Report Neural Network project
Bidirectional Generative Adversarial Networks

Fabrizio Casadei - Matricola: 1952529
Edoardo Papa - Matricola: 1962169

SAPIENZA
UNIVERSITÀ DI ROMA

April 22, 2021

Chapter 1

Overview

This Neural network project has been chosen to study and then implement, a machine learning framework called **Bidirectional Generative Adversarial Networks**, also simply known as **BIGANs**.
This method belongs to the family of frameworks used in *computer vision*, whose deal is to achieve a high level of understanding of digital images or videos, through the acquisition, processing and the analyzation of them.
More in the specific, we are talking about *Generative models*, which model data in a generative fashion, in order to construct new data thanks to the approximation of probability distribution of the information.
The aim of these models is, given a certain target function $f : X \to Y$, through a learning process on a certain dataset D, estimate the probability $P(X|Y)$ and $P(Y)$.
This branch is in a continuous upgrading, which in the past years, is come out with different new techniques and architectures, like the one, object of this report.

The goal for the implemented system is to generate realistic images based on the learned dataset, with an auto-valuation ability for improvements.
Moreover the image will be generated from an encoded set of data, called *latent space* (that could be chosen randomly, for the generation of new data), one specific component of the system will be in charge of being qualified in the conversion of an image to that representation.

1.1 The origin

Thanks to the ability of recognize pattern information present in the structure of the data itself, new *unsupervised* tasks in computer vision have been introduced, expanding the pipeline of work, from *supervised learning* using *Deep convolution networks* to *Unsupervised*, with competitive results in visual feature learning.

The *BIGAN* is a direct variation of the classical *Generative Adversarial Network* (Ian Goodfellow, 2014), able to learn generative models of arbitrarily complex data distribution. This unsupervised model summarizes the distribution of input variables, to create and generate new coherent instances in reference to the Dataset.
Why is it called Adversarial?
This deep-learning generative architecture, it's centred on two sub-models, the **Generator** and the **Discriminator**. These can be seen as two entities that try to overcome on the other, in fact, we can make this brief summary about their intentions:

- the *generator*, taken an arbitrary input vector, generate the most plausible images concerning the problem domain, trying to make it enough good to cheat the *Discriminator*.
- the *discriminator*, has the duty of understanding which of the images that arrive in input are either real (directly sampled from the dataset), or fake (image by the generator).

follows a general scheme of the *GAN*.

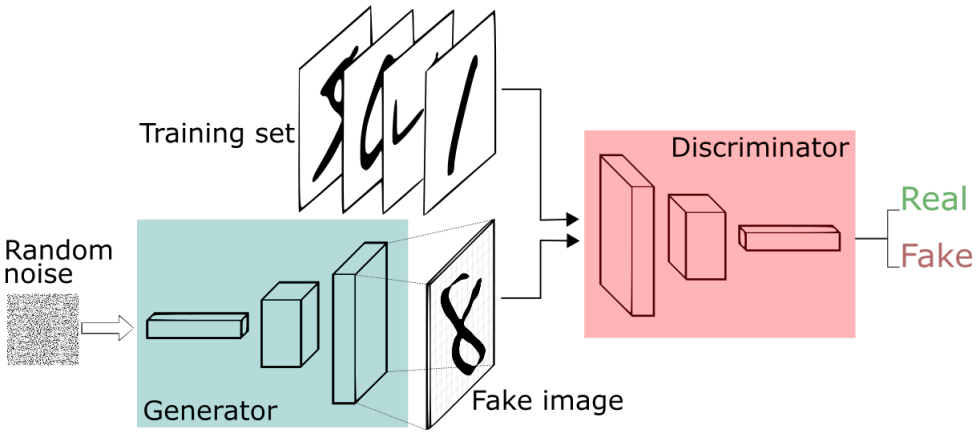


Figure 1.1: "The *GAN* framework"

The *BIGAN* twists this structure, adding another component called **Encoder**, whose task is to map a certain instance data to its intrinsic latent representation. In this case, the *discriminator* discriminates not only the generator data flow but also the encoder representation.
More details about these components will be expressed in next chapters.

1.2 The Dataset

The dataset chosen for the develop of the whole project's system has been the *MNIST*, a very well known, and frequently used dataset in machine learning applications.
The *MNIST* is made up of images that represent handwritten digits, and actually (could be upgraded) it's composed of 60.000 instances for the training set, and 10.000 elements for the *test set*.
Each image has a single colour channel, therefore, is a *grey-scale* set of images, and the resolution is 28×28. the digits fall in the range from 0 up to 9, and sometimes it's quite ambiguous even for a person to understand the correct label.

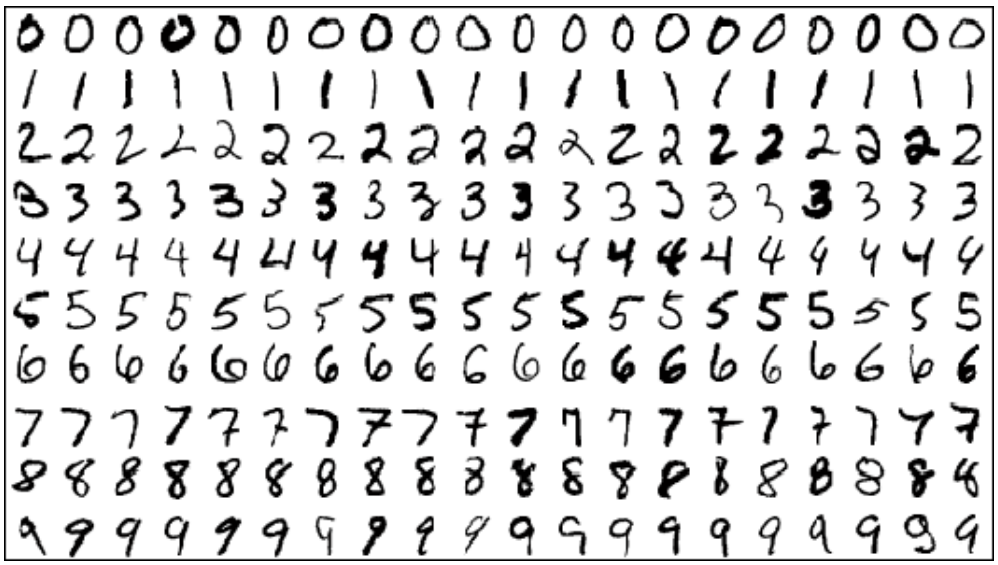


Figure 1.2: "The *MNIST* dataset"

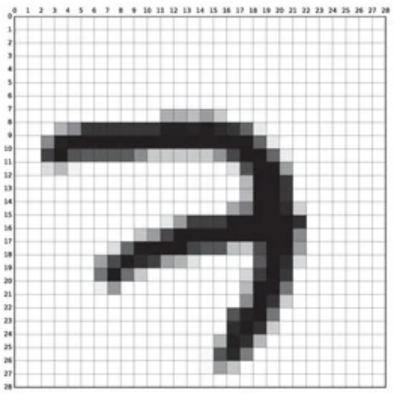


Figure 1.3: "An ambiguous instance, 3 or 7?"

Moreover, in the final chapter we have tested the *BIGAN* on another dataset, so called *SVHN*. In this case, we are talking about real-world colour images representing Street View House Numbers. The dataset is composed of more than 600.000 instances, each one of dimension 32×32 (cropped digits version).
Below is shown a cluster of samples taken from the dataset.

Figure 1.4: "The *SVHN* dataset"

Chapter 2

Background

In this chapter, we are going to introduce the theoretical notions useful for the comprehension of the implementation section.
First, we focus on the main part of the project, how to learn and structure the *BIGAN*, and then we introduce the classifier model used to evaluate its performance.

2.1 BIGAN

In the overview we have started the discussion about the *BIGAN*, which will be concluded here, analyzing the components and their connections.
We have introduced that *BIGAN* in addition to the *generator* G and the *discriminator* D, standard parts of the *GAN framework*, introduce the *encoder* E, a model in charge to produce a mapping between a certain data instance x and the corresponding latent representation z.
The discriminator D, discriminates not only in data space but together in data and latent space, hence, tuples $(x, E(x))$ against $(z, G(z))$, where the latent component is either an encoder output $E(x)$ or a generator input z. The discriminator is therefore modified to take input from the latent space, predicting $P(Y|x, z)$ where $Y = 1$ if x is real and $Y = 0$ if x is generated.

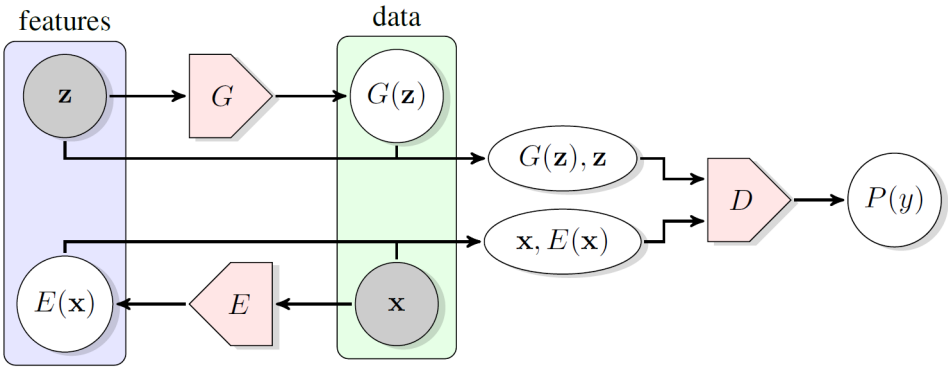


Figure 2.1: "Structure of the Bidirectional Generative Adversarial Networks"

The *encoder* E has to learn how invert the *generator* G. As shown in the image the two modules cannot directly communicate, the encoder never sees generator outputs, there-

fore, both $E(G(z))$ and $G(E(x))$ are never computed in the learning process. E and G must learn to invert one another in order to fool the discriminator. With this condition, a latent representation z may be thought as a "label" for x, achieved without the need of a supervision.
The *BIGAN target function*, used to train the model is defined as *minmax* objective:

$$\min_{G,E} \max_{D} V(D, E, G)$$

$$V(D,E,G) := \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{X}}}\big[\underbrace{\mathbb{E}_{\mathbf{z}\sim p_E(\cdot|\mathbf{x})}\left[\log D(\mathbf{x},\mathbf{z})\right]}_{\log D(\mathbf{x},E(\mathbf{x}))}\big] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{Z}}}\big[\underbrace{\mathbb{E}_{\mathbf{x}\sim p_G(\cdot|\mathbf{z})}\left[\log\left(1-D(\mathbf{x},\mathbf{z})\right)\right]}_{\log(1-D(G(\mathbf{z}),\mathbf{z}))}\big]$$

Figure 2.2: Objective function for G,E & D

Therefore, G and E will try to maximize it, in order to fool D, which want to minimize this error. It's used a monotonic logarithmic function to simplify the computation, and the term $(1 - D(G(z), z))$ represents the wrong prediction of the discriminator on the image generated.
The global minimum for the objective function of G and E is reached if and only if $P_{EX} = P_{GZ}$, which means having the same probability for a certain instance x to be a true image and a fake image (both 0.5).
In order to fool a perfect discriminator, E and G must satisfy this condition:

$$\mathbf{x} \in \hat{\Omega}_{\mathbf{X}} \wedge E(\mathbf{x}) = \mathbf{z} \qquad\qquad \mathbf{z} \in \hat{\Omega}_{\mathbf{Z}} \wedge G(\mathbf{z}) = \mathbf{x}$$

With $\hat{\Omega}_X$ and $\hat{\Omega}_Z$ represent the data samples.
In addition we can extend the concept of objective function doing a parallel with the *autoencoder* framework. The *encoder* and *generator* objective given an optimal *discriminator* $C(E;G) := max_D V(D;E;G)$ can be rewritten as l_0 a classical *autoencoder* loss function. this lead to the following objective.

$$\begin{aligned} C(E,G) = \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{X}}}\left[\mathbf{1}_{\left[E(\mathbf{x})\in\hat{\Omega}_{\mathbf{Z}}\wedge G(E(\mathbf{x}))=\mathbf{x}\right]}\log f_{EG}(\mathbf{x},E(\mathbf{x}))\right] + \\ \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{Z}}}\left[\mathbf{1}_{\left[G(\mathbf{z})\in\hat{\Omega}_{\mathbf{X}}\wedge E(G(\mathbf{z}))=\mathbf{z}\right]}\log\left(1-f_{EG}(G(\mathbf{z}),\mathbf{z})\right)\right] \end{aligned}$$

With $\mathbf{1}$ the *indicator function* or a *characteristic function*, defined on a set X that indicates membership of an element in a subset A of X, having the value 1 for all elements of A and the value of 0 for all elements of X not in A.

We finally introduce an "inverse" objective Λ, which uses the same elements of the old objective, and provides a stronger gradient signal to G and E.

$$\Lambda(D,G,E) = \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{X}}}\big[\underbrace{\mathbb{E}_{\mathbf{z}\sim p_E(\cdot|\mathbf{x})}\left[\log\left(1-D(\mathbf{x},\mathbf{z})\right)\right]}_{\log(1-D(\mathbf{x},E(\mathbf{x})))}\big] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{Z}}}\big[\underbrace{\mathbb{E}_{\mathbf{x}\sim p_G(\cdot|\mathbf{z})}\left[\log D(\mathbf{x},\mathbf{z})\right]}_{\log D(G(\mathbf{z}),\mathbf{z})}\big]$$

Figure 2.3: Objective function G & E

2.2 K-NN

In this section, we explain a useful classifier, extremely easy to characterized, called **K-NN** or **K-nearest neighbours**, that will be used to evaluate the performance of the *BIGAN*.
This model belongs to a type of *ML* models called *instance-based learning* or *lazy learning*. this group of techniques allow to define a model without specify any kind of parameters, in fact to make a classification or a regression task, are used all the instance of a certain dataset.
The performance of K-NN strictly depends on the K hyperparameter, which defines how many neighbours are taken in account to make a prediction, and the *distance function*, that evaluates how much two instances are distant in the dataset.
a very common example for the distance function is the *euclidean distance* whose formula is specified below.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.1}$$

With n the number of attributes related to a certain instance.

K-NN requires the whole training set to be stored, leading to expensive computation if the dataset is large. Another important aspect is that increasing k brings to smoother classification regions. let's describe the algorithm steps to classify instances, the aspect in which we are interested.

1. Find the K nearest neighbours N_k of the new instance x in a dataset D
2. Assign to x the most common label among the majority of neighbours.
   the likelihood of a certain class c for a new instance x is

$$P(c|x, D, K) = \frac{1}{K} \sum_{i \in N_k(x,D)} Identity(y_i = c) \tag{2.2}$$

   we compute this probability for all the classes, and we take the greater

$$c* = argmax_{c_i \in C} \;\; P(c_i|x, D, K) \tag{2.3}$$

Chapter 3

Implementation

With the theoretical notions described in the precedent chapter, it's now possible to treat more clearly the implementations aspects of this project.
We can fragment this crucial section into three main topics: the models, the learning step and finally the implementation behind evaluations.

3.1 Models

As explained the *BIGAN* it's mainly composed by three kind of components: the *Descriminator*, the *Generator* and the *Encoder*.
All these take as input a batch of elements from the dataset equal to 256, wherein the case of G, each element is a latent representation with shape $256 \times 1 \times 1$ instead for E is a $32 \times 32 \times 1$ image, while for D, both the output of G and E are taken as input with the same shape of images and latent representations

The **Generator** is made up of six layers which are the following:

Layers	Operations	Activation functions
1	2D Transpose convolution and batch normalization	Relu
2	2D Transpose convolution and batch normalization	Relu
3	2D Transpose convolution and batch normalization	Relu
4	2D Transpose convolution and batch normalization	Relu
5	2D Transpose convolution and batch normalization	Relu
6	2D convolution	Hyperbolic tangent

The *2D transpose convolution* has been used to upsample the image data that we want to extract, in addition, has been proven the real effectiveness in the *Generative models* like the *GAN*. The Batch normalization operation takes the previous results and normalizes them, preventing data from being too large cause the instability of network performance. For the hidden layers has been used one of the classical activation function to introduce the non-linearity which is the *Relu*, very useful to exploit the learning process and avoid any kind of saturation.
As last layer, we have used a convolutional layer to reach the desired output and the

Hyperbolic tangent activation function in order of producing pixel values in the range between -1 and 1, which is a very common practice for image generator.
Why using this pixel intensity representation? The classical 8-bit format in the range of [0,255] could be a real problem considering the small weights used between layers.
The best way of acting is to distribute pixel with zero-centred values in the range[-1,1].
This kind of reasoning comes up from the application of standard *Gaussian distribution* zero-centred with no fixed standard deviation.
Indeed, the image can be simply rescaled in a more canonical representation in the range of [0,1].

Now we pass in the analysis of the **Encoder**. Follows a table with its structure.

Layers	Operations	Activation functions
1	2D convolution and batch normalization	Leaky Relu
2	2D convolution and batch normalization	Leaky Relu
3	2D convolution and batch normalization	Leaky Relu
4	2D convolution and batch normalization	Leaky Relu
5	2D convolution and batch normalization	Leaky Relu
6	2D convolution and batch normalization	Leaky Relu
7	2D convolution	Linear

In this case, we have 7 layers, composed of 2D convolutions and batch normalization operations. Using the convolutional layers we are increasing the feature maps flatting the image information using correct stride values.
As activation function has been used the *Leaky Relu* rather than the *Relu*, this function has a small slope for negative values, and as result, negative values are not mapped to zero. This solves the problem of dying values or dead neurons from the normal *rectified linear unit*, involving a speed up in the training in different cases. Here it's absolutely an appropriate choice, allowing value less than zero. In the final layer, we have just used a Linear activation function, concerning the regression task.
One very important point that has not been highlighted in the above table, is the usage of a very effective trick which comes from the *Variational Autoencoder*, called *Reparametrization*.
The main idea which has come up from years of research is that encoders lead to a better latent representation when the results are sampled from a learned distribution. Moreover, this trick allows the reduction of variance in the gradients. With this procedure instantaneously arises a problem when we think about directly sampling from a defined Gaussian distribution. This problem is given by the fact of having a non-differentiable process, as the random sampling, hence, the backpropagation cannot be computed.
This can be solved approximating a sampling differentiable process, let's see how.
Starting from the flatten features extracted, we use the first half to characterize the mean, and the second half to represent the logarithmic standard deviation. The standard deviation is utilized to generate a certain stochastic noise in order to finally produce the final distribution from which sample the latent space. Follows the relative formula:

$$z = \mu + \sigma \odot \epsilon \tag{3.1}$$

Using this sampling process it's now possible to learn the network through the backpropagation operation.

The last missing model it's now the **Discriminator**, as usual, let's do a recap on its organization. In this case, we need more than one table for doing this, that's because the Discriminator take both images x and latent variables z as input, and defines two initial separated layers structure for them. After having propagated each input through the correct path, the results are combined to flow into a final series of layers.
We start presenting the x path.

Layers	Operations	Activation functions
1	2D convolution	Leaky Relu
2	2D dropout	/
3	2D convolution and batch normalization	Leaky Relu
4	2D dropout	/
5	2D convolution and batch normalization	Leaky Relu
6	2D dropout	/
7	2D convolution and batch normalization	Leaky Relu
8	2D dropout	/
9	2D convolution and batch normalization	Leaky Relu
10	2D dropout	/

Continue with the layers block of z.

Layers	Operations	Activation functions
1	2D convolution	Leaky Relu
2	2D dropout	/
3	2D convolution	Leaky Relu
4	2D dropout	/

In the end, the common final section.

Layers	Operations	Activation functions
1	2D convolution	Leaky Relu
2	dropout	/
3	2D convolution	Leaky Relu
4	dropout	/
5	2D convolution	Sigmoid

This 2-ways structure allows to learn with different weights and make separated inferences on the two kinds of data, in order of having an independent evaluation and extraction of information. all these latent data are then merged together and elaborated in the final section to achieve correct discriminations.
We have already described several components present here, however, two new argumentations are still needed.
Let's start with the insertion of 2D dropout and dropout. These layers temporally and

randomly remove network units with a certain probability at each step. The first is used when we have multidimensional data like an image, while the second has been used after the combination of the 2 paths for x and z, which leads to a flattened combination of them. The dropout technique permits the reduction of the *overfitting* over the training, hence, it belongs to the *regularization* methods. One final note is a motivation under the usage of the *sigomoid* output unit. The task of a *discriminator* is to understand whether an image is real or fake, in this case, an appropriate approach is considering the whole process a binary classification task that motivates the choice done for the activation function.

3.2 Learning

The learning process follows in A faithful way all the aspects described in precedence, adding some implementation details, that must be discussed.
The first important thing to explicit is the number of epochs used, this aspect has been variated several times, but at least the models has been learned for 200 epochs, a good trade-off between reliable results and the time needed for the process.
Before defining the learning loop, some operations are been performed, are been obviously created the models from the class described in the previous section, and then initialized their weights according to the type of each layer (convolutional, linear, and batch-normalizing) using a Normal distribution with specific values for the *mean* and *standard deviation*. As last preliminary step, has been defined two different optimizers, both of the type *Adam*, one for the G & E, and a separate one for the D.
This because we use two different error functions, the one shown in figure 2.2 for D, while for G & E that in figure 2.3.

The following steps are repeated for each step in every epoch, and can be divided into two main parts:

1. **The discriminator learning**. First, the gradients of D are set to zero, in order of having a reset of partial derivatives, then a random latent variable z is defined and used to perform $G(Z)$. To compute the loss we still miss the Encoder representation, hence, from the image x (actually we are talking about a mini-batch of images, but just to simplify the explanation, we make this assumption) we obtain its latent representation $E(x)$.
   Subsequently, using all these values are computed $D(x, E(x))$ and $D(G(z), z)$. Finally we calculus the loss, backpropagating it from the output layer to the first layer of D using the optimizer, in order of performing the update of all the parameters.

2. **Encoder and Generator learning**. In this second part, after having set the gradients of G and E to zero, we use all the values already carried out to estimate $D(x, E(x))$ and $D(G(z), z)$, for achieving the loss of D. It's important to notice the fact of having now different values from D, due to the update.
   In the end, one more time we perform the backpropagation and consequent update using the optimizer of G & E.

A common important aspect in the two learning parts which has been not highlighted yet is the possibility of adding noise. In fact, we compute four different random noise, two

for the first part and another two for the second.
These variables are used to variate the input for D. Therefore, in both cases the actual Discriminator predictions are given by $D(x + \epsilon, E(x))$ and $D(G(z) + \epsilon, z)$. This practice has been proved to involve a better representation in the quality of the images given by having smoothed the probability distribution taken as input from D. A complete analysis will be carried out in the next chapter, making a comparison between result with and without it.

3.3 Evaluation

After having generated images through the *BIGAN* we need something to evaluate it. More in the details we want to implement a K-NN model to classify them. The real implementation choice of this section is which value assign to the K hyperparameter. How suggests in the paper *arXiv:1605.09782*, the 1-NN is the best choice.
Training the pre-implemented model from *sklearn* on MNIST and testing it has been reached an accuracy around the 97%.
We have also measured the time needed to perform the predictions on a test set made up of 10.000 instances(a well-known drawback of the so-called lazy algorithms), which is about 23 minutes.
Now we have all the elements to making out final evaluations and conclusion.

Chapter 4

Conclusion

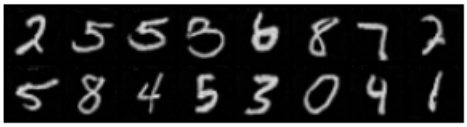
We are now presenting the final chapter of this report, whose aim is to highlight the most significant results achieved and conclude the treatment making overall considerations.

4.1 Results

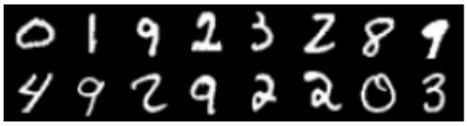
Several tests have been performed, the most significant that we are going to analyze are these two:

1. Performance with or without Noise addition.
2. Performance with or without Reparametrization trick.

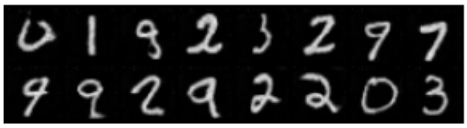
Let's begin from the graphical results achieved with the development dataset (*MNIST*), showing images generated by $G(z)$ and $G(E(x))$. The following are the results with noise and the reparametrization trick. In addition, is exhibited the loss functions over epochs.



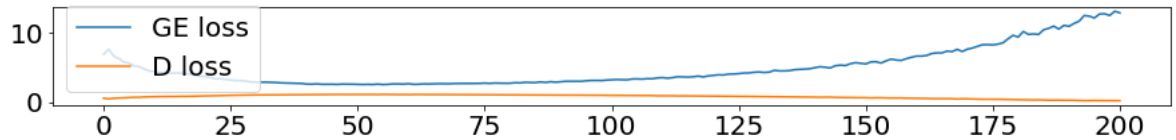
$G(z)$



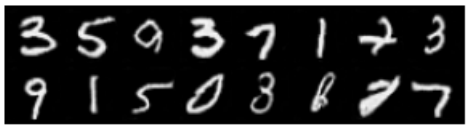
x



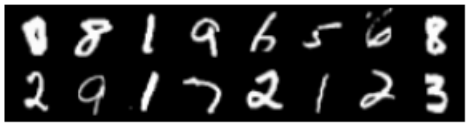
$G(E(x))$



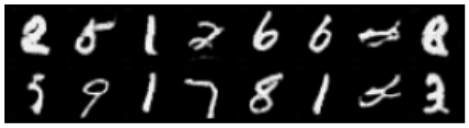
The next data represents the outcomes from the learning without the use of the noise addition (reparametrization used).



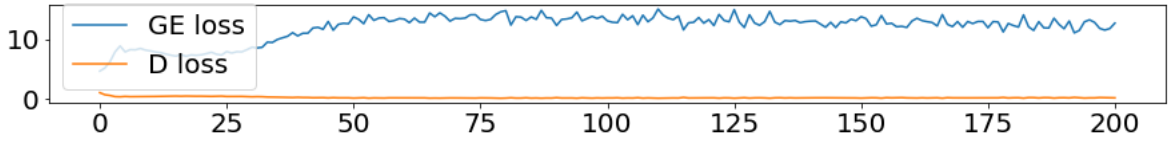
$G(z)$



x

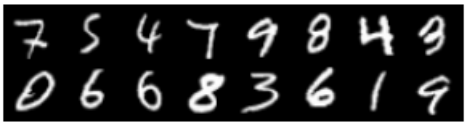


$G(E(x))$



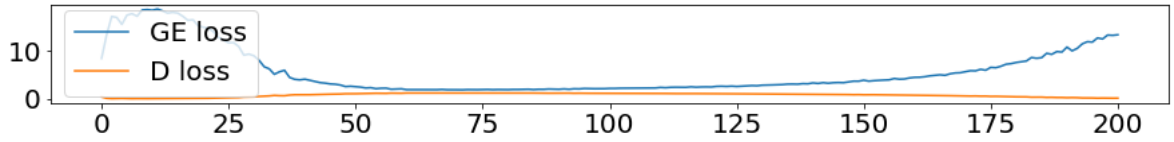
Then we have the graphics and plots from the learning without *reparametrization* (noise added).

$G(z)$



x

$G(E(x))$

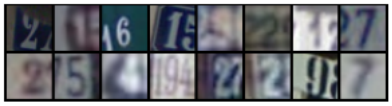


To conclude the results of these tests, we illustrate the table about the *accuracy* values (correct classifications over the whole predictions) of the three types of configurations.

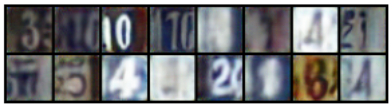
Noise enabled	Reparametrization Enabled	Accuracy (%)
Yes	Yes	75%
No	Yes	50%
Yes	No	53%

Finally we expose upshots on *SVHN* dataset.

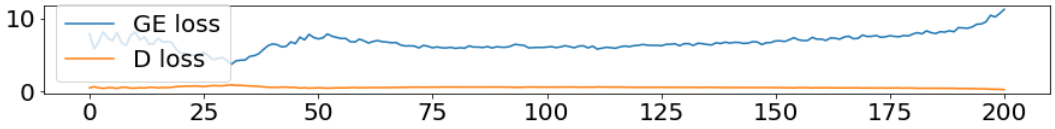
$G(z)$



x



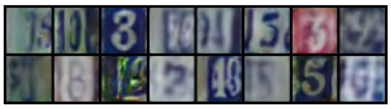
$G(E(x))$



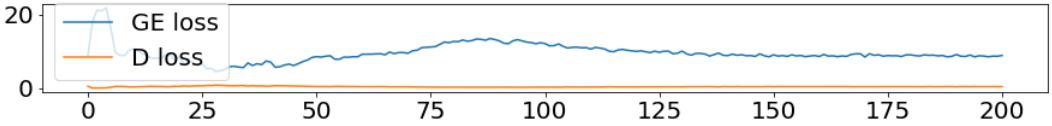
Results on *SVHN* adding noise

$G(z)$

x



$G(E(x))$



Results on *SVHN* without noise addition

4.2 Final considerations

In this last section, we are going to talk about the results achieved. As we can see, the images generated by a $G(z)$ with the noise and *reparametrization* are the best that we've obtained, indeed, most of the generated digits are undistinguished from the real ones. In any case, all the images generated by these different BiGAN models are correct reconstruction for most samples. Even the latent space extracted in that case reach a high level of accuracy, also considering the number of epochs and the space dimension (128) used.
Moreover, from the plots of the two losses, it's possible to notice the relationship between them. In fact, when the discriminator becomes more accurate, the loss of the *Encoder* and *Generator* increases as a consequence.
It's not easy to cheat D that in this case, is never reached and beaten from G & E.
From the information presented it's possible to appreciate that the noise helps us to have good accuracy and enhance the loss difference between Generator-Encoder and Discriminator. This because noise slows the D learning, which in general is much faster than the one for G & E.
For what concern results without the *reparametrization* it's possible to notice some background artefacts in the images generated. This because the latent distribution built from E it's used without sampling, leading to a slower learning in this sense. To sum up, we have done a lot of tests and here were report just a few, the most important ones. The best model we've obtained is the one that adds noise and uses the *reparametrization* trick with an accuracy of 75%.

Before concluding this report we would suggest several possible further implementations related to this project.

- A better usage of a variable noise over the epochs, in order of increasing the accuracy and the distance between D and G & E.
- Use the so called trick "Train with labels", which tells directly to the discriminator whether the images are real or fake. In this case, the *Discriminator* instead of generating a probability of x being real or fake, will now generate the probability of the class.
- Implement and tune other several *Regularization* methods like the *early stopping* or *Data augmentation*.
- Manage this project to perform good results with different and more difficult datasets, like for example the *CelabA* (very long learning session expected).

It's not always easy to achieve good results using generative models, exactly for this reason we could be very satisfied with the outcomes registered, leading to have, from a human point of view, a really hard uncertainty on finding which image is real and which is not.