1. Introduction

Symmetry is very important. This is a fact that the majority of the scientific community would most certainly agree with. We can justify this claim by invoking Noether's theorem, which basically states that for every symmetry inside a system there is a conserved quantity. Thus, finding a symmetry in a system immediately reveals the presence of a quantity that is conserved under some transformation. As abstract as this may sound, the applications of this fact range from the physical laws that describe the most fundamental principles of the world that surrounds us, all the way up to Big Data problems, where finding a symmetry in a database can considerably deepen our understanding of the data it contains.

Even though the concept of symmetry might be an inherently human construct, bearing in mind the technological revolution that Machine Learning and AI fuel, especially in the Physical Sciences [3], one can ask the following question: Is it possible for a computer to understand such a concept? We set out from this question, as the attempt to find an answer can offer valuable insights about Machine Learning and Artificial Intelligence, but also because it is a first step towards the development of a *Symmetry Detector* that would become a potent tool in our quest to acquire knowledge about the world that surrounds us.

The first objective of this study is to develop a generalized model of the algorithm for symmetry classification of the article [2]. In that article, Barenboim, Hirn, and Sanz develop an algorithm that assesses the presence of 4 groups of symmetry and the absence of symmetry in 2D images. This task was based in a binary classification, first training a Fully Connected Neural Network (FCNN) on images of the contours of physical potentials, much like altitude maps of mountains where the contours designate points of the same altitude. The task of the FCNN was basically to predict if a given point belonged to a contour or to the background. This method is therefore limited to a 2D representation with categorical variables, but can be generalized to deal with an arbitrary number of continuous variables by switching from a binary classification to a regression task. This procedure is described in section 2.

The result of the first objective is an algorithm that can look for symmetries in a database, when given the task of predicting one of the features present in that database. This algorithm must be capable of finding the best pairs of features for such task and then look for symmetries after performing it, in the same way as it is done in [2]. Even if the algorithm gives acceptable result in the training tasks that use functions of two variables as targets, a real-world test must be performed in order to confirm its functionality and to prove that its results are consistent, interpretable and credible. This is our second objective: To study a real-world case with real data, where the symmetric properties of the problem are known and can be used to test the results and their meaning. For this purpose, in section 3 we study the physical process $e^+e^- \to Z \to e^+e^-$ using simulated data from the LEP Collider.

2. A Symmetry Detector

2.1. First Step: Regression Task

Our definition of a symmetry detector is an algorithm that can find symmetries in a system, such as an image or a data set, where some kind of prediction task can be performed. In [2], the symmetry detector would look for symmetries in images through a task of reconstructing a set of contours in each image. In our case, we search for symmetries in a data set (for example a csv file) with multiple related features, where a task of predicting one of them in terms of the others can be performed. One of the most classical examples of this type of task would be performing a regression of the salary of a person, given the level of education, the age, the job position, etc.

To construct the symmetry detector, we start with the regression task (the Decoy Task, as it is called in [2]). For each of the five symmetry groups studied in [2], we generate 1800 different 2D potentials $V(x, y)$, functions of two variables which present that symmetry. For each potential, we train a FCNN whose objective is to reproduce the shape of the potential, meaning that given x and y, the FCNN has to be able to predict the value of the potential $V(x, y)$. Examples of those potentials are shown in figure 1. For each of these functions, a FCNN is trained ($1800 \times 5 = 9000$ networks in total).

The potentials, such as the ones shown in figure 1, were generated randomly using the following procedure:

- Define a Function $f : \mathbb{R} \to \mathbb{R}^5 / f(x) = (x, x^2, x^3, x^4, x^5)$. This function can be defined from any power of x to any other higher power, but we chose $x^1 \to x^5$ in order to use the same training potentials as in the article [2].
- Define random potentials for each symmetry using the previous function f:
  - $O(2) : V(x, y) = f(\sqrt{x^2 + y^2}) \cdot \boldsymbol{c}$
  - $T : V(x, y) = f(k_1 x + k_2 y) \cdot \boldsymbol{c}$
  - $T_n : V(x, y) = k_1 \sin(k_2 x + k_3 y) + k_4 \cos(k_5 x + k_6 y)$
  - $Z_2 : V(x, y) = f(k_1 x^2 + k_2 y^2) \cdot \hat{\boldsymbol{C}} \cdot f(k_3 x^2 + k_4 y^2)^t$
  - $\varnothing : V(x, y) = f(x) \cdot \hat{\boldsymbol{C}} \cdot f(y)^t$

  where k_i are random numbers between 0 and 1, $\boldsymbol{c}$ is a vector with 5 random components between 0 and 1 and $\hat{\boldsymbol{C}}$ is a 5×5 matrix with random values between 0 and 1.
- Create a grid of uniformly spaced points: 100 values of x between -1 and 1, and for each value of x, 100 values of y between -1 and 1.
- Calculate the value of the potential for each point of the grid.
- In order to account for the agglomeration of values in some areas and the scarceness in other regions, as well as for points where a divergence of the potential could appear, we redefine the values of the potentials using quantiles. In a nutshell, the point with

the lowest value of the potential (as well as all the points with the same value) acquires the value of 0. The next smallest value of the potential is assigned the value of 2. The next one gets the value of 3. After assigning these new values to every value of $V(x, y)$ that we have we scale them to range between 0 and 1. This procedure reflects better the symmetry of the potential, as shown in figure 1.

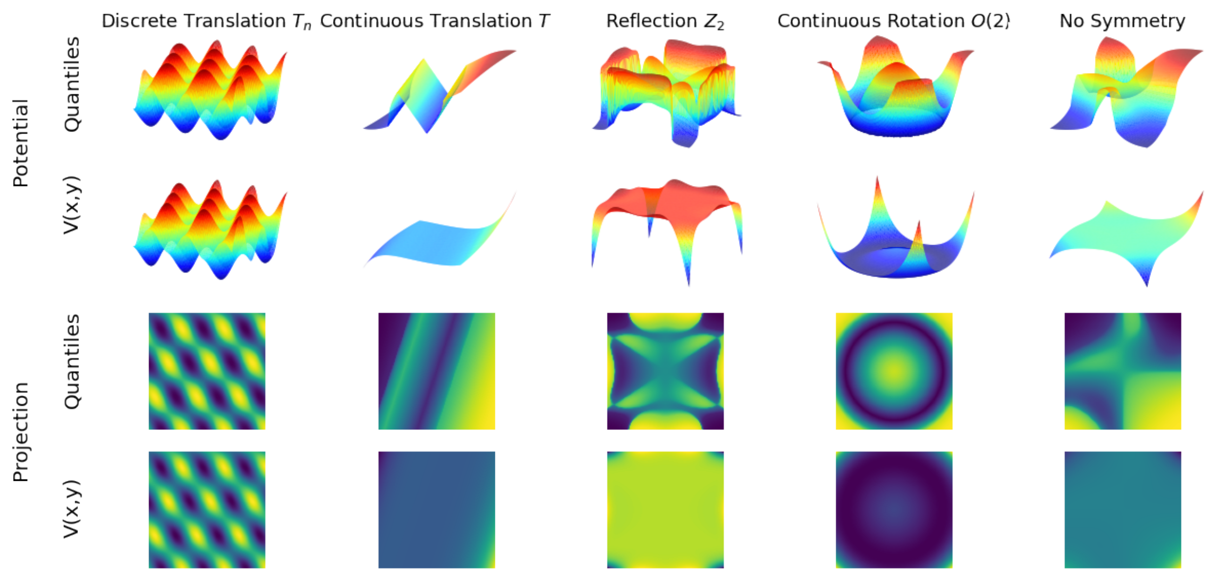

FIGURE 1: Examples of potentials $V(x, y)$ used for training the FCNNs, and their projection in 2D, where the symmetry is clearly identifiable. A comparison between their shape using a quantile transformation or without it is presented. When using quantiles, the local details of the symmetry stand out more, appearing more clearly.

2.2. Architecture of the FCNNs

The architecture of the networks that were trained to learn the shape of the potentials was selected to perform the prediction task with an average Normalized Root Mean Square Error (RMSE) below a threshold of 1% and taking into account computation time constraints. This architecture remained constant throughout the whole process. In order to select the optimal architecture, 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 Hidden 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.

A summary of the choices is presented in table 1.

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	→				

2.3. Second Step: Classification Task

The results of the prediction task are of little importance, as our objective is to find an algorithm that can distinguish between the symmetries of the potentials. In order to check if the behaviour of the FCNNs depends on the symmetry, meaning that the networks learn in a different manner when faced with different symmetries, we analyze the information encoded in the last hidden layer. This procedure is mostly equivalent to the one performed in [2]: perform a Principal Component Analysis (PCA) to reduce the dimensionality and project the information onto a 2D image. If the networks used the symmetric properties of the potentials to learn their shape, we (maybe making use of a computer) would be able to classify the images from PCA in 5 groups, according to the symmetry of the function those networks were trained to predict.

The PCA procedure is a dimensionality reduction technique with the objective to project the most information possible onto two dimensions (in our case). Assume that we want to get a PCA image from a network whose last hidden layer is composed of just three neurons. Take one pair of values $\{a, b\}$ and feed it to the already trained network to ask for $V(a, b)$. Each of the three neurons of the last hidden layer will give an output, an activation. As we have three outputs, we have three numbers that we can treat as coordinates (x, y, z) in a 3D system. This way, each pair of values given to the network will result in one point in a 3D coordinate system.

In our case the last hidden layer is composed of 200 neurons, which means that we have to deal with 200 dimensions, a system that is inherently impossible to imagine for us. However, we can use a simple technique such as PCA to reduce the number of dimensions to just two, projecting as much information as we can onto a regular image. This technique is roughly based on performing a linear fit in the 200 dimension space, keeping the direction that fits the data best. Afterwards, we find the direction that, being orthogonal to the first one, offers the best linear fit for the data. We can perform this procedure until we obtain a new 200-dimensions coordinate system, where each of the coordinates is called a Principal Component. If we now plot just the first two components, the ones that encode the most information about the distribution of the points, we get a 2D image. Some of the images obtained in our case are shown in figure 2.

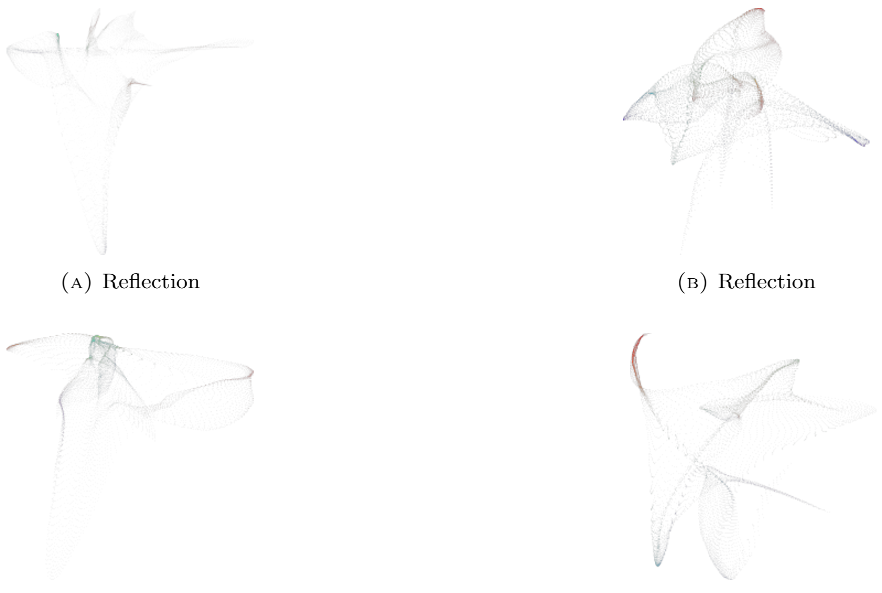
(A) Reflection
(B) Reflection
(C) No symmetry
(D) No symmetry

Figure 2: Examples of images obtained using PCA from the outputs of the last hidden layer of a FCNN trained to predict the value of a potential with reflection symmetry [figures (A) and (B)] and a potential with no symmetry [figures (C) and (D)].

As it is shown in figure 2, from these images it is virtually impossible to conclude anything about the symmetry of the original potential with the naked eye. In fact, sometimes images coming from different symmetries are more similar than some images coming from the same symmetry. For example, the one shown in figure 2a is arguably closer to 2c than to 2b. However, using a virtual eye (i.e. a Convolutional Neural Network, CNN) we can perform this classification task with a much higher degree of precision. Actually, by fine-tuning the pre-trained CNN model for image classification ResNet18 [4], we get a model that can achieve an accuracy of 96% on the test set. If we compare this result to classifying the images randomly, which would result in a 20% accuracy, it is clear that the CNN was able to find patterns in the images from PCA, and such patterns depended on the symmetry of the original potentials. Therefore, we can conclude that the FCNNs that learned the shape of the potential did so by using its symmetry to a certain degree.

2.4. Symmetry Detector: General Procedure

The fact that the symmetry was used in the learning process opens the door to deploying this algorithm onto real databases and look for hidden symmetries there. For a generic database with related features, the procedure for this can be summarized like this:

- Define a Target. The first task we have to perform is a prediction task. This means predicting one feature, one variable, in terms of some other variables, mimicking the *given x and y, predict V(x,y)* case.
- Find the best pairs of features to predict the target. These can be selected manually or an algorithm can be used. Such algorithm may take into account the importance of each feature (using for example Random Forests [5]) and/or its correlation to the target. The correlation can be more complex than the regular Pearson's correlation,

for example one can use the ϕ_K correlation method [1]. The goal is to select pairs of variables that, being good predictors of the target, have little correlation between themselves.

- For each selected pair of variables, train a FCNN to predict the target and project the outputs of the last hidden layer onto a PCA image.
- Use the already trained CNN model for classifying the images resulting from the previous step, and see if they encode information about any symmetry hidden in the database.

In our case, we select the pairs of features following an automated procedure to optimize computation time for databases with a high number of features:

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, like for example calculating linear combinations.
3. From all the features, except the target, find the ones that are highly correlated (linear correlation) between themselves. 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. By default n_f is 10. If there are less than 10 features, n_f is 2/3 of the total. However, this value is arbitrary and you can select as many as you want.
5. From those n_f features, find the ϕ_K correlation with the target, and keep just the n features that best correlate with the target. Typically, if n_f is not very high (for example less than 10), one can ignore the part where the number of features is reduced by setting $n = n_f$.
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 ϕ_K correlation between them and the correlation between the features and the target. One example (the one that we use) 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 pairs that have the highest score.

3. Physics Case: $e^+e^- \to Z \to e^+e^-$

In order to further test the capabilities of this symmetry-detection algorithm, we should apply it to a real database with well known symmetries. Thus, we shall study a case in physics, the process $e^+e^- \to Z \to e^+e^-$. More precisely, we will be studying g_V and g_A, the vector and axial-vector couplings of the interaction respectively and their ratio g_V/g_A. This study is based on determining their value using simulated data and symmetry predictions. Ideally, we want to be able to use the symmetry predictions of our detector to not only find out the measured values of these couplings, but also to maybe offer new insights such as different possibilities for the value of those couplings, that could be related to new symmetry groups.

First, we must get a symmetry score for the *real* data, consisting of 50000 events of the process $e^+e^- \to Z \to e^+e^-$ generated with MadGraph5. From those events, 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 $\boldsymbol{p} = (500, 0, 0, 500)$ GeV.

For the symmetry detector to work, we need a set of variables (features) and a target. One possibility for the target is a value proportional to the differential cross-section. We could evaluate a theoretical expression to find its value for each event. However, as doing that would imply using measured values of g_V and g_A and the value we want to have as a target is something that we can measure, we proceed as follows:

- Calculate the value of $\cos\theta$ (see figure 3) as $\cos\theta = \boldsymbol{p}\cdot\boldsymbol{p'}/|p|\cdot|p'|$, for each event.
- Make an histogram of the values of $\cos\theta$. The height of the bins is the number of events with its value of $\cos\theta$ in a certain range.
- Treat the height of the bin as the target. For each event that lies in a certain $\cos\theta$ range, the value of the target will be the number of events in that same bin, so the function that we are trying to predict is the height of the $\cos\theta$ bins.

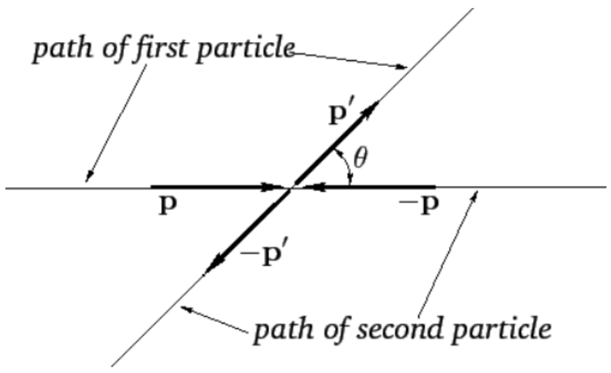

FIGURE 3: Diagram of two particles scattering. The angle between them, θ, can be calculated as $\cos\theta = \boldsymbol{p}\cdot\boldsymbol{p'}/|p|\cdot|p'|$.

Let us discuss now the features that we will use to predict the target. 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 data set because it is 6 orders of magnitude smaller than the momentum) and we also have the energy term fixed (and thus the modulus of the momentum), the number of independent variables is reduced to two. Due to the fact that the initial momenta are fixed and only have z component, we just need p'_z to calculate the target. We will pass the x and y components of the momenta of both final particles and let the detector choose which variables to keep and which to discard.

Every time that the detector is used, the FCNN learns to predict the target in a slightly different way, due to the stochastic nature of the learning process. This may yield different symmetry scores every time it is passes a data set, but we can average over multiple runs to get a more precise score. The average score of 1000 runs is presented in table 2.

TABLE 2: Predictions for the symmetries present in the data set of of 50000 events of the process $e^+e^- \to Z \to e^+e^-$. The errors presented are purely statistical and with a confidence interval of 1σ.

Symmetry	**Score**
Continuous Rotation	0.50 ± 0.02
Continuous Translation	0.009 ± 0.004
Discrete Translation	0.06 ± 0.02
None	0.003 ± 0.003
Reflection	0.43 ± 0.02

After applying the symmetry detector to the data set 1000 times, we found that it always selected two independent variables for training the initial Neural Network. As it was given $\{p'_x, p'_y, -p'_x, -p'_y\}$ 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 automated feature selection part was successful. The average value of the normalized RMSE of the predictions on the validation set was about 0.9%, so we can also conclude that the FCNN 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.

The values presented in table 2 are not very informative on their own. We can observe that we are dealing with a very symmetric process, mainly categorised as having Continuous Rotation and Reflection Symmetry. However, there is no direct way for extracting the value of g_V and g_A or their ratio g_V/g_A. Therefore, we have to simulate synthetic data using a probability distribution for this process that takes into account different ratios.

To generate distributions of $\cos\theta$ for different ratios, we can use a probability distribution proportional to the differential cross-section. If we omit all the factors that are constant for every event and normalize in the interval $[-1, 1]$, we can write

$$\frac{d\sigma}{d\cos\theta} \propto \frac{(g_V^2 + g_A^2)^2(1+\cos^2\theta) + 8g_V^2 g_A^2 \cos\theta}{\frac{8}{3}(g_V^2 + g_A^2)^2} = \mathcal{P}(\cos\theta) \ , \tag{1}$$

where $\mathcal{P}(\cos\theta)$ is normalized so that

$$\int_{-1}^{1} \mathcal{P}(\cos\theta)\, d\cos\theta = 1 \ . \tag{2}$$

Using this expression, we can generate distributions for $\cos\theta$, which can be directly compared to the real ones, as shown in figure 4. Generating distributions and analyzing their symmetries for different values of g_V/g_A, we can compare the results to the values found in table 2, and find out if there is any possible ratio from the simulations that would fit the real data. This comparison is presented in figure 5.

———— Work still in progress ————

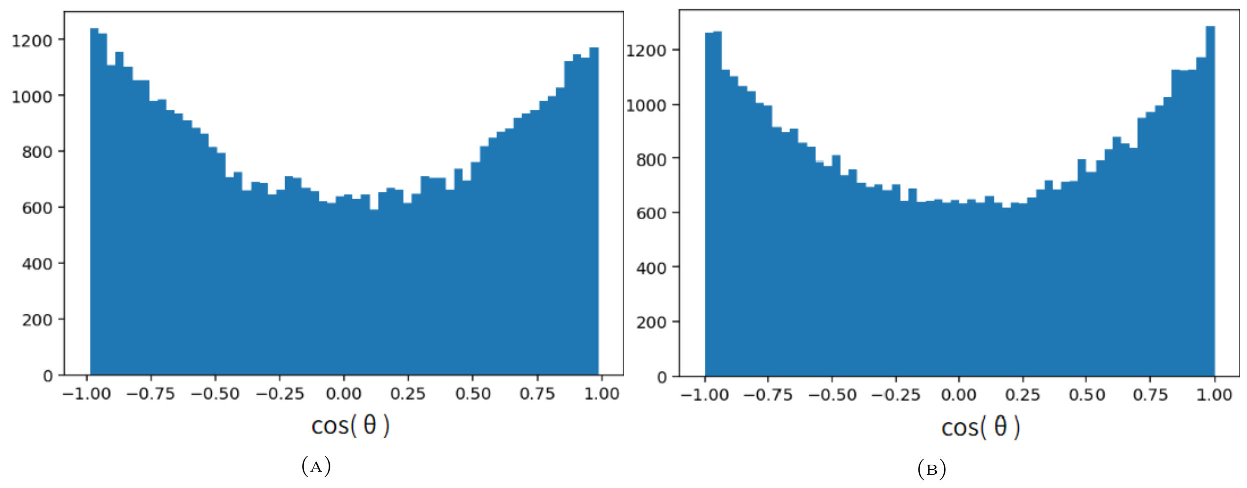

FIGURE 4: Distribution of the values of $\cos\theta$ for a set of 50000 events of $e^+e^- \to Z \to e^+e^-$. (A) Real data generated with `MadGraph5`. (B) Simulated data generated using a simple method in `python`, taking equation (1) as the probability distribution for $\cos\theta$, with $g_V/g_A = 10000$.

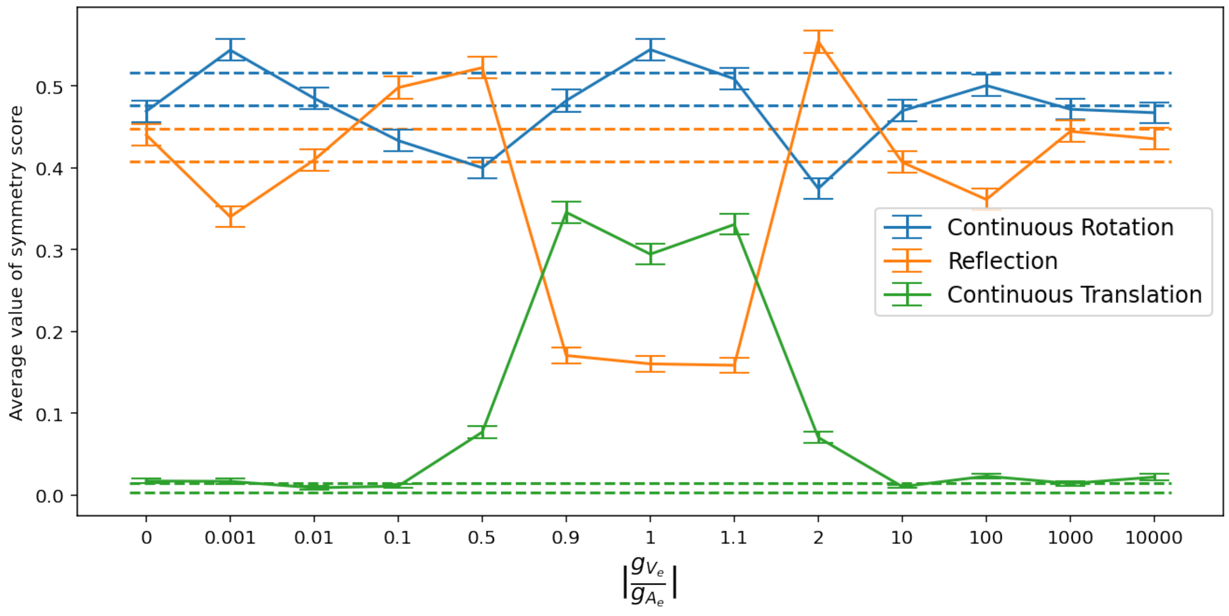

FIGURE 5: Average symmetry scores for distributions of $\cos\theta$, proportional to the differential cross-section from a set of 50000 events of $e^+e^- \to Z \to e^+e^-$, for different values of the absolute value of the ratio of the couplings $|g_V/g_A|$. The dashed line corresponds to the upper and lower limits of the values presented in table 2.

References

[1] M Baak et al. «A new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristics». In: *Computational Statistics & Data Analysis* 152 (2020), p. 107043.

[2] Gabriela Barenboim, Johannes Hirn, and Veronica Sanz. «Symmetry meets AI». In: *arXiv preprint arXiv:2103.06115* (2021).

[3] Giuseppe Carleo et al. «Machine learning and the physical sciences». In: *Reviews of Modern Physics* 91.4 (2019), p. 045002.

[4] Kaiming He et al. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[5] Jeremy Rogers and Steve Gunn. «Identifying feature relevance using a random forest». In: *International Statistical and Optimization Perspectives Workshop" Subspace, Latent Structure and Feature Selection"*. Springer. 2005, pp. 173–184.