Deep Convolutional Generative Adversarial Networks (DCGAN)

ECE57000: Artificial Intelligence
David I. Inouye

DCGAN: Randomly generated bedrooms show slightly odd but almost realistic bedrooms

DCGAN can show interpolation between imaginary hotel rooms

Removing certain filters can modify the generated images (in this case, a “window” filter).

Figure 6: Top row: un-modified samples from model. Bottom row: the same samples generated with dropping out ”window” filters. Some windows are removed, others are transformed into objects with similar visual appearance such as doors and mirrors. Although visual quality decreased, overall scene composition stayed similar, suggesting the generator has done a good job disentangling scene representation from object representation. Extended experiments could be done to remove other objects from the image and modify the objects the generator draws.

Simple vector arithmetic in latent space of DCGAN can generate new faces

How do we learn this implicit generative model?
Train two deep networks simultaneously

DCGAN requires transposed convolutions, BatchNorm, and a few other training tricks

- Transposed convolutions (for upsampling)

- BatchNorm (for stabilizing training)

- A few tricks
DCGAN generator upsamples the size of the image in multiple stages.

Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution $Z$ is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a $64 \times 64$ pixel image. Notably, no fully connected or pooling layers are used.
Transposed convolution can be used to upsample an tensor/image to have higher dimensions

- Also known as:
  - Fractionally-strided convolution
  - Improperly, deconvolution
- Remember: Convolution is like matrix multiplication
  \[ y = x \ast f \iff \text{vec}(y) = A_f \text{vec}(x) \]
- Transpose convolution is the transpose of \( A_f \):
  \[ \text{vec}(y) = A_f^T \text{vec}(x) \]
Convolution operator with corresponding matrix

\[(\ast)\]

https://github.com/naokishibuya/deep-learning/blob/master/python/transposed_convolution.ipynb
Transposed convolution operator with corresponding matrix

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 \\
4 & 2 & 0 & 0 \\
0 & 4 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 2 & 1 \\
3 & 1 & 4 & 2 \\
0 & 3 & 0 & 4 \\
1 & 0 & 1 & 0 \\
2 & 1 & 1 & 1 \\
4 & 2 & 3 & 1 \\
0 & 4 & 0 & 3 \\
0 & 0 & 1 & 0 \\
0 & 0 & 2 & 1 \\
0 & 0 & 4 & 2 \\
0 & 0 & 0 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
2 & 0 & 0 & 0 \\
7 & 1 & 0 & 0 \\
14 & 2 & 0 & 0 \\
19 & 3 & 0 & 0 \\
2 & 0 & 1 & 0 \\
10 & 1 & 2 & 1 \\
22 & 3 & 4 & 2 \\
6 & 0 & 0 & 3 \\
4 & 1 & 1 & 0 \\
16 & 0 & 2 & 1 \\
10 & 0 & 4 & 2 \\
6 & 0 & 0 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
1 & 3 & 0 & 0 \\
2 & 3 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
2 & 7 & 14 & 12 \\
6 & 16 & 21 & 21 \\
6 & 14 & 23 & 21 \\
4 & 11 & 22 & 12 \\
\end{array}
\]

Reshaped input

Reshaped output

https://github.com/naokishibuya/deep-learning/blob/master/python/transposed_convolution.ipynb
Transposed convolution can be **equivalent** to a simple convolution with zero rows/columns added (added zeros simulate fractional strides)

Original input

Zero-padded input

Kernel

Output
**BatchNorm dynamically** normalizes each feature to have zero mean and unit variance

- Normalize input batch to each layer after **every update**
  1. Input is minibatch of data $X^t$ at iteration $t$
  2. Compute mean and standard deviation for every feature
     \[ \mu_j^t, \sigma_j^t \forall j \in \{1, \ldots, d\} \]
  3. Normalize each feature (note **different for every batch**)
     \[ \tilde{x}_j^t = \frac{(x_j^t - \mu_j^t)}{\sigma_j^t} \]
  4. Output $\tilde{X}^t$

BatchNorm dynamically normalizes each feature to have zero mean and unit variance

- Only normalize batches during training (model.train())
- Turn off after training (model.eval())
  - Use running average of mean and variance
    \[
    \mu_{run}^t = \lambda \mu_{run}^{t-1} + (1 - \lambda) \mu_{batch}^t
    \]
    \[
    \sigma^2_{run}^t = \lambda \sigma^2_{run}^{t-1} + (1 - \lambda) \sigma^2_{batch}^t
    \]
- Surprisingly effective at stabilizing training, reducing training time, and producing better models
- Not fully understood why it works

Resources for GANs

▸ DCGAN Tutorial
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

▸ GAN training tips/hacks
  ▸ https://github.com/soumith/ganhacks

▸ GAN common problems
  ▸ https://developers.google.com/machine-learning/gan/problems