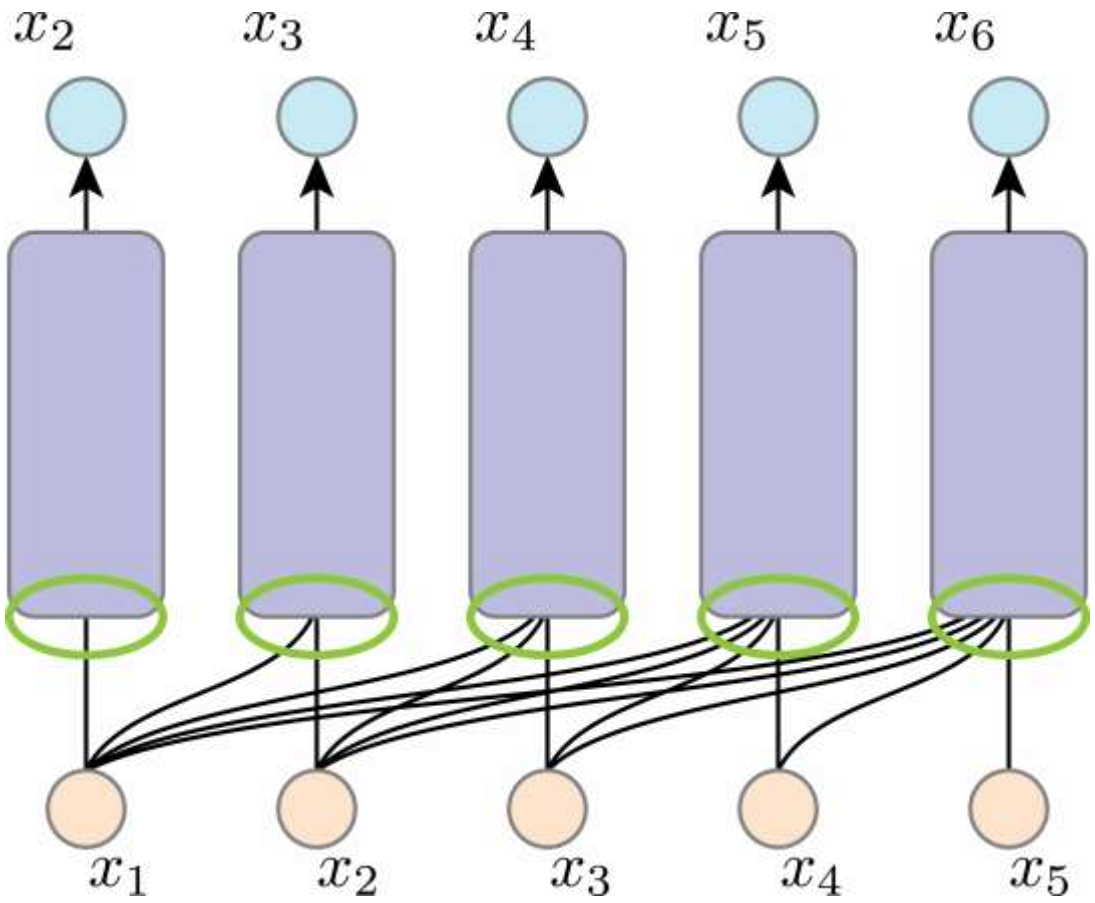
Autoregression is not architecture-specific

This figure implements this formulation:

$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$

In this example:

- 5 networks ...
- each has 1 to 5 inputs

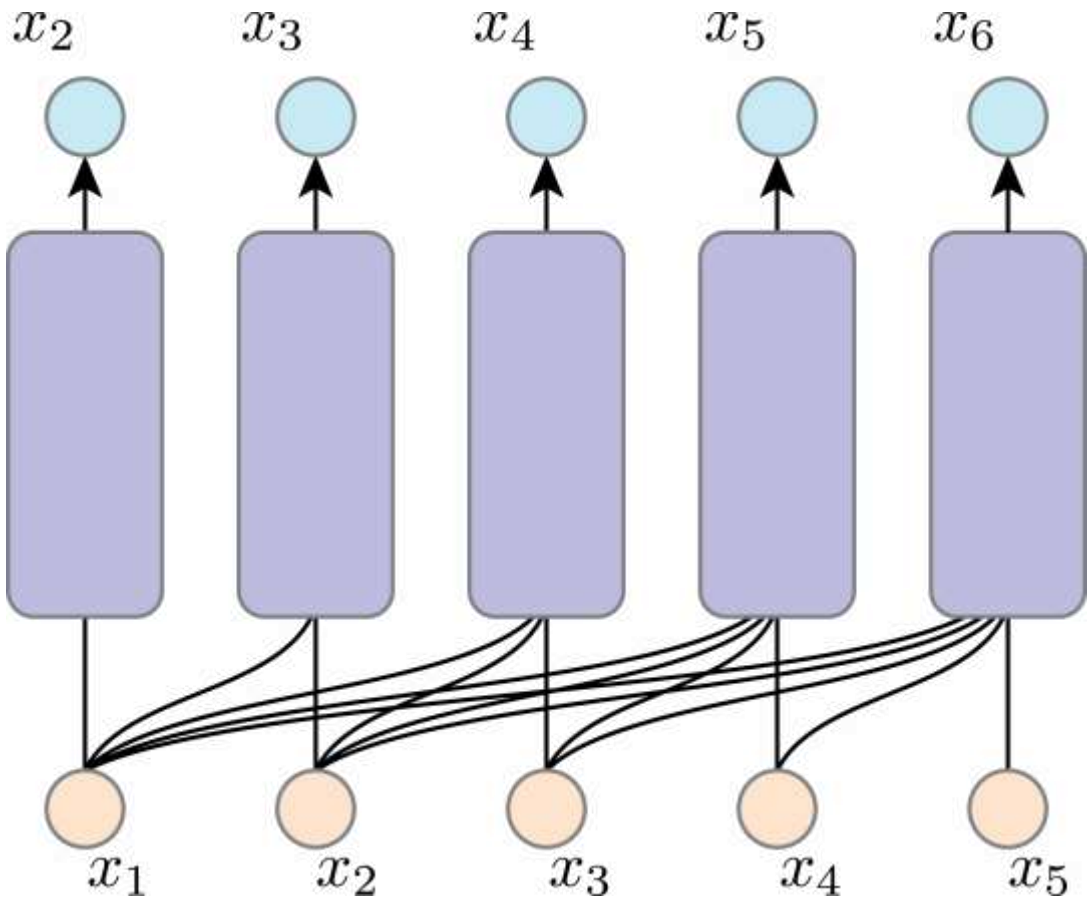


Autoregression is not architecture-specific

This figure implements this formulation:

$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$

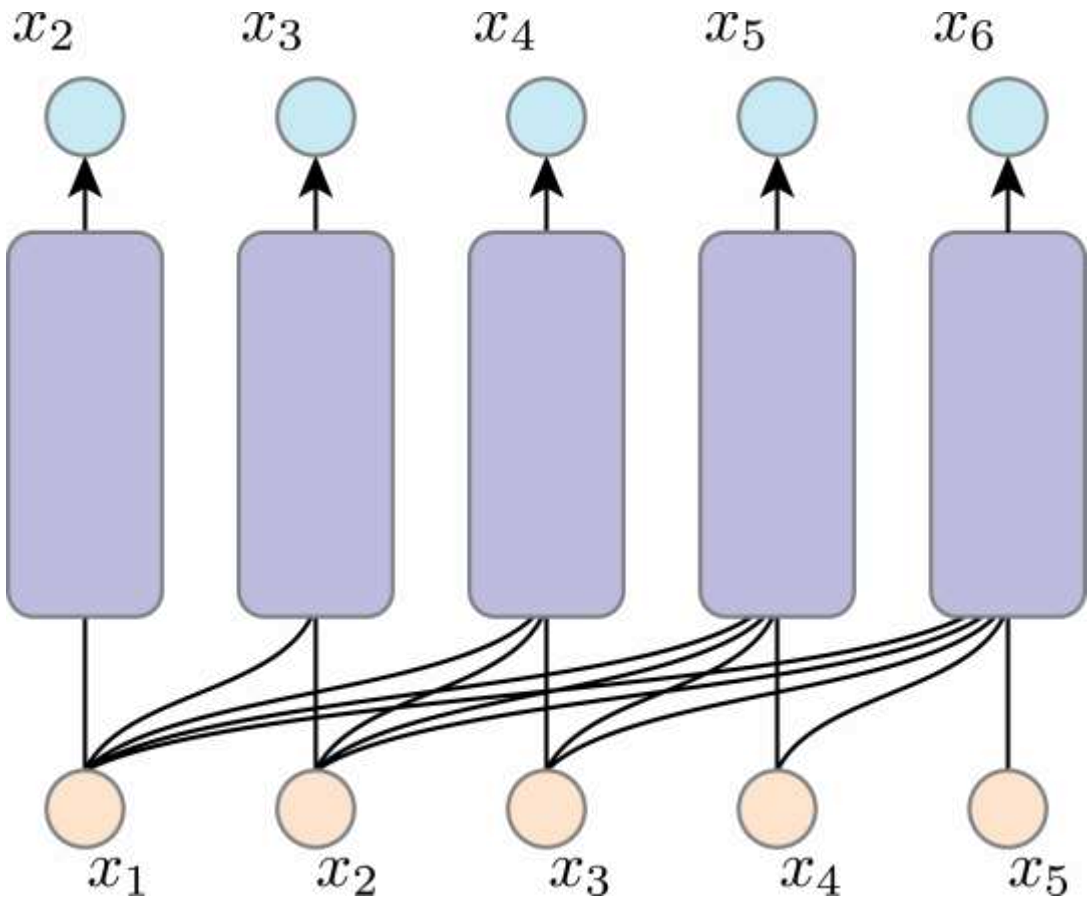
Can we do this efficiently?



Autoregression w/ Shared Computation

This figure implements this formulation:

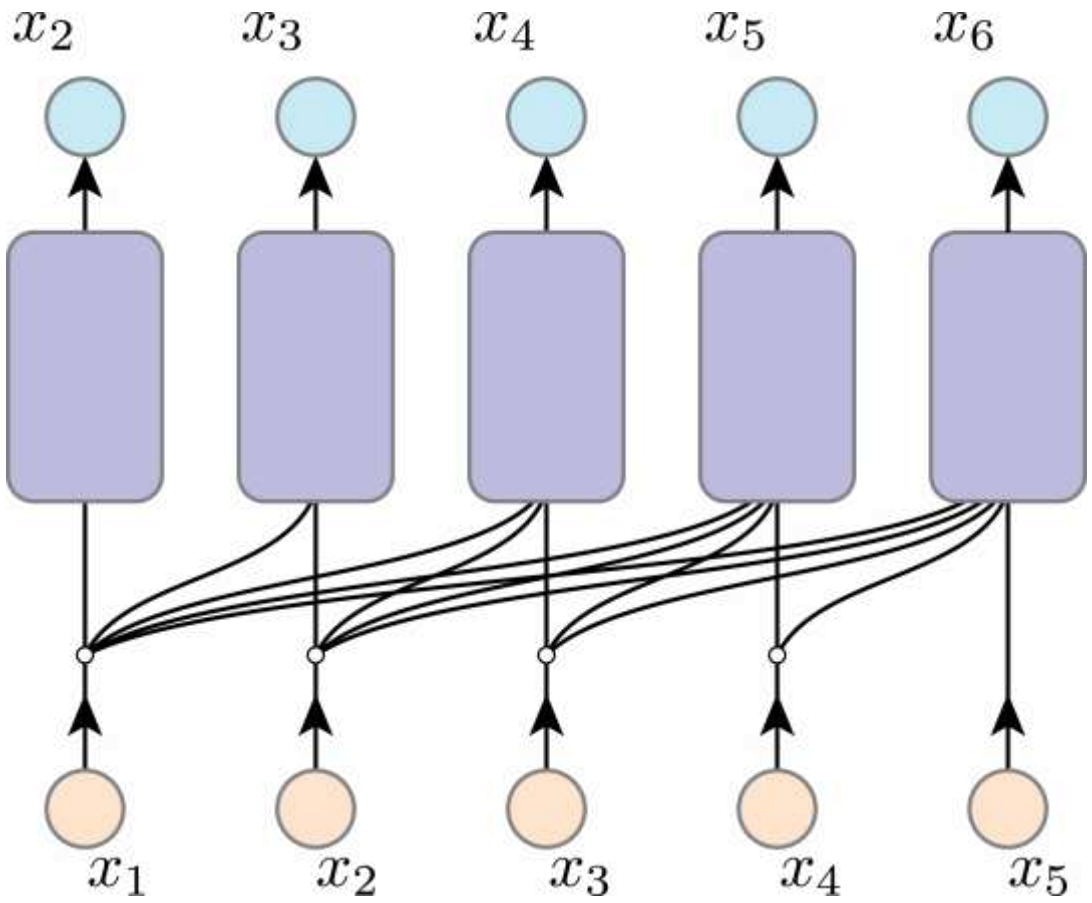
$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$



Autoregression w/ Shared Computation

This figure implements this formulation:

$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$

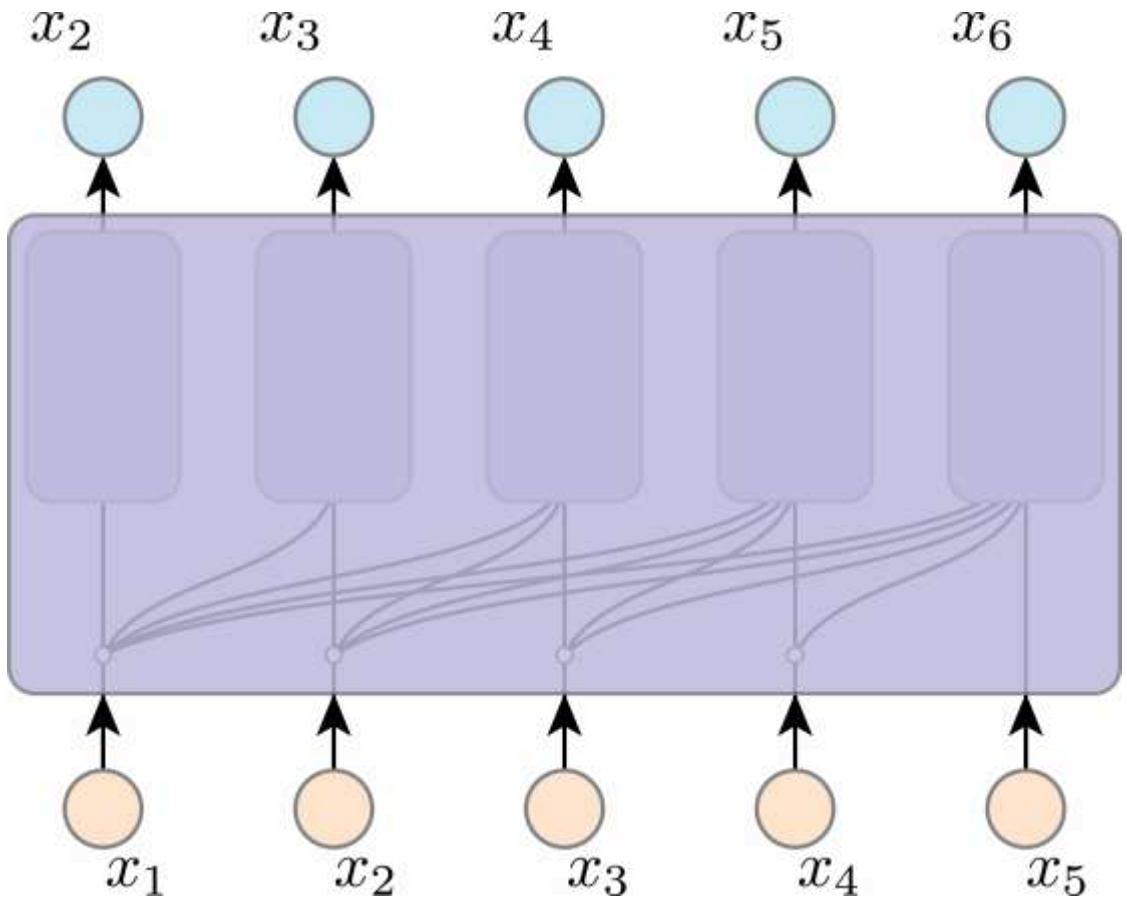


(this figure is equivalent to previous one)

Autoregression w/ Shared Computation

This figure implements this formulation:

$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$



Autoregression w/ Shared Computation

We can implement:

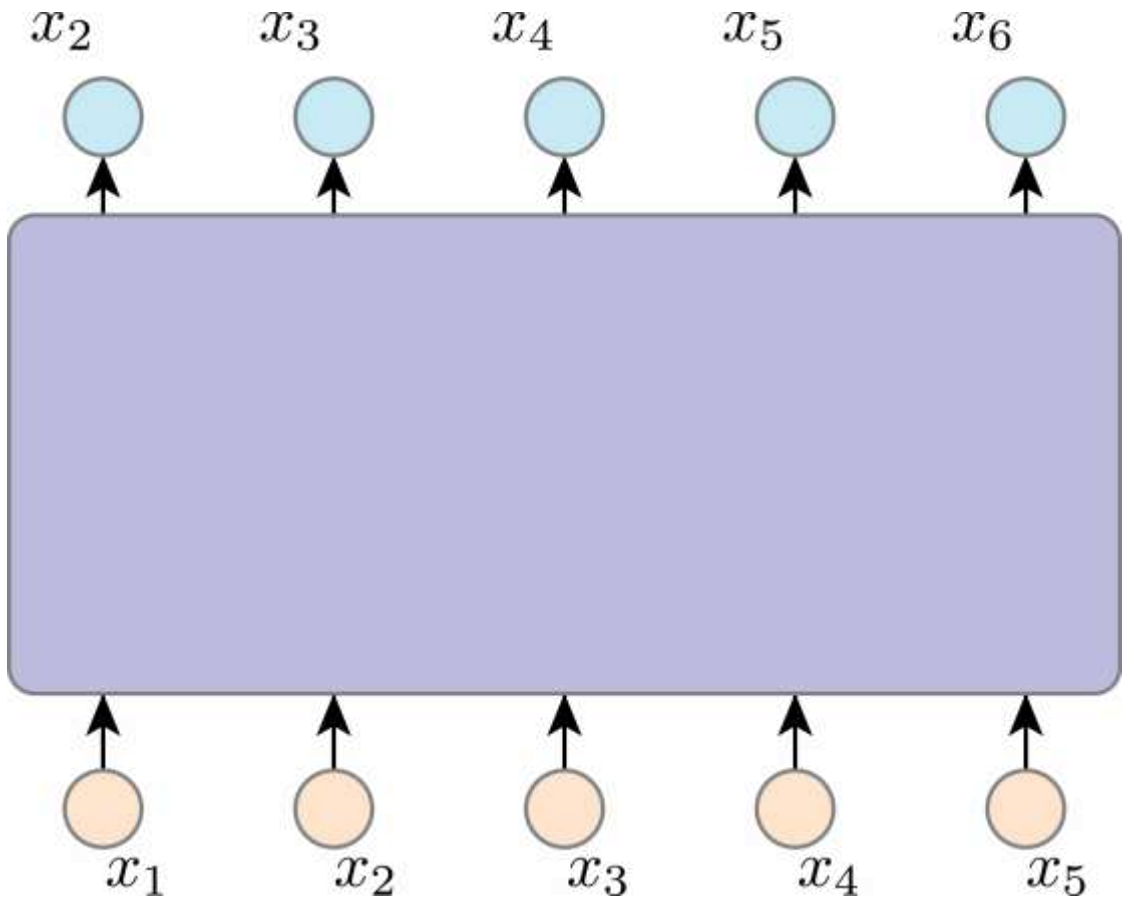
$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$

... by one network, with:

- shared architecture
- shared weights
- shared **computation**

if:

- output x_i **not** depend on x_j for any $j \geq i$



Autoregression w/ Shared Computation

We can implement:

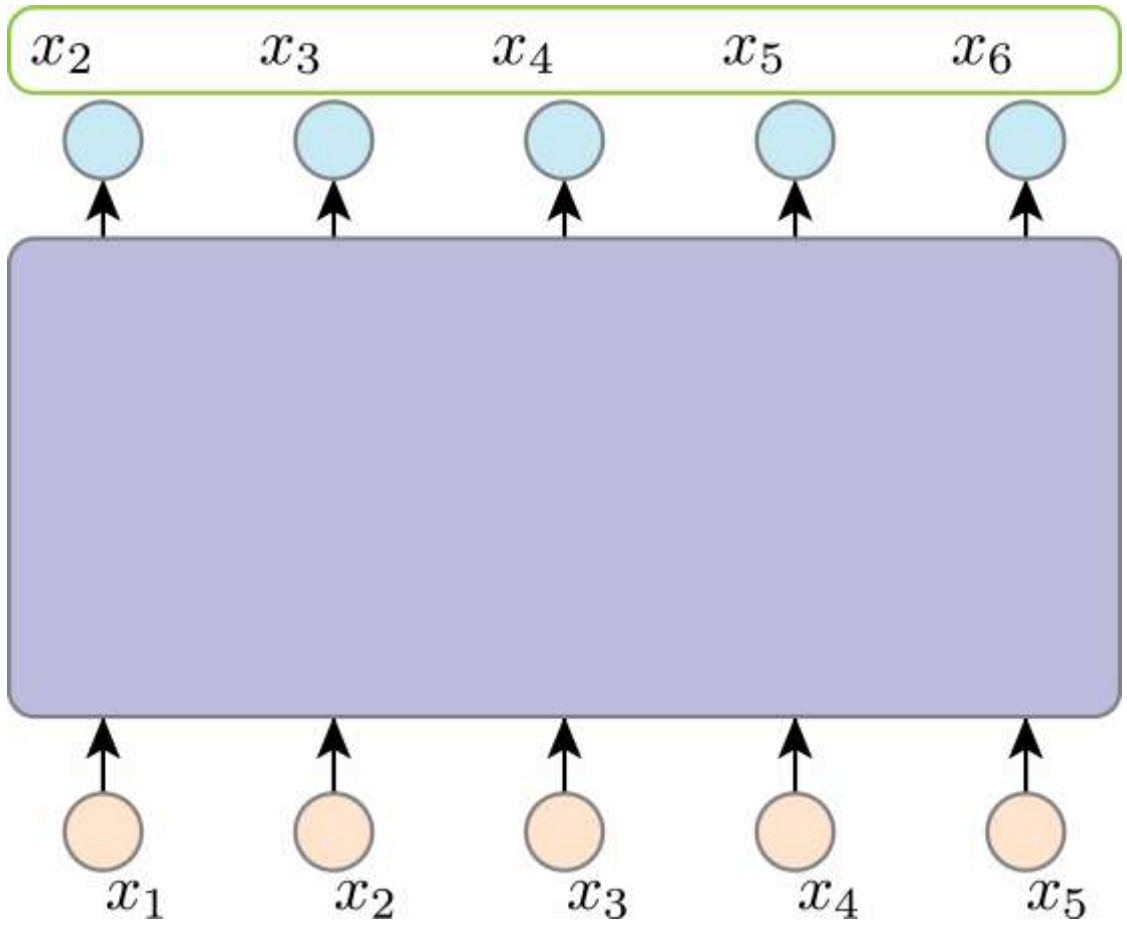
$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$

... by one network, with:

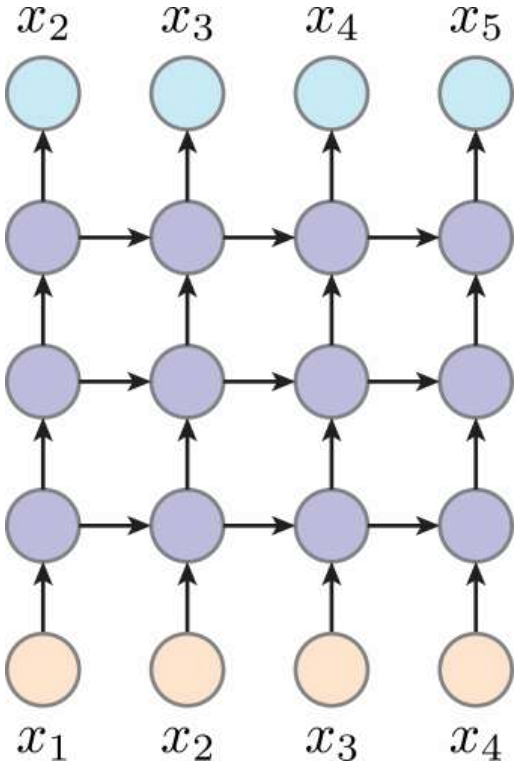
- shared architecture
- shared weights
- shared **computation**

if:

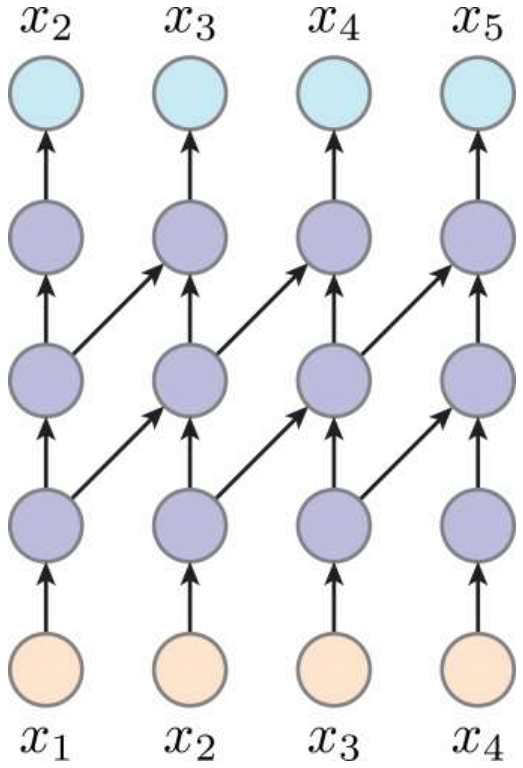
- output x_i **not** depend on x_j for any $j \geq i$



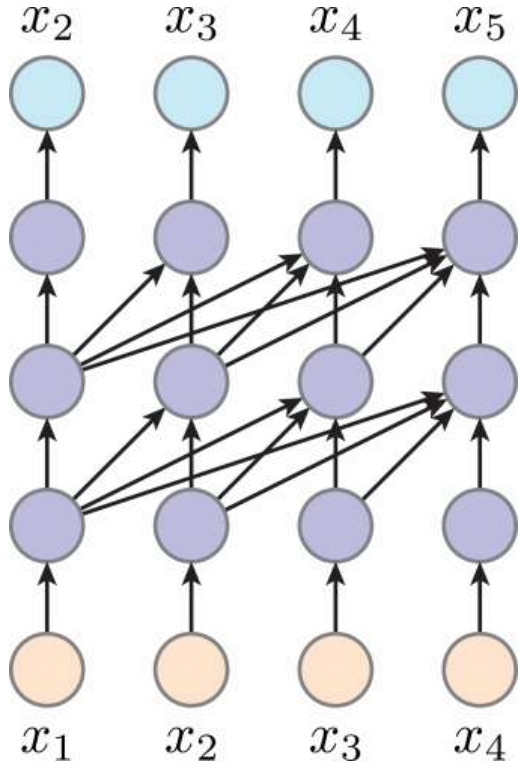
Common Architectures for Autoregression



RNN



CNN

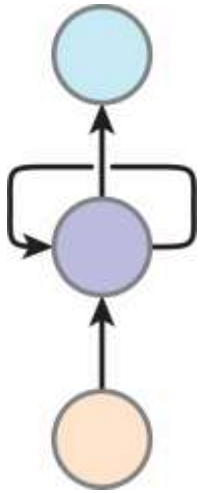


Attention

See also my lecture at 6.8300: “11: Sequence Modeling”: https://drive.google.com/file/d/1lOYsyImXl3caWlsplfThyuvgz-

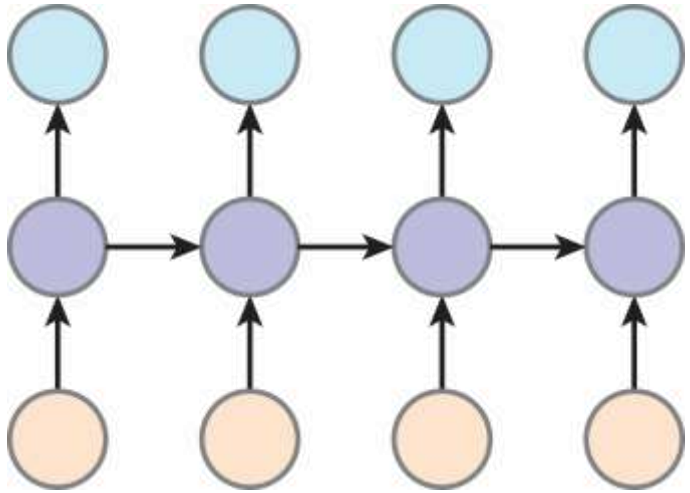
Brief: Recurrent Neural Network (RNN) for AR

one RNN unit



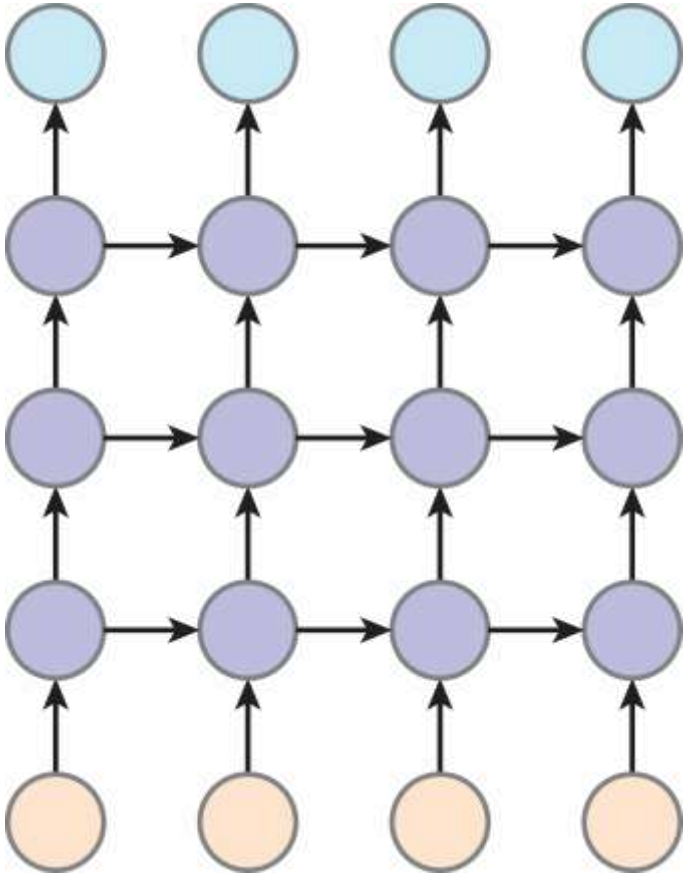
Brief: Recurrent Neural Network (RNN) for AR

unfold in “time”



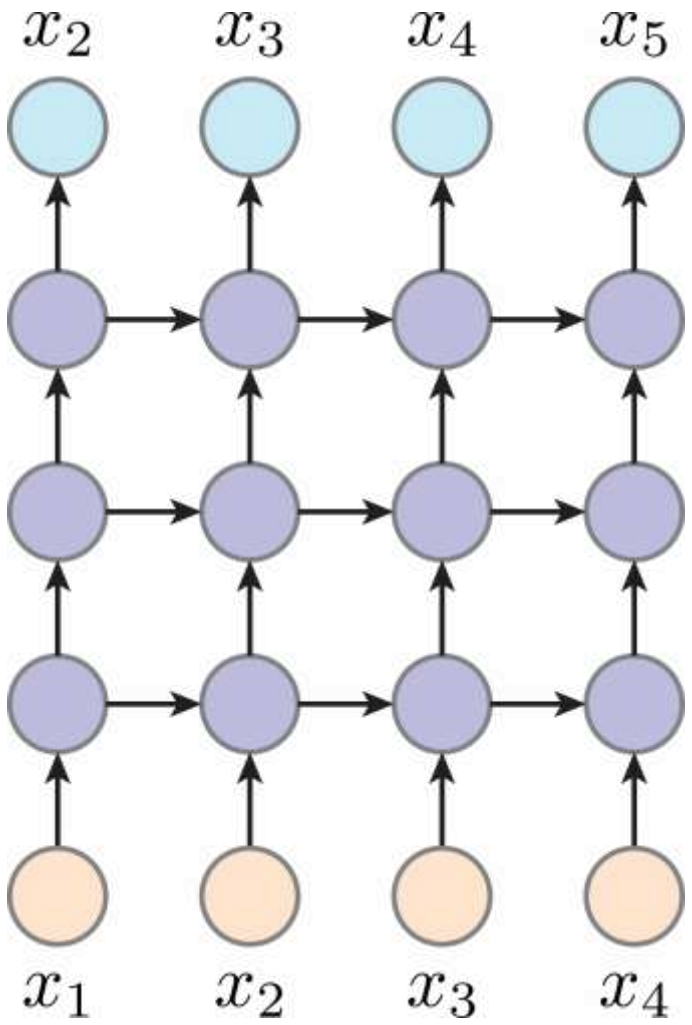
Brief: Recurrent Neural Network (RNN) for AR

go deep

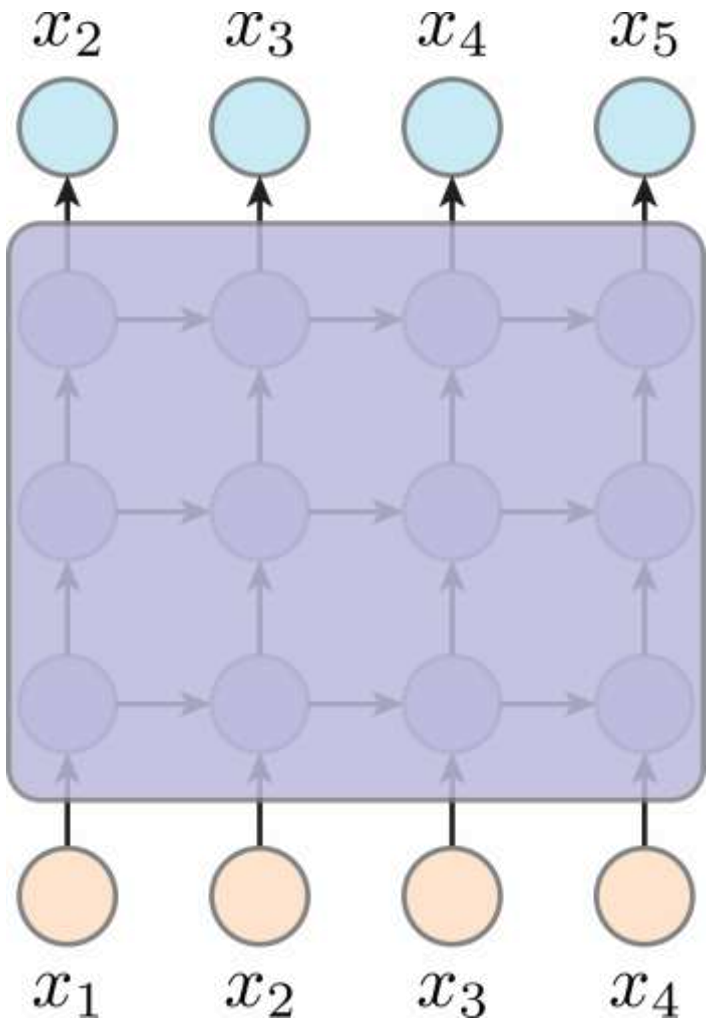


Brief: Recurrent Neural Network (RNN) for AR

shift target by one step

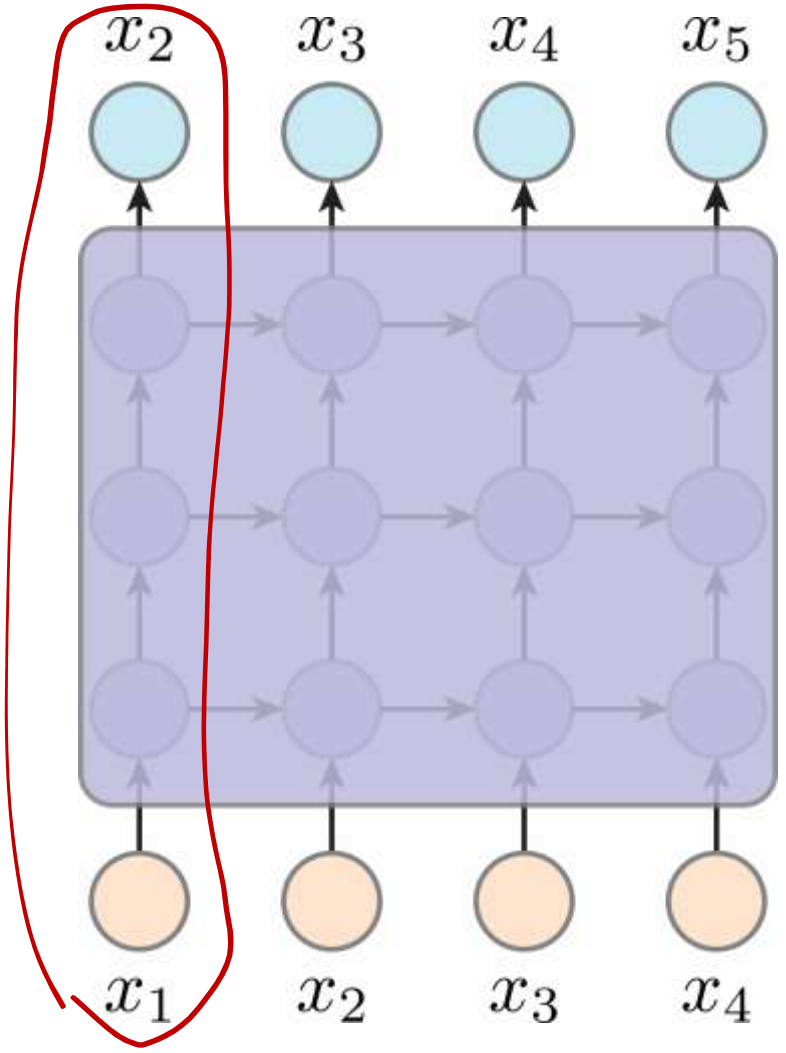


Brief: Recurrent Neural Network (RNN) for AR



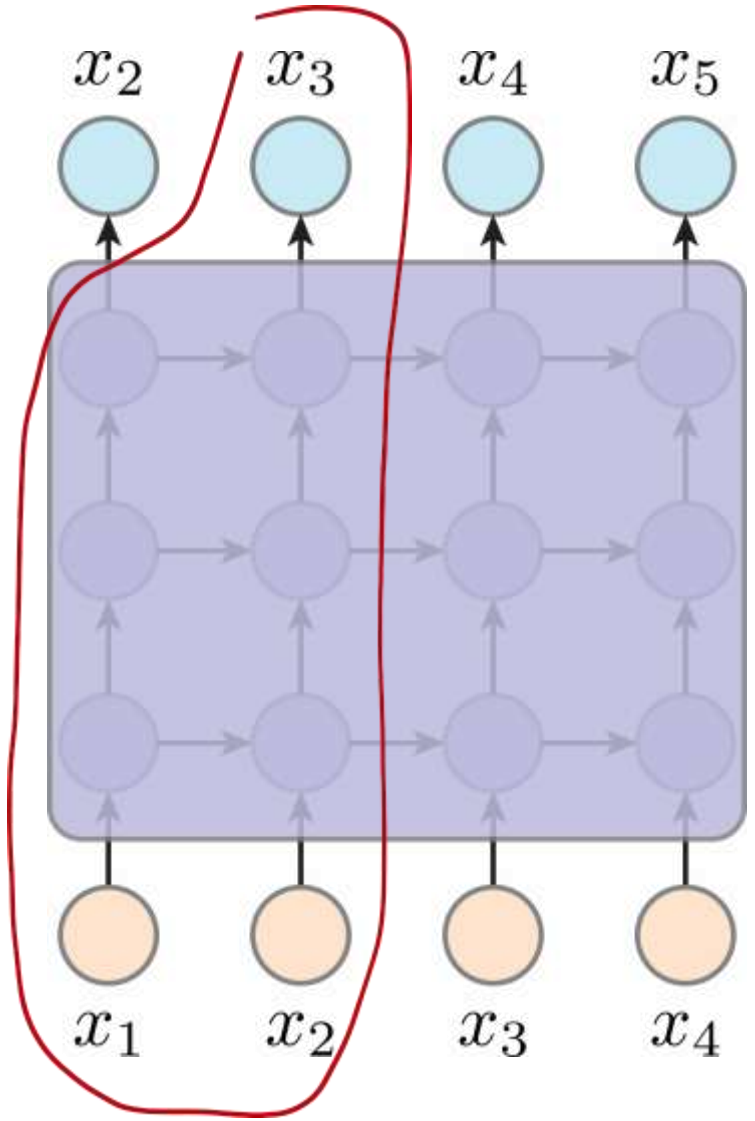
Brief: Recurrent Neural Network (RNN) for AR

$p(x_2 \mid x_1)$



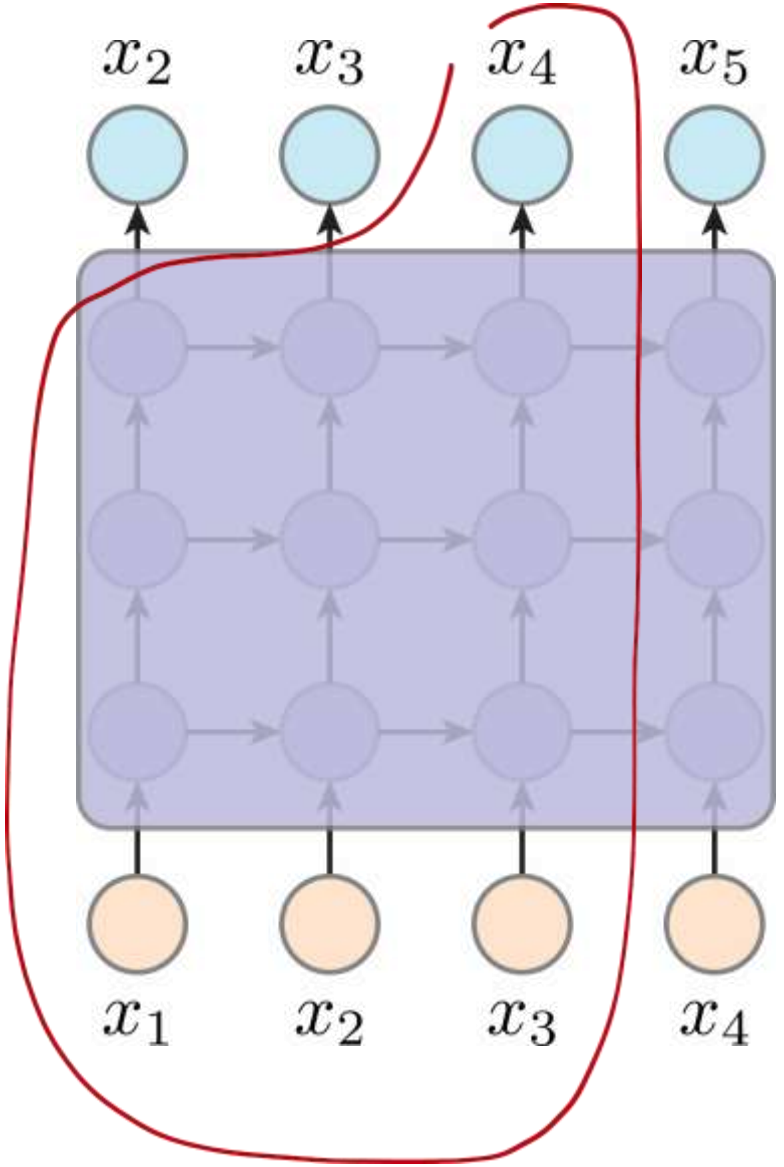
Brief: Recurrent Neural Network (RNN) for AR

$p(x_3 \mid x_{1,2})$



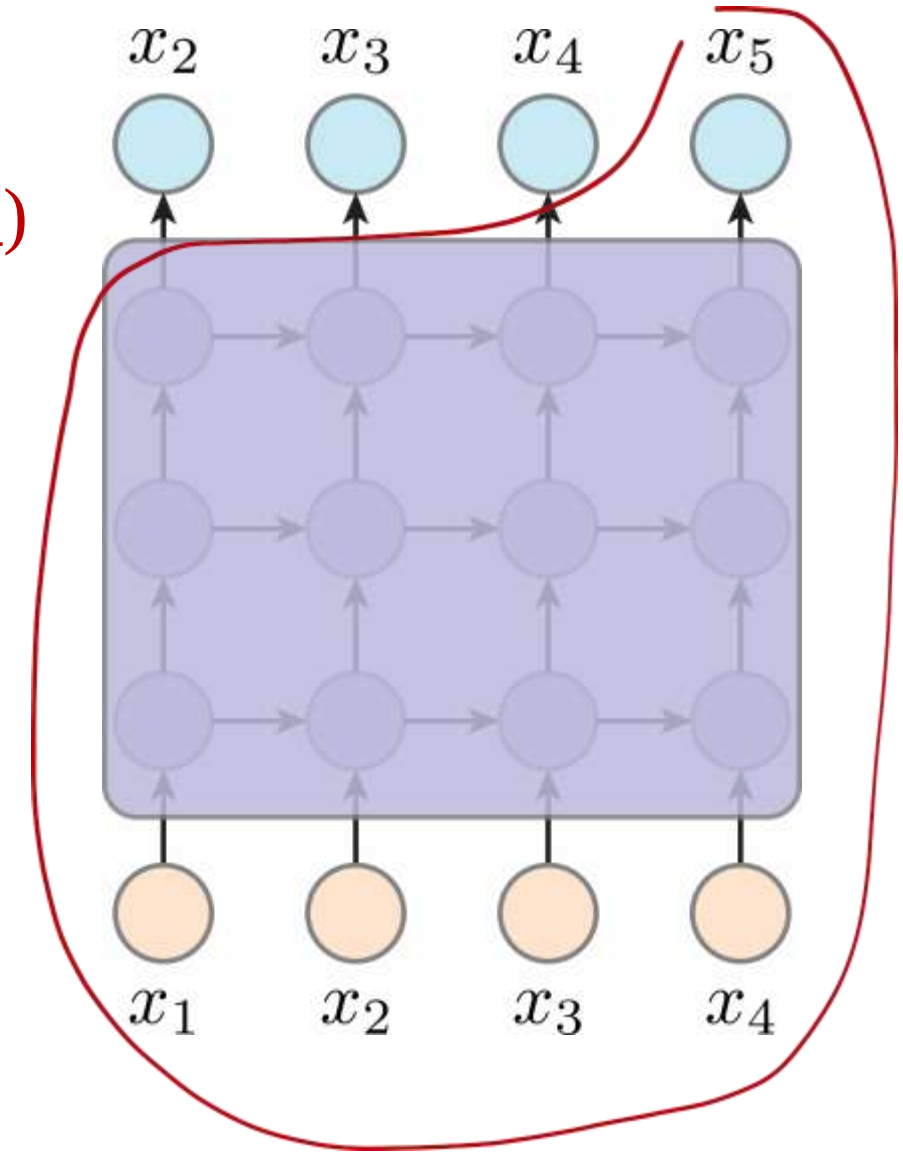
Brief: Recurrent Neural Network (RNN) for AR

$p(x_4 \mid x_{1,2,3})$

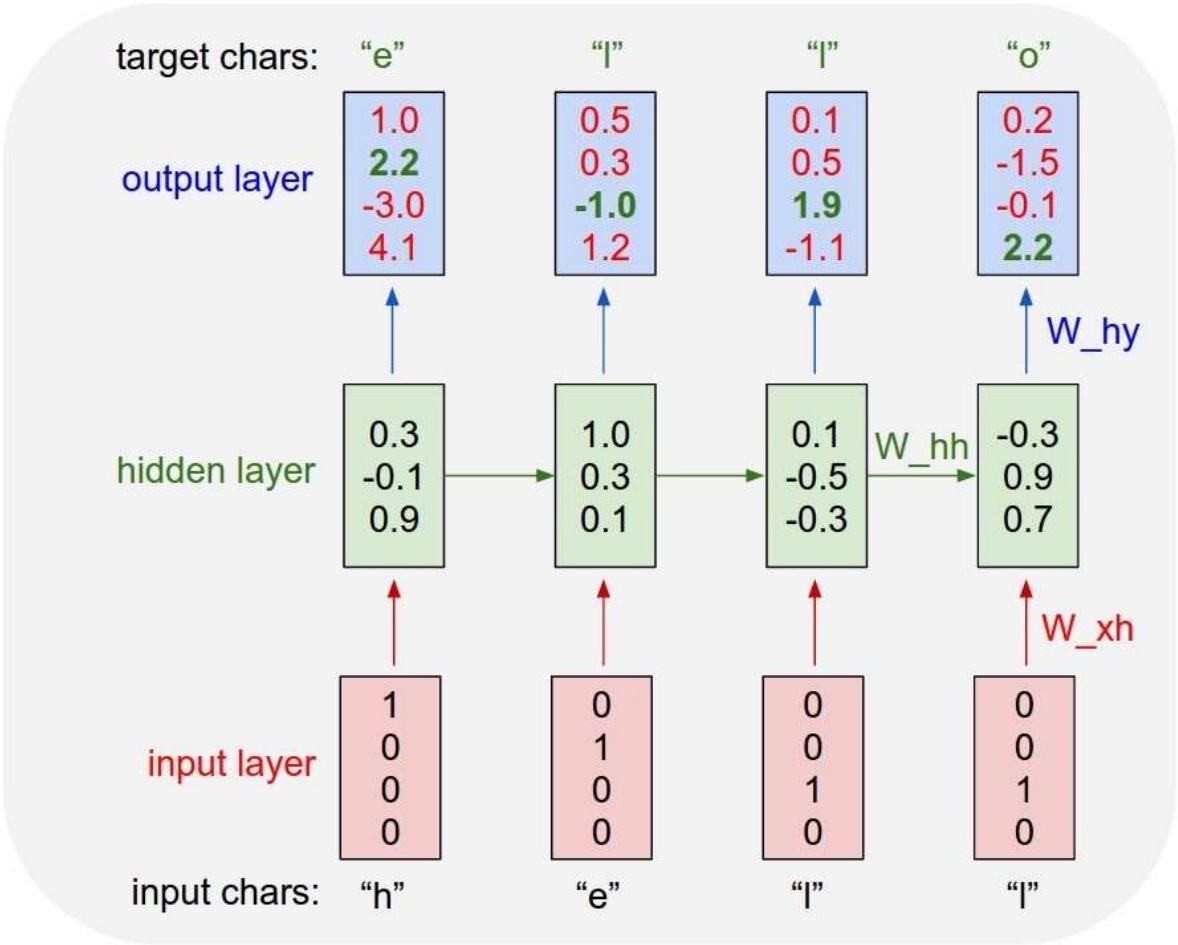


Brief: Recurrent Neural Network (RNN) for AR

$p(x_5 \mid x_{1,2,3,4})$



Example: Char-RNN



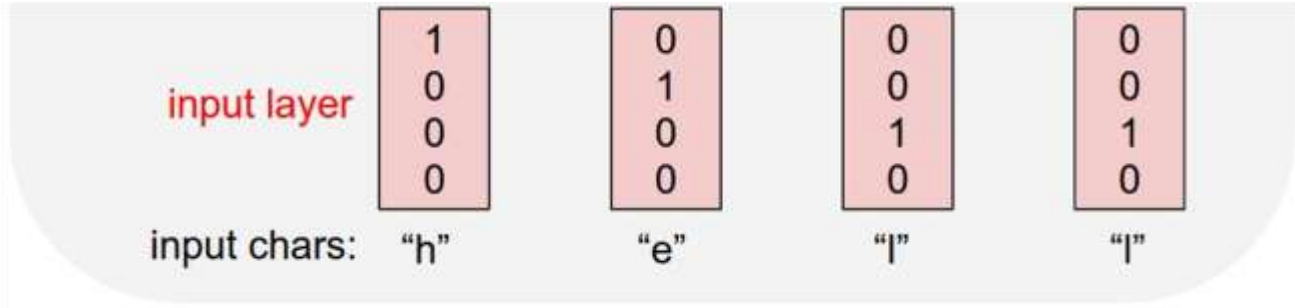
Andrej Karpathy. "The Unreasonable Effectiveness of Recurrent Neural Networks", blog post, 2015

Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



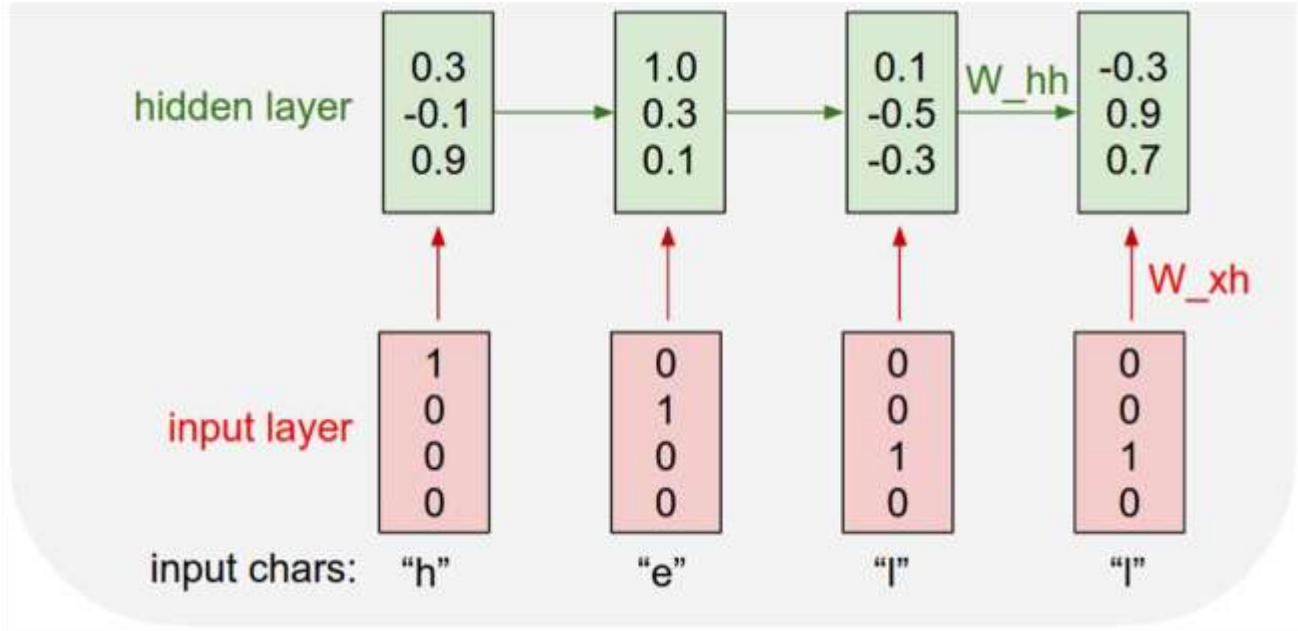
Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



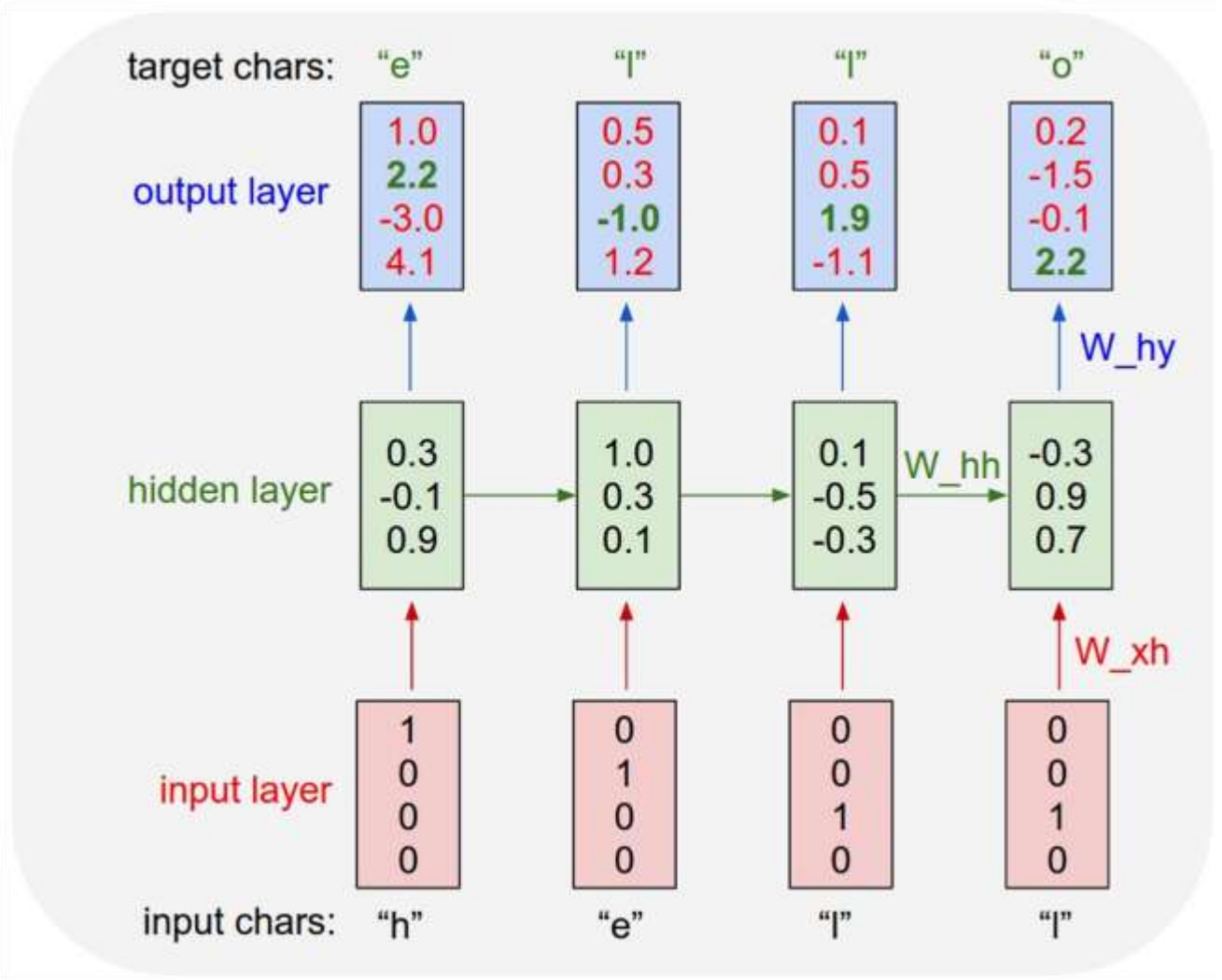
Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



Language Modeling

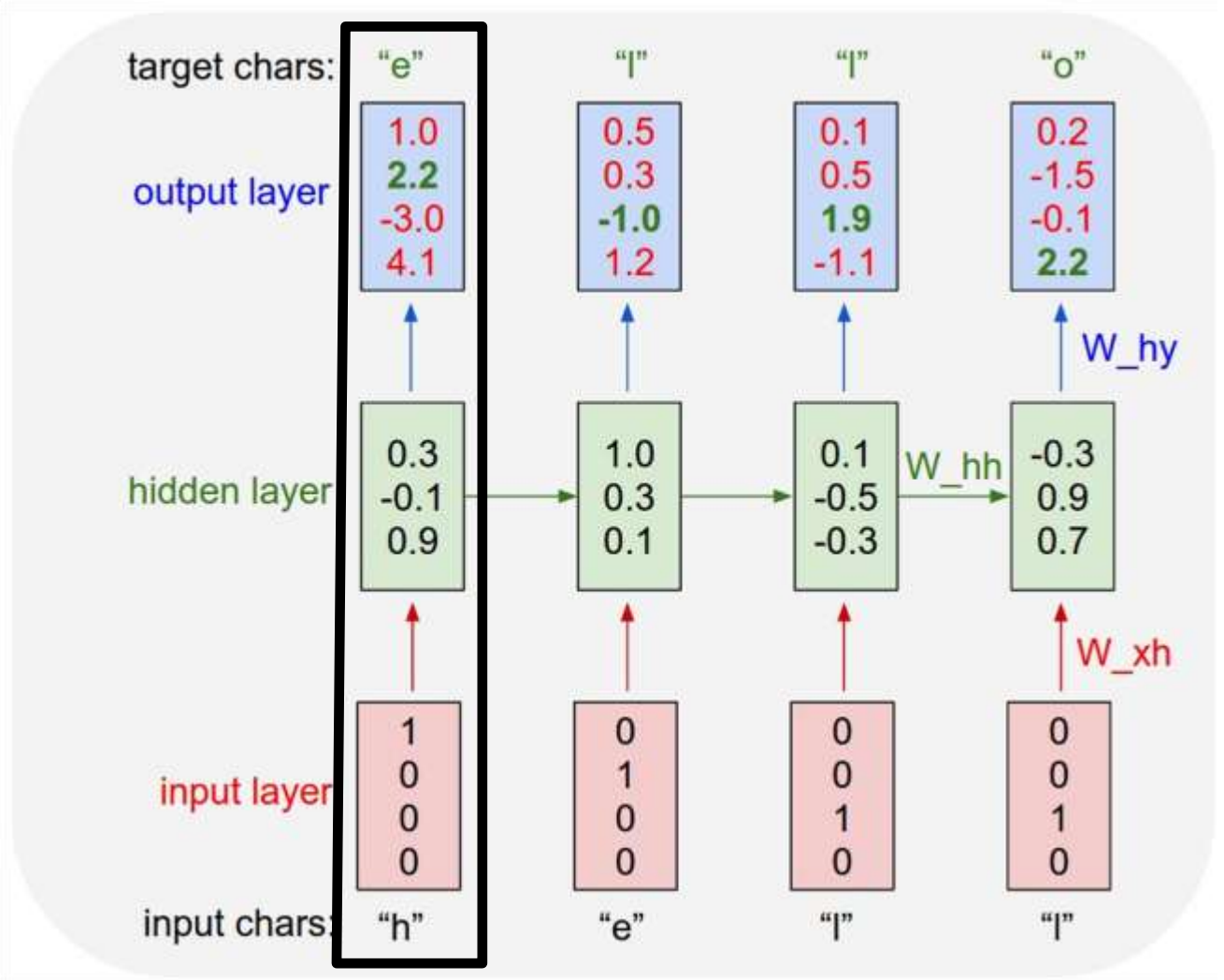
Given characters 1, 2, …, t-1, model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

Given “h”, predict “e”



Language Modeling

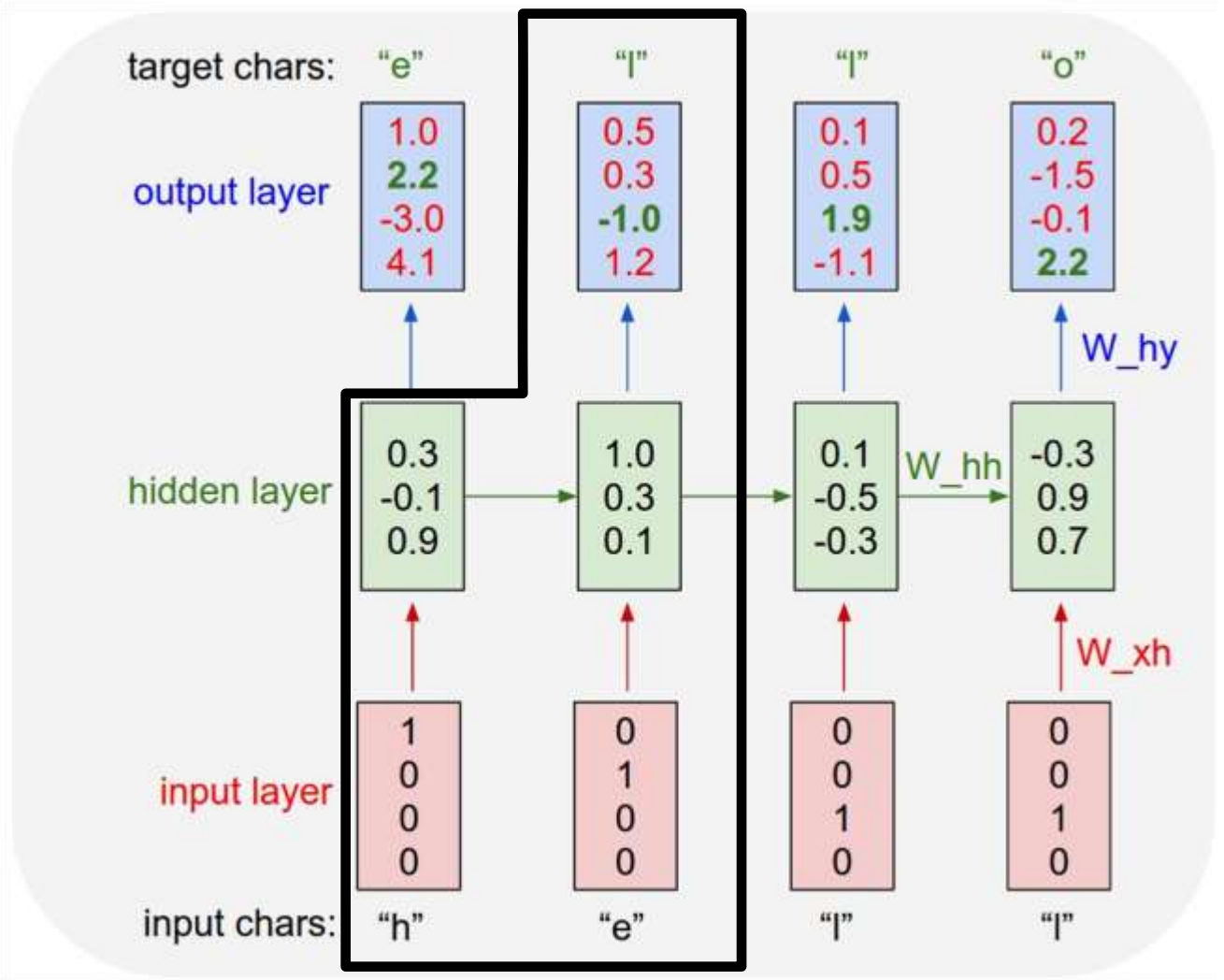
Given characters 1, 2, …, t-1, model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

Given “he”, predict “l”



Language Modeling

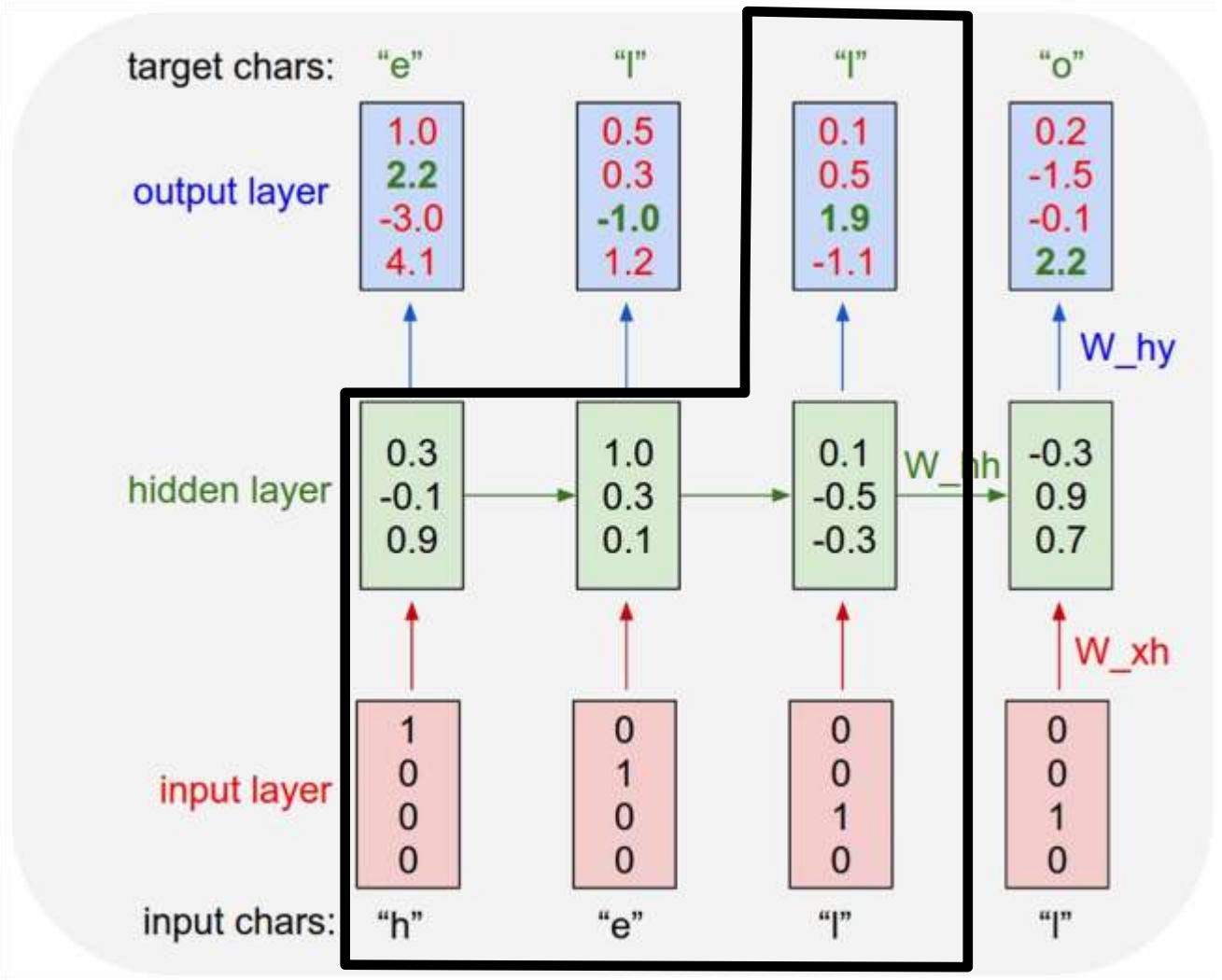
Given characters 1, 2, …, t-1, model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

Given “hel”, predict “l”



Language Modeling

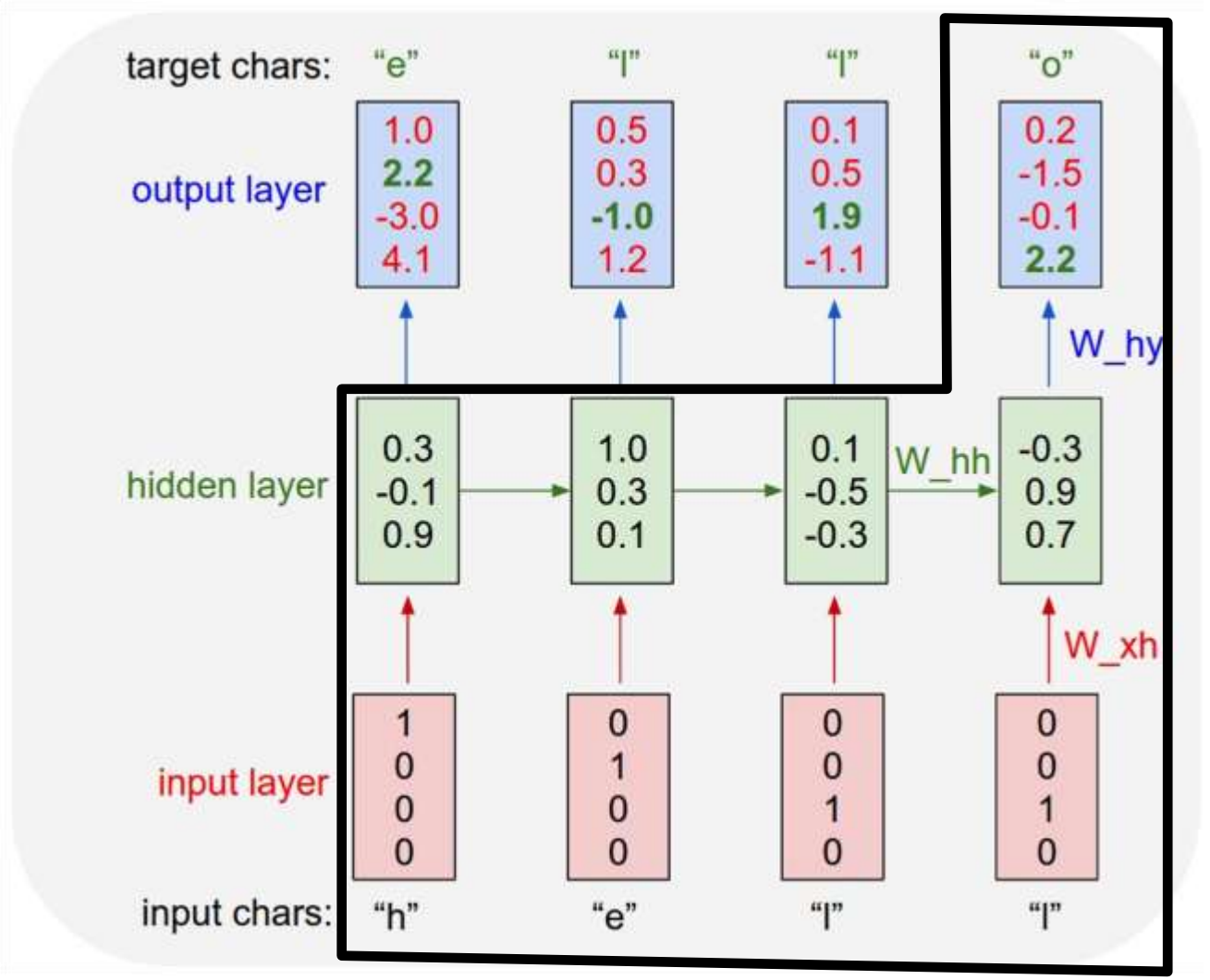
Given characters 1, 2, …, t-1, model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

Given “hell”, predict “o”

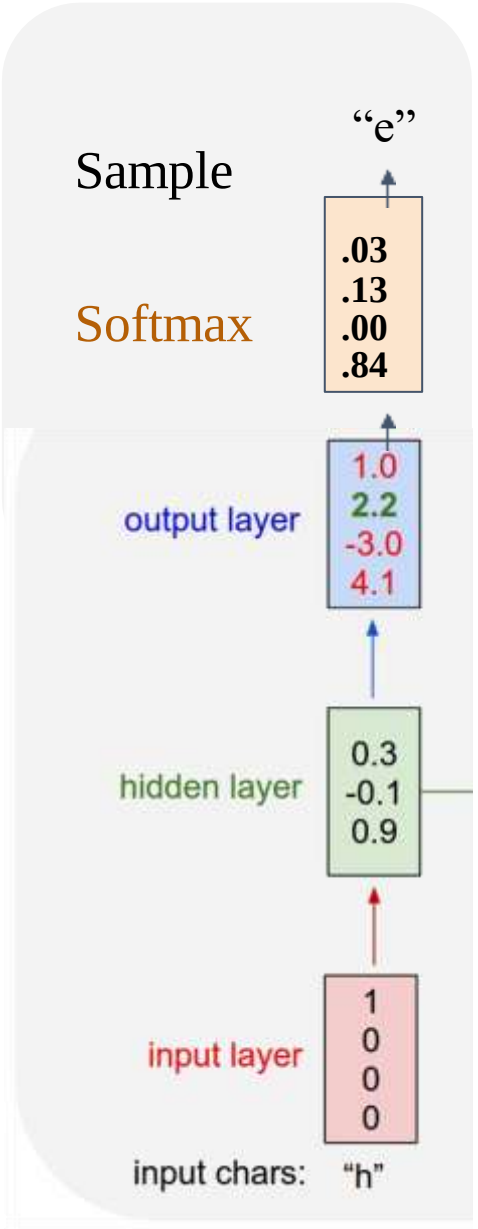


Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

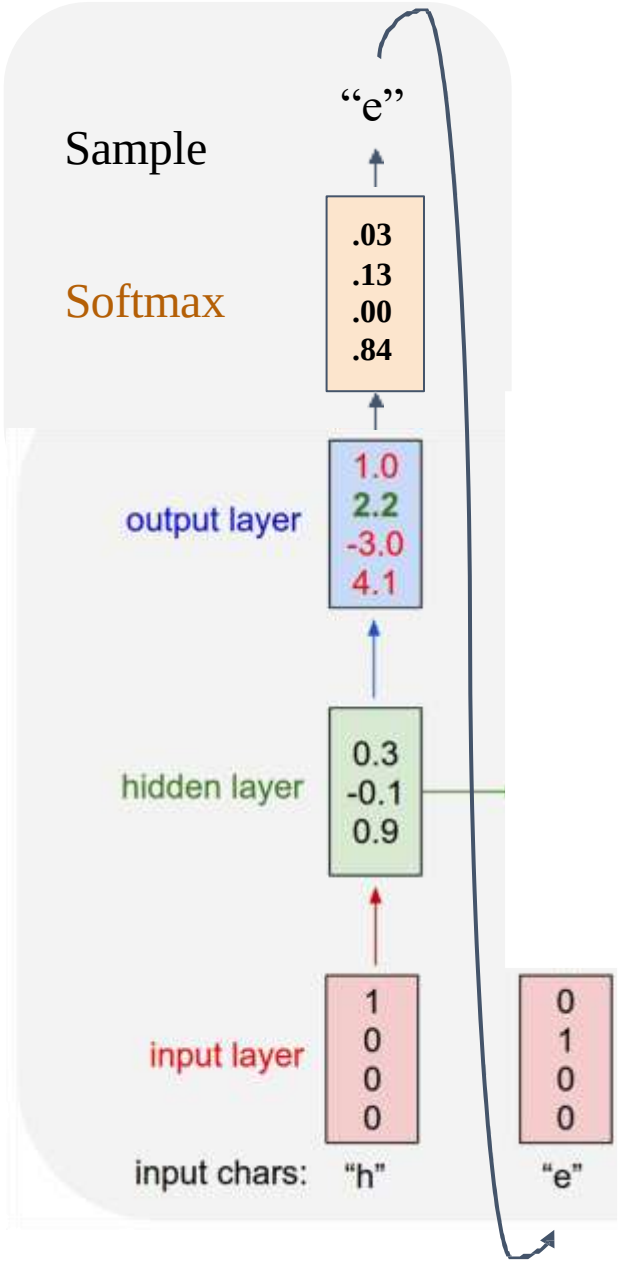


Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

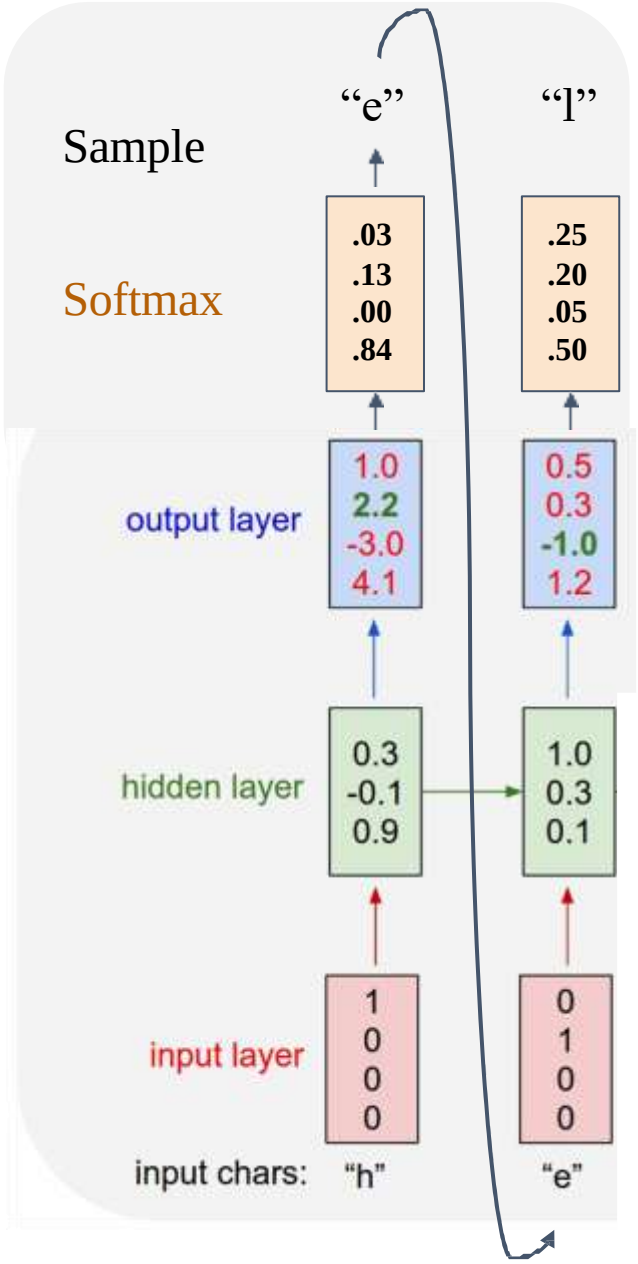


Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

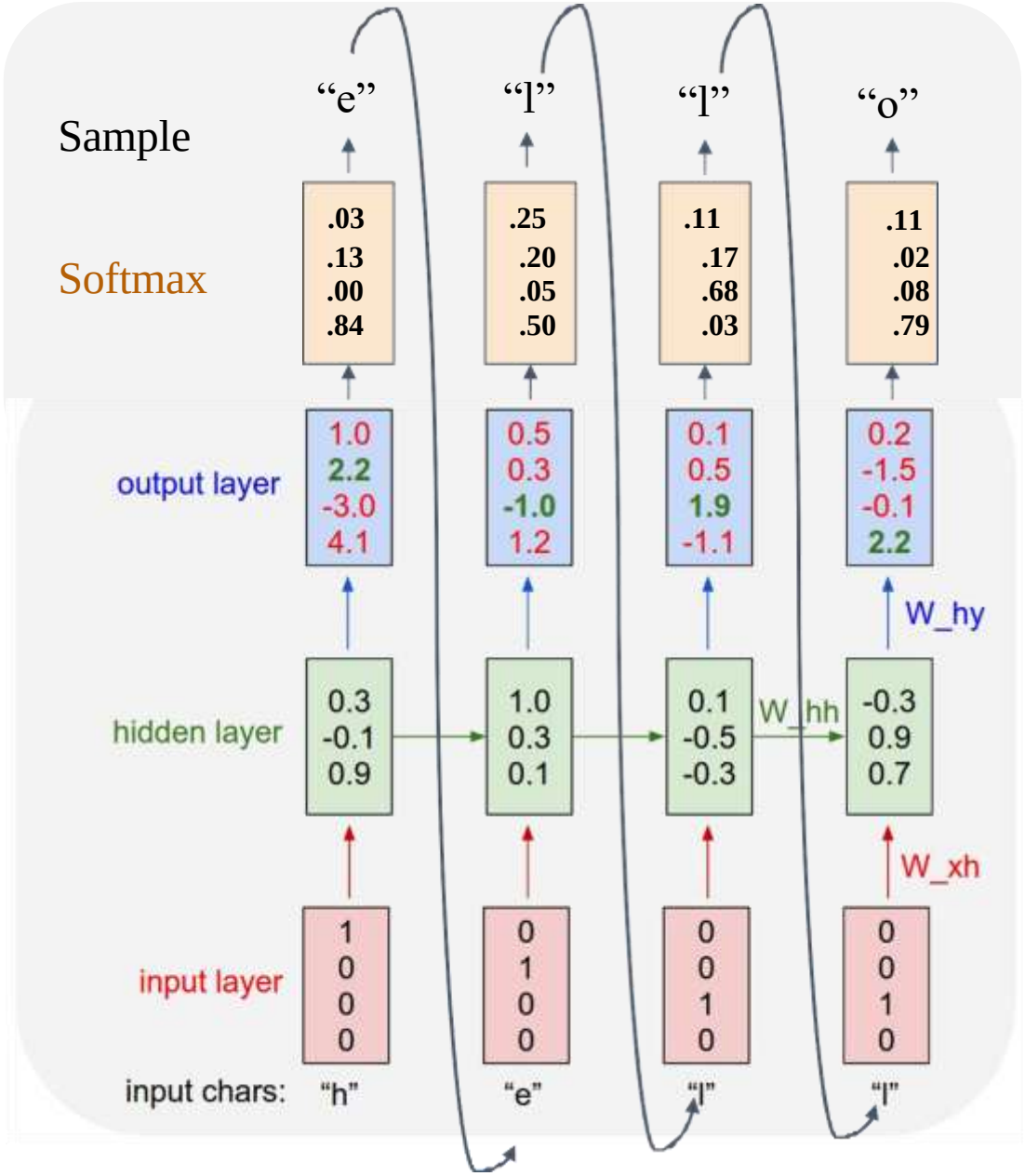


Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

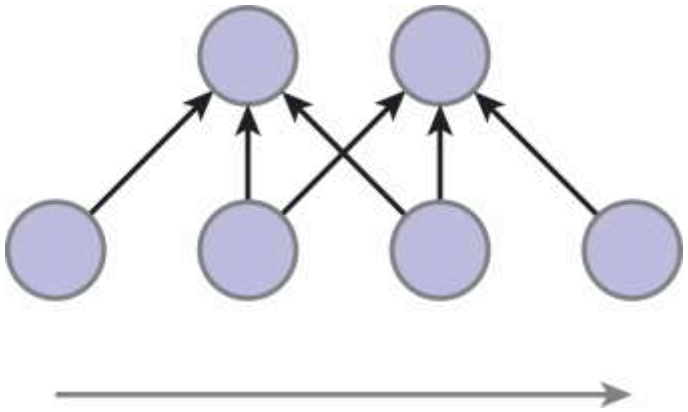
Training sequence: ”hello”

Vocabulary: [h, e, l, o]



Brief: Convolutional Neural Network (CNN) for AR

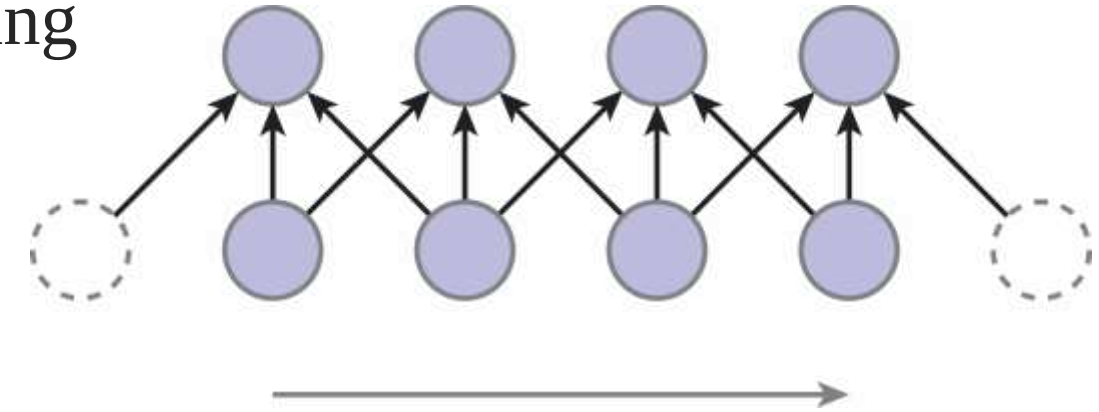
1-D
convolution



“time”
axis

Brief: Convolutional Neural Network (CNN) for AR

w/ padding

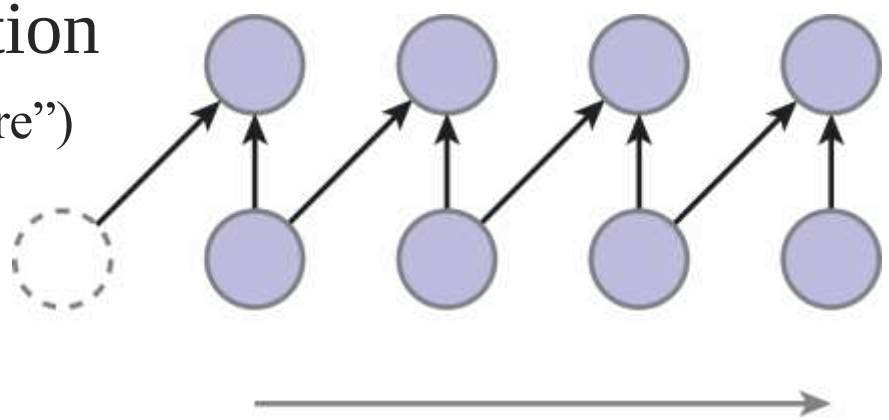


“time” axis

Brief: Convolutional Neural Network (CNN) for AR

causal convolution

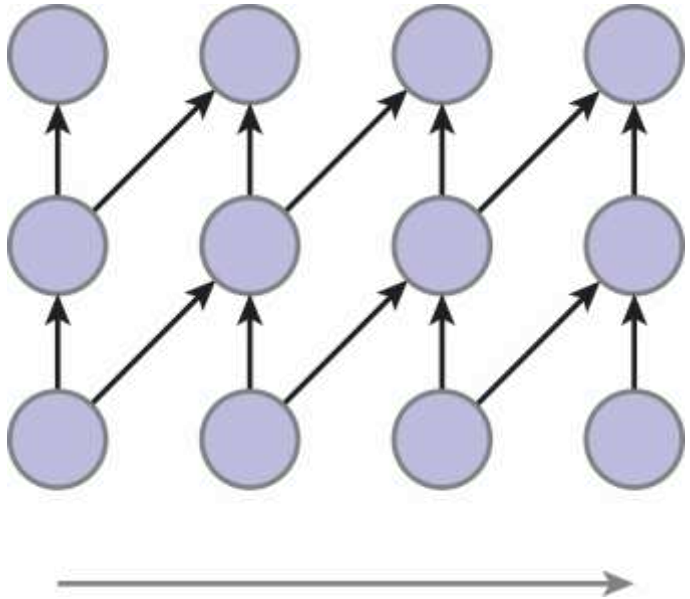
(not depend on “future”)



“time” axis

Brief: Convolutional Neural Network (CNN) for AR

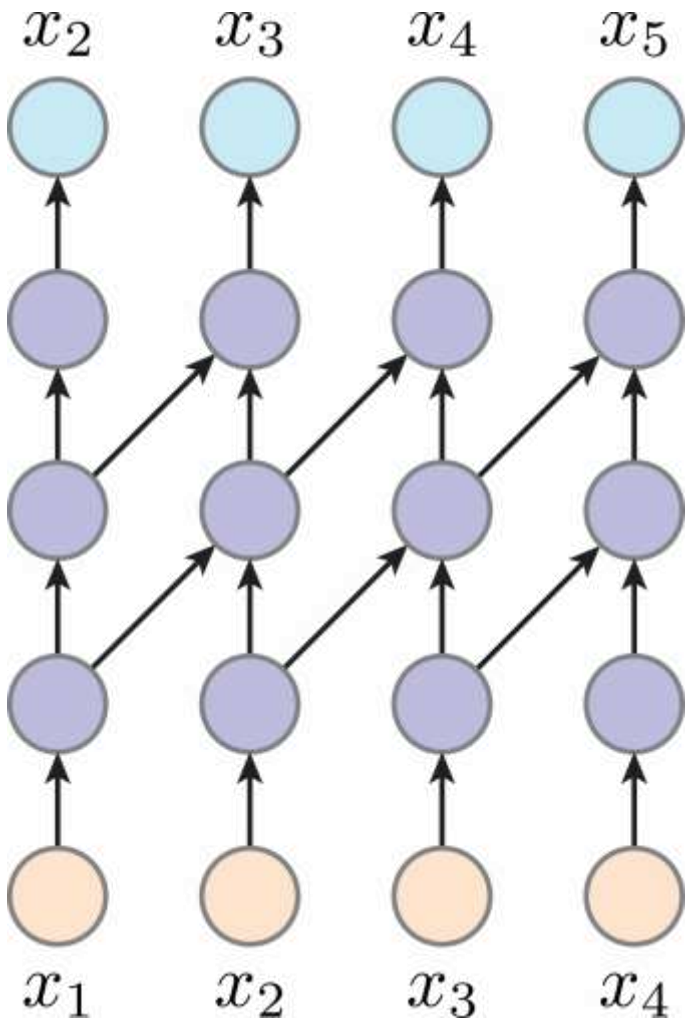
go deep



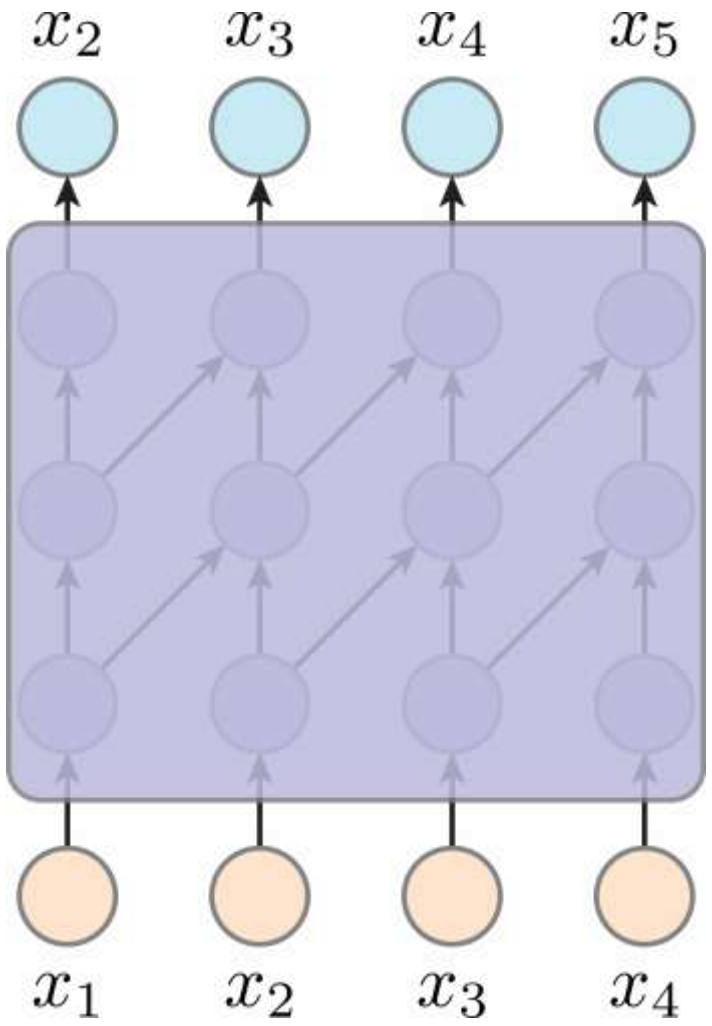
“time”
axis

Brief: Convolutional Neural Network (CNN) for AR

shift target by one step

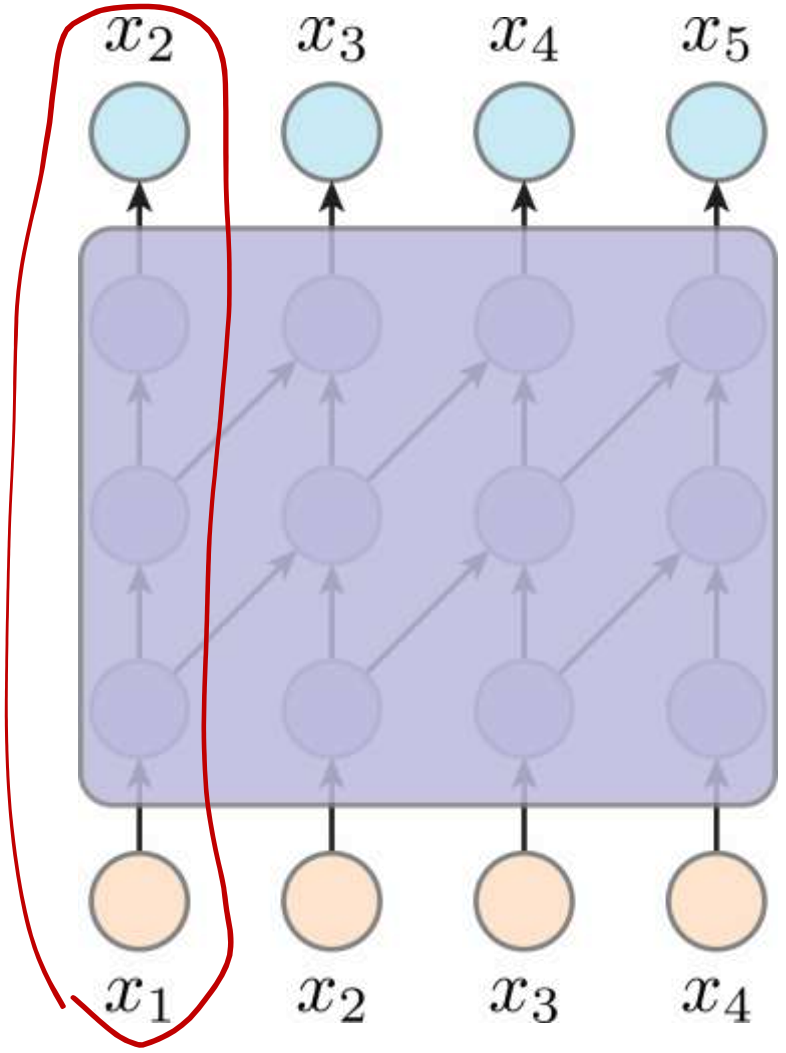


Brief: Convolutional Neural Network (CNN) for AR



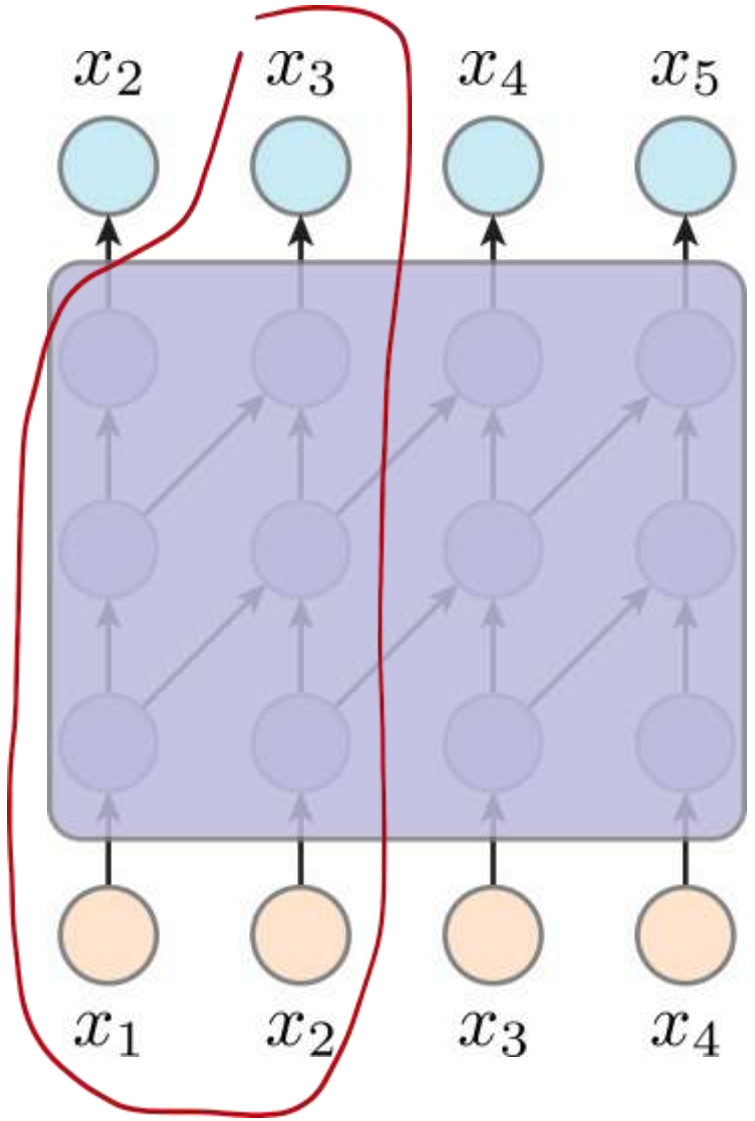
Brief: Convolutional Neural Network (CNN) for AR

$p(x_2 \mid x_1)$



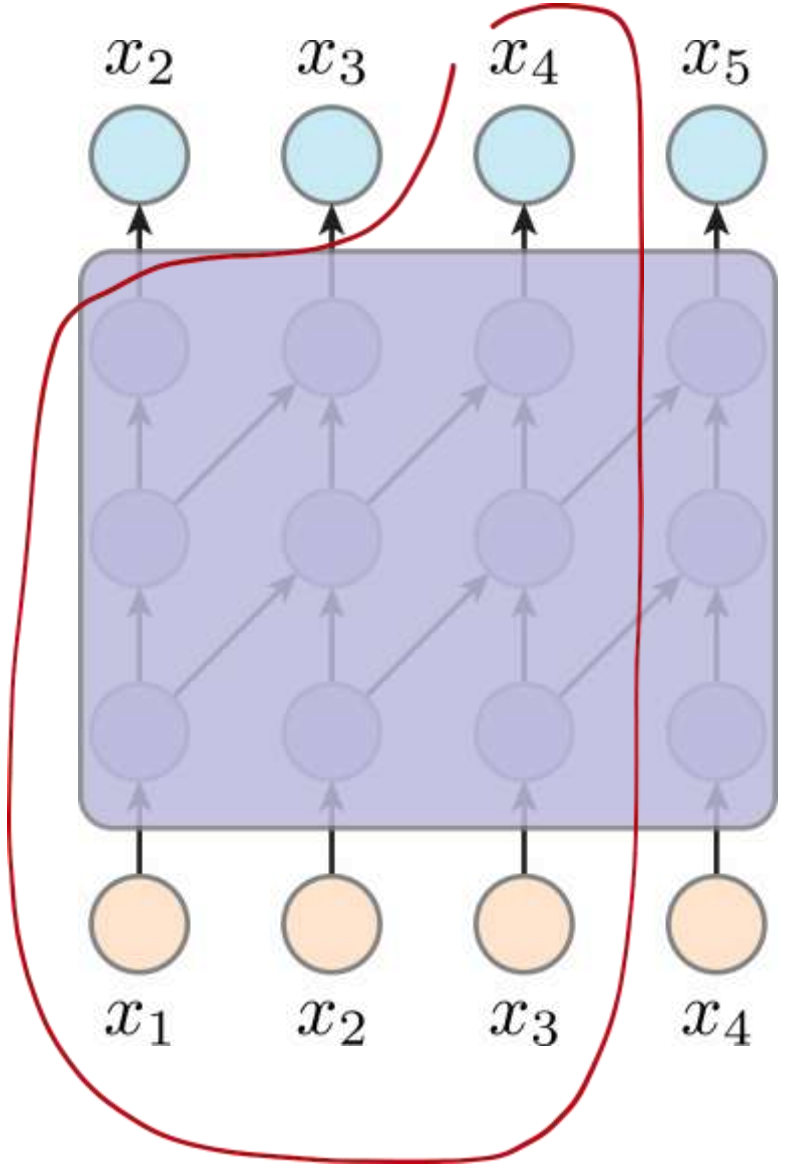
Brief: Convolutional Neural Network (CNN) for AR

$p(x_3 \mid x_{1,2})$



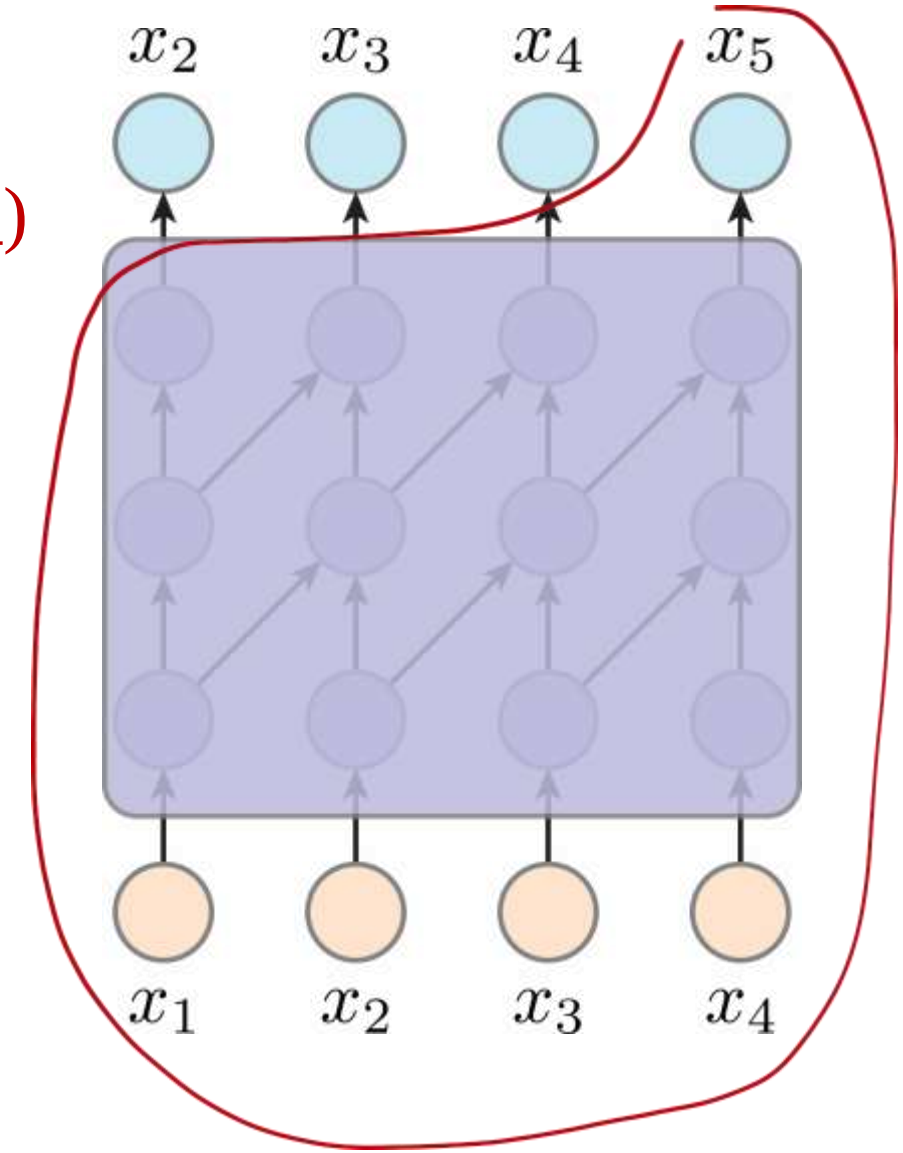
Brief: Convolutional Neural Network (CNN) for AR

$p(x_4 \mid x_{1,2,3})$

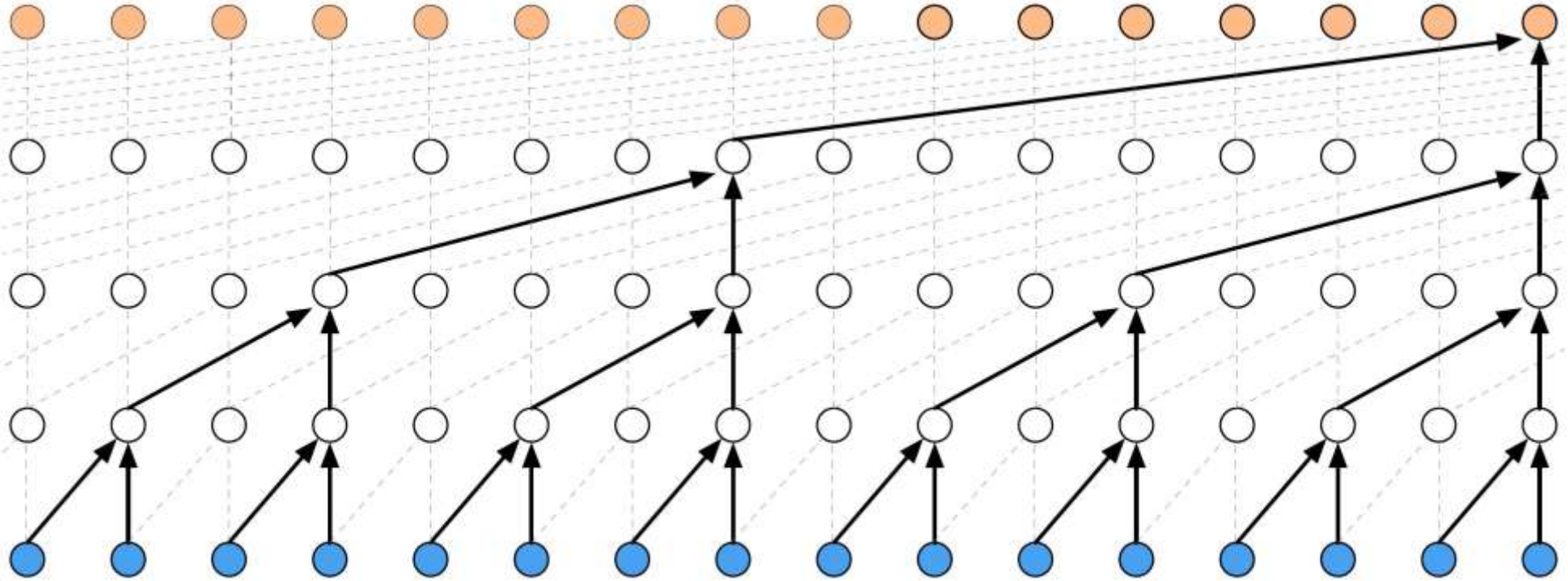


Brief: Convolutional Neural Network (CNN) for AR

$p(x_5 \mid x_{1,2,3,4})$



Example: WaveNet



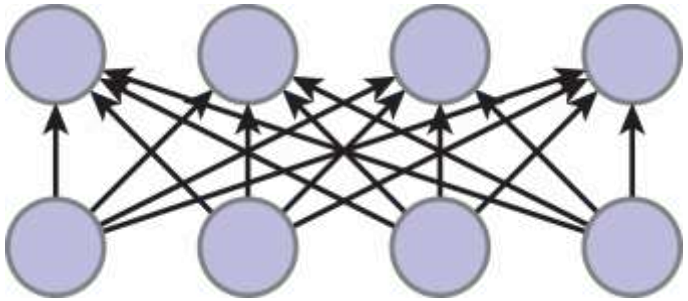
Audio generation with 1-D dilated causal conv

van den Oord, et al. “WaveNet: A Generative Model for Raw Audio”, 2016

Brief: Attention (Transformer) for AR

full attention

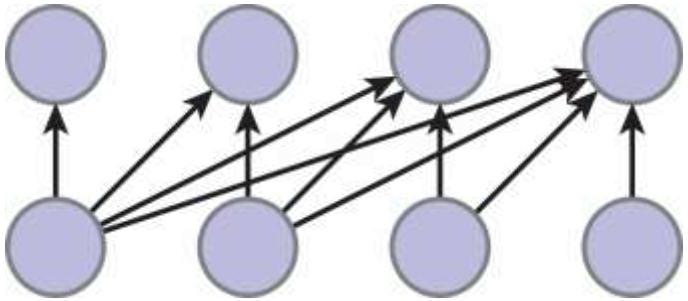
(every step sees all steps)



Brief: Attention (Transformer) for AR

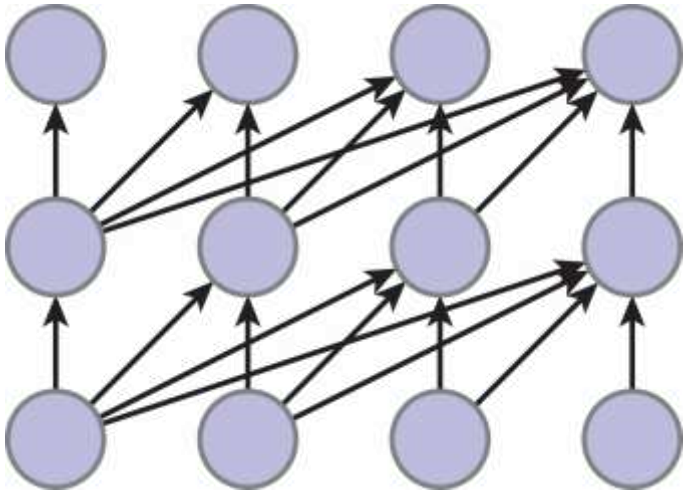
causal attention

(not depend on “future”)



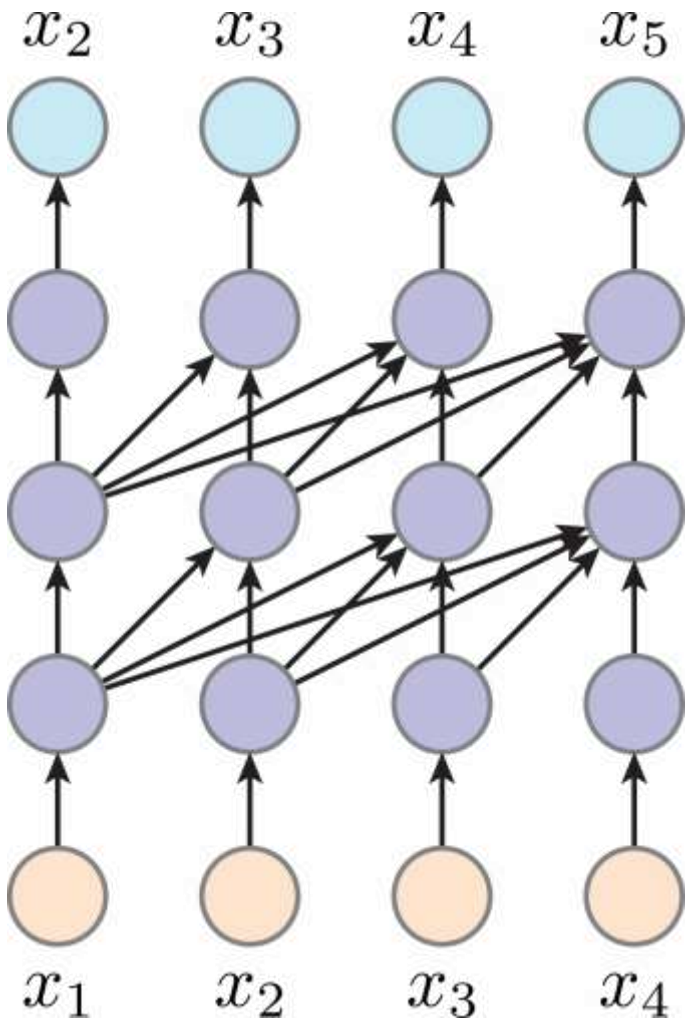
Brief: Attention (Transformer) for AR

go deep

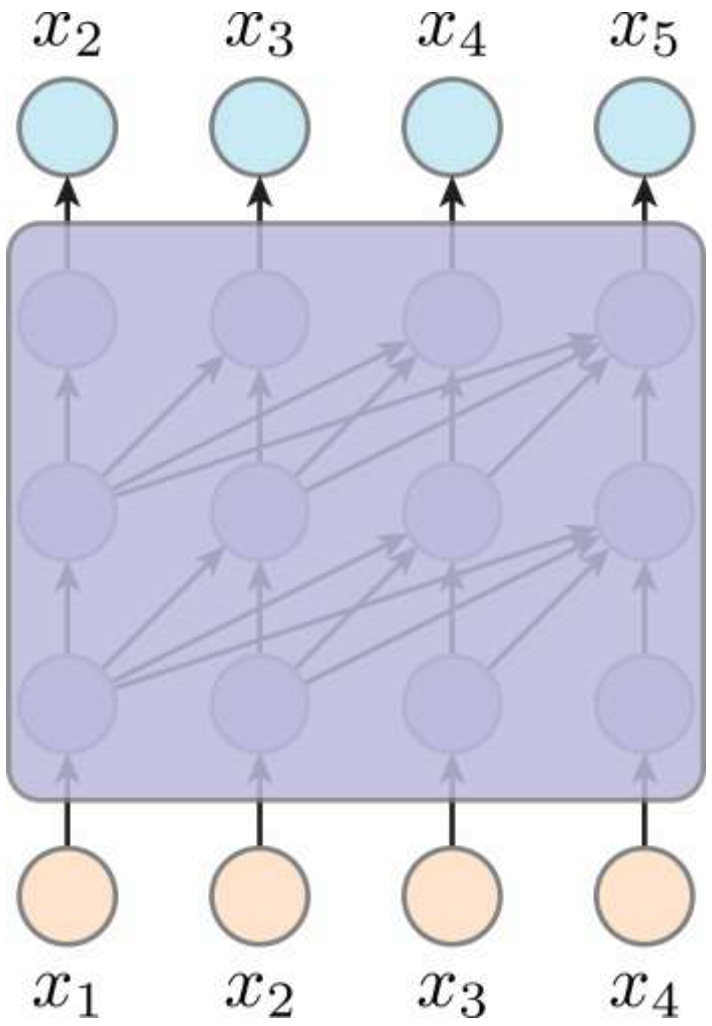


Brief: Attention (Transformer) for AR

shift target by one step

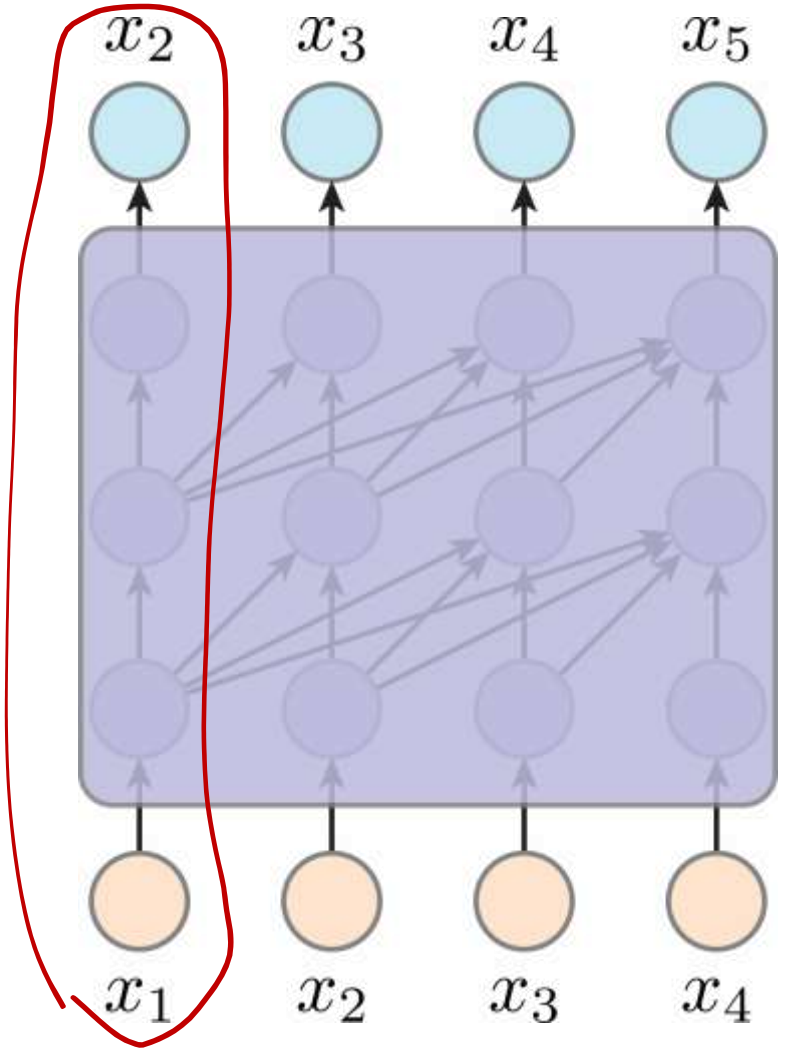


Brief: Attention (Transformer) for AR



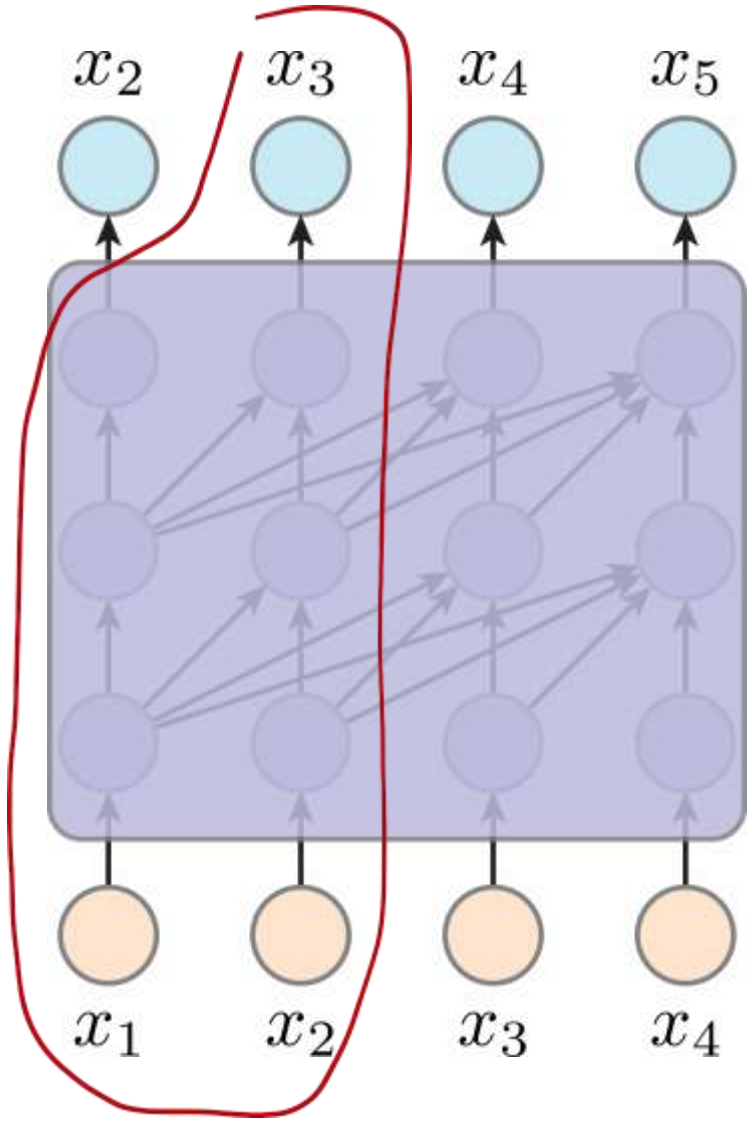
Brief: Attention (Transformer) for AR

$p(x_2 \mid x_1)$



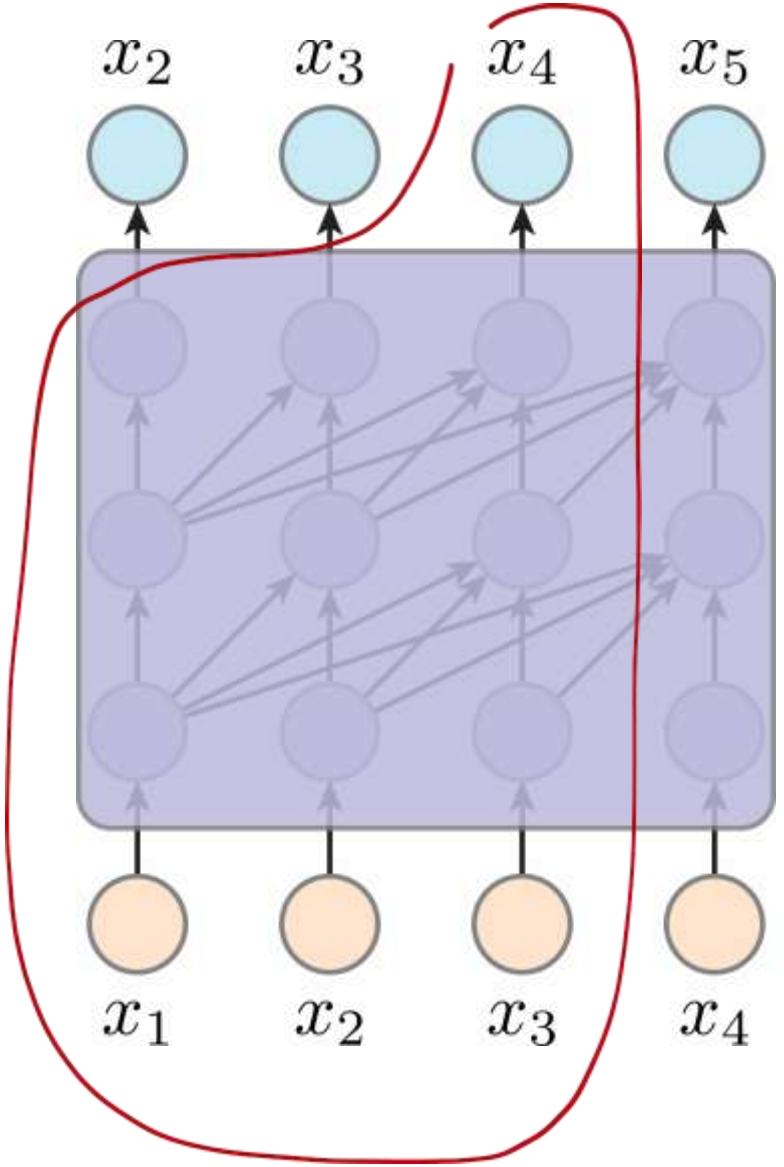
Brief: Attention (Transformer) for AR

$p(x_3 \mid x_{1,2})$



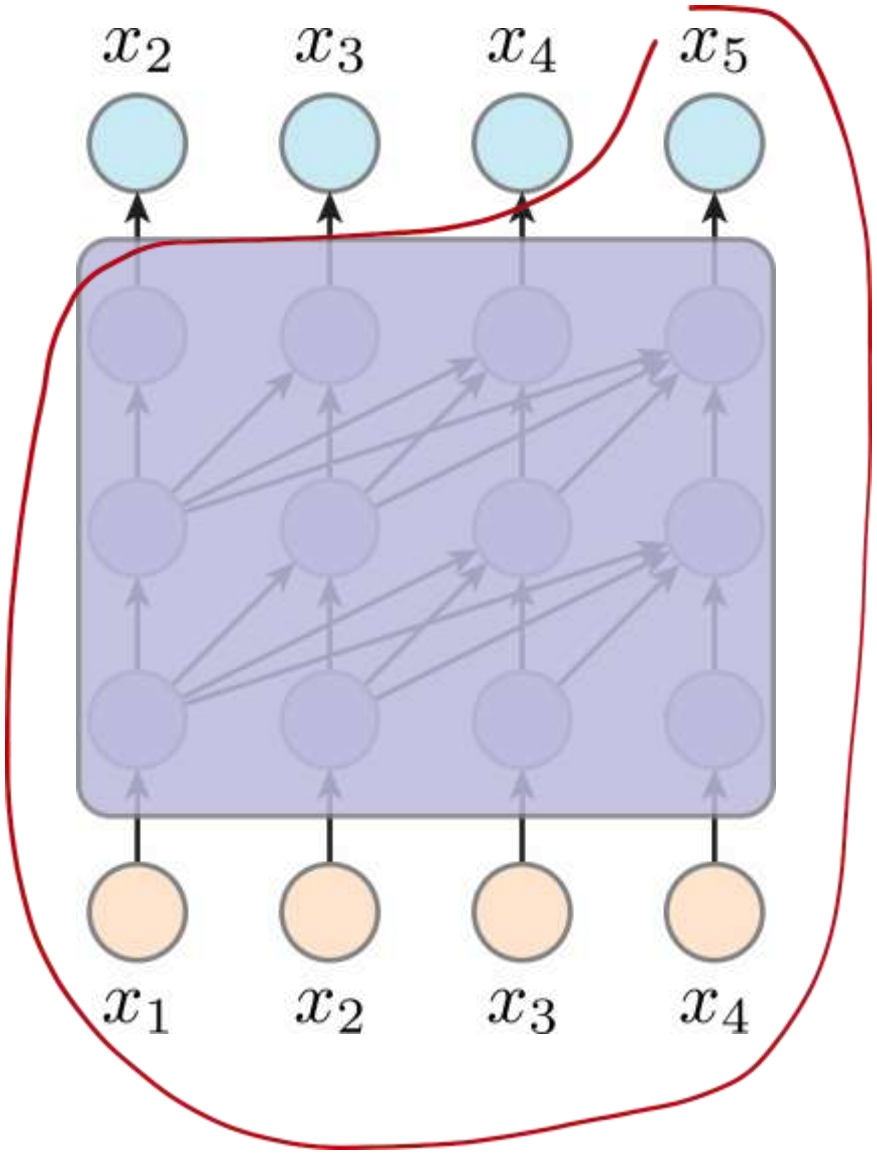
Brief: Attention (Transformer) for AR

$p(x_4 \mid x_{1,2,3})$

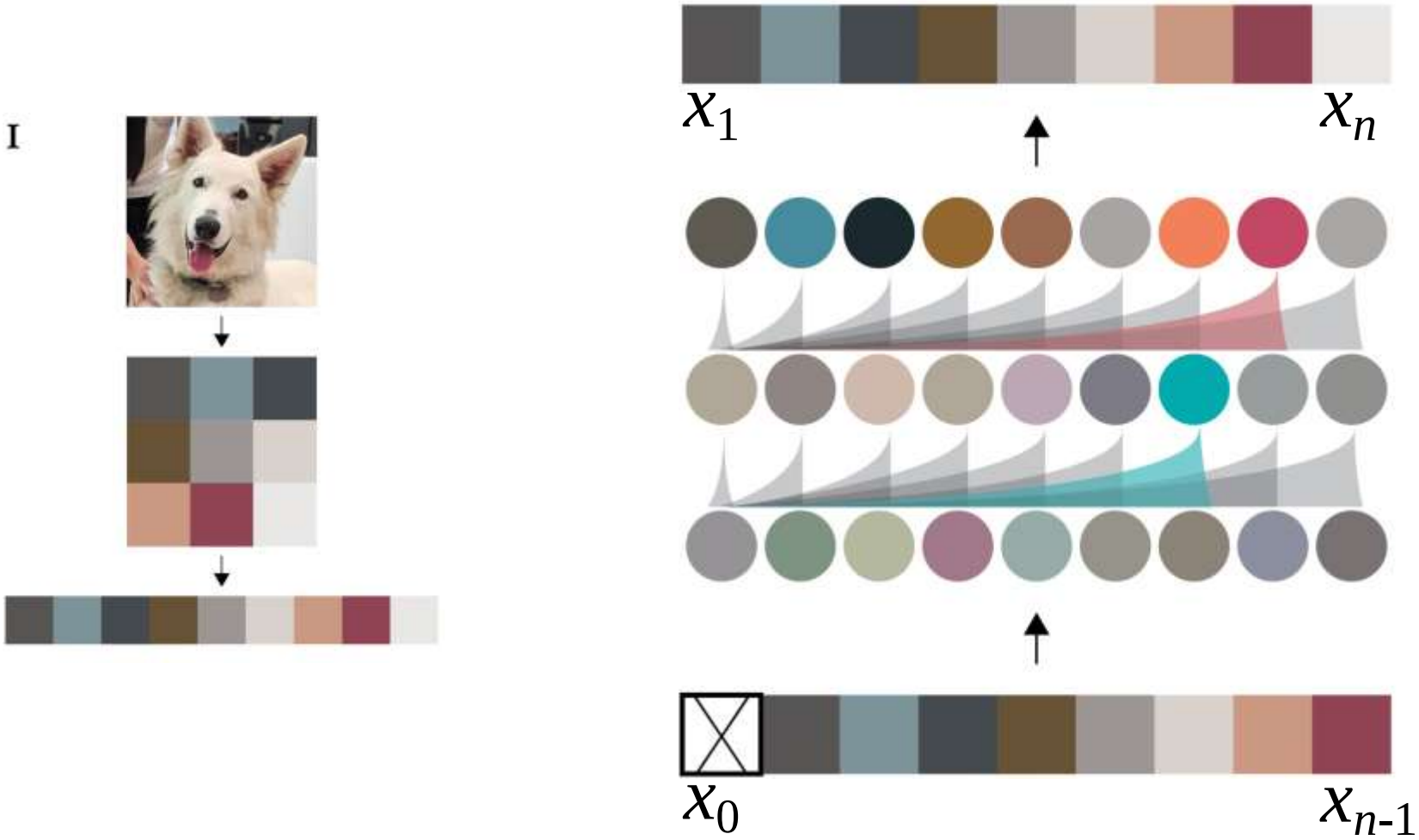


Brief: Attention (Transformer) for AR

$p(x_5 \mid x_{1,2,3,4})$



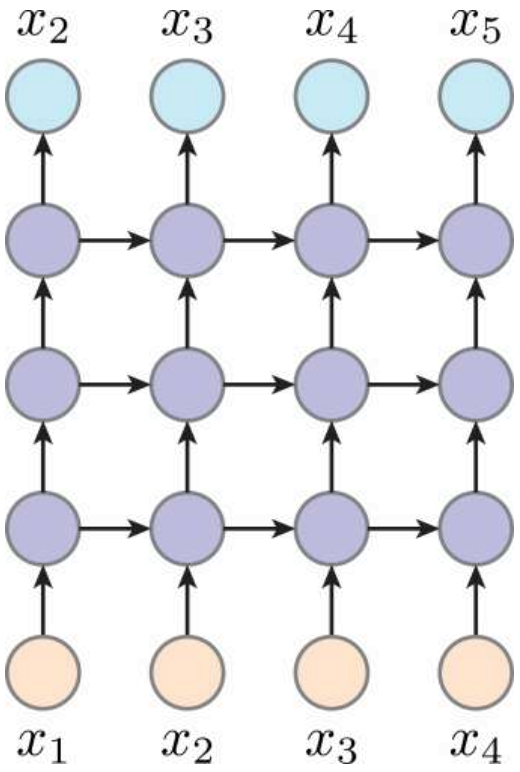
Example: image GPT (iGPT)



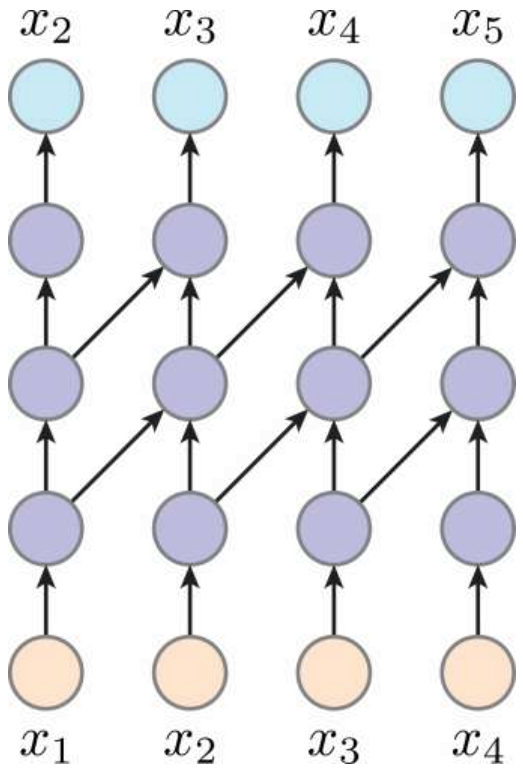
This figure is adapted from the original paper.

Chen, et al. “Generative Pretraining from Pixels”, ICML 2020

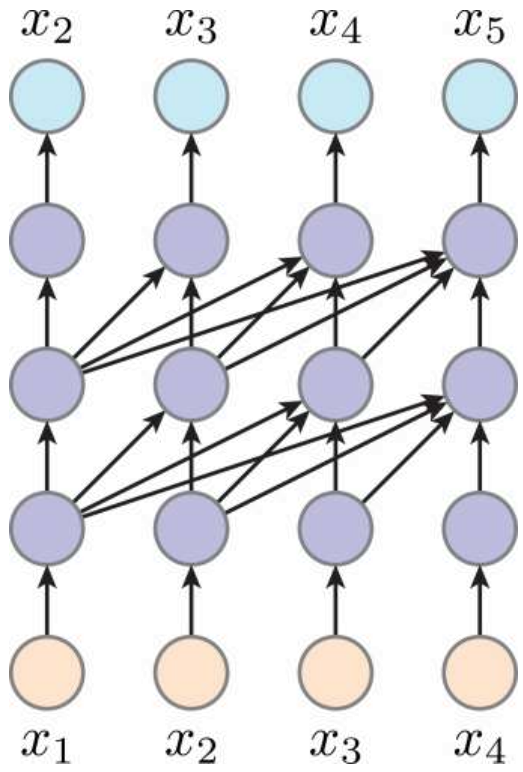
Summary: Network Architectures for AR



RNN



CNN



Attention

Summary: Autoregressive Models

Takeaways:

- Joint distribution ⇒ product of conditionals
- Inductive bias:
  - shared architecture, shared weight
  - induced order
- Inference: autoregressive
- Training: teacher-forcing

- Can be done by RNN, CNN, and Transformers

Main References

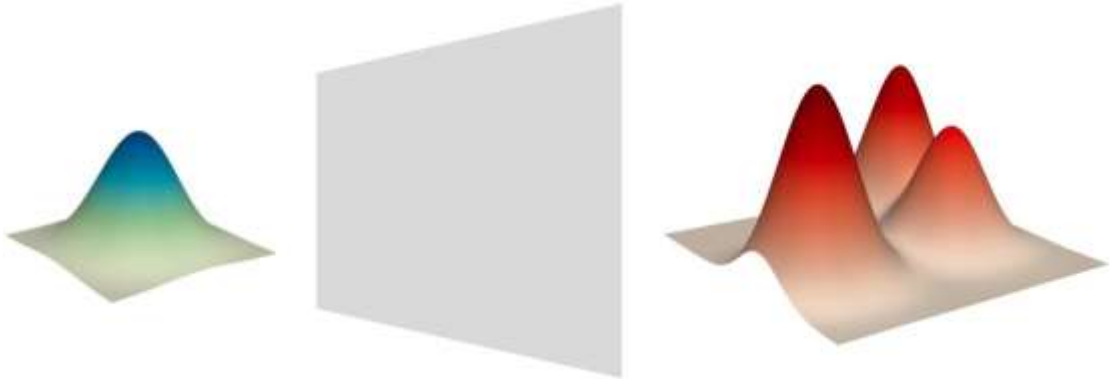
- Bengio and Bengio. “Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks”, NeurIPS 1999
- van den Oord, et al. “Pixel Recurrent Neural Networks”, ICML 2016
- Radford, et al. “Improving Language Understanding by Generative Pre-Training”, 2018

Recap.

How to Model Joint Distributions?

Solution 1: Modeling by **independent** latents (e.g., VAE)

- mapping: independent ⇒ dependent
- strict assumption for **high-dim** data (e.g., 32x32x3 pixels)
- often with **low-dim** latents
- a good building block, but often not sufficient



independent

dependent

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

in any <u>order</u>:

$$
\begin{aligned}
p(A, B, C) &= p(A)p(B \mid A)p(C \mid A, B) \\
&= p(A)p(C \mid A)p(B \mid A, C) \\
&= p(B)p(A \mid B)p(C \mid A, B) \\
&= p(B)p(C \mid B)p(A \mid B, C) \\
&= p(C)p(A \mid C)p(B \mid A, C) \\
&= p(C)p(B \mid C)p(A \mid B, C)
\end{aligned}
$$

* This is a foundation of Masked Autoregressive (MAR) models:
Li, et al. Autoregressive Image Generation without Vector Quantization, 2024

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

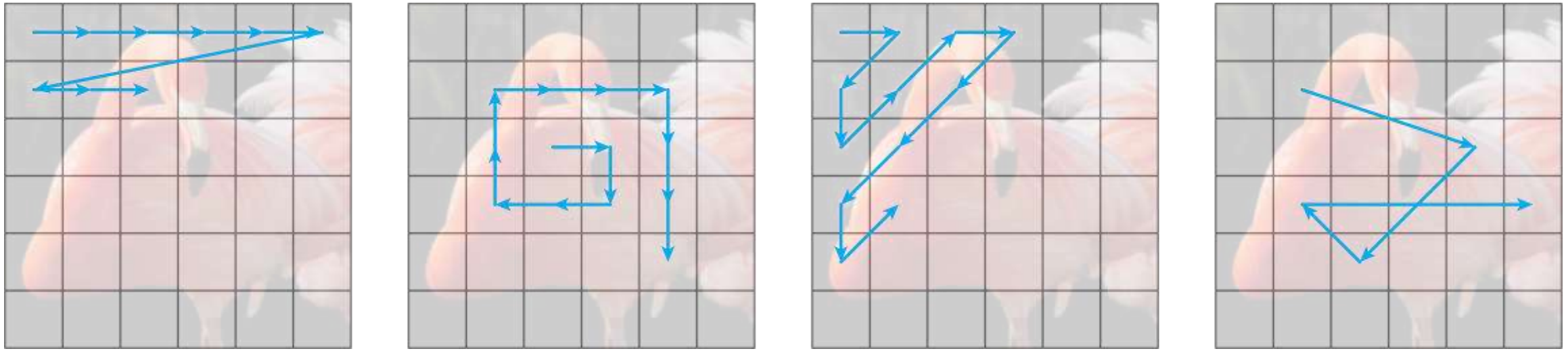
in any <u>partition</u>:

$$p(A, B, C, D) = p(A, B)p(C, D \mid A, B)$$
$$= p(C, D)p(A, B \mid C, D)$$
$$= p(A, B, C)p(D \mid A, B, C)$$
$$= \ldots$$

* This is a foundation of Masked Autoregressive (MAR) models:
Li, et al. Autoregressive Image Generation without Vector Quantization, 2024

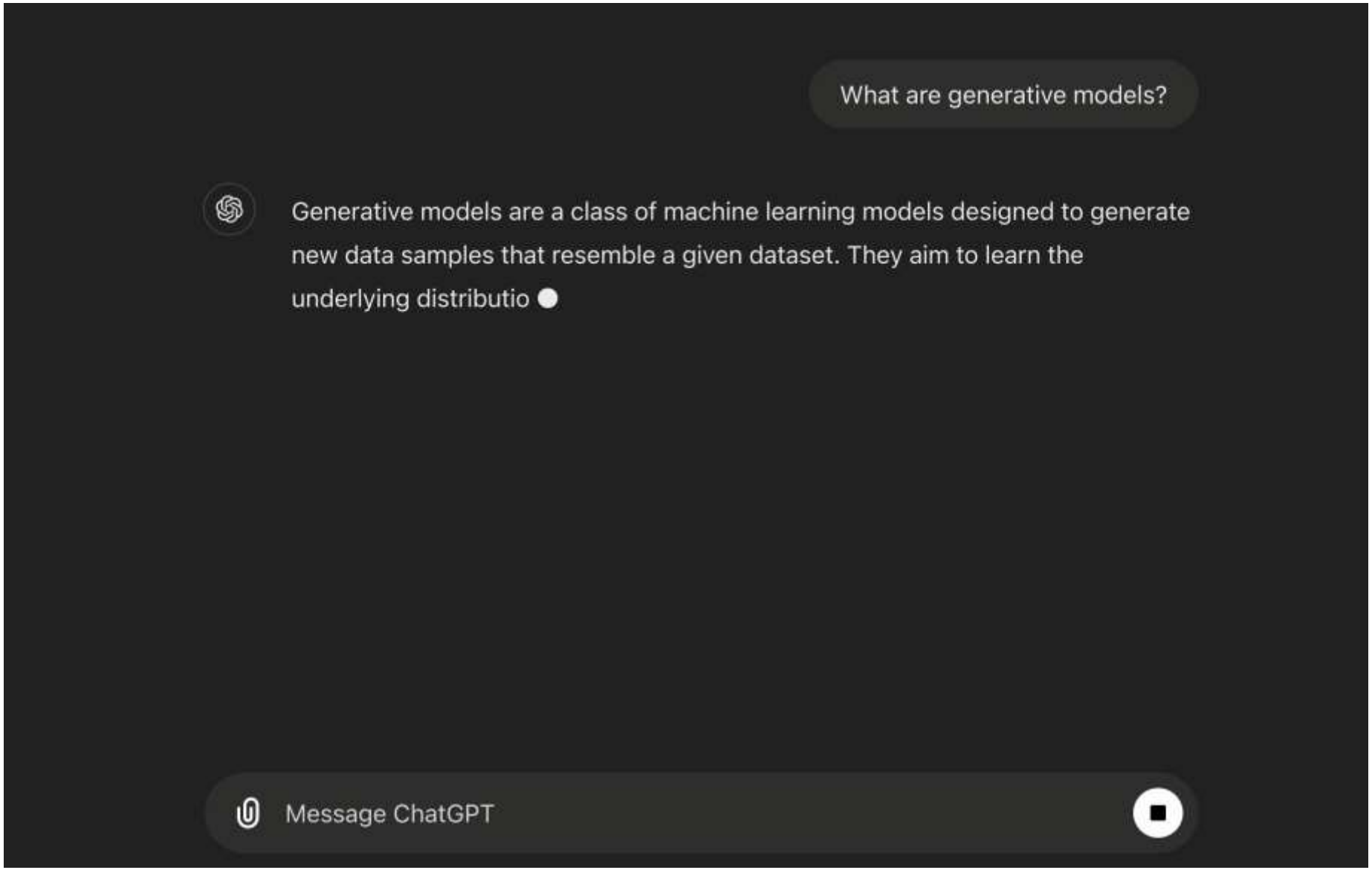
Dependency Graphs

- Some dependency graphs may induce **simpler** distributions ...



E.g., see: Hua, et al. “Self-supervision through Random Segments with Autoregressive Coding (RandSAC)”, ICLR 2022

ChatGPT: Next Token Prediction

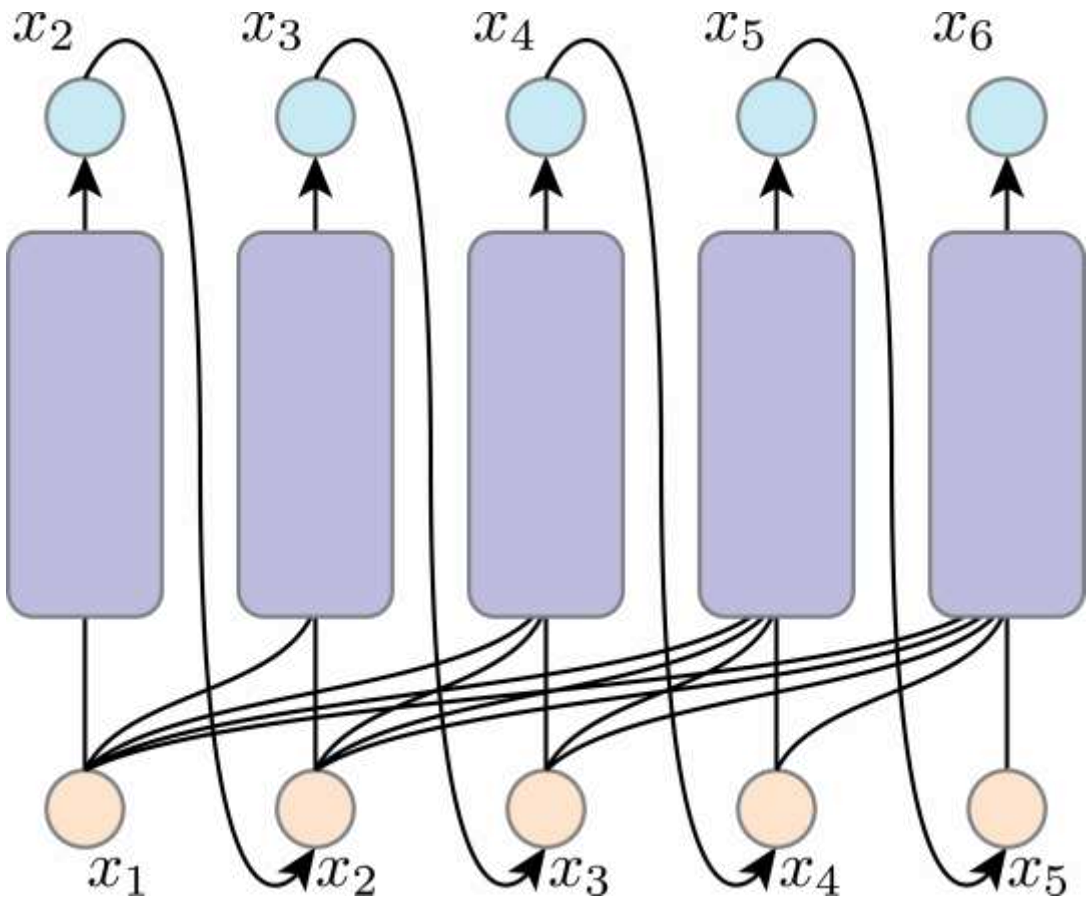


Inference: Autoregressive

Note:

- This is a **recursive** process

- but **not** necessarily done by RNN
- can be done by **any** architecture (e.g., CNN or Transformers)



Training: Teacher-Forcing

Teacher-forcing

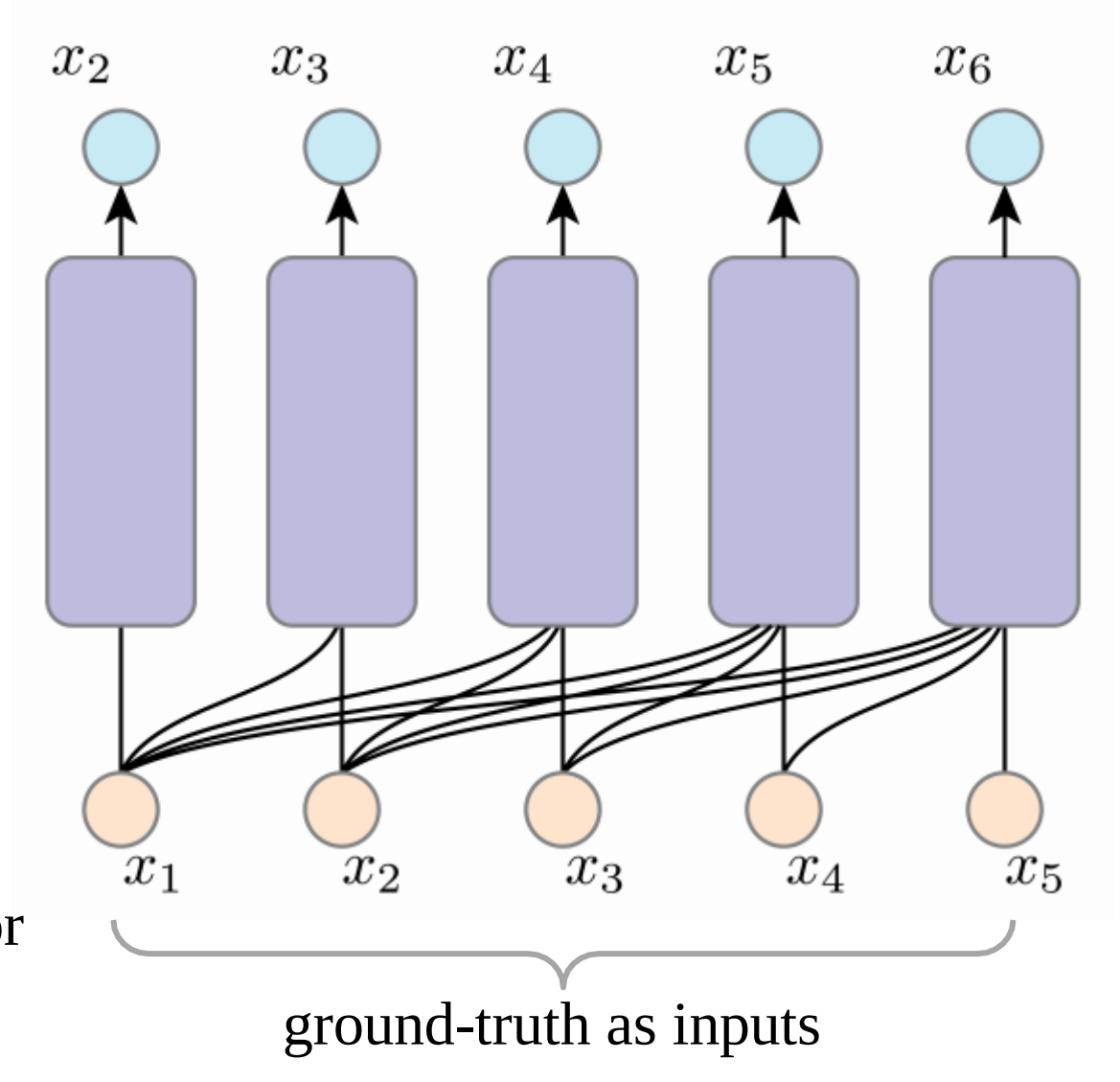
- Inputs are not from previous outputs
- Inputs are from ground-truth data

Pros:

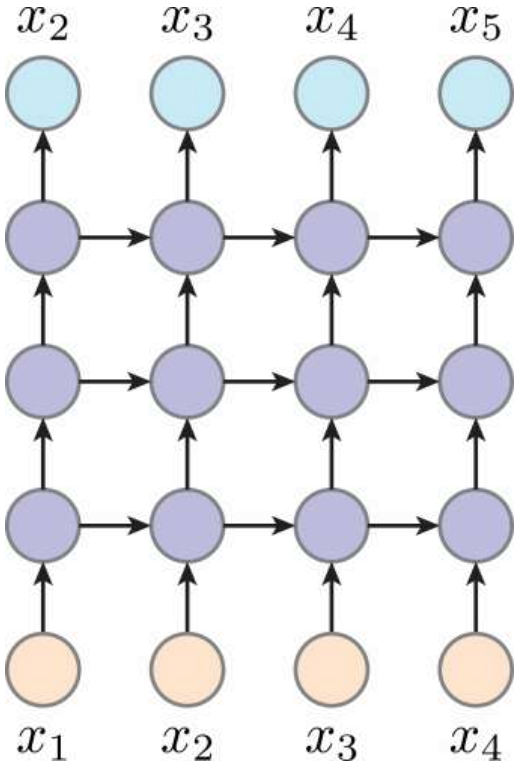
- backprop path is much shorter
- ground-truth inputs can ease training

Cons:

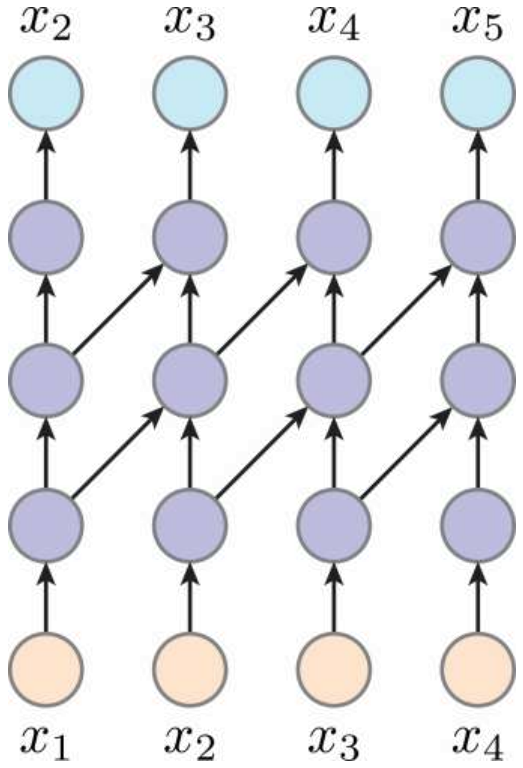
- inconsistent training/inference
- distribution shift: can't see its own error



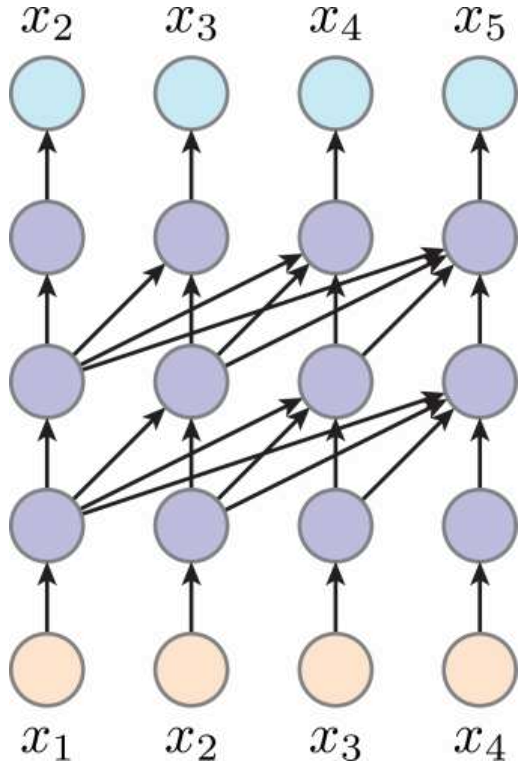
Common Architectures for Autoregression



RNN



CNN



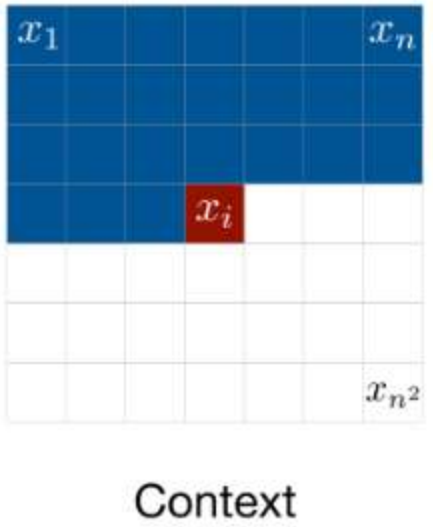
Attention

See also my lecture at 6.8300: “11: Sequence Modeling”: https://drive.google.com/file/d/1lOYsyImXl3caWlsplfThyuvgz-

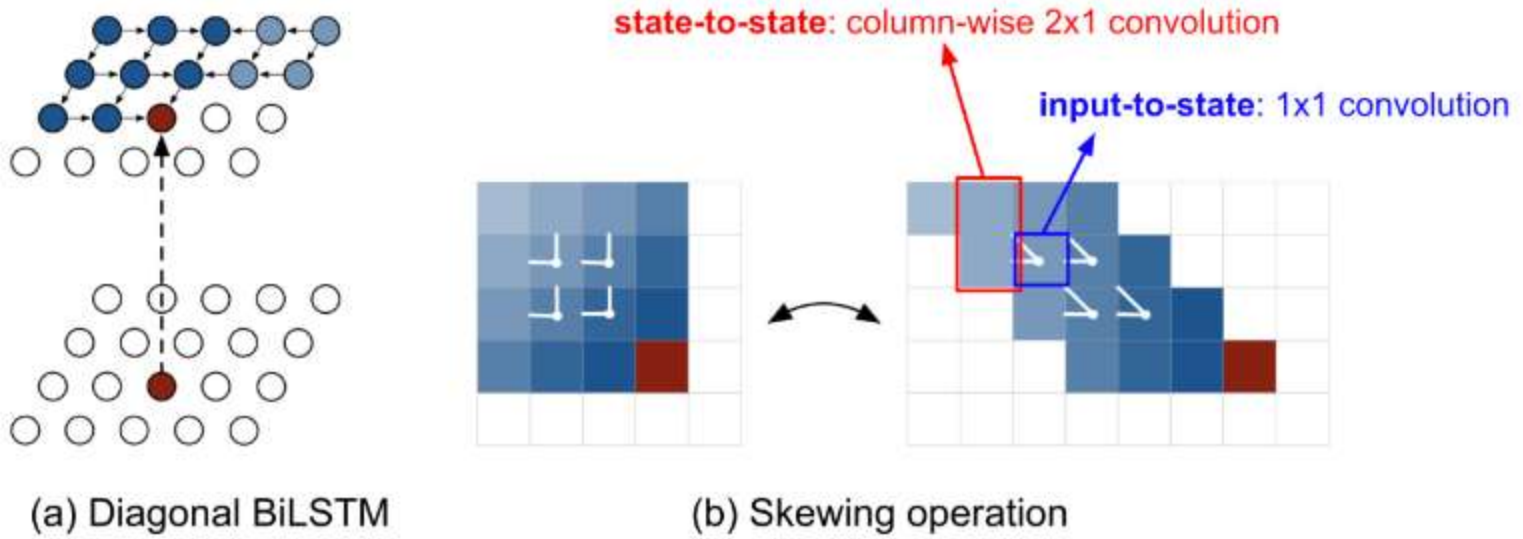
AR Models i.

PixelRNN *[van der Oord et al. 2016]*

The image is generated one pixel at a time and each new pixel is sampled conditional on the pixels that have been seen before.



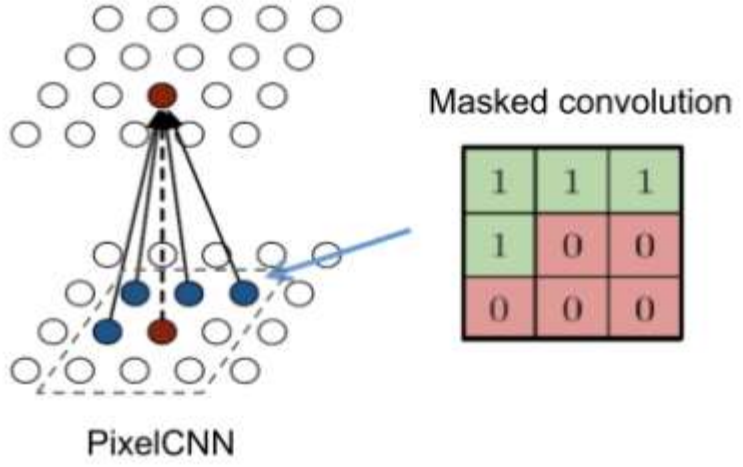
PixelRNN *[van der Oord et al. 2016]*



PixelCNN *[van der Oord et al. 2016]*

- A faster implementation uses multiple convolutional layers without pooling to define a bounded context box.

- The convolution kernel is masked so that the future context is not seen.



Results

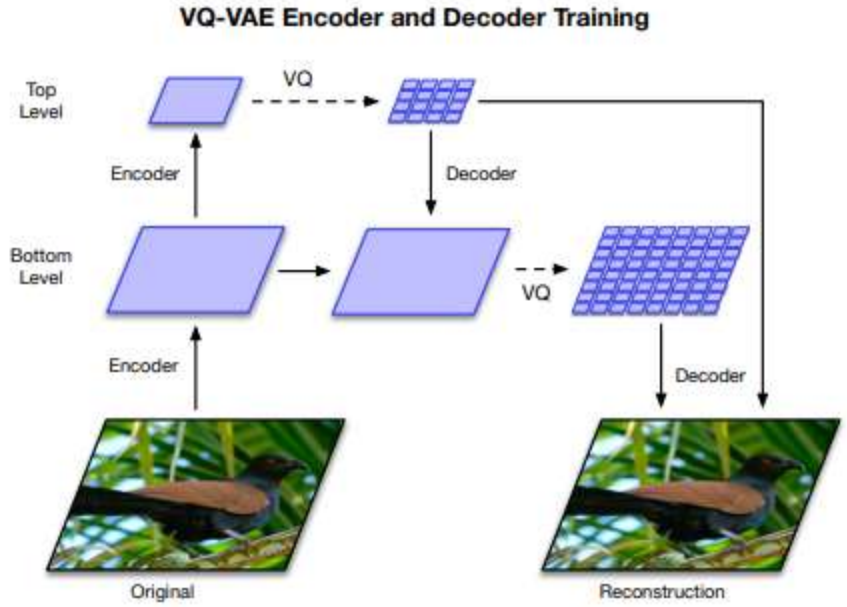
32x32 CIFAR-10

32x32 ImageNet

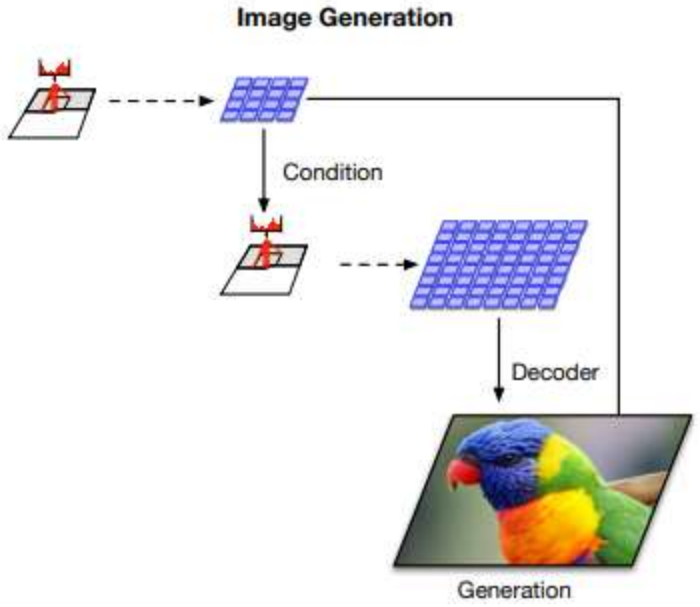
Results

Figure 1. Image completions sampled from a PixelRNN.

VQ-VAE and VQ-VAE2



(a) Overview of the architecture of our hierarchical VQ-VAE. The encoders and decoders consist of deep neural networks. The input to the model is a 256×256 image that is compressed to quantized latent maps of size 64×64 and 32×32 for the *bottom* and *top* levels, respectively. The decoder reconstructs the image from the two latent maps.



(b) Multi-stage image generation. The top-level PixelCNN prior is conditioned on the class label, the bottom level PixelCNN is conditioned on the class label as well as the first level code. Thanks to the feed-forward decoder, the mapping between latents to pixels is fast. (The example image with a parrot is generated with this model).

DALL-E：Zero-Shot Text-to-Image Generation

- **Stage 1.** We train a discrete variational autoencoder (dVAE)[1] to compress each 256×256 RGB image into a 32×32 grid of image tokens, each element of which can assume 8192 possible values. This reduces the context size of the transformer by a factor of 192 without a large degradation in visual quality (see Fig-

- **Stage 2.** We concatenate up to 256 BPE-encoded text tokens with the $32 \times 32 = 1024$ image tokens, and train an autoregressive transformer to model the joint distribution over the text and image tokens.

$$\ln p_{\theta,\psi}(x, y) \geqslant \mathop{\mathbb{E}}_{z \sim q_\phi(z \mid x)} \Big(\ln p_\theta(x \mid y, z) - \beta \, D_{\mathrm{KL}}(q_\phi(y, z \mid x), p_\psi(y, z)) \Big), \quad (1)$$

where: images x, captions y, and the tokens z

- q_ϕ denotes the distribution over the 32×32 image tokens generated by the dVAE encoder given the RGB image x^2;
- p_θ denotes the distribution over the RGB images generated by the dVAE decoder given the image tokens; and
- p_ψ denotes the joint distribution over the text and image tokens modeled by the transformer.

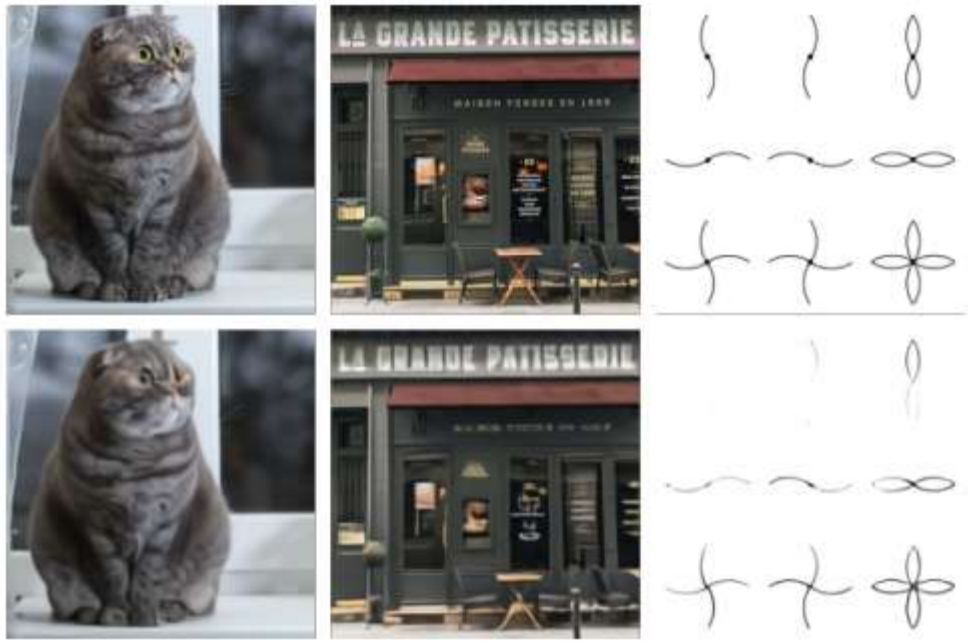
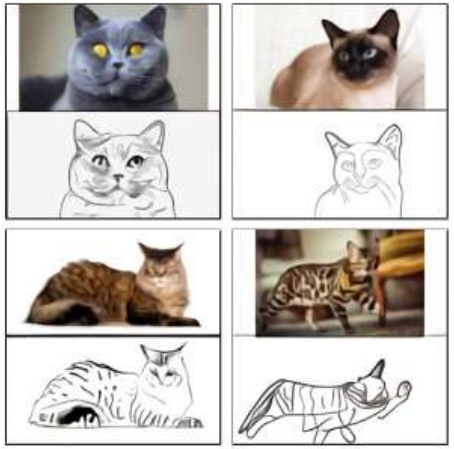
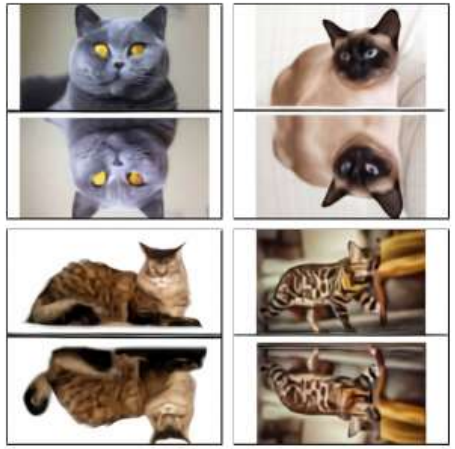


Figure 1. Comparison of original images (top) and reconstructions from the discrete VAE (bottom). The encoder downsamples the spatial resolution by a factor of 8. While details (e.g., the texture of the cat's fur, the writing on the storefront, and the thin lines in the illustration) are sometimes lost or distorted, the main features of the image are still typically recognizable. We use a large vocabulary size of 8192 to mitigate the loss of information.

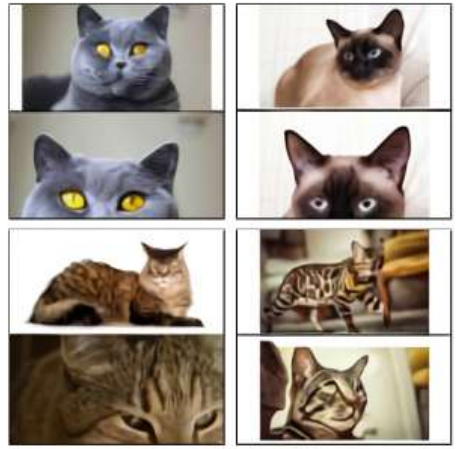
DALL-E：Zero-Shot Text-to-Image Generation



(a) “the exact same cat on the top as a sketch on the bottom”



(b) “the exact same photo on the top reflected upside-down on the bottom”



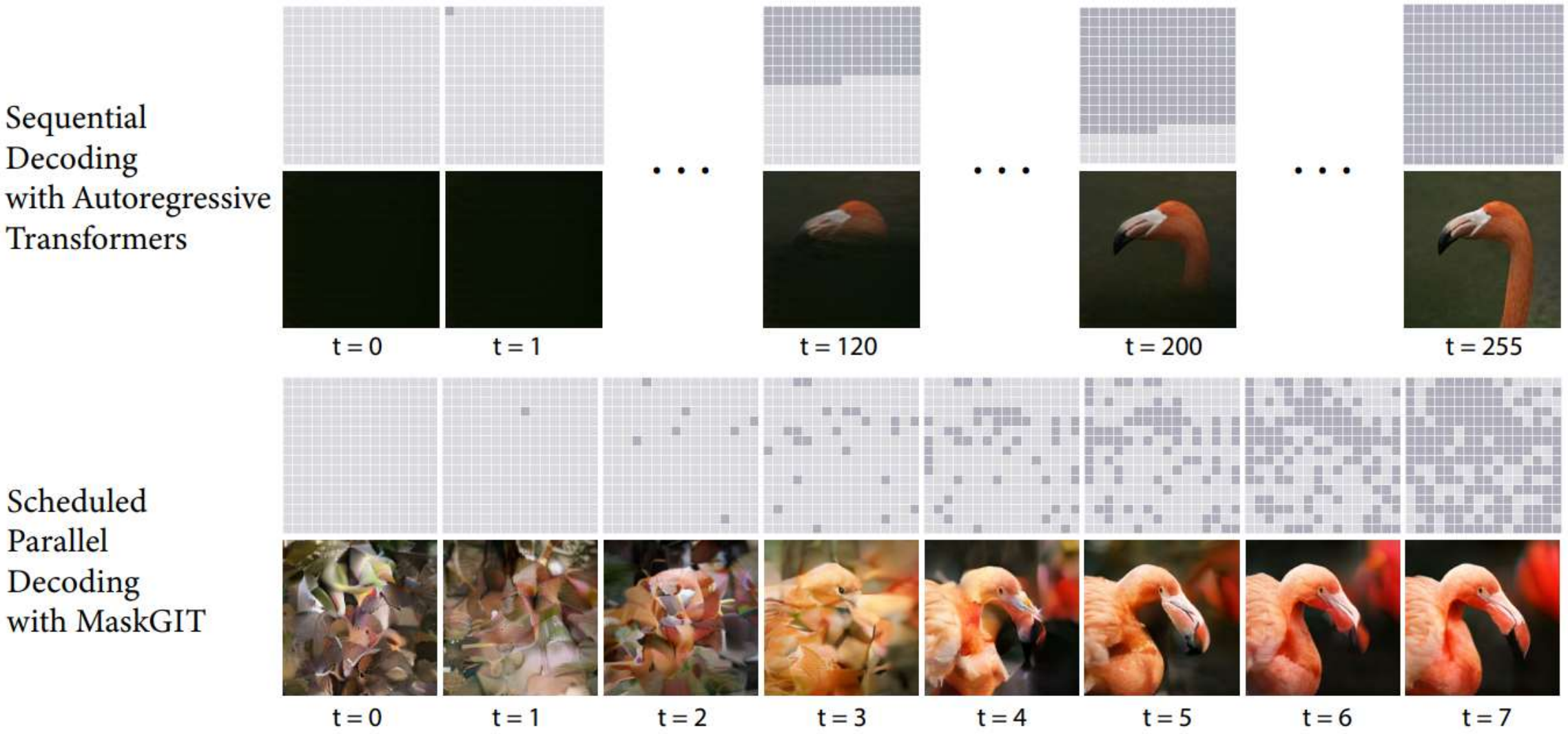
(c) “2 panel image of the exact same cat. on the top, a photo of the cat. on the bottom, an extreme close-up view of the cat in the photo.”

(d) “the exact same cat on the top colored red on the bottom”

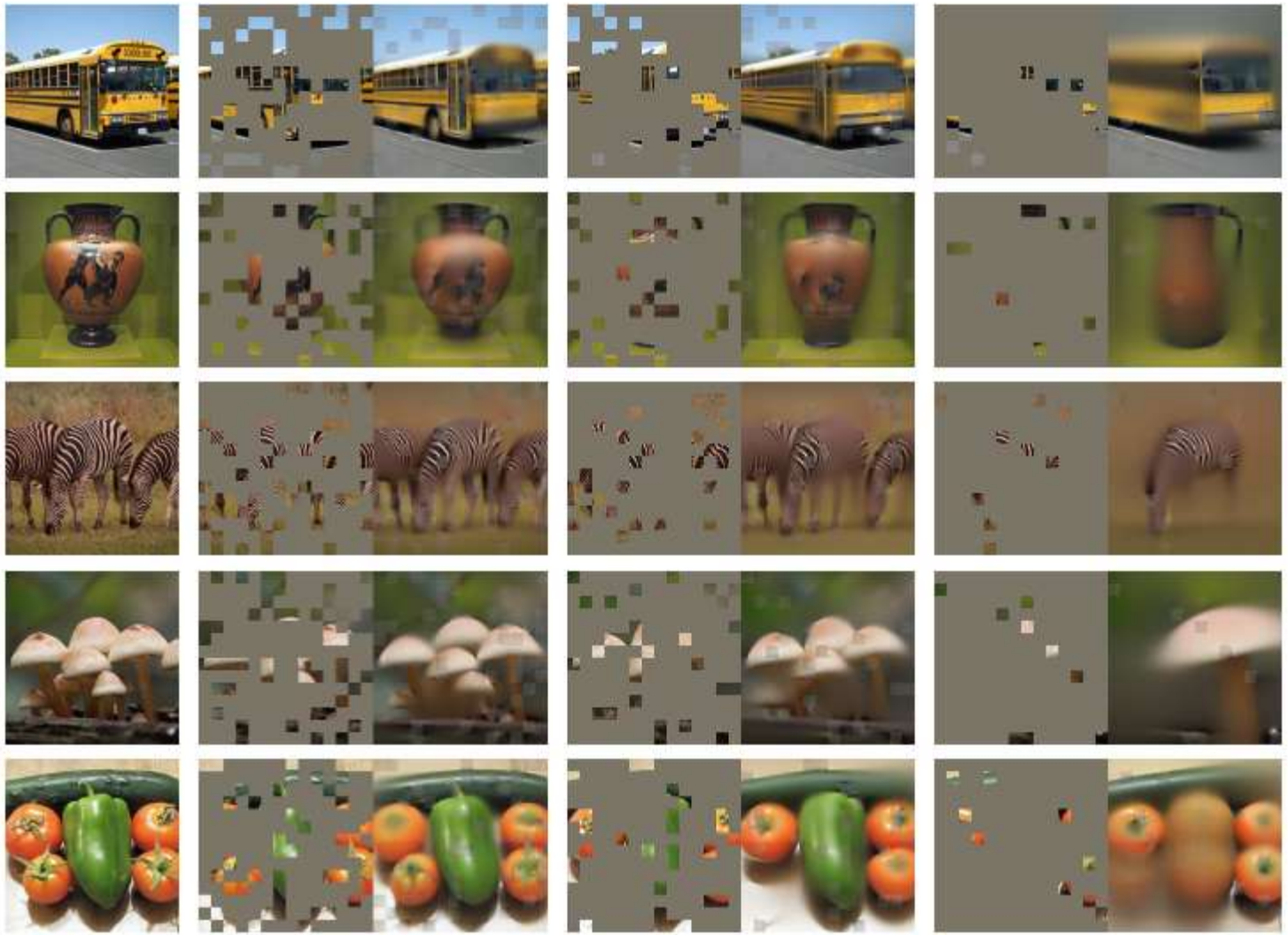
(e) “2 panel image of the exact same cat. on the top, a photo of the cat. on the bottom, the cat with sunglasses.”

(f) “the exact same cat on the top as a postage stamp on the bottom”

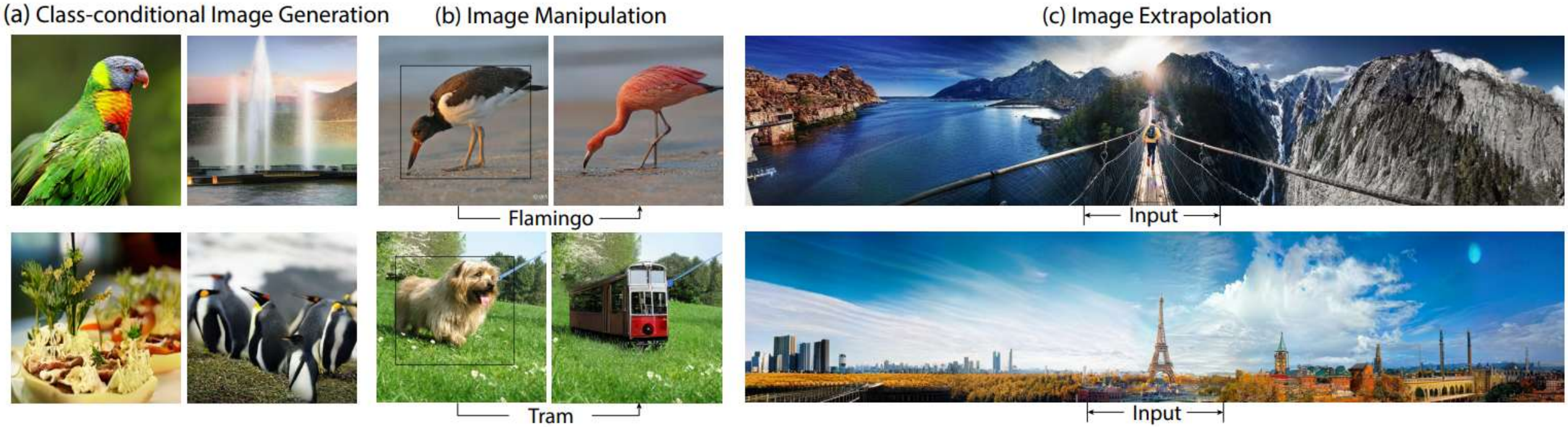
MaskGIT: Masked Generative Image Transformer



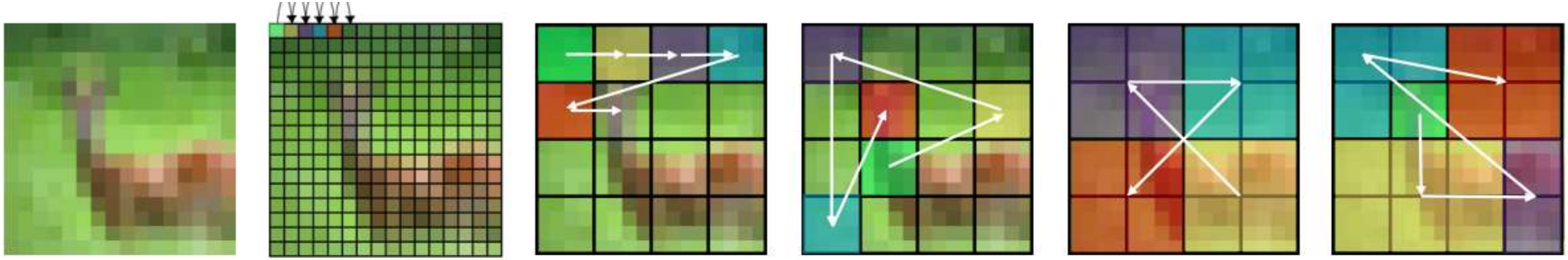
Masked Auto-Encoder / BERT-Style



MaskGIT: Masked Generative Image Transformer



RANDSAC：RANDOM SEGMENTS WITH AUTOREGRESSIVE CODING



(a) (b) (c) (d) (e) (f)

Figure 2: **Autoregressive Prediction Schemes.** Left-to-right: (a) original image from CIFAR 10; (b) raster-order pixel prediction; (c) raster-order patch prediction; (d) stochastic patch prediction; (e) stochastic square segment prediction ($M = 2$); (f) stochastic blob segment prediction ($K = 5$).

RANDSAC：RANDOM SEGMENTS WITH AUTOREGRESSIVE CODING

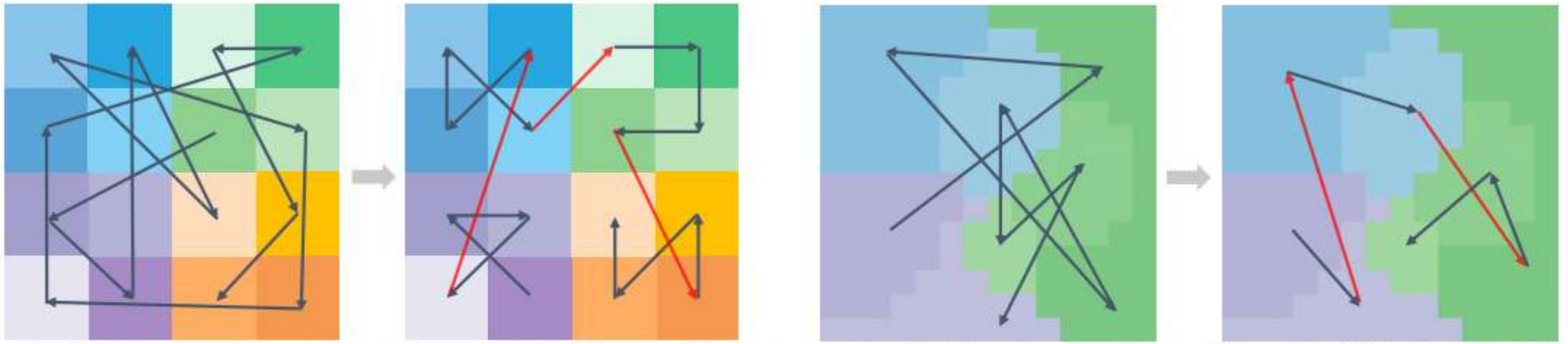
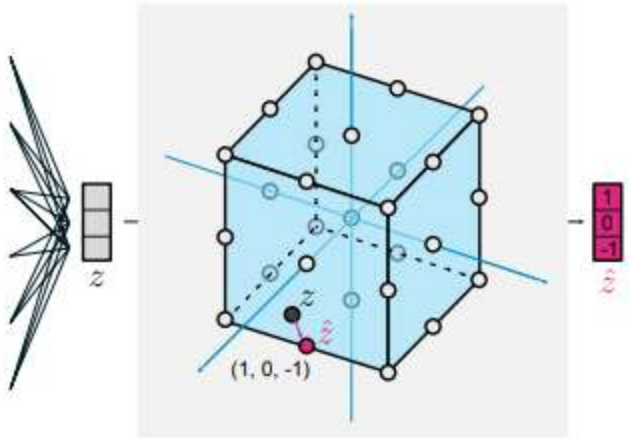


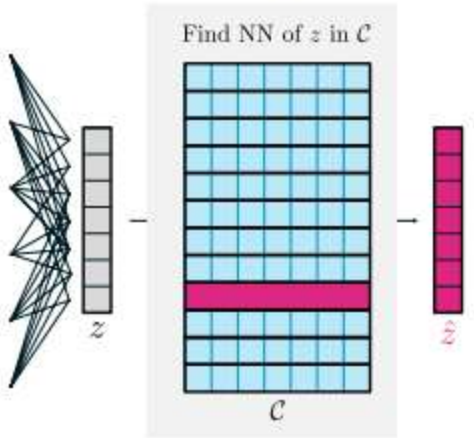
Figure 3: **Hierarchical Segment Serialization.** We partition an image into a hierarchy of segments (segments are illustrated by color and tokens within segment by shade). Autoregressive prediction is done by following a traversal of randomly generated hierarchical partitions.

FSQ: FINITE SCALAR QUANTIZATION

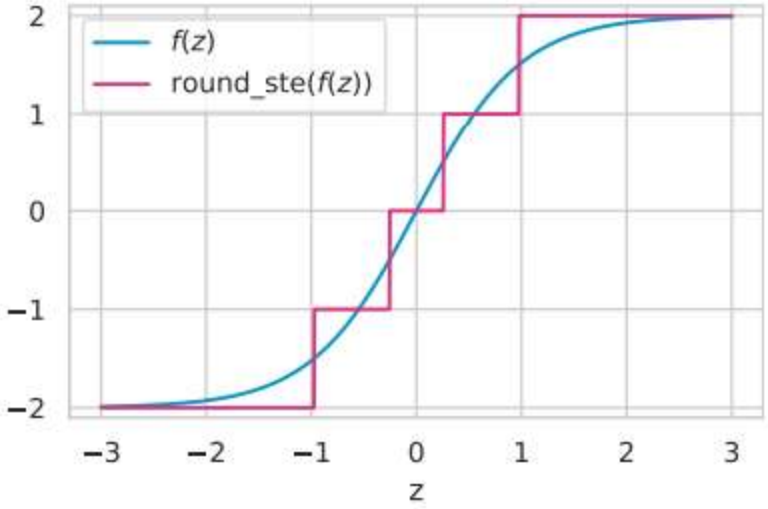
FSQ



VQ



	VQ	FSQ
Quantization	$\arg\min_{c \in \mathcal{C}} \|\|z - c\|\|$	$\text{round}(f(z))$
Gradients	STE	STE
Aux. Losses	Commitment, codebook, entropy loss	-
Tricks	EMA on codebook, codebook splitting projections, ...	-
Parameters	Codebook	-

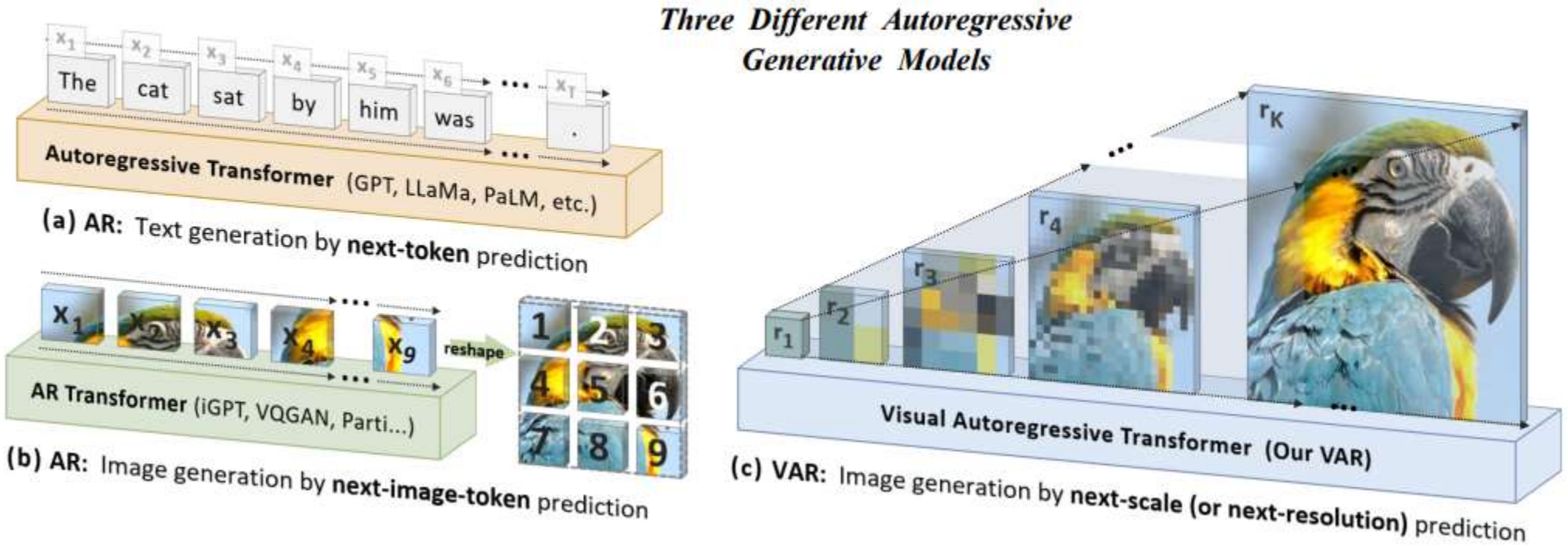


Target Size $\|\mathcal{C}\|$	2^8	2^{10}	2^{12}	2^{14}	2^{16}
Proposed $\mathcal{L}$	$[8, 6, 5]$	$[8, 5, 5, 5]$	$[7, 5, 5, 5, 5]$	$[8, 8, 8, 6, 5]$	$[8, 8, 8, 5, 5, 5]$

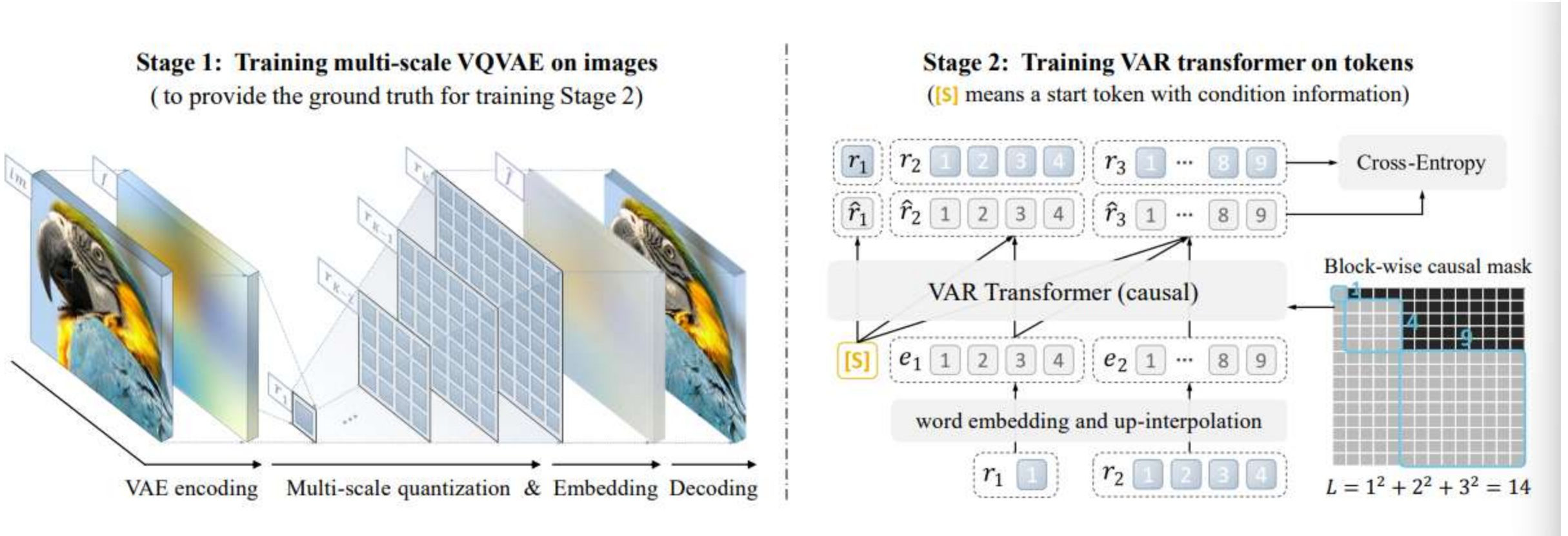
FSQ: FINITE SCALAR QUANTIZATION

Figure 5: Non-cherry-picked samples from our FSQ (top) and VQ (bottom) MaskGIT models for 4 imagenet classes (330, 320, 510, 454). We show two samples per model per category. Both models get very comparable sample quality, as reflected by the metrics in Fig. 4.

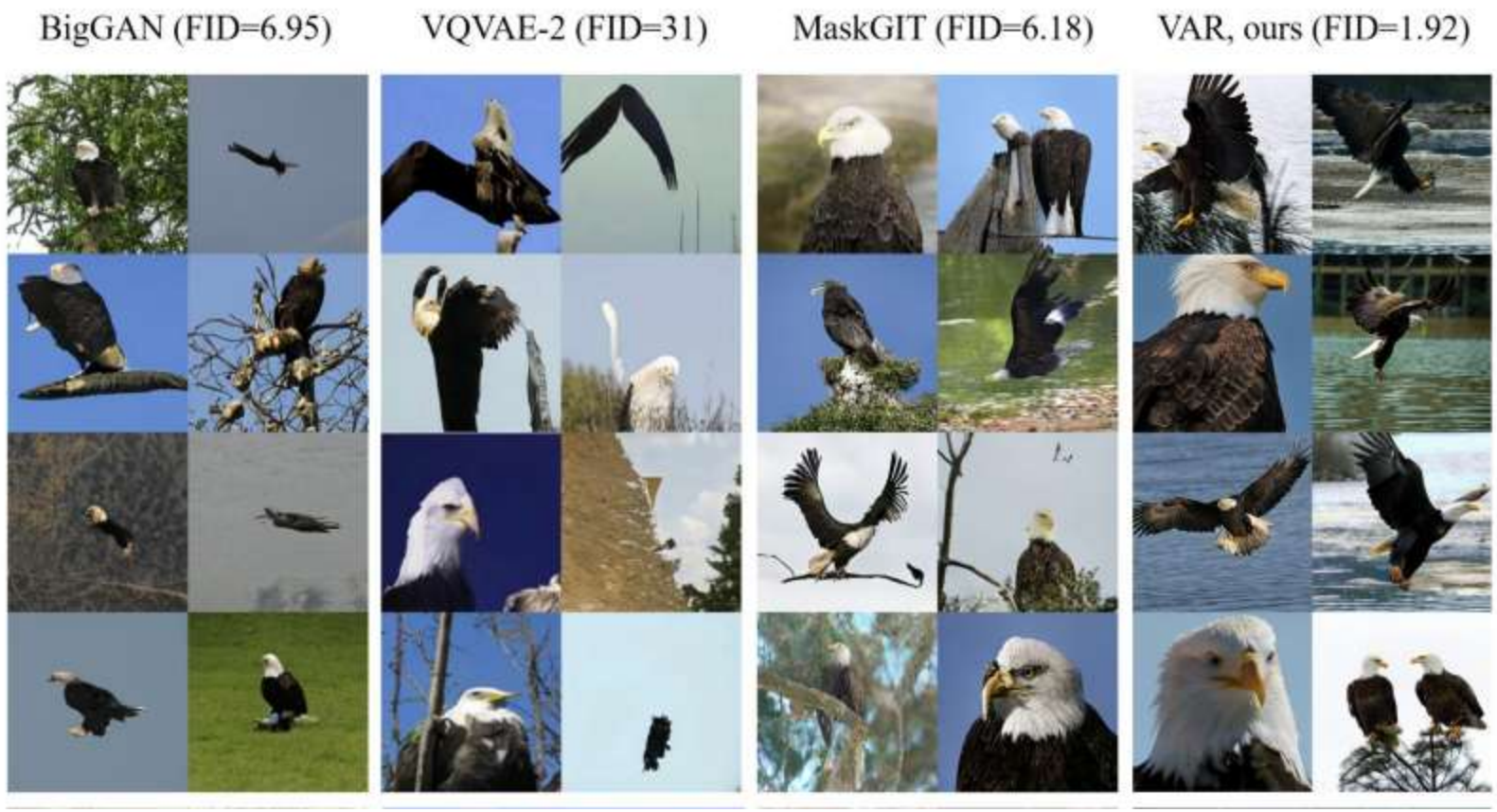
VAR: Visual Autoregressive Modeling



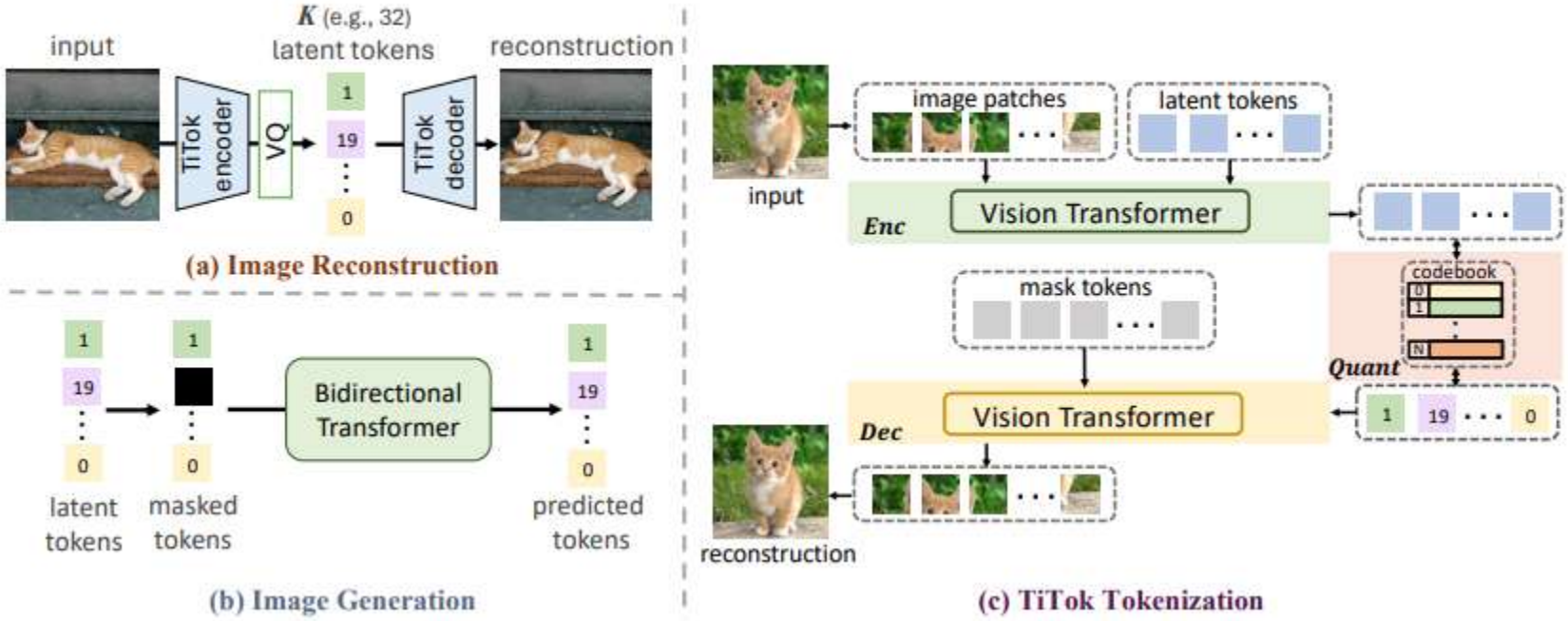
VAR: Visual Autoregressive Modeling



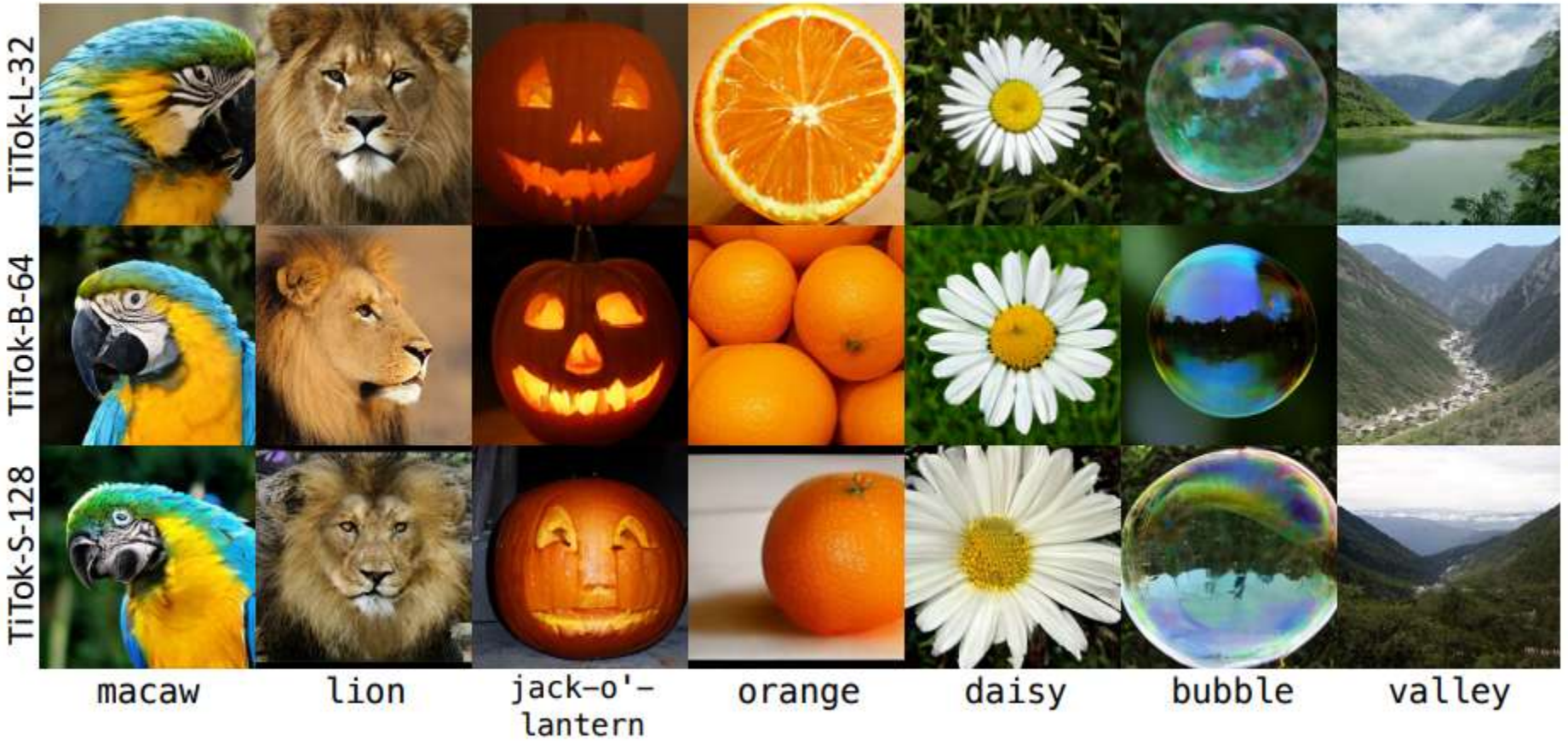
VAR: Visual Autoregressive Modeling



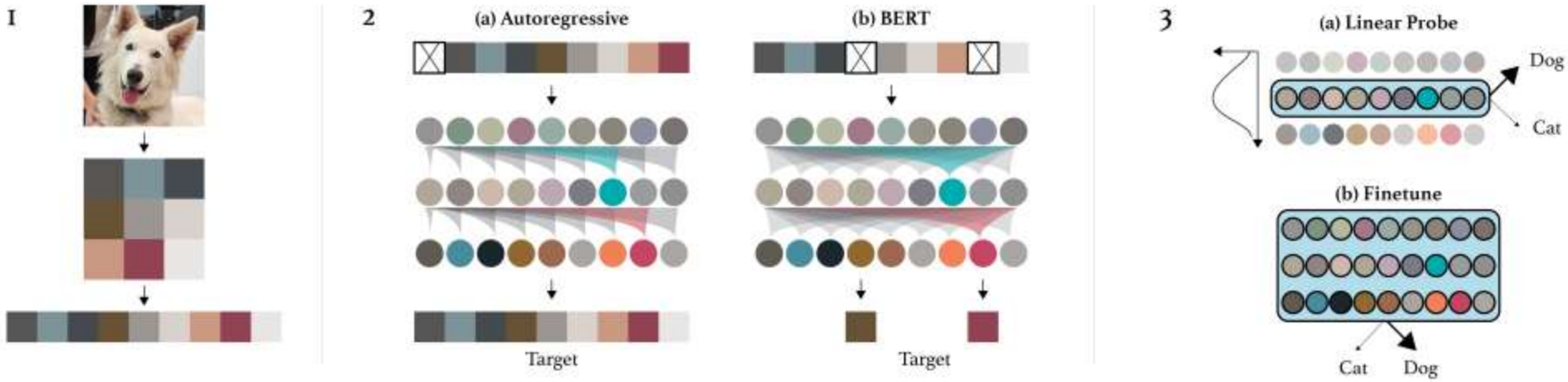
TiTok: An Image is Worth 32 Tokens for Reconstruction and Generation



TiTok: An Image is Worth 32 Tokens for Reconstruction and Generation



iGPT：Generative Pretraining from Pixels



A GPT-2 scale model learns strong image representations as measured by linear probing, fine-tuning, and low-data classification