Lecture 12: Unsupervised and Reinforcement Learning

Announcements

- A1, A2, and project proposal grades are released on Gradescope
- A3 due Tue 11/15
- Project milestone due Fri 11/18
- Remember: 6 late days provided in the class, you can use up to 4 per assignment / project milestone
- Extra credit opportunity: +0.25% on final class grade for attending upcoming guest lecture live (applied post-curve, does not affect curve)
    - Mon 11/14 in-person, Dr. Barbara Engelhardt, Genomics: Advanced Topics

Supervised learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, semantic segmentation, object detection, instance segmentation

Right effusion

Classification

Now: Unsupervised learning

Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, representation / feature learning, density estimation, etc.

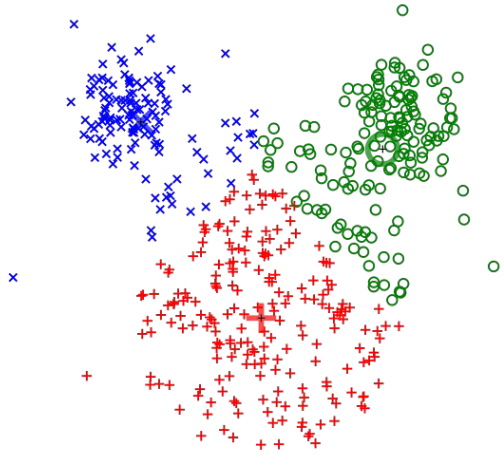
Now: Unsupervised learning

Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, representation / feature learning, density estimation, etc.



K-means clustering

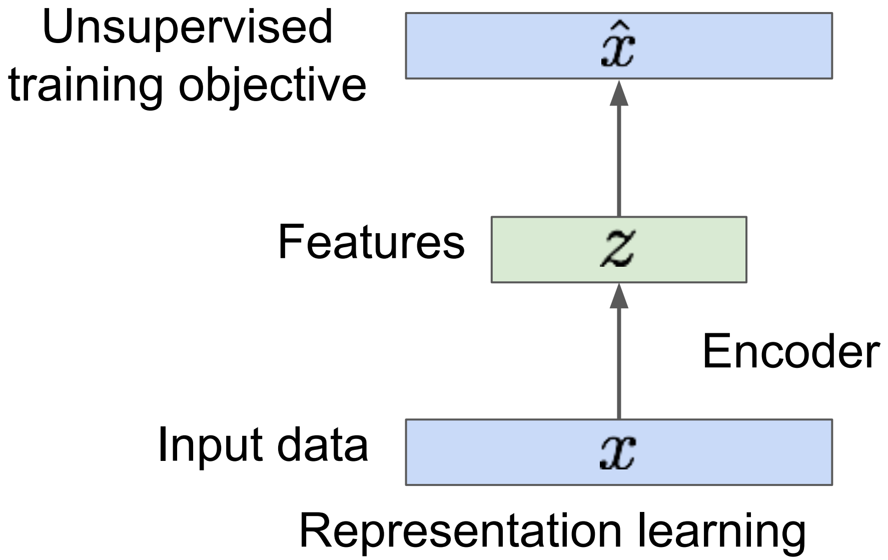
Now: Unsupervised learning

Data: x

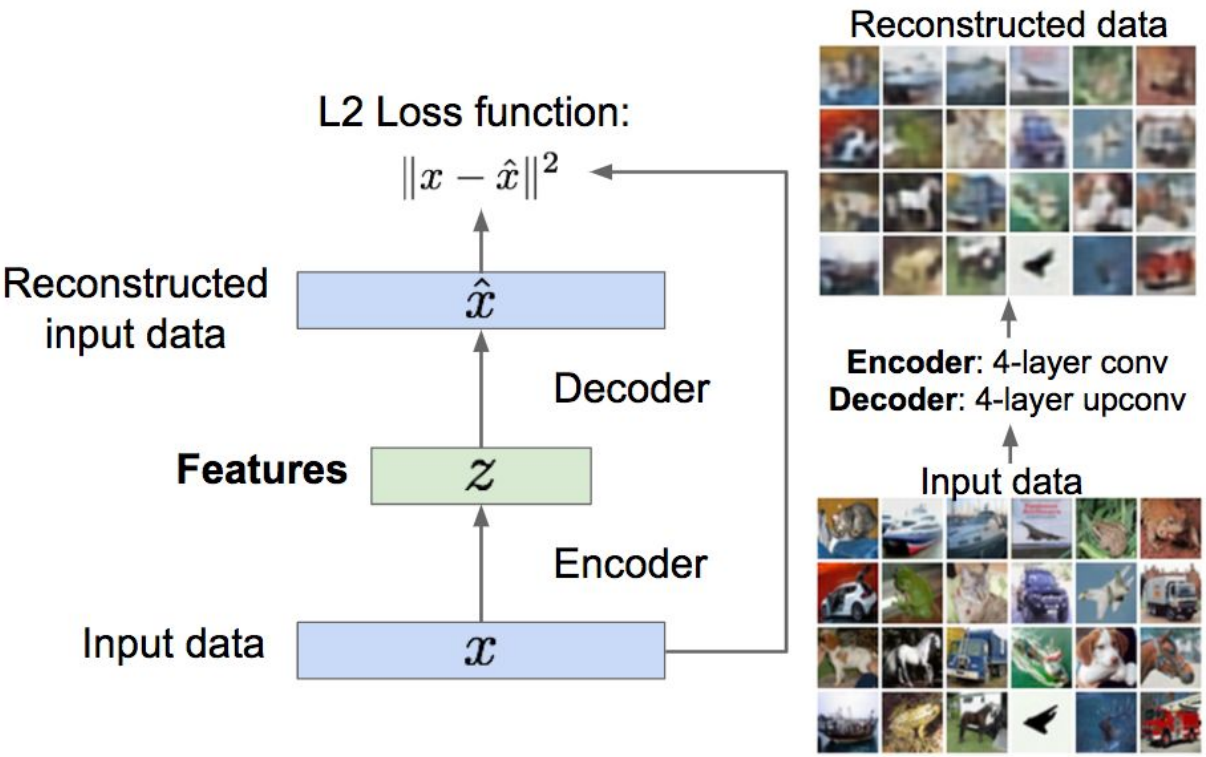
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, representation / feature learning, density estimation, etc.

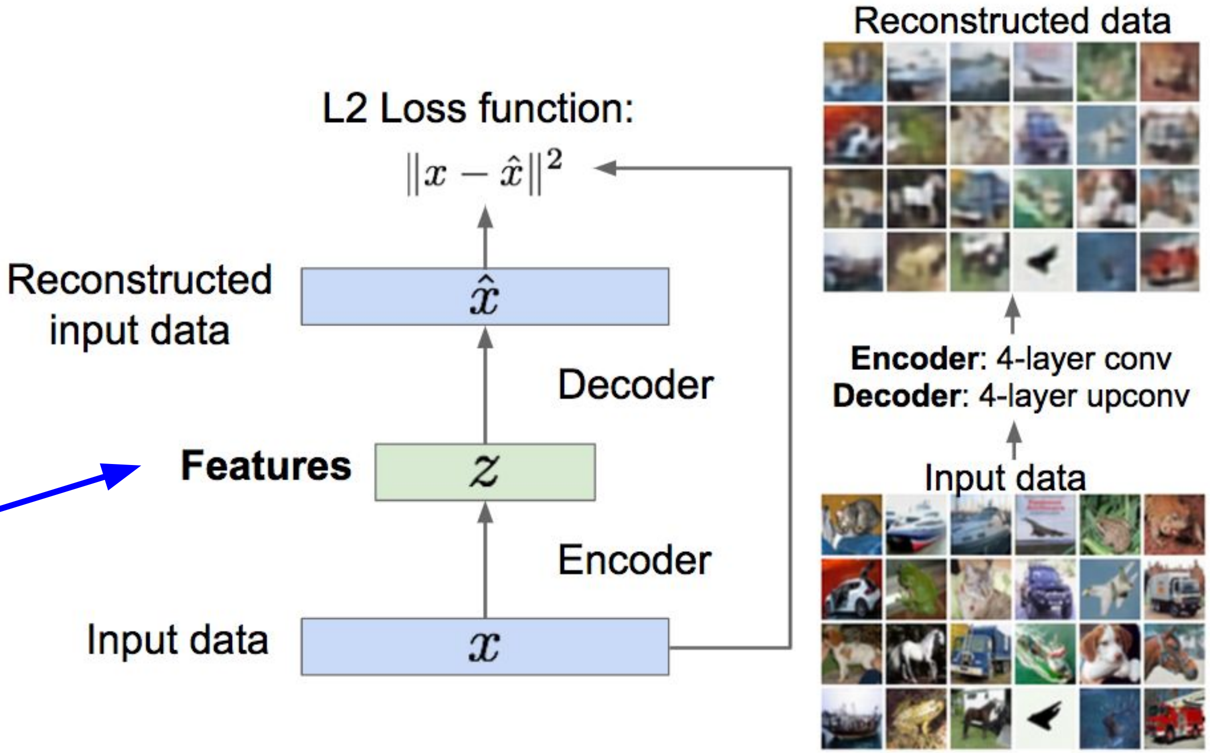


Unsupervised representation learning: autoencoders



Autoencoders

Unsupervised representation learning: autoencoders

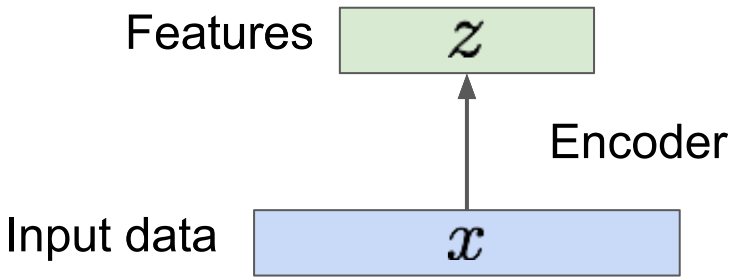


(Feature representation)

Autoencoders

Representation learning: autoencoders

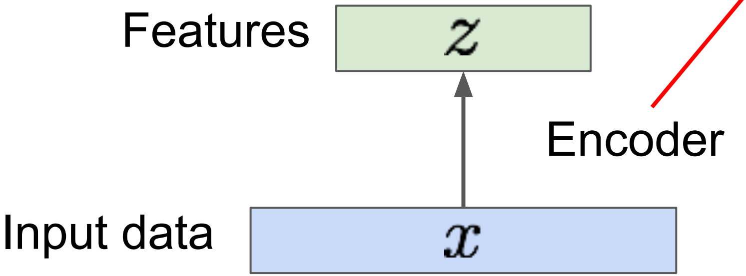
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

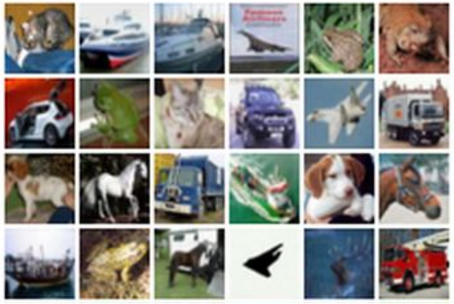


Representation learning: autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





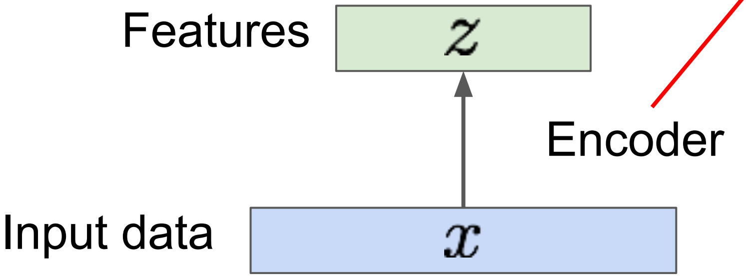
Representation learning: autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN



Representation learning: autoencoders

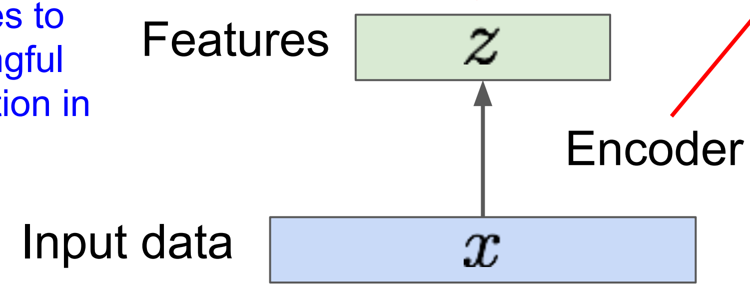
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

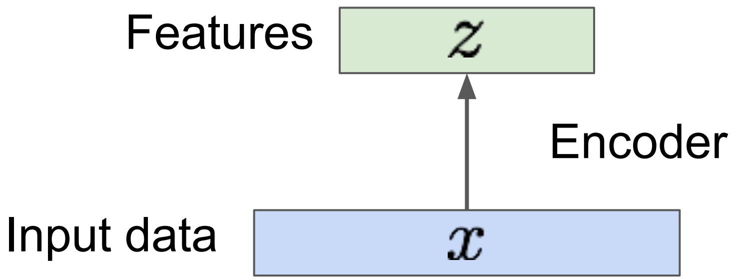
A: Want features to capture meaningful factors of variation in data

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN



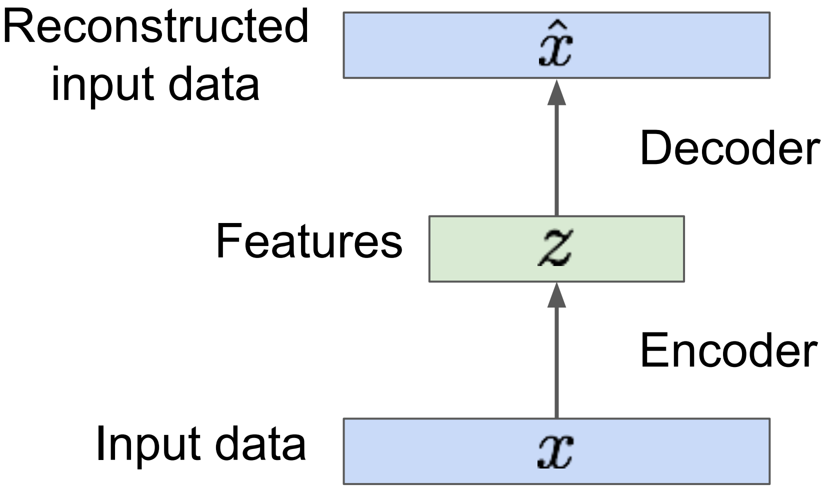
Representation learning: autoencoders

How to learn this feature representation?



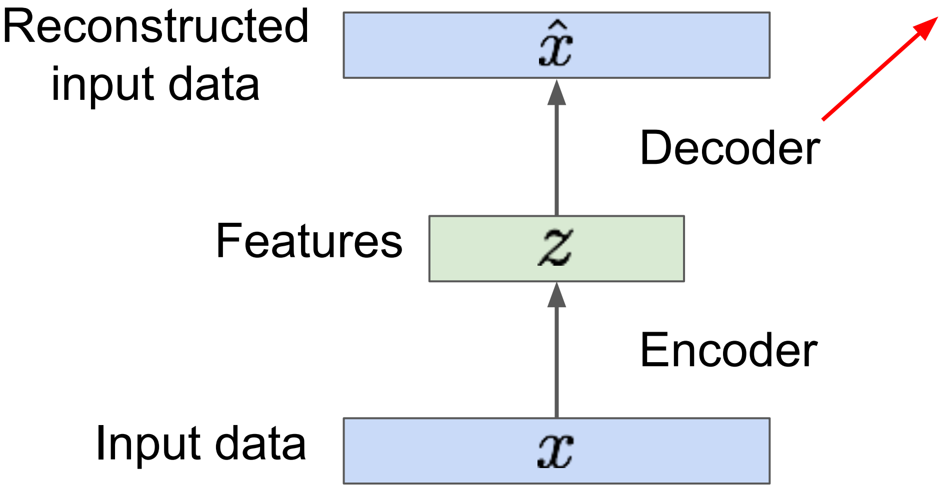
Representation learning: autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself



Representation learning: autoencoders

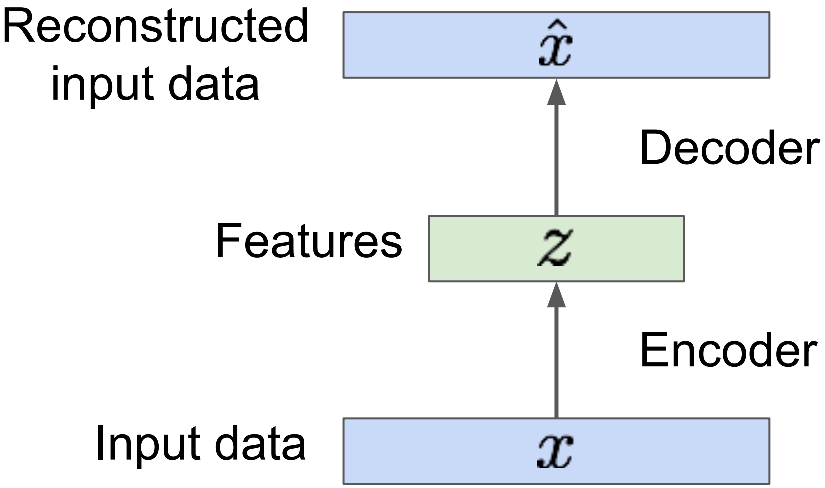
How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

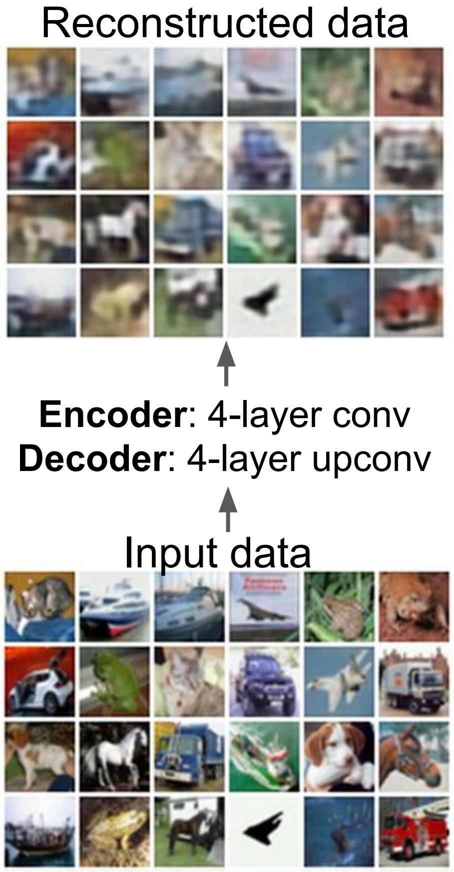


Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)

Representation learning: autoencoders

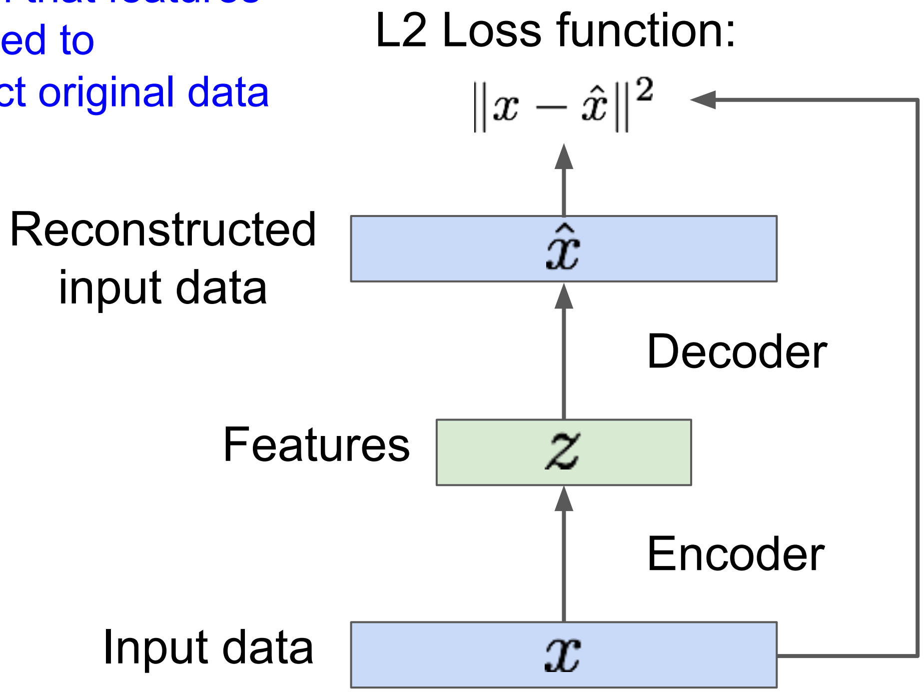
How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

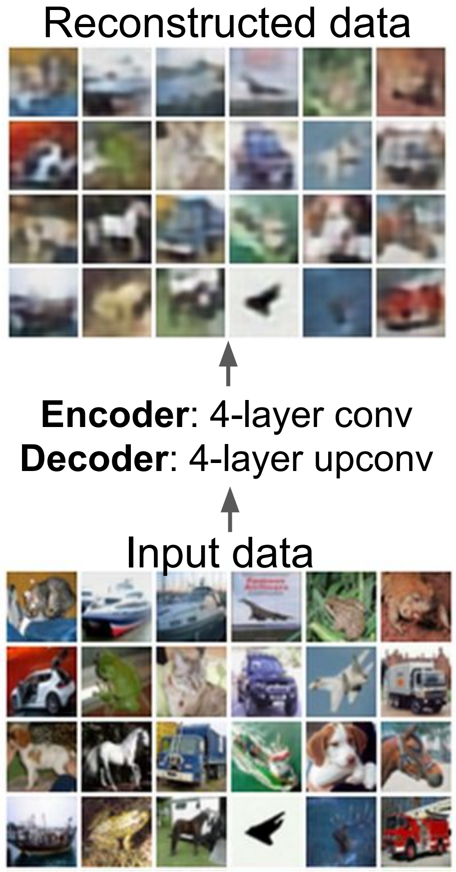




Representation learning: autoencoders

Train such that features can be used to reconstruct original data

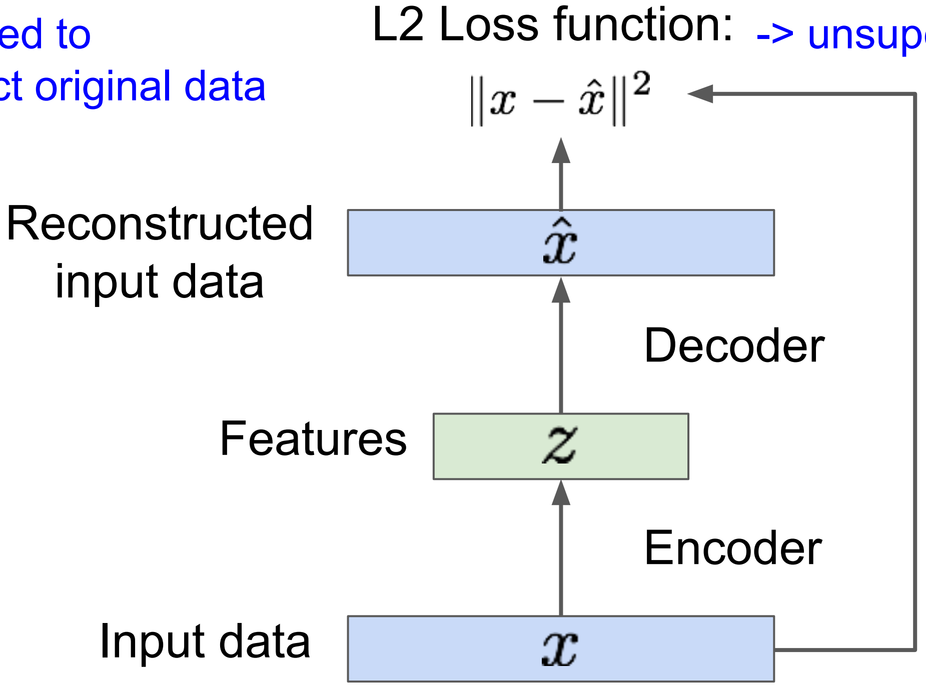


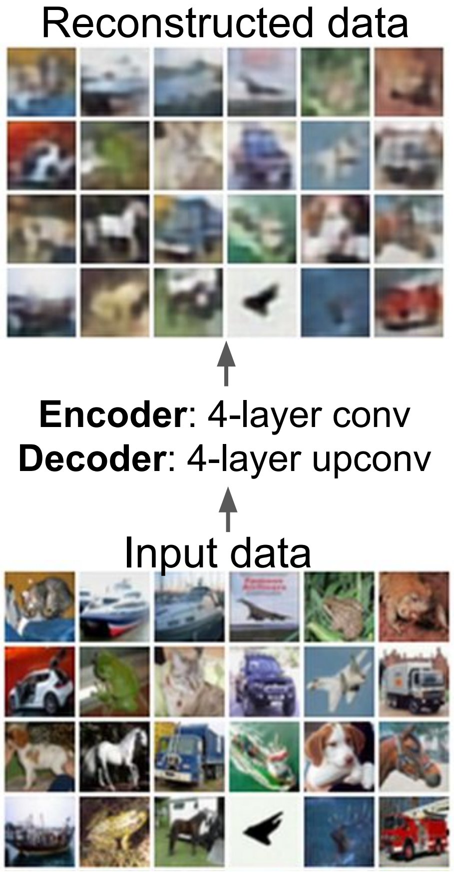


Representation learning: autoencoders

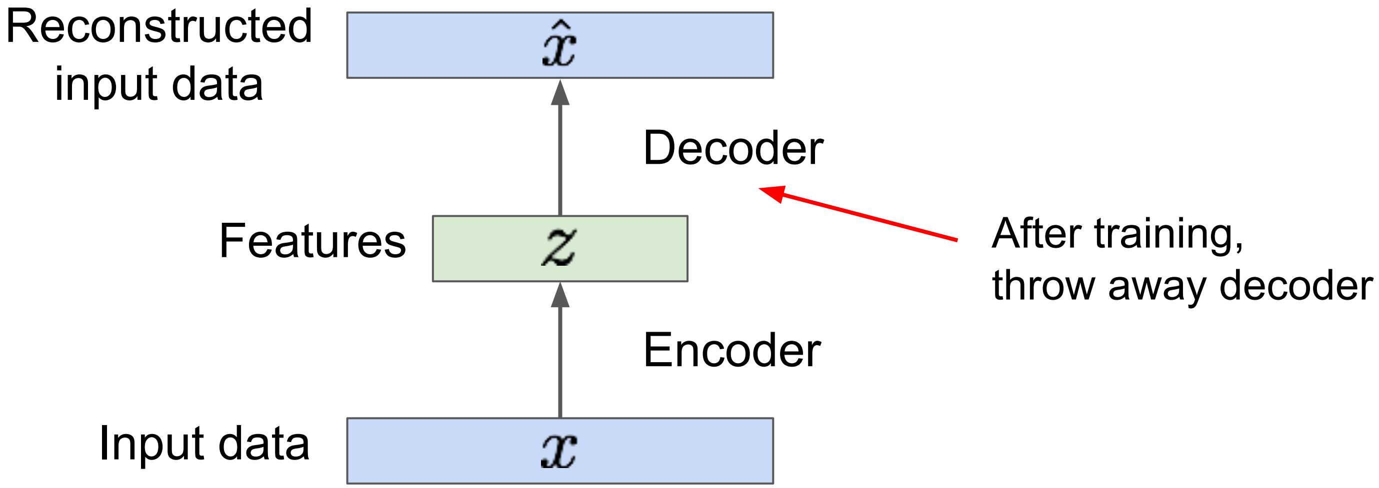
Train such that features can be used to reconstruct original data

Doesn't use labels! -> unsupervised



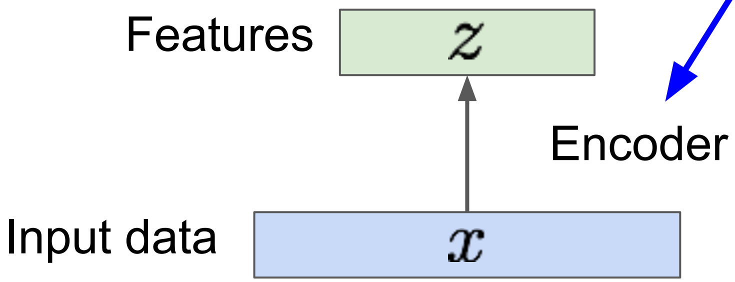


Representation learning: autoencoders

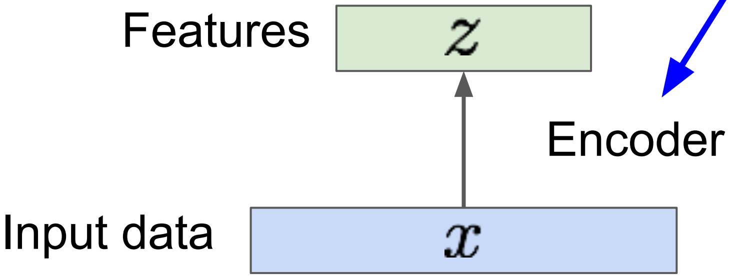


Representation learning: autoencoders

Encoder network can now be used as a feature extractor! Should be semantically meaningful features due to autoencoder loss from training.



Representation learning: autoencoders

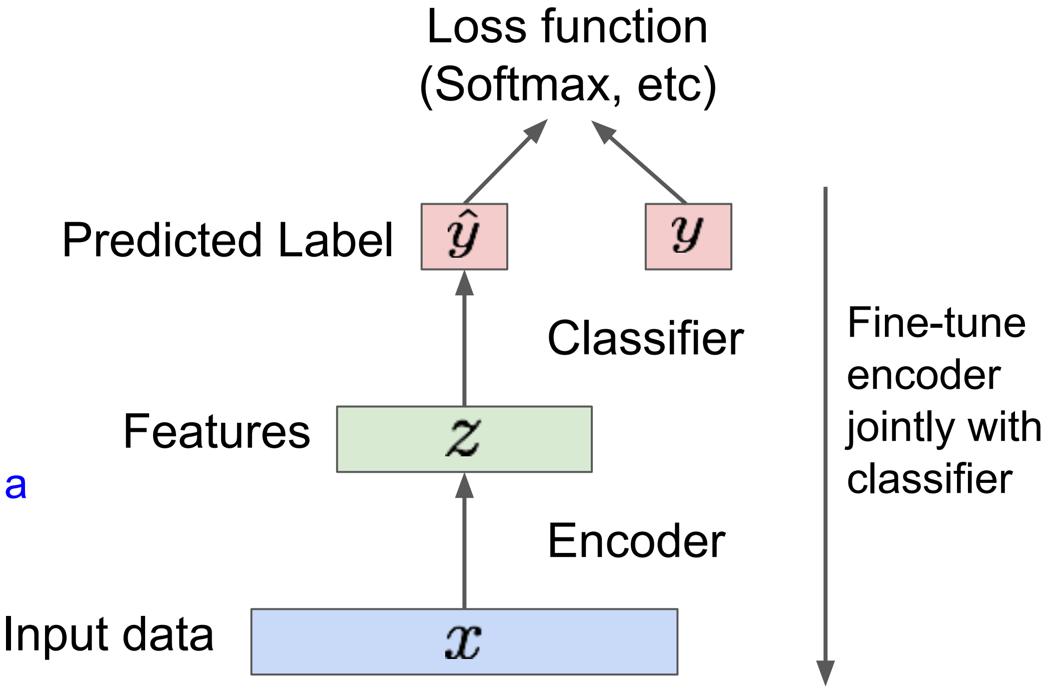


Encoder network can now be used as a feature extractor! Should be semantically meaningful features due to autoencoder loss from training.

Features can be used for clustering, retrieval (e.g. find the closest patient to this one), etc.

Representation learning: autoencoders

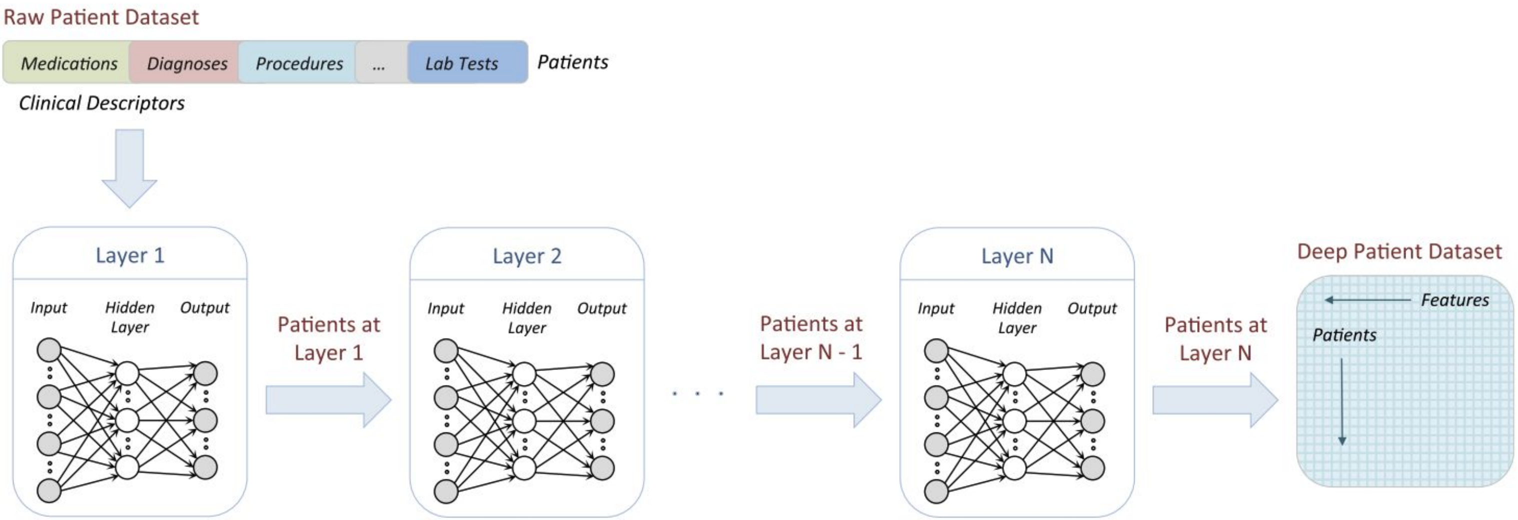
In supervised learning tasks, an encoder trained in an unsupervised way (potentially on larger amounts of data) can also be used as a feature extractor for the task, or to initialize a supervised model



Miotto 2016

- Used stack of denoising autoencoders (add noise to inputs to avoid overfitting) to learn feature representation from EHR data of 700,000 patients from Mount Sinai

- Used learned feature representation for downstream disease classification tasks

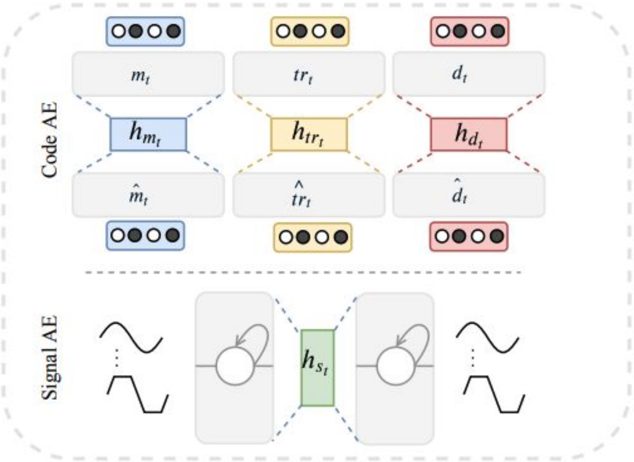


Miotti et al. Deep Patient: An Unsupervised Representation to Predict the Future of Patients from the Electronic Health Records, 2016.

Darabi 2019

- Autoencoder-based unsupervised representation learning for **multimodal data** of 200,000 records from 250 hospital sites (eICU collaborative Research Database)

- Used feature representation to train models for downstream mortality, readmission prediction tasks

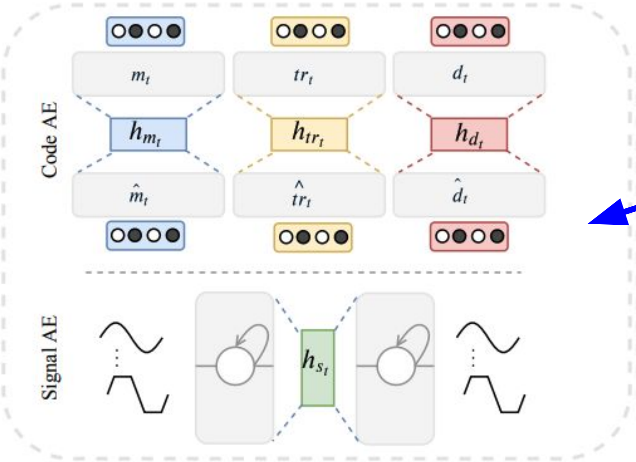


Darabi et al. Unsupervised Representation for EHR Signals and Codes as Patient Status Vector, 2019.

Darabi 2019

- Autoencoder-based unsupervised representation learning for **multimodal data** of 200,000 records from 250 hospital sites (eICU collaborative Research Database)

- Used feature representation to train models for downstream mortality, readmission prediction tasks



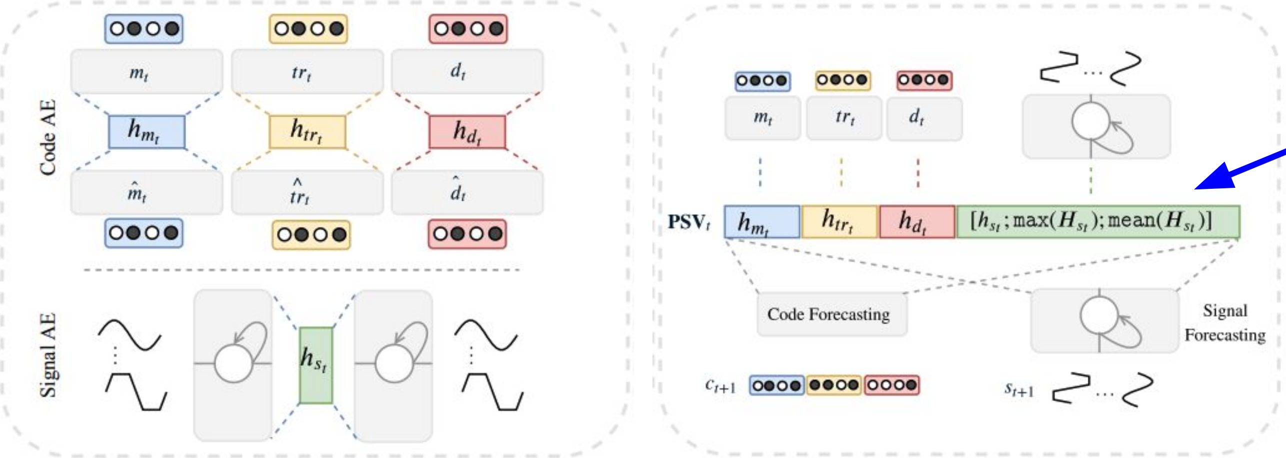
Autoencoder for each code-based modality (e.g. medication, treatment, diagnosis), and signal time-series (e.g. heart rate)

Darabi et al. Unsupervised Representation for EHR Signals and Codes as Patient Status Vector, 2019.

Darabi 2019

- Autoencoder-based unsupervised representation learning for **multimodal data** of 200,000 records from 250 hospital sites (eICU collaborative Research Database)

- Used feature representation to train models for downstream mortality, readmission prediction tasks

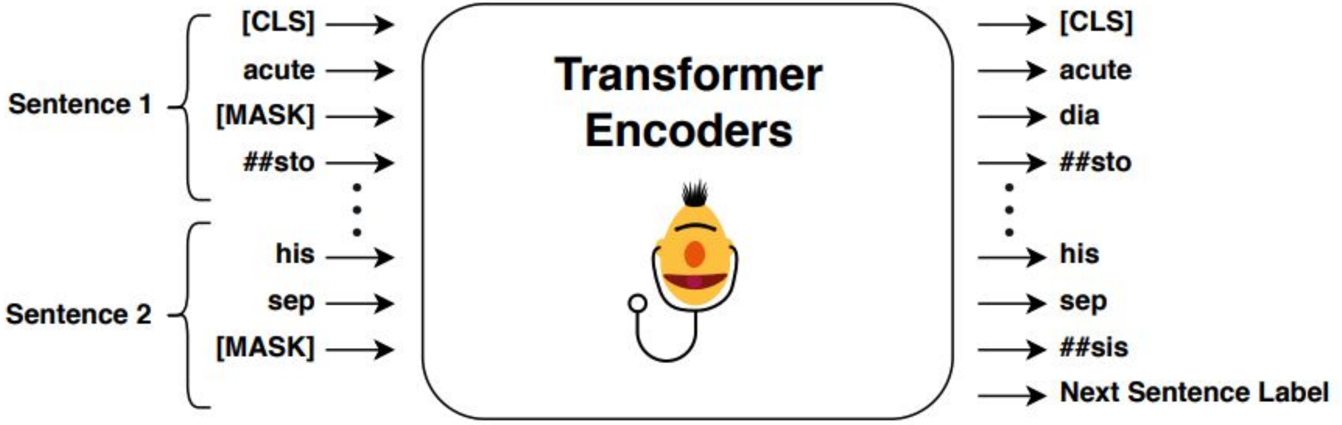


Concatenate feature representations from each autoencoder, and further fine-tune on predicting future elements in data

Darabi et al. Unsupervised Representation for EHR Signals and Codes as Patient Status Vector, 2019.

Aside: self-supervised learning

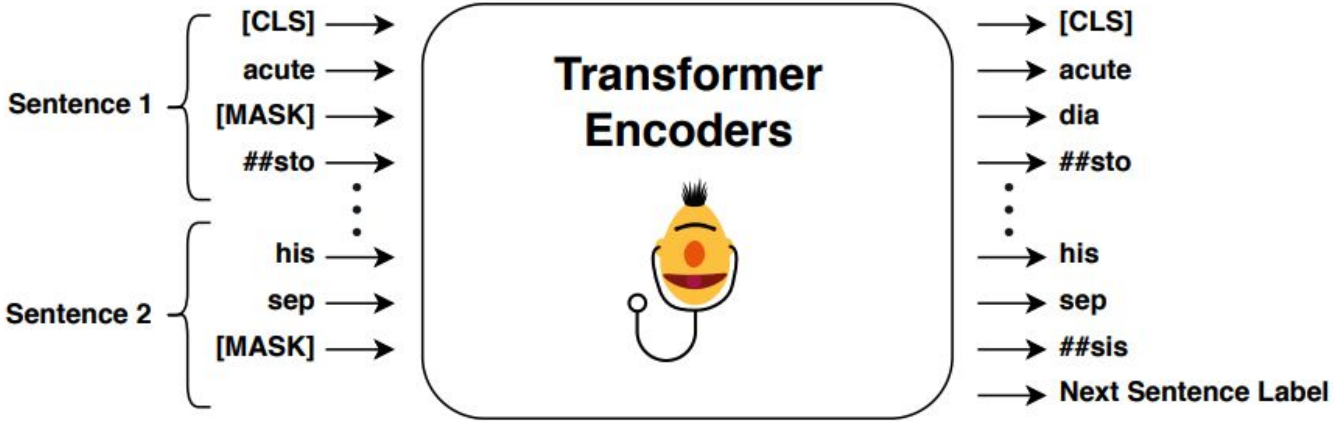
- Also learns representations without external (e.g., manually provided) labels, but instead using labels generated from inherent structure in the data
- Remember BERT training



Huang et al. ClinicalBert: Modeling Clinical Notes and Predicting Hospital Readmission, 2019.

Aside: self-supervised learning

- Also learns representations without external (e.g., manually provided) labels, but instead using labels generated from inherent structure in the data
- Remember BERT training



Also a lot of recent work in contrastive learning. E.g., two transformed versions of an image should have <u>similar</u> representations to each other, and <u>different</u> from transformed versions of other images

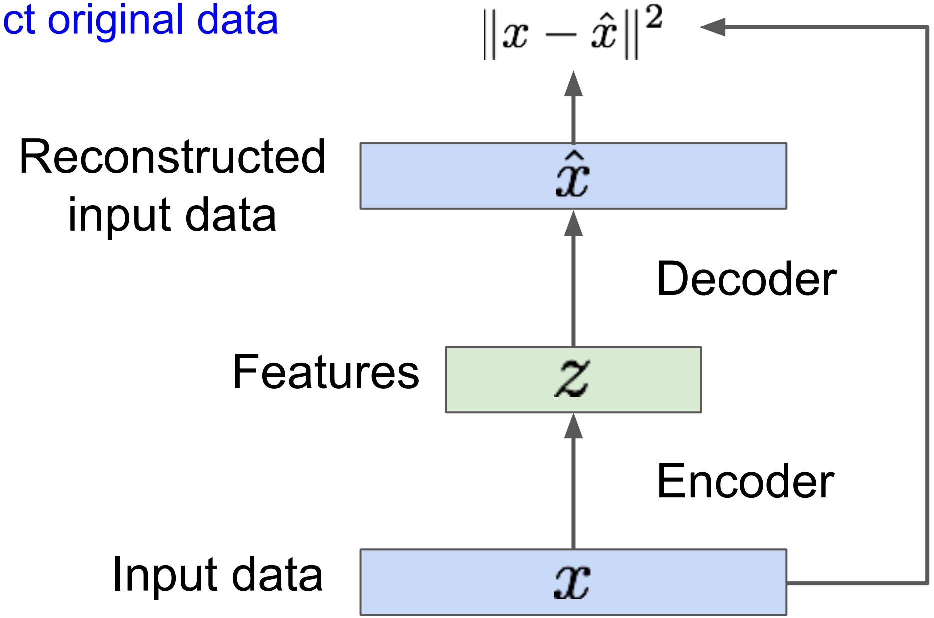
Huang et al. ClinicalBert: Modeling Clinical Notes and Predicting Hospital Readmission, 2019.

Representation learning: autoencoders

Train such that features can be used to reconstruct original data

L2 Loss function:

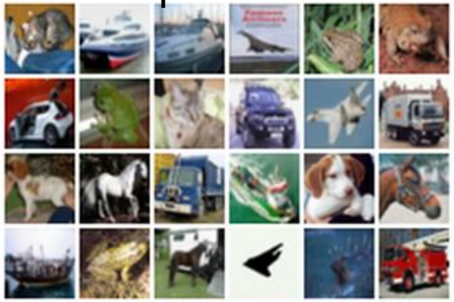
Doesn't use labels!
-> unsupervised



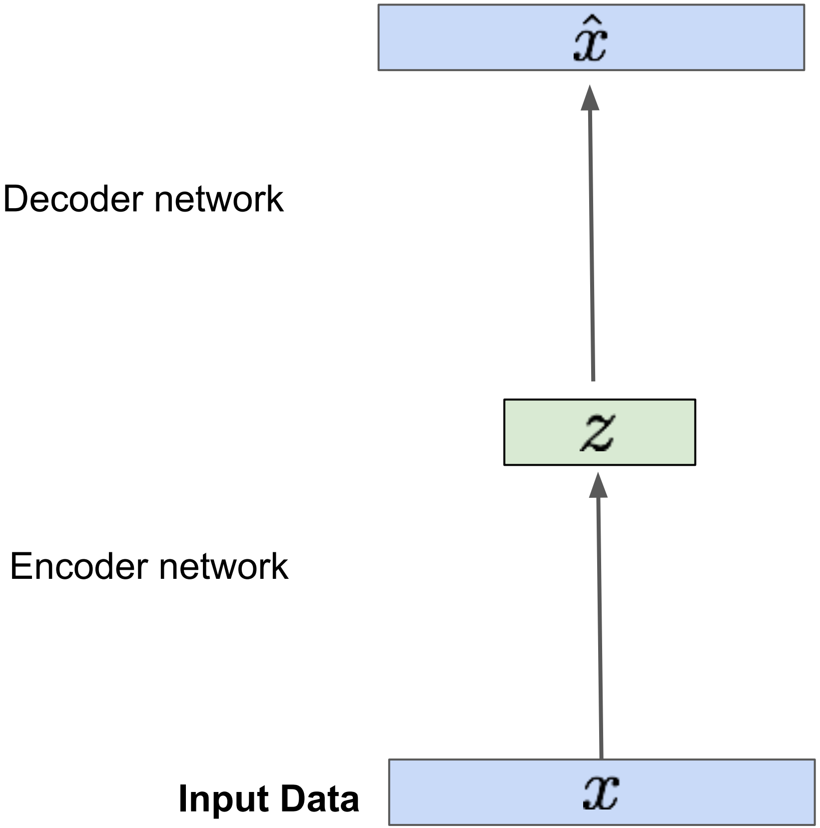
Reconstructed data

Encoder: 4-layer conv
Decoder: 4-layer upconv

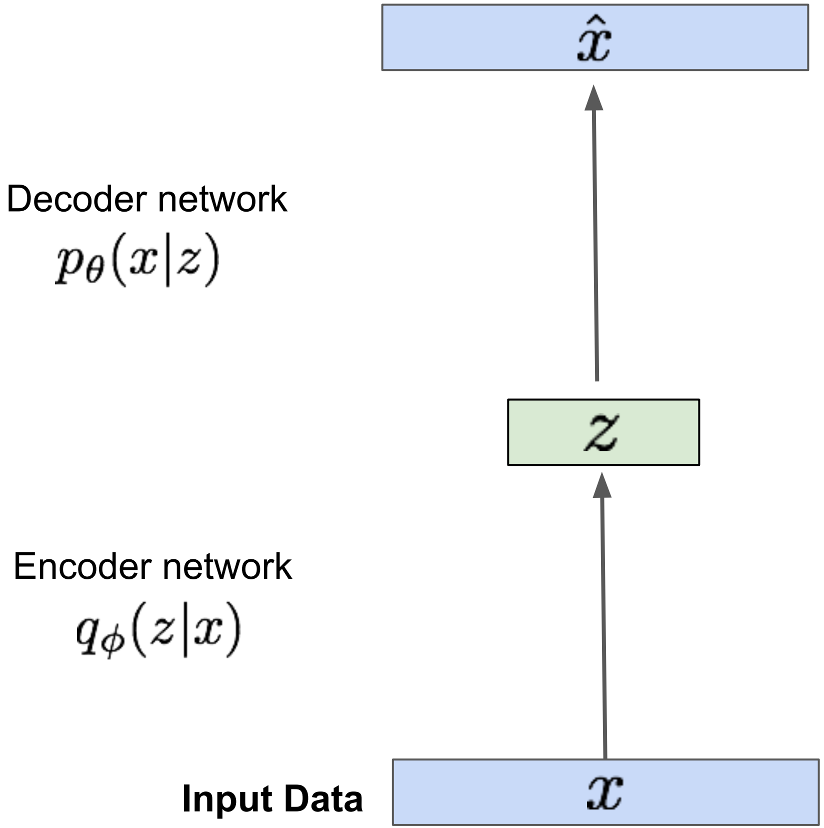
Input data



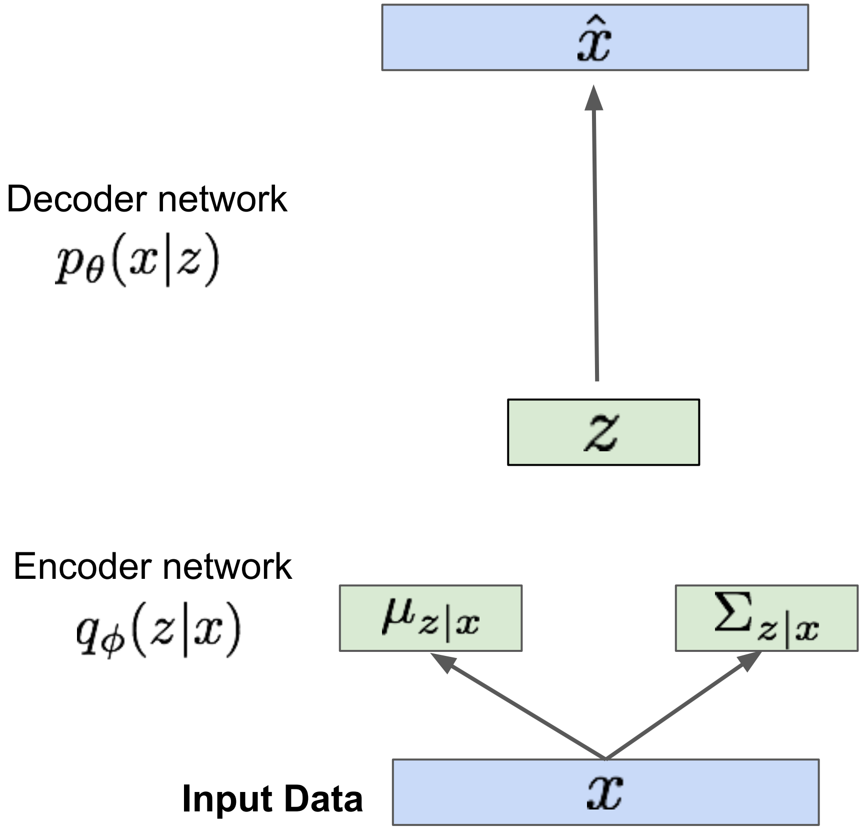
Probabilistic version: variational autoencoder



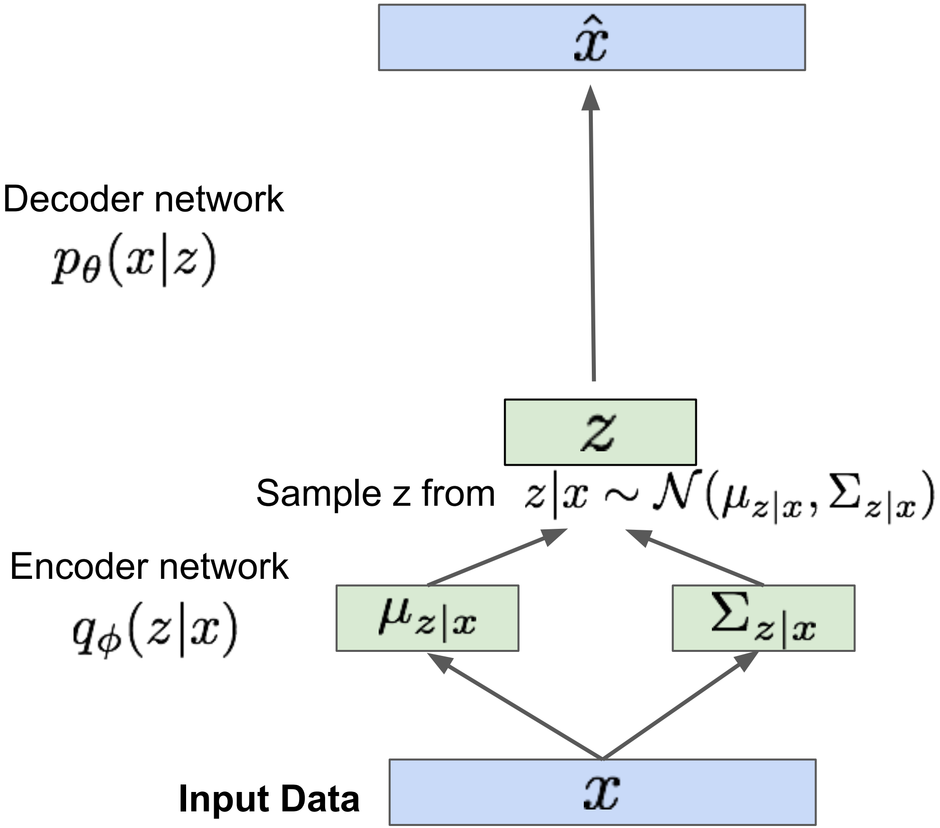
Probabilistic version: variational autoencoder



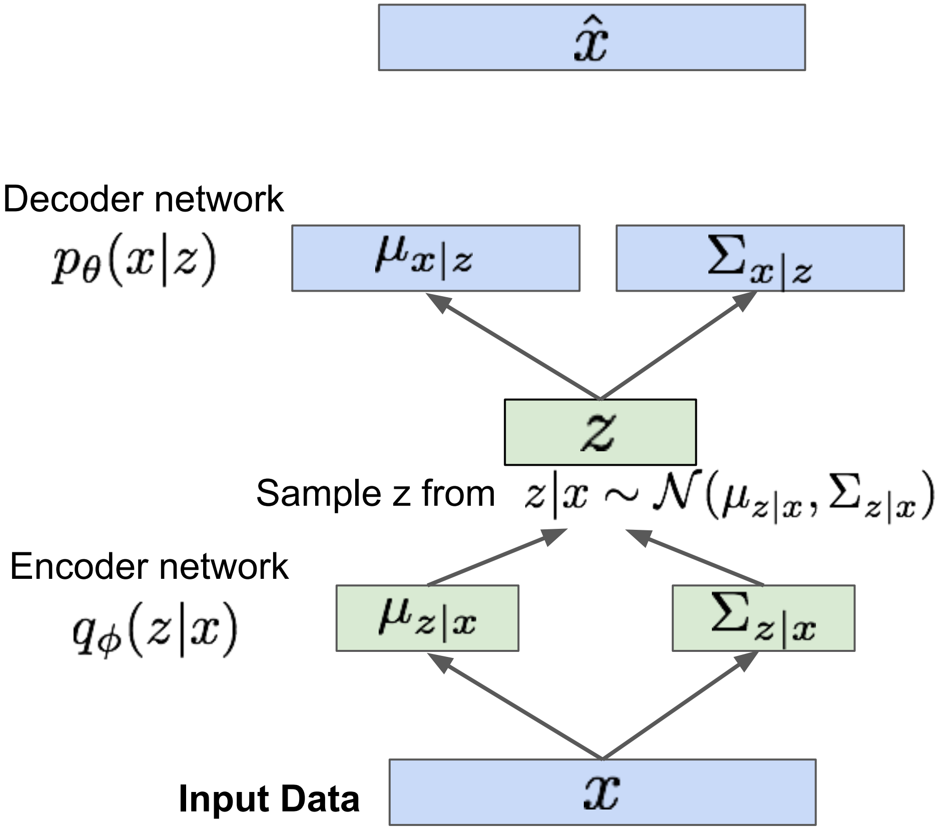
Probabilistic version: variational autoencoder



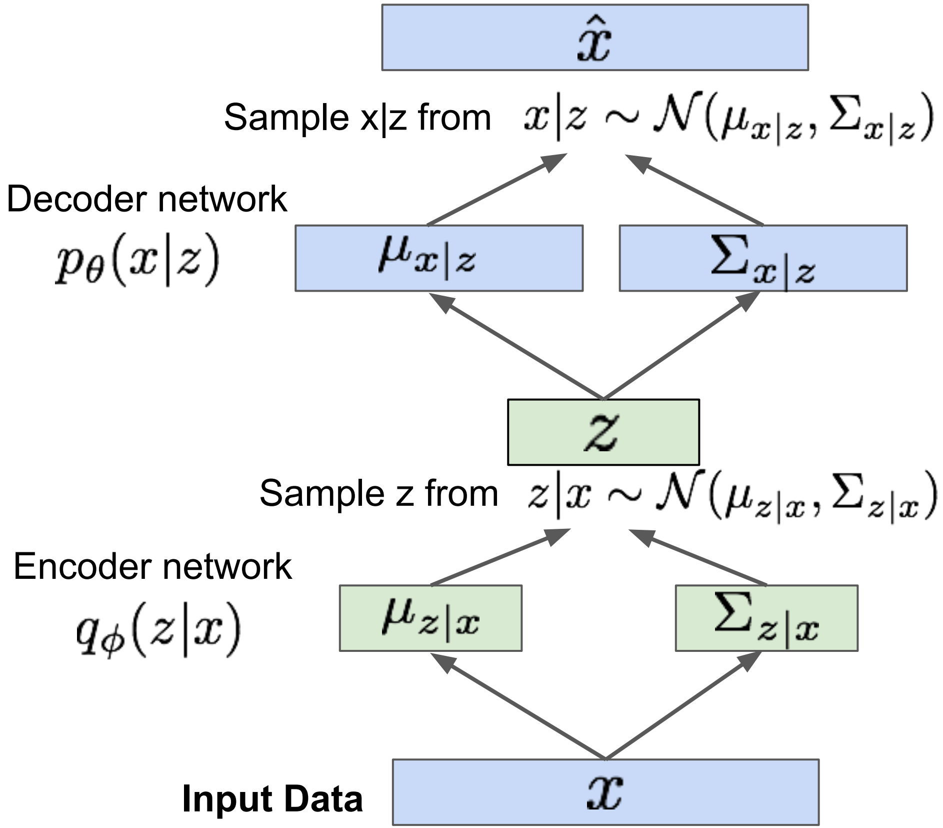
Probabilistic version: variational autoencoder



Probabilistic version: variational autoencoder



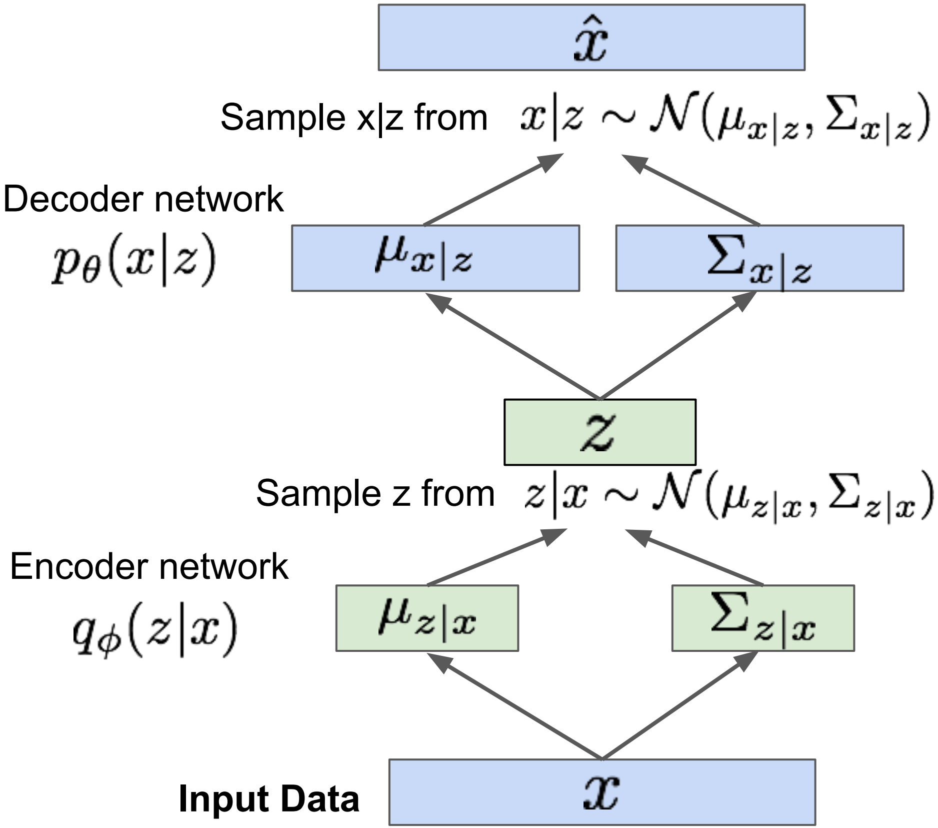
Probabilistic version: variational autoencoder



Probabilistic version: variational autoencoder

Loss function

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

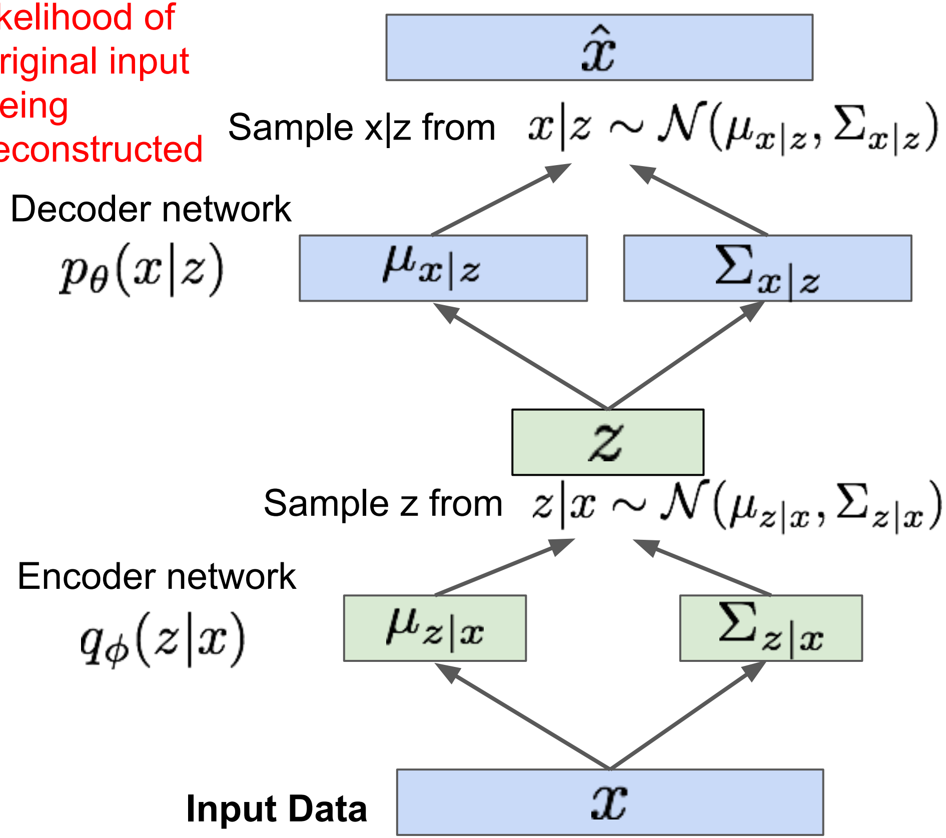


Probabilistic version: variational autoencoder

Maximize likelihood of original input being reconstructed

Loss function

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



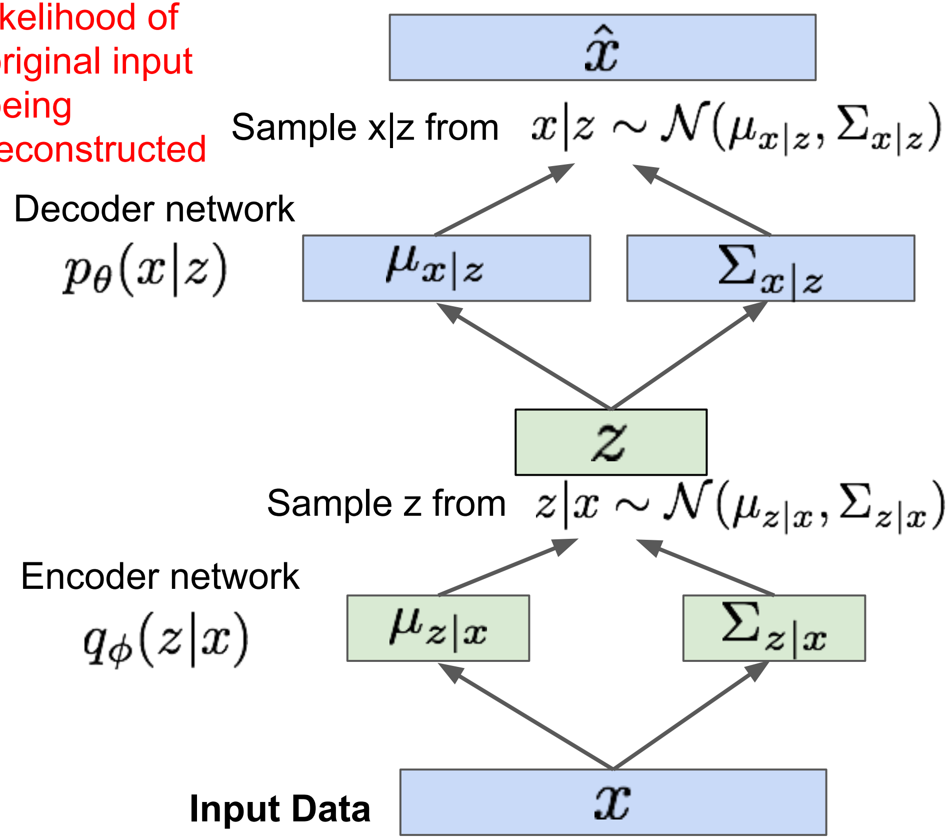
Probabilistic version: variational autoencoder

Maximize likelihood of original input being reconstructed

Loss function

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

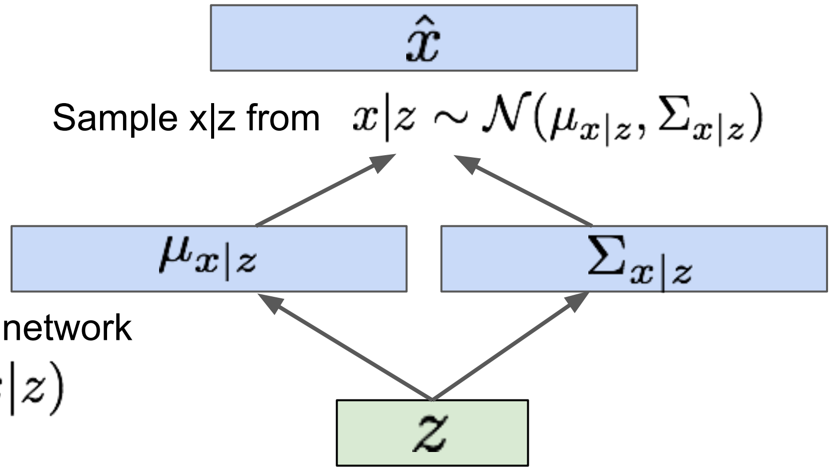
Make output distribution of encoder close to a prior



Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data

Use decoder network. Now sample z from prior!

Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$



Decoder network

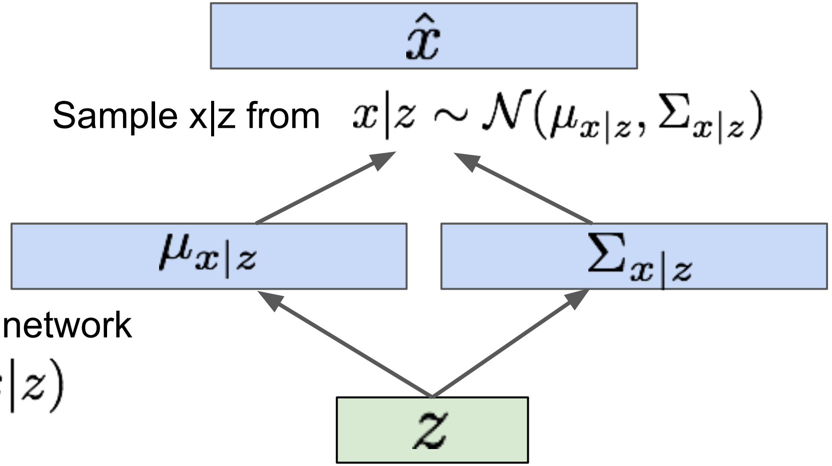
$p_\theta(x|z)$

Sample z from $z \sim \mathcal{N}(0, I)$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data

Use decoder network. Now sample z from prior!

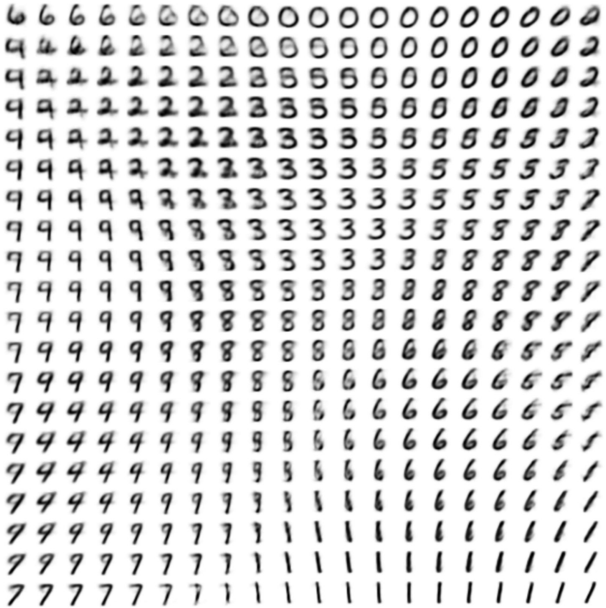


Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

Decoder network

$p_\theta(x|z)$

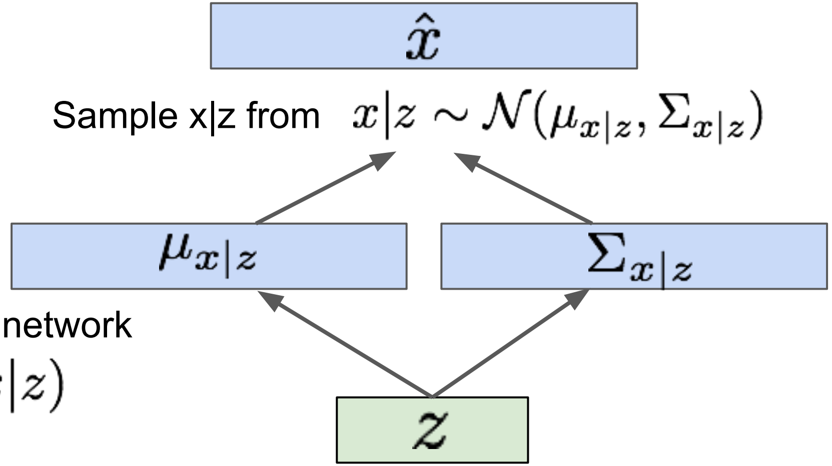
Sample z from $z \sim \mathcal{N}(0, I)$



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data

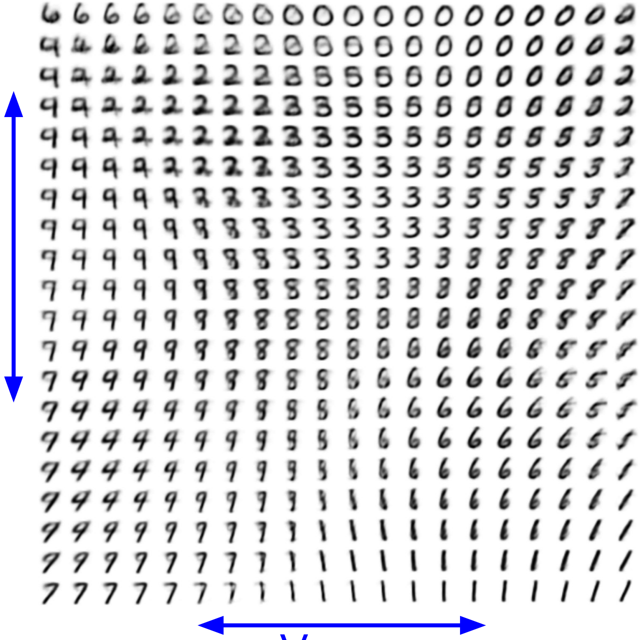
Use decoder network. Now sample z from prior!



Decoder network
$p_\theta(x|z)$

Sample z from $z \sim \mathcal{N}(0, I)$

Data manifold for 2-d **z**



Vary $\mathbf{z_1}$

Vary $\mathbf{z_2}$

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Since variational autoencoders learn distribution of the data, can also be used to generate new (synthetic) data

Different dimensions of **z** encode interpretable factors of variation

Degree of smile

Vary $\mathbf{z}_1$

Vary $\mathbf{z}_2$

Head pose

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Another approach for learning to generate data: generative adversarial networks (GANs)

Motivation: Want to sample (generate data) from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

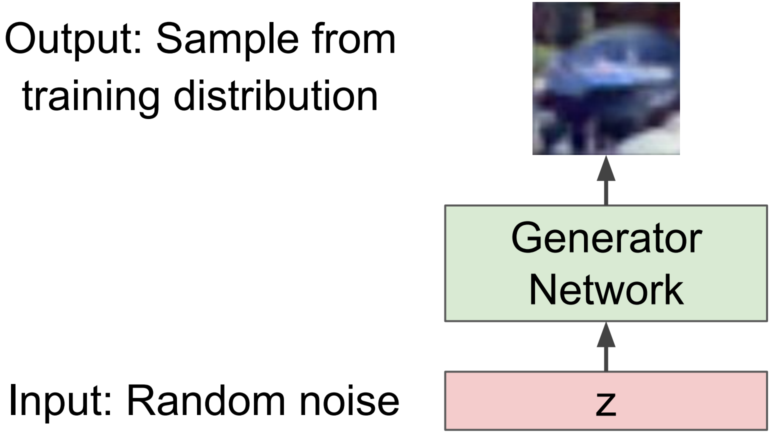
Another approach for learning to generate data: generative adversarial networks (GANs)

Motivation: Want to sample (generate data) from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!



Another approach for learning to generate data: generative adversarial networks (GANs)

Motivation: Want to sample (generate data) from complex, high-dimensional training distribution. No direct way to do this!

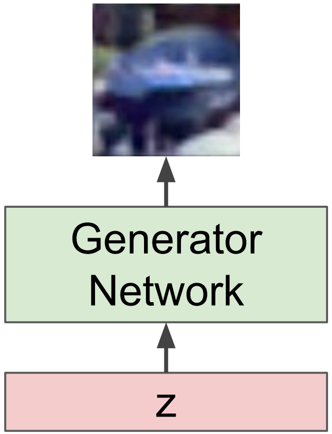
Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!

If goal is generating high quality samples, most current state-of-the-art approaches based on this

Output: Sample from training distribution

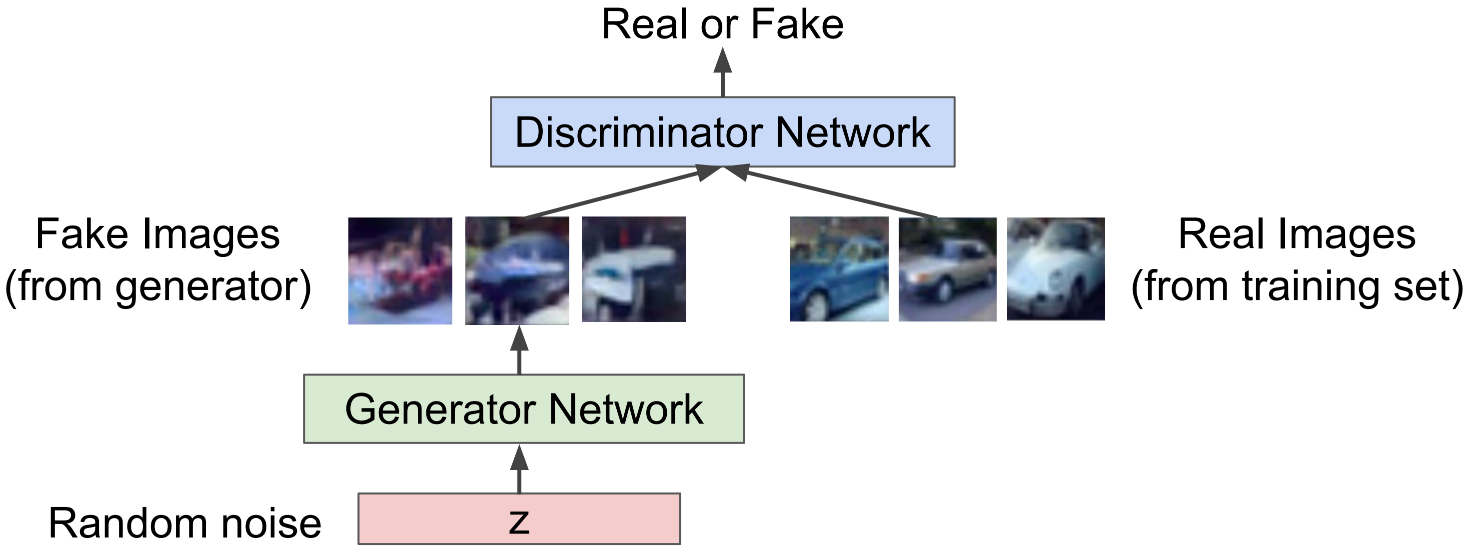


Input: Random noise

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) that image is real

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) that image is real

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **In practice: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

2. **In practice: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **In practice: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Same objective of fooling discriminator, but this objective has some nice properties that make optimization work better in practice

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Aside: Jointly training two networks is challenging, can be unstable. Lots of active research to improve GAN training.

2. **In practice: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Same objective of fooling discriminator, but this objective has some nice properties that make optimization work better in practice

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Putting it together: GAN training algorithm

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Putting it together: GAN training algorithm

Some find k=1 more stable, others use k > 1, no best rule.

More recent GAN variants alleviate this problem, better stability!

for number of training iterations **do**

  for $\boxed{k \text{ steps}}$ **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

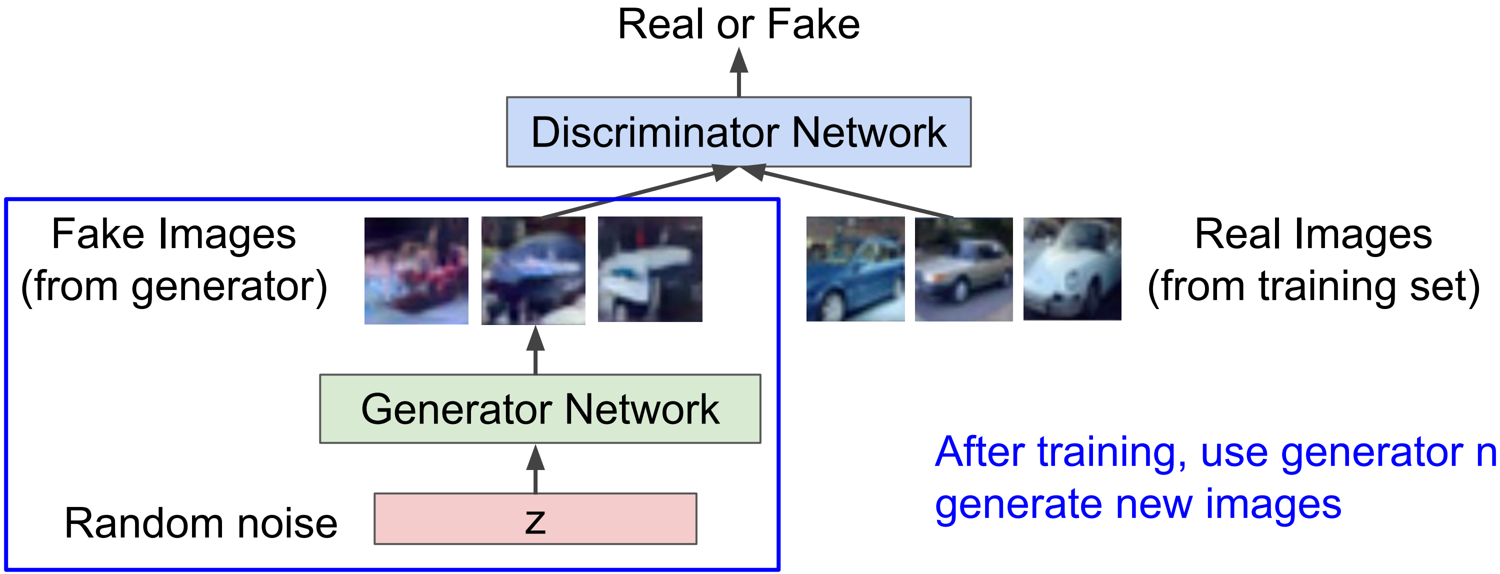
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GANs: Two-player game

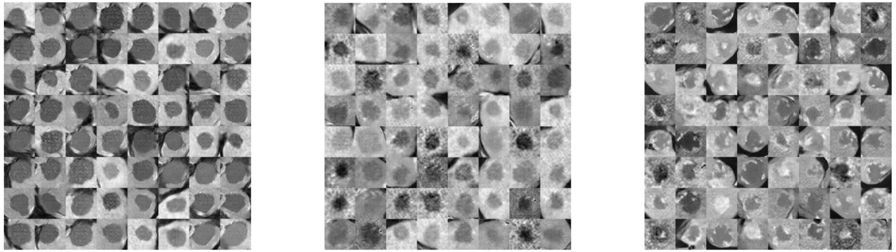
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

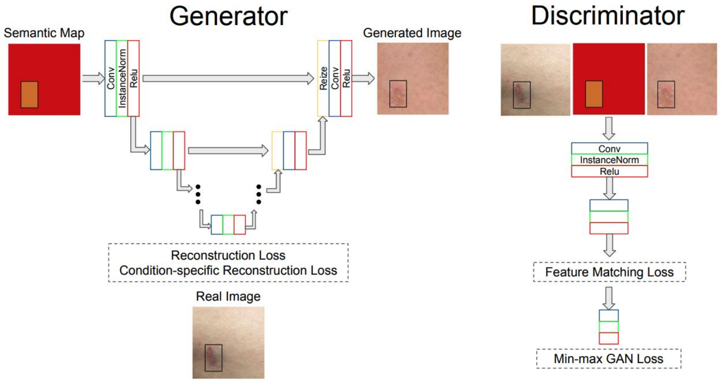


After training, use generator network to generate new images

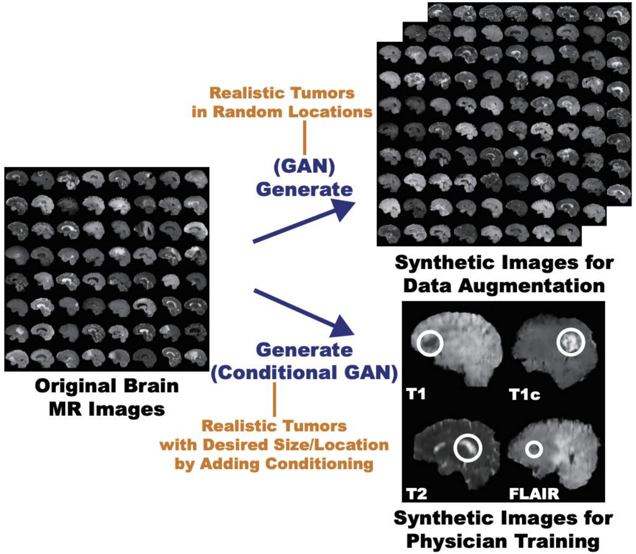
Example: GAN-based medical image synthesis



Liver lesions of different types (Frid-Adar 2018)



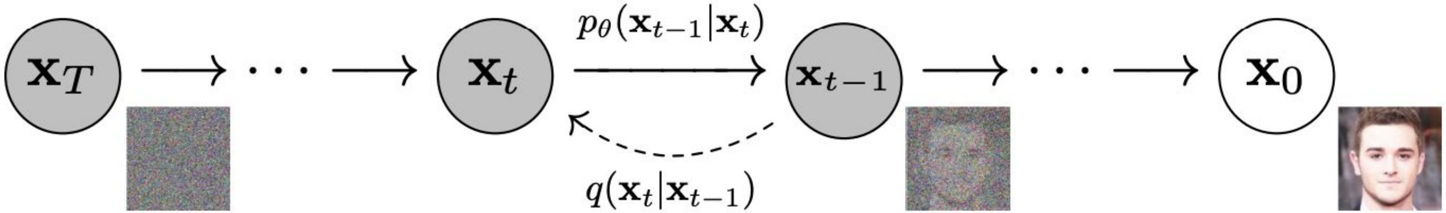
Dermatology lesions (Ghorbani 2019)



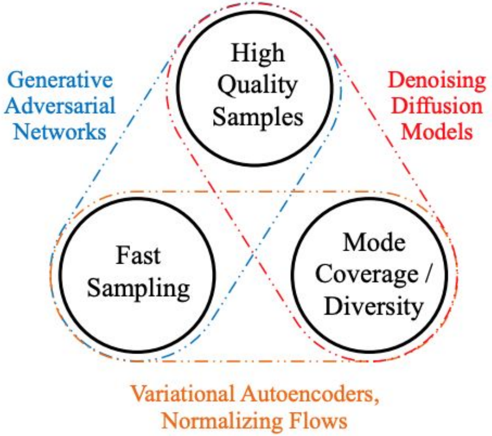
Brain MRIs with lesions (Han 2018)

Can be used for data augmentation!

Aside: diffusion models are an emerging class of generative models



Progressively corrupt training data through adding noise (R->L in figure), then train model to reverse the noising process. At the end, obtain a model that can go from noise to realistic generated images!



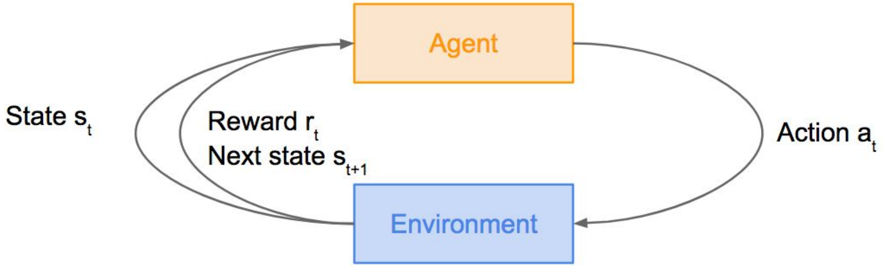
Ho et al. 2020

Bottom figure credit: Xiao et al. 2022

A third paradigm of learning: reinforcement learning

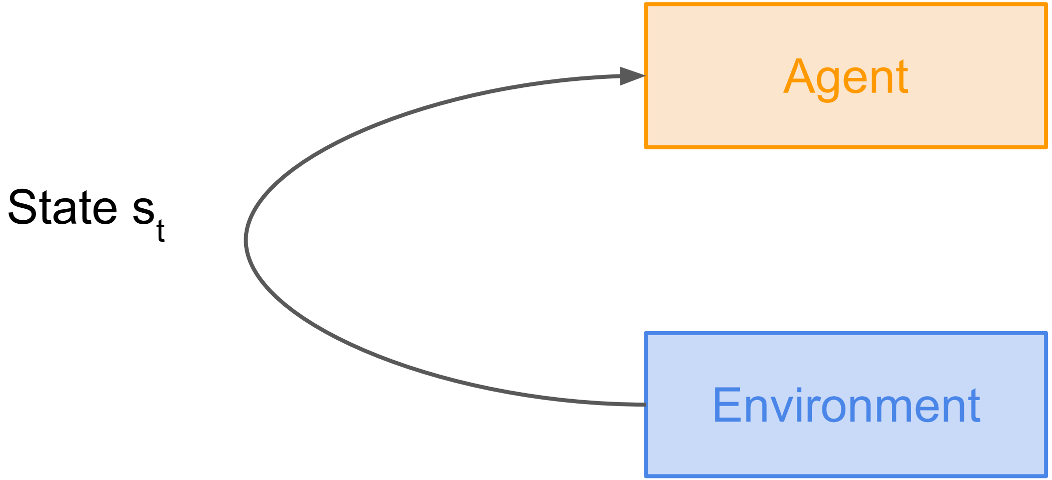
Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward

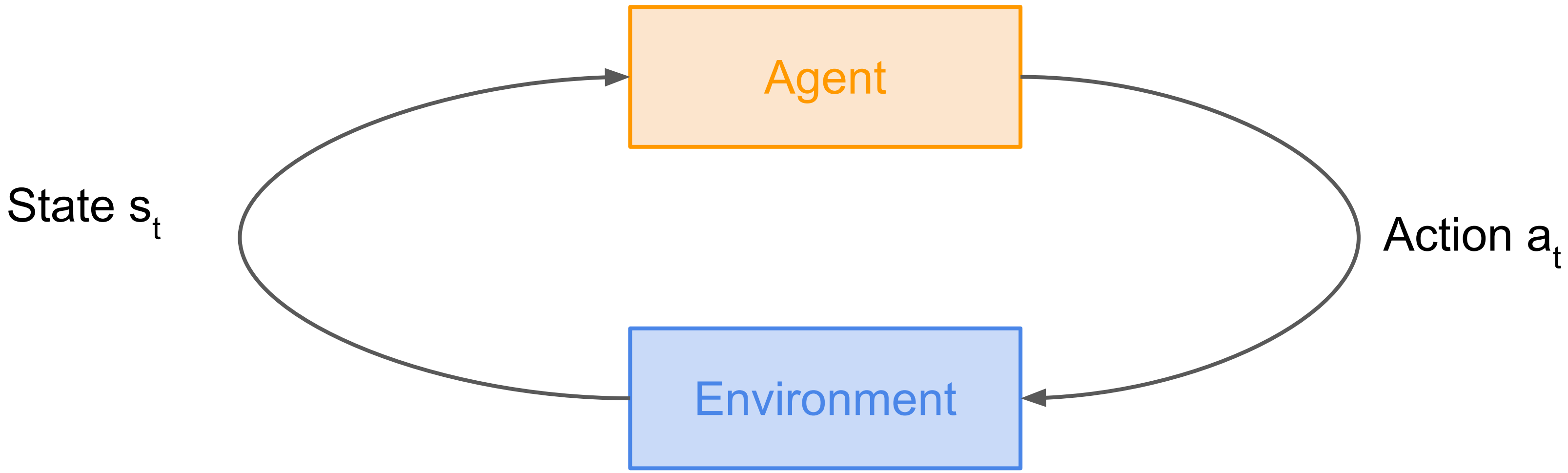


Reinforcement learning

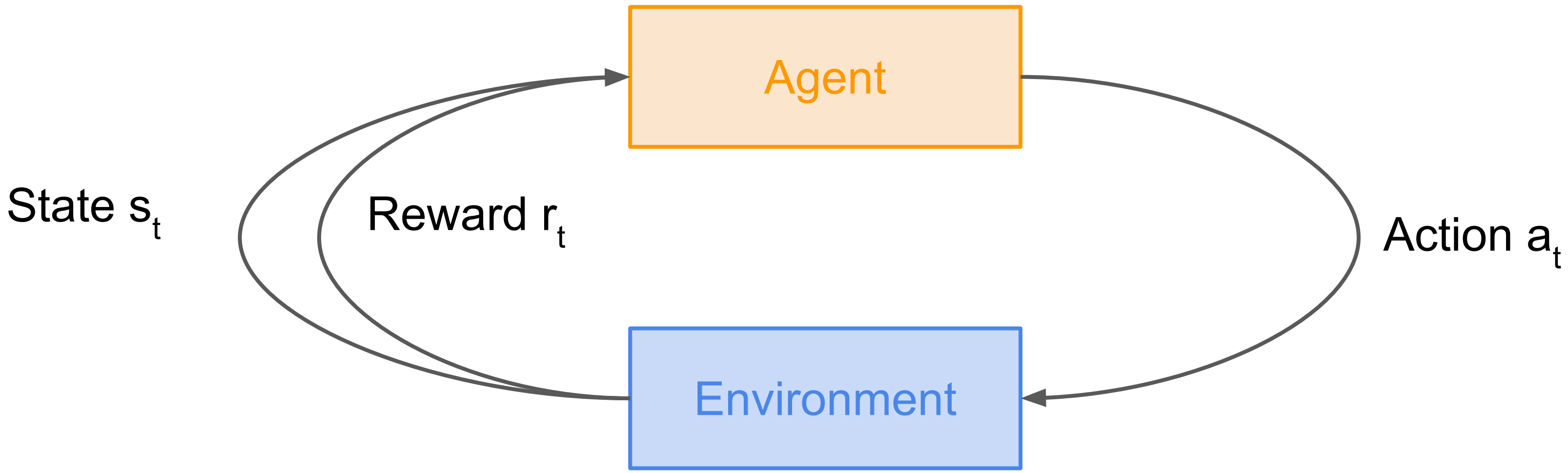
Reinforcement learning



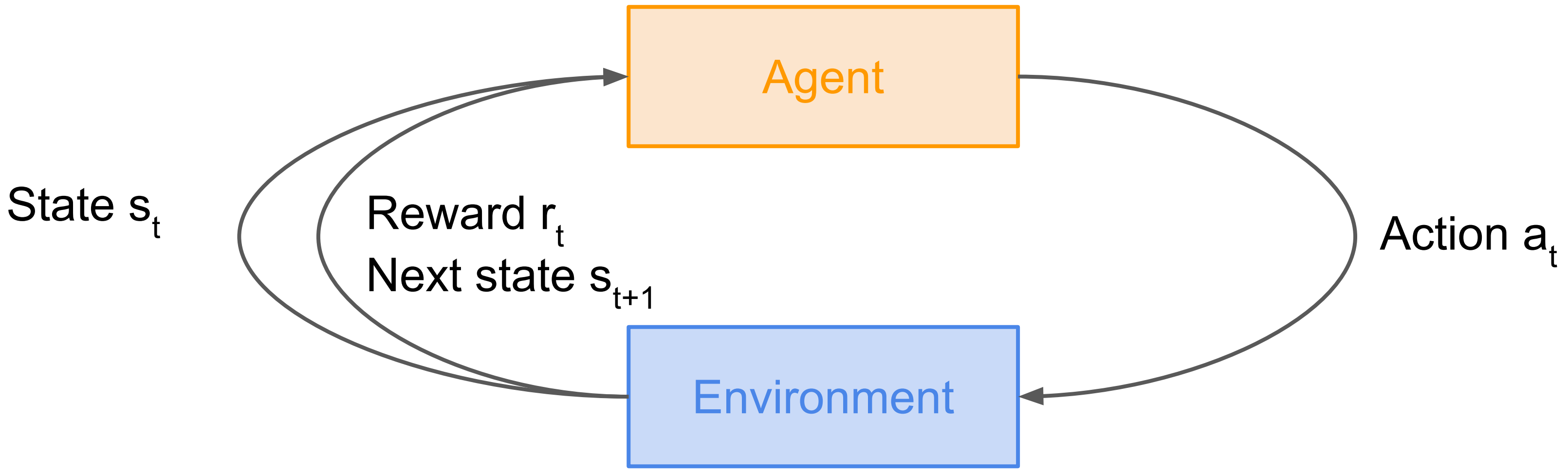
Reinforcement learning



Reinforcement learning



Reinforcement learning



Q-learning (one class of RL methods)

Learn a function (called Q-function) to estimate the expected future reward from taking a particular action from any given state:

$$Q(s, a; \theta)$$

function parameters (weights)

Q-learning (one class of RL methods)

Learn a function (called Q-function) to estimate the expected future reward from taking a particular action from any given state:

$$Q(s, a; \theta)$$

function parameters (weights)

If the function is a deep neural network => **deep q-learning**!

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Famous example: playing Atari games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Q-network architecture

$Q(s, a; \theta)$:
neural network
with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network architecture

Output expected future reward from taking each of the 4 possible actions

$Q(s, a; \theta)$:
neural network
with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Policy gradients (another class of RL methods)

What is a problem with Q-learning?
The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

Policy gradients

What is a problem with Q-learning?
The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

But the policy can be much simpler: just close your hand
Can we learn a policy directly, e.g. finding the best policy from a collection of policies?

Policy gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

Policy gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Policy gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

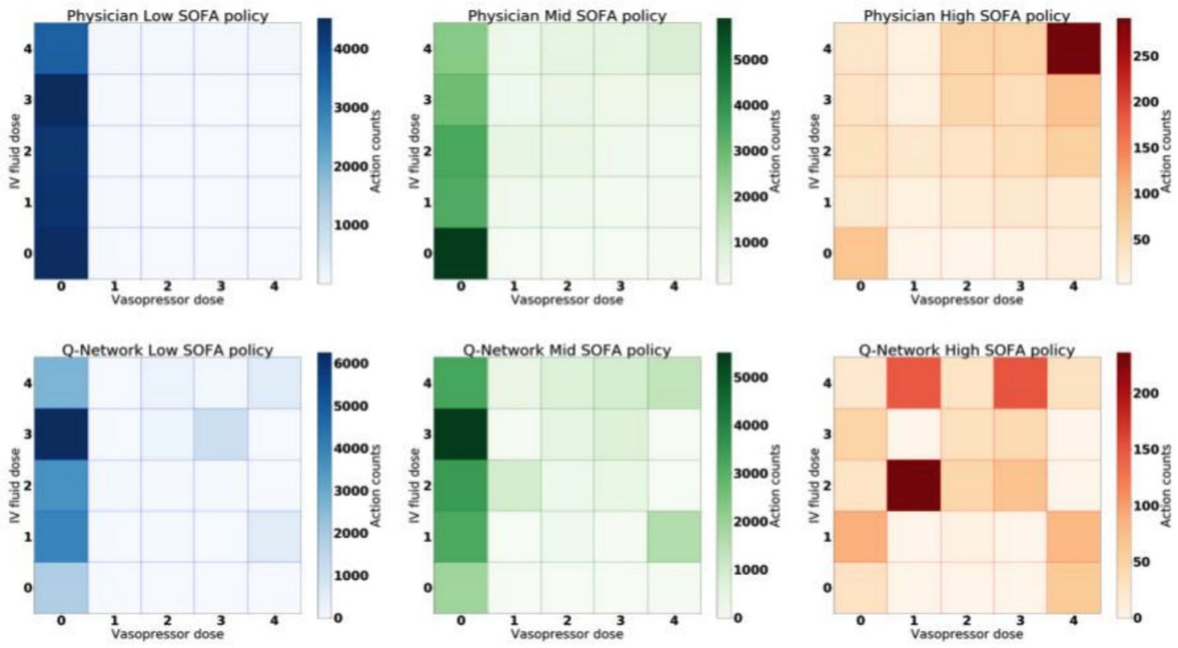
How can we do this?

Gradient ascent on policy parameters!

Example: Raghu et al. 2017

Learned a Q-learning based policy to take treatment actions for sepsis patients, using the MIMIC dataset

5x5 possible policy actions at any timestep



Raghu et al. Deep Reinforcement Learning for Sepsis Treatment, 2017.

Summary

- Unsupervised learning
  - Autoencoders and variational autoencoders
  - Generative Adversarial Networks (GANs)
- Reinforcement learning

Next time: Guest lecture with Dr. Barbara Engelhardt (Genomics: Advanced Topics), extra credit opportunity

Next lecture after that: Interpretability, Fairness, and Ethics