DIPLOMA THESIS

Generating functional connectivity matrices with Generative Adversarial Networks

István Hajdu

Supervisor: Dávid Légrády, PhD
Associate Professor
Institute of Nuclear Technology

BME
2019

MŰEGYETEM 1782

Diplomamunka feladat a Fizikus mesterképzési szak hallgatói számára

A hallgató neve: Hajdu István	specializációja: Fizikus MSc - orvosi fizika
A záróvizsgát szervező tanszék neve: BME NTI	

A témavezető neve: Dr. Légrády Dávid
- tanszéke: BME NTI
- beosztása: egy.docens
- email címe: legrady@reak.bme.hu

A kidolgozandó feladat címe: Funkcionális konnektivitás mátrixok generálása Generative Adversarial Network modellek segítségével

A téma rövid leírása, a megoldandó legfontosabb feladatok felsorolása:
Az elmúlt néhány évtized egyik legdinamikusabban fejlődő orvosi képalkotó módszere a Mágneses Rezonancia Képalkotás (MRI). A modalitás nagy előnye, hogy eltérő mérési eljárásokkal különféle kontrasztok érhetőek el, így egyes alanyokról számos eltérő anatómiai és funkcionális információ nyerhető. Az agyi képalkotás (neuroimaging) területén különösen népszerűek az MRI alapú kísérletek, mivel a módszer jó térbeli és egyre jobb időbeli felbontású funkcionális képalkotásra képes. Az fMRI adatokból egyes agyterületek közti kapcsolatok erősségére becsülhető, sőt a teljes agy kapcsolati hálózatát jellemző funkcionális konnektivitási mátrix is kiszámítható.
Az utóbbi években egyre növekvő alanyszámú (néhány száz fős) MRI kísérletek lehetővé tették gépi tanulási algoritmusok alkalmazását agyi felvételek kiértékelésében. A gépi tanulás egyik legfrissebb területe a mély tanulás (deep learning), illetve azon belül is a konvolúciós neurális hálók, melyek eredetileg kétdimenziós képek osztályozására lettek kidolgozva, azonban könnyen kiterjeszthetőek funkcionális konnektivitási mátrixok elemzésére is. A konvolúciós háló alapú osztályzás feltehetően tovább javítható az úgynevezett Generative Adversarial Network (GAN) modellek felhasználásával, amelyek képesek a meglévő tanító minták (konnektivitási mátrixok) halmazát további szimulált mintákkal feldúsítani.
A hallgató feladata a konvolúciós hálók és GAN-ok szakirodalmának széleskörű áttekintése, valamint a GAN módszer adaptálása szimulált konnektivitási mátrixok generálásához.

A záróvizsga kijelölt tételei:

Dátum:

Hallgató aláírása:	Témavezető aláírása*:	Tanszéki konzulens aláírása:	A témakiírását jóváhagyom (tanszékvezető aláírása):

*A témavezető jelen feladatkiírás aláírásával tudomásul veszi, hogy a BME TVSZ 145. és 146.§ alapján az egyetem a képzési célok megvalósulása érdekében a szakdolgozatok, illetve diplomamunkák nyilvánosságát tartja elsődlegesnek. A hozzáférés korlátozása csak kivételes esetben, a dékán előzetes hozzájárulásával lehetséges.

Alulírott **Hajdu István** a Budapesti Műszaki és Gazdaságtudományi Egyetem fizikus MSc szakos hallgatója kijelentem, hogy ezt a diplomamunkát meg nem engedett segédeszközök nélkül, önállóan, a témavezető irányításával készítettem, és csak a megadott forrásokat használtam fel.
Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból vettem, a forrás megadásával jelöltem.

Budapest, 2019. 06. 06.

……...
aláírás

Contents

1 Introduction

The most rapidly improving medical imaging modality of the last decades is Magnetic Resonance Imaging (MRI). With MRI one can perform measurements with different contrast, and also with different goals. Imaging of anatomy can be performed measuring proton density, or acquiring magnetization relaxation (T1 and T2 relaxation) maps. Functional imaging can be performed by measuring the blood-oxigen-level contrast (BOLD signal) in the brain either as a response to external stimuli, or in resting state. Functional connectivity matrices can be used to describe the relationship of resting-state activations of brain regions.

Machine Learning is another rapidly evolving field of computer sciences, beginning to intersect with a variety of fields of natural sciences. Some examples are bioinformatics, astronomy, and cognitive sciences. Machine Learning is knowingly a data-driven discipline, thus, with the increasing number of available fMRI measurements, application of such learning algorithms becomes relevant.

Machine Learning models called Convolutional Networks have been used on connectivity matrices to predict diagnosis of Autism Spectrum Disorder [1], Amnestic Mild Cognitive Impairment [2], and to estimate brain age [3]. The goal of this thesis is to develop methods to generate convincing synthetic data with special generative models, called Generative Adversarial Networks (GANs) [4].

First, we will cover the theoretical basis of resting state fMRI connectivity matrices. Then, we dive into the details of the classification algorithms, followed by the theory of Generative Adversarial Networks. We explain the difficulties of properly training GANs, and offer a solution to this problem, trough evaluating thorough experiments.

The repository of our code is available online:
https://github.com/hajduistvan/connectome_gan.

2 Introduction to fMRI connectivity matrices

In the following sections, we will cover the basic principles used in functional MRI, assuming that the reader is familiar with magnetic resonance imaging. After that, we discuss resting-state fMRI, and how the functional connectivity matrices can be derived from such measurements.

Unlike in anatomical imaging, where the image is constructed by measuring proton density and relaxation times, a functional MRI image is aquired by measuring the so-called blood-oxygenation-level dependent (BOLD) signal. The mechanism is as follows: since the neurons do not have oxygen and glucose reservoirs, neural activity induces a reaction in the nearby blood vessels, releasing more oxygen to them than to inactive neurons. Since oxygenated hemoglobin is diamagnetic, and deoxygenated hemoglobin is paramagnetic, this change of magnetic susceptibility can be measured by the MR scanner [5]. The most commonly used fMRI sequence is the EPI, whose description is out of the scope of this thesis. The BOLD signal can be seen on Figure 1.

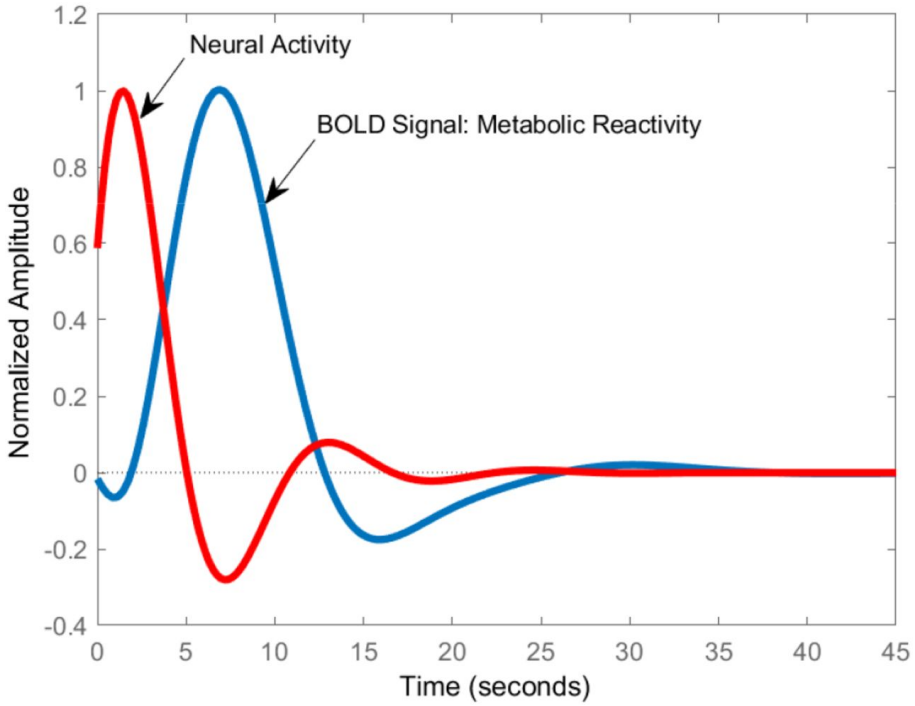


Figure 1: The BOLD signal and neural activity [6].

In event-related fMRI, the patient is given a particular task, and our goal is to gain knowledge about the brain's activity during this task. In resting-state fMRI, however,

no task is given to the patient. During this measurement, we can identify aspects of intrinsic brain activity, meaning synchronous BOLD signal changes in multiple brain regions. These brain regions can be organized into network modes, which consists of strongly correlated brain regions. Figure 2. shows some examples of these networks.

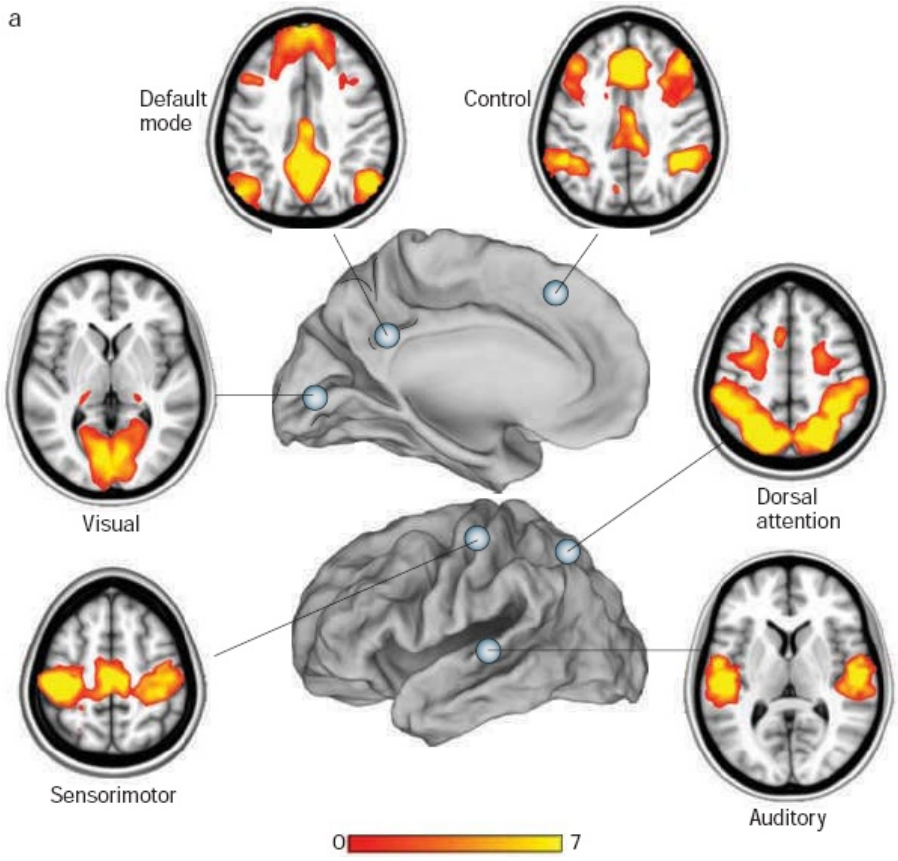


Figure 2: Taking a seed region denoted with blue circles, we can generate correlation maps of brain regions that share similar neuronal activity to that of the seed. Six of the major networks can be seen here: visual, sensorimotor, auditory, default mode, dorsal attention, and executive control. The scale numbered 0–7 indicates relative correlation strength. [7]

In order to reduce resting state fMRI time series to connectivity matrices, generally, we need to take the following steps:

1. Acquire fMRI time series
2. Define ROIs based on a brain atlas: either single brain regions (e.g. based on Harward-Oxford cortical and subcortical structural atlas [8]), or a network of brain regions

3. Average BOLD signal in each ROI for every timestep

4. Calculate time-dependent correlation metric of choice (e.g. full normalized temporal correlation) for each pair of regions

5. Use correlation coefficients as corresponding matrix elements

In Figure 3, we can see an example for the resulting connectivity matrices. In short, we can view connectivity matrices as a way to compress functional connectivity measurements into a small, intuitive, and algorithmically easy-to-handle form.

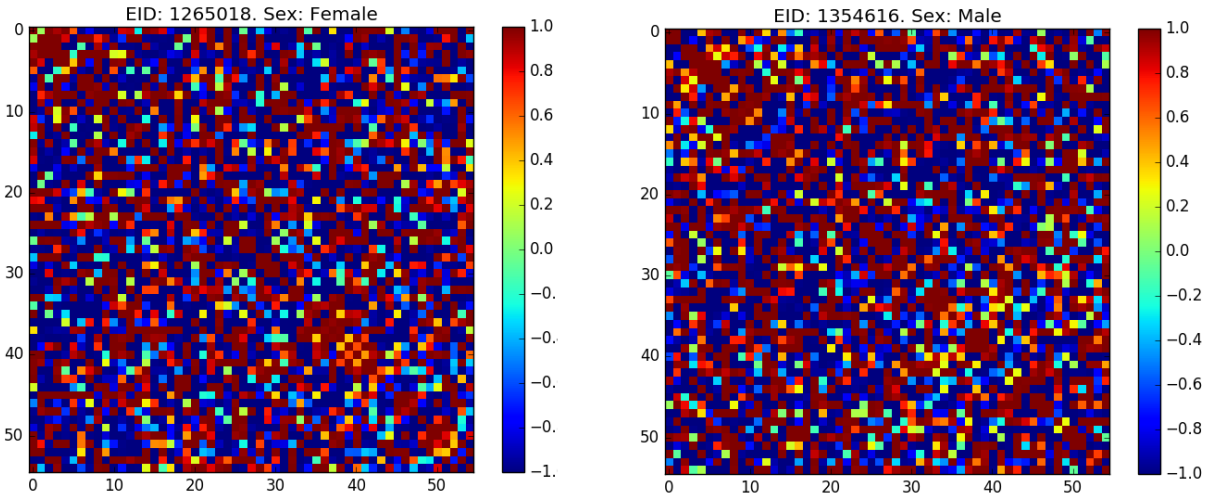


Figure 3: Connectivity matrix examples from our dataset, where a female (left), and a male (right) patient's connectivity can be seen. Both matrices are constructed from 55 ROIs.

The technical details of connectivity matrix acquisition is left to Section 4.1, since it is applicable only to those who use the same dataset, and this way omitted from the general discussion. It is important to mention that the goal of this thesis is not to discuss the origins and neurological aspects of these matrices, but to use them as inputs to our algorithms. Thus, the introduction to the connectivity matrices was kept brief, tough the referenced sources may supply with additional knowledge.

In the following sections, we dive into the main driver topics of this thesis, neural networks.

3 Introduction to Generative Adversial Networks

In this section we dive into the theoretical foundations of my thesis. First, we cover Neural Networks (NNs) and their most used application: classification. After that, we cover the basic theory of General Adversarial Networks (GANs), including the issues regarding to their training.

3.1 Neural Network paradigm

Machine Learning has gained immerse popularity with the rediscovery of Neural Networks. In the 1980's, Hinton et. al. proposed back-propagation[9] as an optimization method for parametric models, that is used until today in Deep Learning. In short, Neural Networks:

1. Represent a parametric, differentiable function $f(\boldsymbol{x}, \boldsymbol{w})$
2. Are a composition of simple functions, called layers:

$$f(\boldsymbol{x}, [\boldsymbol{w}_1, ..., \boldsymbol{w}_N]) = (f_N \circ ... \circ f_1)(\boldsymbol{x})$$

In orded to optimize Neural Networks, a loss function must be defined, which measures the goodness of the network's output. The optimization happens via Stochastic Gradient Descent (SGD), minimizing L:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \mu \frac{\partial \left(\frac{1}{M} \sum_{i=1}^{M} L\left(f(\boldsymbol{x}_i, \boldsymbol{w}^{(t)}) \right) \right)}{\partial \boldsymbol{w}^{(t)}}, \tag{1}$$

where $\mu \geq 0$ is called the learning rate. The average of the loss values over M examples covers the stochastic part of SGD, meaning we calculate the loss over a batch of samples sized M. In each iteration, a new batch of inputs is used.

The most common layers used to construct NNs fall into 3 categories:

1. Linear functions with trainable parameters ($\boldsymbol{w}$): these layers are responsible for learning the proper transformations to achieve the appropriate output. Two of the most widely used such layers:

   Fully Connected layer: matrix multiplication of the weights and the inputs: $\boldsymbol{x}_{j+1} = \boldsymbol{W}_j \boldsymbol{x}_j$. They can represent arbitrary linear transformation of their

inputs, at the cost of having many trainable parameters, usually more than necessary.

Convolutional layer: convolution of the inputs and the weigths as kernel: $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j * \boldsymbol{W}_j$. They are best for structured, grid-like data, like images. Parameter sharing over the image means fewer parameters need to be trained. Good when we have á priori information about the dependencies of the pixels, e.g. in images only a small local area is needed to process per layer to find meaningful features.

2. Nonlinear functions that have no trainable parameters: they are needed so the whole NN has higher complexity than linear functions. Usually used after each layer.

3. Normalizing and regularizing layers: They are used to aid optimization. We do not use these in out work.

3.2 Classification and Regression with Neural Networks

Supervised learning tasks involve labeled input data denoted $(\boldsymbol{x}_i, y_i)$, and the NN is trained to match its input to the corresponding label. For sample i the network's prediction is $f(\boldsymbol{x}_i, \boldsymbol{w}) = \hat{y}_i$. Two branches of supervised learning are classification and regression. In regression, labels are real numbers. In this case, one of the appropriate loss functions are the mean squared error (MSE):

$$L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 \tag{2}$$

In classification, on the other hand, the labels are discrete categories: $y_i \in 0, 1, ..., S$. In practice, these labels are one-hot encoded. This means that the label is a vector full of zeroes except at the index corresponding to the label's category, for example: $y_i = s \iff \boldsymbol{y}_{ij} = \delta_{js}$. From now, we can interpret $\boldsymbol{y_i}$ as a categorical probability density with deterministic outcome s_i. If we construct our classificator NN to output a categorical density as label prediction, we can measure the distance between the predicted and real labels as the cross-entropy of two probability distributions:

$$L(\boldsymbol{y_i}, \hat{\boldsymbol{y}}_{\boldsymbol{i}}) = -\sum_{j=0}^{S} \boldsymbol{y_{ij}} \log \hat{\boldsymbol{y}}_{\boldsymbol{i}j} \tag{3}$$

3.3 Generative Adversarial Networks

Unlike supervised learning, unsupervised learning is a task where we use NNs regarding the $p(\boldsymbol{x})$ distribution of the data. This can include clustering, or more relevant to us, generative models. Generative models are used to estimate said $p(\boldsymbol{x})$ distribution, and provide ways to artificially generate elements of this distribution. One popular approach to this problem is called the Generative Adversarial Network (GAN).

Goodfellow et. al. [4] proposed this type of network to be composed of two competing sub-networks:

> **Generator**: denoted as $g(\boldsymbol{z}, \boldsymbol{w_g})$, the generator is a neural network that takes a latent random variable $\boldsymbol{z}$ and outputs an artificial sample $\hat{\boldsymbol{x}}$:
>
> $$g(\boldsymbol{z}, \boldsymbol{w_g}) = \hat{\boldsymbol{x}}$$
>
> **Discriminator**: denoted as $d(\boldsymbol{x}, \boldsymbol{w_d})$ it is also a neural network that takes samples as input and outputs the probability that the input came from the data distribution rather than from the generator:
>
> $$d(\boldsymbol{x}, \boldsymbol{w_d}) = P(\boldsymbol{x} \in X_{data})$$

During training, d is optimized to give good estimates whether the data is real or generated, while the generator is trained to "fool" the discriminator. This leads to a 2-player min-max objective:

$$\max_{\boldsymbol{w_d}} \min_{\boldsymbol{w_g}} L(\boldsymbol{w_g}, \boldsymbol{w_d}) = \log(d(\boldsymbol{x}, \boldsymbol{w_d})) + \log(1 - d(g(\boldsymbol{z}, \boldsymbol{w_g}), \boldsymbol{w_d})) \tag{4}$$

For clarity regarding practical implementation via SGD see algorithm 1:

- Select hyperparameters k, μ_g, μ_d M, λ
- Initialize w_g and w_d weights

for N_{it} steps **do**

  for k steps **do**

    - Sample batch of noise samples $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_M\}$ from noise prior $p_g(\boldsymbol{z})$
    - Sample batch of real data samples $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_M\}$ from $p_{data}(\boldsymbol{x})$
    - Update discriminator weights to maximalize L:

$$\boldsymbol{w}_{\boldsymbol{d}}^{(t+1)} = \boldsymbol{w}_{\boldsymbol{d}}^{(t)} + \mu_d \nabla_{\boldsymbol{w}_{\boldsymbol{d}}^{(\boldsymbol{t})}} \frac{1}{M} \sum_{i=1}^{M} \left[\log d(\boldsymbol{x}_i, \boldsymbol{w}_{\boldsymbol{d}}^{(\boldsymbol{t})}) + \log \left(1 - d(g(\boldsymbol{z}_i, \boldsymbol{w}_{\boldsymbol{g}}^{(t)}), \boldsymbol{w}_{\boldsymbol{d}}^{(t)}) \right) \right] \tag{5}$$

  end

  - Sample batch of noise samples $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_M\}$ from noise prior $p_g(\boldsymbol{z})$
  - Update generator weights to minimalize L:

$$\boldsymbol{w}_{\boldsymbol{g}}^{(t+1)} = \boldsymbol{w}_{\boldsymbol{g}}^{(t)} - \mu_g \nabla_{\boldsymbol{w}_{\boldsymbol{g}}^{(\boldsymbol{t})}} \frac{1}{M} \sum_{i=1}^{M} \log \left(1 - d(g(\boldsymbol{z}_i, \boldsymbol{w}_{\boldsymbol{g}}^{(t)}), \boldsymbol{w}_{\boldsymbol{d}}^{(t)}) \right) \tag{6}$$

end

Algorithm 1: SGD training of a vanilla GAN. The number of discriminator cycles over generator cycles, k, is a hyperparameter, commonly $k = 10$. Other hyperparameters are learning rates μ, batch size M, and number of iterations N_{it}.

From (4) it can be shown that the optimal discriminator output, for a fixed generator is

$$d(\boldsymbol{x}, \boldsymbol{w}_{\boldsymbol{d}}^{*}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}, \tag{7}$$

where $p_g(\boldsymbol{x})$ is the generator's output distribution. In the case where this is true, the generator's objective can be rewritten as:

$$
\begin{aligned}
L(\boldsymbol{w_g}) &= L(\boldsymbol{w_g}, \boldsymbol{w_d^*}) \\
&= \log d(\boldsymbol{x}, \boldsymbol{w_d^*}) + \log(1 - d(g(\boldsymbol{z}, \boldsymbol{w_g}), \boldsymbol{w_d^*})) \\
&= \log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} + \log \frac{p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \\
&= -\log(4) + D_{KL}\left(p_{data} \,\middle\|\, \frac{p_{data} + p_g}{2}\right) + D_{KL}\left(p_g \,\middle\|\, \frac{p_{data} + p_g}{2}\right) \\
&= -\log(4) + JSD(p_{data} \parallel p_g),
\end{aligned}
\tag{8}
$$

where D_{KL} is the Kullback-Leibner divergence:

$$
D_{KL}(P \parallel Q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}, \tag{9}
$$

and JSD is the Jensen-Shannon divergence:

$$
JSD(P \parallel Q) = \frac{1}{2} D_{KL}\left(P \,\middle\|\, \frac{P+Q}{2}\right) + \frac{1}{2} D_{KL}\left(Q \,\middle\|\, \frac{P+Q}{2}\right). \tag{10}
$$

Therefore, we can summarize GAN training as:

1. The discriminator is trained to approximate the Jensen-Shannon divergence of real and generated data.

2. The generator is trained to minimize the JSD by providing better data estimations.

3.4 Wasserstein GAN

The issue with using Jensen-Shannon divergence as our metric is that for distant probability distributions, the gradient os JSD vanishes. Consider the task when we wish to approximate the mean of a gaussian distribution by minimalizing JSD. In Figure 4, we take two gaussian distribution, both with standard deviation 1, and the difference of their means is $\Delta\mu$. We calculate JSD and its derivative with respect to $\Delta\mu$:

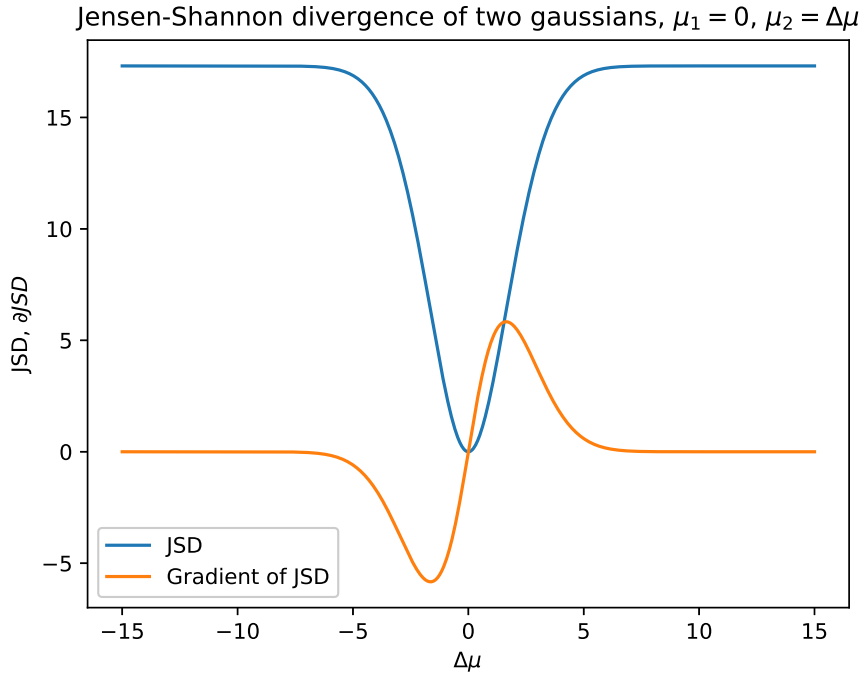


Figure 4: The gradient of Jensen-Shannon divergence with respect to $\Delta\mu$ quickly reaches zero, rendering training obsolete.

Since we train our generator with gradient descent, the disappearance of the gradient is a fatal problem. The only case when the gradient is not 0 is when the probability densities has overlapping supports, where both are non-zero. Unfortunately, in Deep Learning, such initialization is very rare.

Arjovsky et al.[10] proposed a new probability metric to replace JSD, called Wasserstein or Earth Mover distance. EM (or W) distance is defined as

$$W(P,Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right], \tag{11}$$

where $\Pi(P,Q)$ is the set of all possible joint distributions $\gamma(x,y)$ whose marginals are P and Q. Intuitively, $\gamma(x,y)$ indicates how much "mass" should be transported from x to y in order to transform P to Q. The EM distance is the "cost" of the optimal transport plan.[10] While this is numerically intractable, with the Kantorovich-Rubinstein duality we can rewrite (11) as

$$W(P,W) = \sup_{\|d\|_L \leq 1} \mathbb{E}_{x\sim P} \left[d(x) \right] - \mathbb{E}_{x\sim Q} \left[d(x) \right], \tag{12}$$

where the supremum is taken over all the 1-Lipsitz functions $d : X \to \mathbb{R}$. Figure 5. illustrates that this distance metric solves the vanishing gradient problem on the previous example.

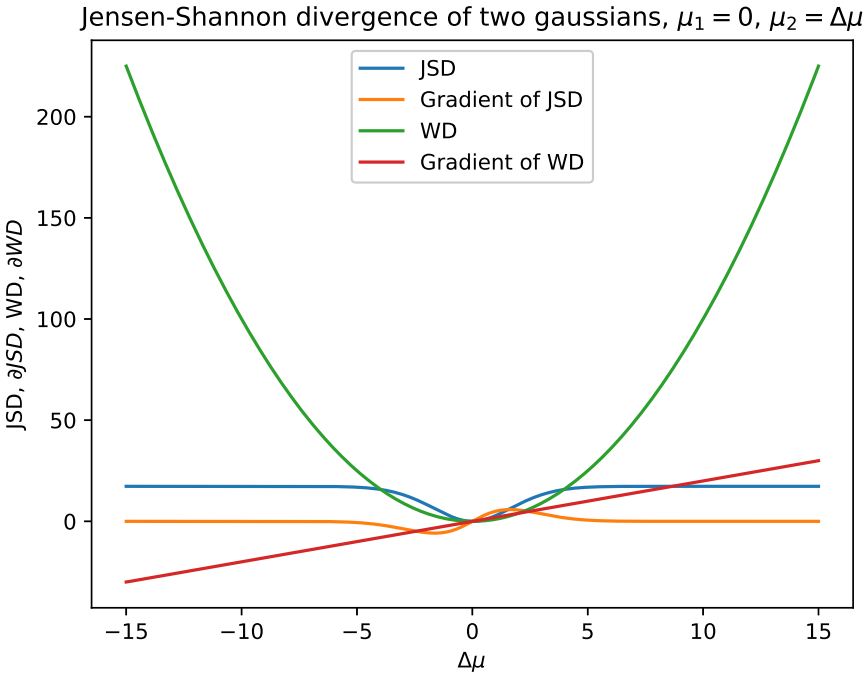


Figure 5: Unlike JSD, Wasserstein Distance (WD) has clear non-zero gradients everywhere, except at perfect convergence.

For our application, we define P and Q as the probability distributions of real and generated samples, and d as the discriminator neural network. This way, the discriminator is trained to estimate the Wasserstein Distance of the real and generated samples, while the generator is trained to minimalize this distance. We will later refer to the Wasserstein Distance as the Wasserstein Loss, since this is the training objective for both the generator and the discrimiantor.

However, now we need to find a way to enforce the 1-Lipsitz criterium on the discriminator. By definition, for a 1-Lipsitz function

$$\|f(x) - f(y)\| \leq \|x - y\| \tag{13}$$

holds for all x and y. For a neural network and all of it's possible imputs, this is an intractable condition. Gulrajani et. al.[11] found a way to give an approximate solution.

The idea is that the loss function of the discriminator is extended with a term called Gradient Penalty. In short, the discriminator is evaluated at random linear interpolations of real and generated samples, and at this point, the gradient of the discriminator with respect to its input is enforced to move towards 1. Mathematically:

$$GP = \lambda \mathbb{E}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\tilde{\boldsymbol{x}}}} \left[\left(\left\| \nabla_{\tilde{\boldsymbol{x}}} d(\tilde{\boldsymbol{x}}) \right\|_2 - 1 \right)^2 \right] \tag{14}$$

Finally, our training algorithm for Wasserstein-GANs (WGANs) are as follows:

- Select hyperparameters k, μ_g, μ_d M, λ
- Initialize w_g and w_d weights

for N_{it} steps **do**

  for k steps **do**

    - Sample batch of noise samples $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_M\}$ from noise prior $p_g(\boldsymbol{z})$
    - Sample batch of real data samples $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_M\}$ from $p_{data}(\boldsymbol{x})$
    - Sample random numbers $\{\epsilon_1, ..., \epsilon_M\}$ from $U[0, 1]$

    $\hat{\boldsymbol{x}}_i = g(\boldsymbol{z}_i, \boldsymbol{w_g})$

    $\tilde{\boldsymbol{x}}_i = \epsilon_i \boldsymbol{x}_i + (1 - \epsilon_i)\hat{\boldsymbol{x}}_i$

$$\boldsymbol{w}_{\boldsymbol{d}}^{(t+1)} = \boldsymbol{w}_{\boldsymbol{d}}^{(t)} - \nabla_{\boldsymbol{w_d}} \left[\frac{1}{M} \sum_{i=1}^{M} d(\hat{\boldsymbol{x}}_i, \boldsymbol{w_d}) - d(\boldsymbol{x}_i, \boldsymbol{w_d}) + \lambda \left(\left\| \nabla_{\tilde{\boldsymbol{x}}} d(\tilde{\boldsymbol{x}}, \boldsymbol{w_d}) \right\|_2 - 1 \right)^2 \right] \tag{15}$$

  end

  - Sample batch of noise samples $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_M\}$ from noise prior $p_g(\boldsymbol{z})$
  - Update generator weights to minimalize L:

$$\boldsymbol{w}_{\boldsymbol{g}}^{(t+1)} = \boldsymbol{w}_{\boldsymbol{g}}^{(t)} - \nabla_{\boldsymbol{w_g}} \frac{1}{M} \sum_{i=1}^{M} -d(g(\boldsymbol{z}_i, \boldsymbol{w_g}), \boldsymbol{w_d}) \tag{16}$$

end

Algorithm 2: SGD training of a Wasserstein GAN. λ hyperparameter is commonly chosen 10.

In the following sections, we cover some important practical, yet general aspects of GAN implementations.

3.5 Deep Convolutional GANs

When Goodfellow et al. proposed GANs in 2014[4], they used multi-layer perceptrons, meaning their neural networks consisted of Fully Connected layers. As well known in the field, for image-like data convolutional layers are more suited.

Alec Radford et al. [12] first reported succesful training of GANs with a pure convolutional architecture. Their experiments concluded in several practical guidelines:

1. Instead of pooling layers, use strided convolutions
2. Use Batch Normalization for both the discriminator and generator
3. Remove fully connected layers
4. Use ReLu activations for the generator, Tanh for the last layer
5. Use LeakyReLu activation for the discriminator

Figure 6. show the generator architecture used in [12]. Worth noting that since the generator is broadening a 1-dimensional random vector into a 2-dimensional image. To achieve this they used a new type of convolutional layer, called Fractionally Strided Convolution or Transposed Convolution, where the output of the layer is wider instead of being slimmer.

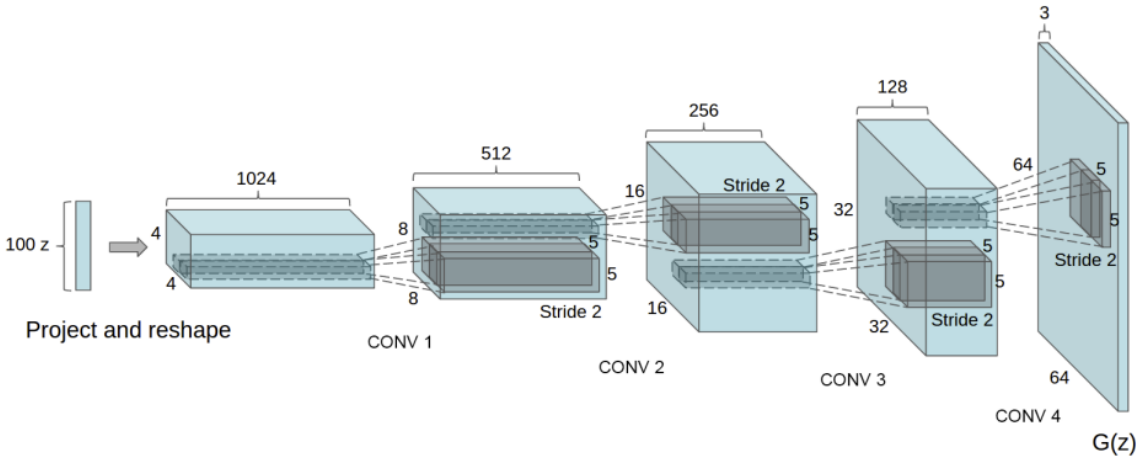
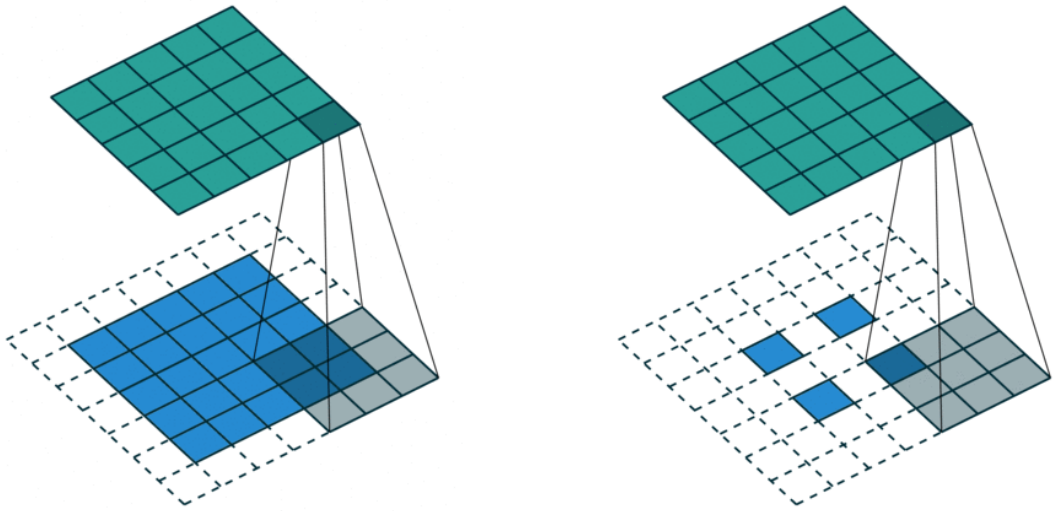


Figure 6: The architecture of a DCGAN's generator[12].



(a) 2D convolution using a kernel size of 3, stride of 1 and padding

(b) Transposed 2D convolution with no padding, stride of 2 and kernel of 3

Figure 7: Difference between regular convolution operation (left) and transposed convolution operation (right). The latter is needed in the DCGAN's generator. Blue (bottom) data the inputs of the layer, and green (top) are the outputs. Images from [13].

As we will see in our experiment setups, some of these practical guidances are kept, while some are thrown away: we didn't use Batch Normalization layers, because they make elements of the same batch statistically dependent, a direct route to mode collapse. Also, we used LeakyReLu in both the discriminator and the generator, to reduce the number of dead neurons. The other gudelines are kept.

3.6 Conditional GANs

So far the only input to the generator was a random vector denoted $\boldsymbol{z}$, and the input of the discriminator was either a real or artificial $\boldsymbol{x}$ data vector. The input $\boldsymbol{x}$ had no label that the GAN was aware of, thus they were *unconditional* regarding the class of the image.

Remember tough, that we wish to improve the performance of classification training, thus the generated data must have a corresponding class label, and also be a representative member of its class.

This way, we need to *condition* the generator to the class which we wish to synthetize. Also, since the discriminator is supervising the training of the generator, the discriminator has to 'know' what class it is dealing with, alas it has to be conditioned the same way.

This was, the training algorithm of the GAN is further modified as follow, reaching it's final form:

- Select hyperparameters k, μ_g, μ_d M, λ
- Initialize w_g and w_d weights

for N_{it} steps **do**

  for k steps **do**

    - Sample batch of noise samples $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_M\}$ from noise prior $p_g(\boldsymbol{z})$
    - Sample batch of real data samples with corresponding labels $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_M, y_M)\}$ from $p_{data}(\boldsymbol{x}, y)$
    - Sample random numbers $\{\epsilon_1, ..., \epsilon_M\}$ from $U[0, 1]$

    $\hat{\boldsymbol{x}}_i = g(\boldsymbol{z}_i, y_i \boldsymbol{w_g})$

    $\tilde{\boldsymbol{x}}_i = \epsilon_i \boldsymbol{x}_i + (1 - \epsilon_i)\hat{\boldsymbol{x}}_i$

$$\boldsymbol{w_d}^{(t+1)} = \boldsymbol{w_d}^{(t)} - \nabla_{\boldsymbol{w_d}} \left[\frac{1}{M} \sum_{i=1}^{M} d(\hat{\boldsymbol{x}}_i, y_i, \boldsymbol{w_d}) - d(\boldsymbol{x}_i, y_i, \boldsymbol{w_d}) + \lambda \left(\|\nabla_{\tilde{\boldsymbol{x}}} d(\tilde{\boldsymbol{x}}, y_i, \boldsymbol{w_d})\|_2 - 1 \right)^2 \right] \tag{17}$$

  end

  - Sample batch of noise samples $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_M\}$ from noise prior $p_g(\boldsymbol{z})$
  - Sample batch of label samples $\{y_1, ..., y_M\}$ from marginal label distribution $p_{data}(y)$
  - Update generator weights to minimalize L:

$$\boldsymbol{w_g}^{(t+1)} = \boldsymbol{w_g}^{(t)} - \nabla_{\boldsymbol{w_g}} \frac{1}{M} \sum_{i=1}^{M} -d(g(\boldsymbol{z}_i, y_i, \boldsymbol{w_g}), y_i, \boldsymbol{w_d}) \tag{18}$$

end

Algorithm 3: SGD training of a Conditional Wasserstein GAN.

3.7 Issues with GANS and attempts at metrics

The main issue with GAN training is that the optimizing criteria, e.g. the Wasserstein Distance, is approximated by the discriminator itself. This way, if the discriminator's training is poor, we can not realiably tell if the training is converging. In the domain of natural images, when we can look at the generator's output and tell if it 'looks real', this is less of a problem, albeit not enough for comparing models.

In our settings, the data are functional connectivity matrices, which are hard to evaulate with 'our eyes'. Also the reliability of the generated data's class-correctness is crucial. Thus, we need a realiable metric that directly informs us about the quality of the generated outputs.

Since most community applications work with natural images, Heusel et al.[14] came up with an empirical solution to compare different GAN models. Their observation is that the outputs of the final average pool layer of an ImageNet-trained Inception model approximately follow a multivariative gaussian distribution for real natural images. For generated data, the Inception's activations also follow a gaussian distribution, and the distance between the two gaussians correlate well with the goodness of the artificial data. The authors called this metric the Fréchet Inception Distance (FID), which is the Wasserstein Distance of the 2 aforementioned gaussian distributions.

Let us denote the mean and covariance of the activations μ_{real} and Σ_{real}, and the mean and covariance for generated samples μ_{gen} and Σ_{gen}. This way, the FID for the generator's output is calculated as:

$$FID = \|\mu_{real} - \mu_{gen}\|_2^2 + \mathrm{Tr}\Big\{\Sigma_{real} + \Sigma_{gen} - 2\left(\Sigma_{real}\Sigma_{gen}\right)^{\frac{1}{2}}\Big\} \tag{19}$$

In Figure 8, they ran several experiments where they added different disturbances to real images. Calculating the FID between undisturbed and increasingly disturbed images, they found good correlation between data goodness and FID.

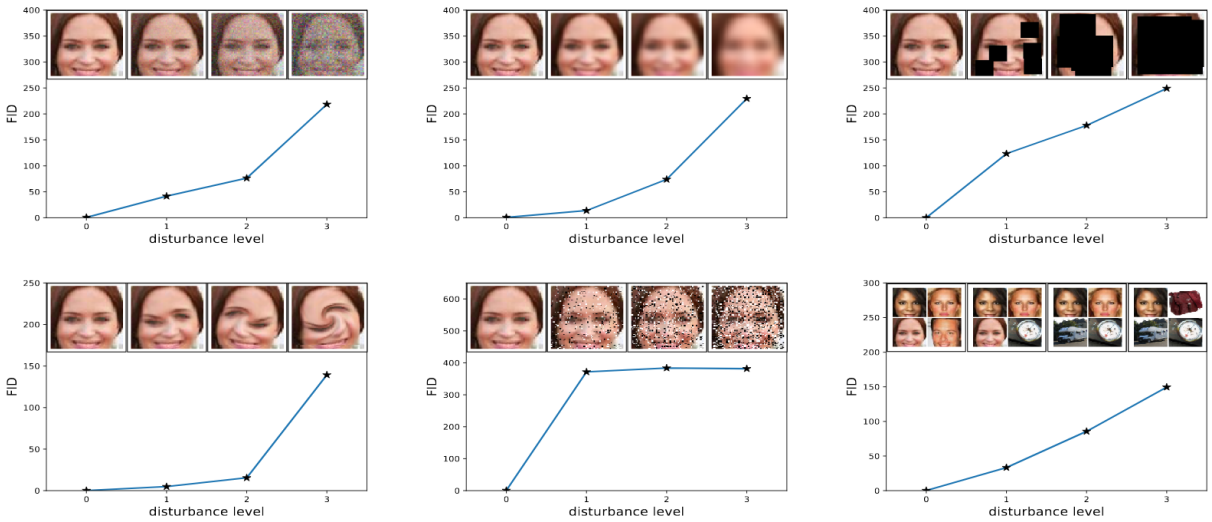


Figure 8: FID responds well to the increase of different disturbances. Top row: gaussian noise, gaussian blur, implanted black rectangles. Bottom row: random swirl, salt & pepper noise, and contamination with images from different domain.[14]

This formula lacks two important properties: it is not class specific, so the class-correctness can not be measured, and it is calculated via a network trained on natural images, while our data are connectivity matrices. During the following section, we discuss our solutions to these issues.

4 Experiments

Our goal was to generate connectivity matrices of convincing quality. In order to achieve this, we first aimed to design an appropriate metric that reflects the goodness of the generated data.

Also, our initial task was to generate matrices with brain age as associated labels, which is regression. We found, however, that this task was not reasonable within the given time, since this way generating label-correct matrices is a highly complicated problem to optimize. This way, instead of predicting brain age, we worked with gender prediction from the same matrices, which, as being a simple binary classification task, was easily carried out.

4.1 The UK Biobank dataset

The UK Biobank Dataset [15] is a wide-spread initative taken place in the United Kingdom to collect biomedical, neurological, genetic data of around a hundred thousand individuals. Our connectivity matrix dataset originates from this databank, consisting of 19,818 matrices.

One key method used in preprocessing is Independent Component Analysis, or ICA. It is a computational method for separating a multivariative close-to-gaussian signal to additive, statistically independent, non-gaussian subsignals. If we denote the signal as the observed $\boldsymbol{x} = (x_1, ..., x_m)^T$ random vector, components as $\boldsymbol{s} = (s_1, ..., s_n)^T$, the task is to find linear transformation $\boldsymbol{W}$ so that $\boldsymbol{s} = \boldsymbol{W}\boldsymbol{x}$, where $\boldsymbol{s}$ is a maximally independent vector, where independence is measured by $F(s_1, ..., s_n)$. In for example linear noiseless ICA, $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{s}$, meaning we have to iteratively find the mixing matrix $\boldsymbol{A}$, and the source vector $\boldsymbol{s}$. This can be done via loss functions that enforces the non-gaussianity of the calculated $\boldsymbol{s}$ vectors, and that minimize the mutual information between elements of $\boldsymbol{s}$. When converged, regular or pseudo inverse can be used to transform $\boldsymbol{A}$ to $\boldsymbol{W}$. As we will see, this method will be used to find the signal of each independent brain network.

The raw, original fMRI time series is preprocessed in a complex, rigorous way by the UK Biobank team. The following description originates from the documentation [16]:

1. First, the raw timeseries is motion-corrected using MCFLIRT [17] algorithm

2. Grand-mean intensity normalisation of the entire 4D dataset by a single multiplicative factor

3. Highpass temporal filtering (Gaussian-weighted least-squares straight line fitting, with $\sigma = 50s$

4. EPI unwarping with B_0 field maps [18]

5. GDC unwarping [19]

6. Structured artefacts are removed by ICA+FIX processing [20]: Independent Component Analysis followed by FMRIB's ICA-based X-noiseifier [21], where FIX was hand-trained on 40 Biobank rfMRI datasets.

Now, the rfMRI data is preprocessed. Next steps are about connectivity matrix extraction.

1. Calculate group-average resting-state networks using 4100 datasets:

   (a) Temporarily demean and each timeseries and normalize it's variance according to [20]

   (b) Group-PCA output was generated by MIGP [22]

   (c) Group-ICA tool [20] applying spatial ICA with dimensionality 100, meaning 100 distinct ICA components are generated. These are the parcellations of the gray matter, however, they are not binary masks, can overlap, and each can be composed of separate areas, unlike when brain atlases are used.

   (d) Since ICA identifies signal sources, different noises are present in the output. This way, they remove artefactual components, leaving 55 components. The resulting group-ICA components are visualized in Figure 9.

2. Map each group-ICA spatial map component onto each subject's rfMRI timeseries data to obtain one timeseries for each component. In this setting, each ICA component is viewed as a ‚node'.

3. For each subject, calculate full normalized temporal correlation between every pair of node timeseries, using the FSLNets toolbox[23]

4. Values were Gaussianised from Pearson correlation scores (r-values) into z-statistics

5. Finally, a Tanh function is applied by us to squeeze values to range (-1,1).

During our experiments, the data was split: the training set contained 14864 matrices, the validation and test set consisted of 2477 matrices each. The training set was 53% female and 47% male, which is relatively balanced, thus accuracy remained a good metric.

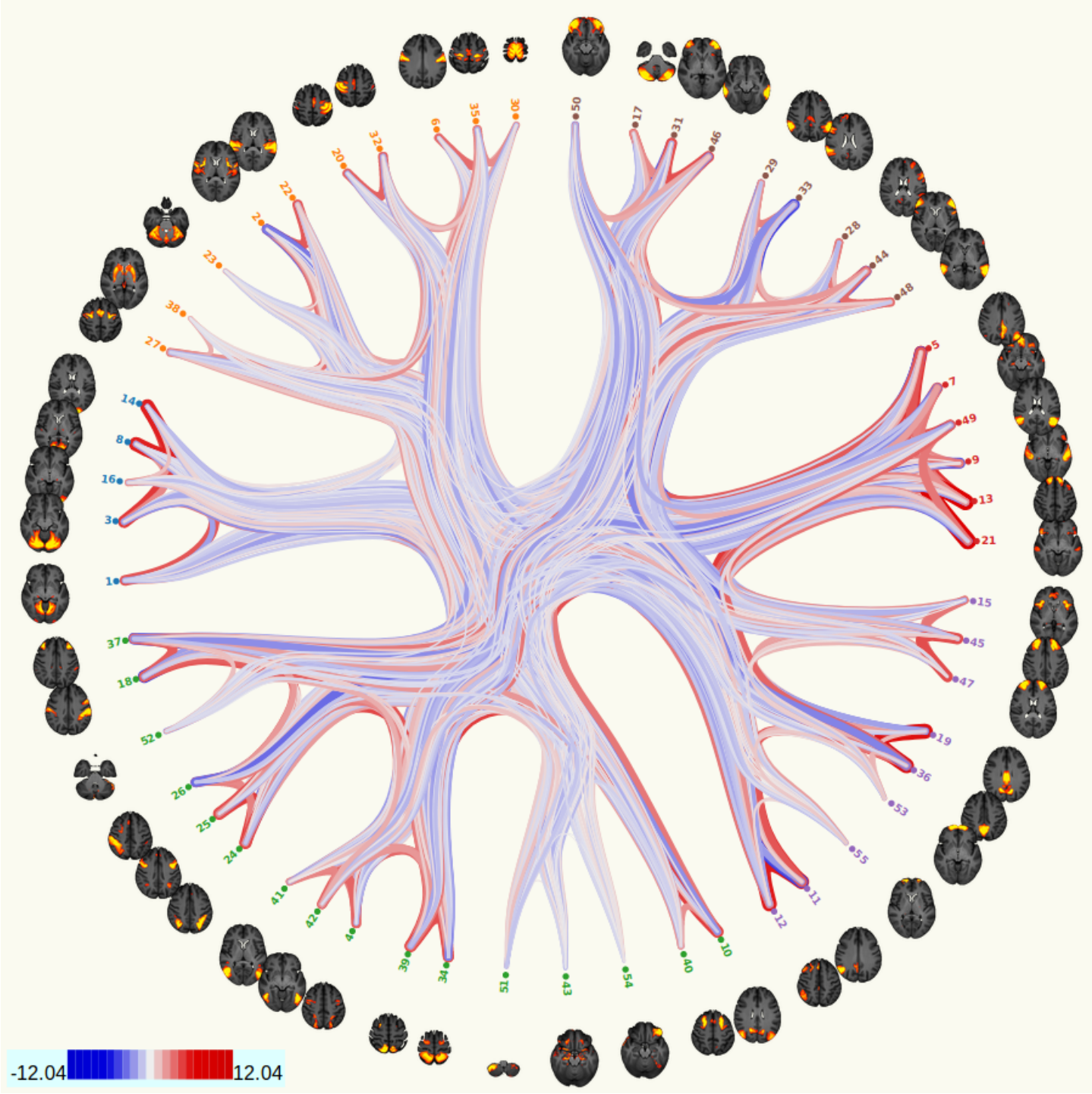


Figure 9: Summary of connectivity for the group-averaged dataset. Note that all 55 ROIs are present. The lines represent each connectivity matrix element. Red means strong positive correlation, blue strong anti-correlation, and white represents no correlation. Image from [24].

4.2 Classification experiments

4.2.1 Architecture

In order to obtain a baseline for our GAN training, we trained several classification networks. More specifically, we run a random architecture search over a discretized range of each hyperparameter. Let's start with the core, constant aspects of the classificator networks' architectures. Similarly to Meszlényi et al [2], we used convolutional networks, and the architecture can be seen in Figure 10.

For feature extraction, two convolutional layers were used. Let N_{ROI} denote the number of ROIs, meaning the input matrix has shape $N_{ROI} \times N_{ROI}$. In natural image processing, square shaped kernels are usually used, e.g. 3x3 or 7x7. The reason behind this is that in such images, nearby pixels have strong correlation between them. In connectivity matrices, however, this is not the case. In the first layer's input, the matrices themselves, each row or column of the matrices represent data about the statistics of one node. This way meaningful feature extraction can only happen if the kernel covers exactly one row or column. In the second layer's input, the matrices are reduced into vectors with length N_{ROI}. This way, the second layer's kernel must cover the remaining area perfectly, having one real number for each matrix.

Of course, convolutional layers can have channels as well. Let us denote the number output channels of the two convolutional layers c_1 and c_2. This way, the output of the first layer is shaped $N_{ROI} \times c_1$, while the second layer's output is shaped c_2. The kernels are shaped $1 \times c_1 \times N_{ROI} \times 1$ and $c_1 \times c_2 \times 1 \times N_{ROI}$, respectively, following (c_{in}, c_{out}, w, h) order.

After feature extraction, the last, fully connected layer performs classification, having a single real number for the whole matrix. It's input is shaped c_2, while it's output is one scalar. Finally, a sigmoid layer is applied to map real numbers to values of a probabibility distribution. Thinking of the classificator as an estimator for probability distribution over the dataset, we can define the sigmoid as:

$$\mathbf{P}(Y_i = 1 | X_i; \mathbf{w}) = \sigma(\hat{\mathbf{y}}_\mathbf{i}) = \frac{1}{1 + e^{-f(x_i)}}, \tag{20}$$

where $f(x_i)$ is the last layer's output for sample i. In our experiments we realized that increasing the number of fully connected layers yielded no significant advantage. This way, the only free parameters for the architecture are c_1 and c_2.

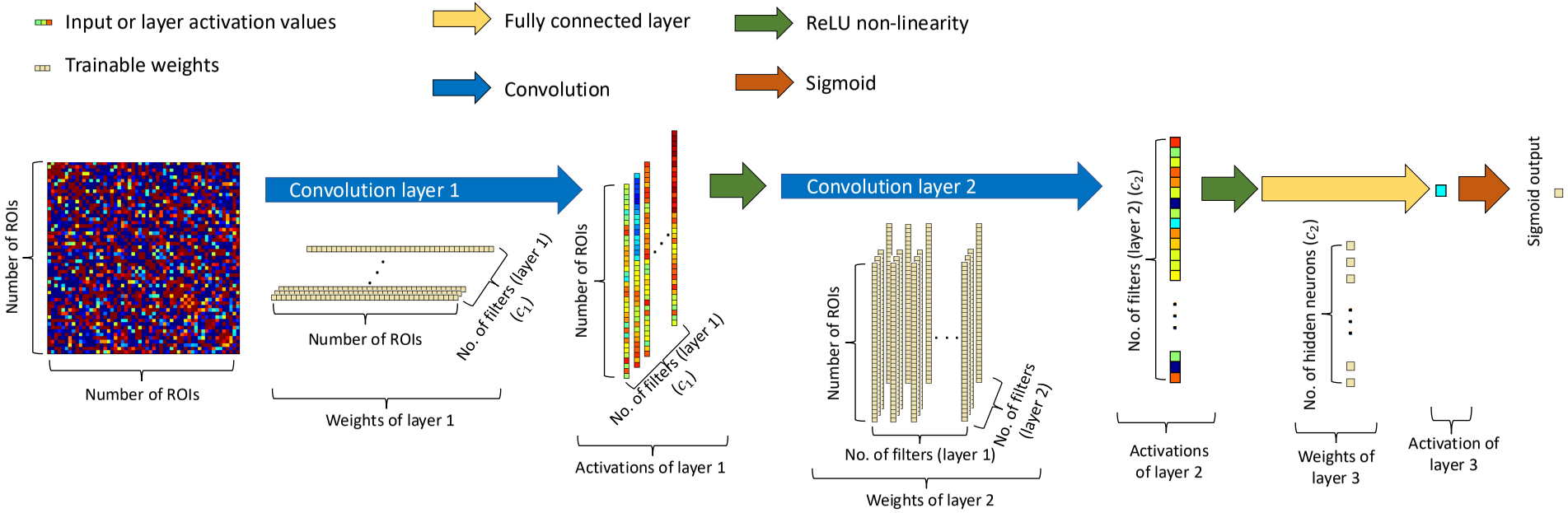


Figure 10: The general architecture of a simple classificator CNN suited for functional connectivity matrices. The architecture is parametrized with 2 integers: c_1 and c_2.

4.2.2 Hyperparameter search

In addition to finding good values for c_1 and c_2, during training, additional hyperparameters must be defined. We limited our training to the use of the simplest yet effective gradient descent variant: Stochastic Gradient Descent with Nesterov momentum. This optimizer has 3 hyperparameters: learning rate , momentum, and weight decay, leading us to have five hyperparameters in total. We used random grid search for all of these, meaning we chose a meaningful discretized interval for them:

1. c_1: 2 to 256 with multiplicative step 2 (8 values)
2. c_2: 2 to 256 with multiplicative step 2 (8 values)
3. learning rate: $3 \cdot 10^{-5}$ to $3 \cdot 10^{-1}$ with multiplicative step 3 (9 values)
4. momentum: $1-t$ where t ranges from 0.01 to 0.15 with additive step 0.01 (15 values)
5. weight decay: 10^{-6} to 10^{-2} with multiplicative step 10 (5 values)

In total, we have 43200 possible configurations laid in 5 dimensions. The hyperparameter search can be best described with the following simple algorithm:

```
◦ Define hyperparameter configuration space H
◦ Initialize pseudorandom number generators
for N_nets steps do
    ◦ Sample hyperparameters h from H
    ◦ epoch=0; stop_iter = False; best_val_loss=inf
    while epoch < N_epoch and not stop_iter do
        ◦ Train network for 1 epoch
        ◦ val_loss = validate(network)
        if val_loss < best_val_loss then
            save()
            best_val_loss = val_loss
        end
    end
    stop_iter = should_stop()
end
```

Algorithm 4: Classification hyperparameter search.

In simple terms, we save the networks when their validation loss is the lowest. The function *should_stop* watches the validation loss over the epochs, and is designed to detect overfitting and convergence. This is needed to reduce the total time cost of the hyperparameter search.

We trained around 1200 networks, which took around 20 hours of GPU time. The best results can be seen in Table 1:

Validation Loss	Validation Acc.	# of params	c_1	c_2	Lr.	Mom.	Wd.
0.319	0.8619	3619329	256	256	0.00729	0.96	0.001
0.322	0.8627	915585	256	64	0.00243	0.87	0.0001
0.324	0.8623	908545	128	128	0.02187	0.97	1E-05
0.324	0.8623	464961	256	32	0.00729	0.88	0.001
0.324	0.8655	126993	256	8	0.00729	0.97	0.001
0.381	0.8304	1333	8	2	0.06561	0.97	0.0001

Table 1: Some of the best results of the hyperparameter search.

Two networks will be in our interest. In the first row, we have the best loss and accuracy, this way, we will use this model when we evaluate predictive power related metrics. We will call this network Learner.

Note that the last line shows a network that did not achieve outstanding accuracy, but has incredibly low amount of parameters. This will be our Reference Network, which we will discuss in the next section.

4.3 Wasserstein Activation Distance

As mentioned in Section 3.7, at time of writing, the most accurate metric for evaluating generated images in a direct way is the Fréchet Inception Distance. However, this is not applicable in our settings, since the Inception network is used for natural images, and FID is not class-specific. To resolve these issues, we follow the most straightforward path:

1. Have a network trained on connectivity matrices (Reference Network), and use it's second convolutional layer's output as activations
2. Train Reference Network to predict the same classes as those we wish to generate correctly
3. Calculate (19) for all classes, and average them

This way, our new metric, named Wasserstein Activation Distance (WAD), is computed as follows:

$$WAD = \frac{1}{C}\sum_{c=1}^{C}\left\{\left\|\mu_{real}^{(c)} - \mu_{gen}^{(c)}\right\|_2^2 + \mathrm{Tr}\left\{\Sigma_{real}^{(c)} + \Sigma_{gen}^{(c)} - 2\left(\Sigma_{real}^{(c)}\Sigma_{gen}^{(c)}\right)^{\frac{1}{2}}\right\}\right\} \tag{21}$$

4.3.1 Reference Network

In the last row of Table 1, we can see the hyperparameters of the network that we chose to be our reference network, which will provide us the activations necessary to compute WAD. The reason behind this choice is that this was the network that achieved relatively good loss and accuracy, while has incredibly low amount of parameters. This way, the activations of the second convolutional layer have low dimensionality, meaning mean and covariance calculations are more stable (curse of dimensionality), and cheaper. Also, since the dimension of the activations are exactly 2, we can easily visualize them. We can see the architecture of the reference network in Figure 11.

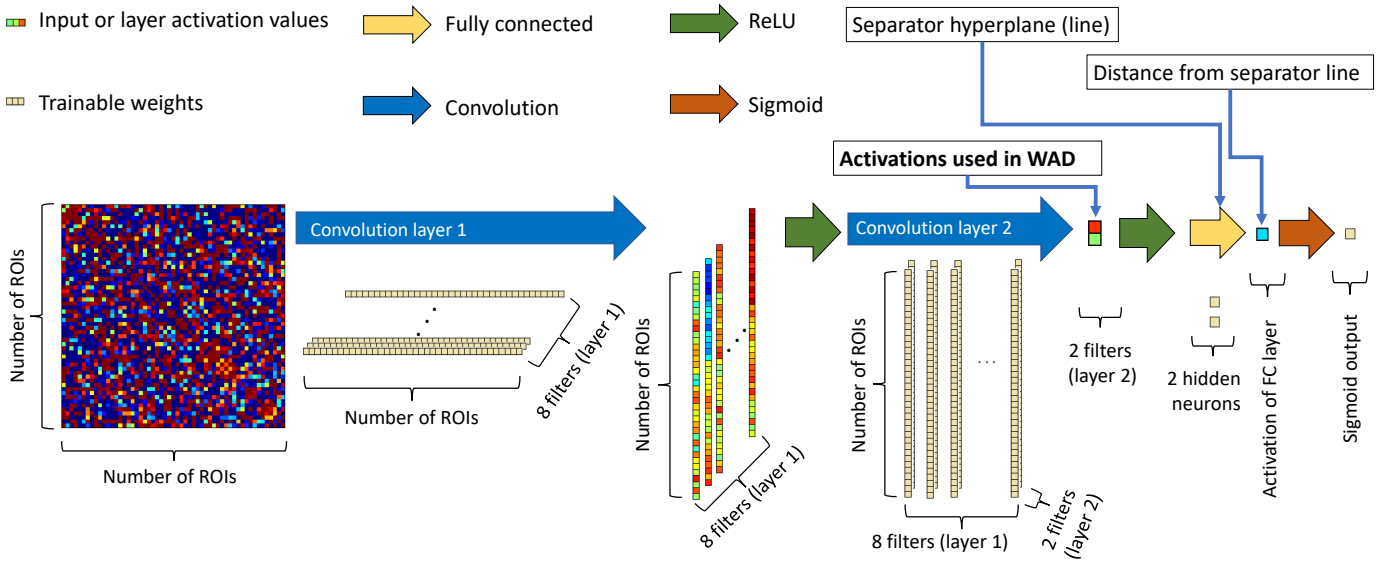


Figure 11: The architecture of the Reference Network.

Note that as we can see in Figure 11, the convolution layers yield two real numbers for each matrix, meaning that we can place each connectivity matrix in a two-dimensional space. This way, the last, fully connected layer represents the following function:

$$y = w_1 x_1 + w_2 x_2, \tag{22}$$

where w_1 and w_2 are the two weights of the last FC layer, and x_1, x_2 are the output coordinates of the last convolutional layer. Since we have a sigmoid function as the last activation, and we threshold predictions at 0.5, $y > 0$ elements will be assigned class 1, and $y < 0$ elements will have class 0. This way, we can think of the last layer as a separator line ($w_1 x_1 + w_2 x_2 = 0$) between class 1 and 2 activations.

In Figure 12, we visualize the Reference Network's activations for 7000 real examples. Notice that the distribution of the activations are approximately gaussians for each class, and they overlap. Overlapping occurs because the Reference Network's final accuracy was 83%.

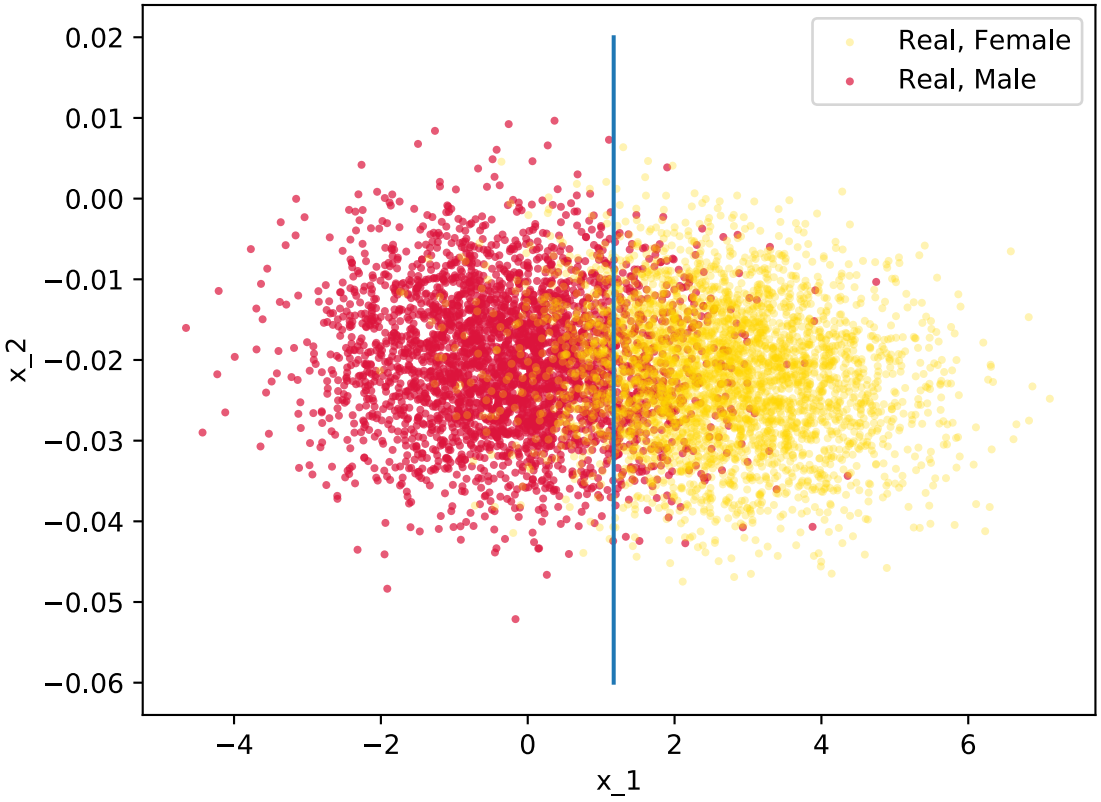


Figure 12: Reference Network's activations for 7000 real examples. Yellow dots represent female, while red dots represent male participants. Also, the blue line is the separator line.

4.3.2 Learning Loss

As mentioned before, in the first row of Table 1, we have the classificator network with the best validation performance. We will use this network, to evaluate an absolute metric to evaluate the performance of the GAN and WAD itself: how much class-specific information is retained in the generated data. To do this, we use the generated data as the training dataset, use the regular validation dataset for early stopping, then evaluate said network on the test set. We will call the test loss arising from this setting the Learning Loss. The Learning Loss of the real data was 0.32.

While being an absolute measure, Learning Loss takes around 10 minutes on a high-end GPU to evaluate, in contrast to WAD taking milliseconds. This means Learning Loss not usable in GAN hyperparameter search or early stopping, rather as a final testing

method.

4.3.3 Performance of WAD

In order to verify that WAD is a useful metric, we conducted a simple experiment. In short, instead of comparing real matrices to generated matrices, we compared the same real matrices with matrices that have noise added to them. This noise was symmetrical, uniform in range $(-1, 1)$. During the experiment, we added the noise to the matrices with different weights. Denoting the noise ratio with α, the noise with $\boldsymbol{z}$, and the original matrix $\boldsymbol{x}$, we get the noisy matrix as:

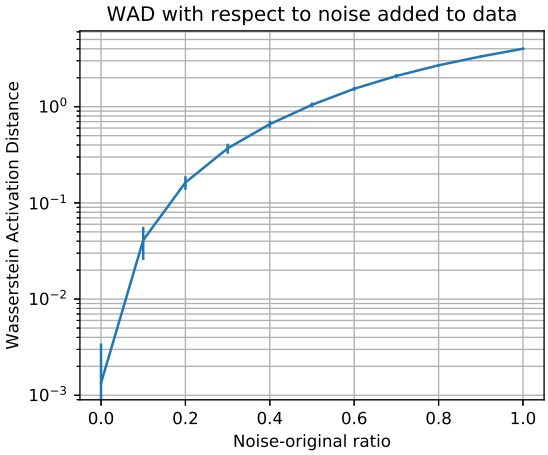
$$\tilde{\boldsymbol{x}} = (1 - \alpha)\boldsymbol{x} + \alpha\boldsymbol{z}. \tag{23}$$

Of course, the original and the noisy matrices are from mutually exclusive parts of the train dataset, both containing 7000 matrices. The size of the batches (7000) is important to be kept constant, as the calculated mean and covariance of the activations depend on it.

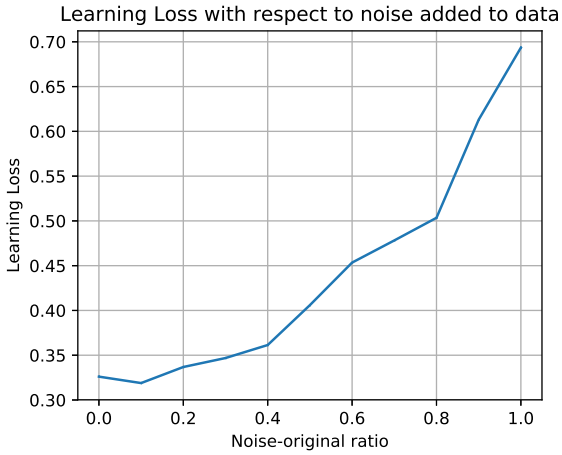
In Figure 13, we can see the result of the WAD performance test. In Figure 13a, we ran 50 measurements for each noise-ratio, and the errorbars represent 95% confidence interval. Note that WAD is monotone with low variance, thus captures well the goodness of the data.

In Figure 13b, we present the Learning Loss of the same noisy data. As we can see, both WAD and this training-based loss are monotone, while the latter appears to be more noisy.

Also, note that a noise with weight 0.1 is not visible as loss increase in Figure 13b, meaning that it is not necessary meaningful to train a GAN to have WAD below $\approx 3{\cdot}10^{-2}$.



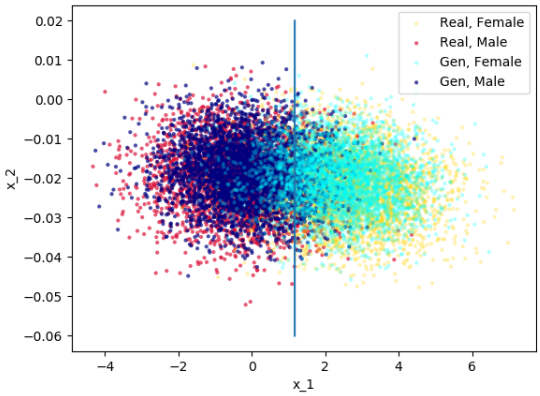
(a) WAD of original and noisy data



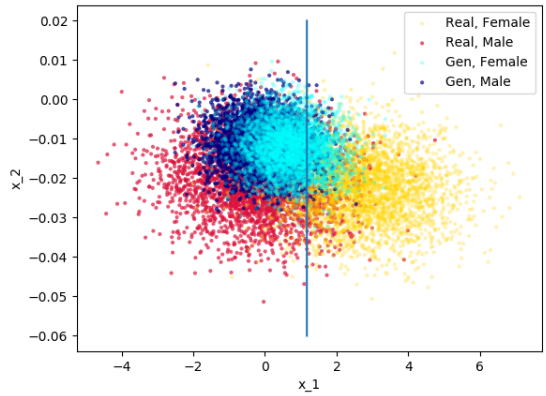
(b) Test loss of CNN trained on noisy data.

Figure 13: Comparison of the two GAN-related metrics: WAD and using artificial data as training data.

For visualization, in Figure 14a. and 14b, we can see the corresponding activations to noisy matrices, with noise ratio 0.1 and 0.7, respectively. Note that the noise with ratio 0.1 makes almost no visible difference, while the 0.7 ratio significantly changes the activations: the classes are displaced, their variance decreased, and their overlap increased.



(a) Noise ratio is 0.1.



(b) Noise ratio is 0.7.

Figure 14: Reference Network's activations for 7000 original (yellow and red dots) and 7000 noisy matrices. Comparison of effects of low amount of noise (left) and high amount of noise (right).

We should note that WAD has it's own limits. We can think of the Reference Network's first 2 convolutional layers as a projection from a $\frac{55\times(55-1)}{2} = 1485$ dimensional space to a

two-dimensional space. In this case, it would be easy to find disturbances in the data that would not affect the distribution of the activations, yet would render the data useless. It is one of our future goals to solve this issue.

In the following section, we describe our GAN setups, and use the described WAD to aid GAN training.

4.4 Training of GANs

To fully describe our experiments with GAN training, we will first go through the architecture of our GAN, briefly discussing the actual hyperparameters. Next, we present the algorithm for WAD-aided training, with some visualizations of the training process.

4.4.1 GAN Architecture

In Figure 15. and 16. we can view the general architecture for the label-conditioned generator and discriminator. Both Networks were trained with Adam optimizers. The final hyperparameters are hand-engineered, since the training of a single GAN takes around 3 hours on a high-end GPU.

In the generator, first we can find 2 parallel fully connected layers, which operate on the input noise and on the label, respectively. Their output is concatenated along the feature dimension. From now, we can find 2 Transposed Convolution layers, that expand and transform the concatenated features to a non-symmetrical matrix that has shape $N_{ROI} \times N_{ROI}$. Now, this matrix is symmetrized in a simple way ($\boldsymbol{x'} = \frac{1}{2}(\boldsymbol{x} + \boldsymbol{x}^T)$), then a $Tanh$ function is applied to squeeze the values to range (-1, 1). The exact hyperparameters can be seen in Table 2. The dimension of the input noise was 32.

f_1	f_2	c_1	Slope	Lr.	Wd.	β_1	β_2
64	64	32	0.01	10^{-2}	10^{-4}	0.5	0.99

Table 2: Hyperparameters for the generator. f_1, f_2, c_1 represent layer widths, *Slope* is the LeakyReLU's slope. The other parameters are the optimizer's.

In the discriminator, we can find a similar pattern: the output of the first convolutional layer that operates on the matrix is concatenated with the output of the label-processing Transposed Convolutional layer's output. Then, an another convolutional layer is applied, followed by a fully connected layer. In Wasserstein GAN, there is no nonlinearity on the end of the discriminator. The hyperparameters can be seen in Table 3.

c_1	l_1	c_2	Slope	Lr.	Wd.	β_1	β_2
96	96	48	0.01	0.1	10^{-4}	0.5	0.99

Table 3: Hyperparameters for the discriminator. c_1, l_1, c_2 represent layer widths, *Slope* is the LeakyReLU's slope. The other parameters are the optimizer's.

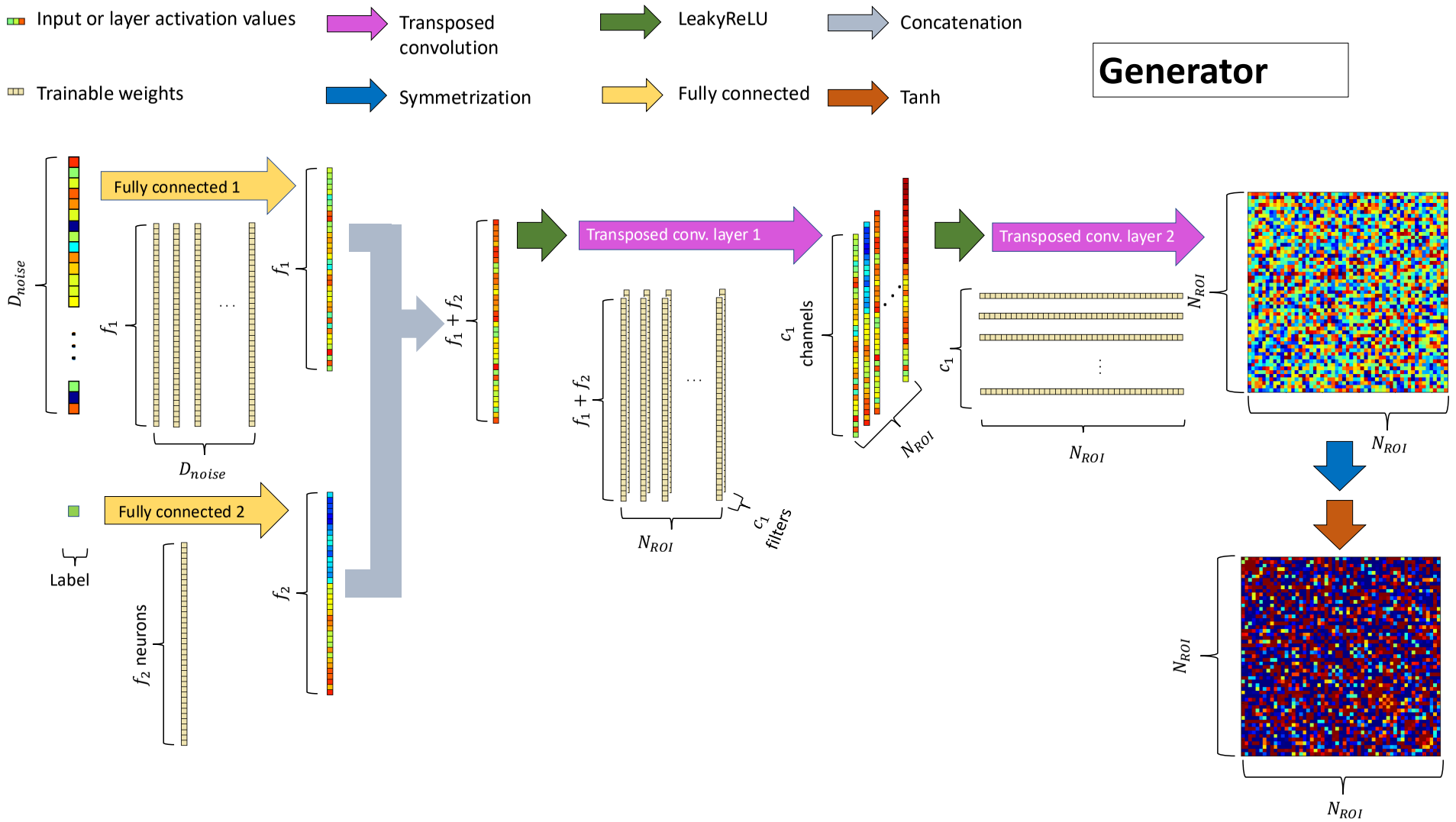


Figure 15: The architecture of the generator network.

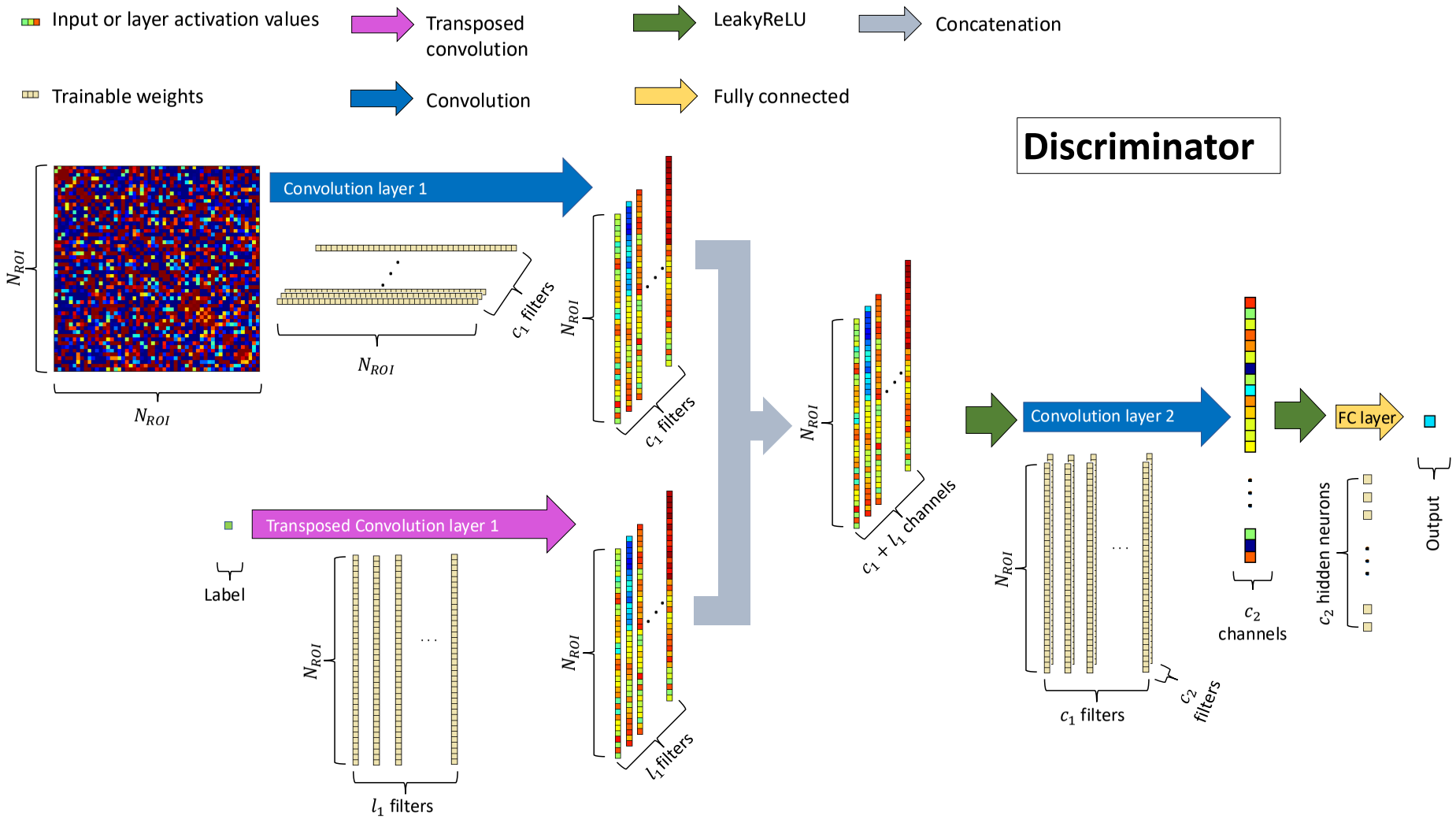


Figure 16: The architecture of the discriminator network.

4.5 Examining GAN Training

In this section we examine the training of our GAN. We have two simultaneous goals here: to prove that WAD is useful in GAN training, and to examine how well we managed to train our GAN. To do this, at each step, we evaluate WAD and the Wasserstein Loss. To compare these metrics, we calculate the Learning Loss at meaningful GAN training steps. Finally, the Learning Loss at WAD's minimum will measure the quality of the generated examples.

For training our GAN, we use Algorithm 3. and after each generator update, we evaluate WAD. Fortunately, all of the 14864 training examples fit in our GPU, meaning each step equals one epoch. The total training consisted of 360 epochs, which took 3 hours.

The best WAD value was reached at step 174, so we plotted the losses up to step 180: in Figure 17, we can see the Wasserstein Activation Distance during training, while in Figure 18, the discriminator's estimate for the Wasserstein Distance of the real and generated data van be viewed.

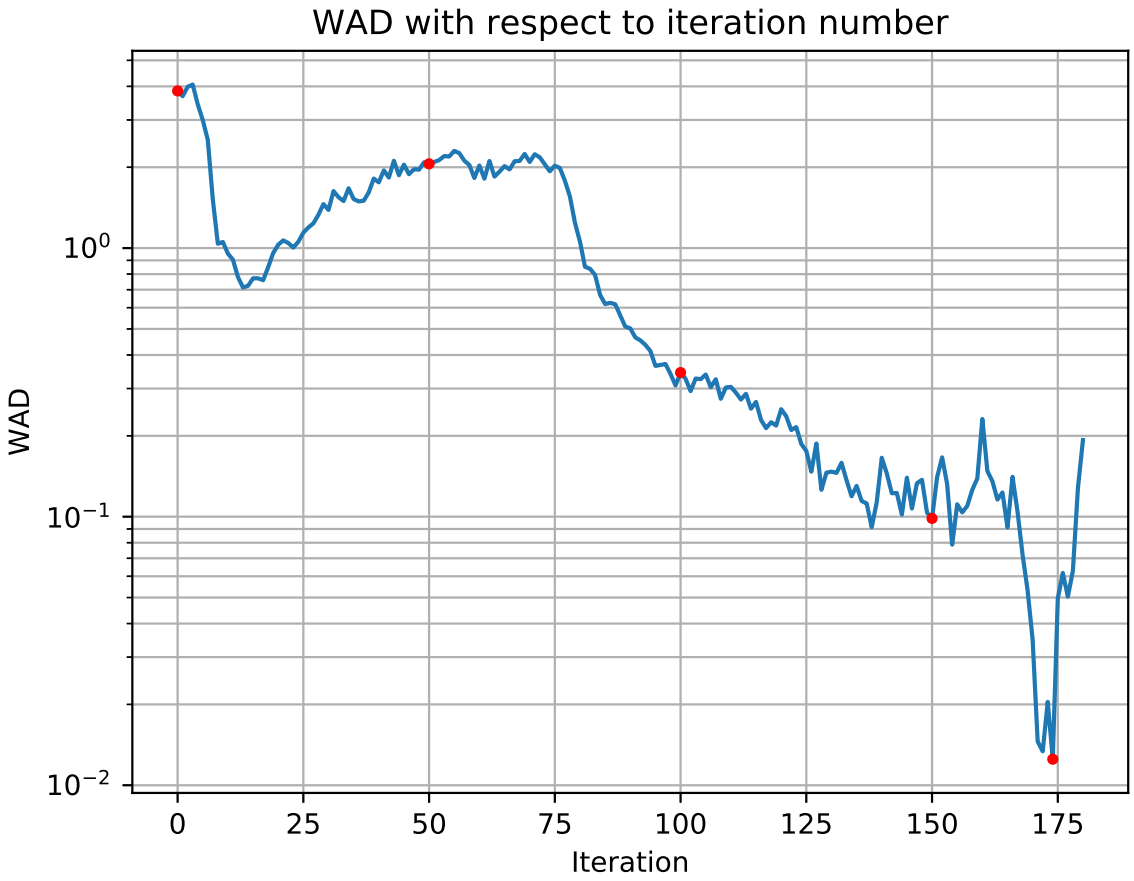


Figure 17: The Wasserstein Activation Distance for the GAN during training. The red dots mark steps that are further examined.

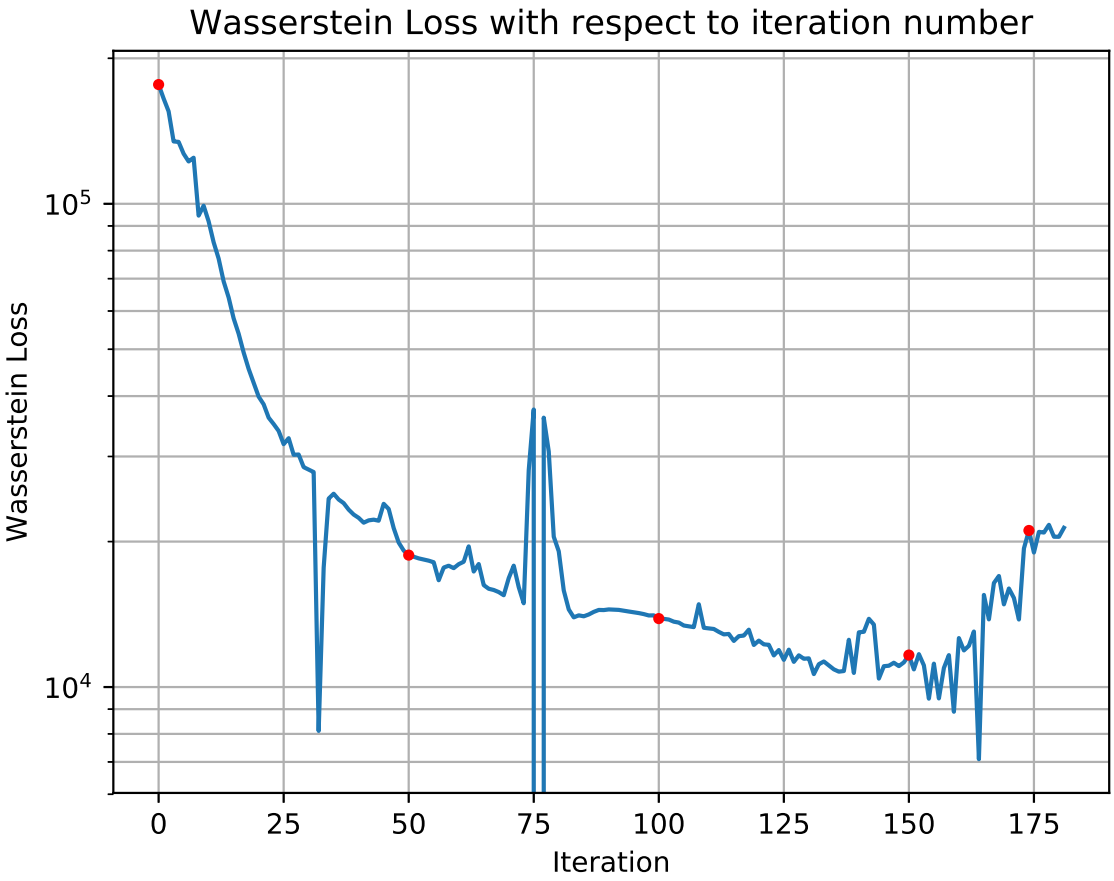
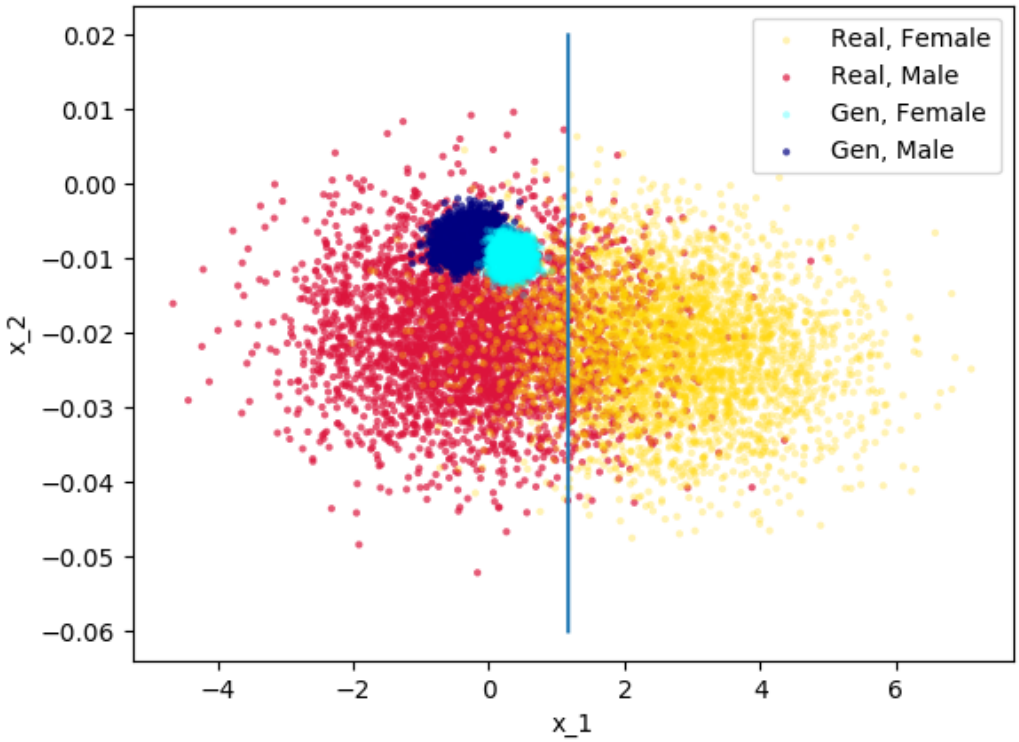
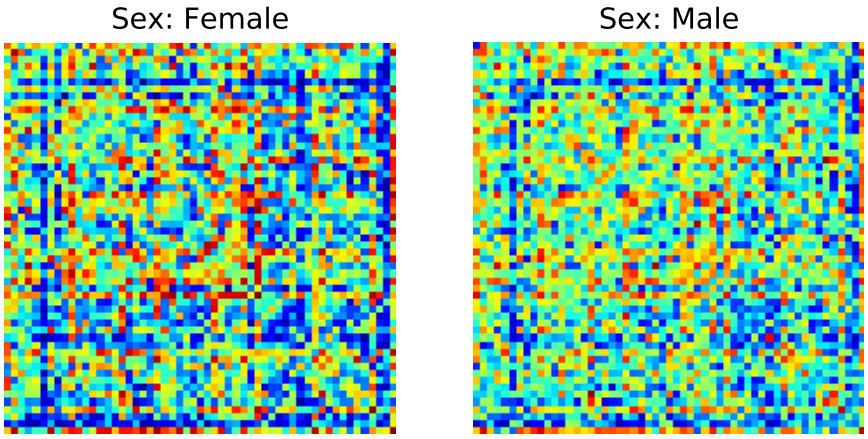


Figure 18: The Wasserstein Loss for the GAN during training. Do not confuse with WAD, as this is the training objective that is maximised by the discriminator and minimized by the generator. The steps marked with red dots, along with the minimum at step 76, are further examined.

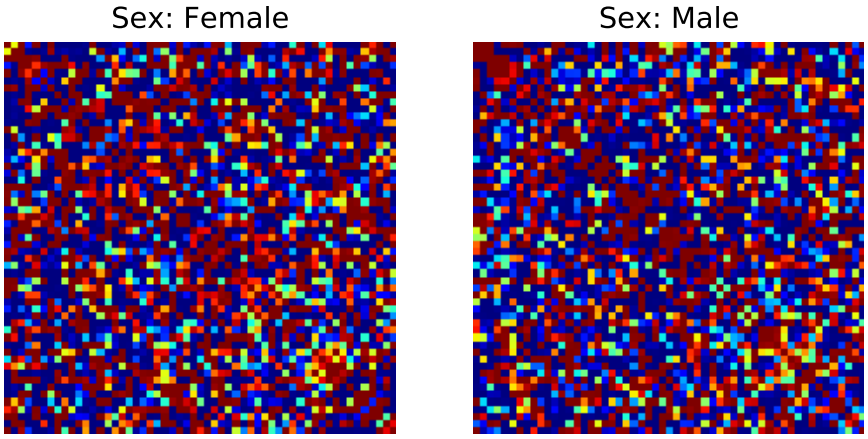
To further analyze the GAN training, in Figures 19 - 21. we visualize the Reference Net's activations and the generated matrices at steps 1, 100, and 174. In Figure 19, just after the first epoch, we can see that the activations have lower variance, and misplaced, and we can visually conclude that the generated matrices are not realistic. In Figure 20, however, we can see that the activations have appropriate variance, but still not class-correct. Also, it is now hard to visually evaluate the generated matrices. Last, after iteration 174, the activations almost perfectly match the real examples' activations, giving us a minimum of WAD.



(a) Reference Net's activations for generated examples at step 0.

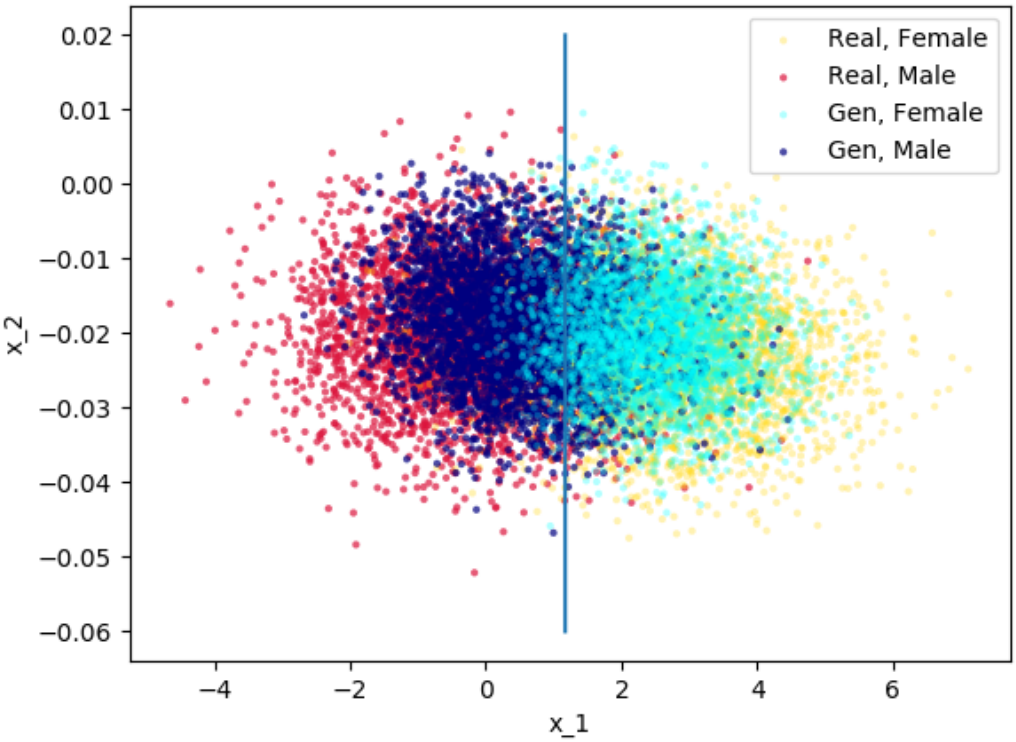


(b) Generated matrices at step 0.

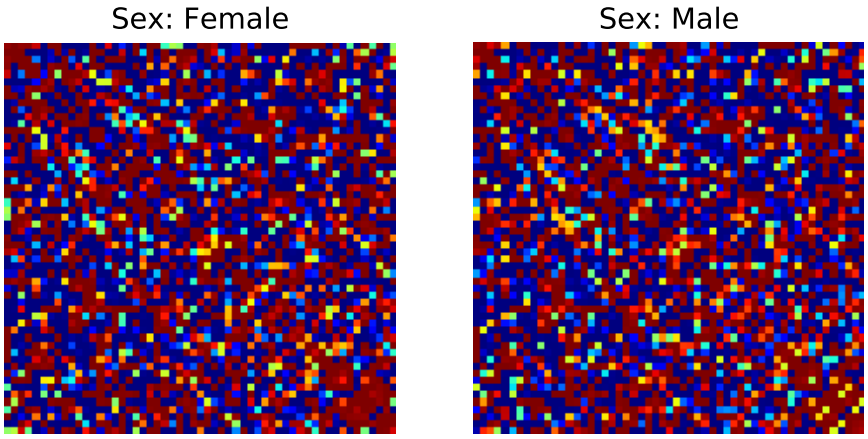


(c) Examples of real matrices for comparison.

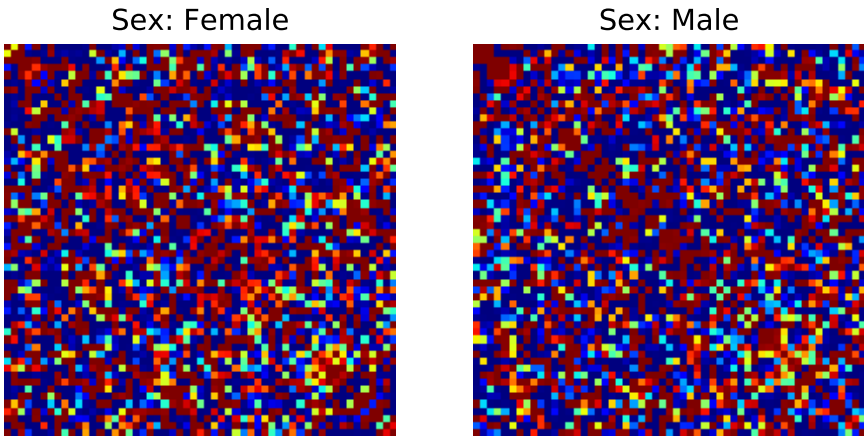
Figure 19: Activations and generator outputs at step 0.



(a) Reference Net's activations for generated examples at step 100.

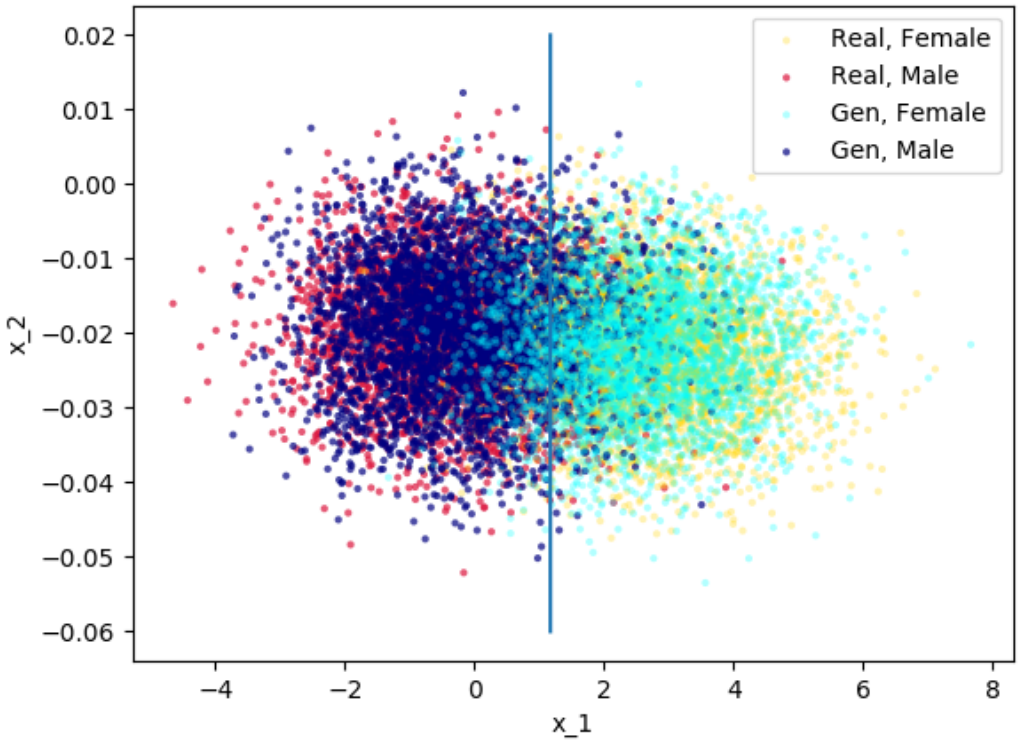


(b) Generated matrices at step 100.

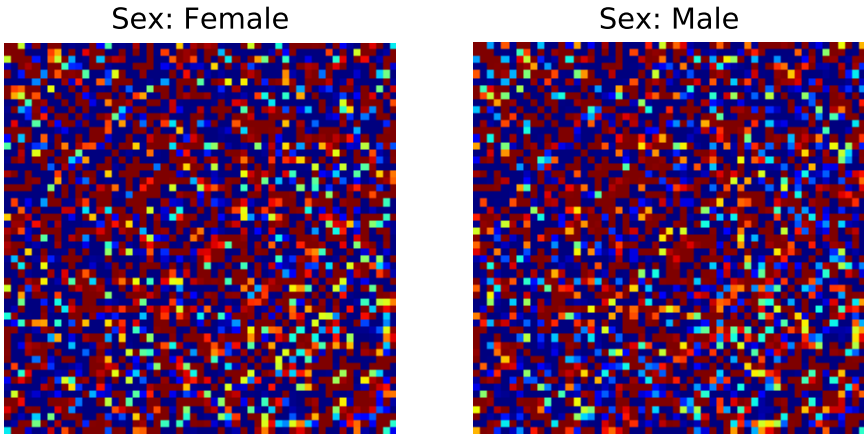


(c) Examples of real matrices for comparison.

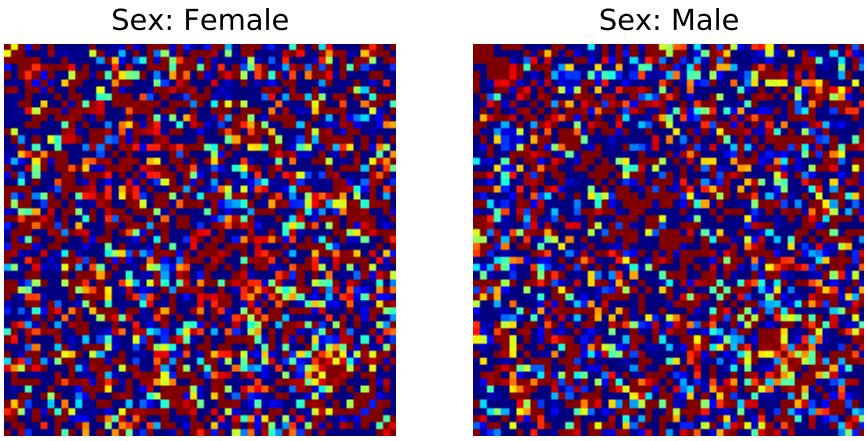
Figure 20: Activations and generator outputs at step 100.



(a) Reference Net's activations for generated examples at step 174.



(b) Generated matrices at step 174.



(c) Examples of real matrices for comparison.

Figure 21: Activations and generator outputs at step 174.

4.5.1 Final Results

As our first final goal, we argue why using WAD was necessery for our GAN training.

First, WAD is independent of GAN architecture, thus viable for hyperparameter and architecture search, and for comparing models.

Second, it is better for early stopping the GAN with WAD than with only the Wasserstein Loss. To see this, we evaluated the Learning Loss at steps marked by red in Figures 17. and 18. Also, along these points, we evaluated the Learning Loss at the minimums of WAD and the Wasserstein Loss, since early stop would happen in these points.

The results are summarized in Table 4. We can clearly see that the early stopping with WAD yielded better Learning Loss than with Wasserstein Loss. Of course, we can see in Figures 17. and 18. that both metrics are quite noisy, having many outliers, including the minima. We should note that during training, smoothing or removing outliers for early stopping purposes is cumbersome and ambiguous. Still, we compared the two metrics when the plots are smoothed with a moving average filter of length 10 steps: the smoothed WAD curve had its minimum at step 300, with value 0.025, and a test loss of 0.478, while the minimum of the smoothed Wasserstein Loss can be found at step 254 with value 8906, and a test loss of 0.498, meaning that with less significance, but WAD still outperformed the Wasserstein Loss.

The best Learning Loss was at step 150, meaning that WAD is not perfectly accurate. There are presumably many reasons behind this. First, same as with FID, so far WAD is not theoretically backed up, meaning we do not exactly know it's limits. Also, it is Neural Network based, meaning the training of the Reference Network might affect the performance of WAD. Furthermore, even if we found a way to 'perfectly' train the Reference Network, it projects the high-dimensional matrices to a two-dimensional plane. This mean that it should be possible to perturb the data in ways that the activations do not change, or worse, in ways that WAD decreases while information is clearly lost. This later argument can explain the outliers in the WAD plot in Figure 17.

Step	Learning Loss	Test Acc.	WAD	Wasserstein Loss	Note
1	0.88	0.497	4	$1.27 \cdot 10^5$	First epoch
50	0.8	0.52	2	$1.85 \cdot 10^4$	-
76	0.686	0.57	2	-843	Wasserstein L. minimum
100	0.496	0.769	0.35	$1.37 \cdot 10^4$	-
150	**0.464**	**0.783**	**0.098**	$\mathbf{1.06 \cdot 10^4}$	**Best Learning Loss**
174	0.5	0.767	$4.5 \cdot 10^{-3}$	$1.85 \cdot 10^4$	WAD minimum
Real	0.322	0.855	-	-	-

Table 4: Learning Losses (test losses) of generated data extracted at different GAN training steps. Last row corresponds to the real training dataset.

We can conclude that, using WAD as a guide to train GANs, we can achieve better results than with the Wasserstein Loss. Further improvements must be made in order to WAD become fully reliable. One of the promising directions for our further experiments is creating an ensemble of reference networks, with properties that make sure that they realize different projections, hopefully better reflecting the information retained in generated data. Also, we tested WAD only with uniform noise in Section 4.3.3. Finding new, meaningful ways to test the metric should yield better performance.

As for the success of training of our GAN, the final generated samples do not contain all the information as in the training data, since it has higher Learning loss than the real matrices. However, we see that some information is retained, since the accuracy corresponding to it's Learning Loss is well above random level (76.7 % vs 50-60%).

5 Conclusion

The goal of this thesis was to generate convincing functional connectivity matrices using Generative Adversarial Networks. I gained deep understanding of these matrices and how they are extracted from the raw fMRI data. After choosing the appropriate dataset, the UK Biobank's resting state fMRI connectivity matrix repository of 19,818 examples, I chose to refine the goal to generating artificial connectivity matrices that retain information relating to the subject's gender.

In order to obtain the appropriate GAN algorithm, after cumbersome experimentation, I combined several approaches used by the machine learning community. The resulting network was trained using Wasserstein Loss with gradient penalty. Wasserstein Loss is favourable over it's predecessor, the Jensen-Shannon Divergence, since the vanishing gradient problem is elimianted. For a Wasserstein GAN, the discriminator must be Lipsitz-1 continous, this was approximately enforced with Gradient Penalty. The architecture of the Wasserstein GAN was deep, convolutional, and conditioned to class labels.

I realized that training GANs while monitoring solely the training objective is not sufficient, since their values depend on the performance of the discriminator, rendering hyperparameter search and early stopping nearly impossible. To resolve this issue, I ported the powerful Fréchet Inception Distance (FID) metric to the domain of functional connectivity matrices in a class-sensitive setting. FID operates by measuring the Wasserstein Distance between the gaussian-distributed activations of an Imagenet-trained Inception Network's last Average Pool for real and generated samples.

To achieve this, using hyperparameter search I obtained the optimal network for activation extraction, called the Reference Network. I solved the issue of class-specifity by calculating the Wasserstein Distance of the activations of real and generated examples for each class, and took the average of them. This way, a new metric arised, which i called the Wasserstein Activation Distance (WAD). Like FID, WAD is a metric that does not depend on the architecture of the GAN, meaning it is suitable for manual or automatic hyperparameter search. This way, I found the optimal hyperparameters for the GAN, then evaluated the performance of WAD and the GAN as well. I also ran a simple test on WAD to observe the relationship between uniform noise added to the training data and WAD, finding WAD to be a monotone function of the intensity of noise added, and also having low variance.

I compared training with WAD to training with the Wasserstein Loss, since my hypothesis was that WAD is more suitable for validating GAN training than simply using the training objective itself. To validate the generated matrices' quality, I used a training-based metric, called the Learning Loss, which was computed by using the generated examples as training examples for a classificator, early stopping it with the validation dataset, then calculating the test loss. The Learning Loss is an absolute metric, since it reflects the amount of class-specific information retained in the generated data, meaning it is suitable for measuring the performance of the GAN and WAD.

During thorough experimentation, I realized that WAD is indeed more useful for GAN hyperparameter search and training than the simple Wasserstein Loss. However, I came to the conclusion that WAD is still incomplete, since the minimum of WAD and the minimum of the Learning Loss was at different steps. I found that the reason behind this is that WAD in it's current form uses only one classificator network with low amount of parameters, meaning it is easy to perturbe the data in ways that destroy class-specific information while the activations of the Reference Network change in a way that implies the opposite.

Despite this flaw, I managed to train the GAN with WAD to the point when the generated matrices can be used to train a classificator up to the accuracy of 76.7%, which means that WAD is indeed useful for GAN training.

In my future endeavours, I will address the issues regarding to WAD. First, I will examine whether it is possible to use an ensemble of reference networks with properties that minimize the possibility of data perturbations which are WAD currently insensitive to. If appears possible, I will attempt to give theoretical explanations for my experimental findings, and conduct a wider range of tests to find what perturbations are captured well by WAD.

List of Figures

List of Tables

References

[1] Anibal Sólon Heinsfeld, Alexandre Rosa Franco, R Cameron Craddock, Augusto Buchweitz, and Felipe Meneguzzi. Identification of autism spectrum disorder using deep learning and the abide dataset. *NeuroImage: Clinical*, 17:16–23, 2018.

[2] Regina J Meszlényi, Krisztian Buza, and Zoltán Vidnyánszky. Resting state fmri functional connectivity-based classification using a convolutional neural network architecture. *Frontiers in neuroinformatics*, 11:61, 2017.

[3] Hongming Li, Theodore D Satterthwaite, and Yong Fan. Brain age prediction based on resting-state functional connectivity patterns using convolutional neural networks. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 101–104. IEEE, 2018.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] Scott A Huettel, Allen W Song, Gregory McCarthy, et al. *Functional magnetic resonance imaging*, volume 1. Sinauer Associates Sunderland, MA, 2004.

[6] Charles Schaper. Analytic model of fmri bold signals for separable metrics of neural and metabolic activity. *bioRxiv*, page 573006, 2019.

[7] Dongyang Zhang and Marcus E Raichle. Disease and the brain's dark energy. *Nature Reviews Neurology*, 6(1):15, 2010.

[8] David C Van Essen and Donna L Dierker. Surface-based and probabilistic atlases of primate cerebral cortex. *Neuron*, 56(2):209–225, 2007.

[9] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[10] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

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

[13] Paul-Louis Pröve. An introduction to different types of convolutions in deep learning, 2019. 05. 16. url: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d.

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[15] Cathie Sudlow, John Gallacher, Naomi Allen, Valerie Beral, Paul Burton, John Danesh, Paul Downey, Paul Elliott, Jane Green, Martin Landray, et al. Uk biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS medicine*, 12(3):e1001779, 2015.

[16] UK Biobank. Uk biobank brain imaging documentation, 2019. 05. 16. url: https://biobank.ctsu.ox.ac.uk/crystal/docs/brain$_m$ri.pdf.

[17] Mark Jenkinson, Peter Bannister, Michael Brady, and Stephen Smith. Improved optimization for the robust and accurate linear registration and motion correction of brain images. *Neuroimage*, 17(2):825–841, 2002.

[18] Jan Kybic, Philippe Thévenaz, Arto Nirkko, and Michael Unser. Unwarping of unidirectionally distorted epi images. *IEEE transactions on medical imaging*, 19(2):80–93, 2000.

[19] Fidel Alfaro-Almagro, Mark Jenkinson, Neal K Bangerter, Jesper LR Andersson, Ludovica Griffanti, Gwenaëlle Douaud, Stamatios N Sotiropoulos, Saad Jbabdi, Moises Hernandez-Fernandez, Emmanuel Vallee, et al. Image processing and quality control for the first 10,000 brain imaging datasets from uk biobank. *Neuroimage*, 166:400–424, 2018.

[20] Christian F Beckmann and Stephen M Smith. Probabilistic independent component analysis for functional magnetic resonance imaging. *IEEE transactions on medical imaging*, 23(2):137–152, 2004.

[21] Ludovica Griffanti, Gholamreza Salimi-Khorshidi, Christian F Beckmann, Edward J Auerbach, Gwenaëlle Douaud, Claire E Sexton, Enikő Zsoldos, Klaus P Ebmeier, Nicola Filippini, Clare E Mackay, et al. Ica-based artefact removal and accelerated fmri acquisition for improved resting state network imaging. *Neuroimage*, 95:232–247, 2014.

[22] Stephen M Smith, Aapo Hyvärinen, Gaël Varoquaux, Karla L Miller, and Christian F Beckmann. Group-pca for very large fmri datasets. *NeuroImage*, 101:738–749, 2014.

[23] UK Biobank. Fslnets toolbox, 2019. 05. 16. url: https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/fslnets.

[24] UK Biobank. Uk biobank connections between brain regions, 2019. 05. 16. url: https://www.fmrib.ox.ac.uk/datasets/ukbiobank/netjs$_d$100/.