



Carnegie Mellon University
Language Technologies Institute

Deep Generative Models

11-777 Multimodal Machine Learning Fall 2020

Paul Liang

pliang@cs.cmu.edu

 @pliang279

Admin

Matching for midterm presentation

→ Give feedback to relevant teams

Due by Wednesday 10/28 8pm ET

<https://forms.gle/fEQgmk4g6Yfop6Ct5>

Peer-feedback preferences

This form is meant to help create a better matching for the next peer-feedback process. Please let us which teams you would like to give feedback to by specifying your top six teams.

The top ranked team should be the most relevant team for you (but not your own team) and the sixth most relevant team should have rank six.

This form is due by Wednesday 10/28 8pm ET.

Note: Specifying your preferences does not guarantee that you will be asked for feedback for these teams but an integer linear program will do its best to achieve that.

Your email address (**twoertwe@andrew.cmu.edu**) will be recorded when you submit this form. Not you? [Switch account](#)

Rank relevant teams

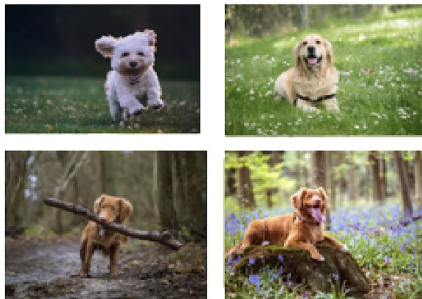
	1	2	3	4	5	6
Team 1: Multimodal sentiment analysis (CMU- MOSEI)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Team 2:						

Used Materials

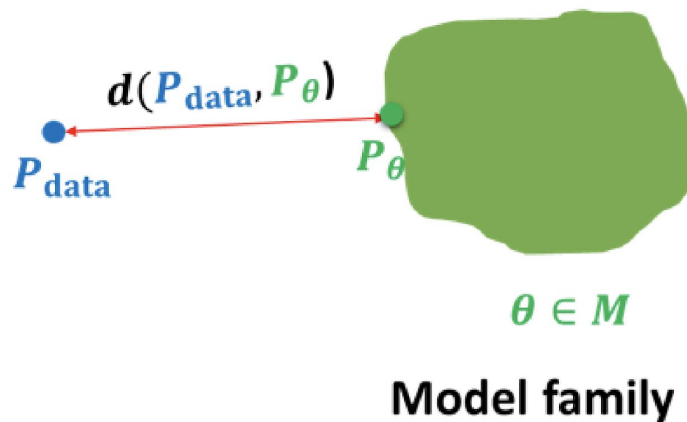
Acknowledgement: Some of the material and slides for this lecture were borrowed from the 10-708 PGM class at CMU taught by Eric Xing with guest lectures by Zhiting Hu, and the DGM class at Stanford taught by Stefano Ermon and Aditya Grover.

Contents

- Generative models
- Variational autoencoders
- Generative adversarial nets



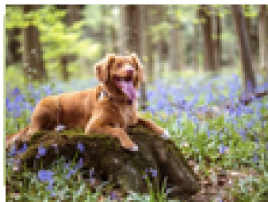
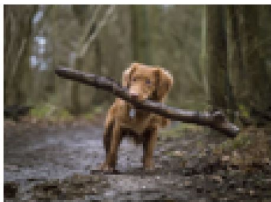
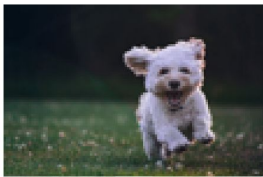
$$x_i \sim P_{\text{data}} \\ i = 1, 2, \dots, n$$



Part 1: Generative models

Learn to model $p(\mathbf{x})$ where x = text, images, videos, multimodal data

- Given x , **evaluate** $p(x)$ - realistic data should have high $p(x)$ and vice versa
- **Sample** new x according to $p(x)$ - sample realistic looking images
- Unsupervised **representation** learning - we should be able to learn what these images have in common, e.g., ears, tail, etc. (features)



INPUT (x)	RECONSTRUCTION (AETR)	RECONSTRUCTION (Gen-RNN)
unable to stop herself, she briefly, gently, touched his hand.	unable to stop herself, she leaned forward, and touched his eyes.	unable to help her , and her back and her into my way.
why didn't you tell me?	why didn't you tell me?	why didn't you tell me?"
a strange glow of sunlight shines down from above, paper white and blinding, with no heat.	the light of the sun was shining through the window, illuminating the room.	a tiny light on the door, and a few inches from behind him out of the door.
he handed her the slip of paper.	he handed her a piece of paper.	he took a sip of his drink.

Part 1: Generative models

Sometimes we also care about $p(x|c)$ - **conditional generation**

- c is a category (e.g. faces, outdoor scenes) from which we want to generate images

We might also care about $p(x_2|x_1,c)$ - **style transfer**

- c is a stylistic change e.g. negative to positive



From negative to positive

consistently slow .
consistently good .
consistently fast .

my goodness it was so gross .
my husband 's steak was phenomenal .
my goodness was so awesome .

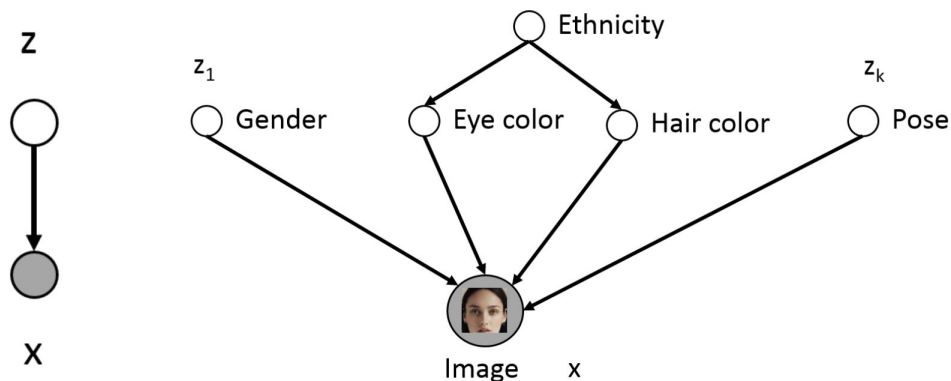
it was super dry and had a weird taste to the entire slice .
it was a great meal and the tacos were very kind of good .
it was super flavorful and had a nice texture of the whole side .

Latent variable models

- Lots of variability in images \mathbf{x} due to gender, eye color, hair color, pose, etc.
- However, unless images are annotated, these factors of variation are not explicitly available (latent).
- Idea: explicitly model these factors using latent variables \mathbf{z}

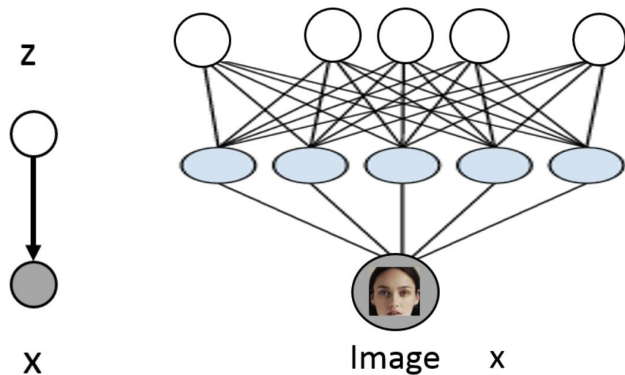


Latent variable models



- Only shaded variables x are observed in the data
- Latent variables z are unobserved - correspond to high-level features
 - We want z to represent useful features e.g. hair color, pose, etc.
 - But very difficult to specify these conditionals by hand and they're unobserved
 - Let's **learn** them instead

Latent variable models



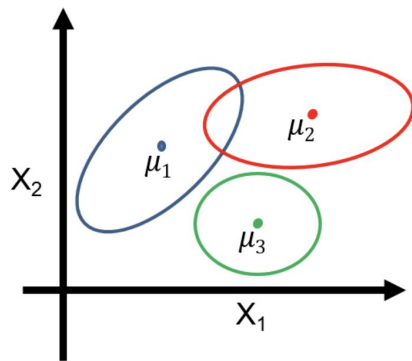
- Put a prior on z $\mathbf{z} \sim \mathcal{N}(0, I)$
- $p(\mathbf{x} | \mathbf{z}) = \mathcal{N}(\mu_{\theta}(\mathbf{z}), \Sigma_{\theta}(\mathbf{z}))$ where $\mu_{\theta}, \Sigma_{\theta}$ are neural networks
- Hope that after training, z will correspond to meaningful latent factors of variation - useful features for unsupervised representation learning
- Given a new image x , features can be extracted via $p(z|x)$

Starting simple: Mixture of Gaussians

Mixture of Gaussians (Bayes network $z \rightarrow x$)

$$\mathbf{z} \sim \text{Categorical}(1, \dots, K)$$

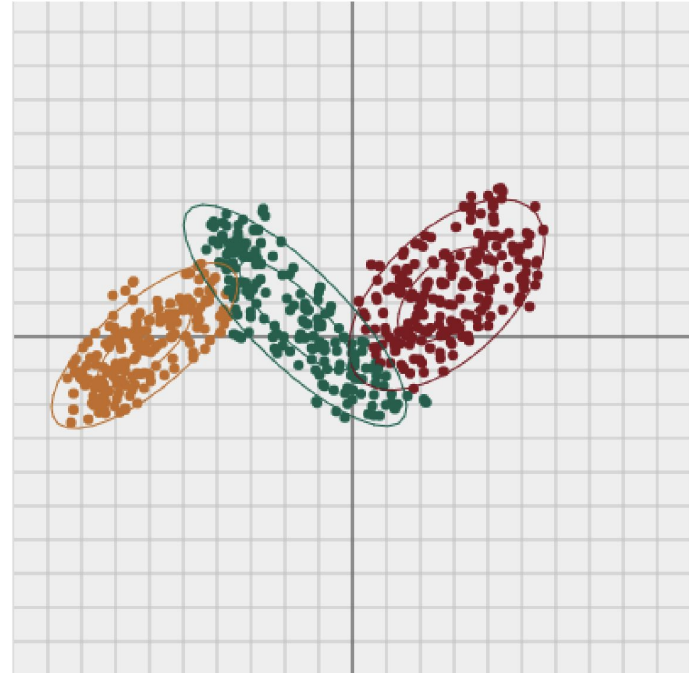
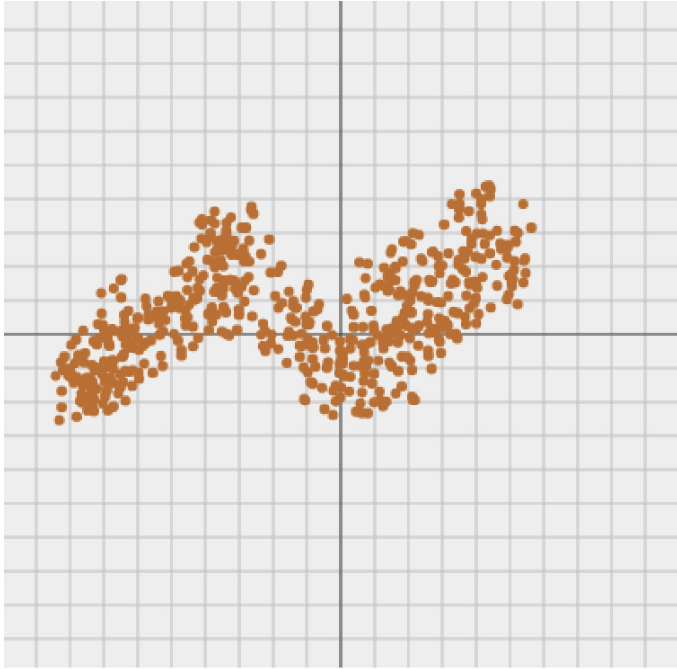
$$p(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mu_k, \Sigma_k)$$



Generative process

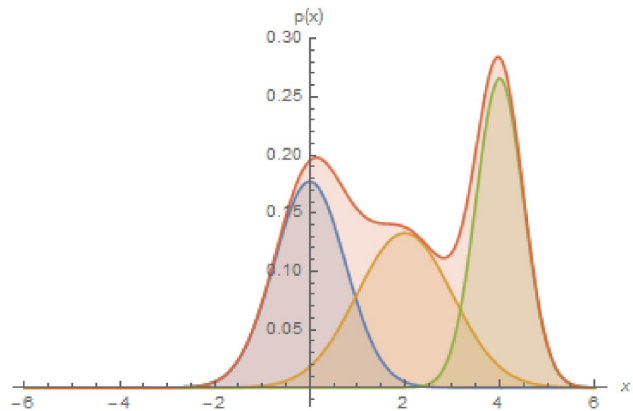
1. Pick a mixture component by sampling z
2. Generate a data point by sampling from that Gaussian

Starting simple: Mixture of Gaussians



Starting simple: Mixture of Gaussians

Combining simple models into more expressive ones



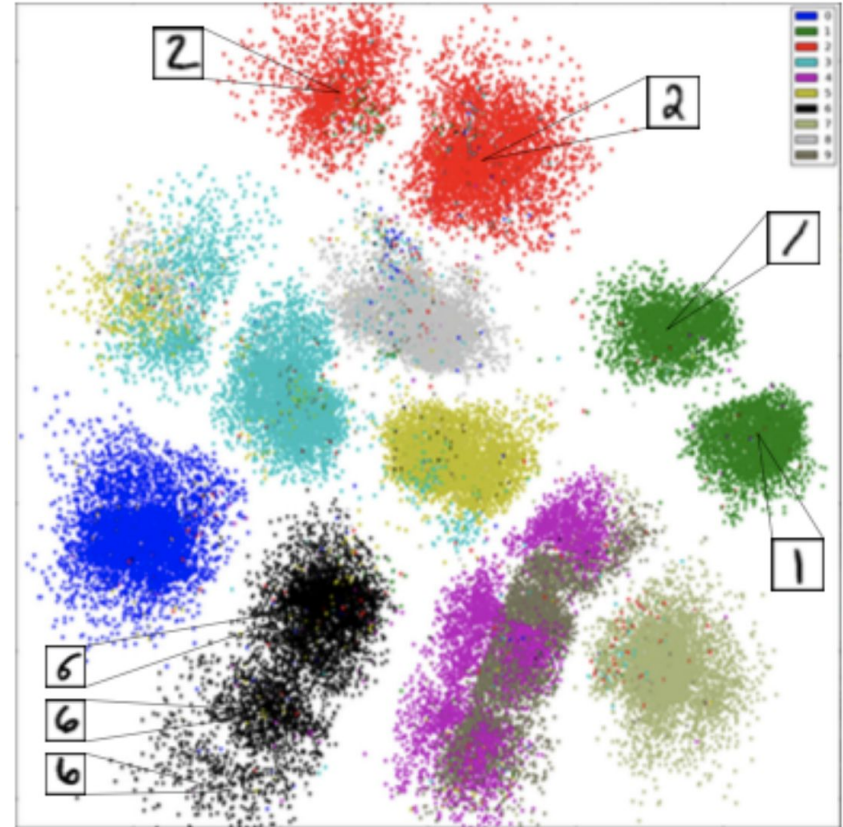
$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x} | \mathbf{z}) = \sum_{k=1}^K p(\mathbf{z} = k) \underbrace{\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}_{\text{component}}$$

can solve using expectation maximization

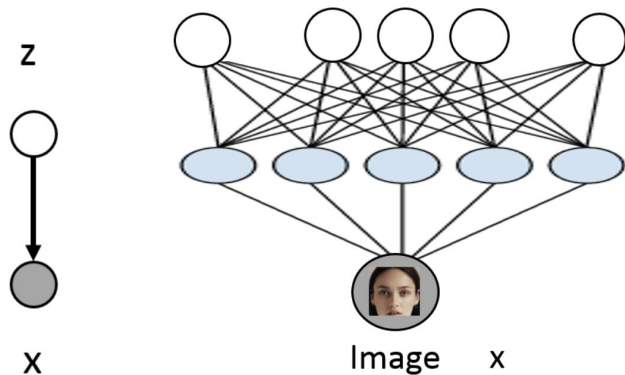
Starting simple: Mixture of Gaussians

Unsupervised clustering of digits

- Discovers clusters corresponding to factors of variation in the data
- Can generate new samples
- Cannot learn features of data i.e. $p(z|x)$



From GMMs to VAEs



- Put a prior on z $\mathbf{z} \sim \mathcal{N}(0, I)$
- $p(\mathbf{x} | \mathbf{z}) = \mathcal{N}(\mu_{\theta}(\mathbf{z}), \Sigma_{\theta}(\mathbf{z}))$ where $\mu_{\theta}, \Sigma_{\theta}$ are neural networks
- Hope that after training, z will correspond to meaningful latent factors of variation - useful features for unsupervised representation learning
- Even though $p(x|z)$ is simple, marginal $p(x)$ is much richer/complex/flexible
- Given a new image x , features can be extracted via $p(z|x)$: natural for unsupervised learning tasks (clustering, representation learning, etc.)

Learning parameters of VAE

- Learning parameters of VAE: we have a joint distribution $p(\mathbf{X}, \mathbf{Z}; \theta)$
- We have a dataset \mathbf{D} where for each datapoint the \mathbf{x} variables are observed (e.g. images, text) and the variables \mathbf{z} are not observed (latent variables)
- We can try maximum likelihood estimation:

$$\log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$$

Learning parameters of VAE

- Learning parameters of VAE: we have a joint distribution $p(\mathbf{X}, \mathbf{Z}; \theta)$
- We have a dataset \mathbf{D} where for each datapoint the \mathbf{x} variables are observed (e.g. images, text) and the variables \mathbf{z} are not observed (latent variables)
- We can try maximum likelihood estimation:

$$\log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$$



Need cheaper approximations to optimize for VAE parameters

intractable :-)

- if \mathbf{z} binary with 30 dimensions, need sum 2^{30} terms

- if \mathbf{z} continuous, integral is hard

Evidence Lower Bound

- Log-likelihood function with partially observed latent variables is hard to compute:

$$\log \left(\sum_{\mathbf{z} \in \mathcal{Z}} p_{\theta}(\mathbf{x}, \mathbf{z}) \right) = \log \left(\sum_{\mathbf{z} \in \mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_{\theta}(\mathbf{x}, \mathbf{z}) \right) = \log \left(\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \right)$$

$q(\mathbf{z})$ should be a simple distribution

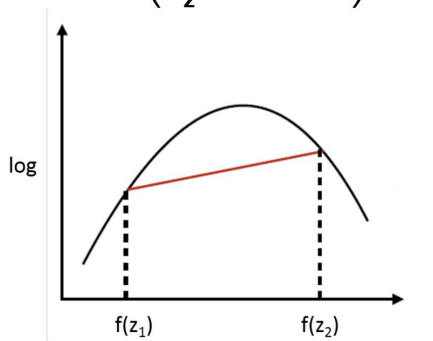
Evidence Lower Bound

- Log-likelihood function with partially observed latent variables is hard to compute:

$$\log \left(\sum_{\mathbf{z} \in \mathcal{Z}} p_{\theta}(\mathbf{x}, \mathbf{z}) \right) = \log \left(\sum_{\mathbf{z} \in \mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_{\theta}(\mathbf{x}, \mathbf{z}) \right) = \log \left(\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \right)$$

- Use Jensen's inequality for concave functions, i.e. $\log(px + (1-p)x') \geq p \log(x) + (1-p) \log(x')$.

$$\log \left(\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} [f(\mathbf{z})] \right) = \log \left(\sum_{\mathbf{z}} q(\mathbf{z}) f(\mathbf{z}) \right) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log f(\mathbf{z})$$



Evidence Lower Bound

- Log-likelihood function with partially observed latent variables is hard to compute:

$$\log \left(\sum_{\mathbf{z} \in \mathcal{Z}} p_{\theta}(\mathbf{x}, \mathbf{z}) \right) = \log \left(\sum_{\mathbf{z} \in \mathcal{Z}} \frac{q(\mathbf{z})}{q(\mathbf{z})} p_{\theta}(\mathbf{x}, \mathbf{z}) \right) = \log \left(\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \right)$$

- Use Jensen's inequality for concave functions, i.e. $\log(px + (1-p)x') \geq p \log(x) + (1-p) \log(x')$.

$$\log \left(\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} [f(\mathbf{z})] \right) = \log \left(\sum_{\mathbf{z}} q(\mathbf{z}) f(\mathbf{z}) \right) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log f(\mathbf{z})$$

Choosing $f(\mathbf{z}) = \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}$

$$\log \left(\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \right) \geq \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\log \left(\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right) \right]$$

Evidence Lower Bound (ELBO)

Evidence Lower Bound

- ELBO holds for any probability distribution $q(\mathbf{z})$ over latent variables:

$$\begin{aligned}\log p(\mathbf{x}; \theta) &\geq \sum_{\mathbf{z}} q(\mathbf{z}) \log \left(\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right) \\ &= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_{\theta}(\mathbf{x}, \mathbf{z}) - \underbrace{\sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z})}_{\text{Entropy } H(q) \text{ of } q} \\ &= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_{\theta}(\mathbf{x}, \mathbf{z}) + H(q)\end{aligned}$$

- Equality holds if $q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x})$:

$$\log p(\mathbf{x}; \theta) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q)$$

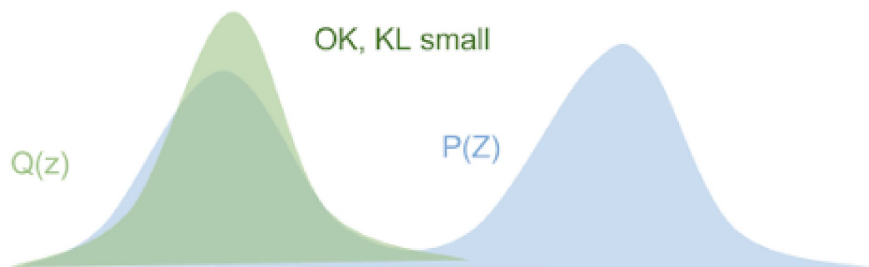
- We want to choose $q(\mathbf{z})$ to be as **close** to $p(\mathbf{z}|\mathbf{x})$ as possible, while being **easy to compute**

The KL divergence

- The KL divergence for variational inference is:

$$\mathbf{D}_{KL}(q(z)||p(z|x)) = \int q(z) \log \frac{q(z)}{p(z|x)} dz$$

- Intuitively, there are three cases
 - a. If q is low then we don't care (because of the expectation).
 - b. If q is high and p is high then we are happy.
 - c. If q is high and p is low then we pay a price.
- Note that p must be > 0 wherever $q > 0$



Evidence Lower Bound

- Starting from the KL divergence:

$$D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x}; \theta)) = - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + \log p(\mathbf{x}; \theta) - H(q) \geq 0$$

- Re-derive ELBO from KL divergence:

$$\log p(\mathbf{x}; \theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q)$$

- Equality holds if $q = p(\mathbf{z}|\mathbf{x})$ because $KL(q\|p) = 0$:

$$\log p(\mathbf{x}; \theta) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q)$$

Evidence Lower Bound

- Starting from the KL divergence:

$$D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x}; \theta)) = - \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + \log p(\mathbf{x}; \theta) - H(q) \geq 0$$

- Re-derive ELBO from KL divergence:

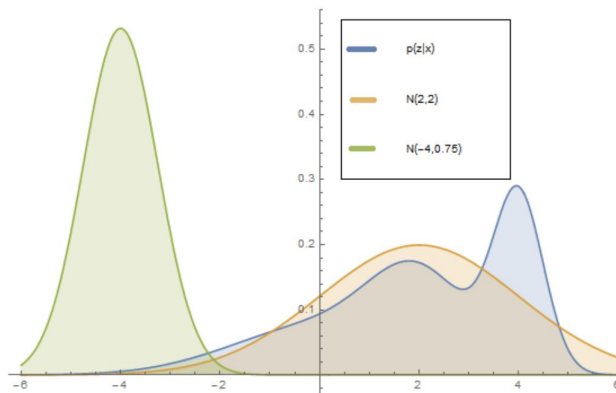
$$\log p(\mathbf{x}; \theta) \geq \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q)$$

- Equality holds if $q = p(\mathbf{z}|\mathbf{x})$ because $KL(q\|p) = 0$:

$$\log p(\mathbf{x}; \theta) = \sum_{\mathbf{z}} q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}; \theta) + H(q)$$

- In general, $\log p(\mathbf{x}; \theta) = \text{ELBO} + D_{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x}; \theta))$
- The closer the chosen q is to $p(\mathbf{z}|\mathbf{x})$, the closer the ELBO is to the true likelihood.

Variational Inference



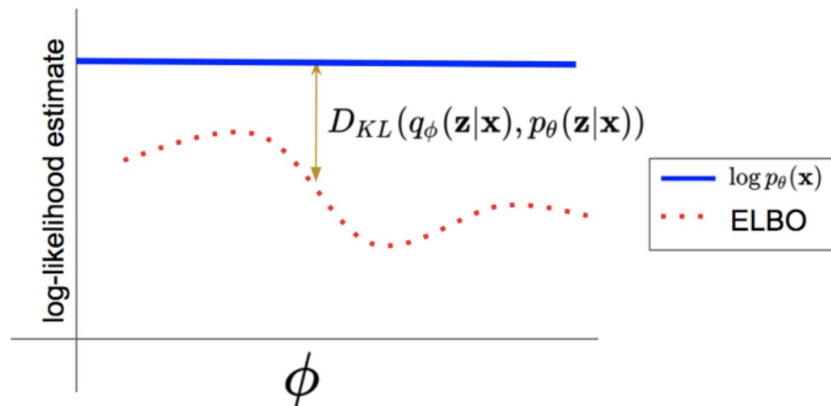
Suppose $q(\mathbf{z}; \phi)$ is a (tractable) probability distribution over the hidden variables parameterized by ϕ (variational parameters)

- For example, a Gaussian with mean and covariance specified by ϕ

$$q(\mathbf{z}; \phi) = \mathcal{N}(\phi_1, \phi_2)$$

- Variational inference: optimize variational parameters so that $q(\mathbf{z}; \phi)$ is as close as possible to $p(\mathbf{z}|\mathbf{x}; \theta)$ while being simple to compute
- E.g. in figure, posterior (in blue) is better approximated by orange Gaussian than green

Variational Inference



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

- In practice how can we learn encoder parameters $p(\mathbf{z}|\mathbf{x}; \theta)$ and variational (decoder) parameters jointly? $q(\mathbf{z}; \phi)$

Learning the parameters

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

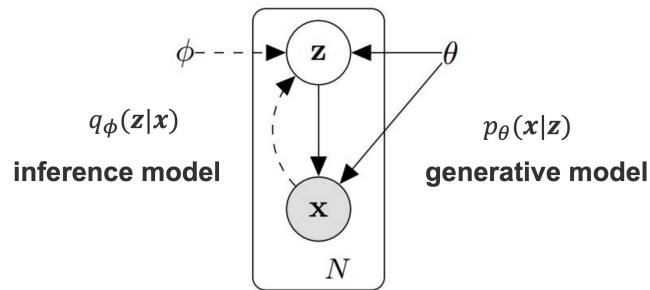


Figure courtesy: Kingma & Welling, 2014

Learning the parameters

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

$\underbrace{\hspace{10em}}_{\text{reconstruction}} \quad \underbrace{\hspace{10em}}_{\text{prior}}$

What does the training objective $\mathcal{L}(\mathbf{x}; \theta, \phi)$ do?

- First term encourages $\hat{\mathbf{x}} \approx \mathbf{x}^i$ (\mathbf{x}^i likely under $p(\mathbf{x}|\hat{\mathbf{z}}; \theta)$)
- Second term encourages $\hat{\mathbf{z}}$ to be likely under the prior $p(\mathbf{z})$

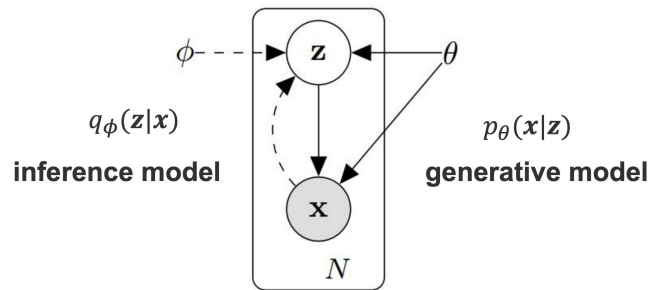


Figure courtesy: Kingma & Welling, 2014

Learning the parameters

$$\begin{aligned} \mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= \underbrace{E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)]}_{\text{reconstruction}} - \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{prior}} \end{aligned}$$

reconstruction prior

What does the training objective $\mathcal{L}(\mathbf{x}; \theta, \phi)$ do?

- First term encourages $\hat{\mathbf{x}} \approx \mathbf{x}^i$ (\mathbf{x}^i likely under $p(\mathbf{x}|\hat{\mathbf{z}}; \theta)$)
- Second term encourages $\hat{\mathbf{z}}$ to be likely under the prior $p(\mathbf{z})$

- 1 Take a data point \mathbf{x}^i
- 2 Map it to $\hat{\mathbf{z}}$ by sampling from $q_\phi(\mathbf{z}|\mathbf{x}^i)$ (*encoder*)
- 3 Reconstruct $\hat{\mathbf{x}}$ by sampling from $p(\mathbf{x}|\hat{\mathbf{z}}; \theta)$ (*decoder*)

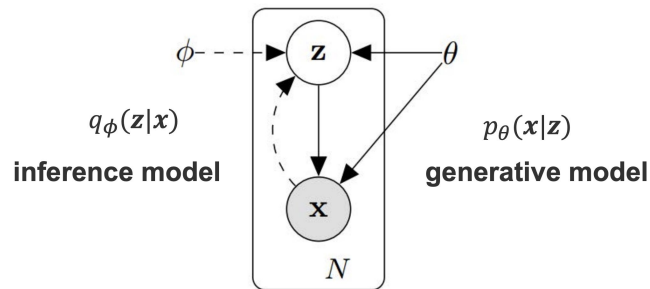
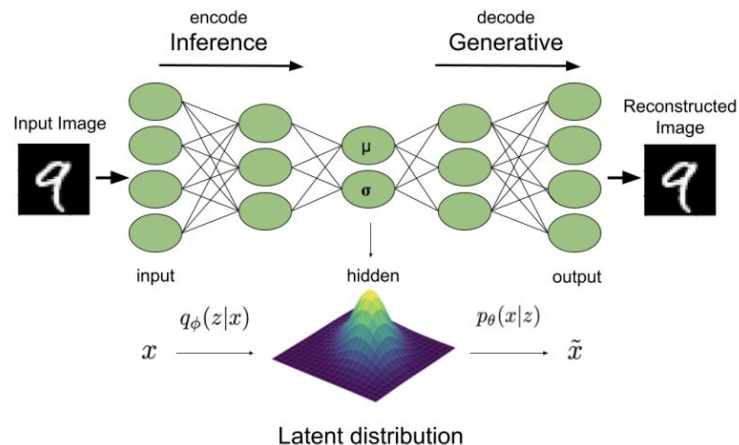


Figure courtesy: Kingma & Welling, 2014



Learning the parameters

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

- We need to compute the gradients $\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x}; \theta, \phi)$


easy

Learning the parameters

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

- We need to compute the gradients $\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x}; \theta, \phi)$


easy

$$\begin{aligned}\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi) &= \nabla_\theta E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &= \nabla_\theta E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)]\end{aligned}$$

Learning the parameters

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

- We need to compute the gradients $\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x}; \theta, \phi)$


easy

$$\begin{aligned}\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi) &= \nabla_\theta E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &= \nabla_\theta E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\nabla_\theta \log p(\mathbf{x}|\mathbf{z}; \theta)]\end{aligned}$$

Learning the parameters

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

- We need to compute the gradients $\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x}; \theta, \phi)$


easy

$$\begin{aligned}\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi) &= \nabla_\theta E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \\ &= \nabla_\theta E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\nabla_\theta \log p(\mathbf{x}|\mathbf{z}; \theta)] \\ &\approx \frac{1}{n} \sum_{i=1}^n \nabla_\theta \log p(\mathbf{x}|\mathbf{z}_i; \theta)\end{aligned}$$

Learning the parameters

$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\end{aligned}$$

- We need to compute the gradients $\underbrace{\nabla_\theta \mathcal{L}(\mathbf{x}; \theta, \phi)}_{\text{easy}}$ and $\underbrace{\nabla_\phi \mathcal{L}(\mathbf{x}; \theta, \phi)}_{\text{tricky}}$
- Expectations also depend on ϕ

$$\nabla_\phi \mathcal{L}(\mathbf{x}; \theta, \phi) = \nabla_\phi E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

Reparameterization trick

- Want to compute a gradient with respect to ϕ of

$$E_{q(\mathbf{z}; \phi)}[r(\mathbf{z})] = \int q(\mathbf{z}; \phi) r(\mathbf{z}) d\mathbf{z}$$

where \mathbf{z} is now **continuous**

Reparameterization trick

- Want to compute a gradient with respect to ϕ of

$$E_{q(\mathbf{z}; \phi)}[r(\mathbf{z})] = \int q(\mathbf{z}; \phi) r(\mathbf{z}) d\mathbf{z}$$

where \mathbf{z} is now **continuous**

- Suppose $q(\mathbf{z}; \phi) = \mathcal{N}(\mu, \sigma^2 I)$ is Gaussian with parameters $\phi = (\mu, \sigma)$. These are equivalent ways of sampling:
 - Sample $\mathbf{z} \sim q_\phi(\mathbf{z})$
 - Sample $\epsilon \sim \mathcal{N}(0, I)$, $\mathbf{z} = \mu + \sigma\epsilon = g(\epsilon; \phi)$

Reparameterization trick

- Want to compute a gradient with respect to ϕ of

$$E_{q(\mathbf{z}; \phi)}[r(\mathbf{z})] = \int q(\mathbf{z}; \phi) r(\mathbf{z}) d\mathbf{z}$$

where \mathbf{z} is now **continuous**

- Suppose $q(\mathbf{z}; \phi) = \mathcal{N}(\mu, \sigma^2 I)$ is Gaussian with parameters $\phi = (\mu, \sigma)$. These are equivalent ways of sampling:
 - Sample $\mathbf{z} \sim q_\phi(\mathbf{z})$
 - Sample $\epsilon \sim \mathcal{N}(0, I)$, $\mathbf{z} = \mu + \sigma\epsilon = g(\epsilon; \phi)$
- Using this equivalence we compute the expectation in two ways:

$$E_{\mathbf{z} \sim q(\mathbf{z}; \phi)}[r(\mathbf{z})] = E_{\epsilon \sim \mathcal{N}(0, I)}[r(g(\epsilon; \phi))] = \int p(\epsilon) r(\mu + \sigma\epsilon) d\epsilon$$

$$\nabla_\phi E_{q(\mathbf{z}; \phi)}[r(\mathbf{z})] = \nabla_\phi E_\epsilon[r(g(\epsilon; \phi))] = E_\epsilon[\nabla_\phi r(g(\epsilon; \phi))]$$

Reparameterization trick

- Want to compute a gradient with respect to ϕ of

$$E_{q(\mathbf{z}; \phi)}[r(\mathbf{z})] = \int q(\mathbf{z}; \phi) r(\mathbf{z}) d\mathbf{z}$$

where \mathbf{z} is now **continuous**

- Suppose $q(\mathbf{z}; \phi) = \mathcal{N}(\mu, \sigma^2 I)$ is Gaussian with parameters $\phi = (\mu, \sigma)$. These are equivalent ways of sampling:

- Sample $\mathbf{z} \sim q_\phi(\mathbf{z})$
- Sample $\epsilon \sim \mathcal{N}(0, I)$, $\mathbf{z} = \mu + \sigma\epsilon = g(\epsilon; \phi)$
- Using this equivalence we compute the expectation in two ways:

$$E_{\mathbf{z} \sim q(\mathbf{z}; \phi)}[r(\mathbf{z})] = E_{\epsilon \sim \mathcal{N}(0, I)}[r(g(\epsilon; \phi))] = \int p(\epsilon) r(\mu + \sigma\epsilon) d\epsilon$$

$$\nabla_\phi E_{q(\mathbf{z}; \phi)}[r(\mathbf{z})] = \nabla_\phi E_\epsilon[r(g(\epsilon; \phi))] = E_\epsilon[\nabla_\phi r(g(\epsilon; \phi))]$$

- Easy to estimate via Monte Carlo if r and g are differentiable w.r.t. ϕ and ϵ is easy to sample from (backpropagation)
- $E_\epsilon[\nabla_\phi r(g(\epsilon; \phi))] \approx \frac{1}{k} \sum_k \nabla_\phi r(g(\epsilon^k; \phi))$ where $\epsilon^1, \dots, \epsilon^k \sim \mathcal{N}(0, I)$.

Reparameterization trick

$$\nabla_{\phi} \mathcal{L}(\mathbf{x}; \theta, \phi) = \nabla_{\phi} E_{q_{\phi}(z|\mathbf{x})}[\log p(\mathbf{x}|z; \theta)] - D_{KL}(q_{\phi}(z|\mathbf{x}) || p(z))$$

$$\nabla_{\phi} E_{q_{\phi}(z|\mathbf{x})}[\log p(\mathbf{x}|z; \theta)] = \nabla_{\phi} E_{\epsilon}[\log p(\mathbf{x}|\mu + \sigma\epsilon; \theta)] \quad \text{reparameterize}$$

Reparameterization trick

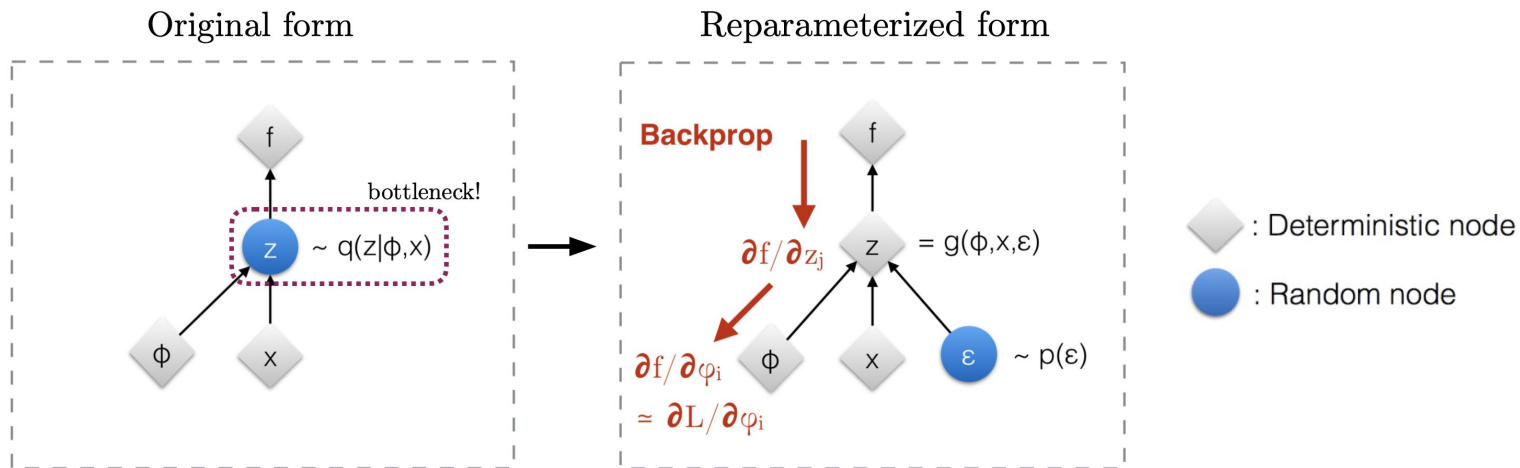
$$\nabla_{\phi} \mathcal{L}(\mathbf{x}; \theta, \phi) = \nabla_{\phi} E_{q_{\phi}(z|\mathbf{x})}[\log p(\mathbf{x}|z; \theta)] - D_{KL}(q_{\phi}(z|\mathbf{x}) || p(z))$$

$$\begin{aligned} \nabla_{\phi} E_{q_{\phi}(z|\mathbf{x})}[\log p(\mathbf{x}|z; \theta)] &= \nabla_{\phi} E_{\epsilon}[\log p(\mathbf{x}|\mu + \sigma\epsilon; \theta)] \quad \text{reparameterize} \\ &= E_{\epsilon}[\nabla_{\phi} \log p(\mathbf{x}|\mu + \sigma\epsilon; \theta)] \end{aligned}$$

Reparameterization trick

$$\nabla_{\phi} \mathcal{L}(x; \theta, \phi) = \nabla_{\phi} E_{q_{\phi}(z|x)}[\log p(x|z; \theta)] - D_{KL}(q_{\phi}(z|x) || p(z))$$

$$\begin{aligned} \nabla_{\phi} E_{q_{\phi}(z|x)}[\log p(x|z; \theta)] &= \nabla_{\phi} E_{\epsilon}[\log p(x|\mu + \sigma\epsilon; \theta)] \quad \text{reparameterize} \\ &= E_{\epsilon}[\nabla_{\phi} \log p(x|\mu + \sigma\epsilon; \theta)] \\ &\approx \frac{1}{n} \sum_{i=1}^n [\nabla_{\phi} \log p(x|\mu + \sigma\epsilon_i; \theta)] \end{aligned}$$



Learning the parameters

$$\begin{aligned}
 \mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
 &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
 &= \underbrace{E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)]}_{\text{reconstruction}} - \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\text{prior}}
 \end{aligned}$$

reconstruction

prior

1. Take a datapoint x_i .
2. Map it to μ, σ using $q_\phi(\mathbf{z}|x_i)$. **encoder**
3. Sample $\epsilon \sim N(0, I)$ and compute $\hat{\mathbf{z}} = \mu + \sigma\epsilon$. **reparameterize**
4. Reconstruct \hat{x} by sampling from $p(\mathbf{x}|\hat{\mathbf{z}}; \theta)$. **decoder**

Differentiable using reparameterization trick

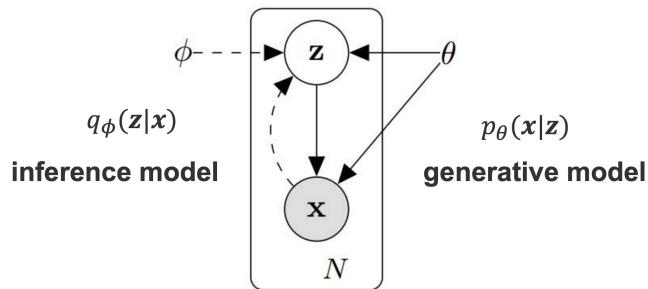
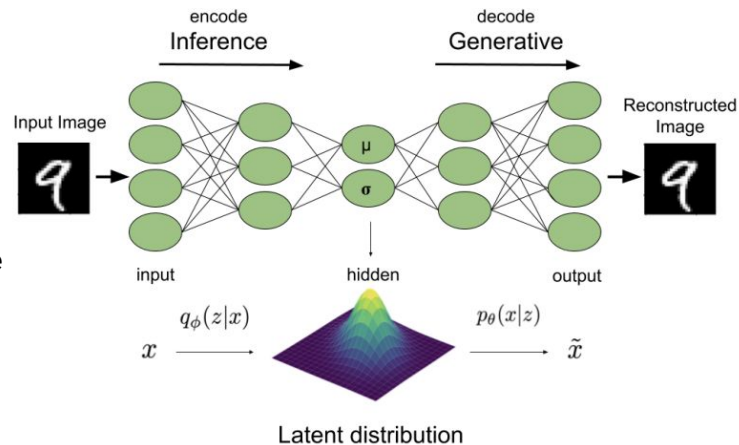


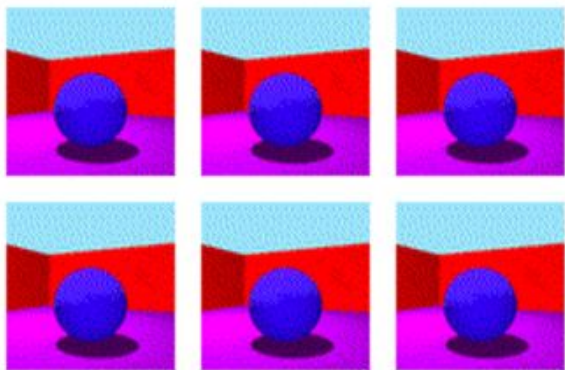
Figure courtesy: Kingma & Welling, 2014



VAE for disentanglement

Disentangled representation learning

- Very useful for style transfer: disentangling **style** from **content**



disentanglement_lib

[Locatello et al., ICML 2019]



[Gatys et al., CVPR 2016]

From negative to positive

consistently slow .
consistently good .
consistently fast .

[Shen et al., NeurIPS 2017]

my goodness it was so gross .
my husband 's steak was phenomenal .
my goodness was so awesome .

it was super dry and had a weird taste to the entire slice .
it was a great meal and the tacos were very kind of good .
it was super flavorful and had a nice texture of the whole side .

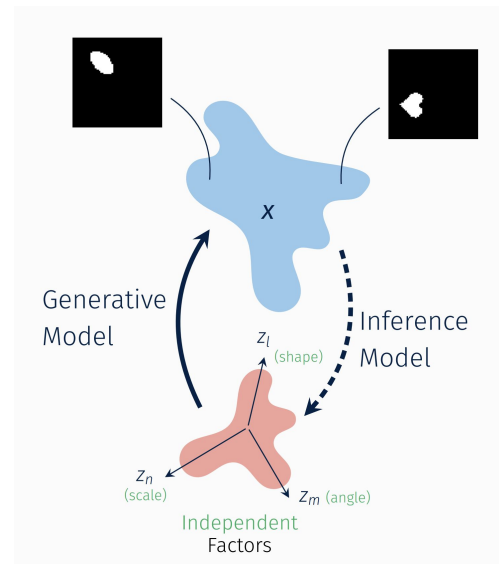
VAE for disentanglement

Disentangled representation learning

- Very useful for style transfer: disentangling **style** from **content**

$$\mathcal{L}_\beta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \cdot \text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

- beta-VAE: beta = 1 recovers VAE, beta > 1 imposes stronger constraint on the latent variables to have independent dimensions



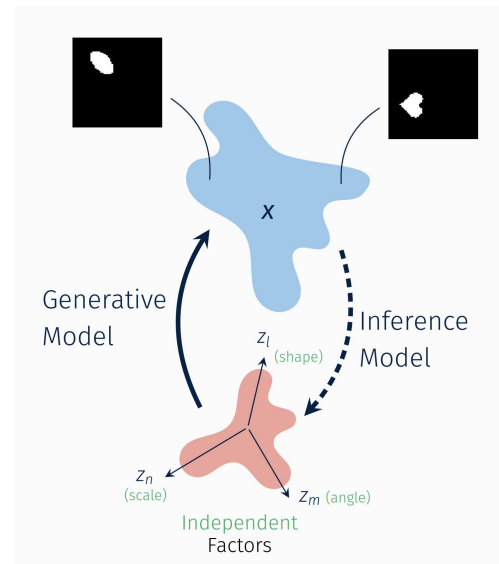
VAE for disentanglement

Disentangled representation learning

- Very useful for style transfer: disentangling **style** from **content**

$$\mathcal{L}_\beta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \cdot \text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

- beta-VAE: beta = 1 recovers VAE, beta > 1 imposes stronger constraint on the latent variables to have independent dimensions
- Difficult problem!
 - Positive results [Hu et al., 2016, Kulkarni et al., 2015]
 - Negative results [Mathieu et al., 2019, Locatello et al., 2019]
 - Better benchmarks & metrics to measure disentanglement [Higgins et al., 2017, Kim & Mnih 2018]




VAE for multimodal data

Language: *And he I don't think he got mad when hah I don't know maybe.* *Too much too fast, I mean we basically just get introduced to this character...* *All I can say is he's a pretty average guy.*

Vision: *Gaze aversion* *Uninformative* *Contradictory smile*

Acoustic: *(frustrated voice)* *(angry voice)* *(disappointed voice)*



Z_y Joint multimodal factor (sentiment/emotion)

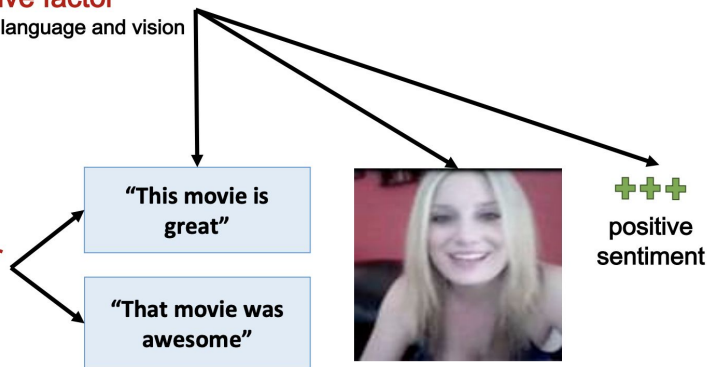
Z_{a1} Language specific factor

Z_{a2} Visual specific factor

Z_{a3} Audio specific factor

(1) Multimodal discriminative factor
models the label and variations across both language and vision

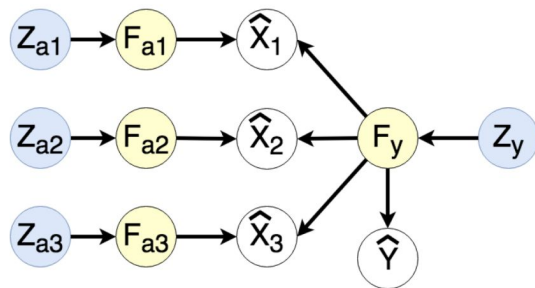
(2) Language generative factor
models variations within language



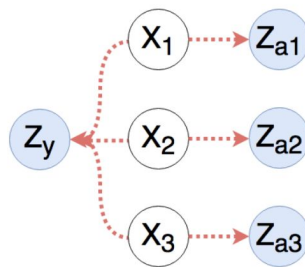
Disentangled factors

VAE for multimodal data

Language: *And he I don't think he got mad when hah I don't know maybe.* *Too much too fast, I mean we basically just get introduced to this character...* *All I can say is he's a pretty average guy.*



(a) MFM
Generative Network



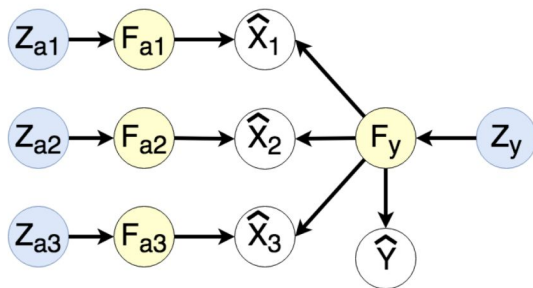
(b) MFM
Inference Network

- Z_y Joint multimodal factor (sentiment/emotion)
- Z_{a1} Language specific factor
- Z_{a2} Visual specific factor
- Z_{a3} Audio specific factor

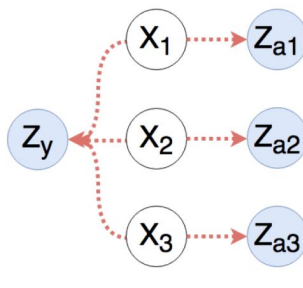
Disentangled factors

VAE for multimodal data

Language: *And he I don't think he got mad when hah I don't know maybe.* *Too much too fast, I mean we basically just get introduced to this character...* *All I can say is he's a pretty average guy.*



(a) **MFM**
Generative Network



(b) **MFM**
Inference Network

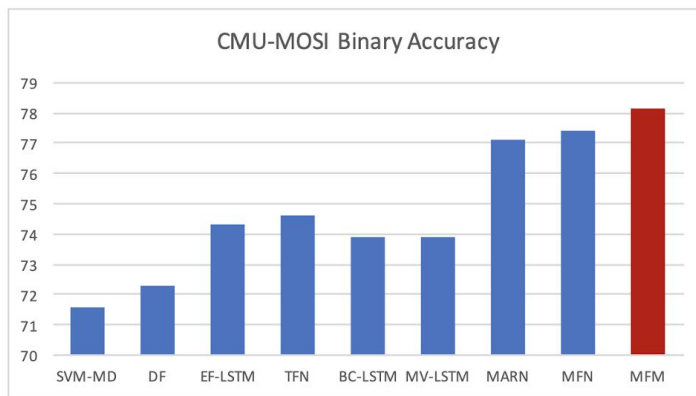
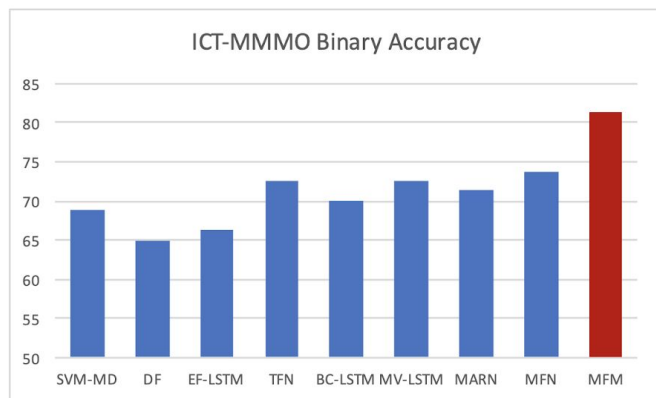
reconstruction $\sum_{i=1}^M c_{X_i} (\mathbf{X}_i, F(G_{ai}(\mathbf{Z}_{ai}), G_y(\mathbf{Z}_y)))$

prediction $c_Y(\mathbf{Y}, D(G_y(\mathbf{Z}_y)))$

prior $\lambda D_{\text{KL}}(Q(\mathbf{z}|\mathbf{x}) || P(\mathbf{z}))$

VAE for multimodal data

Discriminative performance

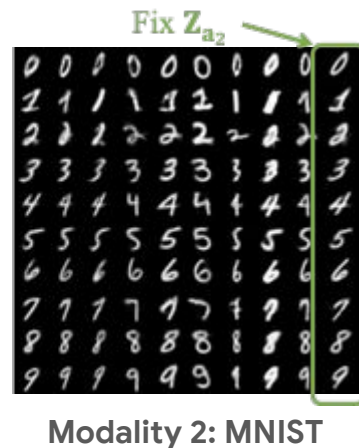
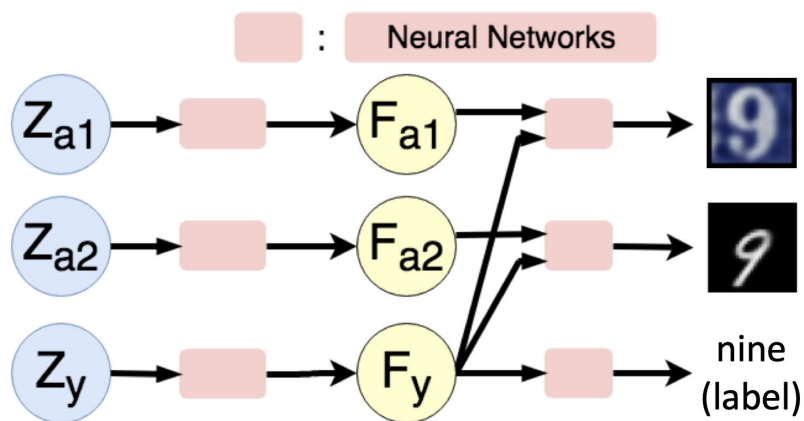


MFM achieves strong performance on 6 multimodal time-series datasets

MFM can be applied on any multimodal fusion encoder

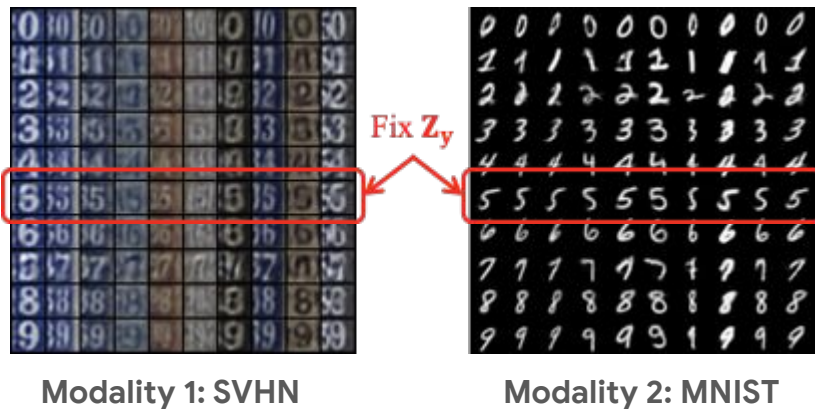
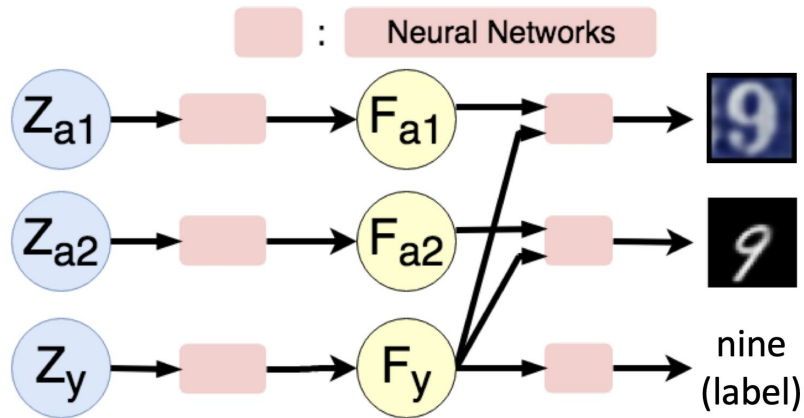
VAE for multimodal data

Disentangling factors of variation: controllable generation



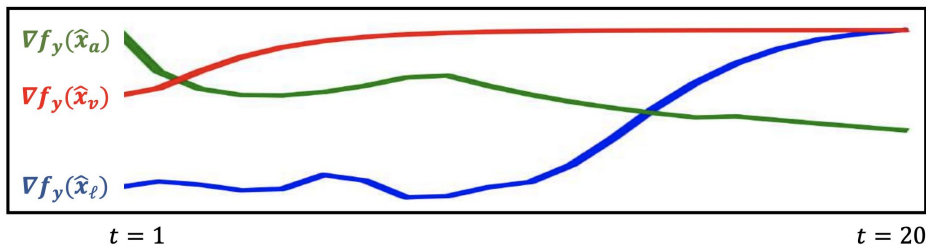
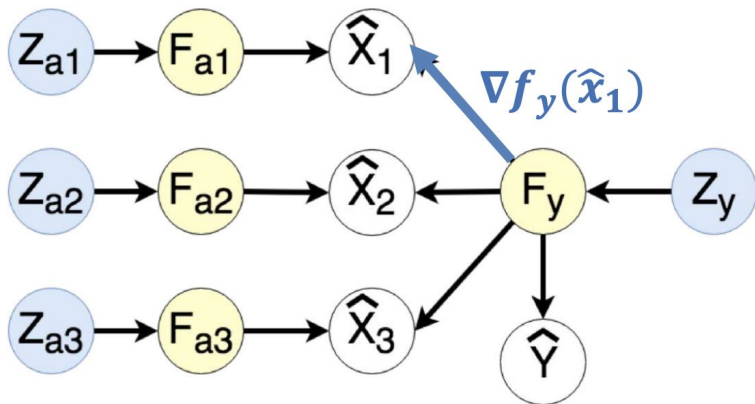
VAE for multimodal data

Disentangling factors of variation: controllable generation



VAE for multimodal data

Interpreting modality-specific factors



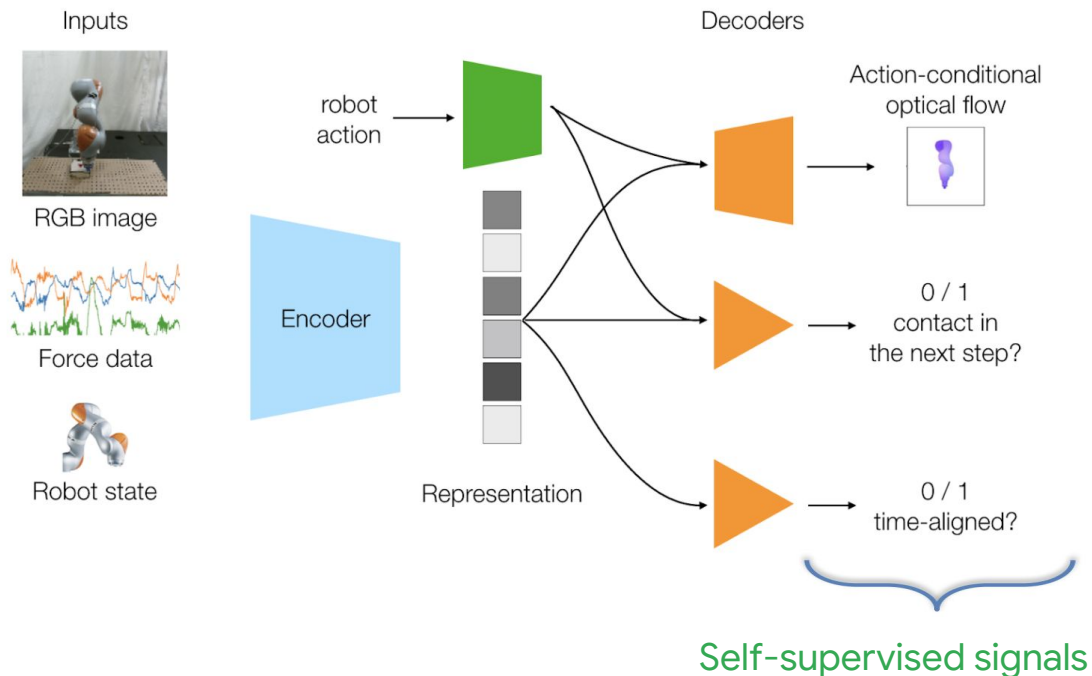
language Umm, in a way, a lot of the themes in "never let me go", which were very profound and deep.



VAE for multimodal data

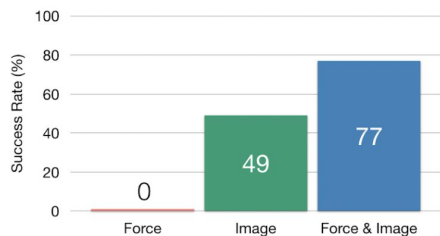
VAEs beyond reconstruction

- It can be hard to reconstruct high-dimensional input modalities
- Combine VAEs with self-supervised learning: reconstruct **important signals** from the input



VAE for multimodal data

High success rate from multimodal signals

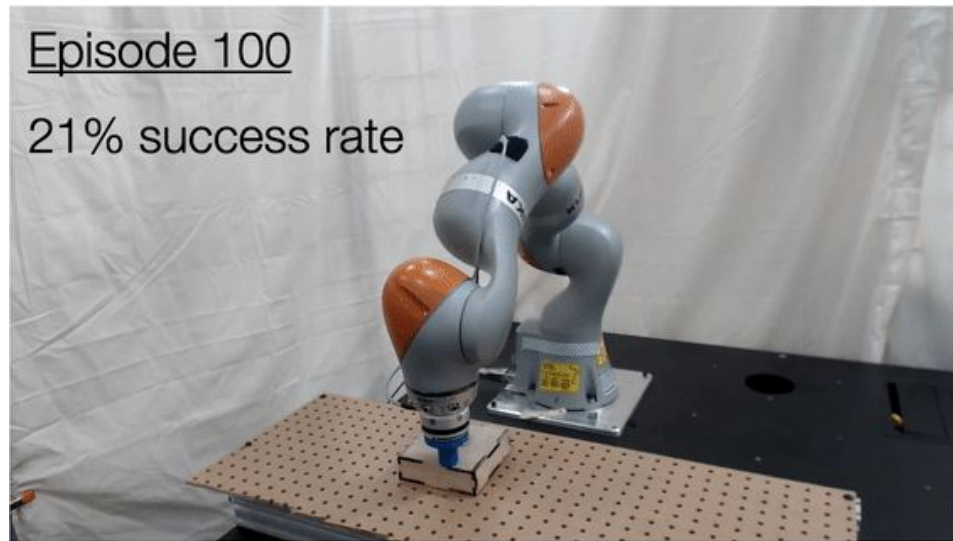


Simulation Results
(Randomized box location)

Force Only: Can't find box

Image Only: Struggles with peg alignment

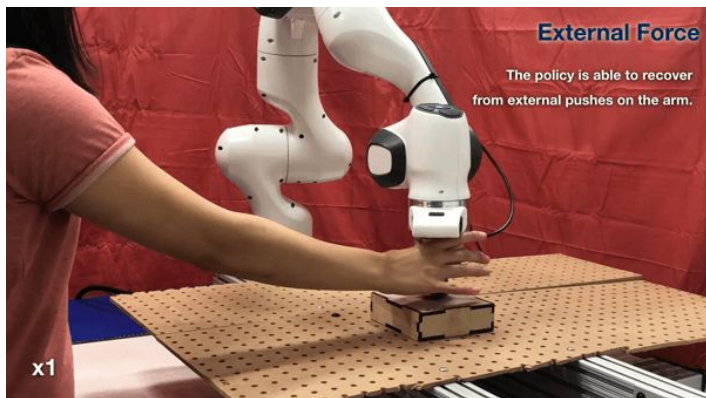
Force & Image: Can learn full task completion



VAE for multimodal data

Robustness to:

- external forces
- camera occlusion
- moving targets



Summary: generative models

VAEs

- Relatively easier to train
- Explicit inference network $q(z|x)$
- More blurry images (due to reconstruction)

Query



Prominent attributes: White, Fully Visible Forehead, Mouth Closed, Male, Curly Hair, Eyes Open, Pale Skin, Frowning, Pointy Nose, Teeth Not Visible, No Eyewear.

VAE



GAN



VAE/GAN



Query



Prominent attributes: White, Male, Curly Hair, Frowning, Eyes Open, Pointy Nose, Flash, Posed Photo, Eyeglasses, Narrow Eyes, Teeth Not Visible, Senior, Receding Hairline.

VAE



GAN



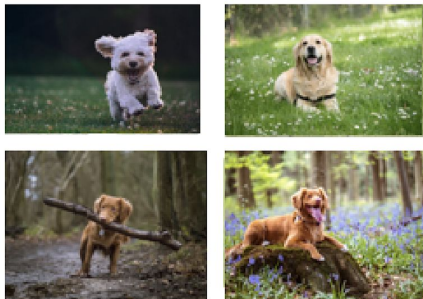
VAE/GAN



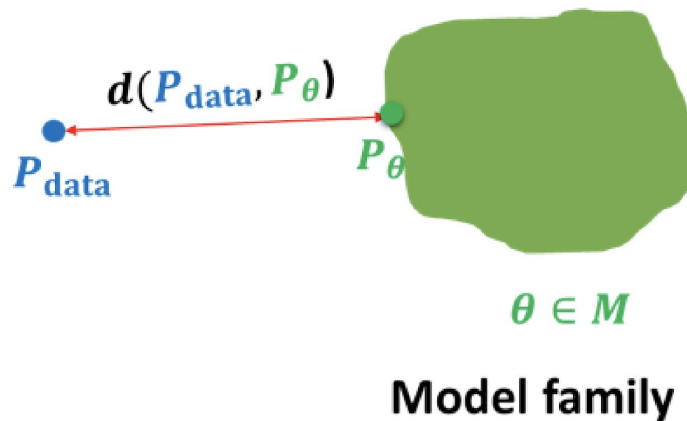
Part 3: Generative Adversarial Nets

Beyond likelihood-based learning:

- Difficulty in evaluating and optimizing $p(x)$ in high-dimensions
- High $p(x)$ might not correspond to realistic samples

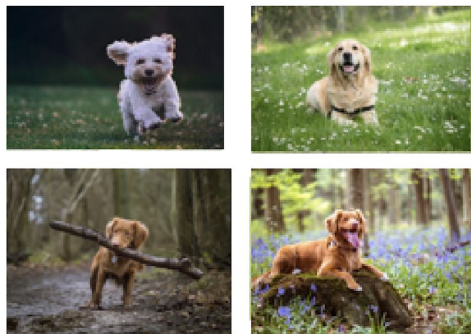


$$x_i \sim P_{\text{data}} \\ i = 1, 2, \dots, n$$



Part 3: Generative Adversarial Nets

Towards likelihood-free learning



$$S_1 = \{\mathbf{x} \sim P\}$$

vs.



$$S_2 = \{\mathbf{x} \sim Q\}$$

Given a finite set of samples from two distributions, how can we tell if these samples are from the same distribution? (i.e. $P = Q$?)

Part 3: Generative Adversarial Nets

Given $S_1 = \{\mathbf{x} \sim P\}$ and $S_2 = \{\mathbf{x} \sim Q\}$, a **two-sample test** considers the following hypotheses

- Null hypothesis $H_0: P = Q$
- Alternate hypothesis $H_1: P \neq Q$

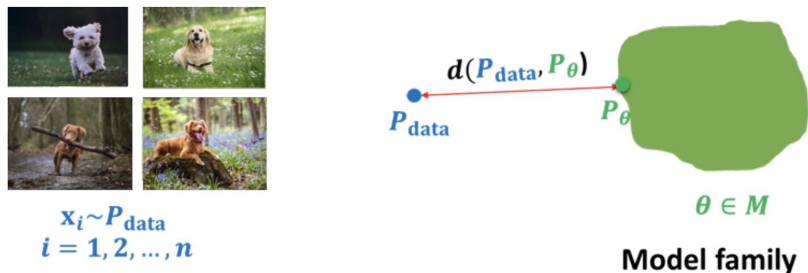
Test statistic T compares S_1 and S_2 e.g., difference in means, variances of the two sets of samples

If T is less than a threshold α , then accept H_0 else reject it

Key observation: Test statistic is **likelihood-free** since it does not involve the densities P or Q , only samples

Part 3: Generative Adversarial Nets

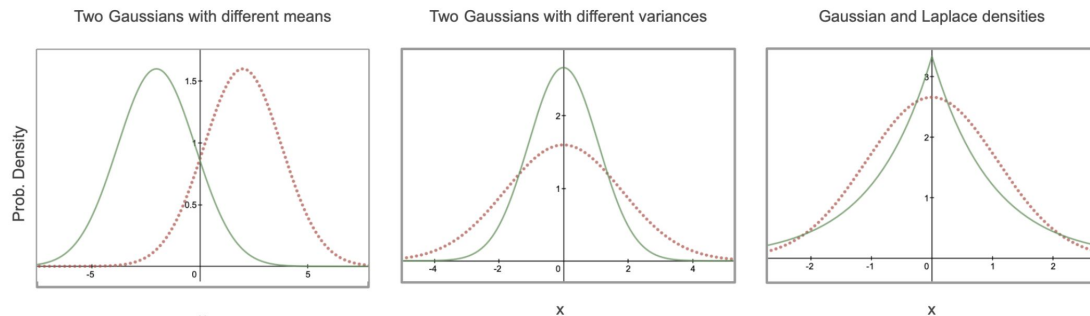
Towards likelihood-free learning



- Assume we have access to $S_1 = \mathcal{D} = \{\mathbf{x} \sim p_{\text{data}}\}$
- In addition, we have our model's distribution p_{θ}
- Assume that our model's distribution permits efficient sampling of $S_2 = \{\mathbf{x} \sim p_{\theta}\}$
- **Alternate notion of distance between distributions:** Train the generative model to minimize a two-sample test objective between S_1 and S_2

Part 3: Generative Adversarial Nets

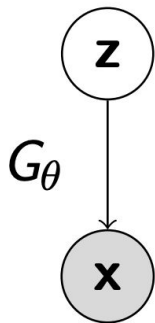
Towards likelihood-free learning



- Problem: finding a two-sample test objective in high-dimensions is hard
- In the generative model setup, we know that S_1 and S_2 come from different distributions p_{data} and p_{θ} respectively
- **Key idea: learn** a statistic that **maximizes** a suitable notion of distance between the two sets of samples S_1 and S_2

Part 3: Generative Adversarial Nets

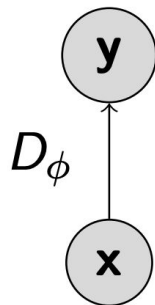
- A 2 player minimax game between a **generator** and a **discriminator**



- **Generator:** a directed latent variable model from z to x
- Minimizes the two-sample test objective: in support of null hypothesis $p_{\text{data}} = p_\theta$

Part 3: Generative Adversarial Nets

- A 2 player minimax game between a **generator** and a **discriminator**



- **Discriminator:** any function (e.g. neural network) that tries to distinguish ‘real’ samples from the datasets from ‘fake’ samples generated by the model
- Maximizes the two-sample test objective: in support of alternative hypothesis $p_{\text{data}} \neq p_\theta$

Training the discriminator

- Training objective for **discriminator**

$$\max_D V(G, D) = E_{\mathbf{x} \sim p_{\text{data}}} [\log D(\mathbf{x})] + E_{\mathbf{x} \sim p_G} [\log(1 - D(\mathbf{x}))]$$

- For a fixed generator G , the discriminator performs binary classification between true samples (assign label 1) vs fake samples (assign label 0)
- Optimal discriminator:

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

Training the generator

- Training objective for **generator**

$$\min_G V(G, D) = E_{\mathbf{x} \sim p_{\text{data}}} [\log D(\mathbf{x})] + E_{\mathbf{x} \sim p_G} [\log(1 - D(\mathbf{x}))]$$

- For the optimal discriminator $D_G^*(\cdot)$, we have

$$\begin{aligned} & V(G, D_G^*(\mathbf{x})) \\ &= E_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \right] + E_{\mathbf{x} \sim p_G} \left[\log \frac{p_G(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \right] \\ &= E_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}{2}} \right] + E_{\mathbf{x} \sim p_G} \left[\log \frac{p_G(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}{2}} \right] - \log 4 \\ &= \underbrace{D_{KL} \left[p_{\text{data}}, \frac{p_{\text{data}} + p_G}{2} \right] + D_{KL} \left[p_G, \frac{p_{\text{data}} + p_G}{2} \right]}_{2 \times \text{Jensen-Shannon Divergence (JSD)}} - \log 4 \\ &= 2D_{JSD}[p_{\text{data}}, p_G] - \log 4 \end{aligned}$$

More about divergences

- Also known as the **symmetric** KL divergence

$$D_{JSD}[p, q] = \frac{1}{2} \left(D_{KL} \left[p, \frac{p+q}{2} \right] + D_{KL} \left[q, \frac{p+q}{2} \right] \right)$$

- Properties

- $D_{JSD}[p, q] \geq 0$
- $D_{JSD}[p, q] = 0$ iff $p = q$
- $D_{JSD}[p, q] = D_{JSD}[q, p]$
- $\sqrt{D_{JSD}[p, q]}$ satisfies triangle inequality \rightarrow Jensen-Shannon Distance

- Optimal generator for the JSD/Negative Cross Entropy GAN

$$p_G = p_{\text{data}}$$

GAN training

- Sample minibatch of m training points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}$ from \mathcal{D}
- Sample minibatch of m noise vectors $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots, \mathbf{z}^{(m)}$ from p_z
- Update the generator parameters θ by stochastic gradient **descent**

$$\nabla_{\theta} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m \log(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)})))$$

- Update the discriminator parameters ϕ by stochastic gradient **ascent**

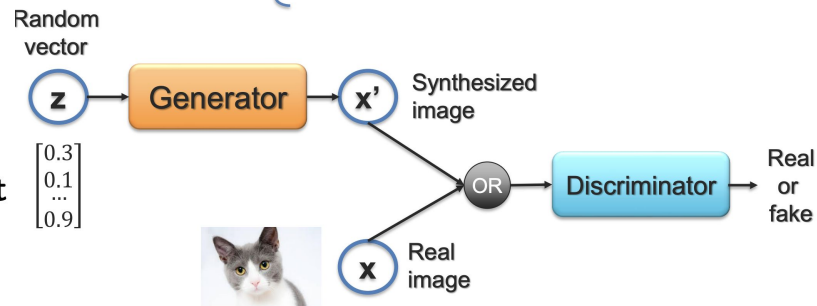
$$\nabla_{\phi} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\phi} \sum_{i=1}^m [\log D_{\phi}(\mathbf{x}^{(i)}) + \log(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)})))]$$

- Repeat for fixed number of epochs

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

Optimization:

- 1 Fix generator, and update discriminator
- 2 Fix discriminator, and update generator



GAN extensions

Progress in face generation

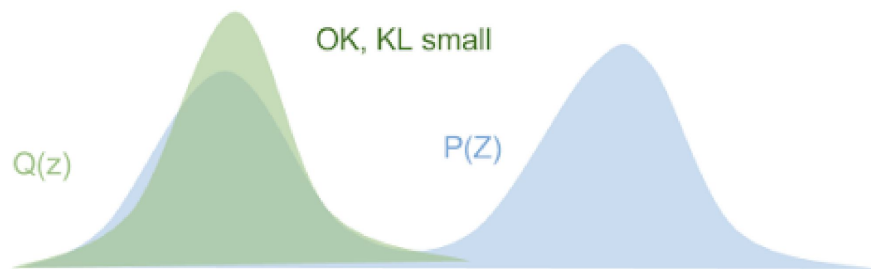


Figure from Goodfellow

GAN extensions

$$D_{JSD}[p, q] = \frac{1}{2} \left(D_{KL} \left[p, \frac{p+q}{2} \right] + D_{KL} \left[q, \frac{p+q}{2} \right] \right)$$

- If our data are on a **low-dimensional manifold** of a high dimensional space, the model's manifold and the true data manifold can have a **negligible intersection** in practice
- KL divergence is undefined or infinite so the loss function and gradients may not be continuous and well behaved



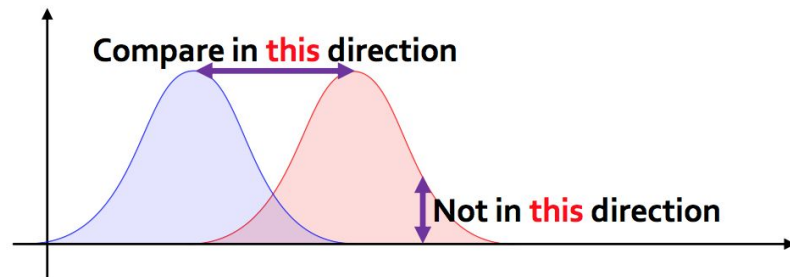
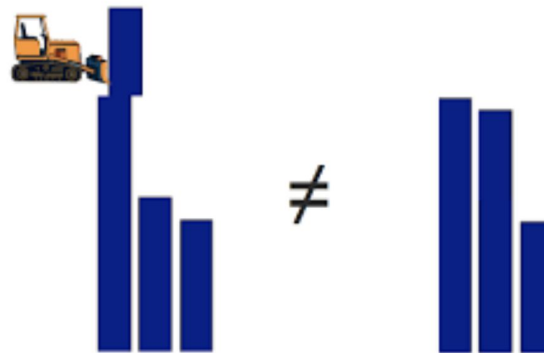
GAN extensions

$$W(P||Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

- $\Pi(P, Q)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are P and Q , respectively
- $\gamma(x, y)$ indicates a plan to transport “mass” from x to y , when deforming P into Q .

The Wasserstein (or Earth-Mover) distance is then the “cost” of the **optimal** transport plan

- The Wasserstein Distance is well defined
- Earth Mover’s Distance: minimum transportation cost for making one pile of dirt in the shape of one probability distribution to the shape of the other distribution



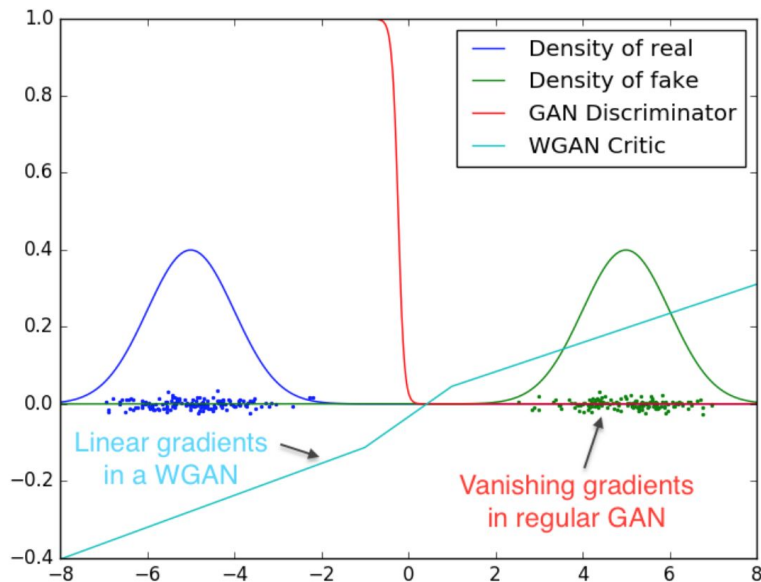
GAN extensions

Wasserstein GAN:

Dual form of Earth Mover's distance

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} [f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta} [f(x)]$$

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D(G(z^{(i)})))$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^m [f(x^{(i)}) - f(G(z^{(i)}))]$	$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f(G(z^{(i)}))$



Optimal discriminator and critic when learning to differentiate two Gaussians. The discriminator of a GAN saturates and results in vanishing gradients. WGAN critic provides clean gradients on all parts of the space. (Arjovsky et al. 2017)

GAN extensions

Wasserstein GAN:

Dual form of Earth Mover's distance

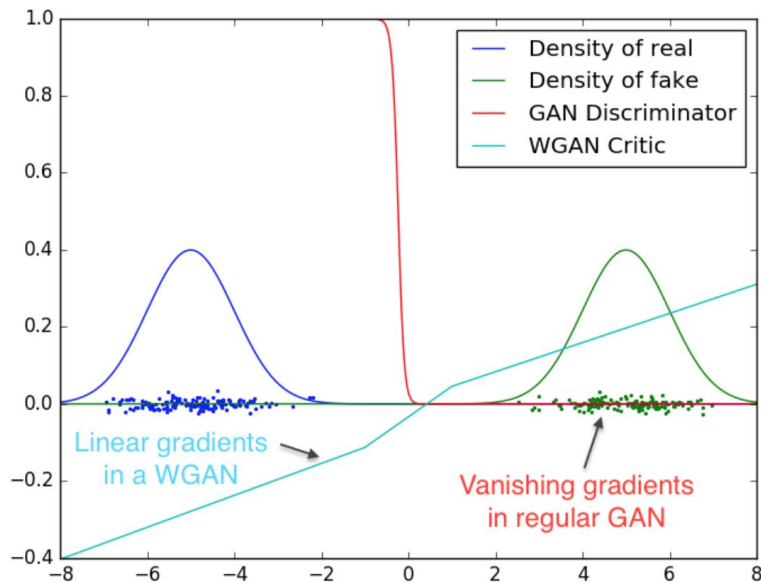
$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} [f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta} [f(x)]$$

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D(G(z^{(i)})))$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^m [f(x^{(i)}) - f(G(z^{(i)}))]$	$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f(G(z^{(i)}))$

$$\|f(x_1) - f(x_2)\| \leq K \|x_1 - x_2\|$$

- Bounded derivative -> function cannot change too quickly
- In practice, use gradient clipping to enforce Lipschitz continuity

[Arjovsky et al., ICML 2017]

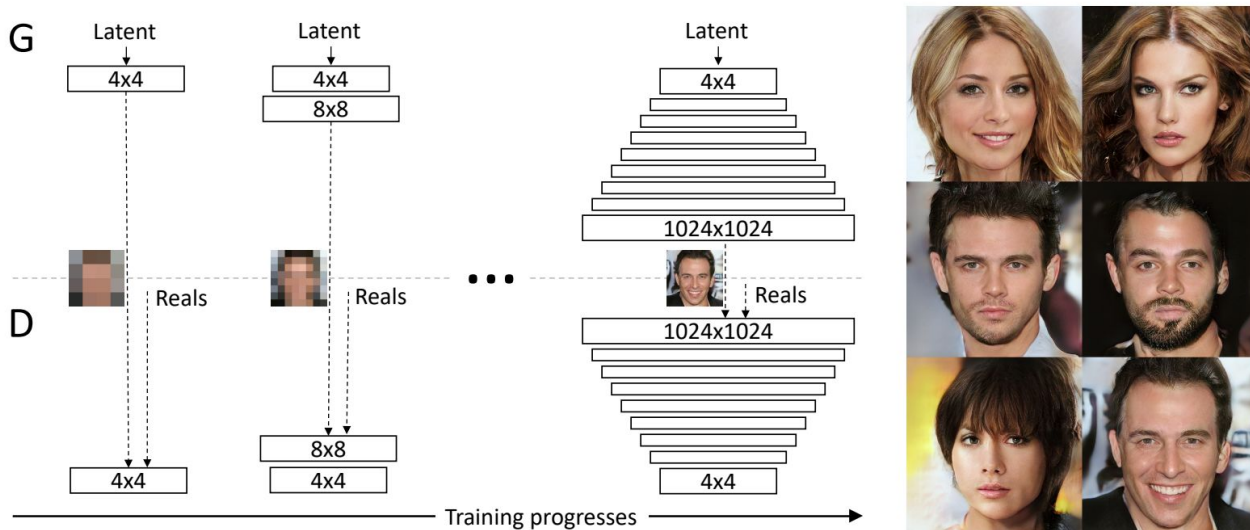


Optimal discriminator and critic when learning to differentiate two Gaussians. The discriminator of a GAN saturates and results in vanishing gradients. WGAN critic provides clean gradients on all parts of the space. (Arjovsky et al. 2017)

GAN extensions

Progressive GAN

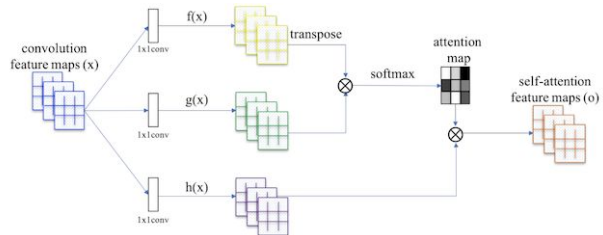
- Starts with low resolution and gradually add layers to generator and discriminator



GAN extensions

Scaling up GANs

1. Self-Attention and Hinge Loss (focus)



2. Class-conditioned generation (accuracy)
3. Spectral normalization (stability)



Examples of Large High-Quality 512×512 Class-Conditional Images Generated by BigGAN.
Taken from: Large Scale GAN Training for High Fidelity Natural Image Synthesis.

GAN extensions

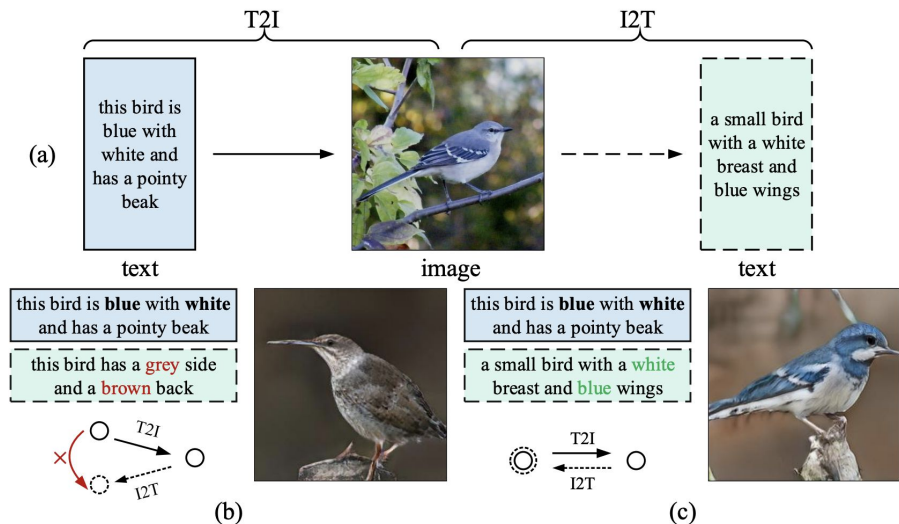
Lots of current work:

- Better loss functions (see Lucic et al., 2018 for a large-scale comparison)
- Optimization tricks (Tim Salimans et al. 2016, Soumith Chintala and more)
- Better evaluation metrics
 - Inception Score (IS) (Salimans et al 2016)
 - Frechet Inception Distance (FID) (Heusel et al. 2017)
 - Precision, Recall and F1 (Lucic et al. 2018)
- Applications:
 - Text style transfer (Shen et al., NeurIPS 2017; Zhao et al., ICML 2018; Li et al., AAAI 2020)
 - Cross-modal generation

GAN applications

Image generation from text

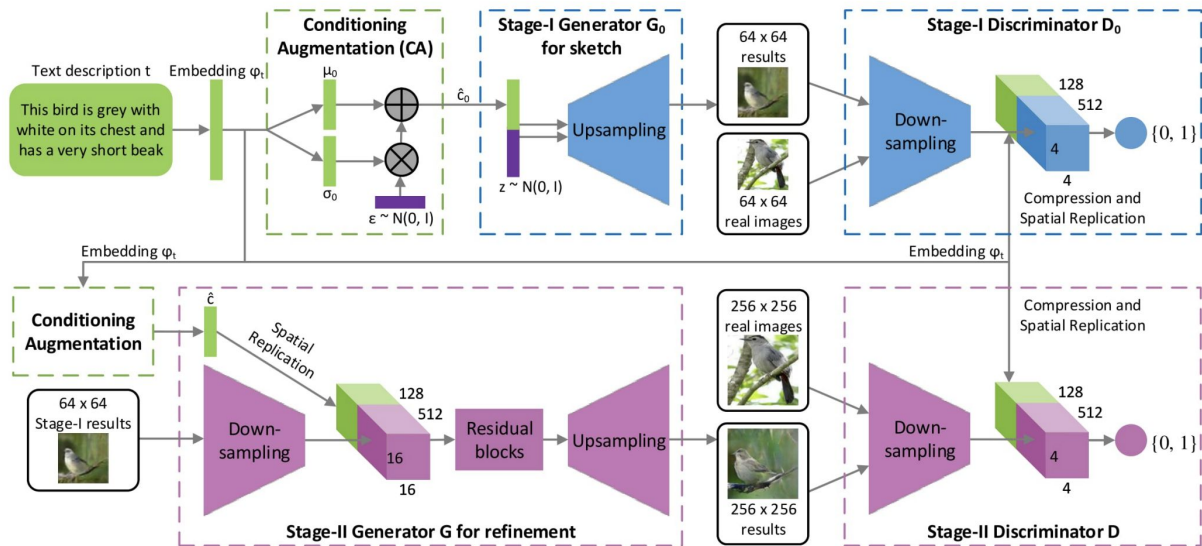
- MirrorGAN: text to image via **redescription**
- Bears resemblance to conditional GANs for with text as context variable
- CycleGAN: ensuring cycle consistency of generated outputs



GAN applications

Image generation from text

- StackGAN: generation over multiple stages
- Stage 2 **refines** stage 1



GAN applications

Image generation from text

- StackGAN: generation over multiple stages
- Stage 2 **refines** stage 1



GAN applications

Semantically Multi-modal Image Synthesis

- Generating based on real image and new semantic mask
- Disentangled latent code controls semantics (e.g. clothes, hair, face, pants)

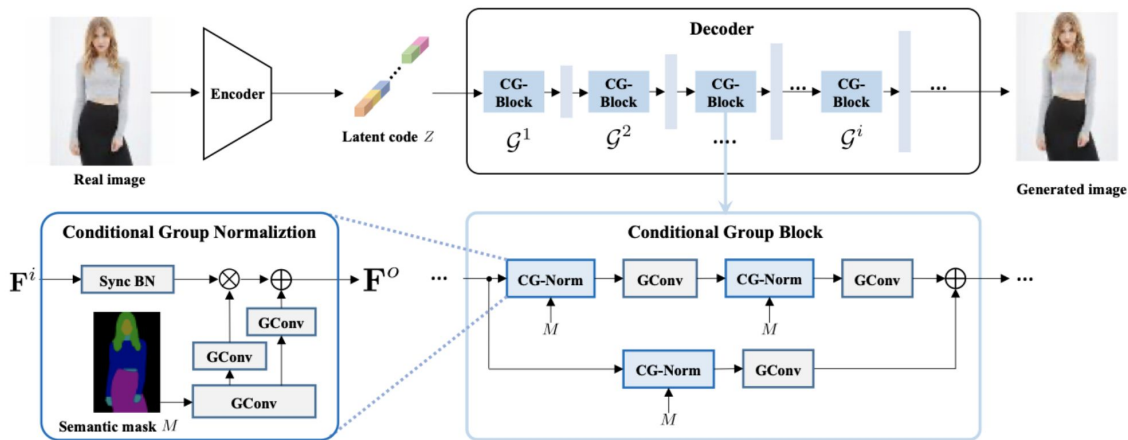


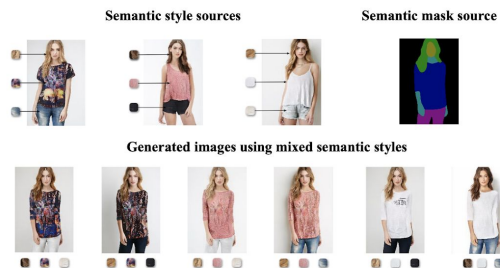
Figure 2: Architecture of our generator (GroupDNet). “GConv” means group convolution and “Sync BN” represents synchronized batch normalization. \mathcal{G}^i is the group number of i -th layer. Note normally $\mathcal{G}^i \geq \mathcal{G}^{i+1}$ for $i \geq 1$ for GroupDNet.

GAN applications

- Semantically Multi-modal Image Synthesis



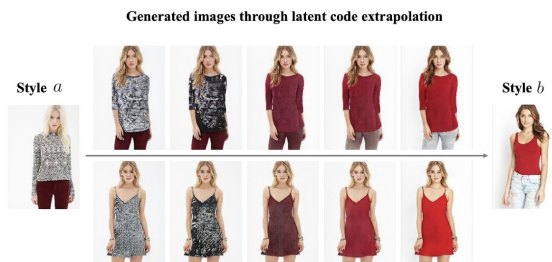
(a) Semantically multi-modal image synthesis



(b) Appearance mixture



(c) Semantic manipulation

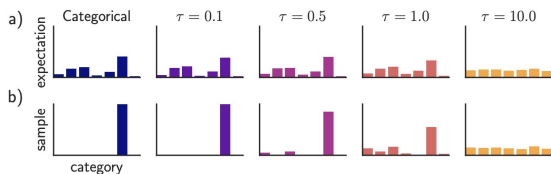


(d) Style morphing

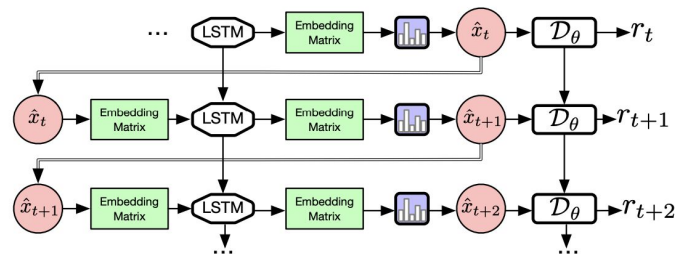
GAN applications

GANs for text generation

1. Text data is discrete
 - Discriminator gradient does not exist for samples from categorical distribution
 - Gradient sparse due to large dictionary size
2. Text is sensitive to noise (small disturbances easily alters the meaning of text)
3. Sparse discriminator feedback (feedback only makes sense on full sentences)



Gumbel-softmax to approximate samples from the categorical generator distribution (Jang et al. 2016)

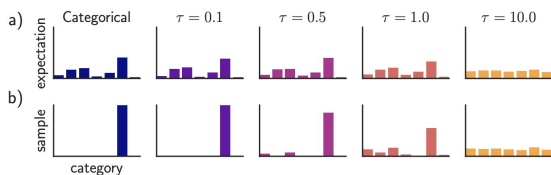


Dense feedback with **policy gradients** and **reward** signals (d'Áutume et al. 2019)

GAN applications

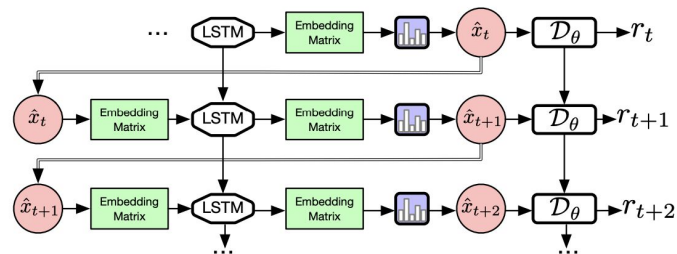
GANs for text generation

1. Text data is discrete
 - Discriminator gradient does not exist for samples from categorical distribution
 - Gradient sparse due to large dictionary size
2. Text is sensitive to noise (small disturbances easily alters the meaning of text)
3. Sparse discriminator feedback (feedback only makes sense on full sentences)



Gumbel-softmax to approximate samples from the categorical generator distribution (Jang et al. 2016)

More next week!



Dense feedback with **policy gradients** and **reward** signals (d'Áutume et al. 2019)

Summary: generative models

VAEs

- Relatively easier to train
- Explicit inference network $q(z|x)$
- More blurry images (due to reconstruction)



Encoder



Decoder



Discriminator

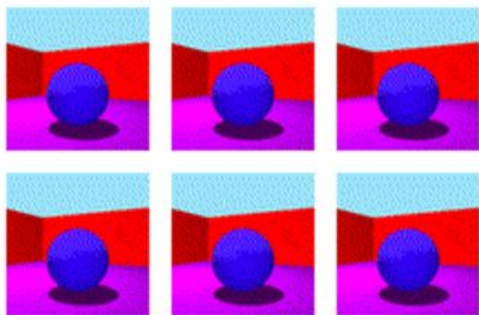
real/fake

- Adversarial autoencoders (Makhzani et al., 2015)
- Autoencoder GANs (Rosca et al., 2017)

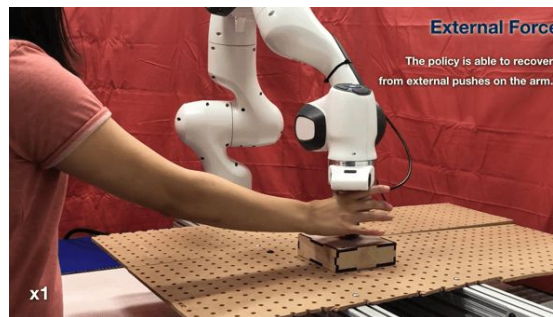
GANs

- Requires many optimization tricks (prone to mode collapse, adversarial objective)
- Implicit generative model (unless using bidirectional GAN)
- Sharper images (due to discriminator)

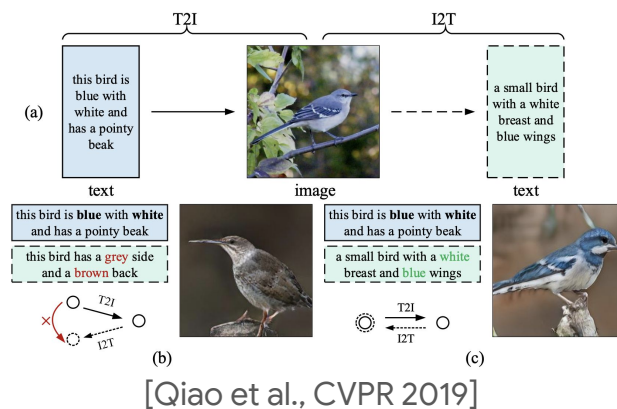
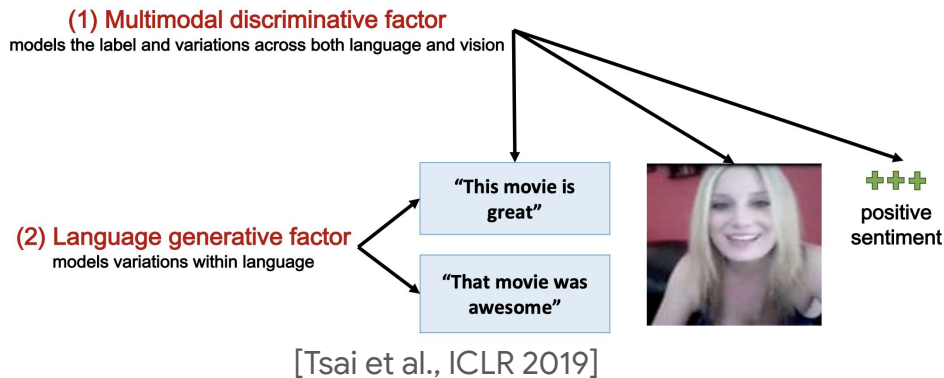
Summary: multimodal applications



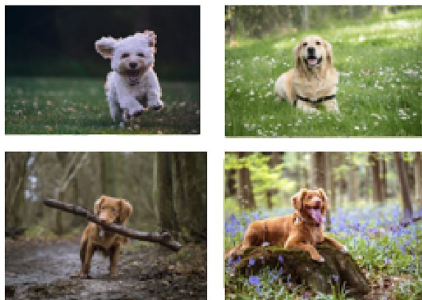
disentanglement_lib
[Locatello et al., ICML 2019]



[Lee et al., ICRA 2019]



Roadmap

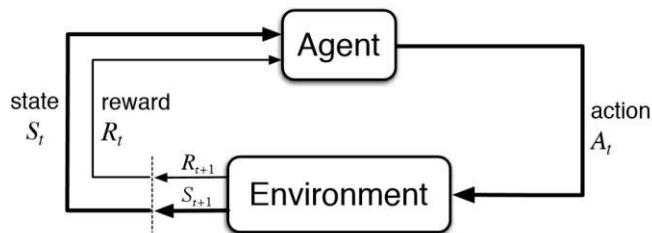


$$x_i \sim P_{\text{data}} \\ i = 1, 2, \dots, n$$

Generative models

- VAE, GANs for continuous data

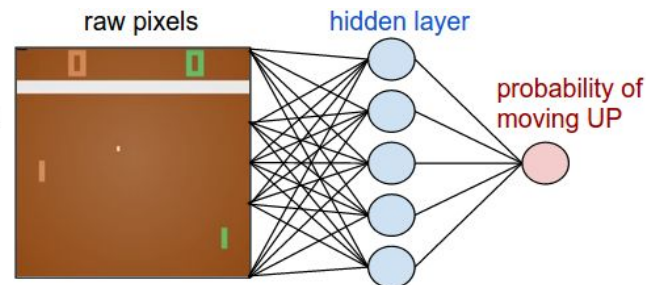
10/22



Reinforcement learning 1

- agent, env, rewards
- value based learning

10/27



Reinforcement learning 2

- policy gradients
- revisit generative models for discrete outputs

10/29