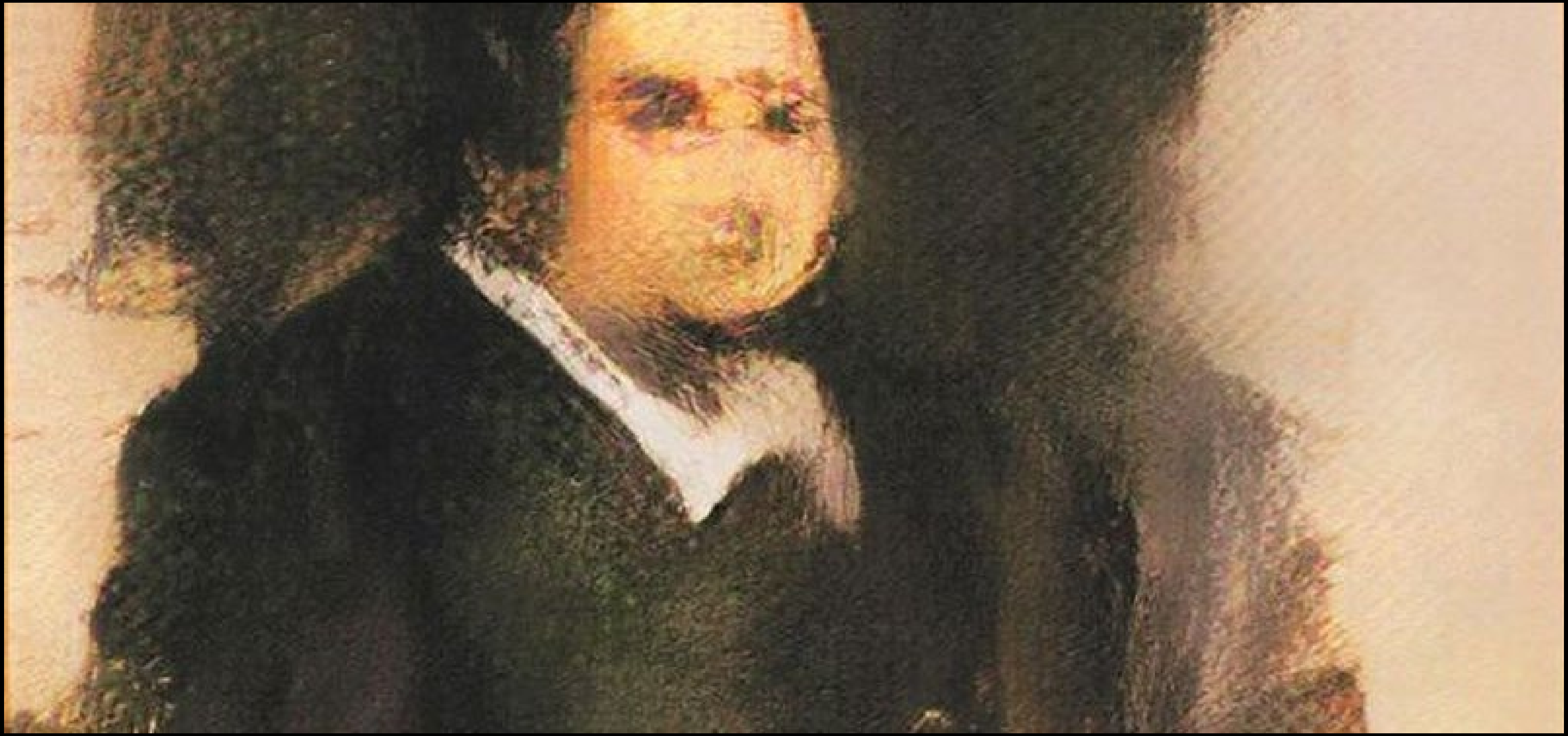


VAEs and GANs
CS 446/546

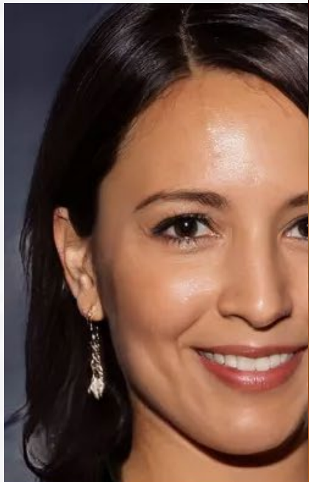
First…A Message from Mark

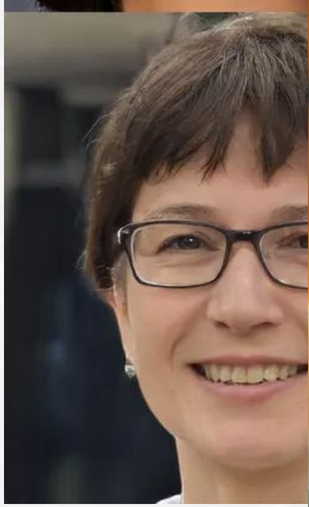
Outline

- Applications of GANs/VAEs
- VAEs
- GANs

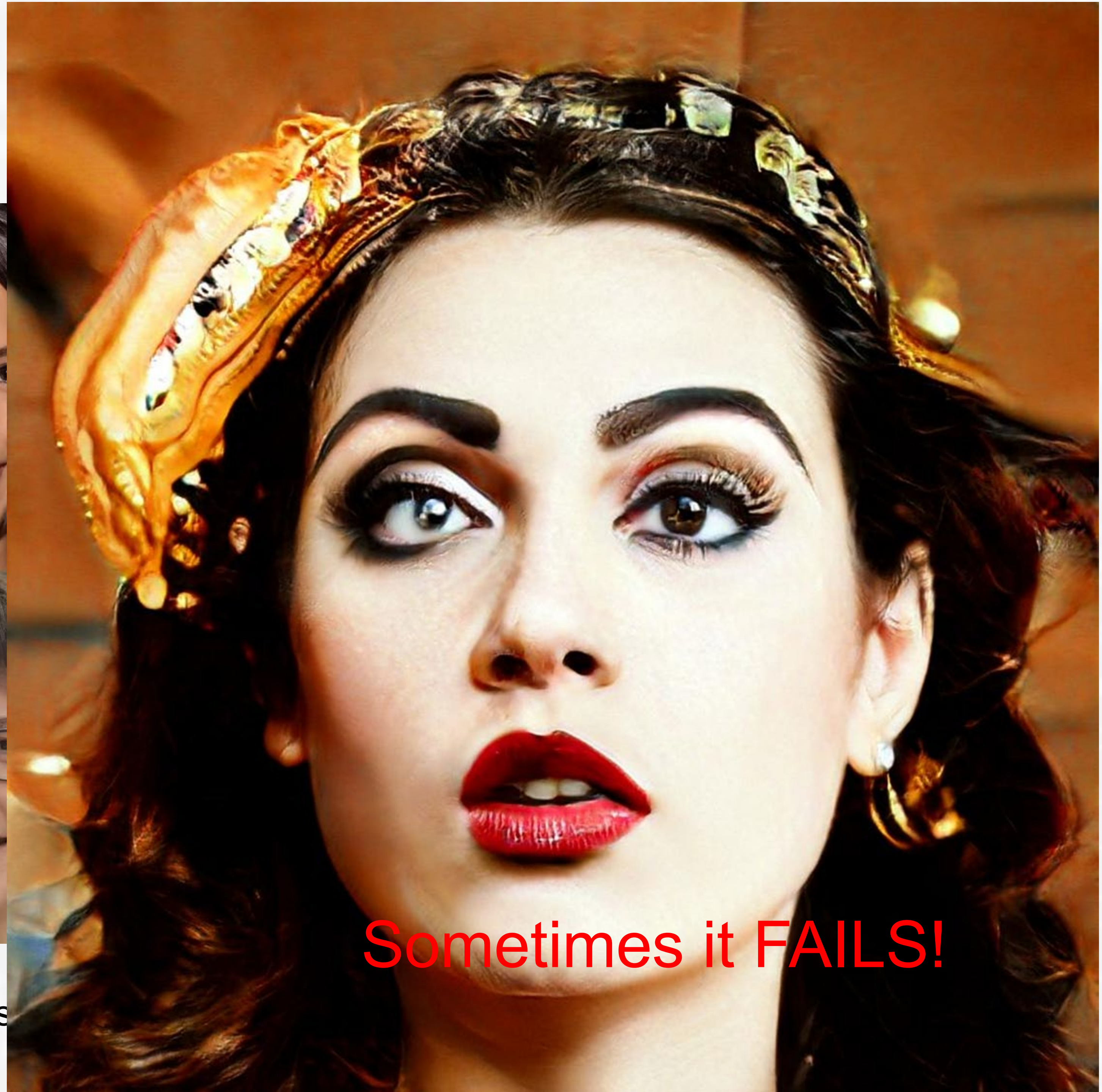
GAN Applications

Thispersondoesnotexist.com

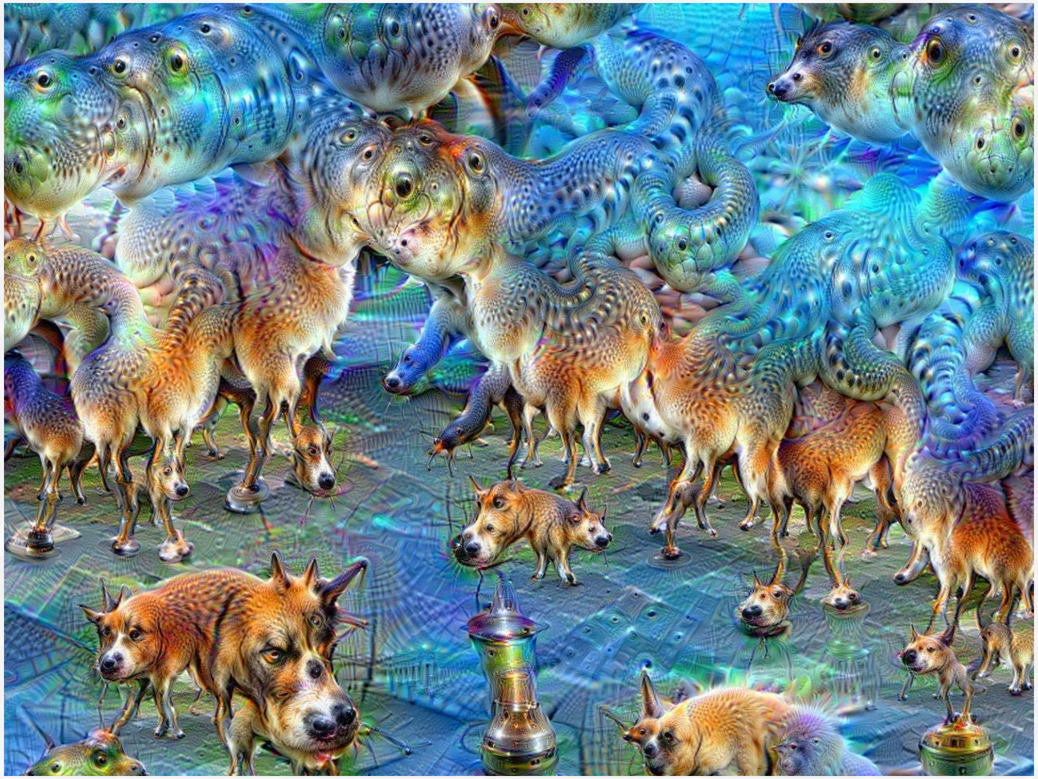




Thispersondoes



GAN Applications: Google DeepDream



*Technically not using a GAN, bust still fundamentally uses a CNN as a generative model.

https://www.youtube.com/watch?v=sh-MQboWJug

GAN Applications: Image/Video Upsampling

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi
Twitter
{cledig,ltheis,fhuszar,jcaballero,aacostadiaz,aaitken,atejani,jtotz,zehanw,wshi}@twitter.com

Abstract

Despite the breakthroughs in accuracy and speed of single image super-resolution using faster and deeper convolutional neural networks, one central problem remains largely unsolved: how do we recover the finer texture details when we super-resolve at large upscaling factors? The behavior of optimization-based super-resolution methods is principally driven by the choice of the objective function. Recent work has largely focused on minimizing the mean squared reconstruction error. The resulting estimates have high peak signal-to-noise ratios, but they are often lacking high-frequency details and are perceptually unsatisfying in the sense that they fail to match the fidelity expected at the higher resolution. In this paper, we present SRGAN, a generative adversarial network (GAN) for image super-resolution (SR). To our knowledge, it is the first framework capable of inferring photo-realistic natural images for 4× upscaling factors. To achieve this, we propose a perceptual loss function which consists of an adversarial loss and a content loss. The adversarial loss pushes our solution to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images. In addition, we use a content loss motivated by perceptual similarity instead

1. Introduction

The highly challenging task of estimating a high-resolution (HR) image from its low-resolution (LR) counterpart is referred to as super-resolution (SR). SR received substantial attention from within the computer vision research community and has a wide range of applications [63, 71, 43].

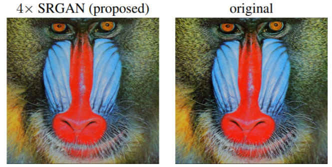
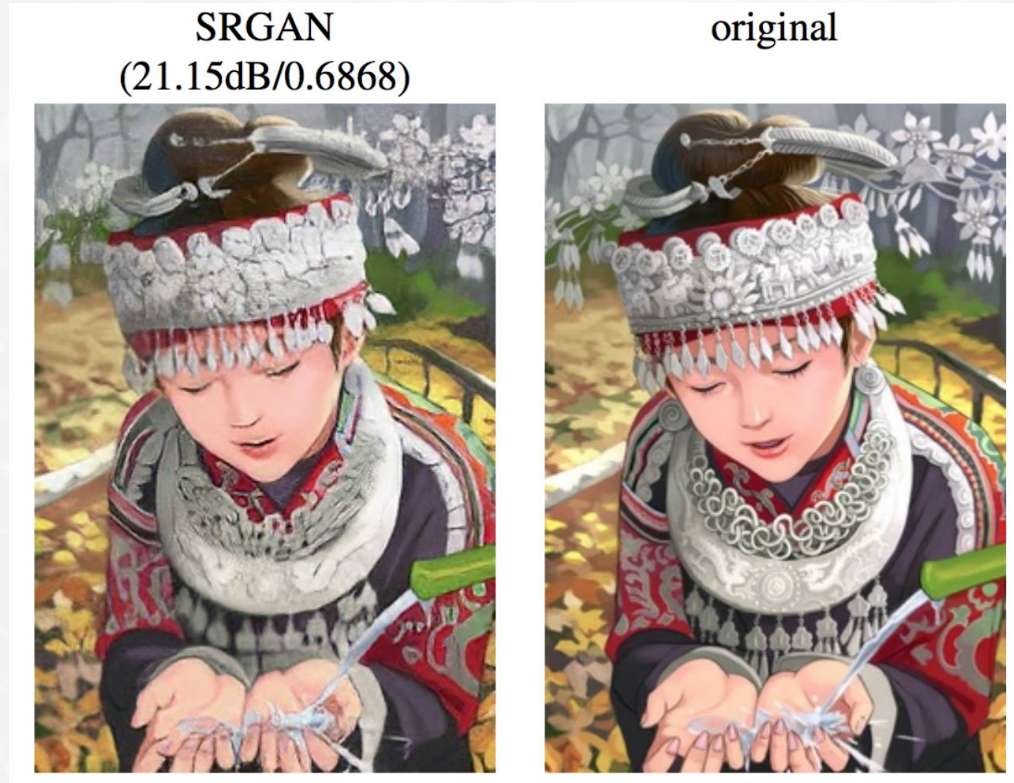


Figure 1: Super-resolved image (left) is almost indistinguishable from original (right). [4× upscaling]

The ill-posed nature of the underdetermined SR problem is particularly pronounced for high upscaling factors, for which texture detail in the reconstructed SR images is typically absent. The optimization target of supervised SR algorithms is commonly the minimization of the mean



*Teco Gan:
https://www.youtube.com/watch?v=pZXFXtfd-Ak&t=45s

LEARNING TEMPORAL COHERENCE VIA SELF-SUPERVISION FOR GAN-BASED VIDEO GENERATION

Mengyu Chu*, You Xie*, Jonas Mayer, Laura Leal-Taixé, Nils Thuerey
Department of Computer Science
Technical University of Munich
Munich, Germany
{mengyu.chu, you.xie, jonas.a.mayer, leal.taixe, nils.thuerey}@tum.de

ABSTRACT

We focus on temporal self-supervision for GAN-based video generation tasks. While adversarial training successfully yields generative models for a variety of

GAN Applications: Image Colorization

Image Colorization using Generative Adversarial Networks

Kamyar Nazeri, Eric Ng, and Mehran Ebrahimi

Faculty of Science, University of Ontario Institute of Technology
2000 Simcoe Street North, Oshawa, Ontario, Canada L1H 7K4
{kamyar.nazeri,eric.ng,mehran.ebrahimi}@uoit.ca
http://www.ImagingLab.ca/

Abstract. Over the last decade, the process of automatic image colorization has been of significant interest for several application areas including restoration of aged or degraded images. This problem is highly ill-posed due to the large degrees of freedom during the assignment of color information. Many of the recent developments in automatic colorization involve images that contain a common theme or require highly processed data such as semantic maps as input. In our approach, we attempt to fully generalize the colorization procedure using a conditional Deep Convolutional Generative Adversarial Network (DCGAN). The network is trained over datasets that are publicly available such as CIFAR-10 and Places365. The results between the generative model and traditional deep neural networks are compared.

1 Introduction

The automatic colorization of grayscale images has been an active area of research in machine learning for an extensive period of time. This is due to the large variety of applications such color restoration and image colorization for animations. In this manuscript, we will explore the method of colorization using generative adversarial networks (GANs) proposed by Goodfellow et al. [1]. The network is trained on the datasets CIFAR-10 and Places365 [2] and its results will be compared with those obtained using existing convolutional neural networks (CNN).

Models for the colorization of grayscales began back in the early 2000s. In 2002, Welsh et al. [3] proposed an algorithm that colorized images through texture synthesis. Colorization was done by matching luminance and texture information between an existing color image and the grayscale image to be colorized. However, this proposed algorithm was defined as a forward problem, thus all solutions were deterministic. Levin et al. [4] proposed an alternative formulation to the colorization problem in 2004. This formulation followed an inverse approach, where the cost function was designed by penalizing the difference between each pixel and a weighted average of its neighboring pixels. Both of these proposed methods still required significant user intervention which made the solutions less than ideal.

GAN Applications: 3D GANs

Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling

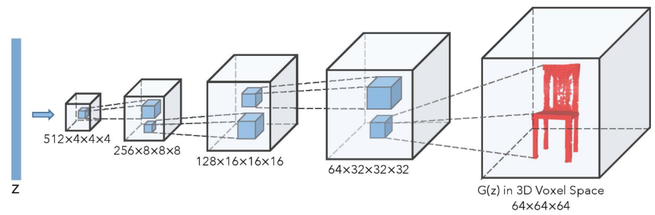


Figure 1: The generator of 3D Generative Adversarial Networks (3D-GAN)

Figure 2: Shapes synthesized by 3D-GAN

Abstract

We study the problem of 3D object generation. We propose a novel framework, namely 3D Generative Adversarial Network (3D-GAN), which generates 3D objects from a probabilistic space by leveraging recent advances in volumetric convolutional networks and generative adversarial nets. The benefits of our model are three-fold: first, the use of an adversarial criterion, instead of traditional heuristic criteria, enables the generator to capture object structure implicitly and to synthesize high-quality 3D objects; second, the generator establishes a mapping from a low-dimensional probabilistic space to the space of 3D objects, so that we can sample objects without a reference image or CAD models, and explore the 3D object manifold; third, the adversarial discriminator provides a powerful 3D shape descriptor which, learned without supervision, has wide applications in 3D object recognition. Experiments demonstrate that our method generates high-quality 3D objects, and our unsupervisedly learned features achieve impressive performance on 3D object recognition, comparable with those of supervised learning methods.

https://www.youtube.com/watch?v=mfx7uAkUtCI

GAN Applications: GauGAN

Semantic Image Synthesis with Spatially-Adaptive Normalization

Taesung Park[1,2*] Ming-Yu Liu[2] Ting-Chun Wang[2] Jun-Yan Zhu[2,3]

[1]UC Berkeley [2]NVIDIA [2,3]MIT CSAIL

Figure 1: Our model allows user control over both semantic and style as synthesizing an image. The semantic (e.g., the existence of a tree) is controlled via a label map (the top row), while the style is controlled via the reference style image (the leftmost column). Please visit our website for interactive image synthesis demos.

Abstract

We propose spatially-adaptive normalization, a simple but effective layer for synthesizing photorealistic images given an input semantic layout. Previous methods directly feed the semantic layout as input to the deep network, which is then processed through stacks of convolution, normalization, and nonlinearity layers. We show that this is suboptimal as the normalization layers tend to "wash away" semantic information. To address the issue, we propose using the input layout for modulating the activations in normal-

https://github.com/NVlabs/SPADE.

1. Introduction

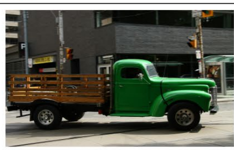
Conditional image synthesis refers to the task of generating photorealistic images conditioning on certain input data. Seminal work computes the output image by stitching pieces from a single image (e.g., Image Analogies [16]) or using an image collection [7, 14, 23, 30, 35]. Recent methods directly learn the mapping using neural networks [3, 6, 22, 47, 48, 54, 55, 56]. The latter methods are

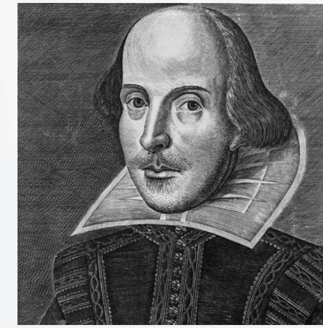
http://nvidia-research-mingyuliu.com/gaugan/

GAN Applications: Text/ Caption Generation

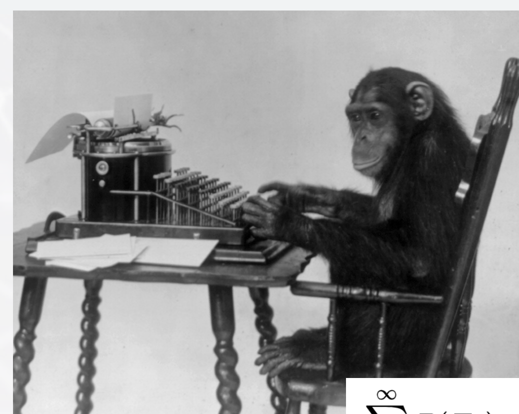
Generating Diverse and Accurate Visual Captions by Comparative Adversarial Learning

Dianqi Li[1]*, Qiuyuan Huang[2], Xiaodong He[3]*, Lei Zhang[2], Ming-Ting Sun[1]
[1]University of Washington, [2]Microsoft Research, [3]JD AI Research
{dianqili, mts}@uw.edu, xiaodong.he@jd.com, {leizhang, qihua}@microsoft.com

Pizza				
MLE	a pizza sitting on top of a white plate	a pizza sitting on top of a white plate	a close up of a pizza on a table	a pizza sitting on top of a pan
G-GAN	a pizza on a plate on a wooden table	a pizza sitting on a plate next to a glass of wine	the pizza is covered with cheese and tomatoes	a close up of a sliced pizza on a plate
CAL	a cheese pizza on a plate sits on a table	a plate of pizza and a glass of beer on the table	a pizza topped with lots of toppings is ready to be cut	a partially eaten pizza is being cooked on a pan
Car				
MLE	a green truck parked in a parking lot	a black truck is parked in a parking lot	a group of buses driving down a street	a city bus stopped at a bus stop
G-GAN	a green garbage truck in a business district	an antique black car sitting in a parking lot	a city street filled with taxis and buses	people are waiting in line as the bus travel down the road
CAL	a large green truck driving past a tall building	an old style truck parked in a parking space near a building	the city buses are driving through the traffic	people gather to a street where a bus get ready to board

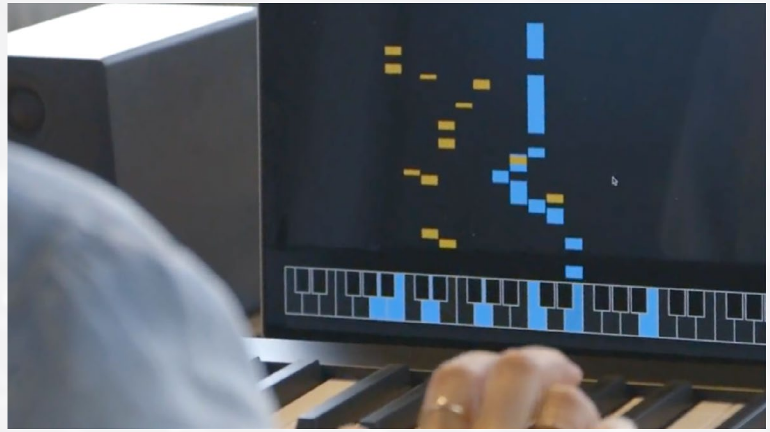


```
QUEENE:
I had thought thou hadst a Roman; for the oracle,
Thus by All bids the man against the word,
Which are so weak of care, by old care done;
Your children were in your holy love,
And the precipitation through the bleeding throne.

BISHOP OF ELY:
Marry, and will, my lord, to weep in such a one were prettiest;
Yet now I was adopted heir
Of the world's lamentable day,
To watch the next way with his father with his face?

ESCALUS:
The cause why then we are all resolved more sons.

VOLUMNIA:
O, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, it is no sin it shoul
And love and pale as any will to that word.

QUEEN ELIZABETH:
But how long have I heard the soul for this world,
And show his hands of life be proved to stand.

PETRUCHIO:
I say he look'd on, if I must be content
To stay him from the fatal of our country's bliss.
His lordship pluck'd from this sentence then for prey,
And then let us twain, being the moon,
were she such a case as fills m
```

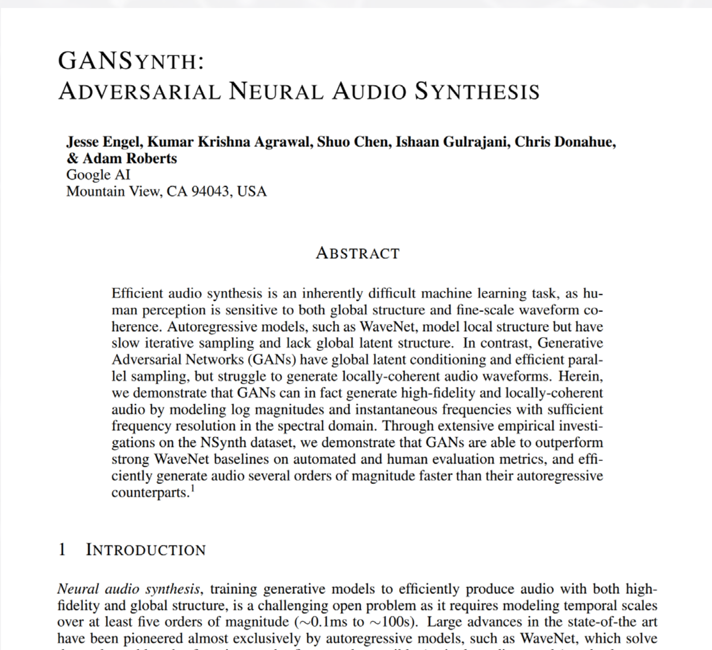


$$\sum_{k=1}^{\infty} P(E_k) = \sum_{k=1}^{\infty} p = \infty$$

GAN Applications: Music Generation



https://experiments.withgoogle.com/ai/ai-duet/view/



GANSYNTH:
ADVERSARIAL NEURAL AUDIO SYNTHESIS

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, & Adam Roberts
Google AI
Mountain View, CA 94043, USA

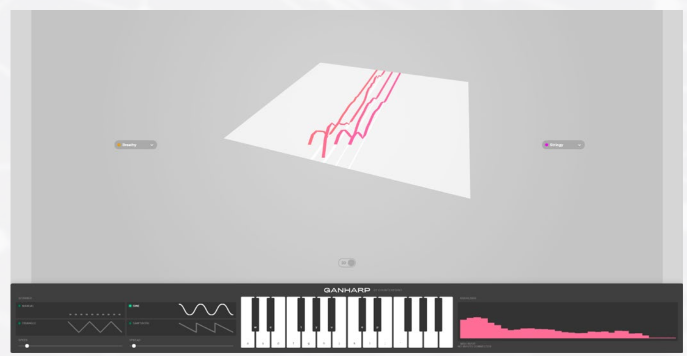
ABSTRACT

Efficient audio synthesis is an inherently difficult machine learning task, as human perception is sensitive to both global structure and fine-scale waveform coherence. Autoregressive models, such as WaveNet, model local structure but have slow iterative sampling and lack global latent structure. In contrast, Generative Adversarial Networks (GANs) have global latent conditioning and efficient parallel sampling, but struggle to generate locally-coherent audio waveforms. Herein, we demonstrate that GANs can in fact generate high-fidelity and locally-coherent audio by modeling log magnitudes and instantaneous frequencies with sufficient frequency resolution in the spectral domain. Through extensive empirical investigations on the NSynth dataset, we demonstrate that GANs are able to outperform strong WaveNet baselines on automated and human evaluation metrics, and efficiently generate audio several orders of magnitude faster than their autoregressive counterparts.[1]

1 INTRODUCTION

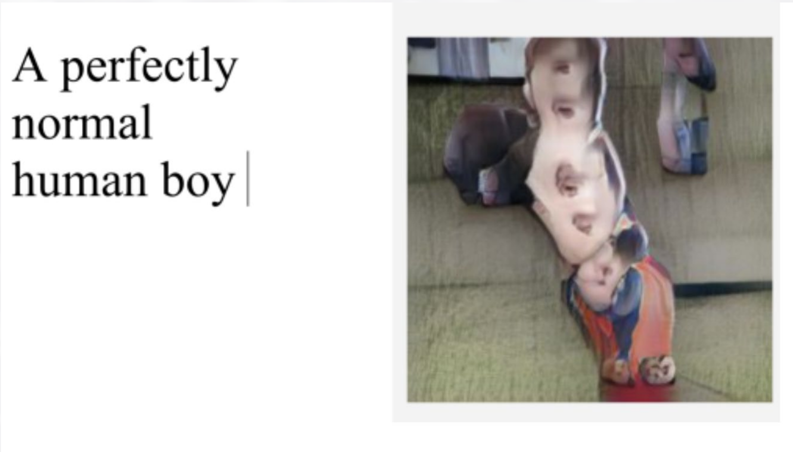
Neural audio synthesis, training generative models to efficiently produce audio with both high-fidelity and global structure, is a challenging open problem as it requires modeling temporal scales over at least five orders of magnitude (∼0.1ms to ∼100s). Large advances in the state-of-the art have been pioneered almost exclusively by autoregressive models, such as WaveNet, which solve

https://storage.googleapis.com/magentadata/papers/gansynth/index.html



https://ganharp.ctpt.co/

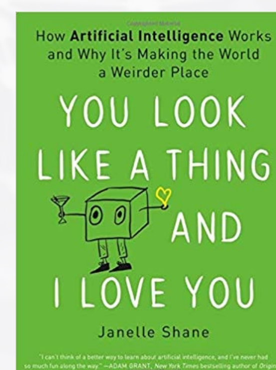
GAN FAILS


A perfectly normal human boy


HEARTED TREAT
FART BOOBY
LUBRICANT
SO BE IT
MILD FEEL
OKAY BRO
CRAP
BUNS BUNS BUNS
LOVING HORN
HA HA HA
THE END
BIOHAZARD
FLYING WHALE
CANDY HOLE
ANTS CAN STAY
DOOMED
FIGHT FIGHT FIGHT
NO BLINKING
T-REX
(shuffle)
MOUTHY HAMSTER
BUTTOCK
BODY PARTY!
THE OO
FLUFF
ALL HAIL THE CHICKEN
KITTEN BAND
WOOF HOOFHOOF
WICKED *EAT A TREE*
BATH TOWELS
YES YES YES YES YES
SUPER BEAR
BALLS
HORSE LOOP
DANCING ON A LOAF
BEANS


Festus
Freshly Marbled Water, Only 2% Sulfur. Perfect for when your face emits ultraviolet light, this decadent beret and floral-laden concoction adds a warm undertone of fresh scent, with notes of water, lemon essence, wabi wax, and bergamot.


Chrysanthemum
Traditionally associated with bad post-apocalyptic stories, this drink conjures up images of a dark lab, smouldering and broken. A smoky, fruity scent radiates from the lily of the valley, and with every whiff of citrus and tropical fruit on hand, each smell quenches the 1000 notes of chrysanthemum.

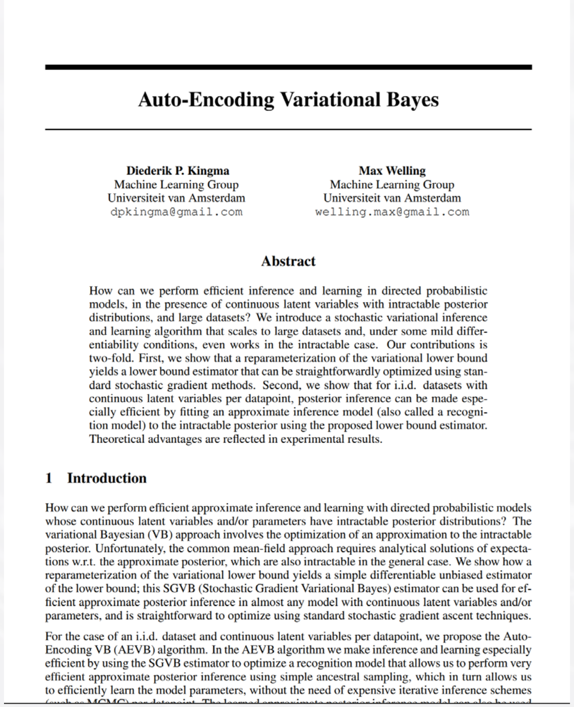

How Artificial Intelligence Works and Why It's Making the World a Weirder Place
YOU LOOK LIKE A THING AND I LOVE YOU
Janelle Shane

VAE Background: Autoencoders

- Kingma and Welling published "Auto-Encoding Variational Bayes" in 2013.

- Recall that an Autoencoder (AE) is a (symmetric) feed-forward NN containing a **bottleneck layer** and trained using **reconstruction loss**.

- AE can naturally be divided into two comparable components: An **encoder network** and a **decoder network**. The encoder induces a form of dimensionality reduction (e.g. PCA), while the decoder can be used to generate synthetic data.



Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

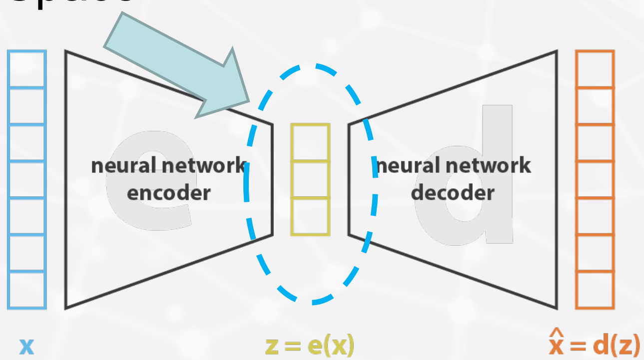
How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

1 Introduction

How can we perform efficient approximate inference and learning with directed probabilistic models whose continuous latent variables and/or parameters have intractable posterior distributions? The variational Bayesian (VB) approach involves the optimization of an approximation to the intractable posterior. Unfortunately, the common mean-field approach requires analytical solutions of expectations w.r.t. the approximate posterior, which are also intractable in the general case. We show how a reparameterization of the variational lower bound yields a simple differentiable unbiased estimator of the lower bound; this SGVB (Stochastic Gradient Variational Bayes) estimator can be used for efficient approximate posterior inference in almost any model with continuous latent variables and/or parameters, and is straightforward to optimize using standard stochastic gradient ascent techniques.

For the case of an i.i.d. dataset and continuous latent variables per datapoint, we propose the Auto-Encoding VB (AEVB) algorithm. In the AEVB algorithm we make inference and learning especially efficient by using the SGVB estimator to optimize a recognition model that allows us to perform very efficient approximate posterior inference using simple ancestral sampling, which in turn allows us to efficiently learn the model parameters, without the need of expensive iterative inference schemes

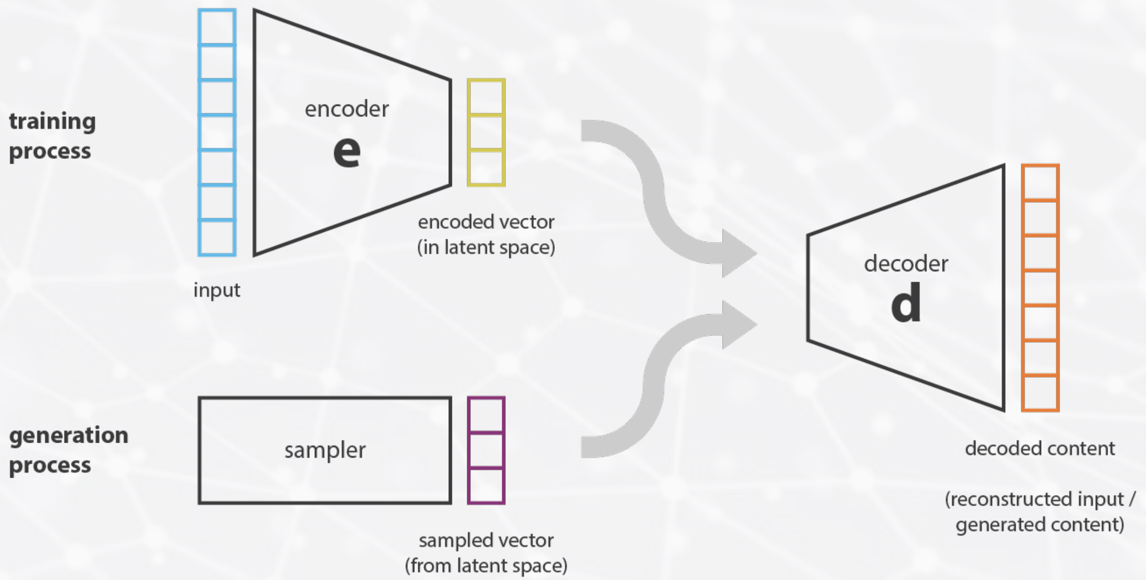
Feature Latent Space



$$\text{loss} = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2$$

VAE Background: Autoencoders

- Importantly, variational autoencoders (VAEs) add a stochastic mechanism (a **random vector**) that enables the network to generate synthetic outputs; additionally, VAEs **regularize the latent space**.



VAE Background: Variational Inference (brief)

- The goal of **variational inference** is to approximate a conditional density of **latent variables** (denoted z), given **observed variables** (denoted x), using optimization. This conditional density can be used to produce point or interval estimates for latent variables, form predictive densities of new data, etc.

- As usual, we can write the conditional density as:

$$p(z \mid x) = \frac{p(z, x)}{p(x)}$$

VAE Background: Variational Inference

$$p(z \mid x) = \frac{p(z,x)}{p(x)}$$

• Here the denominator contains the marginal density of the observations, also known as the **evidence**. We can calculate the evidence by marginalizing out the latent variables:

$$p(x) = \int p(z,x)dz$$

• In many cases, this integral is intractable and so we must resort to approximation techniques. On the one hand, we can use Monte Carlo techniques to generate a numerical approximation to the exact posterior using samples.

• By contrast, variational inference provides an analytical solution to the posterior distribution.

VAE Background: Variational Inference

$$p(z \mid x) = \frac{p(z, x)}{p(x)}$$

• In variational inference, we specify a family Q of density functions (e.g. Gaussians) over latent variables. Each $q(z) \in Q$ is a candidate approximation to the exact conditional.

• Our goal is to find the best candidate, i.e., the one closest in KL divergence to the exact condition. Accordingly, we solve the following optimization problem:

$$\boxed{q^*(z) = \arg\min_{q(z) \in Q} KL(q(z) \,\|\, p(z \mid x))}$$

• Once found, q* is the best approximation for the condition – with the family Q. The complexity of the family determined the complexity of this optimization problem.

VAE Background: Variational Inference

$$\boxed{q^*(z) = \arg\min_{q(z)\in Q} KL(q(z) \,||\, p(z \mid x))}$$

- This objective is, however, in general computable because it requires the aforementioned evidence:

-

VAE Background: Variational Inference

$$\boxed{q^*(z) = \underset{q(z) \in Q}{\arg\min}\, KL(q(z) \,\|\, p(z \mid x))}$$

- This objective is, however, in general computable because it requires the aforementioned evidence:

$$q^*(z) = \underset{q(z) \in Q}{\arg\min}\, KL(q(z) \,\|\, p(z \mid x))$$

$$= E_q[\log q(z)] - E_q[\log p(z \mid x)]$$

$$= E_q[\log q(z)] - E_q[\log p(z, x)] + \log p(x)$$

-

$$p(x) = \int p(z, x) dz$$

VAE Background: Variational Inference

$$\boxed{q^*(z) = \arg\min_{q(z)\in Q} KL(q(z) \,\|\, p(z \mid x))}$$

- Because we cannot compute the KL-divergence directly, we instead optimize an alternative objective that is equivalent to the KL-divergence up to a constant; this alternative function is called the **evidence lower-bound** (ELBO):

$$\boxed{\text{ELBO(q)}=E_q[\log p(z,x)] - E_q[\log q(z)]}$$

$$q^*(z) = \arg\min_{q(z)\in Q} KL(q(z) \,\|\, p(z \mid x))$$
$$= E_q[\log q(z)] - E_q[\log p(z,x)] + \log p(x)$$

- The ELBO is the negative KL divergence of q*, plus logp(x) (which is a constant with respect to q(z)).

- **Maximizing the ELBO is equivalent to minimizing the KL-divergence**.

VAE Background: Variational Inference

$$\boxed{\text{ELBO(q)}=E_q[\log p(z,x)]-E_q[\log q(z)]}$$

- Let's further analyze ELBO:

$$\begin{aligned}\text{ELBO(q)} &= E_q[\log p(z)]+E_q[\log p(x \mid z)]-E_q[\log q(z)] \\ &= E_q[\log p(x \mid z)]-KL(q(z) \,\|\, p(z))\end{aligned}$$

- Notice that ELBO is maximal when: (1) the latent variables explain the data (the likelihood expressed by the first term) and (2) when the variational density is close to the prior.

Another property of ELBO is that it lower-bounds the (log) evidence, $\log p(x) \geq ELBO(q)$ for any q(z).

To see this, note: $$\log p(x) = KL(q(z) \,\|\, p(z \mid x) + ELBO(q)$$

(recall that $KL \geq 0$ – why?)

VAE

• In summary, **the ELBO defines the objective function underlying variational inference**.

• However, in order to complete the specification of this objective function, we still need to define the ELBO with respect to the previously mentioned family of densities, Q.

VAE

• In summary, the ELBO defines the objective function underlying variational inference. However, in order to complete the specification of this objective function, we still need to define the ELBO with respect to the previously mentioned family of densities, Q.

• There are, naturally, many different families from which to choose. In practice for improved tractability, a common choice is the so-called **mean-field variational family**; for this set of functions, the latent variables are assumed to be mutually independent, so that each is governed by a distinct factor in the variational density.

$$q(z) = \prod_{j=1}^{m} q_j(z_j)$$

VAE

- Using the ELBO and mean-field family, we have now fully specified the approximate conditional inference problem as an optimization problem.

- In general, maximizing the ELBO is far from trivial. Again, there are many optimization techniques available for this task. One common approach is to use coordinate ascent variational inference (CAVI, due to Bishop*). CAVI iterative optimizes each factor of the mean-field variational density, while holding the others fixed – in this way we arrive at a local optimum for the ELBO.

Algorithm 1: Coordinate ascent variational inference (CAVI)

Input: A model $p(\mathbf{x}, \mathbf{z})$, a data set $\mathbf{x}$

Output: A variational density $q(\mathbf{z}) = \prod_{j=1}^{m} q_j(z_j)$

Initialize: Variational factors $q_j(z_j)$

while *the* ELBO *has not converged* **do**

    for $j \in \{1, \ldots, m\}$ **do**

        Set $q_j(z_j) \propto \exp\{\mathbb{E}_{-j}[\log p(z_j \mid \mathbf{z}_{-j}, \mathbf{x})]\}$

    end

    Compute $\text{ELBO}(q) = \mathbb{E}[\log p(\mathbf{z}, \mathbf{x})] - \mathbb{E}[\log q(\mathbf{z})]$

end

return $q(\mathbf{z})$

- Where $p(z_j | z_{-j}, x)$ denotes the total is the "total conditional" (i.e. $p(z_j)$ given x and all latent variables except z_j, as seen with Gibbs sampling.

*See: *Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.*

VAE

- We previously showed that minimizing our VAE objective is equivalent to **maximizing the ELBO**:

$$\boxed{\text{ELBO(q)} = E_q[\log p(x \mid z)] - KL(q(z \mid x) \,\|\, p(z))}$$

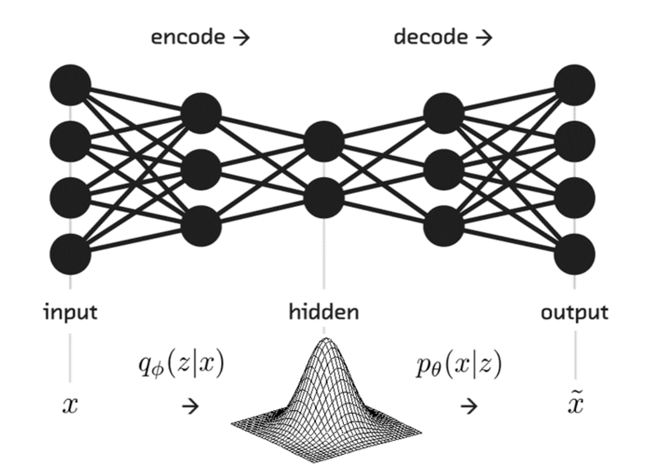
Notice that the RHS involves (3) quantities:

(1) q(z) (also written q(z|x)) a projection of the data x into the latent space

(2) z, the latent variable

(3) p(x|z) the distribution generating the data, given the latent variable.

- This structure is equivalent to an autoencoder, where q(z|x) is the encoder network; z is the encoded representation, and p(x|z) is the decoder network.



VAE

$$\boxed{\mathrm{ELBO(q)} = E_q[\log p(x \mid z)] - KL(q(z \mid x) \,\|\, p(z))}$$

- For a VAE, we assume that the encoder projects the input to a standard normal (i.e. $\mathrm{q}(z|x) = N(\mu(x), \Sigma(x))$; furthermore, we assume the latent distribution is a standard normal, i.e., $\mathrm{p}(z) = N(0, I)$.

In fact, this KL divergence term is analytically solvable:

$$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

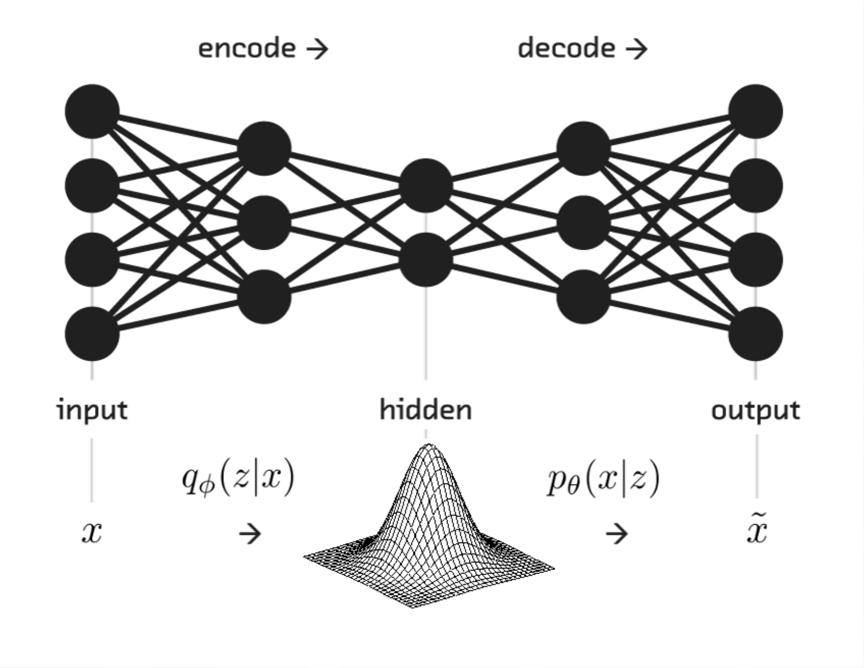
- In summary, $\mathrm{q}(z|x)$ is represented by a neural network, where the NN maps input data (x) to a mean vector $\mu(x)$ and (diagonal) covariance matrix $\Sigma(x)$ (the parameters of the latent space).

- By minimizing the indicated KL divergence, we encourage the latent space to conform with a standard Normal.

VAE

$$\text{ELBO(q)} = E_q[\log p(x \mid z)] - KL(q(z \mid x) \parallel p(z))$$

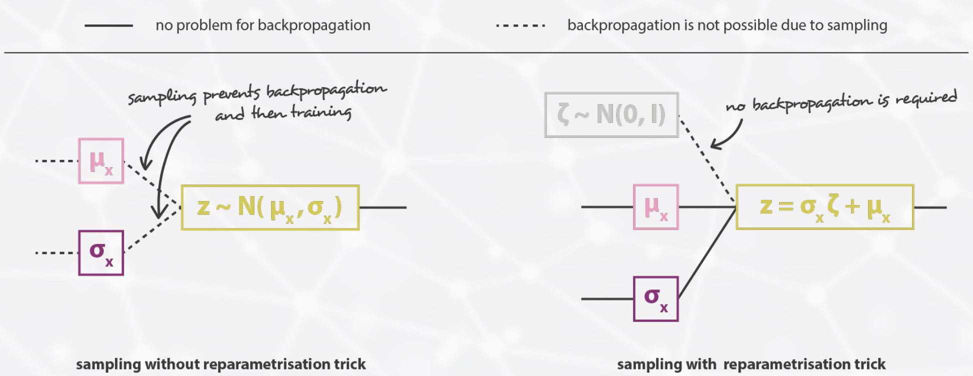
• Notice that the first term on the RHS is equivalent to MLE; so, to maximize this term we want to minimize the reconstruction error of the decoder with respect to a given an input x, the associated encoding z, and the reconstruction this encoding.

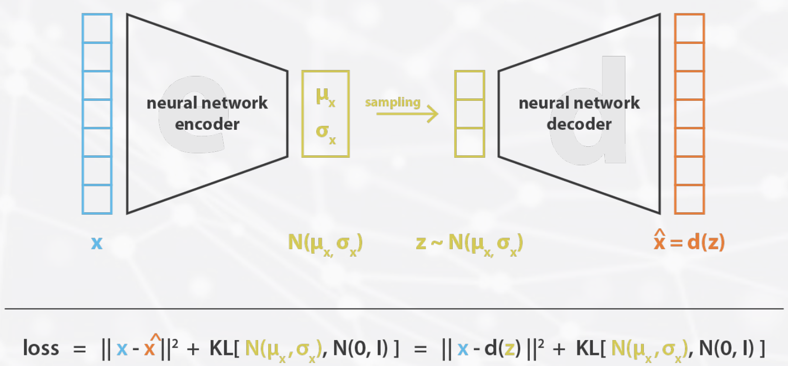


VAE

- We are almost done – however, recall that we want the latent parameter (z) corresponding with the input (x) to be sampled $z \sim N(\mu(x), \Sigma(x))$.

However, in order to enable training of the q(z|x) network using backpropagation, the sampling process must exist outside of the network itself. To achieve these, we use the so-call "**reparameterization trick**" (inverse sampling of a Gaussian).

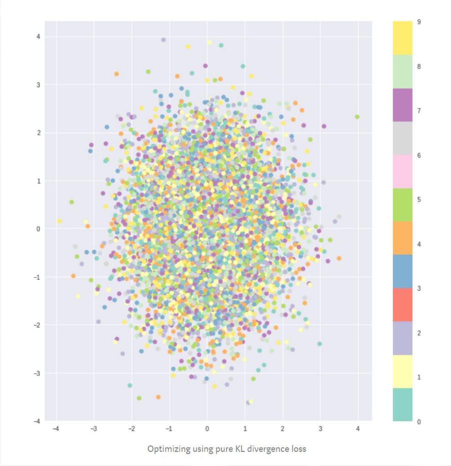


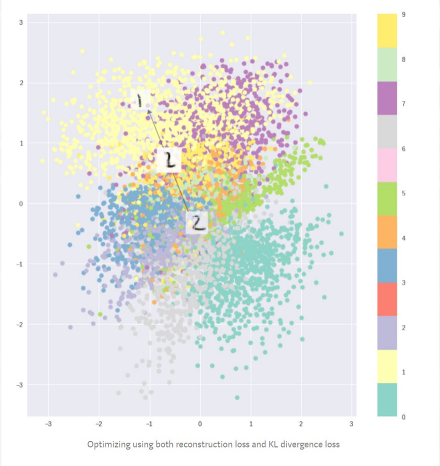


VAE

• Imposing a structure on the latent space (i.e. Gaussian) is a powerful idea for generative models. This approach has the effect of regularizing the latent space (and hence avoiding overfitting to the data).

• Optimizing with both reconstruction loss and KL divergence loss additionally enforces "similarity embedding" – which is to say, similar inputs to the VAE are mapped close to one another in the latent space.





VAE

- Reconstructing faces with a VAE:

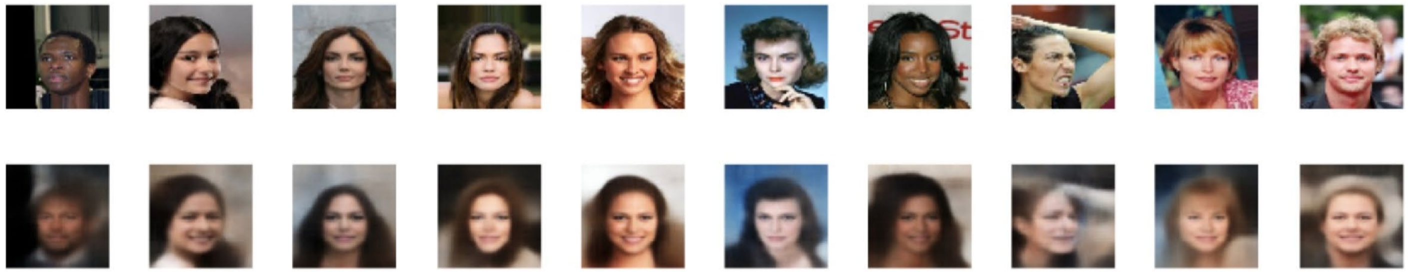


Figure 3-18. Reconstructed faces, after passing through the encoder and decoder

- Generating synthetic faces with a VAE:

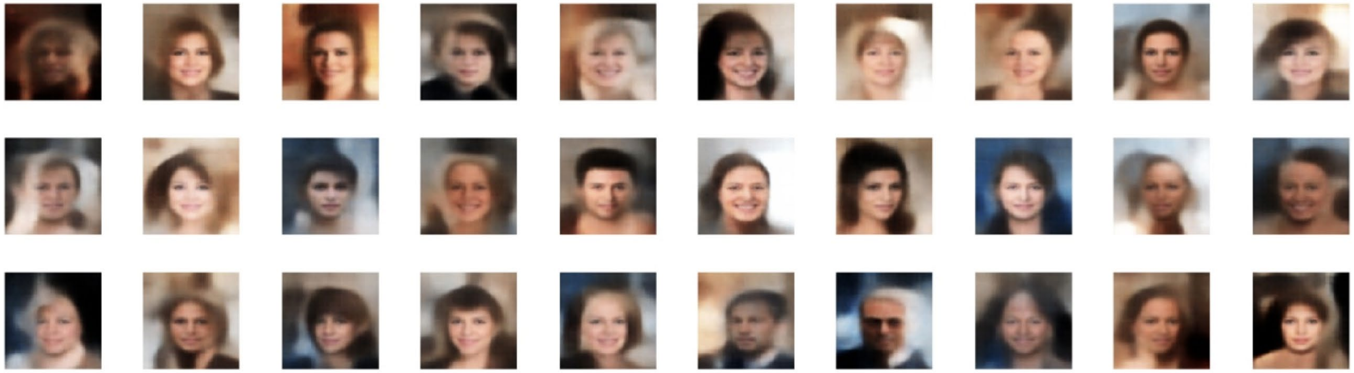


Figure 3-20. New generated faces

VAE: Latent Space Arithmetic

• Note that it is possible to manipulate the latent space associated with a generative model using **latent space arithmetic**.

•For instance, suppose we wish to vary a particular attribute of our generated synthetic data. The CelebA dataset includes annotations with various attributes, e.g., wearing hat, smiling, etc.

VAE: Latent Space Arithmetic

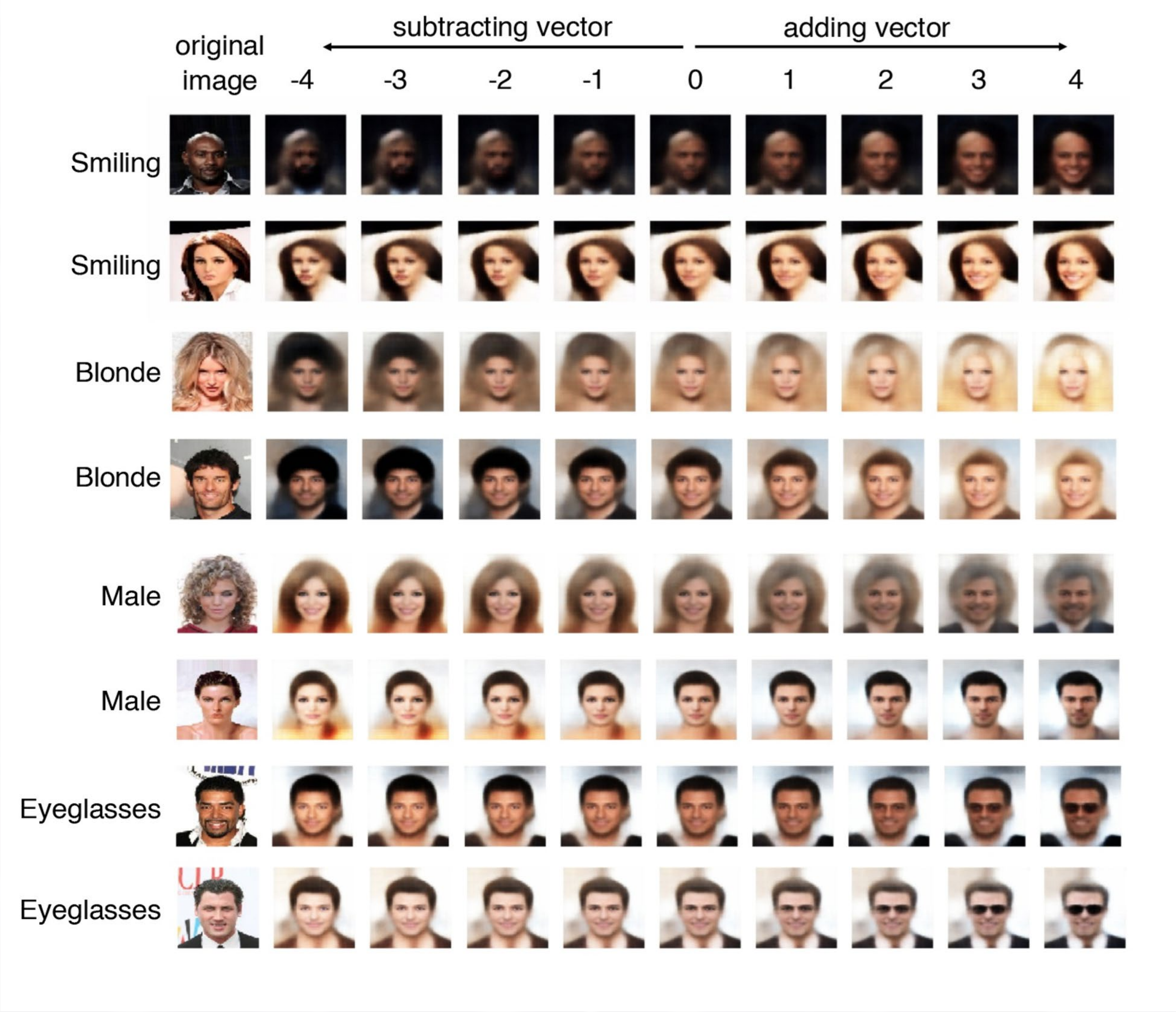
- In a similar vein to the latent space arithmetic seen with word-embedding models (e.g. Word2Vec), one can use vector arithmetic to meaningfully augment latent vectors.

- For example, if we want to generate faces that are "smiling", we could in principle take the average latent embedding of all the *faces with the attribute smiling* in our training set and subtract from this the average latent embedding of all the faces *without the attribute smiling*. This gives us a vector in the latent space pointing from "non-smiling" to "smiling".

- Now to apply "smiling" to a latent embedding, we apply the following transformation:

$$\mathbf{z}' = \mathbf{z} + \alpha(\mathbf{feature\_vector})$$

VAE: Latent Space Arithmetic



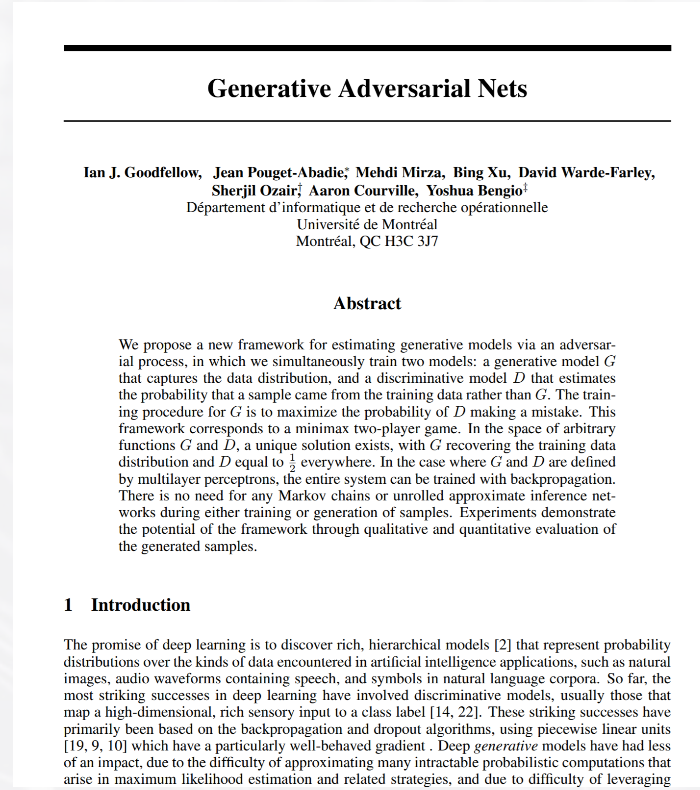
GAN

- The original GAN paper (Goodfellow *et al*, 2014) is one of the most influential ML papers in recent years.

- Simply put, a GAN is a battle between two adversaries: **the generator** and **the discriminator**.

- The generator attempts to convert random noise into observations that appear as though they were sampled from the original dataset.

- Conversely, the discriminator tries to predict whether an observation comes from the original dataset or is a forgery produced by the generator.



Generative Adversarial Nets

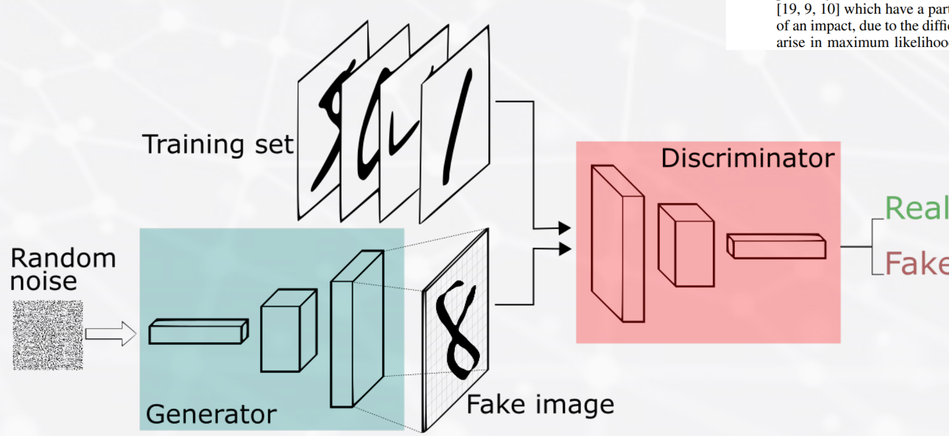
Ian J. Goodfellow, Jean Pouget-Abadie*, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

1 Introduction

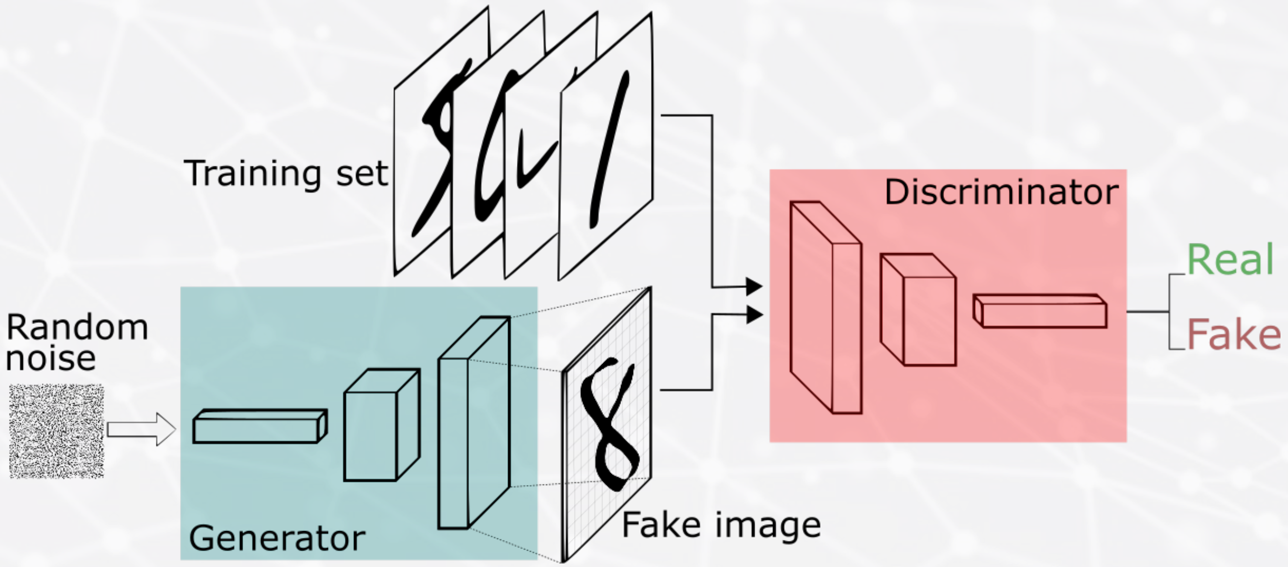
The promise of deep learning is to discover rich, hierarchical models [2] that represent probability distributions over the kinds of data encountered in artificial intelligence applications, such as natural images, audio waveforms containing speech, and symbols in natural language corpora. So far, the most striking successes in deep learning have involved discriminative models, usually those that map a high-dimensional, rich sensory input to a class label [14, 22]. These striking successes have primarily been based on the backpropagation and dropout algorithms, using piecewise linear units [19, 9, 10] which have a particularly well-behaved gradient . Deep *generative* models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging



GAN

• At the beginning of this process, the generator outputs noisy images and the discriminator predicts randomly.

• The key to GANs lies in **how we effect the training of the two networks in tandem**, so that as the generator becomes more adept at fooling the discriminator, the discriminator must adapt in order to maintain its ability to spot "fakes".



GAN

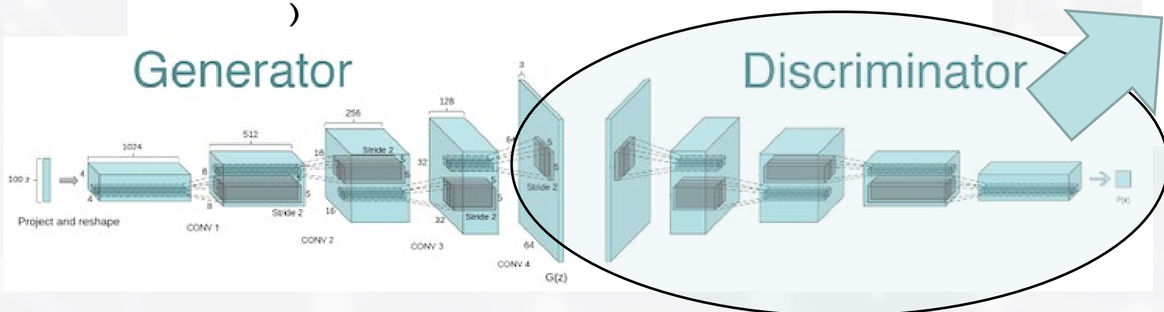
- Here's an example specification of a GAN; the architecture of the discriminator is given on the right.

Discriminator: define input; stack convolutional layers; flatten the last convolutional layer, etc.; note that a stride of size 2 in the conv layers will reduce the overall size of the tensor; the final "dense" layer (using sigmoid activation) ensures the output is a scalar in the range [0,1], corresponding with the probability that the input image is real.

```
gan = GAN(input_dim = (28,28,1)
        , discriminator_conv_filters = [64,64,128,128]
        , discriminator_conv_kernel_size = [5,5,5,5]
        , discriminator_conv_strides = [2,2,2,1]
        , discriminator_batch_norm_momentum = None
        , discriminator_activation = 'relu'
        , discriminator_dropout_rate = 0.4
        , discriminator_learning_rate = 0.0008
        , generator_initial_dense_layer_size = (7, 7, 64)
        , generator_upsample = [2,2, 1, 1]
        , generator_conv_filters = [128,64, 64,1]
        , generator_conv_kernel_size = [5,5,5,5]
        , generator_conv_strides = [1,1, 1, 1]
        , generator_batch_norm_momentum = 0.9
        , generator_activation = 'relu'
        , generator_dropout_rate = None
        , generator_learning_rate = 0.0004
        , optimiser = 'rmsprop'
        , z_dim = 100
        )
```

Layer (type)	Output Shape	Param #
discriminator_input (InputLa	(None, 28, 28, 1)	0
discriminator_conv_0 (Conv2D	(None, 14, 14, 64)	1664
activation_1 (Activation)	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
discriminator_conv_1 (Conv2D	(None, 7, 7, 64)	102464
activation_2 (Activation)	(None, 7, 7, 64)	0
dropout_2 (Dropout)	(None, 7, 7, 64)	0
discriminator_conv_2 (Conv2D	(None, 4, 4, 128)	204928
activation_3 (Activation)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0
discriminator_conv_3 (Conv2D	(None, 4, 4, 128)	409728
activation_4 (Activation)	(None, 4, 4, 128)	0
dropout_4 (Dropout)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 1)	2049

Total params: 720,833
Trainable params: 720,833
Non-trainable params: 0



GAN

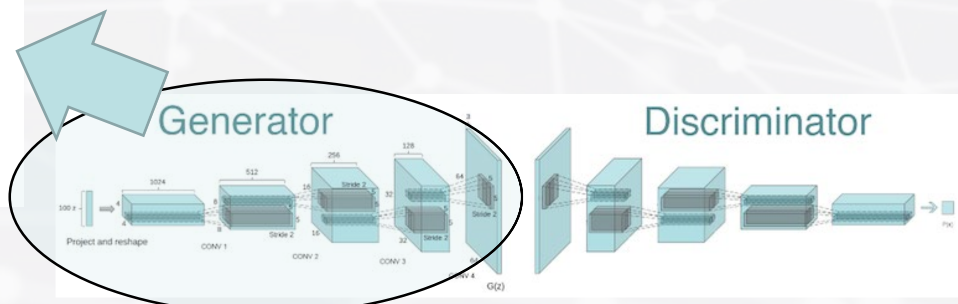
• The input to the generator is a vector, usually drawn from a MVN; the output is an image of the same size as the original dataset.

• The generator serves the same purpose as the decoder for a VAE, in that it converts a vector from the latent space into an image. The trope of mapping from a low-to-high dimensional space is common in DL; for a CNN, this operation is commonly known as **deconvolution** (also: **transposed convolution**).

Layer (type)	Output Shape	Param #
generator_input (InputLayer)	(None, 100)	0
dense_9 (Dense)	(None, 3136)	316736
batch_normalization_10 (Batc	(None, 3136)	12544
activation_36 (Activation)	(None, 3136)	0
reshape_4 (Reshape)	(None, 7, 7, 64)	0
up_sampling2d_10 (UpSampling	(None, 14, 14, 64)	0
generator_conv_0 (Conv2D)	(None, 14, 14, 128)	204928
batch_normalization_11 (Batc	(None, 14, 14, 128)	512
activation_37 (Activation)	(None, 14, 14, 128)	0
up_sampling2d_11 (UpSampling	(None, 28, 28, 128)	0
generator_conv_1 (Conv2D)	(None, 28, 28, 64)	204864
batch_normalization_12 (Batc	(None, 28, 28, 64)	256
activation_38 (Activation)	(None, 28, 28, 64)	0
generator_conv_2 (Conv2D)	(None, 28, 28, 64)	102464
batch_normalization_13 (Batc	(None, 28, 28, 64)	256
activation_39 (Activation)	(None, 28, 28, 64)	0
generator_conv_3 (Conv2D)	(None, 28, 28, 1)	1601
activation_40 (Activation)	(None, 28, 28, 1)	0

Total params: 844,161
Trainable params: 837,377
Non-trainable params: 6,784

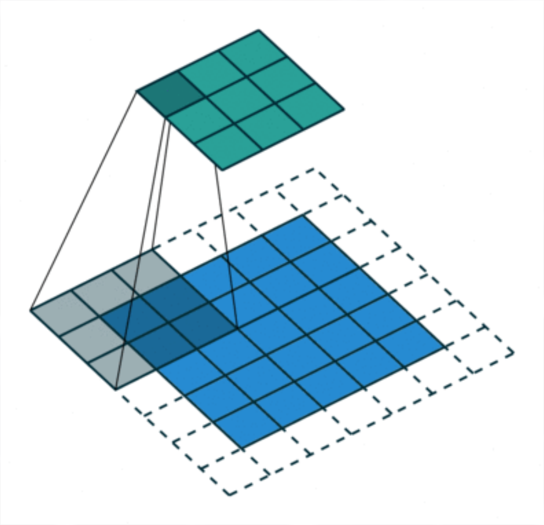
```
gan = GAN(input_dim = (28,28,1)
        , discriminator_conv_filters = [64,64,128,128]
        , discriminator_conv_kernel_size = [5,5,5,5]
        , discriminator_conv_strides = [2,2,2,1]
        , discriminator_batch_norm_momentum = None
        , discriminator_activation = 'relu'
        , discriminator_dropout_rate = 0.4
        , discriminator_learning_rate = 0.0008
        , generator_initial_dense_layer_size = (7, 7, 64)
        , generator_upsample = [2,2, 1, 1]
        , generator_conv_filters = [128,64, 64,1]
        , generator_conv_kernel_size = [5,5,5,5]
        , generator_conv_strides = [1,1, 1, 1]
        , generator_batch_norm_momentum = 0.9
        , generator_activation = 'relu'
        , generator_dropout_rate = None
        , generator_learning_rate = 0.0004
        , optimiser = 'rmsprop'
        , z_dim = 100
        )
```



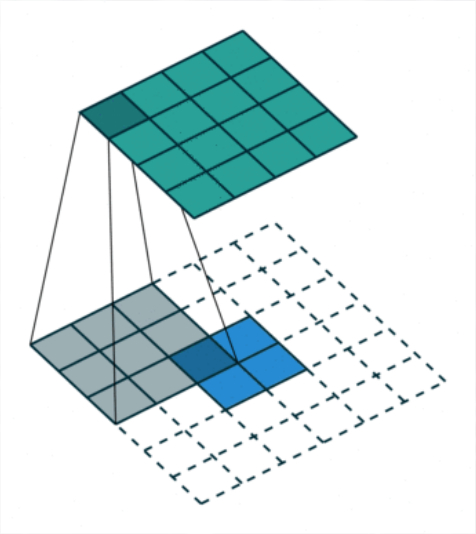
GAN: Transposed Convolution

• The transposed convolution operation is effected by performing a "backward strided convolution".

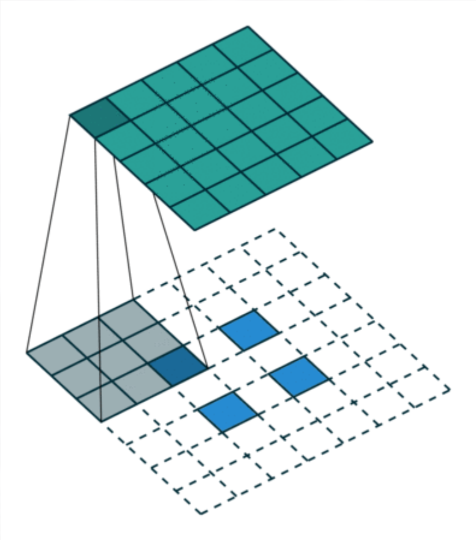
• In the images below, the blue maps are inputs; cyan maps are outputs.



Basic convolution with padding=1, stride =2



Transposed conv with no padding, no stride



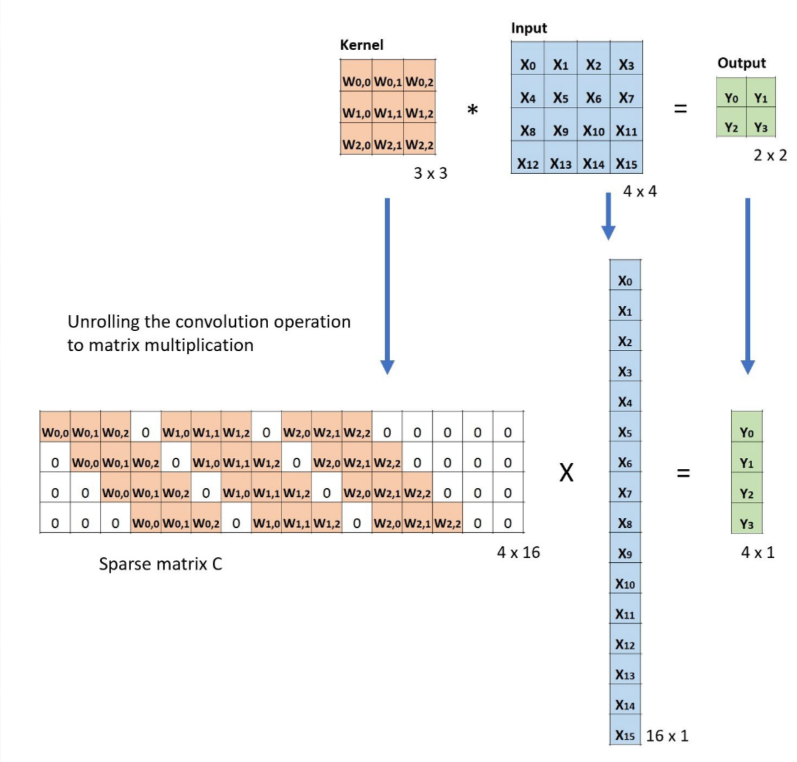
Transposed conv with no padding and stride

• Traditionally, one could achieve up-sampling by applying interpolation schemes (e.g. bilinear interpolation). Modern architectures such as NNs, however, **tend to let the network itself learn the proper transformation automatically**, without human intervention.

GAN: Transposed Convolution

• Let's dive a little deeper into the contrast between convolution and transposed convolution.

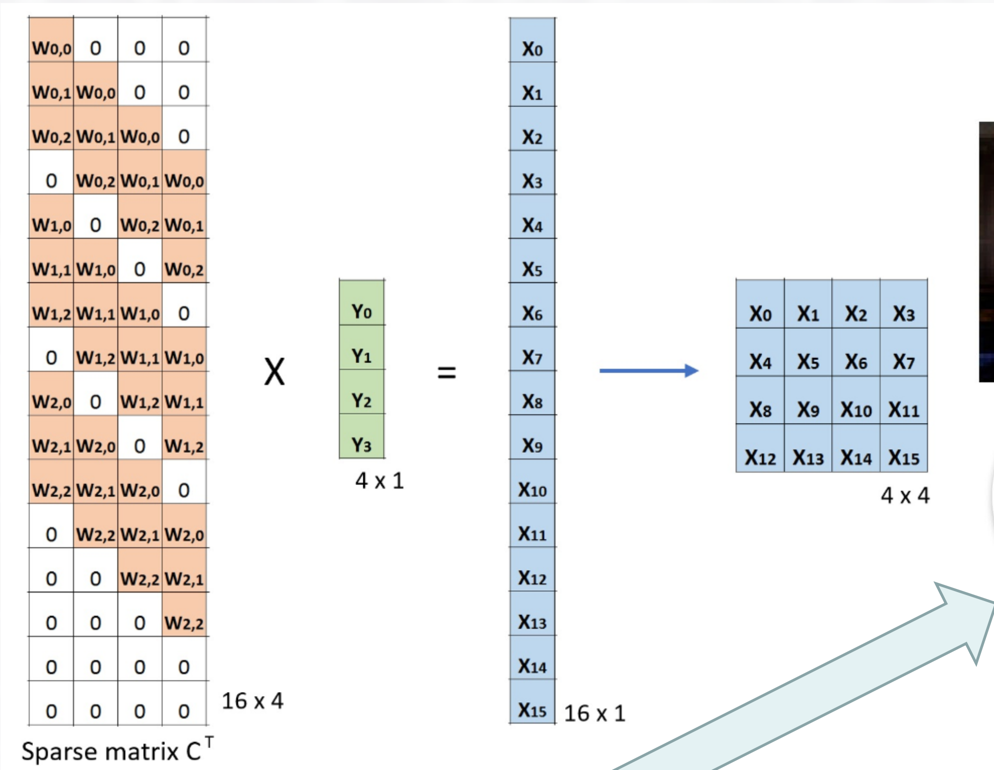
• With convolution, consider *C* as the kernel, *Large* as the input, and *Small* as the output image after convolution. Following convolution, we down-sample the large image into a small output image, i.e. **C x Large = Small**.

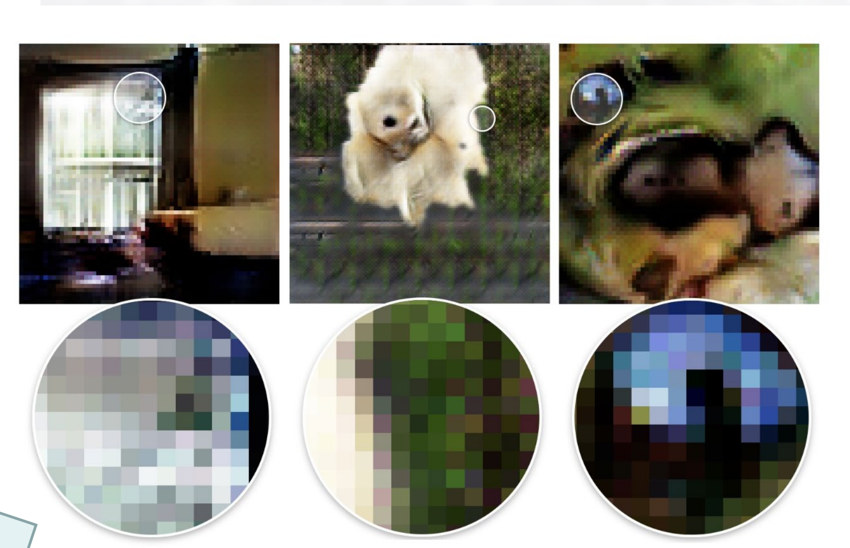


• In the example shown, we take a 4x4 input matrix and flatten it to 16x1; in addition we transform the 3x3 kernel into a 4x16 sparse, orthogonal matrix. Using matrix multiplication, the resultant matrix is 4x1, which we then subsequently transform back to a 2x2 output.

GAN: Transposed Convolution

• If, we multiply the equation **C x Large = Small**, by C^T, we arrive at: **C^TxSmall = Large**. In this way multiplication by the transposed convolution yields an up-sampling procedure. (for reference: we encountered this operation previously when discussing Hinton's work with AEs).





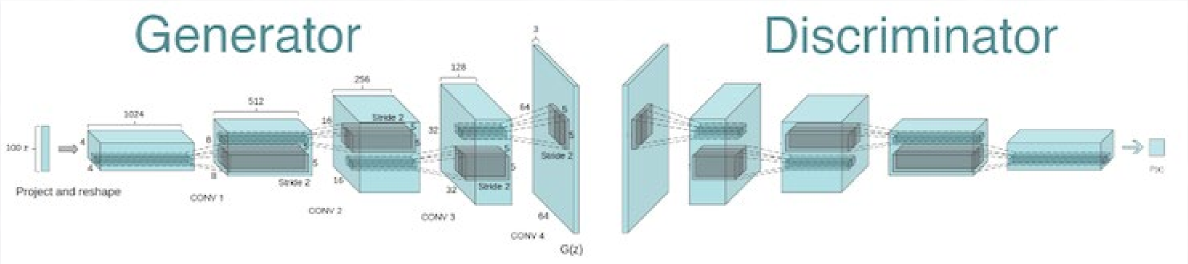
• Note that in practice, using a transposed convolution can lead to the presence of **checkerboard artifacts**; to alleviate this, practitioners commonly apply a two-step process instead: (i) bilinear up-sample, followed by (ii) convolution.

• For a comprehensive treatment of these topics, see: https://arxiv.org/abs/1603.07285

GAN: Training

• In general, training the discriminator amounts to a supervised learning problem: we create a training set of (randomly inserted) real observations from the dataset interspersed with outputs produced by the generator (label 1 for true image, 0 for fakes). Recall that binary cross-entropy loss is defined:

$$L(y,p) = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log(p_i) + \left(1-y_i\right)\log(1-p_i)\right)$$



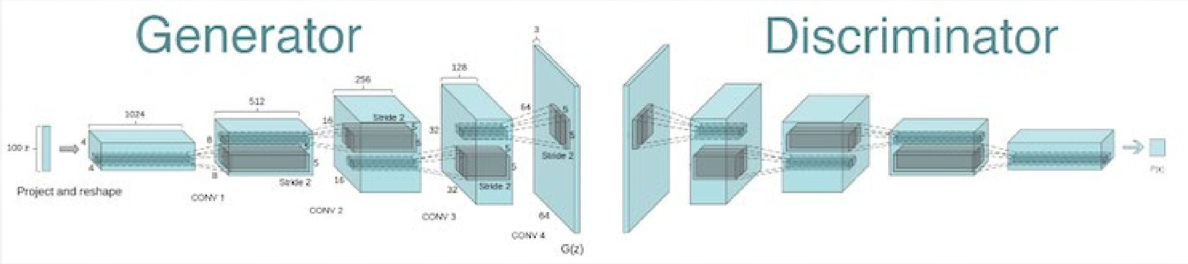
GAN: Training

• In general, training the discriminator amounts to a supervised learning problem: we create a training set of (randomly inserted) real observations from the dataset interspersed with outputs produced by the generator (label 1 for true image, 0 for fakes). Recall that binary cross-entropy loss is defined:

$$L(y,p) = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right)$$

• To train the GAN discriminator D, we calculate the loss when comparing predictions for real images $p_i = D(x_i)$ to the response $y_i = 1$ and predictions for generated images $p_i = D(G(z_i))$ to the response $y_i = 0$. Therefore, for the GAN discriminator, minimizing the loss function can be written as follows:

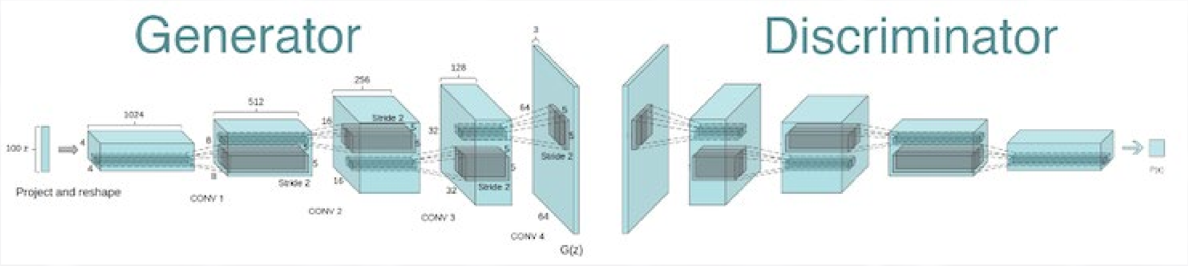
$$\boxed{\min_D -\left(E_{x\sim p_X}\left[\log D(x)\right] + E_{z\sim p_Z}\left[\log\left(1-D(G(z))\right)\right]\right)}$$



GAN: Training

• Training the generator is considerably more difficult, as we don’t readily have access to a training set that tells us the true image that a particular point in the latent space should be mapped to, for instance.

• To train the generator, we connect it to the discriminator by feeding the output from the generator into the discriminator so that the output from the combined model is the probability that a generated image is *real* (according to the discriminator).



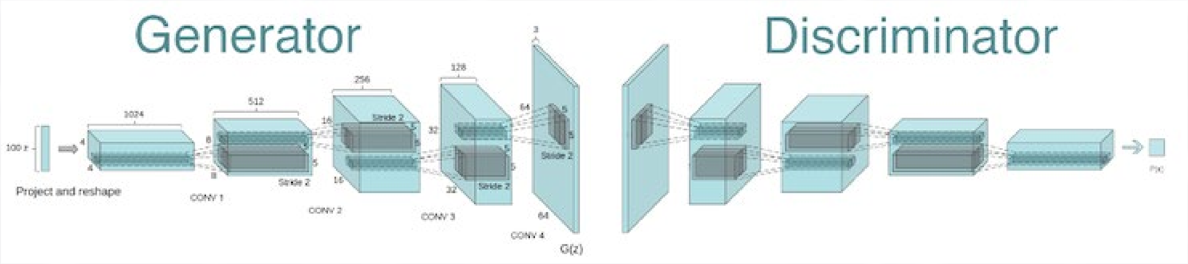
GAN: Training

• Training the generator is considerably more difficult, as we don't readily have access to a training set that tells us the true image that a particular point in the latent space should be mapped to, for instance.

• To train the generator, we connect it to the discriminator by feeding the output from the generator into the discriminator so that the output from the combined model is the probability that a generated image is *real* (according to the discriminator).

• We can train the combined model by creating training batches consisting of randomly generated latent vectors as input and a response which is set to 1, since we want to train the generator to produce images that the discriminator thinks are real. The loss is just binary cross-entropy loss between the output from the discriminator and the response vector of 1.

• Importantly, **we freeze the weights of the discriminator while we are training the combined model, so that only the generator's weights are updated**.

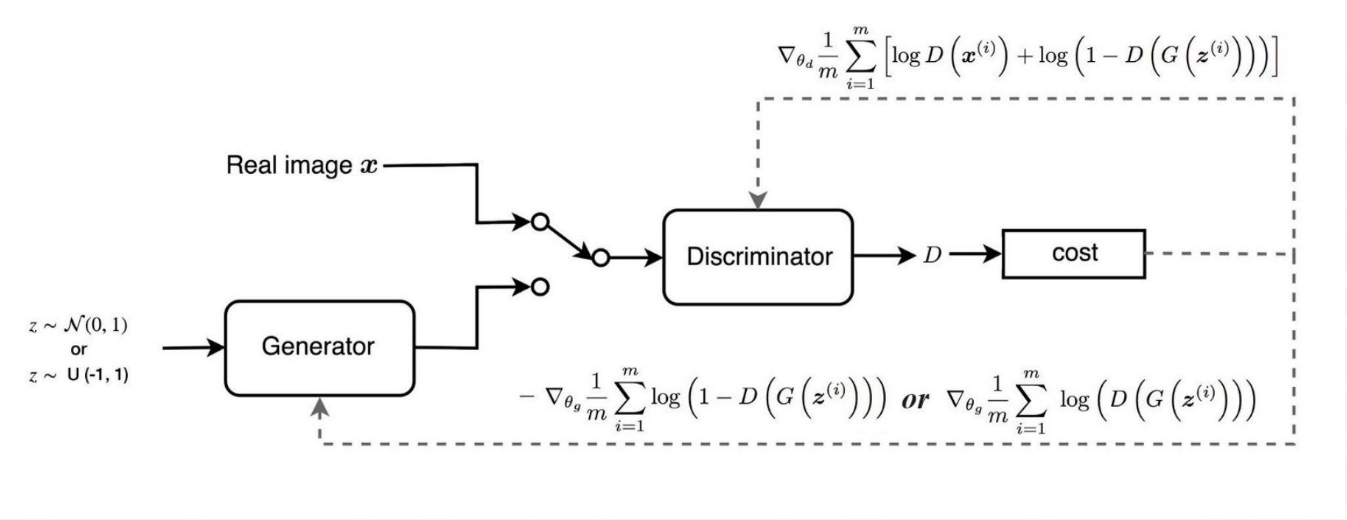


GAN: Training

$$\boxed{\min_D -\left(E_{x\sim p_X}\left[\log D(x)\right]+E_{z\sim p_Z}\left[\log\left(1-D(G(z))\right)\right]\right)}$$

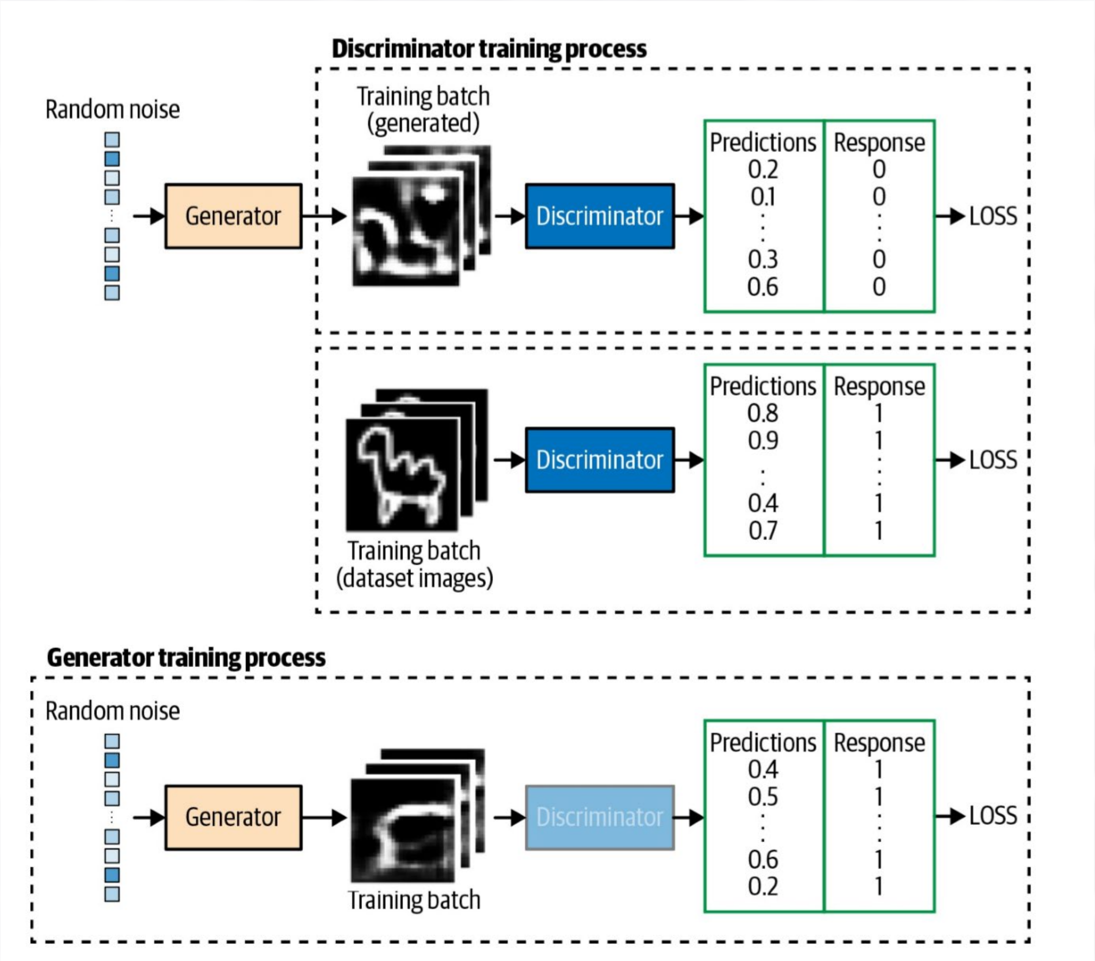
- To train the GAN generator G, we calculate the loss when comparing predictions for the generated images $p_i = D(G(z_i))$ to the response $y_i = 1$. Therefore for the GAN generator, the minimizing loss function can be written as follows:

$$\boxed{\min_G -\left(E_{z\sim p_Z}\left[\log\left(D(G(z))\right)\right]\right)}$$



GAN: Training

$$\min_D -\left(E_{x\sim p_X}\left[\log D(x)\right]+E_{z\sim p_Z}\left[\log\left(1-D(G(z))\right)\right]\right)$$

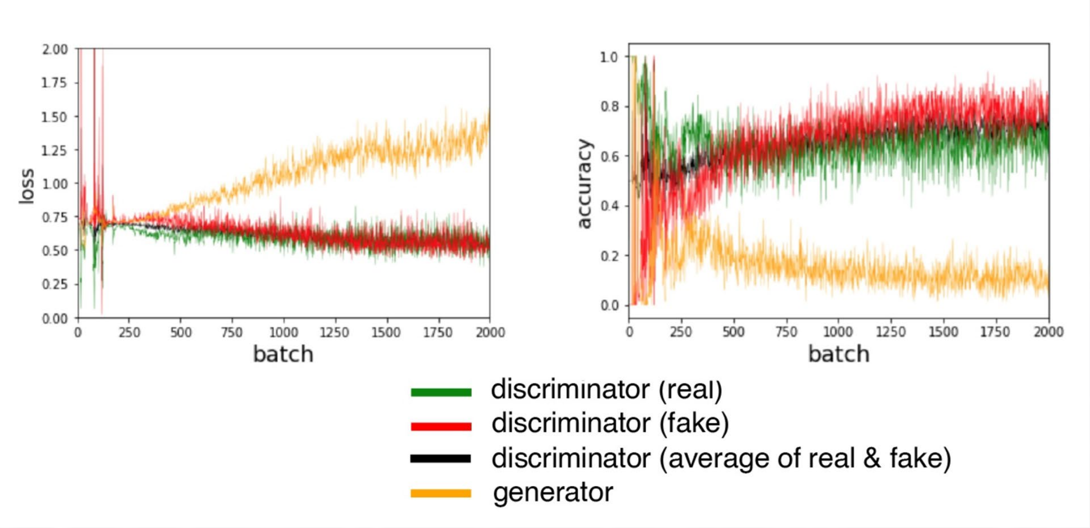


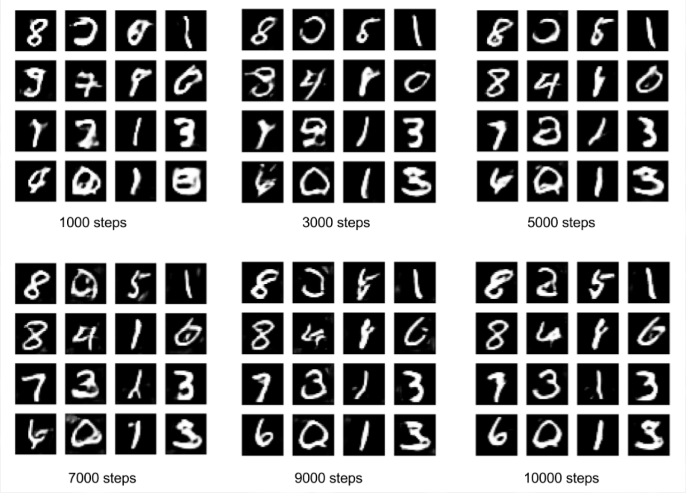
$$\min_G -\left(E_{z\sim p_Z}\left[\log\left(D(G(z))\right)\right]\right)$$

GAN: Training

• GAN training is equivalent to a **zero-sum non-cooperative game**. From a game theory context, the GAN model converges when the discriminator and the generator reach a **Nash equilibrium**.

• If trained properly (which commonly requires the use of several "tricks" which we mention next), the discriminator and generator will converge to an equilibrium that allows the generator to learn meaningful information from the discriminator and the quality of the images will improve.





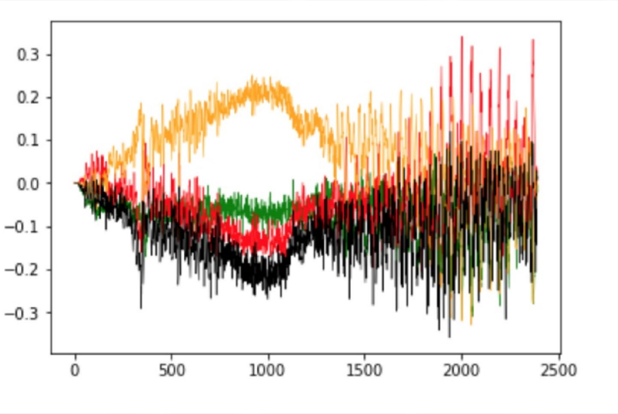
GAN: Challenges

• GANs are <u>notoriously difficult to train</u>, for several reasons:

• **Mode Collapse**: Mode collapse occurs when the generator finds a small number of samples that fool the discriminator and therefore isn't able to produce any examples other than this limited set.

• This can occur, say if we train the generator over several batches without updating the discriminator in between. In this situation, the generator would be inclined to find a singly observation that always fools the discriminator (the mode).

• **Oscillating Loss**: The losses of the discriminator and generator oscillate wildly. GANs are trained successfully when we observe a loss stabilization (shown in the previous slide); unfortunately, oscillating loss is common to vanilla GAN approaches.

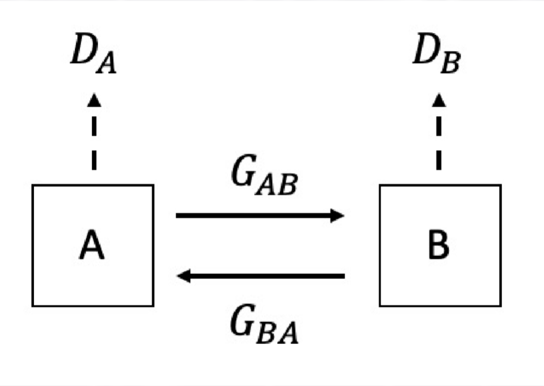


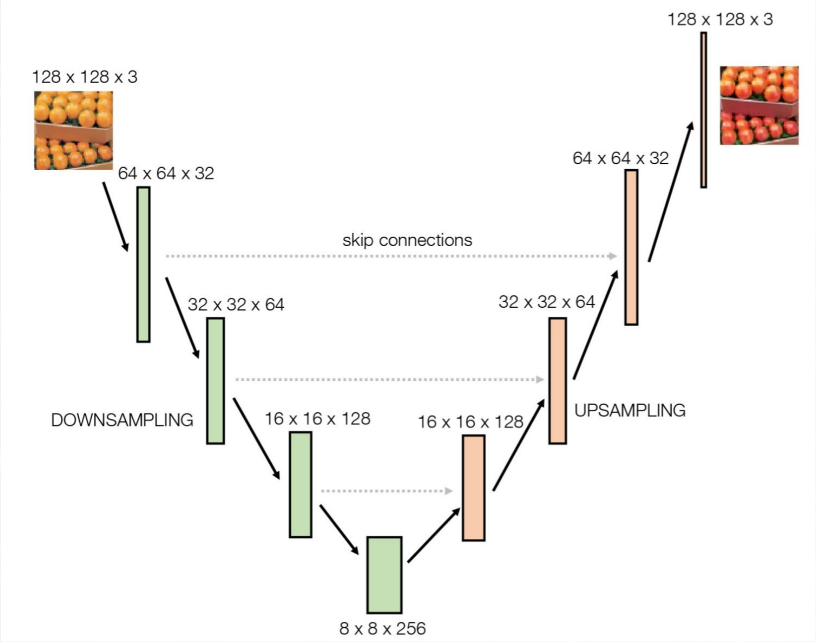
GAN: CycleGAN

- For the image translation task, CycleGAN trains without using paired examples.

- CycleGAN is composed for (4) models: two generators and two discriminators. The first generator G_{AB} converts images from domain A to domain B; whereas the second generator G_{BA}, converts images from domain B to domain A.

- The authors employ a **U-Net architecture** (shown on the right) for the generator models.





GAN: CycleGAN

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Jun-Yan Zhu* Taesung Park* Phillip Isola Alexei A. Efros
Berkeley AI Research (BAIR) laboratory, UC Berkeley

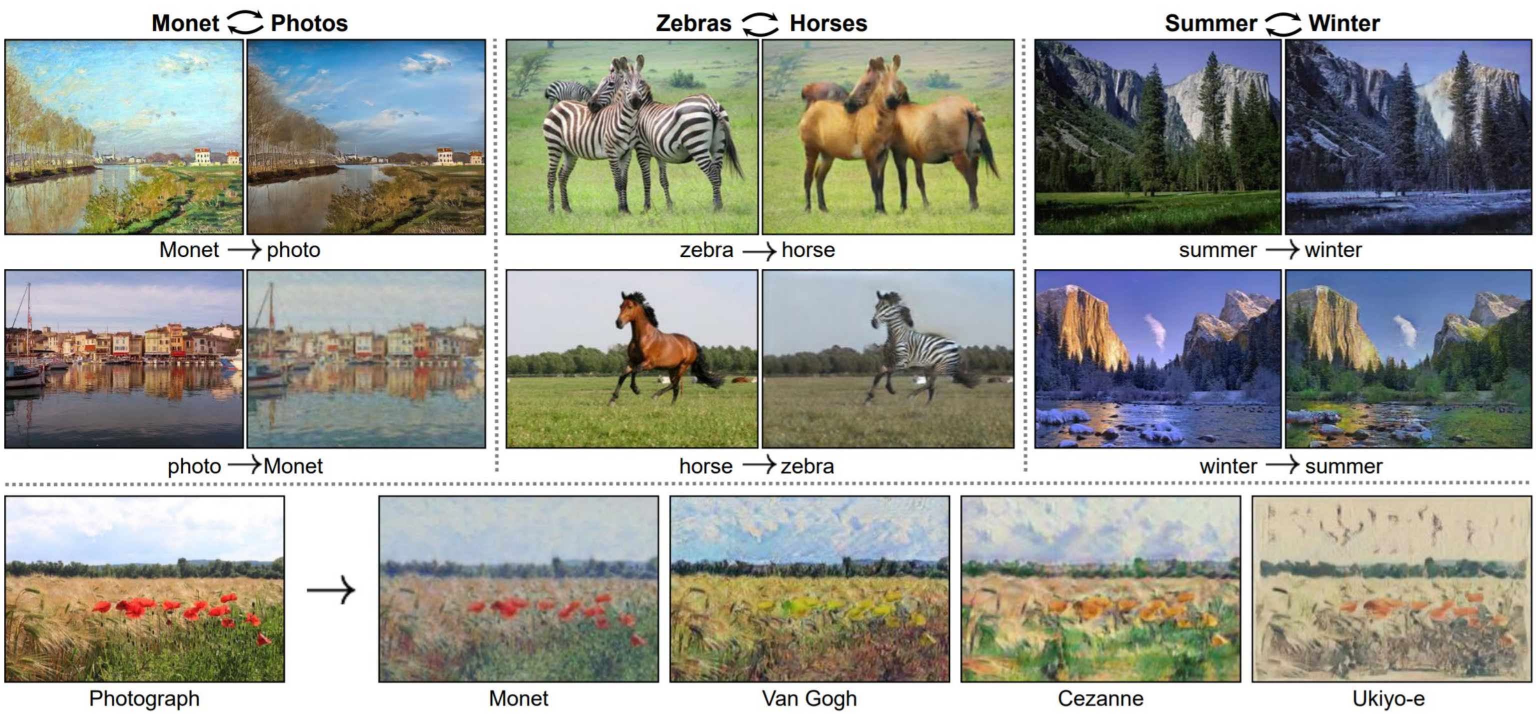
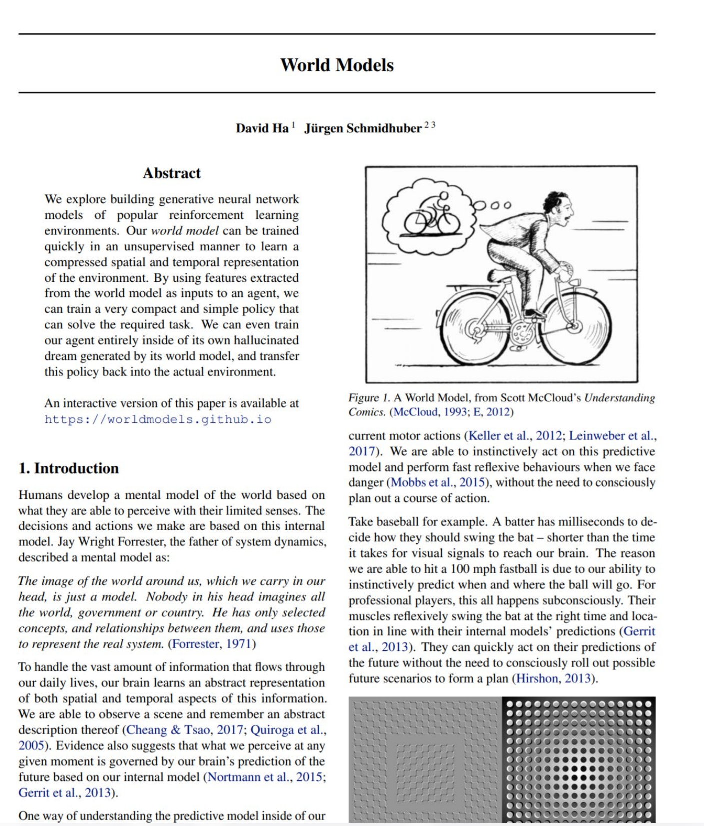


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image

"World Models" GAN

- Ha and Schmidhuber (NeurIPS, 2018) presented "World Models", a paradigm for training RL agents using a VAE, whereby an agent is trained:

"**entirely insides of its own hallucinated dream generated by its world model, and transfer this policy back into the actual environment.**"



World Models

David Ha [1] Jürgen Schmidhuber [2 3]

Abstract

We explore building generative neural network models of popular reinforcement learning environments. Our *world model* can be trained quickly in an unsupervised manner to learn a compressed spatial and temporal representation of the environment. By using features extracted from the world model as inputs to an agent, we can train a very compact and simple policy that can solve the required task. We can even train our agent entirely inside of its own hallucinated dream generated by its world model, and transfer this policy back into the actual environment.

An interactive version of this paper is available at `https://worldmodels.github.io`

1. Introduction

Humans develop a mental model of the world based on what they are able to perceive with their limited senses. The decisions and actions we make are based on this internal model. Jay Wright Forrester, the father of system dynamics, described a mental model as:

The image of the world around us, which we carry in our head, is just a model. Nobody in his head imagines all the world, government or country. He has only selected concepts, and relationships between them, and uses those to represent the real system. (Forrester, 1971)

To handle the vast amount of information that flows through our daily lives, our brain learns an abstract representation of both spatial and temporal aspects of this information. We are able to observe a scene and remember an abstract description thereof (Cheang & Tsao, 2017; Quiroga et al., 2005). Evidence also suggests that what we perceive at any given moment is governed by our brain's prediction of the future based on our internal model (Nortmann et al., 2015; Gerrit et al., 2013).

One way of understanding the predictive model inside of our

Figure 1. A World Model, from Scott McCloud's *Understanding Comics.* (McCloud, 1993; E, 2012)

current motor actions (Keller et al., 2012; Leinweber et al., 2017). We are able to instinctively act on this predictive model and perform fast reflexive behaviours when we face danger (Mobbs et al., 2015), without the need to consciously plan out a course of action.

Take baseball for example. A batter has milliseconds to decide how they should swing the bat – shorter than the time it takes for visual signals to reach our brain. The reason we are able to hit a 100 mph fastball is due to our ability to instinctively predict when and where the ball will go. For professional players, this all happens subconsciously. Their muscles reflexively swing the bat at the right time and location in line with their internal models' predictions (Gerrit et al., 2013). They can quickly act on their predictions of the future without the need to consciously roll out possible future scenarios to form a plan (Hirshon, 2013).

https://worldmodels.github.io/

"World Models" GAN

The pipeline consists of (3) fundamental components:

(1) The **Vision Model (V)**, A VAE that encodes high-dimensional observations into a low-dimensional latent vector.

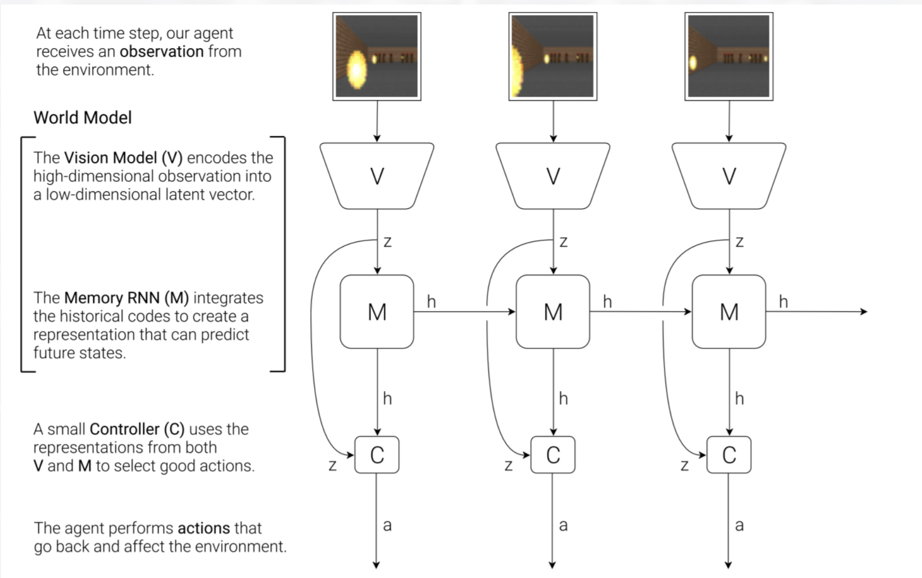
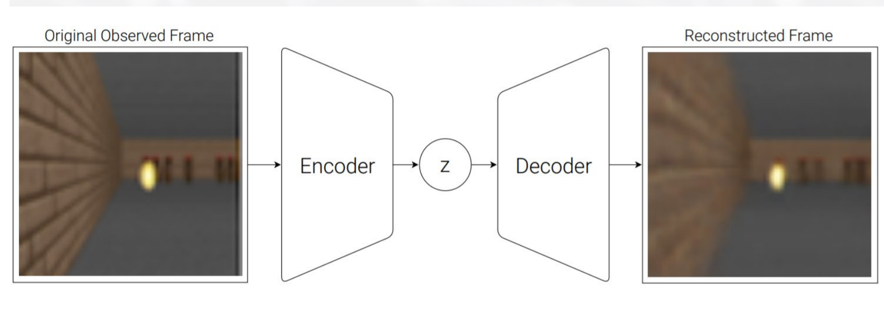


Figure 4. Our agent consists of three components that work closely together: **Vision (V)**, **Memory (M)**, and **Controller (C)**

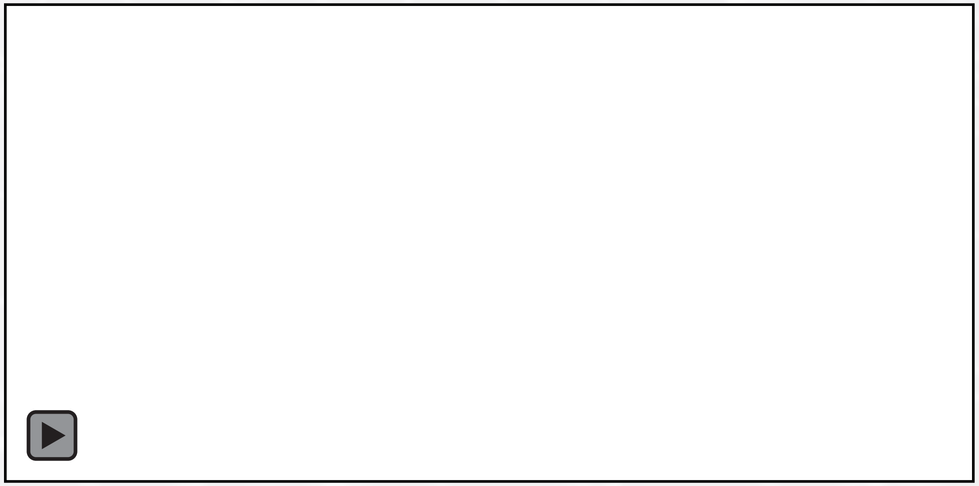


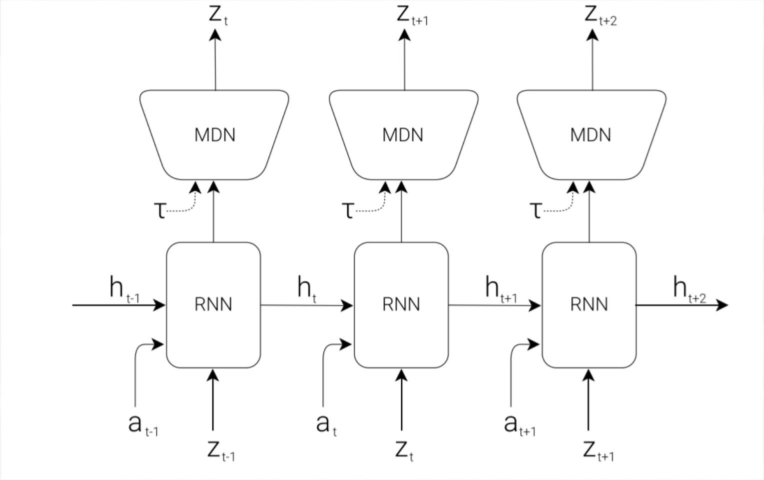
“World Models” GAN

The pipeline consists of (3) fundamental components:

(2) A **Memory RNN** (**M**): this unit approximates $p(z_t)$ using a GMM; the RNN is trained to output the probability distribution of the next latent vector z_{t+1} given the current and past information available to it -- specifically predict: $p(z_{t+1}|at, zt, ht)$

Technically, M uses an MDN (mixture density network), which has been used previously for “sequence generation” (e.g. handwriting).

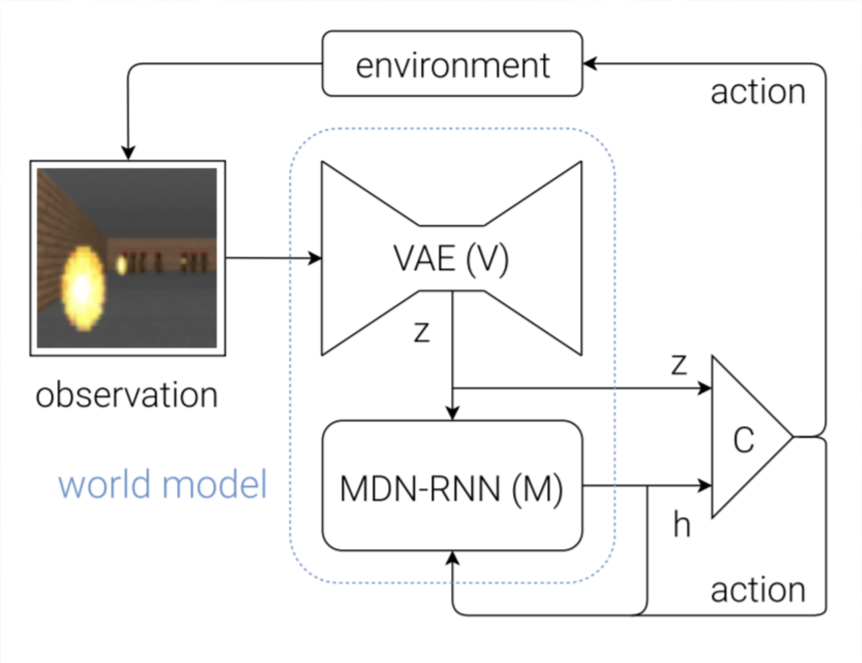




"World Models" GAN

The pipeline consists of (3) fundamental components:

(3) A **controller** (**C**) (a <u>simple</u>) RL agent that determined the course of actions to take in order to maximize the expected cumulative reward of the agent during a rollout of the environment.

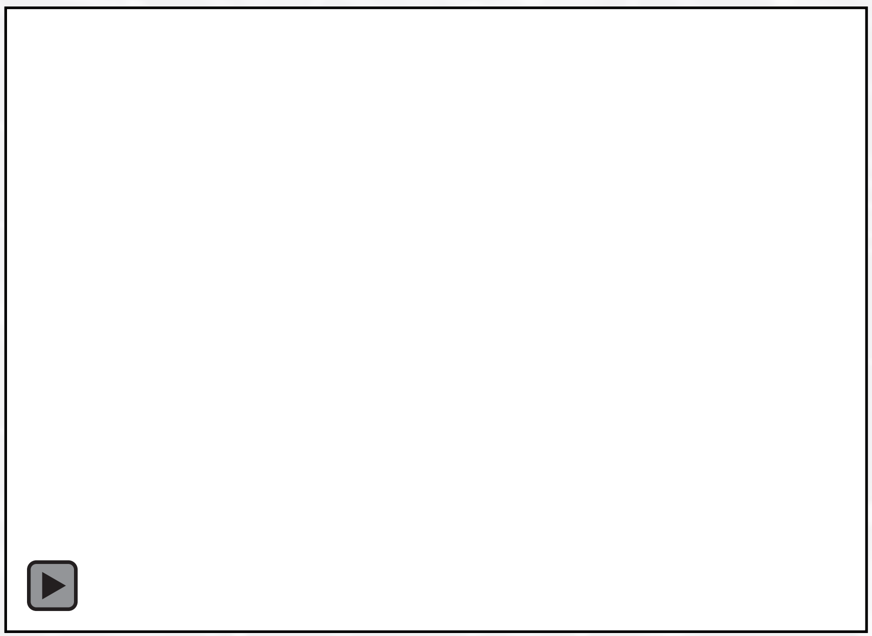


MODEL	PARAMETER COUNT
VAE	4,446,915
MDN-RNN	1,678,785
CONTROLLER	1,088

“World Models” GAN

Training with simulated dreams!

• Because the model can predict the future (!), the authors can use it to generate hypothetical racing scenarios on its own. They produce the probability distribution of given the current states, and sample a zt+1 in place of a real observation. The controller acts in the hallucinated environment generated by M.



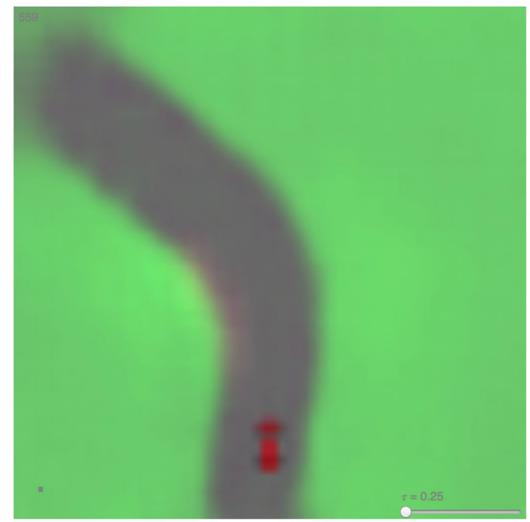
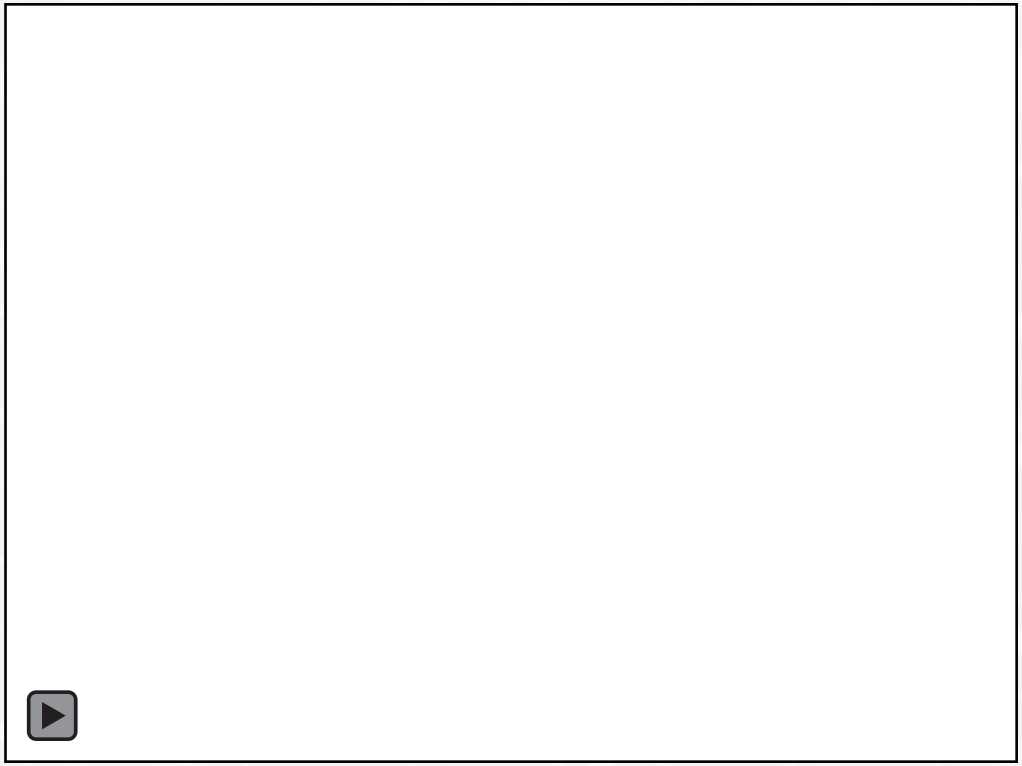


Figure 13. Our agent driving inside of its own dream world. Here, we deploy our trained policy into a fake environment generated by the MDN-RNN, and rendered using the VAE’s decoder. In the demo, one can override the agent’s actions as well as adjust τ to control the uncertainty of the environment generated by M.

“World Models” GAN

VizDoom from World Models.

Figure 15. Our agent discovers a policy to avoid hallucinated fire-balls. In the online version of this article, the reader can interact with the environment inside this demo.



Fin