Phys. Lett. B 858 (2024) 138996

Contents lists available at ScienceDirect

Physics Letters B

journal homepage: www.elsevier.com/locate/physletb

Letter

Group-invariant machine learning on the Kreuzer-Skarke dataset

Christian Ewert [a], Sumner Magruder [b], Vera Maiboroda [c,d], Yueyang Shen [e], Pragya Singh [f], Daniel Platt [g,*]

[a] German Center for Neurodegenerative Diseases (DZNE), Bonn, Germany
[b] Yale University, New Haven, CT, USA
[c] University Paris-Saclay, Gif-sur-Yvette, France
[d] Alternative Energies and Atomic Energy Commission (CEA) Paris-Saclay, Irfu/DPhP, Gif-sur-Yvette, France
[e] Statistics Online Computational Resource, Computational Medicine and Bioinformatics, University of Michigan, Ann Arbor, MI, USA
[f] Department of Computing, Imperial College London, London, UK
[g] Department of Mathematics, Imperial College London, London, UK

ARTICLE INFO

Editor: N. Lambert

Dataset link: http://www.github.com/danielplatt/kreuzer-skarke-ML

Keywords:
Machine learning
Invariant machine learning
Calabi–Yau
String compactifications
Hodge number
Kreuzer-Skarke

ABSTRACT

We use supervised machine learning to predict Hodge numbers for Calabi-Yau threefolds encoded by reflexive polyhedra. The Hodge number is invariant to the order of the vertices and the swapping of axes. Incorporating these properties, i.e. the invariance of column and row permutations for a matrix containing the polyhedron's vertices, promises better performance for Hodge number prediction. On a medium-sized subset of the Kreuzer-Skarke dataset, we train and evaluate approaches with different degrees of invariance. Our comparison shows that machine learning models incorporating symmetries actually outperform models that do not, with our best model achieving almost 97% accuracy.

1. Introduction

Calabi-Yau threefolds are widely studied objects in pure mathematics and string theory. Heterotic string theory is ten-dimensional and Calabi-Yau threefolds are candidates to construct string compactifications down to four dimensions, with the hope of modelling real-world phenomena. However, most such Calabi-Yau manifolds do not satisfy certain consistency conditions dictated by physics [13, Section 3] and are therefore not suitable as models for string theory. Through the definition of "favourable manifold", one datum that enters the checks of the consistency conditions are the Hodge numbers of the Calabi-Yau manifold [13, p.7]. Many Calabi-Yau threefolds can be obtained as codimension one submanifolds of toric 4-folds, which are encoded by a reflexive polytope obtained as the image of their moment map. There are 473,800,776 reflexive polytopes in dimension 4, which give rise to an even larger number of Calabi-Yau threefolds and a complete list of these polytopes was published by Kreuzer and Skarke [15]. The Hodge numbers of all Calabi-Yau threefolds corresponding to reflexive polytopes have been computed explicitly, but other string theoretic consistency conditions have only been checked for a few Calabi-Yau manifolds. Checking these consistency conditions is a computationally hard problem and it is desirable to have a fast method to identify the most promising candidates before running full computations. This prompted, for example, the works of MacFadden et al. [17] and Berglund et al. [3]. We study the problem of predicting Hodge numbers as a model case to demonstrate that machine learning is suitable as a fast method for this type of check.

Deep learning architectures are often built to make use of the data structure of the inputs to generate an output. For example, multi-layer perceptrons (MLPs) are used for vector-valued inputs and a trained network relies on the fixed order of the vector components for accurate inference. Filters in convolutional neural networks (CNNs) have been designed to implement translation equivariance on image- or tensor-valued inputs. While reflexive polytopes are also encoded as matrices (a stack of their vertices), the Hodge number is invariant to column and row permutations corresponding to a reordering of vertices and

* Corresponding author.
E-mail address: daniel.platt.berlin@gmail.com (D. Platt).

https://doi.org/10.1016/j.physletb.2024.138996
Received 5 April 2024; Received in revised form 21 August 2024; Accepted 28 August 2024
Available online 2 September 2024

a swapping of axes. Incorporating invariance of the output with respect to some action on the inputs promises better performance. In a project at the *London Geometry and Machine Learning (LOGML) Summer School 2022*, we tested various models with different degrees of invariance. In this paper, we compare the performance of the different approaches. Python code for the experiments in this article is available at www.github.com/danielplatt/kreuzer-skarke-ML.

2. Related work

Several supervised machine learning models have been trained to predict Hodge numbers of Calabi-Yau manifolds. This line of research was started by He [12], where a shallow, fully-connected neural network was used to learn Hodge numbers of complete intersection Calabi-Yau threefolds. Following this, more sophisticated neural networks [5,6,9] were applied to the problem, improving the original prediction accuracy. Like the reflexive polytopes dataset, this problem is invariant under the action of a large symmetry group. Aslan et al. [1] applied group-invariant machine learning models to the same dataset further improving performance. Berglund et al. [2] trained a neural network to predict the Hodge numbers of Calabi-Yau threefolds arising from polytopes of the dataset by Kreuzer and Skarke [15]. Machine learning was applied to a dataset of Calabi-Yau manifolds in weighted projective space in Berman et al. [4] and Hirst and Gherardini [14].

3. Dataset and methods

3.1. Dataset

In Kreuzer and Skarke [16], the 473,800,776 reflexive four-dimensional polyhedra were classified, giving rise to (at least) this many Calabi-Yau threefolds. For each reflexive polytope, the dataset lists information about the dual polytope alongside the Euler characteristic and Hodge numbers of the corresponding Calabi-Yau threefold. In dimension four, reflexive polyhedra must have between 5 and 36 vertices and different vertex numbers admit different numbers of polyhedra, with the set of reflexive polyhedra with 16 vertices being the largest. For fast training, to be able to compare many different machine learning models, we consider the medium-sized subset of around 78,000 reflexive polyhedra in dimension four with 26 vertices and their corresponding first Hodge numbers. We use this subset as an example in this paper but we expect similar results for other numbers of vertices. This subset includes polyhedra encoded as matrices $M \in \mathbb{R}^{4\times 26}$ (26 vertices stacked in column vectors) and their corresponding Hodge numbers. We randomly partition the dataset into 80% training data and 20% test data once and use this split for every training and evaluation.

The Hodge number $h^{1,1}(M)$ is invariant to the actions of the group $\mathrm{GL}(4,\mathbb{Z})\times S_{26}$ on M and the dataset contains a single representative in each $\mathrm{GL}(4,\mathbb{Z})\times S_{26}$-orbit. However, there is no canonical way to choose random elements and this group is infinite. Therefore, in this paper, we focus on the Hodge number's invariance to the actions of the subgroup $S_4 \times S_{26}$, i.e. row and column permutations of M. More specifically, applying a row permutation yields an isomorphic polyhedron and applying a column permutation merely changes the order of vertices in M but leaves the polyhedron unchanged.

3.2. Preprocessing

In addition to the dataset above which we refer to as the *original dataset*, we obtain group-invariant features from the dataset by applying the preprocessing step from Aslan et al. [1, Section 2.2], denoted by π and referred to as the *preprocessed dataset*.[1] The rationale behind this is that precomposing any model $\phi : \mathbb{R}^{4\times 26} \to \mathbb{R}$ with an approximately group-invariant map $\pi : \mathbb{R}^{4\times 26} \to \mathbb{R}^{4\times 26}$ yields an approximately group-invariant composition $\phi\circ\pi$. In practice, we implemented this as follows: instead of training our machine learning models ϕ on matrices $M \in \mathbb{R}^{4\times 26}$, we instead used the training data $\pi(M)$. We also build a permuted version of the original dataset where columns and rows of the matrices are randomly permuted (denoted as M_p) and refer to this as the *permuted dataset*. For the last dataset, we apply the aforementioned preprocessing π to obtain group-invariant features on the permuted dataset ($\pi(M_p)$) and refer to this as *preprocessed permuted dataset*.

3.3. Algorithms

In this section, we describe and compare a range of different algorithms. For all models trained by us, we initialised the parameters at random and we optimised the hyperparameters for all models, except for the model *CNN* which serves as a baseline.

Random guess (baseline) For each polytope, we randomly assign one of the 35 Hodge numbers appearing in the dataset.

Majority class only (baseline) For each polytope, we choose the most frequent Hodge number in the dataset, i.e. the Hodge number 16.

MLP with reduced input (baseline) Berglund et al. [2] trained a neural network to predict $h^{1,1}$ from features invariant to column and row permutations, e.g. the number of vertices in the polytope and its dual. While they trained their method on matrices $M \in \mathbb{R}^{4\times k}$ with arbitrary k, we use their results on the entire dataset as a baseline for our subproblem where $k = 26$ and refer to it as *MLP with reduced input*.

XGBoost We used the gradient-boosted decision tree implementation of Chen and Guestrin [7]. All inputs were flattened, we trained with a maximum depth of 6 and used 50% of the $4 * 26 = 104$ features to build each tree. The $L1$ regularisation factor was 1 and the model was trained for a maximum of 60000 epochs and with early stopping using patience 100. The model was trained on the regression objective with Tweedie loss.

CNN (baseline) This neural network consists of a convolutional neural network (CNN) and subsequent multi-layer perceptron (MLP). The CNN consists of 2D convolution layers with kernel size 2 and 32 and 64 channels respectively, ReLU activations, and 2×2 max-pooling layers. The MLP consists of two hidden linear layers with 128 and 50 neurons respectively and ReLU activations. The network was trained for 1000 epochs using the Adam optimiser with early stopping using patience 100. The model is trained on the classification objective with cross-entropy loss.

Vision transformer We use the transformer neural network from Dosovitskiy et al. [8] which has approximately 25 million parameters over 19 layers and uses 4 attention heads. For the attention heads, we use a patch size of 4×4. The $\mathbb{R}^{4\times 26}$ input matrix is augmented to $\mathbb{R}^{52\times 52}$ via stacking of random row and column permutations. The model was trained for 60 epochs using the Adam optimiser on the classification objective with cross-entropy loss.

Invariant MLP This model [11] contains three equivariant layers with 256 channels and cross-connections between layers followed by sum-pooling of each channel and five fully-connected layers with $(256,256,256,256,35)$ neurons. We trained the model with the Adam optimiser and batch size 32 for 300 epochs with early stopping using patience 12. This model is invariant to column and row permutations of the input and uses a mean squared error loss in a regression setting.

[1] We use the implementation from github.com/danielplatt/Fundamental-Domain-Projections.

Table 1
Test accuracies for the task of predicting the first Hodge number of a Calabi-Yau threefold given by a reflexive polytope. Given are the mean performance and sample standard deviation over five runs. "$+\pi$" denotes that the machine learning model was trained on preprocessed data as outlined in Aslan et al. [1, Section 2.2]. We evaluate the methods' performance on the original dataset and a version of the dataset where the columns and rows of each input matrix are randomly permuted.

Model	Accuracy Original Dataset	Accuracy Permuted Dataset
Random Guess	2.86%	2.86%
Majority Class Only	11.28%	11.28%
MLP with reduced input*	46.89%	46.89%
XGBoost	55.02%	29.70%
XGBoost+π	70.63%	59.84%
CNN	56.71 ± 0.38%	30.78 ± 0.49%
CNN+π	71.98 ± 0.61%	61.28 ± 0.35%
Vision Transformer	62.06 ± 0.44%	43.70 ± 0.60%
Vision Transformer+π	69.00 ± 0.48%	61.02 ± 0.63%
Invariant MLP	79.30 ± 0.90%	78.45 ± 0.92%
Invariant MLP+π	78.15 ± 1.73%	77.96 ± 1.50%
PointNet++	95.04 ± 0.39%	86.56 ± 0.52%
PointNet++ +π	95.15 ± 0.97%	90.75 ± 0.44%
MLP with inv. features	**96.86±0.31**%	92.04 ± 0.54%
MLP with inv. features+π	96.66 ± 0.30%	**95.37±0.37**%

* This model was trained and evaluated on a superset containing polytopes with a wide variety of vertices, not just 26 vertices. Furthermore, the inputs contain information about the polytopes' dual (see method description for details).

PointNet++ The PointNet++ model [19] is a neural network architecture designed for the processing of point clouds extending PointNet [18] with local instead of global information aggregation via graphs. We used the implementation from Fey [10]. The model consists of four linear layers with a message-passing mechanism for local feature aggregation. Each of these layers is followed by a ReLU activation and the block of these layers is followed by a global aggregation of features which are used by a single subsequent linear layer to make predictions. We trained the model using the Adam optimiser and batch size 64 for 300 epochs with early stopping using patience 15. This model is invariant under column permutations, but it is *not* invariant under row permutations, as the coordinates of the vertices are the inputs to the first layer of the network. It was trained with a cross-entropy loss function.

MLP with inv. features This model consists of a column- and a row-encoder and a single decoder. Each encoder contains a single MLP that is applied independently to each column or row of the input. Per column or row, the respective encoder obtains a feature of length 256. Max-pooling operations are applied separately across column and row features similar to the pooling in PointNet [18]. The column-encoder obtains a feature invariant to column permutations while the row-encoder obtains a feature invariant to row permutations. Both features are added and processed by the decoder into the Hodge number prediction. The MLPs in the encoder contain two hidden and one output layer with 256 neurons while the MLP in the decoder contains three hidden layers with 256 neurons and the output layer. The decoder MLP uses batch normalization before the activation. Except for the very last layer, all layers have a ReLU activation. The model was trained in a classification setting with a cross-entropy loss function, Adam optimiser, and batch size 32 for 300 epochs with early stopping using patience 12.

4. Results and discussion

In this section, we compare the methods described in Section 3.3 on the test sets of the datasets described in Section 3.2: the original and permuted datasets as well as group-invariant features based on these datasets, i.e. the preprocessed (denoted by $+\pi$) and preprocessed permuted datasets. For every method, we train one instance on the original dataset and one on the preprocessed dataset and report the accuracies on the respective corresponding test sets. In addition and without retraining, we compare these instances on the permuted versions of the datasets. We list the accuracies in Table 1 and visualise these results in Fig. 1. Note that the two models *MLP with reduced input* and *Invariant MLP* are already invariant under $S_4 \times S_{26}$, so composing with π makes no difference.

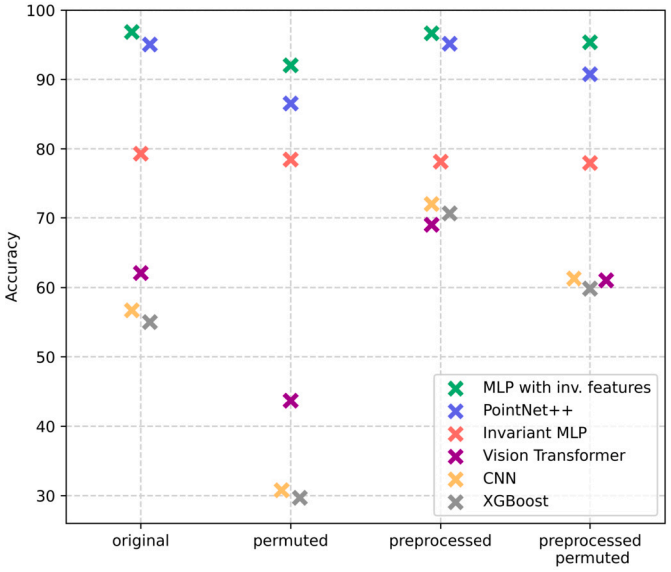

Fig. 1. Performance summary of models across the four different datasets: the *original* dataset, a random permutation of the original dataset (*permuted*), the group-invariant preprocessing applied to the original (*preprocessed*) and applied to the randomly permuted dataset (*preprocessed permuted*).

We make the following observations:

1. High accuracy: Our two best models predict Hodge numbers with accuracies above 95%.
2. Classification seems to perform better than regression: While it might seem more natural to formulate the task of predicting Hodge numbers as a regression task, some teams found that a formulation as a classification task, i.e. a multi-label output, performed better. Our best methods also use a classification setting.
3. Accuracy declines significantly (29.6-46%) if inputs are permuted and models are not permutation-invariant: Models such as *XGBoost*, *CNN*, and *Vision Transformer* are not by design permutation-invariant and have also not been trained with suitable data augmentation. As expected, their performance declines when permuted versions of the inputs are presented (see Table 1).
4. Building in group invariance improves performance: We notice that the models *MLP with inv. features*, *Invariant MLP*, and *PointNet++* perform better than other models that have no group-invariance built in. More specifically, the *Invariant MLP* is invariant under row and column permutations, *PointNet++* is invariant under column permutations, and the *MLP with inv. features* contains parts to extract features invariant under row and column permutations but not both.
5. Preprocessing step π improves performance: We note that making the models more group-invariant by applying the preprocessing step π increases performance for all models that are not already fully group-invariant on the task in which input matrices were randomly permuted. For the models *XGBoost*, *CNN*, and *Vision Transformer*, performance improves more than two times the sample standard deviation, for the other models the performance change (a decrease in two cases) is less than two times the sample standard deviation. For the models *MLP with inv. features* and *PointNet++*, performance is further improved by applying the preprocessing step π for the scenario with random permutations.

6. Partially invariant models outperform fully invariant models: We observe that the two partially invariant models *MLP with inv. features* and *PointNet++* perform better than the fully group-invariant model *Invariant MLP*. We assume that partially invariant models are a compromise between *performance* and *generalisability* (to input changes that they are invariant to). On one hand, designing a neural network with built-in invariance involves enforcing the retrieval of invariant features which can introduce bottlenecks and over-constrain its architecture which, in turn, negatively affects its performance. On the other hand, neglecting the invariances, architectures can have sufficient capacity without bottlenecks but lack the desired invariances and generalisability. Therefore, partially invariant models perform well because they are not over-constrained and additionally benefit from increased generalisability caused by their partial invariance.
7. Polytopes with other numbers of vertices: We conducted our experiments on the dataset of polytopes with 26 vertices. In addition, we also trained and tested a shallow neural network analogously on datasets containing polytopes with other numbers of vertices to test if our findings can be replicated on these datasets. Though the overall accuracy between vertex numbers varied (lower for fewer vertices and higher for more vertices), we found points 2, 3, and 5 confirmed on these other datasets. Due to computational constraints, we did not train all models on all datasets.

5. Conclusion

Our experiments show that the Hodge numbers of Calabi-Yau manifolds defined by reflexive polytopes can be predicted with good accuracy using machine learning.

Furthermore, we observe that building group invariance into the machine learning models improves performance: we see that models that are by definition invariant or partially invariant perform better than models that are not. We also see that models that are not group-invariant benefit from applying a group-invariant preprocessing step in most cases.

Interestingly, our strongest models are those that are partially invariant but not invariant under the full group. We assume that this is the case because the partially invariant models are a compromise between *performance* and *generalisability* (to the input changes that they are invariant to). More specifically, partially invariant models perform well as their architectures are not over-constrained to implement full invariance while benefitting from increased generalisability through their partial invariance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is available at http://www.github.com/danielplatt/kreuzer-skarke-ML.

Acknowledgements

This project was started during the *London Geometry and Machine Learning 2022* summer school, and we thank the organisers for creating such a positive and productive environment. We thank Benjamin Aslan and David Sheard for helpful conversations. We also thank the anonymous referee for her or his very helpful suggestions which led to great improvements of the manuscript. DP gratefully acknowledges funding from the Simons Collaboration in Special Holonomy. YS acknowledges funding from Statistical Online Computational Resources (SOCR). PS thanks Imperial College London for its support. The work was carried out independently and Imperial was not involved as an official partner.

References

[1] B. Aslan, D. Platt, D. Sheard, Group invariant machine learning by fundamental domain projections, in: S. Sanborn, C. Shewmake, S. Azeglio, A. Di Bernardo, N. Miolane (Eds.), Proceedings of the 1st NeurIPS Workshop on Symmetry and Geometry in Neural Representations, 2023, pp. 181–218, https://proceedings.mlr.press/v197/aslan23a.html.

[2] P. Berglund, B. Campbell, V. Jejjala, Machine learning Kreuzer–Skarke Calabi–Yau threefolds, arXiv URL: https://arxiv.org/abs/2112.09117, 2021.

[3] P. Berglund, Y.H. He, E. Heyes, E. Hirst, V. Jejjala, A. Lukas, New Calabi–Yau manifolds from genetic algorithms, Phys. Lett. B 850 (2024) 138504, https://doi.org/10.1016/j.physletb.2024.138504.

[4] D.S. Berman, Y.H. He, E. Hirst, Machine learning Calabi-Yau hypersurfaces, Phys. Rev. D 105 (2022) 066002, https://doi.org/10.1103/PhysRevD.105.066002.

[5] K. Bull, Y.H. He, V. Jejjala, C. Mishra, Machine learning CICY threefolds, Phys. Lett. B 785 (2018) 65–72, https://doi.org/10.1016/j.physletb.2018.08.008.

[6] K. Bull, Y.H. He, V. Jejjala, C. Mishra, Getting CICY high, Phys. Lett. B 795 (2019) 700–706, https://doi.org/10.1016/j.physletb.2019.06.067.

[7] T. Chen, C. Guestrin, Xgboost: a scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, 2016, pp. 785–794.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16x16 words: transformers for image recognition at scale, in: International Conference on Learning Representations, 2021, https://openreview.net/pdf?id=YicbFdNTTy.

[9] H. Erbin, R. Finotello, Inception neural network for complete intersection Calabi–Yau 3-folds, Mach. Learn.: Sci. Technol. 2 (2021) 02LT03, https://doi.org/10.1088/2632-2153/abda61.

[10] M. Fey, Point cloud classification, https://colab.research.google.com/drive/1D45E5bUK3gQ40YpZo65ozs7hg5l-eo_U.

[11] J. Hartford, D. Graham, K. Leyton-Brown, S. Ravanbakhsh, Deep models of interactions across sets, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 1909–1918, https://proceedings.mlr.press/v80/hartford18a/hartford18a.pdf.

[12] Y.H. He, Machine-learning the string landscape, Phys. Lett. B 774 (2017) 564–568, https://doi.org/10.1016/j.physletb.2017.10.024.

[13] Y.H. He, S.J. Lee, A. Lukas, C. Sun, Heterotic model building: 16 special manifolds, J. High Energy Phys. 2014 (2014), https://doi.org/10.1007/jhep06(2014)077.

[14] E. Hirst, T.S. Gherardini, Calabi-Yau four-, five-, sixfolds as $\mathbb{P}^n_{\mathbf{w}}$ hypersurfaces: machine learning, approximation, and generation, Phys. Rev. D 109 (2024) 106006, https://doi.org/10.1103/PhysRevD.109.106006.

[15] M. Kreuzer, H. Skarke, CY/4d: reflexive polyhedra in 4 dimensions, http://hep.itp.tuwien.ac.at/~kreuzer/CY/.

[16] M. Kreuzer, H. Skarke, Complete classification of reflexive polyhedra in four dimensions, Adv. Theor. Math. Phys. 4 (2000) 1209–1230, https://doi.org/10.4310/atmp.2000.v4.n6.a2.

[17] N. MacFadden, A. Schachner, E. Sheridan, The DNA of Calabi-Yau hypersurfaces, arXiv: https://arxiv.org/abs/2405.08871, 2024.

[18] C.R. Qi, H. Su, K. Mo, L.J. Guibas, PointNet: deep learning on point sets for 3D classification and segmentation, in: IEEE Conference on Computer Vision and Pattern Recognition 2017, IEEE Computer Society, 2017, pp. 77–85.

[19] C.R. Qi, L. Yi, H. Su, L.J. Guibas, PointNet++: deep hierarchical feature learning on point sets in a metric space, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc., 2017, https://proceedings.neurips.cc/paper_files/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.