

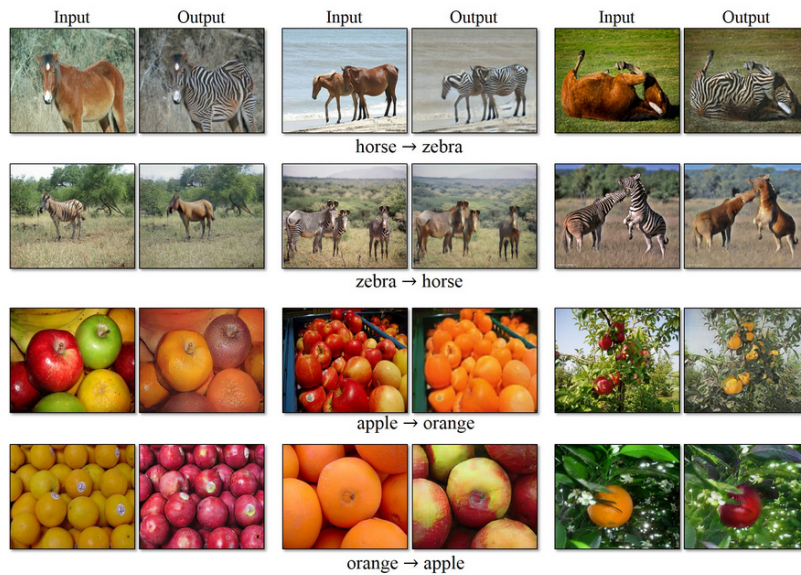
50.039 – Theory and Practice of Deep Learning

Alex

Week 10: Generative Adversarial Networks II

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

Object Transfiguration



Pictures from <https://github.com/junyanz/CycleGAN>



Key takeaways

- Wasserstein GAN are trained by minimizing (an approximation to) the wasserstein distance. Wasserstein distance is a distance between probability distributions
- the wasserstein distance can be approximated by maximizing a neural net output, which is the difference of two averages - one over real data, and one over samples from a generator
- be able to explain the rough idea of cyclegans and their applications
- be able to discuss the implication of artistic randomness when cyclegans are used to modify content
- be able to name example purposes for which the use of cyclegans is dangerous because they extrapolate features from the training data



Problem:

given a dataset $D_n = \{x_1, \dots, x_n\}$, for example of images, be able to generate *similar* images as in the dataset.

Critical thinking: What is similar ?

Formally:

- given D_n , learn a model G (by its parameters) so that one can randomly sample new samples using this model by

$$x = f(z), z \sim N(0, I_d)$$

- the learning process should enforce some kind of similarity between the D_n and $x = G(z), z \sim N(0, I_d)$

What kind of similarity ?

D_n are samples drawn from some distribution P_{data} . $x = G(z), z \sim N(0, I_d)$ allows to sample from the trained distribution P_{GAN} . Optimally one wanted to measure a loss between P_{data} and P_{test} , and learn a model so that $P_{GAN} = P_{data}$. Why this is not directly doable ?

Practically, one has to use measures between the finite sample set D_n and a set of samples drawn from f .

1 Wasserstein GANs

<https://arxiv.org/abs/1701.04862> and (for better training with a gradient penalty

instead of gradient clipping:) <https://arxiv.org/abs/1704.00028> draws a lot of interest currently. For an accessible explanation see: https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490. For an explanation out of class that touches the minimal math please see:

Idea:

- Your training data follows some unknown probability distribution μ . You don't know μ , but you have a finite sample: your training samples.
- GANs by $f_w(z), z \sim N(0, I_d)$ are a method to sample from some probability distribution ν . By training a GAN == selecting the parameters w of f_w you select a distribution ν to sample from.

Since your neural network generator is a function to sample from ν , why not directly minimize a distance measure between these two ν and μ ?!

Wasserstein-1-Distance a.k.a Earth Movers distance is a distance between two probability measures, which can be efficiently computed on GPUs.

The idea ... Suppose for the start you have a finite set of points

$$\{x_i, i = 1, \dots, n\},$$

Suppose you have now a joint probability measure γ over pairs (x_i, x_k) , that is

$$\gamma(x_i, x_k) \geq 0, \sum_{i=1}^n \sum_{k=1}^n \gamma(x_i, x_k) = 1$$

Then you can define a real number $e(\gamma)$ by computing the average euclidean distance between pairs (x_i, x_k) , weighted by the joint probability γ :

$$e(\gamma) = \sum_{i=1}^n \sum_{k=1}^n \|x_i - x_k\| \gamma(x_i, x_k)$$



You get for every joint probability $\gamma(x_i, x_k)$ such a weighted euclidean distance between pairs

You can use this idea to define a distance between two probability measures two probability measures over these sets of points: $\mu(x_i)$ and $\nu(x_i)$, that is

$$\begin{aligned} \mu(x_i) &\geq 0, \sum_{i=1}^n \mu(x_i) = 1, \\ \nu(x_i) &\geq 0, \sum_{i=1}^n \nu(x_i) = 1 \end{aligned}$$



A simple version of the Wasserstein distance between μ and ν is the smallest $d(\gamma)$ which can be computed over all joint distributions γ , which have μ and ν as their marginal distributions, that is:

$$\sum_k \gamma(x_i, x_k) = \mu(x_i)$$

$$\sum_i \gamma(x_i, x_k) = \nu(x_k)$$

in math: Lets define $\Gamma(\mu, \nu)$ to be the set of all joint distributions γ such that μ and ν are their marginals as above:

$$\Gamma(\mu, \nu) = \left\{ \gamma(\cdot, \cdot) \text{ is a joint probability on } \mathcal{X} \times \mathcal{X} \text{ and } \sum_k \gamma(x_i, x_k) = \mu(x_i), \sum_i \gamma(x_i, x_k) = \nu(x_k) \right\}$$

then a simple Version of Wasserstein distance is given as:

Wasserstein distance for the case of a discrete probability:

$$W_1(\mu, \nu) = \inf_{\gamma \in \Gamma(\mu, \nu)} e(\gamma)$$

$$\inf_{\gamma \in \Gamma(\mu, \nu)} \sum_{i=1}^n \sum_{k=1}^n \|x_i - x_k\| \gamma(x_i, x_k)$$

More generally you can extend this definition from a finite set of points $\{x_i, i = 1, \dots, n\}$ to probabilities with densities.

In that case one would replace the $\sum_i \sum_k$ weighted by $\gamma(x_i, x_k)$ by an integral over the joint density $\gamma(x, y)$:

$$\sum_{i=1}^n \sum_{k=1}^n \|x_i - x_k\| \gamma(x_i, x_k)$$

$$\rightsquigarrow \int \|x - y\| \gamma(x, y) dx dy$$

Suppose I have two probability measures $\mu(x)$ and $\nu(x)$ both over some space \mathcal{X} , then you can look at all joint probability densities $\gamma(x_1, x_2)$ defined in $\mathcal{X} \times \mathcal{X}$ such that

- if you integrate out the second variable in γ , then you obtain μ
- if you integrate out the first variable in γ , then you obtain ν

in math:

$$\int_{\mathcal{X}} \gamma(x_1, x_2) dx_2 = \mu(x_1)$$

$$\int_{\mathcal{X}} \gamma(x_1, x_2) dx_1 = \nu(x_2)$$

Let now $\Gamma(\mu, \nu)$ be the space of all joint probability densities γ in $X \times X$ which have μ and ν as their marginals.

Wasserstein distance for the general case

Given a metric $d(x, y)$ (which can be a euclidean distance $d(x, y) = \|x - y\|_2$), it is given as:

$$W_1(\mu, \nu) = \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{X \times X} d(x, y) \gamma(x, y) dx dy$$

To understand this, consider a special case: we use euclidean distance $d(x, y) = \|x - y\|_2$, and we have only a finite set of points $\{x_1, \dots, x_i, \dots, x_n\}$, then for one particular probability $\gamma(x, y) = P(X = x, Y = y)$:

$$\int d(x, y) d\gamma(x, y) = \sum_{i,j} \|x_i - x_j\| P(x_i, x_j)$$

This is an averaged euclidean distance between pairs (x_i, x_j) , weighted by the probability that we observe in our data a pair (x_i, x_j)

An Example of how $\Gamma(\mu, \nu)$ looks like :

This is example does not need to be memorized for quiz or exam. Its for your understanding of this space by an example.

Consider the space $Z = \{0, 1\}$. Let us define two exemplary probability measures μ, ν :

$$\mu(x = 0) = 0.3$$

$$\nu(y = 0) = 0.5$$

We want to know what $\Gamma(\mu, \nu)$ looks like!

First, the table below gives (actually all) the joint distributions $\gamma(x, y)$ which satisfy these marginals:

	y=0	y=1	$\mu(x)$
x=0	$0.3 - 0.3q$	$0.0 + 0.3q$	0.3
x=1	$0.2 + 0.3q$	$0.5 - 0.3q$	0.7
$\nu(y)$	0.5	0.5	

For every $q \in [0, 1]$ the table defines a joint probability $\gamma[q](x, y)$

So every $\gamma(x, y) \in \Gamma(\mu, \nu)$ has the shape (for some $q \in [0, 1]$)

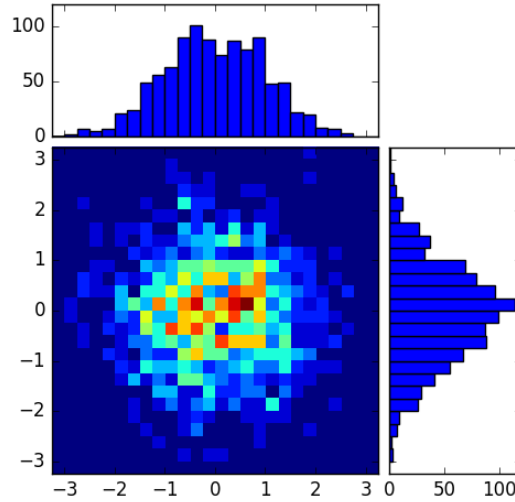
$$\gamma[q](x, y) = \begin{cases} 0.3 - 0.3q & \text{if } x = 0, y = 0 \\ 0.0 + 0.3q & \text{if } x = 0, y = 1 \\ 0.2 + 0.3q & \text{if } x = 1, y = 0 \\ 0.5 - 0.3q & \text{if } x = 1, y = 1 \end{cases}$$

Above case-based definition $\gamma[q](x, y)$ is just a way to rewrite the table as a joint distribution. Therefore the space is given as

$$\Gamma(\mu, \nu) = \{ \gamma[q](\cdot, \cdot), q \in [0, 1] \}$$

with $\gamma[q](\cdot, \cdot)$ defined as above. It should be now intuitive what $\Gamma(\mu, \nu)$ can be.

For a discrete distribution over many possible states (not just $Z = \{0, 1\}$), there can be many different joint distribution as in the graphic below:



There can be many 2d-distributions (middle square) which satisfy the marginal distributions shown on top and right

1.1 How to use the Wasserstein distance to train a GAN?

First idea: train a generator ν to minimize the Wasserstein distance between the distribution underlying the generator sampling and the distribution underlying the training data. We can formulate a preliminary

Generator ν training goal:

$$\nu = \operatorname{argmin}_{\nu} W(\mu, \nu)$$

steps to obtain a wasserstein GAN

- write the wasserstein distance as a difference of two expectations of a function f , maximized over the function f

$$W_1(\mu, \nu) = \max_{f \in Lip_1} E_{x \sim \mu}[f(x)] - E_{y \sim \nu}[f(y)]$$

- replace the expectations $E[\cdot]$ by mini-batch averages, one time over the real data, one time over the data from the generator
- represent the function f , the so-called critic, by the output of a neural net. train the critic NN using the objective to be maximized in order to solve this maximization problem in a neural net (typically with some gradient norm penalty)
- interleave the last step with a step which optimizes the generator

Problem: we don't know both distributions, we can only obtain some samples from the generator and we can sample trivially from the training data.

The theorem is out of class, you should just remember that the distance can be computed as the maximum of two expectations, one expectation over the training data, and one expectation from the generator – bcs this is what we need to use for GAN training as you will see.

Kantorovich-Rubinstein Duality Theorem

The Wasserstein distance for a euclidean metric can be computed as (Kantorovich-Rubinstein Duality Theorem, see a theorem in chapter 5 of the book by Cedric Villani, *Optimal Transport: Old and New*):

$$\begin{aligned} W_1(\mu, \nu) &= \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{X \times X} \|x - y\| d\gamma(x, y) \\ &= \max_{f \in Lip_1} E_{x \sim \mu}[f(x)] - E_{y \sim \nu}[f(y)] \\ f \in Lip_1 &\Leftrightarrow |f(x) - f(y)| \leq \|x - y\|_2 \quad \forall x, y \end{aligned}$$

How this can be used ? I want to show you that you can use a deep neural net to represent f , and to minimize a loss to find a good f .

$$W_1(\mu, \nu) = \max_{f \in Lip_1} E_{x \sim \mu}[f(x)] - E_{y \sim \nu}[f(y)]$$

Lets forget now the restriction $f \in Lip_1$ in the maximum – for a moment, and only consider a maximum.

The Wasserstein distance can be computed as a difference of two expectations – one from the data distributioun μ , and one from the generator distribution. We do not have these expectations, but we have CLT!!

$$\begin{aligned} W_1(\mu, \nu) &= \max_{f \in Lip_1} E_{x \sim \mu}[f(x)] - E_{y \sim \nu}[f(y)] \\ &\approx \max_{f \in Lip_1} \frac{1}{n} \sum_{x_i \sim Data}^n f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d}^n f(y_i) \end{aligned}$$

Now we have a maximum over some function over two minibatches – one drawn from the data, one drawn from our Generator.

Maximum over some average of minibatches – f can be represented by a neural net! The maximum can be obtained by loss minimization

$$\begin{aligned} W_1(\mu, \nu) &\approx \max_{f \in Lip_1} \frac{1}{n} \sum_{x_i \sim Data}^n f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d}^n f(y_i) \\ &= \min_{f \in Lip_1} -1 \left(\frac{1}{n} \sum_{x_i \sim Data}^n f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d}^n f(y_i) \right) \\ &= \min_{f \in Lip_1} L, \\ L &= -1 \left(\frac{1}{n} \sum_{x_i \sim Data}^n f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d}^n f(y_i) \right) \end{aligned}$$

This is something which one can do with a neural net toolbox!!

```
critic_optimizer=torch.nn.optim.Adam(f.parameters(),lr=...)
L= #that diff of averages with f
L.backward()
critic_optimizer.step()
```

What is the conclusion: The wasserstein distance $W_1(\mu, \nu)$ can be approximated by loss minimization of

$$L = -1 \left(\frac{1}{n} \sum_{x_i \sim Data}^n f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d}^n f(y_i) \right)$$

This allows us now to formulate a training of Wasserstein GANs as an interleaved loop of two steps. The other step will be the generator v training goal

$$v = \operatorname{argmin}_v W(\mu, v)$$

Wasserstein GAN training

Let f be called the critic. f replaces the discriminator in the classic GAN.

- define a neural net model for the generator g . it takes a minibatch of vectors as input, and outputs a minibatch of images.
 - define a neural net model for the critic f . it takes a minibatch of images as output and produces a real number
 - add all the bells and whistles, optimizer, learning rate scheduler, model saving, every n steps output of example images, and so on.
1. fix the generator g . Compute approximation to the wasserstein distance by training to find a good critic f . Do this by a small number of gradient descent steps on minimizing this loss:

$$f \leftarrow \operatorname{argmin}_{f \in Lip_1} L \text{ (!) only a few grad descent steps}$$

$$L = -1 \left(\frac{1}{n} \sum_{x_i \sim \text{Data}} f(x_i) - \frac{1}{n} \sum_{y_i = \text{Generator}(z_i), z_i \sim N(), I_d} f(y_i) \right) + \lambda R(f)$$

$$R(f) = \text{for example } \frac{1}{n} \sum_{z_i = tx_i + (1-t)y_i, t \sim U([0,1])} [\max(0, \|\nabla f(z_i)\| - 1)^2]$$

I explain $R(f)$ below.

You do a few steps of gradient descent, and not just one, because you want to get close to the minimum $\min_{f \in Lip_1} L$ (if you do too many steps, you may encounter mode collapse, on the other hand doing this on small minibatches again presents a moving target and injects noise).

2. fix f . Now train the generator g . Do one step of gradient descent on

$$G \leftarrow \operatorname{argmin}_G - \frac{1}{n} \sum_{y_i = G(z_i), z_i \sim N(), I_d} f(y_i)$$

Why does the generator step look like that ?

$$W_1(\mu, \nu) \approx \max_{f \in Lip_1} \frac{1}{n} \sum_{x_i \sim Data} f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d} f(y_i)$$

so

$$\nu = \operatorname{argmin}_\nu W(\mu, \nu) \approx \operatorname{argmin}_g \max_{f \in Lip_1} \frac{1}{n} \sum_{x_i \sim Data} f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d} f(y_i)$$

Now note:

- when we optimize for g , then f is fixed.

$$\operatorname{argmin}_g \frac{1}{n} \sum_{x_i \sim Data} f(x_i) - \frac{1}{n} \sum_{y_i = Generator(z_i), z_i \sim N(), I_d} f(y_i)$$

- the first term $\frac{1}{n} \sum_{x_i \sim Data} f(x_i)$ does not depend on the generator g , therefore it can be dropped when we keep f fixed and optimize for g

This explains the generator objective.

$R(f)$??? The reason for such term is to approximately enforce $f \in Lip_1$. The objective without it does not enforce this constraint. See <https://openreview.net/pdf?id=B1hYRMbCW>

For a quick idea, if in the one-dimensional case $|f'(x)| \leq 1$, then

$$\begin{aligned} f(x) - f(y) &= \int_{z=y}^{z=x} f'(z) dz \\ \Rightarrow |f(x) - f(y)| &= \left| \int_{z=y}^{z=x} f'(z) dz \right| \leq \int_{z=y}^{z=x} |f'(z)| dz \leq \int_{z=y}^{z=x} 1 dz = x - y \leq |x - y| \\ \Rightarrow |f(x) - f(y)| &\leq |x - y| \end{aligned}$$

For that reason one tries to enforce $\|\nabla f(z_i)\| \leq 1$


1.2 A code example

<https://github.com/jalola/improved-wgan-pytorch>

WGAN-GP (Gulrajani et al.) <https://arxiv.org/pdf/1704.00028.pdf> and WGAN-LP (Petzka et al) <https://openreview.net/pdf?id=B1hYRMbCW> are as of 2018/2019 okay. **GAN training takes days for one hyperparameter setting!!!**

2 Alternatives to GANs

Out of exams:

- variational autoencoders <https://arxiv.org/pdf/1606.05908.pdf> 
- Plug and play generative network (PPGN), (Nguyen et al, 2016) <https://arxiv.org/abs/1612.00005>, Complicated :), related to sampling from a markov chain. Can also generate very realistic images.

3 Modeling Datasets

- the DCGAN paper
- WGAN-GP (Gulrajani et al.) results <https://arxiv.org/pdf/1704.00028.pdf>
- Karras et al., **Progressive Growing of GANs** <https://openreview.net/forum?id=Hk99zCeAb> (weeks???)

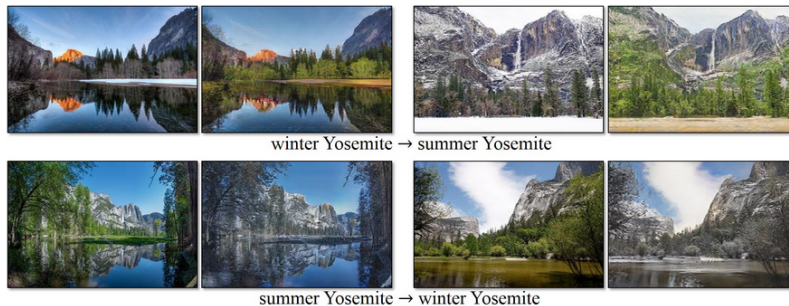
But: very interesting idea to start to learn something simple first, as with human learning!

4 CycleGans and the like

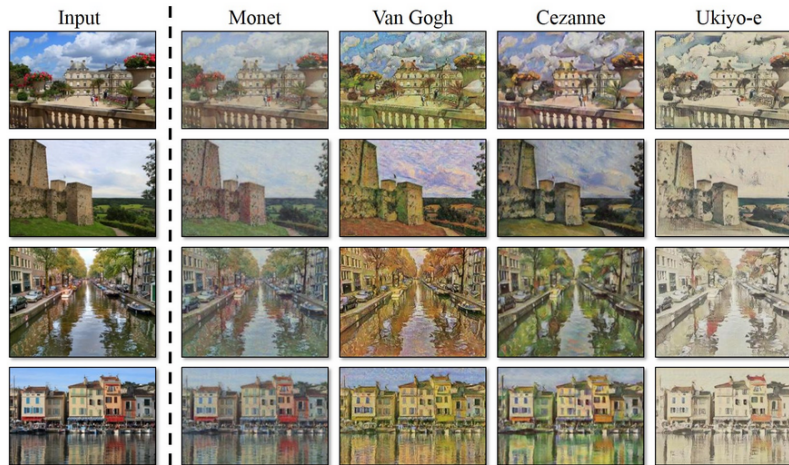
Idea is here: do not input some abstract code $z \in [-1, +1]^D$ into the generator for getting a zebra. Input an image $z = Image$ into the generator to get a zebra! – see CycleGan <https://arxiv.org/abs/1703.10593>, <https://github.com/junyanz/CycleGAN>, <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix> – run the code to get a feeling for the quality of results!



Season Transfer



Collection Style Transfer



Pictures from <https://github.com/junyanz/CycleGAN>

See Fig 3 in <https://arxiv.org/abs/1703.10593>: you have now a generator G mapping from horses to zebras, and one generator F mapping back.

$$G : \text{Horses} \rightarrow \text{Zebras}$$

$$F : \text{Zebras} \rightarrow \text{Horses}$$

Now you can map Horses onto Zebras and back using the two generators.

In practice Domain to Domain Transfer requires additional constraints to make it work in practice!¹

How to train?

Very clear structured idea: Gan losses for generators for both domains + cycle reconstruction losses for both directions.

Cycle reconstruction losses for both directions: eq(2) in the paper. Idea for additional constraint: take an image x , map it to other domain by F , and map it back by G , should result in something similar:

$$F \circ G(x) \approx x$$

$$\|F \circ G(x) - x\|_1 = \text{additional loss term to be added to the loss for keeping it small}$$

¹This is what i expect you to understand here

Do you see what kind of structure you learn by this $F \circ G$ such that $\|F \circ G(x) - x\|_1$ is minimized?

Full objective is in eq(3).

for what applications is CycleGan a bad idea?

Unsupervised Cross Domain Generation <https://arxiv.org/abs/1611.02200>
- maps faces to emoji without supervision.

Tricks:

- mapping $g \circ f$ is decoder-encoder with decoder g and encoder f . Here the encoder is given, not trained as feature layer from a well-trained discriminative neural network. Success comes from this very well given encoder.
- source domain includes the target domain!
- Plus additional constraints: eq (6) an emoji is mapped to almost itself,
- eq(5): a similar autoencoder constraint (compare: one side of a cycle gan) - in target space (emojis),
- 3-class GAN loss (eq 3).
- Implicit tricks: Target domain is more smooth than source, variability of results is more likely acceptable (the other way round is much harder to achieve).

Sketches to Shoes, Handbags to Shoes, Male to Female and the other way round:
<https://arxiv.org/abs/1706.00826>

5 Towards better GANs

Combine them with physical models in 3D, which encode constraints, e.g. how a face should look like when projected onto a plane.