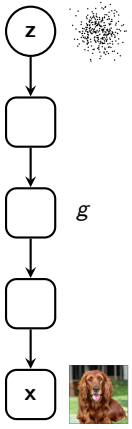
Aalto University

CS-E4890: Deep Learning
Generative adversarial networks

Alexander Ilin

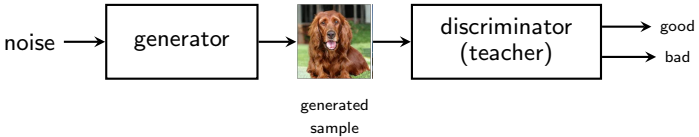
Generative model of VAE

- Recall the generative model model of the VAE:
  - Hidden variables $\mathbf{z}$ are normally distributed: $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$
  - Data vectors $\mathbf{x}$ are nonlinear transformations of the latent variables with possible noise $\boldsymbol{\varepsilon}$:
$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta}) + \boldsymbol{\varepsilon}$$
- VAE is an explicit density model because we define an explicit parametric for of $p(\mathbf{x})$, for example:
$$p(\mathbf{x}) = \prod_i \int p(\mathbf{z}_i) p(\mathbf{x}_i \mid \mathbf{z}_i, \boldsymbol{\theta}) d\mathbf{z}_i = \prod_i \int \mathcal{N}(\mathbf{z}_i \mid 0, \mathbf{I}) \mathcal{N}(\mathbf{x}_i \mid g(\mathbf{z}_i, \boldsymbol{\theta}), \sigma^2 \mathbf{I}) d\mathbf{z}_i$$
- In this lecture, we consider generative adversarial networks (GAN) which is an implicit density model. We can draw samples from the model but we do not explicitly define $p(\mathbf{x})$.



Generative adversarial networks (GAN)

- GANs were originally proposed by Goodfellow et al. (2014).
- GAN consists of a generator which generates samples and a discriminator which tells whether the generated samples are good or bad.



- The generator can be any parametric differentiable model (a deep neural network) that generates random samples.
- The discriminator is a classifier (a deep neural network) that classifies the generated samples into two classes.

Simplest GAN generator

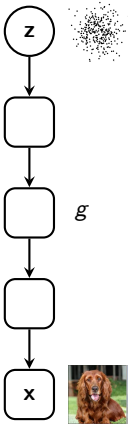
- A popular choice for the GAN generation process:
  - sample from an isotropic Gaussian distribution

$$\mathbf{z} \sim \mathcal{N}(0, I)$$

  - transform $\mathbf{z}$ into the data space by a deep neural network

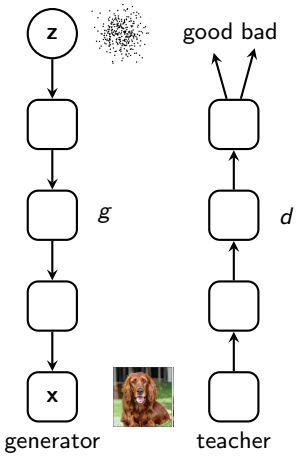
$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta})$$

- Other popular strategies:
  - Apply additive or multiplicative noise to hidden layers.
  - Concatenate noise to hidden layers.



GAN discriminator

- The generator is guided by the teacher network that assesses the quality of the generated samples.
  - Teacher: a classifier that separates samples into classes "good" and "bad".
- How to train the teacher $d(\mathbf{x}, \boldsymbol{\theta}_d)$?
  - Class "good": samples from the training set.
  - Class "bad": samples generated by the generator.
- The teacher is more traditionally called *discriminator*.



Training the GAN discriminator

- The discriminator $d(\mathbf{x})$ learns to separate generated samples from training data:

$$\text{(bad=generated=fake)} \quad 0 < d(\mathbf{x}) < 1 \quad \text{(good=real)}$$

- The discriminator can be trained by maximizing the following log-likelihood function:

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log d(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_g} \log(1 - d(\mathbf{x})) \to \max_d$$

  or equivalently:

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log d(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})} \log(1 - d(g(\mathbf{z}))) \to \max_d$$

- In practice, if we have N real examples $\mathbf{x}_i$ and N generated examples $g(\mathbf{z}_i)$, we minimize the standard binary cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log d(\mathbf{x}_i) - \frac{1}{N}\sum_{i=1}^{N} \log(1 - d(g(\mathbf{z}_i)))$$

Training the GAN generator

- The generator $g(\mathbf{z})$ is trained to produce samples that are classified as real by the discriminator:

$$\text{(bad=generated=fake)} \quad 0 < d(\mathbf{x}) < 1 \quad \text{(good=real)}$$

- The generator can be trained by maximizing the following function:

$$\mathbb{E}_{\mathbf{x}\sim p_g} \log d(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})} \log d(g(\mathbf{z})) \to \max_g$$

- In practice, if we have N generated examples $g(\mathbf{z}_i)$, we can minimize the following loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log d(g(\mathbf{z}_i))$$

A non-saturating objective for the generator

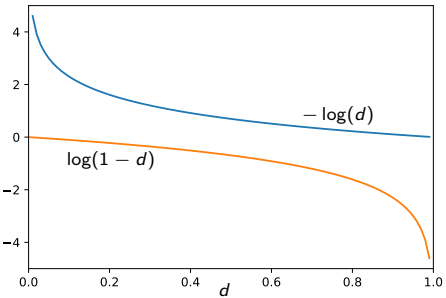
- In principle, there are two ways to train the generator:
  - To minimize the probability of being fake:

$$\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})} \log(1 - d(g(\mathbf{z}))) \to \min_g$$

  - To maximize the probability of being real:

$$\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})} \log(d(g(\mathbf{z}))) \to \max_g$$

  Both formulations result in the same fixed point.



- The latter formulation provides much stronger gradients. In the beginning of training, the discriminator can reject samples produced by the generator with high confidence ($d \approx 0$). In this regime, the generator receives almost no gradient information, which slows down the convergence.

GAN training procedure

1. Update the discriminator:
   - Sample N examples $\mathbf{x}_i$ from the training set.
   - Generate N samples $g(\mathbf{z}_i)$ using the generator.
   - Compute the binary cross-entropy loss

$$\mathcal{L}_d = -\frac{1}{N}\sum_{i=1}^{N} \log d(\mathbf{x}_i) - \frac{1}{N}\sum_{i=1}^{N} \log(1 - d(g(\mathbf{z}_i)))$$

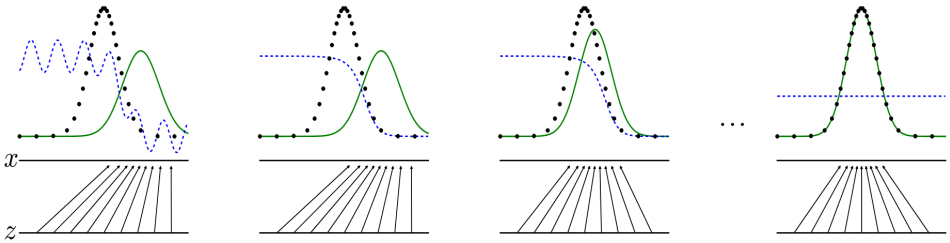
   - Update $\boldsymbol{\theta}_d$ by stochastic gradient descent: $\boldsymbol{\theta}_d \leftarrow \boldsymbol{\theta}_d - \nabla_{\boldsymbol{\theta}_d}\mathcal{L}_d$

2. Update the generator:
   - Generate N samples $g(\mathbf{z}_i)$ using the generator.
   - Compute the loss function

$$\mathcal{L}_g = -\frac{1}{N}\sum_{i=1}^{N} \log d(g(\mathbf{z}_i))$$

   - Update $\boldsymbol{\theta}_g$ by stochastic gradient descent: $\boldsymbol{\theta}_g \leftarrow \boldsymbol{\theta}_g - \nabla_{\boldsymbol{\theta}_g}\mathcal{L}_g$ (gradients flow through the discriminator).

Illustration of GAN training



$d(x)$ is a partially accurate classifier.

$d(x)$ is trained to discriminate samples from data.

After an update to g, gradient of d has guided $g(z)$ to flow to regions that are more likely to be classified as data.

After several steps of training, g and d reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions $d(x) = 0.5$.

Images from (Goodfellow et al., 2014)

Minimax game

- A popular view at GANs: It is a two-player minimax game in which the generator tries to fool the discriminator and the discriminator tries to catch the fakes.
- The game can be described with one objective:

$$v(g, d) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log d(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} \log(1 - d(g(\mathbf{z})))$$
$$g^* = \arg \min_g \max_d v(g, d)$$

- The discriminator is trained to maximize $v(g, d)$. The generator tries to minimize $v(g, d)$.
- The equilibrium (also known as Nash equilibrium) is a saddle point of v.
- Nash equilibrium: No player has anything to gain by changing only their own strategy.

GAN theoretical results (Goodfellow et al., 2014)

- For fixed generator g, the optimal discriminator is given by

$$d_g^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}$$

- If we use the optimal discriminator $d_g^*(\mathbf{x})$ to tune the generator, minimization of the GAN loss $\mathcal{L}_g$ is equivalent to minimization of the Jensen-Shannon divergence between the model's distribution p_g and the data distribution p_{data}:

$$JSD\big(p_{\text{data}} \,\|\, p_g\big) = KL\left(p_{\text{data}} \,\middle\|\, \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \,\middle\|\, \frac{p_{\text{data}} + p_g}{2}\right)$$

  The global minimum of $\mathcal{L}_g$ is achieved if and only if $p_g = p_{\text{data}}$.

Difficulties in training GANs

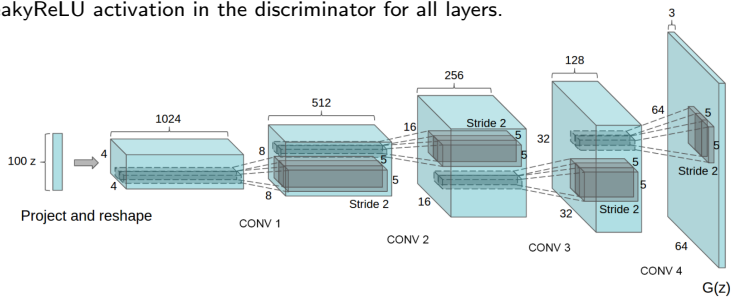
- Training of GANs is often unstable: the convergence may be slow or difficult to achieve.
- A typical problem is so-called *mode collapse*: the generator produces the same output point (or slight variations of the same output, e.g., different views of the same dog) that the discriminator believes is most likely to be real rather than fake.
- There has been a lot of progress in GAN research and the results obtained with modern GANs look very impressive.

Deep Convolutional GAN (DCGAN)

(Radford et al., 2015)

Deep Convolutional GAN (DCGAN) (Radford et al., 2015)

- A simple GAN architecture that can be trained in a relatively stable manner:
  - Replace any pooling layers with strided convolutions (discriminator).
  - Use transposed convolutions in the generator.
  - Remove fully connected hidden layers.
  - Use batchnorm in both the generator and the discriminator.
  - Use ReLU activation in generator for all layers except for the output, which uses Tanh.
  - Use LeakyReLU activation in the discriminator for all layers.



Tricks to improve stability of GAN training

Wasserstein GAN

(Arjovsky et al., 2017)

(Gulrajani et al., 2017)

Earth-Mover distance

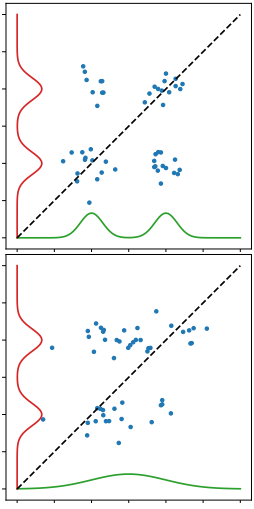
- Instead of the Jensen-Shannon divergence

$$JSD(p_{\text{data}}, p_g) = KL(p_{\text{data}} \,\|\, p_m) + KL(p_g \,\|\, p_m),$$
$$p_m = (p_{\text{data}} + p_g)/2$$

WGAN proposes to compare p_{data} and p_g using the Earth-Mover distance or Wasserstein-1:

$$W(p_{\text{data}}, p_g) = \inf_{\gamma \in \prod(p_{\text{data}}, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

where $\prod(p_{\text{data}}, p_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are p_{data} and p_g.



Earth-Mover distance

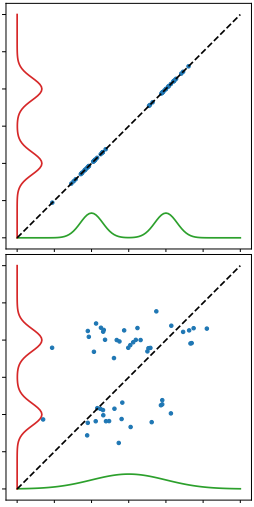
- Instead of the Jensen-Shannon divergence

$$JSD(p_{\text{data}}, p_g) = KL(p_{\text{data}} \,\|\, p_m) + KL(p_g \,\|\, p_m),$$
$$p_m = (p_{\text{data}} + p_g)/2$$

WGAN proposes to compare p_{data} and p_g using the Earth-Mover distance or Wasserstein-1:

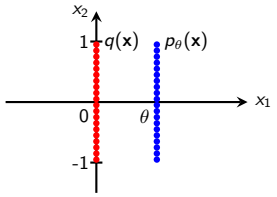
$$W(p_{\text{data}}, p_g) = \inf_{\gamma \in \prod(p_{\text{data}}, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

where $\prod(p_{\text{data}}, p_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are p_{data} and p_g.

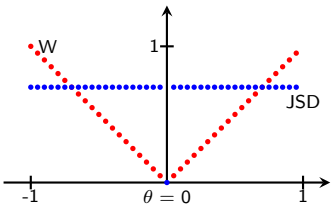


Earth-Mover distance vs JSD

- Suppose we want to compute the distance between distributions $q(\mathbf{x})$ and $p_\theta(\mathbf{x})$. x_2 is uniformly distributed and x_1 is constant in both distributions.



- Earth-Mover distance is a smooth function of θ.
- JSD is constant everywhere except $\theta = 0$.
- The Earth-Mover distance provides a better training signal.



Estimation of the Earth-Mover distance

$$W(p_{\text{data}}, p_g) = \inf_{\gamma \in \prod(p_{\text{data}}, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

- The Earth-Mover distance is intractable. However, the following holds:

$$W(p_{\text{data}}, p_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p_{\text{data}}}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]$$

where the supremum is over all the 1-Lipschitz functions f, that is functions that satisfy:

$$\|f(x_2) - f(x_1)\| \leq \|x_2 - x_1\|$$

- In order to compute the distance, we need to find function f that maximizes the difference of $\mathbb{E}[f(x)]$ under the two distributions.
- If we cannot find f that gives a non-zero difference, the two distributions are identical.
- If we do not restrict f in any way, the difference can be made infinitely large.
- The constraint $\|f\|_L \leq 1$ limits the set of possible functions making the distance well defined.

Estimation of the Earth-Mover distance with a neural network

- If we use K-Lipschitz functions:

$$\|f(x_2) - f(x_1)\| \leq K \|x_2 - x_1\| ,$$

  we compute the distance up to a constant multiplier K:

$$K \cdot W(p_{\text{data}}, p_g) = \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_{\text{data}}}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]$$

- We can model functions f with a neural network f_w with parameters w and limit the weight space $\mathcal{W}$ to be compact (to guarantee that functions are K-Lipschitz).
  - For example, we can limit the weight values $w_i \in [-0.01, 0.01]$.
- Thus, we can estimate the distance between two distributions by solving the following optimization problem:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_{\text{data}}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]$$

Wasserstein GAN (Arjovsky et al., 2017)

1. Update the discriminator:
   - Sample N examples $\mathbf{x}_i$ from the training set.
   - Generate N samples $g_\theta(\mathbf{z}_i)$ using the generator.
   - Compute loss
   $$\mathcal{L}_d = \frac{1}{N}\sum_{i=1}^{N} f_w(g_\theta(\mathbf{z}_i)) - \frac{1}{N}\sum_{i=1}^{N} f_w(\mathbf{x}_i)$$
   - Update w by stochastic gradient descent: $w \leftarrow w - \nabla_w \mathcal{L}_d$
   - Clip the weights $w_i \leftarrow \text{clip}(w_i, -c, c)$.

2. Update the generator:
   - Generate N samples using the generator.
   - Compute loss:
   $$\mathcal{L}_g = -\frac{1}{N}\sum_{i=1}^{N} f_w(g_\theta(\mathbf{z}^{(i)}))$$
   - Update $\boldsymbol{\theta}_g$ by stochastic gradient descent: $\boldsymbol{\theta}_g \leftarrow \boldsymbol{\theta}_g - \nabla_{\boldsymbol{\theta}_g} \mathcal{L}_g$ (gradients flow through the discriminator).

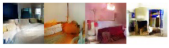
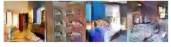
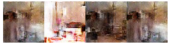
Improved Training of Wasserstein GANs (Gulrajani et al., 2017)

- Clipping in WGAN leads to optimization difficulties: without careful tuning of the clipping threshold c, the gradients can vanish or explode.
- Weight clipping biases the discriminator towards much simpler functions.
- Gulrajani et al. (2017) proposed an alternative to clipping. The loss function minimized by the discriminator contains a gradient penalty term:

$$\mathcal{L}_{d,gp} = \mathcal{L}_d + \underbrace{\lambda \mathbb{E}_{\hat{x}\sim p_{\hat{x}}} \left(\|\nabla_{\hat{x}} f_w(\hat{x})\|_2 - 1 \right)^2}_{\text{gradient penalty}}$$

  - The norm of the gradient should be at most 1 for 1-Lipschitz functions.
  - The authors show that the optimal critic should contain straight lines with gradient norm 1 connecting coupled points from p_{data} and p_g.
  - $\hat{x}$ are sampled uniformly along straight lines between pairs of points sampled from p_{data} and p_g.

Generated samples

DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)			
G: No BN and a constant number of filters, D: DCGAN				
G: 4-layer 512-dim ReLU MLP, D: DCGAN				
No normalization in either G or D				
Gated multiplicative nonlinearities everywhere in G and D				
tanh nonlinearities everywhere in G and D				
101-layer ResNet G and D				

Spectral Normalization

(Miyato et al., 2018)

Motivation

- The motivation is similar to the one of Wasserstein GAN.
- For the conventional form of the GAN objective function

$$v(g, d) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log d(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})} \log(1 - d(g(\mathbf{z})))$$

  the optimal discriminator is given by

$$d_g^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}} + p_g(\mathbf{x})}$$
$$d_g^*(\mathbf{x}) = \text{sigmoid}(f^*(\mathbf{x})), \text{ with } f^*(\mathbf{x}) = \log p_{\text{data}}(\mathbf{x}) - \log p_g(\mathbf{x})$$

- The derivatives of the optimal discriminator can be unbounded and even incomputable:

$$\nabla_{\mathbf{x}} f^*(\mathbf{x}) = \frac{1}{p_{\text{data}}(\mathbf{x})} \nabla_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) - \frac{1}{p_g(\mathbf{x})} \nabla_{\mathbf{x}} p_g(\mathbf{x})$$

- Proposed solution: Introduce some regularity condition to the derivatives of $f(\mathbf{x})$.

Regularization of discriminator

- Inspired by WGAN, the authors propose to search for the discriminator from the set of K-Lipschitz continuous functions:

$$d_g \leftarrow \arg \max_{\|f\|_{\text{Lip}} \leq K} v(g, d)$$

  where $\|f\|_{\text{Lip}}$ means the smallest value M such that $\|f(\mathbf{x}) - f(\mathbf{x}')\| \,/\, \|\mathbf{x} - \mathbf{x}'\| \leq M$ for any $\mathbf{x}$, $\mathbf{x}'$.
- Intuition: We do not want the discriminator to change too fast. Using the teacher-student analogy, we want our teacher to give constructive feedback instead of saying 'this is very wrong'.
- Note: We do not need to change the objective (like in WGAN), we can still use the conventional GAN objective $v(\mathbf{w}, \boldsymbol{\theta})$!

Spectral normalization of a linear layer

- First consider a linear discriminator $f(\mathbf{x}) = \mathbf{W}\mathbf{x}$. In this case, the Lipschitz norm $\|f\|_{\text{Lip}}$ is given by the spectral norm $\sigma(\mathbf{W})$ of $\mathbf{W}$ which is the largest singular value of $\mathbf{W}$.
- *Spectral normalization*: We want $\mathbf{W}$ to satisfy the constraint $\sigma(\mathbf{W}) = 1$, which we achieve by:

$$f(\mathbf{x}) = \frac{\mathbf{W}}{\sigma(\mathbf{W})}\mathbf{x}$$

- To implement this, we need an efficient (and differentiable) way to compute the spectral norm $\sigma(\mathbf{W})$ of a matrix. Miyato et al. (2018) propose to use the power iteration method:

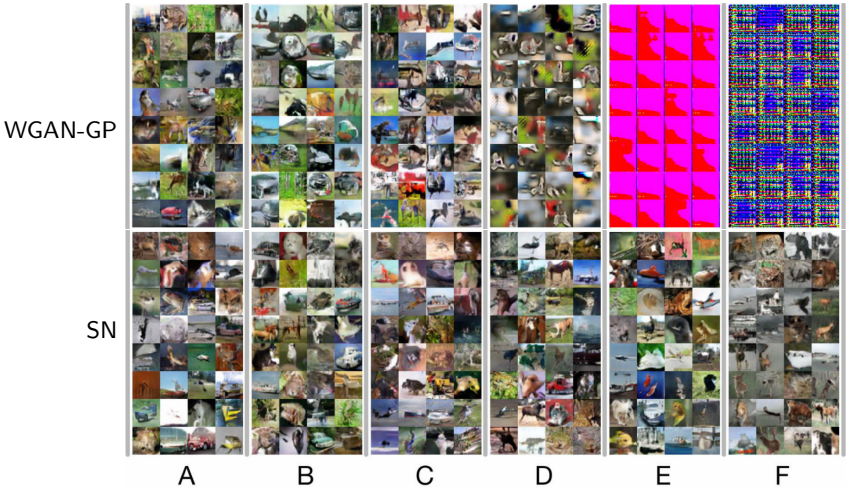
$$\mathbf{v} \leftarrow \frac{\mathbf{W}^\top \mathbf{u}}{\|\mathbf{W}^\top \mathbf{u}\|}, \qquad \mathbf{u} \leftarrow \frac{\mathbf{W}\mathbf{v}}{\|\mathbf{W}\mathbf{v}\|}, \qquad \sigma(\mathbf{W}) = \mathbf{u}^\top \mathbf{W}\mathbf{v}$$

  where $\mathbf{u}$ and $\mathbf{v}$ are vectors that are updated in each forward pass (before training $\mathbf{u}$ and $\mathbf{v}$ are initialized randomly).

Spectral normalization in a deep network

- For a deep neural network, the authors propose to apply spectral normalization to each linear layer. One needs to update vectors $\mathbf{u}_l$ and $\mathbf{v}_l$ for each linear layer l.
- The authors show that this guarantees that $\|f\|_{\text{Lip}} \leq 1$ for popular choices of nonlinearities, such as ReLU.
- For convolutional layers, weights $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}} \times h \times w}$ are treated as a 2-D matrices of dimension $d_{\text{out}} \times (d_{\text{in}} hw)$.

Generated samples



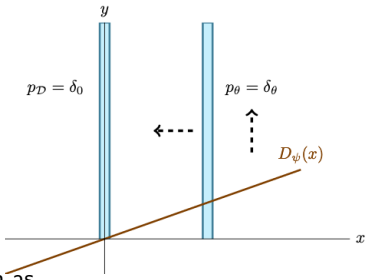
For different architectures (A–F), SN provides more stable results.

Zero-centered gradient penalties

(Mescheder et al., 2018)

A toy problem for studying the convergence of GANs

- Mescheder et al. (2018) studied the convergence in this optimization problem using a simple example:
  - The true data distribution is a Dirac-distribution concentrated at 0.
  - The generator distribution is $p_\theta = \delta_\theta$.
  - The discriminator is linear $d_\phi(x) = \phi x$.



- We can write different variants of the GAN objective function as

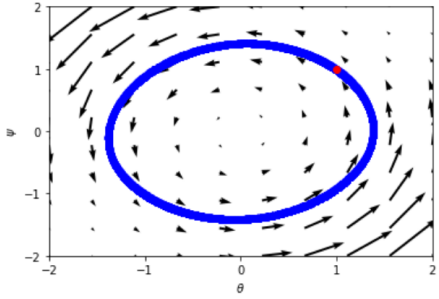
$$v(\theta, \phi) = \mathbb{E}_{p_{\text{data}}(x)}[f(-d_\phi(x))] + \mathbb{E}_{p(z)}[f(d_\phi(g_\theta(z)))]$$
$$v(\theta, \phi) \to \min_\theta \max_\phi$$

where $f(t) = -\log(1 + \exp(-t))$ yields the conventional GAN objective:

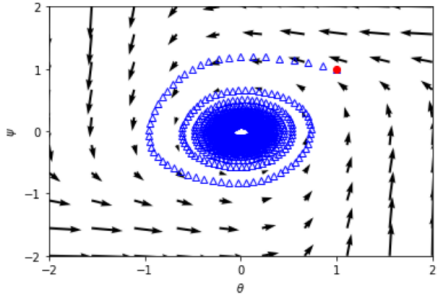
$$v(g, d) = \mathbb{E}_{x \sim p_{\text{data}}} \log d(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - d(g(z)))$$

Convergence of original GAN

- Parameters θ, ϕ can be optimized by simultaneous or alternating gradient descent.
- The optimization trajectories can be visualized on the 2d plane.

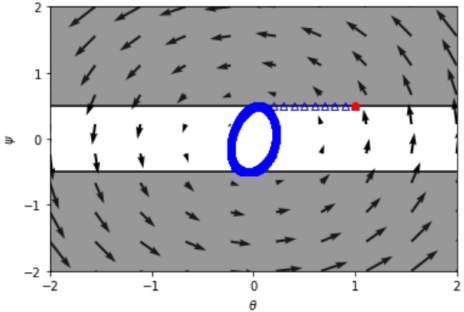


Unregularized GAN training does not always converge to the Nash-equilibrium.

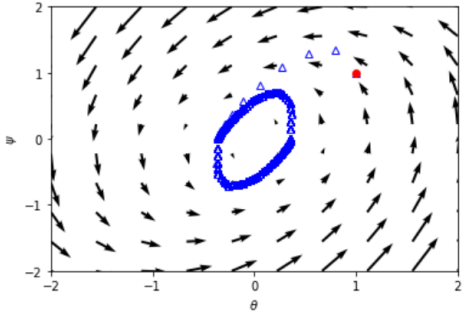


GAN with a non-saturating objective for the generator converges, albeit with an extremely slow convergence rate.

- WGAN and WGAN-GP with a finite number of discriminator updates per generator update do not always converge to the equilibrium point.



WGAN ($n_d = 5$)

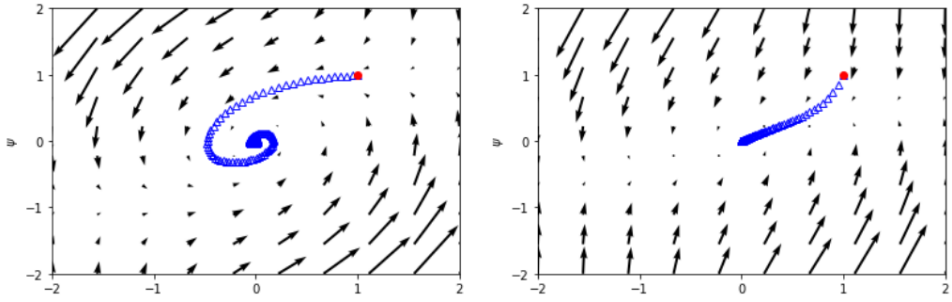


WGAN-GP ($n_d = 5$)

Zero-centered gradient penalties

- Penalizing the gradients of the discriminator has a positive effect on convergence: The discriminator is penalized for deviating from the Nash-equilibrium.
- The authors proposed *zero-centered* gradient penalty: Penalize gradients either on real data (R_1) or on generated samples (R_2):

$$R_1 = \frac{\gamma}{2}\mathbb{E}_{x\sim p_{\text{data}}}(\|\nabla d(x)\|^2), \quad R_2 = \frac{\gamma}{2}\mathbb{E}_{x\sim p_g}(\|\nabla d(x)\|^2)$$



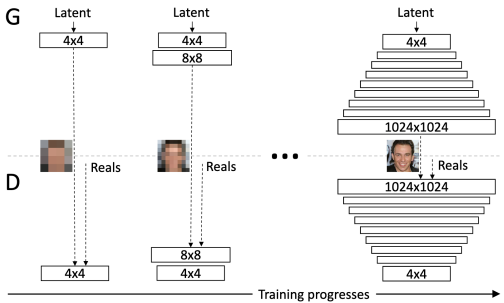
R_1 and R_2 are equivalent for the toy problem.

Progressive growing (ProGAN)

(Karras et al., 2018)

Progressive growing of GANs (Karras et al., 2018)

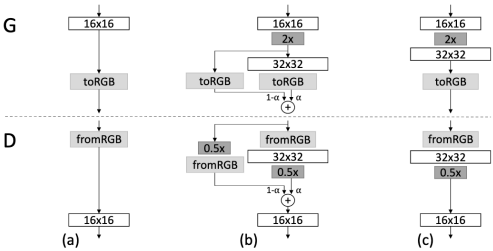
- The idea: start by building a generative model for low-resolution images, then progressively increase the resolution by adding layers to the networks.
- Generation of smaller low-resolution images is more stable because the problem is much simpler compared to the end goal.
- The training time is reduced because many iterations are done at lower resolutions.



Progressive growing

- When doubling the resolution, the new layers are faded in smoothly: During the transition (b) the layers that operate on the higher resolution are treated like a residual block, whose weight α increases linearly from 0 to 1.

- When training the discriminator, real images are downscaled to match the current resolution of the network.
- During a resolution transition, the authors interpolate between two resolutions of the real images, similarly to how the generator output combines two resolutions.



2x: doubling the resolution using nearest neighbor filtering
0.5x: halving the image resolution using average pooling
toRGB: project feature vectors to RGB colors with 1×1 convolutions
fromRGB does the reverse using 1×1 convolutions

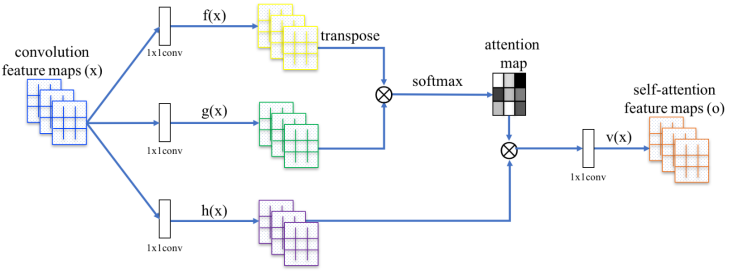
ProGAN: Generated samples

Self-Attention GAN (SAGAN)

(Zhang et al., 2018)

Self-Attention GAN (SAGAN) (Zhang et al., 2018)

- SAGAN: Using a self-attention module (inspired by the transformers) in the generator.



- The results suggest that a self-attention module is beneficial for improving the quality of the generated images.

Images generated with a large-scale SAGAN (Brock et al., 2018)

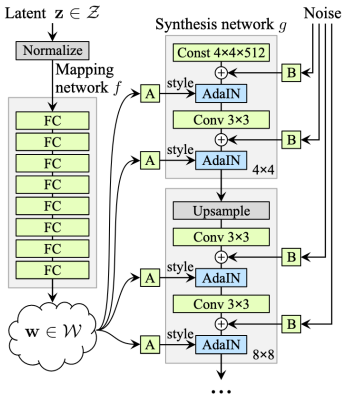
Style-Based Generators (StyleGAN)

(Karras et al., 2018)

- Motivated by the style-transfer literature (Huang and Belongie, 2017), the authors re-design the generator architecture.
- The generator starts from a learned constant input.
- The "style" of the image at each convolution layer is adjusted using adaptive instance normalization:

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$

  where $\mathbf{x}_i$ is one feature map, $\mu(\mathbf{x}_i)$ and $\sigma(\mathbf{x}_i)$ are its mean and standard deviation.
- The style vectors $(\mathbf{y}_s, \mathbf{y}_b)$ are produced from latent code z by an MLP.
- Additional noise is injected directly into the network.

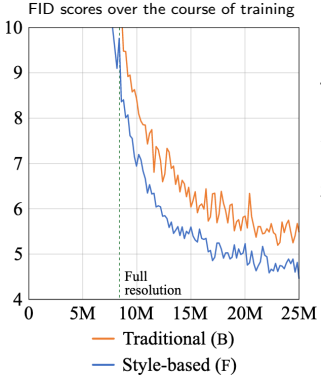


StyleGAN improves the quality of generated images

- Automatic evaluation of the quality of generated images is not a trivial task. One popular metric is called Fréchet Inception distance (FID) (Heusel et al., 2017). It is a Fréchet distance between two Gaussian distributions $\mathcal{N}(\mathbf{m}_r, \mathbf{C}_r)$ and $\mathcal{N}(\mathbf{m}_g, \mathbf{C}_g)$:

$$\text{FID} = \|\mathbf{m}_r - \mathbf{m}_g\|_2^2 + \text{Tr}(\mathbf{C}_r + \mathbf{C}_g - 2(\mathbf{C}_r\mathbf{C}_g)^{1/2})$$

- Statistics $\mathbf{m}_r$ and $\mathbf{C}_r$ are computed in the following way:
  - Propagate real images through an Inception-v3 classifier pre-trained on natural images.
  - Compute mean $\mathbf{m}_r$ and covariance matrix $\mathbf{C}_r$ of the outputs of one of the layers.
- $\mathbf{m}_g$ and $\mathbf{C}_g$ are computed similarly on generated images.

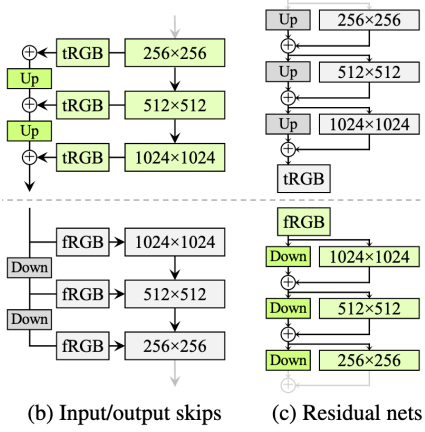


Horizontal axis denotes the number of training images seen by the discriminator. The dashed vertical line marks the point when training has progressed to full 1024^2 resolution.

StyleGAN2 (Karras et al., 2019)

- Replaced adaptive instance normalization with weight demodulation.
- New architectures of the generator and discriminator.
- Removed progressive growing.

	Configuration	FID ↓
A	Baseline StyleGAN [24]	4.40
B	+ Weight demodulation	4.39
C	+ Lazy regularization	4.38
D	+ Path length regularization	4.34
E	+ No growing, new G & D arch.	3.31
F	+ Large networks (StyleGAN2)	**2.84**
	Config A with large networks	3.98



New architectures of the generator and the discriminator in StyleGAN2.

StyleGAN3: Alias-Free GAN (Karras et al., 2021)

- StyleGAN2 has a "texture sticking" problem: unwanted dependence of the synthesis process on absolute pixel coordinates.
  - Above: an image generated from a latent code (I guess the input of the synthesis network).
  - Below: The average of images generated from a small neighborhood around the same code.
  - The intended result is uniformly blurry because all details should move together. However, with StyleGAN2 many details stick to the same pixel coordinates, showing unwanted sharpness.
- StyleGAN3 fixes this problem:
  - Interprete all signals in the network as continuous
  - Re-design the architecture of the synthesis network to make it fully equivariant to translation and rotation.

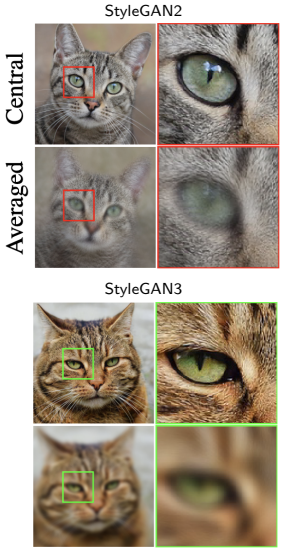
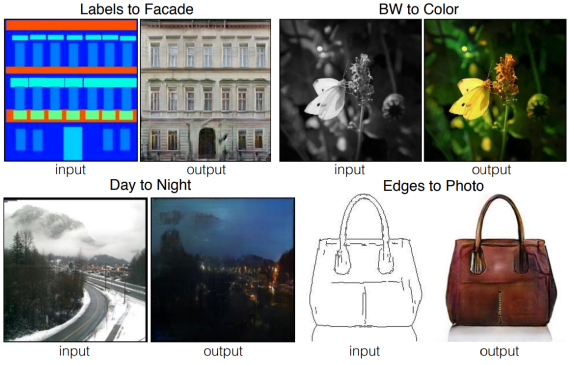


Image-to-Image Translation with Conditional GANs

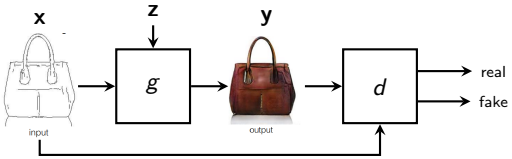
(Isola et al., 2017)

Image-to-image translation task

- In many applications, we need to generate a sample conditioned on some other data.
- Example: image-to-image translation (Isola et al., 2017).
- We need to generate image **y** conditioned on a given image **x**.



Conditional generation with GANs



- The objective of a conditional GAN can be expressed as

$$v(g,d) = \mathbb{E}_{\mathbf{x},\mathbf{y}}[\log d(\mathbf{x},\mathbf{y})] + \mathbb{E}_{\mathbf{x},\mathbf{z}}[\log(1 - d(\mathbf{x}, g(\mathbf{x},\mathbf{z}))]$$

- The discriminator's job remains unchanged.
- The generator is tasked to not only fool the discriminator but also to be near the ground truth output:

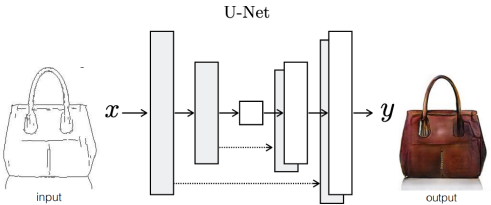
$$\mathcal{L}(g) = v(g,d) + \lambda \mathbb{E}_{\mathbf{x},\mathbf{y},\mathbf{z}} \left\| \mathbf{y} - g(\mathbf{x},\mathbf{z}) \right\|_1$$

  Using L1 results in less blurring compared to L2.

Conditional generation with GANs

- The generator has a U-net architecture.
- The source of noise is the dropout in the intermediate layers (still little stochasticity in the output).
- The generated images look more realistic compared to the model trained only with the L1 loss.

$$\mathcal{L}(g) = v(g, d) + \lambda \mathbb{E}_{\mathbf{x},\mathbf{y},\mathbf{z}} \|\mathbf{y} - g(\mathbf{x}, \mathbf{z})\|_1$$



The structure of the generator

Examples of generated images

Home assignment

Assignment 08_gan

- You need to implement and train on MNIST:
  - DCGAN
  - WGAN with gradient penalty (WGAN-GP)

Recommended reading

- Papers cited in the lecture slides.