Deep-Learning-TAU-0510-7255-Spring-2020

Homework 3

Ophir Shurany 304867716

Tim Mironov 321297111

Theory: Question 1

Two two-dimensional joint probability distributions are given:

$$\forall x,y \in P(x,y)\,; x=0, y\sim U(0,1)$$
$$\forall x,y \in Q(x,y)\,; x=\theta, 0\le\theta\le 1, y\sim U(0,1)$$

Section a: $\theta \neq 0$

Therefore:

$$x=0 \quad \Rightarrow P_1(0,y)=1\,;\; Q_1(0,y)=0$$
$$x=\theta\neq 0 \Rightarrow P_2(\theta,y)=0\,;\; Q_2(\theta,y)=1$$

The distance between the probabilities P and Q, when $\theta \neq 0$, is by the three given measures as follows:

Kullback–Leibler Divergence:

Relative entropy is additive for independent distributions, then:

$$D_{KL}(P||Q)=D_{KL}(P_1||Q_1)+D_{KL}(P_2||Q_2)$$

The log-loss is the KL divergence between two probability distributions P and Q:

$$D_{KL}(P||Q)=D_{KL}(P_1||Q_1)+D_{KL}(P_2||Q_2)=$$
$$=\int_0^1 P_1(0,y)\log\left(\frac{P_1(0,y)}{Q_1(0,y)}\right)dy+\int_0^1 P_2(\theta,y)\log\left(\frac{P_2(\theta,y)}{Q_2(\theta,y)}\right)dy$$
$$=\int_0^1 1\underbrace{log\left(\frac{1}{0}\right)}_{\to+\infty}dy+\int_0^1 \underbrace{0\cdot log\left(\frac{0}{1}\right)}_{0\cdot-\infty\to 0}dy=1\cdot\infty+0=\boxed{+\infty}$$

Note that:

$$\lim_{x\to 0} x\cdot\ln(x)=\lim_{x\to 0}\frac{\ln(x)}{x^{-1}}\underbrace{=}_{L'Hopital}\lim_{x\to 0}\frac{x^{-1}}{-x^{-2}}=\lim_{x\to 0}-x=0$$

Jensen-Shannon Divergence

$$D_{JS}(P||Q)=\frac{1}{2}D_{KL}\left(P\parallel\frac{P+Q}{2}\right)+\frac{1}{2}D_{KL}\left(Q\parallel\frac{P+Q}{2}\right)$$
$$=\frac{1}{2}D_{KL}\left(P_1\parallel\frac{P_1+Q_1}{2}\right)+\frac{1}{2}D_{KL}\left(Q_1\parallel\frac{P_1+Q_1}{2}\right)+\frac{1}{2}D_{KL}\left(P_2\parallel\frac{P_2+Q_2}{2}\right)+\frac{1}{2}D_{KL}\left(Q_2\parallel\frac{P_2+Q_2}{2}\right)$$
$$\Rightarrow\frac{1}{2}\left[\int_0^1 P_1(0,y)\log\left(\frac{2P_1(0,y)}{P_1(0,y)+Q_1(0,y)}\right)dy+\int_0^1 Q_1(0,y)\log\left(\frac{2Q_1(0,y)}{P_1(0,y)+Q_1(0,y)}\right)dy\right]$$
$$+\frac{1}{2}\left[\int_0^1 P_2(\theta,y)\log\left(\frac{2P_2(\theta,y)}{P_2(\theta,y)+Q_2(\theta,y)}\right)dy+\int_0^1 Q_2(\theta,y)\log\left(\frac{2Q_2(\theta,y)}{P_2(\theta,y)+Q_2(\theta,y)}\right)dy\right]$$
$$\Rightarrow\frac{1}{2}\left[\int_0^1 1\log\left(\frac{2}{1+0}\right)dy+\int_0^1 0\log\left(\frac{0}{1+0}\right)dy\right]+\frac{1}{2}\left[\int_0^1 0\log\left(\frac{0}{0+1}\right)dy+\int_0^1 1\log\left(\frac{2}{0+1}\right)dy\right]$$
$$\Rightarrow\frac{1}{2}\int_0^1(\log(2)+0+0+\log(2))dy=\boxed{\log(2)}$$

Wasserstein Distance

$$W\big(P(x,y),Q(x,y)\big) = \inf_{\gamma\sim\Pi(P,Q)} \sum_{\xi,\zeta} \gamma(\xi,\zeta)\|\xi-\zeta\| = \inf_{\gamma\sim\Pi(P,Q)} E_{(\xi,\zeta)\sim\gamma}\{\|\xi-\zeta\|\}$$

Since $P=\delta_0$ and $Q=\delta_\theta$ are point masses located at points 0 and θ in $\mathbb{R}^1$, we can use the usual Euclidean norm on $\mathbb{R}^1$ as the distance function, then:

$$W_1(P,Q) = |\theta - 0| = \boxed{|\theta|}$$

Section b: $\theta = 0$

When $\theta = 0$, the probability distribution Q overlap P, $P \equiv Q$. Therefore, it can be figured that the distance between the distributions **should be zero by all measures**. The distance measures are calculated as follows:

Kullback–Leibler Divergence:

$$D_{KL}(P||Q) = \int_0^1 P(0,y)\log\left(\frac{P(0,y)}{Q(0,y)}\right)dy = \int_0^1 0\log(1)\,dy = \boxed{0}$$

Jensen-Shannon Divergence

$$D_{JS}(P||Q) = \frac{1}{2}\int_0^1 P(0,y)\log\left(\frac{2P(0,y)}{P(0,y)+Q(0,y)}\right)dy + \frac{1}{2}\int_0^1 Q(0,y)\log\left(\frac{2Q(0,y)}{P(0,y)+Q(0,y)}\right)dy$$
$$= \frac{1}{2}\int_0^1 0\underbrace{log(1)}_{\equiv 0}\,dy + \frac{1}{2}\int_0^1 0\underbrace{log(1)}_{\equiv 0}\,dy = \boxed{0}$$

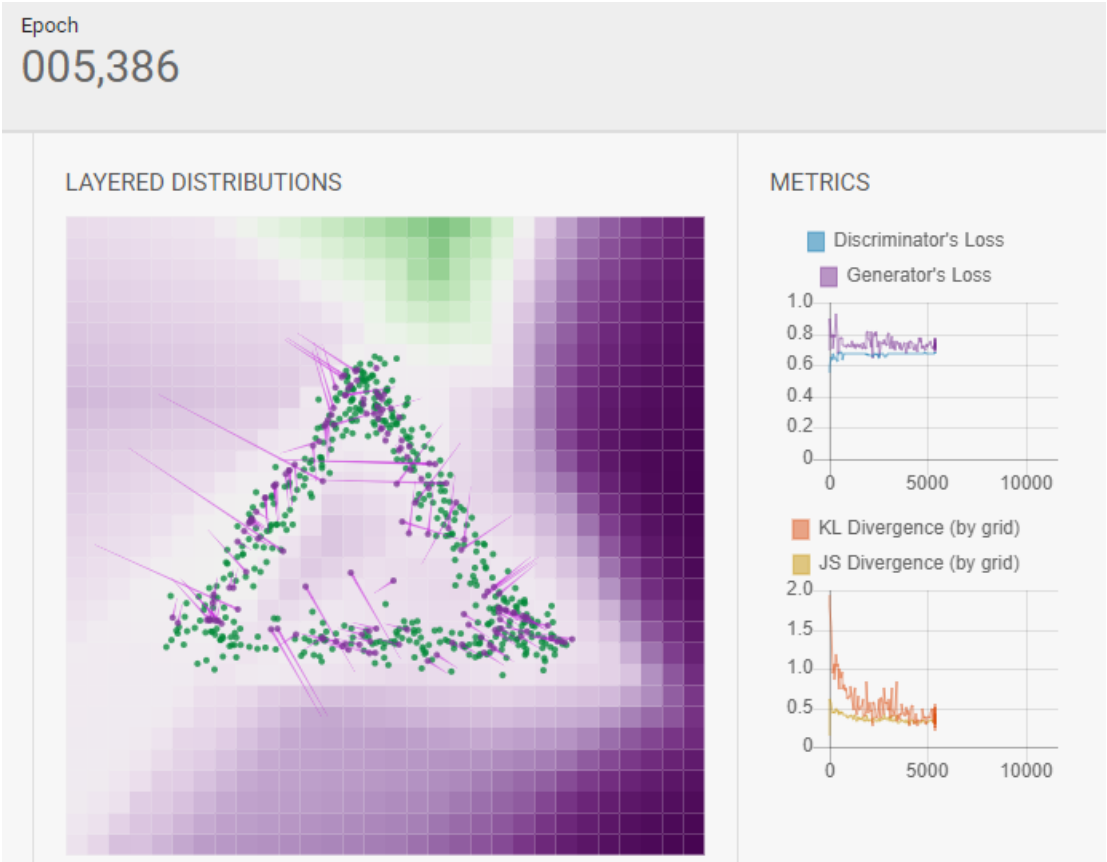
Wasserstein Distance

$$W_1(P,Q) = |0-0| \equiv \boxed{0}$$

Section c:

- D_{KL} is asymmetric and when the two distributions are supported on non-overlapping domains, we get $D_{KL} \to \infty$, which makes the divergence not useful as a metric.
- D_{JS} , though symmetric, still fails to provide a useful gradient when the distributions are supported on non-overlapping domains.
- Wasserstein metric also has the advantage of being a true metric: a measure of distance in a space of probability distributions.
- Wasserstein metric provides a smooth measure, which is important for a stable learning process using gradient descents.
- In GANs implementation Wasserstein GANs avoid problems with vanishing gradients and much less vulnerable to getting stuck than minimax based GANs.

Practical: Question 2



As it can be observed from the fading colors of the background, the discriminator cannot longer distinguish between the real and the fake distribution. Moreover, the losses and the divergences values have been converged.

Practical: Question 3

In this question we were asked to experiment with a variational autoencoder, on the Fashion MNIST database, with the M1 model described in Kingma et al.

Results

The VAE was trained end-to-end for 50 epochs. The loss is presented in the following figure:

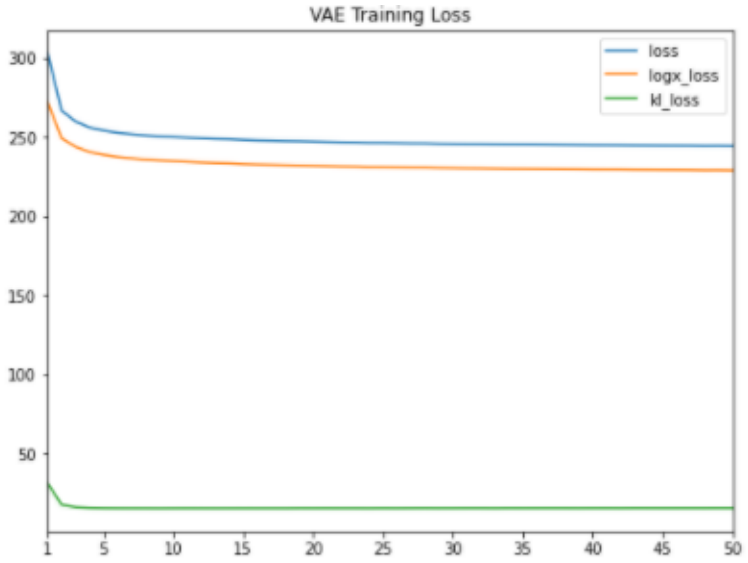


Figure 1 - The VAE training loss

The training loss represents how well the original training images, inserted to the encoder, were reconstructed by the decoder[1]. From inspection of the total loss in Figure 1, it appears that the model greatly converged in the first 5 epochs and slightly improved afterwards until the last epoch.

From a deeper inspection, it appears that the reconstruction loss (logx_loss) is much more dominant than the KL divergence (kl_loss). Therefore, it can be claimed that the training is much more affected from the reconstruction loss than the KL loss.

In addition, this might result a problem where the model will try to improve its decoder, which generates the images, without improving the latent variable resulted by the encoder. The latent variable holds the information about the sample image in a form of a gaussian distribution. When the KL loss doesn't improve, the real images distribution and the generated images distribution won't eventually converge.

This might indicate about the problem shown in the KL divergence in comparison to other measures in question 1 above.

An Encoder and a Generator were constructed out of the trained VAE. The encoder then used to create latent variables out of the Fashion MNIST training set, in order to use with an SVM to classify each latent variable to the different classes in the set. The difference is that the SVM used only a much smaller number of examples in its fitting process compared to a regular CNN classifier (like Lenet-5 we used in Ex1).

[1] Calculated by the "variational lower bound" as described in Kingma and Welling 2014, "*Auto-Encoding Variational Bayes*".

Sets of 100, 600, 1000 and 3000 examples were tested, when each trained SVM was checked on the whole testing set afterwards**. The SVM classifier kernel has chosen to be the default radial basis function,** because of its Euclidean distance-like behavior. The training set was shuffled and divided into an equal number of labels from each class. A fixed seed value of 0 was chosen. The results are presented as follows:

Accuracy	Training set size	Time [s]
65.05%	100	1.85
76.83%	600	2.35
78.43%	1000	3.04
80.17%	3000	6.26

Table 1 - SVM classification results based on the different number of examples in each training set

From Table 1 it appears that as the number of examples improves the SVM prediction accuracy. Yet, it appears that the number of examples used in the SVM fitting is drastically smaller than to the total number of examples found in the Fashion MNIST training set - 60,000 in number. This shows the power of the encoder in revealing the satisfying information needed to recognize the real image label, without inspecting the whole data as is.

In addition, the generator was also tested with random samples of the latent variable, in order to reveal its power in generating images similar to the ones found in the dataset. An example is presented:

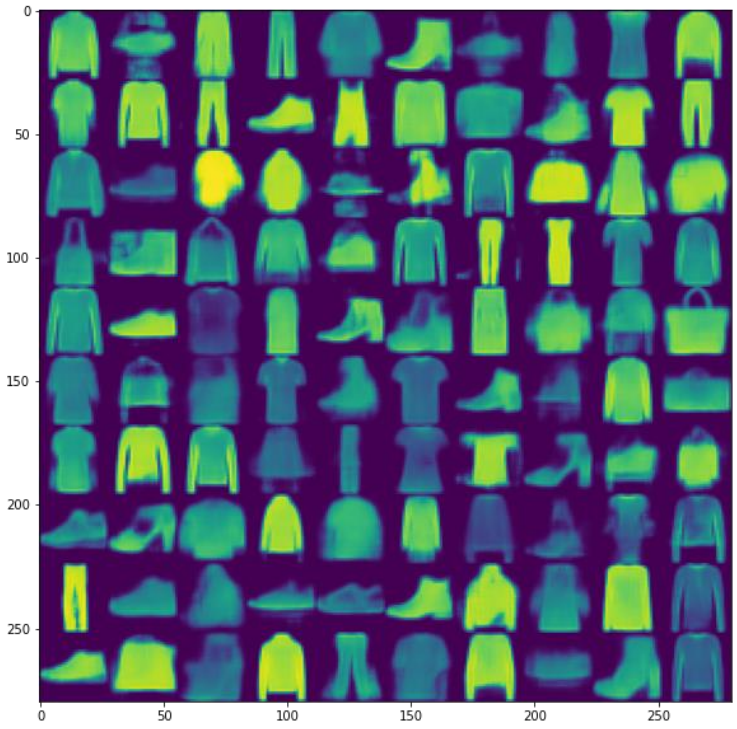


Figure 2 - Random images generated by the VAE decoder

From Figure 2 it appears that the generator has a problem in its generated images. Some images have white holes in it and some even combine different classes from the dataset (in {75,100} appears a combination of a shoe and a bag). This might be a result of the difference in the loss components, which was described above. It is assumed that weighing the loss components by adding a hyperparameter to the

KL loss would even the components and therefore would prevent from improving the decoder over the encoder.

Practical: Question 4

Intro:

In this question we have implemented two GAN models, similar to the architecture proposed in the "Improved Training of Wasserstein GANs" article. First architecture is based on DCGAN with standard loss defined in the original article and the second architecture is a WGAN, based on Wasserstein metric loss, with clipped weights.

In both cases two models were created – one for the generator and one for the discriminator. The generator uses random 128 vector and produces Fashion MNIST images, trying to mimic the training data. The discriminator is getting both the real images from the dataset and the "fake" images produced by the generator. The discriminator tries to learn to recognize the "real" data from the "fake", while in the same time the generator updates it's weights to try and "fool" the discriminator.

Architecture:

Both DCGAN and WGAN generators use three layers of Transposed Convolution with ReLU activation. All layers use Batch Normalization with last layer using tanh activation and having an output of (28, 28, 3), similar to the Fashion MNIST images. We use kernels of (5, 5) with stride of 2, gradually bringing the image to the desired dimensions.

The discriminators use three layered architecture as well, with Batch Normalization, kernels of (5,5) and stride of 2. The activation function used is Leaky ReLU and the last layer is Dense with 1 unit.

Optimization:

In both cases we use Adam optimizer for the generator and the discriminator, with learning rate of 0.0002 and beta1 of 0.5. The models were trained for 90,000 iterations with batch size of 32. In the WGAN training we ran 5 iterations per discriminator on every 1 iteration of generator. In DCGAN the ratio was 1:1.

In the WGAN architecture we clipped the weights by value at 0.1which was chosen as optimal parameter.

Results:

a. **DCGAN:**

The model has converged for many different configurations but has achieved best results (visually tested and in terms of loss stability) for the chosen parameters.

We can see that the generator loss starts at exceptionally large values with random noise values produced at first iterations. As the generator learns to produce better results, the discriminator improves gradually as well and around iteration 10,000 the generator loss starts increasing with discriminator loss decreasing:

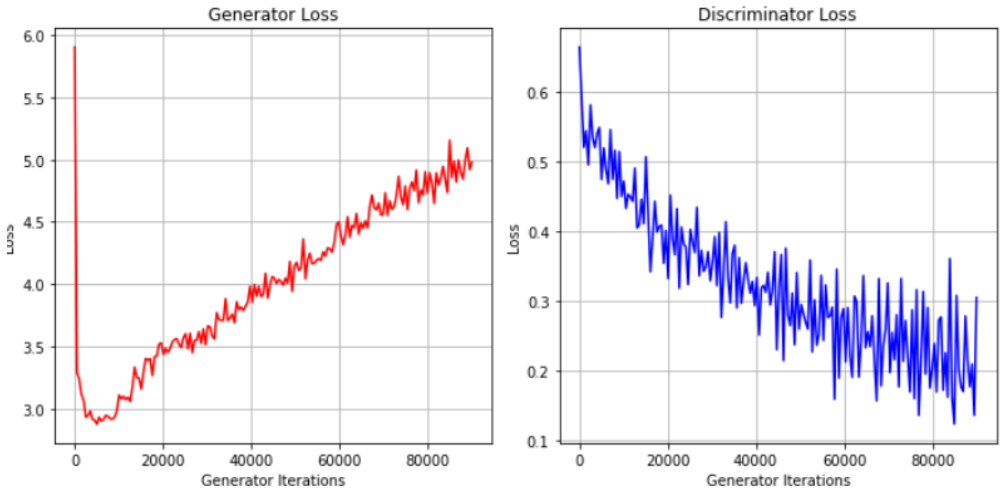


Figure 4 – DCGAN Training Loss

Inspecting the produced images for different iterations we can see that in fact, despite increasing loss, the generator improves and produces better images at later iterations:

Figure 5 – Random images produced by DCGAN generator
Left: Iteration 30,000 Right: Iteration 90,000

b. WGAN:

This model had trouble converging and started to produce images visually similar to the dataset only when the 1:5 ratio of generator to discriminator was applied and the 0.1 weight clipping was used in the training. The training process was ran for 90,000 iterations:

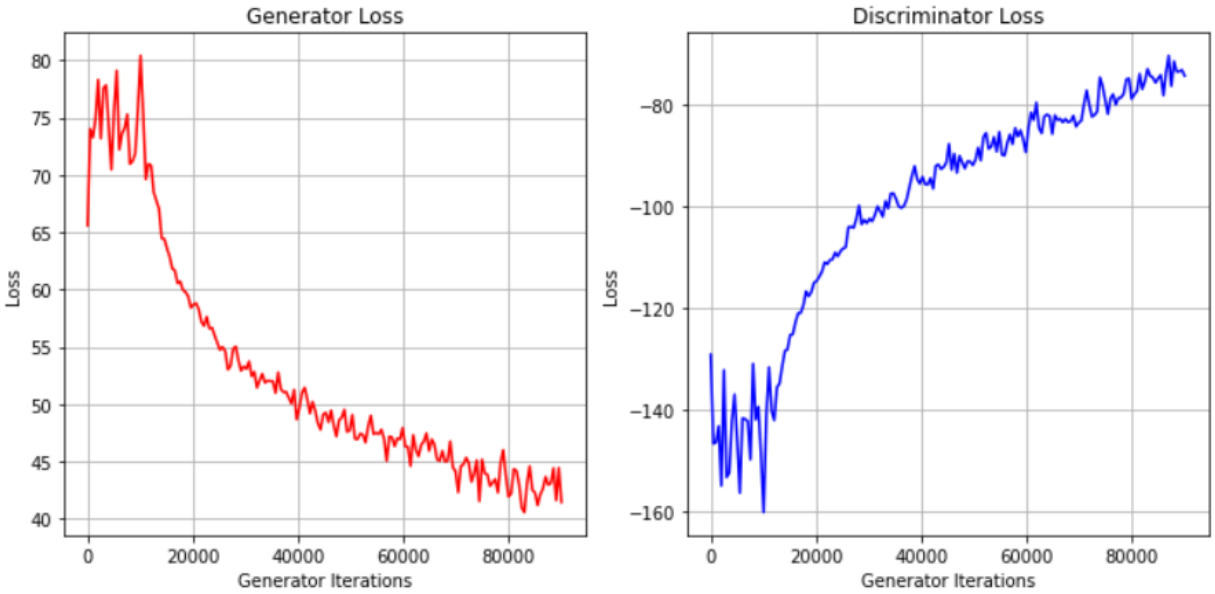


Figure 6 – WGAN Training Loss

We can see a significant improvement in the generator loss around iteration 10,000 where we assume "mode convergence" took place. We can examine the images and compare them:

Figure 7 – Random images produced by WGAN generator
Left: Iteration 10,000 Right: Iteration 90,000

Comparison:

We can visually compare images produced by the two architectures and the respective training data. For example we can look at the "T Shirt" label class:

Figure 8 – Images comparison from the same class "T-Shirt":
*Left: **DCGAN** Middle: **WGAN** Right: Fashion **MNIST***

Another class comparison for "Dress" label:

Figure 9 – Images comparison from the same class "Dress":
*Left: **DCGAN** Middle: **WGAN** Right: Fashion **MNIST***

In both cases we can see the desired behavior of the generators, though the quality of the images produced is visually lower than the training set images. In addition we can examine that some images produced are "undefined", for example in DCGAN:

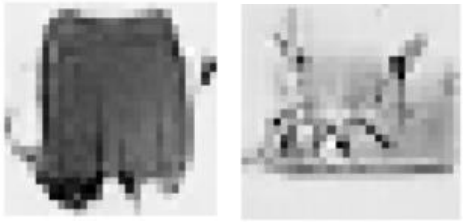


Figure 10 – DCGAN produced "poor quality" images

Or in WGAN:

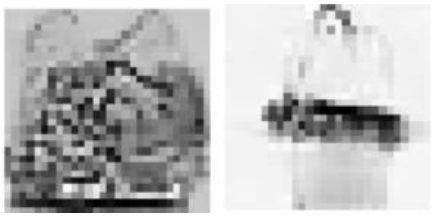


Figure 11 – WGAN produced "poor quality" images

Another important comparison is between the discriminator loss plots of DCGAN and WGAN respectively. The DCGAN discriminator loss converges to zero, as it learns to recognize the generator output at it's loss expense. The WGAN critic's loss drops rapidly initially and then slowly "climbs" to zero:

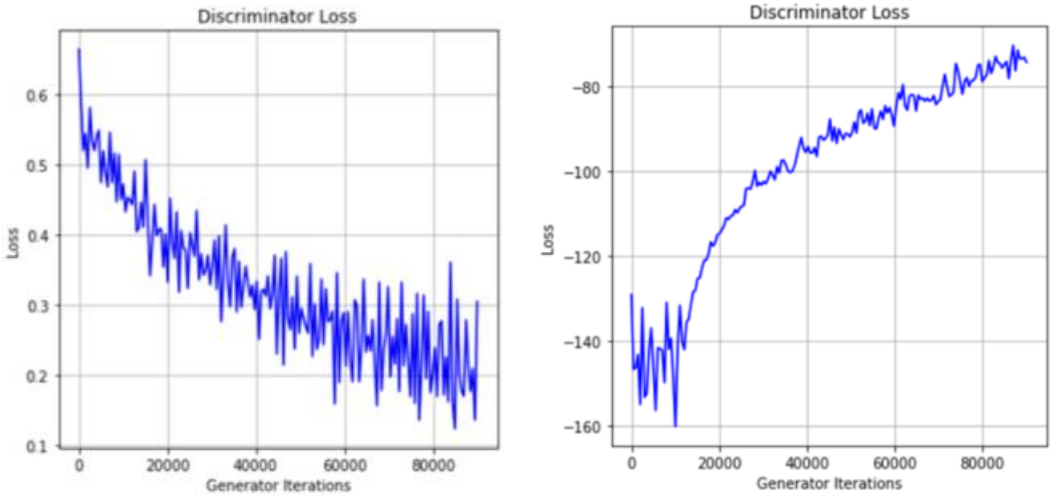


Figure 12 – DCGAN Discriminator Loss (LEFT) and WGAN Critic Loss (RIGHT)

These result were consistent with the results obtained in the original papers:

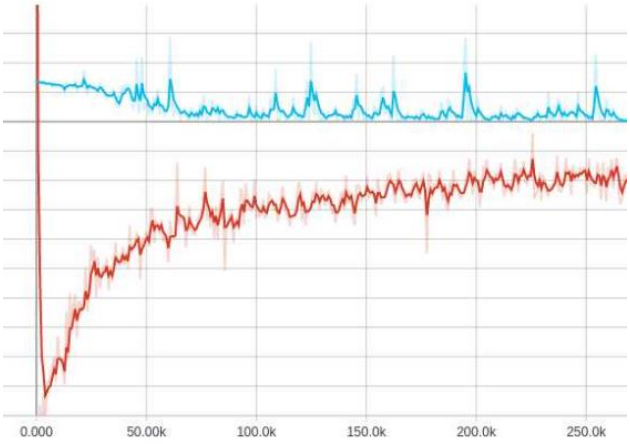


Figure 13 – DCGAN Discriminator Loss (BLUE) and WGAN Critic Loss (RED)