Assignment 3: Generative Models

TA: Du Ang (daa@stu.ouc.edu.cn)

Due date: June 6, 2018

1 Introduction

In this assignment, there are several tasks that you need to finish. At first, you will implement an Autoencoder to reconstruct Fashion-MNIST images. After that, you will implement a basic Generative Adversarial Network (GAN) and a Deep Convolutional Generative Adversarial Network (DCGAN) to generate Fashion-MNIST-like images. At last, after implementing the above models, you are encouraged to challenge the bonus task — implementing an Emoji CycleGAN, which can convert between Windows-style and Apple-style emojis.

2 Preparation

2.1 Requirement

To implement the models in this assignment, at first, you need to install a Deep Learning framework on your computer system. Deep Learning framework TensorFlow and PyTorch are both welcomed to implement this assignment. Ubuntu (14.04 or 16.04) is recommended as the system, since installing Deep Learning framework TensorFlow and PyTorch are much easier on it.The installation guide and tutorials of TensorFlow and PyTorch can be found on their official websites.

Further more, you may meet some problems when doing this assignment. Then you'd better search online and find the answers in some reputable open-source communities like GitHub and Stack Overflow.

2.2 Dataset

2.2.1 Modified Fashion-MNIST

Fashion-MNIST is an image dataset consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes, which can be seen in Figure 1. To save the computational expense, we resized all the images into a size of 24×24, getting a modified version of Fashion-MNIST.

The modified Fashion-MNIST dataset should be used when you implement the Autoencoder in Section 3.1, the basic GAN in Section 4.1, and the DCGAN in Section 4.2.

Figure 1: The examples in Fashion-MNIST dataset (each class takes three rows).

2.2.2 Emoji dataset

I am sure that you are no stranger to emoji, since we all use them on social network everyday. Emoji are ideograms and smileys used in electronic messages and web pages. Emoji exist in various genres, including facial expressions, common objects, places and types of weather, and animals. They are much like emoticons, but emoji are actual pictures instead of typographics. Emojis from different platforms may have different style. In Figure 2, there are emoji examples from Windows platform and Apple platform. In Section 5, you will implement an emoji CycleGAN to convert between these two styles.

I have shared the above two datasets on Baidu Netdisk. Dowload link address: https://pan.baidu.com/s/1_qqzM9WS-YB6aQzxNk4SMA, extracting password: `wtey`.

3 Autoencoder

An Autoencoder is an artificial neural network used for unsupervised learning of efficient codings. The aim of an Autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. the Actually, Autoencoder you need to implement here is not a generative model. But recently, the Autoencoder concept and its variants have become more widely used for learning generative models of data.

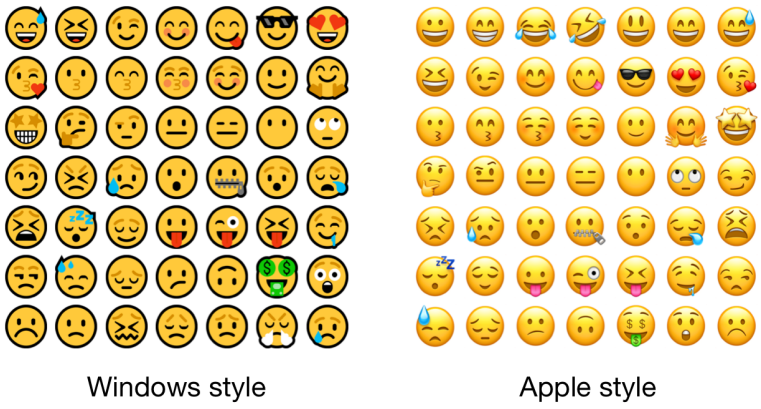

Figure 2: Emoji examples of Windows style and Apple style.

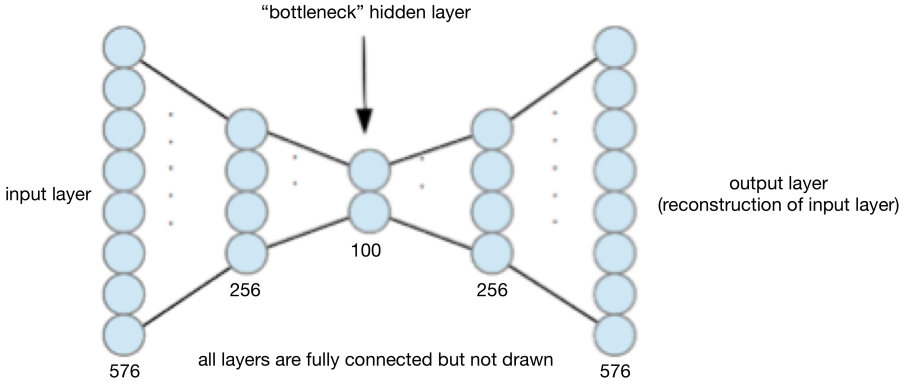

Figure 3: The architecture of the Autoencoder you should implement

3.1 Warm-up: Implement an Autoencoder

In this section you will implement an Autoencoder to reconstruct the images in Fashion-MNIST dataset. The Autoencoder you should implement consists of two parts, the encoder part and the decoder part.

Like Figure 3 shows, the Autoencoder consists of one input layer, three hidden layers and one output layer. In detail, the input layer contains 576 neural units, three hidden layers contain 256, 100, 256 neural units respectively, and the output layer contains 576 neural units. When loading and outputting the images, you should reshape them to fit the right dimension.

3.2 Autoencoder Experiments

After implementing the architecture of the above Autoencoder, train it on Fashion-MNIST dataset. Use L_2 loss as the training loss, which can be described by the following equation:

$$L(x, x') = |x - x'|^2$$

where x represents an original image in Fashion-MNIST dataset, x' represents a generated fake image.

After finish training the model, use it to construct images from Fashion-MNIST dataset and make a comparison between the constructed ones and the original ones.

4 Generative Adversarial Network (GAN)

Generative Adversarial Network (GAN) are a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks contesting with each other in a zero-sum game framework. They were introduced by Ian Goodfellow *et al.* in 2014. This technique can generate images that look realistic.

To implement a GAN, you need to specify three things: 1) the generator, 2) the discriminator, and 3) the training procedure. The generator and the discriminator can be implemented with fully-connected neural networks or convolutional neural networks. In the following part, you should implement both of them successively to generate images based on Fashion-MNIST dataset.

4.1 Implement a basic GAN

In this part, you will implement a GAN with fully-connected layers in the generator and discriminator.

4.1.1 Discriminator

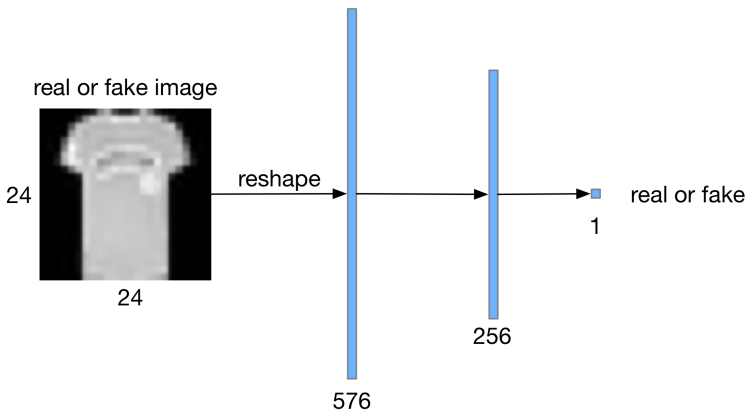

Figure 4: The discriminator architecture of the basic GAN you need to implement.

The discriminator of this basic GAN is a fully-connected neural network, whose architecture is shown in Figure 4. The discriminator contains an input layer with 576 neural units, a hidden layer with 256 neural units, and an output layer with 1 neural unit.

4.1.2 Generator

The generator of this basic GAN is also a fully-connected neural network, whose architecture is shown in Figure 5. The generator contains an input layer with 100 neural units, which should sample from a unit Gaussian distribution. The generator also contains a hidden layer with 256 neural units and an output layer with 576 neural units.

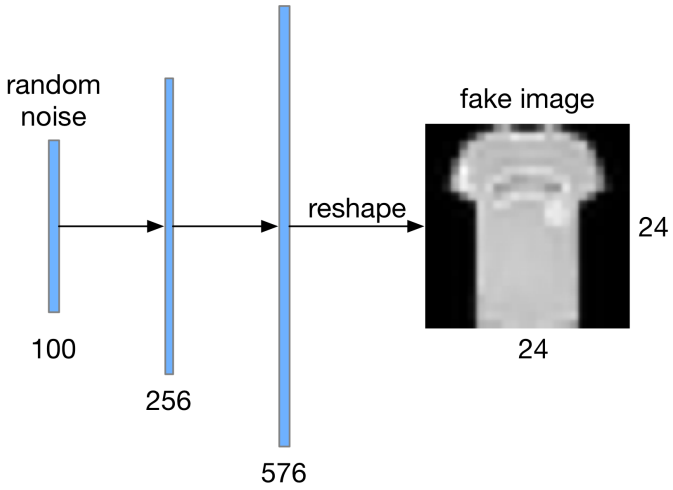

Figure 5: The generator architecture of the basic GAN you need to implement.

4.1.3 Training loop

Next, you will implement the training loop for the basic GAN. The pseudo-code for the training procedure is shown in Algorithm 1. The actual implementation is simpler than it may seem from the pseudo-code: this will give you practice in translating math to code. Note that when we compute the loss of the discriminator and the generator, we use the least-square loss.

There are 5 numbered bullets in the code to fill in for the discriminator and 3 bullets for the generator. Each of these can be done in a single line of code, although you will not lose marks for using multiple lines.

4.2 Implement a Deep Convolutional GAN (DCGAN)

In this part, you will implement a Deep Convolutional GAN (DCGAN). A DCGAN is simply a GAN that uses a convolutional neural network as the discriminator, and a network composed of transposed convolutions as the generator. To implement the DCGAN, we also need to specify three things: 1) the generator, 2) the discriminator, and 3) the training procedure. Similarly, we will develop each of these three components in the following subsections.

4.2.1 Discriminator

The discriminator in this DCGAN is a convolutional neural network that has the architecture shown in Figure 6.

Algorithm 1 GAN Training Loop Pseudocode

procedure TRAINGAN
  Draw m training examples $\{x^{(1)}, ..., x^{(m)}\}$ from the data distribution p_{data}
  Draw m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from the noise distribution p_z
  Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, ..., m\}$
  Compute the (least-squares) discriminator loss:

$$J^{(D)} = \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(x^{(i)}) - 1\right)^2\right] + \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(G(z))\right)^2\right]$$

  Update the parameters of the discriminator
  Draw m new noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from the noise distribution p_z
  Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, ..., m\}$
  Compute the (least-square) generator loss:

$$J^{(G)} = \frac{1}{m}\sum_{i=1}^{m}\left[\left(D(G(z^{(i)}) - 1)^2\right)\right]$$

  Update the parameters of the generator

There are 3 convolutional layers in the discriminator, all of them use a kernel of 3 and a stride of 2. Batch Normalization and ReLU activation are used after the convolutional layer except the last one, for which Sigmoid activation is used. The dimensions of all feature maps have been marked in Figure 6.

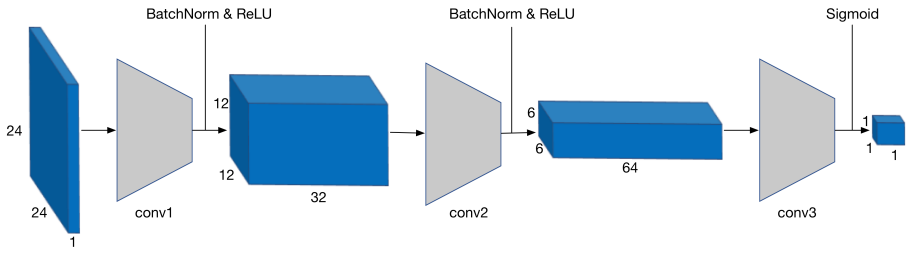

Figure 6: The discriminator architecture of the DCGAN you need to implement.

4.2.2 Generator

Now, we will implement the generator of the DCGAN, which consists of a sequence of transpose convolutional layers that progressively upsample the input noise sample to generate a fake image. The generator we'll use in this DCGAN has the architecture shown in Figure 7.

There are 3 transpose convolutional layers in the generator, which are analogous to the convolutional layers in the discriminator. Batch Normalization and ReLU activation are used after the convolutional layer except the last one, for

which tanh activation is used. Similarly, the dimensions of all feature maps have been marked in Figure 7.

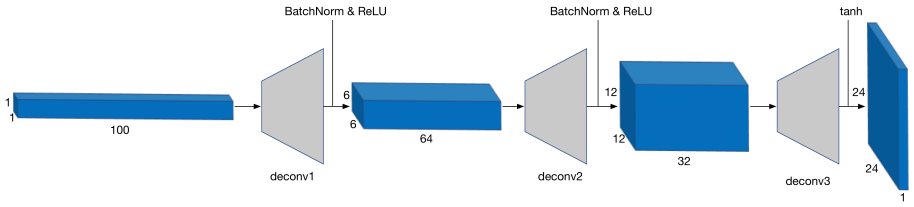

Figure 7: The generator architecture of the DCGAN you need to implement.

4.2.3 Training loop

A DCGAN is simply a GAN with a specific type of generator and discriminator; thus, we train it in exactly the same way as a standard GAN by following Algorithm 1.

4.3 GAN Experiments

For both the basic GAN and the DCGAN you implement, you should train them on the Fashion-MNIST dataset mentioned in Section 2.2.1.

After training, you should sample noises and load it into the trained generators to generate fake images. Make a comparison between the generated images and the original images. What's more, you can vary the sampled noises and observe what it will cause on the generated images.

5 Bonus Task: Implement an Emoji CycleGAN

I admire you for having the courage to come here. Here are the instructions to implement the CycleGAN architecture. Most of the instructions are adopted from the CycleGAN assignment in CSC 321 Winter 2018 course in Toronto University, and you are allowed to adopt their code to complete this assignment.

5.1 Motivation: Image-to-Image Translation

Say you have a picture of a sunny landscape, and you wonder what it would look like in the rain. Or perhaps you wonder what a painter like Monet or van Gogh would see in it? These questions can be addressed through image-to-image translation wherein an input image is automatically converted into a new image with some desired appearance. Recently, Generative Adversarial Networks have been successfully applied to image translation, and have sparked a resurgence of interest in the topic. The basic idea behind the GAN-based approaches is to use a conditional GAN to learn a mapping from input to output images. The loss functions of these approaches generally include extra terms (in addition to the standard GAN loss), to express constraints on the types of images that are generated.

A recently-introduced method for image-to-image translation called CycleGAN is particularly interesting because it allows us to use un-paired training data. This means that in order to train it to translate images from domain X to domain Y, we do not have to have exact correspondences between individual images in those domains. For example, in the paper that introduced CycleGANs, the authors are able to translate between images of horses and zebras, even though there are no images of a zebra in exactly the same position as a horse, and with exactly the same background, etc.

Thus, CycleGANs enable learning a mapping from one domain X (say, images of horses) to another domain Y (images of zebras) without having to find perfectly matched training pairs.

To summarize the differences between paired and un-paired data, we have:

- Paired training data: $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
- Un-paired training data:
  - Source set: $\{x^{(i)}\}_{i=1}^{N}$ with each $x^{(i)} \in X$
  - Target set: $\{y^{(j)}\}_{j=1}^{M}$ with each $y^{(j)} \in Y$
  - For example, X is the set of horse pictures, and Y is the set of zebra pictures, where there are no direct correspondences between images in X and Y

5.2 Emoji CycleGAN

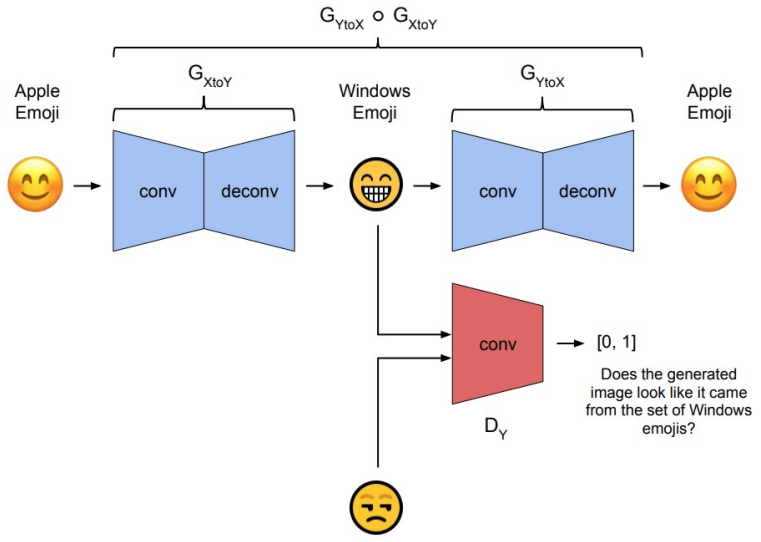

Figure 8: The architecture of the Emoji CycleGAN.

Now we'll build a CycleGAN and use it to translate emojis between two different styles, in particular, Windows ↔ Apple emojis. Figure 8 shows the architecture of the Emoji CycleGAN.

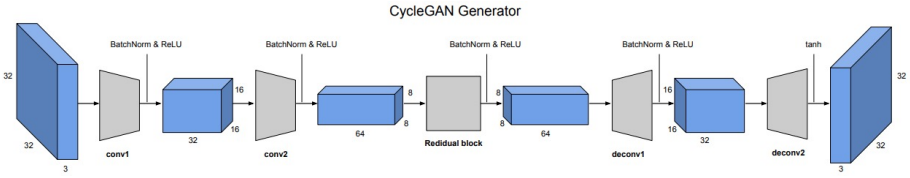

Figure 9: The architecture of the Emoji CycleGAN generator.

5.3 Generator

As is shown in Figure 9, the generator in the CycleGAN has layers that implement three stages of computation: 1) the first stage *encodes* the input via a series of convolutional layers that extract the image features; 2) the second stage then *transforms* the features by passing them through one or more *residual blocks*; and 3) the third stage *decodes* the transformed features using a series of transpose convolutional layers, to build an output image of the same size as the input.

The residual block used in the transformation stage consists of a convolutional layer, where the input is added to the output of the convolution. This is done so that the characteristics of the output image (e.g., the shapes of objects) do not differ too much from the input.

Note: There are two generators in the CycleGAN model, $G_{X \to Y}$ and $G_{Y \to X}$, but their implementations are identical. Thus, in the code, $G_{X \to Y}$ and $G_{Y \to X}$ are simply different instantiations of the same class.

5.4 CycleGAN Training Loop

Finally, we will implement the CycleGAN training procedure following Algorithm 2, which is more involved than the procedure in Algorithm 1. There is a lot of symmetry in the training procedure, because all operations are done for both $X \to Y$ and $Y \to X$ directions.

5.5 Cycle Consistency

The most interesting idea behind CycleGANs (and the one from which they get their name) is the idea of introducing a cycle consistency loss to constrain the model. The idea is that when we translate an image from domain X to domain Y, and then translate the generated image back to domain X, the result should look like the original image that we started with. The cycle consistency component of the loss is the mean squared error between the input images and their reconstructions obtained by passing through both generators in sequence (i.e., from domain X to Y via the $X \to Y$ generator, and then from domain Y back to X via the $Y \to X$ generator). The cycle consistency loss for the $Y \to X \to Y$ cycle is expressed as follows:

$$\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - G_{X \to Y}(G_{Y \to X}(y^{(i)})))^2$$

The loss for the $X \to Y \to X$ cycle is analogous.

Algorithm 2 CycleGAN Training Loop Pseudocode

procedure TRAINCYCLEGAN

Draw a minibatch of samples $\{x^{(1)}, ..., x^{(m)}\}$ from domain X

Draw a minibatch of samples $\{y^{(1)}, ..., y^{(m)}\}$ from domain Y

Compute the discriminator loss on real images:

$$J^{(D)}_{\text{real}} = \frac{1}{m}\sum_{i=1}^{m}(D_X(x^{(i)}) - 1)^2 + \frac{1}{n}\sum_{j=1}^{n}(D_Y(y^{(j)}) - 1)^2$$

Compute the discriminator loss on fake images:

$$J^{(D)}_{\text{fake}} = \frac{1}{m}\sum_{i=1}^{m}(D_Y(G_{X\to Y}(x^{(i)})))^2 + \frac{1}{n}\sum_{j=1}^{n}(D_X(G_{Y\to X}(y^{(j)})))^2$$

Update the discriminators

Compute the $Y \to X$ generator loss:

$$J^{(G_{Y\to X})} = \frac{1}{n}\sum_{j=1}^{n}(D_X(G_{Y\to X}(y^{(j)})) - 1)^2 + J^{(Y\to X\to Y)}_{\text{cycle}}$$

Compute the $X \to Y$ generator loss:

$$J^{(G_{X\to Y})} = \frac{1}{m}\sum_{i=1}^{m}(D_Y(G_{X\to Y}(x^{(i)})) - 1)^2 + J^{(X\to Y\to X)}_{\text{cycle}}$$

Update the generators

5.6 CycleGAN Experiments

After training the Emoji CycleGAN, pick some images from the emoji dataset of Windows style, use Emoji CycleGAN you trained to convert them to Apple style. And in turn, pick some Apple style emojis and convert them to Windows style.

6 Submission and Grading

After various parts of the assignment are completed, the following files including

1. your code and your trained models
2. a PDF report containing your results and the analysis of your experiments

should be zipped and submitted to ouceecv@163.com. Please sign your name and "Assignemnt3" in your email theme, e.g. "YourName_Assginemt3".

Be sure to finish and submit Assignment 3 before the due date June 6, 2018. Then we will grade your assignment based on the files your submitted. The following is a breakdown of how each part of this assignment is scored:

Part	**Points**
Autoencoder	15 points
Basic GAN	20 points
DCGAN	25 points
*Emoji CycleGAN	10 points (extra)
Report	40 points
Total Points	100 points + 10 points (extra)

Normally, the total score of this assignment is 100 points. But if you can solve the bonus task, you have the chance to get extra 10 points. In other words, you can get 110 points for this assignment at most.

If you have any questions, be free to contact daa@stu.ouc.edu.cn.

7 Further Resources

For further reading on GANs in general, and CycleGANs in particular, the following links may be useful:

1. Unpaired image-to-image translation using cycle-consistent adversarial networks (Zhu *et al.*, 2017)
2. Generative Adversarial Nets (Goodfellow *et al.*, 2014)
3. An Introduction to GANs in Tensorflow
4. Generative Models Blog Post from OpenAI
5. Official PyTorch Implementations of Pix2Pix and CycleGAN