Deep Convolutional GANs

: Meaning of Latent Space

ISL Lab Seminar
Hansol Kang
Contents

Review of GAN

DCGAN

Experiment

Summary
Review of GAN

• Adversarial nets

"Generative Adversarial Networks"

Goal

Method

Vs

1) Global Optimality of \( p_g = p_{data} \)
2) Convergence of Algorithm
DCGAN*  
A FEW MOMENTS LATER
• Introduction

“I have the strongest MLP army.”

“I have too.”

“What are they doing?”

“We have a better CNN than MLP”

Vanilla GAN

DCGAN

DCGAN

• Contributions

Generating Natural Image

Vector arithmetic properties

Deep Convolutional GANs

Filter Visualization

Image Classification using D
DCGAN

- Contributions

- Generating Natural Image
- Vector arithmetic properties
- Deep Convolutional GANs
- Filter Visualization
- Image Classification using D
DCGAN

- Approach and Model Architecture

Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).

Use batchnorm in both the generator and the discriminator.

Remove fully connected hidden layers for deeper architectures.

Use ReLU activation in generator for all layers except for the output, which uses Tanh.

Use LeakyReLU activation in the discriminator for all layers.
DCGAN

- Approach and Model Architecture

Strided Convolution

Fractional Convolution (Transposed Convolution)
DCGAN

- Approach and Model Architecture

Batch Normalization

\[
\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)
\]

Except for these layers.

Output layer of Generator
Input layer of Discriminator
DCGAN

• Approach and Model Architecture

No fully connected layer

Classical CNN

GAP (Global Average Pooling)

http://nmhkahn.github.io/Casestudy-CNN
DCGAN

- Approach and Model Architecture

No fully connected layer

![DCGAN Diagram](https://raw.githubusercontent.com/znxlwm/pytorch-MNIST-CelebA-GAN-DCGAN/master/pytorch_DCGAN.png)
DCGAN

- Approach and Model Architecture

ReLU, Tanh, LeakyReLU

- Generator: ReLU, Tanh
- Discriminator: LeakyReLU, Sigmoid

http://gmelli.org/RKB/Rectified_Linear_Unit_(ReLU)_Activation_Function
DCGAN

• Details of Adversarial Training

  • Mini-batch stochastic gradient descent (SGD); mini-batch size of 128
  • All weights initialized from a zero-centered Normal distribution with standard deviation 0.02
  • Leaky slope 0.2
  • Adam optimizer; lr = 0.0002, beta1 = 0.9, beta2 = 0.5
DCGAN

• Details of Adversarial Training

LSUN dataset

1 epoch
DCGAN

- Details of Adversarial Training

LSUN dataset

5 epochs
**DCGAN**

- **Empirical Validation of DCGANs Capabilities**

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Accuracy (400 per class)</th>
<th>max # of features units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Layer K-means</td>
<td>80.6%</td>
<td>63.7% (±0.7%)</td>
<td>4800</td>
</tr>
<tr>
<td>3 Layer K-means Learned RF</td>
<td>82.0%</td>
<td>70.7% (±0.7%)</td>
<td>3200</td>
</tr>
<tr>
<td>View Invariant K-means</td>
<td>81.9%</td>
<td>72.6% (±0.7%)</td>
<td>6400</td>
</tr>
<tr>
<td><strong>Exemplar CNN</strong></td>
<td>84.3%</td>
<td>77.4% (±0.2%)</td>
<td>1024</td>
</tr>
<tr>
<td>DCGAN (ours) + L2-SVM</td>
<td>82.8%</td>
<td>73.8% (±0.4%)</td>
<td>512</td>
</tr>
</tbody>
</table>

- CIFAR-10
- Classification
- Domain robustness
DCGAN

- Empirical Validation of DCGANs Capabilities

Table 2: SVHN classification with 1000 labels

<table>
<thead>
<tr>
<th>Model</th>
<th>error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN</td>
<td>77.93%</td>
</tr>
<tr>
<td>TSVM</td>
<td>66.55%</td>
</tr>
<tr>
<td>M1+KNN</td>
<td>65.63%</td>
</tr>
<tr>
<td>M1+TSVM</td>
<td>54.33%</td>
</tr>
<tr>
<td>M1+M2</td>
<td>36.02%</td>
</tr>
<tr>
<td>SWWAE without dropout</td>
<td>27.83%</td>
</tr>
<tr>
<td>SWWAE with dropout</td>
<td>23.56%</td>
</tr>
<tr>
<td>DCGAN (ours) + L2-SVM</td>
<td>22.48%</td>
</tr>
<tr>
<td>Supervised CNN with the same architecture</td>
<td>28.87% (validation)</td>
</tr>
</tbody>
</table>

SVHN (Street View House Numbers) dataset
DCGAN

- Investigating and Visualizing The Internals of The Networks

Walking in the latent space
DCGAN

• Investigating and Visualizing The Internals of The Networks (cont.)

Visualizing the discriminator features

![Random filters vs Trained filters](image)
DCGAN

• Investigating and Visualizing The Internals of The Networks (cont.)

Forgetting to draw certain objects

Noise(z)

Latent code

Filters(Conv)

Generation

YOU DIED

in charge of windows

in charge of beds

in charge of lamps

in charge of doors
DCGAN

• Investigating and Visualizing The Internals of The Networks (cont.)

Forgetting to draw certain objects
DCGAN

- Investigating and Visualizing The Internals of The Networks (cont.)

Vector arithmetic on face samples

smiling woman
neutral woman
neutral man

= smiling man
DCGAN

- Investigating and Visualizing The Internals of The Networks (cont.)

Vector arithmetic on face samples

![Diagram showing vector arithmetic on face samples](image-url)
DCGAN

• Investigating and Visualizing The Internals of The Networks (cont.)

Vector arithmetic on face samples

Results of doing the same arithmetic in pixel space
DCGAN

• Investigating and Visualizing The Internals of The Networks (cont.)

Vector arithmetic on face samples
Experiment

• Code

GAN_PyTorch

Various GAN implementations based on PyTorch. This project consists of simple and standard models with a relatively short code length, and only simple functions are implemented. The standard version has various functions rather than the simple version. It also provides a UI (the standard version is loaded and executed). In fact, I don’t know if UI is comfortable.

Implementation list

• Vanilla GAN: Simple | Standard & UI
• DCGAN: Simple |
• InfoGAN: Simple |

Experiment Environment

• Windows 10 Enterprise
• Intel i7-3770k
• RAM 12.0 GB
• NVIDIA GTX TITAN
• Python 3.6.4
• PyTorch 0.4.0
• torchvision 0.2.1

Vanilla_Standard.py and for_UI.py

• The UI supports batch size, epoch size, learning rate, and dataset settings.
• Save the log file as csv.

https://github.com/messy-snail/GAN_PyTorch
Experiment

• Code

class Generator(nn.Module):
    def __init__(self, latent_sz):
        super(Generator, self).__init__()
        self.tconv1 = nn.ConvTranspose2d(latent_sz, 1024, 4, 1, 0)
        self.tconv2 = nn.ConvTranspose2d(1024, 512, 4, 2, 1)
        self.tconv3 = nn.ConvTranspose2d(512, 256, 4, 2, 1)
        self.tconv4 = nn.ConvTranspose2d(256, 128, 4, 2, 1)
        self.tconv5 = nn.ConvTranspose2d(128, 3, 4, 2, 1)

        self.bn1 = nn.BatchNorm2d(1024)
        self.bn2 = nn.BatchNorm2d(512)
        self.bn3 = nn.BatchNorm2d(256)
        self.bn4 = nn.BatchNorm2d(128)

    def forward(self, input):
        x = F.relu(self.bn1(self.tconv1(input)))
        x = F.relu(self.bn2(self.tconv2(x)))
        x = F.relu(self.bn3(self.tconv3(x)))
        x = F.relu(self.bn4(self.tconv4(x)))
        x = F.tanh(self.tconv5(x))

        return x

class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, 4, 2, 1)
        self.conv2 = nn.Conv2d(128, 256, 4, 2, 1)
        self.conv3 = nn.Conv2d(256, 512, 4, 2, 1)
        self.conv4 = nn.Conv2d(512, 1024, 4, 2, 1)
        self.conv5 = nn.Conv2d(1024, 1, 4, 1, 0)

        self.bn2 = nn.BatchNorm2d(256)
        self.bn3 = nn.BatchNorm2d(512)
        self.bn4 = nn.BatchNorm2d(1024)

    def forward(self, input):
        x = F.leaky_relu(self.conv1(input), 0.2)
        x = F.leaky_relu(self.bn2(self.conv2(x)), 0.2)
        x = F.leaky_relu(self.bn3(self.conv3(x)), 0.2)
        x = F.leaky_relu(self.bn4(self.conv4(x)), 0.2)
        x = F.sigmoid(self.conv5(x))

        return x

https://github.com/messy-snail/GAN_PyTorch
Experiment

• Results#1 CelebA

Ground Truth

Results are cherry picked

Vanilla GAN :  
DCGAN :  

Still have this sample

Epoch 1
Epoch 5
Epoch 100
Epoch 1
Epoch 5
Epoch 30
Experiment

- Results#2 LSUN)

Ground Truth

Results are cherry picked

Vanilla GAN: 
DCGAN: 

Epoch 1  Epoch 5  Epoch 12

Epoch 1  Epoch 2  Epoch 5

Epoch 1  Epoch 2  Epoch 5
Experiment

• Results#3 Korean Idol – Transfer trial

Ground Truth

• I used weights and biases generated by celebA learning.

• I wanted the effect of transfer learning but failed.

Maybe these factors (Asian, cropping image)
Experiment

• Results#4 Korean Idol

Ground Truth

• 10000 images

Insufficient data set

Epoch 1
Epoch 5
Epoch 30
Epoch 50
Epoch 100
Epoch 150
Summary

- Stable set of architectures for training generative adversarial networks
- Good representations of images for supervised learning and generative modeling
- Sometimes collapse a subset of filters to a single oscillating mode
- Latent code has a special meaning, not a simple noise component.

[Instability of GAN]
## Future work

<table>
<thead>
<tr>
<th>Paper Review</th>
<th>Proposed Model</th>
<th>Tips</th>
<th>Mathematical Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>□ Vanilla-GAN</td>
<td>□ SpyGAN</td>
<td>□ Document</td>
<td>□ Information theory</td>
</tr>
<tr>
<td>□ DCGAN</td>
<td>(working title)</td>
<td>□ Programming</td>
<td></td>
</tr>
</tbody>
</table>

- □ InfoGAN
- □ Unrolled GAN
- □ Wasserstein GAN
- □ LS GAN
- □ BEGAN
- □ Pix2Pix
- □ Cycle GAN
Appendix

• Issues at the VAE Seminar (18.07.23)

✓ Issue#1 Performance of VAE and GAN
✓ Issue#2 Log likelihood
✓ Issue#3 Dimension of latent code
✓ Issue#4 Why manifold?

Durk Kingma

*Machine Learning researcher at OpenAI*

2. Auto-Encoding Variational Bayes

- Mathematically very difficult papers

Intuitive explanation

: I refer to this video

오토인코더의 모든 것

https://www.youtube.com/watch?v=o_peo6U7IRM
Appendix

• Issue #1 Performance of VAE and GAN

“Compared to GAN, **VAE is relatively blurred** and I do not know why.”

“Cost function”

**VAE** \( \min L(\phi, \theta, x) \)

\[
L(\phi, \theta, x) = \text{Reconstruction Error} + \text{Regularization}
\]

**GAN** \( \min \max V(G, D) \)

\[
V(G, D) = \approx \text{D Loss} + \approx \text{G Loss}
\]
Appendix

• Issue #1 Performance of VAE and GAN

VAE Loss = Recon. Error + Regularization

GAN Loss = G_Loss + D_Loss

VAE vs. GAN
1. Optimize
2. Image Quality
3. Generalization
Appendix

• Issue #2 Log likelihood

Question about log likelihood

“Summation and monotonically increasing”

MLE (Maximum Likelihood Estimation) : Unknown parameter estimation from observation

\[ \hat{\theta} = \arg \max_{\theta} p(y | \theta) \]

\[ = \arg \max_{\theta} \prod_{i} p(y_i | \theta) \rightarrow \arg \max_{\theta} \log \left( \prod_{i} p(y_i | \theta) \right) = \arg \max_{\theta} \sum_{i} \log p(y_i | \theta) \]

cf. \[ \log(x) : \text{monotonically increasing function} \]

eg. Gaussian Distribution

Mean and Std

Generation model

\[ \arg \max_{\theta} \sum_{i} \log p_{\theta}(x_i) \]
Appendix

• Issue #3 Dimension of latent code

“Is the latent code dimension always small?”

“Yes”

AE, What’s this? Dimension reduction

Sparse AE

High Low

Interested
Appendix

• Issue #4 Why manifold?

What’s the manifold and Why explain the manifold?

“Concept of manifold and Difference of between AE and VAE”

Concept of manifold

Subspace = Manifold

High

Low

Purpose of AE and VAE

Purpose of AE: Manifold Learning

Unsupervised Learning

Purpose of VAE: Generative Model

Unsupervised Learning

Assumption (manifold hypothesis)

Uniform sampling
Appendix

- PyTorch (Variable length inputs)

**CIFAR-10**
- Shape = {Size} torch.Size([128, 3, 32, 32])
- Shape = {Size} torch.Size([128, 64, 16, 16])
- Shape = {Size} torch.Size([128, 16384])

**CelebA**
- Shape = {Size} torch.Size([128, 3, 218, 178])
- Shape = {Size} torch.Size([128, 64, 109, 89])
- Shape = {Size} torch.Size([128, 620864])