[course site]

#DLUPC

Deep Generative Models II

Santiago Pascual de la Puente
santi.pascual@upc.edu

PhD Candidate
Universitat Politecnica de Catalunya
Technical University of Catalonia

TALP UPC

Outline

- Introduction
- Taxonomy
- PixelCNN & Wavenet
- Variational Auto-Encoders (VAEs)
- Generative Adversarial Networks (GANs)
- Conclusions

Recap from previous lecture…

What is a generative model?

Key Idea: our model cares about what distribution generated the input data points, and we want to mimic it with our probabilistic model. **Our learned model should be able to make up new samples from the distribution, not just copy and paste existing samples!**

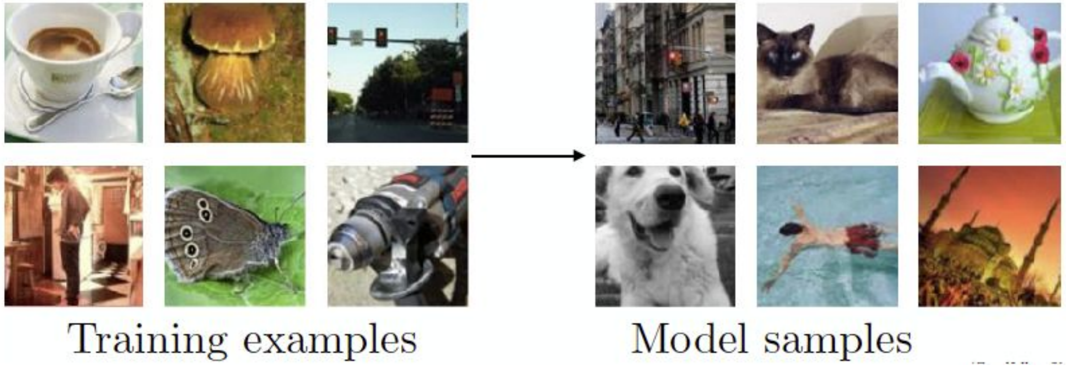


Figure from NIPS 2016 Tutorial: Generative Adversarial Networks (I. Goodfellow)

Taxonomy

Model the probability density function:

- Explicitly
  - With tractable density → PixelRNN, PixelCNN and Wavenet
  - With approximate density → Variational Auto-Encoders
- Implicitly
  - Generative Adversarial Networks

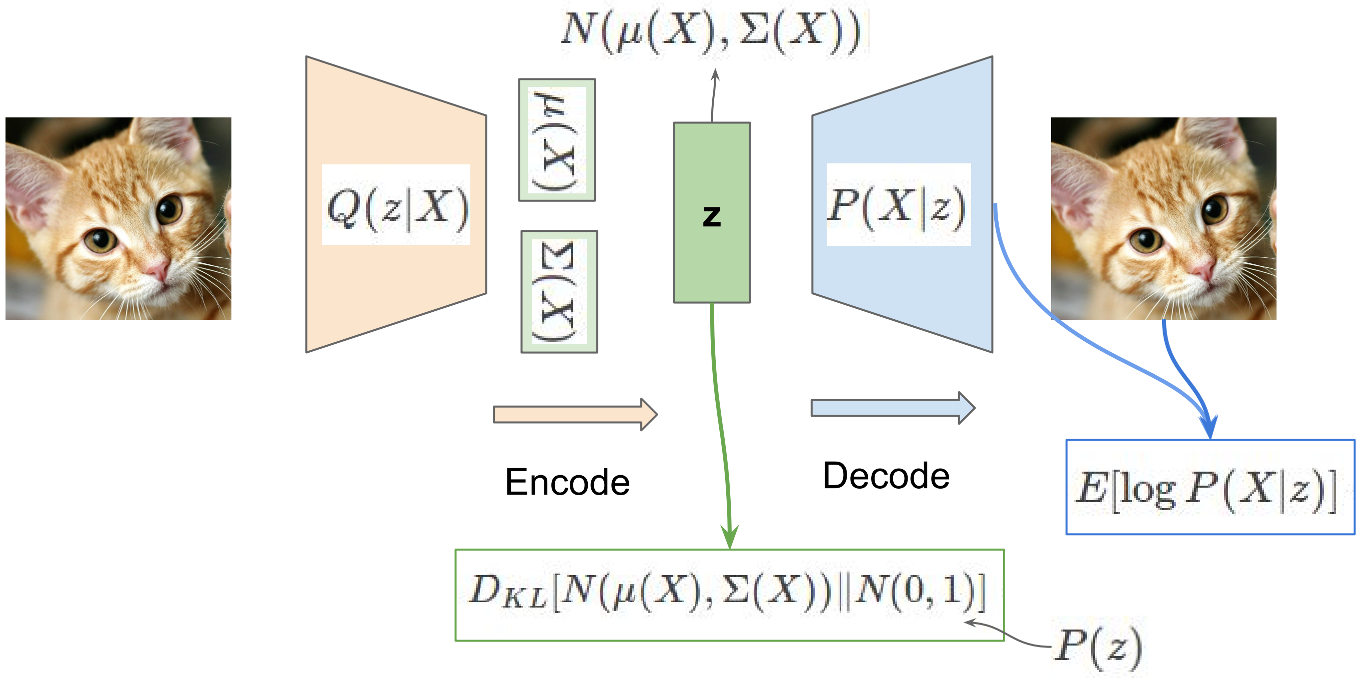
PixelCNN: Factorizing the joint distribution

- Model **explicitly** the **joint probability distribution** of data streams **x** as a product of element-wise conditional distributions for each element x_i in the stream.
  - **Example:** An image **x** of size (n, n) is decomposed scanning pixels in raster mode (Row by row and pixel by pixel within every row)
  - Apply **probability chain rule**: x_i is the i-*th* pixel in the image.

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

Variational Auto-Encoder

We can compose our encoder - decoder setup, and place our VAE losses to regularize and reconstruct.



Generative Adversarial Networks

The GAN Epidemic

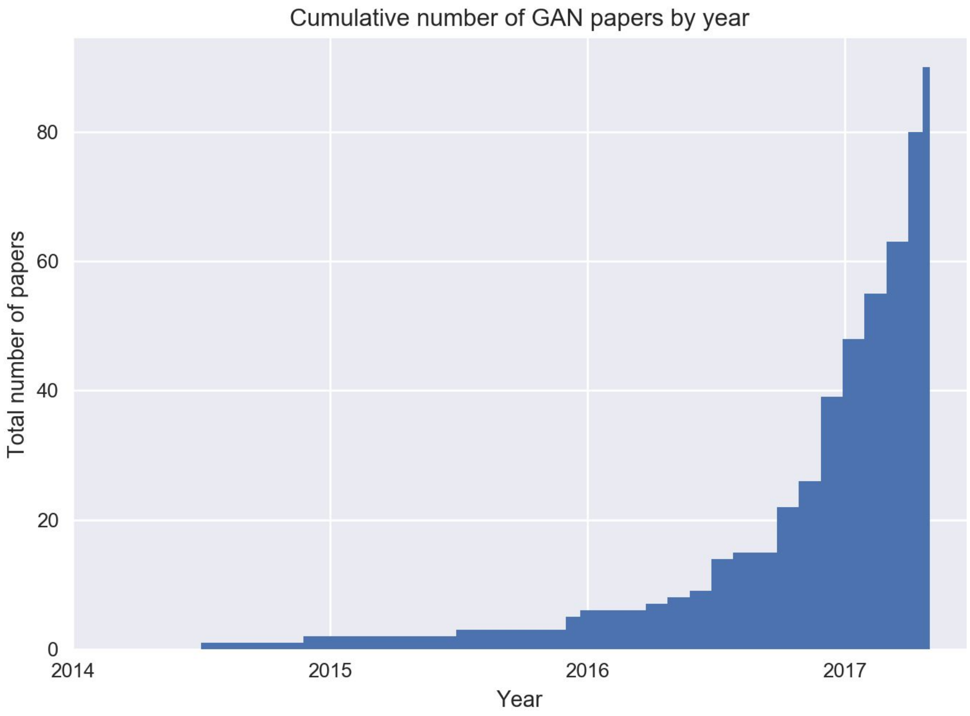


Figure credit: https://github.com/hindupuravinash/the-gan-zoo

Generative Adversarial Networks (GANs)

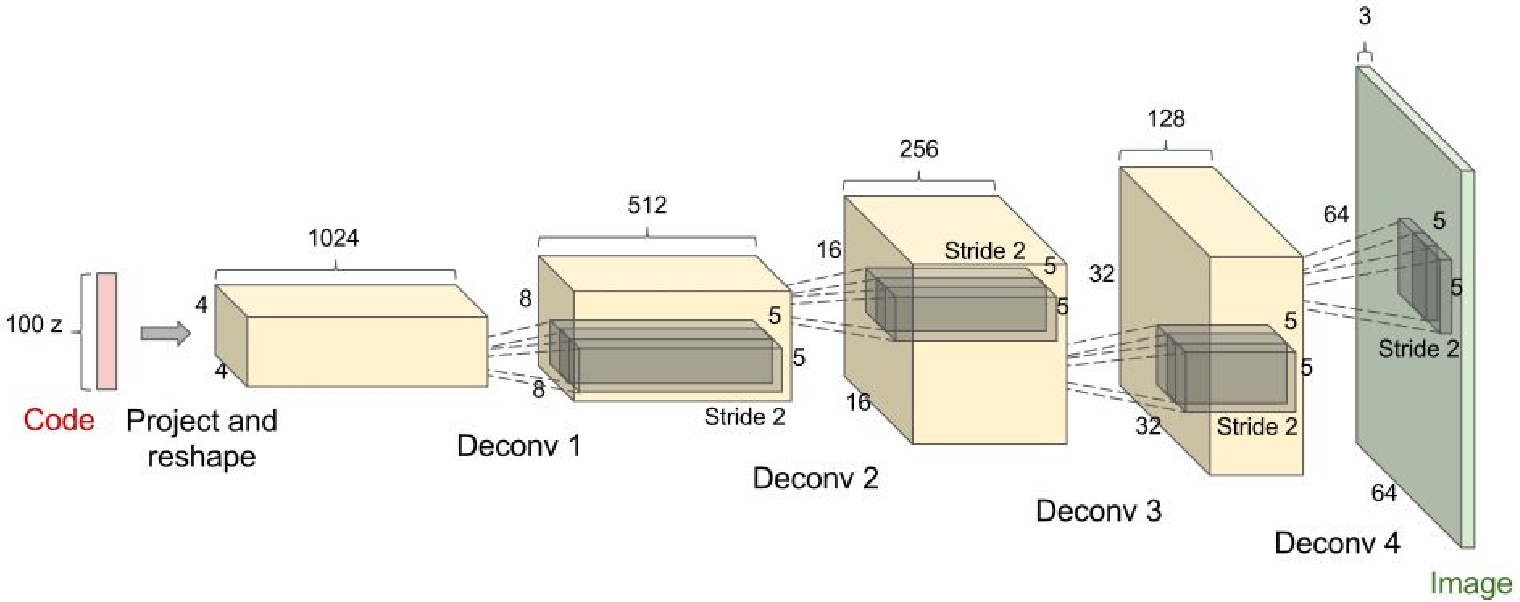
We have two modules: **Generator** (G) and **Discriminator** (D).

- They "fight" against each other during training→ **Adversarial Training**
- G mission: Fool D to missclassify.
- *D* mission: Discriminate between *G* samples and real samples.

The generator

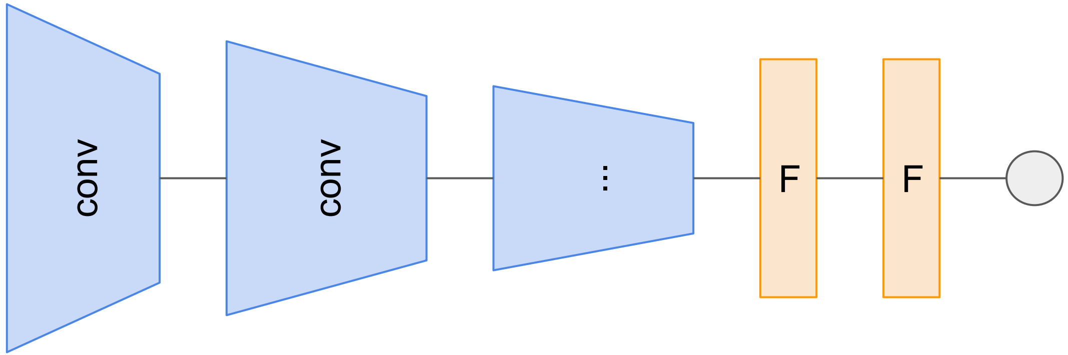
Deterministic mapping from a latent random vector to sample from Pmodel (or Pg), which should be similar to Pdata

E.g. DCGAN:



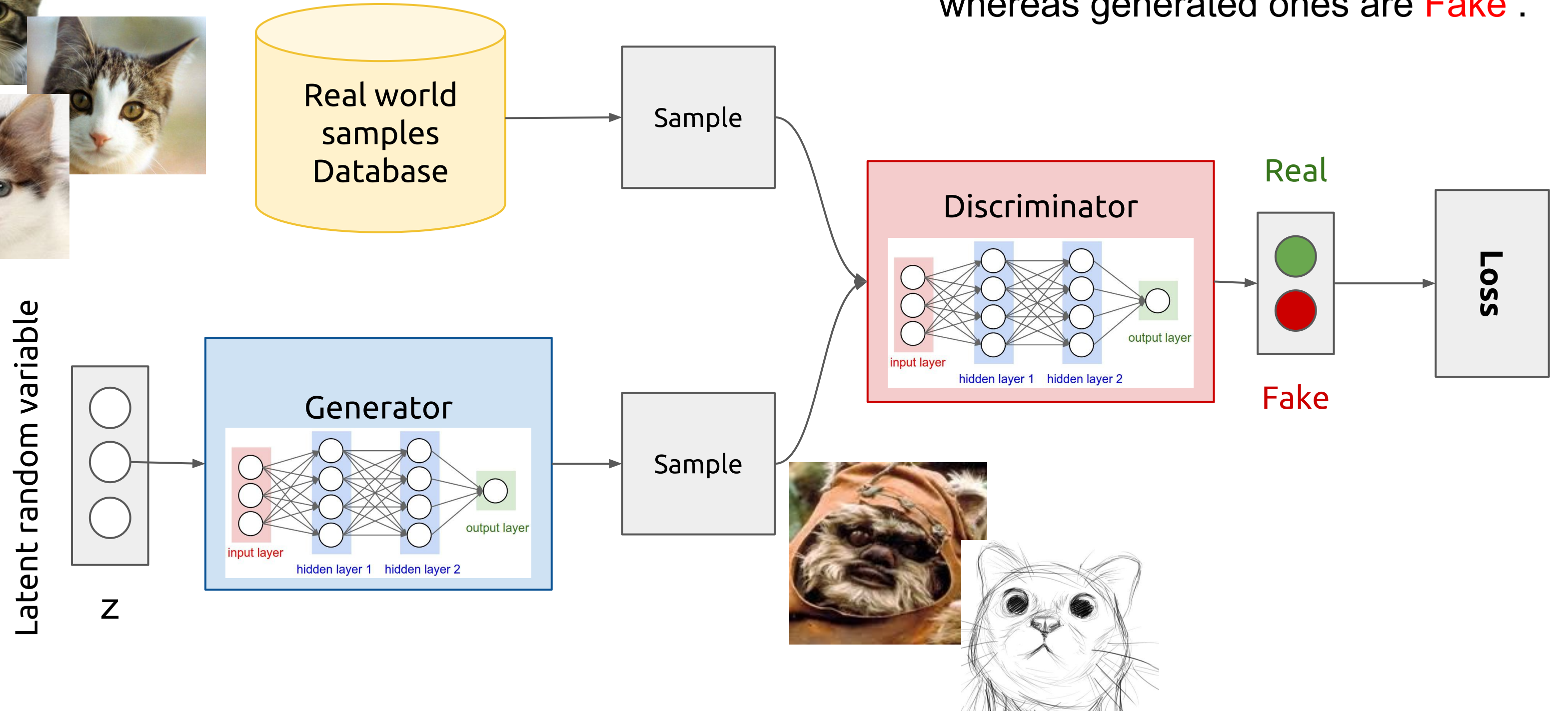
The discriminator

Parameterised function that tries to distinguish between real samples from Pdata and generated ones from Pmodel.



Adversarial Training (conceptual)

D determines database images are Real, whereas generated ones are Fake .



Adversarial Training

We have networks **G** and **D**, and **training set with pdf *Pdata***. Notation:

- ***θ(G)***, ***θ(D)*** (Parameters of model **G** and **D** respectively)
- ***x*** ~ Pdata (M-dim sample from training data pdf)
- ***z*** ~ *N(0, I)* (sample from prior pdf, e.g. N-dim normal)
- **G(z)** = ***ẍ*** ~ Pmodel (M-dim sample from G network)

D network receives **x** or **ẍ** inputs → decides whether input is real or fake. It is **optimized to learn: x is real (1), ẍ is fake (0) (binary classifier).**

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim \text{p}_\text{data}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right).$$

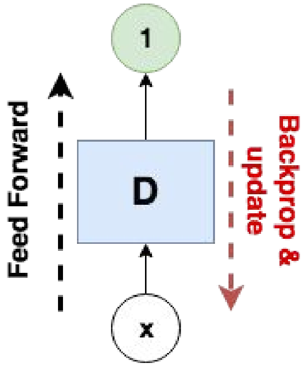
G network maps sample **z** to G(**z**) = **ẍ** → it is **optimized to maximize D mistakes.**

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$$

NIPS 2016 Tutorial: Generative Adversarial Networks. Ian Goodfellow

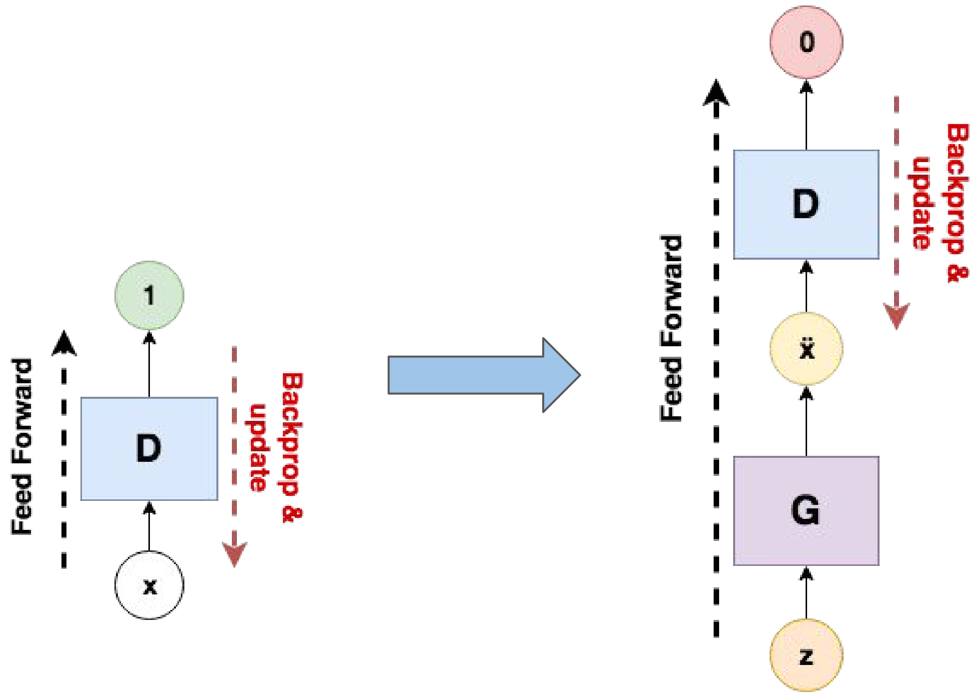
Adversarial Training (batch update) (1)

- Pick a sample ***x*** from training set
- Show ***x*** to **D** and update weights to output 1 (real)



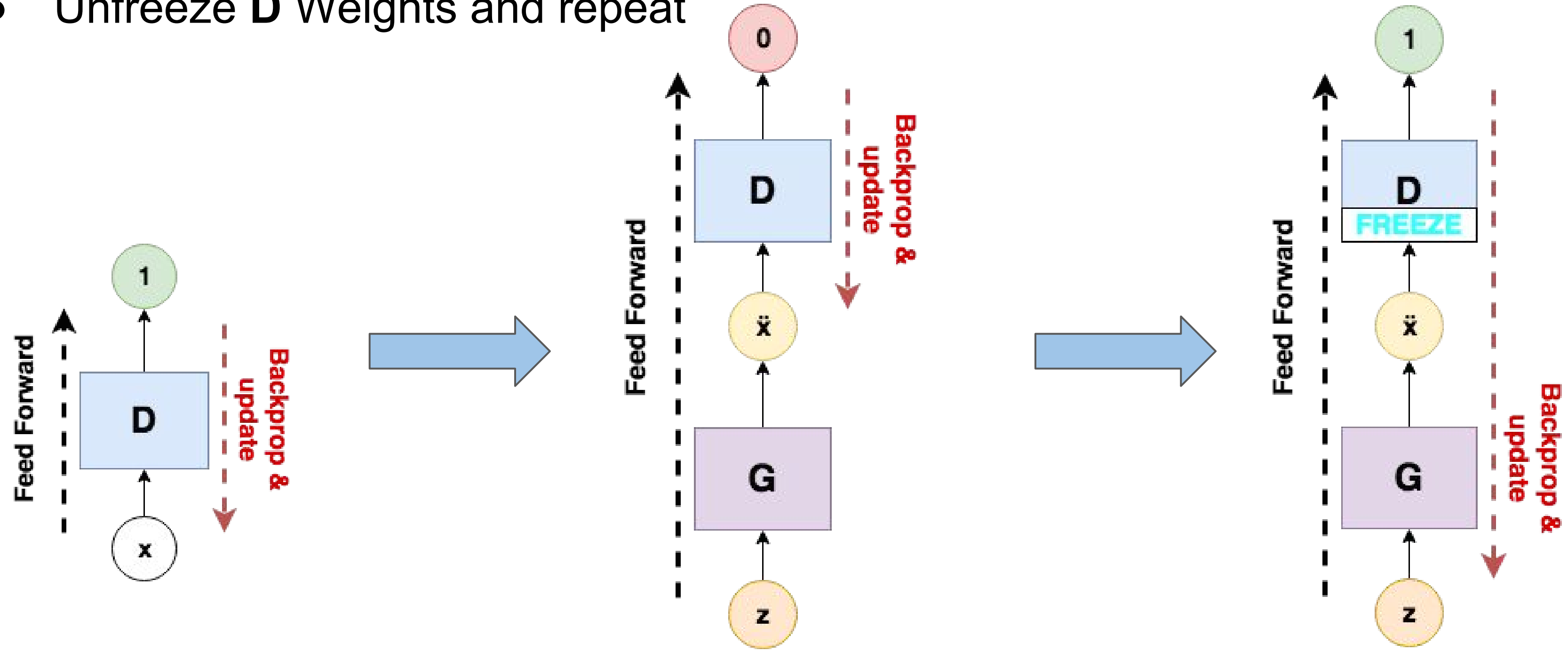
Adversarial Training (batch update) (2)

- **G** maps sample ***z*** to ***ẍ***
- show ***ẍ*** and update weights to output 0 (fake)



Adversarial Training (batch update) (3)

- Freeze **D** weights
- Update **G** weights to make **D** output 1 (just **G** weights!)
- Unfreeze **D** Weights and repeat



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

Discriminator training

for k steps **do**

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

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

end for

Generator training

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

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

Discriminator training

for k steps **do**

Q: BUT WHY K DISCRIMINATOR UPDATES?

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

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

end for

Generator training

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

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

Discriminator training

for k steps **do**

- Sample miniba[...] $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minib[...] $x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
- Update the disc[...] its stochastic gradient:

$$[\ldots]) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\Big].$$

end for

Q: BUT WHY K DISCRIMINATOR UPDATES?

A: WE WANT STRONG DISCRIMINATIVE FEATURES TO BACKPROP TO GENERATOR.

Generator training

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

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

end for

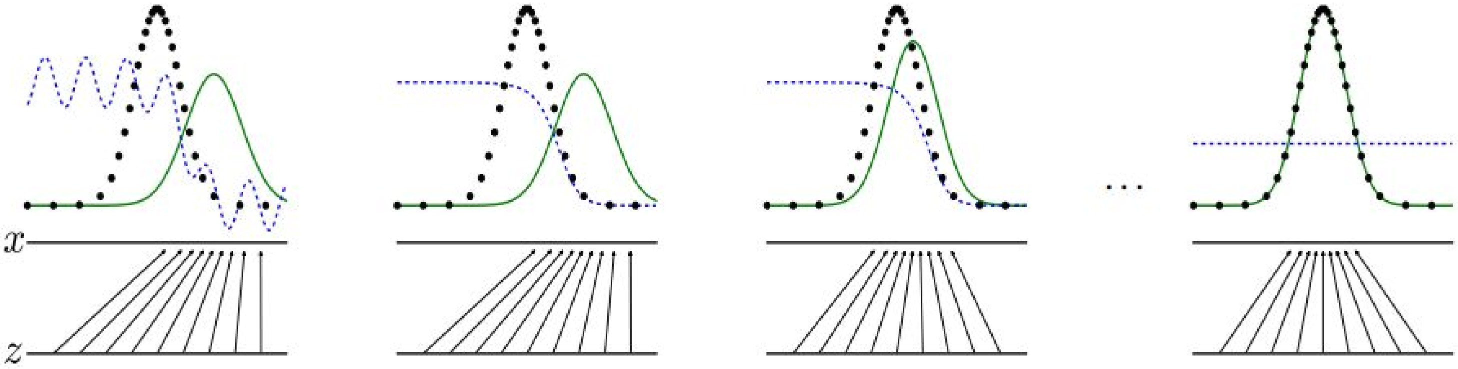
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training GANs dynamics

Iterate these two steps until convergence (which may not happen)

- Updating the discriminator should make it better at discriminating between real images and generated ones (**discriminator improves**)
- Updating the generator makes it better at fooling the current discriminator (**generator improves**)

Eventually (we hope) that the generator gets so good that it is impossible for the discriminator to tell the difference between real and generated images. Discriminator accuracy = 0.5



Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.

Key Idea: **D** is trained to detect fraud, (its parameters learn discriminative features of “what is real/fake”). As backprop goes through **D** to **G** there happens to be information leaking about the requirements for bank notes to look real. This makes **G** perform small corrections to get closer and closer to what real samples would be.

Adversarial Training Analogy: is it fake money?

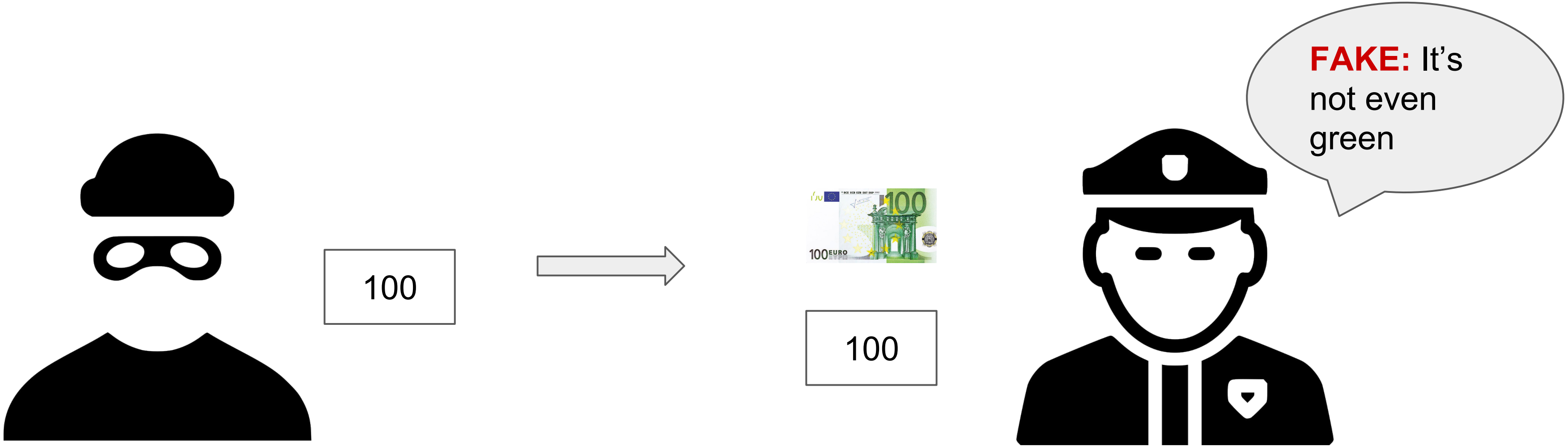
Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.

Key Idea: D is trained to detect fraud, (its parameters learn discriminative features of “what is real/fake”). As backprop goes through **D** to **G** there happens to be information leaking about the requirements for bank notes to look real. This makes **G** perform small corrections to get closer and closer to what real samples would be.

Caveat: this means GANs are not suitable for discrete tokens predictions (e.g. words) → in that discrete space there is no “small change” criteria to get to a neighbour (but can work in a word embedding space for example).

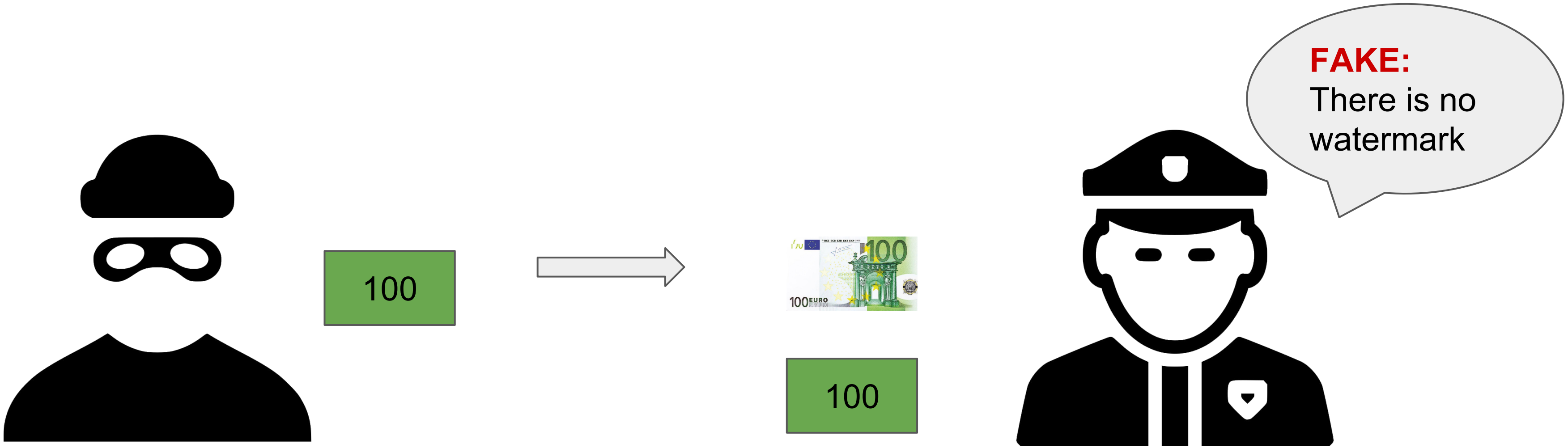
Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.



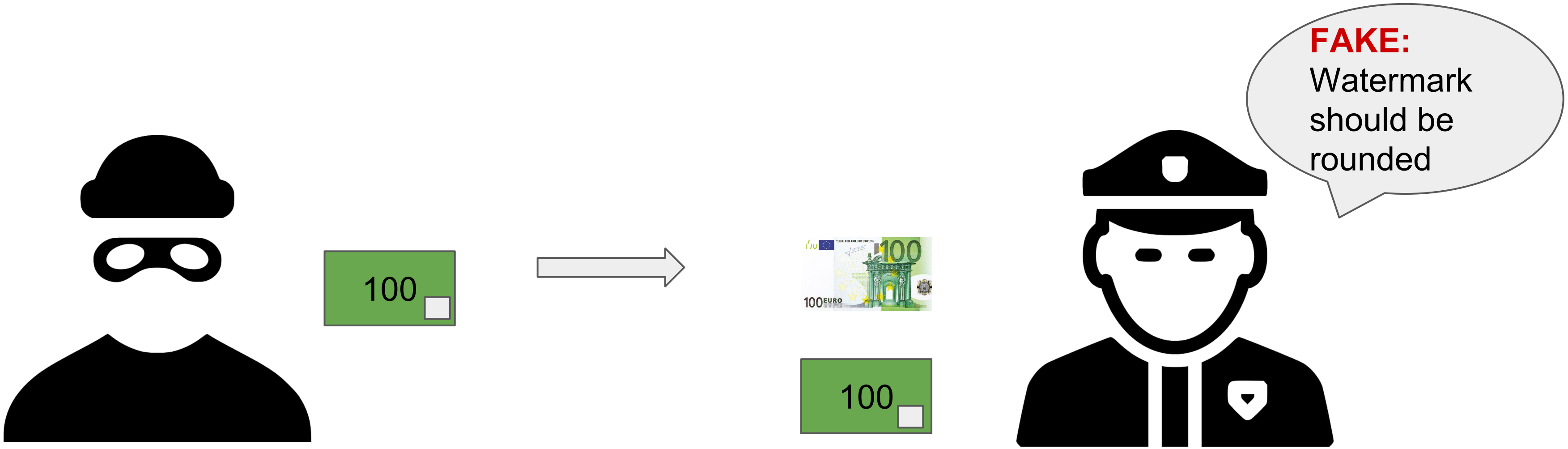
Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.



Adversarial Training Analogy: is it fake money?

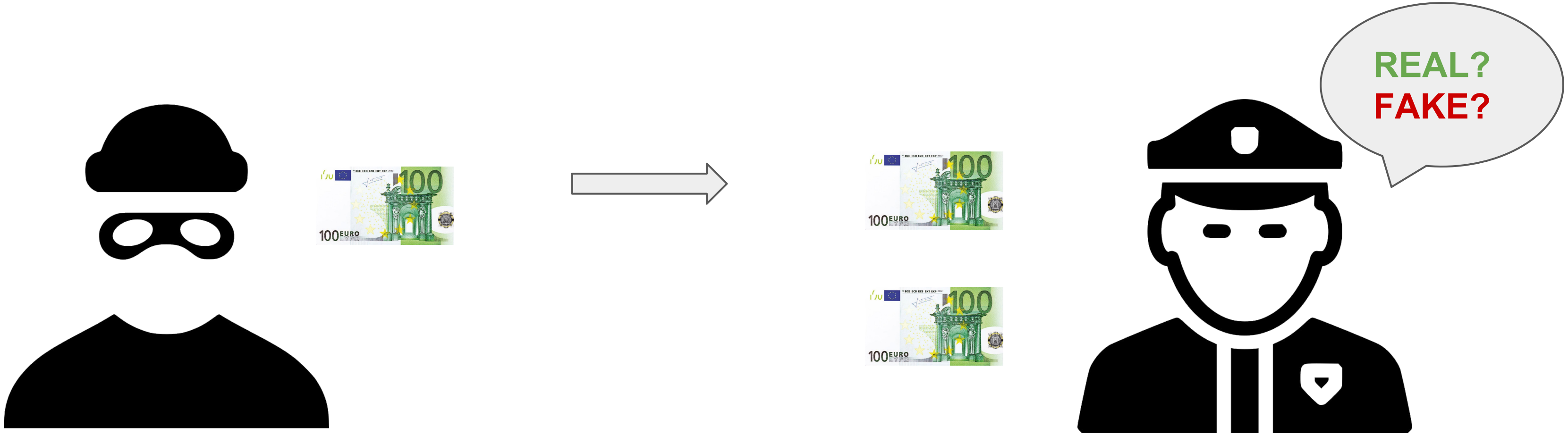
Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.



Adversarial Training Analogy: is it fake money?

Imagine we have a counterfeiter (**G**) trying to make fake money, and the police (**D**) has to detect whether money is real or fake.

After enough iterations, and if the counterfeiter is good enough (in terms of **G** network it means “has enough parameters”), the police should be confused.

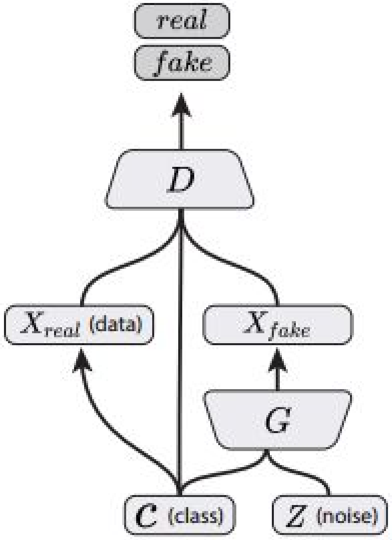


Conditional GANs

For details on ways to condition GANs:
[Ways of Conditioning Generative Adversarial Networks (Wack et al.)]

GANs can be conditioned on other info extra to **z**: text, labels, speech, etc..

z might capture random characteristics of the data (variabilities of plausible futures), whilst **c** would condition the deterministic parts !



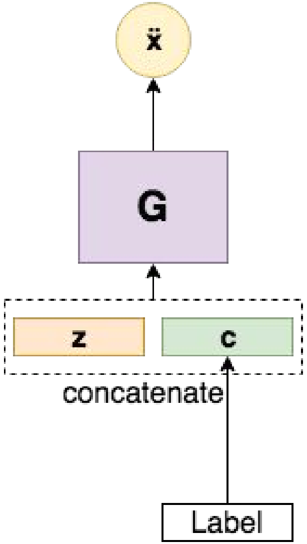
Conditional GAN
(Mirza & Osindero, 2014)

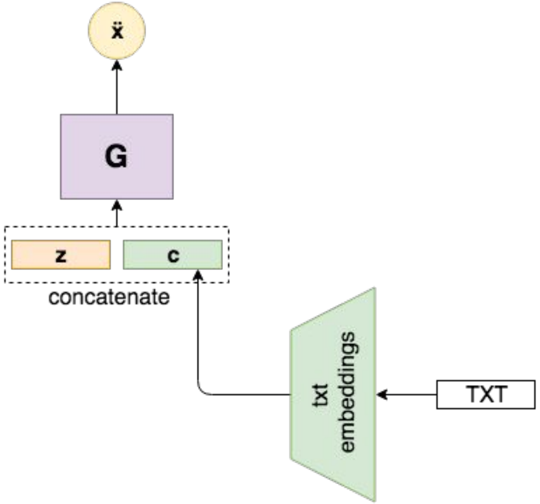
Conditional GANs

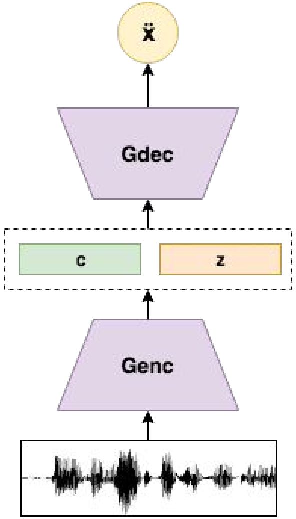
For details on ways to condition GANs:
[Ways of Conditioning Generative Adversarial Networks (Wack et al.)]

GANs can be conditioned on other info extra to **z**: text, labels, speech, etc..

z might capture random characteristics of the data (variabilities of plausible futures), whilst **c** would condition the deterministic parts !







Caveats

Where is the downside...?

GANs are tricky and hard to train! We do not want to minimize a cost function. Instead **we want both networks to reach Nash equilibria (saddle point)**.

- Formulated as a "game" between two networks
- Unstable dynamics: hard to keep generator and discriminator in balance
- Optimization can **oscillate** between solutions
- Generator can collapse

Caveats

Where is the downside...?

GANs are tricky and hard to train! We do not want to minimize a cost function. Instead **we want both networks to reach Nash equilibria (saddle point)**.

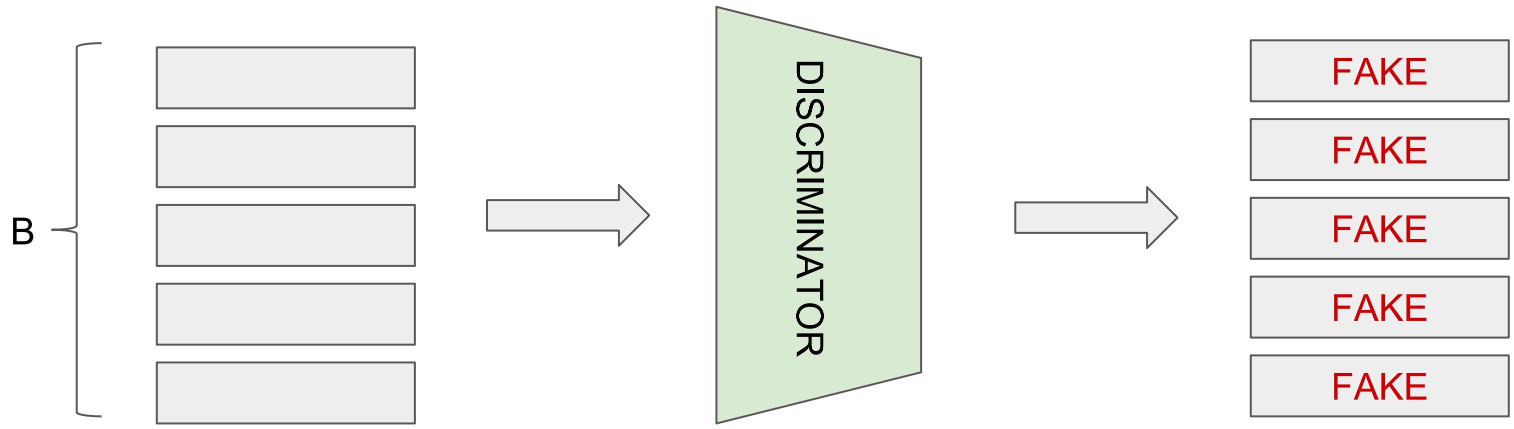
Because of extensive experience within the GAN community (with some does-not-work-frustration from time to time), you can find some tricks and tips on how to train a vanilla GAN here: https://github.com/soumith/ganhacks

[(Salimans et al. 2016)](#)

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

- When collapse is imminent, gradient of D may point in similar directions for many similar points → There is no coordination b/w D gradients as it processes each sample independently in the minibatch.

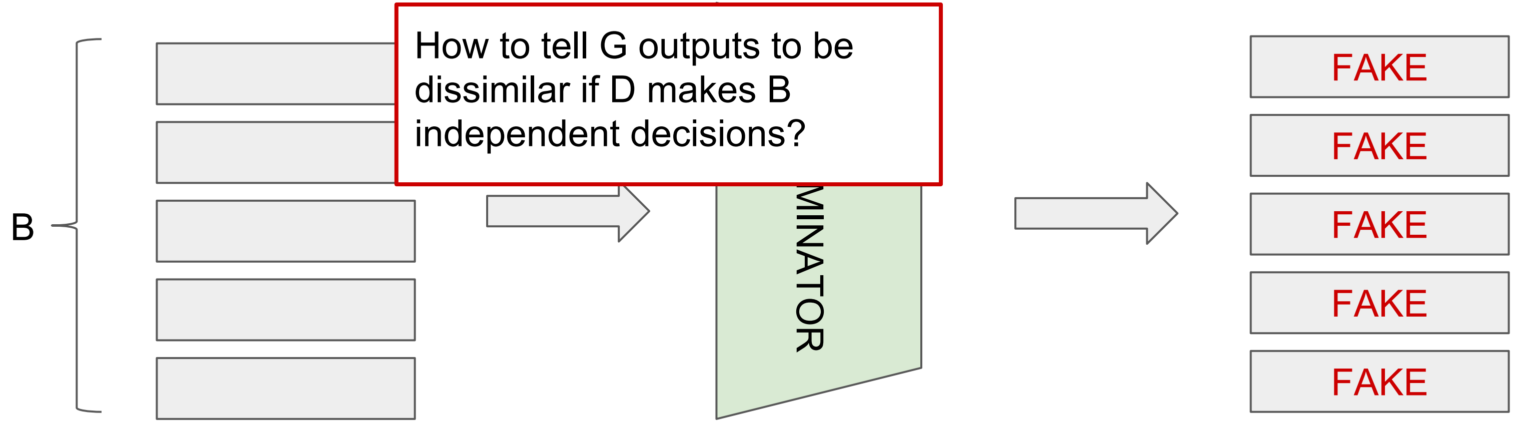


(Salimans et al. 2016)

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

- When collapse is imminent, gradient of D may point in similar directions for many similar points → There is no coordination b/w D gradients as it processes each sample independently in the minibatch.

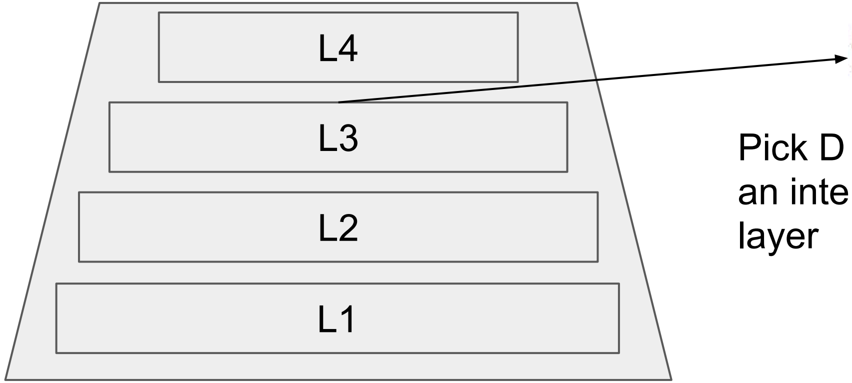


(Salimans et al. 2016)

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

- Looking at multiple examples combined could potentially help avoiding collapse of generator → model the *closeness* b/w examples in a minibatch



$$\mathbf{f}(\boldsymbol{x}_i) \in \mathbb{R}^A$$

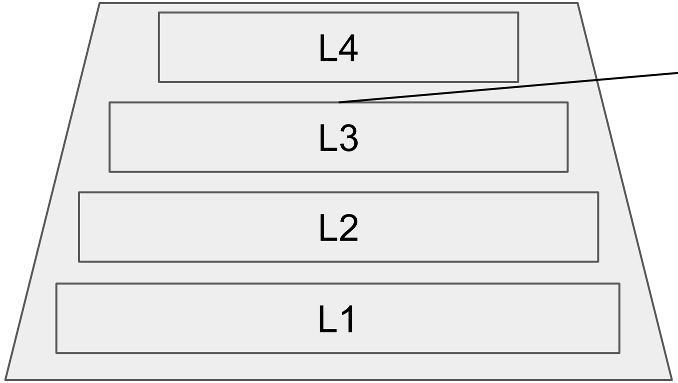
Pick D features in an intermediate layer

(Salimans et al. 2016)

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

- Looking at multiple examples combined could potentially help avoiding collapse of generator → model the *closeness* b/w examples in a minibatch



$$\mathbf{f}(\boldsymbol{x}_i) \in \mathbb{R}^A$$

Pick D features in an intermediate layer

Multiply by learnable tensor T

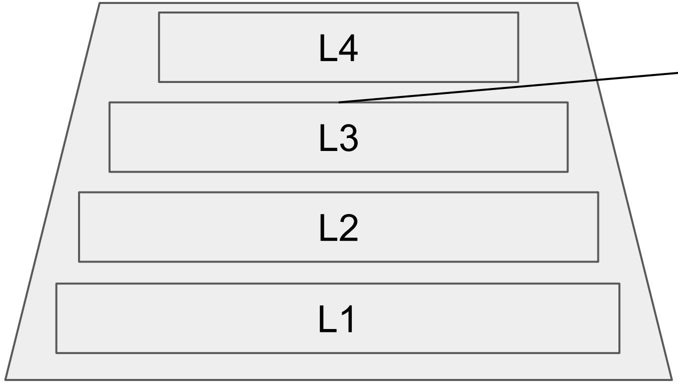
$$T \in \mathbb{R}^{A\times B\times C}$$

[(Salimans et al. 2016)]

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

- Looking at multiple examples combined could potentially help avoiding collapse of generator → model the *closeness* b/w examples in a minibatch



$\mathbf{f}(\boldsymbol{x}_i) \in \mathbb{R}^A$

Pick D features in an intermediate layer

Multiply by learnable tensor T

$T \in \mathbb{R}^{A \times B \times C}$

$M_i \in \mathbb{R}^{B \times C}$

Obtain Matrix M

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

- Looking at multiple examples combined could potentially help avoiding collapse of generator → model the *closeness* b/w examples in a minibatch

Introduce a notion of distance between rows of interaction matrix M

$$\mathbf{f}(\boldsymbol{x}_i) \in \mathbb{R}^A$$

$$T \in \mathbb{R}^{A\times B\times C}$$

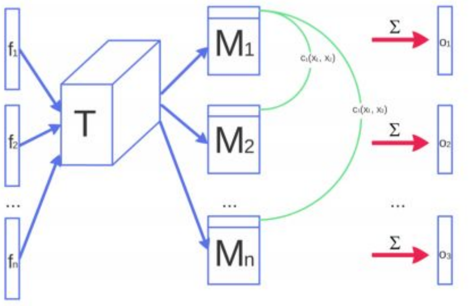
$$M_i \in \mathbb{R}^{B\times C}$$

$$c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||M_{i,b} - M_{j,b}||_{L_1}) \in \mathbb{R}$$

$$o(\boldsymbol{x}_i)_b = \sum_{j=1}^{n} c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}$$

$$o(\boldsymbol{x}_i) = \Big[o(\boldsymbol{x}_i)_1, o(\boldsymbol{x}_i)_2, \ldots, o(\boldsymbol{x}_i)_B\Big] \in \mathbb{R}^B$$

$$o(\mathbf{X}) \in \mathbb{R}^{n\times B}$$

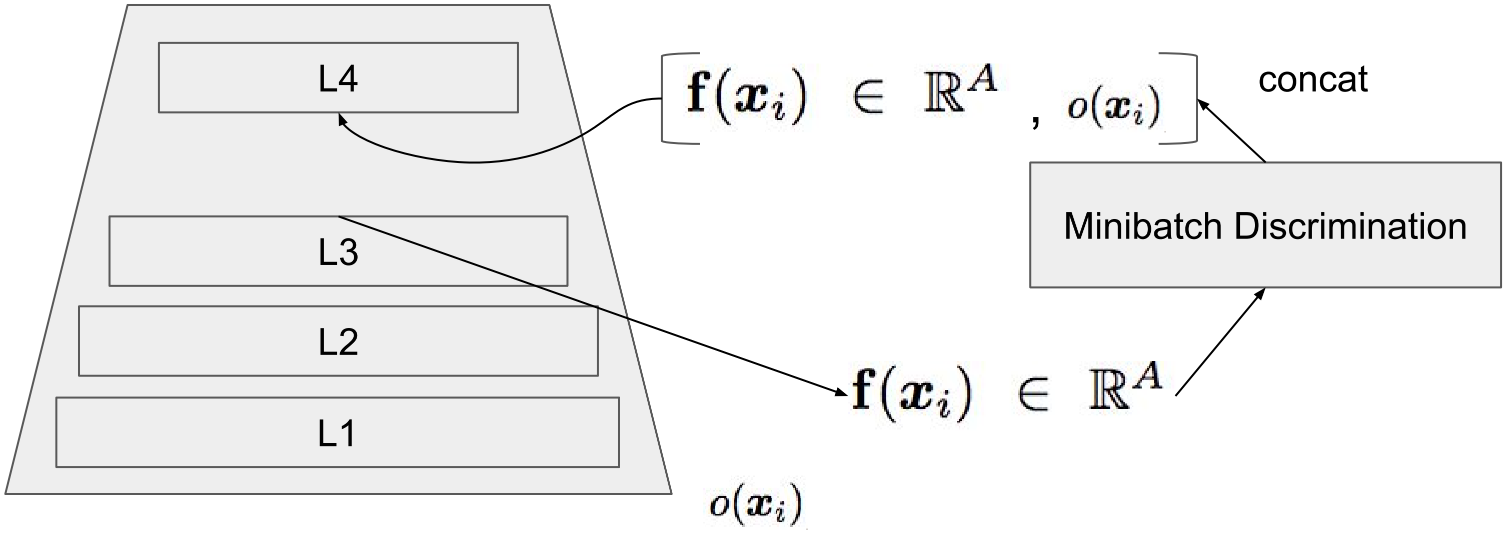


(Salimans et al. 2016)

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

- Looking at multiple examples combined could potentially help avoiding collapse of generator → model the *closeness* b/w examples in a minibatch

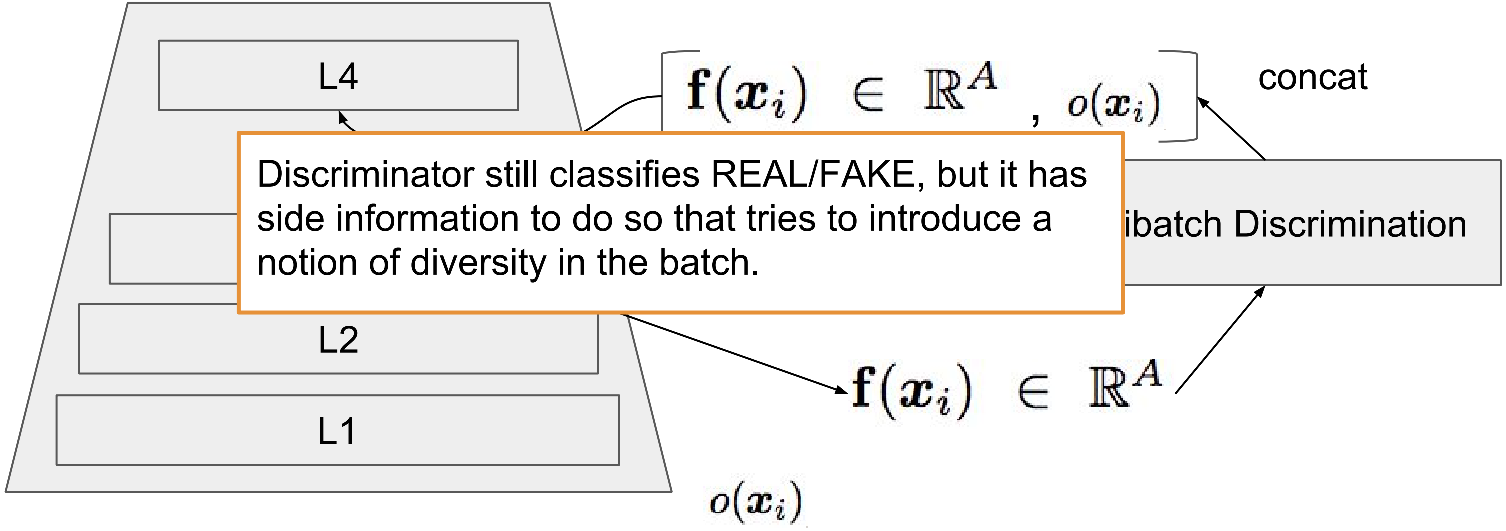


(Salimans et al. 2016)

Solving the collapse: Minibatch Discrimination

Generator collapse = parameter setting where it always emits sample sample.

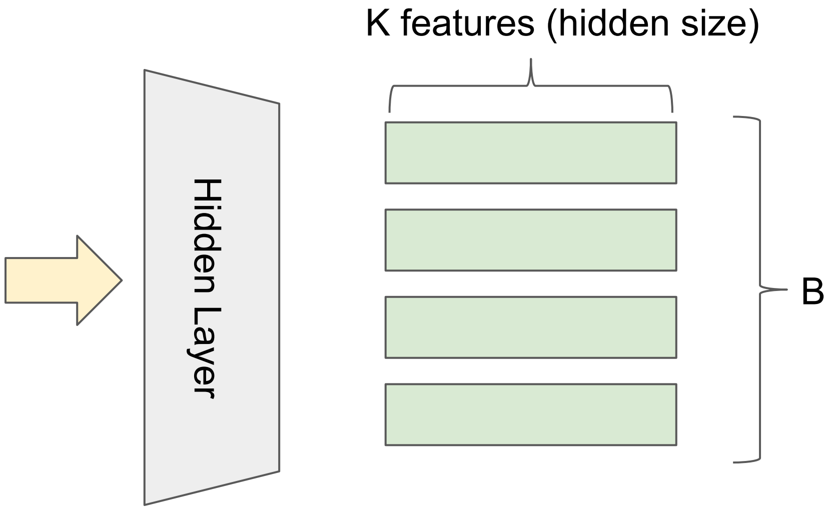
- Looking at multiple examples combined could potentially help avoiding collapse of generator → model the *closeness* b/w examples in a minibatch



Discriminator still classifies REAL/FAKE, but it has side information to do so that tries to introduce a notion of diversity in the batch.

Virtual Batch Normalization

Batch Normalization is a technique to re-standardize every layer output distribution to be N(0, I).



We normalize each k dimension with minibatch statistics

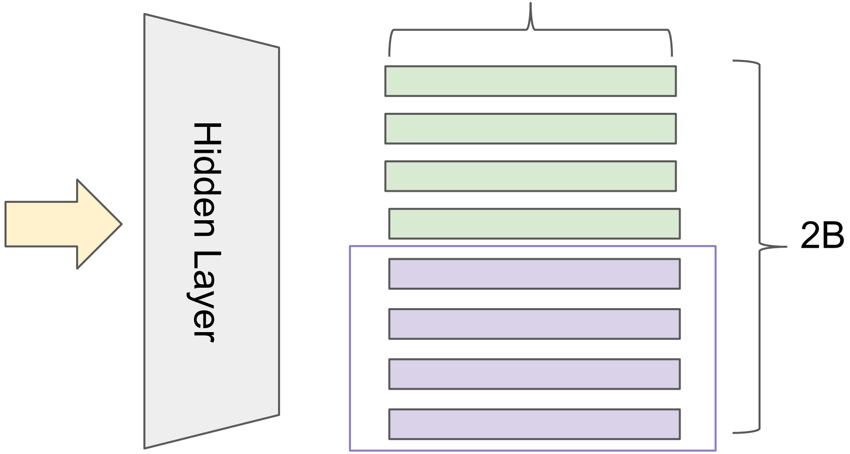
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

(Ioffe & Szegedy 2015)

Virtual Batch Normalization

Batch Normalization provokes intra-batch correlation, thus being G prone to mode collapse → we can use a ref batch to smooth the statistics of our minibatch:

K features (hidden size)



reference

We normalize each k dimension with minibatch statistics

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

(Salimans et al. 2016)

Least Squares GAN

Least Squares Generative Adversarial Networks, Mao et al. 2016

Main idea: shift to loss function that provides smooth & non-saturating gradients in D

- **Because of sigmoid saturation** in binary classification loss, G gets no info when D gets to label true examples → **vanishing gradients make G no learn**
- Least squares loss improves learning with notion of distance of *Pmodel* to *Pdata*:

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x})-1)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})))^2\right]$$

$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-1)^2\right],$$

Other GAN implementations

Other GAN implementations try to stabilize learning dynamics, imposing new divergences to be measured by D → gradients flow better and loss gets correlated with generation quality:

- Wasserstein GAN (Arjovsky et al. 2017)
- BEGAN: Boundary Equilibrium Generative Adversarial Networks (Berthelot et al. 2017)
- Improved Training of Wasserstein GANs (Gulrajani et al. 2017)

GAN Applications

So far GANs have been extensively used in computer vision tasks:

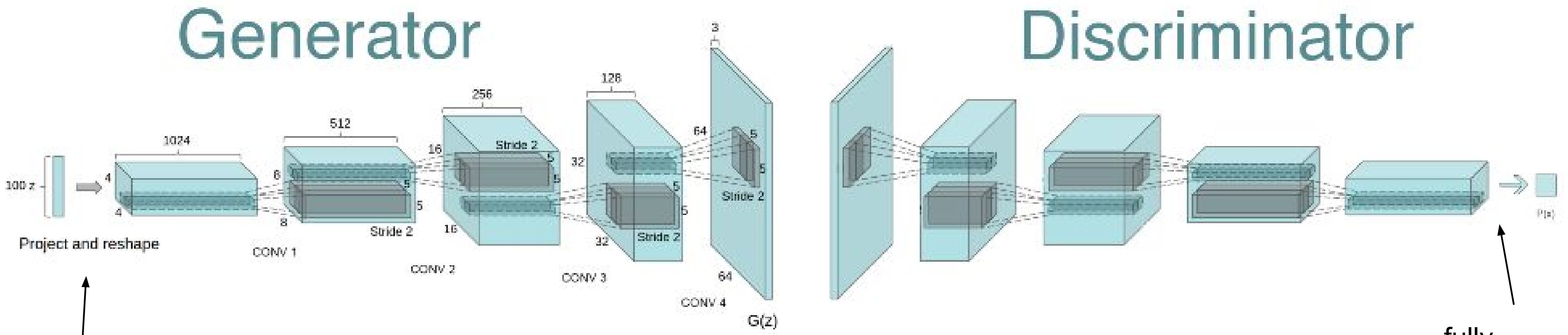
- Generating images/generating video frames.
- Unsupervised feature extraction. Representation learning.
- Manipulating images (like a photoshop advanced level).
- Image coding/Super Resolution.
- Transferring image styles.

But now they're extending to other fields, like speech!

- Speech Enhancement (Waveform)
- Unpaired Voice Conversion (Spectrum)
- Speech synthesis post-filtering (Spectrum)

Generating images/frames

Deep Conv. GAN (DCGAN) effectively generated 64x64 RGB images in a single shot.



fully connected

Strided transposed conv layers

Conv layers, no max-pool

fully connected

([Radford et al. 2015](#))

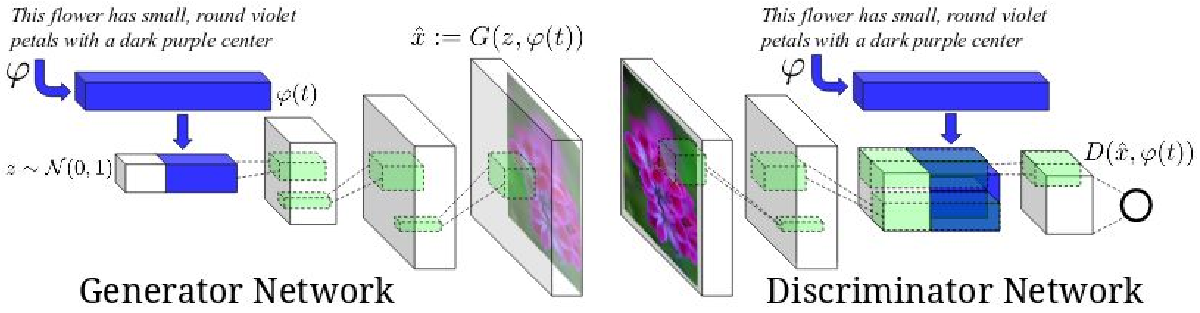
Generating images/frames

Deep Conv. GAN (DCGAN) effectively generated 64x64 RGB images in a single shot. For example bedrooms from LSUN dataset.

(Radford et al. 2015)

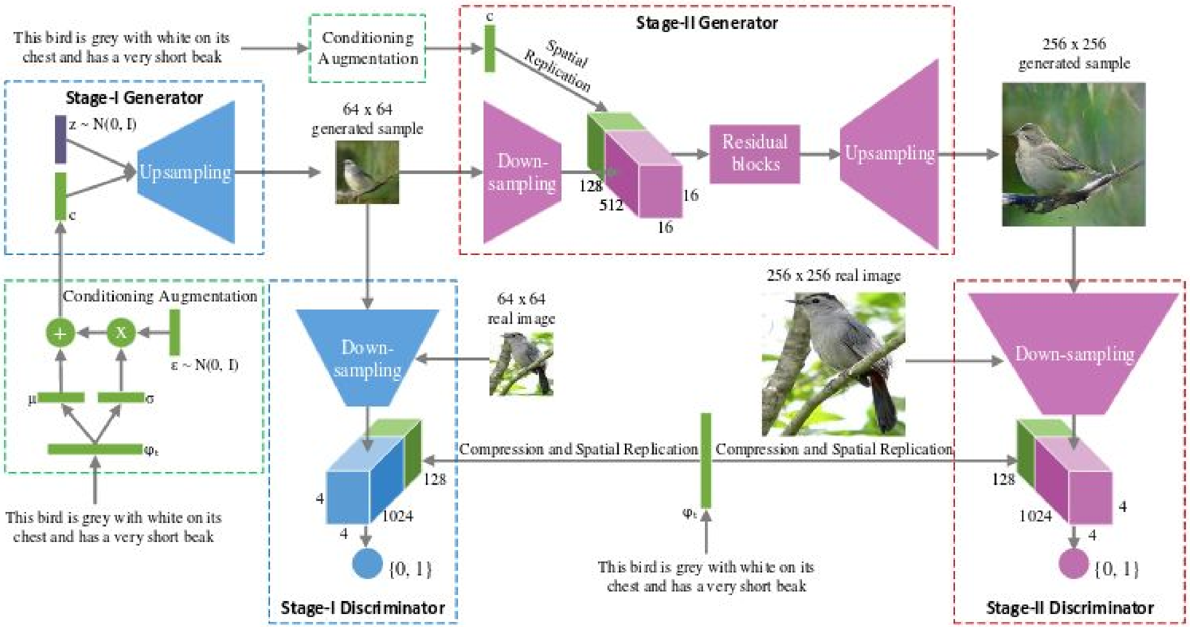
Generating images/frames conditioned on captions

Generative Adversarial Text to Image Synthesis (Reed et al. 2016b)

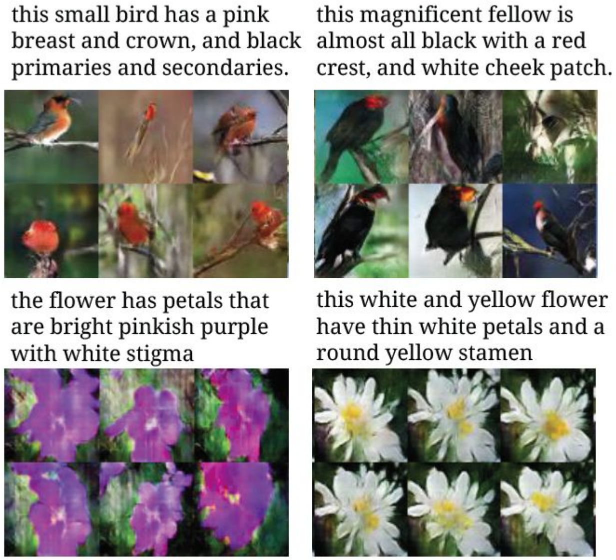


Generating images/frames conditioned on captions

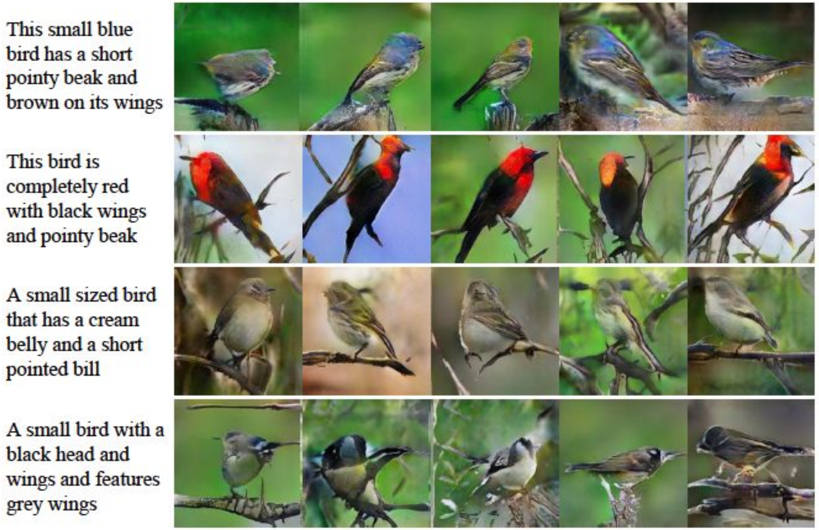
StackGAN (Zhang et al. 2016). Increased resolution to 256x256, conceptual two-stage: ‘Draft’ and ‘Fine-grained detalis’



Generating images/frames conditioned on captions



([Reed et al. 2016b])

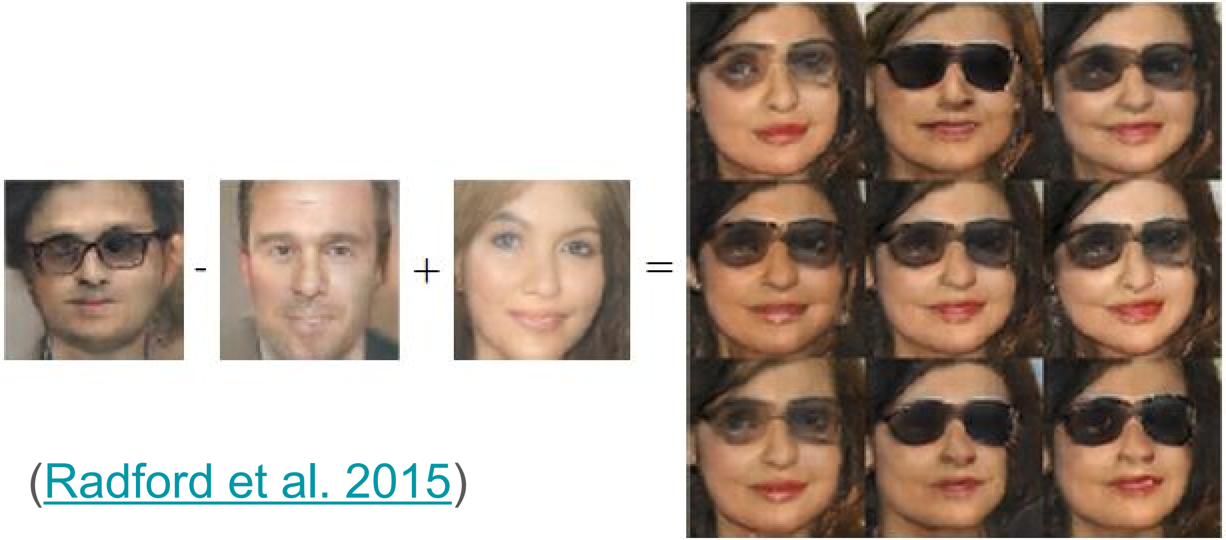


([Zhang et al. 2016])

Unsupervised feature extraction/learning representations

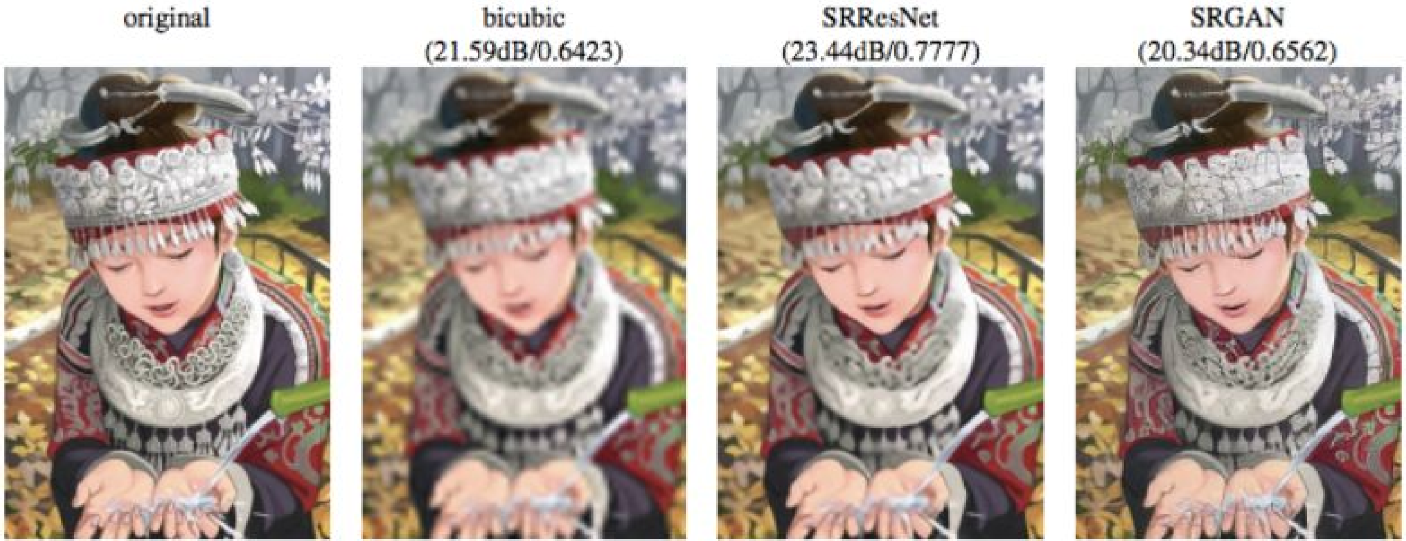
Similarly to word2vec, GANs learn a distributed representation that disentangles concepts such that we can perform semantic operations on the data manifold:

v(Man with glasses) - v(man) + v(woman) = v(woman with glasses)



(Radford et al. 2015)

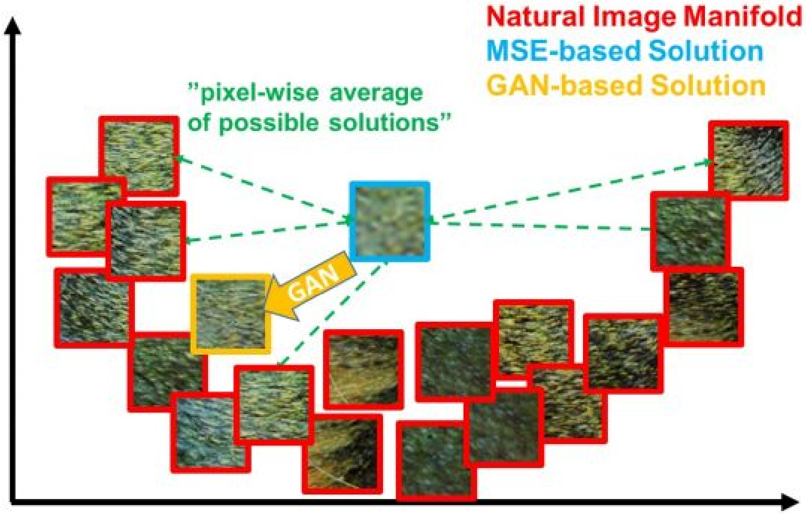
Image super-resolution



(Ledig et al. 2016)

Bicubic: not using data statistics. SRResNet: trained with MSE. SRGAN is able to understand that there are multiple correct answers, rather than averaging.

Image super-resolution



(Ledig et al. 2016)

Averaging is a serious problem we face when dealing with complex distributions.

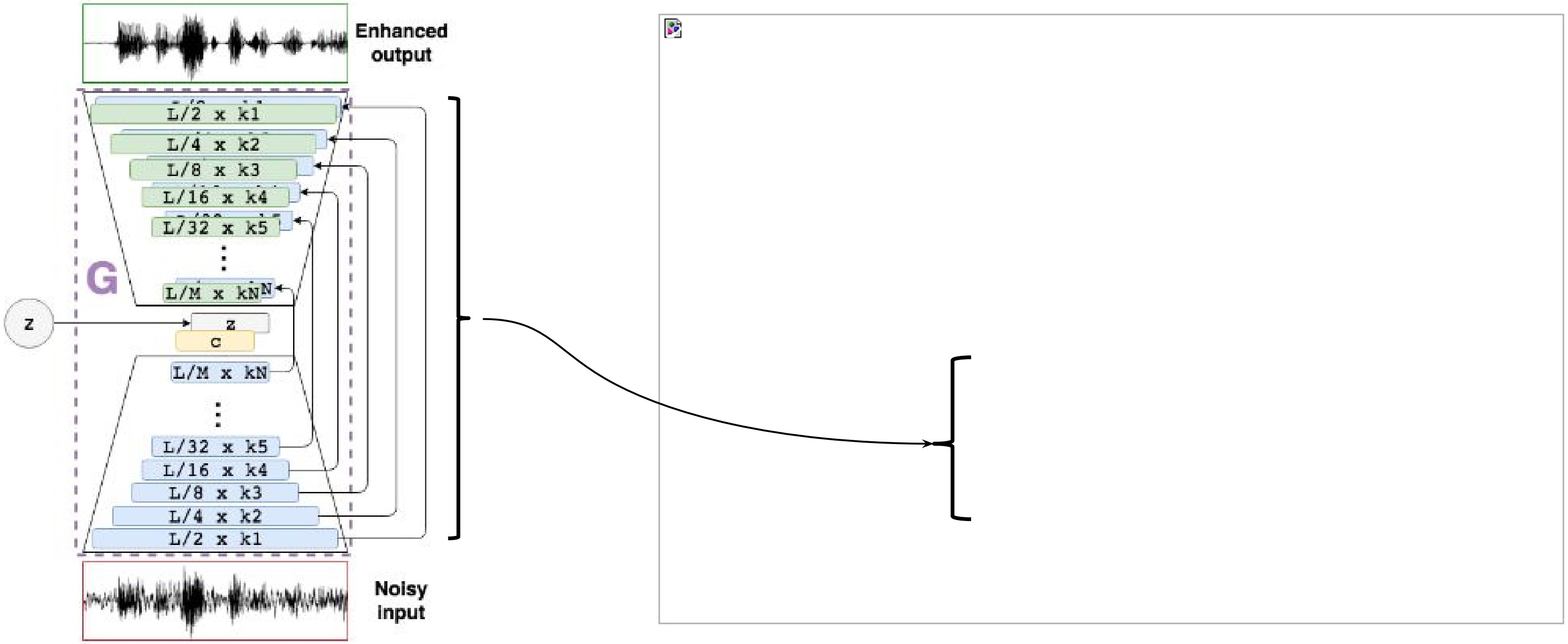
Manipulating images and assisted content creation

https://youtu.be/9c4z6YsBGQ0?t=126

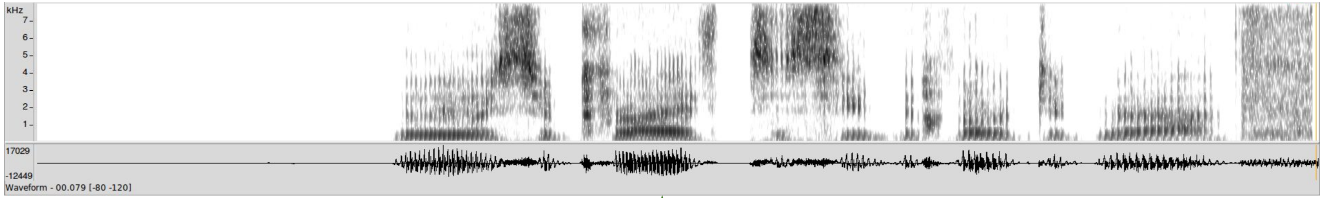
(Zhu et al. 2016)

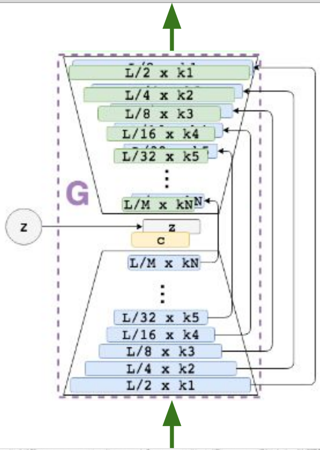
Speech Enhancement

Speech Enhancement GAN (Pascual et al. 2017)

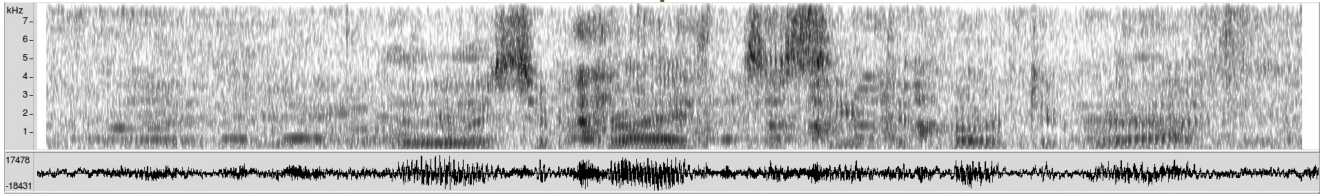


Speech Enhancement





Samples available: http://veu.talp.cat/segan/



Conclusions

Conclusions

- Generative models are built to learn the underlying structures hidden in our high-dimensional data.
- There are currently three main deep generative models: pixelCNN, VAE and GAN.
- PixelCNN factorizes an explicitly known discrete distribution with probability chain rule
    - They are the slowest generative models for their recursive nature.
- VAEs and GANs are capable of factorizing our highly-complex PDFs into a simpler prior of our choice **z**.
- Once we trained VAEs with variational lower bound, we can generate new samples from our learned **z** manifold.
- GANs use an implicit learning method to mimic our data distribution Pdata.
    - GANs are the sharpiest generators at the moment, and a very active research field.
    - They are the hardest ones to train though, because of their equilibria dynamics.

Thanks! Questions?

References

- NIPS 2016 Tutorial: Generative Adversarial Networks (Goodfellow 2016)
- Pixel Recurrent Neural Networks (van den Oord et al. 2016)
- Conditional Image Generation with PixelCNN Decoders (van den Oord et al. 2016)
- Auto-Encoding Variational Bayes (Kingma & Welling 2013)
- https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/
- https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
- Tutorial on Variational Autoencoders (Doersch 2016)
- Improved Techniques for Training GANs (Salimans et al. 2016)
- Generative Adversarial Networks: An Overview (Creswell et al. 2017)
- Generative Adversarial Networks (Goodfellow et al. 2014)