UC Berkeley · CSW182 | [Deep Learning]
Designing, Visualizing and Understanding Deep Neural Networks (2021)

CSW182 (2021)· 课程资料包 @ShowMeAI

视频
中英双语字幕

课件
一键打包下载

笔记
官方笔记翻译

代码
作业项目解析

视频 · B 站 [扫码或点击链接]
https://www.bilibili.com/video/BV1Ff4y1n7ar

课件 & 代码 · 博客 [扫码或点击链接]
http://blog.showmeai.tech/berkeley-csw182

Berkeley　循环神经网络　可视化　梯度策略
Q-Learning　风格迁移　模仿学习　元学习
计算机视觉　机器学习基础　生成模型　卷积网络

Awesome AI Courses Notes Cheatsheets 是 **ShowMeAI** 资料库的分支系列，覆盖最具知名度的 **TOP50+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n

 # Awesome AI Courses Notes Cheatsheets· 持续更新中

知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094

微信公众号

资料下载方式 2：扫码点击底部菜单栏

称为 **AI 内容创作者？** 回复 [添砖加瓦]

Generative Modeling

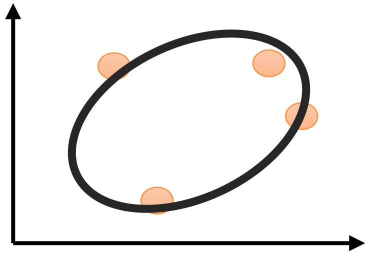
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley

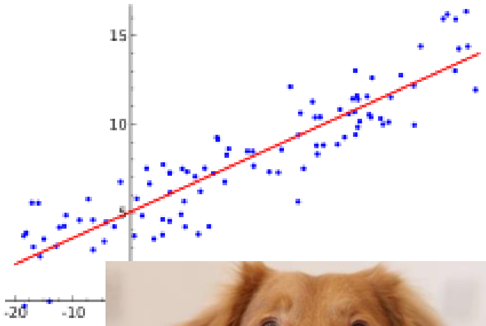
Probabilistic models

$p(x)$



Why would we want to do this?

$p(y|x)$



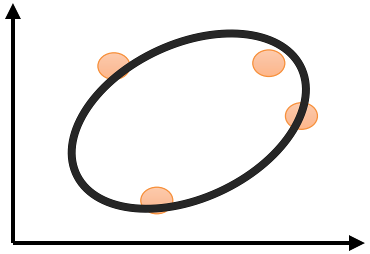
$f_\theta(x) = y$

[object label]

Generative models

$p(x)$

$\mathcal{N}(\mu, \sigma)$



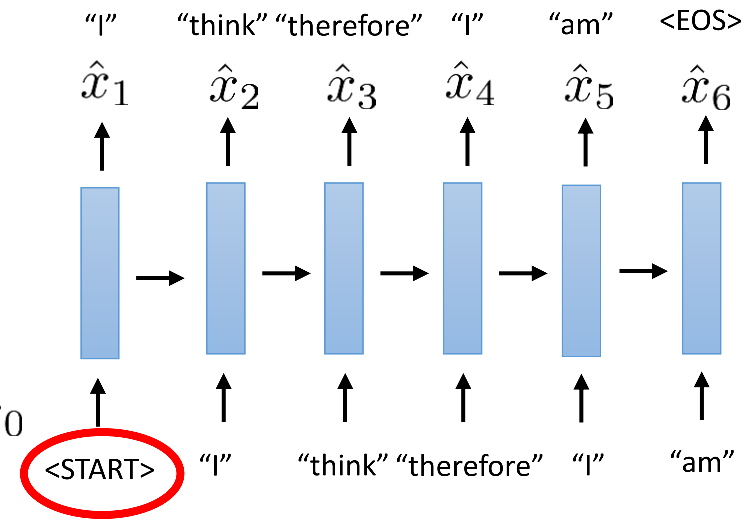
Today: can we go from “language models” to “everything models”?

This is called **unsupervised learning**

Just different ways to solve the same problem!

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

$$p(x) = p(x_1|x_0)p(x_2|x_{0:1})p(x_3|x_{0:2})p(x_4|x_{0:3})p(x_5|x_{0:4})p(x_6|x_{0:5})$$

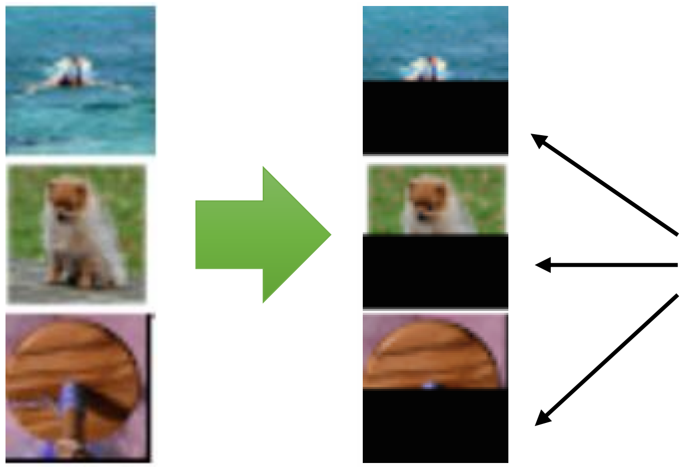


Why would we want to do this?

Same reasons as language modeling!

- Unsupervised pretraining on lots of data
- Representation learning
- Pretraining for later finetuning
- Actually generating things!

Can we “language model” images?

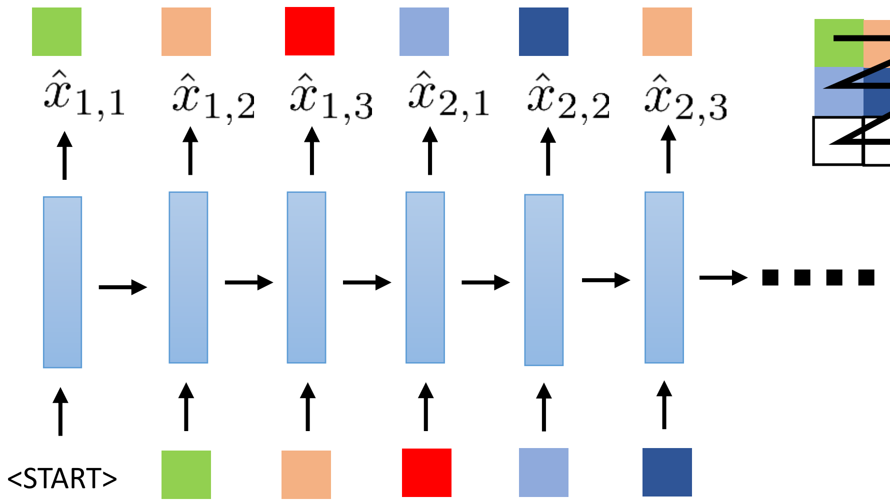


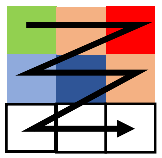
ask the “image language model” to fill in the blanks

before	now
time step = word	time step = pixel

some (arbitrary) ordering on pixels

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$





This is basically the main idea, but there are some details we need to figure out!

- How to order the pixels?
- What kind of model to use?

Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

Autoregressive generative models

Main principle for training:

1. Divide up x into dimensions $x_1, \dots, x_n$
2. Discretize each x_i into k values
3. Model $p(x)$ via the chain rule

each of these is just a softmax

$$p(x) = p(x_1)p(x_2|x_1)p(x_3|x_{1:2})p(x_4|x_{1:3})p(x_5|x_{1:4})p(x_6|x_{1:5})$$

4. Use your favorite sequence model to model $p(x)$

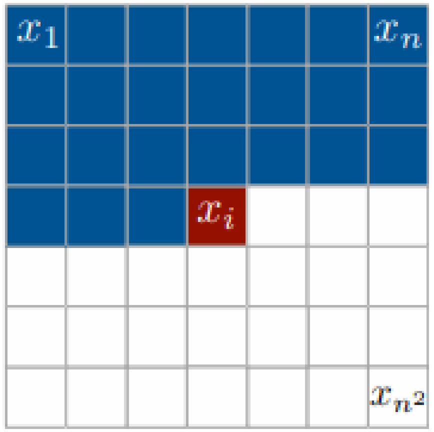
Using autoregressive generative models:

Sampling: ancestral sampling in sequence (x_1, then x_2, etc.)

Completion: feed in actual values for known x_i values

Representations: same idea as ELMo or BERT

PixelRNN



Pixels generated one at a time, left-to-right, top-to-bottom:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

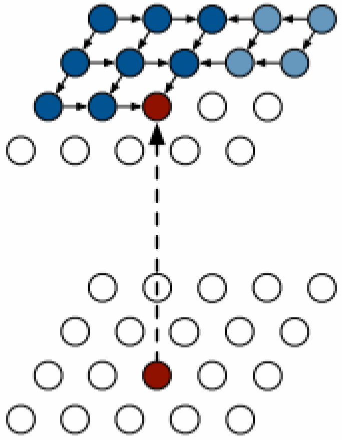
Generate one color channel at a time:

$$p(x_{i,R} | \mathbf{x}_{<i}) p(x_{i,G} | \mathbf{x}_{<i}, x_{i,R}) p(x_{i,B} | \mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

256-way softmax

Some practical considerations:

- It's very slow
- Row-by-row LSTMs might struggle to capture spatial context (pixels right above are "far away")
- Many practical improvements and better architectures are possible!

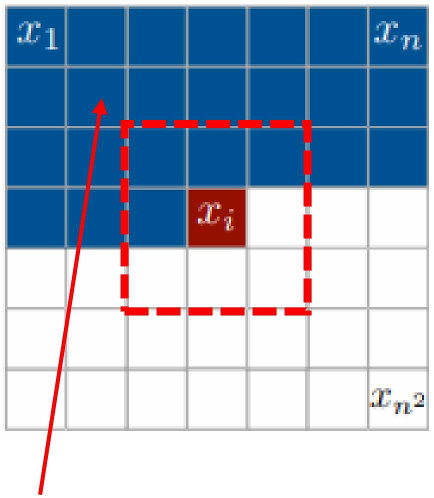


Diagonal BiLSTM

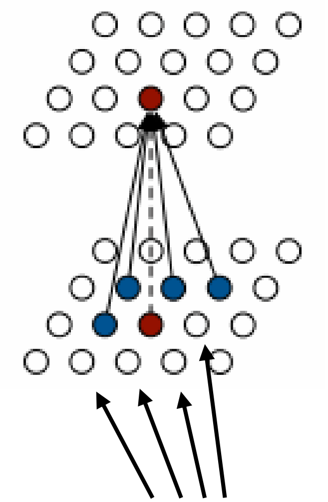
Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

PixelCNN

Idea: make this much faster by not building a full RNN over all pixels, but just using a convolution to determine the value of a pixel based on its neighborhood



this pixel still influences x_i! why?



these are **masked out** because they haven't been generated yet

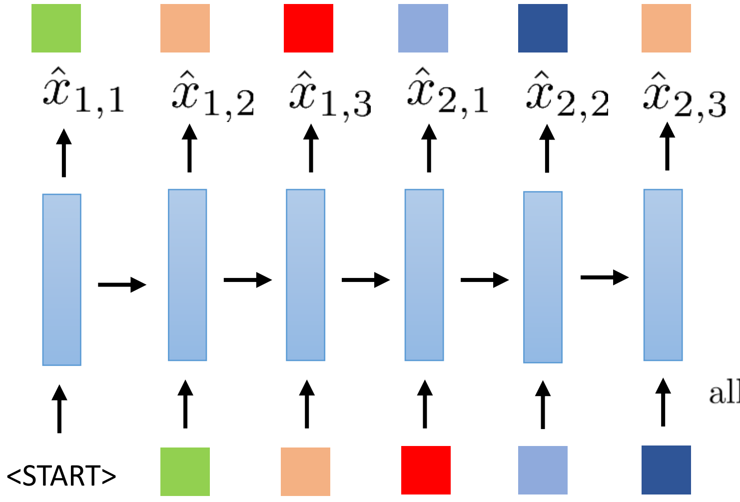
Question: can we parallelize this?

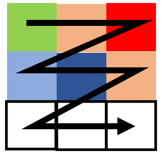
During **training**?

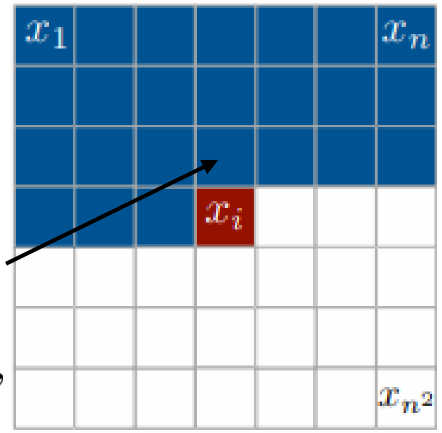
During **generation**?

Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

Pixel Transformer





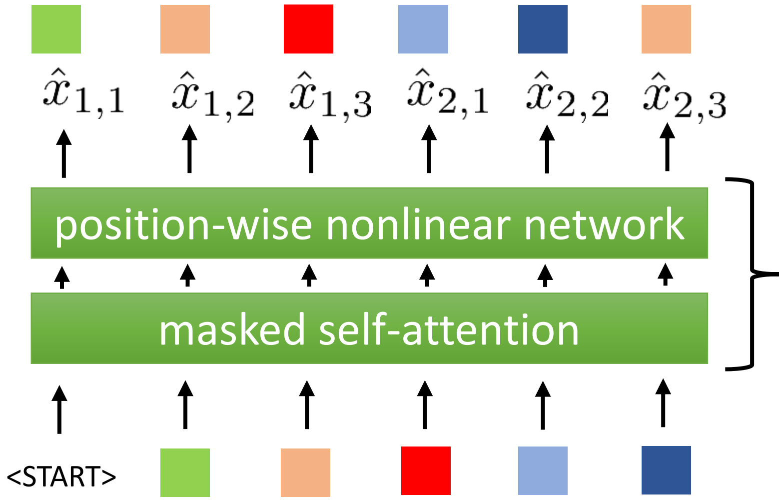


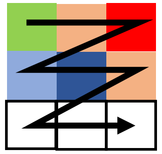
Problem: the number of pixels can be **huge**

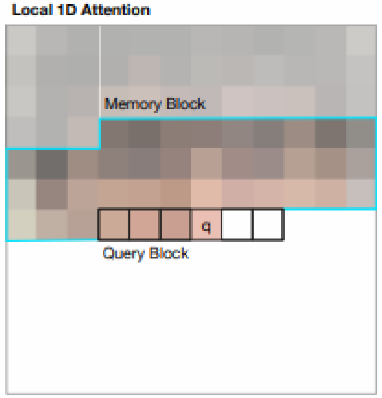
attention can become prohibitively expensive

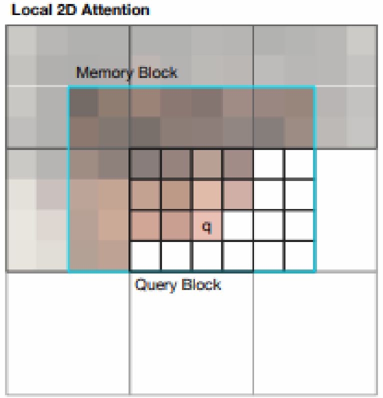
Idea: only compute attention for pixels that are not too far away

(looks a little like PixelCNN)





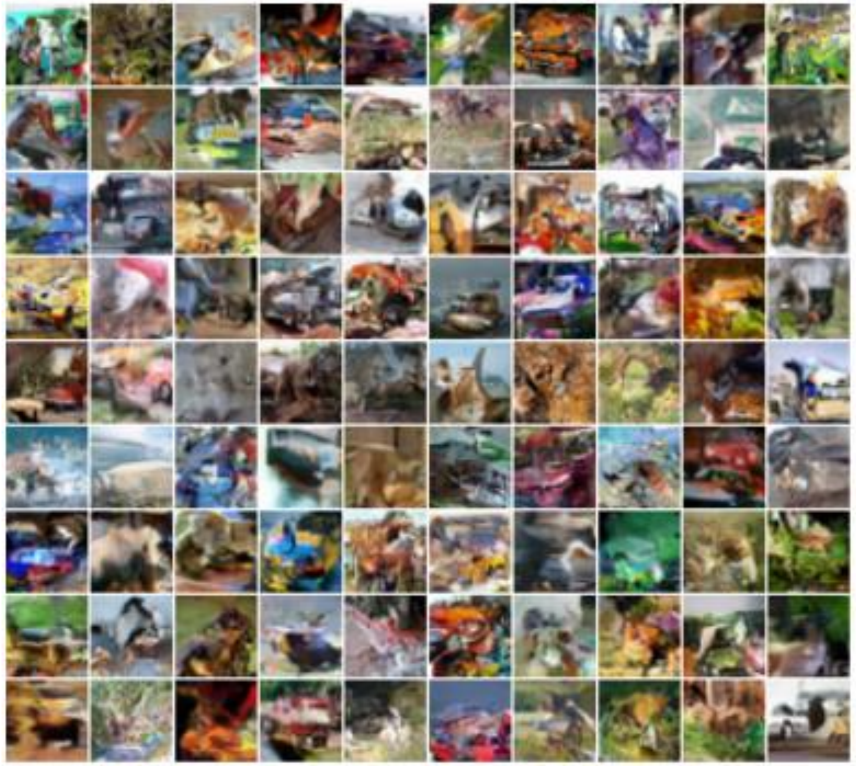




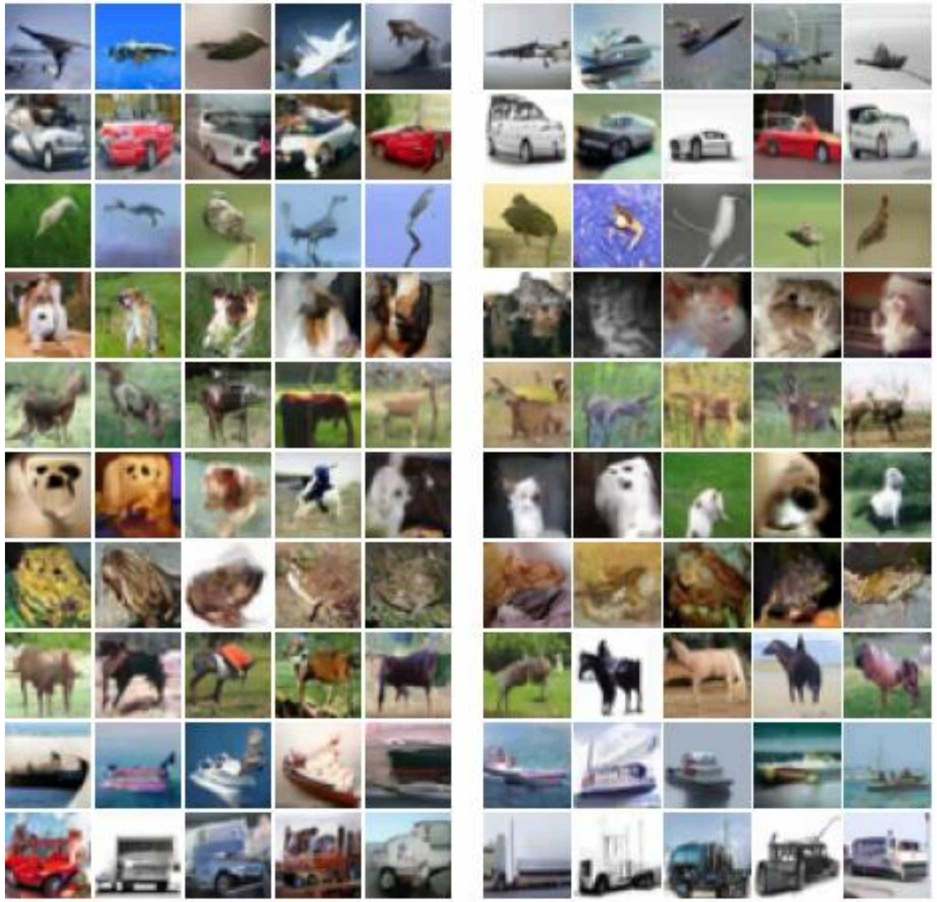
Parmar et al. **Image Transformer**. 2018.

PixelRNN vs. Pixel Transformer

PixelRNN



Transformer



All models trained on CIFAR-10

Parmar et al. **Image Transformer**. 2018.

Conditional autoregressive models

What if we want to generate something **conditioned** on another piece of information?

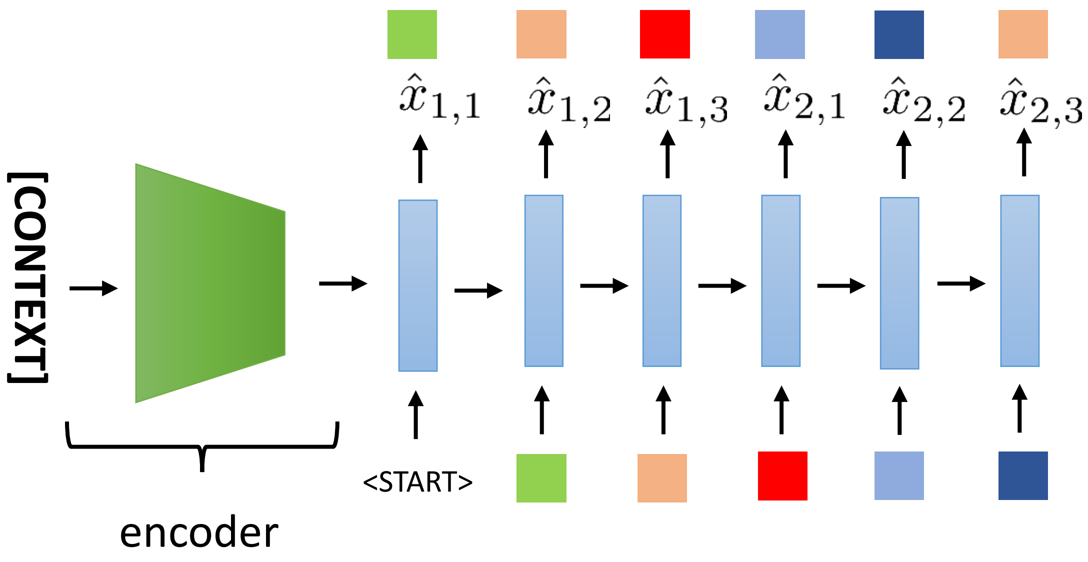
Examples:

- Generate images of specific types of objects (e.g., categories)
- Generate distributions over actions for imitation learning conditioned on the observation
- Many other examples!

Just like conditional language models!

Encoder can be **extremely simple** (e.g., generate images of a class)

Encoder can be **extremely complex** (e.g., multimodal policy in IL)



Conditional autoregressive models

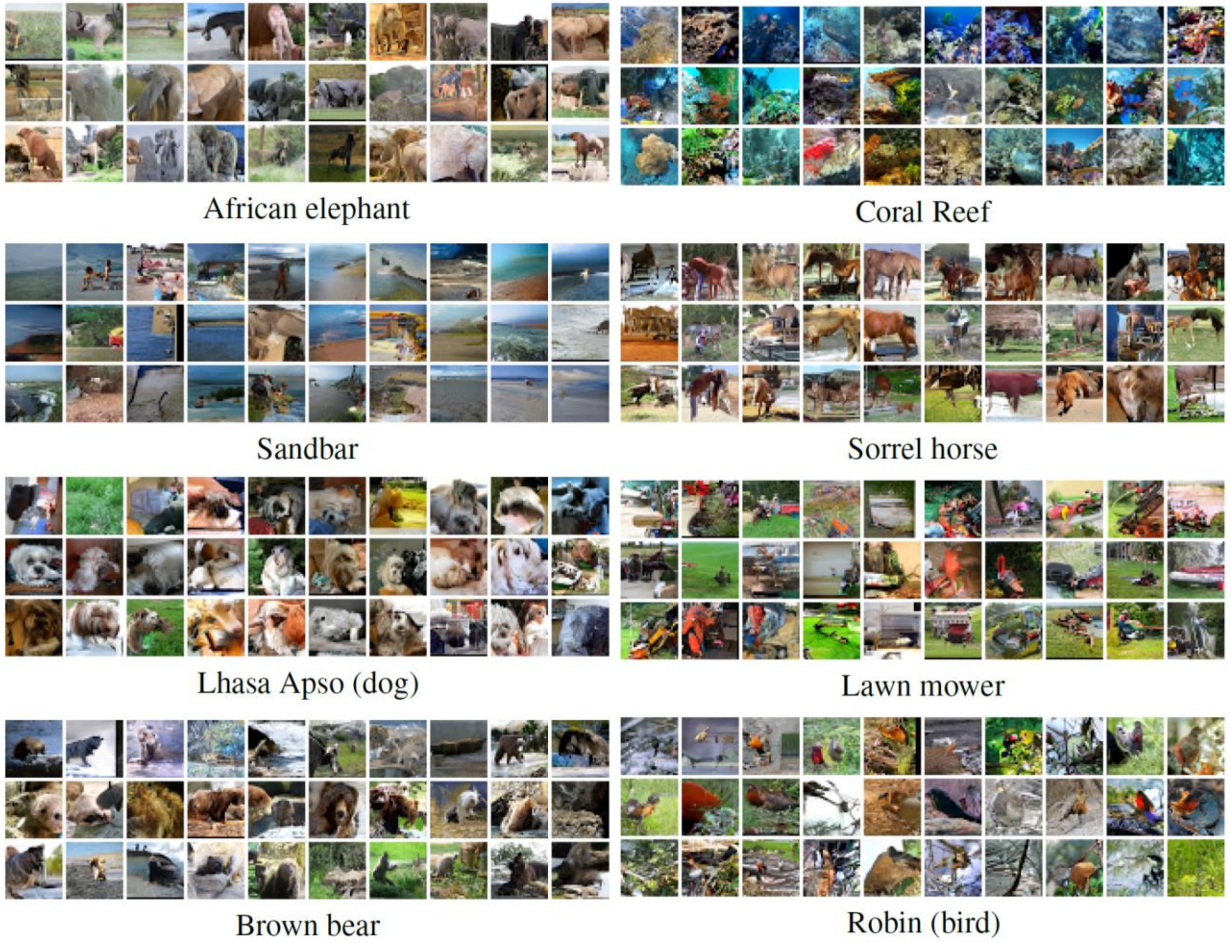


Figure 3: Class-Conditional samples from the Conditional PixelCNN.

Van den Oord et al. **Pixel Recurrent Neural Networks**. 2016.

Tradeoffs and considerations

- Autoregressive generative models are "language models" for other types of data
  - Though more accurate to say that language models are just a special type of autoregressive generative model

- Can represent autoregressive models in many different ways
  - RNNs (e.g., LSTMs)
  - Local context models like PixelCNNs
  - Transformers

- Tradeoffs compared to other models we'll learn about:

  + provide full distribution with probabilities

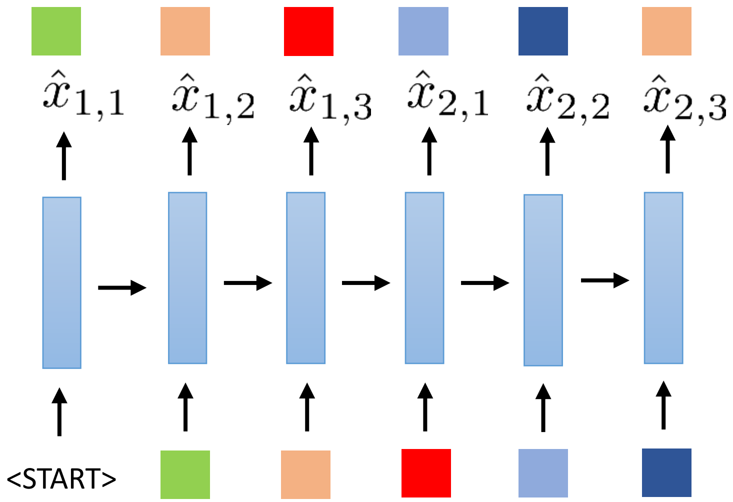
  + conceptually very simple

  - very slow for large datapoints (e.g., images)

  - generally limited in image resolution

Autoencoders

A 30,000 ft view…

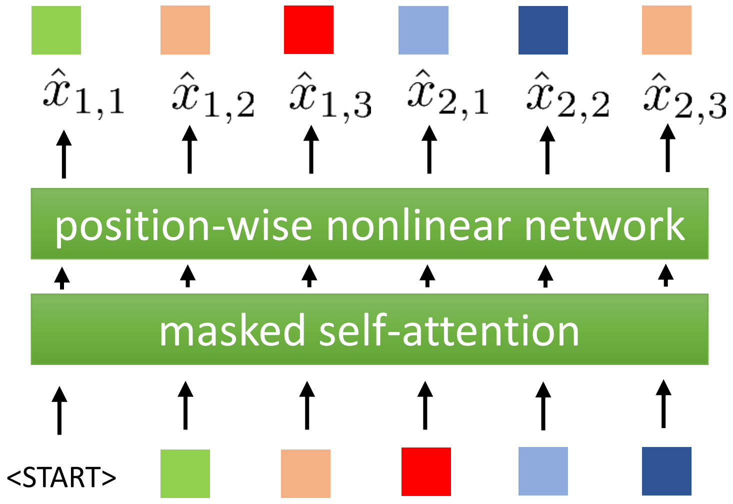


goal is to produce the **same** image

some structure here makes this task of "reconstructing" the input image nontrivial

In this case, it's the fact that each pixel must be constructed from **preceding** pixels

image goes in as input

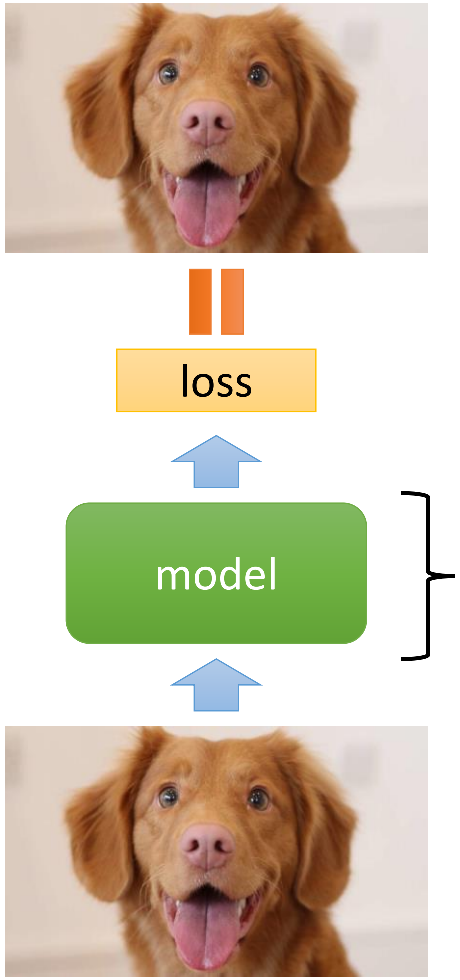


structure comes from **masking** in the self-attention operator (otherwise this would be trivial!)

remember BERT…

A 30,000 ft view…

A general design for generative models?



some structure here makes this task of "reconstructing" the input image nontrivial

i.e., prevents learning an "identity function"

Examples of structure that we've seen:

- RNN/LSTM sequence models that must predict a pixel's value based only on "previous" pixels
- "PixelCNN" models that must predict a pixel's value based on a (masked) neighborhood
- Pixel transformer, which must make predictions based on **masked** self-attention

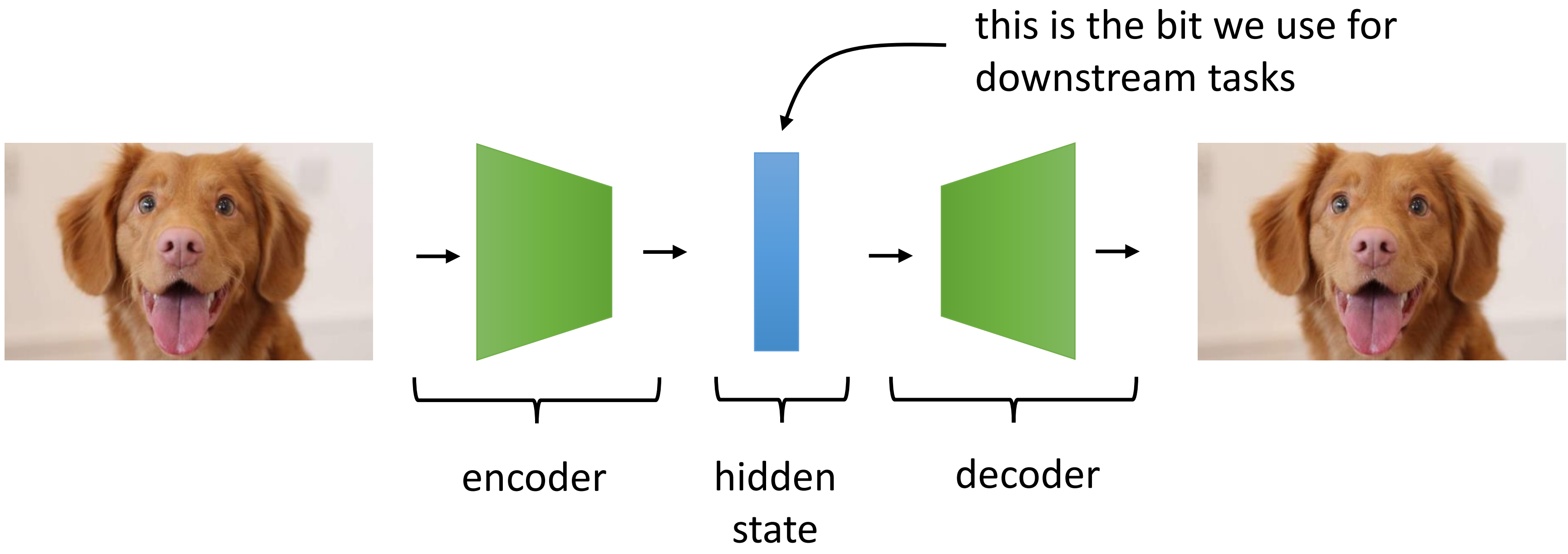
This is all **spatial** structure, can we use more abstract structure instead?

The autoencoder principle

Basic idea: train a network that **encodes** an image into some **hidden state**, and then **decodes** that image **as accurately as possible** from that **hidden state**

Such a network is called an **autoencoder**

Forcing structure: something about the design of the model, or in the data processing or regularization, must force the autoencoder to learn a **structured** representation



The types of autoencoders

Forcing structure: something about the design of the model, or in the data processing or regularization, must force the autoencoder to learn a **structured** representation

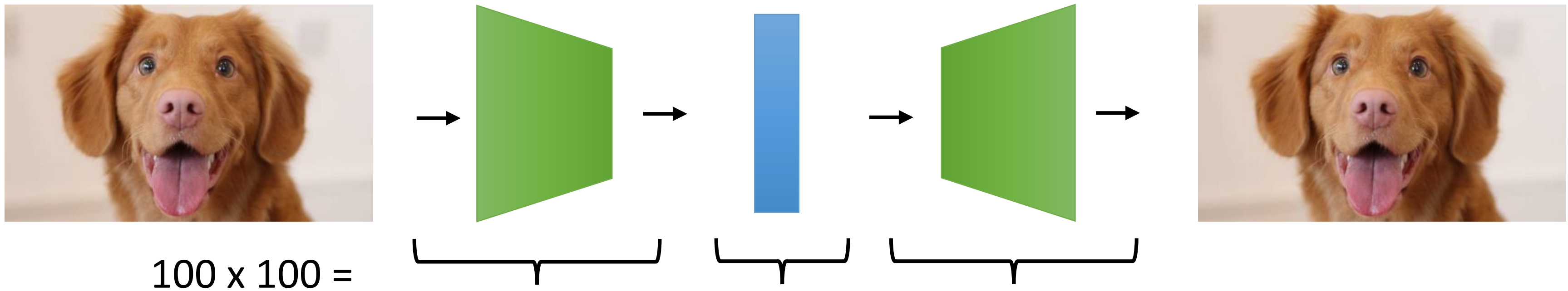
Dimensionality: make the **hidden state** smaller than the **input/output**, so that the network must **compress** it

Sparsity: force the **hidden state** to be sparse (most entries are zero), so that the network must **compress** the input

Denoising: corrupt the **input** with **noise**, forcing the autoencoder to learn to distinguish **noise from signal**

Probabilistic modeling: force the **hidden state** to agree with a **prior distribution** (this will be covered next time)

(Classic) Bottleneck autoencoder

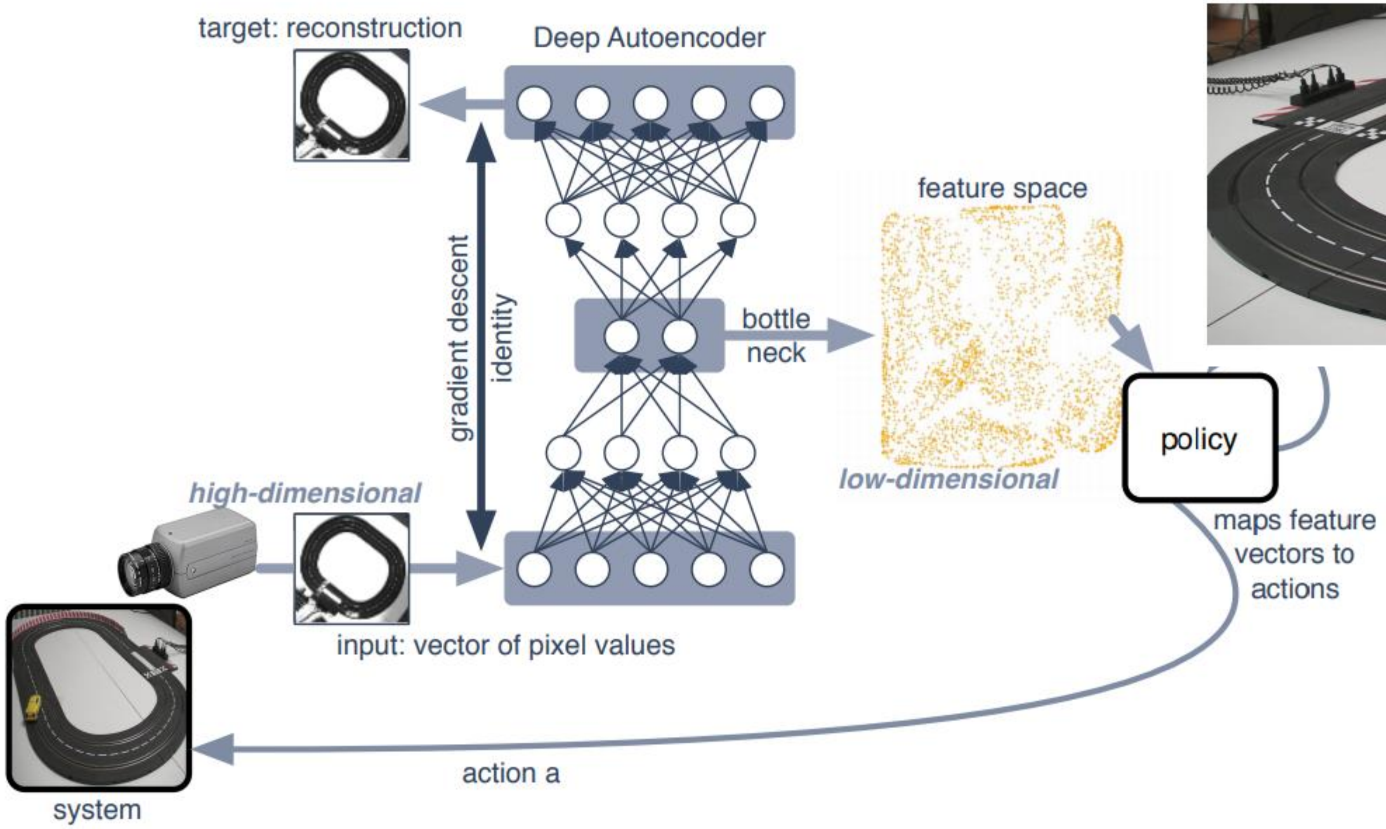


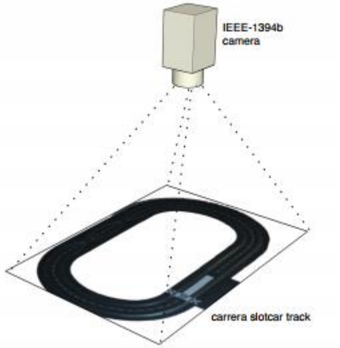
This has some interesting properties:

- If both encoder and decoder are **linear** (which is usually not very interesting), this exactly recovers PCA
- Can be viewed as "non-linear dimensionality reduction" – could be useful simply because dimensionality is lower and we can use various algorithms that are only tractable in low-dimensional spaces (e.g., discretization)

Today, this design is rather antiquated and rarely used, but good to know about historically

Bottleneck autoencoder example





Lange & Riedmiller. **Autonomous reinforcement learning from raw visual data**. 2012.

Sparse autoencoder

Idea: can we describe the input with a small set of “attributes”?

This might be a more **compressed** and **structured** representation

Aside:

This idea originated in neuroscience, where researchers believe that the brain uses **sparse** representations (see “sparse coding”)

Pixel (0,0): #FE057D
Pixel (0,1): #FD0263
Pixel (0,2): #E1065F
⋮

NOT structured

“dense”: most values non-zero

Idea: “sparse” representations are going to be more structured!

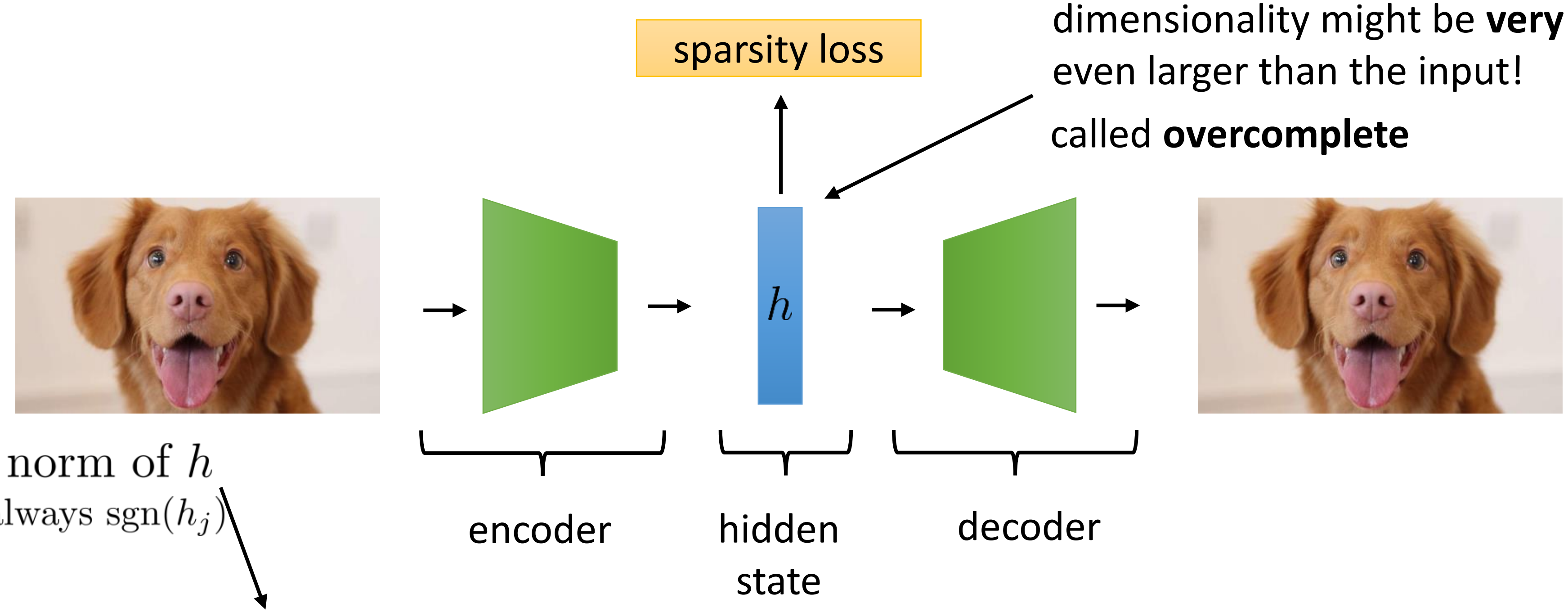
has_ears: 1
has_wings: 0
has_wheels: 0
⋮

very structured!

“sparse”: most values are zero

there are many possible “attributes,” and most images don’t have most of the attributes

Sparse autoencoder



dimensionality might be **very** large, even larger than the input!

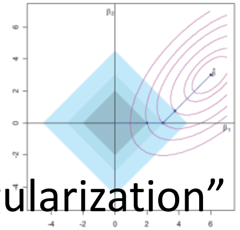
called **overcomplete**

Why?

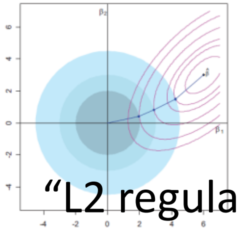
ℓ_1 norm of h

gradient is always $\text{sgn}(h_j)$

simple sparsity loss: $\sum_{j=1}^{D} |h_j|$



"L1 regularization"



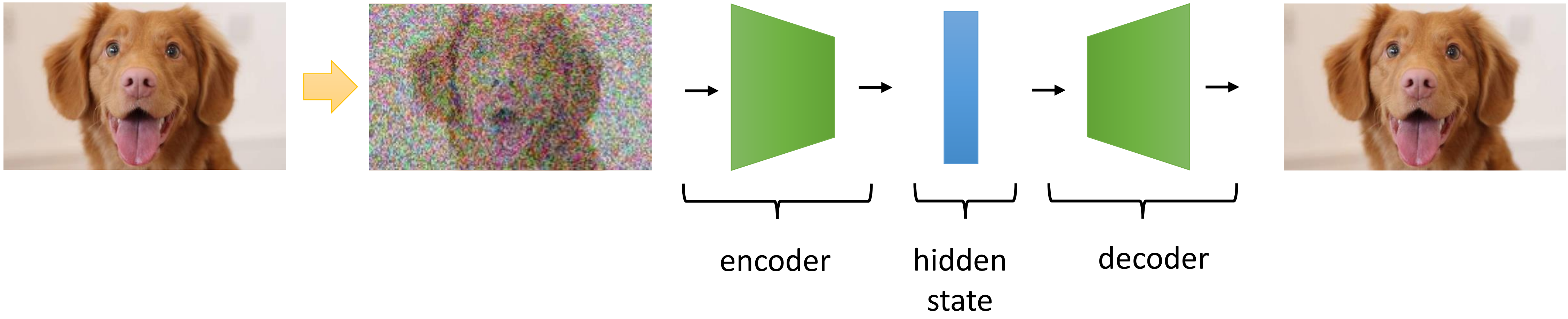
"L2 regularization"

There are other kinds of sparsity losses/models:

- Lifetime sparsity
- Spike and slab models

Denoising autoencoder

Idea: a good model that has learned meaningful structure should "fill in the blanks"



There are **many variants** on this basic idea, and this is one of the most widely used simple autoencoder designs

The types of autoencoders

Forcing structure: something about the design of the model, or in the data processing or regularization, must force the autoencoder to learn a **structured** representation

Dimensionality: make the **hidden state** smaller than the **input/output**, so that the network must **compress** it

+ very simple to implement

- simply reducing dimensionality often does not provide the structure we want

Sparsity: force the **hidden state** to be sparse (most entries are zero), so that the network must **compress** the input

+ principled approach that can provide a "disentangled" representation

- harder in practice, requires choosing the regularizer and adjusting hyperparameters

Denoising: corrupt the **input** with **noise**, forcing the autoencoder to learn to distinguish **noise from signal**

+ very simple to implement

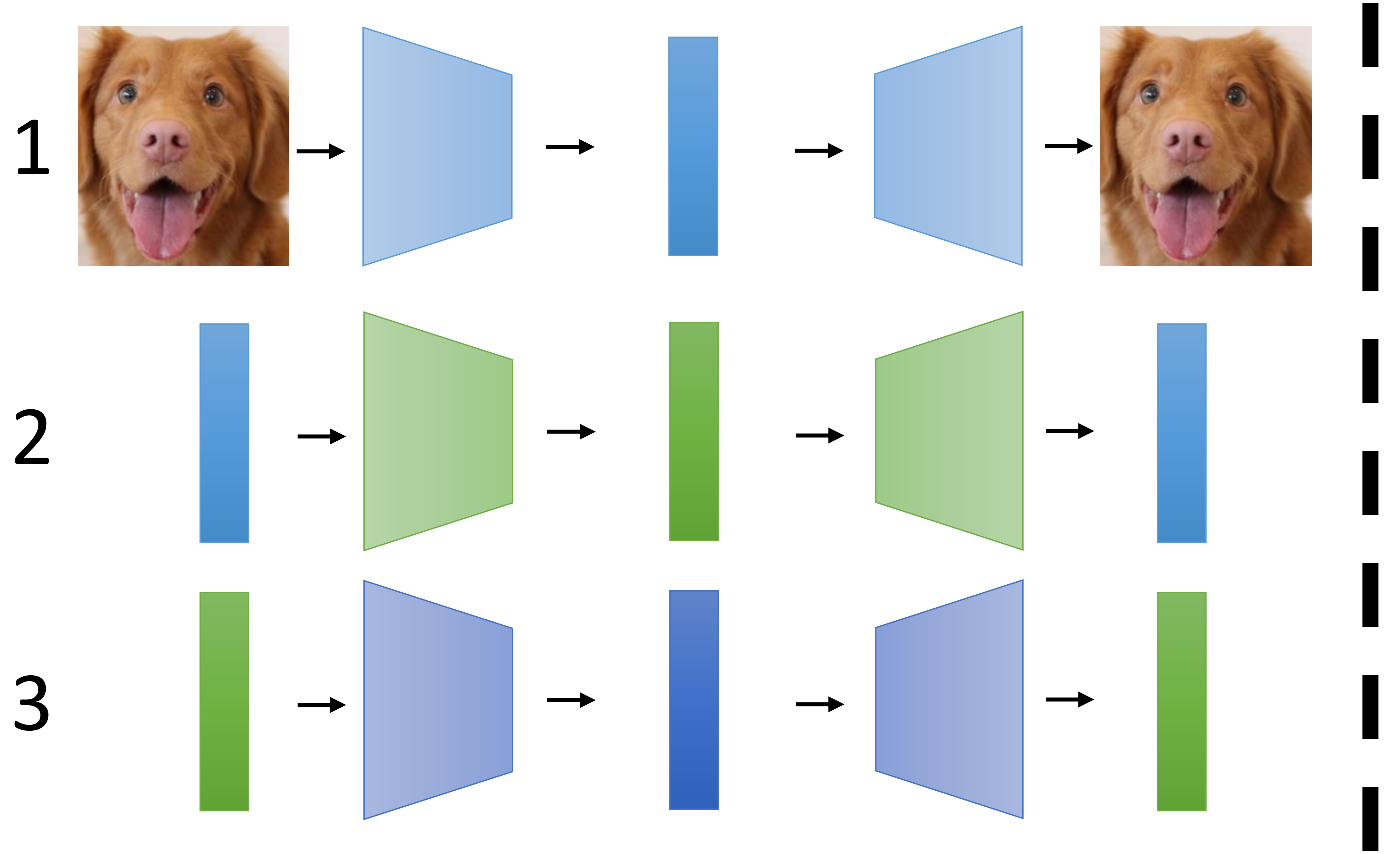
- not clear which layer to choose for the bottleneck, many ad-hoc choices (e.g., how much noise to add)

Probabilistic modeling: force the **hidden state** to agree with a **prior distribution** (this will be covered next time)

We'll discuss this design in much more detail in the next lecture!

Layerwise pretraining

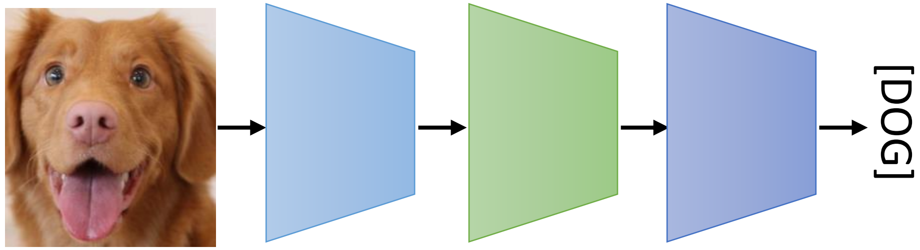
The early days of deep learning…



For a while (2006-2009 or so), this was one of the dominant ways to train **deep** networks

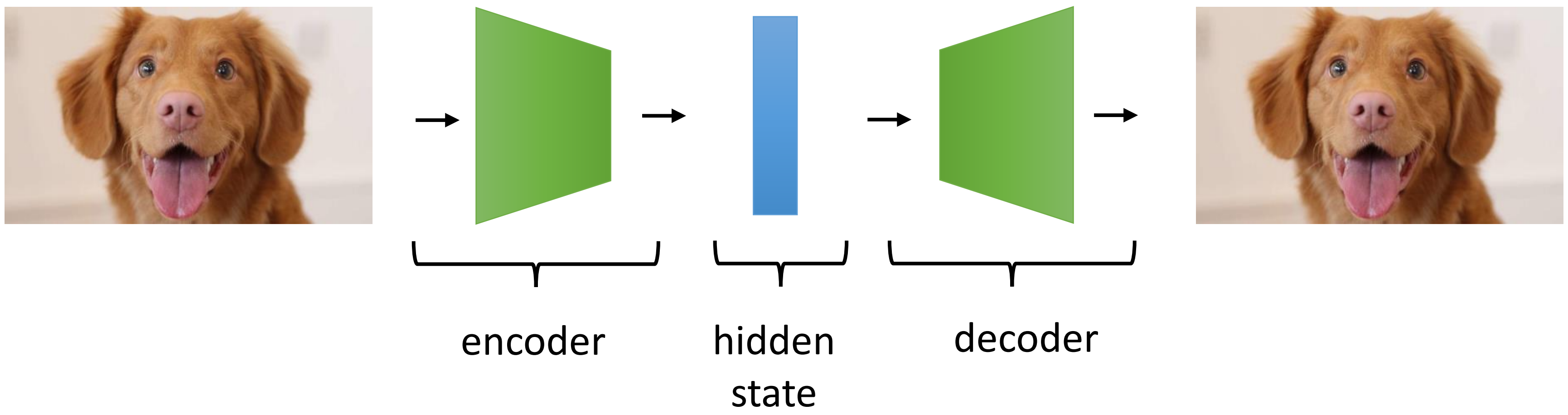
Then we got a lot better at training deep networks end-to-end (ReLU, batch norm, better hyperparameter tuning), and largely stopped doing this

Correspondingly, autoencoders became less important, but they are still useful!



Autoencoders today

- Much **less** widely used these days because there are better alternatives
  - **Representation learning:** VAEs, contrastive learning
  - **Generation:** GANs, VAEs, autoregressive models
- Still a viable option for "quick and dirty" representation learning that is very fast and can work OK
- **Big problem:** sampling (generation) from an autoencoder is hard, which limits its uses
  - The variational autoencoder (VAE) addresses this, and is the most widely used autoencoder today – we will cover this next time!

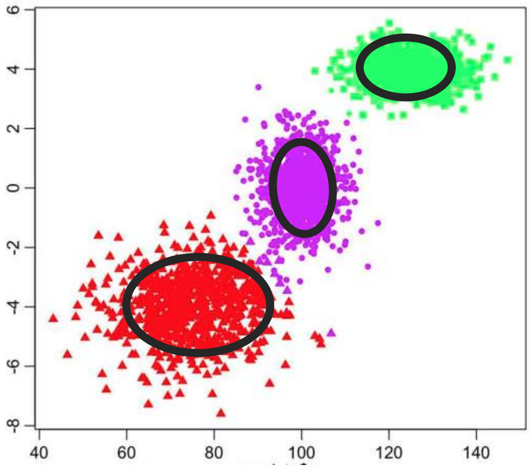


Latent Variable Models

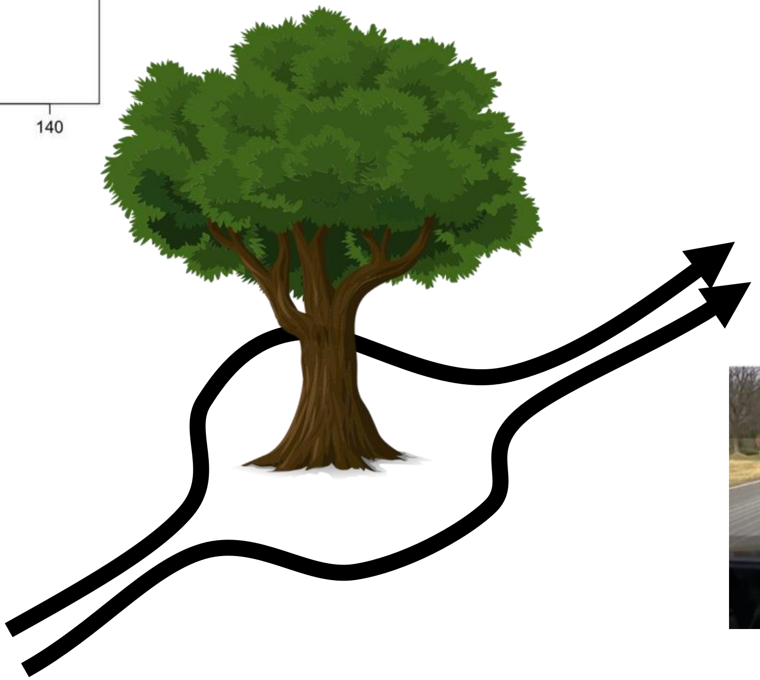
Latent variable models

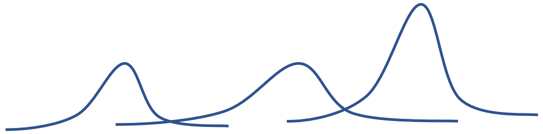
$$p(x) = \sum_z p(x|z)p(z)$$

mixture element

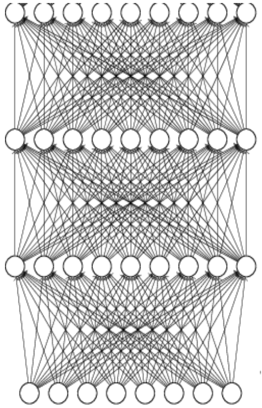


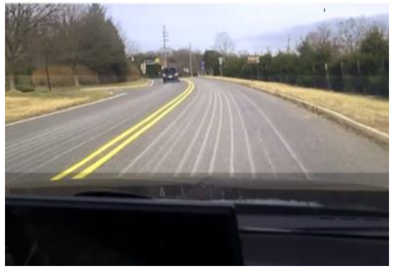
$$p(y|x) = \sum_z p(y|x,z)p(z)$$





$$w_1, \mu_1, \Sigma_1, \ldots, w_N, \mu_N, \sigma_N$$



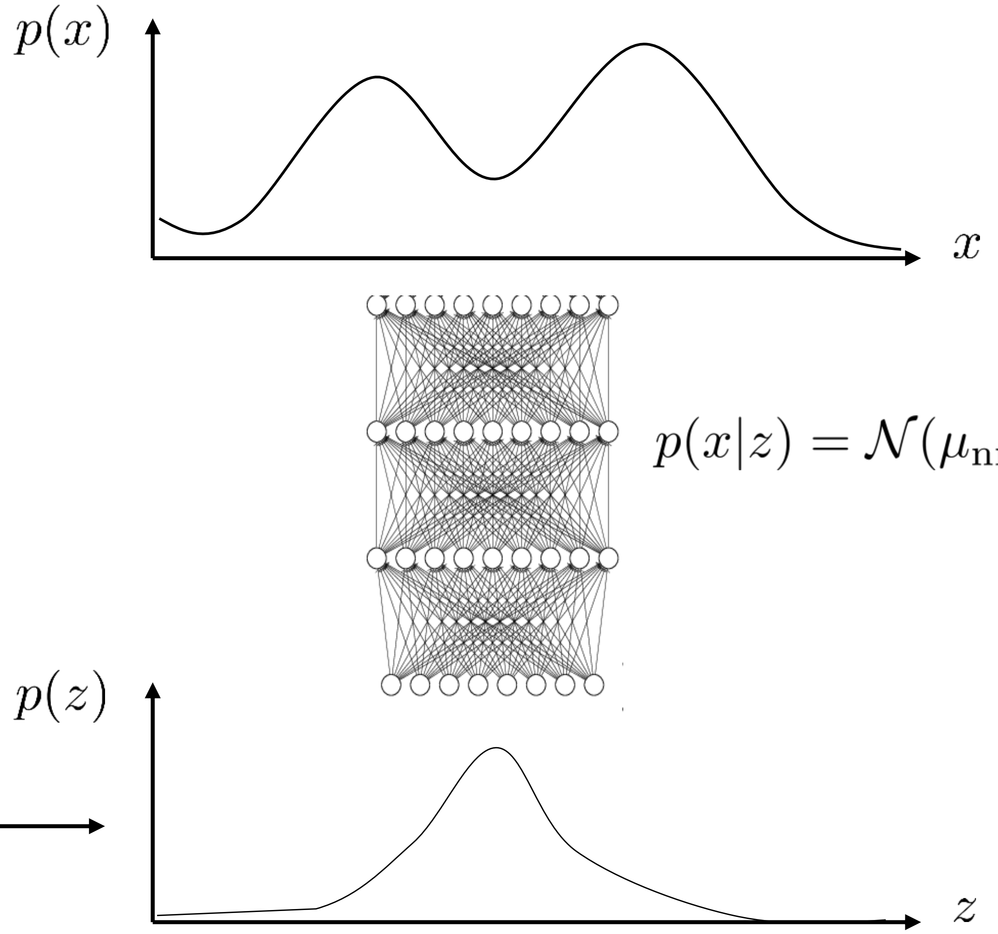


Latent variable models in general

$$p(x) = \int p(x|z)p(z)dz$$

"easy" distribution
(e.g., conditional Gaussian)

"easy" distribution
(e.g., Gaussian)



$$p(x|z) = \mathcal{N}(\mu_{\mathrm{nn}}(z), \sigma_{\mathrm{nn}}(z))$$

"easy" distribution
(e.g., Gaussian)

How do we train latent variable models?

the model: $p_\theta(x)$

the data: $\mathcal{D} = \{x_1, x_2, x_3, \ldots, x_N\}$

maximum likelihood fit:

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \log p_\theta(x_i) \qquad\qquad p(x) = \int p(x|z)p(z)dz$$

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \log \left(\int p_\theta(x_i|z)p(z)dz \right)$$

completely intractable

Estimating the log-likelihood

alternative: *expected* log-likelihood:

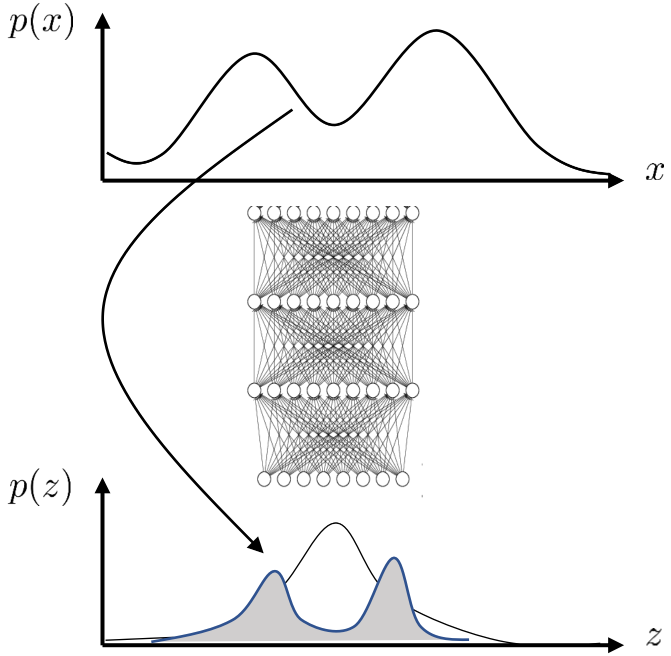
$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i E_{z \sim p(z|x_i)}[\log p_\theta(x_i, z)]$$

but... how do we calculate $p(z|x_i)$?

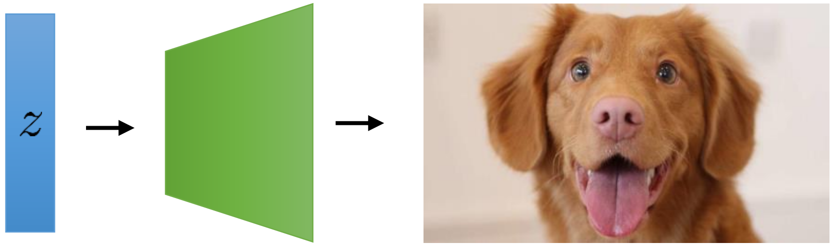
this is called **probabilistic inference**

intuition: "guess" most likely z given x_i, and pretend it's the right one

...but there are many possible values of z so use the distribution $p(z|x_i)$



Latent variable models in deep learning



$p(z)$ $\quad p_\theta(x|z)$

$\mathcal{N}(0, \mathbf{I})$

A latent variable deep generative model is (usually) just a model that turns random numbers into valid samples (e.g., images)

Please don't tell anyone I said this, it destroys the mystique

There are many types of such models: VAEs, GANs, normalizing flows, etc.

Using the model for **generation**:

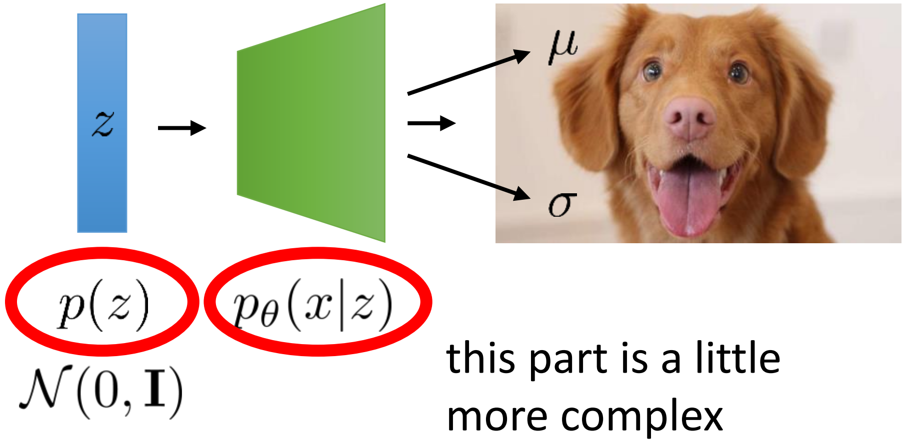
1. sample $z \sim p(z)$ "generate a vector of random numbers"

2. sample $x \sim p(x|z)$ "turn that vector of random numbers into an image"

Today: how do we represent and use this

Next time: how do we train this

Representing latent variable models



this part is a little more complex

this part is easy, just generate (e.g.) Gaussian random numbers

How do we represent the distribution over x?

Option 1: Pixels are continuous-valued

$$p_\theta(x|z) = \mathcal{N}(\mu_\theta(z); \sigma_\theta(z))$$

mean is a neural net function

variance is (optionally) a neural net function

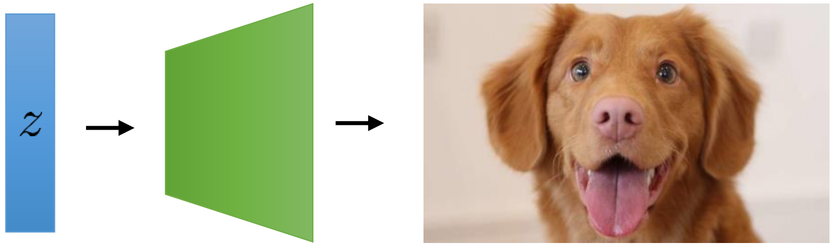
easy choice: let σ just be a constant

either a learned constant (independent of z)

or chosen manually (e.g., 1)

This just reduces to MSE loss!

Representing latent variable models



$p(z)$ $\quad p_\theta(x|z)$

$\mathcal{N}(0, \mathbf{I})$

How do we represent the distribution over x?

Option 2: Pixels are discrete-valued

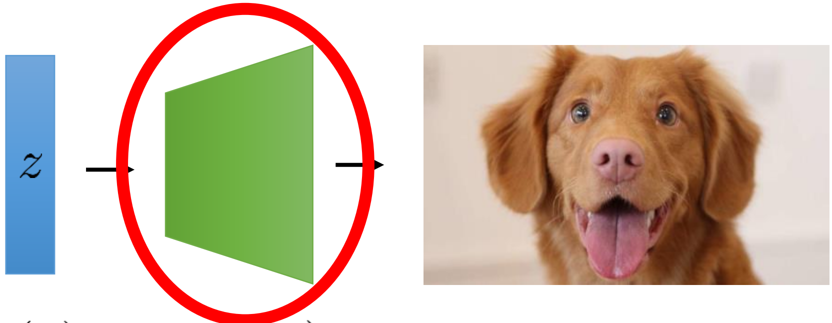
Could just a 256-way softmax, just like in PixelRNN or PixelCNN!

(this works very well, but is a little bit slow)

Other choices (not covered in this lecture): discretized logistic, binary cross-entropy

especially common for best performing models

Representing latent variable models



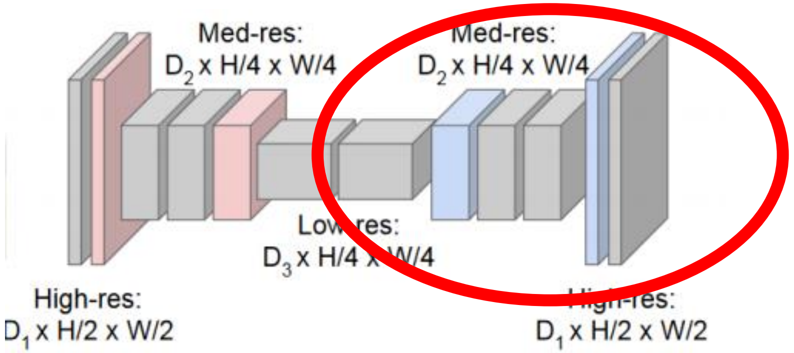
$p(z)$ $p_\theta(x|z)$ what architecture should we use?

$\mathcal{N}(0, \mathbf{I})$

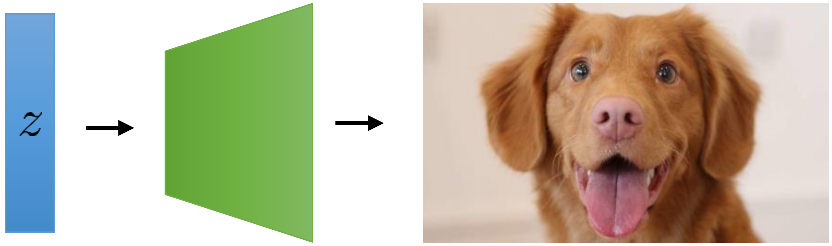
Easy choice: just a big fully connected network (linear layers + ReLU)

works well for tiny images (e.g., MNIST) or non-image data

Better choice: transpose convolutions



Training latent variable models



$p(z)$ $\quad p_\theta(x|z)$

$\mathcal{N}(0, \mathbf{I})$

$$1\text{a. sample } z \sim p(z|x_i)$$
$$1\text{b. reduce } -\log p(x_i|z) \text{ with SGD}$$

variational autoencoders (VAEs)

Three basic choices:

1. Perform inference to figure out $p(z|x_i)$ for each training image x_i

   Then minimize *expected* NLL $E_{p(z|x_i)}[-\log p(x_i|z)]$

2. Use an *invertible* mapping z to x **normalizing flows**

3. Match the distribution $E_{z\sim p(z)}[p_\theta(x|z)]$ to data distribution

generative adversarial networks (GANs)

UC Berkeley · CSW182 | [Deep Learning]
Designing, Visualizing and Understanding Deep Neural Networks (2021)

CSW182 (2021)· 课程资料包 @ShowMeAI

视频
中英双语字幕

课件
一键打包下载

笔记
官方笔记翻译

代码
作业项目解析

视频 · B 站 [扫码或点击链接]
https://www.bilibili.com/video/BV1Ff4y1n7ar

课件 & 代码 · 博客 [扫码或点击链接]
http://blog.showmeai.tech/berkeley-csw182

Berkeley　循环神经网络　可视化　梯度策略
Q-Learning　风格迁移　模仿学习　元学习
计算机视觉　机器学习基础　生成模型　卷积网络

Awesome AI Courses Notes Cheatsheets 是 ShowMeAI 资料库的分支系列，覆盖最具知名度的 **TOP50+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets· 持续更新中			
知识图谱	**图机器学习**	**深度强化学习**	**自动驾驶**
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094

微信公众号

资料下载方式 2：扫码点击底部菜单栏

称为 **AI 内容创作者？** 回复 [添砖加瓦]