Combining Generative Model Families

Volodymyr Kuleshov

Cornell Tech

Lecture 12

Announcements

- All the presentation slots should be filled!
    - If you haven't booked a presentation slot or I haven't reached out to you, let me know!

Lecture Outline

1. Finish Energy-Based Models
   - Likelihood-based learning
   - Markov Chain Monte Carlo
   - (Persistent) Contrastive Divergence
2. Combining Model Families
   - Autoregressive models + VAEs: PixelVAE
   - Autoregressive models + Flows: Autoregressive flows
   - Flows + VAEs: Flow-based posteriors
   - Multi-modal VAEs
   - VAEs + RNNs: Variational RNNs
   - Flows + GANs: FlowGAN
   - GANs + VAEs: Adversarial Autoencoders
   - GANs + VAEs: InfoGAN, InfoVAE, β-VAE

Energy based models: learning and inference

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

Pros:

1. can plug in pretty much any function $f_\theta(\mathbf{x})$ you want

Cons (lots of them):

1. Sampling is hard
2. Evaluating likelihood (learning) is hard
3. Feature learning is even harder

Curse of dimensionality: The fundamental issue is that computing $Z(\theta)$ numerically (when no analytic solution is available) scales exponentially in the number of dimensions of $\mathbf{x}$.

Can we still learn p? Yes! (but it will not be as fast)

Exponential families: moment matching

Consider an exponential family model

$$p(x;\theta) = \frac{\exp(\theta^T f(x))}{Z(\theta)}$$

whose log-likelihood equals

$$\frac{1}{|D|}\log p(D;\theta) = \frac{1}{|D|}\sum_{x\in D}\theta^T f(x) - \log Z(\theta).$$

Taking the gradient, and using our expression for the gradient of the partition function, we obtain the expression

$$\nabla_\theta \frac{1}{|D|}\log p(D;\theta) = \frac{1}{|D|}\sum_{x\in D} f(x) - \mathbb{E}_{x\sim p}[f(x)]$$

This is the difference between the expectations of the natural parameters under the empirical (i.e. data) and the model distribution.

Approximate learning techniques in ExpFams

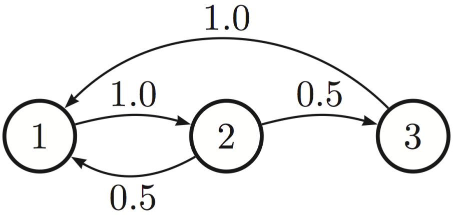
To compute gradients, we need to sample from the model. But this is hard!

We will look at two approximate methods:

1. **MCMC sampling** from the distribution at each step of gradient descent; we then approximate the gradient using Monte-Carlo.
2. **(Persistent) contrastive divergence**, a variant of MCMC sampling which re-uses the same Markov Chain between iterations.

Markov Chains: Definition

- A (discrete-time) Markov chain is a sequence of random variables $S_0, S_1, S_2, \ldots$ with $S_i \in \{1, 2, \ldots, d\}$, intuitively representing the state of a system.
- The initial state is distributed according to a probability $P(S_0)$
- All subsequent states are generated from $P(S_i \mid S_{i-1})$ that depends only on the previous random state.



Markov assumption: the probability $P(S_i \mid S_{i-1})$ is the same at every step i. The transition probabilities in the entire process depend only on the given state and not on how we got there.

Markov Chains: Stationary Distribution

If the initial state S_0 is drawn from a vector probabilities p_0, we may represent the probability p_t of ending up in each state after t steps as

$$p_t = T^t p_0 \qquad T \in \mathbb{R}^{d \times d} \text{and} \, T_{ij} = P(S_{\text{new}} = i \mid S_{\text{prev}} = j).$$

The limit $\pi = \lim_{t\to\infty} p_t$ (when it exists) is called a stationary distribution of the Markov chain. It's an eigenvector of T.

A sufficent condition for a stationary distribution is called detailed balance:

$$\pi(x')T(x \mid x') = \pi(x)T(x' \mid x) \text{ for all } x, x'$$

It is easy to show that such a π must form a stationary distribution. Just sum both sides of the equation over x and simplify:

$$\pi(x') = \sum_x \pi(x)T(x' \mid x) \text{ for all } x \text{ means } \pi \text{ is eigenvector of } T$$

Markov Chain Monte Carlo

The idea of MCMC will be to construct a Markov chain whose states will be joint assignments to the variables in the model and whose stationary distribution will equal the model probability

$$p(x;\theta) = \frac{\exp(\theta^T f(x))}{Z(\theta)}.$$

An MCMC algorithm defines a transition operator T specifying a Markov chain, an initial variable assignment x_0 and performs the following steps.

1. Run the Markov chain from x_0 for B burn-in steps.
2. Run the Markov chain for N sampling steps and collect all the states that it visits.

Assuming B is sufficiently large, the latter collection of states will form samples from p. We may then use these samples for Monte Carlo integration (or in importance sampling).

Constructing MCMC chains with Metropolis-Hastings

The MH method constructs a transition operator $T(x' \mid x)$ from two components:

- A transition kernel $Q(x' \mid x)$, specified by the user (something simple, like $x +$ noise)
- An acceptance probability for moves proposed by Q, specified by the algorithm as

$$A(x' \mid x) = \min\left(1, \frac{P(x')Q(x \mid x')}{P(x)Q(x' \mid x)}\right).$$

  - Encourages us to move towards more likely points in the distribution (imagine for example that Q is uniform)
  - When Q suggests a move to a low-probability region, we do that a certain fraction of the time.

At each step of the Markov chain, we choose a new point x' according to Q. Then, we either accept this proposed change (with probability α), or with probability $1 - \alpha$ we remain at our current state.

Proof of Metropolis-Hastings method

Given any Q the MH algorithm will ensure that P will be a stationary distribution of the resulting Markov Chain. More precisely, P will satisfy the detailed balance condition with respect to the MH Markov chain.

To see that, first observe that if $A(x' \mid x) < 1$, then $\frac{P(x)Q(x'|x)}{P(x')Q(x|x')} > 1$ and thus $A(x \mid x') = 1$. When $A(x' \mid x) < 1$, this lets us write:

$$\begin{aligned} A(x' \mid x) &= \frac{P(x')Q(x \mid x')}{P(x)Q(x' \mid x)} \\ P(x')Q(x \mid x')A(x \mid x') &= P(x)Q(x' \mid x)A(x' \mid x) \\ P(x')T(x \mid x') &= P(x)T(x' \mid x), \end{aligned}$$

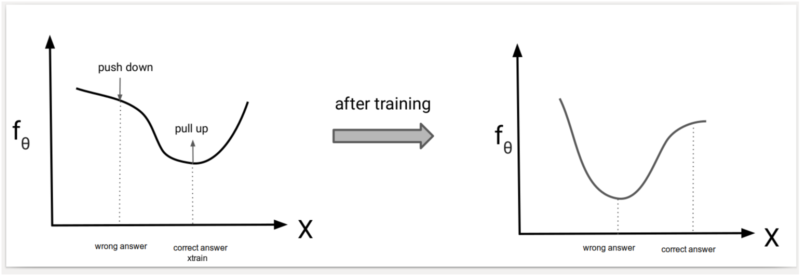
which is simply the detailed balance condition. $T(x \mid x')$ is full transition operator of MH obtained by applying both Q and A.

Sampling from Energy based models

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

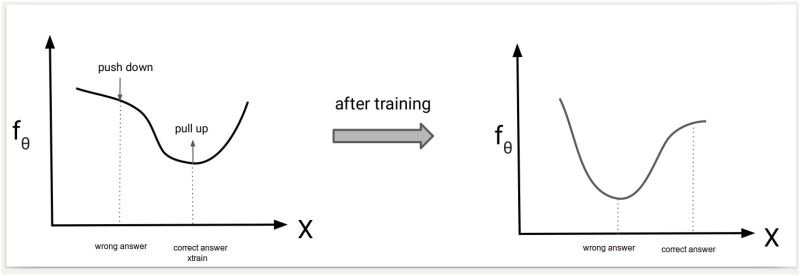
- No direct way to sample like in autoregressive or flow models. Main issue: cannot easily compute how likely each possible sample is
- However, we can easily compare two samples $\mathbf{x}, \mathbf{x}'$.
- Use iterative approach based on Metropolis-Hastings MCMC:
  1. Initialize x^0 randomly, $t = 0$
  2. Let $x' = x^t + \text{noise}$
     1. If $f_\theta(x') > f_\theta(x^t)$, let $x^{t+1} = x'$
     2. Else let $x^{t+1} = x'$ with probability $\exp(f_\theta(x') - f_\theta(x^t))$
  3. Go to step 2
- Works in theory, but can take a very long time to converge

(Persistent) Contrastive Divergence



- Goal: maximize $\frac{f_\theta(x_{train})}{Z(\theta)}$
- **Idea**: Instead of evaluating $Z(\theta)$ exactly, use a Monte Carlo estimate.
- **Contrastive divergence algorithm**: sample $x_{sample} \sim p_\theta$ with MCMCM, take step on $\nabla_\theta \left(f_\theta(x_{train}) - f_\theta(x_{sample})\right)$. Make training data more likely than typical sample from the model. Recall comparisons are easy in energy based models!
- **Persistent** CD: reuse the Markov chain across SGD steps

Training intuition



- Goal: maximize $\frac{f_\theta(x_{train})}{Z(\theta)}$. Increase numerator, decrease denominator.
- **Intuition**: because the model is not normalized, increasing the un-normalized probability $f_\theta(x_{train})$ by changing θ does **not** guarantees that x_{train} becomes relatively more likely (compared to the rest).
- We also need to take into account the effect on other "wrong points" and try to "push them down" to *also* make $Z(\theta)$ small.

Energy based models: pros and cons

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

Pros:

1. Can plug in pretty much any function $f_\theta(\mathbf{x})$ you want
2. Can be combined with other model families
3. Can be combined with ideas from graphical models

Cons:

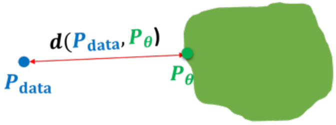
1. Sampling is hard
2. Evaluating likelihood (learning) is hard
3. Feature learning is even harder

Lecture Outline

1. Finish Energy-Based Models
   - Likelihood-based learning
   - Markov Chain Monte Carlo
   - (Persistent) Contrastive Divergence
2. Combining Model Families
   - Autoregressive models + VAEs: PixelVAE
   - Autoregressive models + Flows: Autoregressive flows
   - Flows + VAEs: Flow-based posteriors
   - Multi-modal VAEs
   - VAEs + RNNs: Variational RNNs
   - Flows + GANs: FlowGAN
   - GANs + VAEs: Adversarial Autoencoders
   - GANs + VAEs: InfoGAN, InfoVAE, β-VAE

Summary

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \dots, n$



$\theta \in M$

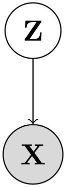
Model family

Story so far

- Representation: Latent variable vs. fully observed
- Objective function and optimization algorithm: Many divergences and distances optimized via likelihood-free (two sample test) or likelihood based methods
- Each have Pros and Cons

Plan for today: Combining models

Variational Autoencoder



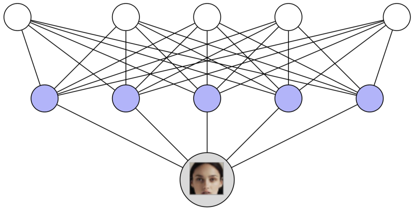
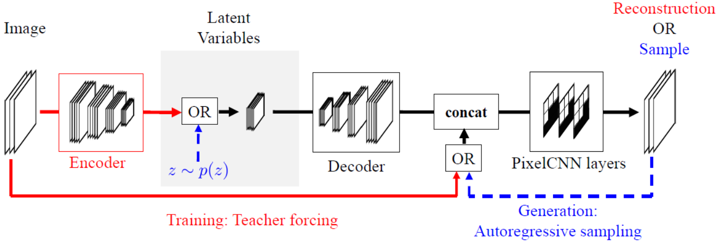


Image x

A mixture of an infinite number of Gaussians:

1. $\mathbf{z} \sim \mathcal{N}(0, I)$
2. $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})\right)$ where μ_θ,Σ_θ are neural networks
3. $p(\mathbf{x} \mid \mathbf{z})$ and $p(\mathbf{z})$ usually simple, e.g., Gaussians or conditionally independent Bernoulli vars (i.e., pixel values chosen independently given $\mathbf{z}$)
4. **Idea**: increase complexity using an autoregressive model

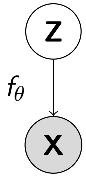
PixelVAE (Gulrajani et al.,2017)



Gulrajani et. al, 2017

- $\mathbf{z}$ is a feature map with the same resolution as the image $\mathbf{x}$
- Autoregressive structure: $p(\mathbf{x} \mid \mathbf{z}) = \prod_i p(x_i \mid x_1, \cdots, x_{i-1}, \mathbf{z})$
  - $p(\mathbf{x} \mid \mathbf{z})$ is a PixelCNN
  - Prior $p(\mathbf{z})$ can also be autoregressive
  - Can be hierarchical: $p(\mathbf{x} \mid \mathbf{z}_1)p(\mathbf{z}_1 \mid \mathbf{z}_2)$
- State-of-the art log-likelihood on some datasets; learns features (unlike PixelCNN); computationally cheaper than PixelCNN (shallower)

Autoregressive flow

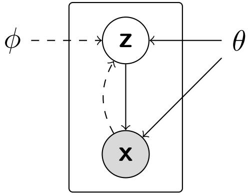


- Flow model, the marginal likelihood $p(\mathbf{x})$ is given by

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

  where $p_Z(\mathbf{z})$ is typically simple (e.g., a Gaussian). More complex prior?
- Prior $p_Z(\mathbf{z})$ can be autoregressive $p_Z(\mathbf{z}) = \prod_i p(z_i \mid z_1, \cdots, z_{i-1})$.
- Autoregressive models are related to flows.Just another MAF layer.
- See also neural autoregressive flows (Huang et al., ICML-18)

VAE + Flow Model

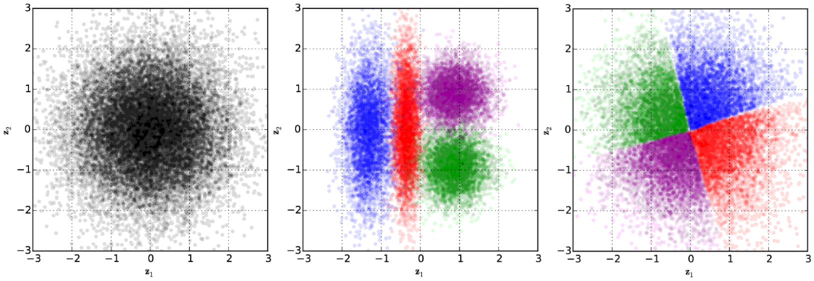


$$\log p(\mathbf{x};\theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x};\phi)\log p(\mathbf{z},\mathbf{x};\theta) + H(q(\mathbf{z}|\mathbf{x};\phi)) = \underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}}$$

$$\log p(\mathbf{x};\theta) = \mathcal{L}(\mathbf{x};\theta,\phi) + \underbrace{D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))}_{\text{Gap between true log-likelihood and ELBO}}$$

- $q(\mathbf{z}|\mathbf{x};\phi)$ is often too simple (Gaussian) compared to the true posterior $p(\mathbf{z}|\mathbf{x};\theta)$, hence ELBO bound is loose
- **Idea:** Make posterior more flexible: $\mathbf{z}' \sim q(\mathbf{z}'|\mathbf{x};\phi)$, $\mathbf{z} = f_{\phi'}(\mathbf{z}')$ for an invertible $f_{\phi'}$ (Rezende and Mohamed, 2015; Kingma et al., 2016)
- Still easy to sample from, and can evaluate density.

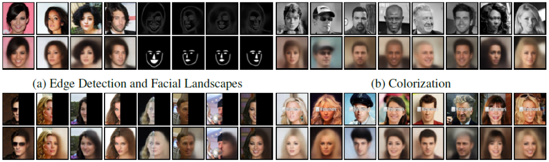
VAE + Flow Model



(a) Prior distribution (b) Posteriors in standard VAE (c) Posteriors in VAE with IAF

Posterior approximation is more flexible, hence we can get tighter ELBO (closer to true log-likelihood).

Multimodal variants



Wu and Goodman, 2018

- **Goal:** Learn a joint distribution over the two domains $p(x_1, x_2)$, e.g., color and gray-scale images Can use a VAE style model:

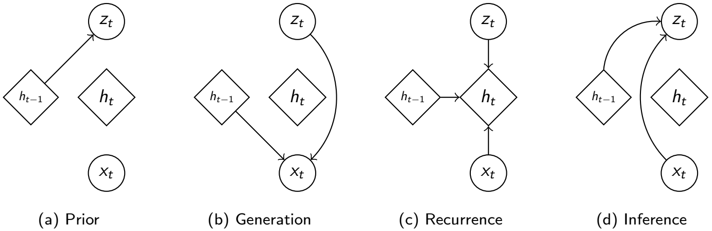
$$\begin{tikzcd} & \mathbf{z} \arrow[dl] \arrow[dr] & \\ \mathbf{x_1} & & \mathbf{x_2} \end{tikzcd}$$

- Learn $p_\theta(x_1, x_2)$, use inference nets $q_\phi(z \mid x_1)$, $q_\phi(z \mid x_2)$, $q_\phi(z \mid x_1, x_2)$.

Variational RNN

- **Goal:** Learn a joint distribution over a sequence $p(x_1, \cdots, x_T)$
- VAE for sequential data, using latent variables $z_1, \cdots, z_T$. Instead of training separate VAEs $z_i \rightarrow x_i$, train a joint model:

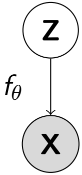
$$p(x_{\leq T}, z_{\leq T}) = \prod_{t=1}^{T} p(x_t \mid z_{\leq t}, x_{<t}) p(z_t \mid z_{<t}, x_{<t})$$



Chung et al, 2016

- Use RNNs to model the conditionals (similar to PixelRNN)
- Use RNNs for inference $q(z_{\leq T}|x_{\leq T}) = \prod_{t=1}^{T} q(z_t \mid z_{<t}, x_{\leq t})$
- Train like VAE to maximize ELBO. Conceptually similar to PixelVAE.

Combining losses

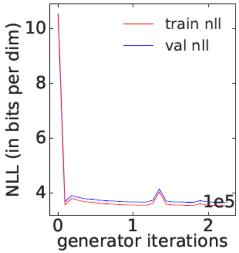


- Flow model, the marginal likelihood $p(\mathbf{x})$ is given by

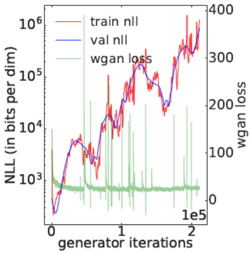
$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

- Can also be thought of as the generator of a GAN
- Should we train by $\min_\theta D_{KL}(p_{data}, p_\theta)$ or $\min_\theta JSD(p_{data}, p_\theta)$?

FlowGAN



(a) MLE



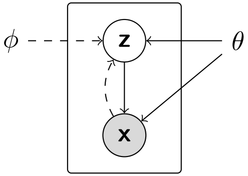
(b) ADV

Although $D_{KL}(p_{data}, p_\theta) = 0$ if and only if $JSD(p_{data}, p_\theta) = 0$, optimizing one does not necessarily optimize the other. If $\mathbf{z}, \mathbf{x}$ have same dimensions, can optimize $\min_\theta KL(p_{data}, p_\theta) + \lambda JSD(p_{data}, p_\theta)$

Objective	Inception Score	Test NLL (in bits/dim)
MLE	2.92	**3.54**
ADV	**5.76**	8.53
Hybrid ($\lambda = 1$)	3.90	4.21

Interpolates between a GAN and a flow model

Adversarial Autoencoder (VAE + GAN)



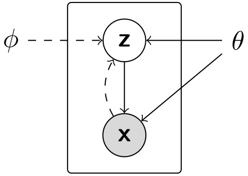
$$\log p(\mathbf{x};\theta) = \underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}} + D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))$$

$$\underbrace{E_{\mathbf{x}\sim p_{data}}[\mathcal{L}(\mathbf{x};\theta,\phi)]}_{\approx\text{training obj.}} = E_{\mathbf{x}\sim p_{data}}[\log p(\mathbf{x};\theta) - D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))]$$

$$\overset{\text{up to const.}}{\equiv} -\underbrace{D_{KL}(p_{data}(\mathbf{x})\|p(\mathbf{x};\theta))}_{\text{equiv. to MLE}} - E_{\mathbf{x}\sim p_{data}}[D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))]$$

- Note: regularized maximum likelihood estimation (Shu et al, *Amortized inference regularization*)
- Can add in a GAN objective $-JSD(p_{data}, p(\mathbf{x};\theta))$ to get sharper samples, i.e., discriminator attempting to distinguish VAE samples from real ones.

An alternative interpretation



$$\underbrace{E_{\mathbf{x}\sim p_{data}}[\mathcal{L}(\mathbf{x};\theta,\phi)]}_{\approx\text{training obj.}} = E_{\mathbf{x}\sim p_{data}}\left[\log p(\mathbf{x};\theta) - D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))\right]$$

$$\overset{\text{up to const.}}{\equiv} -D_{KL}(p_{data}(\mathbf{x})\|p(\mathbf{x};\theta)) - E_{\mathbf{x}\sim p_{data}}\left[D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))\right]$$

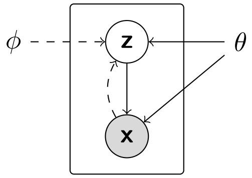
$$= -\sum_{\mathbf{x}} p_{data}(\mathbf{x})\left(\log\frac{p_{data}(\mathbf{x})}{p(\mathbf{x};\theta)} + \sum_{\mathbf{z}} q(\mathbf{z}\mid\mathbf{x};\phi)\log\frac{q(\mathbf{z}\mid\mathbf{x};\phi)}{p(\mathbf{z}|\mathbf{x};\theta)}\right)$$

$$= -\sum_{\mathbf{x}} p_{data}(\mathbf{x})\left(\sum_{\mathbf{z}} q(\mathbf{z}\mid\mathbf{x};\phi)\log\frac{q(\mathbf{z}\mid\mathbf{x};\phi)p_{data}(\mathbf{x})}{p(\mathbf{z}|\mathbf{x};\theta)p(\mathbf{x};\theta)}\right)$$

$$= -\sum_{\mathbf{x},\mathbf{z}} p_{data}(\mathbf{x})q(\mathbf{z}\mid\mathbf{x};\phi)\log\frac{p_{data}(\mathbf{x})q(\mathbf{z}\mid\mathbf{x};\phi)}{p(\mathbf{x};\theta)p(\mathbf{z}|\mathbf{x};\theta)}$$

$$= -D_{KL}(\underbrace{p_{data}(\mathbf{x})q(\mathbf{z}\mid\mathbf{x};\phi)}_{q(\mathbf{z},\mathbf{x};\phi)}\|\underbrace{p(\mathbf{x};\theta)p(\mathbf{z}|\mathbf{x};\theta)}_{p(\mathbf{z},\mathbf{x};\theta)})$$

An alternative interpretation

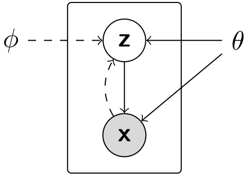


$$E_{\mathbf{x}\sim p_{data}}[\underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}}] \equiv -D_{KL}(\underbrace{p_{data}(\mathbf{x})q(\mathbf{z}\mid\mathbf{x};\phi)}_{q(\mathbf{z},\mathbf{x};\phi)} \,\|\, \underbrace{p(\mathbf{x};\theta)p(\mathbf{z}|\mathbf{x};\theta)}_{p(\mathbf{z},\mathbf{x};\theta)})$$

- Optimizing ELBO is the same as matching the inference distribution $q(\mathbf{z},\mathbf{x};\phi)$ to the generative distribution $p(\mathbf{z},\mathbf{x};\theta) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z};\theta)$
- **Intuition**: $p(\mathbf{x};\theta)p(\mathbf{z}|\mathbf{x};\theta) = p_{data}(\mathbf{x})q(\mathbf{z}\mid\mathbf{x};\phi)$ if
  1. $p_{data}(\mathbf{x}) = p(\mathbf{x};\theta)$
  2. $q(\mathbf{z}\mid\mathbf{x};\phi) = p(\mathbf{z}|\mathbf{x};\theta)$ for all $\mathbf{x}$
  3. Hence we get the VAE objective:
     $-D_{KL}(p_{data}(\mathbf{x})\|p(\mathbf{x};\theta)) - E_{\mathbf{x}\sim p_{data}}[D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))]$
- Many other variants are possible! VAE + GAN:

$$-JSD(p_{data}(\mathbf{x})\|p(\mathbf{x};\theta)) - D_{KL}(p_{data}(\mathbf{x})\|p(\mathbf{x};\theta)) - E_{\mathbf{x}\sim p_{data}}[D_{KL}(q(\mathbf{z}\mid\mathbf{x};\phi)\|p(\mathbf{z}|\mathbf{x};\theta))]$$

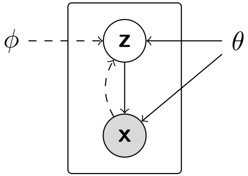
Adversarial Autoencoder (VAE + GAN)



$$E_{\mathbf{x}\sim p_{data}}[\underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}}] \equiv -D_{KL}(\underbrace{p_{data}(\mathbf{x})q(\mathbf{z}\mid\mathbf{x};\phi)}_{q(\mathbf{z},\mathbf{x};\phi)} \| \underbrace{p(\mathbf{x};\theta)p(\mathbf{z}|\mathbf{x};\theta)}_{p(\mathbf{z},\mathbf{x};\theta)})$$

- Optimizing ELBO is the same as matching the inference distribution $q(\mathbf{z},\mathbf{x};\phi)$ to the generative distribution $p(\mathbf{z},\mathbf{x};\theta)$
- **Symmetry:** Using alternative factorization: $p(\mathbf{z})p(\mathbf{x}|\mathbf{z};\theta) = q(\mathbf{z};\phi)q(\mathbf{x}\mid\mathbf{z};\phi)$ if
  1. $q(\mathbf{z};\phi) = p(\mathbf{z})$
  2. $q(\mathbf{x}\mid\mathbf{z};\phi) = p(\mathbf{x}|\mathbf{z};\theta)$ for all $\mathbf{z}$
  3. We get an *equivalent* form of the VAE objective: $-D_{KL}(q(\mathbf{z};\phi)\|p(\mathbf{z})) - E_{\mathbf{z}\sim q(\mathbf{z};\phi)}[D_{KL}(q(\mathbf{x}\mid\mathbf{z};\phi)\|p(\mathbf{x}|\mathbf{z};\theta))]$
- Other variants are possible. For example, can add $-JSD(q(\mathbf{z};\phi)\|p(\mathbf{z}))$ to match features in latent space (Zhao et al., 2017; Makhzani et al, 2018)

Information Preference



$$E_{\mathbf{x}\sim p_{data}}[\underbrace{\mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{ELBO}}] \equiv -D_{KL}(\underbrace{p_{data}(\mathbf{x})q(\mathbf{z}\mid\mathbf{x};\phi)}_{q(\mathbf{z},\mathbf{x};\phi)} \,\|\, \underbrace{p(\mathbf{x};\theta)p(\mathbf{z}|\mathbf{x};\theta)}_{p(\mathbf{z},\mathbf{x};\theta)})$$

- ELBO is optimized as long as $q(\mathbf{z},\mathbf{x};\phi) = p(\mathbf{z},\mathbf{x};\theta)$. Many solutions are possible! For example,
  1. $p(\mathbf{z},\mathbf{x};\theta) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z};\theta) = p(\mathbf{z})p_{data}(\mathbf{x})$
  2. $q(\mathbf{z},\mathbf{x};\phi) = p_{data}(\mathbf{x})q(\mathbf{z}|\mathbf{x};\phi) = p_{data}(\mathbf{x})p(\mathbf{z})$
  3. Note $\mathbf{z}$ and $\mathbf{z}$ are independent. $\mathbf{z}$ carries no information about $\mathbf{x}$. This happens in practice when $p(\mathbf{x}|\mathbf{z};\theta)$ is too flexible, like PixelCNN.
- **Issue:** Many more variables than constraints

InfoGAN

- Explicitly add a mutual information term to the objective

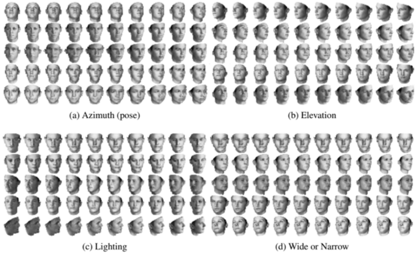
$$-D_{KL}(\underbrace{p_{data}(\mathbf{x})q(\mathbf{z} \mid \mathbf{x};\phi)}_{q(\mathbf{z},\mathbf{x};\phi)} \,\|\, \underbrace{p(\mathbf{x};\theta)p(\mathbf{z}|\mathbf{x};\theta)}_{p(\mathbf{z},\mathbf{x};\theta)}) + \alpha MI(\mathbf{x},\mathbf{z})$$

- MI intuitively measures how far **x** and **z** are from being independent

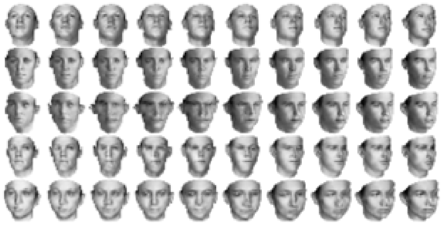
$$MI(\mathbf{x},\mathbf{z}) = D_{KL}\left(p(\mathbf{z},\mathbf{x};\theta) \| p(\mathbf{z})p(\mathbf{x};\theta)\right)$$

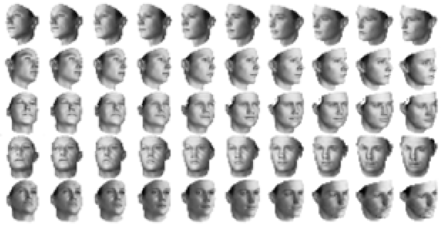
- InfoGAN (Chen et al, 2016) used to learn meaningful (disentangled?) representations of the data

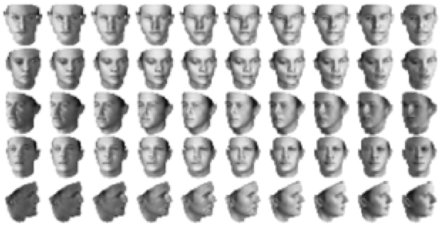
$$MI(\mathbf{x},\mathbf{z}) - E_{\mathbf{x}\sim p_\theta}[D_{KL}(p_\theta(\mathbf{z}|\mathbf{x}) \| q_\phi(\mathbf{z}|\mathbf{x}))] - JSD(p_{data}(\mathbf{x}) \| p_\theta(\mathbf{x}))$$

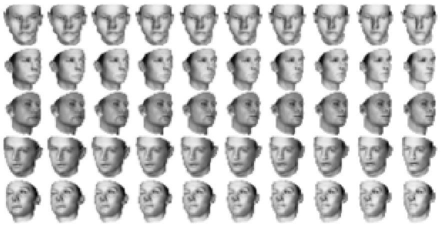


InfoGAN


(a) Azimuth (pose)


(b) Elevation


(c) Lighting


(d) Wide or Narrow

β-VAE

Model proposed to learn disentangled features (Higgins, 2016)

$$-E_{q_\phi(\mathbf{x},\mathbf{z})}[\log p_\theta(\mathbf{x}|\mathbf{z})] + \beta E_{\mathbf{x}\sim p_{data}}\left[D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))\right]$$

It is a VAE with scaled up KL divergence term. This is equivalent (up to constants) to the following objective:

$$(\beta-1)MI(\mathbf{x};\mathbf{z}) + \beta D_{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z})) + E_{q_\phi(\mathbf{z})}[D_{KL}(q_\phi(\mathbf{x}|\mathbf{z})\|p_\theta(\mathbf{x}|\mathbf{z}))]$$

See *The Information Autoencoding Family: A Lagrangian Perspective on Latent Variable Generative Models* for more examples.

Conclusion

- We have covered several useful building blocks: autoregressive, latent variable models, flow models, GANs.
- Can be combined in many ways to achieve different tradeoffs: many of the models we have seen today were published in top ML conferences in the last couple of years
- Lots of room for exploring alternatives in your projects!
- Which one is best? Evaluation is tricky. Still largely empirical