

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

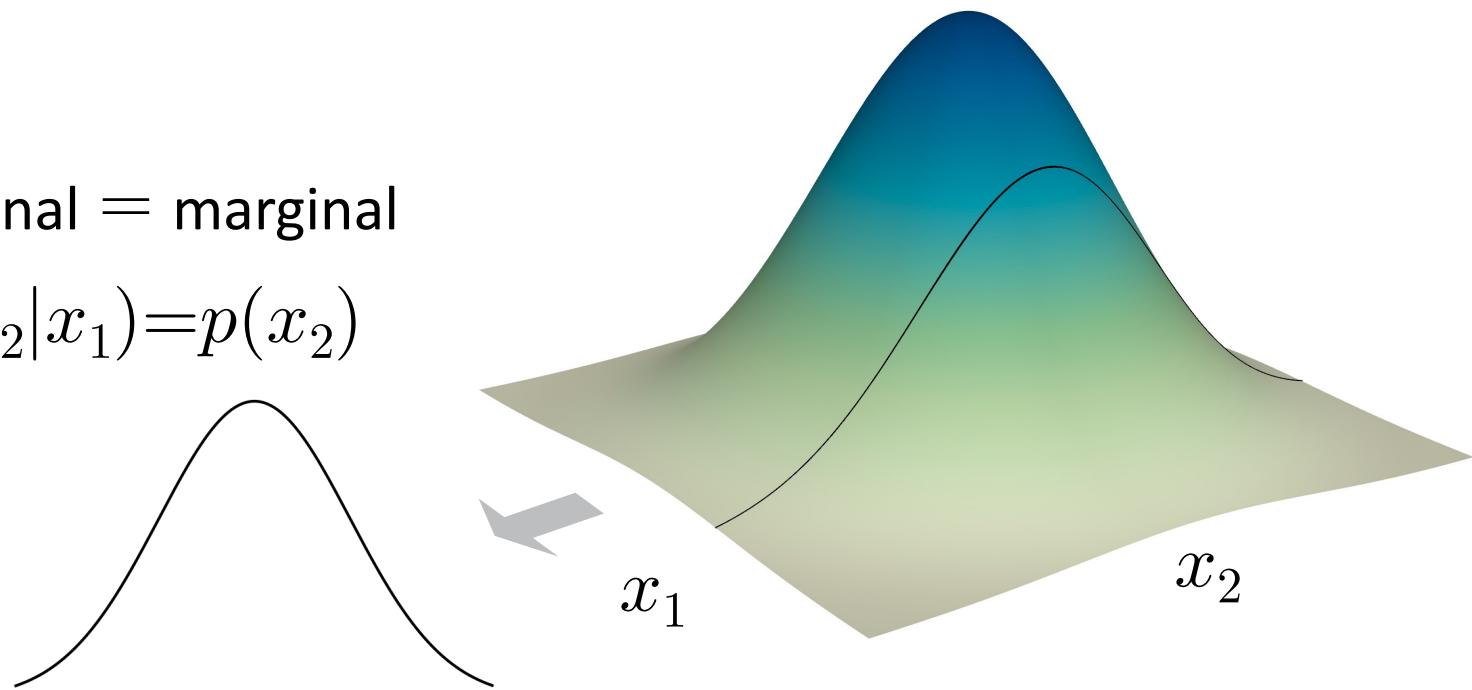
Joint Distribution

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

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

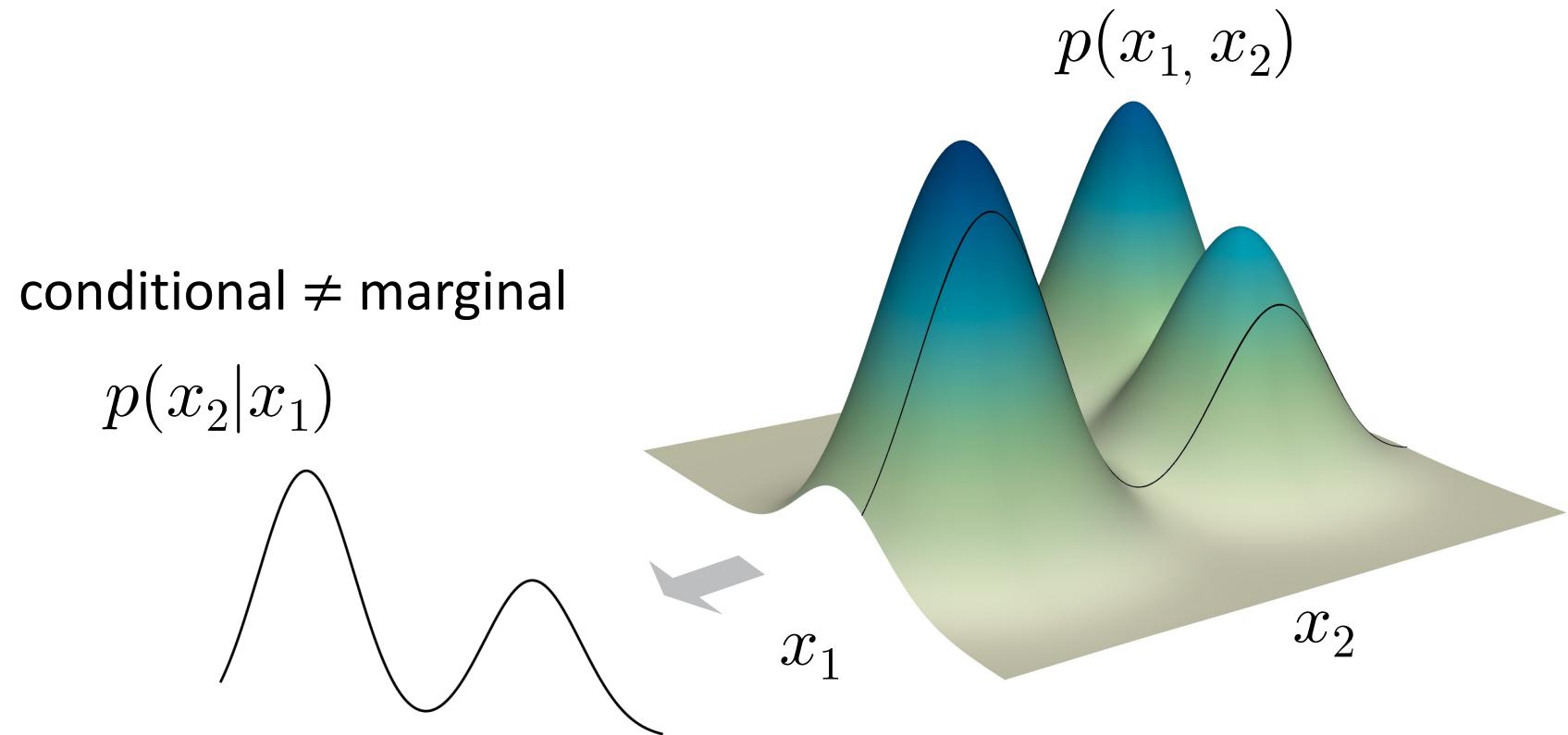
conditional = marginal

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



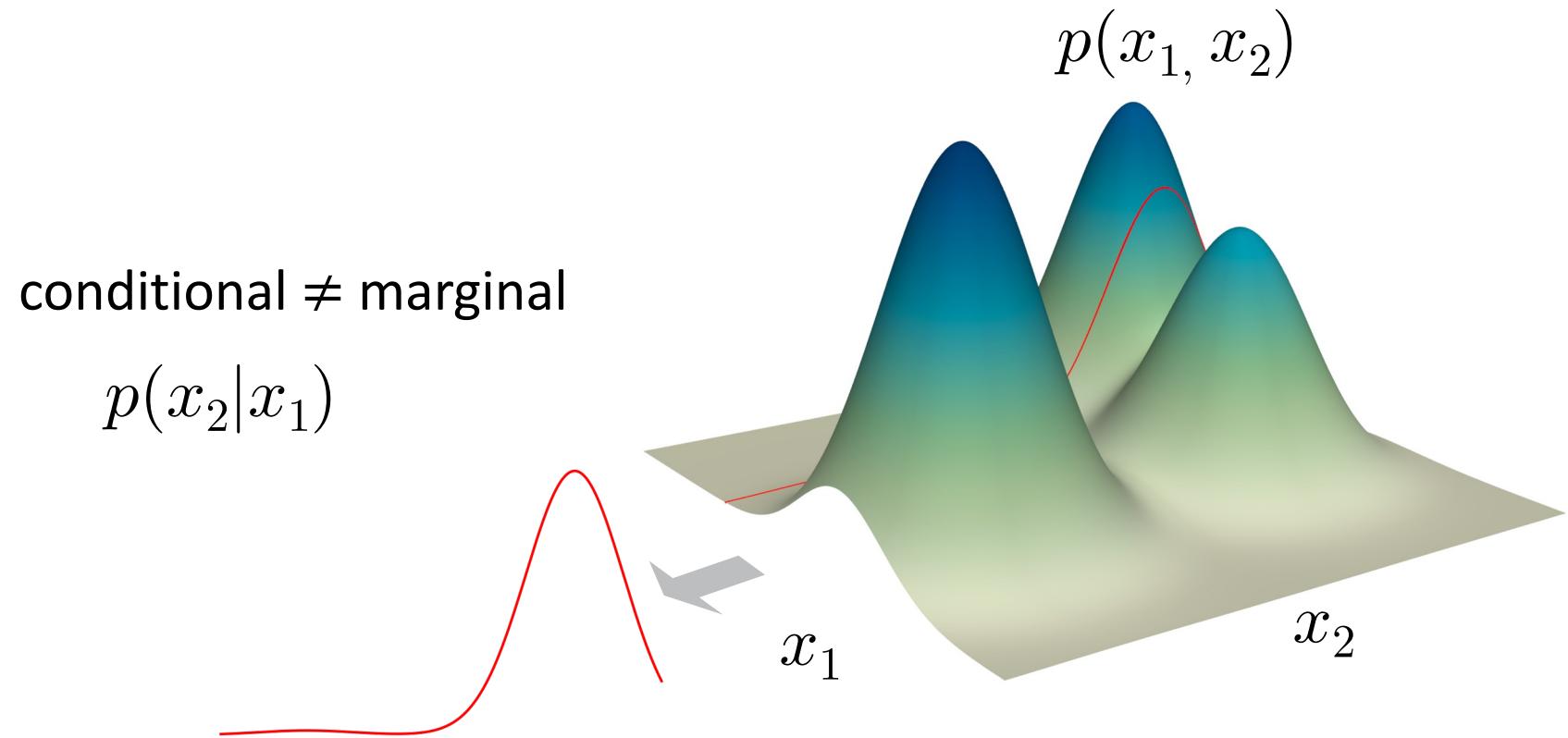
Joint Distribution

Real-word problems always involve dependent variables



Joint Distribution

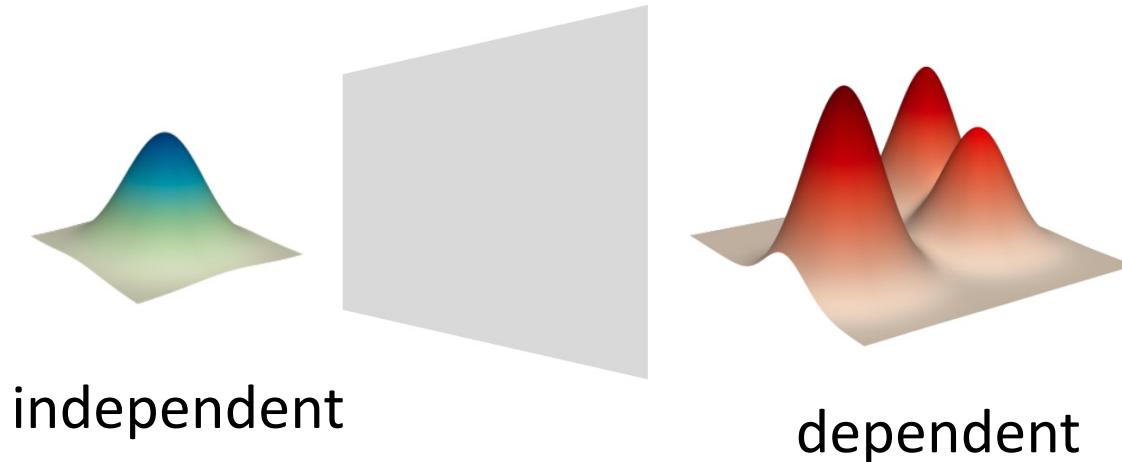
Real-word problems always involve dependent variables



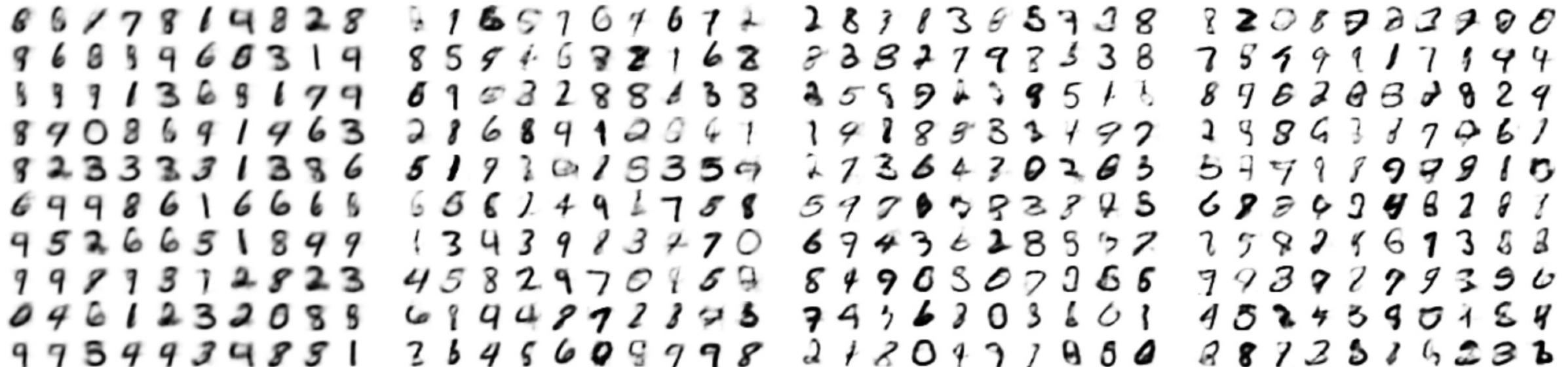
How to Model Joint Distributions?

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

- mapping: independent \Rightarrow 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

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

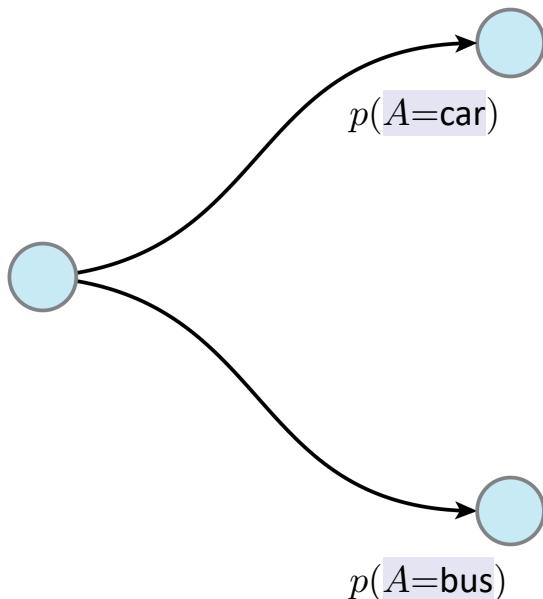
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

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

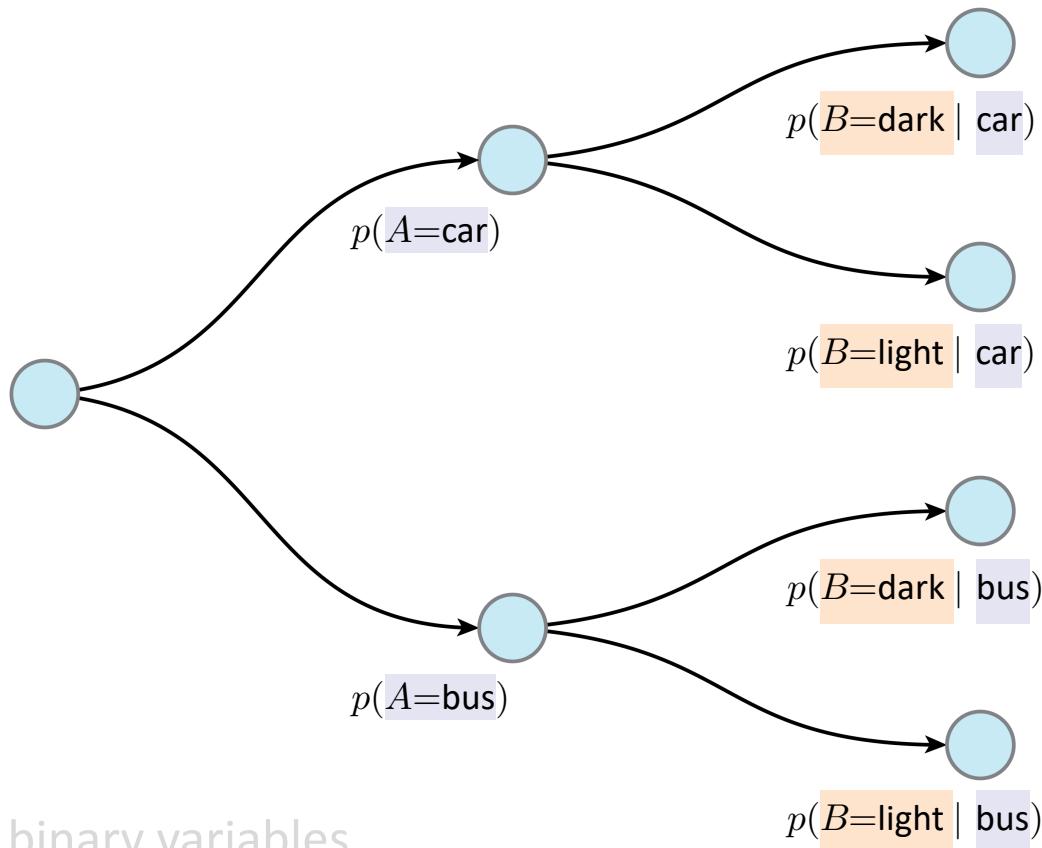


Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

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



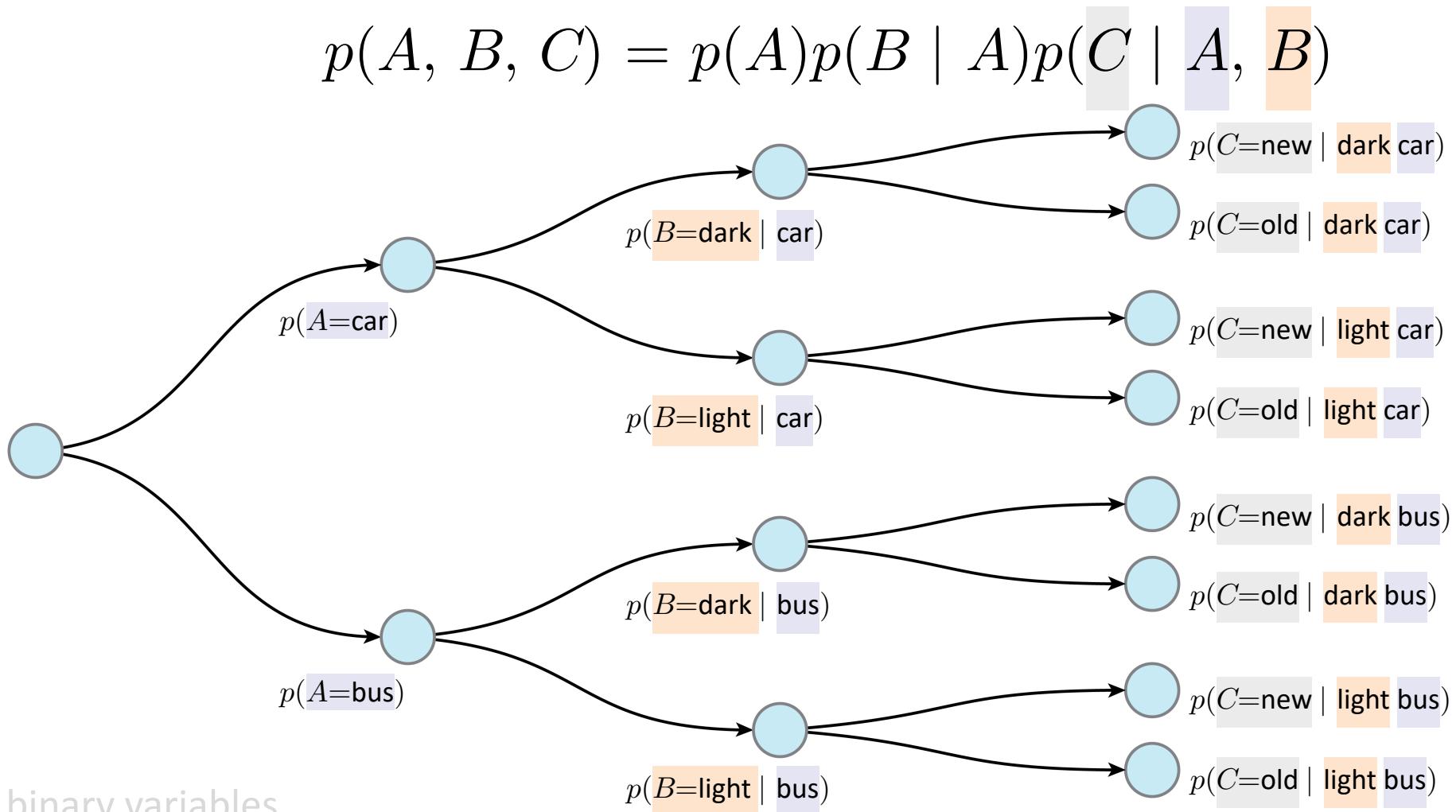
*example: binary variables

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

$$p(A, B, C) = p(A)p(B | A)p(C | A, B)$$



*example: binary variables

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

in any order:

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

Conditional Distribution Modeling

Chain rule:

Any joint distribution can be written as a product of conditionals

in any partition:

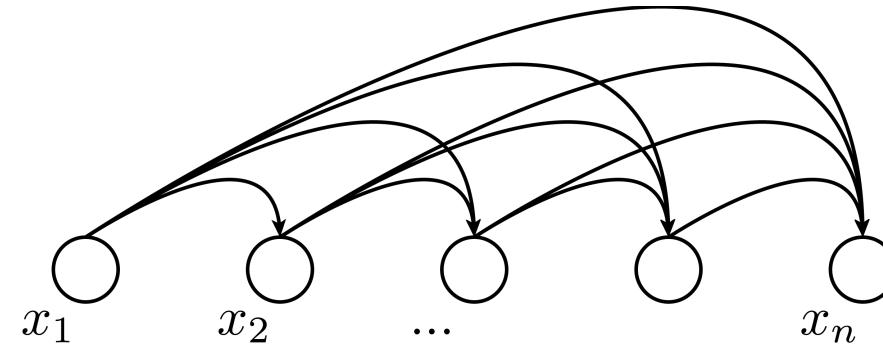
$$\begin{aligned} p(A, B, C, D) &= p(A, B)p(C, D \mid A, B) \\ &= p(C, D)p(A, B \mid C, D) \\ &= p(A, B, C)p(D \mid A, B, C) \\ &= \dots \end{aligned}$$

Case Study: Conditional Distribution Modeling

Case 1: Partitioning the input representation space x

Example: Autoregressive Models on text tokens or pixels

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



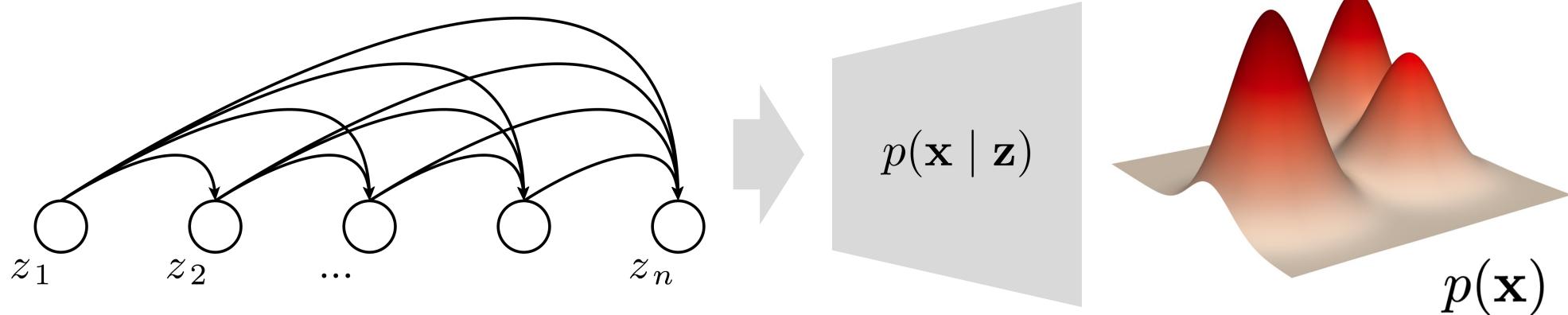
Case Study: Conditional Distribution Modeling

Case 2: Partitioning the latent representation space z

Example: Autoregressive Models on VQ-VAE tokens

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

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

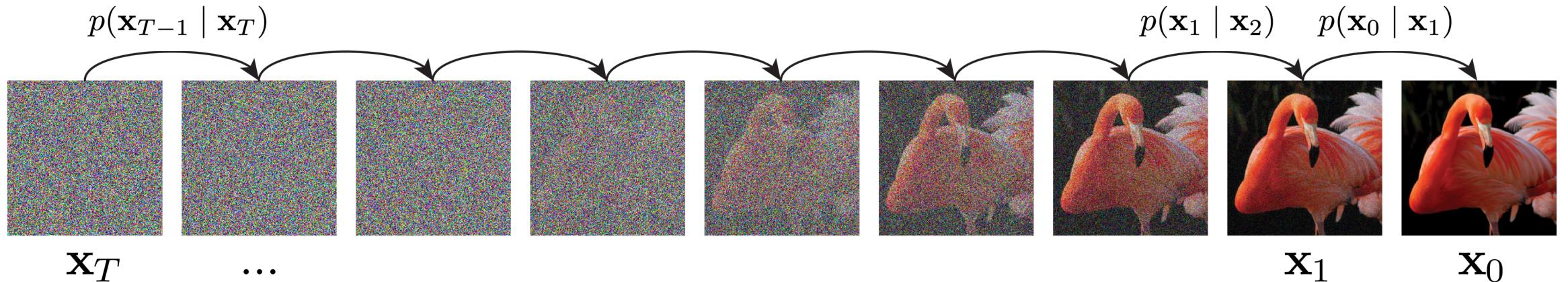


Case Study: Conditional Distribution Modeling

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

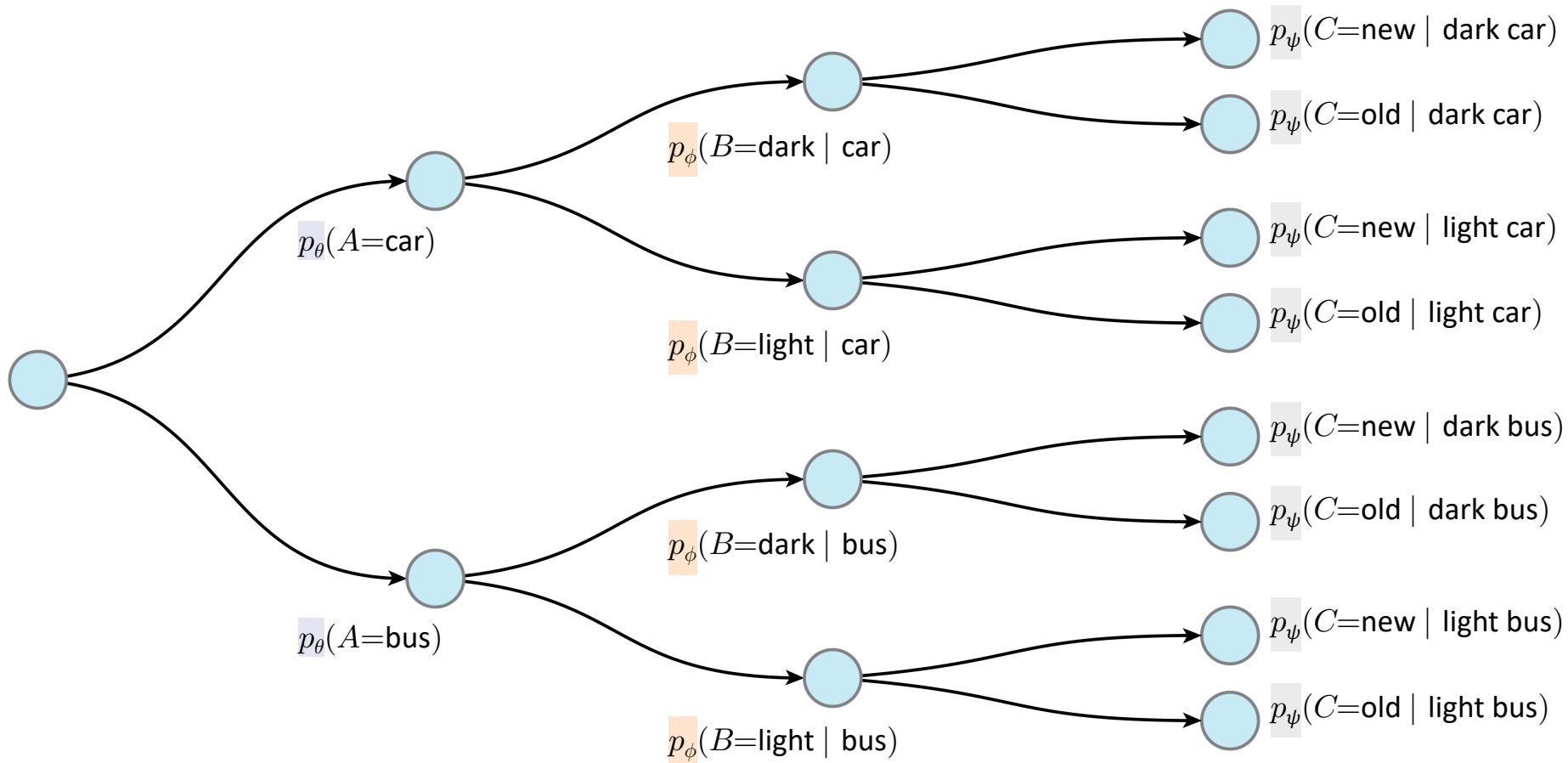
Modeling each conditional distribution with a neural network

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

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

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

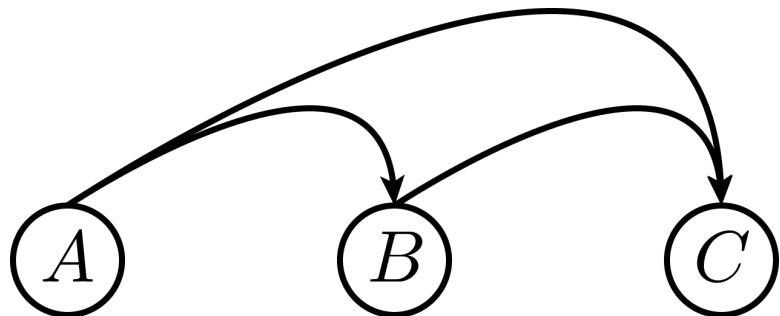
Note:

- 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 \Rightarrow induce a dependency graph
- Dependency graphs reflect prior knowledge

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

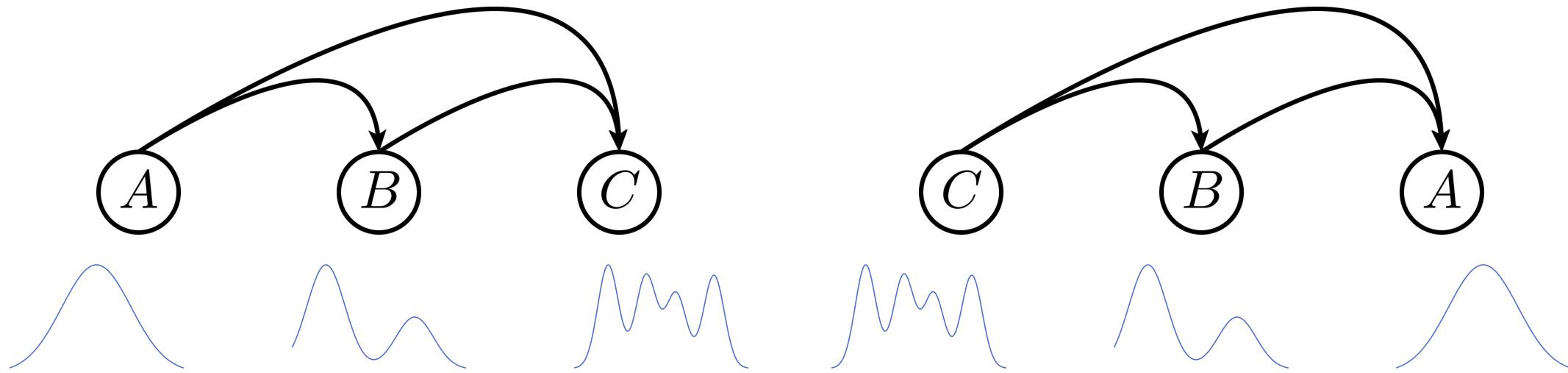


Dependency Graphs

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

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

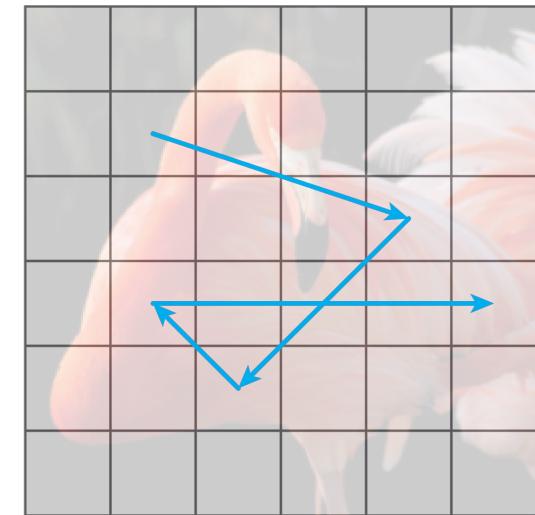
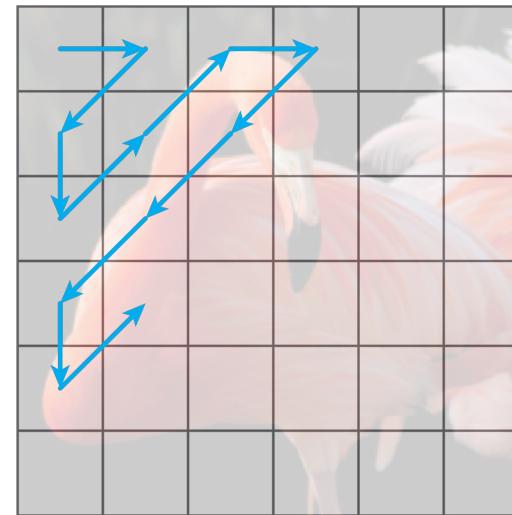
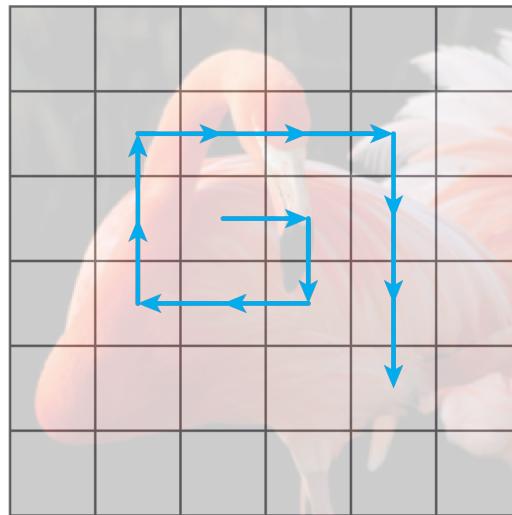
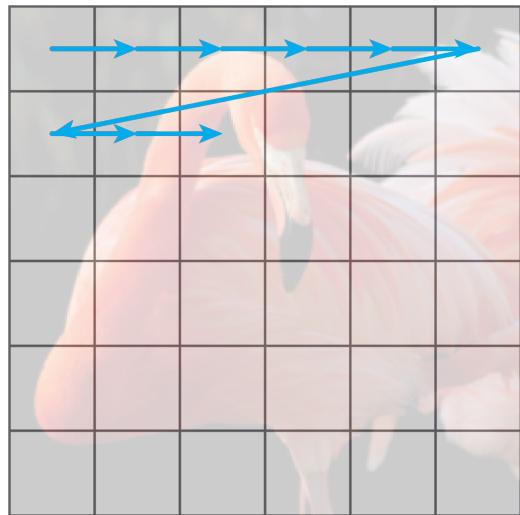
$$p(A, B, C) = p(C)p(B \mid C)p(A \mid B, C)$$



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

Dependency Graphs

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



Conditional Distribution Modeling

Summary:

- Joint distribution \Rightarrow 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

What are generative models?



Generative models are a class of machine learning models designed to generate new data samples that resemble a given dataset. They aim to learn the underlying distribution.

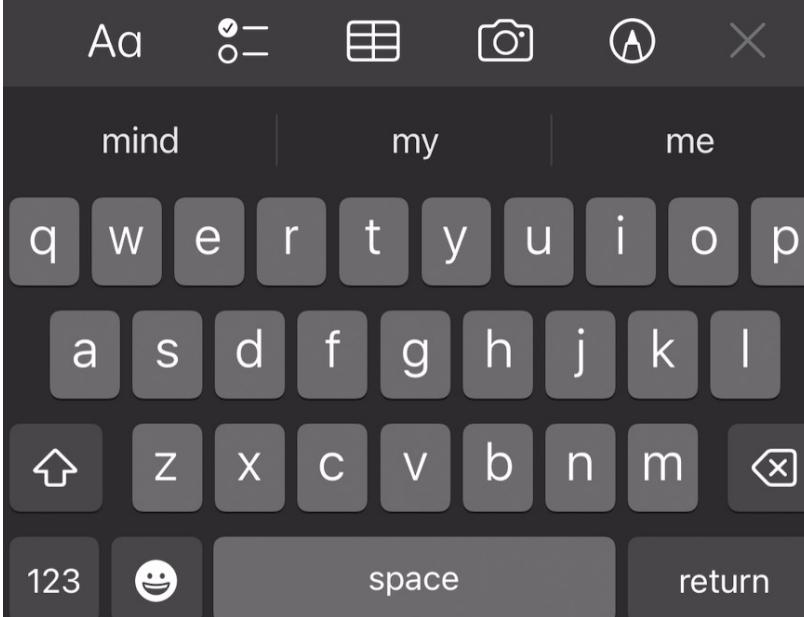


Message ChatGPT



Your Keyboard

the first thing i noticed
was that the first thing
that came to |



Auto + Regression

Auto: “self”

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

Regression:

- estimating relationship between variables

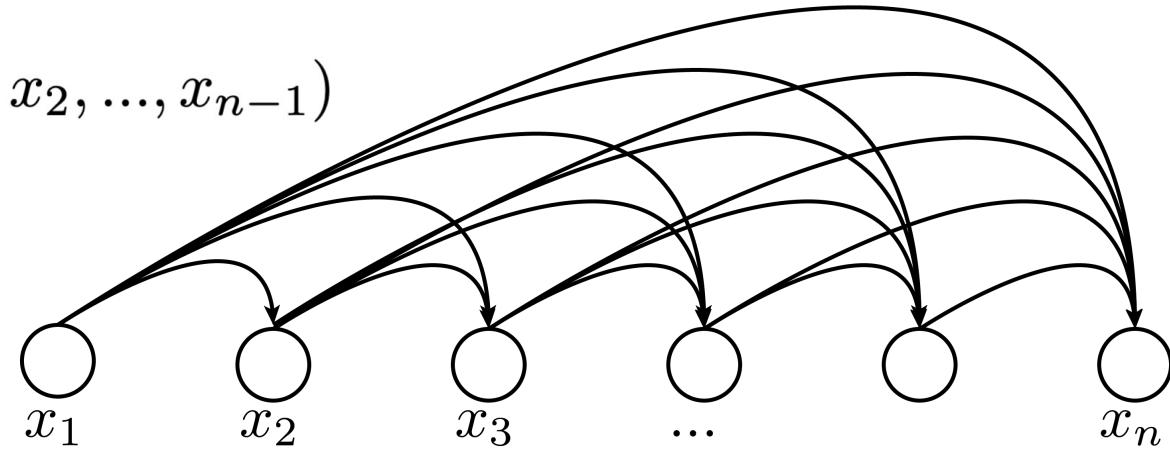
Note:

- “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:

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



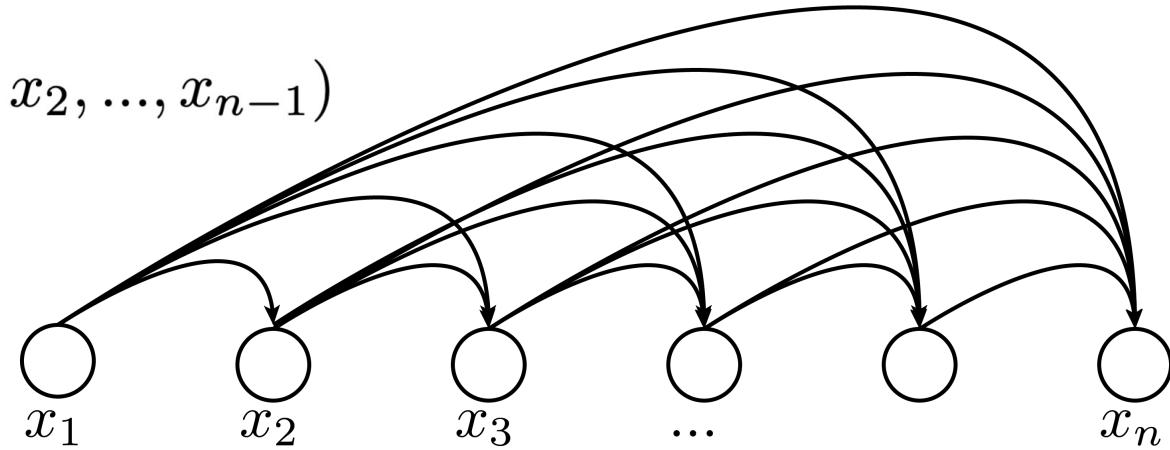
Conceptually, ...

- 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:

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



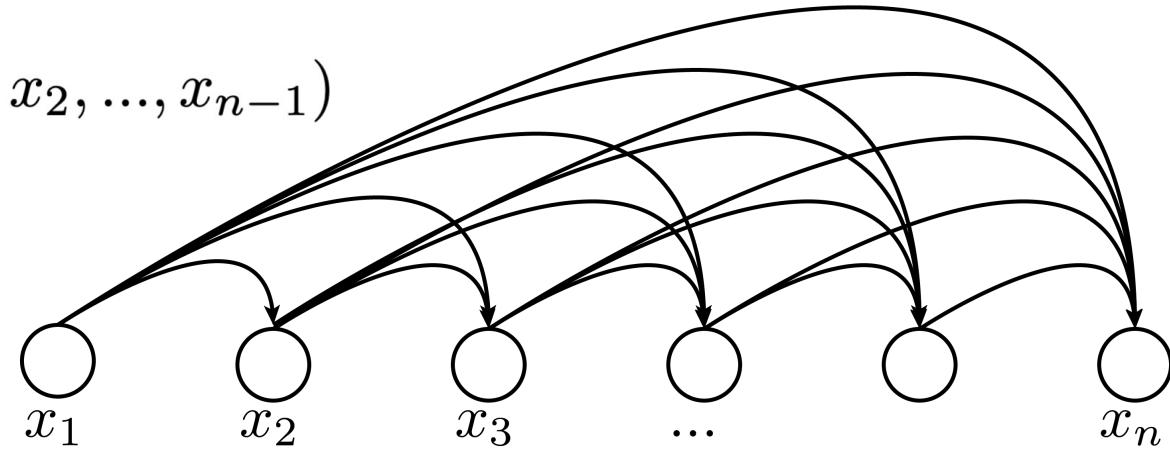
Conceptually, ...

- 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:

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



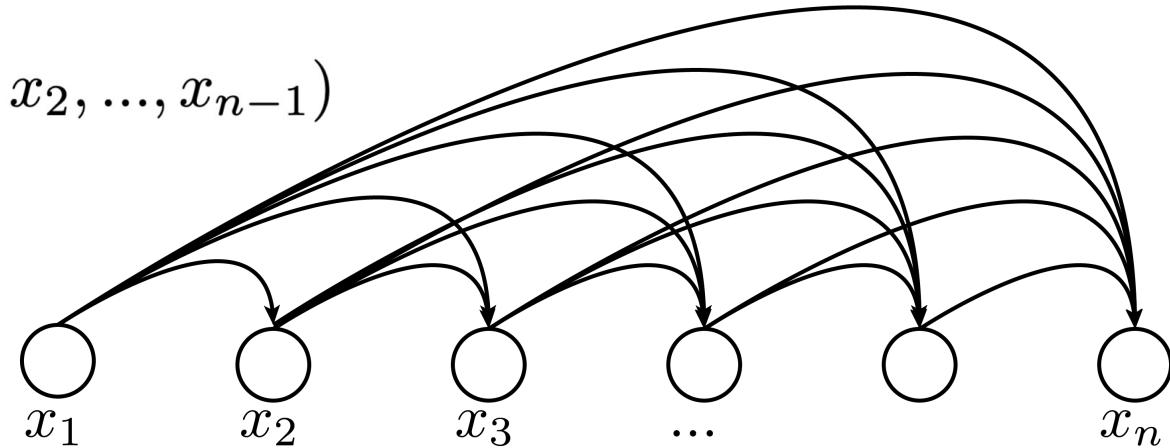
Conceptually, ...

- each $p(\cdot | \cdot)$ 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:

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



This formulation makes **no** compromise/approximation

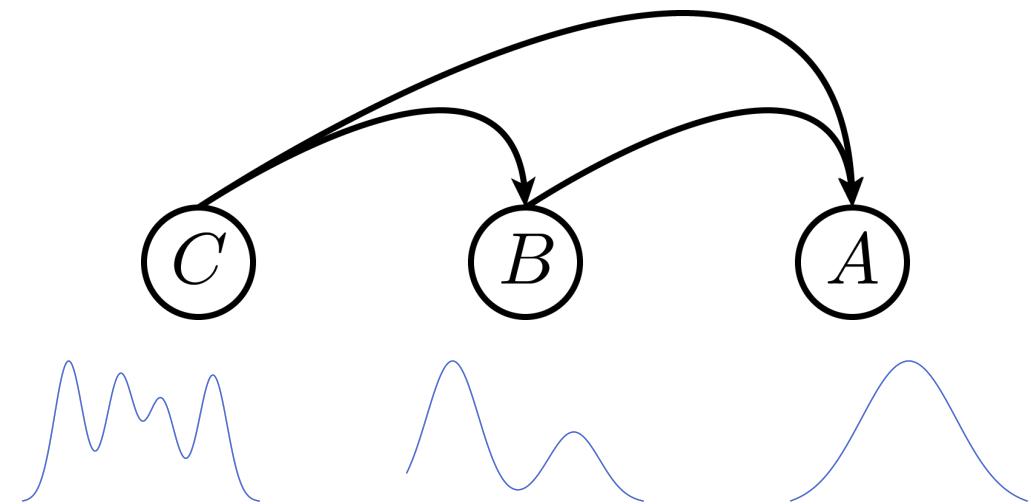
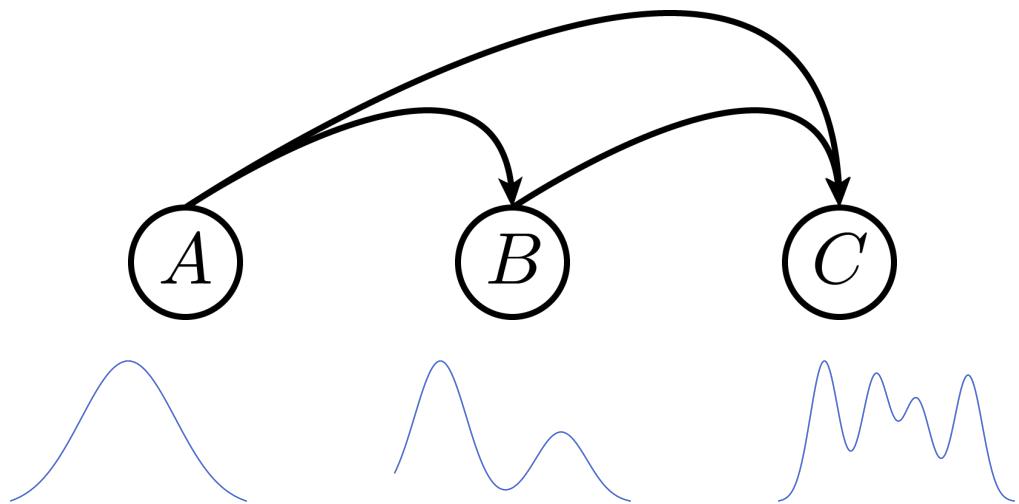
- This decomposition is always valid (just chain rule)
- But some are easier to model: “inductive bias” ...

Inductive Bias

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

$$p(A, B, C) = p(A)p(B | A)p(C | A, B)$$

$$p(A, B, C) = p(C)p(B | C)p(A | B, C)$$



Your phone's keyboard is Autoregressive:

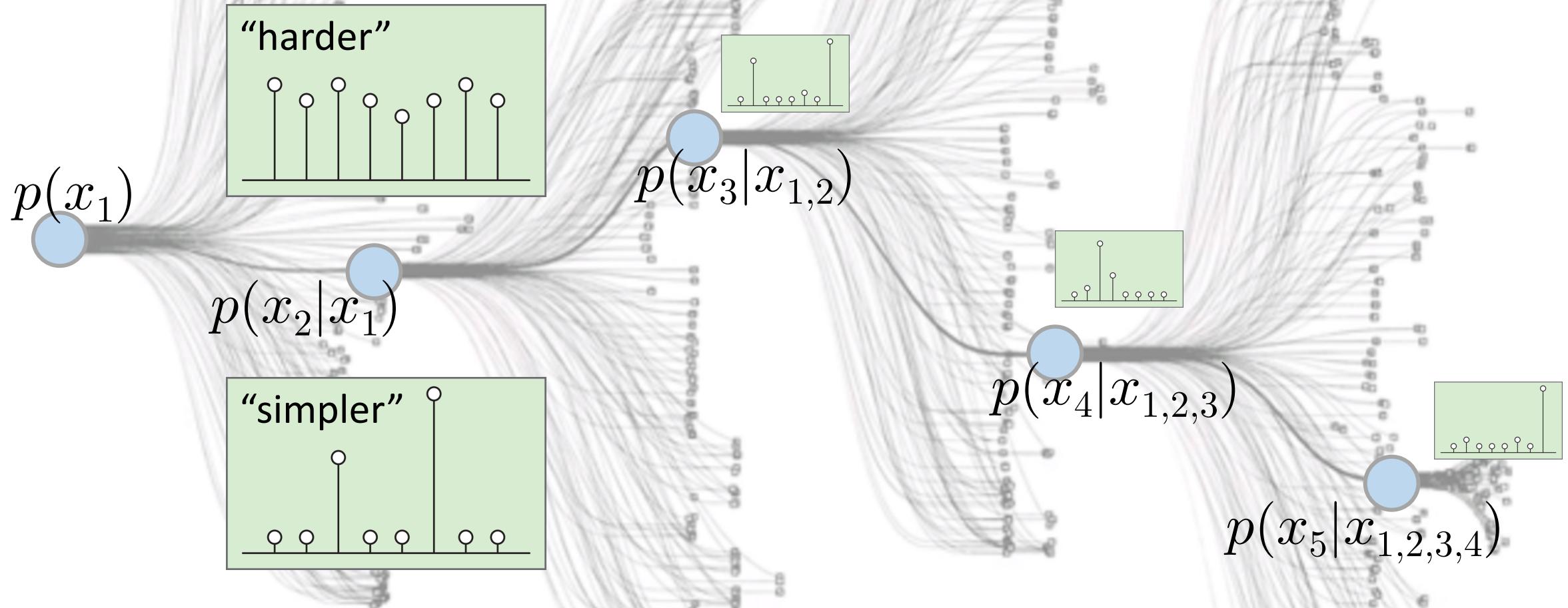
Previous outputs can largely reduce the next plausible outputs.

**the first thing i noticed
was that the first thing
that came to |**



Inductive Bias

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



Example: every p is a categorical distribution

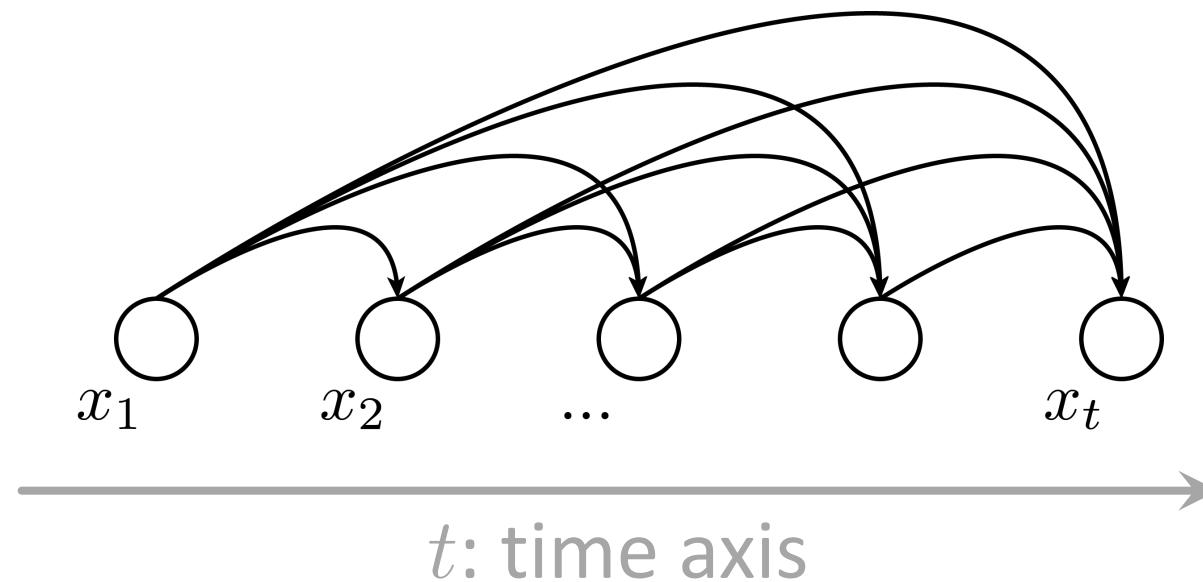
Illustration adapted from AlphaGo

Inductive Bias

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

Example: “next token prediction”

- Temporal modeling implies an inductive bias



Inductive Bias

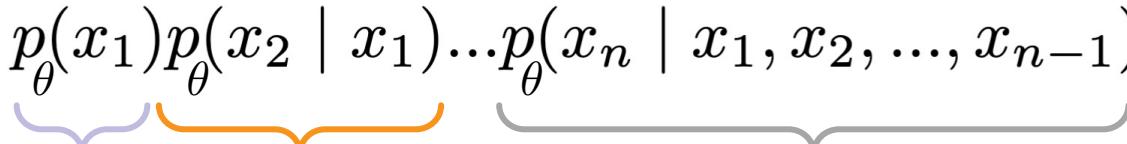
We want the decomposed distributions to be represented by “similar” neural networks ...

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


- 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 ...

$$p(x_1, x_2, \dots, x_n) = p_{\theta}(x_1) p_{\theta}(x_2 | x_1) \dots p_{\theta}(x_n | x_1, x_2, \dots, x_{n-1})$$


- 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:

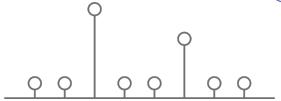
$$p(x_1, x_2, \dots, x_n) = p(x_1)p(x_2 | x_1)\dots p(x_n | x_1, x_2, \dots, x_{n-1})$$

But **inductive biases** introduce approximations:

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

Representing One Distribution

$$p_{\theta}(x_i \mid \underbrace{x_1, x_2, \dots, x_{i-1}}_{\text{Network inputs}})$$

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

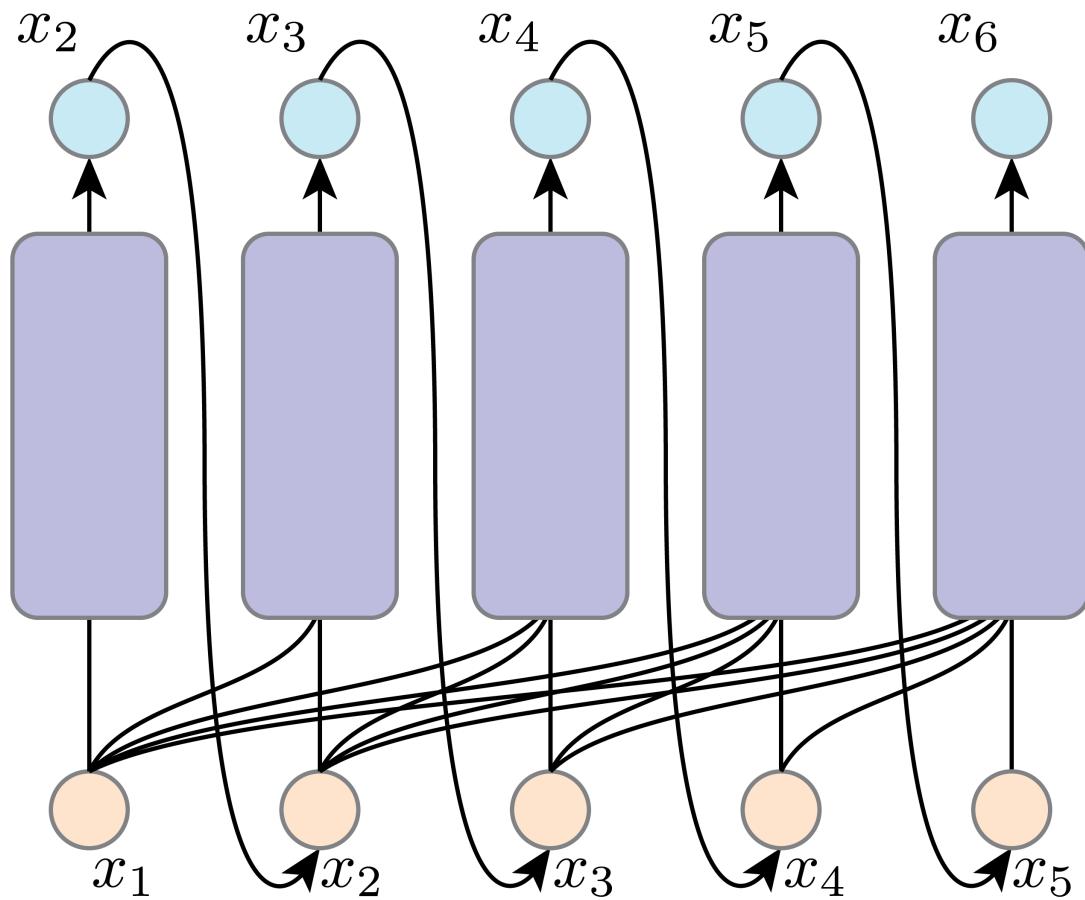
Note:

- 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

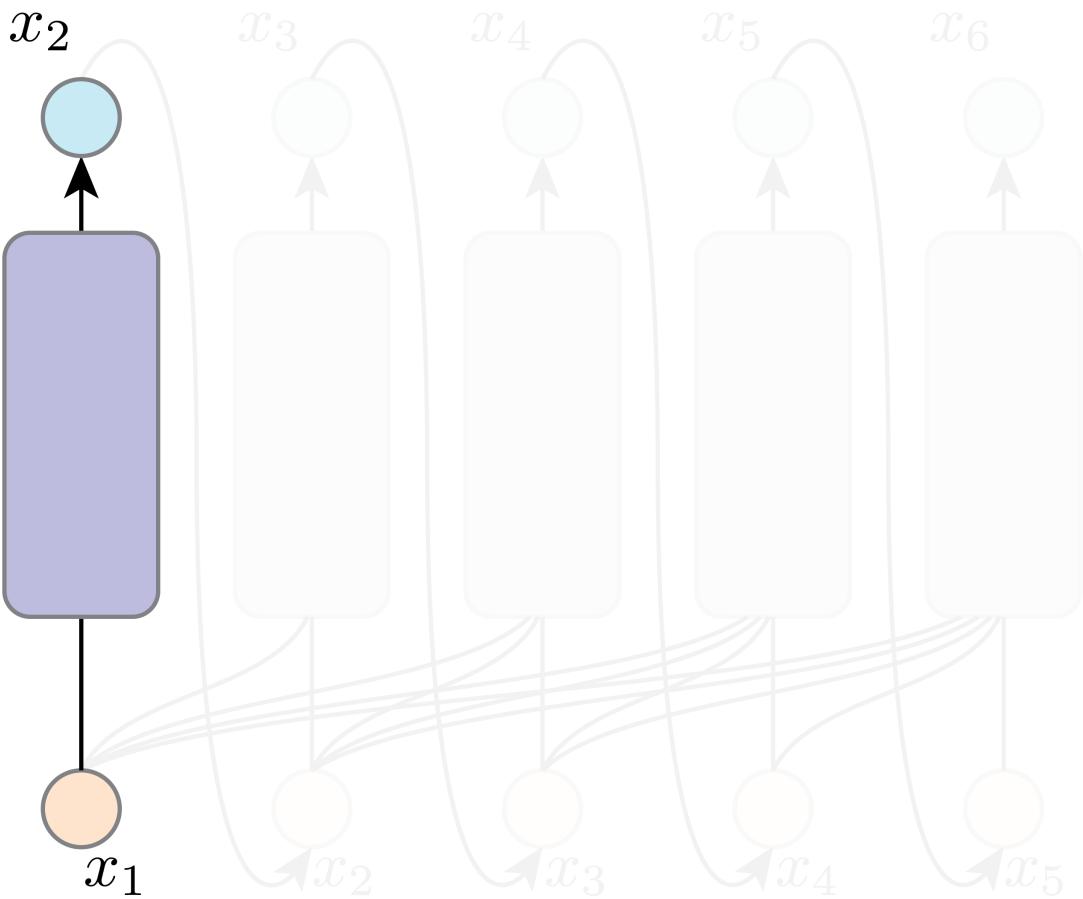
This figure implements this formulation:

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



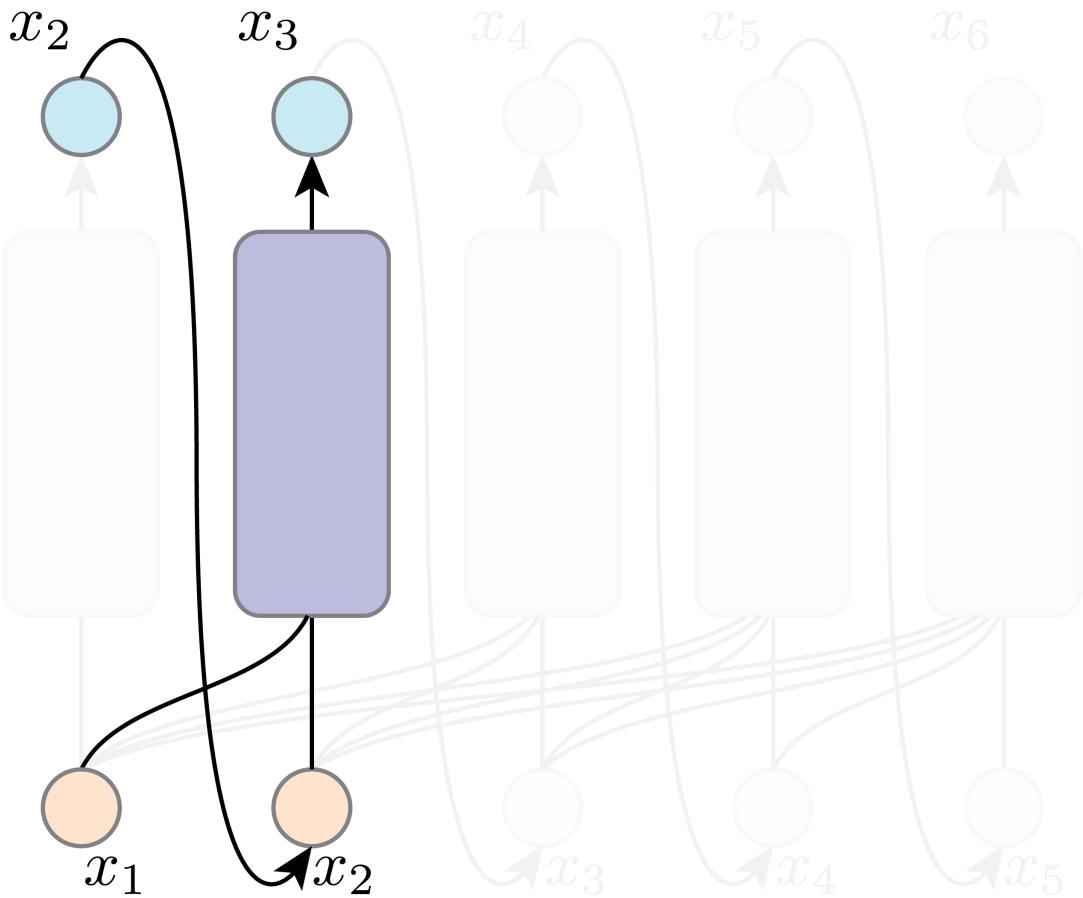
Inference: Autoregressive

- This **net** models $p(x_2 | x_1)$
- **1 input**
- **1 output**



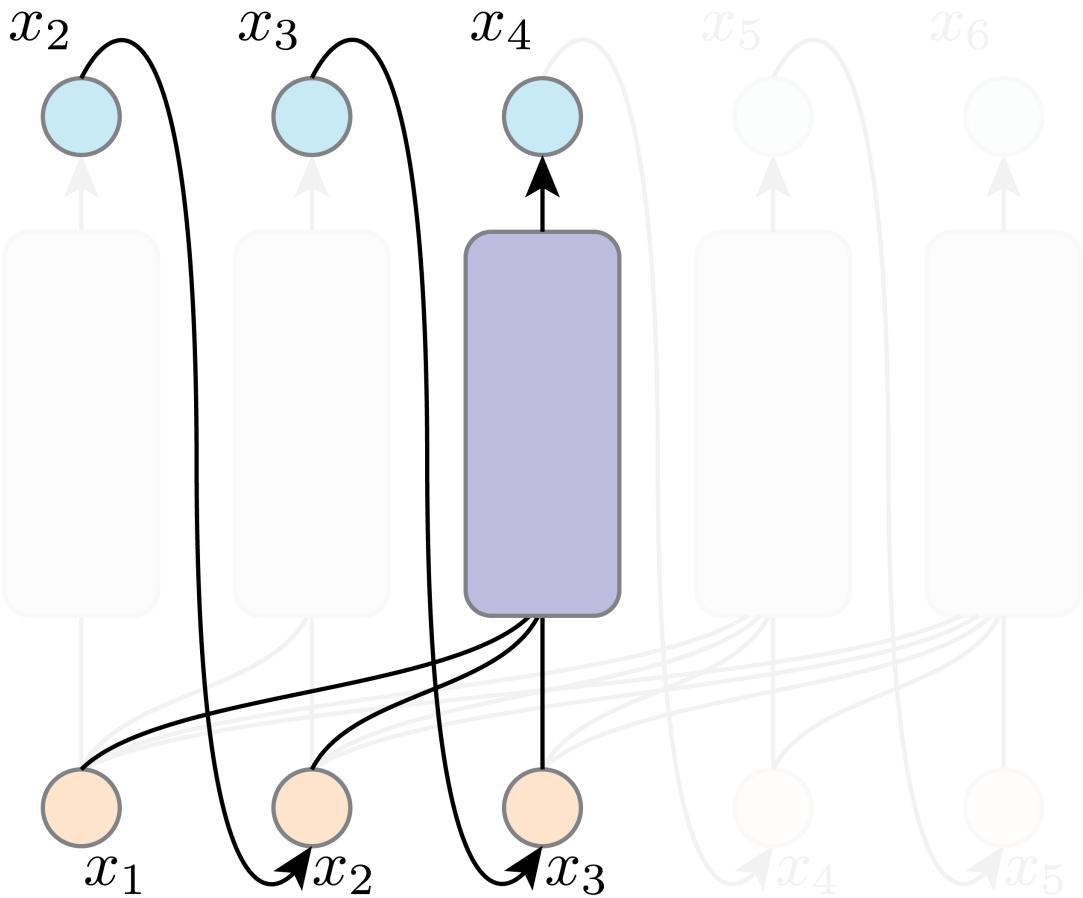
Inference: Autoregressive

- This **net** models $p(x_3 | x_{1,2})$
- **2 inputs**
- **1 output**
- inputs: outputs from previous steps



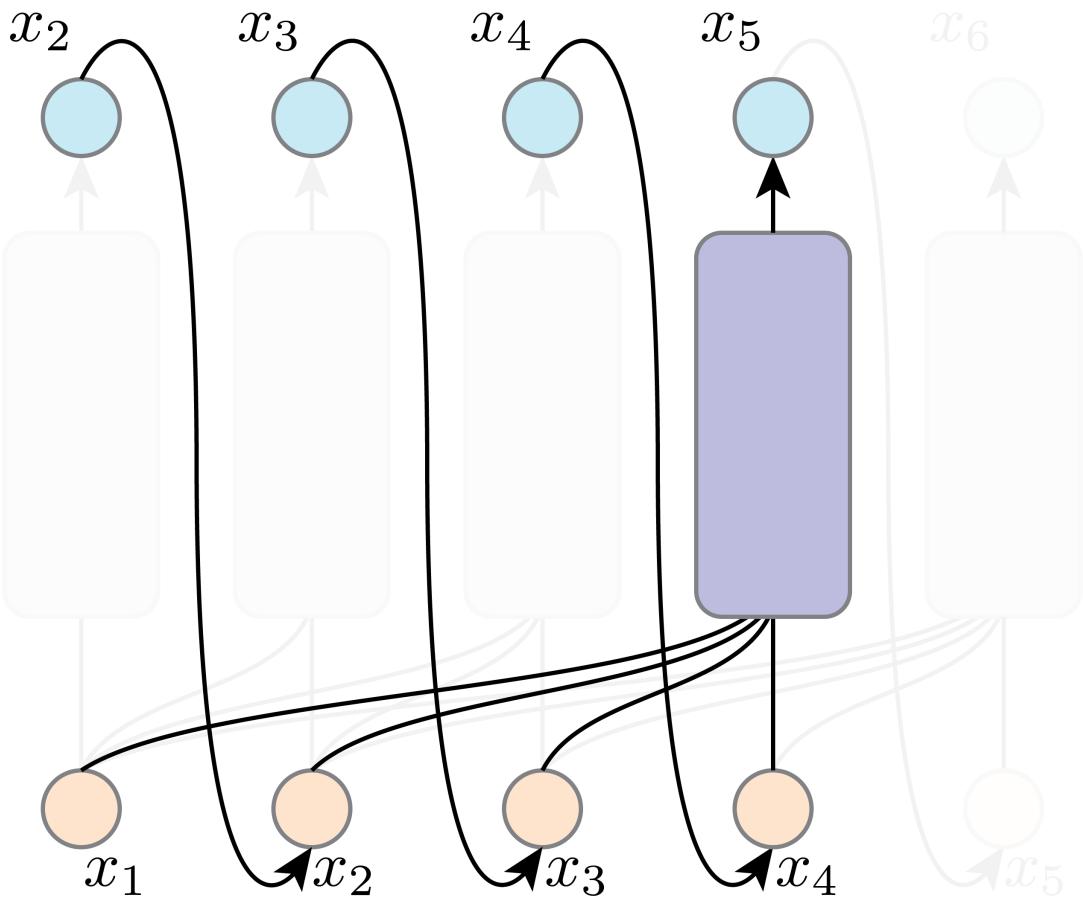
Inference: Autoregressive

- This **net** models $p(x_4 | x_{1,2,3})$
- **3 inputs**
- **1 output**
- inputs: outputs from previous steps



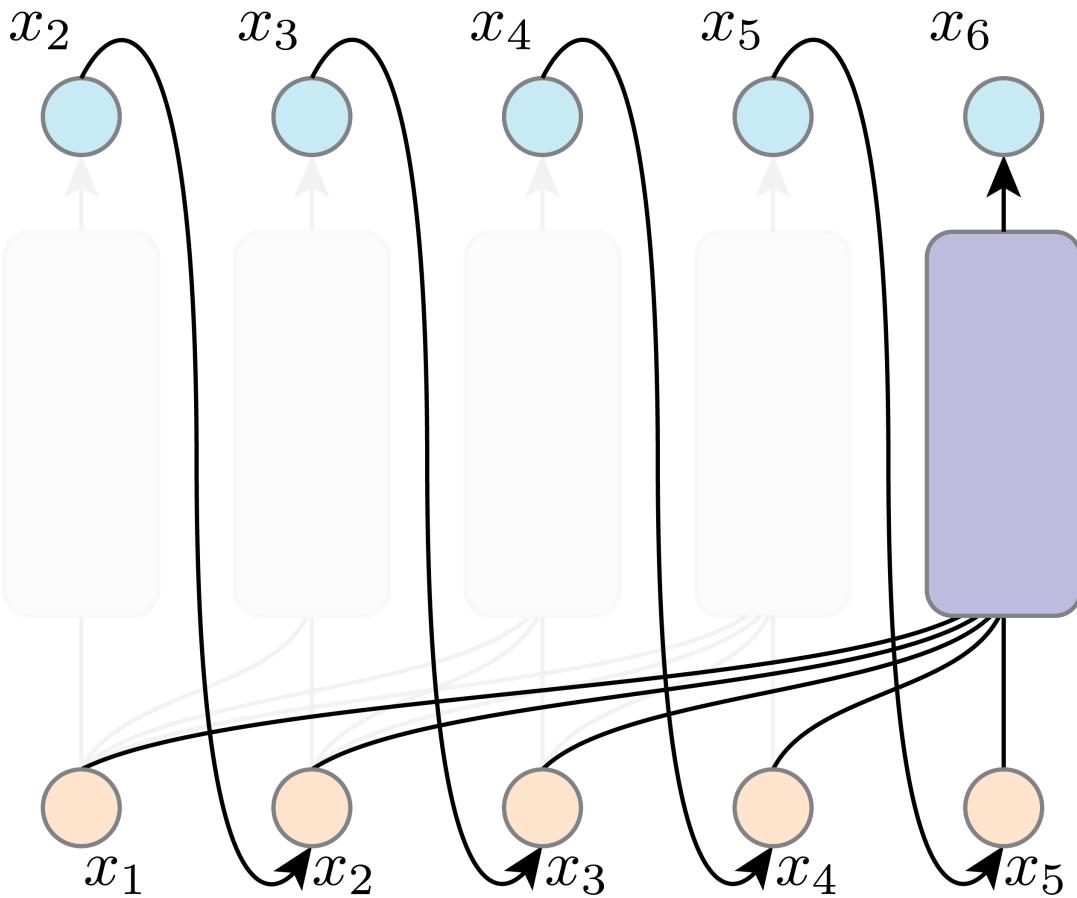
Inference: Autoregressive

- This **net** models $p(x_5 | x_{1,2,3,4})$
- **4 inputs**
- **1 output**
- inputs: outputs from previous steps



Inference: Autoregressive

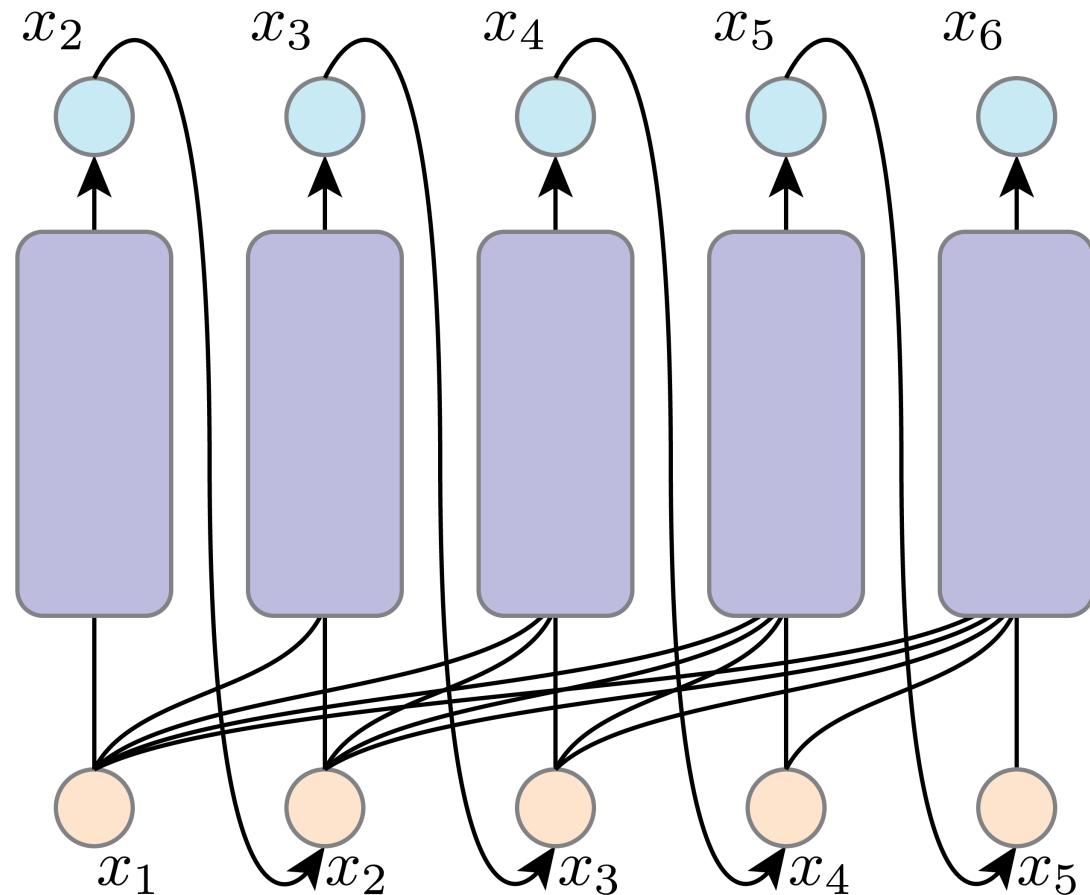
- This **net** models $p(x_6 | x_{1,2,3,4,5})$
- **5 inputs**
- **1 output**
- inputs: outputs from previous steps



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)



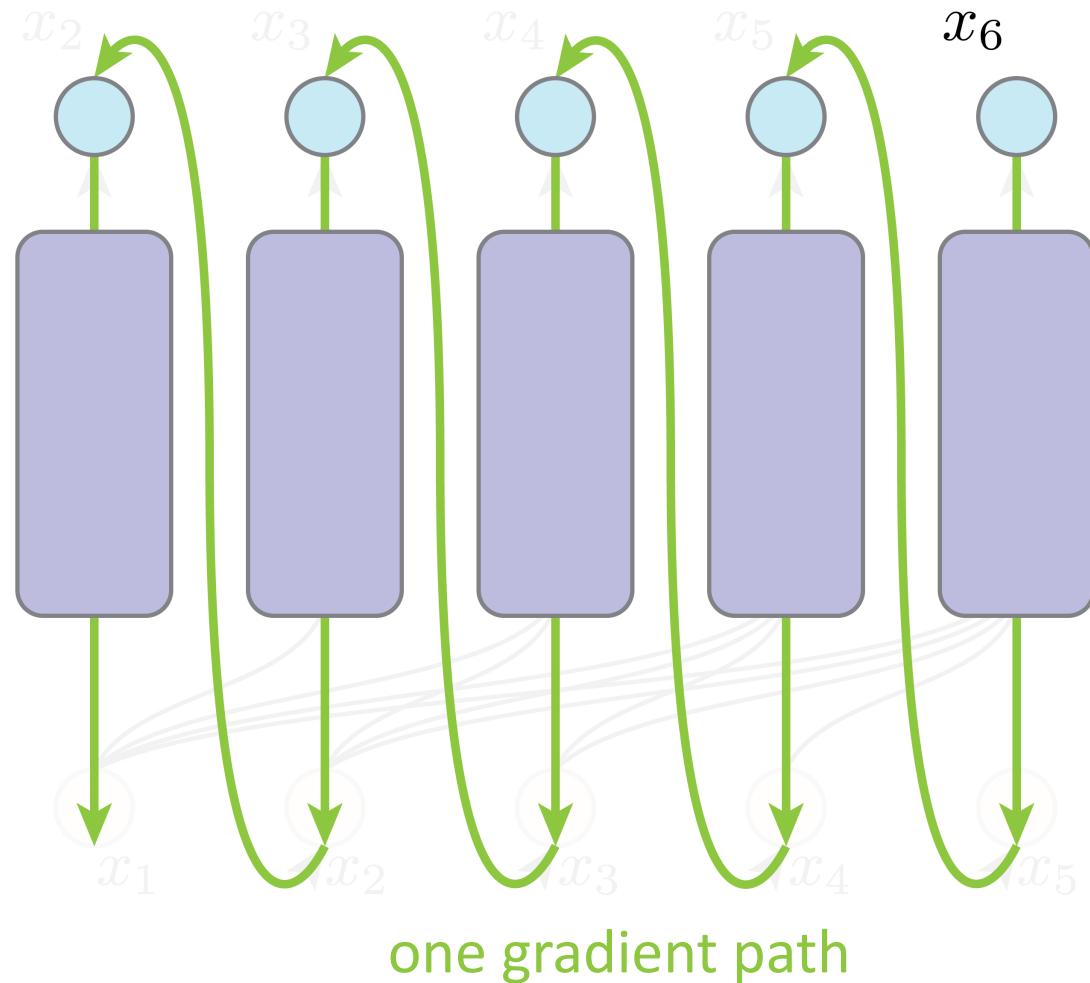
What if we backprop through this graph “as-is”?

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.



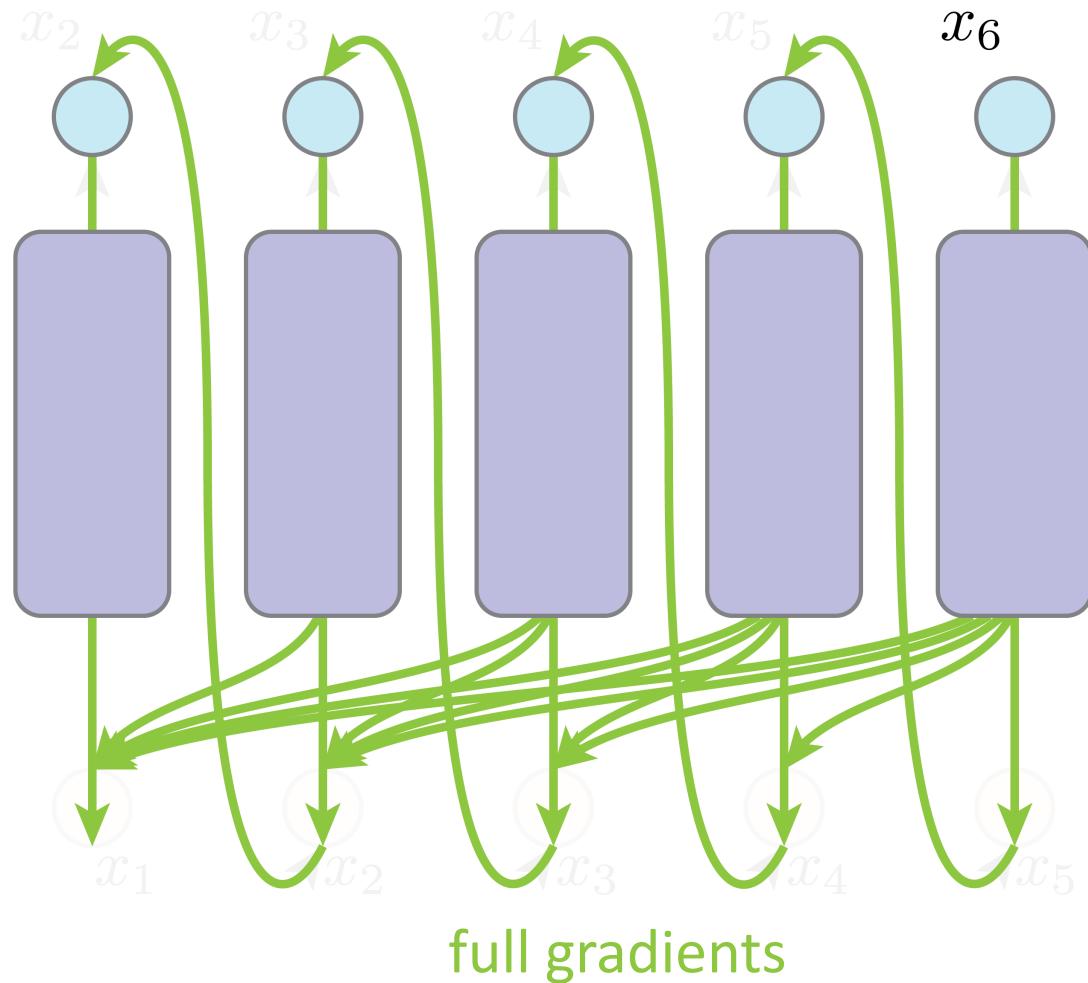
What if we backprop through this graph “as-is”?

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)

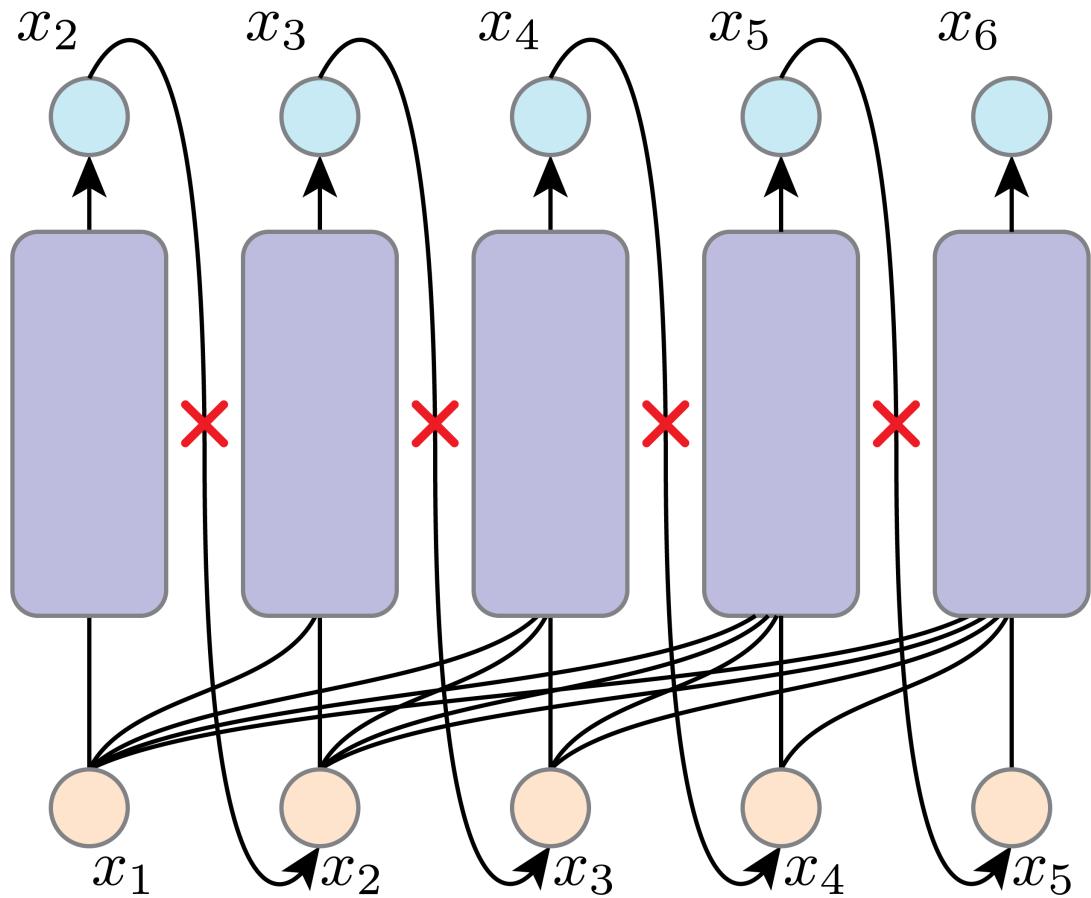
It's **infeasible** to train the AR model following its **inference graph**.



Training: Teacher-Forcing

Teacher-forcing

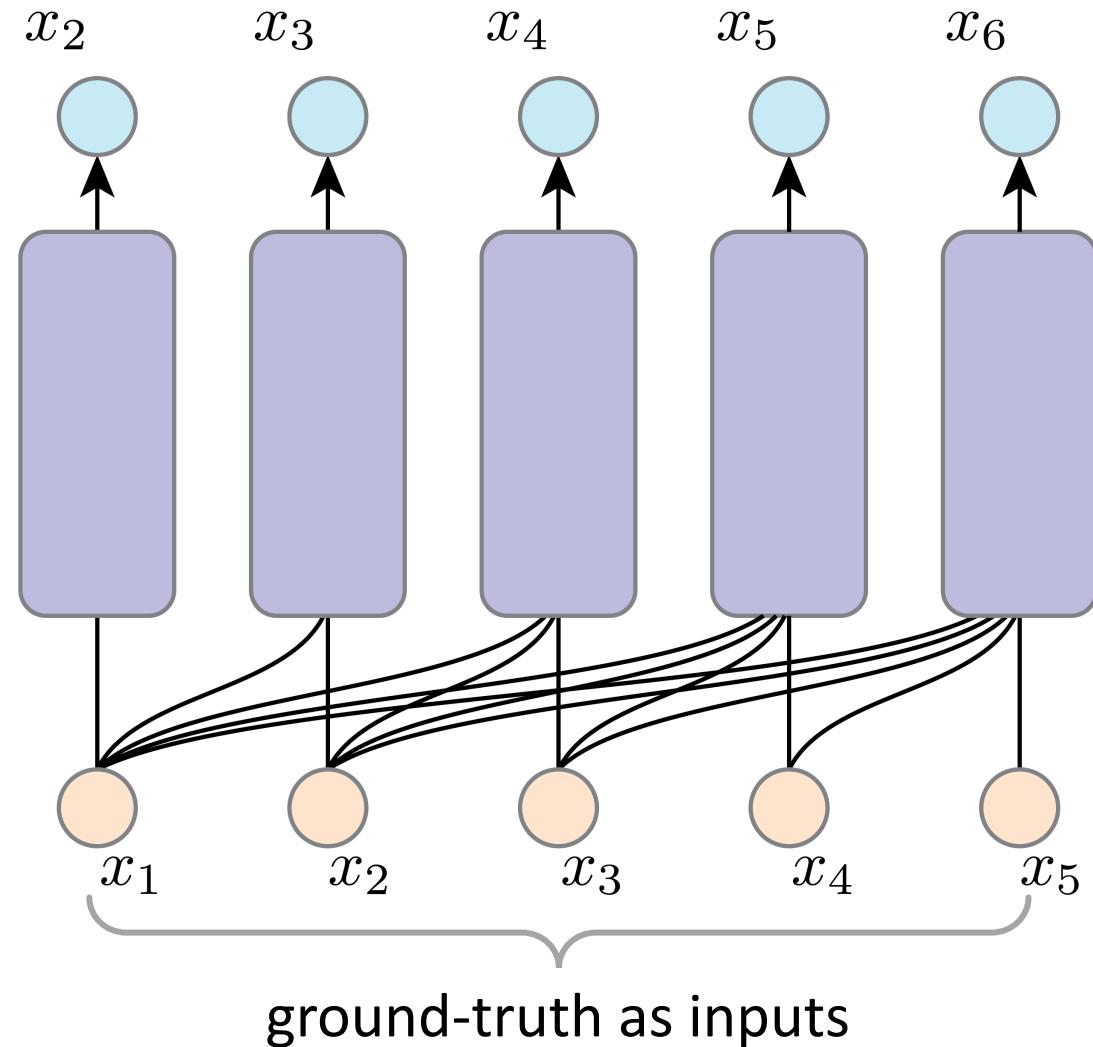
- Inputs are not from previous outputs
- Inputs are from ground-truth data



Training: Teacher-Forcing

Teacher-forcing

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



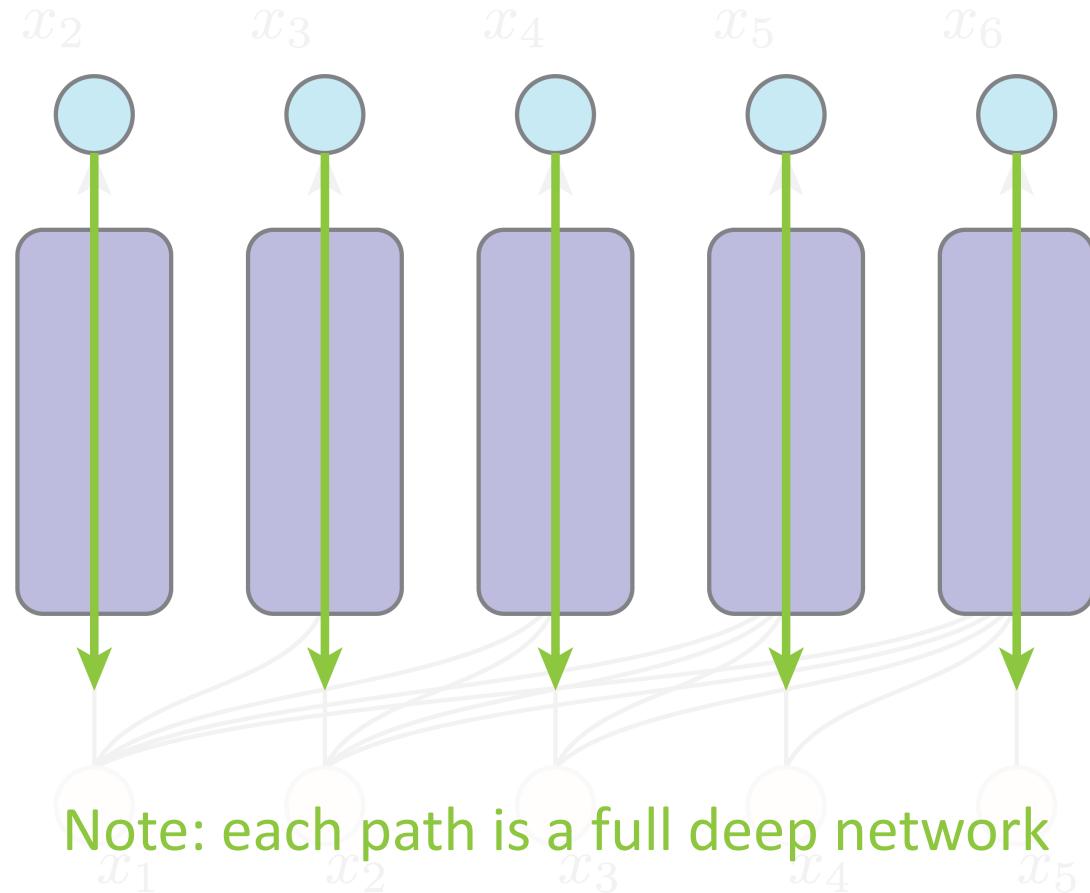
Training: Teacher-Forcing

Teacher-forcing

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

Pros:

- backprop path is much shorter



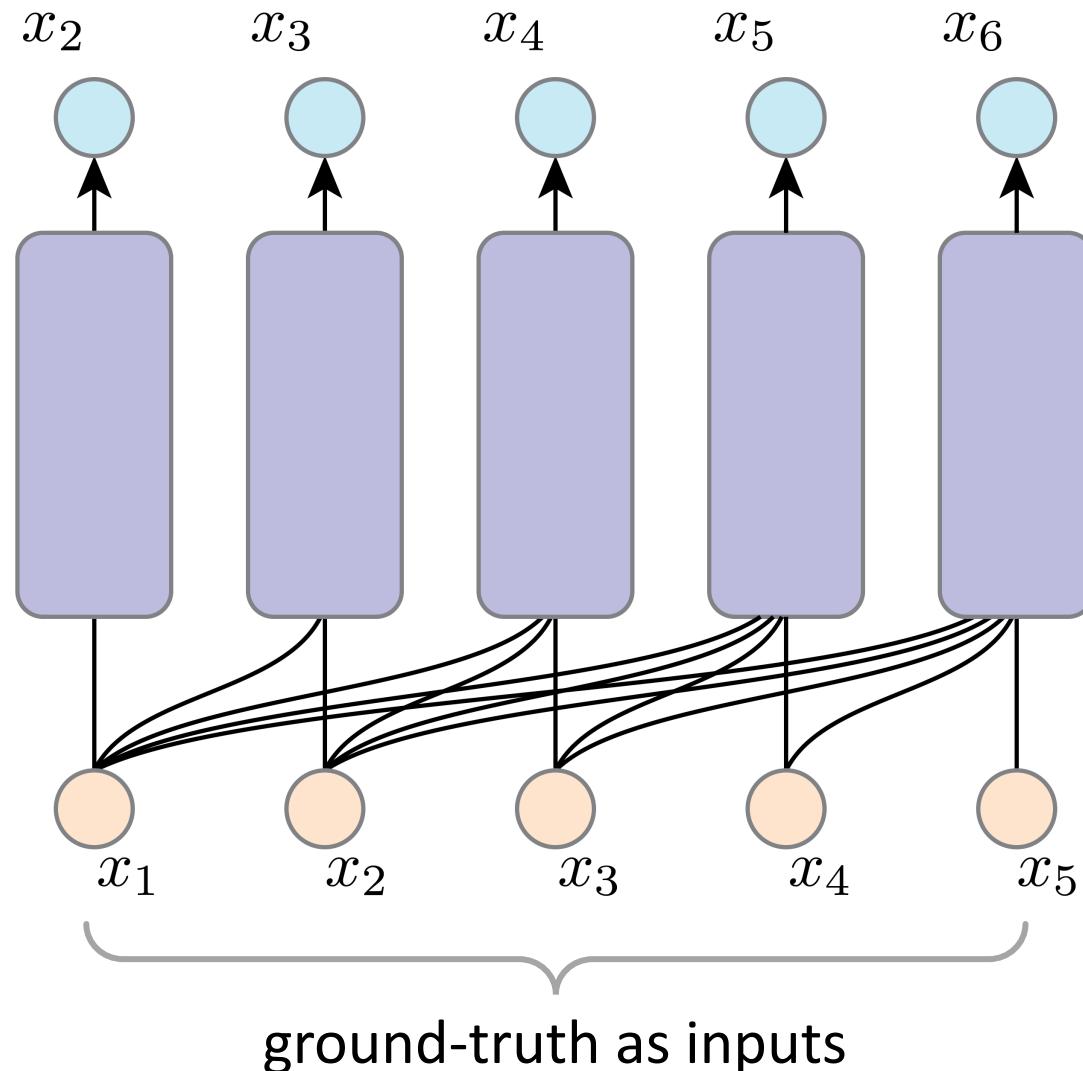
Training: Teacher-Forcing

Teacher-forcing

- Inputs are not 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

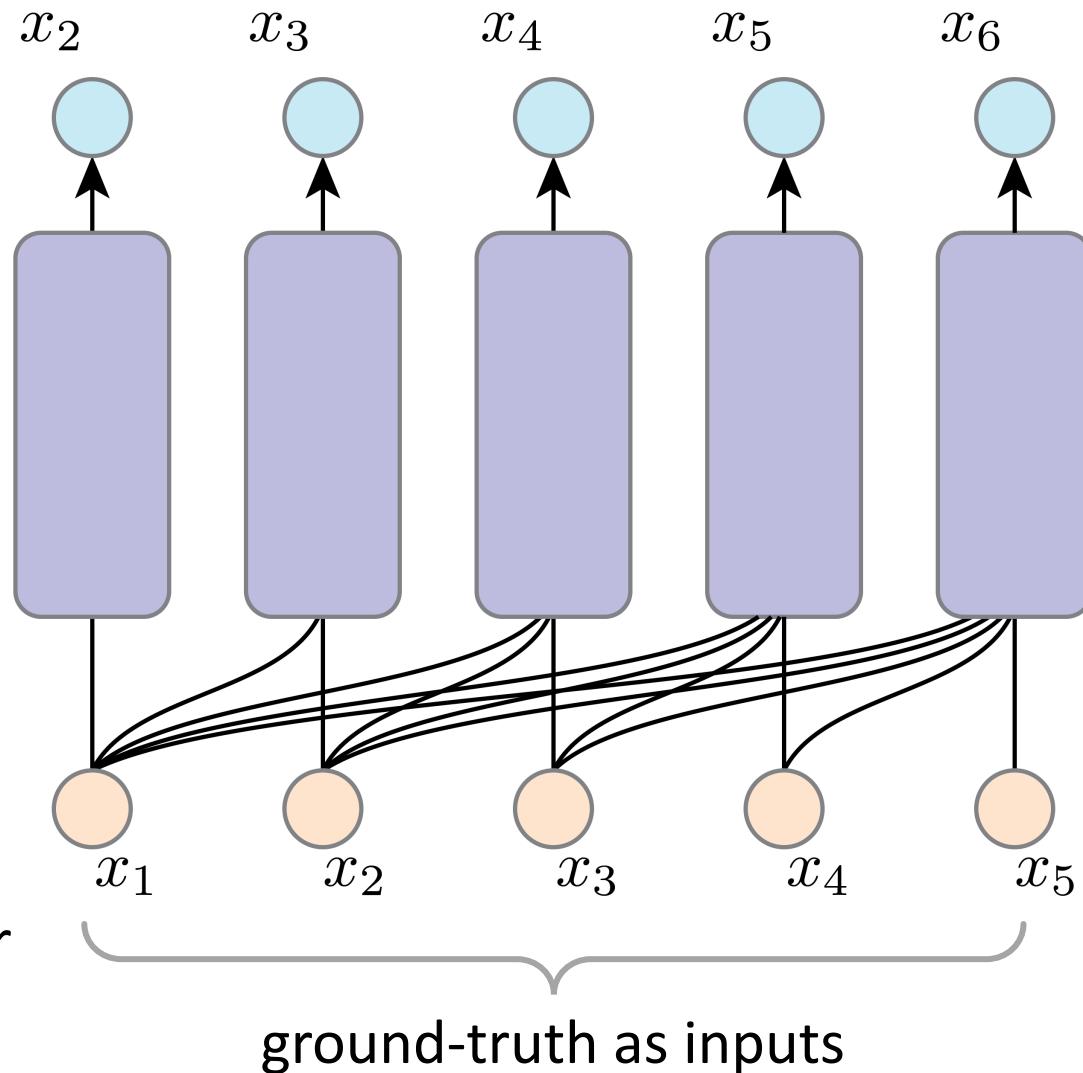
- Inputs are not from previous outputs
- Inputs are from ground-truth data

Pros:

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

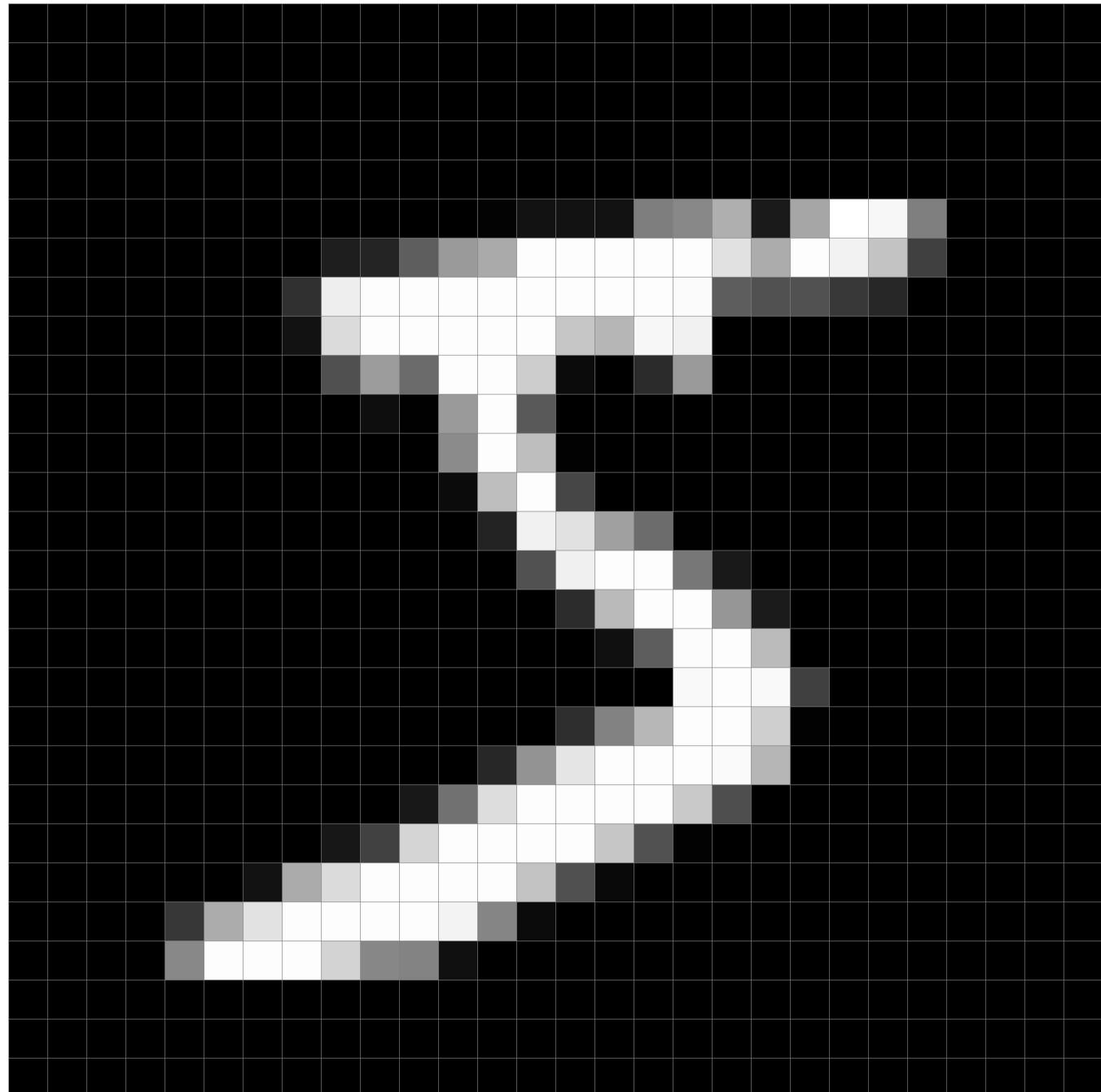
Cons:

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



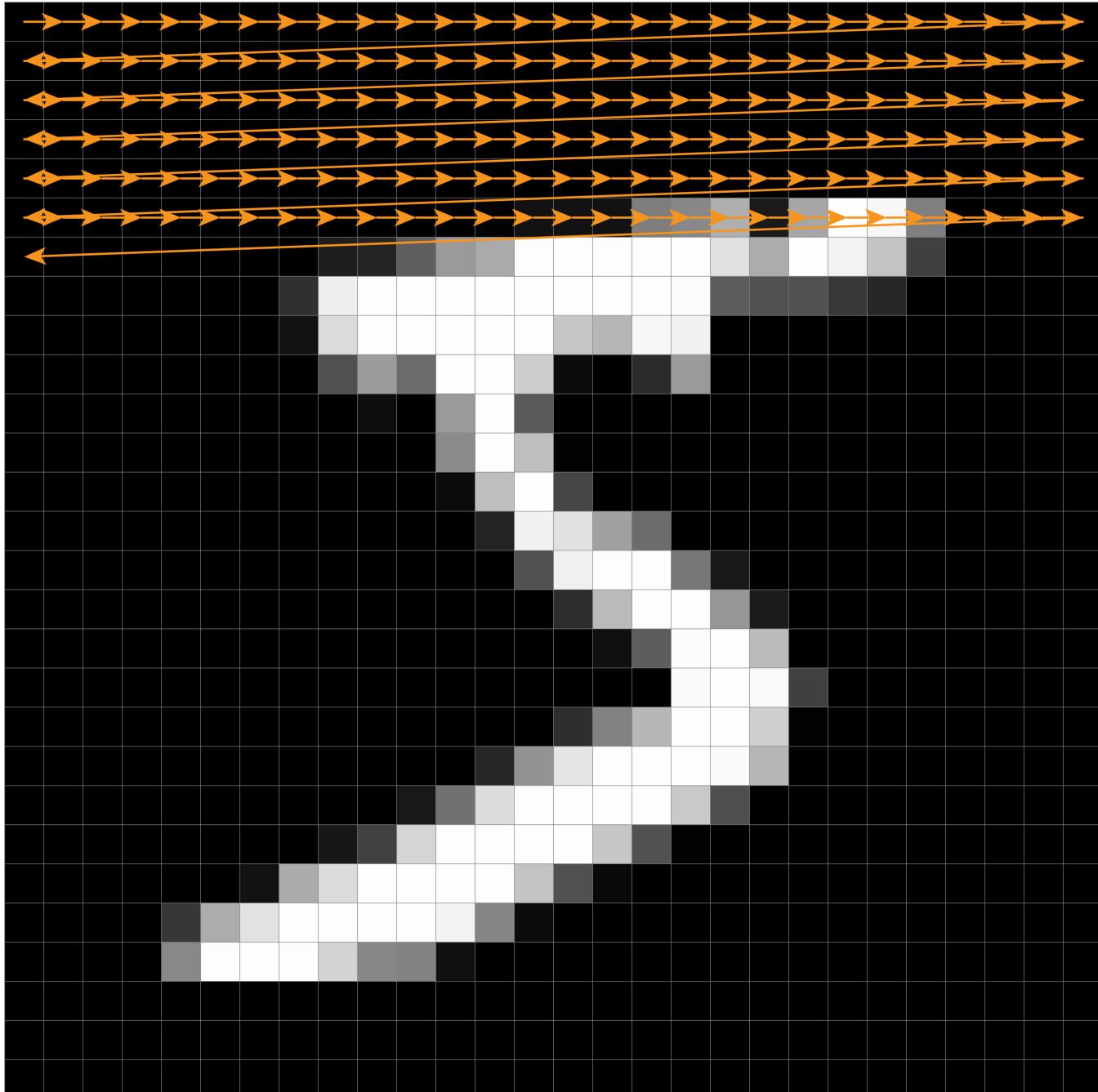
Running example: AR on MNIST

- an image as a sequence of pixels



Running example: AR on MNIST

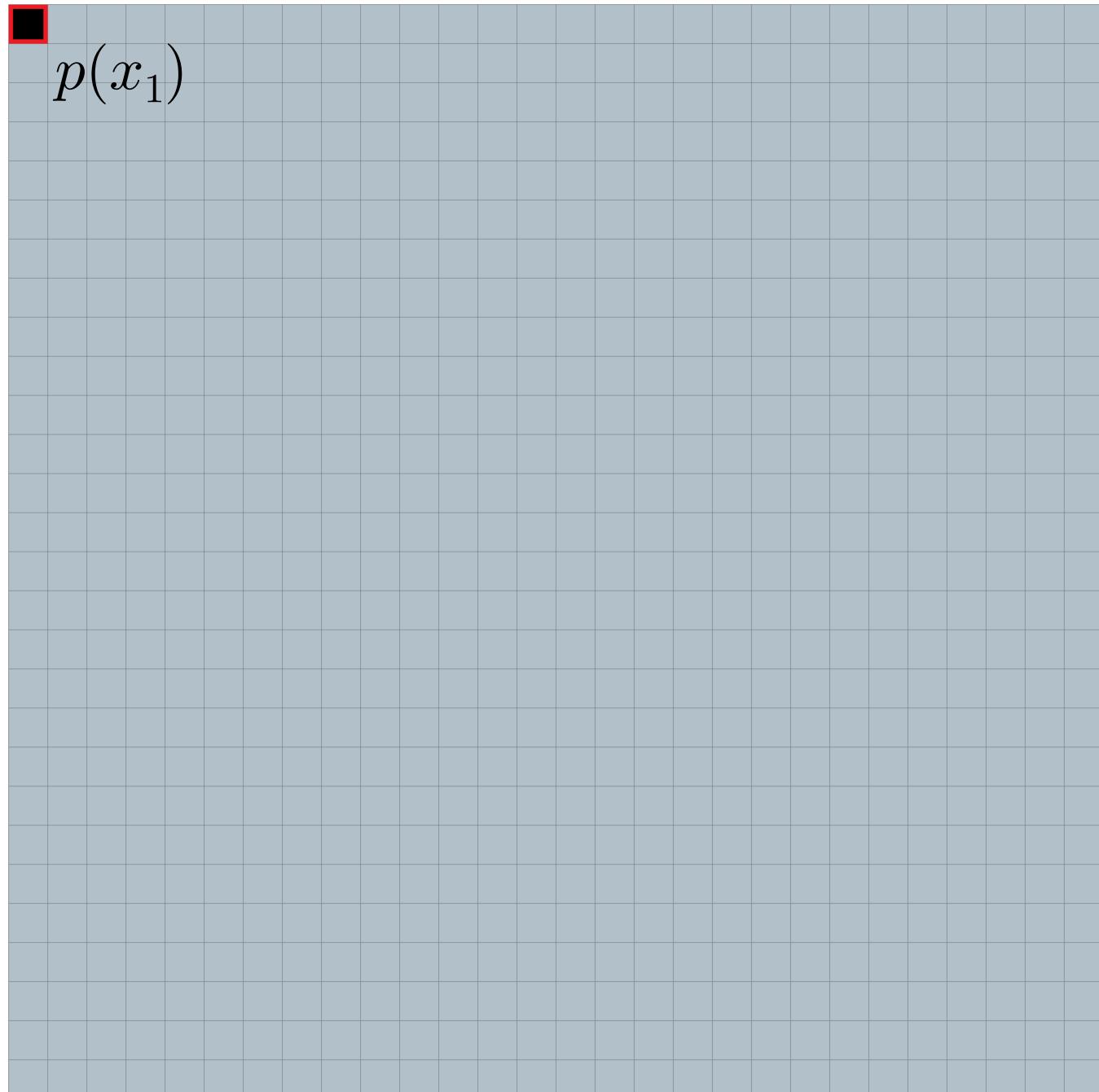
- an image as a sequence of pixels
- scan by **raster order**



Running example: AR on MNIST

Inference: Autoregressive

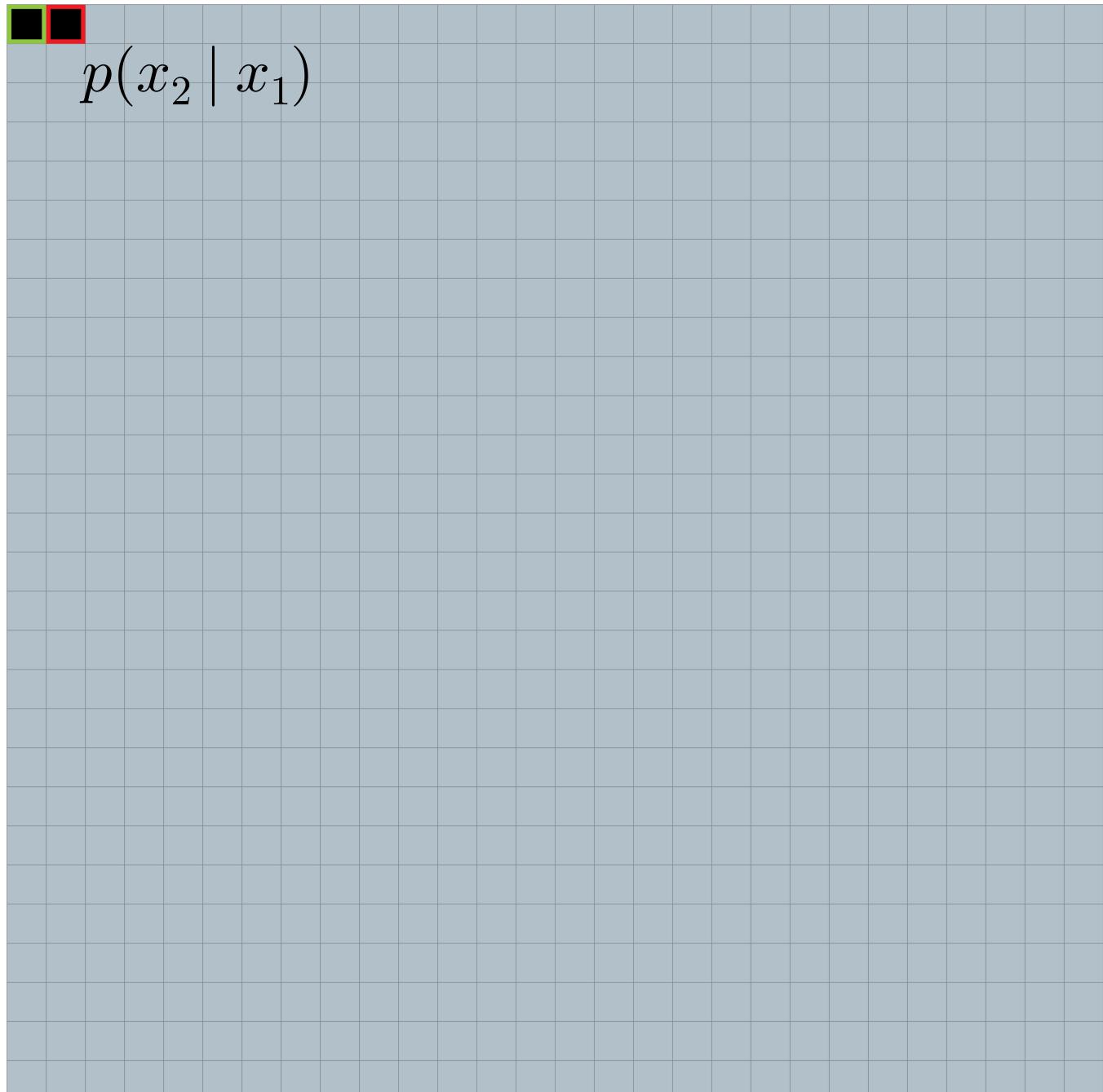
- sample **this** pixel from $p(x_1)$



Running example: AR on MNIST

Inference: Autoregressive

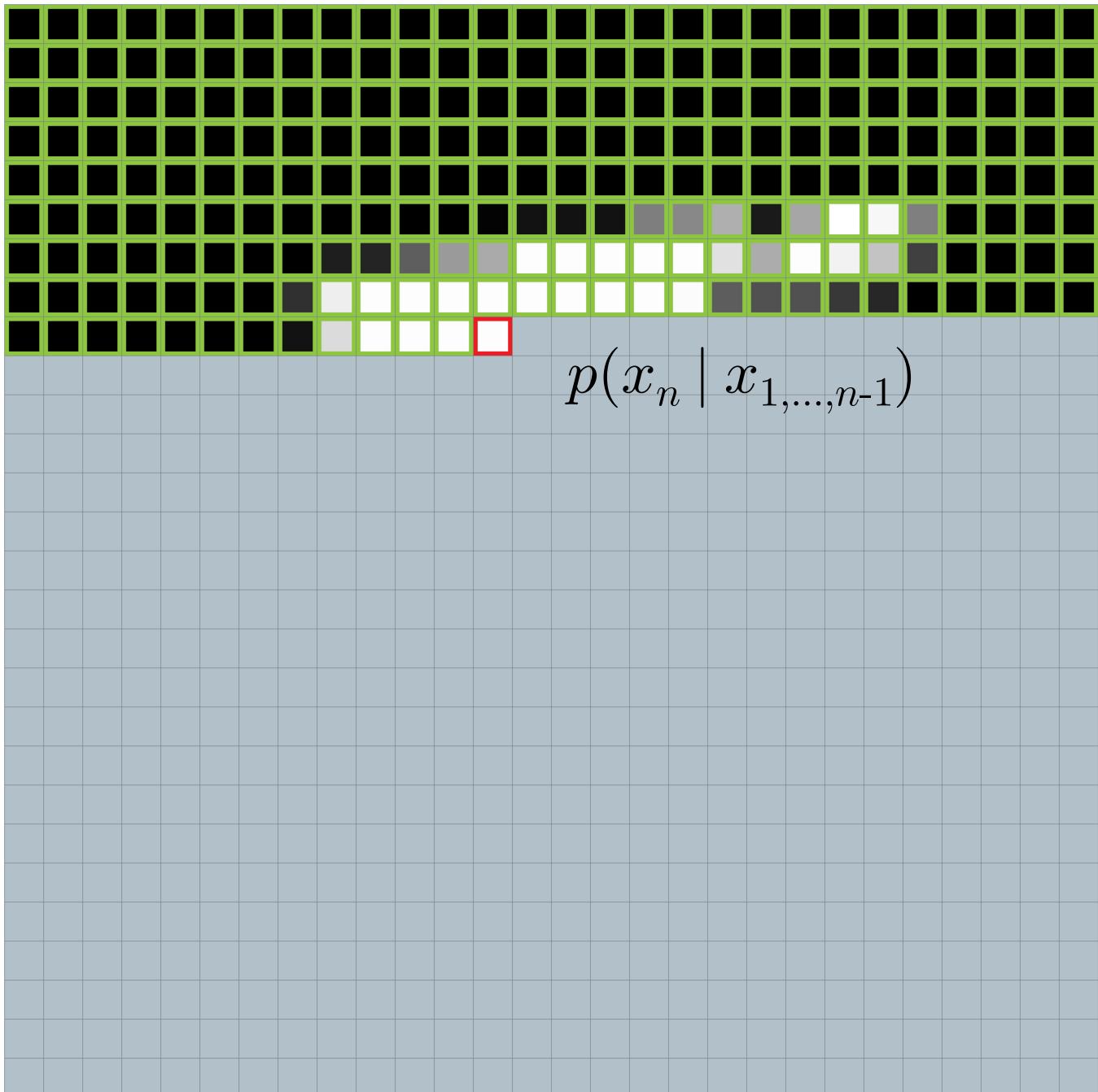
- sample **this** pixel from $p(x_2 | x_1)$
- **this** is output from previous step,
input for current step
- network for this step:
 - 1 **input**
 - 1 **predict**



Running example: AR on MNIST

Inference: Autoregressive

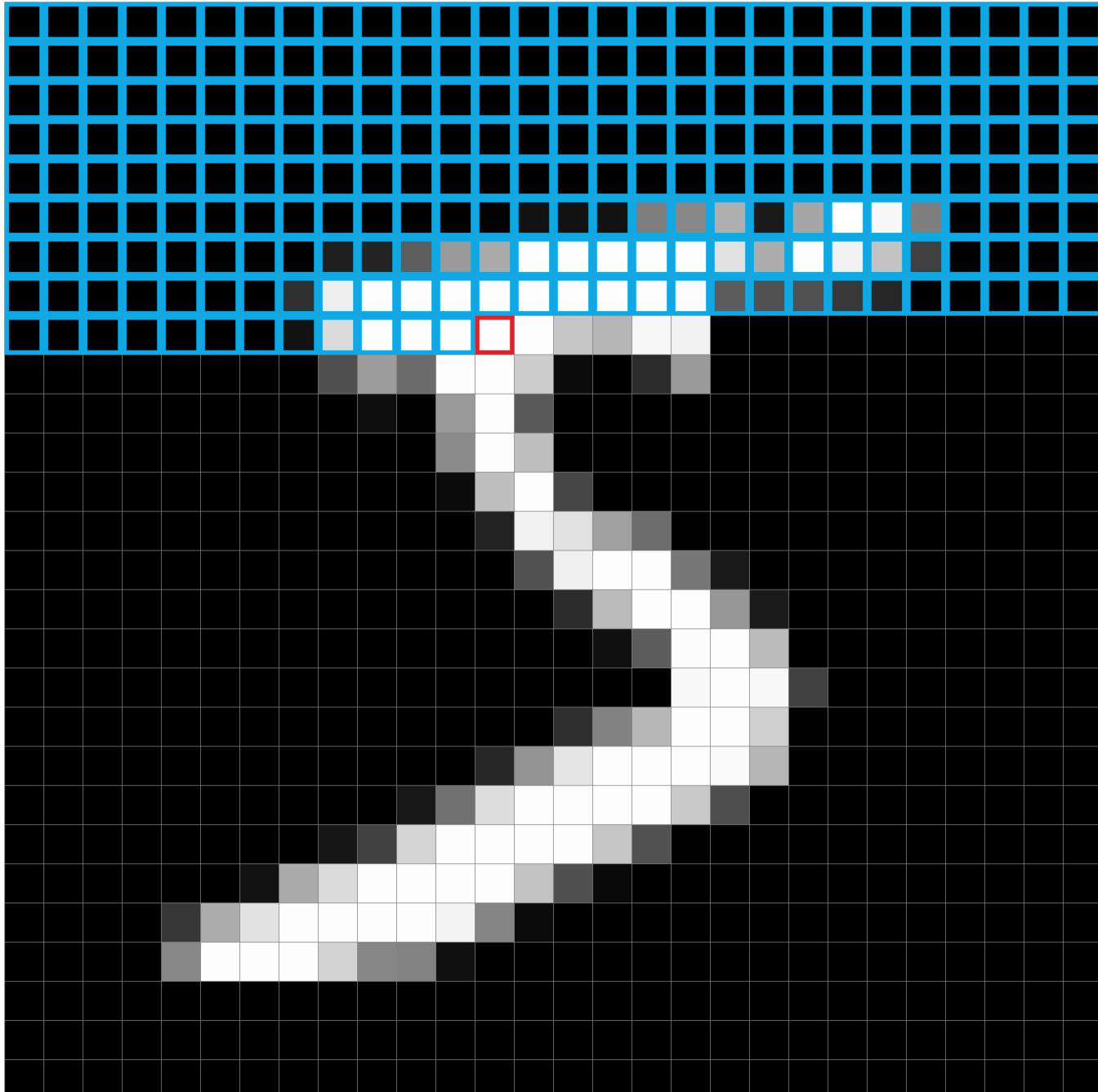
- sample **this** pixel from $p(x_n | x_{1,\dots,n-1})$
- **these** are outputs from previous steps,
inputs for current step
- network for this step:
 - $(n - 1)$ **inputs**
 - 1 **predict**



Running example: AR on MNIST

Training: Teacher-Forcing

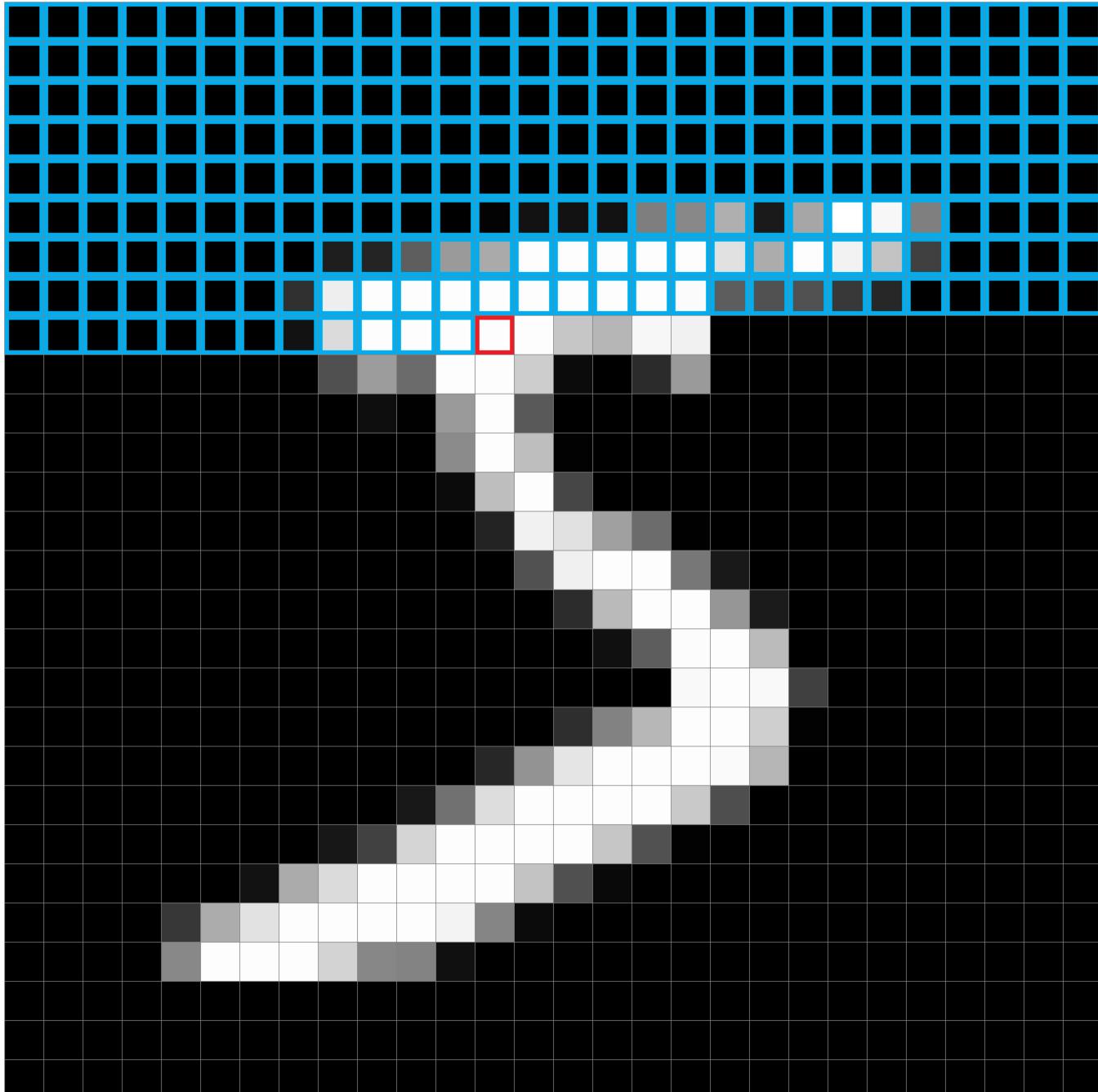
- model **this** pixel by: $p(x_n \mid x_{1,\dots,n-1})$
- **these** are outputs from ground-truth,
inputs for current step
- network for this step:
 - $(n - 1)$ **inputs**
 - 1 **predict**



Running example: AR on MNIST

Note:

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



Autoregressive Models

Summary:

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

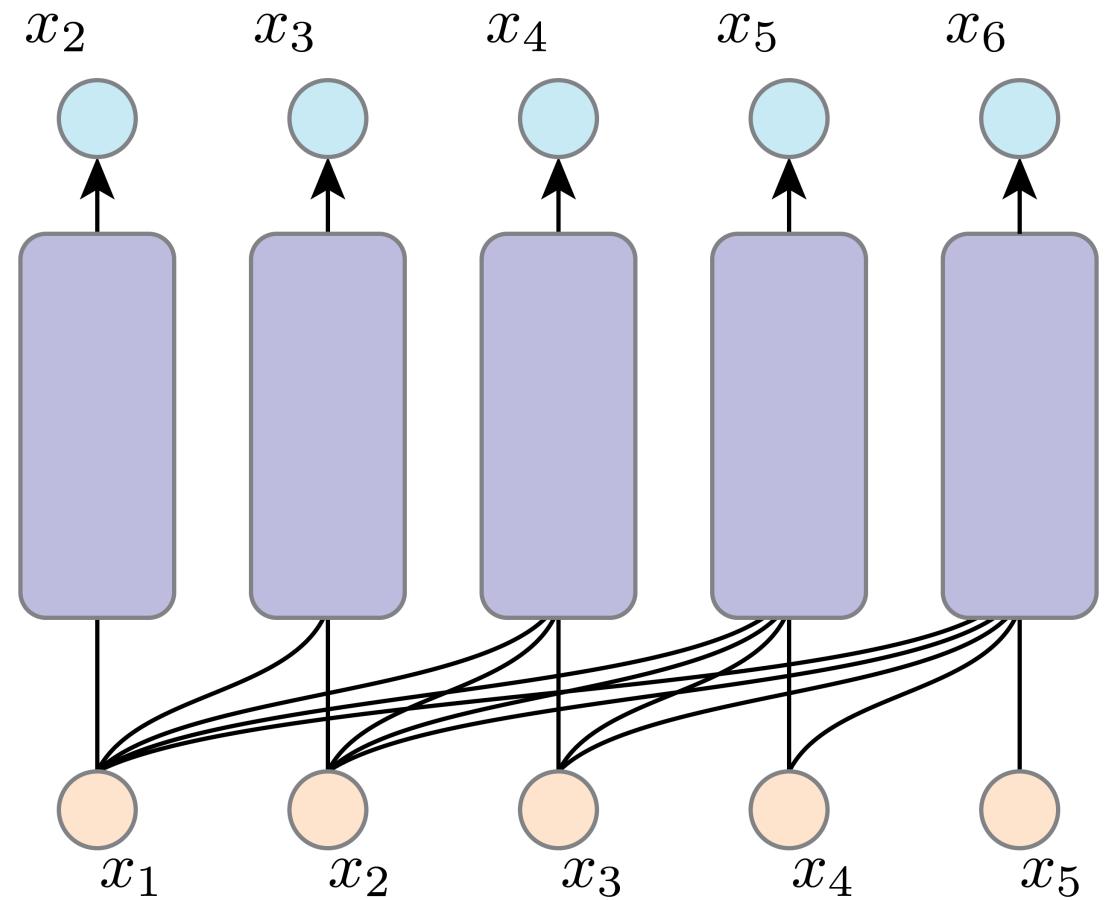
These are not specific to a certain type of network architectures.

Network Architectures for Autoregressive Modeling

Autoregression is not architecture-specific

This figure implements this formulation:

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



(showing training case for simplicity)

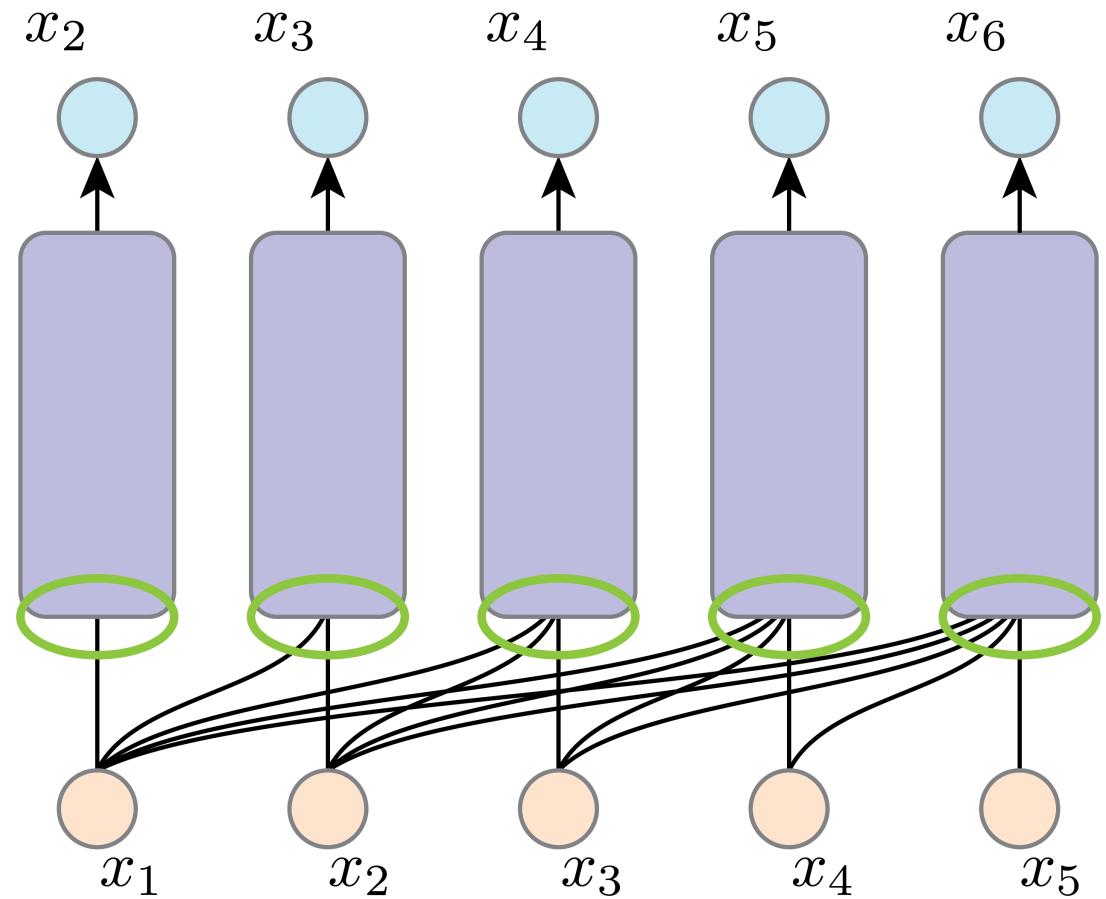
Autoregression is not architecture-specific

This figure implements this formulation:

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

In this example:

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

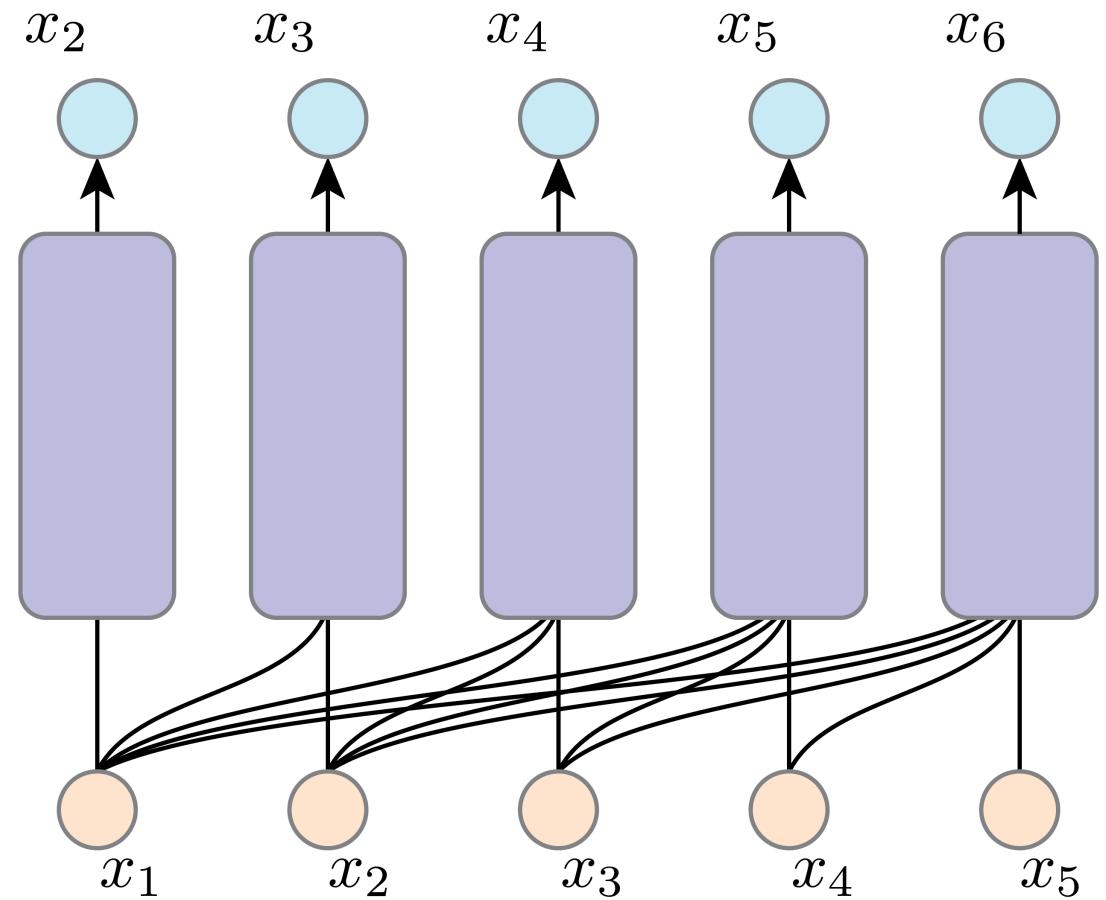


Autoregression is not architecture-specific

This figure implements this formulation:

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

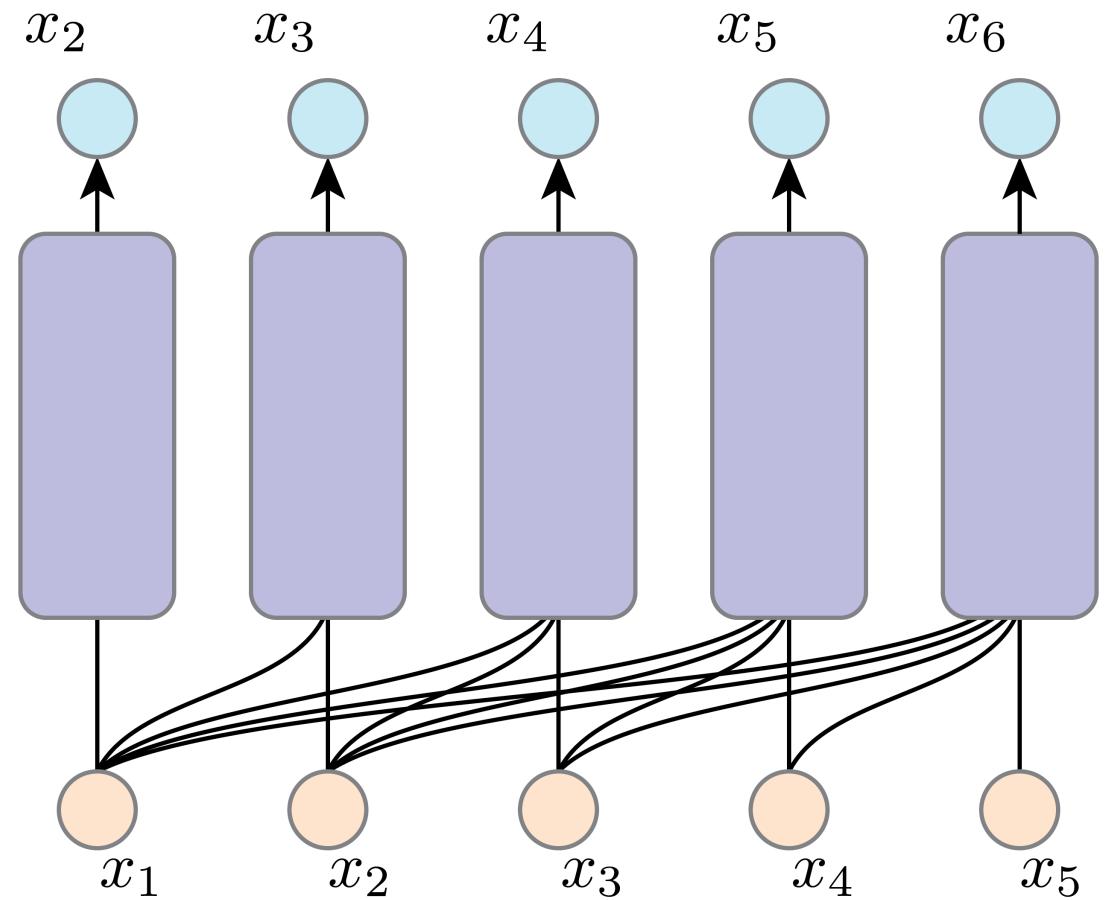
Can we do this efficiently?



Autoregression w/ Shared Computation

This figure implements this formulation:

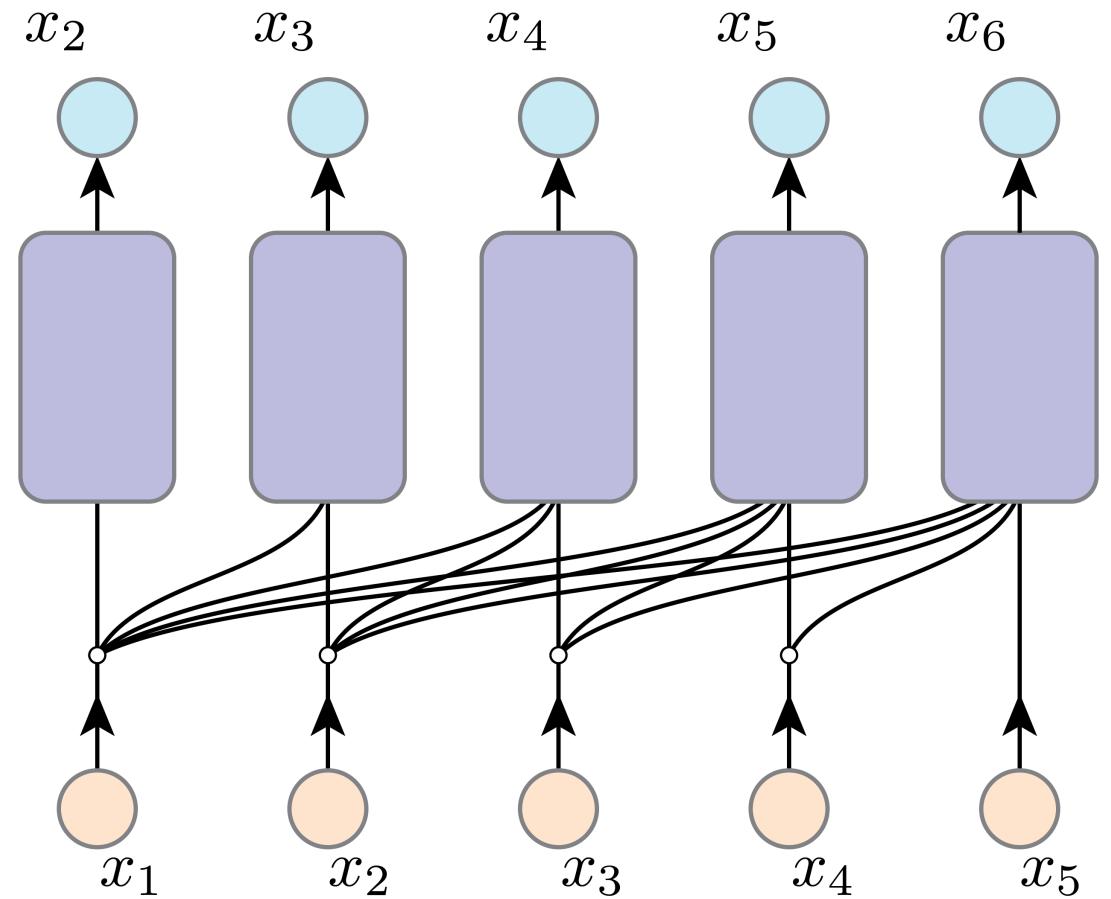
$$p(x_1, x_2, \dots, x_n) = \prod_{i=1}^n p(x_i | x_1, x_2, \dots, x_{i-1})$$



Autoregression w/ Shared Computation

This figure implements this formulation:

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

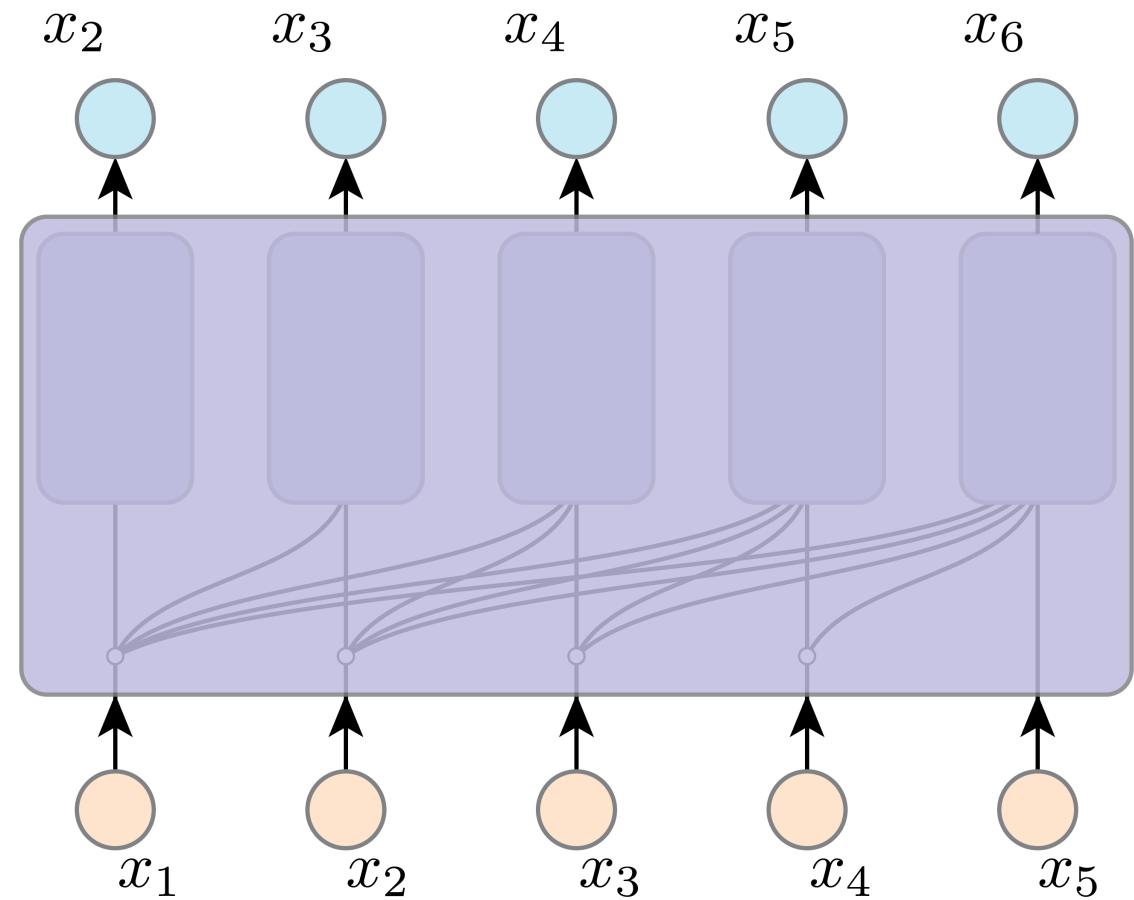


(this figure is equivalent to previous one)

Autoregression w/ Shared Computation

This figure implements this formulation:

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



Autoregression w/ Shared Computation

We can implement:

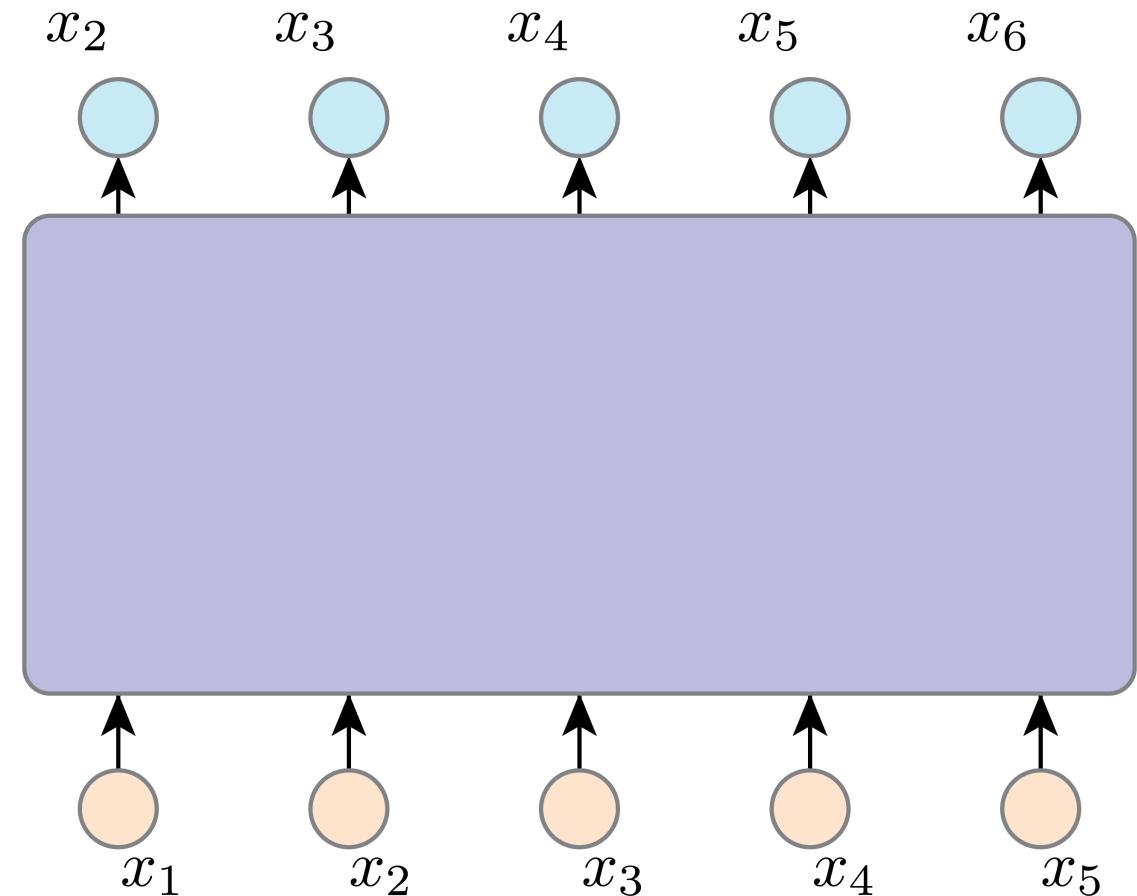
$$p(x_1, x_2, \dots, x_n) = \prod_{i=1}^n p(x_i | x_1, x_2, \dots, x_{i-1})$$

... 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:

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

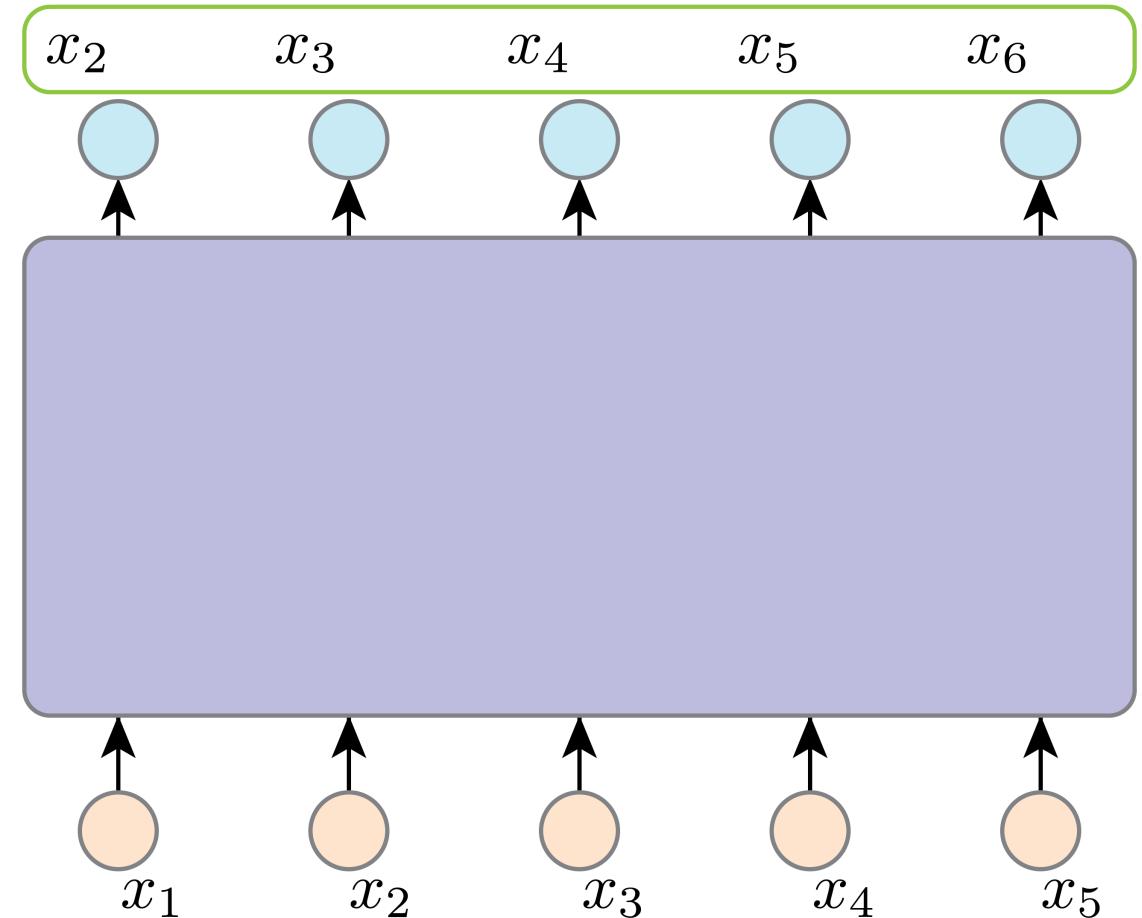
... by one network, with:

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

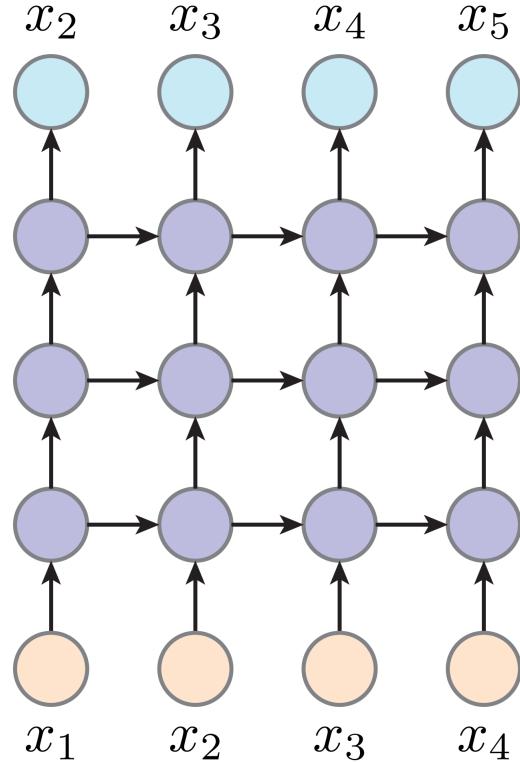
if:

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

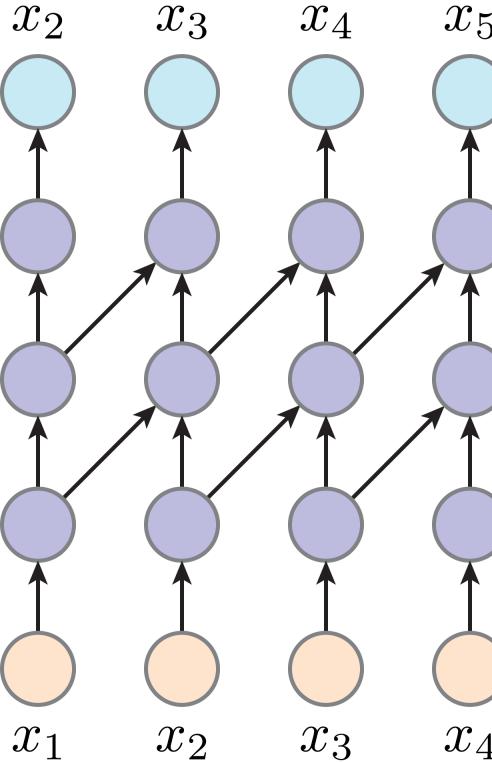
targets: shifted by one step



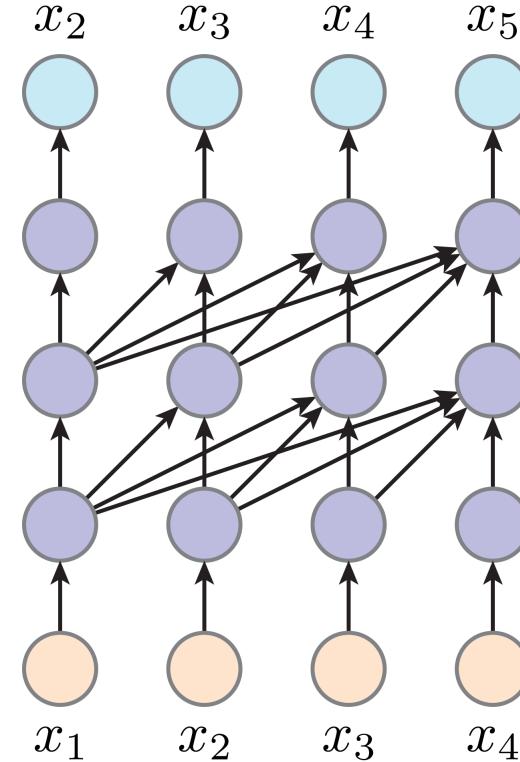
Common Architectures for Autoregression



RNN



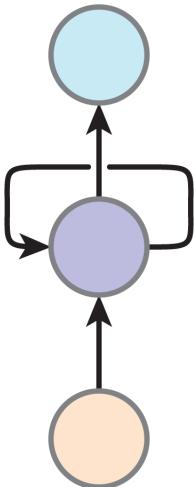
CNN



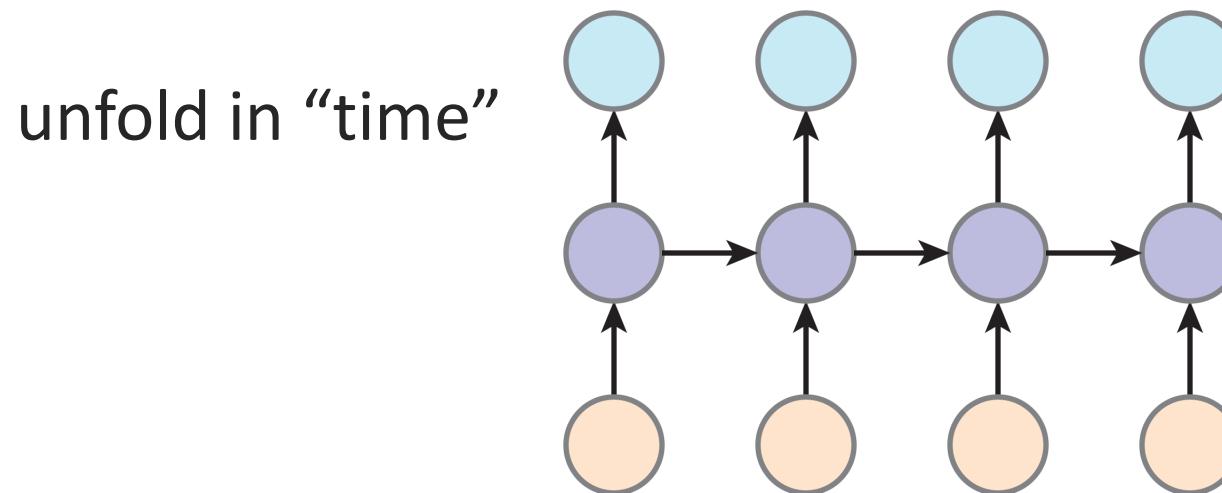
Attention

Brief: Recurrent Neural Network (RNN) for AR

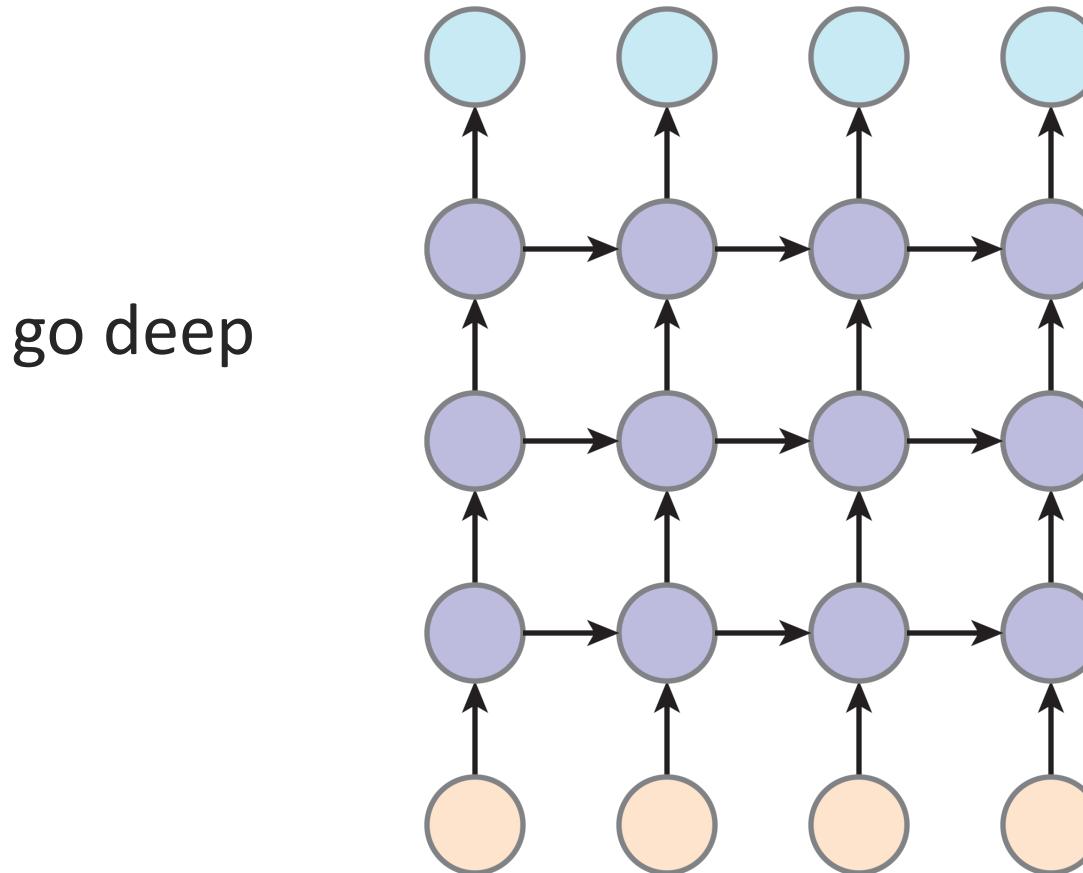
one RNN unit



Brief: Recurrent Neural Network (RNN) for AR

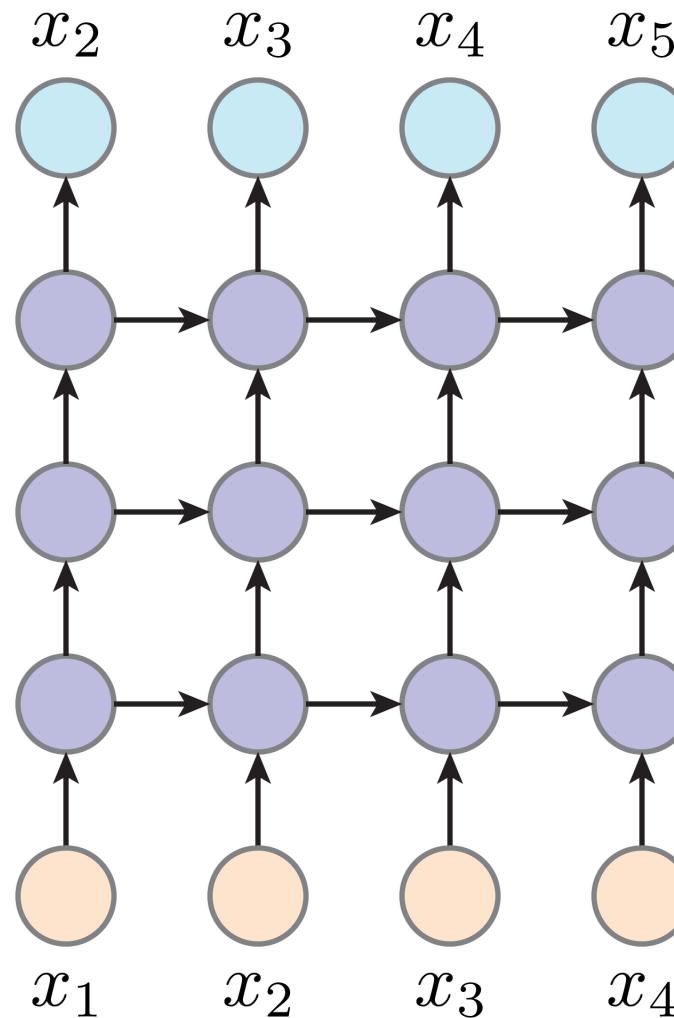


Brief: Recurrent Neural Network (RNN) for AR

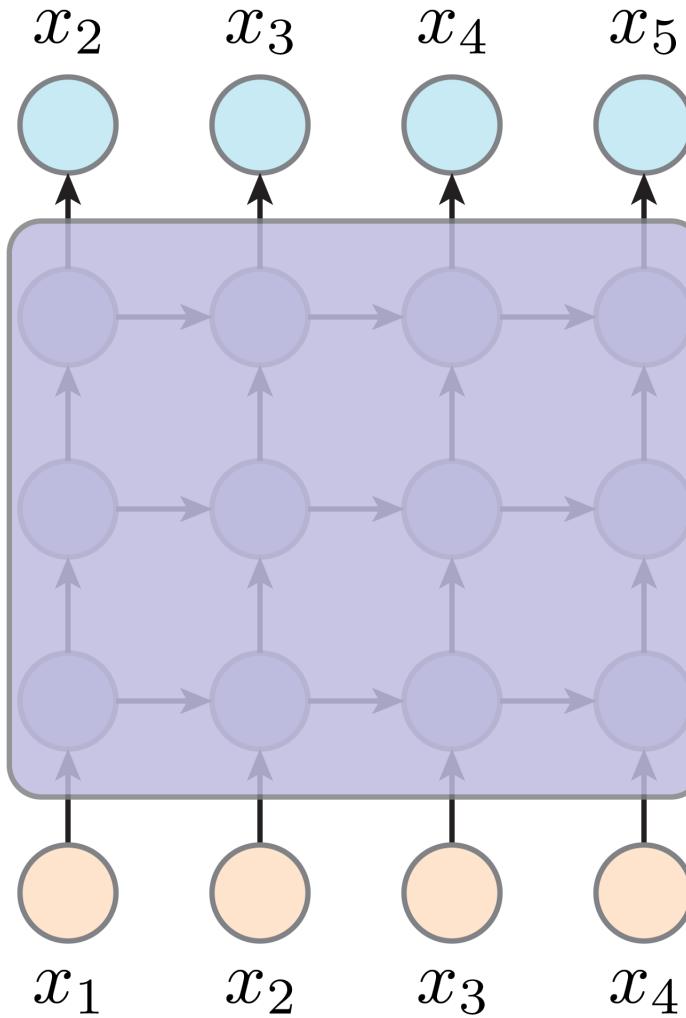


Brief: Recurrent Neural Network (RNN) for AR

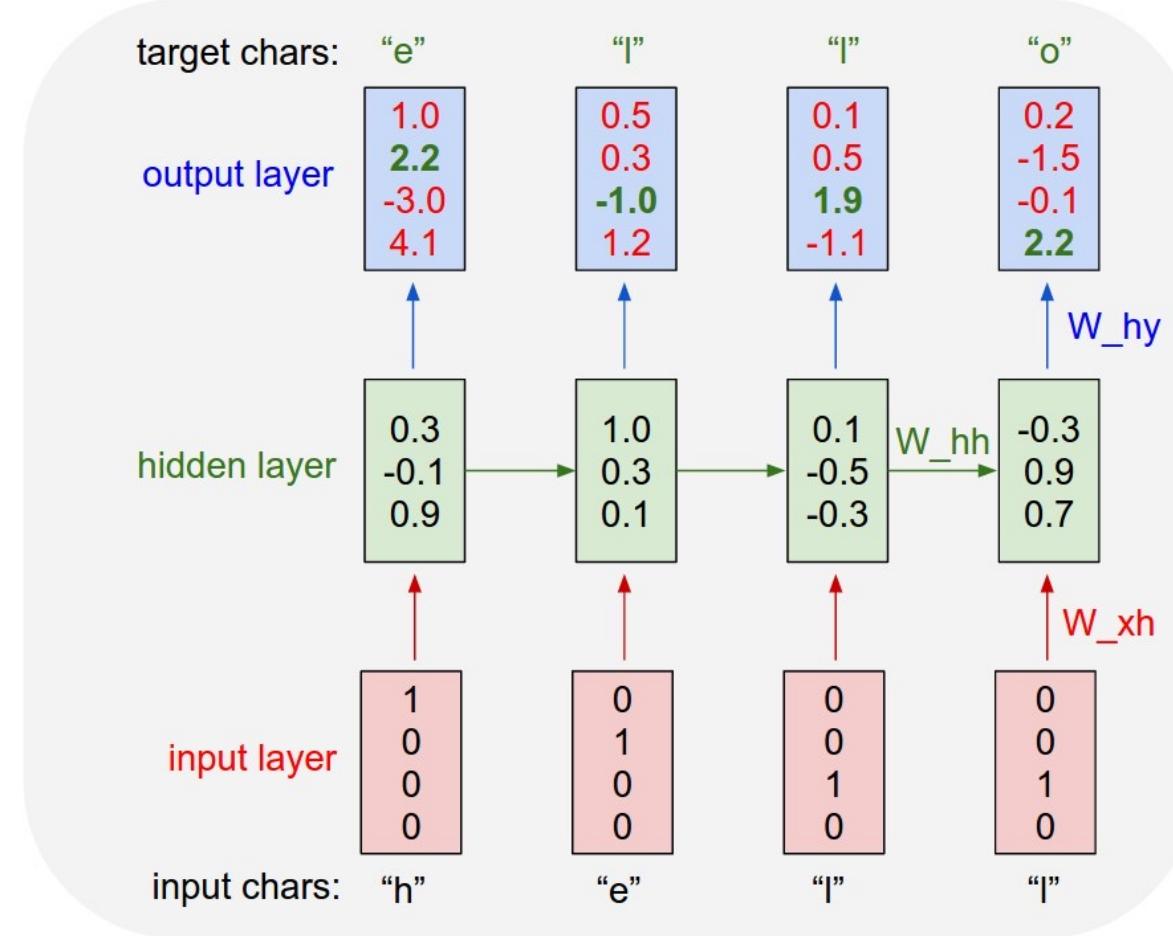
shift target by one step



Brief: Recurrent Neural Network (RNN) for AR

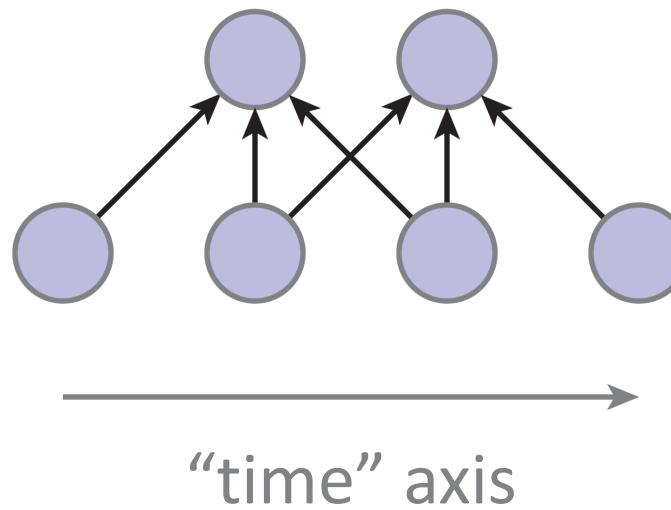


Example: Char-RNN

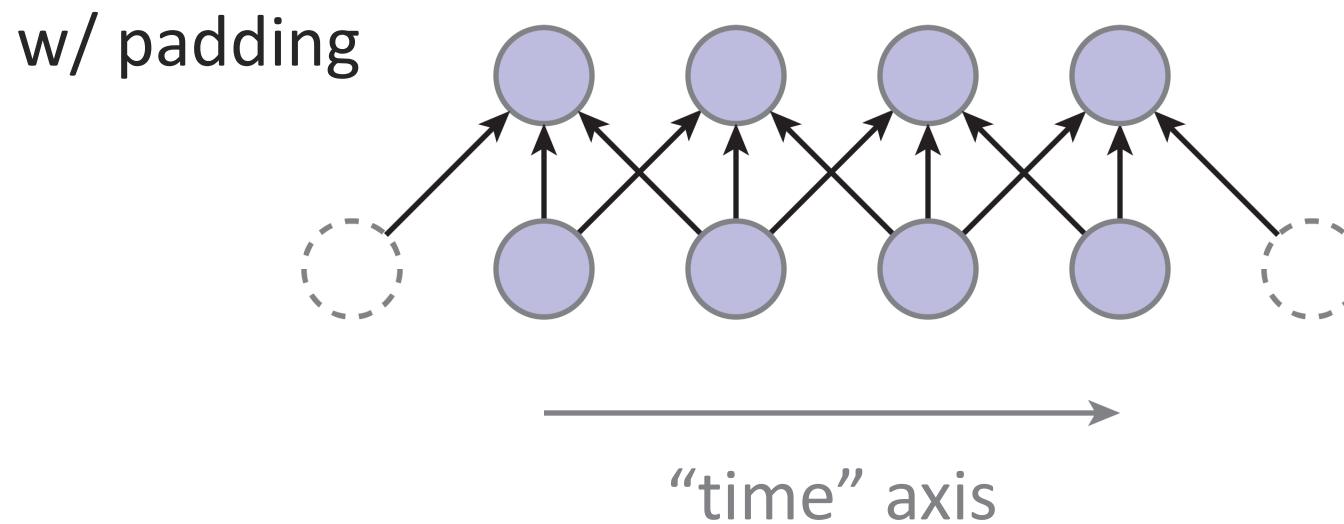


Brief: Convolutional Neural Network (CNN) for AR

1-D convolution

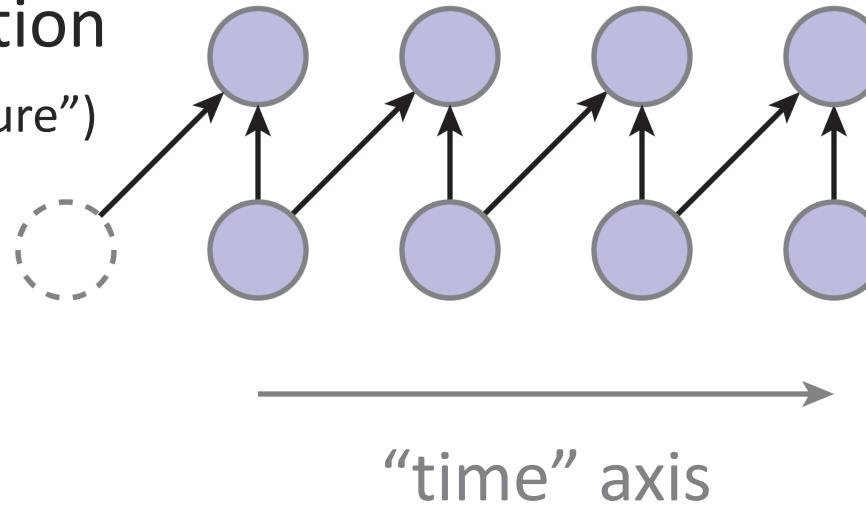


Brief: Convolutional Neural Network (CNN) for AR

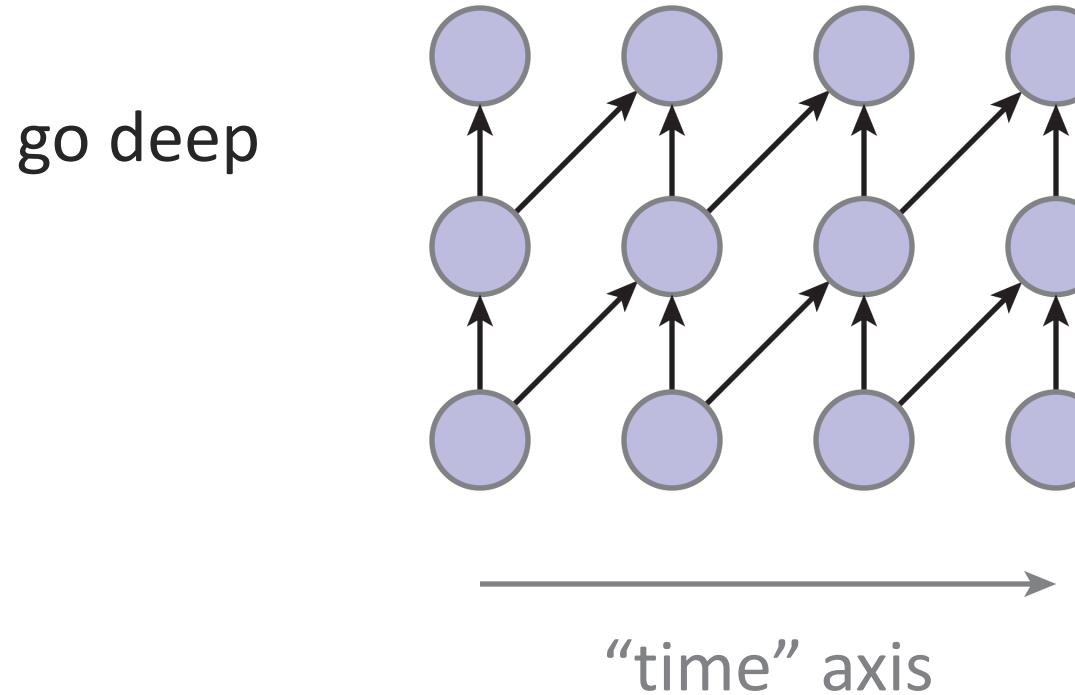


Brief: Convolutional Neural Network (CNN) for AR

causal convolution
(not depend on “future”)

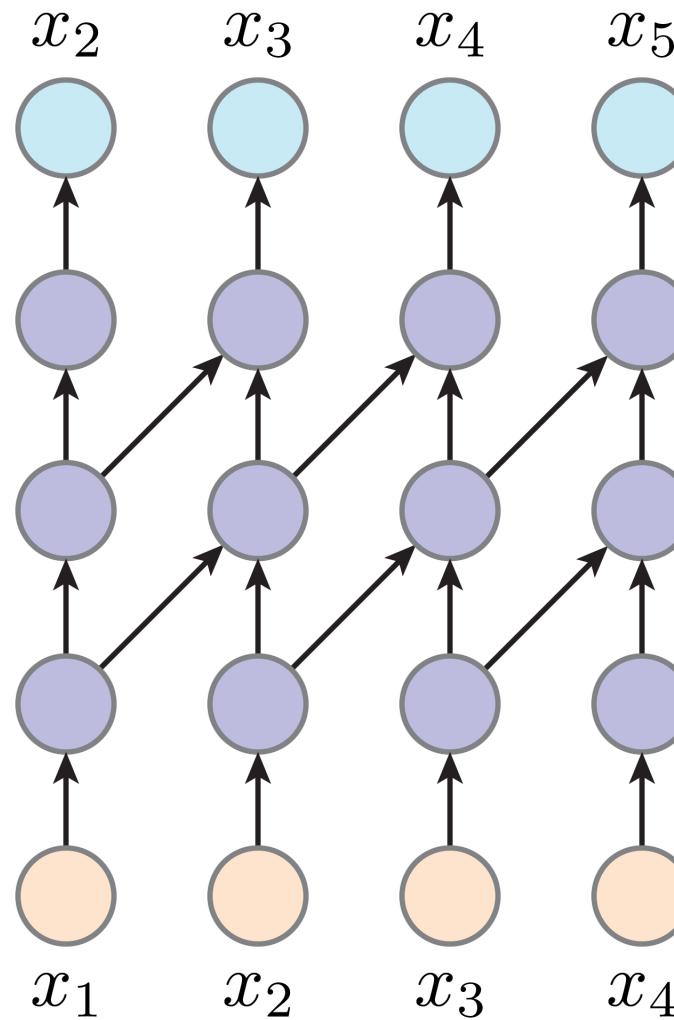


Brief: Convolutional Neural Network (CNN) for AR

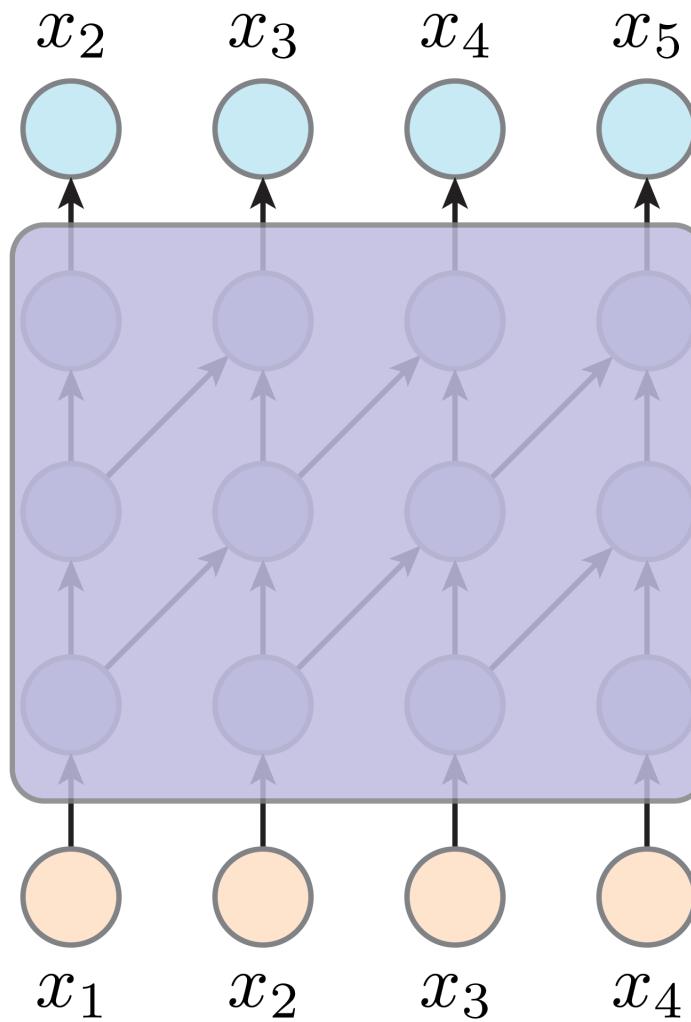


Brief: Convolutional Neural Network (CNN) for AR

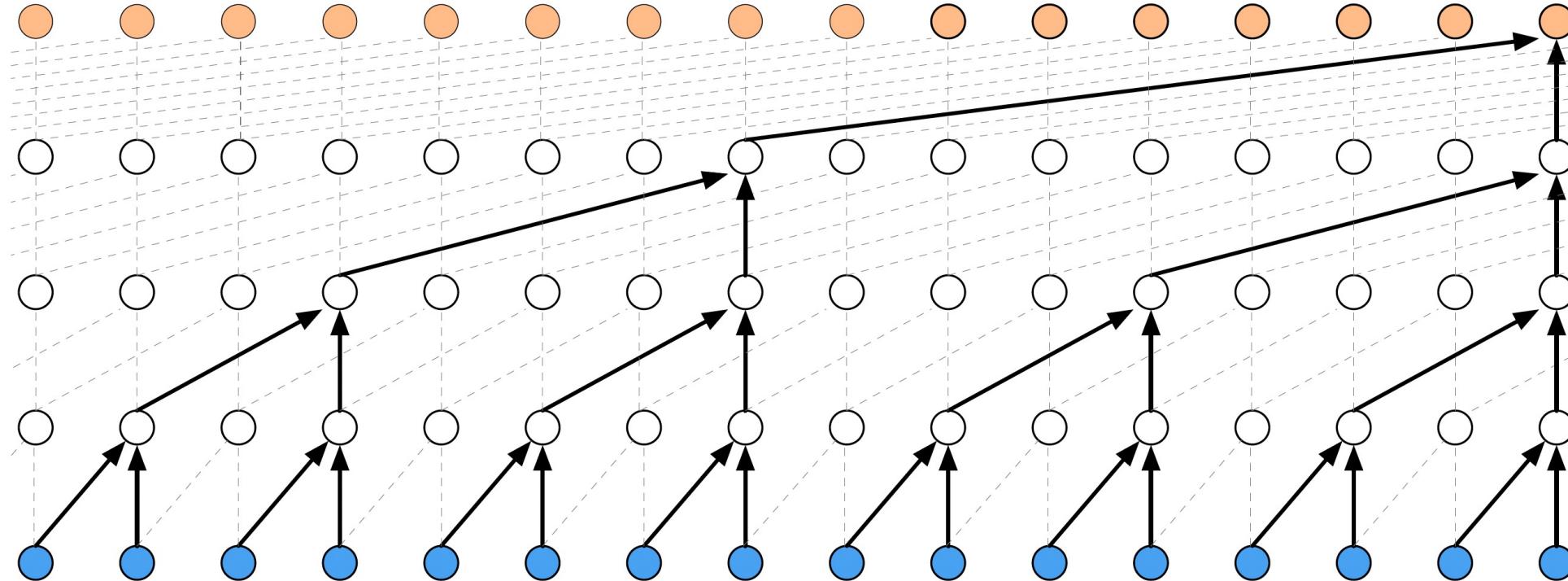
shift target by one step



Brief: Convolutional Neural Network (CNN) for AR



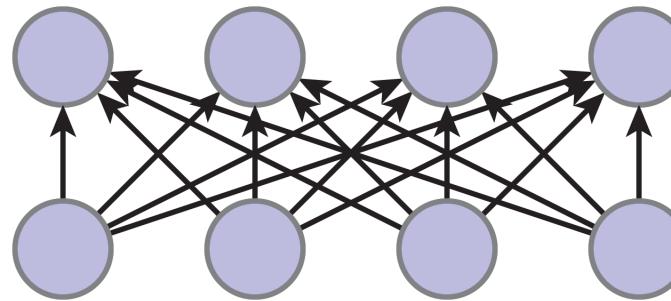
Example: WaveNet



Audio generation with 1-D dilated causal conv

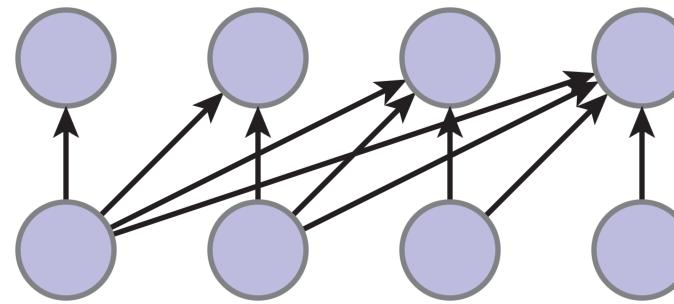
Brief: Attention (Transformer) for AR

full attention
(every step sees all steps)

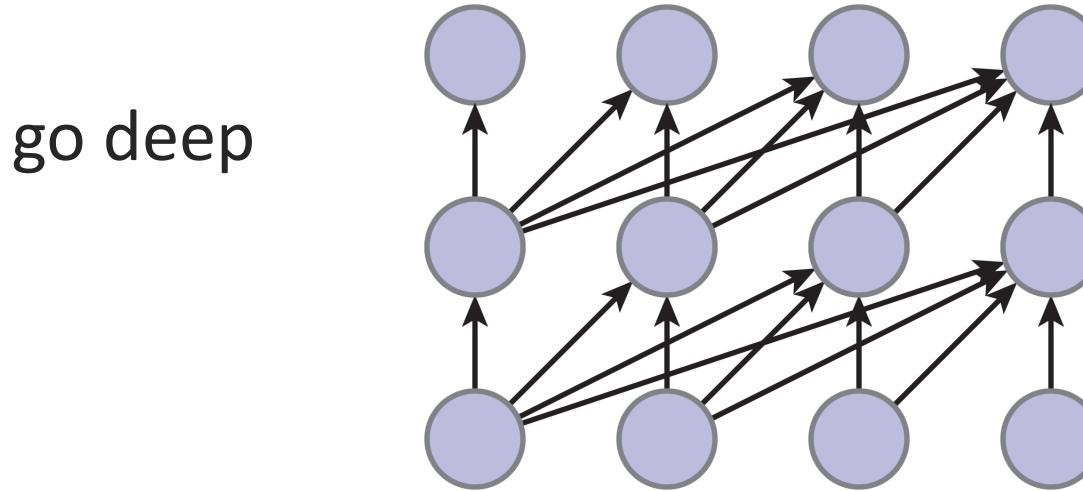


Brief: Attention (Transformer) for AR

causal attention
(not depend on “future”)

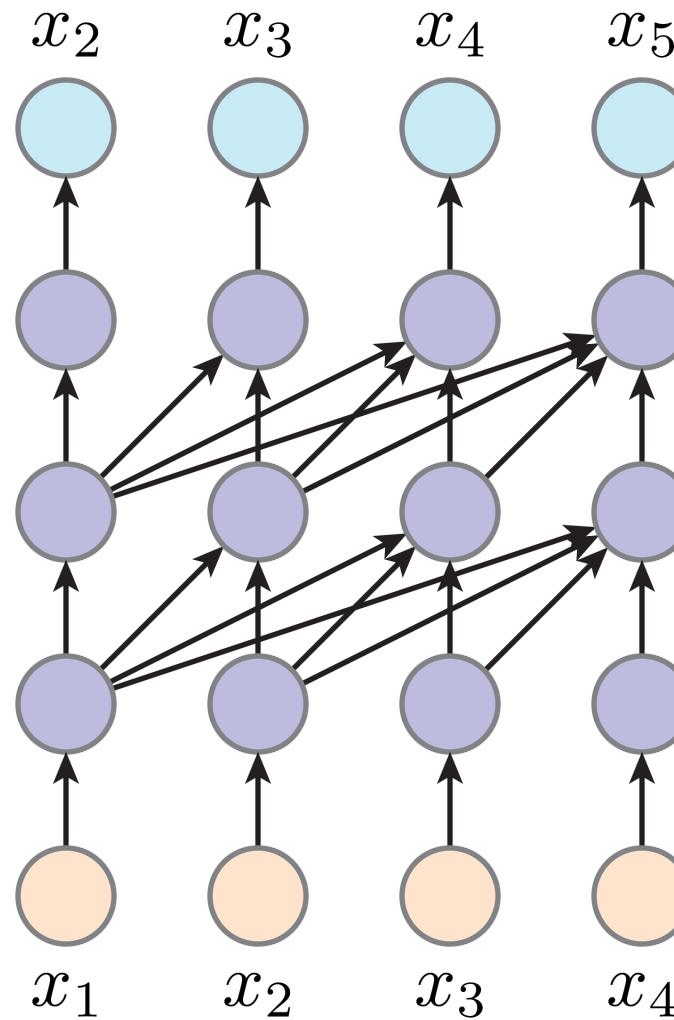


Brief: Attention (Transformer) for AR

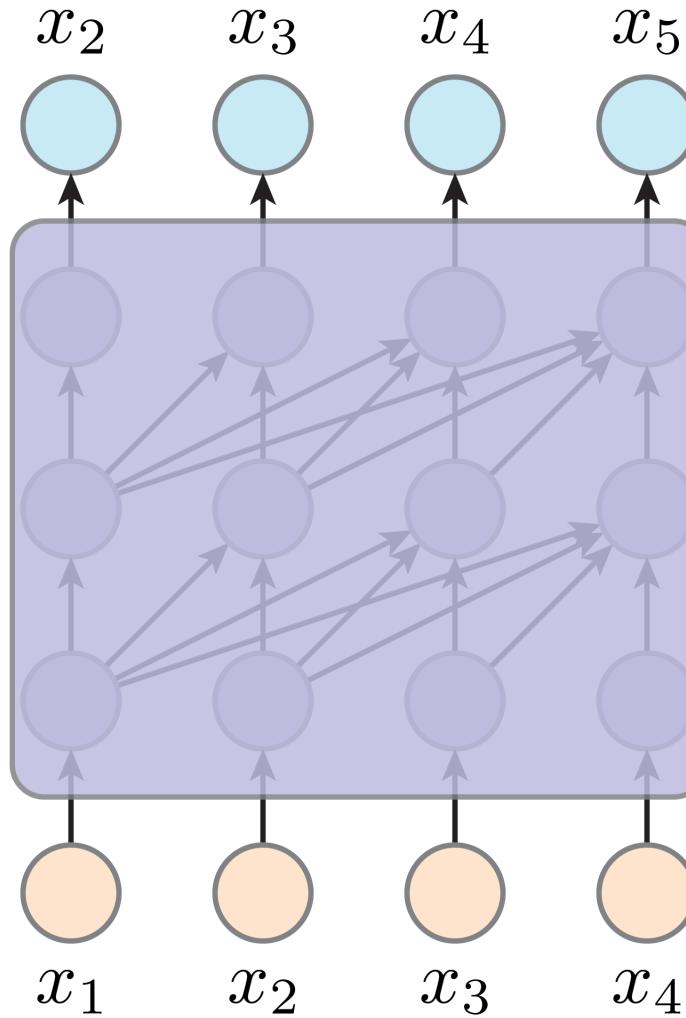


Brief: Attention (Transformer) for AR

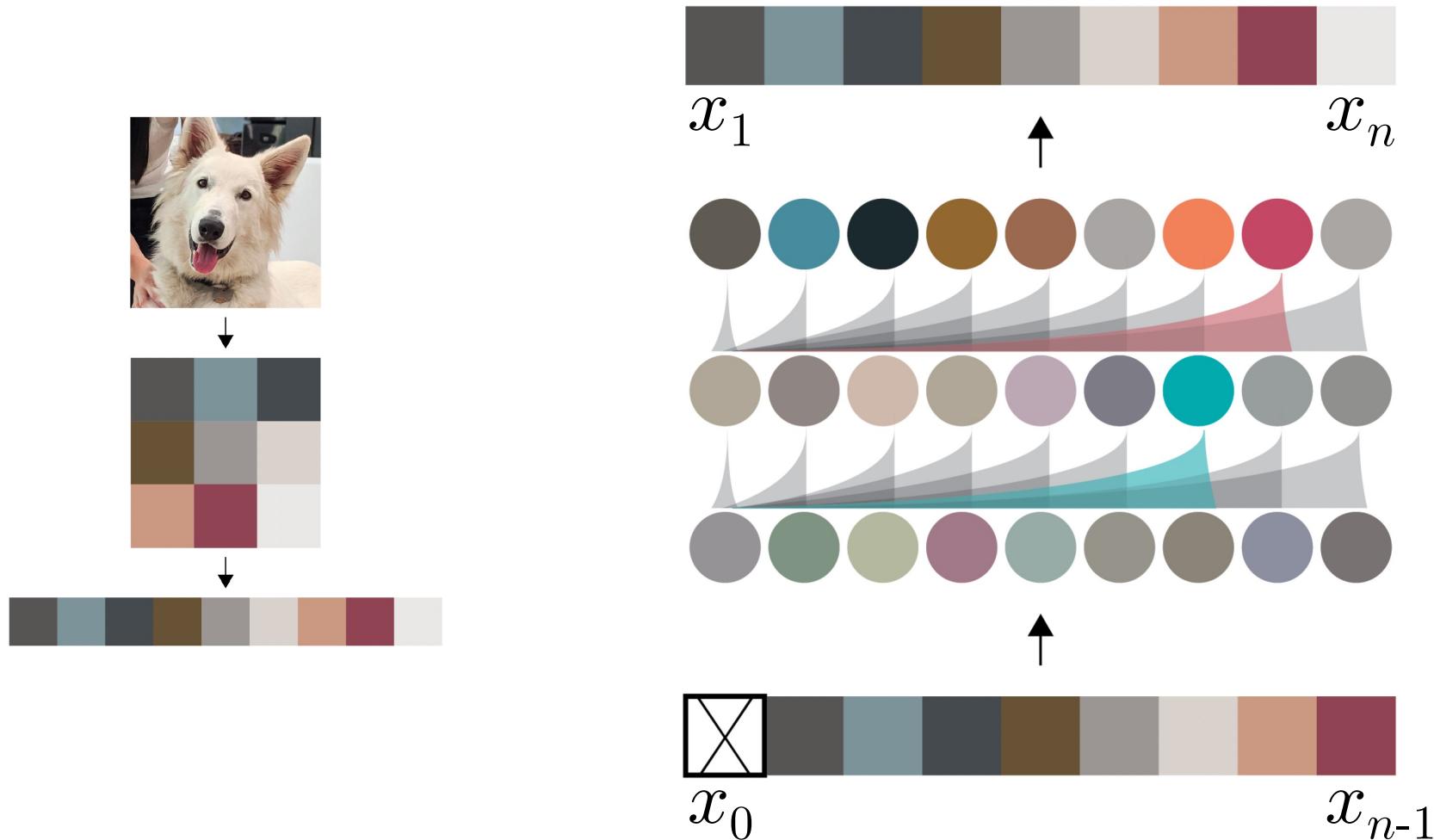
shift target by one step



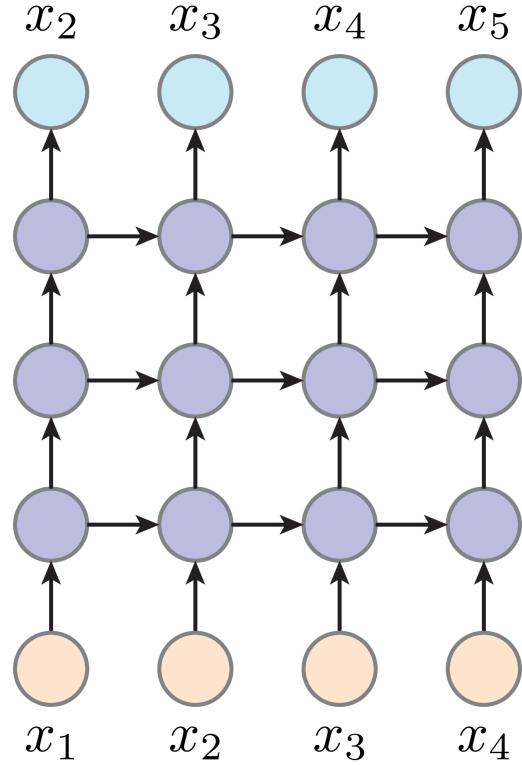
Brief: Attention (Transformer) for AR



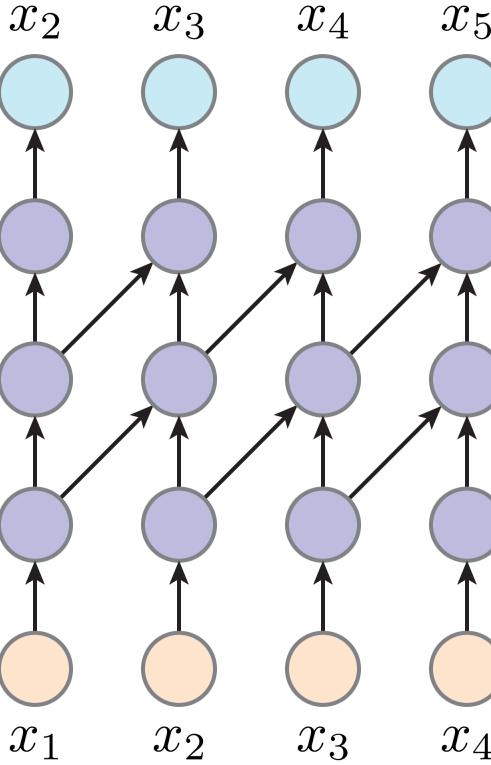
Example: image GPT (iGPT)



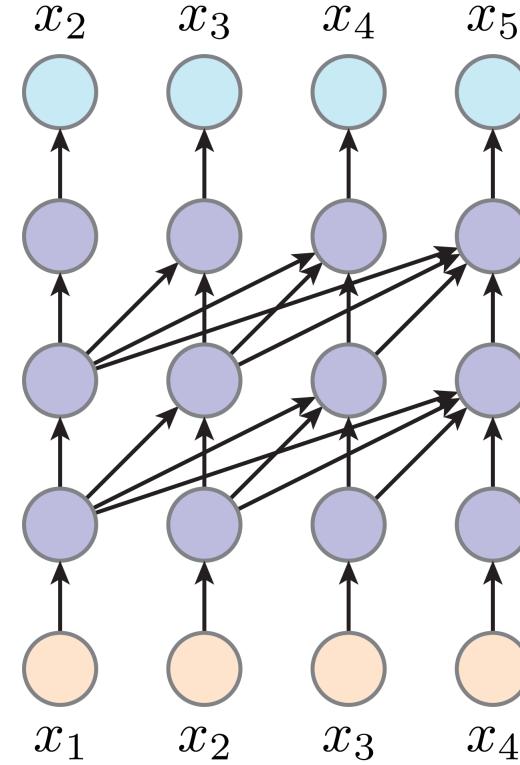
Summary: Network Architectures for AR



RNN



CNN



Attention

Summary: Autoregressive Models

Takeaways:

- Joint distribution \Rightarrow 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