ROBOT INTELLIGENCE LAB

Generative Model

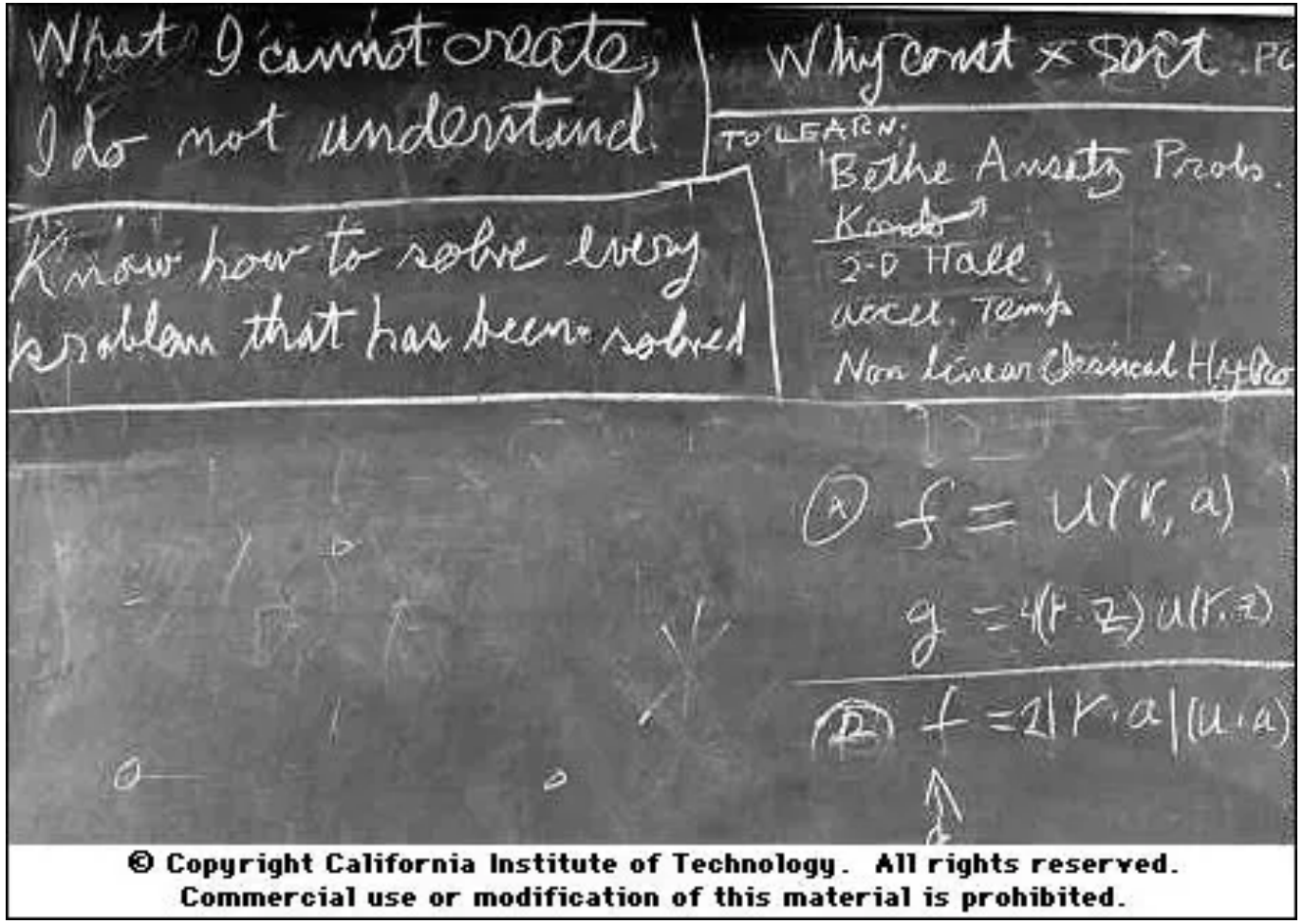
Autoregressive Model & Maximum Likelihood Learning

Sungjoon Choi, Korea University

kakaobrain

Insights or ~~too Cliche~~

ROBOT INTELLIGENCE LAB



Richard Feynman
(1918~1988)

Introduction

ROBOT INTELLIGENCE LAB

Deep Generative Models

CS236 - Fall 2019

Course Description

Generative models are widely used in many subfields of AI and Machine Learning. Recent advances in parameterizing these models using deep neural networks, combined with progress in stochastic optimization methods, have enabled scalable modeling of complex, high-dimensional data including images, text, and speech. In this course, we will study the probabilistic foundations and learning algorithms for deep generative models, including variational autoencoders, generative adversarial networks, autoregressive models, and normalizing flow models. The course will also discuss application areas that have benefitted from deep generative models, including computer vision, speech and natural language processing, graph mining, and reinforcement learning.

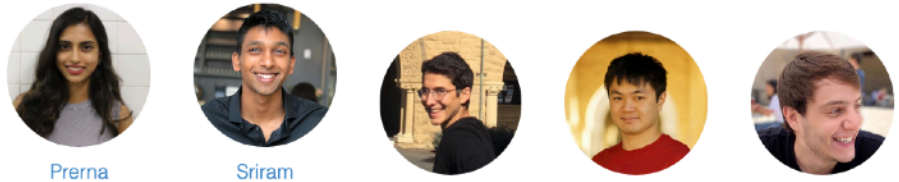
Course Notes | Syllabus | Piazza | Office Hours | Poster Session

Course Instructors

Stefano Ermon | Aditya Grover

Course Assistants

Kristy Choi | Yang Song | Rui Shu | Amaury Sabran | Kaidi Cao



Prerna Dhareshwar | Sriram Somasundaram | Arnaud Autef | Xingyu Liu | Kevin Zakka

deepgenerativemodels.github.io

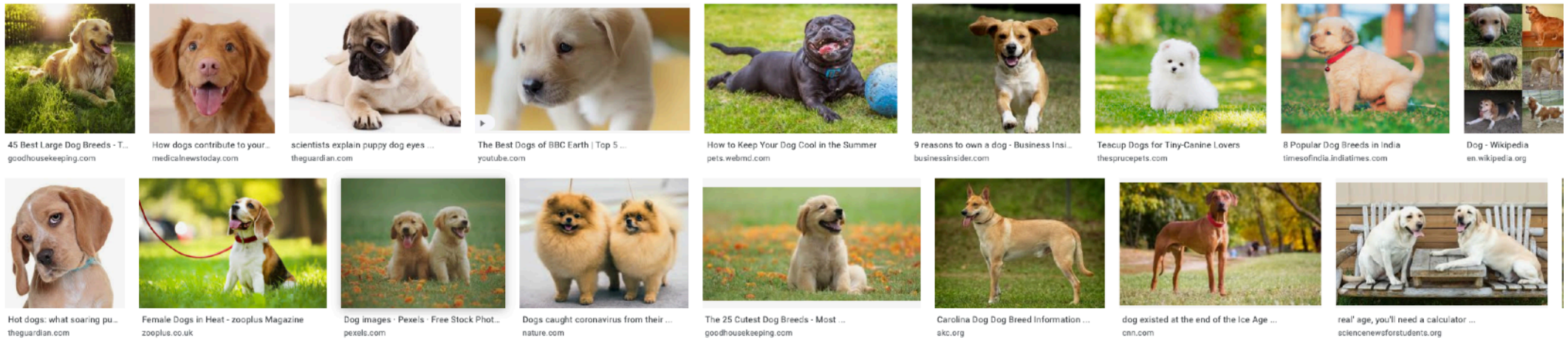
Introduction

ROBOT INTELLIGENCE LAB

What does it mean to learn a generative model?

Learning a Generative Model

ROBOT INTELLIGENCE LAB



Google Search: Dog

- Suppose that we are given images of dogs.

Learning a Generative Model

- Suppose that we are given images of dogs.
- We want to learn a probability distribution $p(\mathbf{x})$ such that
  - **Generation**: If we sample $\mathbf{x}_{new} \sim p(\mathbf{x})$, then $\mathbf{x}_{new}$ should look like a dog (aka sampling).
  - **Density estimation**: $p(\mathbf{x})$ should be high if $\mathbf{x}$ looks like a dog, and low otherwise (aka anomaly detection).
  - **Unsupervised representation learning**: We should be able to learn what these images have in common, e.g., ears, tail, etc (aka feature learning).
- Then, how can we represent $p(\mathbf{x})$?

Learning a Generative Model

ROBOT INTELLIGENCE LAB

- Three important questions
  - What is the representation for the model family $\mathscr{M}$?
  - What is the objective function $d(\cdot)$?
  - What is the optimization procedure for minimizing $d(\cdot)$?

Basic Discrete Distributions

ROBOT INTELLIGENCE LAB

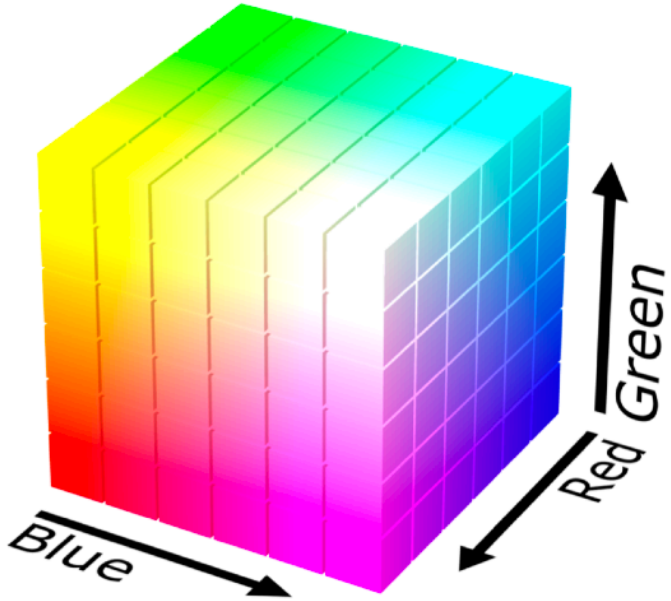
- Bernoulli distribution: coin flip
  - $D = \{\text{Heads}, \text{Tails}\}$
  - Specify $P(X = \text{Heads}) = p$. Then, $P(X = \text{Tails}) = 1 - p$.
  - Write: $X \sim \text{Ber}(p)$

- Categorical distribution: m-sided dice
  - $D = \{1, \cdots, m\}$
  - Specify $P(Y = i) = p_i$, such that $\sum_{i=1}^{m} p_i = 1$.
  - Write: $Y \sim \text{Cat}(p_1, \cdots, p_m)$

Example

ROBOT INTELLIGENCE LAB

- Modeling an RGB joint distribution (of a single pixel)
  - $(r, g, b) \sim p(R, G, B)$
  - Number of cases?
    - $256 \times 256 \times 256$
  - How many parameters do we need to specify?
    - $255 \times 255 \times 255$



https://en.wikipedia.org/wiki/RGB_color_space

Example

- Suppose we have $X_1, \ldots, X_n$ of n binary pixels (i.e., a binary image).
  - Number of cases?
    - $2 \times 2 \times \cdots \times 2 = 2^n$
  - How many parameters do we need to specify $P(X_1, \ldots, X_n)$?
    - $2^n - 1$

Structure Through Independence

ROBOT INTELLIGENCE LAB

- What happens if $X_1, \ldots, X_n$ are **independent**?
  - $p(x_1, \ldots, x_n) = p(x_1)p(x_2)\cdots p(x_n)$
  - Number of cases?
    - 2^n
  - How many parameters do we need to specify $p(x_1, \ldots, x_n)$?
    - n
  - What it mean is that 2^n **entries can be described by just** n **numbers**!
  - In other words, this **independence assumption** is too strong to model useful distributions.

Conditional Independence

ROBOT INTELLIGENCE LAB

- Three important rules
  - **Chain rule**
    - $p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$
  - **Bayes' rule**
    - $p(x \mid y) = \dfrac{p(x, y)}{p(y)} = \dfrac{p(y \mid x)p(x)}{p(y)}$
  - **Conditional independence**
    - Two events A and B are conditionally independent given event C if $p(A \cap B \mid C) = p(A \mid C)p(B \mid C)$.
    - If $x \perp y \mid z$, then $p(x \mid y, z) = p(x \mid z)$.

Conditional Independence

ROBOT INTELLIGENCE LAB

- Using the **chain rule**,
  - $p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \cdots, x_{n-1})$

- How many parameters do we need to specify $p(x_1, \ldots, x_n)$?
  - $p(x_1)$: 1 parameter
  - $p(x_2 \mid x_1)$: 2 parameters (one per $p(x_2 \mid x_1 = 0)$ and one per $p(x_2 \mid x_1 = 1)$)
  - $p(x_3 \mid x_1, x_2)$: $2^2 = 4$ parameters
  - Hence, $1 + 2 + 2^2 + \cdots 2^{n-1} = 2^n - 1$, which is the same as before.

- Why?

Conditional Independence

- Now, suppose $X_{i+1} \perp X_1, \ldots, X_{i-1} \mid X_i$ (aka **Markov assumption**), then
  - $p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2) \cdots p(x_n \mid x_{n-1})$

- How many parameters do we need to specify $p(x_1, \ldots, x_n)$?
  - 1 for $p(x_1)$ and 2 for $p(x_{i+1} \mid x_i)$ where $i = 1,2,\ldots, n-1$
  - Hence, $1 + 2(n-1) = 2n - 1$

  - In other words, by leveraging the **Markov assumption**, we get **an exponential reduction** on the number of parameters.
  - **Auto-regressive models** leverage this **conditional independency**.

ROBOT INTELLIGENCE LAB

Auto-regressive Model

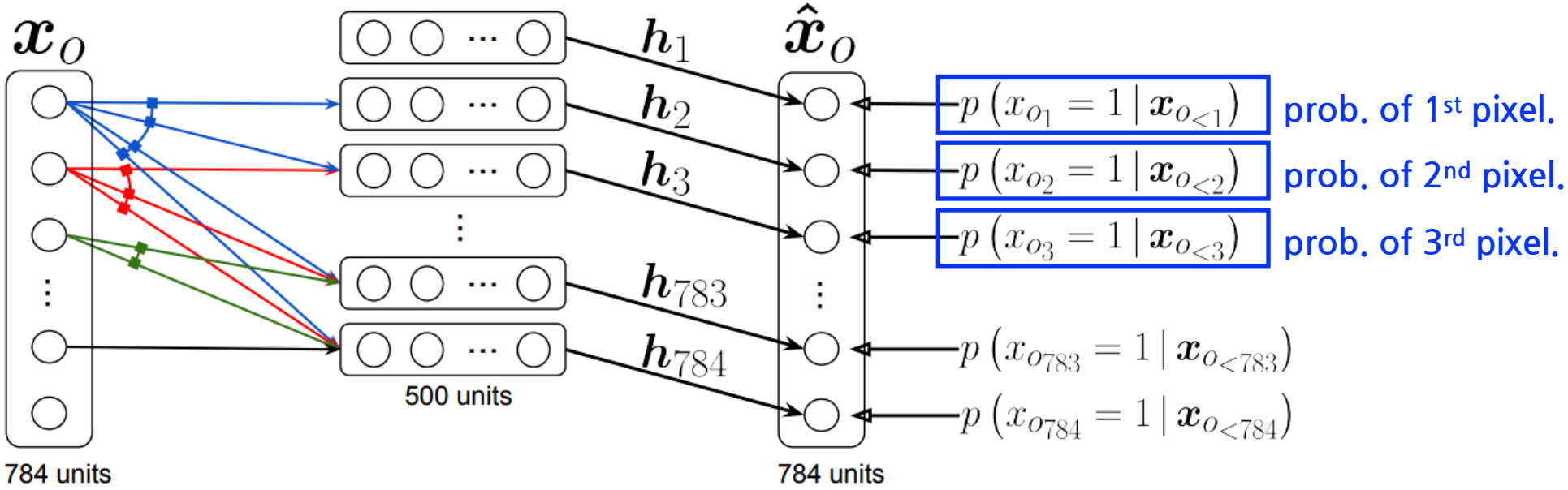
Auto-regressive Model

ROBOT INTELLIGENCE LAB

- Suppose we have 28×28 binary pixels.
- Our goal is to learn $p(x) = p(x_1, \ldots, x_{784})$ over $x \in \{0,1\}^{784}$.
- Then, how can we parametrize $p(x)$?
  - Let's use **the chain rule** to factor the joint distribution. .
  - In other words,
    - $p(x_{1:784}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_{1:2}) \cdots$
    - This is called an **autoregressive model**.
    - Note that we need **an ordering** of all random variables.

NADE: Neural Autoregressive Density Estimator

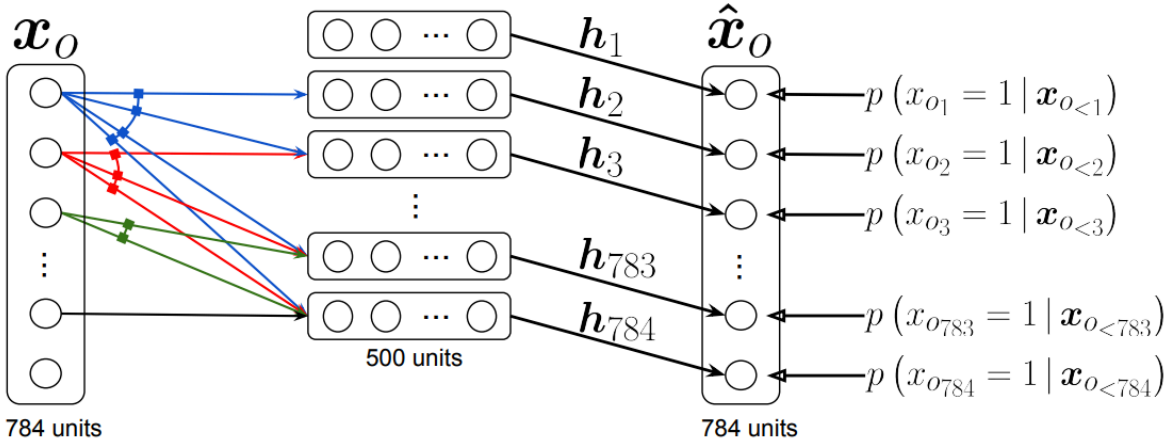
ROBOT INTELLIGENCE LAB



- The probability distribution of i-th pixel is
  - $p(x_i \mid x_{1:i-1}) = \sigma(\alpha_i \mathbf{h}_i + b_i)$ where $\mathbf{h}_i = \sigma(W_{<i} x_{1:i-1} + \mathbf{c})$

https://arxiv.org/abs/1605.02226

NADE: Neural Autoregressive Density Estimator



- **Tie (or reuse) weights** to reduce the number of parameters and speed up computation.
- For example

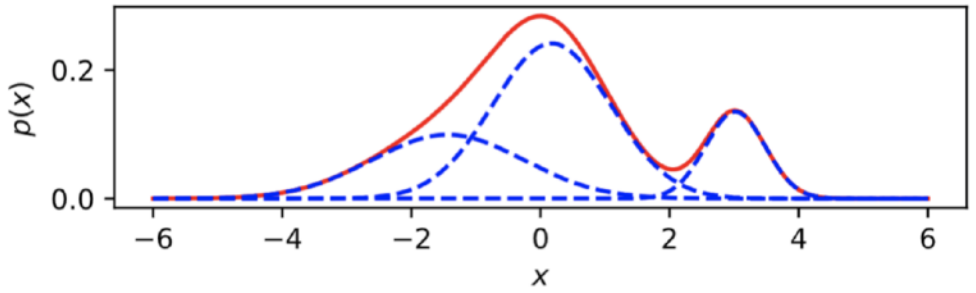
$$\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \\ w_1 \\ \vdots \end{pmatrix}}_{A_2} x_1\right) \quad \mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ w_1 & w_2 \\ \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) \quad \mathbf{h}_4 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$$

https://arxiv.org/abs/1605.02226

NADE: Neural Autoregressive Density Estimator

- **NADE** is an **explicit** model that can compute the **density** of the given inputs.
- BTW, how can we compute the **density** of the given image?
  - Suppose that we have a binary image with 784 binary pixels (i.e., $\{x_1, x_2, \ldots, x_{784}\}$).
  - Then, the joint probability is computed by
    - $p(x_1, \ldots, x_{784}) = p(x_1)p(x_2 \mid x_1) \cdots p(x_{784} \mid x_{1:783})$ where each conditional probability $p(x_i \mid x_{1:i-1})$ is computed independently.

- In case of modeling continuous random variables, a mixture of Gaussian (MoG) can be used.



https://arxiv.org/abs/1605.02226

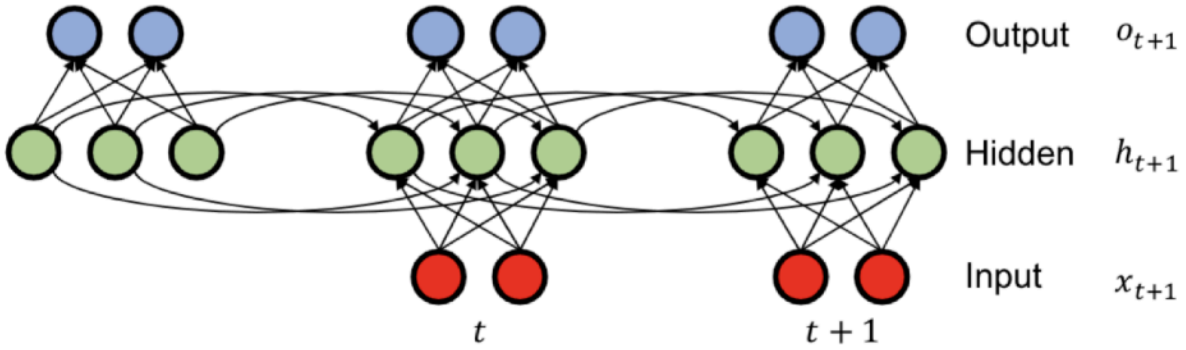
RNN: Recurrent Neural Network

ROBOT INTELLIGENCE LAB

- **Challenge**: when modeling $p_\theta(x_t \mid x_{1:t-1})$, the history $x_{1:t-1}$ keeps getting longer.

$$\mathbf{h}_2 = \sigma\left(\underbrace{\begin{pmatrix} \vdots \\ w_1 \\ \vdots \end{pmatrix}}_{A_2} x_1\right) \quad \mathbf{h}_3 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ w_1 & w_2 \\ \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) \quad \mathbf{h}_4 = \sigma\left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$$

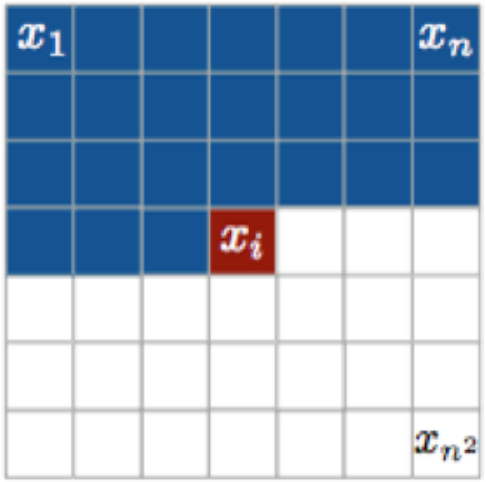
- **Idea**: keep a summary and recursively update it.



Pixel RNN

ROBOT INTELLIGENCE LAB

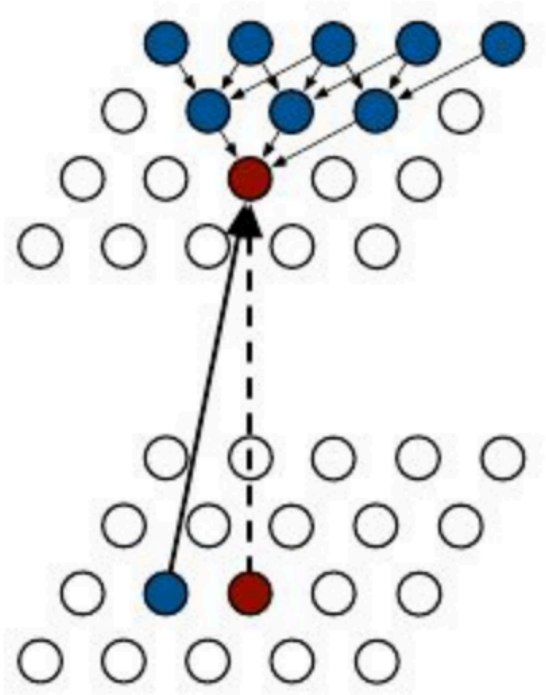
- We can use recurrent neural networks (RNNs) to define an auto-regressive model.
- For example, for an $n \times n$ RGB image,
  - $p(x) = \Pi_{i=1}^{n^2} p(x_{i,R} \mid x_{<i}) p(x_{i,G} \mid x_{<i}, x_{i,R}) p(x_{i,B} \mid x_{<i}, x_{i,R}, x_{i,G})$
- There are two model architectures in Pixel RNN based on the **ordering** of the chain:
  - Row LSTM
  - Diagonal BiLSTM



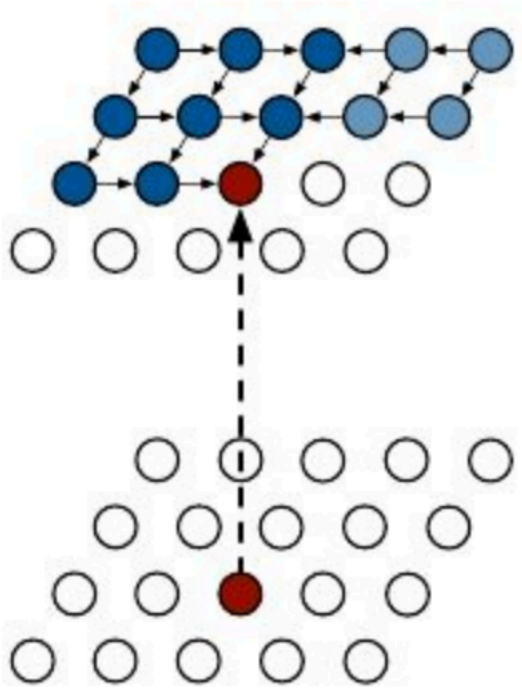
https://arxiv.org/abs/1601.06759

Pixel RNN

ROBOT INTELLIGENCE LAB



Row LSTM



Diagonal BiLSTM

https://arxiv.org/abs/1601.06759

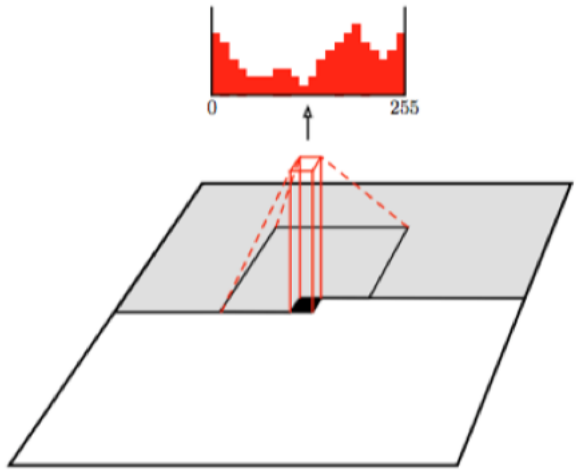
Pixel RNN

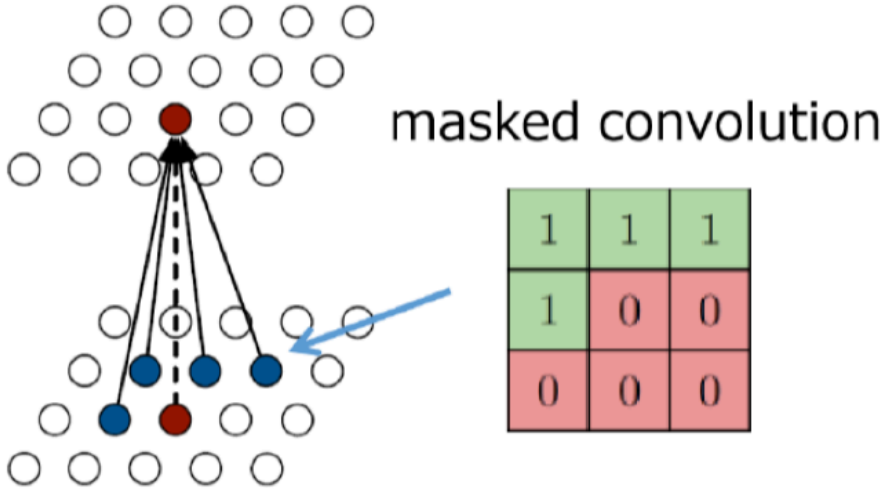
ROBOT INTELLIGENCE LAB

- Results on downsampled ImageNet.
- It is **very slow** due to the sequential likelihood evaluation.
- Note that the results change all the time.

https://arxiv.org/abs/1601.06759

PixelCNN

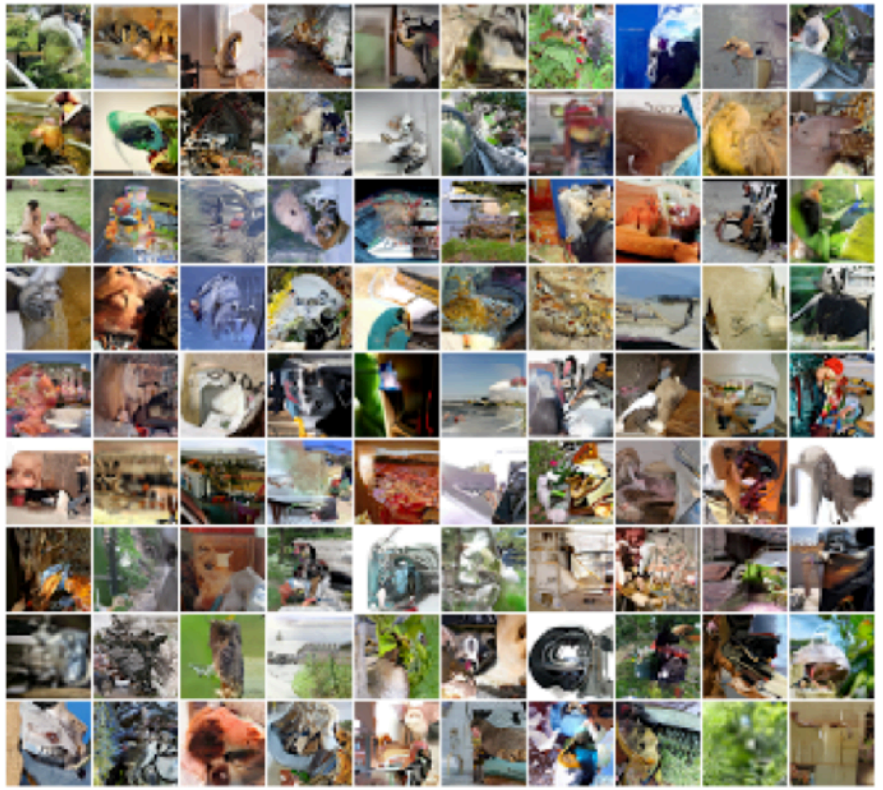




- **Idea**: Use convolutional architecture to predict the next pixel given context (a neighborhood of pixels).
- **Challenge**: It still has to be **autoregressive**. Masked convolutions preserve raster scan order (similar to PixelRNN).

PixelCNN

ROBOT INTELLIGENCE LAB

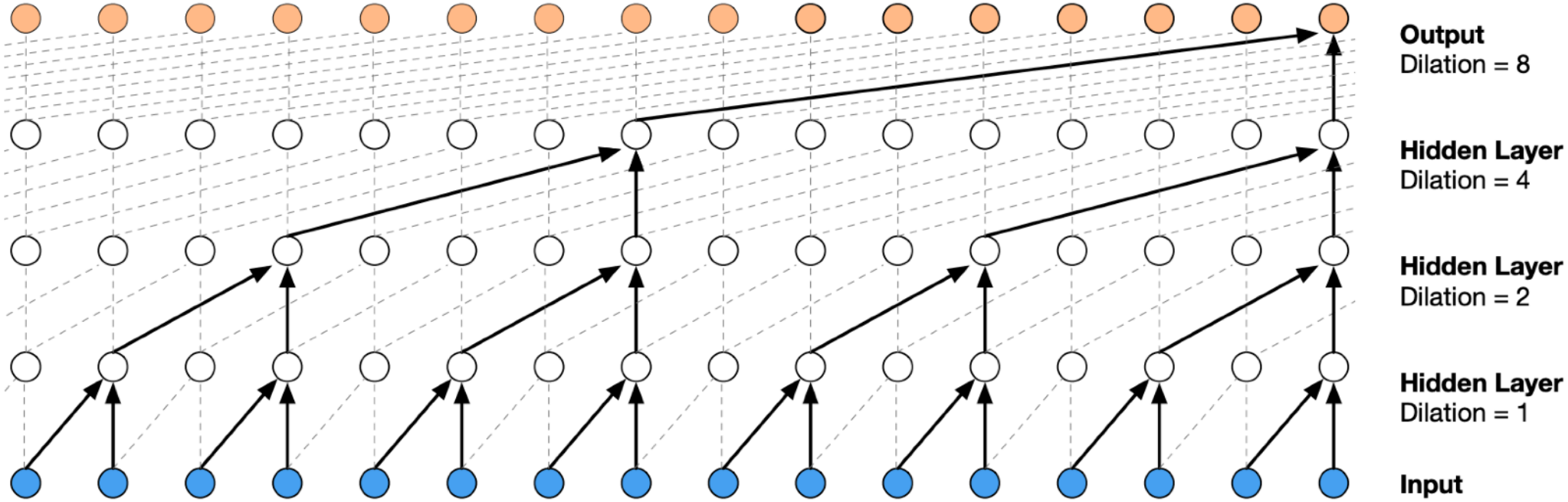


- Similar (or worse) performance to PixelRNN, but **much faster in training**.
- But **generation is still slow** due to its sequential manner.

https://arxiv.org/abs/1606.05328v2

WaveNet

ROBOT INTELLIGENCE LAB



- Dilated convolutions increase the receptive field: kernel only touches the signal at every 2^d entries.

https://arxiv.org/abs/1609.03499

Summary of Autoregressive Models

- Easy to **sample** from
  - Sample $\bar{x}_0 \sim p(x_0)$
  - Sample $\bar{x}_1 \sim p(x_1 \mid x_0 = \bar{x}_0)$
  - ⋯ and so forth (in a sequential manner, hence slow)
- Easy to **compute probability** $p(x = \bar{x})$
  - Compute $p(x_0 = \bar{x}_0)$
  - Compute $p(x_1 = \bar{x}_1 \mid x_0 = \bar{x}_0)$
  - Multiply together (sum their logarithms)
  - ⋯ and so forth
  - Ideally, we can compute all these terms in parallel.
- Easy to be extended to **continuous** variables. For example, we can chose mixture of Gaussians.

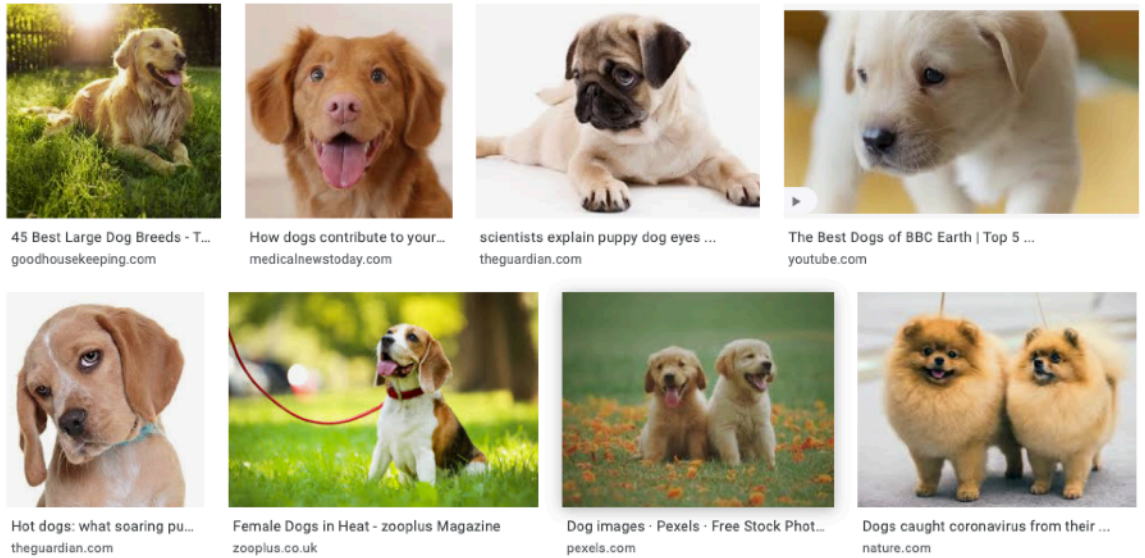
Summary of Autoregressive Models

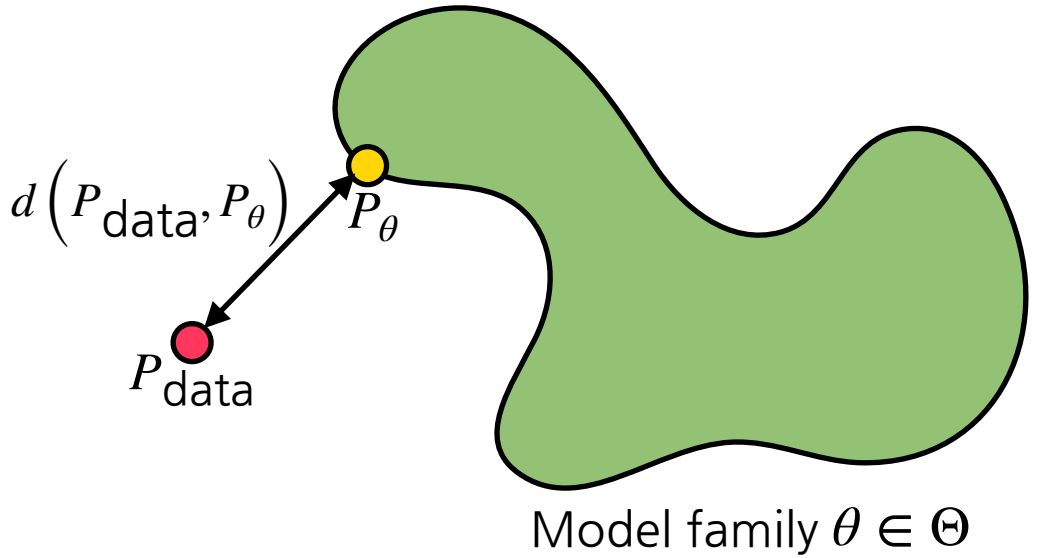
- Hard to **model** as
  - it requires an ordering
  - it cannot learn features in an unsupervised way
- Hard to generate samples **efficiently** as
  - the generation is sequential

ROBOT INTELLIGENCE LAB

Maximum Likelihood Estimation

Maximum Likelihood Learning





- Given a training set of examples, we can cast the generative model learning process as finding the **best-approximating density** model from the **model family**.
- Then, how can we **evaluate** the goodness of the approximation?

Maximum Likelihood Learning

ROBOT INTELLIGENCE LAB

- KL-divergence

$$\mathbf{D}(P_{\text{data}}||P_\theta) = \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log\left(\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}\right)\right] = \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}$$

- We can simplify this with

$$\begin{aligned}\mathbf{D}(P_{\text{data}}||P_\theta) &= \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log\left(\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}\right)\right] \\ &= \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_{\text{data}}(\mathbf{x})\right] - \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_\theta(\mathbf{x})\right]\end{aligned}$$

- As the first term does not depend on P_θ, minimizing the KL-divergence is **equivalent** to maximizing the expected log-likelihood.

$$\arg\min_{P_\theta} \mathbf{D}(P_{\text{data}}||P_\theta) = \arg\min_{P_\theta} -\mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_\theta(\mathbf{x})\right] = \arg\max_{P_\theta} \mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_\theta(\mathbf{x})\right]$$

Maximum Likelihood Learning

ROBOT INTELLIGENCE LAB

- Approximate the expected log-likelihood

$$\mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_\theta(\mathbf{x})\right]$$

with the empirical log-likelihood

$$\mathbf{E}_{\mathcal{D}}\left[\log P_\theta(\mathbf{x})\right] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

- **Maximum likelihood learning** is then:

$$\max_{P_\theta} \ \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

- **Variance** of Monte Carlo estimate is high:

$$V_P[\hat{g}] = V_P\left[\frac{1}{T}\sum_{t=1}^{T} g(x^t)\right] = \frac{V_P[g(x)]}{T}$$

Empirical Risk Minimization

- For maximum likelihood learning, **empirical risk minimization (ERM)** if often used.
- However, **ERM** often suffers from its overfitting.
  - Extreme case: The model remembers all training data

$$p(x) = \frac{1}{|\mathscr{D}|} \sum_{i=1}^{|\mathscr{D}|} \delta(x, x_i).$$

- To achieve better generalization, we typically restrict the **hypothesis space** of distributions that we search over.
- However, it could deteriorate the performance of the generative model.

Maximum Likelihood Learning

- Usually, MLL is prone to under-fitting as we often use simple parametric distributions such as spherical Gaussians.

- What about other ways of measuring the similarity?
  - **KL-divergence** leads to MLL (maximum likelihood learning).
  - **Jensen-Shannon divergence** leads to generative adversarial networks.
  - **Wasserstein distance** leads to Wasserstein auto-encoder (WAE) or adversarial auto-encoder (AAE).

Thank You