50.021 -AI

Alex

Week 11: Generative Adversarial Networks

[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.]

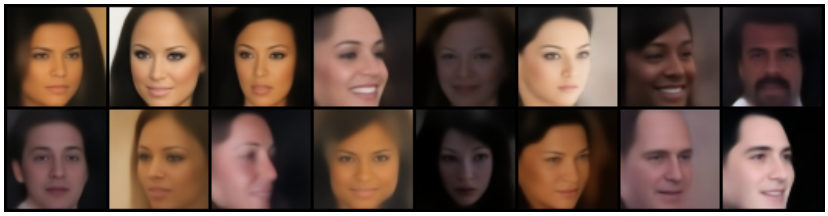
Generative Adversarial Networks

You are expected to understand the general structure of Generative Adversarial Networks, that is what is the goal, what is the function of generator and discriminator, what are the problems when using them, what is the loss formulation.

For a deeper insight, check for example Ian Goodfellow's Tutorial on this topic `https://arxiv.org/pdf/1701.00160.pdf`. Also useful `https://github.com/soumith/ganhacks`

Problem statement:

Given a dataset, create a deep neural network that can generate data samples **similar** to the dataset.



Synthetic faces created with a boundary equilibrium GAN.

discriminator and generator - the GAN approach

The **generator** is a neural network which takes in a d-dimensional random code vector $z \in [-1, +1]^D$ and is expected to output an image.

It has weights, and these weights will be trained as part of the procedure. How to train them? The discriminator helps to understand that

The **discriminator** is a neural network which takes pairs of images as inputs. Each pair contains a real and an image which is the output

of a generator. The discriminator has two classes as output: real and fake. It is trained to label the real images as class 1 ("real"), and the generated ones as fake.

Since we know which images comes from the generator, and which comes from the original dataset, we can automatically label them! Once we have labels, we are ready to train the discriminator.

The set up is - see page 17 / 19 in Goodfellow's tutorial:

- interleaved training: step 1: train the discriminator
  - fix generator weights, generate fake samples, label them as class 0 (fake class)
  - train discriminator weights by pairs of fake and real dataset samples
- interleaved training: step 2: train the generator
  - fix discriminator weights, train generator weights so as to make the prediction of generated samples to have high scores for class 1 (real)

How to model the goals as losses ?

Generator G tries to generate samples such that the discriminator D thinks they are real. Lets assume that the discriminator outputs probabilities $D \in [0,1]$. Then the aim is $D(G(z)) \to 1$. Maximizing a non-negative function, is equal to minimizing its negative logarithm. As loss L_G of the generator one can write:

$$L_G = \text{argmin}_G E_{z \sim P_z}[-1 \cdot \log D(G(z))]$$
$$\hat{L}_G = \text{argmin}_G \sum_{z_i \in minibatch(z-codes)} -1 \cdot \log D(G(z_i))$$

P_z is the sampling distribution for the random input codes for the generator G

The discriminator tries to classify real samples as real that is $D(x) \to 1, x \in data$ and classify the fake samples as fake, that is $D(G(z)) \to 0$. How to arrive at a minimization objective from this ?

One component will be minimizing the negative logarithm of $D(x)$ for x being drawn from the real data, so

$$\text{argmin}_D E_{x \sim P_{data}}[-1 \cdot \log D(x)]$$
$$\text{argmin}_D \sum_{x_i \sim minibatch(data)} -1 \cdot \log D(x_i)$$

As for the goal $D(G(z)) \to 0$, we can write it as: $1 - D(G(z)) \to 1$, and then arrive at the loss of the discriminator L_D

$$L_D = \text{argmin}_D E_{x \sim P_{data}}[-1 \cdot \log D(x)] + E_{z \sim P_z}[-1 \cdot \log(1 - D(G(z)))]$$
$$\hat{L}_D = \text{argmin}_D \sum_{x_i \sim minibatch(data)} -1 \cdot \log D(x_i) + \sum_{z_i \in minibatch(z-codes)} -1 \cdot \log(1 - D(G(z)))$$

This is a two-player adversarial game.

The generator training tries to make generator samples pass the check by the discriminator as real. The discriminator has the real images of the dataset as reference. A well trained generator causes high loss in the discriminator.

The discriminator training tries to make generator samples fail the check by the discriminator. The discriminator training tries to make the discriminator loss low.

Why does this generates samples close to those of the real dataset?[1]

Above is the principled idea. There are many tricks and hacks involved in getting it running `https://github.com/soumith/ganhacks`, or the loss in equation (5) in the cyclegan paper.

[1] It reminds me of co-evolution of virus and host. A stuffed nose virus will not kill you, ebola (likely) does. Stuffed nose viruses have a long history of coexistence with humans, ebola not.

Problems of GANs

Training stability. Convergence of this two player game is not guaranteed. You do not know where training will lead you. What can happen: in every iteration one player reverses the progress of the other player, oscillate but no real progress.

See Nash equilibrium (a solution/game strategy which is optimal for both players, considered that the other player will not change his own solution/strategy) but see `https://arxiv.org/abs/1706.08500`.

Applications

Modeling Datasets

Boundary equilibrium GAN `https://arxiv.org/abs/1703.10717` - see code for in class coding or Fig 18 / page 27 in Goodfellow tutorial. Idea is not image specific.[2]

[2] though network architectures can be

Domain to Domain Transfer

Idea is here: do not input some abstract code for getting a zebra. Input an image to get a zebra! – see CycleGan `https://arxiv.org/`

abs/1703.10593

In practice Domain to Domain Transfer requires additional constraints to make it work in practice![3]

[3] This is what i expect you to understand here

CycleGan: https://arxiv.org/abs/1703.10593 https://github.com/junyanz/CycleGAN - see some results.

Very clear structure: 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:

$$F \circ G(x) \approx x$$
$$\|F \circ G(x) - x\|_1 \longrightarrow \min$$

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).

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.

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

Interpolating between two real datasamples

What is the idea? Given two real faces, handbags, whateverthings x_1, x_2, find input codes which are close to generating them:

$$z_1 = \text{argmin}_z \|x_1 - G(z)\|$$
$$z_2 = \text{argmin}_z \|x_2 - G(z)\|$$

Now you can interpolate between z_1 and z_2 in generator input space (e.g. linear interpolation, but a spherical interpolation can be better sometimes). For simplicity lets to it linearly:
$z_{(k)} = az_1 + (1-a)z_2$, $a \in [0,1]$, e.g. $a = \{1/5, 2/5, 3/5, 4/5\}$
Now map the interpolates $z_{(k)}$ back to an interpolated face. Face-morphing.
Boundary equilibrium GAN - you can do this morphing with the code for in class coding.

Warning: the black art of unsupervised learning is: You get out what you put in – by either constraints or prior probabilities over some statistics. It is not as generic as the power of supervised learning.

Code - no homework

You should see another amazon AMI with code from github from carpedm20 and the CelebA Dataset.

Your tasks:

- Train your own generator using a GPU - I suggest 80k to 100k iterations for okay results.
  `python main.py -dataset=CelebA -use_gpu=True`
- if you get a python3 stringIO error, then do in `trainer.py`: change `import StringIO` to: `from io import StringIO export LC_ALL=C`
- Insight: sample the space, find for every output sample the nearest face in the training dataset. You can do this as a function of gamma parameter in the BEGAN formulation.
- Fun: write an interface that takes two photos of you, preprocesses and morphs into each other.