Machine Learning Project

Authors (*Lorenzo Mannocci, Giuseppe Valentini*).

Data Science And Business Informatics

mannocci.lore@gmail.com valentinigiuseppe10@gmail.com.

ML course (654AA), Academic Year: 2020-21

Date: 23/01/2021

Type of project: **B**

Abstract

The purpose of this report is to describe the project related to the Machine Learning course. We use different libraries (Keras, PyTorch, sklearn. . .), in order to compare the same model. For each task a GridSearch is carried out, trying to minimize the number of trials, choosing the parameters to be tested in a smart way. We mainly use a KFold CV as a validation schema and a final retraining for the best model over the entire training.

1 Introduction

The goal of this report is to find the best model for each task (MONK's dataset and ML Cup), searching in a smart way for the best combination of parameters through Grid-Search. Furthermore, we try to compare not only different models, but also the same model, using different libraries, evaluating their performance in terms of metrics and execution time. We try different parameters, such as optimization algorithms (SGD, Adam, RMSPROP, . . .), weight initialization, activation function (relu, sigmoid, tanh. . .), learning rate, batches (stochastic, mini-batch, batch), momentum (Nesterov or not) and finally regularization when necessary to avoid overfitting.

2 Method

- Used libraries: Keras, Tensorflow, PyTorch, Sklearn, Pandas, Numpy, Matplotlib.
- Preprocessing: For the Monk's dataset, we do a one-hot encoding for all the attributes, obtaining 18 columns, including the target one. Instead of for ML-Cup, no preprocessing is needed.
- Validation schema: For the Monk's dataset, the test set has been already provided, so we work only on the training set. Instead for the ML Cup, we split the dataset provided in TR and TS (70-30%). However we use a StratifiedKFold CV for Monk

and a KFold CV for ML-Cup, with a number of folds equal to 5 for Monk and 3 for ML-Cup. In order to do this we use the method of sklearn library, which allows us to set the shuffle of records before the splitting. Just for KNN and SVM we use a hold out validation. Finally we retrain the best model over the whole training set.

- Preliminary trials: At first we create the custom loss function (Mean Euclidean Error), which has some problems in training after a certain number of epochs in Keras (all metrics are set to nan), so we decide to use MSE for training and still compute MEE as required in order to have a standard metric.
  For the ML Cup, we decide to use a single neural network model, which predicts both targets variables.

3 Experiments

3.1 Monk Results

The neural network takes 17 features as input and just with one hidden layer we obtain great results. The number of units for the layer for each task is indicated in table 1 . For the hidden layer, the activation function used is the relu, while for the output layer (1 neuron) it is the sigmoid function. We try, among other things, stochastic, mini-batch and batchsize, finally deciding, on the basis of the learning curves, for $mini-batch = 10$. The model is trained with the MSE, while the table 1 shows the MEE, in particular the average of a 5-fold cross validation for the training. Finally we try different weights inizitialization, using at the end a normal distribution (mean 0 and standard deviation 0.005) and zero bias.

For task Monk 3 we observe that (without regularization) the learning curves show overfitting and instability. Therefore we decide, for the second problem, to stabilize the descent of the gradient, inserting a momentum with value shown in the table 1, obtaining better results. Then, in order to avoid overfitting, we introduce a L2 regularization with the value shown in table 1.
The learning curves of loss and accuracy for the tasks are shown in table 2.

Task	Units	Optimizer	eta	momentum	lambda	MEE (TR/TS)	Accuracy (TR\TS)
MONK 1	4	Adam	0.001	-	-	0.009/0.0011	100%/100%
MONK 2	5	SGD	0.2	0.5	-	0.0107/0.0106	100%/100%
MONK 3 (no reg)	5	Adam	0.002	-	-	0.0156/0.0746	100%/93%
MONK 3	5	SGD	0.004	0.5	0.03	0.2134/0.2009	93%/97%

Table 1: MONK's tasks: hyperparameter and results

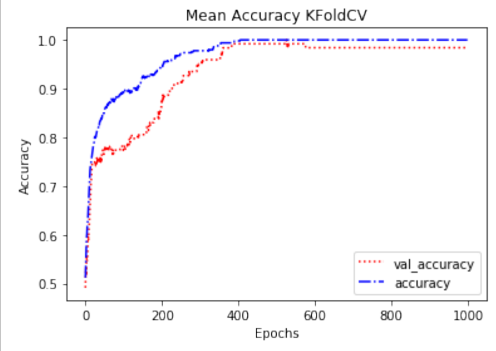
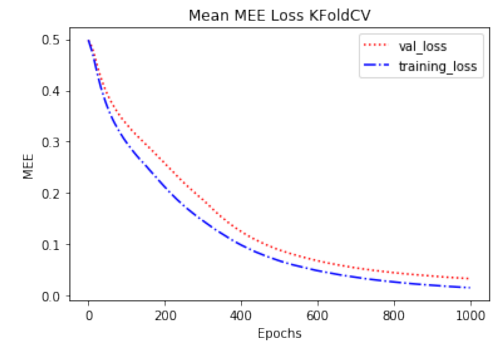
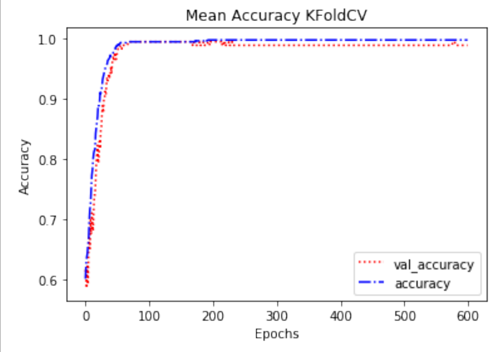
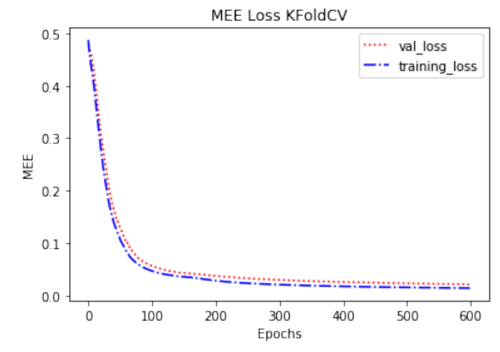
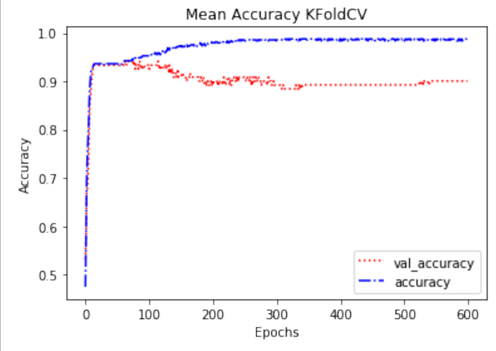
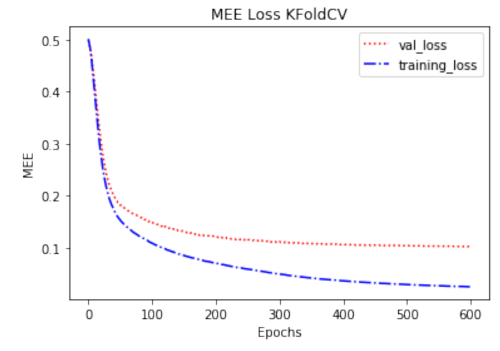
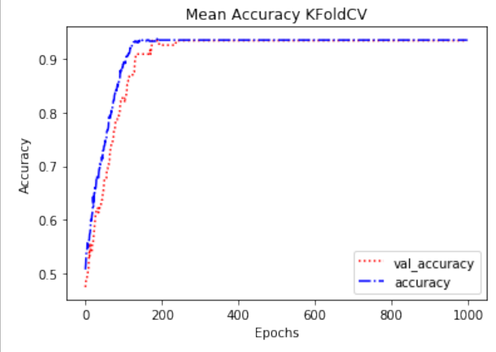
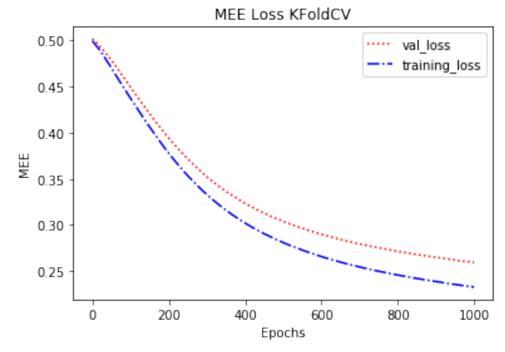
MONK 1		
MONK 2		
MONK 3		
MONK 3 REG		

Table 2: Monk learning curves

3.2 Cup Results

3.2.1 Validation schema

The *ML-CUP20-TR.csv* file is split in training (70%, 1066 records) and internal test (30%, 458 records). Then, we proceed with a K-fold CV with a fold number $k = 3$. Finally, the best model is chosen through a GridSearch, as explained in the following sections.

3.2.2 Preliminary trials

From preliminary trials, we exclude a high learning rate, which led to highly unstable learing curves, even with high momentum that fails to avoid oscillations.
Furthermore, a small initial GridSearch excludes a low momentum (even with small learning rate), so the values sought for this hyperparameter will be between 0.7 and 0.9.
Then, it is excluded that the initialization of the weights significantly influences the final results: in fact, insignificant changes are observed as the initialization of the weights varies (for the bias we try zeros(), i.e. all zero and ones(), i.e. all one; and for the weights, we try *normal* and *truncated_normal_distribution*). We proceed by establishing a zero distribution for the bias and a normal distribution with mean 0 and standard deviation 0.05. Finally, we exclude a batch algorithm, as worse results were obtained in terms of loss. While with an online algorithm, unstable learning curves is obtained (also with lower values of learning rate) and the training time also increases by a lot. It is therefore decided to fix this hyperparameter to a $mini\_batch = 10$.
The variant of the Nesterov momentum is also tested, observing that there are no significant variations in the results, so we initially set this hyperparameter to False.

3.2.3 GridSearch hyperparameters

We run multiple GridSearch, keeping the number of epochs fixed at 1000 in order to compare results in terms of loss. Moreover, a linear activation function has been set for the output layer.

N. Layer = 1	N. Layer = 2	N. Layer = 4
[50,75,100,125,150]	[32,64] [64,32]	[32,64,128,256] [256,128,64,32]

Table 3: Configuration for layers and units

We run three different independent GridSearch, one for each combination of number of layers/number of units as reported in table 3, varying in each GridSearch the hyperparameters reported in 4. As optimizer we use rmsprop and we don't use the momentum nesterov.

Learning rate	Range(0.0001, 0.0003, step = 0.00002)
Regularization	[0.0001, 0.0003]
Momentum	[0.7, 0.8, 0.9]
Activation function	[Relu, tanh]

Table 4: Hyperparameters range for the three GridSearch

From the results obtained by this three GridSearch, three further GridSearch are run on the best models for each number of layers, varying the parameters in table 5.

Optimizer	[rmsprop, SGD]
Nesterov	[True, False]

Table 5: Hyperparameters range for best models of first GridSearch

3.2.4 Best result grid search

The results reported in table 6, are the best for each number of layers (not absolutely). From the trials obtained from the GridSearch (even those not shown in table 6), it is observed that the variant of the Nesterov momentum does not significantly affect the results (the best results show both configurations). Furthermore, the best results of the GridSearch always have as activation function for hidden layer tanh (for this reason it is not showed in the table). Observing the learning curves for all the best models in the different configurations of layers and units, we note that there is a light overfitting, due to the number of epochs. For this reason we set the number of epochs, for the next phases, to 650 (after this epoch validation error starts increasing and training error does not improve anymore).

N. Layers	N. Units	L. Rate	Reg.	Momentum	Nesterov	Optimizer	MEE_TR	STD _TR	MEE_VAL	STD_VAL
1	100	0.00025	0.0001	0.7	True	SGD	2.5499	0.0481	2.9661	0.0837
1	125	0.00019	0.0	0.8	False	SGD	2.5438	0.0762	2.9656	0.1243
2	[32, 64]	0.00007	0.0003	0.7	False	rmsprop	2.5766	0.0078	2.8980	0.1115
2	[64, 32]	0.00007	0.0003	0.7	False	SGD	2.5577	0.0093	2.9499	0.1428
4	[256, 128, 64, 32]	0.00013	0.002	0.9	False	rmsprop	0.3754	0.0476	3.3944	0.0966

Table 6: Best result grid search for each configuration of N. Units

For the four-layer configuration we can observe low loss for training and high loss for validation: this means that the model is in overfitting. We try to increase regularization until a value of 0.6: in this way we reach a good learning curves, but loss drastically gets worse ($MEE\_TR = 16.8832$, $MEE\_VAL = 16.9015$). We can interpret this results, concluding that a four-layer configuration is a model too complex for this task, even if regularized.

3.2.5 Estimation computing time training

We report in table 7 the hardware resources used and the computing time in the retraing process of the best model with three different tools with 650 epochs ($N.Layer = 2, N.Units = [32, 64]$).

Processor	Ram	Tool	Time (seconds)
AMD Ryzen 7, 2.20 GHz, 4 Cores	8 Gb	Keras	95
Intel i7-6500U, 2.50 GHz, 2 Cores	16 Gb	Keras	89
AMD Ryzen 7, 2.20 GHz, 4 Cores	8 Gb	PyTorch	123
Intel i7-6500U, 2.50 GHz, 2 Cores	16 Gb	PyTorch	94
AMD Ryzen 7, 2.20 GHz, 4 Cores	8 Gb	MLP-sklearn	26
Intel i7-6500U, 2.50 GHz, 2 Cores	16 Gb	MLP-sklearn	24

Table 7: Computing time training

3.2.6 Compare different models and tools

We decide to compare the best model of the GridSearch ($N.Layer = 2, N.Units = [32, 64]$) in table 6 in Keras with the same model in PyTorch and MLPRegressor of sklearn. We obtaine the results in table 8.

Tool	MEE_TR	STD_TR	MEE_VAL	STD_VAL
Keras	2.5766	0.0078	2.8980	0.1115
PyTorch	2.9458	0.0568	3.1375	0.0941
MLP sklearn	2.5682	0.0478	2.9974	0.0478

Table 8: Results best model in three different tools

We can observe similar loss results for all three tools: PyTorch shows the highest loss in validation and highest computing time as shown in table 7.

We also try to use SVM for regression and KNN for regression. For this two models, we decide to do a hold out validation and to use the coefficient of determination R2 in order to do model selection. Moreover, we build two different models, one for target x and one for target y: because KNN and SVM predict just one variable at time, but MEE loss has been computed on both predictions.

- KNNRegressor: We run a GridSearch with different values of k ($range(2, 20)$), uniform weights and distance-based weights and different distance metrics: Manhattan, Euclidean and Minkowski with different values for the exponent in $range(3, 20)$. The best model with parameters and loss values is in table 9.

Target	K	Weights	Metric	MEE_TR	MEE_VAL
x	18	Distance-based	Euclidean	0	3.0130
y	7	Distance-based	Manhattan		

Table 9: Results for KNN models for target x and y

- SVMRegressor: We run a GridSearch with different values of regularization parameter C in $range(0.2, 1, step = 0.2)$, of paramter epsilon (epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value) in $range(0.1, 2, step = 0.1)$. We also try different kernel type between linear, polynomial (with degree in $range(1, 10)$), gaussian, hyperbolic tangent (sigmoid) with gamma in $range(0.1, 2, 0.1)$ and default value $scale = \frac{1}{(n\_features * X.var())}$ and $auto = \frac{1}{(n\_features)}$. The best model with parameter and loss values is in table 10.

Target	C	Epsilon	Kernel	Gamma	Degree	MEE_TR	MEE_VAL
x	1.8	0.3	rbf	scale	-	6.1992	7.6977
y	1.8	1.9	rbf	1.9	-		

Table 10: Results for SVM model for target x and y

3.2.7 Final model used for blind test

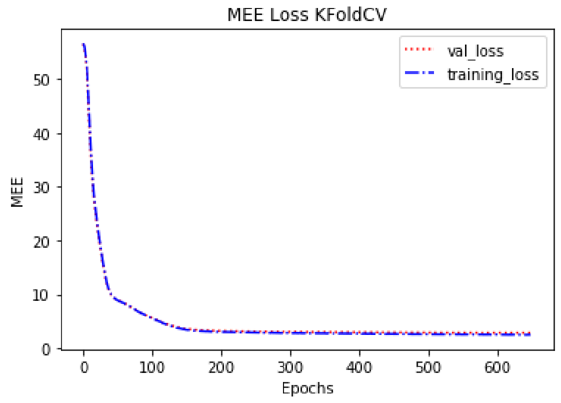
Finally, the final model used for blind test is the one with parameters shown in table 11 and results in table 12 obtained on training and validation. For the test, we decide to retrain 10 times and take the average on the test of these 10 different models, in order to minimize the influence of randomness in the initialization of the weights. The model was chosen among the best results in table 6 based on the MEE on validation. We choose Keras as tool for the final model because it has better results than PyTorch (and MLP-sklearn), although they do not vary too much.
We can observe learnig curves for the final model in figure 1; figure 1.b shows the zoomed plot on the vertical axis.

N. Layers	N. Units	Act. Func. Hidden	Act. Func. Output	L. Rate	Reg.	Momentum	Nesterov	Optimizer
2	[32, 64]	Tanh	Linear	0.00007	0.0003	0.7	False	rmsprop

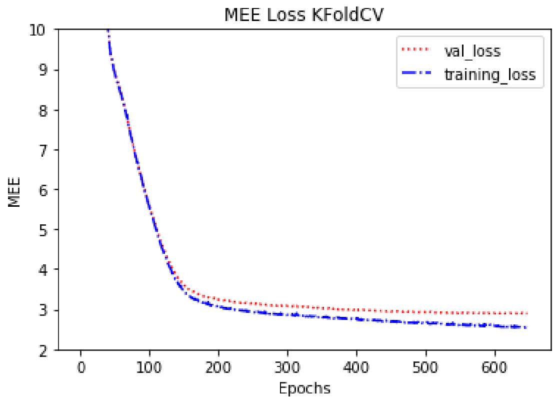
Table 11: Hyperparameters of best model used for Blind test

MEE_TR	STD_TR	MEE_VAL	STD_VAL	MEE_TEST	STD_TEST
2.5766	0.0078	2.8980	0.1115	2.8451	0.0209

Table 12: Results of best model used for Blind test



(a) MEE loss



(b) MEE loss zoomed

Figure 1: MEE loss best model

4 Conclusion

The development of this project made us understand the importance of the theory and knowledge of the different hyperparameters, a key prerequisite for their critical and conscious use. We used different optimizers with default values: in fact we wanted to try different optimization algorithms anyway, but a possible continuation of the project would have been to try different configurations of these advanced optimizers. The most critical phase is the selection of hyperparameters, for which we spend most of the time and computational resources: this justifies our choices for the GridSearches described in section 3.2.3. We have assumed possible values of some hyperparameters from the preliminary trials, sometimes fixing them and we are aware that this is not the right way to proceed, but for time constraints we were obliged to search a trade off between the right way and a plausible way, fixing some parameters. However we are very satisfied with the work done as we were able to compare different tools and models.

BLIND TEST RESULTS:

- Name result files: lore_giuse_ML-CUP20TS.csv
- Nickname: lore_giuse

Agreements

We agree to the disclosure and publication of my name, and of the results with preliminary and final ranking.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[3] A. Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] S. Haykin. *Neural Networks and Learning Machines, 3/e*. PHI Learning, 2010.

[6] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.