Presentation on “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” by Alec Radford, Luke Metz, and Soumith Chintala

John Hancock jhancoc4@fau.edu

November 25, 2018

Overview

Introduction

- In the field of deep learning, we have crossed into an era where we are experimenting with designs that involve more than one neural network.
- Generative adversarial networks are a design pattern to employ two neural networks.
- "Unsupervised representational learning with deep convolutional generative adversarial networks", by Alec Radford, Luke Metz, and Soumith Chintala [1], is a milestone in the development of Generative Adversarial Networks.
- The authors report a reliable architecture that incorporates convolutional neural networks into the generative adversarial network design.
- The authors tout several applications of their design to prove its utility.

- For the remainder of this presentation, we will refer to the paper entitled , "Unsupervised representational learning with deep convolutional generative adversarial networks," as, "the DCGAN's paper," or by its reference number [1], and Alec Radford, Luke Metz, and Soumith Chintala as, "the authors."

Introduction

- A note about the DCGAN's paper: we find the authors of [1] do not include derivations, and very little in the way of performance metrics. Instead, they make the software they report on in the paper publicly available [2]. One must study and execute this publicly available code, or a derivative version of it in order to fully appreciate their work. The authors include graphics they claim their code generates, but without sharing their code, the work would be less meaningful.
- We ran the code from the derivative Tensorflow implementation in [4] to confirm the DCGAN generates images as advertised.
- The authors put their DCGAN's in the class of, "parametric models," of generative image models.
- For contrast the authors write that non-parametric models, "...do matching from a database of existing images".

A generative adversarial network (GAN) is a neural network with two components. Goodfellow *et. al* invent GAN's in [5]. To a first approximation, GAN's work as follows:

- The first component is a *generator* that learns to transform vectors of random numbers into output values that resemble instances from some dataset.
- The second component is a *discriminator* that classifies things into two categories:
  - the class of instances of the dataset, and
  - the class of generator outputs.
- "At convergence, the generators samples are indistinguishable from real data, and the discriminator outputs $\frac{1}{2}$ everywhere. The discriminator may then be discarded" [11]. from Deep learning , Goodfellow *et al.*

Background I

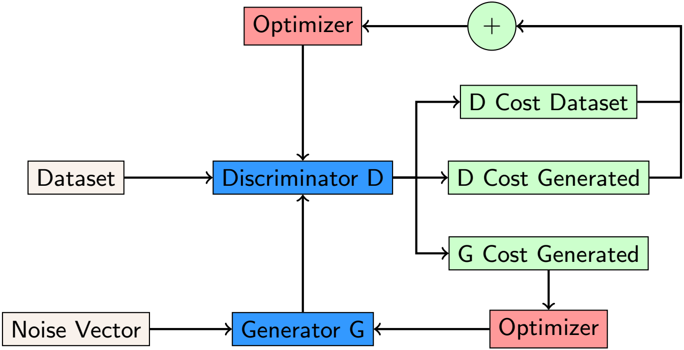
- ... or not. The authors of the DCGAN paper find a use for the discriminator.
- In the context of this paper, the outputs are images. However, researchers use GAN's where the generators create other artifacts. We find an extensive list on Github [12] of over 500 research projects. Some examples from this list are:
  - imputing missing values in datasets,
  - generating music,
  - fraud detection, and
  - playing chess.

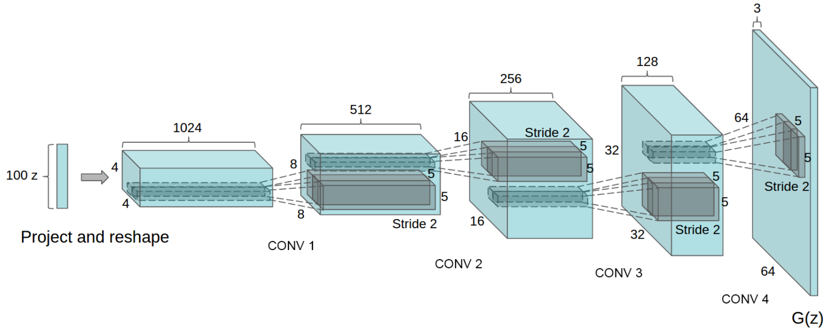
Contributions I

The authors of the paper make several contributions they...

- invent an architecture for DCGAN's,
- use the convolutional layer filters of trained DCGAN's discriminators as feature extractors for doing classifications,
- demonstrate that after training the DCGAN, its filters learn how to represent images, and
- present a method of doing vector arithmetic using DCGAN inputs to do inferences *à la* Word2Vec [6].

Architecture Diagram I




100 z
Project and reshape
1024
4
4
512
8
8
5
5
Stride 2
CONV 1
256
16
16
Stride 2
5
5
CONV 2
128
32
32
5
5
Stride 2
CONV 3
3
64
64
5
5
Stride 2
CONV 4
G(z)

- Here is an example of code the authors write to link the generator and discriminator.

```
gX = gen(Z, *gen_params)

p_real = discrim(X, *discrim_params)
p_gen = discrim(gX, *discrim_params)

d_cost_real = bce(p_real, T.ones(p_real.shape)).mean()
d_cost_gen = bce(p_gen, T.zeros(p_gen.shape)).mean()
g_cost_d = bce(p_gen, T.ones(p_gen.shape)).mean()

d_cost = d_cost_real + d_cost_gen
g_cost = g_cost_d
```

- Keep in mind: the discriminator's output layer uses a softmax activation. We interpret the output layer values as probabilities.
- We associate a high cost with the discriminator giving an output with a high probability for any outptut for generated inputs because that means the generator fooled the discriminator. This is what d_cost_gen does.
- We associate a low cost for the discriminator's doing the same thing for "real," inputs. This is what d_cost_real does.

- We associate a low cost with the generator's producing an output that the discriminator gives a high probability output for.

- The discriminator is a convolutional neural network.
- Here is the code from [2] that defines it:

```
def discrim(X, w, w2, g2, b2, w3, g3, b3, w4, g4, b4, wy):
  h = lrelu(dnn_conv(X, w, subsample=(2, 2),
    border_mode=(2, 2)))
  h2 = lrelu(batchnorm(dnn_conv(h, w2, subsample=(2, 2),
    border_mode=(2, 2)), g=g2, b=b2))
  h3 = lrelu(batchnorm(dnn_conv(h2, w3, subsample=(2, 2),
    border_mode=(2, 2)), g=g3, b=b3))
  h4 = lrelu(batchnorm(dnn_conv(h3, w4, subsample=(2, 2),
    border_mode=(2, 2)), g=g4, b=b4))
  h4 = T.flatten(h4, 2)
  y = sigmoid(T.dot(h4, wy))
  return y
```

- The important thing to notice in the code above is that the authors implement the discriminator as a six layer neural network with four convolutional layers, a flattening layer, and an output layer that uses the sigmoid activation but leaky rectified linear units (ReLU's) elsewhere.

- subsequent work such as the texts for this course - Deep Learning with R [10], and Deep Learning [11] suggests one should employ dropout, but we do not find that the authors of this paper use it when we inspect the code in [2].

- The generator uses fractionally-strided layers.
- The authors of the paper prefer the term, "fractionally-strided." However in the course of research one might encounter layers implemented with the same functionality referred to as deconvolutional layers.
- Here is the code for the generator implementation:

```
def gen(Z, w, g, b, w2, g2, b2, w3, g3, b3, w4, g4, b4, wx):
    h = relu(batchnorm(T.dot(Z, w), g=g, b=b))
    h = h.reshape((h.shape[0], ngf*8, 4, 4))
    h2 = relu(batchnorm(deconv(h, w2, subsample=(2, 2),
      border_mode=(2, 2)), g=g2, b=b2))
    h3 = relu(batchnorm(deconv(h2, w3, subsample=(2, 2),
      border_mode=(2, 2)), g=g3, b=b3))
    h4 = relu(batchnorm(deconv(h3, w4, subsample=(2, 2),
      border_mode=(2, 2)), g=g4, b=b4))
    x = tanh(deconv(h4, wx, subsample=(2, 2), border_mode=(2, 2)))
    return x
```

- The important thing to notice about the generator is that the input layer transforms a vector of random values into a $1024 \times 4 \times 4$ tensor. The authors then add 5 fractionally-strided convolutional layers, and use a tanh activation for the output layer. The authors write that using tanh gives better output as images, and speeds up training [1].

Datasets I

- The authors use three datasets for training:
  - Large Scale Scene Understanding (LSUN),
  - Imagenet 1-K, and
  - Faces.
- The authors two datasets for evaluating unsupervised learning: Canadian Institute for Advanced Research (CIFAR) 10 StreetView House Numbers (SVHN)
- Note: the authors mention that they heuristically removed duplicate images from some dataset(s) to prevent the DCGAN from memorizing images. They do not explicitly say they applied the technique to LSUN but they write about it nearby where they mention LSUN.

LSUN I

- The authors use images of bedrooms from the LSUN dataset [3] as input to their DCGAN.
- Then they find a way to identify and remove the generators feature maps [1] associated with windows. Note: see explanation of feature maps in Dr. Zhu's lectures on CNN's [16].
- After removing these feature maps, the generator no longer produces images of bedrooms with windows.
- The authors claim that this experiment proves the generator is learning representations of objects in an unsupervised manner.

Representation learning visualization I

- The authors did some processing on the first 6 convolutional features of the last convolutional layer of their DCGAN's discriminator to create the image below.
- This is proof that the discriminator is learning features.
  - Note on the left-hand side taken before training that the pictures look like entire bedrooms, but after training the pictures have key portions of a bedroom scene, like windows and beds.
  - This demonstrates how the DCGAN is learning to represent objects in images.

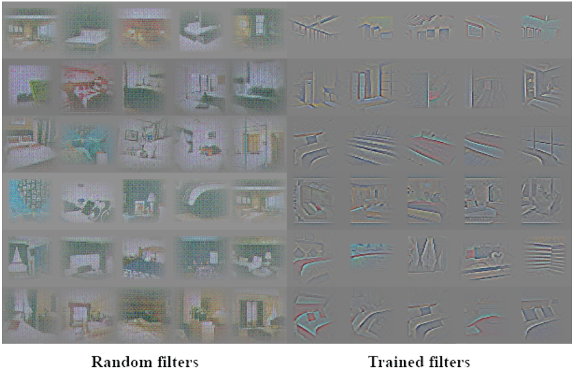


Figure: This image is coped from the paper [1].

Details on fractionally-strided convolutions I

- The code for this paper, as well as many others in the references and that one may find in the course of research, is on Github in the dcgan_code project [2].
- We found that this is not compatible with the current version of Theano that is available to us.
- However the dcgan_code Github project has a link to the DCGAN-tensorflow [4] project that we find more accessible.

CIFAR I

- The authors use images from the **Imagenet 1-k** dataset as input to a DCGAN.
- After training, they take all the convolutional layers the discriminator learns, and use them as feature extractors.
- However they accomplished this, the authors then use the feature extractor they created from the DCGAN's discriminator for a support vector machine based classifier.
- They then used images from **CFAR-10** dataset as inputs to this classifier that they report as 82.8% accurate. They mention similar K-Means based approaches that get lower accuracy so the result is competitive and noteworthy.
- This result is remarkable because they built a classifier that is able to correctly categorize images from one dataset, based on unsupervised learning methods involving a different dataset. It proves GAN's have generalization power.

SVHN I

- We interpret the wording the authors use in section 5.2 to mean that the authors use images from the SVHN dataset as input to a DCGAN.
- It is not clear what the authors mean by, “non-extra set” they use for training their model.
- Is is our understanding that they train a DCGAN on the SVHN dataset, and then build a feature extractor similar to what they do with the DCGAN they build for Imagenet 1-k.
- Using this feature extractor, the authors then incorporate this into an L2-SVM classifier that we suppose is classifying images from SVHN.
- The notable result the authors report is that the classifier they build in this manner gets a lower error rate than a similar classifier built with a standard convolutional neural network.

Faces I

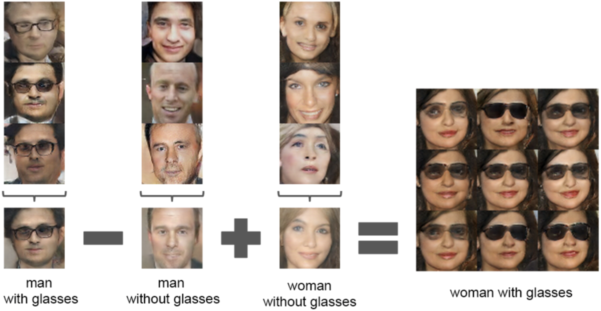
- The authors write that they create a faces dataset of images of faces from randomly selected web sites.
- After training the DCGAN on this dataset the authors do arithmetic on what they call, the "Z-vectors of sets of exemplar samples for visual concepts."
  - We take this to mean they were able to find groups of vectors of random numbers they used for inputs to the generator that produce images that look like something in particular, for example: a smiling man.
  - An important clue for our understanding of the term Z-vector is that the first input parameter of the generator function in the code that accompanies this paper [2] is named, "Z," and other vectors involving random numbers in the code also start with the letter, 'z.'

Faces I

- The authors then do vector addition and subtraction with vectors they obtain from the average values of vectors in different exemplar sets.
- The authors then use the vectors are the results of these arithmetic operations as inputs to the generator.
- Please see the amazing result on the next slide. We feel this is the strongest result of the paper.

Faces I

This image is from the DCGAN paper [1]:



Fractionally-strided convolutions I

- The generator uses convolutional layers that we find called deconvolutional layers in the source code that accompanies this paper [2], and elsewhere, but that in the paper the authors write that we should prefer the term "*fractionally-strided*."
- The computations that comprise fractionally-strided convolutions are not clear to us from the paper or the source code that accompanies it.
- We find the source code unclear because the authors implement fractionally-strided convolutions using library functions, the source code of which we run out of time to peruse. The paper lacks detail on how to compute a fractionally-strided convolutions.

Fractionally-strided convolutions I

- On the other hand, the github project [2] that accompanies the paper [1] links to a Tensorflow implementation of the same code: [4] where the author of this code implements fractionally-strided convolutions using Tensorflow's conv2d_transpose.
- We did some internet searching and found [14].
  - This reference plus using conv2d_transpose in a small example helps us understand precisely how the fractionally-strided convolution operation works.
  - We feel confident to rely on conv2d_transpose because the authors of the paper [1] we review here provide a link to the code in [4] in their own code.

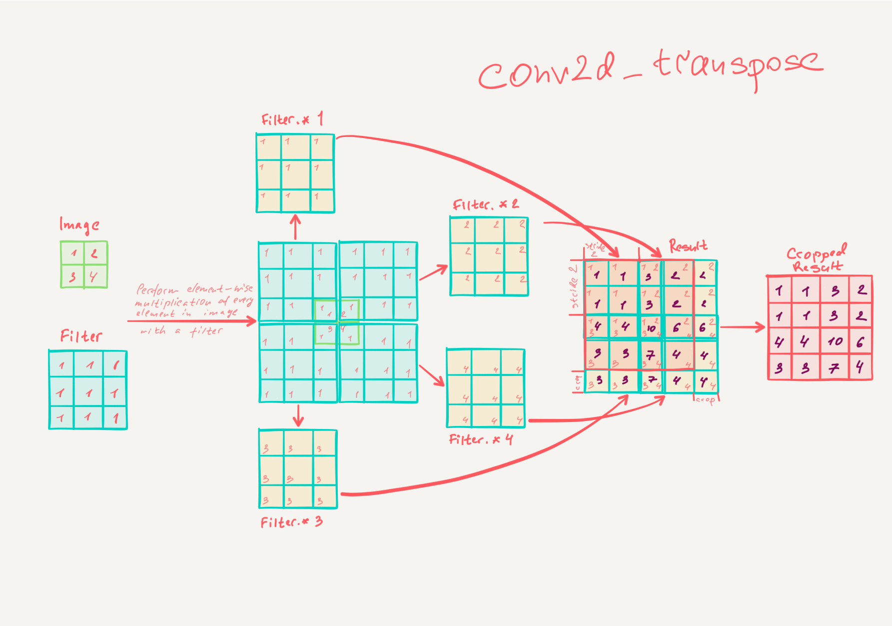
Fractionally-strided convolutions II

- We feel the authors of the DCGAN's paper's endorsement of the Tensorflow code means conv2d_transpose is a valid method for doing what the authors of [1] refer to as fractionally-strided convolutions, and that a good understanding of conv2d_transpose is a good understanding of fractionally-strided convolutions.

We found some example code, and a great diagram from a StackExchange.com discussion [15]. That explains in detail how conv2d_transpose works. This is the diagram we found:

Fractionally-strided convolutions II

Diagram from StackExchange [15]



- Since anyone might write anything in online discussion forums, we decided to confirm that Tensorflow's conv2d_transpose operation works as the digram implies. conv2d_transpose has three important parameters: input tensor, filter, and stride.
- One should be careful not to confuse the term filter we have for conv2d_transpose and the filters that the authors of the DCGAn's paper show on page 9.
- In the context of the DCGAN paper it is better to think of the filter parameter of conv2d_transpose as a kernel for the conv2d_transpose operation, and the filter is the result of applying conv2d_transpose to the input tensor.

Fractionally-strided convolutions I

- The next slide shows a 4x4 input tensor, and the result of applying conv2d_transpose to that tensor, with stride of 1,4,4,1.
- Conv2d_transpose operates on 4 dimensional tensors, so we must embed the 4x4 matrix in a 4-dimensional tensor, and stride through it accordingly. Note on the next slide how most entries in the output tensor are copies of entries in the input tensor, except where the 5x5 kernels must overlap in order to achieve the 16x16 output.

The input tensor:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

The output we can see the entries in the input matrix copied into 5x5 intermediate tensors and then added according to the stride of 4, and values are added when we have overlap. We show a screen shot to prove the code runs.

Fractionally-strided convolutions II

```
jhancock2010@minty ~/git/DCGAN-tensorflow
File  Edit  View  Search  Terminal  Help
result
[[ 1.   1.   1.   1.   3.   2.   2.   2.   5.   3.   3.   3.   7.   4.   4.   4.]
 [ 1.   1.   1.   1.   3.   2.   2.   2.   5.   3.   3.   3.   7.   4.   4.   4.]
 [ 1.   1.   1.   1.   3.   2.   2.   2.   5.   3.   3.   3.   7.   4.   4.   4.]
 [ 1.   1.   1.   1.   3.   2.   2.   2.   5.   3.   3.   3.   7.   4.   4.   4.]
 [ 6.   6.   6.   6.  14.   8.   8.   8.  18.  10.  10.  10.  22.  12.  12.  12.]
 [ 5.   5.   5.   5.  11.   6.   6.   6.  13.   7.   7.   7.  15.   8.   8.   8.]
 [ 5.   5.   5.   5.  11.   6.   6.   6.  13.   7.   7.   7.  15.   8.   8.   8.]
 [ 5.   5.   5.   5.  11.   6.   6.   6.  13.   7.   7.   7.  15.   8.   8.   8.]
 [14.  14.  14.  14.  30.  16.  16.  16.  34.  18.  18.  18.  38.  20.  20.  20.]
 [ 9.   9.   9.   9.  19.  10.  10.  10.  21.  11.  11.  11.  23.  12.  12.  12.]
 [ 9.   9.   9.   9.  19.  10.  10.  10.  21.  11.  11.  11.  23.  12.  12.  12.]
 [ 9.   9.   9.   9.  19.  10.  10.  10.  21.  11.  11.  11.  23.  12.  12.  12.]
 [22.  22.  22.  22.  46.  24.  24.  24.  50.  26.  26.  26.  54.  28.  28.  28.]
 [13.  13.  13.  13.  27.  14.  14.  14.  29.  15.  15.  15.  31.  16.  16.  16.]
 [13.  13.  13.  13.  27.  14.  14.  14.  29.  15.  15.  15.  31.  16.  16.  16.]
 [13.  13.  13.  13.  27.  14.  14.  14.  29.  15.  15.  15.  31.  16.  16.  16.]]
(2018-11-04-py3.6) jhancock2010@minty ~/git/DCGAN-tensorflow $
(2018-11-04-py3.6) jhancock2010@minty ~/git/DCGAN-tensorflow $
(2018-11-04-py3.6) jhancock2010@minty ~/git/DCGAN-tensorflow $
(2018-11-04-py3.6) jhancock2010@minty ~/git/DCGAN-tensorflow $
(2018-11-04-py3.6) jhancock2010@minty ~/git/DCGAN-tensorflow $
(2018-11-04-py3.6) jhancock2010@minty ~/git/DCGAN-tensorflow $
(2018-11-04-py3.6) jhancock2010@minty ~/git/DCGAN-tensorflow $
```

Fractionally-strided convolutions I

References I

- S. Chintala, L. Metz, and A. Radford, Unsupervised representation learning with deep convolutional generative adversarial networks. 2016. [Online]. Available: arXiv:1511.06434v2 [cs.LG]
- S. Chintala, L. Metz, and A. Radford, dcgan_code. (2016, May). Available: `https://github.com/Newmu/dcgan_code`. [Accessed Nov. 19, 2018].
- Large Scale Scene Understanding Challenge (2017, July). Available: `http://lsun.cs.princeton.edu/2017/`. [Accessed Nov. 19, 2018].
- T Kim. (2018, August). Available: `https://github.com/carpedm20/DCGAN-tensorflow`. [Accessed Nov. 19, 2018].
- Y. Bengio, A. Courville,I. Goodfellow, M. Mirza, S. Ozair, J. Pouget-Abadie, D. Warde-Farley, B. Xu, Generative adversarial nets. 2014. [Online]. Available: arXiv:1406.2661v1 [stat.ML]

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," Advances in Neural Information Processing Systems 26 (NIPS 2013), 2013. [Online] Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Creodocs Limited, "Beamer Presentation," *latextemplates.com*, 2018. [Online], Available: http://www.latextemplates.com/templates/presentations/1/presentation_1.zip. [Accessed Nov. 10, 2018].

IEEE, Piscataway, NJ, USA. *IEEE Editorial Style Manual*. 2016. [Online]. Available: http://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE_Style_Manual.pdf, [Accessed Nov. 11, 2018].

Y. LeCun, "Unsupervised Representation Learning," 2017. Accessed on: Nov 11, 2018. [online]. Available: https://www.youtube.com/watch?v=ceD736_Fknc

References III

- F. Chollet, and J.J. Allaire. *Deep Learning with R*. Manning Publications 2018. [E-Book] Available: Safari E-Book.
- Y. Bengio, A. Courville, I. Goodfellow. (2016). "Chapter 20 Deep Generative Models," 2016. [Online] Available: `https://www.deeplearningbook.org/contents/generative_models.html`. [Accessed: Nov. 20, 2018].
- A. Hindupur, "A list of all named GANs!" github.com, Sep. 30, 2018. [Online]. Available: `https://github.com/hindupuravinash/the-gan-zoo`. [Accessed Nov. 21, 2018].
- L. Bottou, P. Haffner, Y. Bengio, Y. LeCun, "Gradient-Based Learning Applied to Document Recognition," Proc of the IEEE, Nov., 1998. [Online]. Available: `http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf`. [Accessed Nov. 21, 2018].

References IV

V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," Jan 2018. [Online]. Available: Github, https://github.com/vdumoulin/conv_arithmetic. [Accessed November 21, 2018].

Stack Exchange user Andriys, "What are deconvolutional layers," datascience.stackexchange.com, answer written July 14 2017, edited Nov. 19, 2017. [Online]. Available: https://datascience.stackexchange.com/questions/6107/what-are-deconvolutional-layers. [Accessed Nov. 22, 2018].

Xingquan Zhu. 2018. Convolutional Neural Networks (CNN) retrieved October 22, 2018 from https://canvas.fau.edu/files/14310707/download?download_frd=1

The End