MLP-NN Simulation ML course (654AA) 2018/2019

A-type project

Edoardo Centamori , Abhinav Dwivedi
edoardo.centamori@sns.it , idwivedi@icloud.com

January 30, 2019

Abstract

We developed a neural network in Python from scratch using only some computational libraries. The network supports alterations in learning rate, regularization, various types of weight initialization, and other hyperparameters. It is able to perform both classification and regression tasks. In addition to this, it also supports model selection with cross validation and grid search on the hyperparameters.

1 Introduction

Our aim is to implement all our theoretical knowledge and understandings about neural networks into a functional project. We also want to experiment various combinations of the hyperparameter to understand how they're correlated to the final outcome. The results of our experiments are discussed later. The model is a classical neural networks trained with backpropagation. It support a weight decaying strategy, momentum, L_2 regularization and different types of weights initializations.

2 Method

We developed a neural network on Python taking advantage of the libraries NumPy, Scikit-learn and Matplotlib.
The code is developed in the module named 'perceptron_mode.pyl' which can be imported and instantiated.
Once an object of the class from the module is created, it can be trained and inspected. Almost all the parameters can be set while initializing the class. Some others however are set automatically (e.g. number of input units and number of output units are determined based on the data form).
Our program is pretty general and allow us to experiment between different possibilities:

- Both *mean euclidean error* (MEE) and *mean square error* (MSE) can be used as loss functions;

- The activation function can be chosen between the *logistic function*, the *hyperbolic tangent* and *rectified linear units (ReLU)*;

- Four types of weights initialization can be performed (from here on $N(x_1, x_2, \cdots, n_x)$ is a multidimensional set of normally distributed variable and can be implemented in python with numpy.random.randn(shape) and S_l is the number of perceptrons in the layer l):

  ▶ 'Type 1' : indicated for ReLU

  $$W_l = N(S_l, S_{l-1})\sqrt{\frac{2}{S_l}}$$

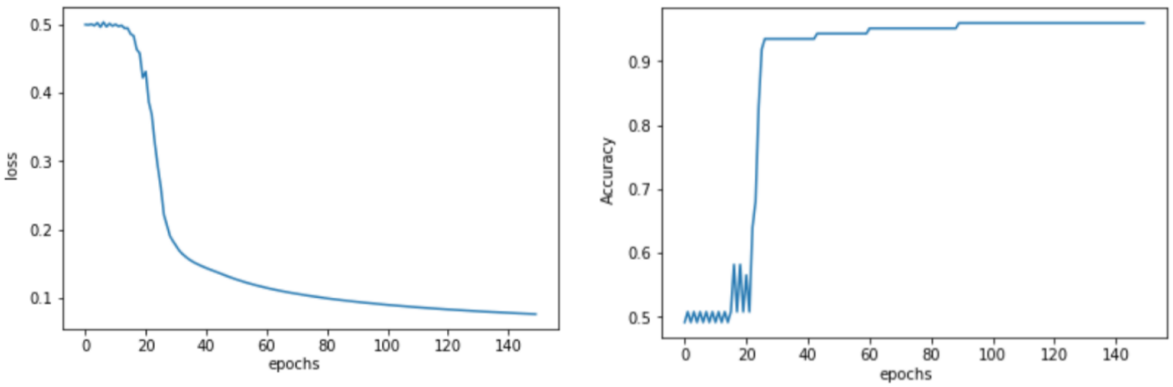
  ▶ 'Type 2' : general purpouse

  $$W_l = N(S_l, S_{l-1})\sqrt{\frac{2}{S_l + S_{l-1}}}$$

  ▶ 'Xavier' : indicated for hyperbolic tangent

  $$W_l = N(S_l, S_{l-1})\sqrt{\frac{1}{S_l}}$$

  ▶ 'Zero': Every weight is set to zero, since the symmetry of exchange of perceptrons in a layer is preserved over iteration of the backpropagation algorithms it turns out that the model cannot perform better than a linear model. Results are shown in figure 1.



(a) Loss

(b) Accuracy

Figure 1: Loss function and accuracy for a linear regression implemented from the neural network with weights initialized to zero.

We decided to experiment these different weight initializations to avoid the common problem of exploding and vanishing gradient, the initializations adapt to different

activation function so that they produce the initial values in a range that is far from both extremes [3].
Since every boolean function can be represented with a two layer neural network [2] we decided to not implement a multi-layer NN. Since the number of required units can grow exponentially for complex boolean function we performed a grid search that allows up to 256 hidden units. Anyway the MONK's problems [1] consist of some simple disjunctive target concepts so in every experiment we ran the algorithm converged toward a small number of units, usually 4 or 5. Data, that were originally label encoded (meaning that the categorical variable assume integer value), were treated with a one-hot encoding in order to prevent any false assumption as a possible order of the values assumed by the variables.

3 Experiments

Performing a full grid search with a lot of hyperparameters that varied on a big range proved to be very expensive, so we decided to perform some multiple grid searches by 'zooming' each time on the parameters. Our theoretical justification is that being the 'MSE' a continuous function of the hyperparameters for grid searches that are fines enough near the best data in the grid lies the real global minimum.
We also wanted to test the validity of [3] prescription, in particular we were interested in seeing if, fixed a particular activation function, than the corresponding (supposed) best weight initialization was the one outputted by our algorithm. The result is that this is not the case, the outputted weight initialization is really unstable and changes even from successive grid search.

3.1 MONK results

From now on the loss will always be 'MEE'. The hyperparameter analysed are:

- The number of hidden units;
- η: the learning rate;
- λ: the regularization parameter;
- α: the momentum parameter;
- The activation function;
- The seed of the random initialization;
- The weights' initialization;
- The number of epochs.

The experiment was successful since we achieved full score in the test in the first two MONK's. Results can be seen in Table 1 and in Table 2.

Task	Units	η	λ	α	mse (TR/TS)	Accuracy (TR/TS)
MONK 1	5	0.15	0.01	0.7	0.05/0.06	100/100
MONK 2	4	0.1	0.1	0.4	0.05/0.06	100/100
MONK 3	4	0.1	0.1	0.4	0.04/0.07	98/94

Table 1: Average prediction results obtained for the MONK's tasks in the training and test set.

The final accuracy in the test set is not 100%, this should be expected since as is explained in [1] the training data coming from the third MONK's problem are affected by noise and 5% of them is misclassified, this means that an accuracy that exceeds 95% is not achievable (to be more accurate, stochatic fluctuation can allow for higher accuracy, but avaraging out the fluctuation over several validation test should cancel this effect).

3.2 Cup results

The data are organized in 10 input columns and 2 output columns. After dividing them into two partitions, features and labels, we then organized the data into validation features, validation labels, trainfeatures and trainlabels. The validation to train ratio was 2:8. Then we scaled the features (both training and validation) separately by using z-score normalization.
We trained our model by some random parameters as a first check, the results were quite promising. We then searched for the best parameters, so we decided to implement the model selection and grid search.
The hyper-parameters with their respective values are given in table 2.

Hyperparameters	Values
Hidden units	4, 16, 50
Activation function	sigm, relu, tanh
η	0.1, 0.5, 0.9
λ	0.01, 0.1, 0.5
α	0.1, 0.5, 0.9
Initialization	Xav, Type 1, Type 2
Epochs	20, 500

Table 2: Hyperparameters used in the model selection.

The combination of all the hyperparameters (total 1458 combinations) were then used to train the model using crossvalidation. The loss and validation loss were then measured and recorded. The log is also appended in the project folder. The winners, i.e. top 5 best hyper parameters obtained are also listed in table 3.

Placement	α	η	λ	Activation	Hidden units	Initialization	Validation Loss
1	0.9	0.1	0.01	tanh	16	Type 2	2.3608
2	0.9	0.1	0.01	tanh	16	Type 1	2.3634
3	0.5	0.5	0.01	tanh	16	Type 2	2.3955
4	0.9	0.1	0.01	tanh	50	Type 1	2.3965
5	0.9	0.1	0.01	tanh	50	Type 2	2.4174

Table 3: Best hyperparameters resulting from grid search.

3.2.1 ML cup blind test

To perform the regression of the data, we preprocessed them as earlier. Then we used all top 5 hyperparameters to train the data and test on blind test dataset. Finally we trained the model with validation based on the best hyperparameters we found via grid search. We then tested that model with the blind test set. The results can be visualized in figure 2.

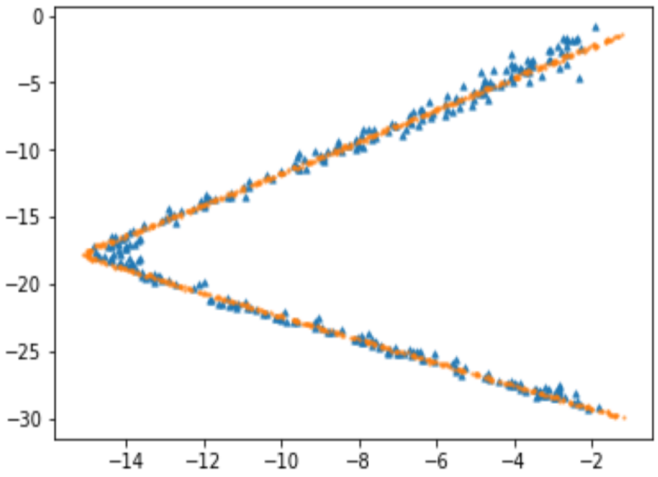


Figure 2: Results in the regression task.

References

[1] S.B. Thrun, et al. MONK's problem dataset, technical teport CS-CMU-91-197, Carnegie Mellon University, Dec. 1991.

[2] Tom M. Mitchell: Machine Learning. McGraw-Hill Science/Engineering/Math, March 1997.

[3] N.Doshi: Deep learning best practice: weight initialization March 2018