Imperial College London

Generative Adversarial Networks

Deep Learning Course

Kevin Webster, Pierre Harvey Richemond

Course outline

1. Introduction to deep learning
2. Neural networks optimisation
3. Convolutional neural networks
4. Introduction to reinforcement learning - part 1
5. Introduction to reinforcement learning - part 2
6. Sequence models
7. Generative adversarial networks (GANs)
8. Variational autoencoders (VAEs)
9. Normalising flows
10. Theories of deep learning

Yann LeCun on GANs

"There are many interesting recent developments in deep learning...The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). This, and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion." – Y. LeCun

GANs: The original formulation

Quick reminder on probabilistic modelling :

- We want to learn a data from its distribution x based on samples (x_i).
- We are using function approximation where model parameters are θ.
- We will maximize the log-likelihood of the data,

$$\log p(x, \theta)$$

- In practice we maximize the expected log-likelihood

$$\mathbb{E}[\log p(x, \theta)]$$

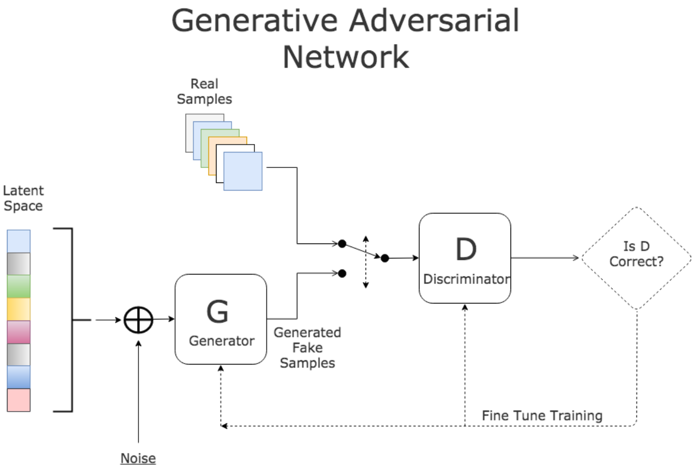
that gets approximated via sampling:

$$\max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log p_{\theta}(x^{(i)})$$

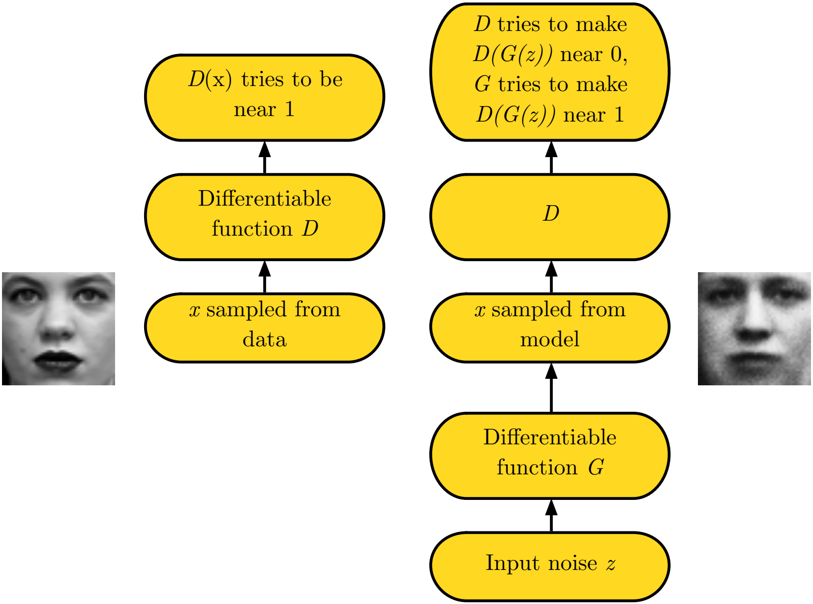
GANs: The original formulation - Splitting things in two

- GANs [Goodfellow et al., 2014] split our probabilistic model in two neural networks : the *generator G* and the *discriminator D*.
- G, with params θ_G, takes a noisy random variable z as an input (with a noise prior distribution $p_z(z)$) and generates candidate samples matching the data distribution x. So the mapping to data space is $G(z, \theta_G)$.
- D, with params θ_D, takes a sample data point as an input and outputs a single scalar $D(x, \theta_d)$. $D(x)$ represents the probability that x came from the data, rather than the generator-implied distribution p_G.

GANs: The original formulation



GANs: The original formulation



$D(\mathrm{x})$ tries to be near 1

Differentiable function D

x sampled from data

D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1

D

x sampled from model

Differentiable function G

Input noise z

Interpretation

- The discriminator learns using traditional supervised learning techniques, dividing inputs into two classes (real or fake) - it is ’the police’.
- The generator (counterfeinter) is trained to fool the discriminator.
- These two neural networks improve by playing an ’adversarial’ game.

Adversarial training (1)

- We train D to maximize the probability of assigning the correct (binary) label both to training examples and samples from G ;
- We train G to maximize $\log[D(G(z))]$, or, equivalently, minimize $\log[1 - D(G(z))]$.
- Like so, training becomes a min-max game :

$$\min_{\theta_G} \quad \max_{\theta_D} \quad \mathbb{E}_{x \sim p_x}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

$$= \min_{\theta_G} \quad \max_{\theta_D} \quad J_D(\theta_G, \theta_D) + J_G(\theta_G, \theta_D)$$

$$= \min_{\theta_G} \quad \max_{\theta_D} \quad V(G, D)$$

Adversarial training (2)

- This formulation lends itself to theoretical mathematical analysis (min-max : either optimization or game theory).
- The discriminator wishes to minimize $J_D(\theta_G, \theta_D)$ and must do so while controlling only θ_D. The generator wishes to minimize $J_G(\theta_G, \theta_D)$ and must do so while controlling only θ_G.
- In practice, we must implement the game using an iterative, numerical approach. Training D to completion in the inner loop of training is computationally prohibitive.
- Instead, we alternate between k steps of training D and one step of training G. This results in D being maintained near its optimal solution, so long as G changes slowly enough.

Potential issues in practice

- Stability : alternate descent/ascent on min/max (saddle point)
- How to choose params like k and/or control the quality of learning of both networks ?
- Gradient saturation : The output of D will be a probability (most likely coming from a sigmoid unit) and can therefore saturate yielding gradient zero. Two main issues could arise.
- Poor initialization would pin D near zero and provide little training gradient (signal) to the discriminator
- Once the generator works well for some part of the distribution, we could stay stuck in that $D = 1$ region for a while (mode collapse)

All these issues are real and very practical concerns that have been iteratively addressed in an onslaught of papers (1500+) in the last three years. We will present some of those practical improvements.

Naive minibatch-SGD GAN algorithm

Algorithm 1 Standard GAN algorithm.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
        • Update discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$
    Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    Update generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
```

Theoretical analysis

But first question : do GANs have convergence guarantees ?
If so, in which sense ?

Likelihood ratio

The cost used for the discriminator is just the standard cross-entropy loss that is minimized when training a standard binary classifier with a sigmoid output. The only difference is that the classifier is trained on two minibatches of data; one coming from the dataset, where the label is 1 for all examples, and one coming from the generator, where the label is 0 for all examples.

We will see that by training the discriminator, we are able to obtain an estimate of the ratio

$$\frac{p_{data}(x)}{p_{model}(x)}$$

at every point x.

Likelihood ratio - 2

The discriminator trained till optimality is

$$D_G^*(x) = \frac{q_{\mathrm{data}}(x)}{q_{\mathrm{data}}(x) + p_G(x)} = \mathrm{sigmoid}(f^*(x)), \tag{1}$$

$$\text{where } f^*(x) = \log q_{\mathrm{data}}(x) - \log p_G(x), \tag{2}$$

and its derivative

$$\nabla_x f^*(x) = \frac{1}{q_{\mathrm{data}}(x)} \nabla_x q_{\mathrm{data}}(x) - \frac{1}{p_G(x)} \nabla_x p_G(x) \tag{3}$$

can be unbounded or even incomputable.

Elementary proof sketch

Just expanding V yields

$$\min_G \quad \max_D \quad V(D,G) = \int_x \Big[p_r(x) \log D(x) + p_G(x) \log(1 - D(x))\Big] dx$$

We now solve for the optimal discriminator D^*. If

$$Y = a \log(y) + b \log(1 - y)$$

then

$$y^* = \frac{a}{a+b} = \frac{1}{1 + \frac{b}{a}}$$

Therefore

$$D^*(x) = \frac{1}{1 + \frac{p_G(x)}{p_r(x)}}$$

.

Likelihood ratio training, illustrated

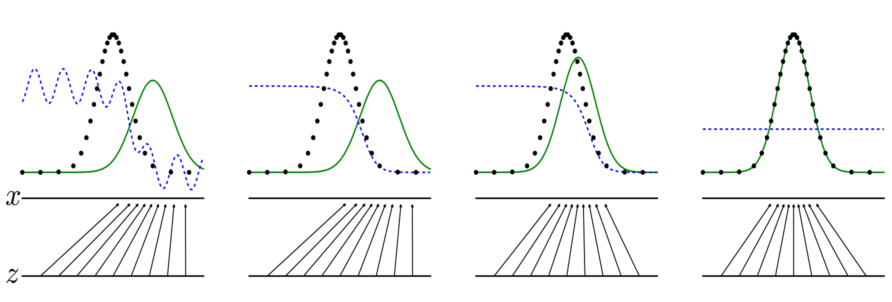


Figure 1: Several steps of adversarial training. The data distribution is black dots, the generator-induced distribution is the green line, and the discriminative distribution is the blue dotted line.

GAN Convergence theorem (Goodfellow, 2014)

Theorem

The global minimum of the virtual training criterion $C(G) = V(D_G^*, G)$ *is achieved if and only if* $p_g = p_{data}$. *At that point,* $C(G)$ *achieves the value* $-\log 4$.

At any point in training (for any p_G), we have:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right)$$

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$$

where *KL* is the Kullback-Leibler divergence between two distributions, and *JSD* is its symmetrized version, the *Jensen-Shannon* divergence.

On balancing G and D, by Ian Goodfellow

’Many people have an intuition that it is necessary to somehow balance the two players to prevent one from overpowering the other. If such balance is desirable and feasible, it has not yet been demonstrated in any compelling fashion. The author’s present belief is that GANs work by estimating the ratio of the data density and model density. This ratio is estimated correctly only when the discriminator is optimal, so it is fine for the discriminator to overpower the generator. Sometimes the gradient for the generator can vanish when the discriminator becomes too accurate. The right way to solve this problem is not to limit the power of the discriminator, but to use a parameterization of the game where the gradient does not vanish.’ [Goodfellow, 2017]
TL;DR: As the discriminator saturates we get vanishing gradients (the density ratio in the KL blows up).

An illustration : Mode collapse

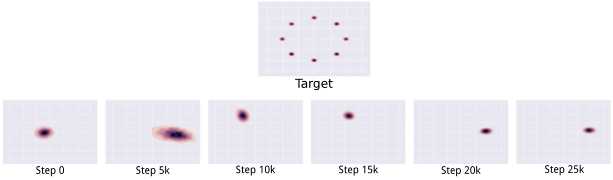


Figure 2: On a Gaussian mixture toy dataset - rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one. On a real dataset, this would translate into low-diversity generative samples.

First attempts at robust implementation (2015)

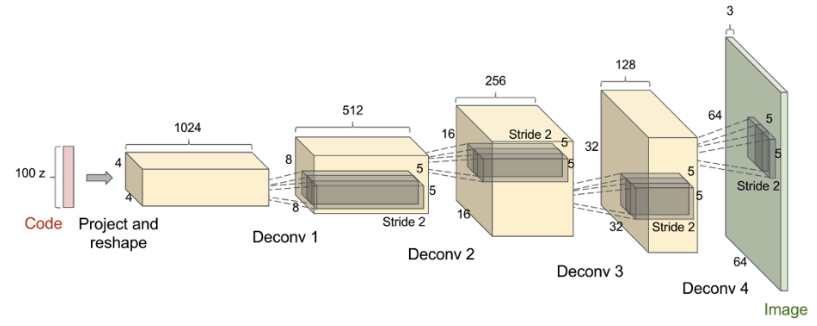


Figure 3: Generator network in the DCGAN architecture [Radford et al., 2015].

First attempts at robust implementation

- To generate images, we use convolutional networks : a standard convnet outputting a scalar from a sigmoid unit as a discriminator, and a reverse convolutional network with *transposed* convolutions (or *deconvolutions*).
- Early model design makes heavy use of batch normalization (in most layers of both the discriminator and the generator, with the two minibatches for the discriminator normalized separately) and of an all-convolutional architecture (no pooling nor “unpooling” layers - when the generator needs to increase the spatial dimension of the representation, it uses transposed convolution with a stride greater than 1).
- However, further stabilization was still required to scale up to more than 32 or 64 square pixels.

GAN training hacks - 1

Taken from Soumith Chintala’s GitHub:

- Normalize the inputs normalize the images between -1 and 1 Tanh as the last layer of the generator output
- Modify the loss function Flip labels when training generator: real = fake, fake = real
- Use a spherical Z Don’t sample from a uniform distribution - sample from a Gaussian distribution.
- BatchNorm Construct different mini-batches for real and fake, i.e. each mini-batch needs to contain only all real images or all generated images.

GAN training hacks - 2

- Avoid Sparse Gradients:ReLu, MaxPool LeakyReLU = good (in both G and D) - For Downsampling, use: Average Pooling, Conv2d + stride, For Upsampling, use: PixelShuffle, ConvTranspose2d + stride
- Use soft and noisy labels Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3 (for example). Make the labels the noisy for the discriminator: occasionally flip the labels when training the discriminator

Wasserstein GAN (1) - 2017

- Motivation : recall the empirical log-likelihood maximization

$$\max_\theta \quad \frac{1}{m}\sum_{i=1}^{m} \log p_\theta(x^{(i)}) \underset{m\to\infty}{\sim} \quad \min_\theta \quad KL(p_x||p_\theta)$$

- [Arjovsky et al., 2017] argue that this only makes sense if the model density p_θ exists (!), hence not in situations where we are dealing with data distributions supported by low dimensional manifolds.
- 'It is then unlikely that the model manifold and the true distribution's support have a non-negligible intersection, and this means that the KL distance is not defined (or simply infinite).'
- Another simpler way to put it is a thought experiment where $p_x = N(0,1)$. Imagine either case of $p_g = N(3,1)$ or $p_g = N(6,1)$, then $\nabla D(p_g) = 0$. Said otherwise, GANs cannot recover from a poor initialization (where there is no overlap between the supports of the real data distribution and the generator's).

Wasserstein GAN (2)

- This disjoint support problem is largely due to the use of the KL or JSD metric (explicitly involving the likelihood ratio). It can be alleviated by using optimal transport instead, and its metric the *Wasserstein* distance.
- The Wasserstein-1 distance between $\mathbb{P}_\alpha$ and $\mathbb{P}_\beta$ is given by

$$\inf_{\pi \in \Pi(\mathbb{P}_\alpha, \mathbb{P}_\beta)} \mathbb{E}_{(x,y)\sim\pi}[|x - y|]$$

  where $\Pi(\mathbb{P}_\alpha, \mathbb{P}_\beta)$ is the set of all coupled random variables π whose marginals are $\mathbb{P}_\alpha$ and $\mathbb{P}_\beta$.
- Intuitively the distance is how much ’mass’ must be transported in order to transform the distribution.
- This generalizes to a W_p analogue of the L^p norm.
- In one dimension only we have the formula that for two r.v.s X and Y whose inverse cumulative density functions are known,

$$W_p(X, Y) = \left(\int_0^1 |F_X^{-1}(u) - F_Y^{-1}(u)|^p du \right)^p$$

Some simple examples

W_p distance between two Diracs :

$$W_p(\delta_{a_1}, \delta_{a_2}) = |a_1 - a_2|$$

W_2 distance between two Gaussians :

$$W_2^2\big(\mathcal{N}(m_1, C_1), \mathcal{N}(m_2, C_2)\big) = \|m_1 - m_2\|_2^2 + \mathrm{tr}\big(C_1 + C_2 - 2\big(C_2^{1/2} C_1 C_2^{1/2}\big)^{1/2}\big)$$

(also known as *Frechet* distance).

Earth moving, illustrated

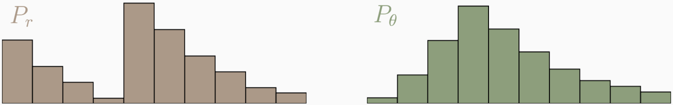


Figure 4: Two discrete distributions P_r and P_θ.

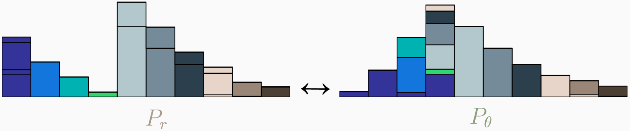
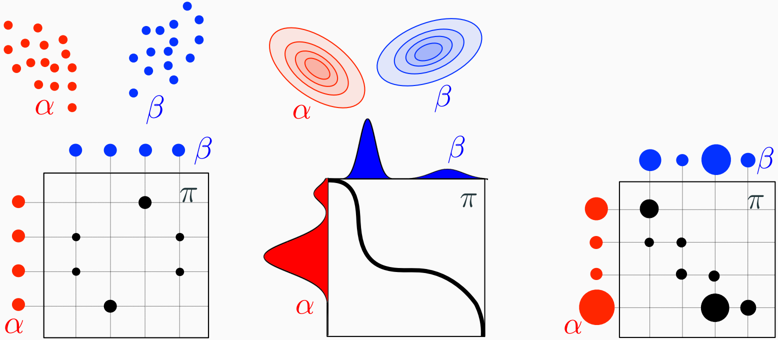


Figure 5: *Earth-moving* between P_r and P_θ.

The Wasserstein distance, illustrated



Discrete

Continuous

Discrete plan

Figure 6: Schematic view of input measures (α, β) and couplings π coming in the optimal transport.

Optimal transport - Interpolation of Gaussians

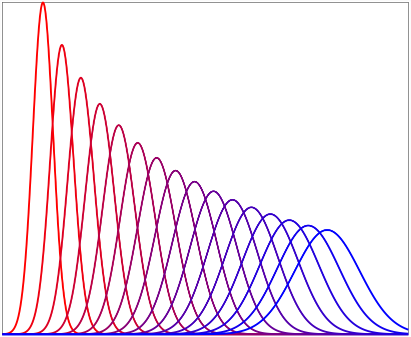


Figure 7: Computation of the *displacement interpolation* between two 1-d Gaussian distributions. The Wasserstein barycenter is unimodal, unlike the L^2 one. W_1 is also called *the Earth Mover distance*.

Wasserstein GAN (3)

For more on the topic, see Cedric Villani's opus magnum Optimal transport: old and new, or Gabriel Peyre's Computational Optimal Transport, whose figures you saw above.
Unlike the parameterized KL we have:

Theorem
For G continuous and locally Lipschitz in its parameters θ, the map $\theta \to W(\mathbb{P}_x, \mathbb{P}_\theta)$ is continuous and diffentiable, a.e.

The Wasserstein distance metric is about as weak as convergence in distribution (so weaker than the KL), but still sensible and better behaved.
Our programme is therefore to replace *KL* with *W*, and compute its gradient ∇_θ.

Kantorovich-Rubinstein duality

In the case of W_1 only, we can make the infimum in the definition tractable using the duality formula

$$W_1(\mathbb{P}_x, \mathbb{P}_\theta) = \sup_{||f||_L \leq 1} \quad \mathbb{E}_{x\sim\mathbb{P}_x}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

where the supremum (integral probability metric) is over the class of 1-Lipschitz test functions. Therefore we can differentiate and get the representation formula:

Theorem

If $\mathbb{P}_\theta$ is the distribution of $G_\theta(Z)$ with Z a r.v. of density p, then we have for all $\mathbb{P}_x$

$$\nabla_\theta W_1(\mathbb{P}_x, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f^*(G_\theta(z))]$$

where f^* realizes the sup *in the Kantorovich-Rubinstein duality above.*

Loss function in practice

How do we find f^* in practice ?

Well we can approximate it via a neural network with weights w and backpropagate through $\mathbb{E}_{z\sim p(z)}[\nabla_\theta f_w(G_\theta(z))]$.

This is because the WGAN value function is straight out of Kantorovich duality:

$$\min_G \quad \max_D \quad \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{y\sim\mathbb{P}_g}[D(y)], \qquad y = G(z) \quad , (z \sim p(z))$$

The Lipschitz constraint is then enforced (very) naively via weight clipping.

This way, the discriminator D has now turned into a *critic* f. The fact that the Wasserstein distance is continuous and differentiable a.e. means that we can (and should) train the critic till optimality (correlation between discriminator loss, and sample quality).

The WGAN algorithm

Algorithm 2 The WGAN algorithm. All experiments in the paper used the values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Input: $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch
  size. $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Input: $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
  while $\theta$ has not converged do
      for $t = 0, ..., n_{\text{critic}}$ do
          • Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
          • Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
          • $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
          • Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
          • $w \leftarrow \text{clip}(w, -c, c)$
      Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
      $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
      $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
```

Differences in practice between WGAN and GAN

- No log in the loss - the output of D is no longer a probability, hence we do not apply any sigmoid units at the output of D
- Weight clipping for D
- Train D more than G
- Use RMSProp instead of ADAM
- Use a low learning rate, [Arjovsky et al., 2017] uses $\alpha = 0.00005$
- Monitor discriminator loss (TensorBoard) to check for convergence

WGAN Gradients, illustrated

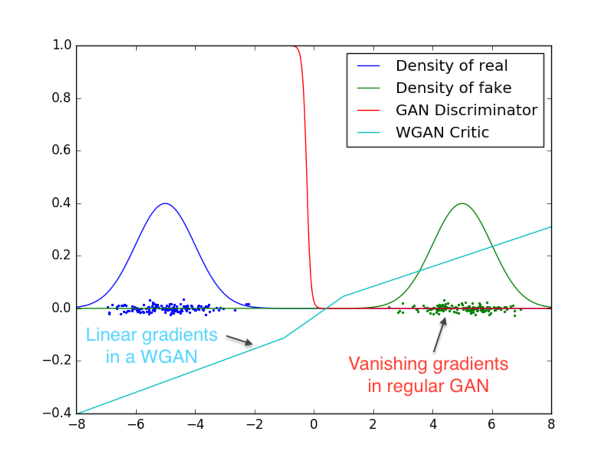


Figure 8: Optimal discriminator and critic when learning to differentiate two Gaussians. The discriminator of a standard GAN saturates and results in vanishing gradients. The WGAN critic provides clean gradients.

More on WGAN

- We have replaced $f(x) = \log(x)$ by $f(x) = x$ by duality in the loss

$$\min_{\theta_g} \max_{\theta_d} \quad \mathbb{E}_{x \sim \mathcal{D}_{real}} f(\mathbf{D}(\mathbf{x}; \theta_d)) + \mathbb{E}_{z \sim \mathcal{G}} f(1 - \mathbf{D}(\mathbf{G}(\mathbf{z}; \theta_g); \theta_d))$$

- This buys us some numerical stability, but in practice, the Wasserstein-GAN is not robust to large learning rates and complex optimizers *a la* ADAM (works with RMSProp).
- WGAN-GP (Gradient Penalty) [Gulrajani et al., 2017] is a notable extension. W_1 weight clipping only really cares about first moment matching - so, combat critic underfitting with adding a gradient regularization term

$$\lambda \cdot \left(||\nabla_x D_w(x)||_2 - 1 \right)^2$$

- Re-metrization alternatives : Least Squares GAN, FisherGAN, SobolevGAN, CramerGAN, Frechet distance... even Wasserstein auto-encoders using similar principles. Endless list on ArXiv, see the excellent review paper [Kurach et al., 2018]...

Spectral Normalization (2018) - 1

- The performance control of the discriminator is still challenging in WGAN. This is because of the basic idea of enforcing the Lipschitz bound via weight clipping.
- Can we normalize discriminator weights in a more principled way ?
- Assuming neural network nonlinearities are Lipschitz themselves, we can control the Lipschitz constant of a multilayer NN by constraining the spectral norm, or largest singular value, $\sigma(W_l)$ of each layer:

$$||f||_{Lip} \leq \prod_{l=1}^{L+1} \sigma(W_l) = \prod_{l=1}^{L+1} \max_{||h||_2 \leq 1} ||W_l h||_2$$

- Spectral normalization [Miyato et al., 2018] simply normalizes the spectral norm of each weight matrix W so that it satisfies Lipschitz constraint 1:

$$\hat{W}_{SN}(W) := \frac{W}{\sigma(W)}$$

Spectral Normalization - 2

- Can we compute or approximate $\sigma(W_l)$ in a tractable way other than singular value decomposition ?
- Yes, with the power iteration method : for square layers, compute $b_{k+1} \leftarrow W_l b_k / ||W_l b_k||$ iteratively from random b_0.
- In general, we begin with vectors $\tilde{\boldsymbol{u}}$ randomly initialized for each weight. If there is no multiplicity in the dominant singular values and if $\tilde{\boldsymbol{u}}$ is not orthogonal to the first left singular vectors, we can produce the first left and right singular vectors through the following update rule:

$$\tilde{\boldsymbol{v}} \leftarrow W^{\mathrm{T}} \tilde{\boldsymbol{u}} / \|W^{\mathrm{T}} \tilde{\boldsymbol{u}}\|_2, \ \tilde{\boldsymbol{u}} \leftarrow W \tilde{\boldsymbol{v}} / \|W \tilde{\boldsymbol{v}}\|_2.$$

  We can then approximate the spectral norm of W with the pair of approximative singular vectors:

$$\sigma(W) \approx \tilde{\boldsymbol{u}}^{\mathrm{T}} W \tilde{\boldsymbol{v}}.$$

Spectral Normalization - 3 - Algorithm

Algorithm 3 SGD with spectral normalization

```
Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
For each update and each layer $l$:
Apply power iteration method to a unnormalized weight $W^l$:
    $\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l\|_2$    (4)
    $\tilde{\boldsymbol{u}}_l \leftarrow W^l \tilde{\boldsymbol{v}}_l / \|W^l \tilde{\boldsymbol{v}}_l\|_2$    (5)
Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:
    $\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}} W^l \tilde{\boldsymbol{v}}_l$
Update $W^l$ with SGD on minibatch data $\mathcal{D}_M$ with learning rate $\alpha$:
    $W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M)$
```

Stabilizing saddle SGD : Prediction methods (2017)

Saddle-point optimization problems have the general form

$$\min_u \max_v \mathcal{L}(u,v)$$

for a loss function $\mathcal{L}$ and variables u and v. The standard stochastic gradient method to solve saddle-point problems involving neural networks alternates between updating u with a gradient *descent* step, and then updating v with a gradient *ascent* step:

$$\begin{aligned} u^{k+1} &= u^k - \alpha_k \cdot \mathcal{L}'_u(u^k, v^k) \quad | \quad \text{gradient descent in } u\text{, starting at } (u^k, v^k) \\ v^{k+1} &= v^k + \beta_k \cdot \mathcal{L}'_v(u^{k+1}, v^k) \quad | \quad \text{gradient ascent in } v\text{, starting at } (u^{k+1}, v^k)\ . \end{aligned}$$

Problem: one proves, assuming exact gradients, that this scheme oscillates around solutions of a simple bilinear saddle $\mathcal{L}(u,v) = v^T K u$ (exercise with ODEs !)

Prediction methods

Yadav and Goldstein [Yadav et al., 2017] propose to stabilize the training of adversarial networks by adding a *prediction* step. Rather than calculating v^{k+1} using u^{k+1}, we first make a prediction, $\bar{u}^{k+1}$, about where the u iterates will be in the future, and use this predicted value to obtain v^{k+1}.

Prediction Method:

$$\begin{aligned} u^{k+1} &= u^k - \alpha_k \cdot \mathcal{L}'_u(u^k, v^k) &&| \quad \text{gradient descent in } u\text{, starting at } (u^k, v^k) \\ \bar{u}^{k+1} &= u^{k+1} + (u^{k+1} - u^k) &&| \quad \textit{predict} \text{ future value of } u \\ v^{k+1} &= v^k + \beta_k \cdot \mathcal{L}'_v(\bar{u}^{k+1}, v^k) &&| \quad \text{gradient ascent in } v\text{, starting at } (\bar{u}^{k+1}, v^k)\ . \end{aligned}$$

The prediction step tries to estimate where u is going to be in the future, by assuming its trajectory remains the same as in the current iteration.

Prediction methods - Intuition

The standard alternating SGD switches between minimization and maximization steps. There is a risk that the minimization step can overpower the maximization step (or vice versa), in which case the iterates will "slide off" the edge of saddle, leading to instability.

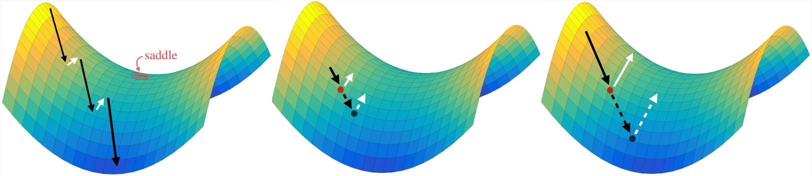


Figure 9: *Left*: If minimization (or, conversely, maximization) is more powerful, the solution path ”slides off” the saddle loss surface and the algorithm becomes unstable (left). *Middle & Right*: The prediction method, enforcing ”momentum in one direction only”.

Prediction methods - Theoretical guarantees

Theorem

Suppose the function $\mathcal{L}(u, v)$ is convex in u, concave in v, and that the partial gradient $\mathcal{L}'_v$ is uniformly Lipschitz smooth in u ($\|\mathcal{L}'_v(u_1, v) - \mathcal{L}'_v(u_2, v)\| \leq L_v\|u_1 - u_2\|$). Suppose further that the stochastic gradients satisfy $\mathbb{E}\|\mathcal{L}'_u(u, v)\|^2 \leq G_u^2$, $\mathbb{E}\|\mathcal{L}'_v(u, v)\|^2 \leq G_v^2$ for scalars G_u and G_v, and that $\mathbb{E}\|u^k - u^\star\|^2 \leq D_u^2$, and $\mathbb{E}\|v^k - v^\star\|^2 \leq D_v^2$ for scalars D_u and D_v. If we choose decreasing learning rates of the form $\alpha_k = \frac{C_\alpha}{\sqrt{k}}$ and $\beta_k = \frac{C_\beta}{\sqrt{k}}$, then the SGD method with prediction converges in expectation, and we have the error bound

$$\mathbb{E}[P(\hat{u}^l, \hat{v}^l)] \leq \frac{1}{2\sqrt{l}}\left(\frac{D_u^2}{C_\alpha} + \frac{D_v^2}{C_\beta}\right) + \frac{\sqrt{l+1}}{l}\left(\frac{C_\alpha G_u^2}{2} + C_\alpha L_v G_u^2 + C_\alpha L_v D_v^2 + \frac{C_\beta G_v^2}{2}\right)$$

where $\hat{u}^l := \frac{1}{l}\sum_{k=1}^{l} u^k$, $\hat{v}^l := \frac{1}{l}\sum_{k=1}^{l} v^k$.

Progressive Growing of GANs (2017)

- Another solution to the disjoint support problem : learn the distribution pinned in place !
- Start with small networks generating 4x4 images and progressively grow them by adding more layers, each doubling the spatial resolution, in a coarse-to-fine fashion.
- According to [Karras et al., 2017], 'This allows the training to first discover large-scale structure of the image distribution and then shift attention to increasingly finer scale detail, instead of having to learn all scales simultaneously. Early on, the generation of smaller images is substantially more stable because there is less class information and fewer modes.'
- This proves so robust that, while best results are obtained with the WGAN-GP loss, one can even use simple least squares.

Progressive Growing of GANs - Architecture

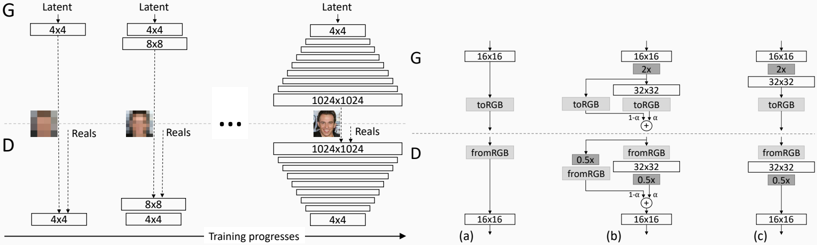


Figure 10: *Left*: Training starts with both generator and discriminator at a spatial resolution of 4x4 pixels - then as training goes further, more layers are added to G and D, doubling the spatial resolution each step. No freezing of layer weights occurs. *Right*: fading in new, double-resolution layers smoothly. α is annealed from 0 to 1.

Progressive Growing of GANs - Results

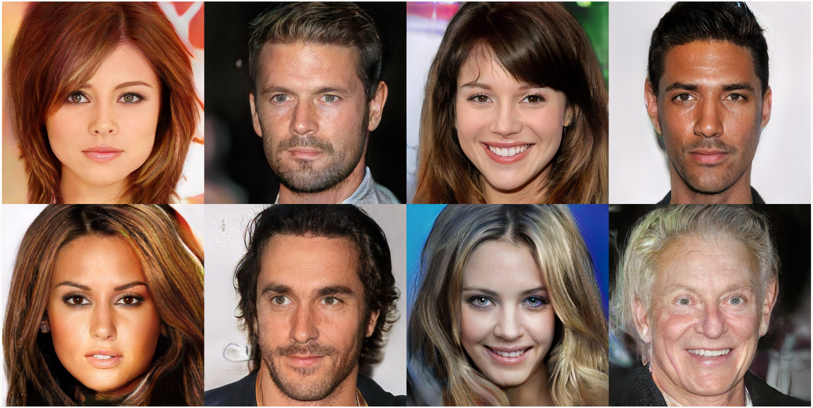


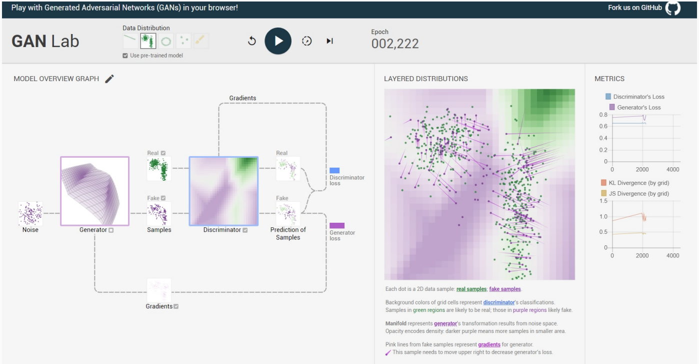
Figure 11: 1024x1024 generated samples on *CelebA-HQ*.

Conclusion

- From MNIST to megapixel synthesis : extraordinary progress in only 3 years
- The current frontier : class-conditional ImageNet [Brock et al., 2018, Zhang et al., 2018]
- Generated samples still suffer from geometry problems - ’global coherence is the primary challenge at high resolution–a model may understand that a spider has ”a number” of legs, and that number is between ”many” and ”lots” but nothing in the networks’ inductive biases really forces it to learn ”eight”’ (Andrew Brock)
- All methods haven’t yet been combined together (progressive growing + spectral norm + prediction)
- Once stabilized, GANs are a valuable, generic distribution inference method
- Hence they also find applications connected to *variational inference* and *reinforcement learning*

Interactive browser visualization

GANLab, made in TensorFlow.js



Each dot is a 2D data sample: real samples; fake samples.

Background colors of grid cells represent discriminator's classifications. Samples in green regions are likely to be real; those in purple regions likely fake.

Manifold represents generator's transformation results from noise space. Opacity encodes density: darker purple means more samples in smaller area.

Pink lines from fake samples represent gradients for generator.
This sample needs to move upper right to decrease generator's loss.

GitHub code repositories

TensorFlow

DCGAN WGAN WGAN-GP Progressive-Growing Generative-Models-Collection Spectral-Norm BigGAN-Colab

Keras

Keras-GAN-repo DCGAN InfoGAN WGAN

Pytorch

PyTorch-GAN-repo WGAN WGAN-GP SN-GAN

References i

Arjovsky, M., Chintala, S., and Bottou, L. (2017).
Wasserstein GAN.
ArXiv e-prints.

Brock, A., Donahue, J., and Simonyan, K. (2018).
Large Scale GAN Training for High Fidelity Natural Image Synthesis.
ArXiv e-prints.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016).
InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.
ArXiv e-prints.

References ii

- Goodfellow, I. (2017).
  NIPS 2016 Tutorial: Generative Adversarial Networks.
  ArXiv e-prints.

- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014).
  Generative Adversarial Networks.
  ArXiv e-prints.

- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017).
  Improved Training of Wasserstein GANs.
  ArXiv e-prints.

References iii

- Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017).
  Progressive Growing of GANs for Improved Quality, Stability, and Variation.
  ArXiv e-prints.

- Kurach, K., Lucic, M., Zhai, X., Michalski, M., and Gelly, S. (2018).
  The GAN Landscape: Losses, Architectures, Regularization, and Normalization.
  ArXiv e-prints.

- Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018).
  Spectral Normalization for Generative Adversarial Networks.
  ArXiv e-prints.

References iv

- Radford, A., Metz, L., and Chintala, S. (2015).
  Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.
  ArXiv e-prints.
- Yadav, A., Shah, S., Xu, Z., Jacobs, D., and Goldstein, T. (2017).
  Stabilizing Adversarial Nets With Prediction Methods.
  ArXiv e-prints.
- Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. (2018).
  Self-Attention Generative Adversarial Networks.
  ArXiv e-prints.
- Zhao, J., Mathieu, M., and LeCun, Y. (2016).
  Energy-based Generative Adversarial Network.
  ArXiv e-prints.