Conditional Generative Adversarial Networks

Simone Campisi
s4341240@studenti.unige.it

Abstract

With this project I want to present a variant of the classical Generative Adversarial Networks (GANs), that is the conditional version, (CGANs) which allows the classical GAN to converge towards one or more desired classes.

1 Introduction

1.1 Generative Adversarial Networks (GANs)

The goal of *generative adversarial network* is to generate new data instances. This model is composed of two *'adversarial'* models:

1. *A generative model G* which produces samples
2. *A discriminative model D* which attempts to distinguish between samples generated from the generator and samples of the training set

In order to learn the generator's distribution p_g over data x, the generator builds a mapping from a prior noise distribution $p_z(z)$ to data space as $G(z; \theta_g)$. The discriminator, $D(x; \theta_d)$, outputs a single scalar representing the probability that x came from training data rather than p_g. D is trained in order to maximize the probability to classify correctly both training examples and samples coming from G. Then, simultaneously, G is trained in order to minimize $\log(1 - D(G(z)))$. D and G play the following two-player *minmax game* with value function V(G,D):

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

[2]

1.2 Conditional Generative Adversarial Networks (CGANs)

The *conditional generative adversarial networks* are an extension of GANs described in the previous paragraph. In fact the GANs models are conditioned on some extra information y, that could be any kind of auxiliary information, but in this case are just class labels. The conditioning is performed by feeding y into both discriminator and generator as additional input layer. The Fig.1 shows the structure differences between the GANs and CGANs previously described. In the generator, the input noise variable $p_z(z)$ and y are combined in a joint hidden representation. In the discriminator x and y are the inputs to the discriminative function. The objective function will be as the following:

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

[1]

1.3 Model Evaluation

Since the discriminator loss and generator loss are not very intuitive and very unstable, a better way to evaluate the quality of the cGAN is to observe and evaluate the samples generated after the completion of the training.

Preprint. Under review.

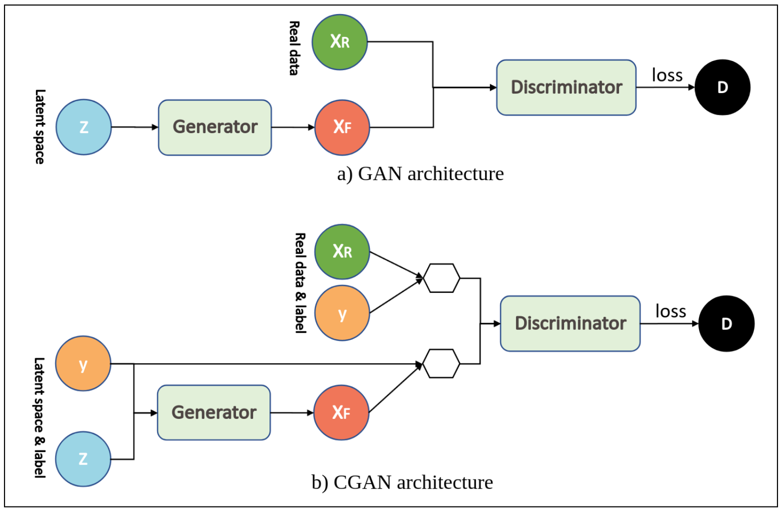


Figure 1: Differences of architecture between GANs and CGANs networks

1.3.1 The Frèchet Inception Distance (FID)

The Frechet Inception Distance (FID for short) is a metric for evaluating the quality of generated images, computing the similarity between real and generated images. The FID score uses the pre-trained *inception v3 model,* which is a convolutional neural network that is 48 layers deep. The pre-trained network can classify images into 1000 object categories. In this model the coding layer, the last pooling layer before the output (classification of the images) is used to capture *computer vision specific features*, obtaining a features vector of 2048 features for both real and generated images. Then, the distance between two distribution is computed with the *Frèchet distance*. A lower FID indicates better quality of the images, instead an high score indicates a lower quality. Given two multivariate Gaussian distribution $N(\mu_x, \Sigma_x)$ and $N(\mu_g, \Sigma_g)$, the Frèchet Distance (FD) is defined as:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + \mathrm{Tr}(\Sigma_x + \Sigma_g - 2\sqrt{(\Sigma_x \Sigma_g)})$$

Where

$$\mu_x = \frac{1}{N}\sum_{i=0}^{N} f(x^{(i)}),\ \Sigma_x = \frac{1}{N-1}\sum_{i=0}^{N}(f(x^{(i)}) - \mu_x)(f(x^{(i)}) - \mu_x)^T$$
$$\mu_g = \frac{1}{N}\sum_{i=0}^{N} f(x^{(i)}),\ \Sigma_g = \frac{1}{N-1}\sum_{i=0}^{N}(f(x^{(i)}) - \mu_g)(f(x^{(i)}) - \mu_g)^T$$

In which f is the *Inception embedding function*.

Σ_x and Σ_g are the covariance matrix for the real and generated feature vector, and Tr (trace linear algebra operation) is the sum of the element along the main diagonal of the square matrix.

1.3.2 The Frèchet Joint Distance (FJD)

FID, however is a metric for *unconditional image generation*, and is focused on some properties like visual quality and sample diversity, which don't fully encapsulate all the different phenomena in the conditional image generation.

In the conditional generation there are important properties to take into account, like *visual quality*, *conditional consistency* (verify if the generated images respect the conditioning), *intra-conditioning diversity* (sample diversity per conditioning).

The Fréchet Joint Distance (FJD), which is able to assess image quality, conditional consistency, and intra-conditioning diversity, computes the Frèchet Distance between two Gaussians defined over the *joint image-conditioning embedding space*. In particular given an embedding function f, a conditioning embedding function h, a conditioning embedding scaling factor α and a merging function g that combines the image embedding with the conditioning embedding into a joint one, the estimation of the parameters $\mu_x, \mu_g, \Sigma_x, \Sigma_g$ becomes:

$$\mu_x = \frac{1}{N}\sum_{i=0}^{N} g(f(x^{(i)}), \alpha h(y^{(i)}),\ \mu_g = \frac{1}{N}\sum_{i=0}^{N} g(f(x^{(i)}), \alpha h(y^{(i)})$$

$$\Sigma_x = \frac{1}{N-1}\sum_{i=0}^{N}(g(f(x^{(i)}), \alpha h(y^{(i)}) - \mu_x)(g(f(x^{(i)}), \alpha h(y^{(i)}) - \mu_x)^T$$

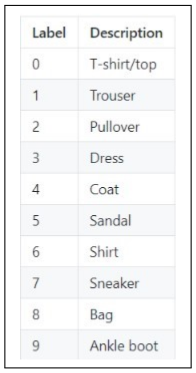
$$\Sigma_g = \frac{1}{N-1}\sum_{i=0}^{N}(g(f(x^{(i)}), \alpha h(y^{(i)}) - \mu_g)(g(f(x^{(i)}), \alpha h(y^{(i)}) - \mu_g)^T$$

in which y is the condition, and α is the *conditioning embedding scaling factor*, that indicates how much is important the conditioning component respect to the image component. In fact when $\alpha = 0$, the conditioning component is ignored and FJD is equivalent to FID. To equally weight the image and the conditioning component, α can be setted to the ratio between the average L_2 norm of the image embedding and the conditioning embeddings. Since, in this project, the condition are class labels, the *conditioning embedding function, h*, is the one-hot encoding. Instead, the *merging function, g* combines the image embedding and the conditioning embedding. In this project the *concatenation* is chosen as merging function.

Computing the FD over the joint image-conditioning distribution, we are able to simultaneously assess image quality, conditional consistency, and intra-conditioning diversity,

1.4 Dataset

The model has been trained the *"Fashion-MNIST"* dataset provided by keras, which is a dataset of Zalando's article images, and costista of a training set of 60,000 examples and a test set of 10,000 examples, where the examples are 28x28 grayscale images and are associated to 10 classes (number from 0 to 9).



Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Figure 2: Class labels for fashion-MNIST dataset

2 Experimental Results

As mentioned in the section 1.2 , the cGAN model (Fig.4c) is composed by a generator, built as in Fig.4a, a discriminator, built as in Fig.4b. For all the following experiments the model has been conditioned with class labels from 0 to 9 repeated 5 times. So, the generated images from the cGAN are not random but correspond to the class labels taken as second input. In this way the output are 5 rows of images, where in each row there are all the classes in order to show the differences between the same classes. The Fig.3 shows how the class labels passed to the model affects the output. In fact, the model has taken as second input, in order, 5 zeros , 5 ones and so on, encoded in one-hot vectors.

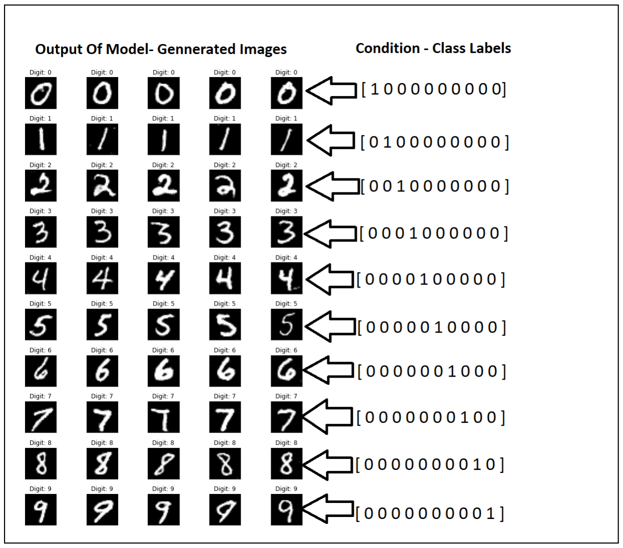


Figure 3: How the class labels, encoded in one-hot vectors, conditions the output of the model. The result has been obtained using 64 of batch size and 20.000 iterations

2.1 Effects of changing the batch size and number of iterations

As first experiment, the cGAN model has been trained with 'Fashion-MNIST' dataset, using different batch size and number of iterations, and then the generated images in each experiment are evaluated using the Frèchet Joint Distance (FJD) (section 1.3.2)

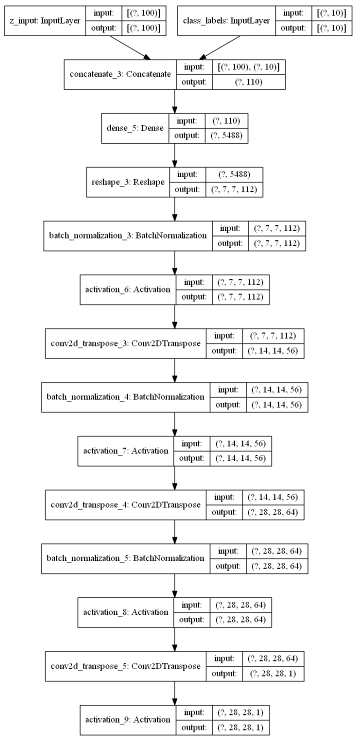
1. $batch\ size = 2$ *and* 15.000 *iterations:* The results of this experiments is showed in Fig. 5. As you can notice with a batch size very small there are very many fluctuations of discriminator's loss but mainly of adversarial loss(Fig.5a). In fact, as you can notice in Fig.5b the generated images are poorly defined. This could be caused by an insufficient number of iterations.

   During the training phase the FJD is computed to evaluate the quality of the generated images and the Fig. 5c shows how this value changes as the number of iterations increases. As you can notice, the FJD score oscillates in a range between about 34 and 43, with a final score of about 36.

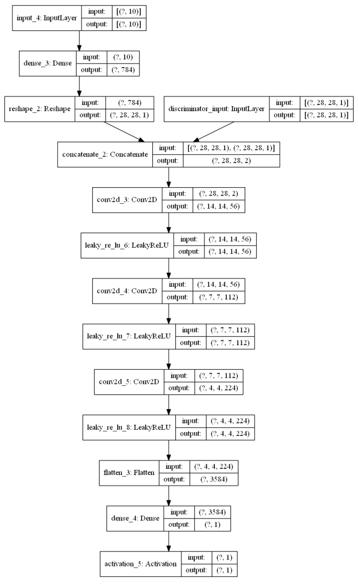
2. $batch\ size = 2$ *and* 100.000 *iterations:* The number of iterations has been increased and there seems to be a slightly improvement of the generated images, and the graphic of the losses (Fig. 6a), also after 100.000 iterations present always many fluctuations, this because such a small batch size requires many more iterations . In the graphic of the FJD (Fig.6c), you can notice that the score tends to become stable in a small range, approximately between 30 and 35, that is a range with smaller values respect to the previous experiment, with a final FJD score of about 31.

3. $batch\ size = 32$ *and* 15.000 *iterations:* The batch size has been increased, and as you can notice, respect to the point 1, the two losses start with a peak, in particular the adversarial loss and then become stable in a smaller range (Fig.7a). Probably, the number of iterations is not sufficient, in fact there are some generated images that are not yet well defined (for example the sandals in the 5^{th} row of the generated images). Here the graphic of FJD has high peak at start and then oscillates between 30 and 35.

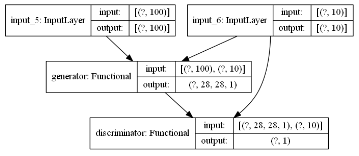
4. $batch\ size = 64$ *and* 200.000 *iterations:* Here the generated (Fig.8b) images are well defined. Observing the graphic of the losses (Fig.8a) you can notice that after about 50.000 iterations the adversarial loss increases because , according to me, the generator is less and less able to foolish the discriminator. Instead the discriminator's loss slowly decreases.


z_input: InputLayer input: [(?, 100)] output: [(?, 100)]
class_labels: InputLayer input: [(?, 10)] output: [(?, 10)]
concatenate_3: Concatenate input: [(?, 100), (?, 10)] output: (?, 110)
dense_5: Dense input: (?, 110) output: (?, 5488)
reshape_3: Reshape input: (?, 5488) output: (?, 7, 7, 112)
batch_normalization_3: BatchNormalization input: (?, 7, 7, 112) output: (?, 7, 7, 112)
activation_6: Activation input: (?, 7, 7, 112) output: (?, 7, 7, 112)
conv2d_transpose_3: Conv2DTranspose input: (?, 7, 7, 112) output: (?, 14, 14, 56)
batch_normalization_4: BatchNormalization input: (?, 14, 14, 56) output: (?, 14, 14, 56)
activation_7: Activation input: (?, 14, 14, 56) output: (?, 14, 14, 56)
conv2d_transpose_4: Conv2DTranspose input: (?, 14, 14, 56) output: (?, 28, 28, 64)
batch_normalization_5: BatchNormalization input: (?, 28, 28, 64) output: (?, 28, 28, 64)
activation_8: Activation input: (?, 28, 28, 64) output: (?, 28, 28, 64)
conv2d_transpose_5: Conv2DTranspose input: (?, 28, 28, 64) output: (?, 28, 28, 1)
activation_9: Activation input: (?, 28, 28, 1) output: (?, 28, 28, 1)

(a) Generator's architecture


input_4: InputLayer input: [(?, 10)] output: [(?, 10)]
dense_3: Dense input: (?, 10) output: (?, 784)
reshape_2: Reshape input: (?, 784) output: (?, 28, 28, 1)
discriminator_input: InputLayer input: [(?, 28, 28, 1)] output: [(?, 28, 28, 1)]
concatenate_2: Concatenate input: [(?, 28, 28, 1), (?, 28, 28, 1)] output: (?, 28, 28, 2)
conv2d_3: Conv2D input: (?, 28, 28, 2) output: (?, 14, 14, 56)
leaky_re_lu_6: LeakyReLU input: (?, 14, 14, 56) output: (?, 14, 14, 56)
conv2d_4: Conv2D input: (?, 14, 14, 56) output: (?, 7, 7, 112)
leaky_re_lu_7: LeakyReLU input: (?, 7, 7, 112) output: (?, 7, 7, 112)
conv2d_5: Conv2D input: (?, 7, 7, 112) output: (?, 4, 4, 224)
leaky_re_lu_8: LeakyReLU input: (?, 4, 4, 224) output: (?, 4, 4, 224)
flatten_3: Flatten input: (?, 4, 4, 224) output: (?, 3584)
dense_4: Dense input: (?, 3584) output: (?, 1)
activation_5: Activation input: (?, 1) output: (?, 1)

(b) Discriminator's architecture


input_5: InputLayer input: [(?, 100)] output: [(?, 100)]
input_6: InputLayer input: [(?, 10)] output: [(?, 10)]
generator: Functional input: [(?, 100), (?, 10)] output: (?, 28, 28, 1)
discriminator: Functional input: [(?, 28, 28, 1), (?, 10)] output: (?, 1)

(c) CGAN 's architecture

Figure 4: Architecture of the first kind of model used in the project

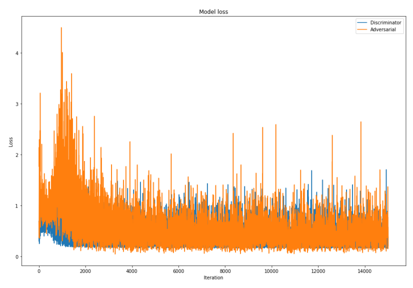
Instead, FJD score, as showed in Fig. 8c, oscillates between about 28 and 32 until 75.000 iterations, and then in average tends to increase. The final FJD score is of about 31.

In conclusion, in order to obtain a good result with generated images well defined, we need to pass, to the model, an appropriate batch size and number of iterations. These parameters can be estimate using the crossvalidation as described in the next section 2.2, and then, as shown in Fig. 8, the model need to be trained until the adversarial loss increases. If it doesn't happens, probably the number of iterations chosen is not enough.

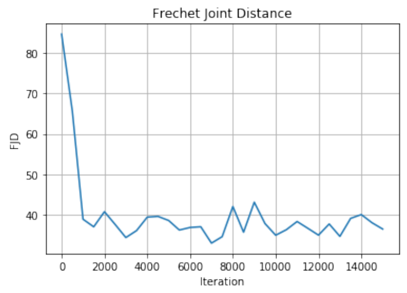
2.2 Crossvalidation

In order to better understand what is the more appropriate batch size value, the project present an implementation of a *crossvalidation* using *K-fold* and the FJD score. Here the crossvalidation is computed keeping fixed the number of iterations (100.000) and iterating for different batch size (2, 32, 64, 128). Since the crossvalidation, for this network, is very expansive, the number of fold chosen, k, is 2. Then, for each batch size kfold is applied. The training set is splitted in training set and validation set. In this way the model is trained using the training set and the FJD score is calculated using the validation set, in order to test the quality of the generated images. After calculating the FJD for each fold is taken the average FJD, and and then the procedure is repeated for the others batch size of the list.

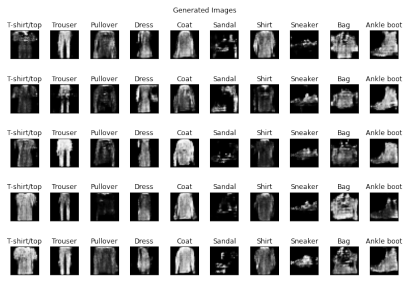
So, the crossvalidation procedure, at the end, returns the average FJD scores computed for each batch size, and the batch size corresponding to minimum FJD will be the best batch size for 100.000 iterations. The histogram in Fig.9 shows the result of the crossvalidation. As you can notice the FJD scores, with 100.000 of iterations, the best FJD score is achieved with 2 of batch size.



(a) Loss of generator and adversarial

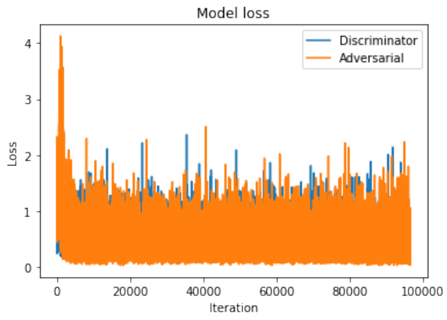


(b) Frèchet Joint distance: about 36 after 15.000 iteration

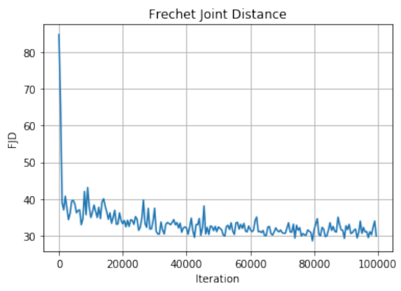


(c) Generated images

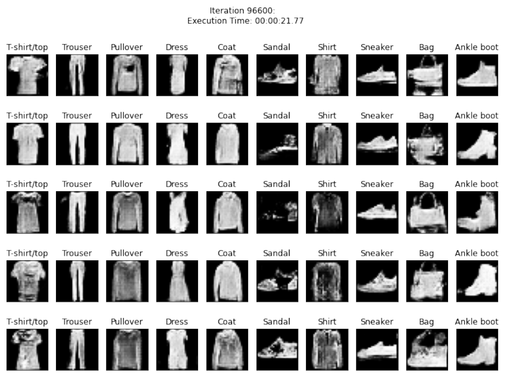
Figure 5: Results with $batch\ size = 2$ and 15.000 iterations



(a) Loss of generator and adversarial

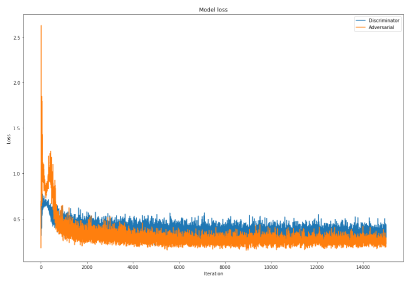


(b) Frèchet Joint distance: about 31 after 100.000 iterations

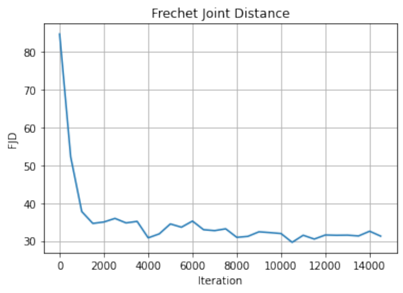


(c) Generated images

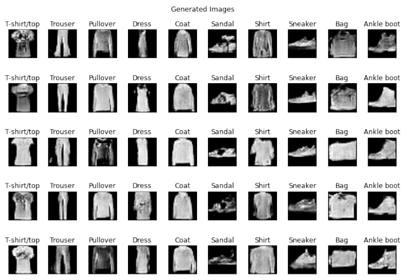
Figure 6: Results with $batch\ size = 2$ and 100.000 iterations



(a) Loss of generator and adversarial

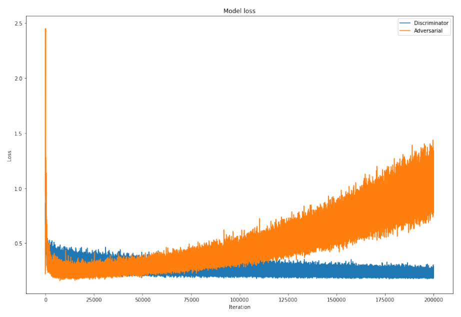


(b) Frèchet Joint distance: about 32 after 15.000 iterations

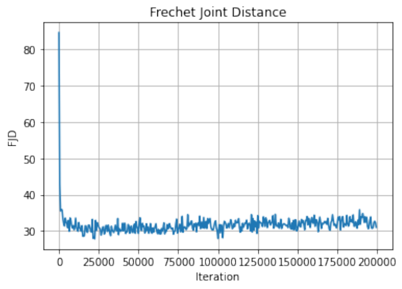


(c) Generated images

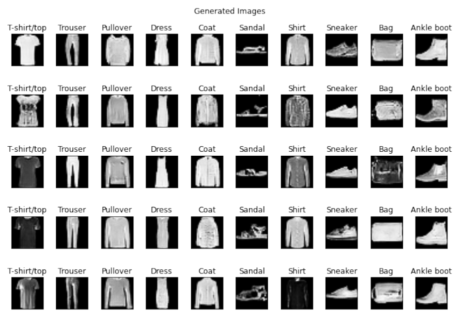
Figure 7: Results with $batch\ size = 32$ and 15.000 iterations



(a) Loss of generator and adversarial



(b) Frèchet Joint distance: about 64 after 200.000 iterations



(c) Generated images

Figure 8: Results with $batch\ size = 64$ and 200.000 iterations

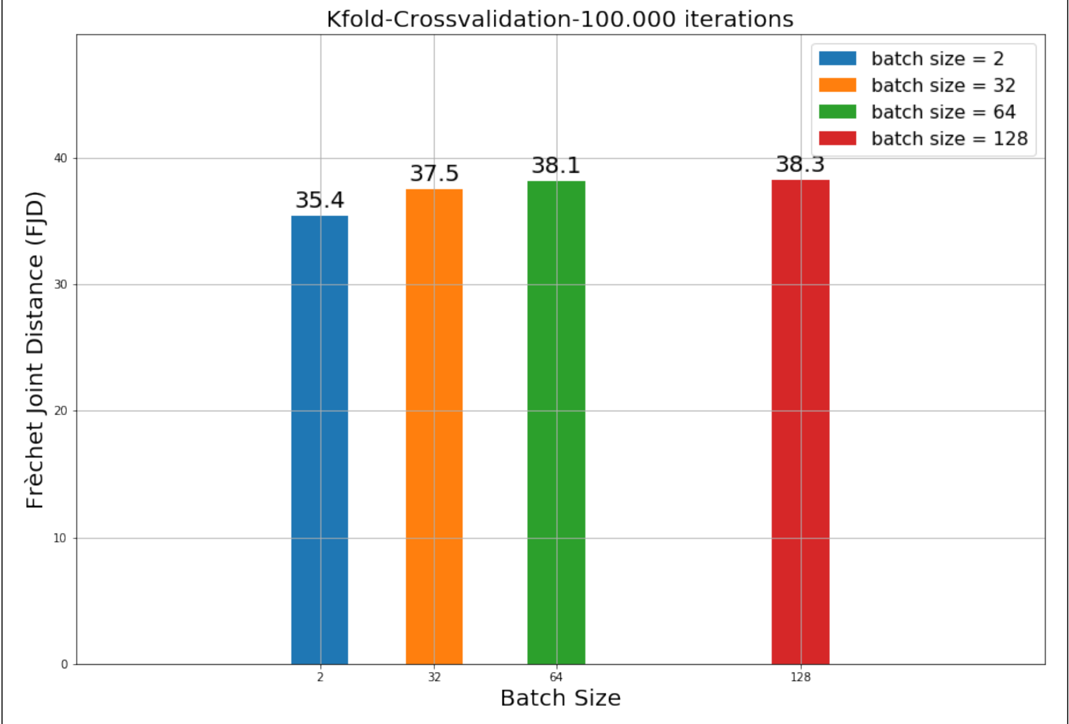


Figure 9: Result Kfold crossvalidation with k=2, with iterations fixed to 100.000 and batch size [2, 32, 64, 128]. On the top of the bins there is the average FJD computed with crossvalidation

References

[1] Conditional Generative Adversarial Nets- Mehdi Mirza, Simon Osindero https://arxiv.org/abs/1411.1784

[2] Generative Adversarial Networks- Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio https://arxiv.org/abs/1406.2661

[3] On the Evaluation of Conditional GANs - Terrance DeVries, Adriana Romero, Luis Pineda, Graham W. Taylor, Michal Drozdzal https://arxiv.org/abs/1907.08175