Machine Learning 2020/21 Project Report

Andrea Laretto (619624) – a.laretto@studenti.unipi.it
Pier Paolo Tarasco (619622) – p.tarasco@studenti.unipi.it
Master Degree in Computer Science

ML course (654AA), Academic Year: 2020/2021
Date: 09/07/2021
Type of project: **A**

Abstract

We developed a multi-layer feed-forward NN from scratch using Python and NumPy. For the ML-CUP we employed an ensemble of 8 neural networks combining 4 models trained with Adam and 4 models with SGD. The two sets of models were selected with two separate grid searches and then refined with a random search with 15% variation on each parameter of the 8 models. For both searches we used 5-fold as CV technique. After splitting an internal test set composed of 20% of the original dataset, we also centered the mean of all training data features.

1 Introduction

In this report we present an extendible multi-layer feed-forward neural network architecture written from scratch using Python and NumPy. The architecture implements (stochastic) gradient descent, grid search, random search, holdout and k-fold cross validation, L2 regularization, classical and Nesterov momentum, learning rate decay, along with the Adam learning algorithm [3]. We report here the results obtained by using the framework to build models for the MONK's dataset [4] (a classification task) and the ML-CUP-2020 dataset (a regression task).

2 Method

The neural network architecture is extensible and developed in a modular way, allowing the user to try different hyperparameters and model characteristics by just few changes in the high-level specification of both how the grid search is performed (e.g.: k-fold or holdout, grid search or random search) and what kind of models should be experimented with (e.g.: how many layers and units, which activation functions and hyperparameters). This also allowed us to try configurations and run different experiments quickly and with ease, and the modularity gave us the freedom to add new features and extensions to the architecture without hassle.

2.1 Framework features

The main algorithms were implemented by extensively exploiting the matrix multiplication and array manipulation capabilities offered by NumPy. For example, we do not consider single data patterns during the training, and treat the input data as entire matrices during backpropagation. This increased the efficiency of the network and came at a higher development cost at the beginning, but it paid off in the long term by allowing us to explore higher training epochs, trying larger models, and testing a higher number of hyper-parameters in the search. We used the `multiprocessing` standard Python library to exploit the intrinsic parallelism of the grid/random searches, as to utilize all the cores provided by our machine and test multiple hyperparameter configurations at a time. Aside from NumPy, we employ no other helper libraries and we have implemented and tested every data processing procedure (splitting, data standardization) from scratch. We present here a brief list of the features provided by our architecture:

- Standard (stochastic) gradient descent with adjustable mini-batch percentage
- k-fold and holdout cross validation
- Adam with adjustable decay rates
- Activation functions provided: *tanh, sigmoid, relu, leakyrelu, linear*
- Mean Squared Error, Mean Euclidean Error, Cross Entropy
- L2 (ridge) regularization
- Classical and Nesterov Momentum
- Linearly decaying learning rate with adjustable decay rates
- Weights initialization: gaussian range, uniform range, normalized initialization heuristic by Glorot, Bengio (2010) [1] (i.e.: uniformly generated in [-a,+a], where $a = \sqrt{6}/\sqrt{\text{fan-in} + \text{fan-out}}$)
- Parallelized grid search with lists of possible hyperparameter values
- Parallelized random search with range of hyperparameter values

The network implements Early Stopping by training until the "`max_epochs`" provided and continuously updating the minimum validation error obtained throughout the epochs. The best epoch reported is then the one where the validation error is minimum. We used a "`max_unlucky_epochs`" parameter in order to prematurely stop the training for already bad-performing networks, thus saving time and increasing the overall efficiency. This is simply used as the maximum number of epochs for which the validation is allowed to increase before reaching a new minimum.

On a more technical note, we remark here how the gradient applied on the network has been normalized by dividing it with the batch size.

2.2 Preprocessing

The MONK's datasets were preprocessed by applying the one-of-k encoding to all three tasks. Compared with the original 6 attributes, we thus obtained 17 binary attributes in the range (-1,+1) since we opted to use the *tanh* activation function. No separate internal test set was split, as it was already provided for the problem.

For the CUP problem, we shuffled the dataset at the beginning of the experimentation and then split it into a separate 20% of internal test set, which we kept in a physically separate file. Then, we calculated the mean for each feature of the remaining 80% training set, which we used to standardize the dataset so that both input and output features had mean $\mu = 0$. These standardization values do not take into account the mean of the entire dataset including the test set, but only that of the training set after the split. We also tried standardizing the standard deviation, but found with some preliminary experiments on the CUP that it sometimes slightly worsened the performance of the models on the validation set, while centering just the mean improved the results compared with no standardization at all. We performed no further feature selection on the dataset.

2.3 Validation schema

For the MONK's dataset, we decided as cross validation technique to apply the *k-fold* approach since the data provided was somewhat limited, using $k = 5$ folds as a compromise between accuracy and computational cost.

We chose again to use for the CUP dataset the *k-fold* cross validation technique with $k = 5$ folds, since we preferred to have a better estimation of the quality of the hyperparameters, despite the higher computation cost compared with a simpler holdout. Just for the CUP, we also applied a refined random search around the best hyperparameters chosen by the first grid search by uniformly perturbing each parameter with a max 15% variation, again using 5-fold. For both problems, we performed a final retraining of the networks on the entire training dataset, with the final retraining epoch chosen as the average of the k best epochs founds by the k-fold.

2.4 Preliminary trials

For both datasets, we first manually experimented with some extreme ranges in order to both establish a general range to explore with the grid search. This was both to narrow down the search range and to qualitatively test that the parameters of the networks gave predictable results that matched our expectations based on the theory, e.g.: unstable learning curves with too high learning rates and low mini-batch size, trying to overfit the task with too many units, underfitting with too high regularization parameters, etc. Within the grid searches of all experiments, we tried to make sure that the hyperparameters chosen were not at the extremes of the search: in these cases, we repeatedly shifted the ranges as to correctly guide the search towards better parameters, while refining the search around the values that were picked more frequently.

3 Experiments

3.1 MONK Results

For the MONK's tasks, we decided to use *batch gradient descent* as it gave smoother learning curves with a better gradient estimation. Since the MONK's is a classification task, we opted to use the *Cross-Entropy* error function to train the models, and thus kept the *logistic function* for the output unit. For the hidden layers, we chose for all three tasks the *tanh function* as activation function. We selected 300 epochs as training upper bound for all the three problems, with the final retraining epoch chosen by averaging the best epochs (with minimum validation error) of the 5-fold.

We first experimented with very small learning rates (from $\eta = 0.01$ to 0.09) which gave smooth but slowly converging curves. The grid search however kept choosing the higher learning rates, while still giving smooth curves on these higher values. We thus chose to run the final grid search shown in Table 1 and Table 2 between the ranges $\eta = 0.1$ to 1.0. We also noticed that the learning curves were very sensible to different weight initializations, with different trends even with the same hyperparameters. For this reason, we chose for all MONK's problems a standard weight initialization that did not have a too large variation range, and we chose *uniform range* (-0.25,0.25) as initialization.

		Configurations searched	
Hyperparameter		**MONK1**	**MONK2**
Network topology		(17,3,1), (17,4,1), **(17,5,1)**	(17,3,1), (17,4,1), **(17,5,1)**
Learning rate	η	0.1, 0.2, 0.4, **0.6**, 0.8, 1.0	0.1, 0.2, 0.4, **0.6**, 0.8, 1.0
Classical momentum	α	0.0, 0.2, 0.4, **0.6**, 0.8, 0.9	0.0, 0.2, 0.4, 0.6, **0.8**, 0.9
L2 regularization	λ	**0.0**	**0.0**

Table 1: Grid search explored on the MONK1 and MONK2 tasks with 5-fold.

		Configurations searched	
Hyperparameter		**MONK3**	**MONK3 (with L2 reg.)**
Network topology		(17,6,1), **(17,9,1)**, (17,12,1)	(17,6,1), **(17,9,1)**, (17,12,1)
Learning rate	η	0.1, 0.2, **0.4**, 0.6, 0.8, 1.0	0.1, 0.2, 0.4, 0.6, **0.8**, 1.0
Classical momentum	α	0.0, 0.2, 0.4, **0.6**, 0.8, 0.9	0.0, 0.2, 0.4, **0.6**, 0.8, 0.9
L2 regularization	λ	**0.0**	1e-5, 1e-4, **1e-3**, 1e-2

Table 2: Grid search explored on the MONK3 task with and without regularization.

In Figure 1 to Figure 4, we report the learning curves of the final retraining on the entire dataset, using the hyperparameters chosen by the grid search. We show in Figure A.1 to Figure A.4 the learning curves for each fold of the chosen 5-fold. The final MONK's results are reported in Table 3, with the accuracies and errors being averaged out over 5 reinitializations. Since the retraining was performed with 5 different reinitializations, we only show the curves of the first trial for displaying purposes.

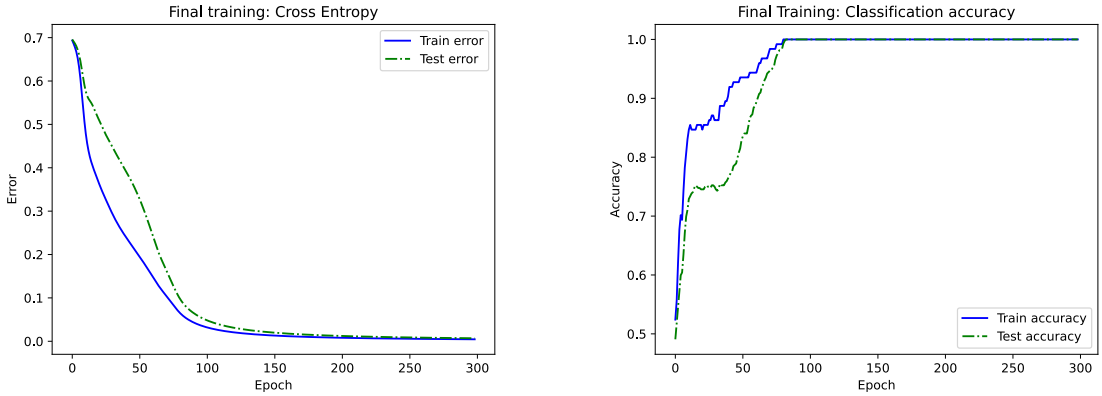

Figure 1: Learning curves of the final retraining for the MONK1 problem.

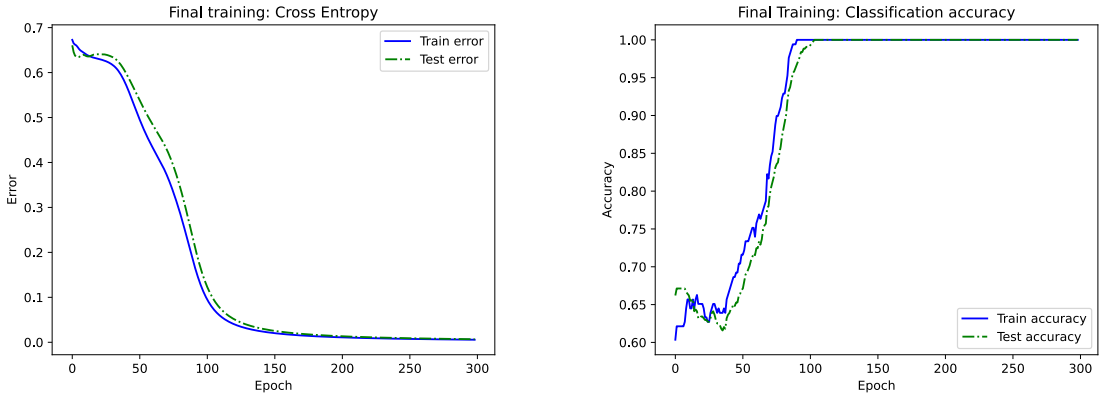

Figure 2: Learning curves of the final retraining for the MONK2 problem.

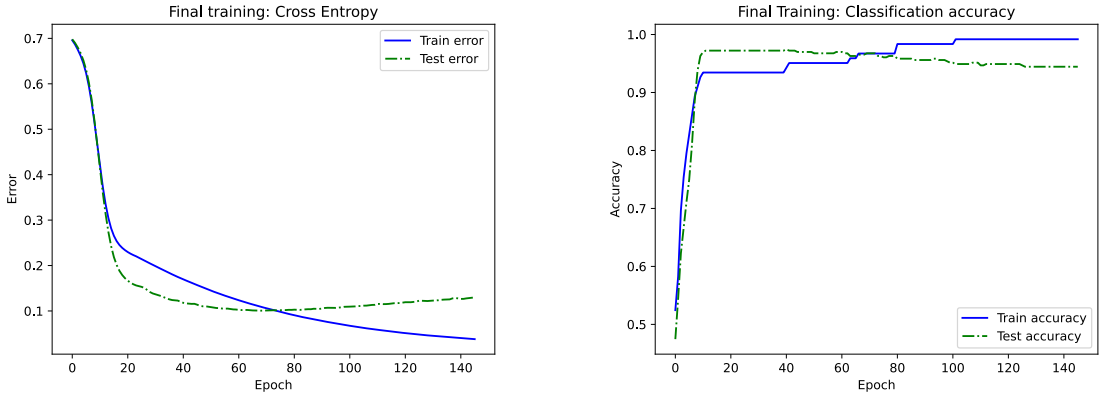

Figure 3: Learning curves of the final retraining for the MONK3 problem (no reg.)

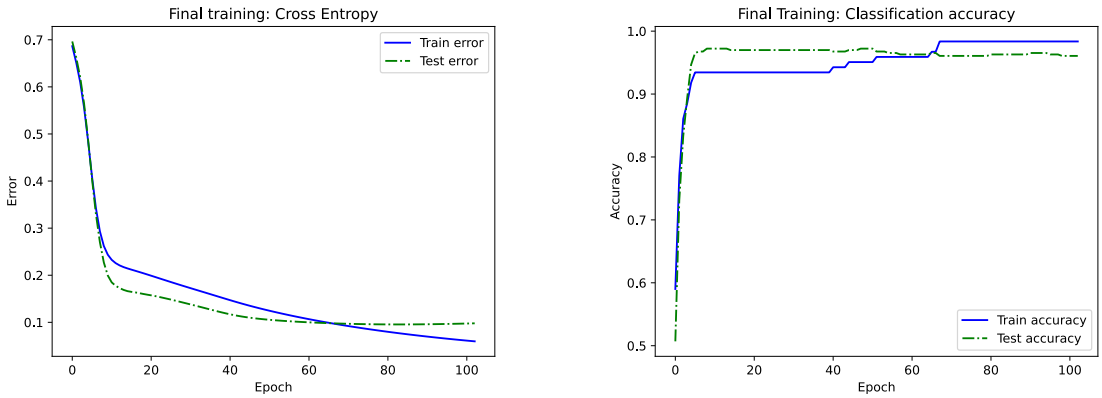

Figure 4: Learning curves of the final retraining for the MONK3 problem (L2 reg.)

Task	Units	Epochs	η	α	λ	MSE (TR/TS)	Accuracy (TR/TS)
MONK1	5	300	0.6	0.6	0.0	1.244e-04/7.691e-04 σ^2 =1.689e-09/8.944e-07	100%/100% σ^2 = 0.0 /0.0
MONK2	5	300	0.6	0.8	0.0	2.865e-05/3.973e-04 σ^2 =1.391e-10/3.342e-07	100%/100% σ^2 = 0.0 /0.0
MONK3	9	147	0.4	0.6	0.0	1.179e-02/4.205e-02 σ^2 =1.580e-06/2.106e-06	99.34%/94.36% σ^2% = 0.376 /0.265
MONK3+reg.	9	106	0.8	0.6	0.001	2.093e-02/2.715e-02 σ^2 =3.171e-06/1.594e-06	97.86%/96.29% σ^2% = 0.161 /0.107

Table 3: Average prediction results obtained for the MONK's tasks over 5 trials.

3.2 Cup Results

For the CUP, we chose to use MSE as loss function, while reporting MEE in all tables and graphs. As a preliminary survey of the problem, we manually explored a set of learning rates and hyperparameters with a very broad but small grid search using holdout with 20% validation set, in order to systematically analyze the learning curves and get a general idea of the magnitude and training quality of the parameters. We found it especially useful at the beginning to visually display the outputs of our model and compare them with the training targets, especially to check for models in considerable underfitting.

After experimenting with standard Gradient Descent, we chose to implement and try to apply the Adam learning algorithm to the problem, which provided satisfactory results on both training and validation. Thus, we chose to combine these two already well-performing methods by using an *ensemble* approach with 8 models. We chose to select the best 4 models trained with Adam and the best 4 models with Gradient Descent in order to combine different models and diversify the ensemble.

To select the best models of the two learning algorithms, we ran two main bigger grid searches with 5-fold as cross validation, and we applied some of our initial findings to narrow down the useful value ranges to explore with the grid searches. We also found that on the CUP using Gradient Descent with Nesterov Momentum improved with respect to Classical Momentum. Furthermore, we chose to explore a smaller range of mini-batch sizes with Adam, while exploiting the full batch for Gradient Descent since we used Nesterov Momentum. For both learning algorithms, we initially tried experimenting with bigger and deeper models such as (128, 64), (128, 128), (64, 64, 32) hidden units, but in terms of validation error they were not giving a significant advantage against smaller (32, 32) networks, especially with respect to the higher computational cost. So for both this and efficiency reasons we chose to restrict the grid searches to smaller networks. For the grid search, we wanted to test both *relu* and *tanh* as activation functions, while using the *linear* activation function for the output units of all models. Finally, we tried as weight initialization both uniform (-0.7,+0.7) and the normalized initialization heuristic [1], with the latter one already producing good results in our initial experiments.

	Configurations searched	
Hyperparameter	**Adam**	**Gradient Descent**
Hidden layers	(16,16),(32,32),(16,16,16)	
Minibatch size (%)	20%, 30%	100%
Learning rate η	5e-4, 1e-3, 5e-3, 1e-2, 5e-2	5e-4, 1e-3, 4e-3, 6e-3, 8e-3, 1e-2
L2 regularization λ	0.0, 5e-6, 1e-5, 5e-5, 1e-4, 5e-4	
Nesterov momentum α	–	0.0, 0.25, 0.5, 0.75
Decay rate 1 β_1	0.8, 0.95	–
Decay rate 2 β_2	0.9, 0.999	–
Activation functions	*tanh*, *relu*	
Weights initialization	Normalized initializ. heuristic, Uniform (-0.7,+0.7)	
Max epochs	1200	

Table 4: Grid searches explored for Adam and Gradient Descent for the CUP.

ID	**Hidden L.**	**Activ.**	**Epochs**	**Batch%**	η	λ	β_1	β_2	**TR (MEE)**	**VL (MEE)**
1	[32, 32]	*tanh*	415	30%	4.794e-03	1.068e-05	0.8458	0.9169	2.450144	2.830357
2	[32, 32]	*tanh*	400	20%	1.453e-03	1.107e-05	0.8000	0.9990	2.484119	2.851004
3	[32, 32]	*tanh*	568	20%	4.588e-03	9.513e-06	0.9786	0.9988	2.506079	2.821136
4	[16, 16]	*tanh*	534	20%	1.042e-02	1.000e-05	0.9796	0.9990	2.533020	2.849910

Table 5: Grid search + random search results for Adam.

ID	**Hidden L.**	**Activ.**	**Epochs**	η	λ	α	Weights Init.	**TR (MEE)**	**VL (MEE)**
5	[32, 32]	*tanh*	1091	3.734e-03	0.0	0.5379	Unif. [-0.7,+0.7]	2.564800	2.841297
6	[32, 32]	*tanh*	1048	4.318e-03	1.090e-05	0.5025	Norm. Init.	2.560759	2.864258
7	[16, 16]	*tanh*	993	4.178e-03	1.371e-05	0.5886	Unif. [-0.7,+0.7]	2.620305	2.922085
8	[32, 32]	*tanh*	1144	4.368e-03	9.230e-06	0.2046	Unif. [-0.7,+0.7]	2.617560	2.879980

Table 6: Grid search + random search results for Gradient Descent.

In the two grid searches we searched 2880 and 1728 hyperconfigurations for Adam and Gradient Descent, respectively. We report the configurations and values tested in Table 4. Then, we sorted both results in terms of validation error and performed a subsequent random search around each of the best 4 configurations, using 36 random perturbations each with 15% variation for each hyperparameter. The best models given by the final random search for both Adam and Gradient Descent are shown in Table 5 and Table 6, with the validation learning curves of the k-fold given in Figure A.5. Finally, the 8 models were all retrained on the entire training set with the epochs chosen according to subsection 2.3, for which we show the final learning curves in Figure 5. We then obtained the final output for the internal test and the blind CUP by averaging the constituent models' outputs, which we graphically show in Figure A.6 to Figure A.8.

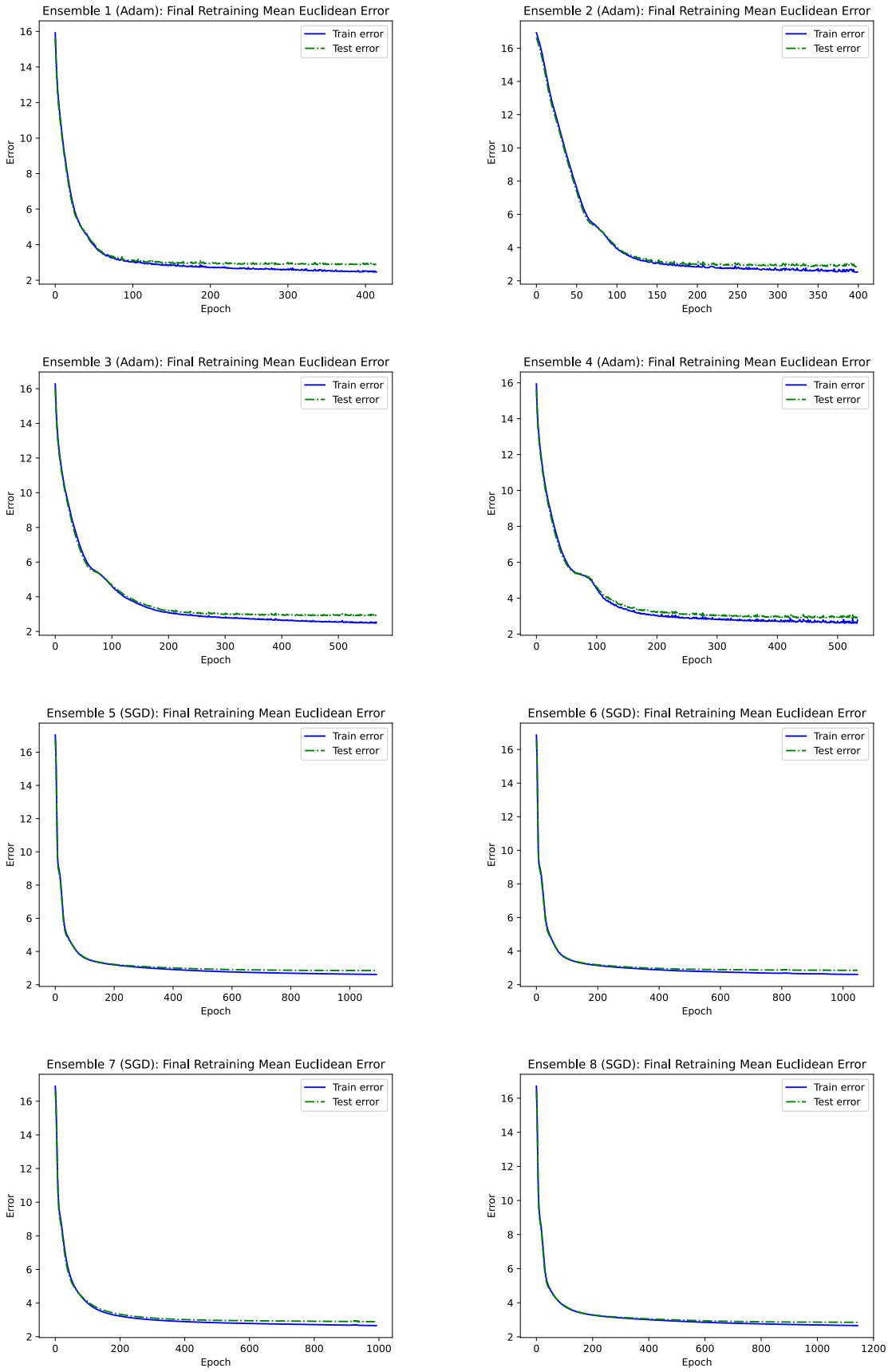

Figure 5: Final retraining learning curves for the 8 ensemble models.

ID	Learning algorithm	Development Set (MEE)	Test Set (MEE)
1	Adam	2.460874	2.876456
2	Adam	2.531615	2.852701
3	Adam	2.529028	2.982597
4	Adam	2.782404	3.034136
5	Gradient Descent	2.617873	2.852938
6	Gradient Descent	2.608901	2.851556
7	Gradient Descent	2.663224	2.900504
8	Gradient Descent	2.661741	2.855027
Ensemble	–	2.511514	**2.810081**

Table 7: Final training and test results for the retrained models and the final ensemble.

The grid searches and final retraining produced interesting results. First, we noticed that *tanh* was chosen for all the models rather than *relu*: we later confirmed this fact by rerunning a smaller grid search, which also showed more stable learning curves for *tanh*. Secondly, the deeper topology with 3 hidden layers (16,16,16) unfortunately was never selected, thereby confirming our preliminary findings about deeper models and possibly the need for a more specific choice of hyperparameters. All the Adam models in Table 5 selected the normalized initialization for initializing the weight, with Gradient Descent preferring the uniform set range. The learning curves produced by the models trained with Adam still showed some irregularities, especially compared with the Gradient Descent ones. This phenomenon can partly be attributed to the low mini-batch percentage, along with the high *decay rates* selected by the random search. We found in our tests, however, that this irregular behaviour often led the network to lower training and validation error (which the grid search also kept choosing), despite producing worse learning curves. Finally, we were very pleased to see that combining the different models in an ensemble substantially reduced both the training and the internal test set results.

We performed the training of all networks on a Intel Core i7-10750H 2.60GHz with 6 cores, for which we report the training time statistics in Table 8.

Task	Time	
Avg. time per epoch, topology (17,5,1) with $n = 124$ patterns	≈ 490	microseconds
MONK1 grid search + retraining	93	seconds
MONK2 grid search + retraining	111	seconds
MONK3 grid search + retraining	139	seconds
MONK3 reg. grid search + retraining	380	seconds
CUP grid search (Adam)	≈ 7	hours
CUP grid search (Gradient Descent)	≈ 5	hours
CUP grid search (total)	≈ 12	hours

Table 8: Training performance statistics tested on our machine.

4 Conclusion

We were very curious to try and implement the Adam learning algorithm, given its popularity and reputation among practitioners. Especially compared to Gradient Descent, controlling the behaviour and quality of the Adam curves proved to be more complex due to the more sensitive and higher number of parameters in the algorithm.

From the scatter plot results, we suspect that the model might still be in the underfitting region since there seem to be outliers and difficult to approximate extreme points on both sides of the graph. With the help of other regularization techniques such as dropout and batch normalization, we could have explored bigger and deeper models which we chose not to explore in detail in the search.

We estimate the final test error of the ensemble model to be **2.810081**, expressed as Mean Euclidean Error. The final predictions on the blind test were saved in the file `lambda00_ML-CUP20-TS.csv`. The name of our group is `lambda00`.

Acknowledgments

We believe this hands-on experience was both fun and extremely useful from a practical and pedagogical point of view, and it allowed us to truly appreciate the theory of Machine Learning at work and witness its concrete effects by getting our hands dirty.

We agree to the disclosure and publication of our names, and of the results with preliminary and final ranking.

References

[1] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In Y. W. Teh and D. M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence, Statistics, AISTATS 2010*, volume 9 of *JMLR Proceedings*, pages 249–256, 2010.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015*.

[4] S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, B. Roger, H. Vafaie, W. Van de Velde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's Problems: A Performance Comparison of Different Learning Algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

Appendix A

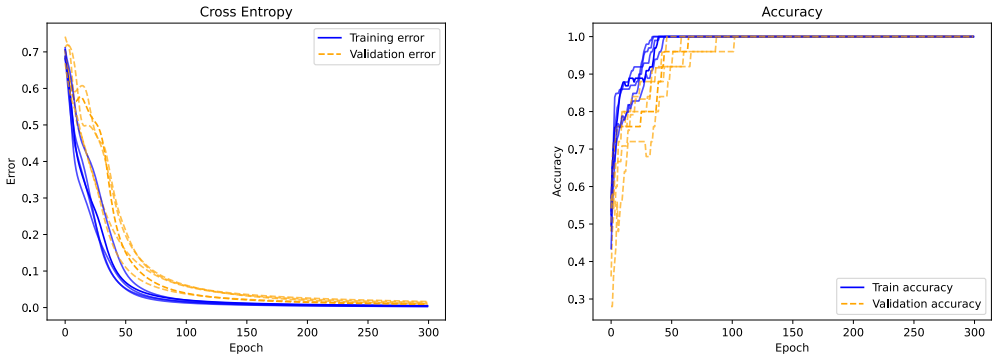

Figure A.1: Grid search learning curves for the MONK1 problem. (5 folds)

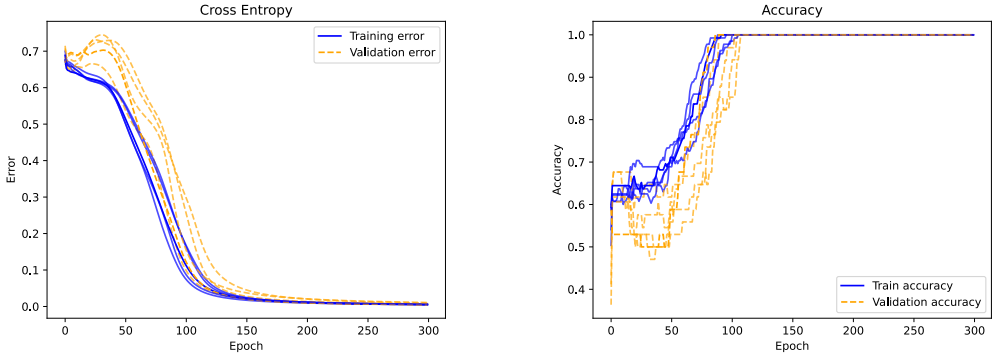

Figure A.2: Grid search learning curves for the MONK2 problem. (5 folds)

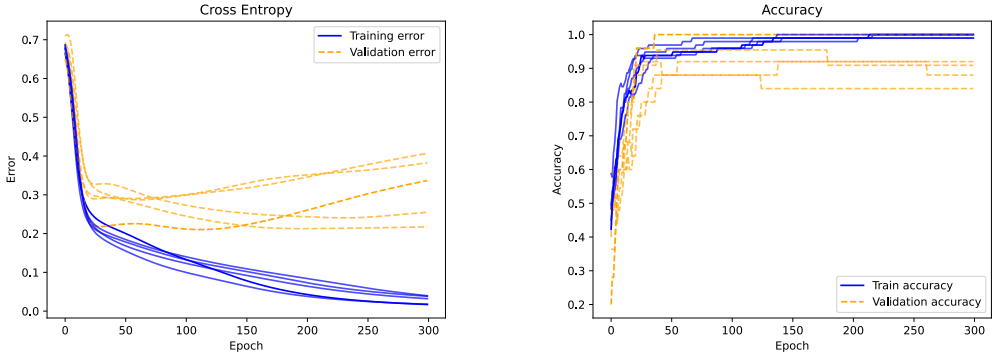

Figure A.3: Grid search learning curves for the MONK3 problem (no regulariz.)

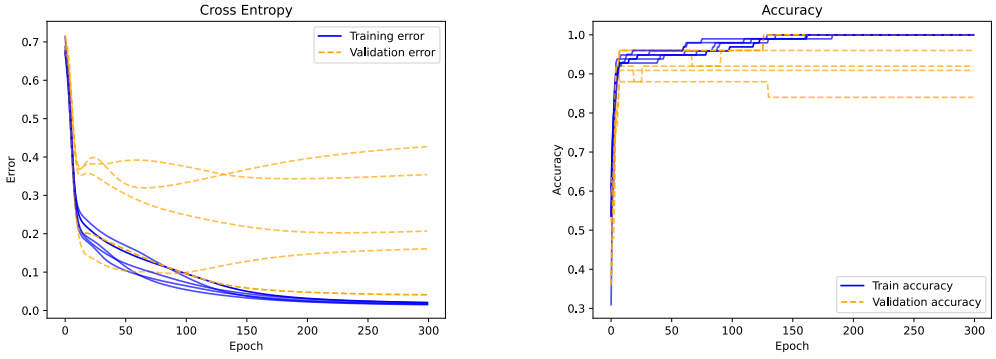

Figure A.4: Grid search learning curves for the MONK3 problem (L2 regulariz.)

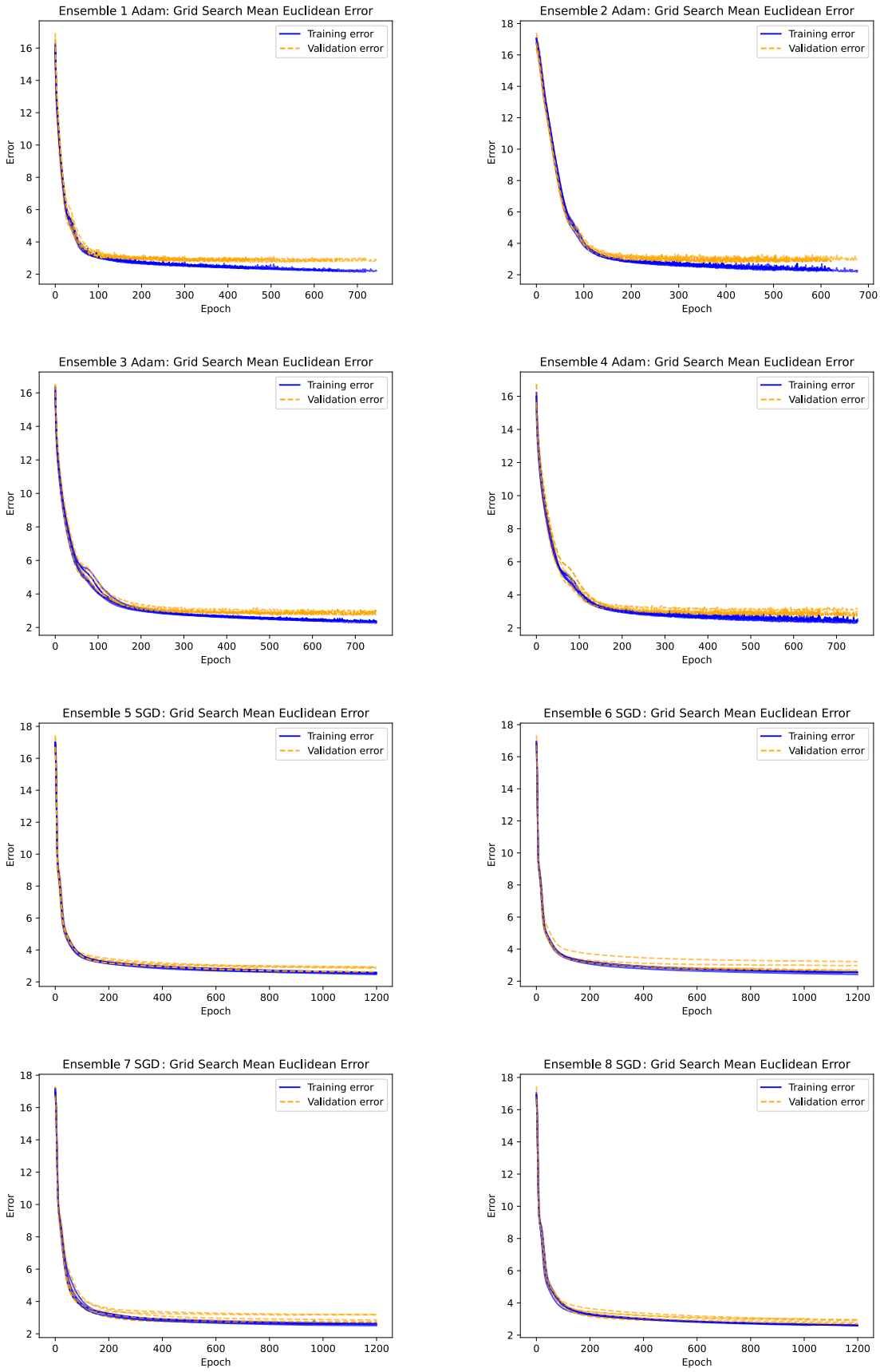

Figure A.5: CUP grid search learning curves for the 8 models (5 folds).

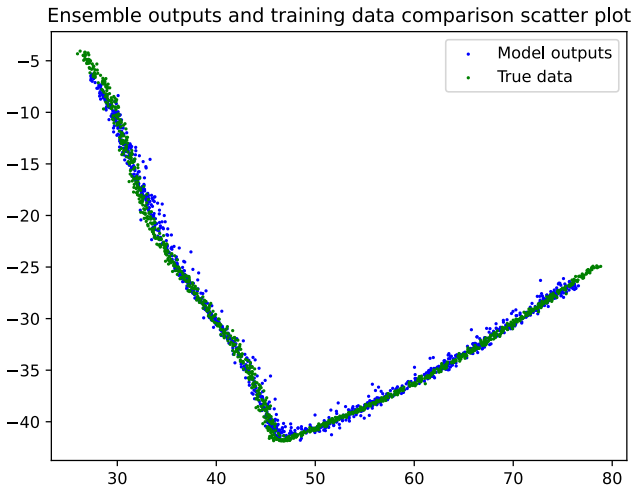

Figure A.6: Final ensemble output for the training set.

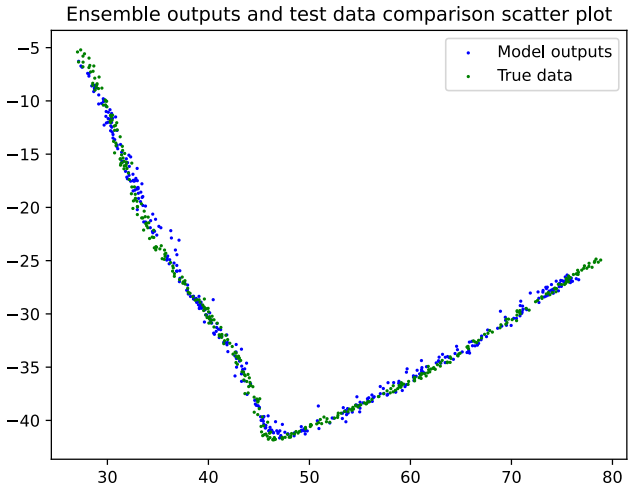

Figure A.7: Final ensemble output for the test set.

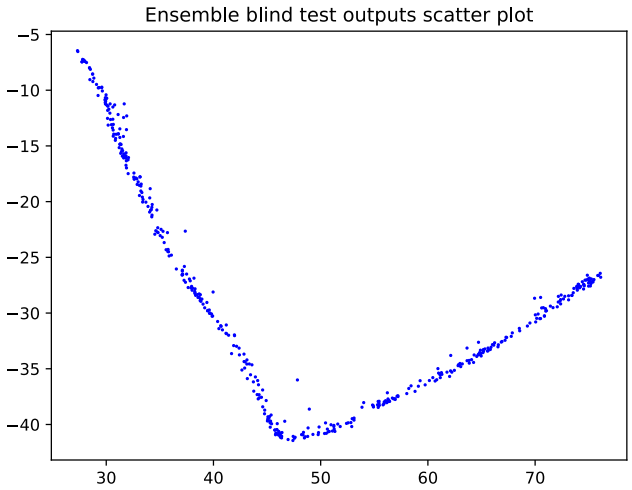

Figure A.8: Final ensemble output for the blind set.