COMP547

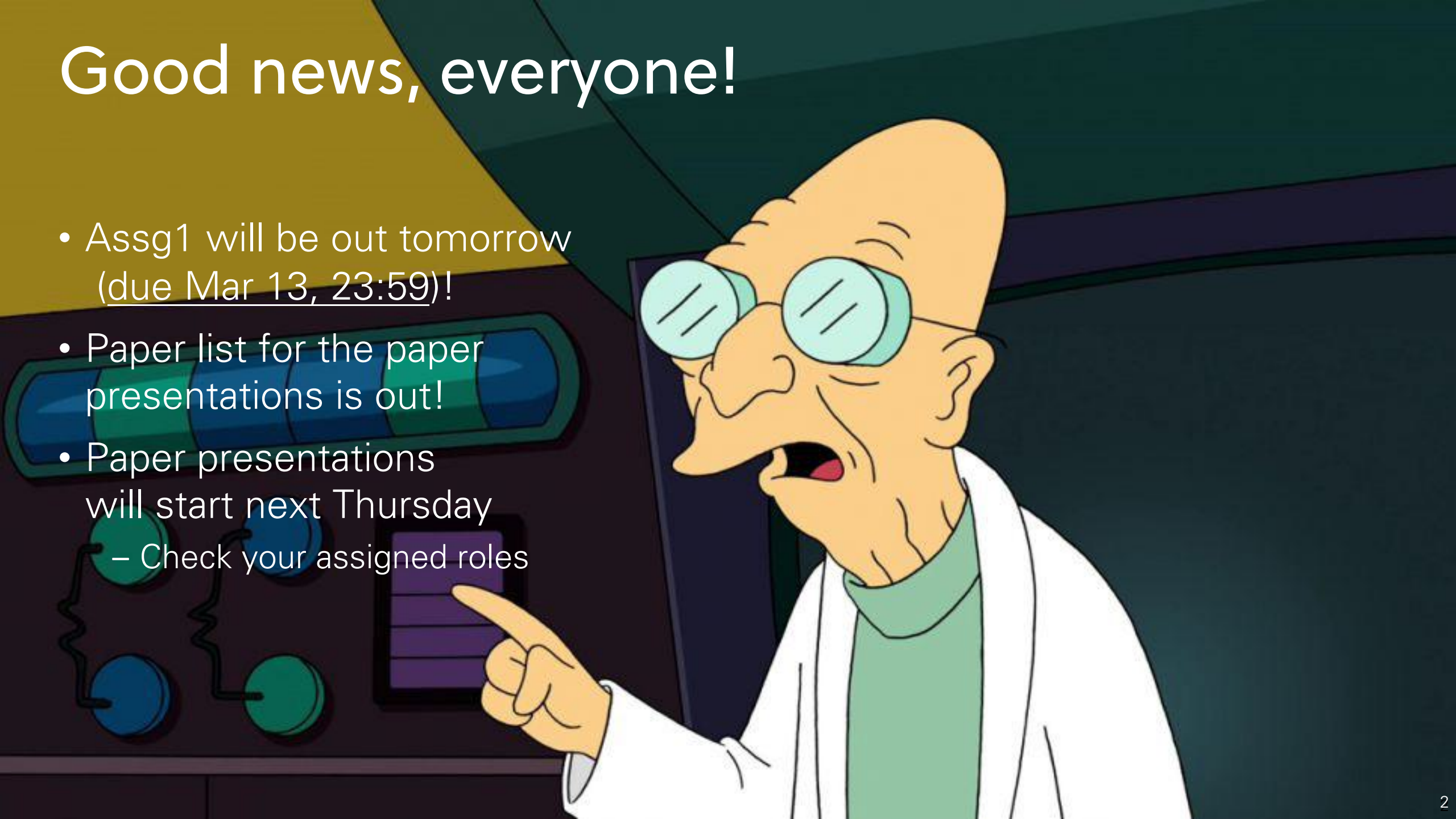
DEEP UNSUPERVISED LEARNING

Lecture #5 – Autoregressive Models

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Spring 2026

Good news, everyone!



- Assg1 will be out tomorrow (due Mar 13, 23:59)!
- Paper list for the paper presentations is out!
- Paper presentations will start next Thursday
  - Check your assigned roles

Paper Reviews

Think deeply about the papers we read and try to learn from them as much as possible (and then even more). If you do not understand something, we should discuss it and dissect it together. Whatever you think others understand, they understand less (the instructor included), but together we will get it.

- Identify the key questions the paper studies, and the answers it provides to these questions.
- Consider the challenges of the problem or scenario studied, and how the paper's approach addresses them.
- Deconstruct the formal and technical parts to understand their fine details. Note to yourself aspects that are not clear to you

Borrowed from Yoav Artzi

Paper Reviewing Guidelines

- When reviewing the paper, start with 1–2 sentences summarizing what the paper is about.
- Continue with the strength of the paper. Outline its contribution, and your main takeaways. What did you learn?
- Highlight shortcomings and limitations. Please focus on weaknesses that fundamental to the method. Unlike conference or journal reviewing, this part is intended for your understanding and discussion.
- Try to suggest ways to address the paper's limitations. Any idea is welcome and will contribute to the discussion.
- Suggest questions for discussion in class. As part of the discussion in class, you are asked to raise these questions during the class.

Borrowed from Yoav Artzi

Previously on COMP547

- memory
- tokens
- attention
- positional encoding
- transformers vs. attention in RNNs

Lecture overview

- motivation
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
- other things to be aware of

Disclaimer: Much of the material and slides for this lecture were borrowed from

—Pieter Abbeel, Wilson Yan, Kevin Frans, Philipp Wu's Berkeley CS294-158 class

—Kaiming He's MIT 6.S978 class

Lecture overview

- **motivation**
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
- other things to be aware of

Likelihood-based models

- **Problems we'd like to solve:**
  - Generate data: synthesize images, videos, speech, text
  - Compress data: construct efficient codes
  - Detect anomalies: i.e. data that is out of distribution

- **Likelihood-based models**
  - **Estimate p_{data} from samples $x^{(1)}, \ldots, x^{(n)} \sim p_{data}(x)$**

- Learns a distribution p that allows:
  - Computing p(x) for arbitrary x
  - Sampling x ~ p(x)

- Today: **discrete** data

Desiderata

- We want to estimate distributions of **complex, high-dimensional data**
  - A 128×128×3 image lies in a ~50,000-dimensional space
- We also want computational and statistical efficiency
  - Efficient training and model representation
  - Expressiveness and generalization
  - Sampling quality and speed
  - Compression rate and speed

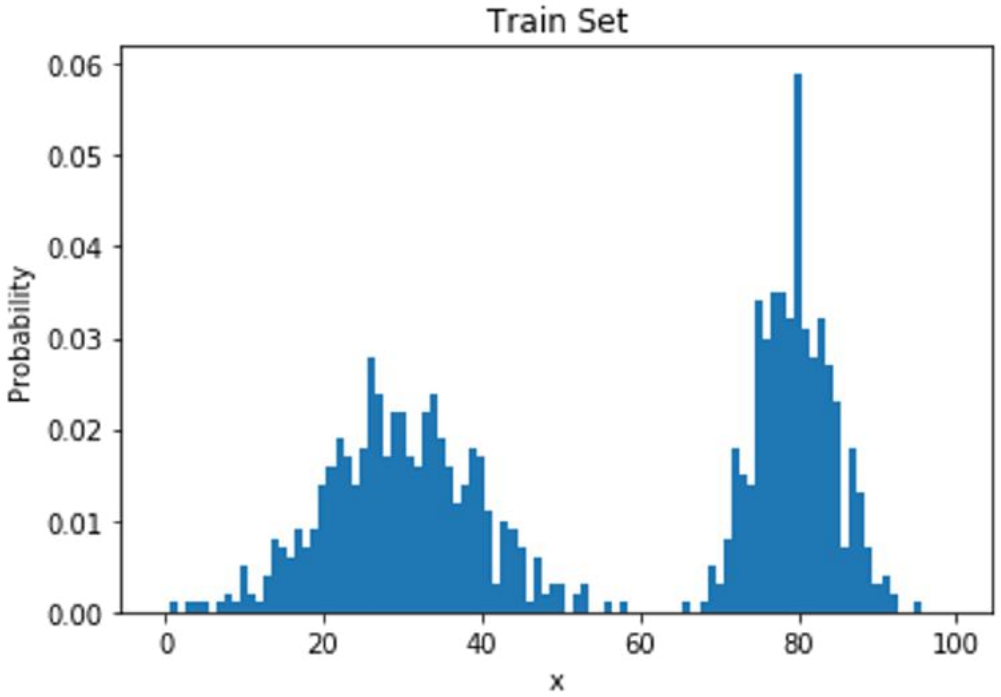
Lecture overview

- Motivation
- 1-Dimensional Distributions
  - Simplest generative model: histogram
  - Parameterized distributions and maximum likelihood
- High-Dimensional Distributions
- Deeper Dive into Causal Masked Neural Models
- Other things to be aware of

Learning: Estimate frequencies by counting

- Recall: the goal is to estimate $\mathbf{p_{data}}$ **from samples** $\mathbf{x^{(1)}, \ldots, x^{(n)} \sim p_{data}(x)}$
- **Suppose** the samples take on values in a finite set {1, …, k}
- The model: a **histogram**
  - (Redundantly) described by k nonnegative numbers: $p_1, \ldots, p_k$
  - To train this model: count frequencies

    p_i = (# times i appears in the dataset) / (# points in the dataset)



Inference and Sampling

- **Inference** (querying p_i for arbitrary i): simply a lookup into the array $p_1, \ldots, p_k$

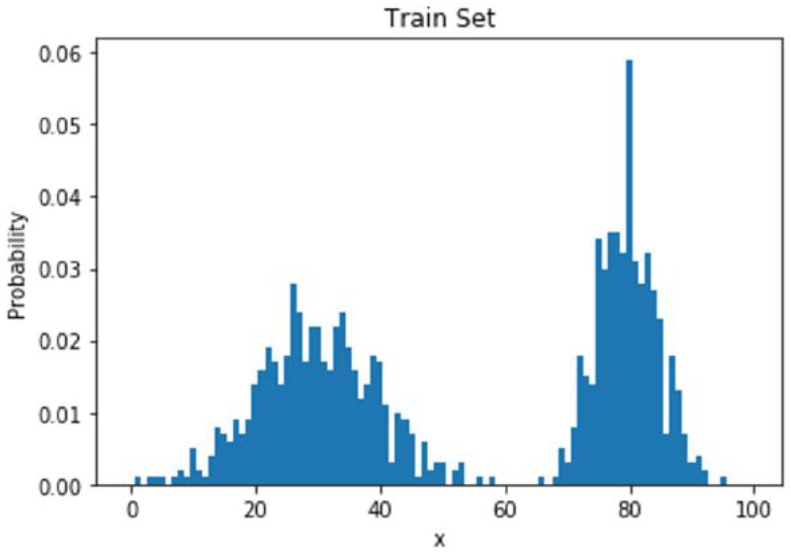
- **Sampling** (lookup into the inverse cumulative distribution function)
  1. From the model probabilities $p_1, \ldots, p_k$, compute the <u>cumulative distribution</u>

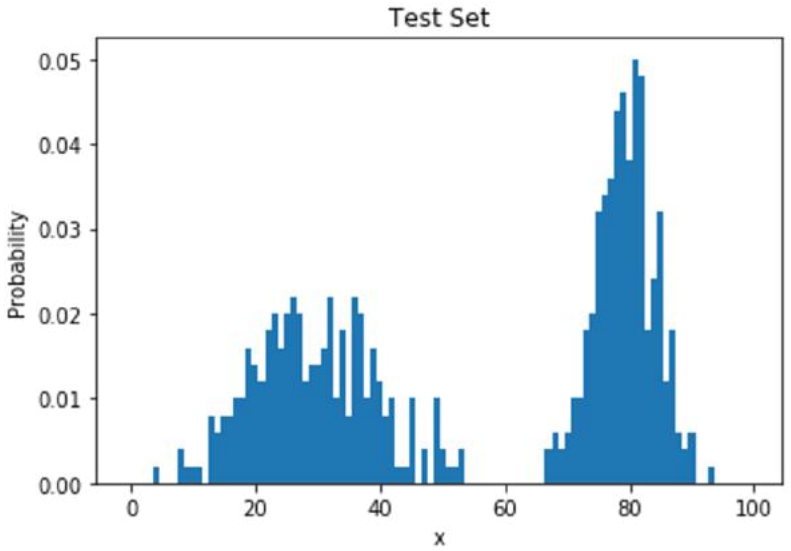
     $$F_i = p_1 + \cdots + p_i \quad \text{for all } i \in \{1, \ldots, k\}$$

  2. Draw a uniform random number $u \sim [0, 1]$
  3. Return the smallest i such that $u \leq F_i$

- **Are we done?**

Problematic even for single variable

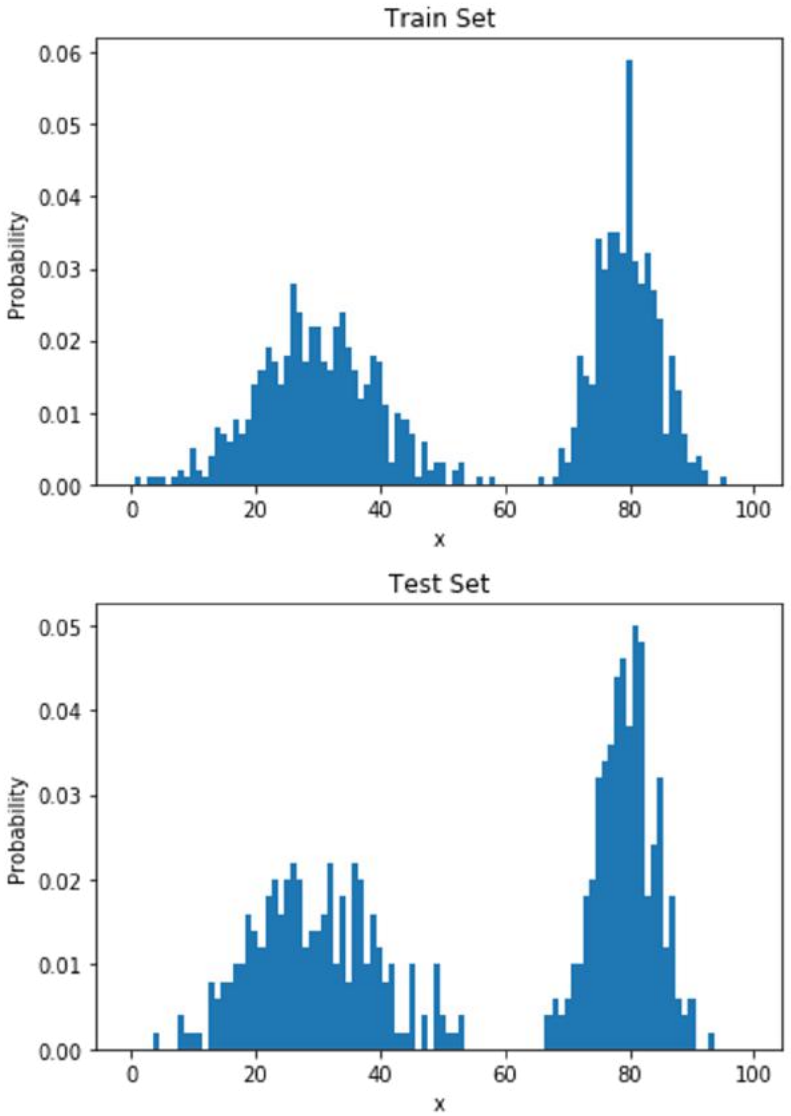


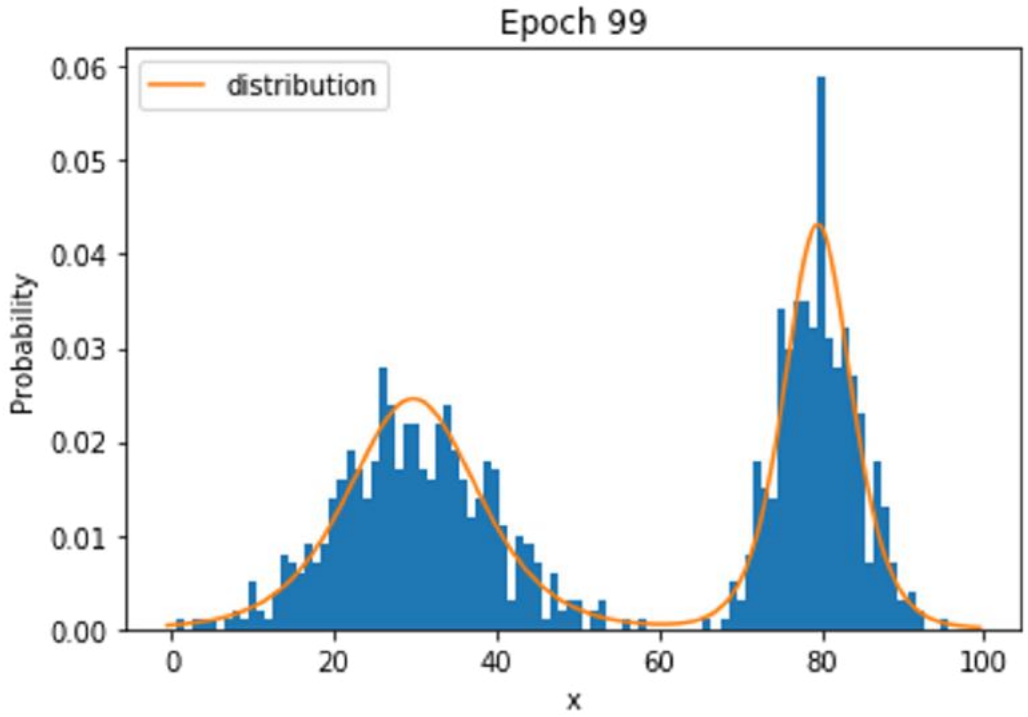


learned histogram = training data distribution

→ often poor generalization

Parameterized distributions





Fitting a parameterized distribution often generalizes better

Likelihood-based generative models

- Recall: the goal is to **estimate p_{data} from $x^{(1)}, \ldots, x^{(n)} \sim p_{data}(x)$**
- We introduce a **parameterized** model $p_\theta(x)$ with the goal to learn θ so that $p_\theta(x) \approx p_{data}(x)$
- To learn θ, we pose a search problem over parameters

$$\arg\min_{\theta} \ \text{loss}(\theta, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)})$$

- Want the loss function + search procedure to:
  - Work with large datasets (n is large, say millions of training examples)
  - Yield θ such that p_θ matches pdata — i.e. the training algorithm works. Think of the loss as a distance between distributions.
  - Note that the training procedure can only see the empirical data distribution, not the true data distribution: we want the model to generalize.

Maximum likelihood

- Maximum likelihood: given a dataset $x^{(1)}, \ldots, x^{(n)}$, find θ by solving the optimization problem

$$\arg\min_{\theta} \ \text{loss}(\theta, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}) = \frac{1}{n} \sum_{i=1}^{n} -\log p_\theta(\mathbf{x}^{(i)})$$

- Statistics tells us that if the model family is expressive enough and if enough data is given, then solving the maximum likelihood problem will yield parameters that generate the data
- Equivalent to minimizing KL divergence between the empirical data distribution and the model

$$\hat{p}_{\text{data}}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[\mathbf{x} = \mathbf{x}^{(i)}]$$

$$\text{KL}(\hat{p}_{\text{data}} \,\|\, p_\theta) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}}[-\log p_\theta(\mathbf{x})] - H(\hat{p}_{\text{data}})$$

Stochastic gradient descent

- Maximum likelihood is an optimization problem. How do we solve it?
- **Stochastic gradient descent** (SGD).
  - SGD minimizes expectations: for f a differentiable function of θ, it solves

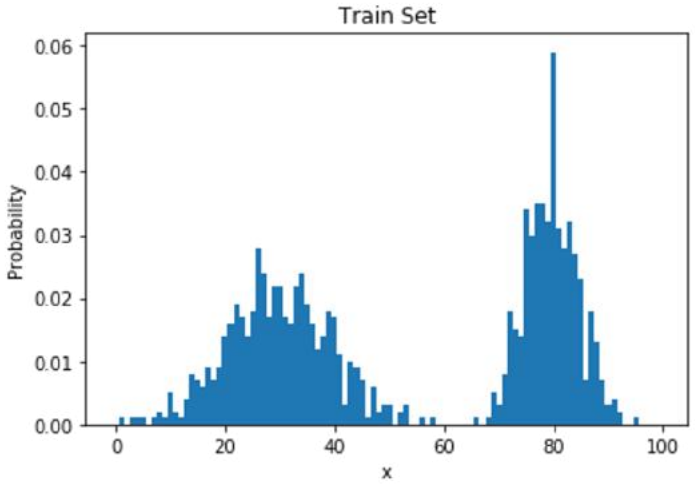
$$\arg\min_{\theta} \mathbb{E}[f(\theta)]$$

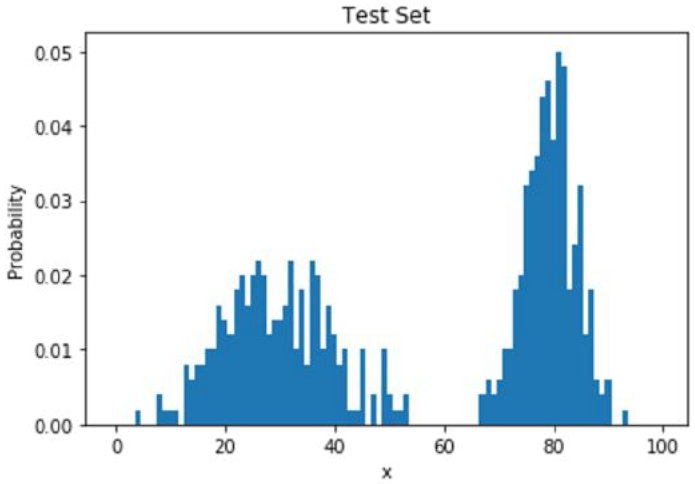
  - With maximum likelihood, the optimization problem is

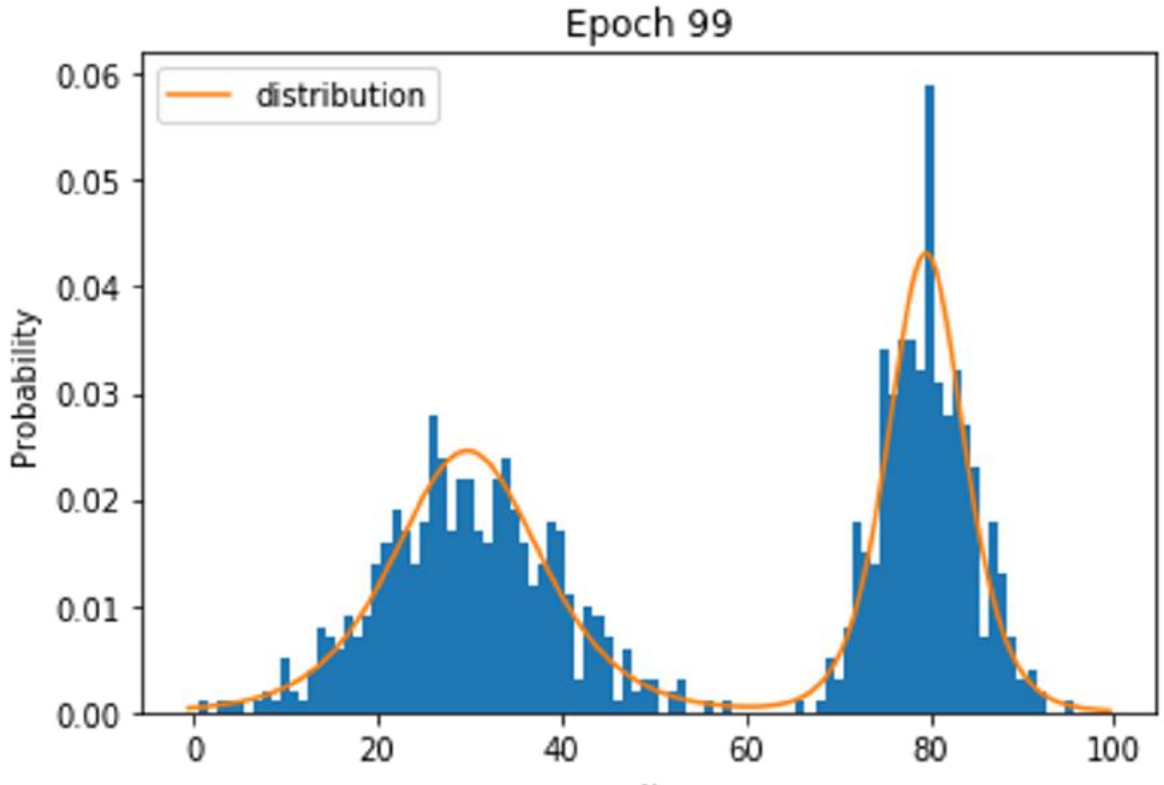
$$\arg\min_{\theta} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}[-\log p_{\theta}(\mathbf{x})]$$

  - **Why maximum likelihood + SGD?** It works with large datasets and is compatible with neural networks.

Our example earlier was the result of fitting parameterized distribution with maximum likelihood





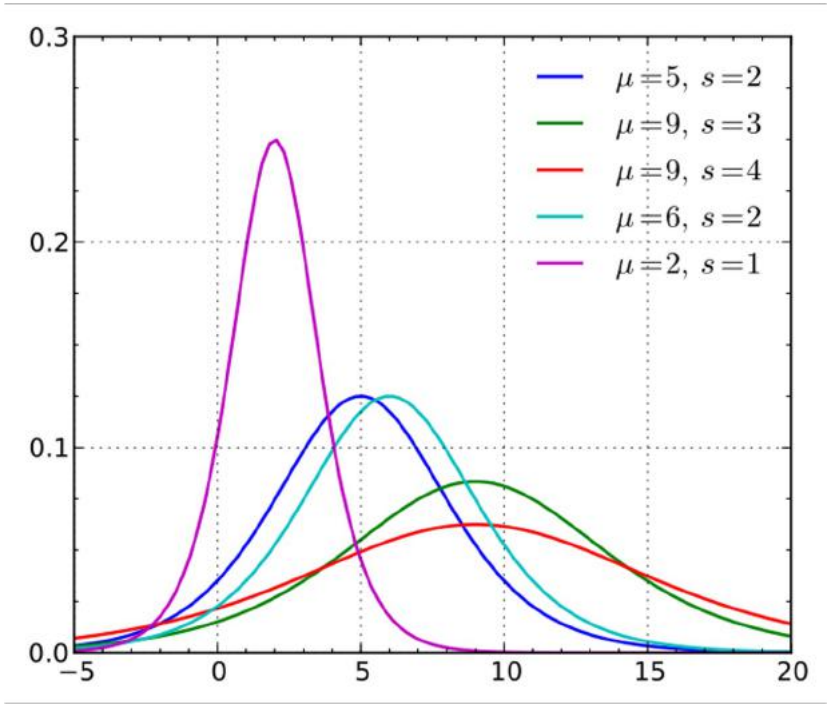


Fitting a parameterized distribution often generalizes better

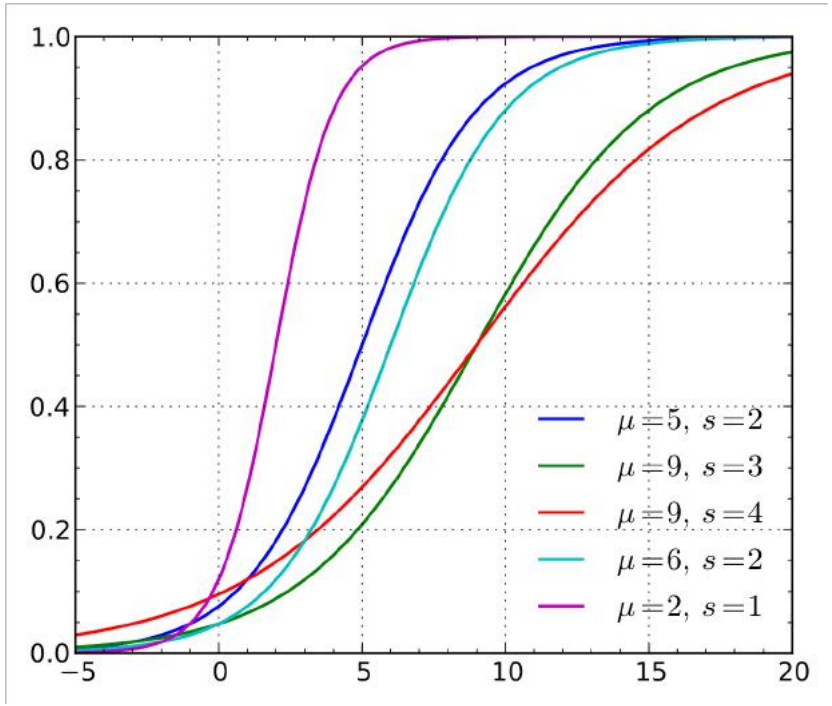
How to represent the distribution?

Data discrete, (many) smooth parameterized distributions continuous

→ Want easy access to cumulative distribution to allow for exact computation of probability mass in each interval + exact backprop → **a good choice: logistic distribution**
PS: alternatively take peace with just using the probability density at center of interval times width, but inexactness can often make debugging harder and could lead to learning (poorly performing) densities with extreme peaks



pdf



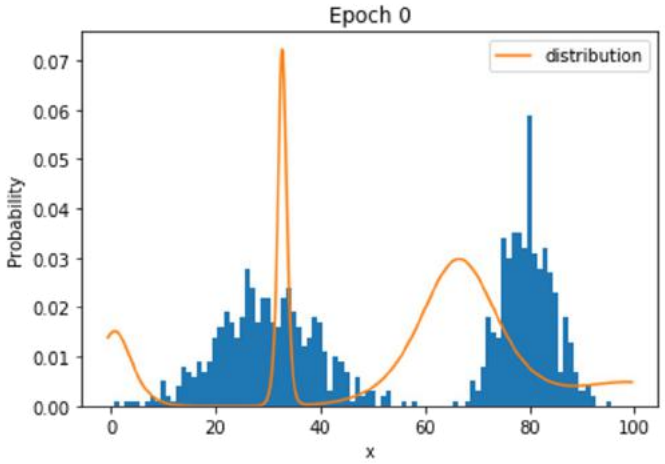
cdf = sigmoid((x - mu) / scale)

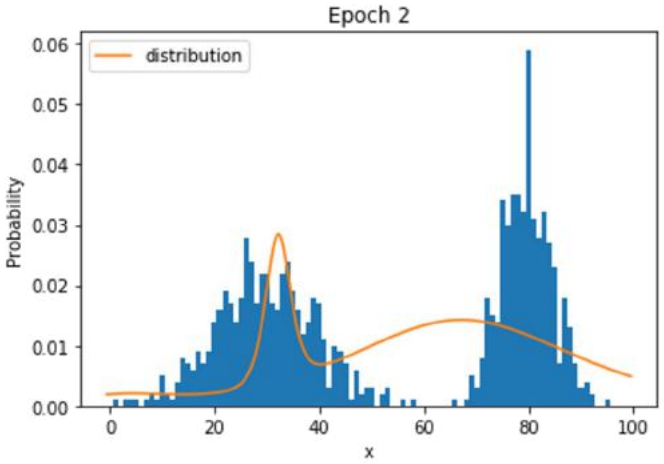
Mixture of Logistics for Multi-modal

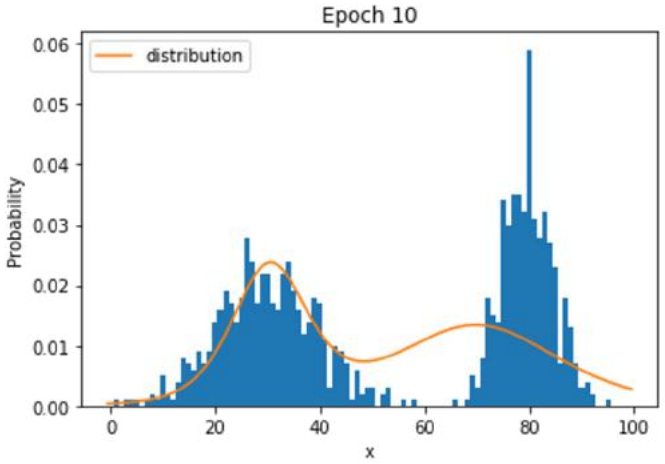
$$\nu \sim \sum_{i=1}^{K} \pi_i \text{logistic}(\mu_i, s_i)$$

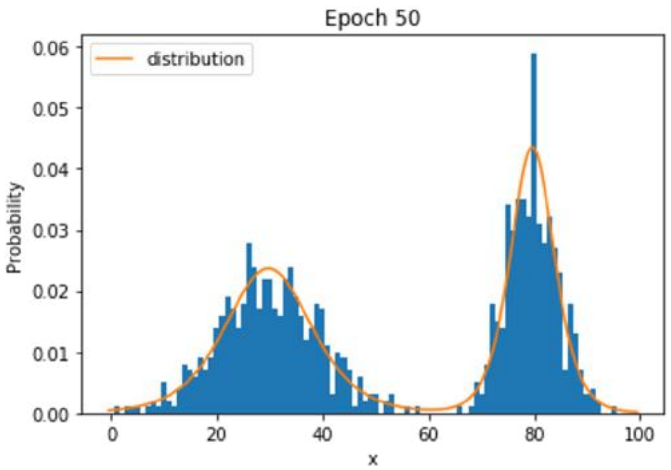
$$P(x|\pi, \mu, s) = \sum_{i=1}^{K} \pi_i \left[\sigma((x + 0.5 - \mu_i)/s_i) - \sigma((x - 0.5 - \mu_i)/s_i)\right],$$

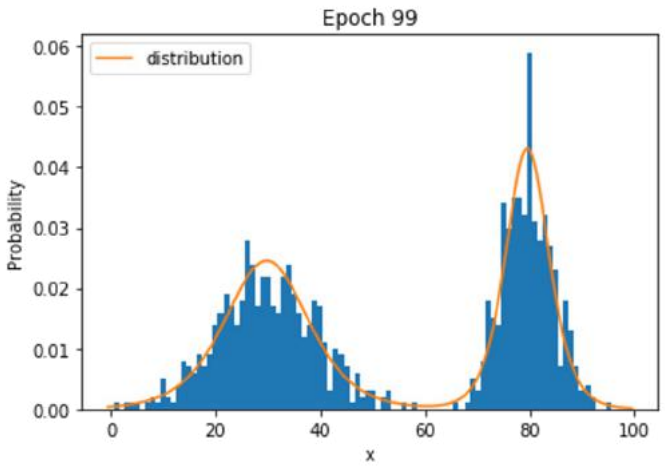
Ex. Training Mixture of Logistics

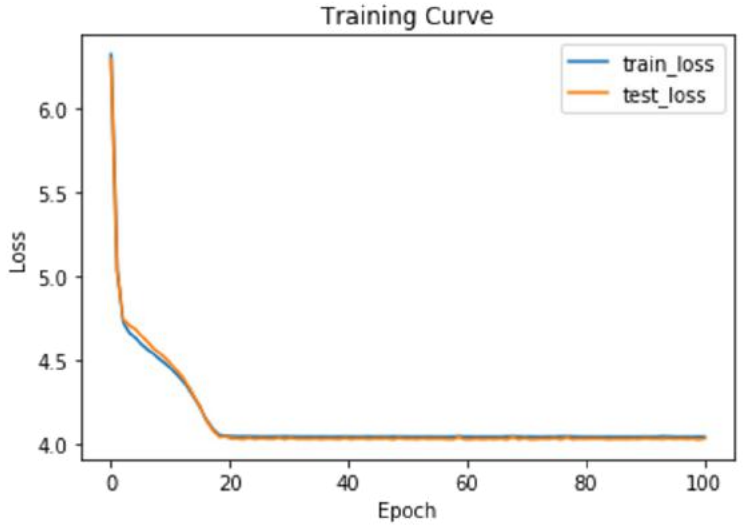












Lecture overview

- motivation
- 1-dimensional distributions
- **high-dimensional distributions**
  - **chain rule**
  - **“practical” incarnations: Bayes’ Nets, MADE, Causal Masked Neural Models, RNNs**
- deeper dive into causal masked neural models
- other things to be aware of

Challenge of high dimensions

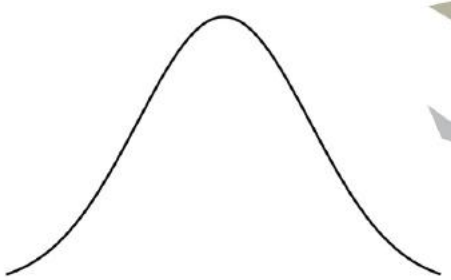
- Simply flattening into 1-D distribution doesn't work there are too many bins, even for simple cases:
  - (Binary) MNIST: 28x28 images, each pixel in {0, 1}
  - There are $2^{784} \approx 10^{236}$ probabilities to estimate
  - Any reasonable training set covers only a tiny fraction of this
  - Each image influences only one parameter. No generalization whatsoever!

Joint Distribution

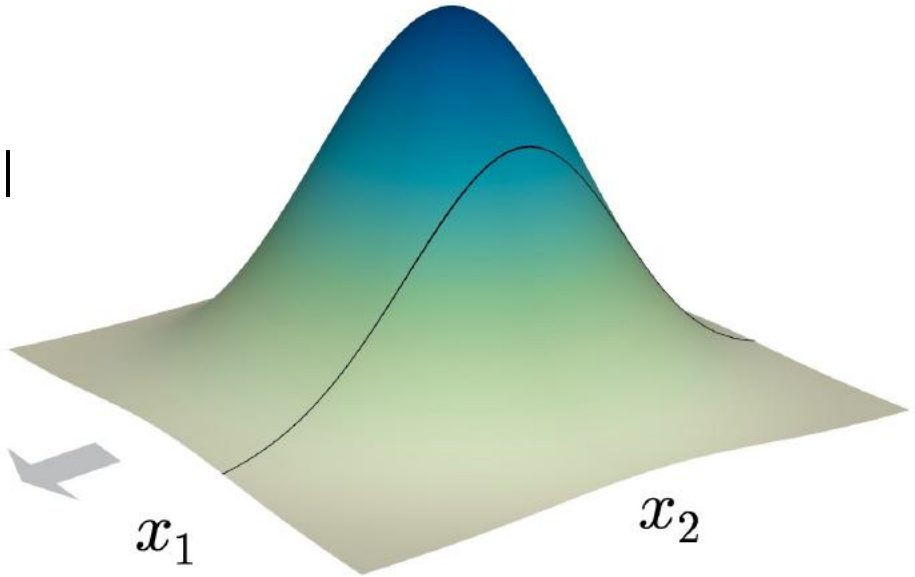
- It's convenient to model joint distributions by **independent** distributions

conditional = marginal

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

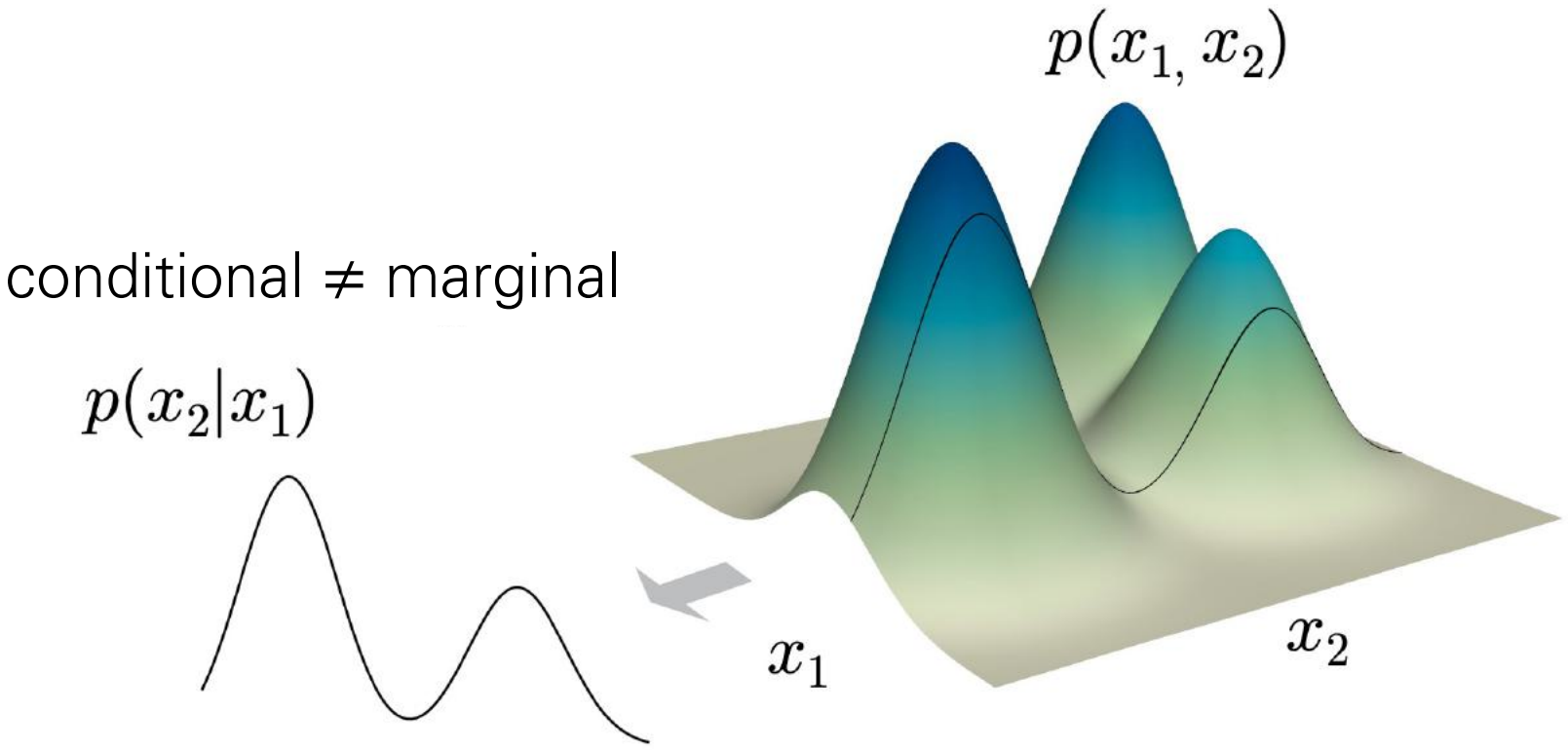


$$p(x_1, x_2)=p(x_1)p(x_2)$$



Joint Distribution

- Real-word problems always involve dependent variables

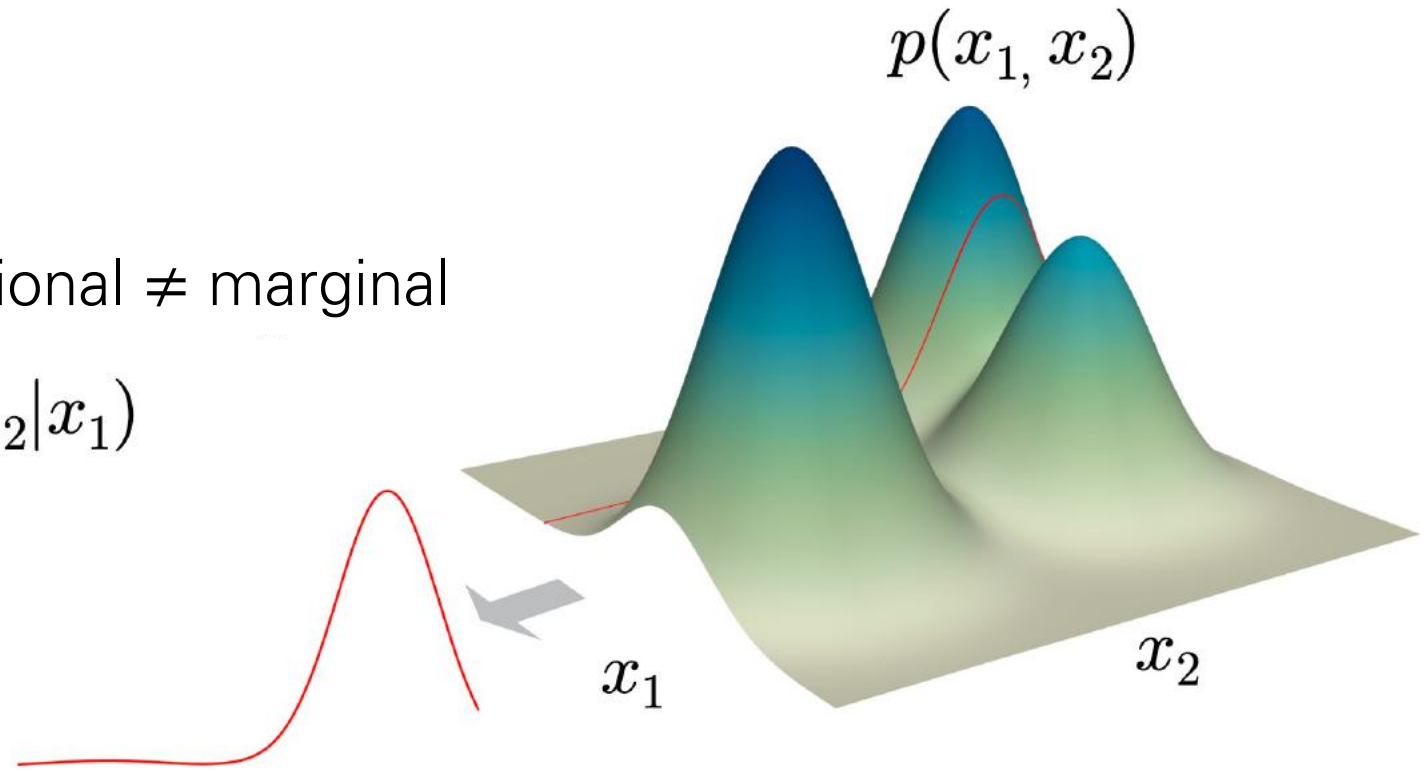


Joint Distribution

- Real-word problems always involve dependent variables

conditional ≠ marginal

$p(x_2|x_1)$



How to Model Joint Distributions?

- **Solution 1:** Modeling by **conditional** distributions

- **Solution 2:** Modeling by **independent** latents

More on this later!

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

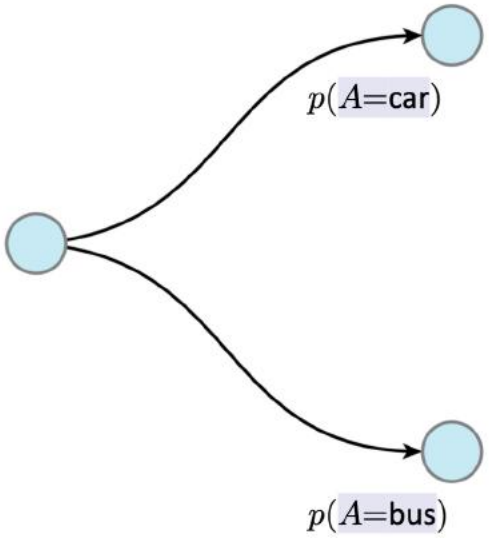
- **Chain Rule:**
  Any joint distribution can be written as a product of conditionals

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

Conditional Distribution Modeling

- **Chain Rule:**
  Any joint distribution can be written as a product of conditionals

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

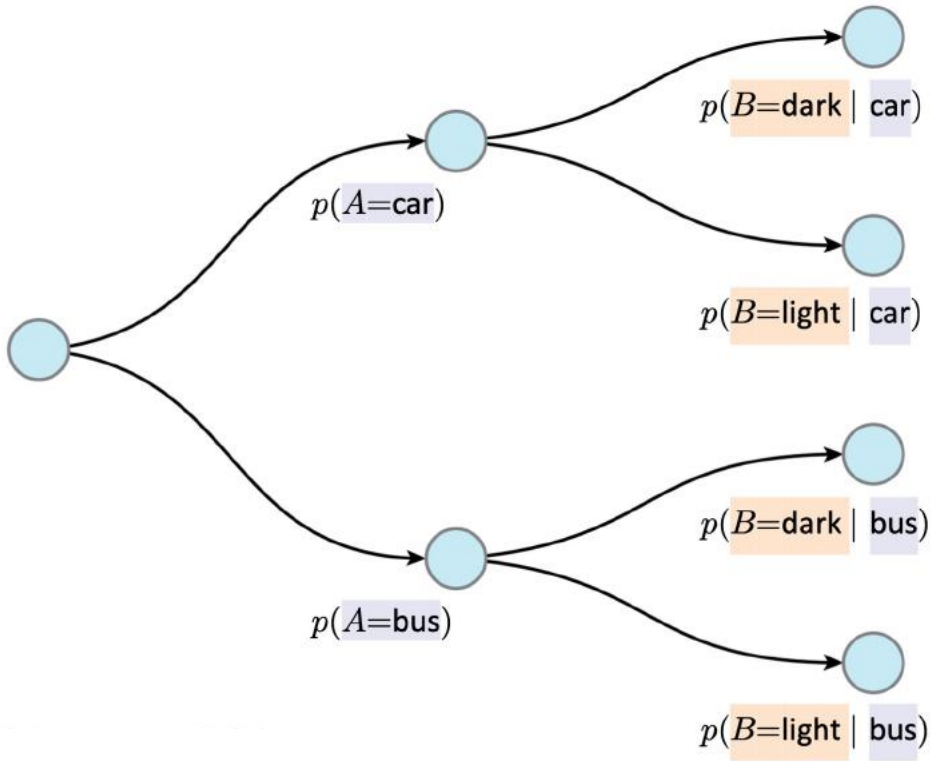


*Assuming binary variables

Conditional Distribution Modeling

- **Chain Rule:**
  Any joint distribution can be written as a product of conditionals

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

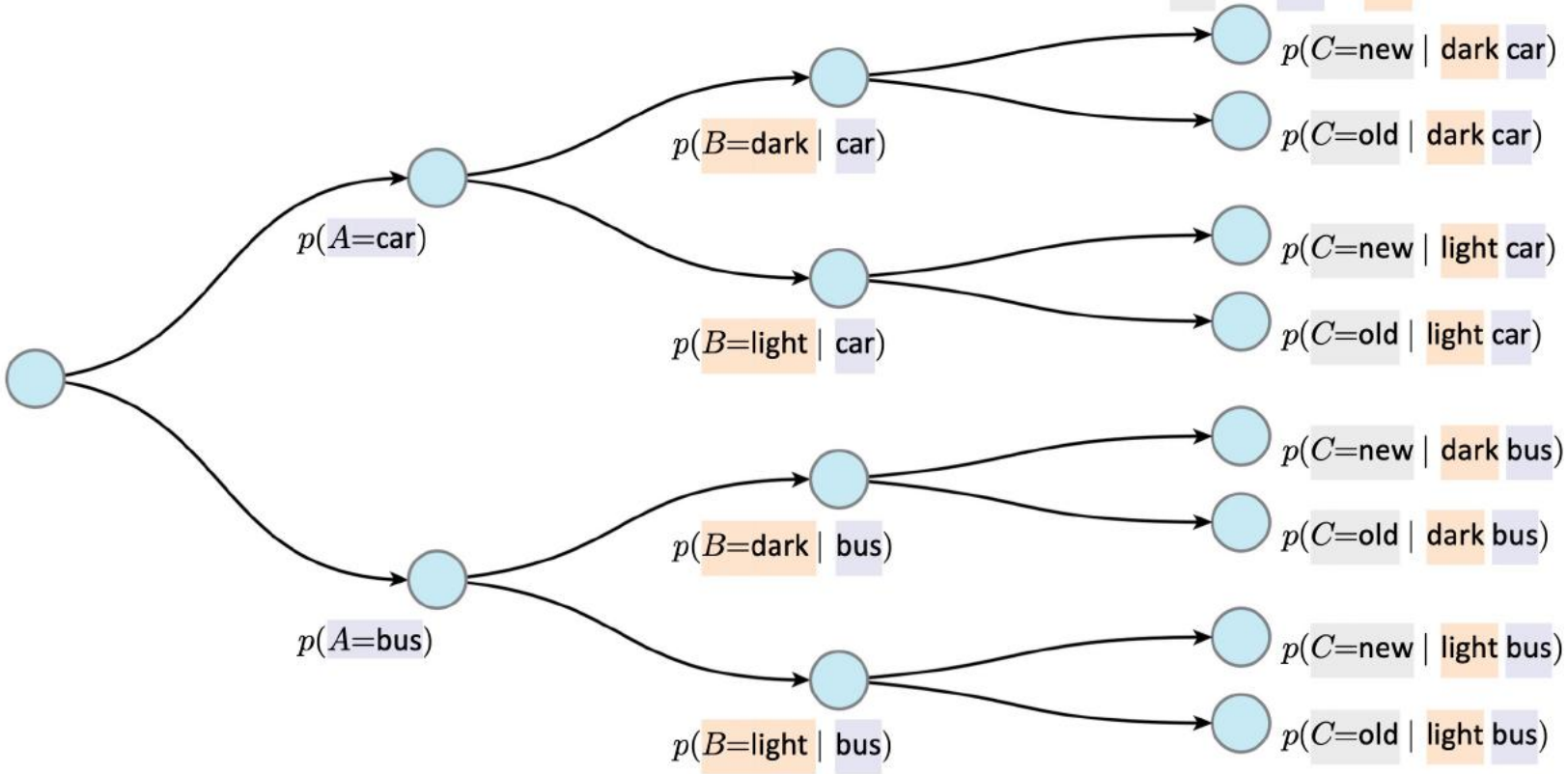


*Assuming binary variables

Conditional Distribution Modeling

- **Chain Rule:**
  Any joint distribution can be written as a product of conditionals

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



*Assuming 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

- Any (multi-variable) joint distribution can be written as a product of conditionals

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$$

- This is called an **autoregressive model**.

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)

Conditional Distribution Modeling

- Any (multi-variable) joint distribution can be written as a product of conditionals

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$$

- This is called an **autoregressive model**.

- Are we done? No!
  - What we need to resolve still: how to efficiently represent and learn each conditional

Autoregressive models

- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid Parents(x_i))$$

Bayes nets and neural nets

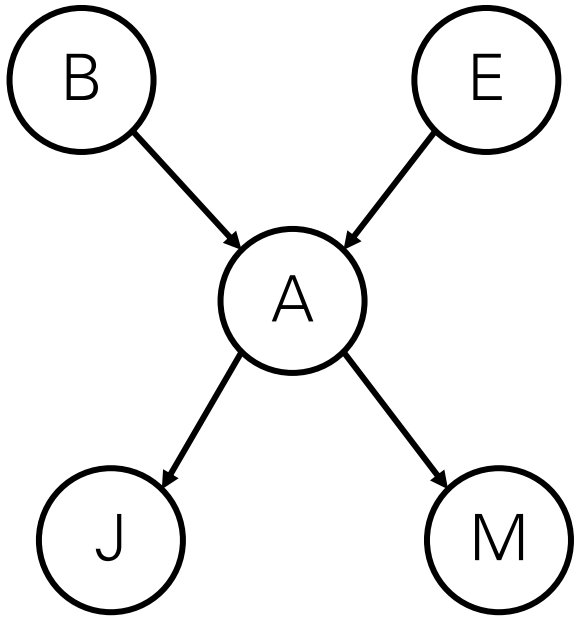
$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid Parents(x_i))$$

$$p_\theta(B, E, A, J, M) = p_\theta(B)p_\theta(E)p_\theta(A \mid B, E)p_\theta(J \mid A)p_\theta(M \mid A)$$

+ If the data has such underlying (causal?) structure, can be a great inductive bias
- But generally, sparsification introduces strong assumptions, limits expressivity
- In standard form has limited parameter sharing between conditionals (though in principle could introduce some sharing)

B	P(B)
+b	0.001
-b	0.999

A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95



E	P(E)
+e	0.002
-e	0.998

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

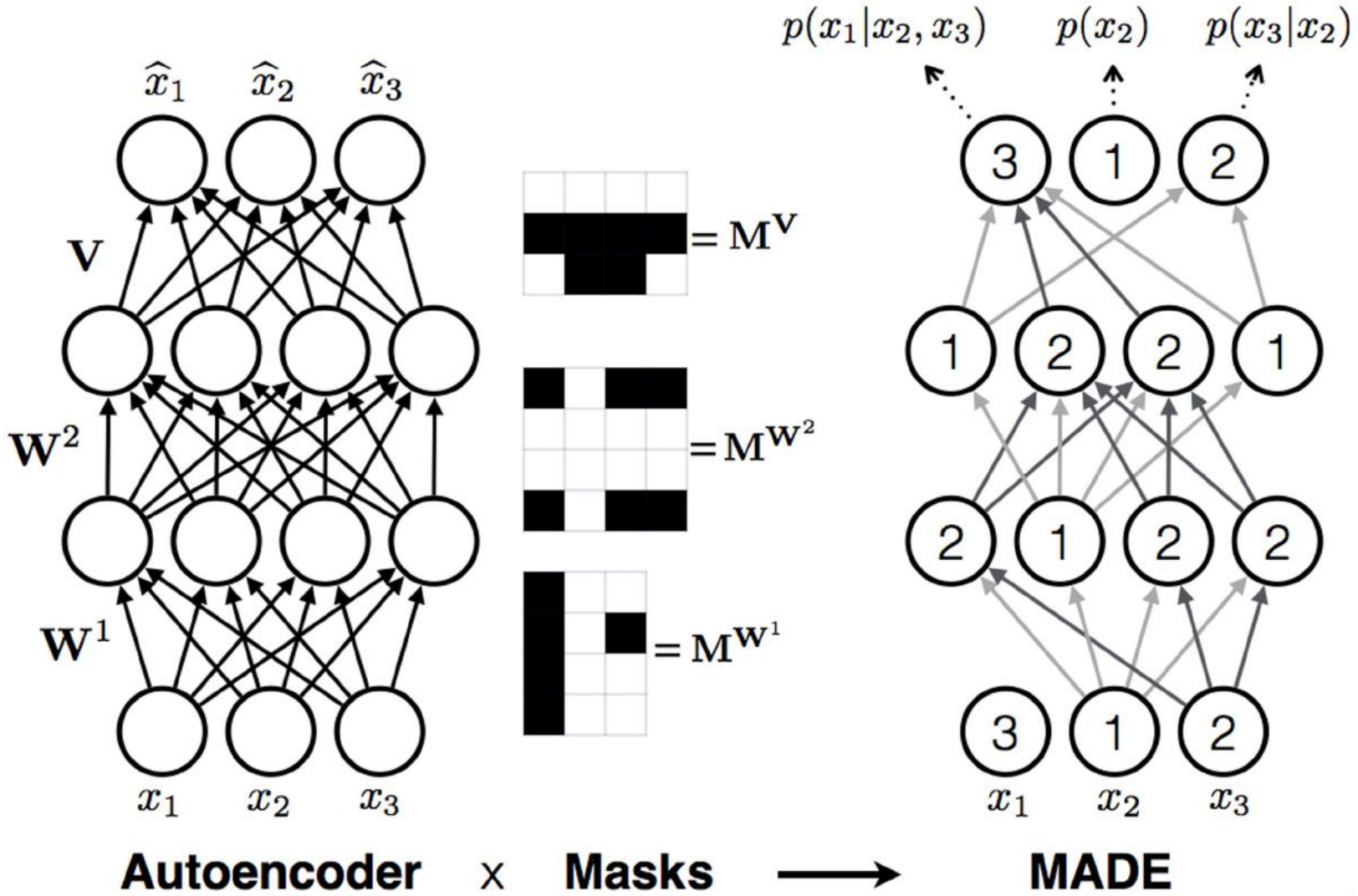
Autoregressive models

- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most cases, leads to poor fits

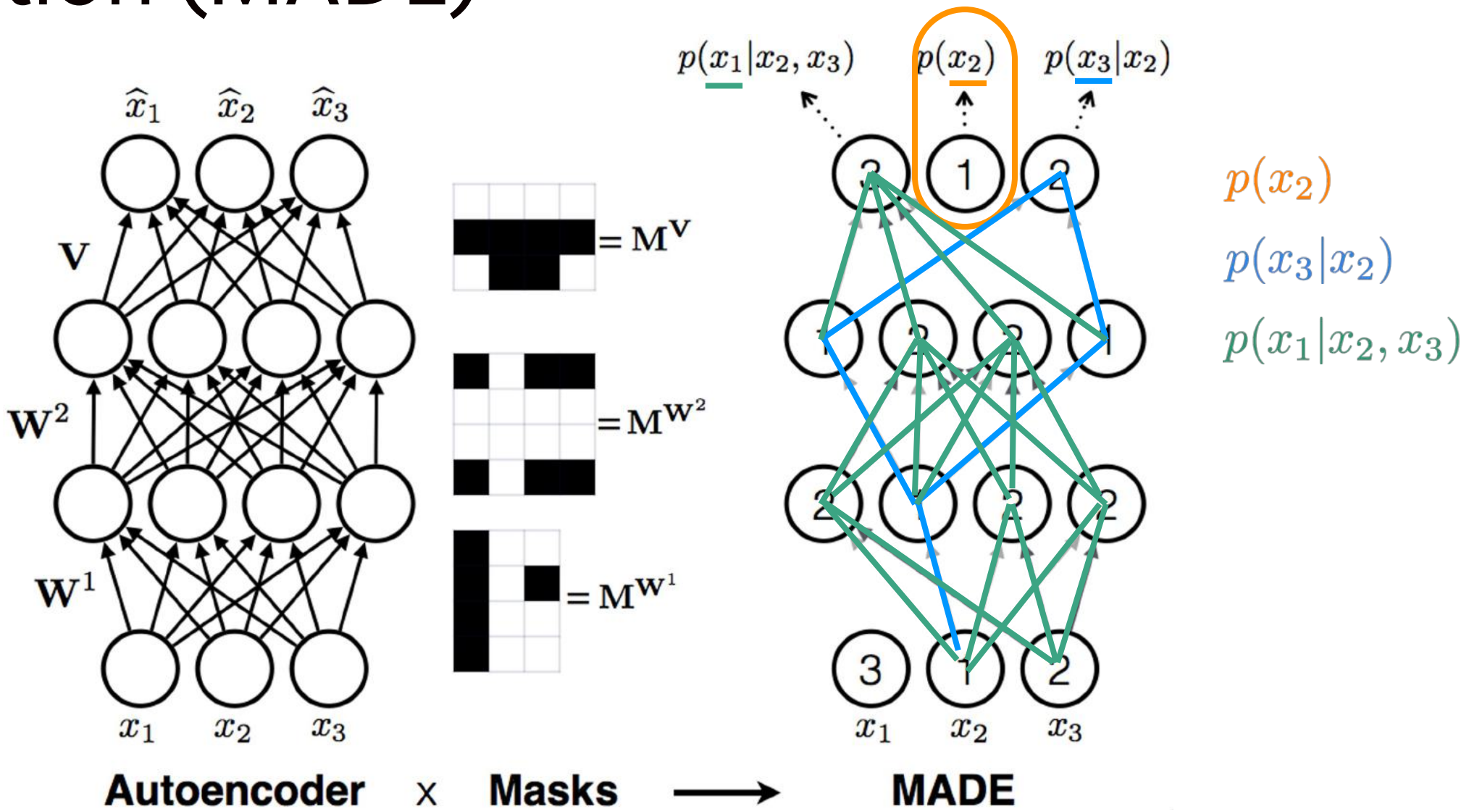
Autoregressive models

- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most cases, leads to poor fits
  - **Solution 2: MADE** – parameterize conditionals with neural net

Masked Autoencoder for Distribution Estimation (MADE)

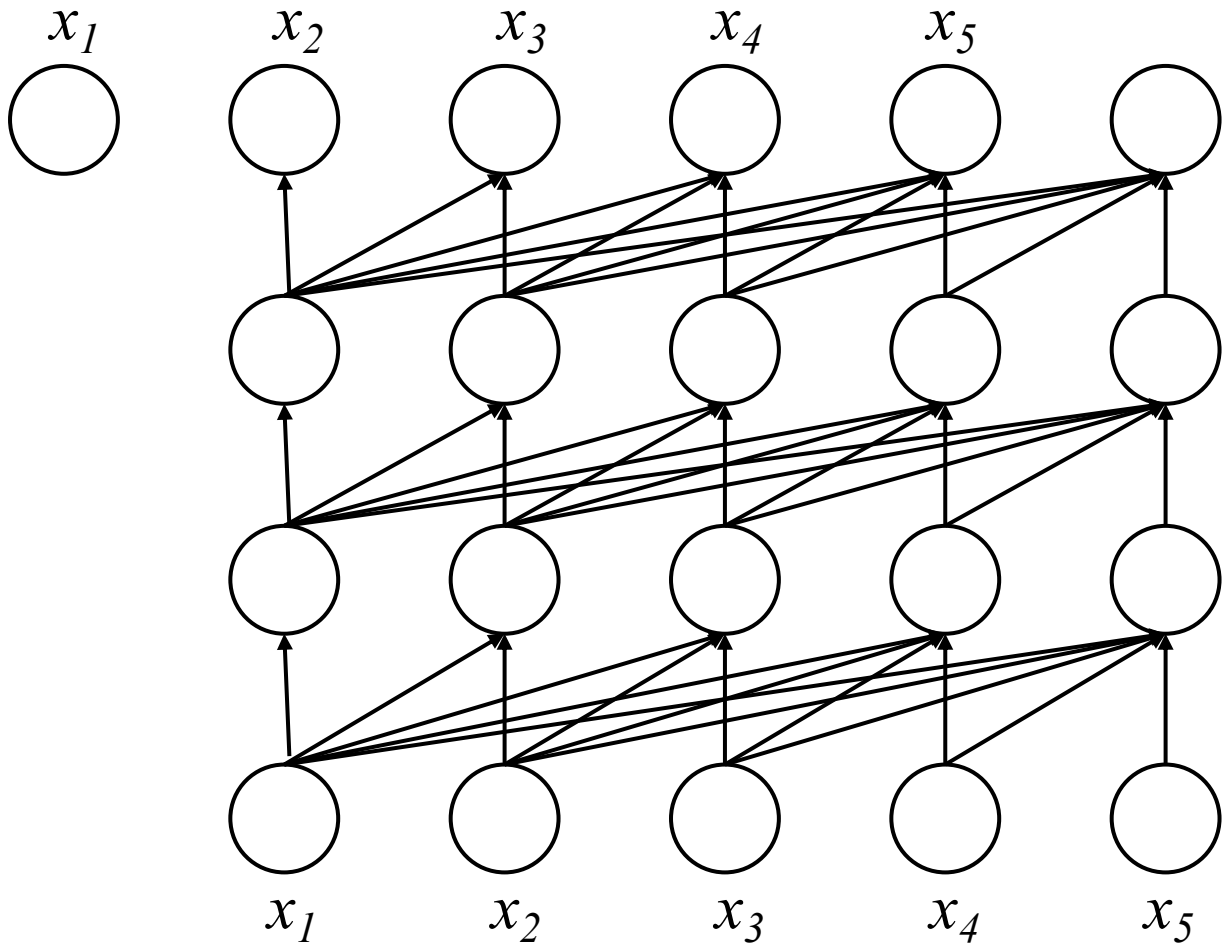


Masked Autoencoder for Distribution Estimation (MADE)



Masked Autoencoder for Distribution Estimation (MADE)

In more modern notation/diagram



Every hidden layer node can be vector valued

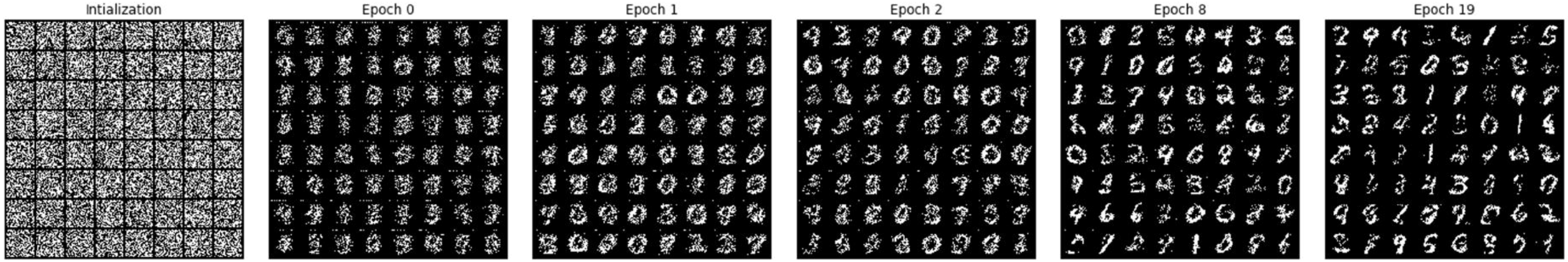
Every edge can be its own MLP connection

MNIST

- Handwritten digits
- 28x28
- 60,000 train
- 10,000 test

- Original: greyscale
- “Binarized MNIST” -- 0/1 (black/white)

MADE on MNIST



MADE results

Table 6. Negative log-likelihood test results of different models on the binarized MNIST dataset.

Model	$-\log p$	
RBM (500 h, 25 CD steps)	≈ 86.34	Intractable
DBM 2hl	≈ 84.62	
DBN 2hl	≈ 84.55	
DARN n_h=500	≈ 84.71	
DARN n_h=500, adaNoise	≈ 84.13	
MoBernoullis K=10	168.95	Tractable
MoBernoullis K=500	137.64	
NADE 1hl (fixed order)	88.33	
EoNADE 1hl (128 orderings)	87.71	
EoNADE 2hl (128 orderings)	85.10	
MADE 1hl (1 mask)	88.40	
MADE 2hl (1 mask)	89.59	
MADE 1hl (32 masks)	88.04	
MADE 2hl (32 masks)	86.64	

MADE results

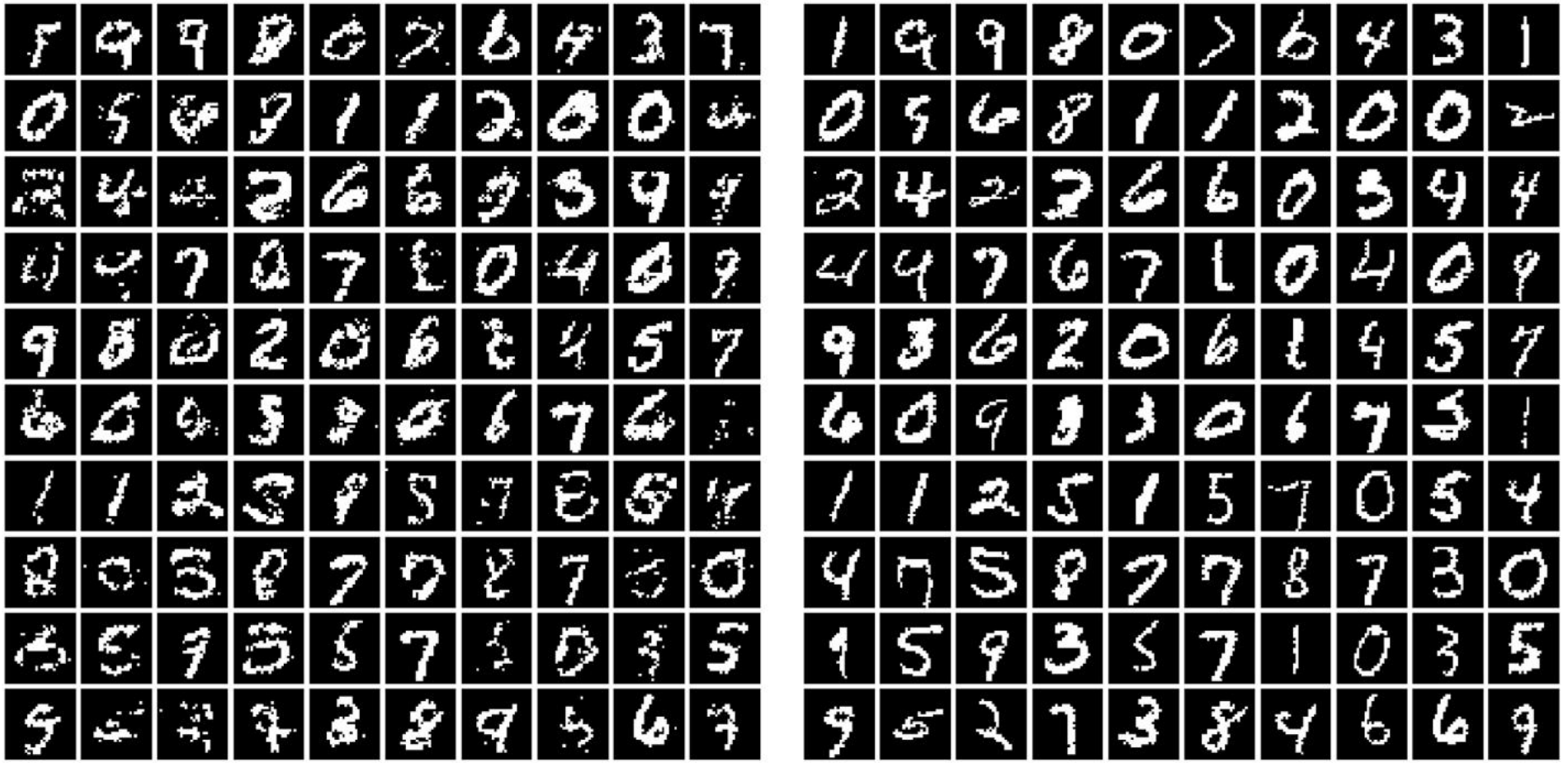
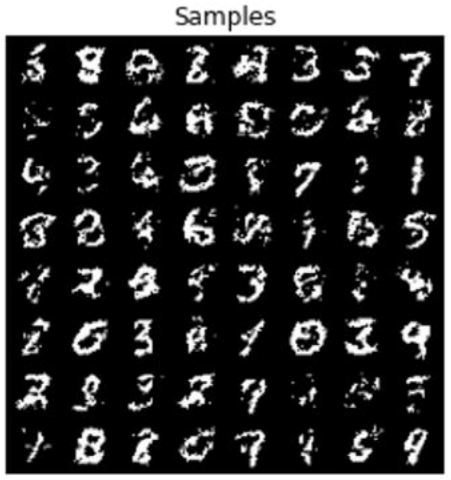


Figure 3. Left: Samples from a 2 hidden layer MADE. Right: Nearest neighbour in binarized MNIST.

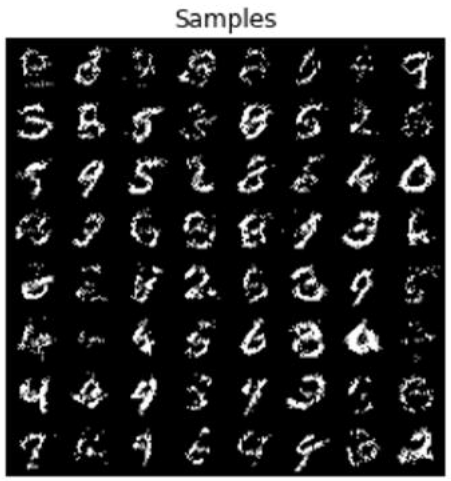
MADE – Different Orderings

- All orderings achieve roughly the same negative log likelihood loss, but samples are different

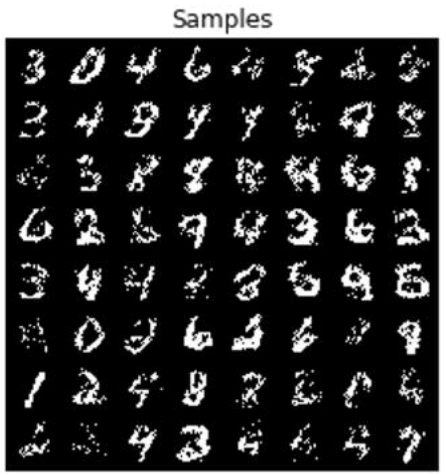
Random Permutation



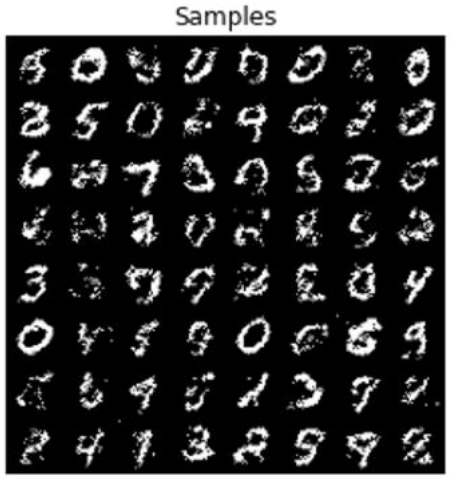
Even then Odd Indices



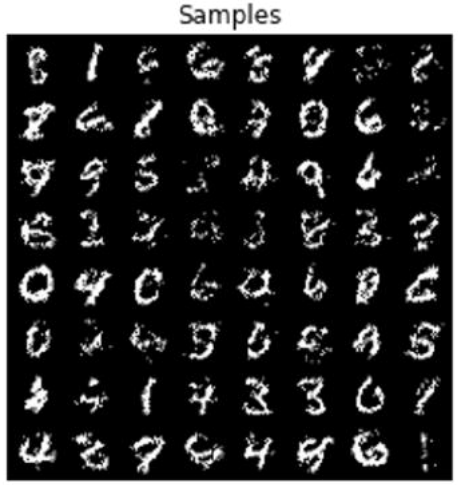
Rows (Raster Scan)



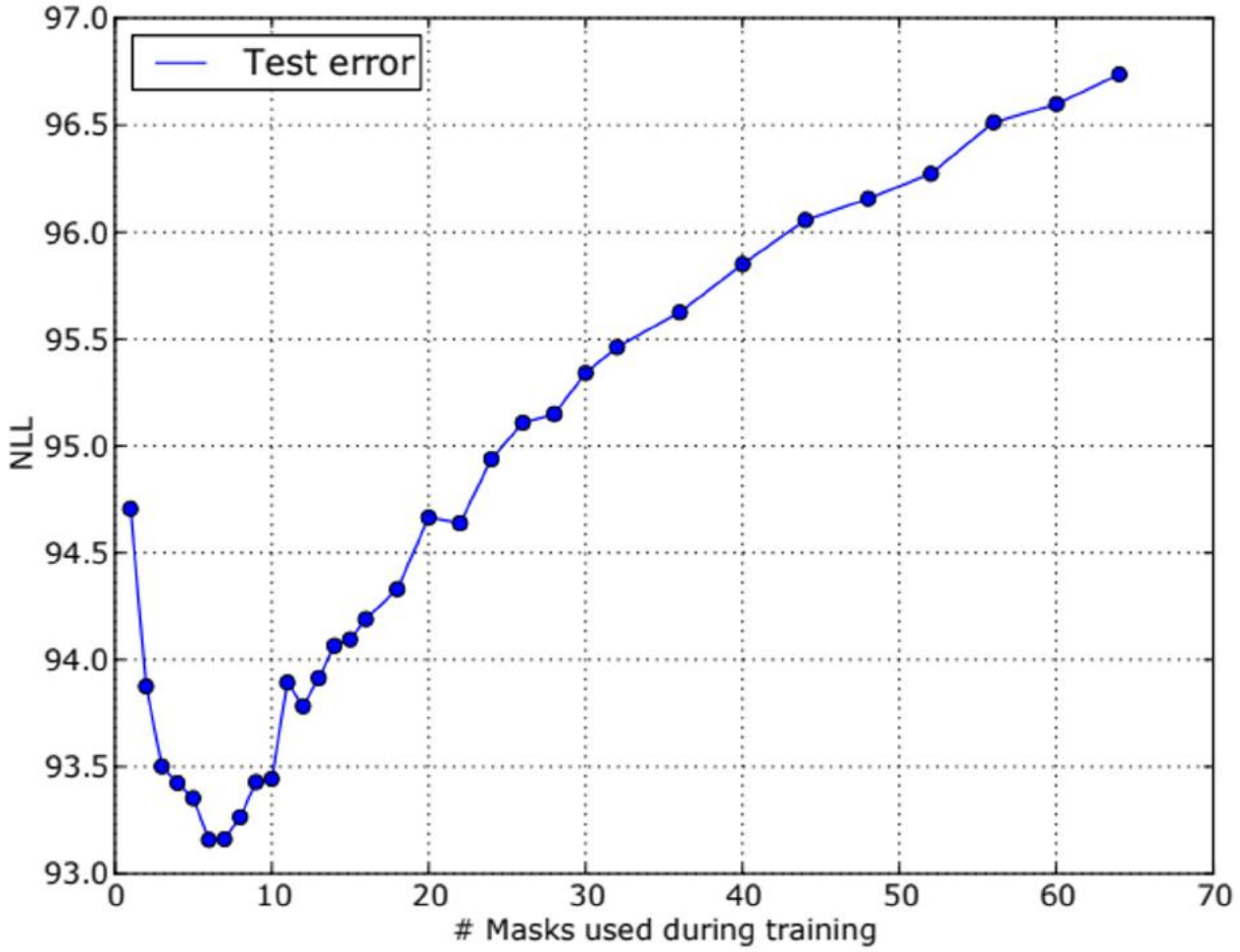
Columns



Top to Middle, Bottom to Middle



MADE: Multiple Orderings



Autoregressive models

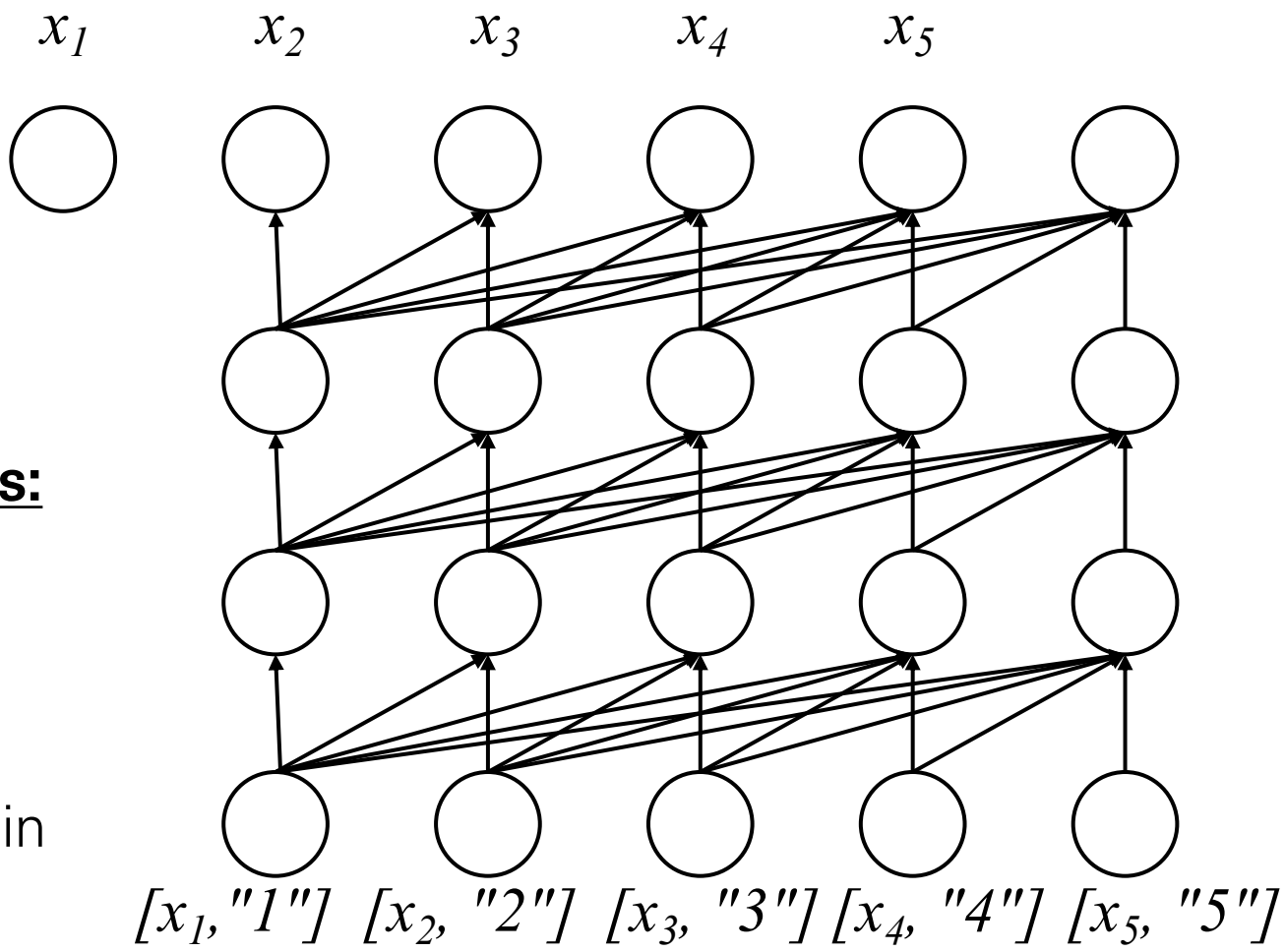
- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most cases, leads to poor fits
  - **Solution 2: MADE** – parameterize conditionals with neural net
    - Expressive, but: not enough parameter sharing for efficient learning

Autoregressive models

- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most cases, leads to poor fits
  - **Solution 2: MADE** – parameterize conditionals with neural net
    - Expressive, but: not enough parameter sharing for efficient learning
  - **Solution 3: Causal Masked Neural Models**
    - Parameterize conditionals with neural net (aka MADE)
      - parameter sharing across conditionals
      - add coordinate coding to still be able to individualize conditionals

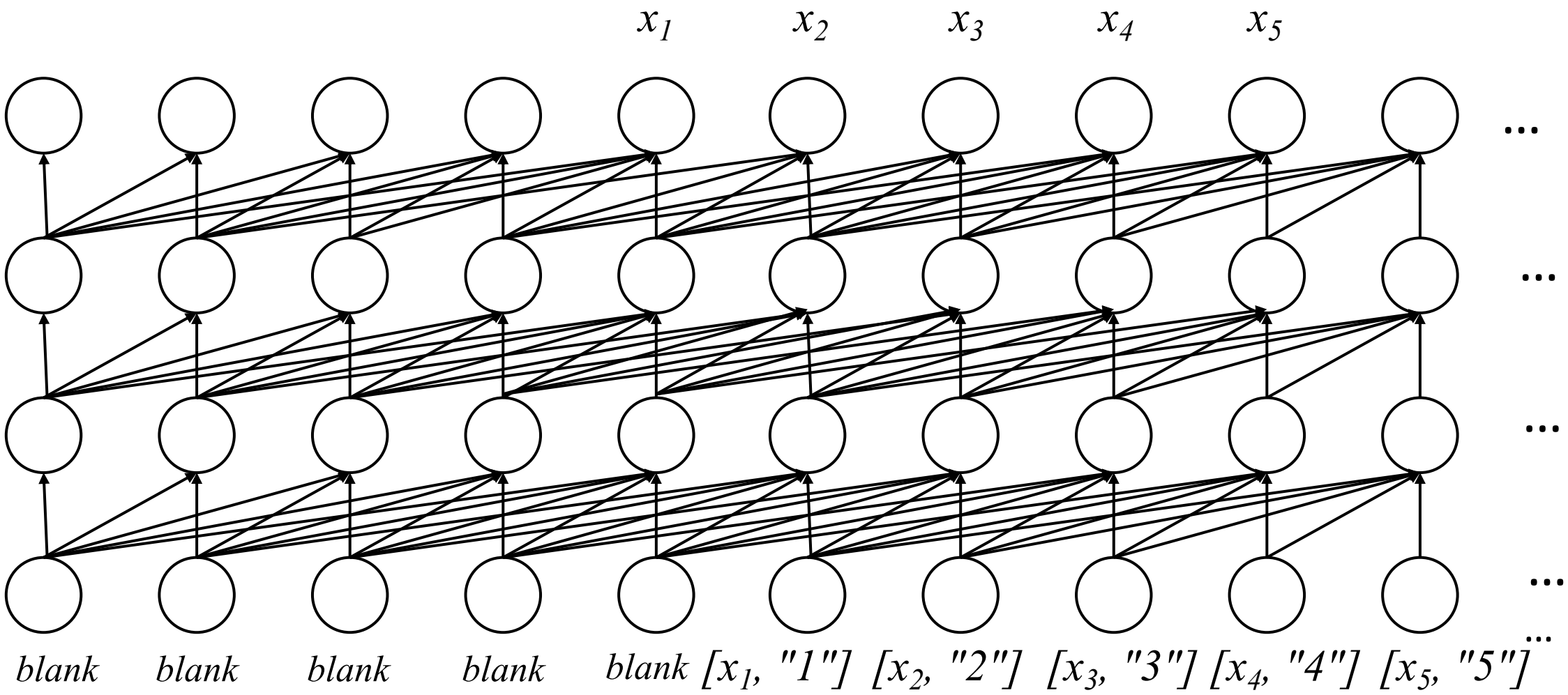
Causal Masked Neural Models

MADE:
- Every hidden layer node can be vector valued
- Every edge can be its own MLP connection

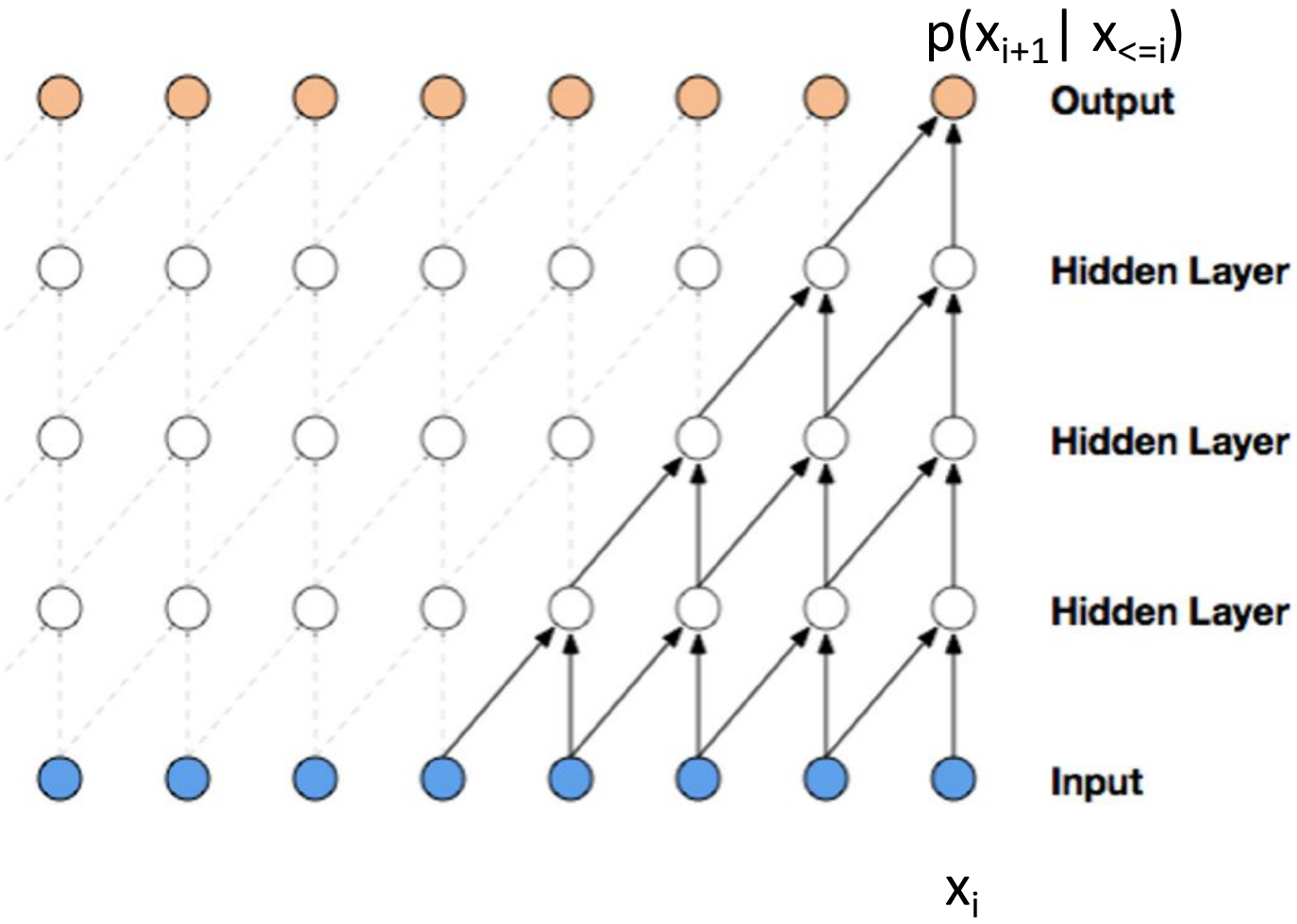


Modern Causal Masked Neural Models:
- Every hidden layer node can be vector valued
- Same parameters when shifting from column i to i+1
- To retain ability to know where we are in sequence: add coordinate coding
- Uniformize fully with padding

Causal Masked Neural Models



Masked Temporal (1D) Convolution

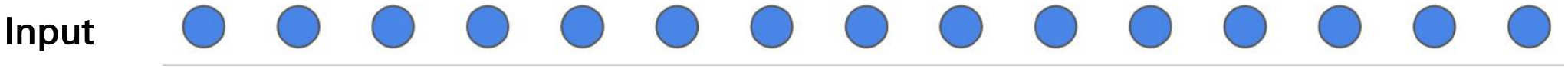


- Easy to implement, masking part of the conv kernel
- Constant parameter count for variable-length distribution!
- Efficient to compute, convolution has hyper-optimized implementations on all hardware

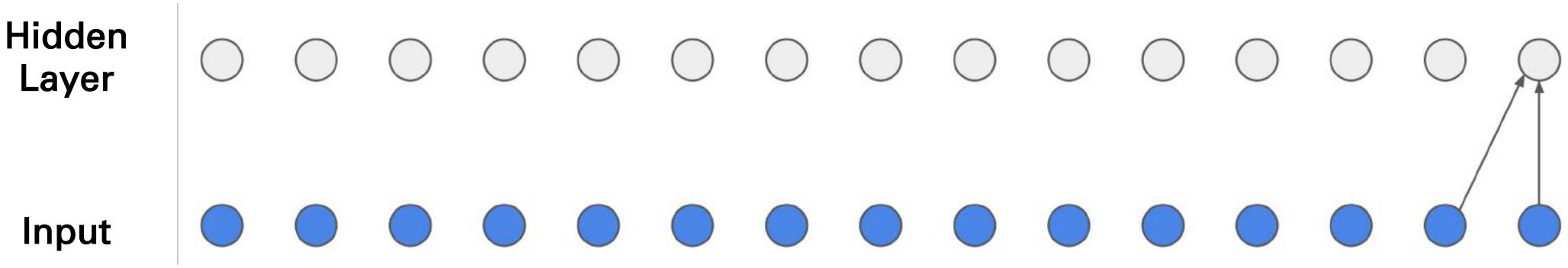
However

- Limited receptive field, linear in number of layers

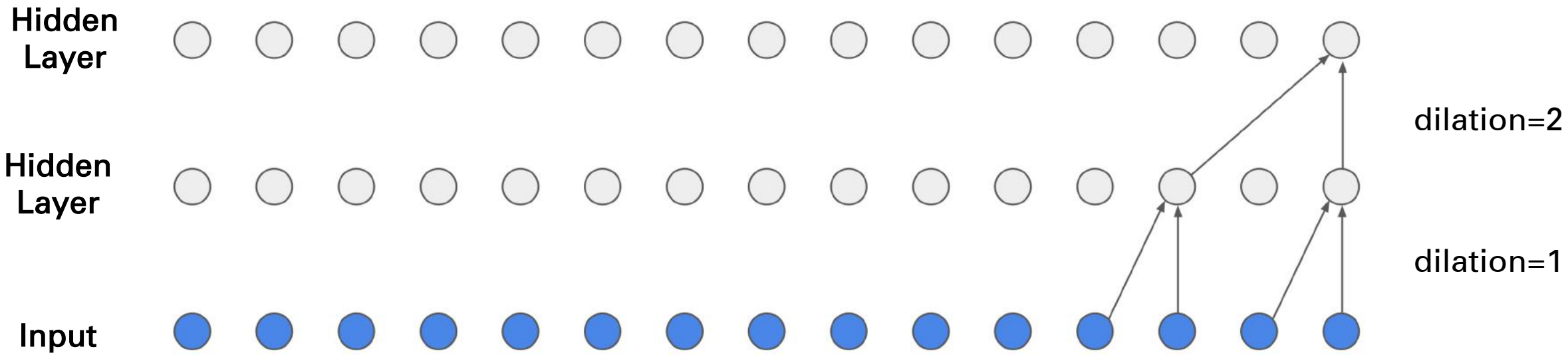
WaveNet – Causal Dilated Convolution



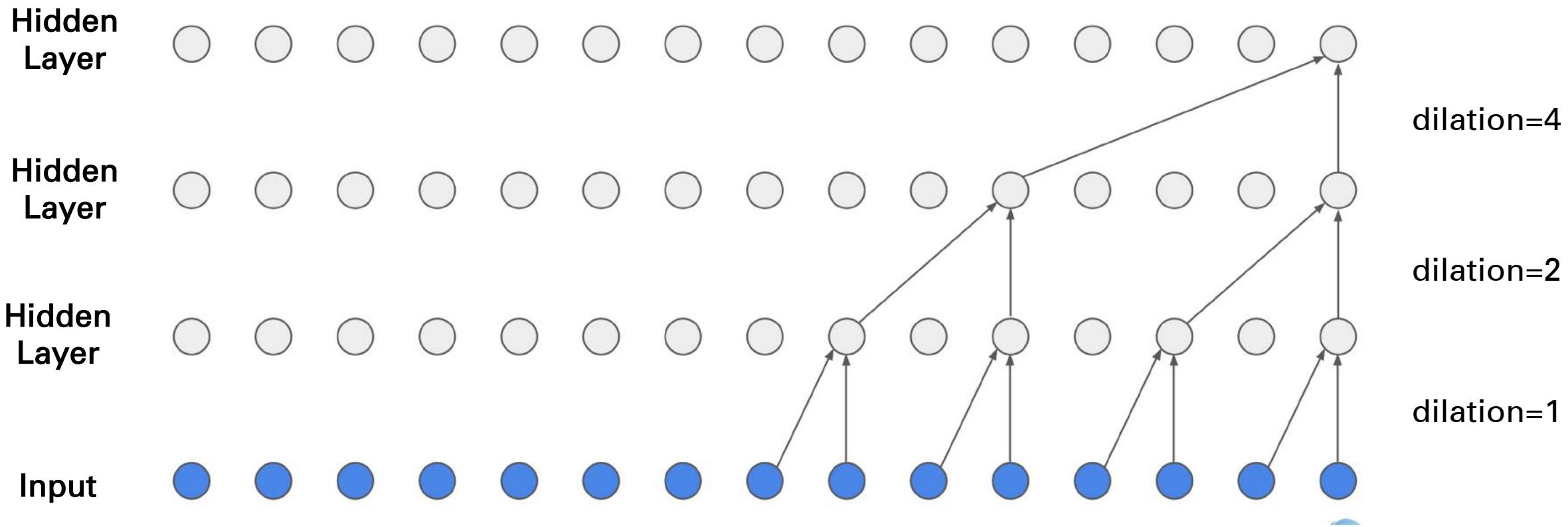
WaveNet – Causal Dilated Convolution



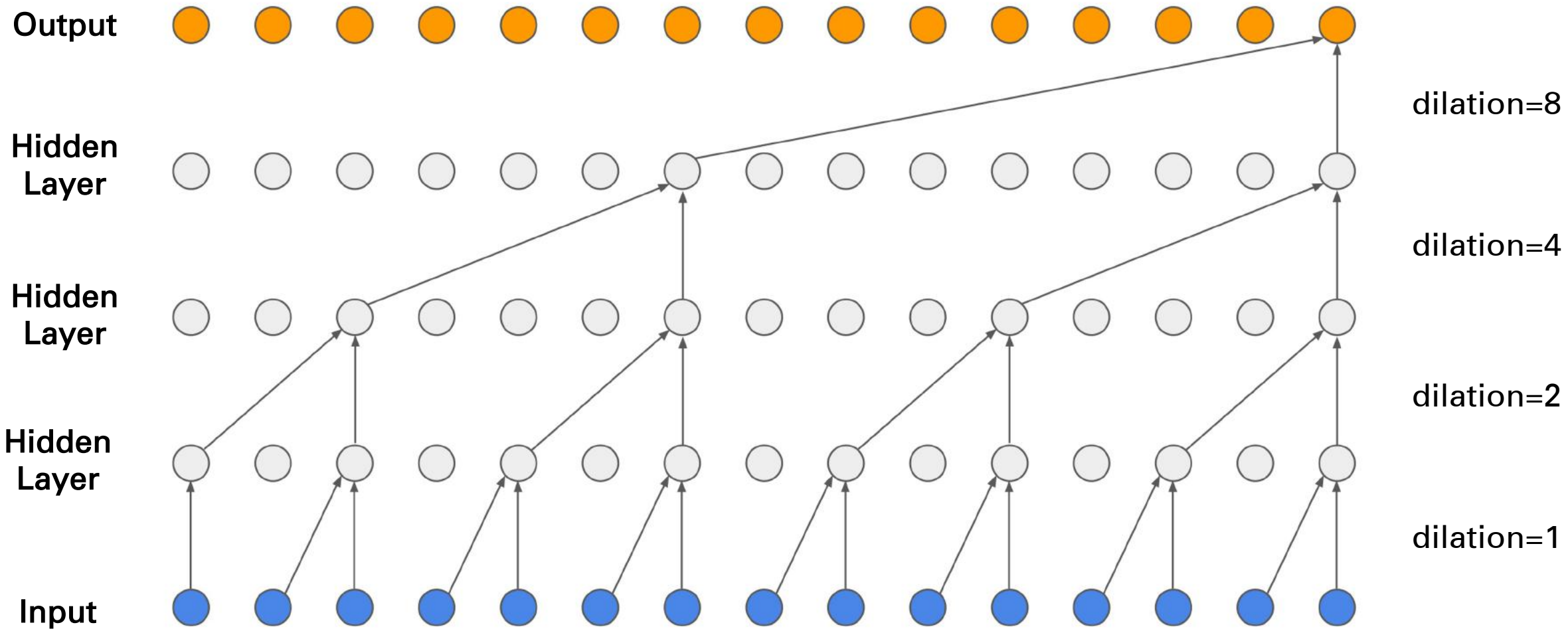
WaveNet – Causal Dilated Convolution



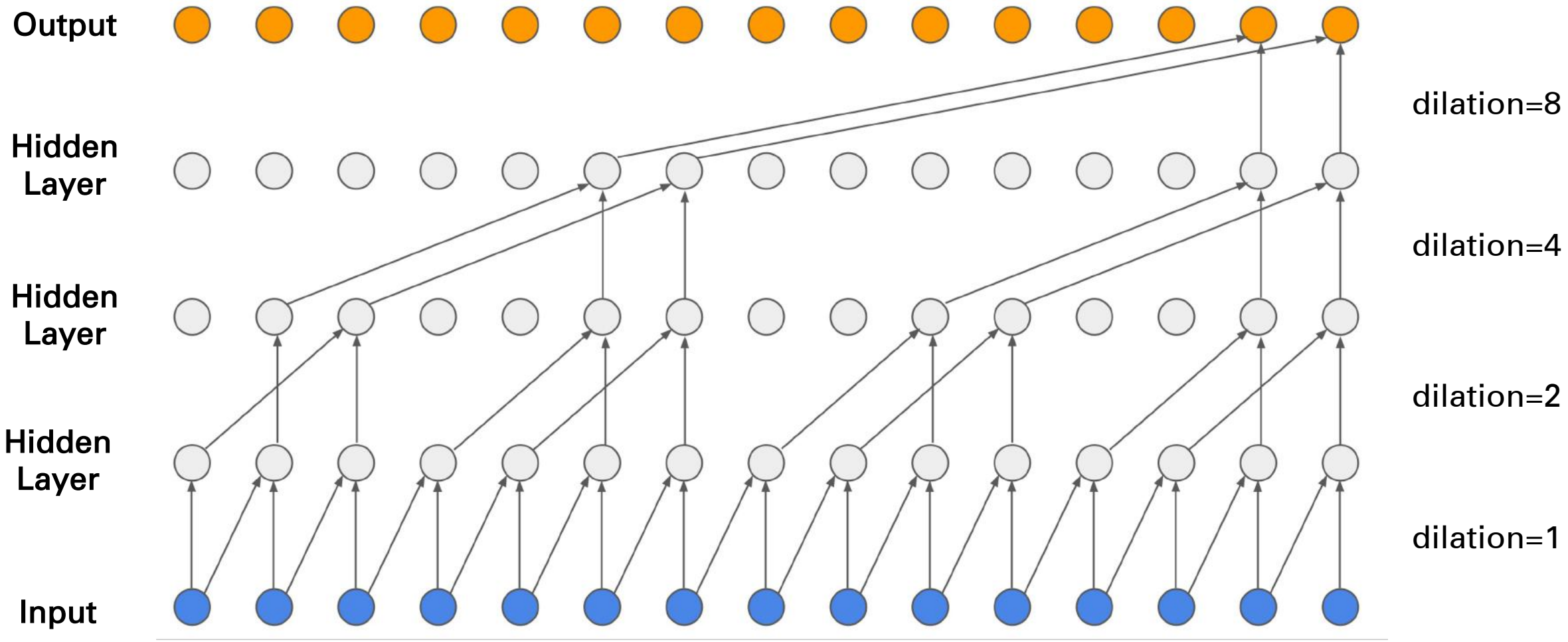
WaveNet – Causal Dilated Convolution



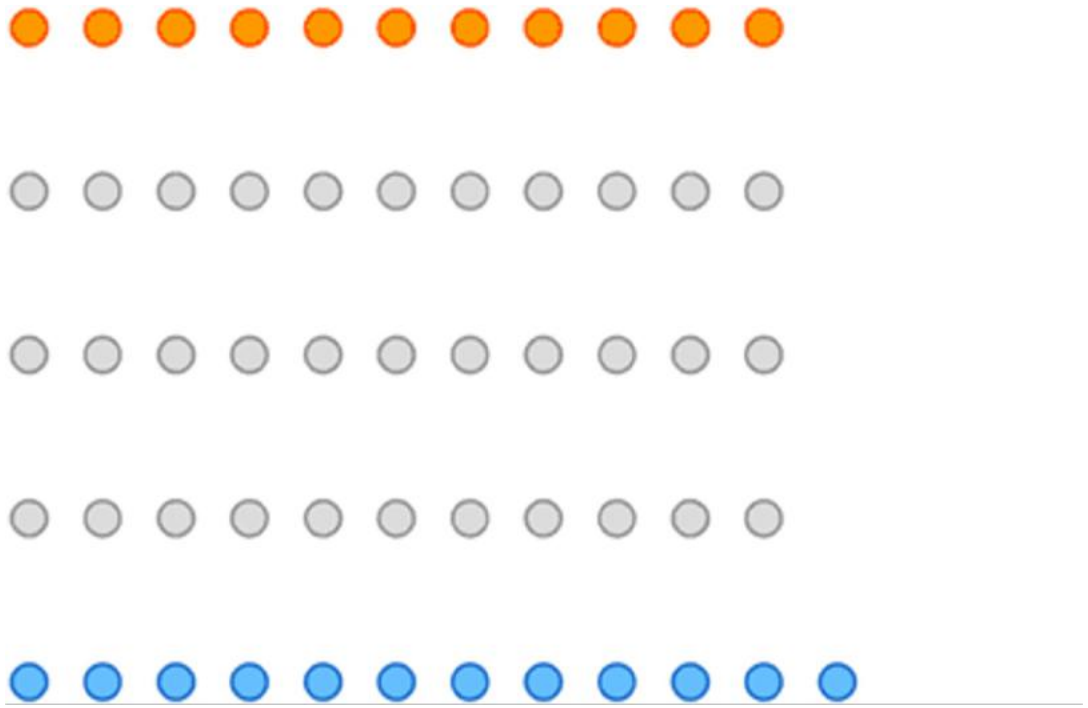
WaveNet – Causal Dilated Convolution



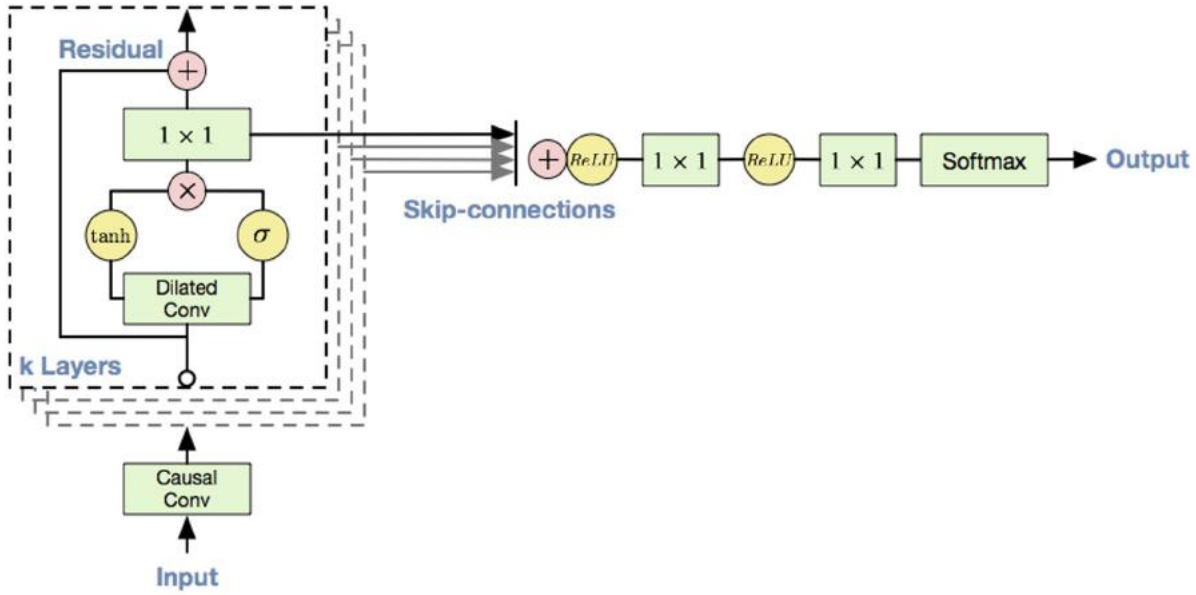
WaveNet – Causal Dilated Convolution



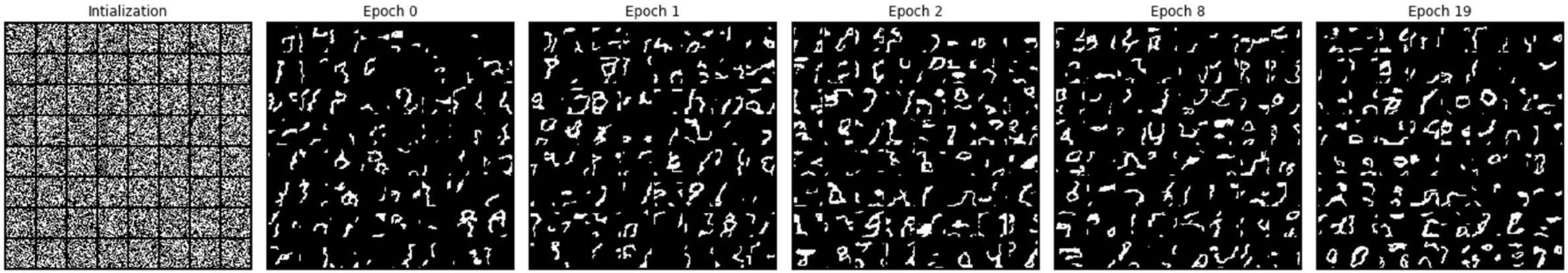
WaveNet



- Improved receptive field: dilated convolution, with exponential dilation
- Better expressivity: Gated Residual blocks, Skip connections

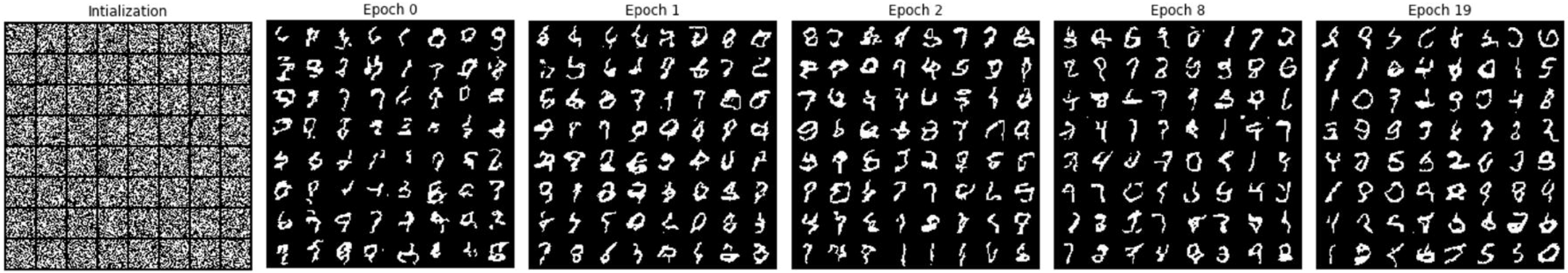


WaveNet on MNIST



WaveNet with Pixel Location Appended on MNIST

- Append (x,y) coordinates of pixel in the image as input to WaveNet



Autoregressive models

- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most cases, leads to poor fits
  - **Solution 2: MADE** – parameterize conditionals with neural net
    - Expressive, but: not enough parameter sharing for efficient learning
  - **Solution 3: Causal Masked Neural Models**
    - Parameterize conditionals with neural net (aka MADE)
      - parameter sharing across conditionals
      - add coordinate coding to still be able to individualize conditionals
    - Expressive and efficient!
      Any buts? possible concern is finite context window, but in practice quite large + retrieval can help

Autoregressive models

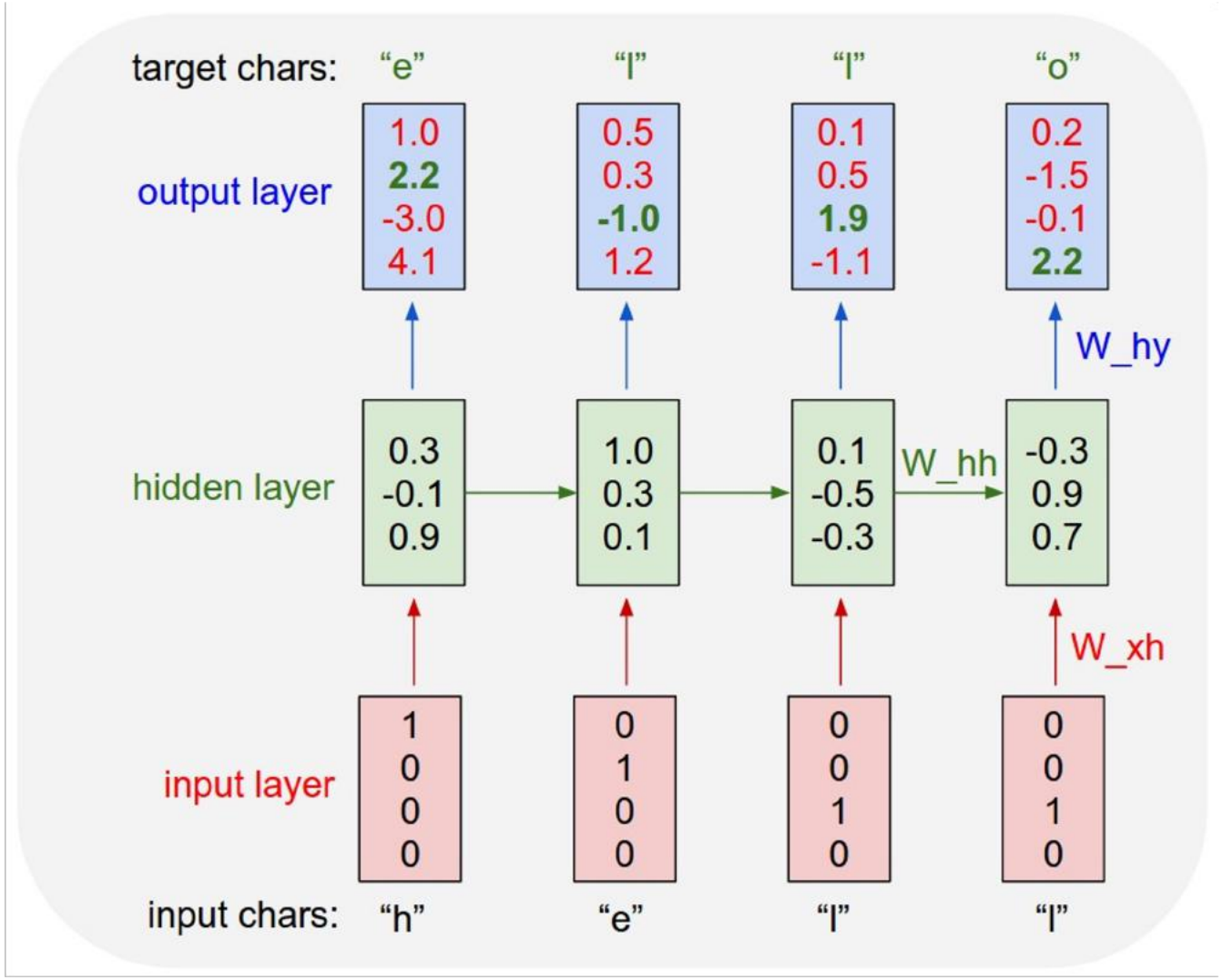
- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most cases, leads to poor fits
  - **Solution 2: MADE** – parameterize conditionals with neural net
    - Expressive, but: not enough parameter sharing for efficient learning
  - **Solution 3: Causal Masked Neural Models**
    - Parameterize conditionals with neural net (aka MADE)
      - parameter sharing across conditionals
      - add coordinate coding to still be able to individualize conditionals
    - Expressive and efficient!
      Any buts? possible concern is finite context window, but in practice quite large + retrieval can help
  - **Solution 4: Recurrent Neural Net** – parameter sharing + "infinite look-back"

RNN autoregressive models - char-rnn

$$\log p(\mathbf{x}) = \sum_{i=1}^{d} \log p(x_i \mid \mathbf{x}_{1:i-1})$$

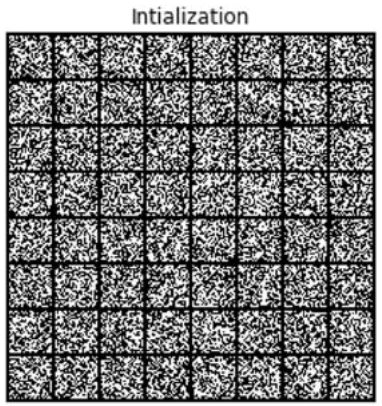
Sequence of characters

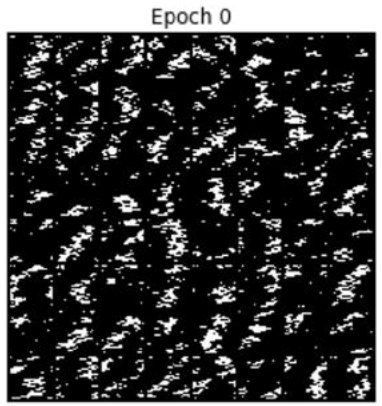
Character at i^{th} position

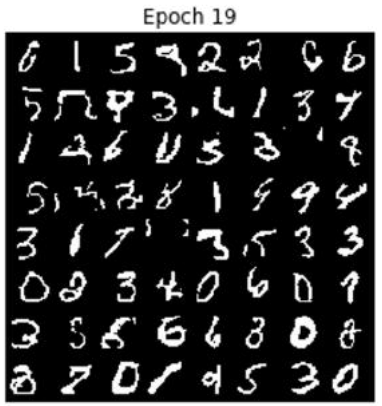


[Diagram: Karpathy, 2015; earlier papers: A Neural Probabilistic Model – Bengio et al, 2003; Sequence to Sequence Learning with NNs – Sutskever, Vinyals, Le, 2014]

RNN on MNIST







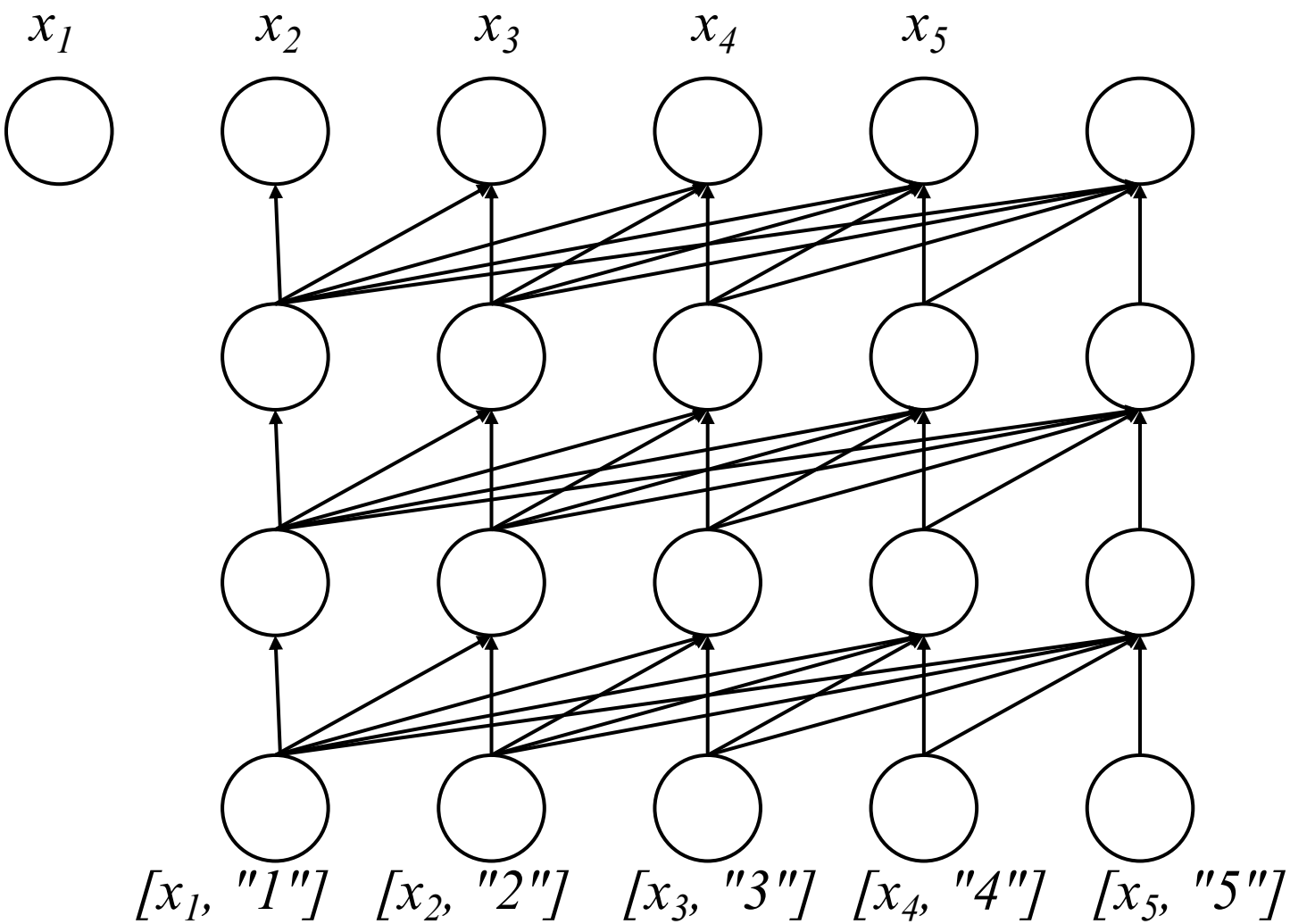
RNN with Pixel Location Appended on MNIST

- Append (x,y) coordinates of pixel in the image as input to RNN

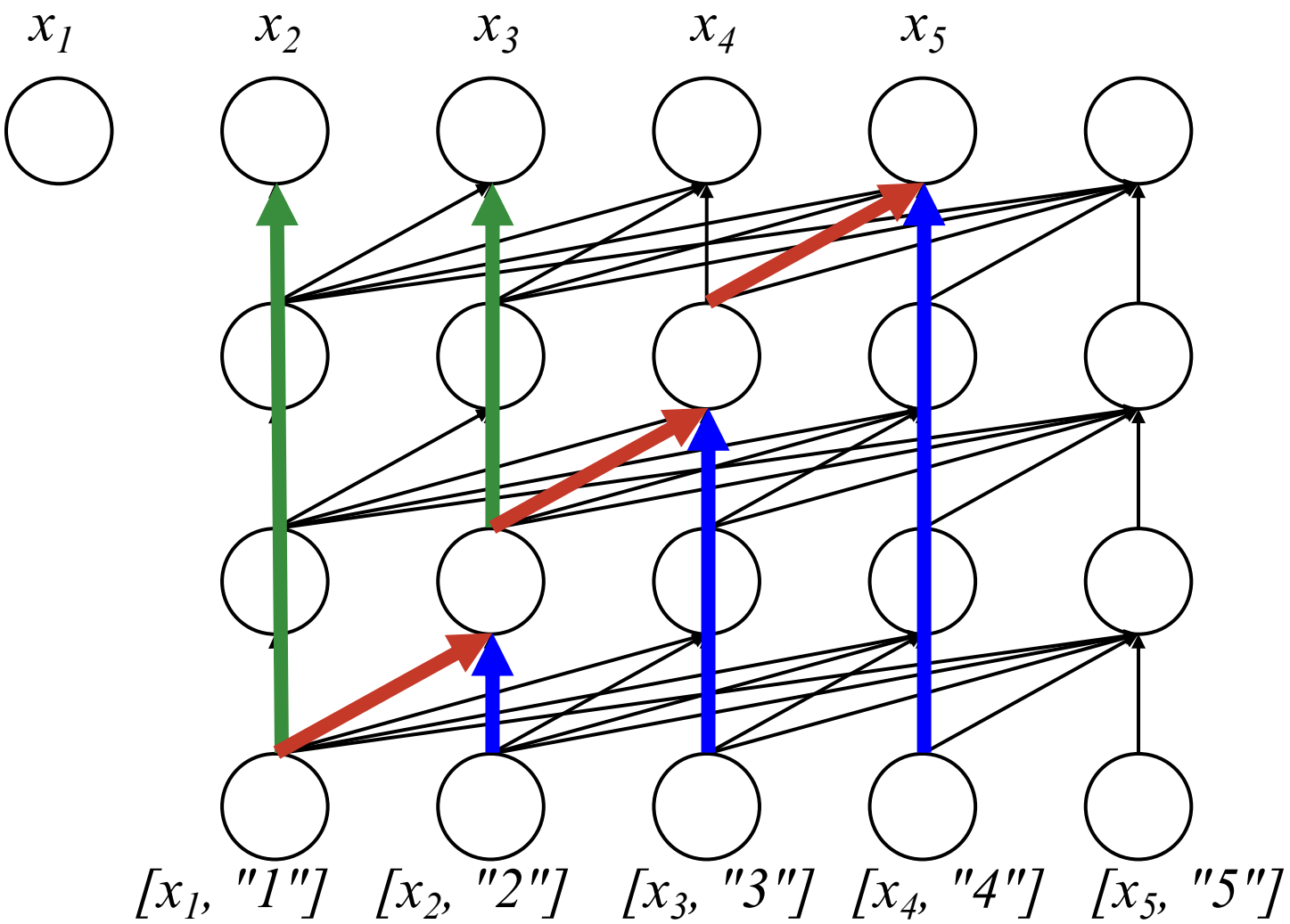
Autoregressive models

- Recall, AutoRegressive Model $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most cases, leads to poor fits
  - **Solution 2: MADE** – parameterize conditionals with neural net
    - Expressive, but: not enough parameter sharing for efficient learning
  - **Solution 3: Causal Masked Neural Models**
    - Parameterize conditionals with neural net (aka MADE)
      - parameter sharing across conditionals
      - add coordinate coding to still be able to individualize conditionals
    - Expressive and efficient!
      Any buts? possible concern is finite context window, but in practice quite large + retrieval can help
  - **Solution 4: Recurrent Neural Net** – parameter sharing + "infinite look-back"
    - Expressive, but: in practice doesn't tend to work as well as next proposal
      - not as amenable to parallelization
      - backprop through time can have exploding / vanishing gradients (there are tricks)
      - hard to truly have signal propagate from long history (i.e. benefit less than "advertised")
      - expressive but maybe not sufficiently expressive / not the right inductive biases

Causal Masked Neural Models



Causal Masked Neural Models



RNN ~ very deep causal masked model with very sparse connectivity:

- Encoder (skip)
- Main diagonal
- Decoder (skip)
- Parameter sharing for each of these

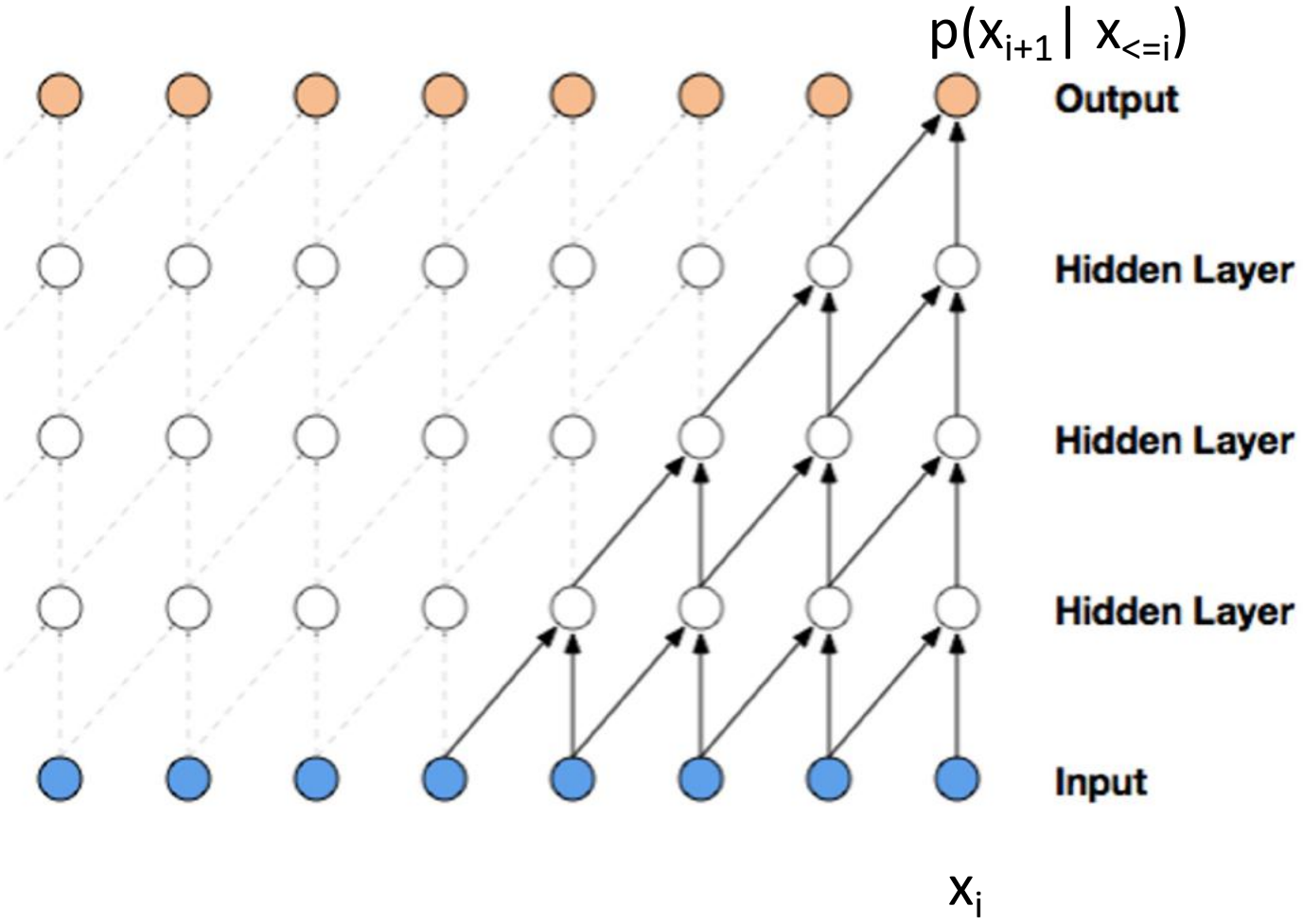
Lecture overview

- motivation
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
  - convolutional
  - attention
  - tokenization
  - caching
- other things to be aware of

Lecture overview

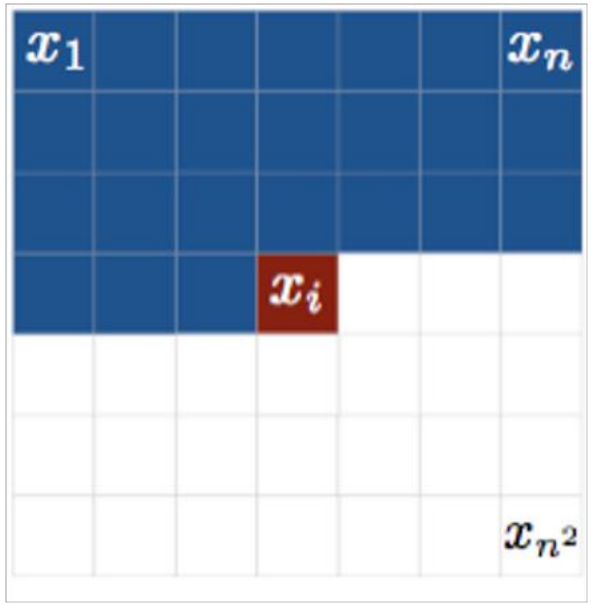
- motivation
- 1-dimensional distributions
- high-dimensional distributions
- **deeper dive into causal masked neural models**
  - **convolutional**
  - attention
  - tokenization
  - caching
- other things to be aware of

Earlier: Masked Temporal (1D) Convolution



Masked Spatial (2D) Convolution - PixelCNN

- Images can be flatten into 1D vectors, but they are fundamentally 2D
- We can use a masked variant of ConvNet to exploit this knowledge
- First, we impose an autoregressive ordering on 2D images:

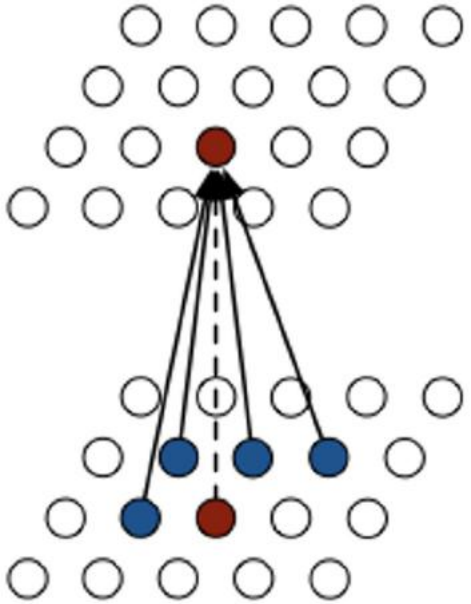


This is called raster scan ordering. (Different orderings are possible, more on this later)

PixelCNN

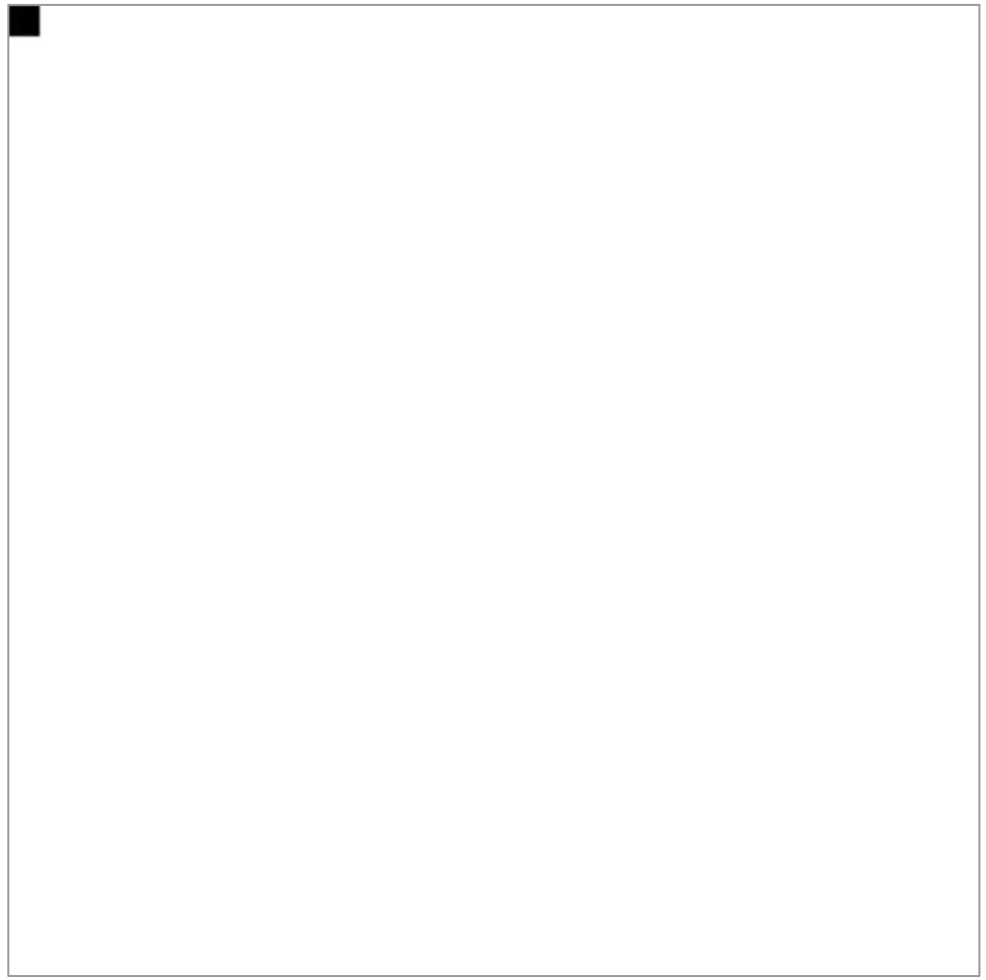
- Design question: how to design a masking method to obey that ordering?
- One possibility: PixelCNN (2016)

1	1	1
1	0	0
0	0	0



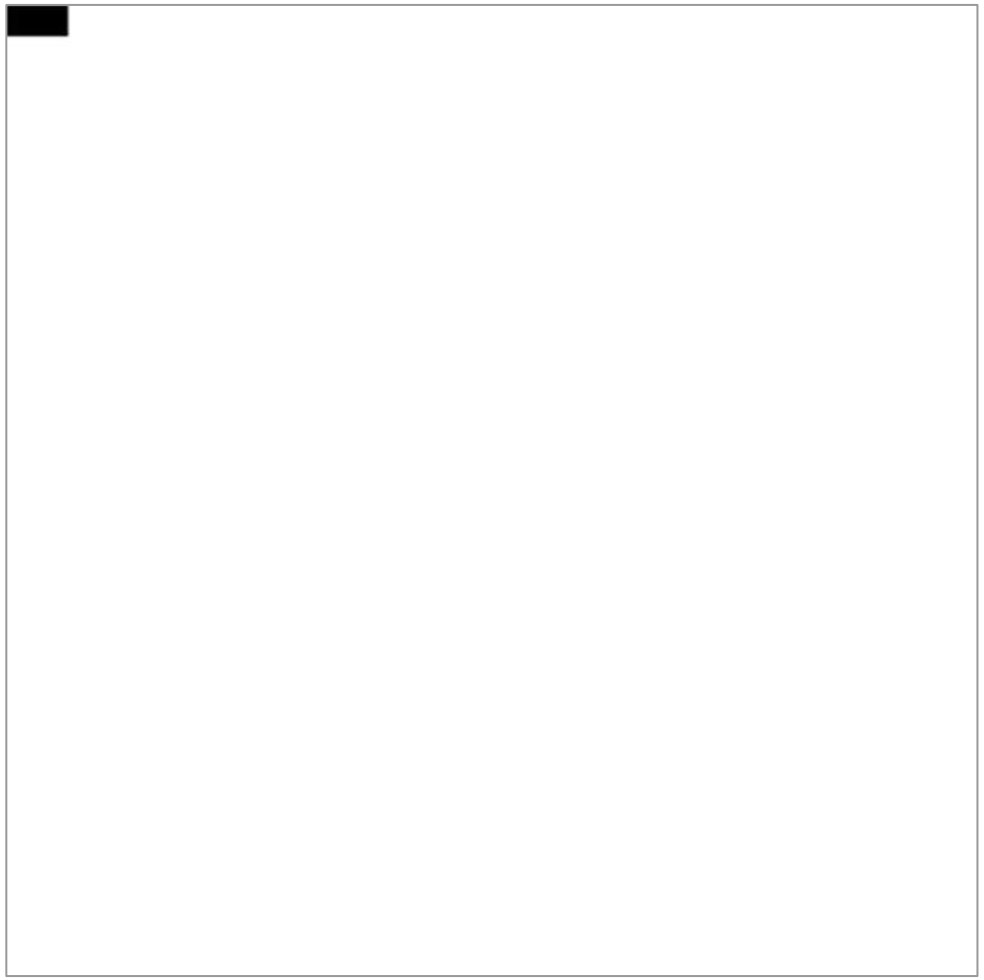
PixelCNN

Softmax Sampling



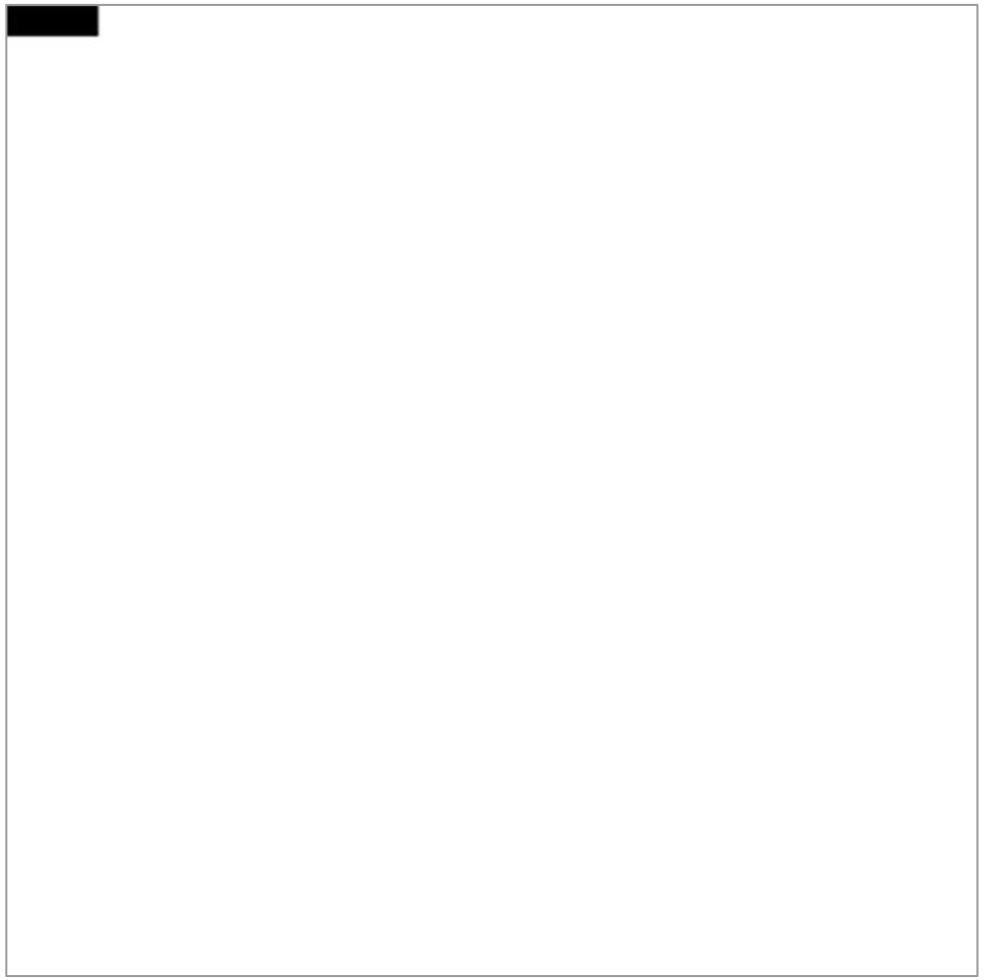
Slide by Aaron van den Oord

Softmax Sampling



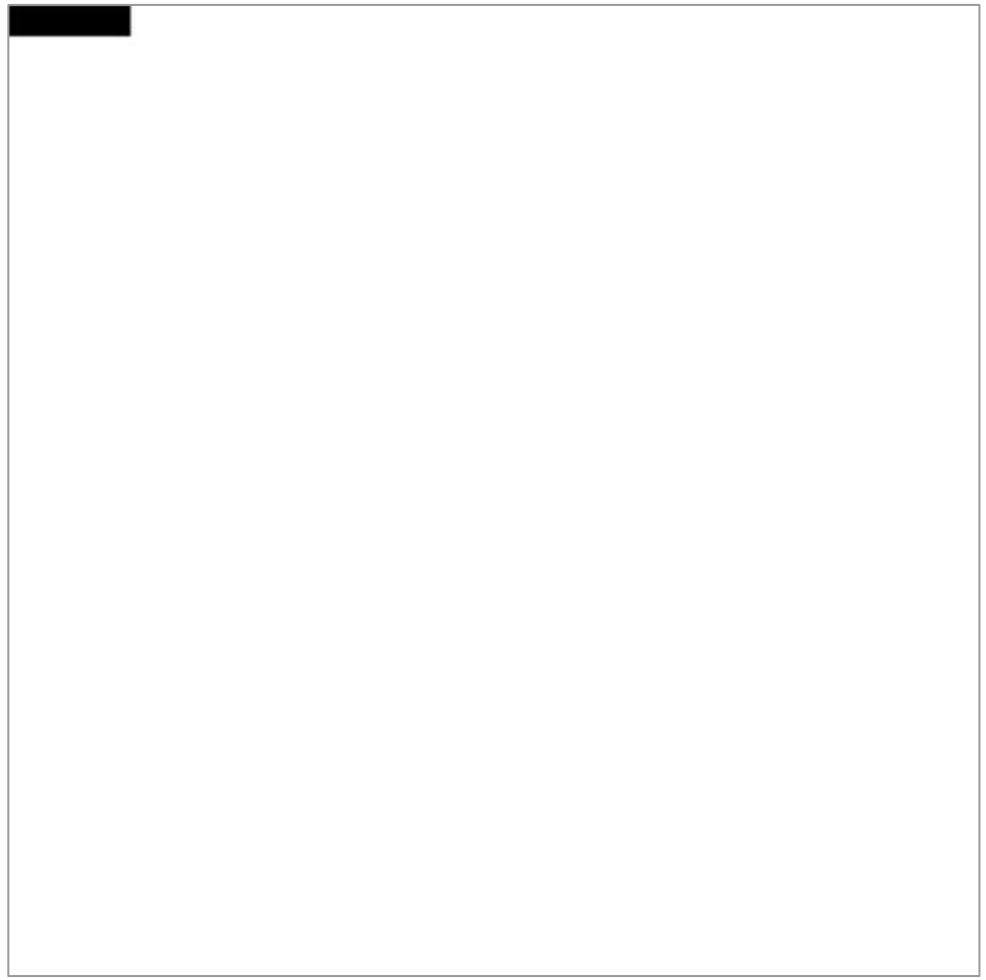
Slide by Aaron van den Oord

Softmax Sampling



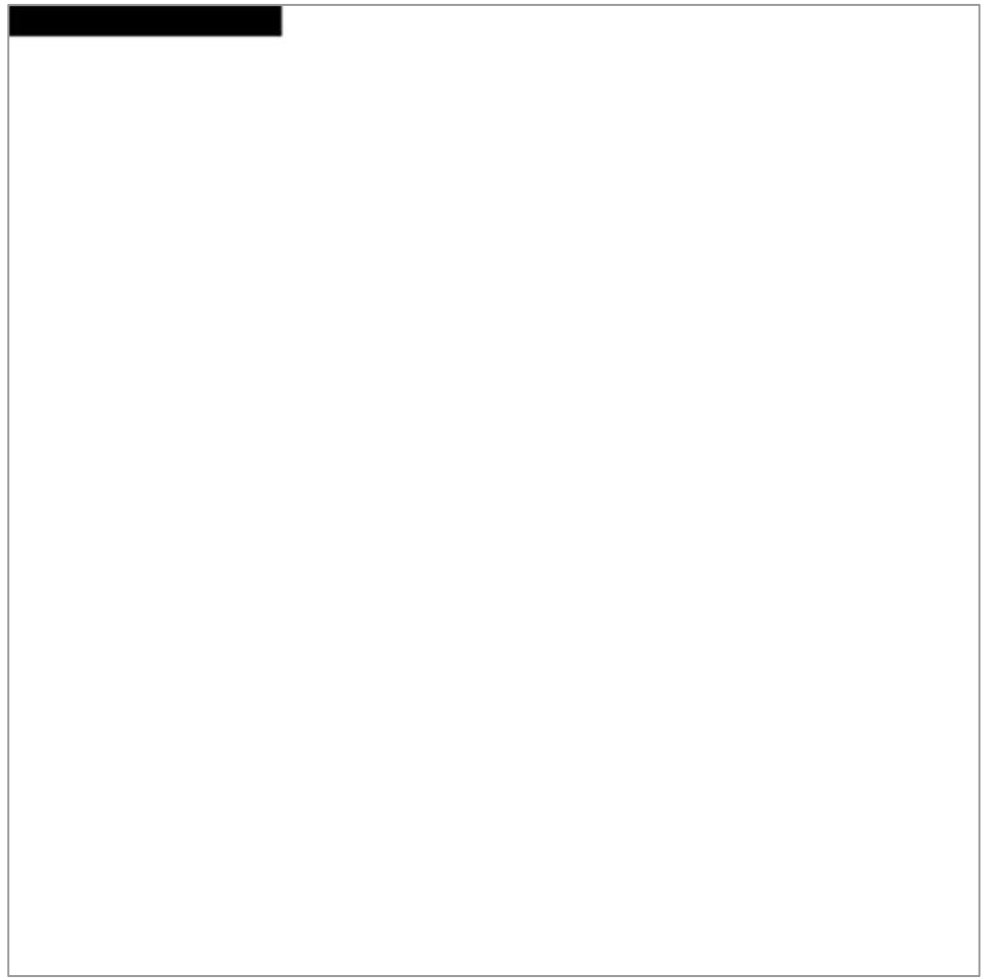
Slide by Aaron van den Oord

Softmax Sampling



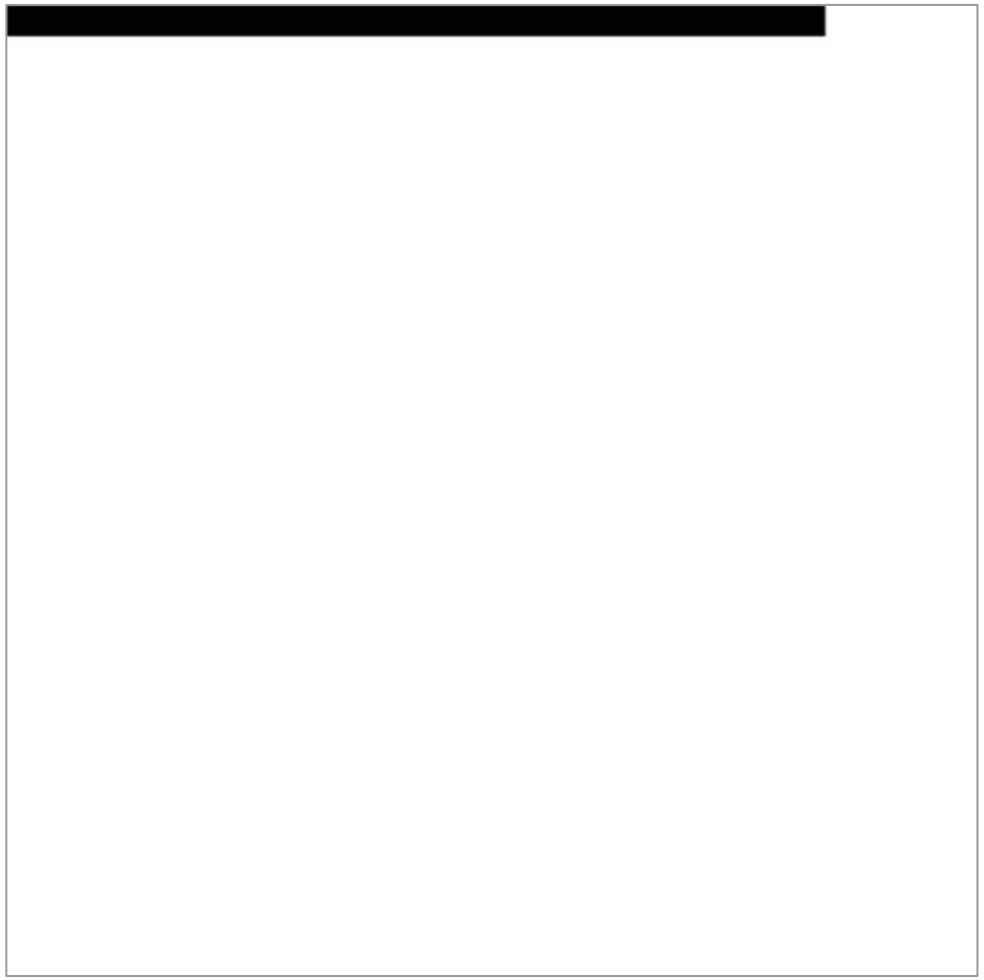
Slide by Aaron van den Oord

Softmax Sampling



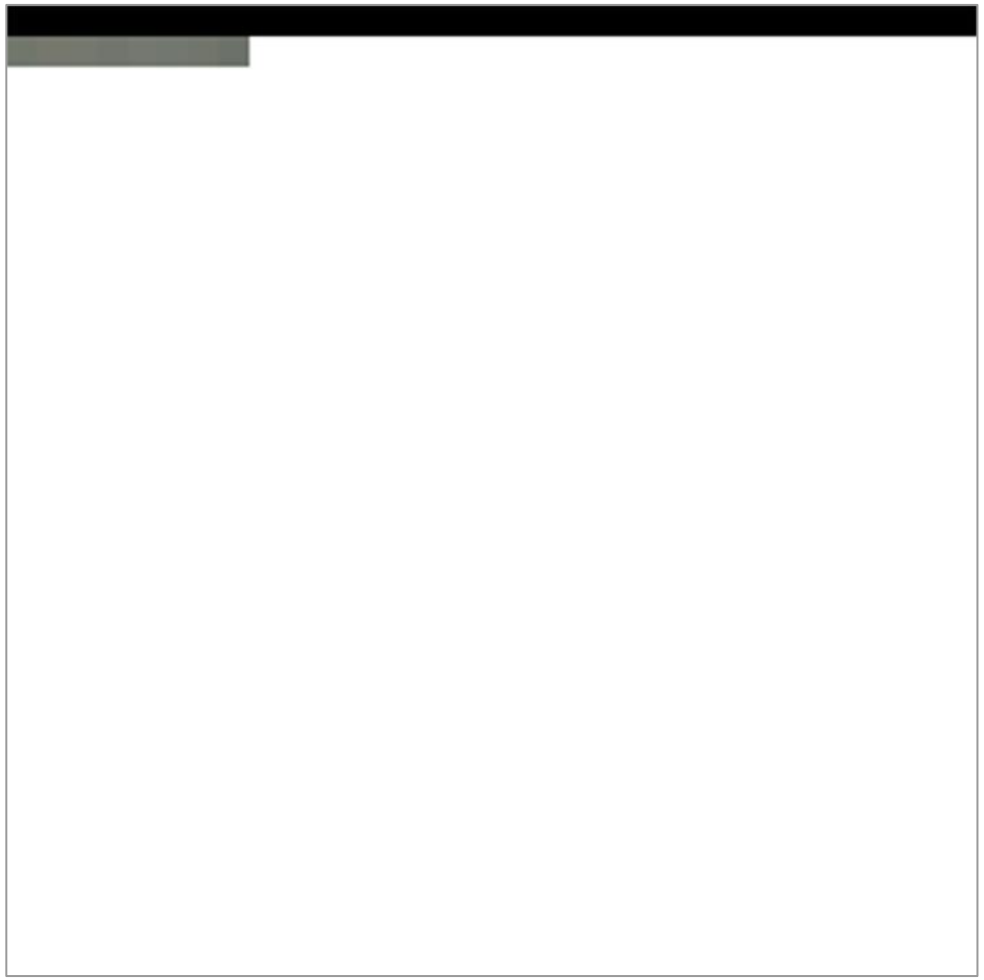
Slide by Aaron van den Oord

Softmax Sampling



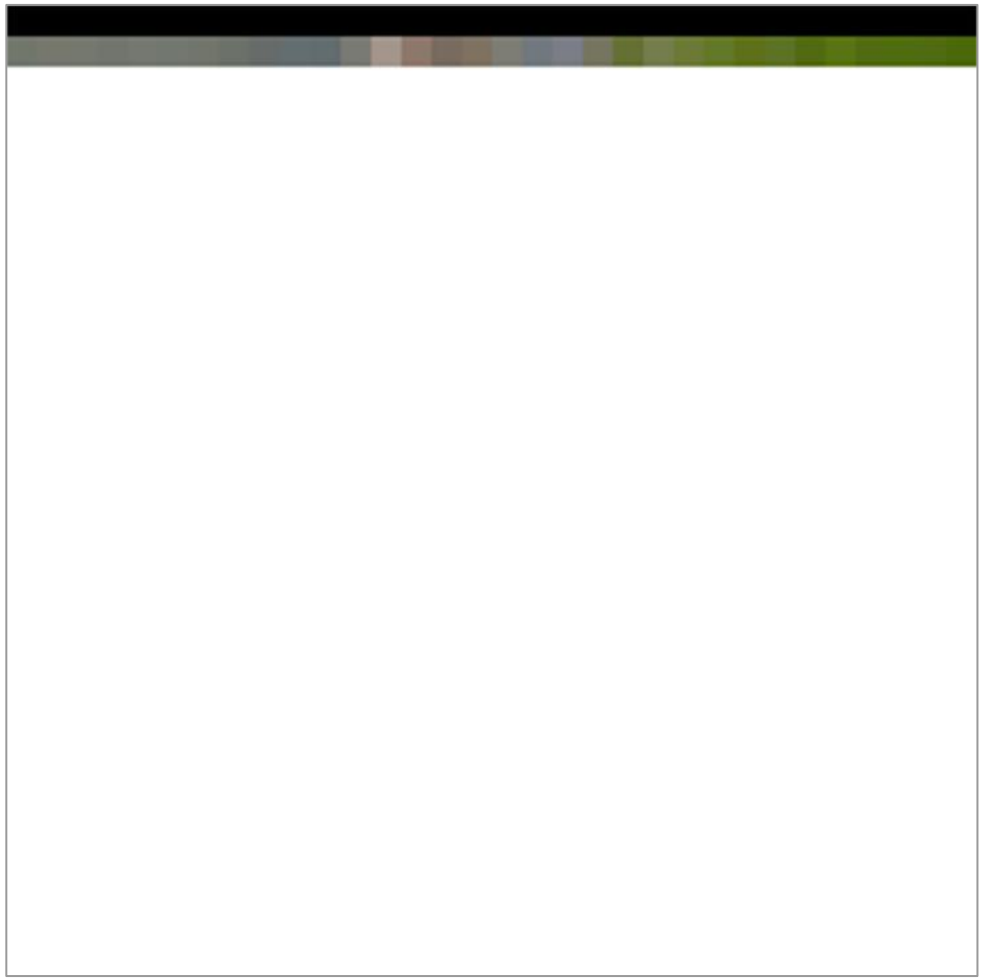
Slide by Aaron van den Oord

Softmax Sampling



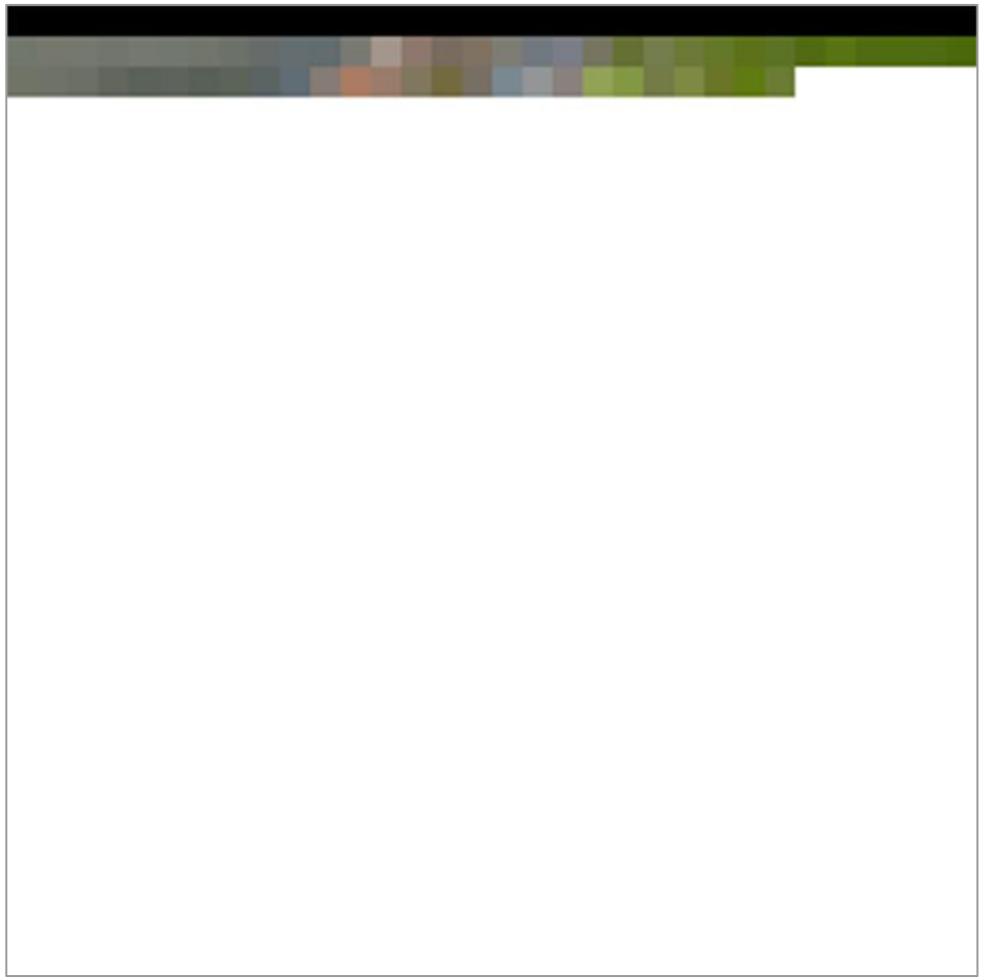
Slide by Aaron van den Oord

Softmax Sampling



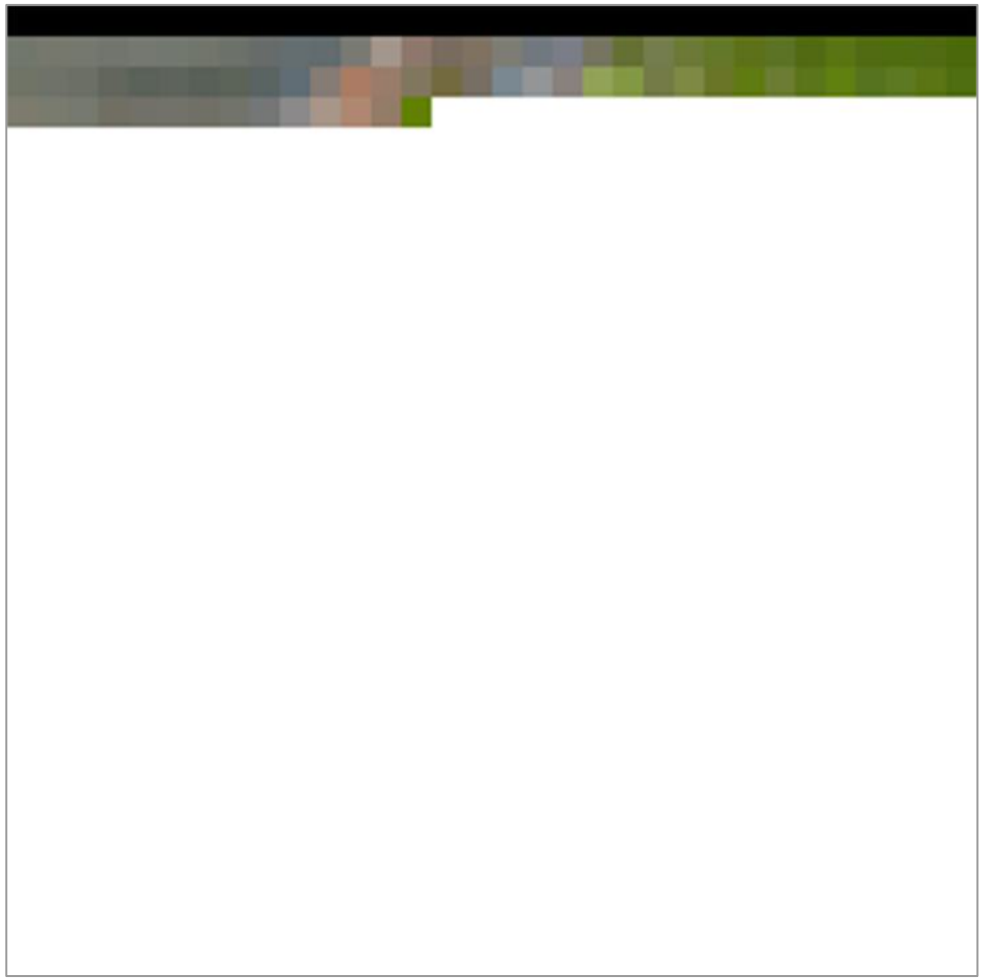
Slide by Aaron van den Oord

Softmax Sampling



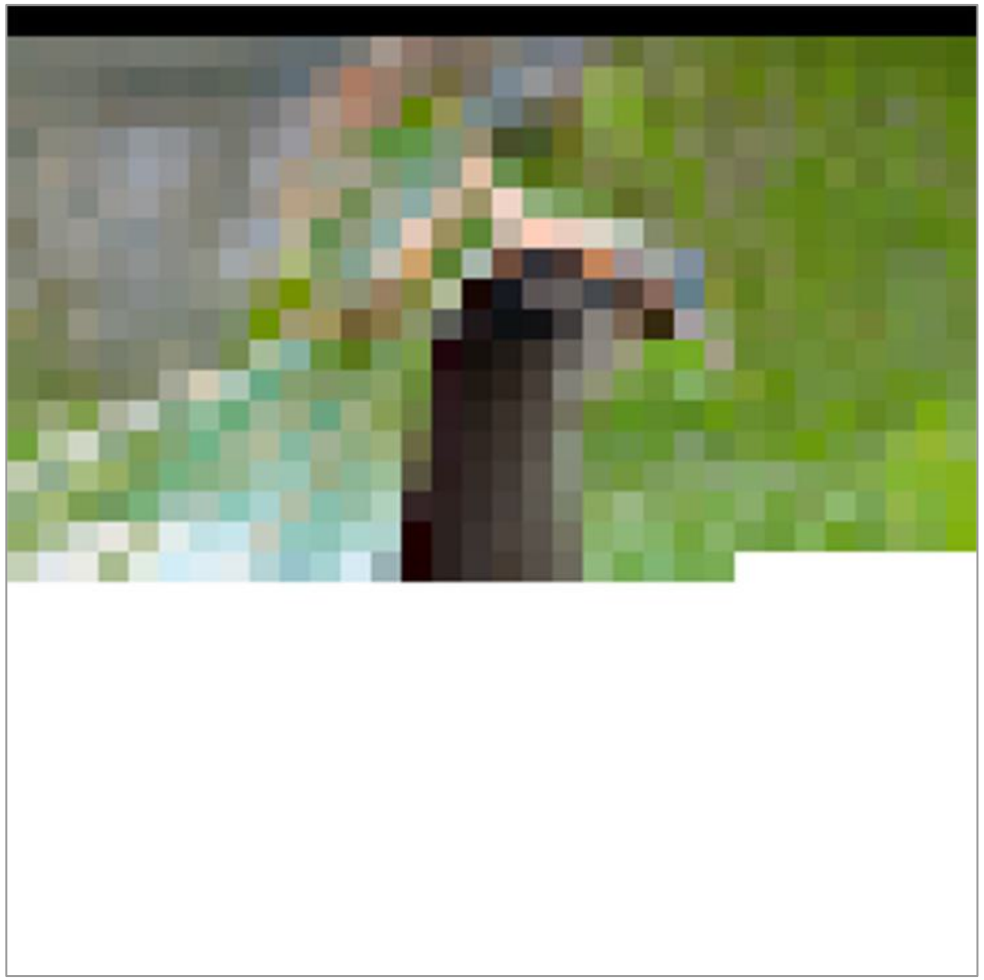
Slide by Aaron van den Oord

Softmax Sampling



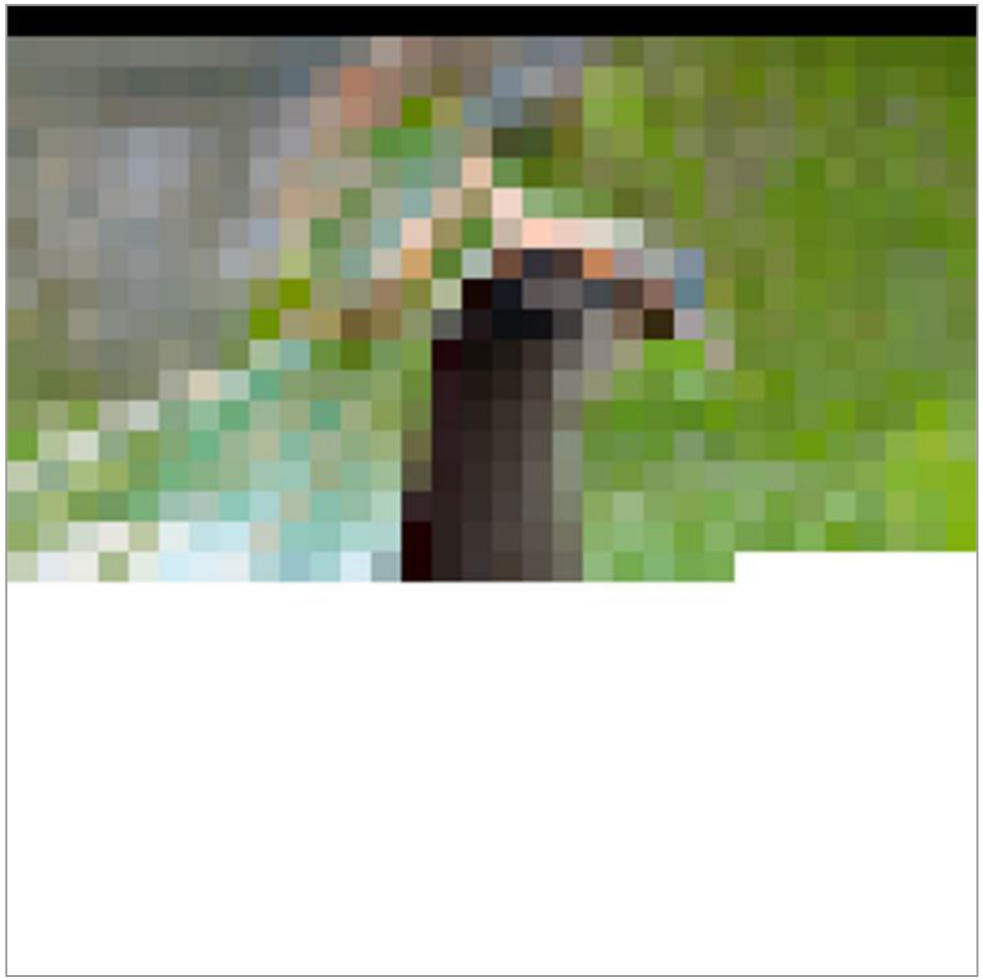
Slide by Aaron van den Oord

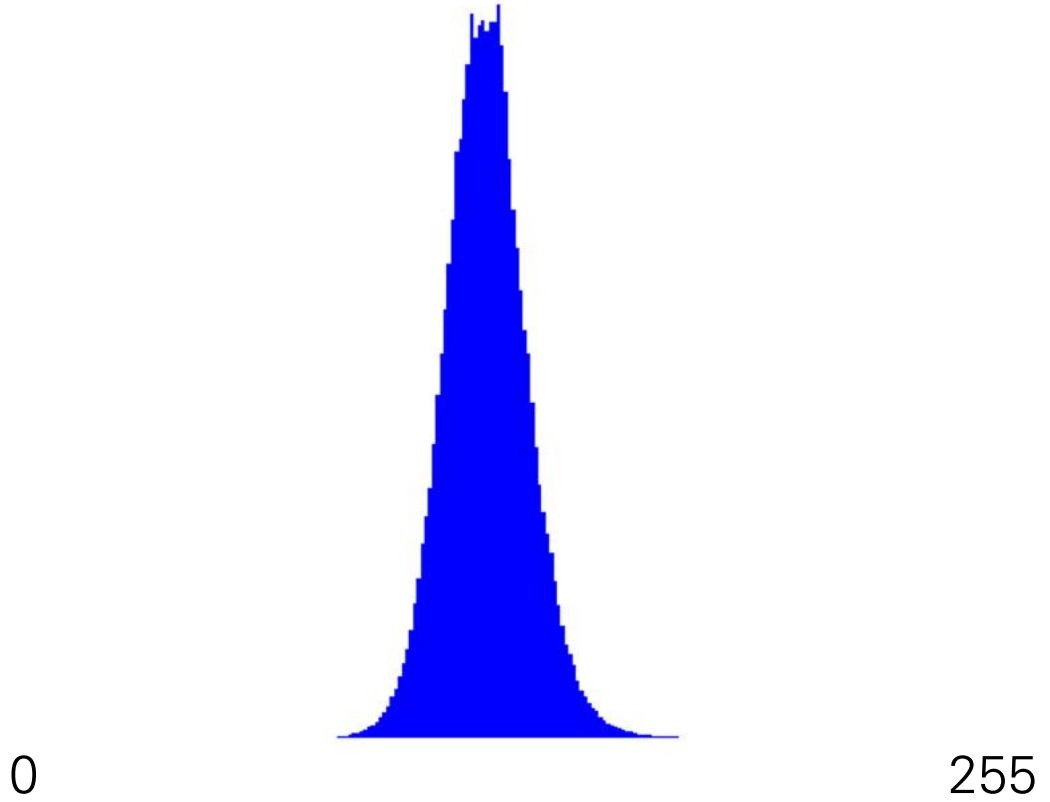
Softmax Sampling



Slide by Aaron van den Oord

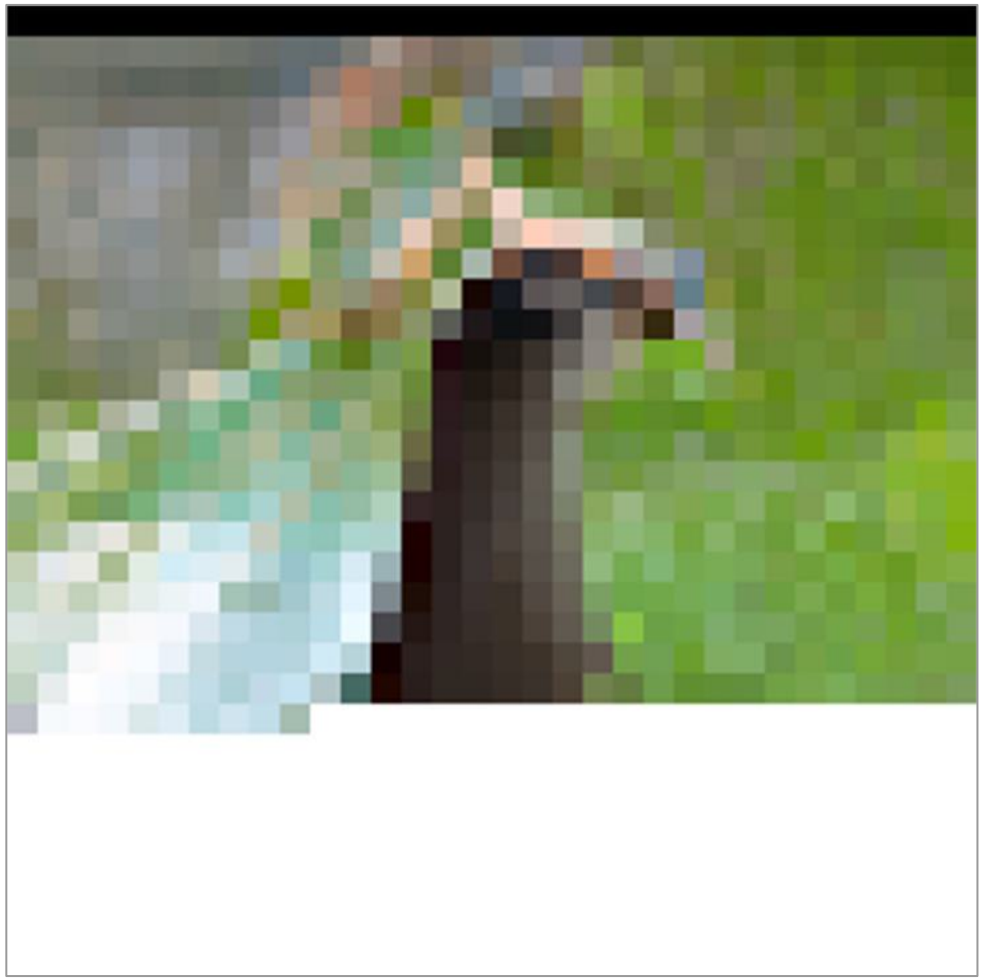
Softmax Sampling

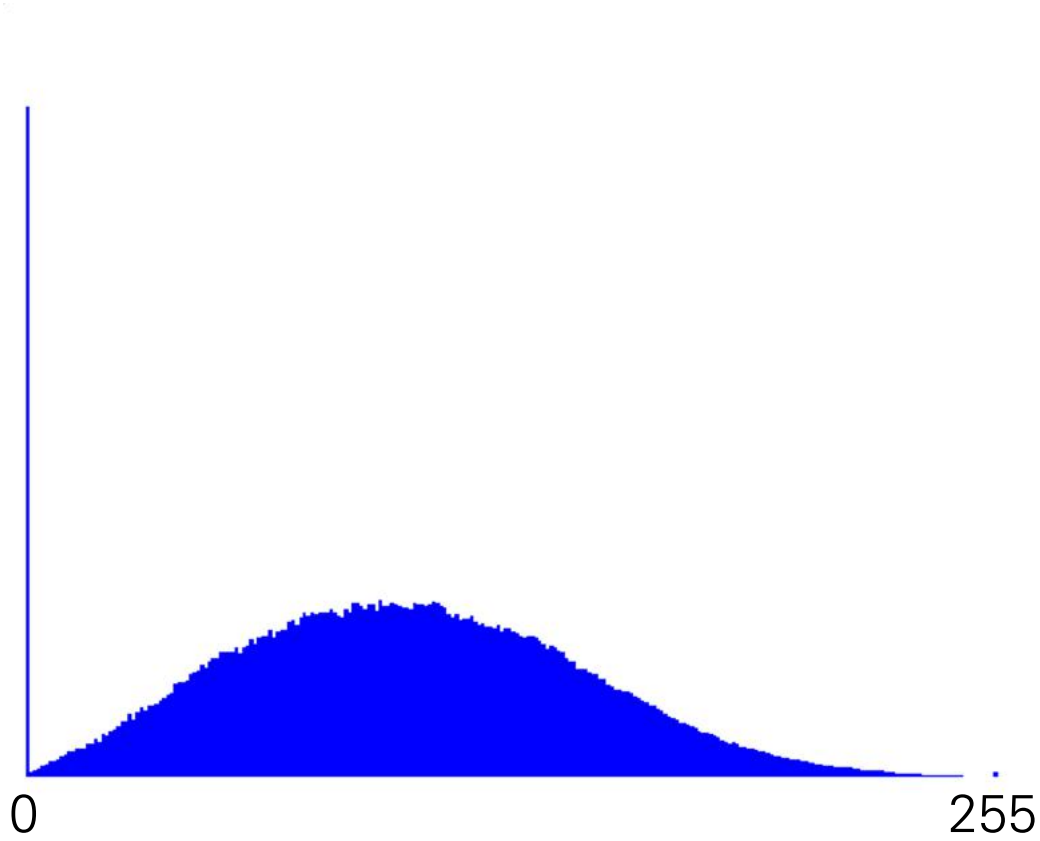




Slide by Aaron van den Oord

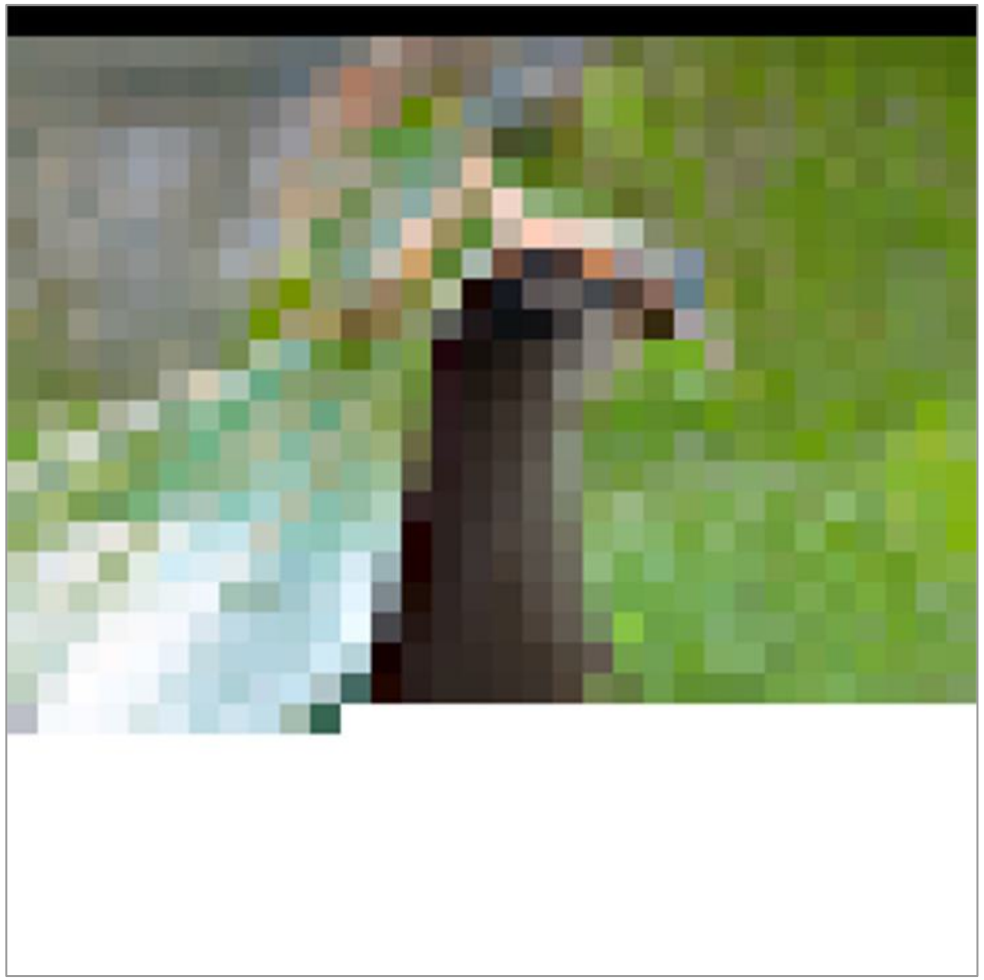
Softmax Sampling

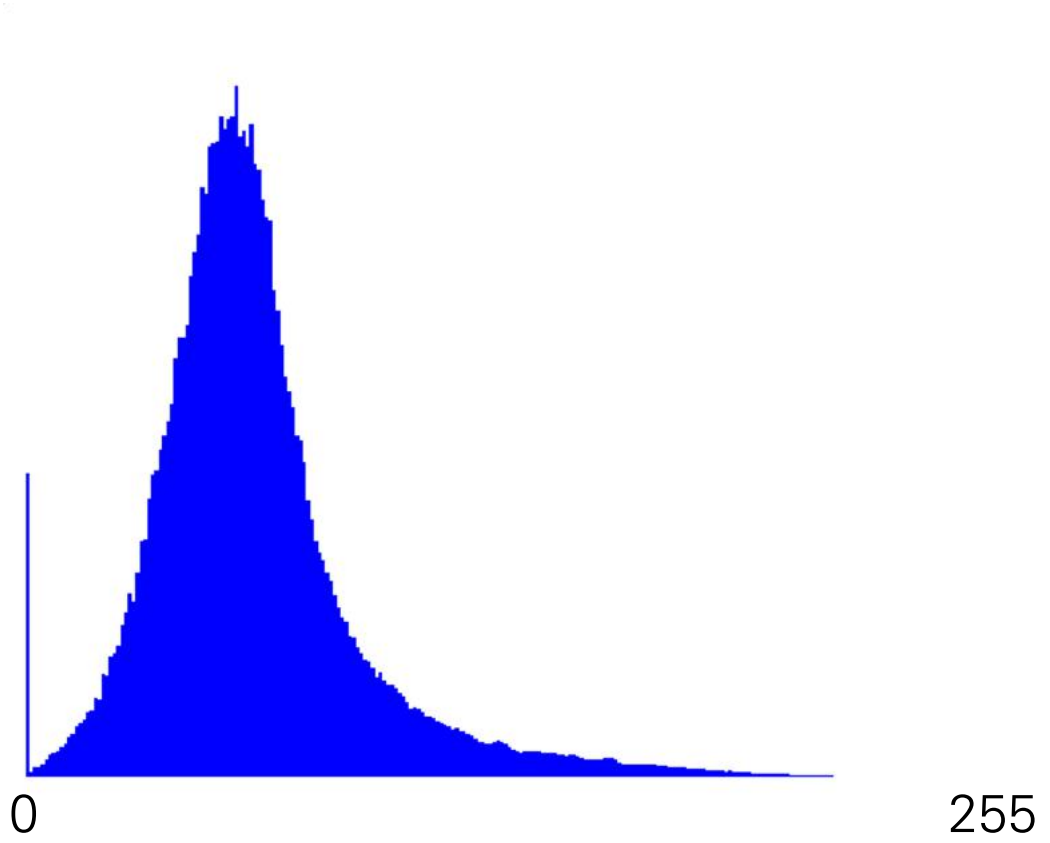




Slide by Aaron van den Oord

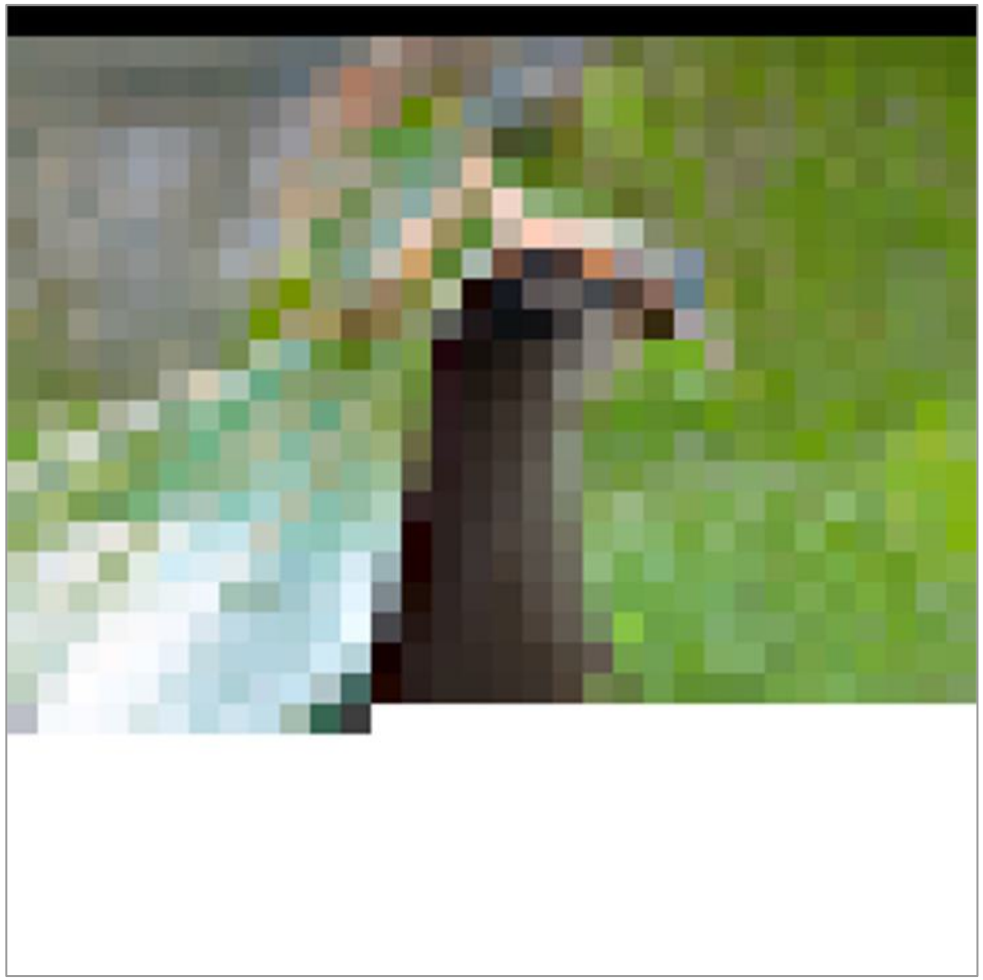
Softmax Sampling

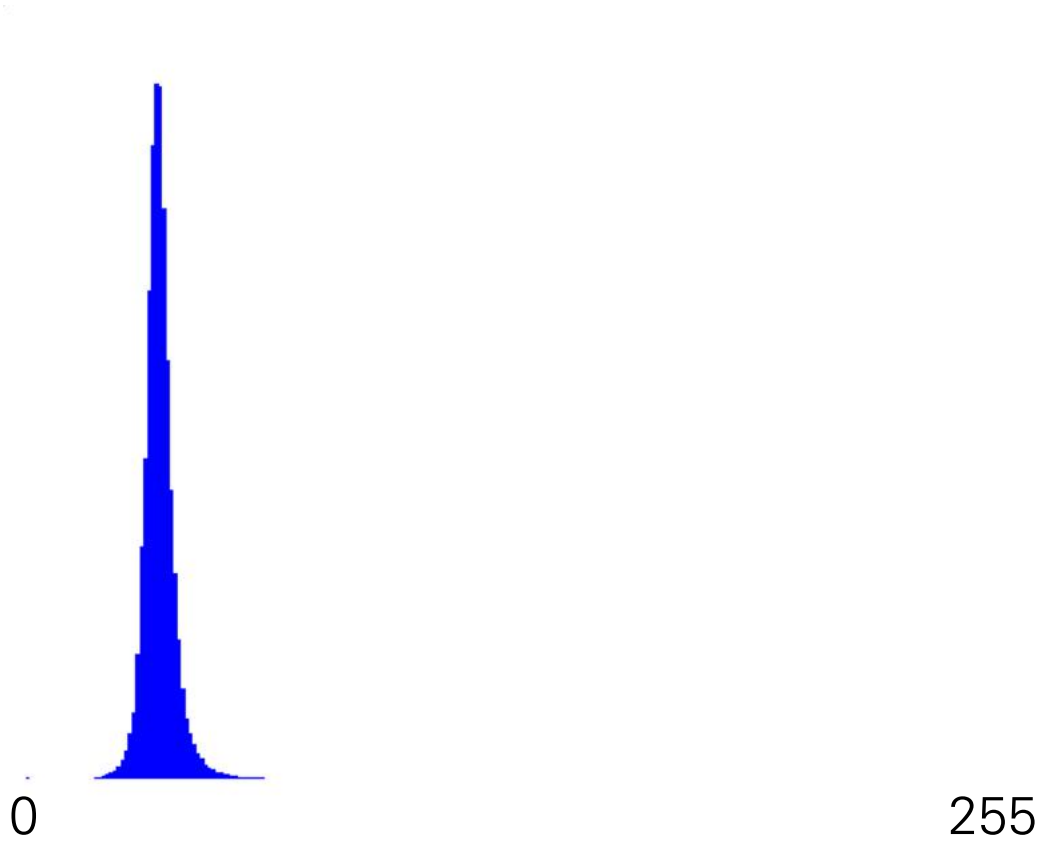




Slide by Aaron van den Oord

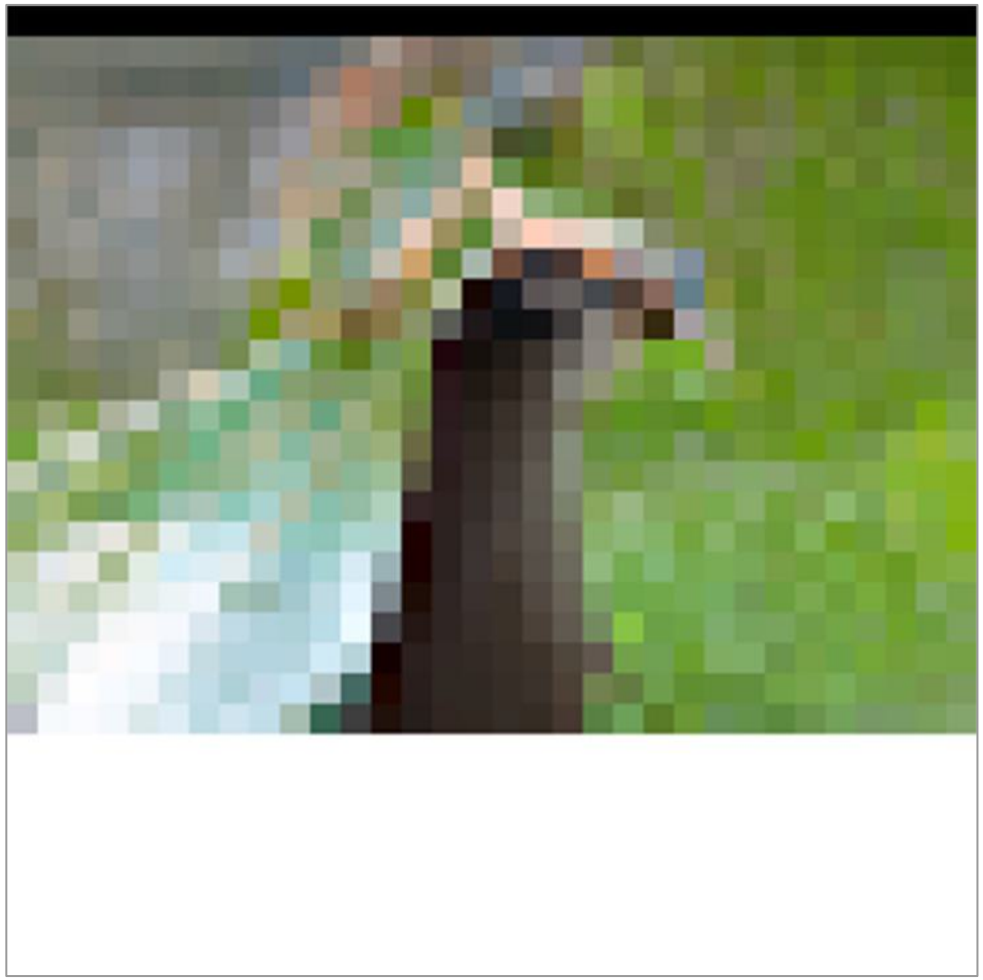
Softmax Sampling

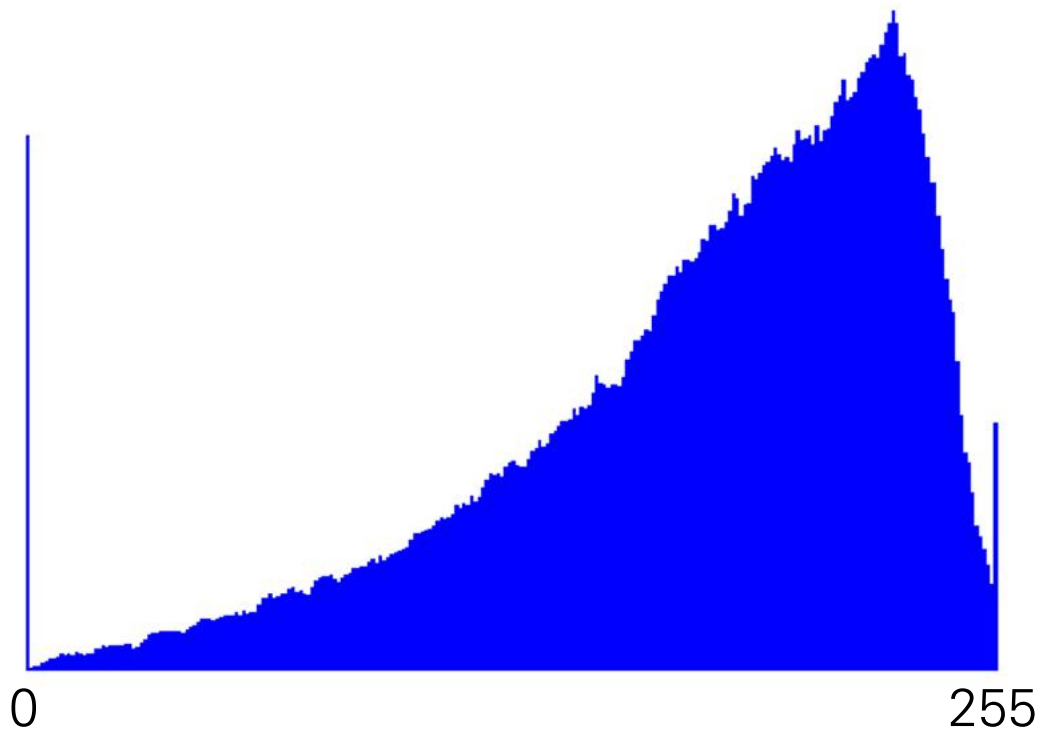




Slide by Aaron van den Oord

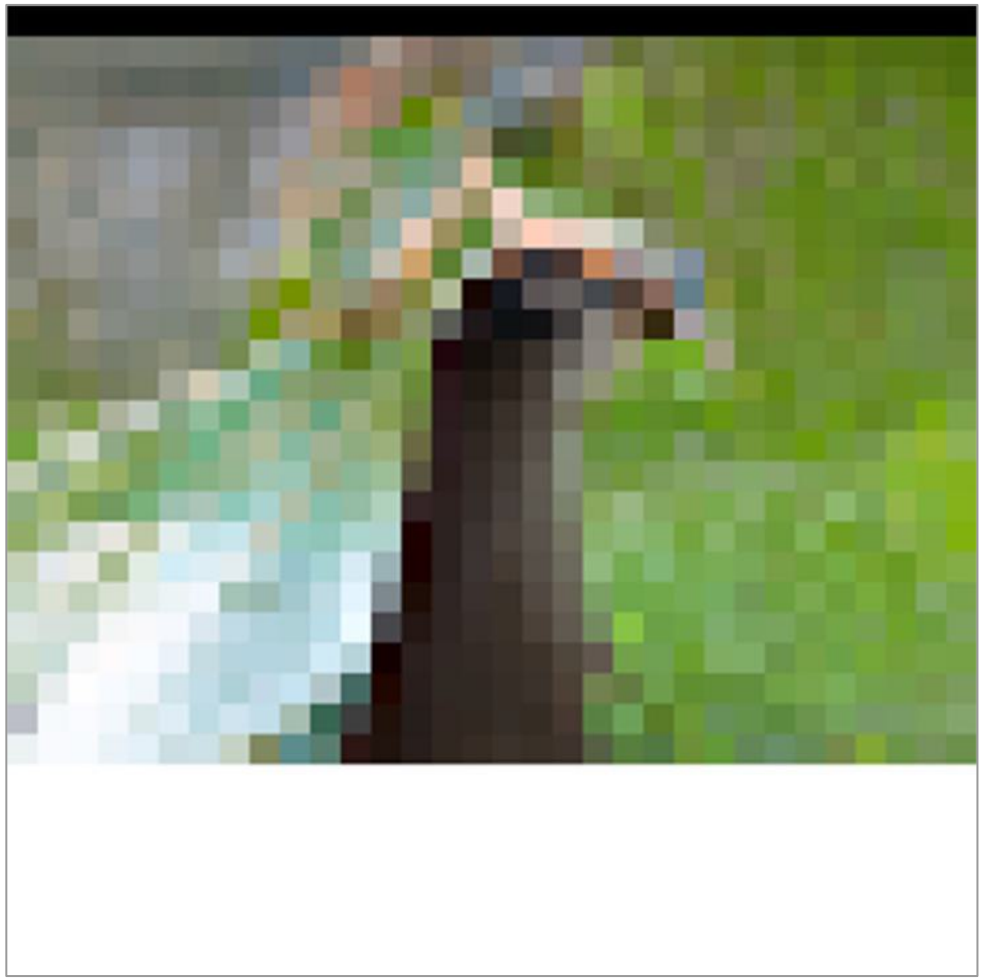
Softmax Sampling

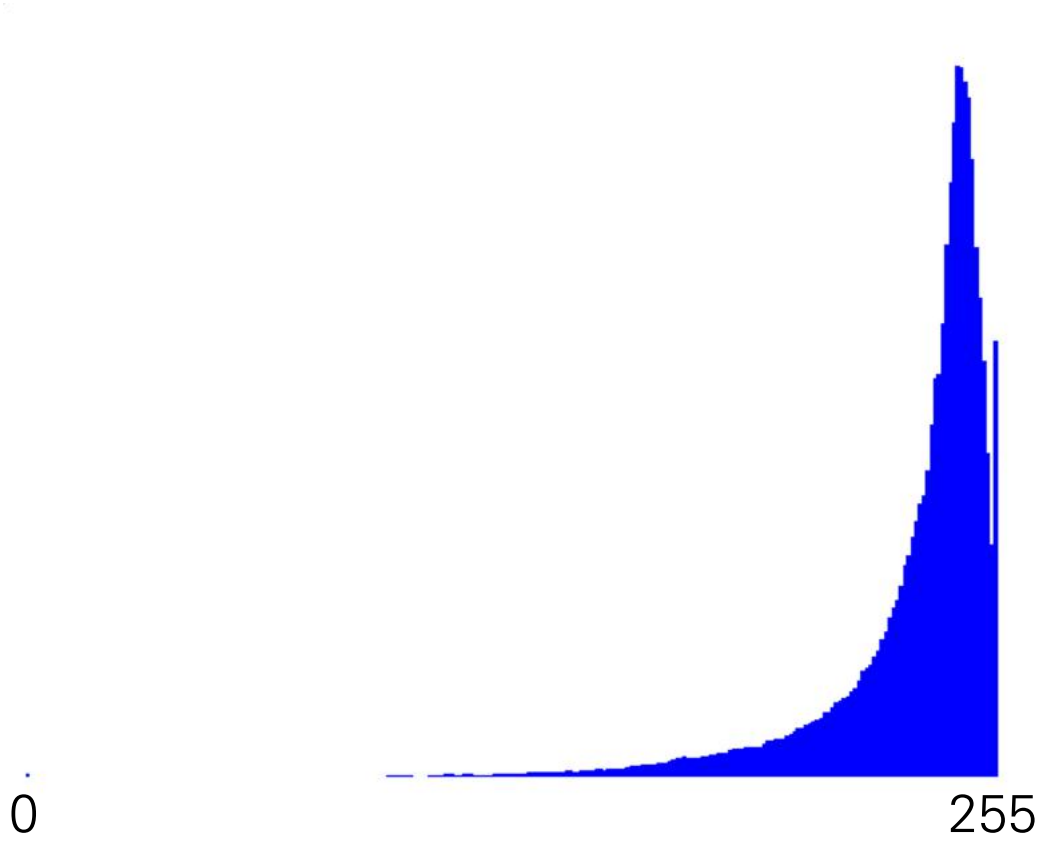




Slide by Aaron van den Oord

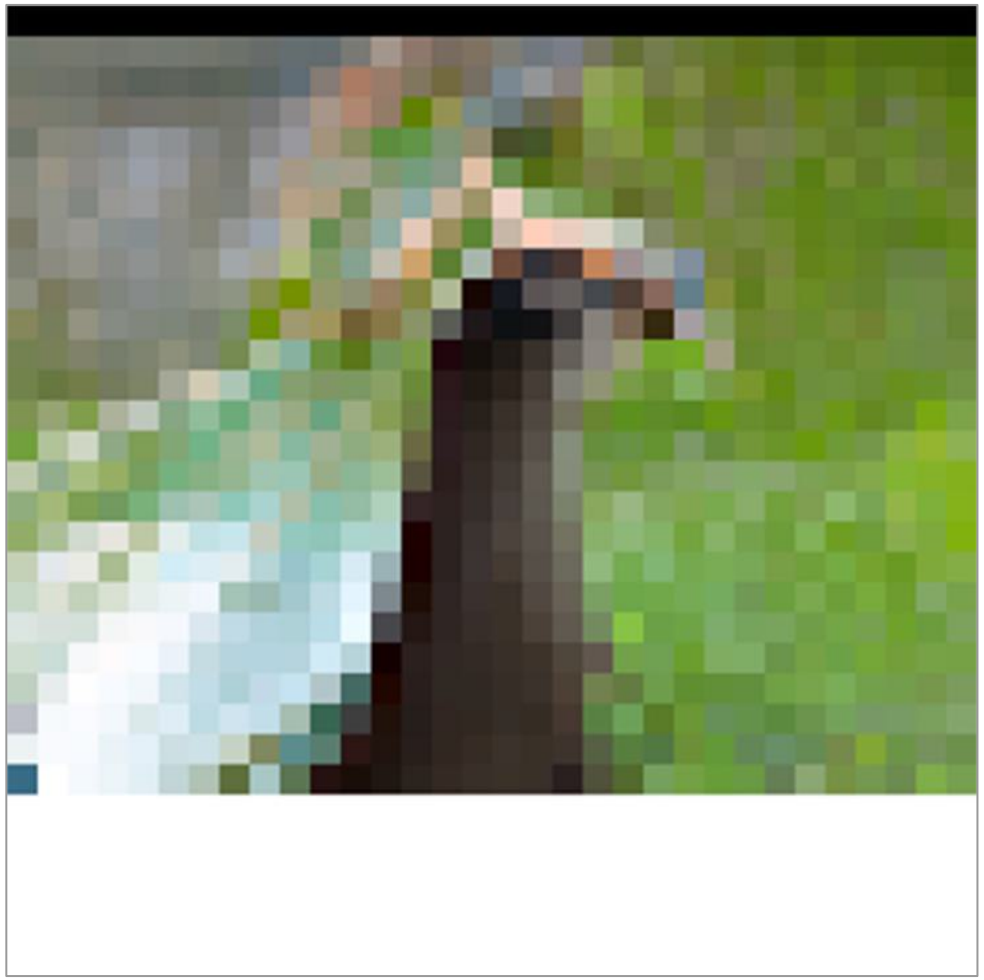
Softmax Sampling

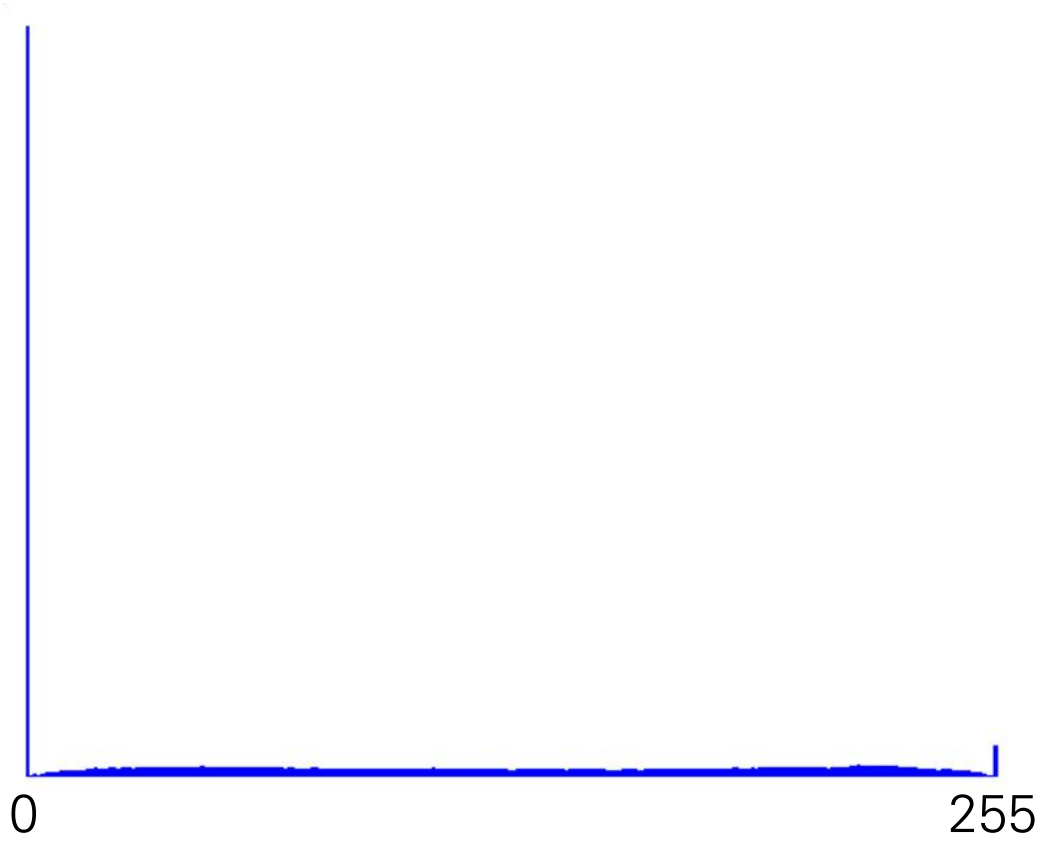




Slide by Aaron van den Oord

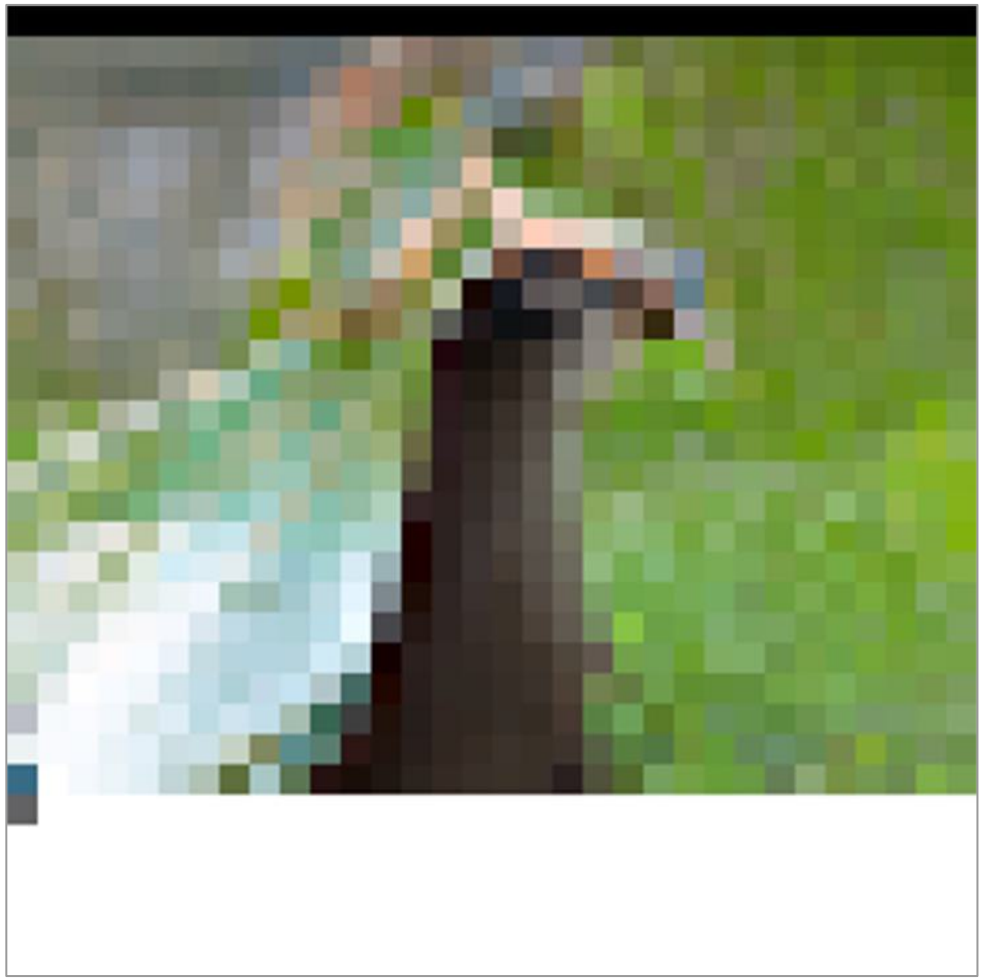
Softmax Sampling

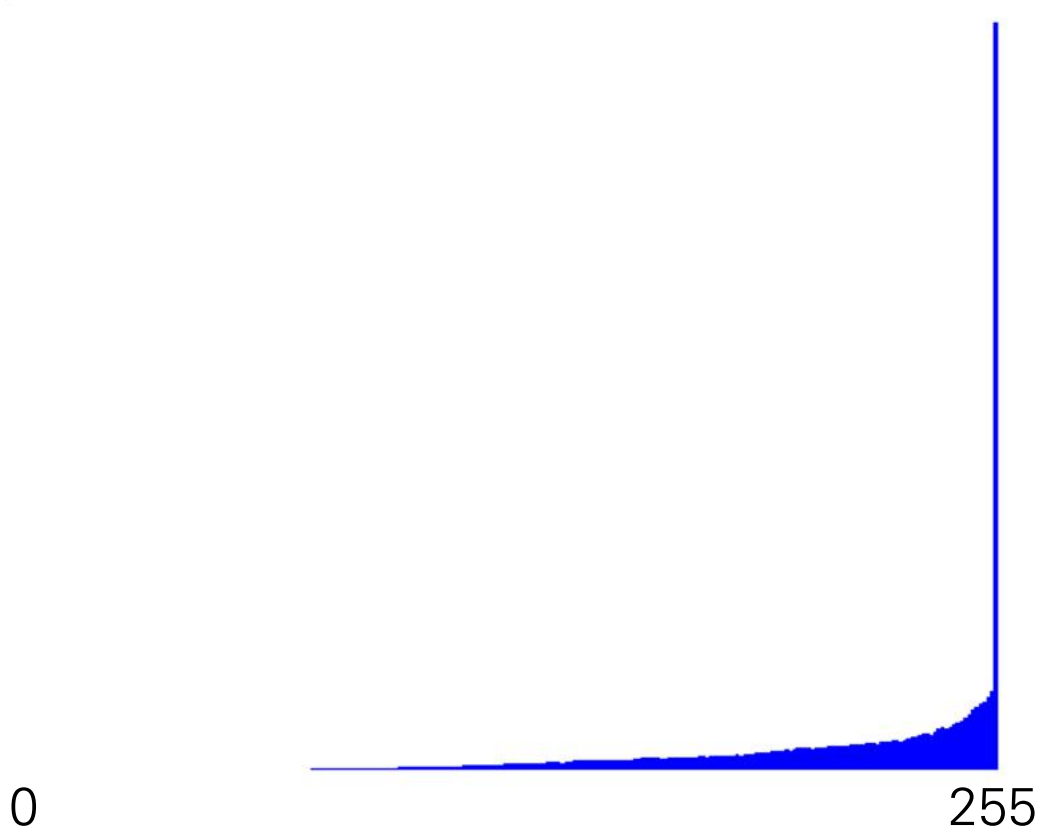




Slide by Aaron van den Oord

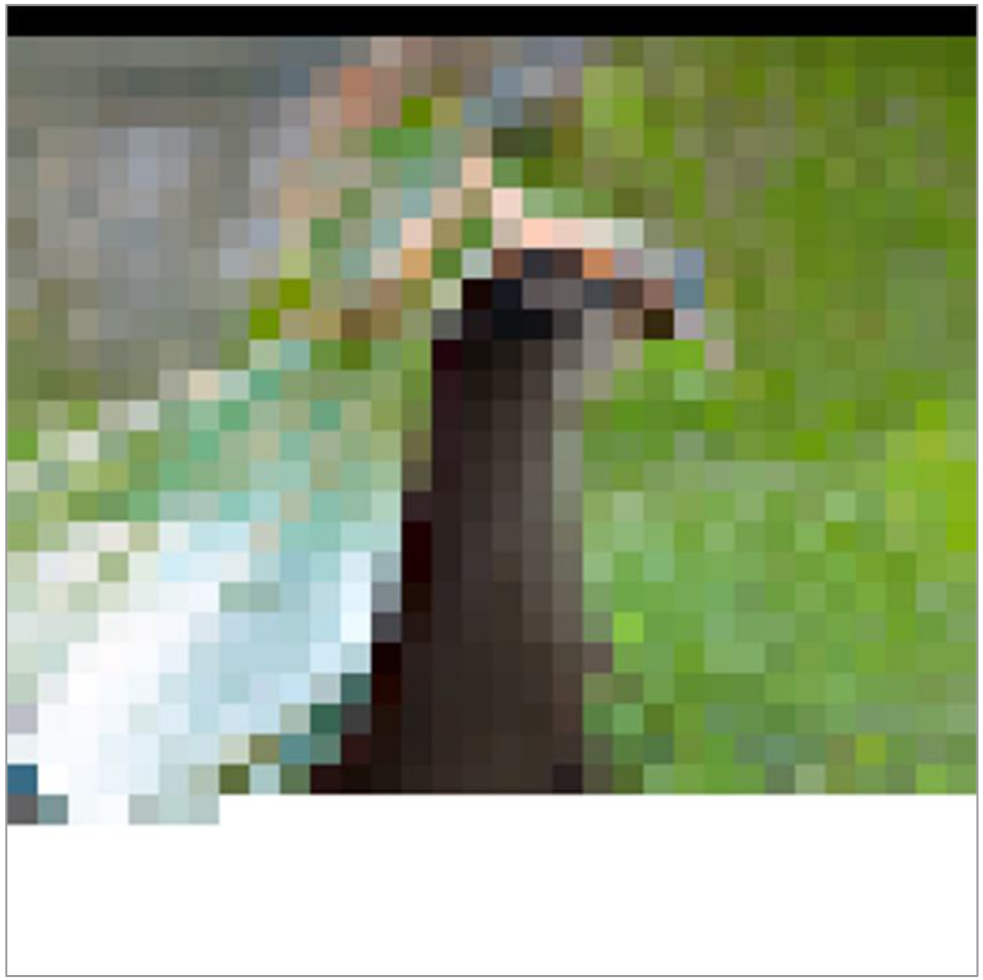
Softmax Sampling

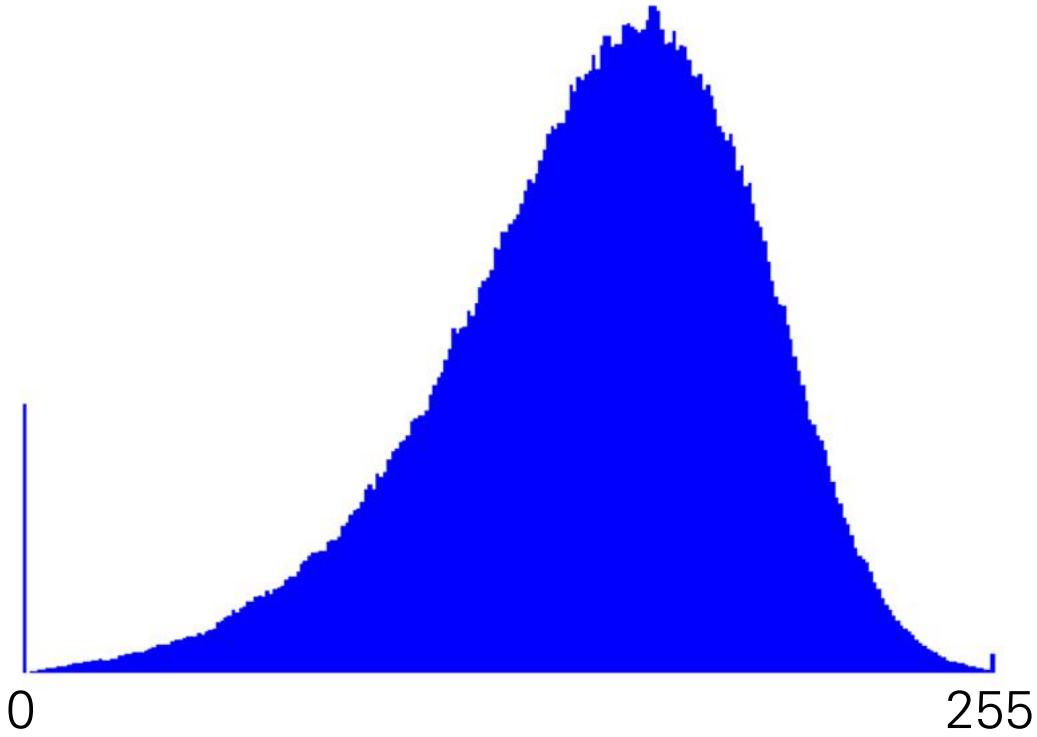




Slide by Aaron van den Oord

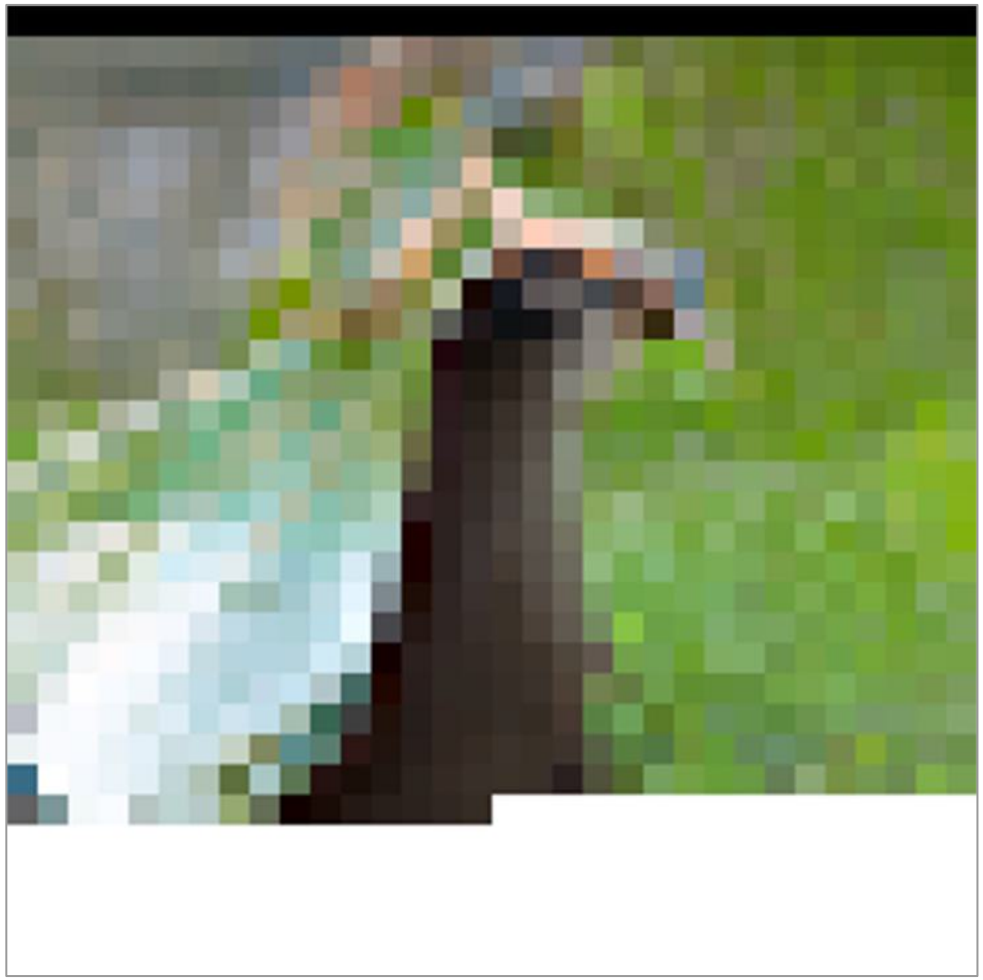
Softmax Sampling

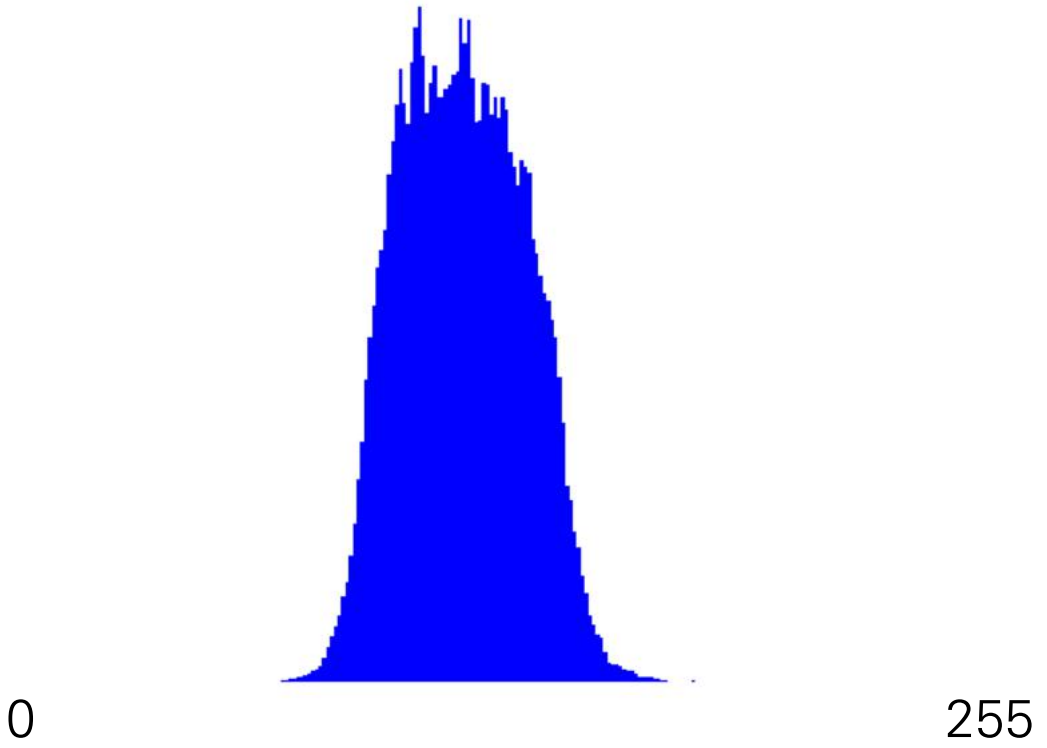




Slide by Aaron van den Oord

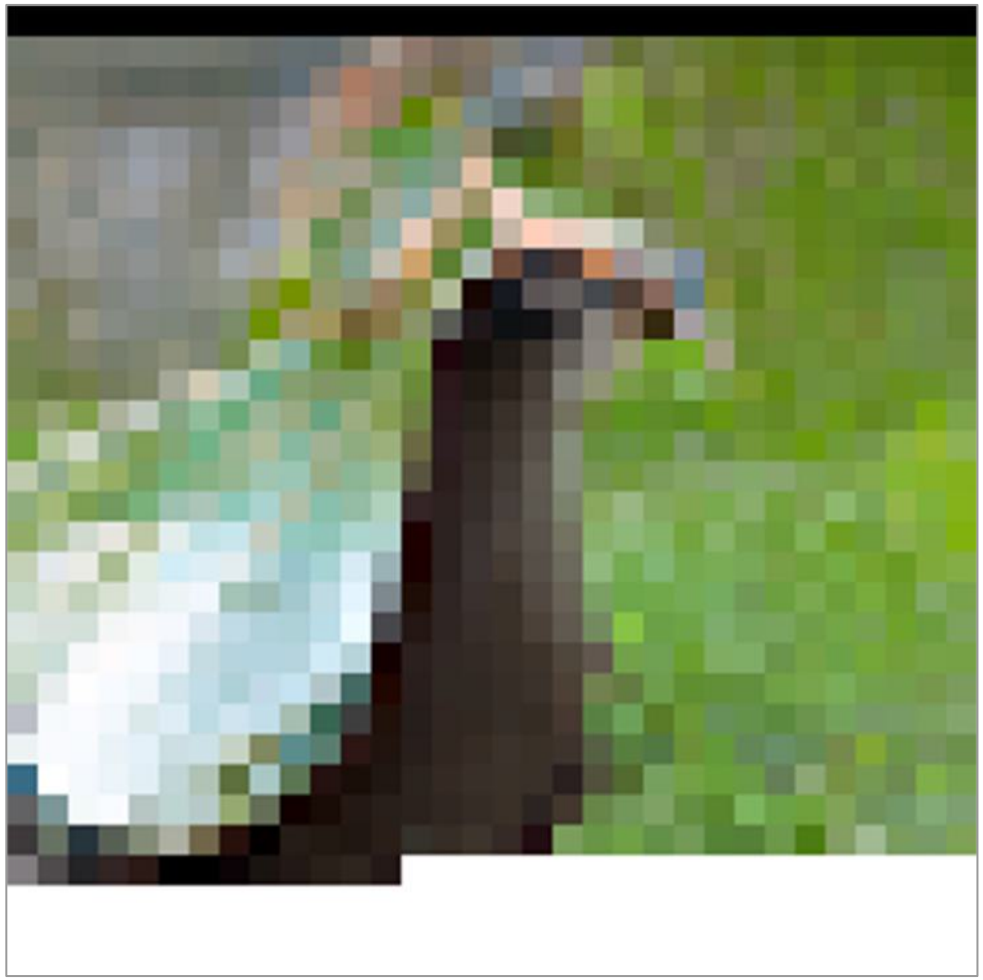
Softmax Sampling

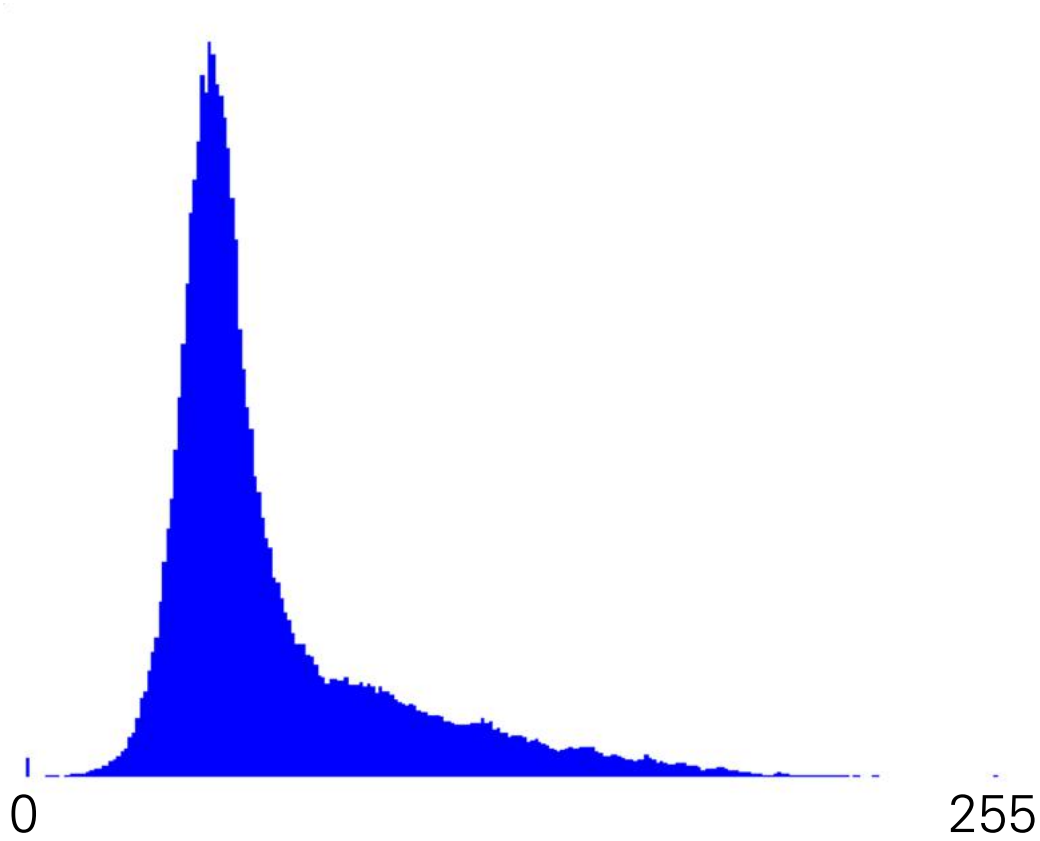




Slide by Aaron van den Oord

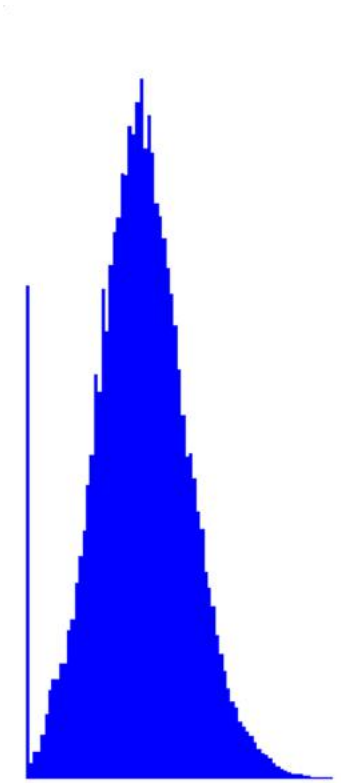
Softmax Sampling





Slide by Aaron van den Oord

Softmax Sampling

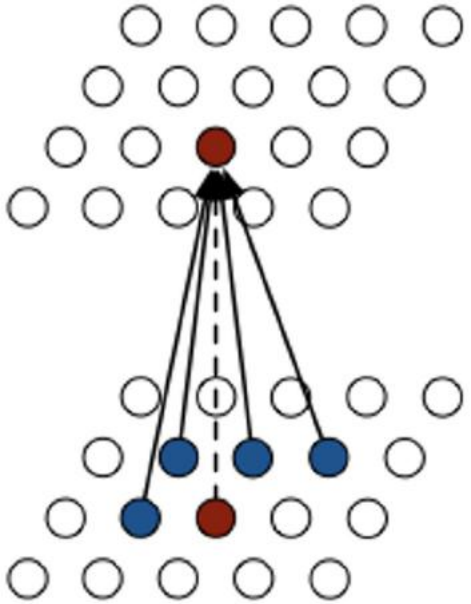


Slide by Aaron van den Oord

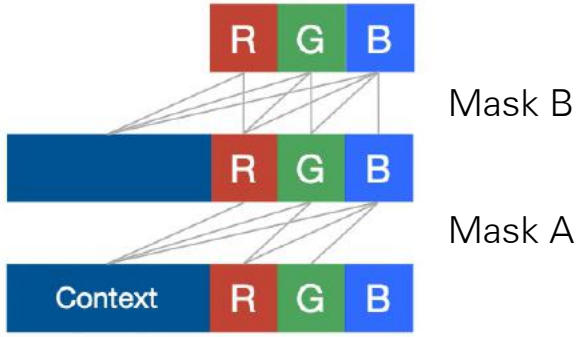
PixelCNN

- Design question: how to design a masking method to obey that ordering?
- One possibility: PixelCNN (2016)

1	1	1
1	0	0
0	0	0

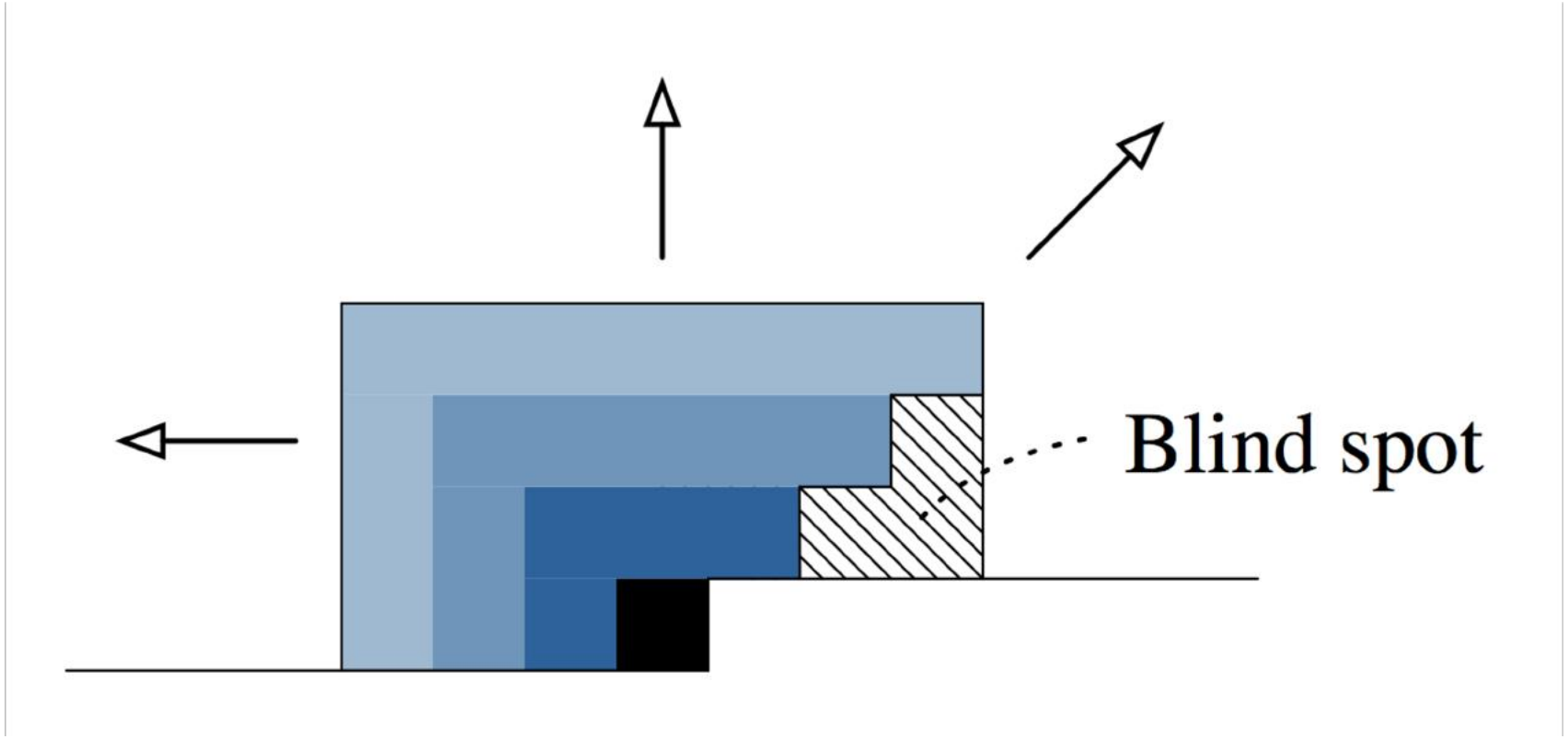


PixelCNN



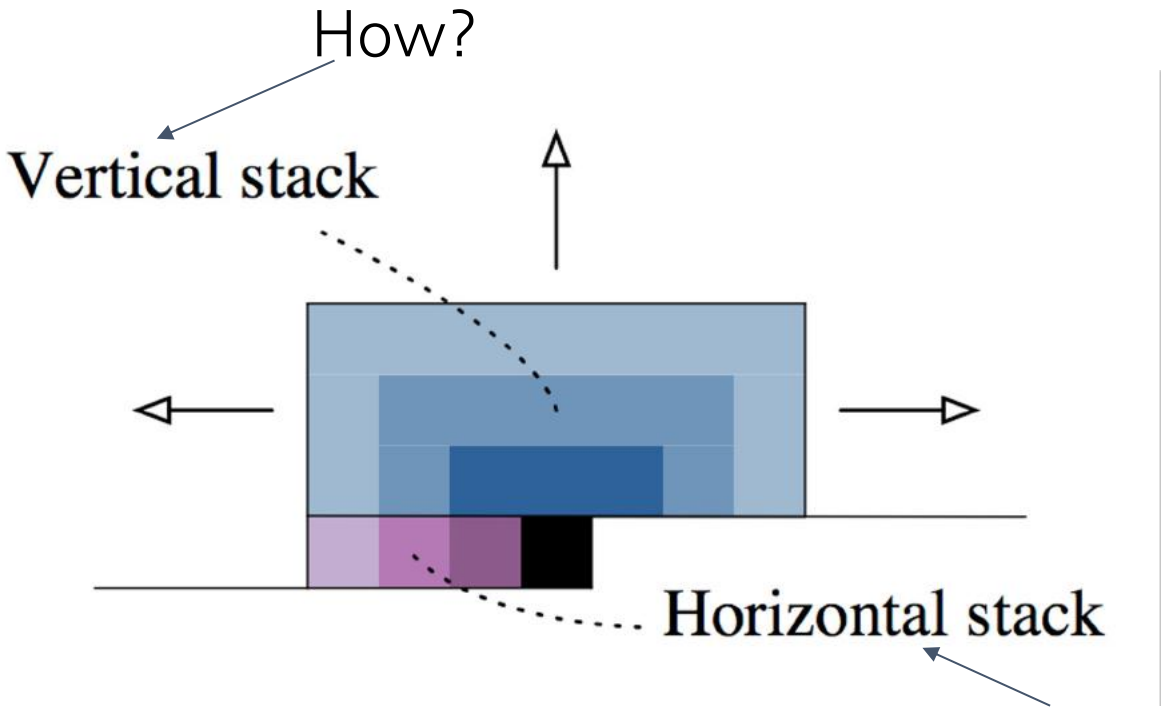
PixelCNN

- PixelCNN-style masking has one problem: blind spot in receptive field



Gated PixelCNN

- Gated PixelCNN (2016) introduced a fix by combining two streams of convolutions



This is easy, we know how to do 1D masked conv

Recap: Maximum likelihood

- Maximum likelihood: given a dataset $x^{(1)}, \ldots, x^{(n)}$, find θ by solving the optimization problem

$$\arg\min_{\theta} \ \text{loss}(\theta, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}) = \frac{1}{n} \sum_{i=1}^{n} -\log p_\theta(\mathbf{x}^{(i)})$$

- Statistics tells us that if the model family is expressive enough and if enough data is given, then solving the maximum likelihood problem will yield parameters that generate the data
- Equivalent to minimizing KL divergence between the empirical data distribution and the model

$$\hat{p}_{\text{data}}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[\mathbf{x} = \mathbf{x}^{(i)}]$$

$$\text{KL}(\hat{p}_{\text{data}} \,\|\, p_\theta) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}}[-\log p_\theta(\mathbf{x})] - H(\hat{p}_{\text{data}})$$

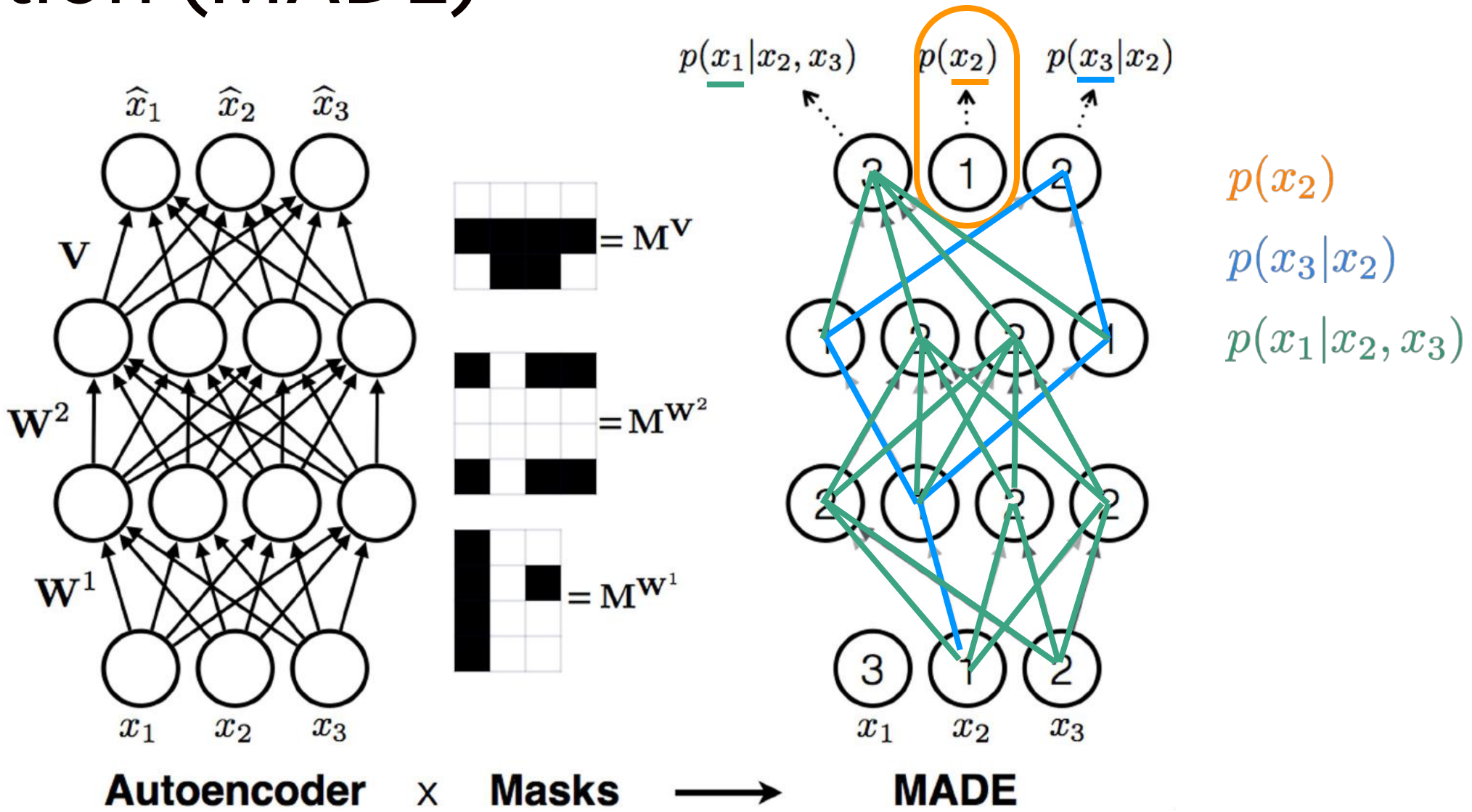
Recap: Conditional Distribution Modeling

- Any (multi-variable) joint distribution can be written as a product of conditionals

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{1:i-1})$$

- This is called an **autoregressive model**.

Recap: Masked Autoencoder for Distribution Estimation (MADE)

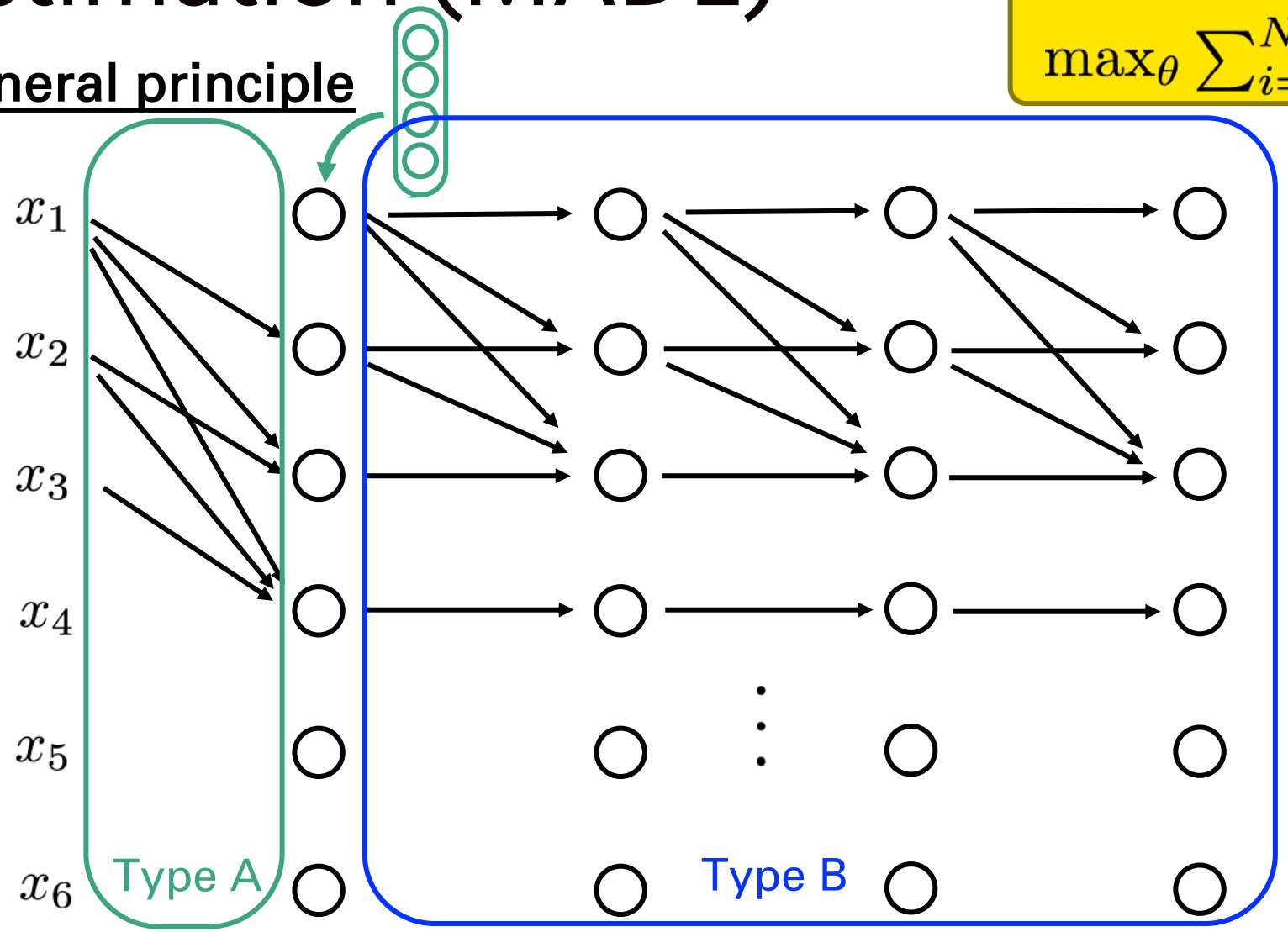


Recap: Masked Autoencoder for Distribution Estimation (MADE)

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdots p(x_6|x_{1:5})$$
$$\max_\theta \sum_{i=1}^{N} \sum_k \log p_\theta(x_k^{(i)}|x_{1:k-1}^{(i)})$$

Sampling is slow!

General principle



$p(x_1)$

$p(x_2|x_1)$

$p(x_3|x_{1:2})$

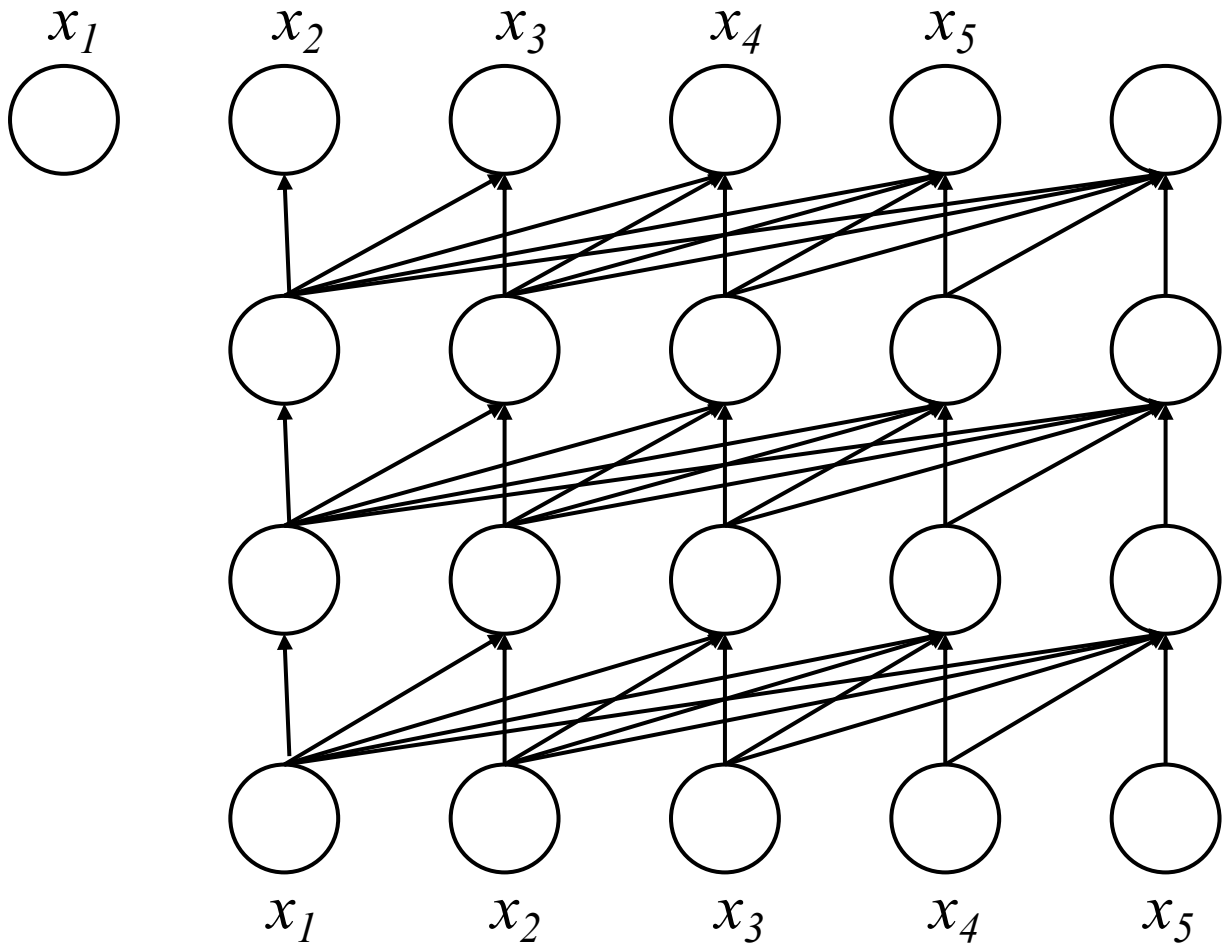
$p(x_4|x_{1:3})$

$p(x_5|x_{1:4})$

$p(x_6|x_{1:5})$

Recap: Masked Autoencoder for Distribution Estimation (MADE)

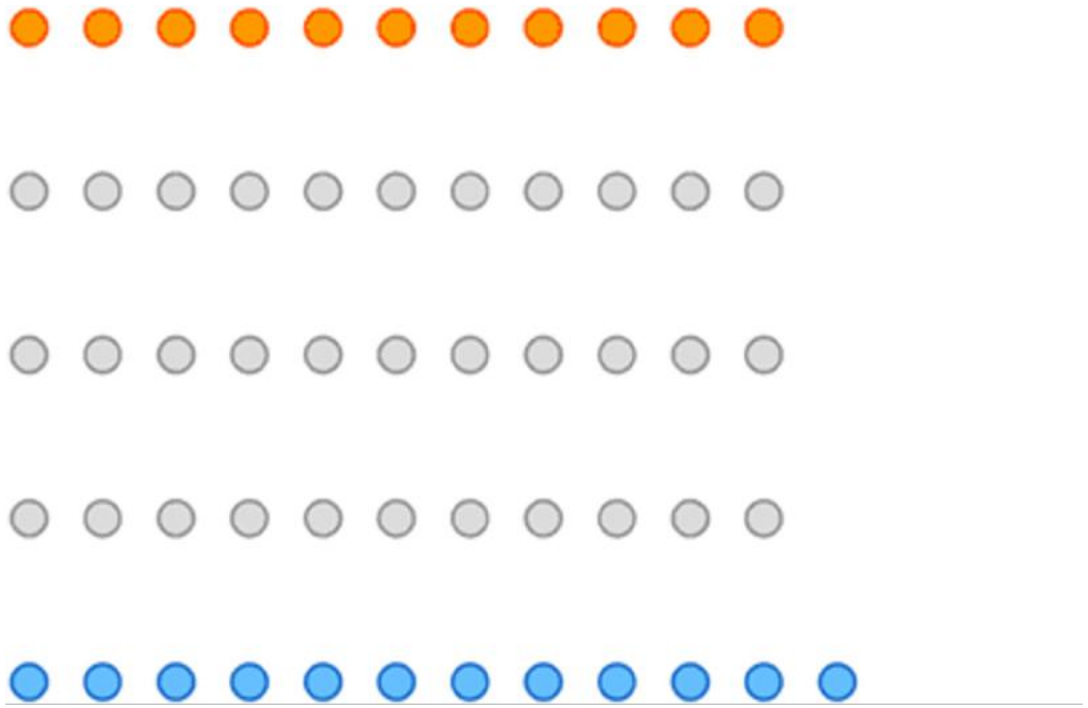
In more modern notation/diagram



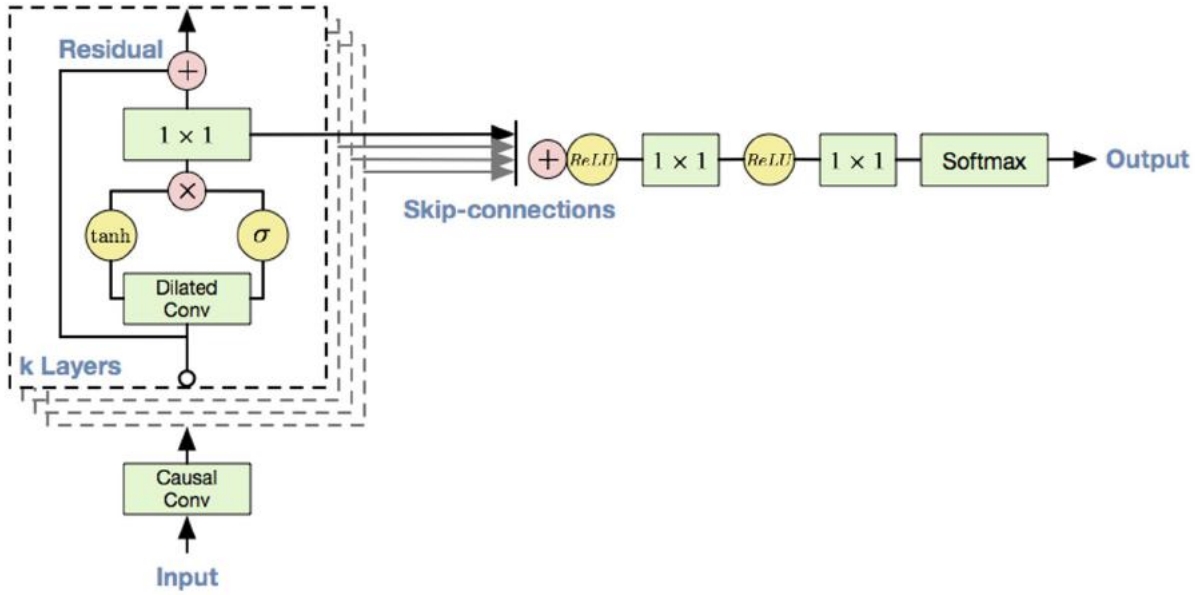
Every hidden layer node can be vector valued

Every edge can be its own MLP connection

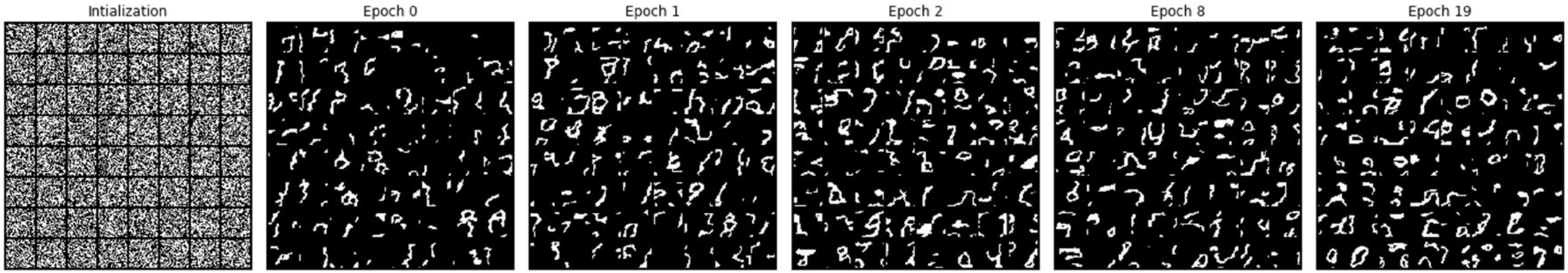
Recap: WaveNet



- Improved receptive field: dilated convolution, with exponential dilation
- Better expressivity: Gated Residual blocks, Skip connections

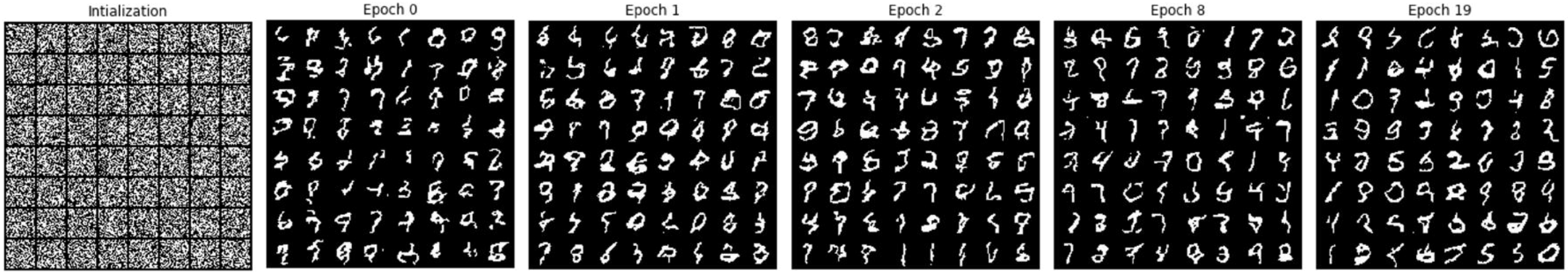


Recap: WaveNet on MNIST



Recap: WaveNet with Pixel Location Appended on MNIST

- Append (x,y) coordinates of pixel in the image as input to WaveNet



- expressive and efficient, but finite context window!

Recap: RNN with Pixel Locations on MNIST

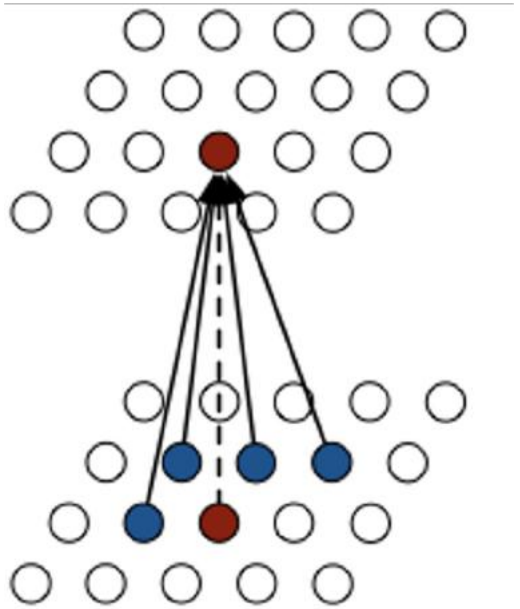
- Append (x,y) coordinates of pixel in the image as input to RNN

- parameter sharing + “infinite look-back”
- hard to train

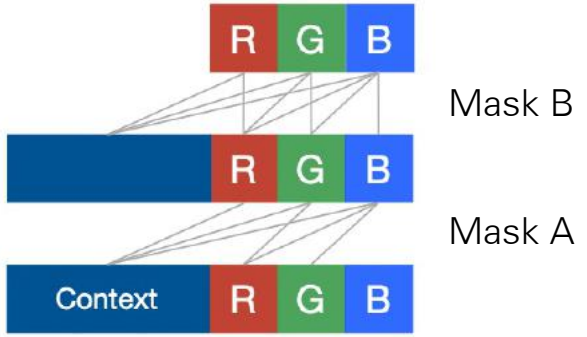
Recap: PixelCNN

- Images can be flatten into 1D vectors, but they are fundamentally 2D
- We can use masked spatial (2D) convolutiont to exploit this knowledge
- First, we impose an autoregressive ordering on 2D images:

1	1	1
1	0	0
0	0	0



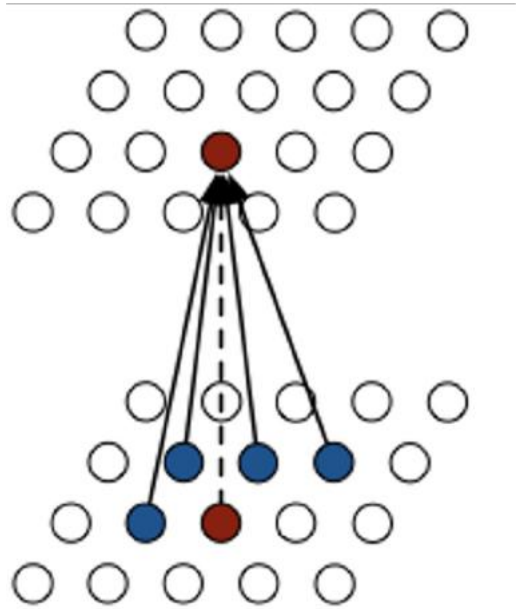
PixelCNN



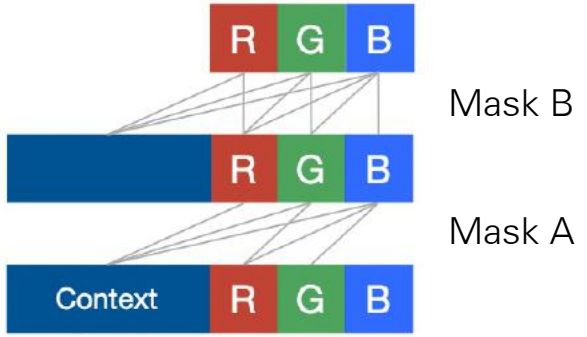
Recap: PixelCNN

- Images can be flatten into 1D vectors, but they are fundamentally 2D
- We can use masked spatial (2D) convolutiont to exploit this knowledge
- First, we impose an autoregressive ordering on 2D images:

1	1	1
1	0	0
0	0	0

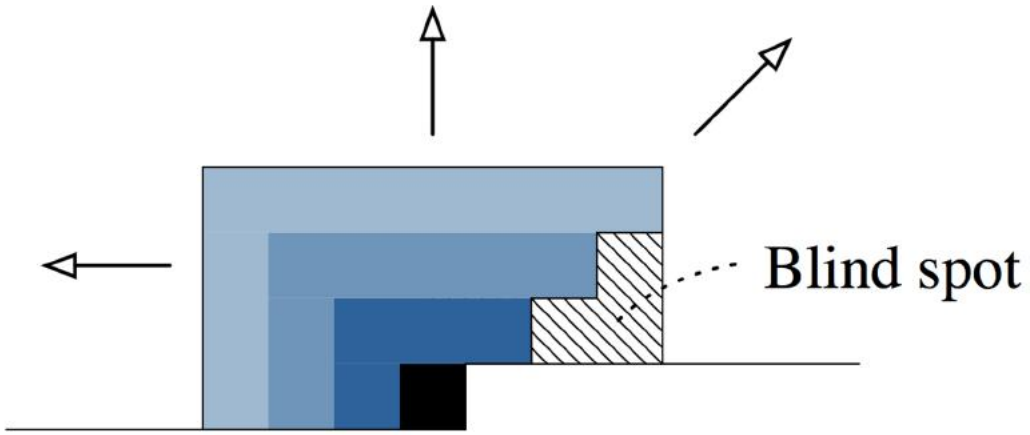


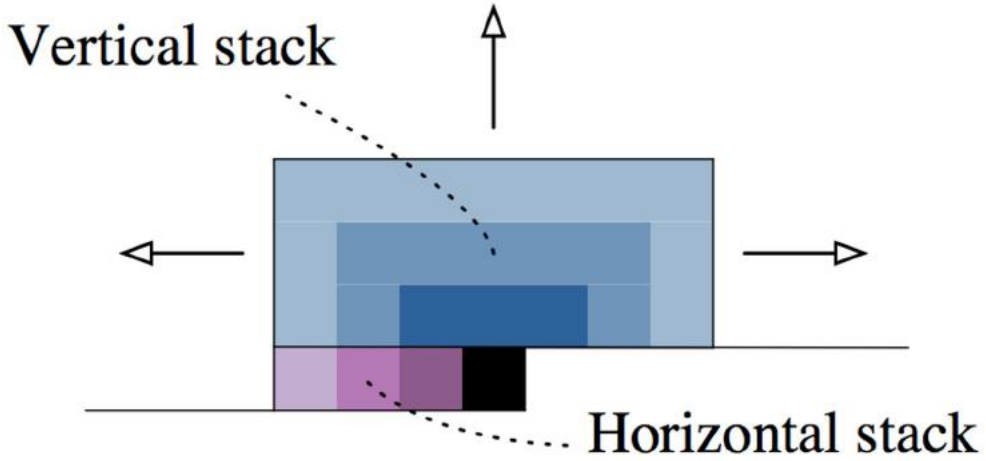
PixelCNN



Recap: PixelCNN

- PixelCNN-style masking has one problem: blind spot in receptive field





- Gated PixelCNN (2016) introduced a fix by combining two streams of convolutions

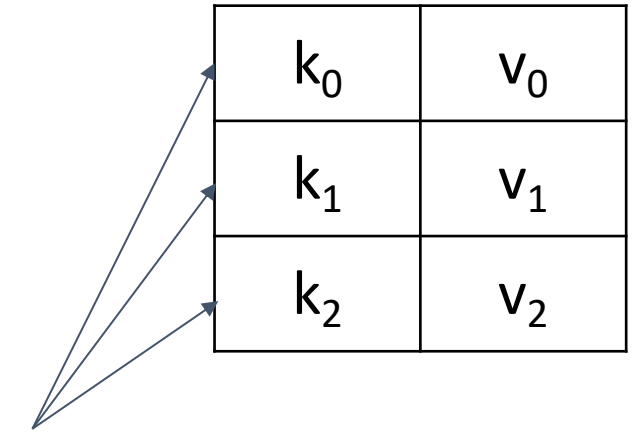
Lecture overview

- motivation
- 1-dimensional distributions
- high-dimensional distributions
- **deeper dive into causal masked neural models**
  - convolutional
  - **attention**
  - tokenization
  - caching
- other things to be aware of

Masked Attention

- A recurring problem for convolution: limited receptive field
  → hard to capture long-range dependencies
- (Self-)Attention: an alternative that has
  - unlimited receptive field!!
  - also O(1) parameter scaling w.r.t. data dimension
  - parallelized computation (versus RNN)

Scaled Dot-Product Attention



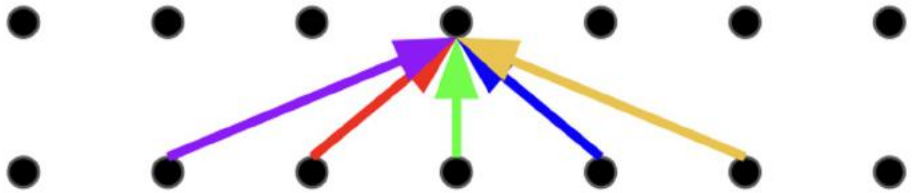
$$A(q, K, V) = \sum_i \frac{e^{s(q,k_i)}}{\sum_j e^{s(q,k_j)}} v_i$$

$$s(q, k) = \frac{q \cdot k}{\sqrt{d}}$$

Matrix Form: QKV are L x D

$$A(Q, K, V) = \text{softmax}(QK^T)V$$

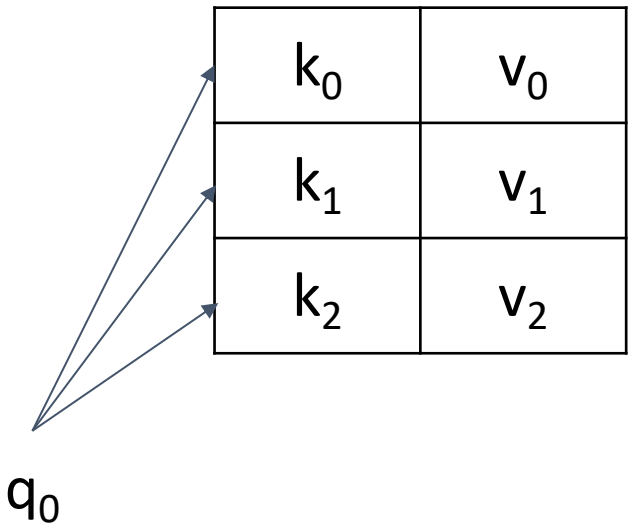
Self-Attention



Convolution

Self-attention

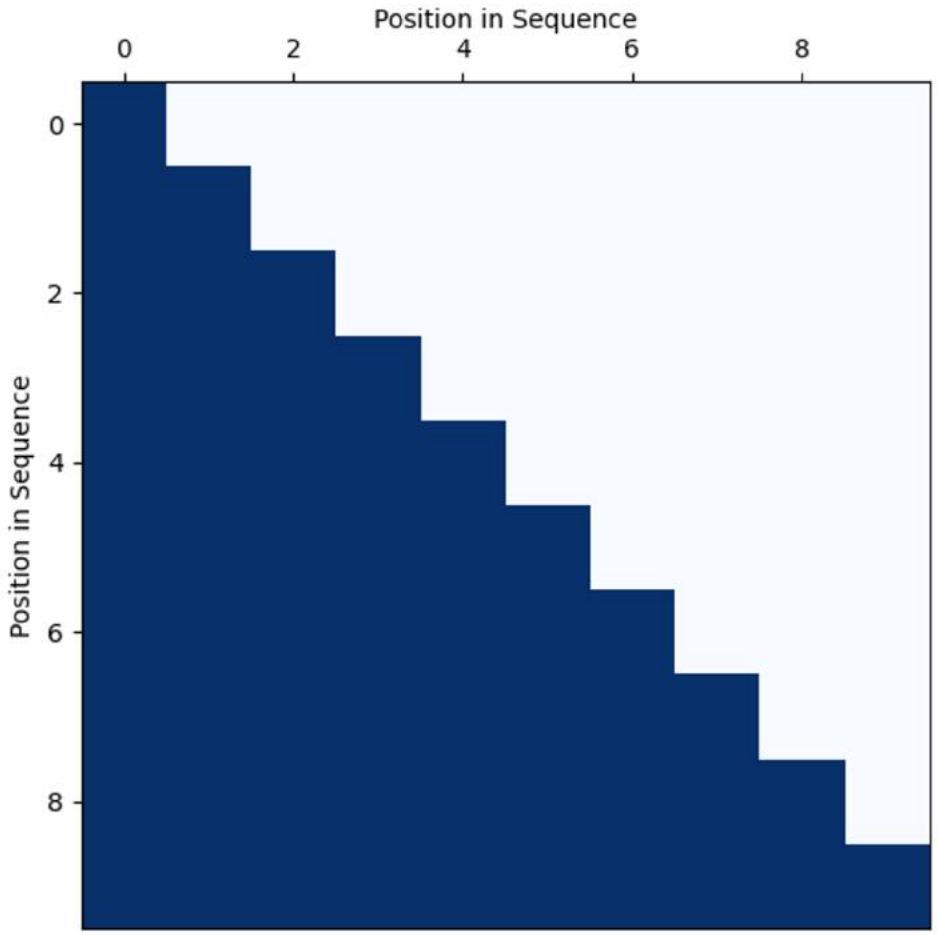
Self-Attention



$$A(Q, K, V) = \text{softmax}(QK^T)V$$

$$Q = XW_Q, K = XW_K, V = XW_V$$

Masked Attention



$$A(q, K, V) = \sum_i \frac{e^{s(q,k_i)}}{\sum_j e^{s(q,k_j)}} v_i$$

$$s(q, k) = \frac{q \cdot k}{\sqrt{d}} - (1 - \text{mask}(q, k)) * 10^{10}$$

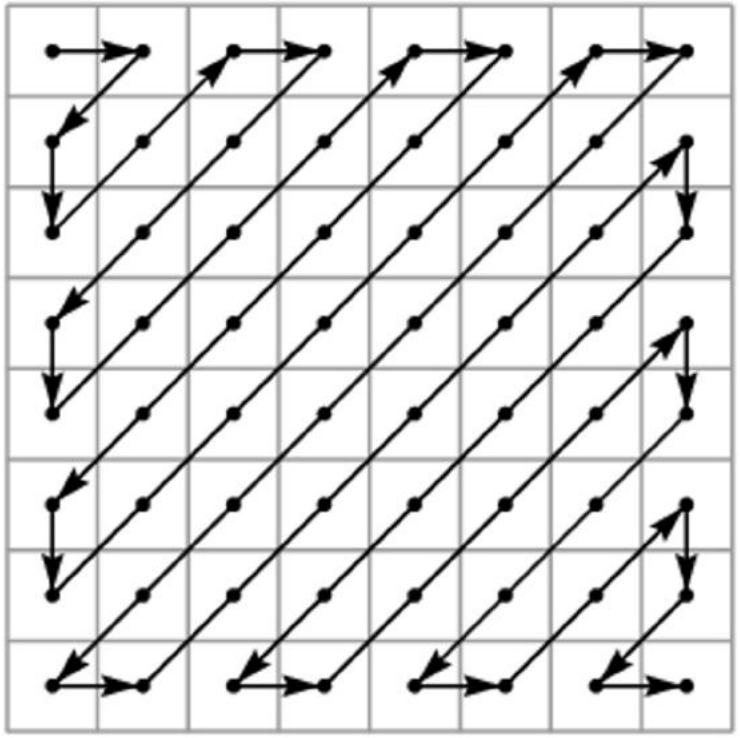
Matrix Form

$$A(Q, K, V) = \text{softmax}(QK^T - (1 - M) * 10^{10})$$

Masked Attention

- Much more flexible than masked convolution. We can design any autoregressive ordering we want

- An example:



Zigzag ordering

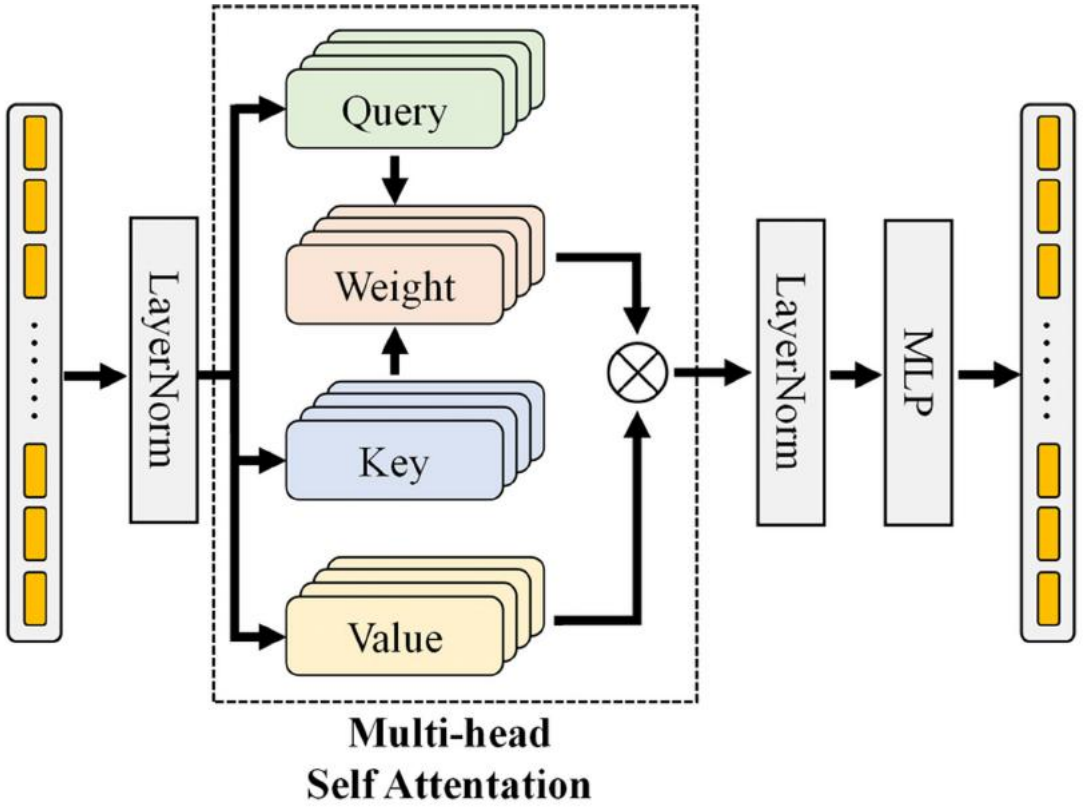
- How to implement with masked conv?
- Trivial to do with masked attention!

Transformers

- Transformer Block
  - Multi-head Self-Attention (MHSA)
  - MLP

$$h = h + \mathrm{MHSA}(\mathrm{LayerNorm}(\mathrm{h}))$$

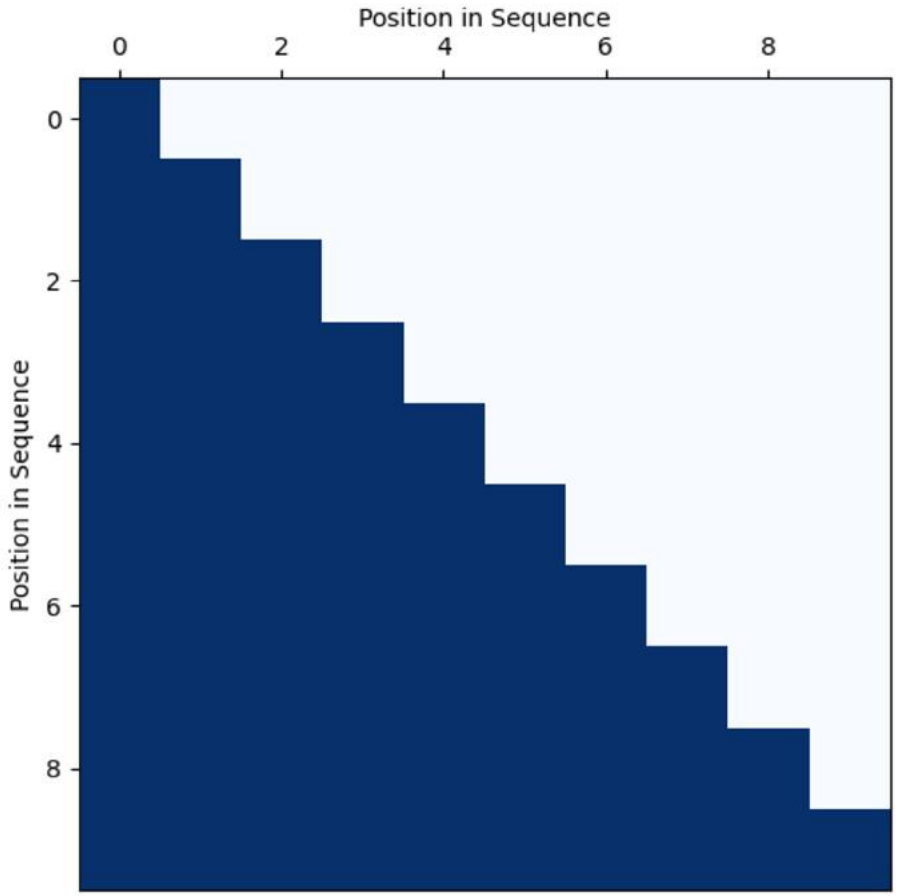
$$h = h + \mathrm{MLP}(\mathrm{LayerNorm}(\mathrm{h}))$$

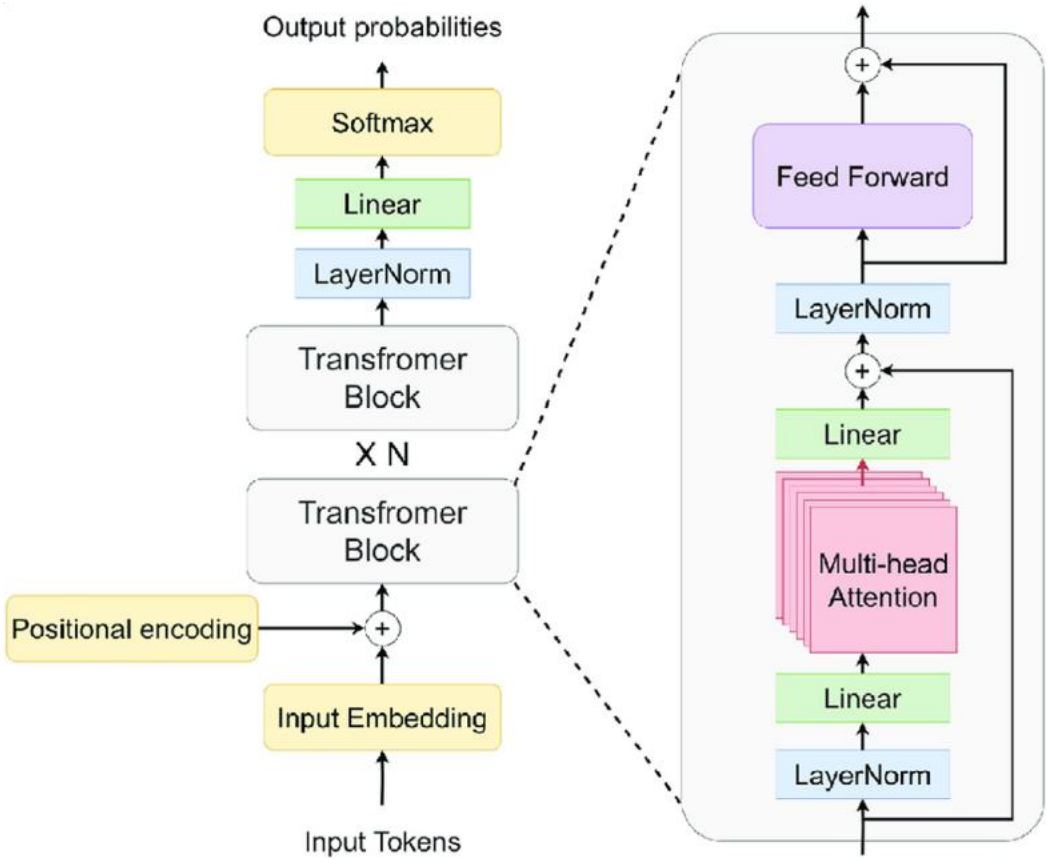


Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Autoregressive Transformers

- Standard transformer with a causal mask applied





Computational Scaling

Assuming sequence length L tokens and dimension D, FLOPs are:

- Multi-Head Attention: $8D^2L + 4DL^2 = O(D^2L + DL^2)$
  - **Quadratic in both D and L!**
- MLP: $16D^2L = O(D^2L)$

Lecture overview

- motivation
- 1-dimensional distributions
- high-dimensional distributions
- **deeper dive into causal masked neural models**
  - convolutional
  - attention
  - **tokenization**
  - caching
- other things to be aware of

Tokens

- Autoregressive transformers are general-purpose, modality-agnostic models

- Can model any complex distribution / modality as long as you can tokenize (discretize) the data

Tokens for Text

- **Characters:** a=0, b=1, c=2, …
  - Small vocabulary
  - Large number of tokens (recall quadratic scaling for attention…)

- **Words**: car=0, apple=1, dog=2, …
  - Large vocabulary
  - Small number of tokens
  - What happens if we introduce a new word?

Byte-Pair Encoding (BPE)

Can we consider something in-between?

- Tokenize based on groupings of characters, prioritizing by frequency

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units." arXiv preprint arXiv:1508.07909 (2015).

Byte-Pair Encoding (BPE)

Sub-word tokenizers: (a=0, umbr=1, ella=2, …)

- Middle-ground between number of tokens, and codebook size
- 1-2 tokens per word
- Generalizes to novel combinations of characters

GPT-1

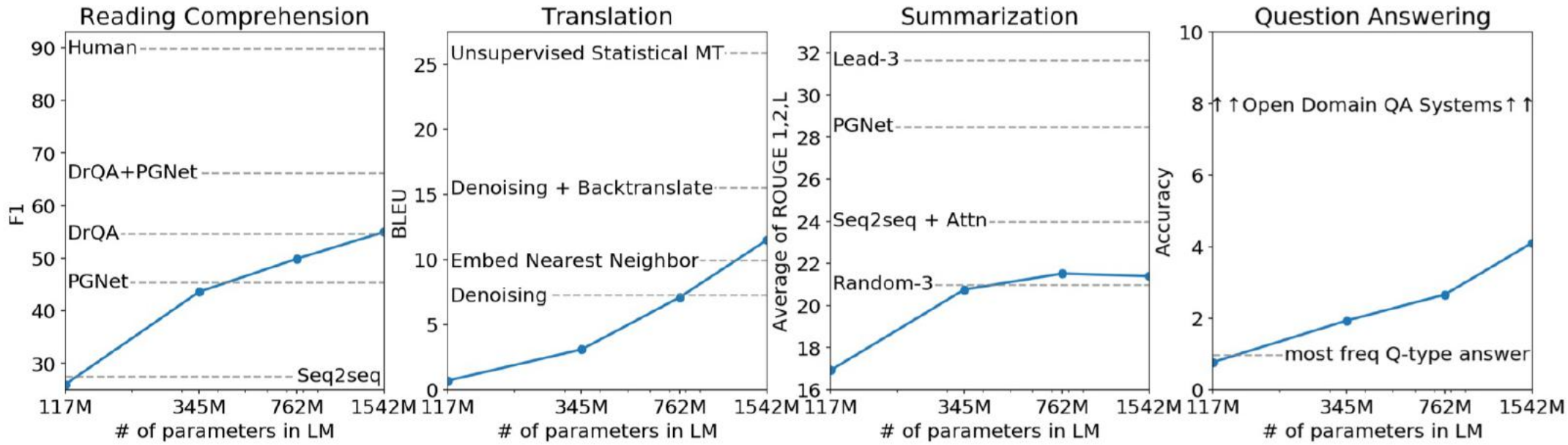
- Unsupervised pre-training followed by supervised fine-tuning
- 100M parameter transformer model
- Finetuning a model pretrained on general language outperforms models designed for each task

Method	Story Cloze	RACE-m	RACE-h	RACE
val-LS-skip [55]	76.5	-	-	-
Hidden Coherence Model [7]	77.6	-	-	-
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2
BiAttention MRU [59] (9x)	-	60.2	50.3	53.3
Finetuned Transformer LM (ours)	**86.5**	**62.9**	**57.4**	**59.0**

Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018).

GPT-2

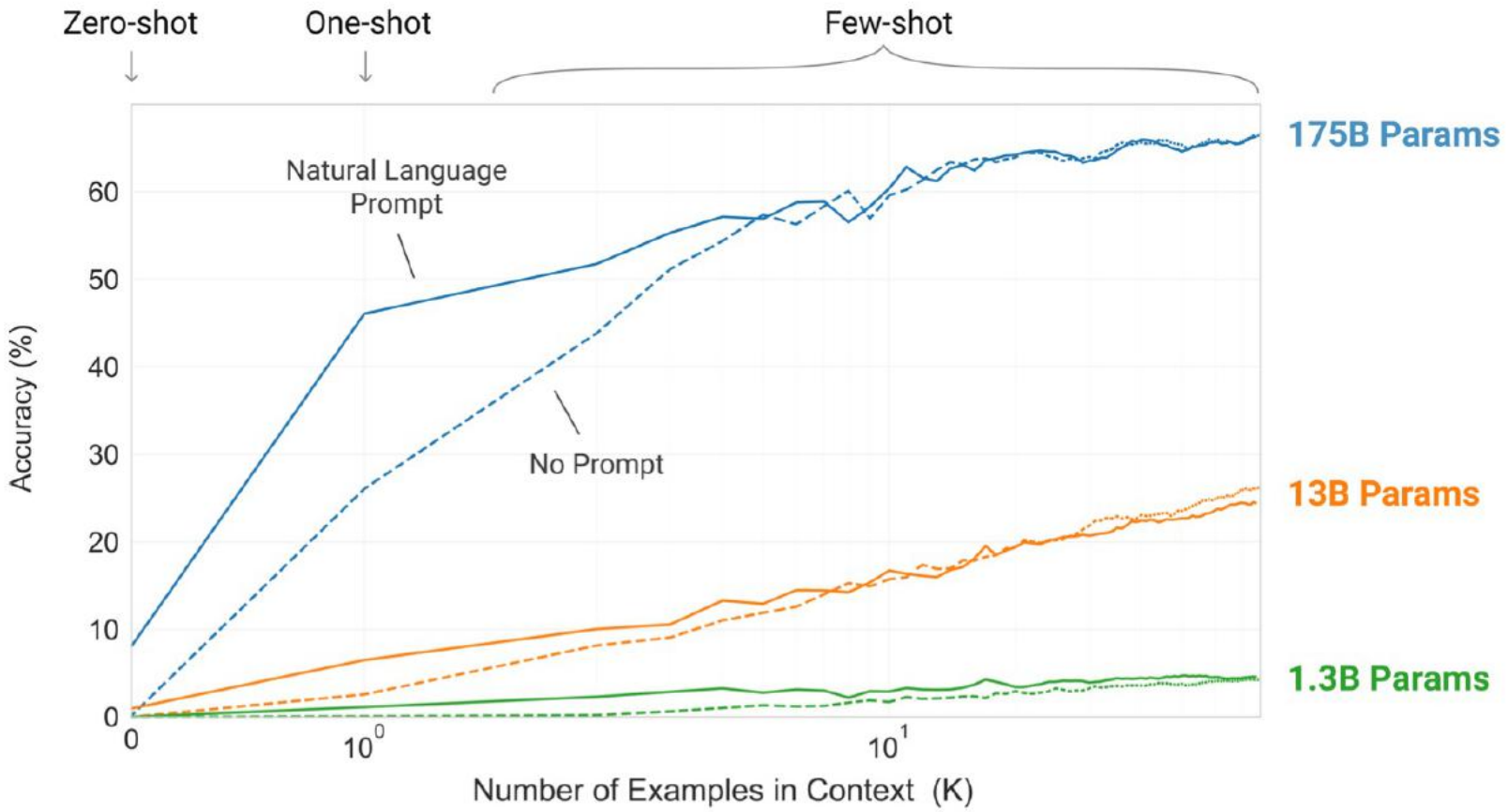
- Scaled data and model size to 1.5B parameters
- Improvements with scale even without supervised finetuning (zero-shot evaluation on downstream tasks)



Radford, Alec, et al. "Language models are unsupervised multitask learners." OpenAI blog 1.8 (2019): 9.

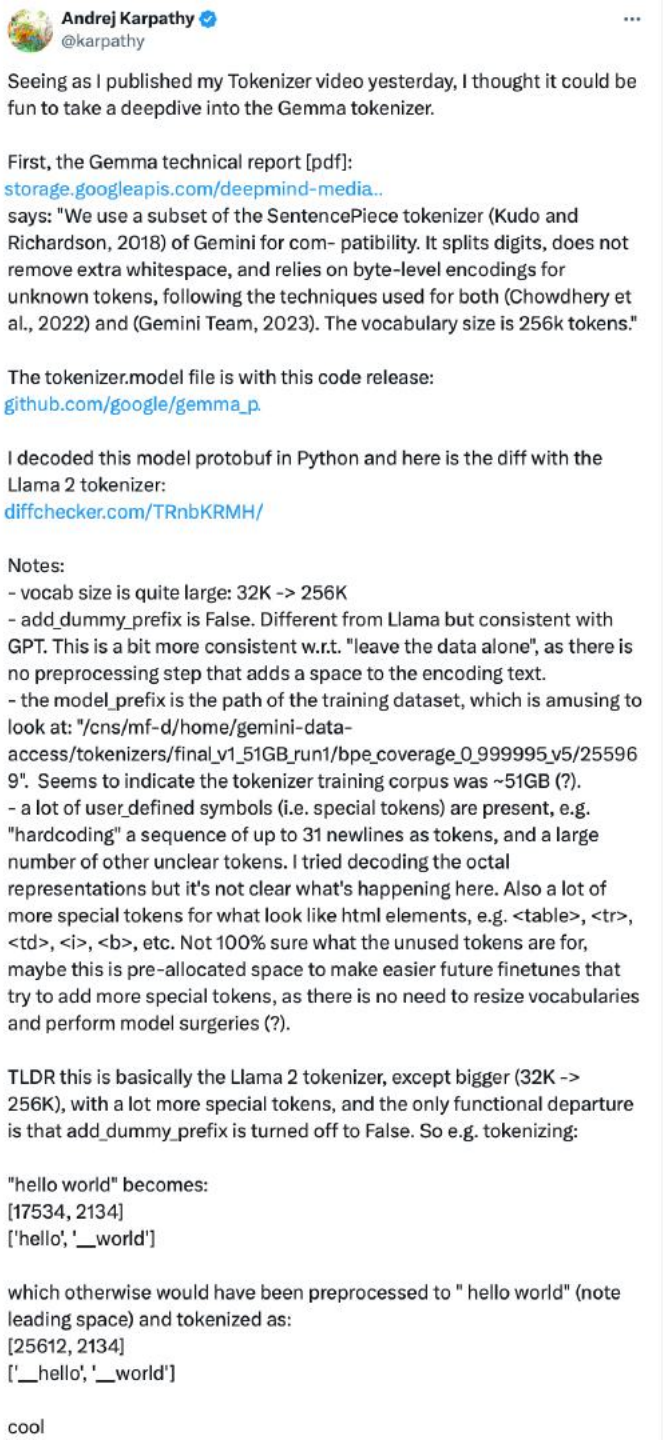
GPT-3

- In-context learning abilities emerge as number of parameters grow



Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.

Gemma



Andrej Karpathy
@karpathy

Seeing as I published my Tokenizer video yesterday, I thought it could be fun to take a deepdive into the Gemma tokenizer.

First, the Gemma technical report [pdf]:
storage.googleapis.com/deepmind-media..
says: "We use a subset of the SentencePiece tokenizer (Kudo and Richardson, 2018) of Gemini for com- patibility. It splits digits, does not remove extra whitespace, and relies on byte-level encodings for unknown tokens, following the techniques used for both (Chowdhery et al., 2022) and (Gemini Team, 2023). The vocabulary size is 256k tokens."

The tokenizer.model file is with this code release:
github.com/google/gemma_p.

I decoded this model protobuf in Python and here is the diff with the Llama 2 tokenizer:
diffchecker.com/TRnbKRMH/

Notes:
- vocab size is quite large: 32K -> 256K
- add_dummy_prefix is False. Different from Llama but consistent with GPT. This is a bit more consistent w.r.t. "leave the data alone", as there is no preprocessing step that adds a space to the encoding text.
- the model_prefix is the path of the training dataset, which is amusing to look at: "/cns/mf-d/home/gemini-data-access/tokenizers/final_v1_51GB_run1/bpe_coverage_0_999995_v5/255969". Seems to indicate the tokenizer training corpus was ~51GB (?).
- a lot of user_defined symbols (i.e. special tokens) are present, e.g. "hardcoding" a sequence of up to 31 newlines as tokens, and a large number of other unclear tokens. I tried decoding the octal representations but it's not clear what's happening here. Also a lot of more special tokens for what look like html elements, e.g. <table>, <tr>, <td>, <i>, <b>, etc. Not 100% sure what the unused tokens are for, maybe this is pre-allocated space to make easier future finetunes that try to add more special tokens, as there is no need to resize vocabularies and perform model surgeries (?).

TLDR this is basically the Llama 2 tokenizer, except bigger (32K -> 256K), with a lot more special tokens, and the only functional departure is that add_dummy_prefix is turned off to False. So e.g. tokenizing:

"hello world" becomes:
[17534, 2134]
['hello', '__world']

which otherwise would have been preprocessed to " hello world" (note leading space) and tokenized as:
[25612, 2134]
['__hello', '__world']

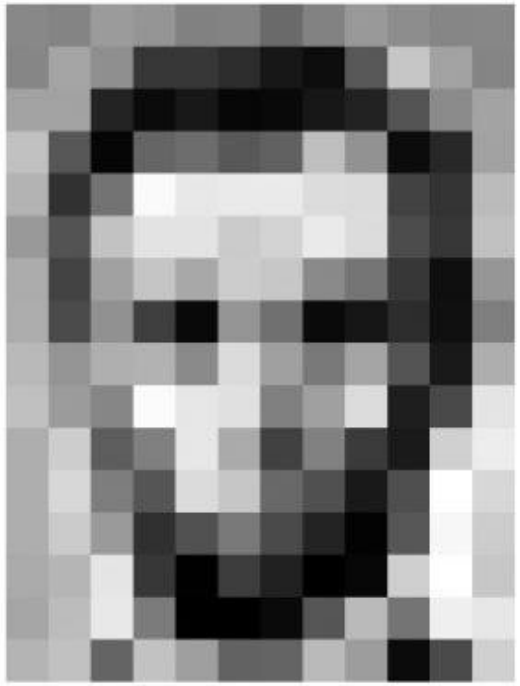
cool

https://twitter.com/karpathy/status/1760350892317098371

Tokens for Images

Consider encoding RGB (H x W x 3) images as tokens

- Raw Pixels: each pixel for each color channel is generally stored as a single byte (0-255)



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

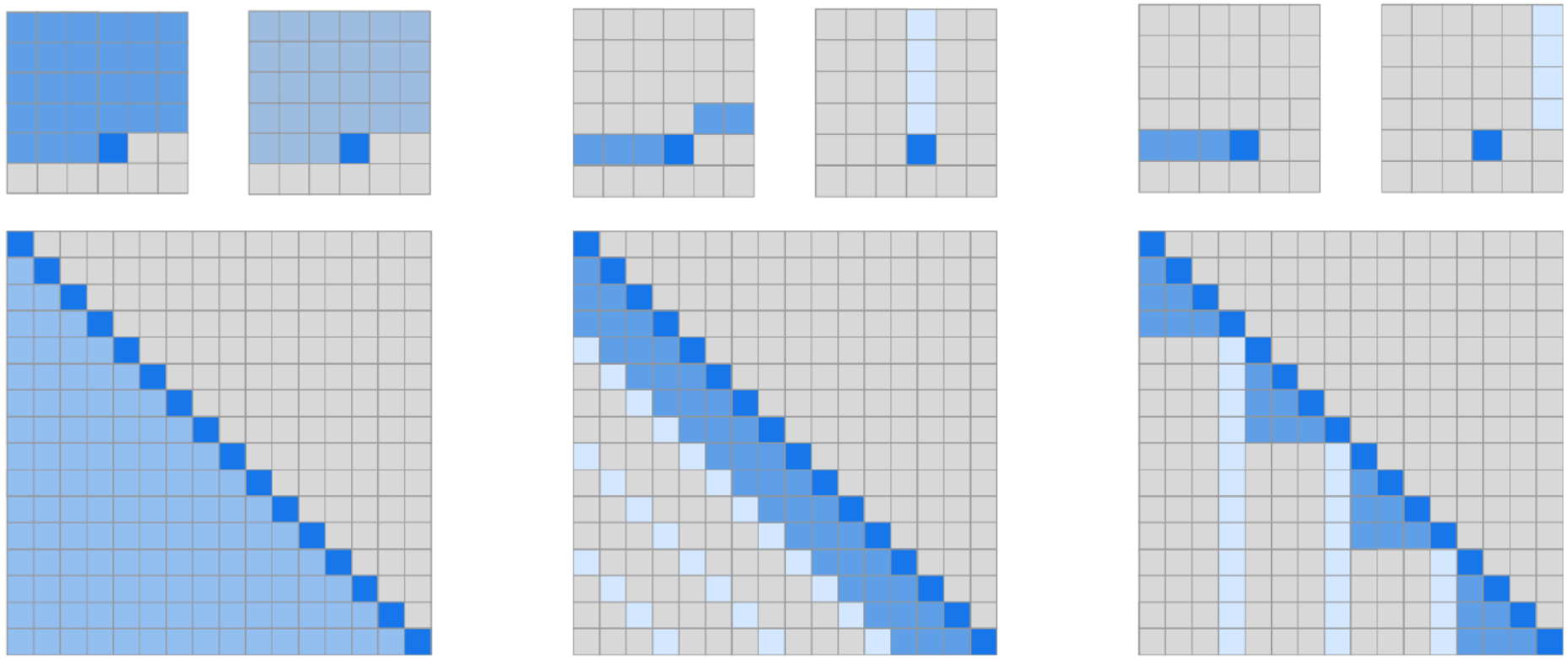
Raw Pixels as Tokens

- 64 x 64 x 3 = 12k tokens
- 256 x 256 x 3 = 196k tokens!

Quadratic scaling of attention makes this prohibitively expensive

Sparse Transformer

- Train a causal transformer, but with special hard-coded masking



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Child, Rewon, et al. "Generating long sequences with sparse transformers." arXiv preprint arXiv:1904.10509 (2019).
Parmar, Niki, et al. "Image transformer." International conference on machine learning. PMLR, 2018.

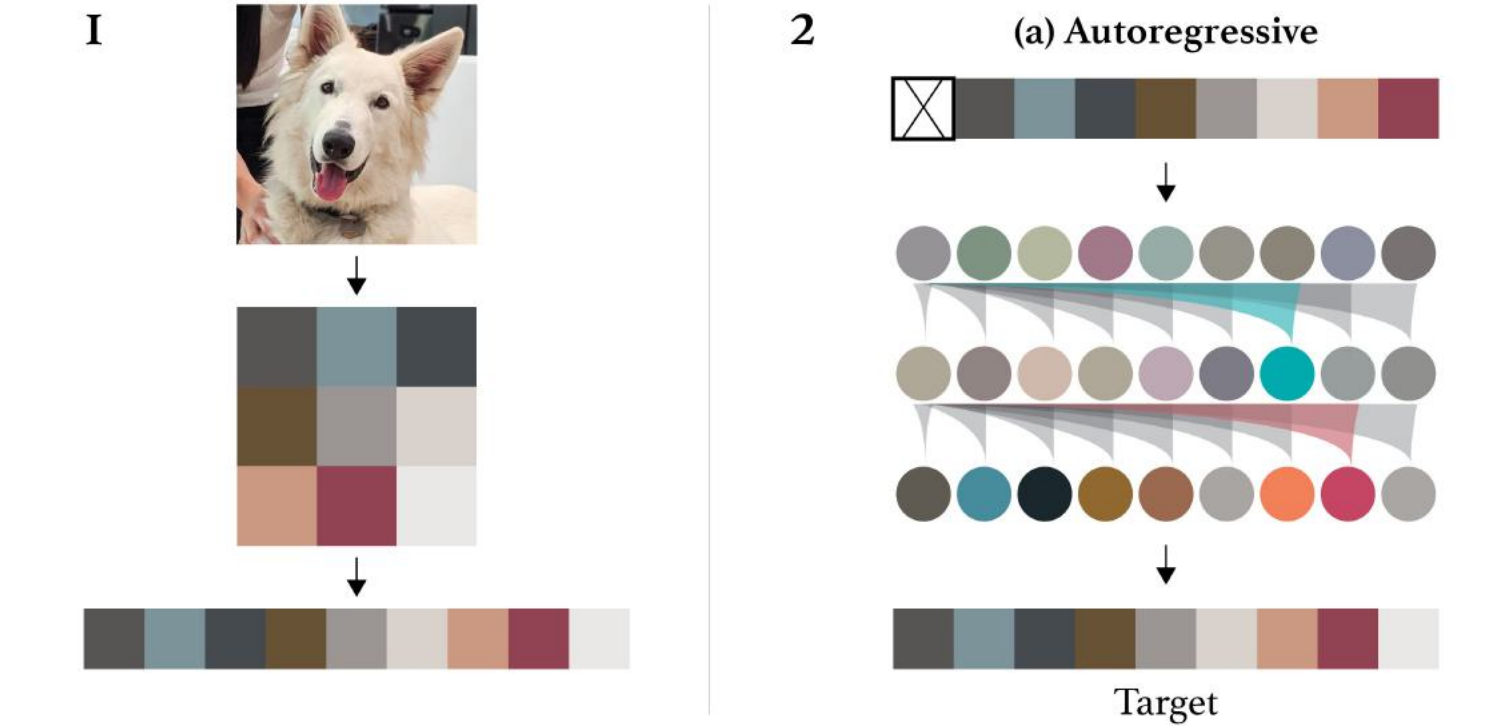
Sparse Transformer

- SOTA results on perplexity, but still lacking in sample quality

Child, Rewon, et al. "Generating long sequences with sparse transformers." arXiv preprint arXiv:1904.10509 (2019).

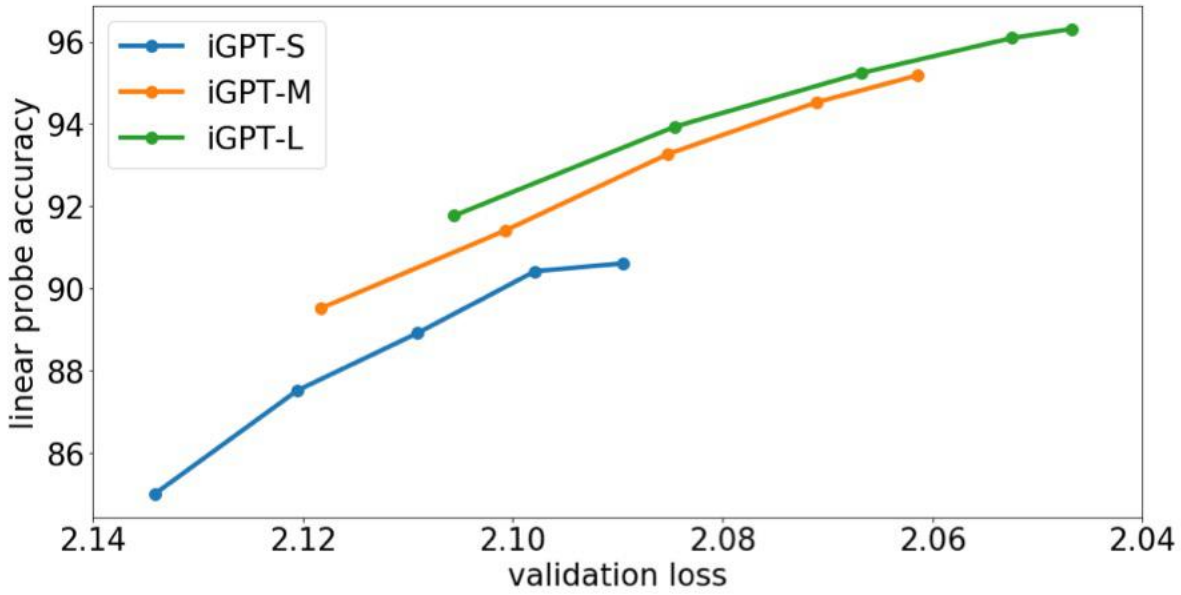
iGPT

- Created a custom 9-bit color palette by clustering RGB pixel values with k-means clustering
  - 3x reduction in sequence length from the original 24-bit color palette



iGPT

Showed strong scaling results on representation learning with generation models compared to contemporary models



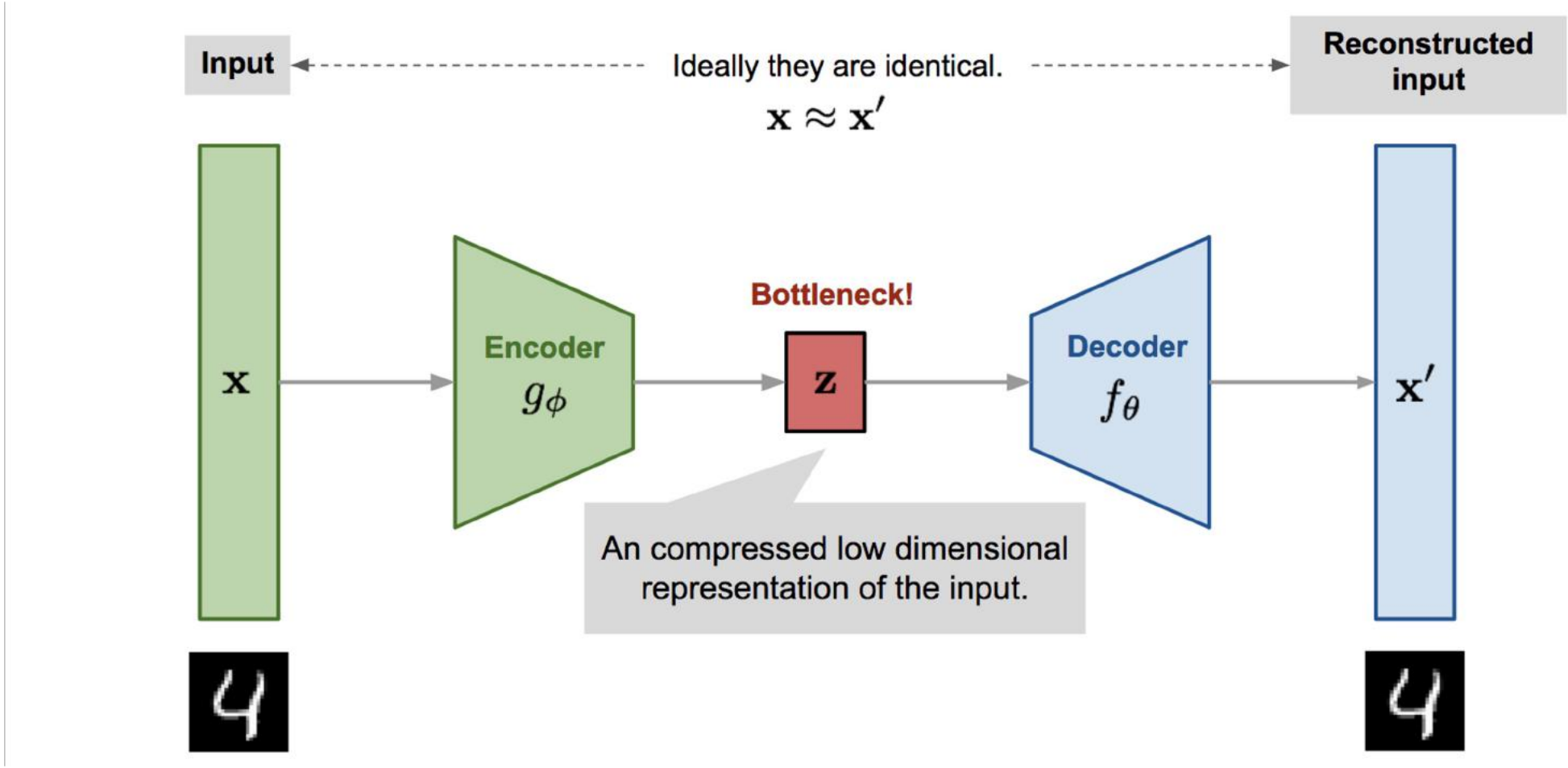
Model	Acc	Unsup Transfer	Sup Transfer
CIFAR-10			
ResNet-152	94		√
SimCLR	95.3	√	
iGPT-L	96.3	√	
CIFAR-100			
ResNet-152	78.0		√
SimCLR	80.2	√	
iGPT-L	82.8	√	
STL-10			
AMDIM-L	94.2	√	
iGPT-L	95.5	√	

Tokens for Images

- But iGPT still needed to train on image as small as 32 x 32
- Can we get better tokens for images?

Discrete Autoencoders

- Learn a discrete autoencoder!



Discrete Autoencoders

Discretization methods:

- Gumbel-Softmax / Concrete Distribution: GS, CD
- Vector-Quantization (VQ):
- Finite Scalar Quantization (FSQ) / Lookup-Free Quantization (LFQ): FSQ, LFQ

Discrete Autoencoders

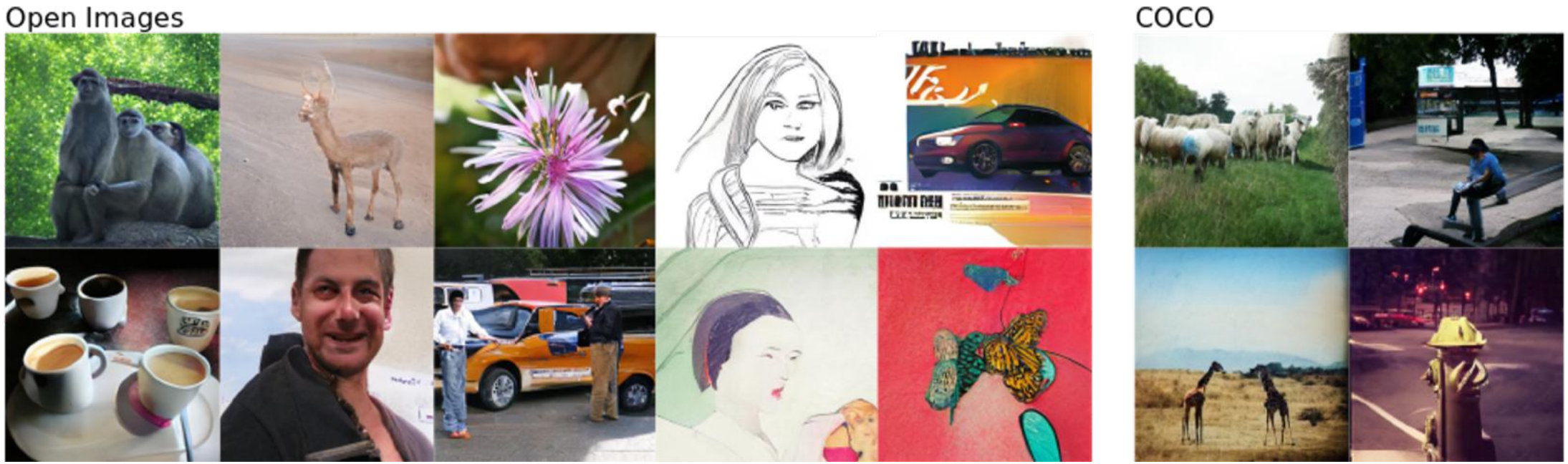
This method of tokenization is lossy

- 256 x 256 x 3 -> 16 x 16
  - 384x compression in bytes from raw pixels
  - 768x reduction in sequence length

VQGAN Transformer

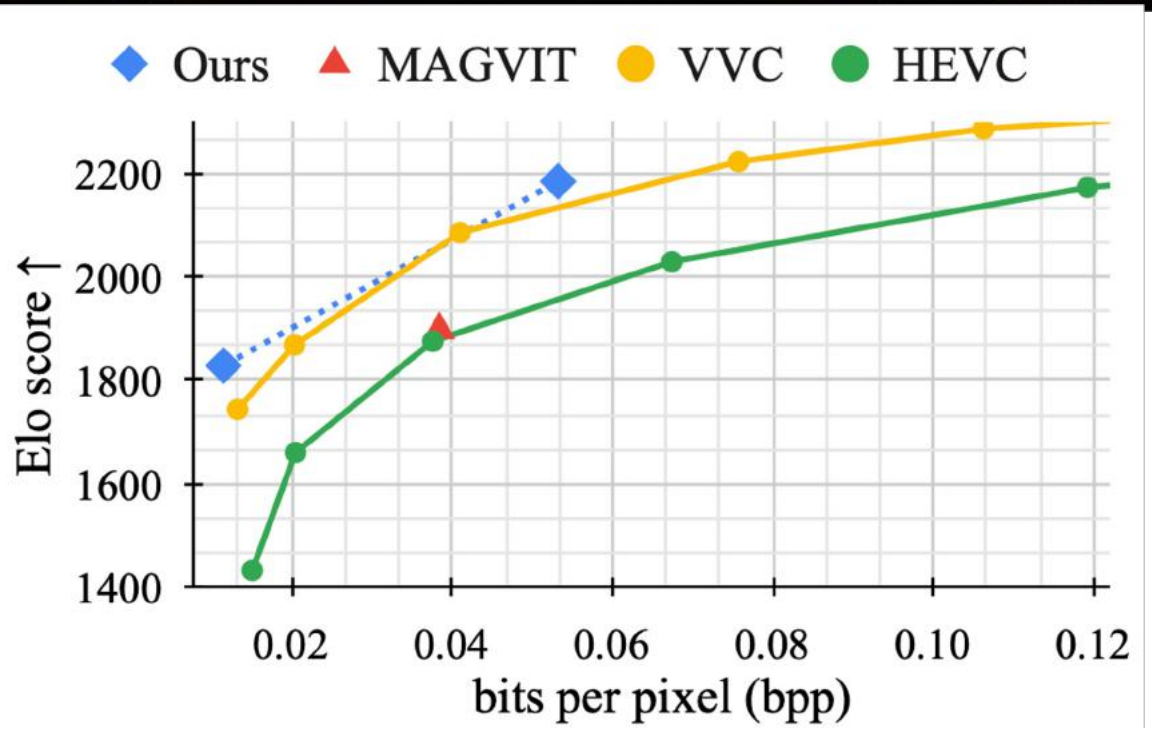
Train an autoregressive transformer to model discrete encodings

- Much more tractable than from pixels. Samples below



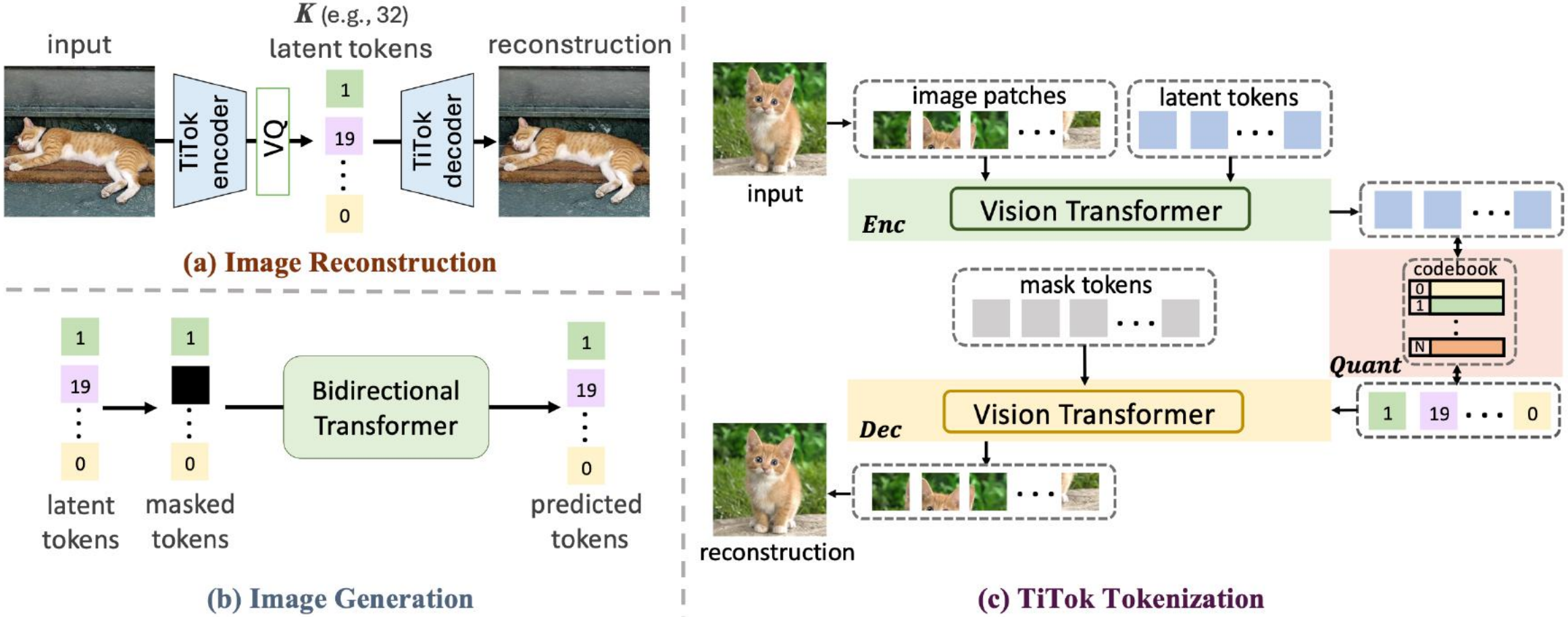
Esser, Patrick, Robin Rombach, and Bjorn Ommer. "Taming transformers for high-resolution image synthesis." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021.

Discrete Encodings for Video



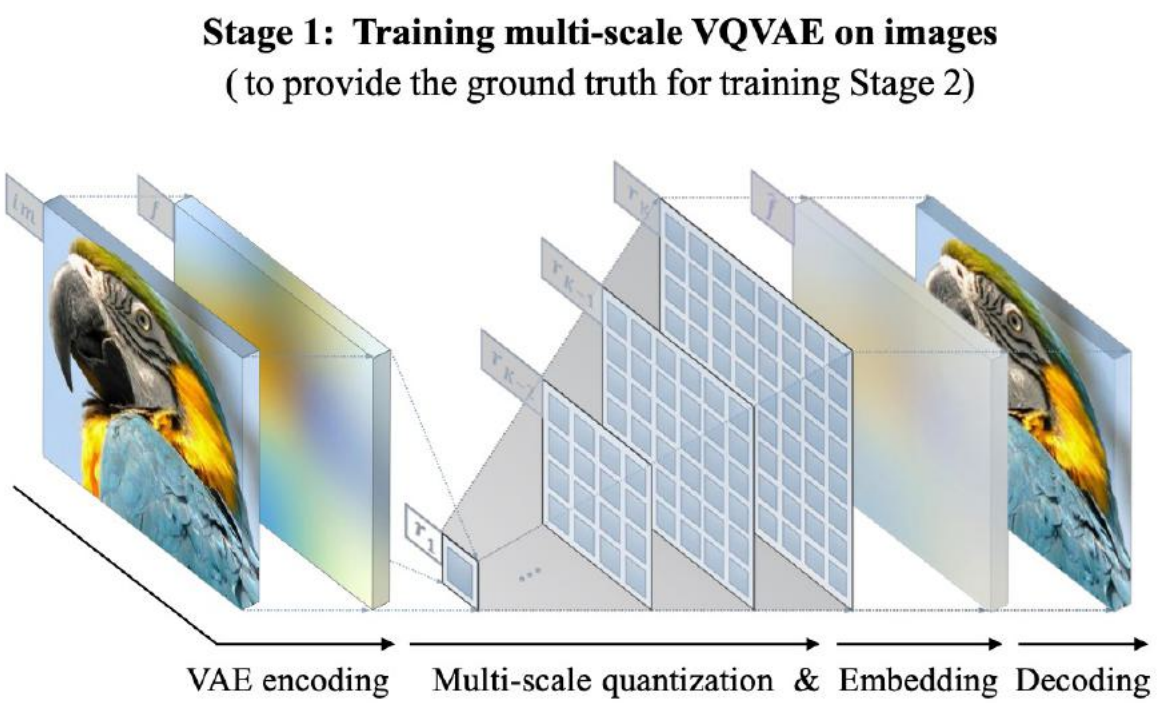
Lijun Yu, José Lezama, Nitesh B. Gundavarapu et al. Language Model Beats Diffusion – Tokenizer is Key to Visual Generation. ICLR 2024.

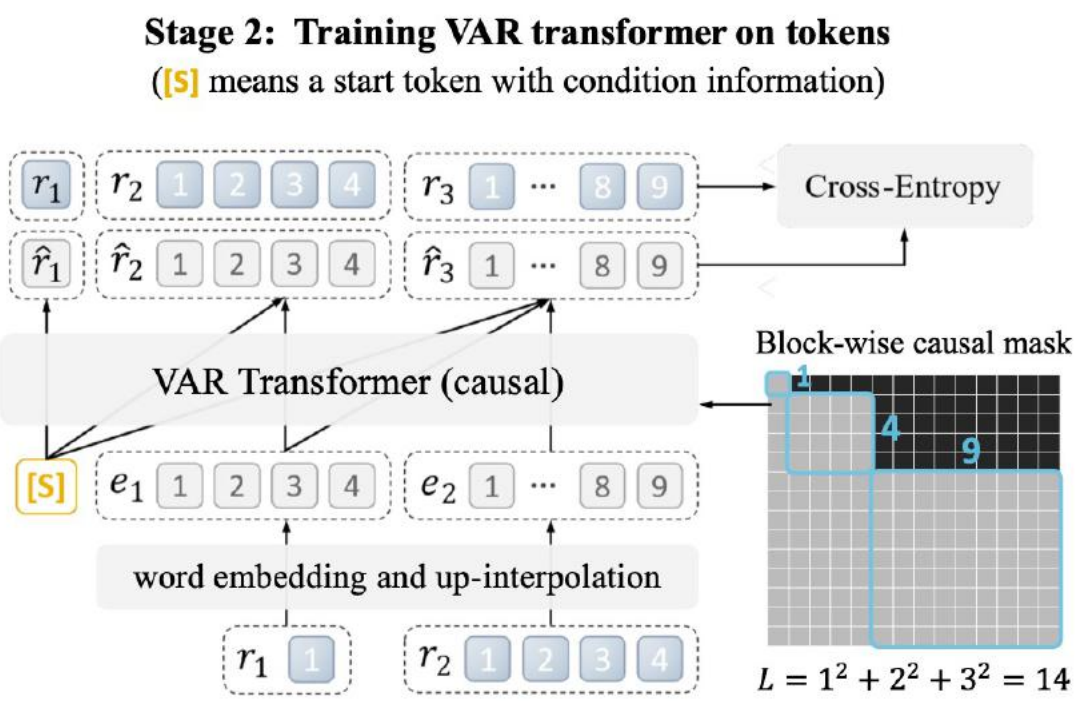
1D Tokenizers for 2D Image Data



Qihang Yu, Mark Weber, Xueqing Deng, Xiaohui Shen, Daniel Cremers, Liang-Chieh Chen. An Image is Worth 32 Tokens for Reconstruction and Generation. NeurIPS 2024.

Multi-Scale Tokenizers





Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, Liwei Wang. Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction. NeurIPS 2024.

VideoPoet

The orient express driving through a fantasy landscape, animated oil on canvas

A golden retriever wearing VR goggles and eating pizza in Paris.

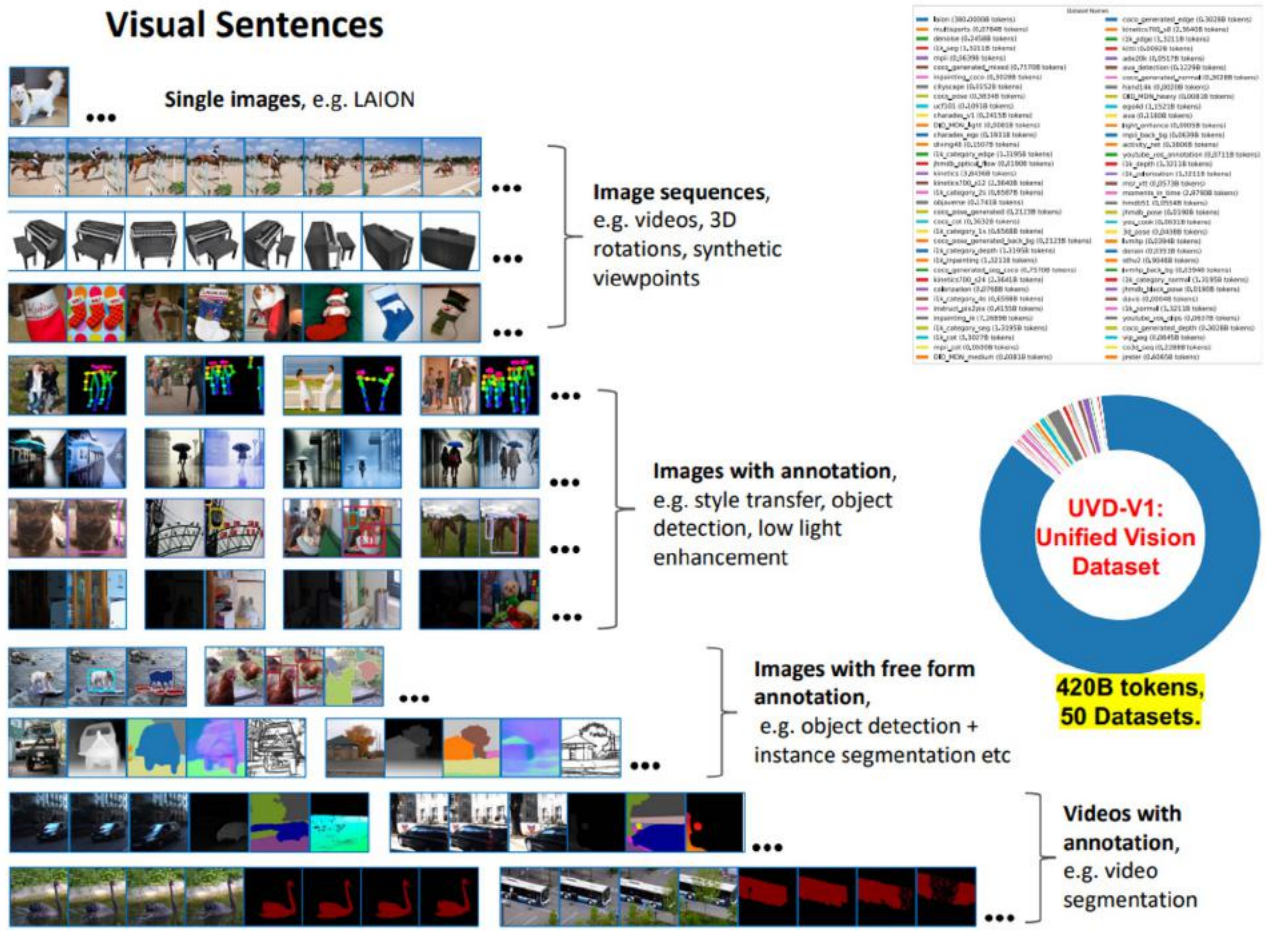
A chicken lifting weights

A shark with a laser beam coming out of its mouth

Kondratyuk, Dan, et al. "Videopoet: A large language model for zero-shot video generation." arXiv preprint arXiv:2312.14125 (2023).

In-Context Learning for Vision

- Train a large autoregressive transformer on sequential visual data



Bai, Yutong, et al. "Sequential modeling enables scalable learning for large vision models." arXiv preprint arXiv:2312.00785 (2023).

In-Context Learning for Vision

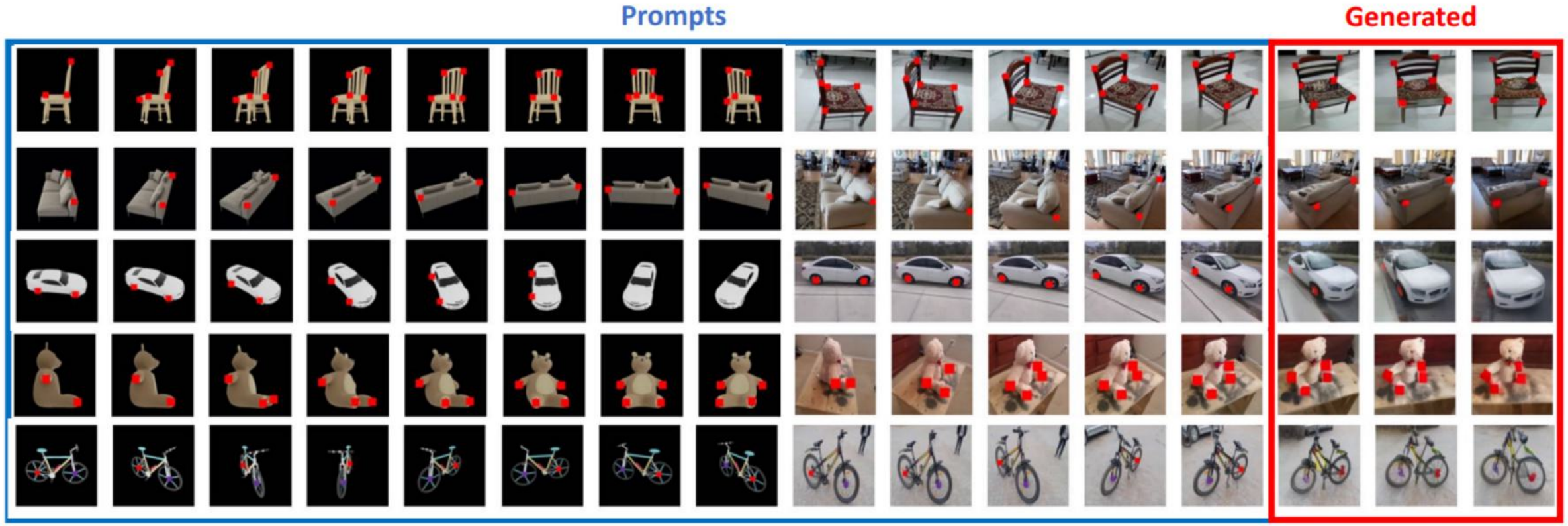
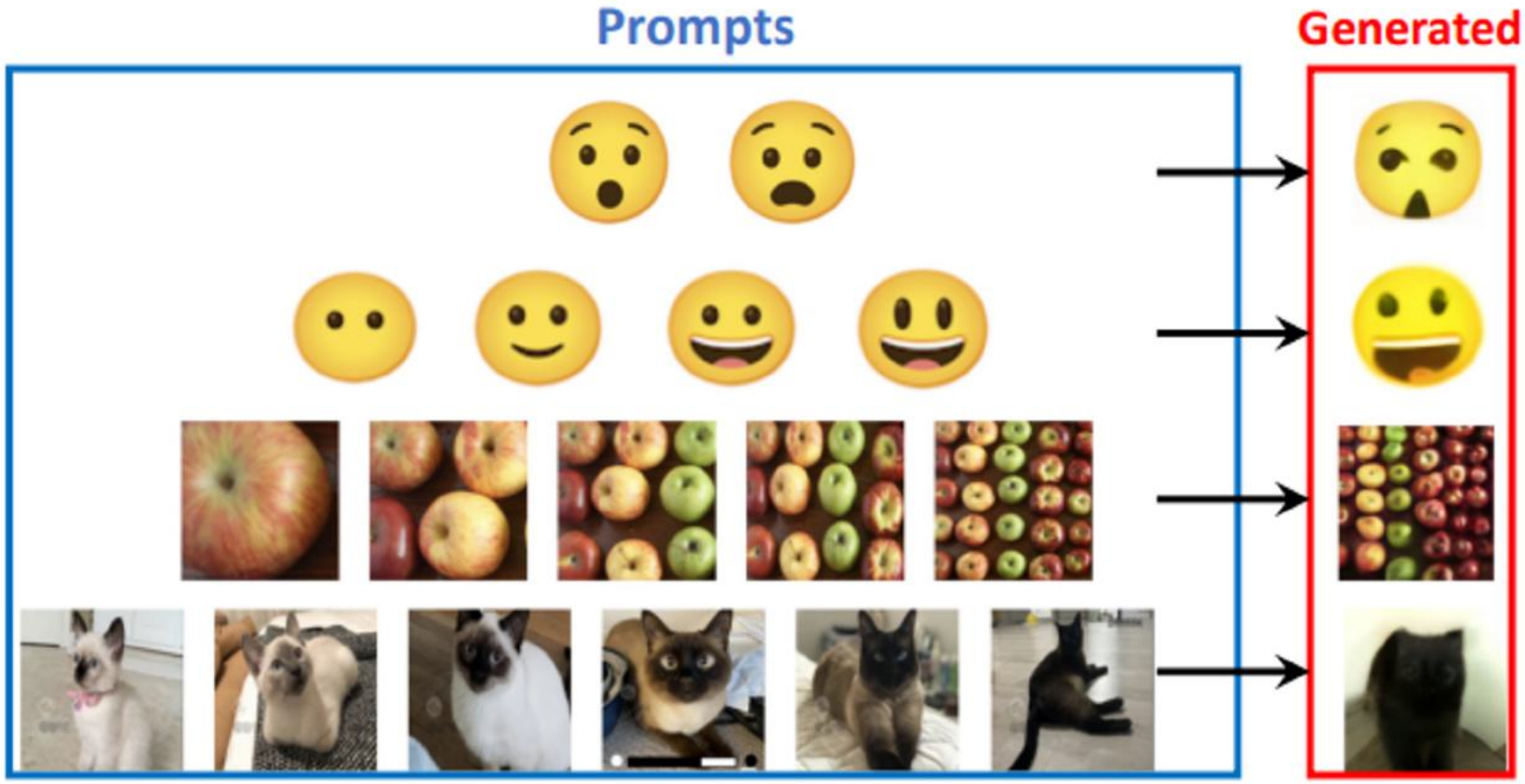
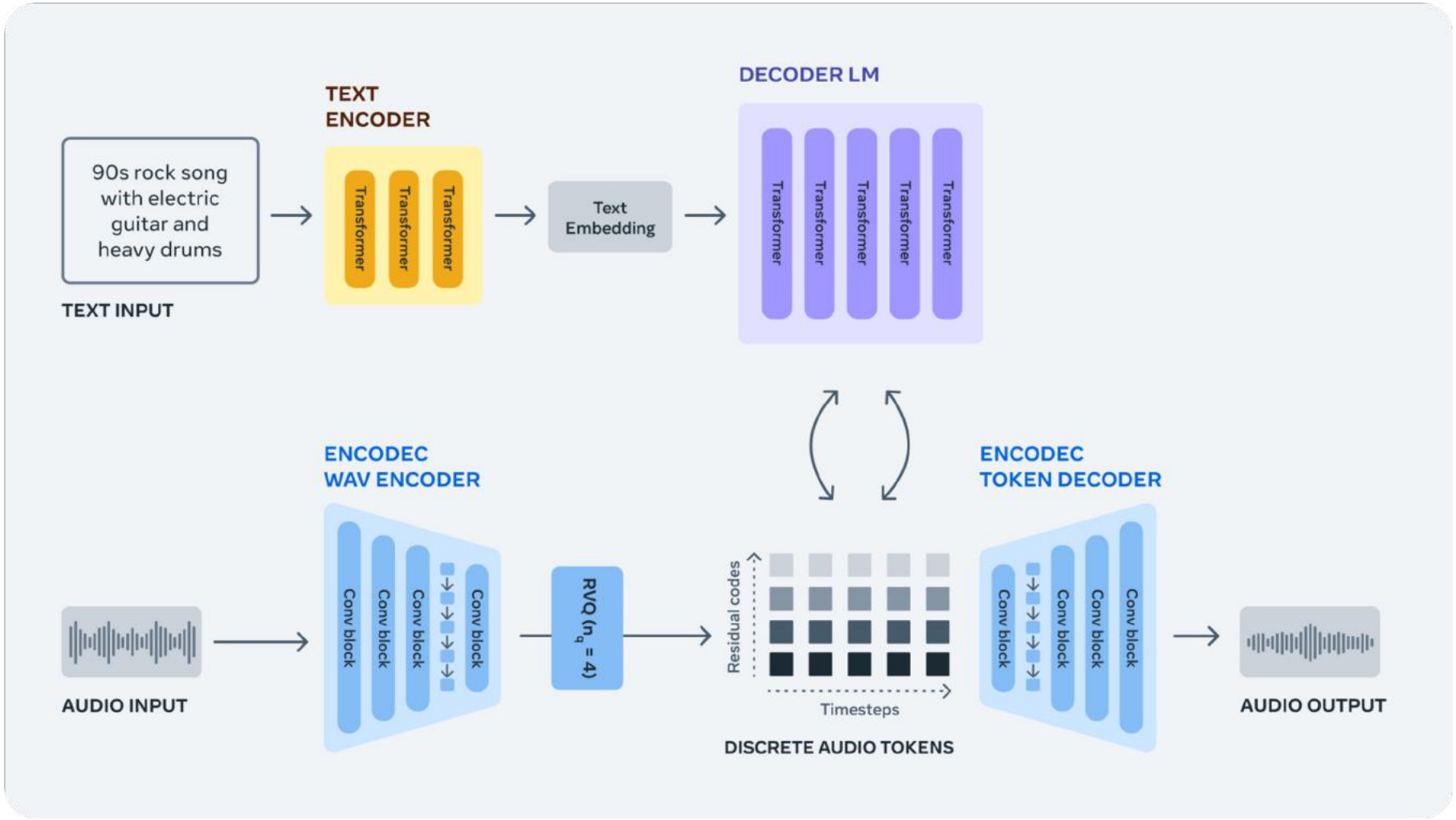


Figure 9. **Task Compositionality**. Examples of prompts that combine two different tasks – object rotation and keypoint tracking.

In-Context Learning for Vision

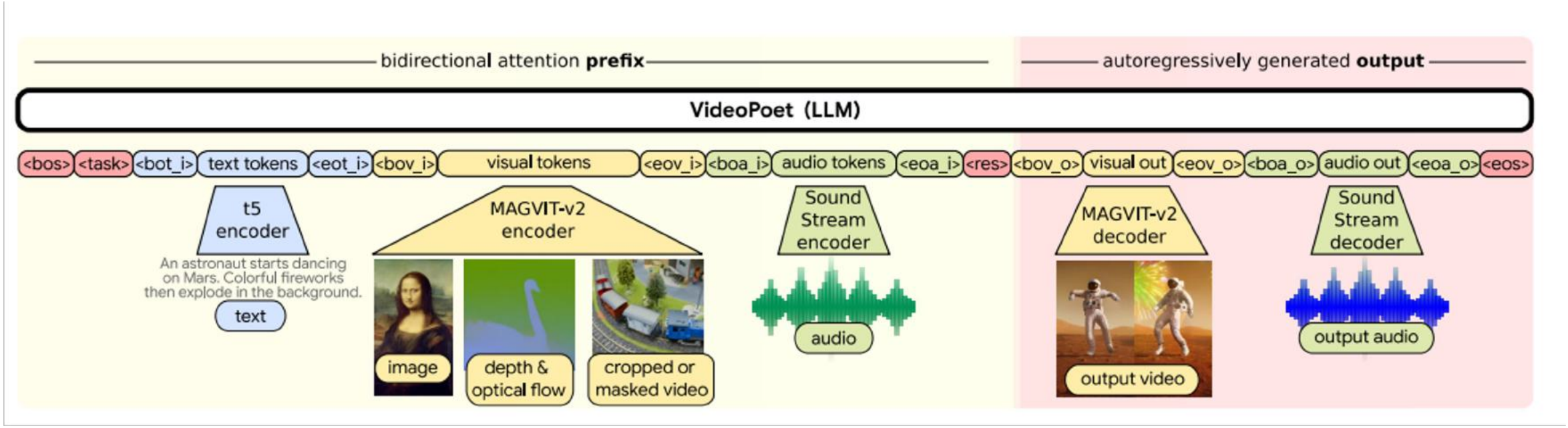


AudioCraft


90s rock song with electric guitar and heavy drums
TEXT INPUT
TEXT ENCODER
Transformer
Transformer
Transformer
Text Embedding
DECODER LM
Transformer
Transformer
Transformer
Transformer
Transformer
AUDIO INPUT
ENCODEC WAV ENCODER
Conv block
Conv block
Conv block
Conv block
RVQ (n_q = 4)
Residual codes
Timesteps
DISCRETE AUDIO TOKENS
ENCODEC TOKEN DECODER
Conv block
Conv block
Conv block
Conv block
AUDIO OUTPUT

Multi-Modal Models

- The general nature of transformers allows you to easily combine different modalities



Lecture overview

- motivation
- 1-dimensional distributions
- high-dimensional distributions
- **deeper dive into causal masked neural models**
  - convolutional
  - attention
  - tokenization
  - **caching**
- other things to be aware of

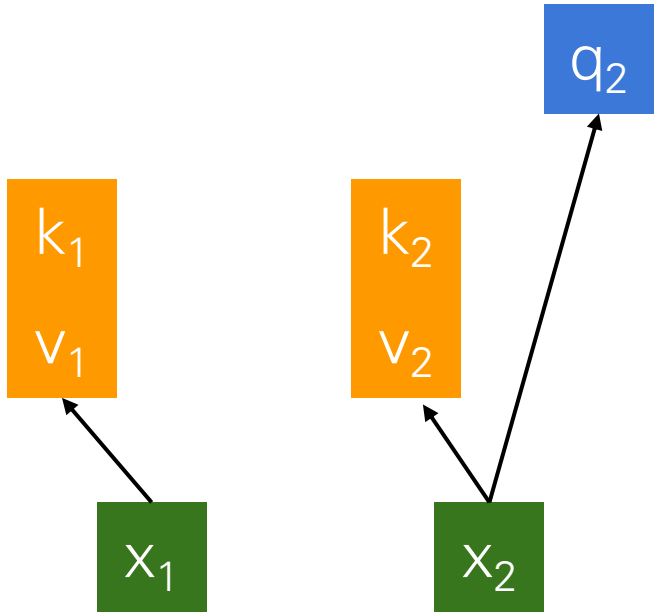
Caching / Efficient Sampling

Naive Sampling

x_1 x_2

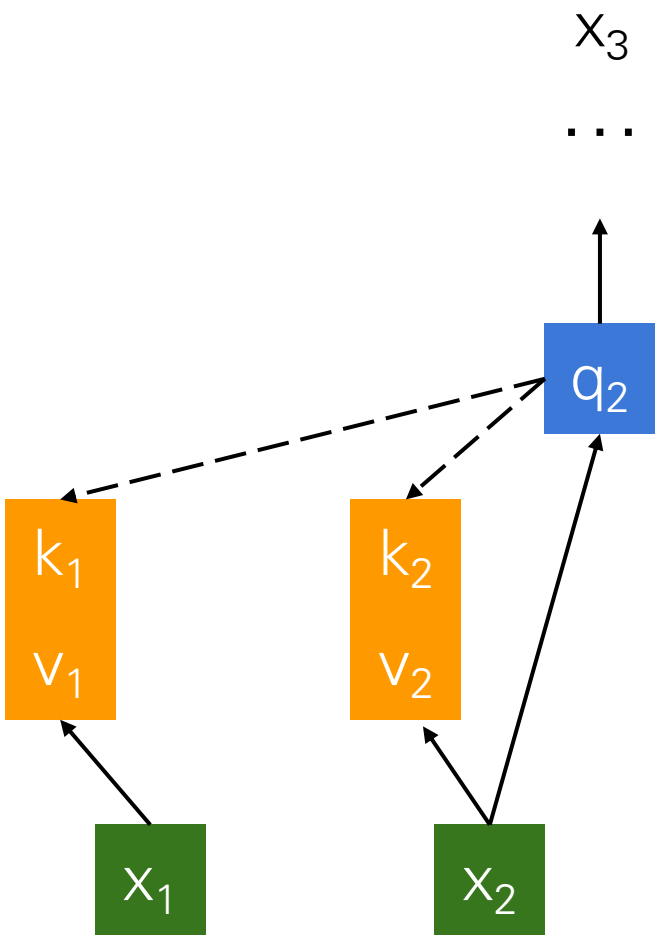
Caching / Efficient Sampling

Naive Sampling



Caching / Efficient Sampling

Naive Sampling



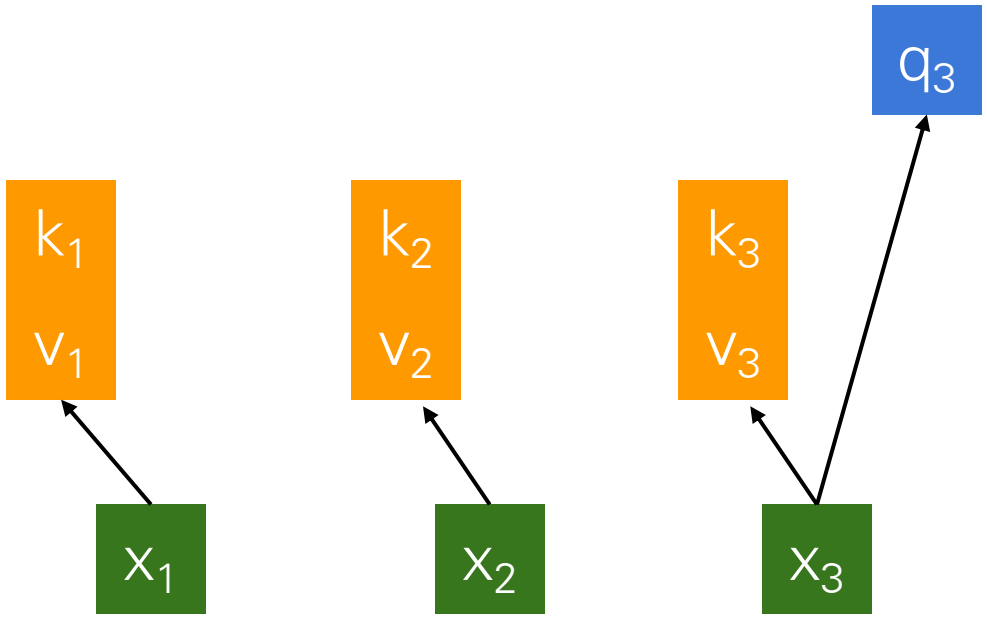
Caching / Efficient Sampling

Naive Sampling

x_1 x_2 x_3

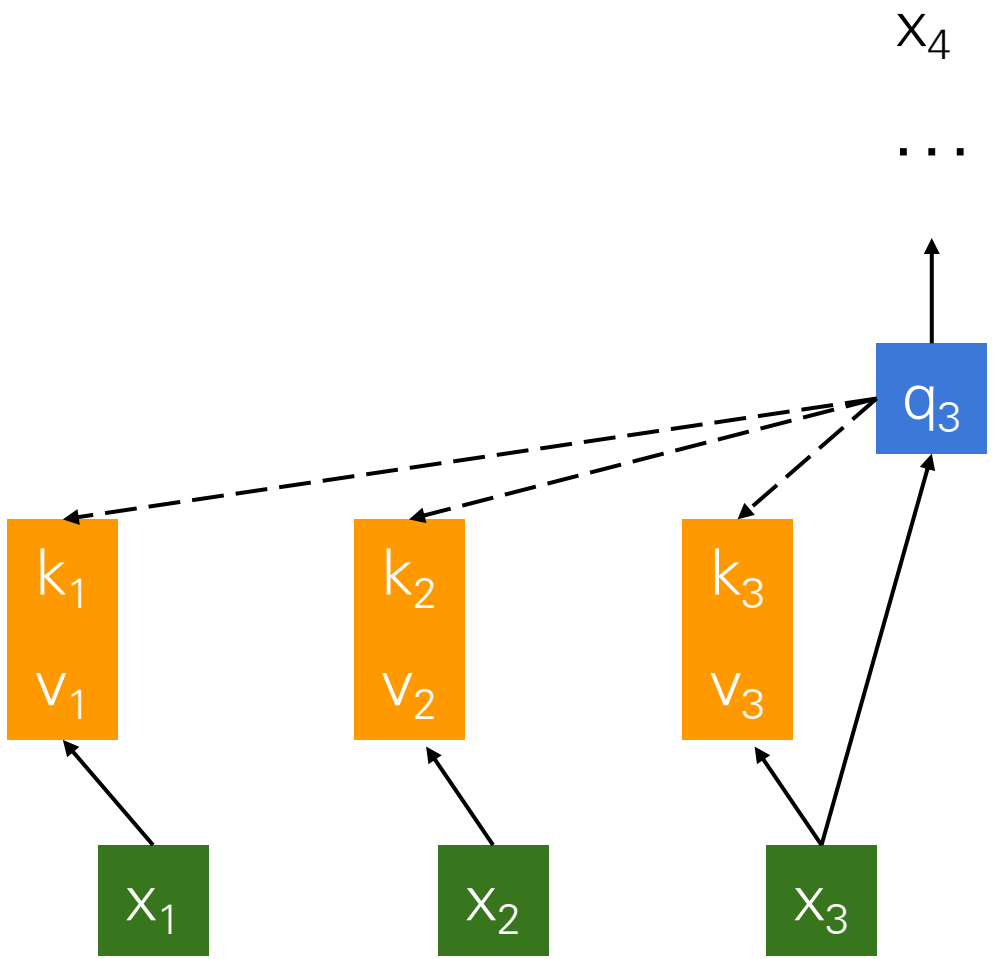
Caching / Efficient Sampling

Naive Sampling



Caching / Efficient Sampling

Naive Sampling

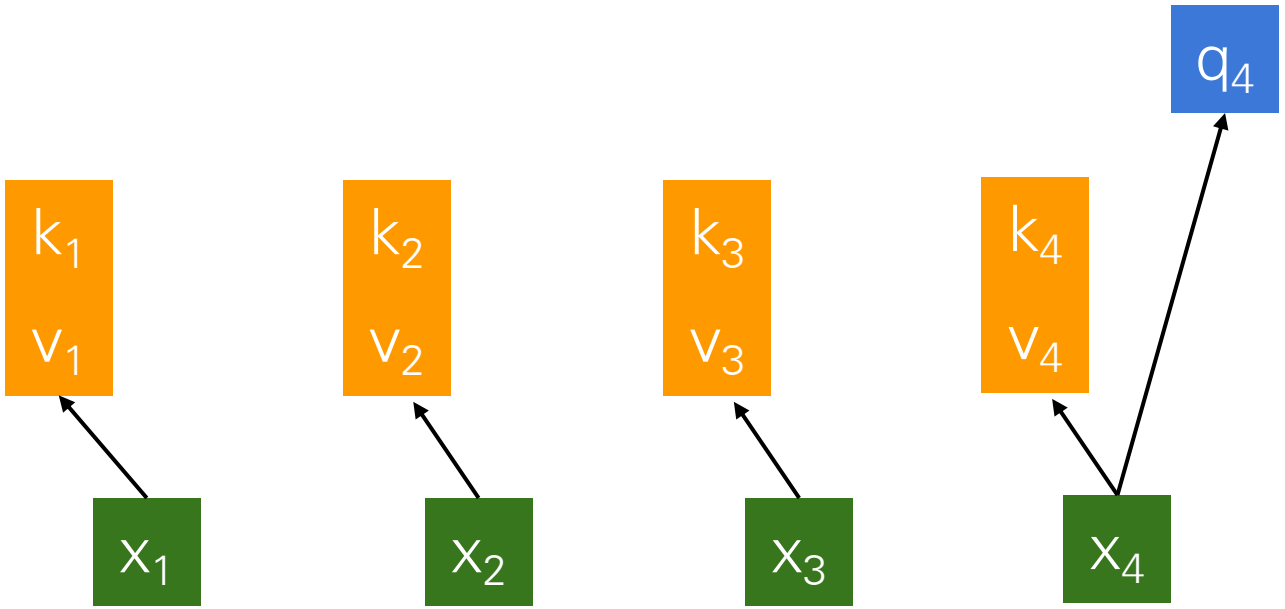


Caching / Efficient Sampling

Naive Sampling

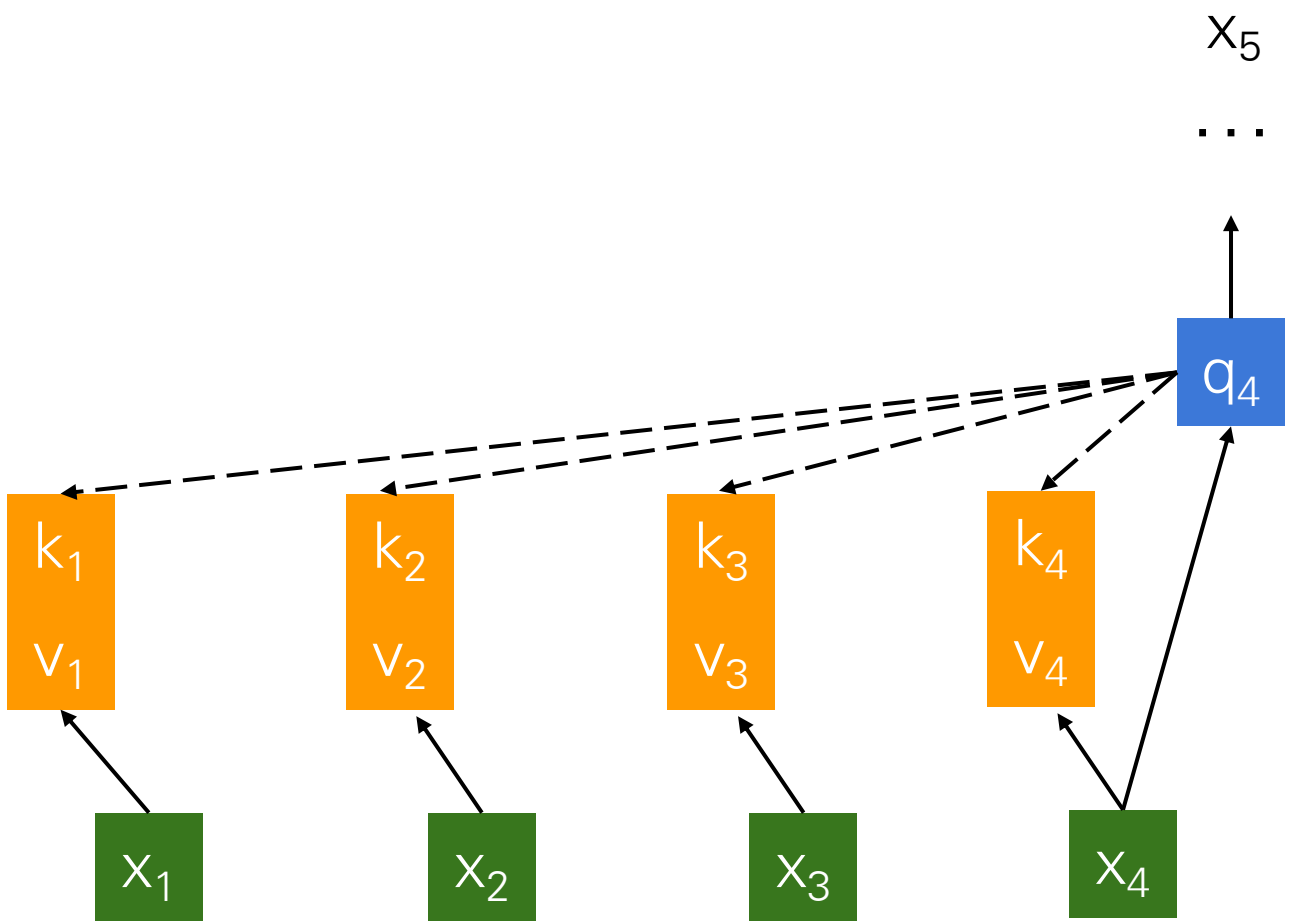
Caching / Efficient Sampling

Naive Sampling



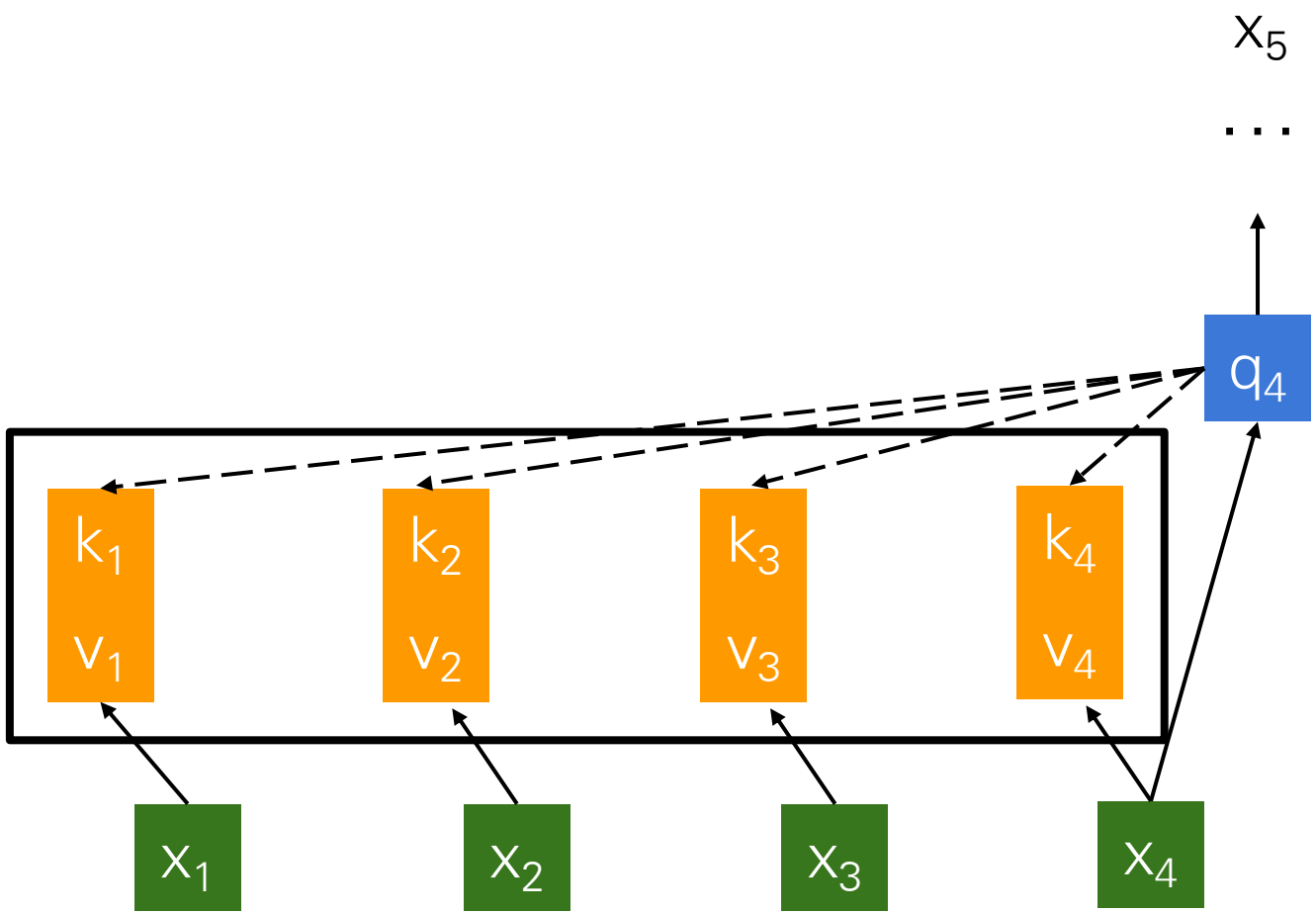
Caching / Efficient Sampling

Naive Sampling



Caching / Efficient Sampling

K / V for all timesteps need to be recomputed for every sampling step

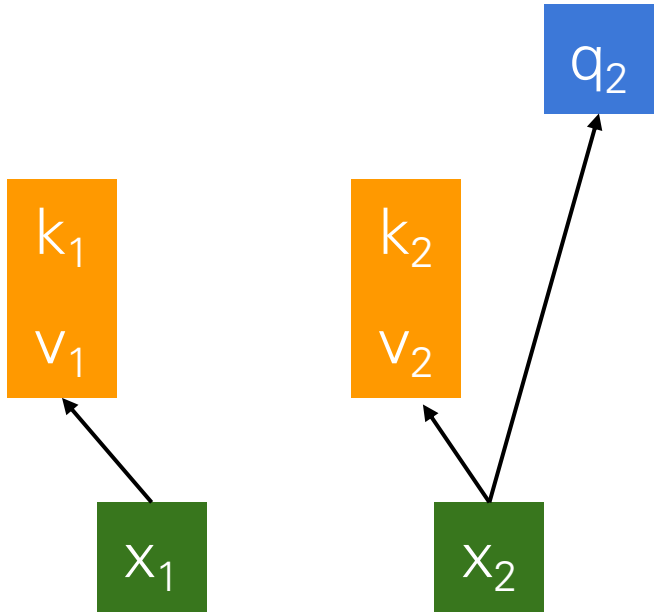


Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step

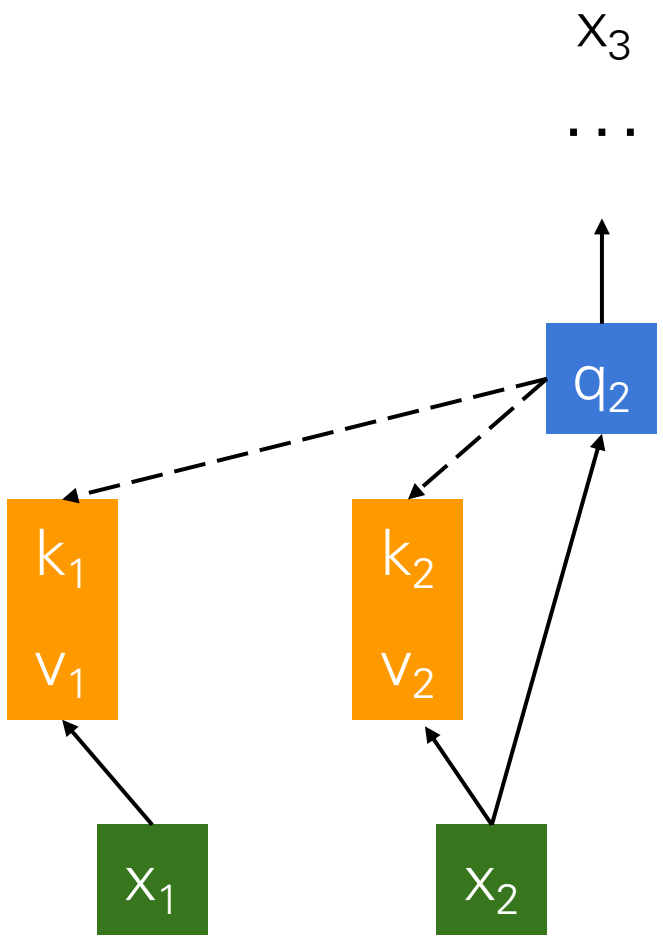
Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



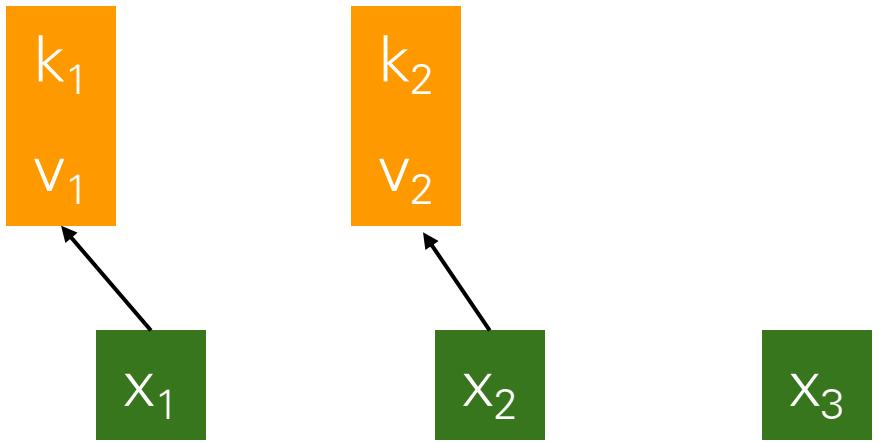
Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



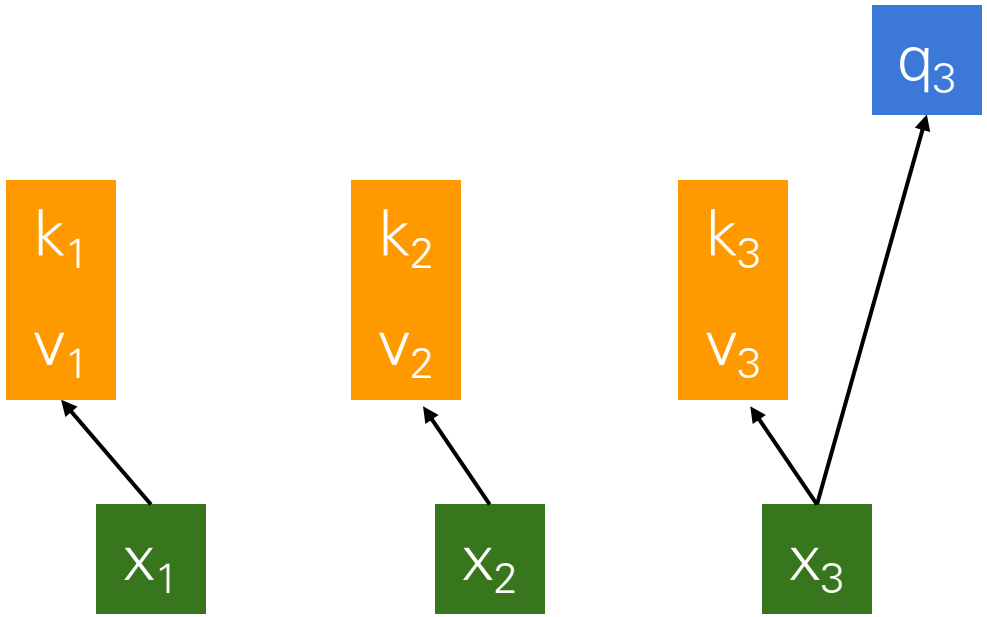
Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



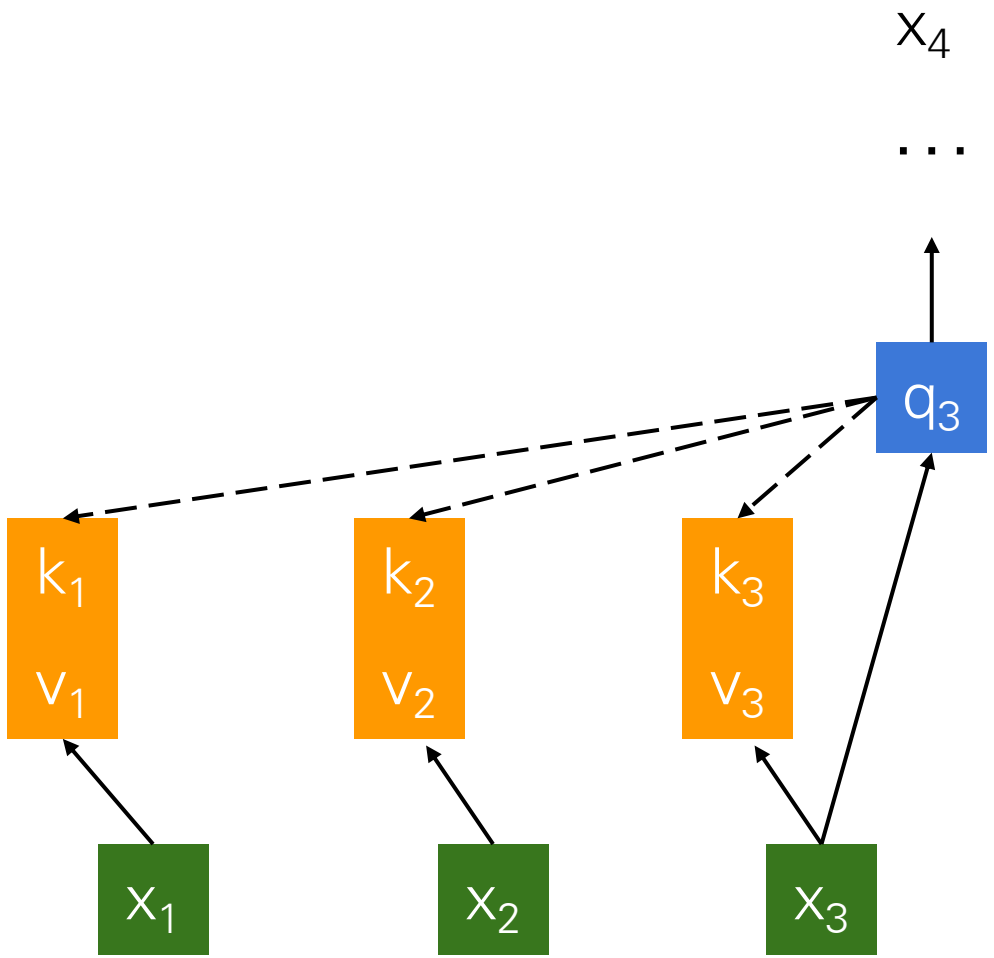
Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



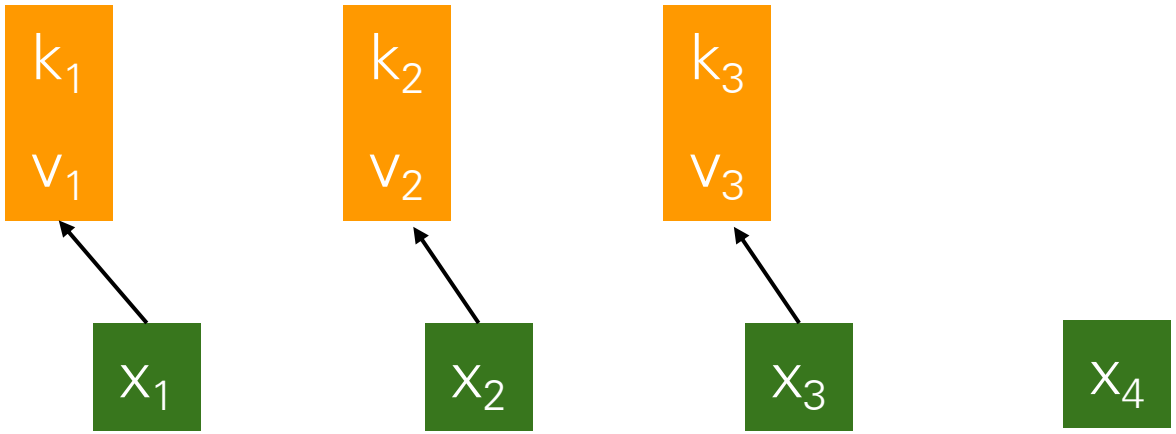
Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



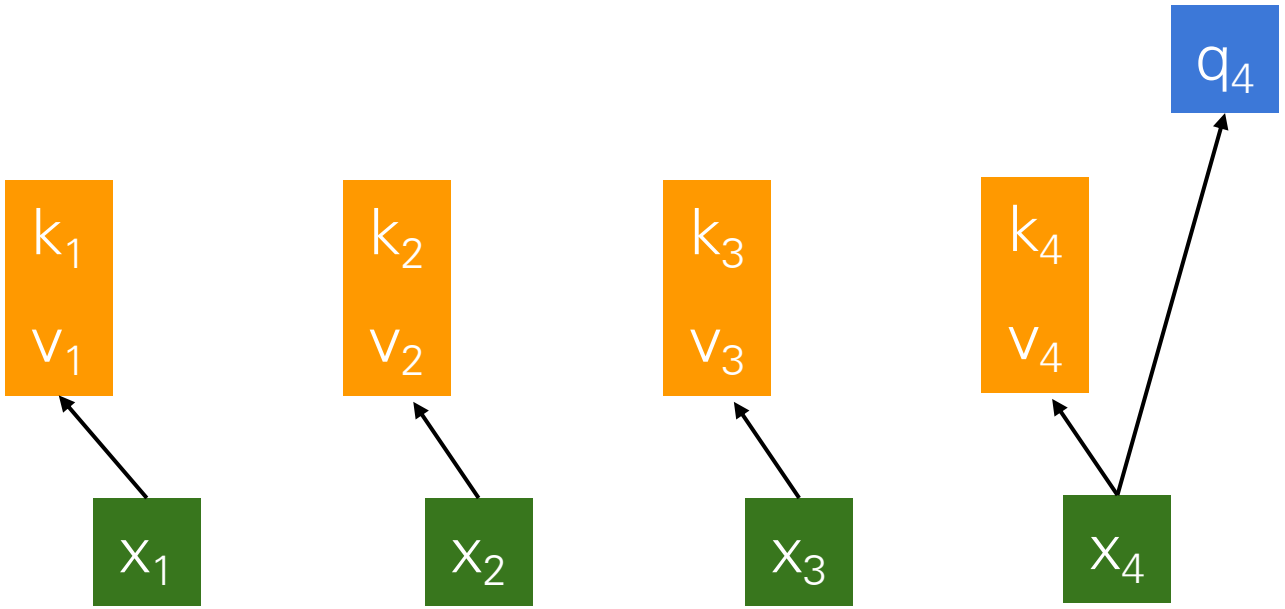
Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



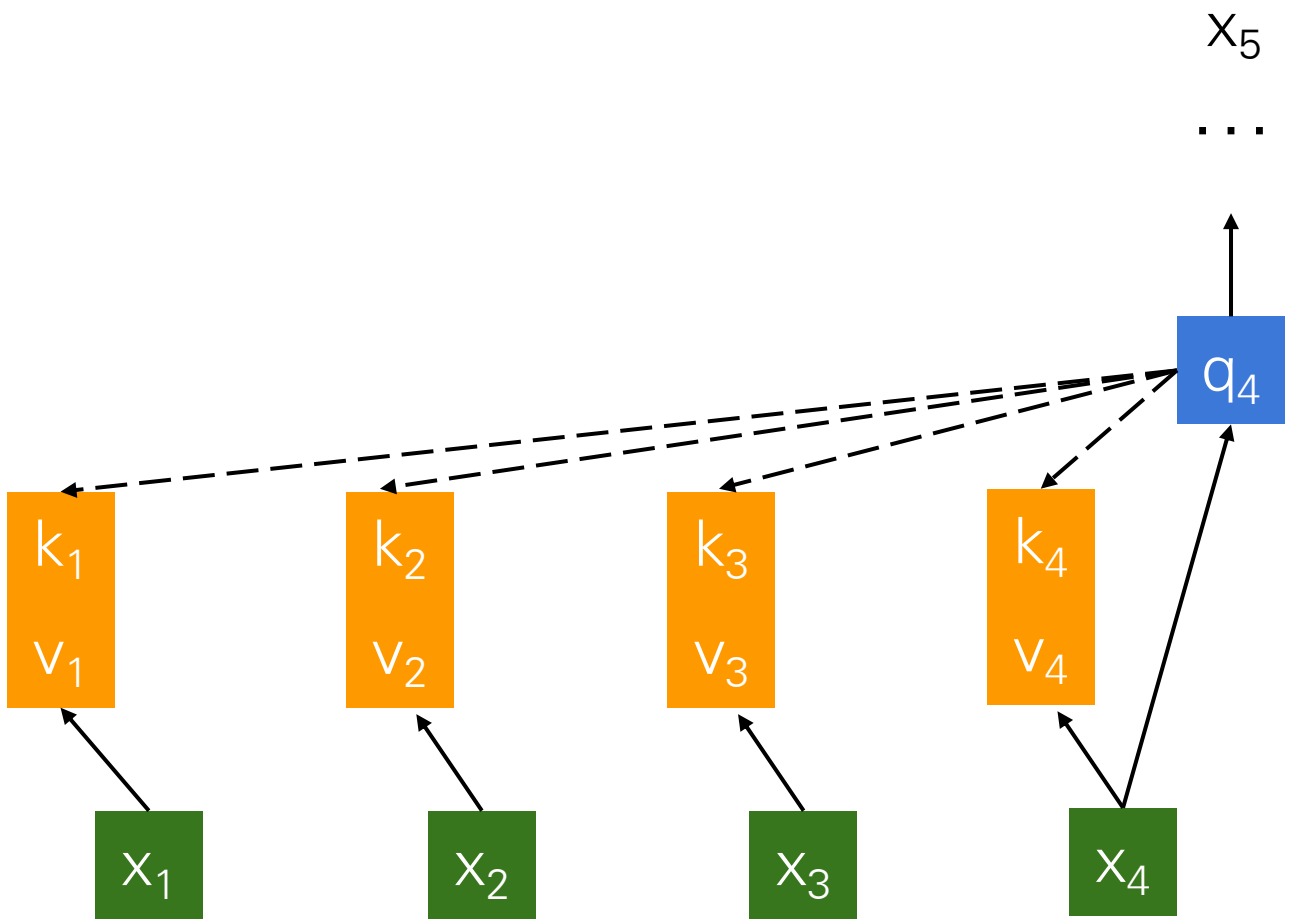
Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



Caching / Efficient Sampling

Key Idea: Cache the K / V for all attention layers each sampling step



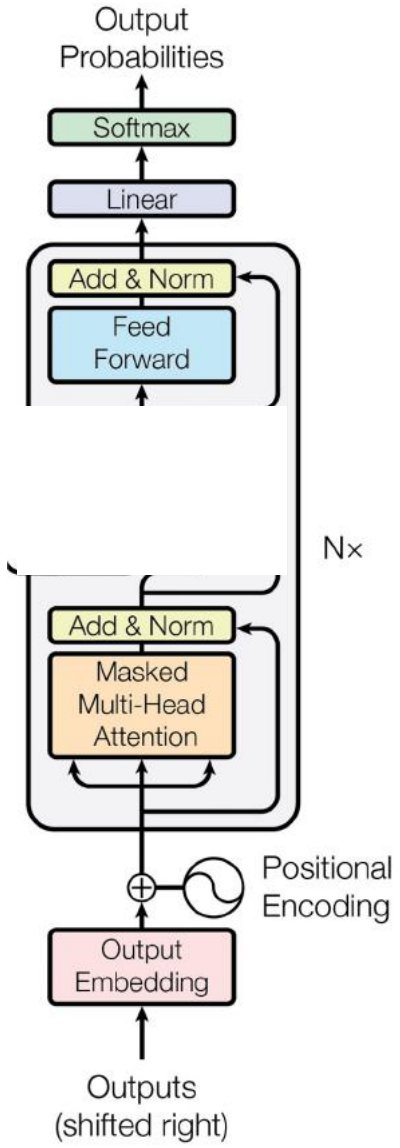
Per-Step

- Naive Sampling: $O(L^2)$
- Caching: $O(L)$

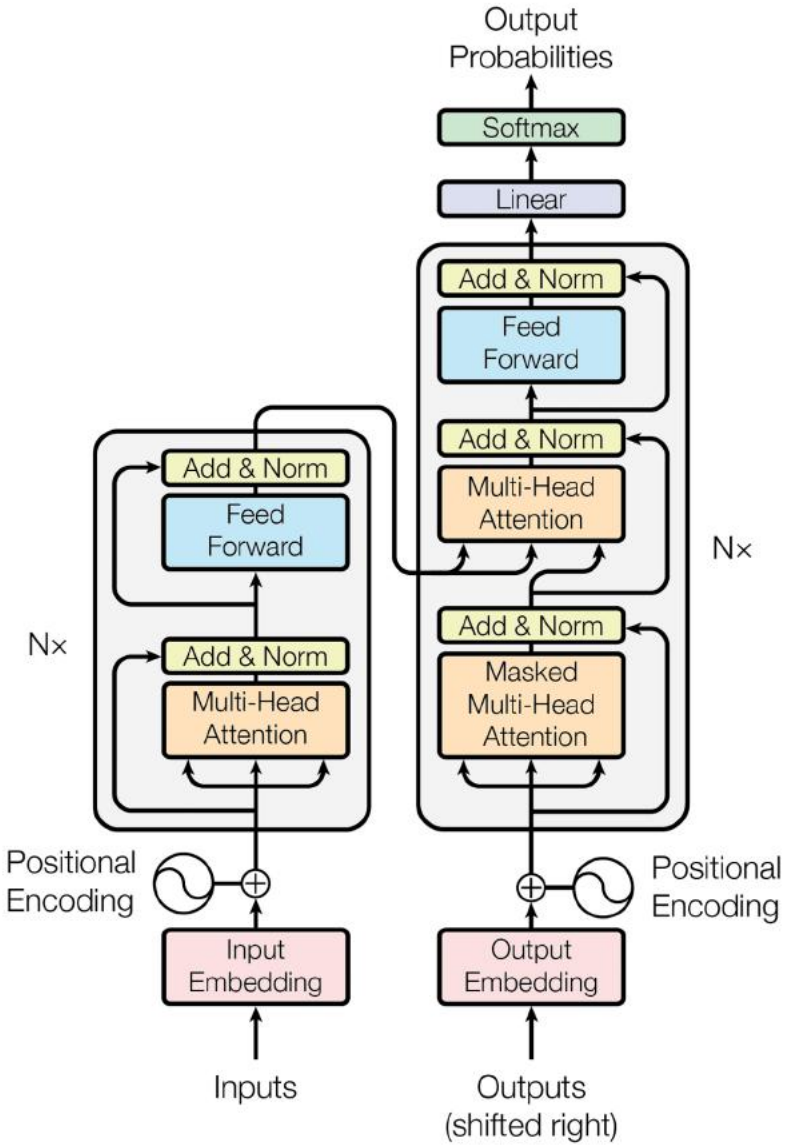
Lecture overview

- motivation
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
- **other things to be aware of**
  - **decoder-only vs. encoder-decoder models**
  - new incarnations of recurrent models
  - alternative / complementary ideas to tokenization

Encoder-Decoder Models



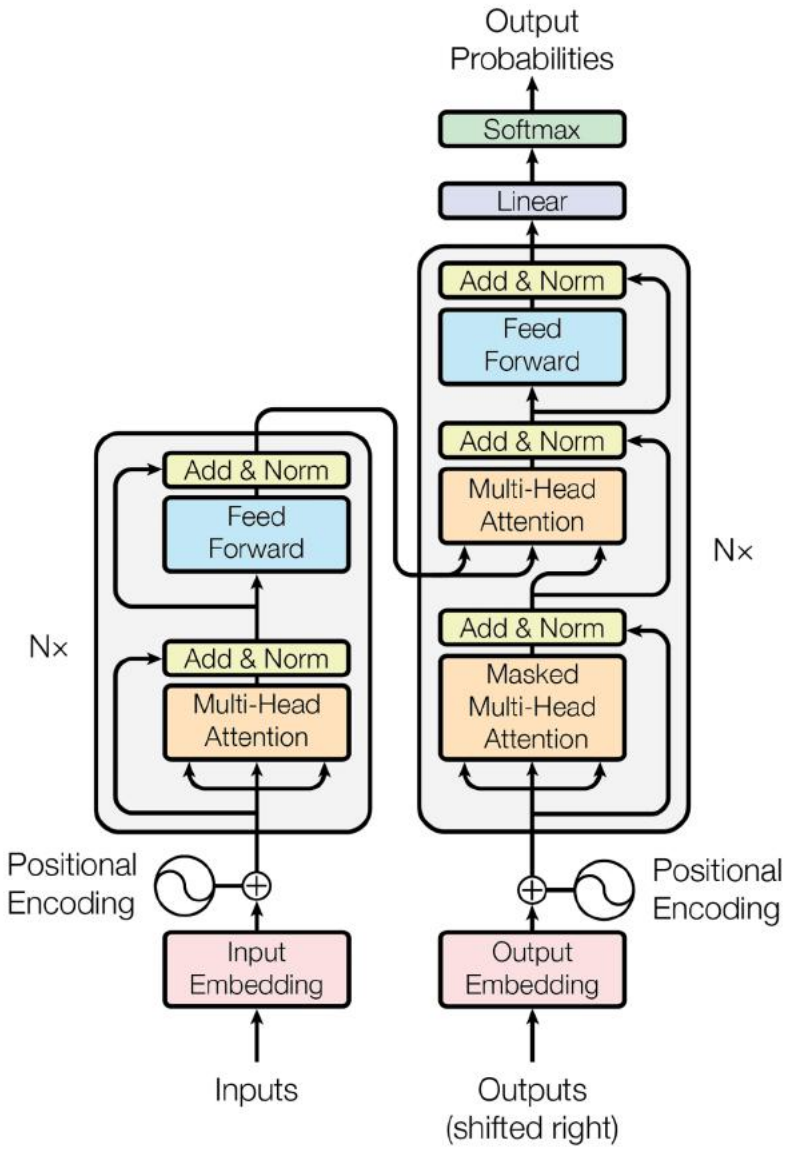
Encoder-Decoder Models



Encoder-Decoder Models

Good architecture choice for cases where you have a clear conditional distribution to model

- Machine Translation
- Text to image generation
- Image captioning
- Video captioning
- Summarization



Encoder-Decoder Models

- Examples:
  - T5 - text model
  - Parti - text to image generation
  - PaLI - image to text generation
- Both model types generally scale similarly, but encoder-decoder model seem to learn more useful (or easier to extract) representations
- Many existing text-image and video generation model condition on features from a T5 encoder. Have not seen as much success from decoder-only models

Parti

Parti-350M **Parti-750M** **Parti-3B** **Parti-20B**

A portrait photo of a kangaroo wearing an orange hoodie and blue sunglasses standing on the grass in front of the Sydney Opera House holding a sign on the chest that says Welcome Friends!

Lecture overview

- motivation
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
- **other things to be aware of**
  - decoder-only vs. Encoder-Decoder models
  - **new incarnations of recurrent models**
  - alternative / complementary ideas to tokenization

Modern Recurrent Models

- Transformers can efficiently model complex distributions
- But scaling to longer sequences can be expensive to do quadratic scaling
- RNNs are O(L) but sequentially unrolling the sequence is slow

How can we compute recurrences in parallel?

Linear State-Space Models

Consider a simple linear state-space model:

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

- A, B, C are discretized version of learnable matrices
- Consider a single SSM layer as a replacement for an attention layer in a transformer

Linear State-Space Models

Parallelization:

$$x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k$$
$$y_k = \overline{\boldsymbol{C}} x_k$$

- Unroll the recurrence

$$x_0 = \overline{\boldsymbol{B}} u_0 \qquad x_1 = \overline{\boldsymbol{AB}} u_0 + \overline{\boldsymbol{B}} u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{AB}} u_1 + \overline{\boldsymbol{B}} u_2$$
$$y_0 = \overline{\boldsymbol{CB}} u_0 \qquad y_1 = \overline{\boldsymbol{CAB}} u_0 + \overline{\boldsymbol{CB}} u_1 \qquad y_2 = \overline{\boldsymbol{CA}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CAB}} u_1 + \overline{\boldsymbol{CB}} u_2$$

- Reorder terms and rewrite as a convolution with kernel size L

$$y_k = \overline{\boldsymbol{CA}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CA}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{CAB}} u_{k-1} + \overline{\boldsymbol{CB}} u_k$$
$$y = \overline{\boldsymbol{K}} * u$$

$$\overline{\boldsymbol{K}} \in \mathbb{R}^L = (\overline{\boldsymbol{CB}}, \overline{\boldsymbol{CAB}}, \ldots, \overline{\boldsymbol{CA}}^{L-1} \overline{\boldsymbol{B}})$$

Linear State-Space Models

Parallelization:

- Convolution method
- Parallel (associative) scan

Both parallelization methods are O(L * log(L)) complexity

Linear Attention

- Consider the standard attention operation

$$V_i' = \frac{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right)}.$$

$$\text{sim}\left(q, k\right) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

- Can we try different similarity functions? We only need a non-negative kernel function (e.g. polynomial, or RBF kernel)

Linear Attention

- Given a kernel with feature representation ϕ(x), write the attention as:

$$V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)}, \longrightarrow V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}.$$

$$\left(\phi(Q)\,\phi(K)^T\right) V = \phi(Q)\left(\phi(K)^T V\right)$$

Causal Linear Attention

- Given a kernel with feature representation $\phi(x)$, write the attention as:

$$V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)},$$

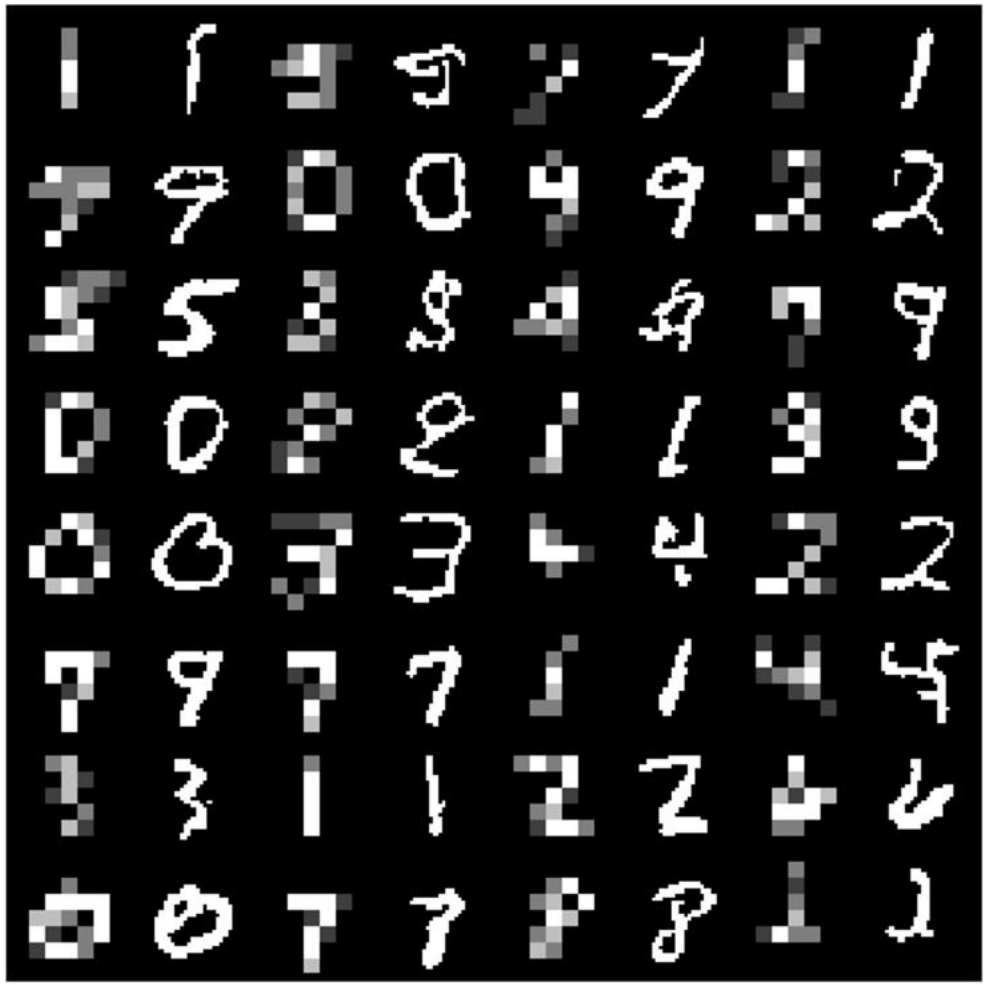
$$\longrightarrow \quad V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{\cancel{N}\ i} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{\cancel{N}\ i} \phi(K_j)}.$$

Recurrent!

Lecture overview

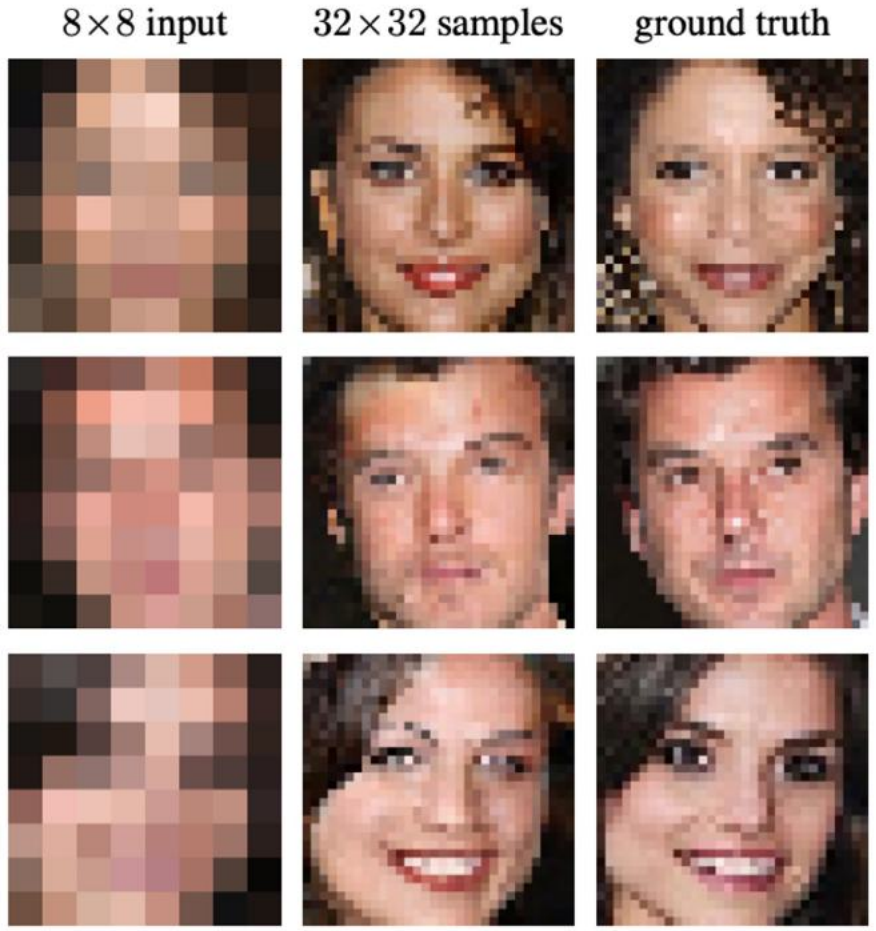
- motivation
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
- **other things to be aware of**
  - decoder-only vs. Encoder-Decoder models
  - new incarnations of recurrent models
  - **alternative / complementary ideas to tokenization**

Super-Resolution with PixelCNN

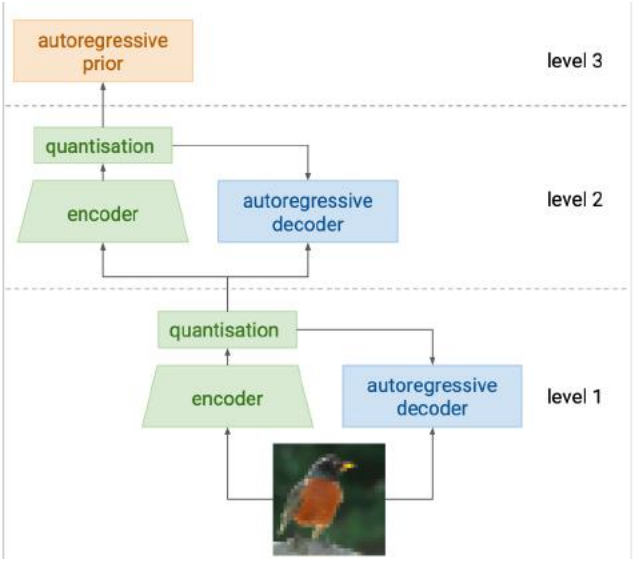


- A PixelCNN is conditioned on 7×7 subsampled MNIST images to generated the corresponding 28×28 image

Super-Resolution with PixelCNN



Hierarchical Autoregressive Models with Auxiliary Decoders

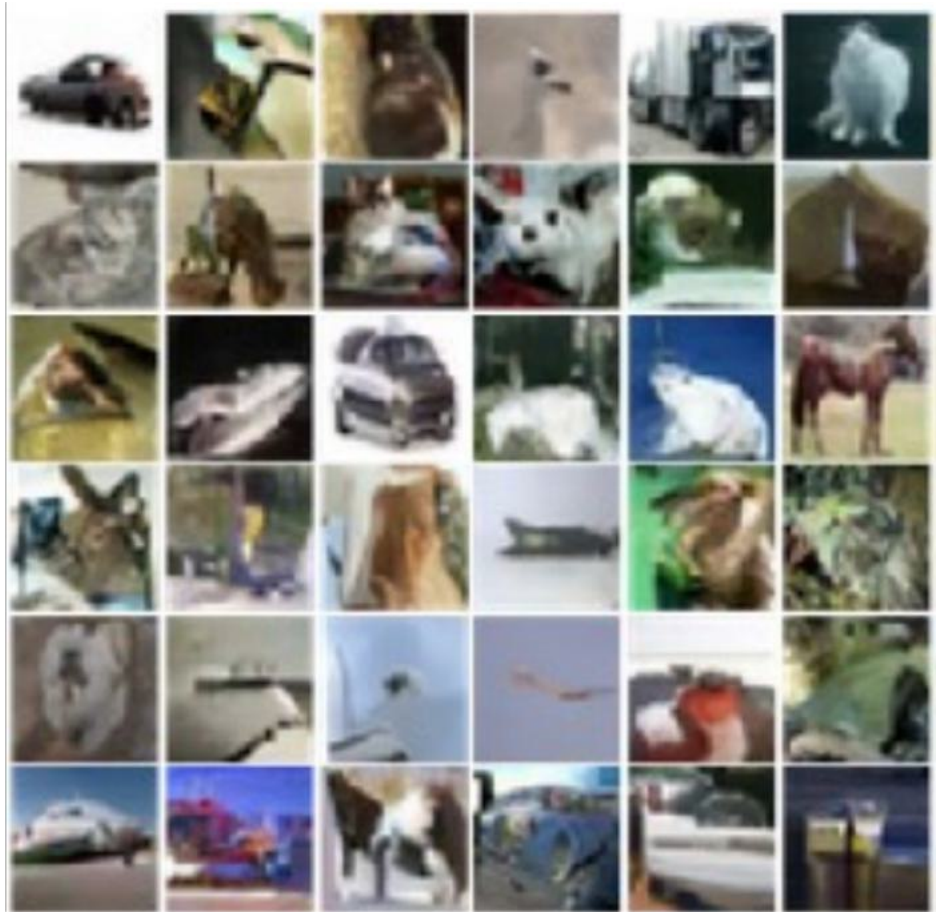


De Fauw, Jeffrey, Sander Dieleman, and Karen Simonyan. "Hierarchical autoregressive image models with auxiliary decoders." arXiv preprint arXiv:1903.04933 (2019).

Hierarchy: Grayscale PixelCNN

- Design an autoregressive model architecture that takes advantage of the structure of data
- Learn a PixelCNN on binary images, and a PixelCNN conditioned on binary images to generate colored images

Grayscal PixelCNN



Next lecture:
Flow-Based Models