GANS ON MNIST DATASET

INTRODUCTION

In this assignment the goal is to analyze the latent space learned by GANs on MNIST dataset. In fact, the Generator neural network receives as an input a random variable “z”. This variable, also called *latent variable* is randomly generated with a given probability distribution (usually Gaussian). Depending on the z value, the network G will generate new images from this noise vector. From the noisy vector, initially the G net creates a “fake digit”; this created digit can be very far from the shape of an actual digit, but since there is another net, called Discriminator, whose aim is to distinguish fake digits from actual digits, the Generator network becomes smarter and smarter in generating these digits. However, as soon as the Generator network becomes closer and closer in fooling the Discriminator, the Discriminator too becomes smarter in order not to be fooled. This basic principle, where the two networks reciprocally improves themselves, it is exploited in the so called Generative Adversarial Networks.

In the provided code, this is already efficiently done (notice that it is usually very difficult to train a GAN) and when train is completed, a complete trained model both for G and D are saved in order to be utilized without the need to re-run the training every time.

Since the Generator is already defined, it is possible to give it as an input a noise vector z to see the resulting output digit.

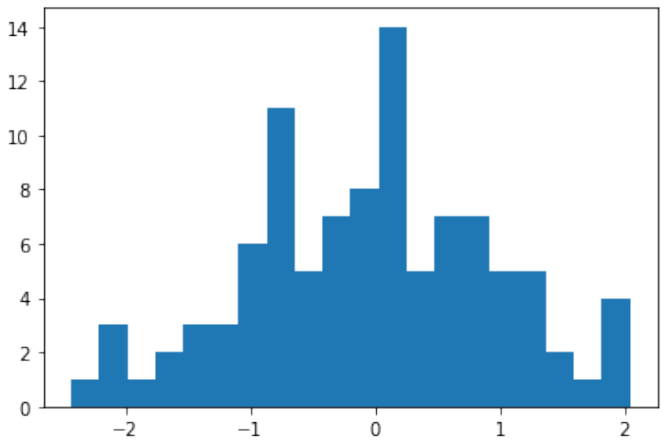
IMPLEMENTATION

As you can see from the Notebook, the first part of the assignment has been realized defining a noise latent vector z1, whose dimension are equal to 1x100 because the desired output in this case is only one digit. The value of this tensor is printed on video together with an histogram representing its distribution. A selected and saved example for z1 is the following:

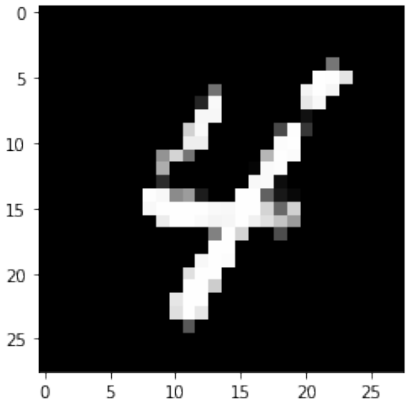
Tensor value:

```
tensor([[-0.5246, -2.1235, -0.8452, -1.6297, -0.1062, -1.4069, -2.0009,  0.0662,
         -0.4463, -0.8619,  0.5718,  1.8879,  0.6554,  1.1358,  1.6444, -0.6465,
         -1.2378,  0.5754, -0.1568, -1.9519, -0.2154,  0.3608, -0.3135,  0.9309,
         -0.6639,  0.9428, -1.0353, -0.5890, -0.1994,  1.3495, -1.0097,  0.2016,
          0.8989, -0.7107,  0.9155,  2.0404,  0.9524, -0.2804,  0.4828, -0.2753,
         -0.9204,  0.8397,  0.3279,  0.4686,  0.7144, -0.6699,  1.3064,  2.0039,
          1.2647,  0.1152,  0.1417,  1.2936, -0.7121,  0.5539,  0.0391,  0.1821,
          0.2236, -0.6630, -0.2366,  0.3563,  0.0712,  0.9412, -1.4426, -0.7424,
          1.5398, -0.4837,  0.0224, -2.0344, -1.7195, -1.1574,  0.5373, -0.2874,
          0.9126,  0.0864,  1.5405, -0.1713, -0.0750,  1.8937, -0.8164, -1.0415,
          0.0110, -0.0855,  0.1603,  0.0712, -0.8584,  0.0669, -1.4125,  0.7162,
         -0.0803, -1.2156,  0.3864, -0.4687,  0.5058, -1.0830,  0.7523,  0.1055,
         -2.4314, -1.0712,  0.0896,  1.2317]], device='cuda:0')
```

Tensor Distribution:



After this, z1 is fed in to Generator, creating the img_vector = G(z1). This vector is plotted and the corresponding generated output digit from this noise input is the following:



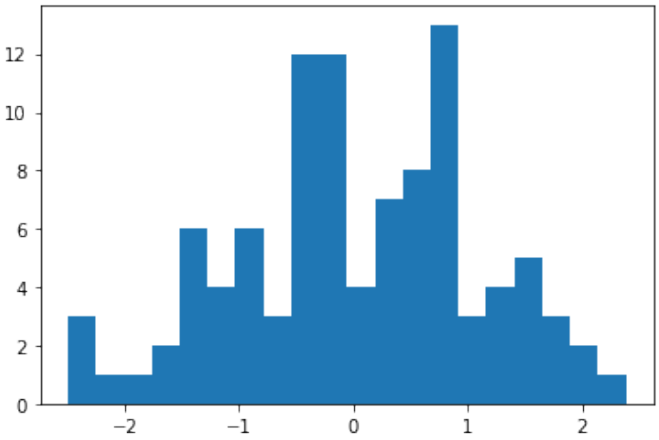
Same procedure is done for the identification of another latent variable z2 generating a second digit: it is important to select a z2 which generates a digit different from the previous, in order to provide a clear understanding of the mechanism of GANs in the next point, where we will combine the two vectors.

Saved z2 and corresponding generated digit are the following:

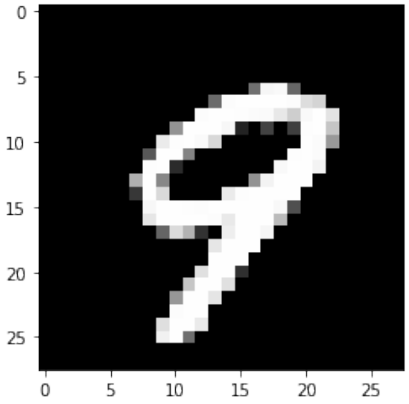
Tensor value:

```
tensor([[-7.0163e-02, -8.5930e-01, -1.2648e+00,  1.5899e+00,  8.1931e-01,
         -4.5141e-01, -1.8228e+00,  7.5103e-01, -5.3330e-01,  8.7597e-01,
         -1.3577e+00, -6.0210e-01, -1.1393e+00, -2.4051e-01, -1.3886e+00,
          2.4858e-03,  2.0045e-01,  8.0405e-01, -3.6199e-01, -9.6014e-01,
          9.3782e-01,  6.0615e-01,  1.4015e+00,  1.8939e+00, -1.2483e+00,
          1.8889e+00, -4.7994e-01,  4.9604e-01,  1.5581e-01, -1.7889e-01,
         -2.3290e+00, -2.6130e-01,  1.6281e+00, -1.5140e+00,  1.4967e+00,
         -1.7370e+00,  1.7150e+00, -2.3327e+00, -5.7657e-01,  2.0309e-01,
          4.3451e-01,  4.0825e-01, -3.2851e-01,  6.5404e-01, -7.4931e-01,
         -9.2656e-01,  5.5352e-01,  7.6904e-01,  2.3843e-01,  6.8643e-01,
         -8.5754e-01,  2.3822e+00, -2.9856e-01,  1.2795e+00, -9.1842e-01,
          6.0848e-01, -1.4246e+00,  5.6974e-01,  8.4008e-01,  7.8772e-01,
          1.9660e+00,  1.1613e+00, -3.7844e-01, -4.2614e-01, -1.4546e-01,
          7.1293e-01,  7.4635e-01, -9.6265e-02,  8.3347e-01,  1.1418e-01,
          6.6442e-01,  1.9599e+00, -1.3352e+00, -9.7790e-01,  1.6357e+00,
         -1.0654e-01, -4.7570e-01, -1.4148e+00, -1.5421e+00, -1.7355e-01,
         -5.1850e-01, -2.2080e-01, -3.8337e-01,  1.1680e+00, -2.4949e+00,
          3.4977e-01,  8.6314e-01, -2.7845e-01, -1.1356e+00, -4.6073e-01,
         -2.1300e+00, -3.1214e-01,  1.1441e+00,  1.5899e+00,  1.3054e+00,
          7.4985e-01,  2.0061e-01, -5.8368e-02,  3.8741e-01,  1.5408e-01]],
       device='cuda:0')
```

Tensor Distribution:

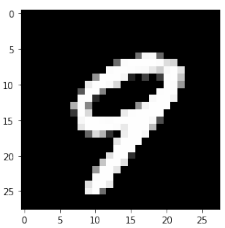
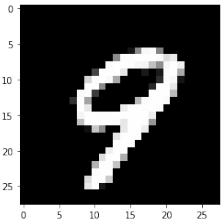
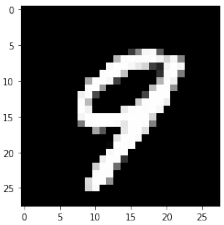
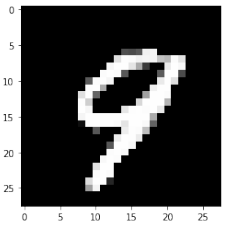
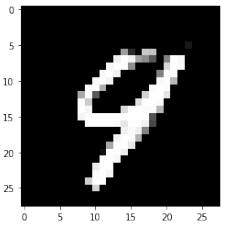
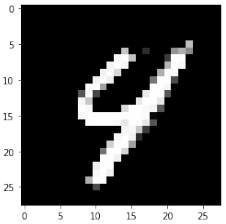
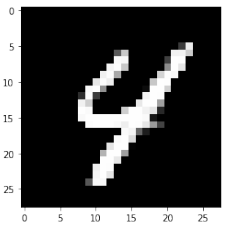
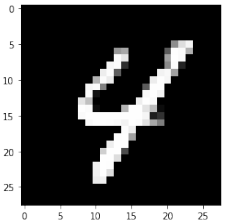


Generated Digit:

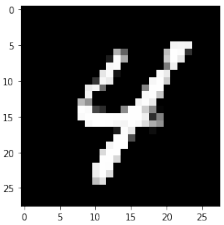
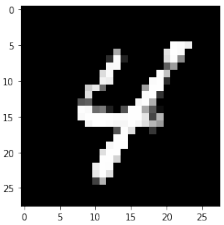
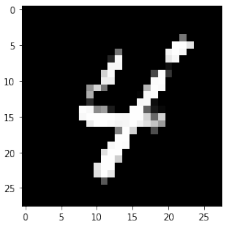


As you can notice, z1 and z2 have been saved and it is possible to overwritten this values running the cells. This has been done because otherwise everytime you run the code the resulting z = a*z1 + (1-a)*z2 would always change, and since generated digits are not always so good, this might lead to not clear results.

In the following cell a for loop is used to generate the digits with a change of a coefficient, from a = 0 to 1 with a step of 0.1 as requested by specification. In this way, eleven digits will be plot in order to understand the mechanism of digit generation. Obtained digits are, from a = 0 to a =1, the following:

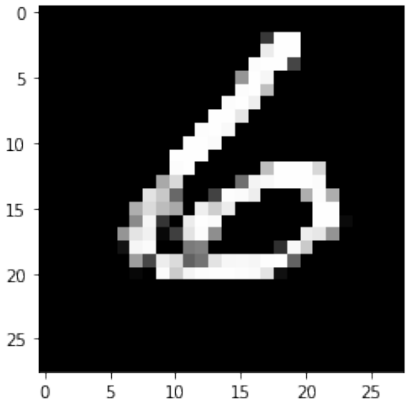









You can notice that as expected the first digit is equal to the one generated by vector z2, since the weight of digit generated by z1 is = 0, viceversa the last digit is equal to the one generated by vector z1 because in this case is = 0 the weight for digit generated by z2. For all the intermediate values between this range, you can see that the digits resembles both a 9 and a 4. This is the

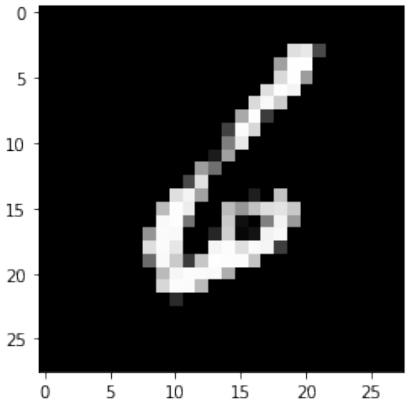
mechanism of digit generation: creating an intermediate value of z between two values that generates two different digits, resulting generated digit is an intermediate one between them!

Just as an additional part, a third variable z3 has been created! Corresponding digit is the following:

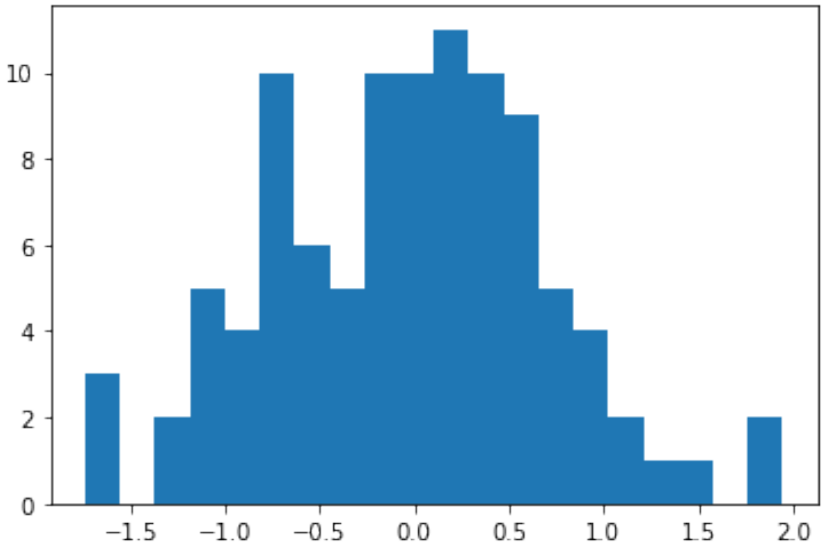


and another combination between z1, z2, z3 is created as a vector

z = a*z1 + |0.5-a|*z2 + (1-a)*z3 obtaining a similar representation to the series reported above (entire representation can be seen in the Notebook). However, I want to report the image for a = 0.5, which corresponds to an equal combination of the three of them:



Of course, this even if this digit resembles a ‘6’, this is not well defined, but it is interesting to notice that the histogram representing the distribution of z for a = 0.5 does not have a very clear peak, but instead has different components where values is high. This is because the vector is an equal average between the three distributions:


10
8
6
4
2
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

PART 2 – VISUALIZE HOW EACH DIGIT OCCUPIES THE LEARNED LATENT SPACE

In part two of the assignment the goal is to visualize how each digit occupies the learned latent space. To do so, a noise vector z is given as input to the Generator. Resulting generated digits are then fed into a previously created Neural Network. In this case, I decided to use a Three Fully Connected Layer Neural Network created in Assignment 1, with Batch Normalization and SGD optimizer.

This process of classifying the created digit is repeat for 1000 and 10000 times, in order to create 1000 and 10000 different digits.

Once the created digits have been classified, it is possible to visualize the map that relates input z to the created digit, that is exactly our request.

To obtain a 2D visualization of the occupied latent space (the space where is possible to visualize how generated output changes in function of z) a dimensionality reduction method is required. T-SNE creates a reduced feature space where similar samples are modeled by nearby points and dissimilar samples are modeled by distant points with high probability. Every generated digit is created starting from a noise z vector of 100 random elements. In principle, to visualize the latent space are required 100 dimensions, one for each element of the vector. Of course this is not possible, and these 100 dimensions are “condensed” using t-SNE (which substantially provides a singular values decomposition) into only 2 dimensions. In this way, it is possible to plot the latent space and similar digits (digits with the same label) should be clustered according to close value of z.

For reasons that will be explained later, after results presentation, also another 2-dimensional space visualization has been reported.

In fact, every digit is composed by an image generated by 784 values (each one corresponding to a pixel value) and this values of course depends on the input noise vector z to the Generator. A mapping between generated images and corresponding classification has also been created. Even in this case, to have a 2D visualization we need to convert the 784 values into 2 dimensions.

This is done with the already implemented library sklearn.manifold TSNE, a function that efficiently reduces the size of an input to specified dimensions.

CODE

In the code, first our network for digit classification is created, initialized and trained. Then the function evaluate_model() is created, in order to be called as many times as required by the for loop where this function is called. This function receives an input (whose dimensions must be coherent with the network, in this case 784), computes the outputs as model(inputs). Every output will have some properties and prediction, which in this case is also the label for the generated digits since we can only rely on the judgment of the net. The identified prediction is stored in the array classes, which at the end of the algorithm will have 1000 or 10000 elements.

After the definition of this function, the main part of the code begins: max_iter is the variable to modify in order to have 1000 or 10000 generated digits.

A for loop that iterates as many times as the specified value of max_iter is used: at every iteration, latent vector z (1x100) is initialized as a Random vector with Gaussian Distribution.

Generated image (img_vec) = G(z) is obtained and this image is evaluated with the previously described evaluate_model() function. This function will classify the image and the label will be stored in the classes array later to be used. After that, current z vector value is stored in z_vec variable and same is done for the current image value, that is stored into b vector.

At the end of this cell, we will have a vector classes to store the labels, vector z_vec to store the z vectors and vector b to store the images.

To use these vectors, it is necessary to convert them into arrays. Since both z and b are originally tensors of dimension 3 (1 dimension for the total number of elements, two dimensions for the element array), they are converted into 2D arrays.

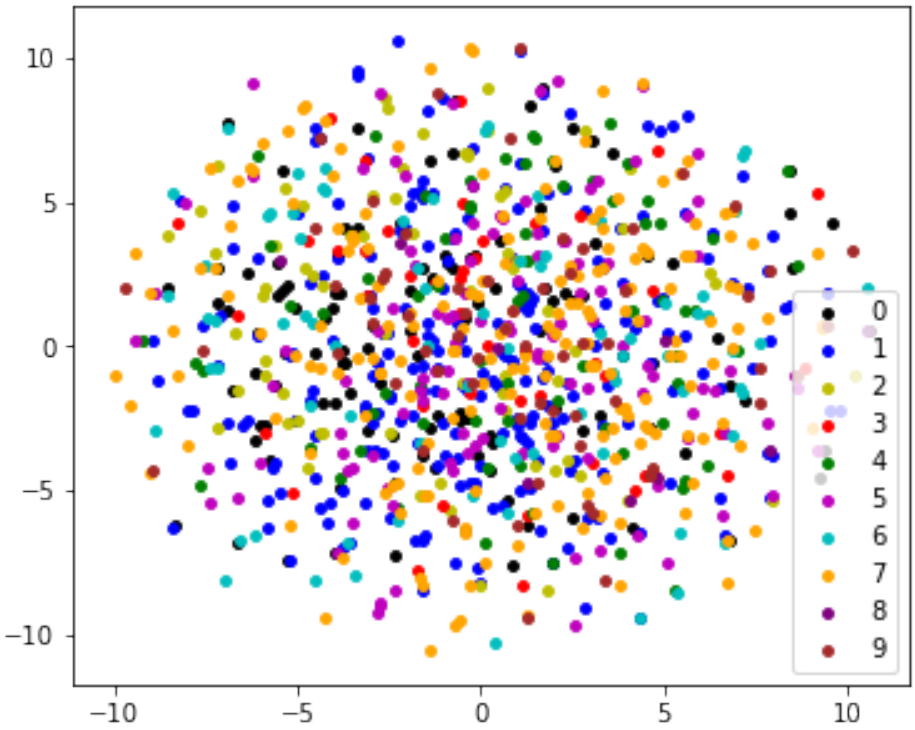
At this point, z_vec (max_iter,100) and b (max_iter,784) are passed to TSNE function, that returns as output two corresponding vectors X_z_2d, X_b_2d.

At the end, two cells are used to plot obtained vectors. In these cells firstly the range of the target_ids is defined, together with targets names that corresponds to digit from 0, 1, .. to 9. Then with 3 nested loops every point defined by the 2 dimensions of X_2d is plotted according to the label that has been given to it with the previous classification.

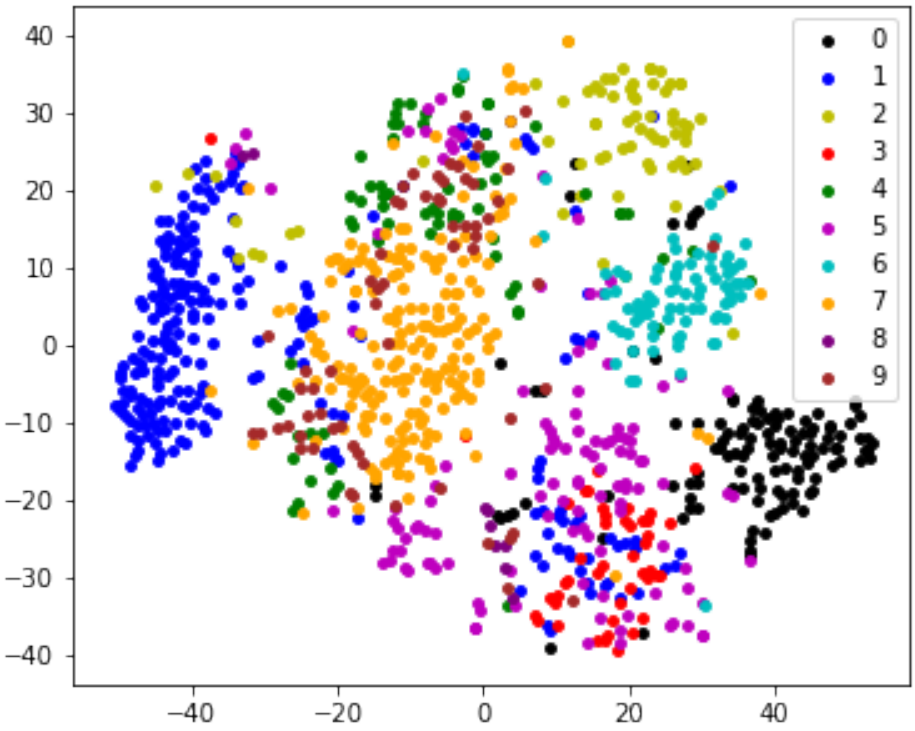
RESULTS

From the 1000 z's noise vectors, the two following plots have been obtained:

1) Z's Latent Space

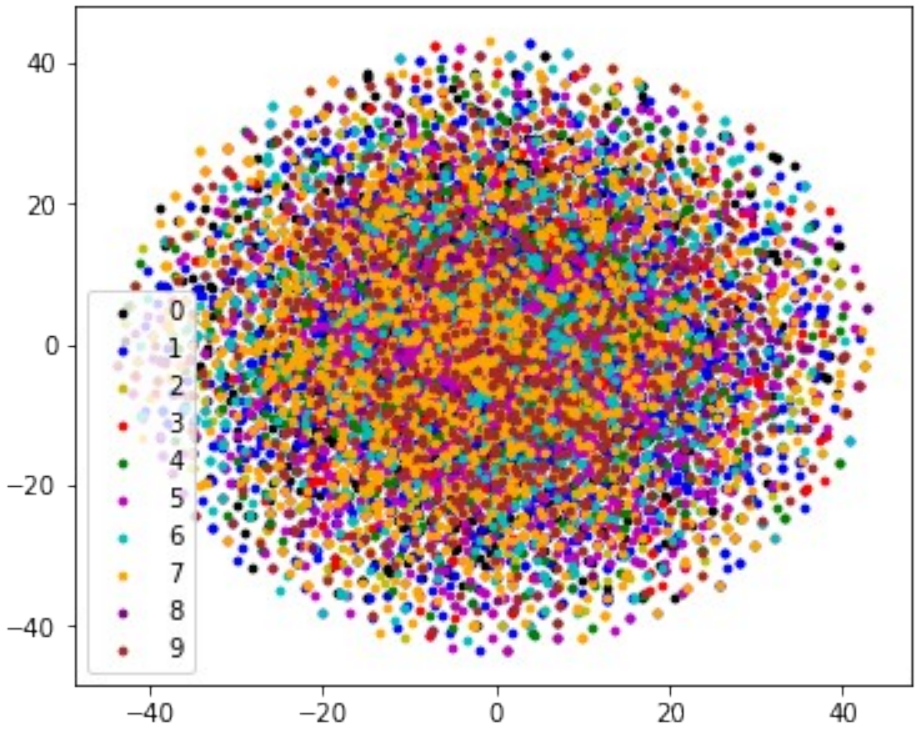


2) Image Data Space

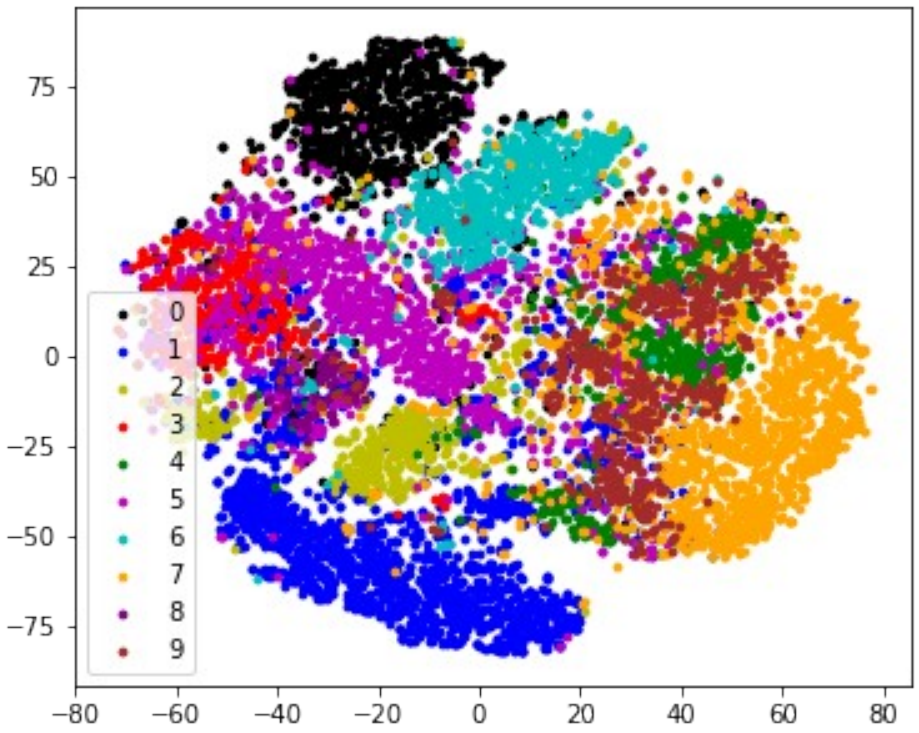


From the 10000 z’s noise vectors, the two following plots have been obtained:

1) Z’s Latent Space



2) Image Data Space



First, we should notice that z's space is not divided into clusters, but it is gaussianly distributed from the origin towards the edges. According to what stated before, this result should not be correct because t-SNE should be able to cluster data with the same values. It is obviously difficult to have two similar z's vectors, however since t-SNE extracts the predominant characteristics, it should be able to locate points generating the same digits close to each other.

This is not happening in our case: in the papers that I read, this fact is motivated explaining that t-SNE is not able to correctly extract features from Gaussian generated vectors. In a certain sense, the elements of the randomly generated z's vectors are "too random" to let t-SNE find a correlation between them!

On the other hand, you can see that clustering has been correctly done with the Image Data Space (and this is the reason why also this space has been reported): it is true that the 784 elements of a generated image are coming from the same random z's, but in this case the correlation between two similar digits can be more easily extracted by the t-SNE because the 784 elements will be similar, since the Generator Network has already adjust these values in order to produce a "good" digit from 0,1, .. 9.

Taking into considerations how each digit occupies the latent space, this is very difficult to say for the reason explained above, but both from 1000 and 10000 elements we can notice that some digits occupy more space with respect to others: for example, the digits '1' and '7' are absolutely the ones that occupy more space and if you think about how GANs works, this is reasonable: it is more easy to fool the Discriminator with a simple digit like '1' or '7'! On the other hand, it is very difficult for the Generator to fool D with a more complex digit like '8', and as you can see the occupied space by '8' is very small (color is similar to the magenta of 5, but it is purple).

Notice also that clusters corresponding to same digit have different positions in the 1000 ad 10000 plots! This is reasonable, since the assigned coordinates are fictitious coordinate generated by t-SNE, and actually the coordinate of a cluster is explained in relationship with the coordinates of another cluster: the further a cluster is from another cluster, the more different the digits of these two clusters are: take a look for example at cluster for digit '1' and '0': they are always on the opposite side of the plot, even if their absolute position within the plot change. Similarly, '1' and '7' clusters are always close to each other because they share more common features.

In addition, a latent space for random uniformly distributed noise vectors is plotted at the end of the Jupiter notebook.