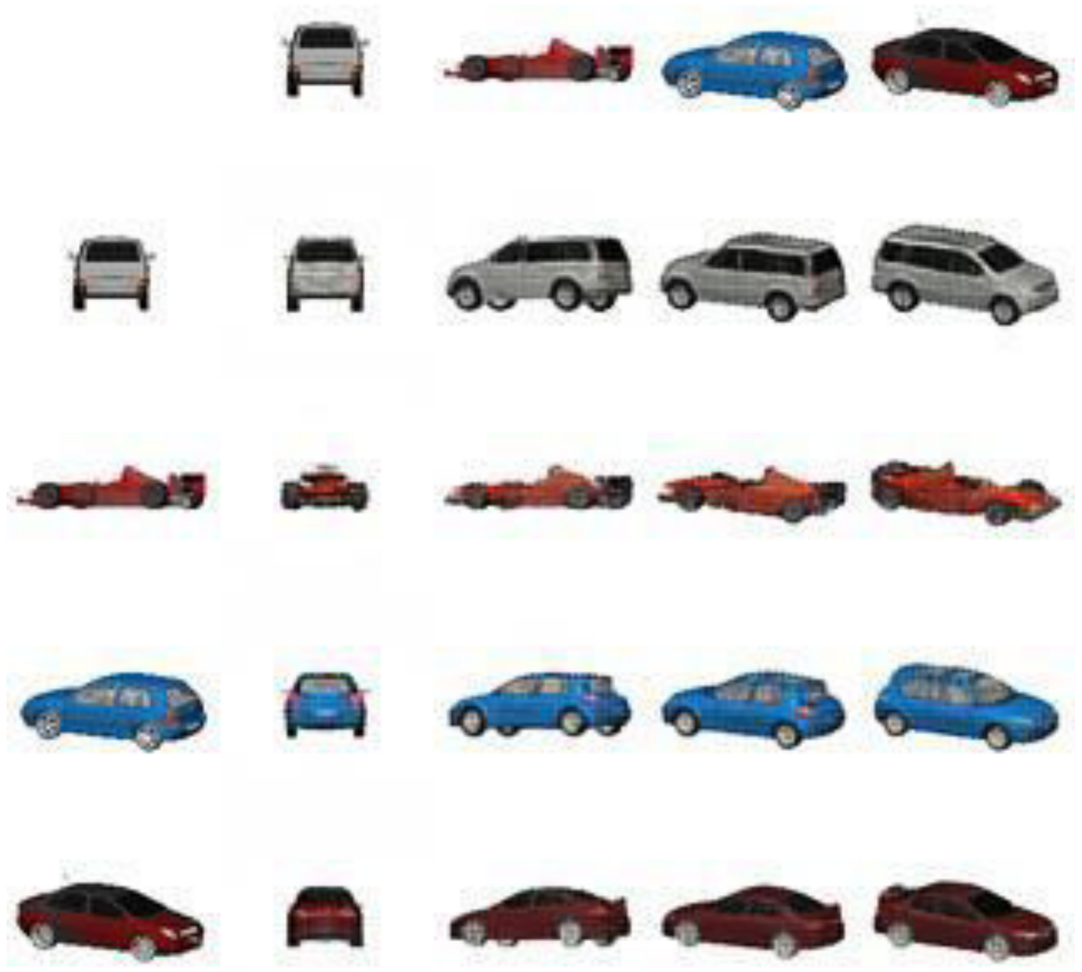
Week 4: Variational Models

Limitations of AR Models

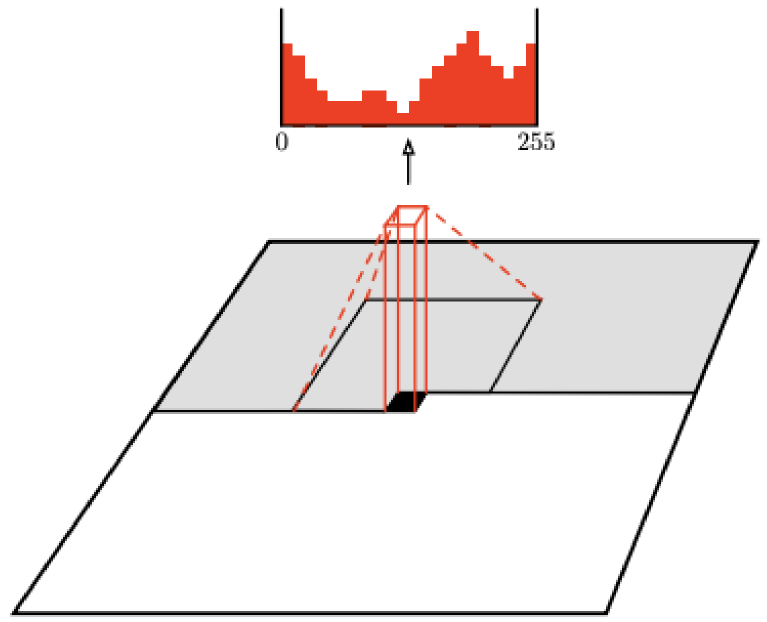
- Last week: autoregressive models
- They work well for not very long sequences e.g., text
- Major limitations:
  - O(n): Inference runtime *linear* in sequence length
  - Require some ordering
  - Cannot take advantage of low rank (next slide)

Illustration - Low-Rank Data

- Let's look at Cars3D
- Synthetic and very simple
- Every image specified by only three factors:
  - Model – [1…M]
  - Azimuth – [0-360]
  - Elevation – [-90,90]



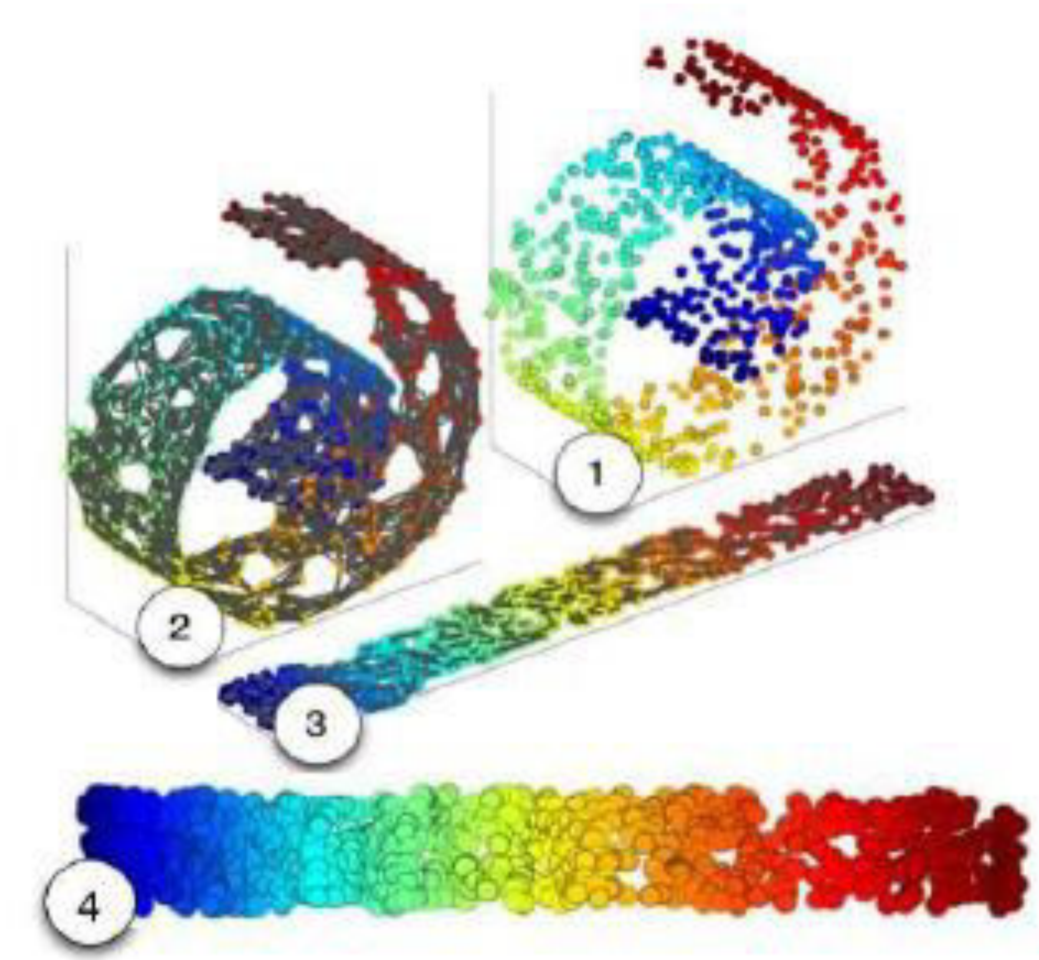
AR Models for Cars3D



- Decide on some ordering of the pixels
- Train a model to predict $p(x_{n+1}|x_1,x_2,\ldots,x_n)$
- Not the best fit:
  - No natural orderin
  - There are 64X64X3 pixels, so requires as many NN evaluations
  - Approach too complex for such a simple dataset
- Still, many well cited papers do this with massive neural nets:
  - PixelCNN/PixelRNN (Google), ImageGPT (OpenAI)

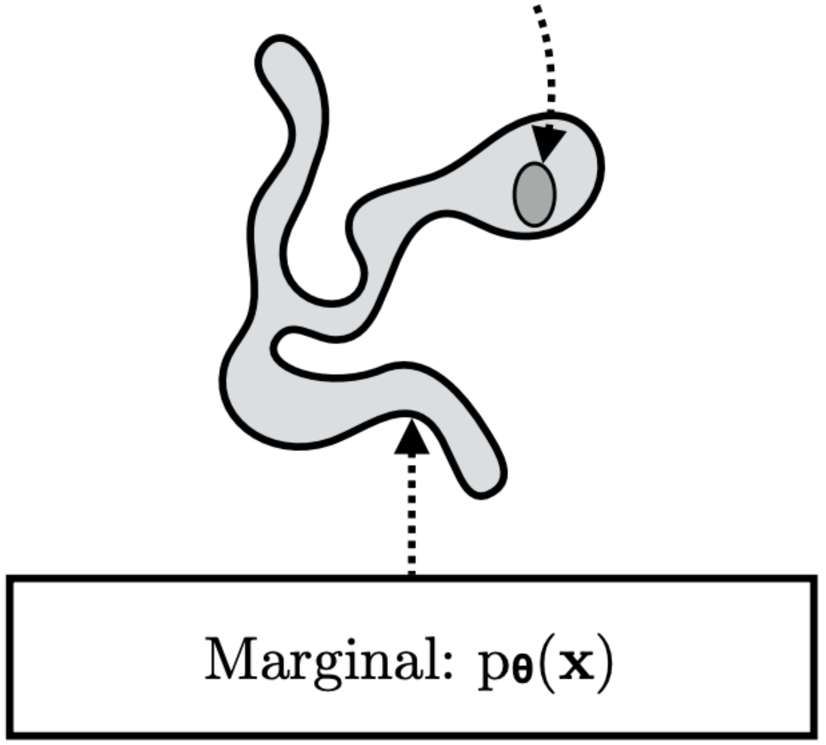
Low-Rank Manifold Assumption

- Generally, the data we observe in the world is of high dimension
- Often generated by a small number of factors of variation
- When the factors are unknown, they are called *hidden* or *latent*



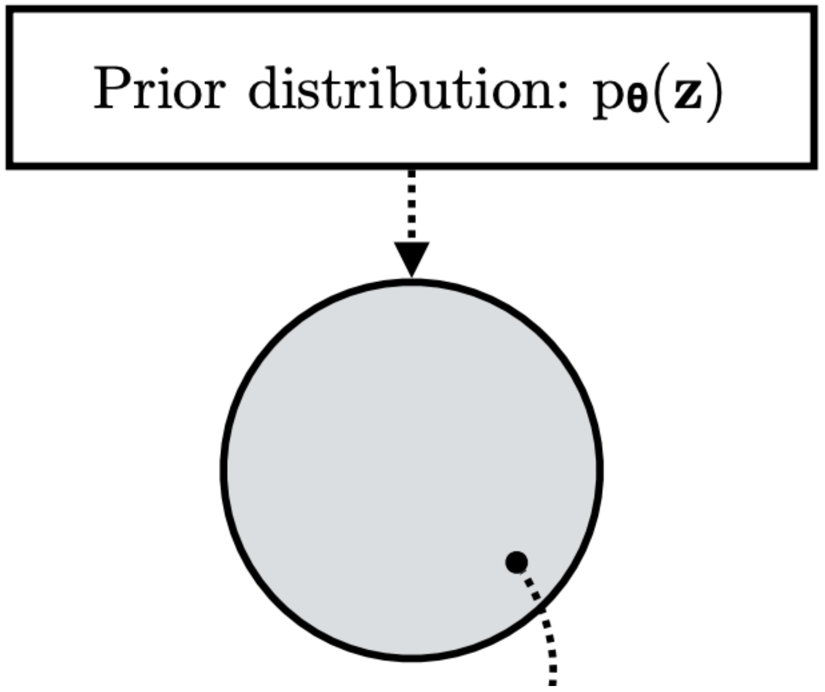
Generative Model Objective

- Assume we have a set of training data (e.g. images) $x_1.x_2..x_N$
- We want to learn a function p(x) that:
  - The train/test images have a high likelihood (p(x))
  - Easy to sample from it
  - Can use it to compute point estimates
- Same as last week



Latent Variable Representation

- A simple model for this data
- Data represented by a hidden (latent) variable *z* of (typically) low-dimension
- Latent variable has a simple distribution p(z)
  - E.g. Uniform, Gaussian, Multinomial
- Code p(z) is a compact representation of p(x)



Example: Car3D Prior

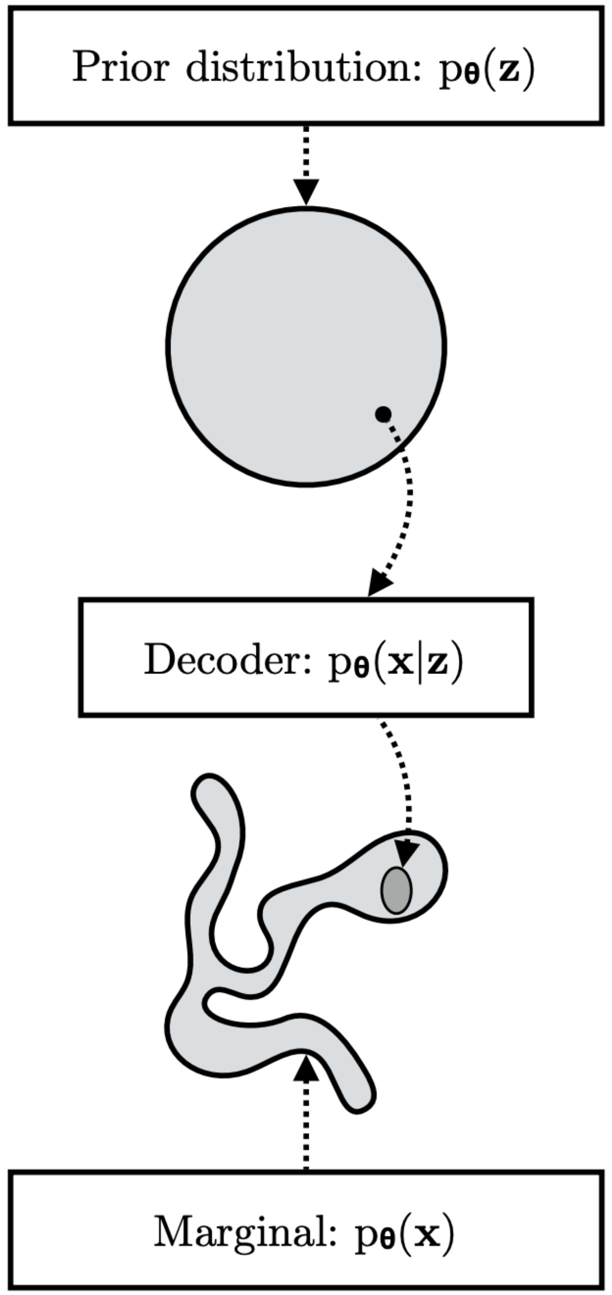
- In Cars3D we can choose a prior term over three hidden factors
- The model may choose to represent: z_1=type, z_2=pose, z_3=azimuth
- The z factors can be a complex combination of the factors though
- We may choose a uniform distribution over each factor

$$p(z) = \prod_{f=1}^{3} U(z_f)$$

Generator $G_\theta(z)$

- A latent variable z is mapped into a distribution over images x using function G
- Intuitively, G(z) is the most likely image given latent code z
- The images are typically of much higher dimension than z
- The function can be complex e.g. transformer

$p_\theta(\mathbf{x},\mathbf{z}) = p_\theta(\mathbf{z})\ p_\theta(\mathbf{x}|\mathbf{z})$

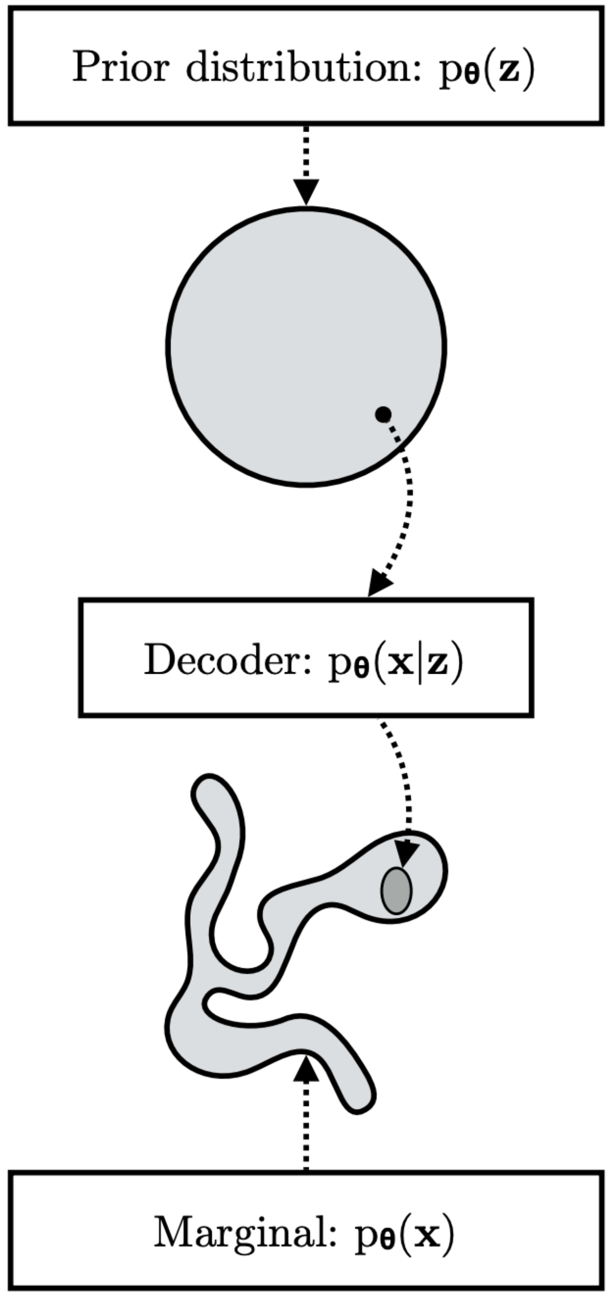

Prior distribution: $p_\theta(\mathbf{z})$
Decoder: $p_\theta(\mathbf{x}|\mathbf{z})$
Marginal: $p_\theta(\mathbf{x})$

Posterior $p_\theta(x|z)$

- Our objective is to have a probabilistic model
- We model the probability of image x given code z
- Here we use a simple Gaussian PDF

$$p_\theta(x|z) = \frac{1}{Z} e^{-\frac{|x - G_\theta(z)|^2}{2\sigma_p^2}}$$

$$p_\theta(x,z) = p_\theta(z)\ p_\theta(x|z)$$

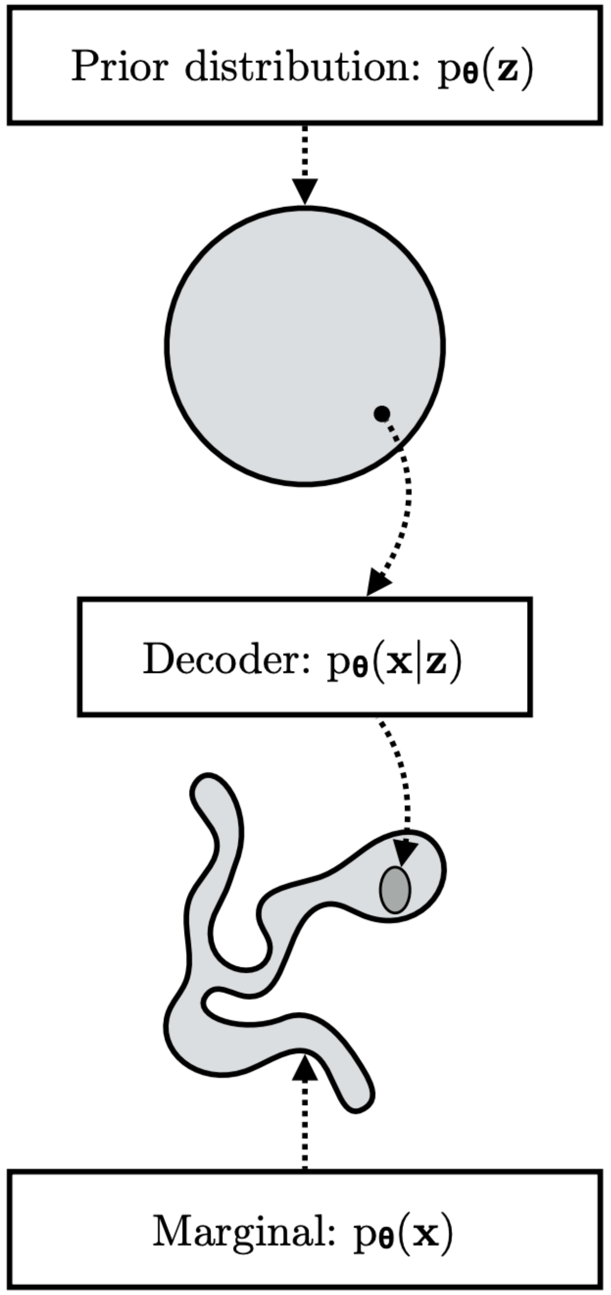


Likelihood $p_\theta(x)$

- The distribution of x under of model
- A function of the prior and posterior

$$p_\theta(x) = \sum_z p_\theta(x, z) = \sum_z p_\theta(x|z) p_\theta(z)$$

$\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) = \mathrm{p}_{\boldsymbol{\theta}}(\mathbf{z})\ \mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$

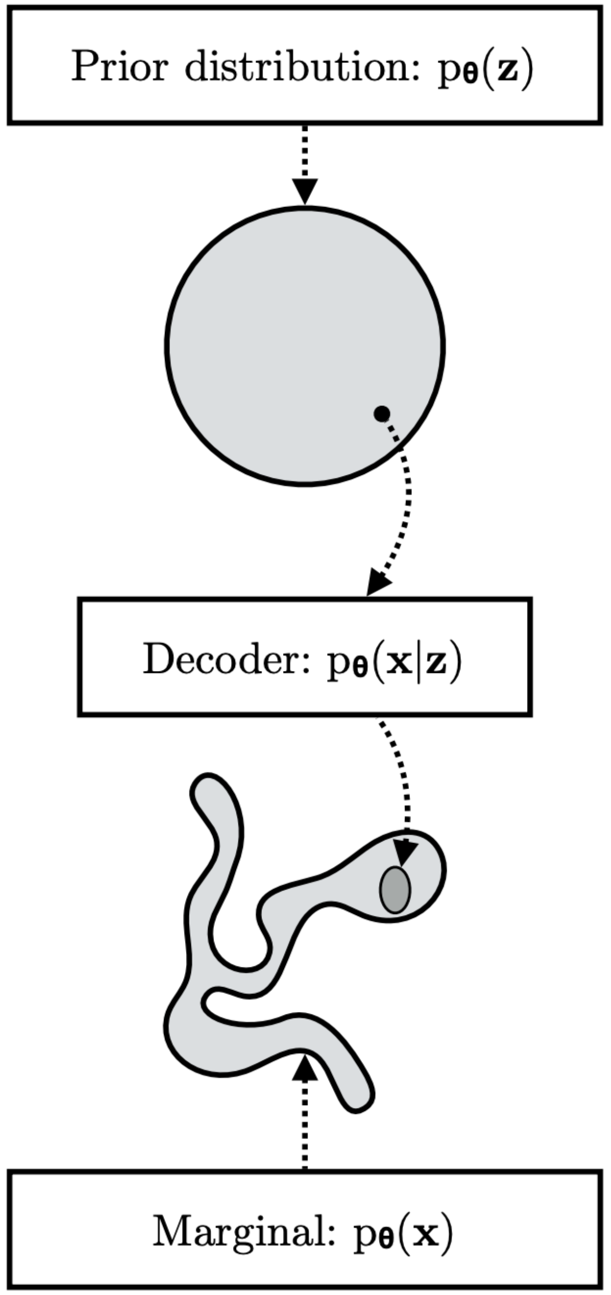

Prior distribution: $\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{z})$
Decoder: $\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
Marginal: $\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x})$

$$p_\theta(\mathbf{x},\mathbf{z}) = p_\theta(\mathbf{z})\; p_\theta(\mathbf{x}|\mathbf{z})$$

Two Challenges

- Efficiently compute the value of $p_\theta(x)$ provided x
- Find the optimal parameters such the data is most likely under the model (max-likelihood)

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


Prior distribution: $p_\theta(\mathbf{z})$
Decoder: $p_\theta(\mathbf{x}|\mathbf{z})$
Marginal: $p_\theta(\mathbf{x})$

Preliminary: Importance Sampling

- Assume that you wanted to estimate the following:

$$\bar{f} = \sum_x p(x) f(x)$$

- When f is a function and p is a PDF
- This can also be written:

$$\bar{f} = E_{x \sim p(x)} f(x)$$

Monte Carlo Estimation

- Computing the integral might be hard $\bar{f} = E_{x\sim p(x)} f(x)$

- Instead, we can perform a sampling-based approach
  - Sample N values from p(x): x1,x2..xN
  - Estimate the average value of f on samples from p

$$\tilde{f} = \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

Simple Estimation May not Work

- Several problems can occur:
  - We may not know how to sample from p
  - Sampling from p may require too many samples - see whiteboard

Proposal Distribution

- Propose another probability distribution Q(x) such that:
  - It is easy to sample from
  - High probability at high values of p(x)f(x)
- Can we estimate expectation with respect of Q instead of P?
- No! this estimator is **biased**

$$E_{x\sim q}f(x)$$

Unbiased estimator

- Estimation with respect to Q requires importance weighting

$$E_{x\sim p} f(x) = \sum_x p(x) f(x) = \sum_x q(x) \frac{p(x)}{q(x)} f(x) = E_{x\sim q} w(x) f(x)$$

- Where $w(x) = \frac{p(x)}{q(x)}$
- It will have low-variance when $q(x) \propto p(x) f(x)$
- I.e. require fewer samples for a good estimate

Back to Inference

- We want to estimate p(x), we know both p(x|z) and p(z)
- This is a simple expectation

$$p(x) = \sum_x p(x|z)p(z) = E_{z \in p_z} p(x|z)$$

- However $p(x|z) = \begin{cases} \|x - G(z)\| \gg 0 : 0 \\ \|x - G(z)\| \approx 0 : \delta_z \end{cases}$

- Nearly all z we sample will yield 0 p(x|z), too many samples needed

Idea: Importance Sampling for Inference

- Let us propose distribution q(z), and compute expectations

$$p(x) = \sum_x p(z)p(x|z) = \sum_x q(z)\frac{p(z)}{q(z)}p(x|z) = E_{z\sim q}\frac{p(z)}{q(z)}p(x|z)$$

- In practice, we case about log(p), so using Jensen’s inequality:

$$\log(p(x)) = \log(E_{z\sim q}\frac{p(z)}{q(z)}p(x|z)) \geq E_{z\sim q}\log(\frac{p(z)}{q(z)}p(x|z))$$

Simplifying the Expression

- The expression simplifies to:

$$E_{z\sim q}\log(\frac{p(z)}{q(z)}p(x|z)) \;=\; E_{z\sim q}\log(p(x|z)) + E_{z\sim q}\log(\frac{p(z)}{q(z)}) \;=$$
$$E_{z\sim q}\log(p(x|z)) - KL(q||p)$$

- The term is called the Evidence Lower BOund (ELBO)
- First term is the expectation of p(x|z) with respect to q (instead of p)
- The second term penalizes large difference between p and q

What is the Best Prosposal Distribution

- How to select the best proposall distribution Q for image x?
- Idea: optimize it!

$$p(x) \geq E_{z\sim q}\log(p(x|z)) - KL(q||p)$$

- It follows that: $p(x) \geq max_q E_{z\sim q}\log(p(x|z)) - KL(q||p)$

- In fact , if we are free to select any q, equality follows

What is the Theoretically Optimal Q?

- Let us substitute q(z) = p(z|x) - also called the posterior

$$log(p(x))) \geq log(\frac{p(x|z)p(z)}{q(z)}) = log(\frac{p(x,z)}{q(z)}) = log(\frac{p(x,z)}{p(z|x)}) = log(p(x))$$

- There is therefore no proposal better than the posterior
- However, it is typically infeasible to compute
- Direct optimization of q(z) is therefore the practical path

Variational Approximation in Practice

- In practice, we want q(z) we be easy to sample from
- Let us choose a Gaussian
  - Independent dimension
  - Each dimension d has $q(z_d) = \mathcal{N}(\mu_d, \sigma_d^2))$
- Optimization = finding the optimal set of. μ_d, σ_d for all d

Estimation of Likelihood of Entire Dataset

- We may want to estimate likelihood of multiple images $x_1..x_N$
- For each image x_i, we find optimal proposal distribution q_i
- Estimate log likelihood via:

$$log(p(x_i)) \geq E_{z\sim q_i} \log(p(x_i|z)) - KL(q_i|p)$$

- Log-likelihood of entire dataset becomes:

$$log(p(D)) = \sum_i log(p(x_i)) \geq \max_{q1..q_N} \sum_i E_{z\sim q_i} \log(p(x_i|z)) - \sum_i KL(q_i|p)$$

Wait, But Where did P(x|z) Come From?

- In the previous discussion, we assume we know P(x|z)
- In fact, we need to learn p(x|z)
- Remember that: $p_\theta(x|z) = \frac{1}{Z} e^{-\frac{|x - G_\theta(z)|^2}{2\sigma_p^2}}$

- So this means we need to learn the generator function G

Maximum-Likelihood (ML) Estimation of G

- We seek generator G under which the observed data is most likely
- We therefore jointly optimize the parameters of G and of $q_1..q_N$

$$max_G \log(p(D)) \geq max_{G,q_1..q_N} \sum_i E_{z \sim q_i} \log(p(x_i|z)) - KL(q_i|p)$$

- Where optimizing over qi is optimizing over the mean and covariance

Re-Parameterization Trick

- Optimizing over $E_{z \sim q_i} \log(p(x_i|z))$ sounds poorly defined
- This is because the RV z depends on the optimization parameters
- We solve this by rewriting $z = \mu + \sigma \odot \epsilon \quad \epsilon \sim \mathcal{N}(0, I)$

$$E_{\epsilon \sim N(0,I)} p_G(x|\mu + \sigma \odot \epsilon)$$

- Expectation is over random samples from the normal distribution
- Optimization is over the deterministic parameters

Latent Optimization

- The formalism requires optimization of mean, var for every sample x_i
- Cumbersome for large training sets
- Might cause overfitting

Amortized Inference

- Breakthrough work of Kingma and Welling
- Proposed to replace the per-sample q_i by a learned encoder

$$(\mu_i, \sigma_i) = S(x_i)$$

- The formalism is called the variational autoencoder (VAE)

$$max_{G,S} \log(p(D)) \geq max_{G,S} \sum_i E_{z \sim \mathcal{N}(S(x_i))} \log(p(x_i|z)) - KL(\mathcal{N}(S(x_i))|p(z))$$

VAE Optimization in Practice

- To simply the maths, we often:
  - Choose p(z) = N(0,I)
  - Estimate the expectation using a single normal sample
- The optimization becomes:

$$min_{S,G} \sum_x \|x - G(\mu_x + \sigma_x \odot \epsilon)\|^2 + KL(\mathcal{N}(\mu_x, \sigma_x^2)||\mathcal{N}(0, I))$$

- Where the KL can be evaluated precisely:

$$KL(\mathcal{N}(\mu, \sigma^2)||\mathcal{N}(0, I)) = \mu^2 + \sigma^2 - log(\sigma) - 1$$

Recap: Amortized Variational Inference

- To find the best generator G, we need to estimate data likelihood
- Simply taking expectation over z is too inefficient
- Learn the best proposal distribution q for each x
- Jointly optimize of G and q for all training set
- Instead of optimizing q for each x, train an encoder

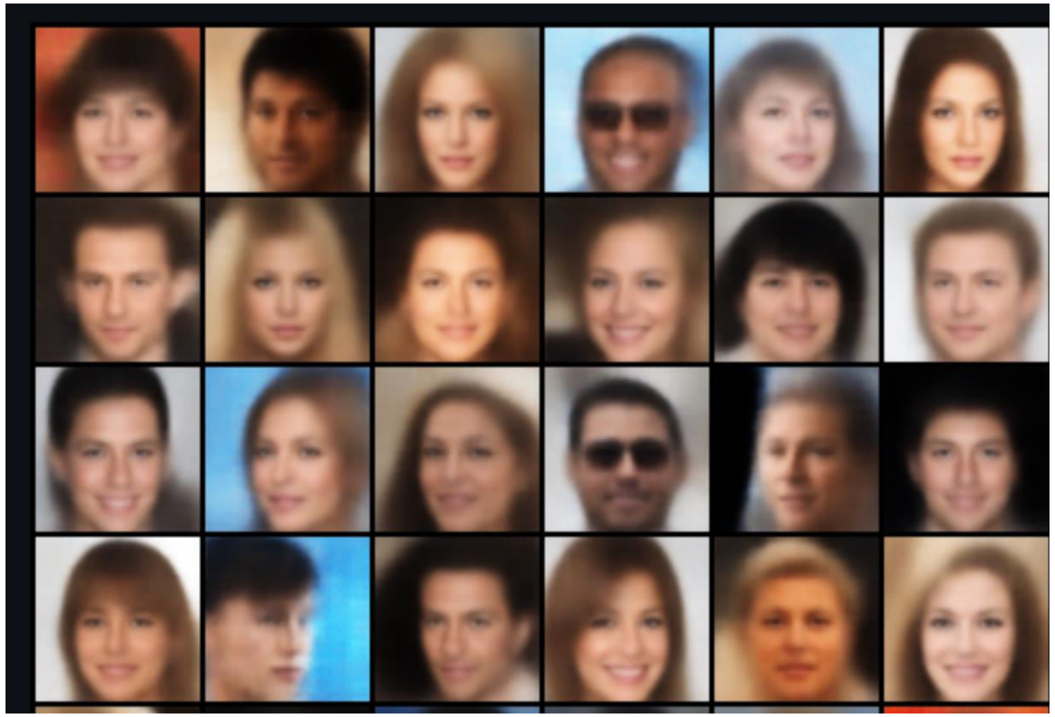
Code

- Latent optimization (GLO): https://github.com/yedidh/glann/blob/master/glo.py
- Amortized inference (VAE):

https://github.com/AntixK/PyTorch-VAE/blob/master/models/vanilla_vae.py

VAE Samples

- While VAE is theoretically sound, the results are blurry



Using VAE for Point Estimation

- VAE can be use for computing p(x)
- We do not have to make the variational approximation

$$p(x) = E_{z \sim q_x} p(x|z) \frac{p(z)}{q_x(z)}$$

- About 100 samples of z are recommended for good accuracy

Conditional VAE

- Let’s get more practice with variational inference
- Assume that every sample x is also conditional on c

A Few Manipulations

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

- Choosing proposal q(x|c,z):

$$p(x,c) = E_{z\sim q(z|x,c)} \frac{p(z|c)}{q(z|x,c)} p(x|c,z)p(c)$$

- With the variational approximation, the conditional ELBO is:

$$\log(p(x,c)) = E_{z\sim q(z|x,c)} \log(p(x|c,z)) + \log(\frac{p(z|c)}{q(z|x,c)}) + \log(p(c))$$

cVAE Loss

- Simplifying, this is the cVAE loss:

$$L(S,G) = \sum_{(x,c)} E_{z\sim q_{x,c}} \|x - G(c,\, z)\|^2 + KL(q_{x,c}||p(z|c)) + \log(p(c))$$

- It is also possible to do this with latent optimization (LORD, week 10)

Summary

- Some datasets are well described by low-rank latent variables
- Variational inference allows likelihood calculation
- Can be used for training better generators of the data
- VAE proposed to amortize the proposal distributions
- Conditional inference has a very similar form