Pokemon Generation with Wasserstein GAN

Tiange Xiang

The University of Sydney, Sydney NSW 2006, AU

Abstract. The main topic this paper focuses on is supervised learning. Deep learning models perform well when there is sufficient large labeled data set, with the aid of Generative Adversarial Nets (aka GAN), the stacked neural network models can now not only make image identifications but image generations. However, as many mature experiments and tests may point out, there are certain limitations generating new images with a GAN. By adjusting and replacing loss functions of both generator and discriminator to Earth-Mover (EM) or wasserstein distance, the generatioin process appeared to be much more stable and efficient. Detailed analysis will be discussed, an experiment will be placed and selected results will be presented to prove the capability of a GAN with wasserstein loss function (aka Wasserstein Generative Adversarial Nets or WGAN).

1 Introduction

Automatic data generation is always a huge topic that the filed of computer science concerns. Before the invension of GAN, the task of generating naturally refers to match and classifications. Old-fashion applications are only able to assist data generation to a certain extent, for instance, K-nearest neighbors algorithm [5] were usually prefered to estimate a reasonable prediction rather than creating fresh stuffs. Inspired by neural networks, we hope to implement a particular deep learning model that can handle the generation problems smoothly.

A classic architecture of convolutional neural network is an input layer with the maximum number of neurons in some dimensions followed by convolution layers that repeatedly squeeze the number of neurons in all dimensions through kernels. A flattened dense layer with activation function is usually placed at the tail, acting as the output layer to show results. This kind of convolutional architecture works well when dealing tasks such as vision recognition problems [8] or classification problems. Whereas, when the result is required to be a multi dimensional data, an image, the classic convolutionary workflow will not suffice. It is easy to think of 'reversing' convolutional layers so that giving a high dimension data input, instead of just squeezing the shape, squeeze the number of neurons in only one dimension and, somehow, extend other dimensions. In other word, deconvolutional layers, need to be considered when trying to generate images through a neural network.

Apart from an image generator neural network, another CNN trained with real

image data sets can be adopted to classify whether the given image is fake or not. Just as the format, an adversarial relationship takes place between the two networks. Therefore, the two neural networks can train against each other, finally and hopefully, the generated image is hard to classify, then our goal is satisfied. The exact model and improvement methods will be introduced in the related work section.

The purpose of this paper is to discuss and describe the principle of wasserstein GAN. Besides of the thories, a particular experiment using pokemons as training data sets will be fully implemented. Performance will be evaluated based on reuslts and the whole process will be covered in the following sections as well.

2 Related work

It is commonly accepted that deep learning models are capable of analyzing large data sets with labels. With the idea 'generative', a fabulous combined neural network model 'Generative Adversarial Nets' (aka GAN) [2] is designed to help people be able to spawn reasonable productions automatically. This kind of model gives great result when there are enough training datasets, however, training a GAN is not as perfect as we assume. During the training process, certain disadvantages of using GAN to generate images will be presented.

Regarding the flaw a GAN may have, several derivatives have risen during past years. Mao, X et al. [7] proposed Least Square GAN ,which is a GAN with least square loss function rather than the original JS divergence. With utilizing LSGAN to generate fake images, there is an obvious quality improvement which proves the feasibility of replacing the loss function of GAN. Whereas, training a LSGAN is not effective as we expected. Mao, X et al. contributed another paper [6] focusing on the study of efficiency of LSGAN, demonstrated that it cost much more than training a GAN.
In the same year when Mao, X et al. proposed LSGAN, Arjovsky et al. [1] proposed another improved GAN, Wasserstein GAN, which use earth-mover loss function or wasserstein loss function to replace the original ones. A significant performance improvement is demonstrated during the experiments they have done. More efforts have been contributed to maximize the performance of WGAN.

Based on the basics of GAN, all variants showed possible ways to improve the performance of neural networks. As in this paper, WGAN is adopted to emphasized on. Although, all relevant theories have already been enuncited, there is a significant shortage on real life applications, which is not persuasive enough to prove the capability of both GAN and WGAN outside of a lab. The purpose of this paper is trying to combine both theoretical model and a real task as a whole, and then analyze the experimental results.

3 Method

3.1 Architecture of GAN

As introduced before, a GAN is a stacked model that easily stack two neural networks one after another. This kind of sequce allows the latter neural network with convolutional layers to take the output of the prior network as the input and do classification task, which is similar as discriminating the result of the first network. Therefore, the name 'discriminator' is given to represent that the neural network can classify the input as whether real or fake. While the first network, has to generate a 3D (2D if gray image) matrix as output, which can be interpreted as a generated image and then feed into the discriminator to classify. On the basis of the generating feature, it has been given the name 'generator'. In order to make such architecture work, the input of the generator needs to be decided. Inspired by the structure of autoencoder network, the generator can be reckoned as decoding a vector through repeatedly deconvolutional layers to a multi dimensional matrix. The stacked model can be summarised as giving an input vector, generate a image and classify whether it is real or fake (see Fig. 1).

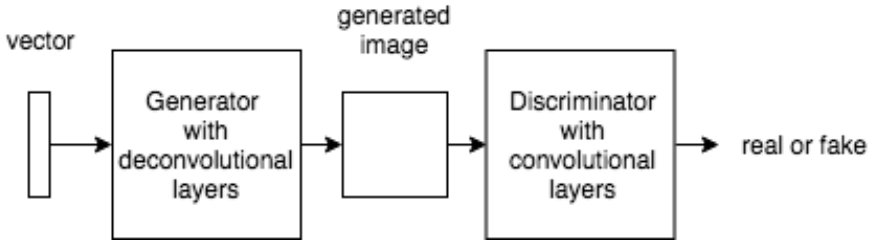

Fig. 1. General workflow of GAN

3.2 Training a GAN

The training process of both neural networks can be achieved by the most general approach: backpropagation [4], which this paper will not concentrate on. Before training, a large set of labeled images will be fed into the discriminator, however, loading all training data every epoch costs non-negligible time. Considering the performance of the training, random selected mini batch is prefered rather than using full batch.
As Goodfellow, I et al. stated, the loss function of a GAN [2] can be described as:

$$V(G,D) = \mathbb{E}_{x\sim P_r}[logD(x)] + \mathbb{E}_{z\sim P_z(z)}[log(1-D(G(z)))] \quad (1)$$

Where P_r is the result distribution of real image and P_g is the result distribution of generated image. To have good results, we hope it can have a perfect classification ability. Therefore, the aim of training the model is trying to minimize the result of equation Eq. 1. For discriminator, we hope to classify every real data to be 1 and generated data to be 0, hence maximizing Eq. 1, loss function for discriminator [2] is:

$$\max_D(\mathbb{E}_{x\sim P_r}[logD(x)] + \mathbb{E}_{z\sim P_z(z)}[log(1 - D(G(z)))]) \tag{2}$$

Notice that only right half of Eq. 1 related to G(z), therefore, the loss function for generator [2] is:

$$\min_G(\mathbb{E}_{z\sim P_z(z)}[log(1 - D(G(z)))]) \tag{3}$$

In practice, it is recommanded not to optimize D ahead of G, a better trained D would result in overfitting. Alternatively, we optimize D k steps and optimize G one step only.
When generator is good enough to generate images that are very similar to the real ones, the discriminator will not classify correct results anymore, which is a fifty-fifty probrability case. In this case, the optimal discriminator $D^*_G(x)$ [2] is:

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{4}$$

As $p_{data} = p_g$ ($D^*_G(x) = \frac{1}{2}$), global minimum of the virtual training criterion C(G) can be now achieved by replacing the original discriminator loss function of Eq. 1 to the optimal one Eq. 4:

$$\begin{aligned} C(G) &= \max_D V(G, D) \\ &= \mathbb{E}_{x\sim P_r}[logD^*_G(x)] + \mathbb{E}_{z\sim P_z(z)}[log(1 - D^*_G(G(z)))] \\ &= \mathbb{E}_{x\sim P_r}[log\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + \mathbb{E}_{z\sim P_z(z)}[log\frac{p_g(x)}{p_{data}(x) + p_g(x)}] \end{aligned} \tag{5}$$

When p_{data} approaches p_g, C(G), also known as JS divergence will be -2log2. Which is the global minimum loss value.

3.3 Potential problems and Wasserstein GAN

Arjovsky et al. stated that using JS divergence to represent the likelyhood between real and generated data distribution is not reliable. As explained in the paper, JS divergence can be mutative. In some cases, the value of JS divergence will maintain at -2log2, which is impossible to provide any help when we are trying to optimize the generator. Hence, leads to mode collapse [1].
A great alternative function to simulate the distance of the two data distributions: Earth-Mover (EM) distance [1] is proposed to replace JS divergence:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma\in\prod(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{6}$$

Where$\prod(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$.
The extrodinary advantage of EM distance can be demonstrated by a simply example [1] that mentioned in the paper.

Example 1. (Learning parallel lines). Let Z ∼ U [0, 1] the uniform distribution on the unit interval. Let $\mathbb{P}_0$ be the distribution of (0,Z) $\in R^2$ (a 0 on the x-axis and the random variable Z on the y-axis), uniform on a straight vertical line passing through the origin. Now let $g_\theta(z) = (\theta,z)$ with θ a single real parameter. It is easy to see that in this case,

- $\mathrm{W}(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|$,
- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} log2 & if\theta \neq 0, \\ 0 & if\theta = 0. \end{cases}$

It is easy to observe from the example that when $\theta_t \to 0$, the sequence $(\mathbb{P}_{\theta_t})_{t\in\mathbb{N}}$ under EM distance, but does not converge under JS divergence. The following figure shows both EM distance and JS divergence as a function of θ [1]:

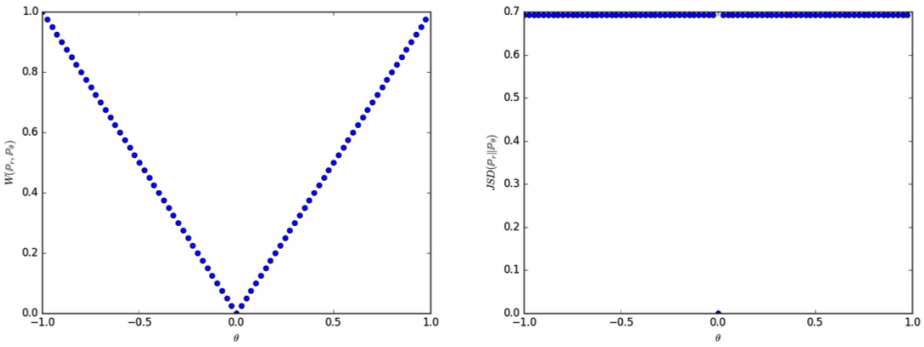

Fig. 2. EM distance vs θ and JS divergence vs θ in example 1.

As former example points out, $\mathrm{W}(\mathbb{P}_0, \mathbb{P}_\theta)$ may have better properties when optimized over $JS(\mathbb{P}_0, \mathbb{P}_\theta)$. However, in Eq. 6, solving $\inf_{\gamma\in\prod(\mathbb{P}_r,\mathbb{P}_g)}$ is beyond the area of computing. Arjovsky et al. proposed a similar equation that utilizes the good features of Lipschitz continuity [1]:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{7}$$

With the constraint of Lipschitz continuity, we can restrict the maximum local difference of every two x in function f(x) within the Lipschitz constant K:

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|| \tag{8}$$

Whereas in Eq. 7, K is replaced by 1, which influences only the growth rate of f(x) but has no effect on the increasing direction.
Notice that in Eq. 7 only right half expression related to generator, the target to train gnerator is equivalent to minimize the right half. The loss function for WGAN generator [1] is :

$$-\mathbb{E}_{x\sim P_g}[f_w(x)] \tag{9}$$

The loss function for WGAN discriminator is the opposite of Eq. 7 [1], which indicates the difference between the wasserstein distances of real and generated data. We want to minimize the expression:

$$\mathbb{E}_{x\sim P_g}[f_w(x)] - \mathbb{E}_{x\sim P_r}[f_w(x)] \tag{10}$$

Implementing the changes of WGAN over GAN in neural networks is simple and straight forward:

- Remove the sigmoid activation layer at the end of discriminator.
  The discriminator of GAN uses sigmoid function to do the binary classification task, whereas, the discriminator of WGAN focuses on regression task.
- Clip the value of trained weights within a small interval.
  This effort contributed by applying the Lipschitz continuity on discriminator layers. The value does not necessarily matter, as the Lipschitz constant K is only used to indicate directions.

4 Experiment

In this section, the specific structure of the WGAN I constructed for the project will be covered. A real life experiment: pokemon generation, will be used to prove the capability of WGAN.

4.1 Implement WGAN

The construction of discriminator is a convolutional neural network (CNN) structure that applies filters to extract various features from the given image. Every convolution layer is followed by a batch normalization [1] layer and an activation layer. Instead of using ReLU activation function, LeakyReLU is preferred as no neurons will be dead (never activated) during the training process. Moreover, a dropout layer can be considered to be put after every activation layer, hence avoiding to rely on some particular neurons excessively. The specific network structure with primary layers is given by:

[1] Gulrajani, I el at. [3] proposed an optimized WGAN: WGAN-GP, which recommands to use no batch normalization layers in discriminator due to the influence of weight clipping.

Layer	Shape	Activation	Batch normalization
input	batch size, 3, 128, 128	no	no
convolution	batch size, 16, 64, 64	LeakyReLU	yes
convolution	batch size, 32, 32, 32	LeakyReLU	yes
convolution	batch size, 64, 16, 16	LeakyReLU	yes
convolution	batch size, 128, 8, 8	LeakyReLU	yes
dense	batch size, 1, 8, 8	no [1]	no

Compared to the discriminator, there are many approches to implement the upsample process in the generator. Regular convolution layers are supposed to be replaced by deconvolution layers or transposed convolution layers. It has been proved during experiment that, kernels and padding method need to be maintained the same throughout the whole network, or apparent fluctuations will be detected in generated results. Based on above conclusions, input values will be transmitted through the combined transposed convolution, batch normalization and activation layers repeatedly:

Layer	Shape	Activation	Batch normalization
input	batch size, 100 (noise dimension)	no	no
dense	batch size, 256*4*4, 1, 1	ReLU	yes
reshape	batch size, 256, 4, 4	no	no
transposed convolution	batch size, 128, 8, 8	ReLU	yes
transposed convolution	batch size, 64, 16, 16	ReLU	yes
transposed convolution	batch size, 32, 32, 32	ReLU	yes
transposed convolution	batch size, 16, 64, 64	ReLU	yes
transposed convolution	batch size, 3, 128, 128	Tanh	no

As referenced in the paper [1], hyperparameters are determined as:

Hyperparameter	Value
Mini-batch size	32
LeakyRelu slope	0.2
RMSprop optimizer learning rate	0.00005
Batch normalnization momentum	0.8
Noise dimension	100
kernel size	5 x 5
Padding type	same
Weight clip value	[-0.01, 0.01]
Discriminator training epoches	5 per generator training epoch

After successfully constructing a WGAN model, we are desired to know whether the model can handle real life tasks. An experiment of generating pokemons will be used to prove the capability of WGAN.

[1] We use sigmoid activation function to do binary classification task in GAN, but in WGAN, we need to simulate the wasserstein distance.

4.2 Data preparation

Training a WGAN requires as many data sets as possible, whereas the original pokemon images are not sufficient to support the diversity of results. Following image augmentation techniques can be used to enrich the data set.

1. Flip images both horizontally and vertically.

Fig. 3. original image, horizontally flipped image and vertically flipped image

2. Crop blank area of one side and pad the other side.

Fig. 4. original image, cropped and padded right, and cropped and padded left

Another two directions can also be cropped and padded using the same technique.

4.3 Result

With a fully implemented WGAN and an abundant pokemon training dataset, we can now begin to feed real images and train the model. Online cloud deep learning server is recommandedd as we can use GPU to boost the training speed. Although it can be trained on CPU as well, the process will be expected to be extremely slow (15 minutes per 100 epoches on 128x128 RGB images). In this

section, results with and without image augmentation techniques will be compared, and a final generated pokemon image will be demonstrated.

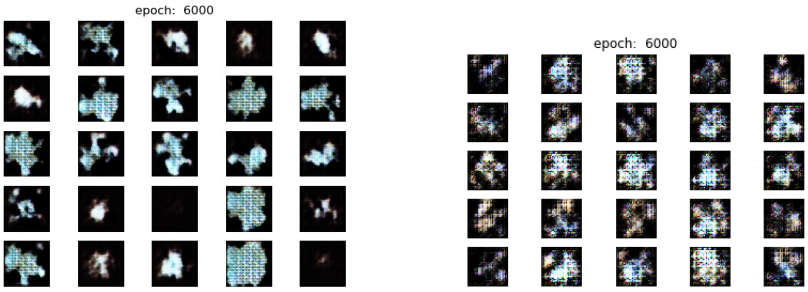

Fig. 5. Generated pokemon with(left) and without(right) image augmentation techniques both at 6000^{th} epoches.

From Fig. 5, it is clear that the loss function of WGAN with a much larger data set converges significantly slower, therefore, the training process goes slow at the early epoches (6000 in Fig. 5). While the right hand side has already finished sketching the countour of pokemons, the left hand side is still working on it.

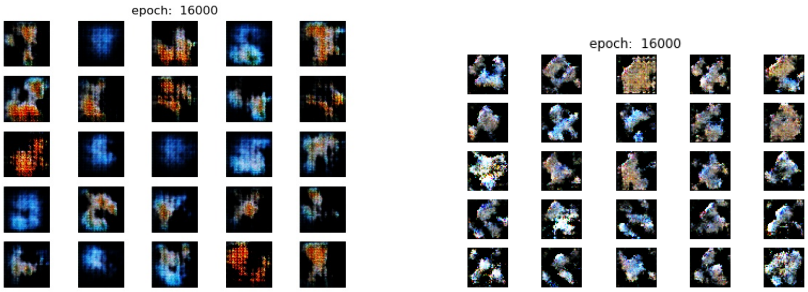

Fig. 6. Generated pokemon with(left) and without(right) image augmentation techniques both at 16000^{th} epoches.

Fig. 6 reveals that the generated pokemon without image augmentation begins to lack diversity and creativity as a result of insufficient training set, on the other side, the generated pokemon with image augmentation techniques finally begin to fill the details in each 'pokemons'. Also, we can observe that there are blur boundaries of all bright parts in the left figure, which is exactly what real

pokemon images have.

The final result of the generated image is adopted at epoch 33100:

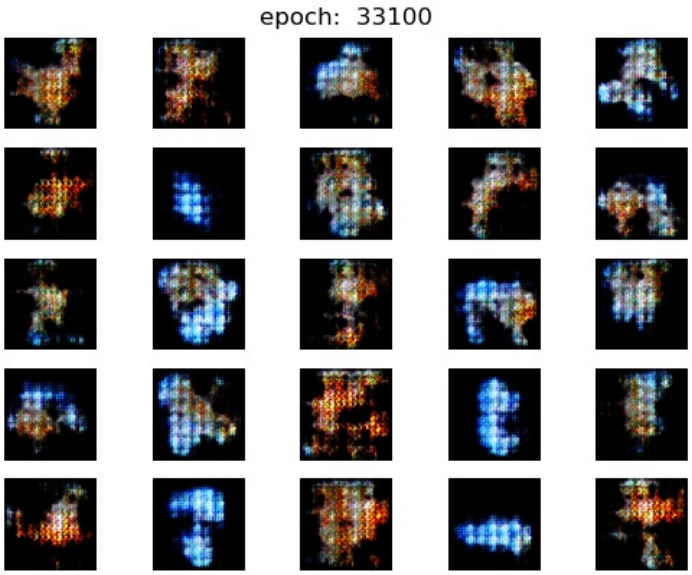

Fig. 7. Presented generated pokemon at 33100^{th} epoches.

5 Conclusion and future work

This paper demonstrated the principle and theory behind GAN and WGAN. It was shown that the limitations of GAN may lead to significant flaw on data generation. By replacing original loss functions of GAN to wasserstein distance (WGAN), the results would not have any mode collapse at all. A pokemon generation experiment with detailed process and explanation was made to prove the capability of WGAN, experimental results were presented and analyzed.

However, the final reuslt is not as satisfied as expected, furthur study and modification should be carrying out. The future work can start with the following points:

- Remove batch normalization layers in discriminator due to the influcence of weight clipping and apply a gradient penalty as a part of the loss function [3].
- Both generator and discriminator can use a smaller kernel size for convolution and transposed convolution layers in order to eliminate the hollow black area which may appear in generated images.

When tasks are relevant with generating data, WGAN model is great enough to be adopted. However, the loss function is hard to converge which leads to an extremely tedious training process. Moreover, if the model has been trained insufficiently, the quality of generated data will be far from the expectation. More works need to be done aims at improving the performance of GAN or WGAN.

References

1. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017)
2. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
3. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: Advances in Neural Information Processing Systems. pp. 5767–5777 (2017)
4. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural networks for perception, pp. 65–93. Elsevier (1992)
5. Laaksonen, J., Oja, E.: Classification with learning k-nearest neighbors. In: Neural Networks, 1996., IEEE International Conference on. vol. 3, pp. 1480–1483. IEEE (1996)
6. Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z., Smolley, S.P.: On the effectiveness of least squares generative adversarial networks. IEEE transactions on pattern analysis and machine intelligence (2018)
7. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Smolley, S.P.: Least squares generative adversarial networks. In: Computer Vision (ICCV), 2017 IEEE International Conference on. pp. 2813–2821. IEEE (2017)
8. Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. In: Proceedings of the IEEE international conference on computer vision. pp. 1520–1528 (2015)