Machine Learning Project Report

Authors (*Roberto Esposito, Marco Mazzei, Antonio Zegarelli*).
Master Degree in Computer Science - Artificial Intelligence, Big Data and Software.
Email: r.esposito8@studenti.unipi.it - m.mazzei13@studenti.unipi.it - a.zegarelli@studenti.unipi.it
ML course (654AA), Academic Year: 2021-2022
Date: 04/01/2022
Type of project: **A**

Abstract

The following report illustrates an implementation of a framework for building neural networks. The programming language used is *Python*. After exploiting thousands of configurations thanks to the grid search method using k-fold cross-validation, we test our code on the monk dataset. Then we apply it to the ML Cup dataset through an ensemble of 10 models.

1 Introduction

The goal of this project consists in developing, from scratch, a framework for building neural networks able to solve either classification or regression tasks.
The project wants to use a rigorous method to select the best configurations and then validate them. In order to find the best configurations we develop a grid search using k-fold cross-validation.
The performances of the models for classification tasks were evaluated by looking at the "Monk's Problem" [1]. Afterward, we evaluate the regression models for the given "ML-CUP-21" dataset.

2 Method

We implemented a multi-layer fully connected feed-forward Neural Network that learns its weights through backpropagation using the Stochastic Gradient Descent (SGD) over multiple epochs, we also implemented some additional features, reported in detail in **subsection 2.2**, that can be used to obtain different models with different performances. Instead, **subsection 2.3** describes how the best models were selected for "ML-CUP-21".

2.1 Libraries

The programming language used for the project is Python. The libraries used are:

- `NumPy` to optimize matrix operations;
- `Pandas` to read/modify/write datasets;
- `MatPlotLib` to make the plots related to the models;
- `ScikitLearn` to perform preprocessing operations;
- `Multiprocessing` to parallelize the grid search.

2.2 Implementation choices

We implemented a class `NeuralNetwork` that represents a Neural Network capable of optimizing any given loss through backpropagation. The input dimension is specified by the parameter *input_size*. The network is fully connected and its number of hidden layers and sizes are specified using the parameter *layer_sizes*, each layer is implemented using the `Layer` class.
The parameters *act_hidden* and *act_out* permit to specify the activation functions of the hidden layers and of the output layer. All the activation functions and their derivatives are implemented with the `ActivationFunction` class (the ones implemented are Relu, LeakyRelu, Elu, Sigmoid, TanH and Linear). Moreover, we implemented different weight initializations which can be specified using the parameter *w_init*: Xavier, He and random ranged.
`Loss` class is used for implementing the loss functions: we decided to use Mean Squared Error (MSE) both for Regression and Classification tasks.
To evaluate Regression tasks we used Mean Euclidean Error (MEE) as an accuracy metric, while Classification tasks were evaluated using the classic accuracy function. These metrics were implemented with the `Metric` class.
The `Optimizer` class implements the SGD algorithm with backpropagation, which also implements:

- Momentum: specifying *ALPHA*

- Nesterov Momentum: specifying *ALPHA* and *nesterov*
- Linear learning rate decay: can be tuned in *variable_eta* which is a dictionary passed to the optimizer that if used must contain both *eta_tau* and *tau*.
- Weights regularization

The class `WeightRegularizer` implements the Tikhonov regularization that can be tuned by using the parameter *LAMBDA*.
Moreover, the training can be done using minibatch by specifying the *minibatch_size*. For the momentum with minibatch we take the mean of the delta for the last minibatches in the *momentum_window*. We also implemented the early stopping that can monitor the validation metric or validation loss by specifying the parameter *monitor*, instead with *min_delta* can be specified the minimum change in the monitored quantity to be considered as an improvement and with *patience* we can set the number of epochs with no improvement after which the training will be stopped.

2.3 Validation schema

The ML-CUP-21 dataset was split into two parts: the **development set** (80%) and the **internal test set** (20%). The development set was used to do model selection, while the internal test set was used to evaluate the generalization capabilities of the chosen models on unseen data. We stress the fact that the internal test set was never watched before evaluating the generalization capabilities.
The module `grid_search` implements a grid search on a given set of hyper-parameter values. Each configuration is evaluated using a **5-fold cross-validation** on the development set. This procedure consists in splitting five times the development set into training set (80%) and validation set (20%), each time using a different part for the validation set, then training the model with this configuration of hyper-parameters on the training set and evaluating it on the validation set. For each configuration, we report the average Mean Euclidean Error on the training set and validation set and their standard deviations.
Subsequently, the 10 configurations that achieve the lowest Mean Euclidean Error on the validation set are selected as the best models. Finally, the ensemble of these 10 models is selected as the final model.

3 Experiments

3.1 Monk Results

The inputs have been transformed with one-hot encoding on the Monks datasets obtaining an input layer of 17 units. The neural network consists of one hidden layer with 15 units with ReLu as activation function for Monk 1 and Monk 2, Tanh for Monk 3, and 1 output unit with Sigmoid as activation function. The loss is computed using the Mean Square Error (MSE) on batch gradient descent using 500 epochs. We need to highlight how the Monks datasets are sensitive to weight initialization. For this reason, we initialized the weights in the range [-0.003, 0.003] for Monk 1 and Monk 2 and with He initialization for Monk 3. In **Table 2**, we show all the results obtained.

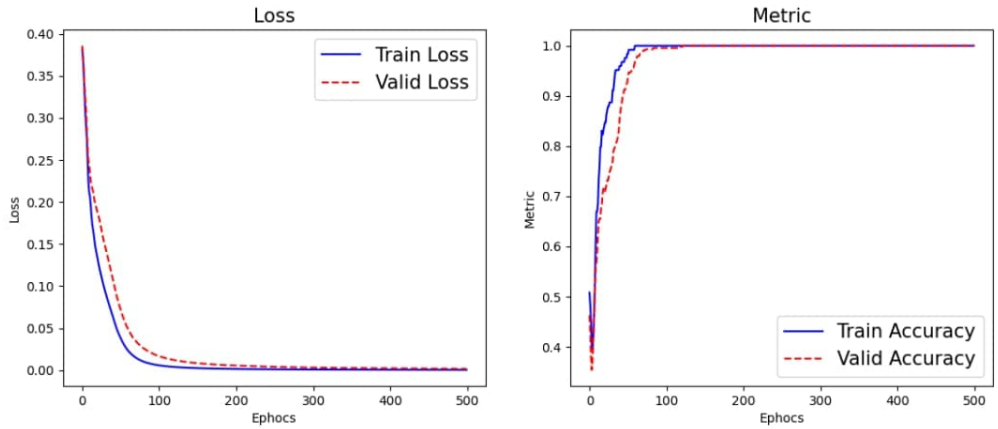
Task	**eta**	**alpha**	**lambda**
MONK1	0.76	0.8	0
MONK2	0.8	0.8	0
MONK3	0.2	0.9	0
MONK3+reg.	0.2	0.9	0.0001

Table 1: Hyper-parameters used

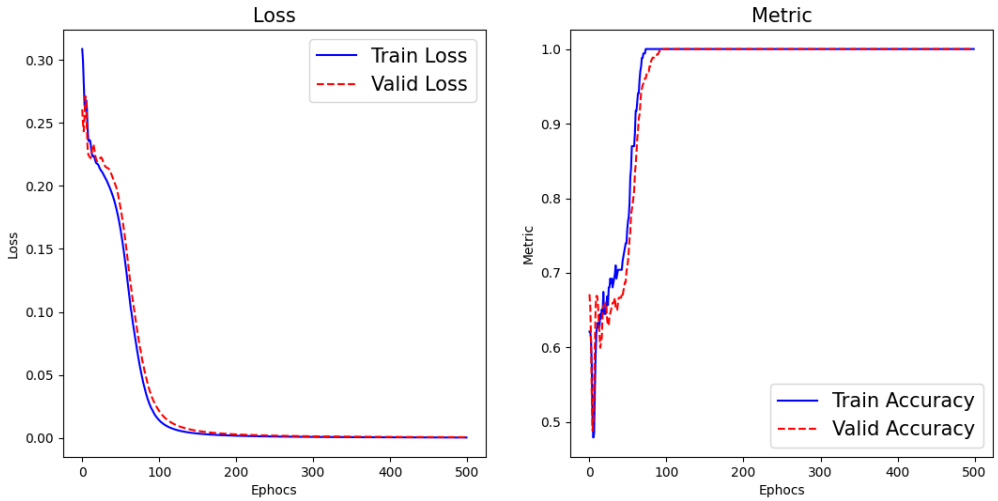
Task	**MSE (TR/VL)**	**Accuracy (TR/VL)(%)**	**MSE/Accuracy (TS)**
MONK1	0.0009/0.0081	100%/100%	0.0081/100%
MONK2	0.0004/0.0006	100%/100%	0.0006/100%
MONK3	0.0542/0.0660	93.81%/92%	0.0361/97.22%
MONK3+reg.	0.0549/0.0855	93.81%/92%	0.0455/97.22%

Table 2: Average prediction results obtained for the MONK's tasks.

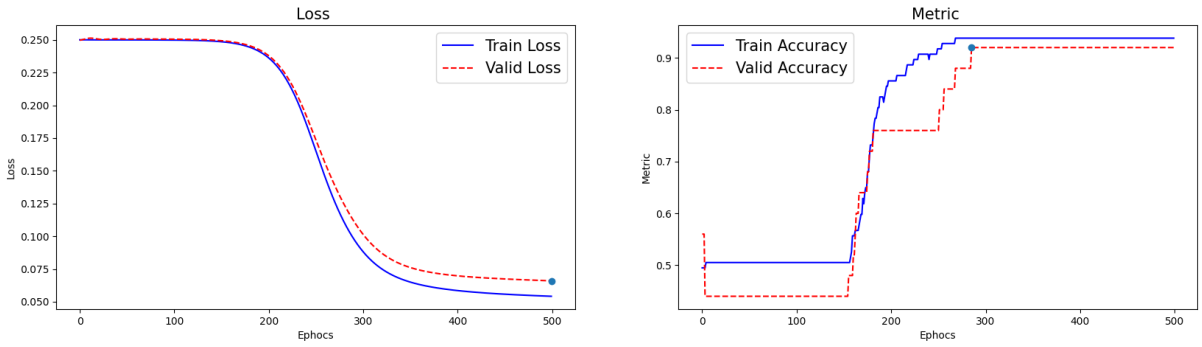
Figure 1 shows the learning and accuracy plots on each one of the three Monks datasets; **Figure 2** shows the same plots, but for Monk 3 applying Tikhonov regularization.
Looking at the plots it's clear how the network converges on average after around 150 epochs.



(a) Monk 1.



(b) Monk 2.



(c) Monk 3 without regularization.

Figure 1: Monk plots

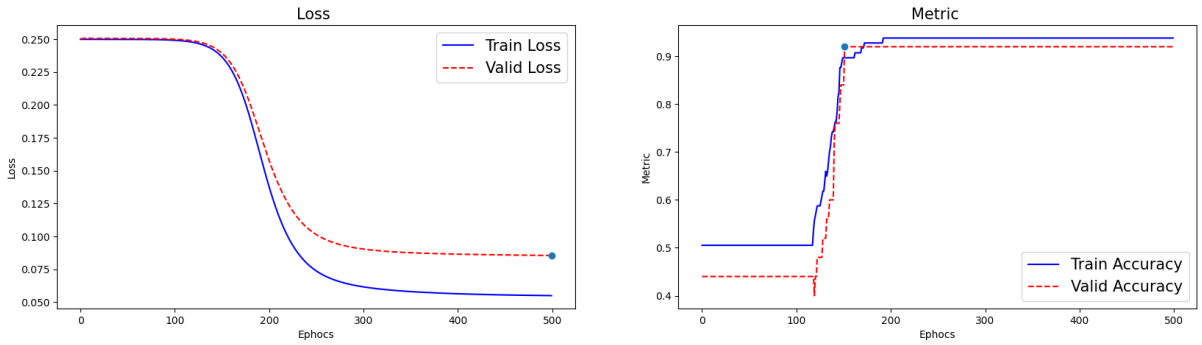


Figure 2: Monk 3 with regularization.

3.2 Cup Results

To pick the best models for the ML-CUP-21 we ran the grid search using as development set only 80% of the ML-CUP-21 dataset. We started with a coarse grid search, then taking each time the best parameters we were able to refine the search space, thus reducing the space of all the possible configurations. In **Table 3**, we have reported the hyperparameters used during the grid search.

Hyperparameters	**Possible Values**
`layer_sizes`	(70, 2), (36, 36, 2), (38, 36, 2), (38, 39, 2), (40, 40, 2), (45, 45, 2), (10, 70, 30, 2)
`act_hidden`	relu, sigmoid, tanh
`act_out`	identity
`w_init`	random_ranged, he, xavier
`loss`	mee
`Nesterov`	False, True
`early_stopping`	False, True
Hyperparameters	**Range (min, max)**
`ETA`	[0.01, 0.9]
`ALPHA`	[0.01, 0.9]
`LAMBDA`	[1e-7, 0.01]
`minibatch_size`	[1, len(training_set)]
`epochs`	[200, 1500]

Table 3: Hyperparameters values used in the grid search. The role of each hyperparameter is described in **subsection 2.2**.

After experiencing little improvements over the continuously refined grid searches we decided to stop. In **Table 4**, we reported the top 10 configurations and their performances on the validation set (average MEE in the 5-fold-cross-validation and standard deviation of MEE).

MEE	**std**	**eta**	**lambda**	**alpha**	**act_hidden**	**w_init**	**layer_sizes**
1.110965	0.045527	0.0675	1.875e-05	0.85	sigmoid	xavier	(10, 70, 30, 2)
1.111118	0.048412	0.0675	7.5e-05	0.775	sigmoid	xavier	(10, 70, 30, 2)
1.089931	0.053360	0.1	0.0001	0.8	sigmoid	xavier	(40, 40, 2)
1.090128	0.057025	0.06	1e-05	0.8	sigmoid	he	(38, 36, 2)
1.093497	0.039910	0.07	0.0001	0.9	sigmoid	he	(38, 36, 2)
1.090328	0.073973	0.07	0.0001	0.8	sigmoid	he	(38, 39, 2)
1.090471	0.051719	0.06	0.0001	0.9	sigmoid	he	(45, 45, 2)
1.081114	0.046988	0.11	1e-06	0.75	sigmoid	xavier	(36, 36, 2)
1.084705	0.059526	0.11	7.5e-05	0.78	sigmoid	xavier	(36, 36, 2)
1.088935	0.075475	0.07	1e-05	0.85	sigmoid	he	(38, 36, 2)

Table 4: Average and standard deviation of the MEE on the validation set obtained with 5-fold-cross-validation of the best 10 models.

Then we picked the 10 best models in terms of Mean Euclidean Error on the validation set during cross-validation and we did the ensemble of them computing the mean of the predicted results over training, validation and test set. In **Table 5**, we reported the performances on the training, validation and internal test set of the top 10 models and their ensemble.
Since the ensemble model shows an improvement over the validation set we decided to use it to predict the results of the blind test set. We think the model that we built will be a good approximation of the real values on unseen data since the internal test was not used during the model selection phase. For this reason, we think that the MEE on the blind set will be around **1.10873618**.

Network	Training set MEE	Validation set MEE	Internal test set MEE
network_1	1.026875735	1.13387360	1.14411516
network_2	1.04174182	1.14028492	1.16557484
network_3	1.01780425	1.10457440	1.16806941
network_4	0.99845373	1.10004699	1.12129304
network_5	0.99781807	1.078324333	1.15379372
network_6	0.98259203	1.08738387	1.12046269
network_7	0.94273935	1.07920166	1.09761987
network_8	0.96867636	1.07508763	1.12572061
network_9	0.98288808	1.08398578	1.13051194
network_10	0.94894647	1.08721313	1.11624452
ensemble	0.97160094	1.07807707	**1.10873618**

Table 5: Mean Euclidean Error of the 10 best model and their ensemble.

3.3 Computing time

Thanks to the multiprocessing library which allows the parallelization (5 threads used), and since we split up the configurations over multiple pcs, we were able to test $\sim 20k$ configurations for the grid search in $\sim 40h$ with an average of $\sim 10ms$ per epoch. Hardware used:

- Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
- Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
- AMD Ryzen 7 4700u @ 2.00GHz

4 Conclusion

Thanks to this project we had the possibility to implement and better understand what we have studied during the course. We were able to build a Neural Network framework from scratch learning the importance of both model selection and model assessment. At this point we can comprehend the importance of selecting the best hyper-parameters because they can have an high impact over the model performance. The results of the blind test set are saved in the file `Overflow_ML-CUP21-TS.csv`.

Acknowledgments

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

References

[1] Sebastian 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, Tom 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, Pittsburgh, PA, January 1991.

Appendix A

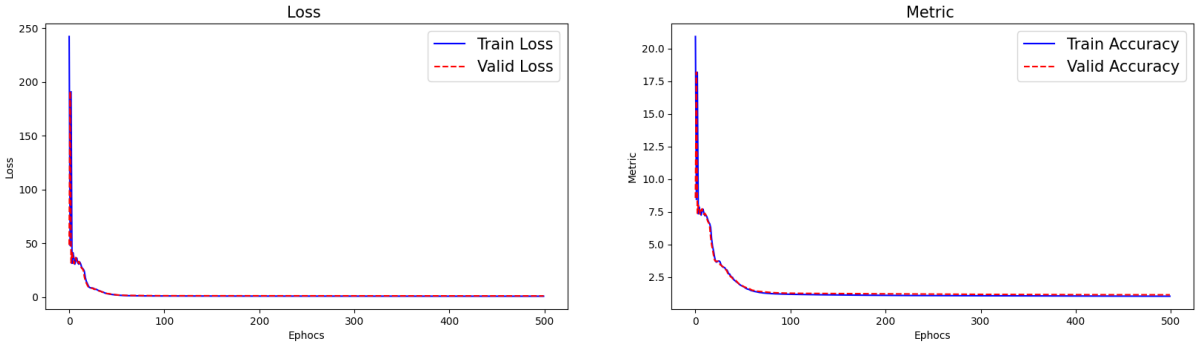


Figure 3: Model 1.

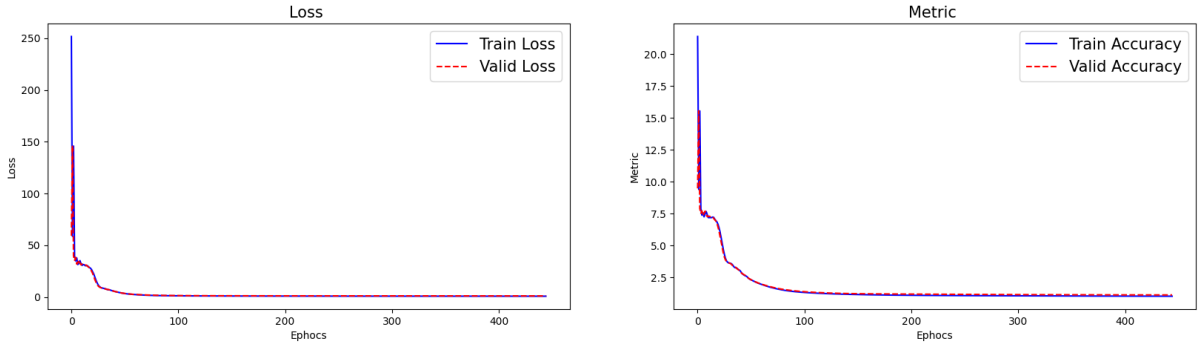


Figure 4: Model 2.

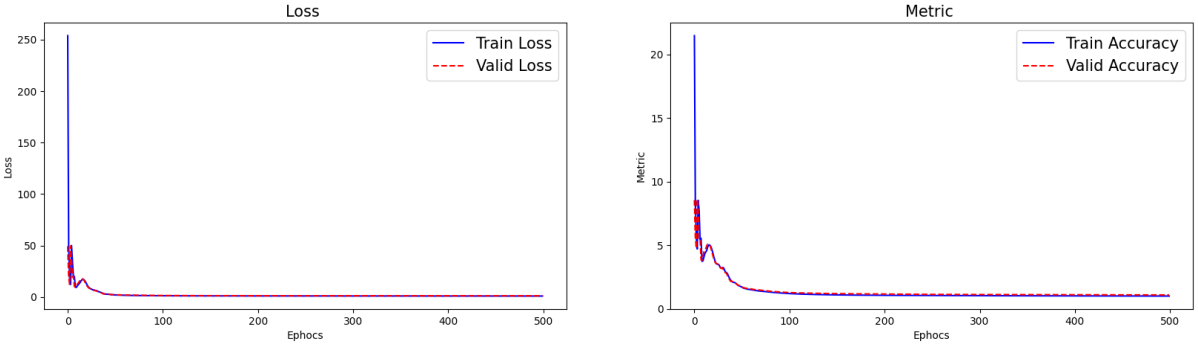


Figure 5: Model 3.

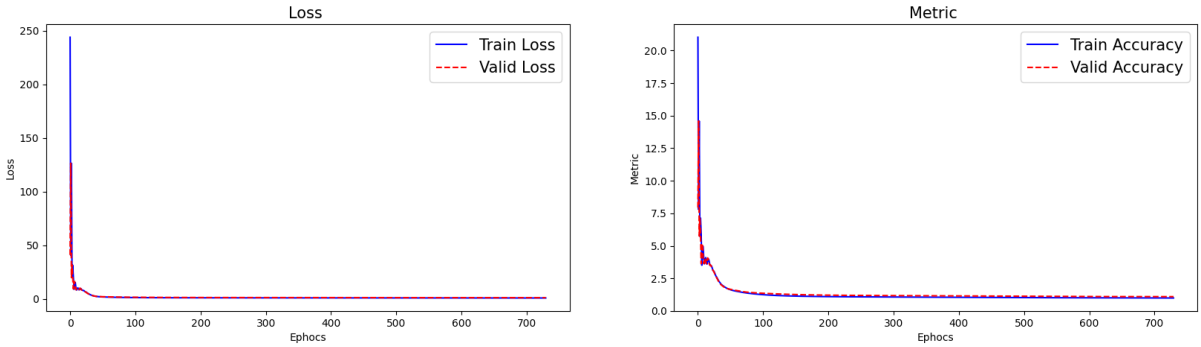


Figure 6: Model 4.

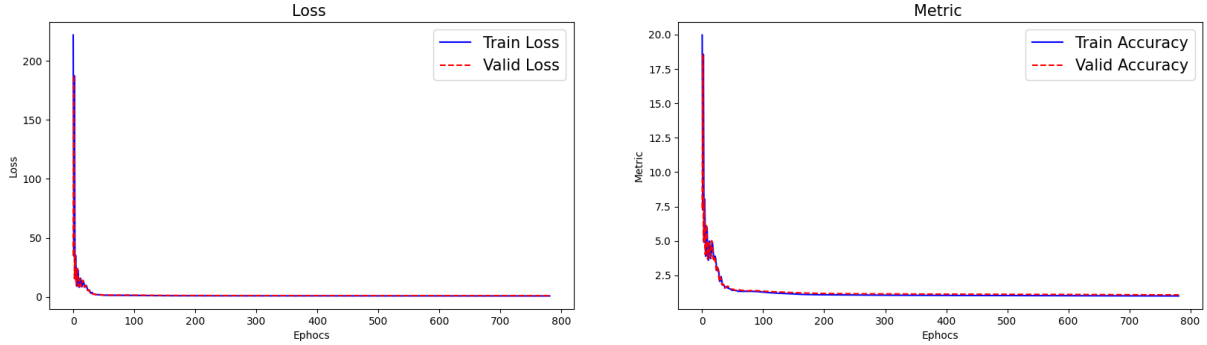


Figure 7: Model 5.

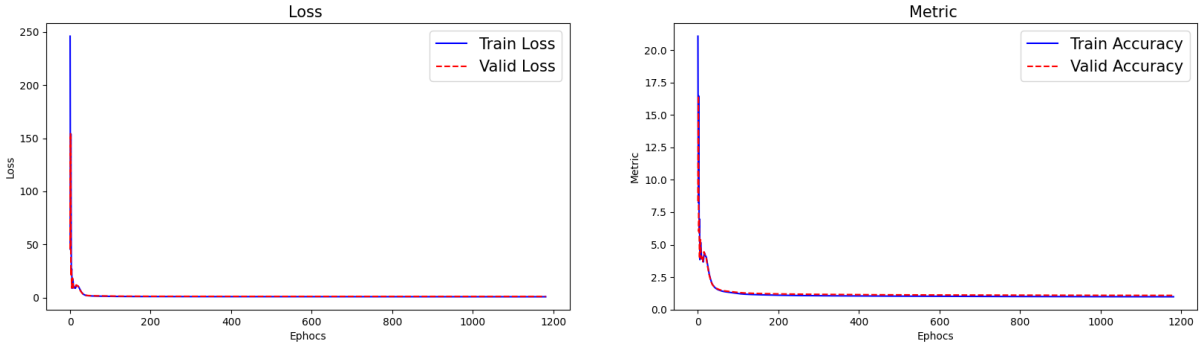


Figure 8: Model 6.

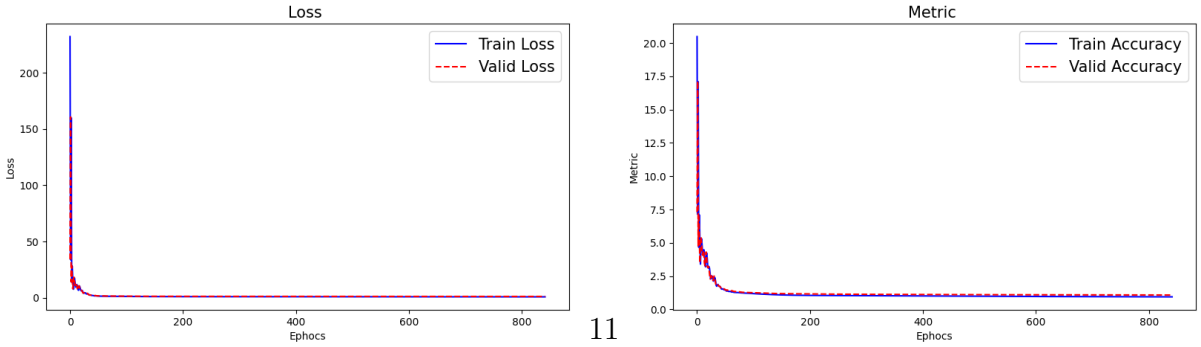


Figure 9: Model 7.

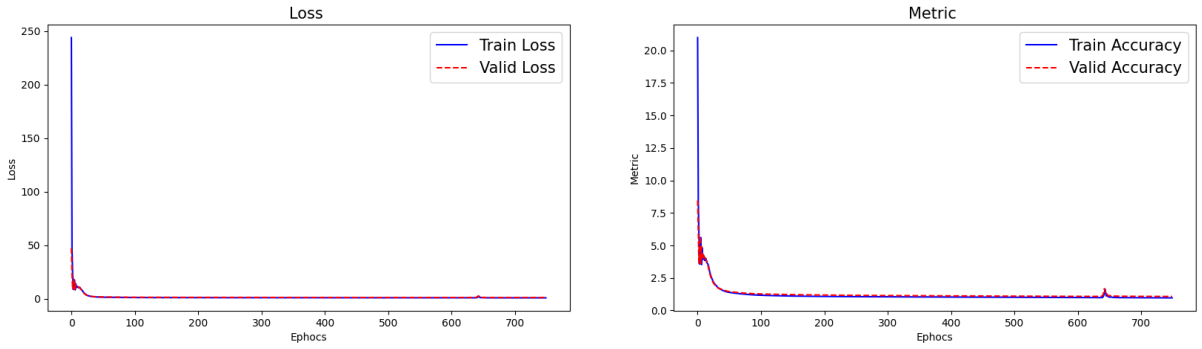


Figure 10: Model 8.

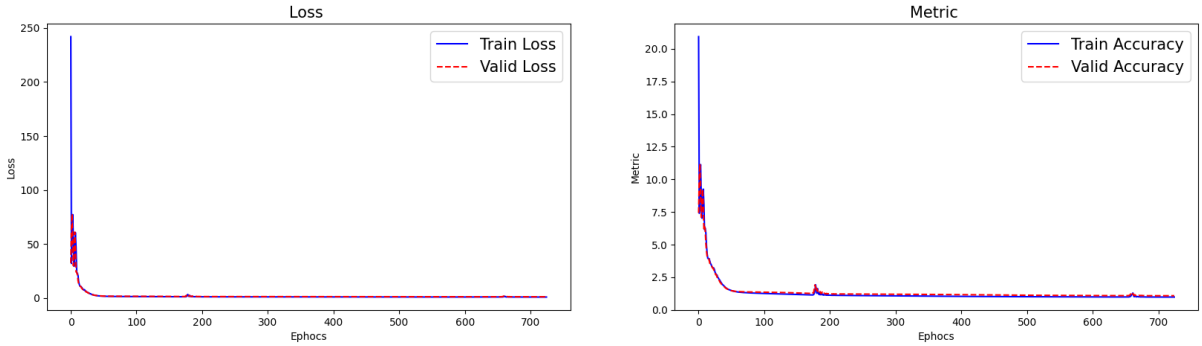


Figure 11: Model 9.

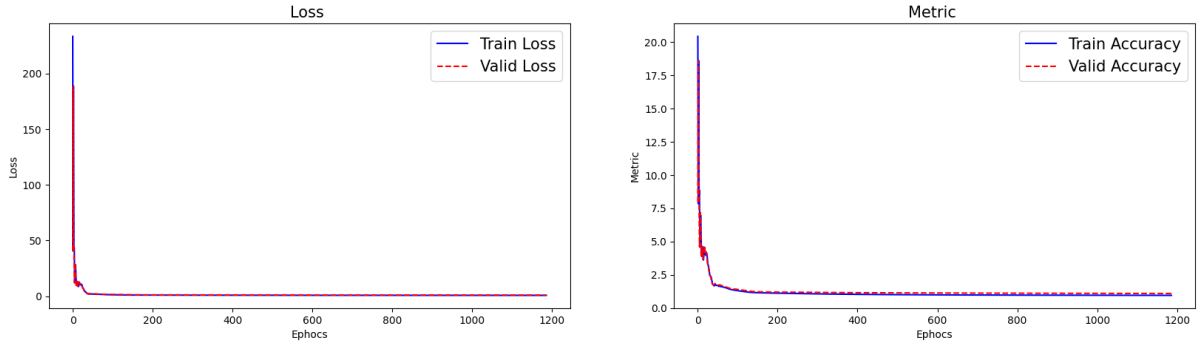


Figure 12: Model 10.