

Report

Water Cherenkov detector reconstruction by
Generative Adversarial Neural networks

Grigolia Giorgi

Supervisor: Dr. Cristovao Vilela

CERN
2021.

Contents

1	Introduction	3
2	Existing methods	3
3	Neural networks	4
4	Event reconstruction with generative CNN	5
5	Efforts to move on deep convolutional GANs	6
6	Wasserstein GAN	9
7	Conclusion	11

1 Introduction

In a variety of physics experiment it can be beneficial to have targets with very large mass. One of the most cost effective ways to do so is using water. One important characteristic of water is that as light moves through it approximately 3/4 of the speed of light in vacuum. When a charged particle travels through the water with higher speed than light, Cherenkov radiation is emitted. Charged particles also emit spherical waves due to particle polarization around the charge. In the case when $v_p > c/n$ these waveforms overlap and interfere according to The Huygens principle. Resulting emission happens with an angle θ with respect to the particle direction.

$$\theta = \frac{c}{nv_p}$$

It is possible to detect this radiation cone and predict particle direction and energy based on amount of emitted light. Capturing of the signal happens using PMTs or photomultiplier tubes. For every charged particle there is some threshold total energy, for example electron theoretically needs to have 0.8MeV energy to radiate.

Our study is based on architecture of Super-Kamiokande detector. It has volume of 50 kilotons and is located inside the Kamioka mine in Gifu Prefecture, Japan. Super-K has two separated regions inner and outer detector. Inner detector with the volume of 32 kilotons is surrounded by 11000 inward-facing 20-inch PMTs, enough for 40% coverage. One of the main purposes of Super-Kamiokande and other water Cherenkov experiments is to detect and study neutrinos through observations of solar, atmospheric and artificially generated neutrinos. In 1998 a very important discovery of neutrino oscillations was made in Super-K, while observing atmospheric neutrinos created by muon decays, which implied they should be massive particles. Detection happens by two main ways, first when neutrino goes W exchange converting into equivalent charged leptons and second by elastic scattering with electron. Interaction might happen via exchange of Z bosons, in which case they do not convert into charged leptons. This interactions can be detected through transfer of energy on the target.

2 Existing methods

Existing algorithm for event reconstruction is called fitQun, it was developed specially for Super-K experiment and has feature to reconstruct multi ring events. It employs maximum likelihood estimation. An event topology hypothesis together with its associated kinematic parameters θ , which include the vertex position, particle creation times, the azimuthal and zenith angles of the particle directions, as well as their momenta, are considered in the likelihood function during a fit.

$$L(\Gamma, \theta) = \prod_j^{unhit} P_j(unhit|\Gamma, \theta) \prod_i^{hit} (1 - P_i(unhit|\Gamma, \theta) \times f_q(q_i|\Gamma, \theta) \times f_t(t_i|\Gamma, \theta))$$

Two indices run over all PMT which register and do not register hit separately. Likelihood density for measuring charge q_i is $f_q(q_i|\Gamma, \theta)$ and for making hit at observed time t_i is $f_t(t_i|\Gamma, \theta)$. Likelihood can be written also in terms of expected number of photoelectron on each PMT.

3 Neural networks

In a mathematics function is defined as mapping which assigns values from one set to another. We can also imagine a type of function which takes image as an input and determines what kind of object is shown on the image. However, it is practically impossible to solve such problem analytically and create above mentioned function. The aim of machine learning is to find as accurate approximation as possible, but rather than having the human attempt to write a program for this, a goal is to set computer find the set of rules. In other words goal is to write a program which itself generates a function. The set of rules can be represented as neural network. A perceptron is practically a function defined by $n + 1$ numbers or weights, taking vector $x_1, x_2, \dots, x_n$ as an input and gives some output

$$f(x) = g(w_0 + \sum_{i=1}^n x_i w_i)$$

here w_0 is called bias, and g is called activation function. without activation function neural network can describe linear relation but in machine learning we also want to describe nonlinear behaviour. There are many types to chose from, one common example is logistic sigmoid function.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

It can map infinity to $[0, 1]$ interval, which is natural to use in classification problem, when output represents probability of input object to be in one class. It is possible to have multiple outputs by using multiple perceptrons. In that case we can consider weight vectors instead of just one number and all the weights can be represented by matrix. And function will get a form

$$f(\mathbf{x}) = g(\mathbf{w}\mathbf{x})$$

It is also possible to add several layers of perceptrons, which are called hidden layers. Each hidden layer feeds into the next until you reach output one. The training process includes progressively changing weights to improve network as accurate approximation of desired function. In this process training data is fed to the network, which includes inputs and subsequent correct outputs. On every step the actual output is compared to known correct output using cost function. The cost function quantifies how accurate was the network while predicting the output. Many different types of cost functions exist, overall cost is determined by averaging all the output node difference to the real value. Take initial weights w_0 and cost function J the algorithm aims to find local minimum

of cost function by numerically calculating gradient and taking step to negative direction at given point. if we take a look at how output of the neuron is calculated

$$f(a) = g\left(\sum_{i=0}^n w_i a_i\right)$$

It is evident that component except g are determined on values in previous layers, thus process is repeated for every layer recursively. This process is called backpropagation. Ideally after some iterations all parameters will reach such value that cost function will be as close to minimum as possible.

4 Event reconstruction with generative CNN

The traditional use of neural networks in regards to the analysis experimental data would be end-to-end event classification. Neural network would be able to take measurements of PMT's as an input image and predict corresponding parameters. Starting from handwritten character recognition first proposed by Fukushima in 1979 [2], there exist a lot of examples of using neural networks for similar tasks. However, we are exploring different approach.

Ground work which was done to explore neural network which predicts hit charges and times for each PMT's for a given set of track parameters. The work is based on 2017 paper about generating different objects using convolutional networks [3].

The network takes input parameters corresponding to the particle identification, that is either an electron or a muon neutrino indicated by binary numbers, 1 if it is given particle and 0 if it is not. The next are x , y and z positions of vertex where collision takes place, followed by three components of momentum and energy. As shown on the figure 1 first fully connected layer connects each triplet to 512 nodes which is followed by FC2, then all nodes are combined and followed by FC3 and FC4. As a result we get vector of length 1024. convolutional layers come next, on this stage image has the size 21×11 , with 64 layers. After 2 more convolutional layers, final output is 168×88 image with 3 layers. They represent mean charge, variance and hit probability for each PMT.

It is important to understand that implicitly we assume that distribution of charge measured by each PMT is Gaussian and therefor we are looking for parameters of this Gaussian distribution, together with the probability that photon will reach given PMT.

The data used in the training is not real experimental data, but it is generated using WCSim water Cherenkov detector simulation package created by Water Cherenkov Machine Learning Group [4]. The simulated detector is the Hyper-Kamiokande experiment intermediate water Cherenkov detector, cylinder with 10m height and 6m diameter. In our work only such PMT measurements were taken into account which are located on the walls of the cylinder, hence we only got rectangular images. This fact simplifies network architecture but still gives the chance to study key properties of network, which can be upgraded in future.

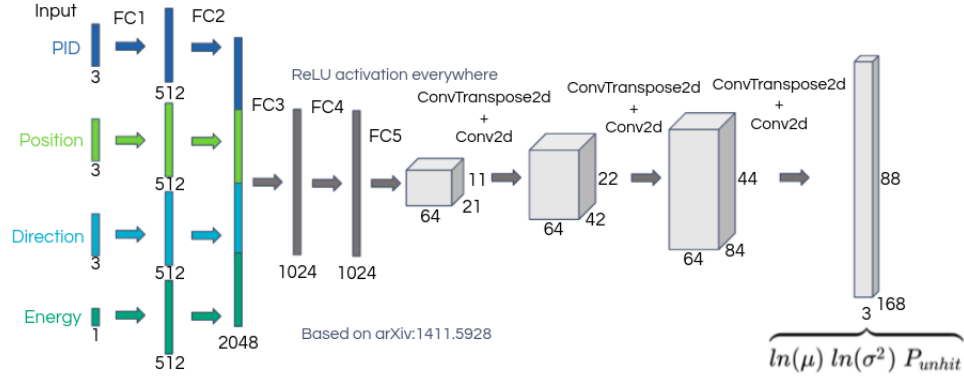


Figure 1: Generative neural network

You can see the training data example on the figure 2

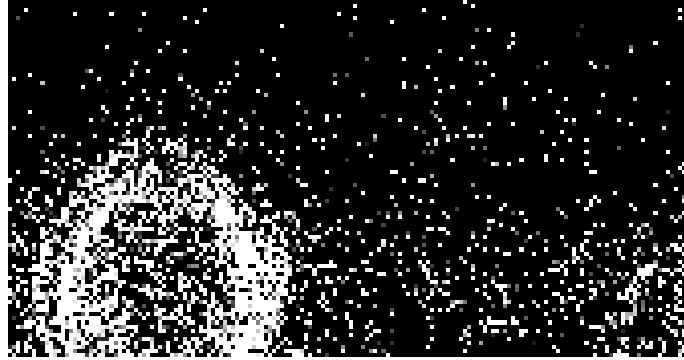


Figure 2: Example of a training data

Two data sets corresponding to electrons and muon were used in the training, where I recreated results of the previous work. You can see reconstruction of mean charges measured over all PMT's on the figure 3. Initial parameter for these events are selected by us for representational purposes.

5 Efforts to move on deep convolutional GANs

Generative adversarial networks, shortly named as GANs, are clever way to train a generative network by changing loss function with another network called a discriminator, which classifies generated images as fake or real. This two models are trained together and compete in zero-sum game until equilibrium is reached

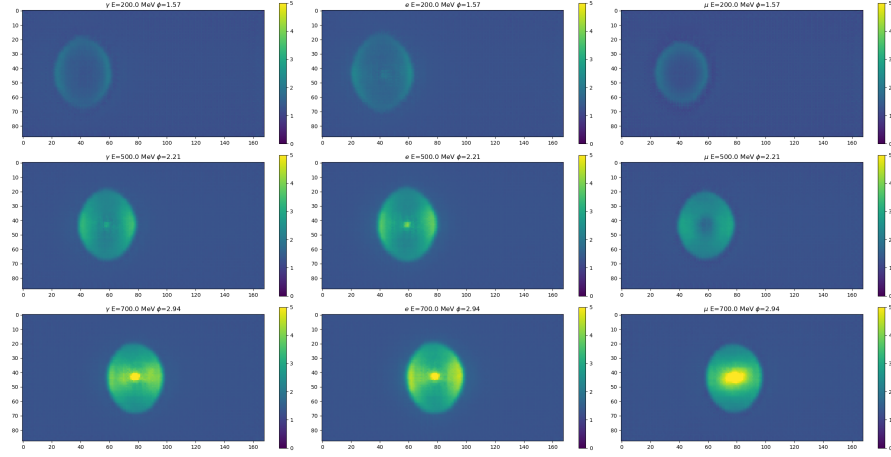


Figure 3: Mean charge measured by the detector

when discriminator is mistaken around half of the time.

Initial step to move on GANs was creating random event from the distribution parameters which was generated by existing CNN. the process was as follows first we generate normally distributed random numbers and then modulated them mean value and variance for each PMT. After that we generated numbers with uniform distribution between 0 and 1 and took into account only thous PMTs which heat probability was higher than given number, and finally we ignored PMTs with negative charge as they are non-physical. You can see the result on figure 4

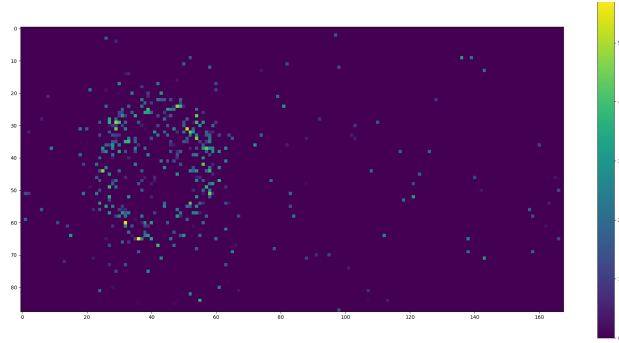


Figure 4: Image of the event generated based on given distributions for each PMT

After adding this functionality to our generator, we proceed to create discriminator network which is essential for any GAN. The aim of discriminator is

to receive data, in our case 88×168 image and give probability of this image to be real, basically a number between 0 and 1. In terms of tensor this is an $1 \times 88 \times 168$ tensor. We used DCGAN template given by pytorch creators [5] and changed its architecture according to our input. Network uses 5 convolutional layer, followed by 1 final fully connected layer. while we reduce dimensions of of each layer padding is used. Padding is a method which is used to prevent loosing pixels on the perimeter of the image. Solution is straightforward, pixels with zero value are added on the boundary. Stride, which defines the step size of a kernel is set to 2 most of the time. The exact architecture of the discriminator can be seen on the figure 5. Immediately after we ensured that code was running

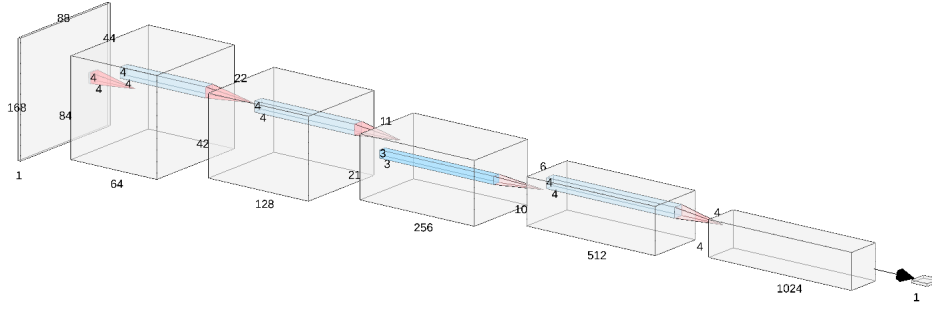


Figure 5: Image of the event generated based on given distributions for each PMT

without any technical difficulties we have, we have decided to take changes into training process. Progressing from usual DCGAN (deep convolutional adversarial neural network) to conditional GAN (cGAN). We should mention that GANs are trained completely unsupervised which mean they are build to generate realistic images without any particular labels. Contrary to that in our application proper reconstruction aims to generate physically accurate image to initially given parameters. In conditional GAN labels are also taken into account during the training process, but this process is not so straightforward. In our case we have tried embedding label as an channel to the input. Input vector is connected to 88×168 image by fully connected layer and discriminator is trained with image which has two channels.

Another important factor which we want to take into account is that physical system behaviour is not totally deterministic, that means it includes certain randomness, which is caused by ever existing background fluctuations. To mimic above mentioned behaviour we introduce random input together with input parameters. At the beginning we chose to add 5 random parameter to existing 9. Important ground work also exists for cGANs. We based our training procedure on article called generative adversarial text to image synthesis [6]. As you can see from schematics on figure 6 we train discriminator by giving real image with appropriate label and fake image generated by label and random noise together. Finally we were able to train the network and get some results. Unfortunately,

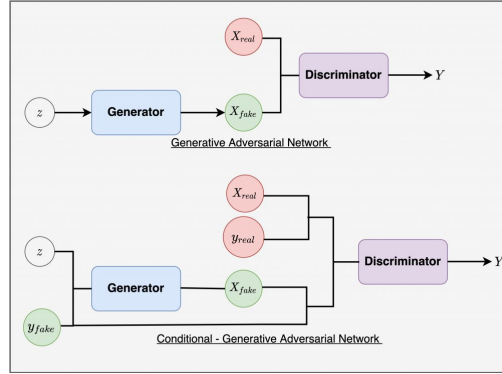


Figure 6: Schematic representation of cGAN and usual GAN

what we have seen from the loss dependent on iterations (graph is given on a figure) our GAN reveals behaviour which is commonly called mode collapse [8]. The system reaches a local minimum where generator manages to always find most plausible output for next iteration of discriminator. The result is that generator goes through a small set of similar outputs and over-optimises for discriminator.



Figure 7: Loss function dependence on iteration

6 Wasserstein GAN

We have decided to use different route and use existing GAN architecture, making as small changes as possible to adapt it to our purposes. This could not

be a long term solution of course but it would help us to understand important aspects of how GANs work in general. We chose Wasserstein GAN [9] for that purpose, as it is considered to be stable in many cases. Loss function for WGAN is quite simple, with the discriminator often called a critic in this case

$$D(x) - D(G(z))$$

where $D(x)$ is discriminator output for real data and $G(z)$ is generator output for fake data. Generator loss is $D(G(z))$. It is important to mention that we usually take average values over batches that is given to the network.

training procedure is similar to previous cases but it is important to mention that discriminator is trained 100 times before any generator training and we train it 5 times for each generator training moving forward.

We took existing GitHub project [10] which uses LSUN data set to generated images of bedrooms by one of the main authors of the original paper Martin Arjovsky. The main thing that was changed was data loader, our training data was changed to become 64×64 image taken from the center of the original. You can see example on figure 8. As it can be seen some traces of rings are present, which will gave us chance to study how network will behave. Training revealed

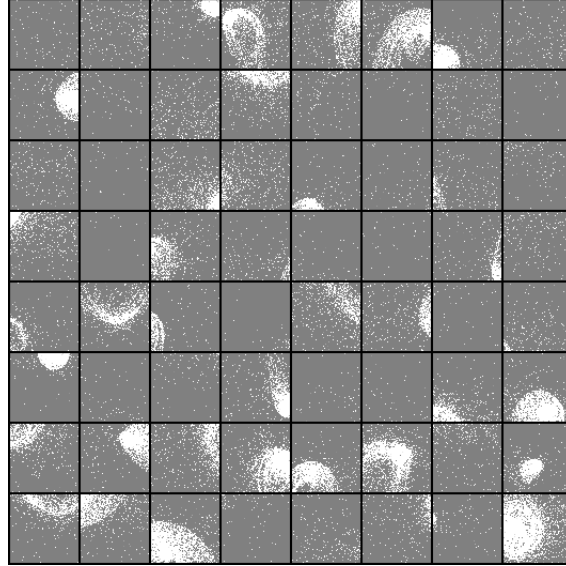


Figure 8: Batch of samples used in WGAN training

very interesting behaviour, mode collapses can be observed on some cases, more importantly network sometimes generates completely blank images for some amount of iteration and then changes to usual picture. While the network was not able to reproduce clear Cherenkov rings with limited amount of training, it still learned to reproduce local features with roughly the same scale as rings in the training sample. We only have managed to train for 4 epoch so far. Results

can be seen below.

Loss function quickly becomes constant with dramatic spikes, this behaviour

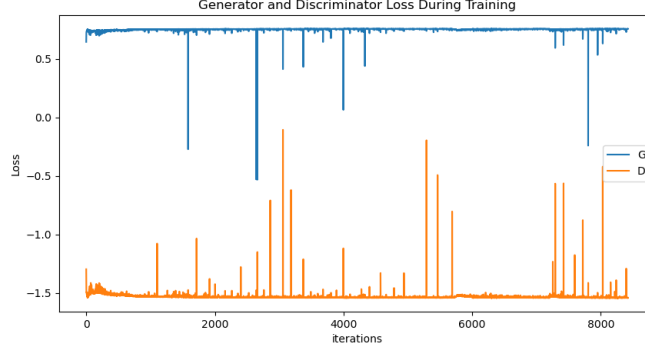


Figure 9: Loss function dependence on generator iterations for WGAN

requires explanation. Finally you can see what is the output batch is after 21500 iterations below on the figure 10.

7 Conclusion

I have made the first attempt to use a GAN on a water-Cherenkov events. Adapted initial generative network to be trained as a conditional GAN, even though results were interesting GAN was diverging and therefore we have decided to replace the data set in the existing original WGAN network with our own dataset. The initial results from that study are very promising. Future plans include to continue training for large number of epochs, replicate results of the original paper for reference, to modify network to take entire image of the event. And finally implement conditionality which will ensure precise results of the reconstruction

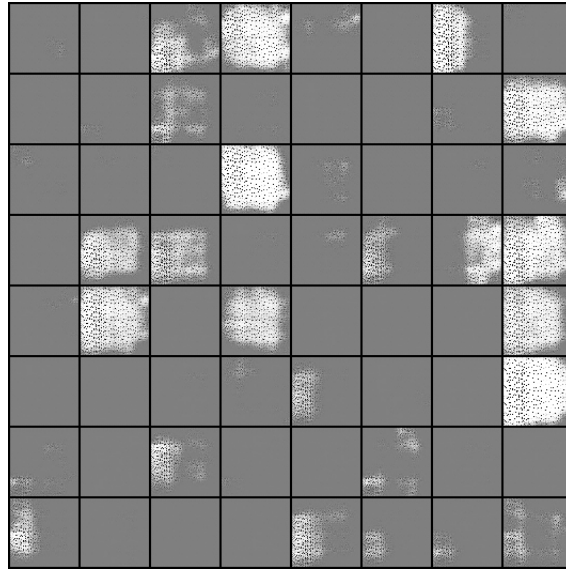


Figure 10: Batch of output images of WGAN after 21500 iterations

References

- [1] Water Cherenkov detectors
<https://www.sheffield.ac.uk/physics/research/particle/neutrino/water-cherenkov>
- [2] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics* 36, 193–202 (1980).
- [3] Alexey Dosovitskiy, Jost Tobias Springenberg, Maxim Tatarchenko and Thomas Brox. Learning to Generate Chairs, Tables and Cars with Convolutional Networks, 2014; arXiv:1411.5928.
- [4] Water Cherenkov Machine Learning Group website
<https://www.watchmal.org>
- [5] Water-Cherenkov detectors
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
- [6] Scott Reed, Zeynep Akata, Xincheng Yan, Lajanugen Logeswaran, Bernt Schiele and Honglak Lee. Generative Adversarial Text to Image Synthesis, 2016; arXiv:1605.05396.
- [7] Conditional GAN (cGAN) in PyTorch and TensorFlow
<https://learnopencv.com/conditional-gan-cgan-in-pytorch-and-tensorflow/>

- [8] Generative adversarial networks: Common problems <https://developers.google.com/machine-learning/gan/problems>
- [9] Martin Arjovsky, Soumith Chintala and Léon Bottou. Wasserstein GAN, 2017; arXiv:1701.07875.
- [10] WGAN code by Martin Arjovsky <https://github.com/martinarjovsky/WassersteinGAN>