DCGAN for classroom images

Liis Kolberg, Mari-Liis Allikivi

Abstract—Generative Adversarial Networks (GAN) have been used for generating images that look real but are in fact generated by an artificial neural network. However, they have shown to be quite unstable. In 2015, authors of [5] proposed a set of guidelines for building generative networks that should make the networks more stable and better to train. They call these instructions DCGAN. In this work we built and trained a DCGAN model on classroom images following the guidelines given in the original article. We demonstrate that the model is indeed able to generate images that resemble classrooms despite the difficult and noisy training images.

Keywords—DCGAN, LSUN classrooms.

INTRODUCTION

The best way to understand a specific approach or method is to try to recreate it. It is especially useful for learning complicated neural network architectures. At the same time the learning process contributes to the scientific and industrial communities by showing the reproducibility or irreproducibility of already published results. Having learned the basic knowledge of neural networks in this semester, we now ought to widen the practical skills out of the scope of this course by replicating a scientific article on unsupervised deep learning. Alec Radford, Luke Metz and Soumith Chintala published a conference paper on “Unsupervised representation learning with Deep Convolutional Generative Adversarial Networks” in 2015 [5]. The goal of the current project is to get an experience in adversarial training by training a Deep Convolutional Generative Adversarial Network (DCGAN) on scenical images of classrooms. We will build and train a model based on the descriptions and guidelines given in the article [5] and visualize the results.

BACKGROUND

To recognize shapes, first learn to generate images

Geoffrey E. Hinton

Generative Adversarial Network

Finding a sufficient amount of labeled data is a major obstacle in machine learning classification tasks. Even though an increasing amount of data is becoming available on the Internet, majority of it is unlabeled. One way to benefit from the abundance of available data is to use unsupervised learning to learn reusable feature representations and then use these intermediate representations on a variety of supervised learning tasks. Generative models are one of the most popular approaches that are applied for this, as the model needs to be able to analyze and understand the essence of the training data before it can generate similar results itself [4]. Generative Adversarial Networks (GANs) [2] are proposed as dominating family of models to build good image
representations and generate “realistic-looking” images. The idea of GANs is to simultaneously train two models: a generative model $G$ and discriminative model $D$. The generative model creates photorealistic images that are similar to the original data distribution and the task of discriminative model is to determine whether a given image looks realistic (image came from the training data) or looks like it has been artificially created by the generative model. An example of a GAN architecture that is trained on MNIST images is in Fig. 1.

![GAN Architecture](image)

Fig. 1: Example of a GAN training on MNIST images. Generator tries to generate images similar to MNIST images so that the discriminator cannot distinguish between real and generated images.

These networks are trained competitively, as a two-player minimax game, until neither of them can make further progress against the other or the generator becomes so good that the discriminator can’t distinguish between real and artificial images.

The objective function of GAN is of form:

$$\min_G \max_D V(G, D) = \min_G \max_D \left( \mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\theta_d}(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \right),$$

where $D_{\theta_d}$ is a multilayer perceptron that outputs a single scalar (likelihood from $[0, 1]$), $p_z$ a prior on input noise variables $z$, $G$ is a differentiable function represented by a multilayer perceptron with parameters $\theta_g$. $D_{\theta_d}(x)$ represents the probability that $x$ came from the data rather than from the generator’s distribution. In other words, the goal is to train the discriminator to maximize the probability of assigning the correct label to both training examples and generated samples. At the same time the generator is trained to to minimize $\log(1 - D_{\theta_d}(G_{\theta_g}(z)))$, the discriminator output for generated fake data $G_{\theta_g}(z)$.

The general steps of training GANs are shown in Fig. 2.

![Algorithm of GAN](image)

Fig. 2: The algorithm of GAN. Here $k$ is the number of steps to apply to the discriminator (hyperparameter).

However, due to the battle of two networks, GANs are known to have several pitfalls. They are unstable to train which means that they often result in generators that produce outputs that don’t make sense. They are hard to tune and to get working. To tackle these issues, Radford and colleagues propose a class of Convolutional Neural Networks (CNN) called Deep Convolutional Generative Adversarial Networks (DCGAN), that have a set of architectural
constraints to stabilize GANs [5]. It is basically a set of guidelines for building architectures for images. The approach is quite popular, the article has already around 840 citations according to Google Scholar.

*Deep Convolutional Generative Adversarial Network*

The main idea of DCGANs is to scale up GAN using CNN architectures. Radford *et al.* managed to achieve stable results by adopting certain architectural constraints to Convolutional GANs.

The guidelines proposed in [5] are the following:

1) Replace pooling layers with strided convolutions in discriminator and fractional-strided convolutions (also known as transposed convolutions) in generator.
2) Use Batch Normalization in both the generator and the discriminator.
3) Remove fully connected hidden layers on top of convolutional features.
4) Use ReLU activation in generator for all layers except for the output, which uses tanh.
5) Use LeakyReLU activation in the discriminator for all layers.

An example model architecture is shown in Fig. 3. No fully connected or pooling layers are used. The input $z$ to the model is a 100 dimensional vector usually sampled from uniform or normal distribution.

**METHODS**

*Details of DCGAN architecture*

Our DCGAN architecture is based on a code from publicly available repository [6]. This was written to train on the CIFAR-10 dataset. We modified the code to achieve as much similarity as possible to the DCGAN architecture described in previous section. This involved making the code suitable for our input size and changing the number of filters in convolutional layers. The architecture of the trained model is shown in Fig. 3. The number of model parameters is shown in Table 1.

We made sure that we use fractional-strided convolutions (also known as deconvolution or transposed convolutions) in generator and strided convolutions in discriminator. Batch Normalization was applied and ReLU/LeakyReLU activations were used as suggested in the guidelines.

As in the DCGAN article [5], the only preprocessing of the images was scaling them to range of tanh activation function $[-1, 1]$. The model was trained with mini-batch stochastic gradient descent with a mini-batch size of 128. We used the Adam optimizer with learning rate of 0.0001 and the momentum term $\beta_1$ with value 0.2. The slope of the leak in the LeakyReLU was set to 0.2. All weights were initialized by default settings. The noise input
\(z\) was from standard normal distribution.

The implementation of this project is in Keras. We trained the model for 25 epochs on Rocket Cluster using a Tesla P100 for approximately 48 hours. In addition we trained a model with hyperparameters such as learning rate and momentum term set to the values indicated in [5]. However we decided to stick to the original trained version.

**Data**

Among other datasets, Radford et al. demonstrated DCGANs on the Large-scale Scene Understanding (LSUN) [7] bedroom images. We trained DCGAN on the LSUN classrooms dataset containing 168103 images. Before training, we cropped the images into \(64 \times 64\) pixels.

A random sample from the images is shown in Fig. 4. It can be seen that the images contain a lot of details to be learned, including many students in classroom. Compared to the bedroom images, it is harder to define the typical parts of classrooms based on these images and therefore we don’t expect as good results as in [5].

**Results**

*Generated images*

We used the trained generator to generate images of classrooms. The samples after two epochs of training are shown in Fig. 5. Two training passes through the classroom dataset didn’t result in as good images as were obtained by training on bedroom images with one pass in [5]. This could be explained by the fact that in case of bedroom images over 3 million training images were used.

Generated classrooms after 25 epochs of training (Fig. 6) demonstrate that our model does not produce high-quality samples. However, although the images are not very sharp, we can still notice the
similarity to classrooms. For example, in the last image of 4th row we can see signs of typical lighting used in classrooms as well as objects that resemble tables.

Fig. 6: Generated classroom images after 25 epochs of training.

The complexity and variety of training images made it complicated to get very good results. Different parameters should have been tried and also longer training could have helped to improve the outcome.

*Walking in the latent space*

According to Radford *et al.*, semantic changes to the image generations if walking in the latent space show that the model has learned interesting and relevant representations [5]. For example, if objects are being added or removed. On the other hand, sharp transitions can indicate the signs of memorization.

![Interpolation between a series of 9 random points in z.](image)

Fig. 7: Interpolation between a series of 9 random points in $z$.

We walked from one point to another in classroom latent space. For that, we interpolated between a series of 9 random points in the input noise variable $z$. The results are shown in Fig. 7. It can be seen that the space learned has relatively smooth transitions. In the 9th row, you see how a classroom with window is slowly transforming into a room with ceiling lights and without window. In the last row you see a whiteboard being transformed into a wall. However, we admit that the images are very fuzzy and need also some imagination to see the classrooms.

*Google image search from generations*

As we are not sufficiently objective to detect classrooms from the generated images, we tested the generated images on Google image search. After a long cherry-picking we managed to generate an image that is identified as a room in the Google
search. The result is shown in Fig. 8. The generated images were also related to a keyword "crowd" on several occasions. This is a reasonable result as a lot of training images depict many students in the classroom.

Fig. 8: Google image search for generated image returns keyword "room".

**Nearest training images**

We wanted to see if the generated images are different enough from the training images to make sure that the generator is not just memorizing certain samples from training images. For that we generated some new images and searched for the nearest matches using Euclidean and cosine distances. Figure 9 shows the nearest matches for 2 generated images. As we can see, the nearest real images are not similar at all to the generated images. At first we considered it a bug in our code, but it turned out not to be the case. Thus we conclude that the selected distances are not able to detect image similarity in the case of very fuzzy images. Another option is that there are no images that are similar enough to the generated ones.

Fig. 9: Two generated images (leftmost column) and five most similar real images from the training set by Euclidean distance (odd rows) and cosine distance (even rows).

**CONCLUSIONS AND FUTURE WORK**

We built a DCGAN model on LSUN classroom dataset. The model was able to generate images that resemble classrooms, however, not very clearly. Probably using a larger training set or training longer could improve the results.

Further investigations into the internals of the network to manipulate the generator representation could be the next steps in this project. For example, Radford and colleagues [5] conducted an experiment to remove window representations from their bedroom generator. As a result, the generator indeed forgot to draw windows in most of the cases. In addition, extending this project to images of faces to explore the vector arithmetic of $z$ representation of the generator would be very interesting.

**DECLARATIONS**

**Acknowledgements**

The authors would like to thank Tambet Matiisen and Ardi Tampuu for the great course and helpful
guidance.

Author’s contributions

LK and MLA gathered the data, conducted the experiment and wrote the report.

REFERENCES


