

## Generative Models, Part 2

Discriminative Model:

- Learn distribution  $p(y | x)$

Generative Model:

- Learn distribution  $p(x)$
- Assign probability to each image in the universe

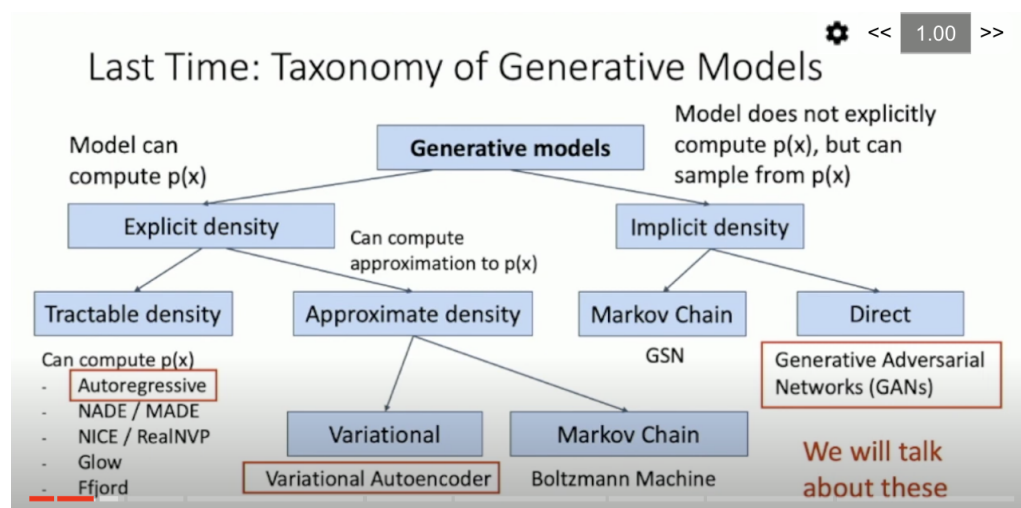
Conditional Generative Model:

- Learn  $p(x, y)$
- Easy to build this type of model when we have built the other types of models

Recall **Bayes' Rule**:

$$\underbrace{P(x | y)}_{\text{Conditional Generative Model}} = \frac{\underbrace{P(y | x)}_{\text{Discriminative Model}}}{\underbrace{P(y)}_{\text{Prior over labels}}} \underbrace{P(x)}_{\text{(Unconditional) Generative Model}}$$

We can build a conditional generative model from other components!



## Variational Autoencoders:

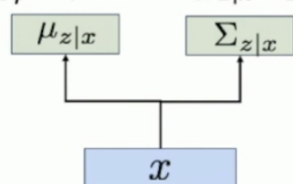
- Gain: In addition to modeling likelihood of data, learn latent representation of data (z)

Jointly train **encoder** q and **decoder** p to maximize the **variational lower bound** on the data likelihood

$$\log p_{\theta}(x) \geq E_{z \sim q_{\phi}(z|x)} [\log p_{\theta}(x|z)] - D_{KL}(q_{\phi}(z|x), p(z))$$

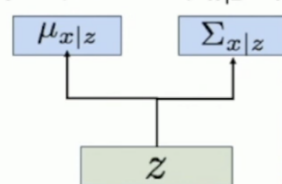
### Encoder Network

$$q_{\phi}(z|x) = N(\mu_{z|x}, \Sigma_{z|x})$$



### Decoder Network

$$p_{\theta}(x|z) = N(\mu_{x|z}, \Sigma_{x|z})$$



- Maximize lower-bound on data-likelihood
- Encoder and decoder have to output probability distributions, not just a normal sample/vector
  - All probability distributions are diagonal gaussian distributions, so only need mu and sigmas to parametrize
- Prior on p(z) is said to be gaussian so we can actually compute KL-divergence

YouTube Home

## Variational Autoencoders

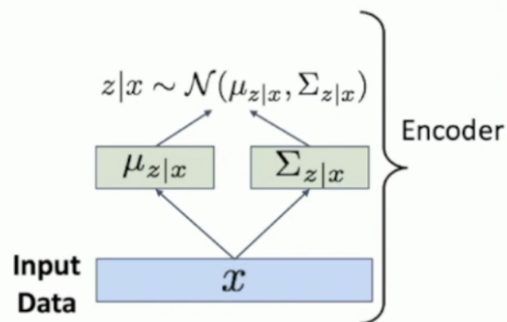
Train by maximizing the **variational lower bound**

$$E_{z \sim q_{\phi}(z|x)} [\log p_{\theta}(x|z)] - D_{KL}(q_{\phi}(z|x), p(z))$$

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**

$$\begin{aligned} -D_{KL}(q_{\phi}(z|x), p(z)) &= \int_z q_{\phi}(z|x) \log \frac{p(z)}{q_{\phi}(z|x)} dz \\ &= \int_z N(z; \mu_{z|x}, \Sigma_{z|x}) \log \frac{N(z; 0, I)}{N(z; \mu_{z|x}, \Sigma_{z|x})} dz \\ &= \sum_{j=1}^J (1 + \log((\Sigma_{z|x})_j^2) - (\mu_{z|x})_j^2 - (\Sigma_{z|x})_j^2) \end{aligned}$$

Closed form solution when  $q_{\phi}$  is diagonal Gaussian and  $p$  is unit Gaussian! (Assume  $z$  has dimension  $J$ )

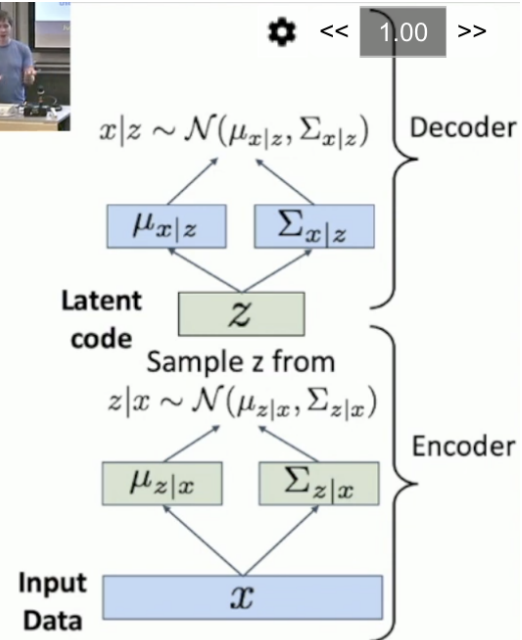


# Variational Autoencoders

Train by maximizing the  
**variational lower bound**

$$E_{z \sim q_{\phi}(z|x)} [\log p_{\theta}(x|z)] - D_{KL}(q_{\phi}(z|x), p(z))$$

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior  $p(z)$ !**
3. Sample code  $z$  from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples
5. **Original input data should be likely under the distribution output from (4)!**



- Blue is data reconstruction term:
  - When you sample  $z$  from latent space, and then put that  $z$  vector back into the decoder, the original image should be most likely
  - Sort of like mean squared error loss in traditional autoencoder, incentivizes model to output the original input data
- Green term is regularization term
  - Predicted distribution should be simple and should be close to gaussian distribution
- Blue and green term are fighting against each other
- Can generate data by sampling from latent space once we have trained VAE
- Each dimension of  $z$  is independent (due to constraint that latent space follows diagonal gaussian)
- Can vary different elements of VAE to get smooth transitions between different aspects of the image
- Can also edit images by introducing perturbations in certain dimension of latent space

# Variational Autoencoder: Summary



Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

## Pros:

- Principled approach to generative models
- Allows inference of  $q(z|x)$ , can be useful feature representation for other tasks

## Cons:

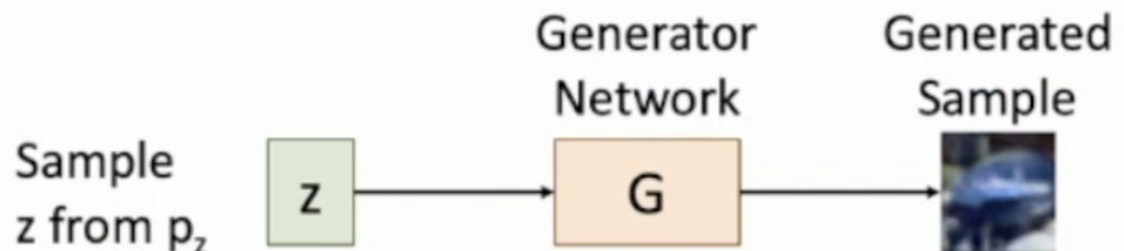
- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

## Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs)
- Incorporating structure in latent variables, e.g., Categorical Distributions

## GANs

- Give up modeling  $p(x)$ , but allow to draw samples from  $p(x)$



Train **Generator Network G** to convert  $z$  into fake data  $x$  sampled from  $p_G$

- 
- Generator and discriminator are fighting against each other
- Hope is that samples from generator will end very similar to samples from actual data

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log (1 - D(G(z)))] \right)$$

-

- $D(x) = 1$  for real data
- Generator does care about left-hand side
- Discriminator wants  $D(x) = 0$  for fake data
- Right side incentivizes generator to be as good as possible

Use alternating gradient descent updates

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \left( E_{x \sim p_{data}} [\log \mathbf{D}(x)] + E_{z \sim p(z)} [\log (1 - \mathbf{D}(\mathbf{G}(z)))] \right)$$

$$= \min_{\mathbf{G}} \max_{\mathbf{D}} V(\mathbf{G}, \mathbf{D})$$

For t in 1, ... T:

1. (Update  $\mathbf{D}$ )  $\mathbf{D} = \mathbf{D} + \alpha_D \frac{\partial V}{\partial \mathbf{D}}$
2. (Update  $\mathbf{G}$ )  $\mathbf{G} = \mathbf{G} - \alpha_G \frac{\partial V}{\partial \mathbf{G}}$

- dffellow et al, "Generative Adversarial Nets", NeurIPS 2014

- Use gradient ascent for D (since we're trying to maximize)
- Use gradient descent for G (since we're trying to minimize)

Problem: Not minimizing any overall loss! No training curves that we can observe during training

Problem #2:  $D(G(z))$  is close to zero

Problem: Vanishing gradients for G (cuz it sucks it in the beginning)

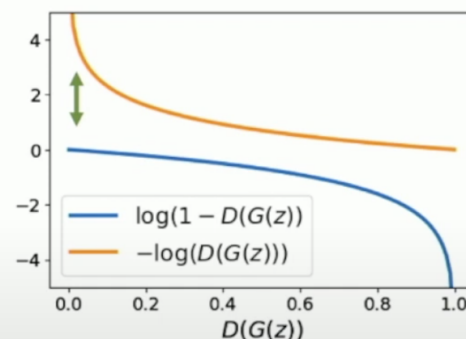
Jointly train generator G and discriminator D with a **minimax game**

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \left( E_{x \sim p_{data}} [\log \mathbf{D}(x)] + E_{z \sim p(z)} [\log (1 - \mathbf{D}(\mathbf{G}(z)))] \right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so  $D(G(z))$  close to 0

**Problem: Vanishing gradients for G**

**Solution:** Right now G is trained to minimize  $\log(1 - D(G(z)))$ . Instead, train G to maximize  $-\log(D(G(z)))$ . Then G gets strong gradients at start of training!



Solution: dffellow et al, "Generative Adversarial Nets", NeurIPS 2014

Is it supposed to be minimize or maximize :o???

## Conditional GANs

- Given class