A Tutorial: Generative Adversarial Networks

Wonjun Ko

wjko@korea.ac.kr

Machine Intelligence Laboratory,
Department of Brain and Cognitive Engineering,
Korea University

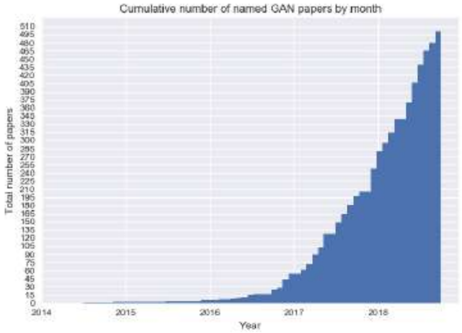
May 2, 2019

Contents

Basic Concepts of Generative Adversarial Networks

Emerging Topic

Generation, Classification/Regression, Adaptation, Augmentation, etc.

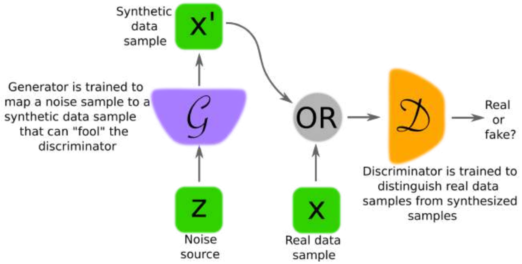


Cumulative number of names GANs papers by month[1]

[1] https://github.com/hindupuravinash/the-gan-zoo

Various Definitions of GANs

- GANs provide a way to learn deep representations without extensively annotated training data [1].
- GANs convert a difficult problem (distribution $\sim$ distribution) to an easy problem (expectation $\sim$ expectation) [2].
- GANs learn a probability distribution [3].
- GANs allow us to synthesize novel data samples from random noise.

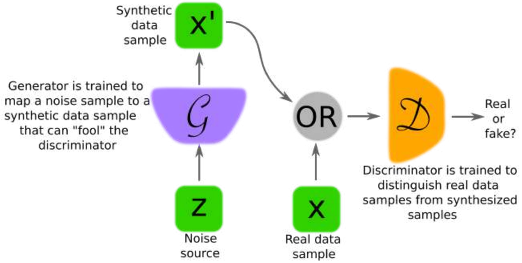


Synthetic data sample
X'
Generator is trained to map a noise sample to a synthetic data sample that can "fool" the discriminator
$\mathcal{G}$
OR
$\mathcal{D}$
Real or fake?
Discriminator is trained to distinguish real data samples from synthesized samples
Z
Noise source
X
Real data sample

A brief architecture of GANs [1]

A Basic Definition of GANs

- The generator $\mathcal{G}$ creates forgeries vs. the discriminator $\mathcal{D}$ aims to tell real and forgeries apart.
- The generator has no direct access to real images, the only way it learns is through its interaction with the discriminator.
- $\mathcal{G} : \mathcal{G}(\mathbf{z}) \rightarrow \mathbb{R}^{|\mathbf{x}|}$ & $\mathcal{D} : \mathcal{D}(\mathbf{x}) \rightarrow (0, 1)$ must be differentiable, not be directly invertible (0 ←fake, and 1 ←real).


Synthetic data sample
X'
Generator is trained to map a noise sample to a synthetic data sample that can "fool" the discriminator
$\mathcal{G}$
OR
$\mathcal{D}$
Real or fake?
Discriminator is trained to distinguish real data samples from synthesized samples
Z
Noise source
X
Real data sample

A brief architecture of GANs [1]

Machine Intelligence Lab

Discriminative vs. Generative Models

A discriminative model

$\boxed{\text{To learn the conditional probability } p(y|x)}$

- To learn a function that maps the input data x to some desired output class label y

A generative model

$\boxed{\text{To learn the joint probability } p(x, y)}$

- To learn both distributions of the input data x and the corresponding label y simultaneously

- The generative model has the potential to understand and explain the underlying structure of the input data even there are no labels.

  $\Rightarrow$ A remarkable benefit when working on *real-world* data modelling problem ($\because$ *unlabelled data*$\gg$*labelled data* in the real world)

Machine Intelligence Lab

Generative Adversarial Networks[2]

- $\mathcal{D}$anielle is a bank teller who discriminates between real money and counterfeit money.
- $\mathcal{G}$eorge is a crook and is trying to make counterfeits.
- The real money $\mathbf{x}$ are randomly sampled from a probability distribution p_{data} which is only known to the Treasury (*i.e.*, *neither Danielle nor George know the function*).
- George's goal $\Rightarrow$ To generate samples $\mathbf{x}'$ from p_{data}
  i.e., The counterfeits are indistinguishable from the real currency.

How can George generate samples from p_{data},
if he doesn't know the true distribution?

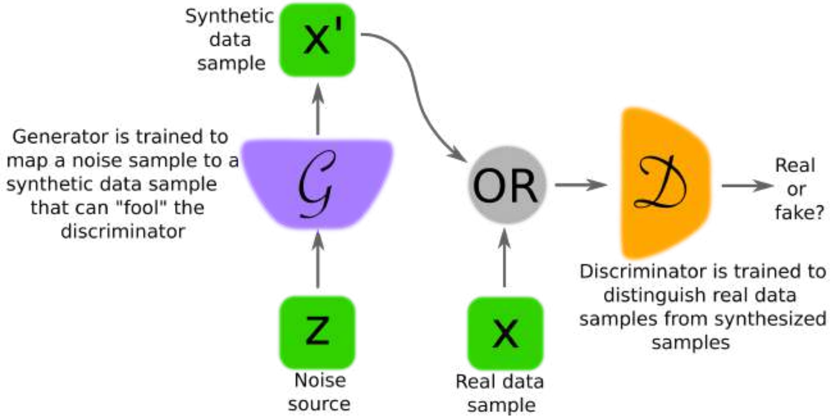
[2] https://blog.evjang.com/2016/06/generative-adversarial-nets-in.html

Generative Adversarial Networks

We can create computationally indistinguishable samples without knowing the true data distribution!

- The true distribution: a method that the Treasury itself is using to generate the real currency **i.e.,** Some efficient distribution for sampling p_{data}
- We can think the efficient distribution as a *natural basis*.
- George can express the same sampling algorithm in bases (*e.g.*, a neural network basis, a Fourier basis, *etc.*) which can be used to a *universal approximator*.

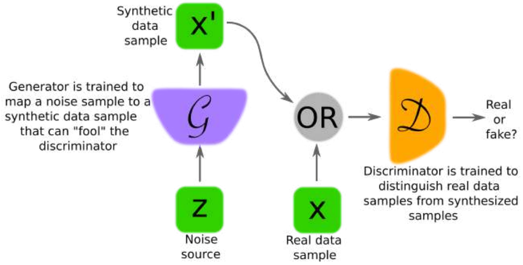
Generative Adversarial Networks



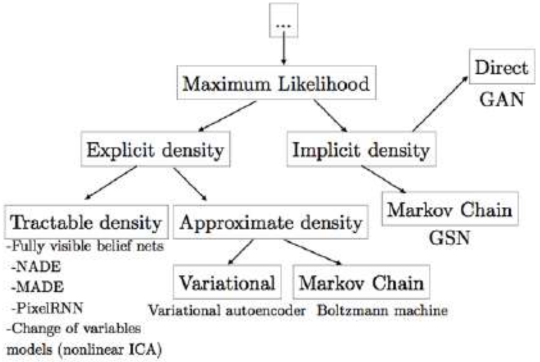
Synthetic data sample
X'
Generator is trained to map a noise sample to a synthetic data sample that can "fool" the discriminator
$\mathcal{G}$
OR
$\mathcal{D}$
Real or fake?
Discriminator is trained to distinguish real data samples from synthesized samples
Z
Noise source
X
Real data sample

A brief architecture of GANs [1]

A implicit generative model


Synthetic data sample
X'
Generator is trained to map a noise sample to a synthetic data sample that can "fool" the discriminator
$\mathcal{G}$
OR
$\mathcal{D}$
Real or fake?
Discriminator is trained to distinguish real data samples from synthesized samples
Z
Noise source
X
Real data sample

A brief architecture of GANs [1]



Taxonomy of generative models [4]

Generator Network

$$\mathbf{x}' = \mathcal{G}(\mathbf{z}; \theta^{(\mathcal{G})})$$

- Generator tries to generate *real-like* samples.
  i.e., $\mathcal{G}(\mathbf{z}; \theta^{(\mathcal{G})}) = \mathbf{x}' \sim p_{\text{data}}$
- Generator MUST be differentiable.
- Generator has NO requirement of invertibility.
- Generator SHOULD be trainable for any size of $\mathbf{z}$.
- $\mathbf{z}$ is a latent code vector.
  - Theoretically, a uniform distribution or even a scalar value can be used for the latent dimension.
  - In practice, a normal (*i.e.*, Gaussian) distribution works well for GANs.
  - Importantly, we can make the latent vector $\mathbf{z}$ conditionally [5, 6].

Machine Intelligence Lab

Discriminator Network

$$\mathcal{D}(\mathbf{x};\theta^{(\mathcal{D})}) \quad \& \quad \mathcal{D}(\mathbf{x}';\theta^{(\mathcal{D})})$$

- Discriminator outputs the probability that shows the input is real.
- Discriminator tries to discriminate the real and generated samples.
  i.e., $\mathcal{D}(\mathbf{x};\theta^{(\mathcal{D})}) = 1$ & $\mathcal{D}(\mathbf{x}';\theta^{(\mathcal{D})}) = 0$
- Discriminator MUST be also differentiable, not be directly invertible.

Min-Max Objective Function

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D})$$

where $V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{\text{data}(\mathbf{x})}}[\log \mathcal{D}(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z})))]$

- $\mathcal{G}$ minimizes when $\mathcal{D}(\mathcal{G}(\mathbf{z})) \to 1$, *i.e.*, $\mathcal{G}$ makes $\mathcal{D}$ exactly fool.
- $\mathcal{D}$ maximizes $\mathcal{D}(\mathbf{x}) \to 1$, and $\mathcal{D}(\mathcal{G}(\mathbf{z})) \to 0$, *i.e.*, $\mathcal{D}$ decides real or fake exactly correct.

Machine Intelligence Lab

Min-Max Objective Function

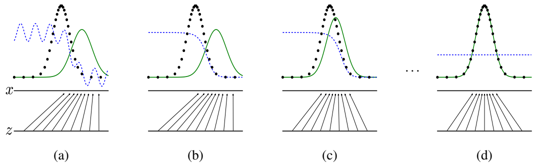
$$V^{(\mathcal{D})} = -\frac{1}{2}\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\log \mathcal{D}(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{p_z(\mathbf{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z})))]$$
$$V^{(\mathcal{G})} = -V^{(\mathcal{D})}$$

- Equilibrium is a saddle point of the discriminator loss [7].
- Generator minimizes the log-probability of the discriminator being correct [7].

Optimal Solution for the Objective Function

- To get the solution, we should assume that both densities p_{data} and p_z are nonzero everywhere.
  i.e., $\forall (a,b) \in \mathbb{R}^2 \backslash (0,0)$, where $a \sim p_{\text{data}}$ and $b \sim p_z$.
  - If not, some input values are never trained, so some values of $\mathcal{D}$ have underdetermined behavior [4].

- Thus, for $V(\mathcal{G},\mathcal{D}) = \int d\mathbf{x} \;\; p_{\text{data}} \log \mathcal{D}(\mathbf{x}) + p_z \log(1-\mathcal{D}(\mathcal{G}(\mathbf{z})))$,
  the optimal $\mathcal{D}$ is $\mathcal{D}* = \dfrac{p_{\text{data}}}{p_{\text{data}} + p_z}$.
  $\because$ The function $y = a\log y + b\log(1-y)$ achieves its maximum in $[0,1]$ at $\frac{a}{a+b}$, where $\forall (a,b) \in \mathbb{R}^2 \backslash (0,0)$.



Distributions of the $\mathcal{D}$ (blue), $\mathcal{G}$ (green), and p_{data} (black) [7]

Optimal Solution for the Objective Function

- When $\mathcal{D} = \mathcal{D}^*_{\mathcal{G}}$,

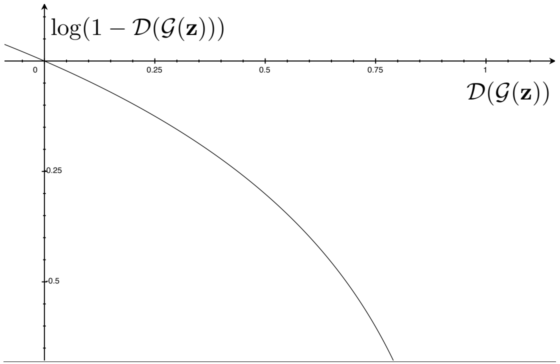
$$\mathbb{E}_{p_{\text{data}(\mathbf{x})}}\left[\log \frac{p_{\text{data}}}{p_{\text{data}} + p_z}\right] + \mathbb{E}_{p_z(\mathbf{z})}\left[\log \frac{p_z}{p_{\text{data}} + p_z}\right]$$
$$\Longrightarrow \text{const.} + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_z}{2}\right) + KL\left(p_z \middle\| \frac{p_{\text{data}} + p_z}{2}\right)$$
$$= \text{const.} + 2 \cdot JS(p_{\text{data}} || p_z).$$

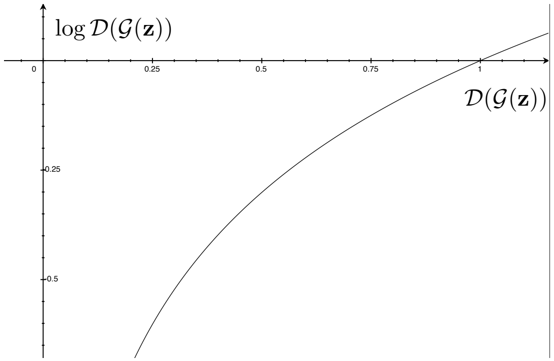
$\therefore$ In the implementation level, we use *binary cross entropy* function for the objective function.

FYI There is no need to use JS divergence only [2]!

- *"The other GANs training approach using variational divergence estimation is a special case of f-divergence approach [2]."*

Practical Min-Max Objective Function





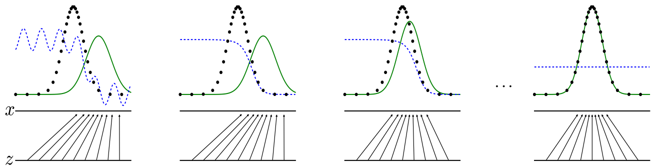
- In practice, we use $\max_{\mathcal{G}} \mathbb{E}_{p_z(\mathbf{z})}[\log \mathcal{D}(\mathcal{G}(\mathbf{z}))] = \min_{\mathcal{G}} \mathbb{E}_{p_z(\mathbf{z})}[(-\log \mathcal{D}(\mathcal{G}(\mathbf{z})))]$ instead of $\min_{\mathcal{G}} \mathbb{E}_{p_z(\mathbf{z})}[\log(1-\mathcal{D}(\mathcal{G}(\mathbf{z})))]$.

  $\Rightarrow$ $-\log\mathcal{D}$ trick!

Implementation - Fully Connected GANs

- First proposed GANs by Goodfellow *et al.* [7]
- First proposal & proof of the *mini-max* objective function & $\exists$ optimal solution respectively

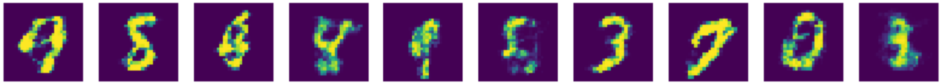
$$\mathbb{E}_{p_{\text{data}(\mathbf{x})}}[\log \mathcal{D}(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z})))]$$



Implementation - Fully Connected GANs

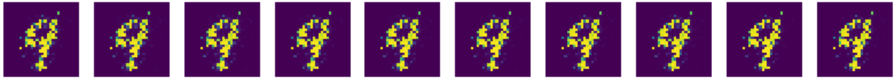
- All codes are basically based on `https://tensorflow.org/tutorials` and `https://github.com/golbin/TensorFlow-Tutorials`, and modified by Ko.

- `https://colab.research.google.com/drive/1SQltkrxxkhDErQj48fJ2qF-f8R183-Ze`

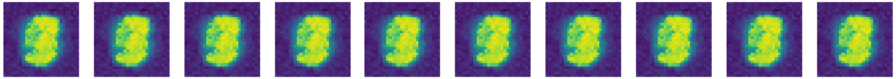


Implementation - D.I.Y.

- Use a uniform distribution (`np.random.uniform`)
  instead of a normal distribution (`np.random.normal`) for the latent vector



- Use $\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z})))$ (`tf.log(1 - D_counterfeits)`)
  instead of $-\log \mathcal{D}(\mathcal{G}(\mathbf{z}))$ (`tf.log(D_counterfeits))`)



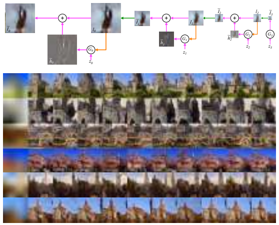
Variants of Generative Adversarial Networks I

Recap.

- Neural networks in GANs have some constraints [4].
  1. Both generator and discriminator MUST be differentiable.
  2. Generator SHOULD be trainable for any size of $\mathbf{z}$.

$\therefore$ We need NOT use fully-connected layers only for the neural networks architecture in GANs.

- Many researches tried to use convolutional neural network (CNN), which is more difficult to train that fully-connected layers, for the generator or/and discriminator [8, 9].

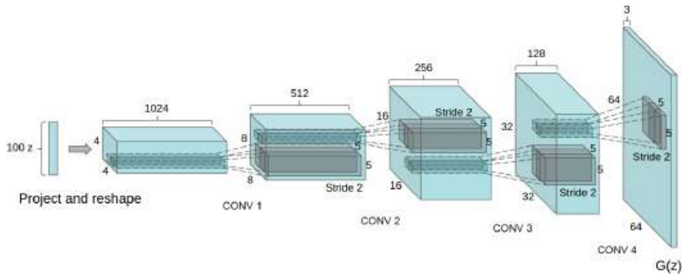


A brief framework and generated samples of LAPGANs [8]

Generated samples of DCGANs [9]

DCGANs

- Solving the GANs optimization, *i.e.*, solving minmax, saddle point, is inherently unstable [4].
- *"Most GANs today are at least loosely based on the DCGAN architecture."* - Ian Goodfellow
- For the generator, the authors used *fractionally-strided convolutions* (deconvolutions) to convert the random noise $\mathbf{z}$ to the high level represetation (*e.g.*, 64×64×3 pixel image).
- No fully-connected or pooling layers are used.



DCGAN generator architecture used for LSUN scene modeling [9]

DCGANs

- The authors explored model architecture by *extensively research and testing* to make the GANs robust [9].
- Some architecture guidelines for stable deep convolutional GANs [9]:
  - Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
  - Use batch normalization in both the generator and the discriminator.

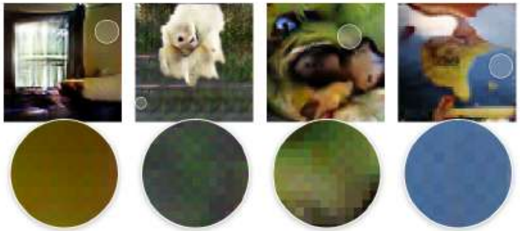
    FYI Do NOT applying batch normalization to the generator output layer and the discriminator input layer!

    $\because$ Sample oscillation and model instability
  - Remove fully-connected hidden layers for deeper architectures.
  - Use ReLU activation in generator for all layers except for the output, which uses $\tanh$.
  - Use leaky ReLU activation in the discriminator for all layers.

Implementation - DCGANs

- For the implementation, we use CNN (`tf.nn.conv2d`) for the discriminator, and use de-CNN (`tf.nn.conv2d_transpose`) for the generator.

FYI Deconvolution might make checkerboard artifacts[3].



- To avoide these checkerboard artifacts, we can use NN-resize convolutions or bilinear-resize convolutions instead of deconvolutions [10].

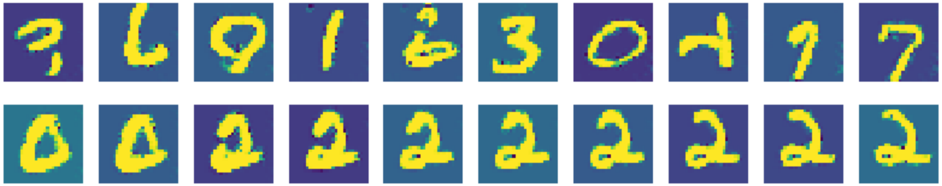
i.e., Use `tf.image.resize_images`→`tf.pad`→`tf.nn.conv2d` instead of `tf.nn.conv2d_transpose`

[3]https://distill.pub/2016/deconv-checkerboard/

Implementation - DCGANs

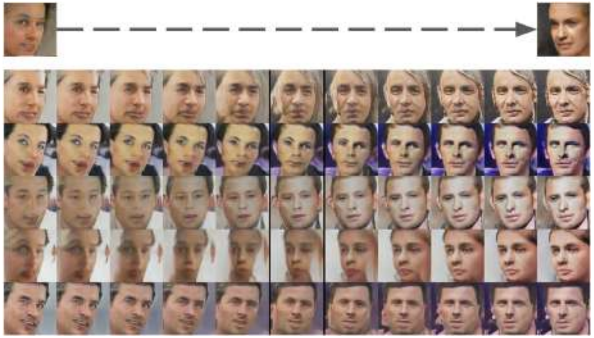
- All codes are basically based on https://tensorflow.org/tutorials and https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/dcgan.py and modified by Ko.

- https://colab.research.google.com/drive/1OUFJGzwCnybM-3poOLpXORoYSTPCNsdA



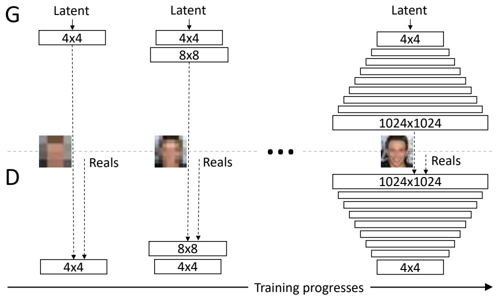
Implementation - D.I.Y.

- In research about generative models, we should check that [9]:
  1. the generator do NOT memorize the sample images,
     - Memorization does NOT mean that the generator learns meaningful features, but learns the mapping of 1:1 matching because of *overfitting*.
  2. When we *walk in the latent space*, the generated samples should show smooth translation, NOT sharp translation.


Samples from interpolated random noise [9]

Recent Achievement of Convolutional GANs

- In 2018, Karras *et al.* [11] proposed a novel approach for the convolutional GANs.



A novel approach used in [11] and its results

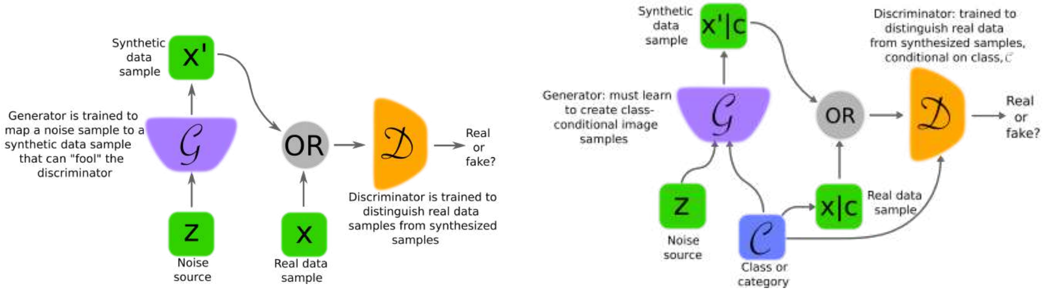
Recap.

- GANs are basically designed for *unsupervised learning* [7], however, thanks to some properties of the random noise $\mathbf{z}$, it is straightforward to apply *supervised learning* to the objective function.
  - We can use conditioned random noise vector for GANs [5, 4].

$$\mathbf{x}' = \mathcal{G}(\mathbf{z}; \theta^{(\mathcal{G})}) \Longrightarrow \mathbf{x}' = \mathcal{G}(\mathbf{z}|\mathbf{c}; \theta^{(\mathcal{G})})$$
$$\mathcal{D}(\mathbf{x}; \theta^{(\mathcal{D})}) \;\&\; \mathcal{D}(\mathbf{x}'; \theta^{(\mathcal{D})}) \Longrightarrow \mathcal{D}(\mathbf{x}|\mathbf{c}; \theta^{(\mathcal{D})}) \;\&\; \mathcal{D}(\mathbf{x}|\mathbf{c}; \theta^{(\mathcal{D})})$$

- We can give a condition vector to the random noise vector.
  - Making both generator and discriminator *class-conditional* [5]

Conditional-GANs


Synthetic data sample
X'
Generator is trained to map a noise sample to a synthetic data sample that can "fool" the discriminator
$\mathcal{G}$
Z
Noise source
OR
X
Real data sample
$\mathcal{D}$
Real or fake?
Discriminator is trained to distinguish real data samples from synthesized samples
Synthetic data sample
X'|C
Generator: must learn to create class-conditional image samples
$\mathcal{G}$
Z
Noise source
$\mathcal{C}$
Class or category
X|C
Real data sample
OR
Discriminator: trained to distinguish real data from synthesized samples, conditional on class, $\mathcal{C}$
$\mathcal{D}$
Real or fake?

A brief architecture of GANs (left) and Conditional-GANs (right) [1]

Conditional-GANs Objective Function

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D})$$

where $V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{\mathsf{data}(\mathbf{x})}}[\log \mathcal{D}(\mathbf{x}|\mathbf{c})] + \mathbb{E}_{p_z(\mathbf{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z}|\mathbf{c})))]$

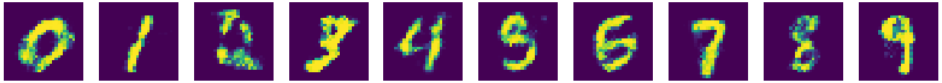
- $\mathcal{G}$ minimizes when $\mathcal{D}(\mathcal{G}(\mathbf{z}|\mathbf{c})) \to 1$, *i.e.*, $\mathcal{G}$ makes $\mathcal{D}$ fool for given condition $\mathbf{c}$.
- $\mathcal{D}$ maximizes $\mathcal{D}(\mathbf{x}|\mathbf{c}) \to 1$, and $\mathcal{D}(\mathcal{G}(\mathbf{z}|\mathbf{c})) \to 0$, *i.e.*, $\mathcal{D}$ tries to decide real data samples and conditionally generated samples.

Implementation - Conditional-GANs

- To embed a condition vector to generator and discriminator in fully-connected GANs [7], we can concatenate (`tf.concat`) the condition vector to input random noise or flattented input image.
- To embed a condtion vector to *convolutional GANs*, it is harder than fully-connected GANs case, but we still can.
  - For the generator, we can keep the concatenation, *i.e.*, concatenation between the condition vector and the random noise.
  - For the discriminator, we concat the condition vector to a convolved feature, *i.e.*, extracted (intermediate) feature from CNN.

Implementation - Conditional-GANs

- All codes are basically based on `https://tensorflow.org/tutorials` and `https://github.com/golbin/TensorFlow-Tutorials`, and modified by Ko.
- `https://colab.research.google.com/drive/1b-9YjH4cYnndE7E1PaD91-p2H-VFjnTA`



Machine Intelligence Lab

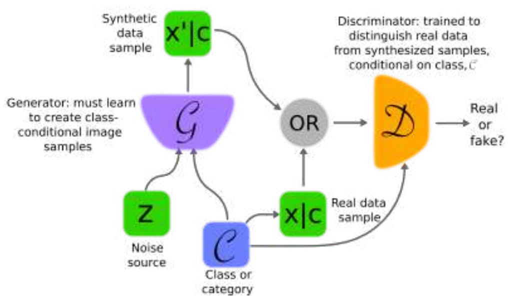
Recap.

- GANs are basically designed for *unsupervised learning* [7], however, thanks to some properties of the random noise $\mathbf{z}$, it is straightforward to apply *supervised learning* to the objective function.
    - We can any form of random noise $\mathbf{z}$ for GANs [6, 4].

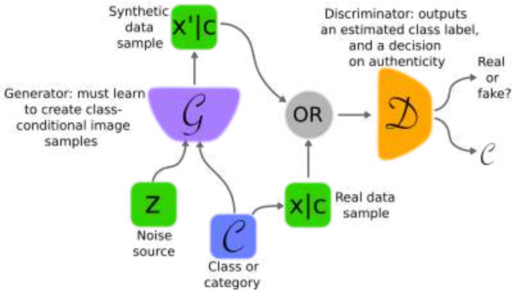
$$\mathbf{x}' = \mathcal{G}(\mathbf{z}; \theta^{(\mathcal{G})}) \Longrightarrow \mathbf{x}' = \mathcal{G}(\mathbf{z}, \mathbf{c}; \theta^{(\mathcal{G})})$$
$$\mathcal{D}(\mathbf{x}; \theta^{(\mathcal{D})}) \ \& \ \mathcal{D}(\mathbf{x}'; \theta^{(\mathcal{D})})$$

- We can decompose the input noise vector into two parts:
    i) $\mathbf{z}$ a source of incompressible noise
    ii) $\mathbf{c}$ a latent code that target the salient structured semantic features of the data distribution
    - Maximizing mutual information between a latent code and a generated sample [6]

InfoGANs



Synthetic data sample
x'|c
Discriminator: trained to distinguish real data from synthesized samples, conditional on class, $\mathcal{C}$
Generator: must learn to create class-conditional image samples
$\mathcal{G}$
OR
$\mathcal{D}$
Real or fake?
Z
Noise source
$\mathcal{C}$
Class or category
x|c
Real data sample



Synthetic data sample
x'|c
Discriminator: outputs an estimated class label, and a decision on authenticity
Real or fake?
Generator: must learn to create class-conditional image samples
$\mathcal{G}$
OR
$\mathcal{D}$
$\mathcal{C}$
Z
Noise source
$\mathcal{C}$
Class or category
x|c
Real data sample

A brief architecture of Conditional-GANs (left) and Info-GANs (right) [1]

InfoGANs Objective Function

- When we take the form of the generator $\mathcal{G}(\mathbf{z}, \mathbf{c})$, the generator is free to ignore the additional latent code $\mathbf{c}$ by finding a solution satisfying $P_{\mathcal{G}}(\mathcal{G}(\mathbf{z}, \mathbf{c})) = P_{\mathcal{G}}(\mathcal{G}(\mathbf{z}))$ in standard GANs objective fuction [6].
- To cope the problem, the authors proposed an *information-theoritic regularization*: there should be high mutural information[4] between the latent code and generated samples, *i.e.*, $I(\mathbf{c}; \mathcal{G}(\mathbf{z}, \mathbf{c}))$ should be high.

[4] $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$

InfoGANs Objective Function

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D})$$

where $V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{\text{data}(\mathbf{x})}}[\log \mathcal{D}(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z}, \mathbf{c})))] - \lambda I(\mathbf{c}; \mathcal{G}(\mathbf{z}, \mathbf{c}))$

- $\mathcal{G}$ minimizes when $\mathcal{D}(\mathcal{G}(\mathbf{z}, \mathbf{c})) \to 1$, *i.e.*, $\mathcal{G}$ makes $\mathcal{D}$ fool for given latent code $\mathbf{c}$.
- $\mathcal{G}$ also minimizes when $I(\mathbf{c}; \mathcal{G}(\mathbf{z}, \mathbf{c}))$ is high, *i.e.*, $\mathcal{G}$ tries to keep the mutual information with the latent code high.
- $\mathcal{D}$ maximizes $\mathcal{D}(\mathbf{x}) \to 1$, and $\mathcal{D}(\mathcal{G}(\mathbf{z}, \mathbf{c})) \to 0$, *i.e.*, $\mathcal{D}$ tries to decide real data samples and generated samples.

InfoGANs Objective Function

- In practice, the mutual information term $I(\mathbf{c}; \mathcal{G}(\mathbf{z}, \mathbf{c}))$ is hard to maximize directly as it requires access to the posterior $P(\mathbf{c}|\mathcal{G}(\mathbf{z}, \mathbf{c}))$ [6], thus the authors obtain an auxiliary distribution $Q(\mathbf{c}|\mathcal{G}(\mathbf{z}, \mathbf{c}))$ to approximate $P(\mathbf{c}|\mathcal{G}(\mathbf{z}, \mathbf{c}))$:

$$\begin{aligned} I(\mathbf{c}; \mathcal{G}(\mathbf{z}, \mathbf{c})) &= H(\mathbf{c}) - H(\mathbf{c}|\mathcal{G}(\mathbf{z}, \mathbf{c})) \\ &= \mathbb{E}_{\mathbf{g}\sim\mathcal{G}(\mathbf{z},\mathbf{c})}[\mathbb{E}_{\mathbf{c}'\sim P(\mathbf{c}|\mathbf{g})}[\log P(\mathbf{c}'|\mathbf{g})]] + H(\mathbf{c}) \\ &= \mathbb{E}_{\mathbf{g}\sim\mathcal{G}(\mathbf{z},\mathbf{c})}[\underbrace{KL(P(\cdot|\mathbf{g})||Q(\cdot|\mathbf{g}))}_{\geq 0} + \mathbb{E}_{\mathbf{c}'\sim P(\mathbf{c}|\mathbf{g})}[\log Q(\mathbf{c}'|\mathbf{g})]] + H(\mathbf{c}) \\ &\geq \mathbb{E}_{\mathbf{g}\sim\mathcal{G}(\mathbf{z},\mathbf{c})}[\mathbb{E}_{\mathbf{c}'\sim P(\mathbf{c}|\mathbf{g})}[\log Q(\mathbf{c}'|\mathbf{g})]] + H(\mathbf{c}) \end{aligned}$$

- Thus we now use,

$$\min_{\mathcal{G},Q} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}, Q)$$

where $V(\mathcal{G}, \mathcal{D}, Q) = \mathbb{E}_{p_{\text{data}(\mathbf{x})}}[\log \mathcal{D}(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z}, \mathbf{c})))] - \lambda\{\mathbb{E}_{\mathbf{g}\sim\mathcal{G}(\mathbf{z},\mathbf{c})}[\mathbb{E}_{\mathbf{c}'\sim P(\mathbf{c}|\mathbf{g})}[\log Q(\mathbf{c}'|\mathbf{g})]] + H(\mathbf{c})\}$.

Machine Intelligence Lab

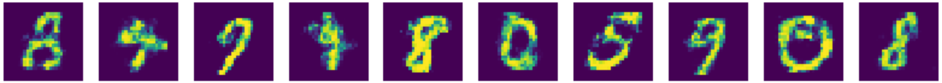
Implementation - InfoGANs

- In practice, the authors parametrize the auxiliary distribution Q as a neural network [6].
- In most experiments, Q and D share all convolutional layers and there is one final fully-connected layer to output parameters for the conditional distribution $Q(\mathbf{c}|\mathbf{x})$ [6].
- To disentangle digit shape from styles on MNIST, we choose to model the latent codes with a categorical distribution [6], $c_1 \sim \text{Cat}(K = 10, p = 0.1)$ (`np.random.multinomial`).
- `To capture variations of the datset, we use uniform distributions [6],` $c_2, c_3 \sim \texttt{Unif}(-1, 1)$ (`np.random.uniform`).

Implementation - InfoGANs

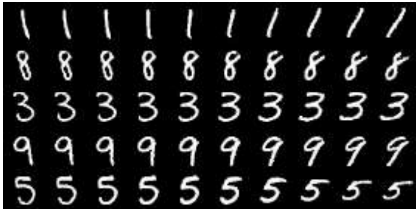
- All codes are basically based on the original paper setting [6] and https://github.com/wiseodd/generative-models/blob/master/GAN/infogan/infogan_tensorflow.py, and modified by Ko.

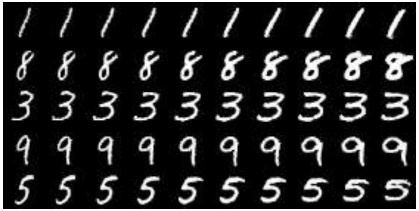
- https://colab.research.google.com/drive/1gsJWiEruWDArWOuA3K7Do4aXo3s4GPkI



Implementation - D.I.Y.

- For the InfoGANs, additional latent codes are allowed.
- In the paper, the authors did not only use a categorical distribution (for categorical code), but also used additional two uniform distributions to control the rotation and width of the MNIST respectively.

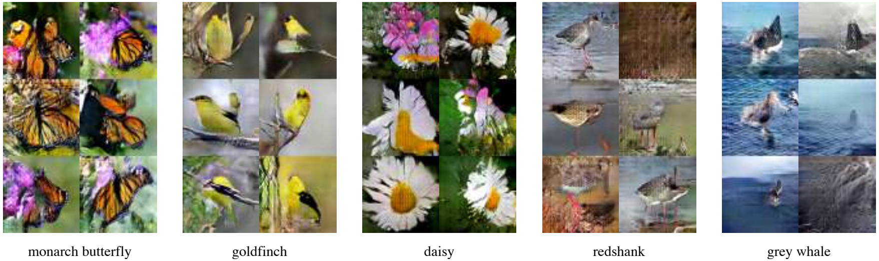

(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)


(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

InfoGANs results from varying latent codes [6]

Auxiliary Classifier GANs

- The authors contributed that [?]:
  - Synthesizing (relatively) high resolution (128×128) image from all 1000 ImageNet classes
  - Measuring how much an image synthesis model actually uses its output resolution
  - Measuring perceptual variability and collapsing behavior.



Results of AC-GANs [?] on the ImageNet dataset

Auxiliary Classifier GANs Objective Function

$$\arg\max_{\mathcal{G}}[\mathcal{L}_C - \mathcal{L}_S] \;\&\; \arg\max_{\mathcal{D}}[\mathcal{L}_C + \mathcal{L}_S]$$

$$\text{where } \mathcal{L}_C = \mathbb{E}[\log P(C = c|\mathbf{X}_{\text{real}})] + \mathbb{E}[\log P(C = c|\mathbf{X}_{\text{fake}})]$$

$$\mathcal{L}_S = \mathbb{E}[\log P(S = \text{real}|\mathbf{X}_{\text{real}})] + \mathbb{E}[\log P(S = \text{fake}|\mathbf{X}_{\text{fake}})]$$

- In the AC-GANs, every generated sample has a corresponding class label, $c \sim p_c$ in addition to the random noise.
- The generator uses both to generate images $\mathbf{X}_{\text{fake}} = \mathcal{G}(c, \mathbf{z})$.
- The discriminator gives both a probability distribution over sources and a probability distribution over the class labels, $P(S|\mathbf{X})$ and $P(C|\mathbf{X})$.
- AC-GANs learn a representation for $\mathbf{z}$ that is independent of class label [?].

Machine Intelligence Lab

Discriminative Model in GANs

- For the past five years, most of the research interests in GANs has been focused on generative models.
- The adversarial mechanism of GANs is straightforward to apply to semi-supervised learning.
- Some researches focused on the semi-supervised learning with the discriminative model in GANs.

SGANs Objective Function

$$\min_{\mathcal{G},\mathcal{D}}\{\mathcal{L}_{\text{supervised}} + \mathcal{L}_{\text{unsupervused}}\}$$

$$\text{where } \mathcal{L}_{\text{supervised}} = -\mathbb{E}_{p_{\text{data}}(\mathbf{x},\mathbf{y})} \log p_{\text{model}}(\mathbf{y}|\mathbf{x}, \mathbf{y} < K+1),$$

$$\mathcal{L}_{\text{unsupervised}} = -\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\log \mathcal{D}(\mathbf{x})] - \mathbb{E}_{p_z(\mathbf{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z})))],$$

$$p_{\text{model}}(\mathbf{y} = j|\mathbf{x}) = \frac{\exp(l_j)}{\sum_{k=1}^{K} \exp(l_k)}, \ K \text{ is the number of class}$$

- The additional term $\mathcal{L}_{\text{supervised}}$ means a *cross-entropy* loss function for the supervised learning.

Implementation - SGANs

- From a practical standpoint, we use an additional *feature matching* technique [12] for stable training.
- Therefore, we use $\mathcal{L}_{\text{supervised}} + \mathcal{L}_{\text{unsupervised}}$ for the discriminator loss.
- And we add an additional term $\mathcal{L}_{\text{feature matching}}$ for the generator loss,

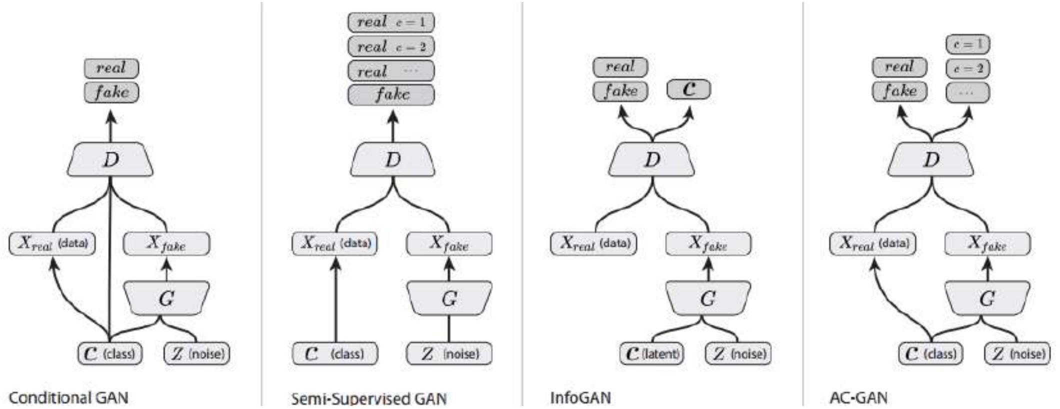
$$\mathcal{L}_{\text{feature matching}} = ||\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\mathbf{f}(\mathbf{x})] - \mathbb{E}_{p_z(\mathbf{z})}[\mathbf{f}(\mathcal{G}(\mathbf{z}))]||_2^2$$

then we use $\mathcal{L}_{\text{unsupervised}} + \mathcal{L}_{\text{feature matching}}$ for the generator loss.

Implementation - SGANs

- All codes are basically based on `https://github.com/nejlag/Semi-Supervised-Learning-GAN`, and modified by Ko.
- `https://colab.research.google.com/drive/148OZKSWa0icLgpi7RQItNvATcp6euLj1`

A Brief Review for C-, S, Info, AC-GANs



An illustration for various model architectures[5]

[5]https://github.com/buriburisuri/ac-gan

Variants of Generative Adversarial Networks II

Various Use of GANs

- Recent studies about GANs are now focused on various use of GANs, rather than generative model or discriminative model.
- For instance, GANs can translate or/and transfer images using a cyclic property [13, 14, 15].
- GANs can be used for domain adpatation using adversarial learning strategy [16, 17, 18].
- Furthermore, GANs improve image resolution [19].

Pix2Pix

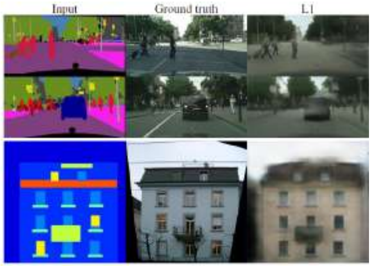
- We can translate grey-scale images to RGB-scale images using CNN easily.

- We can mine large RGB-image data and make these to grey-scale, thus we can get big paired grey-RGB image dataset.
- Thus, we can think image-to-image translation is easy, if we have large paired image dataset.

Machine Intelligence Lab

Pix2Pix

- The authors [13] thought that CNN-based method (with tricks) can work for image-to-image translation.



- As a result, it was failed because pixel-error loss function.

$$\mathcal{L} = \mathbb{E}_{x \in \text{every pixel}}[\mathrm{GT}(x) - \mathrm{Pred}(x)]$$

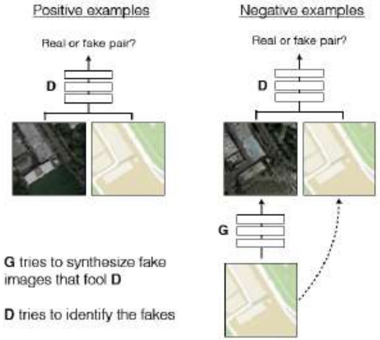
- Because of the expectation, network did not predict photo-realistic value but similar one in average.

Pix2Pix Objective Function

- Because of promising results from GANs, the authors add an adversarial loss for image-to-image translation.

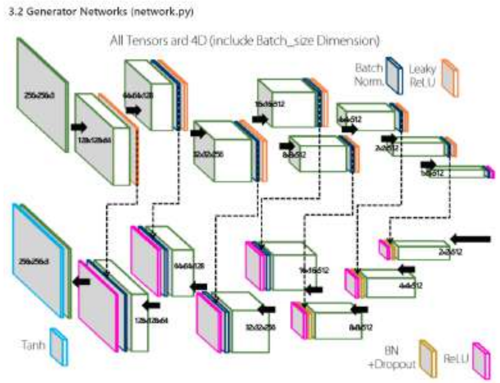
$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D})$$

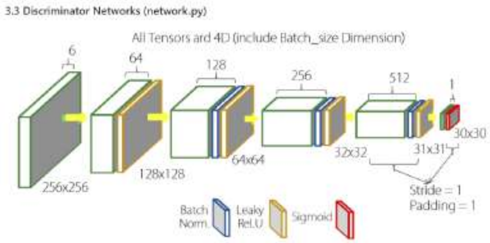
where $V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\log \mathcal{D}(\mathbf{x})] + \mathbb{E}_{p_{\text{data}'}(\mathbf{x}')}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{x}')))] + \mathbb{E}_{p_{\text{data}}(\mathbf{x}), p_{\text{data}'}(\mathbf{x}')}[||\mathbf{x} - \mathcal{G}(\mathbf{x}')||_1]$


Positive examples
Negative examples
Real or fake pair?
Real or fake pair?
D
D
G
G tries to synthesize fake images that fool D
D tries to identify the fakes

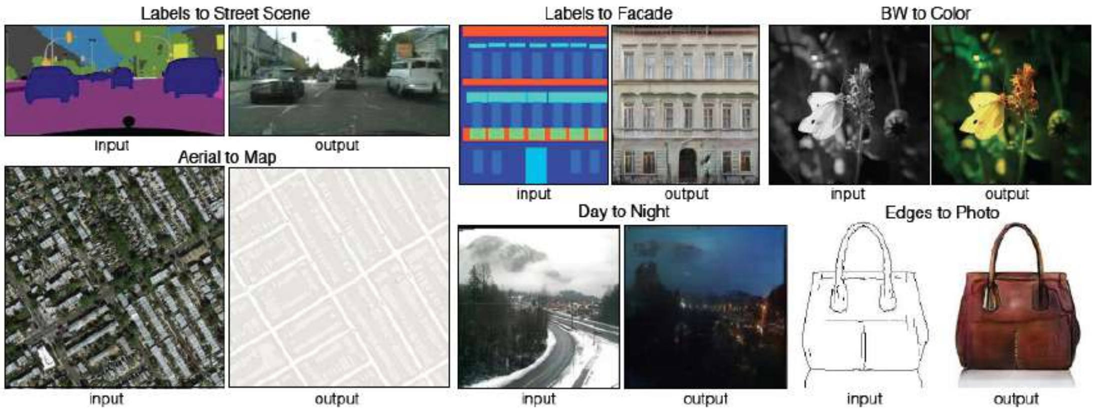
Implementation - Pix2Pix

- The authors used U-Net for the generator and PatchGANs discriminator for the discriminator.





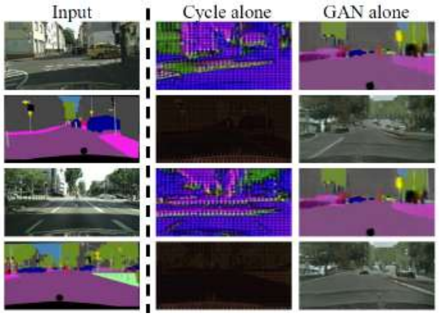
Pix2Pix


Labels to Street Scene
input
output
Aerial to Map
input
output
Labels to Facade
input
output
BW to Color
input
output
Day to Night
input
output
Edges to Photo
input
output

CycleGANs

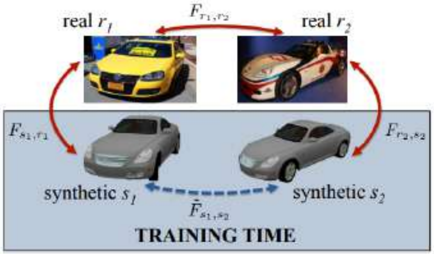
- The authors of Pix2Pix [13] have suceeded ‘paired’ image-to-image translation, thus they focused on ‘unpaired’ image-to-image translation (obviously, we do NOT have any paired Monet’s picture and photo dataset).

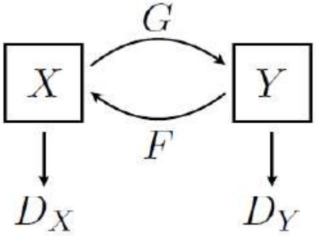
- When we map some data to other data, we need not only focus on a *simplex* mapping but also a *full duplex* mapping [14].
  - When we use a simplex mapping, the mapping function did not capture input style and property.



CycleGANs

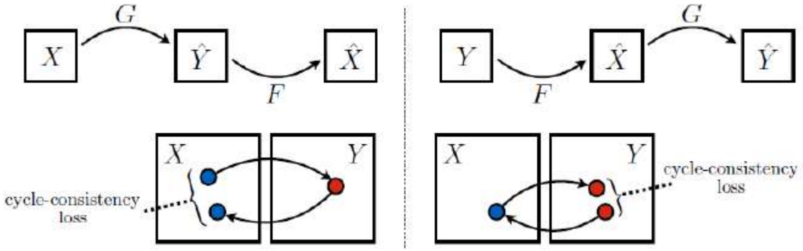
- Therefore to get a meaningfull full duplex mapping, we can implement it using a *cycle consistency*.





CycleGANs Objective Function

- For a cycle consistency, we first define label to image mapping (source to target) and also define image to label mapping (target to source).
- By doing this, when we map to Y, we just check that the transfered (generated) image looks like Y domain and constraint to keep properties using a *backward mapping*.
  - $\because$ We only have X dataset.



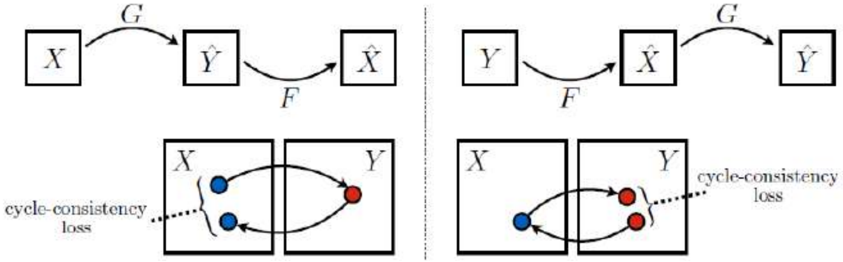
CycleGANs Objective Function

$$\min_{\mathcal{G},\mathcal{F}} \max_{\mathcal{D}} V(\mathcal{G},\mathcal{F},\mathcal{D})$$

where $V(\mathcal{G},\mathcal{D}) = \mathcal{L}_{x\rightarrow y} + \mathcal{L}_{y\rightarrow x}$

$$\mathcal{L}_{x\rightarrow y} = \mathbb{E}_y[\log \mathcal{D}(y)] + \mathbb{E}_x[\log(1-\mathcal{D}(\mathcal{G}(x)))] + \mathbb{E}_x[||\mathcal{F}(\mathcal{G}(x)) - x||_1]$$

$$\mathcal{L}_{y\rightarrow x} = \mathbb{E}_x[\log \mathcal{D}(x)] + \mathbb{E}_y[\log(1-\mathcal{D}(\mathcal{F}(y)))] + \mathbb{E}_y[||\mathcal{G}(\mathcal{F}(y)) - y||_1]$$



Implementation - CycleGANs

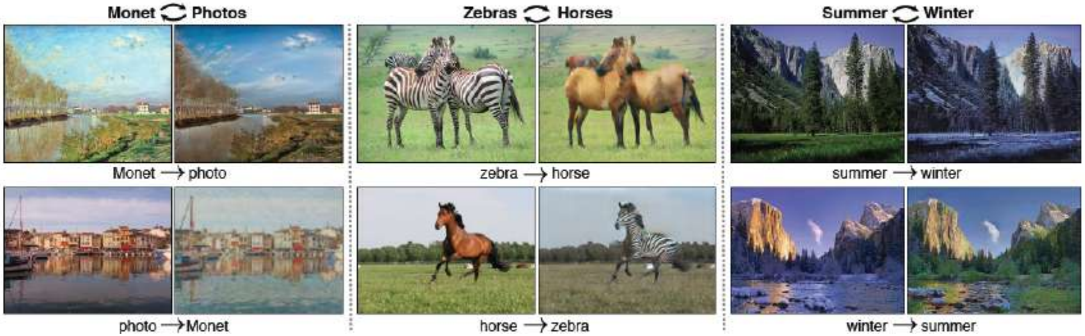
- To get a stabel training, the authors used DCGANs [9] using ResNet structure for generator and PatchGANs discriminator for discriminator.
- They also used Least Square GANs objective function [20] for $\mathcal{L}_{x \to y}$ and $\mathcal{L}_{y \to x}$, for instance,

$$\mathcal{L}_{x \to y} = \mathbb{E}_y[(\mathcal{D}(y) - 1)^2] + \mathbb{E}_x[(\mathcal{D}(\mathcal{G}(x)))^2].$$

TIP The original Jensen-Shannon GANs objective function can cause a *vanishing gradient* problem, and when the vanishing gradient is caused, generator cannot get a meaningful gradient feedback [20].

$\therefore$ We use an MSE to avoid the problem [20].

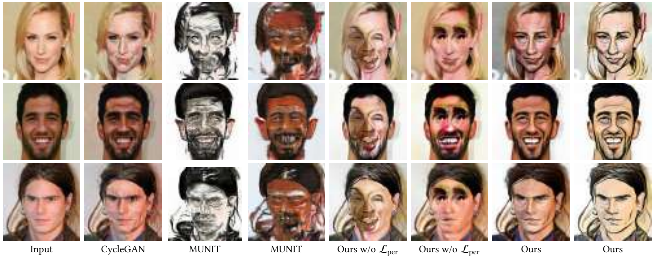
CycleGANs

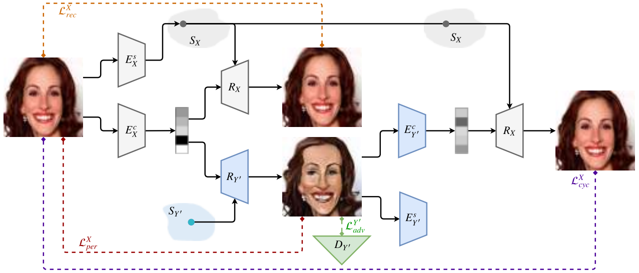


- CycleGANs make that Pix2Pix can work in unpaired dataset using a cycle consistency.
- To get stable training and high-resolution image, CycleGANs use DCGANs architecuture with ResNet, PatchGANs architecture, LSGANs objective function.
- Because of the constraint, it is hard that changing shape, and training procedure gets very long time.

Machine Intelligence Lab

Recent Acheivement of Style Transfer





DiscoGANs

- DiscoGANs [15] focused on both style transfer and shape transfer, while CycleGANs [14] have focused only on a style transfer.

- DiscoGANs [15] used simpler (shallower) network architecture than CycleGANs.
  - ⇒ When the simpler network is used, training can be harder and image resolution can be lower, but getting shape transfer is easier.

- Thus, DiscoGANs used exactly same concepts with CycleGANs, excepts for network architectures (generator: ResNet vs. a simple encoder-decoder, discriminator: PatchGANs vs. DCGANs) and a reconstruction objective funciton (L2 loss).

$$\min_{\mathcal{G},\mathcal{F}} \max_{\mathcal{D}} V(\mathcal{G},\mathcal{F},\mathcal{D})$$

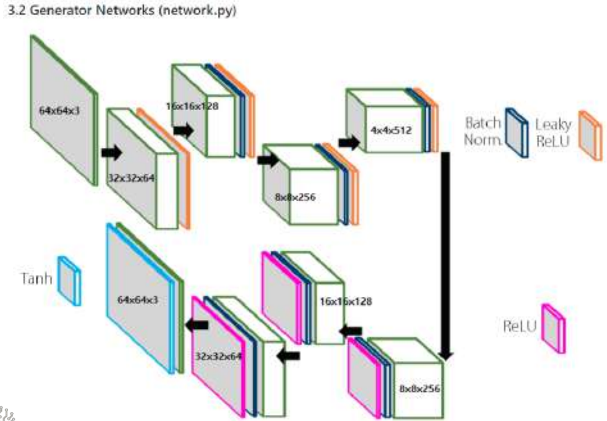
$$\text{where } V(\mathcal{G},\mathcal{F},\mathcal{D}) = \mathcal{L}_{x\to y} + \mathcal{L}_{y\to x}$$

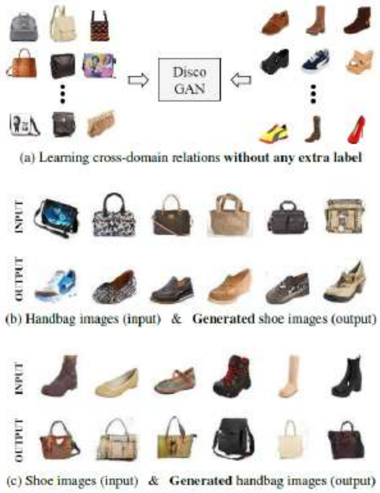
$$\mathcal{L}_{x\to y} = \mathbb{E}_y[\log \mathcal{D}(y)] + \mathbb{E}_x[\log(1-\mathcal{D}(\mathcal{G}(x)))] + \mathbb{E}_x[||\mathcal{F}(\mathcal{G}(x)) - x||_2]$$

$$\mathcal{L}_{y\to x} = \mathbb{E}_x[\log \mathcal{D}(x)] + \mathbb{E}_y[\log(1-\mathcal{D}(\mathcal{F}(y)))] + \mathbb{E}_y[||\mathcal{G}(\mathcal{F}(y)) - y||_2]$$

Machine Intelligence Lab

DiscoGANs





(a) Learning cross-domain relations **without any extra label**

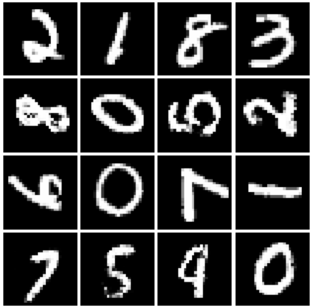
(b) Handbag images (input) & **Generated** shoe images (output)

(c) Shoe images (input) & **Generated** handbag images (output)

Implementation - DiscoGANs

- All codes are basically based on https://github.com/wiseodd/generative-models/blob/master/GAN/disco_gan/disco_gan_tensorflow.py, and modified by Ko.

- https://colab.research.google.com/drive/1GlnHNqs51pmFUsHk9UMkC8_jrp0qg3xU



Domain Adversarial Neural Networks

- The cost of generating labeled data $\Rightarrow$ Highly expensive!
- Learning a discriminative predictor in the *presence of a shift* between training set and test distributions is known as domain adaptation (DA).
    - Fully unlabeled target domain data $\Rightarrow$ unsupervised domain
    - Few labeled samples $\Rightarrow$ semi-supervised domain
- Focus : Combining deep feature learning & domain adaptation within one training process

  i.e. Learning features that combine discriminativeness & domain-invariance

  (i) The predictor is used both during training and at test time.

  (ii) The domain classifier discriminates the source and the target domains during training.
- Domain Adversarial Neural Network (DANN) [16] used adversarial learning strategy to focus on deep feature learning and domain apdatation.

Machine Intelligence Lab

Domain Adversarial Neural Networks

$$S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \sim (\mathcal{D}_\mathrm{S})^n; \qquad T = \{\mathbf{x}_i\}_{i=n+1}^{N} \sim (\mathcal{D}_\mathrm{T}^x)^{n'}$$

X	input space
$Y = 0, 1, \cdots, L-1$	set of L possible labels
$\mathcal{D}_\mathrm{S}$	source domain
$\mathcal{D}_\mathrm{T}$	target domain
S	source sample
T	target sample
$N = n + n'$	total number of samples

- The goal of the learning algorithm is to build a classifier $\eta : X \to Y$ with a low target risk,

$$R_{\mathcal{D}_\mathrm{T}}(\eta) = \Pr_{(\mathbf{x},y)\sim\mathcal{D}_\mathrm{T}} \Big(\eta(\mathbf{x}) \neq y\Big),$$

while having no information about the labels of $\mathcal{D}_\mathrm{T}$.

Machine Intelligence Lab

Domain Adversarial Neural Networks

- Focus : $\mathcal{H}$-divergence
  Given two domain distribution $\mathcal{D}_{\mathrm{S}}^{X}$ and $\mathcal{D}_{\mathrm{T}}^{X}$ over X, and a hypothesis class $\mathcal{H}$, the $\mathcal{H}$-divergence is

$$\begin{aligned} d_{\mathcal{H}}(\mathcal{D}_{\mathrm{S}}^{X}, \mathcal{D}_{\mathrm{T}}^{X}) &\equiv 2 \sup_{\eta \in \mathcal{H}} \left| \Pr_{\mathbf{x} \sim \mathcal{D}_{\mathrm{S}}^{X}} [\eta(\mathbf{x}) = 1] - \Pr_{\mathbf{x} \sim \mathcal{D}_{\mathrm{T}}^{X}} [\eta(\mathbf{x}) = 1] \right| \\ &= 2 \sup_{\eta \in \mathcal{H}} \left| \Pr_{\mathbf{x} \sim \mathcal{D}_{\mathrm{S}}^{X}} [\eta(\mathbf{x}) = 1] + \Pr_{\mathbf{x} \sim \mathcal{D}_{\mathrm{T}}^{X}} [\eta(\mathbf{x}) = 0] - 1 \right| \end{aligned}$$

The empirical $\mathcal{H}$-divergence between two samples $S \sim (\mathcal{D}_{\mathrm{S}}^{X})^{n}$ and $T \sim (\mathcal{D}_{\mathrm{T}}^{X})^{n'}$ can be computed,

$$\hat{d}_{\mathcal{H}}(S, T) = 2 \left(1 - \min_{\eta \in \mathcal{H}} \left[\frac{1}{n} \sum_{i=1}^{n} \mathrm{I}[\eta(\mathbf{x}_i) = 1] + \frac{1}{n'} \sum_{i=n+1}^{N} \mathrm{I}[\eta(\mathbf{x}_i) = 0] \right] \right),$$

where $\mathrm{I}[a]$ is the indicator function which is 1 if predicate a is true, and 0 otherwise.

Machine Intelligence Lab

Domain Adversarial Neural Networks

- Approximation of $\hat{d}_{\mathcal{H}}(S, T)$ by running a learning algorithm on the problem of discriminating between source and target examples
- A new dataset is constructed to approximate,

$$U = \{(\mathbf{x}_i, 0)\}_{i=1}^{n} \cup \{(\mathbf{x}_i, 1)\}_{i=n+1}^{N}$$

  where the examples of the source are labeled 0 and the target are labeled 1.
- Given a generalization error ϵ, the $\mathcal{H}$-divergence is then approximated by

$$\hat{d}_{\mathcal{A}} = 2(1 - 2\epsilon).$$

  The value $\hat{d}_{\mathcal{A}}$ is called the Proxy A-distance (PAD).

Domain Adversarial Neural Networks

- Common strategy to solve DA
  - Upper bound the target error by the source error + domain divergence

Let $\mathcal{H}$ be a hypothesis class of VC dimension d. With probability $1-\delta$ over the choice of samples $S \sim (\mathcal{D}_\mathrm{S})^n$ & $T \sim (\mathcal{D}_\mathrm{T}^X)^n$, for $\forall \eta \in \mathcal{H}$:

$$R_{\mathcal{D}_\mathrm{T}}(\eta) \leq R_\mathrm{S}(\eta) + \sqrt{\frac{4}{n}\left(d \log \frac{2en}{d} + \log \frac{4}{\delta}\right)}$$
$$+ \hat{d}_{\mathcal{H}}(S,T) + 4\sqrt{\frac{1}{n}\left(d \log \frac{2n}{d} + \log \frac{4}{\delta}\right)} + \beta$$

with $\beta \geq \inf_{\eta^* \in \mathcal{H}} [R_{\mathcal{D}_\mathrm{S}}(\eta^*) + R_{\mathcal{D}_\mathrm{T}}(\eta^*)]$, and

$R_\mathrm{S}(\eta) = \frac{1}{n}\sum_{i=1}^{m} \mathrm{I}[\eta(\mathbf{x}_i) \neq y_i]$ is the empirical source risk.

Domain Adversarial Neural Networks

$\exists$ $G_f(\cdot;\theta_f), G_y(\cdot;\theta_y), G_d(\cdot;\theta_d)$

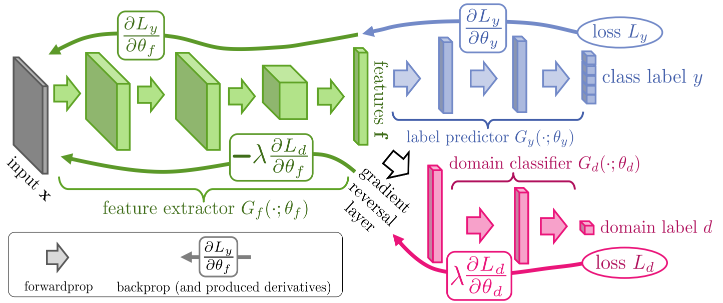
Now we can note the prediction loss & the domain loss repectively by

$$\mathcal{L}_y^i(\theta_f,\theta_y) = \mathcal{L}_y(G_y(G_f(\mathbf{x}_i;\theta_f) : \theta_y), y_i),$$
$$\mathcal{L}_d^i(\theta_f,\theta_d) = \mathcal{L}_d(G_d(G_f(\mathbf{x}_i;\theta_f) : \theta_d), d_i).$$

As the same manner,

$$(\hat{\theta}_f,\hat{\theta}_y) = \underset{\theta_f,\theta_y}{\arg\min} E(\theta_f,\theta_y,\hat{\theta}_d)$$
$$\hat{\theta}_d = \underset{\theta_d}{\arg\max} E(\hat{\theta}_f,\hat{\theta}_y,\theta_d).$$

Domain Adversarial Neural Networks



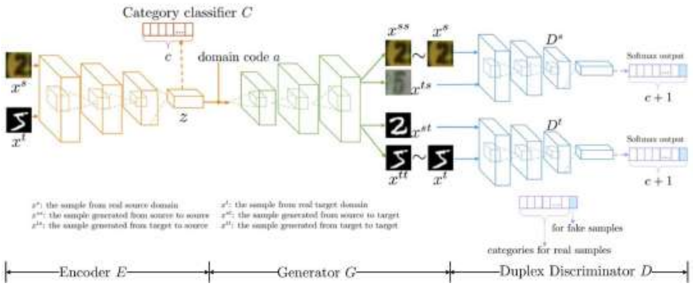
$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features $\mathbf{f}$
label predictor $G_y(\cdot;\theta_y)$
$-\lambda\frac{\partial L_d}{\partial \theta_f}$
input $\mathbf{x}$
feature extractor $G_f(\cdot;\theta_f)$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
domain label d
loss L_d
$\lambda\frac{\partial L_d}{\partial \theta_d}$
forwardprop
$\frac{\partial L_y}{\partial \theta_f}$
backprop (and produced derivatives)

- The updates are very similar to stochastic gradient descent for a feed-forward deep model.
- **Gradient Reversal Layer** (GRL)

- $\theta_f \longleftarrow \theta_f - \mu\left(\frac{\partial \mathcal{L}_y^i}{\partial \theta_f} - \lambda\frac{\partial \mathcal{L}_d^i}{\partial \theta_f}\right)$
- $\theta_y \longleftarrow \theta_y - \mu\frac{\partial \mathcal{L}_y^i}{\partial \theta_y}$
- $\theta_d \longleftarrow \theta_d - \mu\lambda\frac{\partial \mathcal{L}_d^i}{\partial \theta_d}$

Duplex GANs

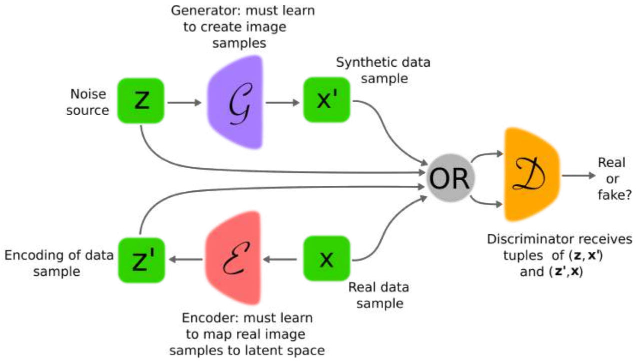
- Domain-invariant generate-able GANs
- DupGANs [18] have three parts, an encoder, a generator, and two discriminators (duplex discriminator).
  - Encoder extract latent code from the input domain.
  - Generator make artificial data using the encoded latent code and an additional domain code ($\approx$ condition vector in C-GANs [5]).
  - Discriminator decides real or fake of inputs and classifies a category of the inputs.



Category classifier C
c
domain code a
x^s
x^t
z
x^{ss}
x^s
x^{ts}
x^{st}
x^{tt}
x^t
D^s
D^t
Softmax output
$c+1$
Softmax output
$c+1$
x^s: the sample from real source domain
x^{ss}: the sample generated from source to source
x^{ts}: the sample generated from target to source
x^t: the sample from real target domain
x^{st}: the sample generated from source to target
x^{tt}: the sample generated from target to target
for fake samples
categories for real samples
Encoder E
Generator G
Duplex Discriminator D

GANs with inference models

- GANs lacked a way to map a given observation, $\mathbf{x}$, to a vector in latent space (often referred to as an *inference mechanism*).
- Several techniques have been proposed to invert the generator of pretrained GANs [21, 22].
- Introducing an inference network in which the $\mathcal{D}$ examine joint (data, latent) paris


Generator: must learn to create image samples
Noise source
z
$\mathcal{G}$
x'
Synthetic data sample
OR
$\mathcal{D}$
Real or fake?
Discriminator receives tuples of ($\mathbf{z}$, $\mathbf{x}'$) and ($\mathbf{z}'$, $\mathbf{x}$)
Encoding of data sample
z'
$\mathcal{E}$
x
Real data sample
Encoder: must learn to map real image samples to latent space

Machine Intelligence Lab

Adversarially Learned Inference

- ALI [22] consider the two following probability distributions over $\mathbf{x}$ and $\mathbf{z}$:

$$q(\mathbf{x},\mathbf{z}) = q(\mathbf{x})q(\mathbf{z}|\mathbf{x})$$
$$p(\mathbf{x},\mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}).$$

- The objective function is to match this two joint distributions.
  $\therefore$ We are assured that the conditional $q(\mathbf{z}|\mathbf{x})$ matches the posterior $p(\mathbf{z}|\mathbf{x})$.

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G},\mathcal{D})$$

where $V(\mathcal{G},\mathcal{D}) = \mathbb{E}_{q(\mathbf{x})}[\log \mathcal{D}(\mathbf{x},\mathcal{G}_{\mathbf{z}}(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\log(1-\mathcal{D}(\mathcal{G}_{\mathbf{x}}(\mathbf{z}),\mathbf{z}))]$

Machine Intelligence Lab

Implementation - ALI

- All codes are basically based on https://github.com/wiseodd/generative-models/blob/master/GAN/ali_bigan/ali_bigan_tensorflow.py, and modified by Ko.

- https://colab.research.google.com/drive/1IYaZzGk55UVbndMQkCslCGBMQqcm3o_R

Alternative Formulation of Generative Adversarial Networks

Recap.

- The first proposed GANs [7] objective function is nothing but Jensen-Shannon divergence between a generated samples distribution and a real samples distribution.
- In other words, training GANs is reducing distance between two distributions, the generated samples distribution and real samples distributions.
- Therefore, we do NOT need to use JS divergence only [2].
  - *"The other GANs training approach using variational divergence estimation is a special case of f-divergence approach."*

Strong & Weak Metric[6]

- We can define a distance function $d(x, y)$ if it satisfies:
    - $d(x, y) \geq 0$
    - $d(x, y) = 0 \Leftrightarrow x = y$
    - $d(x, y) = d(y, x)$
    - $d(x, y) \leq d(x, z) + d(z, y)$
- If we can define $d(x, y)$, then we can define a convergence:

$$x_n \to x \Leftrightarrow \lim_{n\to\infty} d(x_n, x) = 0.$$

- In an arbitrary space, we can define more than one distance function.
    - In other words, a distance in a space can be defined in various ways.

[6] https://www.slideshare.net/ssuser7e10e4/wasserstein-gan-i

Strong & Weak Metric

- Thus, we have to define strongness, weakness, or equality between distance functions.
- $d_1(x_n, x) = 0 \Rightarrow d_2(x_n, x) = 0$
  - d_1 is stronger than d_2.
- $d_1(x_n, x) = 0 \Leftarrow d_2(x_n, x) = 0$
  - d_1 is weaker than d_2.
- $d_1(x_n, x) = 0 \Leftrightarrow d_2(x_n, x) = 0$
  - d_1 and d_2 are equivalent.
- Taking a weak distance as an objective function is important to enhance sample quality!
  - In terms of learning probability distribution, we have to choose the distance carefully.

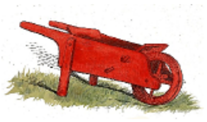
Different Distances

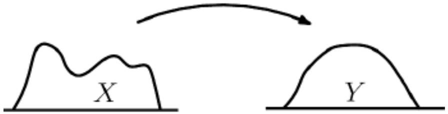
- $\mathcal{X} \leftarrow$ a compact metric set, $\Sigma \leftarrow$ a set of all the Borel subsets of $\mathcal{X}$
- The Total Variation (TV) distance [23]
  - $\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)|$
- The Kullback-Leibler (KL) divergence (it violates 3rd and 4th rule for distance function, thus it is not a metric, but a prematric.)
  - $KL(\mathbb{P}_r || \mathbb{P}_g) = \int \log \left(\frac{P_r(x)}{P_g(x)} \right) P_r(x) \mathrm{d}\mu(x)$
- The Jensen-Shannon (JS) divergence [7]
  - $JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r || \mathbb{P}_m) + KL(\mathbb{P}_g || \mathbb{P}_m)$ where $\mathbb{P}_m = (\mathbb{P}_r + \mathbb{P}_g)/2$
- The Earth-Mover (EM) distance or Wasserstein-1 [3, 24]
  - $W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma}[||x - y||]$
  - With a measurable prior $p(z)$, $W(\mathbb{P}_r, \mathbb{P}_g)$ is *continuous everywhere* and *differentiable almost everywhere*.

Details about Earth-Mover's Distnace

- Probability distribution $\rightarrow$ *Piled dirt*
- The minimum amount of work needed to transform piled dirt r to piled dirt g.

$$\mathrm{EMD}(S_r, S_g) = \sum_{i,j} \frac{f_{ij} d(m_{ri}, m_{gj})}{f_{ij}}$$





A figure for Earth Mover's distance[7]

[7] https://sbl.inria.fr/doc/group__Earth__mover__distance-package.html

Wasserstein Distance

Why should we use W-distance?

Theorem 1 Let $\mathbb{P}_r$ be a fixed distribution over $\mathcal{X}$. Let Z be a random variable (*e.g.*, Gaussian) over another space $\mathcal{Z}$. Let $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ be a function, that will be denoted $g_\theta(z)$ with z the firtst coordinate and θ the second. Let $\mathbb{P}_\theta$ denote the distribution of $g_\theta(Z)$. Then,

1. If g is continuous in θ, so is $W(\mathbb{P}_r, \mathbb{P}_\theta)$.
2. If g is locally Lipschitz and satisfies regularity *assumption 1*, then $W(\mathbb{P}_r, \mathbb{P}_\theta)$ *is continuous everywhere, and differentiable almost everywhere.*
3. Statements 1, 2 are false for the Jensen-Shannon divergence and Kullback-Leibler divergence.

Assumption 1. If there are local Lipschitz constants $L(\theta, z)$, and $\mathbb{E}_{z\sim p}[L(\theta, z)] < +\infty$, then *assumption 1* is satisfied.

Wasserstein Distance

Corollary 1 Let g_θ be any feedforward neural network parameterized by θ, and $p(z)$ a prior over z such that $\mathbb{E}_{z\sim p(z)}[||z||] < \infty$ (*e.g.*, Gaussian, uniform, *etc.*). Then *assumption 1* is satisfied and therefore $W(\mathbb{P}_r, \mathbb{P}_\theta)$ *is continuous everywhere and differentiable almost everywhere*.

- All this shows that EM is much more sensible cost for our problem than (at least) the JS divergence!

Wasserstein Distance

Theorem 2 Let $\mathbb{P}$ be a distribution on a compact space $\mathcal{X}$ and $(\mathbb{P}_n)_{n \in N}$ be a sequence of distributions on $\mathcal{X}$. Then, considering all limits as $n \to \infty$

1 The following statements are equivalent.

- $\delta(\mathbb{P}_n, \mathbb{P}) \to 0$
- $JS(\mathbb{P}_n, \mathbb{P}) \to 0$

2 The following statements are equivalent.

- $W(\mathbb{P}_n, \mathbb{P}) \to 0$
- $\mathbb{P}_n \xrightarrow{\mathcal{D}} \mathbb{P}$ where $\xrightarrow{\mathcal{D}}$ represents convergence in distribution for random variables.

3 $KL(\mathbb{P}_n || \mathbb{P}) \to 0$ or $KL(\mathbb{P} || \mathbb{P}_n) \to 0$ imply the statements in **1**.

4 The statements in **1** imply the statements in **2**.

- Above facts say that the KL, JS, and TV distances are not sensible cost functions when learning distributions supported by low dimensional manifolds.

Machine Intelligence Lab

W-GANs

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||]$$

$\boxed{\text{inf caluation} \rightarrow \text{highly intractable!}}$

- Kantorovich-Rubinstein duality tells us that

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

- If we have a prametrized family of functions $\{f_w\}_{w\in\mathcal{W}}$, we could consider solving the problem

$$\max_{w\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z))],$$

  and this process would yield a calculation of $W(\mathbb{P}_r, \mathbb{P}_\theta)$.
- Furthermore, we could consider differentialting $W(\mathbb{P}_r, \mathbb{P}_\theta)$!

Machine Intelligence Lab

W-GANs

Theorem 3 Let $\mathbb{P}_r$ be any distribution. Let $\mathbb{P}_\theta$ be the distribution of $g_\theta(Z)$ with Z a random variable with density p and g_θ a function satisfying *assumption 1.* Then, there is a solution $f : \mathcal{X} \to \mathbb{P}$ to the problem

$$\max_{||f||_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

and we have

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z))]$$

when both terms are well-defined.

- Weight clipping is needed (at all experiments in the paper, $\mathcal{W} = [-0.01, 0.01]^l$).

+ Additional research topic $\to$ tradeoff of weight clipping

- In results, W-distance is sensible cost function, because it is continuous everywhere and differentiable almost everywhere, further, it is calculatable thanks to Kantorovich-Rubinstein duality and weight clipping.

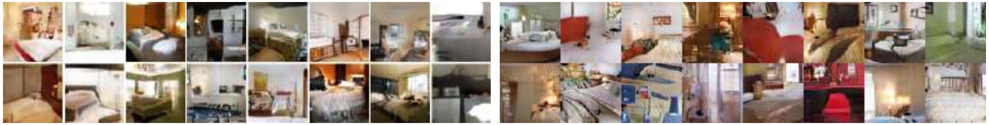
W-GANs

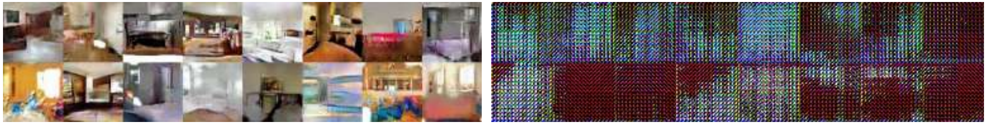
Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

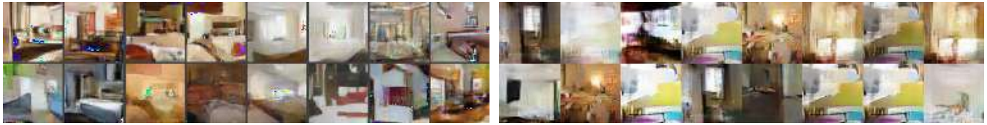
```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
 1: while $\theta$ has not converged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

An algorithm for Wasserstein GANs [3]

W-GANs

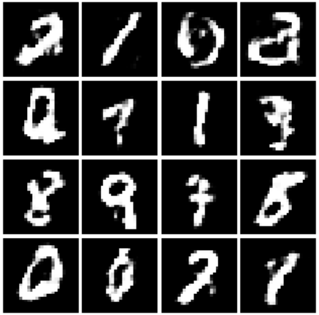

Left: W-GANs, Right: DCGANs


Left: W-GANs w/o BN, Right: DCGANs w/o BN


Left: W-GANs, Right: Fully-connected GANs

Implementation - W-GANs

- All codes are basically based on https://github.com/wiseodd/generative-models/blob/master/GAN/wasserstein_gan/wgan_pytorch.py, and modified by Ko.
- https://colab.research.google.com/drive/1zi_mvdZOvzwMoycnVGD-EdskbYWaLmGK



Recap.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||]$$

$\boxed{\inf \text{ caclulation} \rightarrow \text{highly intractable!}}$

- Kantorovich-Rubinstein duality tells us that

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

Definition Given two metric spaces (X, d_X) and (Y, d_Y), where d_X and d_Y denotes the metric on the set X and Y respectively, a function $f : X \rightarrow Y$ is called K-Lipschitz continuous function if $\exists\ K \in \mathbb{R}^{0,+}$ *s.t.*, $\forall\ x_1, x_2 \in X$, $d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2)$.

- If we have a prametrized family of functions $\{f_w\}_{w\in\mathcal{W}}$, we could consider solving the problem

$$\min_\theta \max_{w\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z))].$$

Machine Intelligence Lab

Recap.

$$\min_{\theta} \max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z))]$$

- As proposed by Arjovsky *et al.*, a simple way to restrict the class of function f that can be modeled by the NN to K-Lipschitz funtion is to perform weight clipping.

  i.e. To enforce the parameters of the network not to exceed a certain value

"This is terrible but simple choice..." *-Arjovsky et al.* [3]

- Weight clipping $\Rightarrow$ extremely limited number of functions

Gradient Penalty

$$\underbrace{\mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z))]}_{\text{original loss}} + \underbrace{\lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}} f_w(\hat{x})||_2 - 1)^2]}_{\text{gradient penalty}}$$

- **Sampling distribution**
  - $\mathbb{P}_{\hat{x}}$: straight lines between pairs of points samples from the $\mathbb{P}_r$ and $\mathbb{P}_{g_\theta(z)}$
  - Enforcing intractable gradient $\rightarrow$ sufficient and experimentally good simple straight lines
- **Penalty coefficient**
  - $\lambda = 10$ works well across a variety of architectures and dataset (toy dataset $\sim$ ImageNet CNNs).
- **No discriminator batch normalization**
  - Batch normalization is no longer valid in gradient penalty setting.
- **Two-sided penalty**

Machine Intelligence Lab

Gradient Penalty

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

```
Require: The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration
    $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
Require: initial critic parameters $w_0$, initial generator parameters $\theta_0$.
 1: while $\theta$ has not converged do
 2:     for $t = 1, ..., n_{\text{critic}}$ do
 3:         for $i = 1, ..., m$ do
 4:             Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
 5:             $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
 6:             $\hat{\boldsymbol{x}} \leftarrow \epsilon\boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
 7:             $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
 8:         end for
 9:         $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m}\sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$
10:     end for
11:     Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p(\boldsymbol{z})$.
12:     $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^m -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
13: end while
```

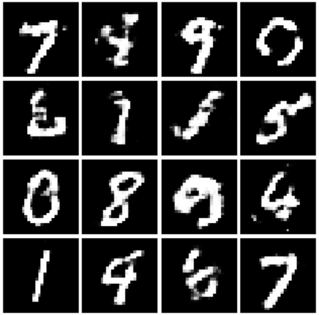
An algorithm for Wasserstein GANs with gradient penalty [24]

Machine Intelligence Lab

Implementation - W-GANs-GP

- All codes are basically based on https://github.com/wiseodd/generative-models/blob/master/GAN/improved_wasserstein_gan/wgan_gp_tensorflow.py, and modified by Ko.

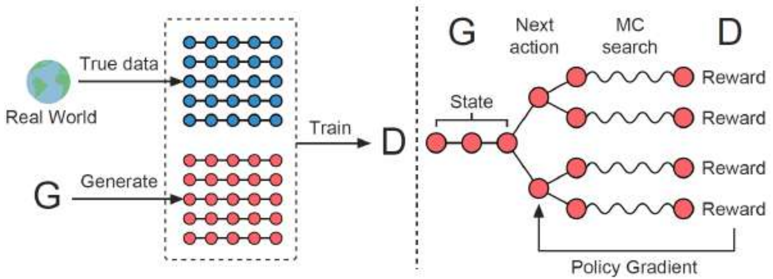
- https://colab.research.google.com/drive/19FfFoXNxKogVwKPFZjINttUCfeYhSEYw



SeqGANs

- Train a θ-parametrized generative model $\mathcal{G}_\theta$ to produce a sequence $Y_{1:T} = (1, \cdots, y_t \cdots, y_T), y_t \in \mathcal{Y}$ where $\mathcal{Y}$ is the vocabulary of candidate tokens [25].
  - In timestep t, the state s is the current produced tokens $(y_1, \cdots, y_{t-1})$ and the action a is the next token y_t to select.
  - The state trainsition is deterministic after an action has been chosen, *i.e.*, $\delta^a_{s,s'} = 1$ for the next state $s' = Y_{1:t}$ if the current state $s = Y_{1:t-1}$ and the action $a = y_t$, for other next states s'', $\delta^a_{s,s''} = 0$.

- Meanwhile, a ϕ-parametrized discriminative model $\mathcal{D}_\phi$ is trained to provide a guidance for improving the generator $\mathcal{G}_\theta$.



SeqGANs via Policy Gradient

- $\mathcal{G}_\theta(y_t|Y_{1:t-1})$ generates a sequence from the start state s_0 to maximize its expected end reward:

$$J(\theta) = \mathbb{E}[R_T|s_0,\theta] = \sum_{y_1 \in \mathcal{Y}} \mathcal{G}_\theta(y_1|s_0) \cdot Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(s_0, y_1),$$

where R_T is the reward for a complete sequence and $Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(s,a)$ is the action-value function of a sequence.

$$Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(a = y_T, s = Y_{1:T-1}) = \mathcal{D}_\phi(Y_{1:T})$$

SeqGANs via Policy Gradient

- To evaluate the action-value for an intermediate state, researches apply N-time MC search with a roll-out policy $\mathcal{G}_\beta$ to sample the unknown last $T - t$ tokens.
  - Go or Chess players sometimes would give up the immediate interests for the long-term victory.

$$\{Y_{1:T}^1, \cdots, Y_{1:T}^N\} = \mathrm{MC}^{\mathcal{G}_\beta}(Y_{1:t}\ N)$$

$$Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(s = Y_{1:t-1,a=y_t}) =$$
$$\begin{cases} \frac{1}{N}\sum_{n=1}^{N} \mathcal{D}_\phi(Y_{1:T}^n), Y_{1:T}^n \in \mathrm{MC}_\beta^{\mathcal{G}}(Y_{1:t}\ N) & \text{for } t < T \\ \mathcal{D}_\phi(Y_{1:t}) & \text{for } t = T \end{cases}$$

- Now the discriminator objective function is

$$\min_\phi -\mathbb{E}_{Y\sim p_{\text{data}}}[\log \mathcal{D}_\phi(Y)] - \mathbb{E}_{Y\sim \mathcal{G}_\theta}[\log(1 - \mathcal{D}_\phi(Y))].$$

SeqGANs via Policy Gradient

- And, the generator objective function is

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1}\sim\mathcal{G}_\theta}\left[\sum_{y_t\in\mathcal{Y}} \nabla_\theta \mathcal{G}_\theta(y_t|Y_{1:t-1}) \cdot Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(Y_{1:t-1}, y_t)\right] \\
&\simeq \sum_{t=1}^{T} \sum_{y_t\in\mathcal{Y}} \nabla_\theta \mathcal{G}_\theta(y_t|Y_{1:t-1}) \cdot Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(Y_{1:t-1}, y_t) \\
&= \sum_{t=1}^{T} \sum_{y_t\in\mathcal{Y}} \mathcal{G}_\theta(y_t|Y_{1:t-1}) \nabla_\theta \log \mathcal{G}_\theta(y_t|Y_{1:t-1}) \cdot Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(Y_{1:t-1}, y_t) \\
&= \sum_{t=1}^{T} \mathbb{E}_{y_t\sim\mathcal{G}_\theta(y_t|Y_{1:t-1})}[\nabla_\theta \log \mathcal{G}_\theta(y_t|Y_{1:t-1}) \cdot Q_{\mathcal{D}_\phi}^{\mathcal{G}_\theta}(Y_{1:t-1}, y_t)]
\end{aligned}$$

- Thus, for the parameters of the generator updating with h-th learning rate η_h,

$$\theta \leftarrow \theta + \eta_h \nabla_\theta (J(\theta))$$

For Deeper Understanding

Introduction

- In 2017, Mescheder *et al.* proposed algorithmic level enhancement for GANs [26].
- While very powerful, GANs are known to be notoriously hard to train.
  - To carefully design the model *(implementational level)*
    - DCGAN [9], Adding instance noise [12]
  - Selecting an easy-to-optimize objective function *(computational level)*
    - f-distance [2], Wasserstein distnace [3], Wasserstein distance with gradient penalty [24]
- The objective function of GANs is a *min-max game*.

Introduction

- If each player has chosen a strategy, and no player can benefit by changing strategies while the other players keep theirs unchanged, then the current set of strategy choices and their corresponding payoffs constitutes a *Nash equilibrium*.

- To stabilize the training of GANs = Finding local Nash equilibria of smooth games

- The authors contribute
  1. Identifying the main reasons why simultaneous gradient ascent often fails to find local Nash equilibria.
  2. Designing a new, more roubst algorithm for finding Nash equilibria of smooth two-player games.
  3. Demonstrating that the proposed method enable stable training of GANs on a variety of architectures and divergence measures.

- The proposed technique is orthogonal to strategies that try to make the GANs-game well defined.
  - Adding instance noise, Using W-divergence, *etc.*

Background

- GANs are best understood in the context of divergence minimization.
- Our goal is to find $\bar{\theta}$ that minimizes the divergence $D(p_0, q_\theta)$, *i.e.*, we want to solve the optimization problem
$$\min_\theta D(p_0, q_\theta).$$
- Most divergence that are used in practice can be represented in the following form:
$$D(p, q) = \max_{f \in \mathcal{F}} \mathrm{E}_{x \sim q}[g_1(f(x))] - \mathrm{E}_{x \sim p}[g_2(f(x))]$$
for some function class $\mathcal{F} \subseteq \mathcal{X} \sim \mathbb{R}$ and convex functions $g_1, g_2 : \mathbb{R} \to \mathbb{R}$. This leads to min-max problem of the form
$$\min_\theta \max_{f \in \mathcal{F}} \mathrm{E}_{x \sim q_\theta}[g_1(f(x))] - \mathrm{E}_{x \sim p_0}[g_2(f(x))].$$

Machine Intelligence Lab

Background

Consider GANs as Smooth Two-Player Games

- A differentiable two-palyer game is defined by two utility functions $f(\phi, \theta)$ and $g(\phi, \theta)$, defined over a common space $(\phi, \theta) \in \Omega_1 \times \Omega_2$.
- Ω_1 corresponds to the possible actions of player 1, Ω_2 corresponds to the possible actions of player 2.
- The goal of player 1 is to maximize f whereas player 2 tries to maximize g.
- In the context of GANs, Ω_1 is the set of possible parameter values for the generator, whereas Ω_2 is the set of possible parameter values for the discriminator.

Background

- Our goal is to find a Nash equilibrium of the game, *i.e.*, a point $\bar{x} = (\bar{\phi}, \bar{\theta})$ given by the two conditions
$$\bar{\phi} \in \arg\max_{\phi} f(\phi, \bar{\theta}) \quad \text{and} \quad \bar{\theta} \in \arg\max_{\theta} g(\bar{\phi}, \theta). \tag{1}$$
If Eq. (1) holds in a local neighborhood of $(\bar{\phi}, \bar{\theta})$, we call a point $(\bar{\phi}, \bar{\theta})$ a local Nash equilibrium.

- Every differentiable two-player game defines a vector field
$$v(\phi, \theta) = \begin{pmatrix} \nabla_\phi f(\phi, \theta) \\ \nabla_\theta g(\phi, \theta) \end{pmatrix}.$$
We call v the associated gradient vector field to the game defined by f and g.

- For the special case of zero-sum two-player games, we have $g = -f$ and thus
$$v'(\phi, \theta) = \begin{pmatrix} \nabla_\phi^2 f(\phi, \theta) & \nabla_{\phi,\theta} f(\phi, \theta) \\ -\nabla_{\phi,\theta} f(\phi, \theta) & -\nabla_\theta^2 f(\phi, \theta) \end{pmatrix}.$$

Background

Lemma For zero-sum games, $v'(x)$ is negative (semi-)definite *iff* $\nabla^2_\phi f(\phi, \theta)$ is negative (semi-)definite and $\nabla^2_\theta f(\phi, \theta)$ is positive (semi-)definite.

Corollary For zero-sum games, $v'(\bar{x})$ is negative semi-definite for any local Nash equilibrium $\bar{x}$. Conversely, if $\bar{x}$ is a stationary point of $v(x)$ and $v'(x)$ is negative definite, then $\bar{x}$ is a local Nash equilibrium.

Simultaneous Gradient Ascent

- The de-facto standard algorithm for finding Nash equilibria of general smooth two-player games is Simultineous Gradient Ascent (SimGA).

Algorithm 1 Simultaneous Gradient Ascent (SimGA)

```
1: while not converged do
2:     $v_\phi \leftarrow \nabla_\phi f(\theta, \phi)$
3:     $v_\theta \leftarrow \nabla_\theta g(\theta, \phi)$
4:     $\phi \leftarrow \phi + h v_\phi$
5:     $\theta \leftarrow \theta + h v_\theta$
6: end while
```

- Main idea of SimGA is that iteratively update the parameters of the two players by simultaneously applying gradient ascent to the utility functions of the two players.
  - $\approx$ Euler approximation

- The paper shows that two major failure causes for this algorithm are: 1) eigenvalues of the Jacobian of the associated gradient vector field with zero real-part; 2) eigenvalues with large imaginary part.

Machine Intelligence Lab

Convergence Theory

Proposition Let $F : \Omega \to \Omega$ be a continuously differential function on an open subset Ω of $\mathbb{R}^n$ and let $\bar{x} \in \Omega$ be so that

1. $F(\bar{x}) = \bar{x}$, and
2. the absolute values of the eigenvalues of the Jacobian $F'(\bar{x})$ are all smaller that 1.

Then there is an open neighborhood U of $\bar{x}$ so that for all $x_0 \in U$, the iterates $F^{(k)}(x_0)$ converge to $\bar{x}$. The rate of convergence is at least linear. More precisely, the error $||F^{(k)}(x_0) - \bar{x}||$ is in $\mathcal{O}(|\lambda_{\text{max}}|^k)$ for $k \to \infty$ where λ_{max} is the eigenvalue of $F'(\bar{x})$ with the largest absolute value.

- In numerics, we often consider functions of the form

$$F(x) = x + hG(x) \tag{2}$$

for some $h > 0$. Finding fixed points of F is then equivalent to finding solutions to the nonlinear euation $G(x) = 0$ for x. For F as in Eq. (2), the Jacobian is given by

$$F'(x) = I + hG'(x).$$

Convergence Theory

- Note that in general neither $F'(x)$ nor $G'(x)$ are symmetric and can therefore have complex eigenvalues.

  Lemma Assume that $A \in \mathbb{R}^{n \times n}$ only has eigenvalues with negative real-part and let $h > 0$. Then the eigenvalues of the matrix $I + hA$ lie in the unit ball *iff*

  $$h < \frac{1}{|\Re(\lambda)|} \frac{2}{1 + \left(\frac{\Im(\lambda)}{\Re(\lambda)}\right)^2}$$

  for all eigenvalues λ of A.

- As $q = \frac{\Im(\lambda)}{\Re(\lambda)}$ goes to infinity, we have to choose h according to $\mathcal{O}(q^{-2})$, which can quickly become extremely small.

- As a result, if $G'(\bar{x})$ has an eigenvalue with small absolute real part but big imaginary part, h needs to be chosen extremely small to still achieve convergence.

Machine Intelligence Lab

Consensus Optimization

- Finding stationary points of the vector field $v(x)$ is equivalent to solving the equation $v(x) = 0$. In the context of two-player games this means solving the two equations

$$\nabla_\phi f(\phi, \theta) = 0 \quad \text{and} \quad \nabla_\theta g(\phi, \theta) = 0.$$

- A simple strategy for finding such stationary points is to minimize $L(x) = \frac{1}{2}||v(x)||^2$ for x. Unfortunately, practically they found it did NOT work well.

- They therefore consider a modified vector field $w(x)$ that is as close as possible to the original vector field $v(x)$, but at the same time still minimizes $L(x)$.

$$w(x) = v(x) - \gamma \nabla L(x)$$

  for some $\gamma > 0$.

Consensus Optimization

- A simple calculation shows that the gradient $\nabla L(x)$ is given by

$$\nabla L(x) = v'(x)^\top v(x).$$

- This vector field is the gradient vector field associated to the modified two-player game given by the two modified utility functions

$$\tilde{f}(\phi, \theta) = f(\phi, \theta) - \gamma L(\phi, \theta) \quad \text{and} \quad \tilde{g}(\phi, \theta) = g(\phi, \theta) - \gamma L(\phi, \theta).$$

The regularizer $L(\phi, \theta)$ encourages agreement between the two players.

Algorithm 2 Consensus optimization

```
1: while not converged do
2:     $v_\phi \leftarrow \nabla_\phi(f(\theta, \phi) - \gamma L(\theta, \phi))$
3:     $v_\theta \leftarrow \nabla_\theta(g(\theta, \phi) - \gamma L(\theta, \phi))$
4:     $\phi \leftarrow \phi + h v_\phi$
5:     $\theta \leftarrow \theta + h v_\theta$
6: end while
```

Consensus Optimization

Lemma Assume $h > 0$ and $A(x)$ invertible for all x. Then $\bar{x}$ is a fixed point of $F(x) = x + hA(x)v(x)$ *iff* it is a stationary point of v. Moreover, if $\bar{x}$ is a stationary point of v, we have

$$F'(\bar{x}) = I + hA(\bar{x})v'(\bar{x}).$$

Lemma Let $A(x) = I - \gamma v'(x)^\top$ and assume that $v'(\bar{x})$ is negative semi-definite and invertile. Then $A(\bar{x})v'(\bar{x})$ is negative definite.

Corollary Let $v(x)$ be the associated gradient vector field of a two-player zero-sum game and $A(x) = I - \gamma v'(x)^\top$. If $\bar{x}$ is a local Nash equilibrium, then there is an open neighborhood U of $\bar{x}$ so that for all $x_0 \in U$, the iterates $F^{(k)}(x_0)$ converge to $\bar{x}$ for $h > 0$ small enough.

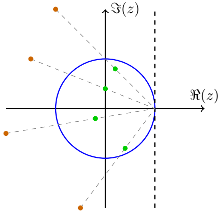
Consensus Optimization

Lemma Assume that $A \in \mathbb{R}^{n\times n}$ is negative semi-definite. Let $q(\gamma)$ be the maximum of $\frac{|\Im|(\lambda)}{|\Re(\lambda)|}$ (possibly infinite) *w.r.t.* λ where λ denotes the eigenvalue of $A - \gamma A^\top A$. Moreover, assume that A is invertible with $|Av| \geq \rho|v|$ for $\rho > 0$ and let

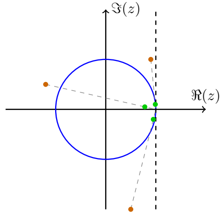
$$c = \min_{v\in\mathbb{S}(\mathbb{C}^n)} \frac{|\bar{v}^\top(A + A^\top)v|}{|\bar{v}^\top(A - A^\top)v|}$$

where $\mathbb{S}(\mathbb{C}^n)$ denotes the unit sphere in $\mathbb{C}^n$. Then

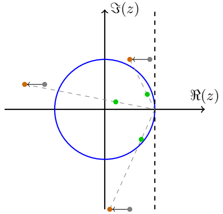
$$q(\gamma) \leq \frac{1}{c + 2\rho^2\gamma}.$$



(a) Illustration how the eigenvalues are projected into unit ball.

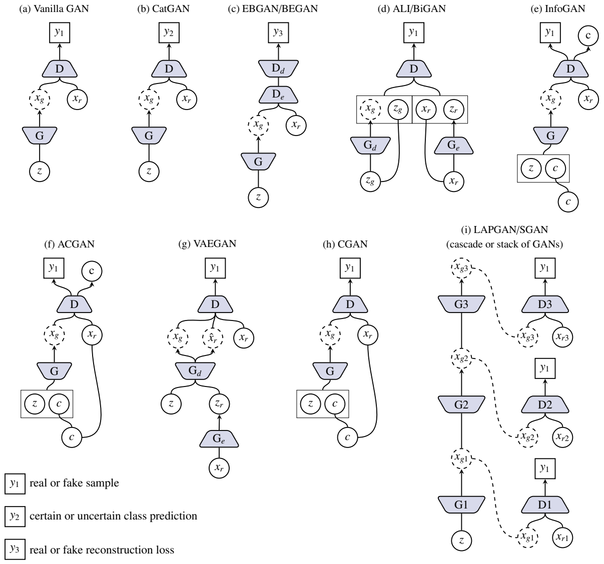


(b) Example where h has to be chosen extremely small.

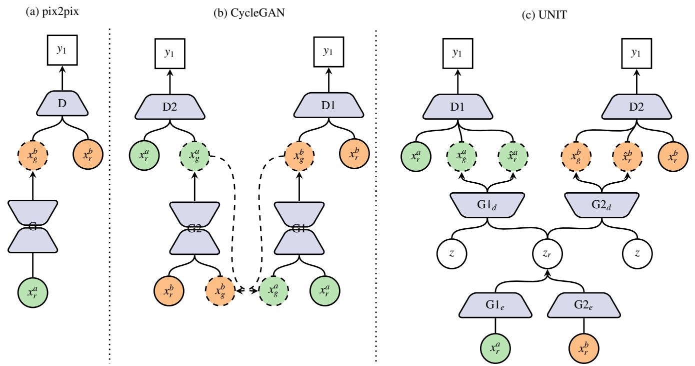


(c) Illustration how our method alleviates the problem.

Review [27]


(a) Vanilla GAN
(b) CatGAN
(c) EBGAN/BEGAN
(d) ALI/BiGAN
(e) InfoGAN
(f) ACGAN
(g) VAEGAN
(h) CGAN
(i) LAPGAN/SGAN
(cascade or stack of GANs)
y_1 real or fake sample
y_2 certain or uncertain class prediction
y_3 real or fake reconstruction loss
D
G
D_d
D_e
G_d
G_e
x_g
x_r
$\hat{x}_r$
z
z_g
z_r
c
G1
G2
G3
D1
D2
D3
x_{g1}
x_{g2}
x_{g3}
x_{r1}
x_{r2}
x_{r3}

Review [27]



🟩 domain A 🟧 domain B ○ real image ◌ generated fake image

pix2pix: x_r^a ⚭ x_r^b Aligned training sample

$\|G(x_r^a) - x_r^b\|_p$: target consistency

CycleGAN: x_r^a ⚯ x_r^b Unaligned training sample

$\|G2(G1(x_r^a)) - x_r^a\|_p + \|G1(G2(x_r^b)) - x_r^b\|_p$: cycle consistency

References I

[1] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, “Generative Adversarial Networks: An Overview,” *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[2] S. Nowozin, B. Cseke, and R. Tomioka, “f-gan: Training Generative Neural Samplers using Variational Divergence Minimization,” in *Advances in Neural Information Processing Systems*, pp. 271–279, 2016.

[3] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN,” *arXiv preprint arXiv:1701.07875*, 2017.

[4] I. Goodfellow, “NIPS 2016 Tutorial: Generative Adversarial Networks,” *arXiv preprint arXiv:1701.00160*, 2016.

[5] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” *arXiv preprint arXiv:1411.1784*, 2014.

[6] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets,” in *Advances in Neural Information Processing Systems*, pp. 2172–2180, 2016.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[8] E. L. Denton, S. Chintala, R. Fergus, *et al.*, “Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks,” in *Advances in neural information processing systems*, pp. 1486–1494, 2015.

[9] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” *arXiv preprint arXiv:1511.06434*, 2015.

[10] C. Donahue, J. McAuley, and M. Puckette, “Synthesizing Audio with Generative Adversarial Networks,” *arXiv preprint arXiv:1802.04208*, 2018.

[11] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive Growing of GANs for Improved Quality, Stability, and Variation,” *arXiv preprint arXiv:1710.10196*, 2017.

[12] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved Techniques for Training GANs,” in *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

[13] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks,” *arXiv preprint*, 2017.

[14] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation using Cycle-consistent Adversarial Networks,” *arXiv preprint arXiv:1703.10593*, 2017.

Machine Intelligence Lab

References II

[15] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, “Learning to Discover Cross-domain Relations with Generative Adversarial Networks,” *arXiv preprint arXiv:1703.05192*, 2017.

[16] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, “Domain-adversarial Training of Neural Networks,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[17] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, “Adversarial Discriminative Domain Adaptation,” in *Computer Vision and Pattern Recognition (CVPR)*, vol. 1, p. 4, 2017.

[18] L. Hu, M. Kan, S. Shan, and X. Chen, “Duplex Generative Adversarial Network for Unsupervised Domain Adaptation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1498–1507, 2018.

[19] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, “Photo-Realistic Single Image Super-Resolution using a Generative Adversarial Network,” *arXiv preprint*, 2016.

[20] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, “Least Squares Generative Adversarial Networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2017.

[21] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, “Adversarially learned inference,” *arXiv preprint arXiv:1606.00704*, 2016.

[22] J. Donahue, P. Krähenbühl, and T. Darrell, “Adversarial feature learning,” *arXiv preprint arXiv:1605.09782*, 2016.

[23] J. Zhao, M. Mathieu, and Y. LeCun, “Energy-Based Generative Adversarial Network,” *arXiv preprint arXiv:1609.03126*, 2016.

[24] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved Training of Wasserstein GANs,” in *Advances in Neural Information Processing Systems*, pp. 5767–5777, 2017.

[25] L. Yu, W. Zhang, J. Wang, and Y. Yu, “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient,” in *AAAI*, pp. 2852–2858, 2017.

[26] L. Mescheder, S. Nowozin, and A. Geiger, “The Numerics of GANs,” in *Advances in Neural Information Processing Systems*, pp. 1825–1835, 2017.

[27] X. Yi, E. Walia, and P. Babyn, “Generative Adversarial Network in Medical Imaging: A Review,” *arXiv preprint arXiv:1809.07294*, 2018.

Machine Intelligence Lab

Thank You!

(Q & A)

wjko@korea.ac.kr