UNIVERSITÀ DI SIENA 1240

Artificial neural network integral estimation through a second artificial neural network

Matteo Fontani
Kevin Madrigali
Leonardo Proietti

Artificial Intelligence
Professor: Edmondo Trentin

Abstract

The project is composed of two different artificial neural networks: the first one is used to create a random function, while the second one has to be trained in order to estimate the integral of the function computed by the first one.
The ultimate goal of the project is to verify the correspondence between the estimate computed by the second ANN and the actual value that we retrieve using numerical integration methods.

1 Introduction

Generally, when we use an MLP neural network, it is not guaranteed that the integral of the function computed by the neural network is equal to one. Therefore, in the case in which we want to estimate a PDF using a neural architecture, it is necessary to include this constraint during the training phase. However, this procedure is really expensive from a computational point of view. The approach consists of using a second MLP network which is able to estimate the value of the integral of the function computed by the first MLP.
Rather than performing explicit integration repeatedly during training, the second network learns to approximate the integral operator itself. This surrogate model is trained offline on a diverse set of functions generated by the first MLP, along with their corresponding integrals computed numerically.

2 The architectures of the MLPs

2.1 ANN1

ANN1 is a simple MLP composed of a single input neuron, two hidden layers with 48 and 24 neurons, respectively, and a single output neuron.
For what concerns the activation functions, in the two hidden layers we used ReLU functions and a sigmoidal function for the output unit; in this way we force the output of the ANN1 to be included between 0 and 1.
This neural architecture does not need any kind of training, because its only goal is to return the value of the function computed over the patterns that we provided in input to it.
The input patterns that we fed to the ANN1 are N real values generated in a random way and normalized in the interval [0, 1]. The weights of this network are initialized randomly using a uniform distribution in the interval [-1, 1].
In output we retrive N real values, which are the values of the function computed by ANN1 over the N input patterns. These N output values will be fed, as an N-dimensional vector, to the second MLP in order to estimate the integral.

2.2 ANN2

ANN2 is another MLP composed of a N-dimensional input layer, two hidden layers with 16 and 8 sigmoidal units, respectively, and a single linear output unit to extract the value of the integral estimated by this network.
The weights of the MLP are initialized uniformly between -1 and 1, and then the network is trained through the Backpropagation algorithm, using as target outputs the actual values of the integrals of the ANN1 functions, that are created using a numeric integration algorithm.
For the Backpropagation, we used the Adaptive Learning Rate and Momentum (Adam): in this way, we obtained a learning rate that adapts itself during the training process.

3 Training of ANN2

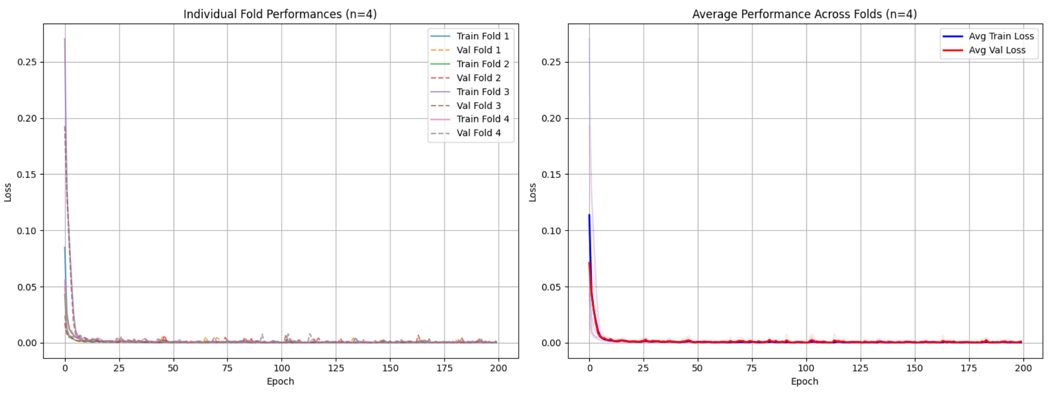
3.1 Dataset Generation

Each sample of the dataset is made of the values of the function $\phi_i^{'}$ computed by ANN1 over the N input patterns $(x_1, \ldots, x_N)$. Targets are generated using the trapezoidal method, which takes the N points in the interval [0, 1] and the N values of the function computed over these points, and returns the value of the numeric integral $\hat{y}_i$ of the function. In particular, we set $N = 500$ as the number of sampling points.
In order to generate the whole dataset we have repeated this process L times, keeping fixed the N initial points and changing the weights of ANN1 between the generation of one sample and the following one. In this way we obtained each time a different function computed by the ANN1.

$$T = \{((\phi_i^{'}(x_1), \ldots, \phi_i^{'}(x_N)), \hat{y}_i), i = 1, \ldots, L\}$$

3.2 Training workflow

We fixed exactly 600 samples for the dataset and we served of the k-fold cross-validation to evaluate the performances of ANN2, analyzing different architectures and picking the best one. In particular, we set $k = 4$, in this way we used three folds for training and one for validation of the network. The cross-validation is performed four times and each time the network weights are reinitialized using a uniform distribution between [-1, 1].
We share the learning curves obtained during the k-fold cross-validation:



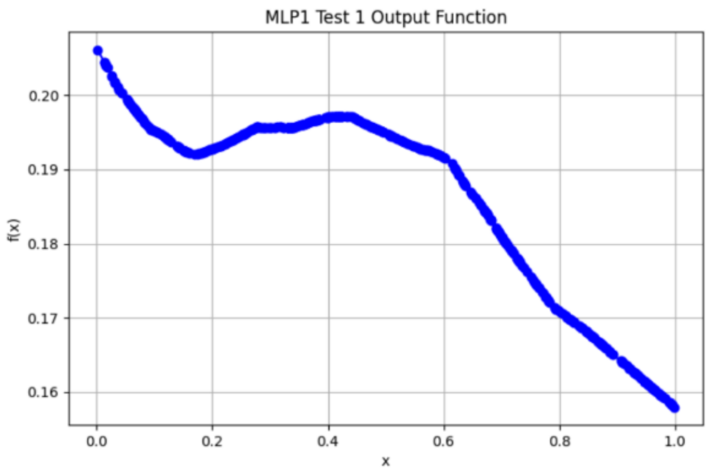
Learning curves during k-fold cross-validation

After the k-fold, we reinitialized the weights of ANN2 and we trained it using the original 600 samples; in addition, we generated 50 brand new samples that we used for validation. In this way, after each epoch of training we evaluated the network performances on the validation set, saving each time the model for which we obtained the best performances.
We decided to use a specific loss function, the hybrid loss: basically, it is a mixture between the MSE and the log-MSE, the first one is more suited for estimating higher values of the integrals, while the second one works better in the estimation of lower values of the integrals. In this way, we obtained a balanced solution for the two cases.
Following the complete formula of the hybrid loss, referred to the single dataset sample that is composed of the integral's estimate y_i made by ANN2, and the target value of the integral $\hat{y}_i$ computed via trapezoidal method. We specify that in our case we set $\alpha = 0.6$, for giving more importance to the MSE than the log-MSE, and $\epsilon = 1e - 6$:

$$\mathcal{L}_{\text{hybrid}}(y_i, \hat{y}_i) = \alpha \cdot (\hat{y}_i - y_i)^2 \; + \; (1-\alpha) \cdot \big(\log\big(1 + |\hat{y}_i| + \varepsilon\big) - \log\big(1 + |y_i| + \varepsilon\big)\big)^2$$

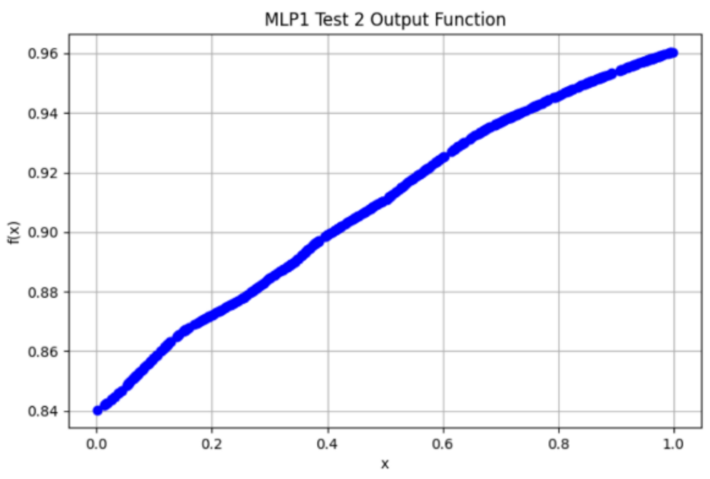
4 Results

The final tests on the ANN2 model we have obtained, are made using 22 independent brand new samples generated by reinitializing each time the weights of ANN1 (as in the generation of the dataset samples).
At the beginning of the project, we started with an ANN1 composed of 3 hidden layers with 60, 100 and 70 neurons, respectively, with ReLU activation functions for the intermediate layers and sigmoidal function for the output unit. However, we saw that we could obtain enough complex functions even using a simpler architecture; therefore, in order to reduce the time to generate the dataset, we decided to simplify the ANN1 architecture, obtaining the final version described in section 2.1.
At the beginning we used also a simplified version of the ANN2 with respect to the final one described in 2.2: we used an architecture composed of a single hidden layer with just 3 sigmoidal units and a linear output unit on top. Following, we show the results obtained using the first versions of ANN1 and ANN2, and using the MSE loss both for ANN2 training and for the performance evaluation:



Output function 1 with old version of ANN1

Numeric integral: 0.18594383198775732
Estimated integral: 0.16750723123550415
MSE Loss: 0.000340



Output function 2 with old version of ANN1

Numeric integral: 0.9061067355221812
Estimated integral: 0.777536928653717
MSE Loss: 0.016530

As we can see, the results obtained are not enough satisfying. So we decided to modify the architecture of ANN2, using the version that we have described in section 2.2, to obtain the results shown below:

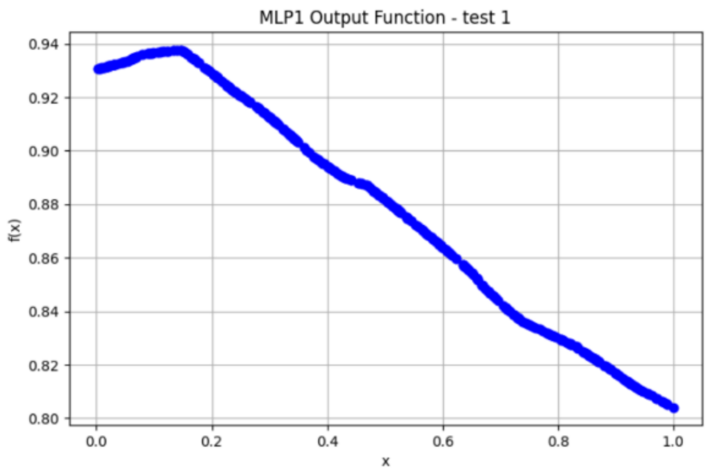
Test 1
Numeric integral: 0.048355761227958816
Estimated integral: 0.05236709117889404
MSE Loss: 0.000016

Test 2
Numeric integral: 0.7402975998253574
Estimated integral: 0.7257595062255859
MSE Loss: 0.000211

Test 3
Numeric integral: 0.9976795948288526
Estimated integral: 0.9889930486679077
MSE Loss: 0.000075

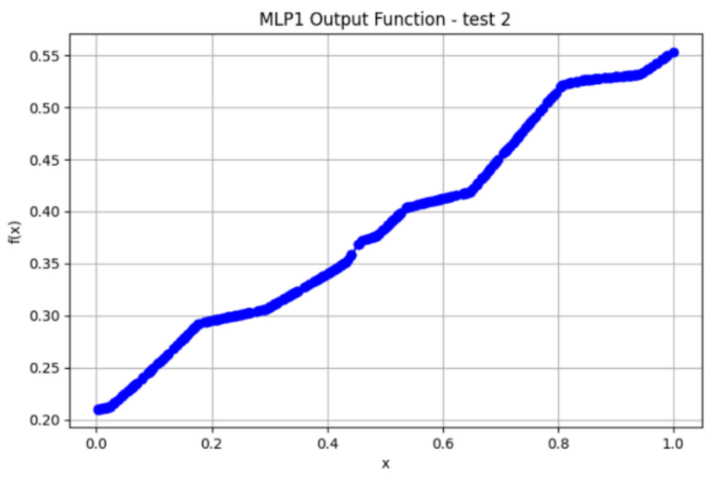
Test 4
Numeric integral: 0.016490457269251205
Estimated integral: 0.01985299587249756
MSE Loss: 0.000011

The estimates are better than before, but we can notice that there is still some error in estimating the integrals, especially for the smallest values.
For this reason, we decided to introduce the hybrid loss, described in section 3.2, for training the ANN2; for the performance evaluation we still used the MSE loss.
With this change, we obtained the following results:



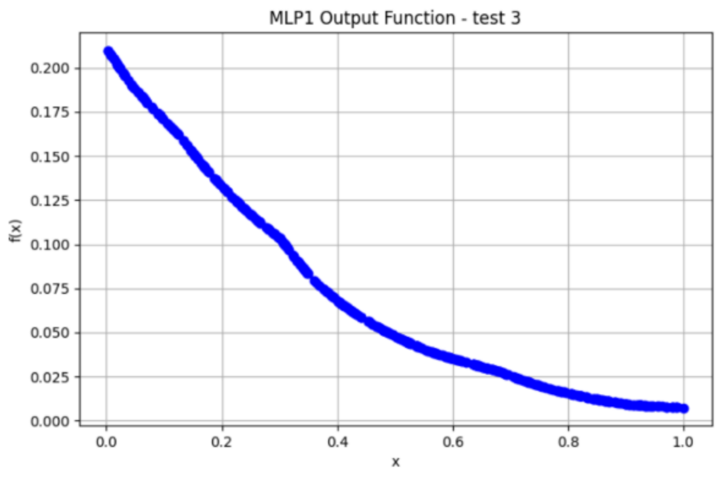
Output function 1 with new version of ANN1

Test 1
Numeric integral: 0.8746267021490297
Estimated integral: 0.8823919296264648
MSE Loss: 0.000060



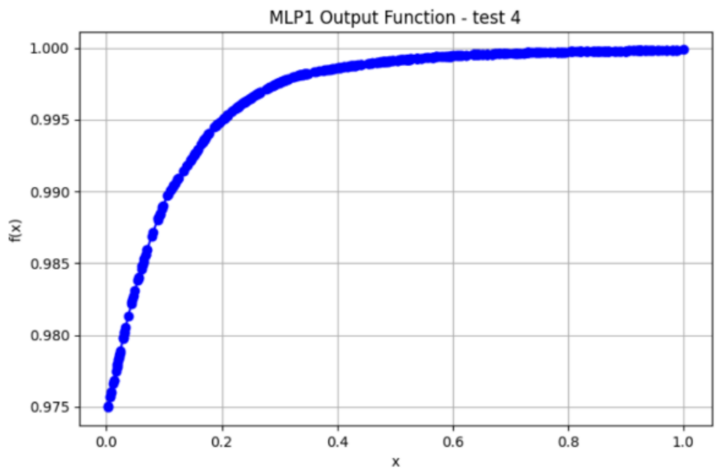
Output function 2 with new version of ANN1

Test 2
Numeric integral: 0.385943265530432
Estimated integral: 0.3842349052429199
MSE Loss: 0.000003



Output function 3 with new version of ANN1

Test 3
Numeric integral: 0.070595026537171 91
Estimated integral: 0.0782848596572876
MSE Loss: 0.000059



Output function 4 with new version of ANN1

Test 4
Numeric integral: 0.9933886197125865
Estimated integral: 0.9856539964675903
MSE Loss: 0.000060

5 Conclusions

We noticed that, analyzing the progress during the advancement of the project, the ANN1 architecture has not to be so structured in terms of number of layers and neurons, in order to obtain enough complicated functions. At the same time, the ANN2 required a more complex architecture with the purpose of returning better estimates of the function's integrals.
Another factor that leads to more accurate estimates is the usage of the hybrid loss, during the ANN2 training, instead of the typical Mean Square Error, that permits to obtain estimates that are closer to the actual values, especially for small values of the integrals. Despite these precautions and the good overall performances that we obtained at the end, the fact that the estimate cannot be even more accurate is probably due to the limited number of sampling points, set to 500, that the ANN2 uses to learn the function's integral.