Report

Alessia De Ponti,
Physics
(Complex Systems)
a.deponti@studenti.unipi.it

Elena Scaglione,
Digital Humanities
(Language Technologies)
e.scaglione1@studenti.unipi.it

Matteo D'Onofrio,
Computer Science
(Big Data)
m.donofrio7@studenti.unipi.it

ML course (654AA), Academic Year: 2022/23
Date: 23/01/2023
Type of project: **B**

Abstract

After implementing several techniques and architectures seen during the course, we selected deep NN (4 hidden layers, 7 units each) with minibatch for the cup. For each hypermodel, we performed an hold-out for model selection and a k-fold for model assessment and, then, another hold-out to select the best model among these with final retraining.

1 Introduction

The aim of our analysis has been to compare the performances of simple models (such as KNN and linear regression) with more sophisticated ones (deep neural networks) on a relatively simple dataset like the one we were given. In particular, we tried to highlight the relationship between the complexity of the model and the overall score obtained (according to the MEE metric).
The models we tried out are: linear regression, ridge regression, KNN regressor, SVR, shallow regressor, deep regressor, RandNN, CNN and cascade correlation.
When implementing a CNN architecture we implicitly assumed that the attributes' order has some kind of significance (eg. column lying next to each other are somehow more related than further apart ones). Note that this is not directly implied by the data. Although this model is mostly used for datasets which underly a pattern-like structure (eg. images), we tried it anyway for experimentation's sake.

2 Method

2.1 Simulators

- **Libraries:** we relied mostly on Keras and Sklearn. However, we used Pytorch on a few simple architectures.
- **Common framework:** in order to compare results obtained starting from different hypermodels, we developed a class as a way of sharing methods among them. Thus, each architecture (treated as an instance of the class) follows the same path for model selection and assessment.
- **Hyperparameters space searching strategies:** firstly we tried grid search which, although computationally expensive, highlights the role of different hyperparameters. However, when the possible combinations are several hundreds, this method is lengthy. For this reason, we opted for the **bayesian optimization** searching strategy, which is a good trade-off between randomness and computational efficiency. Specifically, the number of performed trials is set to 15% of the dimension of the hyperparameters space.
- **Initialization:** weights are initialized uniformly in $[-\sqrt{6/\text{fan-in}}, \sqrt{6/\text{fan-in}}]$, whereas bias is set to zero.
- **Activation functions:** we used ReLu for hidden layers and sigmoid or linear in output layers for classification and regression respectively.
- **Batch size:** we used mostly batch and minibatch technique, however, we tried preliminarly on-line on a few models. This allowed us to see practically the difference that this parameter makes in training (smoothness of the learning curve and training duration).
- **Regularization techniques:** preventing overfitting has been a crucial topic in our analysis. Namely, we set an artificially high number of epochs (500) for training so that it continues until **early stopping** based on the validation score. Moreover, we decided to **decrease the learning rate** when reaching a plateau on validation loss, so to help approach a local minimum.
  Other regularization techniques catered to each architecture have been implemented in the hypermodel's structure, such as dropout and weight decay for NN models, α term in ridge model, C and ε-tube values in SVM.

2.2 Workflow

- **Preprocessing:** we used StandardScaler in KNN model to standardize mean and standard deviation.
- **Validation schema:** K-Fold CV for assessment with internal hold out for model selection.

2.3 Preliminary trials (not pursued)

- **Preprocessing:** we tried to reduce the inputs' dimension using an autoencoder, PCA and feature selection. Although these procedures could be useful to contrast the curse of dimensionality in general, in our specific case none of them improved the overall performance.

- **Pruning:** we used the pruning technique over a deep RandNN and then retrained the final net. Even though this algorithm seemed somewhat promising, the results were not satisfactory, especially if compared with RandNN without pruning.

3 Experiments

3.1 Monk Results

Task	Hypermodel	Hyperparameters	MSE (TR/TS)	Accuracy (TR/TS)
MONK1	DeepNN mb	**units**:6, **depth**:1, **dropout**:0, **lr**:0.01, **decay**:0.001	**TR:** 0.005, **TS:** 0.008	**TR:** 1, **TS:** 1
MONK2	DeepNN mb	**units**:6, **depth**:1, **dropout**:0, **lr**:0.01, **decay**:0.001	**TR:** 0.003, **TS:** 0.003	**TR:** 1, **TS:** 1
MONK3	Linear Pytorch	**lr**: 1	**TR:** 0.07, **TS:** 0.06	**TR:** 0.93, **TS:** 0.97

Table 1: Best prediction results obtained for the MONK's tasks.

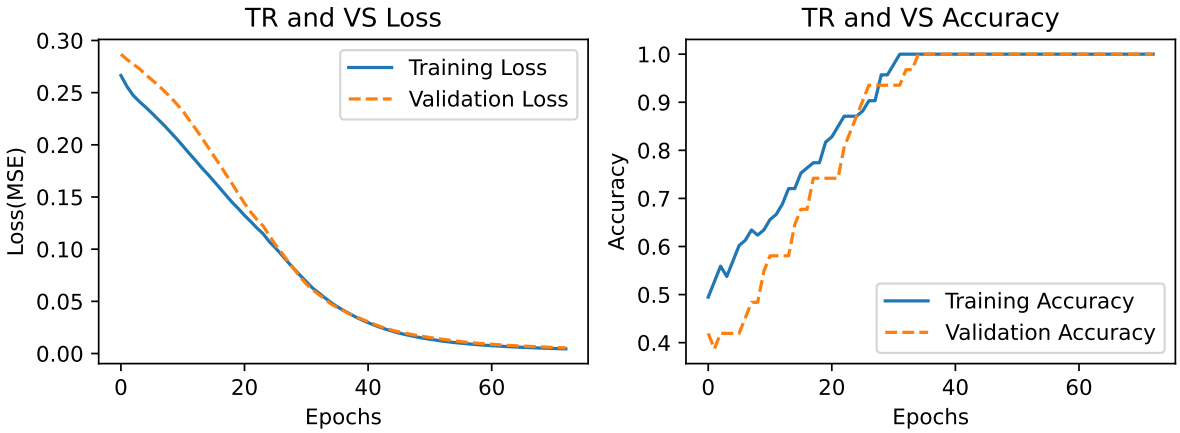

Figure 1: Monk1 DeepNN mb learning curve and accuracy.

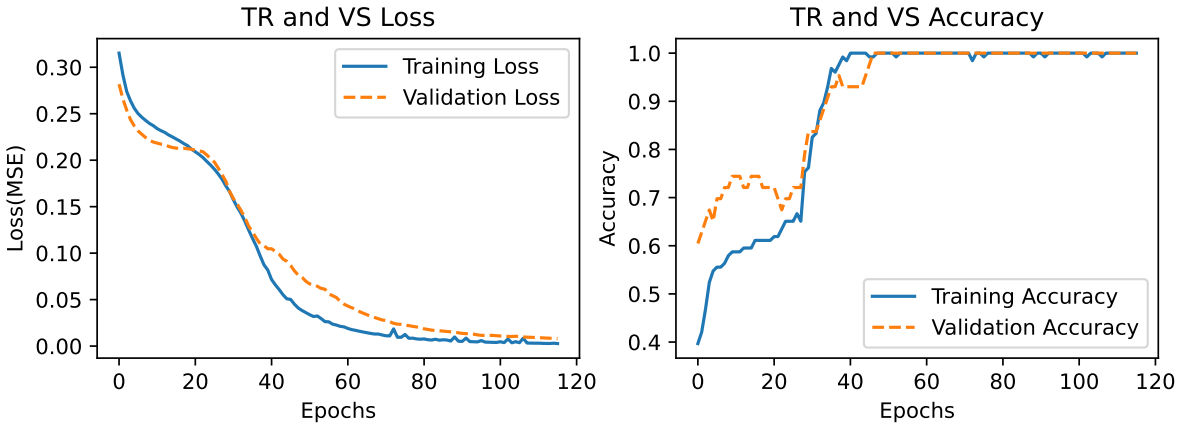

Figure 2: Monk2 DeepNN mb learning curve and accuracy.

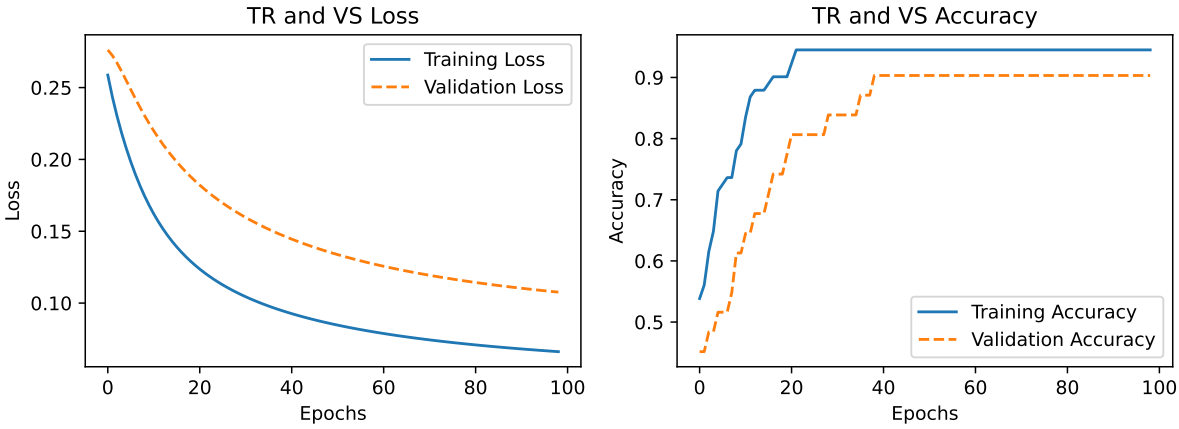

Figure 3: Monk3 Linear Classifier learning curve and accuracy

3.2 CUP Results

- **Validation Schema:** given an hypermodel and its hyperparameters space, the best model is selected with an **internal Hold-out** (VS=25% of development set). A **K-fold CV** (K=5) procedure is chosen to evaluate the model's performance.
  Note that, by doing so, each pattern belongs once to TS, once to VS and thrice to TR.
  At this point, we are left with 5 models (one for each fold), so it is necessary to design another model selection to choose the best one. We perform an **Hold-out** (TR=80% of total data, VS=20%) to pick the model among those with the lowest validation score and carry out a final retrain. To assure replicability, we set a consistent random seed.

- **Screening phase:** During the data preparation phase, we compared the performances of preprocessed and raw data by building a grid search to choose between PCA and select k-best. The highest score was obtained by select k-best with k=number of features, so no column was discarded. We tried to build an autoencoder as well but the model's performance worsened. That's why we decided not to reduce the dimensionality using preprocessing methods.

- **Hyperparameters:** The Table 2 summarizes the hyperparameter space explored for each hypermodel.

Hypermodel	**Hyperparameters**
Ridge reg	α: [10^{-3}, 10^{-5}, 0.7, 0.9, 1, 2], η_0: [0.01, 0.1, 0.001, 1]
KNN	**n neighbors**: range(5, 35, step=2), **metric**:["uniform", "distance"], **weights**:["euclidean", "cityblock"]
SVR	**C**: [0.01, 0.1, 0.3, 0.7, 1.0], ε:[0.01, 0.1, 0.9, 1.5, 2], **kernel**: ['linear', 'rbf']
Keras models	**units:**[4,5,6,7], **depth:**[2, 3, 4], **dropout:**[0, 0.001], **lr:**[0.01, 0.001], **decay:**[0, 0.001]

Table 2: Hyperparameters space

- **Best performances:**

Hypermodel	Hyperparameters	MEE (TR/VS/TS)
Ridge reg	α: 10^{-3}, η_0:0.01	**TR:** 2.13,**VS:** 2.16, **TS:** 2.17 ± 0.08
KNN reg	**n neighbors**: 15, **metric**: euclidean, **weights**: distance	**TR:** 0 ,**VS:** 1.42, **TS:** 1.44± 0.07
SVR	**C**: 1, ε: 0.1, **kernel**: rbf	**TR:** 1.39,**VS:** 1.51, **TS:** 1.49 ± 0.11
DeepNN mb	**units**: 7, **depth**: 4, **dropout**: 0, **lr**:0.01, **decay**:0	**TR:** 1.73 ,**VS:** 1.81, **TS:** 1.86± 0.1
DeepNN b	**units**:7, **depth**: 3, **dropout**: 0, **lr**:0.01, **decay**:0	**TR:** 1.98, **VS:** 2.1, **TS:** 2.23±0.13
RandNN	**units**: 7, **depth**: 2, **dropout**: 0, **lr**:0.01, **decay**:0	**TR:** 3.18, **VS:**3.17,**TS:** 3.23±0.04
CNN	**units**:20, **lr**:0.001, decay:0	**TR:** 2.24,**VS:**2.23, **TS:**2.37 ±0.07
Cascade Correlation	**units**:37	**TR:** 1.86,**VS:** 1.87 **TS:** 1.93 ± 0.05
Linear Regressor Pytorch	**lr**:0.1	**TR:** 2.19, **TS:**2.20 ± 0.04
Shallow Pytorch	**lr**:0.01, **units**: 4	**TR:** 2.23, **TS:**2.25 ±0.04

Table 3: Best performances obtained for the ML-CUP dataset.

- **Computing time:** for regression notebook 1h 15'± 15'.
  CPU: 16-core Apple M1 Chip
  Memory 16 GB DDR4

- **Final CUP model:** The main choosing criterion for the final model is the MEE score on the test set with the relative standard deviation acquired in the model assessment phase. However, we wanted to find a trade-off between generalization capability and model complexity. That's why we generated Figure 4, having on the x axis the number of effective free parameters and on the y axis the model performance evaluated on the MEE metric.

  The model chosen for the CUP, is the deep NN (4 hidden layers, 7 units each) trained with minibatch (learning curve in Figure 5). Looking at the Figure 4, we could have picked KNN, since it has both the lowest complexity and the best performance score. Since this model is instance-based (relies on memory) we cannot be sure about its generalization capability over a blind test set (further details in bullet-point "Discussion"). For this reasons, we decided to select the deep NN model, even though the MEE score was slightly higher. Namely, we expect this model to perform more reliably on unseen data.

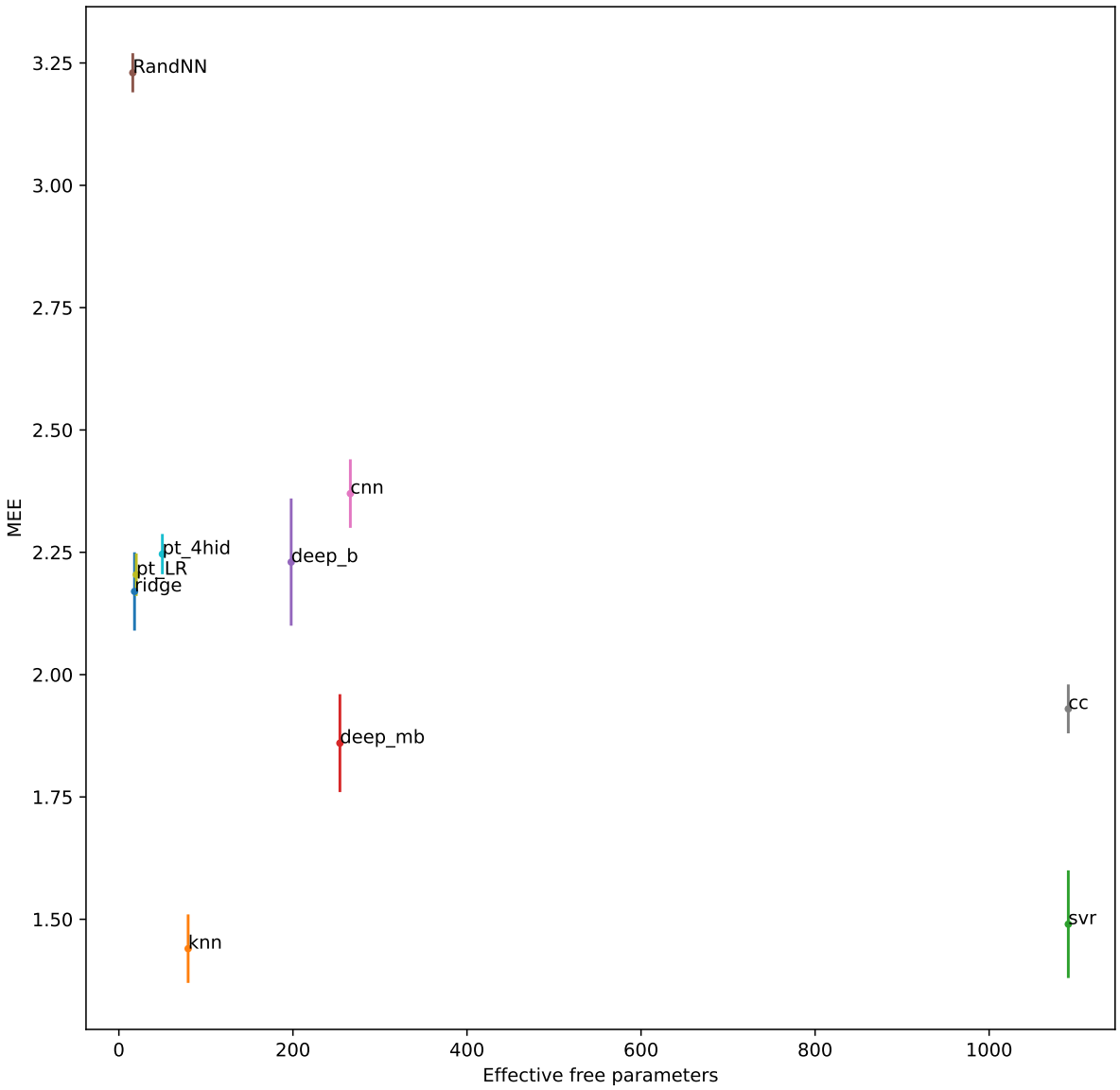

Figure 4: Final plot

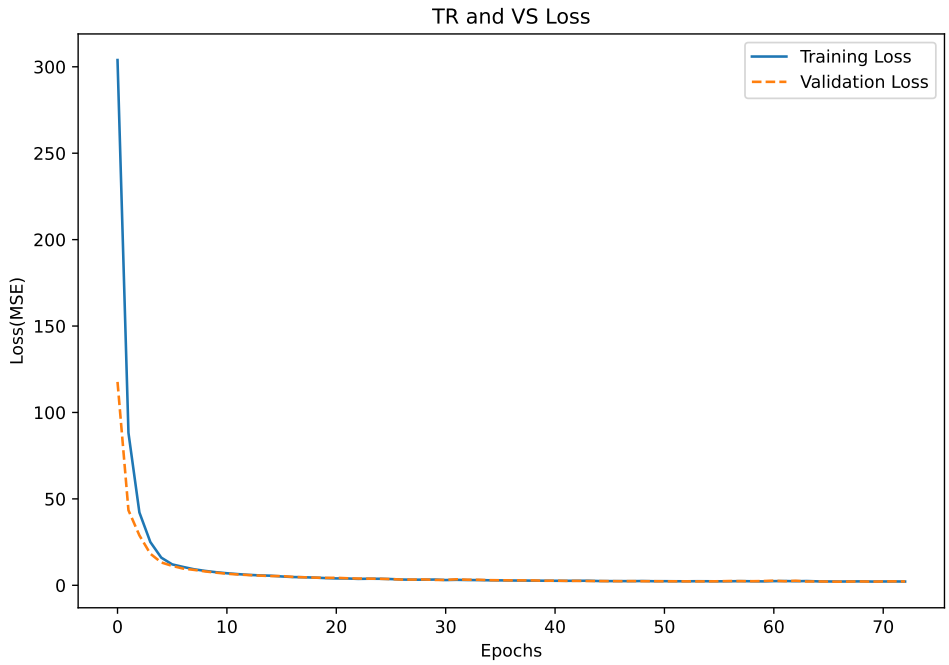

Figure 5: ML-CUP Deep NN with minibatch learning curve

- **Discussion:** Since the beginning, our aim has been to understand how the performance of a model is impacted by its complexity. That's why we chose to explore different architectures with a different number of effective free parameters. The main lesson that this analysis has taught us is surely that not always a complex model is the best solution. Practically, out first (naive) NN implementations had many units and layers, but we quickly realized that decreasing the model's complexity didn't penalize significantly the performance. Moreover, training and prediction time has been a relevant aspect in evaluating the models: not always a longer execution time is equivalent to a better performance. In addition, there is a high variability of training duration among the explored models: it ranges from less than a second to more than ten minutes.
  Another impressive result was the capability of exploiting random initialization to build an effective NN (RandNN) with a significantly smaller number of trainable parameters compared to its fully trainable counterpart. Namely, we were surprised to observe that RandNN's and deepNN's performances were comparable.
  We were stunned to see that a model such as KNN has such great results, even though this technique doesn't require any training. This fact might imply that neighbors are reliable to predict the target. This reasoning is stressed by the fact that SVR's grid search always chooses *RBF* over other kernels. On the other hand, with a model like KNN, we risk to have a poor generalization capability over new data.
  Surprisingly, the CNN architecture worked quite well. This might imply the existence of some kind of pattern-like structure in our dataset but, because of the black-box nature of NN, we don't really know what the model has learnt.

4 Conclusion

To summarize, each model has its upsides and downsides: there's no absolute best model. Our job has been to select one based on our data, knowledge and aim.
In our case, we preferred a more reliable model over a memory-based one, regardless of an higher MEE score.

Blind test results: NoFreeLunch_ML-CUP20TS.csv
Team name: NoFreeLunch