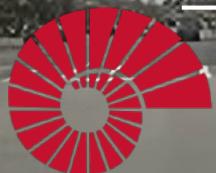


# COMP547

## DEEP UNSUPERVISED LEARNING

Lecture #5 – Autoregressive Models



KOÇ  
UNIVERSITY

Aykut Erdem // Koç University // Spring 2024

# 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

# 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  $p_{\text{data}}$  from samples  $x^{(1)}, \dots, x^{(n)} \sim p_{\text{data}}(x)$
- Learns a distribution  $p$  that allows:
  - Computing  $p(x)$  for arbitrary  $x$
  - Sampling  $x \sim p(x)$
- Today: **discrete** data

# Desiderata

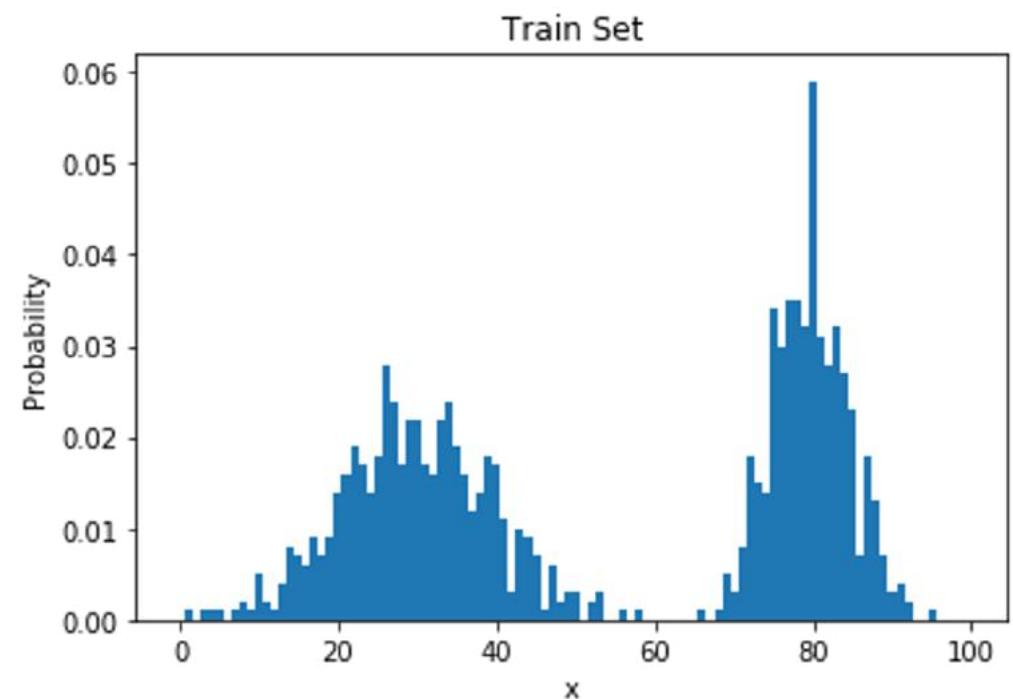
- We want to estimate distributions of **complex, high-dimensional data**
  - A  $128 \times 128 \times 3$  image lies in a  $\sim 50,000$ -dimensional space
- We also want computational and statistical efficiency
  - Efficient training and model representation
  - Expressiveness and generalization
  - 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

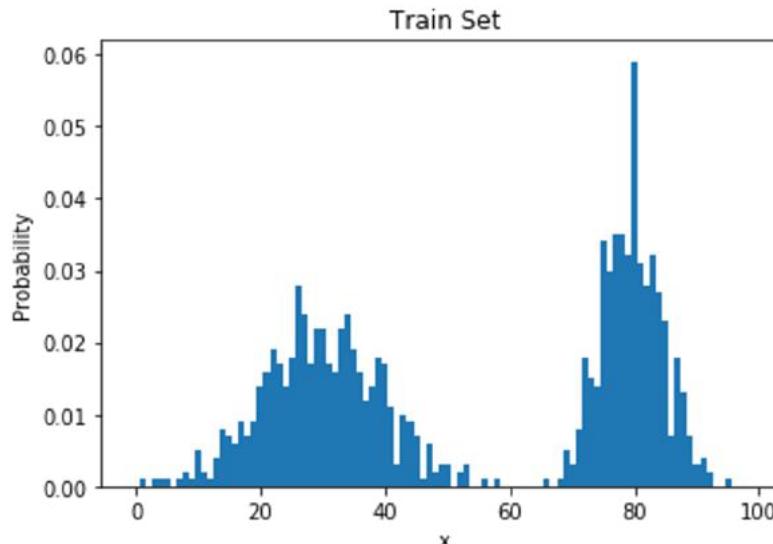
- Recall: the goal is to estimate  $p_{\text{data}}$  from samples  $x^{(1)}, \dots, x^{(n)} \sim p_{\text{data}}(x)$
- **Suppose** the samples take on values in a finite set  $\{1, \dots, k\}$
- The model: a **histogram**
  - (Redundantly) described by  $k$  nonnegative numbers:  $p_1, \dots, p_k$
  - To train this model: count frequencies
$$p_i = (\# \text{ times } i \text{ appears in the dataset}) / (\# \text{ points in the dataset})$$



# Inference and Sampling

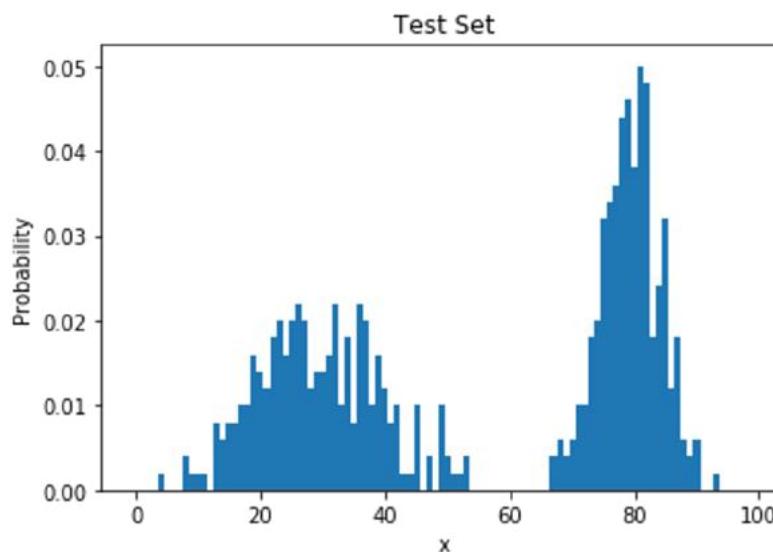
- **Inference** (querying  $p_i$  for arbitrary  $i$ ): simply a lookup into the array  $p_1, \dots, p_k$
- **Sampling** (lookup into the inverse cumulative distribution function)
  1. From the model probabilities  $p_1, \dots, p_k$ , compute the cumulative distribution
$$F_i = p_1 + \dots + p_i \quad \text{for all } i \in \{1, \dots, k\}$$
  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

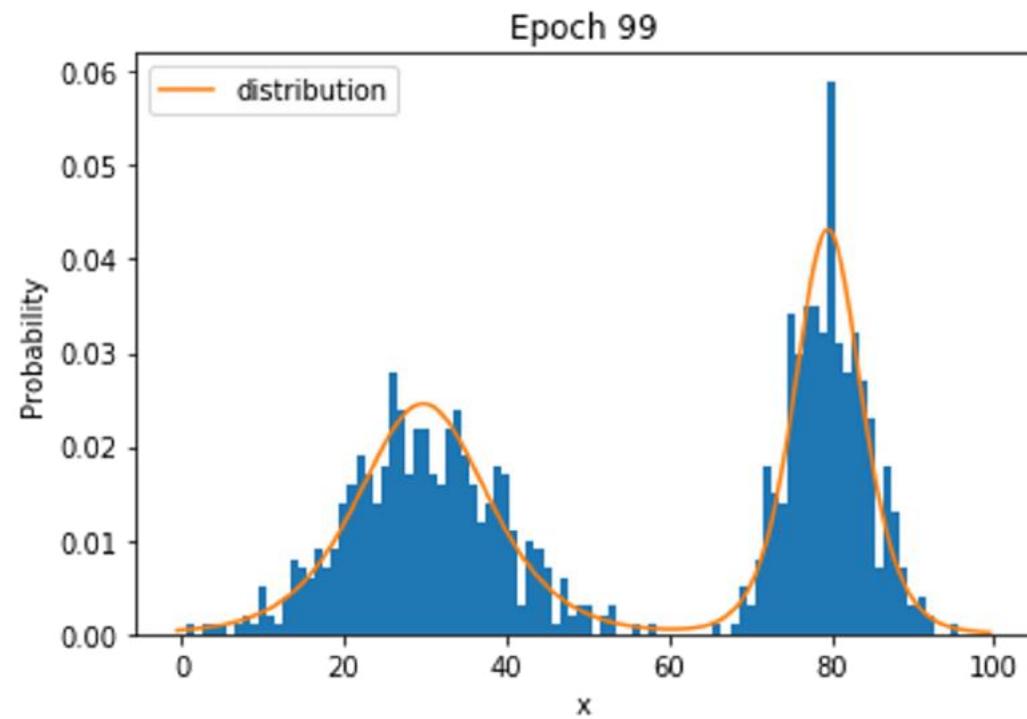
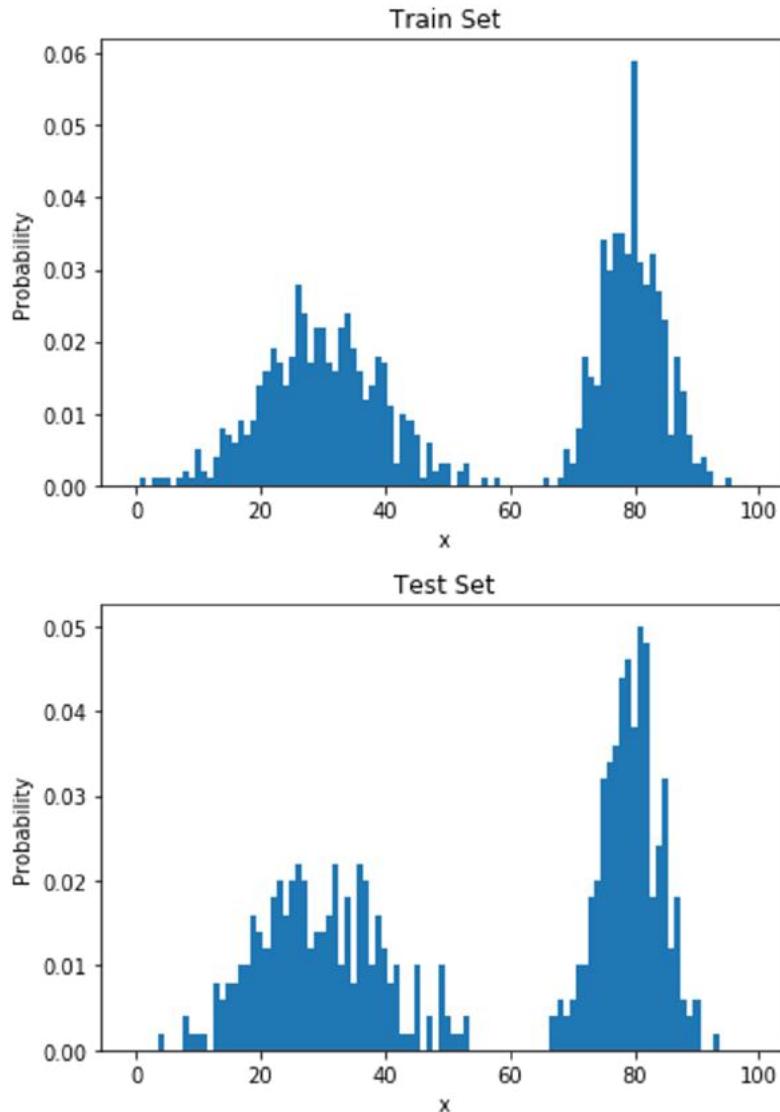


learned histogram = training data distribution

→ often poor generalization



# Parameterized distributions



Fitting a parameterized distribution  
often generalizes better

# Likelihood-based generative models

- Recall: the goal is to estimate  $p_{\text{data}}$  from  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)} \sim p_{\text{data}}(\mathbf{x})$
- We introduce a **parameterized** model  $p_{\theta}(\mathbf{x})$  with the goal to learn  $\theta$  so that  $p_{\theta}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$
- To learn  $\theta$ , we pose a search problem over parameters
$$\arg \min_{\theta} \text{loss}(\theta, \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)})$$
- Want the loss function + search procedure to:
  - Work with large datasets ( $n$  is large, say millions of training examples)
  - Yield  $\theta$  such that  $p_{\theta}$  matches  $p_{\text{data}}$  — 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  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ , find  $\theta$  by solving the optimization problem

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

- 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  $\theta$ , it solves

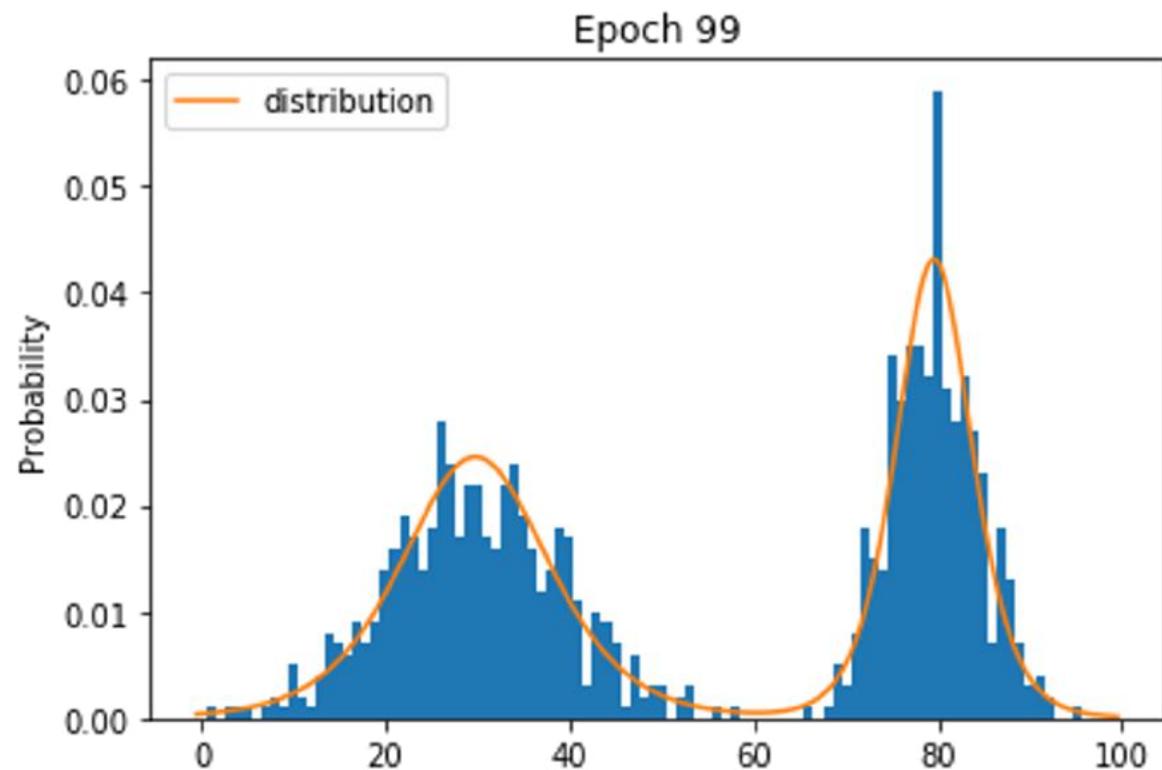
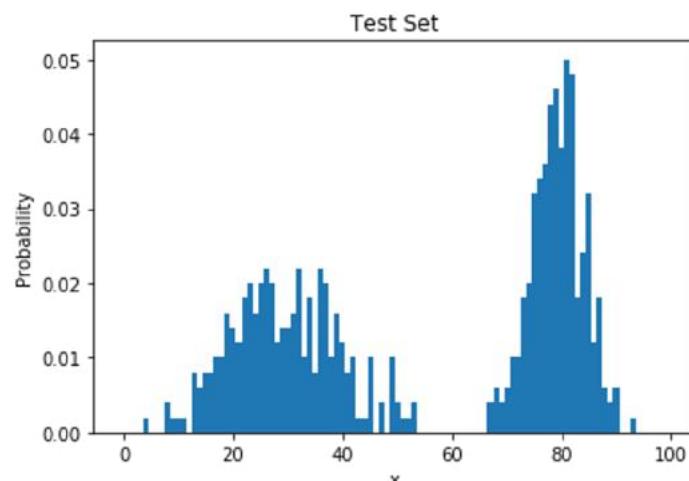
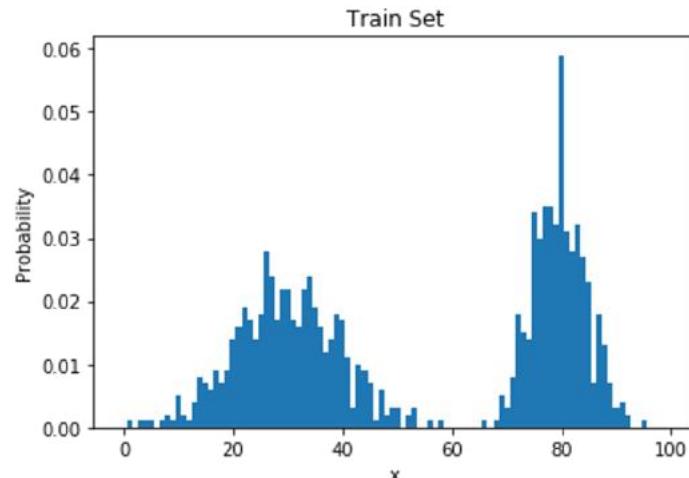
$$\arg \min_{\theta} \mathbb{E}[f(\theta)]$$

- 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



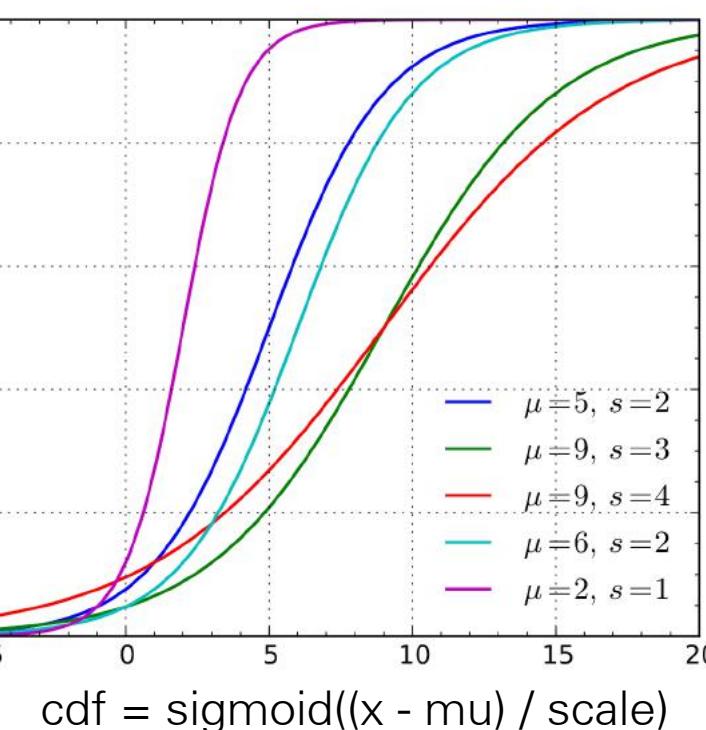
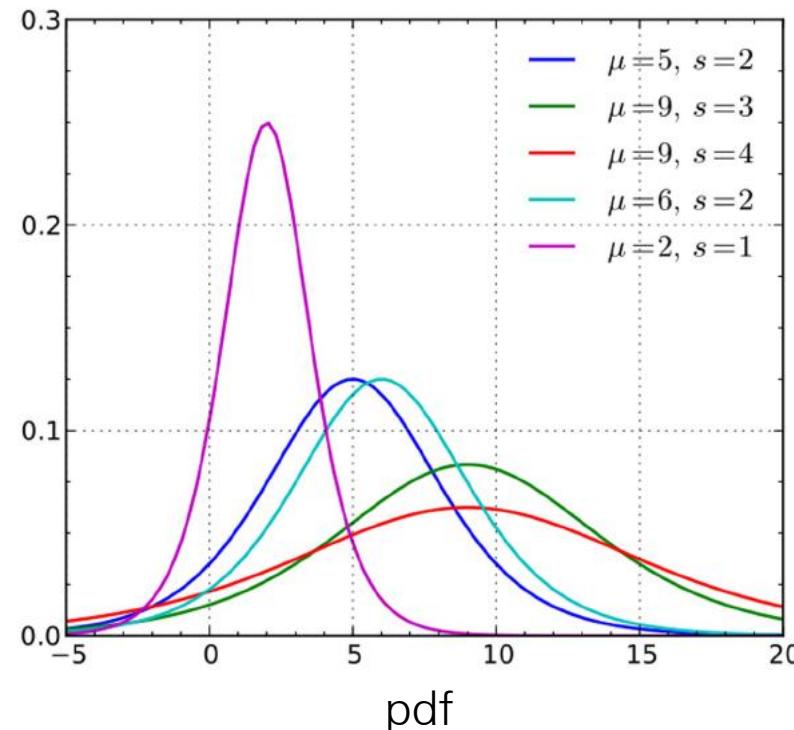
Fitting a parameterized distribution often generalizes better

# 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

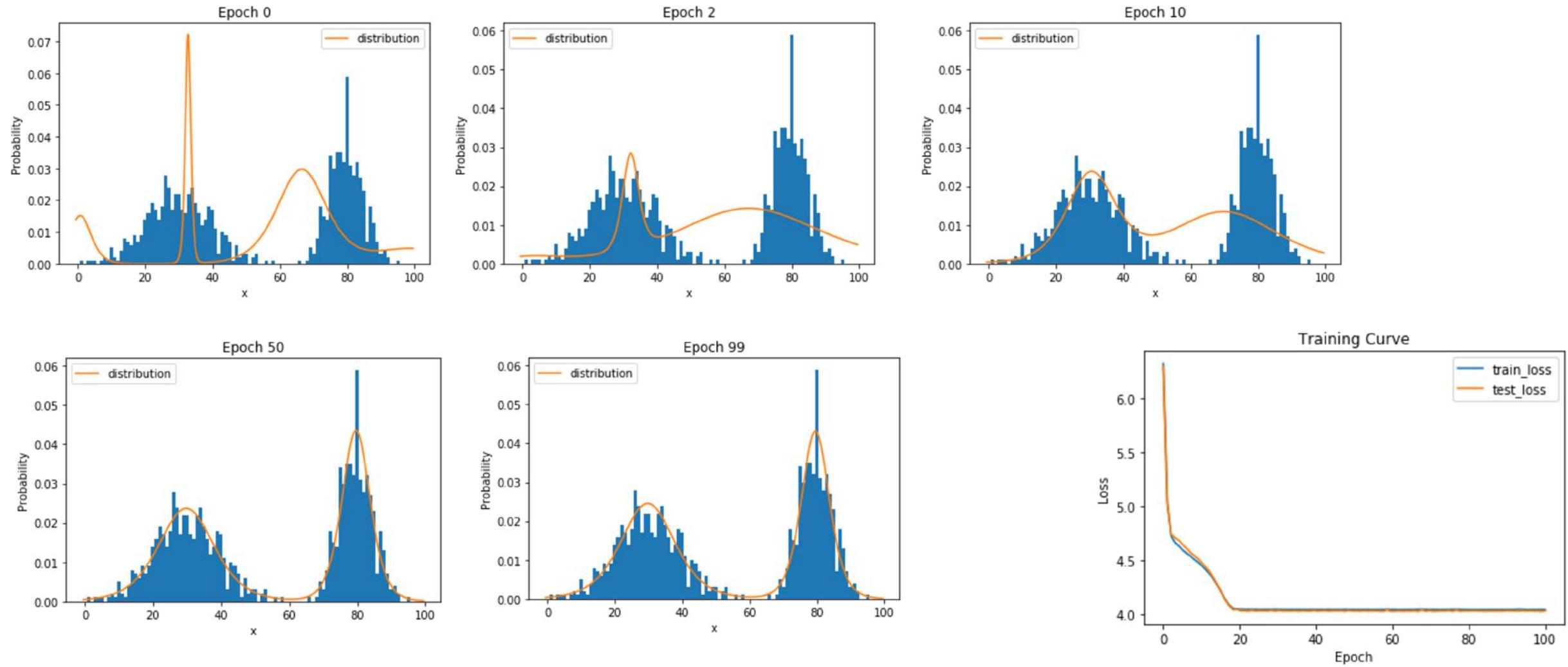


# Mixture of Logistics for Multi-modal

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

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

# 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!

# Chain Rule

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

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

# Bayes nets and neural nets

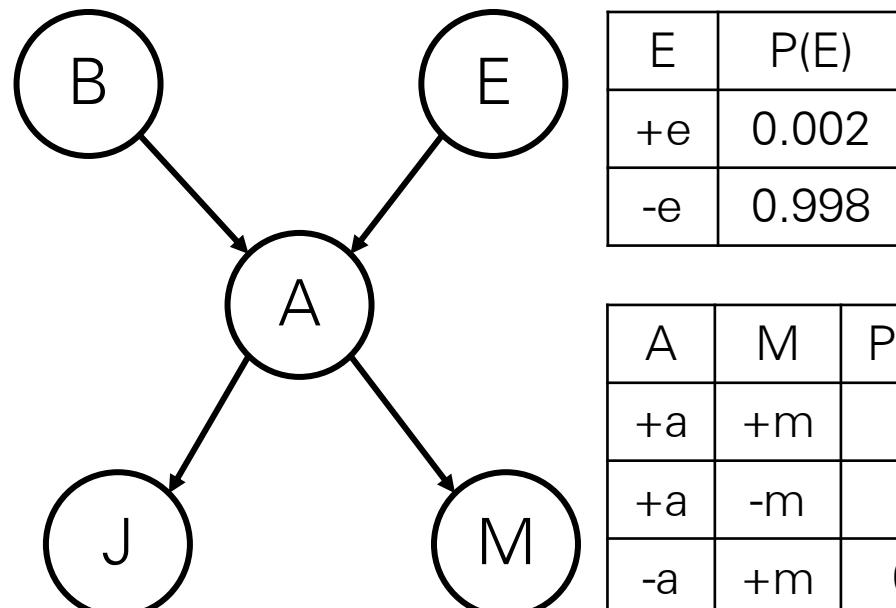
$$p_{\theta}(x) = \prod_{i=1}^d p_{\theta}(x_i \mid \text{Parents}(x_i))$$

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

- + 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

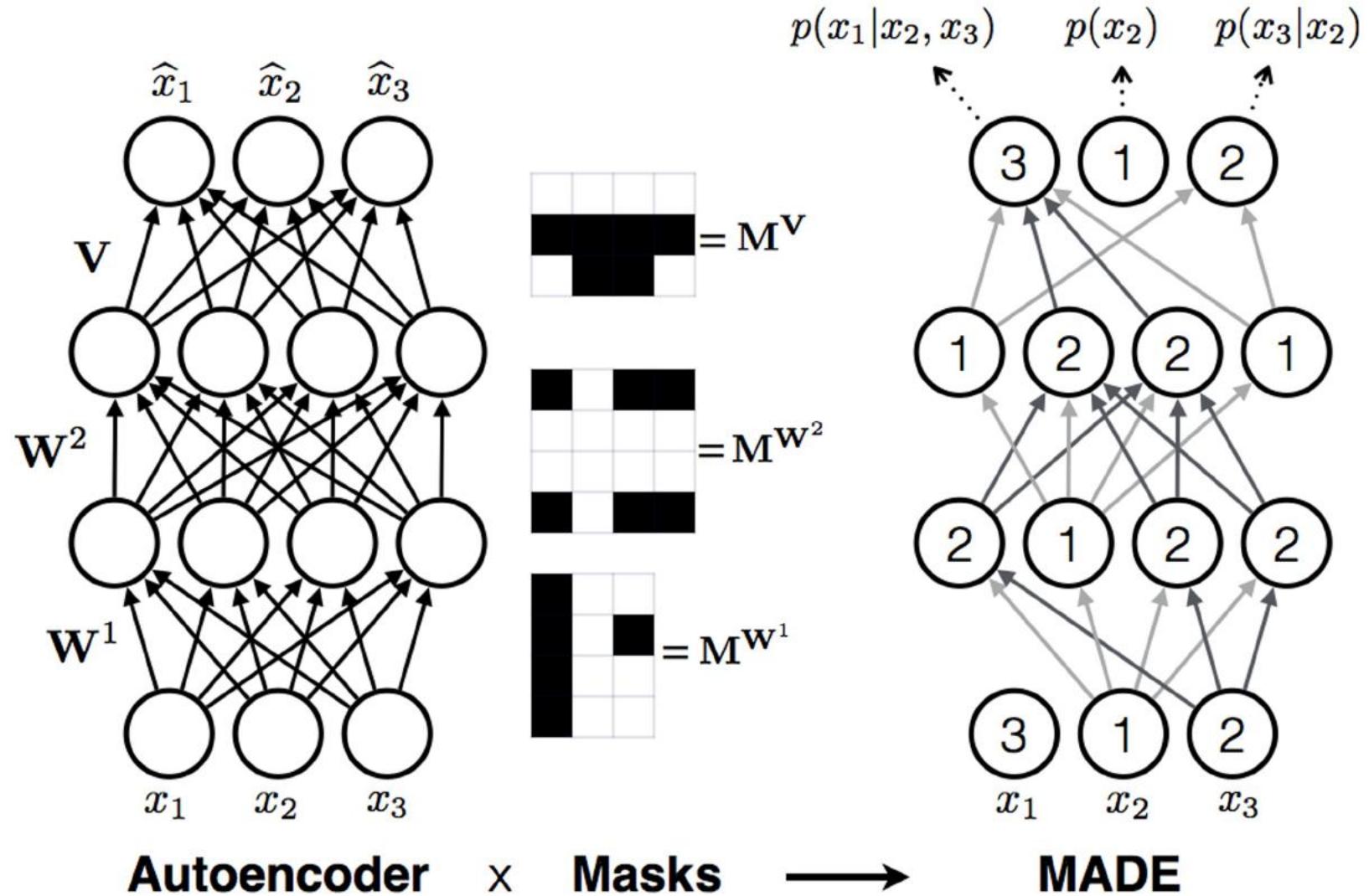
# Autoregressive models

- 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

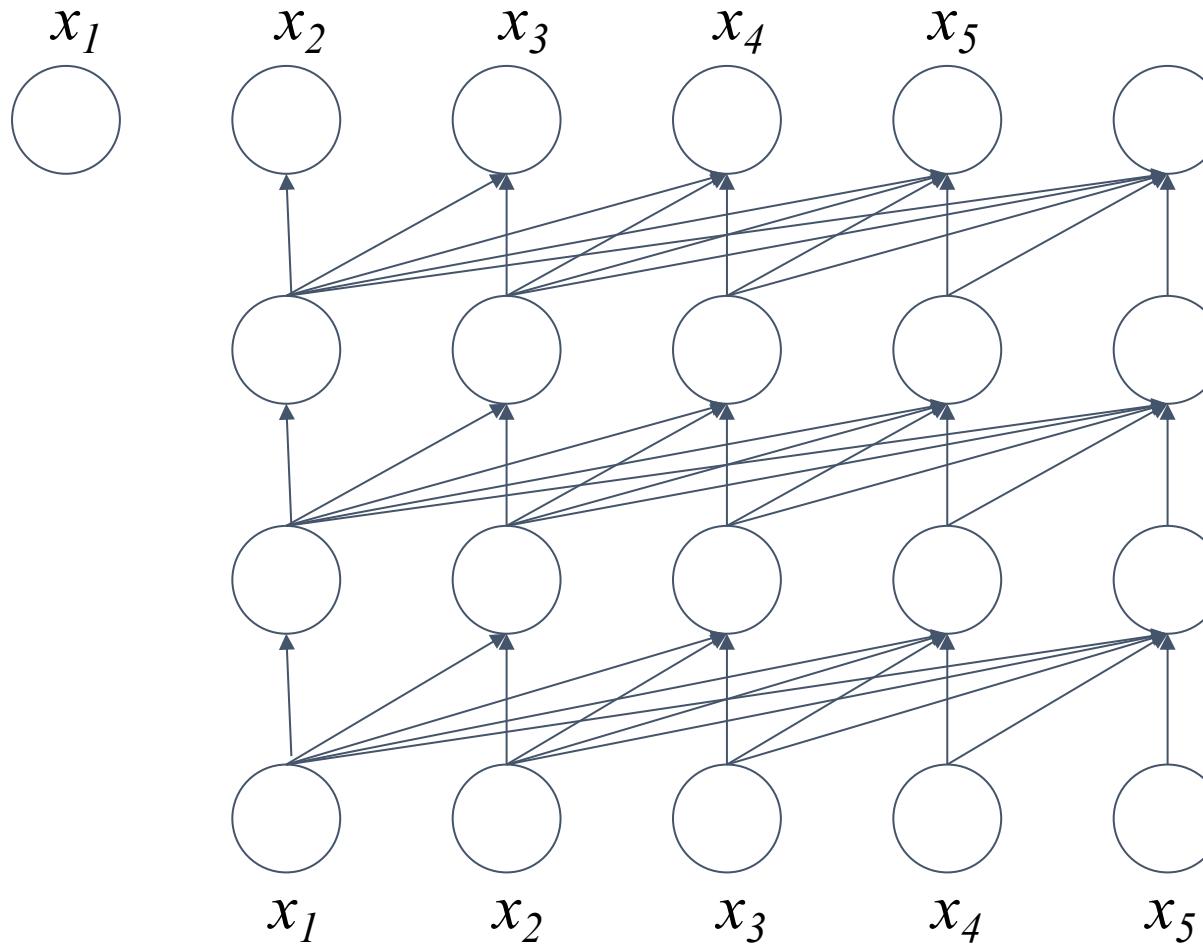
# Masked Autoencoder for Distribution Estimation (MADE)





# Masked Autoencoder for Distribution Estimation (MADE)

In more modern notation/diagram



Every hidden layer node  
can be vector valued

Every edge can be its own  
MLP connection

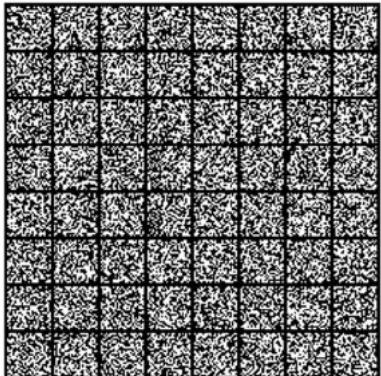
# MNIST

- Handwritten digits
- 28x28
- 60,000 train
- 10,000 test
- Original: greyscale
- “Binarized MNIST” – 0/1 (black/white)

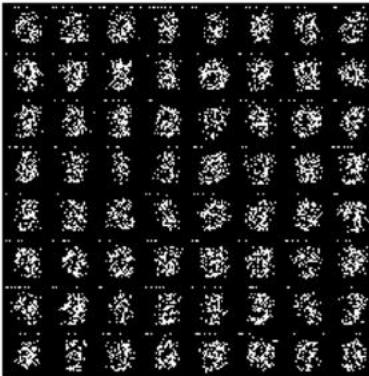


# MADE on MNIST

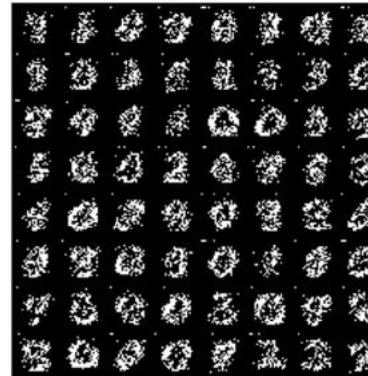
Initialization



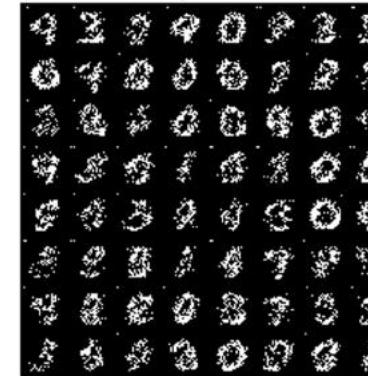
Epoch 0



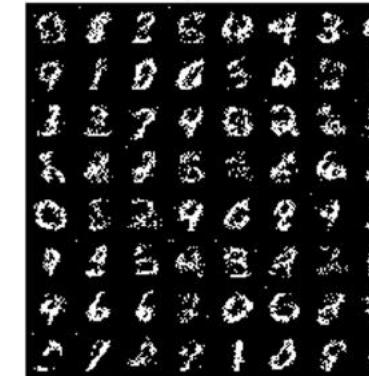
Epoch 1



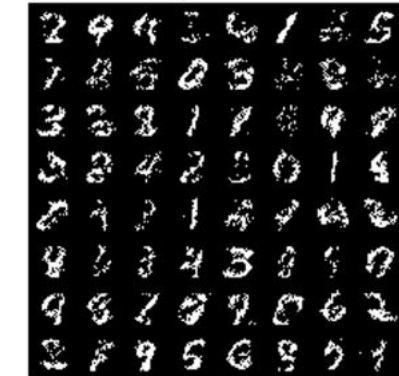
Epoch 2



Epoch 8



Epoch 19



# Masked Autoencoder for Distribution Estimation (MADE)

```
# param: normal FC weight
# x: layer input
# y: autoregressive activations
mask = get_linear_ar_mask(in_size, out_size)
# create mask of pattern
# array([[0., 1., 1.],
#        [0., 0., 1.],
#        [0., 0., 0.]], dtype=float32)
y = tf.matmul(x, param * mask)
```

# 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)	$\approx 86.34$	Intractable
DBM 2hl	$\approx 84.62$	
DBN 2hl	$\approx 84.55$	
DARN $n_h=500$	$\approx 84.71$	
DARN $n_h=500$ , adaNoise	$\approx 84.13$	
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

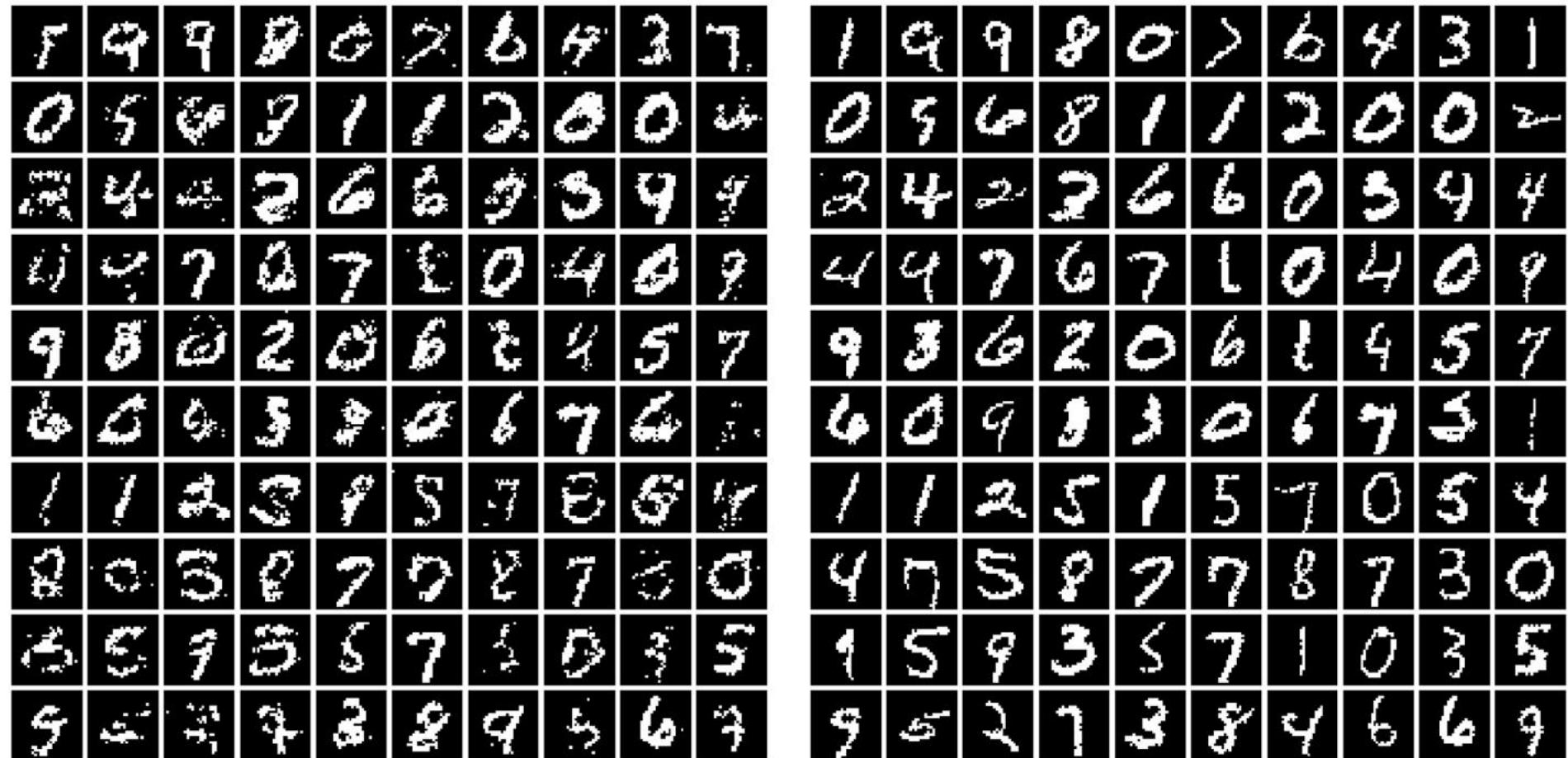
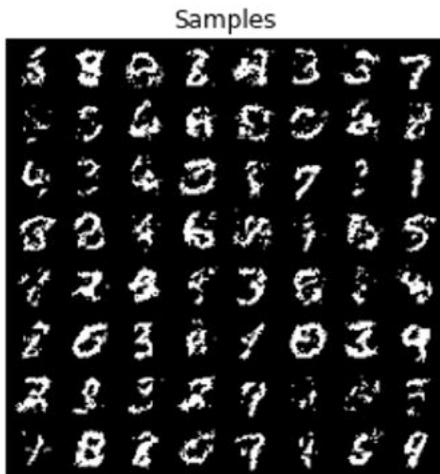


Figure 3. Left: Samples from a 2 hidden layer MADE. Right: Nearest neighbour in binarized MNIST.

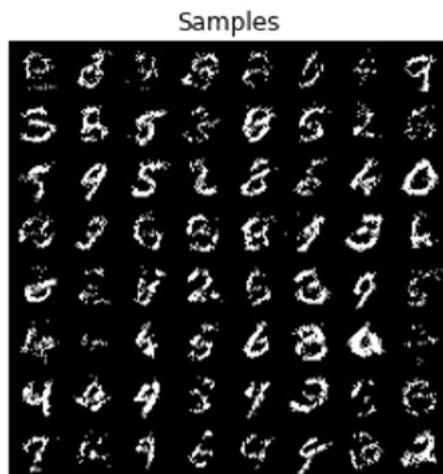
# MADE – Different Orderings

- All orderings achieve roughly the same bits per dim  $\left(\frac{\log_2 p(x)}{\dim(x)}\right)$ , but samples are different

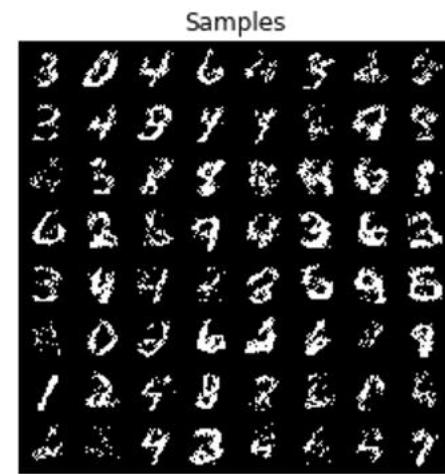
Random  
Permutation



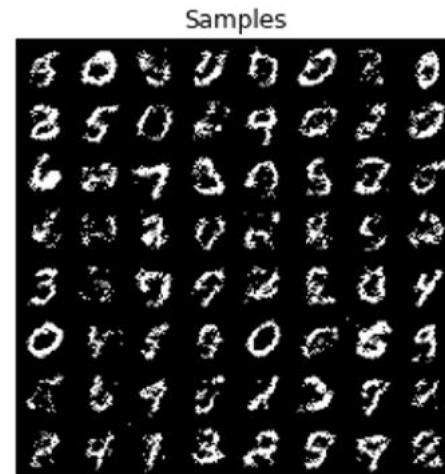
Even then Odd  
Indices



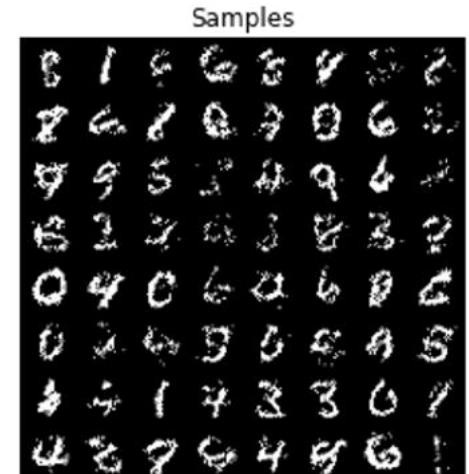
Rows  
(Raster Scan)



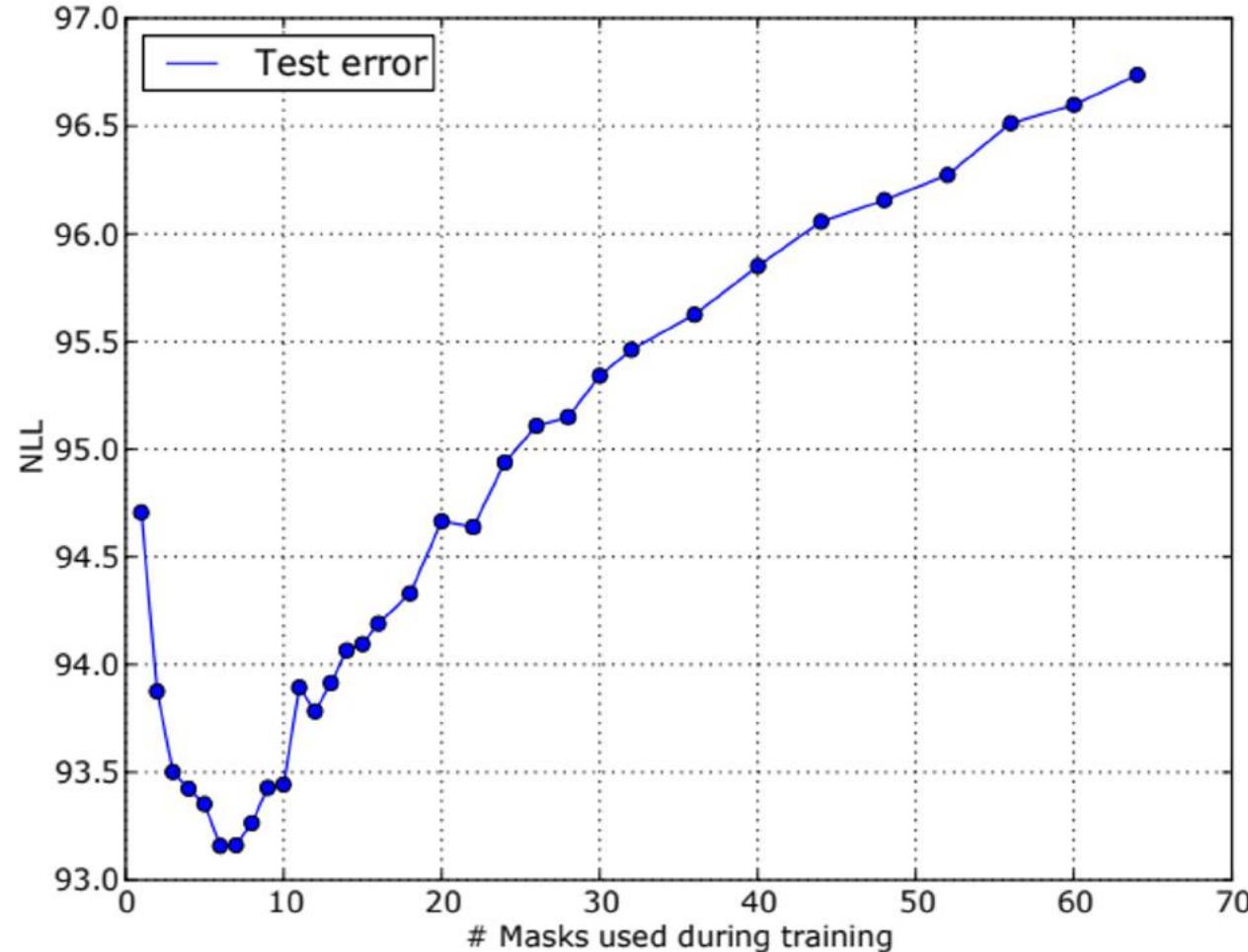
Columns



Top to Middle,  
Bottom to Middle



# MADE: Multiple Orderings



# Autoregressive models

- Recall, AutoRegressive Model  $p_{\theta}(x) = \prod_{i=1}^d p_{\theta}(x_i | x_{1:i-1})$
- 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

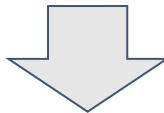
# Autoregressive models

- Recall, AutoRegressive Model 
$$p_{\theta}(x) = \prod_{i=1}^d p_{\theta}(x_i | x_{1:i-1})$$
- 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

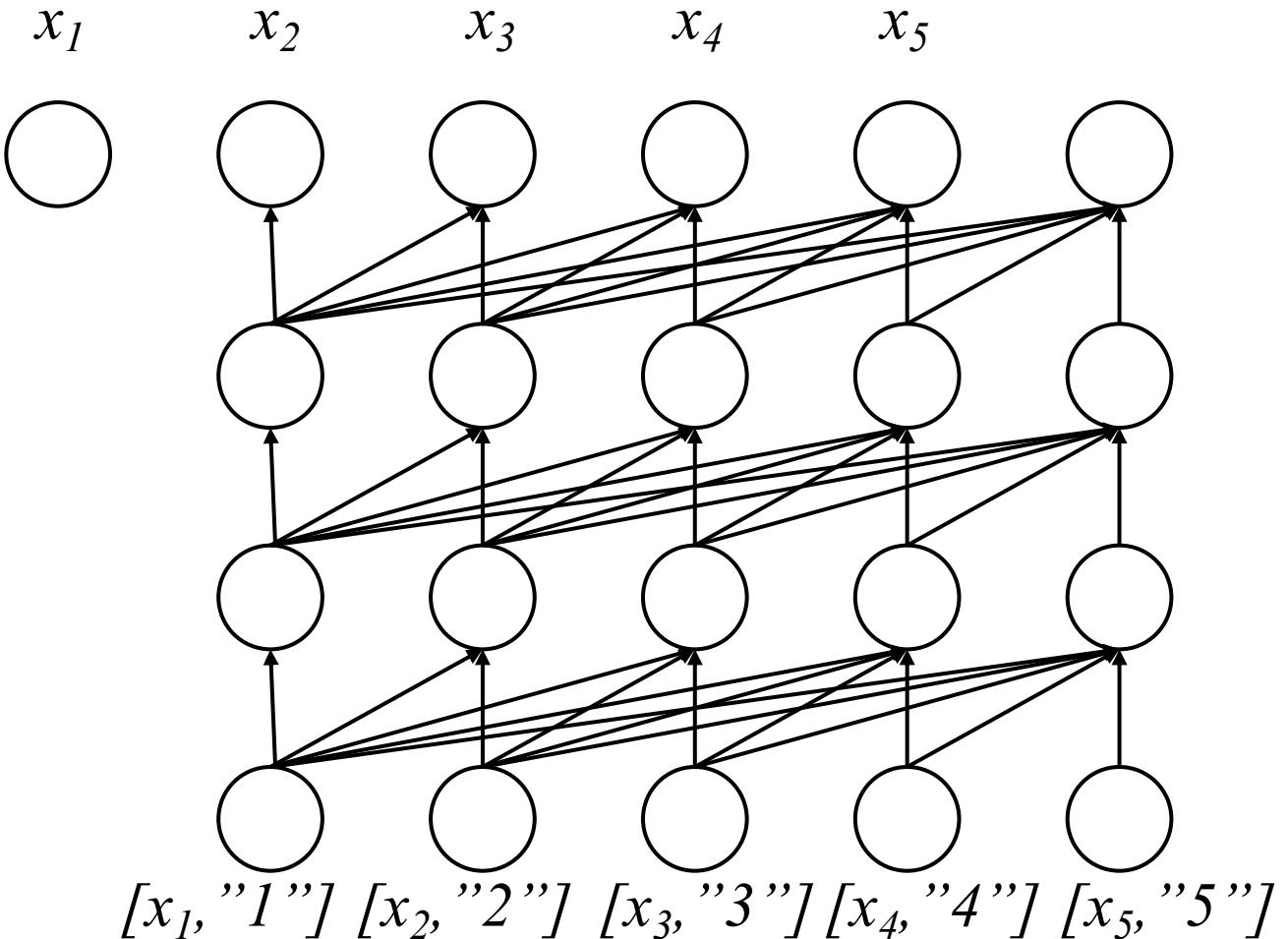
## MADE:

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

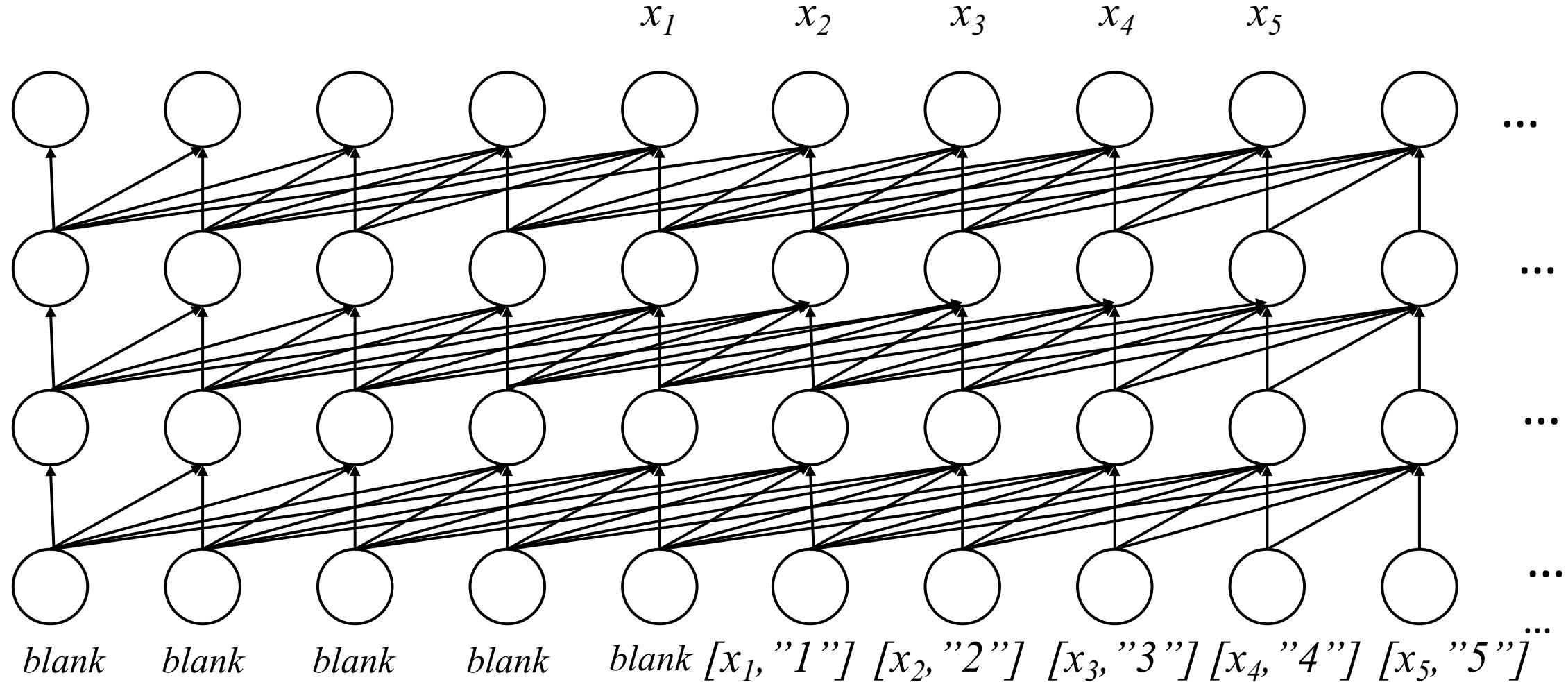


## **Modern Causal Masked Neural Models:**

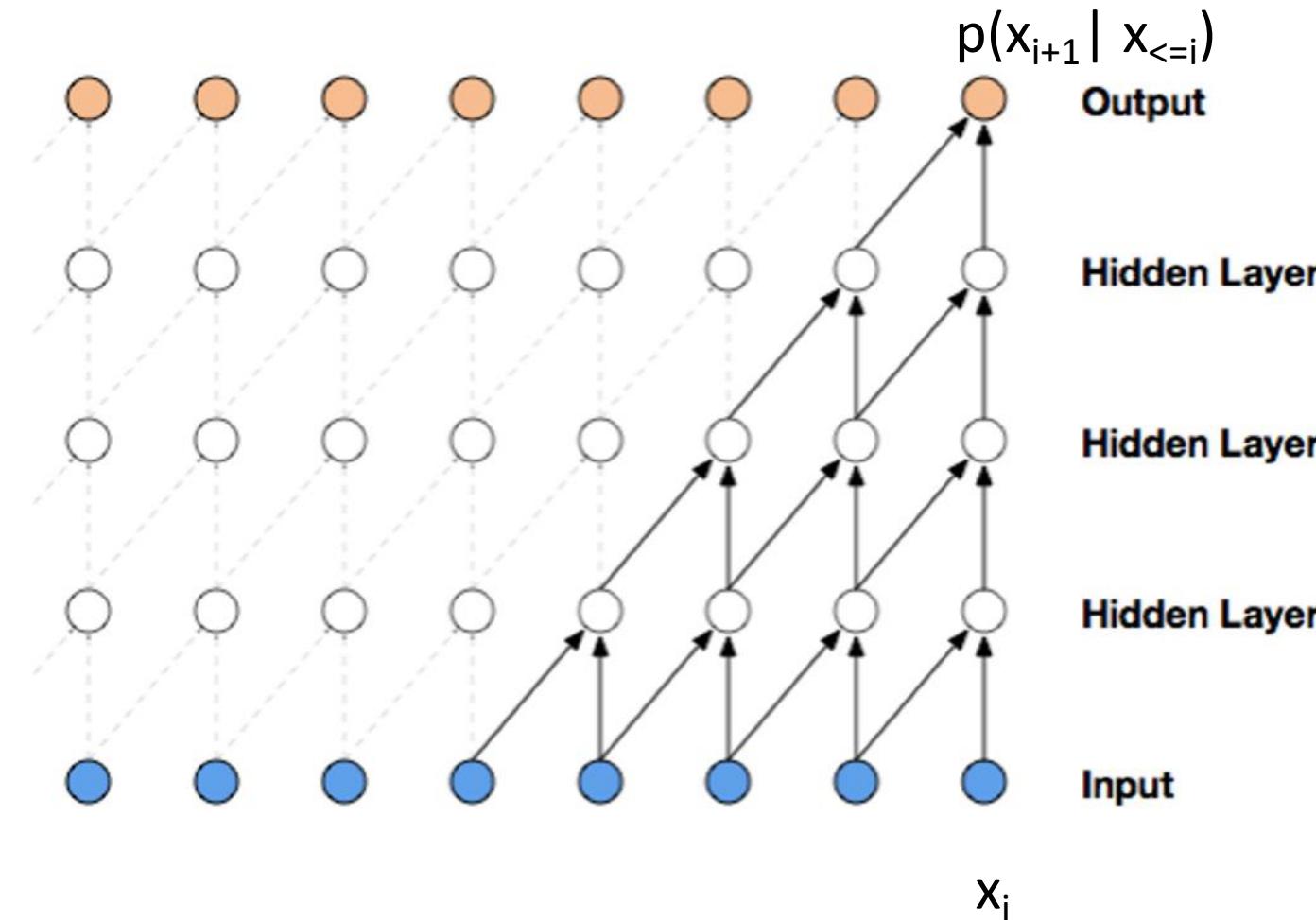
- Every hidden layer node can be vector valued
- Same parameters when shifting from column  $i$  to  $i+1$
- To retain ability to know where we are in sequence: add coordinate coding
- Uniformize fully with padding



# 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

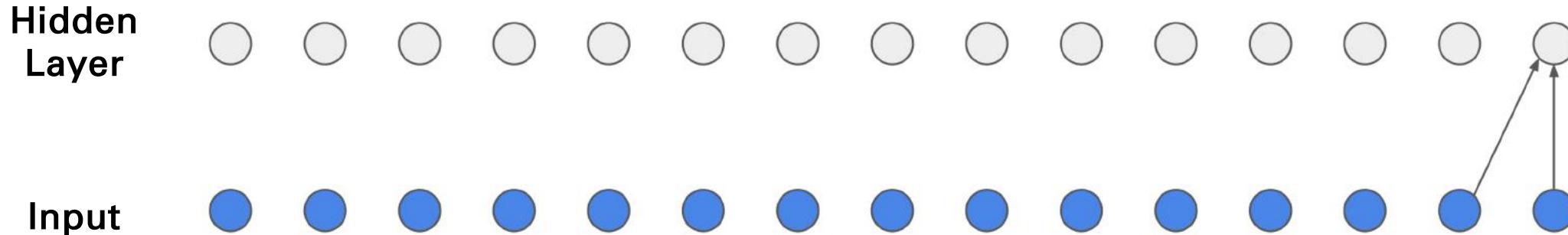
- Limited receptive field, linear in number of layers

# WaveNet – Causal Dilated Convolution

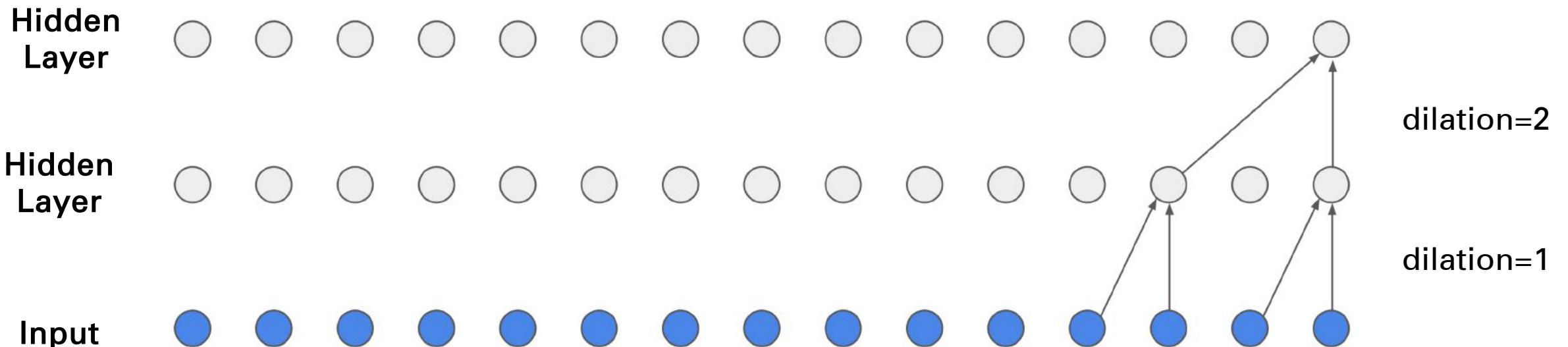
Input



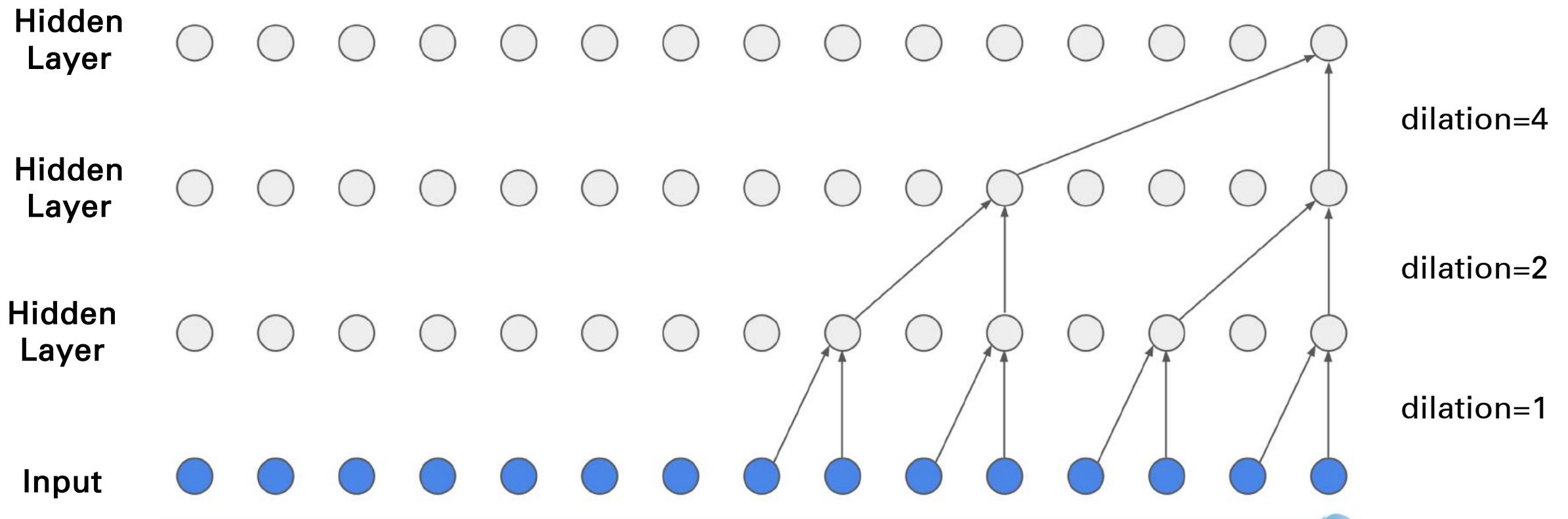
# WaveNet – Causal Dilated Convolution



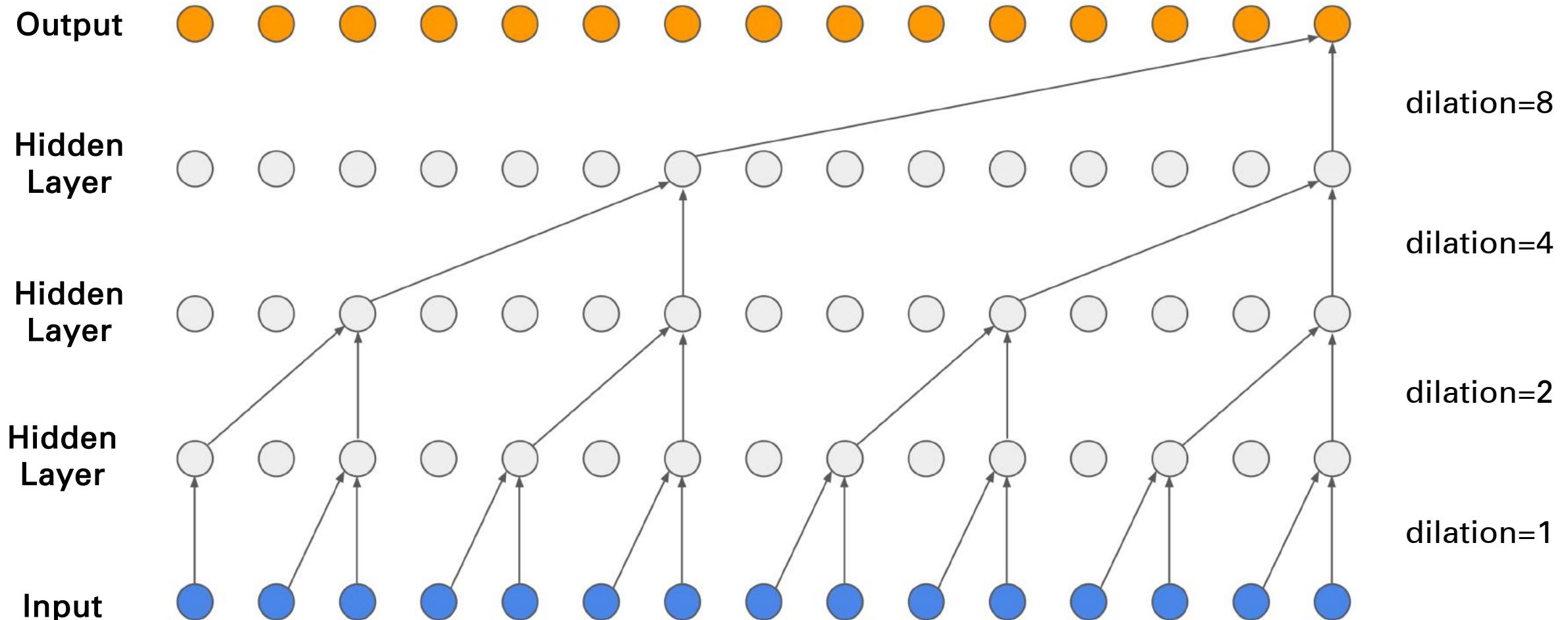
# WaveNet – Causal Dilated Convolution



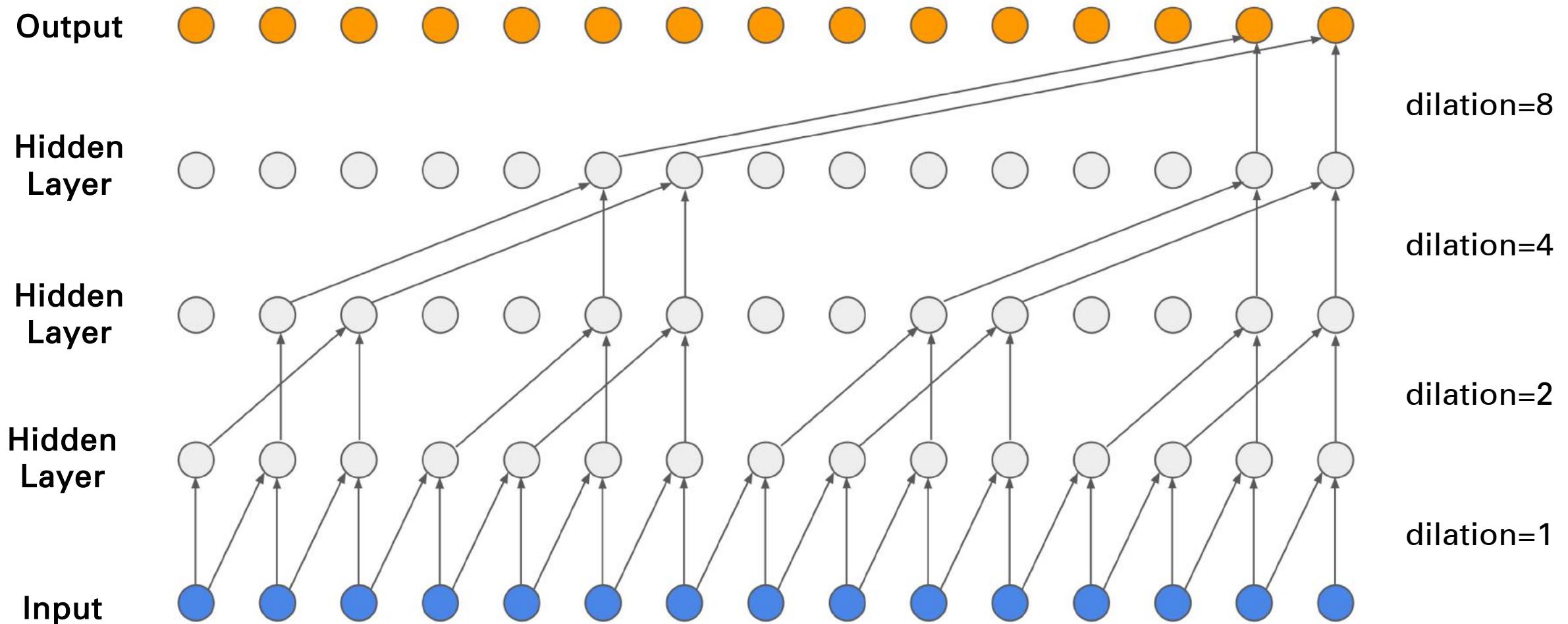
# WaveNet – Causal Dilated Convolution



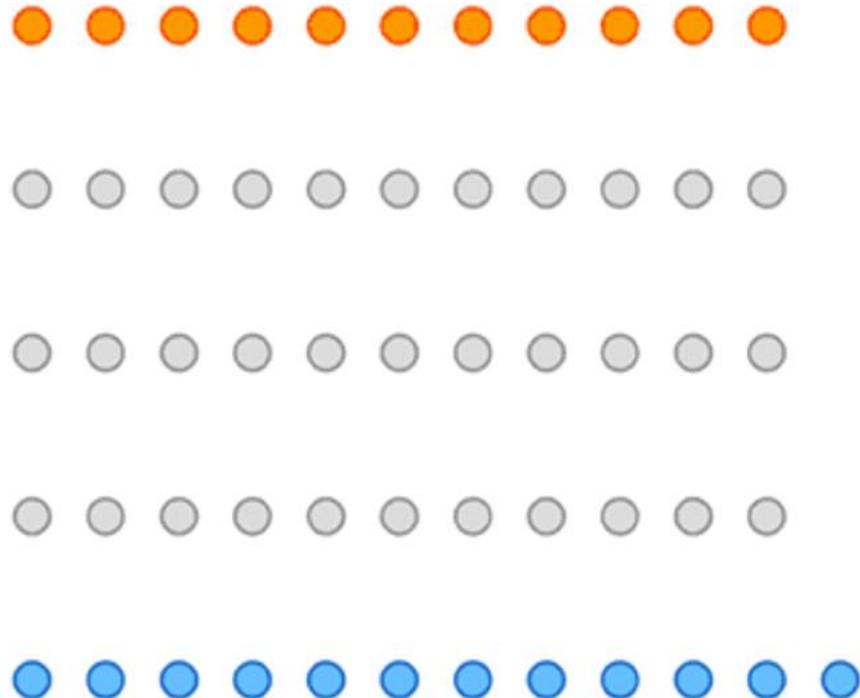
# WaveNet – Causal Dilated Convolution



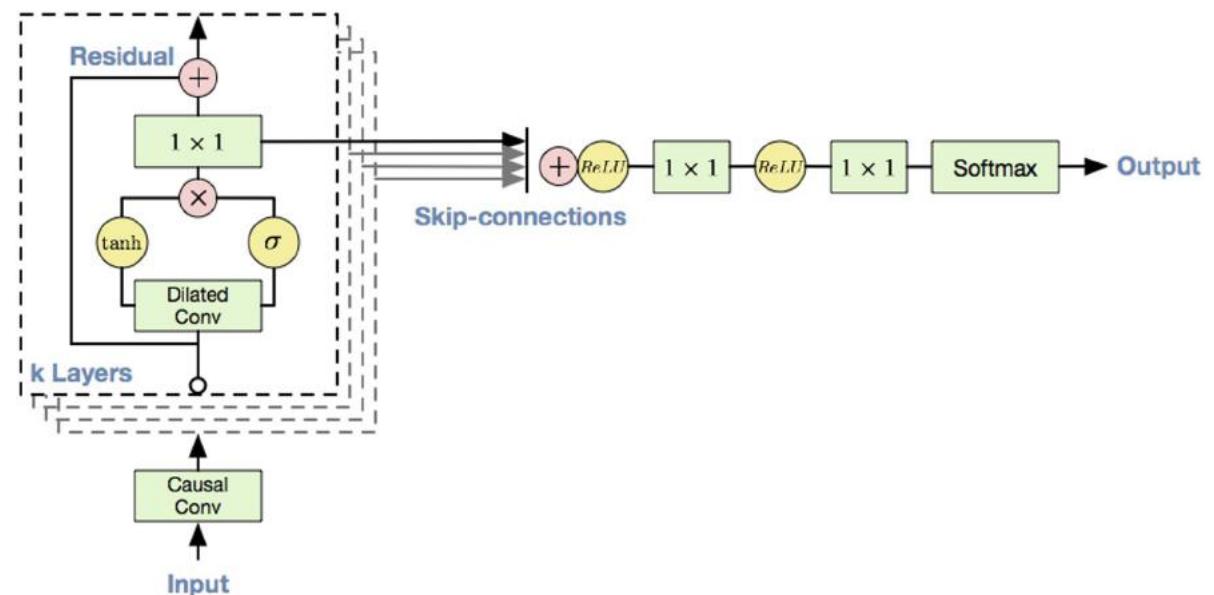
# WaveNet – Causal Dilated Convolution



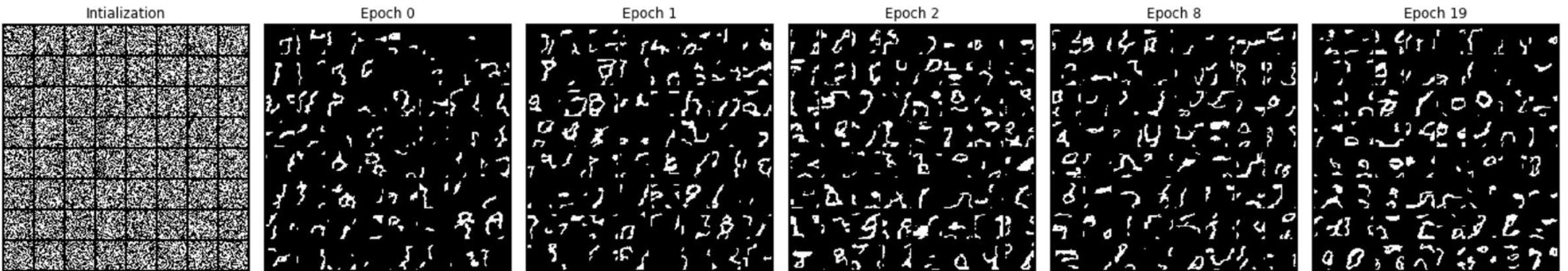
# WaveNet



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

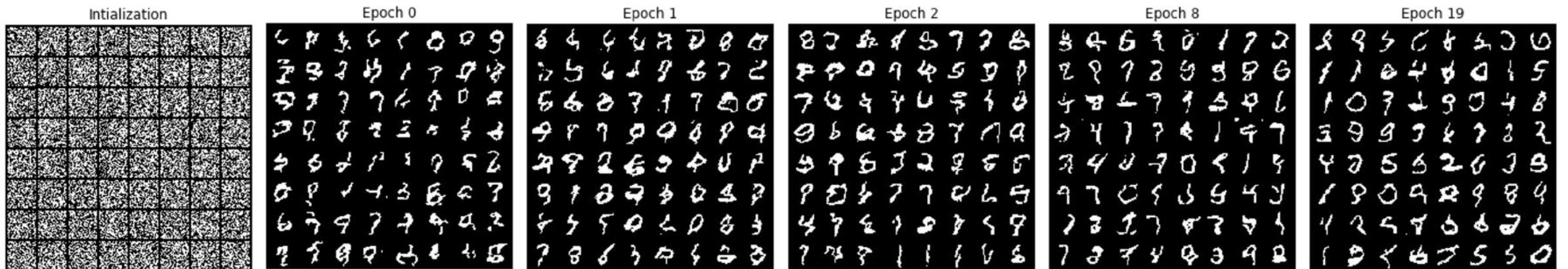


# WaveNet on MNIST



# WaveNet with Pixel Location Appended on MNIST

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



# Masked Temporal (1D) Convolution

```
# More efficient implementation possible by
# padding instead of masking kernels
# k: size of kernel
# kernel: convolution weights
padded_x = tf.pad(x, [
    (0, 0), (k - 1, 0),
    (0, 0), (0, 0)
])
y = tf.nn.conv2d(padded_x, kernel, padding='VALID')
```

# Autoregressive models

- Recall, AutoRegressive Model 
$$p_{\theta}(x) = \prod_{i=1}^d p_{\theta}(x_i | x_{1:i-1})$$
- 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

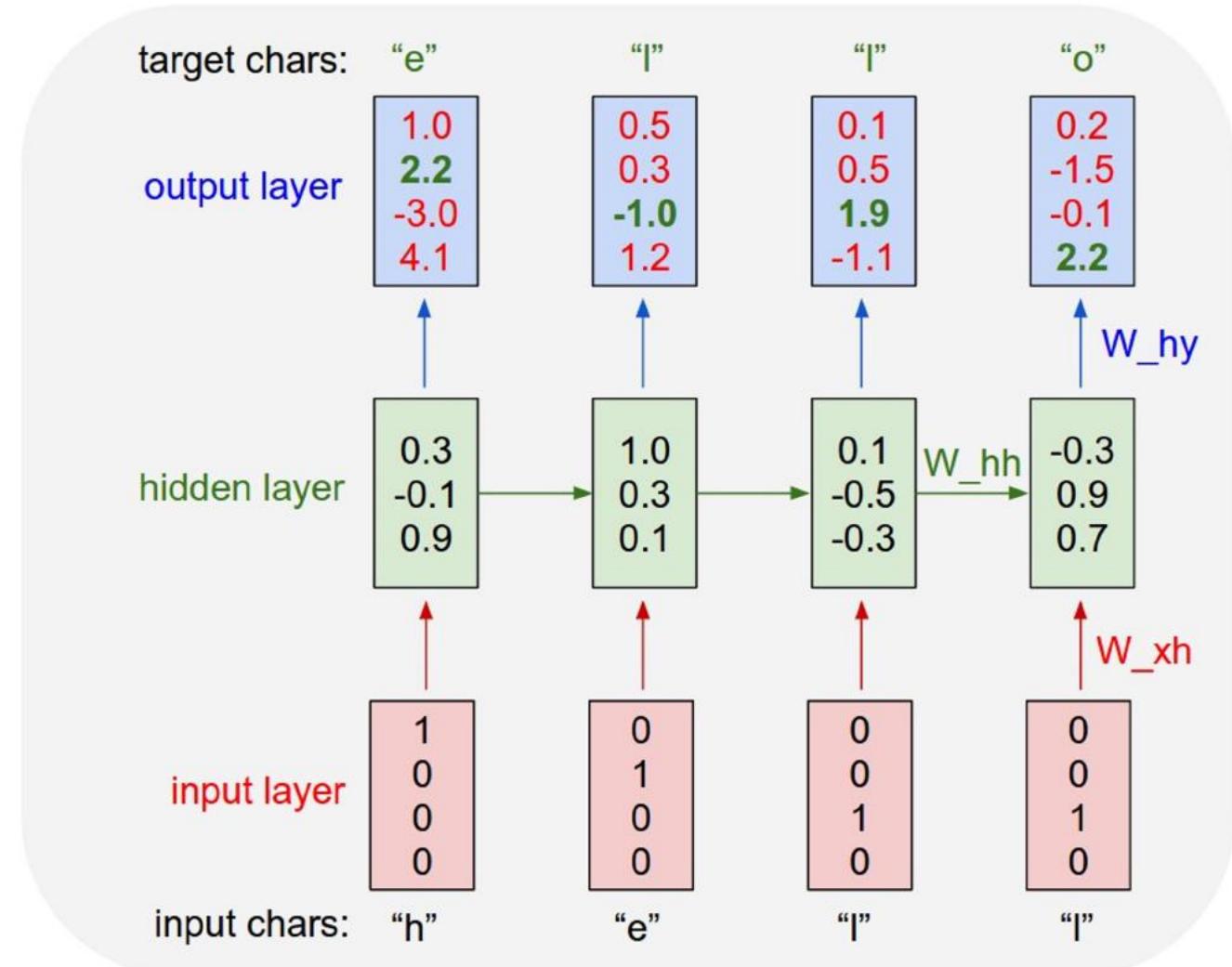
- Recall, AutoRegressive Model 
$$p_{\theta}(x) = \prod_{i=1}^d p_{\theta}(x_i | x_{1:i-1})$$
- How to achieve efficiently representable and learnable parameterizations of the conditionals?
  - **Solution 1: Bayes' Nets** – sparsify conditioning
    - Efficient, but: too strong an assumption in most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

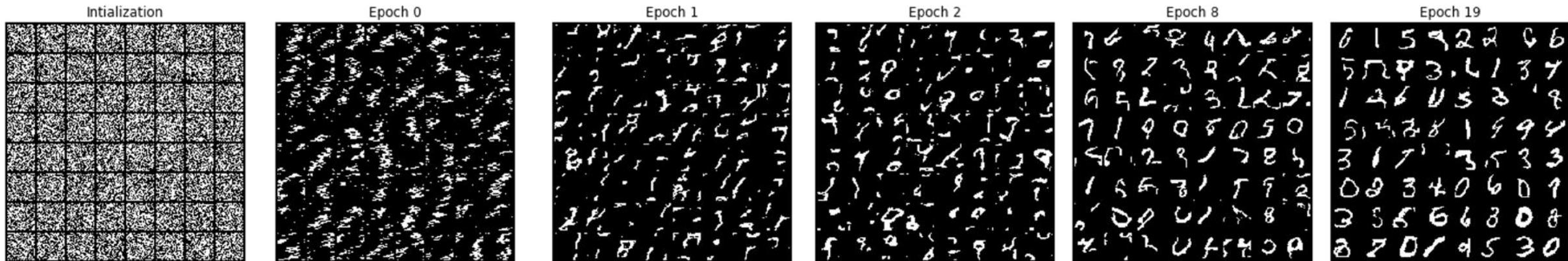
$$\log p(\mathbf{x}) = \sum_{i=1}^d \log p(x_i | \mathbf{x}_{1:i-1})$$

Sequence of characters

Character at  $i^{\text{th}}$  position



# RNN on MNIST



# RNN with Pixel Location Appended on MNIST

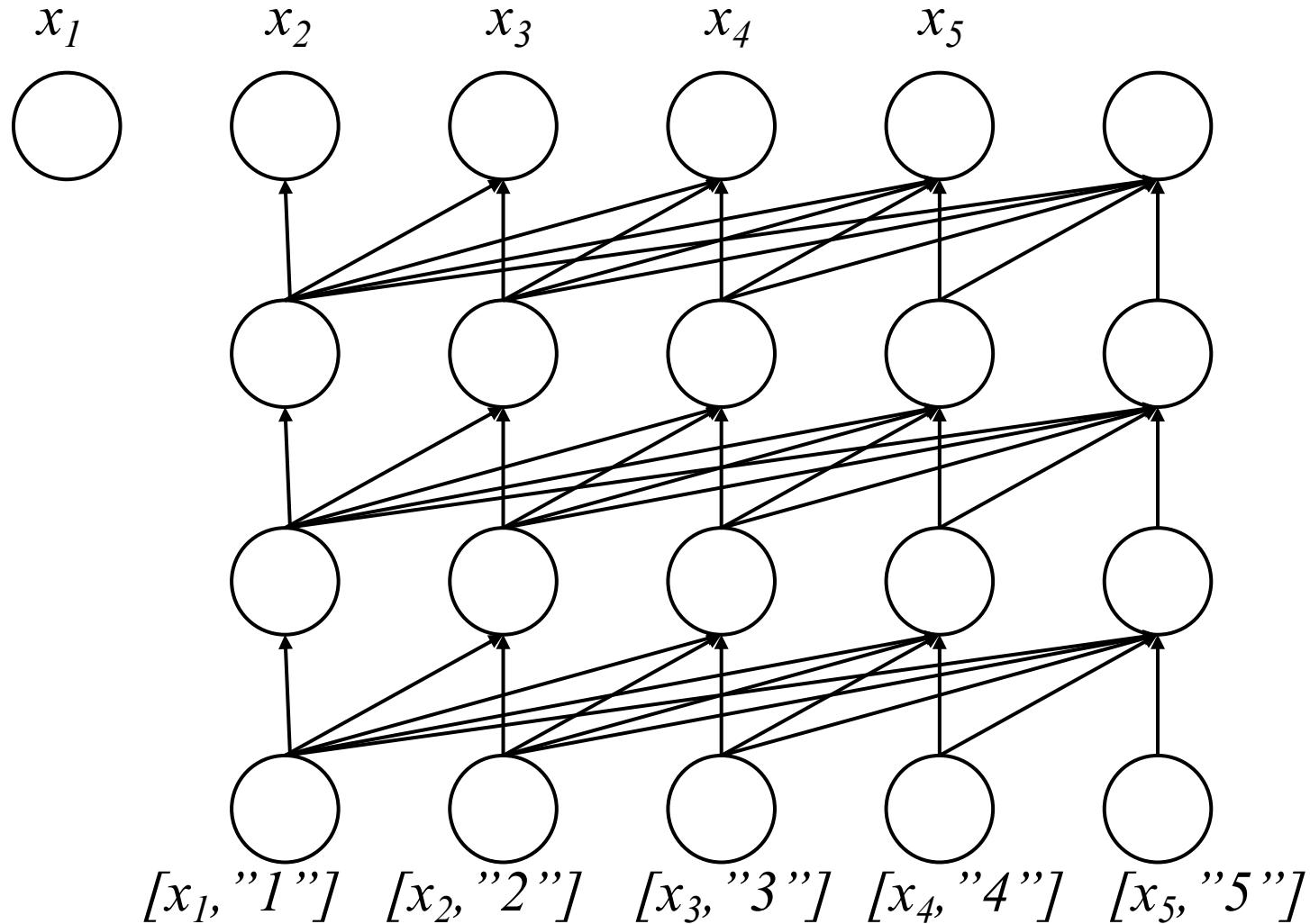
- Append (x,y) coordinates of pixel in the image as input to RNN



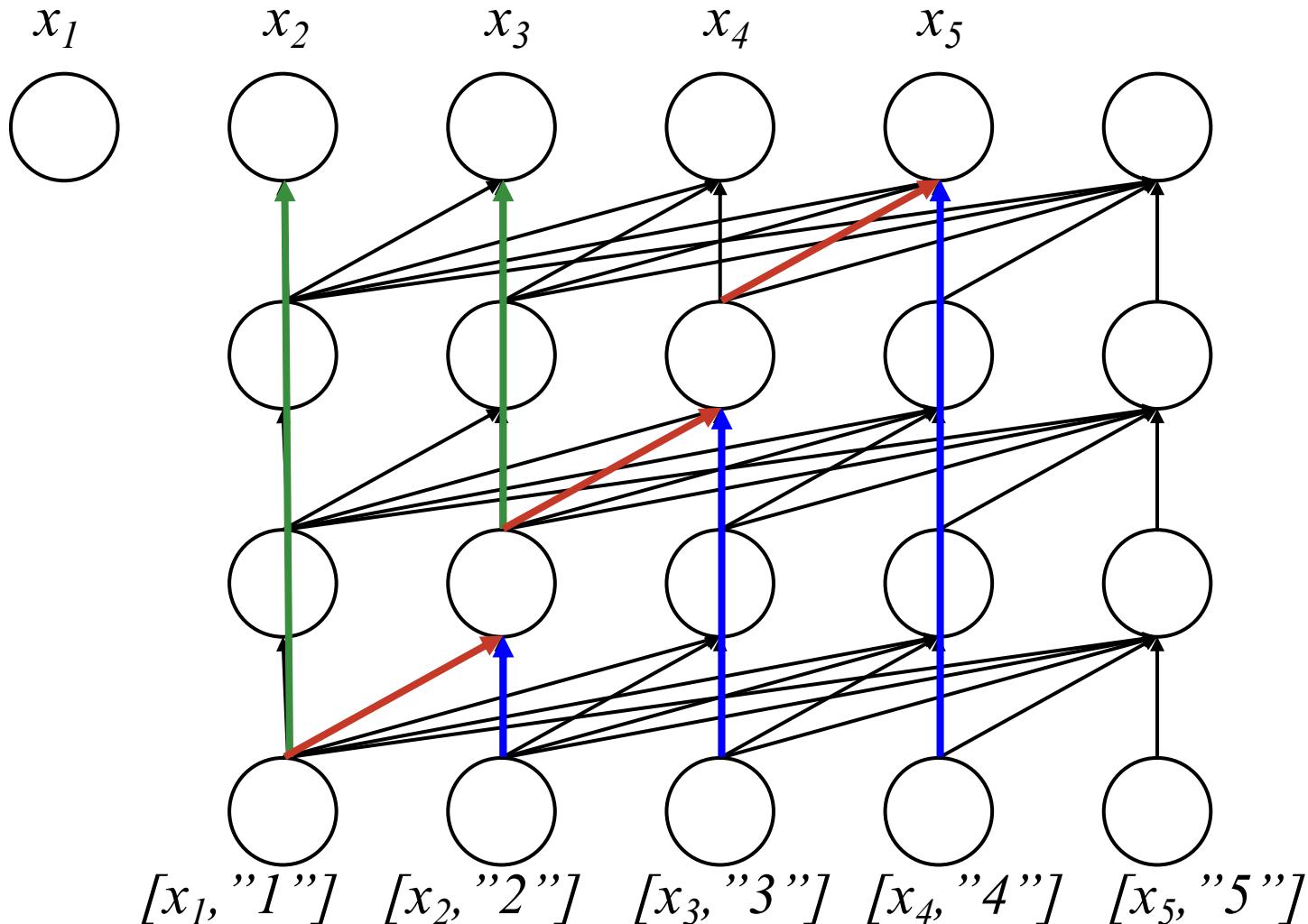
# Autoregressive models

- Recall, AutoRegressive Model 
$$p_{\theta}(x) = \prod_{i=1}^d p_{\theta}(x_i | x_{1:i-1})$$
- 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

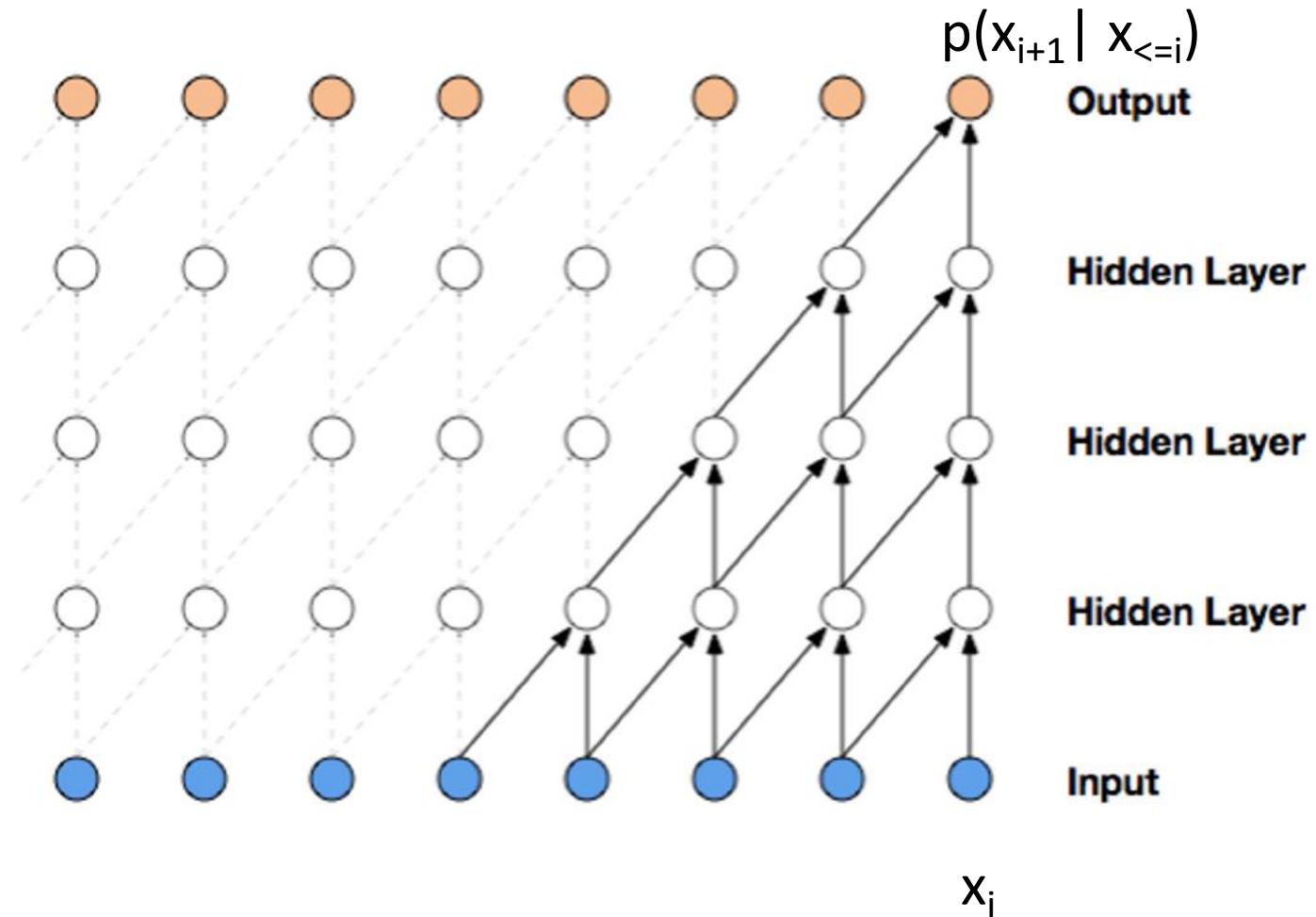
# Lecture overview

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

# Lecture overview

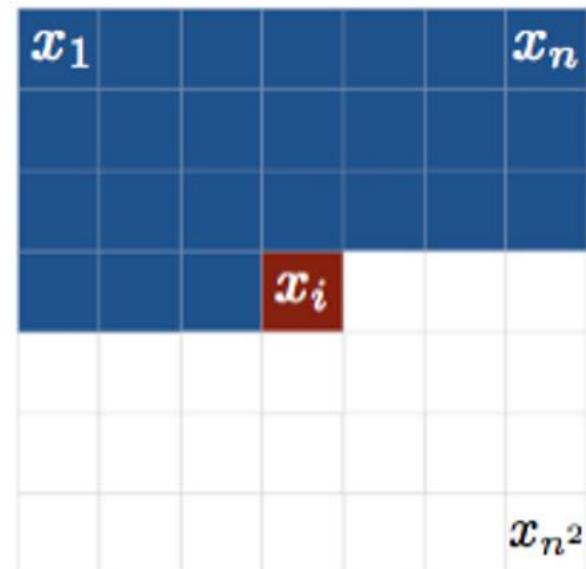
- motivation
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
  - convolutional
  - attention
  - tokenization
  - caching
- other things to be aware of

# 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:

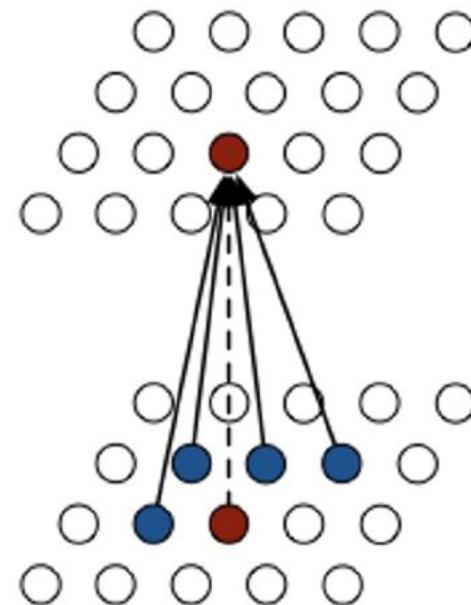


This is called raster scan ordering.  
(Different orderings are possible,  
more on this later)

# PixelCNN

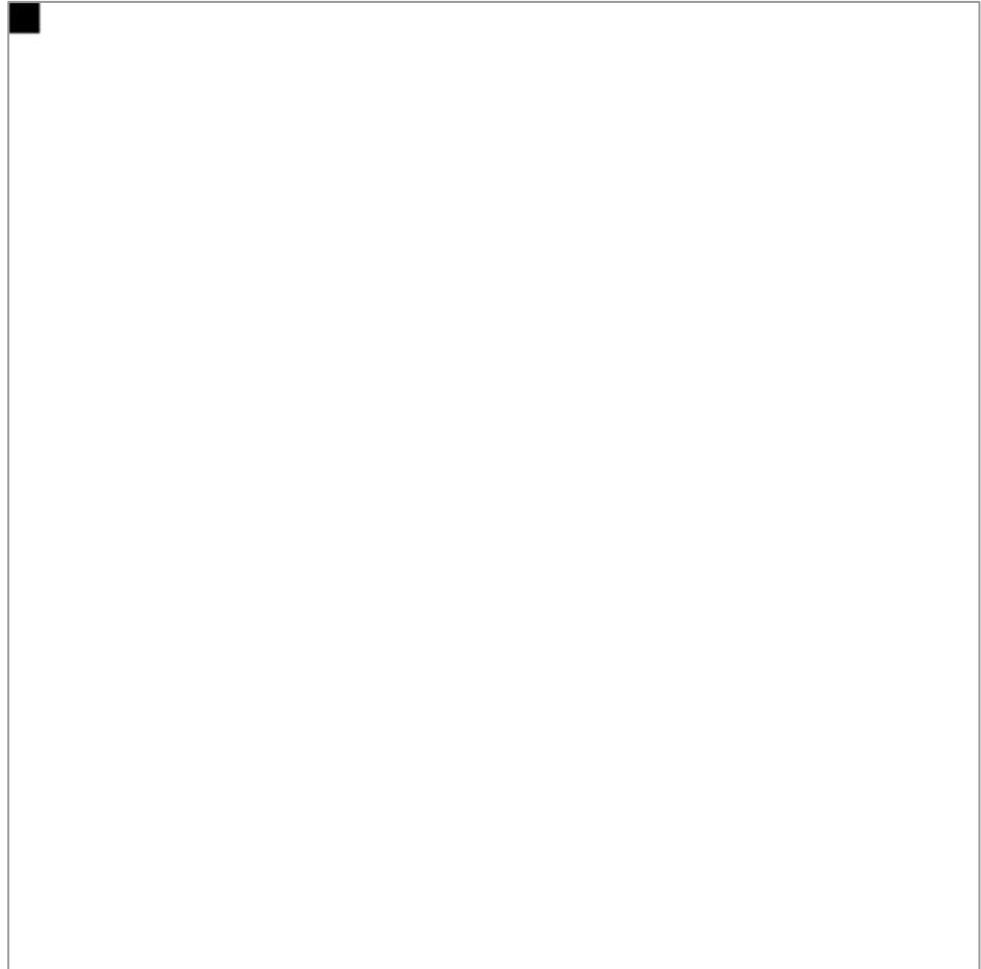
- Design question: how to design a masking method to obey that ordering?
- One possibility: PixelCNN (2016)

1	1	1
1	0	0
0	0	0

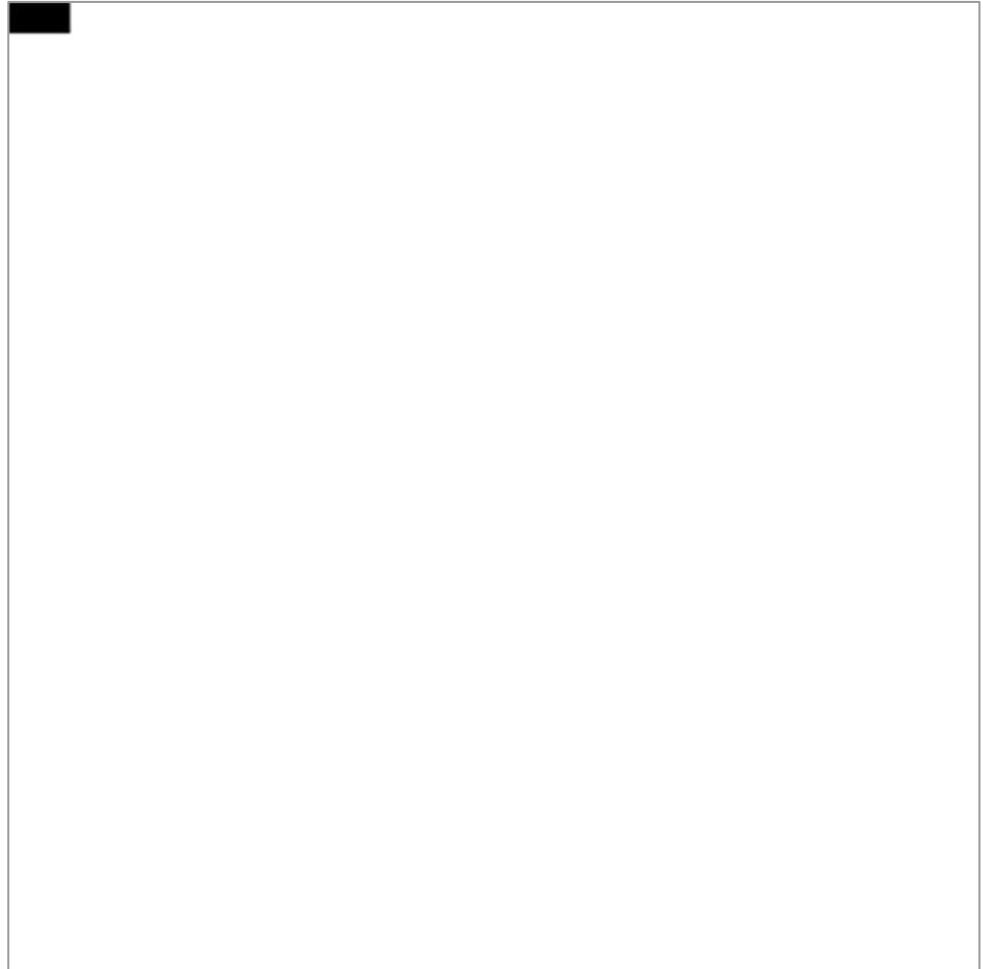


PixelCNN

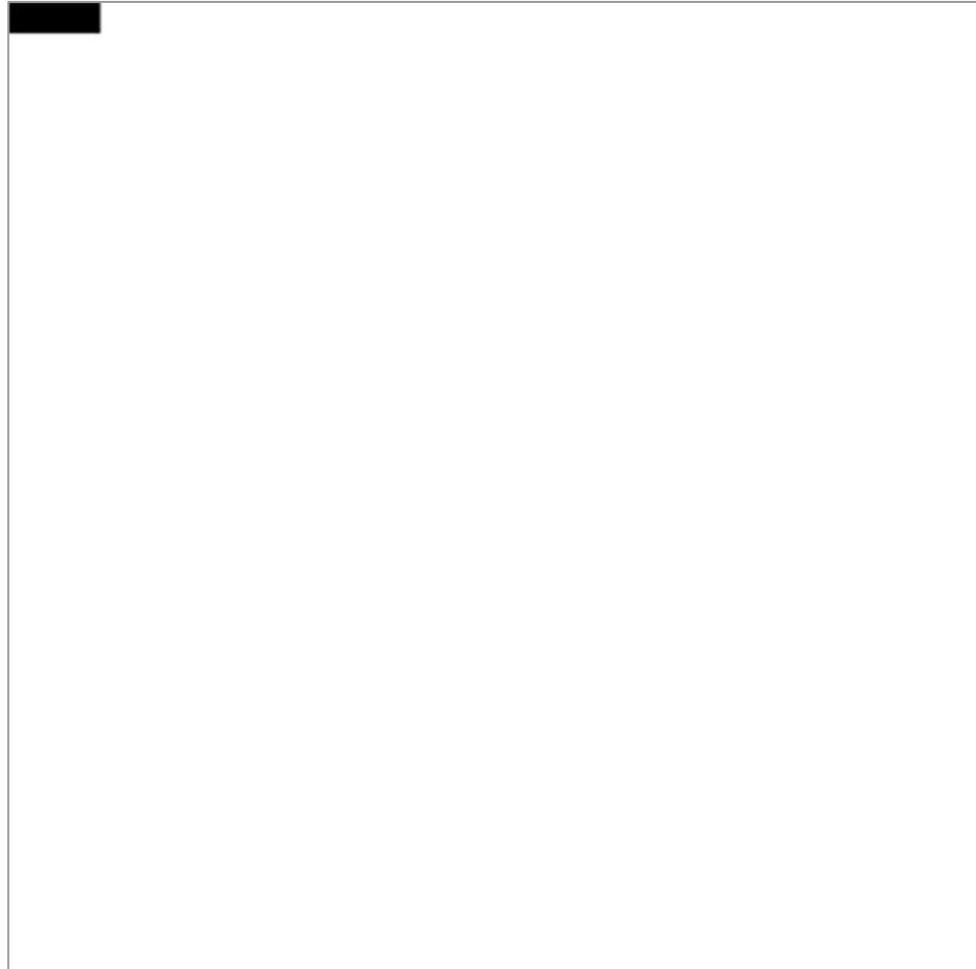
# Softmax Sampling



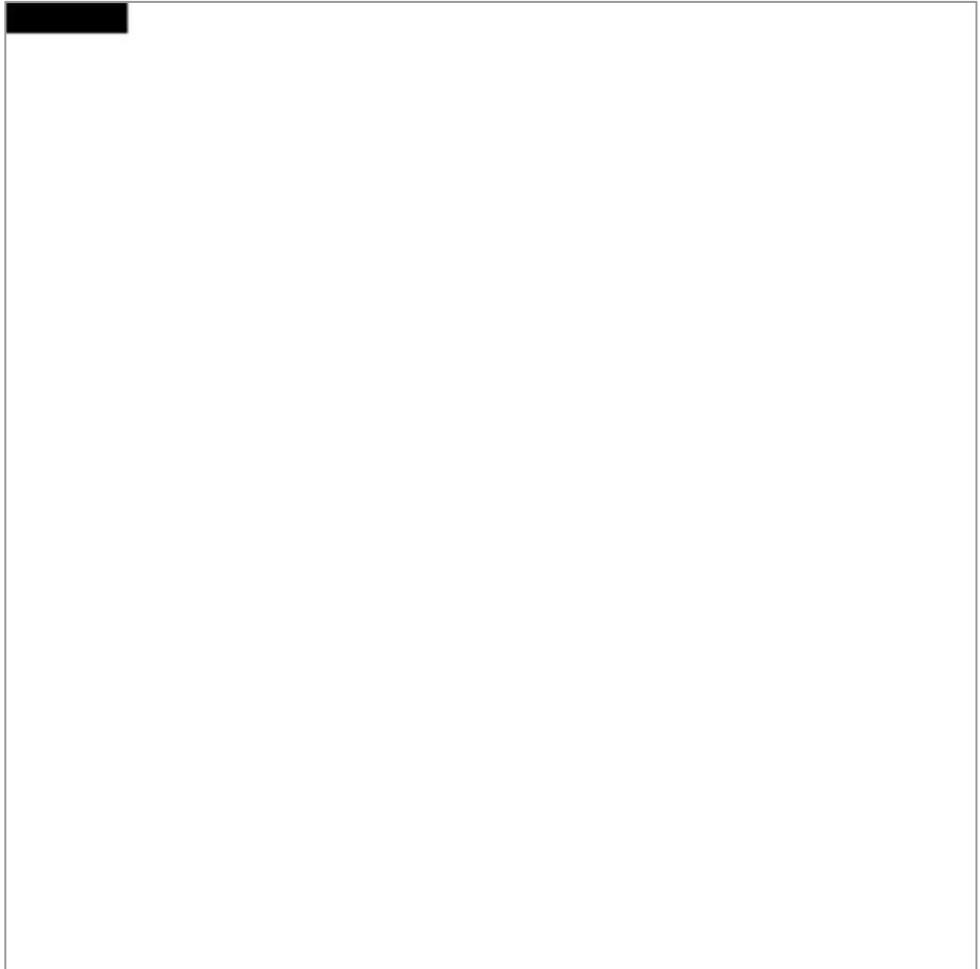
# Softmax Sampling



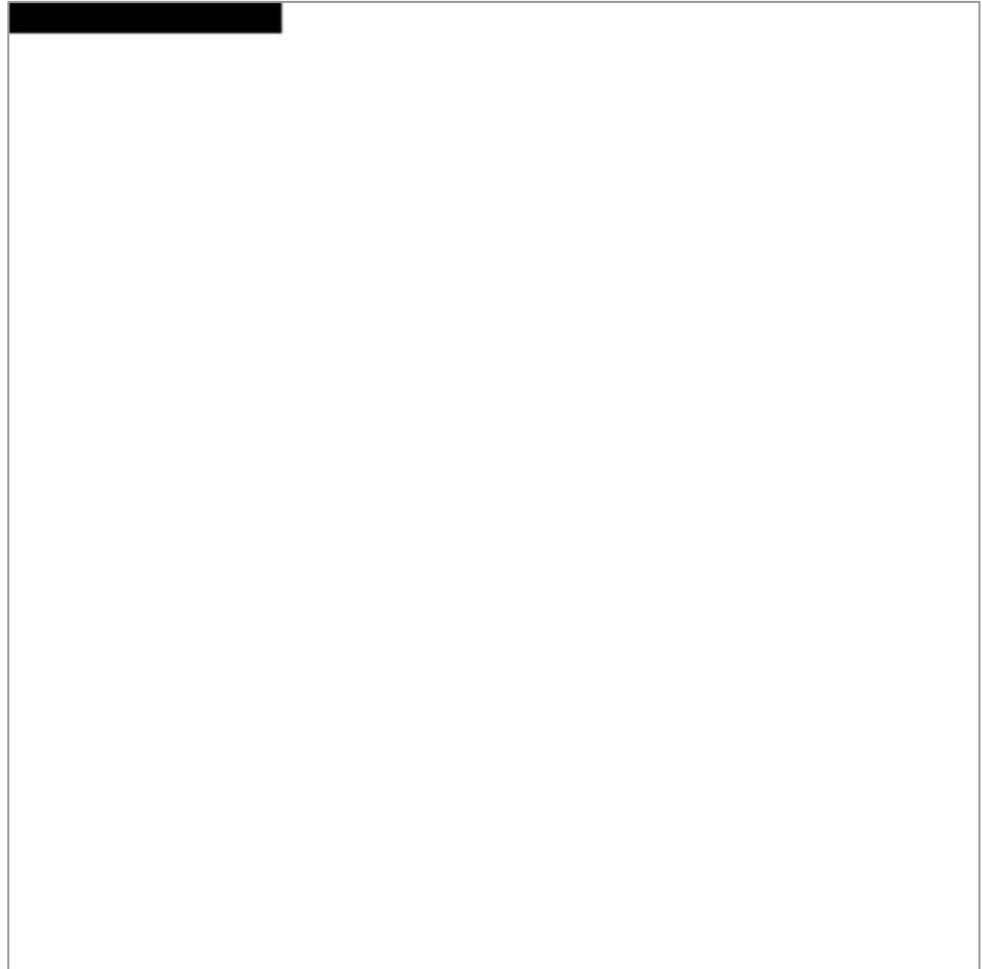
# Softmax Sampling



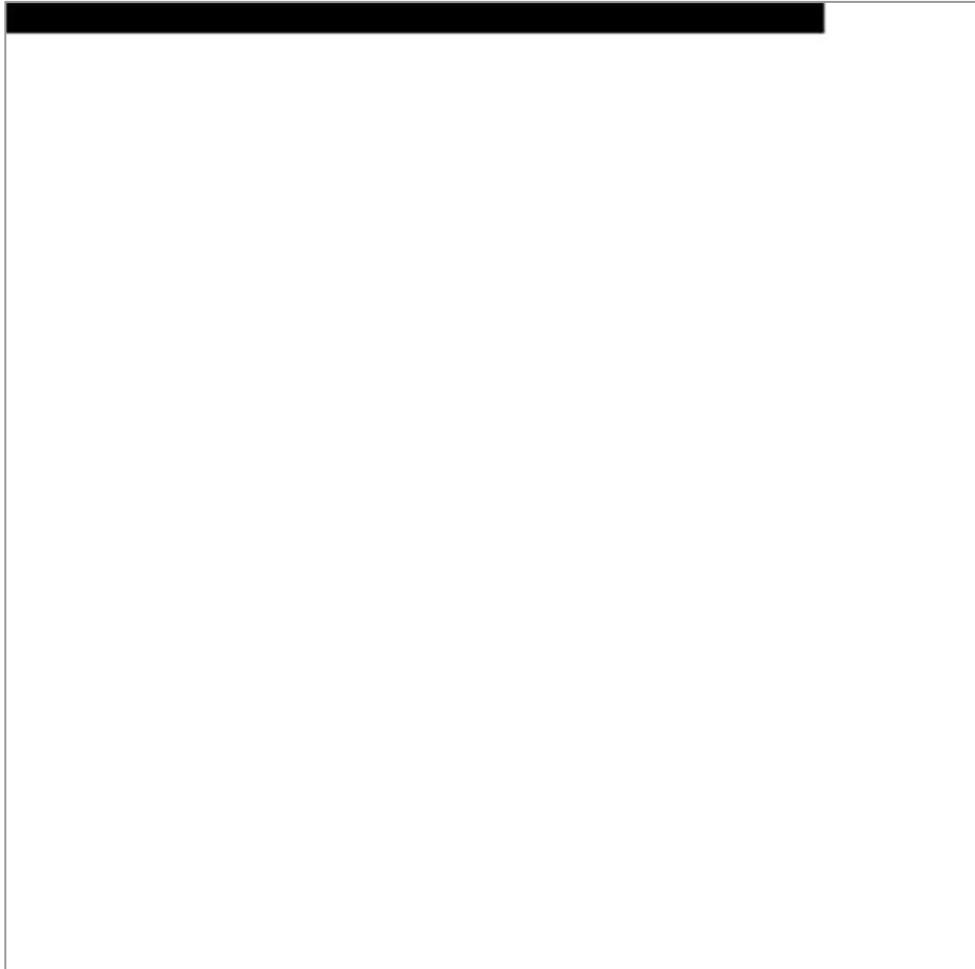
# Softmax Sampling



# Softmax Sampling



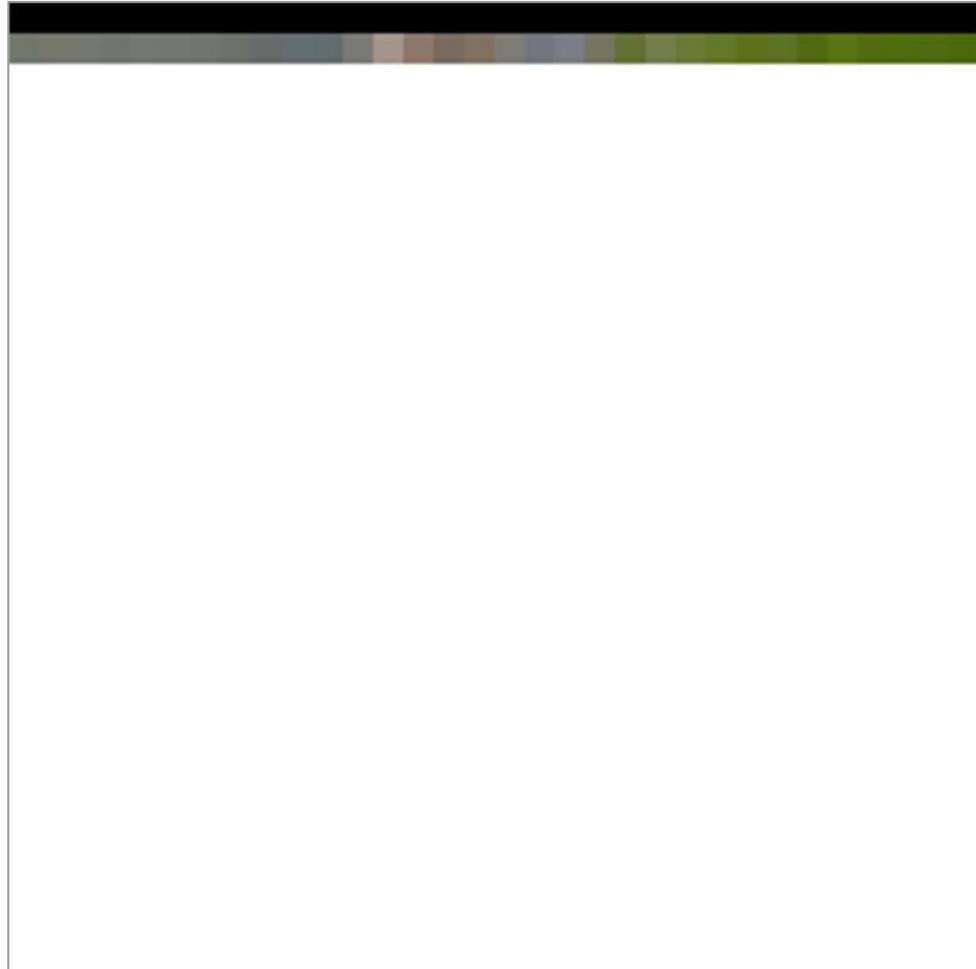
# Softmax Sampling



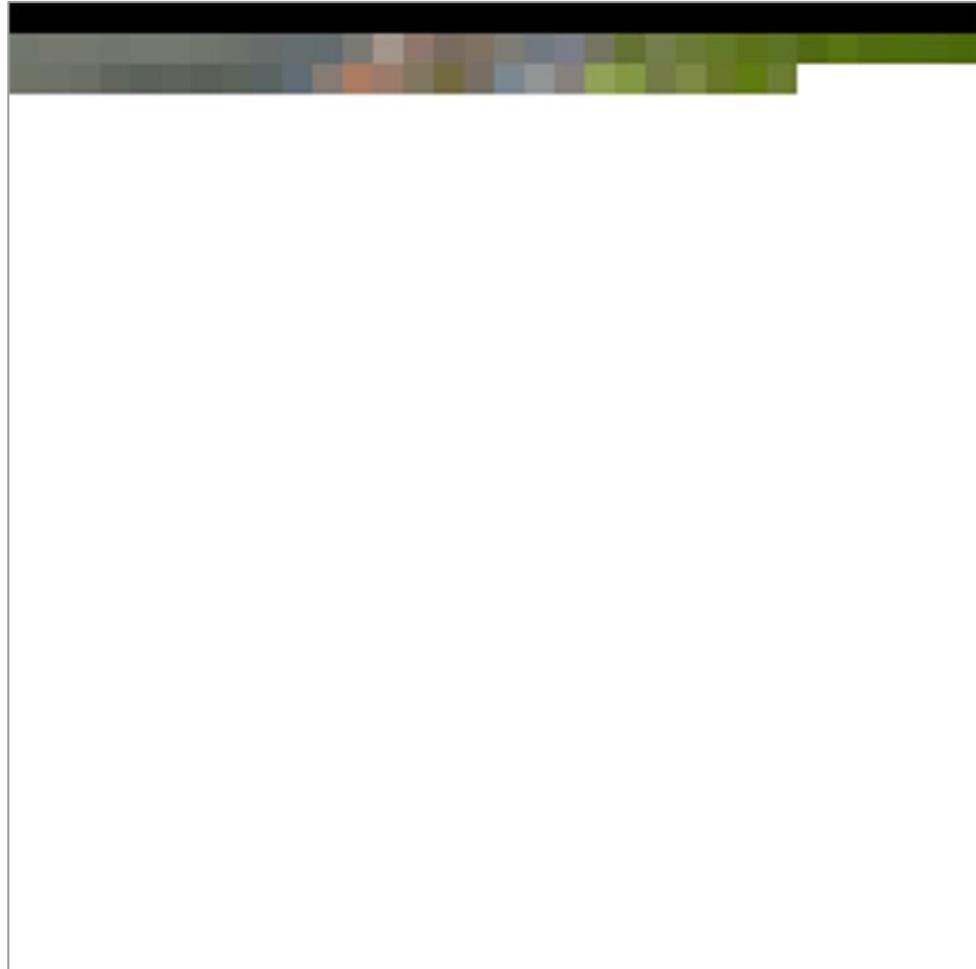
# Softmax Sampling



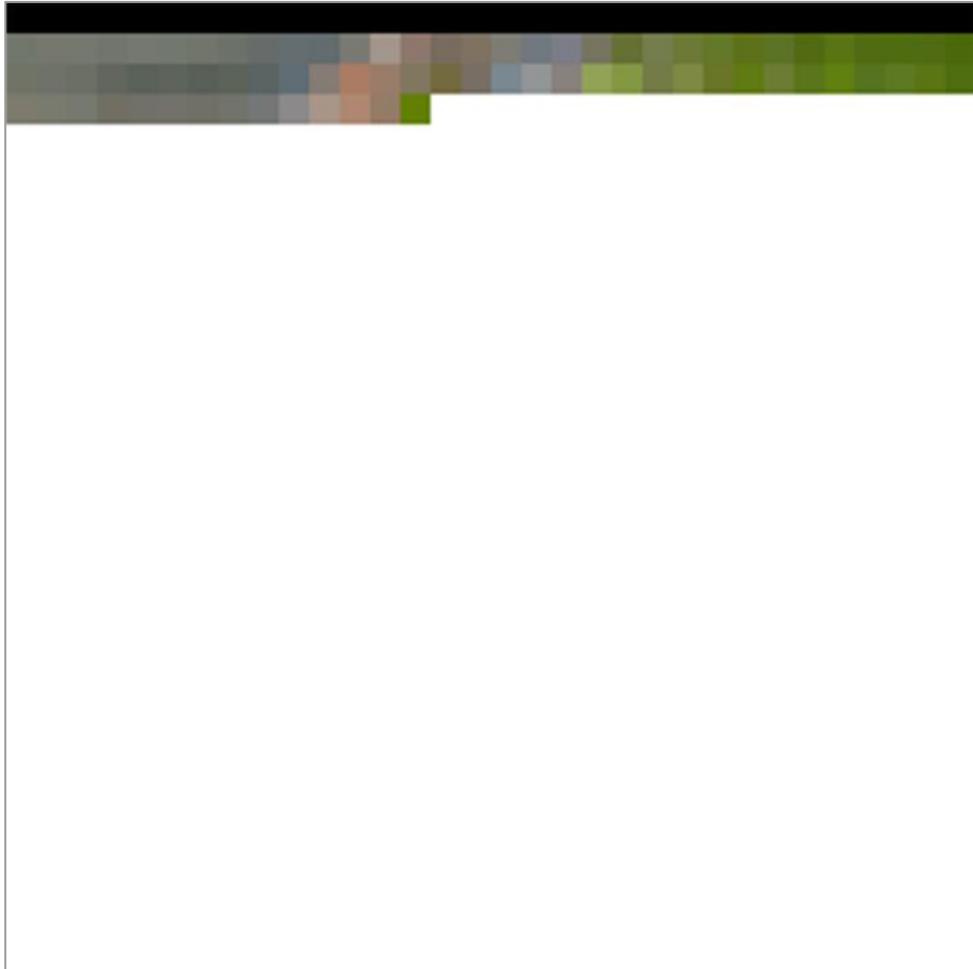
# Softmax Sampling



# Softmax Sampling



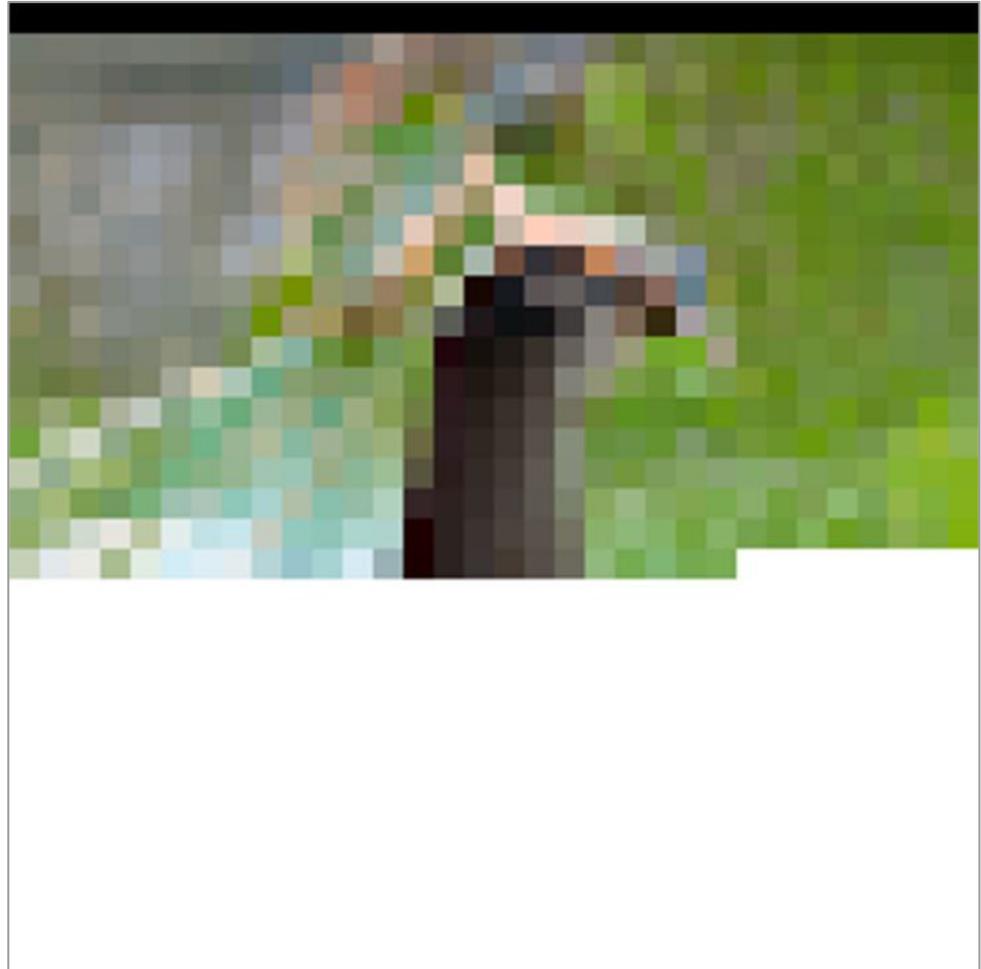
# Softmax Sampling



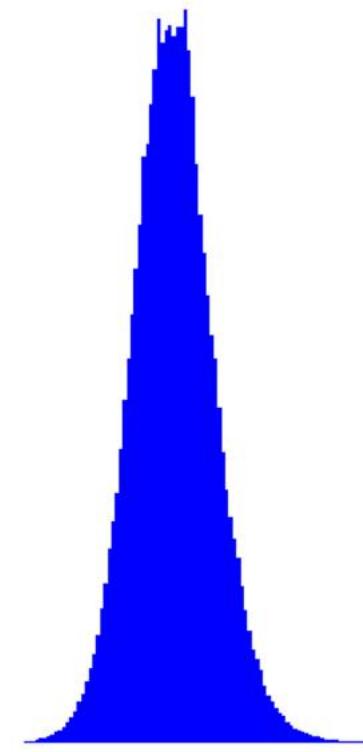
# Softmax Sampling



# Softmax Sampling

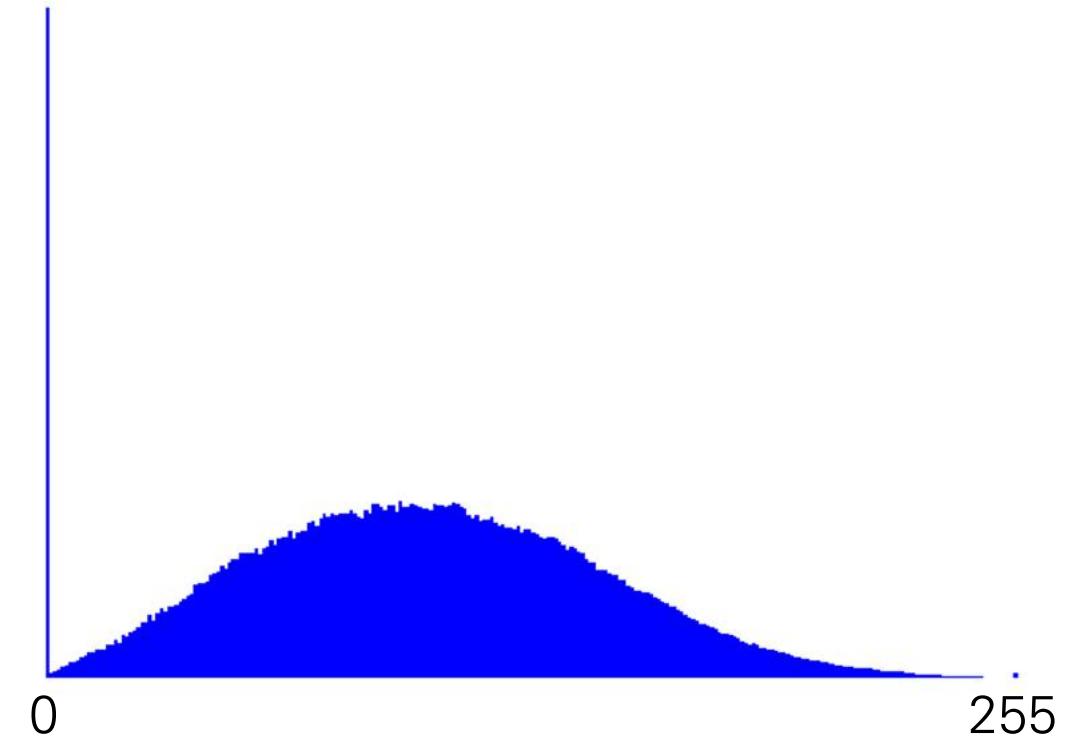
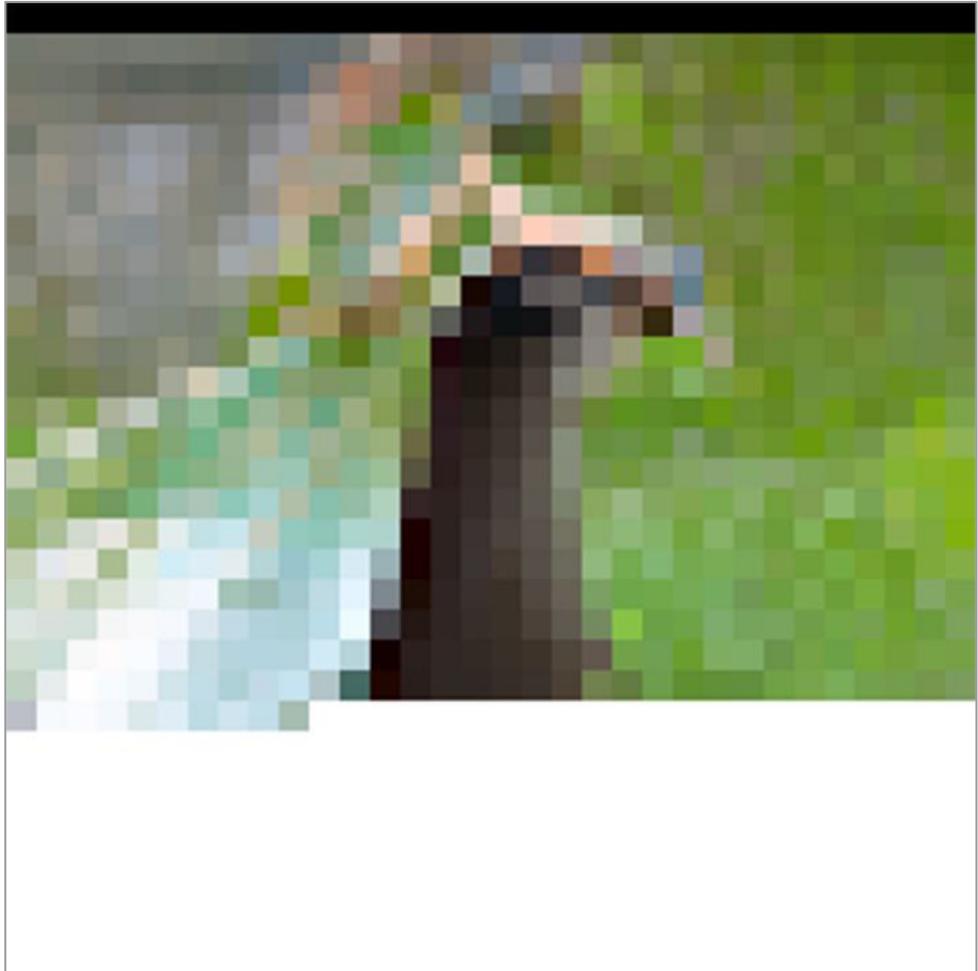


0

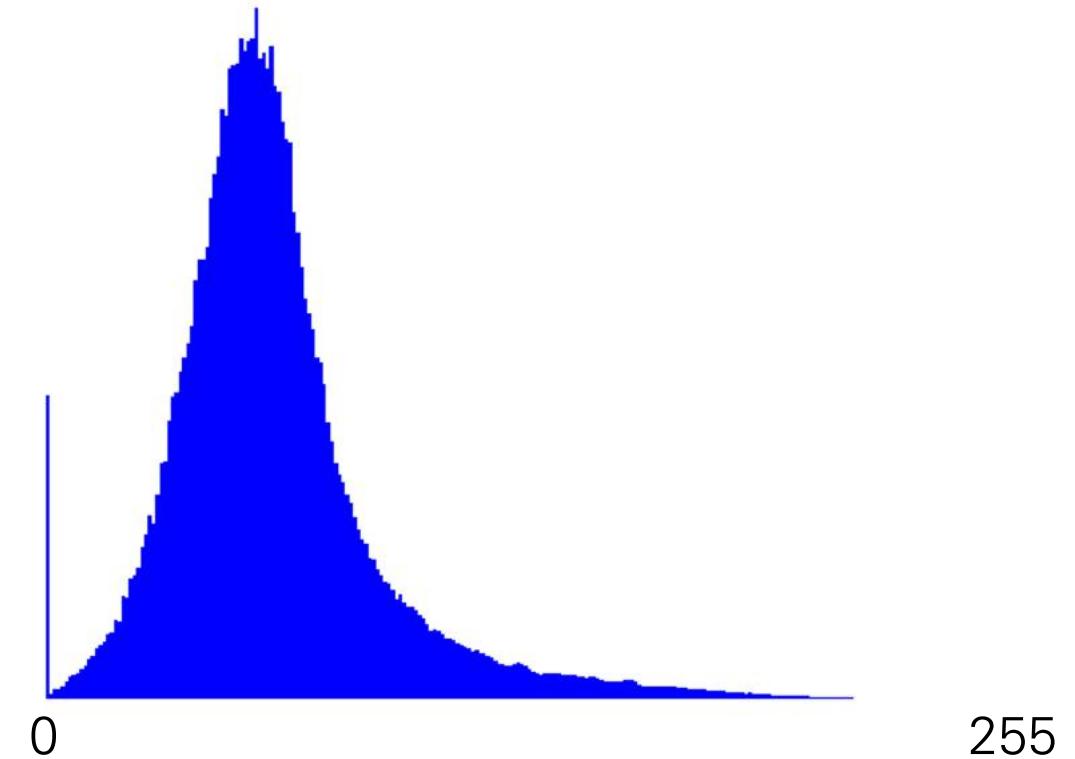
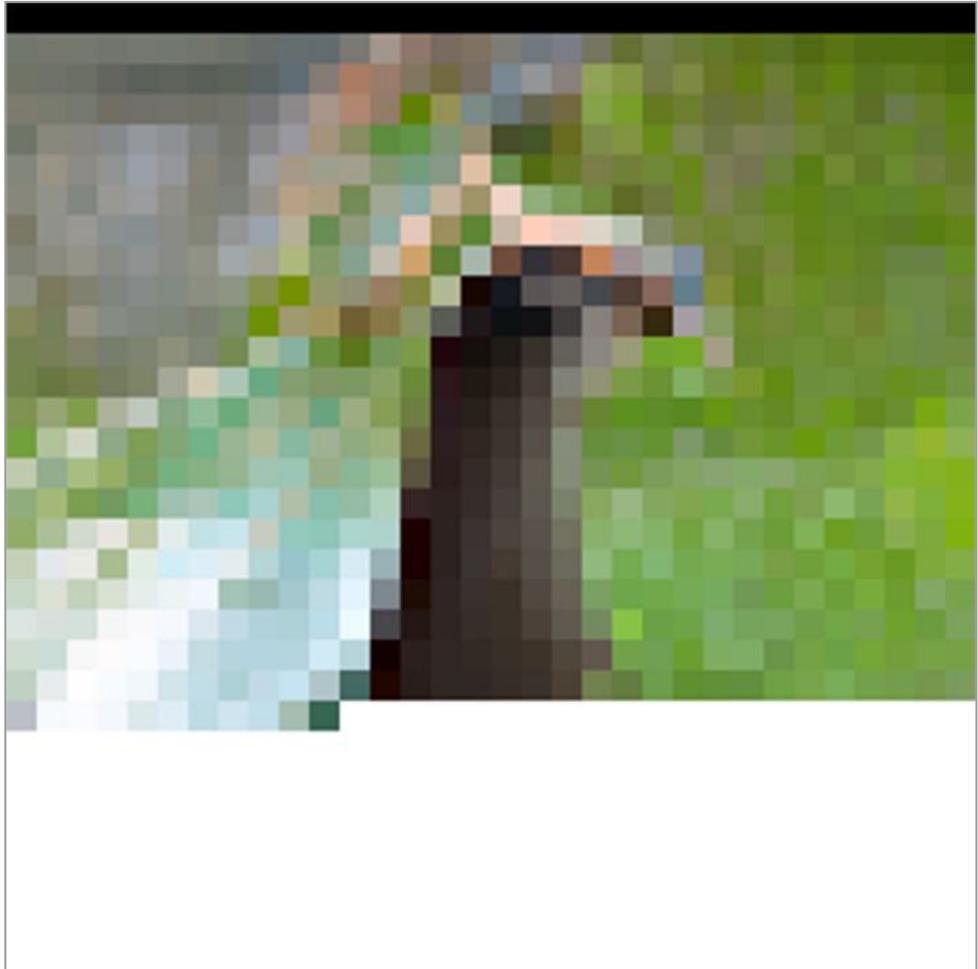


255

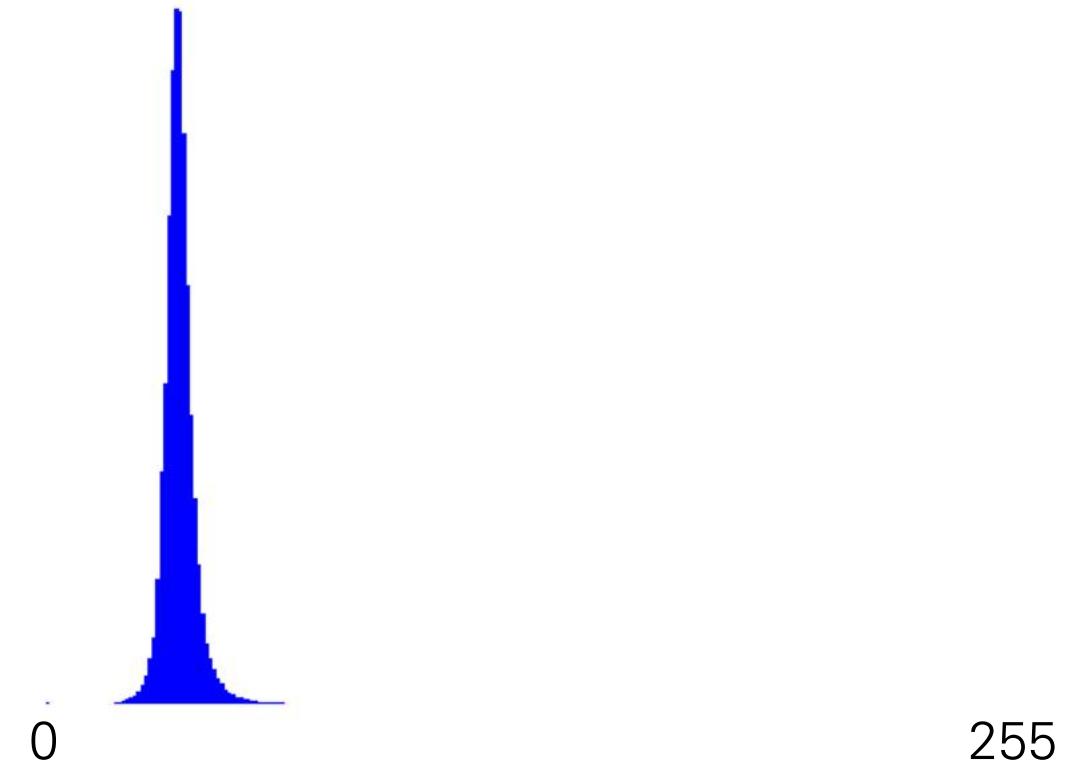
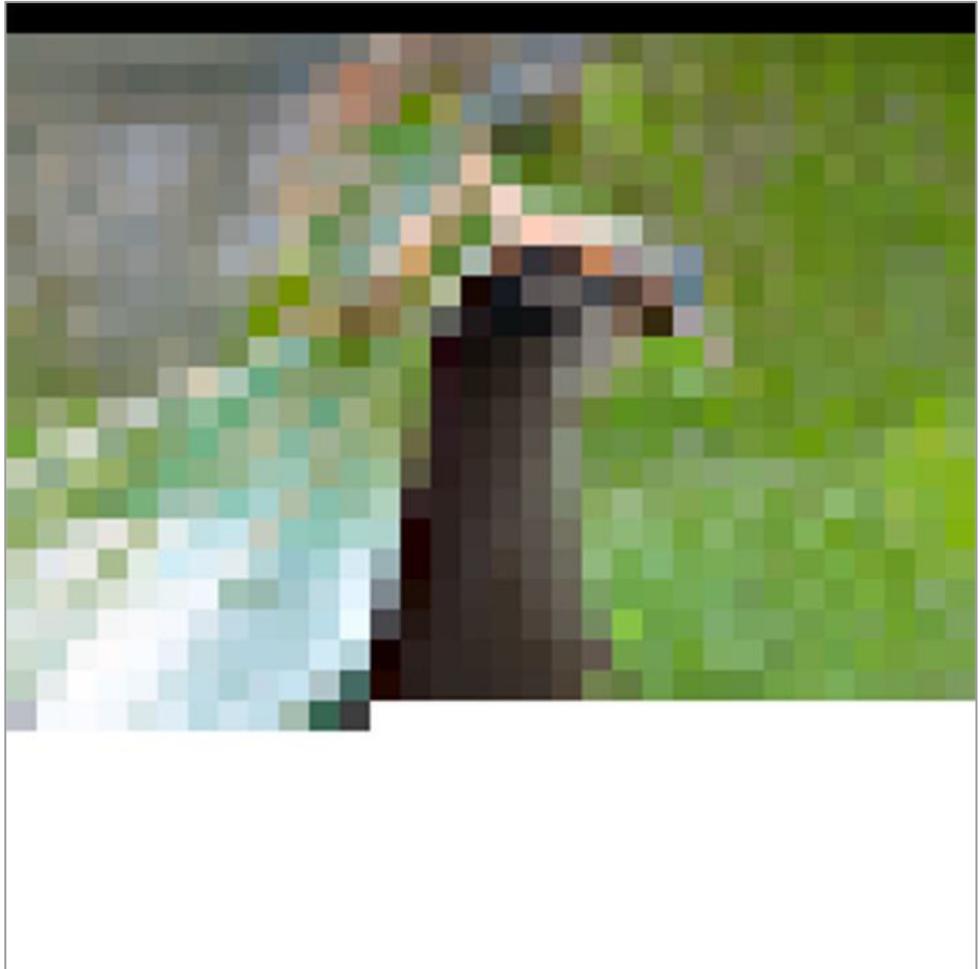
# Softmax Sampling



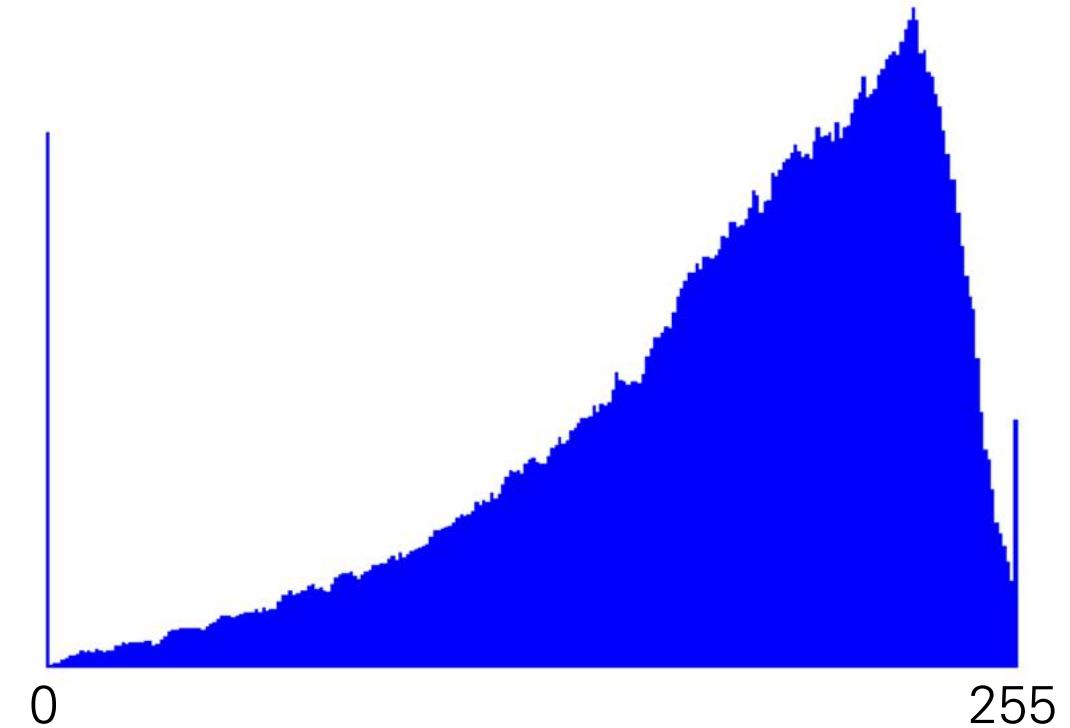
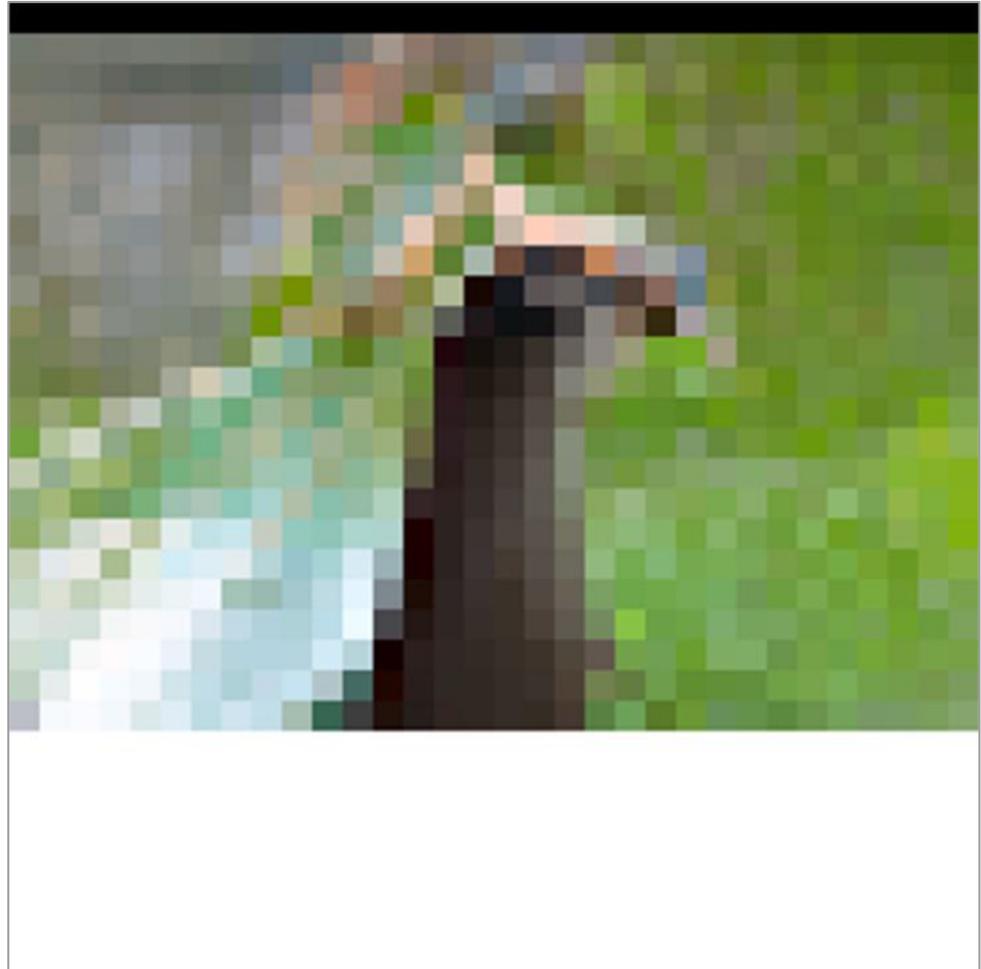
# Softmax Sampling



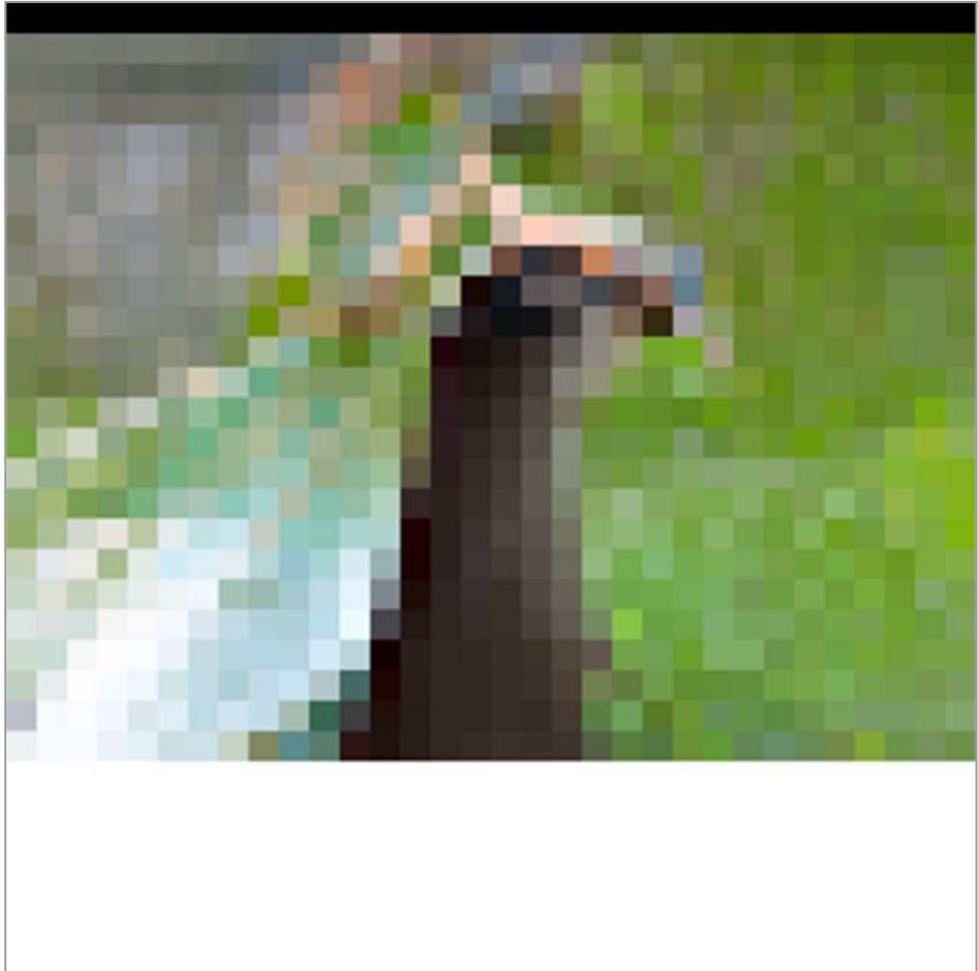
# Softmax Sampling



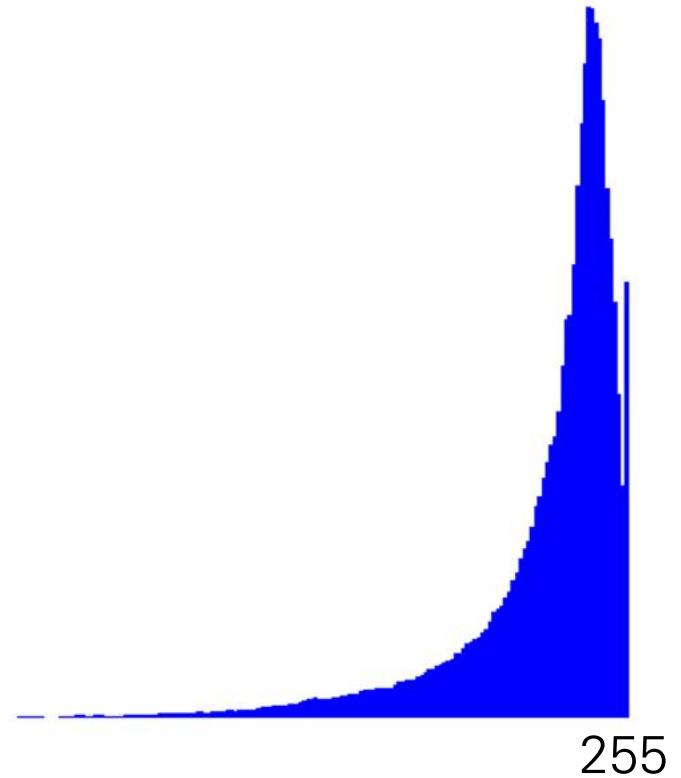
# Softmax Sampling



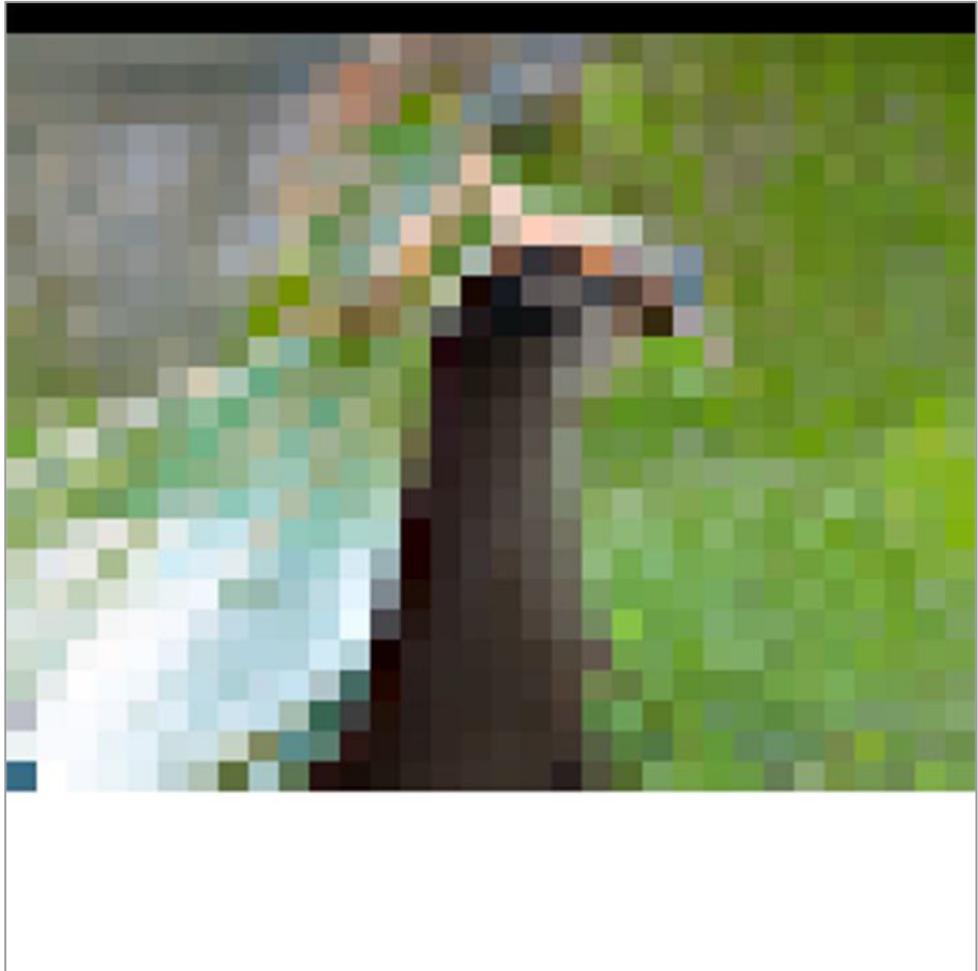
# Softmax Sampling



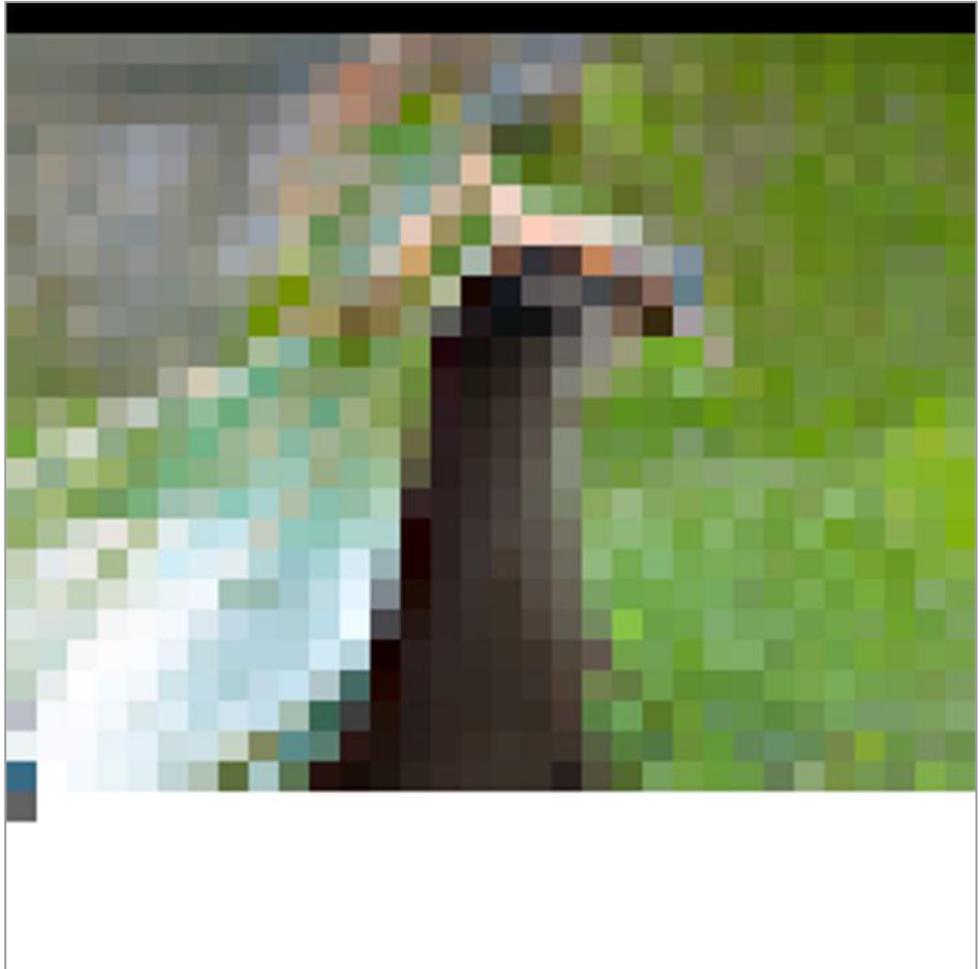
0



# Softmax Sampling

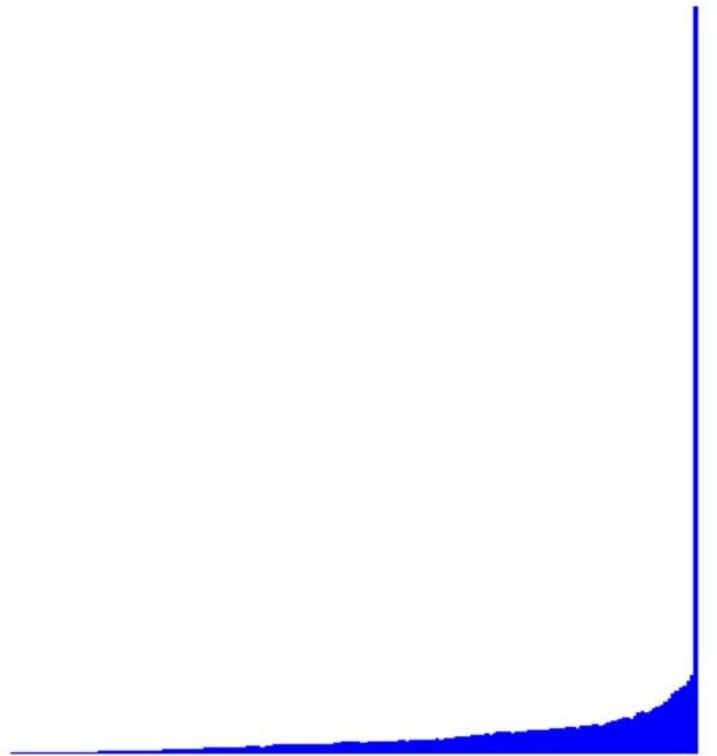


# Softmax Sampling

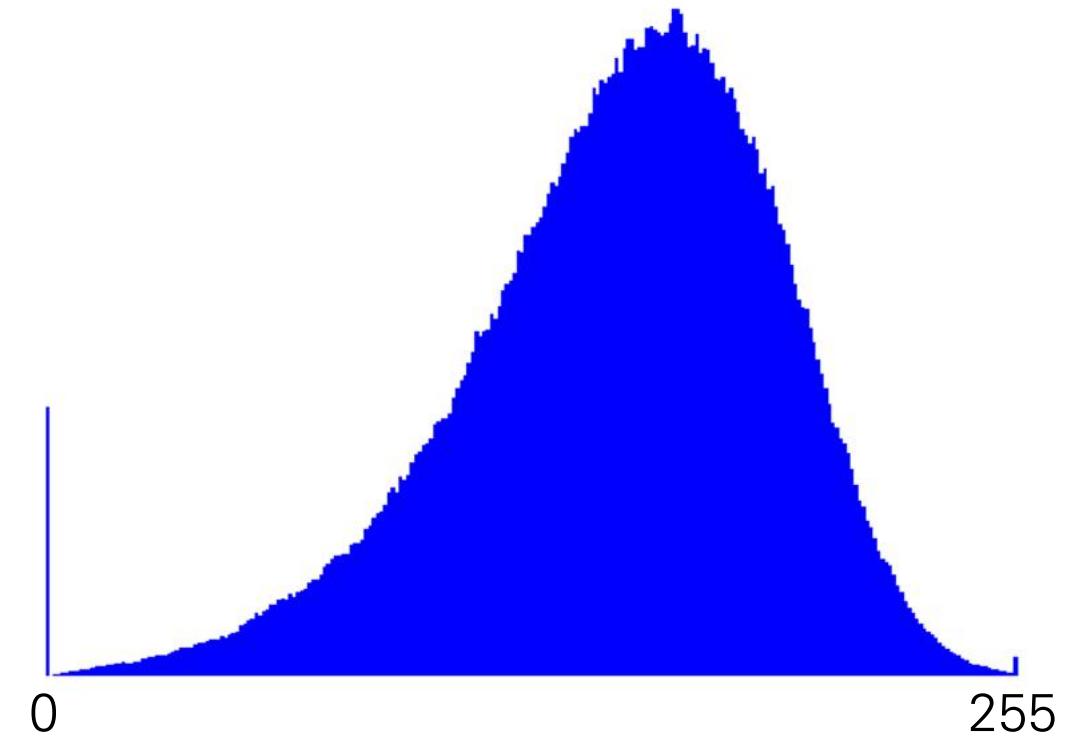
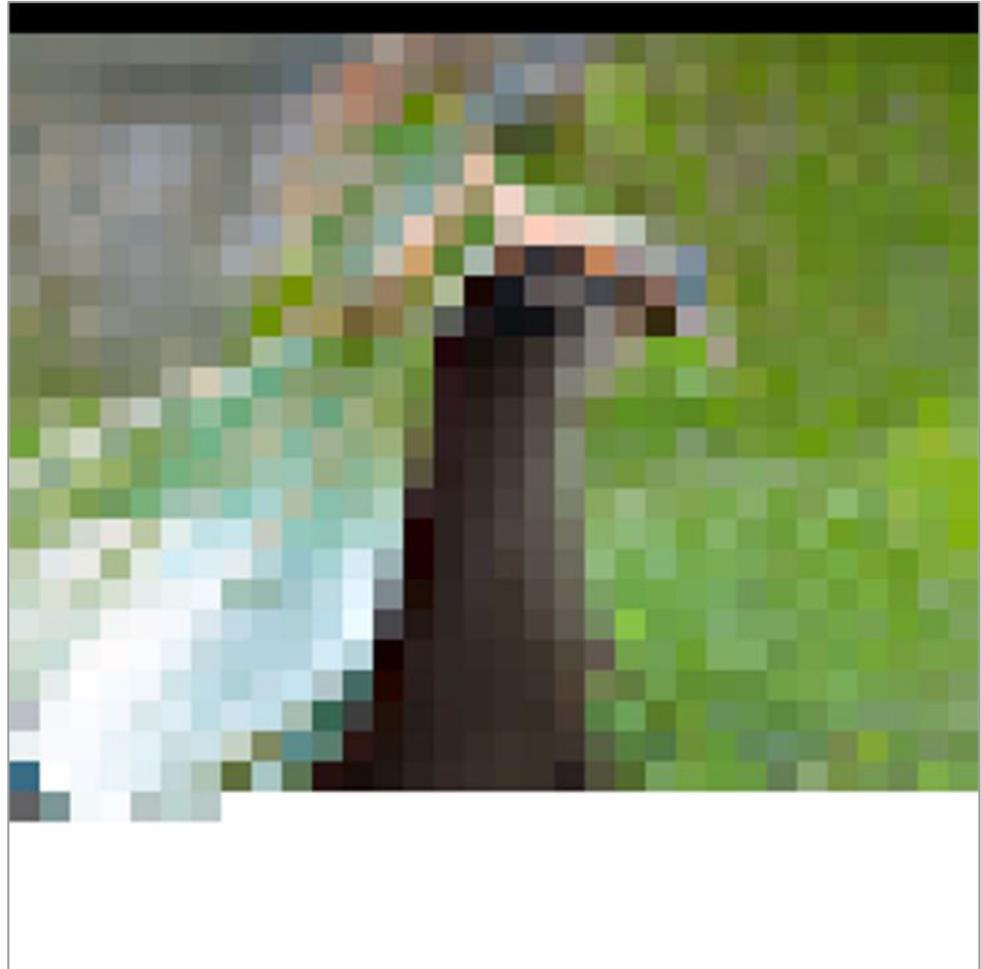


0

255



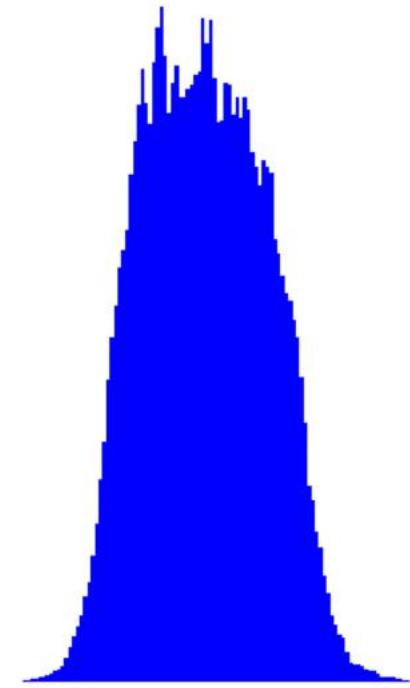
# Softmax Sampling



# Softmax Sampling

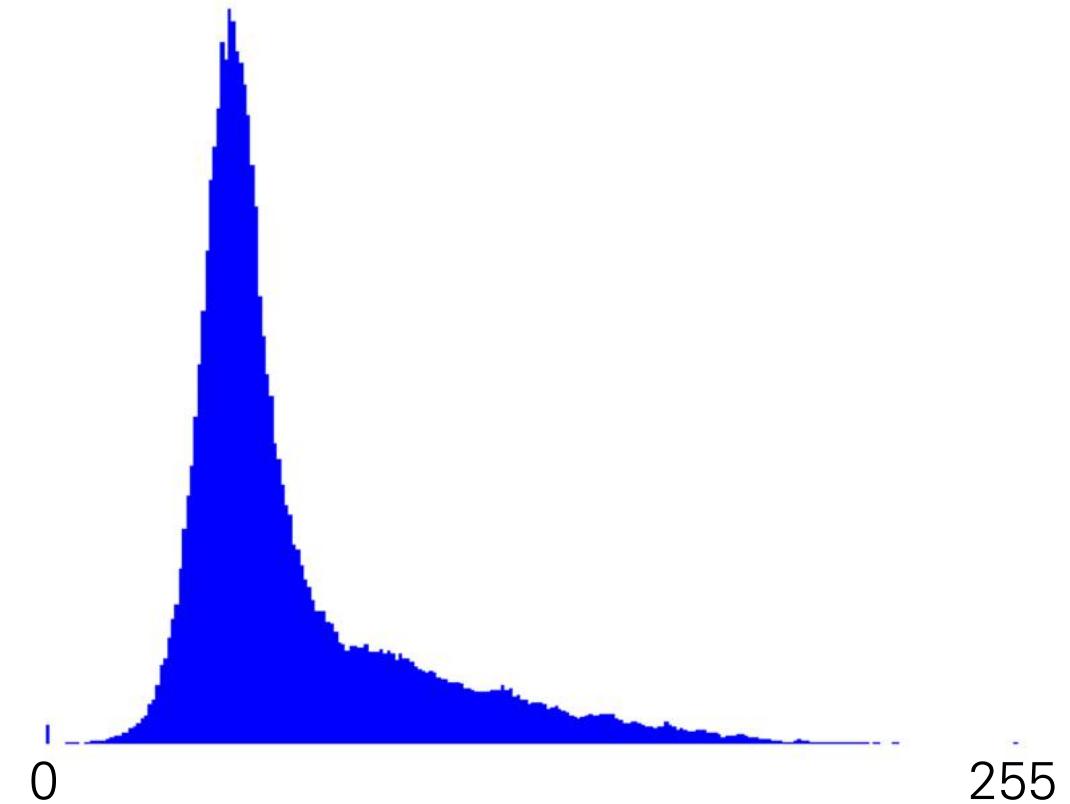


0

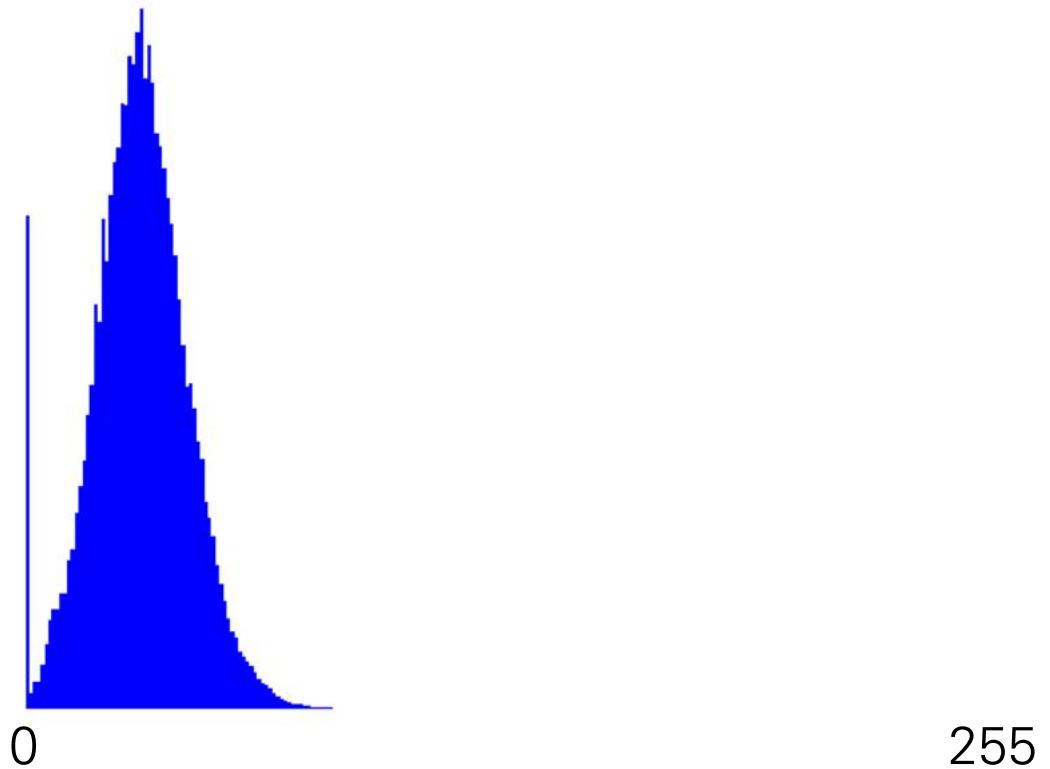
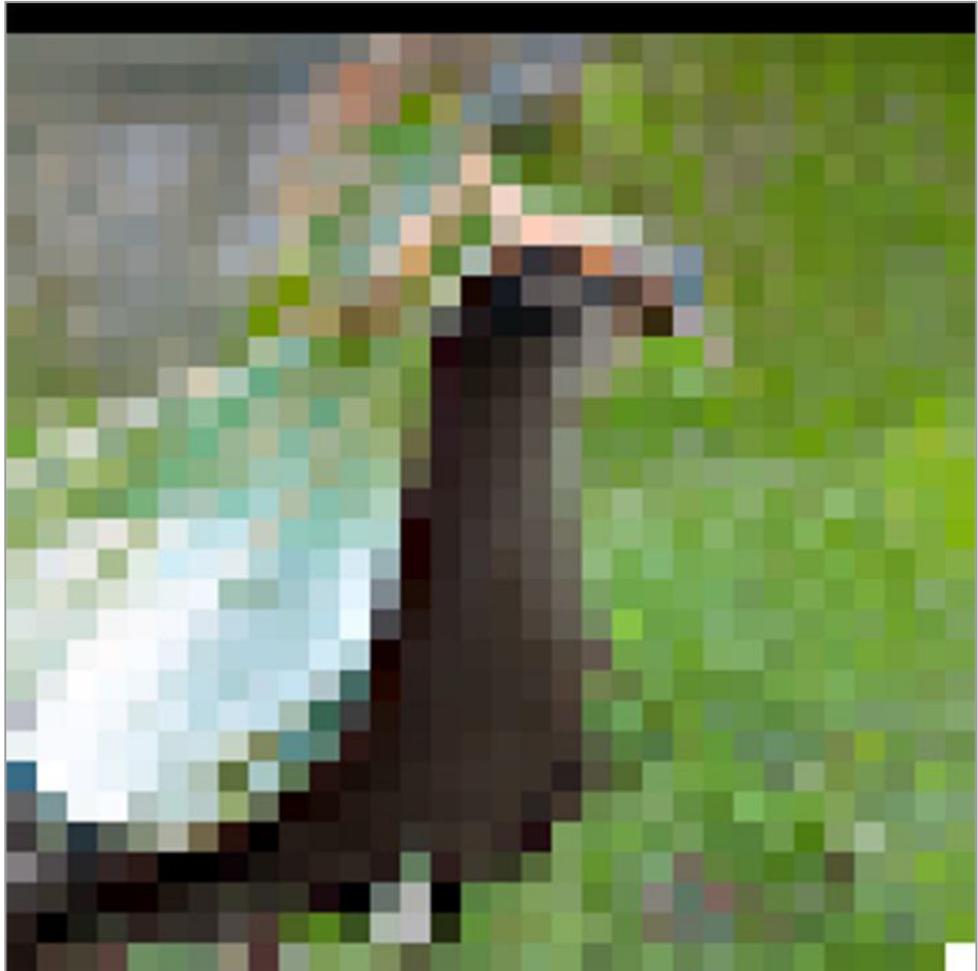


255

# Softmax Sampling



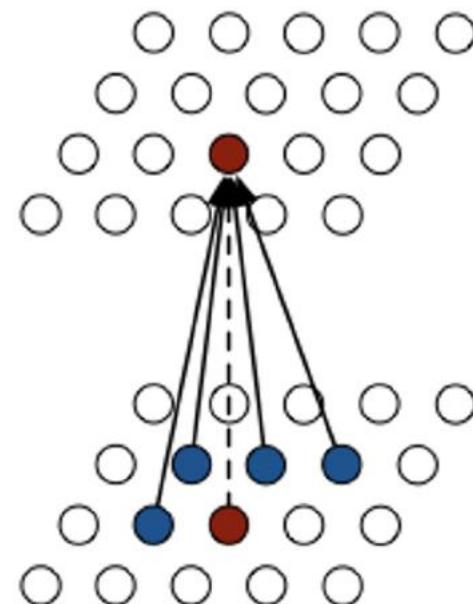
# Softmax Sampling



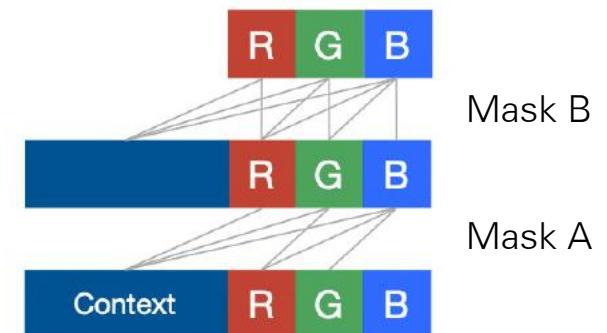
# PixelCNN

- 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

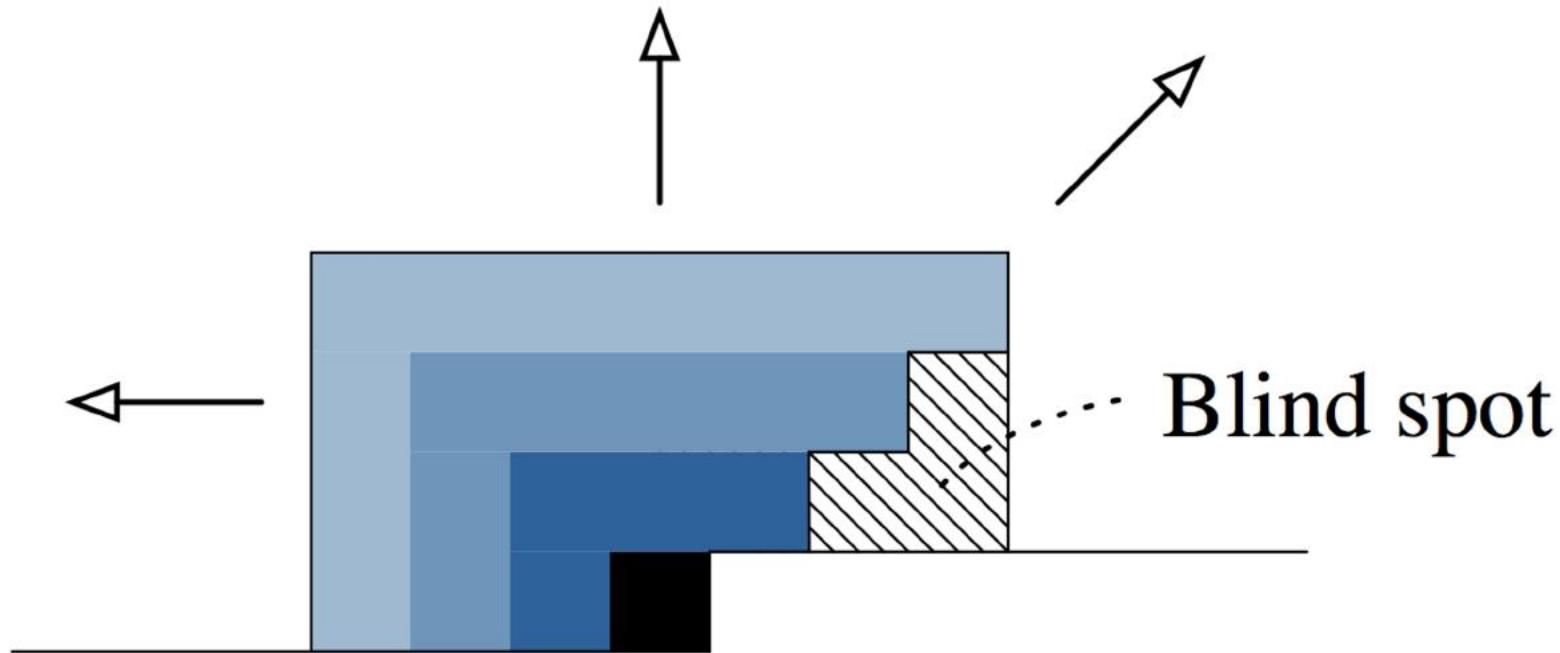


PixelCNN



# PixelCNN

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



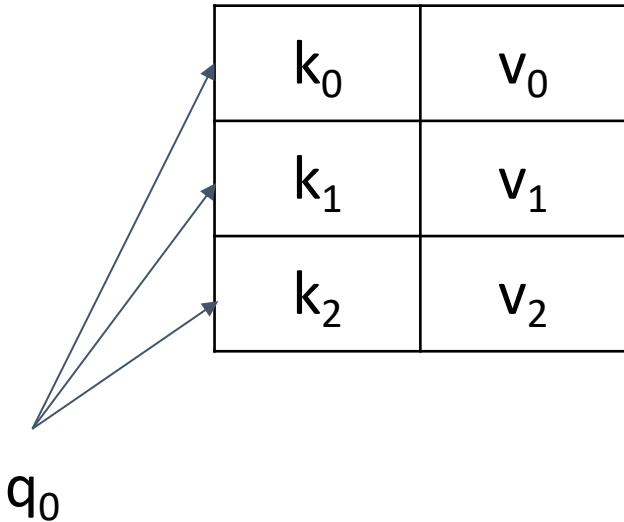
# Lecture overview

- 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



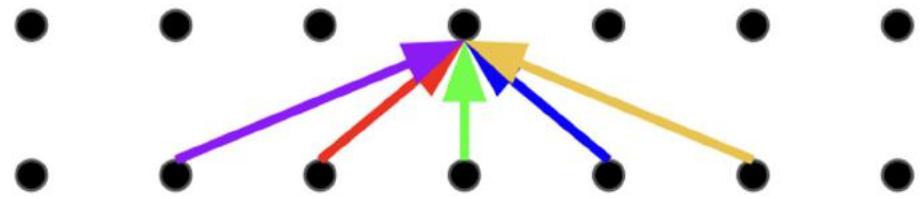
$$A(q, K, V) = \sum_i \frac{e^{s(q, k_i)}}{\sum_j e^{s(q, k_j)}} v_i$$

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

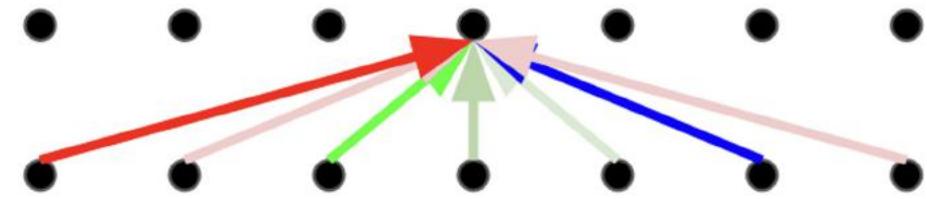
Matrix Form: QKV are  $L \times D$

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

# Self-Attention

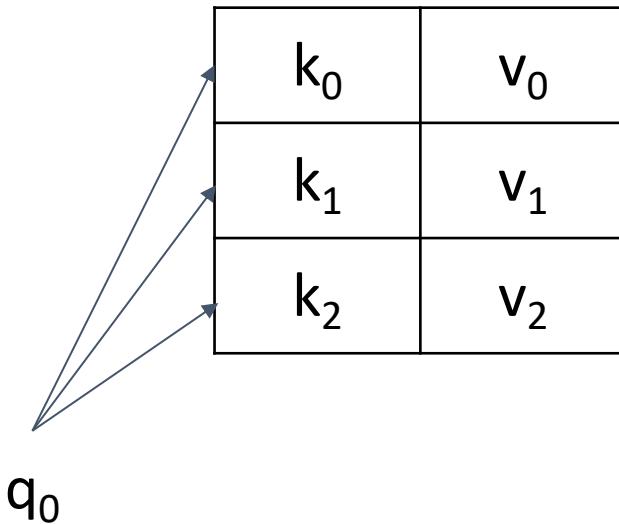


Convolution



Self-attention

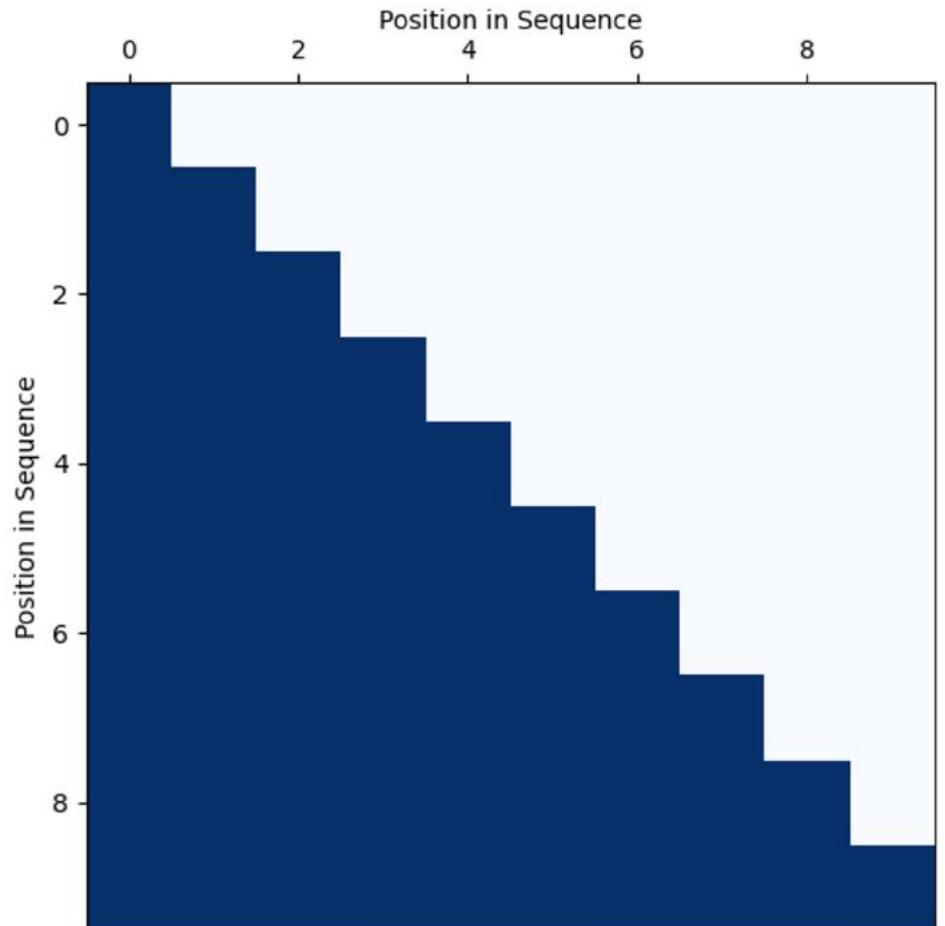
# Self-Attention



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

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

# Masked Attention



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

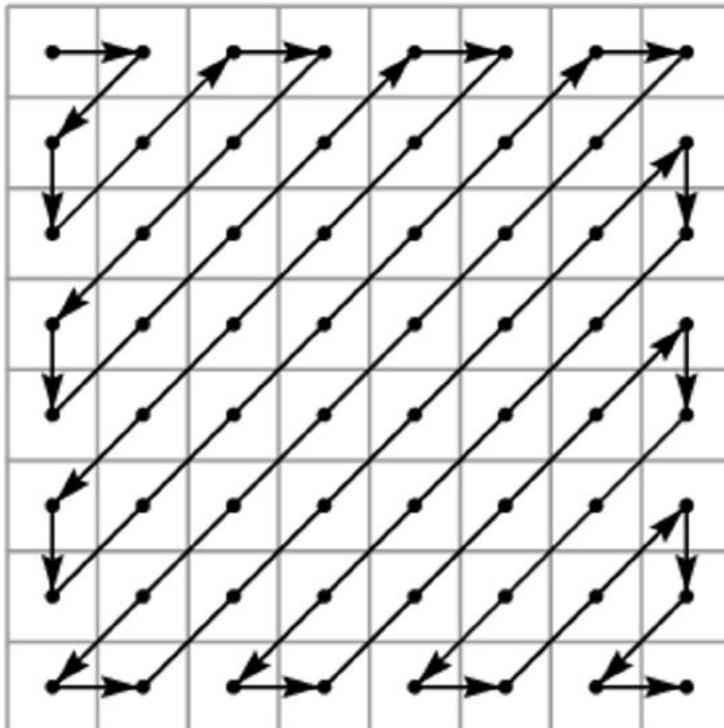
$$s(q, k) = \frac{q \cdot k}{\sqrt{d}} - (1 - \text{mask}(q, k)) * 10^{10}$$

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

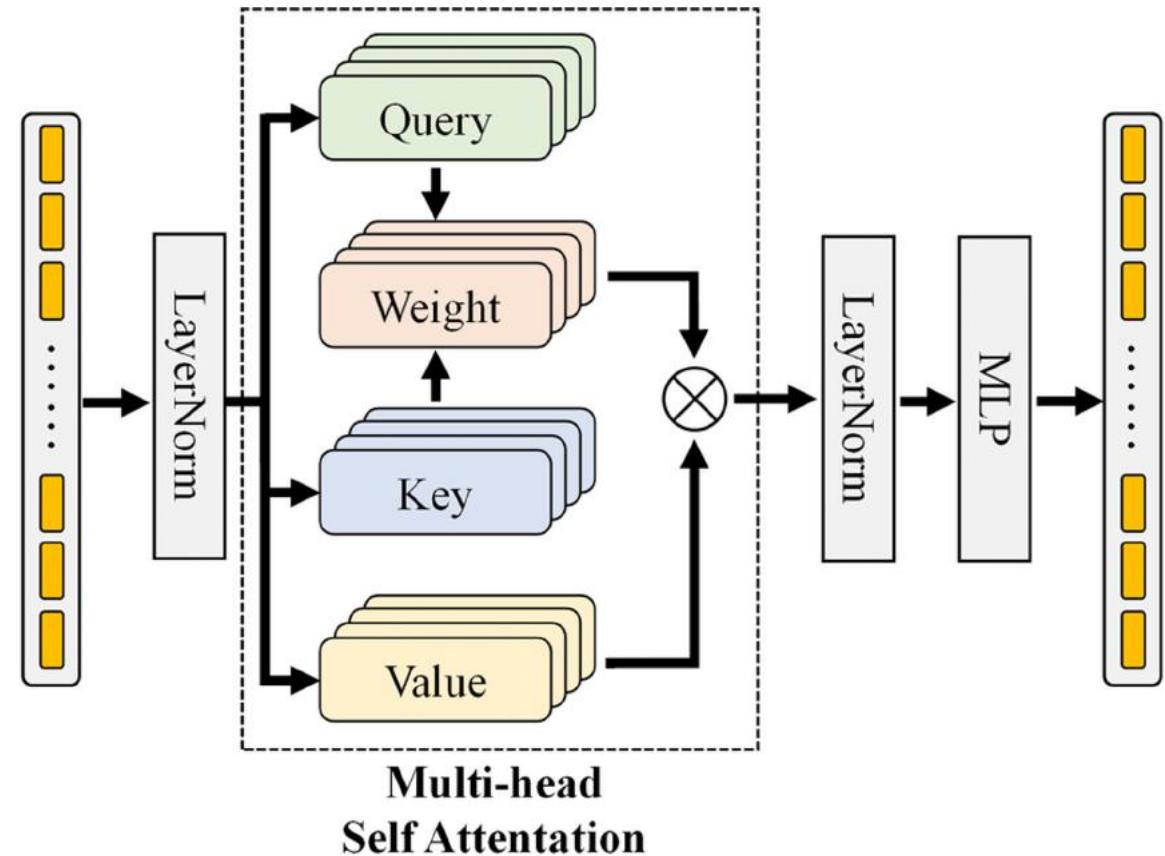
- How to implement with masked conv?
- Trivial to do with masked attention!

# Transformers

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

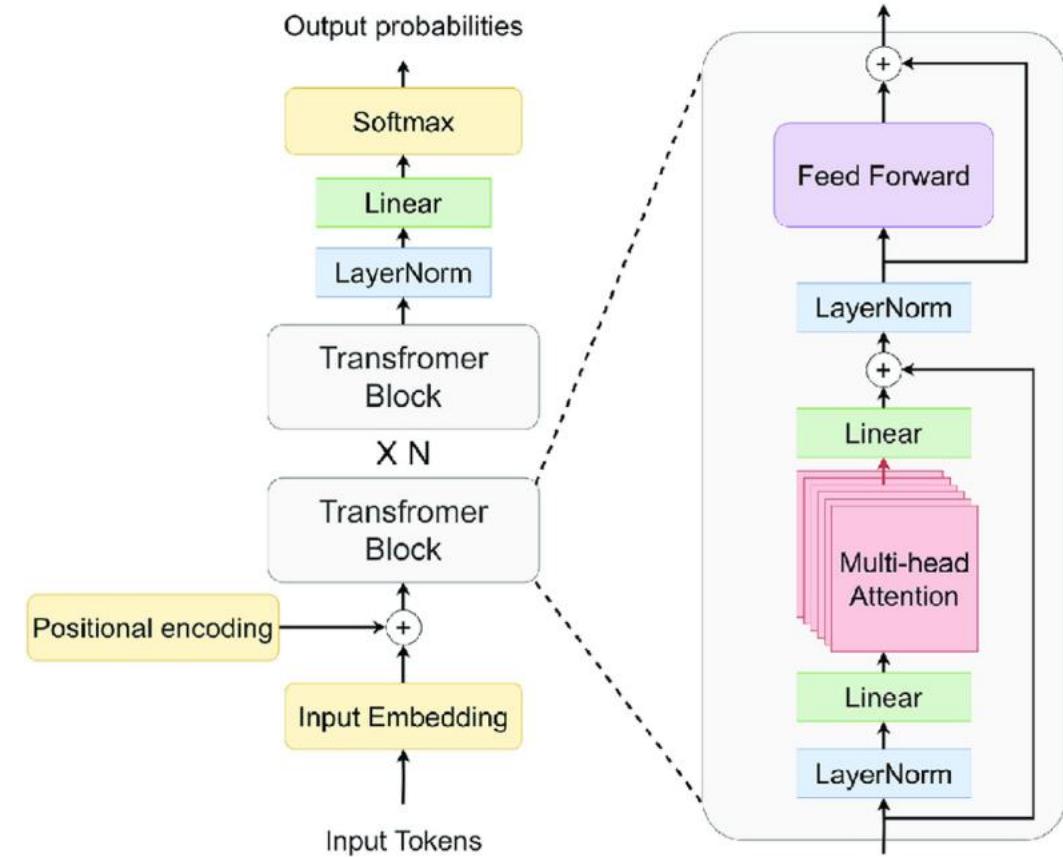
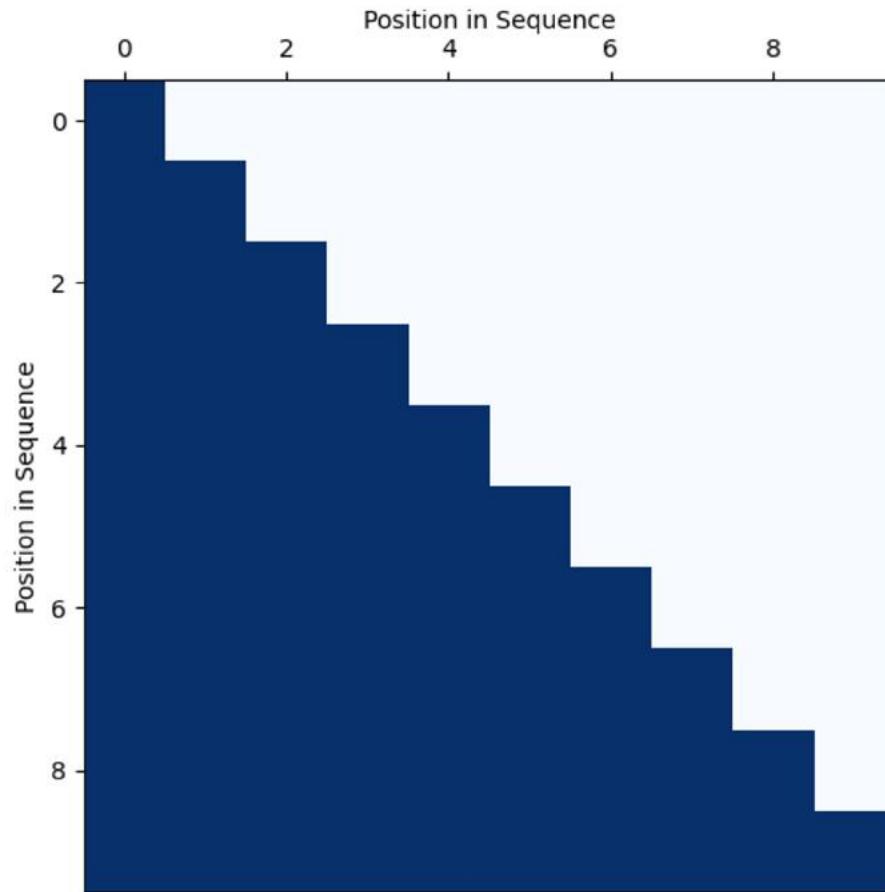
$$h = h + \text{MHS}(\text{LayerNorm}(h))$$

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



# 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  $V$ 
```

# 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

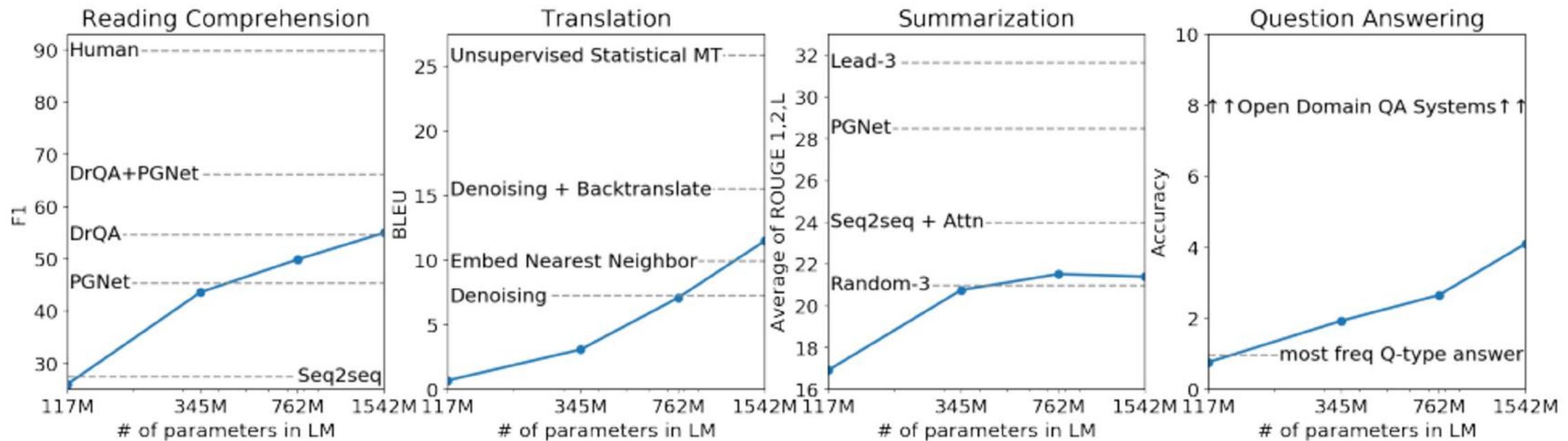
# GPT-1

- 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]	<u>77.6</u>	-	-	-
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2
BiAttention MRU [59] (9x)	-	<u>60.2</u>	<u>50.3</u>	<u>53.3</u>
Finetuned Transformer LM (ours)	<b>86.5</b>	<b>62.9</b>	<b>57.4</b>	<b>59.0</b>

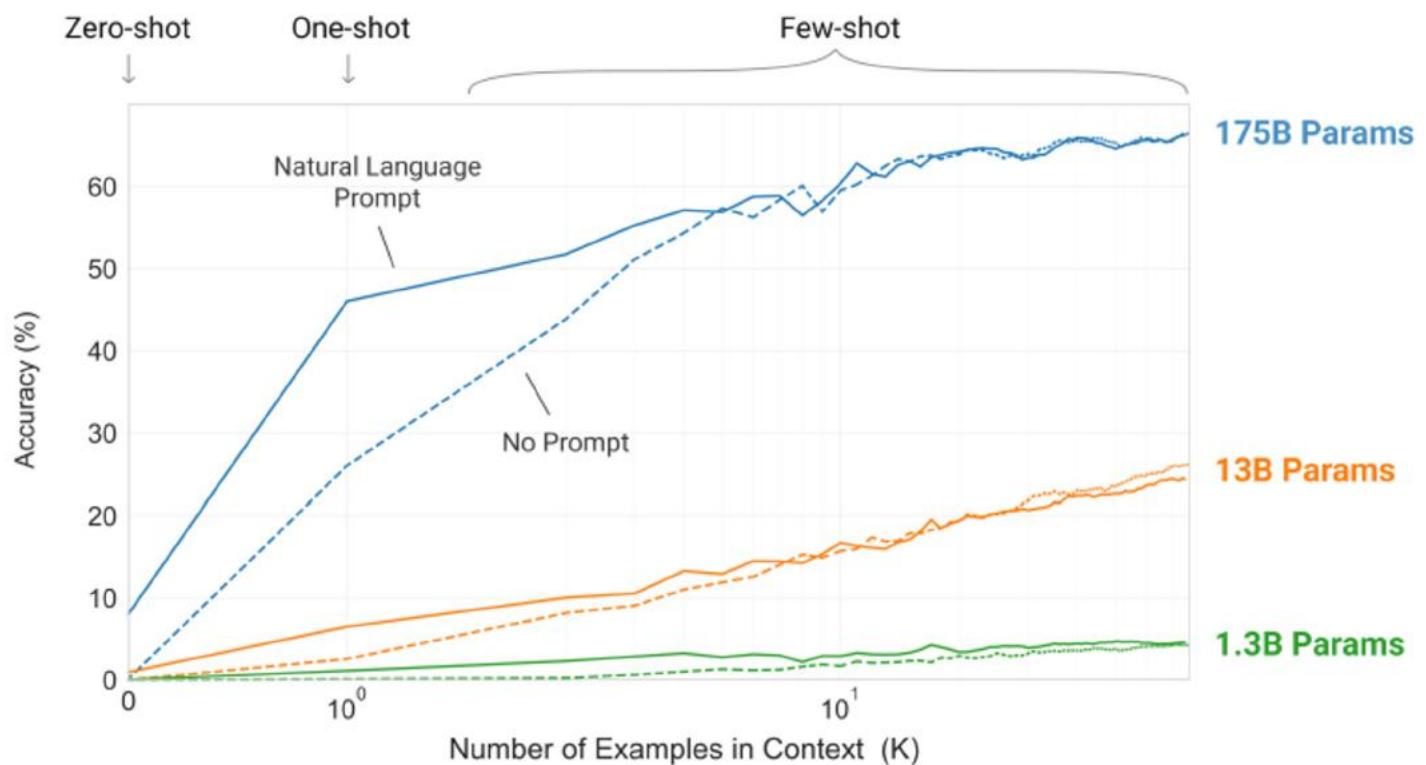
# GPT-2

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



# GPT-3

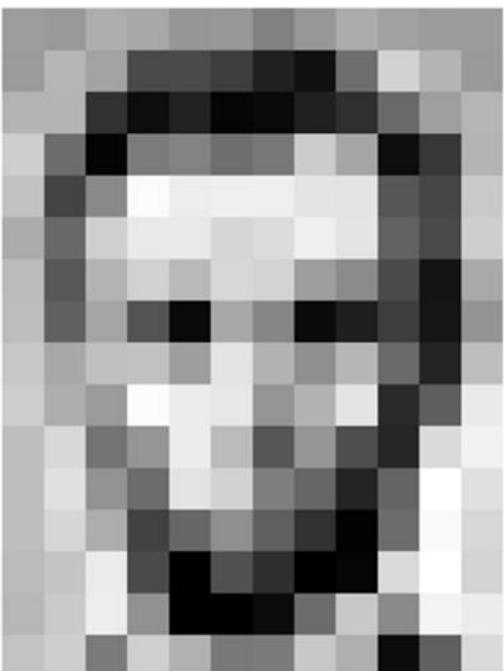
- In-context learning abilities emerge as number of parameters grow



# Tokens for Images

Consider encoding RGB ( $H \times W \times 3$ ) 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	93	17	110	210	180	154
180	180	50	14	34	6	10	33	48	105	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	63	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	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	9	9	12	108	209	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

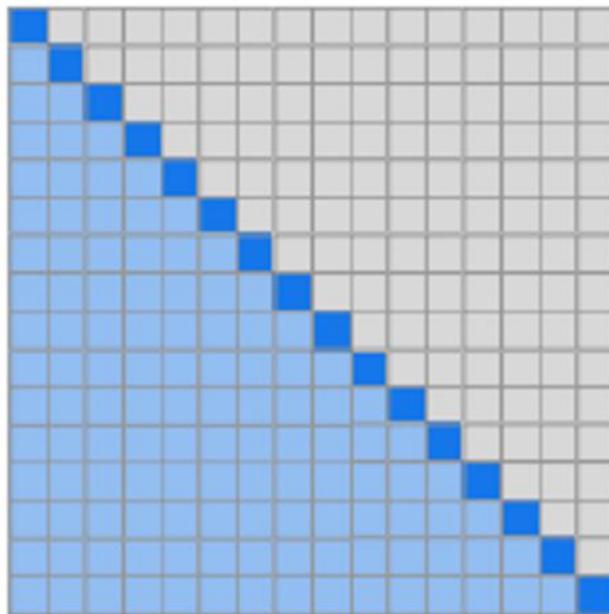
# Raw Pixels as Tokens

- $64 \times 64 \times 3 = 12\text{k}$  tokens
- $256 \times 256 \times 3 = 196\text{k}$  tokens!

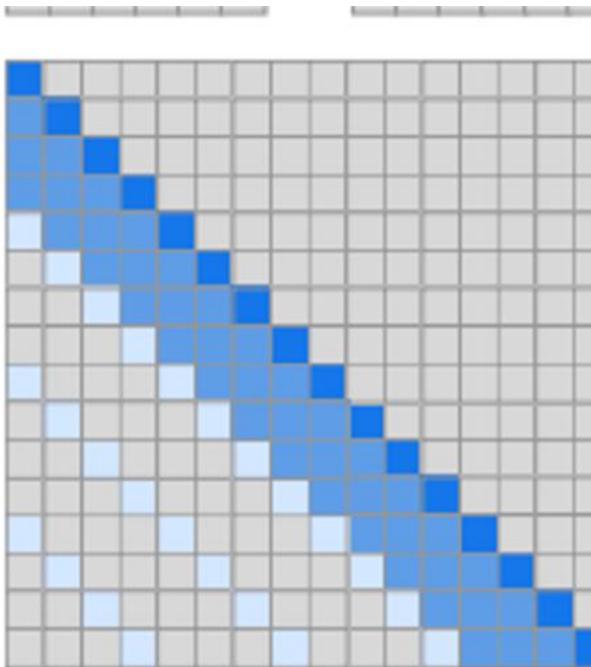
Quadratic scaling of attention makes this prohibitively expensive

# Sparse Transformer

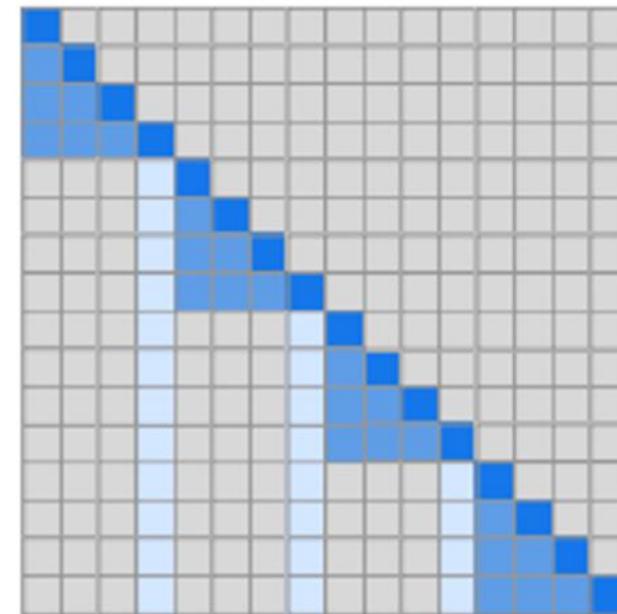
- Train a causal transformer, but with special hard-coded masking



(a) Transformer



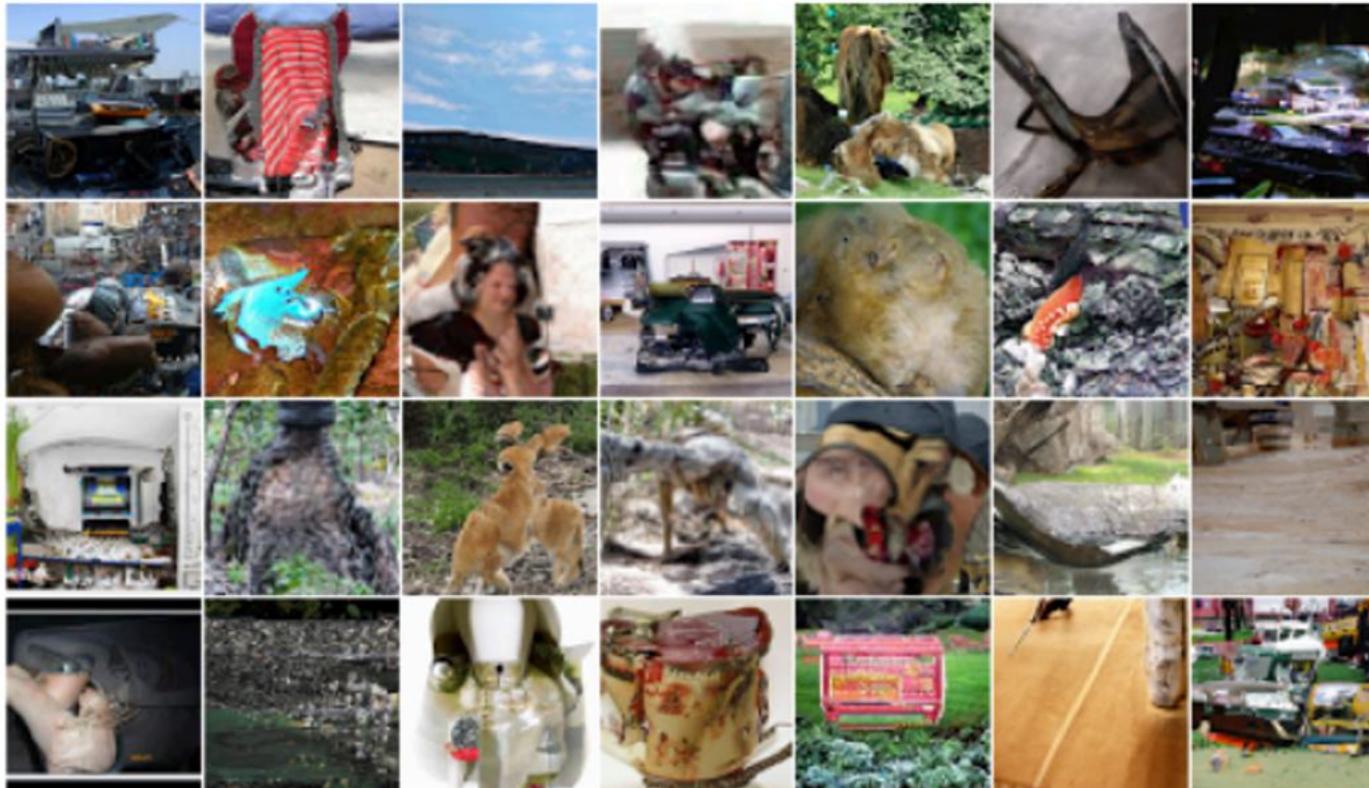
(b) Sparse Transformer (strided)



(c) Sparse Transformer (fixed)

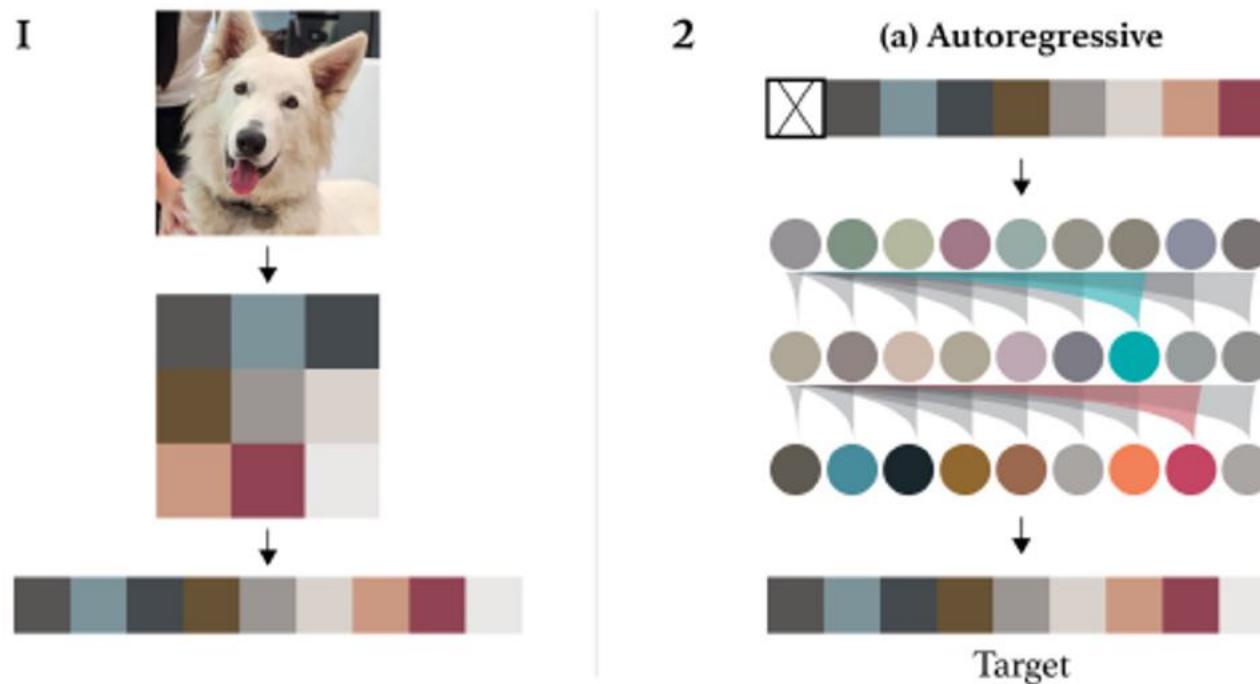
# Sparse Transformer

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



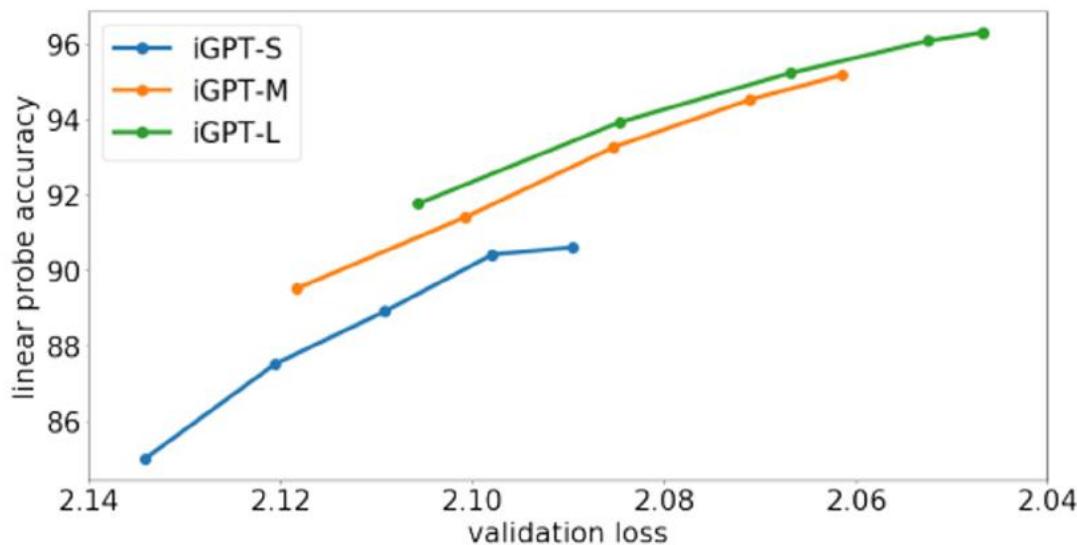
# iGPT

- 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



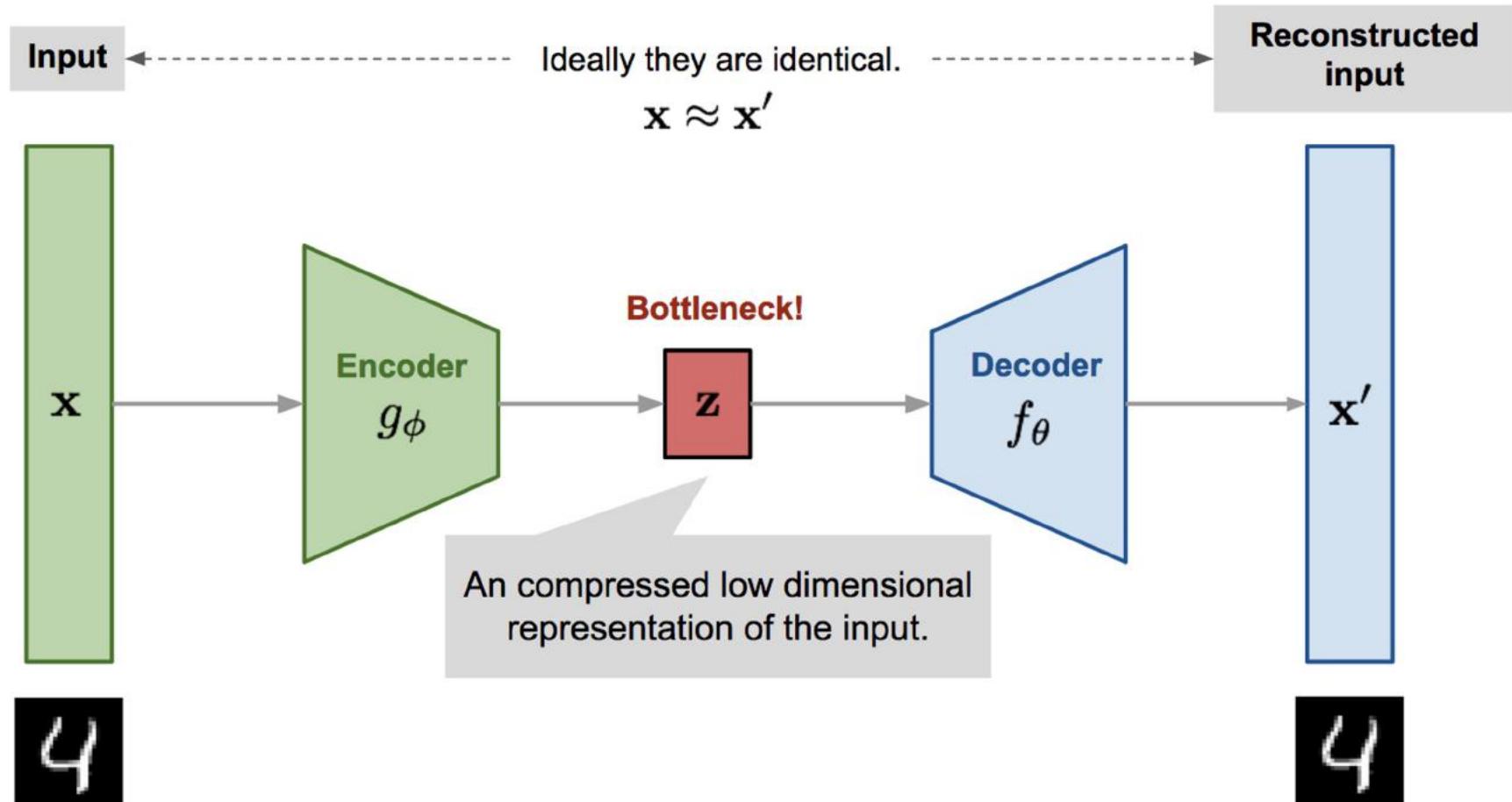
Model	Acc	Unsup Transfer	Sup Transfer
<b>CIFAR-10</b>			
ResNet-152	94		✓
SimCLR	95.3	✓	
iGPT-L	96.3	✓	
<b>CIFAR-100</b>			
ResNet-152	78.0		✓
SimCLR	80.2	✓	
iGPT-L	82.8	✓	
<b>STL-10</b>			
AMDIM-L	94.2	✓	
iGPT-L	95.5	✓	

# Tokens for Images

- But iGPT still needed to train on image as small as  $32 \times 32$
- 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 (LFO):  
[FSQ](#), [LFO](#)

# Discrete Autoencoders

This method of tokenization is lossy

- $256 \times 256 \times 3 \rightarrow 16 \times 16$ 
  - 384x compression in bytes from raw pixels
  - 768x reduction in sequence length

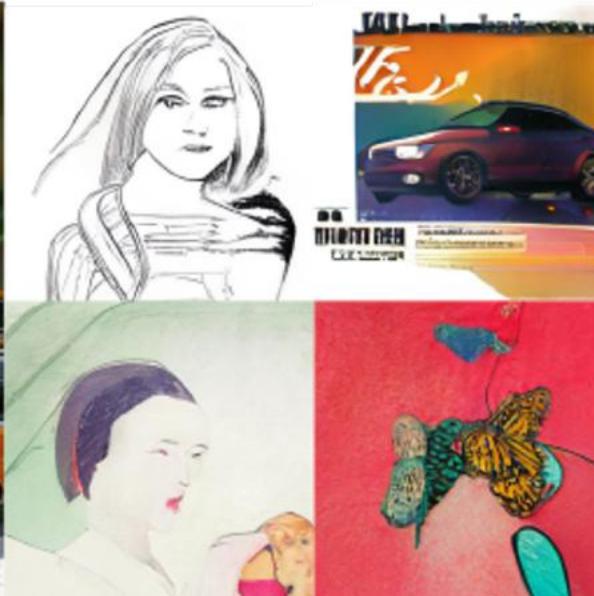


# VQGAN Transformer

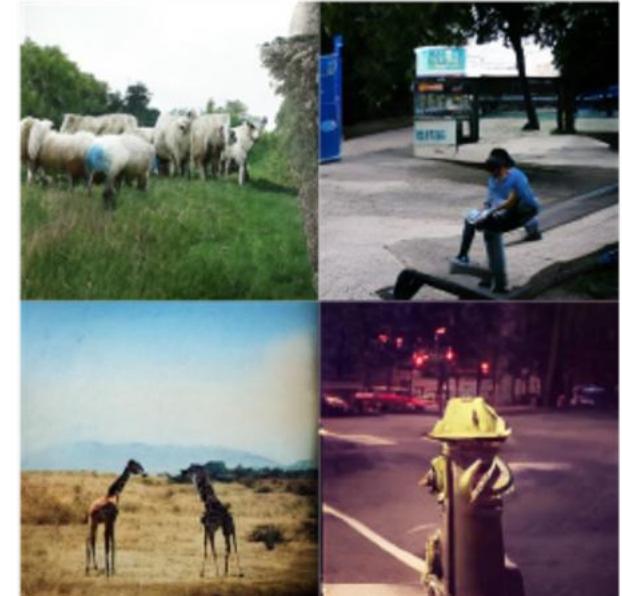
Train an autoregressive transformer to model discrete encodings

- Much more tractable than from pixels. Samples below

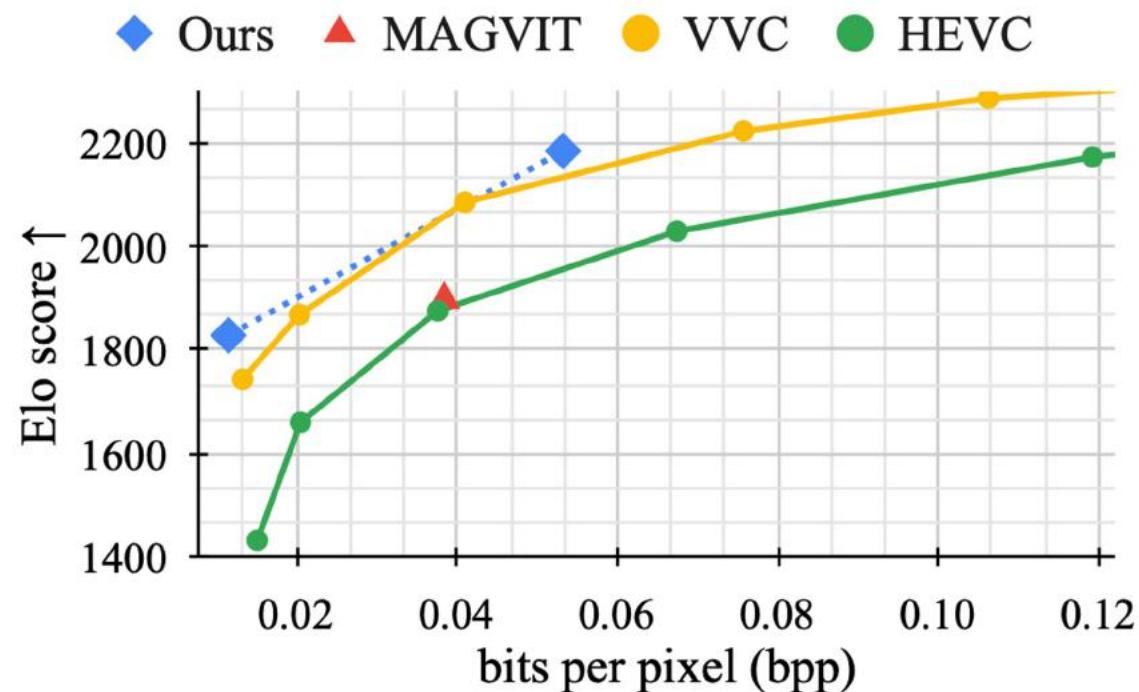
Open Images



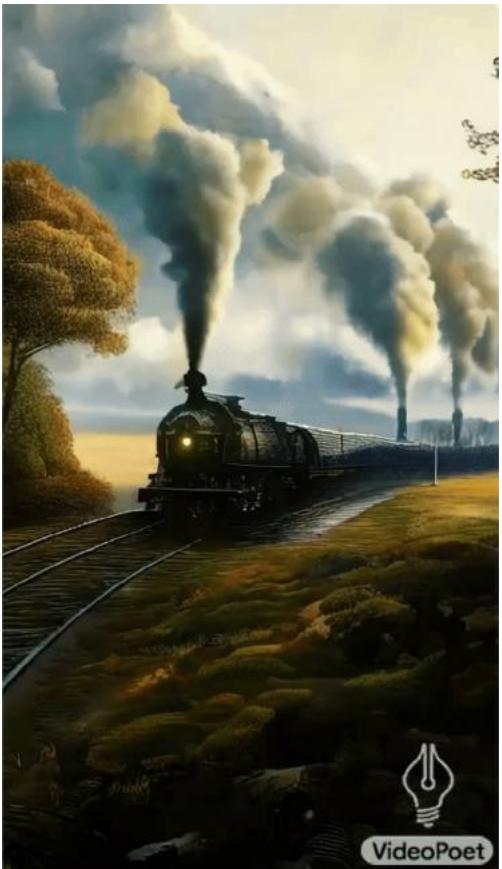
coco



# Discrete Encodings for Video



# VideoPoet



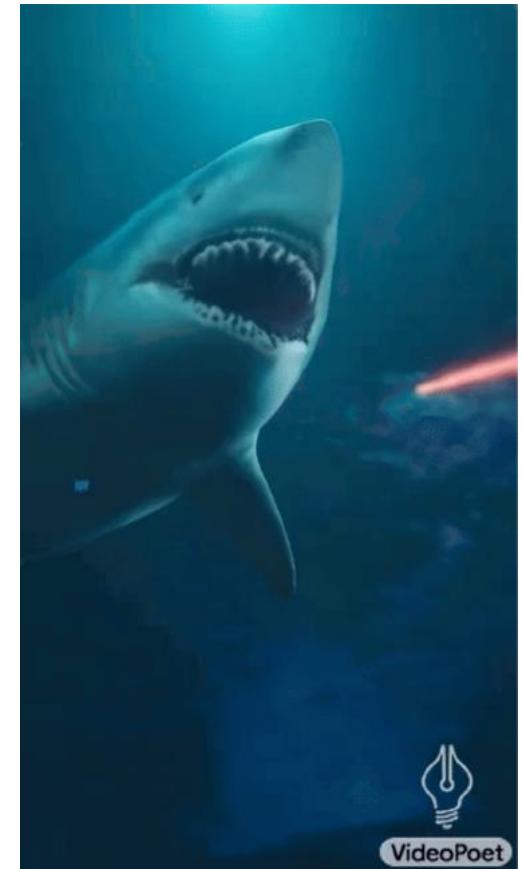
The orient express  
driving through a fantasy  
landscape, animated oil  
on canvas



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



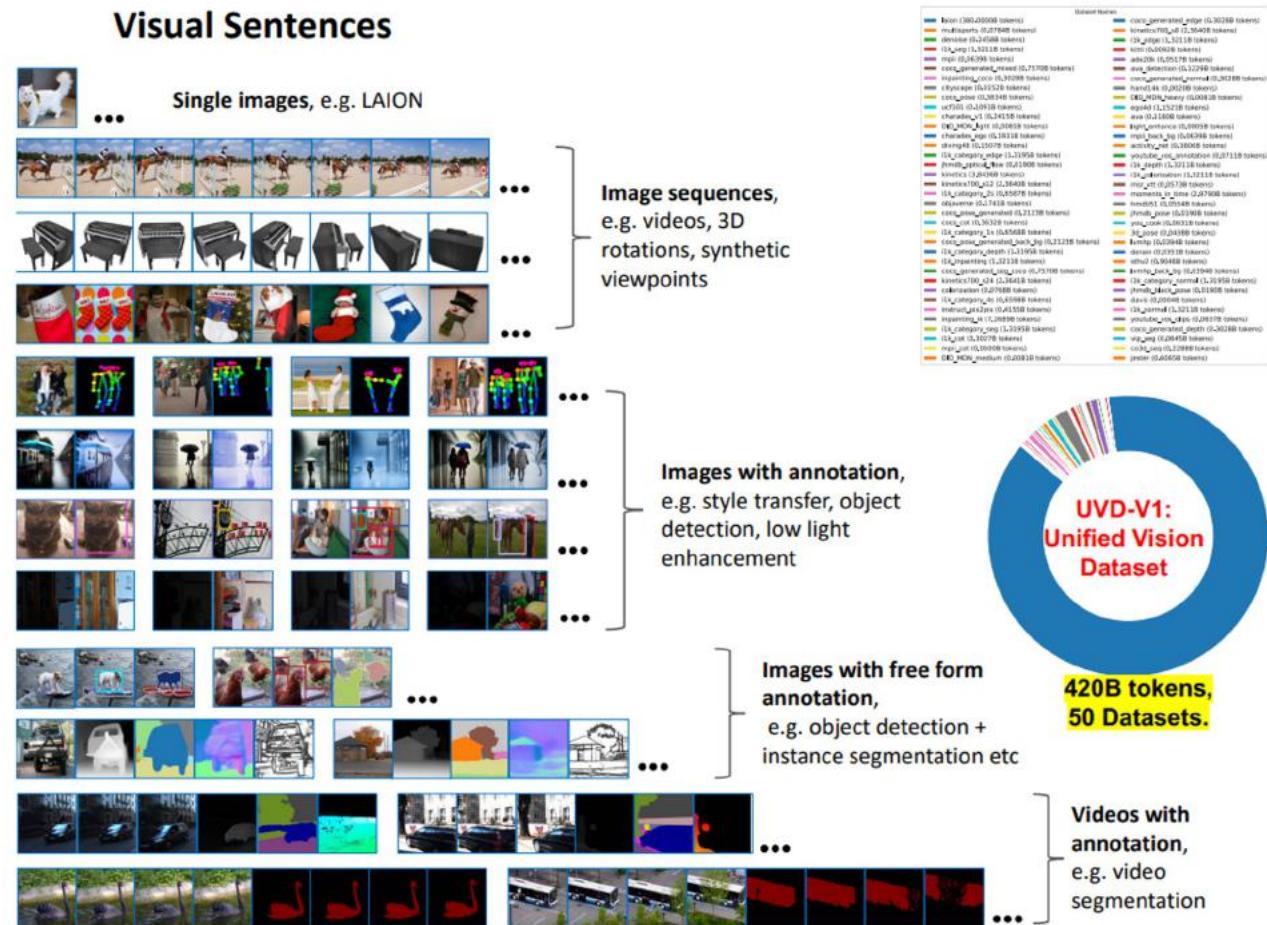
A chicken lifting weights



A shark with a laser  
beam coming out of its  
mouth

# In-Context Learning for Vision

- Train a large autoregressive transformer on sequential visual data



# In-Context Learning for Vision

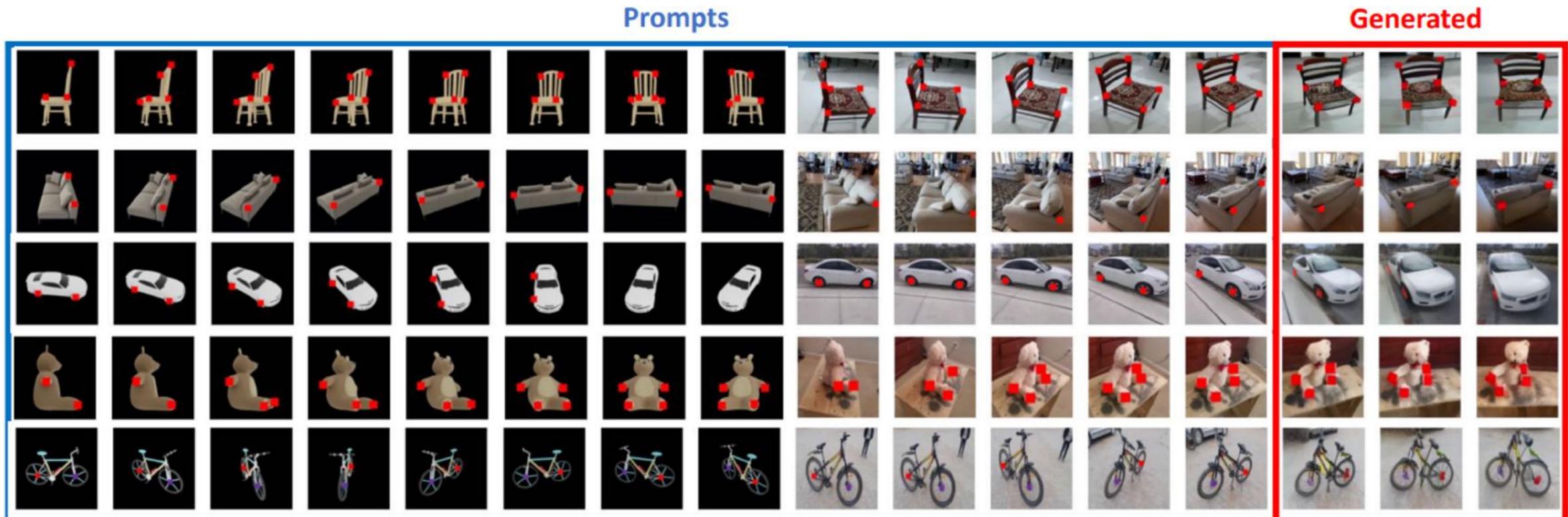
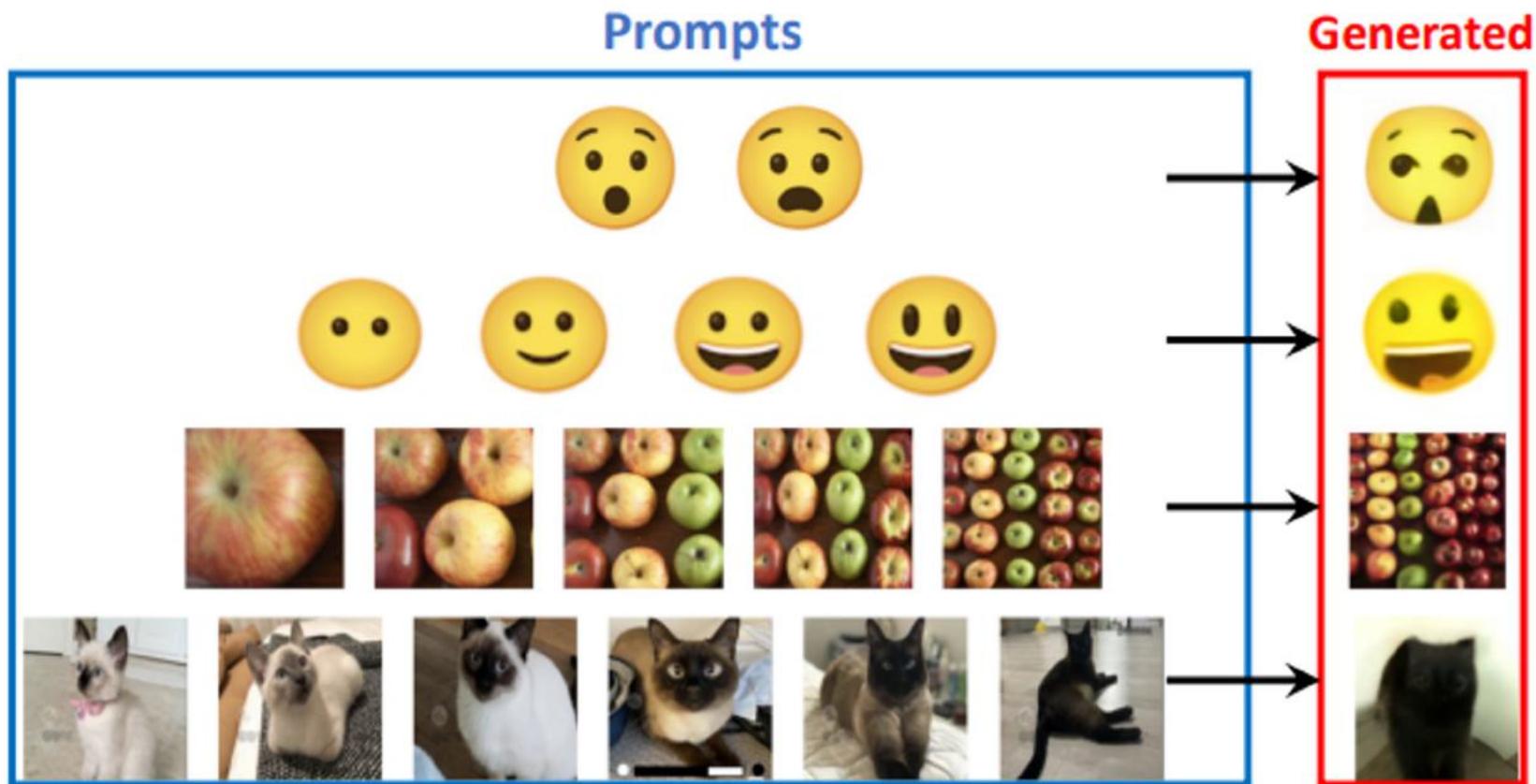
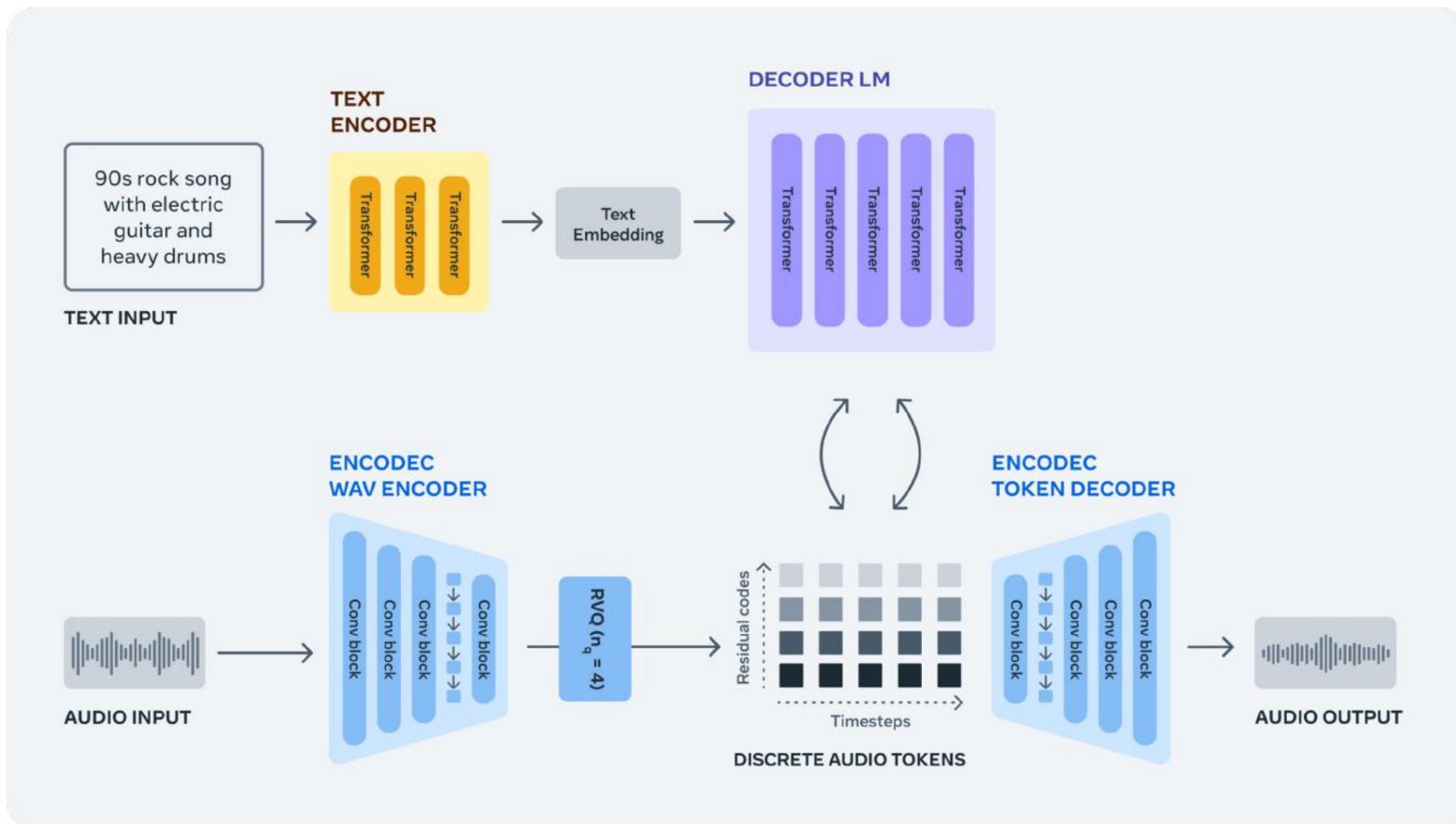


Figure 9. **Task Composability**. Examples of prompts that combine two different tasks – object rotation and keypoint tracking.

# In-Context Learning for Vision

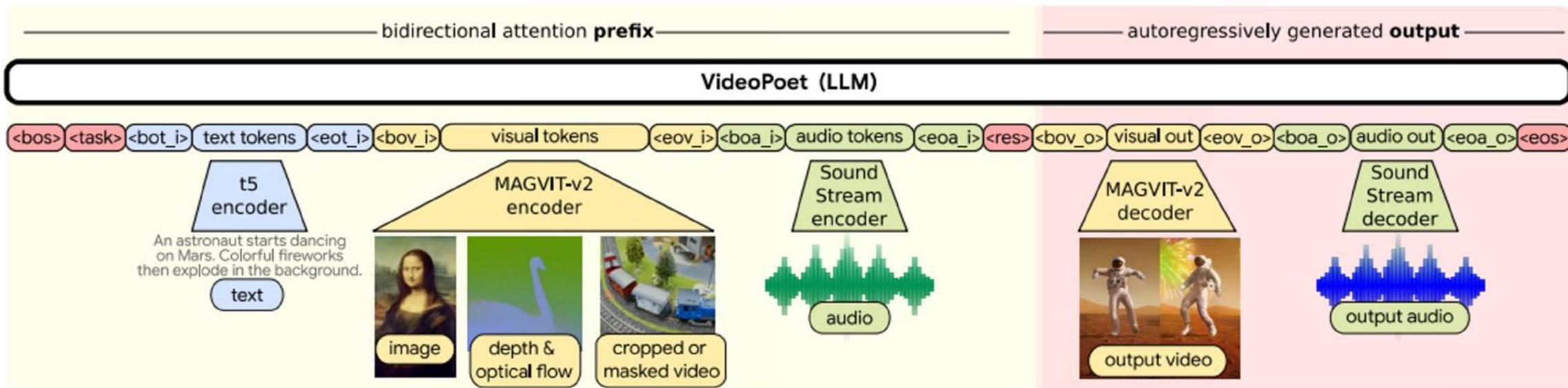


# AudioCraft



# 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

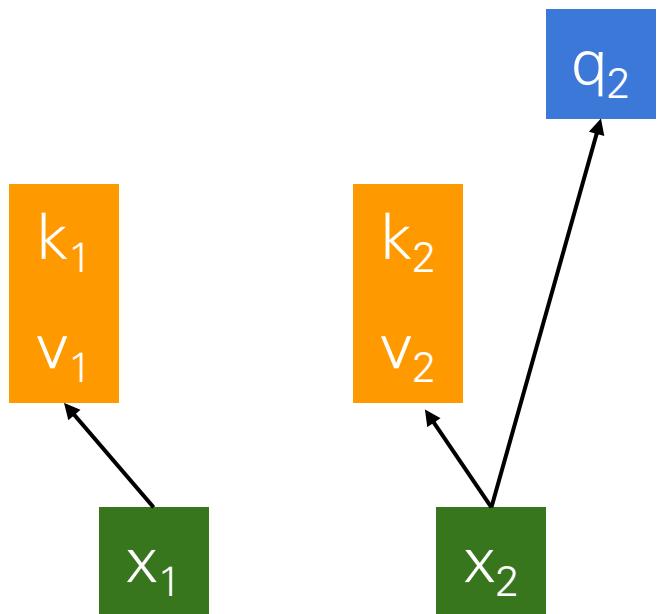
Naive Sampling

$x_1$

$x_2$

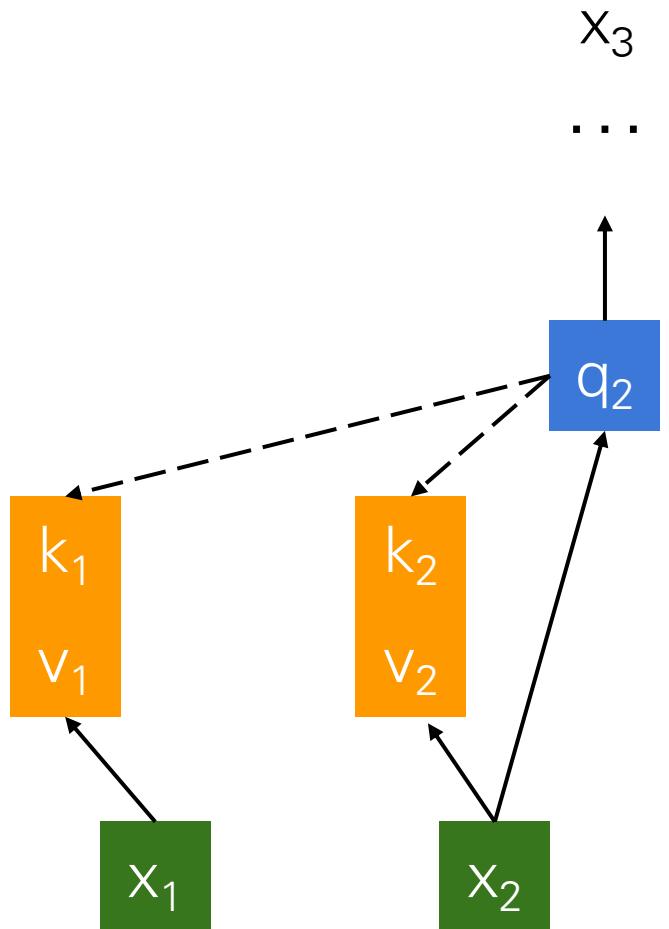
# Caching / Efficient Sampling

## Naive Sampling



# Caching / Efficient Sampling

Naive Sampling



# Caching / Efficient Sampling

## Naive Sampling

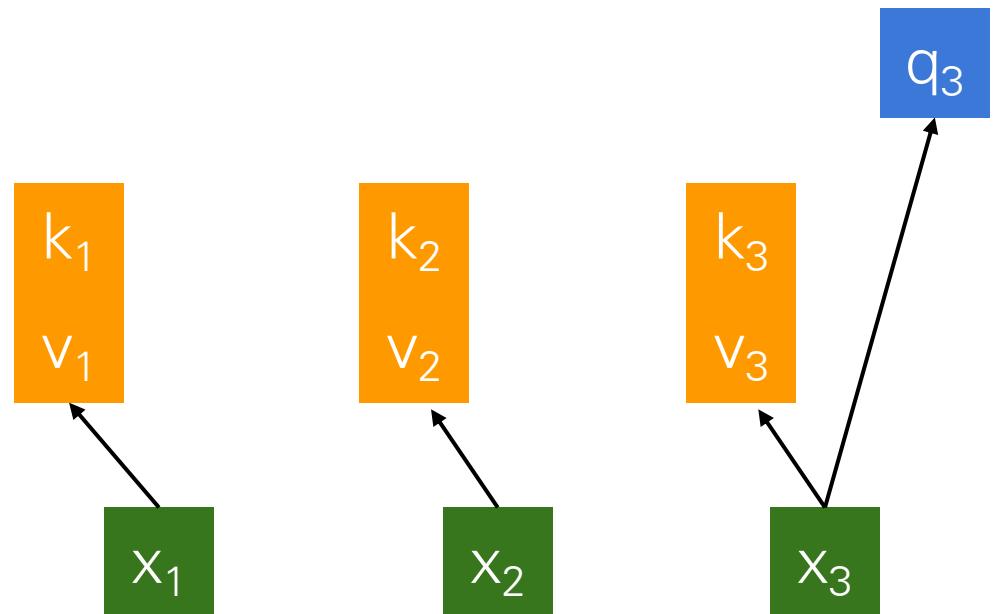
$x_1$

$x_2$

$x_3$

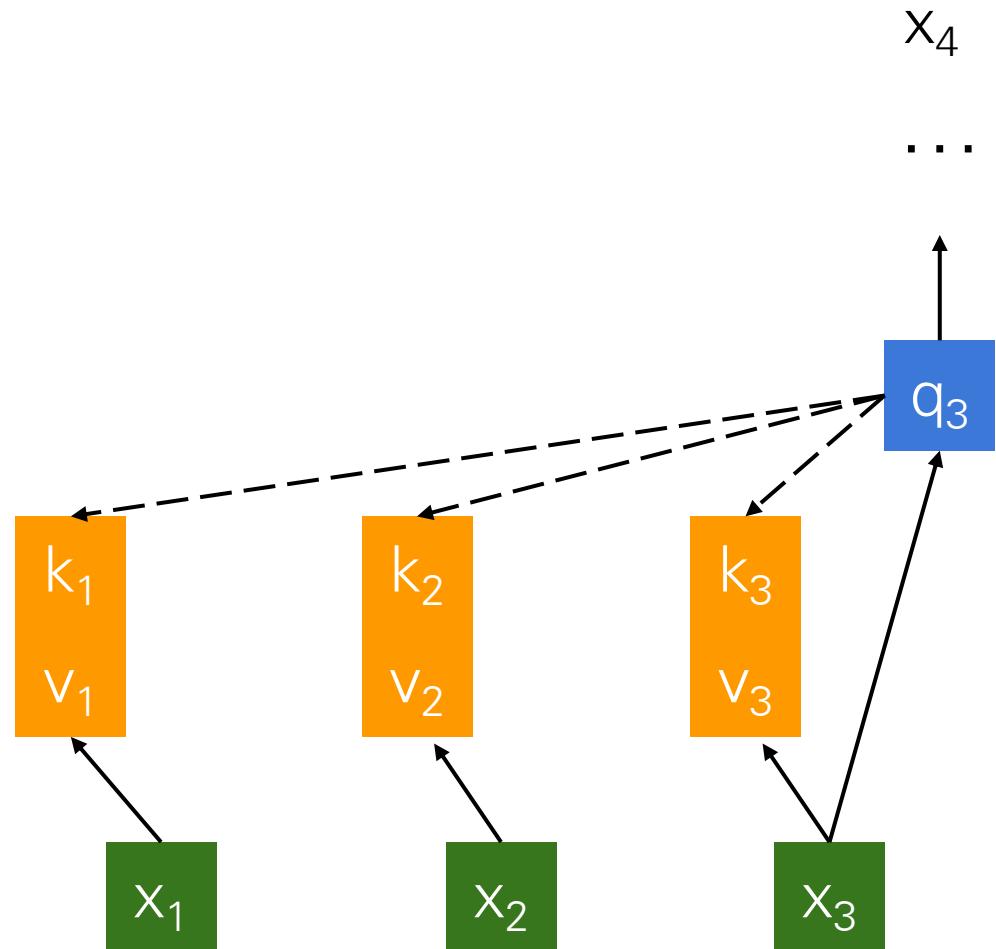
# Caching / Efficient Sampling

## Naive Sampling



# Caching / Efficient Sampling

Naive Sampling



# Caching / Efficient Sampling

## Naive Sampling

$x_1$

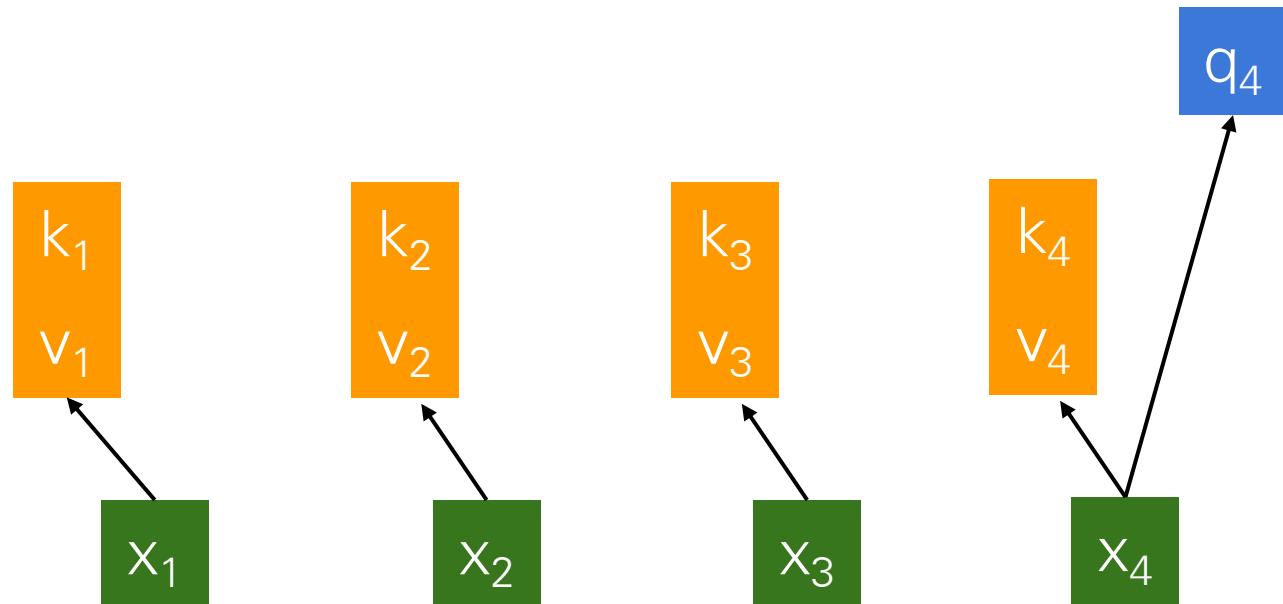
$x_2$

$x_3$

$x_4$

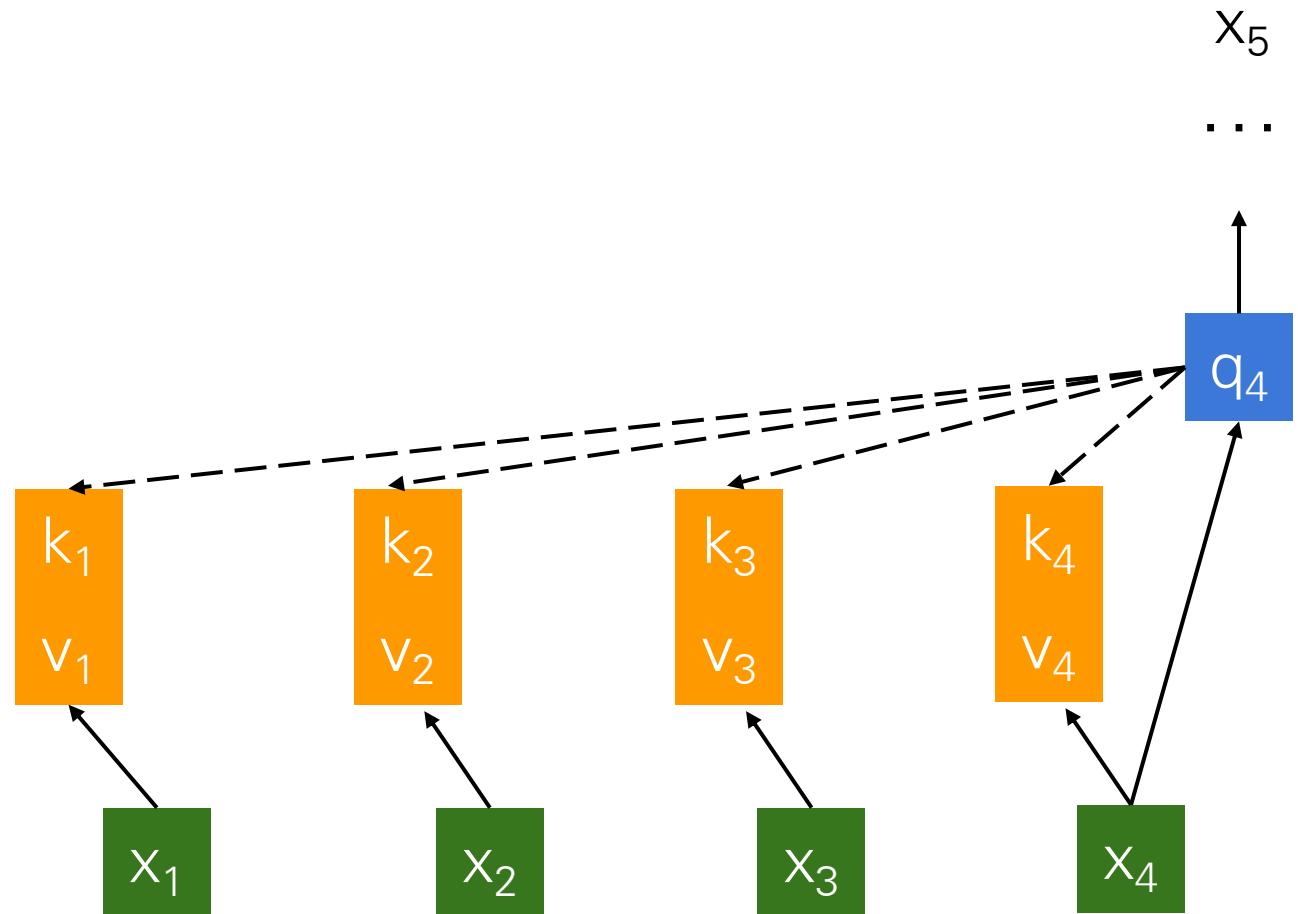
# Caching / Efficient Sampling

## Naive Sampling



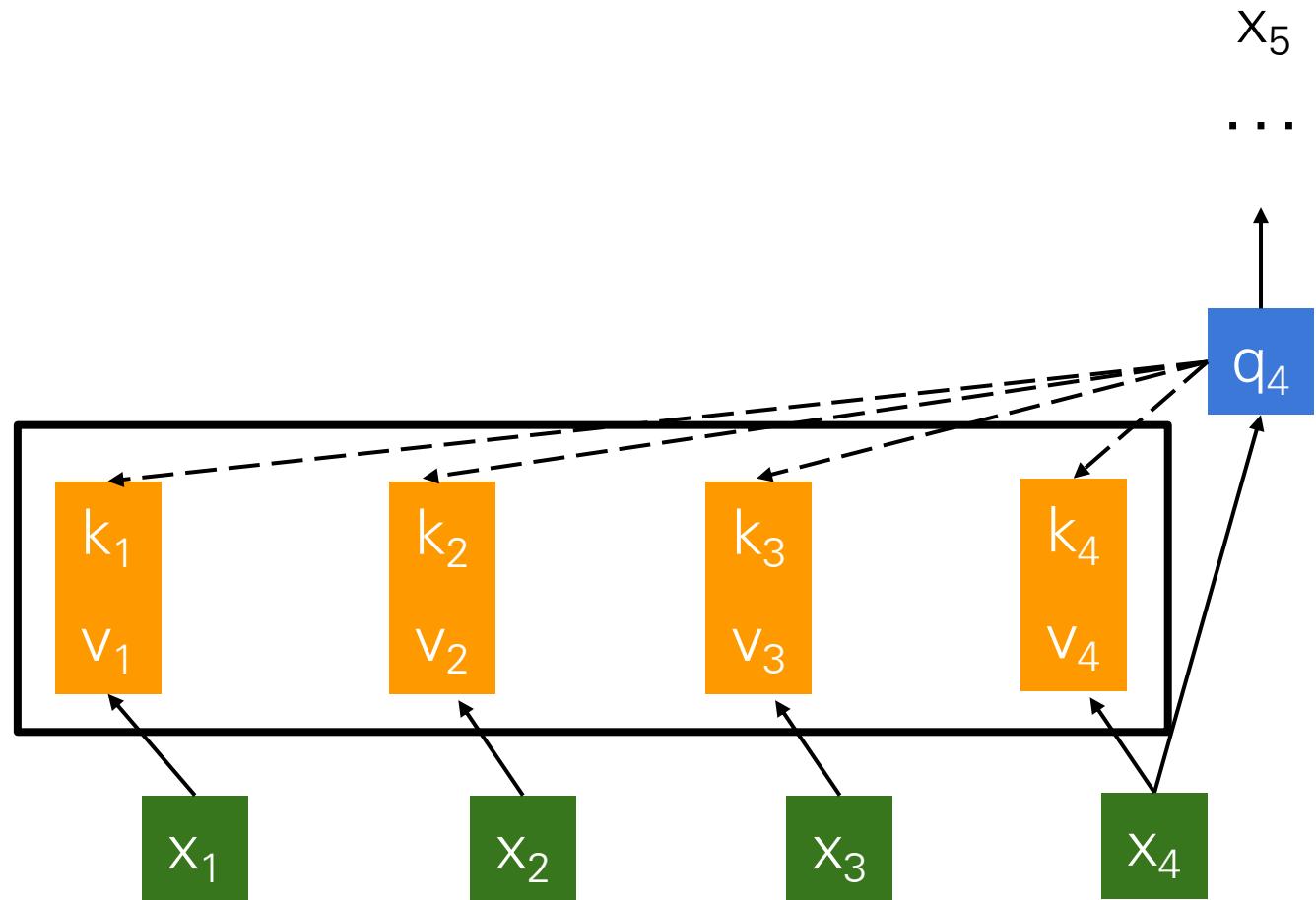
# Caching / Efficient Sampling

Naive Sampling



# Caching / Efficient Sampling

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



# Caching / Efficient Sampling

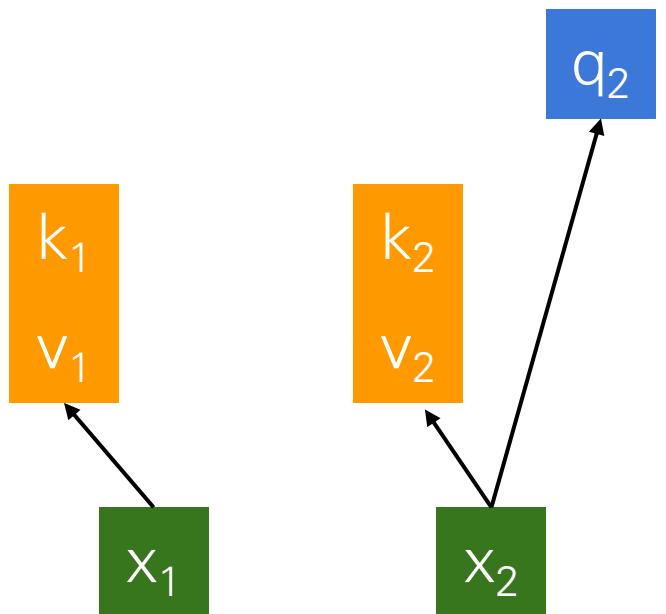
**Key Idea:** Cache the K / V for all attention layers each sampling step

$x_1$

$x_2$

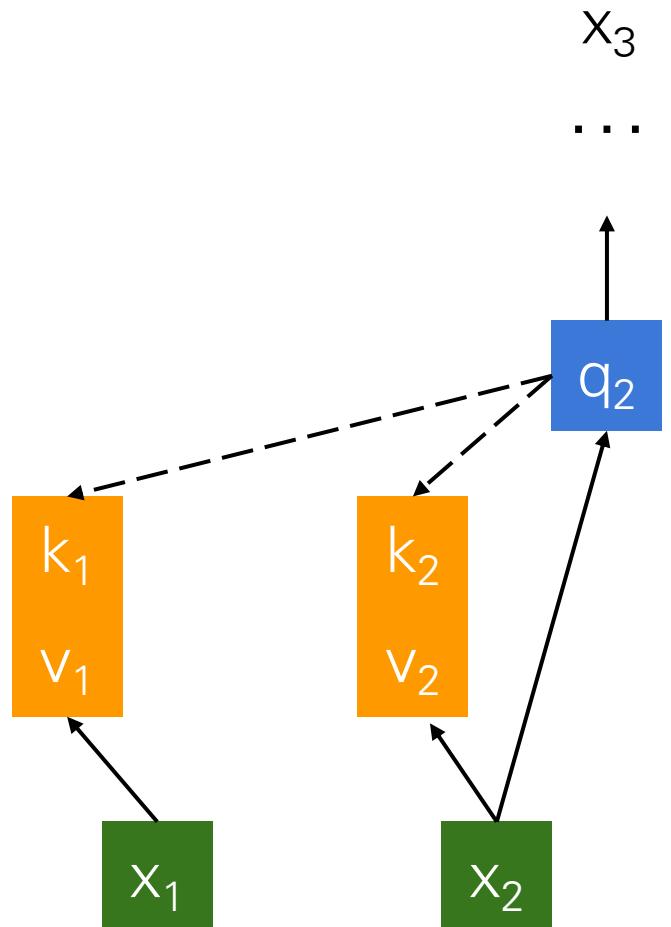
# Caching / Efficient Sampling

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



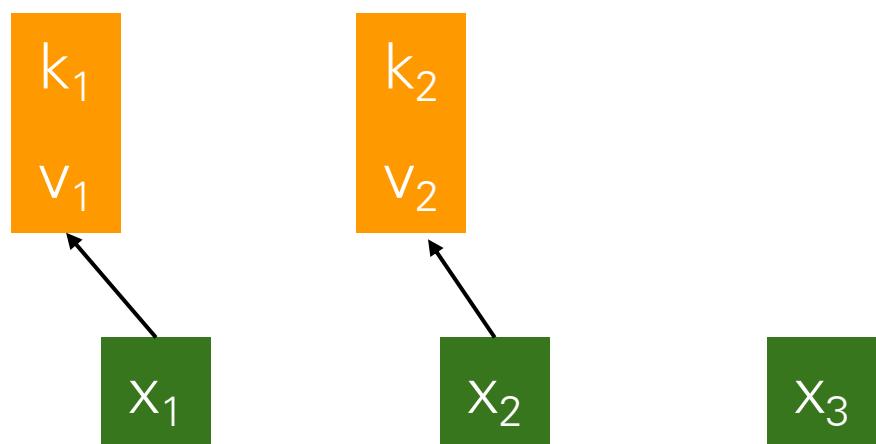
# Caching / Efficient Sampling

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



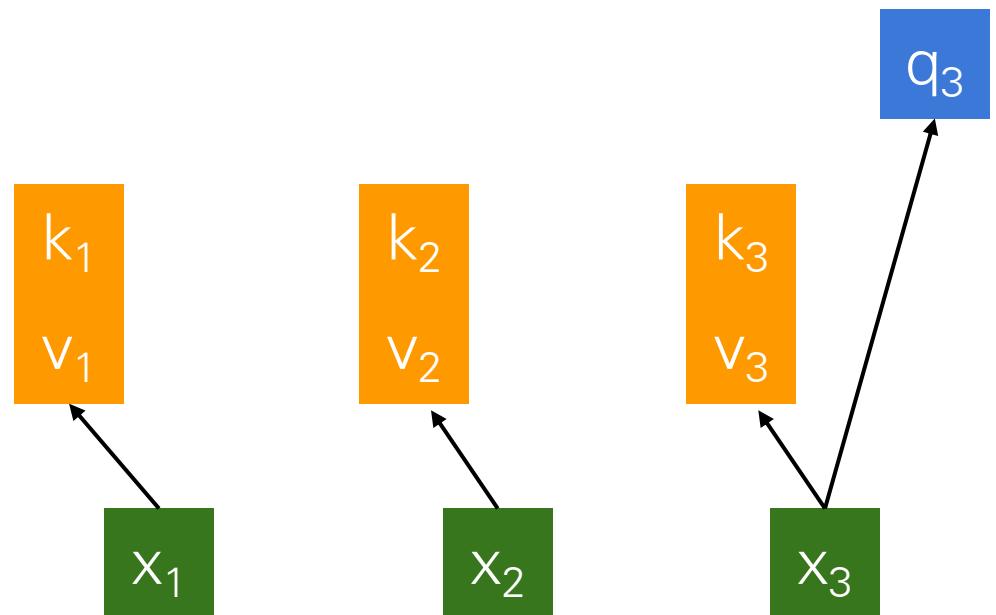
# Caching / Efficient Sampling

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



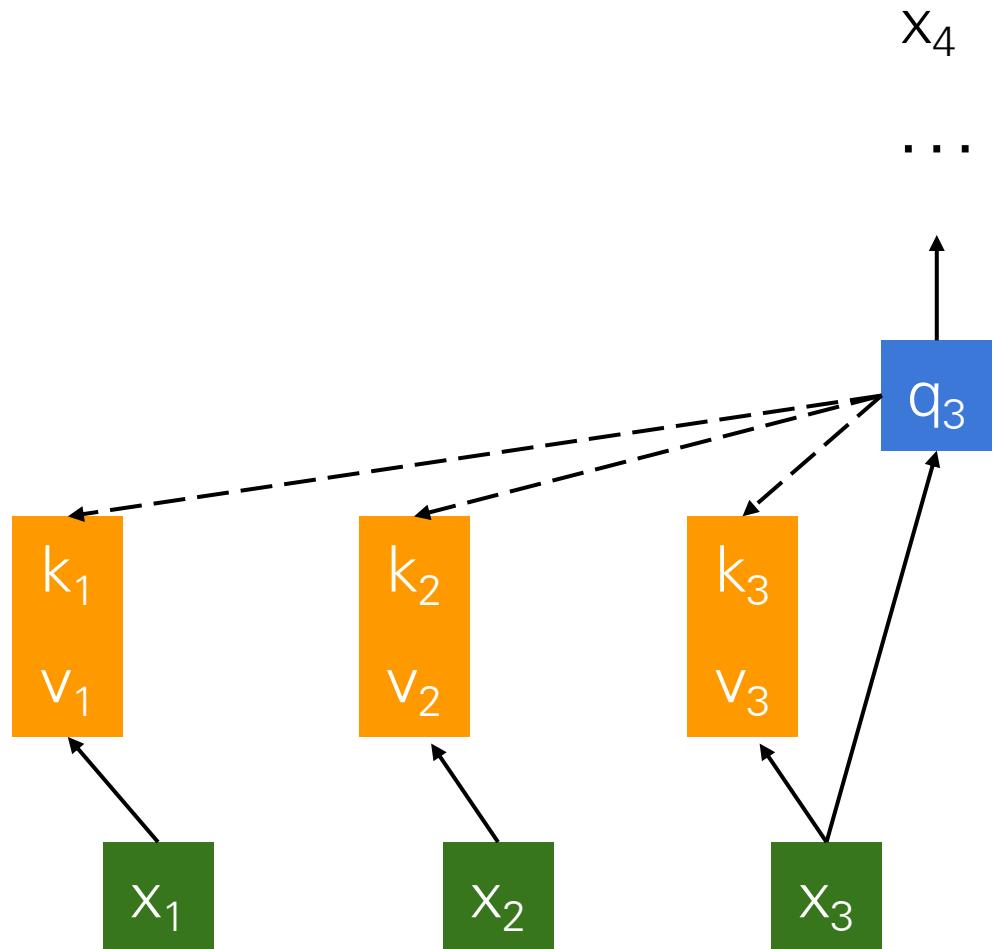
# Caching / Efficient Sampling

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



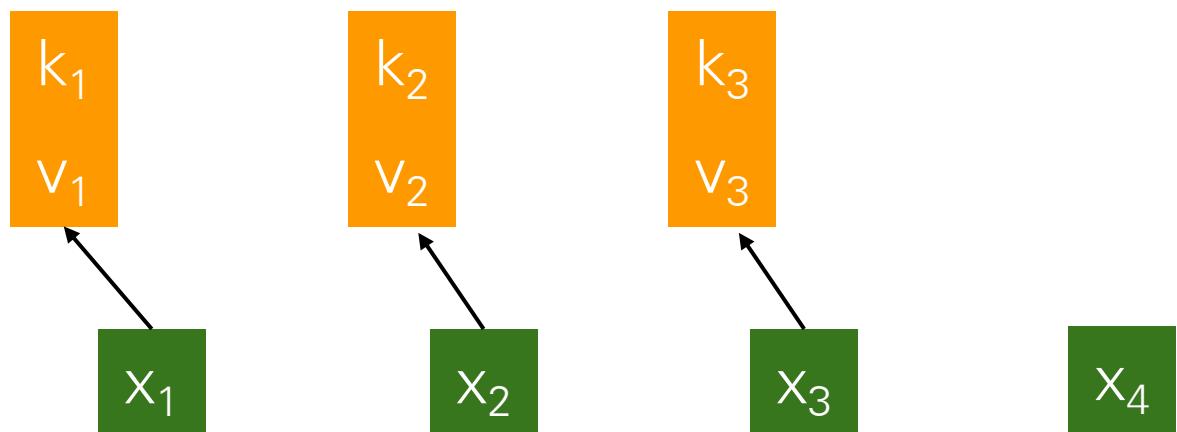
# Caching / Efficient Sampling

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



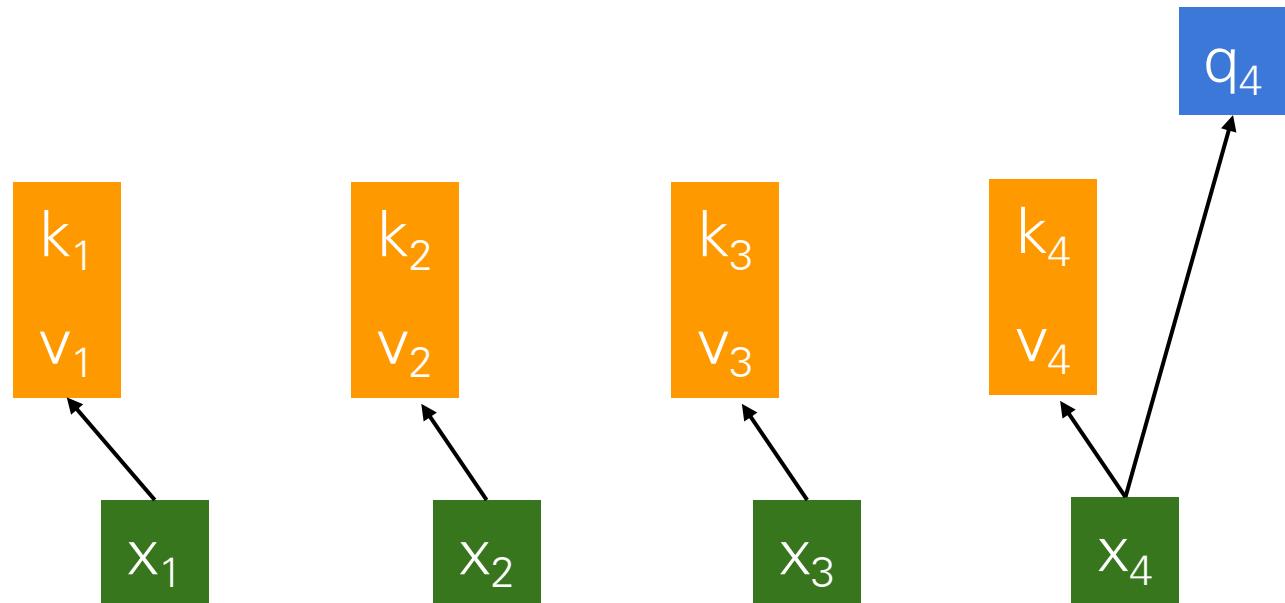
# Caching / Efficient Sampling

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



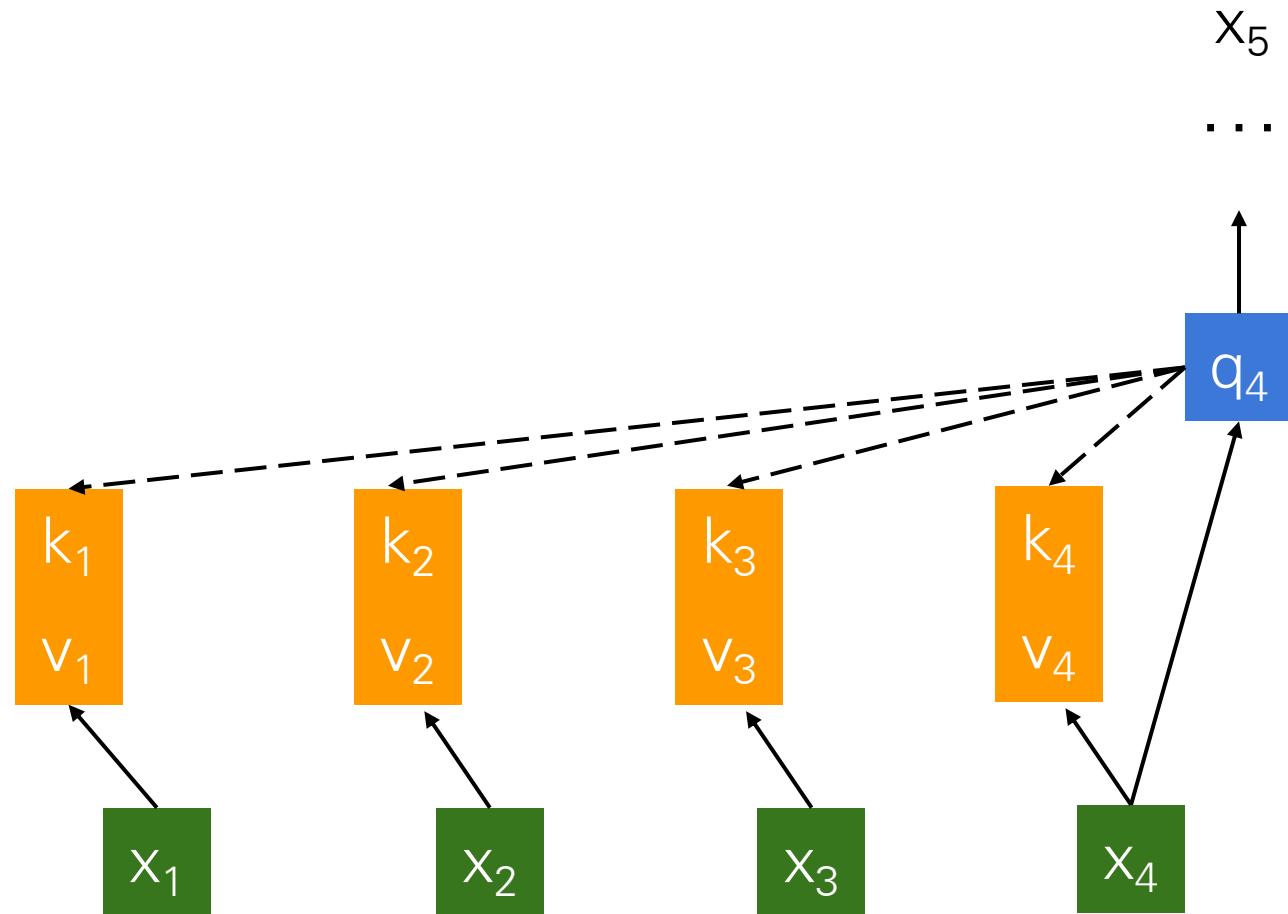
# Caching / Efficient Sampling

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



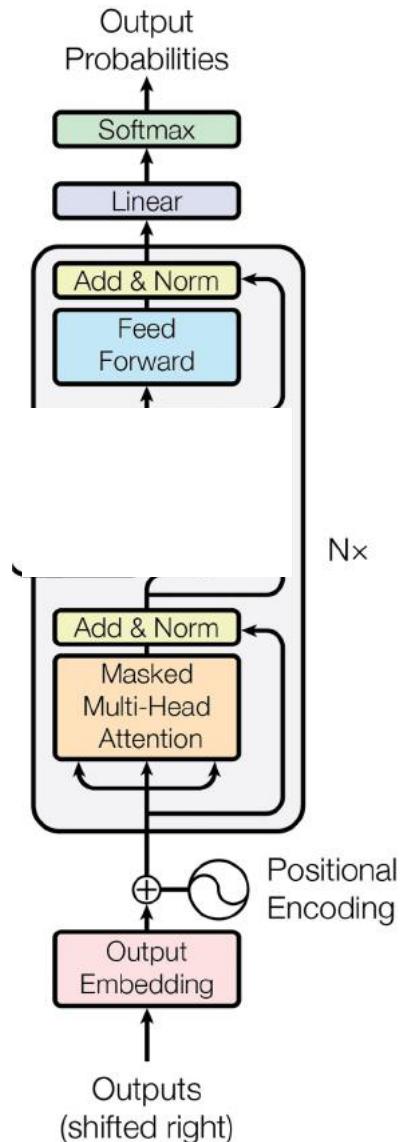
Per-Step

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

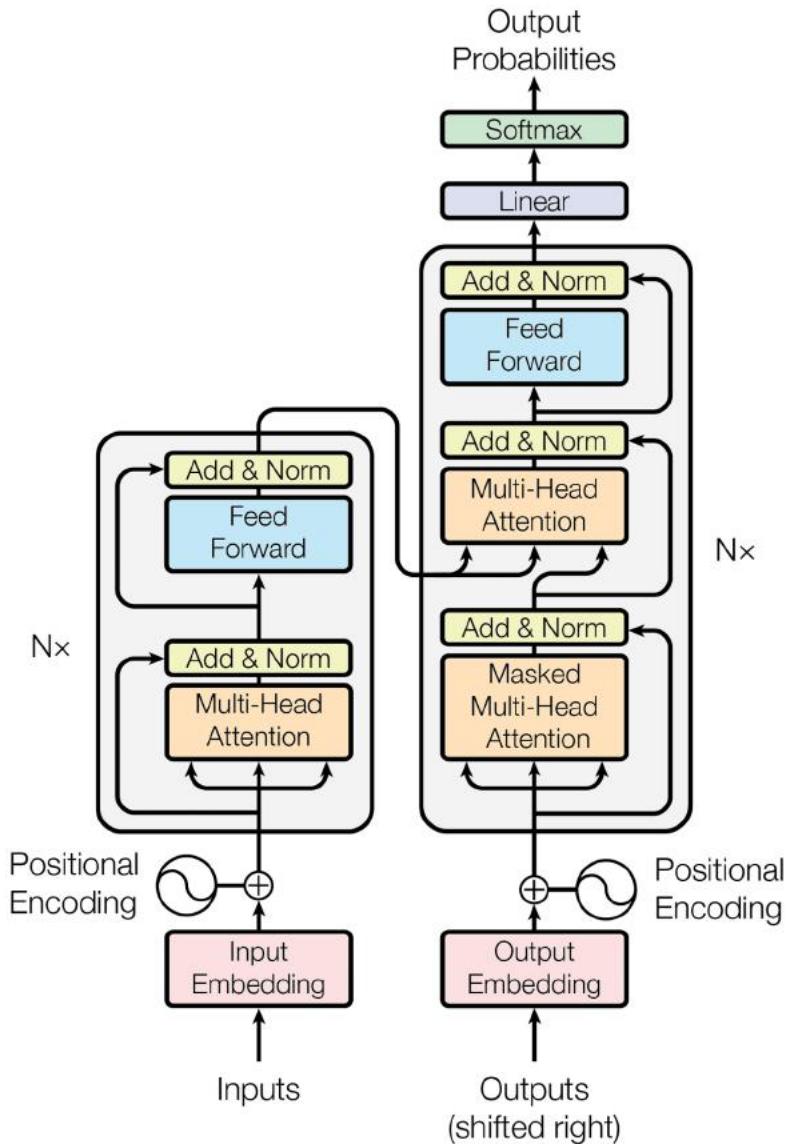
# Lecture overview

- 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



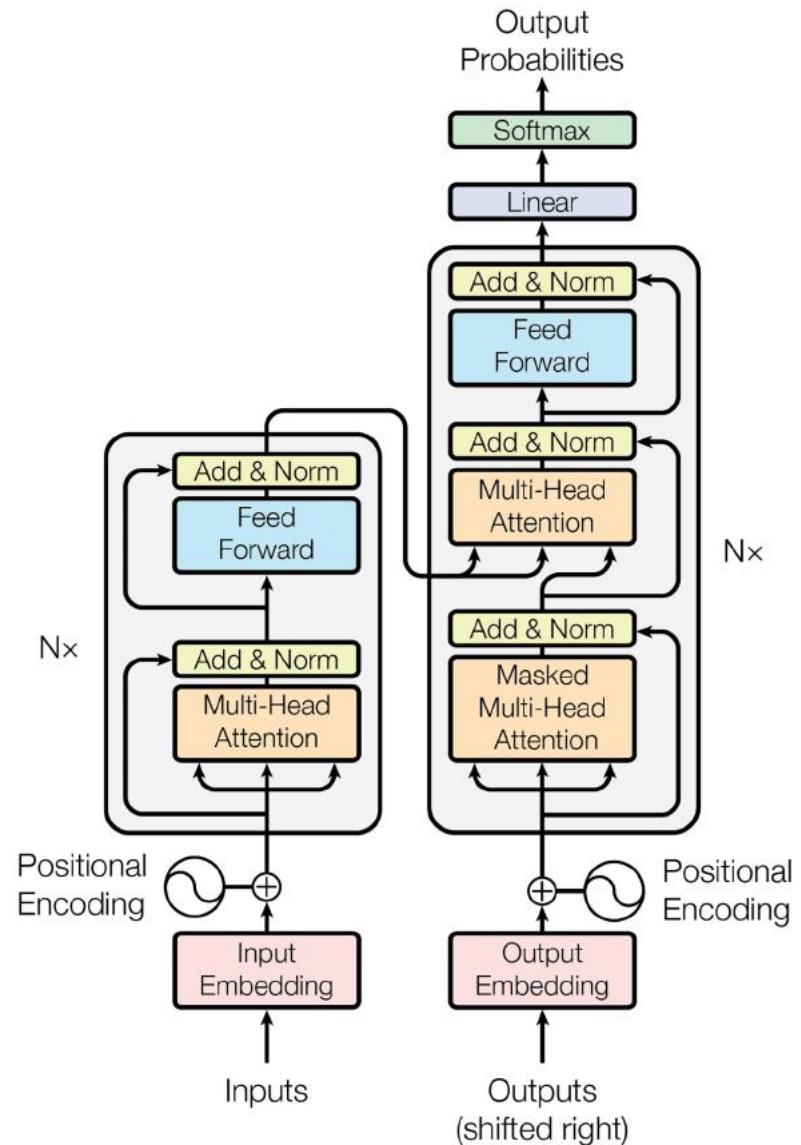
# Encoder-Decoder Models



# 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:

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

$$y_k = \overline{C}x_k$$

- 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:

$$\begin{aligned}x_k &= \overline{\mathbf{A}}x_{k-1} + \overline{\mathbf{B}}u_k \\y_k &= \overline{\mathbf{C}}x_k\end{aligned}$$

- Unroll the recurrence

$$\begin{array}{lll}x_0 = \overline{\mathbf{B}}u_0 & x_1 = \overline{\mathbf{AB}}u_0 + \overline{\mathbf{B}}u_1 & x_2 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{AB}}u_1 + \overline{\mathbf{B}}u_2 \\y_0 = \overline{\mathbf{CB}}u_0 & y_1 = \overline{\mathbf{CAB}}u_0 + \overline{\mathbf{CB}}u_1 & y_2 = \overline{\mathbf{CA}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{CAB}}u_1 + \overline{\mathbf{CB}}u_2\end{array}$$

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

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

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

# 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

$$V'_i = \frac{\sum_{j=1}^N \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^N \text{sim}(Q_i, K_j)}.$$
$$\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

- 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  $\phi(x)$ , write the attention as:

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

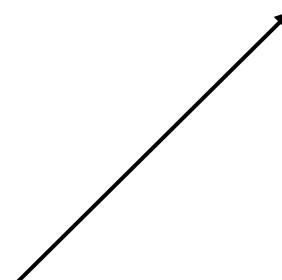
$$(\phi(Q) \phi(K)^T) V = \phi(Q) (\phi(K)^T V)$$

# Causal Linear Attention

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

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

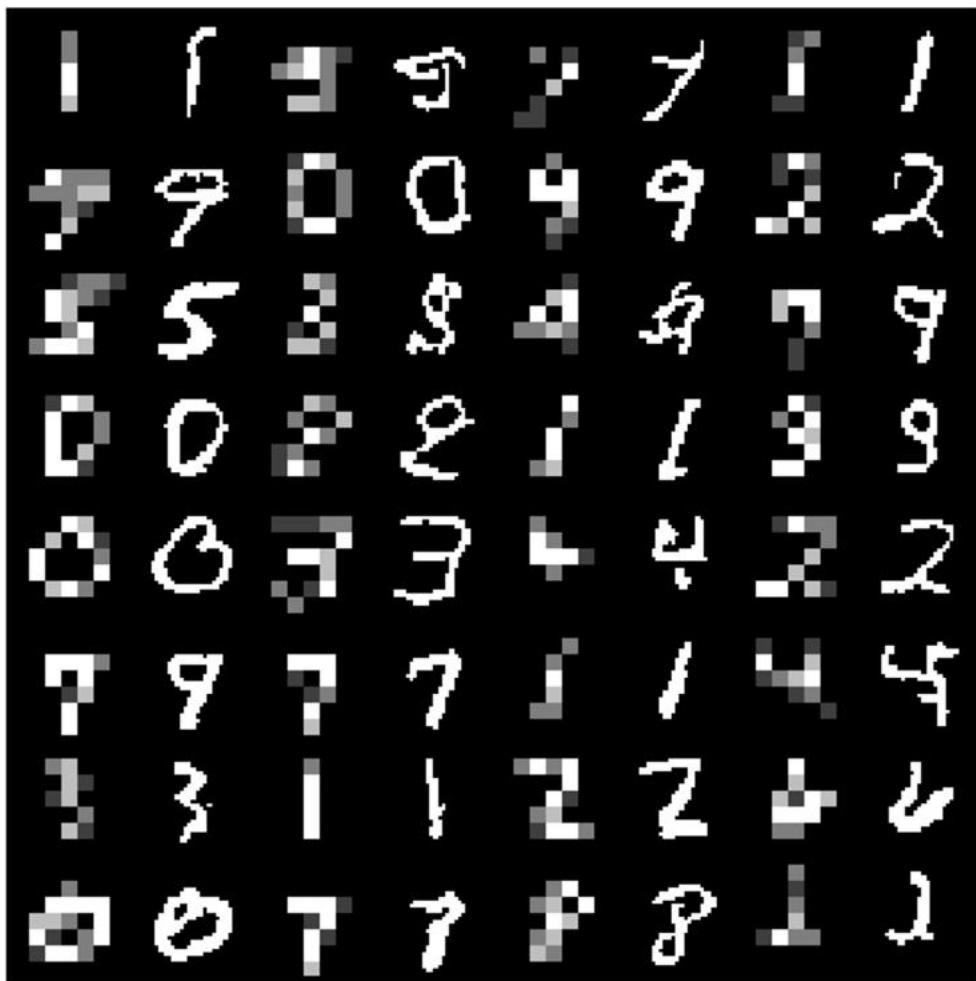
Recurrent!



# Lecture overview

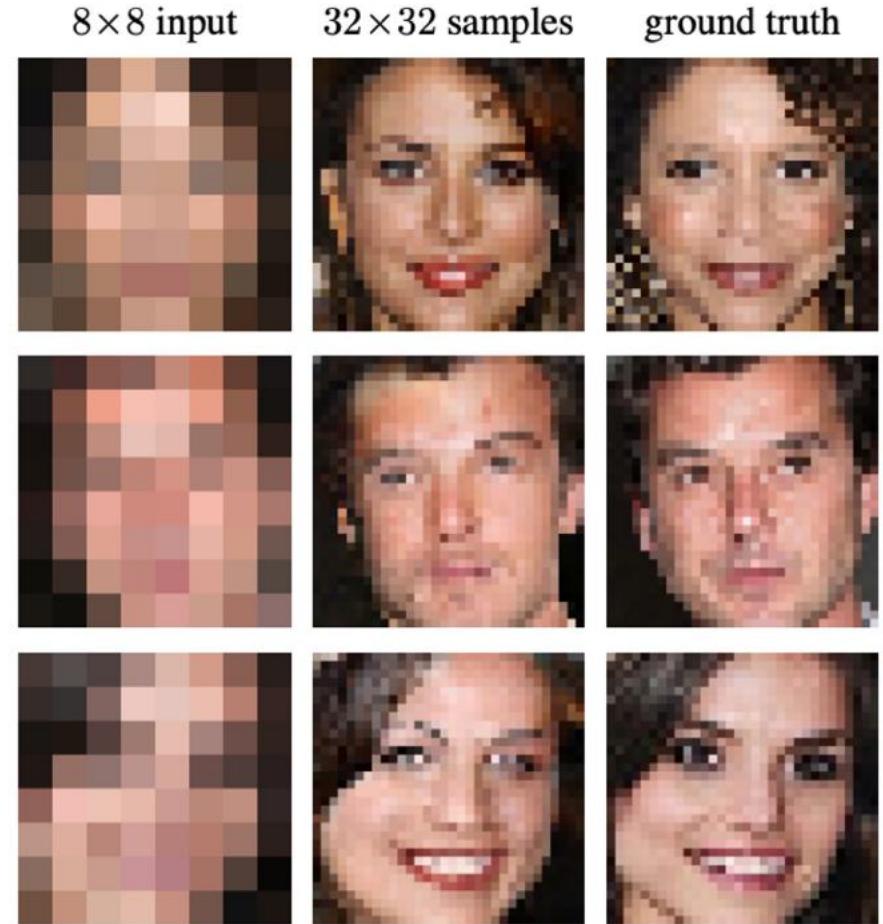
- motivation
- 1-dimensional distributions
- high-dimensional distributions
- deeper dive into causal masked neural models
- other things to be aware of
  - decoder-only vs. Encoder-Decoder models
  - new incarnations of recurrent models
  - alternative / complementary ideas to tokenization

# Super-Resolution with PixelCNN

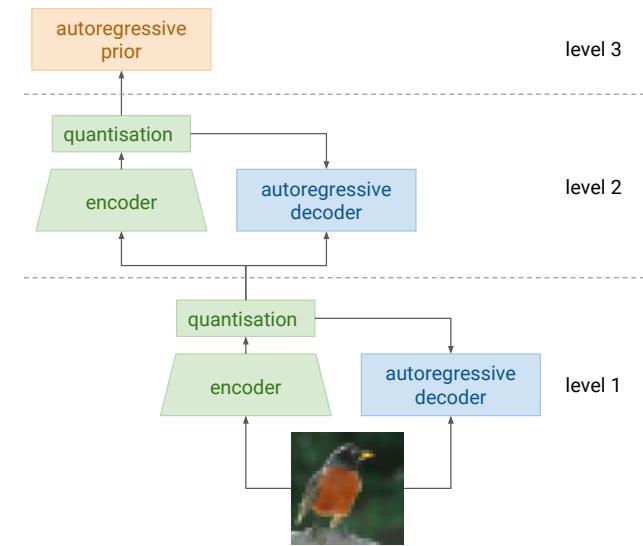


- A PixelCNN is conditioned on  $7 \times 7$  subsampled MNIST images to generate the corresponding  $28 \times 28$  image

# Super-Resolution with PixelCNN



# Hierarchical Autoregressive Models with Auxiliary Decoders



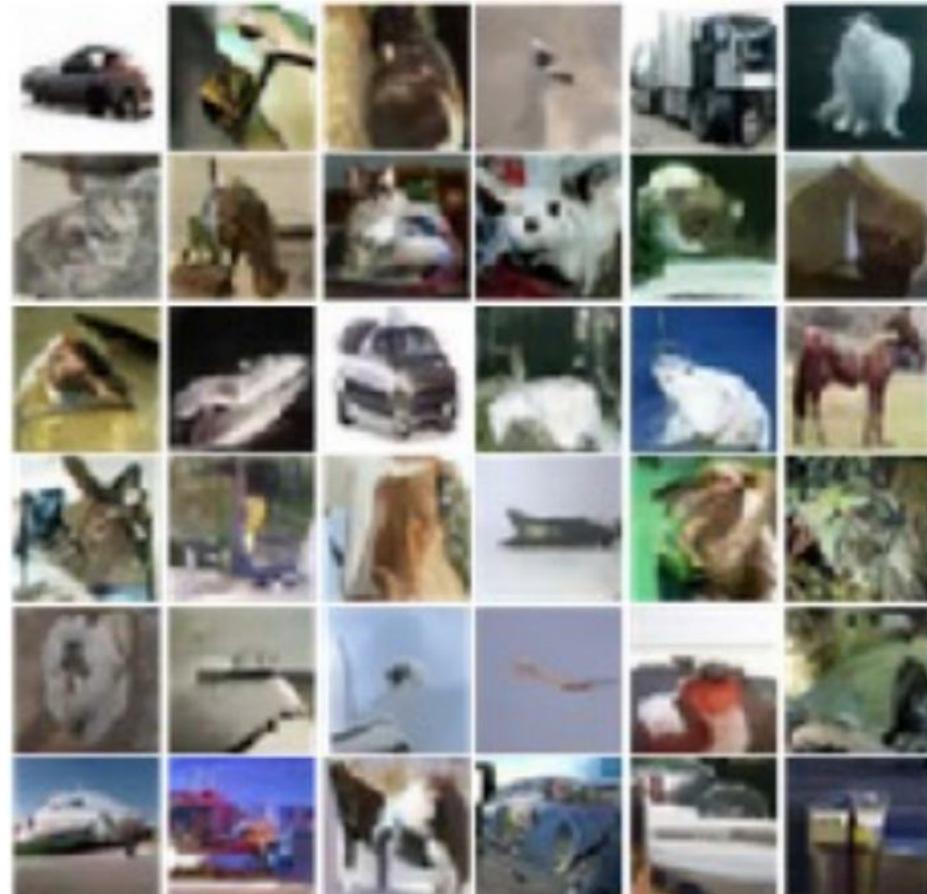
De Fauw, Jeffrey, Sander Dieleman, and Karen Simonyan. "Hierarchical autoregressive image models with auxiliary decoders." arXiv preprint arXiv:1903.04933 (2019).

# 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

# Grayscale PixelCNN



**Next lecture:  
Flow-Based Models**