Deep Generative Learning

Supervised vs. Unsupervised Learning

Supervised	Unsupervised
Data: (x, y)	Data: x
x is data, y is label	Only data, no labels!
Goal: Learn function to map	Goal: Learn hidden structure of the data
Examples: Classification, regression, object detection	Examples: Clustering, feature reduction

Generative Modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

Density Estimation: Learns underlying probability distribution to show where data comes from

Sample Generation: Sample to generate new data similar to input

How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

Why generative models?

Debiasing

Able to create fair and representative datasets

Outlier Detection

Detecting when we encounter something rare

Focus in this lecture on latent variable models

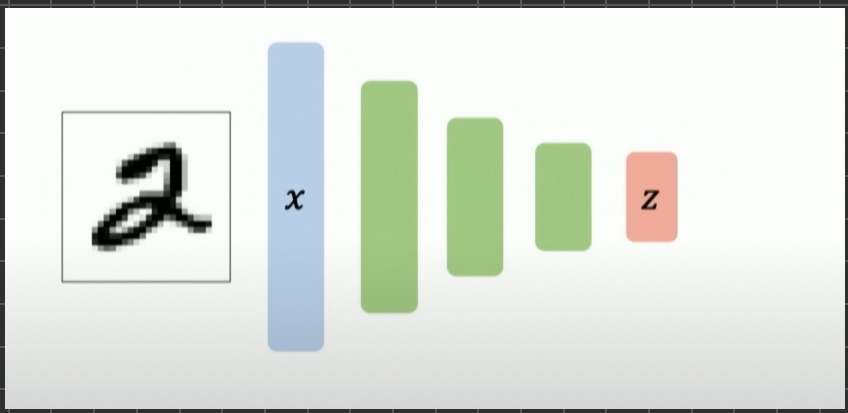
What is a latent variable?

A "shadow" that an object casts

Can we learn the true explanatory factors (latent variables) from only observed data

Autoencoders

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



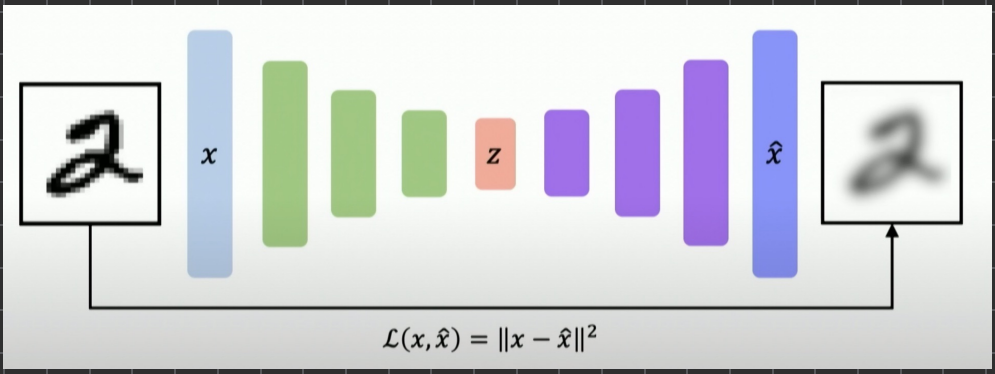
"Encoder" learns mapping from the data to a low-dimensional latent space, z.

We care that z is low dimensional since it can still accurately describe feature rich data without being too specific.

Background

How can we learn this latent space?

Train the model to use these features to reconstruct the original data



Uses squared error loss (doesn't need labels)

Dimensionality of Latent Space

Autoencoding is a form of compression!

Small latent space will lead to training bottleneck

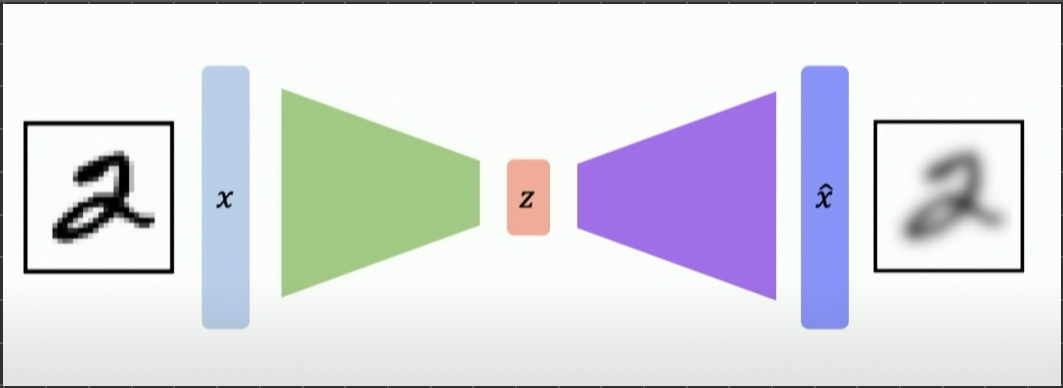
Summary

Bottleneck hidden layer forces network to learn compressed latent representation

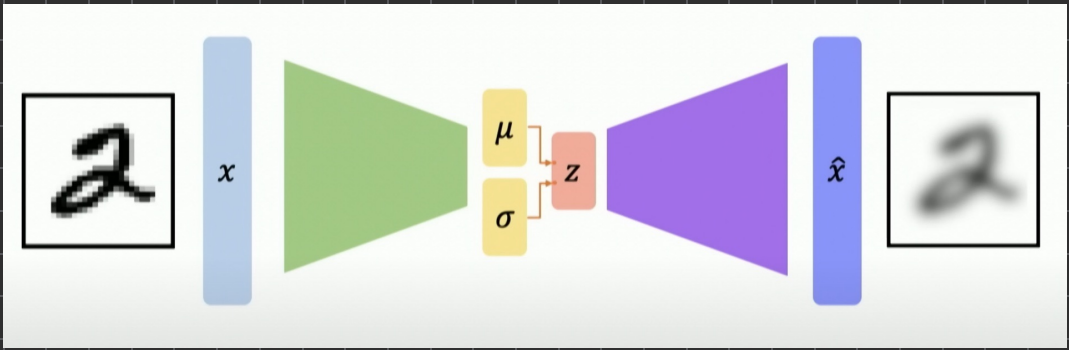
Reconstruction loss forces the latent representation to be detailed

Variational Autoencoders (VAEs)

Traditional:



VAEs



For each latent variable z, we establish a mean (μ) and a standard deviation (σ) to capture a probability distribution.

Encoder computes $q_\phi(z|x)$ Decoder computes $p_\theta(x|z)$

$L(\phi, \theta, x)$ = reconstruction loss + regularization term (VAE loss)

Regularization Term

$$D(q_\phi(z|x) \| p(z))$$

- $p(z)$: Fixed prior on latent distribution
- $q_\phi(z|x)$: Inferred latent distribution

The term D is essentially the regularization term

Common choice of prior

Normal Gaussian

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize network when it tries to "cheat" by clustering points in specific regions

$$\boxed{D(q_\phi(z|x) \| p(z)) = -\frac{1}{2}\sum_{j=0}^{K-1}\left(\sigma_j + \mu_j^2 - 1 - \log(\sigma_j)\right)}$$

KL Divergence is the $-\frac{1}{2}\sum_{j=0}^{K-1}\left(\sigma_j + \mu_j^2 - 1 - \log(\sigma_j)\right)$

Intuition on regularization and the normal prior

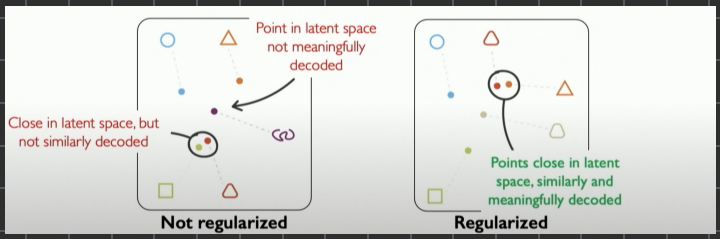
What properties do we want to achieve from regularization?

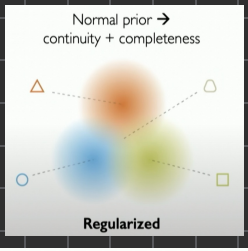
1.) Continuity

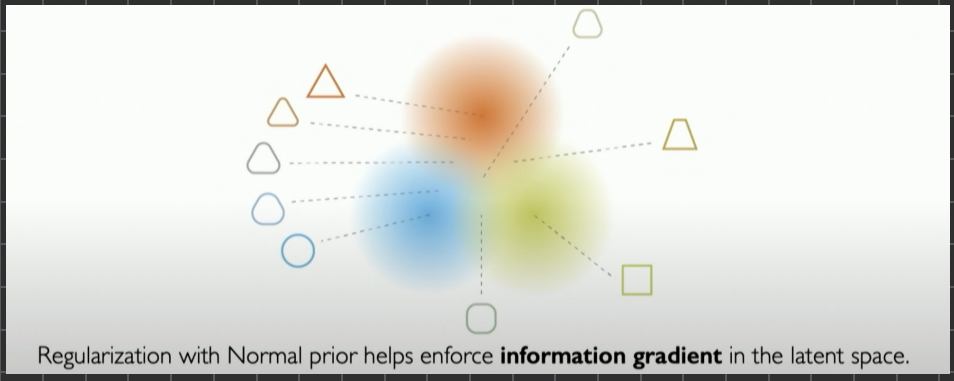
- points that are close in latent space have similar content after decoding

2.) Completeness

- Meaningful data after decoding







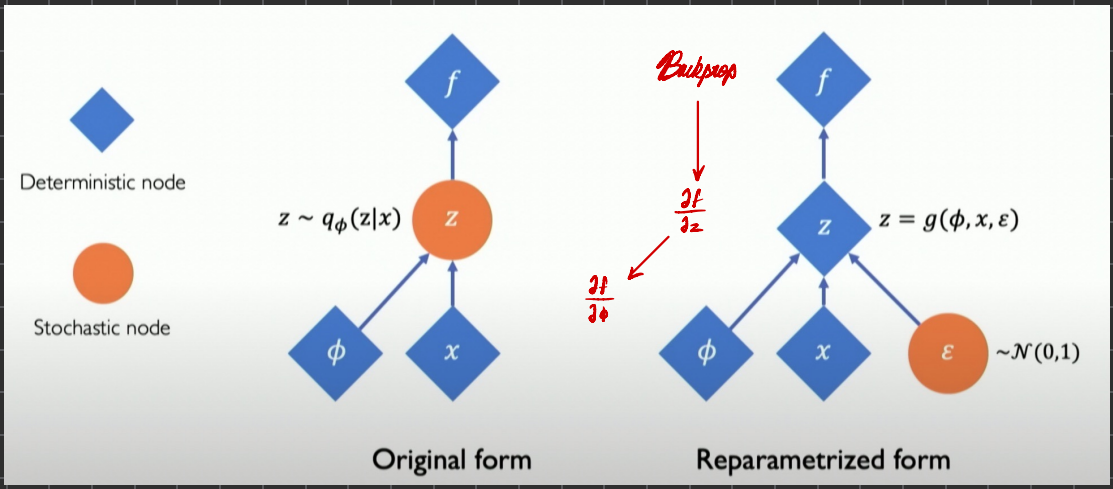
VAE computation graph

Problem: We cannot backpropogate gradients!

Reparameterizing the sample layer

- z is <u>not</u> $\mathcal{N}(\mu, \sigma^2)$
- $z = \mu + \sigma \odot \varepsilon$ (where $\varepsilon \simeq \mathcal{N}(0,1)$)
- Consider the sampled latent vector z as a sum of
  - A fixed vector μ
  - A fixed vector σ, scaled by random constants drawn from prior distribution

Cannot backpropogate through stochastic nodes



VAEs: Latent Perturbation

- Slowly increase or decrease a single latent variable
- Keep all other variable fixed
- Different dimensions of z encodes different interpretable latent features
- Ideally, we want latent variables that are independent of each other
- Enforce diagonal prior on the latent variables to encourage independence

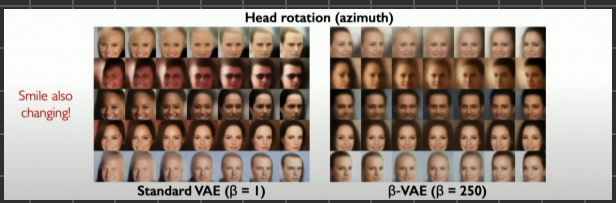
  ⟶ **Disentanglement**

Latent space disentanglement with β-VAEs

Standard VAE loss

$$L(\theta, \phi; x, z, \beta) = \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]}_{\text{Reconstruction Term}} - \beta \underbrace{D_{KL}\left(q_\phi(z|x) \| p(z)\right)}_{\text{Regularization Term}}$$

$\beta > 1$: constrain latent bottleneck, encourage efficient latent encoding



VAE summary

1.) Compress representation of world to something we can use to learn

2.) Reconstruction allows for unsupervised learning

3.) Reparameterization trick to train end-to-end

4.) Interpret hidden latent variables using perturbation

5.) Generate new examples

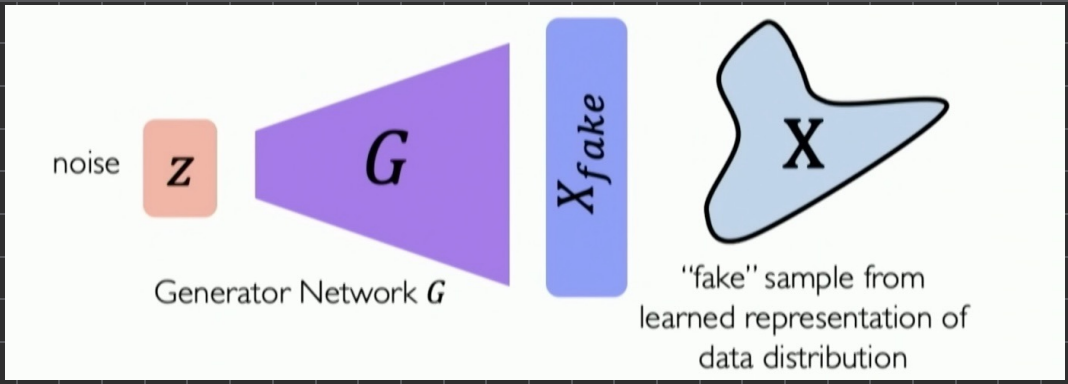
Generative Adversarial Networks (GANs)

What if we just want to sample?

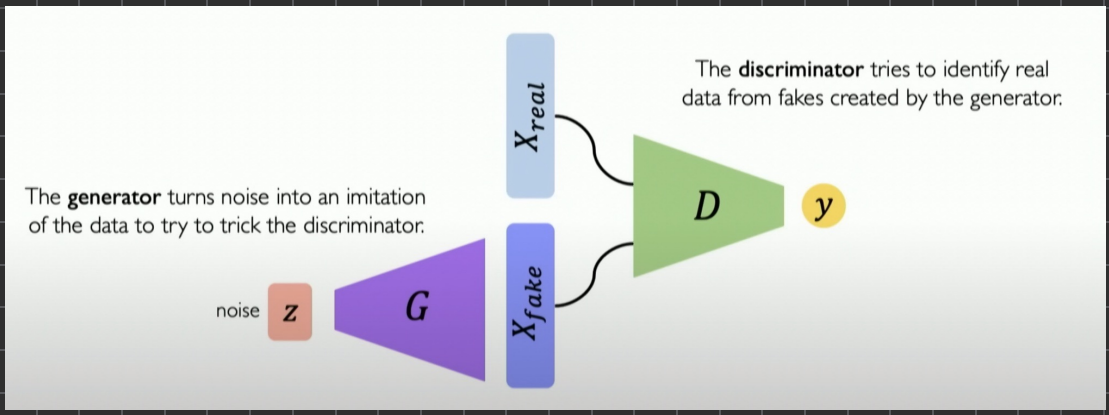
Idea: don't explicitly model density and instead just sample to generate new instances

Problem: Want to model from complex distribution

Solution: Sample something from simple (e.g. noise), learn a transformation to the data distribution



Generative Adversarial Networks (GANs) (cont.)



Intuition behind GANs

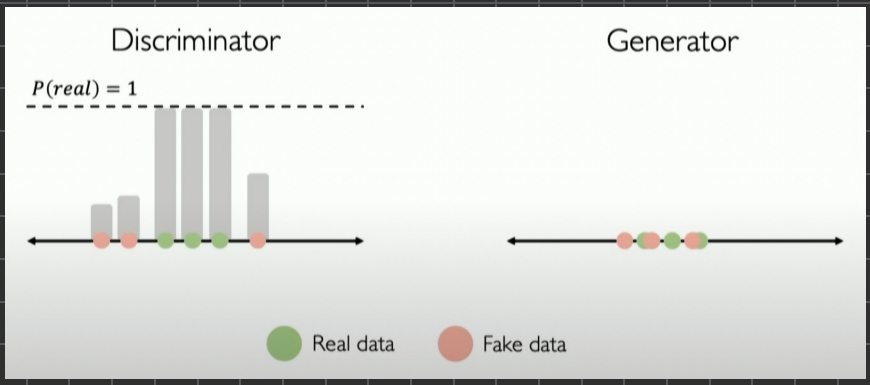
1.) Generator starts from noise to try to create an imitation of the data

2.) Discriminator tries to predict what's real and what's fake

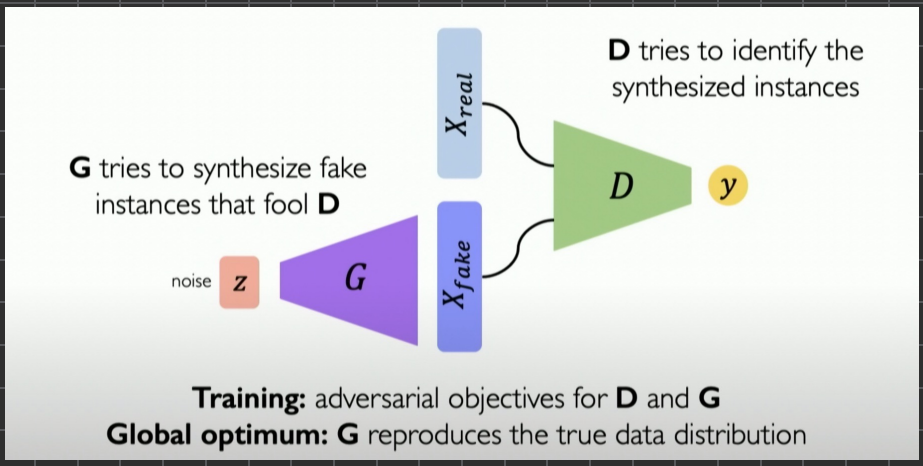
3.) Generator tries to improve imitation of data

4.) Discriminator again tries prediction

5.) Repeat



Training GANs: Loss Function



$$L = \arg\max_D \mathbb{E}_{z,x}\left[\log D(G(z)) + \log(1 - D(x))\right]$$

Fake data

Real data

D tries to identify synthesized instances

$$L = \arg\min_G \mathbb{E}_{z,x}\left[\log D(G(z)) + \log(1 - D(x))\right]$$

G wants to minimize probability that data is detected as fake

Generating New Data with GANs

After training, just use the generator!

GAN Advances and Applications

- Progressive growing of GANs (more and more detail!)
- Conditional GANs (Control the nature of an output)
  - Works for paired translations
- Cycle GAN (learns transformations across domains with unpaired data)