National Research University Higher School of Economics

Faculty of Computer Science
Bachelor's Programme in Data Science and Business Analytics

GROUP TERM PAPER
(Research Project)
< Deep generative models for fast simulation of the tracker at the MPD experiment at NICA>

Prepared by students:
Malchenko Oleg Maksimovich, Group 193 Year 3,
Baminiwatte Arachchiralalage Ranga Nayanakantha, Group 193 Year 3,
Xu Jingtao, Group 193 Year 3

Term Paper Supervisor:
PhD, Leader research fellow, Fedor Ratnikov

Co-supervisor:
PhD, Associate Professor, Artem Maevskiy

Advisor:
PhD, Leader research fellow, Fedor Ratnikov

Moscow
2022

1 Abstract

According to Maevskiy et al.[1], high-energy physics investigations substantially rely on precise detector simulation models in numerous scientific tasks. Still, such simulations are usually very time consuming. Our work is dedicated to search for an improvement to the original generative model based on a Wasserstein Generative Aversarial Network, proposed by A. Maevskiy and his colleagues, which approached to speed up the simulation of the Time Projection Chamber tracker of the Multi-Purpose Detector experiment at the Nuclotron-based Ion Collider Facility accelerator complex in Dubna, Moscow Oblast. Each of the authors of this paper has applied their own method for improving the original model: the first approach was to perform advanced hyperparameter tuning for the WGAN model of Maevsky and his fellow researches; the second approach was to try utilizing a very different generative feed-forward neural network model: Variational Autoencoder, which relies heavily on variational Bayesian inference methods for making simple autoencoders to act as generative models; the third very approach relied on construction of yet another generative model based on Normalizing flows, which are basically a set of bijections that scale or/and translate the target distribution to a standard Gaussian distribution and perform converse bijections from the standard normal sample to generate the estimate of the true distribution. The obtained generative models were then validated and compared to one another in terms of quality of the generated responses, in order to derive the general suggestions on improvement of the original WGAN model.

Contents

2 Introduction

2.1 Relevance and Statement of Purpose

Advance machine learning methods, such as deep learning and machine learning generative models are now widely being applied in adaptation and development of high energy physics experiments due to its effectiveness in processing large amounts of calculated data comprehensively.

This research is a continuation of the attempt of Maevskiy et al.[1] to conduct a deep learning generative model (which we take as our "baseline" or "reference" model), in order to boost the time/quality performance of producing physically-significant pad response simulations, usually obtained from a real Time Projection Chamber (TPC) tracker at a multi-purpose detector, or from some TPC simulators, which are generally very CPU intensive, and hence fast simulation approach for TPC comes in a high demand.

Our main intents in this work were to perform an advanced hyperparameter tuning for the original WGAN model, proposed by Maevsky and his colleagues, and to develop possible substitute models, which could potentially outperform the original model either in terms of accuracy, evaluated by χ^2-metric as a distribution distance measure (explained further), or in terms of time complexity. The "substitute" generative models, which have been considered in our work were based on Variational autoencoders (VAEs), Normalising Flows (NFs) and their further extensions. Both of the above mentioned methods are based on employment of various statistical inference concepts, so that to be able to estimate the population distribution of the data (pad responses) within the feature space from a given fixed-size sample, for example, such as utilization of the evidence lower bound of the true population distribution in case of VAEs or application of the change of variable theorem in case of NFs.

The obtained results on comparative qualities of the models can be further used, in order to create possibly joint generative models along with improving the original one with tuning in better parameters.

2.2 Problem statement and research objective

We must develop two new generative models of different kinds and architectures, and check their generative performance, so that for them to possibly serve as possible substitutes or addendums for the original baseline WGAN generative model of Maevskiy and his research team. Furthermore, we must conduct a research on hyperparameters of the initial mode, so that to suggest a possibly better set of theirs to be further tuned in the main model.

Our respective research objectives are thus:

1. Develop a conditional Variational Autoencoder generative model, which would produce the pad response simulations of the baseline-comparable quality.

2. Develop a conditional Normalizing Flows model, which would produce the pad response simulations of the baseline-comparable quality.

3. Find locally optimal hyperparameters for the baseline model, so that its time/quality performance becomes better, compared to the original parameters-fitted model.

4. Suggest possible ways for further improvement of the reference model, based on the results of our model development and parameter tuning.

2.3 Literature review

Following papers were referred to study the theoretical aspects of various GAN architectures.

- I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *"Generative Adversarial Networks"*, (2014)

- I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, *"Improved training of Wasserstein Gans"*, (2017)

- M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos, *"The Cramer distance as a solution to biased Wasserstein gradients"*, (2017)

Following papers were referred to study the theoretical aspects of VAE's

- D. P. Kingma and M. Welling, *"Auto-encoding variational Bayes"*, (2014) D. P. Kingma and M. Welling, *"An introduction to variational autoencoders"*, (2019)

- P. Baldi, *"Autoencoders, unsupervised learning, and deep architectures"*. In JMLR: Workshop and Conference Proceedings 27:37–50, (2012)

- D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *"Learning internal representations by error propagation"*. In Parallel Distributed Processing. Vol 1: Foundations. MIT Press, Cambridge, MA, (1986)

- C. Doersch, *"Tutorial on Variational Autoencoders"*, (2021)

- D. Bank, N. Koenigstein, and R. Giryes, *"Autoencoders"*, (2021)

- A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, *"Autoencoding beyond pixels using a learned similarity metric"*, (2016)

- I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, *"Beta-VAE: Learning basic visual concepts with a constrained variational framework"*. In Proceedings of the International Conference on Learning Representations, (2016)

- L. Dinh, D. Krueger, and Y. Bengio, *"NICE: Non-linear Independent Components Estimation"*, (2015)

Following papers were referred to study the theoretical aspects of Normalising flows

- L. Dinh, J. Sohl-Dickstein, and S. Bengio, *"Density Estimation Using Real NVP"*, (2017)

- G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, *"Normalizing flows for probabilistic modeling and inference"*, (2021)

- G. Papamakarios, T. Pavlakou, and I. Murray, *"Masked autoregressive flow for density estimation"*, (2018)

- H. Reyes-Gonzalez and R. Torre, *"Testing the boundaries: Normalizing flows for higher dimensional data sets"*, (2022)

- PhD Nathan C. Frey, *"FastFlows: Flow-based models for Molecular Graph Generation"* In Medium, (2022)

- H. Wen, P. Pinson, J. Ma, J. Gu, and Z. Jin, *"Continuous and distribution-free probabilistic wind power forecasting: A conditional normalizing flow approach"*, (2022)

- P. Wielopolski, M. Koperski, and M. Zięba, *"Flow Plugin Network for Conditional Generation"*, (2021)

2.4 Related Work

In the paper [1] (May referred as original paper), a concept is proposed to model the raw electronics responses from TPC tracker of the MPD experiment at the NICA accelerator complex. This model utilize the symmetries of the detector to reduce the dimensionality of the learned representations.The generation Process is conditioned on parameters on small track segments rather than parameters of the whole track at the production point.

The proposed model (May referred as original model)uses a GAN architecture with a Wasserstein distance based objective function, a custom activation function, convolution network for discriminator and a fully-connected network for the generator.

In this model the custom activation function in the outer layer helps to mitigate the problem of

GAN failing to describe steep cutoffs when target space contains 0 values. The activation function is given by the following formula -

$$f(x) = \begin{cases} \alpha T e^{x}; & x < 0 \\ T(\alpha + (1-\alpha)\frac{x}{\delta}); & 0 \leq x \leq \delta \\ T - \delta + x; & \delta \leq x \end{cases} . \tag{1}$$

Here $\delta = 0.01$, $\alpha = 0.1$ and $T = log10_2$ is a threshold.

Overall, the proposed model in this paper demonstrated a promising performance for the most of the metrics considered.

3 Methodology

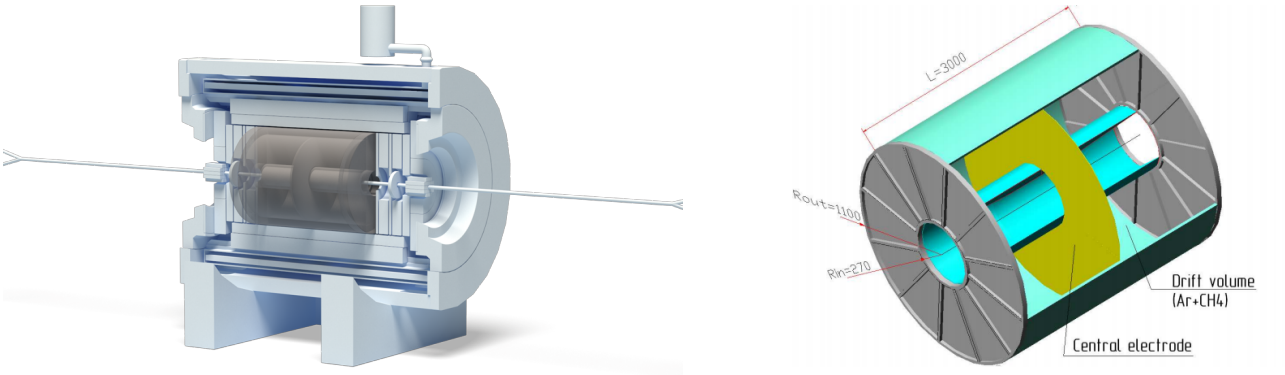
3.1 Approach

In order to improve the simulation process of time projection chamber responses we used 3 approaches,

1. Generative Adversarial Network based approach.
2. Variational Autoencoder based approach.
3. Normalising Flow based approach.

For each approach research was done individually and then later compared with the model proposed in the original paper.

3.2 Data description



(a) 3D model: The MPD time projection chamber

(b) Schematic view of the MPD TPC

Figure 1: Source - Nuclotron-based Ion Collider facility Official website

According to Maevskiy et al. [1], the training dataset is generated by the detailed MPD simulator, which is based on Geant3 transport of particles through the detector materials and realistic simulation of the detector responses based on the first principles.

We have four features that describe the track segment crossing a particular pad plane:

- crossing angle
- dip angle
- pad coordinate
- drift length

It should be noted that it is sufficient to enter only the fractional part of the pad coordinate into the model for first 3 features but for drift length, the characteristics responses depends on their absolute values Therefore, both the fractional part and the complete number are entered to the model as two

separate features making the overall number features to 5.

The data set contained 20,000 segment responses that were taken from detailedMPD simulator, which were split into 75:25 training and validation subsets respectively. The same proportions were used for all model training and validation in this research.

3.3 Evaluation Metrics

To make a more precise quantitative evaluation, the authors of the original paper[1] use a set of metrics. As it was mentioned in the data section, target space depends on 5 features(5 conditional parameters). The goal is to make sure the distributions of these parameters generated by the model agree with training data generated by the TPC simulator for any possible set of conditional parameters.A function of input variables to make comparisons between generated and validation data. For each input image, 1st order moments, i.e. the pad and time coordinates of the response distribution barycenter, the 2nd order moments, i.e. the squared widths of the response distribution and covariance between the pad and time coordinates, and the integrated amplitude are calculated.
Figure 2 depicts the profiles of validation metric distributions as a function of the input variables for real data. For each metric ϵ , it shows the average value μ_ϵ (middle line), and well as the average shifted up and down by a standard deviation of the metric distribution $\mu_\epsilon \pm \sigma_\epsilon$ (top and bottom lines). Line thickness denotes the statistical uncertainty of the corresponding value.

Chi-square($\widetilde{\chi^2}$) : In order to calculate how well these two distributions agree, we look at distributions as one-dimensional projections. We divide the distributions of the conditional variables in to small intervals (n bins) and for each bin take average and standard deviations from both real and generated samples. Then the sum and difference of average and standard deviation are obtained. This procedure is done for every single target variable versus every single conditional variable - each time all the other parameters are marginalized. Chi square($\widetilde{\chi^2}$) is calculated by summing up the difference between the obtained values from the two samples and divide by the sum of the error squares for n bins and then the result is summed over all target-condition combinations. The errors are calculated with bootstrap method .Here we assume all single target-condition pairs are independent and errors are normally distributed. However in reality, the sum of target-condition combinations cannot be independent because components of the sum are correlated with one another.
The mathematical steps of calculating Chi-square($\widetilde{\chi^2}$) are as follows-

Features:

$$X = \{x_1, \ldots, x_n\}, \quad x_i = (x_i^1, x_i^2, x_i^3, x_i^4, x_i^5)^T$$

Targets:

$$Y_X = \{y_1, \ldots, y_n\}, \quad y_i \sim p^{real}(\cdot|x_i), \quad y_i = (y_i^1, y_i^2, y_i^3, y_i^4, y_i^5)^T$$

Predicted targets:

$$\hat{Y_X} = \{\hat{y_1}, \ldots, \hat{y_n}\}, \quad \hat{y_i} \sim p^{gen}(\cdot|x_i), \quad \hat{y_i} = (\hat{y_i}^1, \hat{y_i}^2, \hat{y_i}^3, \hat{y_i}^4, \hat{y_i}^5)^T$$

Per interval statistics, given target component j, feature component k and feature component interval I:,

$$\mu_I^{jk} = sample\ mean(\{y_i^k | x_i^k \in I\}), \quad \hat{\mu_I}^{jk} = sample\ mean(\{\hat{y_i}^k | x_i^k \in I\})$$

$$\sigma_I^{jk} = sample\ std(\{y_i^k | x_i^k \in I\}), \quad \hat{\sigma_I}^{jk} = sample\ std(\{\hat{y_i}^k | x_i^k \in I\})$$

$$u_I^{jk} = \mu_I^{jk} + \sigma_I^{jk}, \quad l_I^{jk} = \mu_I^{jk} - \sigma_I^{jk}$$

$$\hat{u_I}^{jk} = \hat{\mu_I}^{jk} + \hat{\sigma_I}^{jk}, \quad \hat{l_I}^{jk} = \hat{\mu_I}^{jk} - \hat{\sigma_I}^{jk}$$

Chi- Square statistic is given by,

$$\widetilde{\chi^2} = \sum_{j,k,I} [\frac{(u_I^{jk} - \hat{u_I}^{jk})^2}{\epsilon^2_{u_I^{jk}} + \epsilon^2_{\hat{u}_I^{jk}}} + \frac{(l_I^{jk} - \hat{l_I}^{jk})^2}{\epsilon^2_{l_I^{jk}} + \epsilon^2_{\hat{l}_I^{jk}}}]$$

Here ϵ_x is standard error for x.

In the original model proposed in [Maevskiy et al] which would also act as the baseline model in this research, would reach an average $\widetilde{\chi^2}$ value of $440.33(\pm 27.43)$ after training for 4000 epochs

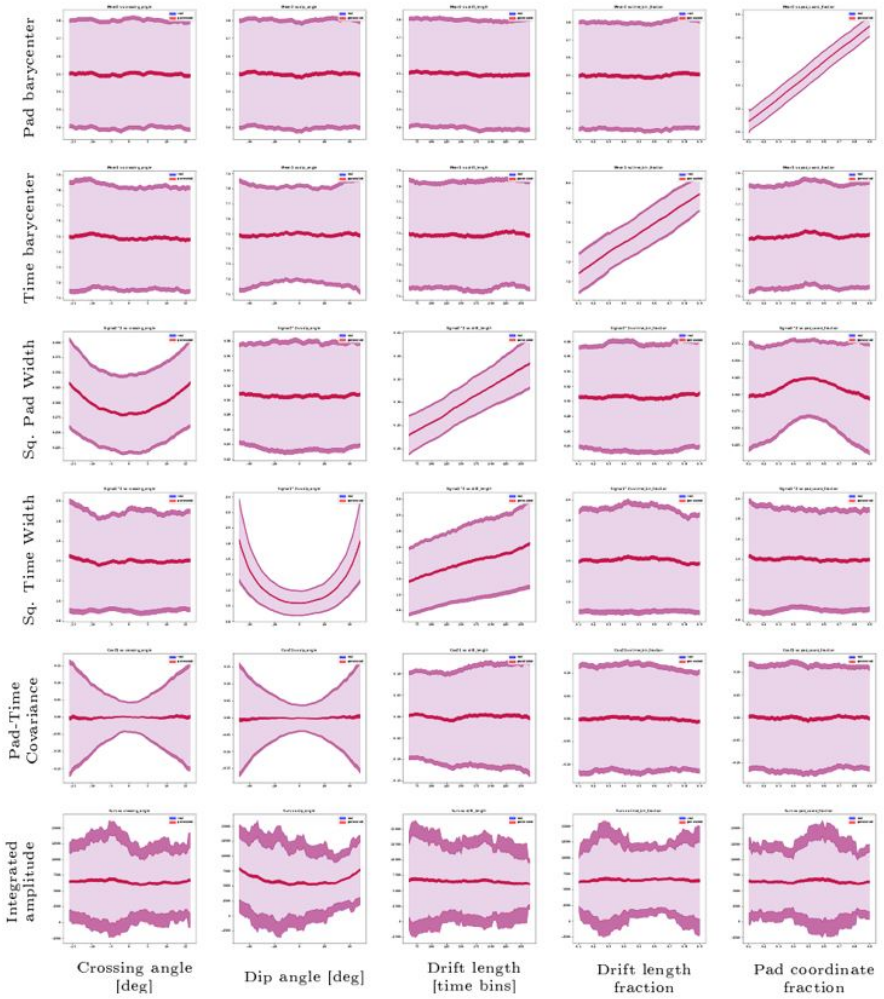


Figure 2: Validation metric distributions as a function of the input variables. For each metric ϵ , it shows the average value μ_ϵ (thick line in the middle), as well as the average plus/minus standard deviation of the metric distribution $\mu_\epsilon \pm \sigma_\epsilon$ (top and bottom lines)

3.4 Development Tools

All the programming components in this research paper were done using python3 development environment with TensorFlow 2.1 framework. Furthermore, computationally intensive models were trained with the help of GPU's provided on remote computers at HSE faculty of Computer Science.

4 Generative Adversarial Networks (GAN)

4.1 Background

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset.
There were 3 types of GAN variants considered in the optimization process. They are namely,

1. Vanilla GAN
2. Wasserstein GAN (With Gradient Penalty)
3. Cramer GAN

4.1.1 Vanilla GAN

This is the most basic form of Generative adversarial network there is. A typical GAN consists of two major components. The generator and the discriminator. The purpose of Generator is to generate fake data samples based on the given input and try to manipulate the Discriminator. At the same time, the Discriminator's job is to distinguish between real and fake data. Both Generator and Discriminator are Neural Networks which run simultaneously when training the model. The more steps (epochs) the model is trained, the better the quality of Generator and discriminator functions. The diagram shows the basic workflow of a GAN:

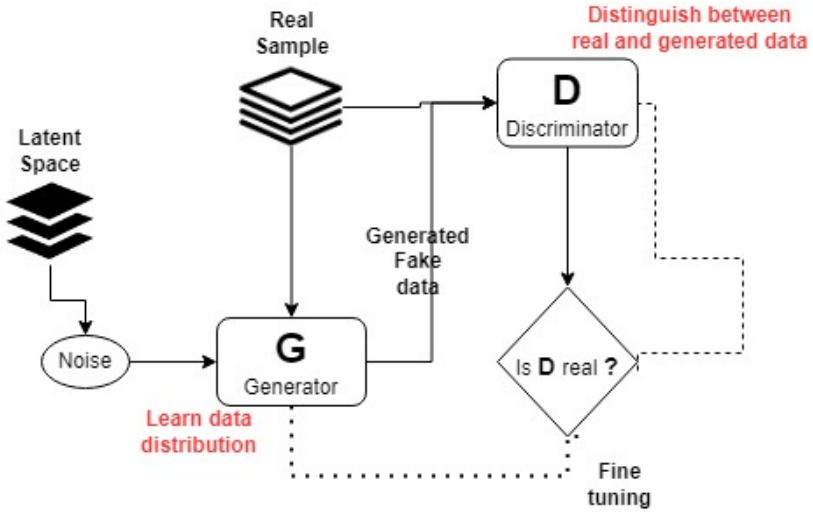


Figure 3: Basic workflow of a GAN

Here, the Generator learns the distribution of data and trained so that probability of Discriminator not realizing fake data is maximized. The Discriminator uses a model that calculates the probability

that the sample that it got is received from the real data and not generated data. The GANs follows a sort of minimax game which, the Generator is trying to reduce the Discriminator's reward (i.e. maximize its loss) Discriminator is trying to minimize its reward $L(D, G)$. Loss function of GAN is given by the following equation:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{2}$$

Here,

$G-Generator, D-Discriminator, P_{data}(x)-distribution\ of\ real\ data, P_z(z)-distribution\ of\ generator, x-sample\ from\ P_{data}(x), z-sample\ from\ P(z), D(x)-Discriminator\ network, G(z)-Generator\ network$

A full description about the functionality of GAN could be found in the original paper[2] given in the reference.

4.1.2 Wasserstein GAN (With Gradient Penalty)

Despite the vanilla GAN model has been successful particularly in image generation there are some flaws in the model. During the training generator may breakdown to a mode where it always reproduces same outputs (collapse of Mode). Another problem is the vanishing gradient (when the discriminator is too perfect the loss function $L(D, G)$ may fall to zero and there won't be a gradient to update the loss during learning iterations) Some of these problems are mitigated by introducing Wasserstein distance which calculates distance between two probability distributions better than the JS divergence method used in vanilla GAN to measure distance between two probability distributions since it provides a meaningful and smooth representation of the distance in-between distributions even when two distributions are in lower dimensions without overlapping. Original Wasserstein distance calculation contains a constrained Lipschitz function. This constraint is enforced via a gradient penalty term. The value function(2) and the gradient penalty(3) of WGAN are as follows.

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)}[D(x)] - \mathbb{E}_{z \sim p_z(z)}[D(G(z))] \tag{3}$$

$$L = \mathbb{E}_{z \sim p_z(z)}[D(G(z))] - \mathbb{E}_{x \sim p_{\text{data}}(x)}[D(x)] + \underbrace{\lambda * \mathbb{E}_{\hat{x} \sim p_{\hat{x}}(\hat{x})}[(||\bigtriangledown_{\hat{x}} D(z)||_2 - 1)^2]}_{Gradient Panelty} \tag{4}$$

Here $p_{\hat{x}}$ is uniform sample along straight lines between pairs of points sampled from real and generator data distribution.λ is the coefficient of gradient penalty. A full description about the functionality

of WGAN(with GP) could be found in the original paper[3] given in the reference.

4.1.3 Cramer GAN

One of the main problems in using Wasserstein distance is that samples sometimes yield biased gradients which may lead to a wrong minimums in the loss function. In a paper published in 2018, Cramer GAN is introduced which uses Cramer distance that has the same appealing properties as the Wasserstein distance, but also provides unbiased sample gradients.
The Cramer distance between two distributions P and Q is given by,

$$l_p(P,Q) = \sup_{f \sim F_q} |\mathbb{E}_{x\sim P} f(x) - \mathbb{E}_{x\sim Q} f(x)| \tag{5}$$

Here, $F_q = [f : f$ is absolutely continuous, $||\frac{\partial f}{\partial x}|| \leq 1]$, here q is the conjugate exponent of p
Cramer GAN loss functions are as follows,
For a real sample x_r P, and two independent generator samples x_g, x'_g Q, Interpolation of real and generated sample is given by $\hat{x} = \epsilon x_r + (1-\epsilon)x_g$
The loss function of discriminator(L_{critic}) is given by,

$$L_{critic} = -(f(x_r) - f(x_g)) + \lambda(||\bigtriangledown_{\hat{x}} f(\hat{x}||_2 - 1)^2 \tag{6}$$

The loss function of generator(L_g) is given by,

$$L_g = ||h(x_r) - h(x'_g)|| + ||h(x_r) - h(x'_g)|| - ||h(x_g) - h(x'_g)|| \tag{7}$$

Here, h is a transformation function which transforms input dimensionality into 256. The generator minimises the crammer distance between transformed variables from real and generated samples.
Full description about the functionality of Cramer GAN could be found in the original paper[4] given in the reference below.

4.2 Task Description

The core objective of this research task is to explore the possibility speeding up the GAN based TPC simulation process via reducing the number of calculations in the model. Also, the possibility of achieving a similar performance to original model based on Wasserstein GAN using alternative GAN architectures, particularly Vanilla GAN and Cramer GAN is explored.

4.3 Prerequisites

4.3.1 Parameters of the GAN models

As mentioned earlier in the description of GANs, it has an algorithmic architecture that combines two neural networks. Therefore, in order to speed up simulation process, some parameters of the neural networks and the GAN architecture were tuned. The following are the parameters of the GAN model which were altered to check how the model would perform for each configuration.

- Learning Rate - One of the most important hyper-parameters of GAN. It is the step size at each epoch(iteration) when minimizing the loss function. For each epoch it is decreased by a factor. Normally takes a value between 0 and 1.
- Dimensions of the Latent Space - The dimensions of the hypothetical space that contains points that represent images so that the Generator could convert a point from the latent space to an image based on the training data set.
- Hidden Layer Configuration - The configuration of the intermediate layer in between input layers and output layers of a neural network, the complexity of the neural network increases when size of the layers increase. Here we focus only on the hidden layers of the generator which has 5 layers.
- Gradient Penalty - Gradient Penalty is a constraint on gradients of the critic's output with respect to inputs to have unit norm. It's weighted by a penalty coefficient λ

4.3.2 Testing Environment

Model design and training is done using the TensorFlow 2.1 framework. The performance of the model is observed using tensor board visualization tool. The computation time depends on architecture and parameters of GAN model as well as the number of GPU's allocated for training the model. At minimum approximately 12 -13 hours to train a model for 4000 epochs.

4.3.3 Baseline model

In order to compare model performances for different configurations, first we establish a baseline model which uses same configurations presented in the original model. It uses a Wasserstein GAN architecture with following parameters.

Learning rate - 0.0001(and decreasing by a factor of 0.999 for each epoch), Latent space dimension - 32, Gradient penalty - 10, Hidden Layer Configuration - $[32, 64, 64, 64, 128]$

Also, it should be noted that generator and discriminator uses RMSprop optimizer method and for each generator step 8 discriminator update steps are made. The model uses a batch size of 32. The

performance of the model is calculated with Chi-Square($\widetilde{\chi^2}$) value the model will reach after running 4000 epochs.

4.4 Models' results and comparison

One of the most important realisations of tuning different parameters was that performance of the models with same configuration even may vary quite significantly. Therefore in order to get an estimate of the average $\widetilde{\chi^2}$ after 4000 epochs each test was run 3 times and mean value was taken. Standard Error of the results were also calculated.

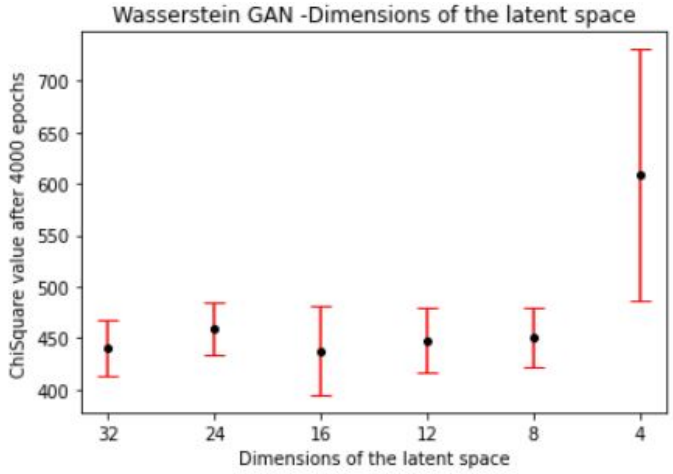
It should be noted that only the results from single test are depicted in the Chi-square plot in order to depict how $\widetilde{\chi^2}$ value changed over the epochs.

4.4.1 Wasserstein GAN - According to Dimension of the latent space

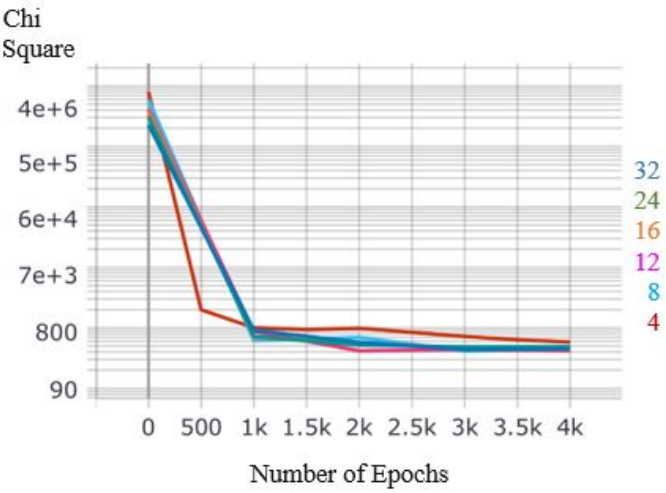
Wasserstein GAN -According to Dimension of the latent space (after 4000 epochs)					
Dimensions of the latent space	Chi-Square TEST 1	Chi-Square TEST 2	Chi-Square TEST 3	Average Chi-Square	Standard Deviation
32	459.5	408.9	452.6	440.33	27.43
24	440.9	448.6	489.1	459.53	25.89
16	388.1	460.7	464.2	437.66	42.96
12	481.1	418.5	444.0	447.86	31.47
8	423.2	481.0	446.4	450.2	29.08
4	582.8	501.3	741.5	608.53	122.15

Table 1: Wasserstein GAN -According to Dimension of the latent space Test results

Here dimension of the original model (32) is reduced by multiples of 4 while keeping other parameters constant. Following plots (Figure 2) are based on the results shown in Table 1. According to the stats, the model performance would not significantly deviate from the baseline model despite the reduction of the dimensions until size 8. After dimension is reduced to 4, there is a significant drop in $\widetilde{\chi^2}$ value



(a) 'Average $\widetilde{\chi^2}$ values according to dimensions of latent space'



(b) Change of $\widetilde{\chi^2}$ value over 4000 epochs

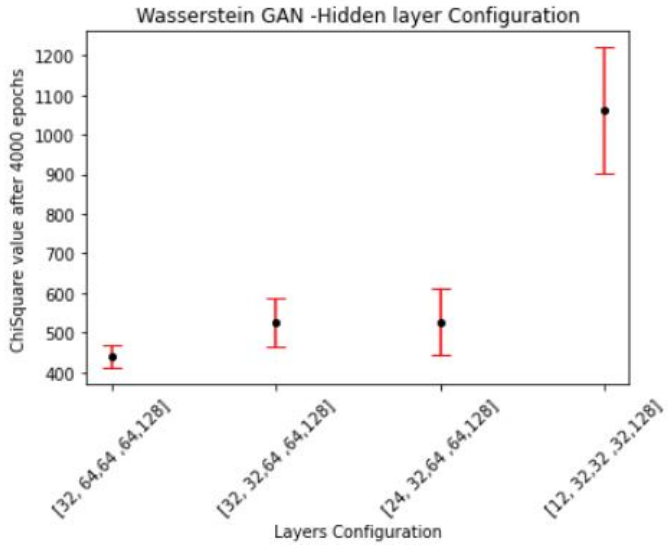
Figure 4

4.4.2 Wasserstein GAN - According to hidden layer configuration

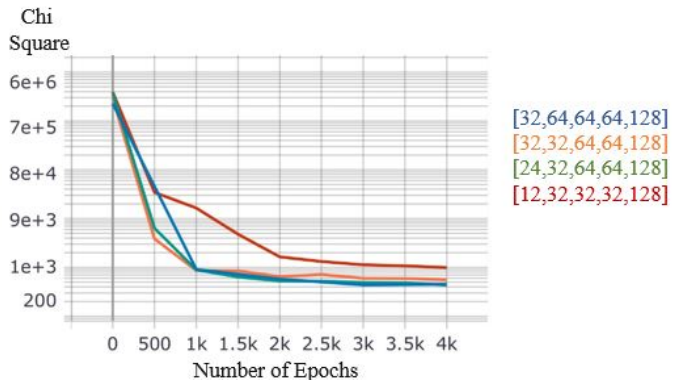
Wasserstein GAN - Changing hidden layer configuration (after 4000 epochs)					
Layers Configuration	Chi-Square TEST 1	Chi-Square TEST 2	Chi-Square TEST 3	Average Chi-Square	Standard Deviation
$[32, 64, 64, 64, 128]$	459.5	408.9	452.6	440.33	27.43
$[32, 32, 64, 64, 128]$	561.7	455	561.7	526.13	61.60
$[24, 32, 64, 64, 128]$	550.9	435.2	597	527.7	83.35
$[12, 32, 32, 32, 128]$	992.1	1243	946.3	1060.46	159.72

Table 2: Wasserstein GAN - Changing hidden layer configuration Test results

Here hidden layer configuration of the generator is reduced in terms of size of each layer keeping other parameters constant. As we know, when the sizes of layers reduce, the number of calculations reduces so in principle, generation should speed up. The compromise between speed and model performance is checked here. Following plots (Figure 3) are based on the results shown in Table 2. According to the plots,a significant deviation from the baseline model performance happens when layer configuration is reduced to $[12, 32, 32, 32, 128]$



(a) Average $\widetilde{\chi^2}$ values according hidden to layer configuration



(b) Change of $\widetilde{\chi^2}$ value over 4000 epochs

Figure 5

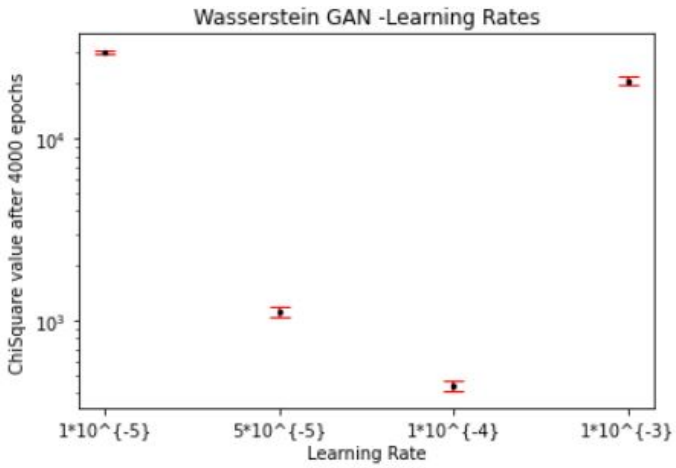
4.4.3 Wasserstein GAN - According to Learning Rate

Wasserstein GAN - Changing Learning Rate (after 4000 epochs)					
Learning Rate	Chi-Square TEST 1	Chi-Square TEST 2	Chi-Square TEST 3	Average Chi-Square	Standard Deviation
$1 * 10^{-5}$	$3.03 * 10^4$	$2.99 * 10^4$	$2.93 * 10^4$	$2.98 * 10^4$	503.322
$5 * 10^{-5}$	1165	1038	1175	1126	76.374
$1 * 10^{-4}$	459.5	408.9	452.6	440.33	27.43
$1 * 10^{-3}$	$2.01 * 10^4$	$2.20 * 10^4$	$1.99 * 10^4$	$2.07 * 10^4$	1139.78

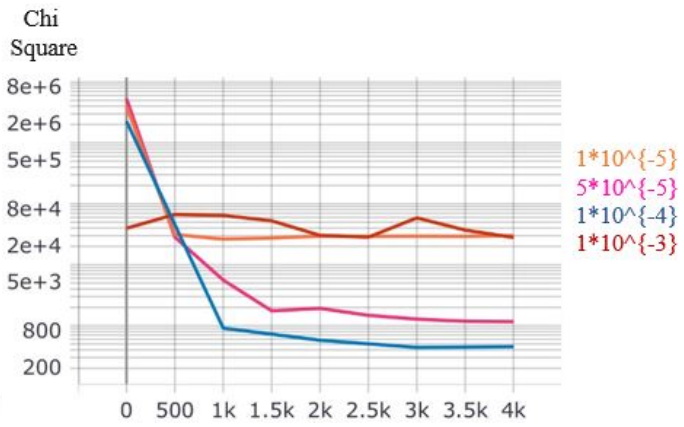
Table 3: Wasserstein GAN - Changing Learning Rate Test results

Here learning rate of the GAN model is both increased and decreased from original learning rate($1 * 10^{-4}$)in order to find the most optimal learning rate holding other parameters constant. Following plots (Figure 4) are based on the results shown in Table 3. According to the plots,optimal learning rate should be somewhere closer to $1 * 10^{-4}$ as changing the rate significantly from that point would worsen the performance of the model.

Note- Plots use log scale on the y-axis



(a) Average $\widetilde{\chi^2}$ values according to learning rate



(b) Change of $\widetilde{\chi^2}$ value over 4000 epochs

Figure 6

4.4.4 Vanilla GAN - According to Gradient Penalty

Here, the performance of the simulation model is tested when original model based on Wasserstein GAN is replaced by Vanilla GAN architecture. In the initial stage of testing it was realised that Vanilla GAN model would perform much better if a gradient penalty term added to loss function. The remaining parameters are kept similar to baseline model.

Note - It was observed that in some tests(specially with lower gradient penalty) the lower $\widetilde{\chi^2}$ values were achieved in in earlier than 4000^{th} epoch. Therefore, both $\widetilde{\chi^2}$ value at 1000^{th} epoch and 4000^{th} epoch are tested.

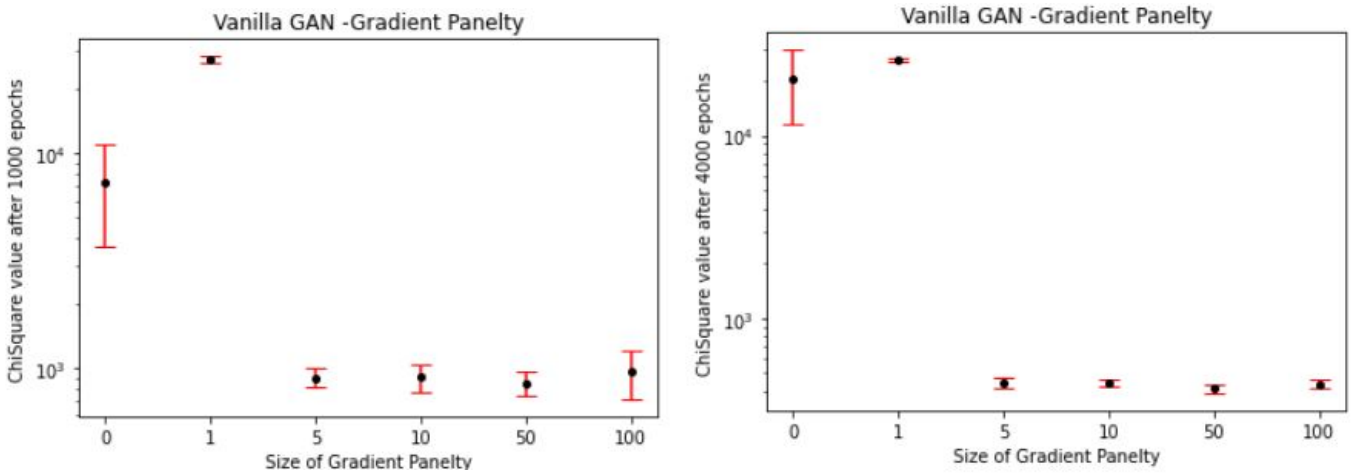
Vanilla GAN - Changing Gradient Penalty (after 1000 epochs)					
Gradient Penalty	Chi-Square TEST 1	Chi-Square TEST 2	Chi-Square TEST 3	Average Chi-Square	Standard Deviation
0	4057	6602	11239	7299.33	3641.42
1	28445	26712	26493	27216.66	1069.38
5	901.6	809.9	993.3	901.6	91.7
10	997	756.9	955.7	903.2	128.37
50	730.7	863.6	952.6	848.96	111.67
100	1240	764.1	886.7	963.6	247.09

Table 4: Vanilla GAN - Changing Gradient Penalty Test results (1000 epochs)

Vanilla GAN - Changing Gradient Penalty (after 4000 epochs)					
Gradient Penalty	Chi-Square TEST 1	Chi-Square TEST 2	Chi-Square TEST 3	Average Chi-Square	Standard Deviation
0	12670	30359	18557	20528.67	9007.8
1	26624	26647	25535	26268.66	635.47
5	416.1	430.8	473.6	440.16	29.87
10	448.7	411.9	442.7	434.43	19.74
50	410.1	389.4	434.5	411.33	22.575
100	463.9	421.2	419.8	434.96	25.06

Table 5: Vanilla GAN - Changing Gradient Penalty Test results (4000 epochs)

Following plots (Figure 5) are based on the results shown in Table 4 and Table 5. According to the plots, increasing the gradient penalty would improve the performance of the model when penalty is greater than 1 . It is noteworthy that initially $\widetilde{\chi^2}$ value increases before it starts starts decreasing along the Gradient Penalty. The best $\widetilde{\chi^2}$ value is achieved when penalty is equal to 50.
Note- Plots use log scale on the y-axis.



(a) Average $\widetilde{\chi^2}$ values (1000 epochs)

(b) Average $\widetilde{\chi^2}$ values (4000 epochs)

Figure 7

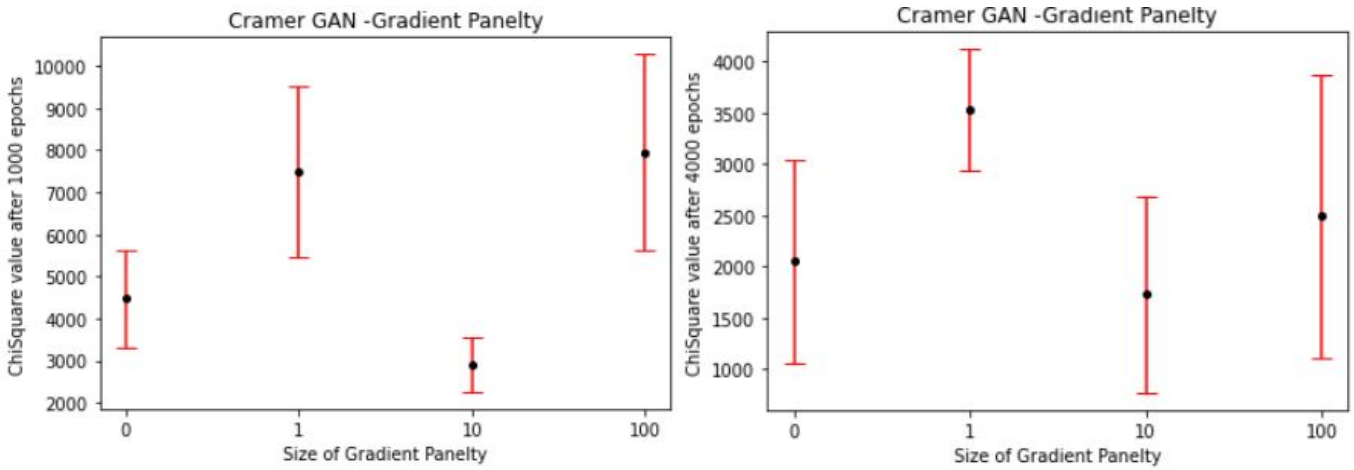
4.4.5 Cramer GAN - According to Gradient Penalty

Here, the performance of the simulation model is tested when it uses Cramer GAN architecture according to size of gradient penalty, while remaining parameters are kept similar to baseline model. Like in the earlier section with Vanilla GAN, both $\widetilde{\chi^2}$ value at 1000^{th} epoch and 4000^{th} epoch are tested.

Cramer GAN - Changing Gradient Penalty				
Gradient Penalty	Average Chi-Square (1000 epochs)	Standard Deviation	Average Chi-Square (4000 epochs)	Standard Deviation
0	4466.66	1157.19	2048.93	986.26
1	7497	2025.92	3530	590.22
10	2913	657.43	1727.66	957.69
100	7942.33	2339.1	2492.66	1377.32

Table 6: Cramer GAN - According to Gradient Penalty Test results

Following plots (Figure 6) are based on the results shown in Table 6. According to the plots, in contrast to Vanilla GAN based model, increasing the gradient penalty would not necessarily improve the performance of the model.The best $\widetilde{\chi^2}$ value is achieved when penalty is equal to 10. Overall, performance of the model is poor compared to Vanilla GAN and Wasserstein GAN models.
Note- Plots use log scale on the y-axis.



(a) Average $\widetilde{\chi^2}$ values (1000 epochs)

(b) Average $\widetilde{\chi^2}$ values (4000 epochs)

Figure 8

4.5 Summary

By testing the performance of TPC response simulation via GANs with different architectures and hyper parameters, it was evident that there is a possibility to improve the model proposed in the original paper as well achieve similar results via alternative GAN architectures.

Key takeaways -

- In comparison to the baseline model, similar model performance could be achieved with Vanilla GAN based models a with gradient penalty constraint. Figure 8 shows evaluations metrics of such a model which uses Vanilla GAN architecture with a gradient penalty of 50. The remaining parameters are kept similar to baseline model. Figure 7 shows the true test signal comparison with the generated ones from this model.

- Hyper parameter tuning on the baseline model such as reducing latent space dimensions to 16 will slightly improve the model. Also reducing hidden dense layer to a certain extent would not affect the model performance significantly.

- Model performance with Cramer GAN was also tested. Despite showing acceptable results, it performs slightly weaker to Wasserstein and Vanilla GAN based models. Also Cramer based GAN is relatively slow in Image generation compared to other due to more computationally complex loss functions.There is possibility to improve Cramer model with with further hyper parameter tuning and optimisation.

- From the evaluation results obtained for different models it was evident that even for same configurations model performance may vary (high standard deviations). Therefore, it is necessary to consider the unstable nature of GANs when comparing the models.

- One of the key drawbacks of GANs are the long computation times and advanced hardware requirements for a smooth generation process. Possible alternative methods such VAEs and Normalising flow methods may give faster results.

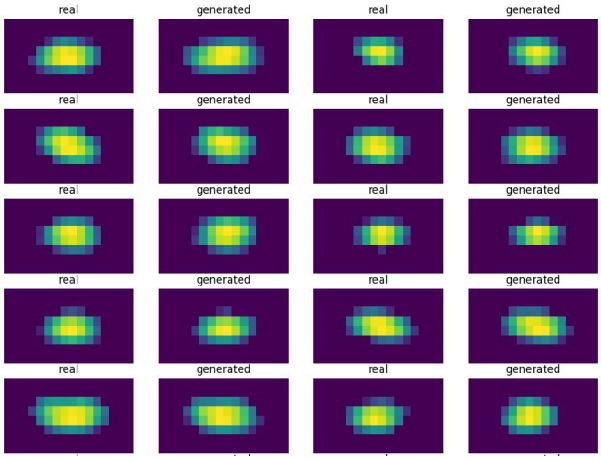


Figure 9: Comparison of true test signals and Vanilla -GAN-generated ones

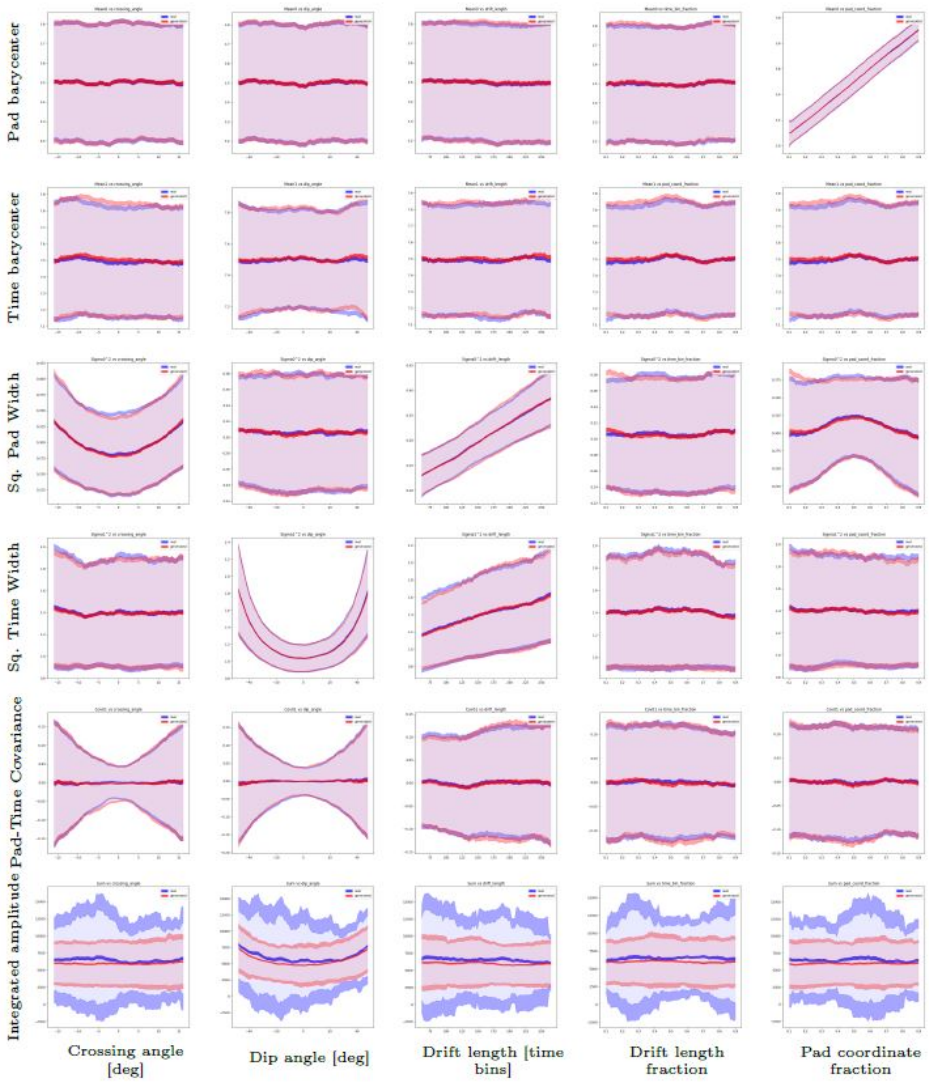


Figure 10: Evaluation metrics of Vanilla GAN based model with a gradient penalty(50) after training for 4000 epochs.

5 Variational Autoencoders

5.1 Background

5.1.1 What an autoencoder is

Another widely applied class of generative neural networks can be introduced by Gaussian variational autoencoders or simply variational autoencoders (VAEs), being an extension of simple autoencoders (AEs).

According to Baldi [7], autoencoders were first introduced in the 1980s by Hinton and the PDP group [8] to address the problem of "backpropagation without a teacher", by using the input data as the teacher. Autoencoders thus can be defined as simple learning circuits that aim to convert inputs into outputs with the least amount of distortion feasible [7].

Let us discuss the mathematical basis of a simple autoencoder, and then explain why the "plain" version of it can hardly be used as a generative model as in the scope of our initial problem.

5.1.2 Mathematical basis of simple AEs and associated problems

Let us first assume that we are given an encoder-decoder task, so that provided with a set of n vectors having some distribution in the feature space $\mathbb{R}^m$: $\mathbf{x} = \{\mathbf{x}_i\}_{i=1}^m \in \mathbf{X} = \mathbb{R}^{n \times m}$, so that $\mathbf{x}_i = (x_{[i,1]}, \dots, x_{[i,n]})^\mathsf{T}$, we are aiming to estimate such $\{\boldsymbol{\phi} : \mathbf{X} \to \mathbf{Z}\}$, and $\{\boldsymbol{\theta} : \mathbf{Z} \to \mathbf{X}\}$ that having $\mathbf{Z} = \mathbb{R}^{n \times (d \ll m)}$ being some "compressed" latent dimension of smaller dimensionality than that of $\mathbf{X}$, we have an objective of minimizing Euclidean distance between the reconstructed and original inputs, that is [10]:

$$\boldsymbol{\phi}, \boldsymbol{\theta} = \underset{\boldsymbol{\phi}, \boldsymbol{\theta}}{\operatorname{argmin}} \|\mathbf{X} - (\boldsymbol{\theta} \circ \boldsymbol{\phi})\mathbf{X}\|_2^2$$

In case of solving the encoder-decoder task in terms of a neural network conduction, we can define $\boldsymbol{\phi}$ as an encoder and $\boldsymbol{\theta}$ as a decoder, so that having some decoder-produced reconstruction of the initial input $\mathbf{x}$: $\mathbf{x}' \in \mathbf{X}$, we define the objective function of the AE-nn as a squared errors function between the inputs and produced reconstructions:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2^2$$

The goal of autoencoders here is to simply reconstruct the initial image from the compressed latent representation with minimal information loss, hence their primary objective can be associated with dimensionality-reduced reconstruction, what in fact helps in case of outlier detection, noise reduction or input "smoothening" tasks [10]. However, reasonable feature sampling from the latent space will be difficult, since the mapping of the encoder function to the latent space is not regularized: it is not going to be easy to assess how the input features are encoded within the latent space, as such encoding is implicit, and thereof simple autoencoders can hardly serve as generative models, since

the latent representation of $\mathbf{X}$ will vary from task to task, and every time a new sampling technique has to be introduced, so that to make the decoder produce some reasonable output from the "fake" latent sample.

Variational autoencoders, however, deal with such an issue by regularising the encoder $\boldsymbol{\theta}$ to have features being mapped to $\mathbf{Z}$ with an intent to approximately follow some interpretable distribution of theirs in $\mathbf{Z}$, for example the standard Gaussian distribution: $\mathcal{N}(0, 1)$, so that feature sampling from $\mathbf{Z}$ becomes way easier [9].

5.2 About variational autoencoders and their mathematical basis

5.2.1 Variational Bayesian inference as a basis of VAEs

Variational autoencoders can be considered a regularized extension of simple AEs, utilizing the concepts of variational Bayesian inference, where we aim to approximate the posterior distribution of $\mathbf{z} \in \mathbf{Z}$ given $\mathbf{x} \in \mathbf{X}$ by having a restricted to some known prior distribution: $p_\theta(\mathbf{z})$.

Let us say that given some sample of $\mathbf{x} \in \mathbf{X}$, we have that, with $\mathbf{z} \in \mathbf{Z}$ being some set of latent variables, sampled from the latent space:

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int_{\mathbf{z}} p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$$

where $p_\theta(\mathbf{z})$ is some fixed prior distribution of $\mathbf{z}$. Usually, as we have already discussed in **4.1.2** the true posterior density of $\mathbf{z}|\mathbf{x}$ is intractable, and hence according to the variational framework is to be approximated by some parametric rather more simple inference model: $q_\phi(\mathbf{z}|\mathbf{x})$. That is we are trying to approximate the real posterior distribution of the latent space by some simple posterior $q_\phi(\mathbf{z}|\mathbf{x})$, with a fixed known prior, so that, in case of standard Gaussian variational inference, we have [6]: $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbb{I})$, $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_\theta(\mathbf{z}), \boldsymbol{\sigma}^2_\theta \mathbb{I})$ and $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}^2_\phi(\mathbf{x})\mathbb{I})$ where $\mathbb{I}$ is the identity matrix, assuming standard normal distribution as a prior for the latent space, with $\boldsymbol{\sigma}^2_\theta$ being fixed $\forall \mathbf{z}, \theta$.

5.2.2 The Evidence Lower Bound (ELBO)

That is with some fixed known prior of $\mathbf{z}$ we want to approximate the intractable posterior distribution of the latent space by $q_\phi(\mathbf{z}|\mathbf{x})$. Now, as we are trying to approximate the posterior distribution, let

us introduce the variational lower bound inequality, which states that [5]:

$$\ln p_\theta(\mathbf{x}) = \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x})\right] \tag{8}$$

$$= \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] = \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})\cdot q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})\cdot q_\phi(\mathbf{z}|\mathbf{x})}\right] \tag{9}$$

$$= \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] + \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \text{ (as KL-divergence)} \tag{10}$$

$$\geqslant \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \text{ (since } \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] = D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))) \geqslant 0) \tag{11}$$

where the $\mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$ term is called the variational lower bound or evidence lower bound (ELBO) of the log-likelihood of $\mathbf{x}$. The above inequality suggests that the Kullback-Leibler divergence between two distributions (the desired posterior and inference-constructed posterior) is minimized when ELBO is maximized w.r.t. ϕ and θ, as far as the original problem requires maximization of the intractable log-likelihood of $\mathbf{x}$.

5.2.3 Defining a VAE

From that, interpreting the population distribution estimation with posterior approximation as an encoder-decoder task, we can define it as being solvable in terms of a neural network with $q_\phi(\mathbf{z}|\mathbf{x})$ as encoder-generated latent space posterior distribution, and $p_\theta(\mathbf{x}|\mathbf{z})$ as decoder-produced posterior distribution of the original input data, given the latent space sample. Now we can define a variational autoencoder as a simple AE with a regularized objective function, which seeks to minimize the distance between true posterior and inference posterior, by considering the variational lower bound as its objective, since we also seek for maximization of ELBO with respect to ϕ and θ [5]:

$$\ln p_\theta(\mathbf{x}) \geqslant \mathcal{L}_{\phi,\theta}(\mathbf{x}) = \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \tag{12}$$

$$= \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x},\mathbf{z})\right] - \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[q_\phi(\mathbf{z}|\mathbf{x})\right] \tag{13}$$

$$= \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x}|\mathbf{z})\right] - \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})}\right] \tag{14}$$

$$= \underbrace{\mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\text{expected negative reconstruction loss}} - \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))}_{\text{approximate posterior regularizer}} \tag{15}$$

Since we are dealing with standard normal distribution as given fixed prior (Gaussian-VAE), we can reconsider the objective ELBO function as follows [6]:

$$\mathcal{L}_{\phi,\theta}(\mathbf{x}) \overset{\text{constant shifted}}{\approx} -\frac{1}{2\cdot \text{tr}(\boldsymbol{\sigma}_\theta^2 \mathbb{I})}\|\mathbf{x} - \boldsymbol{\mu}_{\theta,\ z\sim q_\phi(\mathbf{z}|\mathbf{x})}(\mathbf{z})\|_2^2 - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|\mathcal{N}(\mathbf{0},\mathbb{I}))$$

So that, due to standard normality of the prior, and normality of the inference posterior $q_\phi(\mathbf{z}|\mathbf{x})$ [6]:

$$\mathcal{L}_{\phi,\theta}(\mathbf{x}) \overset{\text{constant shifted}}{\approx} -\frac{1}{2\cdot \operatorname{tr}(\boldsymbol{\sigma}_\theta^2 \mathbb{I})}\|\mathbf{x}-\boldsymbol{\mu}_\theta(\boldsymbol{\mu}_\phi(\mathbf{x}))\|_2^2 - \frac{1}{2}\left(\operatorname{tr}(\boldsymbol{\sigma}_\phi^2(\mathbf{x})\mathbb{I}) + \boldsymbol{\mu}_\phi(\mathbf{x})^\intercal \boldsymbol{\mu}_\phi(\mathbf{x}) - \ln\det(\boldsymbol{\sigma}_\phi^2(\mathbf{x})\mathbb{I})\right)$$

which can now be easily computed and differentiated, as all of its parameters are either given or can be computationally derived by the hidden layers of the VAE-nn. Usually, since we are dealing with likelihood maximization problem, we can re-state the optimization problem, provided with the above objective, as a negative log-likelihood minimization problem, that is we now seek to minimize $-\mathcal{L}_{\phi,\theta}(\mathbf{x})$, and hence define it as our new objective function: $\text{LOSS}_{\phi,\theta}(\mathbf{x})$, which we seek to minimize.

Moreover now, we have that our latent space has a tractable posterior feature distribution, as being forced to move towards the prior's distribution of $\mathcal{N}(\mathbf{0},\mathbb{I})$. That makes it easy for VAE to serve as a generative model, since we can just feed some $\tilde{\mathbf{z}} = (\tilde{\mathbf{z}}_i)^d \sim \mathcal{N}(\mathbf{0},\mathbb{I})$ sample to the decoder, so that having already learnt optimal (in terms of the objective function) $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ parameters, we will have some $\tilde{\mathbf{x}} = \boldsymbol{\mu}_\theta(\tilde{\mathbf{z}})$, which would approximately follow the desired marginal distribution $p_\theta(\mathbf{x})$ [5]. That means VAEs, unlike simple AEs, can serve as valid generative models, due to the tractable nature of their latent space posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and prior $\mathcal{N}(\mathbf{0},\mathbb{I})$ distributions.

5.3 Extensions of VAEs. (β-VAE, CVAE, VAE-GAN, WAE(VAE-WGAN))

There are some interesting extensions of VAEs, which utilize different statistical principles in order to enforce the model to behave in certain ways or produce more clear output. Let us introduce some of theirs, which we are going to consider further or discuss.

5.3.1 β-VAEs

For example, β-VAE, which is basically such an extension of the original VAE, that β-VAE objective function has its approximate posterior regularizer term (as in (15)) being adjusted with $\beta \in \mathbb{R}$ coefficient, so that once $\beta \to 0$, the KL-divergence term fades away, and the model comes to original simple AE with an intractable posterior distribution of the latent space, and whence $\beta \to +\infty$, the posterior distribution basically starts to asymptotically follow standard normal one, and hence posterior feature distribution becomes highly interpretable, but useless, as far as reconstruction loss will have no meaningful impact on the objective function's value [13]. Yet, according to Doersch [9], adjusting the D_{KL} term of (15) does not really change the model's performance in most of the cases, being only relevant, once we need to simulate $\mathbf{z} \sim \mathcal{N}(0, \beta\cdot\mathbb{I})$ for some intent of a reseaecher.

In case of ours, the VAE model was not introduced with the regularization parameter, as usually adjusting it to be not equal to 1 has significantly worsened the model's performance, since increase in β usually corresponded to blurrier output, whilst decrease of β lead to unavailability to generate a good sample from the normal distribution with the encoder-produced parameters, which would follow the distribution of the latent space.

5.3.2 Conditional VAEs

Another type of VAEs, which is quite useful for us is a conditional VAE, which introduces the conditional structure of the ELBO objective function, assuming that now, we have input and desired output being divided into $\mathbf{x}$ and $\mathbf{y}$, where $\mathbf{x}$ can be treated as a condition for getting some desired corresponding $\mathbf{y}$, so that, according to Doersch [9], the unknown population distribution of $p_\theta(\mathbf{y}|\mathbf{x})$ can be estimated as follows:

$$\text{with } \mathbf{z} \sim \mathcal{N}(0, \mathbb{I}) : \; p_\theta(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\tilde{\boldsymbol{\mu}}_\theta(\mathbf{z}, \mathbf{x}), \tilde{\boldsymbol{\sigma}}_\theta^2 \mathbb{I}).$$

$$\ln p_\theta(\mathbf{y}|\mathbf{x}) \geqslant \mathcal{L}_{\phi,\theta}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})} \left[\ln p_\theta(\mathbf{y}|\mathbf{x}, \mathbf{z})\right] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{x}))$$

Let us note that $p_\theta(\mathbf{z}|\mathbf{x}) = N(0, \mathbb{I})$, due to independent sampling of $\mathbf{x}$ and $\mathbf{z}$ during model validation

Utilizing the above structure of the model, we can make it learn some feature-dependent behavior of the input data to be reconstructed, $\mathbf{y}$, so that in order to generate new data, given a feature space sample: $\tilde{\mathbf{x}}$, we can simply feed $\tilde{\mathbf{x}}$ along with some Gaussian standard sample $\tilde{\mathbf{z}} \sim \mathcal{N}(0, \mathbb{I})$ to the model's decoder and get an estimate $\tilde{\mathbf{y}}$ of the corresponding $\mathbf{y}(\mathbf{x})$, i.e.: $\tilde{\mathbf{y}} = \boldsymbol{\mu}_\theta(\tilde{\mathbf{z}}, \tilde{\mathbf{x}})$. Below, in Fig. 11 one can see the architecture of a general CVAE, which has been depicted by Doersch as a node-chart:

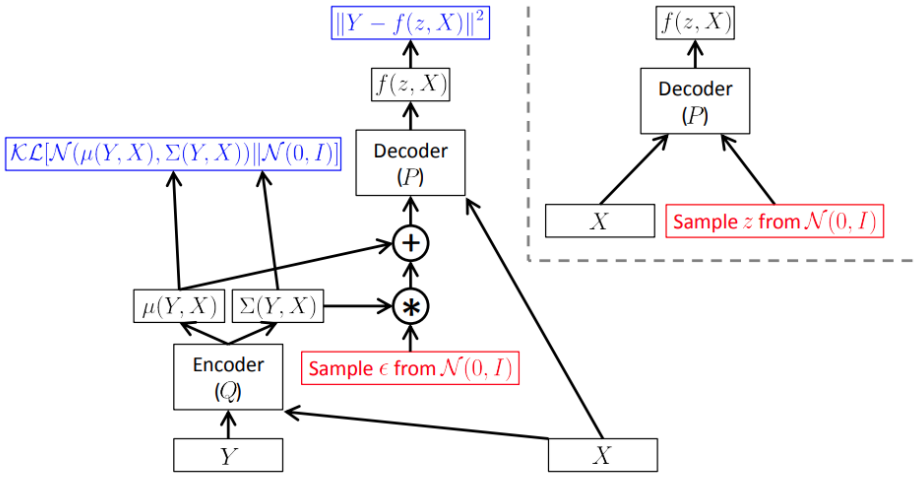


Figure 11: Train stage of CVAE feed-forward NN shown on the left, and validation stage shown on the right, here $f(z, X) = \boldsymbol{\mu}_\theta(\mathbf{z}, \mathbf{x})$

5.3.3 CVAE-GANs and CWAEs and

Another extensions are VAE-GANs and WAEs (VAE-WGANs), along with their conditional extensions, which take the original VAE with all its modelling procedure being preserved, having discriminator of a VAE serving as a generator for a generative adversarial network, Fig. 12. According to Larsen et al. [11], compared to an ordinary VAE, the VAE-GAN model yields significantly better attributes visually that leads to smaller recognition error. Such a "double" technique for image generation and discrimination allows for a way more accurate approximation of the true (population) distribution of $\mathbf{y}|\mathbf{x}$: $p_\theta(\mathbf{y}|\mathbf{x})$ estimation and faster generator learning, due to the specifics of VAEs, yet, such models are usually far more time-complex, as far as GANs' penalty invokes the use of Jensen-Shannon divergence in its loss function. This complexity, however, can be countered by the use of Wasserstein distance instead in WAEs, which enjoy the same very structure as simple VAE-GANs, but employ the Wasserstein distance between generated and real data, as a replacement for original GANs' objective function, thus following the general architectural structure of the original Maevskiy's et al. WGAN-based generative model, introduced in [1].

Below, one can see the picture from [11], which schematically depicts the VAE-GAN's architecture:

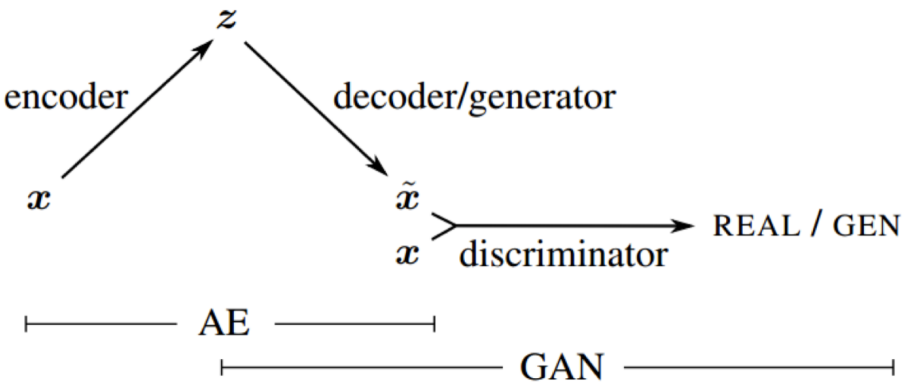


Figure 12: VAE's decoder acting as a generator in VAE-GAN (advanced generator learning)

5.4 Modelling and calculation experiment

5.4.1 Unconditional generative model

At first, a simple unconditional VAE has been designed, so that to be able to look at correctness of our implementation of its, in terms of reconstructing the original images, provided with the initially generated latent space as input for the model's decoder.

Creation of such a model has been performed using "Google Colab" environment for python development, due to its accessibility and flexibility in terms of external library usage and availability

to dynamically address occurring issues without a necessity to debug the code line-by-line.

For a simple VAE network development a tensorflow v.2.8.2 library has been used, since it provides pre-built functions for deep learning models' construction, what easens the task of building custom complex NN models from scratch.

The obtained simple VAE model enjoyed the following layer architecture:

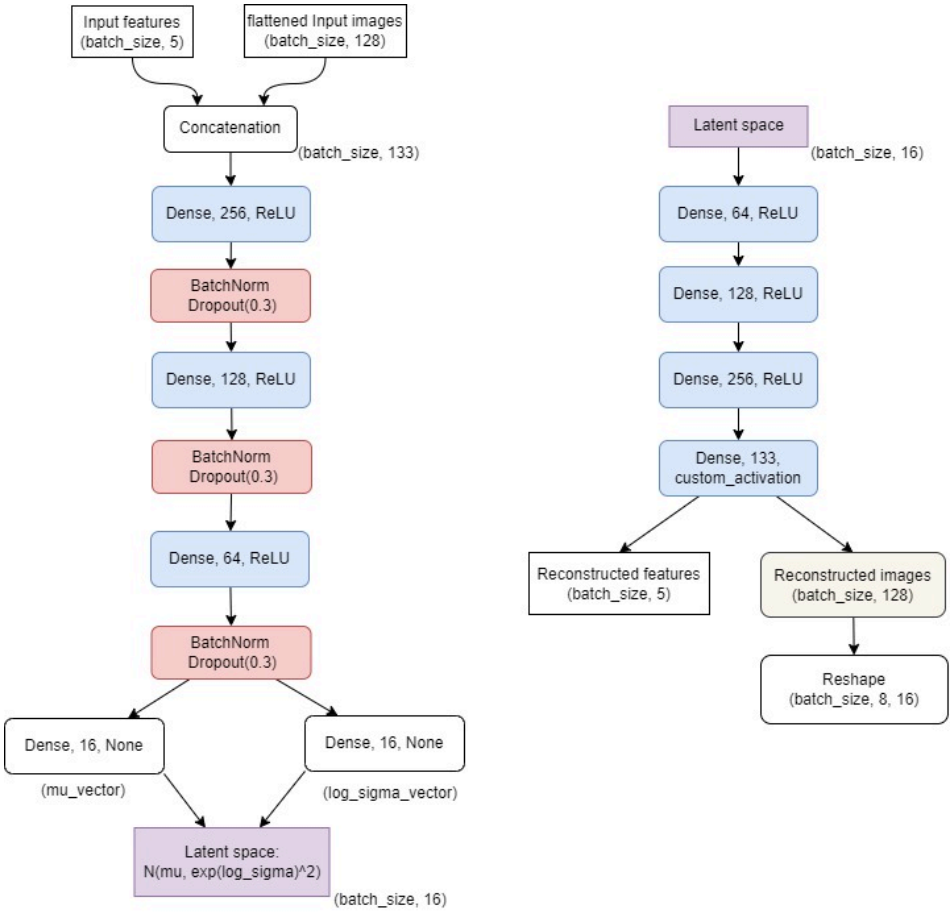


Figure 13: unconditional VAE architecture for the encoder (on the left) and the decoder (on the right) with exemplary parameters used.

Basically such an architecture was developed after assessing the $\boldsymbol{\mu}_\theta(\mathbf{z})$-generated samples in terms of the χ^2 statistic without accounting for the influence of standard deviation (so that to see how model performs on average). In Fig. 20 in appendix one can see the matrix of feature-based metrices produced by the simple VAE of the above structure. In Fig. 21, one can also examine the reconstructed images, produced from the validation set generated latent space. The corresponding χ^2 statistic value for such $\boldsymbol{\mu}_\theta(\mathbf{z})$-generation, evaluated on the test dataset after 4000 epochs, was approximately equal to 726.629, which was worse than the χ^2 metric for $\mathcal{N}(0, \mathbb{I})$-generated results of the baseline conditional WGAN model. Still, the model did improve its unconditionally-obtained χ^2 metric to approximately 440.817 after decreasing the `batch_size` from 256 to 128.

The notable improvement, however, to the original model of WGAN was that the constructed VAE model has much lower training execution time than the baseline WGAN model, being evaluated at the same number of epochs. For example, in the Google Colab environment, the original WGAN model was to run up to 5 hours to train 4000 epochs, whilst for the developed VAE model it took only around 40 minutes to be executed. According to Kingma [6], such behavior of the VAE model can be explained by comparison of its objective function, and objective function of the GAN considered. In VAE we train encoder and decoder sequentially, by having encoder being trained to avoid deviating from the desired distribution of the latent space, whilst decoder learns to reconstruct the initial image based on the minimization of the Euclidean distance term (reconstruction loss). In GANs however, generator and discriminator are trained in another way, which implies slower generator learning, due to additional efforts needed to be put for minimizing the objective of the GAN because of its rather complex structure.

One can also see the unconditionally-generated "fake" pad responses, generated by fitting the decoder of the above model with a sample from standard normal distribution in Fig. 22. As it can be seen, almost all the images look visually compatible to the original images from test dataset, for example. Still, features, which are generated along with the pad responses are not defined to be physically-meaningful, moreover it is impossible to fit the generator part of the network (decoder), so that to produce some physically-correct pad responses based on real features within the framework of the architechture above. Hence, it was decided to follow the original paper's model structure, that is make it conditional, so that to be able to get the image plots based on real or fake feature samples.

5.4.2 Conditional generative model

Now, after examining how the unconditional generative model of VAE works, let us introduce the VAE of conditional architecture, as shown in Fig. 14. Now, the model of the 14 architecture has

behaved quite unexpectedly, since with conditional feature tuning, the generation of the test signal reconstructions, based on the true test features and samples from the standard normal distribution, has faced much worse χ^2 values for the very same parameters of the network. For example, the network of the above architecture with all ReLU activations, `batch_size` of 256 and latent dimension of 16, produced the χ^2 statistic of around 33757.624 on 4000 epochs (nevertheless, running way faster than the original model). Now, the problem was to understand how to deal with that of an issue, and why it has occured. In fact, according to Kingma [5], VAEs, unlike GANs are much easier to train, but are producing less clear images (more blurry ones) due to the architecture employing dimensionality reduction in order to traverse from the input layers to the latent space. This is why VAEs are usually used for denoising the data or doing smart-PCA, but in our case the image size is very small (only 8x16 pixels), and hence with a conditional structure of the above VAE, with decoder being fitted with standard gaussian samples, it is more likely for our network to provide

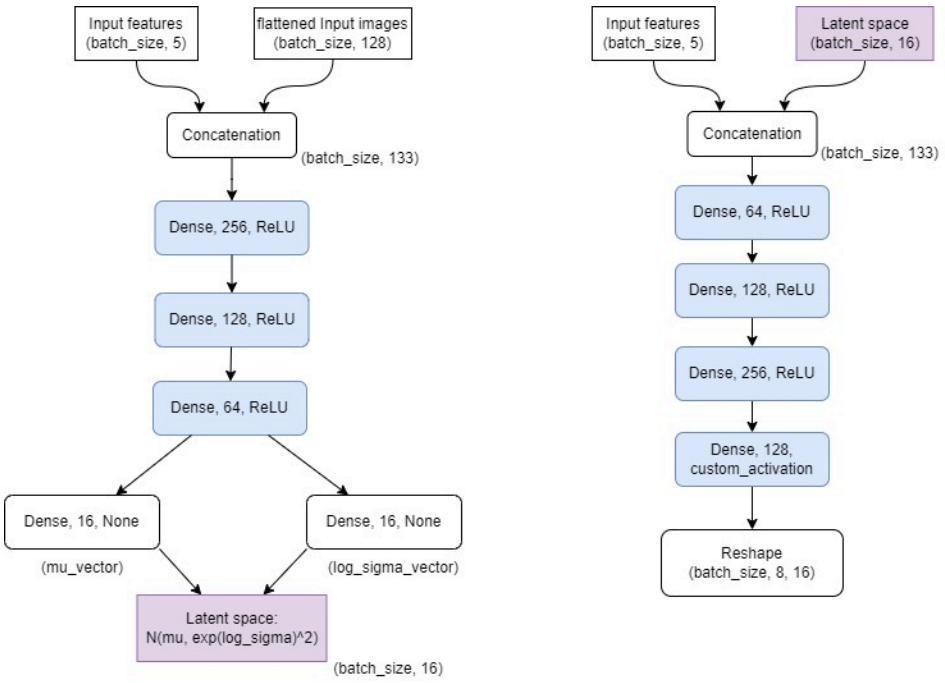


Figure 14: conditional VAE architecture for the encoder (on the left) and the conditional decoder (on the right) with exemplary parameters used.

blurry or inadequate images. A possible solution to this problem may be enlargening the deepness of the neural network, so that to have our first dense layer being of really greater dimension, than that of the images we have, since then the NN would be able to learn not to drop any components of the train images to "bluriness" (something like good-for-overfitting of the train data).

In our case, several remedial measures were tried to increase the test-generated χ^2 statistic:

1. Increasing the latent dimension with 256 `batch_size` tested first. The corresponding χ^2 statistics looked as follows: $(\text{latent\_dim}, \chi^2) = ((16, 32, 64), (33757.624,\ 32162.674,\ 28975.331))^\mathsf{T}$

2. Lowering the `batch_size` with 64 latent_dim: $(\texttt{batch\_size}, \chi^2) =$
$= ((128, 64, 32), (21710.641,\ 17812.893,\ 14490.174))^\mathsf{T}$

But none of the above remedial measures helped really that much for the output of the conditional VAE developed to be compatible to the baseline model. For example, one can see the output of the CVAE on the test dataset in Fig. 23. As it can be seen, the evaluated metrics hardly follow the real ones. Hence, there is still a huge space for improvement of the originally developed CVAE, as its architecture is still quite simple (for example, we did not use convolutions in encoder layers). However, the advantage of execution time performance of CVAE (which follows that one of VAE in terms of fastness) has to be considered as a probable reason to use the developed CVAE of the above structure (or improved) as a fast-learning generator for the baseline WGAN model.

6 Normalising Flows

In addition to these widely used generative neural approaches, another machine learning algorithm has been used in our work is called Normalising Flows(NFs), a family of generative models, first described in NICE [14]. By using the fundamental idea of normalising flows, we can create a new type of neural network called invertible neural network. So what are the connections and differences between Normalising flows and Generative Adversarial Network?

Let's start with the definition of a good generative model. A good generative model should produce the distribution of the sample parameters through known samples that we feed to this model. For example, if we feed a batch of flowers pictures to a model, it will return us new pictures of flowers related with inputs which is generated based on the distribution of given training samples. It is actually hard to run backward propagation in many deep learning methods. Thus, the latent probability distribution (i.e. posterior $p(z|x)$) should be simple enough to make the calculation easily and efficiently. That is the main reason why a Gaussian normal distribution is chosen to be used in NFs generative models.

6.1 Background

The normalising flows is technically a probabilistic generative models based on invertable transformations. It's sampling and density evaluation can be efficient and exact. However, neither of GANs and VAEs can have exact evaluation of the probability density of unobserved data. Besides, training a GAN model can be a vexing problem because of a number of factors including mode collapse and unstability of training. Unlike GANs and VAEs, normalising flows estimates the distribution of parameters directly, because of that, flow-based generative models have some natural advantages. Firstly, training a flow is a very easy task which can be usually solved by using stochastic gradient descent. Secondly, it is possible to obtain an useful latent space and exact inference of latent variables. On the other hand, the main disadvantage of flow-based generative models is that they have relatively higher dimensions compared to Variational Autoencoders which can obtain a sparser latent space. Also, a hideous amount of resources is required for saving the computed gradient in reversible neural network.

Maximum likelihood estimation

$$\bigtriangledown_\theta \, D_{KL}(p_{data} \parallel p_\theta) = \bigtriangledown_\theta E_{x\sim p_{data}(x)}[log_{p_\theta}(x)] = E_{x\sim p_{data}(x)}[\bigtriangledown_\theta log_{p_\theta}(x)] \tag{16}$$

A generative model can be learned to minimise the KL divergence between the data distribution and the model distribution. Since the p_{data} does not depend on parameter θ, the most left expression in $eq.(1)$ is identical to the maximum likelihood estimation objective.

Change of variable theorem Flows can use the Change of Variable Theorem to perform maximum likelihood estimation. Given an observed data variable x, the change of variable formula defines a model distribution on X by: (described in NICE [Dinh et al., 2014])

$$p_X(x) = p_Z(f(x))|\det(\frac{\partial f}{\partial x^T})| \tag{17}$$

where p_Z is a simple prior probability distribution on a latent variable $z \in Z$, and $f : X \to Z$ is a bijection (with an invertible transformation $g = f^{-1}$).

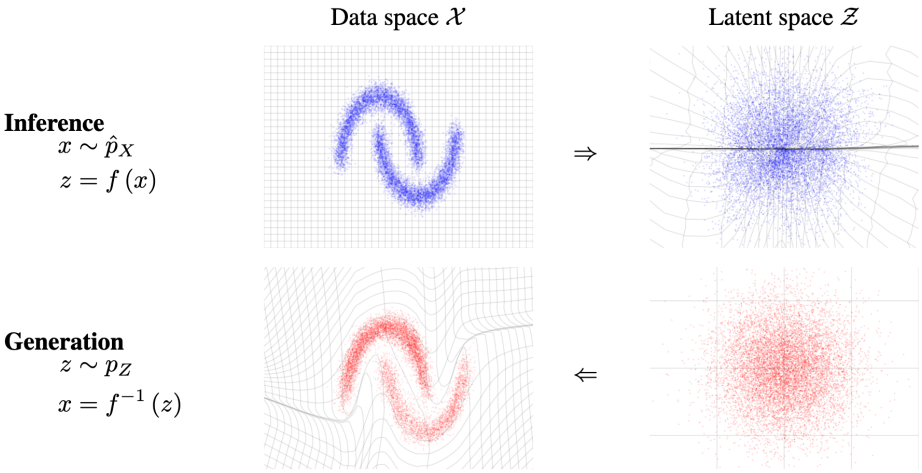


Figure 15: (Reference: DENSITY ESTIMATION USING REAL NVP [15])

From the figure above [15], one can see the fundamental principle of normalising flows, that is to perform a sequence of invertible transformations $f = f_1 \circ f_2 \circ ... \circ f_k$, starting from the prior distribution to a target distribution. A latent distribution of the data is obtained and it can be used for generating new samples of the target distribution through the inverse bijections $f^{-1} = f_k^{-1} \circ f_{k-1}^{-1} \circ ... \circ f_1^{-1}$, known as the generative direction of flows. This simple experiment of Moons dataset is also done by

6.2 Types of NFs model

In recent years, many researchers began to get studies looking at different variations of normalising flows. Among these three iconic types of NFs models, MAFs and Real-NVPs are chosen for the study of fast simulations in this research project.

1. NICE [14]
2. Masked Autoregressive flows [17]
3. Real-NVP [15])

6.2.1 NICE

As described in paper Nonlinear Independent Components Estimation[14], NICE model is a flow-based model which consists of multiple additive coupling layers, and each layer:

$$\begin{aligned} h_1 &= x_1 \\ h_2 &= x_2 + m(x_1) \end{aligned} \tag{18}$$

where x_1 and x_2 are two split parts of x with D dimensions as well as the inverse:

$$\begin{aligned} x_1 &= h_1 \\ x_2 &= h_2 - m(h_1) \end{aligned} \tag{19}$$

where m is an arbitrarily complex function. The equation (18) shows why it is invertible for any m. In general, a more complex layered transformation can be obtained by combining several coupling layers in a network form. Since each coupling layer changes only partition of inputs, the composition of two or more coupling layers should modify every element of inputs by switching the unchanged partitions of input data. In (Dinh et al., 2014), author suggested that at least three coupling layers are necessary to allow all dimensions to influence one another.

6.2.2 Masked Autoregressive flows

use single Gaussian distribution to parameterise the conditionals of of an autoregressive model, s.t. $p(x_i|x_{1:i-1}) = N(x_i|\mu, (\exp \alpha_i)^2)$

Consider $z \sim \pi(z)$ and $x \sim p(x)$ are given and the probability density function $\pi(z)$ is known, Masked Autoregressive Flow aims to learn the pdf of x. MAF generates i-th observation x_i through the conditional ($p(x_i|x_{1:i-1})$ which is an affine transformation of latent space) For generating new data from latent distribution: we have

$$x \sim p(x_i|X_{1:i-1}) = z_i \odot \sigma_i(X_{1:i-1} + \mu_i(X_{1:i-1}), \text{ where z} \sim \pi(z)$$

For density estimation, we have

$$p(x) = \prod_{i=1}^{D} p(x_i|x_{1:i-1})$$

As one can notice, sampling from the previous $x_{1:i-1}$ is required at each iteration, therefore, it is more computationally heavy compared with Real-NVP, in total $\mathcal{O}(D)$ iterations.

6.2.3 Real-NVP

Real-NVP stands for Real Non-Volume preserving Flows which do not strictly limit the Jacobian determinant being zero. With a slight modification of the Additive coupling layer in [14], we obtain:

$$\begin{aligned} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}) \end{aligned} \tag{20}$$

$$\begin{aligned} x_{1:d} &= y_{1:d} \\ x_{d+1:D} &= y_{d+1:D} - t(y_{1:d}) \odot \exp(-s(y_{1:d})) \end{aligned} \tag{21}$$

and the Jacobian matrix of this transformation is:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & \bar{0} \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}} & \operatorname{diag}(\exp(s(x_{1:d}))) \end{bmatrix} \tag{22}$$

Where the $\operatorname{diag}(\exp(s(x_{1:d})))$ is a diagonal matrix expanded by $\exp(s(x_{1:d})$. The log-determinant of this Jacobian matrix is $\prod_{i=1}^{d} \ln \exp(s(x_{1:d})) = \sum_{i=1}^{d} s(x_{1:d})$. It is noteworthy that there is no any calculation required for the determinant of s and t, therefore, both of them can be arbitrarily computationally complex. Interestingly, Real-NVP flows have really efficient sampling and probability density estimation since the scale and shift operations are computed only once in either forward pass or backward pass. In order to see this fact, we can first compute the mean and standard deviation of all elements in parallel since all data inputs are known. y can then be computed in a forward direction. Next, one should note that the inverse transformation is the form of equation below,

$$\begin{aligned} z_{1:k} &= y_{1:k} \\ z_{k+1:d} &= (y_{k+1:d} - \mu(y_{1:k}))/\sigma(y_{1:k}) \end{aligned} \tag{23}$$

all divisions are element-wise operations. The mean and variance are generally obtained through neural networks, so they are not guaranteed to be invertible. The way how Real-NVP works is that it introduce a constraint that each data point x_{d+1} from the transformed distribution depends only on the first $[0 : d]$ elements from the base distribution. However because of this Equation, the sufficient condition of making the whole transformation invertible is not to make sure the invertibility of a neural network.

For the masked convolution, we used the checkerboard masking method which assigns zero to the pixel where it's coordinates is even. Suppose that we have a mask b and rectified convolutional nets s and t, the function form of y is given by,

$$y = b \odot x + (1 - b) \odot (x(s(b \odot x)) + t(b \odot x)) \tag{24}$$

Actually, R-NVP model is an improved version of the Non-Linear Independent Component Estimation (NICE model) which just has no scaling term in bijectors. Since we need scaling to deal with complex distributions of given data.

6.2.4 Conditional flows

According to the initial task of this project, the fast simulation should be done by a conditional generative model as the five numerical features are given to generate new signal images. Therefore, our Real-NVP model should do the task of simulation with data sampled from standard normal distribution and five features described in section 3.3. The basic structure is defined as the following flow chart: In Masking scheme, black blocks are one's and white blocks are zero's, these blocks will

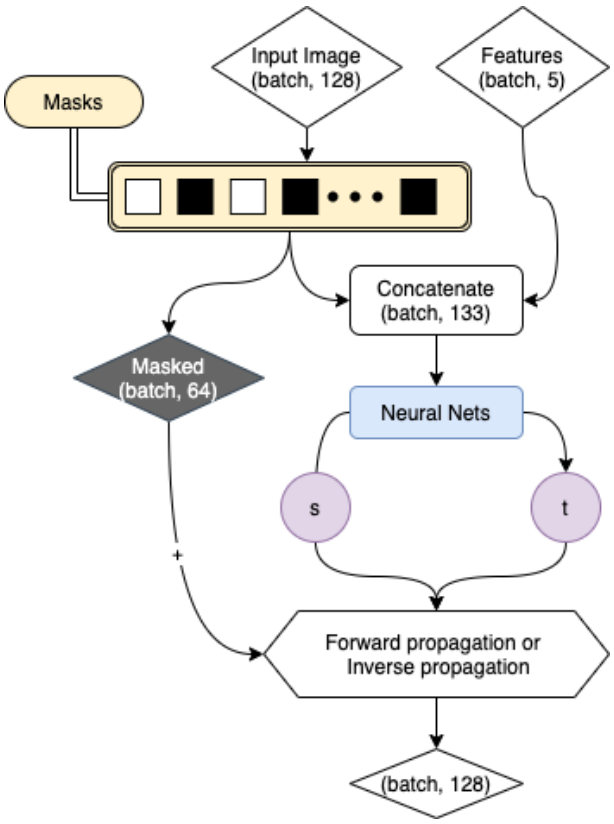
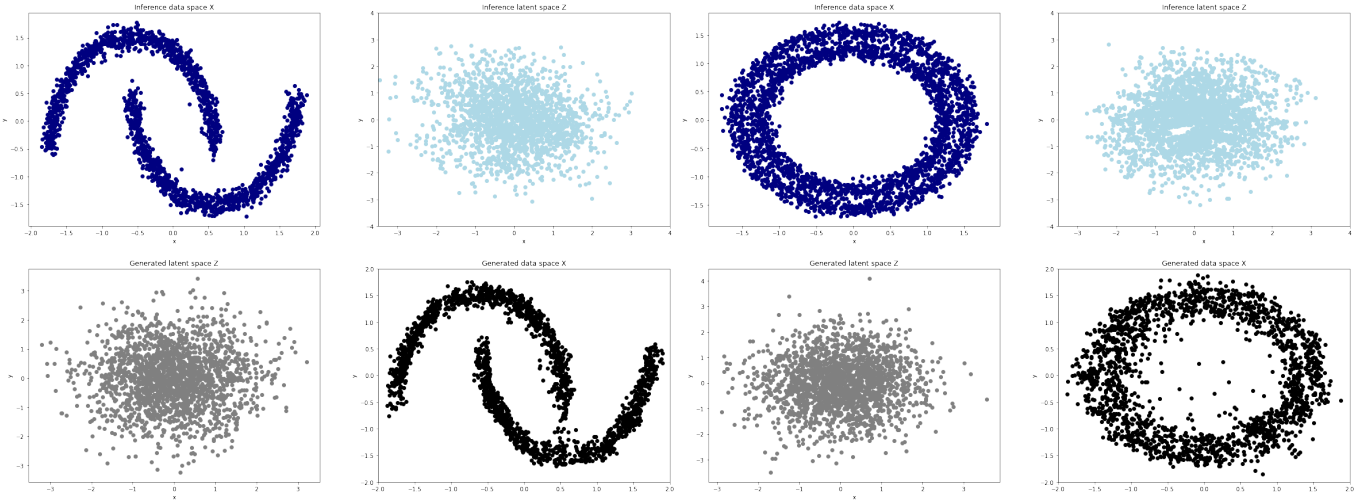


Figure 16: Design of conditional Real-NVP

switch positions in the beginning of each iteration of coupling layers. Then the unmasked data will be concatenated with features resulting in a tensor with dimension (batch, 133). After that, the scale and shift factors are produced by two neural networks. It is also noteworthy that the neural nets are not required to be invertible since their are not a part of Jacobian of bijections. Hence, there is another benefit that the neural nets for S and T can be very complex. The last step is to do the forward or inverse propagation which is defined in eq20 and eq21.

6.3 Experiments on simple datasets

Train on toy datasets Since we have the prior knowlegde of MAFs [17] that they have higher computational costs in sampling, we decided to stick on the implementation and hyper-parameter tuning of the Real-NVP model. We begin our experiments by the realisation of the Real-NVP approach proposed in (Dinh et al., 2016) and test the model on toys dataset Moons and Circles.



(a) Moons (b) Circles

Figure 17

This Real-NVP flow learns an invertible, stable, mapping between a data distribution $p\hat{_X}$ of a toy 2-d dataset and a latent distribution p_Z (typically a Gaussian normal distribution). There is a set of bijections maps samples x from the data distribution in the upper left into approximate samples z from the latent distribution, in the upper right. This corresponds to exact inference of the latent state given the data. Then we can use the generated latent distribution to obtain a new exact generation of the input sample distribution in the lower right through the inverse calculation of bijections.

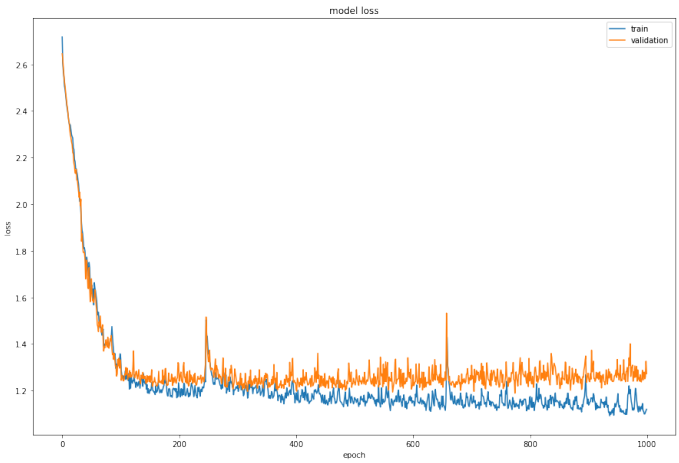


Figure 18: The training loss and validation loss on 1000 epochs.

6.4 Results with TPC data

Before moving to the main result of Normalising flows performance for our fast simulation task, let us first visualise the generated signals: Overall, it is obvious that not all generated signals have

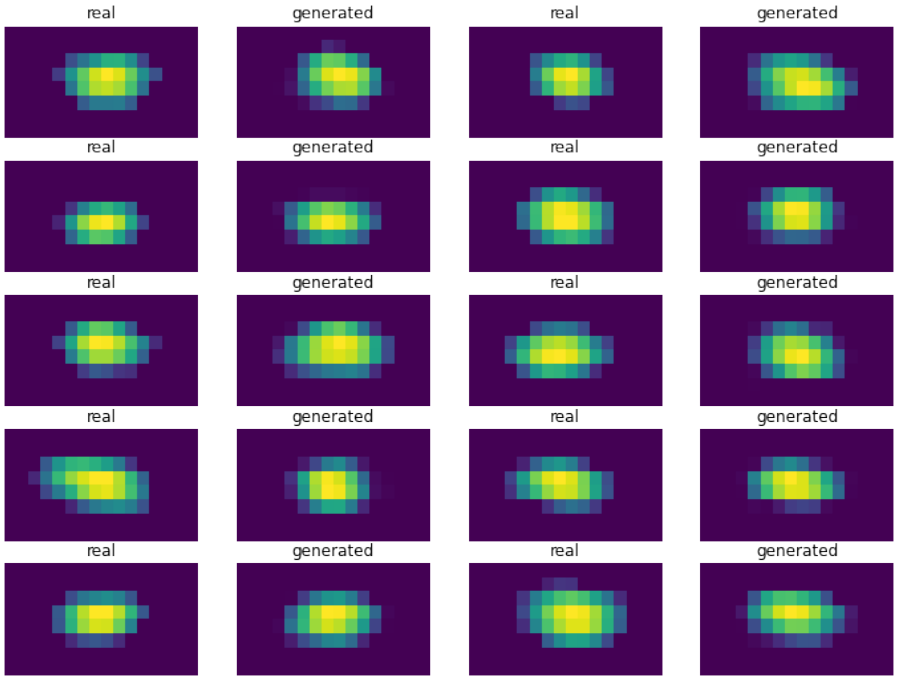


Figure 19: Comparison of Real and generated data [by model TEST 03]

the same numerical features that real data have. We have performed numerous trails to find out the most efficient and accurate combination of hidden layers in the Neural Networks for $S$and T calculation. In 7, we can the number of neurons in each layer. In addition, TEST 3 model has an additional layer with activation function *tanh* for S, it may help to provide a more robust fit.

The architecture of different coupling layers design.			
Layers	TEST 1	TEST 2	TEST 3
T	$[32, 64, 64, 64, 128]$	$[64, 64, 128, 128]$	$[133, 128, 128, 133]$
S	$[32, 64, 64, 64, 128]$	$[64, 64, 128, 128]$	$[133, 128, 128, 128, 133]$

Table 7: RealNVP flows - Changing hidden layer configuration Test results

Real-NVP: According to num_coupling_layer and batch_size						
Layers	batch	epoch	Chi-Square TEST 1	Chi-Square TEST 2	Chi-Square TEST 3	RunTime
2	128	10000	24748.93	22632.42	20332.82	1h 52m
	256	10000	24527.42	26738.54	25082.35	1h 34m
4	128	10000	25710.36	24329.36	24533.36	2h 01m
	256	10000	23533.36	21438.26	21134.52	1h 53m
6	128	10000	21145.95	21678.27	20981.11	2h 16m
	256	10000	18365.01	18475.39	18101.6	2h 08m
12	128	10000	7814.00	7932.23	7769.00	3h 39m
	256	10000	7123.22	7348.13	6969.004	3h 20m

Table 8: Real-NVP: According to num_coupling_layer and batch_size

To summarise the corresponding results, we have to admit that the accuracy of Real-NVP is not comparable with the generative adversarial network. However the generated signals are not that blurred and they do have the alike shape of the real data. The best performance we got is the TEST 3 model with the smallest Chi-squared is 6969.004.

7 Conclusions

The main objective of the research project was to explore the possibilities of improving the original GAN based model and to implement and test alternative generative models to check the possibility of achieving similar results to the original model.

With GAN based approach is was realised that there is possibility achieve similar results to the baseline model with a simpler GAN architecture like vanilla GAN. Also, hyper parameter tuning may help to optimize the baseline model to perform slightly better. Biggest drawbacks compared to VAEs and Normalising flows method is the lengthier training times.

Concerning what can be said about the performance of VAEs in terms of the stated task, we can proclaim that unfortunately, CVAEs of rahter simple structure fail to even beat the $\chi^2 = 10000$ threshold in terms of generated test distribution comparisons. Still, VAEs are very powerful models which are very fast to learn, and hence are not to be omitted in future research in the field of improving the baseline model. According to [11] and [12], we can easily introduce a Variational autoencoder as a fast-learning generative model for some Generative adversarial network, even the Wasserstain-one. However, since so-far-results in terms of χ^2 metric on validation set of the created CVAE are not very compatible with the baseline model's, it may be better to first try using Real-NVPs (NFs) as a supposed generator for the baseline model, since its test performance in terms of χ^2 is better. In general, VAEs are very powerful and can possibly produce results, which comply to the ones produced by the baseline model, however, much more validation tests have to be done in order to achieve the desired results.

For Real-NVP flow based generative model, it is quite hard to push the target distribution to the standard normal distribution and that brings us a very serious problem of generating a new data with the latent space which is not close to standard normal distribution. As a consequence, when we are passing the test features to the flows, it will transform the latent density distribution inversely based on the features of training data. However, the most positive aspects of Real-NVP flow are the faster computation time and ease of training.

In conclusion, the main realisation of this project is that there is significant success in generating high quality images and minimising runtime via advanced machine learning techniques. There is further research that can be done in this sphere.

8 Bibliography

References

[1] A. Maevskiy, F. Ratnikov, A. Zinchenko, and V. Riabov, *"Simulating the Time Projection Chamber responses at the MPD detector using Generative Adversarial Networks"*, In Eur. Phys. J. C 81, 599, (2021) [Online]. Available: https://doi.org/10.1140/epjc/s10052-021-09366-4

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *"Generative Adversarial Networks"*, (2014) [Online]. Avaliable: arXiv:1406.2661

[3] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, *"Improved training of Wasserstein Gans"*, (2017) [Online]. Available: arXiv:1704.00028

[4] M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos, *"The Cramer distance as a solution to biased Wasserstein gradients"*, (2017) [Online]. Available: arXiv:1705.10743

[5] D. P. Kingma and M. Welling, *"Auto-encoding variational Bayes"*, (2014) [Online]. Available: arXiv:1312.6114

[6] D. P. Kingma and M. Welling, *"An introduction to variational autoencoders"*, (2019) [Online]. Available: arXiv:1906.02691

[7] P. Baldi, *"Autoencoders, unsupervised learning, and deep architectures"*. In JMLR: Workshop and Conference Proceedings 27:37–50, (2012) [Online]. Available: http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf

[8] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *"Learning internal representations by error propagation"*. In Parallel Distributed Processing. Vol 1: Foundations. MIT Press, Cambridge, MA, (1986)

[9] C. Doersch, *"Tutorial on Variational Autoencoders"*, (2021) [Online]. Available: arXiv:1606.05908

[10] D. Bank, N. Koenigstein, and R. Giryes, *"Autoencoders"*, (2021) [Online]. Available: arXiv:2003.05991

[11] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, *"Autoencoding beyond pixels using a learned similarity metric"*, (2016) [Online]. Available: arXiv:1512.09300

[12] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf, *"Wasserstein Auto-Encoders"*, (2019) [Online]. Available: arXiv:1711.01558

[13] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, *"Beta-VAE: Learning basic visual concepts with a constrained variational framework"*. In Proceedings of the International Conference on Learning Representations, (2016)

[14] L. Dinh, D. Krueger, and Y. Bengio, *"NICE: Non-linear Independent Components Estimation"*, (2015) [Online]. Available: arXiv:1410.8516

[15] L. Dinh, J. Sohl-Dickstein, and S. Bengio, *"Density Estimation Using Real NVP"*, (2017) [Online]. Available: arXiv:1605.08803

[16] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, *"Normalizing flows for probabilistic modeling and inference"*, (2021) [Online]. Available: arXiv:1912.02762

[17] G. Papamakarios, T. Pavlakou, and I. Murray, *"Masked autoregressive flow for density estimation"*, (2018) [Online]. Available: arXiv:1705.07057

[18] H. Reyes-Gonzalez and R. Torre, *"Testing the boundaries: Normalizing flows for higher dimensional data sets"*, (2022) [Online]. Available: arXiv:2202.09188

[19] PhD Nathan C. Frey, *"FastFlows: Flow-based models for Molecular Graph Generation"* In Medium, (2022) [Online]. Available: `https://towardsdatascience.com/fastflows-flow-based-models-for-molecular-graph-generation-a8327bb9bee1`. [Accessed: 10.06.2022]

[20] H. Wen, P. Pinson, J. Ma, J. Gu, and Z. Jin, *"Continuous and distribution-free probabilistic wind power forecasting: A conditional normalizing flow approach"*, (2022) [Online]. Available: arXiv:2206.02433

[21] P. Wielopolski, M. Koperski, and M. Zięba, *"Flow Plugin Network for Conditional Generation"*, (2021) [Online]. Available: arXiv:2110.04081

9 Appendix

9.1 Additional Images and plots

9.1.1 VAE modelling

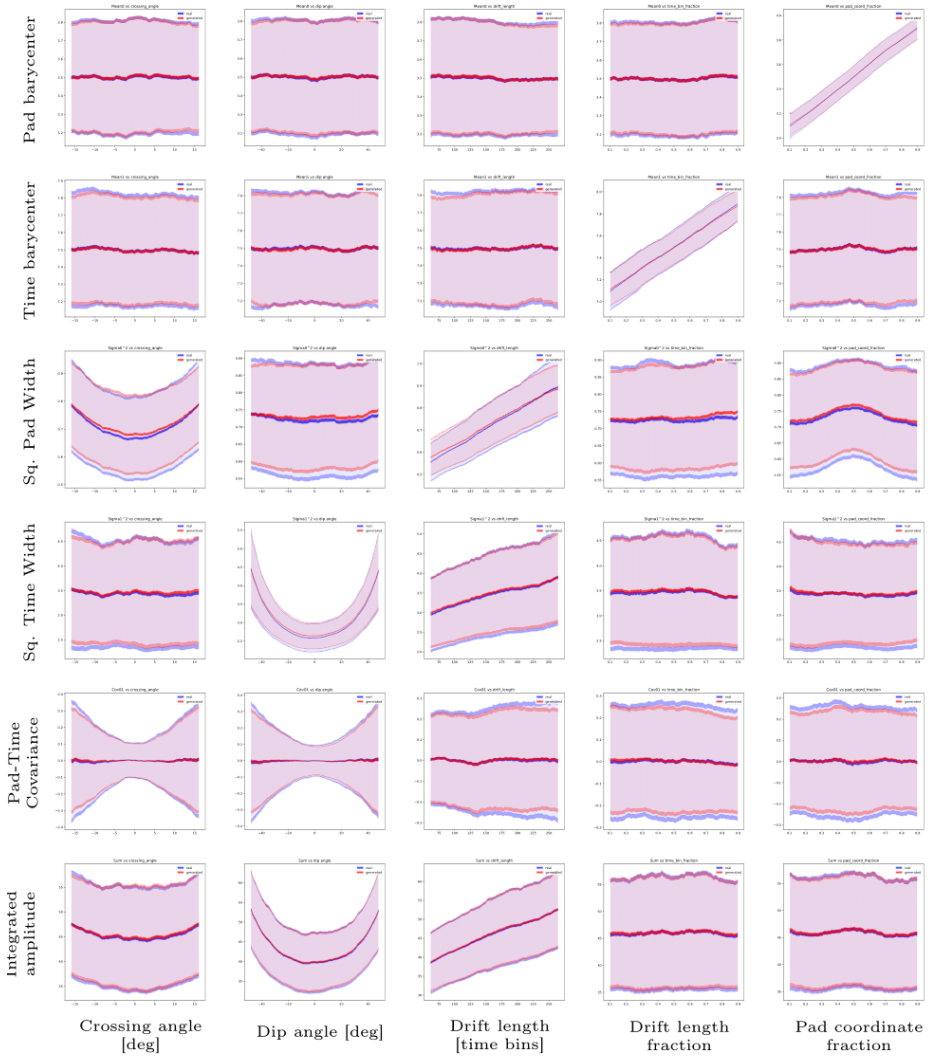


Figure 20: A ($\beta = 1$)-VAE test reconstruction evaluation metrics after training for 4000 epochs, with [256, 128, 64] hidden layers architecture, all ReLU activations, having the latent dimension of 16

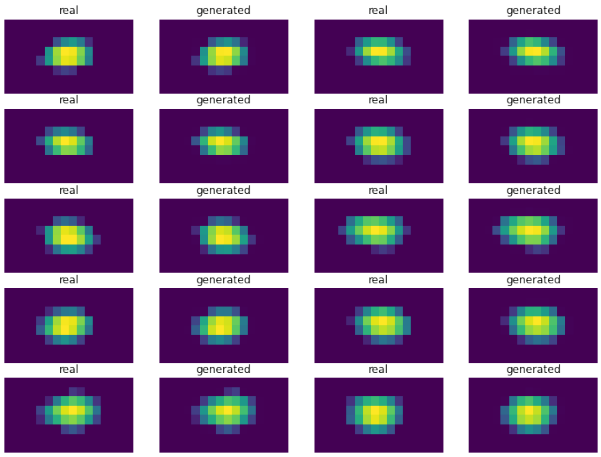


Figure 21: Comparison of true test signals and unconditional-VAE-reconstructed ones

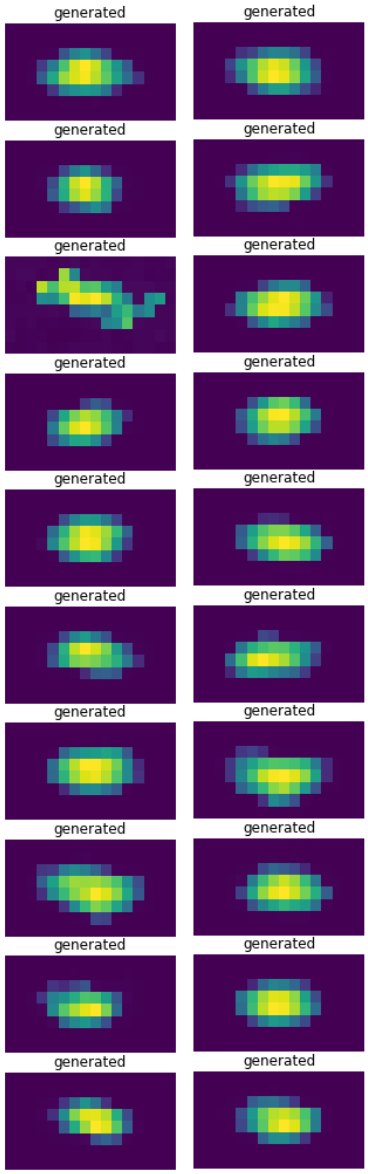


Figure 22: A load of signals generated by the decoder of the Fig. 13 architechture, provided with a sample $\mathbf{z}$ from $\mathcal{N}(\mathbf{0}, \mathbb{I})$ distribution

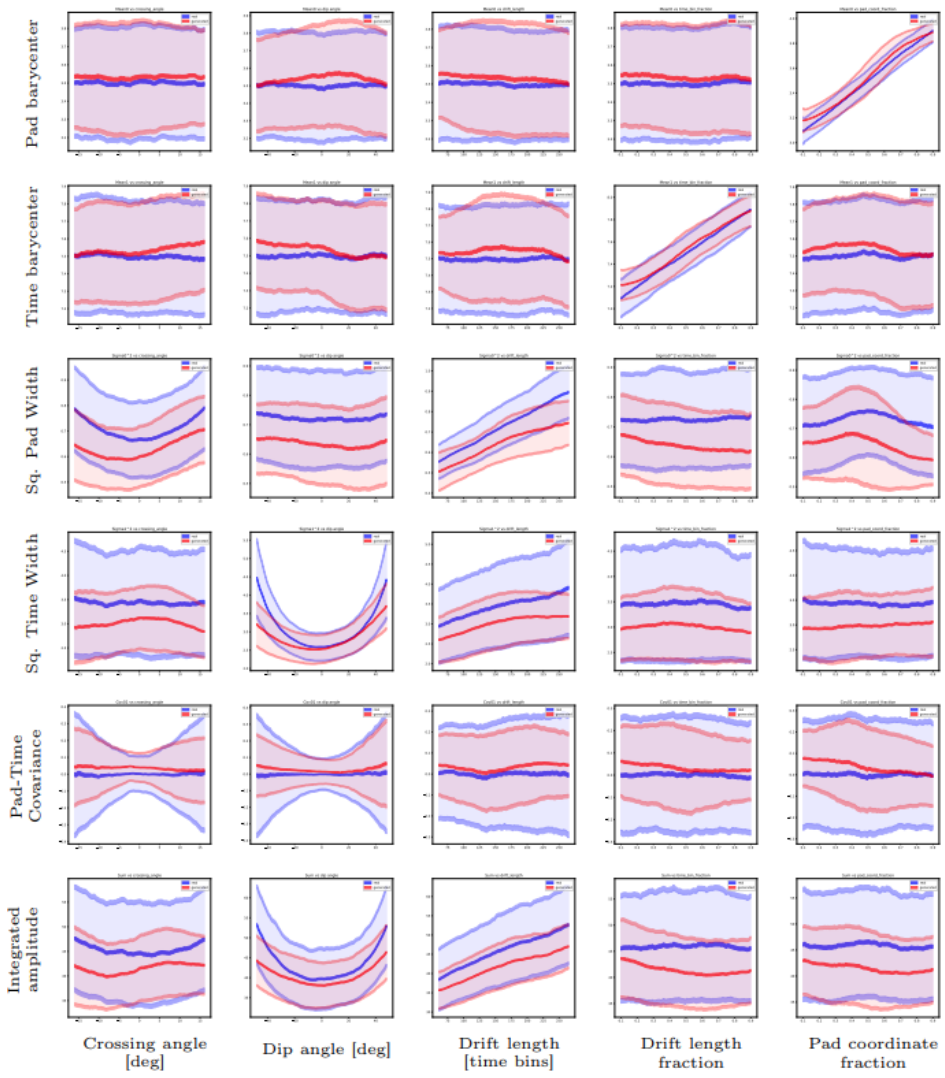


Figure 23: An example of bad reconstruction of the test dataset image parameters, according to the developed CVAE

9.2 Real-NVPs

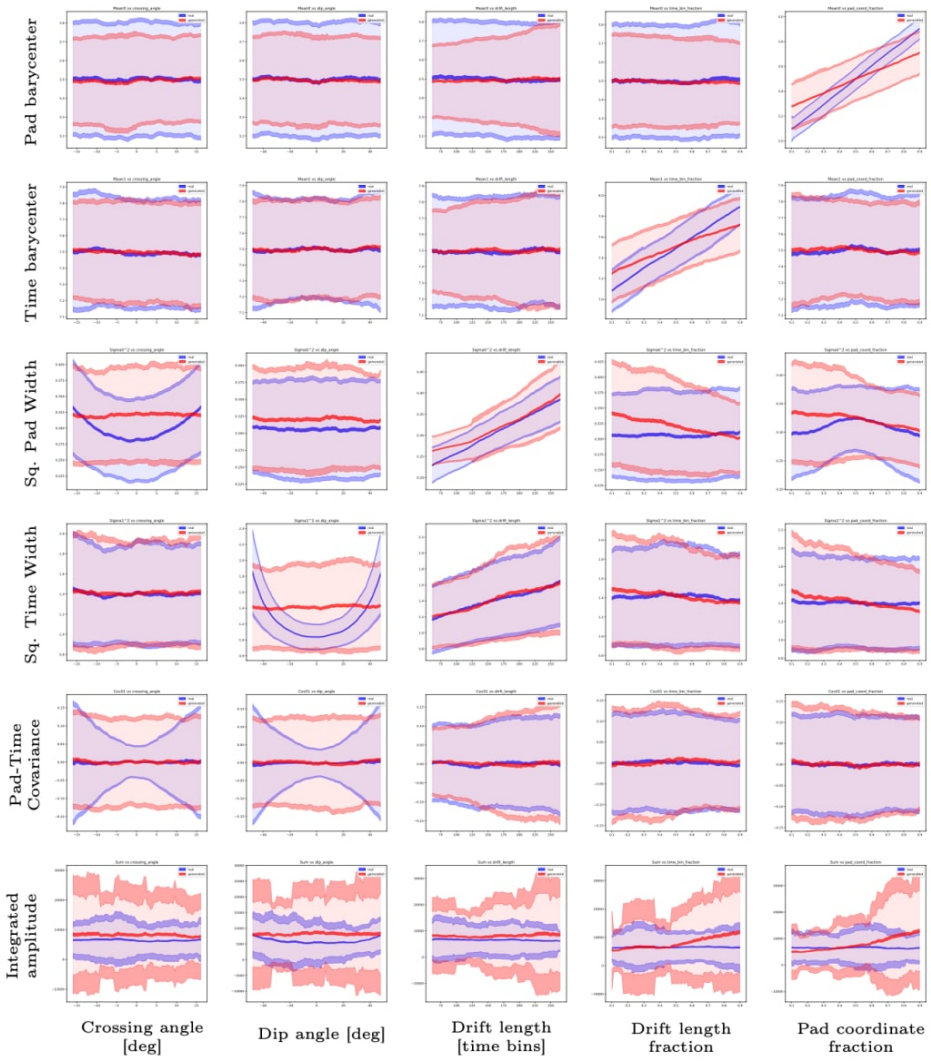


Figure 24: Real-NVP generated test metrics