Generative Adversarial Networks: creating realistic fake examples

Aside

The GAN was invented by [Ian Goodfellow (https://arxiv.org/pdf/1406.2661.pdf)](https://arxiv.org/pdf/1406.2661.pdf) in one night, following a party at a bar !

Our goal is to generate new *synthetic* examples.

Let

- $\mathbf{x}$ denote a *real* example
    - vector of length n
- p_{data} be the distribution of real examples
    - $\mathbf{x} \in p_{\text{data}}$

We will create a Neural Network called the *Generator*

Generator G_{Θ_G} (parameterized by Θ_G) will

- take a vector $\mathbf{z}$ of random numbers from distribution $p_{\mathbf{z}}$ as input
- and output $\hat{\mathbf{x}}$
- a *synthetic/fake* example
    - vector of length n

Let

- p_{model} be the distribution of fake examples

GAN Generator

The Generator will be paired with another Neural Network called the *Discriminator*.

The Discriminator D_{Θ_D} (parameterized by Θ_D) is a binary Classifier

- takes a vector $\tilde{\mathbf{x}} \in p_{\text{data}} \cup p_{\text{model}}$

Goal of Discriminator

$$\begin{array}{lll} D(\tilde{\mathbf{x}}) & = & \text{Real} \quad \text{for } \tilde{\mathbf{x}} \in p_{\text{data}} \\ D(\tilde{\mathbf{x}}) & = & \text{Fake} \quad \text{for } \tilde{\mathbf{x}} \in p_{\text{model}} \end{array}$$

That is

- the Discriminator tries to distinguish between Real and Fake examples

GAN Discriminator

In contrast, the goal of the Generator

Goal of Generator

$$D(\hat{\mathbf{x}}) \quad = \quad \text{Real} \qquad \text{for } \hat{\mathbf{x}} = G_{\Theta_G}(\mathbf{z}) \in p_{\text{model}}$$

That is

- the Generator tries to create fake examples that can fool the Discriminator into classifying as Real

How is this possible ?

We describe a training process (that updates Θ_G and Θ_D)

- That follows an *iterative* game
- Train the Discriminator to distinguish between
    - Real examples
    - and the Fake examples produced by the Generator on the prior iteration
- Train the Generator to produce examples better able to fool the updated Discriminator

Sounds reasonable, but how do we get the Generator to improve it's fakes ?

We will define loss functions

- $\mathcal{L}_G$ for the Generator
- $\mathcal{L}_D$ for the Discriminator

Then we can improve the Generator (parameterized by Θ_G) by Gradient Descent

- updating Θ_G by $-\frac{\partial \mathcal{L}_G}{\partial \Theta_G}$

That is

- The Discriminator will indirectly give "hints" to the Generator as to why a fake example failed to fool

GAN Generator training

GAN Discriminator training

After enough rounds of the "game" we hope that the Generator and Discriminator battle to a stand-off

- the Generator produces realistic fakes
- the Discriminator has only a 50% chance of correctly labeling a fake as Fake

Notation

text	meaning
p_{data}	Distribution of real data
$\mathbf{x} \in p_{\text{data}}$	Real sample
p_{model}	Distribution of fake data
$\hat{\mathbf{x}}$	Fake sample
	$\hat{\mathbf{x}} \notin p_{\text{data}}$
	$\text{shape}(\hat{\mathbf{x}}) = \text{shape}(\mathbf{x})$
$\tilde{\mathbf{x}}$	Sample (real of fake)
	$\text{shape}(\tilde{\mathbf{x}}) = \text{shape}(\mathbf{x})$
D_{Θ_D}	Discriminator NN, parameterized by Θ_D
	Binary classifier: $\tilde{\mathbf{x}} \mapsto \{\text{Real}, \text{Fake}\}$
	$D_{\Theta_D}(\tilde{x}) \in \{\text{Real}, \text{Fake}\}$ for $\text{shape}(\tilde{\mathbf{x}}) = \text{shape}(\mathbf{x})$
$\mathbf{z}$	vector or randoms with distribution $p_{\mathbf{z}}$
G_{Θ_G}	Generator NN, parameterized by Θ_G
	$\mathbf{z} \mapsto \hat{\mathbf{x}}$
	$\text{shape}(G(\mathbf{z})) = \text{shape}(\mathbf{x})$
	$G(\mathbf{z}) \in p_{\text{model}}$

Loss functions

The goal of the generator can be stated as

- Creating p_{model} such that
- $p_{\text{model}} \approx p_{\text{data}}$

There are a number of ways to measure the dis-similarity of two distributions

- KL divergence
    - equivalent to Maximum Likelihood estimation
- Jensen Shannon Divergence (JSD)
- Earth Mover Distance (Wasserstein GAN)

The original paper choose the minimization of the KL divergence, so we illustrate with that measure.

To be concrete. let the Discriminator uses labels

- 1 for Real
- 0 for Fake

The Discriminator tries to maximize

$$-\mathcal{L}_D = \begin{cases} \log D(\tilde{\mathbf{x}}) & \text{when } \tilde{\mathbf{x}} \in p_{\text{data}} \\ 1 - \log D(\tilde{\mathbf{x}}) & \text{when } \tilde{\mathbf{x}} \in p_{\text{model}} \end{cases}$$

That is

- Classify real $\mathbf{x}$ as Real
- Classify fake $\hat{\mathbf{x}}$ as Fake

The per-example Loss for the Generator is

$$\mathcal{L}_G = 1 - \log D(G(\mathbf{z}))$$

which is achieved when the fake example

$$D(G(\mathbf{z})) = 1$$

That is

- the Discriminator mis-classifies the fake example as Real

So the iterative game seeks to solve a minimax problem

$$\min_G \max_D \left(\mathbb{E}_{\mathbf{x} \in p_{\text{data}}} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \in p_z} (1 - \log D(G(\mathbf{z})))\right)$$

- D tries to
    - make $D(\mathbf{x})$ big: correctly classify (with high probability) real $\mathbf{x}$
    - and $D(G(\mathbf{z}))$ small: correctly classify (with low probability) fake $G(\mathbf{z})$
- G tries to
    - make $D(G(\mathbf{z}))$ high: fool D into a high probability for a fake

Note that the Generator improves

- by updating Θ_G
- so as to increase $D(G(\mathbf{z}))$
    - the mis-classification of the fake as Real

Training

We will train Generator G_{Θ_G} Discriminator D_{Θ_D} by turns

- creating sequence of updated parameters
    - $\Theta_{G,(1)} \ldots \Theta_{G,(T)}$
    - $\Theta_{G,(1)} \ldots \Theta_{D,(T)}$
- Trained *competitively*

Competitive training

Iteration t

- Train $D_{\Theta_{D,(t-1)}}$ on samples
  - $\tilde{\mathbf{x}} \in p_{\text{data}} \cup p_{\text{model},(t-1)}$
    - where $G_{\Theta_{G,(t-1)}}(\mathbf{z}) \in p_{\text{model},(t-1)}$
  - Update $\Theta_{D,(t-1)}$ to $\Theta_{D,(t)}$ via gradient $\frac{\partial \mathcal{L}_D}{\partial \Theta_{D,(t-1)}}$
    - D is a maximizer of $\int_{\mathbf{x} \in p_{\text{data}}} \log D(\mathbf{x}) + \int_{\mathbf{z} \in p_{\mathbf{z}}} \log \left(1 - D(G(\mathbf{z})) \right)$
- Train $G_{\Theta_{G,(t-1)}}$ on random samples $\mathbf{z}$
  - Create samples $\hat{\mathbf{x}}_{(t)} \in G_{\Theta_{G,(t-1)}}(\mathbf{z}) \in p_{\text{model}}$
  - Have Discriminator $D_{\Theta_{D,(t)}}$ evaluate $D_{\Theta_{D,(t)}}(\hat{\mathbf{x}}_{(t)})$
  - Update $\Theta_{G,(t-1)}$ to $\Theta_{G,(t)}$ via gradient $\frac{\partial \mathcal{L}_G}{\partial \Theta_{G,(t-1)}}$
    - G is a minimizer of $\int_{\mathbf{z} \in p_{\mathbf{z}}} \log(1 - D(G(\mathbf{z})))$
      - i.e., want $D(G(\mathbf{z}))$ to be high
  - May update G multiple times per update of D

Training code for a simple GAN

Here (https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/generative/ipynb/dcgan_overriding_train_step.ipynb#scrollTo=A
is the code for the training step of a simple GAN.

Code

- GAN on Colab (https://keras.io/examples/generative/dcgan_overriding_train_step/)
- Wasserstein GAN with Gradient Penalty (https://keras.io/examples/generative/wgan_gp/#create-the-wgangp-model)