Optimization of Unpaired Data Translation

Jalansh Munshi, Raksha Rank, Dharmil Patel, Sakshee Patel, Akshat Kaneria

Email: {201601042, 201601055, 201601103, 201601132, 201601242}@daiict.ac.in
Assigned by: *Prof. (Dr.) Nabinkumar Sahu*

Dhirubhai Ambani Institute of Information and Communication Technology,
Gandhinagar, India

1 Abstract and Introduction

Image translation is a class of vision and graphics problem, where the goal is to map the source distribution to the target distribution. This task can be achieved comfortably using paired images. However, paired images might not be available for many other tasks. We shall be working on an approach to translate an image from a source domain X to a target domain Y in the absence of paired examples.

In our project, we present an optimization problem which captures the special characteristics of, for example, one set of images and attempts to translate these characteristics into another set of images in the absence of any paired training data. This problem is broadly known as unpaired image-to-image translation [1]. We are given two sets of data, one of domain X, and a different set of domain Y. We may train the mapping $G : X \rightarrow Y$ such that the output $\hat{y} = G(x)$ is indistinguishable from the data distribution $y \in Y$ by an adversarial network trained to classify $\hat{y}$ apart from y. Theoretically speaking, this objective can generate an output data distribution which matches the original data distribution. However, such a translation does not necessarily guarantee that the mapping takes place in a meaningful way, since there are infinitely many mappings possible.

This problem leads us to add more sense to our problem statement. We emphasize on the fact that our translation should be cycle-consistent. That is, if we translate a sentence from English to Hindi, and then translate it back to English from Hindi, we should arrive back at the original sentence. Hence, we also need to simultaneously add cycle-consistency loss. Furthermore, in order to generate near identity mappings, we also regularize the generator with the help of an identity loss function. The final objective/loss function becomes the following.

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(G, D_X, Y, X) + \mathcal{L}_{cyc}(G, F) + \mathcal{L}_{identity}(G, F), \tag{1}$$

where the losses represent adversarial, cycle-consistency, and identity losses respectively. The above loss function is the function that we need to optimize

in order to achieve the desired mapping. We have divided the optimization task into three different tasks, adversarial loss, cycle-consistency loss, and identity loss.

2 Optimization of Adversarial Loss

The primary objective function of the problem consists of adversarial loss, which was introduced by Goodfellow et. al [2]. Adversarial loss tries to learn the mapping such that the generated data cannot be distinguished from the original data distribution. In addition, the objective function also consists of cycle consistency losses in order to prevent the learned mappings from contradicting each other. Our inherent architecture consists of two generators and discriminators [1]. The generator tries to fool the discriminator by generating fake data, and discriminator tries to classify whether the generated data is fake or real.

First, let us consider the adversarial loss for only one generator (G) and discriminator (D). The data distribution is denoted as $x \sim p_{data}(x)$, and the generated data is denoted by z. The objective function of adversarial loss is given by the Eq. 2.

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x)] + \mathbb{E}_{x\sim p_z(z)}[\log(1 - D(G(z))] \tag{2}$$

Here, D tries to maximize the probability of assigning the correct label to the training samples generated by the G network. Simultaneously, the G network tries to minimize $\log(1 - D(G(z))$ They are basically trying to play a minimax game with the value function $V(D,G)$. For a given value of G, we will first try maximize the cost function w.r.t the classifier D. Then, given the optimal value of the classifier D, we minimize the cost function w.r.t to G. So, Eq. 2 becomes

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x)] + \mathbb{E}_{x\sim p_z(z)}[\log(1 - D(G(z))] \tag{3}$$

Here, p_{data} is the real data distribution, and p_z is the generated data distribution. Eq. 2 can be written as

$$V(D,G) = \int_x p_{data}(x)\log(D(x))dx + \int_z p_z(z)\log(1 - D(G(z))dx \tag{4}$$

Here, z is basically the random noise from which the generator (G) tries to replicate the real data distribution by generating $G(z)$. Our goal is to make $G(z)$ same as the real data samples x. To simplify the Eq. 4 and to convert everything to a single variable, we write $x = G(z) \Rightarrow z = G^{-1}(x) \Rightarrow dz = (G^{-1})'(x)dx$. Hence, Eq. 4 becomes

$$V(D,G) = \int_x p_{data}(x)\log(D(x))dx + \int_z P_z(G^{-1}(x)\log(1 - D(x))(G^{-1})'(x)dx \tag{5}$$

Now, let $p_g(x) = P_z(G^{-1}(x)(G^{-1})'(x)dx$, where p_g is the generated data distribution. Therefore, Eq. 5 becomes

$$\begin{aligned} V(D,G) &= \int_x p_{data}(x)\log(D(x))dx + \int_x \log(1-D(x))dx \\ &= \int_x (p_{data}(x)\log(D(x)) + \log(1-D(x)))dx \end{aligned} \tag{6}$$

We have to first calculate the maximum value of D, for a given value of G. Hence, we will perform

$$\max_D V(D,G) = \frac{\partial V(D,G)}{\partial D(x)} = 0 \tag{7}$$

$$\Rightarrow \frac{p_{data}}{D(x)} - \frac{p_g}{1-D(x)} = 0 \tag{8}$$

$$\Rightarrow \frac{p_{data}\cdot(1-D(x)) - p_g\cdot D(x)}{D(x)\cdot(1-(D(x))} = 0 \tag{9}$$

$$\Rightarrow D(x) = \frac{p_{data}}{p_{data}+p_g} \tag{10}$$

For a given value of G, Eq. 10 gives the optimal value of D. The next task is to find the optimal value f Now, the objective function given in Eq. 6 can be written as

$$C(G) = \min_G V(D,G) = \min_G \int_x (p_{data}(x)\log(D(x)) + p_g(x)\log(1-D(x)))dx \tag{11}$$

In the above objective function, we put the optimal value of D that we found in Eq. 10. Hence, Eq. 11 becomes

$$\begin{aligned} C(G) &= \int_x p_d(x)\log(\frac{p_d(x)}{p_d(x)+p_g(x)}) + p_g(x)\log(\frac{p_d(x)}{p_d+p_g(x)})dx \\ &= \int_x p_d(x)\log(\frac{p_d(x)}{\frac{p_d(x)+p_g(x)}{2}}) + p_g(x)\log(\frac{p_g(x)}{\frac{p_d(x)+p_g(x)}{2}}) - \log 4 \\ &= KL[p_d \parallel \frac{p_d+p_g}{2}] + KL[p_g \parallel \frac{p_d+p_g}{2}] - \log 4 \end{aligned} \tag{12}$$

KL-divergence is a measure of similarity between two quantities. KL-divergence is always ≥ 0. Now, we have to minimize the equation above. In order to minimize it, the value of KL-divergence must be 0.

$$\min_G \max_D V(D,G) = -\log 4 \tag{13}$$

Furthermore, since KL-divergence is a measure of similarity, and its value is 0 for this problem, it implies that

$$\begin{aligned} p_d &= \frac{p_d + p_g}{2}, and \\ p_g &= \frac{p_d + p_g}{2} \\ \Rightarrow p_{data}(x) &= p_g(x), \end{aligned} \tag{14}$$

which is exactly what we wanted to achieve in the first place. This optimization is performed for both the generators and discriminators in our problem statement.

3 Optimization of Cycle-consistency Loss

Adversarial loss alone cannot guarantee that the learned mapping function can map an individual input to the desired output. The objective of cycle consistency loss is to minimize the loss between source and target data distributions. We can take the example of speech data for an example. Cycle consistency loss attempts to preserve the contextual information of the speech during different speech conversions. For two different data distributions x and y, the cycle-consistency losses are given by

$$\mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] \tag{15}$$

$$\mathbb{E}_{y \sim p_{data}(y)}[||G(F(x)) - y||_1] \tag{16}$$

Here, Eqs. 15 and 16 represents forward and backward cycles respectively, and $||\cdot||_1$ depicts the *L1 norm*. In practice, the weighted loss $\lambda_{cyc}\mathcal{L}_{cyc}$ is used in the final objective, where λ_{cyc} is the trade-off parameter for cycle consistency loss, which signifies its importance in the final objective function. The method for optimizing this *L1 norm* is presented in section 5.

4 Optimization of Identity Loss

It is helpful to introduce an additional loss to emphasize the mapping to preserve the composition between the input and output. The generator is regularized so that it produces a near-identity mapping when real samples of the target domain are provided as input to the generator. For instance, for speech data, the goal of identity mapping loss is to preserve the linguistic information. We reconstruct the target distribution and the identity loss distinguishes between real and fake distribution. The identity loss for two data distributions x and y is given by

$$\mathcal{L}_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[||G(y) - y||_1] + \mathbb{E}_{x \sim p_{data}(x)}[||F(x) - x||_1], \quad (17)$$

where $|| \cdot ||$ represents the *L1 norm*. The weighted loss with $\lambda_{id}\mathcal{L}_{id}$ is used in the final objective, where λ_{id} is the trade-off parameter for identity loss, which signifies its importance in the final objective function. The final objective function for the problem with trade-off parameters is as follows -

$$\mathcal{L}_{final} = \mathcal{L}_{GAN} + \lambda_{cyc}\mathcal{L}_{cyc} + \lambda_{id}\mathcal{L}_{id}, \quad (18)$$

The values of λcyc and λ_{id} are empirically chosen. The composition of different losses and trade-off parameters encourages the generator to find the mapping that preserves composition between the input and output. The method for optimizing this *L1 norm* is presented in section 5.

5 Optimizing the L1 Norm

The L1 norm creates a criterion that determines the mean absolute error between each element in the input and the target. The input and target data have same number of elements. L1 norm iterates over all the elements of both the data, calculates the sum of the absolute differences of all elements, and back-propagates the average value of that sum. The model tunes its parameters on the basis of that value. A general form of L1 norm for a batch-size of N is shown in Eq. 19

$$\begin{aligned} l(x, y) &= \mathcal{L} = l_1, ..., l_N \\ l_n &= |x_n - y_n| \\ l(x, y) &= mean(\mathcal{L}) \end{aligned} \quad (19)$$

One of the main optimizing techniques used in machine learning is gradient descent. Gradient descent is a first-order iterative optimization algorithm, which finds the minimum value of a function. As a part of the implementation of our optimization problem on speech data, we have used gradient descent as the inherent optimization algorithm. We have explained it in the next section.

6 Gradient Descent

Gradient descent is used in machine learning for optimizing, i.e. to minimize the cost function, so that we can get models that make correct and accurate predictions. The difference between the actual output and the output predicted by our model is measured using a cost function(C) (a.k.a Loss function). In our case, the cost function is as shown in the Eq. 1. Generally, the cost function is a convex function. Convex function are usually preferred owing to their easier convergence as compared to concave functions. Gradient descent is used when the

parameters of a machine learning model cannot be calculated analytically (for example, by using linear algebra) and thus the parameters need to be searched by an optimization algorithm [3].

Firstly, we initialize the weights of our network randomly to a value which is close to zero. Thereafter, we calculate the gradient, which is mathematically shown by $\frac{\partial c}{\partial w}$. It is a partial derivative of the cost function (c) w.r.t the weights (w). We also use a learning rate α to update the weights with respect to the gradient descent. Basically, the learning rate controls the degree by which we should update the weights w.r.t the gradient. Lower value of α implies a slower convergence, whereas a higher values implies a faster convergence. Mathematically, gradient descent can be shown by

$$w := w - \alpha \frac{\partial c}{\partial w} \tag{20}$$

For the sake of our implementation, we have use the stochastic gradient descent. That is, we update the weights of our model after every iteration in the entire data. An intuitive process of gradient descent is shown in the Figure 1. As shown in the figure, by calculating the derivative, we can determine the direction towards which we need to move, in order to get a lower cost.

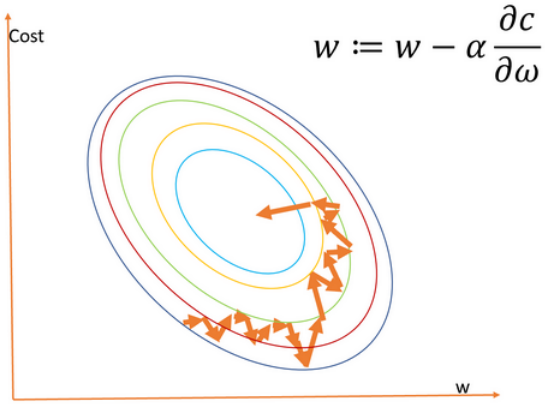

Fig. 1. Depiction of a sample run of the Gradient Descent algorithm.

7 Results and Analysis

We implemented our algorithm using Pytorch library in Python, and trained the model on unpaired speech data. Specifically, we performed the experiment of Non-Audible Murmur (NAM) to Whisper conversion, in order to improve the intelligibility of NAM speech. We used the data CSTR NAM corpus for training purposes [4]. We analyzed the effectiveness of our model by comparing the scatter plots of the original whispered speech and the whispered speech converted by our model. It is clear from the Fig. 2 that our approach is able to capture the original distribution satisfactorily.

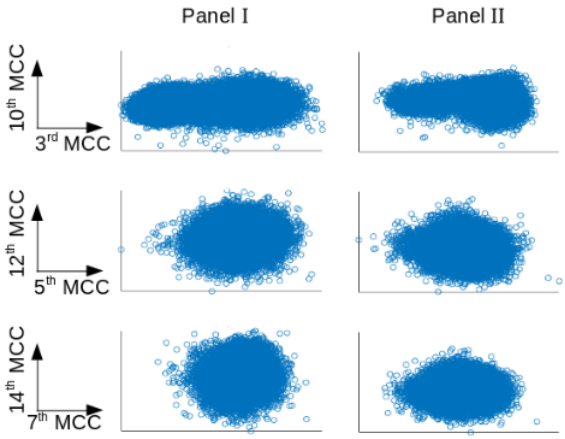

Fig. 2. Scatter plot analysis which shows the effectiveness of our approach.

References

[1] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[3] Renu Khandelwal, "Machine learning gradient descent," in *Medium - https://medium.com/datadriveninvestor/gradient-descent-5a13f385d403, Last Accessed: 14 November 2019*. Medium, 2018.

[4] "Publicly available: The CSTR NAM TIMIT Plus Corpus," URL: `homepages.inf.ed.ac.uk/jyamagis/release/CSTR-NAM-TIMIT-Plus-ver0.81.tar.gz`, {Last Accessed: August 10, 2019}.