Machine learning for complete intersection Calabi–Yau manifolds

Harold Erbin

Mit (Usa) & Cea-List (France)

In collaboration with:

- Riccardo Finotello (Università di Torino)

arXiv: 2007.13379, 2007.15706

Massachusetts Institute of Technology

Funded by the European Union (Horizon 2020)

list cea tech

Outline: 1. Motivations

Motivations

Machine learning

Calabi–Yau 3-folds

Data analysis

Machine learning analysis

Conclusion

Machine learning

Machine Learning (ML)

Set of techniques for pattern recognition / function approximation without explicit programming.

- learn to perform a task implicitly by optimizing a cost function
- flexible $\rightarrow$ wide range of applications
- general theory unknown (black box problem)

Machine learning

Machine Learning (ML)

Set of techniques for pattern recognition / function approximation without explicit programming.

- learn to perform a task implicitly by optimizing a cost function
- flexible $\rightarrow$ wide range of applications
- general theory unknown (black box problem)

Question

Where does it fit in theoretical physics?

Machine learning

Machine Learning (ML)

Set of techniques for pattern recognition / function approximation without explicit programming.

- learn to perform a task implicitly by optimizing a cost function
- flexible $\rightarrow$ wide range of applications
- general theory unknown (black box problem)

Question

Where does it fit in theoretical physics?

- particle physics
- cosmology
- many-body physics
- quantum information
- lattice theories
- string theory

[1903.10563, Carleo et al.]

String phenomenology

Goal

Find "the" Standard Model from string theory

Method:

- type II / heterotic strings, M-theory, F-theory: $D = 10, 11, 12$
- vacuum choice (flux compactification):
  - typically Calabi–Yau (CY) 3- or 4-fold
  - fluxes and intersecting branes

  $\to$ reduction to $D = 4$
- check consistency (tadpole, susy...)
- read the $D = 4$ QFT (gauge group, spectrum...)

String phenomenology

Goal

Find “the” Standard Model from string theory

Method:

- type II / heterotic strings, M-theory, F-theory: $D = 10, 11, 12$
- vacuum choice (flux compactification):
  - typically Calabi–Yau (CY) 3- or 4-fold
  - fluxes and intersecting branes

  $\rightarrow$ reduction to $D = 4$
- check consistency (tadpole, susy. . .)
- read the $D = 4$ QFT (gauge group, spectrum. . .)

No vacuum selection mechanism $\Rightarrow$ string landscape

Landscape mapping

String phenomenology:

- find consistent string models
- find generic/common features
- reproduce the Standard model

Landscape mapping

String phenomenology:

- find consistent string models
- find generic/common features
- reproduce the Standard model

Typical questions:

- understand manifolds
- find parameter distribution
- explore consistent vacua
- find good EFTs for low-energy limit

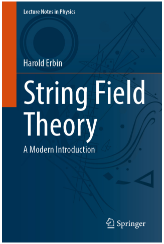
Landscape mapping

String phenomenology:

- find consistent string models
- find generic/common features
- reproduce the Standard model

Typical questions:

- understand manifolds
- find parameter distribution
- explore consistent vacua
- find good EFTs for low-energy limit
- (construct an explicit string field theory)



(to appear in 02/2021)

Number of geometries

Calabi–Yau (CY) manifolds

- CICY (complete intersection in products of projective spaces): 7890 (3-fold), 921,497 (4-fold)
- Kreuzer–Skarke (reflexive polyhedra): 473,800,776 ($d = 4$)

String models and flux vacua

- type IIA/IIB models: 10^{500}
- F-Theory: 10^{755} to 10^{3000} (geometries), $10^{272,000}$ (flux vacua)

[Lerche-Lüst-Schellekens '89; hep-th/0303194, Douglas; hep-th/0307049, Ashok-Douglas; hep-th/0409207, Douglas; 1511.03209, Taylor-Wang; 1706.02299, Halverson-Long-Sun; 1710.11235, Taylor-Wang; 1810.00444, Constantin-He-Lukas]

Challenges

- huge number of possibilities
- difficult math problems (NP-complete, NP-hard, undecidable) [hep-th/0602072, Denef-Douglas; 1009.5386, Cvetič-García-Etxebarria-Halverson; 1809.08279, Halverson-Ruehle; 1911.07835, Halverson-Plesser-Ruehle-Tian]
- methods from algebraic topology: cumbersome, rarely closed-form formulas [Lukas' talk, 07/2020, 20-22']

Challenges

- huge number of possibilities
- difficult math problems (NP-complete, NP-hard, undecidable) [hep-th/0602072, Denef-Douglas; 1009.5386, Cvetič-García-Etxebarria-Halverson; 1809.08279, Halverson-Ruehle; 1911.07835, Halverson-Plesser-Ruehle-Tian]
- methods from algebraic topology: cumbersome, rarely closed-form formulas [Lukas' talk, 07/2020, 20-22']

$\rightarrow$ use machine learning

Selected references: 1404.7359, Abel-Rizos; 1706.02714, He; 1706.03346, Krefl-Song; 1706.07024, Ruehle; 1707.00655, Carifio-Halverson-Krioukov-Nelson; 1804.07296, Wang-Zang; 1806.03121, Bull-He-Jejjala-Mishra; ...

Review: Ruehle '20

Plan

Goal

Compute Hodge numbers for CICY 3-folds

1. introduction to machine learning
2. complete intersection Calabi–Yau (CICY)
3. data analysis for CICY
4. machine learning for CICY

References: [HE-Finotello, 2007.13379, 2007.15706]

Outline: 2. Machine learning

Motivations

Machine learning

Calabi–Yau 3-folds

Data analysis

Machine learning analysis

Conclusion

Definition

Machine learning (Samuel)

The field of study that gives computers the ability to learn without being explicitly programmed.

Machine learning (Mitchell)

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

References:

- Chollet – *Deep Learning with Python* (2018)
- Géron – *Hands-On Machine Learning* (2019)
- Zhang et al. – *Dive Into Deep Learning* (2020), `d2l.ai`

Approaches to machine learning

Learning approaches (task: $x \longrightarrow y$):

- supervised: learn a map from a set or pairs $(x_{\text{train}}, y_{\text{train}})$, then predict y_{data} from x_{data}
- unsupervised: give x_{data} and let the machine find structure (i.e. appropriate y_{data})
- reinforcement: give x_{data}, let the machine choose output y_{data} following some rules, reward good and/or punish bad results, iterate

Applications

General idea = pattern recognition:

- classification / clustering
- regression (prediction)
- transcription / translation
- structuring
- anomaly detection
- denoising
- synthesis and sampling
- density estimation
- conjecture generation

Applications in industry: computer vision, language processing, medical diagnosis, fraud detection, recommendation system, autonomous driving...

Examples

Multimedia applications:

- MuZero, AlphaZero (DeepMind): play chess, shogi, Go
- MuZero, AlphaStar (Deepmind), OpenAI Five, etc.: play video games (Starcraft 2, Dota 2, Atari...)
- GPT-2 (OpenAI): conditional synthetic text sampling (+ general language tasks)
- DeepL: translation
- Yolo: real-time object detection [1804.02767]
- Face2Face: real-time face reenactement (deep fake)

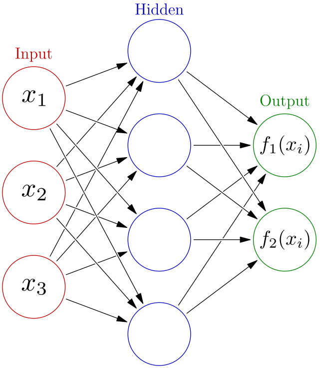
Science applications:

- AlphaFold (DeepMind): protein folding
- (astro)particles [1806.11484, 1807.02876, darkmachines.org]
- astronomy [1904.07248]
- geometrical structures [geometricdeeplearning.com]

Fully connected neural network

$$x_{i_0}^{(0)} := x_{i_0}$$
$$x_{i_1}^{(1)} = g^{(1)}(W_{i_1 i_0}^{(1)} x_{i_0}^{(0)})$$
$$f_{i_2}(x_{i_0}) := x_{i_2}^{(2)} = g^{(2)}(W_{i_2 i_1}^{(2)} x_{i_1}^{(1)})$$

$$i_0 = 1, 2, 3;\ i_1 = 1, \ldots, 4;\ i_2 = 1, 2$$

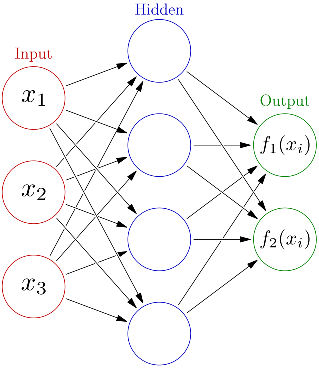


Fully connected neural network

Architecture:

- $N \geq 1$ hidden layers, vector $x^{(n)}$
- link: weighted input $\rightarrow$ weight matrix $W^{(n)}$
- neuron: non-linear, element-wise activation function $g^{(n)}$

$$x^{(n+1)} = g^{(n+1)}(W^{(n)}x^{(n)})$$



Training

Method:

1. fix architecture (number of layers, activation functions. . .)
2. learn weights $W^{(n)}$ from gradient descent

Gradient descent:

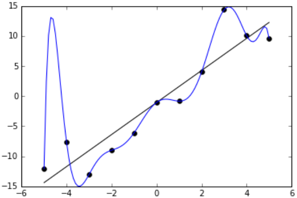
- loss function L: overall error to be decreased

$$L = \sum_{i=1}^{N_{\text{train}}} \mathrm{distance}(y_i^{(\text{train})}, y_i^{(\text{pred})})$$

  common choices: mean absolute error, mean squared error. . .
- optimizer and its parameters (learning rate, momentum. . .)
- ℓ_1 and ℓ_2 weight regularization (penalize high and redundant weights)
- training protocol: early stopping, learning rate decay. . .

Training cycle

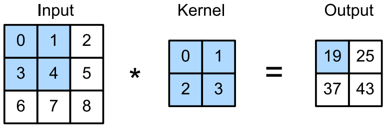
- hyperparameter tuning
  - adapt architecture and optimization for better results
  - search methods: trial-and-error, grid, random, Bayesian, genetic. . .
- main risk: overfitting (= cannot generalize to new data)
  1. split data in training, validation and test sets
  2. train several models on the training set
  3. compare performances on validation set
  4. evaluate performance of the best model on test set



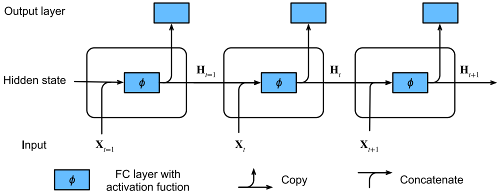
- consider n models in parallel (bagging) to get statistics

Neural network components (1)

- convolutional layer: move window over data, combining values with a kernel (to be learned)
  $\rightarrow$ translation covariance, locality, weight sharing

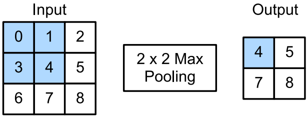


- recurrent layer (LSTM, GRU): keep memory of past information in a sequence
  $\rightarrow$ temporal processing

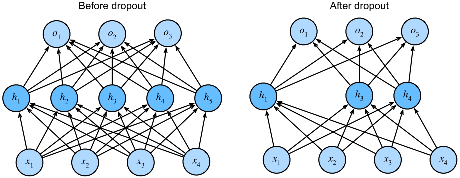


Neural network components (2)

- pooling layer: coarse-graining
  → reduce internal data size, translation/rotation/scale invariances



- dropout layer: deactivate neurons randomly with probability p
  → improve generalization, regularization



- batch normalization layer: normalize data, then scale and shift (learnable parameters)
  → keep stable internal data, regularization

Images: Dive into deep learning

ML workflow

"Naive" workflow:

1. get raw data
2. write neural network with many layers
3. feed raw data to neural network
4. get nice results (or give up)

`xkcd.com/1838`

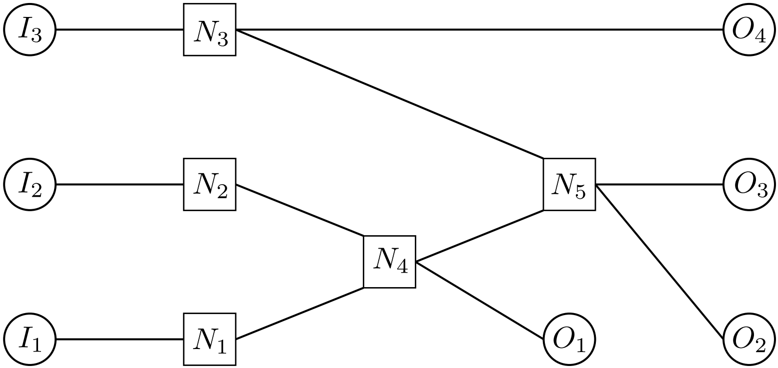
ML workflow

Real-world workflow:

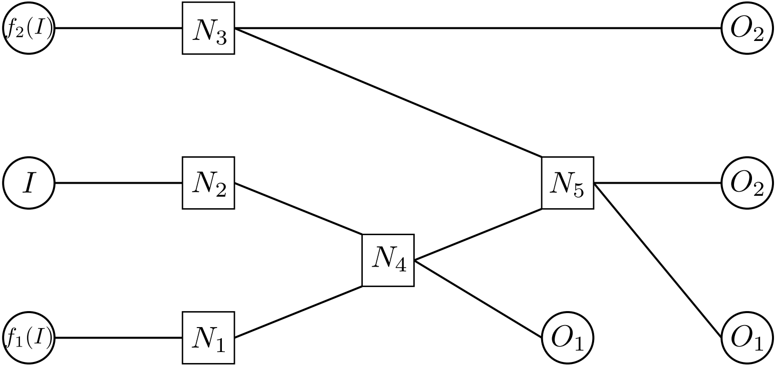
1. understand the problem
2. exploratory data analysis
   - feature engineering
   - feature selection
3. baseline model
   - full working pipeline
   - lower-bound on accuracy
4. validation strategy
5. machine learning model(s)
6. ensembling

Pragmatic ref.: `coursera.org/learn/competitive-data-science`

Advanced neural network



Advanced neural network



Particularities:

- $f_i(I)$: engineered features
- identical outputs (stabilisation)

Why neural networks?

Universal approximation theorem

Under mild assumptions, a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$.

Why neural networks?

Universal approximation theorem

Under mild assumptions, a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$.

Comparisons

- results comparable and sometimes superior to human experts (cancer diagnosis, traffic sign recognition. . .)
- perform generically better than any other machine learning algorithm

Why neural networks?

Universal approximation theorem

Under mild assumptions, a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$.

Comparisons

- results comparable and sometimes superior to human experts (cancer diagnosis, traffic sign recognition...)
- perform generically better than any other machine learning algorithm

Drawbacks

- black box
- magic
- numerical

(= how to extract analytical / predictable / exact results?)

Outline: 3. Calabi–Yau 3-folds

Motivations

Machine learning

Calabi–Yau 3-folds

Data analysis

Machine learning analysis

Conclusion

Calabi-Yau

Complete intersection Calabi–Yau (CICY) 3-fold:

- CY: complex manifold with vanishing first Chern class
- complete intersection: non-degenerate hypersurface in products of m projective spaces
- hypersurface $=$ solution to system of k homogeneous polynomial equations

Calabi-Yau

Complete intersection Calabi–Yau (CICY) 3-fold:

- CY: complex manifold with vanishing first Chern class
- complete intersection: non-degenerate hypersurface in products of m projective spaces
- hypersurface = solution to system of k homogeneous polynomial equations
- described by configuration matrix $m \times k$

$$X = \left[\begin{array}{c|ccc} \mathbb{P}^{n_1} & a_1^1 & \cdots & a_k^1 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{P}^{n_m} & a_1^m & \cdots & a_k^m \end{array} \right], \qquad a_\alpha^r \in \mathbb{N}$$

$$\dim_{\mathbb{C}} X = \sum_{r=1}^{m} n_r - k = 3, \qquad n_r + 1 = \sum_{\alpha=1}^{k} a_\alpha^r$$

- a_α^r power of coordinates on $\mathbb{P}^{n_r}$ in αth equation

Configuration matrix

Examples

- quintic $(a = 0, \dots, 4)$

$$\left[\ \mathbb{P}^4_x \ \middle|\ 5\ \right] \implies \sum_a (X^a)^5 = 0$$

- 2 projective spaces, 3 equations $(a, \alpha = 0, \dots, 3)$

$$\left[\begin{array}{c|ccc} \mathbb{P}^3_x & 3 & 0 & 1 \\ \mathbb{P}^3_y & 0 & 3 & 1 \end{array}\right] \implies \begin{cases} f_{abc} X^a X^b X^c = 0 \\ g_{\alpha\beta\gamma} Y^\alpha Y^\beta Y^\gamma = 0 \\ h_{a\alpha} X^a Y^\alpha = 0 \end{cases}$$

Configuration matrix

Examples

- quintic ($a = 0, \dots, 4$)

$$\left[\begin{array}{c|c} \mathbb{P}^4_x & 5 \end{array}\right] \implies \sum_a (X^a)^5 = 0$$

- 2 projective spaces, 3 equations ($a, \alpha = 0, \dots, 3$)

$$\left[\begin{array}{c|ccc} \mathbb{P}^3_x & 3 & 0 & 1 \\ \mathbb{P}^3_y & 0 & 3 & 1 \end{array}\right] \implies \begin{cases} f_{abc}\, X^a X^b X^c = 0 \\ g_{\alpha\beta\gamma}\, Y^\alpha Y^\beta Y^\gamma = 0 \\ h_{a\alpha}\, X^a Y^\alpha = 0 \end{cases}$$

Classification

- invariances $\to$ topologically equivalent manifolds, redundancy
  - permutation of lines and columns
  - identities between subspaces
- but:
  - constraints $\Rightarrow$ bound on matrix size
  - often $\exists$ "favorable" configuration (simplest description)

Topology

Why topology?

- no metric known for compact CY (cannot perform KK reduction explicitly) [but see: 2012.04656, Anderson-Gerdes-Gray-Krippendorf-Raghuram-Ruehle]
- topological info. $\rightarrow$ properties of 4d low-energy effective action (number of fields, representations, gauge symmetry. . .)

Topology

Why topology?

- no metric known for compact CY (cannot perform KK reduction explicitly) [but see: 2012.04656, Anderson-Gerdes-Gray-Krippendorf-Raghuram-Ruehle]
- topological info. $\to$ properties of 4d low-energy effective action (number of fields, representations, gauge symmetry...)

Topological properties

- Hodge numbers $h^{p,q}$ (number of harmonic (p,q)-forms) here: $h^{1,1}$, $h^{2,1}$
- Euler number $\chi = 2(h^{1,1} - h^{2,1})$
- Chern classes
- triple intersection numbers
- line bundle cohomologies

Topology

Why topology?

- no metric known for compact CY (cannot perform KK reduction explicitly) [but see: 2012.04656, Anderson-Gerdes-Gray-Krippendorf-Raghuram-Ruehle]
- topological info. $\to$ properties of 4d low-energy effective action (number of fields, representations, gauge symmetry. . .)

Topological properties

- Hodge numbers $h^{p,q}$ (number of harmonic (p, q)-forms) here: $h^{1,1}$, $h^{2,1}$
- Euler number $\chi = 2(h^{1,1} - h^{2,1})$
- Chern classes
- triple intersection numbers
- line bundle cohomologies

Datasets

CICY have been classified

- 7890 configurations (but $\exists$ redundancies)
- number of product spaces: 22
- $h^{1,1} \in [0, 19]$, $h^{2,1} \in [0, 101]$
- 266 combinations $(h^{1,1}, h^{2,1})$
- $a^r_\alpha \in [0, 5]$

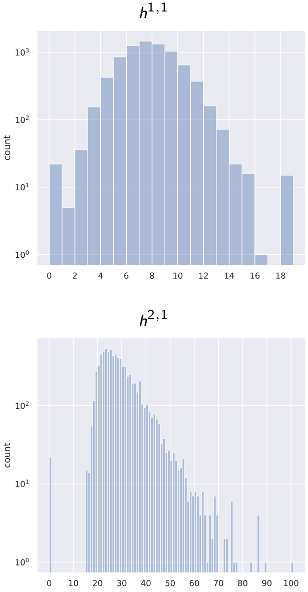
Original data [Candelas-Dale-Lutken-Schimmrigk '88; Green-Hübsch-Lutken '89]:

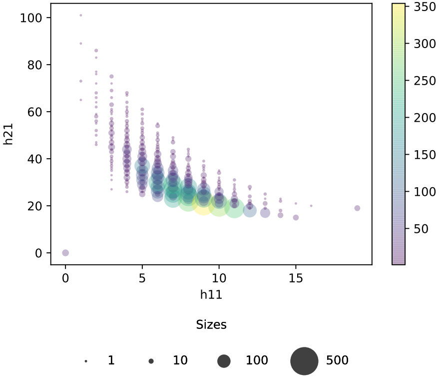
- maximal matrix size: 12×15
- number of favorable matrices: 4874

Favorable data [1708.07907, Anderson-Gao-Gray-Lee]:

- maximal matrix size: 15×18
- number of favorable matrices: 7820

Data





Goal and methodology

Philosophy

Start with the dataset, derive everything from configuration matrix using data analysis and machine learning only.

Current goal

Input: configuration matrix $\longrightarrow$ Outputs: $h^{1,1}$, $h^{2,1}$

Motivations:

1. CICY: well studied, all topological quantities known
   $\rightarrow$ use as a sandbox
2. perform complete data analysis
3. improve over [1706.02714, He; 1806.03121, Bull-He-Jejjala-Mishra]
4. $h^{2,1}$ and favorable dataset not studied before

References: [HE-Finotello, 2007.13379, 2007.15706]

Outline: 4. Data analysis

Motivations

Machine learning

Calabi–Yau 3-folds

Data analysis

Machine learning analysis

Conclusion

Feature engineering

Process of creating new features derived from the raw input data.

Some examples:

- number of projective spaces (rows), $m = \texttt{num\_cp}$
- number of equations (columns), k
- number of $\mathbb{C}P^1$
- number of $\mathbb{C}P^2$
- number of $\mathbb{C}P^n$ with $n \neq 1$
- Frobenius norm of the matrix
- list of the projective space dimensions and statistics thereof
- dimensions of ambient space cohomology $\left\{ \prod_{r=1}^{m} \binom{n_r + a_\alpha^r}{n_r} \right\}$
- K-nearest neighbour (KNN) clustering (with $K = 2, \ldots, 5$)

Feature selection

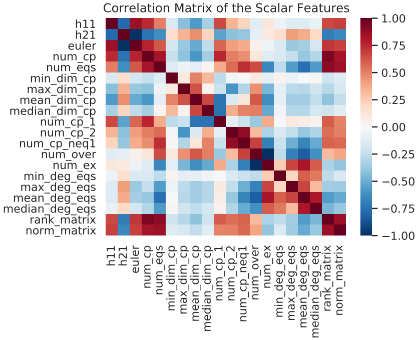
Select the most important features to draw attention of the ML algorithm to salient features in order to ease the learning.

Discovery methods:

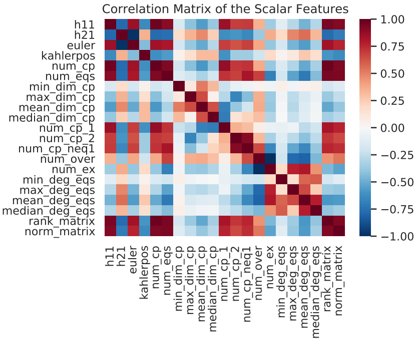
- correlation matrix
- importance from random forests
- scatter plots
- trial and error
- etc.

Correlation matrix

Original



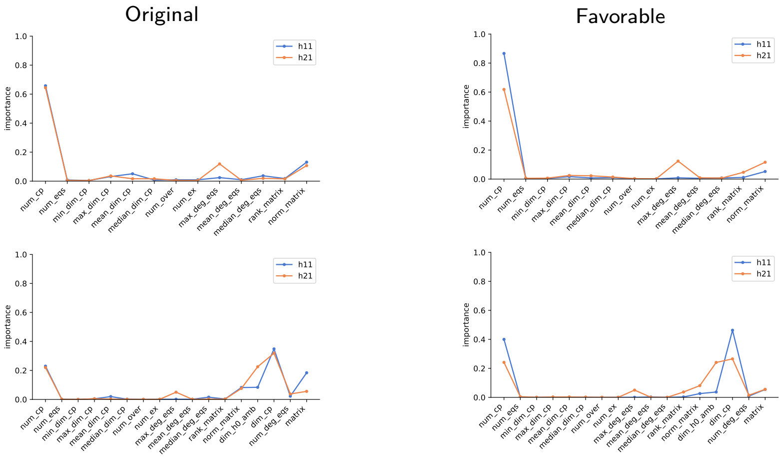
Favorable



Feature importance from random forests

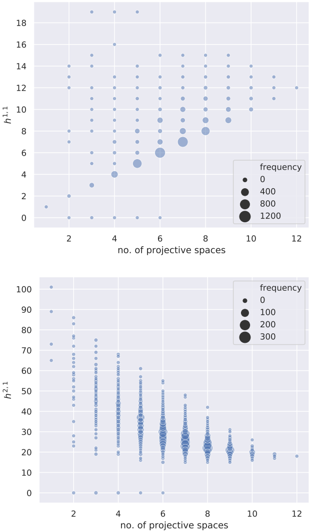
Random forest

Large number of decision trees trained on different subsets. The most relevant features appear at the top of the trees.

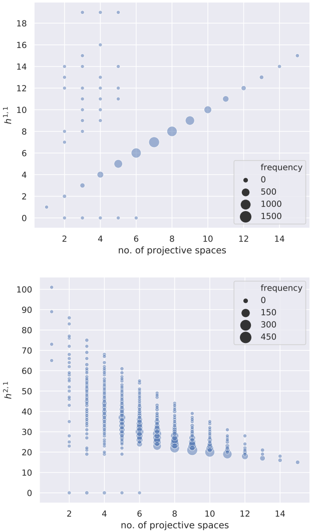


Scatter plots

Original



Favorable



Outline: 5. Machine learning analysis

Motivations

Machine learning

Calabi–Yau 3-folds

Data analysis

Machine learning analysis

Conclusion

Strategy

Questions:

- classification or regression?
- feature engineering?
- data diminution: remove outliers (39 matrices, 0.49%)?
- data augmentation: generate more inputs using invariances?
- single- or multi-tasking?

Strategy

Questions:

- classification or regression?
  - classification: assume knowledge of boundaries
    (in practice, performs less well)
  - regression: better for generalization
    different scales $\rightarrow$ normalize data $\approx$ use continuous variable
    (in practice, not needed)
- feature engineering?
  $\rightarrow$ helps only for non-neural network algorithms
- data diminution: remove outliers (39 matrices, 0.49%)?
  $\rightarrow$ remove outliers from training set
- data augmentation: generate more inputs using invariances?
  $\rightarrow$ adding row/column permutations decreases performance
- single- or multi-tasking?
  $\rightarrow$ multi-tasking slightly decreases performance

Setup

Algorithms:

- linear regression
- linear-kernel SVM
- Gaussian-kernel SVM
- random forests
- gradient boosted trees
- neural networks

Evaluation:

- train/validation/test splits: 80/10/10 and 30/10/60
- optimization using MSE
- final evaluation with accuracy after rounding

Setup

Algorithms:

- linear regression
- linear-kernel SVM
- Gaussian-kernel SVM
- random forests
- gradient boosted trees
- neural networks

Evaluation:

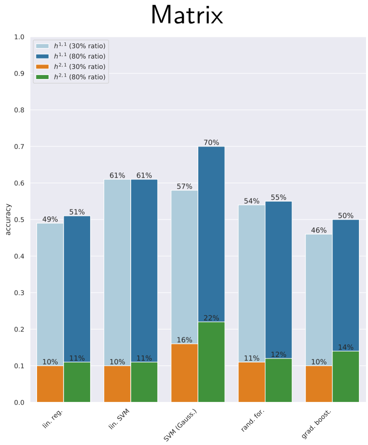
- train/validation/test splits: 80/10/10 and 30/10/60
- optimization using MSE
- final evaluation with accuracy after rounding

Preliminary observations:

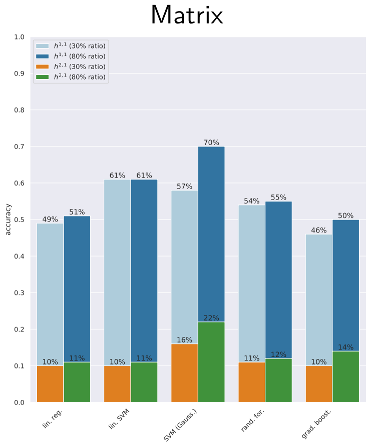
- all algo. give 99 % for $h^{1,1}$ in favorable dataset with engineered features (without engineering: 90-95 % for standard algo.)
- $h^{2,1}$ equivalently hard in both sets

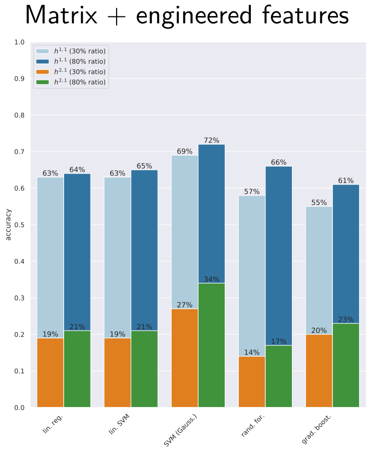
→ focus on original dataset

Results: simple algorithms



Results: simple algorithms

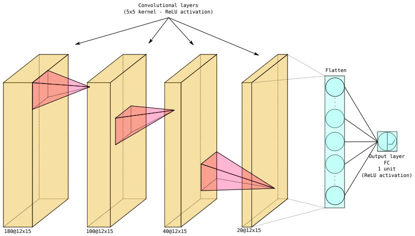




Convolutional neural network

Architecture and training:

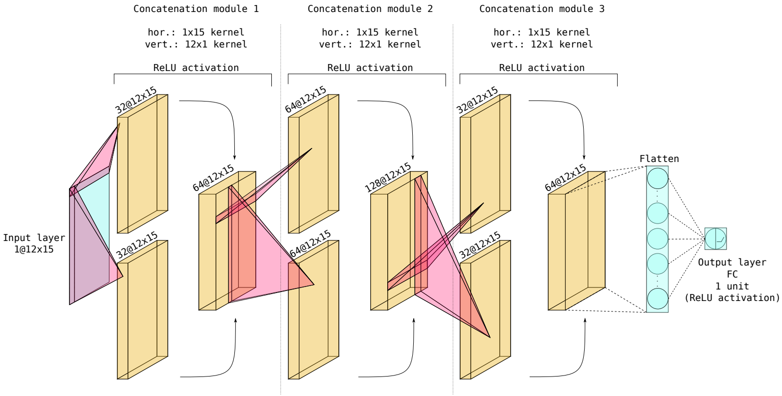
- 4 convolutional layers, kernel 5×5:
  - $h^{1,1}$: 180, 100, 40, 20 units
  - $h^{2,1}$: 250, 150, 100, 50 units
- after each layer: batch normalization, ReLU activation
- at the end: dropout $p = 0.2$, ReLU (enforces positive output)
- early stopping & learning rate decay primordial to increase accuracy beyond 90 %
- number of parameters:
  - $h^{1,1}$: 5.8×10^5
  - $h^{2,1}$: 2.1×10^6



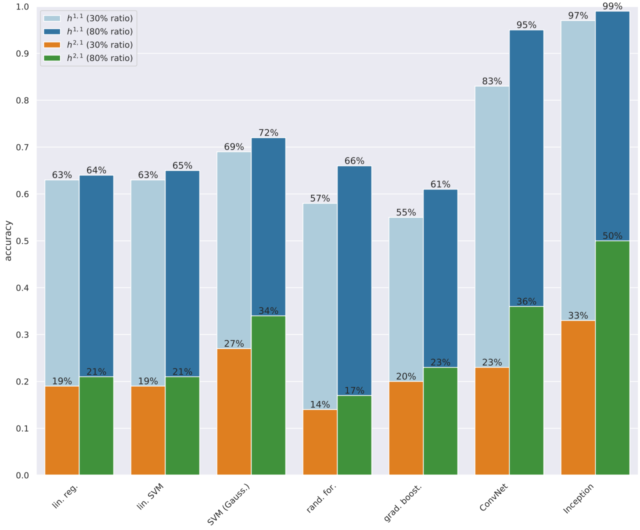
Inception neural network (1)

- designed by Google for computer vision
  $\rightarrow$ breakthrough in image classification
  [Szegedy et al., 1409.4842, 1512.00567, 1602.07261]
- sequence of inception modules
  $\rightarrow$ parallel convolutions with kernels of $\neq$ sizes
- learns different combinations of features at different scales
- 3 inception modules, kernels $(12 \times 1, 1 \times 15)$:
  - $h^{1,1}$: 32, 64, 32 units
  - $h^{2,1}$: 128, 128, 64 units
- numbers of parameters:
  - $h^{1,1}$: 2.3×10^5, $7\times$ less than [1806.03121, Bull-He-Jejjala-Mishra]
  - $h^{2,1}$: 1.1×10^6

Inception neural network (2)

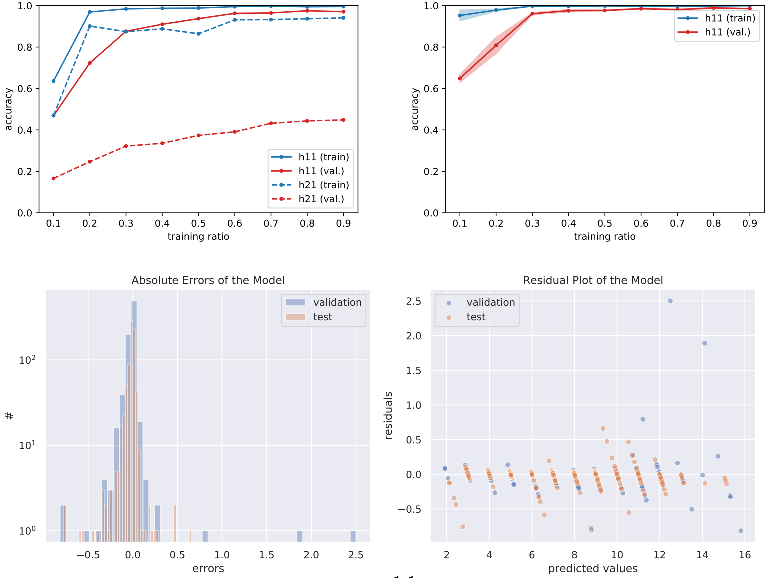


Results



h^{1,1} (30% ratio)
h^{1,1} (80% ratio)
h^{2,1} (30% ratio)
h^{2,1} (80% ratio)
accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
63%
64%
19%
21%
63%
65%
19%
21%
69%
72%
27%
34%
57%
66%
14%
17%
55%
61%
20%
23%
83%
95%
23%
36%
97%
99%
33%
50%
lin. reg.
lin. SVM
SVM (Gauss.)
rand. for.
grad. boost.
ConvNet
Inception

Learning curve and errors



$h^{1,1}$

Why do convolutional / Inception networks work?

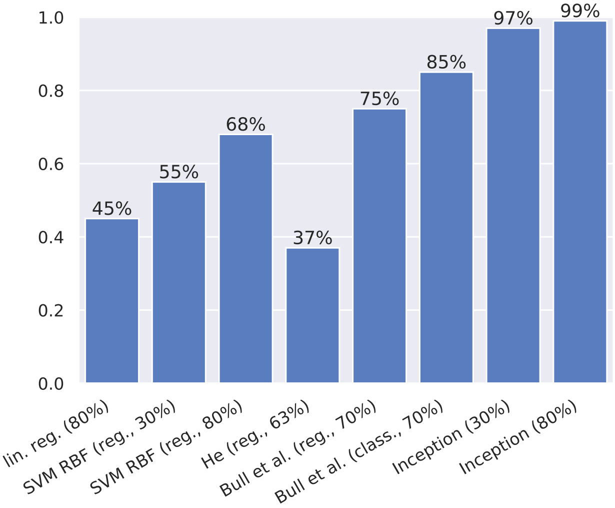
- matrix not invariant under rotation/translation, but conv. layers encodes only translation equivariance (pooling and data augmentation induces invariance under rotation and invariance) [Goodfellow-Bengio-Courville '16]
- $1d$ parallel kernels of maximal sizes: look at all $\mathbb{C}P^n$/equations for each equation/$\mathbb{C}P^n$ at the same time
- weight sharing (convolution): same operations for each $\mathbb{C}P^n$ and equation since they all enter symmetrically (expected for a math formula)

Why do convolutional / Inception networks work?

- matrix not invariant under rotation/translation, but conv. layers encodes only translation equivariance (pooling and data augmentation induces invariance under rotation and invariance) [Goodfellow-Bengio-Courville '16]
- $1d$ parallel kernels of maximal sizes: look at all $\mathbb{C}P^n$/equations for each equation/$\mathbb{C}P^n$ at the same time
- weight sharing (convolution): same operations for each $\mathbb{C}P^n$ and equation since they all enter symmetrically (expected for a math formula)

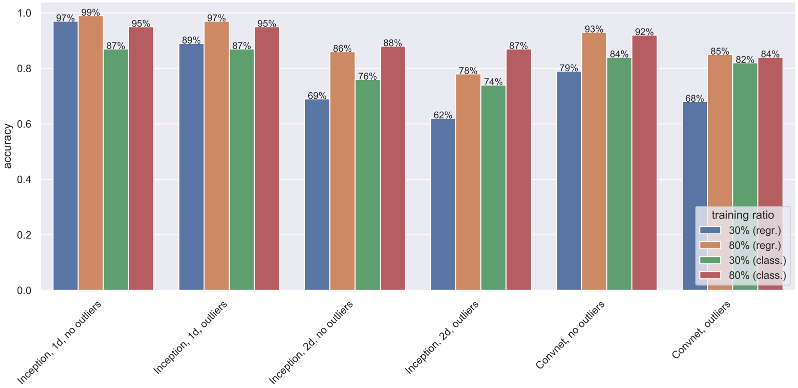
Next: focus on $h^{1,1}$

Comparing architectures



He: 1706.02714; Bull et al.: 1806.03121; percentage: training data

Ablation study



Outline: 6. Conclusion

Motivations

Machine learning

Calabi–Yau 3-folds

Data analysis

Machine learning analysis

Conclusion

Conclusion

Results:

- rigorous data analysis for the computation of Hodge numbers for CICY 3-folds
- almost perfect accuracy for predicting $h^{1,1}$
- accuracy of 50 % for $h^{2,1}$

Outlook:

- improve accuracy for $h^{2,1}$
  1. use engineered data as auxiliary inputs to the Inception network
  2. use another data representation (e.g. graph [Hübsch '92; 2003.13679, Krippendorf-Syvaeri], learned from variational autoencoder. . .)
- dissect neural network data to understand what it learns
- extension to CICY 4-folds