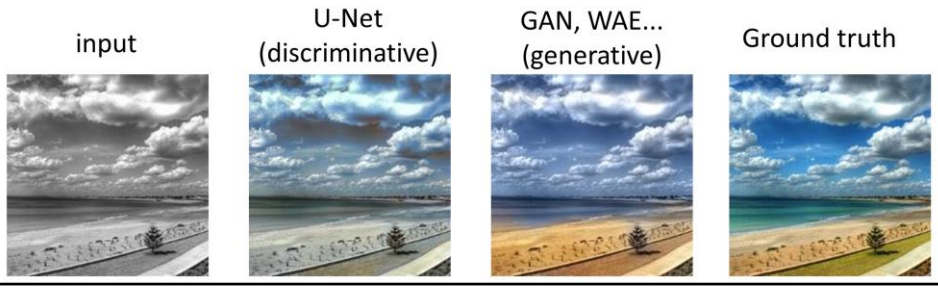
Neural Network Tools and Principles, part 2

Generative models

Computational Intelligence in Games, Spring 2018
Prof. Perttu Hämäläinen
Aalto University

Generative models

- Learn a probability distribution $p(\mathbf{x})$ – e.g., facial images – and a way to draw samples from it
- More common in practice: learn a conditional distribution $p(\mathbf{y} \mid \mathbf{x})$
- Same as approximating some function $\mathbf{y}=f(\mathbf{x})$, but with multiple possible $\mathbf{y}$ for each $\mathbf{x}$
- Example: image colorization. Many possible interpretations.



Simply training a convolutional neural network with input and output pairs makes the system output the average color, which is often unrealistic. Generative model needed to sample valid possible colors.

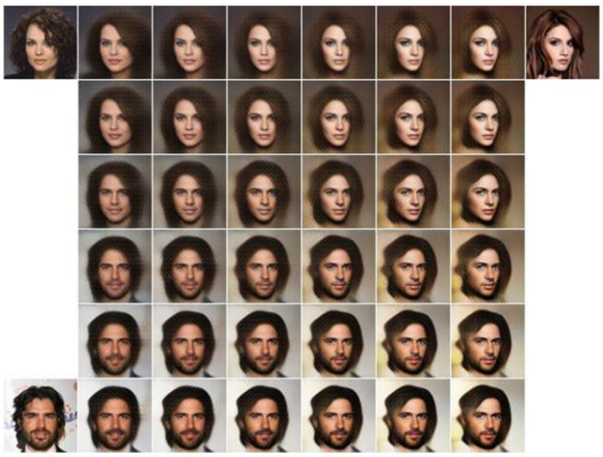
Basic principle

- Sample some random numbers and feed them through a network that converts them into images, sound, text...
- Three types of models: Autoencoders (VAE, WAE), Generative Adversarial Networks (GAN), Flow-based models

Latent space, representation learning

- If one samples image pixels or audio values, one gets noise.
- The models try to learn an *N*-dimensional ”latent space” where all positions (vectors of *N* numbers) in some region map to some meaningful image or sound, when passed through a generator or decoder network.
- If *N* is small, this enforces the networks to learn about relations. E.g., similar images should have similar latent encodings
- Directions often have semantic meaning: Along some axis, male faces might be to the left, and female to the right
- This allows interesting manipulations...

A 2D latent space of an autoencoder trained with faces



https://medium.com/@juliendespois/latent-space-visualization-deep-learning-bits-2-bd09a46920df

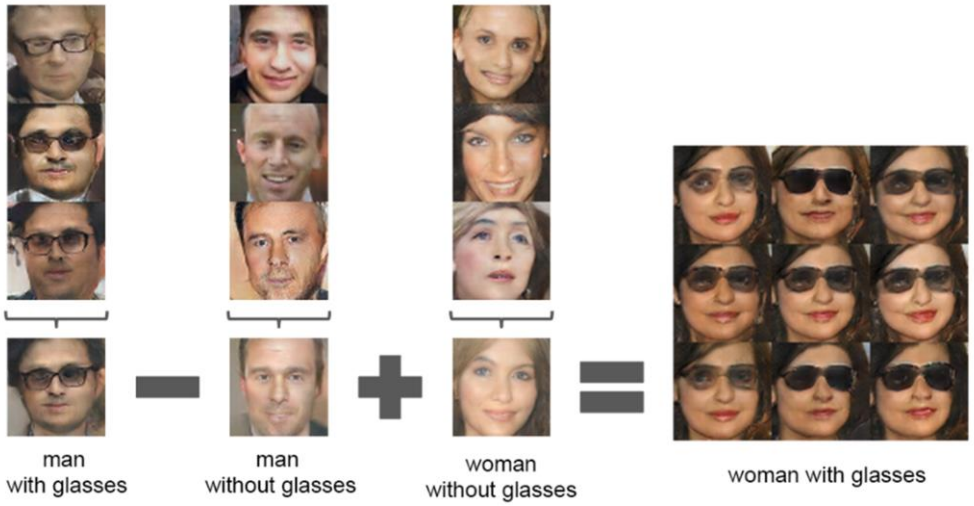
An autoencoder is an encoder-decoder network trained with the same signals as the inputs and target outputs.

Here, a convolutional autoencoder has learned a 2D encoding where the horizontal axis denotes face rotation and vertical denotes personality.

The interpolations are produced simply by linearly interpolating the 2D encodings (position vectors) of the corner images.

If one simply interpolates in the image pixel space, one gets a crossfade effect instead.

Latent space math



Strikingly, one can often perform ”semantic math” with latent space vectors.

Here’s another example. In this case one just adds and subtracts the latent space (encoded) vectors corresponding to images, and gets interesting semantically meaningful results. This seems to emerge as a side result of the networks being forced to represent complex data with just a few numbers.

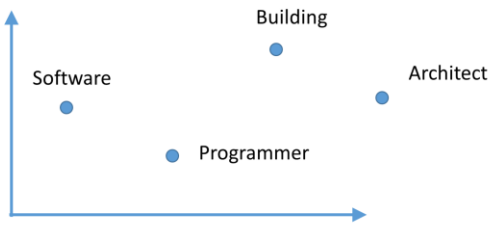
https://arxiv.org/pdf/1511.06434.pdf

Pixel-space math

Again, if one simply adds and subtracts pixels, the results are less interesting.

Same with words

- King - Man + Woman = Queen
- Note that this is equivalent to King - Man = Queen - Woman
- Software - Building + Architect = Programmer
- Precomputed dictionaries of such encodings are available, e.g., Fasttext (https://fasttext.cc/)

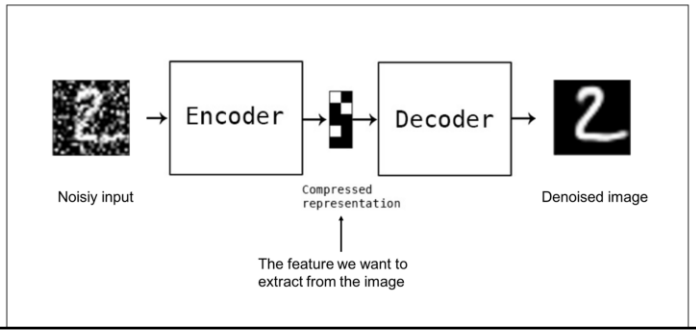


See also:

http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/

Variational autoencoder (VAE, 2014)

- Random samples in the latent space (the encoder outputs) of an autoencoder sometimes generate valid decoded output, sometimes not
- A basic autoencoder does not guarantee what happens when the encoding lies between training examples.
- VAE is an autoencoder where the compressed representation is forced to be normally distributed => easy to sample



Generating new images from this kind of model is simply done by ignoring the encoder, sampling a compressed representation, and passing it through the decoder.

Wasserstein Autoencoder (WAE, ICLR 2018)

- VAE is old and usually doesn't give great results (details beyond the scope of this lecture)
- WAE is the modern version, but a bit more costly to train

Wasserstein Auto-Encoders

Ilya Tolstikhin[1], Olivier Bousquet[2], Sylvain Gelly[2], and Bernhard Schölkopf[1]

[1]Max Planck Institute for Intelligent Systems
[2]Google Brain

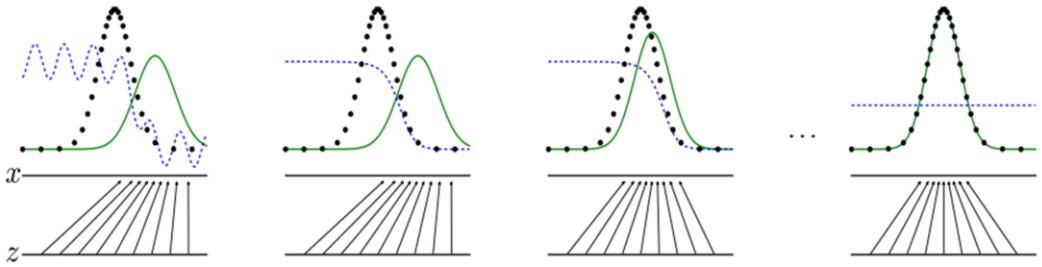
Abstract

We propose the Wasserstein Auto-Encoder (WAE)—a new algorithm for building a generative model of the data distribution. WAE minimizes a penalized form of the Wasserstein distance between the model distribution and the target distribution, which leads to a different regularizer than the one used by the Variational Auto-Encoder (VAE) [1]. This regularizer encourages the encoded training distribution to match the prior. We compare our algorithm with several other techniques and show that it is a generalization of adversarial auto-encoders (AAE) [2]. Our experiments show that WAE shares many of the properties of VAEs (stable training, encoder-decoder architecture, nice latent manifold structure) while generating samples of better quality, as measured by the FID score.

https://arxiv.org/abs/1711.01558

Generative Adversarial Networks (GANs)

- Produces highest-quality image and sound samples (so far)
- A *generator* network maps random vectors **z** to data vector **x**
- A *discriminator* network: 2-class classifier, trained to distinguish actual data from **x**



Original paper by Goodfellow et al.: https://arxiv.org/abs/1406.2661

In the figure, green is the distribution resulting from randomizing **z** uniformly and passing the values through the generator to generate corresponding **x**. The black dots represent the real data distribution. Blue denotes the so-called discriminative distribution, i.e., whether the probability of some x should be increased or decreased.

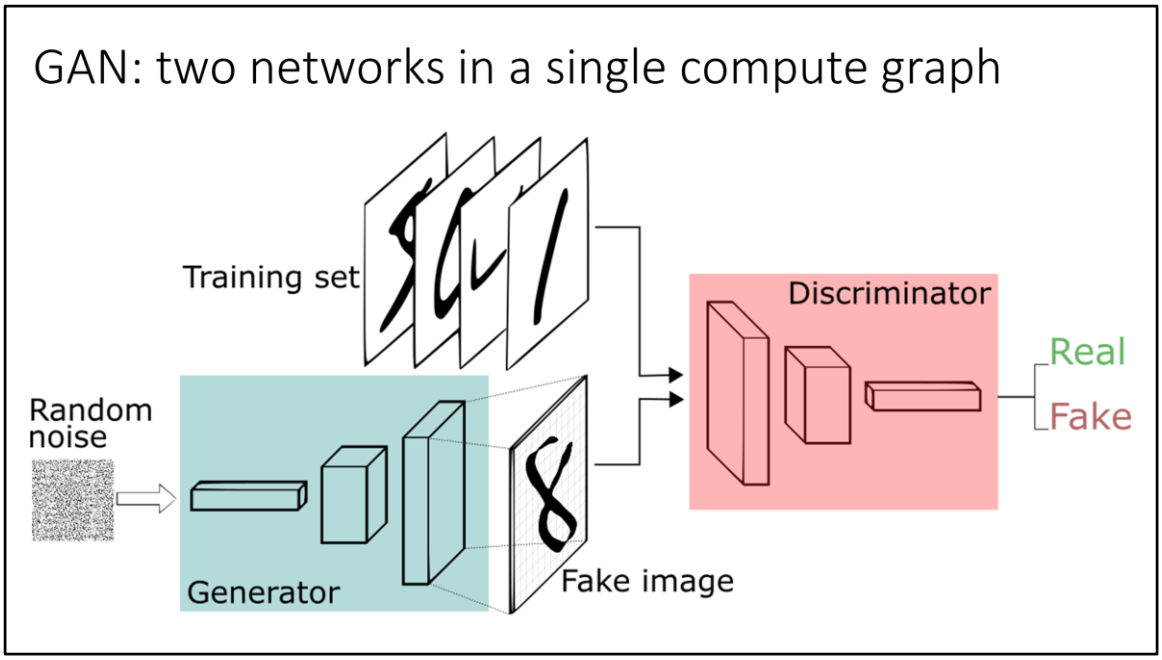


Image source: https://deeplearning4j.org/generative-adversarial-network

Generator is similar to the decoder part in encoder-decoder networks, and the discriminator is similar to an encoder.

GANs are trained in interleaved manner

1. Keep generator fixed, optimize discriminator to maximize discriminator performance with both generated and real data
2. Keep discriminator fixed, optimize generator to minimize discriminator performance with generated data

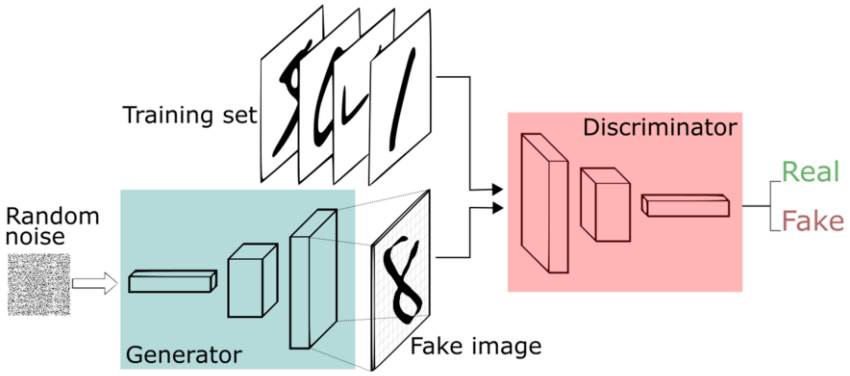
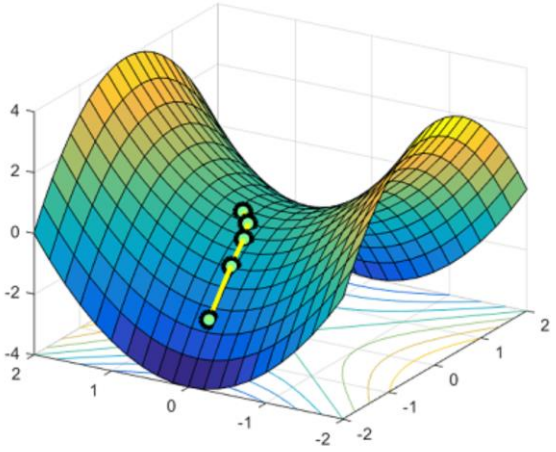


Image source: https://deeplearning4j.org/generative-adversarial-network

Full details in the Goodfellow et al. paper https://arxiv.org/abs/1406.2661

Problem: saddle-point optimization is unstable



In effect, GANs try to find a saddle point, as pointed out by Tom Goldstein: https://www.youtube.com/watch?v=78vq6kgsTa8 (”What do neural loss surfaces look like?”)

However, the optimization may easily fall off the saddle, i.e., the discriminator or generator may “win” the competition

Improving GAN training stability

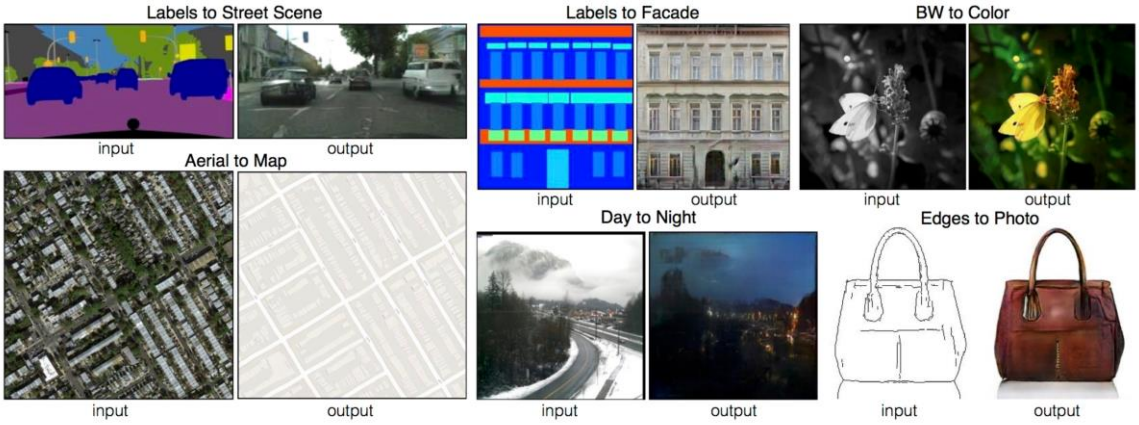
Yadav, A., Shah, S., Xu, Z., Jacobs, D., & Goldstein, T. (2017). Stabilizing Adversarial Nets With Prediction Methods. *arXiv preprint arXiv:1705.07364*. (Using saddle-point optimization theory)

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of Wasserstein gans. In *Advances in Neural Information Processing Systems* (pp. 5769-5779).

Demos and source code

- Browser-based interactive visualization: https://cs.stanford.edu/people/karpathy/gan/
- Browser-based image-to-image (pix2pix) translation: https://affinelayer.com/pixsrv/
- An accessible tutorial: https://deeplearning4j.org/generative-adversarial-network
- State of the art code from NVIDIA:
- https://github.com/tkarras/progressive_growing_of_gans
- https://github.com/NVlabs/stylegan
- https://github.com/NVIDIA/pix2pixHD

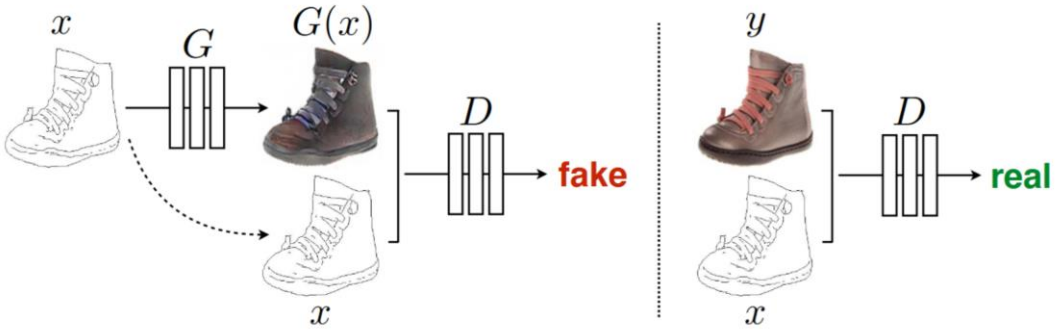
pix2pix (Isola et al. 2017)



https://arxiv.org/pdf/1611.07004.pdf

pix2pix (Isola et al. 2017)

- pix2pix is an example of conditional GAN
- Generator input: edge map, noise
- Discriminator input: edge map, real or generated images



https://arxiv.org/pdf/1611.07004.pdf

Conditional generative model: Generation is not completely random but conditioned on some extra variables such as the edge maps.

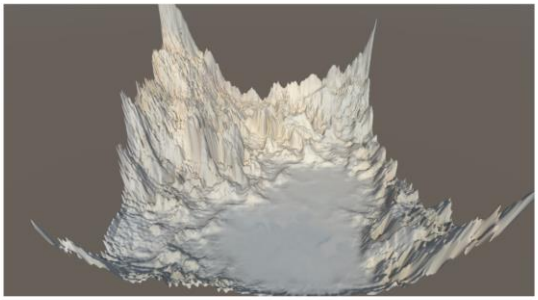
The noise is interpreted as the latent random variables that encode the variation possible without changing the conditioning variable values.

Other example: A game playing agent might sample an action conditioned by the current observation or game state.

pix2pixHD
https://github.com/NVIDIA/pix2pixHD

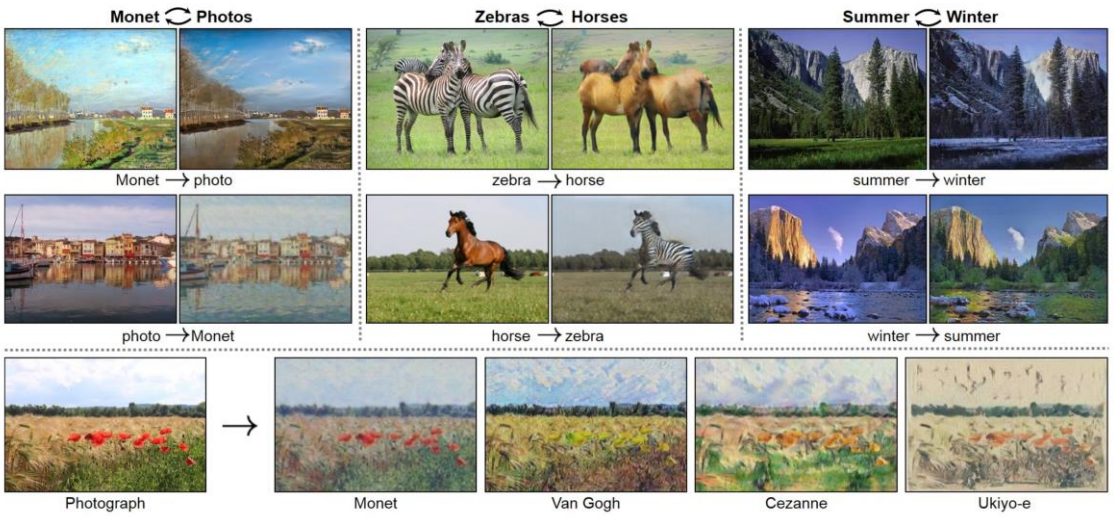
Generating terrain (Beckham 2017)

- Generate heightmaps with GAN
- Use pix2pix to generate textures from heightmaps
- Not yet super good results (dataset problem?)



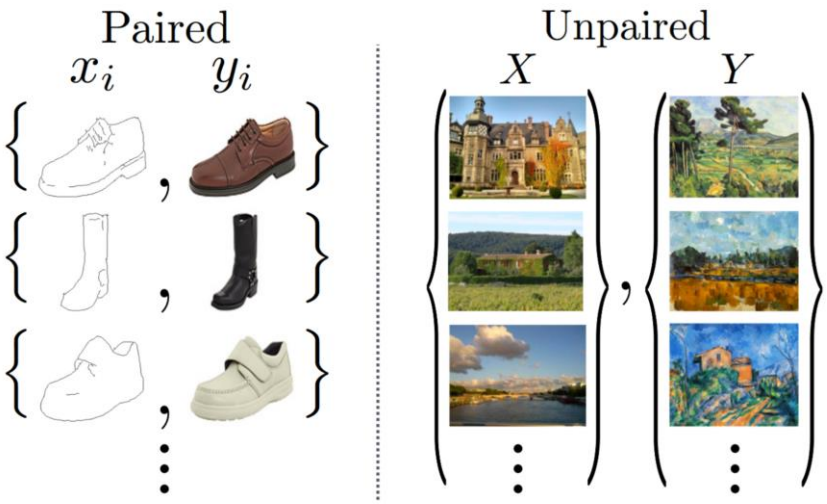
https://arxiv.org/pdf/1707.03383.pdf
https://github.com/christopher-beckham/gan-heightmaps

CycleGAN (Jun-Yan Zhu et al. 2017)



https://github.com/junyanz/CycleGAN

CycleGAN: no training pairs needed!



x and y can be arbitrary samples from different but related image classes.

CycleGAN: no training pairs needed!

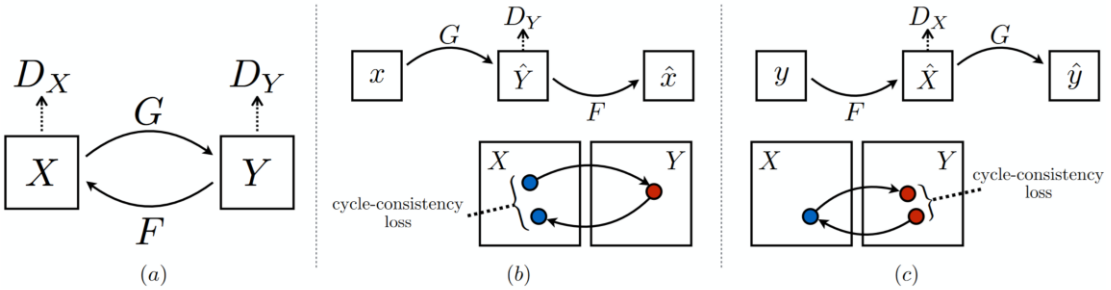


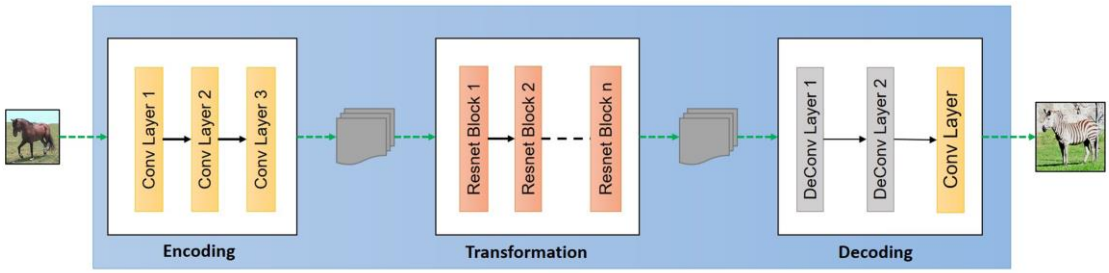
Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

How this works is there's two generator networks G and F that take the training samples as inputs instead of noise, and try to fool two discriminator networks D_X and D_Y

Basically, one tries to minimize the error in translating from one domain to another and back. This error is measured by the discriminator.

https://arxiv.org/pdf/1703.10593.pdf

CycleGAN: no training pairs needed!



Both G and F have this structure.

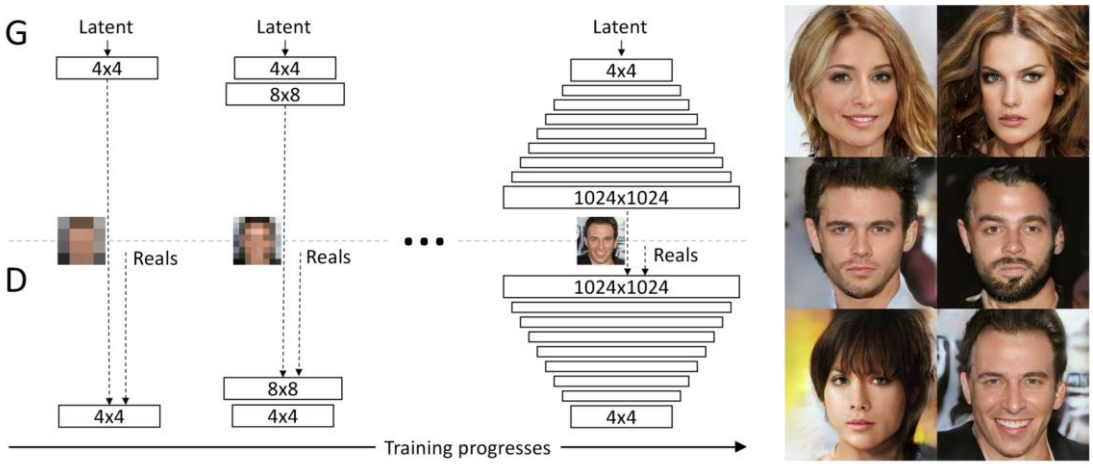
https://hardikbansal.github.io/CycleGANBlog/

Progressive GANs (Karras et al. 2017)

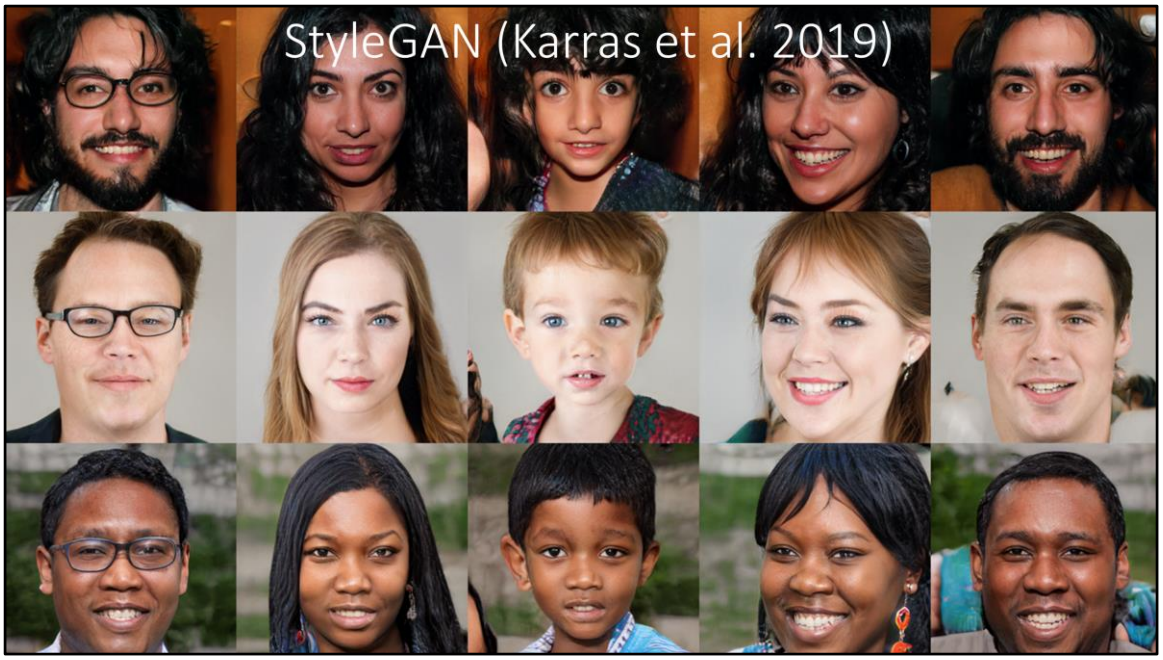
Progressively training with higher and higher resolution allows megapixel outputs.

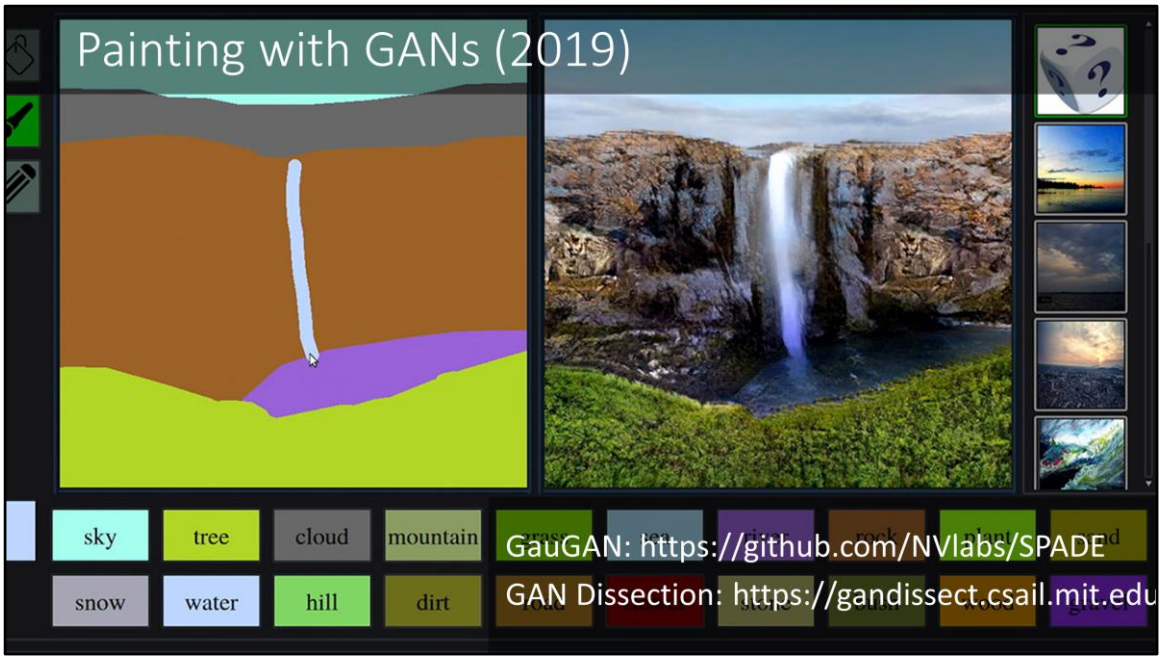
https://arxiv.org/pdf/1710.10196.pdf

Progressive GANs (Karras et al. 2017)



https://arxiv.org/pdf/1710.10196.pdf


StyleGAN (Karras et al. 2019)


Painting with GANs (2019)
sky
tree
cloud
mountain
snow
water
hill
dirt
GauGAN: https://github.com/NVlabs/SPADE
GAN Dissection: https://gandissect.csail.mit.edu

Glow: Generative Flow with Invertible 1×1 Convolutions

Diederik P. Kingma[*†], **Prafulla Dhariwal**[*]
[*]OpenAI
[†]Google AI

Abstract

Flow-based generative models (Dinh et al., 2014) are conceptually attractive due to tractability of the exact log-likelihood, tractability of exact latent-variable inference, and parallelizability of both training and synthesis. In this paper we propose *Glow*, a simple type of generative flow using an invertible 1×1 convolution. Using our method we demonstrate a significant improvement in log-likelihood on standard benchmarks. Perhaps most strikingly, we demonstrate that a flow-based generative model optimized towards the plain log-likelihood objective is capable of efficient realistic-looking synthesis and manipulation of large images. The code for our model is available at `https://github.com/openai/glow`.

1 Introduction

Two major unsolved problems in the field of machine learning are (1) data-efficiency: the ability to learn from few datapoints, like humans; and (2) generalization: robustness to changes of the task or its context. AI systems, for example, often do not work at all when given inputs that are different

[*]Equal contribution.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Figure 1: Synthetic celebrities sampled from our model; see Section 3 for architecture and method, and Section 5 for more results.

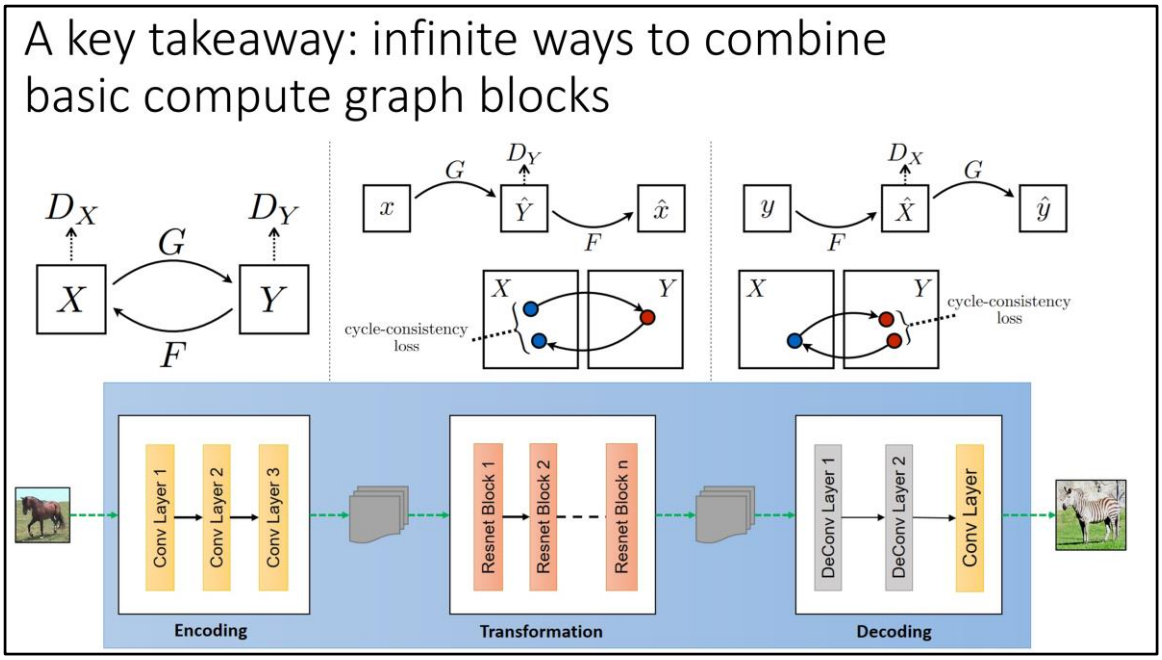
One problem of GANs is that there's no guarantee that the synthesized images actually cover the whole data instead of modeling only a part of it. So-called flow-based are the hot new thing that solves this, although GANs seem to still produce better visual quality. The flow-based models also give you a point-estimate of the probability density, i.e., how probable an image is in light of the training data. This can be useful in some applications.

Summary

- A *generative model* is needed when there are multiple possible outputs for a single input, and the outputs are not discrete-valued
- GANs are the dominant type of generative model
- Training GANs may be unstable, but things are improving rapidly

For discrete-valued outputs, one can train a discriminative model that outputs the probabilities of each possible value, with a separate output neuron for each value. This can then be used in 1) discriminative manner by finding the most probable value or 2) generative manner by sampling a value based on the probabilities. A practical example of this is training a game-playing agent with a discrete action set from from human player data.

Flow-based models are growing in popularity, and WAE also seems to have brought autoencoder-based models back to the spotlight.



Instead of math and algorithms, a lot of the innovation has recently been on finding new architectures that combine common building blocks in a novel way.

Key principles:

1) Create a compute graph that is end-to-end differentiable, allowing gradient propagation and training/optimization.

2) Select or design a loss function

3) In some cases, it makes sense to separate the training process into phases where parts of the graph are trained while others are kept fixed (e.g., the discriminator vs. generator training)