Yes,
we GAN.

VLC = $\iiint$ Vision Learning Control $dVdLdC$

Deep Generative Modelling

Jonathon Hare

Vision, Learning and Control
University of Southampton

Introduction

- What is generative modelling and why do we do it?
- Differentiable Generator Networks
- Variational Autoencoders
- Generative Adversarial Networks

Generative Modelling and Differentiable Generator Networks

Recap: Generative Models

- Learn models of the data: $p(\mathrm{x})$
- Learn *conditional* models of the data: $p(\mathrm{x}|\mathrm{y} = y)$
- Some generative models allow the probability distributions to be evaluated explicitly
  - i.e. compute the probability of a piece of data x: $p(\mathrm{x} = x)$

Recap: Generative Models

- Learn models of the data: $p(\mathrm{x})$
- Learn *conditional* models of the data: $p(\mathrm{x}|\mathrm{y} = y)$
- Some generative models allow the probability distributions to be evaluated explicitly
  - i.e. compute the probability of a piece of data x: $p(\mathrm{x} = x)$
- Some generative models allow the probability distributions to be sampled
  - i.e. draw a sample x based on the distribution: $x \sim p(\mathrm{x})$

Recap: Generative Models

- Learn models of the data: $p(\mathrm{x})$
- Learn *conditional* models of the data: $p(\mathrm{x}|\mathrm{y} = y)$
- Some generative models allow the probability distributions to be evaluated explicitly
  - i.e. compute the probability of a piece of data x: $p(\mathrm{x} = x)$
- Some generative models allow the probability distributions to be sampled
  - i.e. draw a sample x based on the distribution: $x \sim p(\mathrm{x})$
- Some generative models can do both of the above

Recap: Generative Models

- Learn models of the data: $p(\mathrm{x})$
- Learn *conditional* models of the data: $p(\mathrm{x}|\mathrm{y} = y)$
- Some generative models allow the probability distributions to be evaluated explicitly
  - i.e. compute the probability of a piece of data x: $p(\mathrm{x} = x)$
- Some generative models allow the probability distributions to be sampled
  - i.e. draw a sample x based on the distribution: $x \sim p(\mathrm{x})$
- Some generative models can do both of the above
  - e.g. a Gaussian Mixture Model is an explicit model of the data using k Gaussians

Recap: Generative Models

- Learn models of the data: $p(\mathrm{x})$
- Learn *conditional* models of the data: $p(\mathrm{x}|\mathrm{y} = y)$
- Some generative models allow the probability distributions to be evaluated explicitly
  - i.e. compute the probability of a piece of data x: $p(\mathrm{x} = x)$
- Some generative models allow the probability distributions to be sampled
  - i.e. draw a sample x based on the distribution: $x \sim p(\mathrm{x})$
- Some generative models can do both of the above
  - e.g. a Gaussian Mixture Model is an explicit model of the data using k Gaussians
    - The likelihood of data x is the weighted sum of the likelihood from each of the k Gaussians

Recap: Generative Models

- Learn models of the data: $p(\mathrm{x})$
- Learn *conditional* models of the data: $p(\mathrm{x}|\mathrm{y} = y)$
- Some generative models allow the probability distributions to be evaluated explicitly
  - i.e. compute the probability of a piece of data x: $p(\mathrm{x} = x)$
- Some generative models allow the probability distributions to be sampled
  - i.e. draw a sample x based on the distribution: $x \sim p(\mathrm{x})$
- Some generative models can do both of the above
  - e.g. a Gaussian Mixture Model is an explicit model of the data using k Gaussians
    - The likelihood of data x is the weighted sum of the likelihood from each of the k Gaussians
    - Sampling can be achieved by sampling the categorical distribution of k weights followed by sampling a data point from the corresponding Gaussian

Why do generative modelling?

- Try to understand the processes through which the data was itself generated
  - Probabilistic latent variable models like VAEs or topic models (PLSA, LDA, ...) for text
  - Models that try to disentangle latent factors like β-VAE

Why do generative modelling?

- Try to understand the processes through which the data was itself generated
    - Probabilistic latent variable models like VAEs or topic models (PLSA, LDA, ...) for text
    - Models that try to disentangle latent factors like β-VAE
- Understand how likely a new or previously unseen piece of data is
    - outlier prediction, anomaly detection, ...

Why do generative modelling?

- Try to understand the processes through which the data was itself generated
  - Probabilistic latent variable models like VAEs or topic models (PLSA, LDA, . . .) for text
  - Models that try to disentangle latent factors like β-VAE
- Understand how likely a new or previously unseen piece of data is
  - outlier prediction, anomaly detection, . . .
- Make 'new' data
  - Make 'fake' data to use to train large supervised models?
  - 'Imagine' new, but plausible, things?

Differentiable Generator Networks

- Generative Modelling is not new; we've known how to make arbitrarily complex probabilistic graphical models for many years.
  - ...But difficult to train and scale to real data, relying on MCMC.

Differentiable Generator Networks

- Generative Modelling is not new; we've known how to make arbitrarily complex probabilistic graphical models for many years.
  - ...But difficult to train and scale to real data, relying on MCMC.
- The past few years has seen major progress along four loose strands:

Differentiable Generator Networks

- Generative Modelling is not new; we've known how to make arbitrarily complex probabilistic graphical models for many years.
  - ...But difficult to train and scale to real data, relying on MCMC.
- The past few years has seen major progress along four loose strands:
  - Invertible density estimation - A way to specify complex generative models by transforming a simple latent distribution with a series of invertible functions.

Differentiable Generator Networks

- Generative Modelling is not new; we've known how to make arbitrarily complex probabilistic graphical models for many years.
  - ...But difficult to train and scale to real data, relying on MCMC.
- The past few years has seen major progress along four loose strands:
  - Invertible density estimation - A way to specify complex generative models by transforming a simple latent distribution with a series of invertible functions.
  - Autoregressive models - Another way to model $p(x)$ is to break the model into a series of conditional distributions: $p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\ldots$

Differentiable Generator Networks

- Generative Modelling is not new; we've known how to make arbitrarily complex probabilistic graphical models for many years.
  - ...But difficult to train and scale to real data, relying on MCMC.
- The past few years has seen major progress along four loose strands:
  - Invertible density estimation - A way to specify complex generative models by transforming a simple latent distribution with a series of invertible functions.
  - Autoregressive models - Another way to model $p(x)$ is to break the model into a series of conditional distributions: $p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\dots$
  - Variational autoencoders - Latent-variable models that use a neural network to do approximate inference.

Differentiable Generator Networks

- Generative Modelling is not new; we've known how to make arbitrarily complex probabilistic graphical models for many years.
  - ...But difficult to train and scale to real data, relying on MCMC.
- The past few years has seen major progress along four loose strands:
  - Invertible density estimation - A way to specify complex generative models by transforming a simple latent distribution with a series of invertible functions.
  - Autoregressive models - Another way to model $p(x)$ is to break the model into a series of conditional distributions: $p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\dots$
  - Variational autoencoders - Latent-variable models that use a neural network to do approximate inference.
  - Generative adversarial networks - A way to train generative models by optimizing them to fool a classifier

Differentiable Generator Networks

- Generative Modelling is not new; we've known how to make arbitrarily complex probabilistic graphical models for many years.
  - ...But difficult to train and scale to real data, relying on MCMC.
- The past few years has seen major progress along four loose strands:
  - Invertible density estimation - A way to specify complex generative models by transforming a simple latent distribution with a series of invertible functions.
  - Autoregressive models - Another way to model $p(x)$ is to break the model into a series of conditional distributions: $p(x) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\dots$
  - Variational autoencoders - Latent-variable models that use a neural network to do approximate inference.
  - Generative adversarial networks - A way to train generative models by optimizing them to fool a classifier
- **Common thread in recent advances is that the loss functions are end-to-end differentiable.**

Differentiable Generator Networks: key idea

- We're interested in models that transform samples of latent variables $\boldsymbol{z}$ to
  - samples x, or,
  - distributions over samples x
- The model is a (differentiable) function $g(\boldsymbol{z}, \boldsymbol{\theta})$
  - typically g is a neural network.

Example: drawing samples from $\mathcal{N}(\mu, \Sigma)$

- Consider a simple generator network with a single affine layer that maps samples $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \longrightarrow \boxed{g_{\boldsymbol{\theta}}(\boldsymbol{z})} \longrightarrow \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Example: drawing samples from $\mathcal{N}(\mu, \Sigma)$

- Consider a simple generator network with a single affine layer that maps samples $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \longrightarrow \boxed{g_{\boldsymbol{\theta}}(\boldsymbol{z})} \longrightarrow \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

- Note: Exact solution is $\boldsymbol{x} = g_{\boldsymbol{\theta}}(\boldsymbol{z}) = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{z}$ where $\boldsymbol{L}$ is the Cholesky decomposition of $\boldsymbol{\Sigma}$: $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$, lower triangular $\boldsymbol{L}$.

Generating samples

More generally, we can think of g as providing a nonlinear change of variables that transforms a distribution over $\mathbf{z}$ into the desired distribution over $\mathbf{x}$:

$$p_z(\boldsymbol{z}) \longrightarrow \boxed{g(\boldsymbol{z})} \longrightarrow p_x(\boldsymbol{x})$$

Generating samples

More generally, we can think of g as providing a nonlinear change of variables that transforms a distribution over $\mathbf{z}$ into the desired distribution over $\mathbf{x}$:

$$p_z(\boldsymbol{z}) \longrightarrow \boxed{g(\boldsymbol{z})} \longrightarrow p_x(\boldsymbol{x})$$

For any *invertible, differentiable, continuous* g:

$$p_z(\boldsymbol{z}) = p_x(g(\boldsymbol{z})) \left|\det\left(\frac{\partial g}{\partial \boldsymbol{z}}\right)\right|$$

Which implicitly imposes a probability distribution over $\mathbf{x}$:

$$p_x(\boldsymbol{x}) = \frac{p_z(g^{-1}(\boldsymbol{x}))}{\left|\det\left(\frac{\partial g}{\partial \boldsymbol{z}}\right)\right|}$$

Generating samples

More generally, we can think of g as providing a nonlinear change of variables that transforms a distribution over $\mathbf{z}$ into the desired distribution over $\mathbf{x}$:

$$p_z(\boldsymbol{z}) \longrightarrow \boxed{g(\boldsymbol{z})} \longrightarrow p_x(\boldsymbol{x})$$

For any *invertible, differentiable, continuous* g:

$$p_z(\boldsymbol{z}) = p_x(g(\boldsymbol{z}))\left|\det\left(\frac{\partial g}{\partial \boldsymbol{z}}\right)\right|$$

Which implicitly imposes a probability distribution over $\mathbf{x}$:

$$p_x(\boldsymbol{x}) = \frac{p_z(g^{-1}(\boldsymbol{x}))}{\left|\det\left(\frac{\partial g}{\partial \boldsymbol{z}}\right)\right|}$$

Note: usually use an indirect means of learning g rather than minimise $-\log(p(\boldsymbol{x}))$ directly

Generating distributions

- Rather than use g to provide a sample of $\boldsymbol{x}$ directly, we could instead use g to define a conditional distribution over x, $p(\boldsymbol{x}|\boldsymbol{z})$
  - For example, g might produce the parameters of a particular distribution - e.g.:
    - means of Bernoulli
    - mean and variance of a Gaussian

Generating distributions

- Rather than use g to provide a sample of $\boldsymbol{x}$ directly, we could instead use g to define a conditional distribution over x, $p(\boldsymbol{x}|\boldsymbol{z})$
  - For example, g might produce the parameters of a particular distribution - e.g.:
    - means of Bernoulli
    - mean and variance of a Gaussian
- The distribution over $\boldsymbol{x}$ is imposed by marginalising $\boldsymbol{z}$:$p(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})$

Distributions vs Samples

- In both cases (g generates samples and g generates distributions) we can use the reparameterisation tricks we saw last lecture to train models.

Distributions vs Samples

- In both cases (g generates samples and g generates distributions) we can use the reparameterisation tricks we saw last lecture to train models.
- Generating distributions:
  - $+$ works for both continuous and discrete data
  - - need to specify the form of the output distribution

Distributions vs Samples

- In both cases (g generates samples and g generates distributions) we can use the reparameterisation tricks we saw last lecture to train models.
- Generating distributions:
  - $+$ works for both continuous and discrete data
  - - need to specify the form of the output distribution
- Generating samples:
  - $+$ works for continuous data
    - $+$ discrete data is recently possible - we need the STargmax
  - $+$ don't need to specify the distribution in explicit form

Complexity of Generative Modelling

- In classification both input and output are given
  - Optimisation only needs to learn the mapping

Complexity of Generative Modelling

- In classification both input and output are given
  - Optimisation only needs to learn the mapping
- Generative modelling is more complex than classification because

Complexity of Generative Modelling

- In classification both input and output are given
    - Optimisation only needs to learn the mapping
- Generative modelling is more complex than classification because
    - learning requires optimizing intractable criteria

Complexity of Generative Modelling

- In classification both input and output are given
  - Optimisation only needs to learn the mapping
- Generative modelling is more complex than classification because
  - learning requires optimizing intractable criteria
  - data does not specify both input $\boldsymbol{z}$ and output $\boldsymbol{x}$ of the generator network

Complexity of Generative Modelling

- In classification both input and output are given
  - Optimisation only needs to learn the mapping
- Generative modelling is more complex than classification because
  - learning requires optimizing intractable criteria
  - data does not specify both input $\boldsymbol{z}$ and output $\boldsymbol{x}$ of the generator network
  - learning procedure needs to determine how to arrange $\boldsymbol{z}$ space in a useful way and how to map $\boldsymbol{z}$ to $\boldsymbol{x}$

Variational Autoencoders

Variational Autoencoders (VAEs)

The Variational Autoencoder uses the following generative process to draw samples:

$$\boldsymbol{z} \sim p_{\text{model}}(\boldsymbol{z}) \rightarrow \boxed{p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta}) = p_{\text{model}}(\boldsymbol{x}; g_{\boldsymbol{\theta}}(\boldsymbol{z}))} \rightarrow \boldsymbol{x} \sim p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})$$

- The learning problem is to find $\boldsymbol{\theta}$ that maximises the probability of each $\boldsymbol{x}$ in the training set under $p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})p(\boldsymbol{z})d\boldsymbol{z}$
- $p_{\text{model}}(\boldsymbol{z})$ is most often chosen to be $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
- $p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z})$ is chosen according to the data; typically Gaussian for real-valued data (most often just predicting the means, with a fixed diagonal covariance) or Bernoulli for binary data.
  - Intuition: we don't exactly want to exactly create the training examples; we want to create things *like* the training examples

Variational Autoencoders (VAEs)

- Conceptually we can compute $p(\boldsymbol{x}) \approx \frac{1}{n}\sum_i^n p(\boldsymbol{x}|\boldsymbol{z}_i;\boldsymbol{\theta})$ for n samples of $\boldsymbol{z}$, $\{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_n\}$ and just use gradient ascent to do the optimisation

Variational Autoencoders (VAEs)

- Conceptually we can compute $p(\boldsymbol{x}) \approx \frac{1}{n}\sum_i^n p(\boldsymbol{x}|\boldsymbol{z}_i;\boldsymbol{\theta})$ for n samples of $\boldsymbol{z}$, $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ and just use gradient ascent to do the optimisation
  - This isn't tractable in practice; n would need to be *extremely* big!

Variational Autoencoders (VAEs)

- Conceptually we can compute $p(\boldsymbol{x}) \approx \frac{1}{n}\sum_i^n p(\boldsymbol{x}|\boldsymbol{z}_i;\boldsymbol{\theta})$ for n samples of $\boldsymbol{z}$, $\{\boldsymbol{z}_1,\dots,\boldsymbol{z}_n\}$ and just use gradient ascent to do the optimisation
  - This isn't tractable in practice; n would need to be *extremely* big!
- For most $\boldsymbol{z}$, $p(\boldsymbol{x}|\boldsymbol{z})$ will be nearly zero, and hence contribute almost nothing to our estimate of $p(\boldsymbol{x})$

Variational Autoencoders (VAEs)

- Conceptually we can compute $p(\boldsymbol{x}) \approx \frac{1}{n}\sum_i^n p(\boldsymbol{x}|\boldsymbol{z}_i;\boldsymbol{\theta})$ for n samples of $\boldsymbol{z}$, $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ and just use gradient ascent to do the optimisation
  - This isn't tractable in practice; n would need to be *extremely* big!
- For most $\boldsymbol{z}$, $p(\boldsymbol{x}|\boldsymbol{z})$ will be nearly zero, and hence contribute almost nothing to our estimate of $p(\boldsymbol{x})$
- The key idea behind the VAE is to learn to sample values of $\boldsymbol{z}$ that are likely to have produced $\boldsymbol{x}$, and compute $p(\boldsymbol{x})$ just from those

Variational Autoencoders (VAEs)

- Conceptually we can compute $p(\boldsymbol{x}) \approx \frac{1}{n}\sum_i^n p(\boldsymbol{x}|\boldsymbol{z}_i;\boldsymbol{\theta})$ for n samples of $\boldsymbol{z}$, $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\}$ and just use gradient ascent to do the optimisation
  - This isn't tractable in practice; n would need to be *extremely* big!
- For most $\boldsymbol{z}$, $p(\boldsymbol{x}|\boldsymbol{z})$ will be nearly zero, and hence contribute almost nothing to our estimate of $p(\boldsymbol{x})$
- The key idea behind the VAE is to learn to sample values of $\boldsymbol{z}$ that are likely to have produced $\boldsymbol{x}$, and compute $p(\boldsymbol{x})$ just from those
  - Introduce a new function $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ which can take a value of $\boldsymbol{x}$ and produce the distribution over $\boldsymbol{z}$ values that are likely to produce $\boldsymbol{x}$.

Variational Autoencoders (VAEs)

- Conceptually we can compute $p(\boldsymbol{x}) \approx \frac{1}{n}\sum_i^n p(\boldsymbol{x}|\boldsymbol{z}_i;\boldsymbol{\theta})$ for n samples of $\boldsymbol{z}$, $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\}$ and just use gradient ascent to do the optimisation
  - This isn't tractable in practice; n would need to be *extremely* big!
- For most $\boldsymbol{z}$, $p(\boldsymbol{x}|\boldsymbol{z})$ will be nearly zero, and hence contribute almost nothing to our estimate of $p(\boldsymbol{x})$
- The key idea behind the VAE is to learn to sample values of $\boldsymbol{z}$ that are likely to have produced $\boldsymbol{x}$, and compute $p(\boldsymbol{x})$ just from those
  - Introduce a new function $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ which can take a value of $\boldsymbol{x}$ and produce the distribution over $\boldsymbol{z}$ values that are likely to produce $\boldsymbol{x}$.
  - The space of $\boldsymbol{z}$ values that are likely under q should be much smaller than the space of than under prior $p(\boldsymbol{z})$.

Variational Autoencoders (VAEs)

- Conceptually we can compute $p(\boldsymbol{x}) \approx \frac{1}{n}\sum_i^n p(\boldsymbol{x}|\boldsymbol{z}_i;\boldsymbol{\theta})$ for n samples of $\boldsymbol{z}$, $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\}$ and just use gradient ascent to do the optimisation
  - This isn't tractable in practice; n would need to be *extremely* big!
- For most $\boldsymbol{z}$, $p(\boldsymbol{x}|\boldsymbol{z})$ will be nearly zero, and hence contribute almost nothing to our estimate of $p(\boldsymbol{x})$
- The key idea behind the VAE is to learn to sample values of $\boldsymbol{z}$ that are likely to have produced $\boldsymbol{x}$, and compute $p(\boldsymbol{x})$ just from those
  - Introduce a new function $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ which can take a value of $\boldsymbol{x}$ and produce the distribution over $\boldsymbol{z}$ values that are likely to produce $\boldsymbol{x}$.
  - The space of $\boldsymbol{z}$ values that are likely under q should be much smaller than the space of than under prior $p(\boldsymbol{z})$.
  - We can now compute $\mathbb{E}_{\boldsymbol{z}\sim q_\phi} p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})$ easily
    - if the PDF $q(\boldsymbol{z})$, is not $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then how does that help us optimize $p(\boldsymbol{x})$?
    - and how does this expectation relate to $p(\boldsymbol{x})$?

Variational Inference

Log-probability $\log p(x) = \log \int p(x|z)p(z)dz$

Jensen's inequality: $\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$
Log product rule: $\log(a \cdot b) = \log a + \log b$
Log quotient rule: $\log(a/b) = \log a - \log b$

Variational Inference

Log-probability $$\log p(x) = \log \int p(x|z)p(z)dz$$

Proposal $$\log p(x) = \log \int p(x|z)p(z)\frac{q(z|x)}{q(z|x)}dz$$

Jensen's inequality: $\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$
Log product rule: $\log(a \cdot b) = \log a + \log b$
Log quotient rule: $\log(a/b) = \log a - \log b$

Variational Inference

Log-probability $\log p(x) = \log \int p(x|z)p(z)dz$

Proposal $\log p(x) = \log \int p(x|z)p(z)\frac{q(z|x)}{q(z|x)}dz$

Importance weight $\log p(x) = \log \int p(x|z)\frac{p(z)}{q(z|x)}q(z|x)dz$

Jensen's inequality: $\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$
Log product rule: $\log(a \cdot b) = \log a + \log b$
Log quotient rule: $\log(a/b) = \log a - \log b$

Variational Inference

Log-probability	$\log p(x) = \log \int p(x\|z)p(z)dz$
Proposal	$\log p(x) = \log \int p(x\|z)p(z)\frac{q(z\|x)}{q(z\|x)}dz$
Importance weight	$\log p(x) = \log \int p(x\|z)\frac{p(z)}{q(z\|x)}q(z\|x)dz$
Jensen's inequality	$\log p(x) \geq \int q(z\|x) \log \left(p(x\|z)\frac{p(z)}{q(z\|x)} \right) dz$

Jensen's inequality: $\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$
Log product rule: $\log(a \cdot b) = \log a + \log b$
Log quotient rule: $\log(a/b) = \log a - \log b$

Variational Inference

Log-probability	$\log p(x) = \log \int p(x\|z)p(z)dz$
Proposal	$\log p(x) = \log \int p(x\|z)p(z)\frac{q(z\|x)}{q(z\|x)}dz$
Importance weight	$\log p(x) = \log \int p(x\|z)\frac{p(z)}{q(z\|x)}q(z\|x)dz$
Jensen's inequality	$\log p(x) \geq \int q(z\|x) \log \left(p(x\|z)\frac{p(z)}{q(z\|x)}\right) dz$
Rearrange	$\log p(x) \geq \int q(z\|x) \log p(x\|z)dz - \int q(z\|x) \log \frac{q(z\|x)}{p(z)}dz$

Jensen's inequality: $\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$
Log product rule: $\log(a \cdot b) = \log a + \log b$
Log quotient rule: $\log(a/b) = \log a - \log b$

Variational Inference

Log-probability	$\log p(x) = \log \int p(x\|z)p(z)dz$
Proposal	$\log p(x) = \log \int p(x\|z)p(z)\frac{q(z\|x)}{q(z\|x)}dz$
Importance weight	$\log p(x) = \log \int p(x\|z)\frac{p(z)}{q(z\|x)}q(z\|x)dz$
Jensen's inequality	$\log p(x) \geq \int q(z\|x) \log \left(p(x\|z)\frac{p(z)}{q(z\|x)} \right) dz$
Rearrange	$\log p(x) \geq \int q(z\|x) \log p(x\|z)dz - \int q(z\|x) \log \frac{q(z\|x)}{p(z)}dz$
ELBO	$\log p(x) \geq \mathbb{E}_{z \sim q(z\|x)} \log p(x\|z) - D_{\mathrm{KL}}(q(z\|x)\|\|p(z))$

Jensen's inequality: $\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$
Log product rule: $\log(a \cdot b) = \log a + \log b$
Log quotient rule: $\log(a/b) = \log a - \log b$

The Evidence LOwer Bound (ELBO) / variational lower bound

The ELBO expression we just derived is a cornerstone of variational inference:

$$\begin{aligned}\mathcal{L}(q) &= \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z}) - D_{\text{KL}}(q(\boldsymbol{z}|\boldsymbol{x})||p_{\text{model}}(\boldsymbol{z})) \\ &\leq \log p_{\text{model}}(\boldsymbol{x})\end{aligned}$$

- The expectation term looks just like a reconstruction log-likelihood found in normal autoencoders
  - If $p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z})$ is Gaussian, then this is MSE between the true training $\boldsymbol{x}$ and a generated sample computed from $\boldsymbol{z}$, averaged across many $\boldsymbol{z}$'s (each a function of $\boldsymbol{x}$)
- The KL term is forcing the approximate posterior $q(\boldsymbol{z}|\boldsymbol{x})$ towards the prior $p_{\text{model}}(\boldsymbol{z})$.

Why is it called an autoencoder?

- $q(\boldsymbol{z}|\boldsymbol{x})$ is referred to as an encoder; it's used to take $\boldsymbol{x}$ and turn it into a $\boldsymbol{z}$

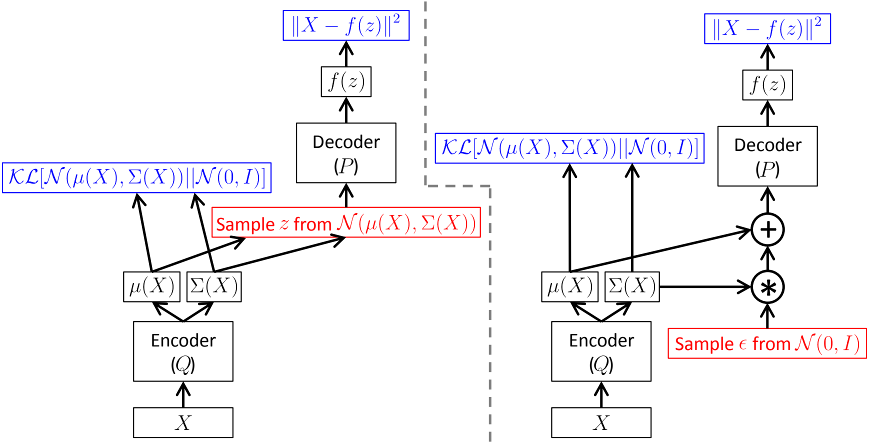
Why is it called an autoencoder?

- $q(\boldsymbol{z}|\boldsymbol{x})$ is referred to as an encoder; it's used to take $\boldsymbol{x}$ and turn it into a $\boldsymbol{z}$
- $p_{\text{model}}(\boldsymbol{x}; g_{\boldsymbol{\theta}}(\boldsymbol{z}))$ is referred to as a decoder network; it takes a $\boldsymbol{z}$ and decodes it into a target $\boldsymbol{x}$

Why is it called an autoencoder?

- $q(\boldsymbol{z}|\boldsymbol{x})$ is referred to as an encoder; it's used to take $\boldsymbol{x}$ and turn it into a $\boldsymbol{z}$
- $p_{\text{model}}(\boldsymbol{x}; g_{\boldsymbol{\theta}}(\boldsymbol{z}))$ is referred to as a decoder network; it takes a $\boldsymbol{z}$ and decodes it into a target $\boldsymbol{x}$
- From a practical standpoint, a VAE is a normal autoencoder with two key differences:
  - the encoder generates a distribution that must be sampled
    - the network produces the sufficient statistics of the distribution (e.g. means and diagonal co-variances for a typical VAE with Gaussian $q(\boldsymbol{z}|\boldsymbol{x})$)
  - the decoder generates a distribution, which, during training the NLL of the true data $\boldsymbol{x}$ is compared against

VAE: Diagram



From Carl Doersch's Tutorial on VAEs - https://arxiv.org/pdf/1606.05908.pdf

VAE Models and Performance

- VAEs can be used with any kind of data
  - the distributions and network architecture just needs to be set accordingly
  - e.g. it's common to use convolutions in the encoder and transpose convolutions in (Gaussian) decoder for image data

VAE Models and Performance

- VAEs can be used with any kind of data
  - the distributions and network architecture just needs to be set accordingly
  - e.g. it's common to use convolutions in the encoder and transpose convolutions in (Gaussian) decoder for image data
- VAEs have nice learning dynamics; they tend to be easy to optimise with stable convergence

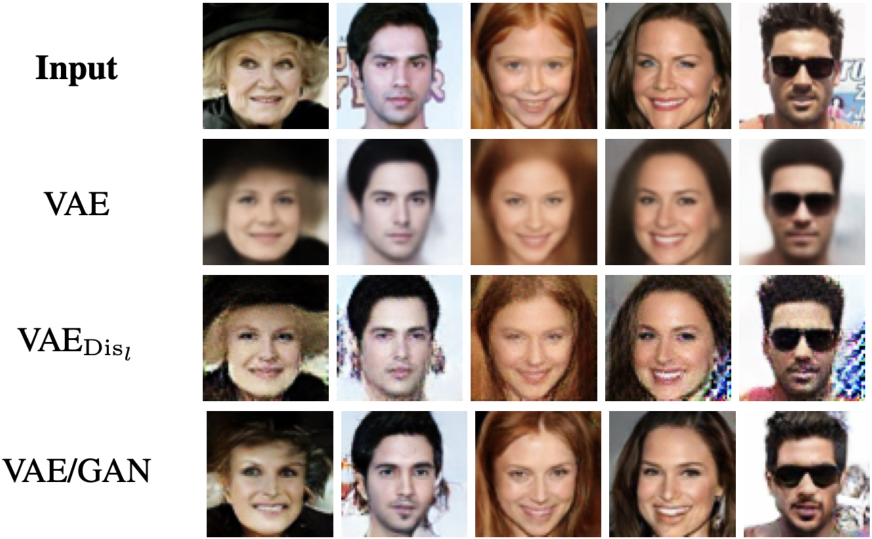
VAE Models and Performance

- VAEs can be used with any kind of data
  - the distributions and network architecture just needs to be set accordingly
  - e.g. it's common to use convolutions in the encoder and transpose convolutions in (Gaussian) decoder for image data
- VAEs have nice learning dynamics; they tend to be easy to optimise with stable convergence
- VAEs have a reputation for producing blurry reconstructions of images
  - Not fully understood why, but most likely related to a side effect of maximum-likelihood training

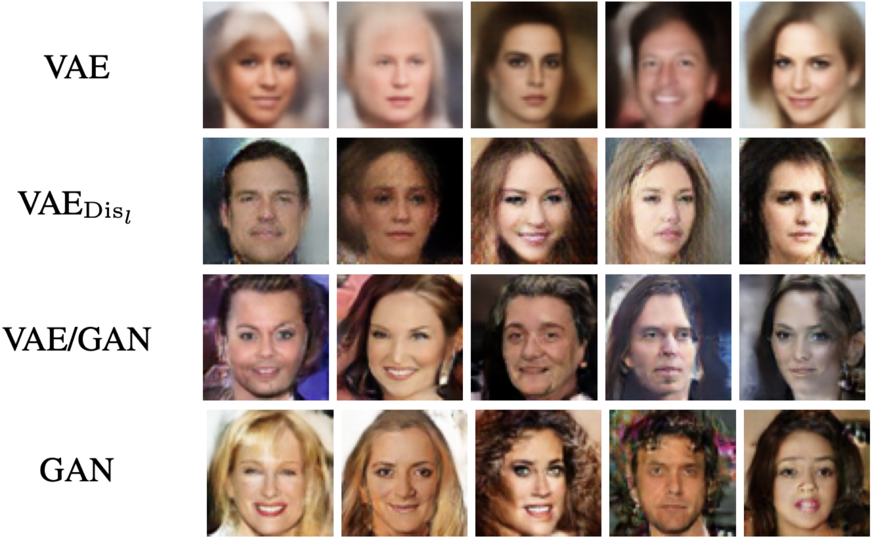
VAE Models and Performance

- VAEs can be used with any kind of data
  - the distributions and network architecture just needs to be set accordingly
  - e.g. it's common to use convolutions in the encoder and transpose convolutions in (Gaussian) decoder for image data
- VAEs have nice learning dynamics; they tend to be easy to optimise with stable convergence
- VAEs have a reputation for producing blurry reconstructions of images
  - Not fully understood why, but most likely related to a side effect of maximum-likelihood training
- VAEs tend to only utilise a small subset of the dimensions of $\boldsymbol{z}$
  - Pro: automatic latent variable selection
  - Con: better reconstructions should be possible given the available code-space

Reconstructions Example



Sampling Example



Generative Adversarial Networks

Generative Adversarial Networks (GANs)

- New (old?![1]) method of training deep generative models

[1] c.f. Schmidhuber

Generative Adversarial Networks (GANs)

- New (old?![1]) method of training deep generative models
- Idea: pitch a generator and a discriminator against each other
  - Generator tries to draw samples from $p(\mathrm{x})$
  - Discriminator tries to tell if sample came from the generator (fake) or the real world

[1] c.f. Schmidhuber

Generative Adversarial Networks (GANs)

- New (old?![1]) method of training deep generative models
- Idea: pitch a generator and a discriminator against each other
  - Generator tries to draw samples from $p(\mathrm{x})$
  - Discriminator tries to tell if sample came from the generator (fake) or the real world
- Both discriminator and generator are deep networks (differentiable functions)

[1] c.f. Schmidhuber

Generative Adversarial Networks (GANs)

- New (old?![1]) method of training deep generative models
- Idea: pitch a generator and a discriminator against each other
  - Generator tries to draw samples from $p(\mathrm{x})$
  - Discriminator tries to tell if sample came from the generator (fake) or the real world
- Both discriminator and generator are deep networks (differentiable functions)
- LeCun quote 'GANs, the most interesting idea in the last ten years in machine learning'

[1] c.f. Schmidhuber

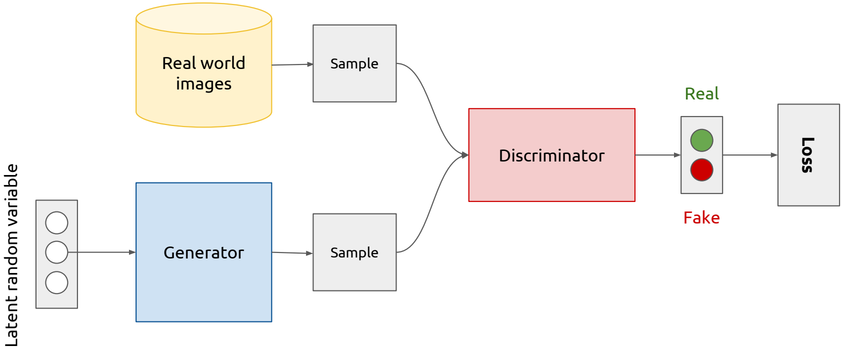
Aside: Adversarial Learning vs. Adversarial Examples

The approach of GANs is called adversarial since the two networks have *antagonistic* objectives.

This is not to be confused with *adversarial examples* in machine learning.

See these two papers for more details:
`https://arxiv.org/pdf/1412.6572.pdf`
`https://arxiv.org/pdf/1312.6199.pdf`

Generative adversarial networks (conceptual)



Picture Credit: Xavier Giro-i-Nieto

More Formally

- The **generator**

$$\boldsymbol{x} = g(\boldsymbol{z})$$

is trained so that it gets a random input $\boldsymbol{z} \in \mathbb{R}^n$ from a distribution (typically $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ or $\mathcal{U}(\mathbf{0}, \boldsymbol{I})$) and produces a sample $\boldsymbol{x} \in \mathbb{R}^d$ following the data distribution as output (ideally). Usually $n << d$.

- The **discriminator**

$$y = d(\boldsymbol{x})$$

gets a sample $\boldsymbol{x}$ as input and predicts a probability $y \in [0, 1]$ (or real-valued logit of a Bernoulli distribution) determining if it is real or fake.

More Practically

- Training a standard GAN is difficult and often results in two undesirable behaviours
  - Oscillations without convergence. No guarantee that the loss will actually decrease...
    - It has been shown that a GAN has saddle-point solution, rather than a local minima.
  - The **mode collapse** problem, when the generator models very well a small sub-population, concentrating on a few modes.
- Additionally, performance is hard to assess and often boils down to heuristic observations.

Deep Convolutional Generative Adversarial Networks (DCGANs)

- Motivates the use of GANS to learn reusable feature representations from large unlabelled datasets.
- GANs known to be unstable to train, often resulting in generators that produce "nonsensical outputs".
- Model exploration to identify architectures that result in **stable** training across datasets with higher resolution and deeper models.

Architecture Guidelines for Stable DCGAN

- Replace pooling layers with strided convolutions in the discriminator and fractional-strided (transpose) convolutions in the generator.
  - This will allow the network to learn its own spatial downsampling.

Architecture Guidelines for Stable DCGAN

- Replace pooling layers with strided convolutions in the discriminator and fractional-strided (transpose) convolutions in the generator.
    - This will allow the network to learn its own spatial downsampling.
- Use batchnorm in both the generator and the discriminator.
    - This helps deal with training problems due to poor initialisation and helps the gradient flow.

Architecture Guidelines for Stable DCGAN

- Replace pooling layers with strided convolutions in the discriminator and fractional-strided (transpose) convolutions in the generator.
  - This will allow the network to learn its own spatial downsampling.
- Use batchnorm in both the generator and the discriminator.
  - This helps deal with training problems due to poor initialisation and helps the gradient flow.
- Eliminate fully connected hidden layers for deeper architectures.

Architecture Guidelines for Stable DCGAN

- Replace pooling layers with strided convolutions in the discriminator and fractional-strided (transpose) convolutions in the generator.
  - This will allow the network to learn its own spatial downsampling.
- Use batchnorm in both the generator and the discriminator.
  - This helps deal with training problems due to poor initialisation and helps the gradient flow.
- Eliminate fully connected hidden layers for deeper architectures.
- Use ReLU activation in the generator for all layers except for the output, which uses tanh.

Architecture Guidelines for Stable DCGAN

- Replace pooling layers with strided convolutions in the discriminator and fractional-strided (transpose) convolutions in the generator.
  - This will allow the network to learn its own spatial downsampling.
- Use batchnorm in both the generator and the discriminator.
  - This helps deal with training problems due to poor initialisation and helps the gradient flow.
- Eliminate fully connected hidden layers for deeper architectures.
- Use ReLU activation in the generator for all layers except for the output, which uses tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Summary

- Generative modelling is a massive field with a long history
- Differentiable generators have had a profound impact in making models that work with real data at scale
- VAEs and GANs are currently the most popular approaches to training generators for spatial data
- We've only scratched the surface of generative modelling
  - Auto-regressive approaches are popular for sequences (e.g. language modelling).
    - But also for images (e.g. PixelRNN, PixelCNN)
  - typically RNN-based
  - but not necessarily - e.g. WaveNet is a convolutional auto-regressive generative model