An Astonishing Title

Emanuele Cosenza
e.cosenza3@studenti.unipi.it

Riccardo Massidda
r.massidda@studenti.unipi.it

ML course, 2019/2020.
January 18, 2020
Type A project.

Abstract

Design and implementation of a multi-layer perceptron with different regulations techniques to avoid overfitting issues. The model selection and the assessment of the learning process on the `ML-CUP19` dataset are validated by using the cross-validation method.

Introduction

The presence of different techniques to improve the performances of an artificial neural network requires the use of formal methods to validate their effectiveness. Implementing from the ground up the network and the validation methods has lead to the execution of different experiments to motivate the design choices.

The proposed solution for the competition over the `ML-CUP19` dataset is a multilayer perceptron designed to be user-configurable as much as possible, allowing a big variety of combinations to be tested independently. The learning of the weights in the network is based on the back-propagation algorithm[1], variations have been introduced in the update rule to achieve regularization or to improve the overall performances.

The network also offers the possibility of early stopping, since it is a recognized good regularization technique, and furthermore it reduces the computational time by not learning for more epochs than required.

A mechanism that automatically executes a grid search over various hyper-parameters combinations has been implemented to perform model selection,

each relevant model generated can then be validated by cross-validation. The estimation of the risk, or model assessment, to evaluate the generalization power of the selected model can be computed by using a separate test set or by the double cross-validation algorithm.

Method

Any implementation of a neural network, and its relative validation mechanisms, requires a certain amount of numerical computational needs, addressed in the described implementation by NumPy[2].

Network

The class `Network` implements the neural network, by using it's constructor is possible to set all the required hyperparameters for the techniques that are subsequently described. The implementation offers methods to learn from a set of examples via back-propagation and to predict sound outcomes for new patterns in forward mode.

The initialization of the weights of the neural network is done by extracting values from a standard normal distribution with variance $\sigma = \frac{2}{n_i+n_o}$ where n_i stands for the number of inputs in the layer and n_o for the number of outputs. This has been proven to be a sound choice[3] in various use cases.

The combination of some hyperparameters could lead to numerical errors due to a phenomen known as gradient explosions[4]. This problem is dealt by normalizing the gradient if it surpassess a certain threshold.

The back-propagation learning algorithm implemented analyzes the patterns by aggregating them using the minibatch technique, to speedup the computation the update rule also considers momentum information, achieving better results with a smaller number of epochs. L2 regularization also is implemented to avoid the overfitting of the training data.

The learning process can be terminated in three different ways:

- A fixed number of epochs can be provided as an hyperparameter, leading the network to be trained for no more than the provided value.
- An early stopping mechanism is implemented by checking if the loss on a given validation set does not improve after a fixed number of epochs. This solution also leads to an implicit regularization of the model, avoiding overfitting of the dataset[5].

- Similar to early stopping if the loss on the training set doesn't improve over a fixed number of epochs the learning process is stopped. This is equivalent to assert that the norm of the gradient in the SGD algorithm is stuck under a certain threshold.

Validation

The lack of a reliable external test set required the development of a strategy to assess the performances of the model by using an internal one. Since the explicit requirement for this report to plot the learning curve of the selected final model against both the training and the test set, a double cross-validation approach is not feasible, given that it produces only a scalar value representing the risk of the family of models. Given this constraint in the validation procedure, the dataset is partitioned in training and test set by random sampling without replacement in proportion 80/20%.

The model selection follows a grid search approach, implemented in `grid.py` as a function capable to perform the Cartesian product over the set of relevant values for each hyperparameter, returning an iterable over all the sound combinations.

The grid search is used for the model selection, executing the cross-validation algorithm implemented in `validation.py` for each possible combination. The implementation shuffles the data, uses by default $k = 5$ fold over the training set and returns the best hyperparameter selection. Given the choice of hyperparameters a new model is trained again by using the whole dataset, except obviously the internal test set.

By using the internal test partition extracted by the dataset it's now possible to perform model assessment and obtain the loss information needed to plot the learning curve of the model.

The mechanism hereby described is used by the script `ml-cup.py` to automatically perform model selection and assessment, other then producing the plots and the result for the blind competition.

Experiments

MONK's dataset

The results illustrated in table 1 and 2 are obtained by averaging eight independent runs for each task. The hidden layer contains 4 hidden units

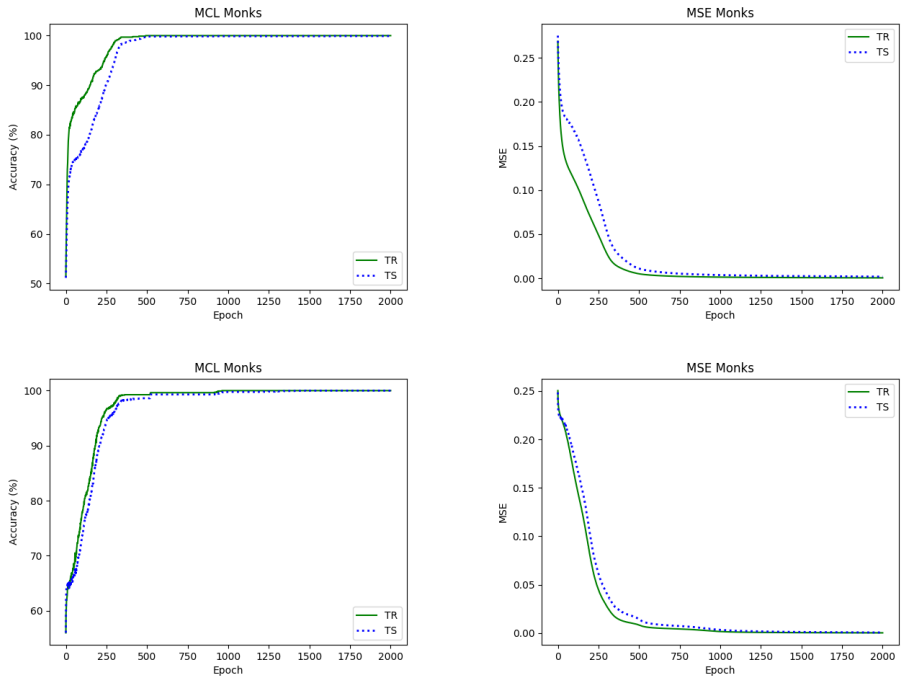
using *tanh* as activation function, while the output layer uses the sigmoid function. The network has been trained for 2000 epochs by using a minibatch of 32 examples, no further technique are used otherwise explicitly noted in the tables.

Table 1: (Experimental results over the MONK's datasets)

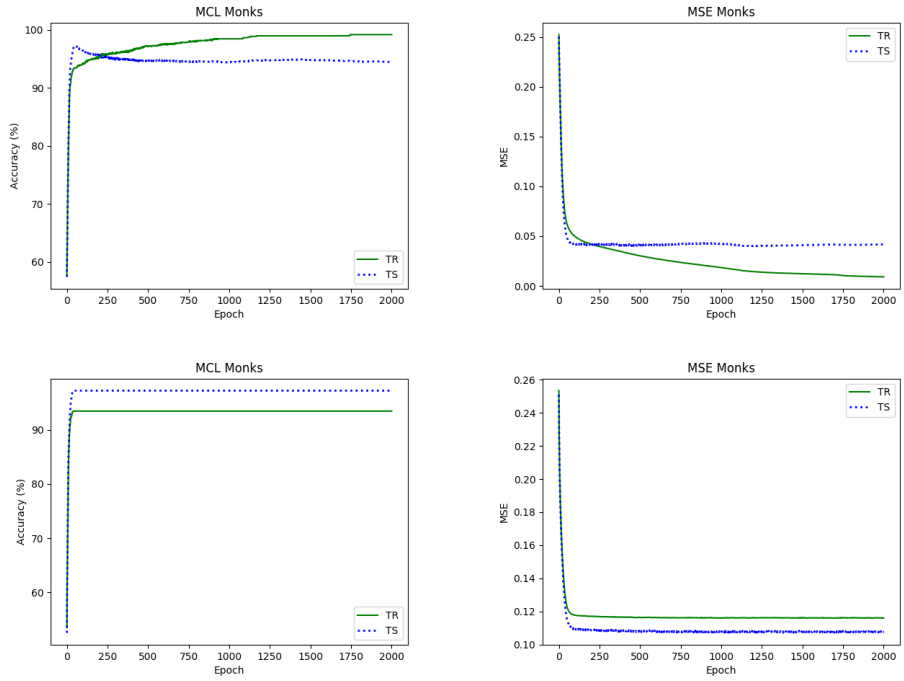
Task	Model	MSE (TR/TS)	Accuracy (TR/TS) (%)
monks-1	$\eta = 0.5$	0.0005/0.0019	100.0%/99.91%
monks-2	$\eta = 0.5$	0.0003/0.0007	100.0%/100.0%
monks-3	$\eta = 0.5$	0.0091/0.0416	99.18%/94.50%
monks-3-reg	$\eta = 0.5, \lambda = 0.01$	0.1160/0.1075	93.44%/97.22%

Table 2: (Plot of MSE and accuracy for the MONK's benchmark)

MSE (TR/TS)	Accuracy (TR/TS) (%)



MSE (TR/TS)	Accuracy (TR/TS) (%)



Cup Results

Conclusions

References

1. Rumelhart, D. E. & McClelland, J. L. *Parallel distributed processing: Explorations in the microstructure of cognition.* (MIT Press, 1986).

2. Oliphant, T. E. *Guide to NumPy.* (Continuum Press, 2015).

3. Glorot, X. & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. 8.

4. Pascanu, R., Mikolov, T. & Bengio, Y. On the difficulty of training recurrent neural networks. (2012).

5. Prechelt, L. Early stopping but when? 15.