An Astonishing Title

Emanuele Cosenza
e.cosenza3@studenti.unipi.it

Riccardo Massidda
r.massidda@studenti.unipi.it

ML course, 2019/2020.
January 22, 2020
Type A project.

Abstract

Design and Python implementation of a multilayer perceptron with momentum and different regularization 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 the network and the validation methods from the ground up has led 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 algorithm is based on the backpropagation 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 using early stopping as a stopping criterion, since it is a recognized 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 hyperparameters combinations has been implemented to perform model selection. A model assessment procedure can then be executed by using a separate test set or by the double cross validation algorithm.

Method

For the implementation, Python has been chosen because of its simplicity and the efficiency of its numerical libraries. In particular, the implementation is based on NumPy[2], which has been used to efficiently manipulate data in form of vectors and matrices. To speed up the learning process, vectorization has been exploited in operations involving vectors and matrices.

Network

The `Network` class represents a neural network. By using its constructor it is possible to set all the required hyperparameters for the techniques that are described later. The class offers methods to learn from a set of examples via backpropagation and to predict sound outcomes for new patterns in forward mode.

The initialization of the weights in each layer 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 considered layer and n_o for the number of outputs. This has been proven to be a sound choice[3] in various use cases.

Different activation functions can be chosen for each layer of the neural network. The possible choices are: *tanh*, the standard logistic function, *ReLU* and the identity function (used only in the output layer for regression tasks).

The implemented backpropagation algorithm analyzes patterns by aggregating them using the minibatch technique. The batch size is a tunable hyperparameter with possible values between 1 (online training) and the size of the training set (batch training). In the gradient descent algorithm, MSE is always used as the cost function. To speedup the computation, the update rule also considers momentum information, achieving convergence with a smaller number of epochs. Standard L2 regularization has also been implemented to avoid the overfitting of the training data.

The combination of some hyperparameters could lead to numerical errors due to a gradient explosion phenomenon.[4]. This problem is dealt with by normalizing the gradient if it surpasses a certain threshold.

The learning process can be terminated with three different stopping criteria:

- 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 for a fixed number of consecutive epochs. This solution also leads to an implicit regularization of the model, avoiding the overfitting of the dataset[5].
- Given a threshold value t, if the loss on the training set does not improve by at least t for a fixed number of consecutive 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: model selection and model assessment

The lack of a reliable external test set led to the development of a strategy to assess the performances of the model by using an internal one. Because of the explicit requirement to plot the learning curve of the selected final model against both the training and the test set, double cross validation has been avoided, since 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 development set and test set by random sampling without replacement in proportion 80/20%. The development set is then used for model selection purposes through a cross validation procedure, while the test set is used to assess the selected final model.

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 model selection, executing the k-fold cross validation algorithm implemented in `validation.py` for each possible combination. The implementation shuffles the data, uses by default $k = 5$ folds over the development set, dividing it in training set and validation set, and finally returns the best hyperparameter selection. Given the final choice of hyperparameters, a new model is trained again by using the whole development set.

By using the internal test partition extracted from the dataset, it is then possible to assess the final model and obtain the loss information needed to plot the learning curve of the model.

The mechanism hereby described is used in the script `ml-cup.py` to automatically perform model selection and assessment. In the same script, plots and results for the blind competition are produced.

Experiments

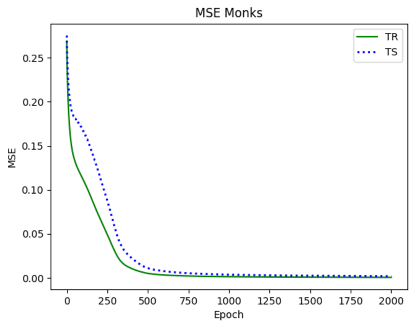
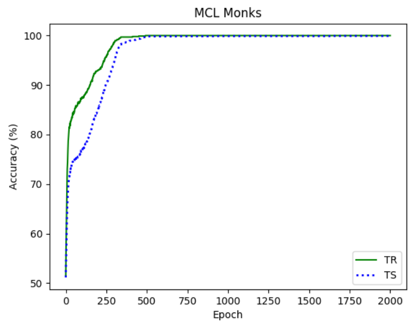
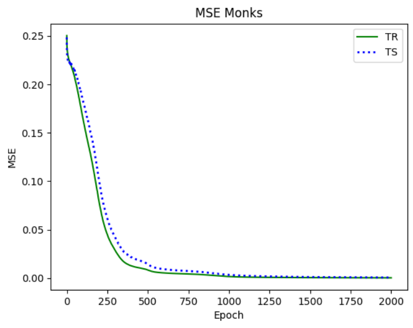
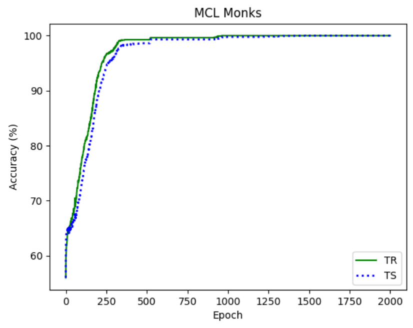
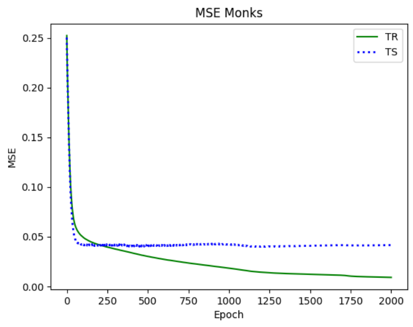
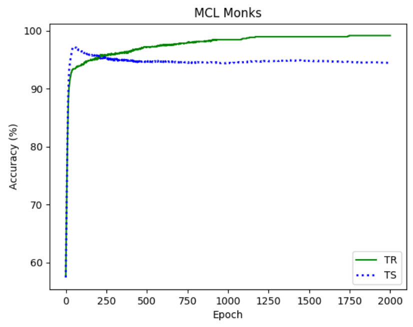
MONK's dataset

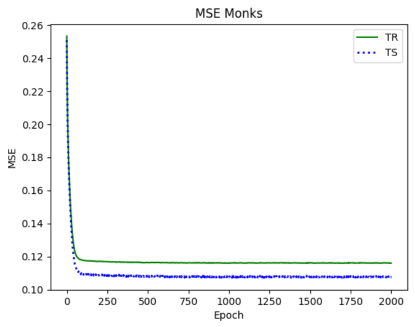
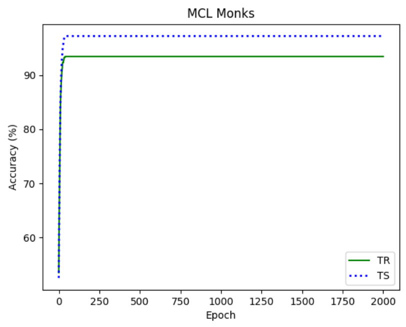
The results illustrated in table 1 and 2 are obtained by averaging eight independent runs for each task. In all the experiments, the employed neural networks are composed by a single hidden layer containing 4 hidden units. Since all three tasks are based on binary classification, the output layer is composed by a single unit with a standard logistic function as the activation function. The network outputs are therefore in the range $(0, 1)$. To get the actual classification prediction, each output is then rounded up to the nearest integer (0 or 1). In the hidden layer, *tanh* is used as the activation function. The networks have been trained for 2000 epochs by using a minibatch of 32 examples. No further techniques are used unless otherwise 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

Screening

A set of preliminary trials, some of which have been automated by the `screening.py` script, have been executed to identify sound ranges for the hyperparameters that will be used for the grid search in the model selection phase. The result of this research is summarized in table 3, while various plots of the learning curves we refer to can be found in the appendix.

Other then finding good fixed step size it has been experimentally verified that when using decaying step size an high initial value η_0 can lead to an oscillating behavior, despite respecting the ratio $\eta_\tau = \frac{\eta_0}{\tau}$ as advised[6].

Another combination proven to produce a chaotic learning curve, due to the noise produced, is the binding of a relatively high learning rate with a small minibatch size, to avoid so we explore only models with a consistent size.

As expected we noticed the Tikhonov regularization (L2), λ, and the momentum α to be more effective respectively with values nearer to 0 for the former and to 1 for the latter.

For what concerns the activation functions we haven't been able to discriminate them in this phase, but we observed that the ReLU function caused numerical issues that lead to the introduction of gradient clipping.

The number of units is also treated as an hyperparameter, we try different order of magnitude and also to use multiple layers.

Table 3: (Range of hyperparameters used in the grid search)

Hyperparameter	Implementation	Values
η	`eta`	$0.5, 0.1, 0.05, 0.01$
τ	`tau`	$100, 200$
η_0	`eta_zero`	1e$-$10
λ	`weight_decay`	$10^{-2}, 10^{-3}$
α	`momentum`	$1 - 10^{-1}, 1 - 10^{-2}$

Model selection

Model assessment

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.

6. Goodfellow, I., Bengio, Y. & Courville, A. *Deep learning.* (The MIT Press, 2016).

Appendix