An Astonishing Title

Emanuele Cosenza
e.cosenza3@studenti.unipi.it

Riccardo Massidda
r.massidda@studenti.unipi.it

ML course, 2019/2020.
January 13, 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[2], 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

A functional neural network, and its relative validation mechanisms, requires a certain amount of numerical computational needs, addressed in the described implementation by NumPy[3].

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 also offers methods to learn from a set of examples via back-propagation and to predict sound outcomes for new patterns in forward mode.

Weight initialization!

The network uses by default the *tanh* function for the hidden layers and the identity function for the output layer, we provide in `activation.py` other activation functions (ReLU and sigmoid) that can be possibly used by the artificial neurons.

The back-propagation learning algorithm implemented analyzes the patterns by aggregating them using the minibatch technique. The update rule is variated to speedup the SGD computation by using momentum information the computation of SGD can speedup, achieving better results with a smaller number of epochs.

To achieve a good generalization power it's fundamental to implement

The L2 regularization is implemented to avoid the risk of overfitting the training data; also an early stopping mechanism to identify a good number of training epochs can have regularization effects as in

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 internally used by the cross-validation algorithm implemented in `validation.py`, the implementation shuffles the data and 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 Results

Cup Results

Conclusions

References

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

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

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