

# 1. Master's Thesis

## 1.1 Summary

The objective of this Master's Thesis is to study whether a neural network can be able to identify symmetries in data sets while learning to perform a task on that set, adapting the procedure of the article *Symmetry Meets AI* to the continuous case. The procedure followed to achieve a model capable of evaluating the understanding of symmetries by a neural network is based on generating thousands of two-dimensional potentials  $V(x, y)$ , two-variable functions, with the same symmetries as those used in the article *Symmetry Meets AI*, and training a neural network for each of them. The objective of these networks is, given  $x \in y$ , to predict the value of the potential. Projecting the results from the last hidden layer of these networks using PCA, we obtain two-dimensional images, with which we train a neural network dedicated to classifying them according to the symmetry of the potential from which these images come. The smallest and most robust of the resulting networks is capable of identifying the original symmetry in 96% of the cases, thus showing that the networks that predict the value of the potential encode information about its symmetry.


## 1.2 Steps

### 1.2.0 Choice of Hyperparameters of the FCNN that predicts $V(x,y)$

The following hyperparameters were studied:

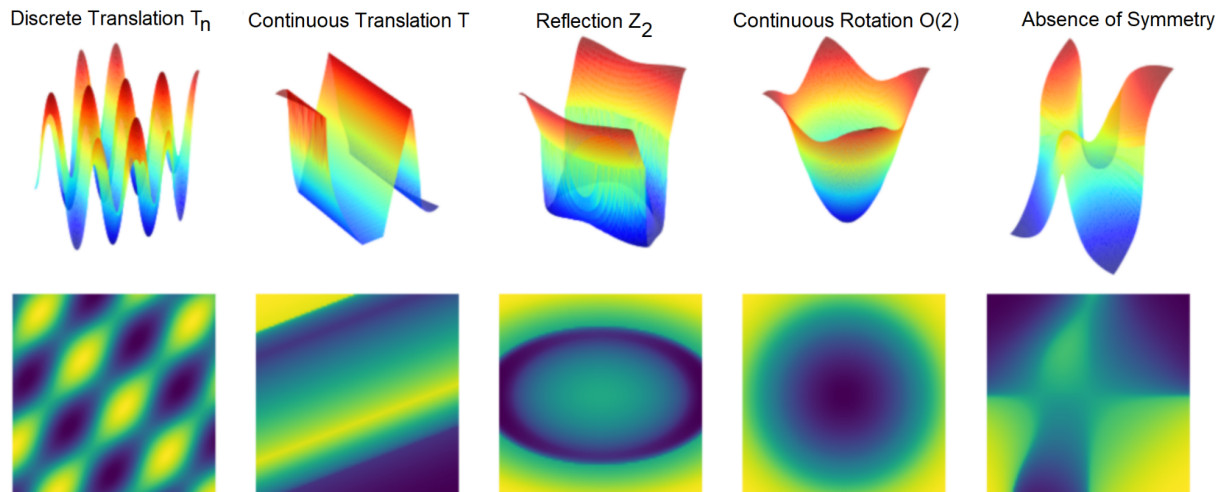
- Max. Learning Rate. There was no significant improvement in the value of RMSE of the points in the validation set for values of Max. LR between 0.01 to 0.001. There was, however, noticeable improvement from Max. LR = 0.1 to Max. LR = 0.01. We chose the value 0.01 as it provided shorter training times.
- Number of Layers. The smallest RMSE of the predictions for the validation set was achieved with 10 hidden layers. More layers resulted in overfitting.
- Number of Neurons per layer. The smallest RMSE of the predictions for the validation set was achieved with 200 neurons in each hidden layer.
- Number of epochs. As there was no significant improvement when increasing the number of epochs beyond 200, this value was chosen with regards to the shorter training time it provided.

**Table 1.** Summary of the results for the hyperparameters of the FCNN that predicts the value of a potential  $V(x,y)$  given  $(x,y)$ . The intensity of the colour is inversely proportional to the RMSE of the predictions of the validation set, meaning that higher intensity colours represent smaller RMSE. Training time increases when moving to the right in the columns of the table.

Max. Learning Rate	0.1		0.01		0.001
Number of Layers	3	5	10	20	
Number of Neurons / Layer	50	100	200	300	500
Number of Epochs	50	100	200	300	500
Training Time					

### 1.2.1 Generate Potentials

The first step is to generate symmetric potentials, functions of 2 variables  $V(x,y)$ . We used the same symmetries that were used in *Symmetry meets AI*:  $O(2)$ ,  $T$ ,  $T_n$ ,  $Z_2$  and  $\emptyset$ , being  $\emptyset$  the absence of symmetry. We generated 1800 different potentials for each of these categories.

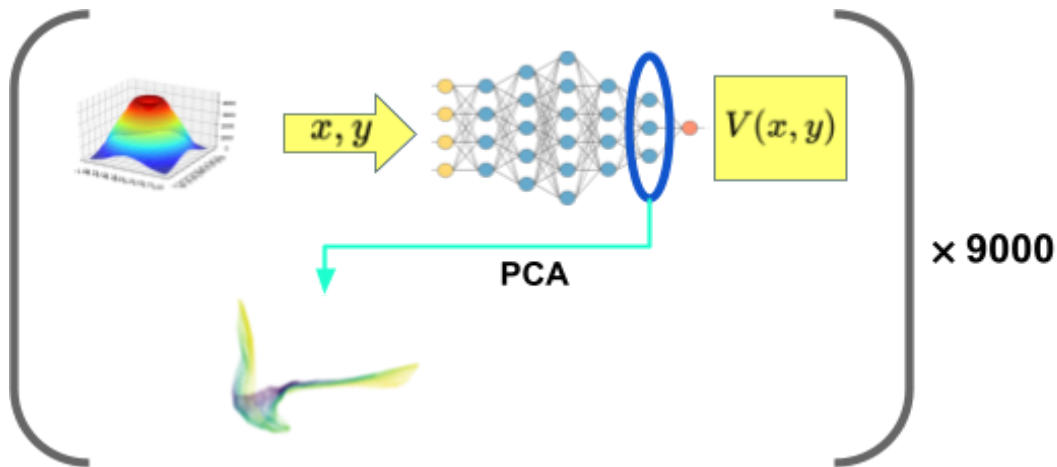


**Figure 1.** Examples of potentials  $V(x,y)$  used for training the FCNNs, and their projection in 2D, where the symmetry is clearly identifiable.

### 1.2.2 Training the FCNNs and obtaining the images from PCA

Using the hyperparameters defined previously, we create a fully connected neural network (FCNN) using the library *fastai*. For each potential we proceed as follows:

1. We create a FCNN and we train it to predict the value of the potential given the pair of variables  $(x,y)$ .
2. After the training, we project the value of the last hidden layer of the network onto a 2D image using PCA: For each pair of variables  $(x,y)$ , we get a response (output) from each of the 200 neurons of the last hidden layer. Treating these responses as coordinates in a 200-coordinate system, we get a point distribution in 200 dimensions. Using PCA, we project the directions that preserve the most information onto a 2D plane, effectively getting an image.
3. We repeat the process 1800 times for each of the five symmetries.



**Figure 2.** Diagram of the process used to obtain images that encode the information from the last hidden layers of the FCNN trained to predict the value of the symmetric potentials.

### 1.2.3 Symmetry detector: CNN

After producing the 9000 images using PCA, we want to test if we are able to classify them according to the symmetry of the original potential. If we are capable of achieving that, it would mean that there is some information about the symmetry encoded in that layer, that we assume to contain the most information about abstract concepts. In that case, the FCNNs that learned to predict the value of the potential did so by making use of its symmetry, thus learning this abstract concept in a certain way.

In order to classify these images, we train a CNN using transfer learning based on ResNet18, ResNet34 and ResNet50. The task is, given the images obtained with PCA, classify them into 5 groups according to the symmetry of the original potential. To get a final model, after training on

the original dataset, we tested all three architectures on images generated by FCNN that performed the same potential-learning task with different architecture. The model that generalised the best was the simplest one, ResNet18, as it got higher accuracy when predicting these new test sets.

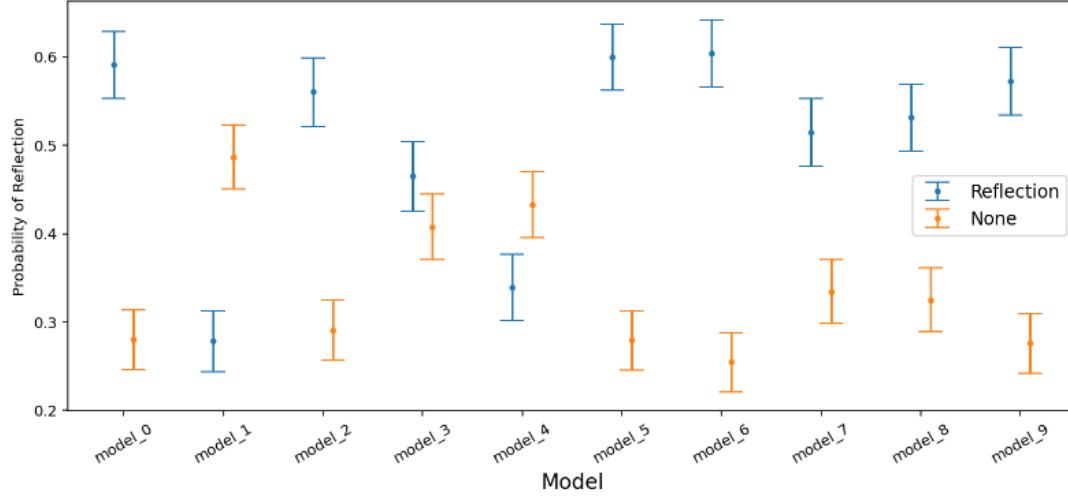
## 1.3 Results

The ResNet18 model was able to classify the images according to the symmetry of their original potentials with an accuracy of 96% in the validation set. It achieved over 90% in other cases with smaller neural networks that learned to predict the potential.

## 2. Initial Symmetry Detector for Databases

We can use the model and the procedures developed in Master's Thesis as the core of a symmetry detector for databases, following these steps:

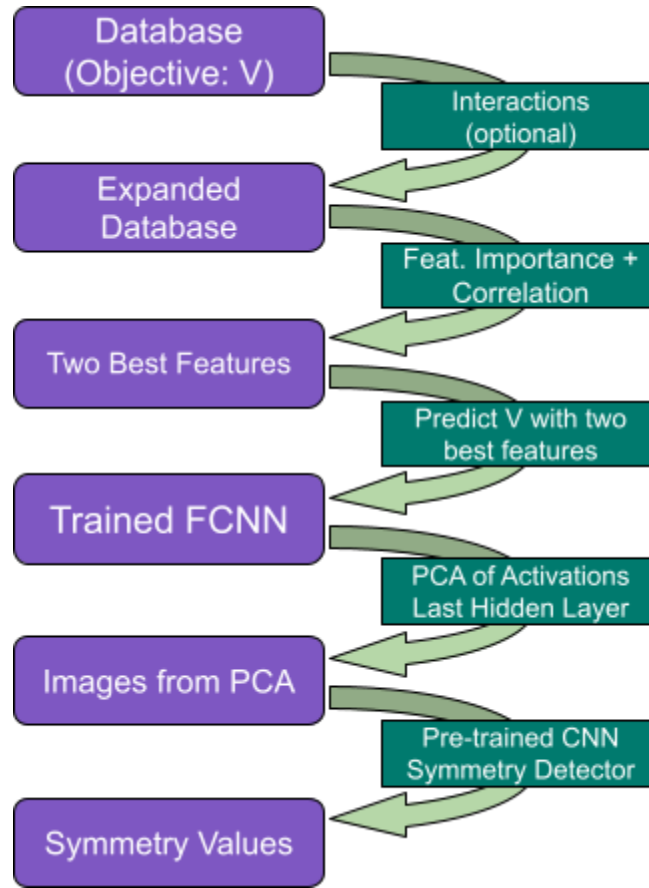
1. Create a mock database to work with.
  - a. Create a symmetric potential as before.
  - b. Add noise variables: Variables with random values
  - c. Interactions: Apply functions to create new features, such as linear combinations of  $x, y$  and the random variables. We can also create them by squaring the existing features or applying any other function to them (exponentiation, square root... ).
  - d. The result is a database with  $x, y, V(x, y)$  and a combination of features
2. Interactions (optional). Apply a series of functions that combine and transform features from a given dataframe to create new ones. This step can slow down considerably the process and I have not enough evidence to confirm that it is beneficial.
3. Feature importance: Get the most important and least correlated features.
  - a. Analyze feature importance using Random Forests (or featurewiz)
  - b. Keep the most important feature.
  - c. For the rest of the features, calculate the correlation with the most important variable.
  - d. Evaluate a metric that takes into account correlation and importance:  
$$e^{\text{Importance}} \times (1 - |\text{correlation}|)^2$$
. See Figure 3.
  - e. Keep the feature with the highest value of the metric. This is the second feature.
4. Train a network to predict the value of the potential (objective) using the two features selected before.
5. Obtain a PCA image from the last hidden layer.
6. Repeat this process several times to reduce statistical fluctuations.
7. Take the average of the predictions of the trained model on the generated PCA images.



**Figure 3.** Comparison of different functions used to choose the second most important variable in the symmetry detector. The list of the models is presented in Table 2.

**Table 2.** Summary of the models tested for the choice of the second variable in the symmetry detector. The performance of each model is presented in Figure 3.

<b>Model 0</b>	$e^{\text{Importance}} \times (1 -  \text{Correlation} )$	<b>Model 5</b>	$\text{Importance} \times (1 -  \text{Correlation} )^2$
<b>Model 1</b>	$\text{Importance}$	<b>Model 6</b>	$e^{\text{Importance}} \times (1 -  \text{Correlation} )$
<b>Model 2</b>	$(1 -  \text{Correlation} )$	<b>Model 7</b>	$e^{\text{Importance}} \times \sqrt{(1 -  \text{Correlation} )}$
<b>Model 3</b>	$\text{Importance} \times (1 -  \text{Correlation} )$	<b>Model 8</b>	$\sqrt{\text{Importance}} \times (1 -  \text{Correlation} )$
<b>Model 4</b>	$\text{Importance}^2 \times (1 -  \text{Correlation} )$	<b>Model 9</b>	$\frac{\text{Importance}}{ \text{Correlation}  + 10^{-20}}$



**Figure 4.** Schematic representation of the Symmetry Detector procedure

The detector had been tested in three scenarios (see figure 5):

1. The original variables  $x, y$  remain in the database, for different number of noise variables.
  - a. On average, it identified correctly the symmetry of the original potential.
2. The original variables  $x, y$  are removed from the database after its creation
  - a. On average, it identified correctly the symmetry of the original potential.
3. The database size is bigger than the original, that was  $100 \times 100$ . We tested it for the case of  $200 \times 200$  and  $500 \times 500$ .
  - a. As the original model was trained on a grid  $100 \times 100$ , it has trouble finding symmetries in bigger databases. However, if we keep just  $100 \times 100$  random elements, discarding the rest, we get acceptable results.

### 3. Symmetry Detector Improved

The symmetry analyzer is an algorithm that, given a database with multiple features and one target, looks for possible 2D symmetries of the target with respect to a pair of features. These features may be part of the original set or functions and combinations of the original ones. Basically, the steps are the following:

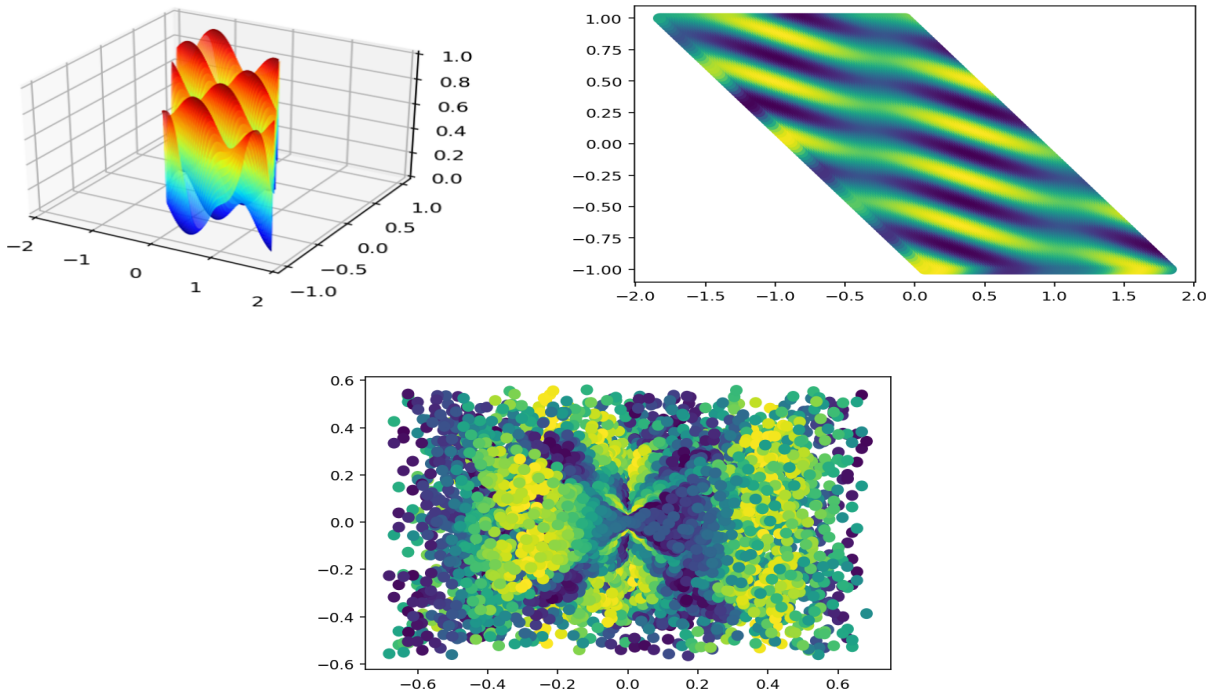
1. Import a database as a pandas DataFrame. If there are any categorical features, use embedding or any algorithm to convert them to numerical values.
2. You can combine features in order to create new ones that may hide some symmetry.
3. From all the features, except the target, find the ones that are highly correlated (linear correlation) between them. Drop those highly correlated features to reduce the size of the database.
4. As the computation time can be an important factor, to further reduce the number of features and drop irrelevant variables, use Random Forest Feature Importance. Order all the features of the database in order of importance and keep the  $n_f$  most important. By doing that, you end up with  $n_f$  features plus the target.
5. From those  $n_f$  features, find the  $\phi_k$  correlation with the target, and keep just the  $n$  features that best correlate with the target.
6. From those  $n$  features, create all the possible unique pairs. There are  $n(n-1)/2$  possibilities. For each pair, evaluate a score function that takes into account the  $\phi_k$  correlation between them and the correlation between the features and the target. One example is, for features  $a$  and  $b$ ,  $score = |\text{corr}(a, \text{targ.})| \times |\text{corr}(b, \text{targ.})| \times (1 - |\text{corr}(a, b)|)$ .
7. In order to further reduce the processing time, one can set a limit on the minimum score necessary to be considered a valid pair. For example, one can keep just half of the features that have the highest score.
8. For each pair of features, train a FCNN to predict the target using just those two features. Use PCA to project the activations of the last hidden layer onto a 2D image.
9. Use a pre-trained CNN to classify the image into one of the 5 symmetry categories. If the prediction is confident enough, record the name of those two features, the probability of the symmetry and the normalised root mean square error (NRMSE) of the predicted values of the target. The NRMSE gives us an idea of the quality of the predictions for those two features, and thus an idea of the confidence we have in the predicted symmetry.
10. Repeat steps 8 and 9 to cover all selected pairs of variables.
11. One can also plot the two features against the target to find out if there is any obvious explanation for the predicted symmetry.



**Test.** Using the original dataframes of the potentials with 126 features generated with different combinations with random variables, we can test this procedure. Let us take a discrete translation symmetric potential and apply this process. We end up with just 13 pairs of variables, after 1.5 minutes. The first value is the value with the highest score and the lowest RMSE (this second result doesn't always happen like this).

	symmetry	feature_1	feature_2	probability	RMSE
0	discrete_translation	-0.9492229813221391y+-0.8789261130466031x	y	1.0000	0.010912
1	discrete_translation	-0.9492229813221391y+-0.8789261130466031x	y**2	0.9979	0.138620
2	discrete_translation	y	-0.6560642586591929y*x	1.0000	0.048751
3	discrete_translation	-0.9492229813221391y+-0.8789261130466031x	0.4883624918871856y**2+0.7212673680780635x**2	1.0000	0.199482
4	discrete_translation	y	0.4883624918871856y**2+0.7212673680780635x**2	1.0000	0.204072
5	discrete_translation	-0.9492229813221391y+-0.8789261130466031x	-0.6560642586591929y*x	1.0000	0.210995
6	discrete_translation	y**2	-0.6560642586591929y*x	1.0000	0.114616
7	none	y**2	0.4883624918871856y**2+0.7212673680780635x**2	0.9874	0.211101
8	continuous_translation	y	y**2	1.0000	0.285456
9	reflection	y	-0.5772320616983628y/x	0.9714	0.075528
10	reflection	-0.9492229813221391y+-0.8789261130466031x	-0.5811099955965784r4**2+-0.654950918902466r1**2	1.0000	0.289792
11	reflection	-0.9492229813221391y+-0.8789261130466031x	-0.5772320616983628y/x	0.9988	0.169958
12	reflection	-0.9492229813221391y+-0.8789261130466031x	0.03717328547018384r3*y	1.0000	0.283265
13	reflection	y	-0.5811099955965784r4**2+-0.654950918902466r1**2	1.0000	0.285342

Plotting the first combination of features, we can see clearly the discrete translation symmetry (this doesn't always happen either). If we were to plot one of the cases that the CNN classified as reflection (being  $V(x,y)$  a discrete translation potential), we could end up with something completely incomprehensible or, just like in the figure below, a more or less clear case of reflection that sprang from a concrete combination of the original features.



## 4. Symmetry Detector Applied to $e^+e^- \rightarrow Z \rightarrow e^+e^-$

Let's now apply the symmetry detector to a case in physics, the process  $e^+e^- \rightarrow Z \rightarrow e^+e^-$ . Starting from the four-momenta of the initial and final particles, we compute the differential cross-section of this process, which we will take as the target that we want to predict. Afterwards, we study the symmetry as we usually do for a general case.

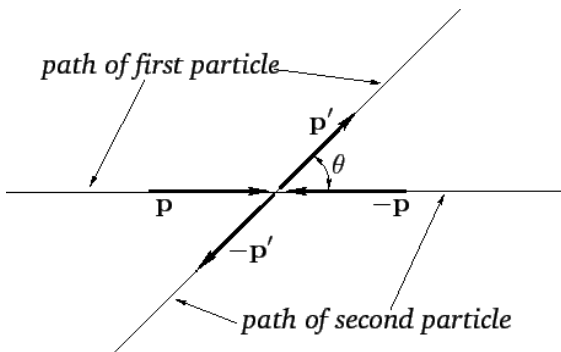
### 4.1 Data and processing

*(todavía no sé muchos detalles, lo completaré más tarde)*

This dataset consists of 50000 events of the process  $e^+e^- \rightarrow Z \rightarrow e^+e^-$  generated with MadGraph5. We will be using the four-momenta of the outgoing particles as well as the fact that all the initial four-momenta are of the form  $p_i = (500, 0, 0, \pm 500)$  GeV.

For the symmetry detector to work, we need a set of variables (features) and a target. This target is usually a function that the initial NN is trained to predict given two features. In our case, we have the 8 components of the final four-momenta. However, not all of them are independent, as we can drop 4 immediately because both particles are electrons in the center of mass reference frame: if we know one four-momentum, we know the other one. As the mass of the electron is a known parameter (it is set to 0 in this dataset because it is 6 orders of magnitude smaller than the momentum) and we also have the energy term fixed, the number of independent variables is reduced to two. We will let the detector choose which variables to keep and which to discard.

In our case, the target will be a value proportional to the differential cross-section of the process. Due to the fact that every value in the dataset will be normalized to meet the criteria of the detector, we don't really care about their absolute value and we can drop any term that remains constant for all of the events. Therefore, the expression of the differential cross-section is just proportional to a function of the  $\theta$  angle (see figure 5).



**Figure 5.** Diagram of two particles scattering. The angle between them,  $\theta$ , can be calculated as  $\cos(\theta) = \mathbf{p} \cdot \mathbf{p}' / |\mathbf{p}| \cdot |\mathbf{p}'|$

The expression for the differential cross-section is proportional<sup>1</sup> to

$$\frac{d\sigma}{d\theta} \propto (g_{V_e}^2 + g_{A_e}^2)^2 (1 + \cos^2 \theta) + 8g_{V_e}^2 g_{A_e}^2 \cos \theta, \quad [1]$$

where  $g_{V_e}$  and  $g_{A_e}$  are the vector and axial vector effective coupling constants. The rest of the terms that have been omitted are the same for every event and therefore irrelevant for our purposes. This expression will be our target. We could modify the value of the couplings and study the effects of this change on the detected symmetry, but for the initial tests we will set  $g_{V_e} = 1$  and  $g_{A_e} = 0$ .

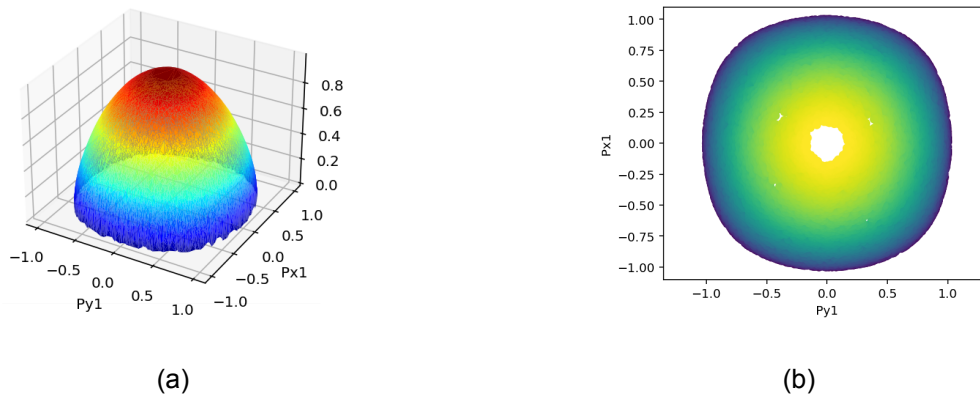
In this particular task,  $\cos\theta$  is calculated using only the value of  $p_z^4$ . We removed this variable  $p_z^4$  from the original dataset that was passed to the neural network.

## 4.2 Results

After applying the symmetry detector to the dataset 1000 times, we found that it always selected two independent variables for training the initial NN. As it was given  $p_x^3, p_y^3, p_x^4, p_y^4, E_3, E_4$  and  $s$ , it always selected a pair of components of the momentum, one in the x direction and one in the y direction, and neglected completely the rest. We can state that the feature selection part was successful.

The average value of the normalized RMSE was about 0.9%, so we can conclude that the NN was able to learn the shape of the target function with good precision. If this was not the case, we would not be able to consider valid the results of the predicted symmetries.

In this case, we expect to get a continuous rotation symmetry result. We can plot the value of the differential cross-section and check this fact, as the represented function will clearly show this symmetry (see figure 6.a and 6.b).

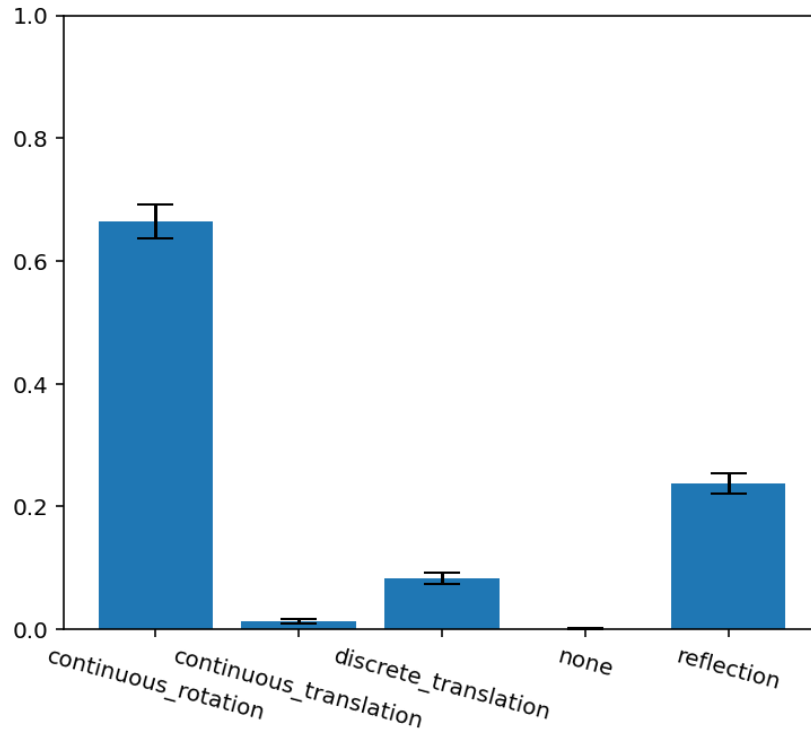


**Figure 6.** Plot of the differential cross section of the process  $e^+ e^- \rightarrow Z \rightarrow e^+ e^-$  in terms of the x and y components of the momentum of one of the final particles. (a) 3D plot. (b) Projection where the color indicates the value of the potential.

<sup>1</sup> Complete expression here:

<http://www.particles.uni-freiburg.de/dateien/vorlesungsdateien/hadroncolliders15/kapitel6>

The results of the detector were as expected, as shown in the figure 7. In the majority of the cases, the algorithm found a continuous rotation symmetry. Maybe due to numerical reasons and to the fact that we always chose a random set of 10000 events, in some cases the symmetry found by the potential was another one, mostly reflection. It is fairly reasonable that reflection symmetry has been found as every function with continuous rotation symmetry is also symmetric under reflection. One important result to consider here is that the detector always found some symmetry, never classifying the target function as having “none” symmetry.



**Figure 7.** Results of the symmetry detector on the  $e^+e^- \rightarrow Z \rightarrow e^+e^-$  differential cross-section dataset. The dominant symmetry is, as expected, the continuous rotation symmetry (67% of the times) followed by reflection (24% of the times).