Machine Learning for Level Truncation in String Field Theory — Notes

Harold Erbin[*] and Riccardo Finotello[†]

Dipartimento di Fisica, Università degli Studi di Torino,
I.N.F.N. – sezione di Torino and Arnold–Regge Center,
via P. Giuria 1, I-10125 Torino, Italy

7th September 2020

Abstract

In the framework of open String Field Theory (SFT), we consider solutions of several observables in different models and at different finite mass level truncations. We then use Machine Learning (ML) to predict the value of the observables at infinite mass level truncation, preferably in a model independent fashion (that is we do not take as input the model which generated the data). The analysis can be found in the different branches of this GitHub repository.

[*] e-mail: erbin@to.infn.it

[†] e-mail: riccardo.finotello@to.infn.it

Contents

1 Preliminary Definitions and Operations

The current analysis includes data of lumps solutions and the Wess–Zumino–Witten (WZW) SU(2) model. Solutions of the models are in general characterised by different variables which however must be uniformly modified to be comparable when merging different datasets. The following study focuses on the Exploratory Data Analysis (EDA) of the various datasets to be able to extract meaningful features for the ML analysis.

The analysis is mainly written in Python using *Jupyter* notebooks and the *Scipy* ecosystem of tools [1]. We use *Scikit-learn* [2] and *Scikit-optimize* [3] for the EDA and the shallow ML analysis on the lumps dataset (the first for training most algorithms and the second for hyperparameter optimisation). We also employ decision tree based algorithms using Microsoft's *LightGBM* library [4]. Plots and figures have been produced using *Matplotlib* [5]. Since we use algorithms based on decision trees, we also perform an in-depth analysis of the variable ranking using the *SHAP* tool [6] to get better insight of the outcome of the algorithms. Finally the deep learning models are built using Google's *Tensorflow* [7] and its high level API *Keras*.

In the GitHub repository the analysis is organised as follows:

- the lumps branch contains the analysis of the lumps dataset (see the README file for more information),
- the wzw branch holds the analysis of the WZW model,
- the aggregate branch contains the model independent analysis,
- the metrics_eval branch contains a preliminary study on the best metric to use for the analysis,
- the report branch contains this report,
- the old branch holds the outdated version of the analysis.

In the following sections we first study the lumps solutions (for them we will also provide a separate ML analysis as a preliminary exploration of the model independent data) and the WZW model. We then focus on aggregating the data and producing a model independent dataset which can in turn be separately studied using ML.

2 Lumps Solutions

2.1 Description

The dataset is composed of 46 different solutions at different radii. Each of them is then composed of lists of several observables of different lengths (varying from 15 to 20 entries each, as shown in Figure 2.1).

Every observable is characterised by its conformal weight (the `weight` column in the dataset), its kind of "oscillations" (the `type` variable, categorical and in lexicographic order), its initialisation point (`init`) and the truncation levels (from level 2 to level 18). The `exp` column contains the label to be predicted from the other variables at finite mass level truncation. It represents the extrapolation at ∞ mass level truncation and takes integer values in the range $[-1, 1]$. In general the entries are real numbers.

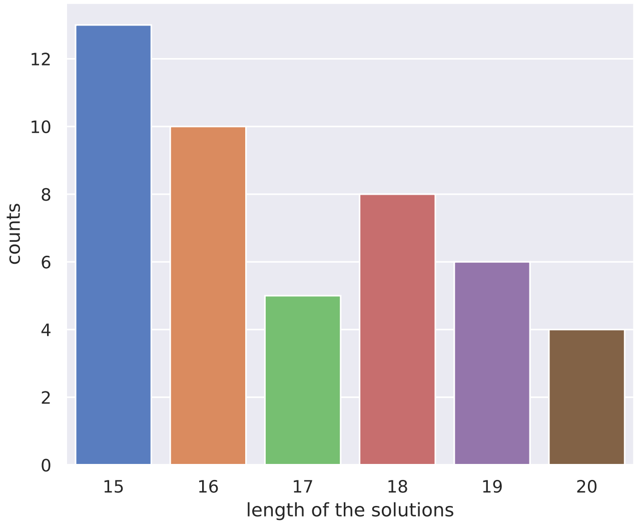


Figure 2.1: Length of the vector-like solutions in the untidied dataset.

2.2 Preparation

The entries of the dataset are vector-like objects which need to be tidied before using them for the analysis. We first artificially insert a new variable (called `solutions`) to label the 46 different rows of the dataset. We then flatten the entire dataset over the values in its rows and create a new table containing only numeric entries: the newly formed dataset has 778 rows and 22 columns.

Before proceeding to the analysis we take into account the possibility of relations between the entries and remove the duplicates from the dataset. In total we remove 46 duplicates corresponding to 6 % circa of the total dataset. After this operation the dataset contains 732 rows and it is ready for the EDA.

2.3 Exploratory Data Analysis

In the EDA section we study the properties of the tidy dataset. We focus on outlier detection and correlations between the variables. We also anticipate that in general we will not use the variables `init` and `solutions` since results should not depend on the particular solution or initial value.

2.3.1 Outliers Detection

The dataset presents several variables having a very large range of variability and may thus contain a large number of outliers. As a first step we define the *interquartile* range for each variable and compute the fraction of outlying samples.[1] In general the truncation levels show a high number of outliers (the last truncation level contains 27 % outlying values, while others have fractions of 20 % outliers in average).

[1] To compute the interquartile range we first compute the 25th and 75th percentiles (i.e. the first and third quartiles) of the distribution of each variable. The range is then defined as 1.5 times the distance between the two quartiles computed from the lower and the upper bounds of the distribution of the values.

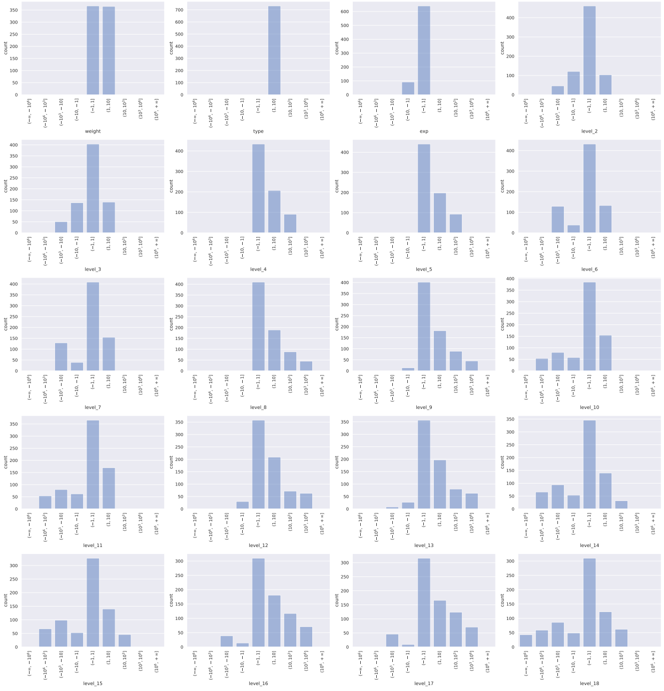


Figure 2.2: Distribution of the values of each variable categorised by order of magnitude.

In Figure 2.2 we show the distribution of the values: in general they are spread across different orders of magnitude and show a tendency to alternate between positive and negative values. This in turn may cause issues when training ML algorithms since the very large variance may result in "too precise" determination of the coefficients, ultimately leading to a poor generalisation ability.

However the large number of outliers is mainly due to data corresponding to higher values of `weight`. In fact we can separately analyse the dataset for `weight` < 1.5 and `weight` ≥ 1.5. In the first case the values of the variables are in general contained in smaller intervals and present a smaller value of the variance as pictorially shown in Figure 2.3, where we can see that the variables are in general $\mathcal{O}(1)$ in values. In the latter the distribution is less uniform and encompasses multiple orders of magnitude (differently from the `weight` < 1.5 case showing a box plot similar to Figure 2.3 does not help in visualising the situation).

This might be connected to particular relations between the observables. The EDA can highlight some of them (see Table 2.1). In fact higher weights present only a specific type of oscillation (`type` is strictly 4), while lower values of `weight` show that the variables include different kind of oscillations. However in this case `type` $= 2$ strictly implies a vanishing weight.

Finally in Figure 2.4 we show the correlation matrices of the variables. As expected, the truncation levels are strongly correlated among themselves and with the `weight` variable. Though milder, there is also a good correlation of the variables with the labels we intend to predict (`exp`), while the `type` variable seems to be completely uncorrelated (this may however be due to the fact of being categorical: a linear regression might be more suitable to do statistical inference on it). Another notable remark is the different behaviour of the correlations between consecutive layers: the large anti-correlations seem to be entirely due to the higher weights, while `weight` < 1.5 seems to imply a correlation only between adjacent couples of levels (e.g. 2-3, 4-5, 6-7, etc.) and their replicas at distance 3 (e.g. 4-5 with 8-9, 12-13 and 16-17, 6-7 with 10-11, 14-

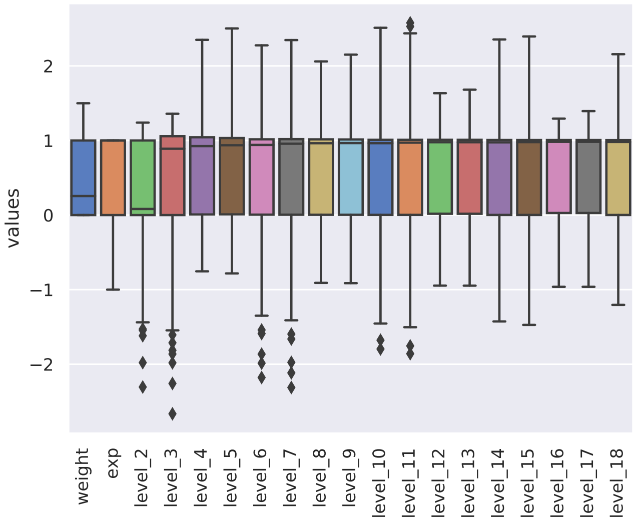
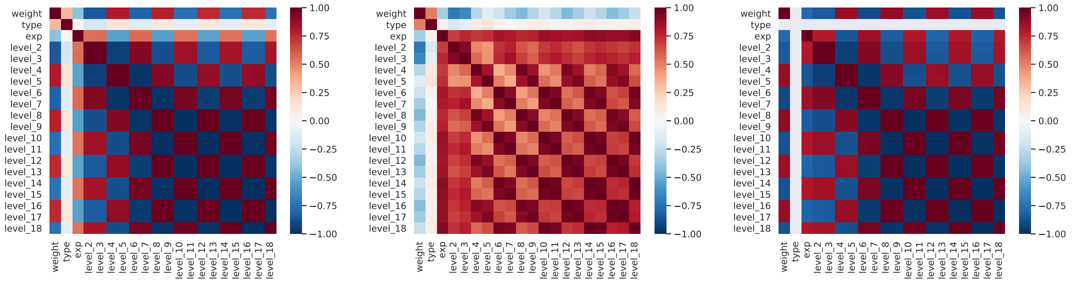


Figure 2.3: Boxplot of the values for `weight` < 1.5. The coloured boxes represent data between the first and third quartiles (the horizontal line is the median), while the "whiskers" show the interquartile range. Isolated points are the outliers.

		weight	
weight	**type**	μ	σ
≥ 1.5	2	—	—
	4	4	2
< 1.5	2	0	0
	4	0.6	0.5

Table 2.1: Relations between the `type` and `weight` variables (μ is its average value and σ its standard deviation).

15 and 18, etc.). This behaviour may point to several possible interpretations. In particular we see that data is oscillating: going towards larger truncation levels data seem to alternate a behaviour similar to a periodic function (sine or cosine in the particular case) with maxima and minima alternating at constant intervals. This may become clear when considering the definition of the correlation factor of two variables X_1 and X_2 given as the ratio between the covariance $\sigma(X_1, X_2) = (X_1 - \bar{X}_1) \cdot (X_2 - \bar{X}_2)$, where $\bar{X}_1$ and $\bar{X}_2$ are the mean of the respective variables, and the product of the separate standard deviations $\sigma(X_1)\sigma(X_2)$). However we also see that this behaviour is particularly present when considering the full dataset (or just **`weight`** ≥ 1.5) while if we introduce a cut at **`weight`** < 1.5 the correlation is definitely less marked, though present.



(a) Full dataset (b) **`weight`** < 1.5. (c) **`weight`** ≥ 1.5.

Figure 2.4: Correlation matrices of the variables.

2.3.2 Principal Components Analysis

We perform the Principal Components Analysis (PCA) of the truncation levels to study their properties and their distribution: it encodes the linear variance of the samples by projecting the distribution onto a new system of coordinates trying to maximise such variance in each new axis. This approach may be interesting both as exploratory analysis and for further directions: we may want to generalise the algorithm to an arbitrary number of truncation levels (and we may want to skip some values). The PCA would then give a solution to always have input of the same size while keeping as much information as possible.

We first study all principal components using the Singular Value Decomposition (SVD) of the matrix holding samples over the rows and the truncation levels over the columns (it is a rectangular matrix and as such cannot be strictly diagonalised to perform spectral analysis). In Figure 2.5 we show the variance explained by each principal component of the matrix (i.e. the fraction of variance of the total set retained by considering only the selected component).

The analysis was performed splitting the values of the truncation levels over ranges of the **`weight`** variable. The two folds have then been separately scaled: values of **`weight`** < 1.5 have been standardised (i.e. centered and scaled by their standard deviation), while the remaining values have been robustly scaled (i.e. centered and scaled by their interquartile) against the outliers present. The result shows that for lower weights the first two principal components account for more than 90 % of the total variance of the values, while for larger weights almost 100 % of the variance is explained by the first component (in fact the 99 % of the variance can be reached using 4 components for **`weight`** < 1.5, while the first component for **`weight`** ≥ 1.5 already contains 99.8 % of the total variance). This is a reflection the distribution of the variables in the dataset: larger weights contain a very large amount of outliers and have larger variance with respect to lower weights. It is then enough to project the values of the truncation levels

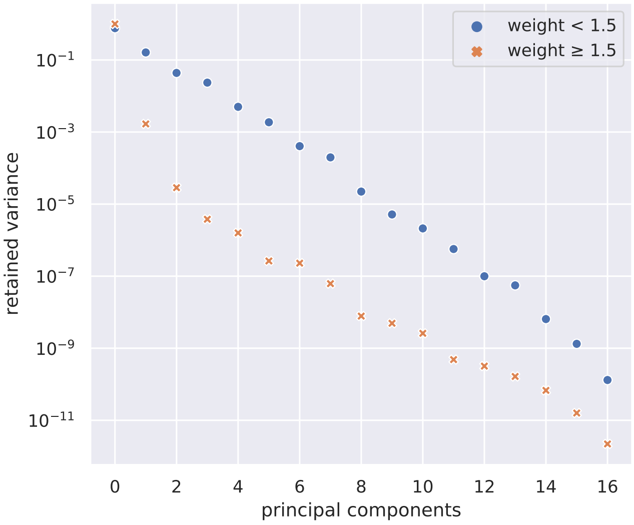


Figure 2.5: Variance explained by the principal components of the truncation levels (log scale).

over the line which contains most of the deviation to reproduce a meaningful distribution.

2.3.3 K-Means Clustering

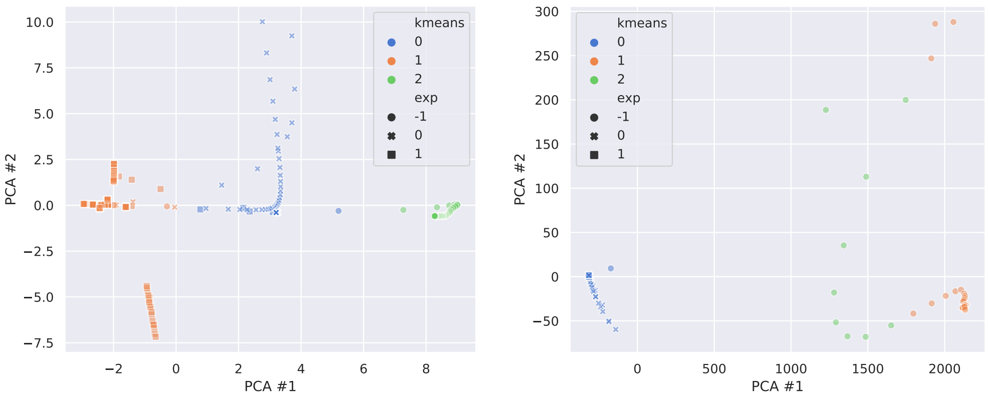
Finally we perform an unsupervised clustering analysis of the truncation levels. The main idea is to infer a structure in the data which may be able to "automatically" reproduce the labels (i.e. the `exp` column) without regression. In other words we study the distribution of the truncation levels and fit it in 3 clusters representing the 3 integer values of the labels.[2] In the ideal scenario there should be a 1:1 relation between the labels of the cluster centroids and the labels in the `exp` variable. Unfortunately the cluster analysis of the truncation levels over the entire dataset highlighted no particular structure in the data and turned out inconclusive.

We then performed the same analysis splitting the dataset in different ranges of the `weight` variable, standardising the data for `weight` < 1.5 and robustly scaling (i.e. scaling according the interquartile range) samples for which `weight` ≥ 1.5. In Figure 2.6 we used the principal components of the truncation levels to plot the distribution of the clusters and the `exp` labels.

	K-Means	
`exp`	μ	σ
-1	1.9	0.4
0	0.1	0.3
1	0.98	0.14

Table 2.2: Average cluster label for `weight` < 1.5 (μ is the average value and σ its standard deviation).

[2] This analysis is strongly related to the finite number of unique values of the label in the dataset (for instance, the number of clusters has been chosen to match the number of unique values of `exp`). It might not be possible to repeat or generalise this procedure to other datasets.



(a) `weight` < 1.5.

(b) `weight` ≥ 1.5.

Figure 2.6: K-Means clusters and `exp` labels plotted using the principal components for visualisation purposes.

As we can see, recognising a structure is challenging when `weight` ≥ 1.5 and in fact different `exp` (or "extrapolation") labels generally belong to different clusters. However the case `weight` < 1.5 seems to be more accurate and shows that in general we can recognise a structure in the data. This is summarised in Table 2.2 where we show that there is a distinguishable relation between the extrapolation column and the average cluster label in the group when `weight` < 1.5, while in general there is a superposition of labels when `weight` ≥ 1.5 (specifically it seems that there are only 2 recognisable clusters since both `exp` $= 0$ and `exp` $= 1$ share identically the same cluster label).

2.4 Statistical Inference

After the EDA we proceed with the study of statistical inference using linear models on the data. The purpose of the analysis is to better understand the contribution of each variable to the predictions and study the relations among different features. In the following we will be mainly interested in highlighting the properties of the dataset, while we only marginally mention the predictive ability of the linear model. We use a simple linear regression without regularisation (using the class `LinearRegression` in `Scikit-learn`).

In order to avoid biasing the results, we split the dataset into a training set made of 80 % of the samples and a development set made of the remaining entries. The division has been made using the unique values of the `solutions` column: we first select what solutions to insert in the training and development folds and then assign the corresponding samples to the different sets. This has the results of keeping similar distributions inside the same set and facilitate the training.[3] We also remove the case `solutions` $= 0$ (i.e. the first entry of the original untidied dataset) because its structure might spoil the generalisation properties since its values are too correlated with the output.

After training the linear model on the training fold, we analyse the outcome on the development set. With 118 degrees of freedom (d.o.f.) we reach a mean squared error (MSE) of 0.16 with a 95 % confidence interval (95 % C.I.) $[0.10, 0.23]$ and a *coefficient of determination* r^2 of

[3] This is again deeply connected with the particular dataset we are using: it might not be possible to generalise such procedure, even though in this case it may help improving the results.

0.68.[4] The interesting part of the analysis is however the *ANalysis Of VAriance* (ANOVA) of the coefficients summarised in Table 2.3 (we do not report the 95 % C.I. for brevity, since in this case the p-value holds more than enough information). As we can see the large variance associated with the values of each variable led to a very precise determination of the coefficients (as feared, even too precise). As a consequence, the p-value associated to the observation of more extreme values than those computed here is vanishing for almost all coefficients. The `type` variable however has a different behaviour: according to its associated p-value we might consider removing it from the input features. However such feature, even though irrelevant for the fit, plays an important role when distinguishing higher and lower weights. We will therefore keep it in the input features to help the algorithms in training (especially the decision trees).[5]

2.5 Model Dependent Machine Learning Analysis

2.5.1 Validation and Evaluation Strategy

As for the previous section, in the ML analysis we use the tidy dataset without the samples corresponding to `solutions` = 0 to try and retain as much generalisation capability as possible. We also keep samples corresponding to the same values of `solutions` in the same set to balance the distribution of the samples as before. Since the number of samples is small, we will use a single validation set to evaluate the algorithms and perform the optimisation of the hyperparameters. However, before choosing the validation strategy, we first split the dataset in two folds: the first holds 90 % of the samples and will be used for training and validation, while the remaining 10 % will be used for testing.

We then choose the size of the validation set to be around 10 % of the total training set as it is typical for datasets of this dimension where most of the samples should be assigned to training. Another motivation for the choice is to try to minimise the error (MSE) made when fitting a linear regression on decreasing size of the set effectively used for training, after removing the validation split: this may clearly not work for anything different from a linear model, but it is however a good starting point. In Figure 2.7 we show training and validation MSE for different sizes of the validation set (with respect to the total training set). In order to keep a reasonable amount of samples in the training split and to contain the validation MSE as much as possible, we finally take the already mentioned 10 % of the unique `solutions` in the validation set.

In Table 2.4 we give a schematic summary of the final choice for training, validation and test sets after assigning the samples corresponding to the chosen `solutions` in each fold.

[4] The coefficient of determination is a measure of the fraction of the variance of the data explained by the model (sometimes it is referred to as *fraction of variance explained*). Its definition is however different from the coefficient of correlation ρ also used in statistics. In fact, let

$$S_{res}(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \qquad S_{tot}(y) = \sum_{i=1}^{N} (y_i - \bar{y})^2,$$

where N is the number of samples, y are the "experimental" data, $\bar{y}$ is the sample average of them, and $\hat{y}$ are the predicted data. Then the coefficient of determination is defined as:

$$r^2 = 1 - \frac{S_{res}(y, \hat{y})}{S_{tot}(y)} = 1 - \mathrm{FVU},$$

where FVU stands for *Fraction of Variance Unexplained*, that is the amount of variance of the data which the model cannot account for. A perfect predictor would have $r^2 = 1$ (i.e. it is capable of reproducing the full variance of the data), while a poor model can have negative values.

[5] We also repeated the same statistical inference removing the `type` variable and, as expected, the ANOVA did not show any difference: all p-values are vanishing in any case. However the predictions suffered from the missing variable.

	coefficient	**standard error**	**Student's t**	**p-value** ($t_{obs} \geq \lvert t \rvert$)
`weight`	0.109	0.015	7	0.0
`type`	0.03	0.05	0.6	0.5
`level 2`	-0.067	0.007	-9	0.0
`level 3`	0.135	0.007	20	0.0
`level 4`	-0.2955	0.0014	-2×10^2	0.0
`level 5`	0.4161	0.0014	3×10^3	0.0
`level 6`	-2904×10^{-4}	3×10^{-4}	-9×10^3	0.0
`level 7`	4527×10^{-4}	3×10^{-4}	1.5×10^3	0.0
`level 8`	-41345×10^{-5}	6×10^{-5}	-7×10^3	0.0
`level 9`	57675×10^{-5}	6×10^{-5}	1.0×10^3	0.0
`level 10`	-31909×10^{-5}	1.3×10^{-5}	-2×10^4	0.0
`level 11`	43550×10^{-5}	1.3×10^{-5}	3×10^4	0.0
`level 12`	-191313×10^{-6}	3×10^{-6}	-5×10^4	0.0
`level 13`	249714×10^{-6}	3×10^{-6}	6×10^4	0.0
`level 14`	-602220×10^{-7}	8×10^{-7}	-3×10^4	0.0
`level 15`	808300×10^{-7}	8×10^{-7}	4×10^4	0.0
`level 16`	59010×10^{-7}	2×10^{-7}	-5×10^3	0.0
`level 17`	103750×10^{-7}	2×10^{-7}	8×10^3	0.0
`level 18`	43700×10^{-8}	7×10^{-8}	4×10^2	0.0

Table 2.3: Analysis of the coefficients of the linear model.

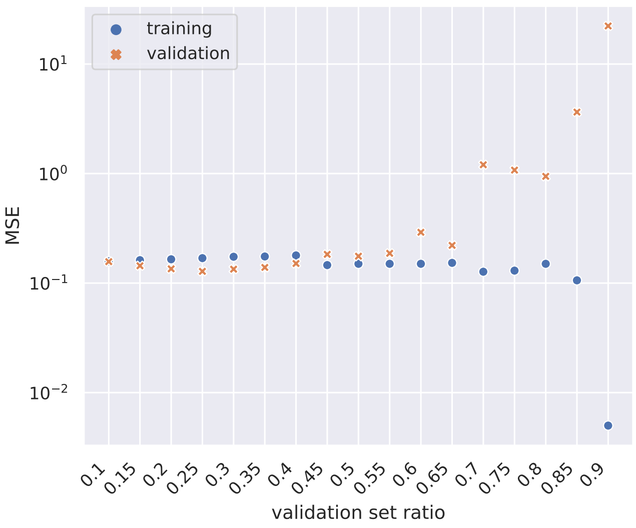


Figure 2.7: MSE in training and validation for different validation ratios.

	total dataset	
	samples	*fraction*
training	584	80%
validation	68	9%
test	80	11%

Table 2.4: Summary of the train/validation/test splits.

2.5.2 Training

For the ML analysis we first scale the truncation levels using a robust scaler (the `RobustScaler` in `Scikit-learn`). We then include also the variables `weight` and `type` into the input features. In Table 2.5 we summarise the predictions of different algorithms after optimisation (we use Bayes optimisation techniques implemented in `Scikit-optimize` to look for the best hyperparameters): *LR* refers to linear regression (with ℓ_2 regularisation), *l-SVR* is the linear implementation of support vector machines for regression, *r-SVR* uses a *kernel trick* (with a Gaussian function, or *radial* function) for support vectors, *RF* are random forests of decision trees, *GBDT* are gradient boosted decision trees, and finally *ANN* refers to artificial neural networks.[6]

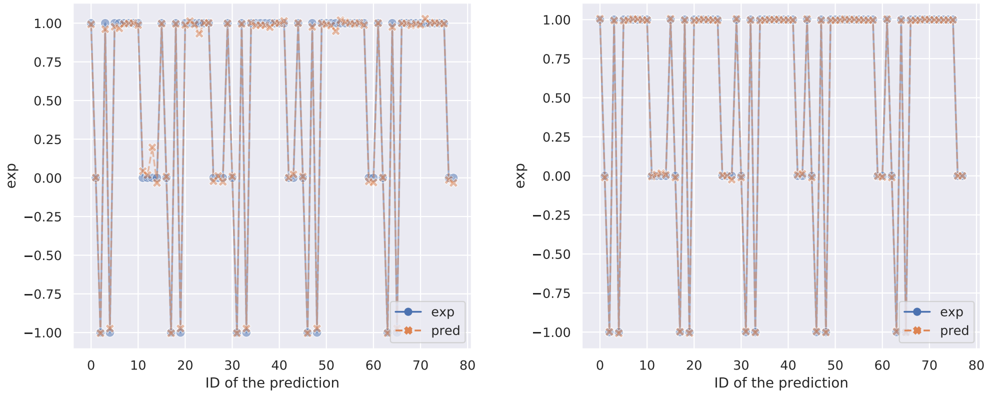
	MSE	**95% C.I.**	**MAE**	r^2
LR (ℓ_2 *reg.*)	0.16	(0.07, 0.25)	0.3	0.68
l-SVR	2.4	(-0.2, 5.1)	0.5	-3.77
r-SVR	0.030	(0.005, 0.046)	0.08	0.95
RF	0.011	(-0.003, 0.025)	0.04	0.98
GBDT	0.00085	(-0.00015, 0.00185)	0.016	1.00
ANN	0.00004	(0.0002, 0.0005)	0.005	1.00

Table 2.5: Summary of the predictions on the test set. The number of d.o.f. has been taken to be constant for alla algorithms ($80 - 21 = 59$) as in the linear model for a direct comparison.

As we can see both *GBDT* and *ANN* are able to deliver consistency and precision in the predictions on the test set. As shown in Figure 2.8 both algorithms perform extremely well on the test set as also the histograms of the residuals in Figure 2.9 are able to show.

Given the results of both algorithms we may therefore be interested in seeing the generalisation abilities and the versatility shown by the ANNs on other datasets. While they both perform well, the ANN might in fact behave better when changing the underlying dataset. As a matter of fact, though extremely good when the training and real world sets have the same distributions, decision trees suffer a lot under small changes in the data: their good performance may therefore be due to the particular range of the label `exp`. For generalisation purposes we will therefore focus on the *ANN*.

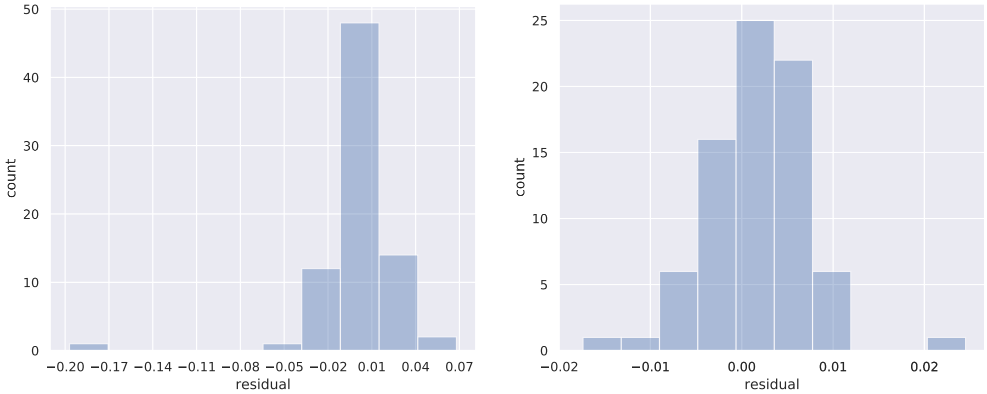
[6] This model is made of 4 hidden layers with 30, 20, 20 and 10 units each and ReLU activation. Batch normalisation (with momentum 0.99) follows each layer. Dropout layers (with rate 0.01) are used only after the first two layers. The output is made of a single unit without activation function. We use the MSE as loss function. Gradient descent is computed using the *Adam* optimiser with default parameters and a mini batch size of 32. Training is early stopped after 5000 epochs without loss improvement on the validation set and the learning rate (initially set at 0.01) is reduced by a factor 0.3 after 2500 epochs without loss improvement. While it is in general important to keep track of the models used, the architecture of the aggregate analysis is substantially different, thus making this model not particularly useful.



(a) *GBDT* predictions and ground truth.

(b) *ANN* predictions and ground truth.

Figure 2.8: Predictions and true values for the best algorithms.



(a) *GBDT* residuals.

(b) *ANN* residual.

Figure 2.9: Histogram of the residuals.

2.5.3 Analysis of the Features

After training the algorithms, we focus on studying the outcome of the decision trees (namely the *RF* and the *GBDT*). We are interested in better understanding the contributions of each variable to the results through the study of the variable ranking (i.e. the feature importances) and the Shapley values. In Table 2.6 we show the choices of the hyperparameters used for training the decision trees algorithms and which will influence to following analysis. As we could have imagined, *GBDT*s prefer a larger number of boosting rounds made of small trees, while *RF* prefer a smaller number of fully grown trees. The relative weight of each sample in the tree changes in both implementations of the decision trees and is definitely more relevant when the trees are smaller (i.e. in the *GBDT*). Both ℓ_1 (`reg_alpha`) and ℓ_2 (`reg_lambda`) regularisation have been used to produce the predictions.

hyperparameters	**RF**	**GBDT**
`num_leaves`	70	10
`max_depth`	500	25
`learning_rate`	—	0.1
`n_estimators`	50	1590
`subsample`	0.99	0.66
`colsample_bytree`	1.00	1.00
`min_child_weight`	10^{-6}	10^{-3}
`reg_alpha`	0.1	1.0
`reg_lambda`	0.12	1.0

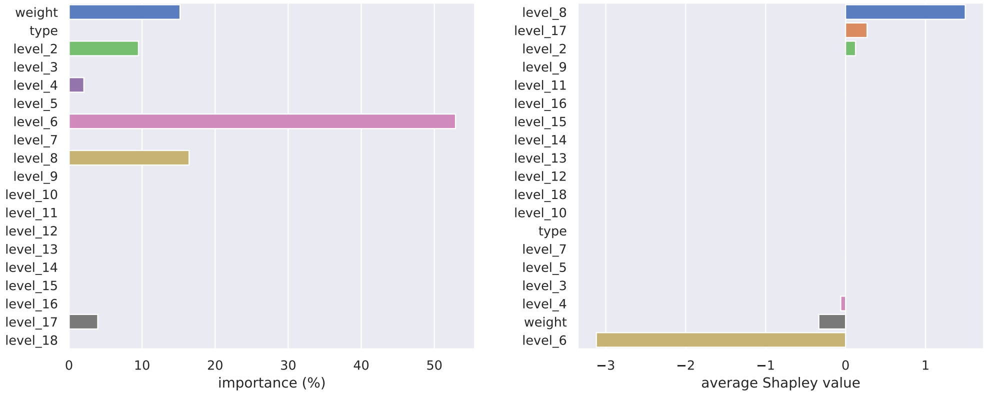
Table 2.6: Hyperparameters used by *RF* and *GBDT*.

The importance of the features is a key property of the decision trees and visually summarises the impact that each feature has on the final prediction: variables with higher importance can be found in the first branches of the trees because they are responsible for the main choices of the algorithm, while more negligible features provide the necessary refinement. The Shapley values are somewhat related and derive from a game theoretic approach to the decision trees: these values encode how a certain feature is influencing the final result, whether by dragging its value with respect to average of the predictions or by pushing it beyond it. In other words, feature importances show how much (as a percentage of the final prediction) a variable is relevant for the algorithm and the Shapley values show if there are variables which tend to drive the results towards a particular value.

In Figure 2.10 we show the variable ranking and the Shapley values for the *RF* algorithm. The first plot shows that RF rely on a particular truncation level (`level_6` to be specific) to produce the predictions, while other features contribute in a less relevant manner. The average Shapley values (second plot) show that most features give a comparable contribution to the prediction, exception made for the mass truncation at `level_6` which seems to drive the final result in a more direct way (according to Figure 2.2 they are among the highly unbalanced values, which may be a reason for such behaviour).

In Figure 2.11 we finally show the same plot for the *GBDT*s. Differently from the *RF*, *GBDT*s seem to rely almost entirely on the 6th truncation level and `weight` discrimination of the samples. In both cases the algorithms seem to ignore almost completely the categorical variable `type` (possibly it could have been removed but the corresponding Shapley value shows that its relevance is so marginal that the result would not have changed).

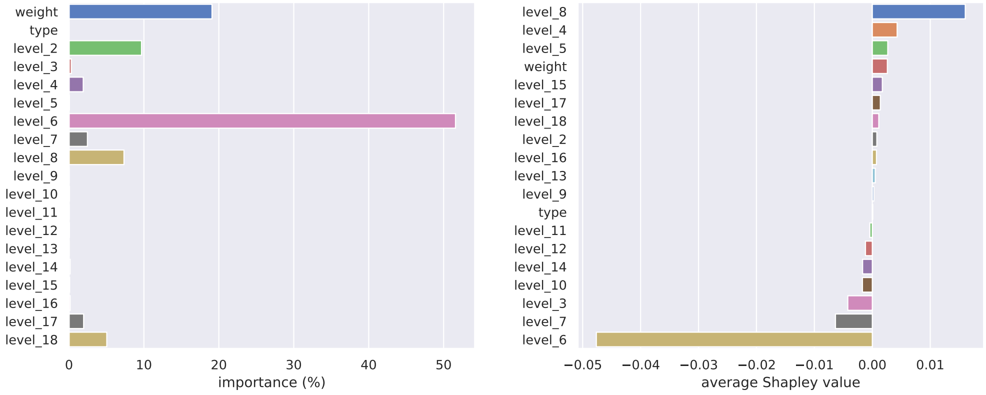
We notice that the scale of the Shapley values is very different between *RF* and *GBDT*:



(a) Variable ranking.

(b) Shapley values.

Figure 2.10: Feature importances and Shapley values of the *RF*.



(a) Variable ranking.

(b) Shapley values.

Figure 2.11: Feature importances and Shapley values of the *GBDT*s.

this is due to the fact that in this case the boosting procedure may be more robust against the large variability of the dataset, showing that in the *GBDT* case the trees are more balanced. Ultimately this is also a consequence of the different order of magnitude of the MSE associated to *RF* and *GBDT*, the latter being almost an order of magnitude more precise than the former.

We also performed this type of analysis on *GBDT* successively removing different subsamples of the features in the attempt to better understand the presence of one feature with a very high importance with respect to all others. Results showed that whatever the subsample of features, the algorithm always chooses the 3rd to 5th level truncation level as most important: after guessing the behaviour based on the first few truncation levels, the algorithm finally predicts the label using one particular feature (`level_6` in the case shown in the figures) and finally refines it using the last truncations levels.

2.5.4 Double Lumps

As a test of the generalisation ability of the algorithms, we perform predictions on double lumps solutions using the algorithms trained in the previous section (i.e. we do not perform again the optimisation and training procedure). The dataset used is made of 20 entries with the same columns as the previous solutions.

Before making predictions, we scale the truncation levels using the same robust scaler we used in the previous section. The predicted labels are then directly comparable with the real values. In Table 2.7 we show the complete list of the predictions made on the double lumps.[7] We kept `weight` and `type` to label each solution (we dropped the truncation levels for brevity). Finally we pictorially show the predictions in Figure 2.12: as imagined, decision trees kept the prediction interval $[-1, 1]$ as the original dataset on which they were trained, while the *ANN* extrapolated more.

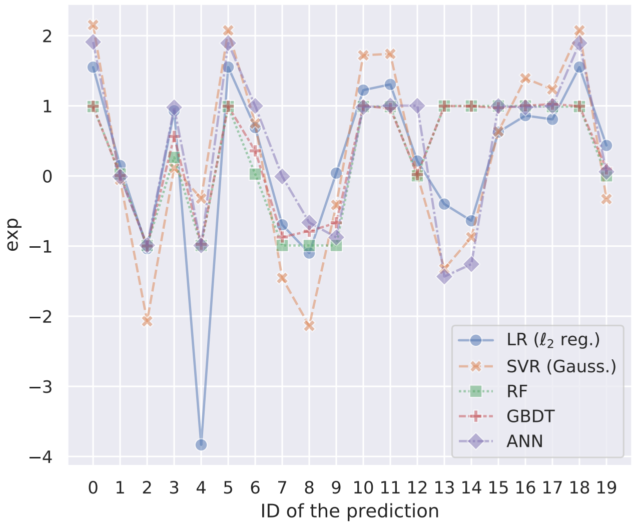


Figure 2.12: Double lumps predictions.

We finally compare the results of the predictions of the double lumps with the extrapolated labels. In order to keep only the most reliable results, we limit the predictions to `weight` < 1.5.

[7] Since the performance has been in general very poor, we removed the predictions using *l-SVR* as model.

weight	type	exp (LR)	exp (r-SVR)	exp (RF)	exp (GBDT)	exp (ANN)
0.000000	2	1.551467	2.153519	0.989096	0.994209	1.910616
0.000000	2	0.148665	-0.054234	0.026711	0.008819	-0.010774
1.000000	4	-1.036330	-2.071016	-0.991434	-0.998615	-0.987881
4.000000	4	0.930320	0.114937	0.264700	0.563773	0.978944
9.000000	4	-3.834771	-0.322144	-0.991861	-0.974030	-0.990880
0.000000	4	1.552892	2.075871	0.989096	0.995166	1.894776
0.027778	4	0.690183	0.749930	0.026711	0.358236	0.998884
0.111111	4	-0.696637	-1.455321	-0.991434	-0.869838	-0.007325
0.250000	4	-1.099292	-2.137033	-0.991434	-0.787782	-0.664277
0.444444	4	0.040714	-0.412954	-0.991434	-0.667371	-0.874694
0.694444	4	1.223861	1.721034	0.989096	1.000537	0.981812
1.000000	4	1.305418	1.741227	0.989096	0.962043	1.004753
1.361111	4	0.213720	0.014285	0.002803	0.019858	1.000996
1.777778	4	-0.400493	-1.323758	0.997782	0.996645	-1.434201
2.250000	4	-0.639370	-0.874269	0.997782	0.995911	-1.257340
2.777778	4	0.625367	0.635726	0.997782	0.971987	0.997438
3.361111	4	0.860903	1.392883	0.997782	1.002204	0.979112
4.000000	4	0.807342	1.234534	0.989096	1.023945	0.998200
0.000000	4	1.552892	2.075871	0.989096	0.995166	1.894776
2.250000	4	0.434775	-0.329326	0.002803	0.101149	0.059882

Table 2.7: Predictions on the double lumps set.

In Table 2.8 we summarise the results which showed a good result for the *r-SVR* algorithm with $r^2 = 0.96$.

	MSE	MAE	r^2
LR	0.3	0.5	0.85
r-SVR	0.10	0.3	0.96
RF	0.6	0.7	0.72
GBDT	0.6	0.7	0.72
ANN	0.4	0.5	0.81

Table 2.8: Metrics computed on the double lumps with `weight` < 1.5.

2.6 Fourier Analysis and Signal Processing

2.6.1 Tidying the Dataset for the Analysis

For the analysis we use the same dataset as in the rest of this section. The main difference is how the `type` variable is considered: using the `get_dummies` function in `pandas`, we convert the values to categorical labels. The outcome of the operation is the conversion of the single `type` to two different columns in correspondence of the two different values taken by the entries (namely the new column `type_2` has 1 where `type` = 2 and 0 elsewhere, and `type_4` has 1 where `type` = 4 and 0 anywhere else).

2.6.2 Training and Validation Strategy

In this part of the analysis we change the approach of the *shallow* learning (i.e. at this stage anything but neural networks). When approaching the problem using linear models, SVM or decision trees we use cross-validation for training and evaluating the algorithms: we use 90 % of the total samples (646 entries) for cross-validation with 9 folds, that is for each iteration we use 10 % of the total number of samples for validation, and 10 % (72 samples) for testing purposes. When dealing with ANNs instead we keep the same approach as in the rest of the section: 80 % of the samples (574 entries) are kept for training, 10 % (72 entries) for validation and 10 % for test (72 samples).

2.6.3 Fourier Transformation and Scaling

Before using the dataset for predictions, we apply an additional transformation to the truncation levels.[8] In particular we process the sequence of approximations using the finite truncations as a signal ultimately converging to the extrapolated label. We apply a Fourier transformation to them using the *Fast Fourier Transform* algorithm in `numpy`. Since the input is purely real, we only compute the independent terms in the expansion using the `np.fft.rfft` function. We then feature engineer the output by separating the complex modulus and argument angle: we then use these features together with `weight` and the categorical `types` to predict the extrapolated label `exp` $\in \mathbb{R}$.

After the extraction of the features we first divide the argument angle by a factor π in order to have all angles in the interval $[-1, 1]$. We finally apply a `StandardScaler` to both complex modulus and angles to standardise the input variables (we do not touch `weight` and the categorical `types`).

2.6.4 Evaluation Metric

In order to properly process the truncation levels as subsequent approximations of the final result, we use a different evaluation metric. Specifically we use the average logarithm (base 10) of the ratio between the finite and predicted residual errors. Namely we use

$$R(y_{\text{true}}, y_{\text{finite}}, y_{\text{pred}}) = \frac{1}{N} \sum_{i=1}^{N} \log_{10} \left| \frac{y^{(i)}_{\text{true}} - y^{(i)}_{\text{pred}}}{y^{(i)}_{\text{true}} - y^{(i)}_{\text{finite}}} \right| , \tag{2.1}$$

where $y^{(i)}_{\text{true}}$ are the values of the predicted label `exp`, $y^{(i)}_{\text{pred}}$ are its predicted values and $y^{(i)}_{\text{finite}}$ are the values of the last finite truncation level. In the case of the *lumps* dataset, the last truncation level is `level_18`.[9] This way large negative numbers signal that *on average* the results of the ML analysis are improving the approximations using the finite level truncations.

2.6.5 Basic Training and Results

In this part of the analysis we train a linear model with ℓ_2 regularisation, a SVM with Gaussian kernel, a GBDT model and an ANN. For the first three models we use the Bayes optimisation of the hyperparameters, while for the ANN the same optimisation is done by hand. In general we use the MSE as scoring function for the optimisation.

The ANN model used in the analysis is a deep fully connected network. We used 6 hidden layers made of 50, 30, 20, 20, 10 and 10 units each with a `LeakyReLU` activation function (0.05

[8] Notice that we do not transform the `weight` or the `type` at any time.

[9] We clearly use the value of `level_18` before applying the Fourier transformation.

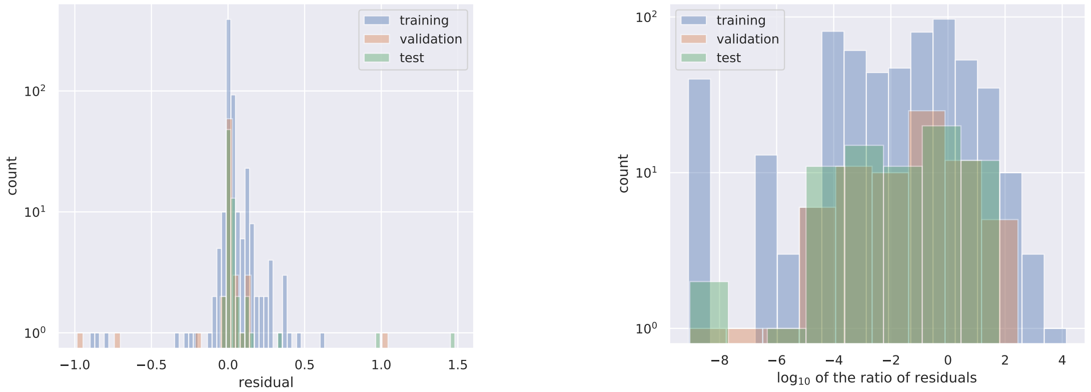
	MSE	MAE	r^2	R
LR	0.26	0.40	0.34	-0.46
SVR	0.19	0.18	0.53	-1.58
GBDT	0.08	0.20	0.80	-0.86
ANN	0.05	0.06	0.88	-1.66

Table 2.9: Predictions of the ML analysis trained the entire lumps dataset. Results refer to the test fold.

as slope factor) after each of them.[10] We also include a dropout rate of 0.03 after the first layer and 0.05 after the subsequent three (we do not insert dropout in the last two hidden layers). We do not include batch normalisation layers as they spoil the results during validation. We use the `Adam` optimiser with an initial learning rate $lr = 0.001$. We then schedule the learning rate to be reduced at each epoch n using

$$lr_n = \begin{cases} lr_{n-1} \times e^{-0.001}, & \text{if } lr_{n-1} > 10^{-6}, \\ 10^{-6}, & \text{if } lr_{n-1} \leq 10^{-6} \end{cases} . \tag{2.2}$$

We also use an early stopping regularisation when the validation loss function does not show improvement for 2500 epochs. We use the MSE as loss function.



(a) Residuals.

(b) Ratio of the predicted and final residuals.

Figure 2.13: Univariate distributions of the residuals using the ANN model on the full dataset.

In Table 2.9 we show the results obtained on the test fold using the full dataset. It seems that while the GBDT reach a better MSE result, SVR succeeds in improving the ratio of the residuals with respect to the finite approximation. Overall the neural network was able to achieve the best results, shown in Figure 2.13.

As a comparison we also trained the same algorithms on different subsamples of the dataset. In Table 2.10 we show the results on the test set obtained by selecting only `type` = 2 samples.[11]

[10] The depth of the network is such that it can learn a complicated function transforming the input while keeping the number of parameters as restricted as possible.

[11] The selection is made prior to subdividing the dataset into training and test sets, thus also the test fold is made of samples of the same `type`.

In Table 2.11 we instead show the results on the test set obtained with `type` = 4 samples.[12] We therefore see that the most problematic samples are usually in the `type` = 4 subset, where the largest improvements on the ratio of the residuals are made. However the `type` = 2 is remarkably well approximated by linear functions. In both cases the ANN showed great improvements in the predictions with respect to the approximation using finite truncation levels.

	MSE	MAE	r^2	R
LR	7.0×10^{-6}	2.0×10^{-3}	1.00	-0.13
SVR	1.6×10^{-5}	2.9×10^{-3}	1.00	-0.07
GBDT	3.0×10^{-4}	8.9×10^{-3}	1.00	0.16
ANN	7.0×10^{-8}	2.0×10^{-4}	1.00	-1.25

Table 2.10: Predictions of the ML analysis trained on `type` = 2 samples. Results refer to the test fold.

	MSE	MAE	r^2	R
LR	0.23	0.36	0.51	-1.00
SVR	0.03	0.05	0.94	-2.42
GBDT	0.47	0.65	-0.02	-0.63
ANN	0.03	0.03	0.94	-3.18

Table 2.11: Predictions of the ML analysis trained on `type` = 4 samples. Results refer to the test fold.

2.6.6 Signal Processing and LSTM Network

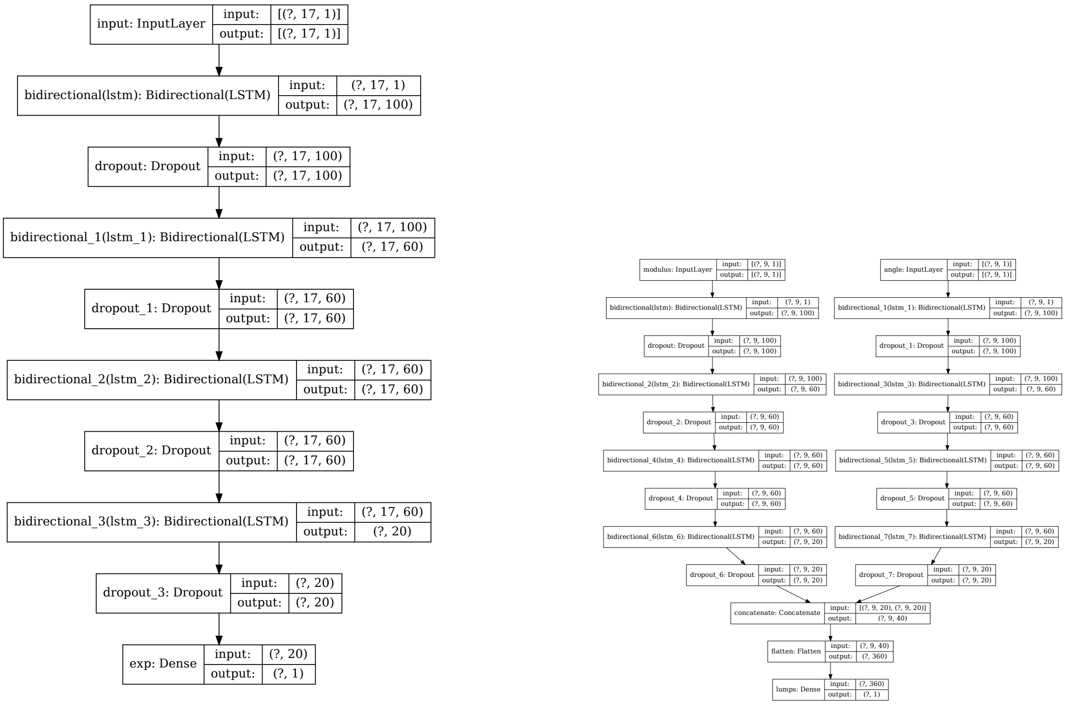
As an additional analysis we also processed the truncation levels as subsequent approximations of the extrapolated labels. We therefore used a *Long Short-Term Memory* (LSTM) neural network to study the finite truncation levels as signals reaching the `exp` labels.[13] In the first straightforward approach we used the unprocessed truncation levels and directly predicted the `exp` label using a single LSTM network. We then compared the results with the same process after feature engineering: we first applied a Fourier transform to the levels and then compute the predictions by combining two LSTM networks separately processing the complex modulus and the argument angle of the Fourier transformed levels. In both cases we drop the categorical `type` and the `weight` variables and use only the truncation levels.

The first model used is made of 4 bidirectional LSTM layers with 50, 30, 30 and 10 units each (we used the default activations to be able to use the `cuDNN` acceleration provided by the framework). We used 10^{-2} ℓ_2 regularisation in the first two layers and 10^{-3} in the last two hidden layers. We also implemented dropout layers with 0.2 dropout rate in the first two layers and 0.1 in the last two. The model was finally compiled with the `Adam` optimizer with an initial learning rate of 0.001. We then scheduled the learning rate to reduce as in (2.2). As in the previous cases an early stopping techniques was used to halt training after 2500 epochs without improving of the MSE, used as loss function.

[12] As in the previous case, the selection is made before splitting the dataset.

[13] LSTM networks are for instance used in making financial or atmospheric predictions using a series of data.

The second model is formed by two sub-architectures mirroring the same layers, regularisation and dropout as the first model. One of the sub-architectures is then fed the complex modulus of the Fourier transformed truncation levels while the other takes the angles as input. The two sub-architectures are then concatenated before outputting a single predicted labels as in Figure 2.14. The entire model is then trained as supervised task on the `exp` labels.


input: InputLayer | input: [(?, 17, 1)] | output: [(?, 17, 1)]
bidirectional(lstm): Bidirectional(LSTM) | input: (?, 17, 1) | output: (?, 17, 100)
dropout: Dropout | input: (?, 17, 100) | output: (?, 17, 100)
bidirectional_1(lstm_1): Bidirectional(LSTM) | input: (?, 17, 100) | output: (?, 17, 60)
dropout_1: Dropout | input: (?, 17, 60) | output: (?, 17, 60)
bidirectional_2(lstm_2): Bidirectional(LSTM) | input: (?, 17, 60) | output: (?, 17, 60)
dropout_2: Dropout | input: (?, 17, 60) | output: (?, 17, 60)
bidirectional_3(lstm_3): Bidirectional(LSTM) | input: (?, 17, 60) | output: (?, 20)
dropout_3: Dropout | input: (?, 20) | output: (?, 20)
exp: Dense | input: (?, 20) | output: (?, 1)
modulus: InputLayer | input: [(?, 9, 1)] | output: [(?, 9, 1)]
angle: InputLayer | input: [(?, 9, 1)] | output: [(?, 9, 1)]
bidirectional(lstm): Bidirectional(LSTM) | input: (?, 9, 1) | output: (?, 9, 100)
bidirectional_1(lstm_1): Bidirectional(LSTM) | input: (?, 9, 1) | output: (?, 9, 100)
dropout: Dropout | input: (?, 9, 100) | output: (?, 9, 100)
dropout_1: Dropout | input: (?, 9, 100) | output: (?, 9, 100)
bidirectional_2(lstm_2): Bidirectional(LSTM) | input: (?, 9, 100) | output: (?, 9, 60)
bidirectional_3(lstm_3): Bidirectional(LSTM) | input: (?, 9, 100) | output: (?, 9, 60)
dropout_2: Dropout | input: (?, 9, 60) | output: (?, 9, 60)
dropout_3: Dropout | input: (?, 9, 60) | output: (?, 9, 60)
bidirectional_4(lstm_4): Bidirectional(LSTM) | input: (?, 9, 60) | output: (?, 9, 60)
bidirectional_5(lstm_5): Bidirectional(LSTM) | input: (?, 9, 60) | output: (?, 9, 60)
dropout_4: Dropout | input: (?, 9, 60) | output: (?, 9, 60)
dropout_5: Dropout | input: (?, 9, 60) | output: (?, 9, 60)
bidirectional_6(lstm_6): Bidirectional(LSTM) | input: (?, 9, 60) | output: (?, 9, 20)
bidirectional_7(lstm_7): Bidirectional(LSTM) | input: (?, 9, 60) | output: (?, 9, 20)
dropout_6: Dropout | input: (?, 9, 20) | output: (?, 9, 20)
dropout_7: Dropout | input: (?, 9, 20) | output: (?, 9, 20)
concatenate: Concatenate | input: [(?, 9, 20), (?, 9, 20)] | output: (?, 9, 40)
flatten: Flatten | input: (?, 9, 40) | output: (?, 360)
lumps: Dense | input: (?, 360) | output: (?, 1)

(a) Straightforward model.

(b) Fourier transformed model.

Figure 2.14: LSTM models used in the analysis.

In Table 2.12 we show the metrics computed using the LSTM networks as described. It seems that using just the truncation levels can actually provide insightful information without using the weight or the oscillation type of the observables. In fact using the Fourier transformed truncation levels provides a non negligible average improvement on the ratio of the residual errors.

	MSE	MAE	r^2	R
LSTM	0.88	0.79	-1.22	-0.11
LSTM + FFT	0.25	0.24	0.36	-1.36

Table 2.12: Predictions of the test fold using the LSTM networks.

2.6.7 Double Lumps Inference

Using the previously trained algorithms we can also try to infer the results of the double lumps. For predictions we use only entries such that `weight` ≤ 1.25 since they represent the most reliable ground truth values available. In Table 2.13 we show the predictions made on the subset of double lumps observables using the previously trained algorithms. We also provide the result obtained by using the last finite truncation level as a comparison. Notice that in order to get to these results we multiplied the output of the ML models by a factor 2.

weight	type	exp	level_18	**LR**	**r-SVR**	**GBDT**	**ANN (FC)**	**LSTM**	**LSTM + FFT**
0.0	2	2.000132	2.010872	0.739581	-5.195649	1.029453	1.977009	1.753636	1.983604
0.0	2	-0.000045	0.000039	0.725948	-17.250588	1.252845	1.834650	0.317282	-0.108339
1.0	4	-2.000161	-2.048203	-0.526719	0.924571	-0.737725	-0.029497	-0.357706	-2.057807
0.0	4	1.998589	2.008174	0.966371	1.449645	1.557821	1.961694	1.752006	1.951245
0.027778	4	0.955293	0.875055	1.689489	1.045768	1.506691	1.976213	-0.422734	1.990272
0.111111	4	-1.083093	-1.174038	1.025812	1.427613	1.596725	1.954148	1.631063	-0.084639
0.25	4	-1.994476	-1.952275	0.415141	1.045758	0.815447	1.998802	-0.069527	1.677992
0.444444	4	-0.827465	-0.573408	0.083566	1.045758	0.818691	1.976839	0.556283	1.897519
0.694444	4	1.213654	1.409107	1.038451	1.045758	2.078447	2.027592	0.224882	1.948670
0.0	4	2.015004	1.585226	1.769235	1.045763	1.657983	2.097705	0.357322	1.990428

Table 2.13: Predictions of the double lumps using trained algorithms. Every prediction has been scaled (mulitplied) by a factor 2 to reach the values shown in the table. ANN refers to the fully connected (FC) model, while the LSTM network are divided into real and complex inputs.

As we can see both from the results and the metrics Table 2.14 using the lumps dataset is not enough to reproduce also the double lumps. In fact even by scaling the results we did not get a satisfactory result. The LSTM network using the Fourier transformed layers proved to work slightly better, but the result is still worse than using the finite truncation levels ($R > 0$). The difference between lumps and double lumps is therefore not just a scale factor. In fact the distributions of the variables are quite different in the two datasets.

	MSE	MAE	r^2	R
LR	1.7	1.1	0.28	1.40
r-SVR	37	3.7	-15.0	1.6
GBDT	2.3	1.3	0.02	1.48
ANN (FC)	4.2	1.6	-0.8	1.2
LSTM	2.2	1.3	0.09	1.4
LSTM + FFT	2.4	0.9	0.01	0.85

Table 2.14: Metrics computed on the double lumps. ANN refers to the fully connected (FC) model, while LSTM models are separately reported for both purely real and complex input.

3 WZW Model

3.1 Description and Preparation

The main difference from the previous dataset is represented by the presence of complex ($\mathbb{C}$) values in certain variables. In particular the truncation levels and the `exp` label have all complex values. Moreover there are additional variables which label the solutions such as the level `k` and the quantum numbers `j` and `m` referring to the SU(2) representation of the solution.

As in the previous case, the dataset is made of 46 vector-like entries which have to be flattened in the tidy version of the dataset. The length of the solutions is different in each entry. The number of solutions for different sizes is summarised in Figure 3.1.

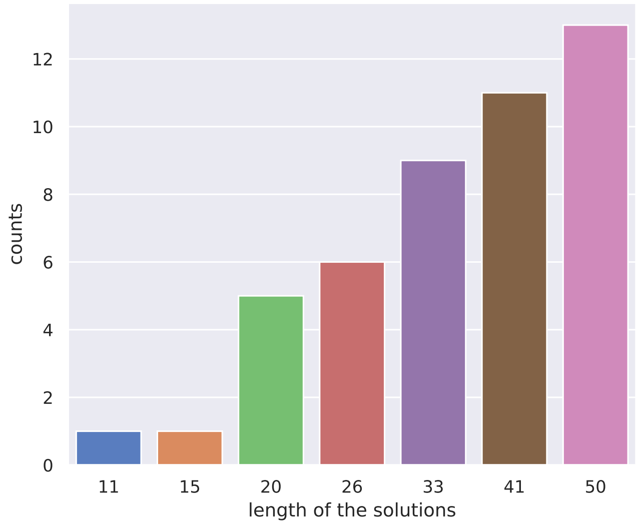


Figure 3.1: Length of various solutions.

In this case the available truncation levels start from the 2nd to the 14th, but since the last 4 levels have mostly empty values (`level_11` has only 15 values, while higher levels have only one or two values) we drop them and stop at level 10. We finally prune the dataset of the duplicates and drop 33 entries (or roughly 2 % of the total dataset) which are identical over all the variables.

The tidy dataset has therefore 1680 row entries and spans 25 variables including the real and imaginary parts of the label `exp`, the real and imaginary parts of the truncation levels, `k`, `weight`, `j` and `m`.

3.2 Exploratory Data Analysis

In the exploratory data analysis we mainly focus on the distribution of the values and patterns in the data. We also look for outliers and relations between the data.

3.2.1 Distribution of the Data

Looking at the summary of the data we recognise immediately some properties of the solutions. In Figure 3.2 we show a visual summary of the statistics associated with the variables labelling each solutions (without the truncation levels which will be studied later).[14] For instance we immediately recognise that both sum and average of the quantum number `m` are vanishing, together with possible correlations between `k`, `j` and `weight`.[15]

3.2.2 Outliers Distribution

Differently from the previous dataset the distribution of the variables is however more balanced (see Figure 3.3), even though the fraction of outliers is much larger than before. However the

[14] In the heatmap values have been normalised by their maximum value in order to be in the range $[-1, 1]$.

[15] In fact `weight` $= \frac{j(j+1)}{k+2}$.

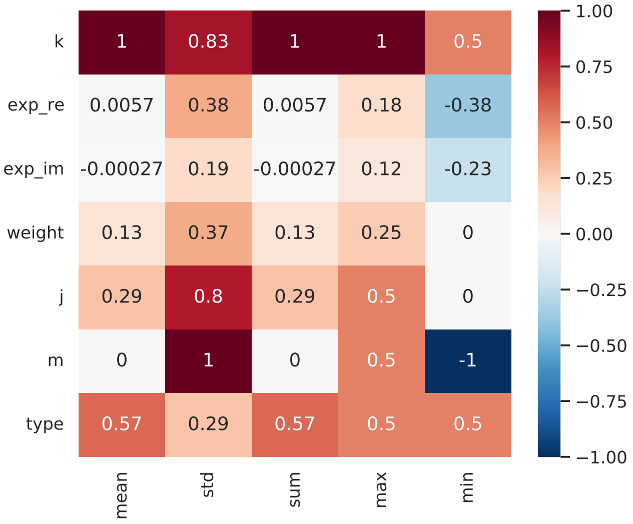


Figure 3.2: Visual representation of the summary statistics (normalised by their maximum value).

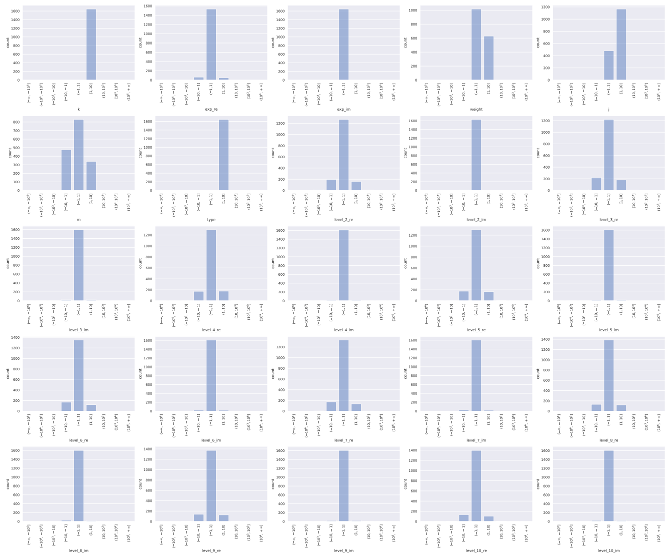
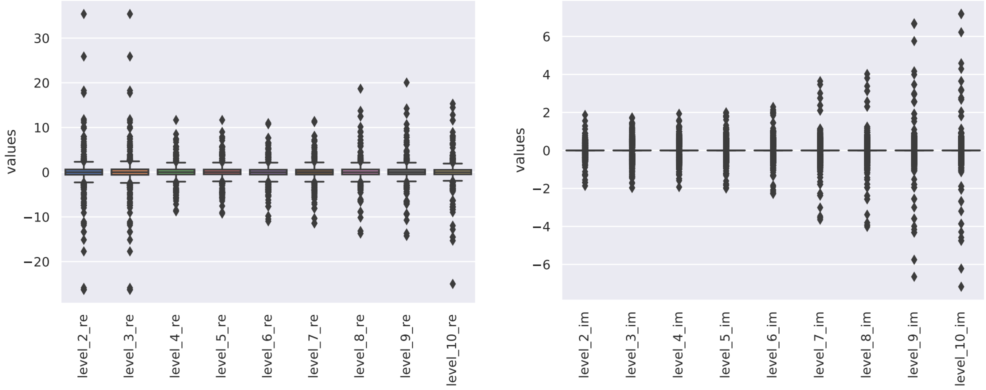


Figure 3.3: Distribution of the variables per order of magnitude.

large number of outliers is mainly due to the presence of non vanishing imaginary part in the truncation levels: only a small number of them is not a real number, thus the average of the imaginary part is narrowly peaked at 0 and any non vanishing contributions is an outlier (see Figure 3.4).



(a) Real part.

(b) Imaginary part.

Figure 3.4: Outlier distribution.

As in the previous dataset there are however strong relations between the data which have been summarised in Table 3.1. In particular we have:

- `weight` ≥ 1.0 and `type` $= 4$ and `k` $\in \{2, 3\} \Rightarrow$ `weight` $= 1$ and `j` $= 0$ and `m` $= 0$,
- `weight` ≥ 1.0 and `type` $= 4$ and `k` $= 4 \Rightarrow$ `weight` $= 1$,
- `type` $= 2 \Rightarrow$ `weight` $= 0$ and `j` $= 0$ and `m` $= 0$.

3.2.3 Correlation Matrix

Finally we show the correlation matrix of the features in Figure 3.5. From the correlations we can no longer recognise the oscillating behaviour as in the previous case. However we notice that real and imaginary parts are separately highly correlated features (though they are completely not correlated between them). Differently from the previous case the `weight` variable is poorly correlated with the other features, apart from the previously mentioned relation with `j`.

3.2.4 Principal Components Analysis

As in the previous case, we also analyse the principal components of the truncation levels. We perform the analysis in two separate ways: in the first we consider the whole group of truncation levels and robustly scale them against outliers (using the `RobustScaler` class in `Scikit-learn`), in the second we separate real and imaginary parts, standardise the first (using the `StandardScaler` in `Scikit-learn`) and robustly scale the latter. We then perform the same analysis as before.

As we see in Figure 3.6 after performing the SVD, in both cases a large part of the variance is already captured by one of the principal components (both the whole and separate datasets retain more than 99 % of the variance with just one component). It may therefore be possible

			weight		**j**		**m**	
weight	**type**	**k**	*mean*	*var*	*mean*	*var*	*mean*	*var*
≥ 1.0	4	2	1.00	0.000	0.00	0.00	0.0	0.00
		3	1.00	0.000	0.00	0.00	0.0	0.00
		4	1.00	0.000	1.35	0.90	0.0	1.39
		5	1.18	0.013	1.76	1.32	0.0	2.10
		6	1.26	0.048	2.35	1.05	0.0	3.00
		7	1.47	0.076	2.79	1.38	0.0	4.01
		8	1.57	0.130	3.21	1.21	0.0	4.93
< 1.0	2	2	0.00	0.000	0.00	0.00	0.0	0.00
		3	0.00	0.000	0.00	0.00	0.0	0.00
		4	0.00	0.000	0.00	0.00	0.0	0.00
		5	0.00	0.000	0.00	0.00	0.0	0.00
		6	0.00	0.000	0.00	0.00	0.0	0.00
		7	0.00	0.000	0.00	0.00	0.0	0.00
		8	0.00	0.000	0.00	0.00	0.0	0.00
	4	2	0.31	0.047	0.67	0.17	0.0	0.50
		3	0.45	0.083	1.00	0.28	0.0	0.83
		4	0.38	0.053	1.00	0.26	0.0	0.77
		5	0.50	0.090	1.33	0.39	0.0	1.18
		6	0.44	0.071	1.33	0.40	0.0	1.14
		7	0.56	0.108	1.67	0.56	0.0	1.67
		8	0.50	0.089	1.66	0.56	0.0	1.64

Table 3.1: Relations between the weight and type variables, and other quantum numbers.

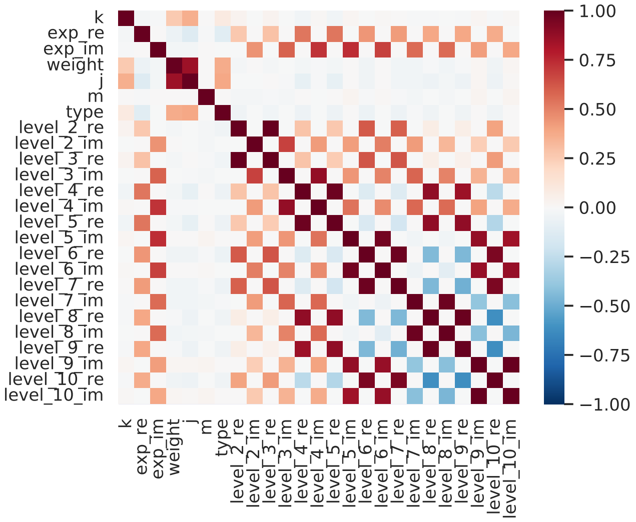
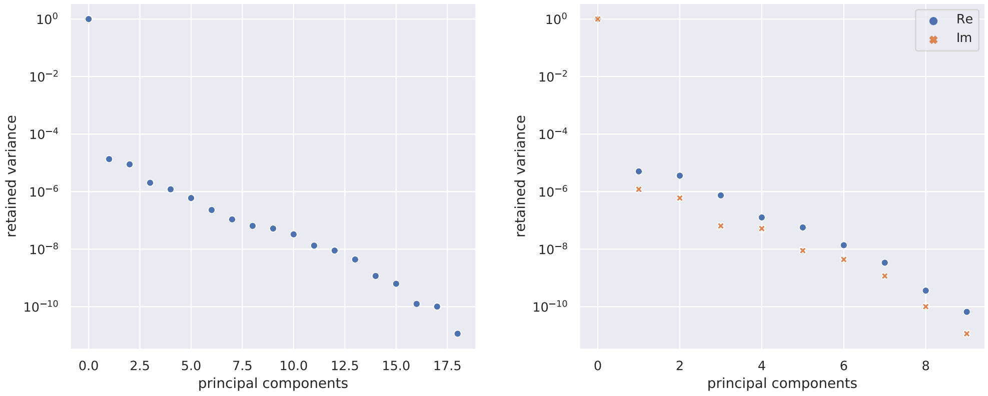


Figure 3.5: Correlation matrix of the WZW model.

to use the principal components to have a fixed input size for the algorithms and be compatible with other datasets.



(a) Whole dataset.

(b) Separate dataset.

Figure 3.6: Principal components of the truncation levels.

3.3 Statistical Inference

As for the previous case, using the EDA data we performed the ANOVA on the WZW model using a simple linear regressor. For this we used 80 % of the dataset for training and the rest as development set: since the data is already labelled, we do not need to separate the samples according to the `solutions` variable. However in this case we will keep all the variables present in the dataset and perform the regression predicting both the real and imaginary parts of `exp` simultaneously. With 313 d.o.f. we reached a MSE of 0.06 with a 95 % C.I. $[0.0, 0.07]$ and $r^2 = 0.83$ (both are better than the previous dataset signalling that features and labels may be more correlated in this case).

In Table 3.2 we show the results of the analysis: we show the choice of the coefficients and their statistics in separate columns for the real and imaginary parts of `exp`). Differently from the previous case the data is a bit more complex and in some cases it shows that we could actually drop some of the variables. For instance we will certainly drop `k` which does not seem to influence the final result (its p-value is very high). Curiously enough, it seems that in order to predict $\mathrm{Re}(exp)$ we could just use the real parts of the variables in the dataset, while the situation for $\mathrm{Im}(exp)$ requires the contributions of both real and imaginary parts of the input features.

3.4 Model Dependent Deep Learning Analysis

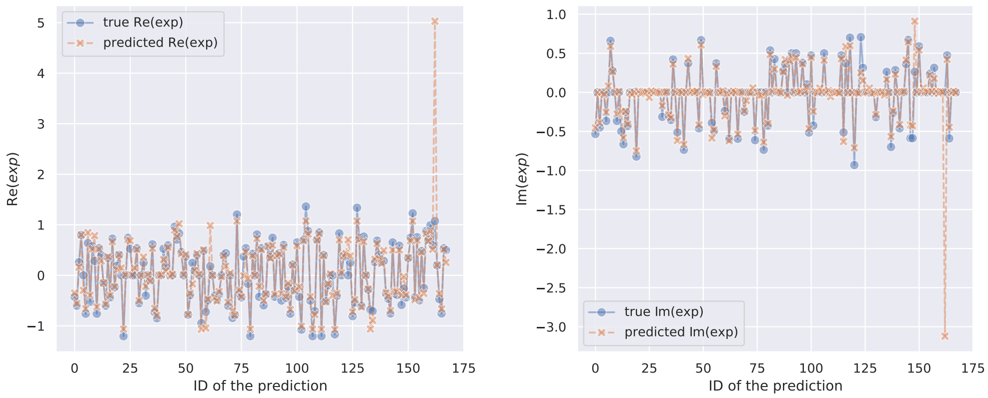
As a prosecution of the exploratory analysis we also performed a prediction analysis using the same ANN model used for the previous dataset. The necessary modifications however concern the input shape of the architecture (here we have more input variables) and the output layer: we are interested in predicting both real and imaginary parts of the output at the same time. This in turn will not be necessary for the aggregate analysis but it might be worth noting the results.

For the learning model we split the dataset into 80 % for training, 10 % for validation and the remaining 10 % as a test set. In general the ANN model behaved extremely well in the training and validation folds, while it performed poorly in the test set: the r^2 score for both

	coeff. – Re	coeff. – Im	std. err. – Re	std. err. – Im	t – Re	t – Im	p-value – Re	p-value – Im
`k`	0.003	-0.005	0.015	0.002	0.176	-2.977	0.861	0.003
`weight`	0.05	-0.005	0.03	0.004	1.520	-1.361	0.130	0.175
`j`	-0.033	0.00035	0.015	0.0017	-2.263	0.211	0.024	0.833
`m`	0.000	0.0003	0.013	0.0014	0.075	0.230	0.940	0.818
`type`	0.00	0.010	0.04	0.0047	0.058	2.234	0.954	0.026
`Re(level 2)`	0.329	0.0088	0.006	0.0007	51.791	12.163	0.000	0.000
`Im(level 2)`	0.02	-0.034	0.09	0.011	0.265	-3.271	0.791	0.001
`Re(level 3)`	-0.349	-0.0086	0.006	0.0007	-55.653	-12.120	0.000	0.000
`Im(level 3)`	-0.07	-0.064	0.05	0.006	-1.242	-10.764	0.215	0.000
`Re(level 4)`	-0.516	0.0145	0.015	0.0017	-35.415	8.785	0.000	0.000
`Im(level 4)`	0.11	0.062	0.06	0.006	1.890	9.801	0.060	0.000
`Re(level 5)`	-0.036	-0.0175	0.014	0.0016	-2.519	-10.871	0.012	0.000
`Im(level 5)`	-0.10	-1.507	0.05	0.006	-2.019	-267.397	0.044	0.000
`Re(level 6)`	-4.931	-0.0256	0.014	0.0016	-340.537	-15.605	0.000	0.000
`Im(level 6)`	0.15	1.939	0.05	0.006	2.960	346.379	0.003	0.000
`Re(level 7)`	4.539	0.0262	0.014	0.0016	328.362	16.776	0.000	0.000
`Im(level 7)`	-0.00	-3.980	0.04	0.005	-0.132	-807.934	0.895	0.000
`Re(level 8)`	-3.71	-0.0383	0.013	0.0015	-279.284	-25.439	0.000	0.000
`Im(level 8)`	-0.04	4.444	0.04	0.005	-0.991	960.701	0.322	0.000
`Re(level 9)`	4.684	0.0406	0.013	0.0014	367.810	28.150	0.000	0.000
`Im(level 9)`	0.08	-2.587	0.03	0.003	2.756	-775.714	0.006	0.000
`Re(level 10)`	0.874	0.0009	0.012	0.0013	75.406	0.693	0.000	0.489
`Im(level 10)`	-0.14	2.687	0.03	0.003	-4.800	840.827	0.000	0.000

Table 3.2: Results of the ANOVA on the linear model.

Re(exp) and Im(exp) dropped respectively to 0.66 and 0.30 in the test set while it was above 0.94 in both training and validation folds.[16] This however seems to be entirely due to a small number of samples in the test set which drove away the MSE and the r^2 score with respect to the validation and training sets. In Figure 3.7 we can clearly see the sample (the same between real and imaginary parts) spoiling the result.



(a) Real part.

(b) Imaginary part.

Figure 3.7: Predictions and true values of the `exp` label.

[16] As a consequence also the MSE plummeted in the test set.

4 Aggregate Analysis

4.1 Preparation of the Input

In this section we focus on predicting both real and imaginary parts of the `exp` label using both the lumps and the WZW datasets together: the idea is to have a model independent architecture able to make predictions without knowledge of the underlying physical model. In fact we will also use part of the double lump solutions to make sure that the ML models are properly independent on the physical models underlying the data.

The datasets of the lumps and WZW models are however defined differently and many variables differ in the two physical models. We start from the tidy datasets which have been prepared for the separate analysis in the previous sections.

We will proceed as follows, before merging the datasets:

- we transform the columns in the lumps dataset to account for the imaginary parts of the truncation levels (identically vanishing), that is we add columns corresponding to the imaginary part of the truncation levels,
- we compute the PCA of the truncation levels in both datasets, keeping 10 components each to maximise the retained variance,
- we then select the `weight`, `type` and the PCA variables in both datasets,
- we *outer join* the datasets on the `weight` and `type` variables.

We finally have a new dataset containing 12 features and 2 labels (i.e. $\mathrm{Re}(exp)$ and $\mathrm{Im}(exp)$), and 2379 samples.

We then prepare the dataset containing data on the double lump solutions. First of all we select only data with weight $h < 1.5$ for reliability. We then apply the same transformations described earlier, but we do not immediately merge the data with the previous dataset. Since these data come from a different model but refer to only one solution, we keep them separate and use them mainly for evaluation purposes in the validation and test sets used in the analysis.

The final result of the preparation is as follows:

- one dataset containing 12 features and 2 labels across 2379 samples (i.e. lumps and WZW data),
- one dataset containing 12 features and 2 labels with 12 samples (i.e. the double lumps).

We can now move to the analysis of the machine learning algorithms. We first need to define a validation strategy for the different datasets.

4.2 Validation Strategy

For the analysis we keep in general 80 % of the samples for training, 10 % for validation and 10 % as a test set as we did before for the two separate datasets. In this case the datasets can be freely shuffled since all the information on the model is already encoded in the variables.[17]

However we treat the double lumps dataset in a different way. The steps necessary to reproduce the input datasets are as follows:

- take the joined dataset with lumps and WZW data and sample it with 80 % of the data for the training set, 10 % for the validation set, 10 % for the test set,

[17] That is we do not need to add a `solutions` column and use that to separate the samples.

- take the double lumps data and split it in half: insert the first half in the validation set, and the second half in the test set.

Before passing the input to the algorithms we scale it using the `StandardScaler` class in `Scikit-learn` in order to standardise the features and simplify the learning process. We also scale the labels to contain the range of variability of the predictions.[18] For this we use the `MinMaxScaler` class in `Scikit-learn`. In both cases we fit the transformers on the training set and then apply the transformation to the validation and test sets.[19]

We focus on predicting both Re(exp) and Im(exp) at the same time with a single model. We will use and compare *r-SVM*s, *GBDT*s and *ANN*s models. However, since the first two algorithms cannot naturally account for two outputs, we use the `MultiOutputRegressor` class in `Scikit-learn` which automatically uses the same estimator to predict separately both labels.

We adopt two different validations strategies. In fact, for support vector machines and gradient boosted trees we use a cross validation strategy: we join the training and validation sets in this case to have 90 % of data in total for training and validation. We adopt 9-folds cross validation to always have 10 % of the data for evaluation. Thus we will in general train the algorithm on 8 folds and evaluate on the 9th: every time there will be a chance to train on the double lumps data which might help the algorithms to predict also those solutions.

4.3 Support Vector Machines

4.3.1 Training

The model used for training is a single architecture to predict both real and imaginary parts of `exp`. Since we use a cross validation strategy, we perform the automatic optimisation of the hyperparameters using the `BayesSearchCV` class in `Scikit-optimize`, which uses Bayes optimisation to look for the best hyperparameters. In the end the best hyperparameters chosen to predict both real and imaginary parts of `exp` are $C = 12$, $\epsilon = 0$, and $\gamma = 10$.[20]

4.3.2 Results

Results on the test fold are summarised in Table 4.1. In general the algorithm performs better on the imaginary part of the label, but results are good also for the real part.

	MSE	MAE	r^2
Re(exp)	0.01	0.04	0.84
Im(exp)	0.0005	0.004	0.97

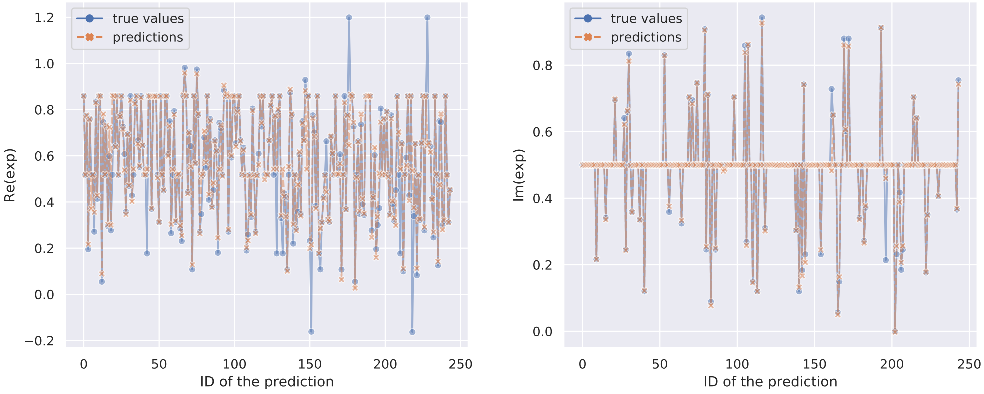
Table 4.1: Summary of the metrics of the *r-SVR* on the test set.

In Figure 4.1 we show the predictions of the real and imaginary parts of `exp` using the *r-SVR* algorithm. As we can see most of the predictions are correctly reproduced by the weights of the support vectors. Some predictions are slightly underestimated in the real part of the label, but the residual plot in Figure 4.2 shows that the residuals are however contained.

[18] Predictions will not be directly comparable, but we must remember to scale any new data before computing the metrics.

[19] We do not simply divide the data by a constant arbitrary number because that would not change anything in the algorithms. It would also add a bias in the analysis, since there is no way to predict what constant factor to use.

[20] C refers to the penalty assigned to samples outlying the *no penalty* margin of support vectors, ϵ to the soft margin width, and γ to the width of the Gaussian kernel.



(a) Real part.

(b) Imaginary part.

Figure 4.1: Predictions and true values using the *r-SVR* algorithm.

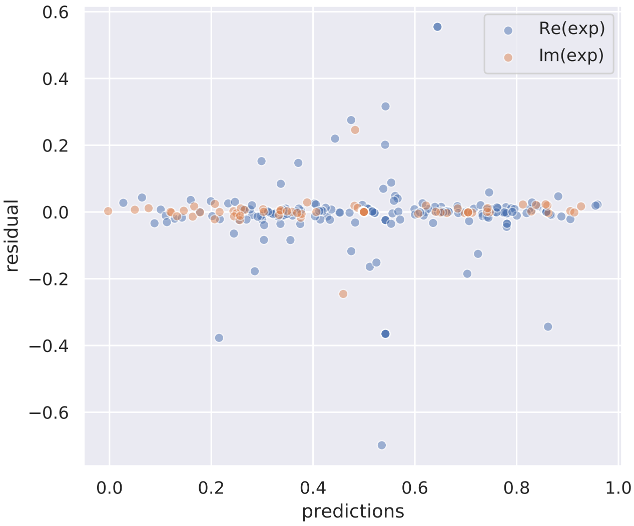


Figure 4.2: Residual plot using the *r-SVR*.

4.4 Gradient Boosted Decision Trees

4.4.1 Training

We use the same training strategy of the *r-SVR* algorithm. The optimal hyperparameters are: `colsample_bytree` $= 0.7$, `learning_rate` $= 0.01$, `max_depth` $= 25$, `min_child_weight` $= 0.1$, `n_estimators` $= 50\,000$, `num_leaves` $= 25$, `reg_alpha` $= 1.0$, `reg_lambda` $= 1000$, `subsample` $= 0.99$.[21]

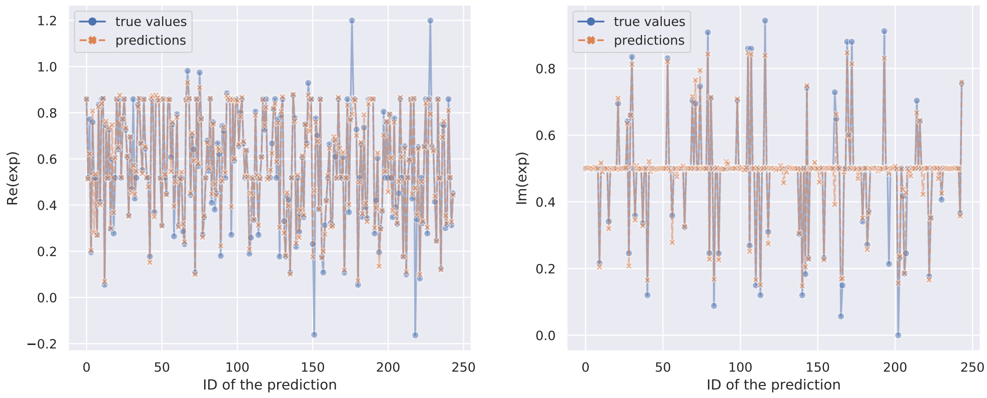
4.4.2 Results

Results on the test sets are shown in Table 4.2.

	MSE	MAE	r^2
$\mathrm{Re}(exp)$	0.007	0.03	0.89
$\mathrm{Im}(exp)$	0.0014	0.012	0.93

Table 4.2: Summary of the metrics of the *GBDT* on the test set.

In Figure 4.3 we show the predictions of the real and imaginary parts of `exp`. As in the previous case, some predictions are slightly underestimated in the real part of the label, but the residual plot in Figure 4.4 shows that the residuals are however contained.



(a) Real part.

(b) Imaginary part.

Figure 4.3: Predictions and true values using the *GBDT* algorithm.

In general we can see that the real part of the label is better approximated by the *GBDT*, which however suffers a bit more in the imaginary part with respect to the *r-SVR*. In general the results are very good since the coefficient of determination r^2 is well above 0.8 for both the labels.

[21] `colsample_bytree` refers to the fraction of features used in each tree, the `learning_rate` to the shrinking parameters used in the gradient descent, `max_depth` is the maximal amount of branches in each tree, `min_child_weight` is the minimum number of samples to branch the tree, `n_estimators` is the number of boosting round, `num_leaves` is the number of leaves in each branch, `reg_alpha` is the amount of ℓ_1 regularisation, `reg_lambda` is the amount of ℓ_2 regularisation, and `subsample` is the fraction of samples used.

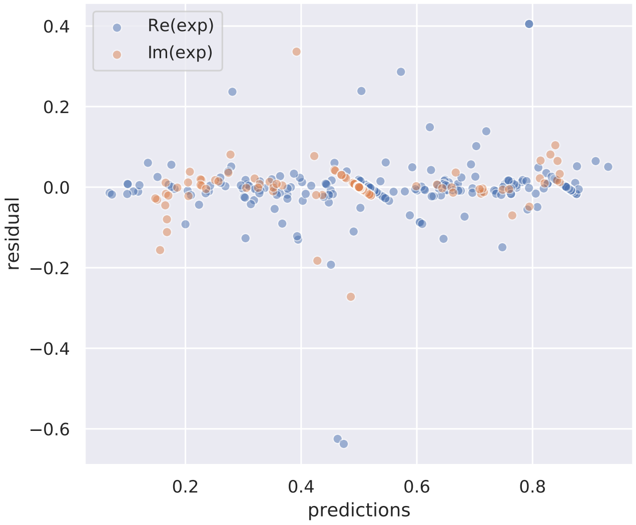


Figure 4.4: Residual plot using the *GBDT*.

4.5 Artificial Neural Network

4.5.1 Model

The model we use for training is a simple fully connected network (summarised in Table 4.3). The architecture is made of 4 hidden layers with 50, 50, 10 and 10 units each and dropout layers (with rate 0.05) after the first two hidden layers (see Figure 4.5). Each hidden fully connected layer is followed by a ReLU activation function. The addition of batch normalisation layers drastically increased the error, so we dropped them entirely in this implementation. The output layers are two separate fully connected layers with 1 unit each containing Re(exp) and Im(exp). No regularisation was introduced.

layer	shape	parameters
input	(12,)	0
fully connected	(50,)	390
dropout	(50,)	0
fully connected	(50,)	930
dropout	(50,)	0
fully connected	(10,)	310
fully connected	(10,)	110
Re(exp) ($output$)	(1,)	11
Im(exp) ($output$)	(1,)	11
Total parameters:	3842	
Trainable parameters:	3842	

Table 4.3: Summary of the network.

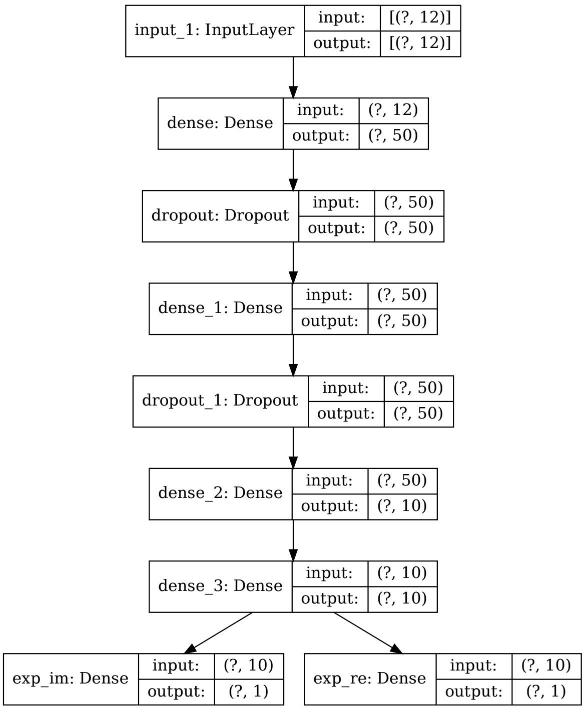
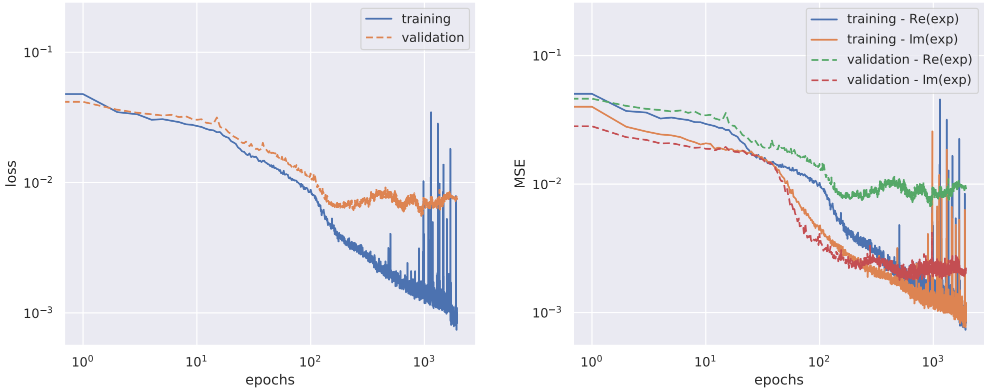


Figure 4.5: Architecture of the ANN.

4.5.2 Training

For training we use the MSE as loss function and weigh it on the two output layers with 0.75 and 0.25 loss weights for the real and imaginary part of the `exp` label respectively.[22] For gradient descent we use the *Adam* optimiser with default values and initial learning rate of 0.001 and a mini batch size of 32. The maximal number of epochs used for training is 20 000 but it greatly exceeds what actually needed for good results. In fact we also implement a callback to early stop the training after 1000 epochs without improvement of the loss function on the validation data. We finally add a callback to reduce the learning rate by a step factor of 0.3 after 750 epochs without improvement on the validation loss.

The loss function and the MSE are displayed in log scale in Figure 4.6. They show a drastic drop in error and loss around 100 epochs of training and a stabilisation after that. Early stopping the network has also the regularisation effect of avoiding the overfit of the traning set.



(a) Weighted loss.

(b) MSE metric.

Figure 4.6: Loss function and errors (log scale).

Differently from the cross validation strategy adopted for *r-SVR* and *GBDT*, for the *ANN* we inserted the double lump solutions only in the validation and test sets. The algorithm is therefore trained only on lumps and WZW model but evaluated on all real world scenarios.

4.5.3 Results

The metrics computed on the test set are shown in Table 4.4. The results are in general good and the coefficient of determination r^2 is very high for both labels. Comparing with the results in the validation set ($r^2 = 0.89$ for $\mathrm{Re}(exp)$ and $r^2 = 0.89$ for $\mathrm{Im}(exp)$), it seems that the architecture does not overfit the validation set.[23]

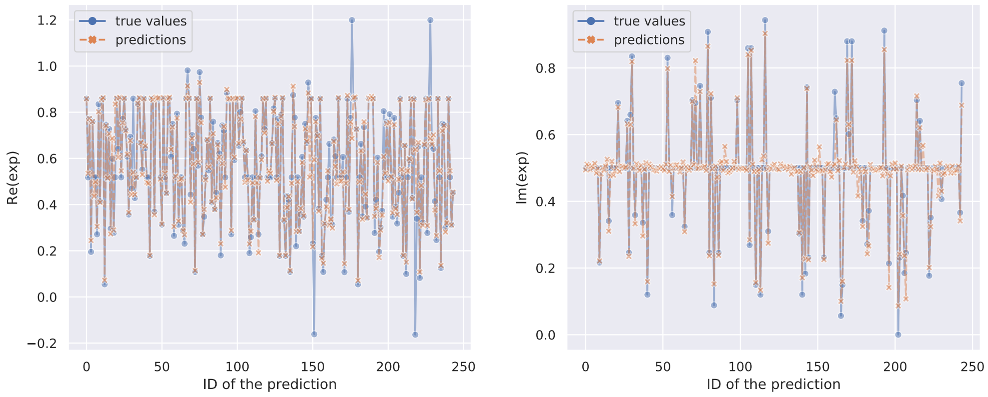
In fact, Figure 4.7 shows that the agreement between predictions and true values in the test set is very good. It also does not show signs of isolated samples producing a completely wrong predictions as for the *r-SVR*. Taking a look at the residuals we can also see that the errors are in general well distributed and do not show sign of patterns (see Figure 4.8).

[22] The loss function must be a scalar metric, thus the MSE in this case is separately computed on the output layers and then combined by multiplying each loss by the corresponding weight. Clearly the sum of all weights should be the unit.

[23] This is in general a risk when using a single validation set.

	MSE	MAE	r^2
Re(exp)	0.010	0.03	0.84
Im(exp)	0.0008	0.016	0.95

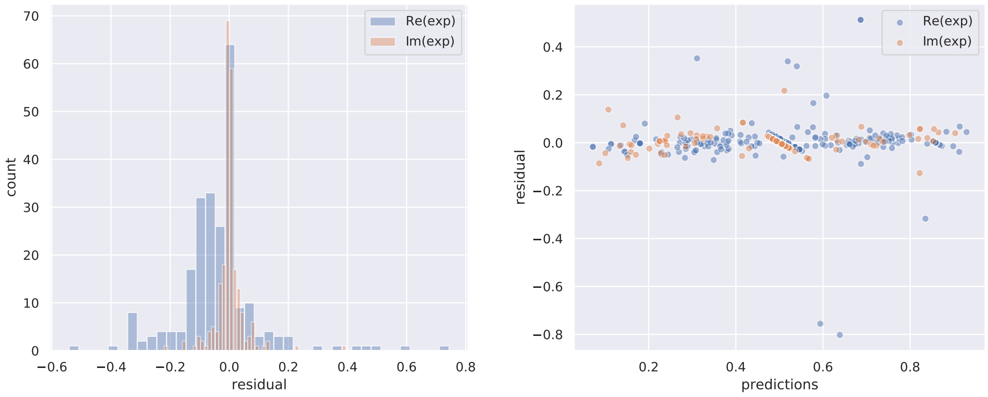
Table 4.4: Metrics of the *ANN* computed on the test set.



(a) Real part.

(b) Imaginary part.

Figure 4.7: Predictions and true values using the *ANN* model.



(a) Univariate distribution.

(b) Residual plot.

Figure 4.8: Residuals of the *ANN* model.

5 Conclusions

In the analysis we show that ML techniques are indeed able to make accurate predictions on lumps and the WZW model in SFT. In fact decision trees and *ANN* models showed promising results on a model dependent basis (i.e. with knowledge of the underlying physics model). In principle it seems to also be possible to merge different datasets and produce meaningful predictions using *ANN* models (and *r-SVR* in some cases) which displayed the best adaptivity to different datasets.

It is still not clear whether it is possible to use algorithms trained on different datasets (even coming from diverse physical models) and make accurate predictions on different data, produced by a different physical model. However when introducing the double lumps directly in the evaluation and test sets, the *ANN*s showed ability to generalise to these solutions. As shown by the coefficient of determination r^2, the variance of the data is greatly explained by the model, which can therefore be used for predictions.

References

[1] P. Virtanen, R. Gommers, T. E. Oliphant *et al.*, 'SciPy 1.0: Fundamental algorithms for scientific computing in python', *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, 'Scikit-learn: Machine learning in python', *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html.

[3] T. Head, MechCoder, G. Louppe *et al.*, *Scikit-optimize/scikit-optimize: V0.5.2*, version v0.5.2, Zenodo, Mar. 2018. DOI: 10.5281/zenodo.1207017;http://web.archive.org/web/20200812085913/https://zenodo.org/record/1207017. [Online]. Available: https://doi.org/10.5281/zenodo.1207017.

[4] G. Ke, Q. Meng, T. Finley *et al.*, 'LightGBM: A highly efficient gradient boosting decision tree', in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio *et al.*, Eds., Curran Associates, Inc., 2017, pp. 3146–3154. [Online]. Available: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

[5] J. D. Hunter, 'Matplotlib: A 2D graphics environment', *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[6] S. M. Lundberg, G. Erion, H. Chen *et al.*, 'From local explanations to global understanding with explainable AI for trees', *Nature Machine Intelligence*, vol. 2, no. 1, pp. 2522–5839, 2020. DOI: 10.1038/s42256-019-0138-9.

[7] M. Abadi, A. Agarwal, P. Barham *et al.*, 'TensorFlow: Large-scale machine learning on heterogeneous systems', 2015. [Online]. Available: https://www.tensorflow.org/.