# Machine Learning Project

Emanuele Cosenza
e.cosenza3@studenti.unipi.it

Riccardo Massidda
r.massidda@studenti.unipi.it

ML course, 2019/2020.
January 28, 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.

## 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. Vectorization has been exploited to speed up the learning process.

### 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 later described. The class offers methods to learn from a set of examples via backpropagation and to predict 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, thought to be 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 different stopping criteria, as seen in figure 1. The meaning of the different scenarios is the following:

a. A fixed number of epochs can be provided as an hyperparameter,

leading the network to be trained for no more than the provided value.

b. The training process is executed up to the loss on the training set reaches a certain provided value.
c. 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.
d. 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].

All of this techniques are bounded by a tunable maximum number of epochs, this is needed to avoid situations where there are no assurances about the effectiveness of the stopping criterion like in the case `b`.

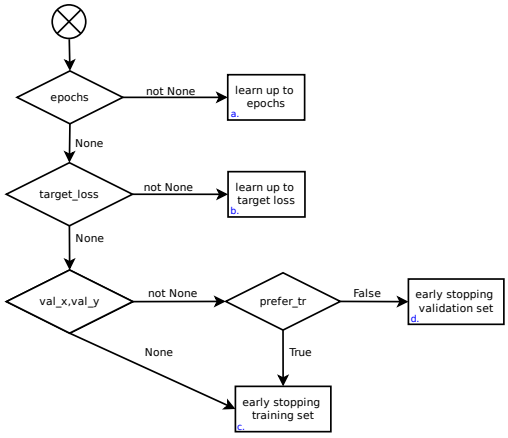



Figure 1: Flow chart for the stopping conditions

## Validation

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 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 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 in the experiments in figure 2, Tikhonov regularization is used for the Monks-3 regularized experiment in figure 3.

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% |

## 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. Some of the plots that we observed and discussed in the screening phase are resumed in the appendix figure **??**.

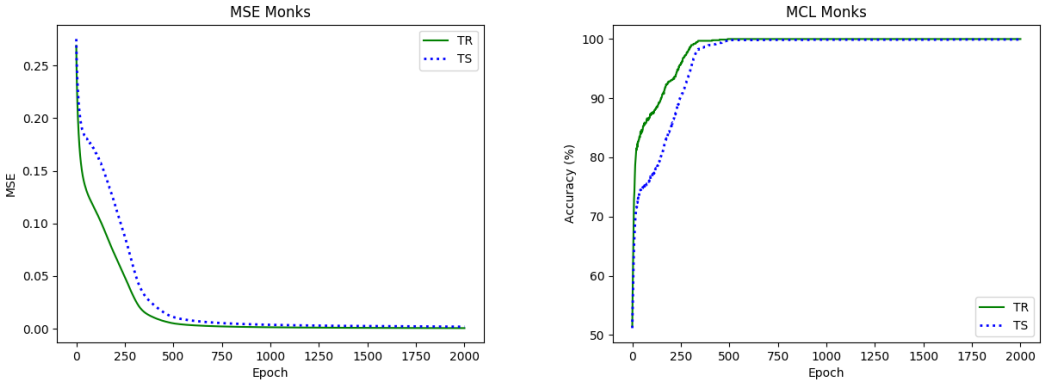
The $\alpha$ value used to tune the momentum effect is more effective when near its maximum allowed value 1, whilst this can generate oscillations in the learning curve we have observed that when using big step size actually smoothers an otherwise noisy curve.

Different $\eta$ values for the fixed learning rate have been observed, we noticed that there is an obvious strict correlation between the $\eta$ value and the size of the minibatch used for learning. We decided to fix the minibatch size and so to investigate different $\eta$ values to improve the performances of the neural network, assuming a practically standard for the minibatch size as 32.

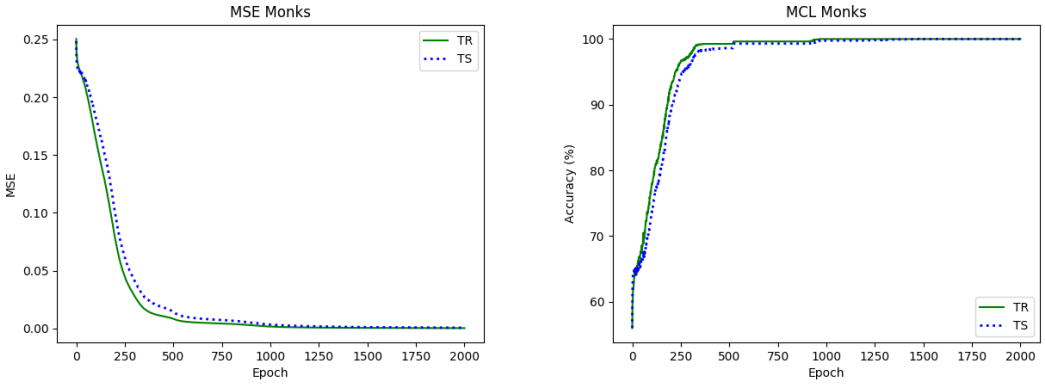
The bounds for decay learning rate are derived by the advises known in literature[6] for the SGD algorithm, so we have been able to confirm a good number of iterations $\tau$ before fixing the learning rate in the order of a few hundreds and to find a good ratio between the initial $\eta_0$ and the final $\eta_\tau$ learning rates.

The value $\lambda$ used for the regularization of the network has been considered effective for little values of the parameter, also considered the fact that early stopping itself is a regularization technique we admitted a case without $\lambda$ regularization at all.

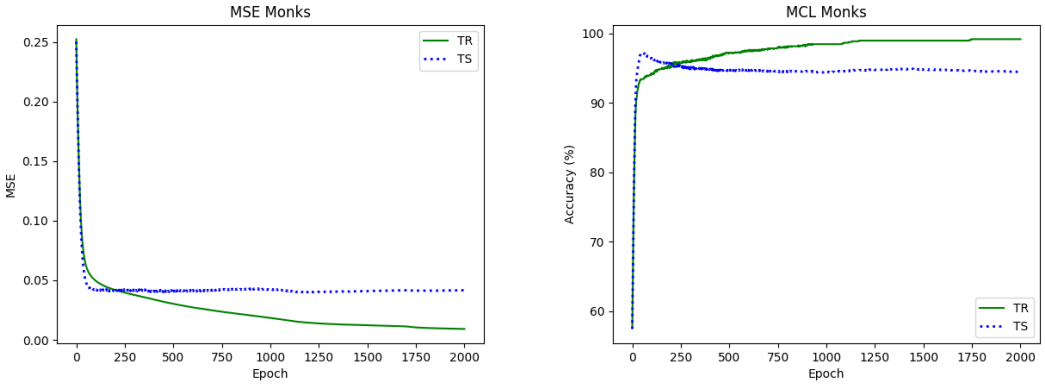
Regarding the early stopping we have found an interesting trade-off for the




(a) Monks-1

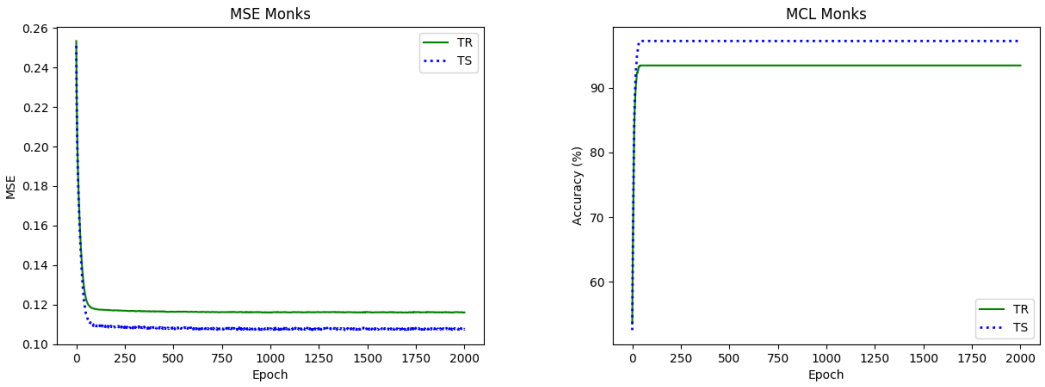



(b) Monks-2




(c) Monks-3

Figure 2: Monks benchmark




(a) Monks-3

Figure 3: Monks-3 with regularization

patience parameter in a few hundreds of epoch, this choice is derived from the observation of the relationship between patience and number of epochs, and consequently patience and error on the experiment validation set. In particular we noticed that the number of epochs quickly grows linearly with the patience, while the error on the VL decreases more than linearly in respect to the patience.

*TODO Hidden layer: activation function and number of units. Funzione di attivazioni la tanh è migliore, relu anche senza max_norm non regge il confronto, sigmoid eliminata in fase di screening. Identità per l'output layer, siamo in regressione.*

These considerations lead to the final grid chose for the model selection that we recall in table 2. The grid describes 108 possible combinations of hyperparameters, of which 72 with fixed learning rate and 36 with decaying learning rate.

Table 2: (Range of hyperparameters used in the grid search)

| | Hyperparameter | Values |
|---|---|---|
| | `topology` | [20,32,2],[20,64,2],[20,32,32,2] |
| $f$ | `f_hidden` | tanh, ReLU |
| $\eta$ | `eta` | 5e-2,1e-2 |
| $\lambda$ | `weight_decay` | 1e-4,5e-5,0 |
| $\alpha$ | `momentum` | 0.99,0.999 |

| | Hyperparameter | Values |
|---|---|---|
| | `minibatch` | 32 |
| | `patience` | 100 |
| $\tau$ | `tau` | 200 |
| $\eta_0$ | `eta_zero` | 0.1 |
| $\eta_\tau$ | `eta` | 0.01 |

**Final model**

Scelta del modello finale secondo modello formale.

Commento sul modello risultante in base alle nozioni dallo screening e ai risultati sul test set.

Commento sul grafico della stima del rischio.

Commento sul grafico dell'output space.

# 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