Lecture 3

Autoregressive Models

6.S978 Deep Generative Models

Kaiming He
Fall 2024, EECS, MIT

Overview

- Conditional Distribution Modeling

- Autoregressive Models

- Network Architectures for Autoregressive Modeling

Conditional Distribution Modeling

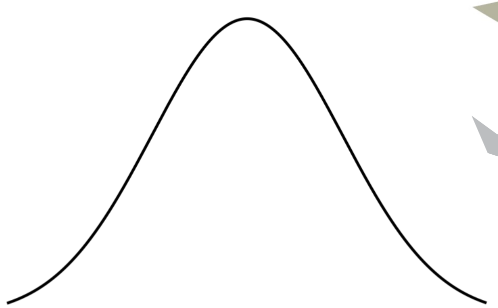
Joint Distribution

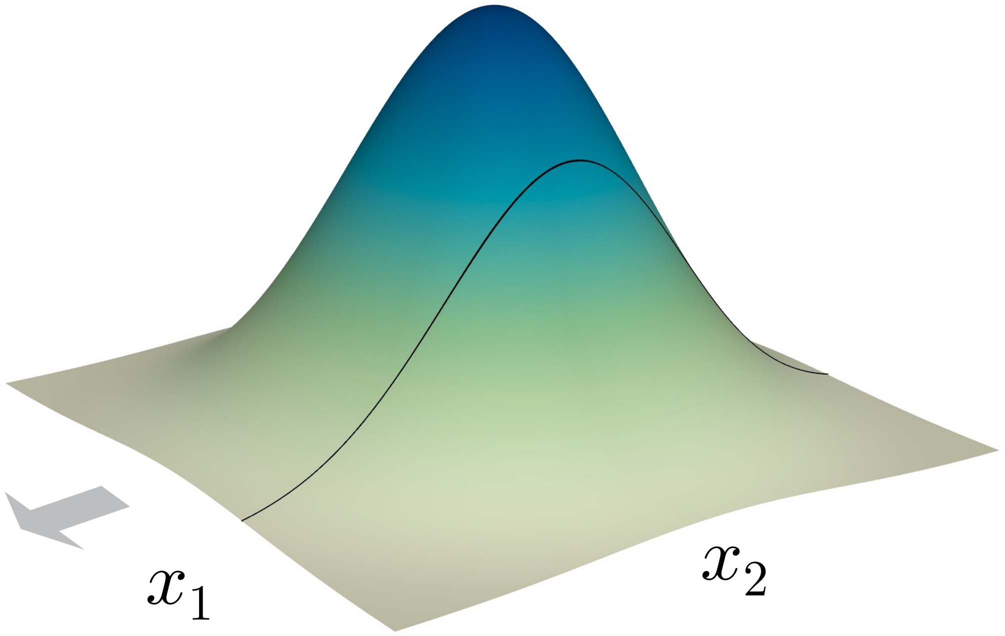
It's convenient to model joint distributions by **independent** distributions

$$p(x_{1,}\, x_2) = p(x_1)p(x_2)$$

conditional $=$ marginal

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



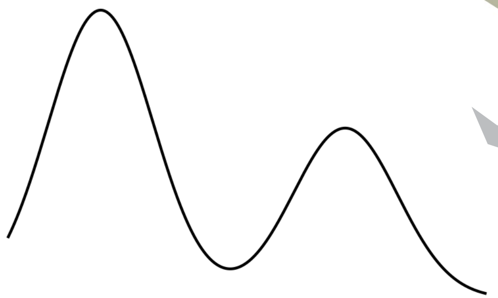


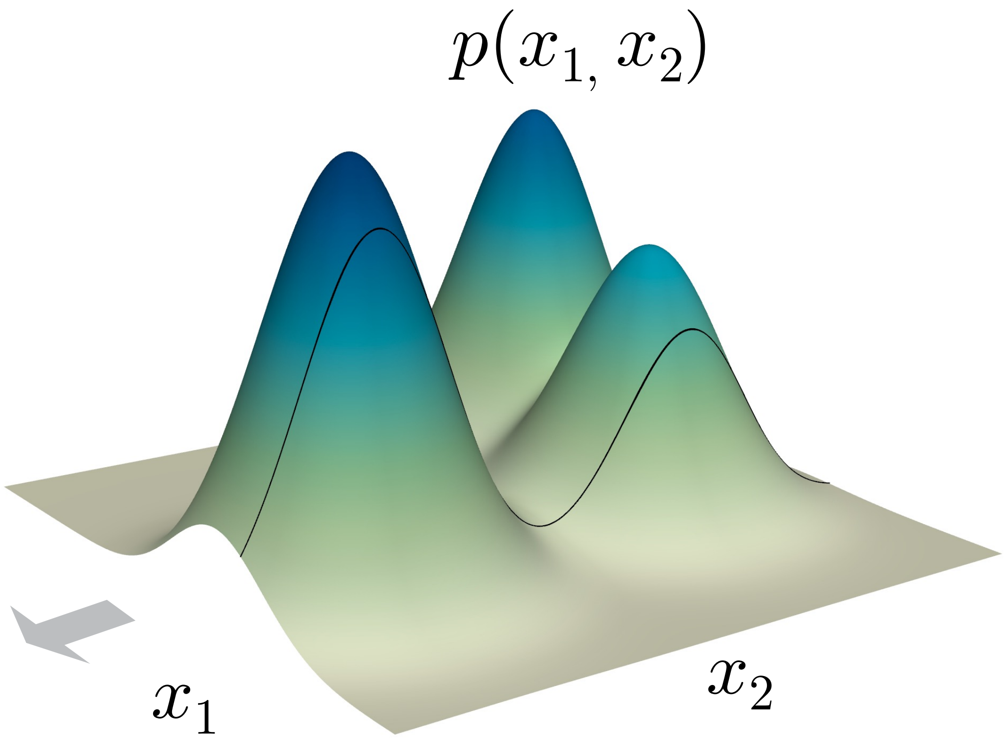
Joint Distribution

Real-word problems always involve dependent variables

conditional ≠ marginal

$p(x_2|x_1)$



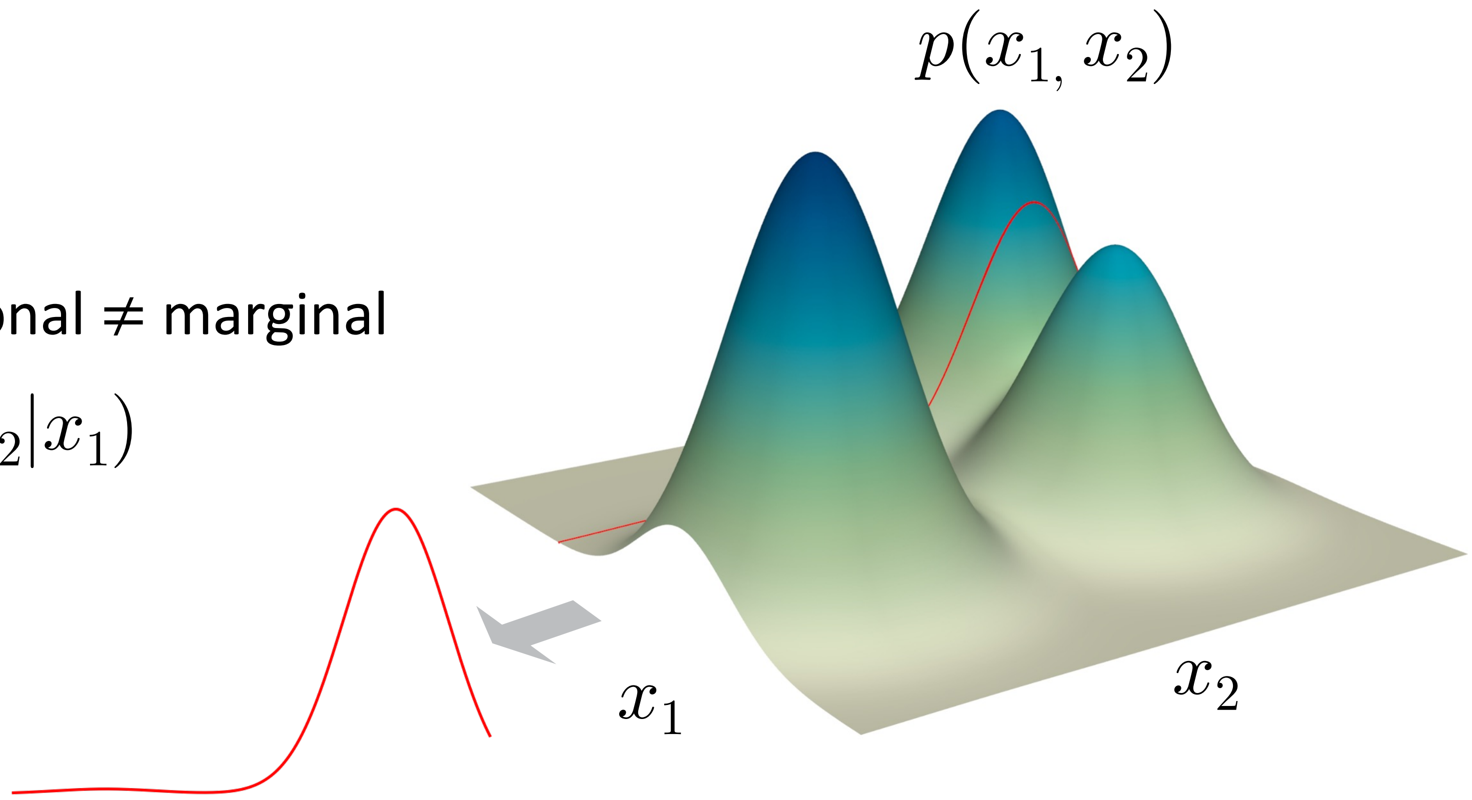


Joint Distribution

Real-word problems always involve dependent variables

conditional $\neq$ marginal

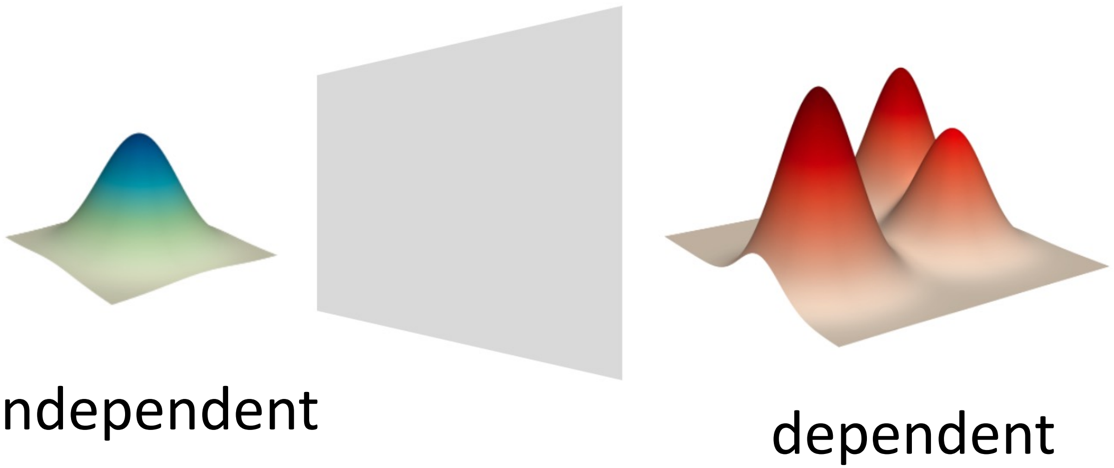
$p(x_2|x_1)$



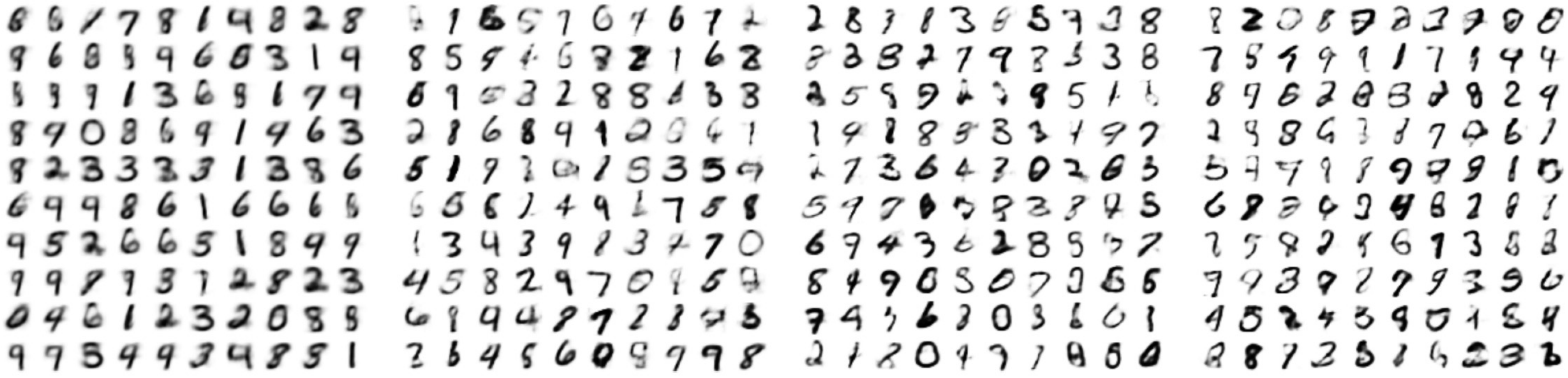
How to Model Joint Distributions?

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



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

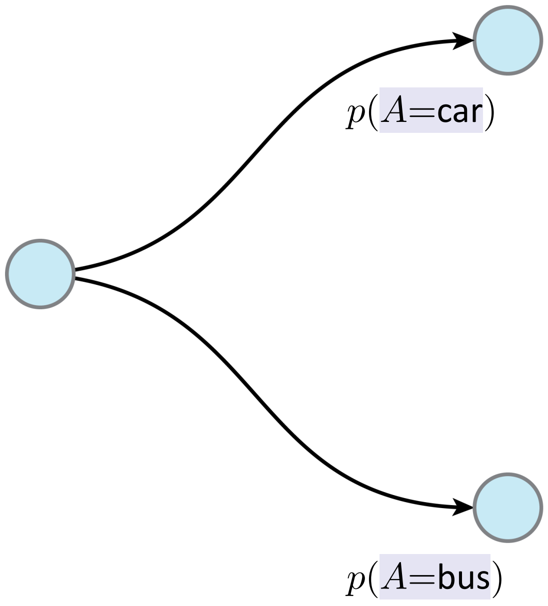
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

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



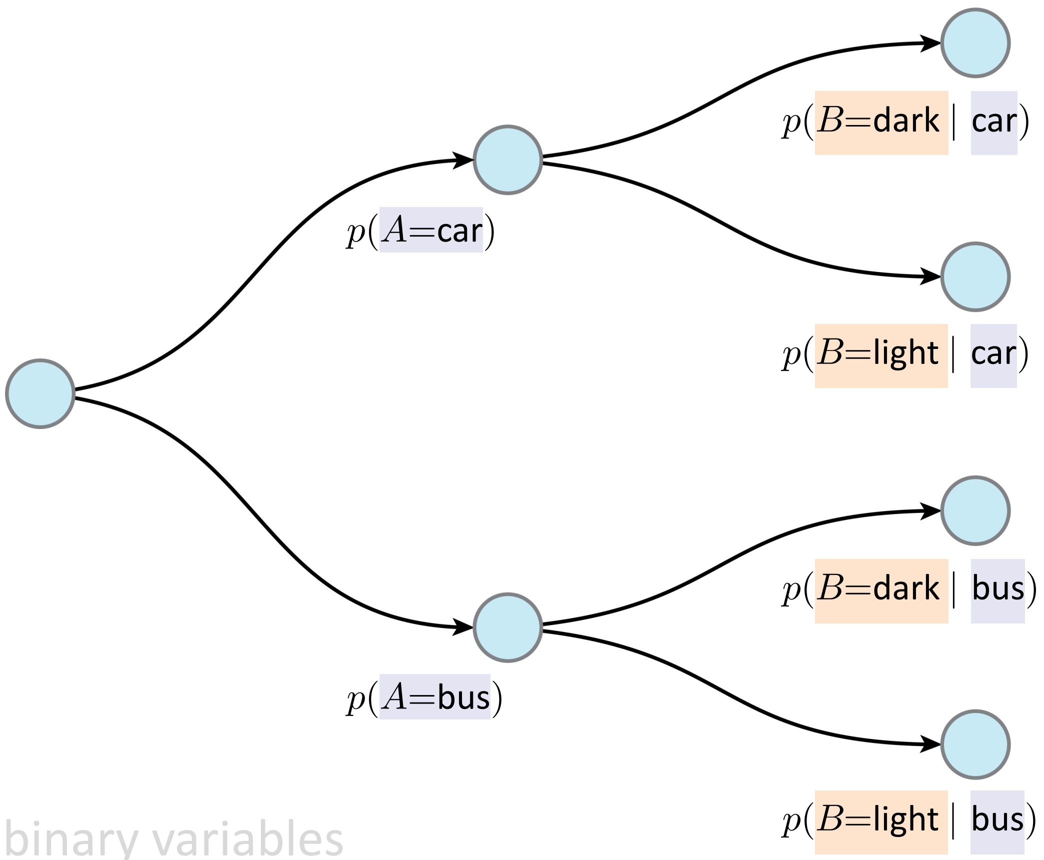
*example: binary variables

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

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



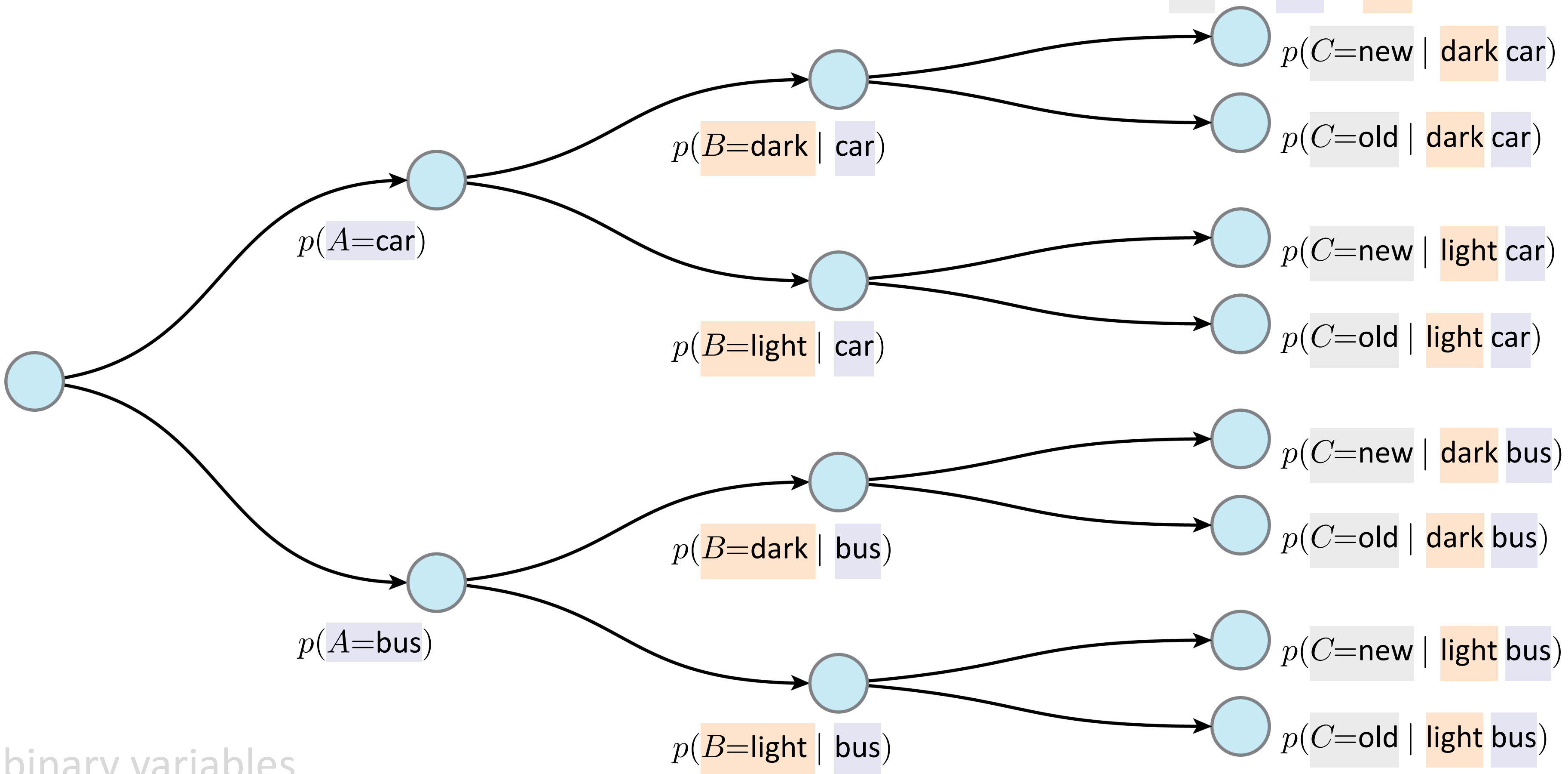
*example: binary variables

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>:

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

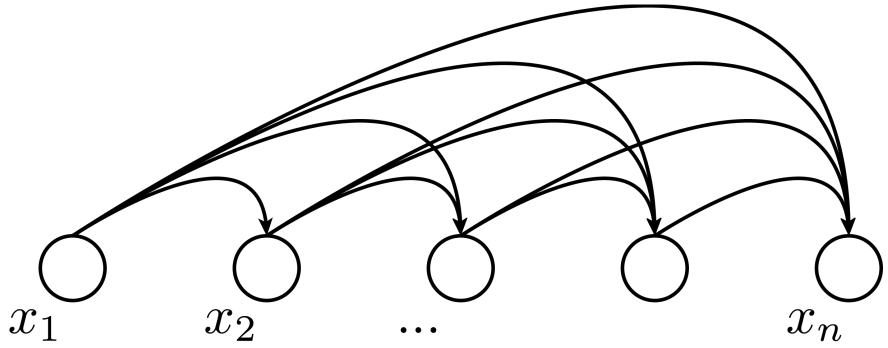
* This is a foundation of Masked Autoregressive (MAR) models:
Li, et al. Autoregressive Image Generation without Vector Quantization, 2024

Case Study: Conditional Distribution Modeling

Case 1: Partitioning the input representation space x

Example: Autoregressive Models on text tokens or pixels

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



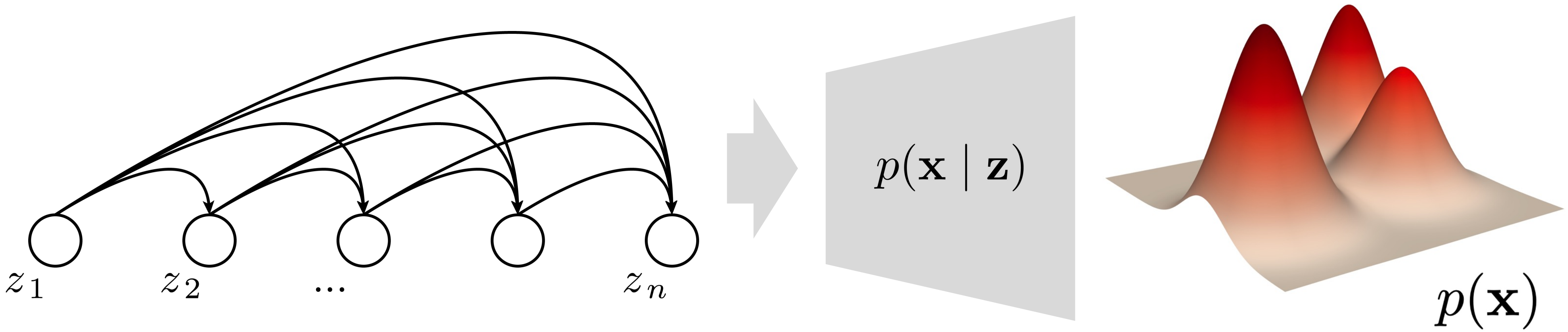
Case Study: Conditional Distribution Modeling

Case 2: Partitioning the latent representation space z

Example: Autoregressive Models on VQ-VAE tokens

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

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

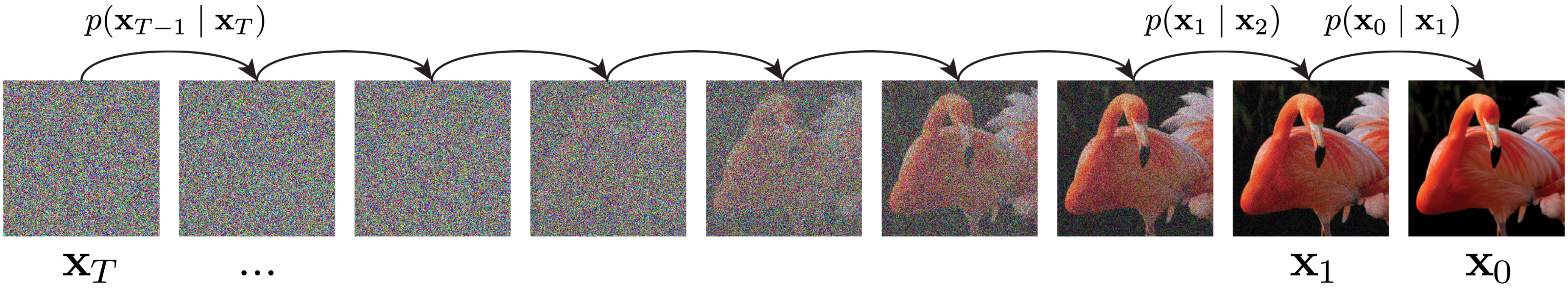


Case Study: Conditional Distribution Modeling

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

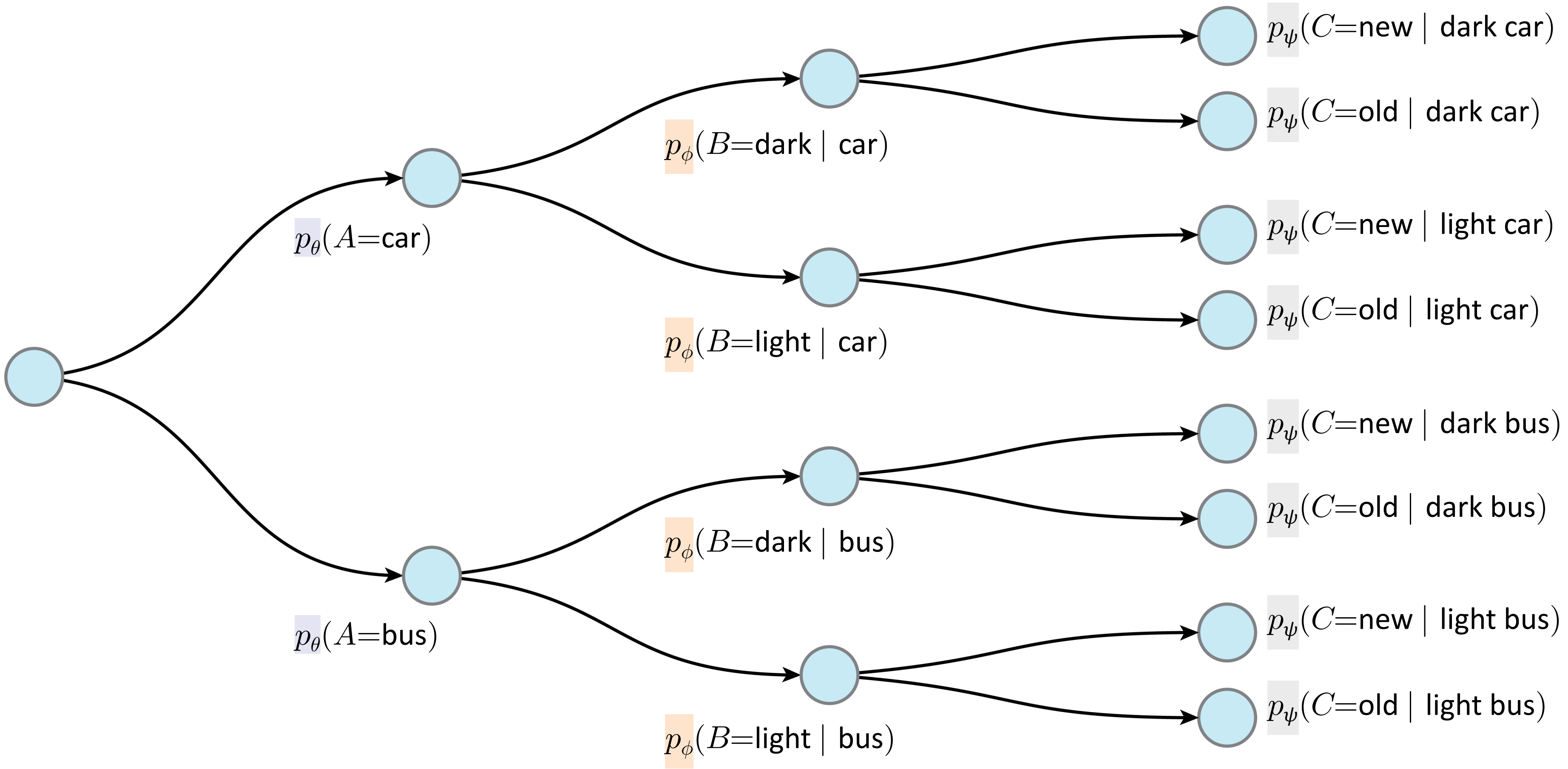
Modeling each conditional distribution with a neural network

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

Conditional Distribution Modeling

Modeling each conditional distribution with a neural network

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



Conditional Distribution Modeling

Modeling each conditional distribution with a neural network

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

Note:

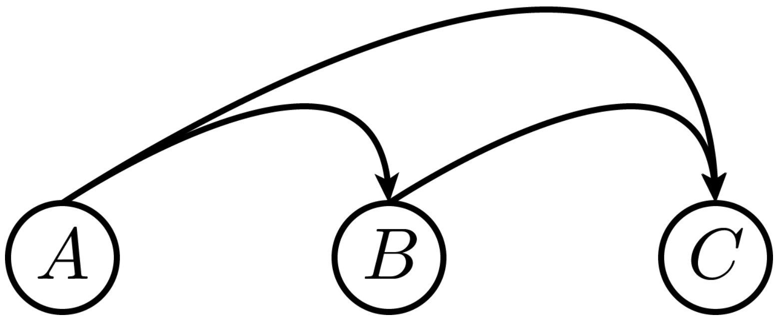
- parameterizing $p(A,\, B,\, C)$ vs. parameterizing $p(C \mid A,\, B)$?
  - $p(A,\, B,\, C)$ has 3 variables
  - $p(C \mid A,\, B)$ has 1 variable and 2 conditions (conditions are network inputs)

- 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

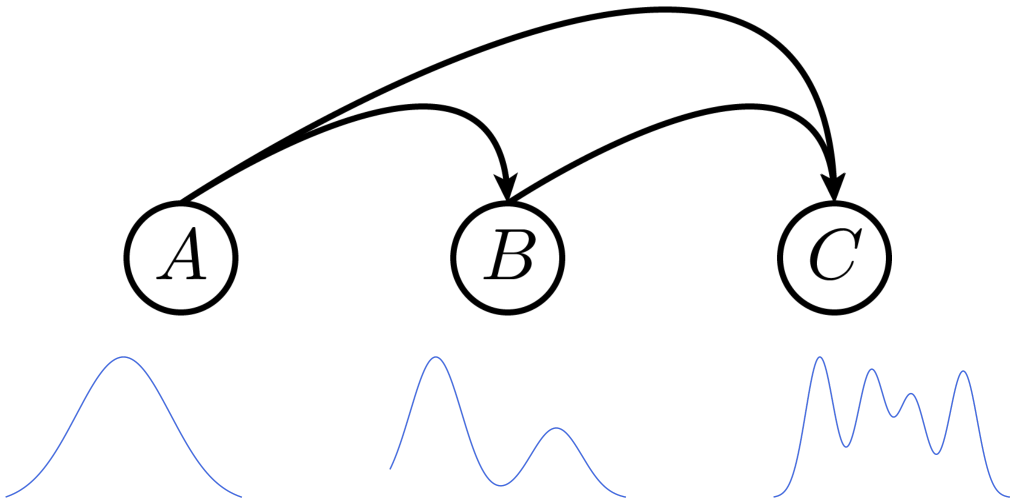
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



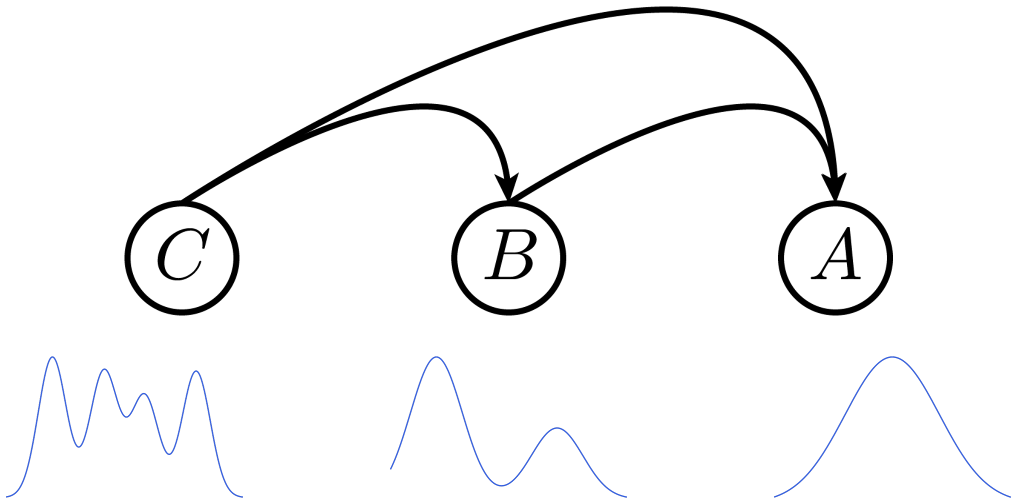
Dependency Graphs

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

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



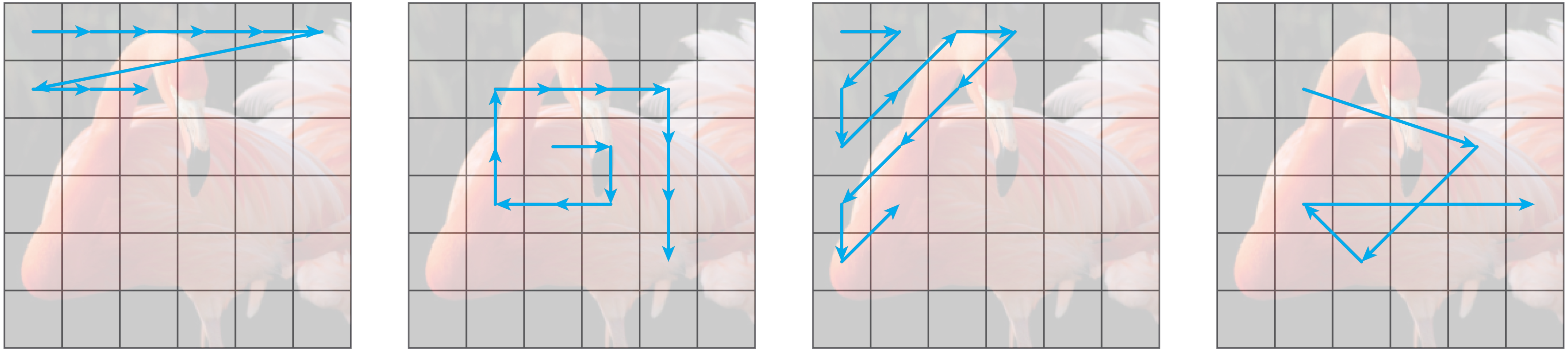
$$p(A,\, B,\, C) = p(C)p(B \mid C)p(A \mid B,\, C)$$



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

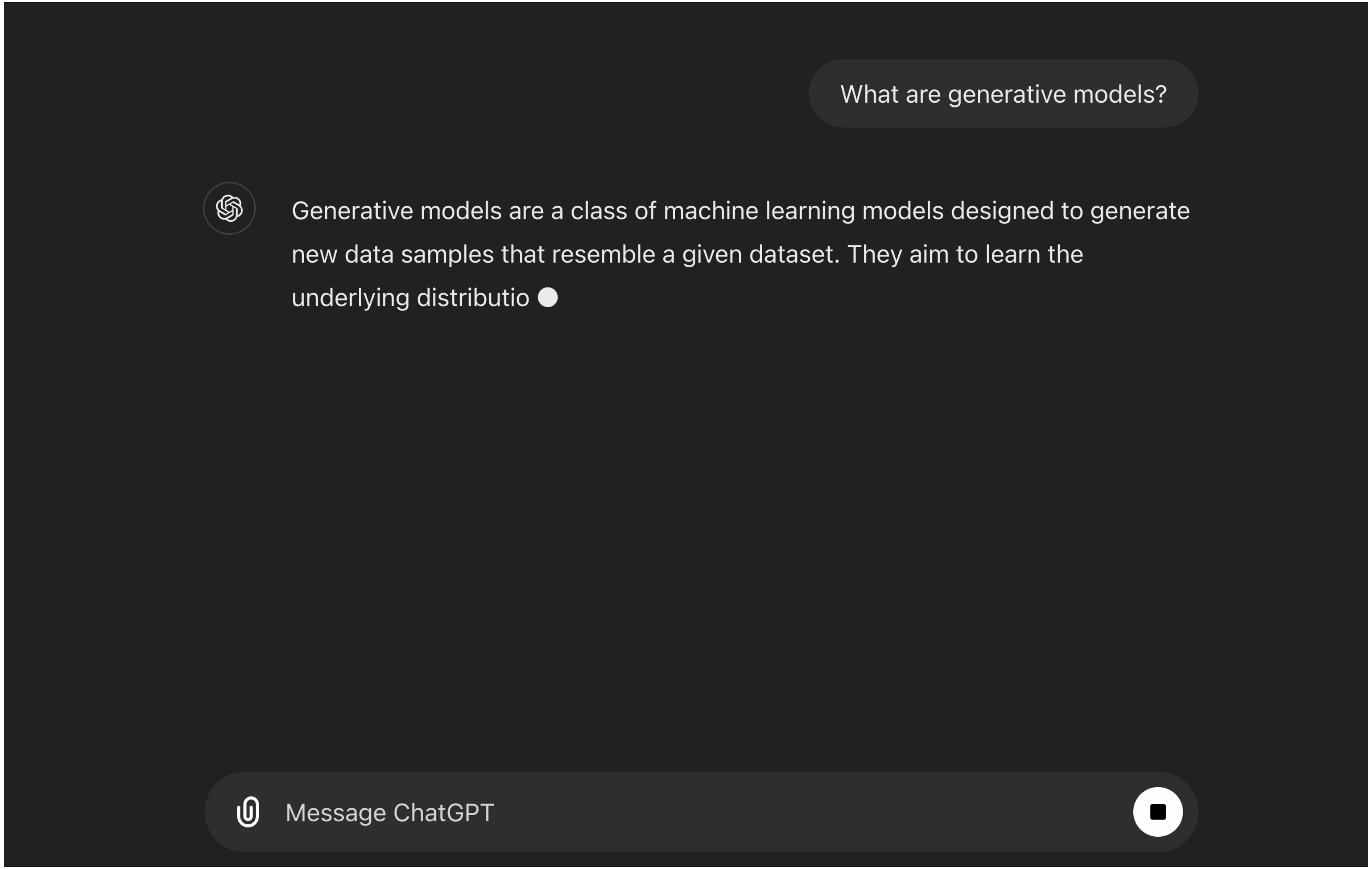
Summary:

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

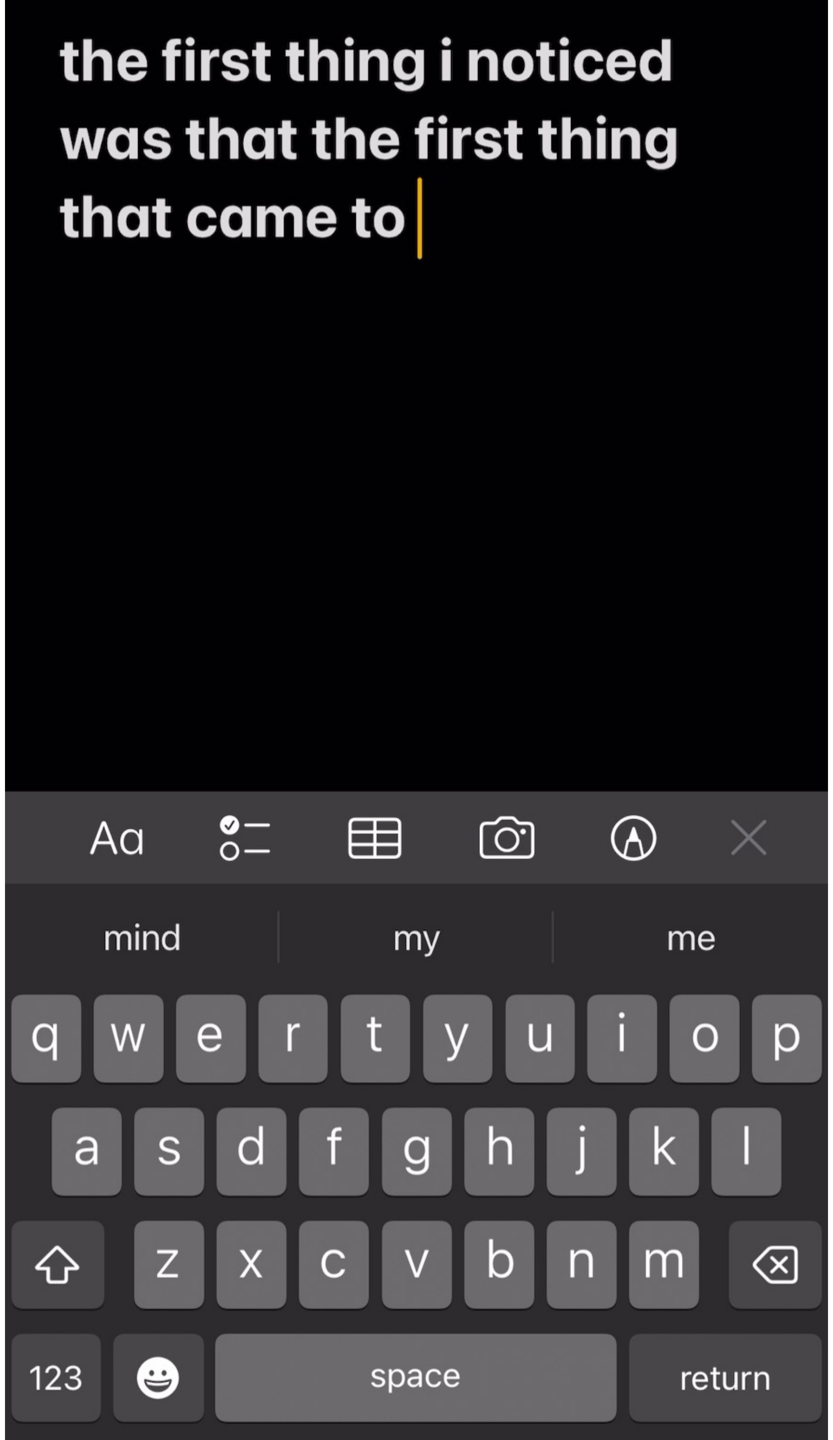
These are <u>not</u> specific to Autoregressive models.

Autoregressive Models

ChatGPT: Next Token Prediction



Your Keyboard



Auto + Regression

Auto: “self”

- using its “own” <u>outputs</u> as inputs for next perditions

Regression:

- estimating relationship between variables

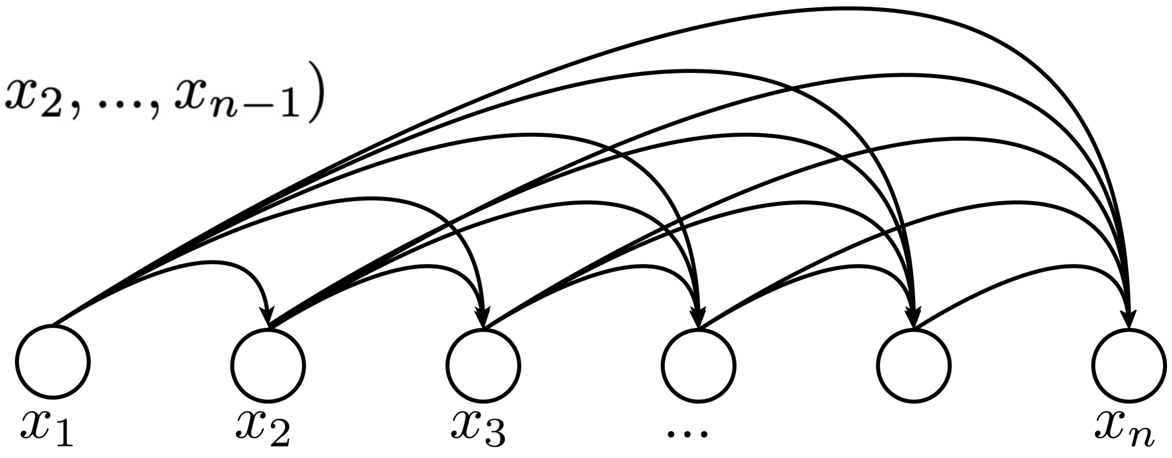
Note:

- “Autoregressive” implies an <u>inference-time</u> 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:

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



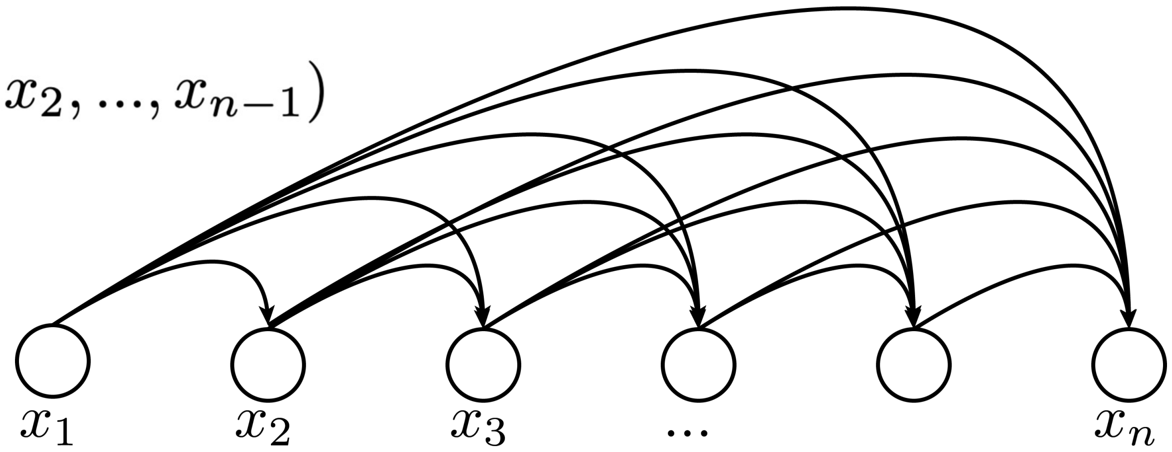
Conceptually, ...

- x can be **any** 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:

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



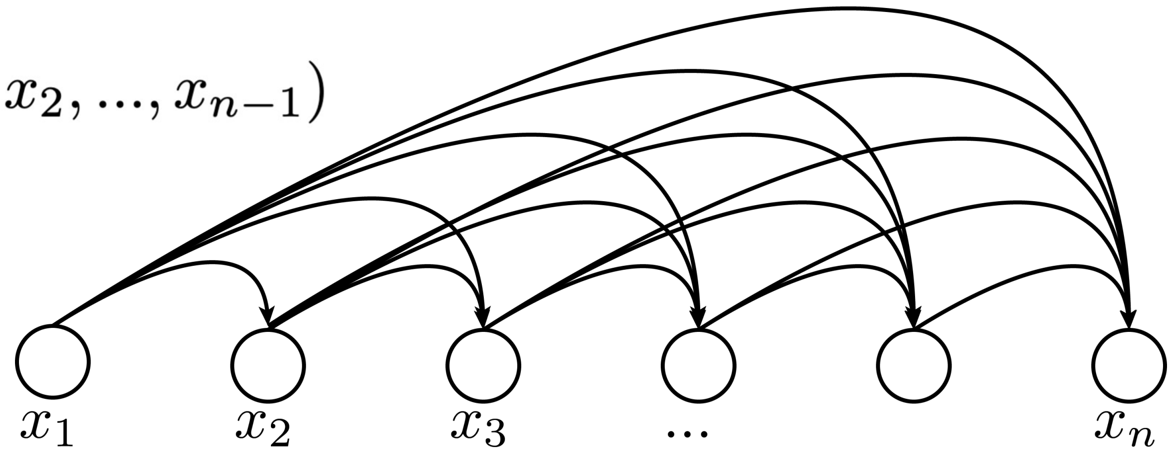
Conceptually, ...

- 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:

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



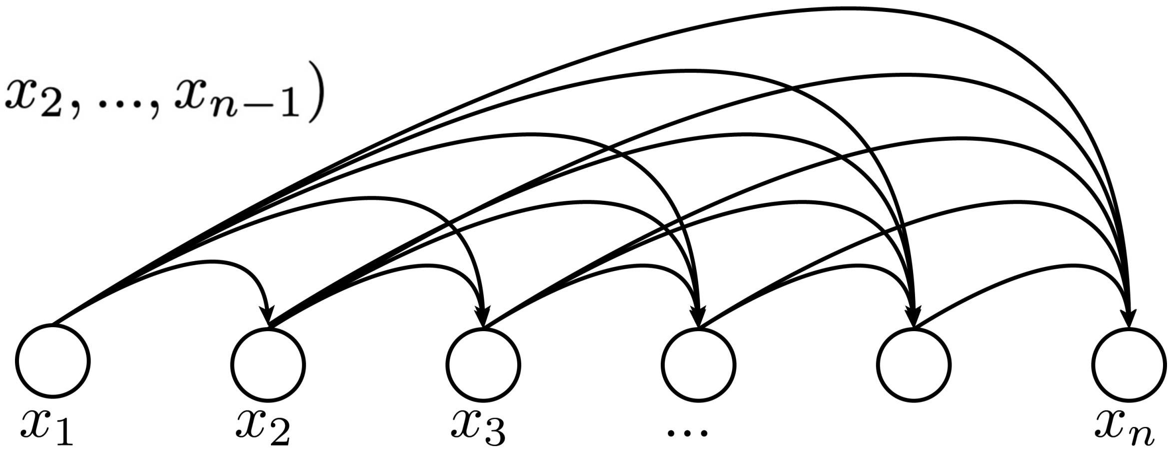
Conceptually, ...

- 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:

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



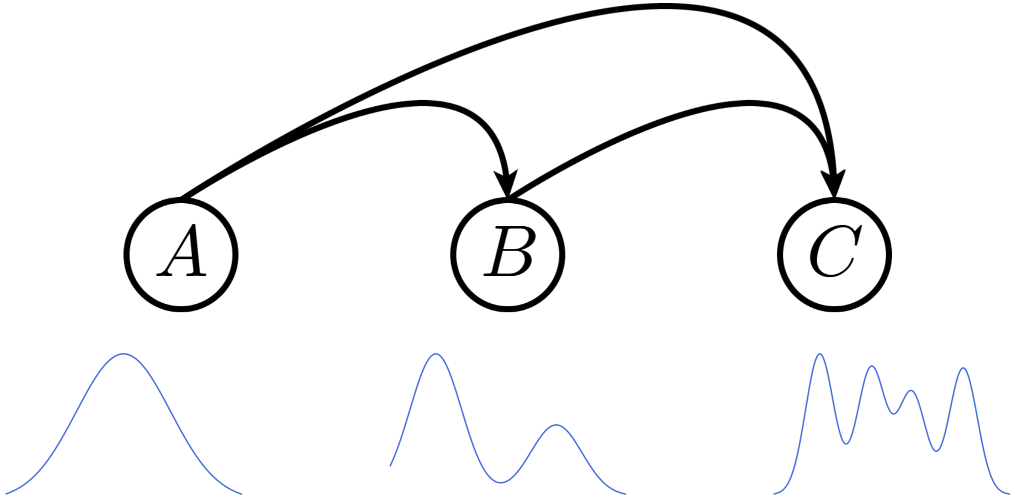
This formulation makes **no** compromise/approximation

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

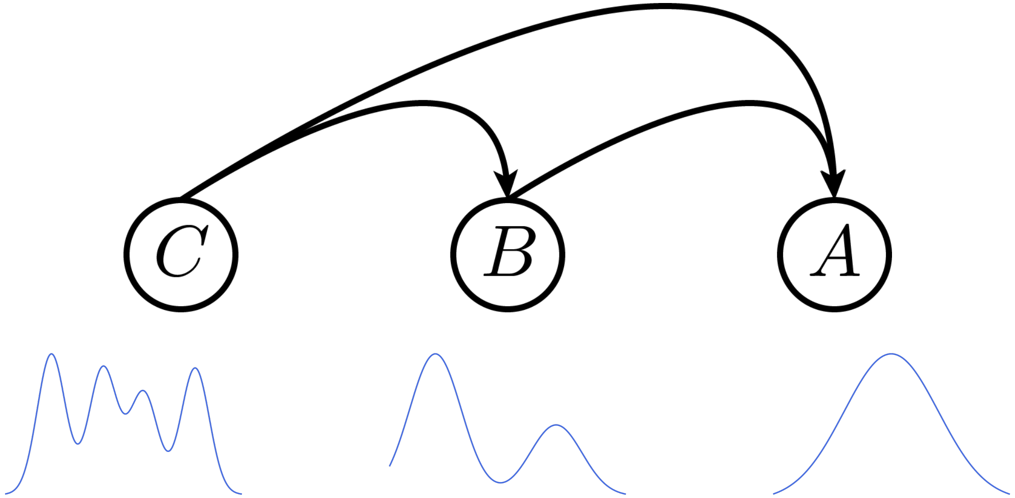
Inductive Bias

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

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

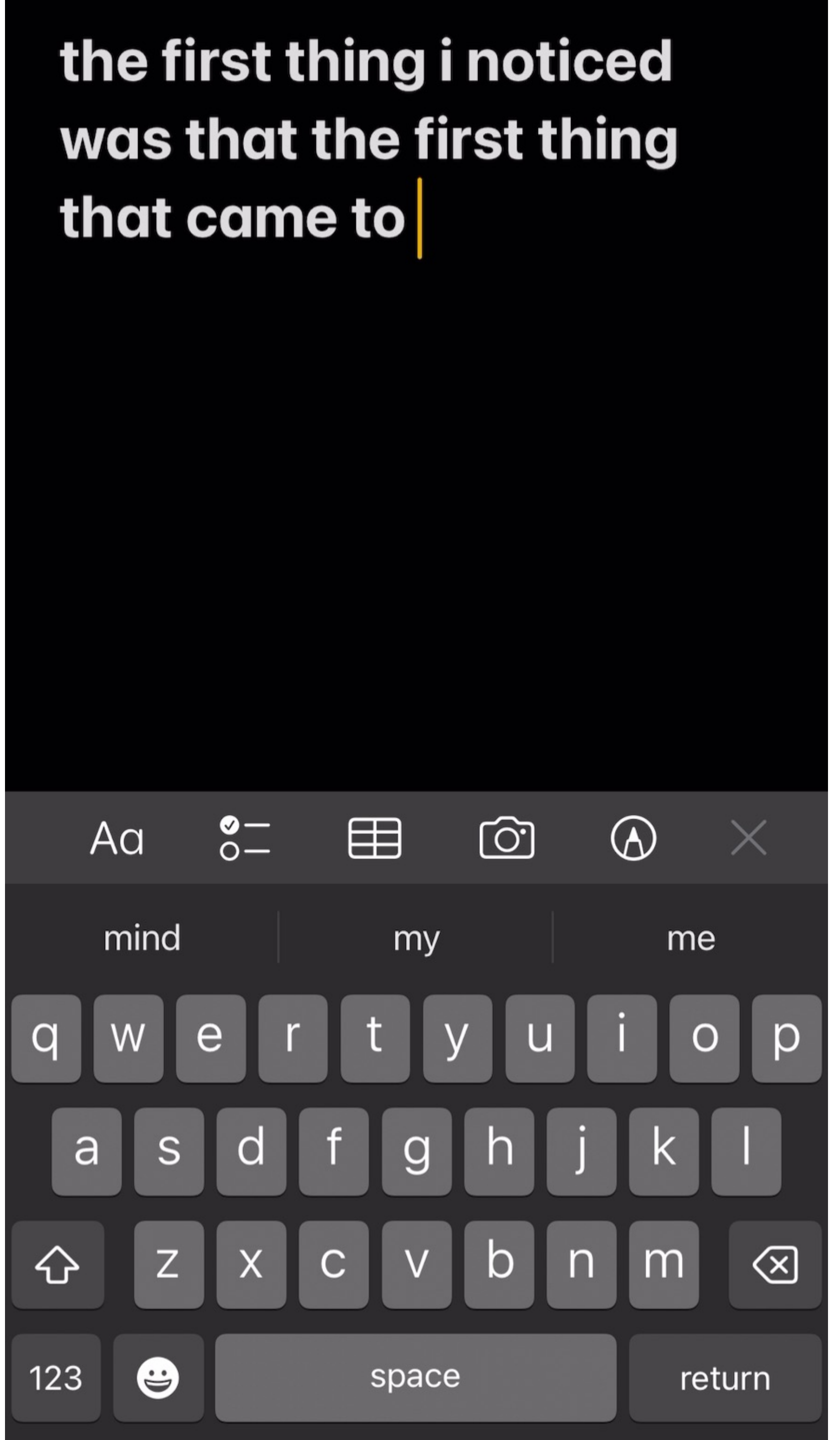


$$p(A, B, C) = p(C)p(B \mid C)p(A \mid B, C)$$



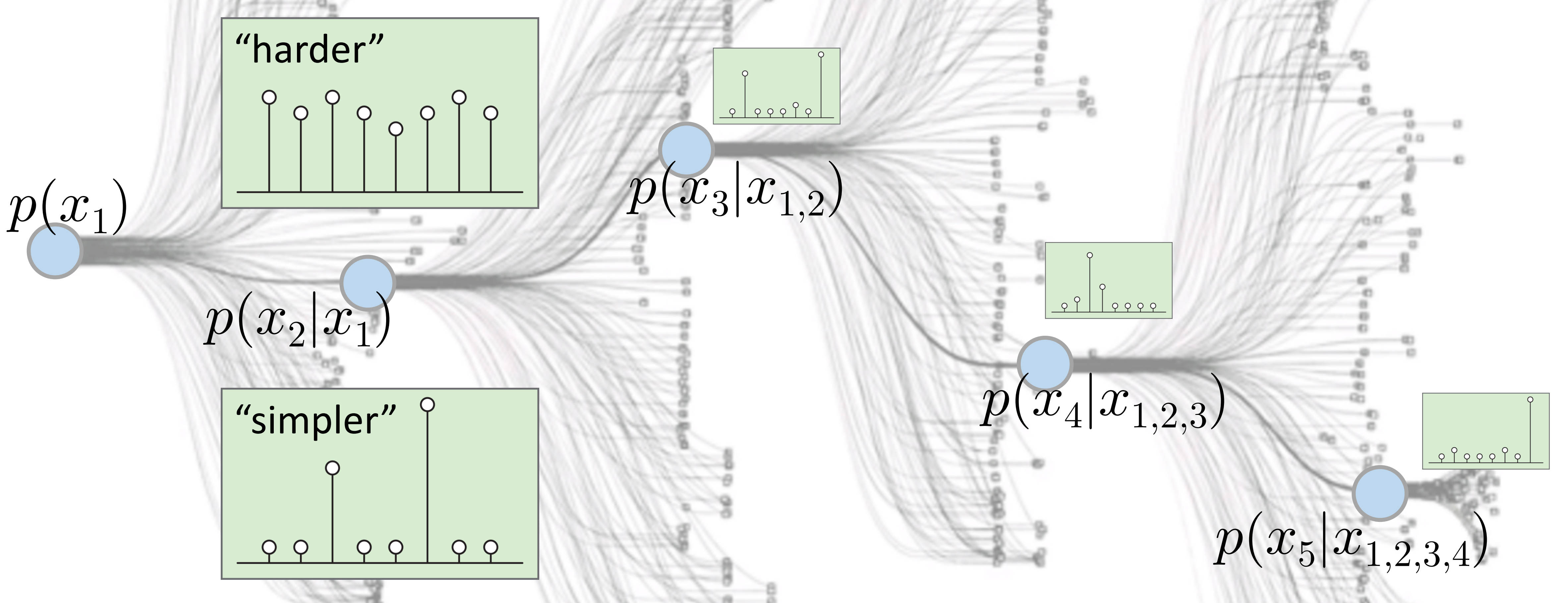
Your phone’s keyboard is Autoregressive:

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

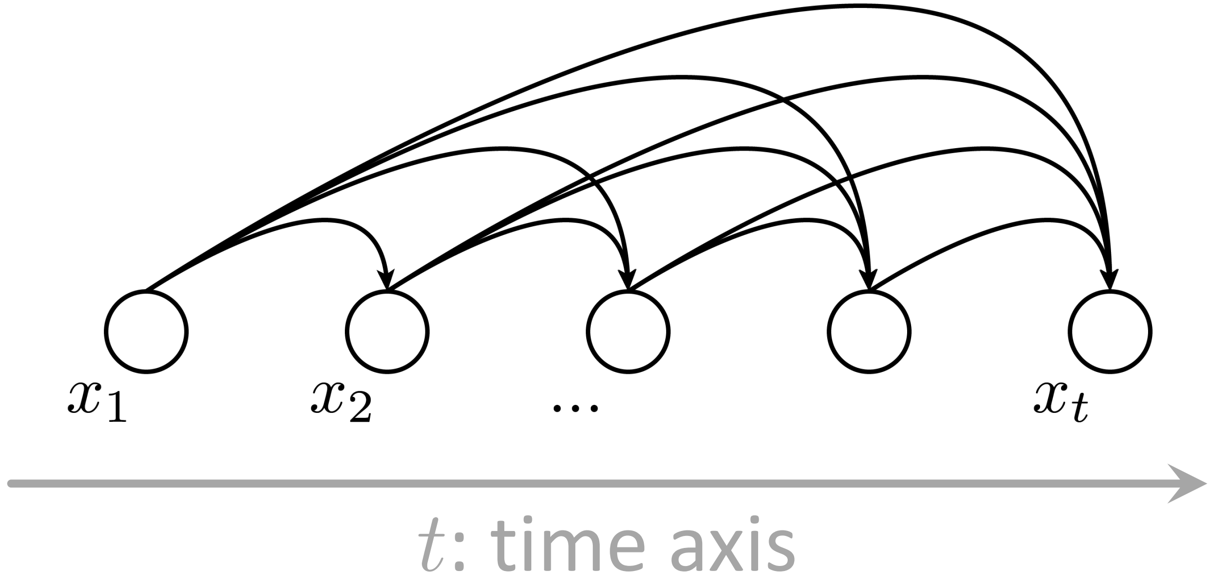
Illustration adapted from AlphaGo

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 ...

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

- 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 ...

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

- 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:

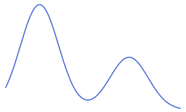
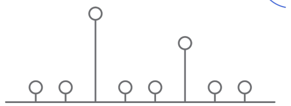
$$p(x_1, x_2, ..., x_n) \quad = \quad p(x_1)p(x_2 \mid x_1)...p(x_n \mid x_1, x_2, ..., x_{n-1})$$

But **inductive biases** introduce approximations:

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

Representing One Distribution

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

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

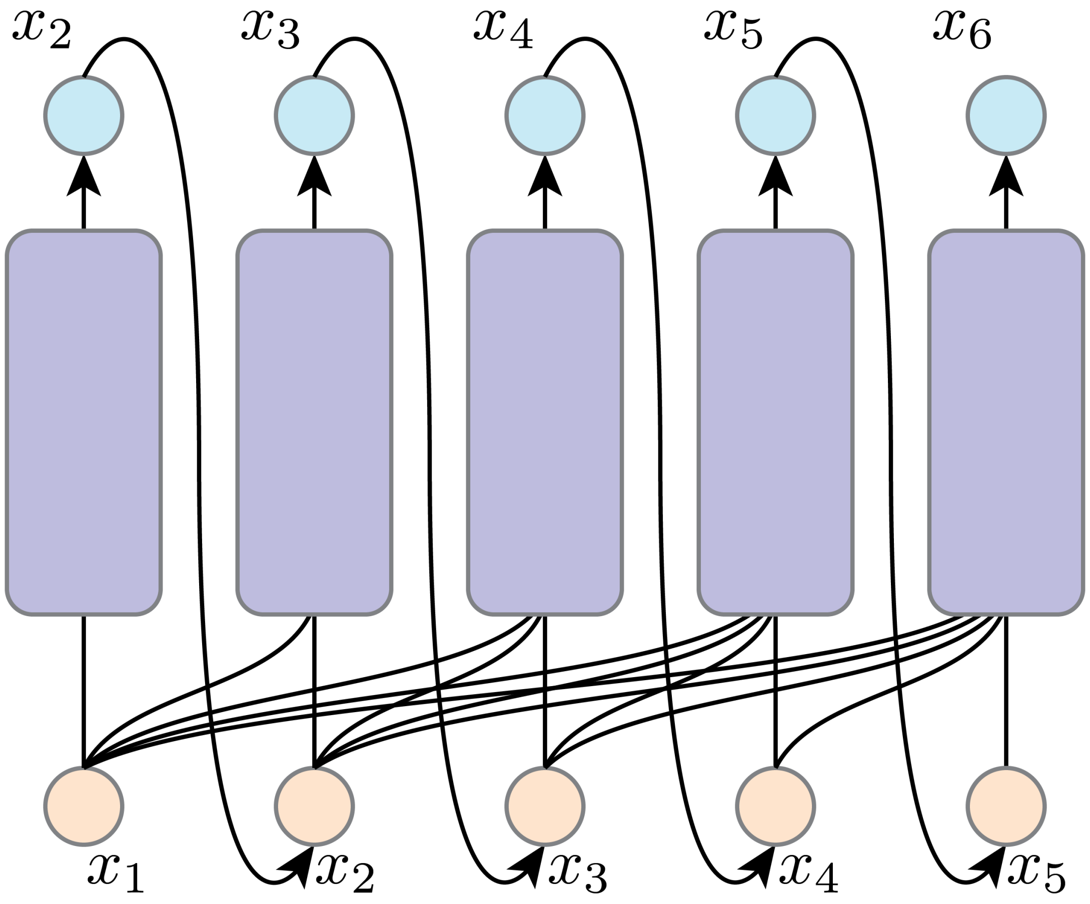
Note:

- 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

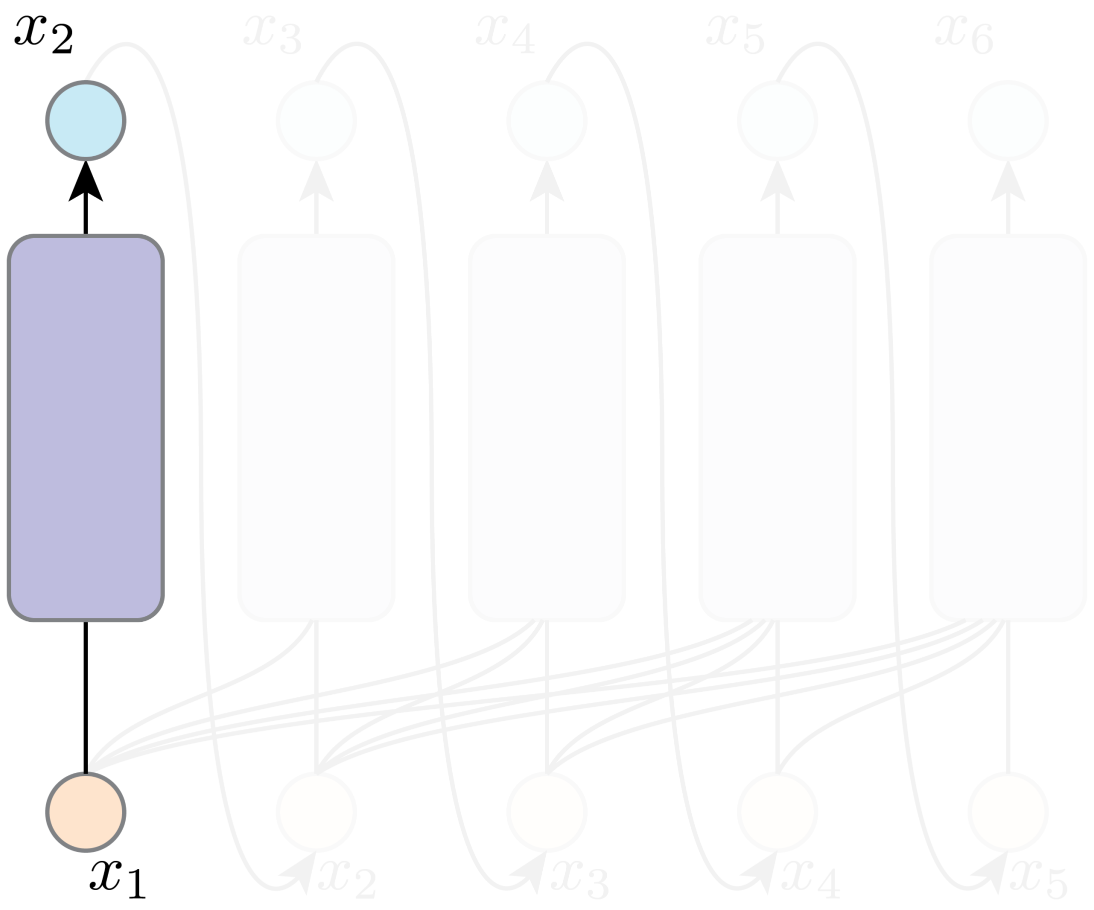
This figure implements this formulation:

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



Inference: Autoregressive

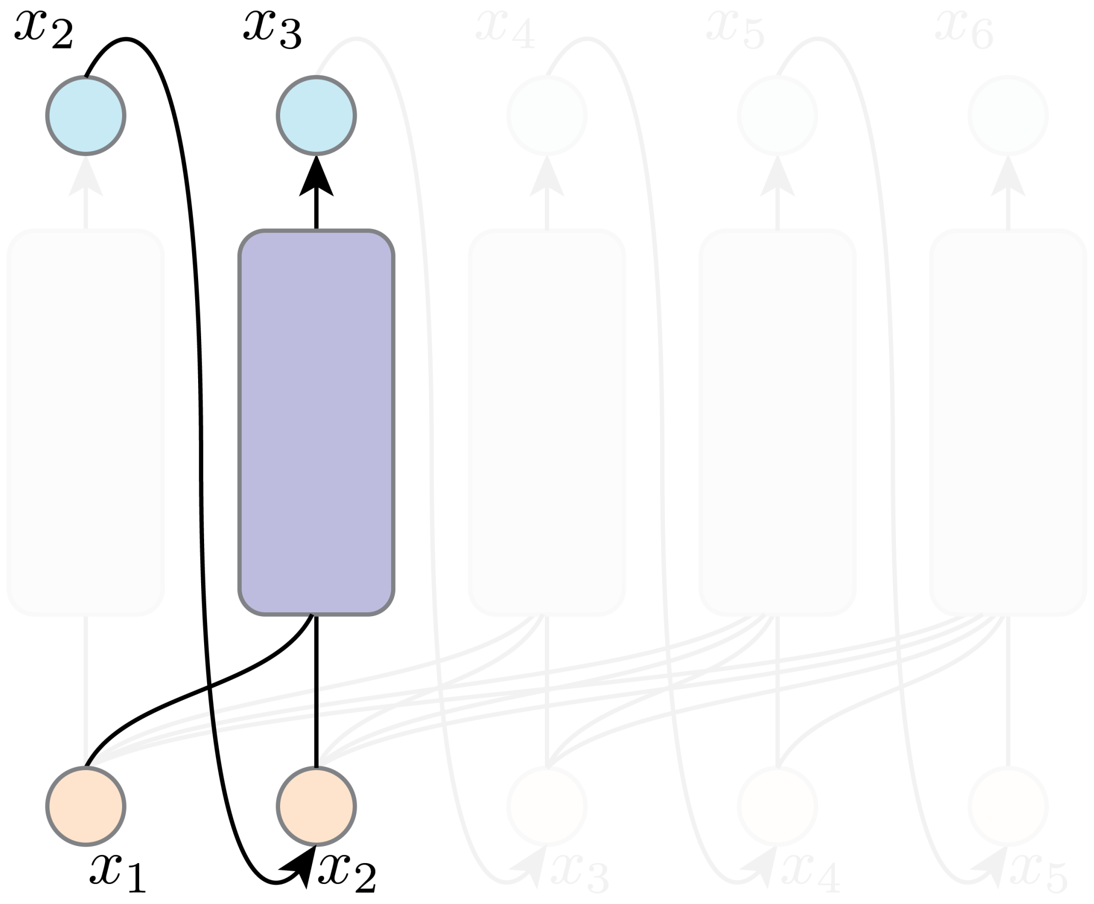
- This net models $p(x_2 \mid x_1)$
- 1 input
- 1 output



Inference: Autoregressive

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

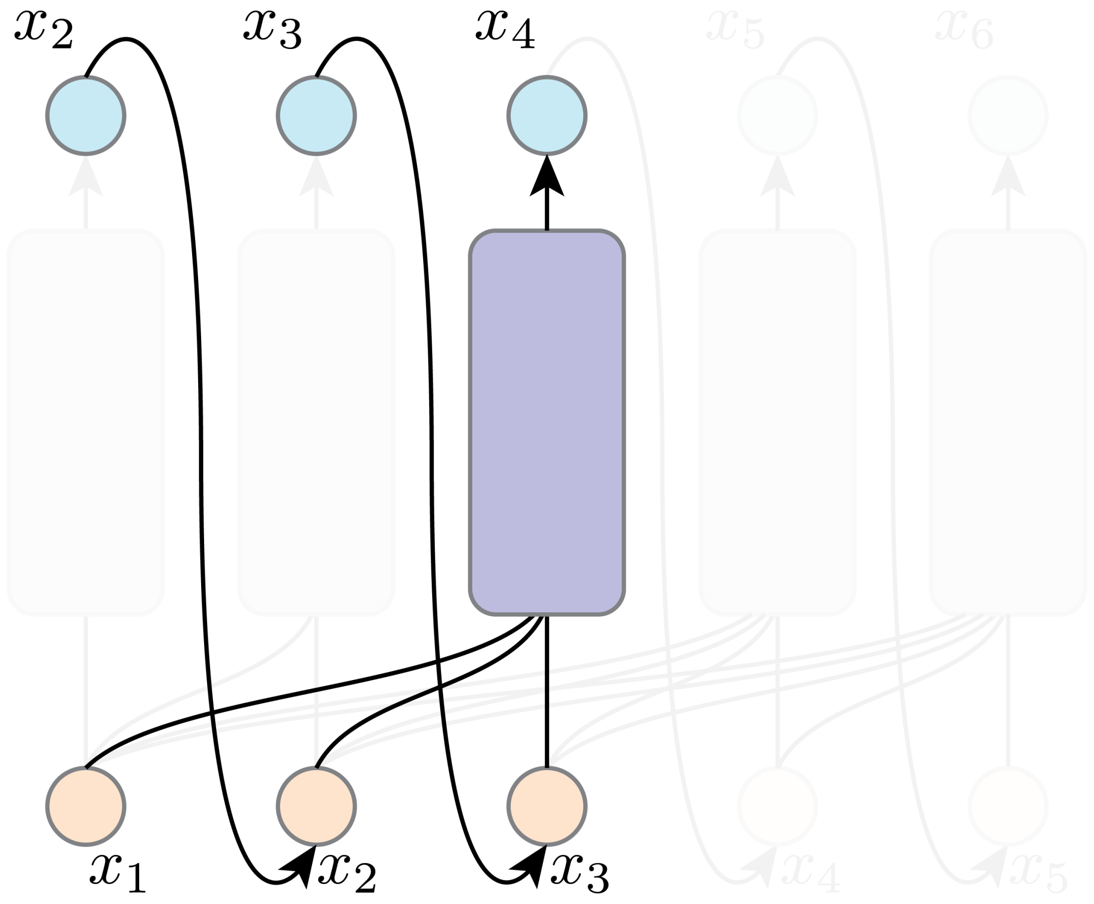
- inputs: outputs from previous steps



Inference: Autoregressive

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

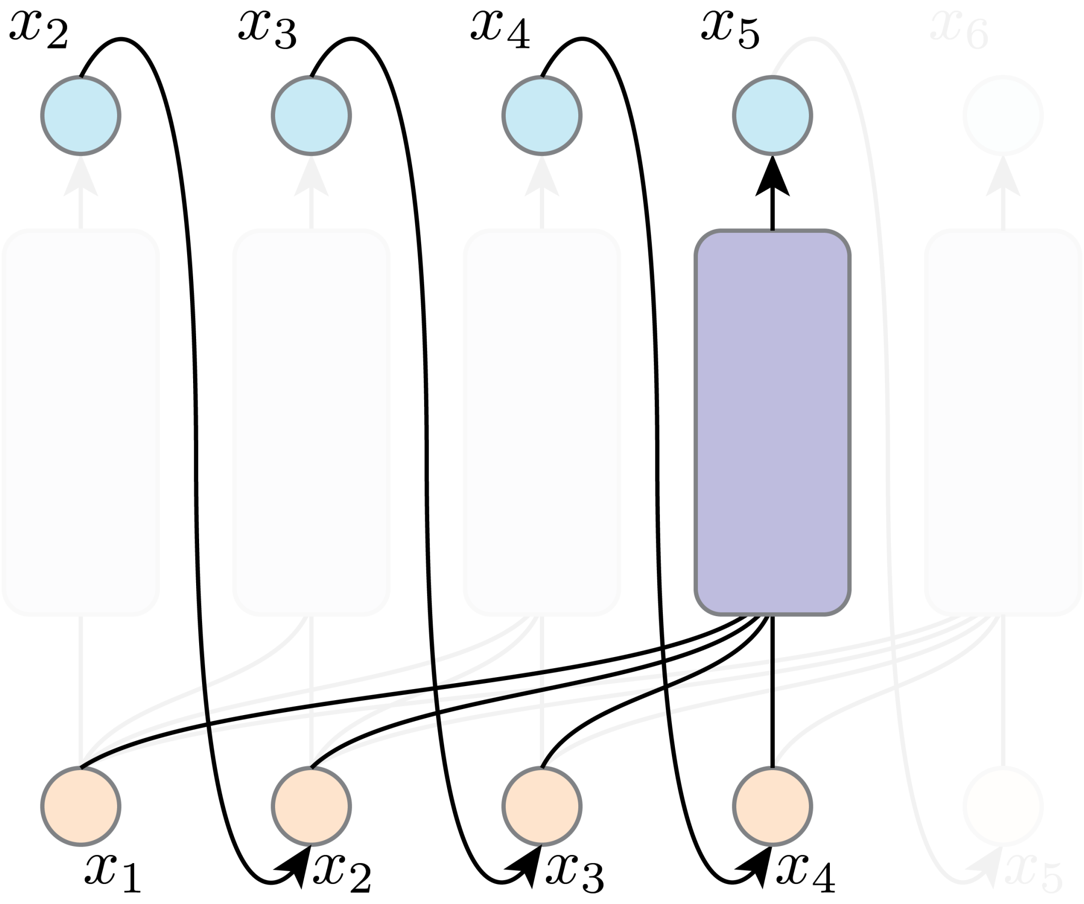
- inputs: outputs from previous steps



Inference: Autoregressive

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

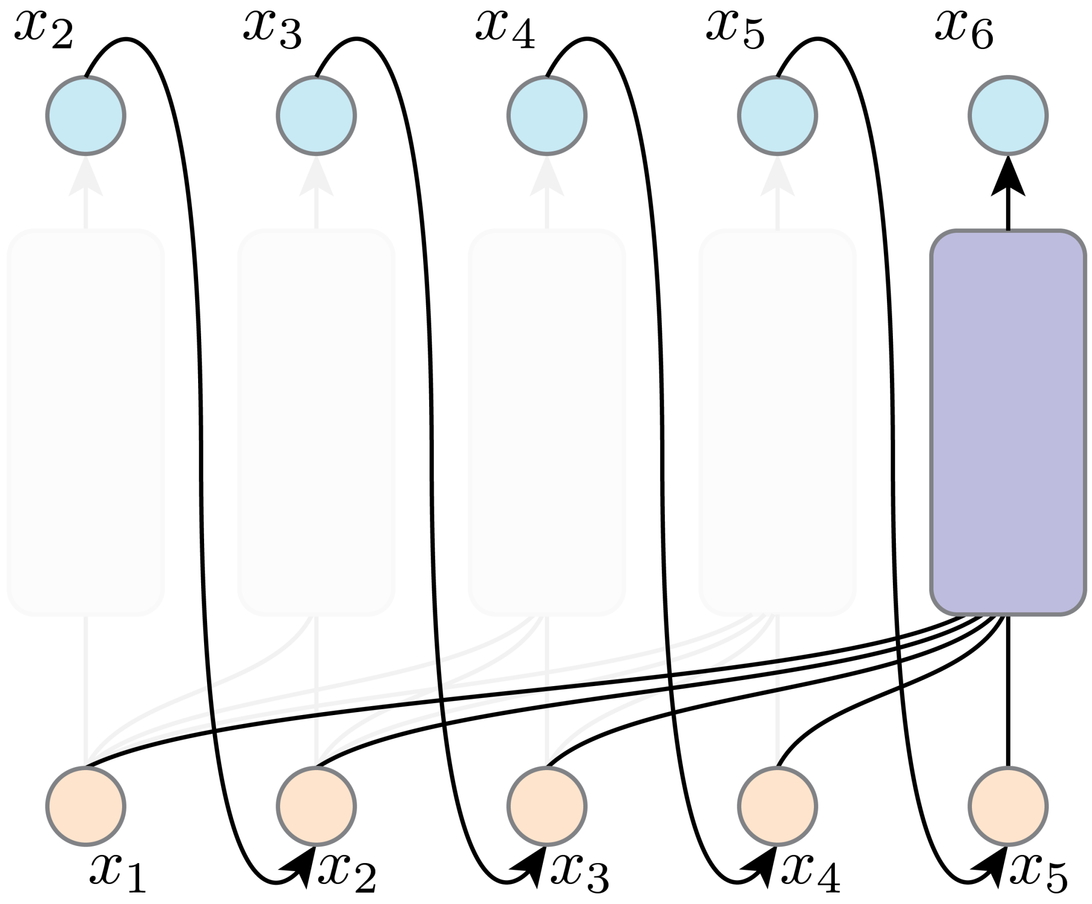
- inputs: outputs from previous steps



Inference: Autoregressive

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

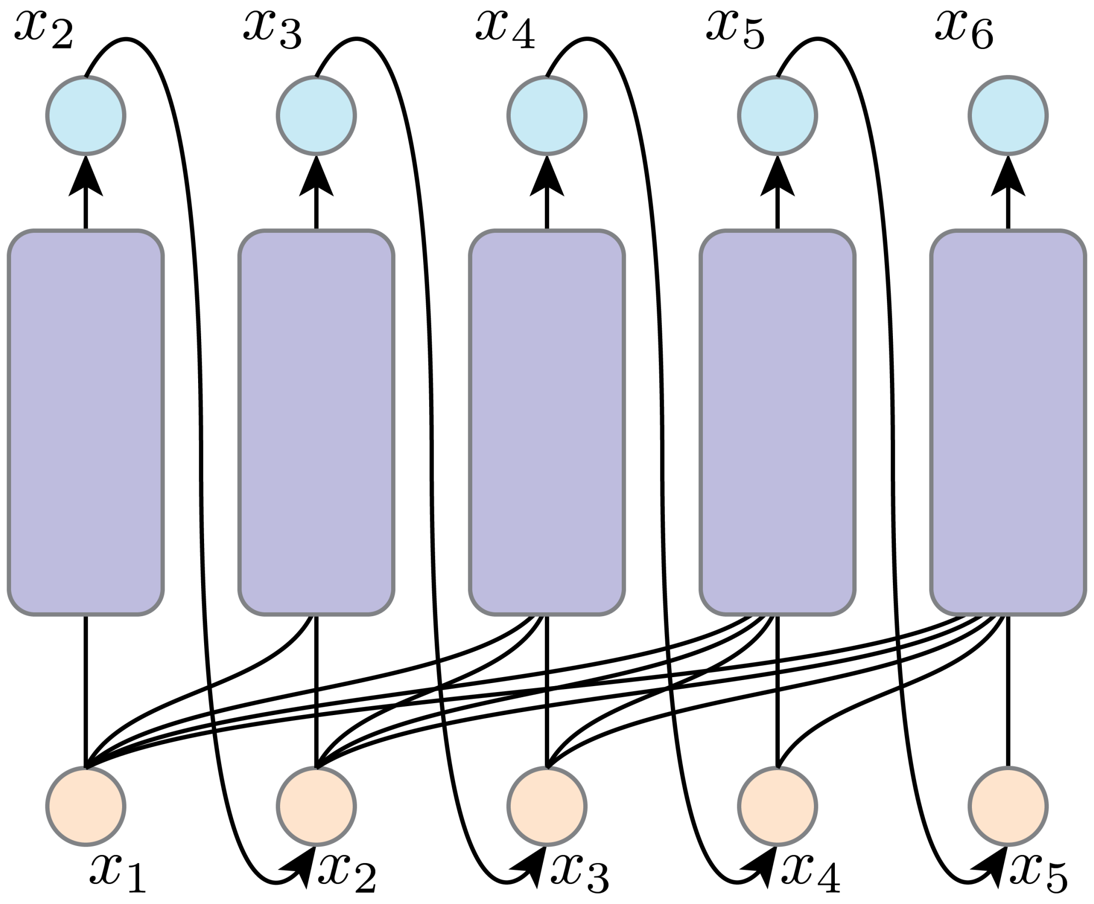
- inputs: outputs from previous steps



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)



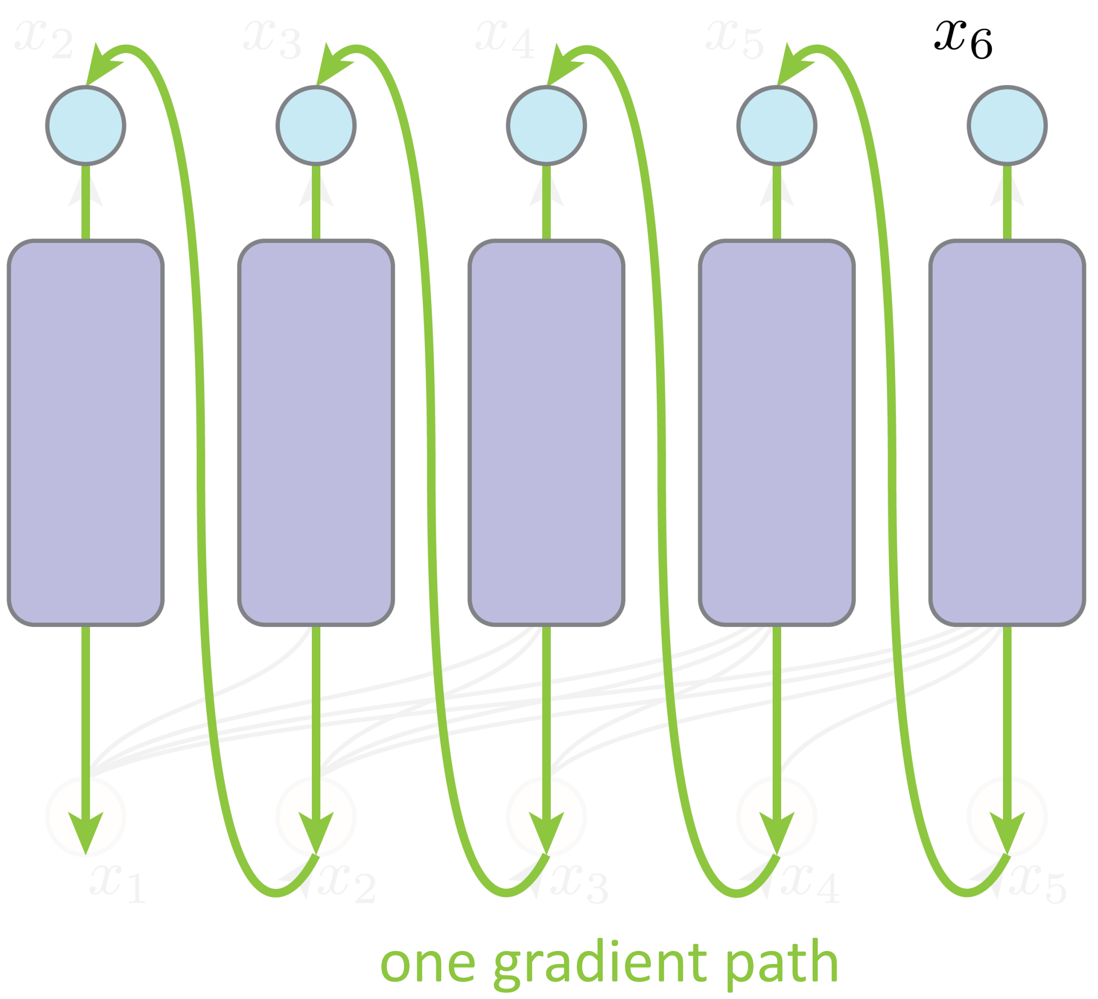
What if we backprop through this graph "as-is"?

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.



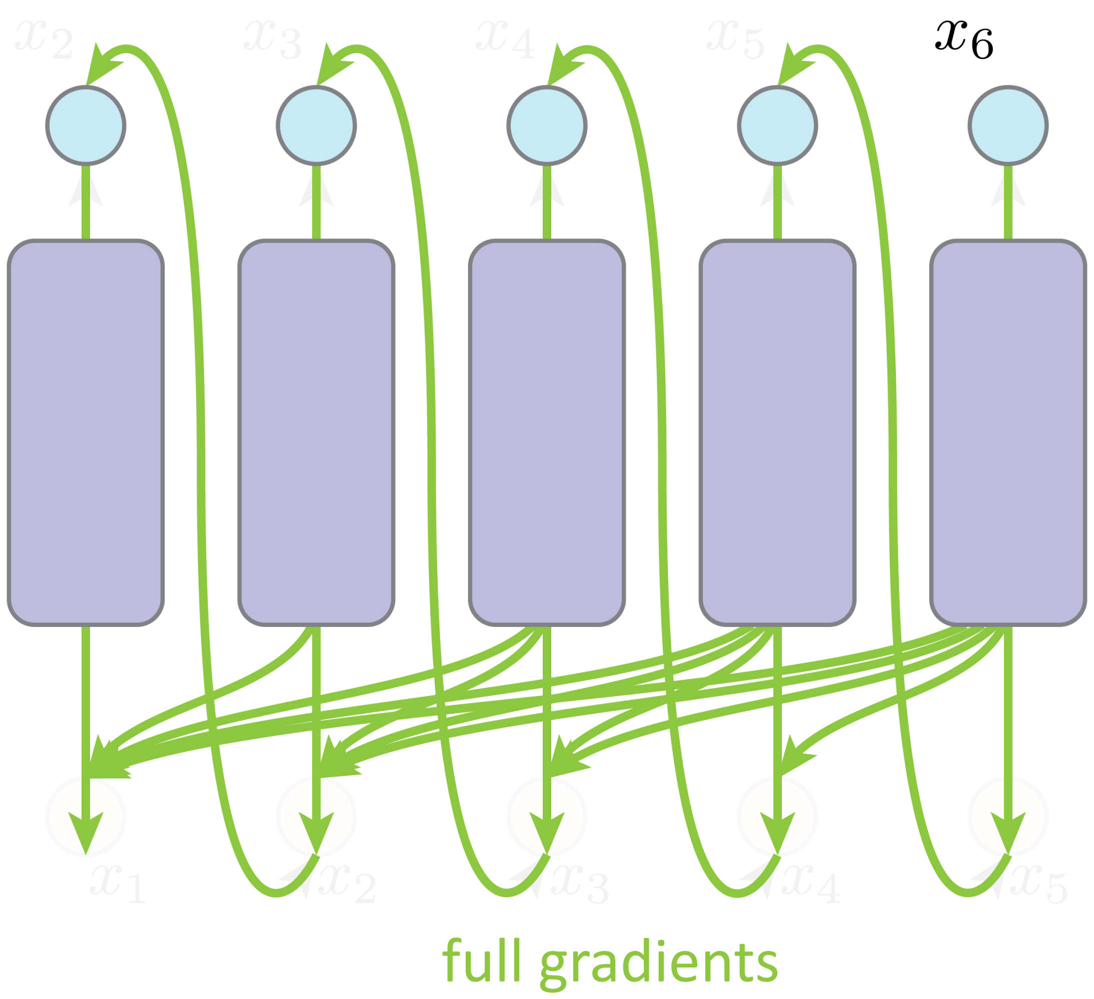
What if we backprop through this graph "as-is"?

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)

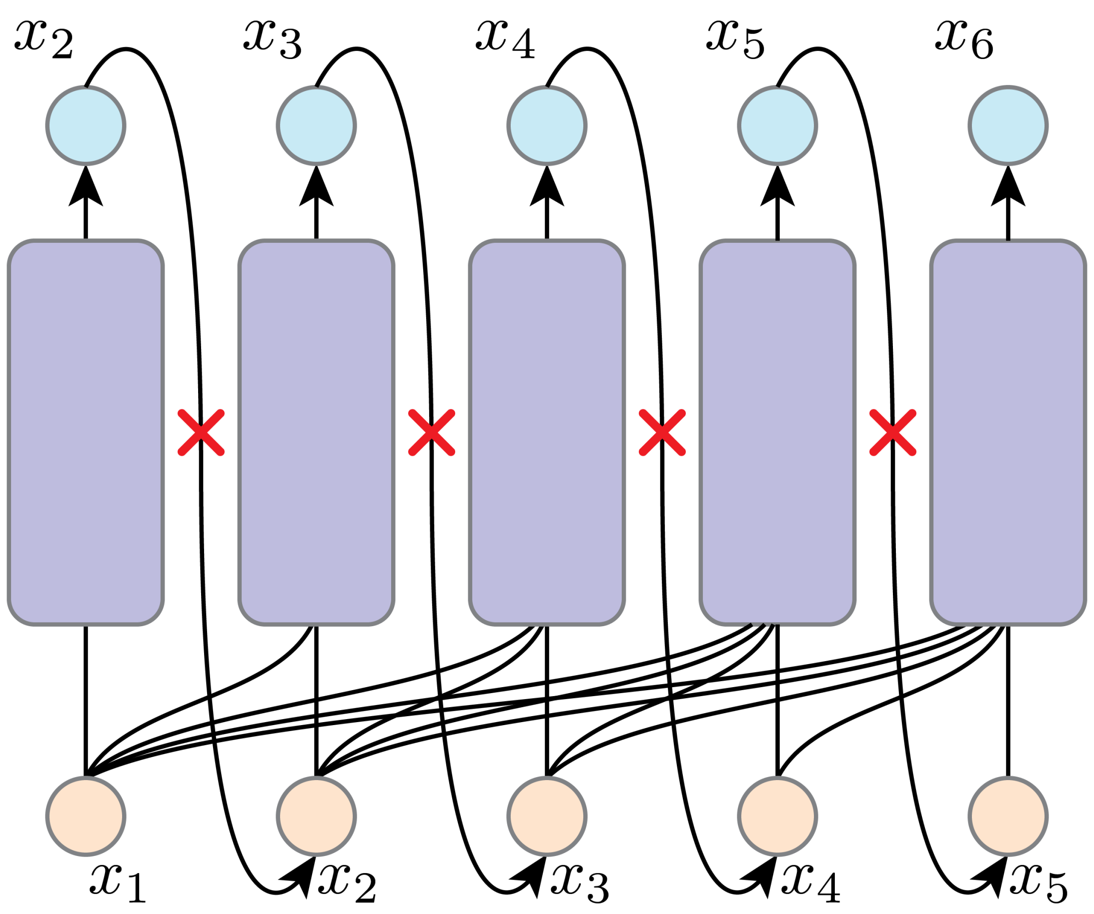
It's **infeasible** to **train** the AR model following its **inference** graph.



Training: Teacher-Forcing

Teacher-forcing

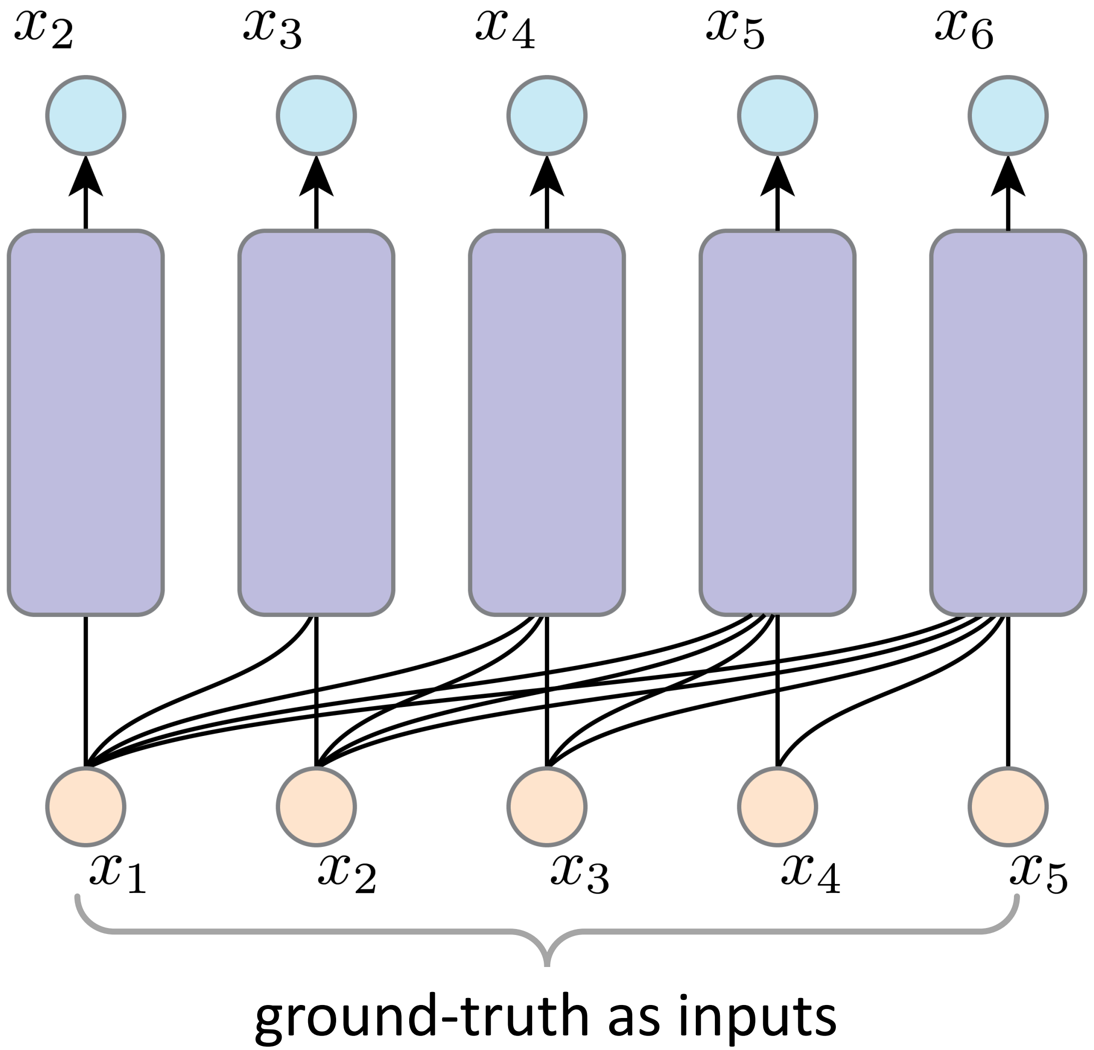
- Inputs are <u>not</u> from previous outputs
- Inputs are from ground-truth data



Training: Teacher-Forcing

Teacher-forcing

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



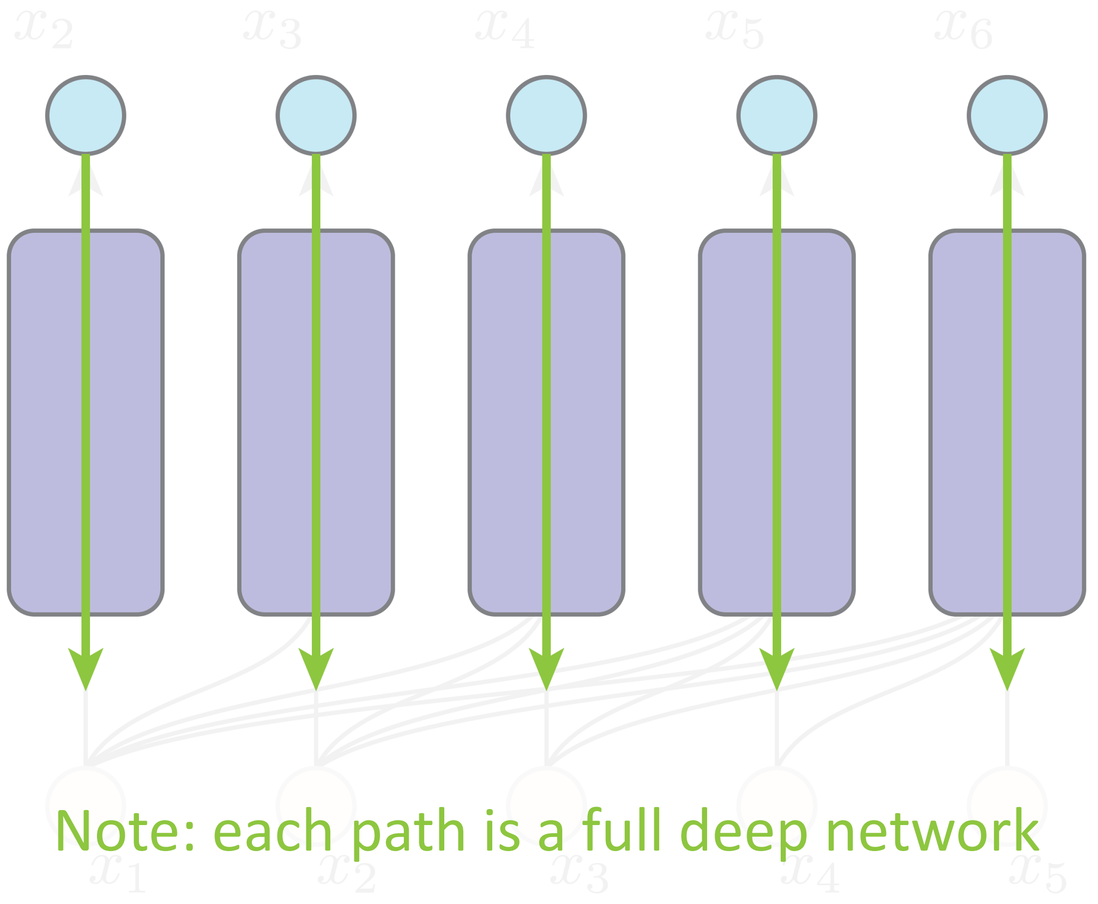
Training: Teacher-Forcing

Teacher-forcing

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

Pros:

- backprop path is much shorter



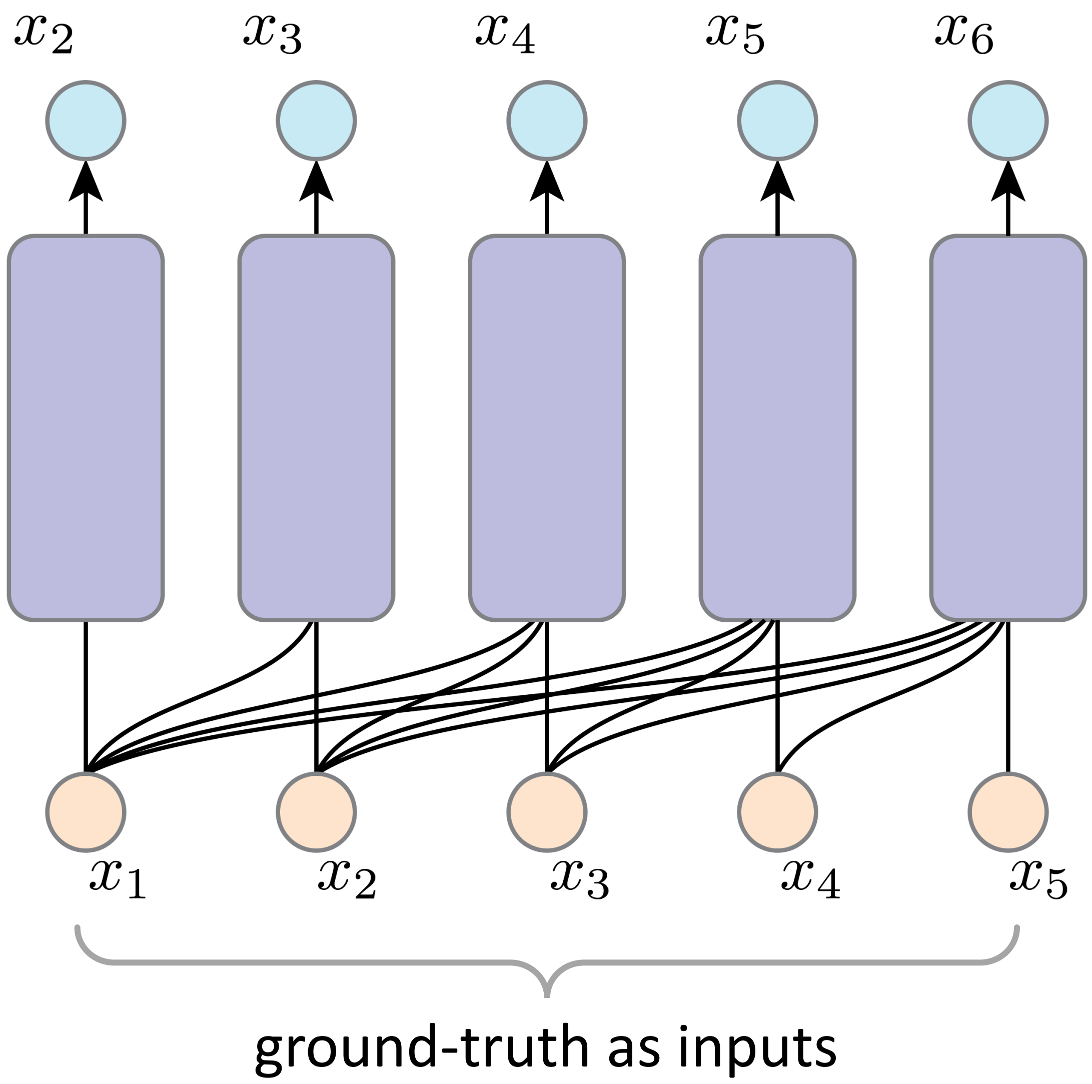
Training: Teacher-Forcing

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

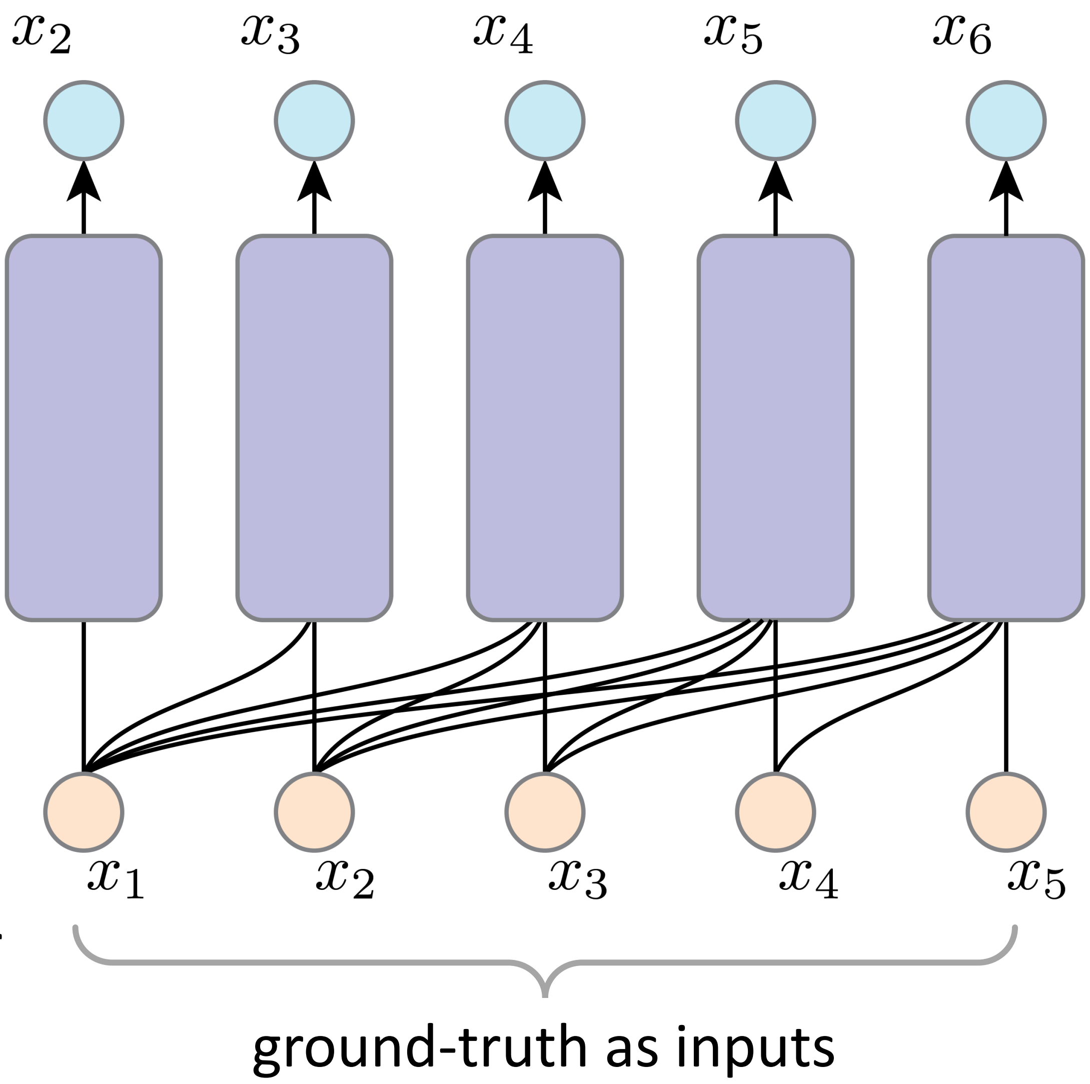
- Inputs are <u>not</u> from previous outputs
- Inputs are from ground-truth data

Pros:

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

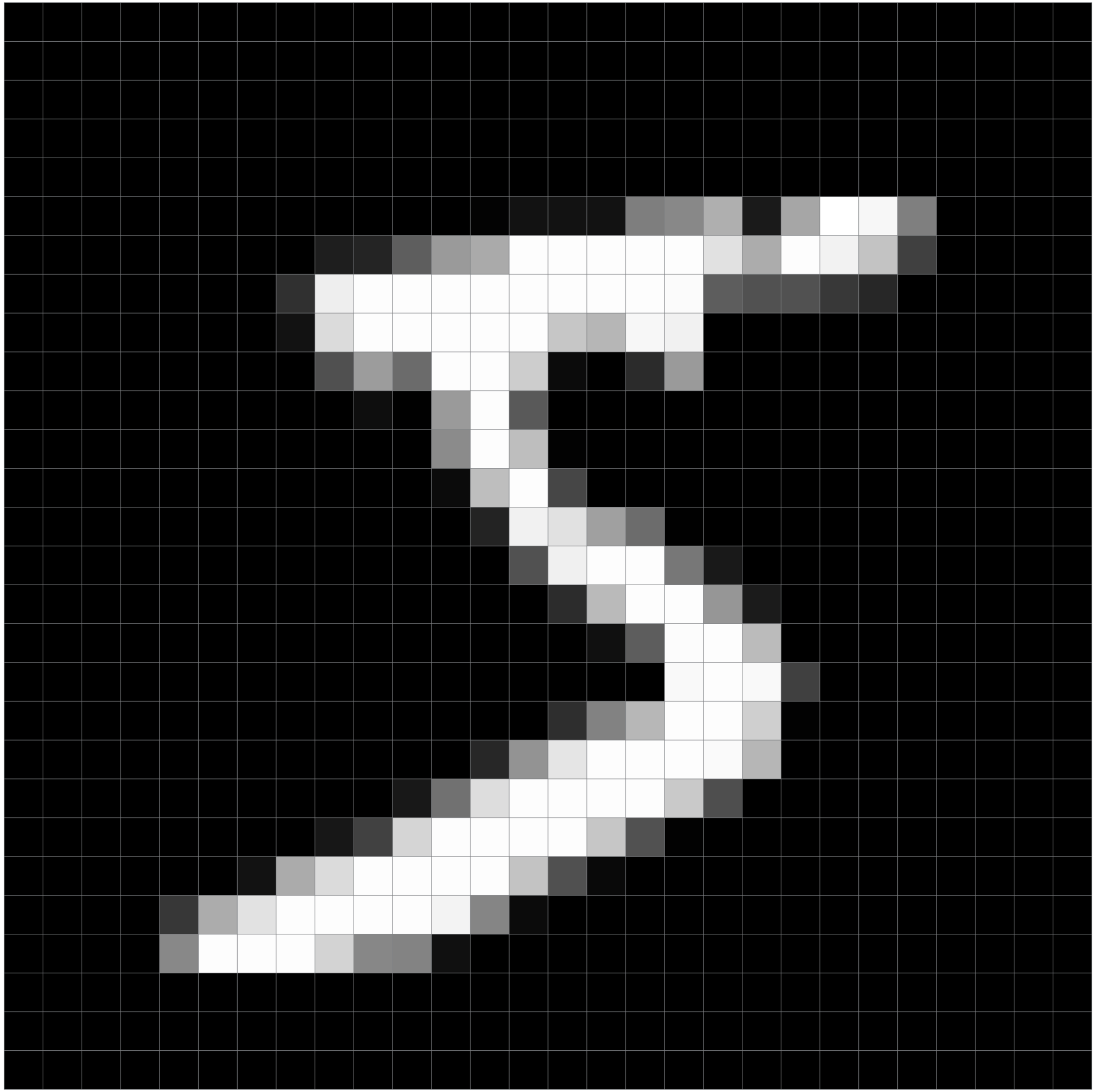
Cons:

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



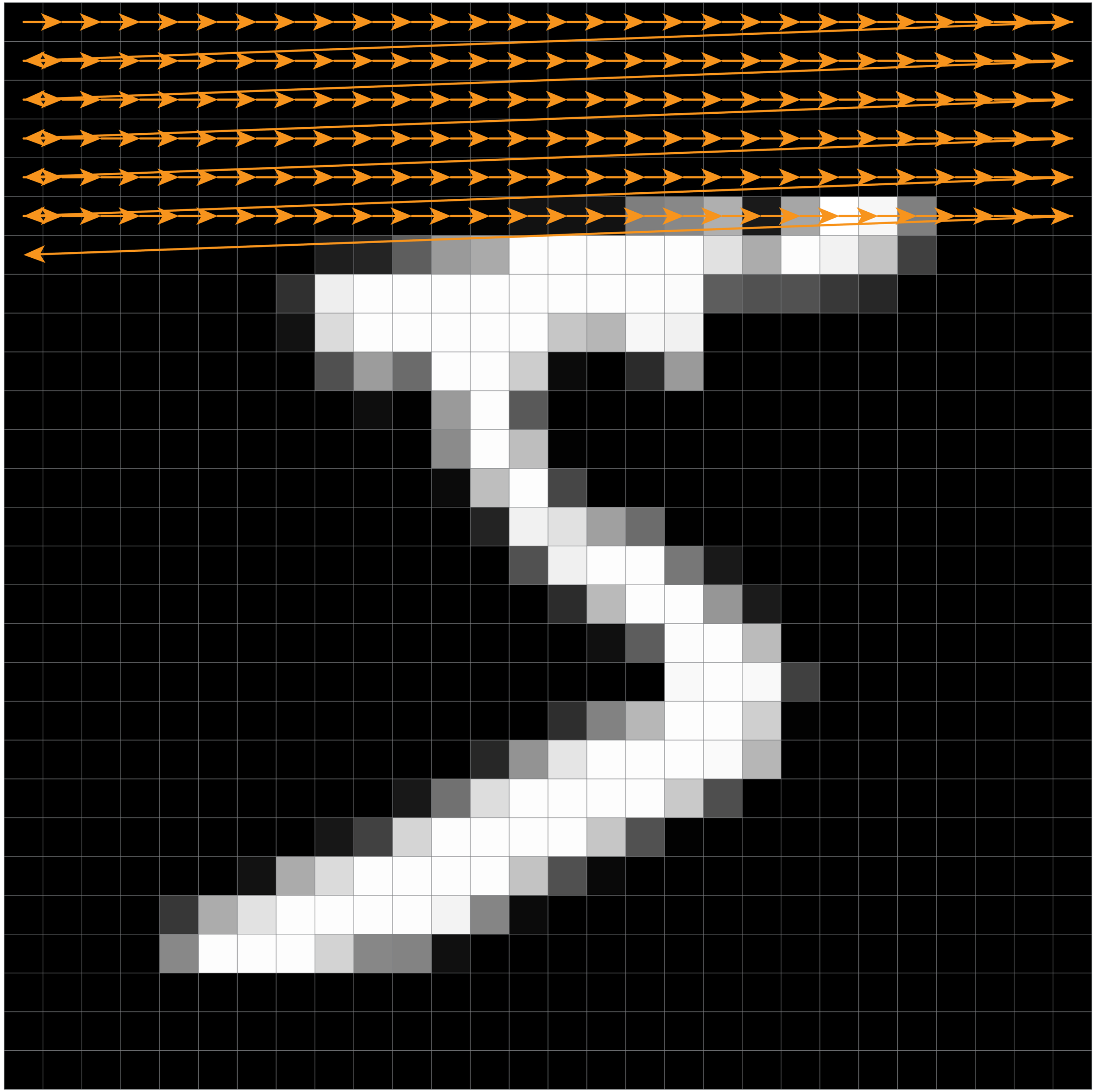
Running example: AR on MNIST

- an image as a sequence of pixels



Running example: AR on MNIST

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

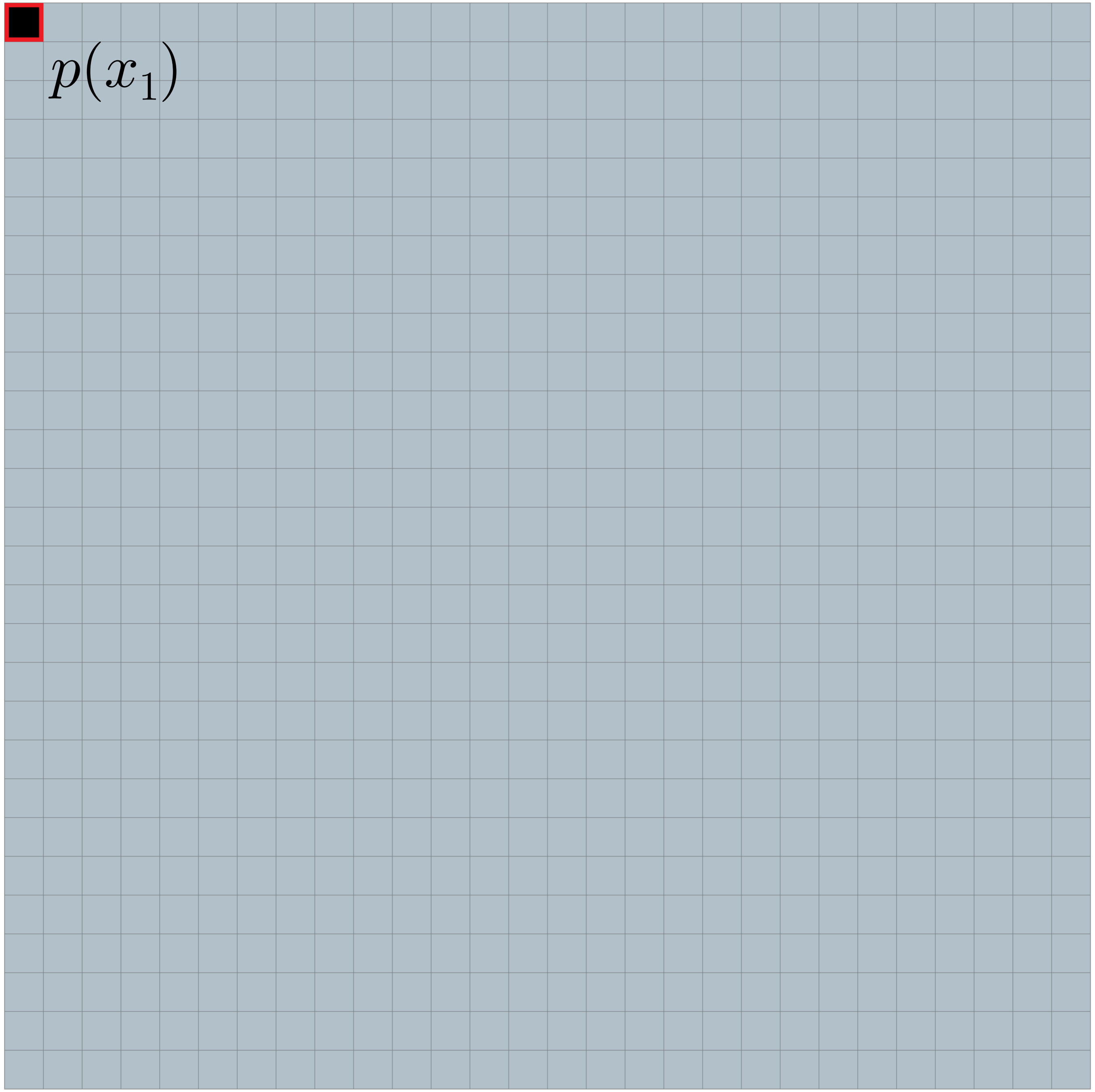


Running example:

AR on MNIST

Inference: Autoregressive

- sample this pixel from $p(x_1)$

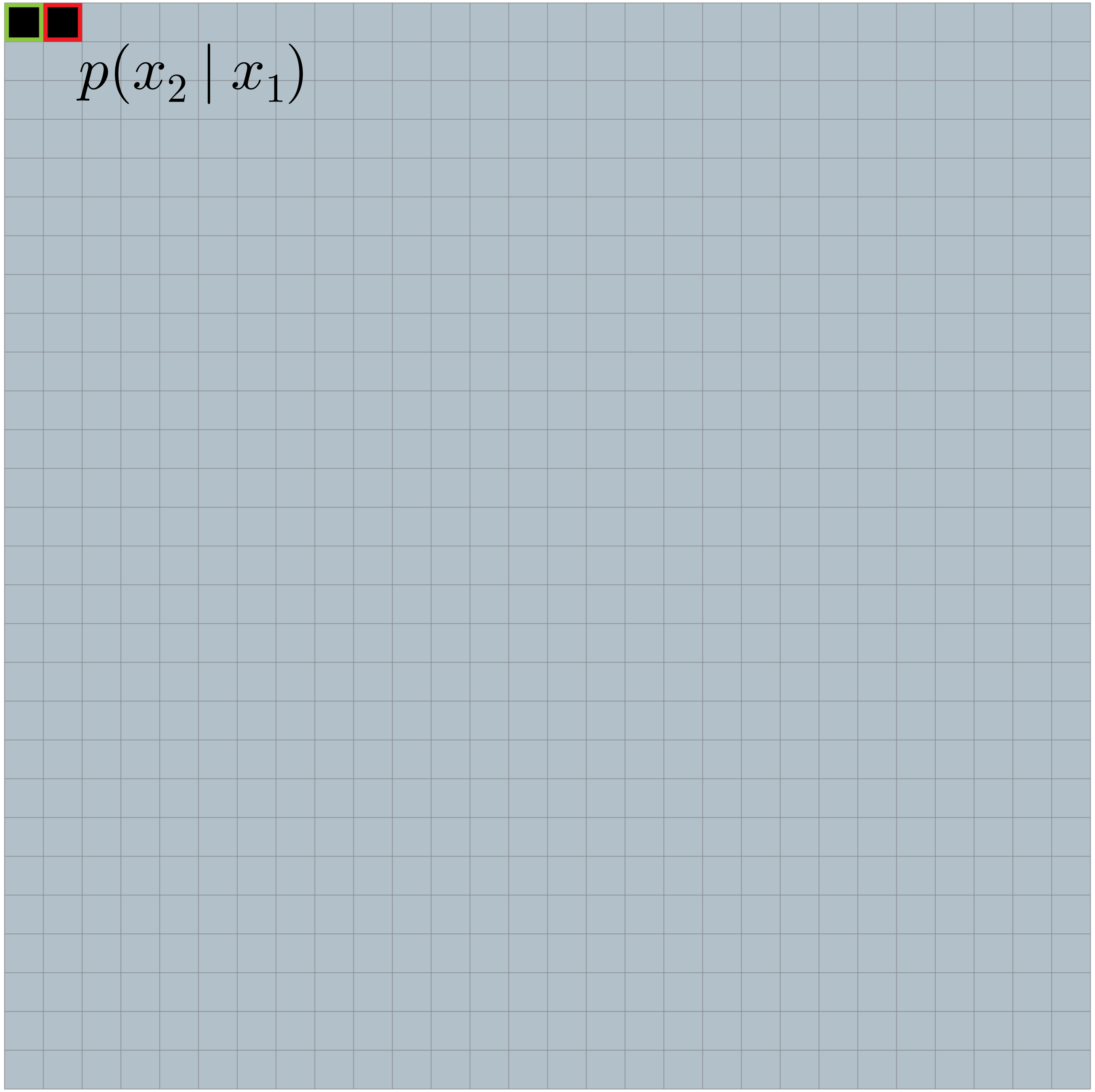


Running example:

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

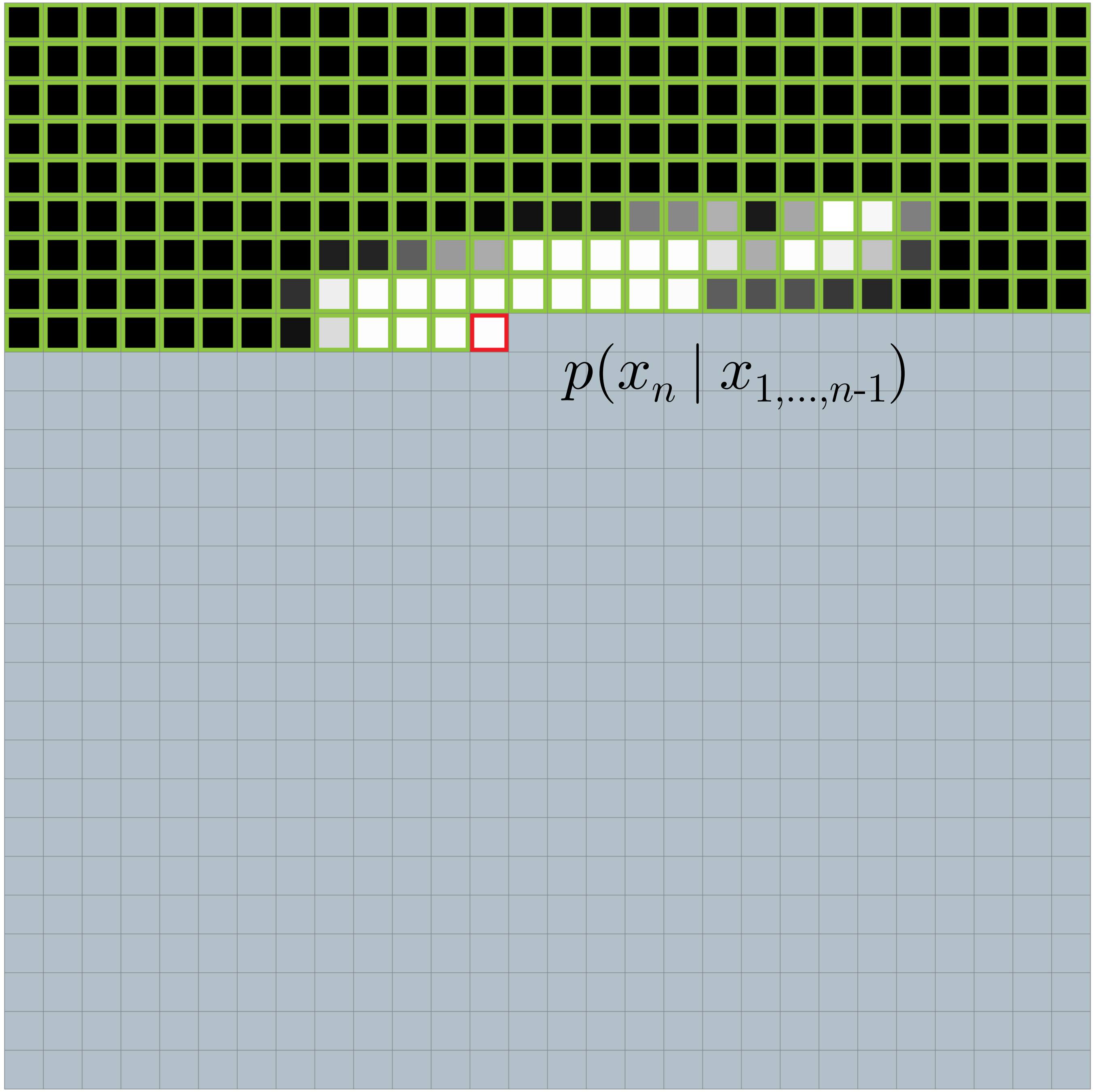


Running example:

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

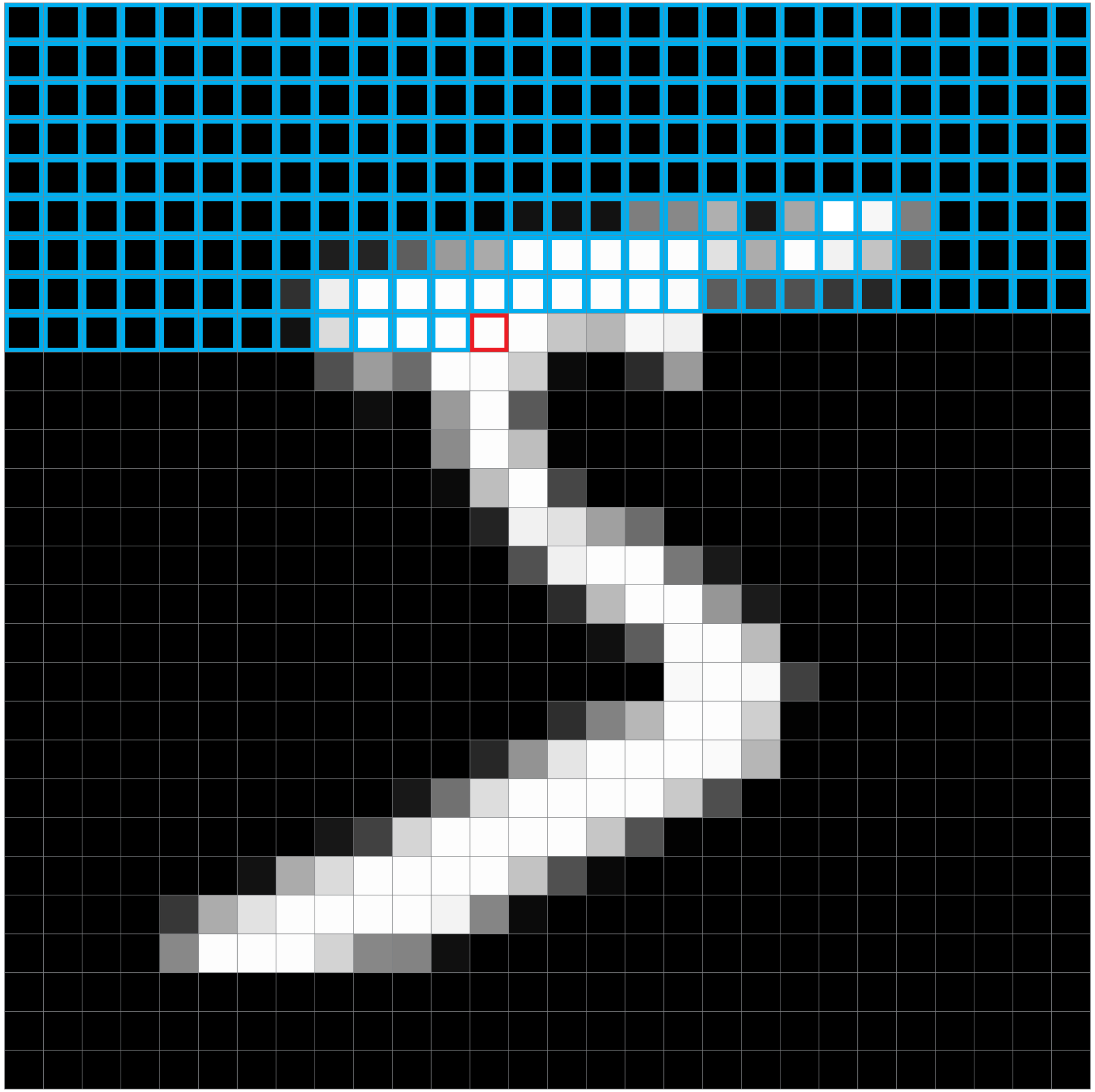


Running example:

AR on MNIST

Training: Teacher-Forcing

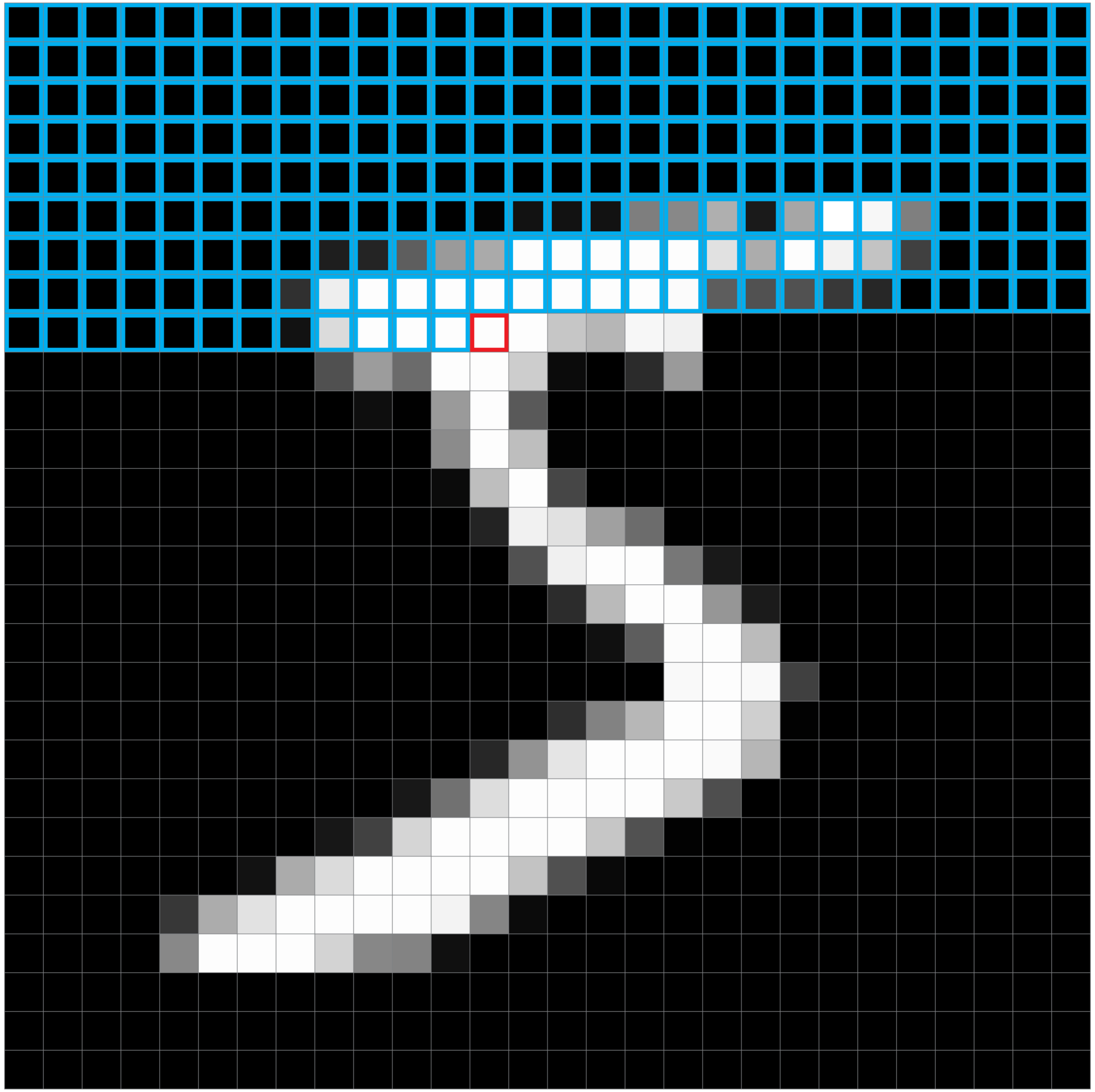
- model this pixel by: $p(x_n \mid x_{1,\ldots,n\text{-}1})$
- these are outputs from ground-truth, inputs for current step
- network for this step:
  - $(n$ - $1)$ inputs
  - 1 predict



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

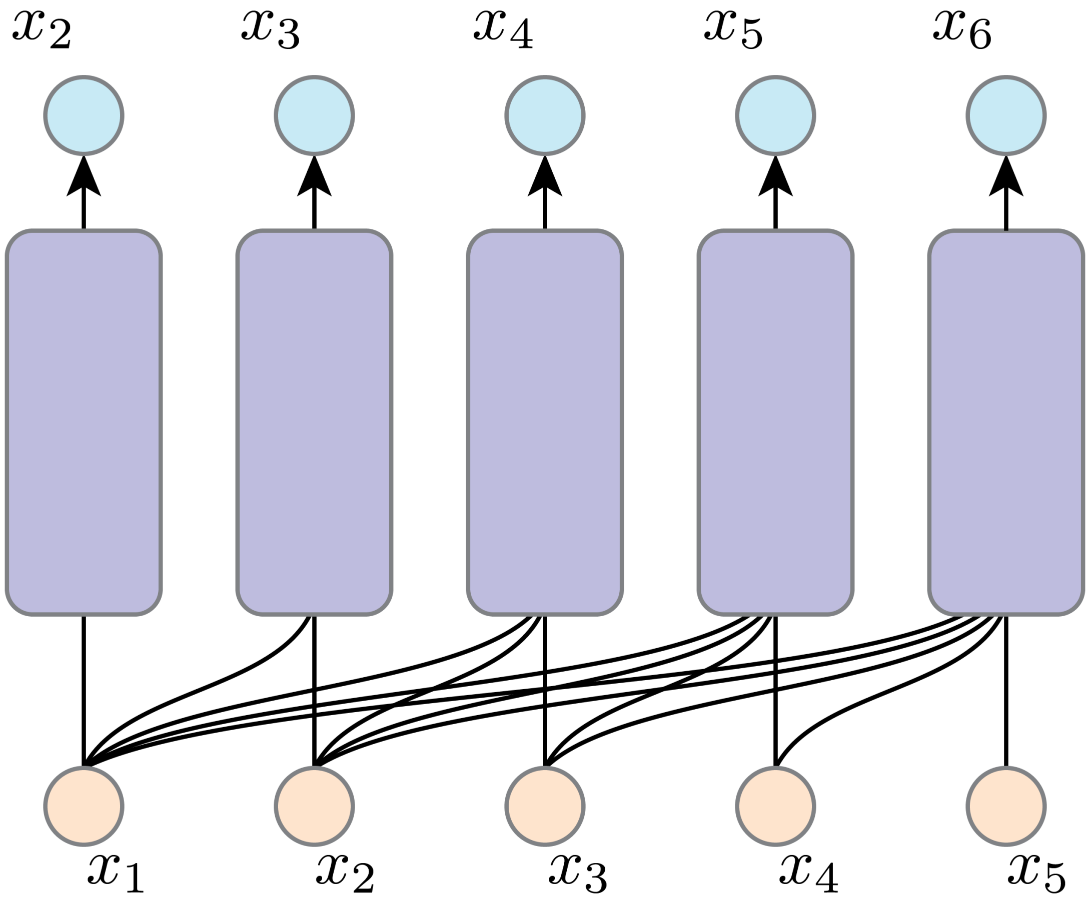
These are <u>not</u> specific to a certain type of network architectures.

Network Architectures for Autoregressive Modeling

Autoregression is not architecture-specific

This figure implements this formulation:

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



(showing training case for simplicity)

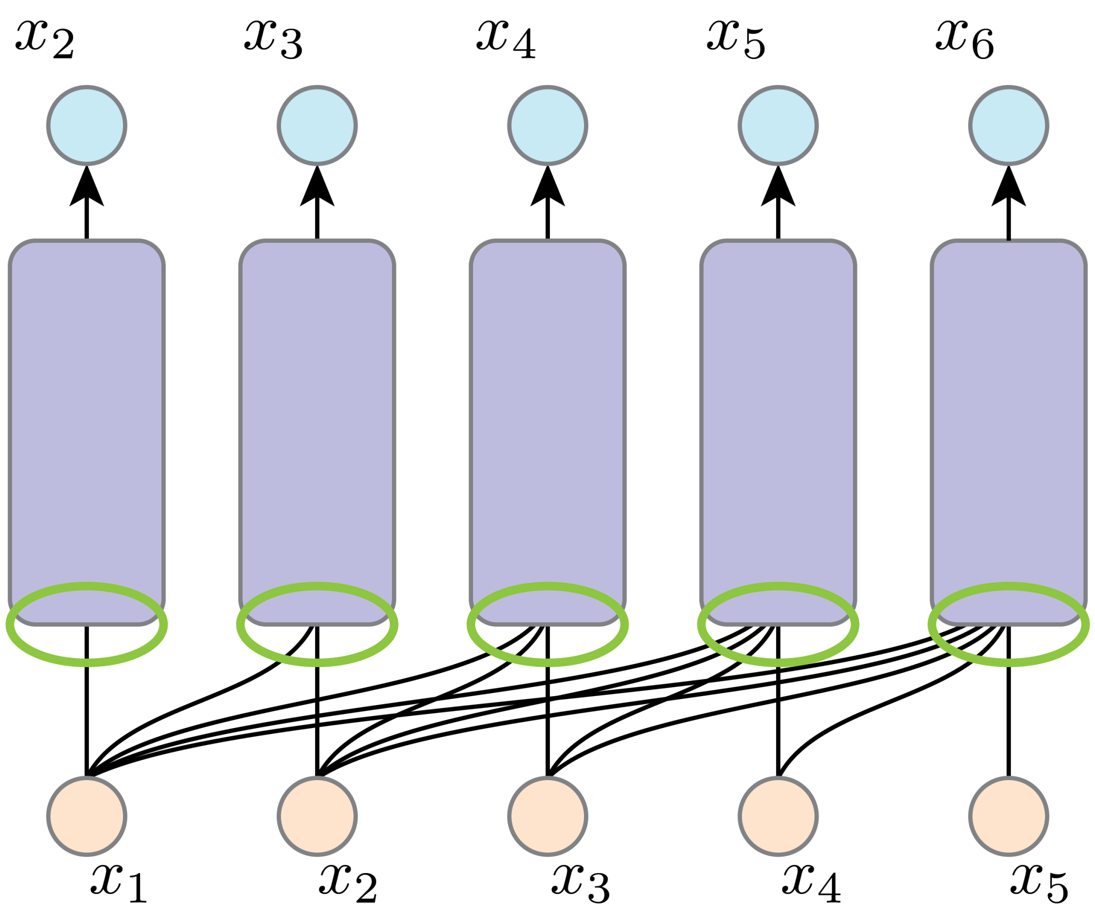
Autoregression is not architecture-specific

This figure implements this formulation:

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

In this example:

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

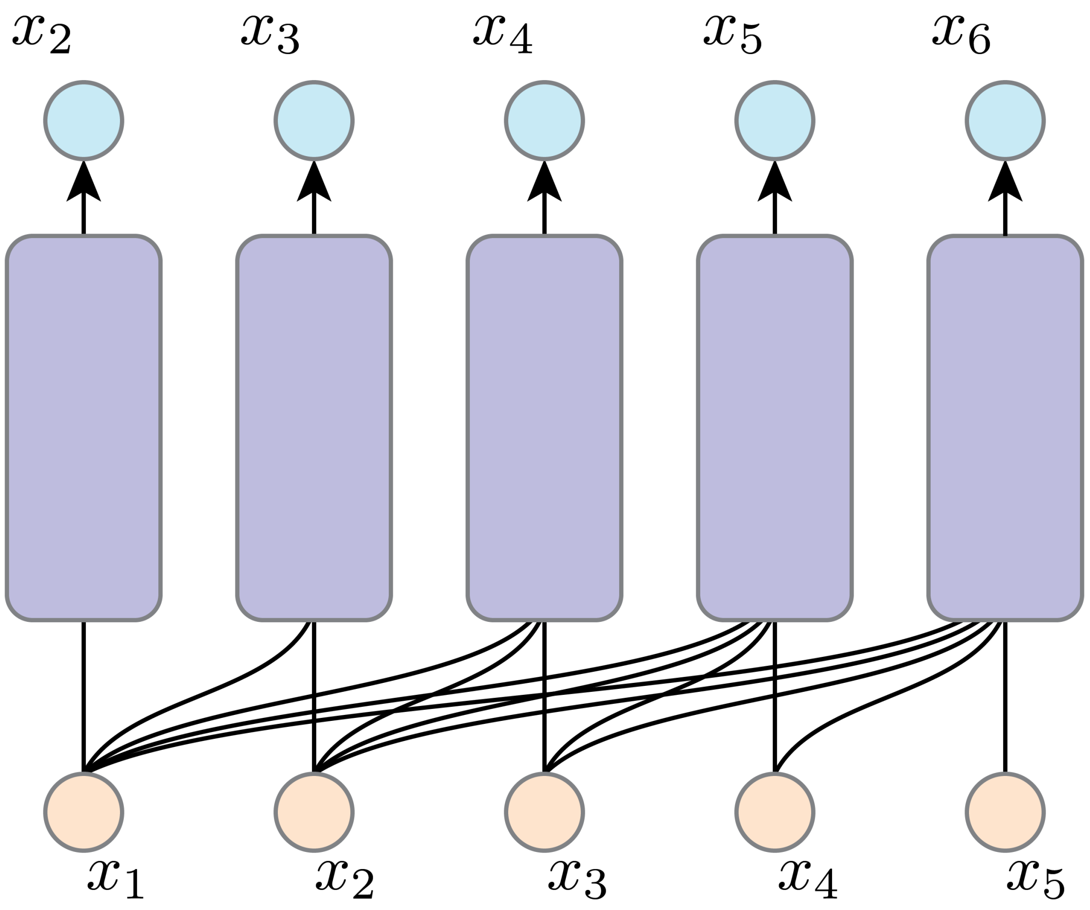


Autoregression is not architecture-specific

This figure implements this formulation:

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

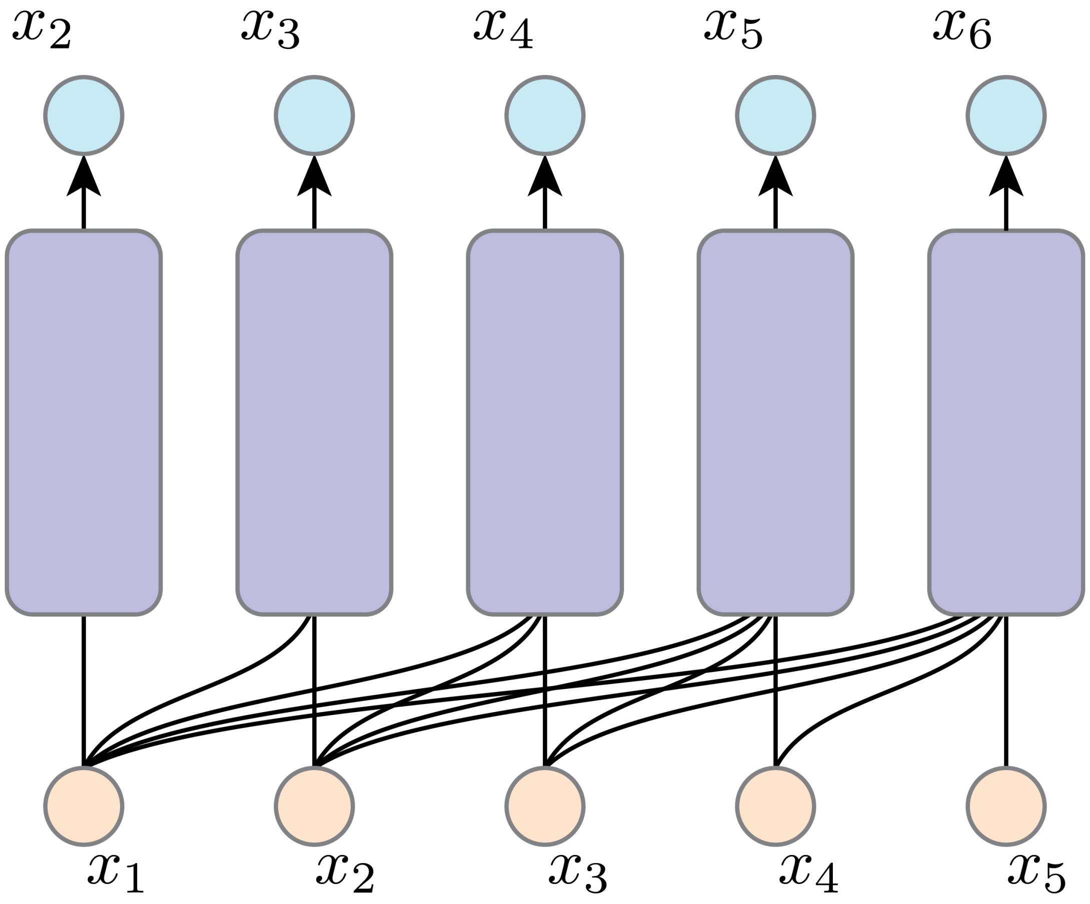
Can we do this efficiently?



Autoregression w/ Shared Computation

This figure implements this formulation:

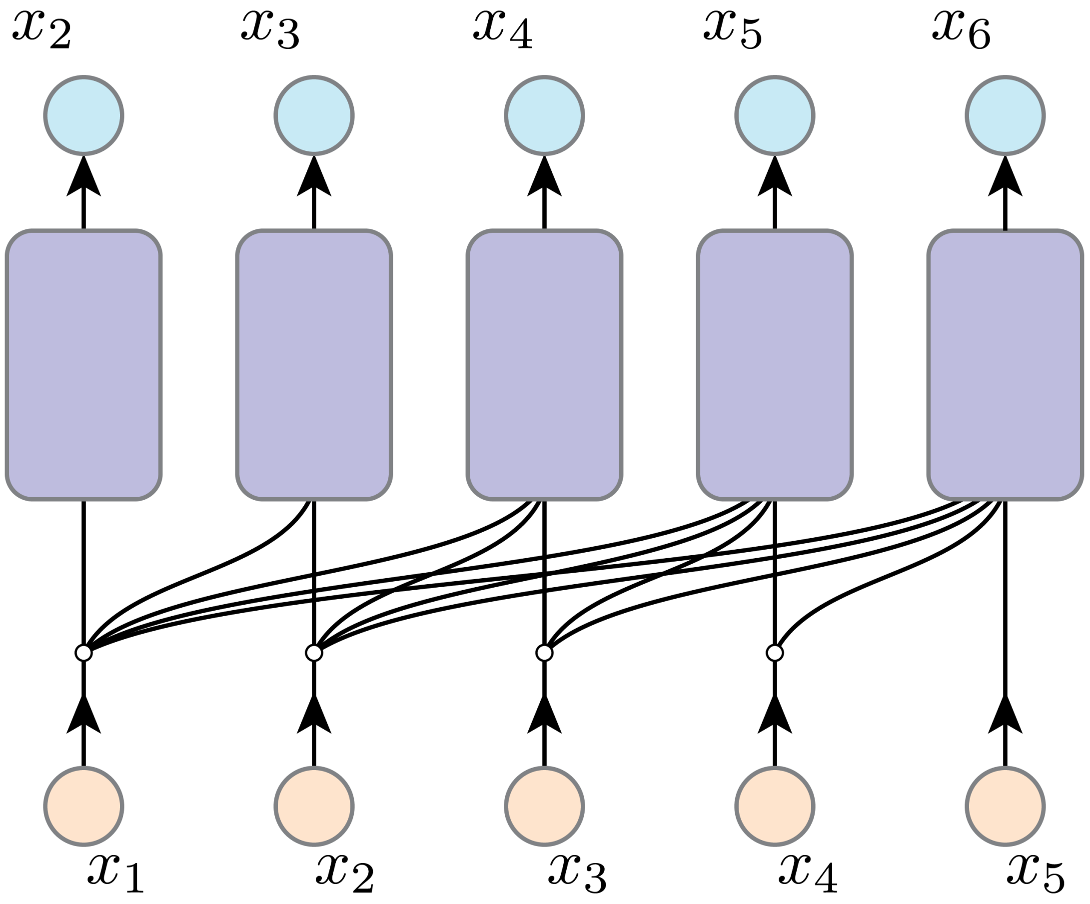
$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$



Autoregression w/ Shared Computation

This figure implements this formulation:

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

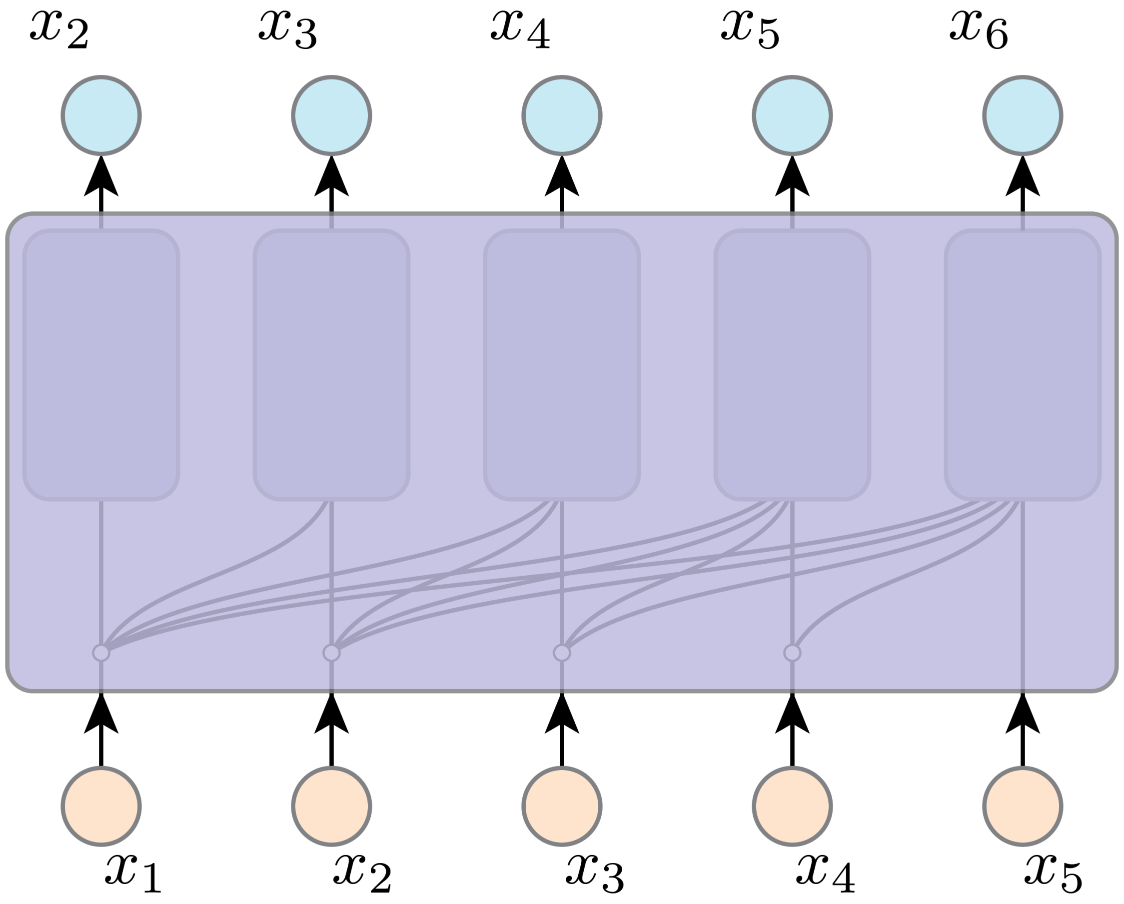


(this figure is equivalent to previous one)

Autoregression w/ Shared Computation

This figure implements this formulation:

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



Autoregression w/ Shared Computation

We can implement:

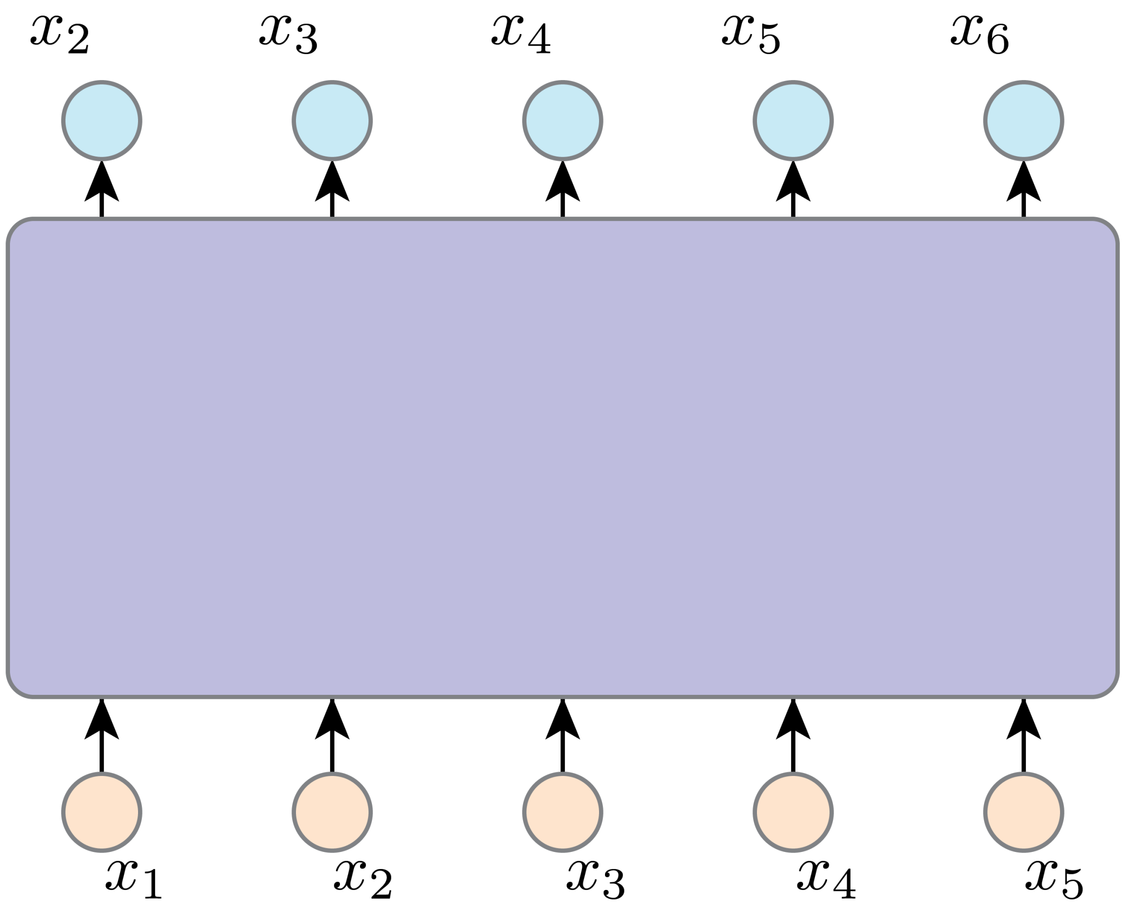
$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

... 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:

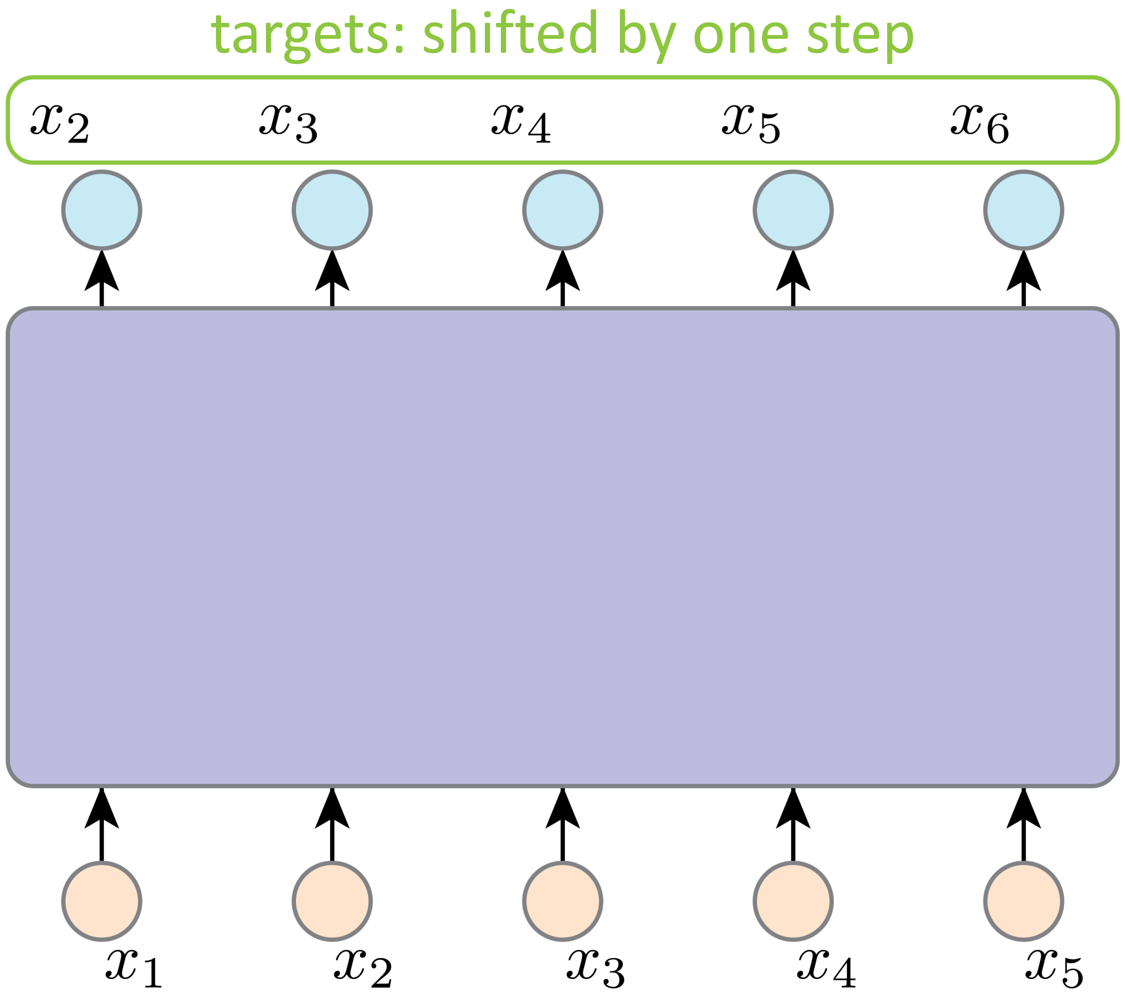
$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

... by one network, with:

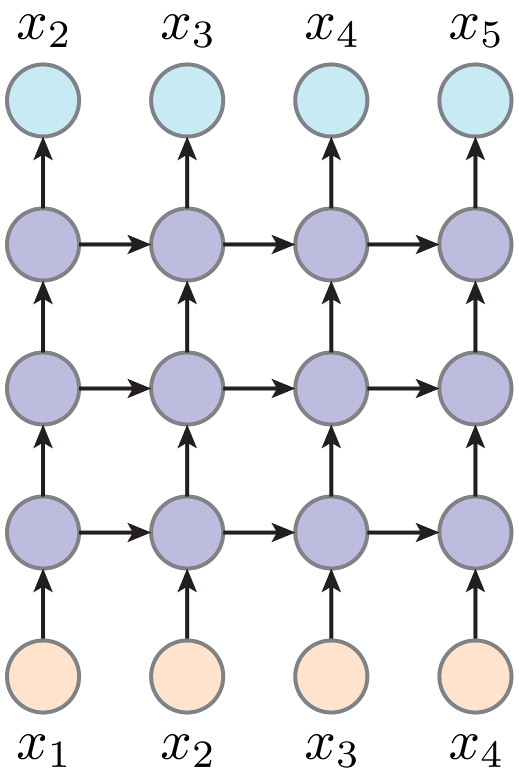
- shared architecture
- shared weights
- shared **computation**

if:

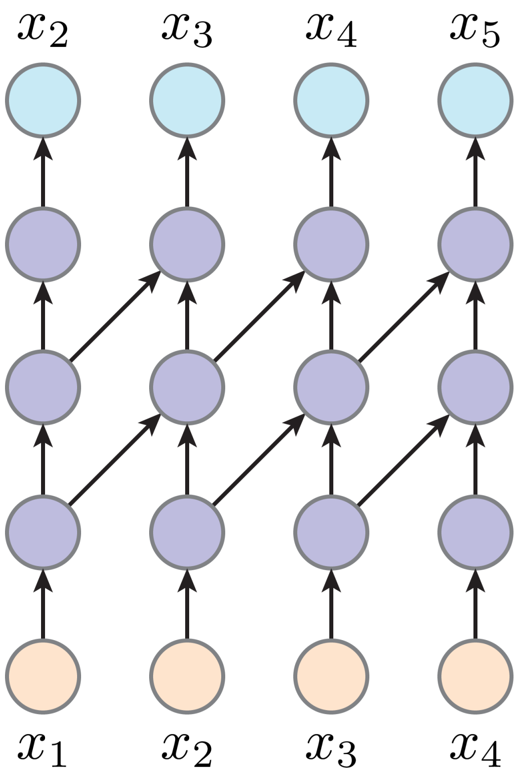
- output x_i **not** depend on x_j for any $j \geq i$



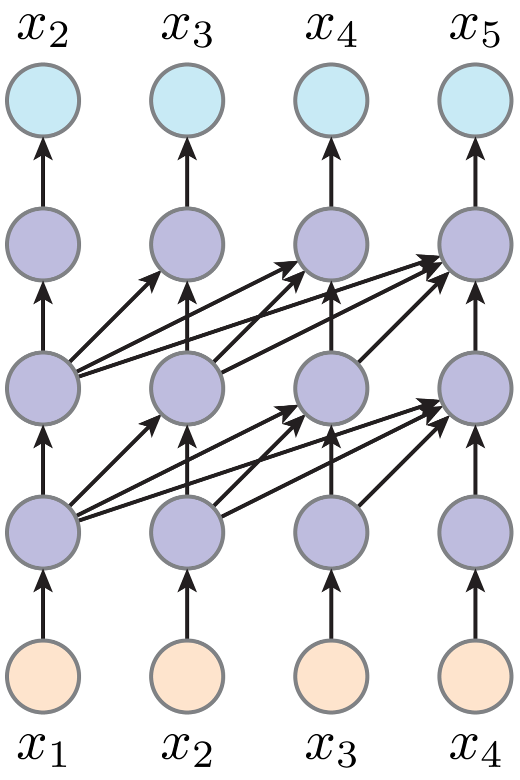
Common Architectures for Autoregression



RNN



CNN

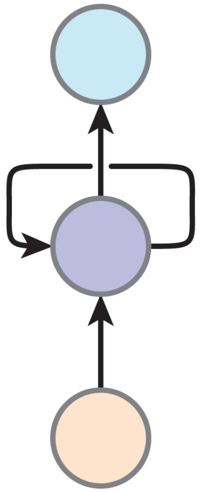


Attention

See also my lecture at 6.8300: "11: Sequence Modeling": https://drive.google.com/file/d/1lOYsyImXl3caWlsplfThyuvgz-8Vn7yR/view?usp=sharing

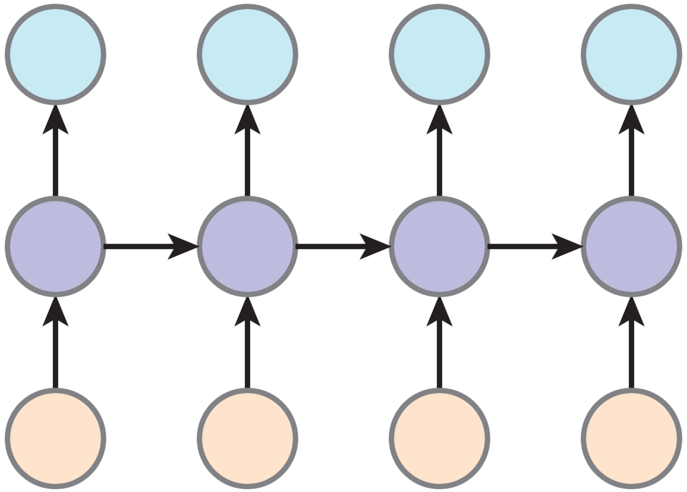
Brief: Recurrent Neural Network (RNN) for AR

one RNN unit



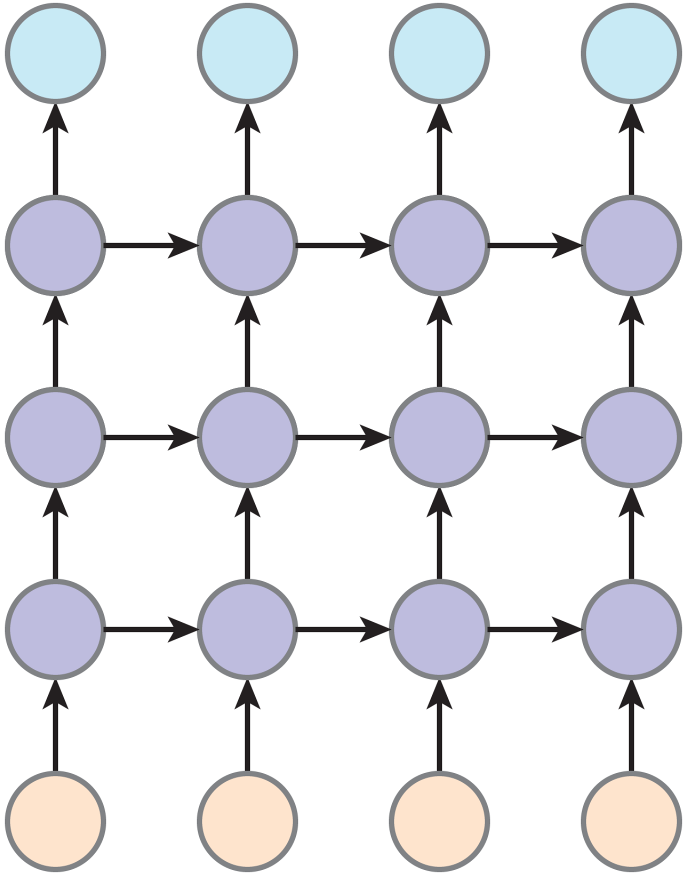
Brief: Recurrent Neural Network (RNN) for AR

unfold in “time”



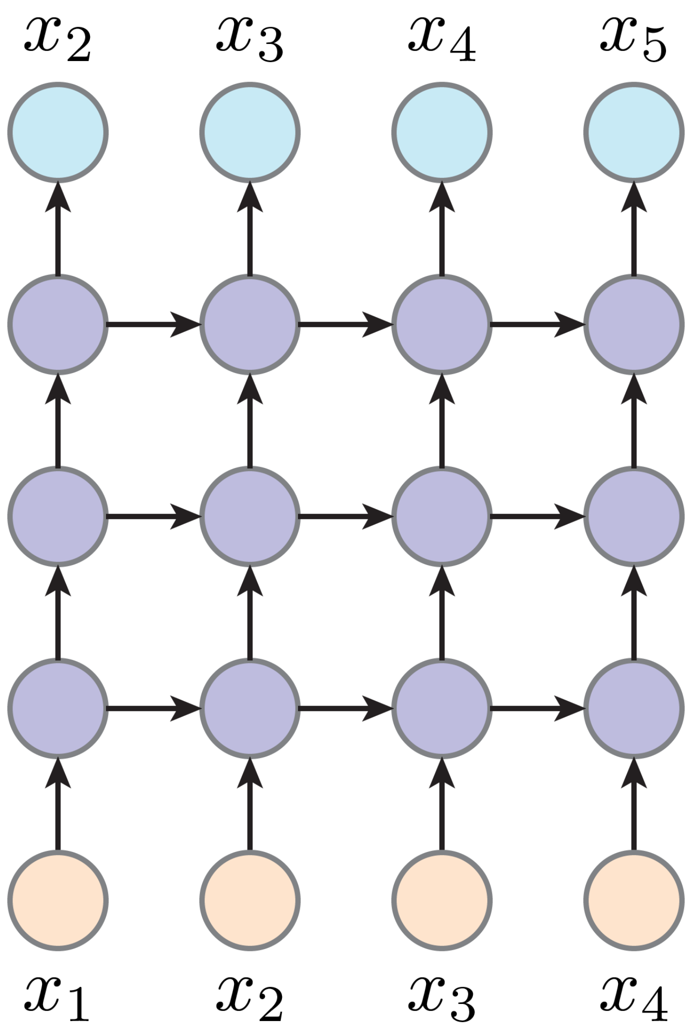
Brief: Recurrent Neural Network (RNN) for AR

go deep

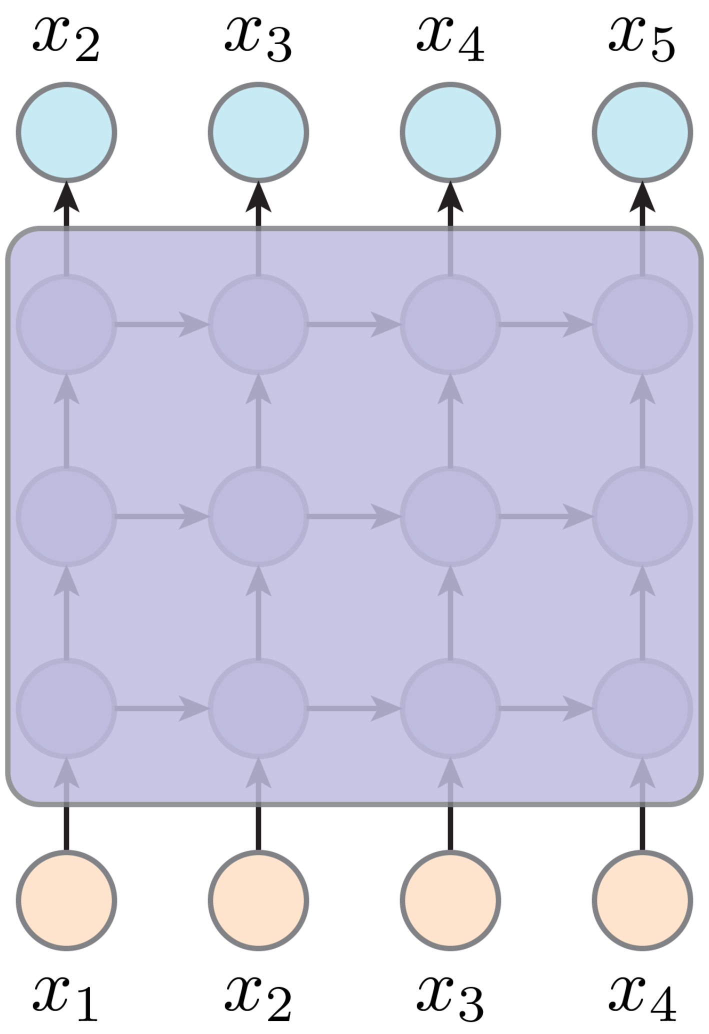


Brief: Recurrent Neural Network (RNN) for AR

shift target by one step

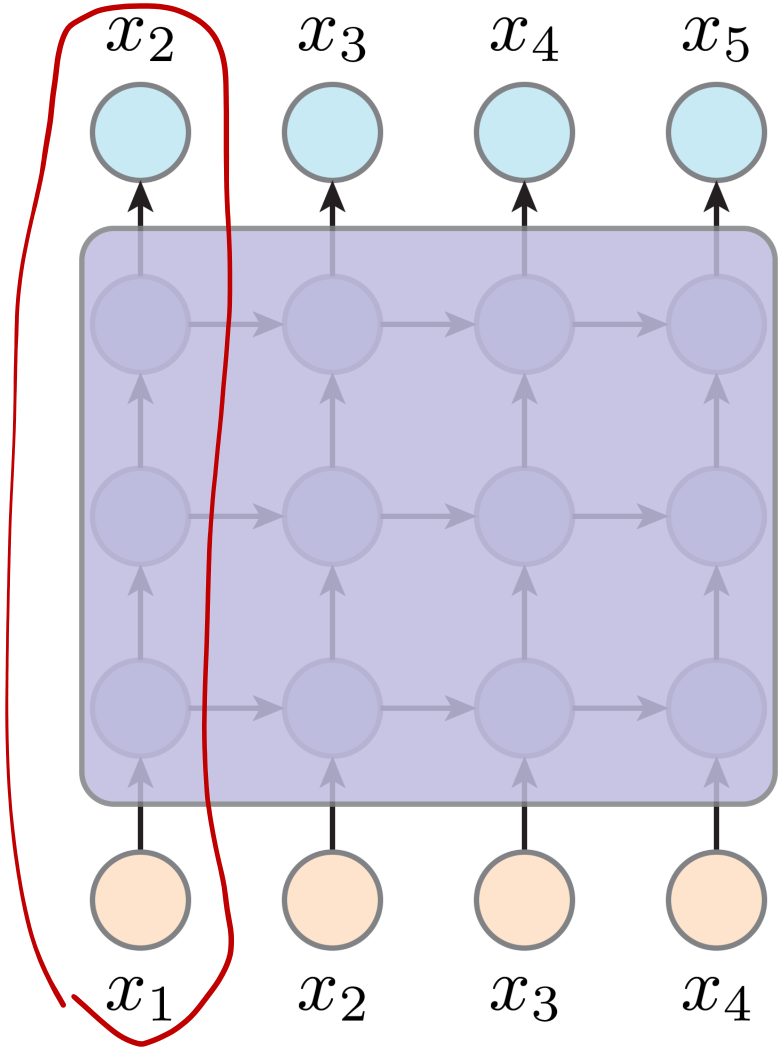


Brief: Recurrent Neural Network (RNN) for AR



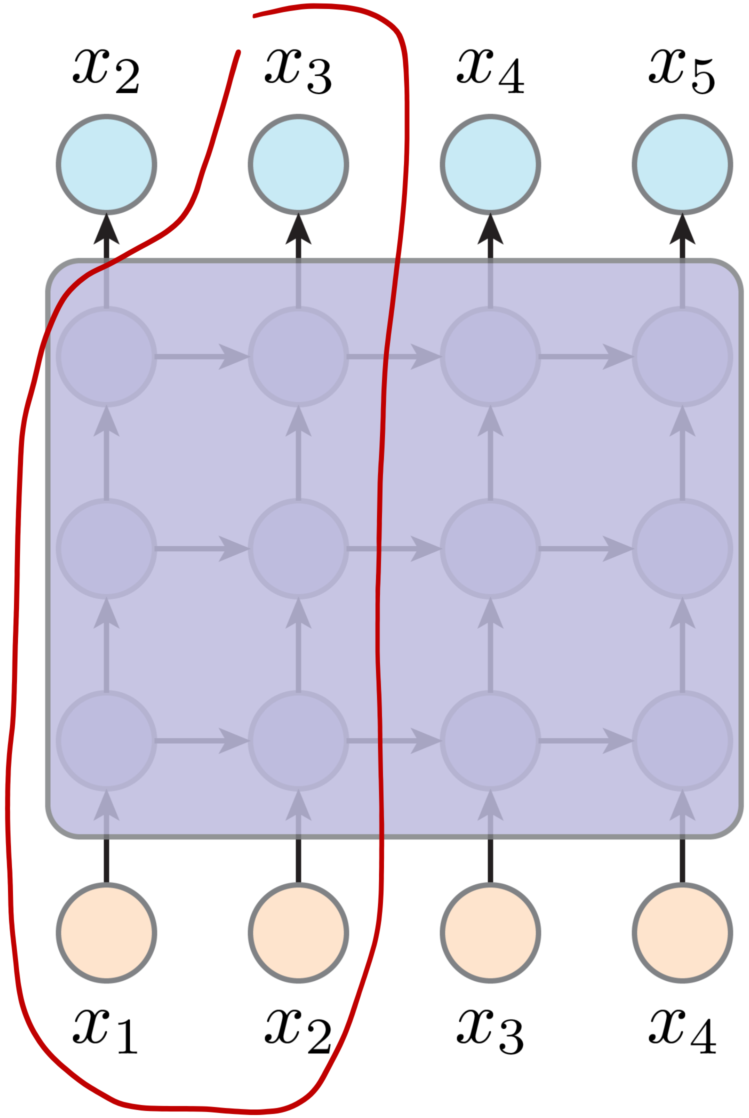
Brief: Recurrent Neural Network (RNN) for AR

$p(x_2 \mid x_1)$

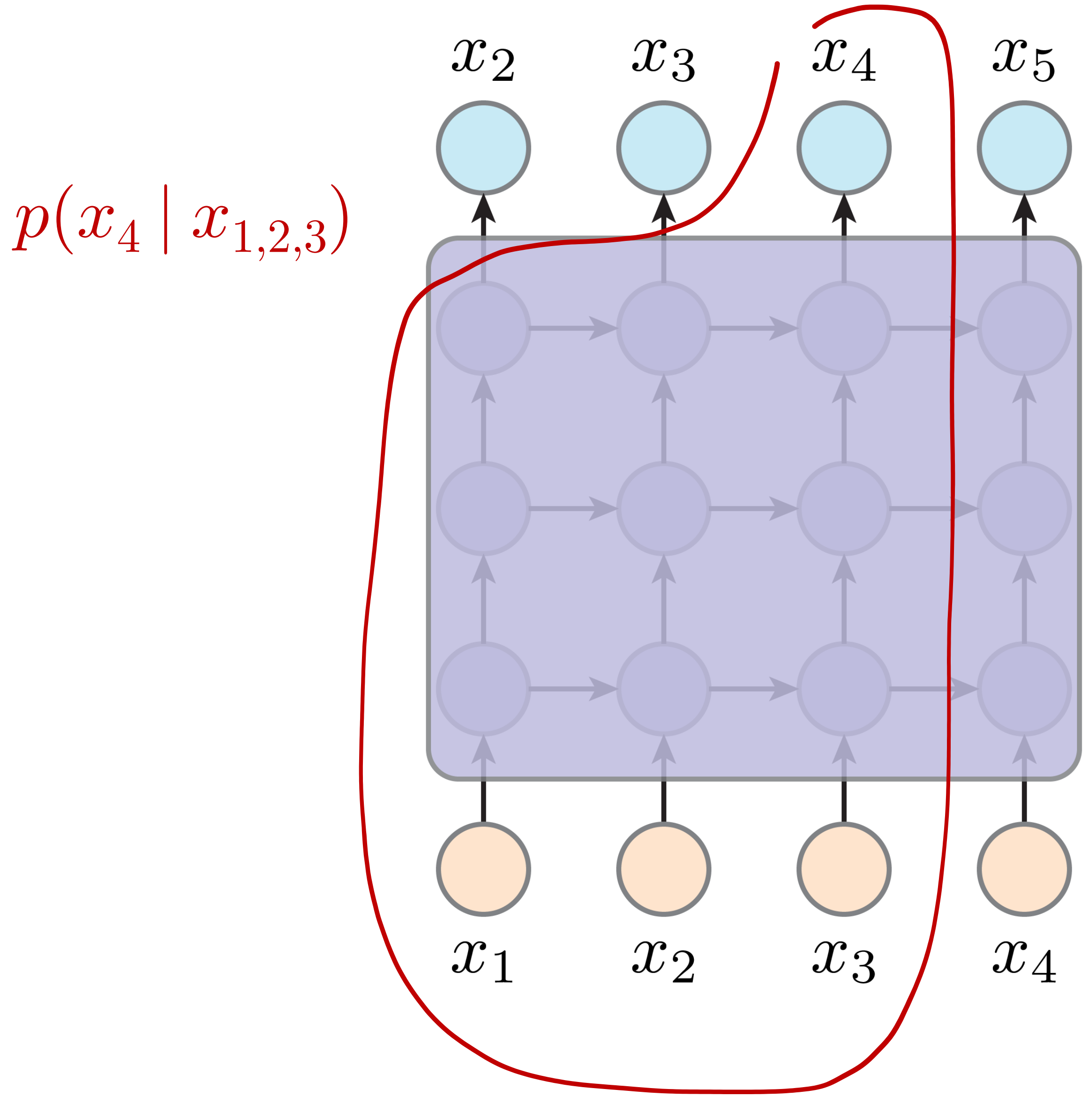


Brief: Recurrent Neural Network (RNN) for AR

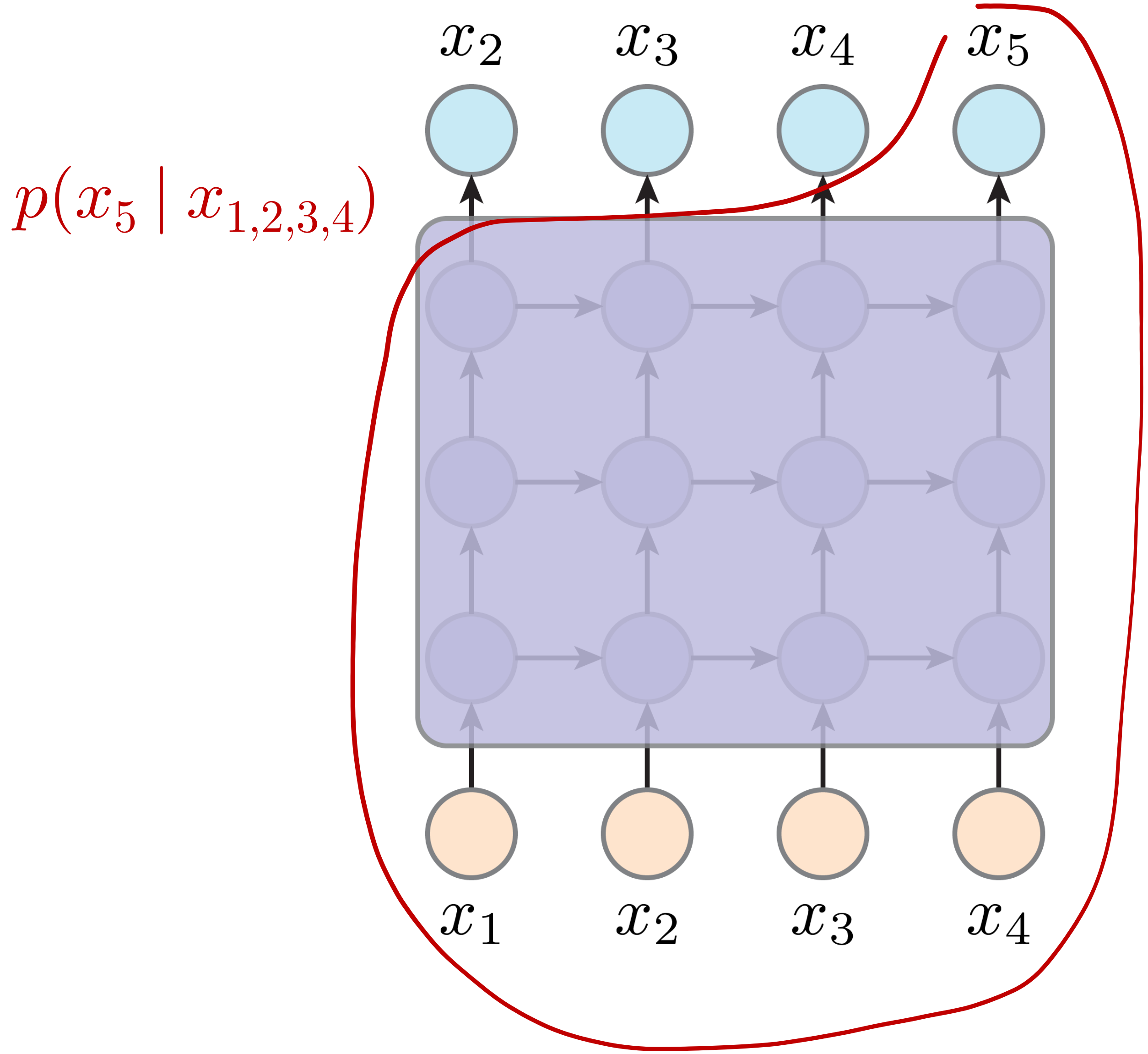
$p(x_3 \mid x_{1,2})$



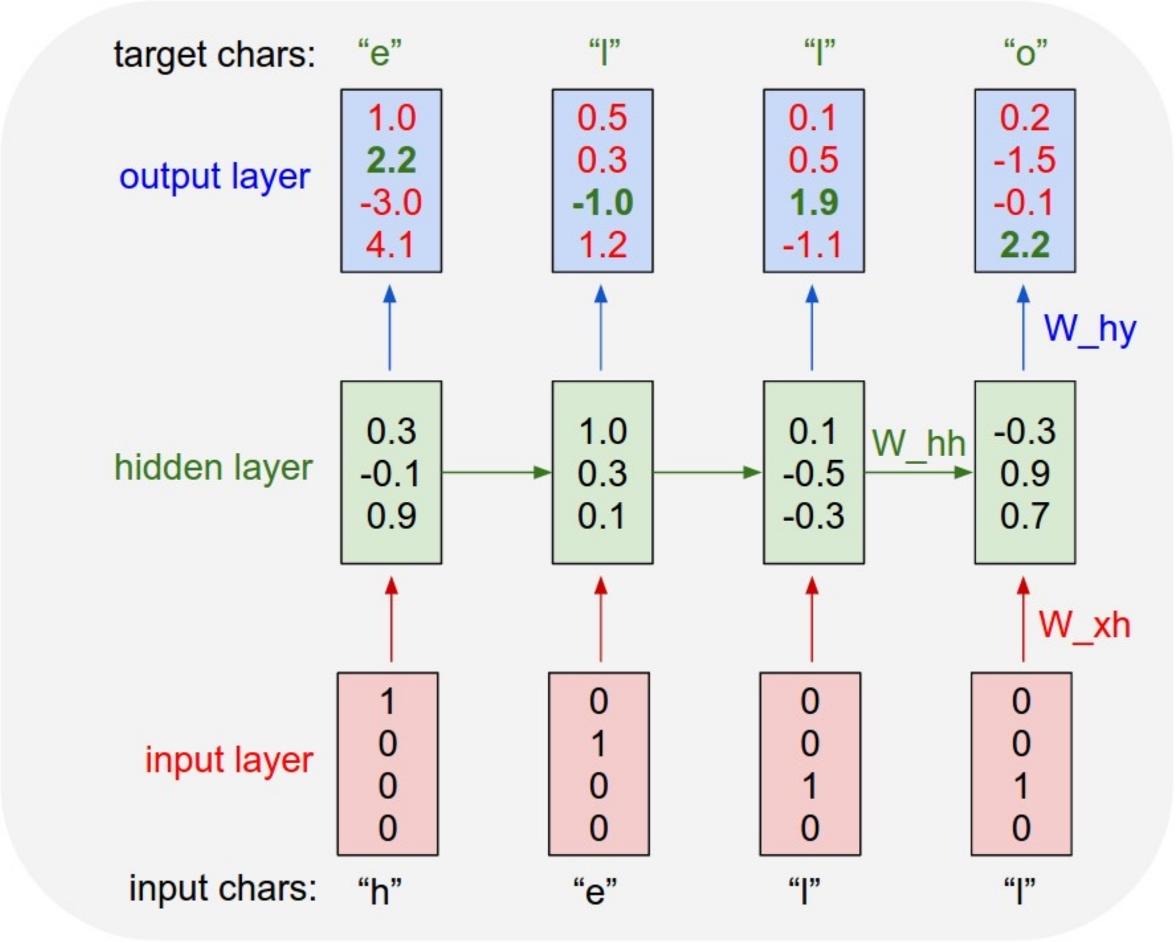
Brief: Recurrent Neural Network (RNN) for AR



Brief: Recurrent Neural Network (RNN) for AR



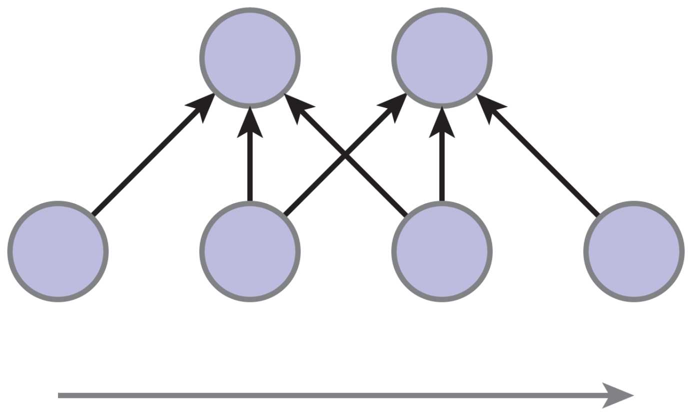
Example: Char-RNN



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

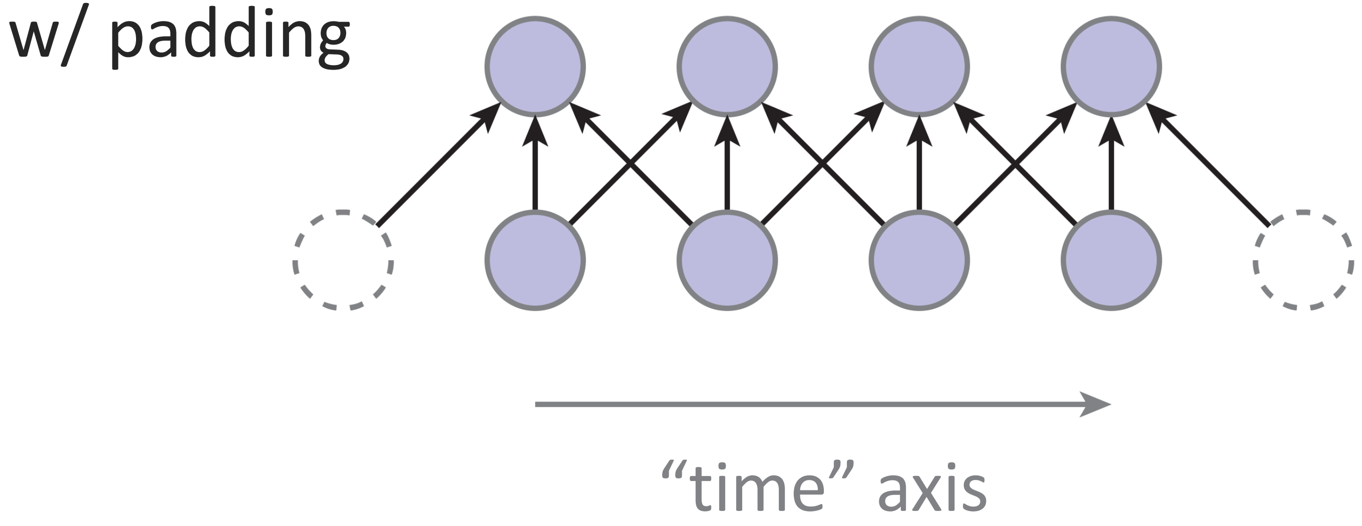
Brief: Convolutional Neural Network (CNN) for AR

1-D convolution



“time” axis

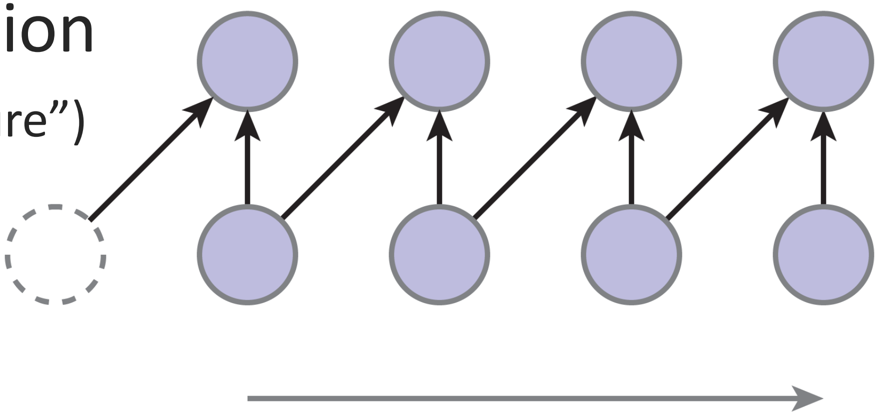
Brief: Convolutional Neural Network (CNN) for AR



Brief: Convolutional Neural Network (CNN) for AR

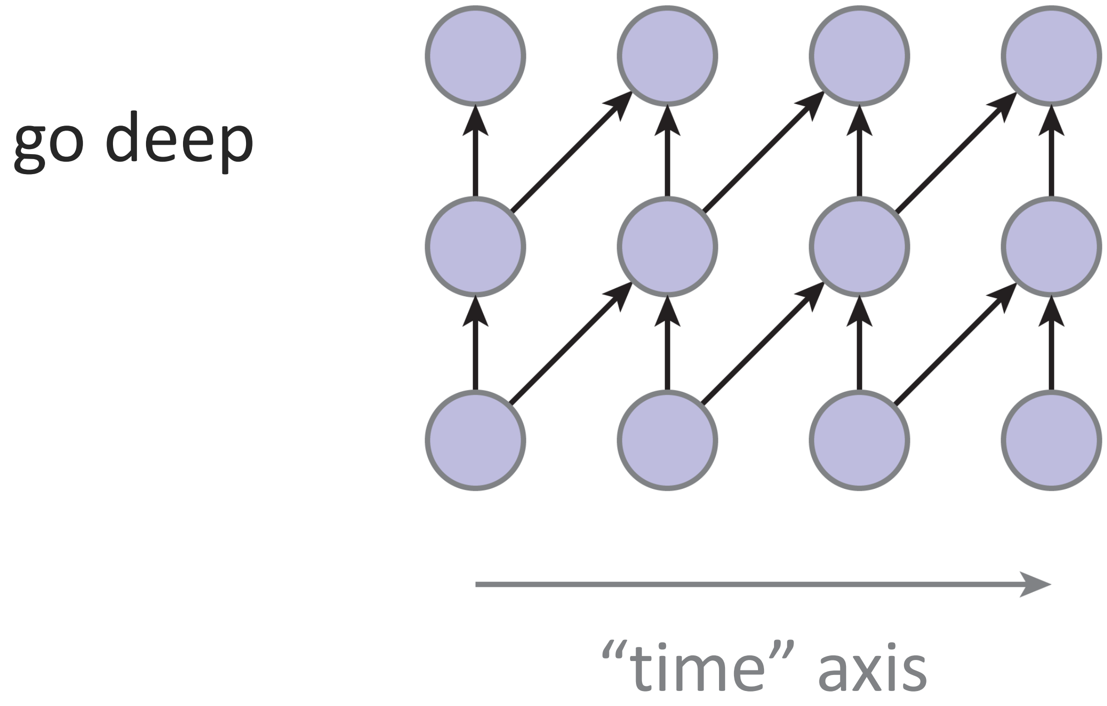
causal convolution

(not depend on “future”)



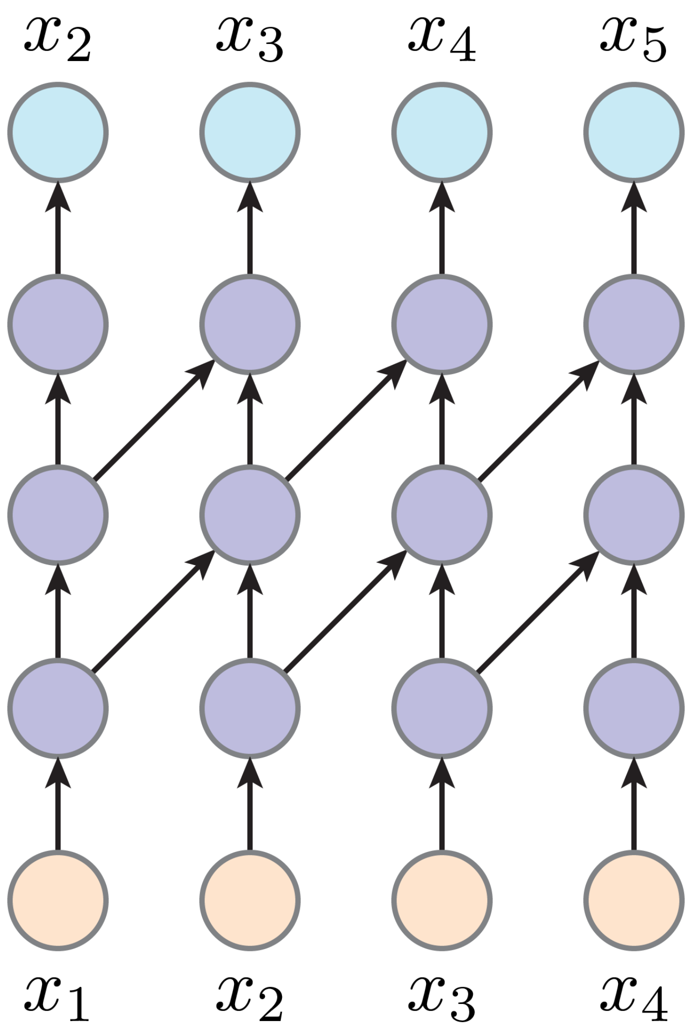
“time” axis

Brief: Convolutional Neural Network (CNN) for AR

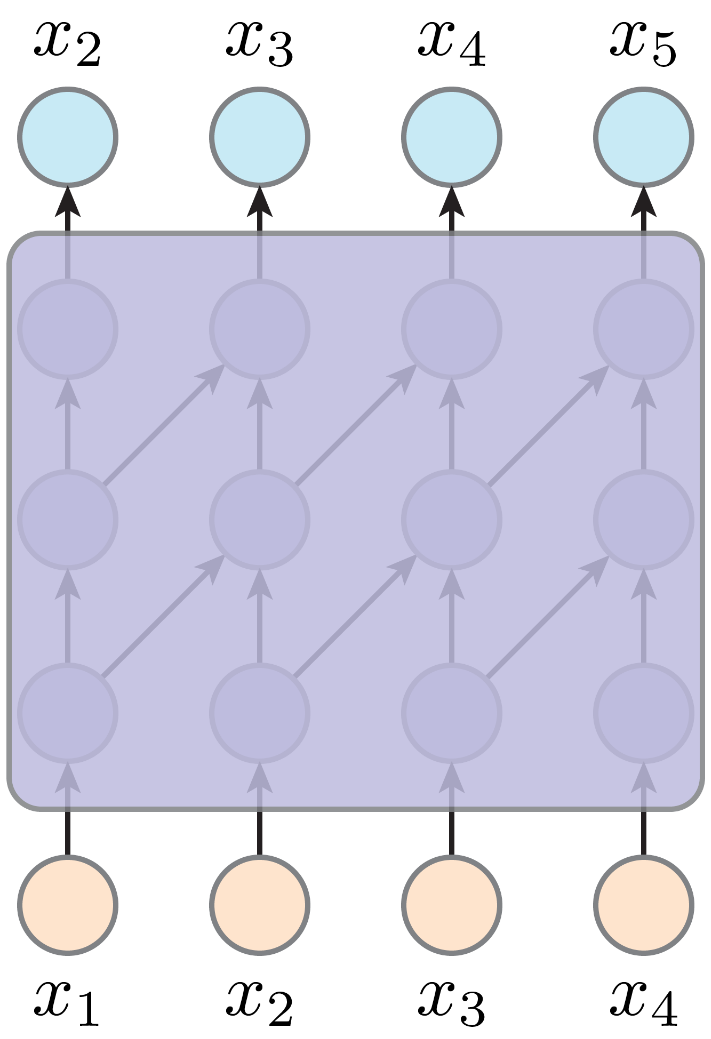


Brief: Convolutional Neural Network (CNN) for AR

shift target by one step

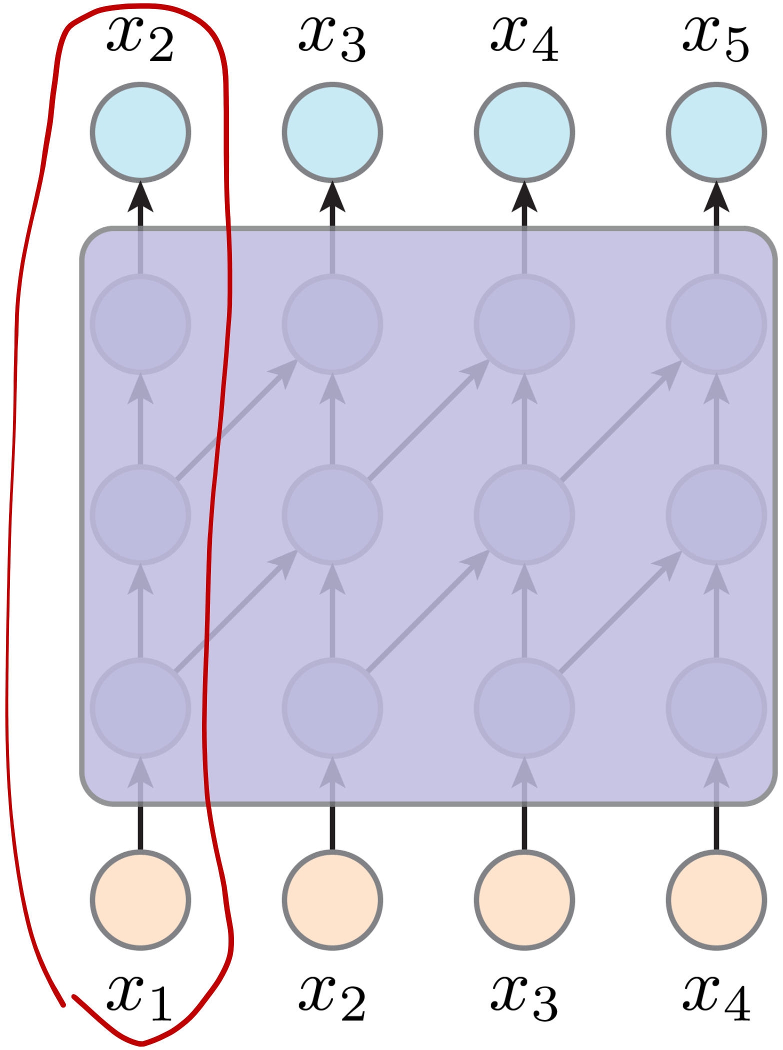


Brief: Convolutional Neural Network (CNN) for AR

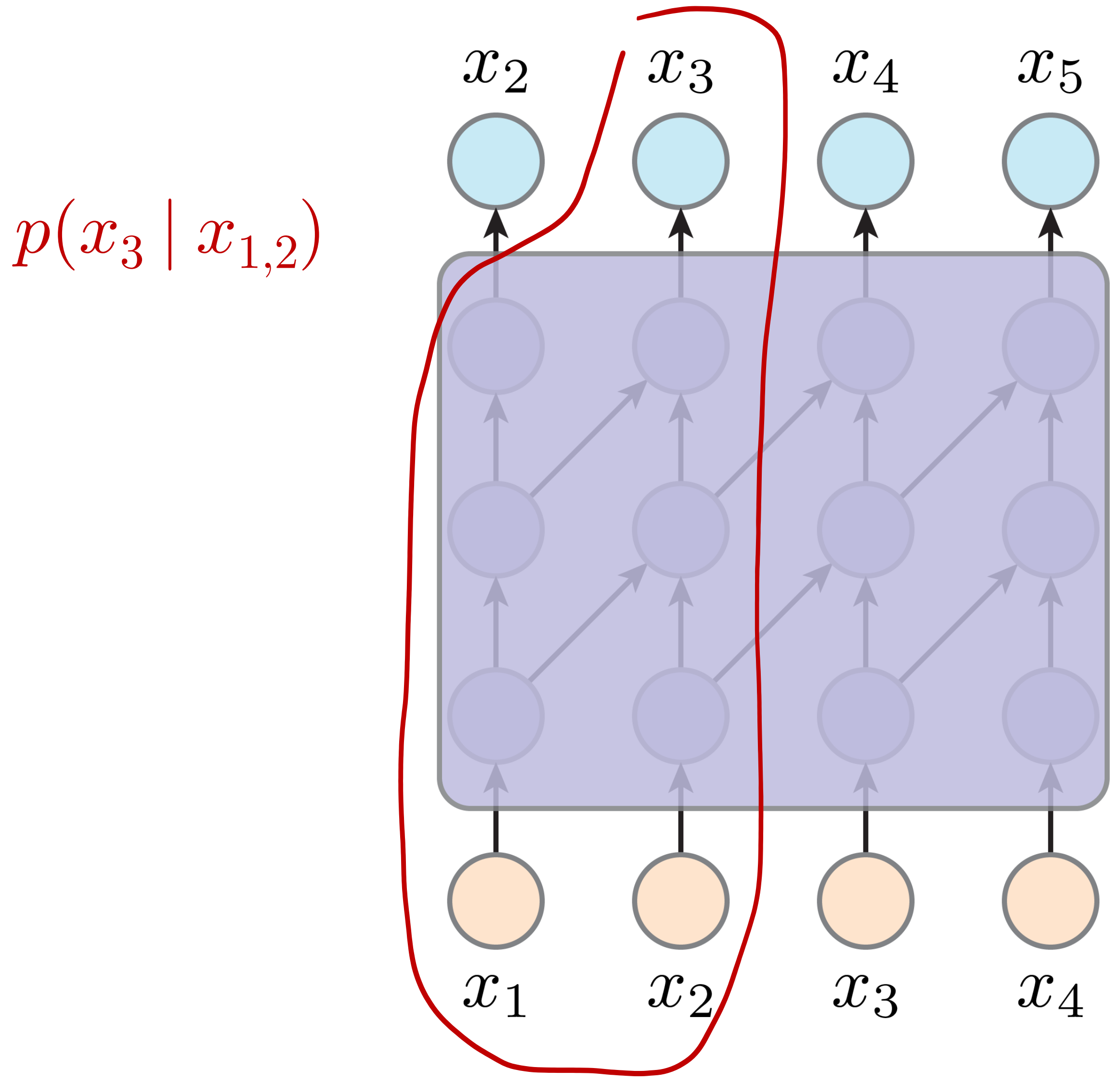


Brief: Convolutional Neural Network (CNN) for AR

$p(x_2 \mid x_1)$

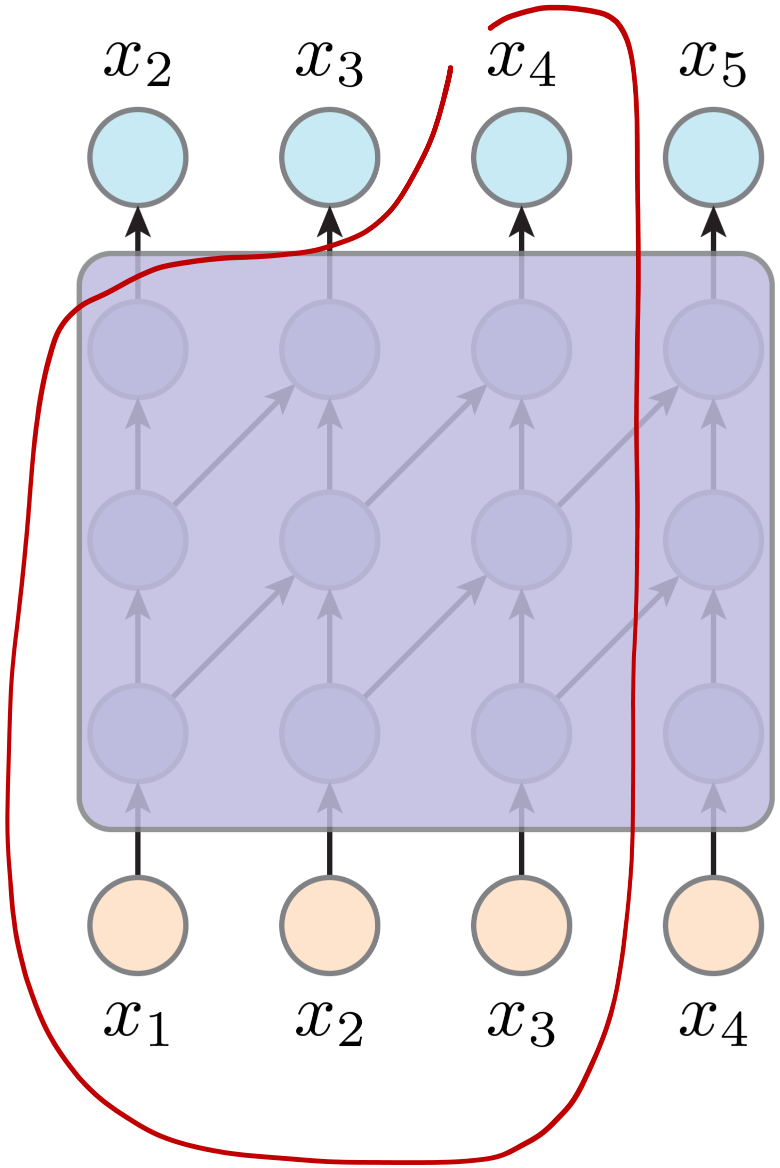


Brief: Convolutional Neural Network (CNN) for AR

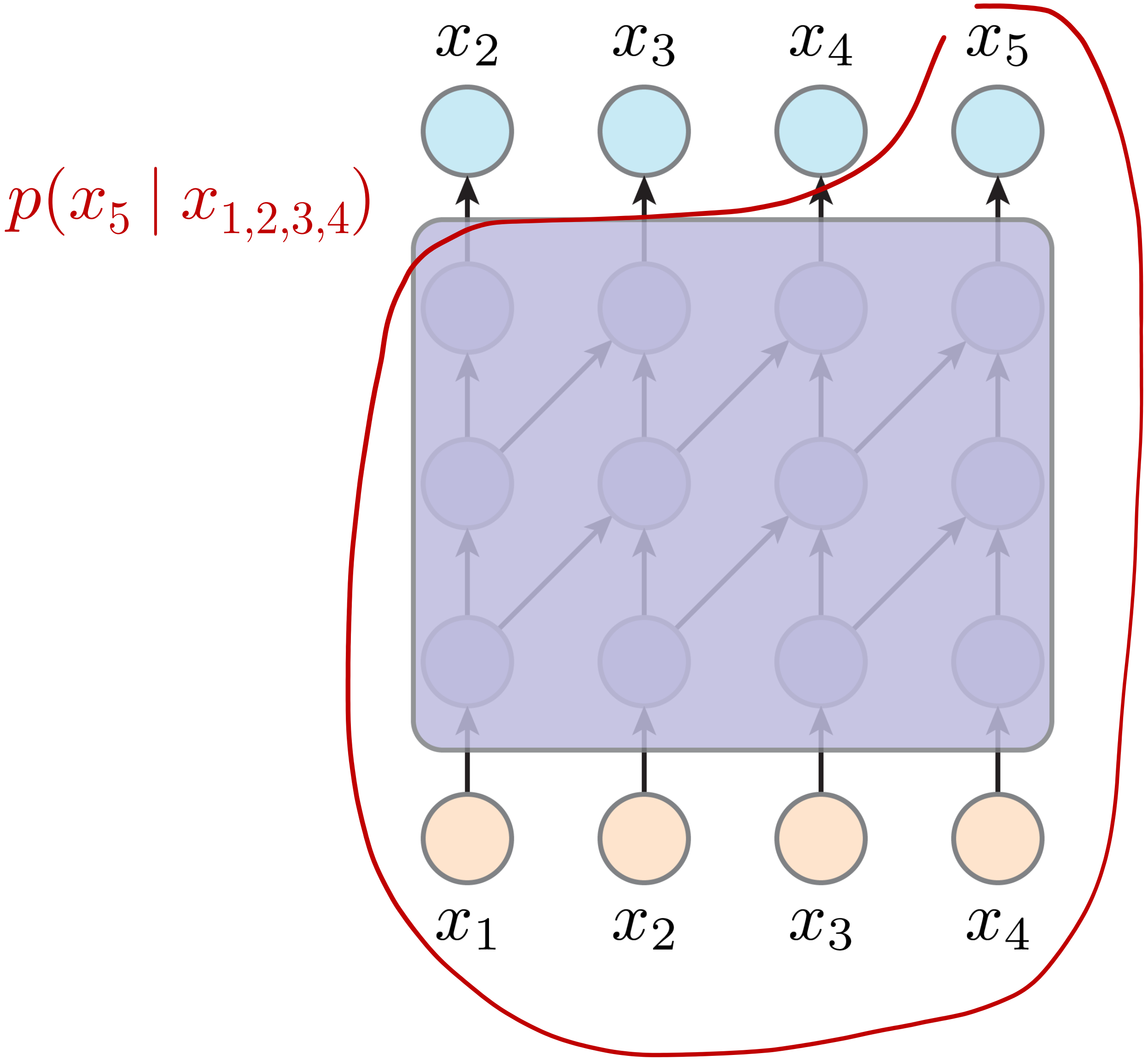


Brief: Convolutional Neural Network (CNN) for AR

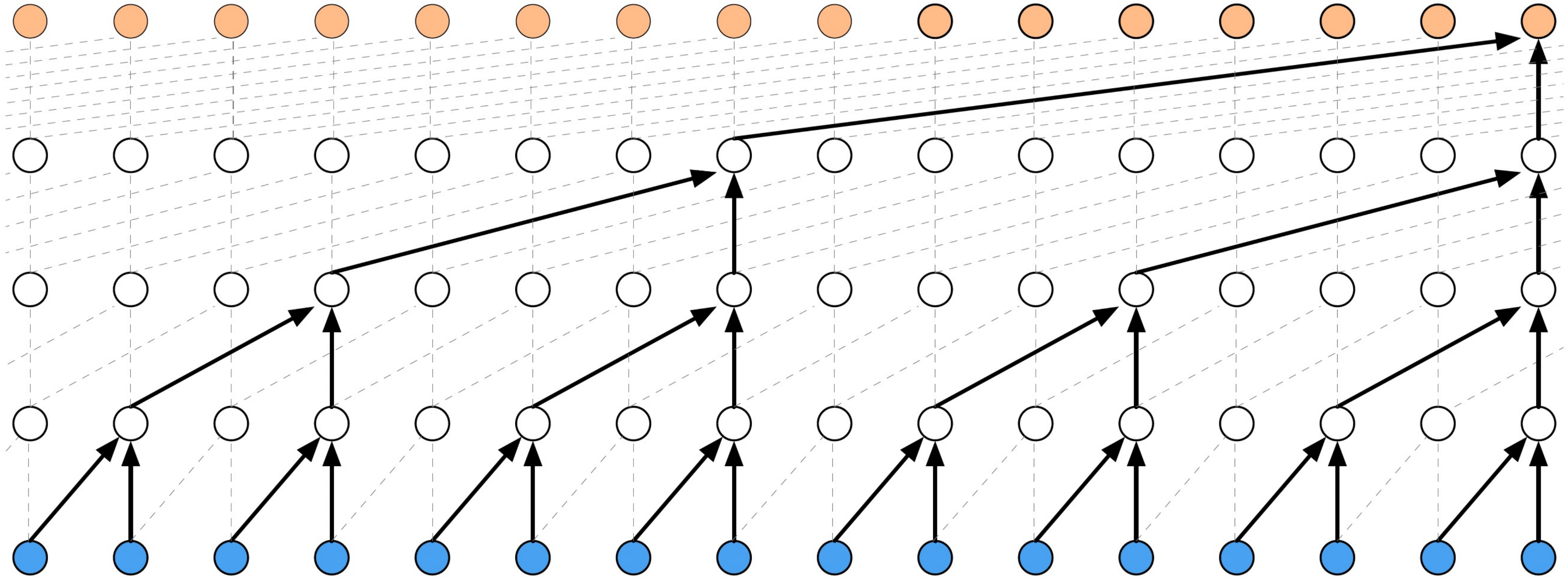
$p(x_4 \mid x_{1,2,3})$



Brief: Convolutional Neural Network (CNN) for AR



Example: WaveNet



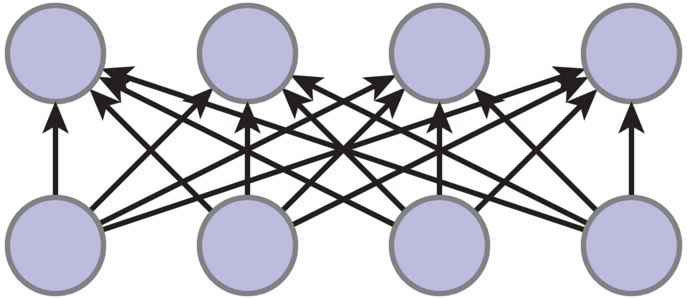
Audio generation with 1-D dilated causal conv

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

Brief: Attention (Transformer) for AR

full attention

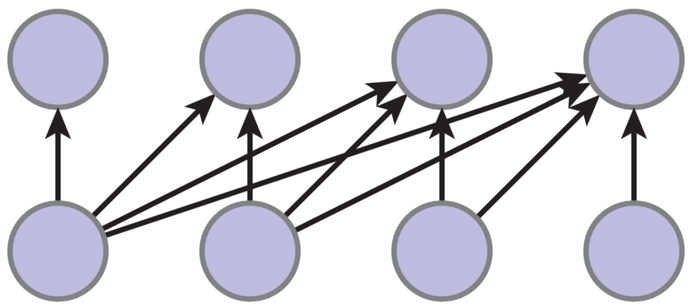
(every step sees all steps)



Brief: Attention (Transformer) for AR

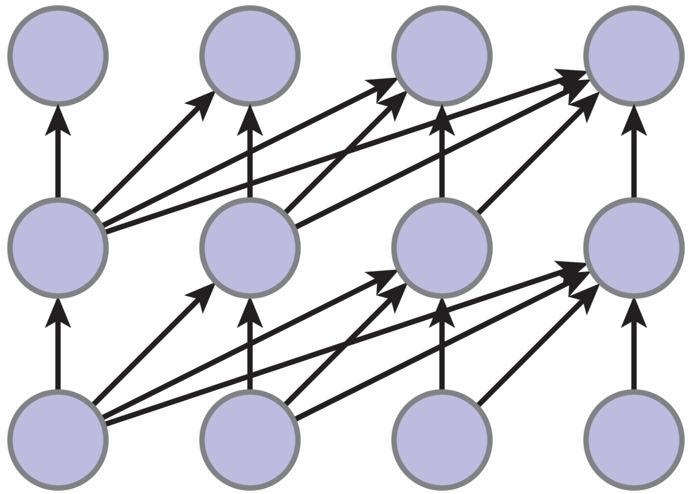
causal attention

(not depend on “future”)



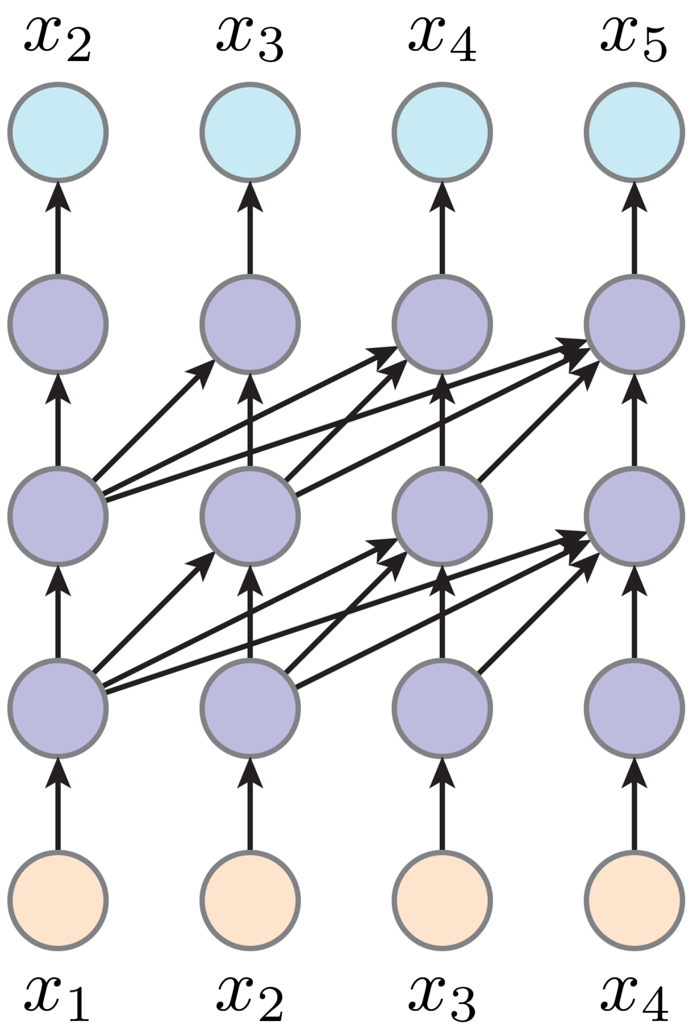
Brief: Attention (Transformer) for AR

go deep

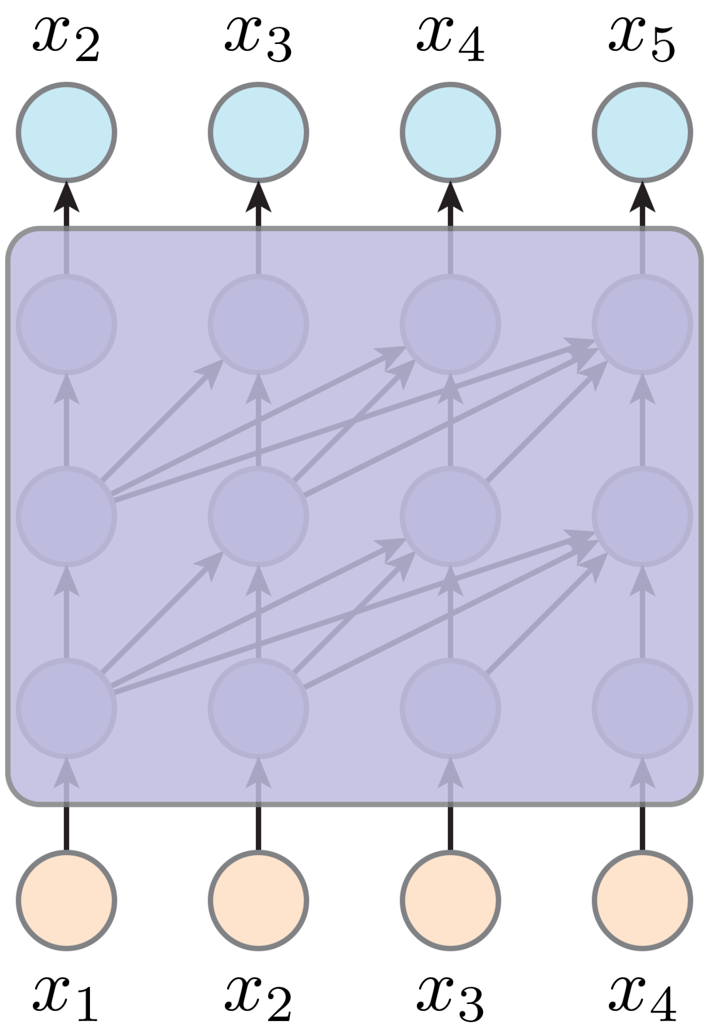


Brief: Attention (Transformer) for AR

shift target by one step

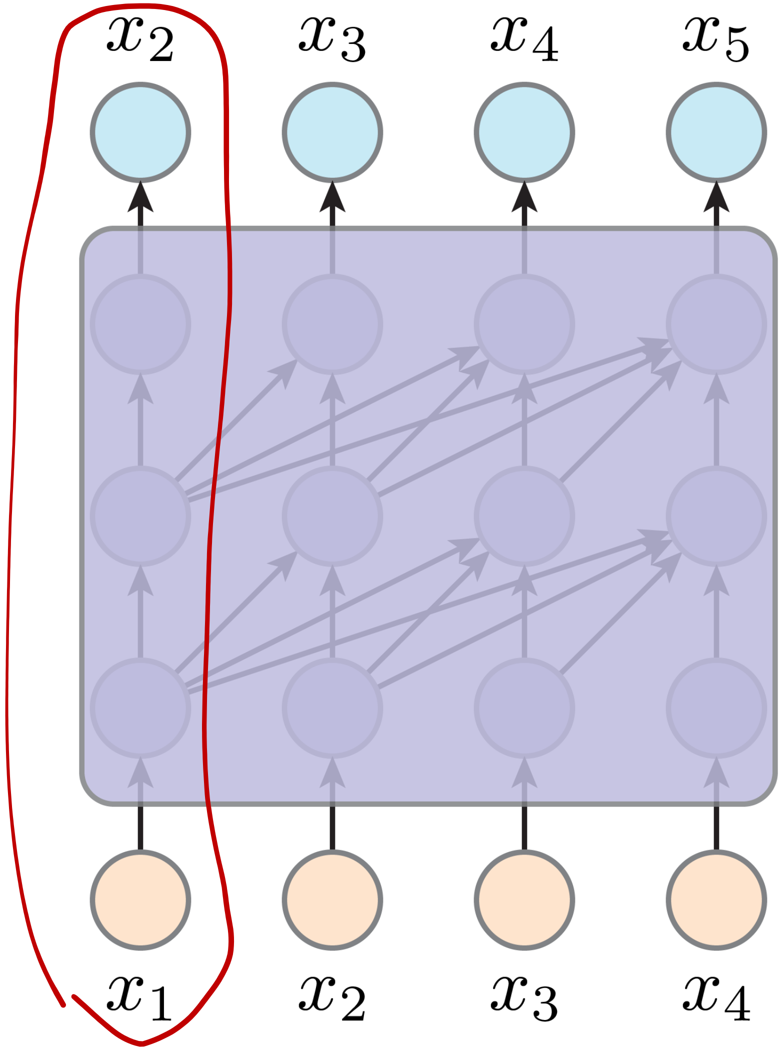


Brief: Attention (Transformer) for AR



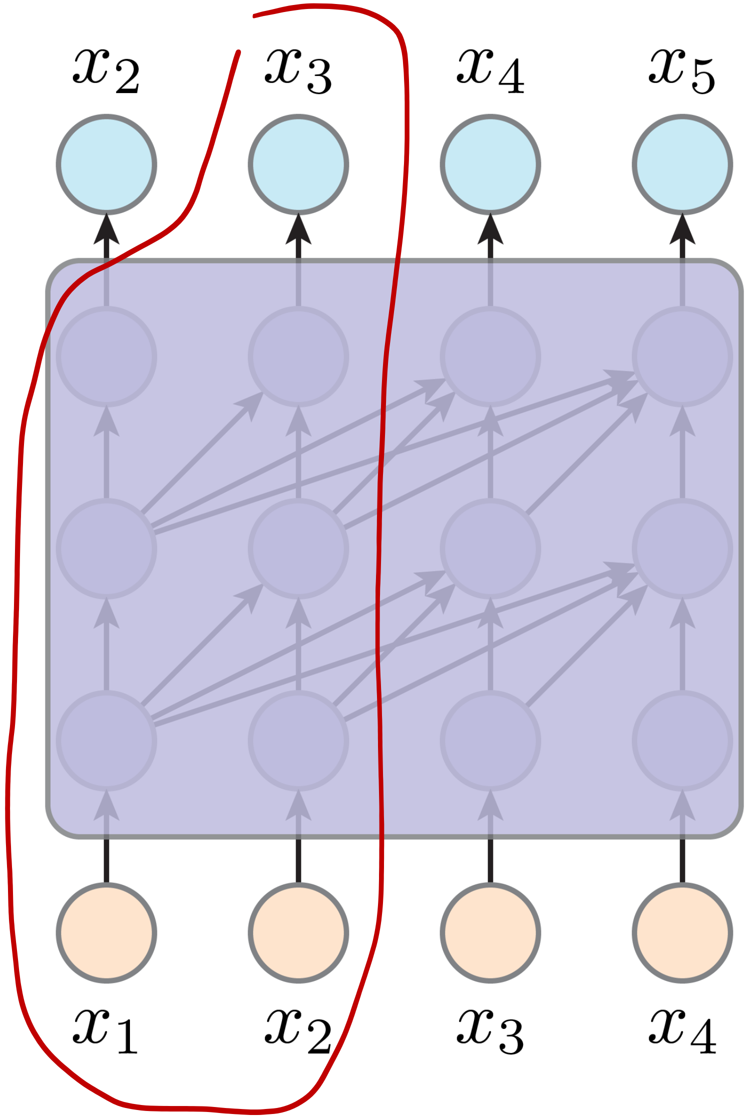
Brief: Attention (Transformer) for AR

$p(x_2 \mid x_1)$

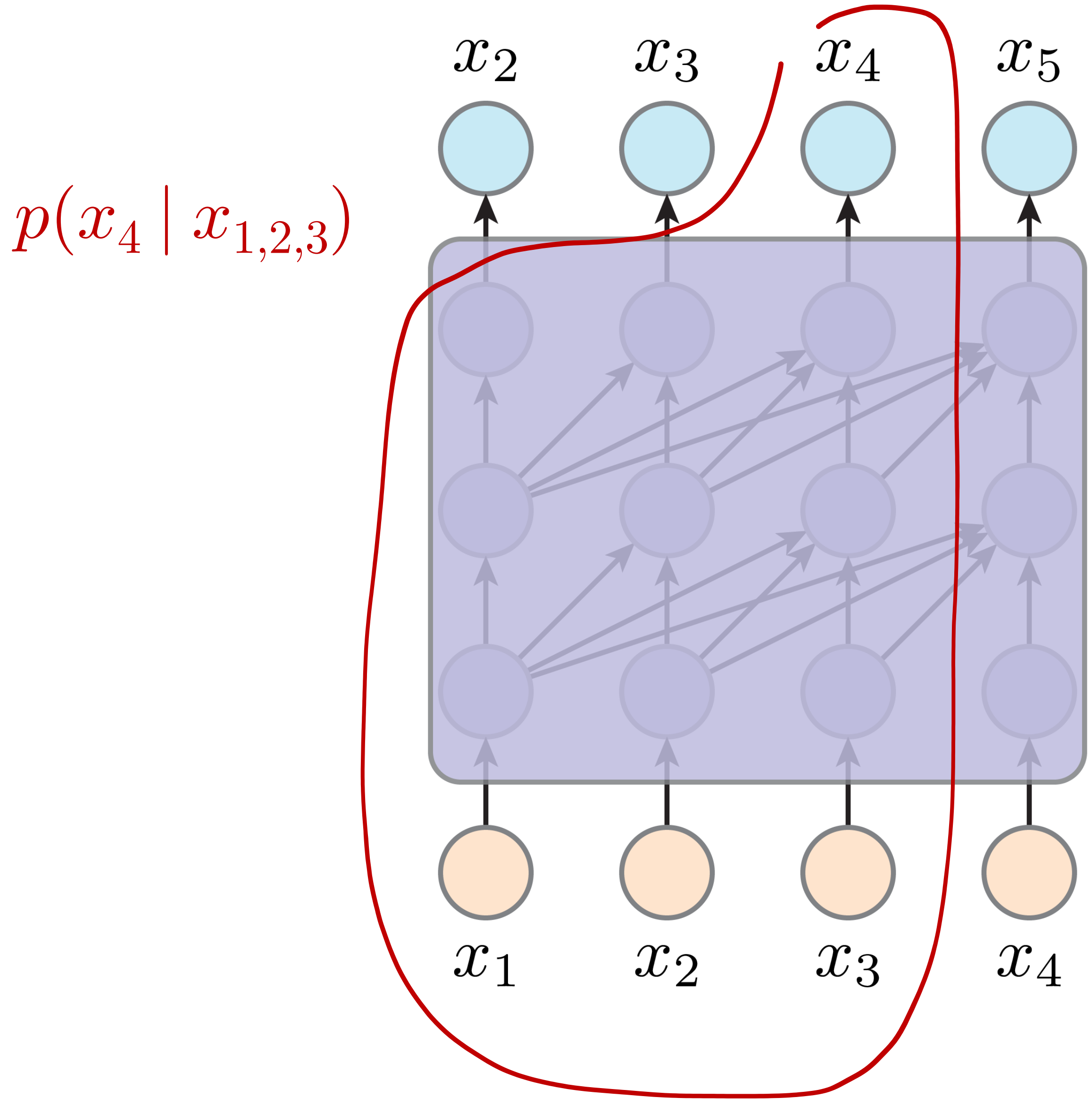


Brief: Attention (Transformer) for AR

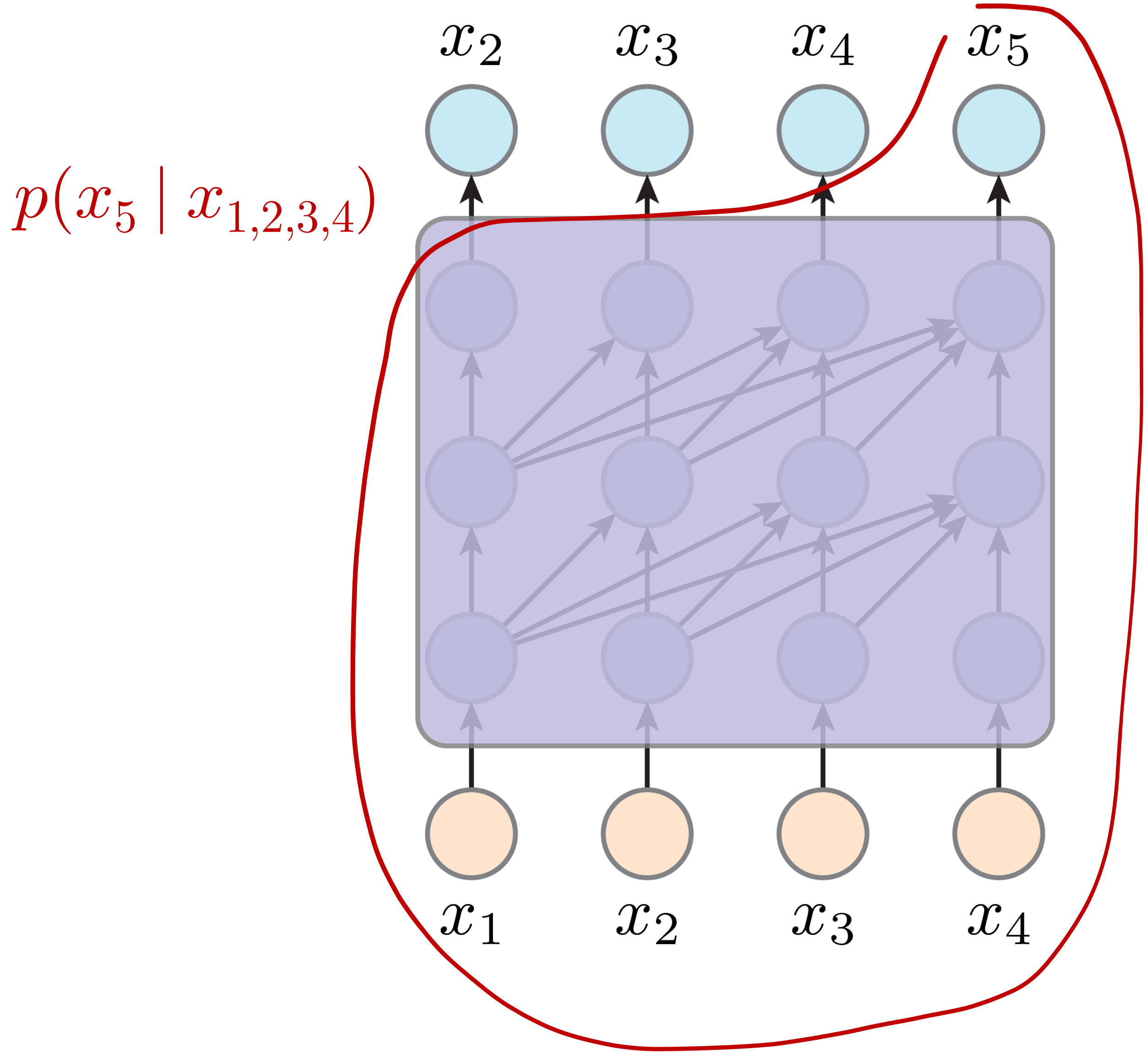
$p(x_3 \mid x_{1,2})$



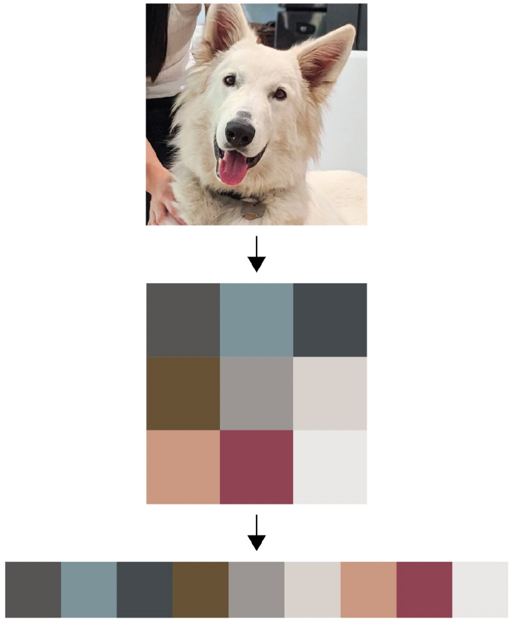
Brief: Attention (Transformer) for AR

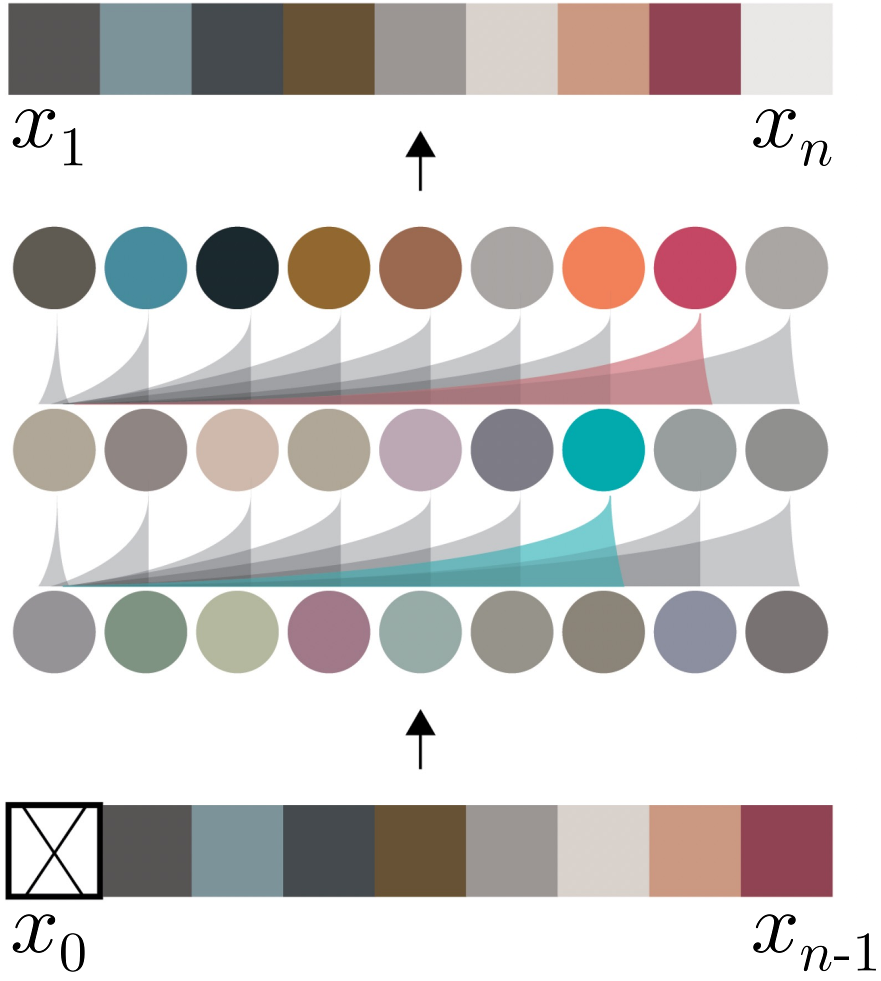


Brief: Attention (Transformer) for AR



Example: image GPT (iGPT)

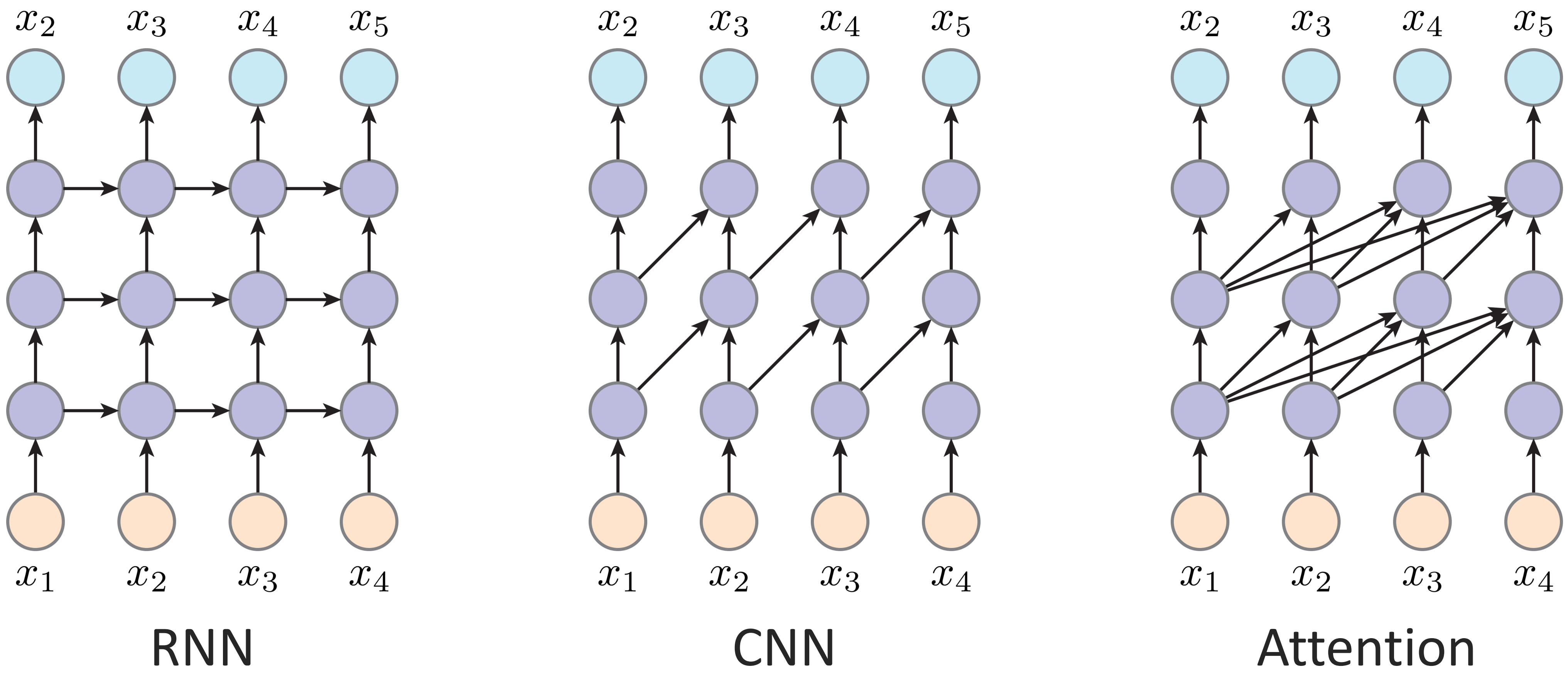




This figure is adapted from the original paper.

Chen, et al. "Generative Pretraining from Pixels", ICML 2020

Summary: Network Architectures for AR



See also my lecture at 6.8300: "11: Sequence Modeling": https://drive.google.com/file/d/1lOYsyImXl3caWlsplfThyuvgz-8Vn7yR/view?usp=sharing

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

This Lecture

- Conditional Distribution Modeling

- Autoregressive Models

- Network Architectures for Autoregressive Modeling

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