December 2024

Sapienza Università di Roma

Machine Learning

Homework 1

1 Solution design

To perform this homework I chose to use neural networks, to be more specific: feedforward neural networks, because of their effectiveness in solving regression tasks. This approach was selected to address all three problems, but tests were made to choose the best architecture I could find.

2 Architecture design

Starting from the output layer, all three architectures use the linear activation function because, since regression can return any real value, we do not want to limit the output into a range. It is expected that non-linearity of the model is given by the previous layers. Moreover, it is important to recall that linear units do not significantly suffer of units saturation problem like other alternatives.
Regarding the loss function, I used the Mean Squared Errors (MSE) since it provides a direct measure of the prediction quality in terms of how far the predicted value is from the real value.
Finally, for the hidden layers, I chose the Rectified Linear Units (ReLU) activation function because it usually offers good performances and correctness. In addition, ReLU does not suffer significantly from saturation and this facilitates the network configuration and task completion.
After choosing the common elements of the architectures of all three problems, I focused on finding the best combination of the remaining features, such as different regularization techniques, the batch size, the number of epochs, the number of layers and the optimization function with its parameter. Consequently, even though I choose the number of layers and the Adam learning rate value as the two hyperparameters to try different combinations, for each task I also experimented with different combinations of other network characteristics to have the best model I could find.

3 Training and performance evaluation

To train the models and compare their performances, I used an approach based on k-fold cross-validation. In particular, I split the training set following best practices, that is, allocating 2/3 of the samples for the training set and 1/3 for the validation set. I also enabled the "shuffle" function to scramble the input samples so the cross-validation process is not influenced by possible input ordering criteria. The model fitting therefore occurs at each step ("fold") of the k-fold cross-validation. However, if the same instance of a model is used for more than one fold, we would have a bias in the training phase because the model would be trained with samples that it has already seen. To avoid this, I made sure that, for a given model, each step of the cross-validation was performed by a new instance. In my case, I used three instances because with 2/3 and 1/3 subdivision, the steps of k-fold cross-validation are three. Now, there is another problem to solve. In fact, each new instance of the model is

created with weights initialized randomly. As a result, each instance will have different initial weights values and this would lead to an unfair cross-validation since the performance of the model could also depend on the initial value of the weights. To overcome this problem, I made sure that the weights initialization is performed using a fixed seed that makes it deterministic.
At the end of the folds, I computed an average of the performances of the three models and I also saved the total training time.

4 2 joints - 2 dimensions

Here follows the description of all the tests with all the combinations and the results obtained that I performed for the 2 joints - 2 dimensions forward kinematics problem. Due to problems with the simulator, I downloaded the dataset files from this link. In particular, for this problem I used the file "r2_21_100k.csv" as training set (which will be appropriately divided into training set and validation set by k-fold cross-validation) and the file "r2_26_100k.csv" as test set. I use files with different seeds to make the problem more realistic because, in real scenarios, the model is trained with some experiment but it is tested with other experiments and situations that are not seen in training will be encountered. I did not perform a preprocessing of the dataset such as adding noise because I am not an expert of forward kinematics, so, I preferred to avoid the risk of inserting "wrong" noise values without realizing it.

4.1 Initial neural network architecture

The basic neural network architecture I chose to begin testing is composed of three layers with 64 neurons each and a batch size of 32. I thought this was a good starting point as it is a common configuration. The activation and loss functions I chose are the ones I previously described. As for the number of epochs, I set a value of 100 even though I knew it was a very high value for a problem of this type. However, I immediately set an early stopping with "patience = 10", that is, if there are no significant improvements in training the model for at least 10 epochs, the training is stopped. Obviously, for this problem the input and output layers will both have two neurons each.

4.2 Optimizer selection

After deciding on the starting architecture of the network, the next step I had to do was to choose the optimizer. In particular, I narrowed it down to two options: Adam and SGD with momentum as both are usually good choices that lead to good results. So, I ran a test for each of the two optimizers. For Adam, I set the learning rate to 0.001 as this is a very commonly used value and a good starting point for any future tuning. As for SGD, I set the learning rate to 0.01 and the momentum to 0.9. The obtained results are reported in the following table. The training column time refers to the entire k-fold cross-validation process. For this specific test, I let the model run for all 100 epochs because, in the multiple tests I ran, I noticed that the loss continued to correctly decrease even in the test set, meaning that, despite the numerous epochs, the MSE continued to decrease without causing overfitting.

Optimizer	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
Adam lr = 0.001	1.6875e-06	1.6947e-06	25'
SGD lr = 0.01 momentum = 0.9	1.0933e-5	1.1555e-05	17'

As we can see from the previous table, the results obtained with both optimizers are excellent. However, despite the fact that SGD requires less training time than Adam, the latter outperforms SGD in results by an order of magnitude, which is why I confirmed my decision to use Adam as the optimizer.

4.3 Neurons number selection

The next step was to test a different number of neurons per hidden layer. I tested with 32, 64 and 128 neurons. The following table shows the results obtained.

Neurons number	**Test set MSE (avg)**
32	2.3852e-06
64	1.1500e-06
128	1.0584e-06

Also in this case, all the combinations returned excellent results but the best performances were obtained by setting 128 neurons per layer. However, the difference in results between 128 and 64 neurons is so low that I thought it was not worth using the most complex network of these three combinations, which would have resulted in a longer training time and in a higher probability of overfitting, for a problem that the other combinations were also able to solve excellently. Consequently, I decided to set 64 neurons for each layer of the neural network.

4.4 Batch size selection

Among the various tests, I also wanted to include the batch size test. Again, I tried the three most common values, namely 32, 64 and 128. The following table shows the results obtained.

Batch size	**Test set MSE (avg)**
32	1.1500e-06
64	1.3660e-06
128	1.9089e-06

Again, each combination resulted in excellent performance, but batch size of 32 performed slightly better. However, after doing extensive testing, I noticed that this batch size rewards, albeit with negligible differences, optimizers with a lower learning rate. Therefore, to avoid prioritizing slow gradient descent models, I chose 64 as the batch size. Furthermore, tests also showed that a batch size of 64 returns better values in the Jacobian difference than batch size of 32.

4.5 Regularizers

A very important aspect of neural networks is evaluating the use of regularizers. Therefore, using the neural network architecture I have discussed so far, I tested different combinations to see how the performances changed. In particular, I compared a model without regularizers, one with a 20% dropout after the first hidden layer, one with a 20% dropout after the first hidden layer and another after the second, and a model with L1 regularization in the last hidden layer. The results obtained are shown below.

Regularizer	**Test set MSE (avg)**
None	1.2434e-06
Dropout x2	0.0008
Dropout	0.0018
L1 reg.	0.05

To obtain the above results I ran several tests with different seeds and then I computed and compared the averages of the test set MSEs. As we can see, the use of regularizers, although still leading to good results, involves a significant reduction in performances. This is probably due to the fact that, since the neural network is small and simple, there is no significant need for regularizers as they penalizes the training phase too much. This is the reason why I decided to not use regularizers in my architecture.

4.6 Hyperparameters research: Adam learning rate and hidden network layers number

Let's summarize the neural network structure. Following the tests conducted and the analysis of the results obtained, it is composed by two neurons in the input layer and two neurons in the output layer. There are three hidden layers with 64 neurons each. The activation function is the ReLU and the loss function is the Mean Squared Error. The activation function of the output layer is linear. The batch size is 64, the epochs number is 100 and the early stopping "patience" parameter is set to 20 as, after all the tests I've done, I've found that, for this problem, 20 is a good value. So, I started testing the combinations of the hyperparameters I selected, that are Adam learning rate: 0.001 - 0.0001 - 0.01 and hidden layers number: 3, 4 and 5. Here follows the table of results obtained, ordered from best to worst performance.

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
5	0.001	1.0863e-06	1.0936e-06	16'
5	0.0001	1.4583e-06	1.4764e-06	26'
3	0.001	1.6875e-06	1.6947e-06	7'
4	0.001	1.7687e-06	1.7868e-06	6.5'
4	0.0001	1.7880e-06	1.8116e-06	22'
3	0.0001	3.0451e-06	3.0874e-06	11'
5	0.01	4.9960e-06	5.0147e-06	18'
4	0.01	7.2081e-06	7.2468e-06	12'
3	0.01	9.1110e-06	9.2391e-06	4.5'

Hereafter, it is also presented the training set and validation set loss chart.

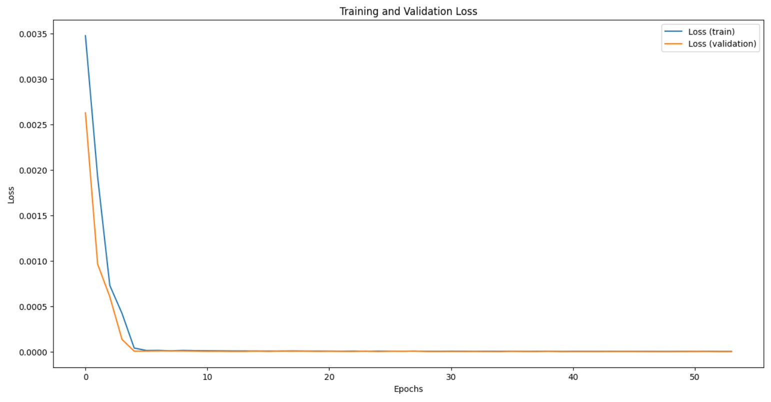

Figure 1: Training and validation loss

We can notice that the results obtained are all excellent and that, despite the very small loss value, no overfitting phenomena occurred as the performances on the test set are also very good. The final values of the losses are different by very small values, but the best is the one with **5 hidden layers** and **Adam learning rate = 0.001**. The table also shows how, for this specific problem, with the same value of Adam lr, the network with the highest number of hidden layers had the lowest loss, except for tests number 4 and 5 where the difference is anyway very low. We can also notice how the network with 5 hidden layers is so powerful that it also obtained the second position with a lr = 0.001. Finally, Adam lr = 0.01 gave the worst performance. This is probably due to the fact that this higher value leads to a much faster descent and therefore the optimizer has more difficulty in converging to the minimum of the function. However, another consequence related to this value is that the training time increases as the learning rate decreases, confirming the impact on the speed of descent of the function in search of its minimum. The best model of the table also has an average training time compared to the others. Anyway, since all the combinations show very low loss, a user may choose the combination he/she prefers based on the time he/she wants to spend in training.

4.7 Jacobians

To calculate the analytical Jacobian and the model one, I respectively followed the formula and the code shown in the assignment. Obviously, to calculate the Jacobian with the model I used the one that obtained the best performances shown previously. Below are reported the results obtained with a test joint theta.

Analytical Jacobian:

$$\begin{bmatrix} -0.01328947 & -0.00769239 \\ 0.19954693 & 0.0997037 \end{bmatrix}$$

Model computed Jacobian:

$$\begin{bmatrix} -0.01290368 & -0.0102526 \\ 0.21635279 & 0.11096982 \end{bmatrix}$$

Difference:

$$\begin{bmatrix} 0.00038579 & 0.0025602 \\ 0.01680586 & 0.01126612 \end{bmatrix}$$

Norm: 0.020397691

Also in this case, the results obtained are remarkable.

4.8 Test with 1000 input samples

As the assignment suggested, I tested the model performances with a much smaller training set. I therefore used the "r2_25_100k.csv" file limited to only 1000 samples and trained with the best combination shown before. During the tests, I noticed that, with so few samples, the loss decreased much more slowly. Consequently, I decided to reduce the number of epochs to 20 and the "patience" parameter of the early stopping to 10. The results obtained are shown below (the loss chart is not reported to not make this report too long).

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
5	0.001	0.0048	0.0042	46"

We can see how, although the loss is obviously higher, the results obtained are still very good.

4.9 Test with incremented dataset

After obtaining the results just discussed, I hypothesized that, using a training set that includes inputs generated by different seeds, the results could further improve. With this approach, in fact, the model will be trained on a set of movements generated differently from each other, and the hypothesis is that this can lead to a better generalization. So, I added to the training set the samples of "r2_23_100k.csv" and I added "r2_27_100k.csv" to test set. The results obtained are reported in the table below.

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
5	0.001	8.3691e-07	8.4228e-07	14'

As we can see, obtained results are even better than before.

5 3 joints - 2 dimensions

To address this task, I ran exactly all the preliminar tests and comparisons I illustrated for the previous one. They returned exactly the same results (with different loss and training time, of course), so, to avoid making this document too wordy, I will not show everything again, so we can focus on the hyperparameter combinations. In this case I used "r3_24_100k.csv" for training (and validation) set and "r3_20_100k" as test set. Moreover, I set "patience" early stopping value to 50 due to the fact that I noticed this problem requires a bit more training than the previous one to have extremely good performances. Of course, in this case we have three neurons in the input layer and thwo neurons in the output layer.

5.1 Hyperparameters research: Adam learning rate and hidden network layers number

As in the previous case, the results obtained for the different combinations of the hyperparameters I chose, sorted by the best performant to the worst, are shown here.

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
5	0.001	3.1532e-06	3.1621e-06	13’
4	0.001	3.6953e-06	3.7270e-06	24’
5	0.0001	4.9661e-06	4.9581e-06	12’
3	0.001	5.9120e-06	5.8637e-06	23’
4	0.0001	7.0959e-06	7.1944e-06	32’
3	0.0001	9.6766e-06	9.6912e-06	22’
4	0.01	1.8535e-05	1.8573e-05	12’
5	0.01	1.9278e-05	1.9118e-05	11.5’
3	0.01	0.0001	0.0001	20’

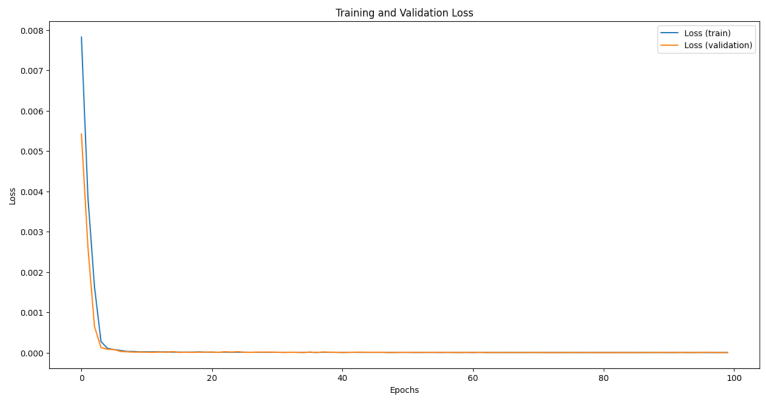

Figure 2: Training and validation loss

Also in this case, the model with **5 hidden layers** and **Adam learning rate = 0.001** performed best. Similarly to the previous task, models with Adam lr = 0.001 achieved better results, followed by those with lr = 0.001, and lastly those with lr = 0.01. Looking at the training times we can notice some values are significantly distant from the others, this is due to the activation of the early stopping. In fact, since this task is more difficult than the previous one, the descent towards the minimum of the error function requires more time and this makes it more likely that there will not be a significant improvement in the loss from a certain epoch onwards.

5.2 Test with 1000 input samples

For the same reasons discussed above, also here I set the number of epochs to 20 and the early stopping ”patience” parameter to 10. The file I used for this test is “r3_30_100k.csv” properly reduced.

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
5	0.001	0.0118	0.0121	23"

As we can see, although the increasing complexity of the problem results in a higher loss, the obtained performances are still very good.

5.3 Test with increased dataset

Finally, I ran the training with the increased dataset by adding the samples from the file "r3_28_100k.csv".

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
5	0.001	2.3282e-06	2.3472e-06	24'

6 5 joints - 3 dimensions

Also for this task I performed all the preliminar tests I did in the previous ones obtaining analogous results. We can therefore focus on the analysis of the hyperparameter combinations. The data sets used are "r5_25_100k.csv" for training (and validation) set and "r5_22_100k.csv" for test set. Here we have five neurons in the input layer and three in the output layer.

6.1 Hyperparameters research: Adam learning rate and network hidden layers number

Let's look at the results obtained for this last problem too.

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
3	0.001	8.1633e-05	8.5376e-05	22’
4	0.001	0.0001	0.0001	12’
5	0.0001	0.0002	0.0002	12’
5	0.001	0.0002	0.0002	11’
4	0.0001	0.0002	0.0002	12’
3	0.0001	0.0002	0.0002	16’
3	0.01	0.0003	0.0003	11’
5	0.01	0.0003	0.0004	11’
4	0.01	0.0004	0.0004	9’

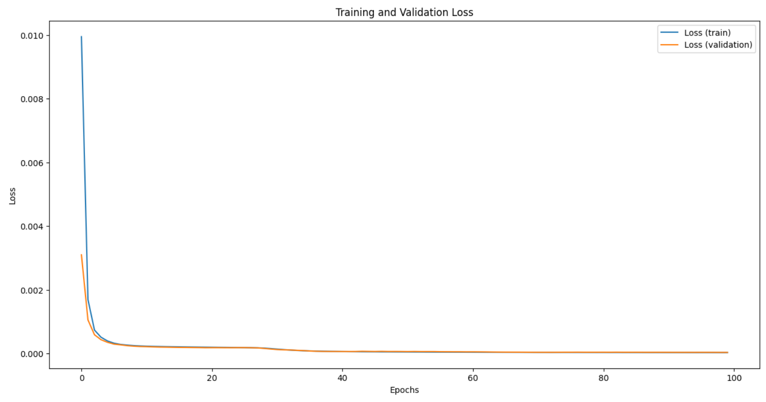

Figure 3: Training and validation loss

It is clearly visible that the most difficult problem led to the highest losses. In this case, the model that performed best is the one composed of three hidden layers and the others have very similar performances between each other. By looking at the training time We can also notice that the best model of the table is the one that was less affected by the early stopping. The results obtained are also in this case very good.

6.2 Test with 1000 input samples

Similarly as before, I trained the ”best” model with only 1000 samples. In this case, during training phase, I noticed that the greater complexity of this third problem required a greater number of epochs to obtain results comparable to the previous ones. For this reason, I set the number of epochs to 50 and the early stopping ”patience” parameter to 20. The file I used for this test is “r5_27_30_100k.csv” properly reduced and test set is “r5_22_100k.csv”.

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
3	0.001	0.0055	0.0164	28”

6.3 Test with augmented dataset

Finally, to perform this last test, I added the samples from the file “r5_30_100k.csv”.

Hidden layers	**Adam lr**	**Validation set MSE (avg)**	**Test set MSE (avg)**	**Training time**
3	0.001	3.5481e-05	3.6182e-05	21’