Autoregressive Models

https://hku-data8018.github.io/

Bo Dai

Spring 2026

- First come first serve

- Student Presentation Schedule
  - https://hkuhk-my.sharepoint.com/:x:/g/personal/bdai_hku_hk/IQCw7NjfThocQZkrtKkYWOexAUxFssmjnC6OhlxUvSBITEo?e=NbkAHd

- Group Info
  - https://hkuhk-my.sharepoint.com/:x:/g/personal/bdai_hku_hk/IQCCN_t6WAxwR4MGYn4QK_LRAeFud4miJGwrJNAYJWO7kU0?e=FVeeVu

- Final Group Presentation Schedule
  - https://hkuhk-my.sharepoint.com/:x:/g/personal/bdai_hku_hk/IQBpwEA-lzQUSorbZL4MNCi-Ae06e_B-tWlQsj2JxTGxgHo?e=3ntO3G

- Slides reference
  - MIT EECS Fall 2024, 6.S978 Deep Generative Models, https://mit-6s978.github.io/
  - Stanford Fall 2023, CS236 Deep Generative Models, https://deepgenerativemodels.github.io/

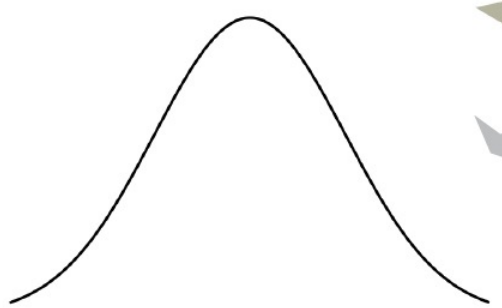
Modeling a joint distribution

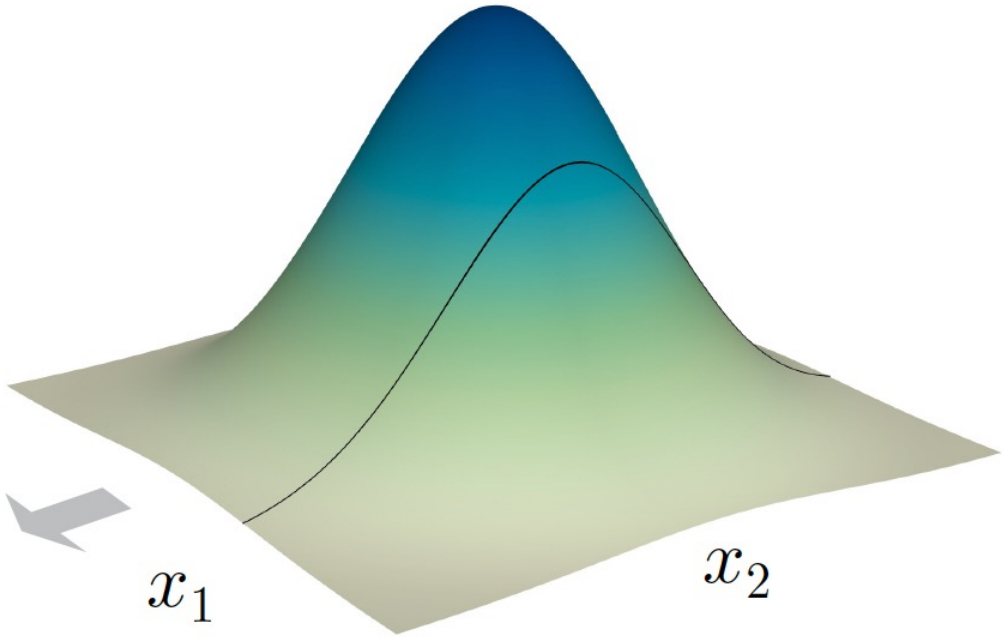
- It's effective to model joint distributions by independent distributions

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

conditional = marginal

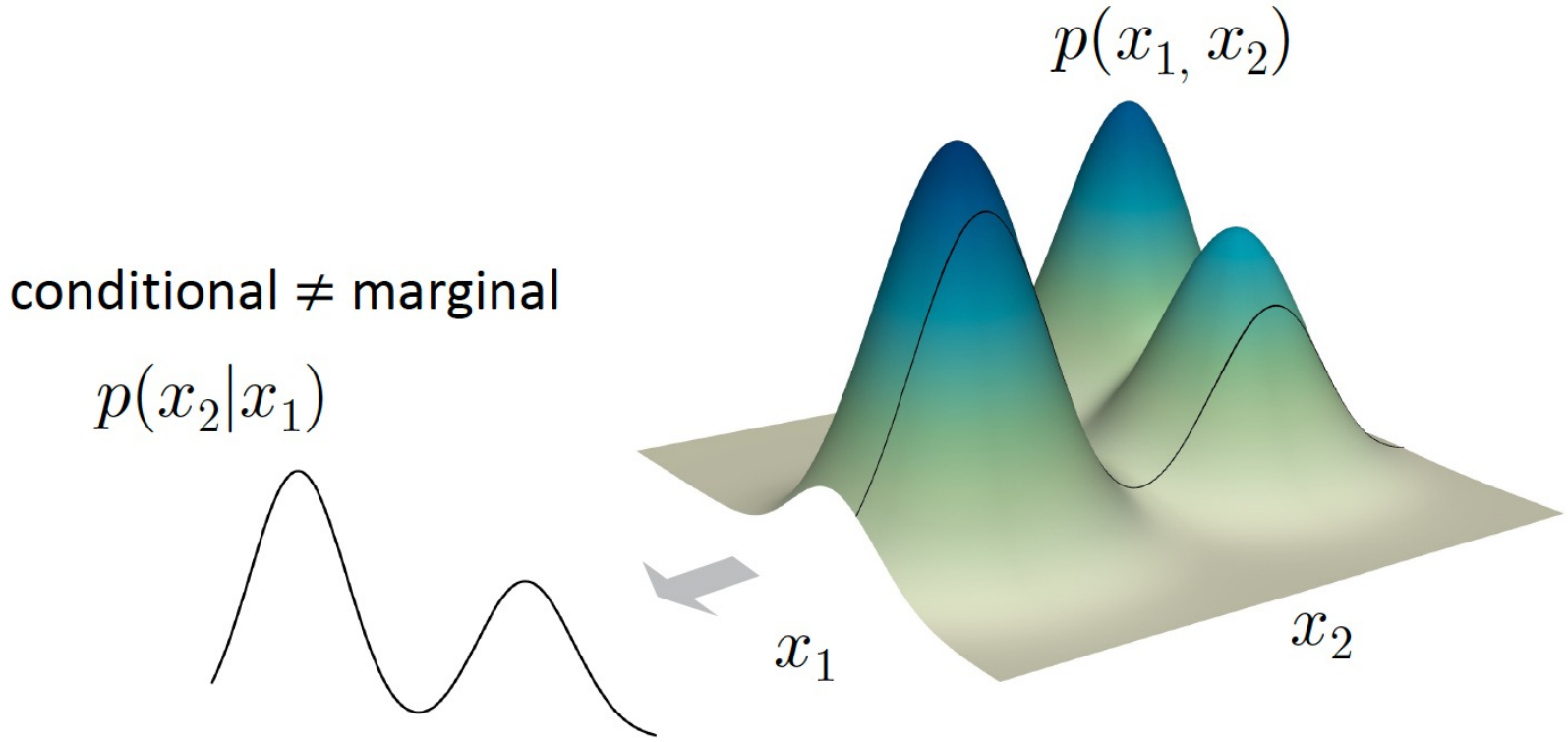
$$p(x_2|x_1) = p(x_2)$$





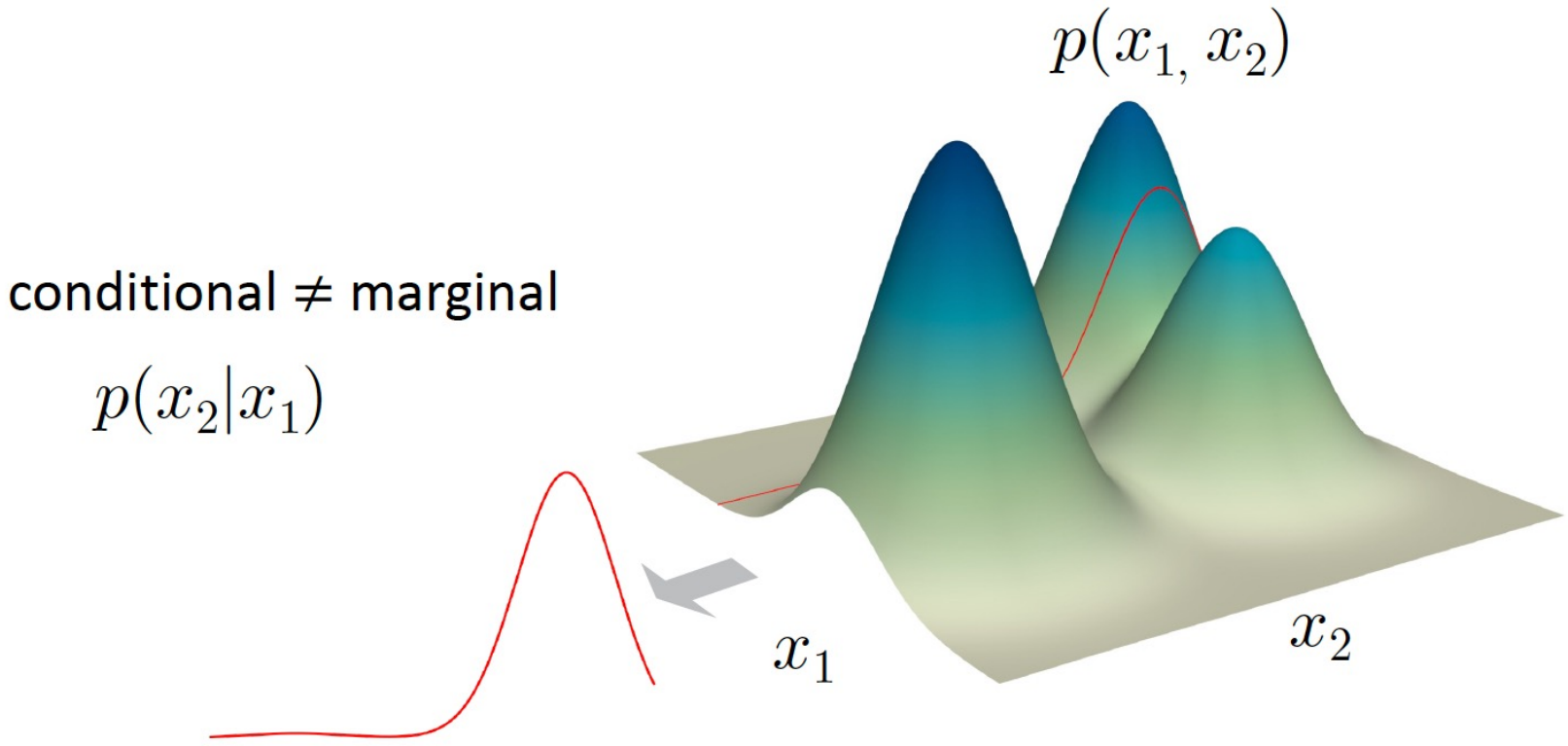
Modeling a joint distribution

- It's effective to model joint distributions by independent distributions
- However, real-world problems always involve dependent variables



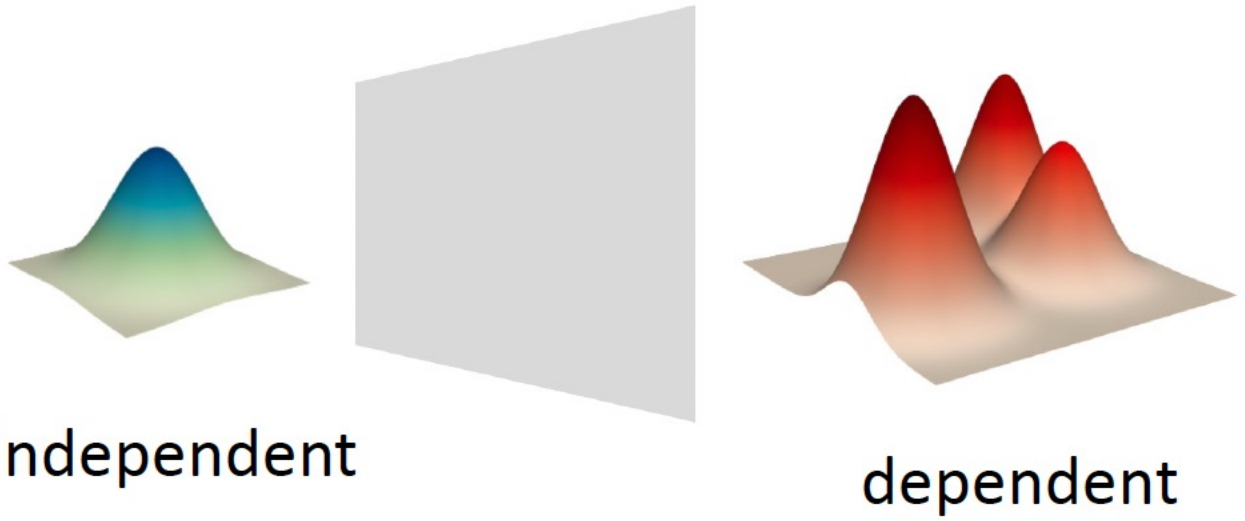
Modeling a joint distribution

- It's effective to model joint distributions by independent distributions
- However, real-world problems always involve dependent variables



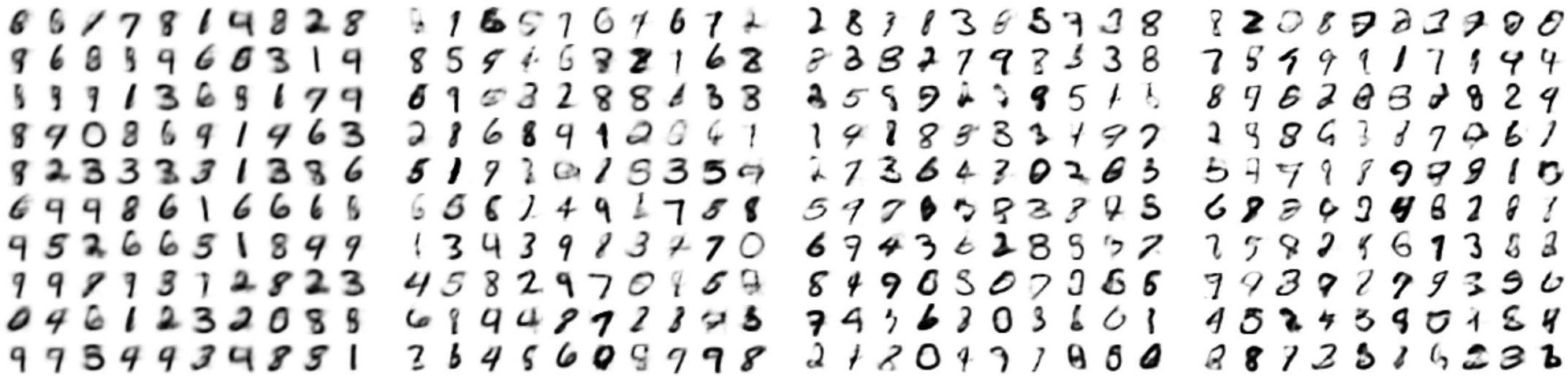
Modeling a joint distribution

- Solution 1: modeling by independent latents (e.g. VAE)
  - Mapping: independent → dependent
  - Strict assumption for high-dimensional data (e.g., 32 x 32 x 3 pixels)
  - Often with low-dimensional latents



Modeling a joint distribution

- Solution 1: modeling by independent latents (e.g. VAE)
  - Mapping: independent → dependent
  - Strict assumption for high-dimensional data (e.g., 32 x 32 x 3 pixels)
  - Often with low-dimensional latents
  - May not be very effective



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

VAE results on 784-d MNIST data

https://arxiv.org/abs/1312.6114

Modeling a joint distribution

- Solution 1: modeling by independent latents (e.g. VAE)
- Solution 2: modeling by conditional distributions using the chain rule

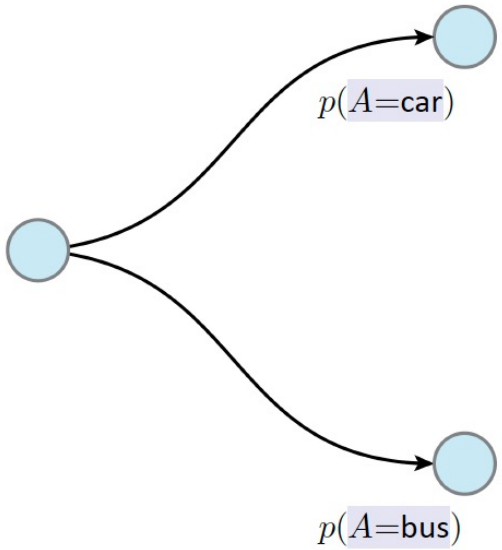
$$p(A,\ B) = p(A)p(B \mid A)$$

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

Modeling a joint distribution

- Solution 1: modeling by independent latents (e.g. VAE)
- Solution 2: modeling by conditional distributions using the chain rule

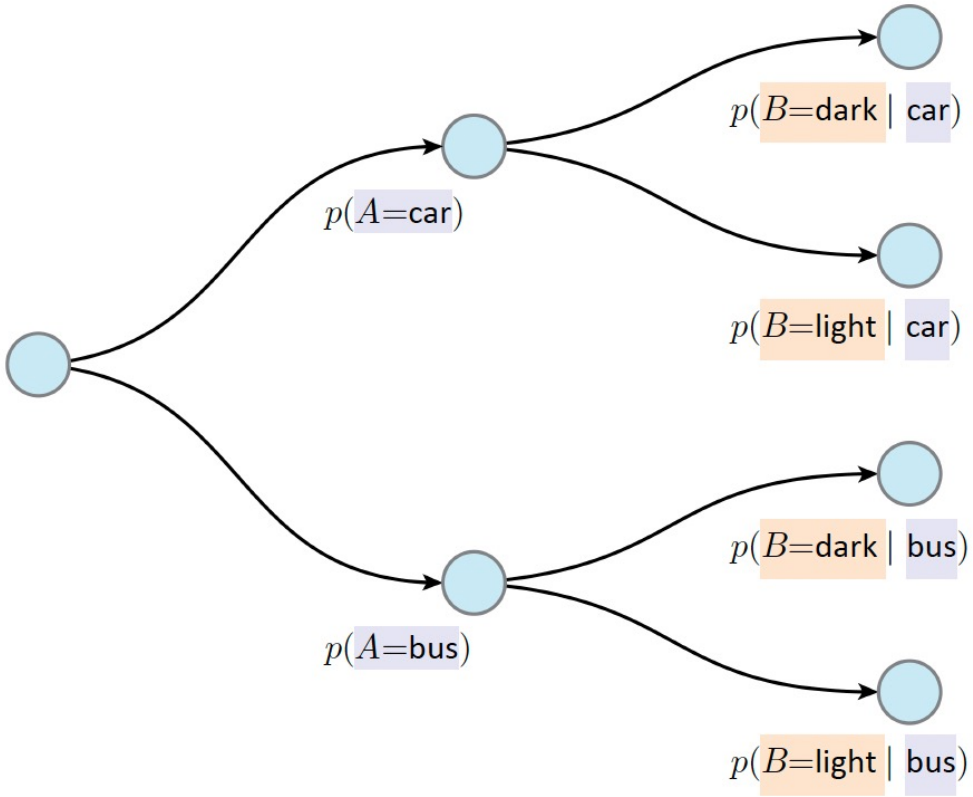
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



Modeling a joint distribution

- Solution 1: modeling by independent latents (e.g. VAE)
- Solution 2: modeling by conditional distributions using the chain rule

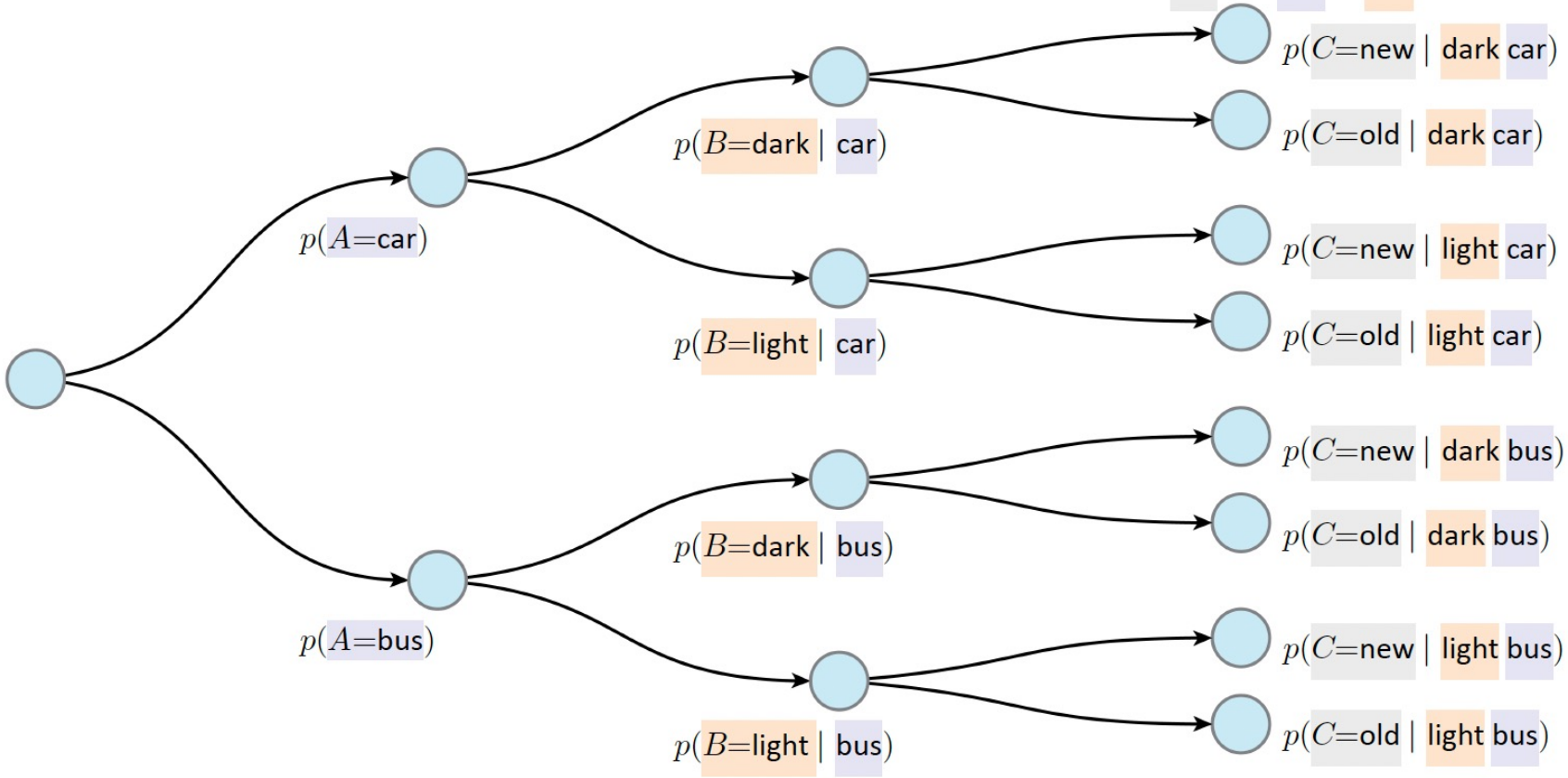
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



Modeling a joint distribution

- Solution 1: modeling by independent latents (e.g. VAE)
- Solution 2: modeling by conditional distributions using the chain rule

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



Modeling a joint distribution

- Solution 1: modeling by independent latents (e.g. VAE)
- Solution 2: modeling by conditional distributions using the chain rule
  - In any order

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

Modeling a joint distribution

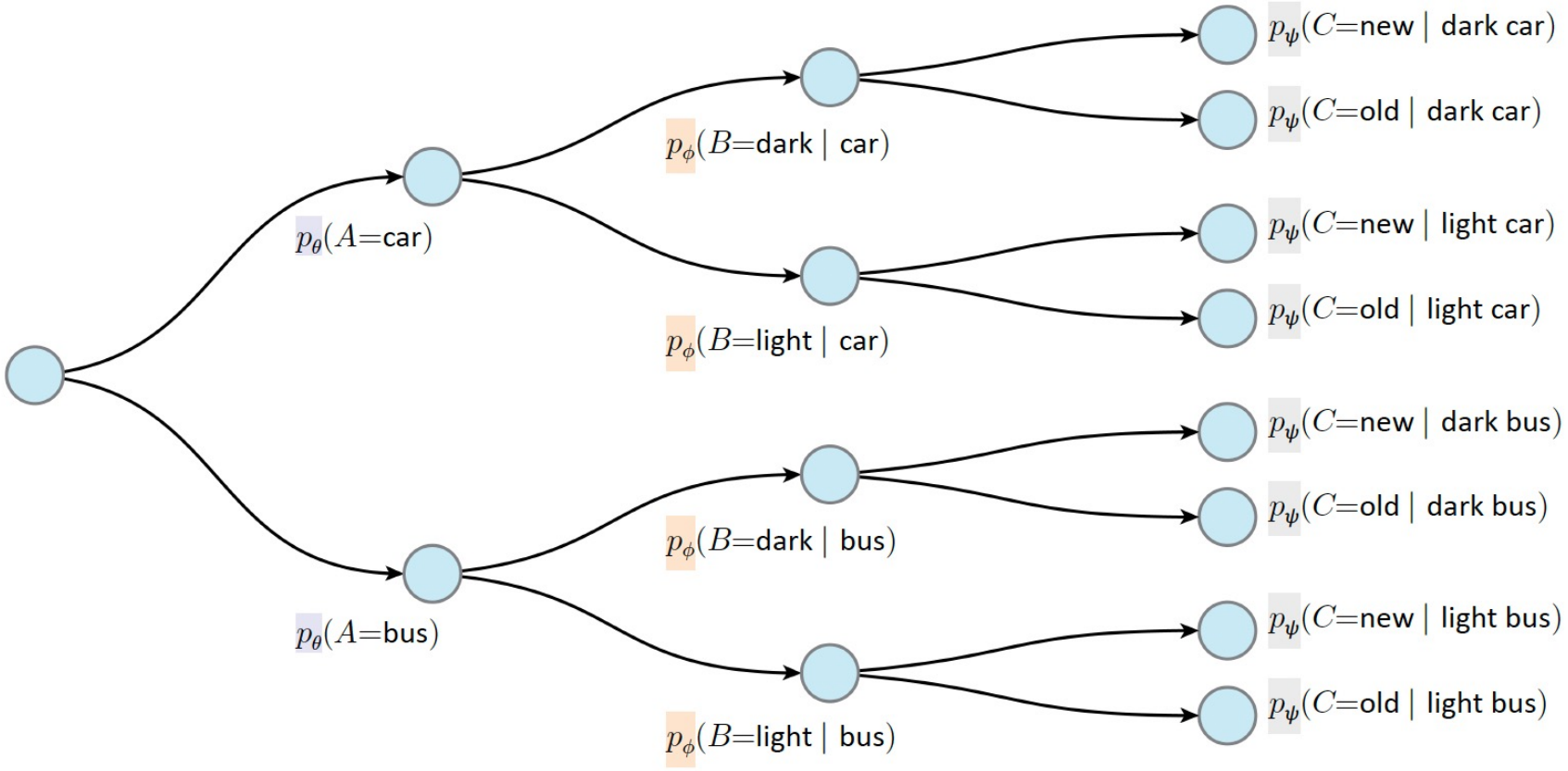
- Solution 1: modeling by independent latents (e.g. VAE)
- Solution 2: modeling by conditional distributions using the chain rule
  - In any order
  - In any partition

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

Modeling a joint distribution

- Modeling by conditional distributions using the chain rule, with a neural network

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



Modeling a joint distribution

- Modeling by conditional distributions using the chain rule, with a neural network
  - Parameterizing p(A, B, C) vs. parameterizing p(C|A, B)

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

Modeling a joint distribution

- Modeling by conditional distributions using the chain rule, with a neural network
  - Parameterizing p(A, B, C) vs. parameterizing p(C|A, B)
    - p(A, B, C) has 3 variables
    - p(C | A, B) has 1 variable and 2 conditions

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

Modeling a joint distribution

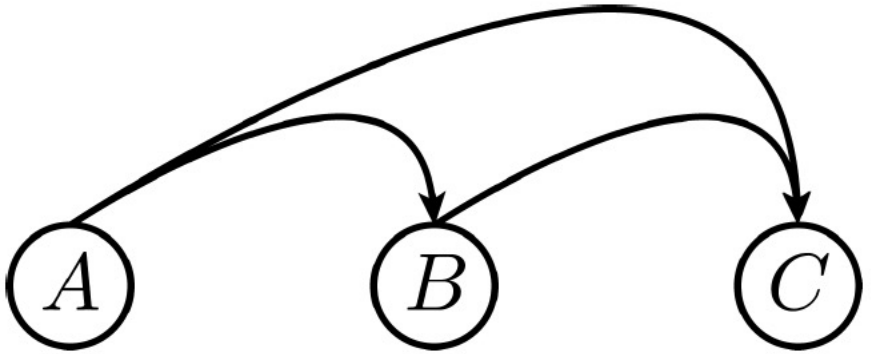
- Modeling by conditional distributions using the chain rule, with a neural network
  - Parameterizing p(A, B, C) vs. parameterizing p(C|A, B)
  - Weight sharing?
    - Conceptually, each p has its own weights, but weight sharing can imply inductive biases

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

Dependency graphs

- Different orders lead to different dependency graphs

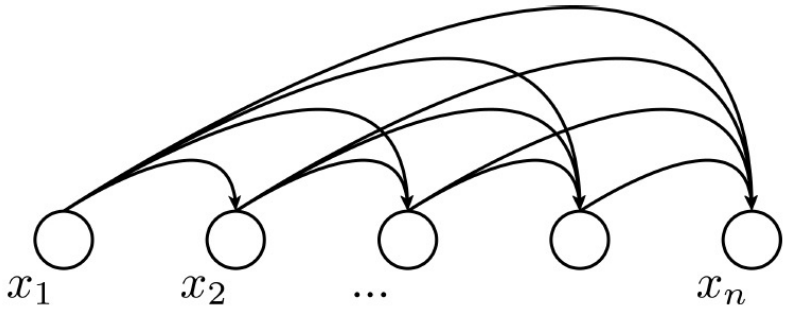
$$p(A, B, C) = p(A)p(B \mid A)p(C \mid A, B)$$



Dependency graphs

- Different orders lead to different dependency graphs, and reflect different prior knowledge

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

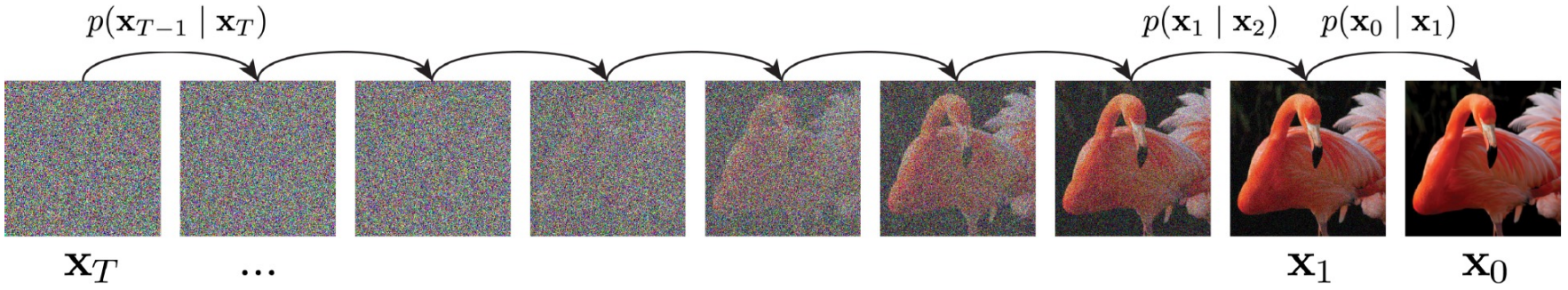


words in a sentence

Dependency graphs

- Different orders lead to different dependency graphs, and reflect different prior knowledge

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

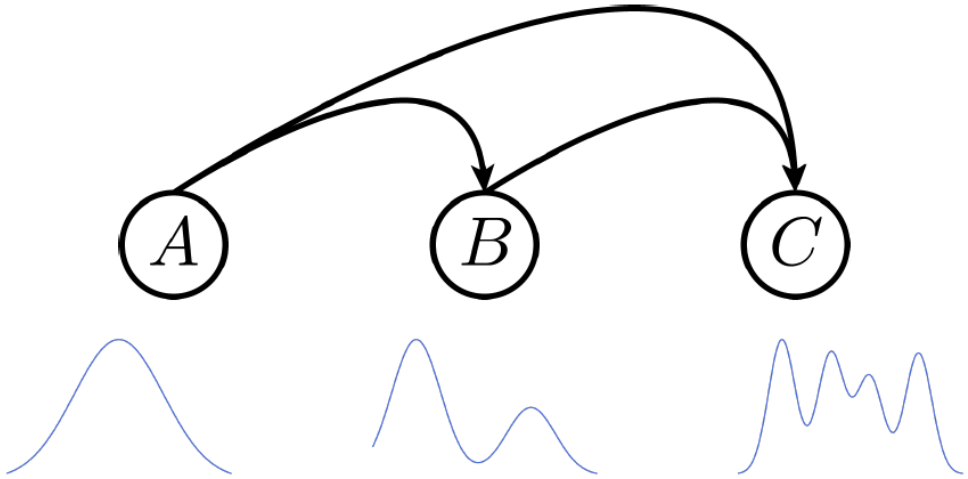


denoising steps in a reverse process

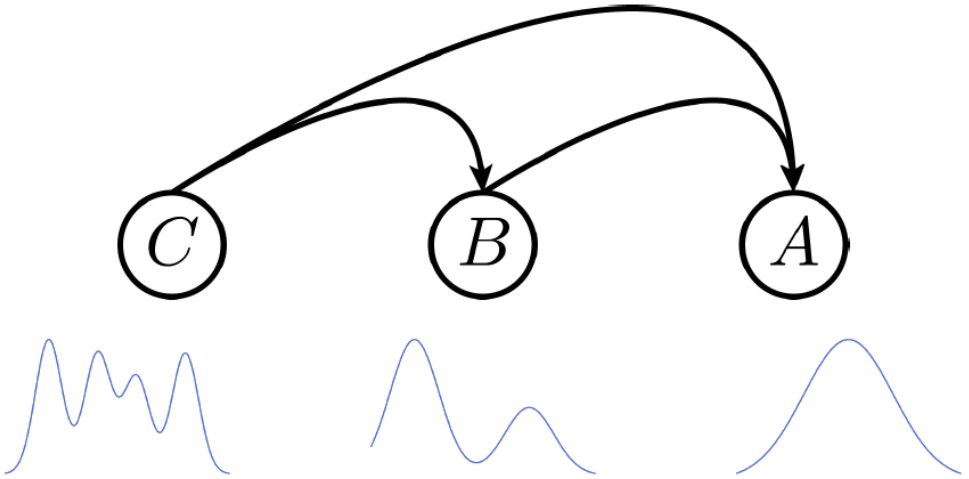
Dependency graphs

- Different orders lead to different dependency graphs, and reflect different prior knowledge
- Different orders lead to distributions of different levels of complexity

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



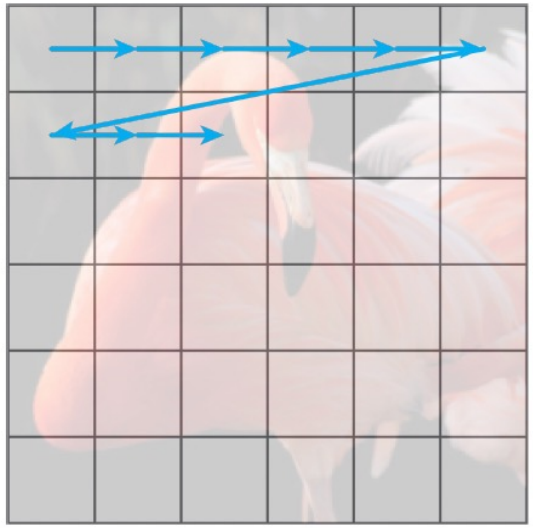
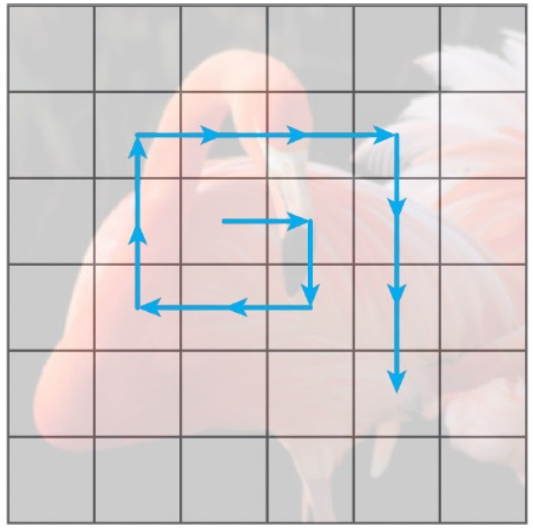
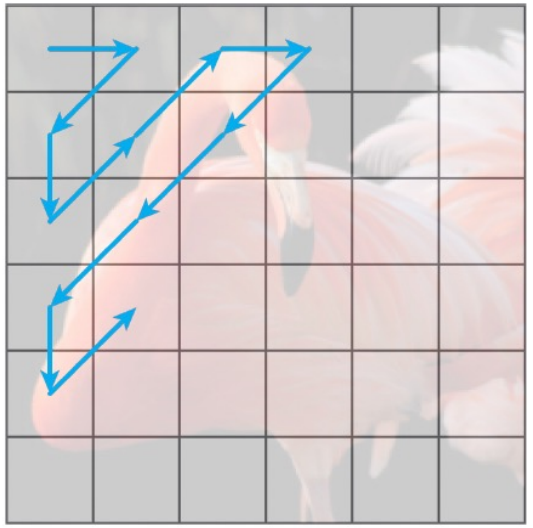
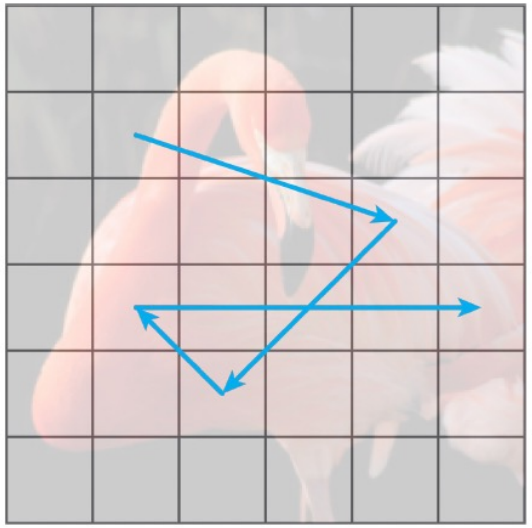
$$p(A, B, C) = p(C)p(B \mid C)p(A \mid B, C)$$



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

Dependency graphs

- Different orders lead to different dependency graphs, and reflect different prior knowledge
- Different orders lead to distributions of different levels of complexity

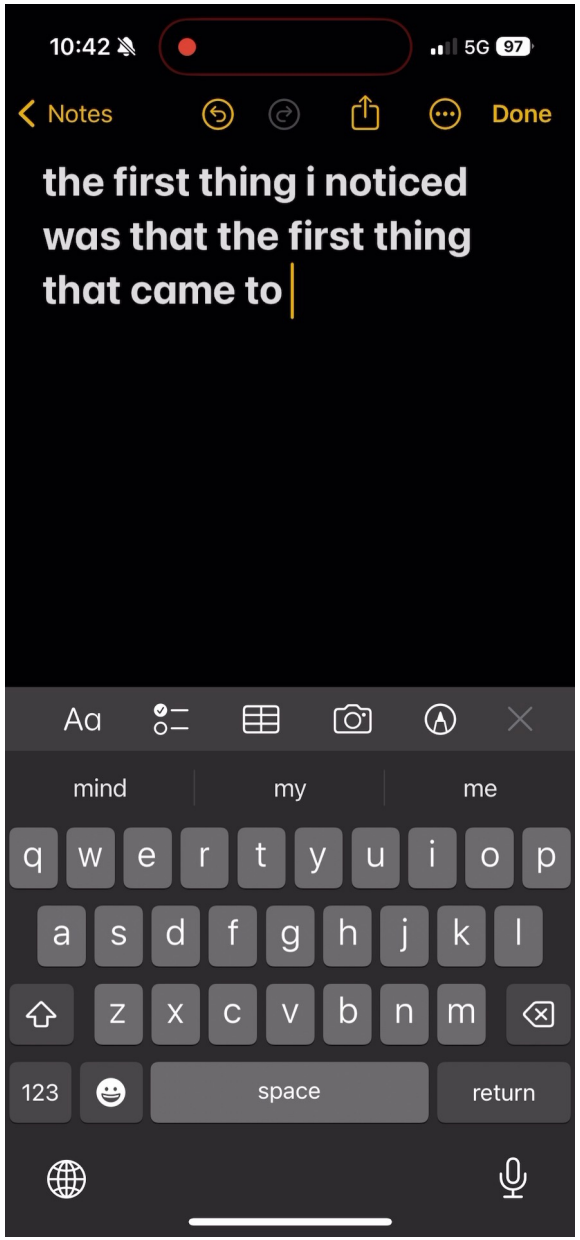





https://arxiv.org/abs/2203.12054

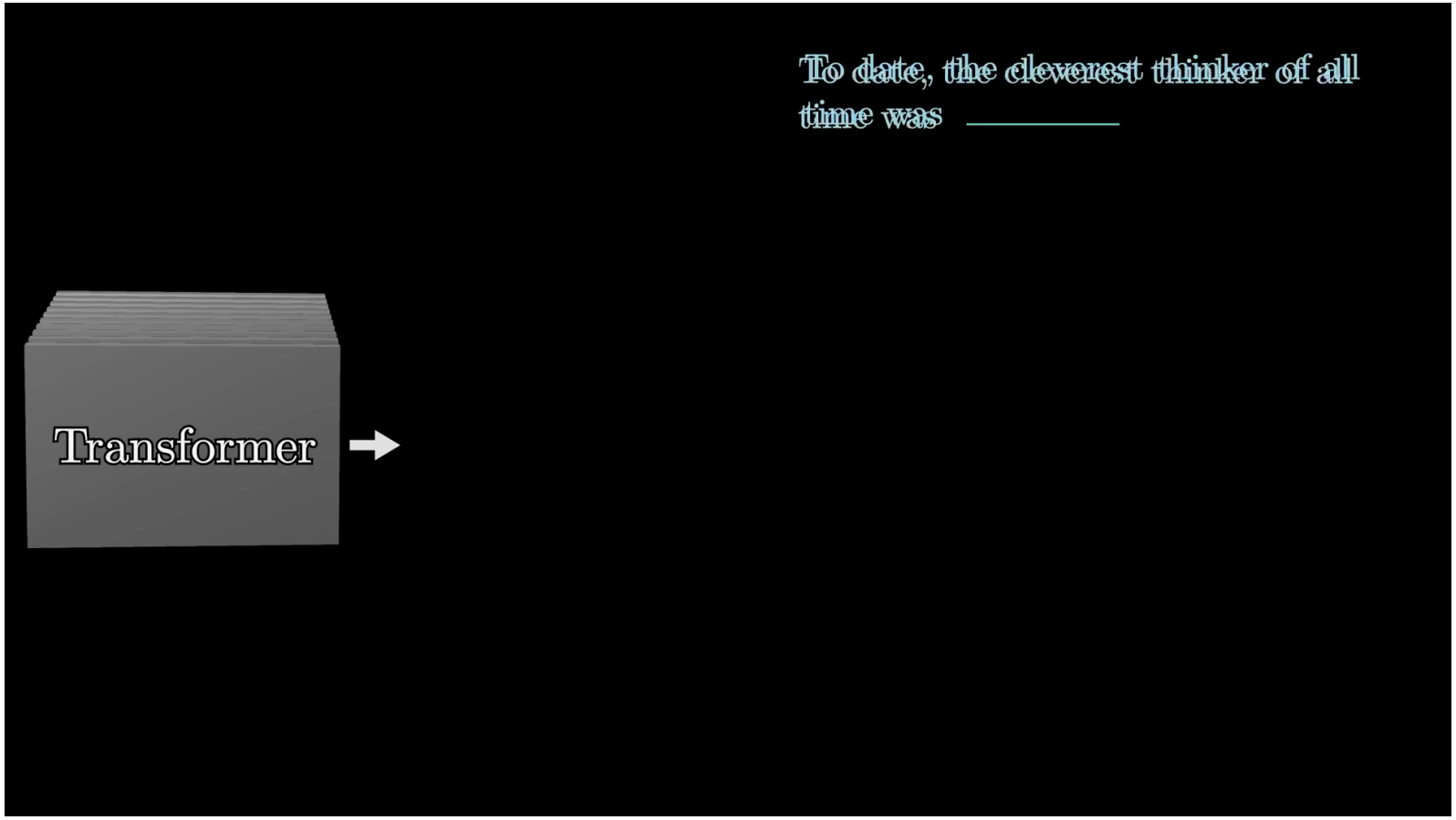
Summary

- Joint distribution ⇒ product of conditionals, divide-and-conquer
- Any order, any partition
- Dependency graphs: induce prior knowledge
- These are not specific to Autoregressive models.

Keyboard


10:42
5G 97
Notes
Done
the first thing i noticed was that the first thing that came to
Aa
mind
my
me
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
return

ChatGPT



Auto + Regression

- Auto: “self”
  - Using its own outputs as inputs for next predictions

- Regression:
  - Estimating relationships between variables

Autoregressive Models

Auto + Regression

- Auto: “self”
  - Using its own outputs as inputs for next predictions

- Regression:
  - Estimating relationships between variables

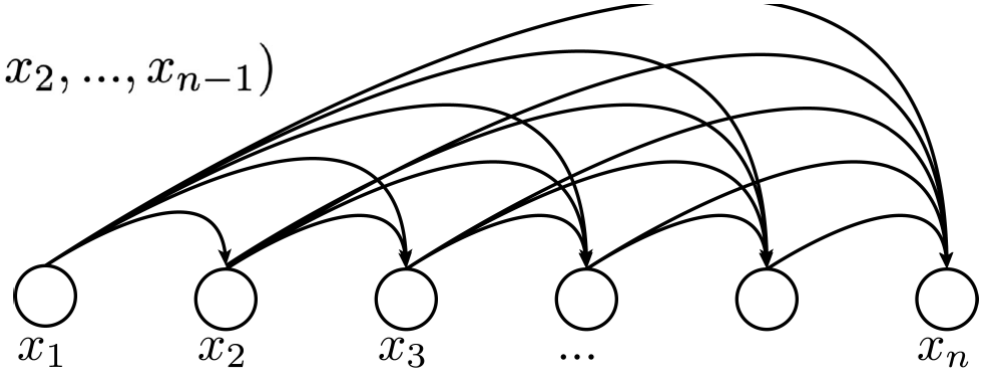
- Autoregressive implies an inference-time behavior
- Training-time is not necessarily autoregressive

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

- Conceptually, x can be any representation
  - Not necessarily sequential/temporal
    - All dims of a vector
    - All pixels of an image
    - 2D, 3D, or High-dimensional arrays

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

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

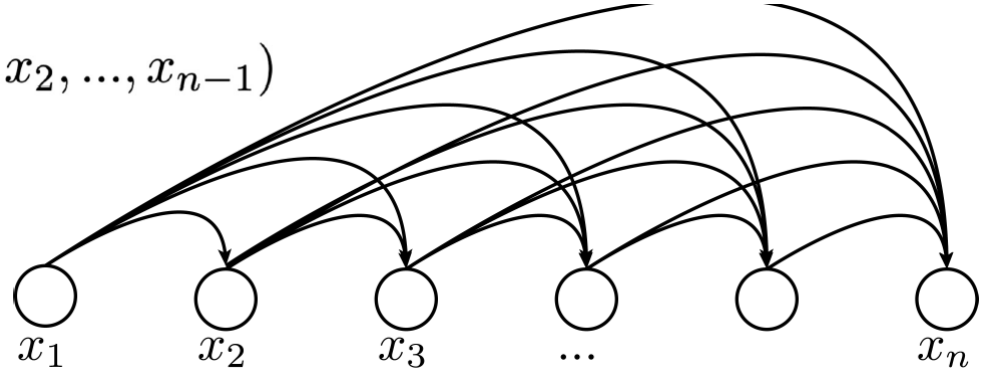


In general, autoregression is a way of modeling joint distribution by a product of conditional distributions

- Conceptually, x can be any order and any partition
  - Reverse order is valid
  - Each of x_i can be a scalar, vector or tensor

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

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

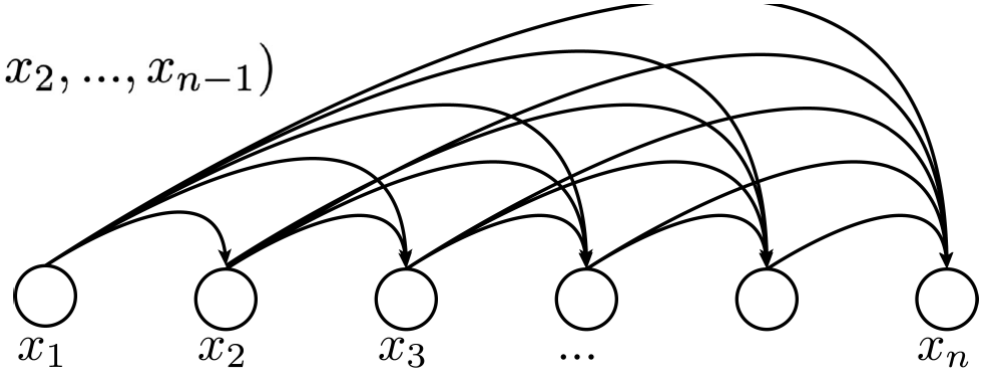


In general, autoregression is a way of modeling joint distribution by a product of conditional distributions

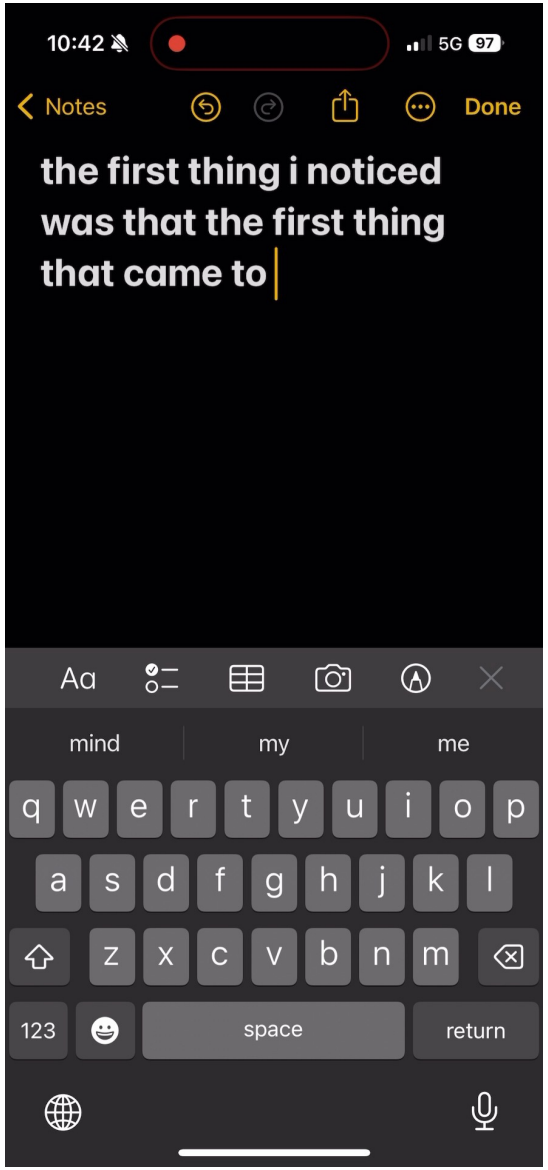
- No compromise/approximation
- Always valid
- But some are easier to model

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

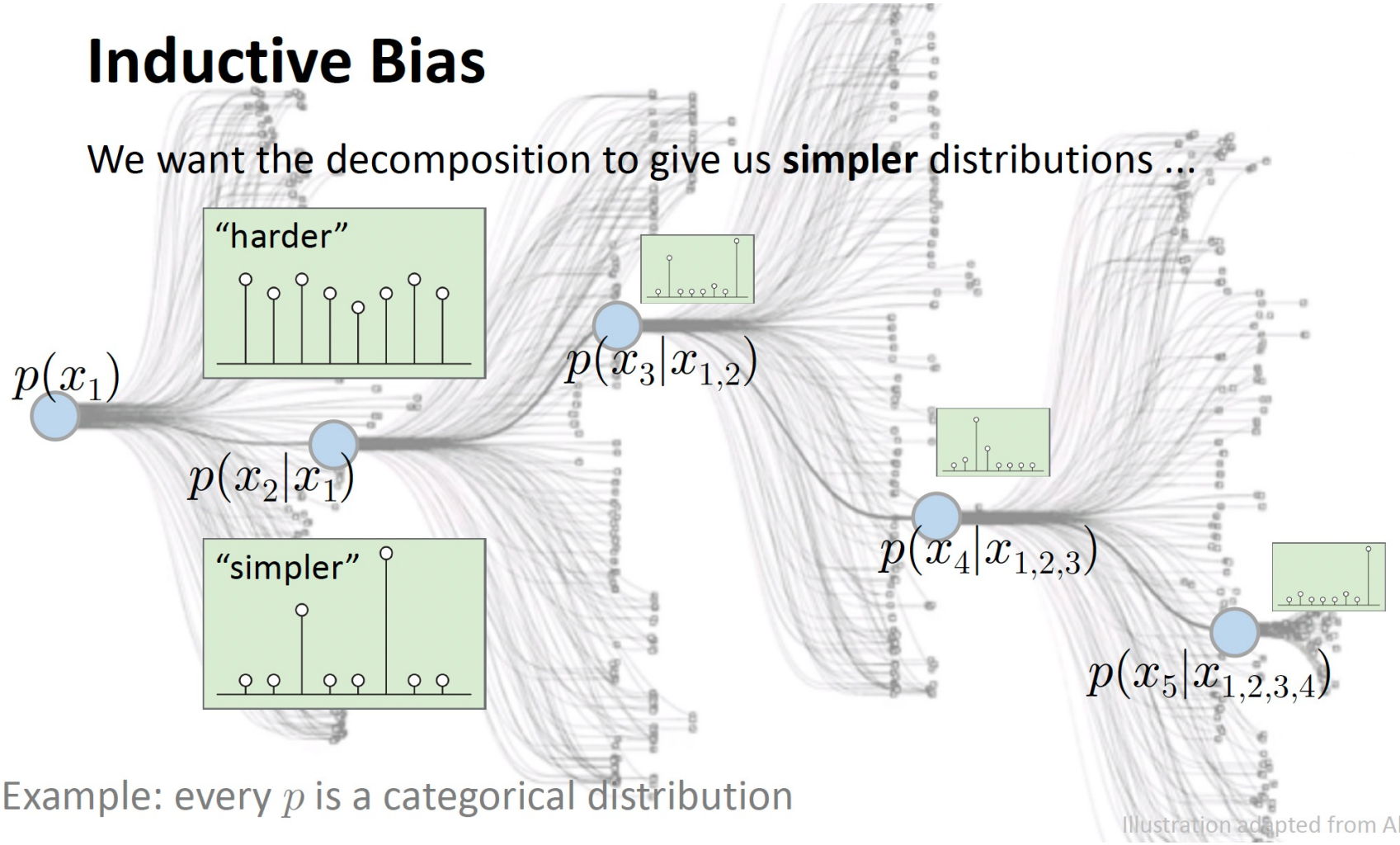
$$= \prod_{i=1}^{n} p(x_i \mid x_1, x_2, ..., x_{i-1})$$



Previous outputs can largely reduce the next plausible outputs.



Previous outputs can largely reduce the next plausible outputs.



Example: every p is a categorical distribution

Illustration adapted from AlphaGo

Previous outputs can largely reduce the next plausible outputs.
We want the decomposition to give us simpler distributions

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

Previous outputs can largely reduce the next plausible outputs.
We want the decomposition to give us simpler distributions

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

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

Previous outputs can largely reduce the next plausible outputs.
We want the decomposition to give us simpler distributions

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

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

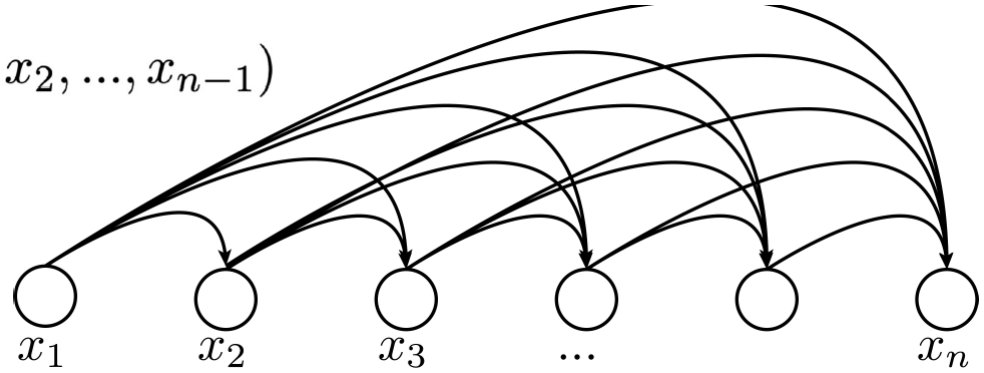
Previous outputs can largely reduce the next plausible outputs.
We want the decomposition to give us simpler distributions

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

- No compromise/approximation
- Always valid

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

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



Previous outputs can largely reduce the next plausible outputs.
We want the decomposition to give us simpler distributions

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

- No compromise/approximation
- Always valid

Shared architectures, shared weights, …

- Inductive bias

Previous outputs can largely reduce the next plausible outputs.
We want the decomposition to give us simpler distributions

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

- No compromise/approximation
- Always valid

Shared architectures, shared weights, …

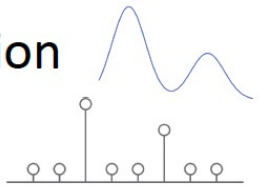
- Inductive bias

Inductive biases introduce approximations

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

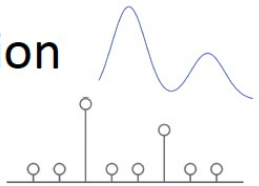


Representing one distribution

- W/ a discrete distribution, this network behaves like classification
- Discrete distribution is popular in AR models, but not a must

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

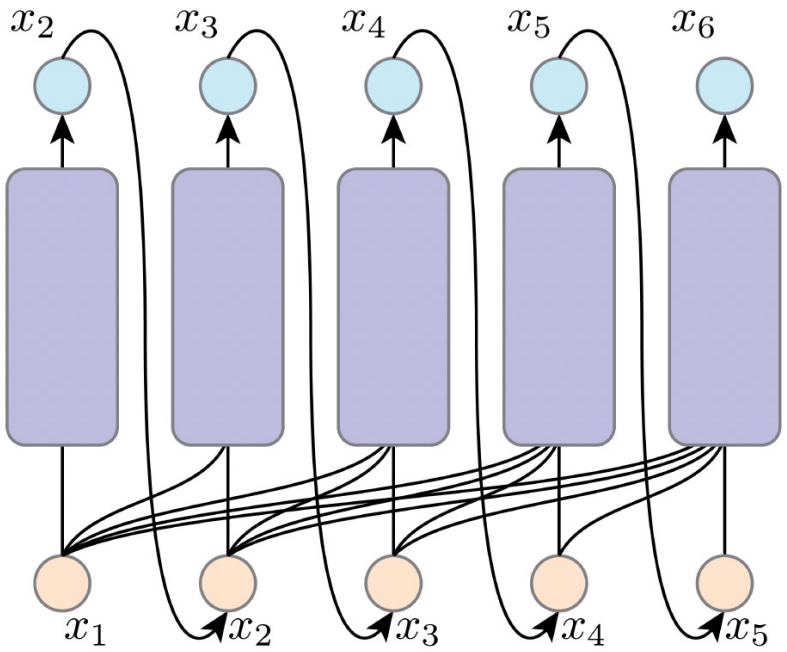
- Network inputs: $x_1, x_2, \ldots, x_{i-1}$
- Network output: a distribution of x_i
  - Continuous distribution
  - Discrete distribution



Inference of autoregression

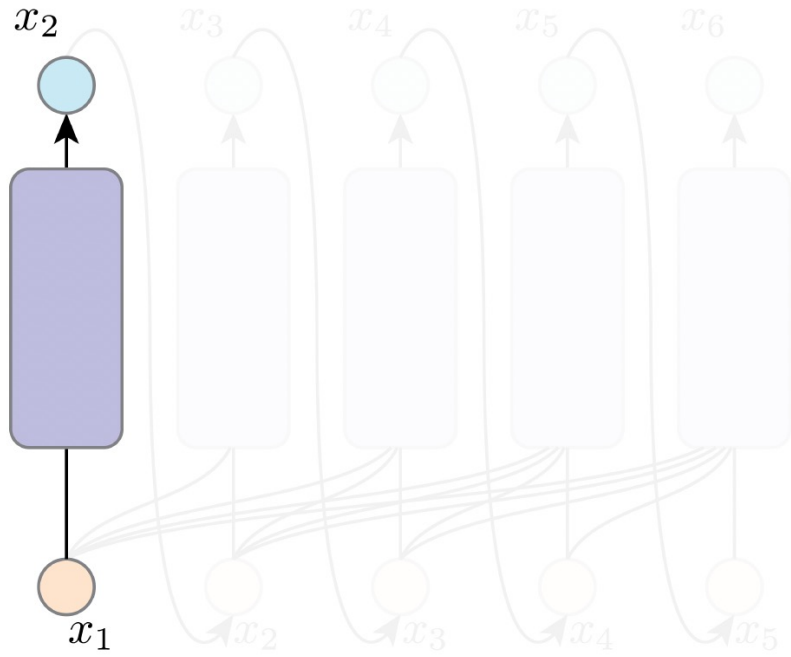
This figure implements this formulation:

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



Inference of autoregression

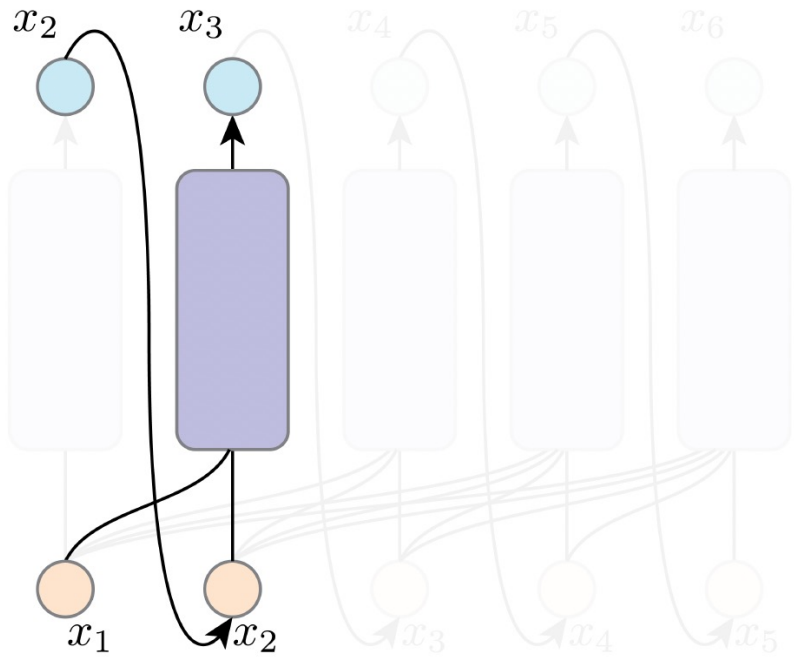
- This net models $p(x_2 \mid x_1)$
- 1 input
- 1 output



Inference of autoregression

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

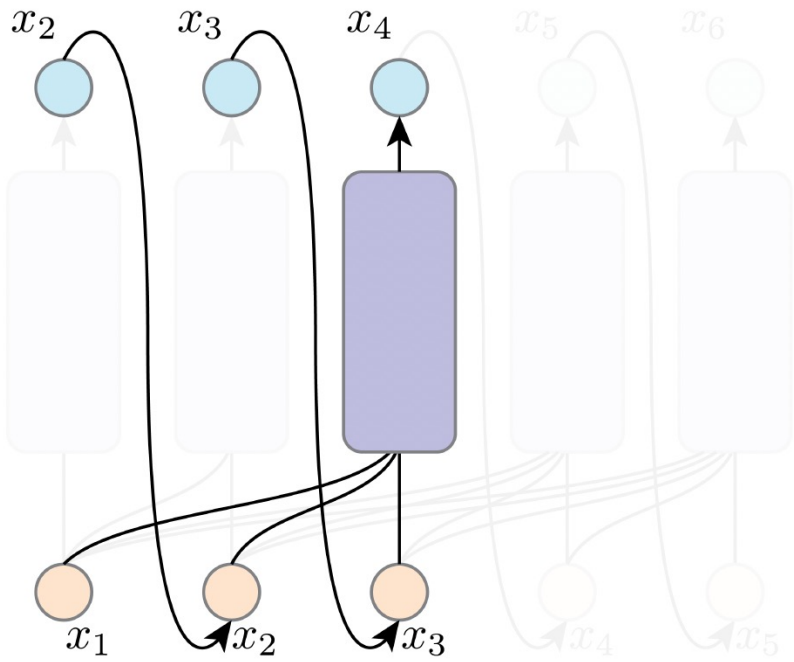
- inputs: outputs from previous steps



Inference of autoregression

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

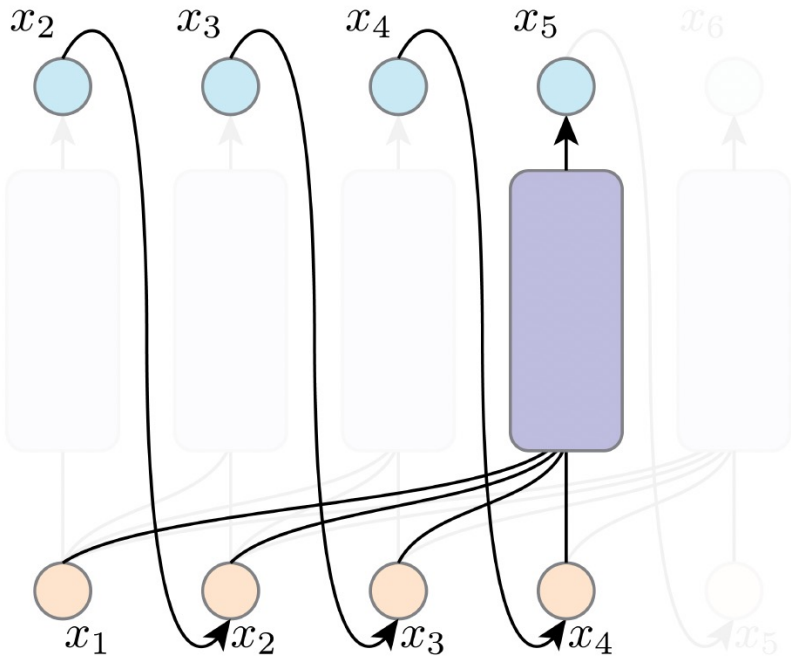
- inputs: outputs from previous steps



Inference of autoregression

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

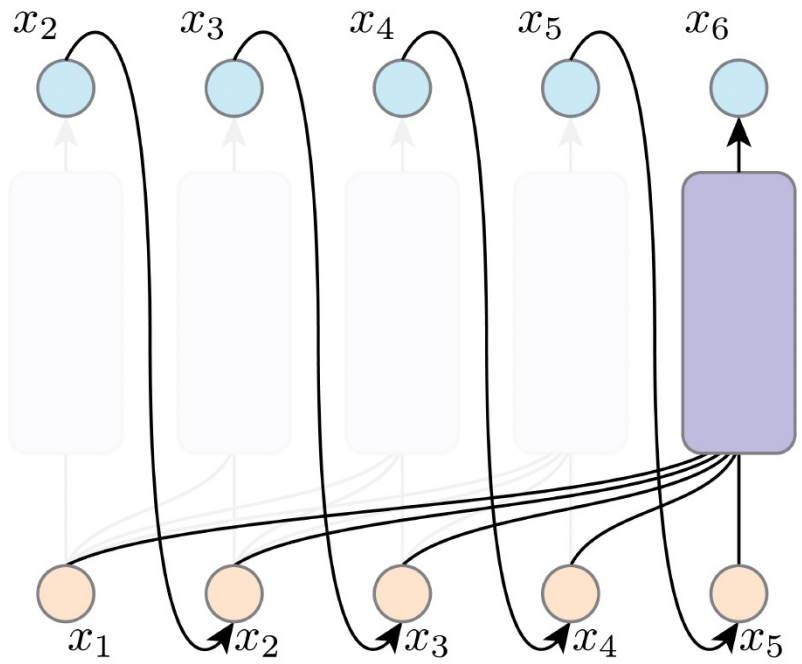
- inputs: outputs from previous steps



Inference of autoregression

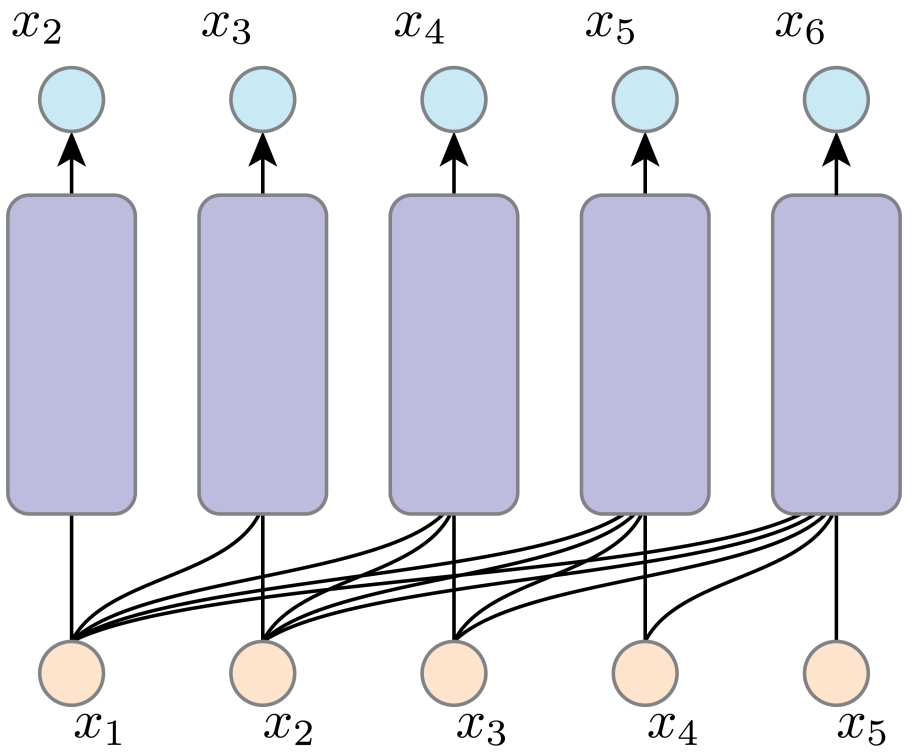
- This net models $p(x_6 \mid x_{1,2,3,4,5})$
- **5 inputs**
- 1 output

- inputs: outputs from previous steps



A recursive process implementing

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

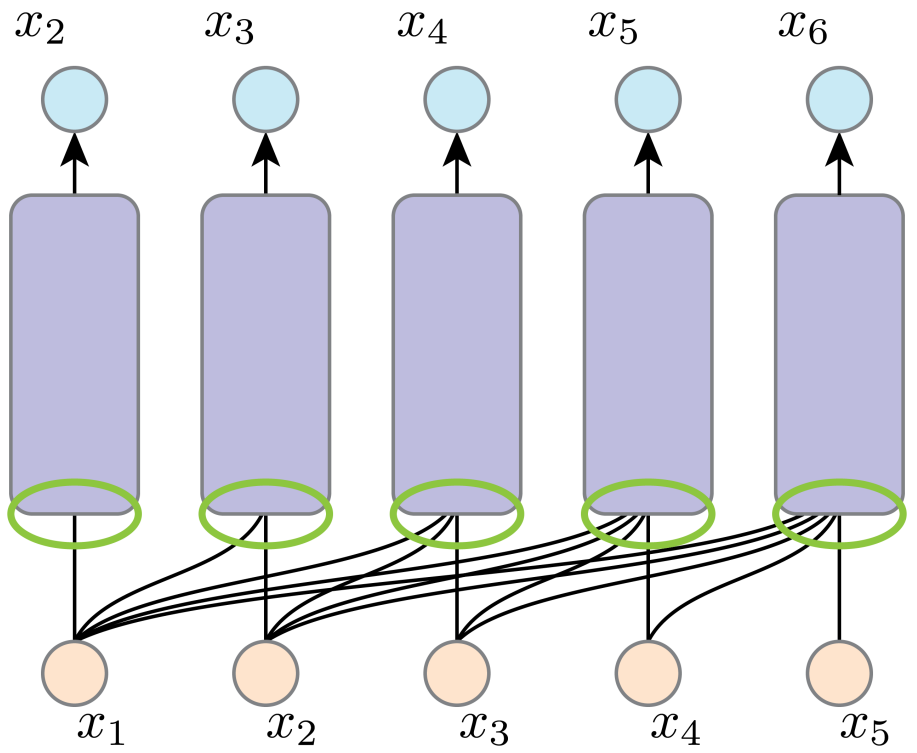


A recursive process implementing

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

In this example:

- 5 networks
- Each has 1 to 5 inputs



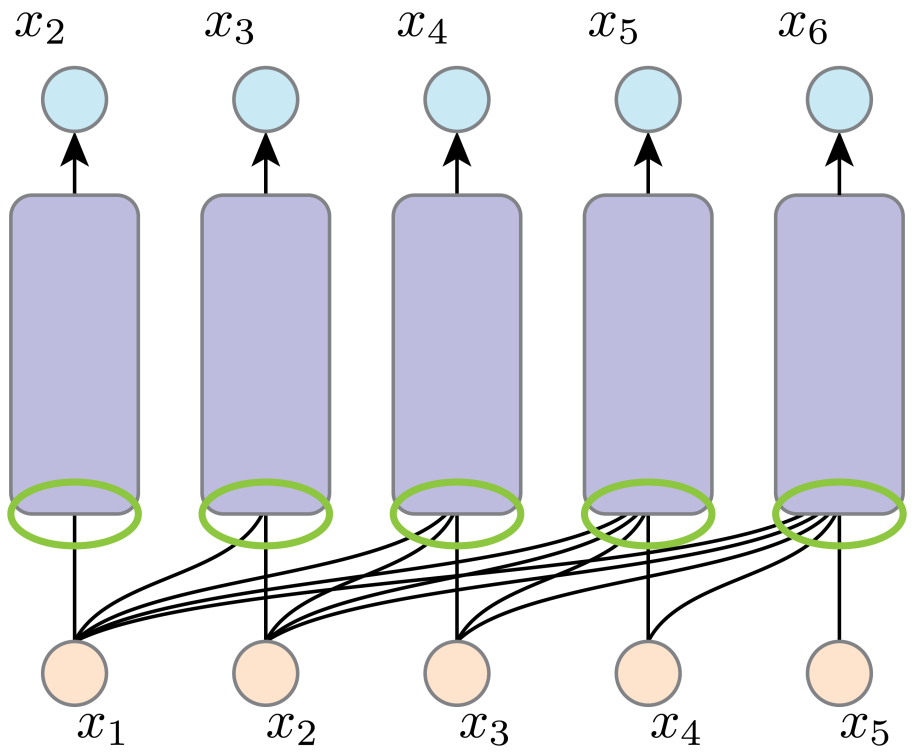
A recursive process implementing

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

In this example:

- 5 networks
- Each has 1 to 5 inputs

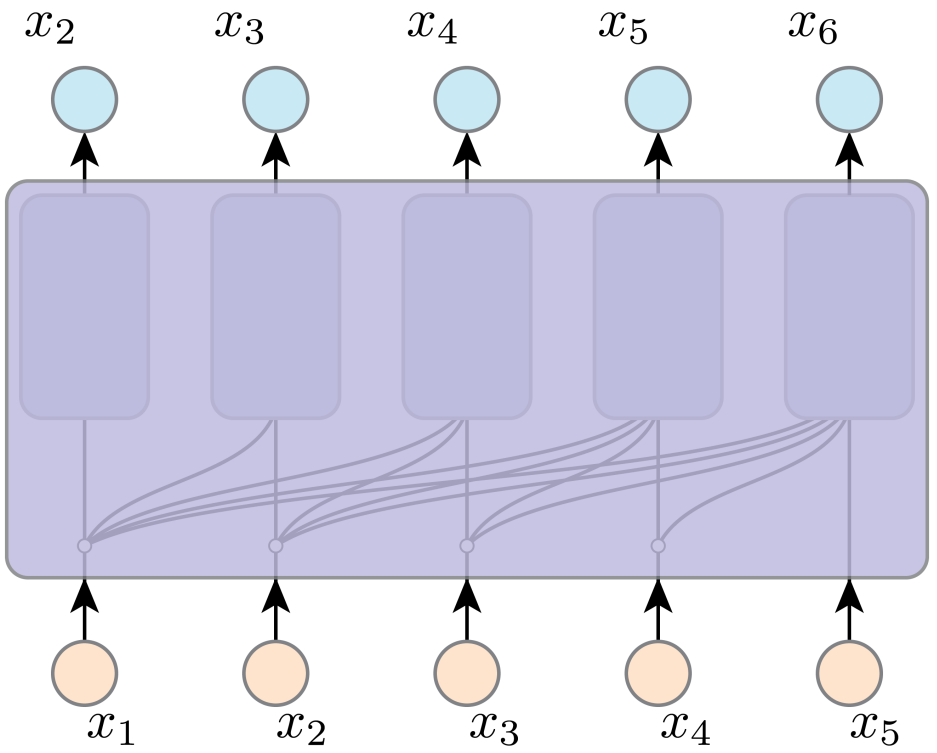
Can we do this more efficiently?



A recursive process implementing

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

Shared computation

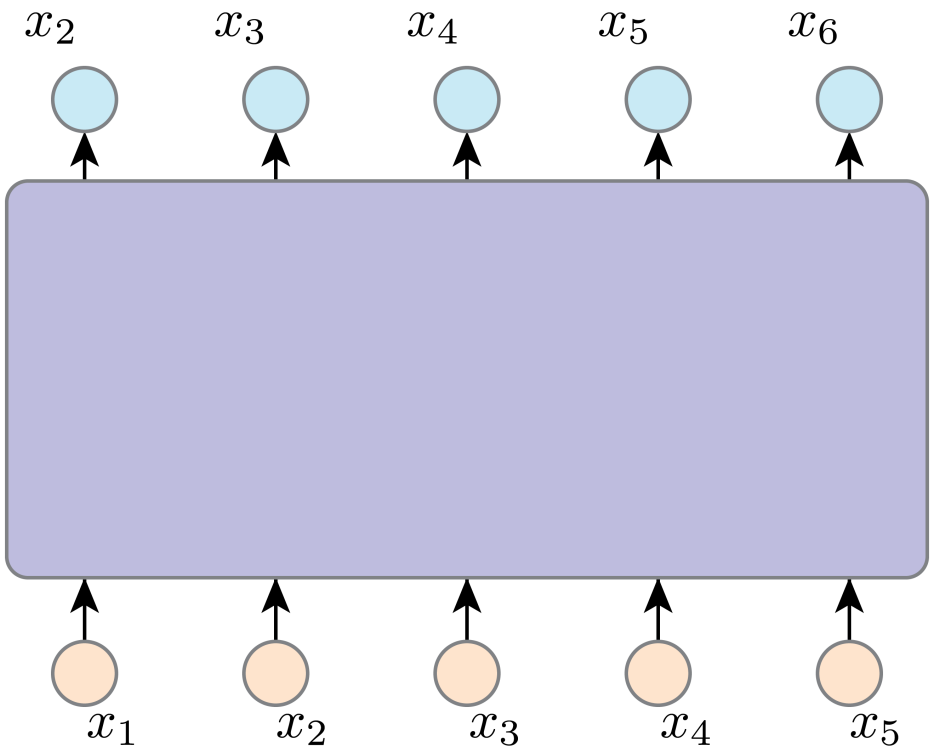


A recursive process implementing

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

Shared computation

- One single network
  - Shared architecture, weights, and computation
  - Output x_i not dependent on x_j for any j >= i

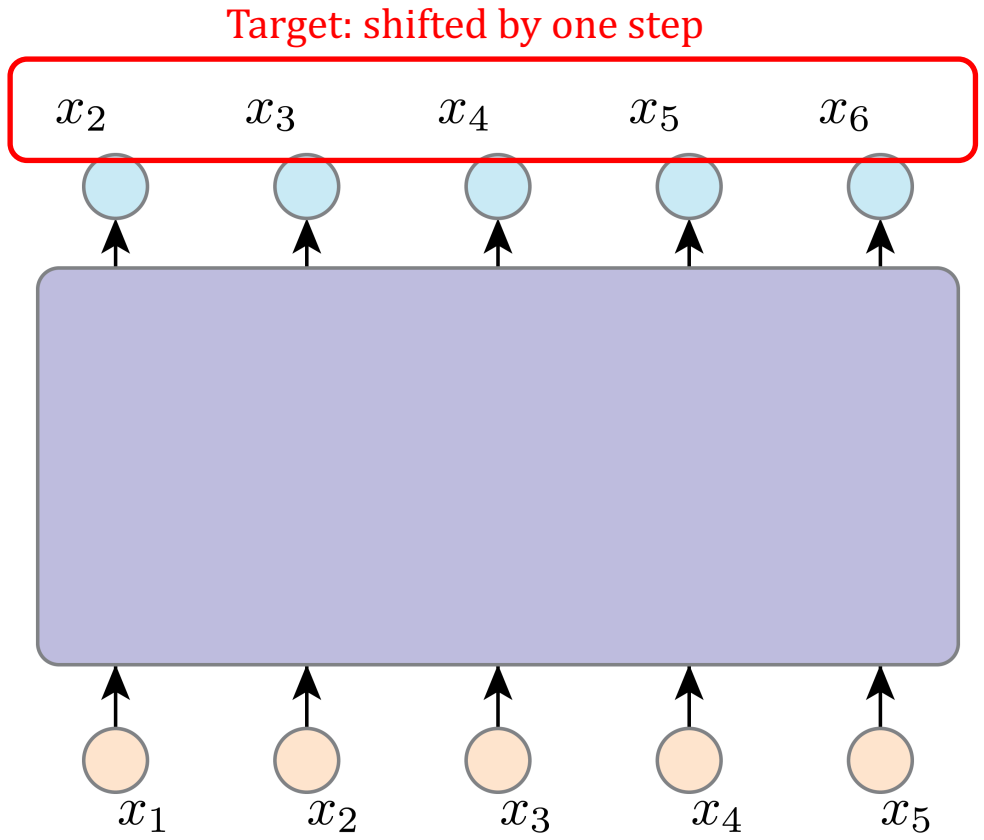


A recursive process implementing

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

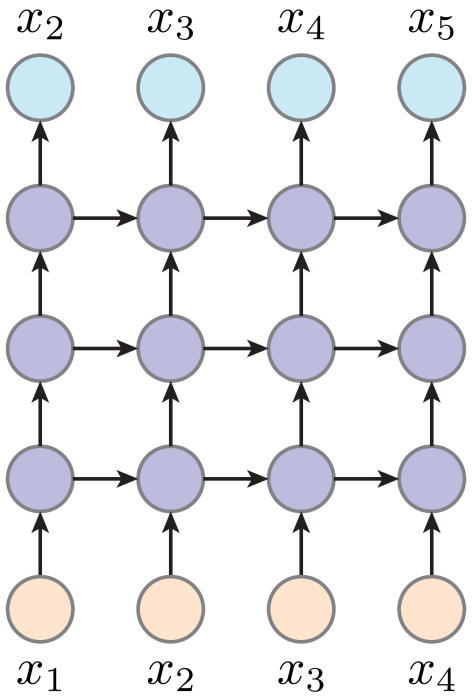
Shared computation

- One single network
  - Shared architecture, weights, and computation
  - Output x_i not dependent on x_j for any j >= i

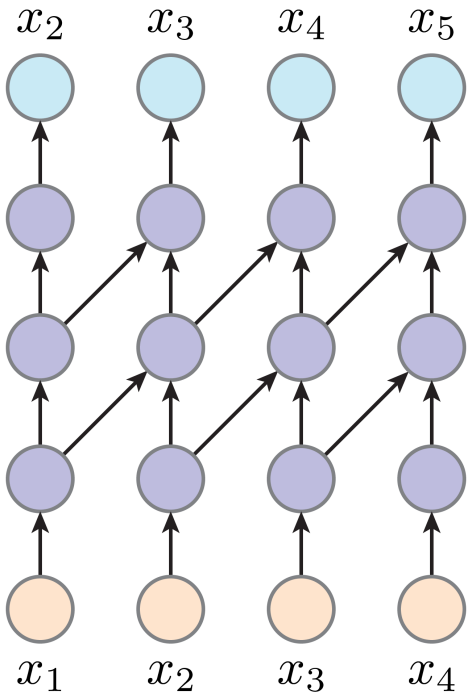


One single network

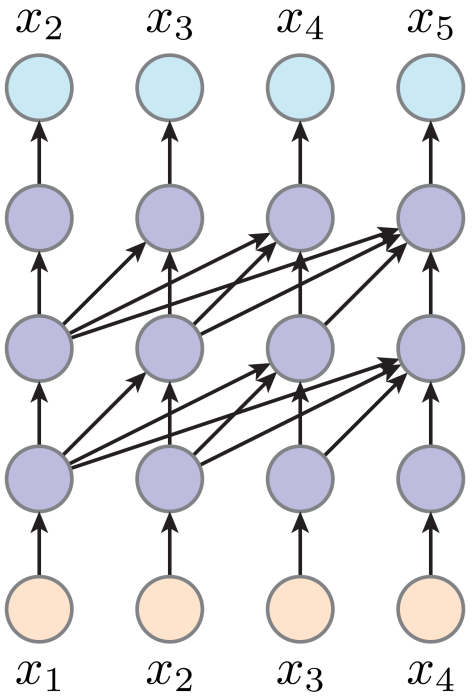
- Shared architecture, weights, and computation
- Output x_i not dependent on x_j for any j >= i



RNN



CNN

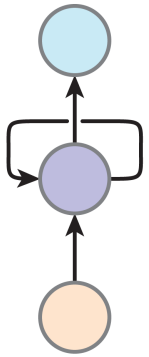


Attention

Recurrent Neural Network

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

$$y_t = W_{hy}h_t + b_y$$

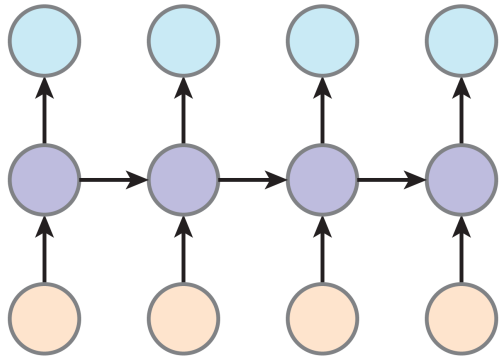


One RNN unit

Recurrent Neural Network

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

$$y_t = W_{hy}h_t + b_y$$

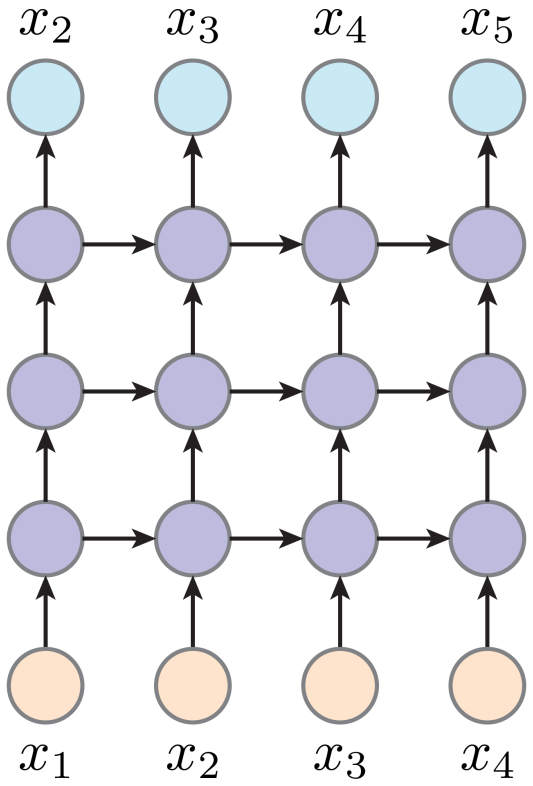


One RNN unit, unfold in time

Recurrent Neural Network

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

$$y_t = W_{hy}h_t + b_y$$

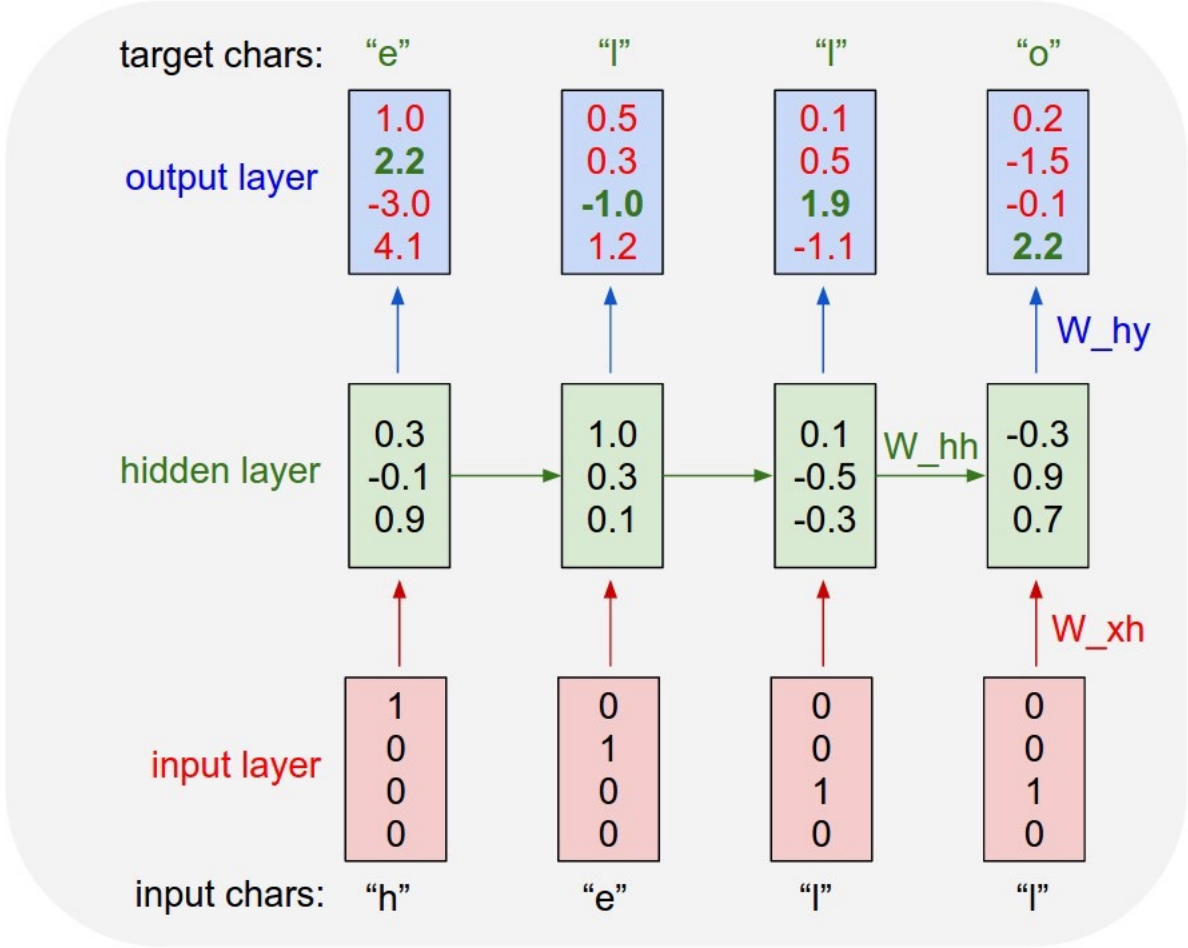


One RNN unit, unfold in time, go deeper

Recurrent Neural Network

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

$$y_t = W_{hy}h_t + b_y$$



CharRNN

https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Network

- Too simple to capture long-term dependency (catastrophic forgetting)

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

$$y_t = W_{hy}h_t + b_y$$

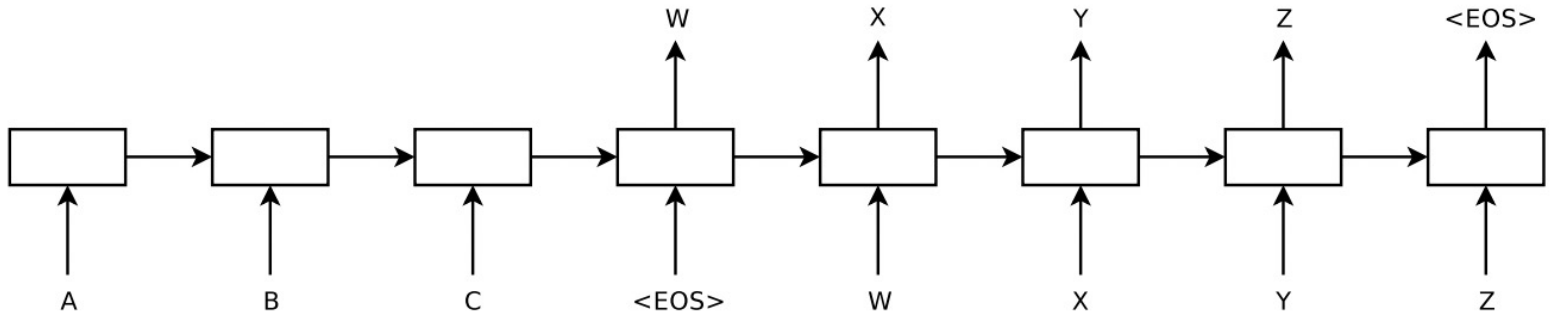
Recurrent Neural Network

- Special variant: Long Short-term Memory (LSTM)

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \quad \text{(forget gate)} \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \quad \text{(input gate)} \\
\tilde{c}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \quad \text{(cell candidate)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad \text{(cell state)} \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \quad \text{(output gate)} \\
h_t &= o_t \odot \tanh(c_t) \quad \text{(hidden state)}
\end{aligned}
$$

Recurrent Neural Network

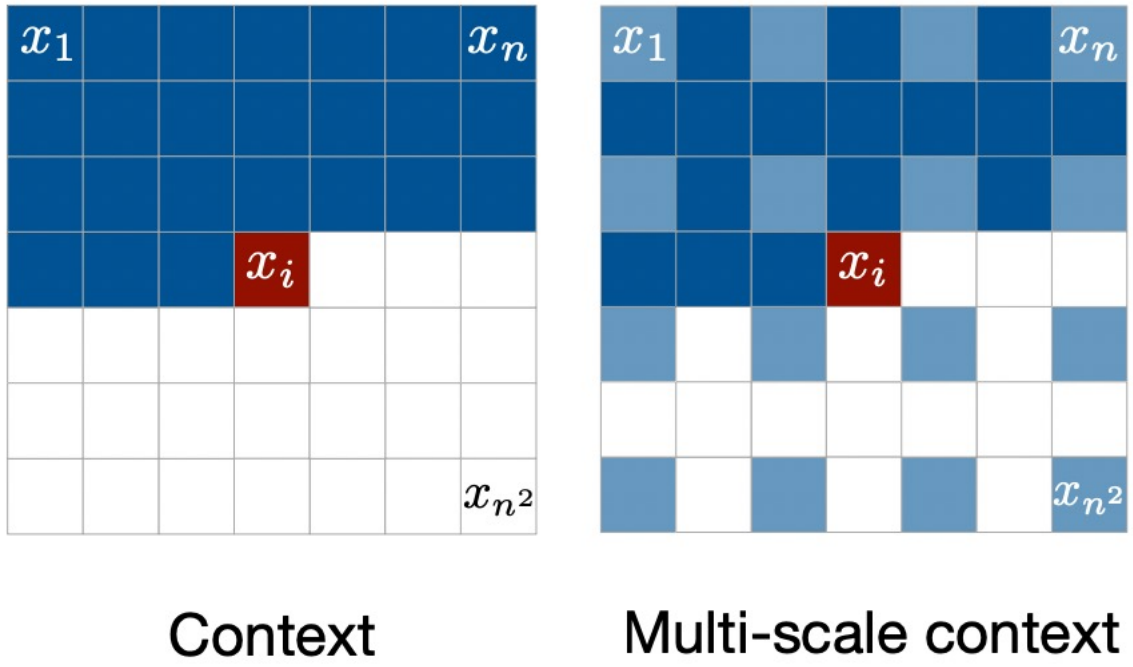
- Special variant: Long Short-term Memory (LSTM)
- A 380M-parameter model



Sequence to Sequence Learning with Neural Networks, NeurIPS 2014

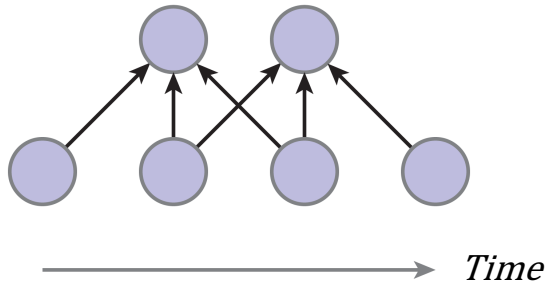
Recurrent Neural Network

- Special variant: Long Short-term Memory (LSTM)
- A 380M-parameter model



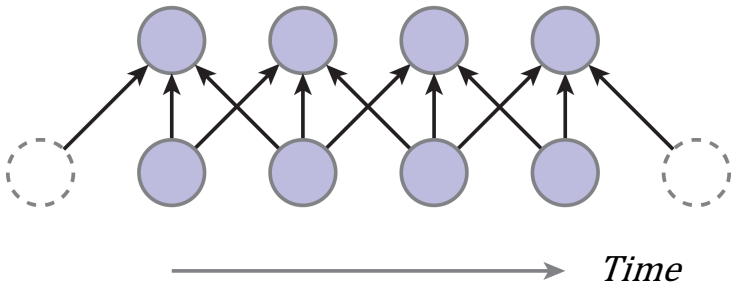
Pixel Recurrent Neural Networks, ICML 2016

Convolutional Neural Network (CNN) for AR



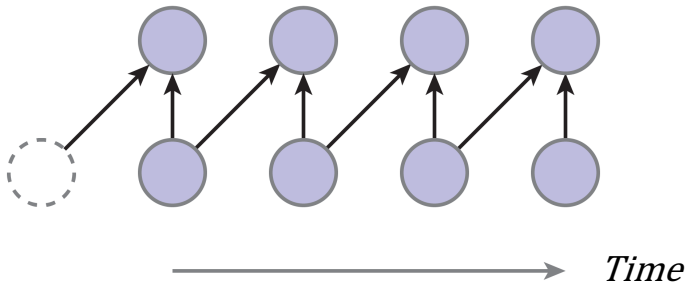
1-D Convolution

Convolutional Neural Network (CNN) for AR



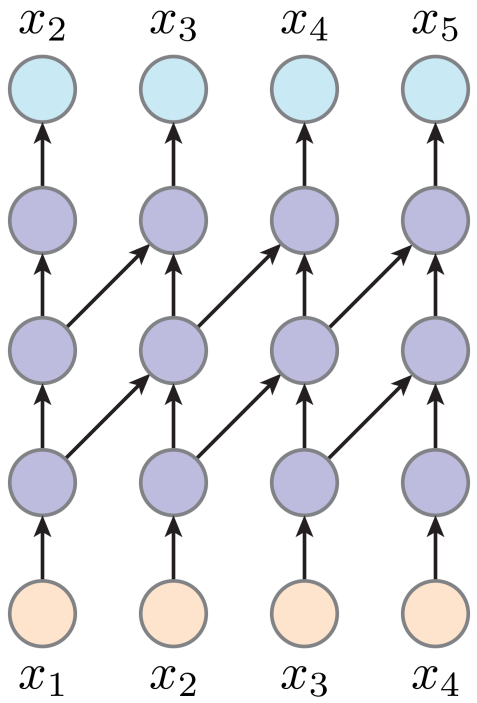
1-D Convolution with padding

Convolutional Neural Network (CNN) for AR



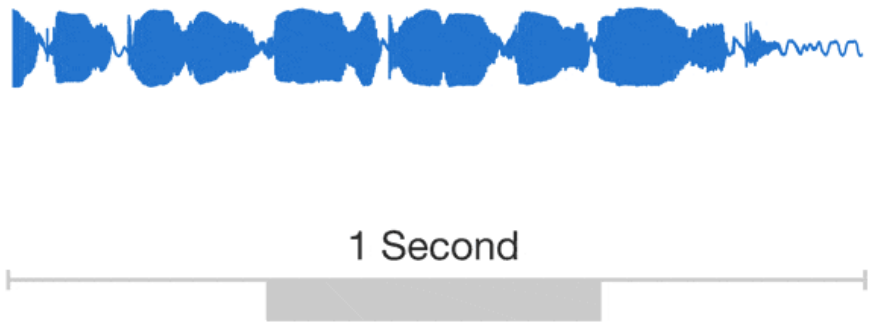
Causal 1-D Convolution with padding

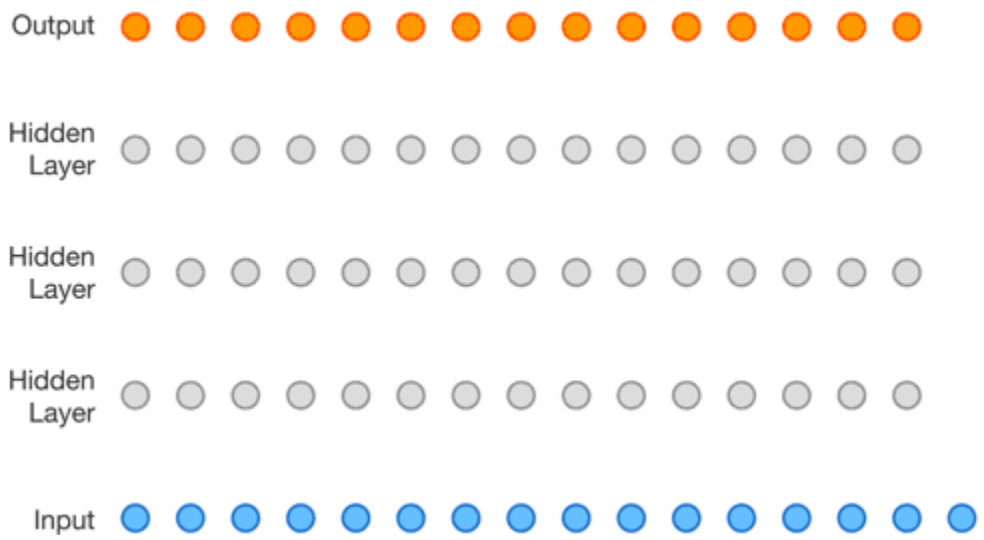
Convolutional Neural Network (CNN) for AR



Causal 1-D Convolution with padding, going deeper

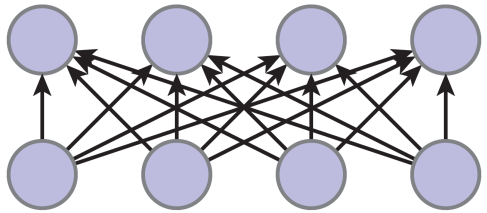
Convolutional Neural Network (CNN) for AR





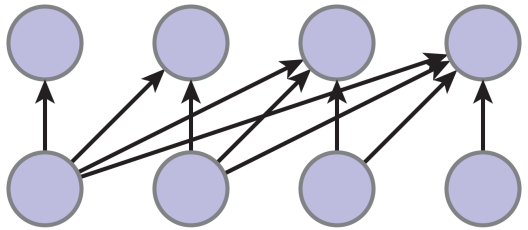
WaveNet: A Generative Model for Raw Audio, 2016

Transformer for AR



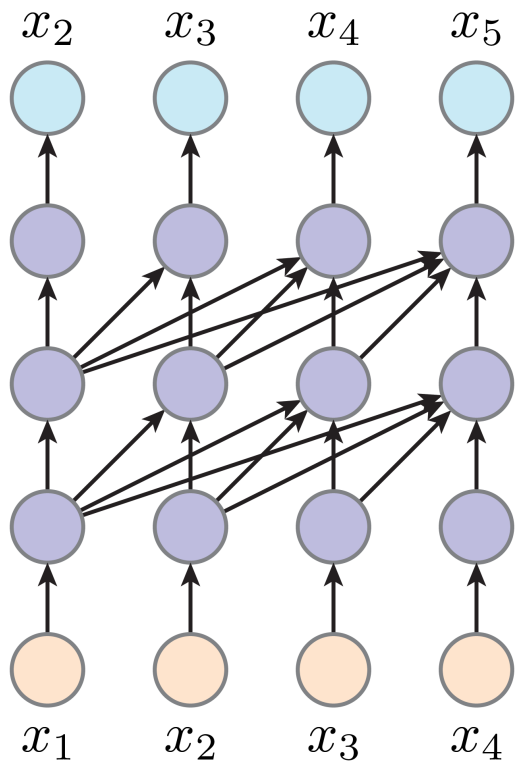
Full attention (every step sees all step)

Transformer for AR



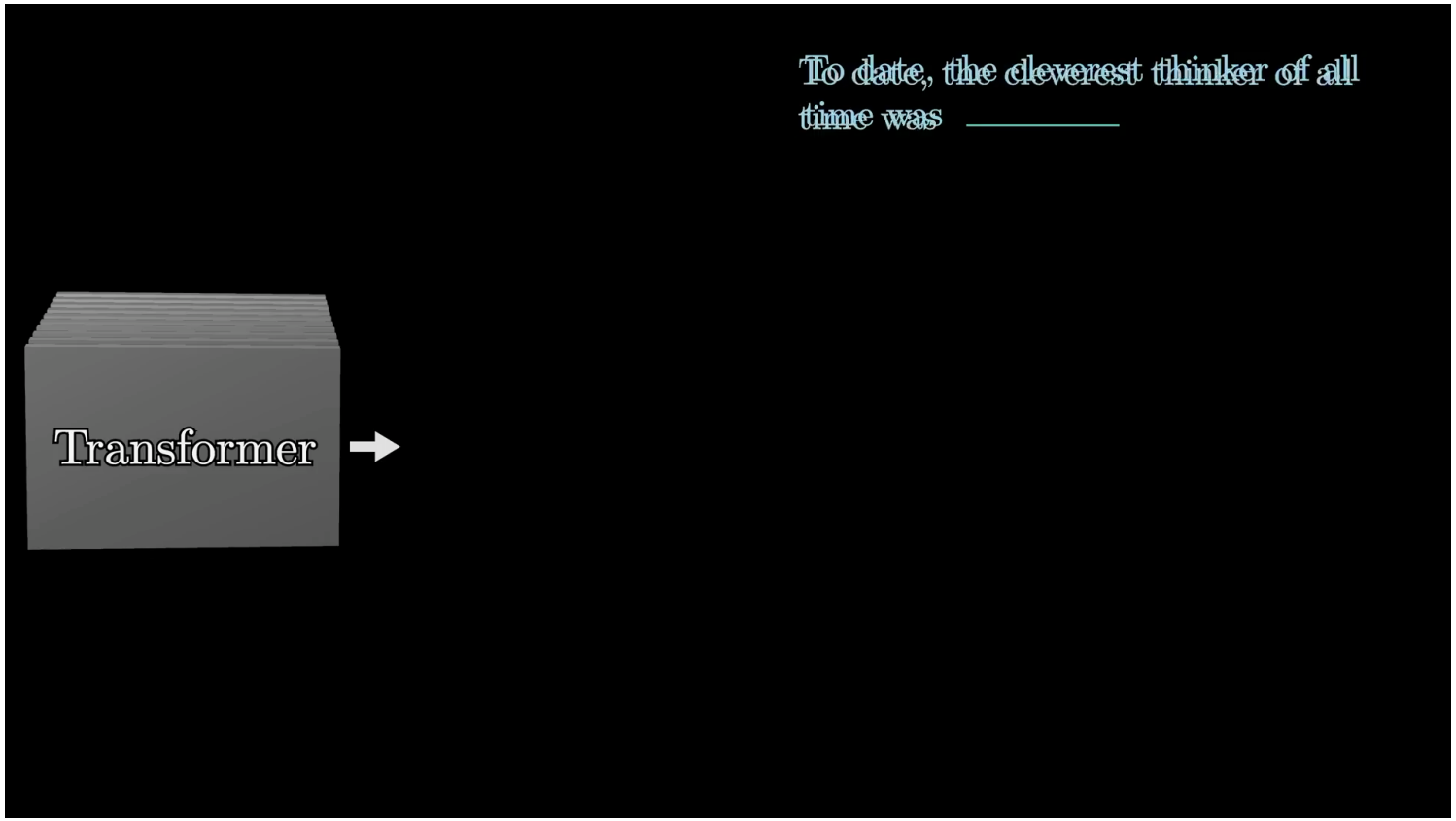
Causal attention (every step sees all previous step)

Transformer for AR



Deeper causal attention (every step sees all previous step)

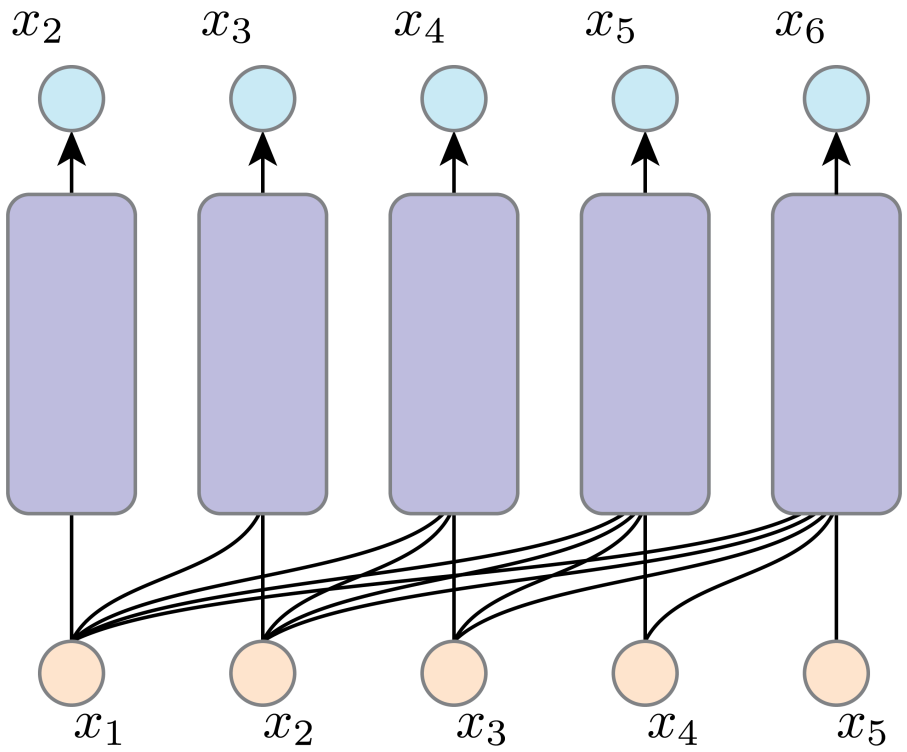
Transformer for AR



Major LLMs

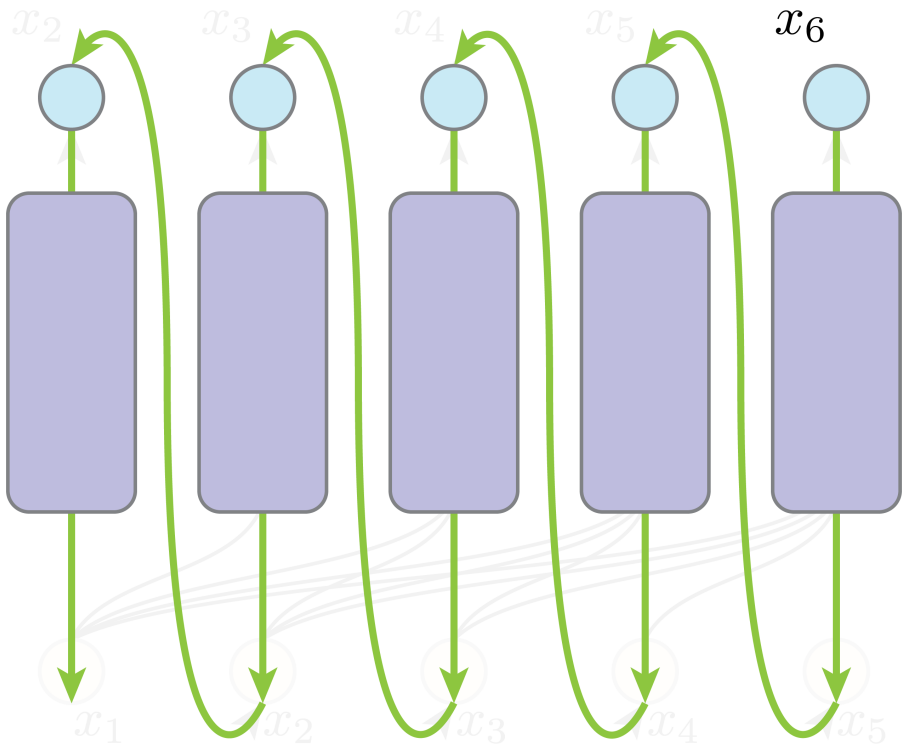
Training AR

- a recursive process implementing



Training AR

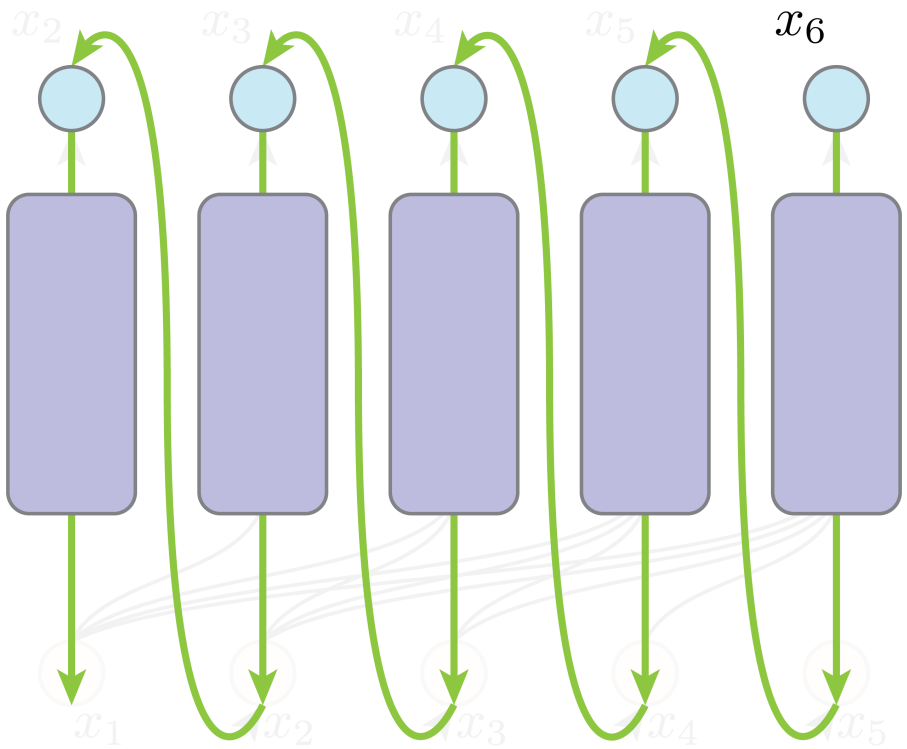
- What if we backprop through this graph “as is” ?



One gradient path

Training AR

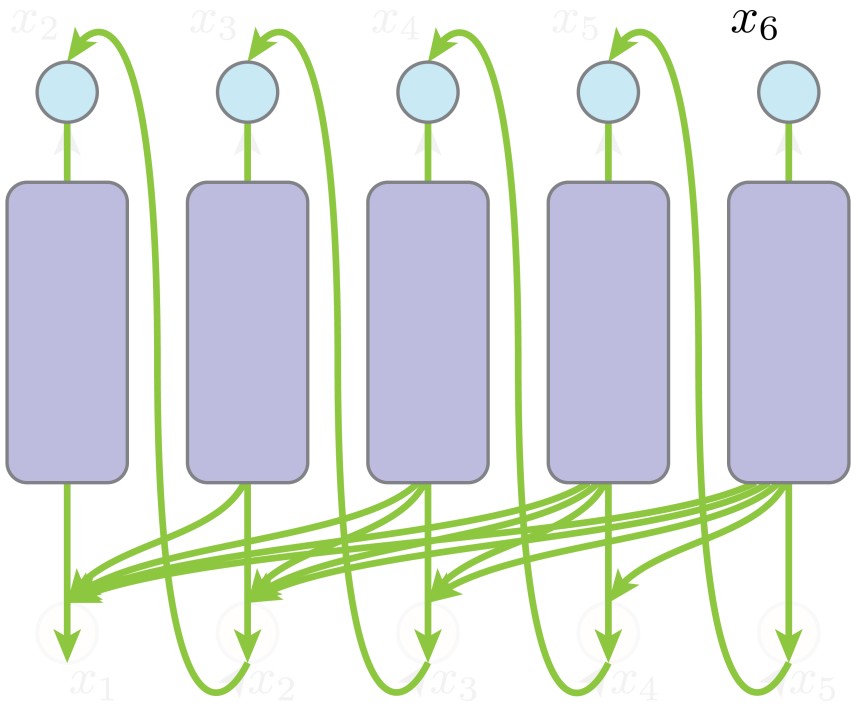
- What if we backprop through this graph "as is"
  - Go through all previous outputs, all previous sampling ops, all previous networks
  - It's infeasible to train AR "as is"



One gradient path

Training AR

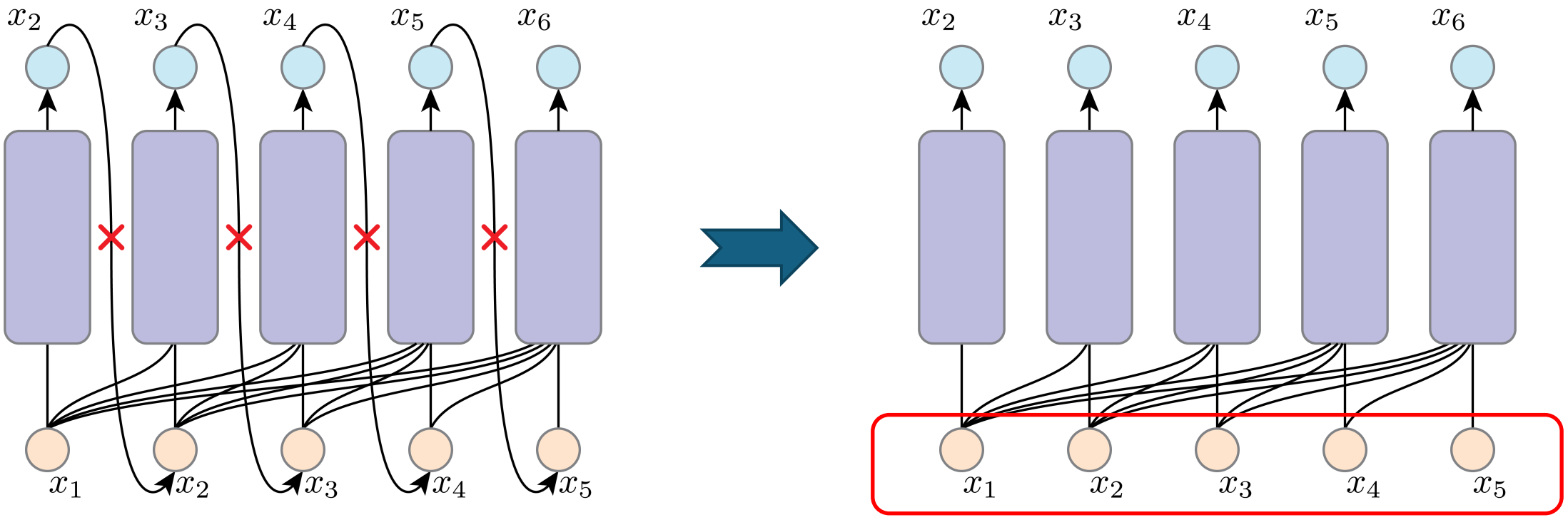
- What if we backprop through this graph “as is”
  - Go through all previous outputs, all previous sampling ops, all previous networks
  - It’s infeasible to train AR “as is”



full gradient path

Training AR: Teacher-Forcing

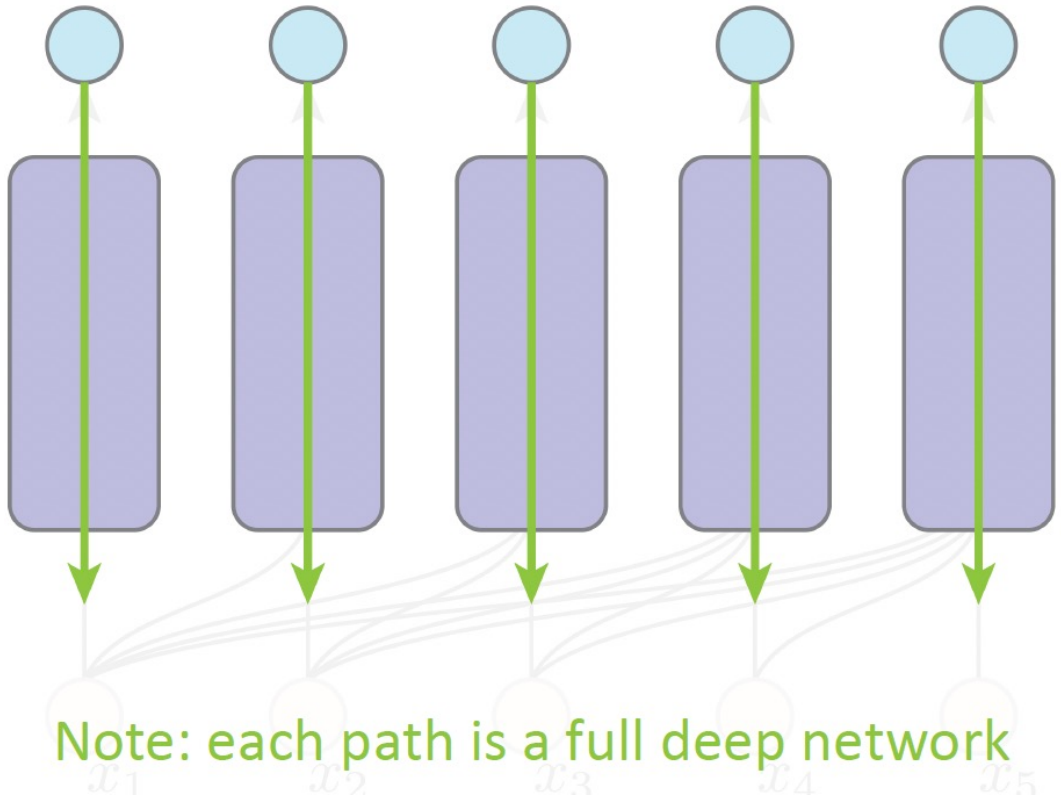
- Inputs are not from previous outputs
- Inputs are from ground-truth data



Ground-truth as inputs

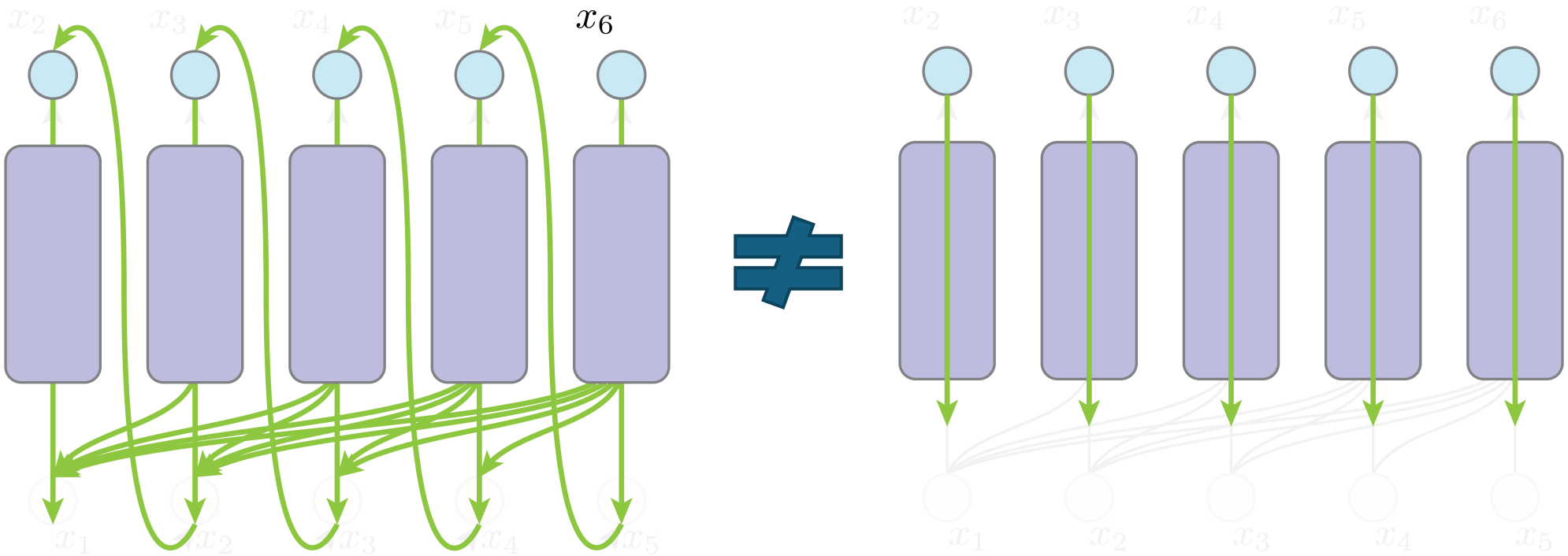
Training AR: Teacher-Forcing

- Inputs are not from previous outputs
- Inputs are from ground-truth data
- Backprop path is much shorter



Training AR: Teacher-Forcing

- Inputs are not from previous outputs
- Inputs are from ground-truth data
- Backprop path is much shorter
- Exposure bias: inconsistent training and inference, and distribution shift

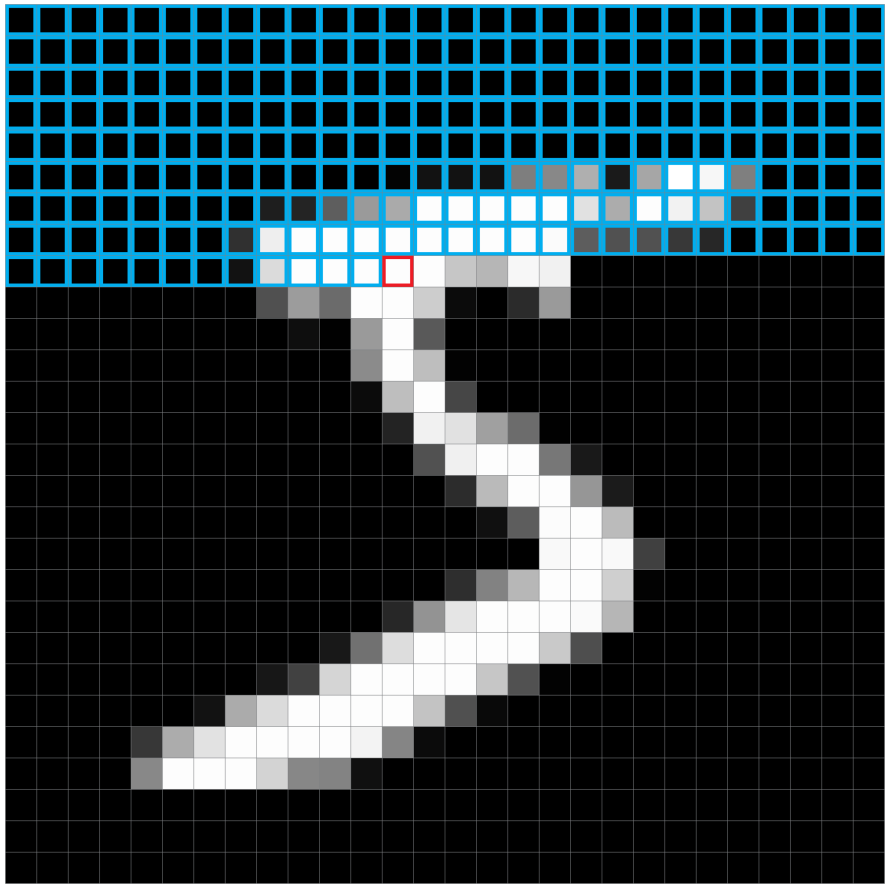


Training AR: Teacher-Forcing

- Exposure bias: inconsistent training and inference, and distribution shift
  - Two stage training

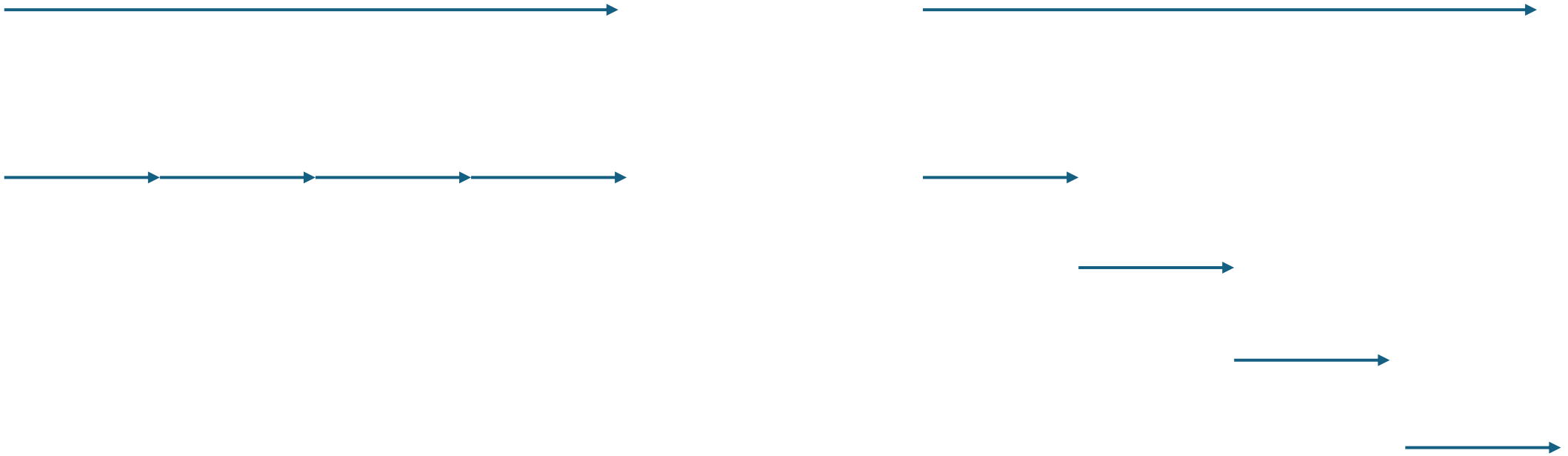
Drawbacks of AR

- Exposure bias: inconsistent training and inference, and distribution shift
- Slow generation speed



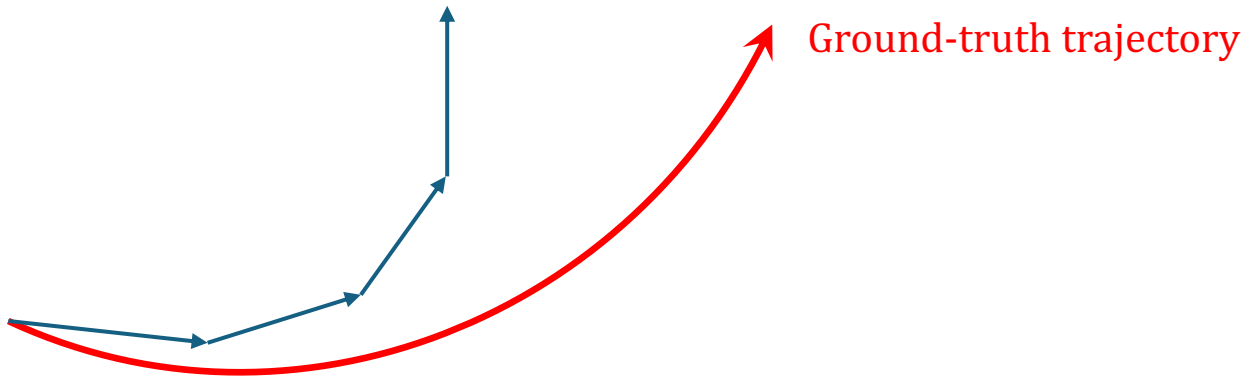
Drawbacks of AR

- Exposure bias: inconsistent training and inference, and distribution shift
- Slow generation speed
  - Hierarchical cascading, block ar, …



Drawbacks of AR

- Exposure bias: inconsistent training and inference, and distribution shift
- Slow generation speed
- Error accumulation during inference

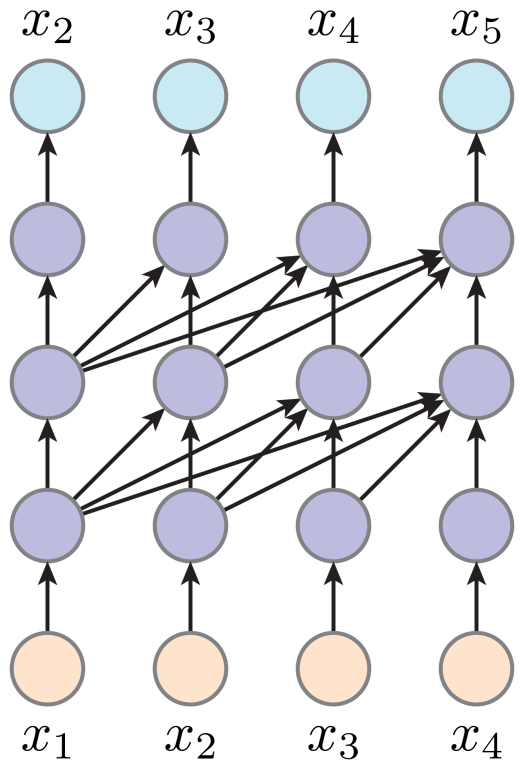


Drawbacks of AR

- Exposure bias: inconsistent training and inference, and distribution shift
- Slow generation speed
- Error accumulation during inference
  - Memory, …

Drawbacks of AR

- Exposure bias: inconsistent training and inference, and distribution shift
- Slow generation speed
- Error accumulation during inference
- Computational complexity to capture dense and long dependencies, $O(n^2)$



Drawbacks of AR

- Exposure bias: inconsistent training and inference, and distribution shift
- Slow generation speed
- Error accumulation during inference
- Computational complexity to capture dense and long dependencies, $O(n^2)$
  - Linear attention, sparse attention, …