

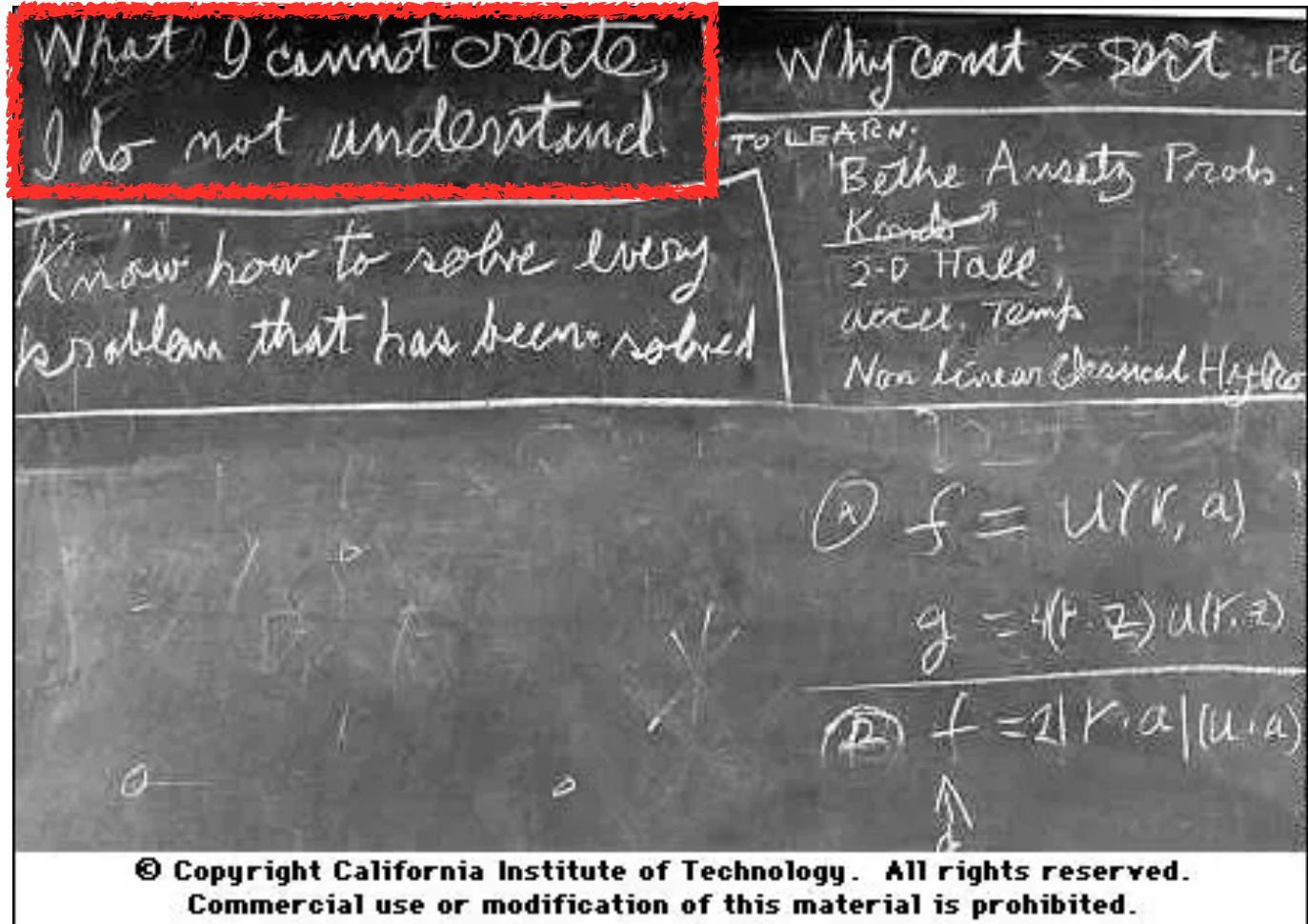
# Deep Learning Basics

## Lecture 9: Generative Models Part 1

**최성준** (고려대학교 인공지능학과)

# Introduction

# Introduction



Richard Feynman  
(1918~1988)

# Introduction

## 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



Kristy Choi



Yang Song



Rui Shu



Amaury Sabran



Kaidi Cao



Perna  
Dhareshwar



Sriram  
Somasundaram



Arnaud Autef



Xingyu Liu



Kevin Zakka

### Course Assistants

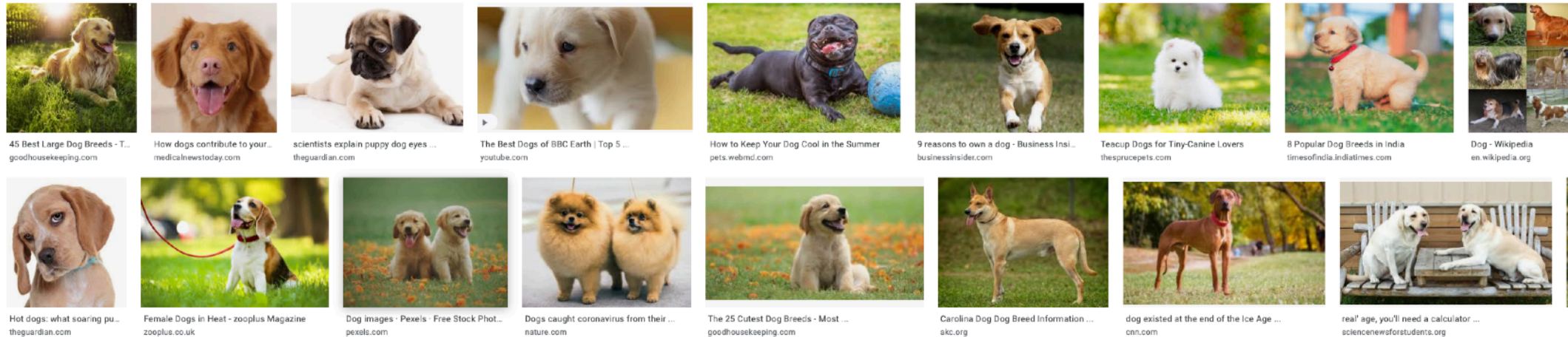
<https://deepgenerativemodels.github.io/>

# Introduction

---

What does it mean to learn a **generative model**?

# Learning a Generative Model



Google Search: Dog

- Suppose we are given images of dogs.

# Learning a Generative Model

---

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

# Basic Discrete Distributions

---

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



# Example

- Modeling an RGB joint distribution (of a single pixel)

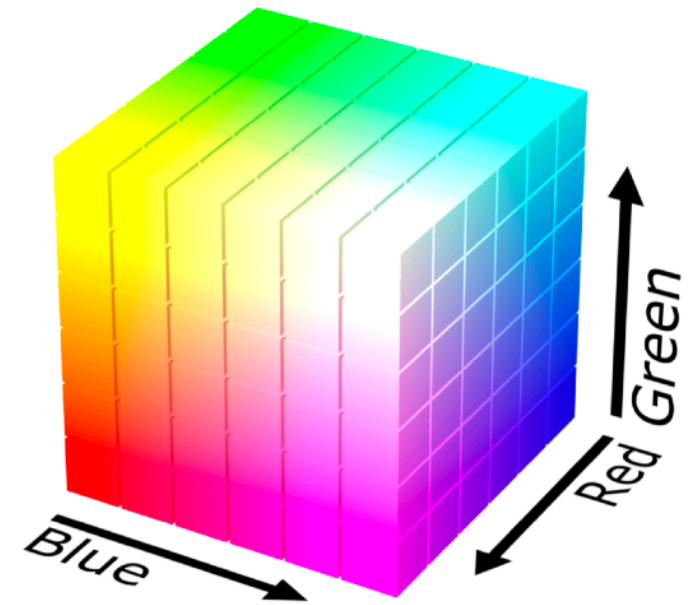
- $(r, g, b) \sim p(R, G, B)$

- 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](https://en.wikipedia.org/wiki/RGB_color_space)

# Example

---



- Suppose we have  $X_1, \dots, X_n$  of  $n$  binary pixels (a binary image).
- How many possible states?

$$2 \times 2 \times \dots \times 2 = 2^n$$

- Sampling from  $p(x_1, \dots, x_n)$  generates an image.
- How many parameters to specify  $p(x_1, \dots, x_n)$ ?

$$2^n - 1$$

# Structure Through Independence

---

- What if  $X_1, \dots, X_n$  are independent, then

$$p(x_1, \dots, x_n) = p(x_1)p(x_2)\cdots p(x_n)$$

- How many possible states?

$$2^n$$

- How many parameters to specify  $p(x_1, \dots, x_n)$ ?

$$n$$

- $2^n$  entries can be described by just  $n$  numbers! But this **independence** assumption is too strong to model useful distributions.

# Conditional Independence

---

- Three important rules

- Chain rule:

$$p(x_1, \dots, x_n) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_n | x_1, \dots, x_{n-1})$$

- Bayes' rule:

$$p(x | y) = \frac{p(x, y)}{p(y)} = \frac{p(y | x)p(x)}{p(y)}$$

- Conditional independence:

$$\text{If } x \perp y | z, \text{ then } p(x | y, z) = p(x | z)$$

# Conditional Independence

---

- Using the chain rule,

$$p(x_1, \dots, x_n) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_n | x_1, \dots, x_{n-1})$$

- How many parameters?

- $p(x_1)$ : 1 parameter
- $p(x_2 | x_1)$ : 2 parameters (one per  $p(x_2 | x_1 = 0)$  and one per  $p(x_2 | x_1 = 1)$ )
- $p(x_3 | x_1, x_2)$ : 4 parameters
- Hence,  $1 + 2 + 2^2 + \dots + 2^{n-1} = 2^n - 1$ , which is the same as before.

- Why?

# Conditional Independence

---

- Now, suppose  $X_{i+1} \perp X_1, \dots, X_{i-1} \mid X_i$  (Markov assumption), then

$$p(x_1, \dots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2) \cdots p(x_n \mid x_{n-1})$$

- How many parameters?

$$2n - 1$$

- Hence, by leveraging the Markov assumption, we get exponential reduction on the number of parameters.
- Auto-regressive models leverage this conditional independency.

# Auto-regressive Model

# Auto-regressive Model

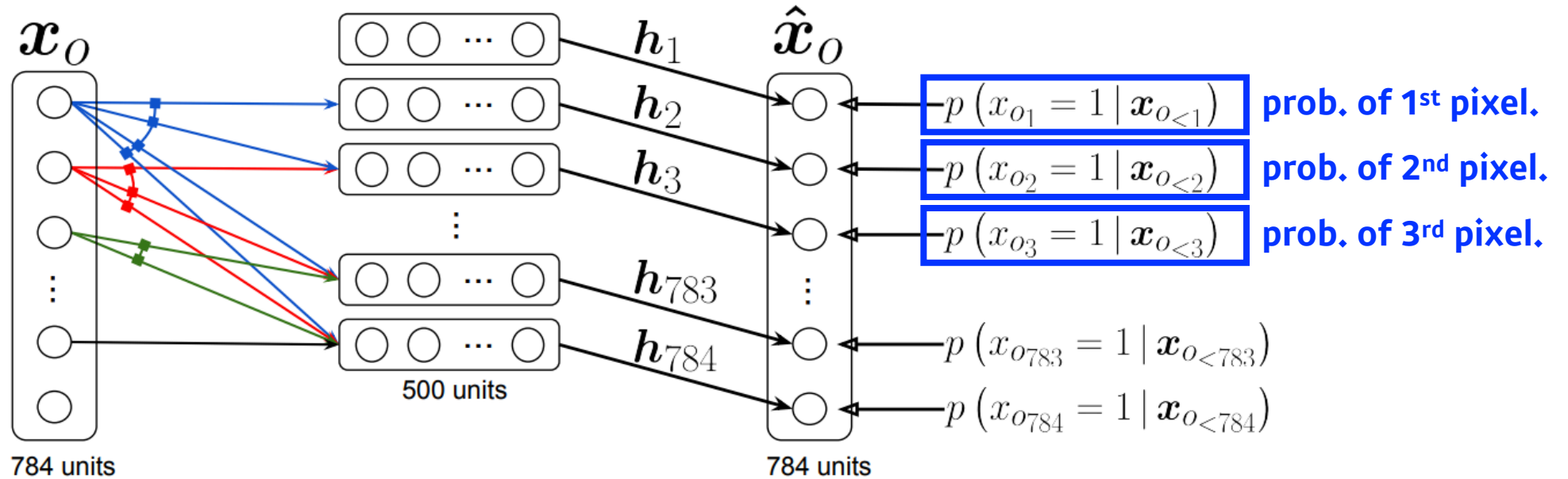
---



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



# NADE: Neural Autoregressive Density Estimator



- The probability distribution of  $i$ -th pixel is

$$p(x_i | x_{1:i-1}) = \sigma(\alpha_i \mathbf{h}_i + b_i) \text{ where } \mathbf{h}_i = \sigma(W_{<i} x_{1:i-1} + \mathbf{c})$$

# NADE: Neural Autoregressive Density Estimator

---

- NADE is an **explicit** model that can compute the **density** of the given inputs.
- How can we compute the **density** of the given image?
  - Suppose we have a binary image with 784 binary pixels,  $\{x_1, x_2, \dots, x_{784}\}$ .
  - Then, the joint probability is computed by

$$p(x_1, \dots, x_{784}) = p(x_1)p(x_2 | x_1) \cdots p(x_{784} | x_{1:783})$$

where each conditional probability  $p(x_i | x_{1:i-1})$  is computed independently.

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

# Pixel RNN

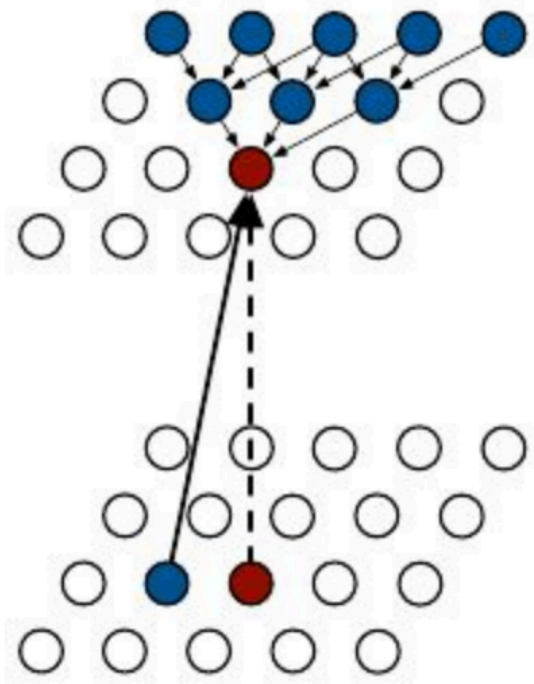
---

- We can also use **RNNs** to define an auto-regressive model.
- For example, for an  $n \times n$  RGB image,

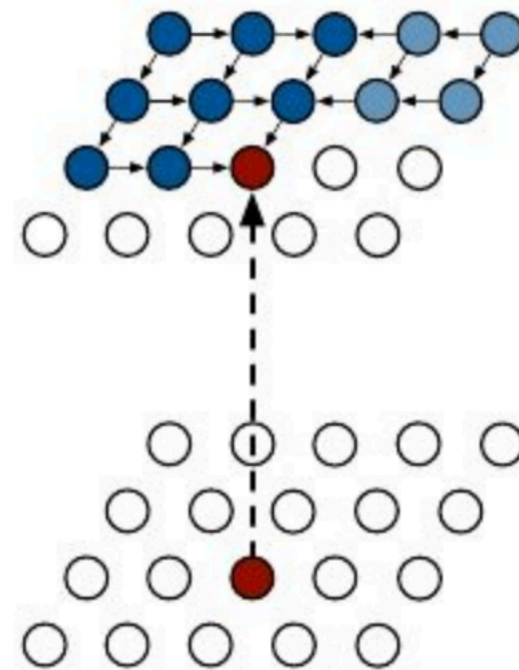
$$p(x) = \prod_{i=1}^{n^2} \underbrace{p(x_{i,R} | x_{<i})}_{\text{Prob. i-th } \mathbf{R}} \underbrace{p(x_{i,G} | x_{<i}, x_{i,R})}_{\text{Prob. i-th } \mathbf{G}} \underbrace{p(x_{i,B} | x_{<i}, x_{i,R}, x_{i,G})}_{\text{Prob. i-th } \mathbf{B}}$$

- There are two model architectures in Pixel RNN based on the **ordering** of chain:
  - Row LSTM
  - Diagonal BiLSTM

# Pixel RNN



Row LSTM



Diagonal BiLSTM

# Thank you for listening

---