Human Emotion Generation with Generative Adversarial Networks

Yiming Fan, Sicong Liu, Wei Wang

KTH Royal Institute of Technology

Abstract. We have trained a Generative Adversarial Network (GAN) for generating human faces. This network was inspired from Deep Convolutional GAN (DCGAN) and CycleGAN. Given an arbitrary human face with a certain emotion (e.g. happy, sad, ...) we could transform the face into the specified emotion.

1 Introduction & Related Work

1.1 Background: The Uprising of GANs

Deep Learning (DL) is recently a hot topic in the field of Computer Vision (CV). Some deep networks have reached good performance in the specific areas of image recognition, classification and generation. Among those networks, the Generative Adversarial Network (GAN) excels in extracting, learning and generating features.

Introduced by Goodfellow [1] in 2014, GAN inspired us with the idea of *MiniMax* training. However the very first generation of GAN remains lots of drawbacks: The difficulty for controlling the step-size of training both the generator and the discriminator and the difficulty for evaluating the performance [2]. Therefore a more complete and mature architecture of GAN is necessary.

Up till present there are various variations of GANs published out. For instance: Wasserstein GAN (W-GAN) solves the problems of discrete input function space and the loss of gradients [3]. Deep Convolutional GAN (DCGAN) expands the model complexity and reduces the problems of early-terminating and the hardness to converge [4]. The Boundary Equilibrium GAN (BEGAN) provides with an elegant and neat training procedure, and reaches a plausible image generation quality [5].

1.2 Problem Specification

Our problem aims at generating human faces with specified emotions. Given a digital image containing a face of human, we will have to design a model that could generate a new digital image of the same face, but with any emotion specified. That means we will have to transfer the emotion on the face into arbitrarily specified ones.

1.3 Dataset

We choose the Facial Expression Recognition 2013 (*fer2013*) dataset for our problem. It consists of 35887 gray-scale facial images of 48×48 pixels, each of which is labeled with one of seven categories: *angry, disgusted, fearful, happy, sad, surprised* and *neutral*.

2 Detail of Implementation

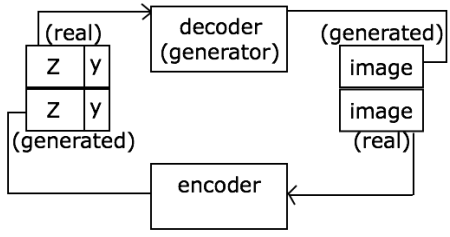

Fig. 1. The overall structure of the network.

2.1 Overall Structure

The network architecture of the whole network looks similar to what of an *autoencoder* structure but not exactly the same. First we feed the network with a digital image $\mathbf{x}$ and a one-hot label $\mathbf{y}$ representing any emotion we like. Then $\mathbf{x}$ is fed into the encoder E to generate the corresponding stochastic noise $\mathbf{z}$:

$$\mathbf{z} = E(\mathbf{x}) \tag{1}$$

The architecture of encoder was inspired from the structure in CycleGAN [6]. It acts like the inverse function of the generator G, which could be also considered as the decoder.

After encoding the generated noise $\mathbf{z}$ together with the label, $\mathbf{y}$ will be sampled through the generator with deep convolutional structure (it was trained within conditional GAN fashion) and we have

$$\mathbf{x}' = G(\mathbf{z}|\mathbf{y}) \tag{2}$$

where $\mathbf{x}'$ is the generated digital image. We write $G(\cdot|\mathbf{y})$ as the form of conditional probability, i.e. the generalized output given condition $\mathbf{y}$. The generated image $\mathbf{x}'$ has exactly the same face as what of $\mathbf{x}$, but with the emotion labeled

as $\mathbf{y}$. If $\mathbf{y}$ represents the identical emotion as the original image $\mathbf{x}$, then $\mathbf{x}$ and $\mathbf{x}'$ will share little or no difference.

We implemented the network based on `Tensorflow`. Due to some package configuration problems we could not manage to train the network on PDC computers and tried training on persona GPUs instead.

2.2 Training Procedures

First we trained the DCGAN within *fer2013* dataset. After successfully training we could generate faces with preset emotions. However we could not yet generate emotions on *arbitrary* human faces. To achieve this we trained E. Then by combining the working encoder E with the generator (decoder) G we obtained a completely functioning network which mean the training procedures were done. We will illustrate those procedures below.

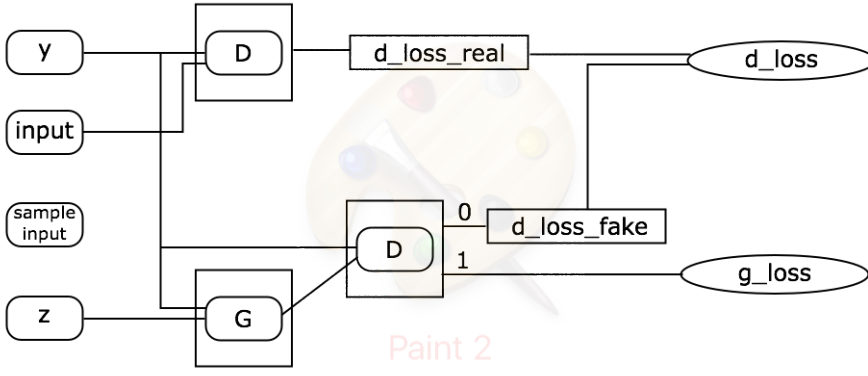

Fig. 2. The computational graph of the DCGAN.

2.2.1 Training the DCGAN. Figure 2 shows the top architecture of the Deep Convolutional GAN. The only top-level modification to the original GAN is the addition of the conditional variable $\mathbf{y}$. In this problem $\mathbf{y}$ represents the one-hot label of various emotions hence it has the size of 7×1. In the process of *forward pass* $\mathbf{y}$ passes the discriminator D and the generator G (which is used during image generation). The discriminator D accepts the generated image $G(\mathbf{z})$ from generator G. After computing the loss via `tf.sigmoid_cross_entropy_with_logits` and `tf.reduce_mean`. Finally we sum up the `D_loss` and `G_loss` respectively

and obtain the objective function (similar to those in Conditional GANs [7]):

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}}\left(S(D(\mathbf{x}|\mathbf{y}))\right) + \mathbb{E}_{\mathbf{x}}\left(S(1 - D(G(\mathbf{x}|\mathbf{y})))\right) \tag{3}$$

where $D(\cdot)$ represents the output of discriminator D, $G(\cdot)$ the output of generator G and $S(\cdot)$ the output of the `tf.sigmoid_cross_entropy_with_logits` function:

$$S(\mathbf{x}, \mathbf{z}) = \mathbf{x} - \mathbf{x} * \mathbf{z} + \log(1 + \exp(-\mathbf{x})) \tag{4}$$

The loss function of the DCGAN is then:

$$L(D) = \mathbb{E}_{\mathbf{x}}\left(S(D(\mathbf{x}|\mathbf{y}),\, \mathbf{1}(D(\mathbf{x}|\mathbf{y})))\right) + \mathbb{E}_{\mathbf{x}}\left(S(D(G(\mathbf{x}|\mathbf{y})),\, \mathbf{0}(D(G(\mathbf{x}|\mathbf{y}))))\right) \tag{5}$$

$$L(G) = \mathbb{E}_{\mathbf{x}}\left(S(D(G(\mathbf{x}|\mathbf{y})),\, \mathbf{1}(D(G(\mathbf{x}|\mathbf{y}))))\right) \tag{6}$$

where $\mathbf{1}(\cdot)$ and $\mathbf{0}(\cdot)$ represent the one-like and zero-like matrices respectively. The detailed computational graphs of D and G are shown.

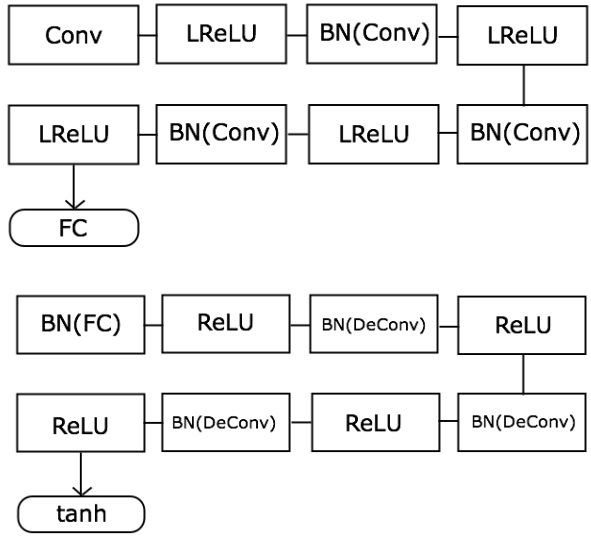

Fig. 3. The computational graphs of the discriminator D (above) and the generator G (below). **Explanation on abbreviations**: `Conv` for 2-dimensional convolution, `LReLU` for leaky `ReLU`, `BN(Conv)` for *Batch Normalization* followed by 2-dimensional convolution, `FC` for fully-connection, `DeConv` for 2-dimensional *De-convolution* and `tanh` for the hyperbolic tangent function.

After computing the loss we update the hyper-parameters of G and D by *Adam-optimizer* [8] which is an advanced version of the *AdaGrad*. For each

timestep t w.r.t. each batch of data $\mathbf{x}$ we have:

$$g_t \leftarrow \partial_\theta f_t(\theta_{t-1}) \tag{7}$$
$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \tag{8}$$
$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \tag{9}$$
$$\hat{m}_t \leftarrow m_t / (1-\beta_1^t) \tag{10}$$
$$\hat{v}_t \leftarrow v_t / (1-\beta_2^t) \tag{11}$$
$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \tag{12}$$

where θ is the hyper-parameter to be updated, α the learning rate, β_1 and β_2 the exponential decay rates for the moment estimates, m the first moment vector and v the second moment vector.

By updating hyper-parameters stochastically we will eventually reach the optimum. In practice we may adjust the step-size for training G in order to prevent the early-termination of the discriminator D. For ease we may simply change the times of updating G to obtain a suitable step-size.

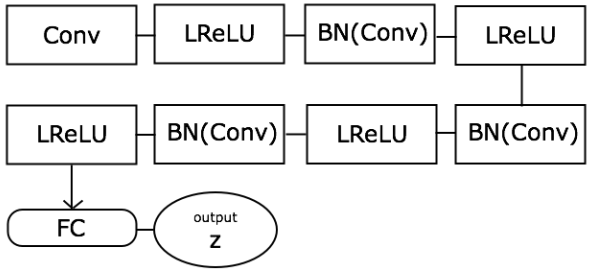

Fig. 4. The computational graph of E.

2.2.2 Training the Encoder. After training a well functioning generator G we will step forward to train the encoder E. The computational graph for E is shown. Similar to what of the discriminator D, the *fully-connected* layer of E gives $\mathbf{z}$ as the output instead of a binary value. Given an image $\mathbf{x}$ as the input the encoder will output the stochastic noise $\mathbf{z}$ with the maximum probability such that

$$\mathbf{z} = E(\mathbf{x}) = E(G(\mathbf{z})) \tag{13}$$

Our aim is to train an encoder that perfectly acts like the 'inverse function' of G. Therefore the loss function on E is defined as:

$$L(E) = \sum_{\mathbf{x}} (\mathbf{x} - G(E(\mathbf{z})))^2 \tag{14}$$

We will update the hyper-parameters using the same optimizer as above. Only when the network reaches its optimum can we assert the completion of training the whole network.

2.3 Some Training Notes

As stated above, we implemented the network within `Tensorflow`. The hyper-parameters are similar as recommended in [4]. The model was trained with stochastic gradient descent (SGD) with a mini-batch of 64. All weights were initialized as zero-centered normal distribution with standard deviation of 0.02. The leak slope of leaky `ReLU` is 0.2. We used *Adam optimizer* with learning rate of 0.0002 and momentum β_1 of 0.5. The model was trained for 50 epochs on a *GeForce GTX 1080* graphic card, and the overall elapsed time was around 100 minutes.

3 Result of Experiments

When training the generator G, we randomly sampled 64 points of stochastic noise $\mathbf{Z}$ after training every 100 batches. With those sampled noise we generated images through G.

Fig. 5. Trained pictures after various epochs(from left to right): 0, 6, 12, 18, 24, 39.

Figure 5 shows a gallery of sample images after various epochs of training. Looking through the gallery we notice that the generated images with the first

few epochs are blurred with lots of noises. This is because neither the generator G nor the discriminator D were sufficiently trained. With the number of epoch increasing we could observe more sampled images being distinguishable and clearer, as hyper-parameters optimized by *Adam optimizer* plateaus.

The sampled images after 24 and 39 epochs were having increasingly higher contrasts, which indicated the expanding output range of the *fully-connected* layer. This also indicates the hyper-parameters of G converges to a satisfactory level.

We take 2 steps for testing the performance of the completely functioning network. First we feed original images (randomly selected from *fer2013*) into the encoder E which produces stochastic noise vectors $\mathbf{z}$. Then we feed $\mathbf{z}$ into G with label 3, which was supposed to form *happy* emotions from original faces. The outcome could be observed in Figure 6.

The comparison indicates that after decoding, most of the images were transfered into *happy* emotions "successfully" with similar outlines and relatively low inaccuracy by intuition. On the other hand, generated images have low contrast after the *fully-connected* layer of generator G which means there were information loss in vectors $\mathbf{z}$ after encoder E.

Fig. 6. Original faces and decoded faces

Moreover, there were some failures upon transferring. Some were due to the wrong storage of datasets such as the fourth images in the last row and others were probably caused by imperfection of encoder E which caused some losses through network, such as the face of *baby* in the second row. Similarly, some pictures were decoded with gender transfer. Taking the first images for instance, the face transfered from male to female, while keeping the outline similar.

Our experiments were not limited to one single emotion. We put original images into E network for generating stochastic noise vectors $\mathbf{z}$. Then they were combined with different labels $\mathbf{y}$ and were taken as input of G. Different labels $\mathbf{y}$ (range from 0 to 6) could influence outcoming faces, which were shown in Figure 7. Each cluster of emotions with different labels were compared with those original faces and compared with self-contained meaning as well. Among all generated clusters, there were quite high percentages of face emotions consistent with their own labels. Especially for image clusters with label 3 and label 5, almost all images looked "successful" to some extent due to both smiling and surprising emotion had unique shapes of mouths whereas other cluster were not so distinguishable but still looked acceptable.

Fig. 7. Test pictures with different emotion after decoding: angry, disgusted, fearful, happy, sad, surprised

Above results demonstrated that it is easier to train generator by using batch normalization, appropriate activation function and replacing pooling layers with fractional-strided convolutions. With the number of epoch increasing, trained images becomes more distinguishable. It also shows that training encoder with CycleGAN is feasible, regardless some deformed transfer.

4 Conclusion

– We fine-tuned the networks inspired by DCGAN and trained our own dataset *fer2013* which was not conducted identically with [4].

- We made an extension on training an encoder inspired by CycleGAN and successfully achieved the transfer from encodes (stochastic noise $\mathbf{z}$ generated from E) to decodes (image data $\mathbf{x}$ generated from G). This encoder together with the Deep Convolutional GAN are encapsulated in the top-level architecture of an *autoencoder*.
- We achieved *image-to-image style transfer* eventually. The output galleries are imperfect and somehow flawed but satisfactory and acceptable. We consider it a success of our project.
- The design, implementation and fine-tuning of our project was time-consuming and there remains many uninterpretable points in the latent layers. There remains some weird output images and some are even inhuman. In the future we might improve the network and make a quantity analysis among encoders and decoders.

References

1. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks (Jun 2014)
2. Arjovsky, M., Bottou, L.: Towards principled methods for training generative adversarial networks (Jan 2017)
3. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan (Mar 2017)
4. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks (Jan 2016)
5. Berthelot, D., Schumm, T., Metz, L.: Began: Boundary equilibrium generative adversarial networks (Apr 2017)
6. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks (Mar 2017)
7. Mirza, M., Osindero, S.: Conditional generative adversarial nets (Nov 2014)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (Jan 2017)