Master Thesis: Using a generative adversarial network for generating microphone array data

Gaspard Ulysse Fragnière

August 2022

1 State-of-the-art

The problem of Acoustical Source Localization (ASL) is an important problem of Acoustics. It arises for instance industrial applications such as the localization of noise on an airplane in a wind tunnel or in smart assistant (e.g. Google Home, Alexa, ...). Traditionnaly this problem is tackled with methods based on the physics of sound propagation (e.g. TDoA, beamforming) or with statistical inference (e.g. Sparse Bayesian Learning).

The recent success of Deep Learning (DL) based method in other field of research (e.g. Computer Vision) led to believe that Deep Neural Networks (DNN) based approaches could provide state-of-the-art result in solving the ASL problem. Castellini et al. (2021), Kujawski et al. (2019), Lee et al. (2021), Ma and Liu (2019), Pinto et al. (2021) and Xu et al. (2021) propose state-of-the-arts DL-based methods for Source Characterization.

A common issue faced while implementing DL-based methods is that significant quantities of well structured data are required. In the literature, the data has been obtained using the following approaches:

- **Real Mesurement**: To create the different samples of such a dataset, sounds emitted with a loudspeaker or human voices are recorded in a real acoustic enviromnent. Eventhough such a method allows for the creation of perfectly realistic samples, it does not come without any issue. Indeed, it is very tedious and time-consuming to record in different environments. Additionally, all the environment for measurement need to physically exists, which limits the quantity of possible samples. Moreover, to build a high quality data set, expensive equipment is required to have an accurate groundtruth (i.e. precisely identify the location of the sources). In the literature, He et al. (2018) and Ferguson et al. (2018) have used such an approach.

- **Synthetic Data**: The sounds used are artificial (i.e. white noise, sine wave). The room acoustic is also simulated. The dry sound is convolved with a simulated Room Impulse Response (RIR) to mimic the effect of room acoustics (e.g. reverberation). Compared to real measurement, this approach allows sample in more diverse environment. Indeed RIR for rooms of arbitrary size, different source position as well as different dry signals can be used for the training. The issue with such a method is the important amount of time and storage required for the creation of the datasets. E.g. Chakrabarty and Habets (2017), Perotin et al. (2018) and Adavanne et al. (2018) created their datasets in this way.

- **Semi-synthetic data**: The creation of such a dataset is similar the creation of synthetic dataset. The difference lies in the fact that the dry sound source used and the RIR are measured and not simulated. Then, the samples of such a dataset are generated by convolving dry sounds with measured RIR. This method is not the best suited, since it is very time-consuming to generate a data set with enough samples for training a DL-based algorithm. Indeed, measuring all the RIR lead to the issues faced with real measurement. Takeda and Komatani (2016) use such an approach for to obtain their data.

Moreover it is to be noted that none of these methods are suitable for online data generation. Indeed, any of the above mentionned method do not allow for creating random sample while training DL-based algorithm. To use such datasets for training, they need to be fully created (and stored) before any training can occur.

1.1 DL-based data generation

In the past years, DL-based approaches have shown to be able to learn and realistically reproduce very complicated data structures (e.g. generation of pictures of faces in the field of Computer Vision). Those breakthroughs lead to believe that similar data generation methods could be used to fix the above-mentionned issues such as offline training, lack of variance in the different samples, ...

Moreover, it is relevant to note that the data used for source characterization in Castellini et al. (2021), Lee et al. (2021), Ma and Liu (2019), Xu et al. (2021) is the Cross Power Spectra (CPS), i.e. a direct representation of the signals received in the array of microphone. Indeed those approaches do not use direct recording of microphone input but instead features extracted from the raw data. This is crucial because it means that recording, simulating or generating raw microphone data is no longer necessary, if features (e.g. CPS) could be generated directly. We therefore need to identify what acoustic quantities:

- have already been generated using a DL approach
- are potential feature for a Source Characterization Algorithm.

1.1.1 Generation of Signal

Neekhara et al. (2019), Kumar et al. (2019), Engel et al. (2019) use Generative Adversial Network (GAN) to generate realistic audio waveform. Neekhara et al. (2019) and Kumar et al. (2019) specifically focus on the generation of audio waveform conditioned on a spectogram (cGAN). On the other hand, Engel et al. (2019) design a GAN to generate realistic audio waveform of single music notes played by an instrument. The data generated in those approaches is single-channel data, but maybe it could be extended to multi-channel to simulate the different signals recorded in an array of microphone. It is relevant to note that the GAN designed by Neekhara et al. (2019) is the one implemented in Vargas et al. (2021) in order to compare the accuracy of a network for single source DoA estimation when trained with different sound classes.

1.1.2 Generation of Impulse Response

Papayiannis et al. (2019) introduce a GAN approach to generate artificial Acoustical Impulse Response (AIR) of different environment in order to generate data for a NN used for classification of acoustic environment.

Ratnarajah et al. (2021) proposes a fast method (NN-based) for generating Room Impulse Response (RIR). The input paramaters of the networks used for creating the IR are the desired dimensions of the rectangular room, listener position, speaker position and reverberation time (T_{60}).

This could be relevant to solve the problem at hand because if we are to be able to generate impulse responses with known source and listener position, we could simply convolve them with the dry source sounds. This way, we could generate raw microphone signal and use them to train a DL-based algorithm for source characterization.

1.1.3 Generation of potential NN feature

Bianco et al. (2020) proposes an approach to generate another acoustic feature: the phase of the relative transfer function (RTF) between two microphones. In this paper a Variational Auto Encoder (VAE) is designed to simultaneously generate phases of RTF and classifying them by their Direction of Arrival (DoA).

Gerstoft et al. (2020) use a GAN to generate Sample Cross Spectra Matrices (CSM). for a given DoA. In their approach, the GAN is trained with data only coming from one DoA, making it unable to generate sample for different DoA. This approach could be extended by creating a conditional Generative Adversial Network (cGAN) taking as input the DoA. Such a GAN would receive a DoA as input and use it to produce a CSM corresponding to the received DoA.

1.1.4 Other possible approaches to generate the data

In Hübner et al. (2021) introduce a low complexity model-based method for generating samples of microphones phases. This method proposed is not based on DL. Indeed, it is based on a statistical noise model, a deterministic direct-path model for the point source, and a statistical model. The claim of this paper is that the low complexity of the proposed model makes it suited for online training data generation.

Vera-Diaz et al. (2021) introduce a CNN for denoising (i.e. removing the effects of reverberation and multipath effects) on the Generalization Cross Correlation (GCC) matrix of an array of microphone. More specifically than a CNN, the network used has a encoder-decoder structure. This means that a possible approach to create the data we want, would be to attempt to invert network proposed. With this we could realistically add noise to GCC matrices and hence making it suitable for training.

2 Fundamentals

2.1 Propagation model and the Cross Spectral Matrix

If we consider M spatially distributed receivers, J uncorrelated sources and a linear propagation model, the complex sound pressure at the mth microphone is given by

$$p(\mathbf{r}_m, \omega) = \sum_{j=1}^{J} h_{mj}(\omega) q(\mathbf{r}_j, \omega) + n(\mathbf{r}_m, \omega) \tag{1}$$

Where ω is the angular frequency and h_{mj} is the transfer function describing the the sound propagation from the jth source to the mth sensor. $n(\mathbf{r}_m, \omega)$ models independent noise. The above equation can be rewrittem as a matrix form

$$\mathbf{p} = \mathbf{H}\mathbf{q} + \mathbf{n} \tag{2}$$

The Cross Spectral Matrix (CSM) or the Sample Covariance Matrix (SCM) is a quantity used in most beamforming algorithm. It can be approximated as

$$\hat{\mathbf{C}} = \frac{1}{B}\sum_{b=1}^{B} \mathbf{p}\mathbf{p}^H \tag{3}$$

with B snapshots. The CSM is used in most beamforming algorithm, because most information about the location of a source signal are contained in it.

2.2 Eigenvalue decomposition and Rank I Cross spectral matrix

If we consider a Cross spectral matrix $\mathbf{A}$ of dimension $M \times M$, it can be factorized into a canonical form: a representation by its eigenvalues and eigenvectors using:

$$\hat{\mathbf{C}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H \tag{4}$$

where $\mathbf{V} = [\mathbf{v}_1^T, \ldots, \mathbf{v}_M^T] \in \mathbb{C}^{M\times M}$, $\mathbf{v}_i$ being the ith eigenvector and where $\mathbf{\Lambda} \in \mathbb{R}^{M\times M}$ is a diagonal matrix, where $\mathbf{\Lambda}_{ii}$ is the ith eigenvalue, corresponding to the ith eigenvector. The eigendecompistion is useful because the eigenvalues provide good insights to separate signal from noise., as well as estimating a source strength.

Sarradj (2010) uses the eigendecomposition to introduces the Rank I Cross Spectral Matrix. If we consider approximated CSM $\hat{\mathbf{C}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$, then the Rank I CSM $\hat{\mathbf{C}}_i$ can be computed with:

$$\hat{\mathbf{C}}_i = \mathbf{v}_i \mathbf{\Lambda}_{ii} \mathbf{v}_i^H \tag{5}$$

As we'll see in the next subsection, the rank I CSM can be used to create a beamforming map for only one eigenvalue, corresponding to one audio source.

2.3 Conventional beamforming

Let us consider the case where there is only one single sound source s. Then all the microphone of the grid will record the sound of the source, all with different time delays, depending on the different microphones position, as well as source's location. Using teh data from all microphone, a beamforming map can be created, namely a map where the maximum value is at the position of the sound source.

More formally, let us consider the sitatuation introduce in the Propagation model. We remind that we have M microphones organised as an array (namely in a plane) but this time only one source (i.e. $J = 1$). Moreover we have the sound pressure vector $\mathbf{p} \in \mathbb{C}^M$ at every microphone and the corresponding approximated CSM $\hat{\mathbf{C}} = \frac{1}{B}\sum_{b=1}^{B} \mathbf{p}\mathbf{p}^H$.

We can then define a scan grid of N potential position of sound sources. For the sources, the assumption made is that the propagation model is a monopole source. Each position of the grid has a position vector ξ_n. For each grid point ξ_n, we compute the expected signal that would be recorded by each microphone. Every microphone as a position vector x_m, for $m \in [1, \ldots, M]$. This allows to introduce the steering vector $g_{n,m} \in \mathbb{C}^M$, defined as

$$g_{n,m} = \frac{\exp(-2\pi i f \Delta t_{n,m})}{4\pi ||x_m - \xi_n||} = \frac{\exp\left[-2\pi i f \frac{||x_m - \xi_n||}{c}\right]}{4\pi ||x_m - \xi_n||} \tag{6}$$

Where f is the sound frequency under consideration and $\Delta t_{n,m}$ is the propagation time from source n to microphone m. $c = 343$ [m/s] is the speed of sound in air. Other definition of the steering vector for different propagation model can be found in Sarradj (2012).

Using the steering vectors $g_{n,m}$ for $(n, m) \in [1, \ldots, N] \times [1, \ldots, M]$ and the approximated CSM $\hat{\mathbf{C}}$, we can compare the model sound pressure, with the recorded sound pressure. By finding a match, the source location can be identified. To do so, we compute the source autopower per scan grid point ξ_n:

$$A(\xi_n) = \frac{1}{2} \frac{\mathbf{g}_n^H \hat{\mathbf{C}} \mathbf{g}_n}{||\mathbf{g}_n||^4} \tag{7}$$

This provides us with a source map. An example of source map can be seen in picture **TODO: add source map**. Moreover, Sarradj (2010) shows that equation 7, can also be used with a Rank I CSM $\hat{\mathbf{C}}_i$, giving

$$A(\xi_n) = \frac{1}{2} \frac{\mathbf{g}_n^H \hat{\mathbf{C}}_i \mathbf{g}_n}{||\mathbf{g}_n||^4} \tag{8}$$

This gives us a beamforming map corresponding to only one single source, the one with associated eigenvalue $\mathbf{\Lambda}_{ii}$

2.4 GAN

Goodfellow et al. (2020) introduce a new approach to solve the problem of generative model. The goal is to learn the probability distribution that was used to generate samples, by observing them.

The method introduced in this paper is called Generarive Adversial Network (GAN). The idea is to create a game (in the game theory sense) where two networks compete against each other. The first network is called a discriminator and its goal is to determine real from fake samples. The other network is a generator with the aim to produce data realisitic enough that the discriminator cannot determine that it has been generated.

The generator takes as input a random vector and use it generate a sample. This random vector is referred to as latent variable. The vector space of latent variable is called latent space. After training, the generator should be a mapping from the latent space to the data space. The latent space is a representation of smaller dimension of the data space.

The discriminator takes as input a real or generated sample and predicts its authenticity. The discriminator is a simple binary classifier. The real sample come from the datasets and the fake are outputs of the generator.

Both models are trained simultaneously. First the generator create a batch of fake sample. This batch is then fed to the discriminator, alongside a batch of real samples. For each of them, the discriminator makes a prediction about their authenticity. The discriminator then gets updated based on how accurate it was at classifying samples and the generator based on how many times fake sample were able to fool the discriminator. We can see here that the training of both networks is supervised, on the contrary of typical generative models.

In a gaming theory framework, both networks are competing in a zero-sum game. This means that if one network performs well at its task and gets rewarded by little weights update, the other must have performed poorly and hence gets penalized by heavy weights updated. E.g. if the discriminator was successful at classifying all samples, it means that the generator had not been able to fool the discriminator by producing any realistic fake samples.

2.5 DCGAN

Radford et al. (2015) introduce GAN, but unlike in Goodfellow et al. (2020), the architecture of the discriminator and generator is not achieved with regular perceptron, but with Convolutional layer. We first remind how a regular perceptron layer works. For an input vector $\mathbf{x} \in \mathbb{R}^k$ and output vector $\mathbf{y} \in \mathbb{R}^l$, a perceptron layer consists of two parts:

- an activation $\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}$
- a non-linearity $y = f(\mathbf{x}; \theta) = \sigma(\mathbf{a})$

where $\mathbf{W} \in \mathbb{R}^{l \times k}$ are the weights of the perceptron and $\mathbf{b} \in \mathbb{R}^l$ its bias. $\sigma(\cdot)$ is called the activation function and is the source of non linearity in neural networks. An example of such activation function is the Rectified Linear Unit (ReLU) where:

$$\sigma_{\text{ReLU}}(\mathbf{a}) = [\max(a_0, 0), \ldots, \max(a_{l-1}, 0)]^T \tag{9}$$

We can now introduce the architecture of a convolutional layer. Without loss of generality, we assume that our network reecive as input an square image with a single channel, i.e. $\mathbf{H_0} \in \mathbb{R}^{d_0 \times (m \times m)}$ with $d_0 = 1$, transforms it with one square kernel, i.e. $\mathbf{K} \in \mathbb{R}^{s \times (r \times r)}$ with $s = 1$ to output another square image with a single channel, i.e. $\mathbf{H_1} \in \mathbb{R}^{d_1 \times (n \times n)}$, with $d_1 = 1$

The relationship between input image $\mathbf{H_0} \in \mathbb{R}^{m \times m}$, output image $\mathbf{H_1} \in \mathbb{R}^{n \times n}$ and kernel $\mathbf{K} \in \mathbb{R}^{r \times r}$ is defined as:

- an activation $\mathbf{A} = \mathbf{H_0} * \mathbf{K}$
- a non-linearity $\mathbf{H_1}_{(i,j)} = \sigma(\mathbf{A}_{(i,j)})$

Radford et al. (2015) proposes a set of constraint that makes the use of convulotional layers possible in GAN setting.

2.6 WGAN

Arjovsky et al. (2017) introduce a new type of Generative Adversial Network (GAN): the Wasserstein GAN (WGAN). The claim is that WGAN improves the stability in learning and get rid of typical problem of the traditional GAN approach such as Mode Collapse or Convergence failure. Mode collapse is when a GAN fails to fully converged because the generator learns only to generate a subset of the real data in order to fool the discriminator. This leads to a situation where the GAN is able to produce realistic data but not all kind of data, which is not a desired behaviour. More specifically, Arjovsky et al. (2017) provides the following insights:

- Introduces the Earth-Mover (EM) or Wasserstein distance. It is a metric used to quantify the distance between two probability distributions.

- Analyses how the EM distance behaves compared to other distances between probability distribution (e.g. Kullback-Leibler distance).
- Define a GAN minimizing an approximation of the EM distance, namely the WGAN.
- Show that unlike traditional GANs, WGANs do not need to maintain a balance when training the discriminator and generator. Indeed in regular GAN approach, it was crucial to avoid the discriminator to become too good before the generator, since this would prevent the generator to learn any distribution.
- Provides useful insights to determine whether the WGAN has converged or not. Determining convergence was a hard problem with the regular GAN approached and it was typically assessed by visually determining if the produced samples were realistic or not.

2.6.1 The Earth-Mover or Wasserstein distance

The goal of WGAN, remains the same as GAN, namely solving the problem of generative model. More precisely, this mean approximating the probability distribution P_r of some data by a distribution P_θ. For this reason, it is necessary to have metrics to quantify distance between two probability distributions P_r and P_m. Example of such distances are the Kullback-Leibler divergence or the Jensen-Shannon divergence. In WGAN, the distance used is the Earth-Mover (EM) distance or Wasserstein-1, defined as

$$W(\mathbb{P}_r, \mathbb{P}_m) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_m)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \tag{10}$$

Where $\Pi(\mathbb{P}_r, \mathbb{P}_m)$ is the set of all joint distribution $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_m$. Informally, $\gamma(x, y)$ shows how much "mass" must be carried to transform $\mathbb{P}_r$ into $\mathbb{P}_m$. The EM distance is then "the cost" of the optimal "transport".

Arjovsky et al. (2017) shows that the EM distance is better than the following metrics to quantify distances between probabilty distributions, namely

- the Total Variation metric, defined as

$$\delta(P_r, P_g) = \sup_A |P_r(A) - P_g(A)| \tag{11}$$

- The Kullback-Leibler divergence defined as

$$KL(P_r||P_g) = \int_x \log\Big(\frac{P_r(x)}{P_g(x)}\Big) P_r(x)dx \tag{12}$$

  It is important to note that $KL(P_r||P_g) \neq KL(P_g||P_r)$ (i.e. not symmetric)

- The Jensen-Shannon divergence. If we have P_m a mixture distribution with $P_m = \frac{P_r+P_g}{2}$, then Jensen-Shannon divergence is defined as

$$JS(P_r, P_g) = \frac{1}{2}KL(P_r||P_m) + \frac{1}{2}KL(P_g||P_m) \tag{13}$$

Arjovsky et al. (2017) uses the following example to explain the reelvance of the EM distance. If we consider probability distributions over $\mathbb{R}^2$. Let us consider the data distribution $(0, y)$ with, $y \sim U[0, 1]$. We consider the the family of distribution P_θ, where $P_\theta = (\theta, y)$ again with $y \sim U[0, 1]$. We want the distance metric to move closer to zero, as $\theta \to 0$. For such a distribution, this is how the different distances behave:

- Total Variation
$$\delta(P_0, P_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{14}$$

- Kullback-Leibler Divergence
$$KL(P_0||P_\theta) = KL(P_\theta||P_0) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{15}$$

- Jensen-Shannon Divergence
$$JS(P_0, P_\theta) = \begin{cases} \log(2) & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{16}$$

- the Earth mover distance
$$W(P_0, P_\theta) = |\theta| \tag{17}$$

This simple example shows that there exist distributions for which the Total Variation, the Kullback-Leibler Divergence and the Jensen-Shannon Divergence do not converge, whereas the Earth mover Distance does. Arjovsky et al. (2017) then confirms the insight provided by this example with the the two following theorems (without proof here, but the proofs can be found in Arjovsky et al. (2017))

Theorem 1: Let P_r be a fixed distribution. Let Z be a random variable. Let g_θ be a deterministic function parametrized by θ and let $P_\theta = g_\theta(Z)$. Then

1. If g is continuous in θ, then so is $W(P_r, P_\theta)$
2. If g is sufficiently nice, then $W(P_r, P_\theta)$ is continuous everywhere and differentiable almost everywhere.
3. Statements 1 and 2 do not hold for neither $JS(P_r, P_\theta)$, $KL(P_r||P_\theta)$ or $KL(P_\theta||P_r)$.

This first theorem shows that out of the four loss functions above-mentionned, only the Eart-Mover has some guarantees of both continuinity and differentiability, which are nice gurantees for a loss function.

Theorem 2: Let P be a distribution and $(P_n)_{n\in\mathbb{N}}$ be a sequence of distributions. The the following is true

1. The following statements are equivalent:
   - $\delta(P_n, P) \to 0$

- $JS(P_n, P) \to 0$

2. The following statements are equivalent:
   - $W(P_n, P) \to 0$
   - $P_n \to P$, with here "$\to$" being convergence in distribution for random variables.
3. $KL(P_n||P) \to 0$ or $KL(P||P_n) \to 0$ implying statement 1.
4. Statement 1 implies statement 2.

The above theorem proves that every distribution that converges under the Kullback-Leiber divergence (in any direction), the Jensen-Shannon divergence or the Total Variation also converges under the Earth-Mover distance. It also shows that a smaller difference between two probability distribution implies a smaller EM distance.

Unfortunately the EM distance is intractable, due to the infinum part in its equation. But using the Kantorovich-Rubinstein duality, it can be reformulated as:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \tag{18}$$

Where the supremum is over 1-Lipschitz functions. It is important to note that if we replace $||f||_L \leq 1$ by $||f||_L \leq K$, i.e. consider also the K-Lipschitz functions, then we obtain $K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta)$. Hence, for a family of functions $\{f_w\}_{w \in \mathcal{W}}$ (all functions being K-Lipschitz), we can consider solving the optimization problem:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z))] \tag{19}$$

Note: we can approximate the solution of the above mentionned problem with a Neural Network with weights $\mathcal{W}$. $\mathcal{W}$ need to be compact to assure that the function f_w are K-Lipschitz. Therefore in order to have $\mathcal{W}$ being compact, Arjovsky et al. (2017) proposes to simply clip the weights, such that they lay in a small box $[-c, c]^l$, e.g. with $c = 0.01$

2.6.2 Necessary changes to turn a GAN into a WGAN

Implementation of a WGAN requires a few changes from implementation of a regular GAN, i.e.

- Use a linear activation function in the output layer of the "discriminator" model (instead of sigmoid). The "discriminator" then becomes a critic that quantify the realness of a sample, instead of discriminating between real or fake.
- Use Wasserstein loss to train the critic and generator models that promotes larger difference between scores for real and generated images.
- Constrain critic model weights to a limited range after each mini batch update (e.g. [-0.01,0.01]). As seen above, this is to ensure, that the function estimated in the for approximting the Wasserstein distance are K-lipschitz, a necessary condition.

- Update the critic model more times than the generator each iteration (e.g. 5). Contrary as in a GAN, this is not a issue. Indeed, switching to the EM distance, allow for more stability when training the two networks in the WGAN. Moreover, the fact that the EM distance is continuous and differentiable means that we should train the critic until optimality.
- Use the RMSProp version of gradient descent with small learning rate and no momentum (e.g. 0.00005).

2.7 WGAN-GP

As we have seen above, WGAN improve the stability in training the critic ($\approx$ discriminator). But it is still subject to poor sample generation, convergence failure or mode collapse. This is due to the weight clipping happening while training the critic. In order to remedy to this, Gulrajani et al. (2017) propose to replace weight clipping by the introduction of a penalization of the norm of gradient of the critic with respect to its input. The issue is that trying to orient the critic to 1-Lipschitz function by weight clipping, biases the critic for too simple function. Gulrajani et al. (2017) observes that implementing the Lipschitz constraint using weights clipping leads to either exploding or vanishing gradient, unless the threshold c used for the clipping is carefully fine-tuned.

2.7.1 The gradient penalty

In order to enforce the Lipschitz constraint, Gulrajani et al. (2017) proposes to add a penalty term to the loss function. The loss function then becomes:

$$L = L' + P \tag{20}$$

where:

- Original loss function:
$$L' = \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g}[D(\tilde{\mathbf{x}})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r}[D(\mathbf{x})] \tag{21}$$
- Penalty:
$$P = \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}}[(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}) - 1||)^2] \tag{22}$$

The goal of the penalty is to enforce the 1-Lipschitz constraint. Indeed, by definition, a function is 1-Lipschitz if and only if its gradient norm is smaller or equal to 1 everywhere. It can be easily seen that the penalty is here to enforce this constraint. In order to make such a penalty tractable, a soft version of the penalty is considered, where the constraint is only enforced on a the gradient norm of a few random samples $\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}$.

The sampling distribution $\mathbb{P}_{\hat{\mathbf{x}}}$ is defined by sampling uniformly on on a line between a pair of points respectively sampled from $\mathbb{P}_r$ and $\mathbb{P}_g$. This was proven experimentally to give sufficiently good results.

In Gulrajani et al. (2017), the penalty coefficient λ was set always to 10 in all experiences done. No batch normalization was used in Gulrajani et al. (2017). They claim that batch normalization shifts the discriminator problem from trying to match a single input to a single output from trying to macth a batch input to a batch output. This makes the penalty invalid, since the penalization is perfomed with each input individually and batch normalization introduce correlation between samples. Instead of batch normalization, Gulrajani et al. (2017) recomnends layer normalizations.

2.8 Assessing performances of a WGAN or WGAN-GP compared to GAN

A problem with regular GANs (from Goodfellow et al. (2020)) is that it is really hard to know when a good model has been found, from looking only at the loss function. The quality of a model is typically assessed by visually looking at the generated sample and deciding if they are realistic or not. But Arjovsky et al. (2017) shows that in a WGAN, the Wasserstein loss is a meaningful loss function, that is correlated with the quality of the sample produced by the generator. This is not true in a regular GAN and is actually one of the main selling point of a WGAN over GAN.

Gulrajani et al. (2017) shows that this property of the WGAN still holds true for the WGAN-GP. The critic loss should typically start at zero, then drops and work its way back to zero. This is when the WGAN-GP has converged and is hence producing realistic samples.

3 Methods

We decided that it made sense to try to generate the Cross Spectral Matrix (CSM), as done in Gerstoft et al. (2020) and extend his work to create a network to generate CSM, conditionally on Direction of Arrival (DoA). Indeed, such a generative network would allow us to have an on-line generation of sufficiently realistic labeled data required to train a neural network for Source Characterization.

More specifically than generating a CSM, we thought it would make more sense to generate separately the eigenvalues and eigenvectors its eigendecomposition. A CSM $\hat{\mathbf{C}}$ can be decomposed as:

$$\hat{\mathbf{C}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H \tag{23}$$

where $\mathbf{V} = [\mathbf{v}_1^T, \ldots, \mathbf{v}_M^T] \in \mathbb{C}^{M\times M}$, $\mathbf{v}_i$ being the ith eigenvector and where $\mathbf{\Lambda} \in \mathbb{R}^{M\times M}$ is a diagonal matrix, where λ_{ii} is the ith eigenvalue, corresponding to the ith eigenvector.

Indeed, since we choose a Generative Adversial Approach, the data will be generated using two networks: a generator and a discriminator/critic. Those two network are competing against each other: the goal of the generator is to produce data realistic enough so that discriminator can not tell it is fake. The goal of the discriminator is to tell whether a given input is real or has been generated. Both the generator and the discriminator have to be trained simultaneously until convergence. A typical issue occuring during the training, is that the discriminator becomes too good at discerning real from fake sample and hence the generator does not improve anymore.

Generating the eigenvalues and eigenvectors instead of the CSM is done in order to help the generator. This allow to normalize all the eigenvectors before feeding them to the discriminator, whether they are real or generated. The eigenvalues can also be scaled the biggest of them is equal to one. This normalization process allows to reduce the size of the space of data to generate.

3.1 Data

3.1.1 Synthetic data

The goal of this thesis was to generate data as realistic as possible, by learning the distribution of Cross Spectral Matrices (CSM) of recorded pressure vectors. As mentionned in the introduction using only real measurment was not a feasible, since the number of measurement required for

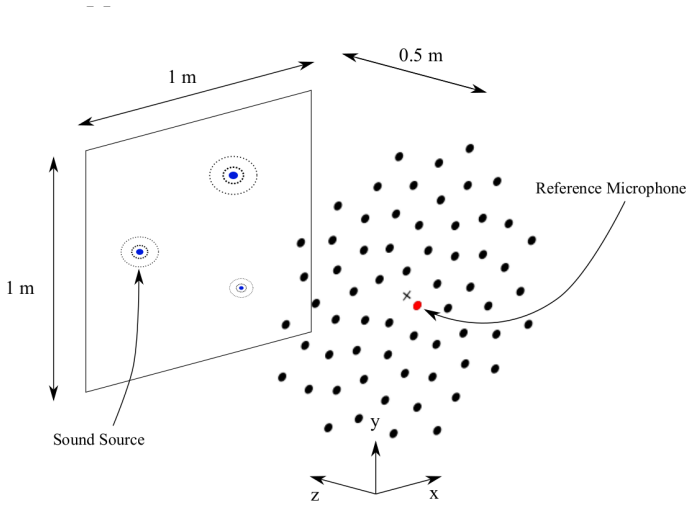

Figure 1: Example of the measurement setup used, with three audio sources

training is too big. For this reason, the chosen approach was to first pretrain the models using synthetic CSM and then fine-tune our model using real measurement. The synthetic CSM were sampled from the complex Wishart distribution using the acoular python package.

3.1.2 Measurement

In order to generate realistic data, measurement needed to be performed. An example of the measurement setup used is displayed in Fig.1. It is made of audio sources located in a 1m $\times$ 1m plane located half a meter in front an array of 64 microphones. The microphones are organized in a circular pattern, with the central one acting as the reference microphone. This means that when the CSM are computed, they are normalized using the autopower of the reference microphone. When performing the actual measurement, we only consider the case of a single audio source. The loudspeaker used in our measurement setup is displayed in Fig.2 and the microphone array in Fig.3.

The measurements are sound pressure $\mathbf{p} \in \mathbb{C}^{64}$ at the 64 microphones of the array. Every recording is for a different position of the audio source. All measurements have a duration 10s and a sampling frequency of 51.2kHz.

From one measurement, we wanted to create many different approximation of the CSM $\hat{\mathbf{C}}$ to have enough data to fine tune our generative model. To do so, we took slices of $\frac{1}{50}$ of total measurement length (i.e. 0.2s). From this slice, B snapshots needed to be created, for the approximation the CSM $\hat{\mathbf{C}}$ (see equation 3). From Gerstoft et al. (2012), we know that the number of snapshots B needed to be at least 4 times the number of receivers M of the array (i.e. at least $4 \cdot 64 = 256$). For this purpose we used snapshots overlapping at 75%. Indeed, Gerstoft et al. (2012) argues that such a number of snapshots is required, for the eigenvalues of the approximated CSM $\hat{\mathbf{C}}$ to be sufficiently close to the eigenvalues of the real CSM $\mathbf{C}$. Having snapshots overlapping at 75% allowed us to have $B = 317 > 256$.

Should something else be specified about the used microphones/loudspeaker?:

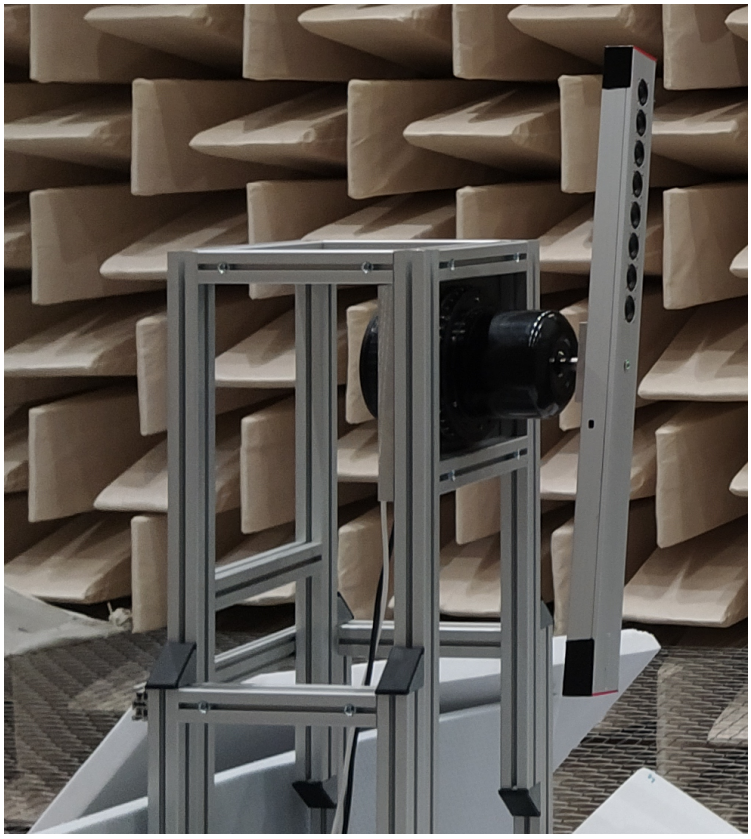

Figure 2: Device used as audio source to create the measurement

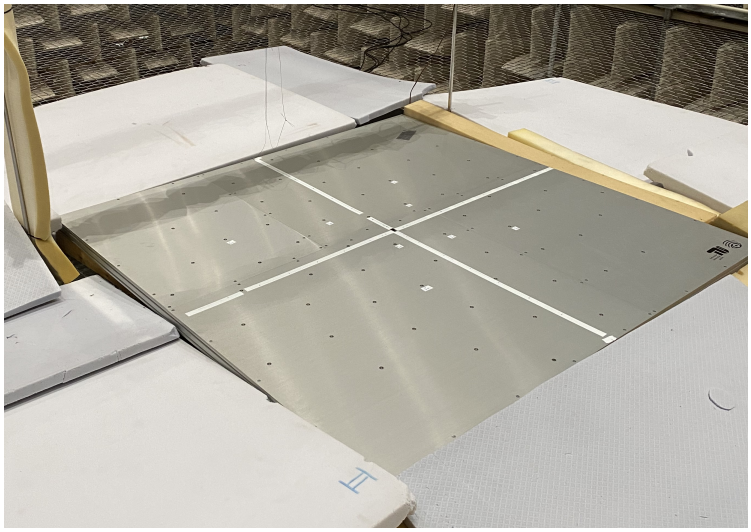

Figure 3: Picture of the array of microphone used to create the real measurement

3.1.3 Comparison between synthetic and measured data

In Fig.4, we show a comparison of the normalized eigenvalues of a synthetic CSM compared with a measured CSM. In the synthetic eigenvalues, we observe first that all the eigenvalues except the first ones are close to zero. When observing the eigenspectrum of the measured data, a slowlier decay can be observed. We conclude that the measured and synthetic data are indeed not fully similar and hence that data synthetized in the above mentionned way cannot replaced measured data.

This can be explained by the fact that the synthetic CSM are closed to the perfection of a mathematical model and hence do not show presence of natural noise created by reverberation for instance.

3.1.4 The data used in this thesis

When trying to generate CSM, we considered the case of a single source located in position on the plane in front of teh microphones array at position $[-0.09, -0.111]$. We considered CSM at Helmotz number $He = 8.74$. The Helmotz number He is used in commonly used in acoustic to represent frequency and is defined as

$$He = \frac{L_c}{\lambda} = L_c f \tag{24}$$

where L_c is the called the characteristic length. Here it is the speed of sound, hence $L_c = 343$ m/s λ is the wavelength under consideraation, and hence $f = \frac{1}{\lambda}$ is the frequency under consideration. First a measurement was chosen and using its source's position, we created the synthetic data. Example of beamforming maps resulting from CSMs from our datasest are displayed in Fig. **TODO: add beamforming map from synthetic data** and Fig. **TODO: add beamforming map from measurement data** (respectively from synthetic and measurement datasets.)

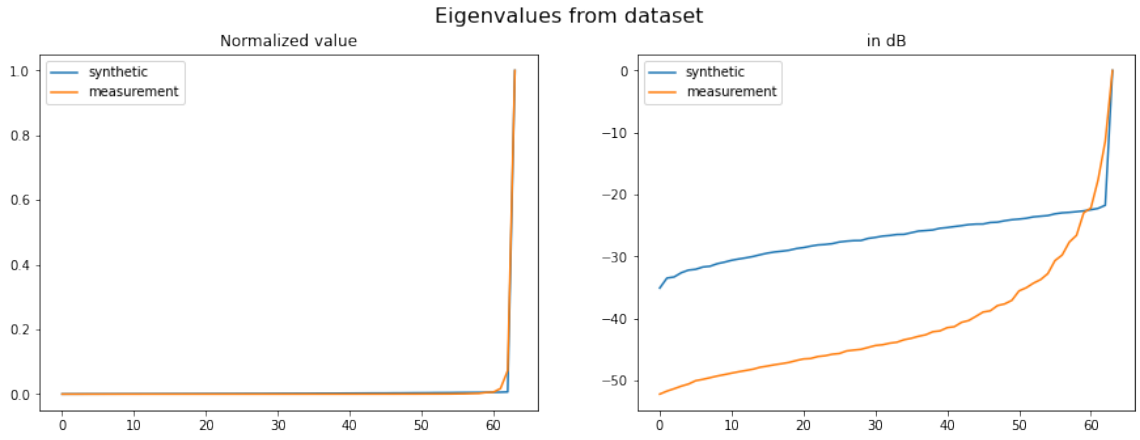

Figure 4: Comparison of the eigenvalues of CSMs approximated from synthetic data (blue) and from measured data (orange).

3.2 Generation of eigenvalues

In order to generate the eigenvalue, a WGAN-GP was built. The implementation was adapted from Nain (2020). Since the data to generate is the eigenvalues of CSMs of dimension 64×64, it is a vector $\lambda \in \mathbb{R}^{64}$.

3.2.1 Architecture

As mentionned in the fundamentals the basic structure is two competing networks, a generator and a discriminator or critic. Both architectures of the generator and critic are illustrated respectively in tables Tab.1 and Tab.2.

Unlike in the implementation of Nain (2020), both inner networks (generator and critic) do not have a convolutional but a perceptron structure. Indeed, a convolutional structure is relevant when the data to generate is an image. Indeed, a convolutional layer is good at seeing patterns in image, by identifying topological structures in neighbouring pixels. We could have stacked pieces of our vector of eigenvalues $\lambda = [\lambda_0, \ldots, \lambda_{63}] \in \mathbb{R}^{64}$ into an image of dimension 8×8 but this would have resulted in an image where most neighbour pixel are not sharing any information about each other, hence making the convolutional operation irrelevant at detecting pattern, at least compared to images.

Another changes that was made compared to the implementation of Nain (2020), is that a ReLU actvation function was added as the last layer of the generator. Indeed before this change our generator would generate eigenvalues slightly smaller than zero. Moreover, we added 1e-100 ((**TODO: check actual value**)) to every eigenvalue produced to ensure that they are positive.

Finally, before being fed to the discriminator, real and generated eigenvalues are scaled such that $0 \leq \lambda_i \leq 1$ for $\lambda_i \in \lambda$. Normalizing eigenvalues in this way is done to reduce the size of the sample space. This is illustrated in Fig.15.

Layer (type)	**Output Shape**	**Param #**
InputLayer	(None, 128)	0
Dense	(None, 256)	32768
LeakyReLU	(None, 256)	0
BatchNormalization	(None, 256)	1024
Dense	(None, 512)	131584
LeakyReLU	(None, 512)	0
Dense	(None, 1024)	525312
LeakyReLU	(None, 1024)	0
Dense	(None, 64)	65600

Table 1: Architecture of the generator used in the WGAN-GP to generate eigenvalues. Total params: 756,288, Trainable params: 755,776, Non-trainable params: 512

Layer (type)	**Output Shape**	**Param #**
InputLayer	(None, 64)	0
Dense	(None, 512)	33280
LeakyReLU	(None, 512)	0
Dense	(None, 256)	131328
LeakyReLU	(None, 256)	0
Dense	(None, 1)	257

Table 2: Architecture of the critic used in the WGAN-GP to generate eigenvalues. Total params: 164,865, Trainable params: 164,865, Non-trainable params: 0

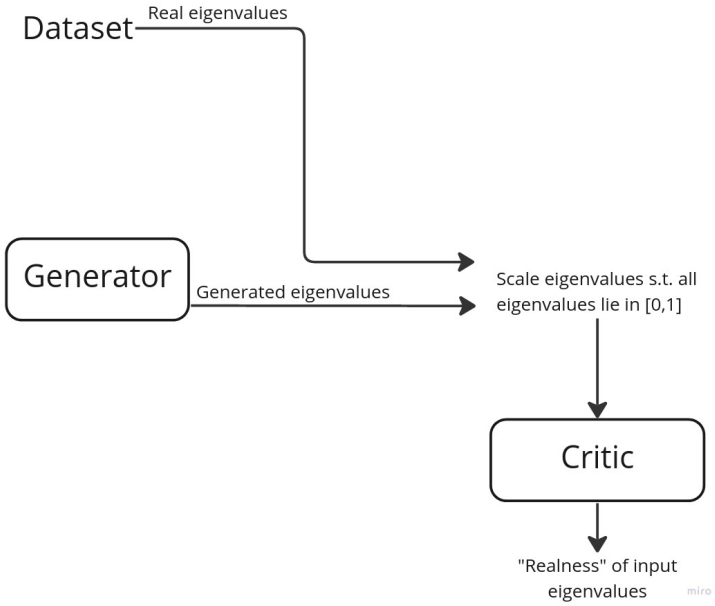

Figure 5: Full structure of the used implementation of the WGAN-GP to generate eigenvalues.

3.2.2 Results

In Fig.6 and Fig.7, we show the loss function when respectively training with the synthetic data and fine-tuing with measurement data. It can be observed that the model converged before the fine-tuning and that it did not have to change so much to adapt to generate measurement data, since the critic's loss function remain really close to zero. Moreover when comparing real and generated eigenvalues sample in Fig.8 and Fig.9 (respectively before and after the fine-tuning), it can be observed that our WGAN-GP seem to produce sufficiently realisitic results, both for synthetic and measurement data.

But when we compare the level of the eigenvalue in Fig.10, it can easily be observed that the WGAN-GP struggles at generating eigenvalues around zero. More specifically, it can be easily observe that if we sort the eigenvalues such that $\lambda_0 \leq \lambda_0 \leq \cdots \leq \lambda_{63}$, then $\exists j > 0$ s.t $\lambda_i = 10^{-10}$ $\forall i \leq j$. We remind that, by the way WGAN-GP has been implemented, 10^{-10} is the default minimum value. **TODO: write further**

3.3 Generation of eigenvalues from their level values

As seen above, the WGAN-GP create convincing spectrum, until they are displayed as level. This led to idea of generating the spectrum from its levels values. We remind here how the level of the eigenvalues $[\lambda_0, \ldots, \lambda_{63}] \in \Lambda$ are computed:

$$L_{\lambda_i} = 10 \log_{10}(\frac{\lambda_i}{\lambda_{63}}) \tag{25}$$

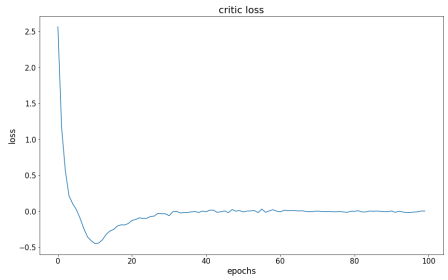

Figure 6: The loss function of the critic while performing initial training of the WGAN-GP for eigenvalues generation (before fine-tuning).

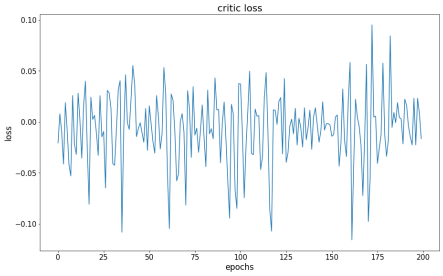

Figure 7: The loss function of the critic while fine-tuning the WGAN-GP for eigenvalues generation.

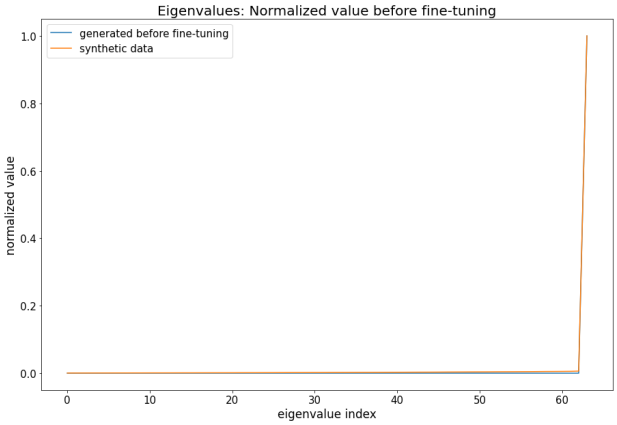

Figure 8: Comparision between real eigenvalues (orange) from the synthetic dataset with sample eigenvalues generated by our WGAN-GP (blue), when trained only with synthetic data (before fine-tuning)

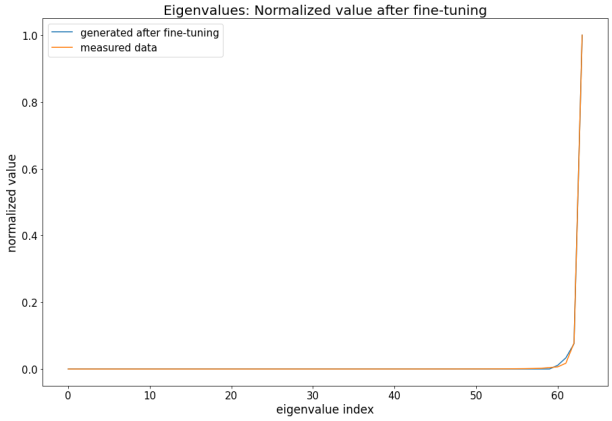

Figure 9: Comparision between real eigenvalues (orange) from the measurement dataset with sample eigenvalues generated by our WGAN-GP (blue), when trained further with measurement data (after fine-tuning)

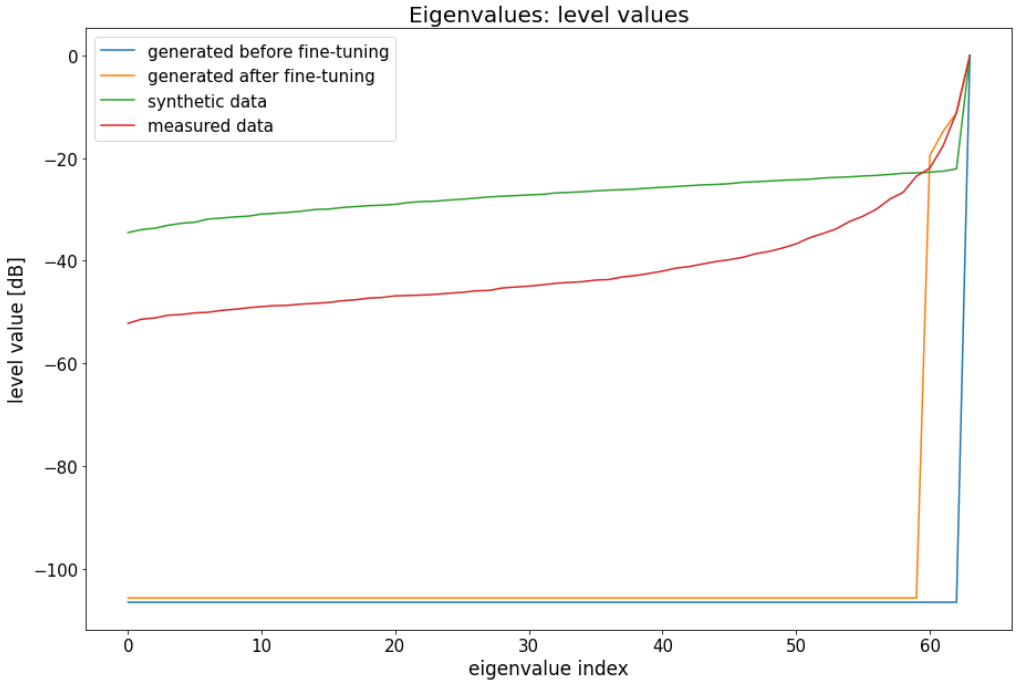

Figure 10: comparison level values of generated eigenvalues before fine-tuning (blue), generated eigenvalues after fine-tuning (orange) eigenvalues of synthetic CSM (green) and eigenvalues of CSM of measured data (red)

Layer (type)	**Output Shape**	**Param #**
InputLayer	(None, 128)	0
Dense	(None, 256)	32768
LeakyReLU	(None, 256)	0
BatchNormalization	(None, 256)	1024
Dense	(None, 512)	131584
LeakyReLU	(None, 512)	0
Dense	(None, 1024)	525312
LeakyReLU	(None, 1024)	0
Dense	(None, 64)	65600
ReLU	(None, 64)	0
Multiply	(None, 64)	0

Table 3: Architecture of the generator used in the WGAN-GP to generate eigenvalues from their level values. Total params: 756,288, Trainable params: 755,776, Non-trainable params: 512

Layer (type)	**Output Shape**	**Param #**
InputLayer	(None, 64)	0
Multiply	(None, 64)	0
Dense	(None, 512)	33280
LeakyReLU	(None, 512)	0
Dense	(None, 256)	131328
LeakyReLU	(None, 256)	0
Dense	None, 1	257

Table 4: Architecture of the critic used in the WGAN-GP to generate eigenvalues from their level values. Total params: 164,865, Trainable params: 164,865, Non-trainable params: 0

where λ_{63} is the biggest eigenvalue.

3.3.1 Architecture

The implementation was again inspired by Nain (2020). More specifically, we adapted the networks created for generating directly the eigenvalues. Since all values L_{λ_i} are non-positive, the generator has been built sucht that the two last steps are first a ReLU, followed by a multiplication by -1. This way we can ensure that our network only produces non-positive spectrum. This approach was also justified by the usage of Leaky ReLUs as activation throughout the generator.

Because we were also using Leaky ReLUs in the critic, the first layer of its network is also a multiplicative layer. Therefore, the critic is not trained to detect real from fake levels, but real from fake negative spectrum, which is equivalent.

The architecures used for the generator and the critic are shown respectively in Tab.3 and Tab.4.

3.3.2 Results

When training this new WGANGP, the losses in Fig.11 and Fig.12 can be observed (respectively before and after fine-tuning). It can be seen that our model converged before fine-tuning, and it

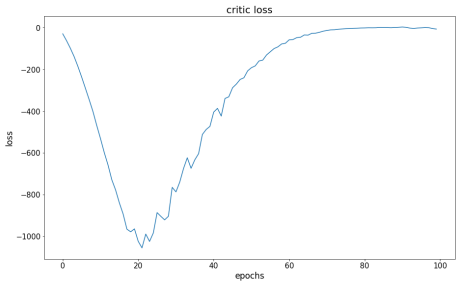

Figure 11: The loss function of the critic while performing initial training of the WGAN-GP for the levels of eigenvalues generation (before fine-tuning).

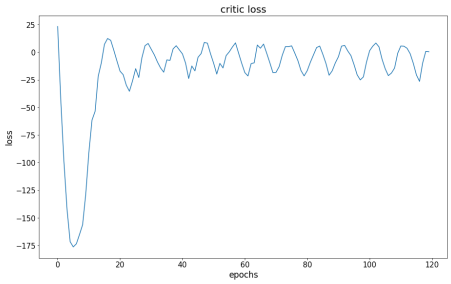

Figure 12: The loss function of the critic while fine-tuning the WGAN-GP for the levels of eigenvalues generation.

then had to adapt a bit during the fine-tuning but converged again.

In Fig.13, we show a comparison between real and generated levels of the eigenvalues of the CSM of synthetic data (before fine-tuning). We show in Fig.14 the same comparison but after fine-tuning the WGAN-GP. It can observed that the obtained level samples look already more realistic than when the network is not trained with levels value of eigenvalues (see Fig.10).

Finally, we show in Fig.15 a comparison between the eigenvalues of eigenvalues of the CSM of measurement and the eigenvalues converted back from their level values. It can easily be observed that generating eigenvalues from their level values yields higher quality samples than when generating them directly.

3.4 Generation of eigenvectors

In order to generate the eigenvector, we decide to start by only generating the eigenvector corresponding to the highest eigenvalue, that we define as main eigenvector. Indeed, using equation 5, we can already compute a Rank I CSM, allowing us to perform beamforming and locate identify source position. We show in Fig.16, an beamforming map, created with a rank I CSM obtained from measurement data.

Moreover, first generating only the main eigenvector can be justified by the fact, it is the most meaningful eigenvector. Indeed, each eigenvector belongs to a different incoherent source. If there would to be multiple coherent source, all the energy and their position merges into a single eigenmode. Since we consider the case where there is only one source, only the eigenvector with the biggest index corresponds to a DoA/source and the remaining 63 eigenvectors corresponds to noise.

In Fig.17, we plot the histograms of the values of the scalars of the eigenvectors with different index. It can be noticed that the values of the eigenvectors with low index seem to follow a gaussian distribution, whereas the values of the eigenvectors with index close to the maximal index are following a more complex distribution.

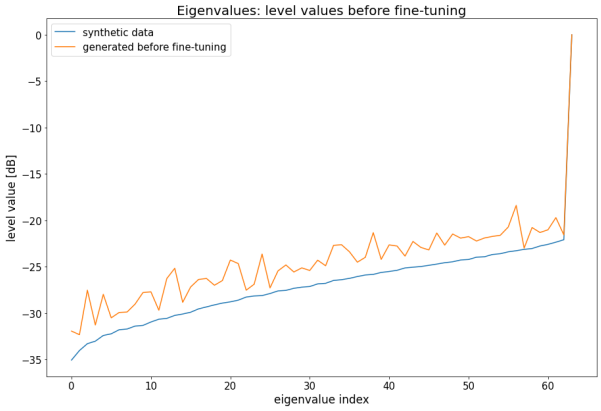

Figure 13: Comparision between the levels of real eigenvalues (blue) from the synthetic dataset with sample levels eigenvalues generated by our WGAN-GP (orange), when trained only with synthetic data (before fine-tuning)

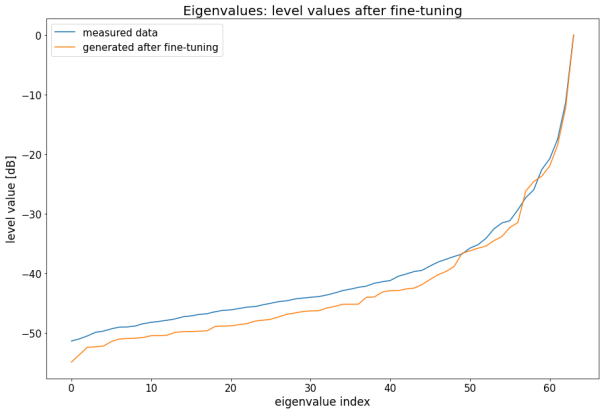

Figure 14: Comparision between the levels real eigenvalues (blue) from the measurement dataset with sample levels of eigenvalues generated by our WGAN-GP (orange), when trained further with measurement data (after fine-tuning)

3.5 WGAN-GP to generate the main eigenvector

3.5.1 Architecture

Similarly as for generating the eigenvalues, the WGAN-GP written to generate the main eigenvector was adapted from Nain (2020). Since the main eigenvector is $\in \mathbb{C}^{64}$, it could be generated using a similar architecture as the one used for the eigenvalues. The architectures both the generator and critic are illustrated respectively in the tables Tab.5 and Tab.6.

In order to reduce the sample space, both the real and generated main eigenvector are normalized before being fed to the discriminator. The full WGAN-GP structure is illutrates in Fig.18.

3.5.2 Results

As for the WAGN-GPs for generating eigenvalues, we train first this WGAN-GP with the main eigenvector of the CSM of synthetic data, and then fine-tune the network using measurement data. When training the network we observe the losses shown in Fig.19 and Fig.20 (respectively initial training and fine-tuning). We can see that the network has converged during the initial training, and had to adapt during the fine-tuing but also shows convergence.

Unlike for the eigenvalues, showing a generated sample of main eigenvector is not going to be really helpful in assessing the quality of the generated sample. Therefore, instead we do the following: generate a sample main eigenvector $\hat{\mathbf{v}}_{63}$ and use equation 5 to create a rank I CSM. It is important to note that the eigenvalue $\mathbf{\Lambda}_{63,63}$ corresponding to the main eigenvector is by definition equal to one. Indeed, it is by defintion the biggest eigenvalues and we scaled the eigenvalues such that all eigenvalues lie in $[0, 1]$. We plot in Fig.21 the beamforming map resulting from performing beamforming with the Rank I CSM created from a sampled main eigenvector. In Fig.22, we plot the

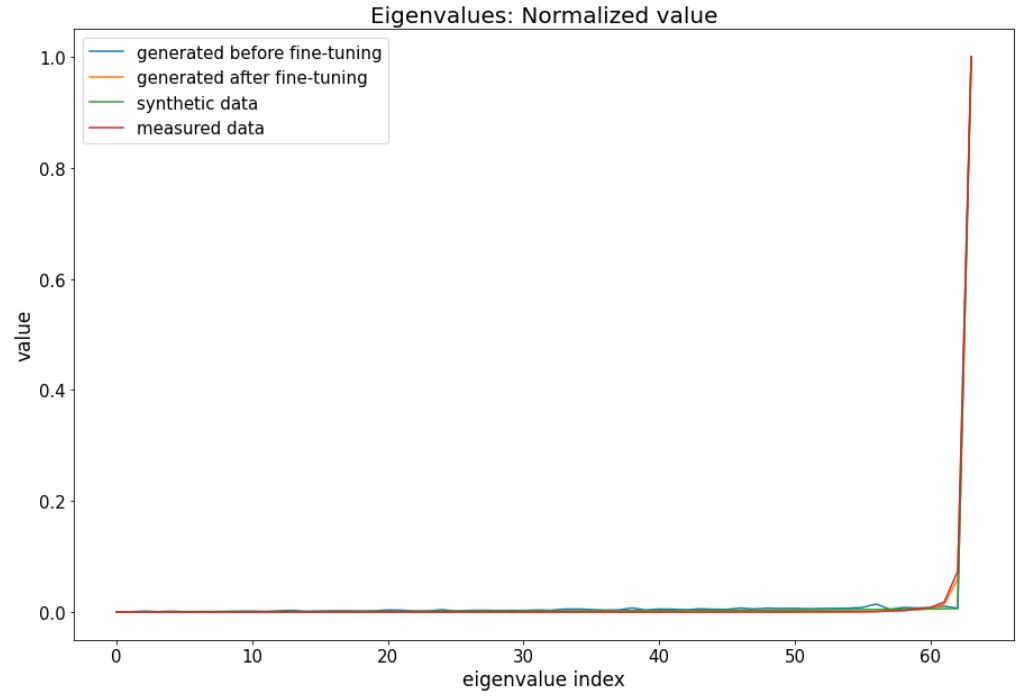

Figure 15: comparison between generated eigenvalues before fine-tuning (blue), generated eigenvalues after fine-tuning (orange), eigenvalues of synthetic CSM (green) and eigenvalues of CSM of measured data (red). The eigenvalues are computed back from their level values.

Layer (type)	**Output Shape**	**Param #**
InputLayer	(None, 128)	0
Dense	(None, 256)	32768
LeakyReLU	(None, 256)	0
BatchNormalization	(None, 256)	1024
Dense	(None, 512)	131584
LeakyReLU	(None, 512)	0
Dense	(None, 1024)	525312
LeakyReLU	(None, 1024)	0
Dense	(None, 128)	131200
Reshape	(None, 1, 64, 2)	0

Table 5: Architecture of the generator used in the WGAN-GP to generate the main eigenvector. Total params: 821,888, Trainable params: 821,376, Non-trainable params: 512

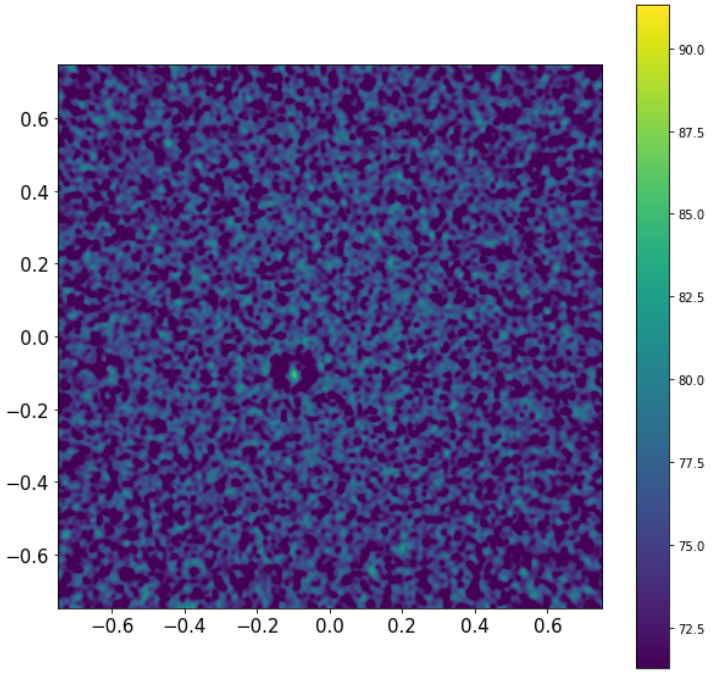

Figure 16: Beamforming map, created with a rank I CSM created from measurement data

Layer (type)	**Output Shape**	**Param #**
InputLayer	(None, 1, 64, 2)	0
Flatten	(None, 128)	0
Dense	(None, 512)	66048
LeakyReLU	(None, 512)	0
Dense	(None, 256)	131328
LeakyReLU	(None, 256)	0
Dense	(None, 1)	257

Table 6: Architecture of the critic used in the WGAN-GP to generate the main eigenvector. Total params: 197,633, Trainable params: 197,633, Non-trainable params: 0

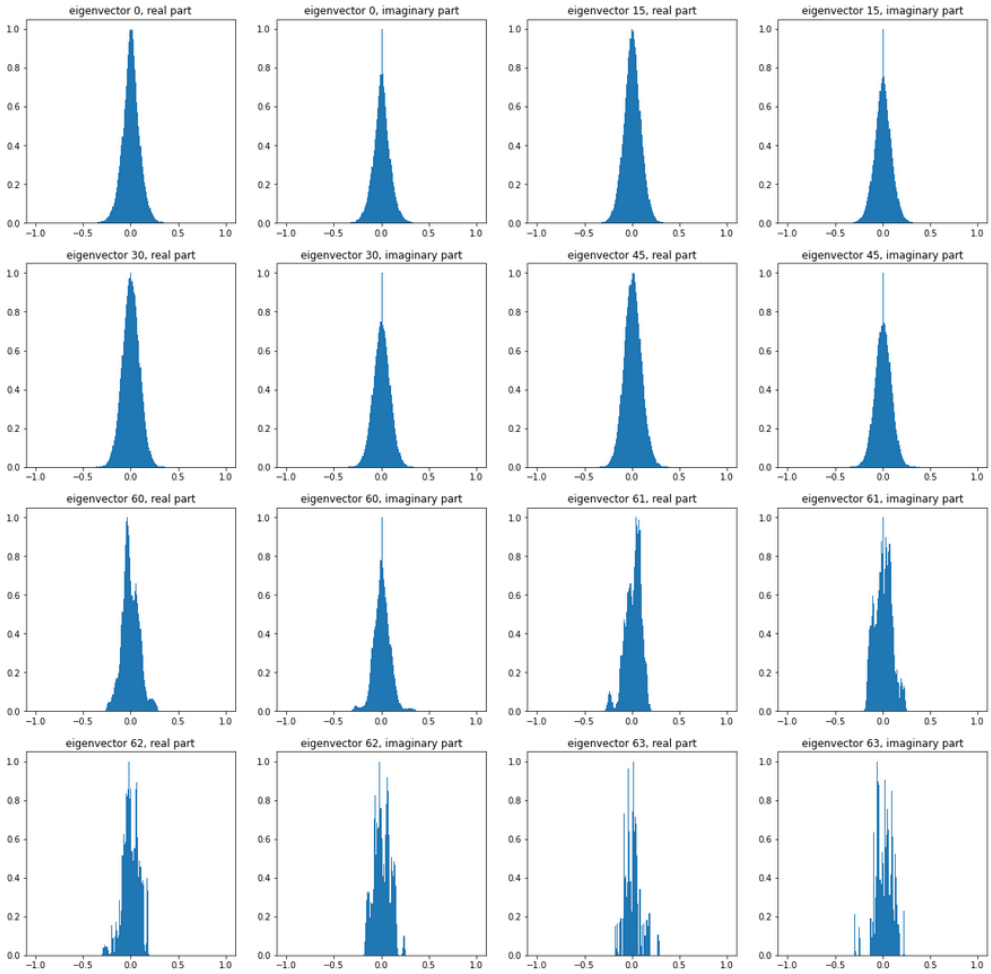

Figure 17: Histograms of the values of the scalars of the eigenvectors with different index. The eigenvector with index 63 is the main eigenvector (two last histograms).

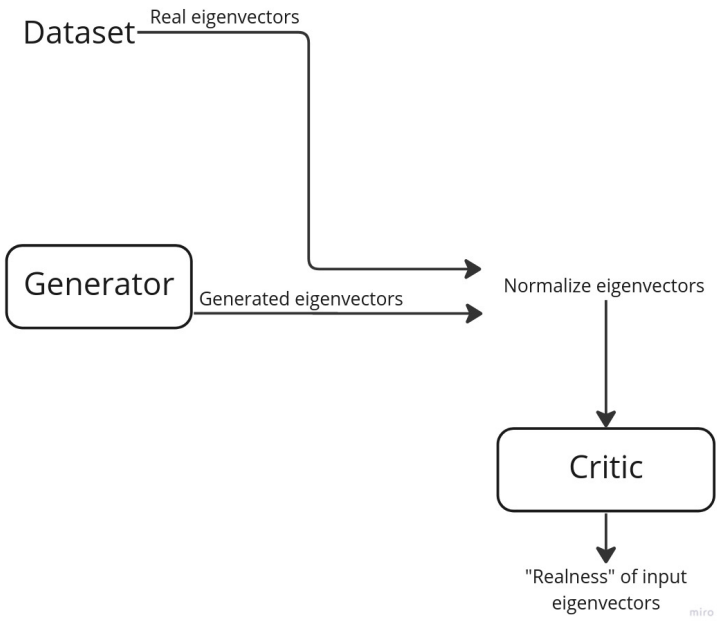

Figure 18: Full structure of the used implementation of the WGAN-GP to generate eigenvectors.

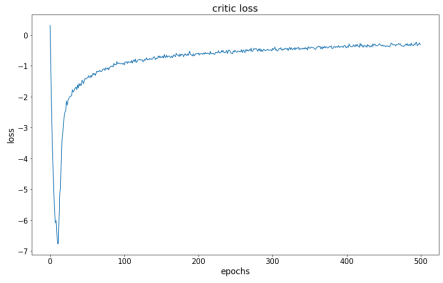

Figure 19: The loss function of the critic while performing initial training of the WGAN-GP for the main eigenvector (before fine-tuning).

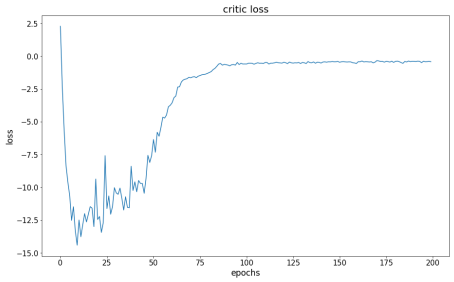

Figure 20: The loss function of the critic while fine-tuning the WGAN-GP for the the main eigenvector generation.

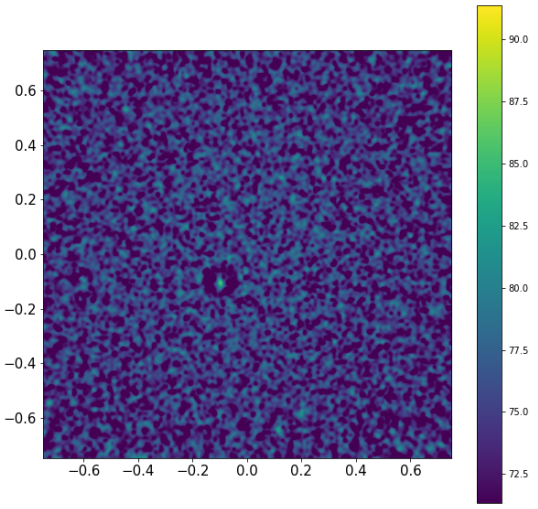

Figure 21: Beamforming map, created with a rank I CSM created from data generated with the WGAN-GP

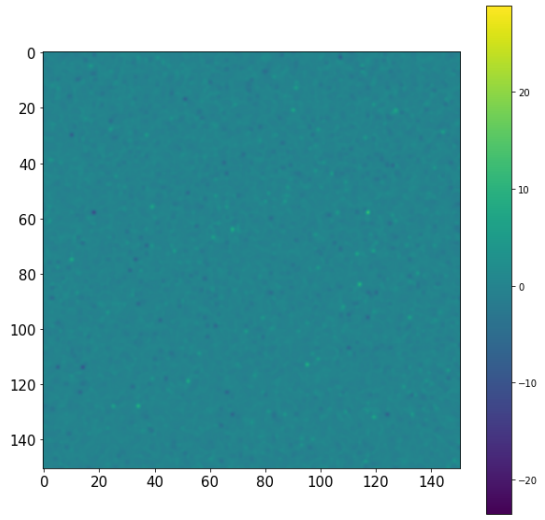

Figure 22: Difference between the beamforming map created rank I CSM from data generated with the WGAN-GP and the beamforming map created rank I CSM from measurement data

difference between a beamforming map computed with a generated main eigenvector and the beamforming map computed with a main eigenvector from the measurement dataset (i.e. map shown in Fig.16). We can observe that the difference of the two map is almost zero everywhere, showing us that our WGAN-GP is allowing us to generate sample close to the original main eigenvector, without being 100% similar.

3.6 WGAN-GP to generate the noise eigenvectors TODO: to see if it is only for noise eigenvectors, or for all evecs directly

3.6.1 Architecture

The approach used to generate the noise eigenvectors is similar as the main eigenvectors. The difference is that this time we wish to generate 63 vectors $\in mathbbC^6 4$. For this prupuse, we stack these 63 vector into a single vector $\in mathbbC^4 032$. Due to the fact that we wish to generate a vector significantly larger, we add to increase the latent dimension from 128 to 4096 **TODO: this value need to be checked when the performance have been assessed**.

The architecture of both the generator and critic used are respectively shown in Tab.7 and Tab.8. Similary as in the WGAN-GP for the main eigenvector, both real and generated eigenvector were normalized before being fed to the critic (see Fig.18)

3.6.2 Results

TODO: to write when working

Layer (Type)	**Output Shape**	**Param #**
InputLayer	(None, 128)	0
Dense	(None, 256)	32768
BatchNormalization	(None, 256)	1024
LeakyReLU	(None, 256)	0
Dense	(None, 1024)	262144
LeakyReLU	(None, 1024)	0
Dense	(None, 4096)	4194304
LeakyReLU	(None, 4096)	0
Dense	(None, 8064)	33030144
LeakyReLU	(None, 8064)	0
Reshape	(None, 63, 64, 2)	0

Table 7: Architecture of the generator used in the WGAN-GP to generate the noise eigenvectors. Total params: 37,520,384, Trainable params: 37,519,872, Non-trainable params: 512

Layer (Type)	**Output Shape**	**Param #**
InputLayer	(None, 63, 64, 2)	0
Flatten	(None, 8064)	0
Dense	(None, 4096)	33034240
LeakyReLU	(None, 4096)	0
Dense	(None, 512)	2097664
LeakyReLU	(None, 512)	0
Dense	(None, 256)	131328
LeakyReLU	(None, 256)	0
Dense	(None, 1)	257

Table 8: Architecture of the critic used in the WGAN-GP to generate the noise eigenvectors. Total params: 35,263,489, Trainable params: 35,263,489, Non-trainable params: 0

Layer (Type)	**Output Shape**	**Param #**
InputLayer	(None, 128)	0
Dense	(None, 4096)	524288
BatchNormalization	(None, 4096)	16384
LeakyReLU	(None, 4096)	0
Reshape	(None, 4, 4, 256)	0
UpSampling2D	(None, 8, 8, 256)	0
Conv2D	(None, 8, 8, 128)	294912
BatchNormalization	(None, 8, 8, 128)	512
LeakyReLU	(None, 8, 8, 128)	0
UpSampling2D	(None, 16, 16, 128)	0
Conv2D	(None, 16, 16, 64)	73728
BatchNormalization	(None, 16, 16, 64)	256
LeakyReLU	(None, 16, 16, 64)	0
UpSampling2D	(None, 32, 32, 64)	0
Conv2D	(None, 32, 32, 128)	73728
BatchNormalization	(None, 32, 32, 128)	512
LeakyReLU	(None, 32, 32, 128)	0
UpSampling2D	(None, 64, 64, 128)	0
Conv2D	(None, 64, 64, 2)	2304
BatchNormalization	(None, 64, 64, 2)	8
Activation	(None, 64, 64, 2)	0

Table 9: Architecture of the generator used in the WGAN-GP to generate snapshots used for creating an approximated CSM. Total params: 986,632, Trainable params: 977,796, Non-trainable params: 8,836

3.7 Generation CSM snapshots

Another approach to generate the Cross-Spectral Matrix is introduced in Gerstoft et al. (2020). In this paper, a method from generating the Cross Spectral Matrix by generating individually the snapshots is proposed. Even though, this method was not created in this thesis, it is worth mentionning and explaining more in detail.

From equation 3, we can see that an approximated of CSM is an averaging of B snaphsots $\mathbf{p}\mathbf{p}^H$. Gerstoft et al. (2020) creates a WGAN-GP to generate those snapshots. We reproduced this network:

3.7.1 Architecture

Once again the architecture used adapted from Nain (2020), but this time with Convolutional layers. The architecture of both the generator and critic are displayed respectively in Tab.9 and Tab.10

Layer (Type)	Output Shape	Param #
InputLayer	(None, 64, 64, 2)	0
ZeroPadding2D	(None, 68, 68, 2)	0
Conv2D	(None, 34, 34, 64)	3264
LeakyReLU	(None, 34, 34, 64)	0
Conv2D	(None, 17, 17, 128)	204928
LeakyReLU	(None, 17, 17, 128)	0
Dropout	(None, 17, 17, 128)	0
Conv2D	(None, 9, 9, 256)	819456
LeakyReLU	(None, 9, 9, 256)	0
Dropout	(None, 9, 9, 256)	0
Conv2D	(None, 5, 5, 512)	3277312
LeakyReLU	(None, 5, 5, 512)	0
Flatten	(None, 12800)	0
Dropout	(None, 12800)	0
Dense	(None, 1)	12801

Table 10: Architecture of the critic used in the WGAN-GP to generate snapshots used for creating an approximated CSM. Total params: 4,317,761, Trainable params: 4,317,761, Non-trainable params: 0

3.7.2 Result

3.8 Data Augmentation

As seen before, we have some quite realistic looking generated eigenvalues. This led to the following idea for increasing the number of training samples (data augmentation). First, for a set of received CSM, decompose them in eigenvalues $\mathbf{\Lambda}$ and eigenvectors $\mathbf{V} = [\mathbf{v}_1^T, \ldots, \mathbf{v}_M^T]$ with:

$$\hat{\mathbf{C}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H \tag{26}$$

The set of all eigenvalues can then be used to train a WGAN-GP. Using this network, we can generated eigenvalues $\hat{\mathbf{\Lambda}}$, which are realistic, up to an unknown scaling factor $c \in \mathbb{R}$.

Then using the generated eigenvalues $\hat{\mathbf{\Lambda}}$ with real eigenvectors $\mathbf{V}$ we can have semi-generated CSM:

$$\hat{\mathbf{C}}_{\text{augm.}} = \mathbf{V}\hat{\mathbf{\Lambda}}\mathbf{V}^H \tag{27}$$

which are realistic up to the scaling factor c. **TODO: show that the scaling factor c does influence the beamforming**.

3.8.1 Comparison between augmented data and real data.

Fig.23 and Fig.24 show respectively the result of performing beamforming on a real CSM issued from the dataset and from a semi generated CSM. Both CSM have been recreated from the eigenvalue decomposition. It can be observed that both results are visually similar. A more careful obserbartion allows to notice that despite looking similar proportinaly, the overall values of in the beamforming

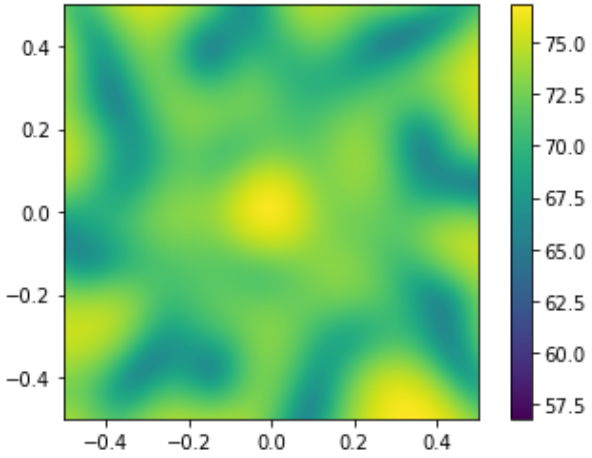

Figure 23: Results of beamforming from a real CSM

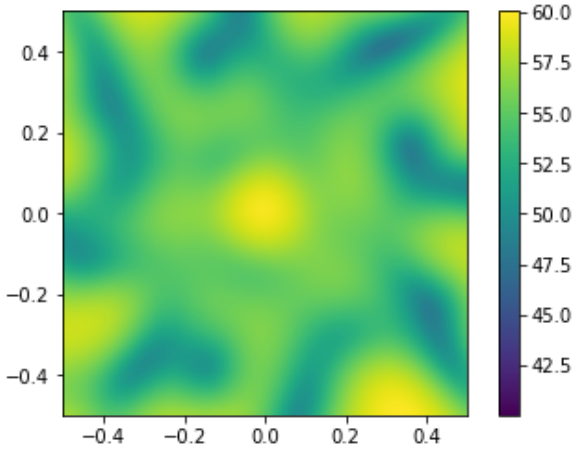

Figure 24: Results of beamforming from a CSM issued from augmented data

map resulting from real data are slightly higher. This is due to the scaling of the eigenvalues taking place in the generative process.

3.8.2 Comparison of different positions

4 Conclusion

TODO: To write at the end

5 Future work

TODO: here need to mention TransGAN, need to mention that the GAN could be made conditional, need to

References

Sharath Adavanne, Archontis Politis, and Tuomas Virtanen. Direction of arrival estimation for multiple sound sources using convolutional recurrent neural network. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 1462–1466. IEEE, 2018.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

Michael J Bianco, Sharon Gannot, and Peter Gerstoft. Semi-supervised source localization with deep generative modeling. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2020.

Paolo Castellini, Nicola Giulietti, Nicola Falcionelli, Aldo Franco Dragoni, and Paolo Chiariotti. A neural network based microphone array approach to grid-less noise source localization. *Applied Acoustics*, 177:107947, 2021.

Soumitro Chakrabarty and Emanuël AP Habets. Broadband doa estimation using convolutional neural networks trained with noise signals. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 136–140. IEEE, 2017.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

Eric L Ferguson, Stefan B Williams, and Craig T Jin. Sound source localization in a multipath environment using convolutional neural networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2386–2390. IEEE, 2018.

Peter Gerstoft, Ravishankar Menon, William S Hodgkiss, and Christoph F Mecklenbräuker. Eigenvalues of the sample covariance matrix for a towed array. *The Journal of the Acoustical Society of America*, 132(4):2388–2396, 2012.

Peter Gerstoft, Herbert Groll, and Christoph F Mecklenbräuker. Parametric bootstrapping of array data with a generative adversarial network. In *2020 IEEE 11th Sensor Array and Multichannel Signal Processing Workshop (SAM)*, pages 1–5. IEEE, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C.Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017. URL `http://arxiv.org/abs/1704.00028`.

Weipeng He, Petr Motlicek, and Jean-Marc Odobez. Deep neural networks for multiple speaker detection and localization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 74–79. IEEE, 2018.

Fabian Hübner, Wolfgang Mack, and Emanuël AP Habets. Efficient training data generation for phase-based doa estimation. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 456–460. IEEE, 2021.

Adam Kujawski, Gert Herold, and Ennes Sarradj. A deep learning method for grid-free localization and quantification of sound sources. *The Journal of the Acoustical Society of America*, 146(3): EL225–EL231, 2019.

Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/6804c9bca0a615bdb9374d00a9fcba59-Paper.pdf`.

Soo Young Lee, Jiho Chang, and Seungchul Lee. Deep learning-based method for multiple sound source localization with high resolution and accuracy. *Mechanical Systems and Signal Processing*, 161:107959, 2021.

Wei Ma and Xun Liu. Phased microphone array for sound source localization with deep learning. *Aerospace Systems*, 2(2):71–81, 2019.

Aakash Nain. Wgan-gp overriding model train step, May 2020. URL `https://keras.io/examples/generative/wgan_gp/`.

Paarth Neekhara, Chris Donahue, Miller Puckette, Shlomo Dubnov, and Julian McAuley. Expediting tts synthesis with adversarial vocoding. *arXiv preprint arXiv:1904.07944*, 2019.

Constantinos Papayiannis, Christine Evers, and Patrick A Naylor. Data augmentation of room classifiers using generative adversarial networks. *arXiv preprint arXiv:1901.03257*, 2019.

Lauréline Perotin, Romain Serizel, Emmanuel Vincent, and Alexandre Guérin. Crnn-based joint azimuth and elevation localization with the ambisonics intensity vector. In *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 241–245. IEEE, 2018.

Wagner Gonçalves Pinto, Michaël Bauerheim, and Hélène Parisot-Dupuis. Deconvoluting acoustic beamforming maps with a deep neural network. 2021.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Anton Ratnarajah, Shi-Xiong Zhang, Meng Yu, Zhenyu Tang, Dinesh Manocha, and Dong Yu. Fast-rir: Fast neural diffuse room impulse response generator. 10.48550. *arXiv preprint ARXIV.2110.04057*, 2021.

Ennes Sarradj. A fast signal subspace approach for the determination of absolute levels from phased microphone array measurements. *Journal of Sound and Vibration*, 329(9):1553–1569, 2010.

Ennes Sarradj. Three-dimensional acoustic source mapping with different beamforming steering vector formulations. *Advances in Acoustics and Vibration*, 2012, 2012.

Ryu Takeda and Kazunori Komatani. Sound source localization based on deep neural networks with directional activate function exploiting phase information. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 405–409. IEEE, 2016.

Elizabeth Vargas, James R Hopgood, Keith Brown, and Kartic Subr. On improved training of cnn for acoustic source localisation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:720–732, 2021.

Juan Manuel Vera-Diaz, Daniel Pizarro, and Javier Macias-Guarasa. Acoustic source localization with deep generalized cross correlations. *Signal Processing*, 187:108169, 2021.

Pengwei Xu, Elias JG Arcondoulis, and Yu Liu. Deep neural network models for acoustic source localization. In *Berlin Beamforming Conference*, 2021.