28462000 - Bourez Nicolas
38502000 - Demaret Dimitri
77102000 - Lhernould Benoit

LINMA2472 - Homework 2 - Group 9

1 Training CGAN on MINST dataset

1.1 Our CGAN model

To improve the initial CGAN, we decided to work on several aspects of the network such as the architecture, the initialization of the weights, the normalization and the deconvolution. It was a hard task because the initial CGAN worked already very well, so we have been inspired by the high quality paper of Takeru Miyato and Masanori Koyama[6].

- **Architecture:** We choose a ResNet architecture. Residual networks are more effective because the residual connections between each bloc of layers make it easy for the network to compute the identity function. The learning is also faster than other deep networks[1].

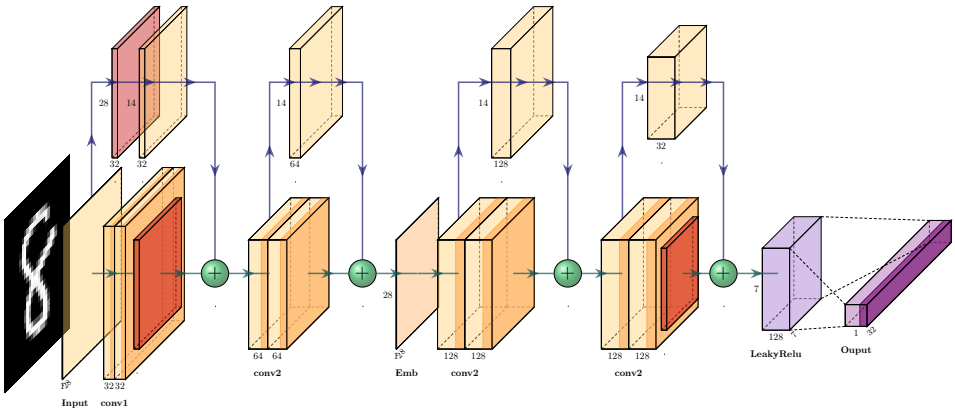


Figure 1: Architecture of the discriminator

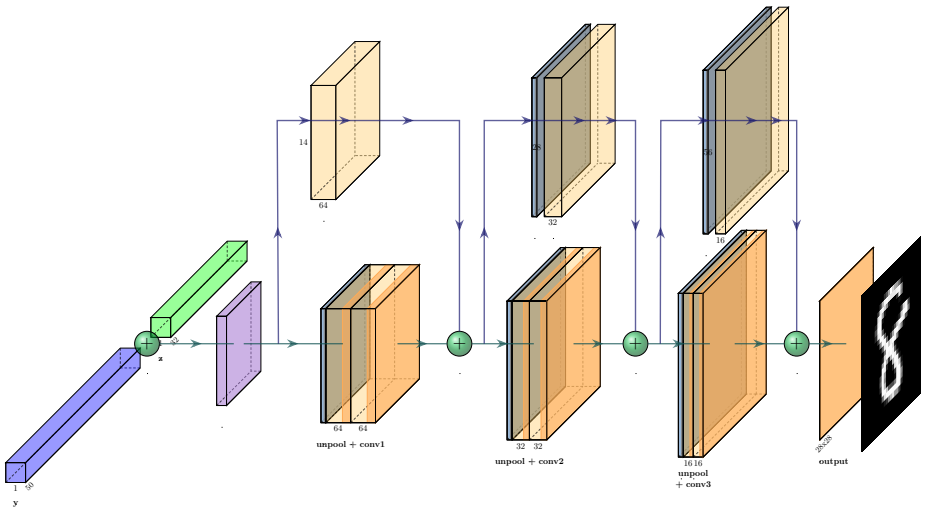


Figure 2: Architecture of the generator

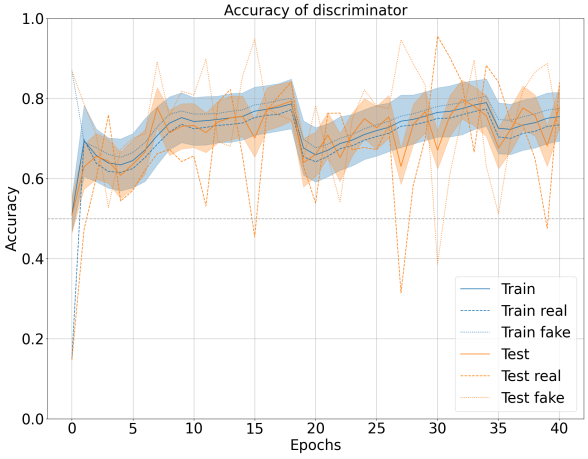
The concatenation is made in the middle of the discriminator and not a the beginning. This is an improvement made by Reed S. et al[4] which proved that it leeds to better inception scores. The structure of the generator is used in Gulrajani et al[5] and the structure of the discriminator in He et alal.[2].Notice that we also use a Batch normalization after each convolution, make a faster and more stable training.

- **Initialization:** We use a Xavier-initialization of the weights between each layer. It allows the activations and gradients to flow effectively during both forward and backpropagation.
- **Normalization:** Intermediate normalization enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with backpropagation. When we apply spectral normalization to the GANs on image generation tasks, the generated examples are more diverse than the conventional weight normalization and achieve better or comparative inception scores[7]. This is because SN regularize the Lipschitz constant.
- **Activation function:** For both architecture, we use the ReLU activation function between each convolutionnal layer.
- **Deconvolution:** Instead of using transposed convolution, we perfom a resize-conv operation. This method helps to reduce artifacts in the generated images[3].

1.2 Training our CGAN

1.2.1 Tunning the advantages

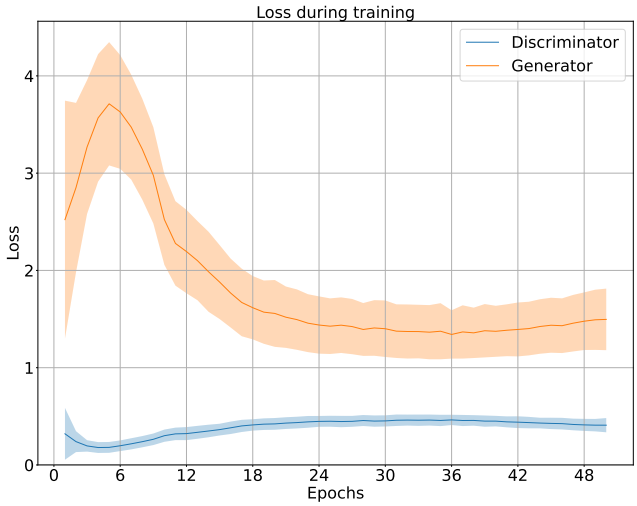
We know that to have a good GAN we need a good generator, but more important, we need a good evaluator. If our teacher don't how to correct our copy, we will never know if we're right or not. To find which of the generator or discriminator was harder to train, we decide to modify the training loop, and increment the advantage of the discriminator if his accuracy was under 0.8, and put it to 1.0 if not. Here is our results :



We can see that the accuracy is slightly increasing to epoch 16, and then drop by almost 1.5 when the training advantages become equal again. That means that our discriminator is way more difficult to train than our generator. We decided to take $discriminator\_advantages = n * (n+1)/2\,/\mathrm{n}|_{n=16} = 8.5 \sim 9$.

1.2.2 Tunning the number of epochs

To find the right number of epoch, we ran a large amount of epoch and find when the model reach convergence :



It seems the models reach convergence at 30 epoch. Here are the final parameters of our model :

Parameter	**CGAN ref**	**OurCGAN**
Device used	GPU	GPU
Number of epochs	15	30
Batch size	64	32
Algorithm/Optimizer used	Adam	Adam
Learning rate (Discriminator)	0.0002	0.0002
Learning rate (Generator)	0.0002	0.00002
Beta1 (Discriminator)	0.5	0.5
Beta1 (Generator)	0.5	0.5
Beta2 (Discriminator)	0.999	0.999
Beta2 (Generator)	0.999	0.999
Generator advantage	1.0	1.0
Discriminator advantage	1.0	9.0
Number parameters (Discriminator)	249,983	618,751
Number parameters (Generator)	229,887	285,375
Training Duration	11 min 59.7 sec	70 min 33

Table 1: Optimization parameters for different CGAN

1.2.3 Comparison of real and generated images

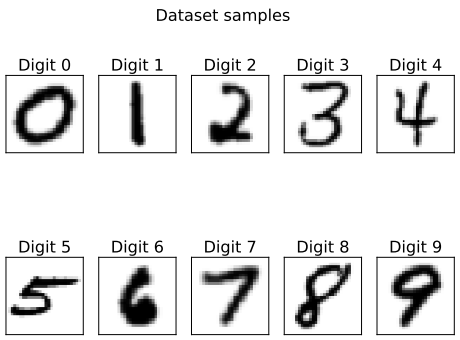


Figure 3: Real data samples

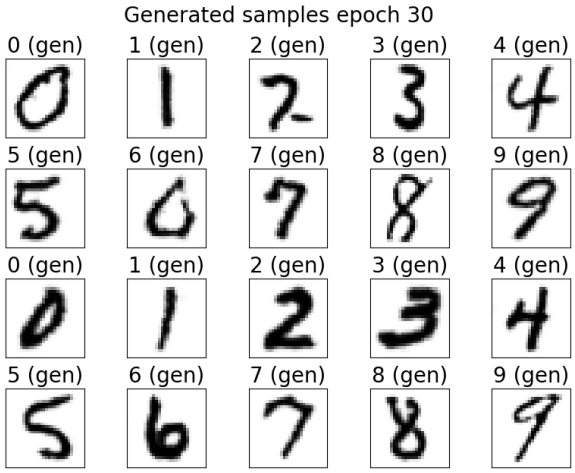
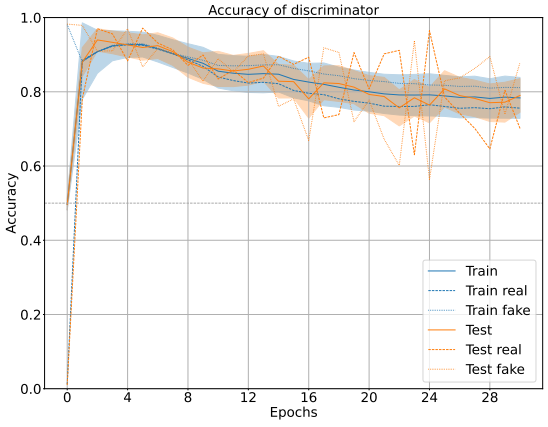
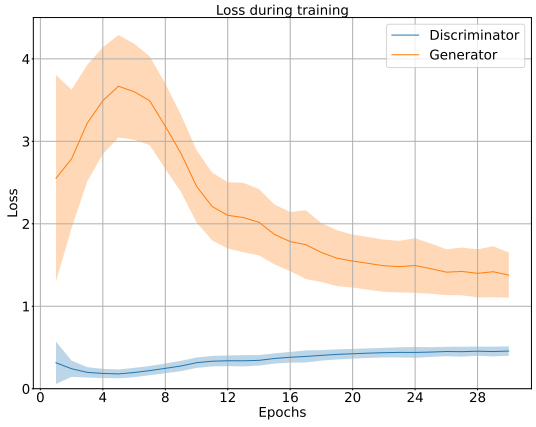


Figure 4: Generated data samples

The CGAN-generated digits closely resemble the MNIST real data samples, with most digits being clear and recognizable. There are minor deviations in some digits like '2' and '6'. Overall, the network shows good learning of the dataset's characteristics.

1.3 Error metrics during training





The evaluation of the CGAN's training progression over 30 epochs demonstrates that while the discriminator began with a marked advantage, reflected in both a lower loss and higher classification accuracy in the training set, its lead is less prominent in the testing set. This pattern suggests that the generator is effectively improving its ability to generalize and create challenging data for the discriminator. The upward trend in accuracy and reduction in loss metrics for both networks indicate successful learning and adaptation, with the generator showing a notable increase in performance, thus reducing the initial performance gap. This overall trend highlights the efficacy of the adversarial training process, with both the generator and discriminator advancing in capability.

1.3.1 Inception score

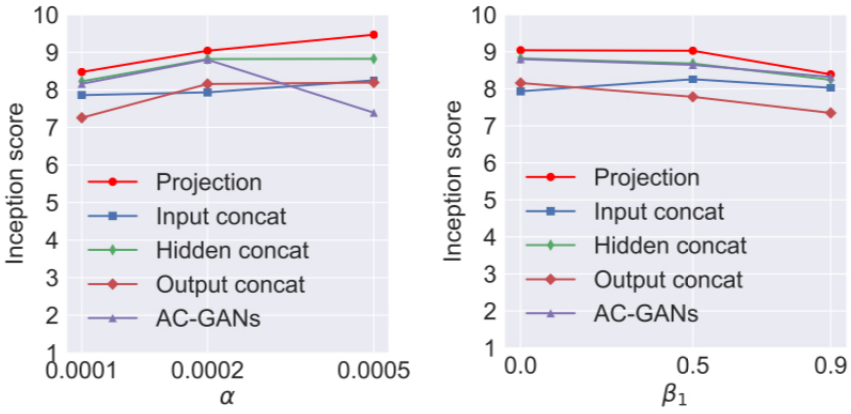
We attempted to implement the Inception Score (IS) to evaluate our CGAN. The IS is generally used for assessing the quality of images generated by GANs, as it provides a measure for both the diversity and the fidelity of the generated images. However, we encountered a significant limitation: the Inception Score is inherently inadequate for datasets like MNIST, which consists of grayscale images of digits. The IS relies on a pre-trained Inception model, which is optimized for color images in more complex datasets like ImageNet with bigger images than in MNIST. This discrepancy means that the Inception model's features are not well-suited to capturing the nuances of the simpler MNIST images, leading to a less meaningful evaluation of our CGAN's performance as we see in table 2.

Epoch	Inception Score (CGAN ref)	Inception Score (Our CGAN)
1	$1.0185312032699585 \pm 0.0013539380161091685$	$1.0222975015640259 \pm 0.001130992197431624$
2	$1.0198116302490234 \pm 0.001112601370550692$	$1.0237021446228027 \pm 0.0014882581308484077$
3	$1.0201919078826904 \pm 0.001420272747054696$	$1.022882342338562 \pm 0.0011722804047167301$
4	$1.0201597213745117 \pm 0.0011492783669382334$	$1.02230703830719 \pm 0.0024651389103382826$
5	$1.0202312469482422 \pm 0.0018963473848998547$	$1.0236608982086182 \pm 0.0017088359454646707$

Table 2: Comparison of CGAN ref and our CGAN Inception Scores

1.4 Improve our CGAN model

To improve our CGAN, we could base our work on a really recent paper, proposing several techniques of projection CGAN[8][6]. Instead of concatening the input with expected labels, they make a projection at the end of the architecture. This increase the performance of a CGAN.



(a) Varying α (fixing $\beta_1 = 0$)

(b) Varying β_1 (fixing $\alpha = 0.0002$)

Figure 11: Inception scores on CIFAR-100 with different discriminator models varying hyper-parameters (α and β_1) of Adam optimizer.

As we can see in the figure 11 from [6], another way to improve is to tune the parameter of the optimizer. We could also change the loss function to something more specialized in our task, like the hinge version of the standard adversarial loss

And finally, instead of given a fixed avantage to the discriminator, we could modifiy the training loop such as the avantages increase and decrease at each epoch, for more flexibility and faster convergence.

2 Explore the latent space

2.1 Diversity in each class

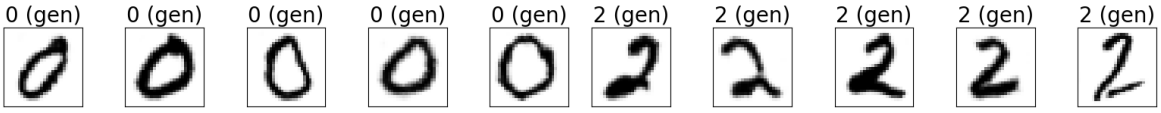


Figure 5: Generate some images of class 0

Figure 6: Generate some images of class 2

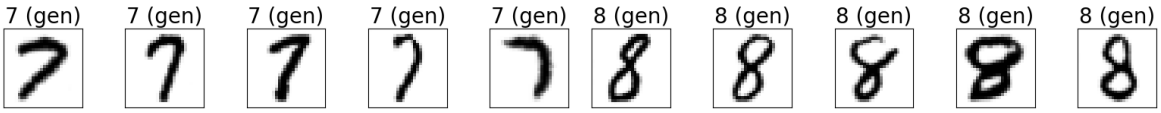


Figure 7: Generate some images of class 7

Figure 8: Generate some images of class 8

Overall, the generator seems to be producing a varied set of results within each class, indicating it has learned a good representation of the handwriting variations present in the training data. The diversity is important for the model to generalize well to new unseen data.

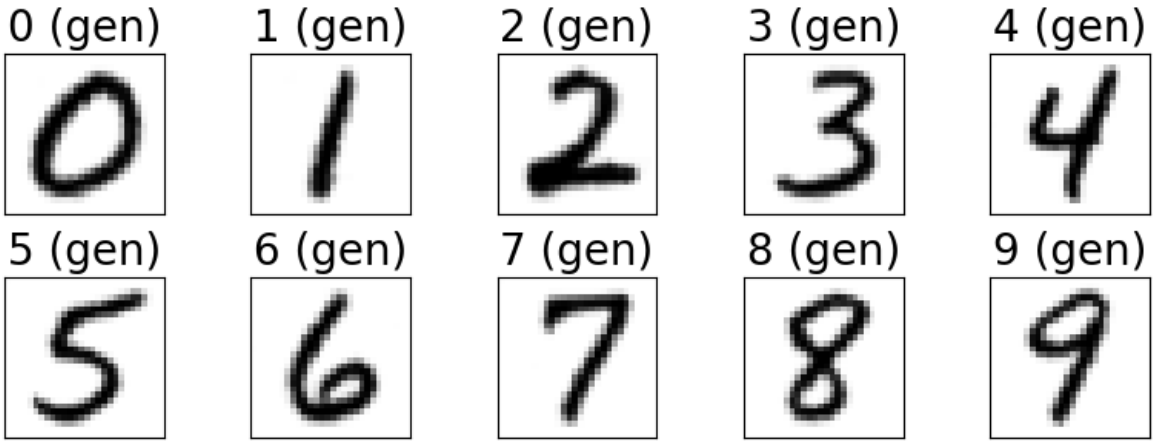


Figure 9: "mean" generated images for each class

Comparing the two, the mean images are more standardized and could be easier for both humans and machine learning models to recognize due to their typical form. In contrast, the randomly generated images demonstrate the diversity in handwriting and might be more challenging to recognize for algorithms not robust to such variations. But this difference in the randomly generated images is crucial for training robust machine learning models that need to handle real-world variability in handwriting.

2.2 Discrimination on each class

Analyzing the discriminator's performance from the data in the table, it's clear that there is a stronger inclination towards correctly identifying fake instances over real ones. The discriminator shows a notably high proficiency in classifying fakes, with accuracy rates reaching as high as 97.8% for Class 9 and not falling below 80.4% for any class. This suggests a strong capability in minimizing false

Class	Real Accuracy	Fake Accuracy
0	83.6%	80.4%
1	48.6%	91.8%
2	70.4%	97.0%
3	45.6%	96.2%
4	64.2%	88.2%
5	69.8%	88.4%
6	64.4%	92.2%
7	72.4%	88.2%
8	64.2%	92.2%
9	59.6%	97.8%

Table 3: Accuracy of real and fake classes

positives. On the other hand, the performance on real instances is less impressive and highly variable across classes. The accuracy dips as low as 45.6% for Class 3, indicating a struggle to correctly identify real instances, which leads to a higher rate of false negatives. The best performance on real instances is seen in Class 0 with an accuracy of 83.6%, which is still lower compared to the fake detection rates.

The discriminator advantage of 9.0 suggests it is significantly better at distinguishing real images from generated ones compared to the generator's ability to create convincing fakes. This imbalance leads to more false negatives, where the discriminator is more likely to mistakenly classify a real image as fake, since it has been fine-tuned to discern real data points very effectively. The lower generator advantage of 1.0 means it produces less convincing fakes, which the discriminator can easily identify, resulting in fewer false positives.

2.3 Outside of distribution

We chose to create fake embedding vectors (fake labels) to see what happens to the samples. First, we put in a list of all of the embedding vectors representing the 10 labels and printed it. It allowed us to know the range of the values creating the labels. We chose the 3 first fake embedding vectors to be in this range and the 2 last embedding vectors to be far from this range. We can distinguish on the plots of the samples generated (Figure 10) the two different cases. The 3 first samples are more or less possible to recognize, even if it seem to be more of an in-between with more than one class, we will see why in the next subsection. The two last samples are not recognizable, the embedding vectors are too far from real labels and thus they generate something very different.

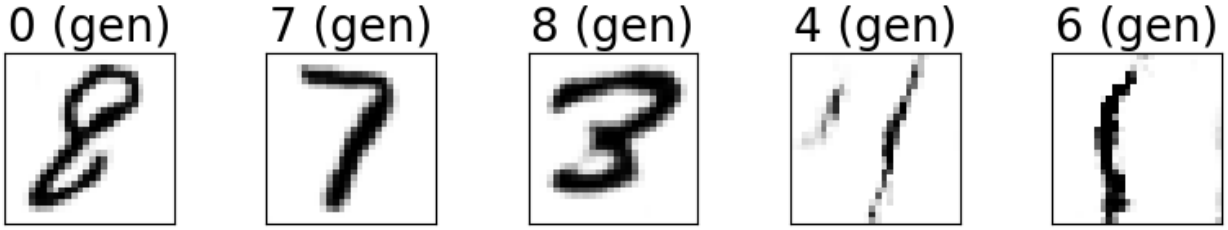


Figure 10: Samples with fake embedding labels

2.4 Interpolation for exploration

To find 2 generated samples visually satisfying, one possibility was to take the mean latent vector (null vector) because as we said earlier, those are easy to recognize. For the other one, we plot random samples, find one sample that is visually satisfying for each class, and retrieve the latent vector that produced this sample. We chose a sample for label 3 which is a bit visually different from the mean one to clearly see the transformation when interpolating between their respective latent vectors (see Figure 11). We can see that the transformation it rather smooth.

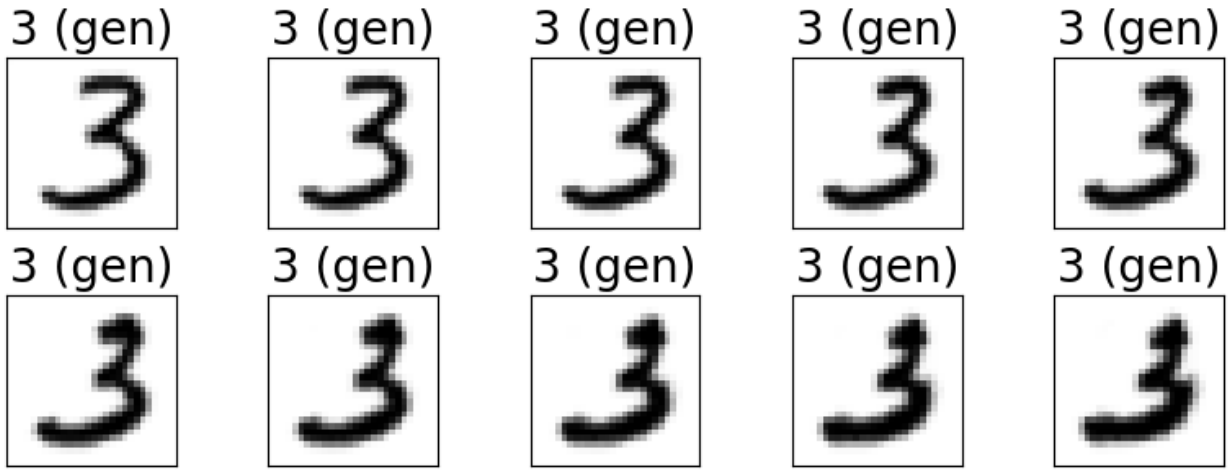


Figure 11: Interpolation within the same class

Now we will concentrate on interpolating between two visually satisfying samples of different classes. As we can see in Figure 12, the transformation is still quite smooth, actually, it seems that the vectors between the two original vectors are slowly losing their belonging to class 2 to become class 8. But the transformation can sometimes go near of another class like we can see with class 9 for the middle samples in Figure 13. We can conclude that the embedding vectors between two embedding vectors representing classes have some parts of different classes. Thus the plots of the samples in the middle are not very satisfying because those are a mixture of classes.

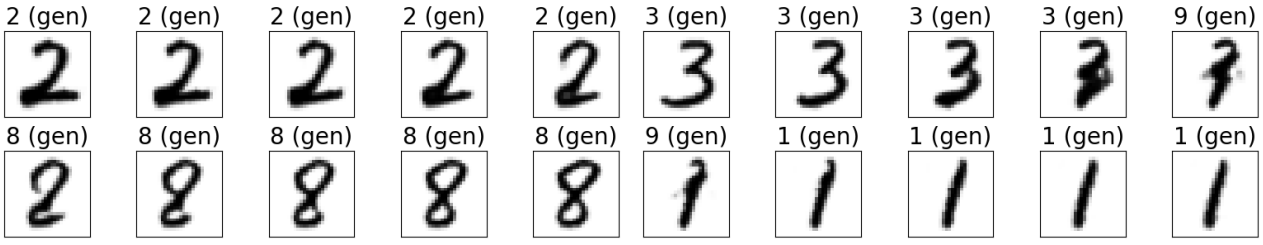


Figure 12: Samples generated for labels 2 to 8

Figure 13: Samples generated for labels 3 to 1

2.5 Noise for exploration

We first put noise on the latent vectors to see the results (see Figure 14. The generator is quite robust to noises on the latent vectors, the digits are obviously more difficult to recognize the more the noise is big but we have a hard time recognizing the digit only with the last value of noise of the homework ($\epsilon = 2$).

Then we added noise on the embedding vectors this time 'see Figure 15). and what we observed was very interesting. The embedding vectors representing labels being near each other in our model, the digits generated quickly became wrong, either a mixture of digits or another class that the initial classes of without any noise. The noise on embedding vectors has an impact bigger than on latent vectors.

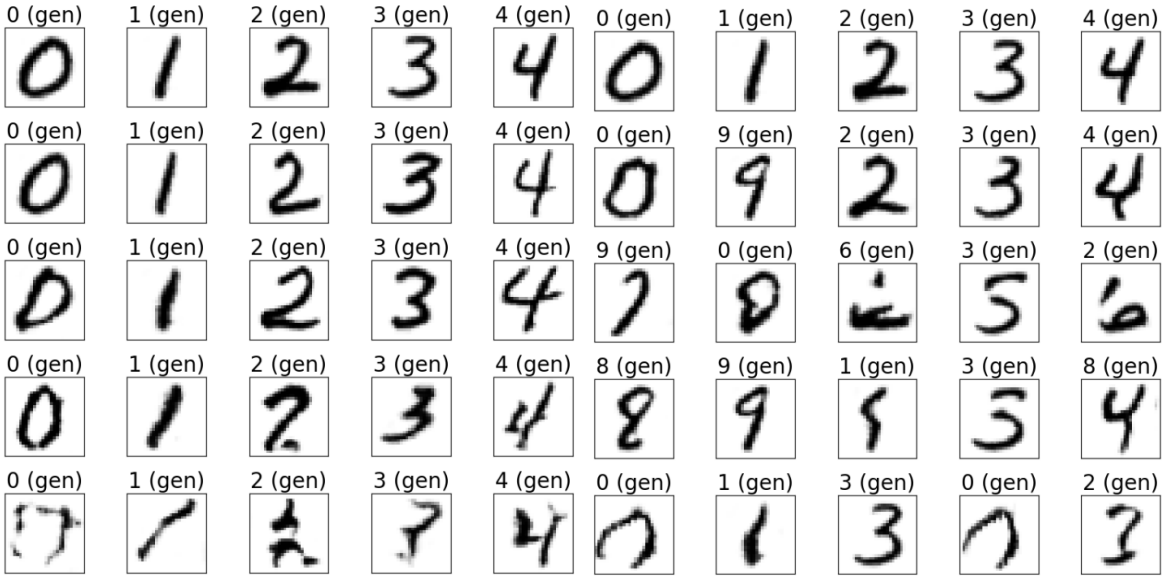


Figure 14: Noises on latent vectors for all values of noises

Figure 15: Noises on embedding vectors for all values of noises

2.6 Conclusion of part 2

In conclusion, the exploration of the latent space gave us an insight of the nature of generative models. The generator's ability to produce diverse representations within each class highlights its effectiveness in capturing the variability inherent in real-world data, which is crucial for creating robust and adaptable models. The discriminator's varying accuracy between real and fake instances underscores the challenges in balancing sensitivity and specificity in model training.

Interpolation within and between classes shows how subtle changes in latent vectors can lead to significant variations in output, suggesting a richly structured latent space. This is further evidenced by the experiments with noise, where even small perturbations can lead to drastic changes in the generated samples, especially on the embedding vectors.

Overall, these findings underscore the importance of thorough exploration and understanding of the latent space in generative models. This helped to improve our model performance.

References

[1] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[2] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027.

[3] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts". In: *Distill* (2016). DOI: 10.23915/distill.00003. URL: http://distill.pub/2016/deconv-checkerboard.

[4] Scott E. Reed et al. "Generative Adversarial Text to Image Synthesis". In: *CoRR* abs/1605.05396 (2016). arXiv: 1605.05396. URL: http://arxiv.org/abs/1605.05396.

[5] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028 (2017). arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028.

[6] Takeru Miyato and Masanori Koyama. "cGANs with Projection Discriminator". In: *CoRR* abs/1802.05637 (2018). arXiv: 1802.05637. URL: http://arxiv.org/abs/1802.05637.

[7] Takeru Miyato et al. "Spectral Normalization for Generative Adversarial Networks". In: *CoRR* abs/1802.05957 (2018). arXiv: 1802.05957. URL: http://arxiv.org/abs/1802.05957.

[8] Ligong Han et al. "Dual Projection Generative Adversarial Networks for Conditional Image Generation". In: *CoRR* abs/2108.09016 (2021). arXiv: 2108.09016. URL: https://arxiv.org/abs/2108.09016.