Deep Learning by Example on Biowulf

Class #4. Generative Adversarial Networks (GANs) and their application to biological data synthesis

Gennady Denisov, PhD
**Intro and goals**

*I. Goodfellow et al., Generative Adversarial Nets. NIPS Proc. 2014*

**What is a GAN?**
- A composite network comprising 2 subnetworks: Generator and Discriminator
- The **G** produces fake data from scratch/noise; learns to **trick** the **D**
- The **D** compares fake data against the true data; learns to **expose** the **G**

**Features:**
*Generative model:* the goal is to generate new, **synthetic** instances of data that can pass for real data

**Examples:**
*Generating face images*  
*BioGANs: GANs for biological image synthesis*
Examples overview

- unsupervised ML
- generative model, functionally similar to VAE
- composite network comprising two subnetworks
- the two subnetworks are trained interactively, by playing a minimax game
- GAN flavors: GAN, DCGAN, WGAN, WGAN-GP,…
Deep Convolutional GAN (DCGAN): a simple example

tensors, units, layers, parameters, hyperparameters, convolution


RNN/1D CNN prototype example from class #2:

**Input:** a set of training sequences of 0’s and 1’s with **binary labels** assigned depending on whether or not a certain (unknown) **motif** is present

**Example:** 01011100101

**Task:** predict the label, or the occurrence of the **unknown** motif, in new, previously unseen sequences.

DCGAN prototype example:

**Input:** a training set of only “good” sequences of 0’s and 1’s, i.e. **all** of them contain a certain motif

**Example:** 01011001100110011000111

**Task:** learn what makes all of the training sequences “good” and then generate new “good” sequences from scratch.

**Challenge:** no labels; only positive examples.

Architecture guidelines for stable DCGANs:

- Use **convolutions** (D) and **transposed convolutions** (G) instead of pooling layers
- Use **BatchNormalization** in both the G and the D.
- Avoid Dense/Fully Connected hidden layers
- ReLU activation in G for all layers except for the output and LeakyReLU activation in D.
The transposed convolution (a.k.a. deconvolution, or fractional-strided convolution)

convolution, transposed convolution, stride, kernel size, padding

V.Dumoulin, F.Visin - A guide to convolution arithmetic for deep learning (2018)

Conv2D

<table>
<thead>
<tr>
<th>input size</th>
<th>i = 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>output size</td>
<td>o = 2</td>
</tr>
<tr>
<td>kernel size</td>
<td>k = 3</td>
</tr>
<tr>
<td>strides</td>
<td>s = 2</td>
</tr>
<tr>
<td>padding</td>
<td>p = 0 ('valid')</td>
</tr>
</tbody>
</table>

Conv2DTranspose

<table>
<thead>
<tr>
<th>input size</th>
<th>i' = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>output size</td>
<td>o' = 5</td>
</tr>
<tr>
<td>kernel size</td>
<td>k' = 3</td>
</tr>
<tr>
<td>strides</td>
<td>s' = 2</td>
</tr>
<tr>
<td>padding</td>
<td>p' = 2</td>
</tr>
</tbody>
</table>

Conv2D:

\[ i + 2p = k + s(o - 1) \]

'valid' padding: \( p = 0 \)
'same' padding: \( o = \text{round}(i / s) \)

Conv2DTranspose:

\[ o' = i' + (i' - 1)(s' - 1) + 2p' - k' + 1 \]

'valid' padding: \( p' = k' - 1 \)
'same' padding: \( o' = i' * s' \)
The simple GAN training code: (1) header, (2) getting data and (3) defining a model

motif, discriminator, compile, loss, optimizer

(1) Header:
- general Python imports
- Numpy imports
- Keras library imports

(2) Get data
- motif
- noise_len

(3) Define a model
- discriminator (D)
The simple GAN training code: (3) defining a model (cont.) and (4) running the model

Lambda, BatchNormalization, Conv1DTranspose, generator, trainable, combined_model, train_on_batch, epoch, save_weights

(3) Define a model
- Conv1DTranspose
- Lambda
- BatchNormalization
- generator (G)
- combined model (GAN)
GAN = D(G(z))

(4) Run the model
- train_on_batch,
- epoch
- save_weights
The GAN optimization objective

Discriminator:
\[ \log D(x) + \log(1 - D(G(z))) \rightarrow \max \]

Generator:
\[ \log (1 - D(G(z))) \rightarrow \min \]

The minimax optimization objective:
\[
\min_G \max_D E_{data} \{ \log D(x) \} + E_{noise} \{ \log (1 - D(G(z))) \}
\]

The simple GAN prediction code

load_weights, predict

Header:
- general Python imports
- Numpy imports

Get data

Define a model

Run the model
- load_weights
- noise
- predict
How to run the simple GAN application on Biowulf?

### Executables

- simple_gan_train.py
- simple_gan_predict.py
- simple_wgan_train.py

```bash
$ sinteractive --gres=gpu:p100:1 --mem=4g
$ module load DLBio/class4
...
$ ls $DLBIO_BIN
simple_gan_train.py simple_gan_predict.py simple_wgan_train.py
```

Using TensorFlow backend.

```plaintext
100: D_loss 0.5485115051269531 G_loss 0.2828627526760101
200: D_loss 0.3392027914524078 G_loss 0.0490138642489910
300: D_loss 0.176022976636886 G_loss 0.0042200786992908
...
3000: D_loss 0.00000000026483797 G_loss 0.0000000000000000
```

```bash
$ ls weights.gan_generator.h5
weights.gan_ator.h5
```

```bash
$ simple_gan_predict.py
...
0: G_count= 1 motif_start= 21 R_count= 0
726: G_count= 727 motif_start= 7 R_count= 9
727: G_count= 728 motif_start= 111 R_count= 9
...
999: G_count= 1000 motif_start= 39 R_count= 10
Motif count in generated sequence: 1000/1000
Motif count in random sequence: 10/1000
```

### A checkpoint file

```bash
weights.gan_generator.h5
```

### Counts of motifs produced by Generator or randomly

```plaintext
100: C_loss -0.0327724255621433 G_loss -0.0370155312120914
200: C_loss -0.0348766297101974 G_loss -0.0597182549536228
...
```
Example 4. BioGANs: GANs for Biological Image Synthesis

A. Osokin e.a. IEEE Int. Conf. on Computer Vision (ICCV), 2017
https://github.com/aosokin/biogans
https://hpc.nih.gov/apps/biogans.html

Fission yeast cells

Bgs4 + Bgs4

Alp14 + Bgs4

Arp3 + Bgs4

Cki2 + Bgs4

Mkh1 + Bgs4

Sid2 + Bgs4

Twa1 + Bgs4

**Biological task:** investigate how the polarity factors interact with one another

**Computational task:** train a GAN on available data and generate synthetic images that visualize a synchronized distribution of multiple polarity factors, together with growth factor Bgs4 at the same stage of a cell cycle (i.e. the data that cannot be produced experimentally)

**Data:** the Localization Interdependency network (**LIN**) dataset

**The BioGANs pipeline (reimplemented in Keras from PyTorch):**

biogans_train.py → biogans_predict.py → biogans_visualize.py
An overview of the BioGANs training code

*The Keras source code:*
biogans_train.py
biogans.predict.py
biogans_visualize.py
options.py, dataloader.py, models.py, gans.py

**Header**
- import statements
- parsing the command line options

**Getting data**
- LIN dataset

**Define a (network) model**
- models available:
  DCGAN,
  DCGAN-separable,
  DCGAN-starshaped

**Run the model**
- GAN algorithms:
  (traditional) GAN
  WGAN
  WGAN-GP
- optimizer: RMSProp

```python
import os, sys, random
import numpy as np
import gans
from dataloader import get_data
from options import parse_training_arguments, process_options
from models import get_network_models

import tensorflow as tf
from tensorflow.keras.optimizers import Adam, RMSprop

# Header
if __name__ == '__main__':
    opt = parse_training_arguments()
    opt, DCGAN_model, gan_algorithm, optimizer = process_options("train", opt)

    # Load data
    dataset, opt.n_classes = get_data(opt, "train")

    # Define a model
    os.environ['CUDA_VISIBLE_DEVICES'] = "0"
    if opt.num_gpus > 1:
        for j in range(1, opt.num_gpus):
            os.environ['CUDA_VISIBLE_DEVICES'] += "," + str(j)

    with tf.device("/cpu:0"):
        random_seed(opt.random_seed)  # Fix random seed
        netG, netD = get_network_models(DCGAN_model, opt, opt.red_portion)

    # Run the model
    if gan_algorithm == "GAN":
        gans.GAN(netG, netD, opt).train(dataset, opt)
    elif gan_algorithm == "WGAN":
        gans.WGAN(netG, netD, opt).train(dataset, opt)
    elif gan_algorithm == "WGAN-GP":
        gans.WGAN_GP(netG, netD, opt).train(dataset, opt)
    else:
        sys.exit("Undefined gan_algorithm: " + gan_algorithm + "\n")
```

[https://hpc.nih.gov/apps/biogans.html](https://hpc.nih.gov/apps/biogans.html)
BioGANs data: the Localization Interdependency Network (LIN) dataset

J. Dodgson et al, https://www.biorxiv.org/content/10.1101/116749v1.full

Features:

- **2D fluorescence microscopy images** of Fission yeast cells, each \((7 \div 14) \times 4 \, \mu \text{m}\)
- 2-channel images of size is **48 x 80 pixels** (1 pixel = 100 nm)
- **Red channel** = protein **Bgs4**, localizes in the **area of active growth**
- **Green channel** = any of 41 different **polarity factors** that **define a cell geometry**
- **170,000 images** for 41 polarity factors available in the LIN dataset.
- the BioGANs application focuses on **Bgs4** and **6 polarity factors** Alp14, Arp3, Cki2, Mkh1, Sid2 and **Tea1**, with total **26,909 images**
The BioGANs DCGAN (i.e. basic) model
BioGANs generator architectures: DCGAN, DCGAN-separable and DCGAN-starshaped

A. Osokin e.a. IEEE Int. Conf. on Computer Vision (ICCV), 2017

How to generate multiple green channels given a signal in a red channel?
The Wasserstein GAN (WGAN)


Problem with traditional GAN: vanishing gradients
due to the last/sigmoid layer in the Discriminator:
\[ D(I, w) = \sigma(F(I, w)) \Rightarrow \nabla_w D = \sigma' \cdot \nabla_w F \rightarrow 0 \text{ at saturation} \]

WGAN ideas:
- get rid of the \( \sigma \) layer \( \Rightarrow \) can no longer use the BCE loss; the \( D \) becomes \( F \)
- rename \( F \) to critic: it will output a score \( s \), not a probability
- use the Earth Mover’s distance (EMD) between the distributions
  of the critic scores \( P_{Data}(s) \) and \( P_{Gen}(s) \) as a new loss function

EMD, a.k.a. Wasserstein loss = \text{minimum} amount of work
to transform one distribution to another

\[ \text{Binary cross entropy loss:} \]
\[ BCE = - \frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log(p_i(w)) + (1-y_i) \cdot \log(1-p_i(w)) \]

\[ \text{Wasserstein loss:} \]
\[ EMD \approx - E [s_{Data} \cdot s_{Gen}] \]

\[ EMD \rightarrow \text{min} \; \text{forces the two distributions} \]
\[ \text{to have maxima at the same locations} \]
WGANS vs WGAN with gradient penalty (WGAN-GP)


How can we limit the growth of $F$ to avoid instability?

(Traditional) GAN:
use sigmoid activation: $D(I) = \sigma(F(I))$

WGAN features:
1. use EMD loss
2. clip all weights after each epoch
   (usually, $c = 0.01$)
3. rename Discriminator to Critic
4. use RMSProp optimizer with $lr = 0.00005$

WGAN-GP features:
1. (3), (4)
2. penalize the gradient (usually, $\lambda = 10$)

$\text{WGAN-GP loss} = EMD + \lambda \cdot \| \nabla F \|$
How to run the BioGANs application on Biowulf?

https://hpc.nih.gov/apps/biogans.html

dsinteractive --mem=40g --gres=gpu:p100:1,lscratch:10

module load biogans

cp $BIOGANS_DATA/*

ls $BIOGANS_SRC
biogans_predict.py  biogans_visualize.py  gans.py  options.py  utils.py
biogans_train.py  dataloader.py  models.py  __pycache__

biogans_train.py  -d <data_folder>  [-m <network_model>]  [-a <gan_algorithm>]

# NOTES:
# network_model = DCGAN (default), DCGAN-separable or DCGAN-starshaped
# gan_algorithm = GAN (default), WGAN or WGAN-GP

Example:
biogans_train.py  -d data/LIN_Normalized_WT_size-48-80_train/Alp14

biogans_predict.py  -i <input_file>  [ other options ]

Example:
biogans_predict.py  -i checkpoints/model.generator.Alp14.DCGAN.GAN.1.h5

biogans_visualize.py  -i <input_file>

Example:
biogans_visualize.py  -i checkpoints/model.generator.Alp14.DCGAN.GAN.1.h5
The Root Mean Squared propagation (RMSprop) optimizer

Basic gradient descent formula for updating weights

\[ w_{t+1} = w_t - \gamma \cdot \nabla w J(w_t) \]

- \( w \) = vector of weights
- \( t \) = update #
- \( \gamma \) = learning rate (a hyperparameter)
- \( \nabla w J \) = gradient of the loss with respect to weights

The RMSprop-based formula for updating weights

\[ w_{t+1} = w_t - \frac{\gamma}{\sqrt{E[\nabla w J(w_t)^2] + \epsilon}} \cdot \nabla w J(w_t) \]

- \( \epsilon \) = small parameter

How to compute the running average:

\[
E[\nabla w J(w) \|^2]_t = \beta \cdot E[\nabla w J(w) \|^2]_{t-1} + (1 - \beta) \cdot \nabla w J(w_t)^2
\]

\( \beta \sim 0.9 \)
Conclusions

1) Intro using a simple example
   - simple GAN that synthesizes a sequence containing a certain motif:
     Discriminator is the same as the network from class #2
     Generator network produces a sequence from random noise
   - the Conv2DTranspose (transposed convolution, a.k.a. deconvolution) layer
   - the BatchNormalization layer
   - the train_on_batch method

2) The BioGANs application:
   - DCGAN, DCGAN-separable and DCGAN-starshaped models
   - WGAN (Wasserstein GAN) and the Earth Mover’s distance (EMD)
   - WGAN-GP: the Wasserstein GAN with gradient penalty loss

3) Other topics:
   - the gradient descent-based optimization algorithm RMSprop