UNIVERSITEIT VAN AMSTERDAM

Unsupervised learning, representation and generative models

Deep Learning

27/11/17

Giorgio Patrini
g.patrini@uva.nl

Overview

- Introduction, manifolds, PCA (Goodfellow's 5.11.3, 13.5)
- Auto-encoders (14)
  - Objective, undercomplete / regularized auto-encoders
  - Denoising auto-encoders, contractive auto-encoders

- **Generative models** (parts of 20)
  - Variational auto-encoder (20.9, 20.10.3)
  - Generative adversarial network (20.10.4, 20.10.6)
  - PixelRNN, models evaluation (20.10.7, 20.14)

Generative models in this lecture

In case you don't know…

- Variational Auto-Encoder (VAE) and Generative Adversarial Network (GAN) are very new research contributions. **Everything in the lecture is at most 4 years old.**
  - VAE [Kingma&Welling'13, Rezende et al.'14]
  - GAN [Goodfellow et al.'13]
- VAE was co-invented at UvA in 2013/2014 !

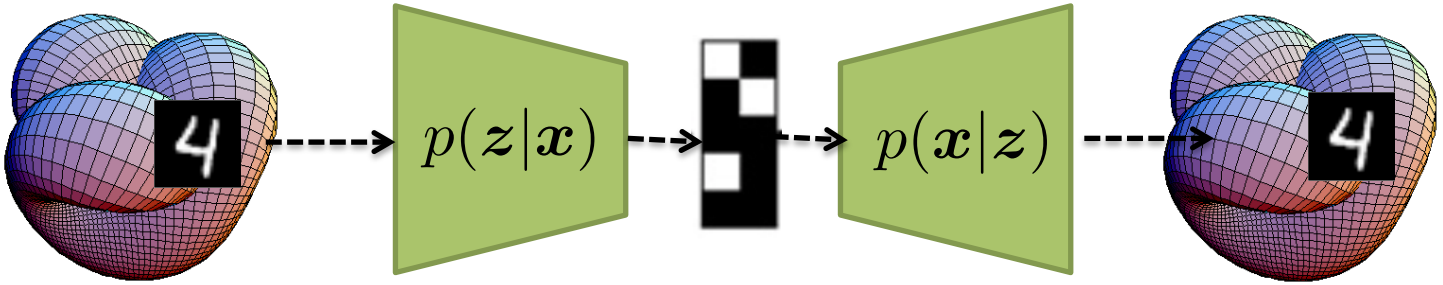
Generative models in this lecture

Therefore:

- Those techniques are changing fast
- We don’t understand everything / don’t know how to fix all current issues yet
- Opportunity to make interesting contributions (e.g. with a good Master thesis)

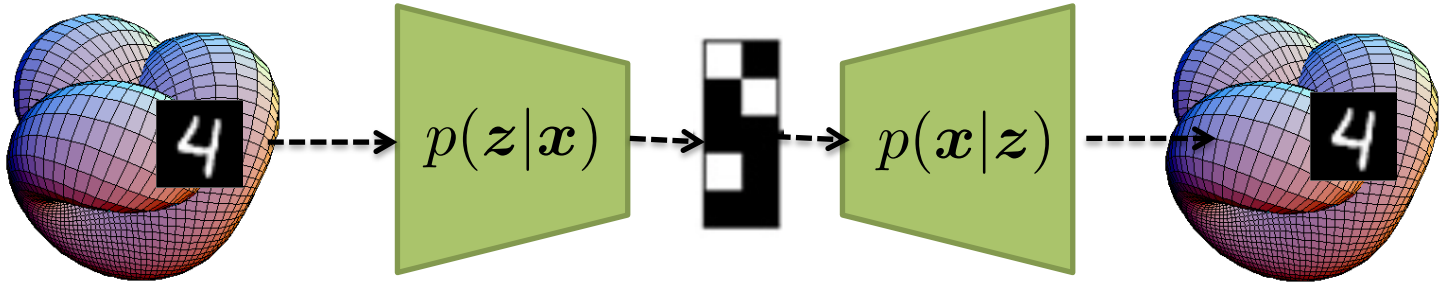
Auto-encoders for generation?

- Training an auto-encoder, we get a decoder $p(\boldsymbol{x}|\boldsymbol{z})$



Auto-encoders for generation?

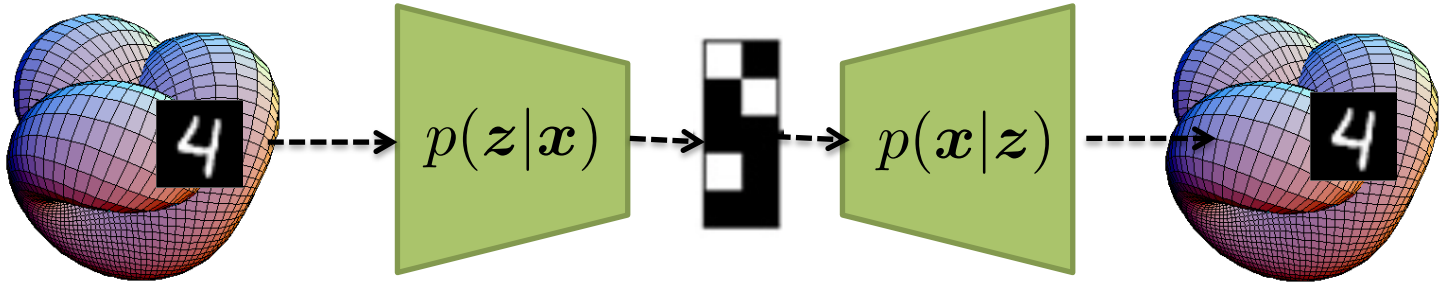
- Training an auto-encoder, we get a decoder $p(\boldsymbol{x}|\boldsymbol{z})$



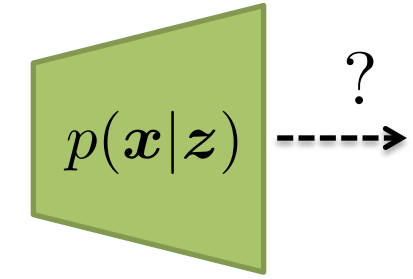
- Can we use it as a generator?

Auto-encoders for generation?

- Training an auto-encoder, we get a decoder $p(\boldsymbol{x}|\boldsymbol{z})$



- Can we use it as a generator?
- Not really… How to sample first $\boldsymbol{z} \sim p(\boldsymbol{z})$? We haven't specified a prior. No guarantee to map onto the manifold for any $\boldsymbol{z}$
- $\boldsymbol{z}$ needs to come from $p(\boldsymbol{z}|\boldsymbol{x})$



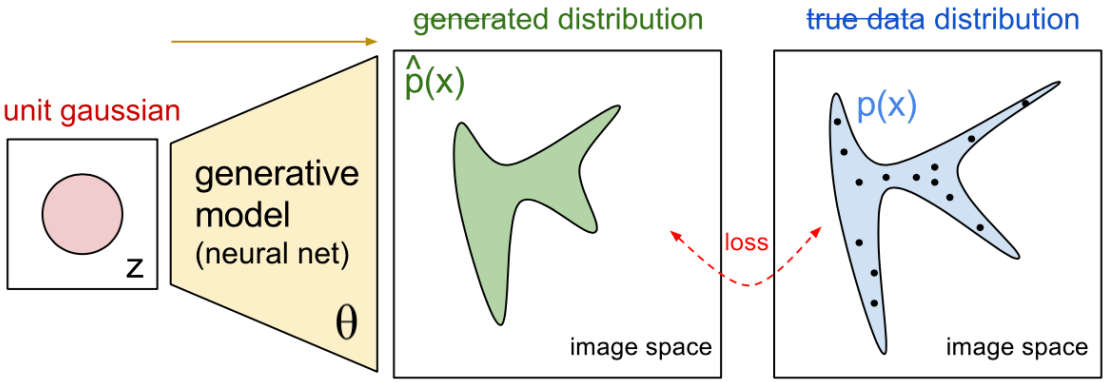
Auto-encoders for generation?

- To be fair: there are ways to extend denoising/ contractive auto-encoders for generation. But they don’t work as well as more recent ideas.

Auto-encoders for generation?

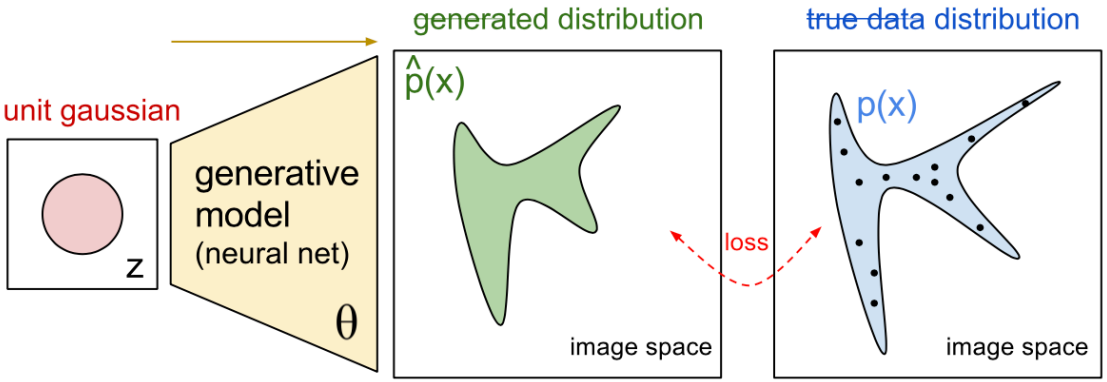
- To be fair: there are ways to extend denoising/ contractive auto-encoders for generation. But they don't work as well as more recent ideas.

- Focus on training a generator:



Auto-encoders for generation?

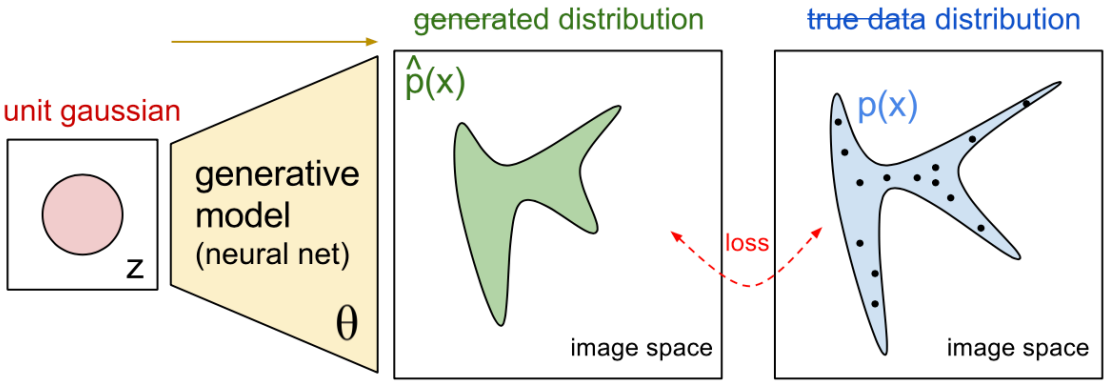
- To be fair: there are ways to extend denoising/ contractive auto-encoders for generation. But they don't work as well as more recent ideas.
- Focus on training a generator:



- VAE and GAN exploit different auxiliary networks for the job; either an encoder (VAE) or a discriminator (GAN)
- Similar to standard auto-encoders, where we train encoder/decoder pair and then keep the encoder.

Manifold hypothesis and generative models

When I cannot create, I cannot understand. R.Feynman



- In order to generate realistic data, i.e. lying on the manifold, the generator must learn a map (chart) for navigating the manifold from a simple latent space.

[https://goo.gl/i5v9VQ OpenAI blog]

Are neural networks capable of generation?

- An experiment of **supervised** learning demonstrated it is possible to navigate the “chairs manifold” with a CNN [Dosovitskiy et al.’15]

Are neural networks capable of generation?

- An experiment of **supervised** learning demonstrated it is possible to navigate the “chairs manifold” with a CNN [Dosovitskiy et al.’15]
- Use a 3D graphical engine to generate chairs by parameters such as: chair type, rotations, etc.
- Then train a CNN (with de-convolutions) as a chair generator from the engine coordinates

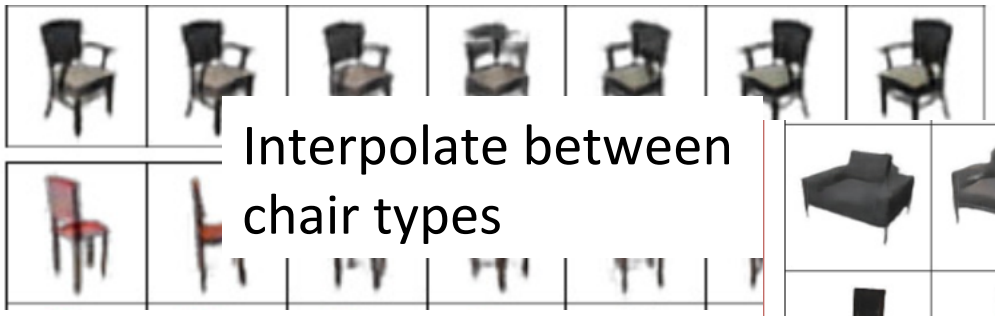
Are neural networks capable of generation?

- An experiment of **supervised** learning demonstrated it is possible to navigate the “chairs manifold” with a CNN [Dosovitskiy et al.’15]
- Use a 3D graphical engine to generate chairs by parameters such as: chair type, rotations, etc.
- Then train a CNN (with de-convolutions) as a chair generator from the engine coordinates

Interpolate between angles

Are neural networks capable of generation?

- An experiment of **supervised** learning demonstrated it is possible to navigate the “chairs manifold” with a CNN [Dosovitskiy et al.’15]
- Use a 3D graphical engine to generate chairs by parameters such as: chair type, rotations, etc.
- Then train a CNN (with de-convolutions) as a chair generator from the engine coordinates



Interpolate between chair types

Overview

- Introduction, manifolds, PCA (Goodfellow's 5.11.3, 13.5)
- Auto-encoders (14)
  - Objective, undercomplete / regularized auto-encoders
  - Denoising auto-encoders, contractive auto-encoders

- Generative models (parts of 20)
  - **Variational auto-encoder** (20.9, 20.10.3)
  - Generative adversarial network (20.10.4, 20.10.6)
  - PixelRNN, models evaluation (20.10.7, 20.14)

Variational auto-encoder (VAE)

- A fully probabilistic (Bayesian) view of auto-encoders
- Model the data generating distribution as

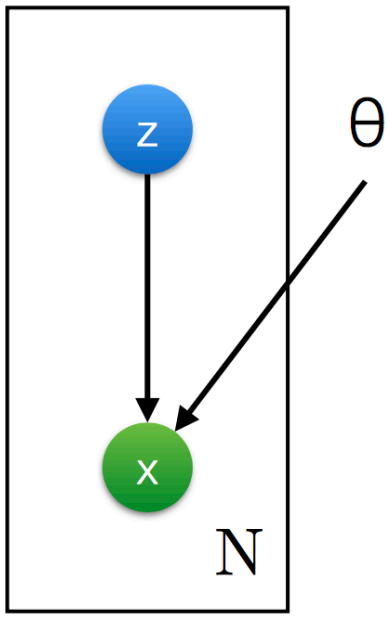
$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$$

Variational auto-encoder (VAE)

- A fully probabilistic (Bayesian) view of auto-encoders
- Model the data generating distribution as

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

- Assume a simple prior $p(\boldsymbol{z})$
- Sampling (generation):
  - $\boldsymbol{z} \sim p(\boldsymbol{z})$
  - $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$

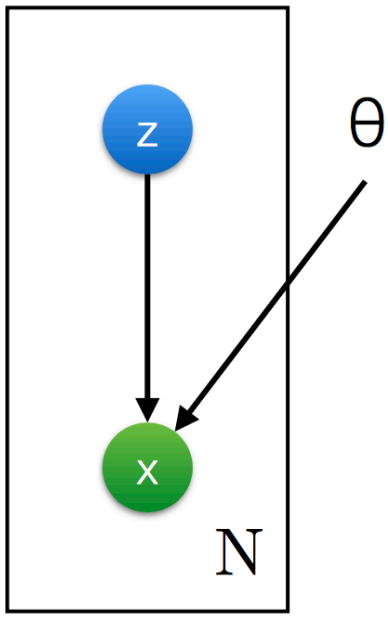


Variational auto-encoder (VAE)

- A fully probabilistic (Bayesian) view of auto-encoders
- Model the data generating distribution as

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

- Assume a simple prior $p(\boldsymbol{z})$
- Sampling (generation):
  - $\boldsymbol{z} \sim p(\boldsymbol{z})$
  - $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$



- $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is the generator (decoder), parametrized by a neural net

Variational auto-encoder (VAE)

Inference (=learning):

- The marginal data distribution: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$

Variational auto-encoder (VAE)

Inference (=learning):

- The marginal data distribution: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$
- **Objective**: max log likelihood

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}$$

Variational auto-encoder (VAE)

Inference (=learning):

- The marginal data distribution: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$
- **Objective**: max log likelihood

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}$$

- In order to compute the likelihood, we need the posterior distribution

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

Variational auto-encoder (VAE)

Inference (=learning):

- The marginal data distribution: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$
- **Objective**: max log likelihood

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}$$

- In order to compute the likelihood, we need the posterior distribution

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

- Unfortunately, the posterior is analytically intractable. This is usual for any interesting model.

Variational auto-encoder (VAE)

- **Bayesian learning**: we model distributions, we don't take a MAP (point estimate) approximation

Variational auto-encoder (VAE)

- **Bayesian learning**: we model distributions, we don't take a MAP (point estimate) approximation
- **Variational inference**: since the posterior is intractable, approximate it with a parametric model:

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$$

Variational auto-encoder (VAE)

- **Bayesian learning**: we model distributions, we don't take a MAP (point estimate) approximation
- **Variational inference**: since the posterior is intractable, approximate it with a parametric model:

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$$

- This turns inference into an optimization problem
- Note: $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ has the probabilistic form of a parametric encoder… hence VAE name

Assume Gaussian prior and approximate posterior

- Typical choice (but not limiting)
  - prior is multivariate Normal $p(\boldsymbol{z}) = N(\mathbf{0}, \boldsymbol{I})$
  - approximate posterior is Gaussian with diagonal covariance, with mean and covariance parametrized by a deterministic encoder $f(\boldsymbol{x})$

Assume Gaussian prior and approximate posterior

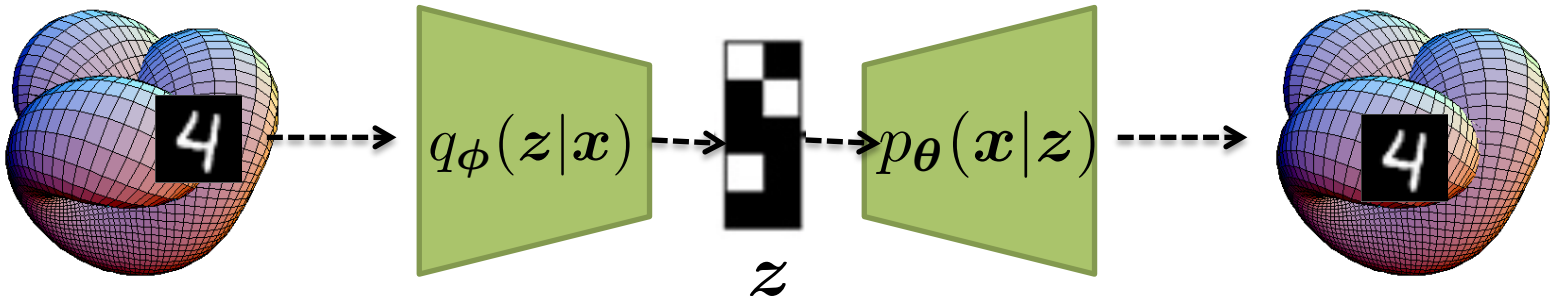
- Typical choice (but not limiting)
  - prior is multivariate Normal $p(\boldsymbol{z}) = N(\mathbf{0}, \boldsymbol{I})$
  - approximate posterior is Gaussian with diagonal covariance, with mean and covariance parametrized by a deterministic encoder $f(\boldsymbol{x})$

$$(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}^2(\boldsymbol{x})) = f(\boldsymbol{x})$$

$$\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{x}), \textsc{Diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$$

Different parameters per point, as function of $f(\boldsymbol{x})$
Although Gaussian, very flexible parametrization.

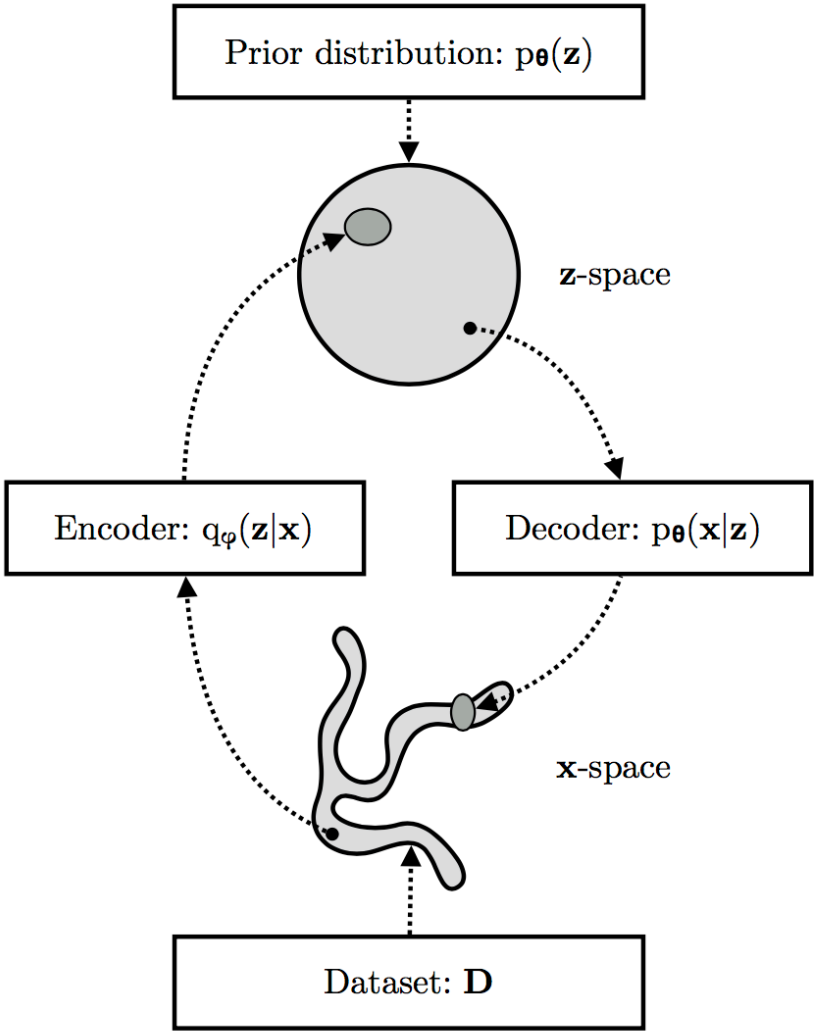
Re-parametrization trick



$$(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\sigma}^2(\boldsymbol{x})) = f(\boldsymbol{x})$$

$$\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{x}), \textsc{Diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$$

Encoder and decoder of VAE



[Kingma PhD thesis]

The VAE objective

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) =$$

The VAE objective

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}$$

The VAE objective

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}
\end{aligned}
$$

The VAE objective

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}
\end{aligned}
$$

The VAE objective

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \\
&= ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) + KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) || p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))
\end{aligned}
$$

By definition of KL

The VAE objective

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}$$

$$= ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) + KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))$$

Defined here by the difference

By definition of KL

$$ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))$$

The variational lower bound

- We obtained a **variational lower bound** (also called Evidence Lower BOund = ELBO) of the log likelihood:

$$\begin{aligned} ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \\ &\leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \end{aligned}$$

- This is a lower bound because KL is non-negative

The variational lower bound

- We obtained a **variational lower bound** (also called Evidence Lower BOund = ELBO) of the log likelihood:

$$ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))$$
$$\leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$$

- This is a lower bound because KL is non-negative
- **Two meanings of maximizing the ELBO**:
  - The lower bound approaches the log likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$

    => better generator
  - The approximate posterior approaches the true posterior

    $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ => better latent representation

The variational lower bound

- Variational objective per data point:

$$
\begin{aligned}
&ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})
\end{aligned}
$$

The variational lower bound

- Variational objective per data point:

$$ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$$
$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$$

- Question: can we solve this by gradient ascent?

The variational lower bound

- Variational objective per data point:

$$ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$$
$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$$

- Question: can we solve this by gradient ascent?
- There is a problem: how to compute gradients of the encoder? (Note: decoder is OK)

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] \neq$$
$$\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \nabla_{\boldsymbol{\phi}} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right]$$

Question: why don't we *just* compute this gradient as is?

- Try implement this and you will understand:

```
# encoding input image
mu = encode(x)

# generate normal random number with given mu, sig
z = tf.random_normal((batch_size, num_dim), mean=mu, stddev=tf.ones_like(mu))

# decode latent
xx = decode(z)
```

- You have samples… now what? No parameters anymore
- Instead, re-parametrize and keep dependencies on *mu*

```
# encoding input image
mu = encode(x)

# generate normal(0, 1) random number
r = tf.random_normal((batch_size, num_dim))

# final latent vector
z = mu + r
```

Code from goo.gl/zCZQm8

Question: why don't we *just* compute this gradient as is?

- Consider a simple univariate Gaussian

$$z \sim N(\mu, 1)$$

- How to compute derivate with respect to μ ?

$$\frac{d}{d\mu} z = \ ?$$

- But we could re-parameterize it and make it possible:

$$z = \mu + \epsilon = N(\mu, 1) \ \text{ with } \ \epsilon \sim N(0, 1)$$

- Now

$$\frac{d}{d\mu} z = 1$$

Re-parameterization trick

- Consider the case of Normal prior and Gaussian variational posterior.
- Original form: cannot compute derivative

$$\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{x}), \mathrm{DIAG}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$$

Re-parameterization trick

- Consider the case of Normal prior and Gaussian variational posterior.
- Original form: cannot compute derivative

$$\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = N(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{x}), \textsc{Diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$$

- Rewrite the variational posterior via a change of variable

$$\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \odot \boldsymbol{\epsilon}$$

$$\boldsymbol{\epsilon} \sim N(\mathbf{0}, I)$$

element-wise product

Re-parameterization trick

- We write the ELBO as

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] =$$

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right]$$

$$\text{with} \quad \boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \odot \boldsymbol{\epsilon}$$

Re-parameterization trick

- We write the ELBO as

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] =$$
$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right]$$
$$\text{with} \quad \boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}(\boldsymbol{x}) \odot \boldsymbol{\epsilon}$$

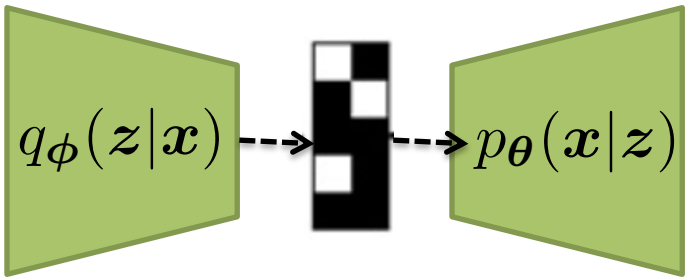
- Now we can access ELBO's derivatives by a Monte Carlo estimate (=sample and average)

$$\nabla_{\boldsymbol{\phi}}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] =$$
$$\mathbb{E}_{\boldsymbol{\epsilon}}\nabla_{\boldsymbol{\phi}}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right]$$

Re-parameterization trick

back-prop: NO

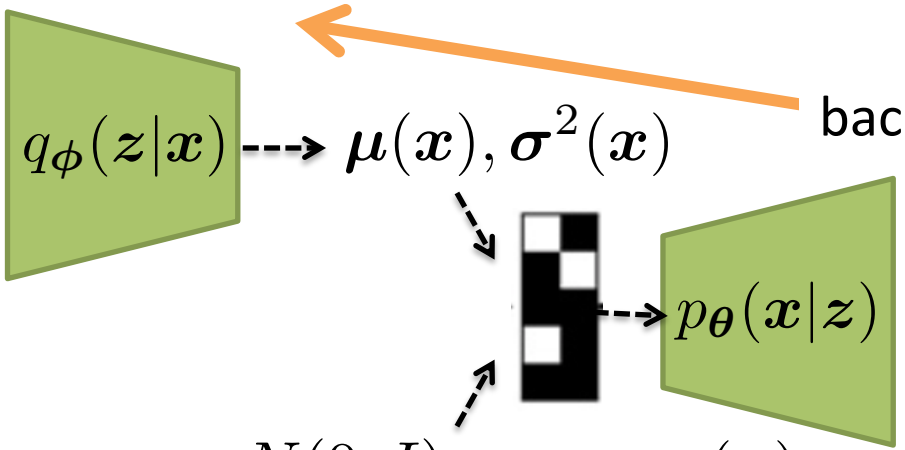
Original graph



$$z \sim N(z; \mu(x), \text{DIAG}(\sigma^2(x)))$$

Re-paremetrized

back-prop: OK



$$\epsilon \sim N(0, I) \qquad z = \mu(x) + \sigma(x) \odot \epsilon$$

a function, after sampling ϵ

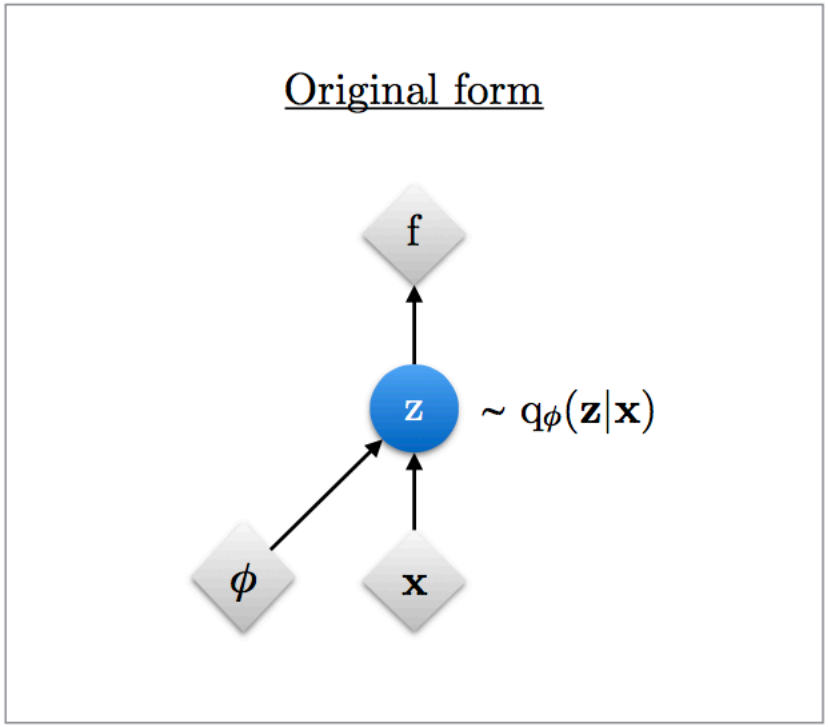
Re-parameterization trick in general

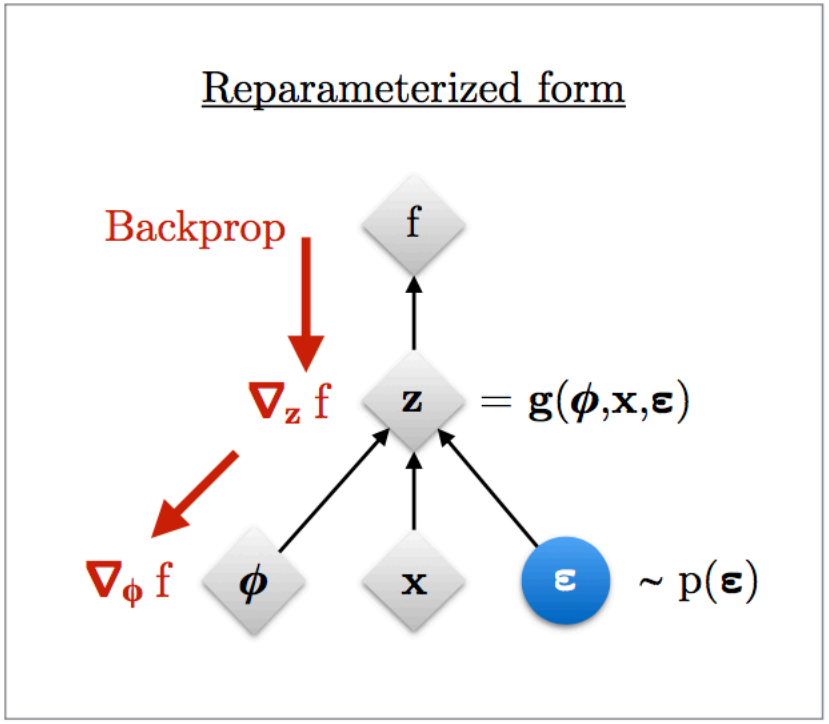
The same trick can be used for several distributions:

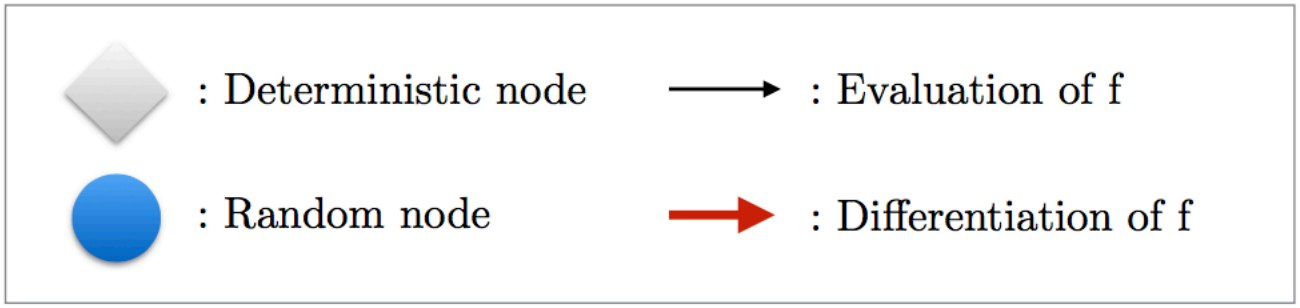
- Location-scale transforms
  - Normal, Laplace, Student t's, Logistic, etc.
- Inverse of CDF
  - Cauchy, Rayleight, Pareto, etc
- Other strategies exist
  - Gamma, Dirichlet, Beta, Chi-Squared, etc

[Kingma&Welling'14]

Re-parameterization trick in general







[Kingma PhD thesis]

REINFORCE: back-prop through discrete variables

- The re-parameterization trick only works when
  - Both encoder and decoder are differentiable
  - **and** latent variables are continuous

REINFORCE: back-prop through discrete variables

- The re-parameterization trick only works when
  - Both encoder and decoder are differentiable
  - **and** latent variables are continuous

- We can use REINFORCE [Williams'92] when those hypotheses are not satisfied.

REINFORCE: back-prop through discrete variables

- The re-parameterization trick only works when
  - Both encoder and decoder are differentiable
  - **and** latent variables are continuous

- We can use REINFORCE [Williams'92] when those hypotheses are not satisfied.
- Idea: approximate an average gradient without computing the derivative. (See 20.9.1)
- Problem: this estimator has high variance.

Training VAE

Data:

$\mathcal{D}$: Dataset

$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model

$p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$: Generative model

Result:

$\boldsymbol{\theta}, \boldsymbol{\phi}$: Learned parameters

$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters

while *SGD not converged* **do**

  $\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)

  $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)

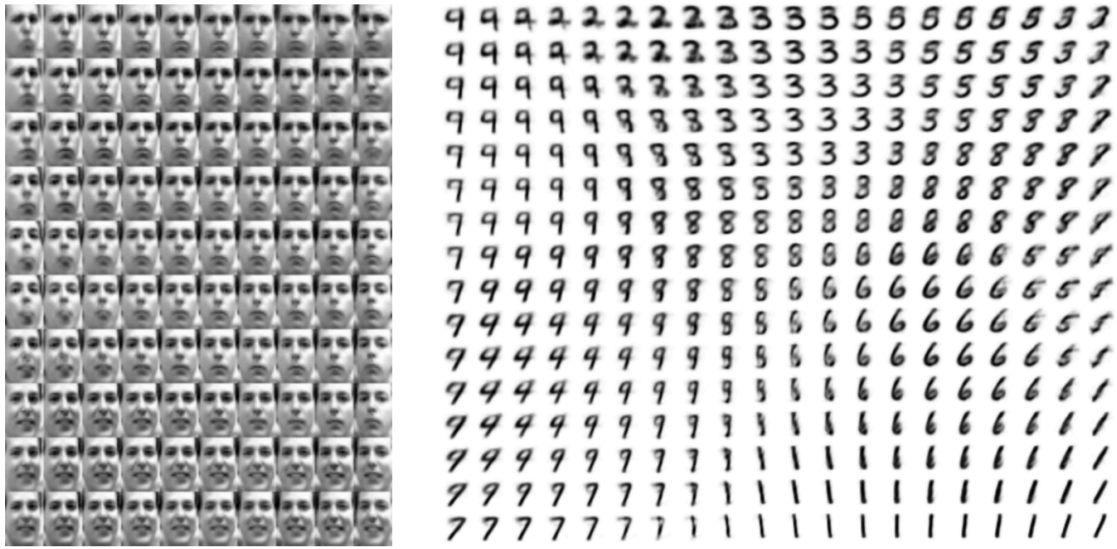
  Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M},\boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M},\boldsymbol{\epsilon})$

  Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer

end

The ELBO's gradients

VAE: navigate the latent space



(a) Learned Frey Face manifold

(b) Learned MNIST manifold

Figure 2.7: Visualizations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables $\mathbf{z}$. For each of these values $\mathbf{z}$, we plotted the corresponding generative $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ with the learned parameters $\boldsymbol{\theta}$.

VAE: random face generation

Trained with convolutions

Application of VAE: natural language synthesis

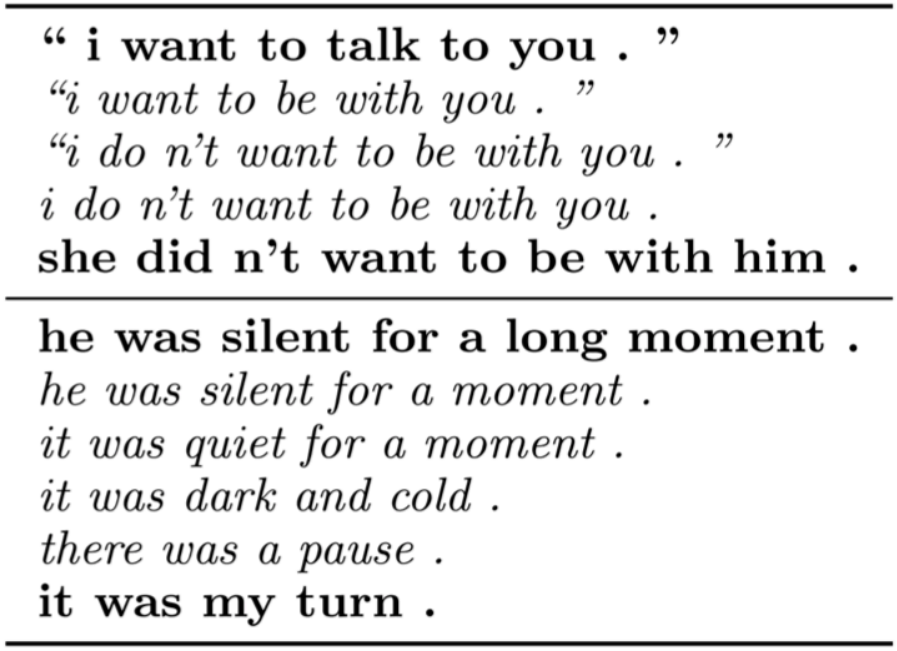


Figure 2.D.2: An application of VAEs to interpolation between pairs of sentences, from [Bowman et al., 2015]. The intermediate sentences are grammatically correct, and the topic and syntactic structure are typically locally consistent.

Application of VAE: image (re)-synthesis

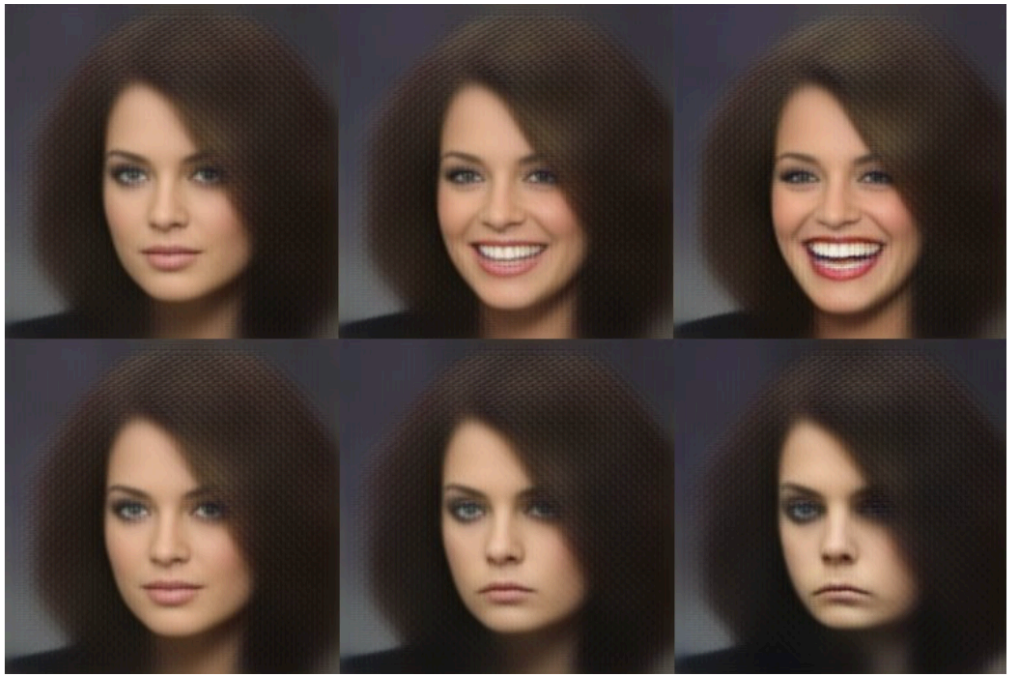
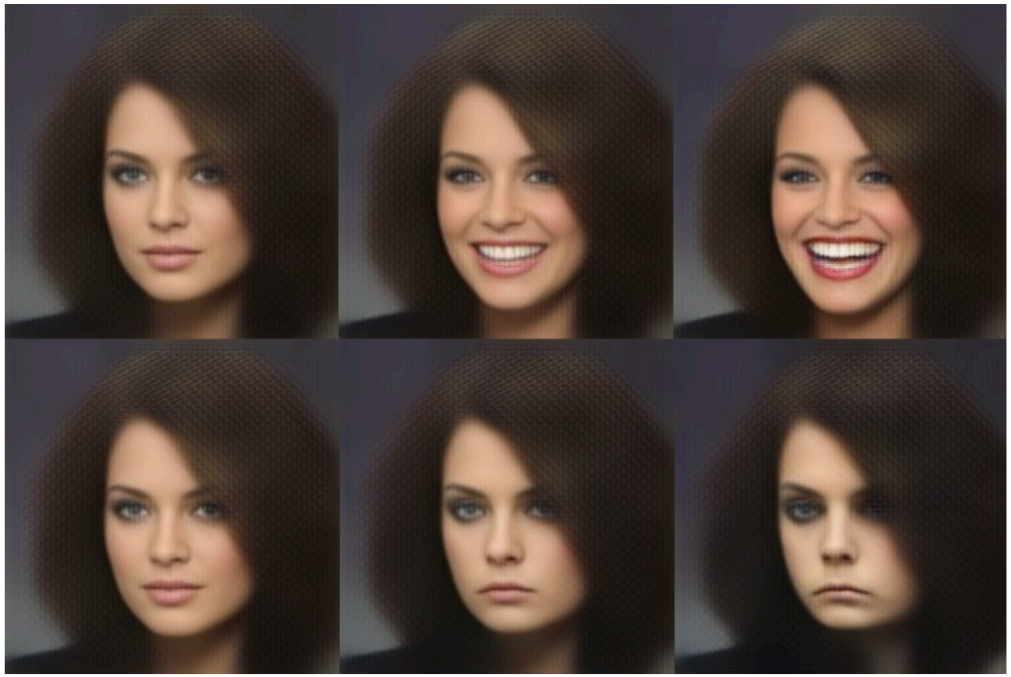


Figure 2.D.3: VAEs can be used for image re-synthesis. In this example by White [2016], an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness. Notice how changing the image along a single vector in latent space, modifies the image in many subtle and less-subtle ways in pixel space.

Application of VAE: image (re)-synthesis



Smile vector:
mean smiling faces –
mean no-smile faces

Latent space arithmetic

Figure 2.D.3: VAEs can be used for image re-synthesis. In this example by White [2016], an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness. Notice how changing the image along a single vector in latent space, modifies the image in many subtle and less-subtle ways in pixel space.

Application of VAE: representation for chemical design

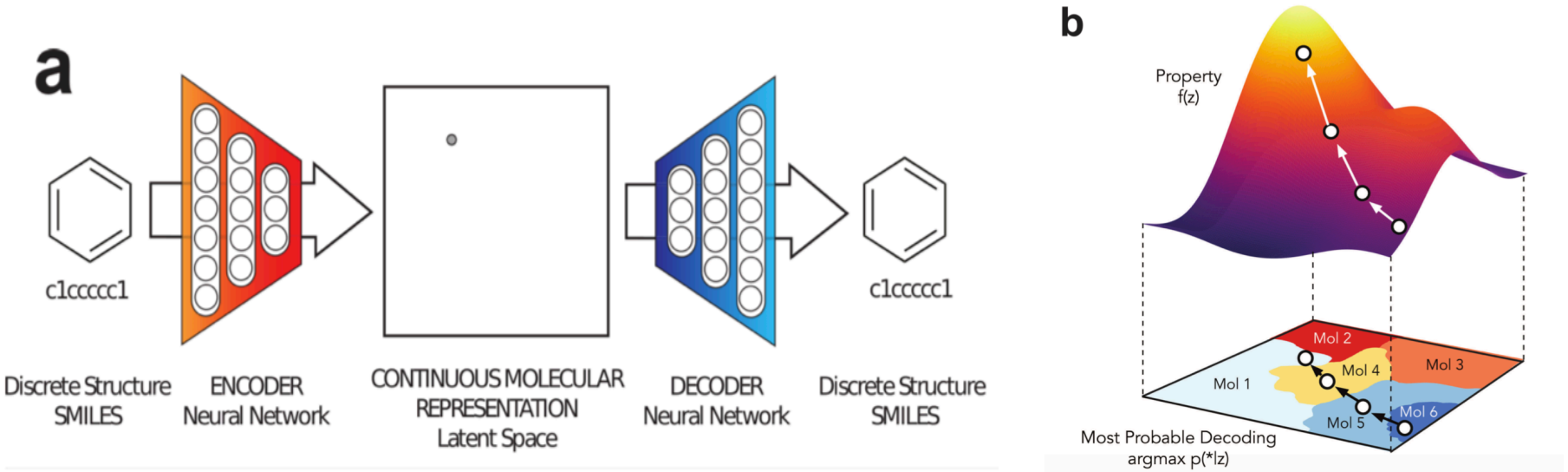


Figure 2.D.1: Example application of a VAE in [Gómez-Bombarelli et al., 2016]: design of new molecules with desired chemical properties. (a) A latent continuous representation $\mathbf{z}$ of molecules is learned on a large dataset of molecules. (b) This continuous representation enables gradient-based search of new molecules that maximizes some chosen desired chemical property given by objective function $f(\mathbf{z})$.

VAE link to auto-encoders

- Per data point objective:

$$
\begin{aligned}
&ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p(\boldsymbol{z}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})
\end{aligned}
$$

VAE link to auto-encoders

- Per data point objective:

$$
\begin{aligned}
&ELBO_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p(\boldsymbol{z}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) - KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))
\end{aligned}
$$

Reconstruction error

Latent space ~ prior

VAE main points

- **Bayesian deep learning**: probabilistic graphical models + neural networks
- **Variational inference**: approximate the intractable posterior with a parametric family by optimization
- **Re-parameterization trick**: allow SGD on computational graphs with stochastic nodes
- **Explicit density model**

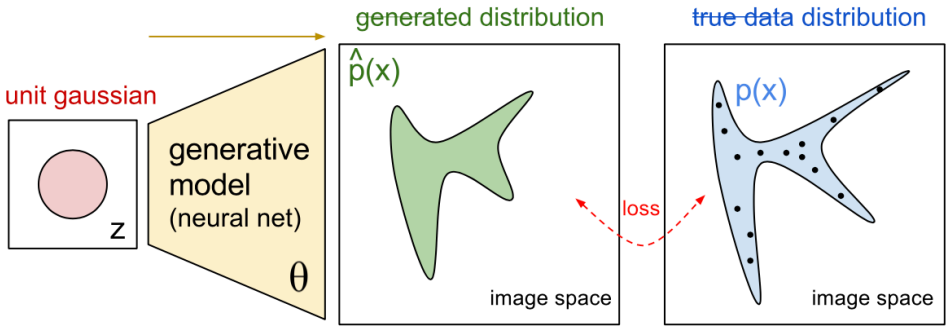
Overview

- Introduction, manifolds, PCA (Goodfellow's 5.11.3, 13.5)
- Auto-encoders (14)
  - Objective, undercomplete / regularized auto-encoders
  - Denoising auto-encoders, contractive auto-encoders

- Generative models (parts of 20)
  - Variational auto-encoder (20.9, 20.10.3)
  - **Generative adversarial network** (20.10.4, 20.10.6)
  - PixelRNN, models evaluation (20.10.7, 20.14)

The idea of GAN

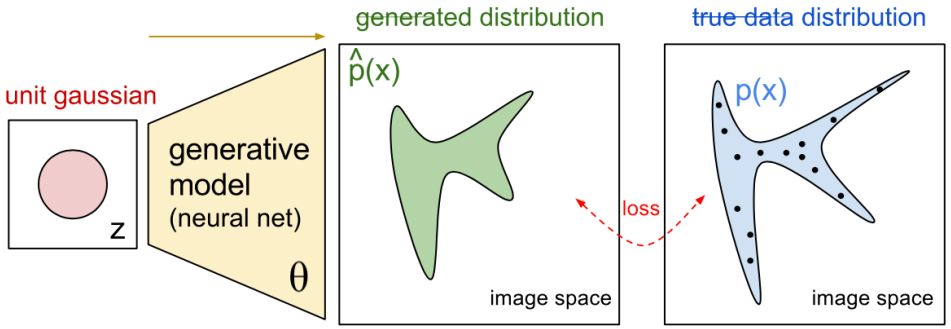
- Can we learn the generator alone (no encoder) ?



- How do we judge its quality?

The idea of GAN

- Can we learn the generator alone (no encoder) ?



- How do we judge its quality?
- The problem seems ill-posed…
  - **Q:** What would be a loss function?
  - **A:** Why don't we **learn the loss** as well?

Generative adversarial network (GAN)

- Two neural networks:
  - A generator $G(\boldsymbol{z}) = \boldsymbol{x}$ takes a random code as input and outputs a (fake) image
  - A discriminator $D(\boldsymbol{x}) \in [0, 1]$ receives an image in input, real or fake (generated), and estimate its probability to be real

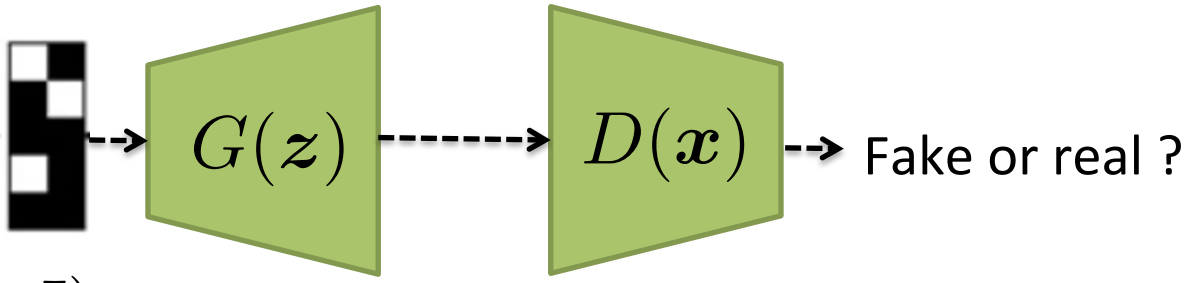
[Goodfellow et al.'14]

Generative adversarial network (GAN)

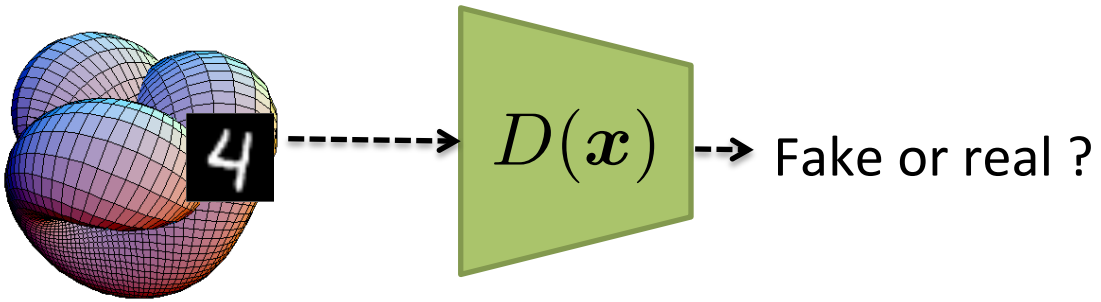
- Two neural networks:
  - A generator $G(\boldsymbol{z}) = \boldsymbol{x}$ takes a random code as input and outputs a (fake) image
  - A discriminator $D(\boldsymbol{x}) \in [0, 1]$ receives an image in input, real or fake (generated), and estimate its probability to be real
- Adversarial training:
  - The generators aims to fool the discriminator = generate (fake but) realistic images
  - The discriminator attempts to distinguish fake and real images

[Goodfellow et al.'14]

Adversarial training of GAN



$z \sim N(0, I)$



The objective

$$\operatorname*{argmin}_G \operatorname*{argmax}_D V(G, D)$$

$$V(G, D) =$$
$$= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})} \log(1 - D(G(\boldsymbol{z})))$$
$$= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{x} \sim p_G(\boldsymbol{x})} \log(1 - D(\boldsymbol{x}))$$

- For a fixed G, the loss for D is effectively the binary cross-entropy. That means: D is a binary classifier for fake/real images.

A zero-sum non-cooperative game

- In the language of **game theory**:
  - **Zero-sum**: loss of one player = gain of the adversary
  - **Non-cooperative**: agents do not collaborate but compete
- In GANs, the spaces of the agents' actions are the parameter spaces of generator and discriminator

A zero-sum non-cooperative game

- In the language of **game theory**:
  - **Zero-sum**: loss of one player = gain of the adversary
  - **Non-cooperative**: agents do not collaborate but compete
- In GANs, the spaces of the agents' actions are the parameter spaces of generator and discriminator

- Solution of those problems are the Nash equilibria:
  - At a Nash equilibrium, there is no **unilateral incentive** to move away, because the objective value would be worse
  - In ML terms: we reach convergence of gradient descent for the joint optimization problem

Theory

- **Proposition 1**: For any generator G, the optimal discriminator is

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

- **Theorem 1**: The global optimum is achieved if and only if

$$p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$$

Theory

- **Proposition 1**: For any generator G, the optimal discriminator is

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

- **Theorem 1**: The global optimum is achieved if and only if

$$p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$$

- That means that at optimum:
  - G learns the data distribution $p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$
  - D gives same probability to images from both

$$D^*(\boldsymbol{x}) = 1/2$$

Proof of proposition 1

Fix the generator G. We have:

$$V(G,D) = \int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) dx + \int p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx$$

$$= \int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) + p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx$$

Proof of proposition 1

Fix the generator G. We have:

$$V(G,D) = \int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) dx + \int p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx$$

$$= \int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) + p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx$$

Now consider the function in the integral:

$$y \to a \log(y) + b \log(1 - y)$$

Its maximum is $y = a/(a+b)$

Proof of proposition 1

Fix the generator G. We have:

$$V(G,D) = \int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) dx + \int p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx$$

$$= \int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) + p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) dx$$

Now consider the function in the integral:

$$y \to a \log(y) + b \log(1 - y)$$

Its maximum is $y = a/(a+b)$

Therefore

$$\underset{D}{\operatorname{argmax}}\, V(G,D) = D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

Theory

- **Proposition 1**: For any generator G, the optimal discriminator is

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

- **Theorem 1**: The global optimum is achieved if and only if

$$p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$$

Proof of theorem 1

If $p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$ then $D^*(\boldsymbol{x}) = 1/2$, therefore

$$\min_G V(G, D^*) =$$

$$= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})} \log D^*(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{x} \sim p_G(\boldsymbol{x})} \log(1 - D^*(\boldsymbol{x}))$$

$$= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})} \log(1/2) + \mathbb{E}_{\boldsymbol{x} \sim p_G(\boldsymbol{x})} \log(1/2) = -\log(4)$$

Proof of theorem 1

If $p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$ then $D^*(\boldsymbol{x}) = 1/2$, therefore

$$\min_G V(G, D^*) =$$

$$= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})} \log D^*(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{x} \sim p_G(\boldsymbol{x})} \log(1 - D^*(\boldsymbol{x}))$$

$$= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})} \log(1/2) + \mathbb{E}_{\boldsymbol{x} \sim p_G(\boldsymbol{x})} \log(1/2) = -\log(4)$$

- This is the objective value when $p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$
- We need to prove that this is also the minimum of the objective. (It is a argmin for G)

Proof of theorem 1

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log \frac{2}{2} \cdot \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

Proof of theorem 1

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log \frac{2}{2} \cdot \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \left[\log \frac{2 \cdot p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})} - \log 2 \right]$$

$$= KL\left(p_{\text{data}} \,\middle\|\, \frac{p_{\text{data}} + p_G}{2} \right) - \log 2$$

Proof of theorem 1

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log \frac{2}{2} \cdot \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \left[\log \frac{2 \cdot p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})} - \log 2 \right]$$

$$= KL\left(p_{\text{data}} \,\middle\|\, \frac{p_{\text{data}} + p_G}{2} \right) - \log 2$$

And likewise:

$$\mathbb{E}_{\boldsymbol{x}\sim p_G(\boldsymbol{x})} \log \frac{p_G(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})} = KL\left(p_G \,\middle\|\, \frac{p_{\text{data}} + p_G}{2} \right) - \log 2$$

Proof of theorem 1

Hence the objective is:

$$V(G, D^*) =$$

$$KL\left(p_{\text{data}} \,\middle\|\, \frac{p_{\text{data}} + p_G}{2}\right) + KL\left(p_G \,\middle\|\, \frac{p_{\text{data}} + p_G}{2}\right) - \log 4$$

Full proofs in [Goodfellow et al.'14]

Proof of theorem 1

Hence the objective is:

$$V(G, D^*) =$$

$$KL\left(p_{\text{data}} \,\middle\|\, \frac{p_{\text{data}} + p_G}{2}\right) + KL\left(p_G \,\middle\|\, \frac{p_{\text{data}} + p_G}{2}\right) - \log 4$$

Which is minimized when both

$$p_{\text{data}} = \frac{p_{\text{data}} + p_G}{2} \quad \text{and} \quad p_G = \frac{p_{\text{data}} + p_G}{2}$$

Therefore $p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$ and the minimum value is again $-\log(4)$

Full proofs in [Goodfellow et al.'14]

A word of caution with the result

- We have proved that the generator will fit the distribution of the real data

A word of caution with the result

- We have proved that the generator will fit the distribution of the real data
- But in practice:
  - **Finite sample size**: training set is finite, not the full distribution
  - **Parametric limit**: the generator has limit capacity, i.e. cannot perfectly represent any distribution
  - **Optimization error:** optimizers can get stuck in local optima or never exactly converge to global optima

A word of caution with the result

- We have proved that the generator will fit the distribution of the real data
- But in practice:
  - **Finite sample size**: training set is finite, not the full distribution
  - **Parametric limit**: the generator has limit capacity, i.e. cannot perfectly represent any distribution
  - **Optimization error:** optimizers can get stuck in local optima or never exactly converge to global optima
- *Note: those three sources of approximation are always present in machine learning*

Optimizing the joint objective

---

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

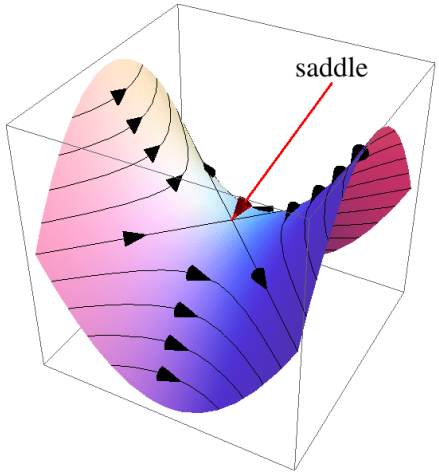
end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Practical issues with training GANs

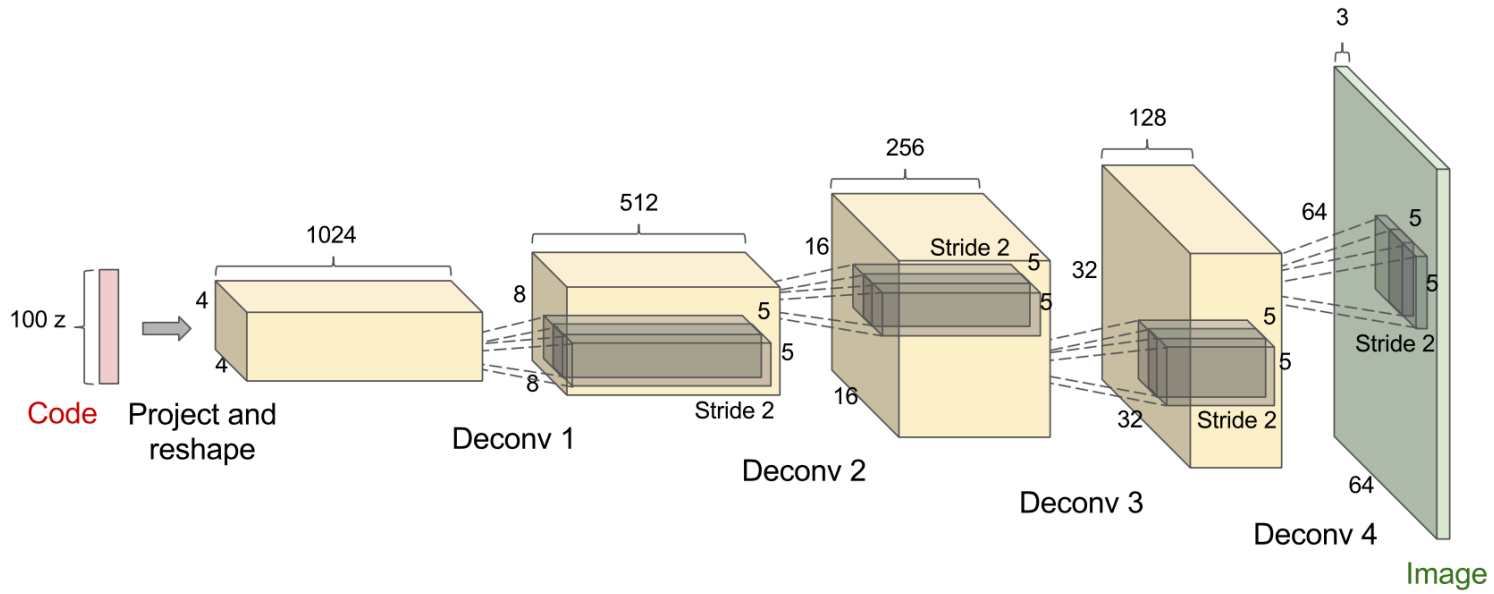


- Difficult to train in practice:
  - Saddle point problem
    - Harder than finding a minima/maxima
  - Balance of updates of D and G:
    - D too weak: no gradient for G to improve
    - D too strong: too hard for G to find a direction to fool it

- The choice of the loss function in the min-max games matters a lot for convergence. **Much research activity on this area.**

Deep Convolutional (DC)GAN

- Convolutional layers for the discriminator.
- De-convolutional layers for the generator:



[Radford et al.'15]

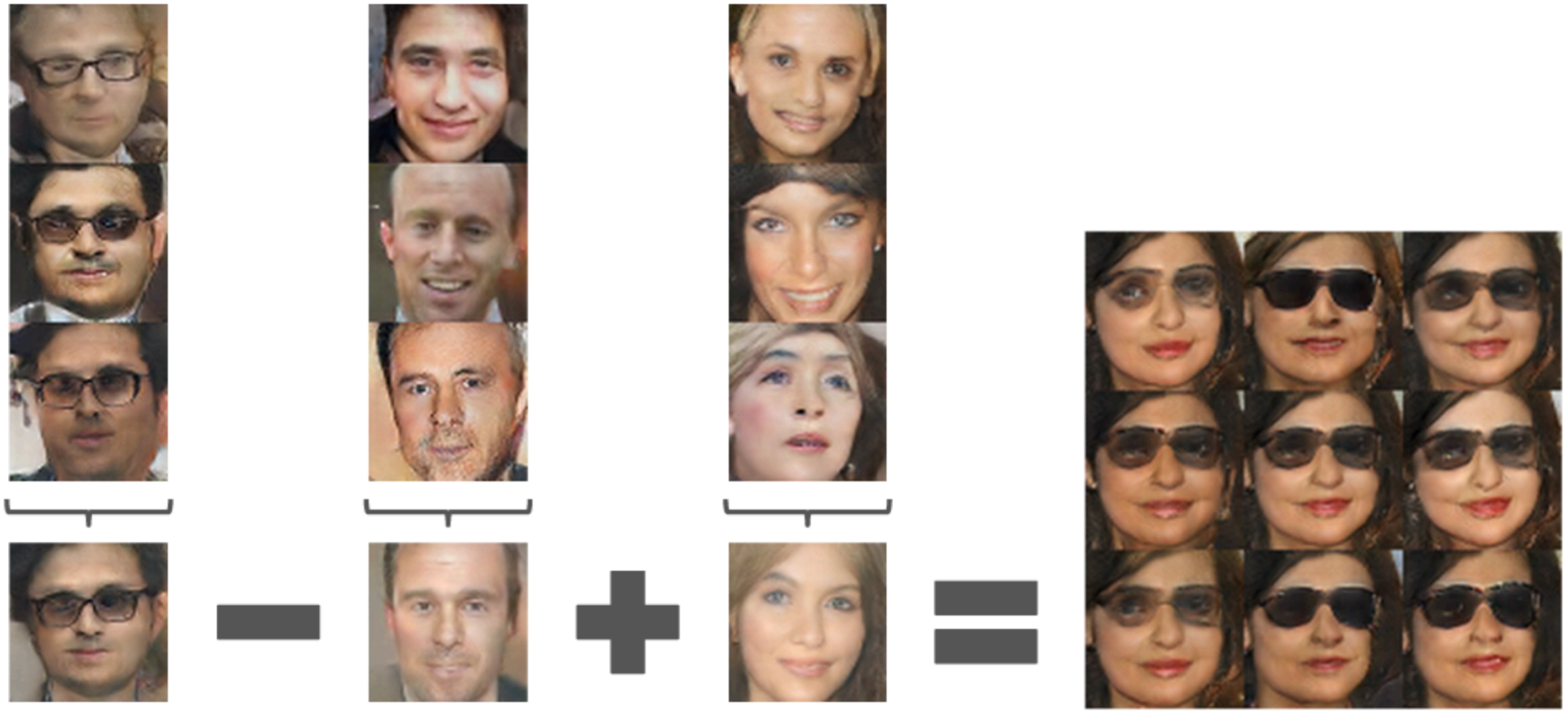
DCGAN: random bedroom generation

[Radford et al.'15]

DCGAN: bedroom space interpolation

[Radford et al.'15]

DCGAN: latent face arithmetic



[Radford et al.'15]

DCGAN: result on ImageNet

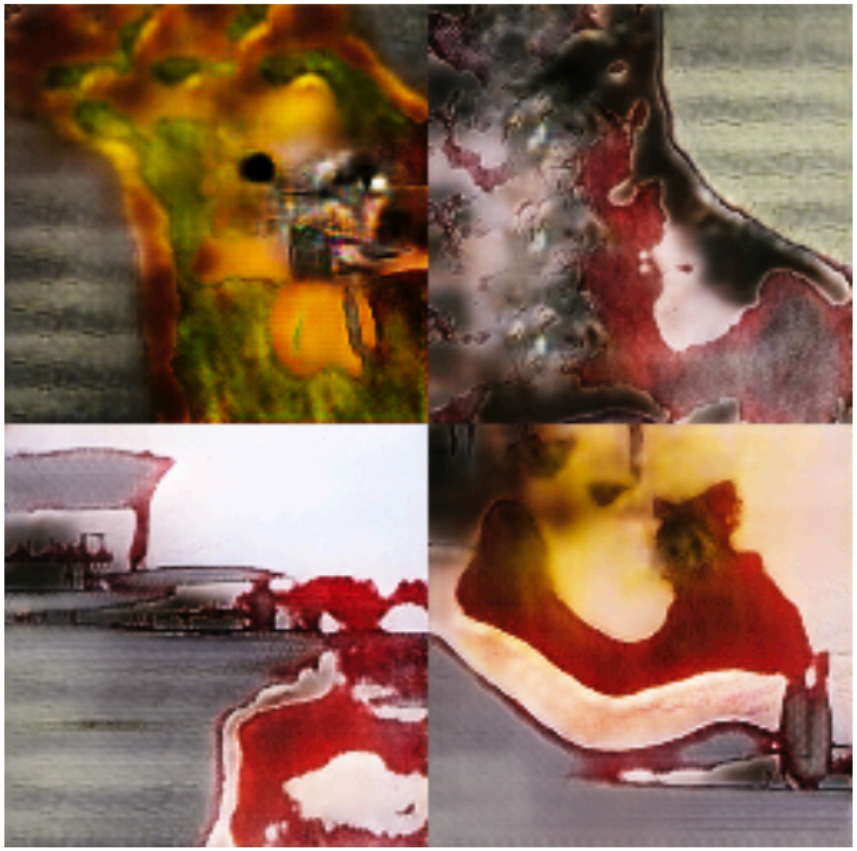


Image from [Salisman et al.'16]

- ImageNet dataset:
  - 1.2M images of 1K classes

Application of GAN: image to image translation (conditional generation)

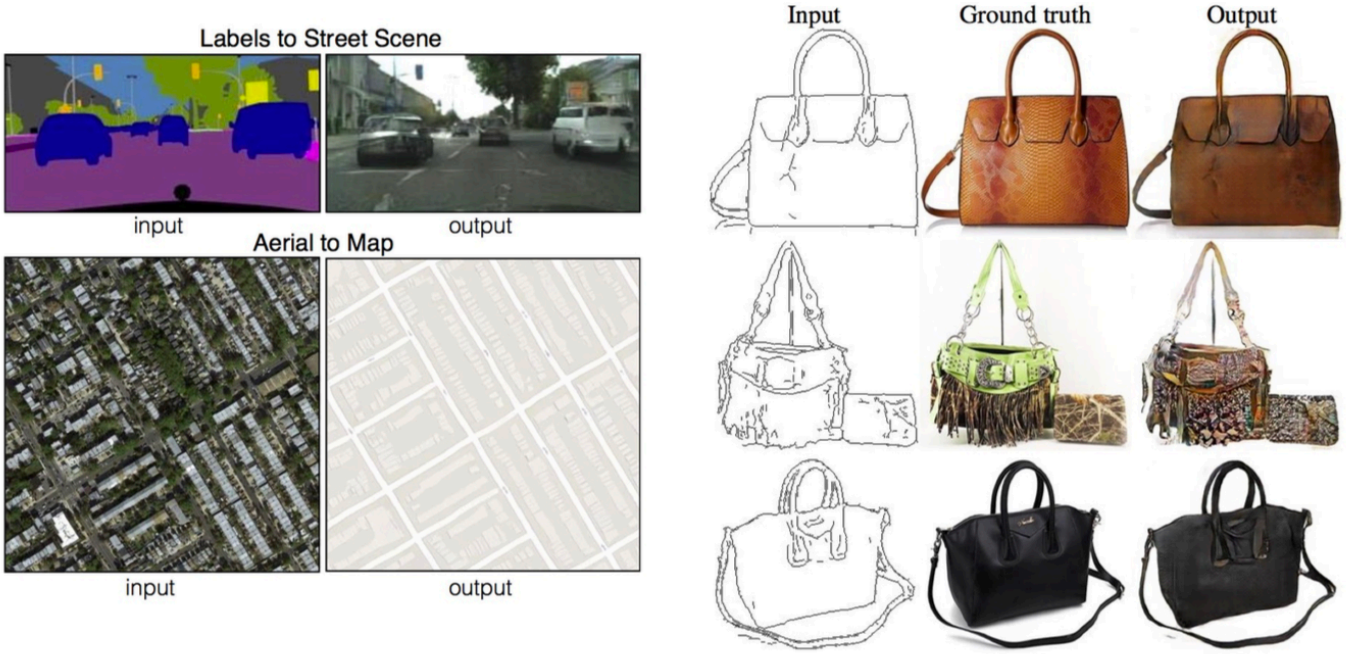


Figure 7: Isola *et al.* (2016) created a concept they called image to image translation, encompassing many kinds of transformations of an image: converting a satellite photo into a map, coverting a sketch into a photorealistic image, etc. Because many of these conversion processes have multiple correct outputs for each input, it is necessary to use generative modeling to train the model correctly. In particular, Isola *et al.* (2016) use a GAN. Image to image translation provides many examples of how a creative algorithm designer can find several unanticipated uses for generative models. In the future, presumably many more such creative uses will be found.

Application of GAN: image to image translation (conditional generation)

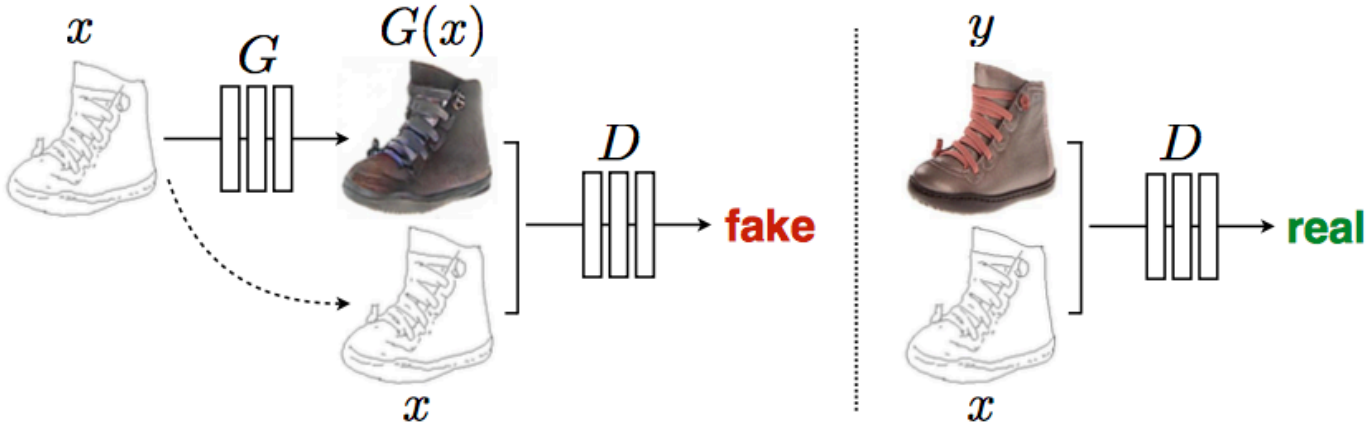


Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

Application of GAN: image super resolution (conditional generation)

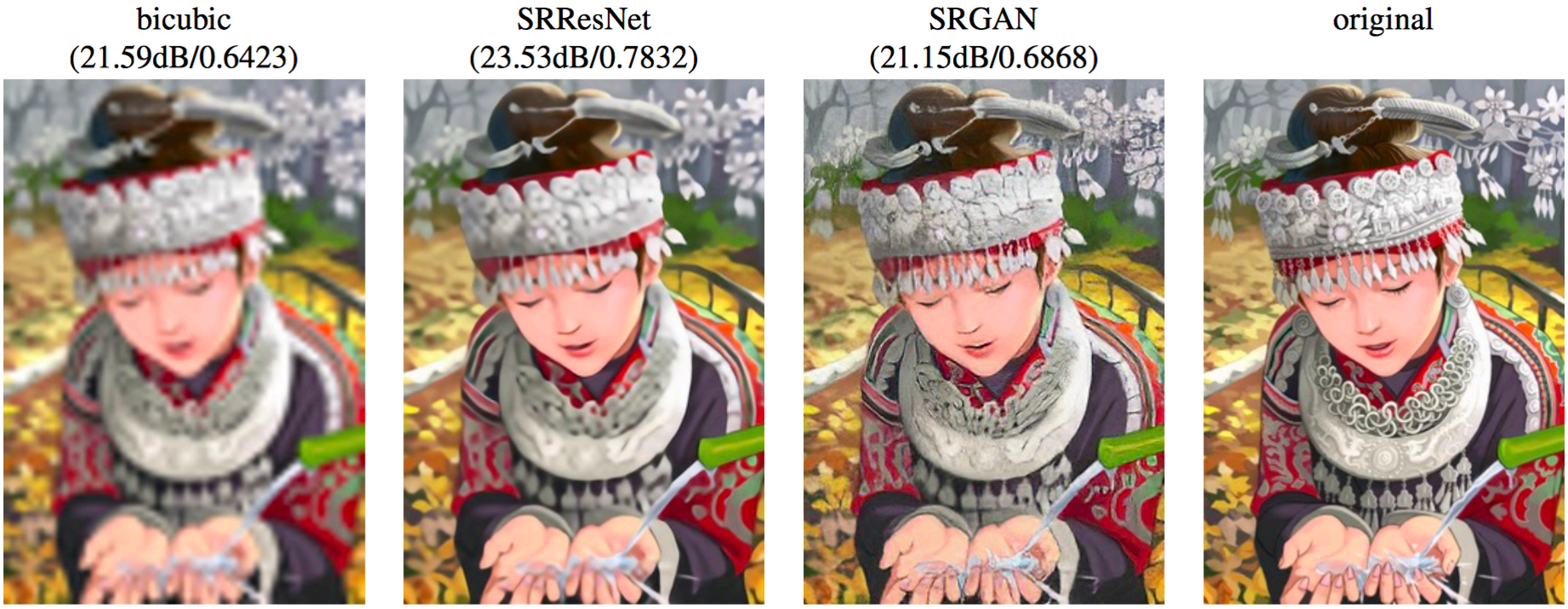


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

[Ledig et al.'17]

GAN main points

- **Adversarial training**: train generator to fool the discriminator
- Form of **consistency**: it is possible in principle to recover the **true** data distribution
- **Implicit density model**: density and likelihood not defined. Instead, optimize “realism”

Application: semi-supervised learning

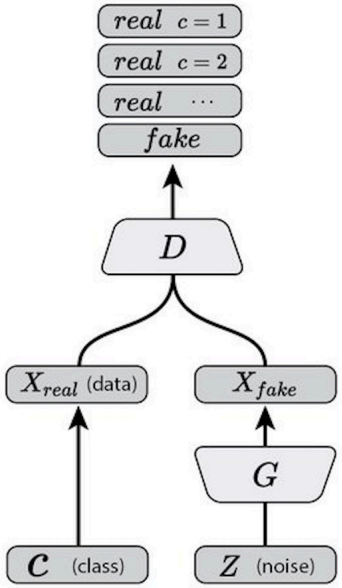
- Both VAE and GAN can be extended for semi-supervised learning:
  - Data $D_L = \{\boldsymbol{X}_L, \boldsymbol{Y}\}$ and $D_U = \{\boldsymbol{X}_U\}$
  - Goal $p(\boldsymbol{y}|\boldsymbol{x})$ (same as with supervised learning)
  - Classification or regression

Application: semi-supervised learning

Main ideas:

- GAN [Salisman et al.'16]: discriminator D is multi-class classifier + fake class. With many unlabelled images, D learns better features simply by setting unlabelled = any class **but** fake.

[Colah https://goo.gl/TgvnH1]

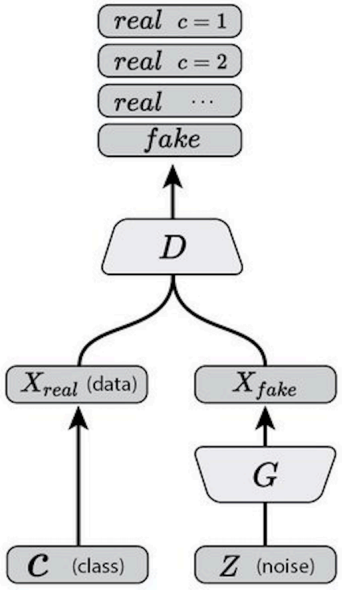


Application: semi-supervised learning

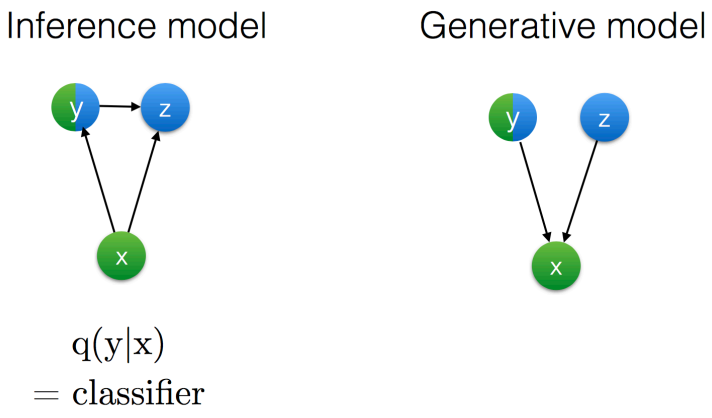
Main ideas:

- GAN [Salisman et al.'16]: discriminator D is multi-class classifier + fake class. With many unlabelled images, D learns better features simply by setting unlabelled = any class **but** fake.

[Colah https://goo.gl/TgvnH1]



- VAE [Kingma et al.'16]: label is part of the latent space. If known, use it to condition the decoder; else, it is inferred. Part of the encoder becomes the classifier: $q(\boldsymbol{y}|\boldsymbol{x})$

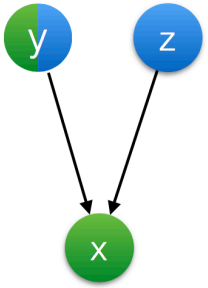


[Kingma https://goo.gl/a76HyH]

Application of semi-supervised VAE: class-conditional generation

Generative model

class-conditional generation; vary z



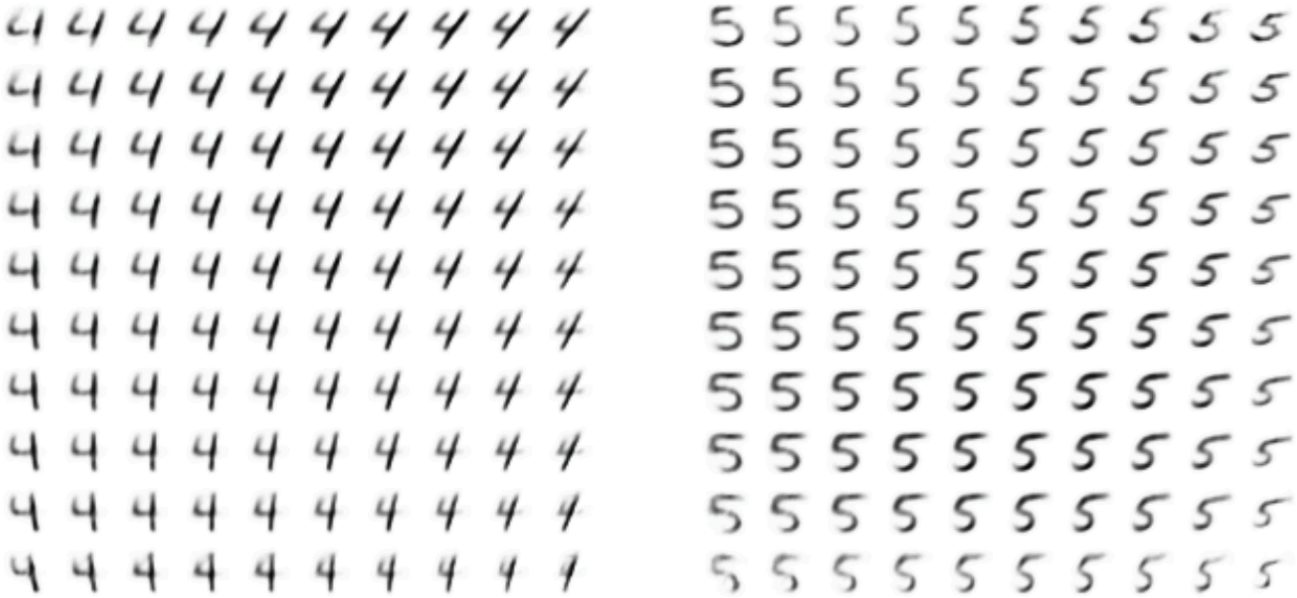
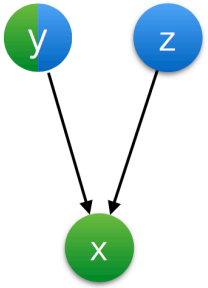


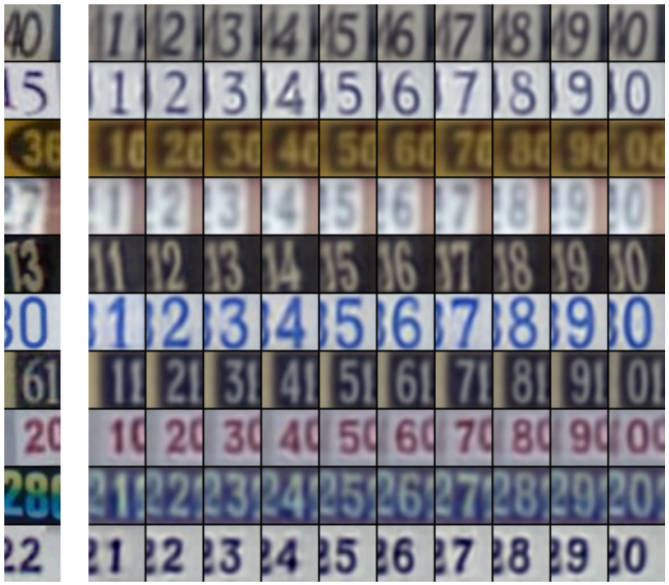
Figure 3.2: Visualization of handwriting styles learned by the model with 2D **z**-space. The images are obtained by fixing the class label, varying the 2D latent variable **z**, and generating the corresponding image **x** through the decoder.

Application of semi-supervised VAE: analogy making

z-fixed generation; vary y. Style vs. content

Generative model





(b) Synthetic analogies of SVHN images.

Figure 3.1: Analogical reasoning with generative semi-supervised models using a high-dimensional **z**-space. The leftmost columns show images from the test set. The other columns show analogical fantasies of **x** by the generative model, where the latent variable **z** of each row is set to the value inferred from the test-set image on the left by the inference network. Each column corresponds to a class label **y**.

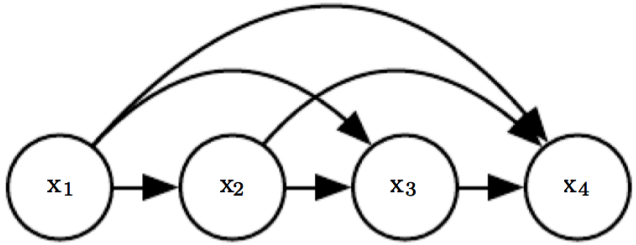
Overview

- Introduction, manifolds, PCA (Goodfellow's 5.11.3, 13.5)
- Auto-encoders (14)
  - Objective, undercomplete / regularized auto-encoders
  - Denoising auto-encoders, contractive auto-encoders

- Generative models (parts of 20)
  - Variational auto-encoder (20.9, 20.10.3)
  - Generative adversarial network (20.10.4, 20.10.6)
  - **PixelRNN, models evaluation** (20.10.7, 20.14)

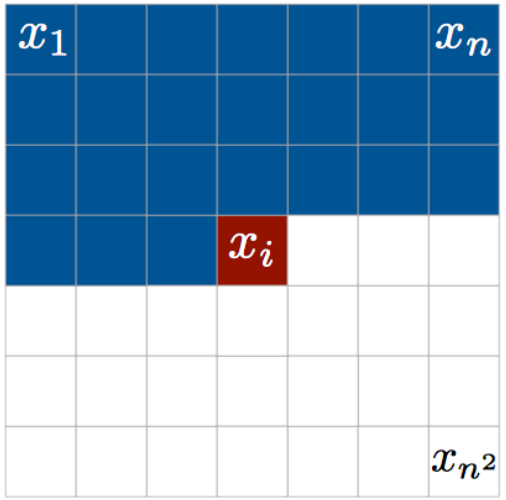
Auto-regressive models

- **Auto-regressive generative models**: generate pixel by pixel, conditionally to the previously generated



$$p(\boldsymbol{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})$$

- **PixelRNN** [van den Oord et al.'16] is auto-regressive
- It runs a deep recurrent neural network pixel by pixel, row by row.



PixelRNN

- No latent variables: $p(\boldsymbol{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})$
- Learn by maximizing **exact** log likelihood

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$$

- Generate by conditional sampling, pixel by pixel

Application of PixelRNN: image completion or inpainting

Figure 1. Image completions sampled from a PixelRNN.

Qualitative comparison

- VAE:
  - Pros: efficient learning and sampling
  - Cons: blurry images
- GAN:
  - Pros: most realistic samples
  - Cons: unstable learning, likely to underfit
- PixelRNN
  - Pros: straightforward to train, exact log likelihood
  - Cons: no latent representation, slow sampling (no parallelization)

Quantitative comparison

In supervised learning is easy:

- Learn on trainset, measure performance (e.g. accuracy, AUC) on testset, compare

Quantitative comparison

In supervised learning is easy:

- Learn on trainset, measure performance (e.g. accuracy, AUC) on testset, compare

With generative models:

- Visual quality… not enough
- Check interpolation in latent space… still a qualitative measure

Quantitative comparison

In supervised learning is easy:

- Learn on trainset, measure performance (e.g. accuracy, AUC) on testset, compare

With generative models:

- Visual quality… not enough
- Check interpolation in latent space… still a qualitative measure
- We can compare the log likelihood on testset $\log p(\boldsymbol{x})$. But :
  - VAE optimizes a lower bound; GAN has no explicit likelihood, but it can be approximated.
  - How to compare different likelihood approximations ?

Quantitative comparison

With generative models:

- Models can underfit and overfit at the same time: memorize cat images (overfit) and completely avoid to learn about dogs (underfit). **Very hard to check.**

Quantitative comparison

With generative models:

- Models can underfit and overfit at the same time: memorize cat images (overfit) and completely avoid to learn about dogs (underfit). **Very hard to check.**
- A heuristic check for overfitting: given a generated image, search in the training set the closest one. Are they “too” similar?

Quantitative comparison

With generative models:

- Models can underfit and overfit at the same time: memorize cat images (overfit) and completely avoid to learn about dogs (underfit). **Very hard to check.**
- A heuristic check for overfitting: given a generated image, search in the training set the closest one. Are they “too” similar?
- Log-likelihood is not necessarily related with quality (realism) of samples [Theis et al.’16]

Quantitative comparison

Evaluation of generative models is an open problem.

A safer approach in applications:

- If the generator is used for a specific application (e.g. semi-supervised classification, super-resolution, etc.), *evaluate the generator by the final task performance, not by itself.*

Question: GAN vs. adversarial examples

Answers by Ian Goodfellow:

- How do they relate? (this may be rather confusing :-)

https://www.quora.com/In-what-way-are-Adversarial-Networks-related-or-different-to-Adversarial-Training

- Is adversarial training (aka the GAN way) effective against adversarial examples?

https://www.quora.com/Is-adversarial-training-effective-against-adversarial-examples-in-general

1 hour of imaginary celebrities

Progressive GAN by NVIDIA [Carras et al.'18]

- https://www.youtube.com/watch?v=36lE9tV9vm0

Material and contact

Lectures material based on:

- Goodfellow's Deep Learning (book)
- Efstratios Gavves's slides from last year
- Larochelle deep learning course https://goo.gl/bvNPDt
- Auto-encoders tutorial in Keras https://goo.gl/9kCxqz
- Durk Kingma PhD thesis (VAE Chapter recommended) https://www.dropbox.com/s/v6ua3d9yt44vgb3/cover_and_thesis.pdf?dl=1
- Goodfellow tutorial on GAN, NIPS 2016 (recommended)

For questions & Master thesis projects: g.patrini@uva.nl

Other references (generative models)

Links:

- Generative models by OpenAI, blog post https://goo.gl/i5v9VQ
- Kingma NIPS15 workshop on VAE https://goo.gl/a76HyH
- Blog post on VAE https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
- DCGAN repo with visual results https://github.com/Newmu/dcgan_code

Papers:

- Kingma & Welling: Auto-encoding variational Bayes, ICLR14
- Rezende et al., Stochastic backpropagation and approximate inference in deep generative models ICML14
- Kingma et al., Semi-supervised learning with deep generative models, NIPS14
- Goodfellow et al., Generative adversarial networks, NIPS14
- Salisman et al., Improved techniques for training GANs, NIPS16
- van den Oord et al., Pixel recurrent neural network, ICML16
- Theis et al., A note on the evaluation of generative models, ICLR16
- Isola et al., Image to image translation, with conditional adversarial networks, CVPR17
- Ledig et al, Photo-realistic single image super-resolution using a generative adversarial network, CVPR17
- Carras et al., Progressive growing of GANs for improved quality, stability, and variation, 2018