"La Sapienza" University of Rome
Faculty of information engineering, information technology and statistics
Department of informatics, automation and control engineering
"ANTONIO RUBERTI"
Degree program: Artificial Intelligence and Robotics

SAPIENZA
UNIVERSITÀ DI ROMA

PROJECT

Conditional GAN

Student: OREL Egor
Matricola: 1836231

Teachers: PIRRI Fiora, ALATI Edoardo

Rome, 2019

What is GAN: Generative Adversarial Networks were introduced as a novel way of generative models training. They consists of two 'adversarial' models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. Both G and D could be a non-linear mapping function, such as a multi-layer perceptron.

To learn a generator distribution p_g over data x, the generator builds a mapping function from a prior noise distribution $p_z(z)$ to data space as $G(z;\ \theta_g)$. And the discriminator, $D(x;\ \theta_d)$, outputs a single scalar representing the probability that x came form training data rather than p_g.

G and D are both trained simultaneously: we adjust parameters for G to minimize $log(1\ -\ D(G(z))$ and adjust parameters for D to minimize $logD(X)$, as if they are following the two-player min-max game with value function $V(G, D)$:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D\big(G(z)\big))].$$

Conditional GAN (cGAN): Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. y could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer.

In the generator the prior input noise $p_z(z)$, and y are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. In the discriminator x and y are presented as inputs and to a discriminative function (embodied again by a MLP in this case). The objective function of a two-player minimax game would be as:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x|y)] + E_{z\sim p_z(z)}[\log(1 - D\big(G(z|y)\big))].$$

A simple structure of a cGAN is illustrated at the Figure 1:

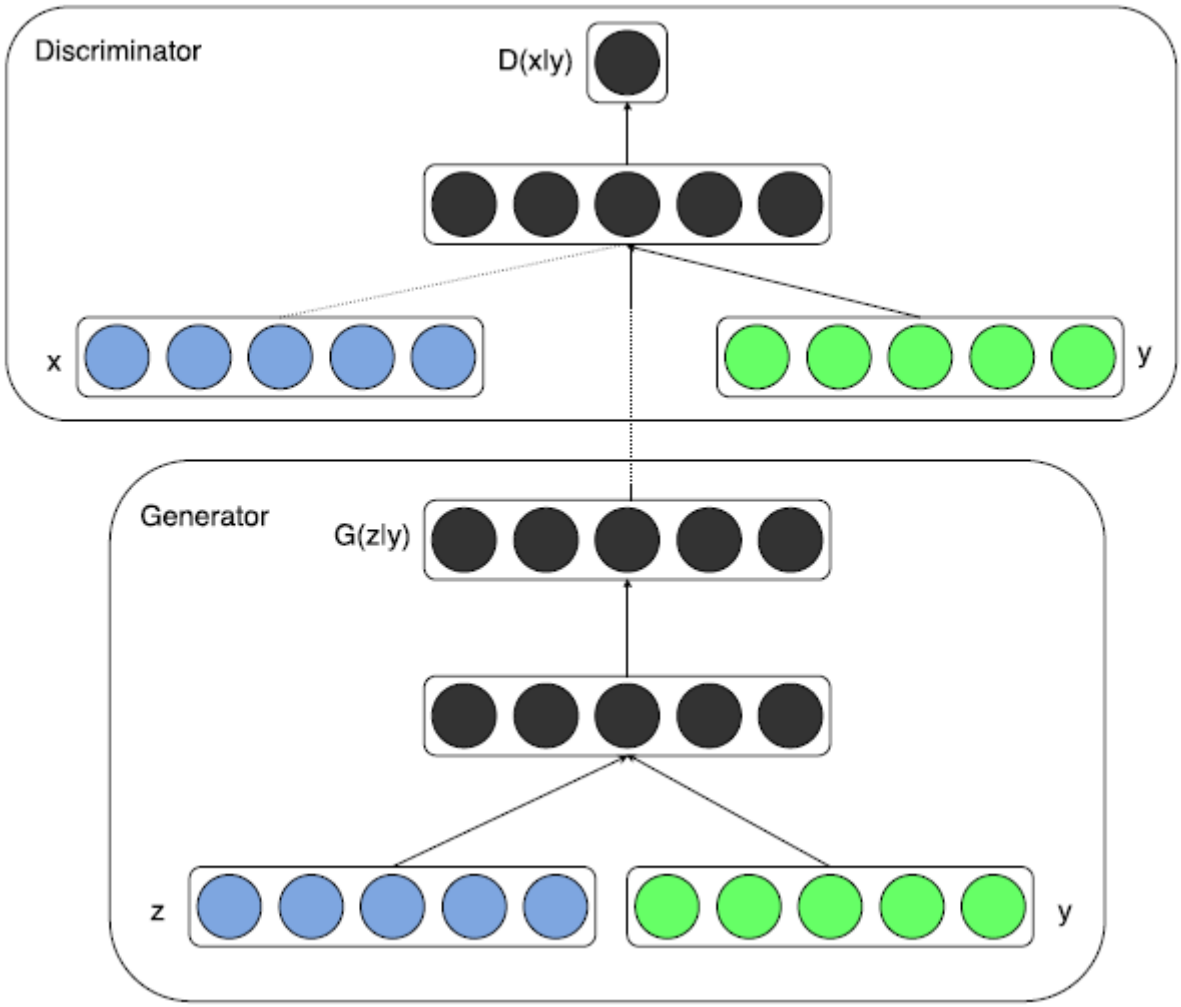

Figure 1 – Conditional adversarial network strucrure

Starting point of the project: the structure and code of the Tensorflow cGAN implementation (training on the standard MNIST dataset) were found here: 1) a blog about cGANs: https://wiseodd.github.io/techblog/2016/12/24/conditional-gan-tensorflow/ ; 2) GitHub: https://github.com/wiseodd/generative-models/blob/master/GAN/conditional_gan/cgan_tensorflow.py

Task

Adaptation of the model for my purposes and training it on a dataset built by myself.

Data

Training corpora consists of photos of a car from front, back and side (Figure 2). It was divided to 3 folders in accordance with names of data classes. Every folder contains 90 photos of the car from different points of view.

Figure 2 – Photos of the car from back, front and side

Preprocessing

Features. Photos were divided to 3 folders in accordance to its class. Each photo was translated from the original RGB photo (4000*3000 pixels) to 1-chanel gray-scale image of size 36*36 pixels. Such small size was chosen to allow easier training of the model (I used my PC without GPU for training). Then every image was converted to a flatten numpy array of size 1296 (= 36*36*1).

Labels. Every class of photos was specified by a one-hot label:

- back – [1,0,0];
- front – [0,1,0];
- side – [0,0,1].

Images after preprocessing are shown at the Figure 3.

Figure 3 – Images after preprocessing

Dataset. Features and their respective labels were formed as matrices of sizes (270, 1296) and (270, 3) respectively. Then they were registered as an iterable dataset using tensorflow.data.Dataset instrument. Outputs from the formed dataset directly come to the tensorflow graph.

Model training

Tensorflow calculating graph was created according to the form of the Figure 1. In the generator net, a noise prior z with dimensionality 100 was drawn from a uniform distribution within the unit hypercube. Both z and y are mapped to the hidden layer with Rectified Linear Unit (ReLu) activation , hidden layer size is 1024. We then have a final sigmoid unit layer as an output for generating the 1296-dimensional samples. The discriminator net has the same structure as the generator one: we map x and y, G(z|y) and y to the hidden layer of size 1024 with ReLu activation function. Then the sigmoid layer tells us the result: whether the generated image fake or not. Training is performed using Adam optimizer.

During the training process, the following algorithm was repeating: 5 training steps of the discriminator → 2 training steps of the generator → 5 steps D →2 steps G→… This approach helps to avoid the discriminator winning too quickly, thus prevents premature stopping of the training process. I tried different combinations of numbers of training loops, but this one gives acceptable results faster.

Training results at iterations 0, 2, 6, 20, 40 are shown at the Figure 4.

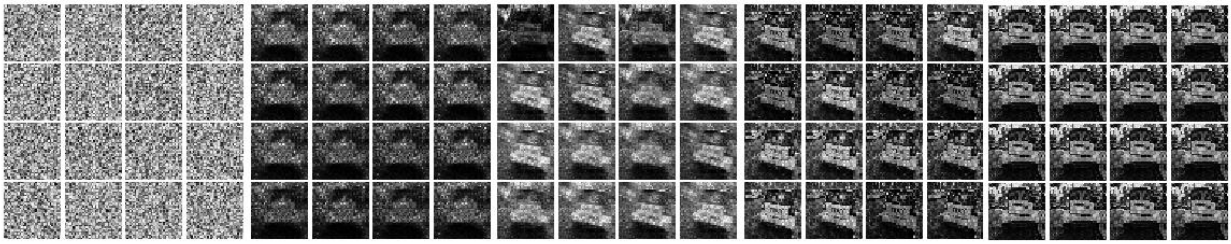

Figure 4 – Intermediate results of training at iterations 0, 2, 6, 20, 40

Results

After training 900000 iterations (each iteration consists of 5 discriminator + 2 generator training loops) using Intel Core-i5 machine with 4Gb CPU, the following results were obtained:

Figure 5 – Generated image of the car (back view)

Figure 6 – Generated image of the car (front view)

Figure 7 – Generated image of the car (side view)

At these images we can see that the trained cGAN gives good results when it is asked to reconstruct a back view. Front view is reconstructed with very high level of noise. Side view is reconstructed very bad and seems more similar to the back view than to the side view.

Losses dynamics are presented at Figure 8 and Figure 9. It is possible to notice that training process converges to the best result (the highest G_loss and the lowest D_loss) at ~40000 step. It means that for better efficiency and time savings earlier stop is needed during the next training session.

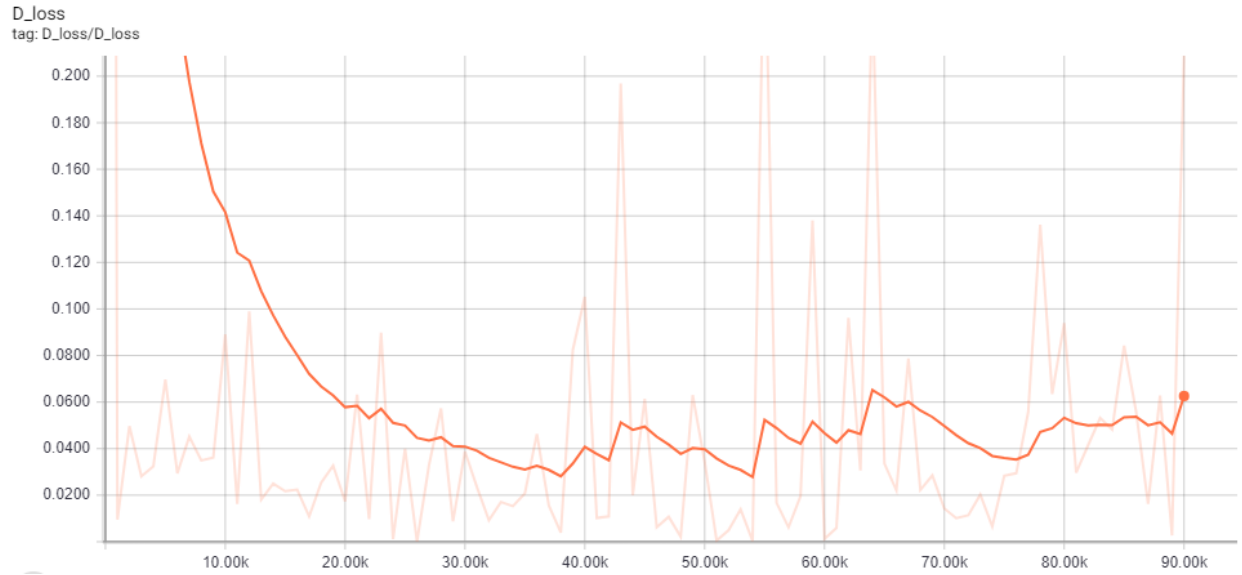

Figure 8 – Dynamics of the discriminator loss function

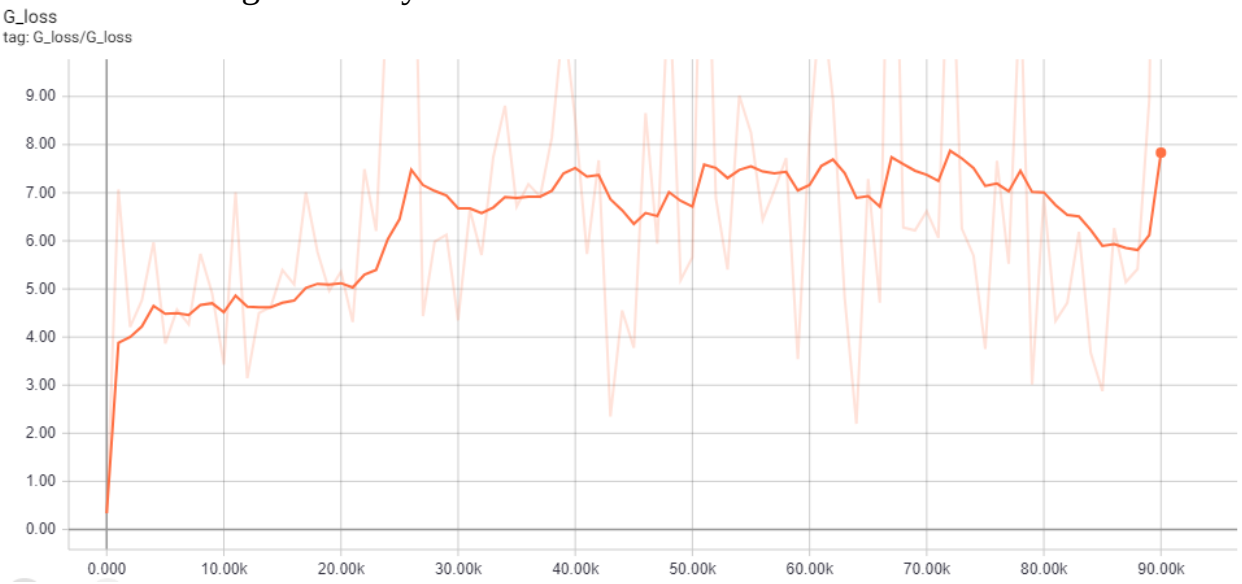

Figure 9 – Dynamics of the generator loss function

Very interesting dynamics is formed by weights and biases histograms (Figure 10, 11). We can notice very fast convergence of generator's weights at initial steps of the "competition" and convergence of generator's biases at ~40000 training steps. It means that training of the generator is finished at this point.

Weights of the discriminator are stable from the starting step. The only thing that is changing are biases of the 2^d layer (with the sigmoid activation function).

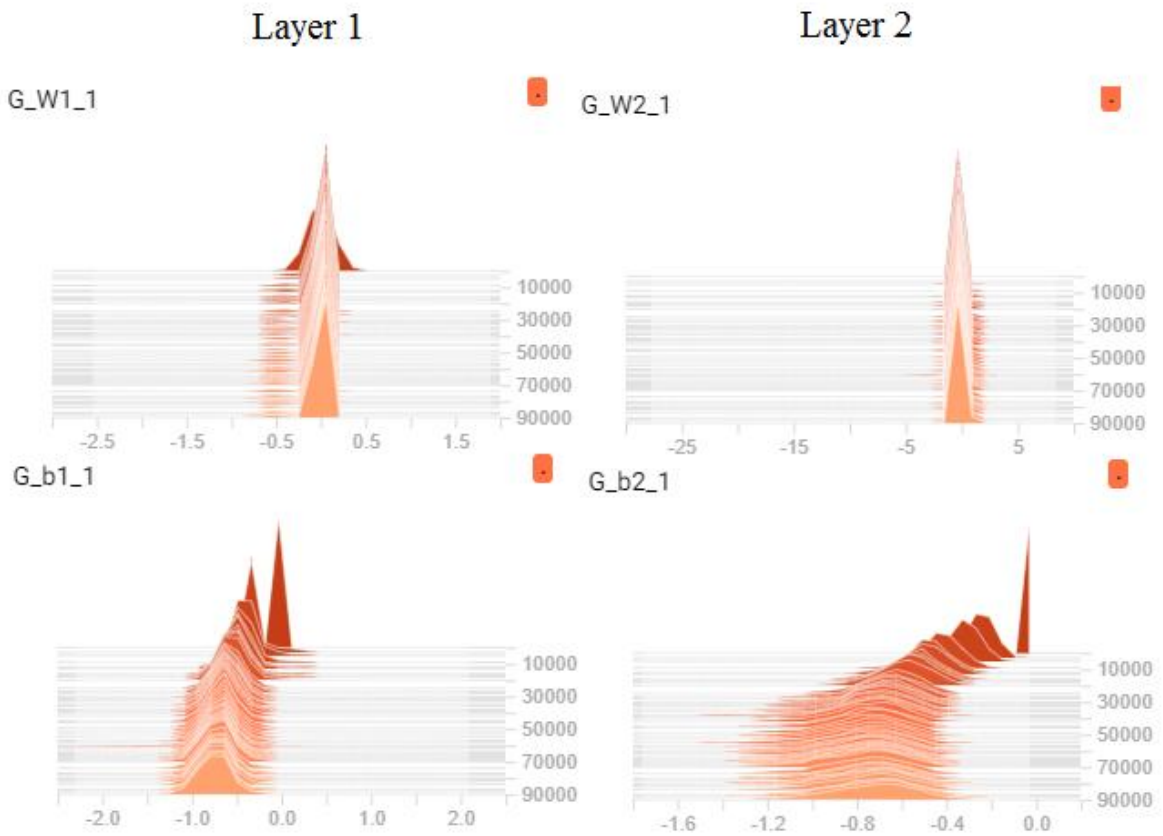

Figure 10 – Weights (1st row) and biases (2d row) histograms of the generator during the training process.

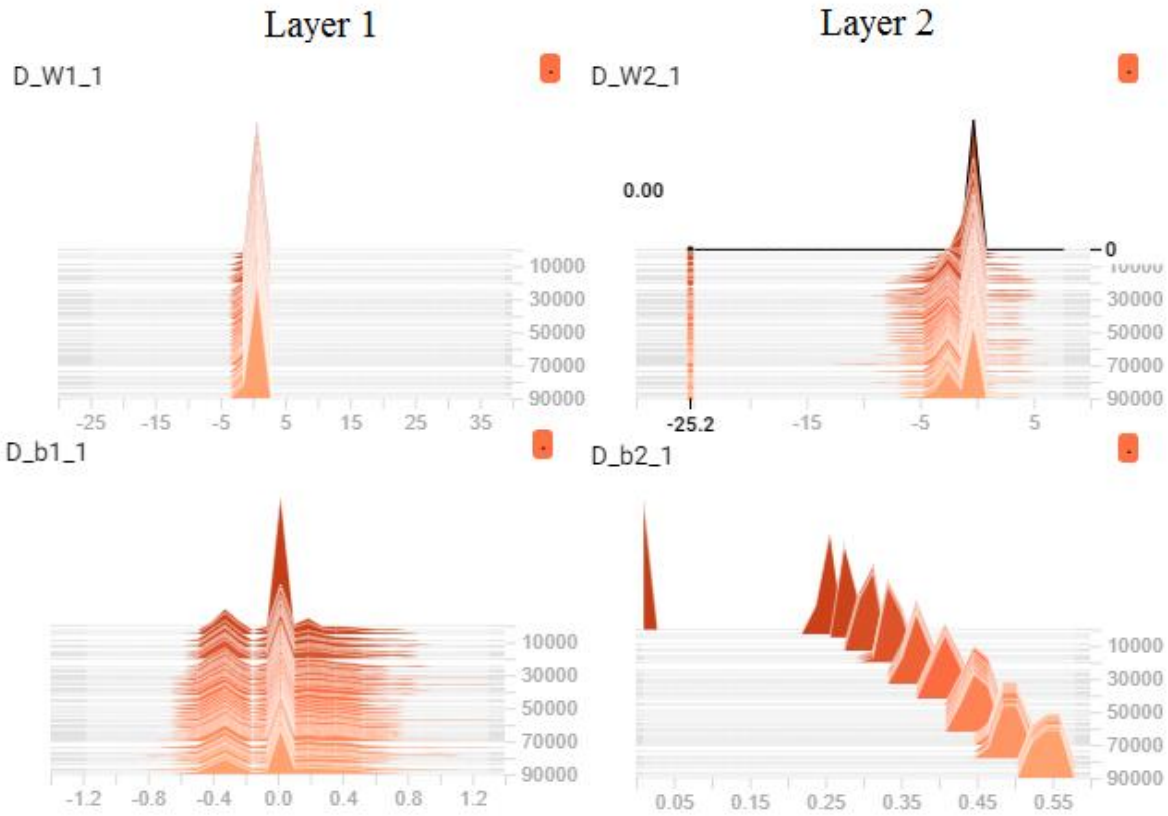

Figure 11 – Weights (1st row) and biases (2d row) histograms of the discriminator during the training process.

Conclusions

Why these results can be such different for these 3 groups?

The first possible issue is that we need much larger size of the dataset for every class of objects, it will give to the network more understanding about image features. In the current case, we have relatively small amount of photos with many different features to be recognized. This lack of data could provoke such a similarity between the results of back and side views.

The second possible issue is that the size of hidden layers can be increased to recognize more possible pixels configurations faster and to allow more flexibility to the weights configuration. Or the training time should be longer. But this requires an increase in computing power.

The third possible issue is that combination of generator and discriminator training cycles can be tuned to find better correlation to improve “the competition”.

Of course, the hyperparameters of neural networks have to be tuned more to achieve better results.

The general meaning of the training dynamics presented above is that we have very fast convergence to a local solution. Despite the obtained intermediate results, the neural network under study is not capable of strong generalization and should be further developed.