Generative Deep Learning

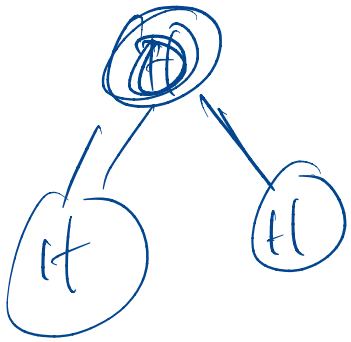
Naeemullah Khan
naeemullah.khan@kaust.edu.sa

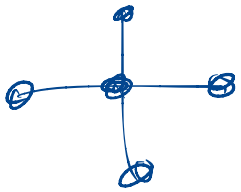
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

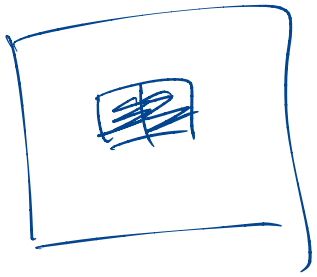
LMH
Lady Margaret Hall

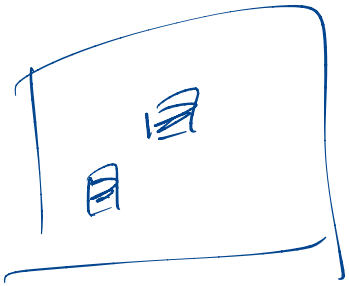
KAUST Academy
King Abdullah University of Science and Technology

July 11, 2023





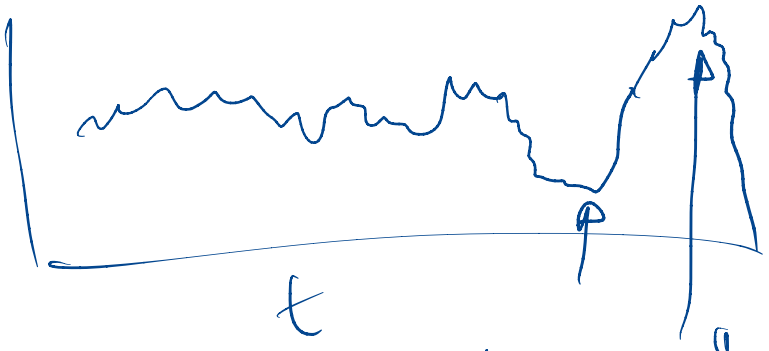




1. Deep Unsupervised Learning
   Generative Model
2. Reinforcement Learning
3. Graph Neural Network

1. Classical ML (Regression, logistic Regression)
2. NN
3. CNN
4. Backpropagation
5. Calculus / linear Algebra / optimization
6. PyTorch / Tensorflow

ML



$X \in \mathbb{R}^d$

Supervised Learning

X, y

$y = f_\theta(X)$

$X \in \mathbb{R}^n$

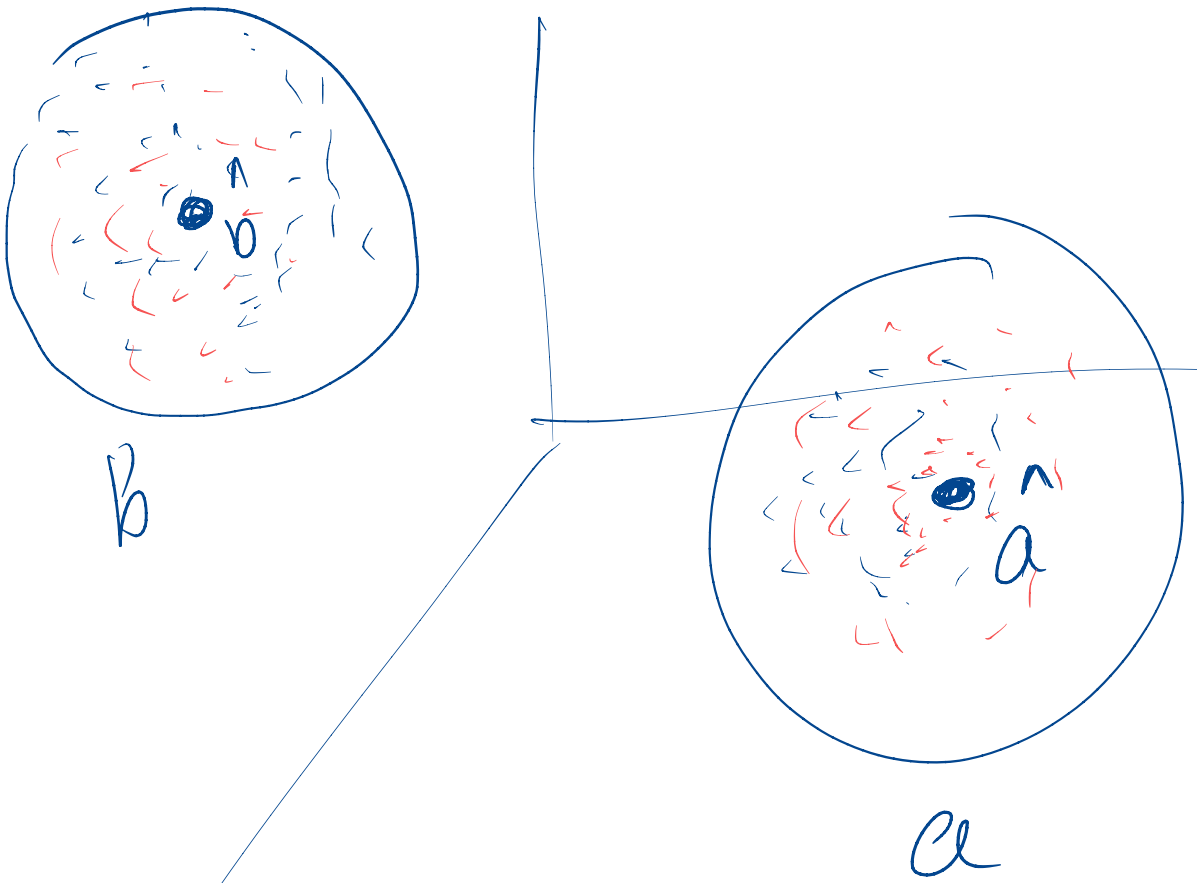
d is large

$X \rightarrow x$

$n \ll d$

Unsupervised Learning

1. Segmentation
2. Structure Data



$X_i \in \mathbb{R}^3$

$X_i \in a$

$\epsilon \sim N(0, \Sigma)$

$a_i = \hat{a} + \epsilon_i$

$b_j = \hat{b} + \epsilon_j$

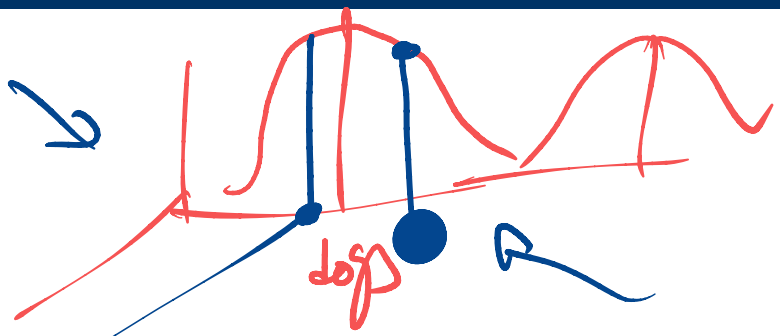
3×10^7 bytes

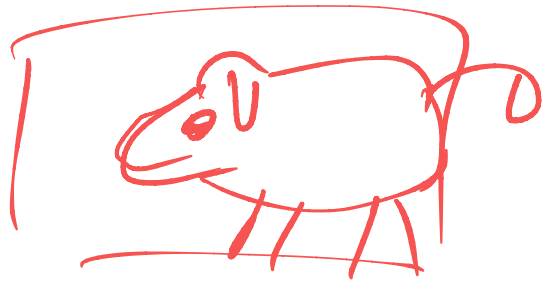
3×10 30 Kb

3×2 byte

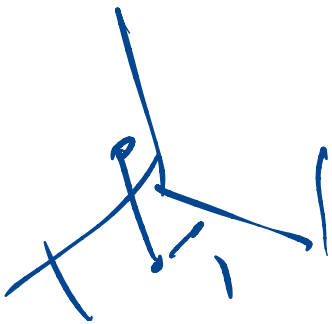
1. Comprehension
2. Structure → supervised tasks
3. Generation ✓✓

Introduction





- Can you imagine an image of dog?
- Now, can you imagine an image of that dog driving a car?
- How are we able to this?
- We excel at extracting knowledge from the data we observe and perform complex reasoning based on it



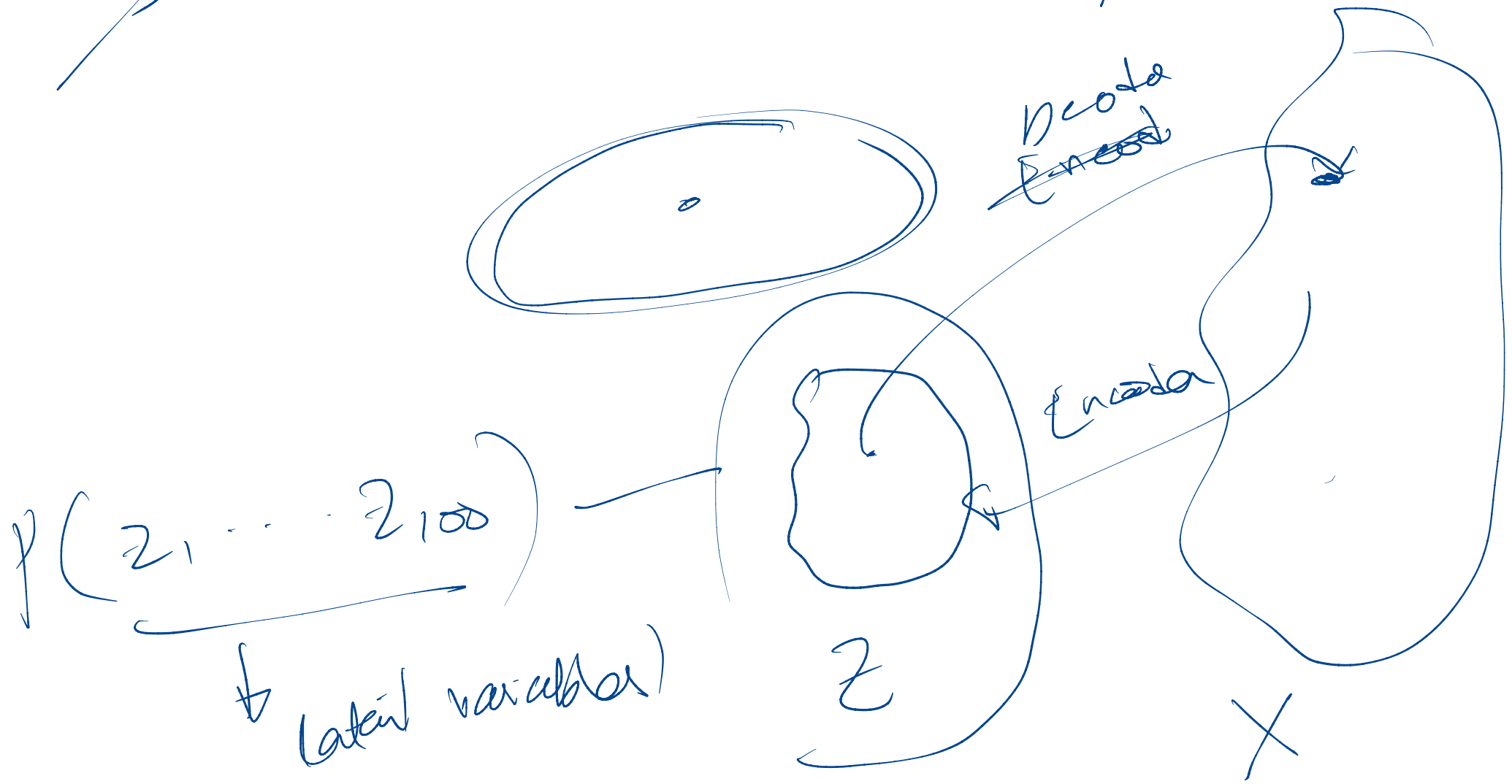
$(256)^{10^6}$ ← $x_1 \cdots x_{10^6}$ $\{0, \cdots 255\}$

$x_1, x_2 \in \{0, 255\}$

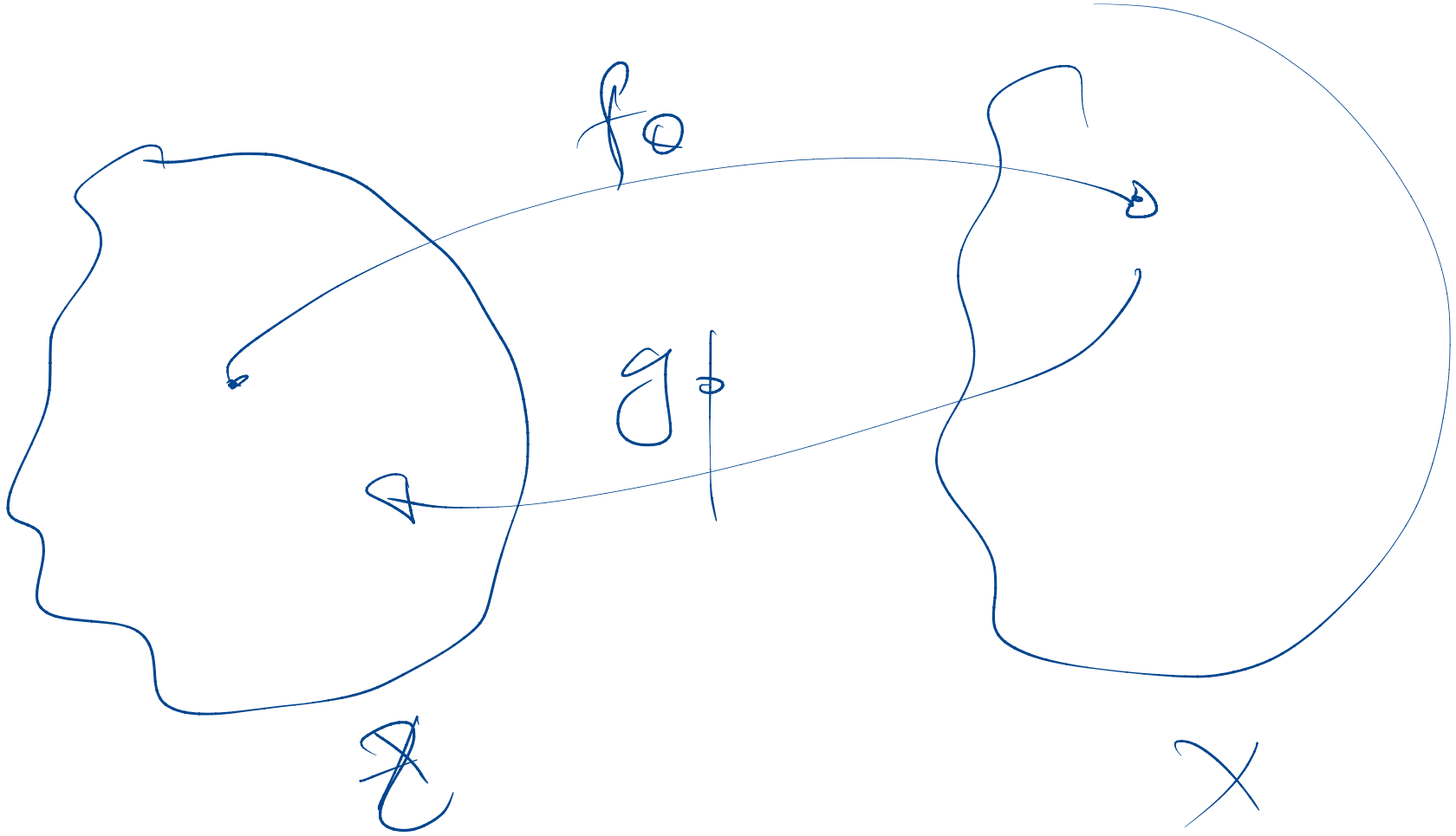
$x \in \{0, 255\}$

$$p(x_1 \dots x_{10^6}) = \prod_{i=1}^{10^6} p(x_i)$$

GAN's, VAE



$$p(z_1, \ldots, z_{10^6}) = \prod_{i=1}^{10^6} p(z_i)$$ Normalized flow



Auto Regressive Models

$$p(x_1, x_2, x_3 \cdots x_6) = p$$

$$p(x_1)\, p(x_2|x_1) \cdot p(x_3|x_2 x_1) \cdot p(x_4|x_3 x_2 x_1)$$

$$p(x_5|x_4, x_3 \cdots x_1) \cdot p(x_6|x_5 \cdots x_1)$$

$$p(x_1, x_2) \cdot p(x_3|x_2 x_1) = p(x_1, x_2 x_3)$$

$$p(x_1, x_2 x_3 x_4) \cdot p(x_5 x_2 x_3 x_4 x_6)$$

$$p(x_1, x_2) = p(x_2|x_1) \cdot p(x_1)$$

$$p(x_2|x_1) = \frac{p(x_1 x_2)}{p(x_1)}$$

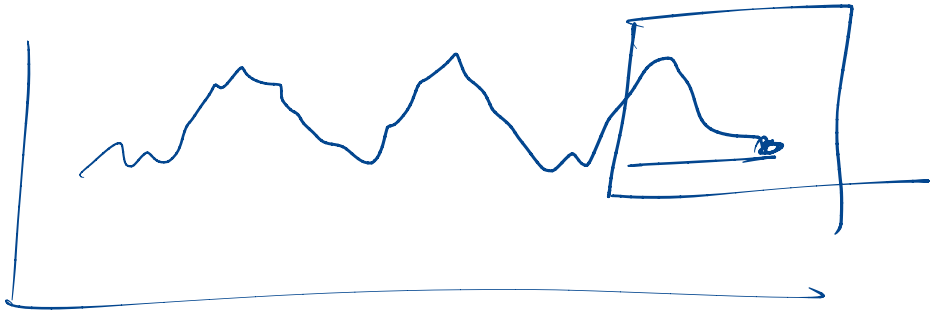
Bayes Rule

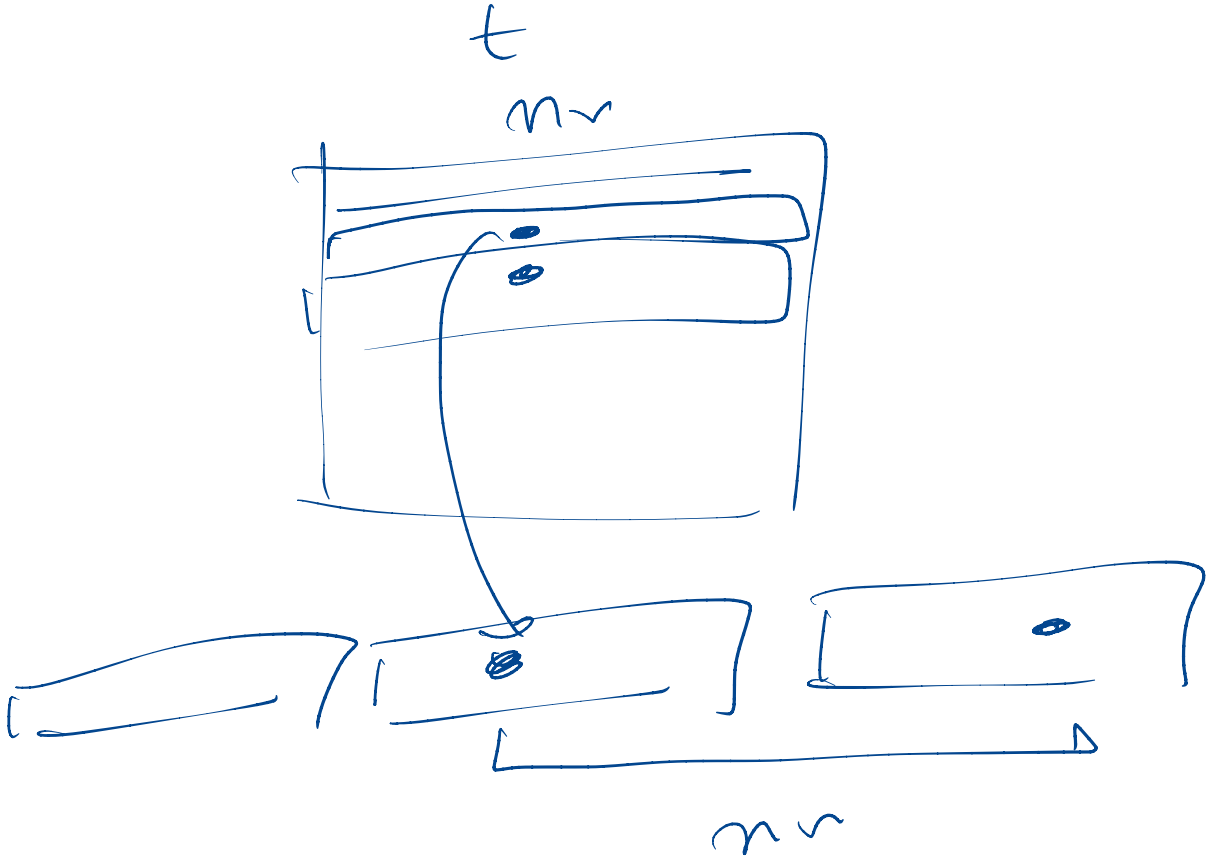
$$P(x_1 \dots x_M)$$

$$= \prod_{i=1}^{M} P(x_i \mid x_{i-1})$$

with some assumption

$$\prod_{i=1}^{M} P(x_i \mid h(x_i))$$





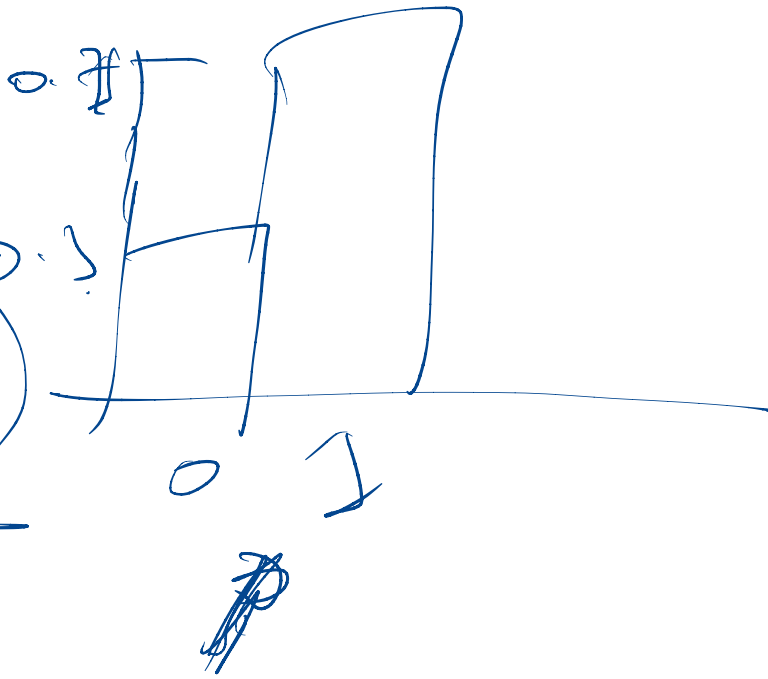
$[x_1\ x_2 \cdots x_M]$ ✓ $\quad X \longrightarrow y$

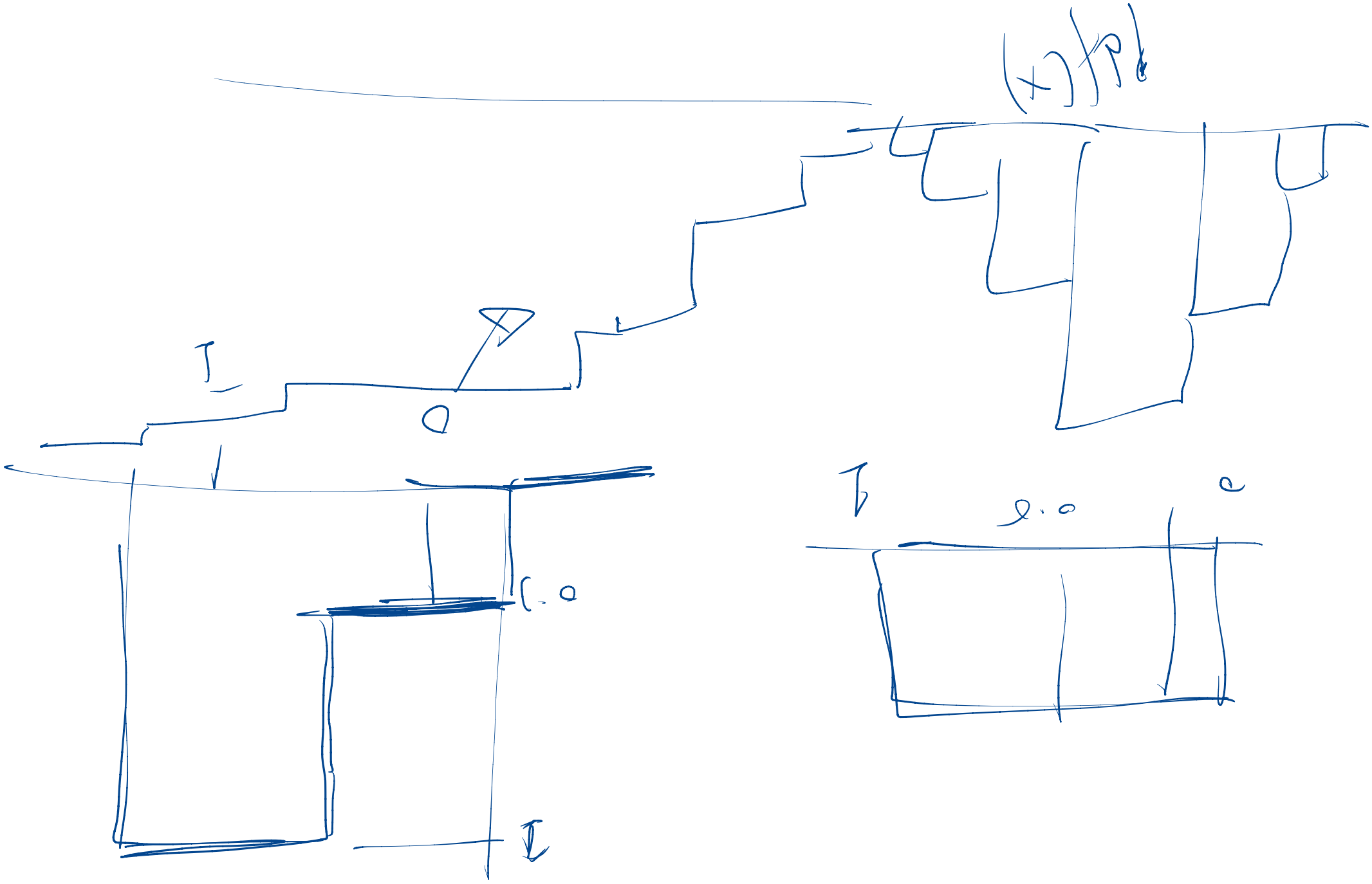
$f(x) = w_0 + w_1 x_1 + w_2 x_2$

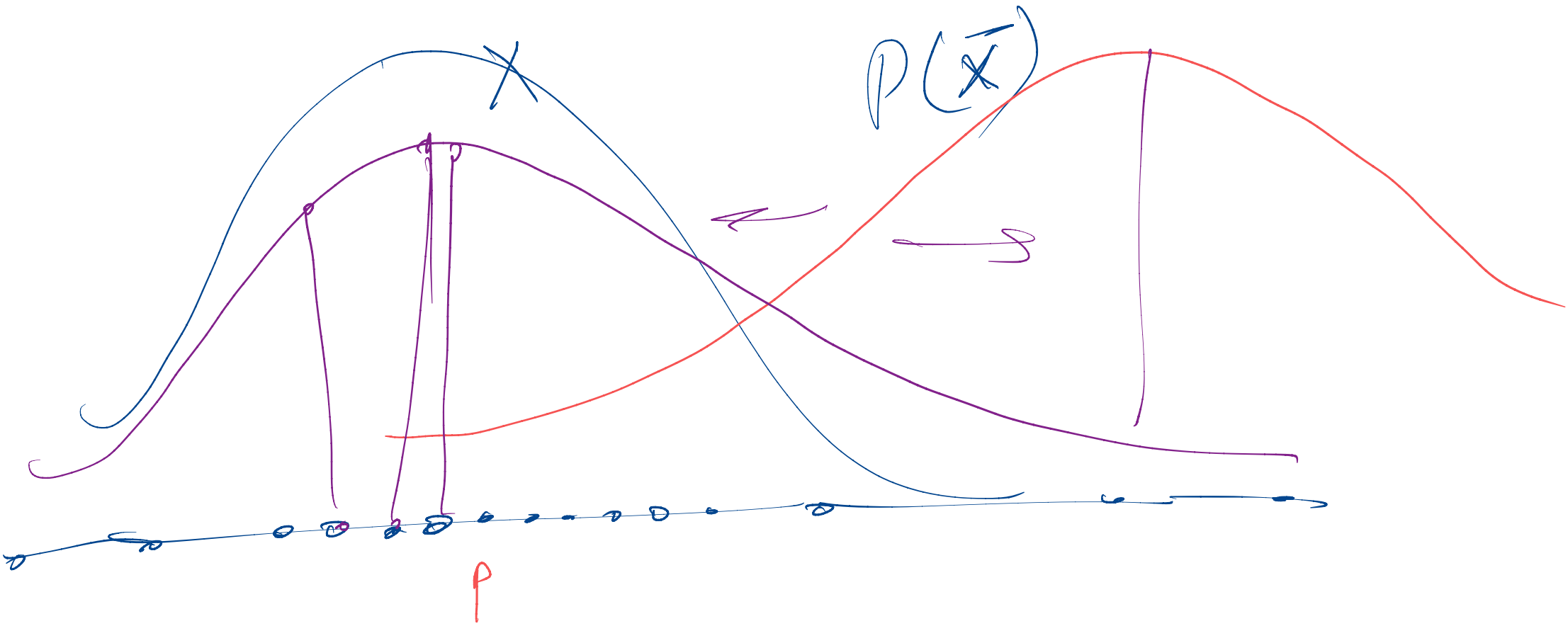
$p(x_1) \cdot p(x_2 | x_1) \cdot p(x_3 | x_2 x_1) \cdot p(x_2 | x_1, x_2 x_1)$

$p(x_2|\cdot) = g(\alpha_0^1 + \alpha_1^1 x_1^1)$ $= f$

$p(x_3 | x_1 x_1) = g(\alpha_0^2 + \alpha_1^2 x_1 + \alpha_2^2 x_2)$






X
p(x)
p

Introduction (cont.)

How can we generate such images/data through AI

What we would need? Let's think about it step by step.

- Some prior data
- A model which will "learn" the data
- A method through which the model will learn
- A method to generate new data from the trained model

This what generative deep learning is all about and what we will be covering in this course.

$$p = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$x_{(1)}, x_{(2)}, \cdots x_{(N)}$

$$p(x_{(1)}, x_{(2)} \cdots x_{(N)}) = \prod_{i=1}^{N} p(x_{(i)})$$ iid

$$\max_{\mu} \prod_{i=1}^{N} p(x_i)$$ → likelihood

$$\max_{\mu} \sum_{i=1}^{N} \log(p_{x_i})$$

$$\max_{\mu} N\left(\log \frac{1}{\sqrt{2\pi}}\right) + \sum_{i=1}^{N} -\frac{(x_i-\mu)^2}{2}$$

LL $\rightarrow \arg\max_{\mu}$

NLL $\rightarrow \arg\min_{\mu} \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2} - \log\left(\frac{1}{\sqrt{2\pi}} \exp^{-\frac{(x-\mu)^2}{2}}\right)$

$$\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{(x-\mu)}{2\sigma^2}}$$

$$\frac{d}{d\mu} \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2}$$

$$\sum_{i=1}^{N} \frac{d}{d\mu} \frac{(x_i-\mu)^2}{2} \qquad 2 \cdot \frac{\sum_{i=1}^{N}(x_i-\mu)}{2} = 0$$

$$\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} \mu = \mu \sum_{i=1}^{N} 1 = N\mu$$

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Introduction (cont.)

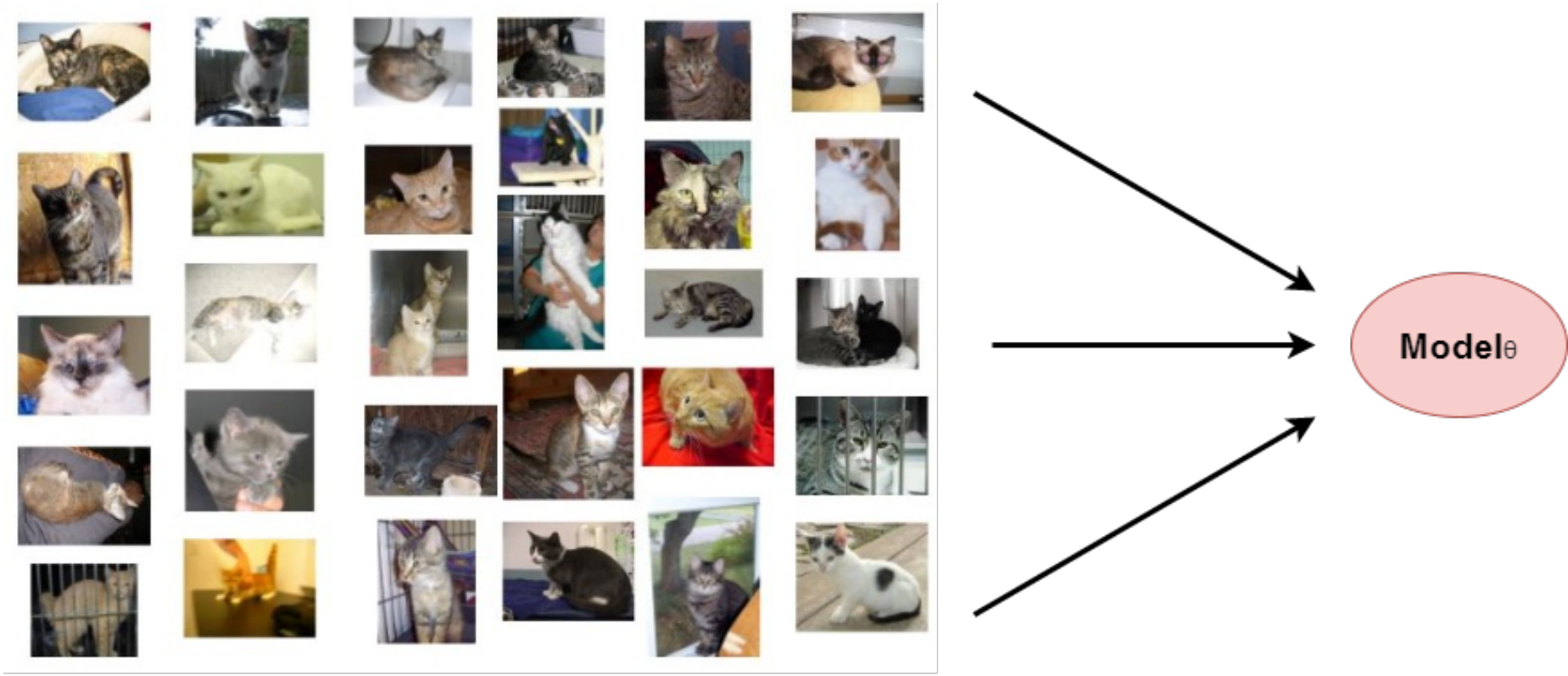


Figure 2: The model learns the cat images data

Introduction (cont.)

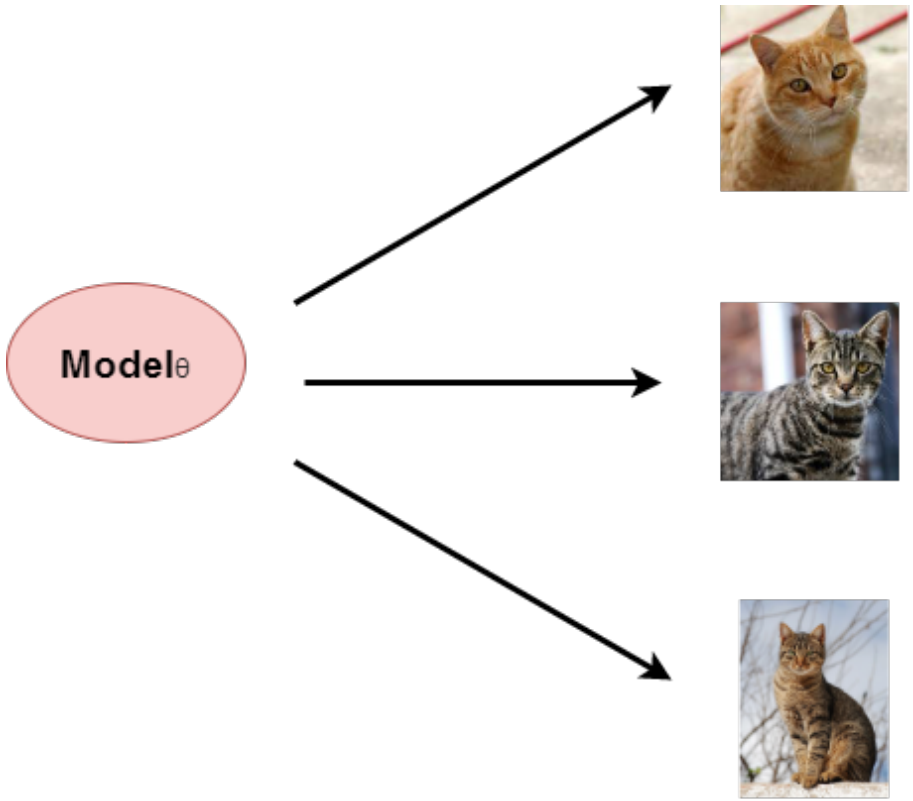


Figure 3: The trained model generates new cat images

Statistical Generative Models

Statistical generative models are generative models that are learned from data.

- Data $\mathcal{X} \sim \mathcal{P}(\mathcal{X})$
- Model $\theta \sim \mathcal{P}(\theta)$
- Optimize model s.t. $\mathcal{P}(\theta) \sim \mathcal{P}(\mathcal{X})$
- Generate $\mathcal{X}_{new} \sim \mathcal{P}(\theta)$

Discriminative vs. Generative

Discriminative Models

- E.g.: Classify between cat and dog images
- Doesn't really need to learn data distribution $\mathcal{P}(\mathcal{X})$
- Goal is to learn $\mathcal{P}(\mathcal{Y} = \text{dog}|\mathcal{X} = x)$

Generative Models

- E.g.: Generate new cat images
- Need to learn data distribution $\mathcal{P}(\mathcal{X})$
- Goal is to maximize $\mathcal{P}(\mathcal{X} = x_{new})$

Progress on Face Generation

Figure 4: Progress in face generation over the years[1]

[1] https://twitter.com/tamaybes/status/1450873331054383104

Completing Incomplete Content

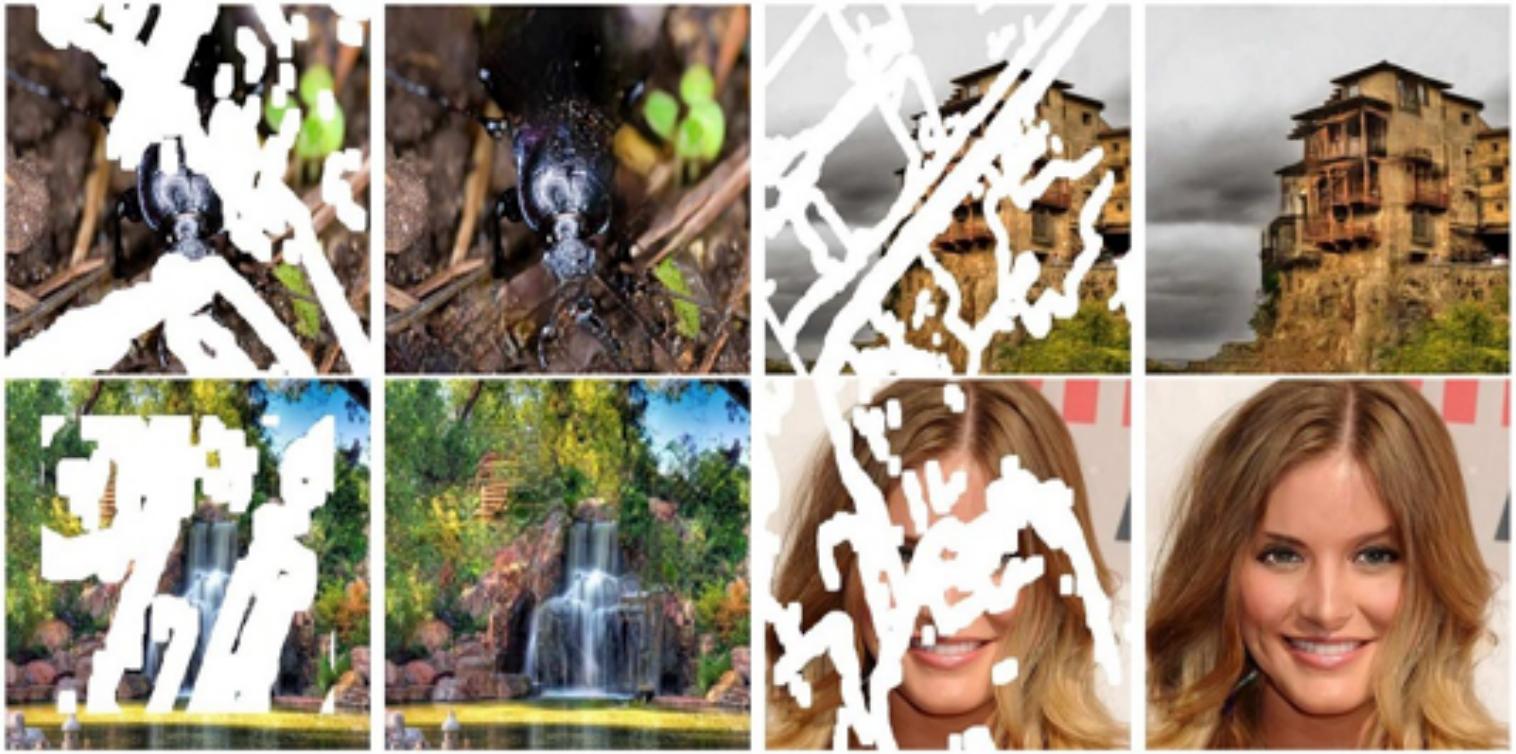


Figure 5: Complete missing patches in image

Completing Incomplete Content (cont.)

Figure 6: Improve image quality

Completing Incomplete Content (cont.)

Figure 7: Color b/w images

Completing Incomplete Content (cont.)

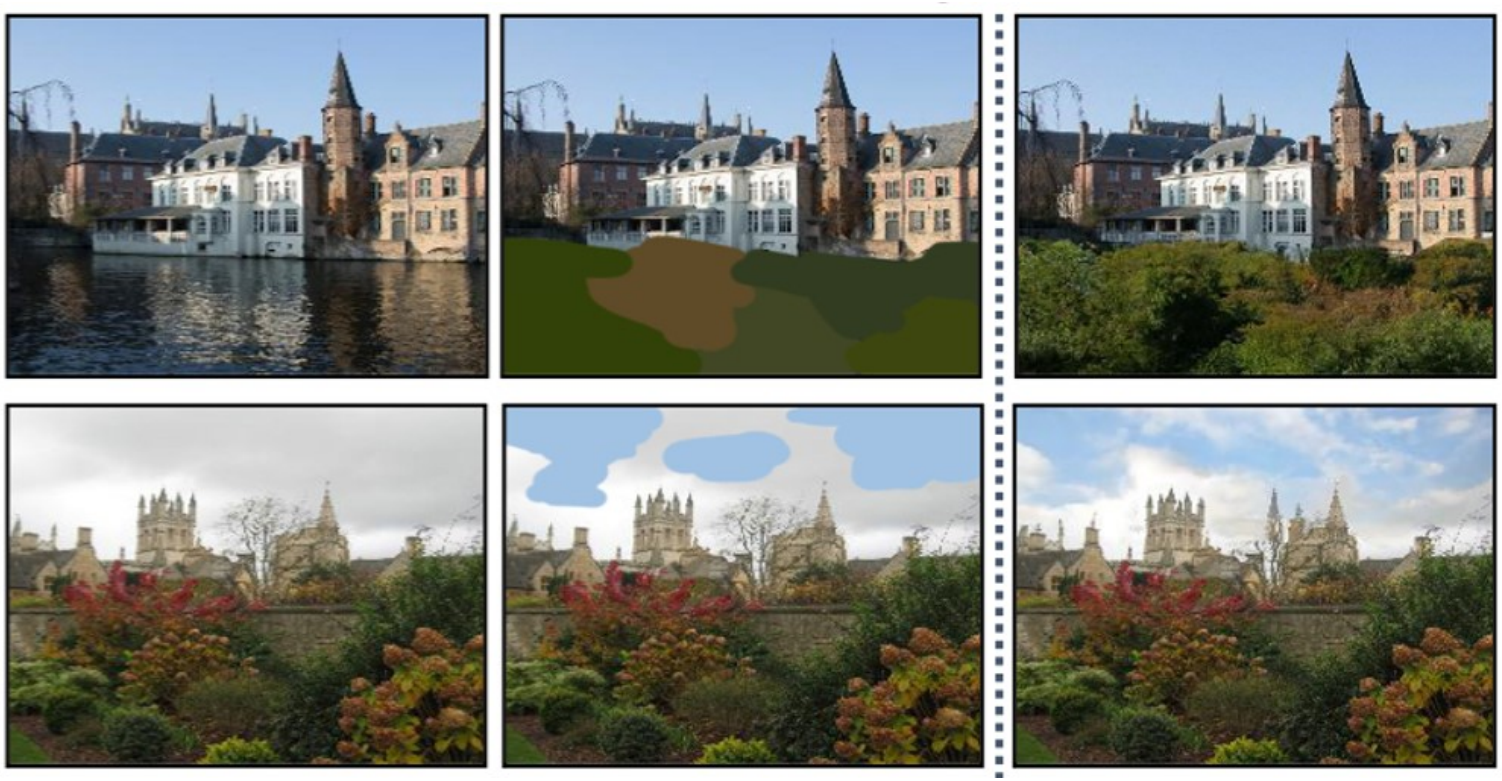


Figure 8: Editing based on strokes in image

Completing Incomplete Content (cont.)

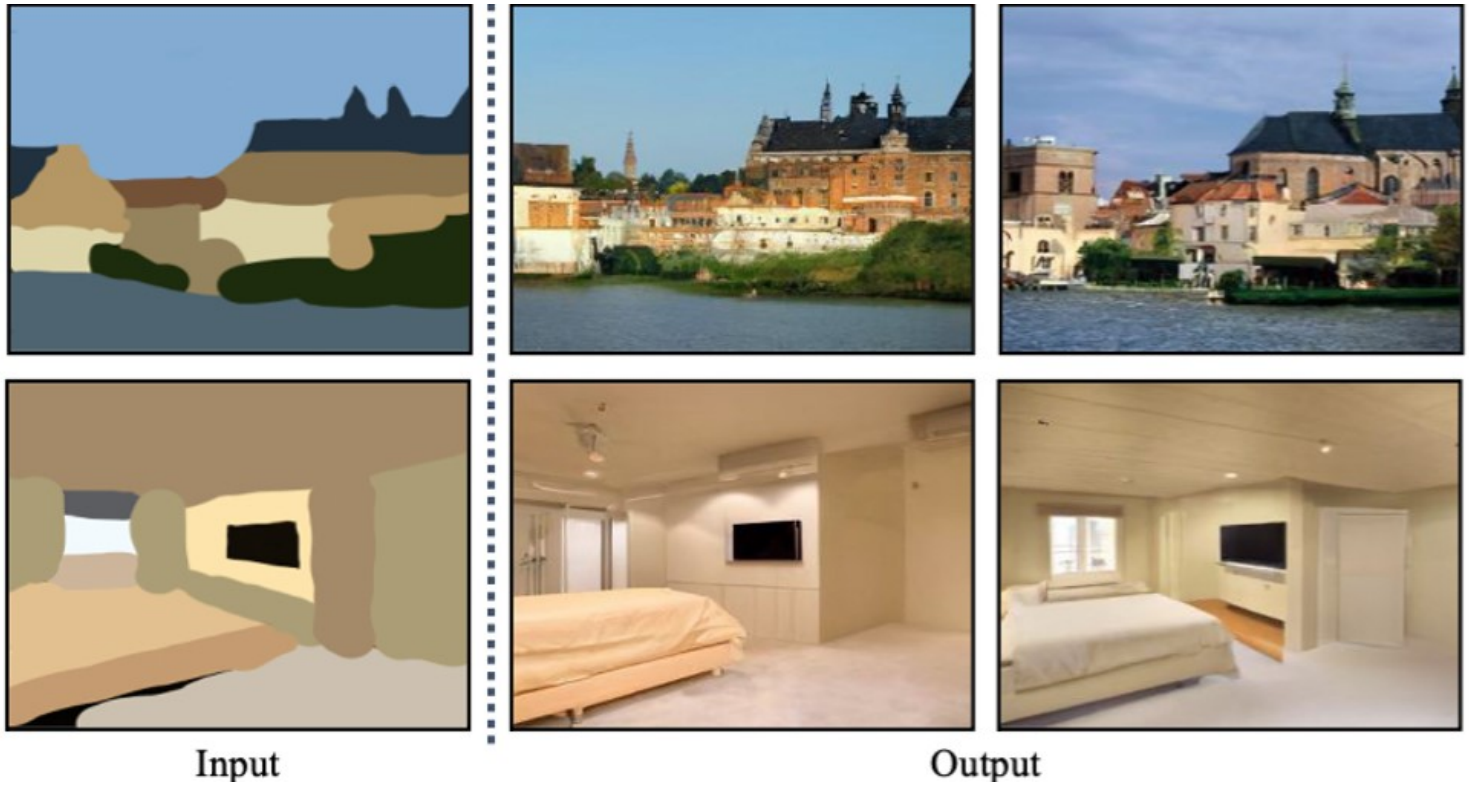


Figure 9: Converting strokes to proper images

Text Generation

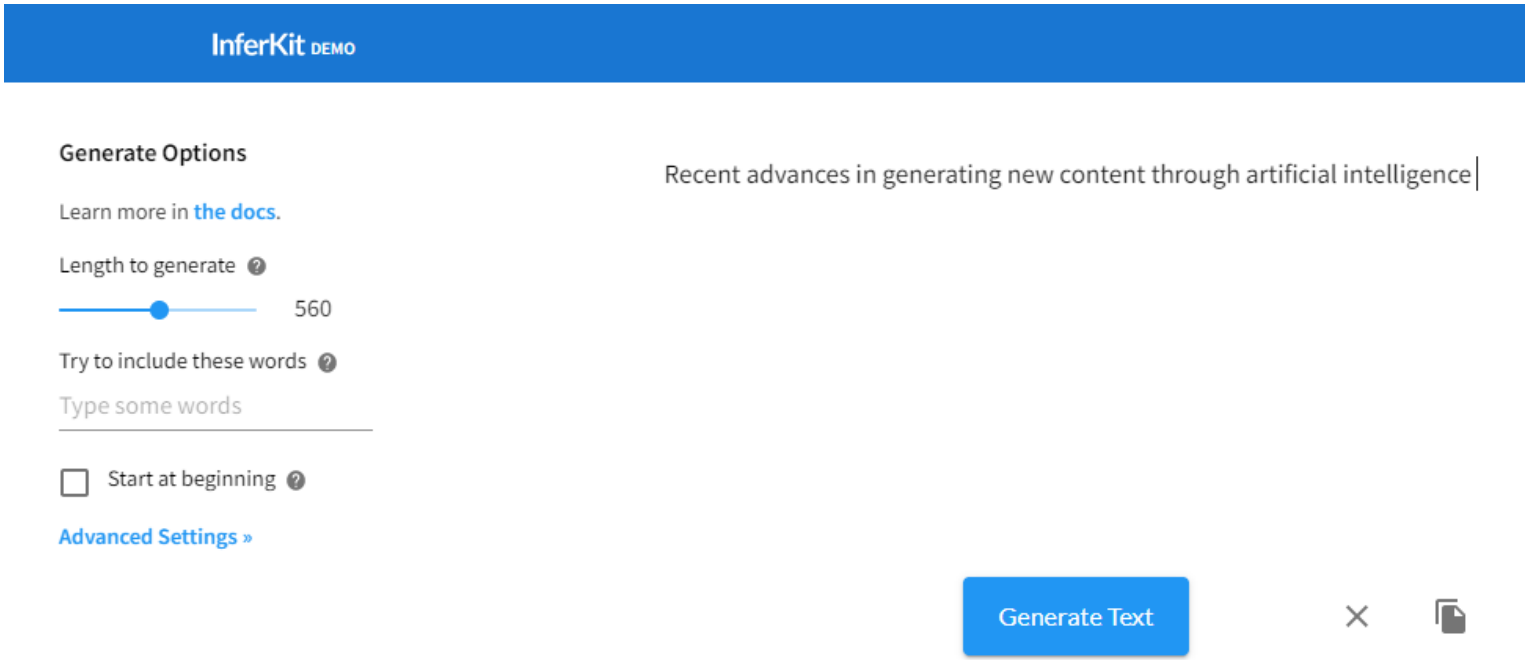


Figure 10: Completing Text. https://app.inferkit.com/demo

What is Deep Unsupervised Learning?

- Capturing rich patterns in raw data with deep networks in a **label-free** way
  - Generative Models: recreate raw data distribution
  - Self-supervised Learning: “puzzle” tasks that require semantic understanding
- But why do we care?

Geoffrey Hinton

(in his 2014 AMA on Reddit)

“The brain has about 10^14 synapses and we only live for about 10^9 seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input (including proprioception) is the only place we can get 10^5 dimensions of constraint per second.”

Yann LeCun

Need tremendous amount of information to build machines that have common sense and generalize

[LeCun-20161205-NeurIPS-keynote]

- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning (cake)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

LeCake

- **(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)**

“Ideal Intelligence”

“Ideal Intelligence” is all about compression (finding all patterns)

- Finding all patterns = short description of raw data (low Kolmogorov Complexity)
- Shortest code-length = optimal inference (Solomonoff Induction)
- Extensible to optimal action making agents (AIXI)

Aside from theoretical interests

- Deep Unsupervised Learning has many powerful applications
  - Generate novel data
  - Conditional Synthesis Technology (WaveNet, GAN-pix2pix)
  - Compression
  - Improve any downstream task with un(self)supervised pre-training
    - Production level impact: Google Search powered by BERT
  - Flexible building blocks

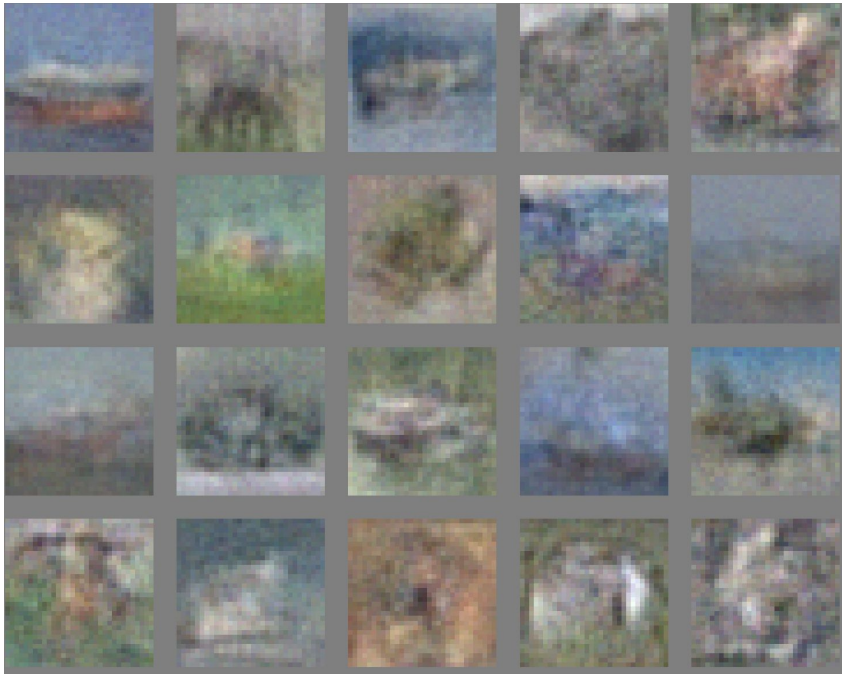
Generate Images

[Deep Belief Nets, Hinton, Osindero, Teh, 2006]

Generate Images

[VAE, Kingma and Welling, 2013]

Generate Images



[GAN, Goodfellow et al. 2014]

Generate Images

[DCGAN, Radford, Metz, Chintala 2015]

Generate Images

[DCGAN, Radford, Metz, Chintala 2015]

Generate Images

bicubic
(21.59dB/0.6423)

SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

[Ledig, Theis, Huszar et al, 2017]

Generate Images

[CycleGAN: Zhu, Park, Isola & Efros, 2017]

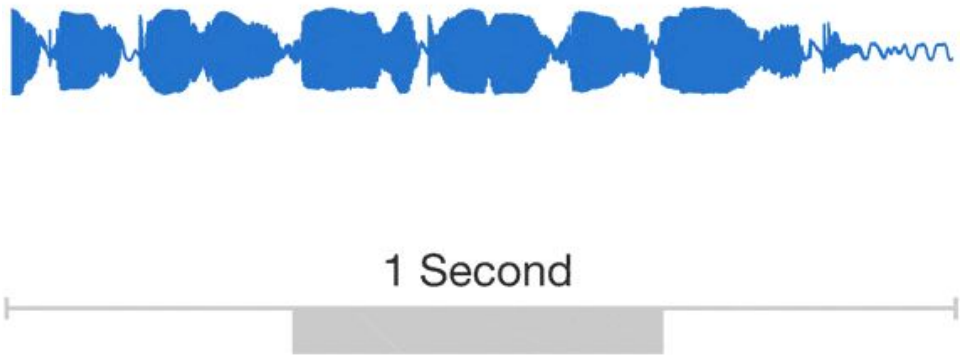
Generate Images

[BigGAN, Brock, Donahue, Simonyan, 2018]

Generate Images

[StyleGAN, Karras, Laine, Aila, 2018]

Generate Audio

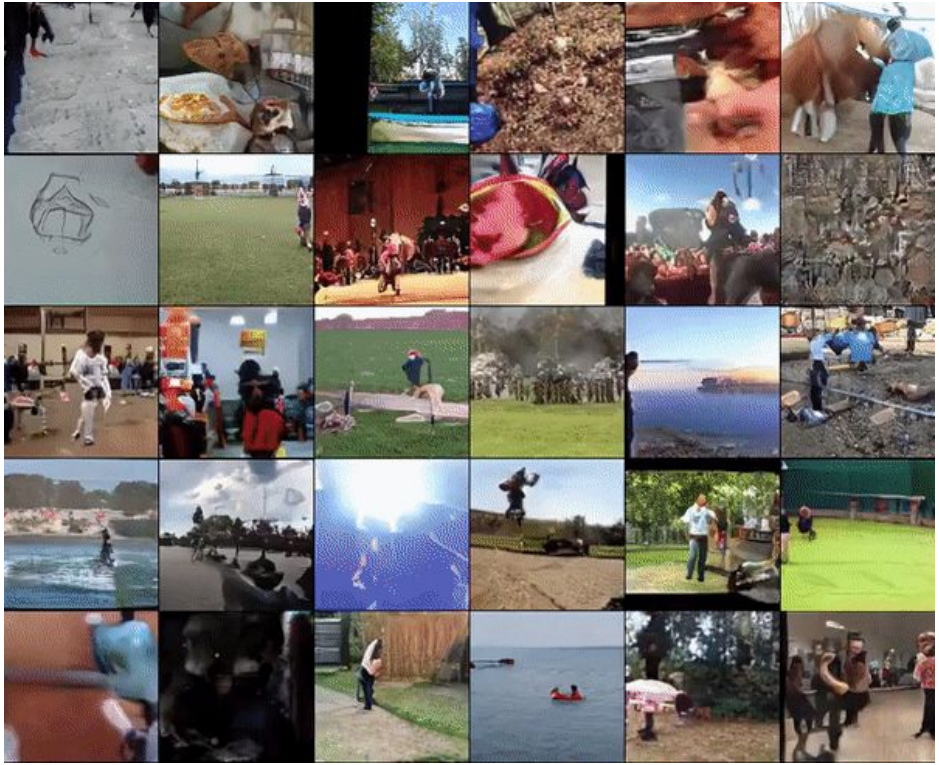


Parametric

WaveNet

[WaveNet, Oord et al., 2018]

Generate Video



DVD-GAN: Adversarial Video Generation on Complex Datasets, Clark, Donahue, Simonyan, 2019

Generate Text

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

[Char-rnn, karpathy, 2015]

Generate Math

```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on
$\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \to
\mathcal{I}$
is an injective and let $\mathfrak q$ be an abelian sheaf on
$X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$
be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on
$\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module.
Then
$\mathcal{F}$ is an abelian catenary over $\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{lemma}
```

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariconly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M^\bullet} = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.

[Char-rnn, karpathy, 2015]

Text Generation (cont.)

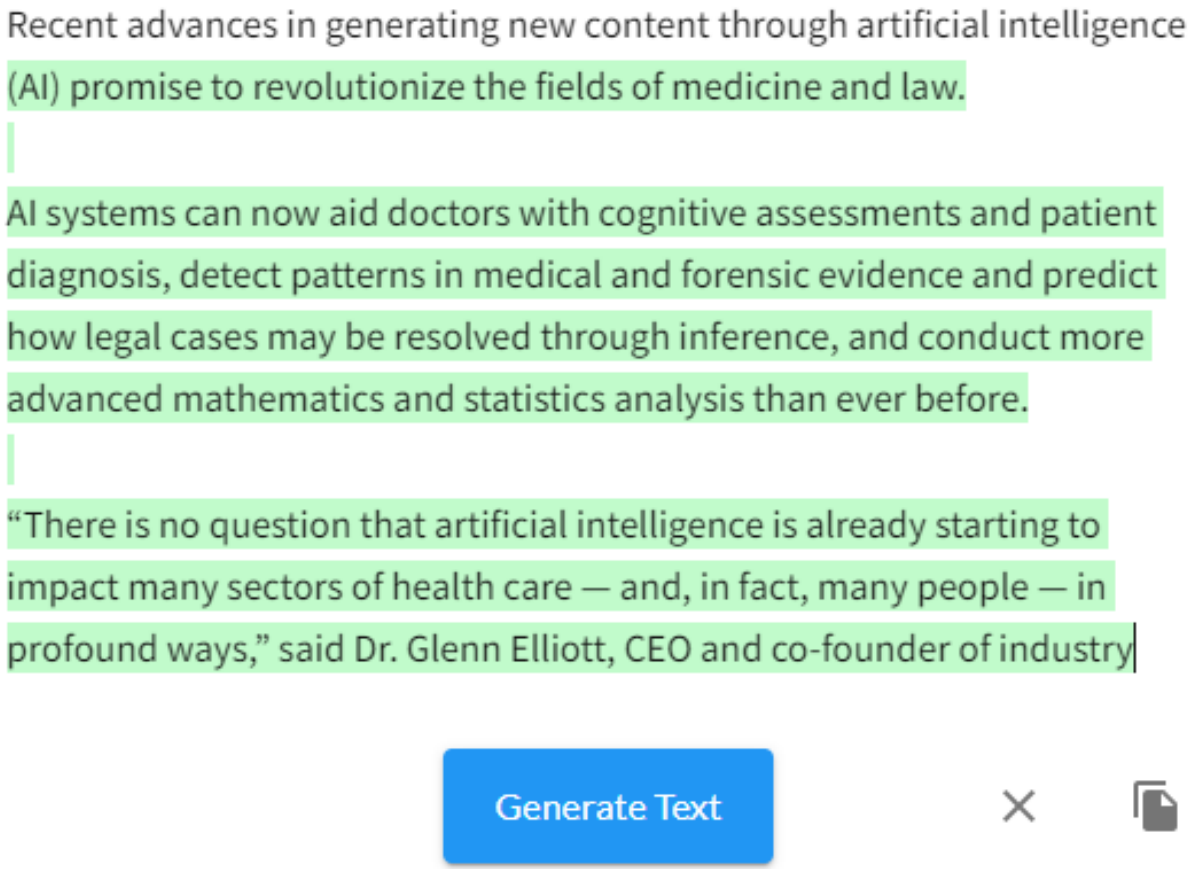


Figure 11: Completing Text. https://app.inferkit.com/demo

Machine Translation

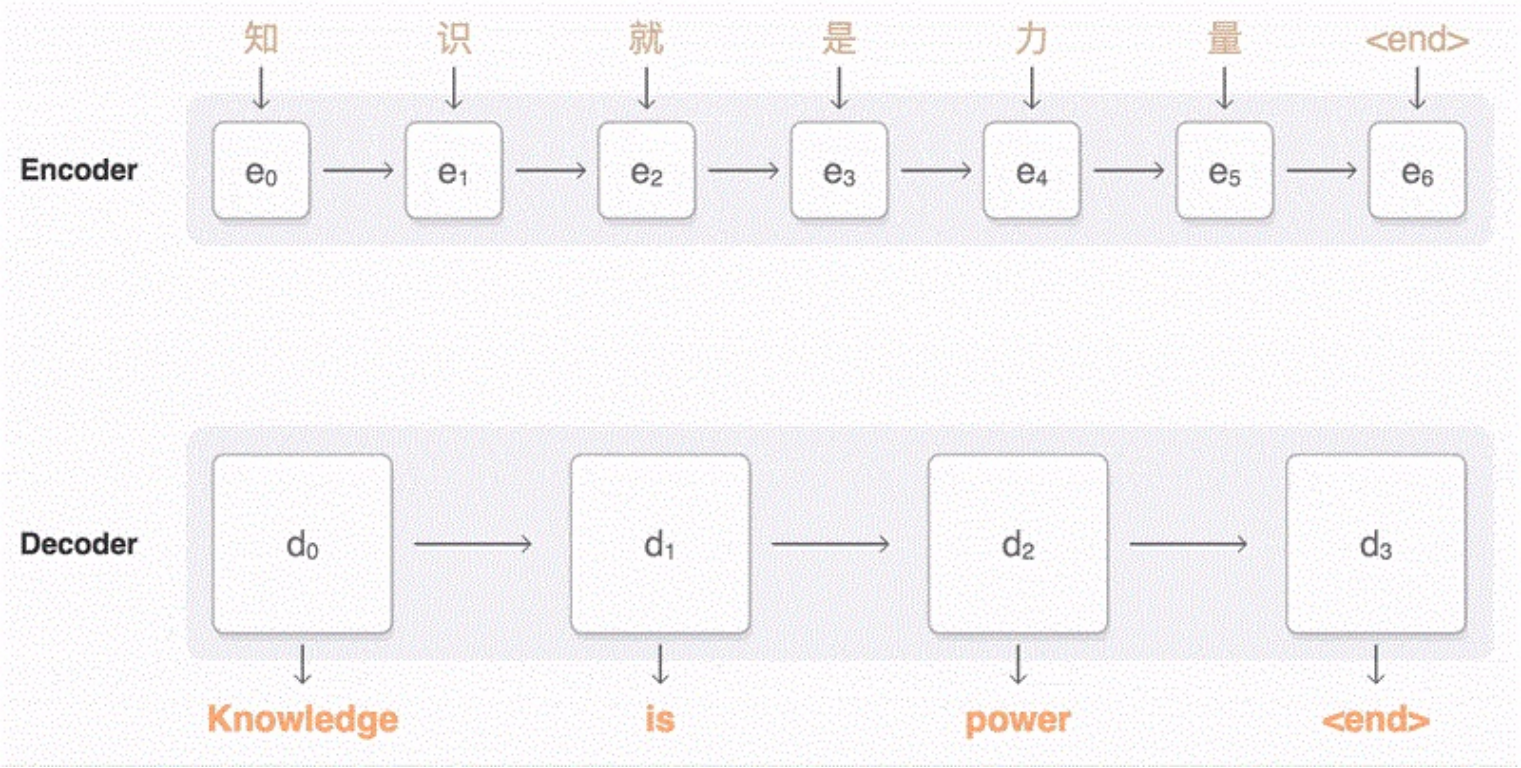


Figure 12: Language translation

Code Generation

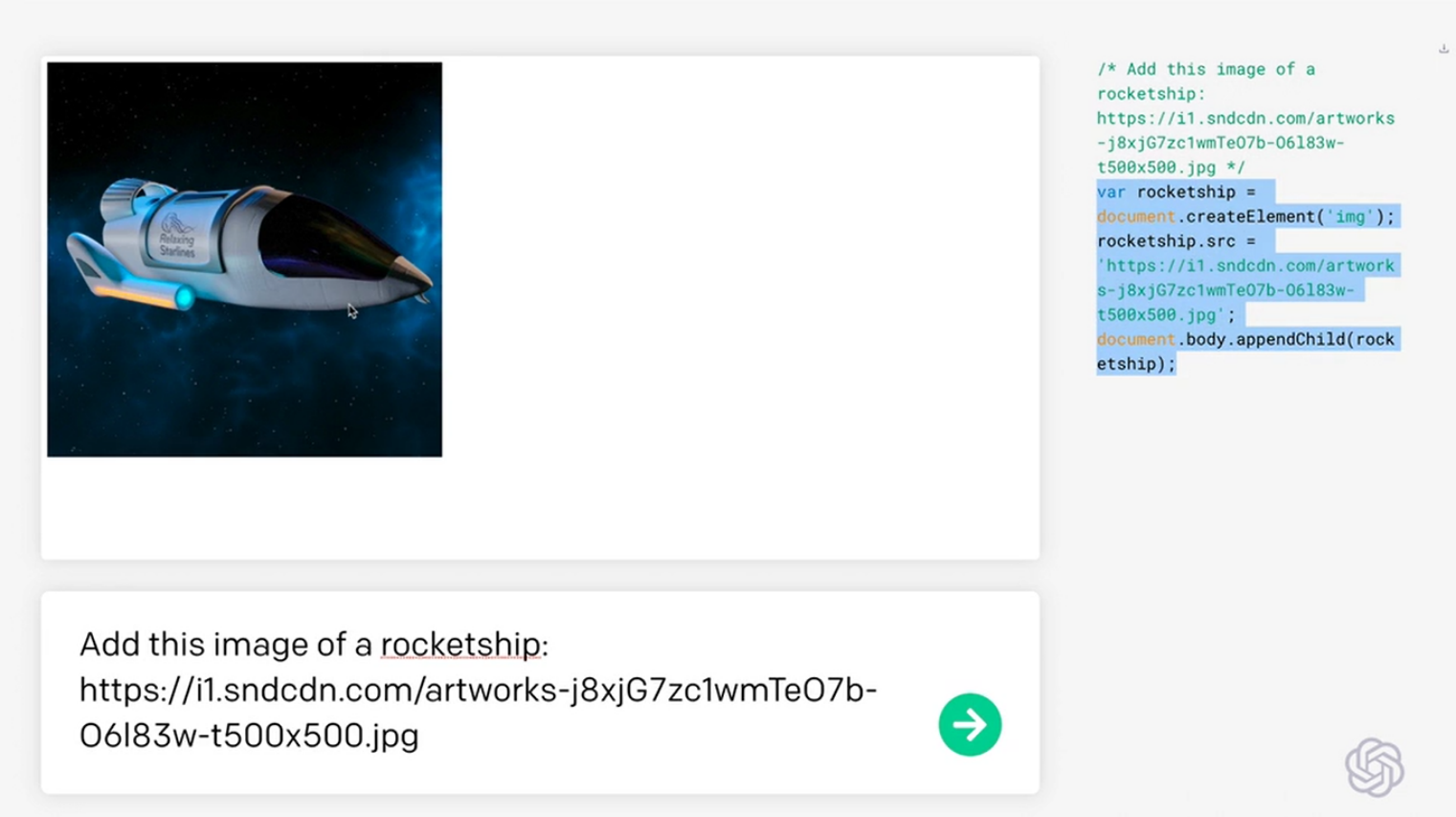


Figure 13: Code Generation. https://openai.com/blog/openai-codex/

Learning a Generative Model

For a given dataset of dog images, we want to learn a probability distribution $\mathcal{P}_{theta}(x)$ over images x such that

- **Generation:** If we sample $x_{new} \sim \mathcal{P}_{theta}(x)$, it should look like a dog image (sampling)
- **Density Estimation:** $\mathcal{P}_{theta}(x)$ should be high if x is cat image and low otherwise (anomaly detection)
- **Unupservised Representation Learning:** We should be able to learn what these images have in common e.g., ears, tail, etc. (features)

Let's Dive Deeper

- How do we represent the distribution of data $\mathcal{P}_{data}(x)$?
- Let's consider a simple case binarized MNIST digits dataset.
- We have an image x of 28 x 28. Flattening it gives us a vector of length 784. Each pixel can be either 0 or 1. $x \in \{0, 1\}^{784}$
- Now, we need to learn the joint distribution $\mathcal{P}(x) = \mathcal{P}(x_1, x_2, \cdots, x_{784})$

Outline

- **Motivation**
- Simple generative models: histograms
- Modern neural autoregressive models
  - Parameterized distributions and maximum likelihood
  - Autoregressive Models
    - Recurrent Neural Nets
    - Masking-based Models

Likelihood-based models

Problems we'd like to solve:

- Generating data: synthesizing images, videos, speech, text
- Compressing data: constructing efficient codes
- Anomaly detection

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

Learns a distribution p that allows:

- Computing p(x) for arbitrary x
- Sampling x ~ p(x)

Today: **discrete** data

Desiderata

We want to estimate distributions of **complex, high-dimensional data**

- A 128x128x3 image lies in a ~50,000-dimensional space

We also want computational and statistical efficiency

- Efficient training and model representation
- Expressiveness and generalization
- Sampling quality and speed
- Compression rate and speed

Outline

- Motivation
- **Simple generative models: histograms**
- Modern neural autoregressive models
  - Parameterized distributions and maximum likelihood
  - Autoregressive Models
    - Recurrent Neural Nets
    - Masking-based Models

Learning: Estimate frequencies by counting

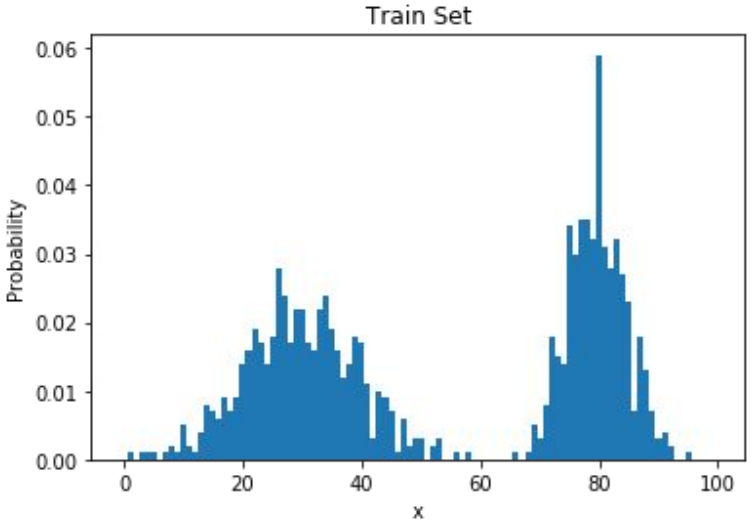
Recall: the goal is to estimate $\mathbf{p_{data}}$ **from samples** $\mathbf{x^{(1)}, \ldots, x^{(n)} \sim p_{data}(x)}$

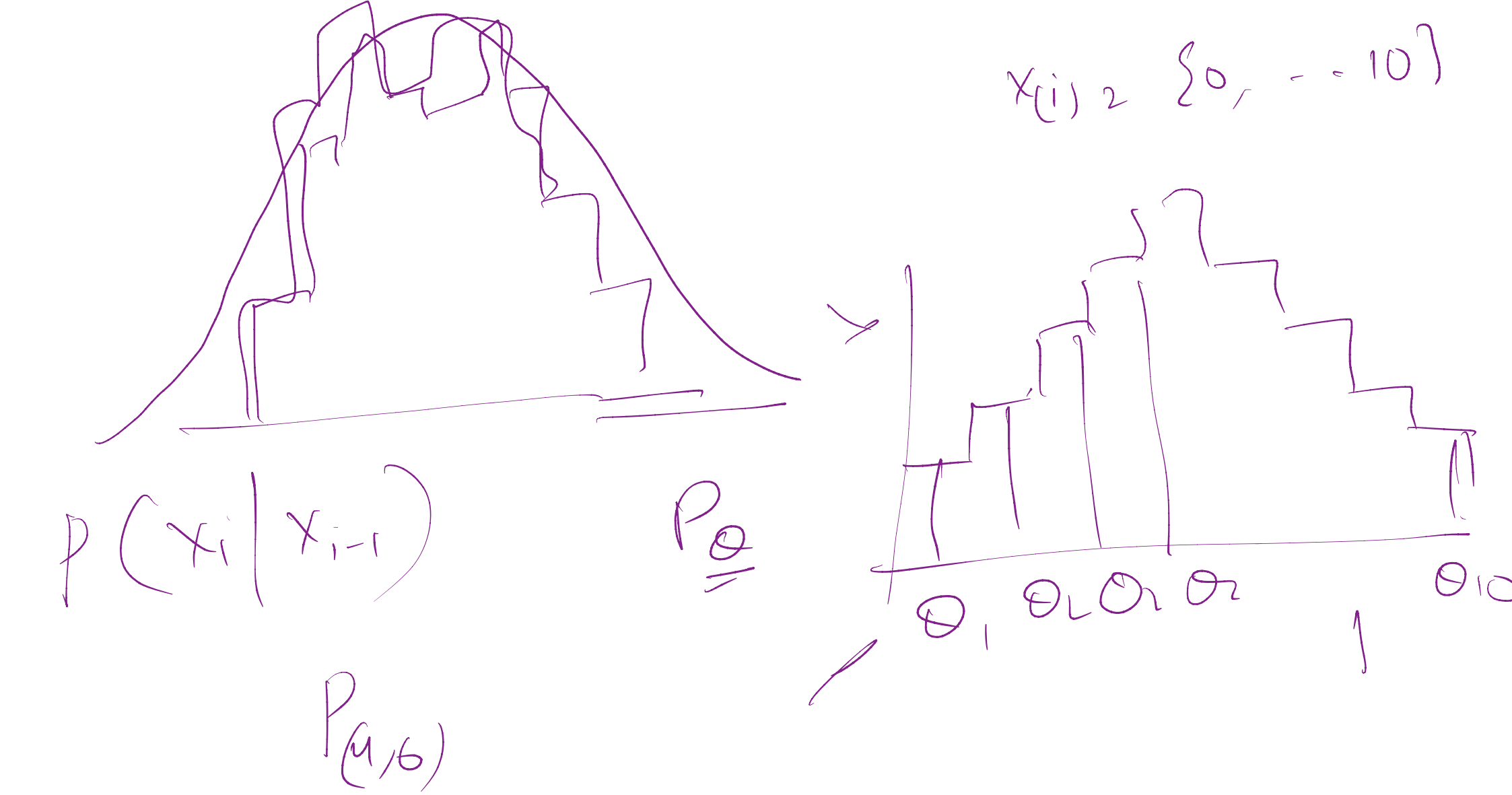
Suppose the samples take on values in a finite set {1, …, k}

The model: a **histogram**

- (Redundantly) described by k nonnegative numbers: $p_1, \ldots, p_k$
- To train this model: count frequencies

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




$x_{(i)} \in \{0, \ldots 10\}$
$p(x_i | x_{i-1})$
P_θ
$P_{(\mu,\sigma)}$
θ_1
θ_{10}
1

Status

- **Issues with histograms**
  - High dimensions: won't work
  - Even 1-d: if many values in the domain, prone to overfitting

- **Solution: function approximation**. Instead of storing each probability, store a parameterized function $p_\theta(x)$

Likelihood-based generative models

Recall: the goal is to **estimate p_{data} from $x^{(1)}, \ldots, x^{(n)} \sim p_{data}(x)$**

Now we introduce **function approximation**: learn θ so that $p_\theta(x) \approx p_{data}(x)$.

- How do we design function approximators to effectively represent complex joint distributions over x, yet remain easy to train?
- There will be many choices for model design, each with different tradeoffs and different compatibility criteria.

Designing the model and the training procedure go hand-in-hand.

Fitting distributions

- Given data $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ sampled from a "true" distribution p_{data}
- Set up a model class: a set of parameterized distributions p_θ
- Pose a search problem over parameters

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

- Want the loss function + search procedure to:
  - Work with large datasets (n is large, say millions of training examples)
  - Yield θ such that p_θ matches p_{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 $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$, find θ by solving the optimization problem

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

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

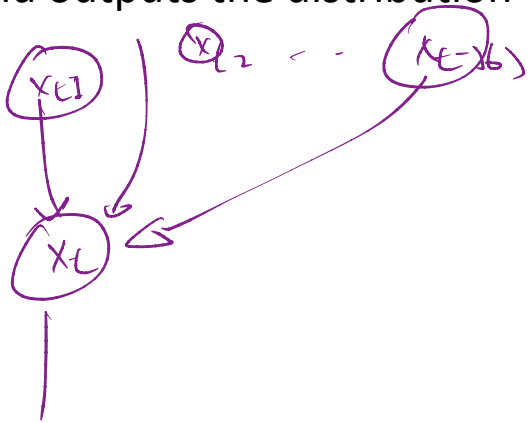
$$\hat{p}_{\text{data}}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[\mathbf{x} = \mathbf{x}^{(i)}]$$

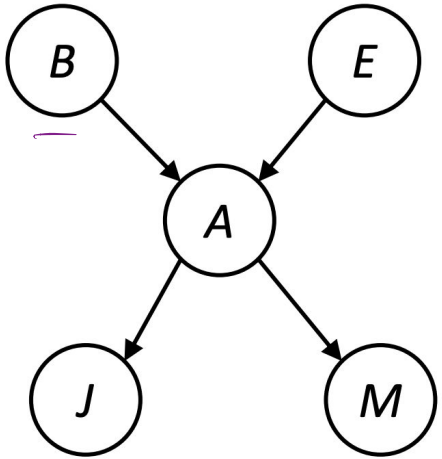
$$\text{KL}(\hat{p}_{\text{data}} \,\|\, p_\theta) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}}[-\log p_\theta(\mathbf{x})] - H(\hat{p}_{\text{data}})$$

Bayes nets and neural nets

Main idea: place a **Bayes net** structure (a directed acyclic graph) over the variables in the data, and model the conditional distributions with neural networks.

Reduces the problem to designing co
variables. We know how to do this: th
as input, and outputs the distribution





B	P(B)
+b	0.001
-b	0.999

E	P(E)
+e	0.002
-e	0.998

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

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

$$\prod_{i=1}^{N} p(x_i \mid x_{i-1})$$

$$\prod_{i=1}^{N} p(x_i \mid x_{i-n} \cdots x_{i-1})$$

Autoregressive models are models where observations from previous time-steps are used to predict the value at current time-step.

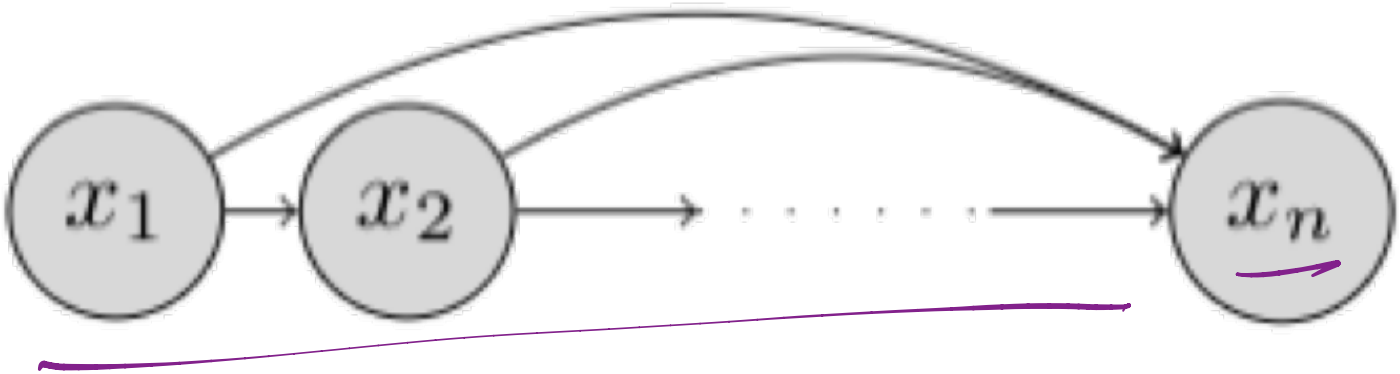


Figure 14: Graphical model for an autoregressive network

Autoregressive Models (cont.)

- Continuing the MNIST dataset example, we can set an ordering for all the random variables from top-left X_1 to bottom-right X_{784}
- Without loss of generality, we can use chain rule for factorization

$$\mathcal{P}(x_1, \cdots, x_{784}) = \mathcal{P}(x_1)\mathcal{P}(x_2|x_1)\mathcal{P}(x_3|x_1, x_2)\cdots\mathcal{P}(x_{784}|x_1, x_2, \cdots, x_{784})$$

- Parameterizing the above and using the sigmoid function for binarization,
  - $\mathcal{P}_{CPT}(X_1 = 1; \alpha^1) = \alpha^1, \mathcal{P}(X_1 = 0) = 1 - \alpha^1$
  - $\mathcal{P}_{logit}(X_2 = 1|x_1; \alpha^2) = \sigma(\alpha_0^2 + \alpha_1^2 x_1)$
  - $\mathcal{P}_{logit}(X_3 = 1|x_1, x_2; \alpha^3) = \sigma(\alpha_0^3 + \alpha_1^3 x_1 + \alpha_2^3 x_2)$

The above is the most simplest case of autoregressive model in which we specify the function as a linear combination of the input elements followed by a sigmoid non-linearity. This is known as Fully-Visible Sigmoid Belief Network (FVSBN).

$$f_i(x_1, x_2, \cdots, x_{i-1}) = \sigma(\alpha_0^{(i)} + \alpha_1^{(i)} x_1 + \cdots + \alpha_{i-1}^{(i)} x_{i-1})$$

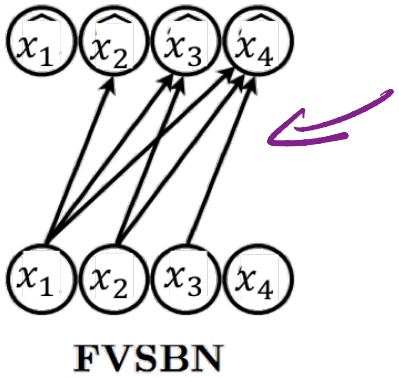
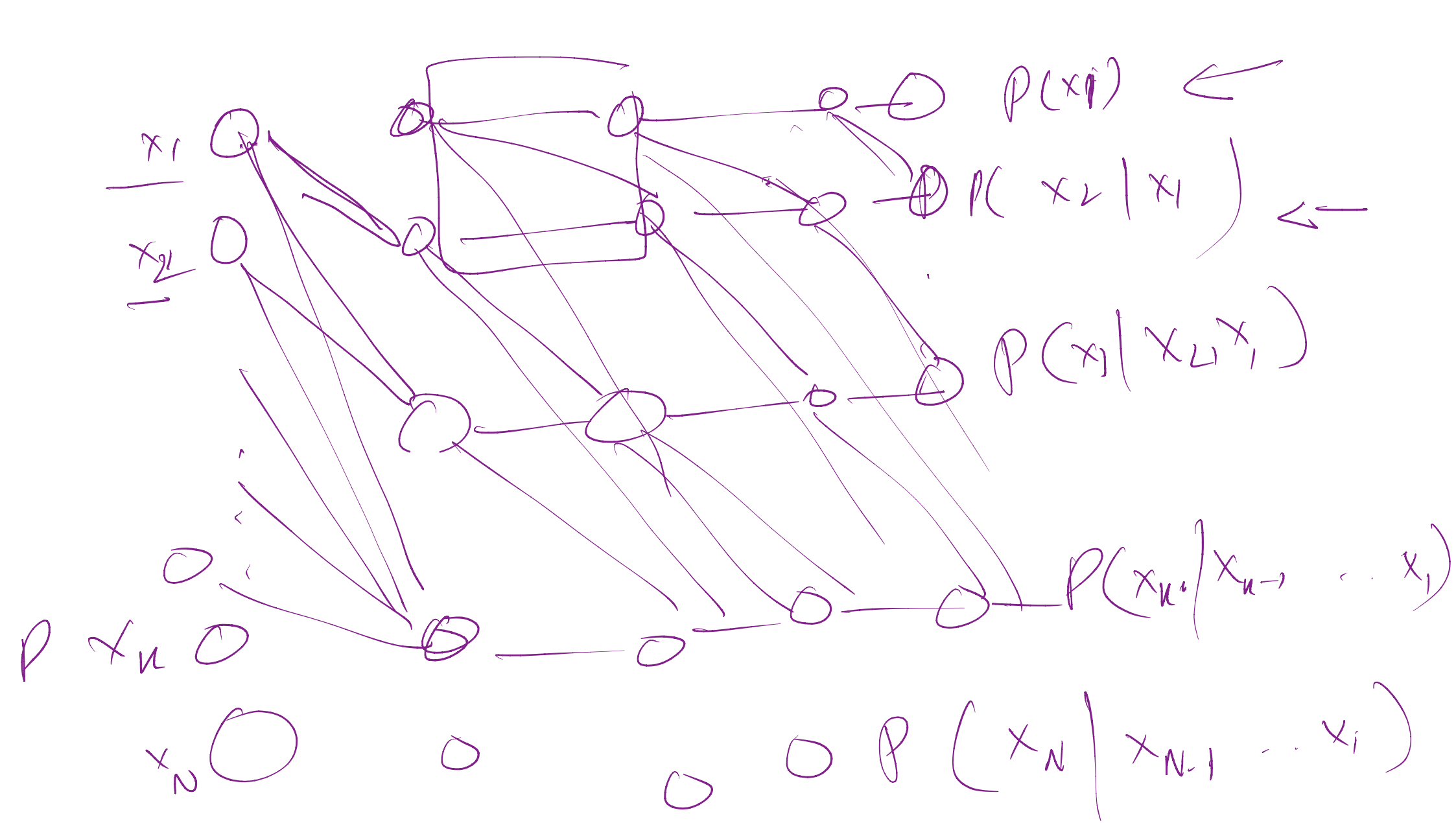
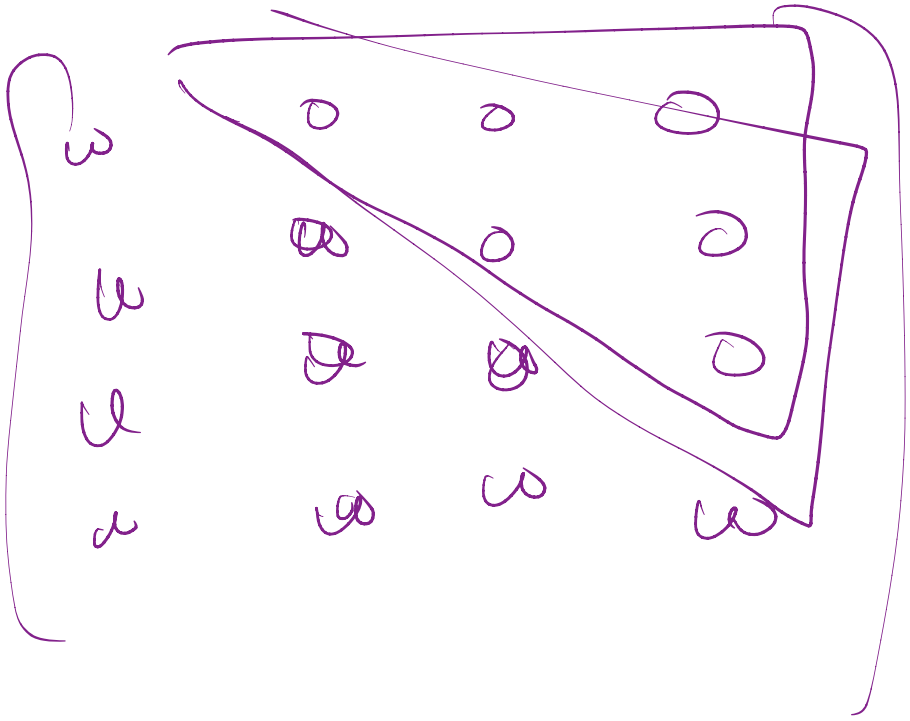


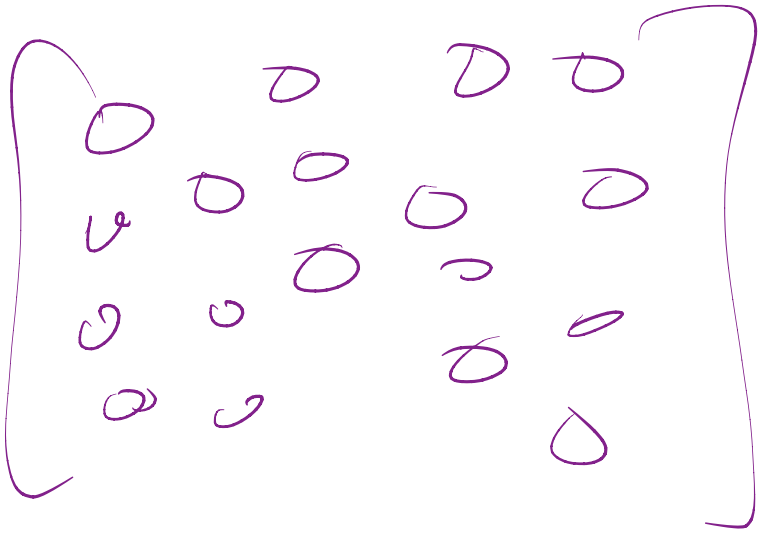
Figure 15: A fully visible sigmoid belief network over 4 variables


x_1
x_2
x_N
$p(x_1)$
$p(x_2|x_1)$
$p(x_3|x_2,x_1)$
$p(x_N|x_{N-1}\ldots x_1)$
$p(x_N|x_{N-1}\ldots x_i)$

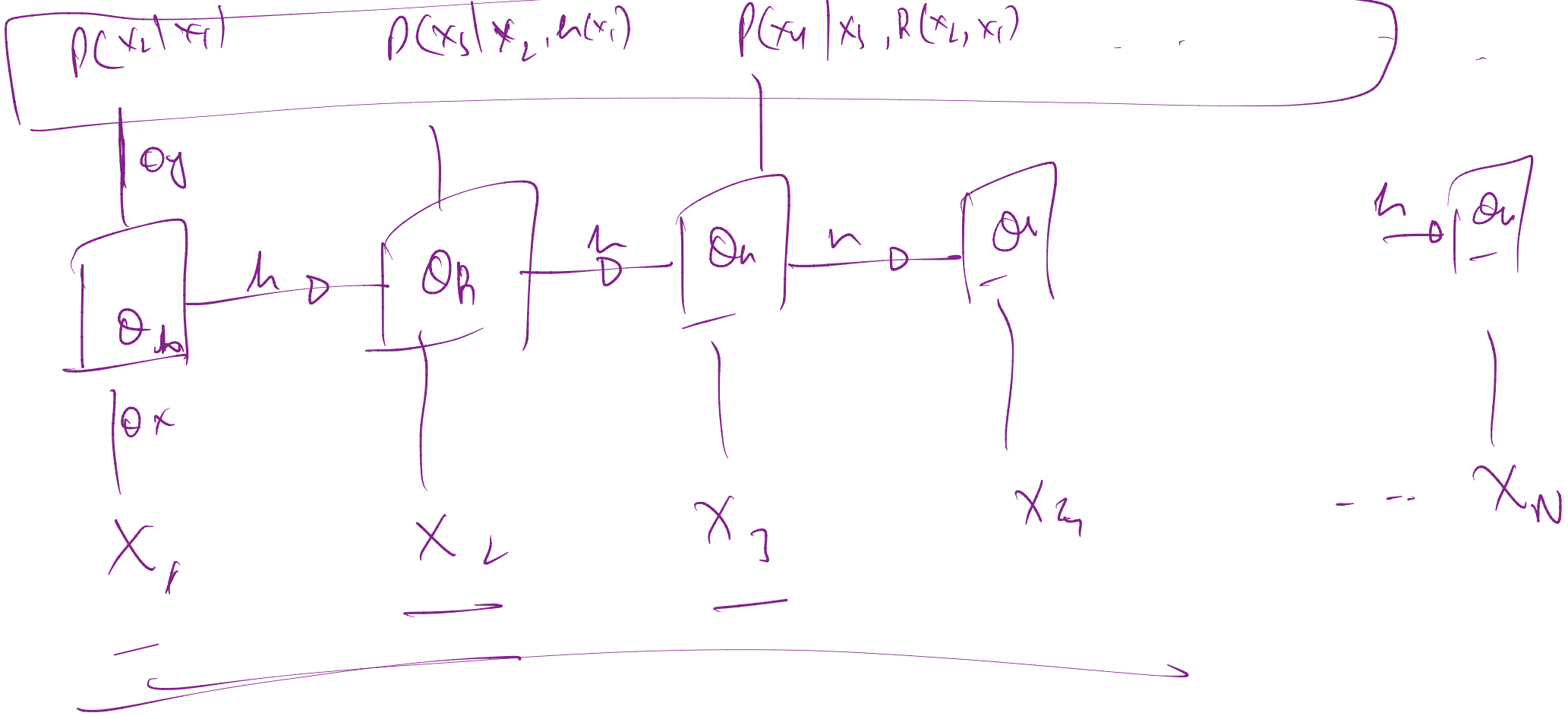
$W =$



$W' =$



$Z = W\underline{X} + \underline{b}$



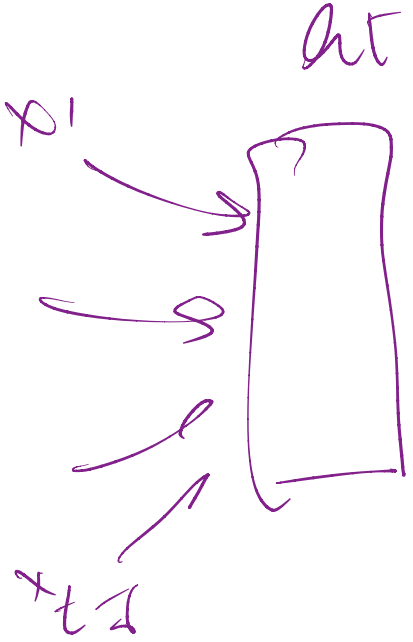
$O_h = W_h$

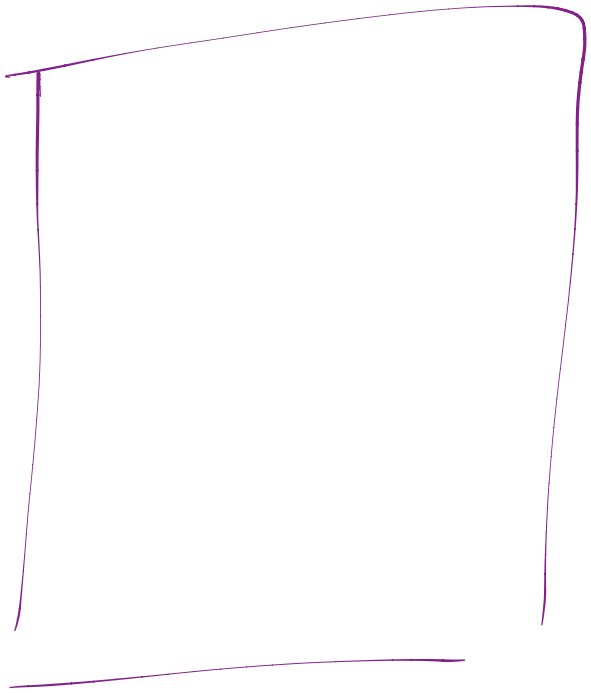
$O_x = W_x$

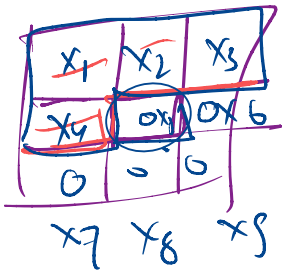
$O_y = W_y$

$$h_t = \sigma(W_h h_{t-1} + W_x x_t)$$

$$y_t = W_y h_t$$






x_1
$\bar{x}_2$
x_3
x_4
0x1
0x6
0
-
0
x_7
x_8
x_5

- The conditional variables $X_i|X_1, X_2, \cdots X_{i-1}$ are bernoulli with parameters

$$\hat{x}_i = \mathcal{P}(X_i = 1|x_1, \cdots, x_{i-1}; \alpha^i) = \mathcal{P}(X_i = 1|x_{<i}; \alpha^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

- To evaluate $\mathcal{P}(x)$, we just multiply all the conditionals.
- To sample new images, we sample each X_i chronologically.
  - Sample $\overline{x}_1 \sim \mathcal{P}(x_1)$ np.random.choice([1,0], p=$[\hat{x}, 1 - \hat{x}]$)
  - Sample $\overline{x}_2 \sim \mathcal{P}(x_2|X_1 = \overline{x}_1)$
  - Sample $\overline{x}_3 \sim \mathcal{P}(x_3|X_1 = \overline{x}_1, X_2 = \overline{x}_2)$
- Total number of parameters $= 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

- The conditional variables $X_i|X_1, X_2, \cdots X_{i-1}$ are bernoulli with parameters

$$\hat{x}_i = \mathcal{P}(X_i = 1|x_1, \cdots, x_{i-1}; \alpha^i) = \mathcal{P}(X_i = 1|x_{<i}; \alpha^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

- To evaluate $\mathcal{P}(x)$, we just multiply all the conditionals.
- To sample new images, we sample each X_i chronologically.
  - Sample $\overline{x}_1 \sim \mathcal{P}(x_1)$ np.random.choice([1,0], p=$[\hat{x}, 1 - \hat{x}]$)
  - Sample $\overline{x}_2 \sim \mathcal{P}(x_2|X_1 = \overline{x}_1)$
  - Sample $\overline{x}_3 \sim \mathcal{P}(x_3|X_1 = \overline{x}_1, X_2 = \overline{x}_2)$
- Total number of parameters $= 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

FVSBN Results

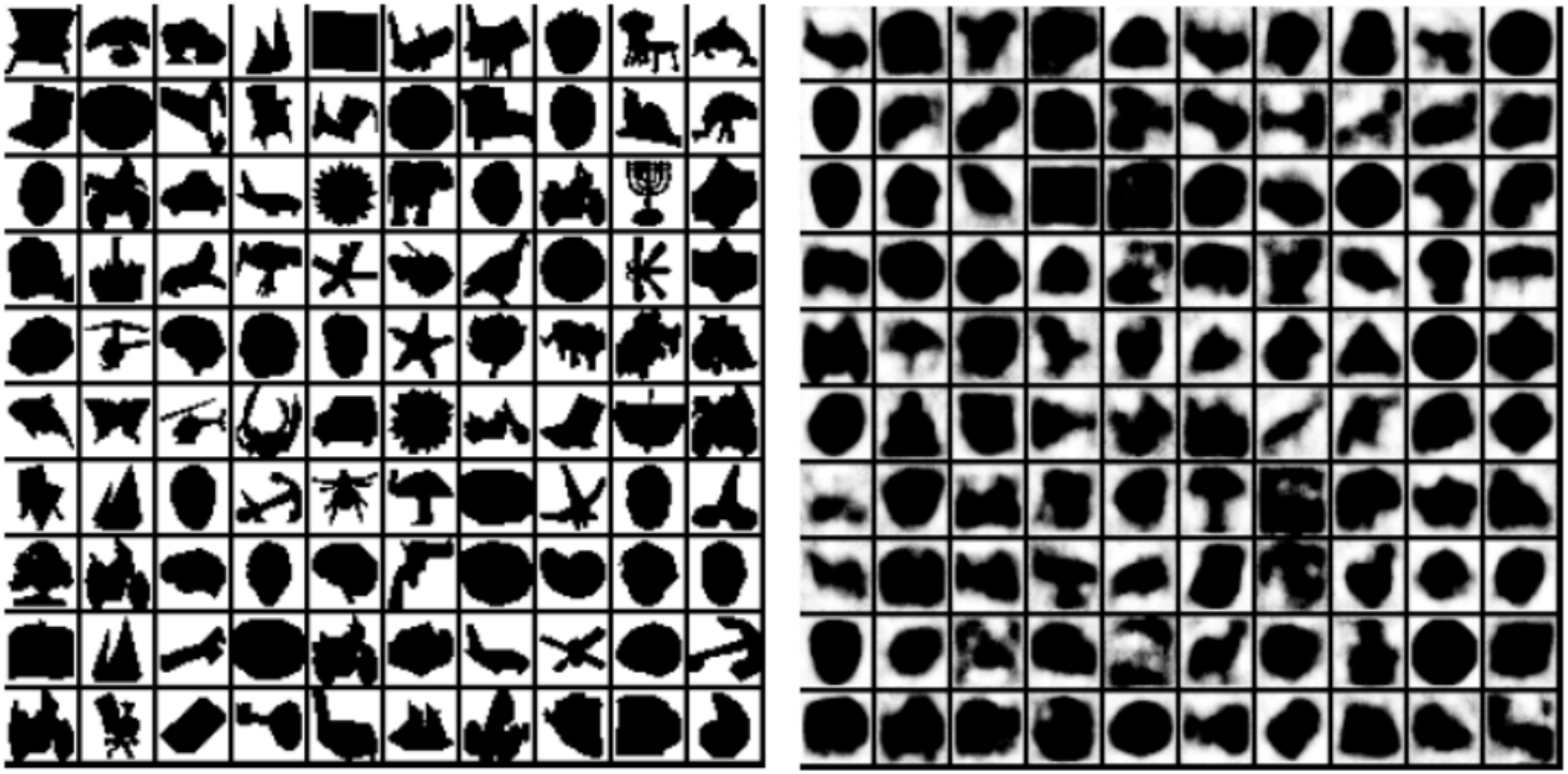
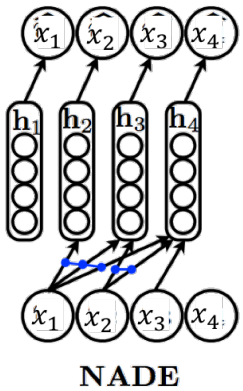


Figure 16: FVSBN results. Left: Training data. Right: Samples generated by model (Learning Deep Sigmoid Belief Networks with Data Augmentation, 2015)

NADE uses a neural network layer instead of just logistic regression to improve model performance.

$$h_i = \sigma(\mathbf{W}_{.,<i} x_{<i} + c)$$

$$f_i(x_1, x_2, \cdots, x_{i-1}) = \sigma(\alpha^{(i)} h_i + b_i)$$



NADE

Figure 17: A neural autoregressive density estimator over four variables. Blue connections donate shared weights.

NADE Results

Figure 18: NADE results. Left: Samples generated by model. Right: Conditional Probabilities $\hat{x}_i$ (The Neural Autoregressive Distribution Estimator, 2011)

Non-Binary Outputs

- What if the output variable is not binary? For example, we have to predict pixel value between 0 and 255.
- One solution: Let $\hat{\mathbf{x}}_i$ parameterize a categorical distribution

$$h_i = \sigma(\mathbf{W}_{.,<i} x_{<i} + c) \tag{1}$$

$$\mathcal{P}(x_i | x_1, \cdots, x_{i-1}) = \text{Cat}(p_i^1, \cdots, p_i^K) \tag{2}$$

$$\hat{\mathbf{x}}_i = (p_i^1, \cdots, p_i^K) = \text{softmax}(A_i h_i + b_i) \tag{3}$$

- Softmax generalizes the sigmoid function $\sigma(\cdot)$ and transforms a vector of K numbers into a vector of K probabilities (non-negative, sum to 1).

- How to model continuous random variables $X_i \in \mathbb{R}$?
- Solution: Let $\hat{\mathbf{x}}_i$ parameterize a continuous distribution.
- This was introduced in RNADE.
- $\hat{\mathbf{x}}_i$ defines the mean and stddev of Gaussian Random Variable (μ_i^j, σ_i^j)

Learning Autoregressive Models

- The goal of learning is to return a model $\hat{\mathcal{M}}$ that precisely captures the distribution $\mathcal{P}_{data}$ from which our data was sampled
- This is in general not achievable because of
  - limited data only provides a rough approximation of the true underlying distribution
  - computational reasons
- Example. Suppose we represent each MNIST digit with a vector $\mathbf{X}$ of 784 binary variables (black vs. white pixel). How many possible states (= possible images) in the model? $2^{784} \approx 10^{236}$. Even 10^7 training examples provide extremely sparse coverage!
- We want to select $\hat{\mathcal{M}}$ to construct the "best" approximation to the underlying distribution $\mathcal{P}_{data}$

- We want to learn the full distribution so that later we can answer any probabilistic inference query
- We want to construct $\mathcal{P}_\theta$ as "close" as possible to $\mathcal{P}_{data}$ (recall we assume we are given a dataset $\mathcal{D}$ of samples from $\mathcal{P}_{data}$)

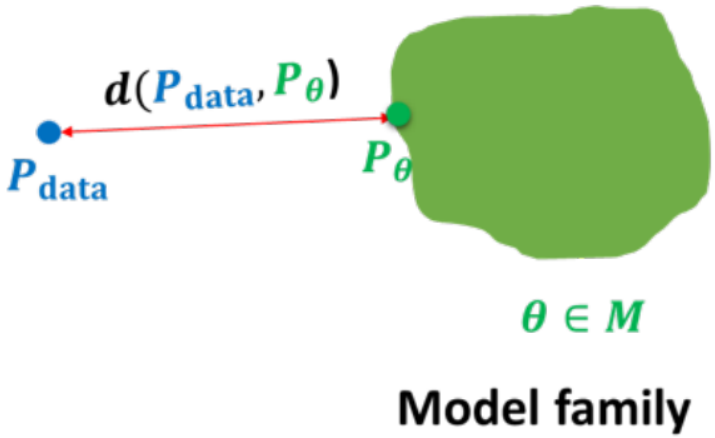


Figure 19: Learning a generative model. $d(\cdot)$ is a distance measure.

KL Divergence

- How do we evaluate "closeness" between model and data distribution?
- **Kullback-Leibler divergence** (KL-divergence) is one possibility:

$$D(\mathcal{P}_{data}||\mathcal{P}_\theta) = E_{x\sim\mathcal{P}_{data}}\left[log\left(\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}\right)\right] = \sum_x \mathcal{P}_{data}(x) log\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}$$

- $D(\mathcal{P}_{data}||\mathcal{P}_\theta)$ iff the two distributions are the same.
- It measures the "compression loss" (in bits) of using $\mathcal{P}_\theta$ instead of $\mathcal{P}_{data}$.

Expected Log-Likelihood

- We can simplify this somewhat:

$$\begin{aligned} D(\mathcal{P}_{data}||\mathcal{P}_\theta) &= E_{x\sim\mathcal{P}_{data}}\left[log\left(\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}\right)\right] \\ &= E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_{data}(x)] - E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \end{aligned} \tag{4}$$

- The first term does not depend on $\mathcal{P}_\theta$. Then, *minimizing* KL divergence is equivalent to *maximizing* the **expected log-likelihood**

$$\begin{aligned} \arg\min_{\mathcal{P}_\theta} D(\mathcal{P}_{data}||\mathcal{P}_\theta) &= \arg\min_{\mathcal{P}_\theta} -E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \\ &= \arg\max_{\mathcal{P}_\theta} E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \end{aligned} \tag{5}$$

  - Asks that $\mathcal{P}_\theta$ assign high probability to instances sampled from $\mathcal{P}_{data}$, so as to reflect the true distribution
  - Because of log, samples x where $\mathcal{P}_\theta(x) \approx 0$ weigh heavily in objective

- Although we can now compare models, since we are ignoring $H(\mathcal{P}_{data})$, we don't know how close we are to the optimum.
- Problem: In general we do not know $\mathcal{P}_{data}$

Maximum Likelihood

- Approximate the expected log-likelihood

$$E_{x \sim \mathcal{P}_{data}}[log \mathcal{P}_\theta(x)]$$

with the empirical log-likelihood:

$$E_{\mathcal{D}} = \frac{1}{\mathcal{D}} \sum_{x \in \mathcal{D}} log \mathcal{P}_\theta(x)$$

- **Maximum likelihood learning** is then:

$$\arg\max_{\mathcal{P}_\theta} \frac{1}{\mathcal{D}} \sum_{x \in \mathcal{D}} log \mathcal{P}_\theta(x)$$

Recurrent Neural Networks (RNN)

- The next form of autoregressive models
- **Challenge**: model $\mathcal{P}(x_t|x_{1:t-1};\alpha^t)$. "History" $x_{1:t-1}$ keeps getting longer.
- **Idea**: keep a summary and recursively update it

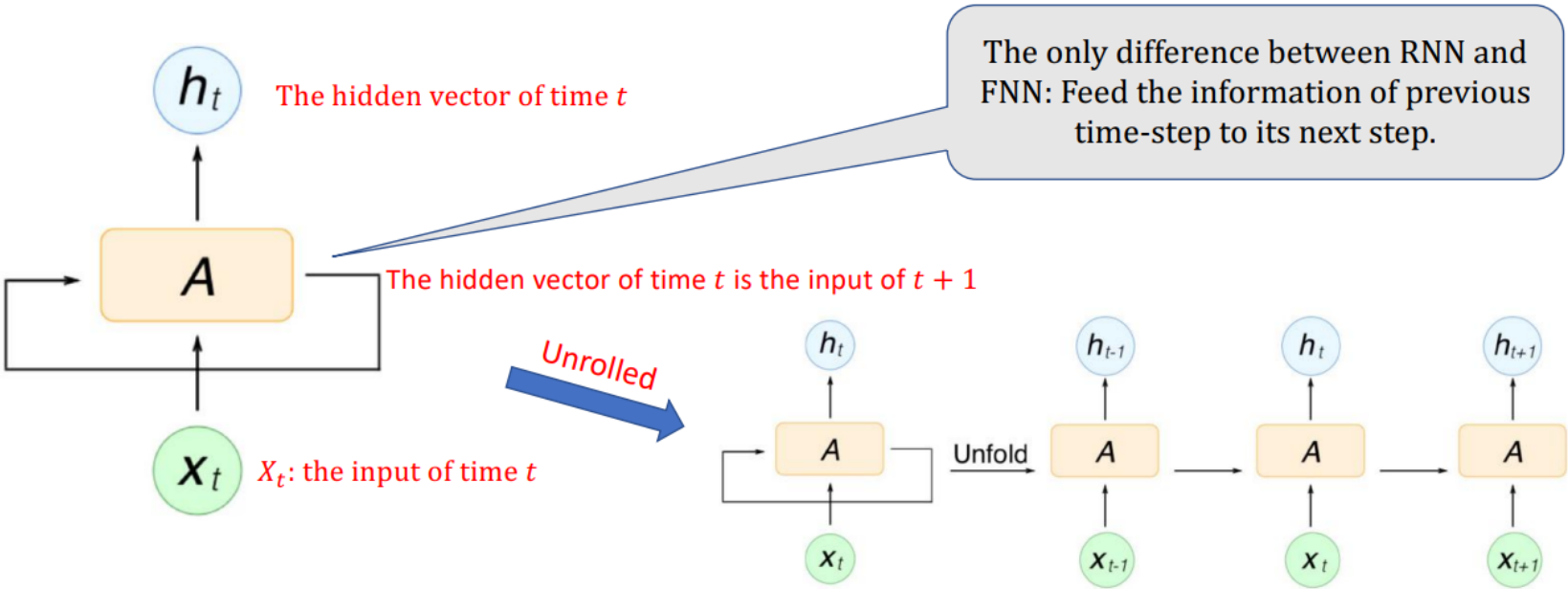


Figure 20: Recurrent Neural Network (RNN) sample architecture.

Recurrent Neural Networks (RNN) (cont.)

Pros:

- Can be applied to sequences of arbitrary length.
- Very general: For every computable function, there exists a finite RNN that can compute it

Cons:

- Still requires an ordering
- Sequential likelihood evaluation (very slow for training)
- Sequential generation (unavoidable in an autoregressive model)
- Can be difficult to train (vanishing/exploding gradients)

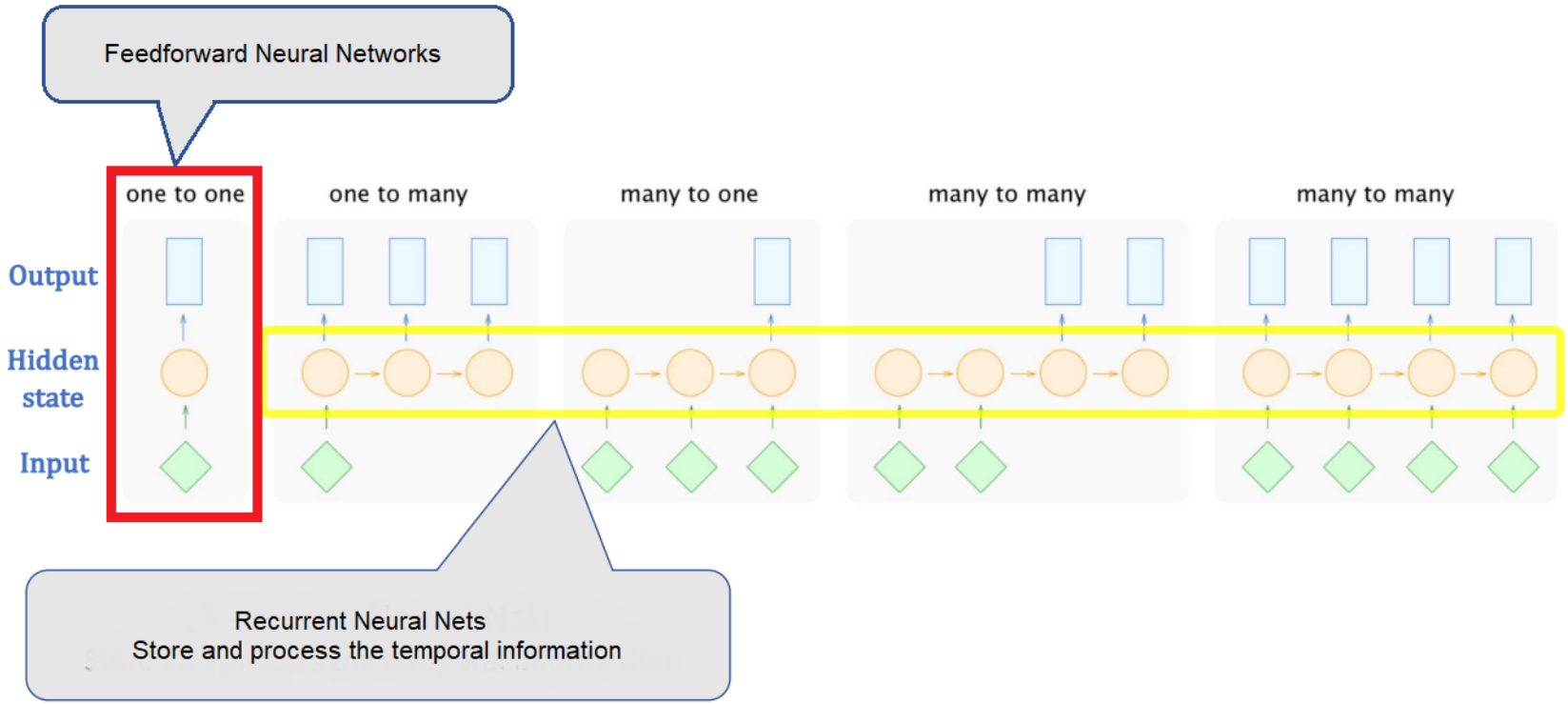


Figure 21: Different sequential modeling types

Image Captioning

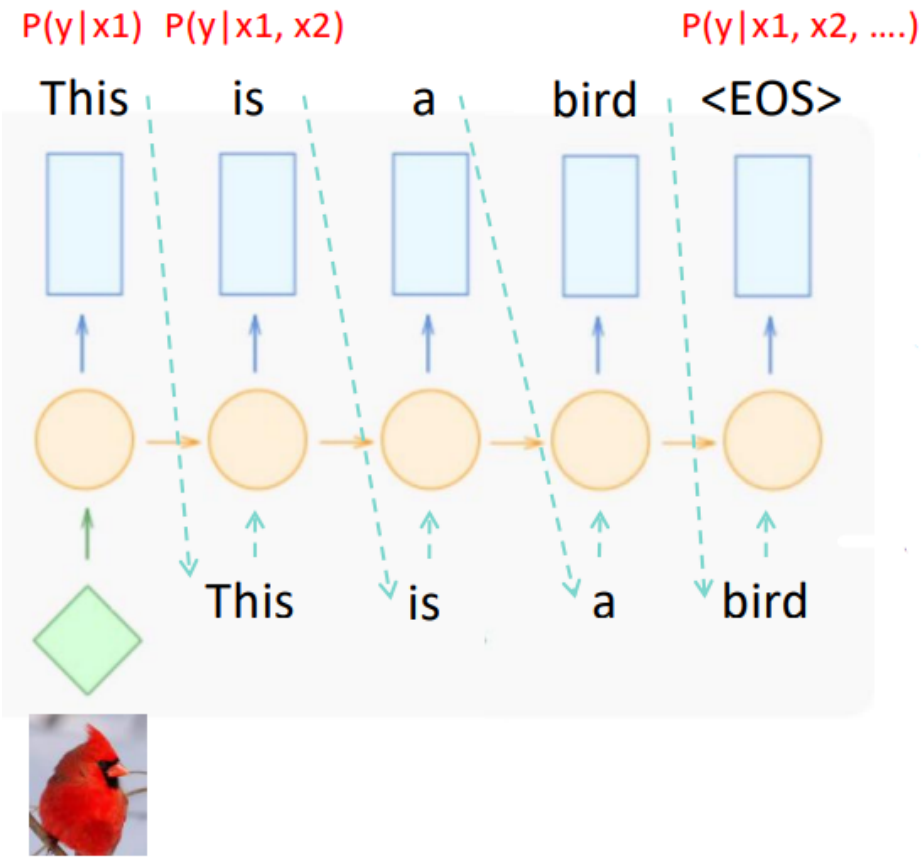


Figure 22: Image captioning with RNNs

Text Generation

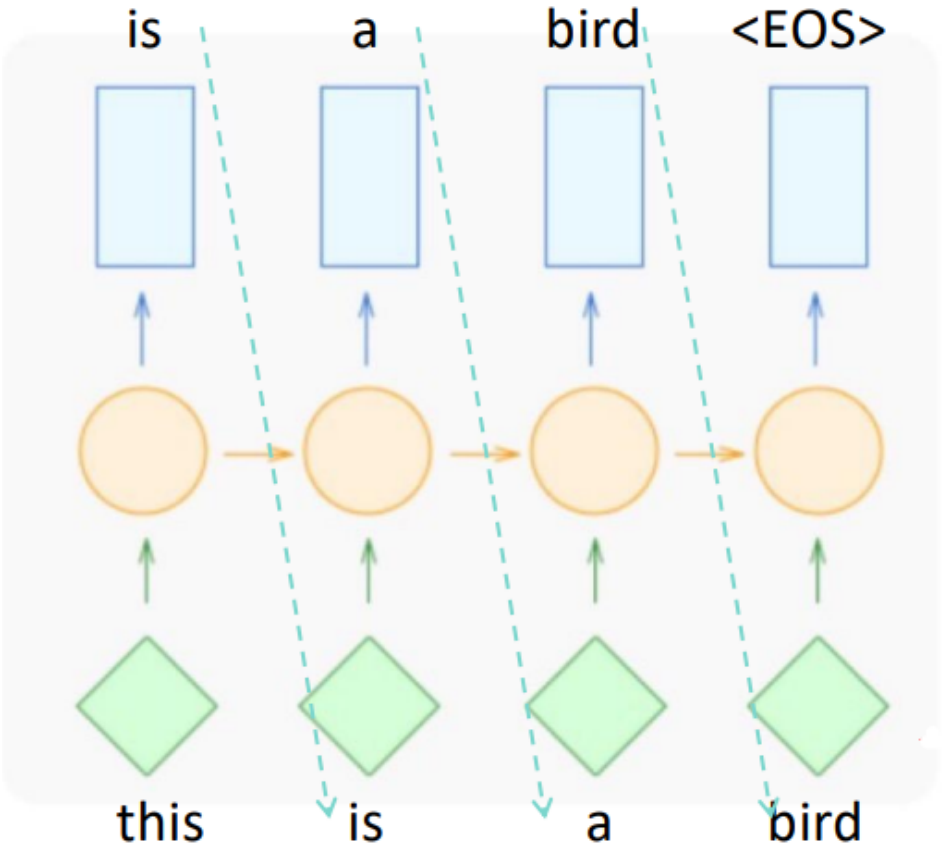


Figure 23: Text generation with RNNs

Chatbot

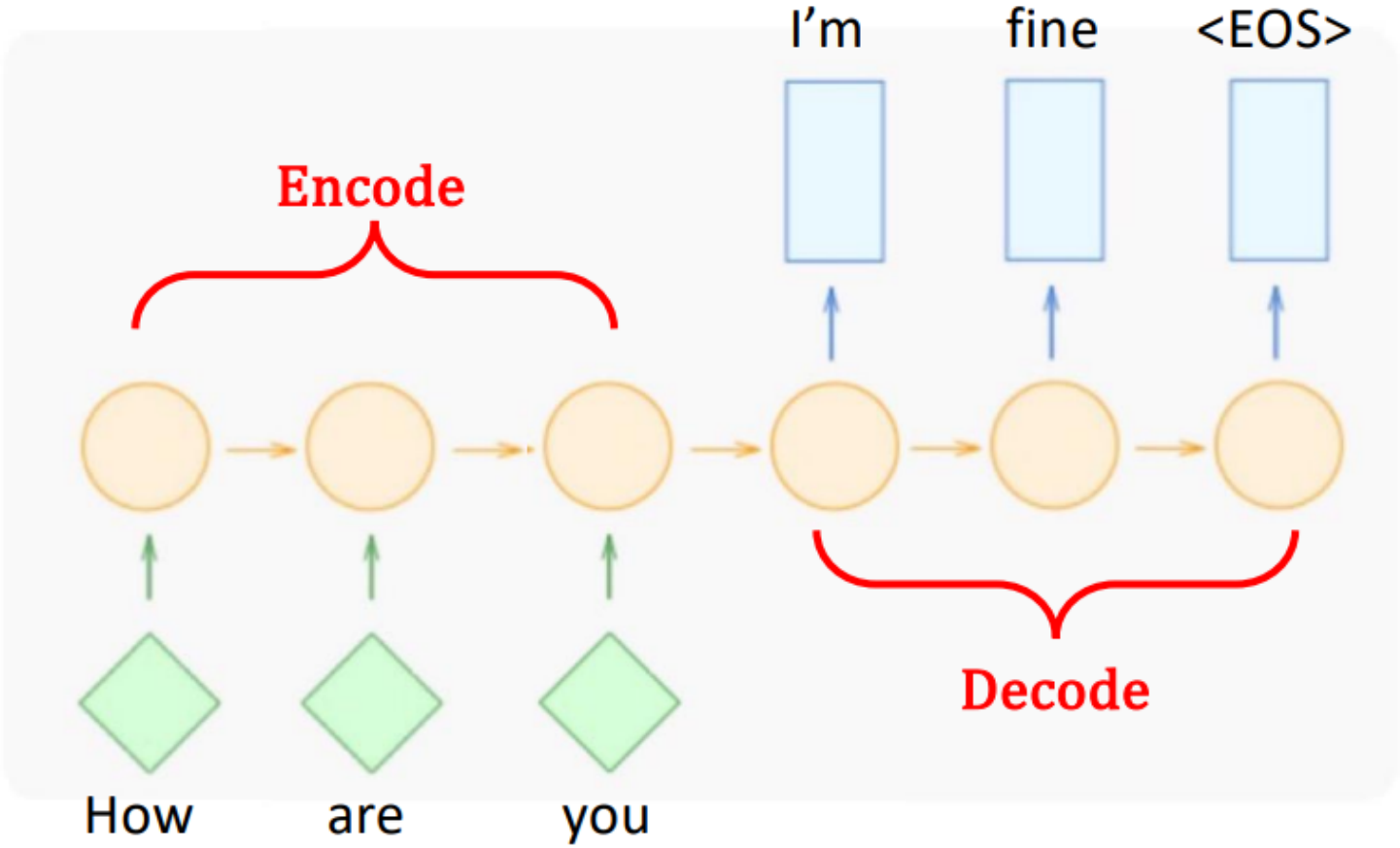


Figure 24: Chatbot with RNNs

- **PixelRNNs**: Sequentially predict each pixel in the image.

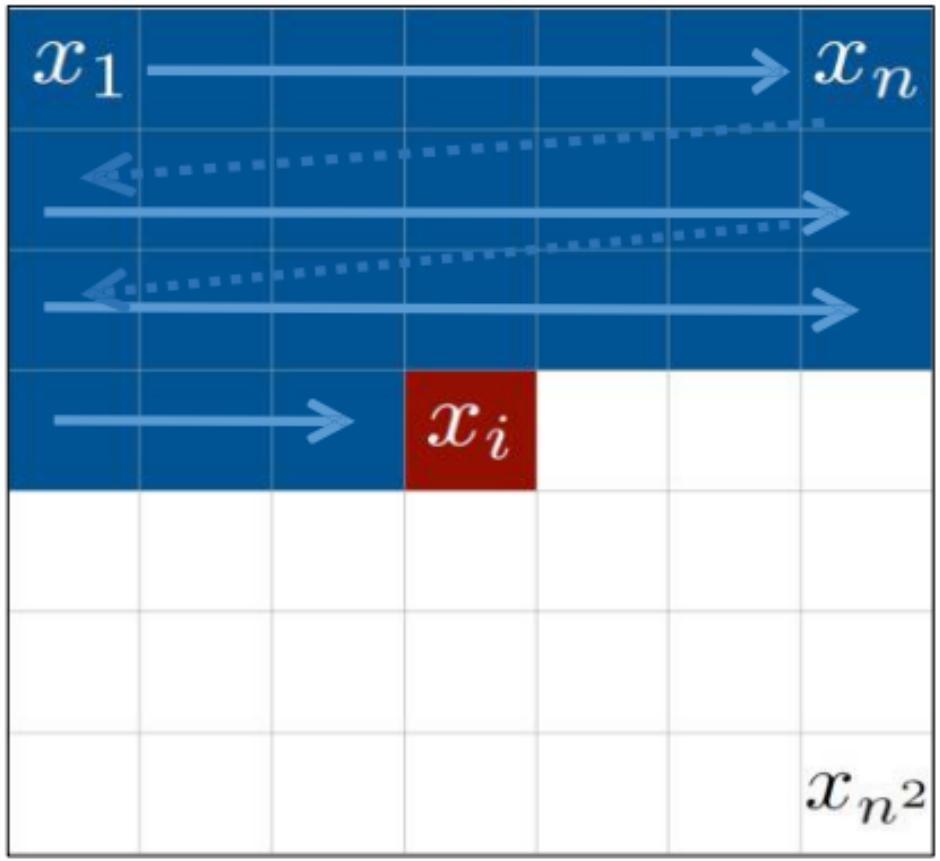


Figure 25: Image generation with pixelRNN

- PixelRNN results

Figure 26: Image generation with pixelRNN. Model trained on Imagenet (32 x 32 pixels)

- **PixelCNNs**: Use the neighbour pixels to predict the new pixel.

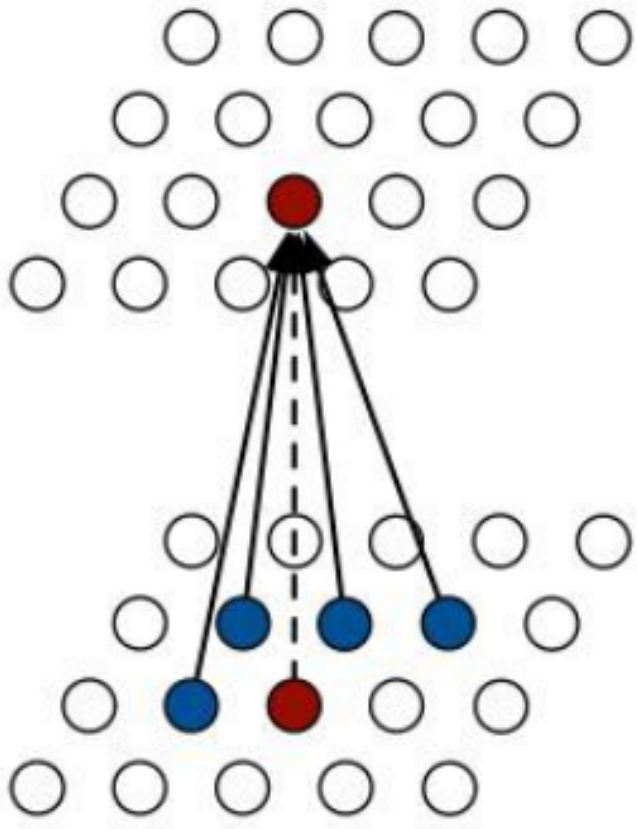


Figure 27: Image generation with pixelCNN

- PixelCNN results

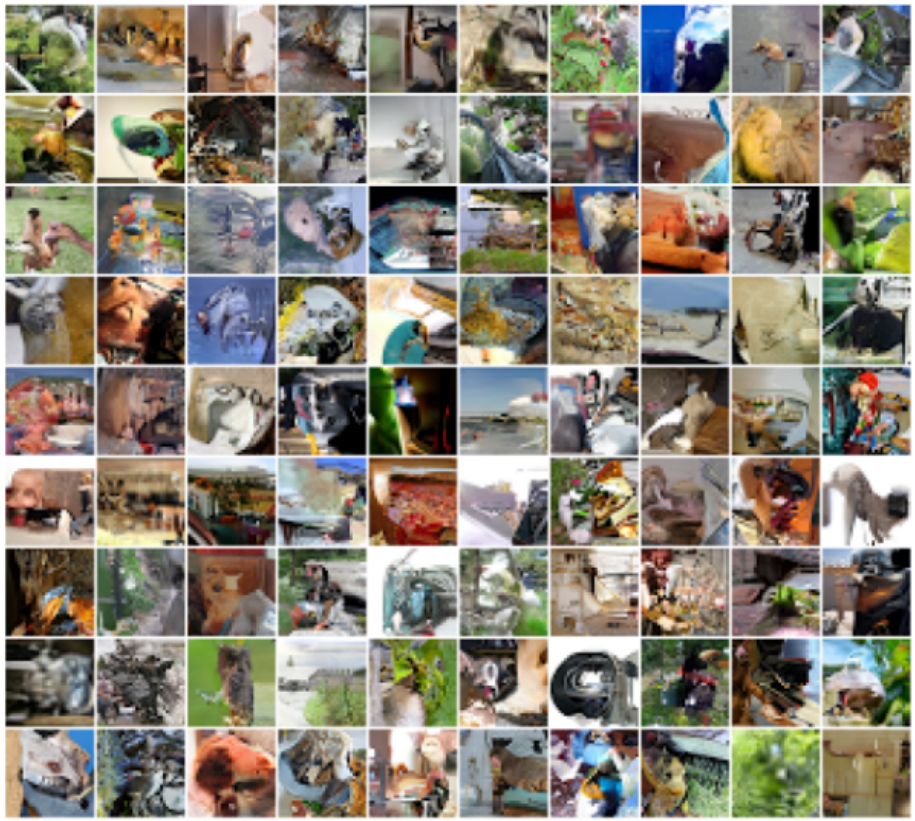


Figure 28: Image generation with pixelCNN. Model trained on Imagenet (32 x 32 pixels)

Summary of Autoregressive Models

- Specify an order for the data.
- Sample a new datapoint recursively
- Easy to compute $\mathcal{P}(X = \overline{x})$. Just compute each $\mathcal{P}(x_i|x_{<i})$ and mulitply them together (or sum their logarithms).
- Easy to extend to multi-class discrete and continuous variables.
- But no natural way to get features, cluster points, do unsupervised learning

Autoencoders

- Family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.
- The idea is to project the input into a latent space and then reconstruct the input from that latent space representation
- Consist of two parts: Encoder and decode.
  - Encoder projects the input to a latent space Z.
  - Decoder takes the encoded embedding vector and reconstructs the input from it.
  - We also use altered versions of input as output which can be even more interesting.

Autoencoders (cont.)

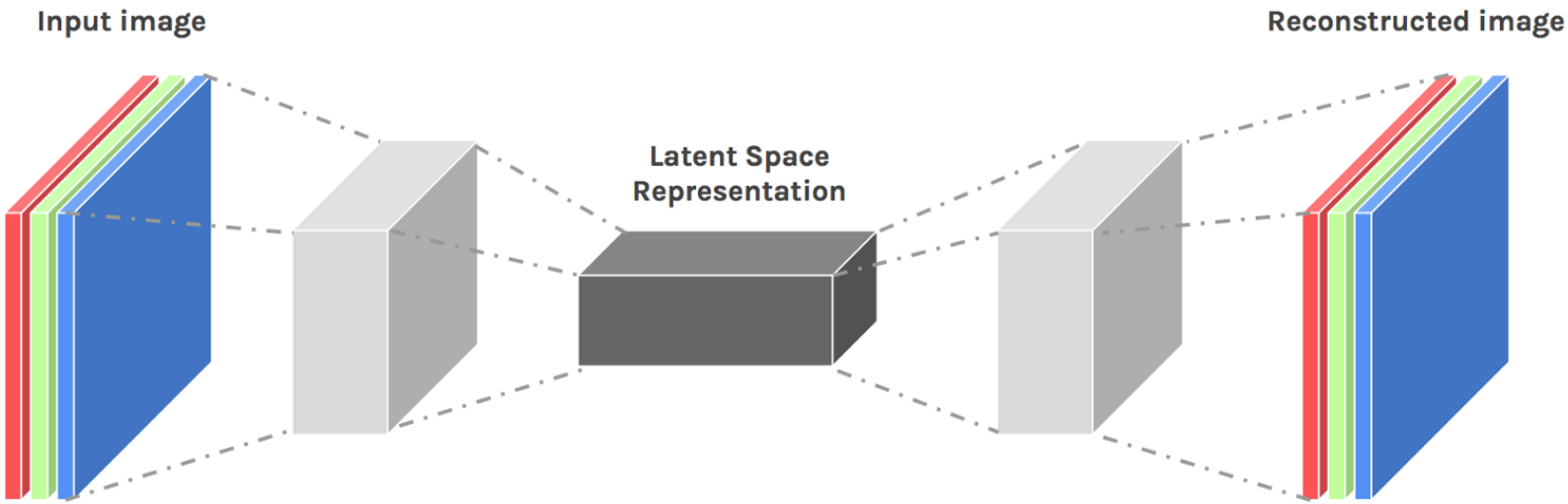


Figure 29: Autoencoder architecture

Autoencoders (cont.)

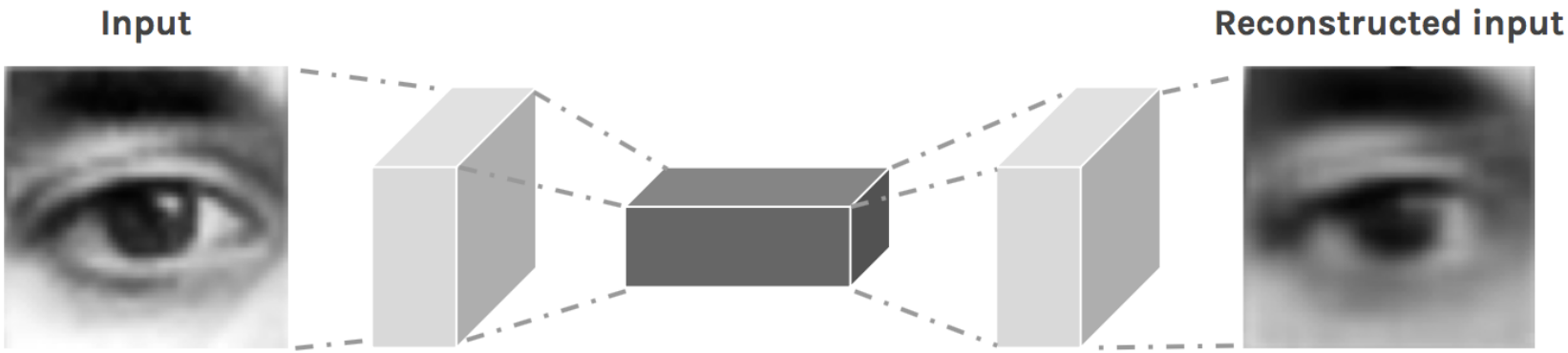


Figure 30: Sample Autoencoder

Autoencoders - Interactive Demo

https://douglasduhaime.com/posts/visualizing-latent-spaces.html

Autoencoders as generative models

- Autoencoders project data into a latent space Z.
- What if we sample a new embedding vector from Z and then have the decoder reconstruct the image from it?
- **Does not work**. Autoencoders just learn a function that maps input to output. The learned latent space is too discontinous to work as a generative model.

Autoencoders as generative models (cont.)

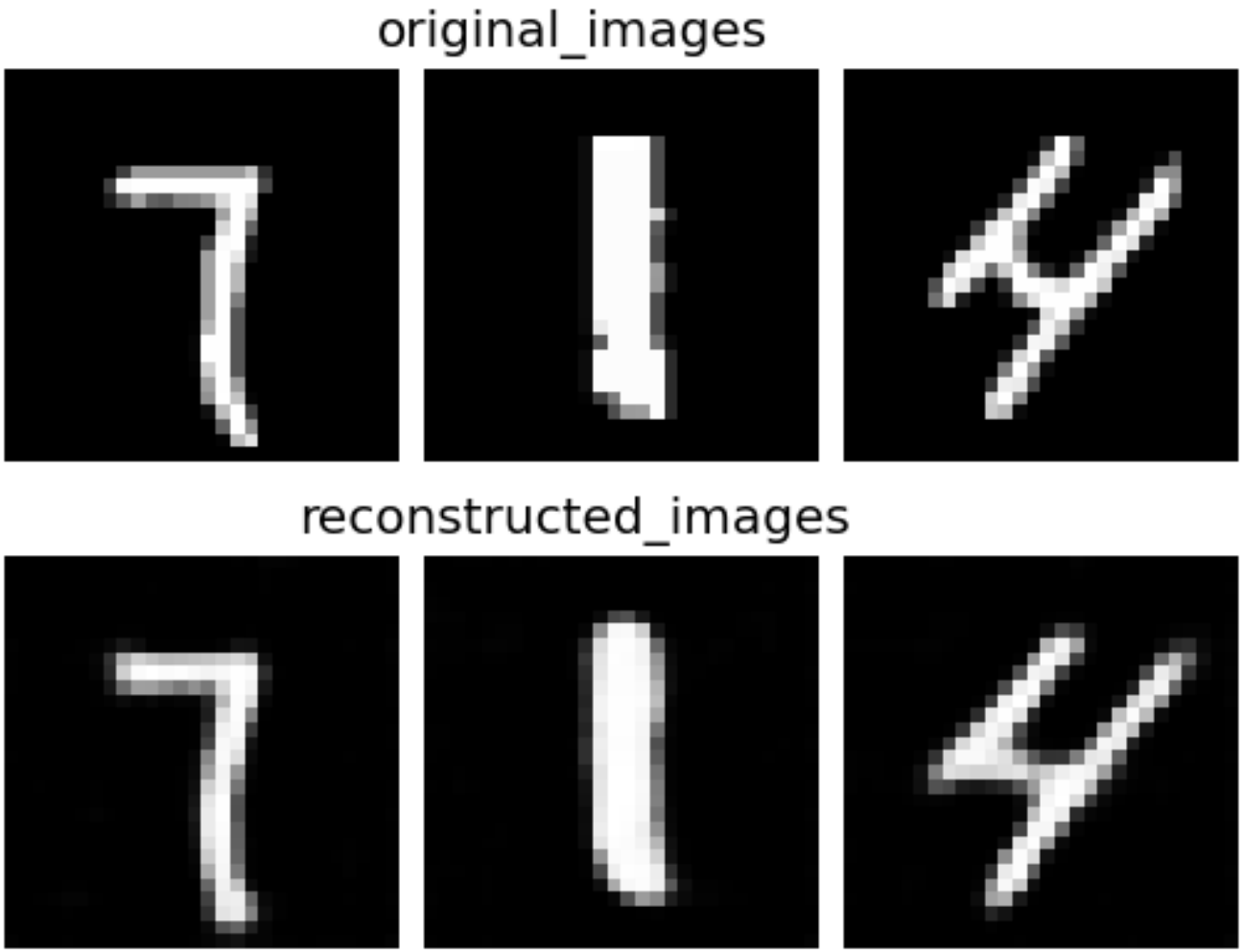


Figure 31: Image reconstruction with autoencoder trained on MNIST digits

Autoencoders as generative models (cont.)

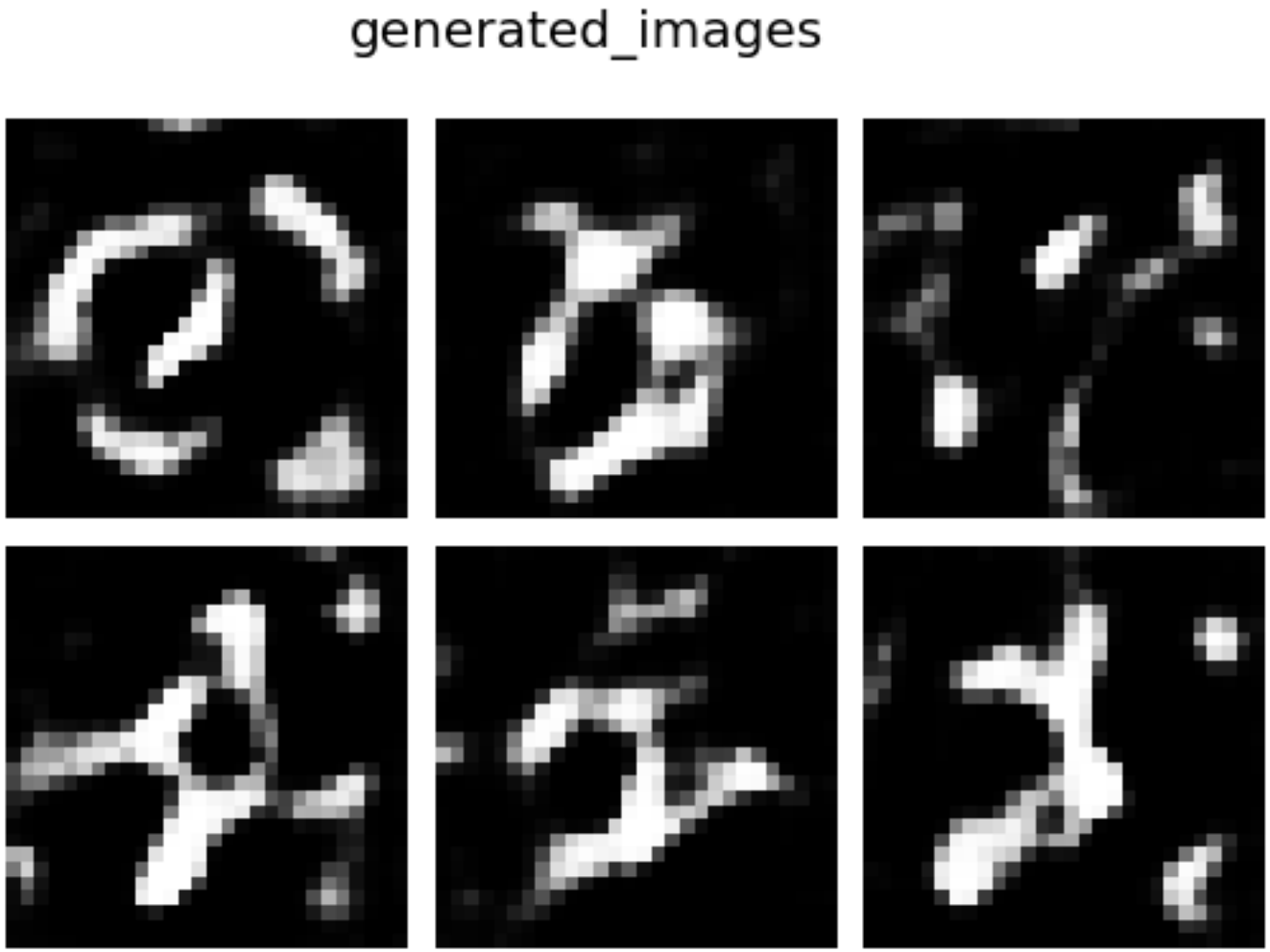


Figure 32: Image generation with autoencoder trained on MNIST digits. Encoding vector sampled from latent space Z and the passed to decoder.

Autoencoders - Applications

- While autoencoders themselves have very low generative power, we will soon talk about a type of autoencoders called **Variational Autoencoders** which are specifically designed for generative modeling.
- Other use cases of Autoencoders include:
  - Data encoding and dimensionality reduction
  - Image denoising and super-resolution
  - Image completion
  - Image colorization

Autoencoders - Applications (cont.)

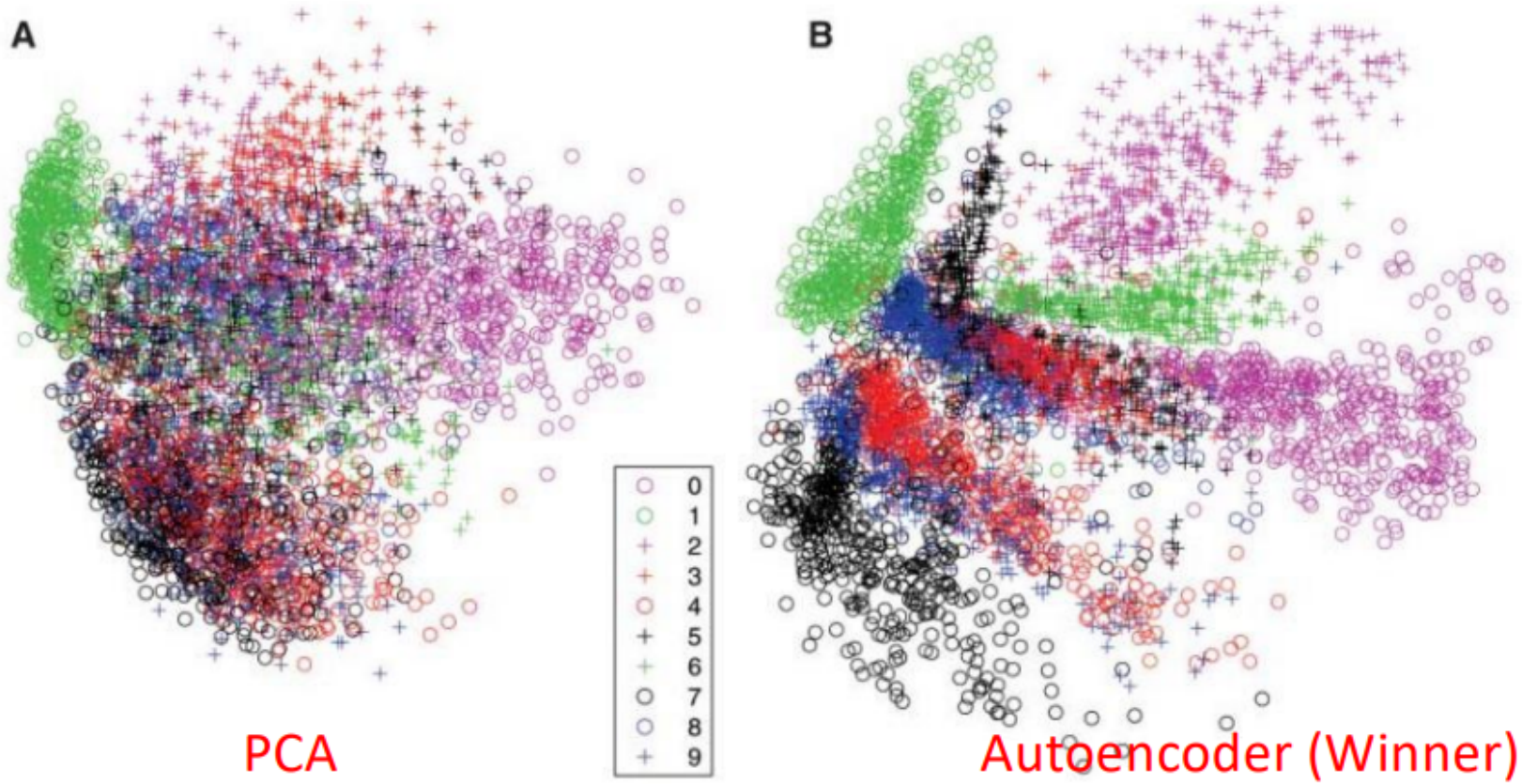


Figure 33: t-SNE visualization on MNIST digits dataset. PCA vs. Autoencoders. The image vector is projected into $\mathbb{R}^2$.

Figure 34: Image super-resolution using Autoencoders

Autoencoders - Applications (cont.)

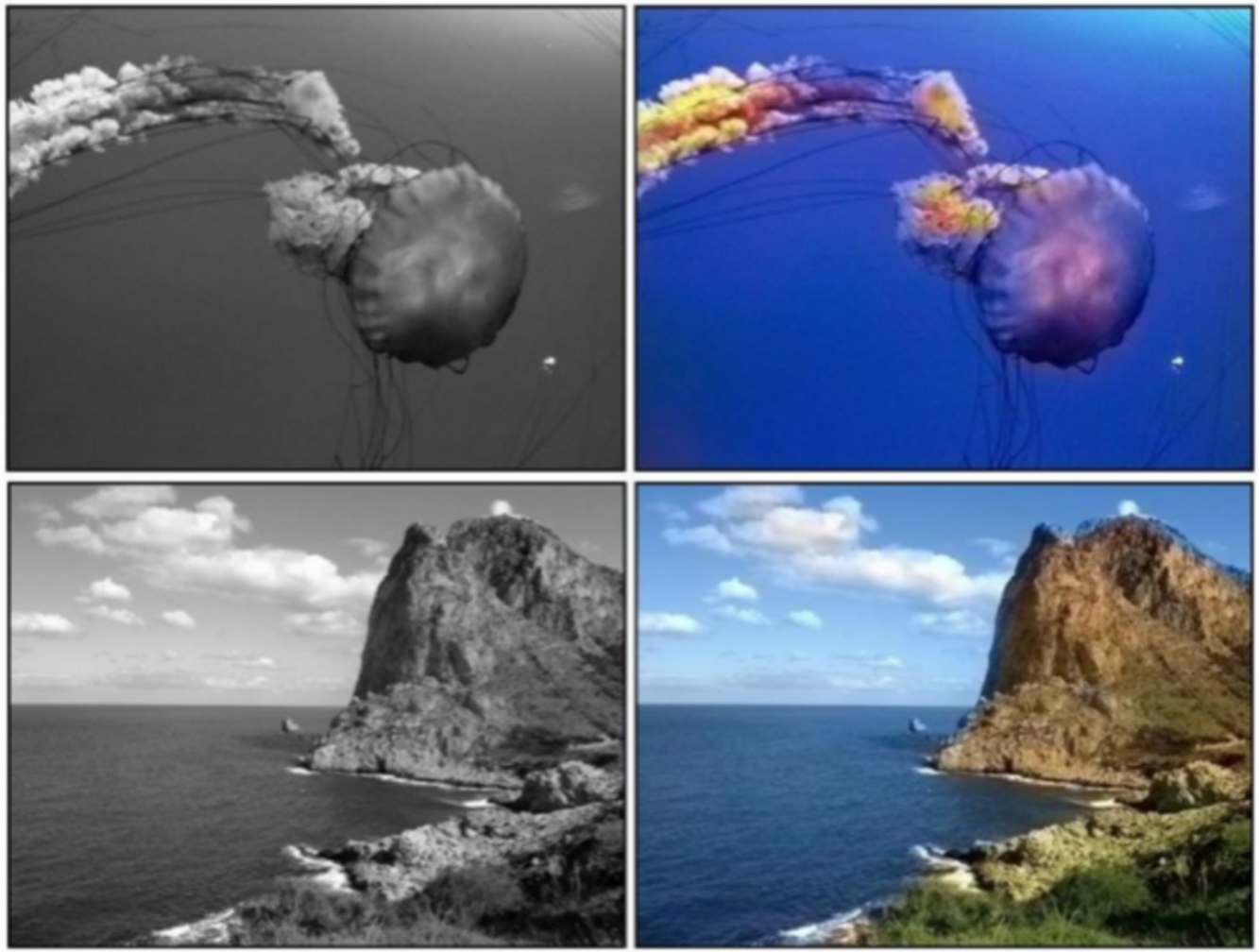


Figure 35: Image colorization using Autoencoders

References

Reference Slides

- Fei-Fei Li "Generative Deep Learning" CS231
- Hao Dong "Deep Generative Models"