

Yes, we GAN.

VLC = $\int \int \int$ Vision
Learning and Control

Deep Generative Modelling

Jonathon Hare

Vision, Learning and Control
University of Southampton

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

Recap: Generative Models

- Learn models of the data: $p(x)$
- Learn *conditional* models of the data: $p(x|y = y)$
- Some generative models allow the probability distributions to be evaluated explicitly
 - i.e. compute the probability of a piece of data x : $p(x = x)$
- Some generative models allow the probability distributions to be sampled
 - i.e. draw a sample x based on the distribution: $x \sim p(x)$
- 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
- 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.
- 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 \mathbf{z} to
 - samples \mathbf{x} , or,
 - distributions over samples \mathbf{x}
- The model is a (differentiable) function $g(\mathbf{z}, \theta)$
 - typically g is a neural network.

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

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

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

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

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

For any *invertible, differentiable, continuous* g :

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

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

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

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

Generating distributions

- Rather than use g to provide a sample of \mathbf{x} directly, we could instead use g to define a conditional distribution over x , $p(\mathbf{x}|\mathbf{z})$
 - 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 \mathbf{x} is imposed by marginalising \mathbf{z} : $p(\mathbf{x}) = \mathbb{E}_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})$

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.
- 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
- Generative modelling is more complex than classification because
 - learning requires optimizing intractable criteria
 - data does not specify both input z and output x of the generator network
 - learning procedure needs to determine how to arrange z space in a useful way and how to map z to x

Variational Autoencoders (VAEs)

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

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

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

Variational Inference

$$\text{Log-probability} \quad \log p(x) = \log \int p(x|z)p(z)dz$$

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

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

$$\text{Jensen's inequality} \quad \log p(x) \geq \int q(z|x) \log \left(p(x|z) \frac{p(z)}{q(z|x)} \right) dz$$

$$\text{Rearrange} \quad \log p(x) \geq \int q(z|x) \log p(x|z) dz - \int q(z|x) \log \frac{q(z|x)}{p(z)} dz$$

$$\text{ELBO} \quad \log p(x) \geq \mathbb{E}_{z \sim q(z|x)} \log p(x|z) - D_{\text{KL}}(q(z|x)||p(z))$$

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:

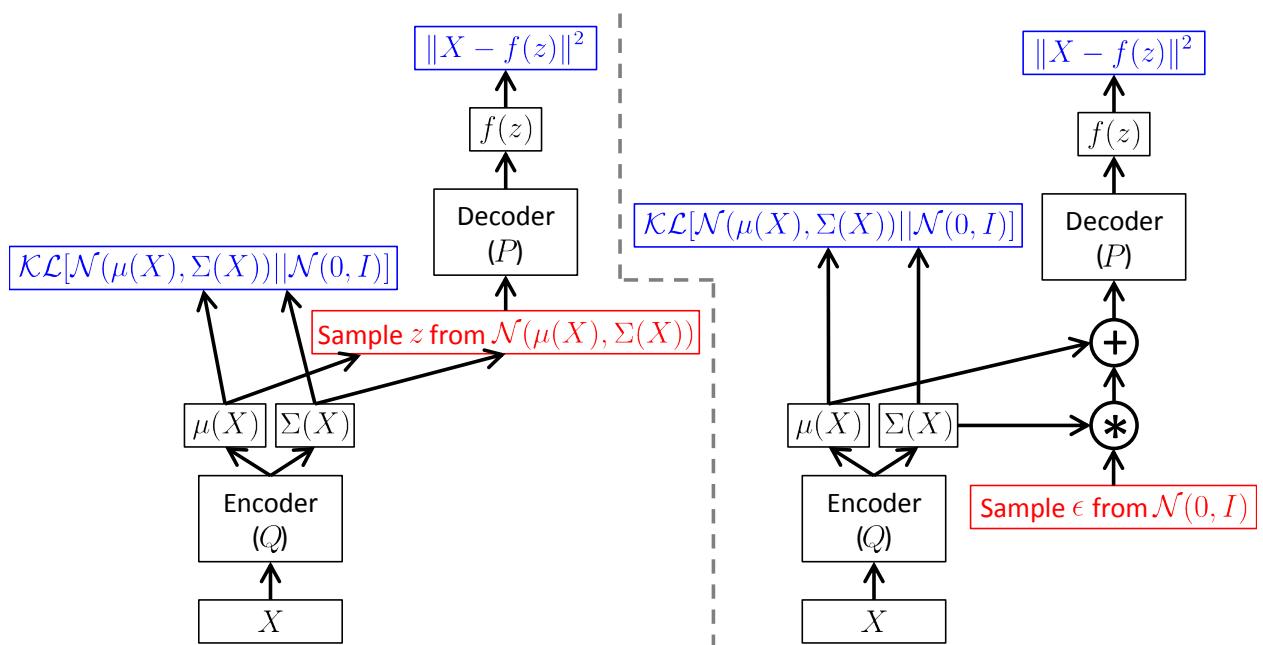
$$\begin{aligned} \mathcal{L}(q) &= \mathbb{E}_{z \sim q(z|x)} \log p_{\text{model}}(\mathbf{x}|z) - D_{\text{KL}}(q(z|x)||p_{\text{model}}(z)) \\ &\leq \log p_{\text{model}}(\mathbf{x}) \end{aligned}$$

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

Why is it called an autoencoder?

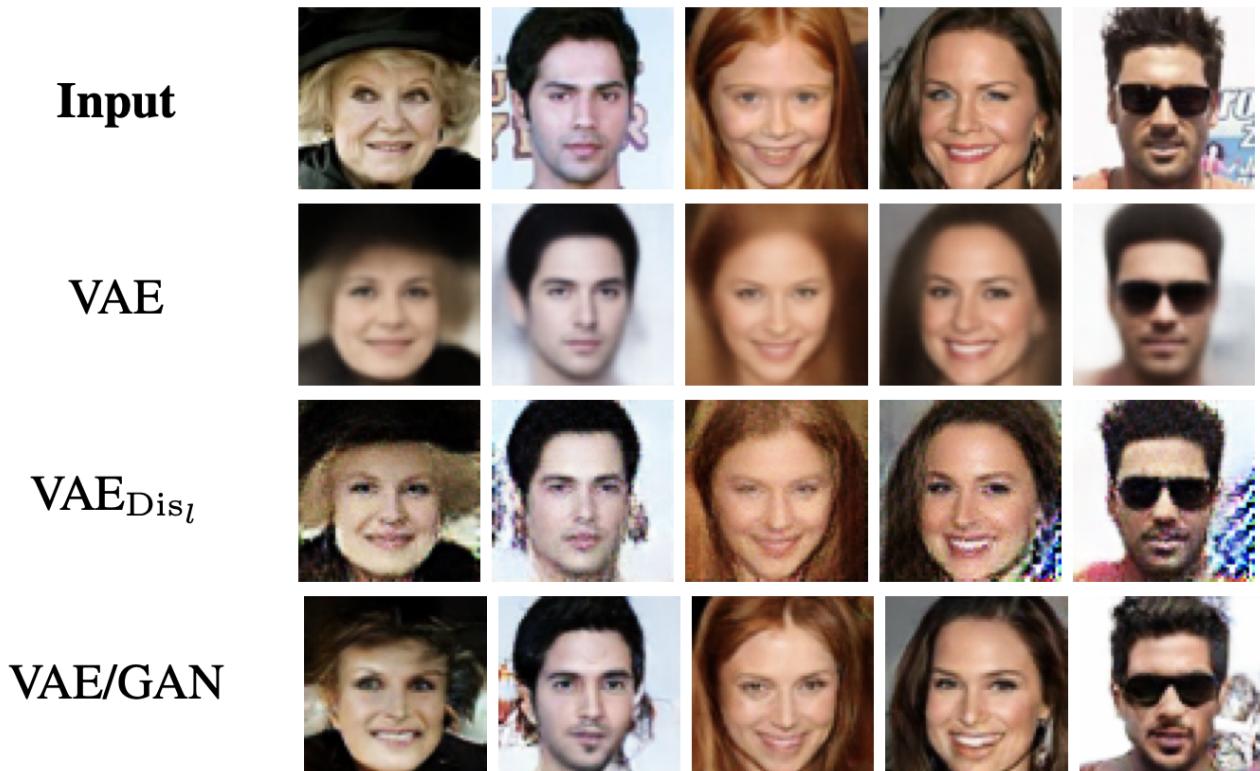
- $q(z|x)$ is referred to as an encoder; it's used to take x and turn it into a z
- $p_{\text{model}}(x; g_{\theta}(z))$ is referred to as a decoder network; it takes a z and decodes it into a target x
- From a practical standpoint, a VAE is a normal autoencoder with two key differences:
 - the encoder 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 $q(z|x)$)
 - the decoder generates a distribution, which, during training the NLL of the true data x is compared against

VAE: Diagram

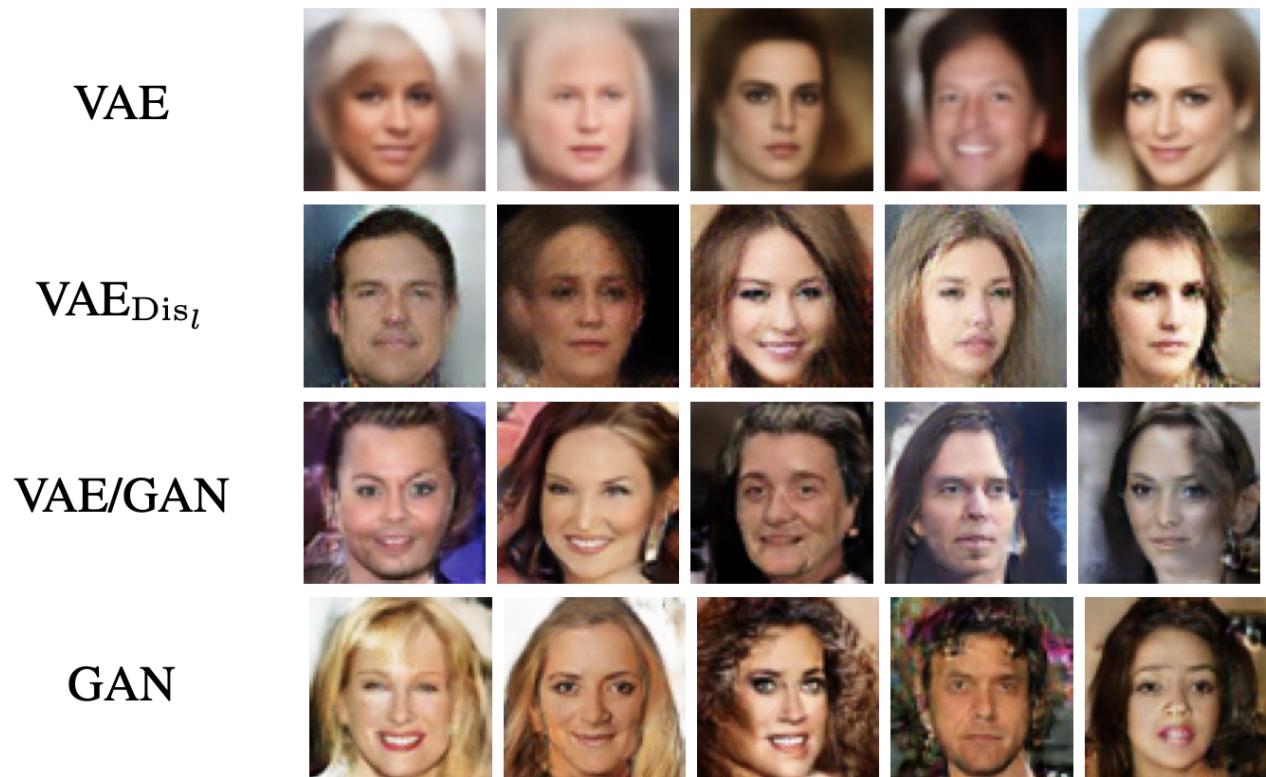


- VAEs can be used with any kind of data
 - the distributions and network architecture just needs to be set accordingly
 - e.g. it's common to use convolutions in the encoder and transpose convolutions in (Gaussian) decoder for image data
- VAEs have nice learning dynamics; they tend to be easy to optimise 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 z
 - Pro: automatic latent variable selection
 - Con: better reconstructions should be possible given the available code-space

Reconstructions Example



Sampling Example



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 $p(x)$
- 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’

¹c.f. Schmidhuber

Aside: Adversarial Learning vs. Adversarial Examples

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

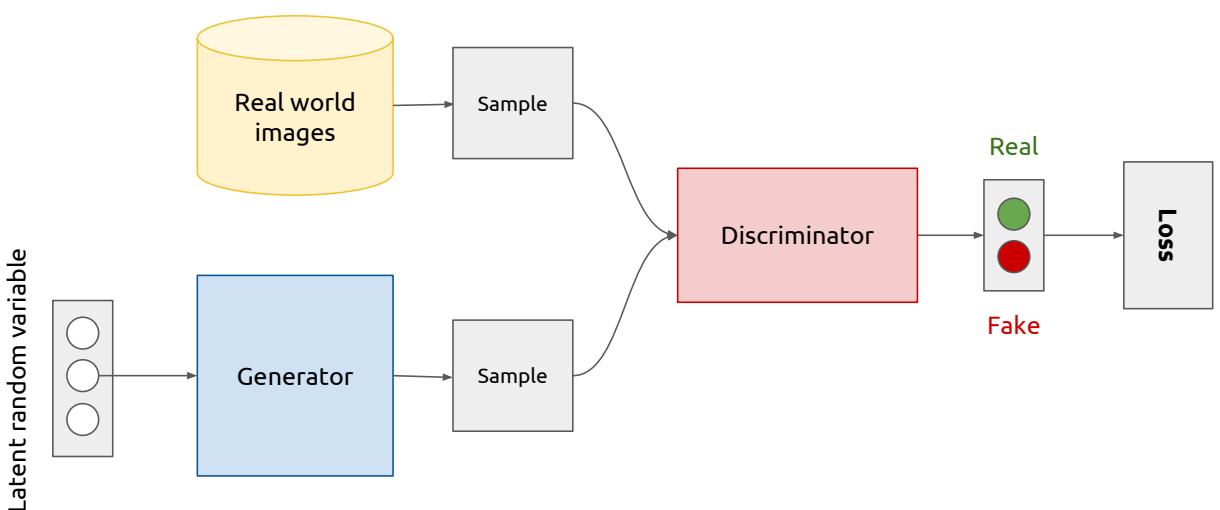
This is not to be confused with *adversarial examples* in machine learning.

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)



More Formally

- The **generator**

$$\mathbf{x} = g(\mathbf{z}) \quad (1)$$

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

- The **discriminator**

$$y = d(\mathbf{x}) \quad (2)$$

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

More Practically

- Training a standard GAN is difficult and often results in two undesirable behaviours
 - ➊ Oscillations without convergence. Contrary to standard loss minimisation, we have no guarantee that the loss will actually decrease.
 - It has been shown that a GAN has saddle-point solution, rather than a local minima.
 - ➋ 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.
- 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