UNIVERSITÀ DEGLI STUDI DI PADOVA

Supervised Deep Learning

Ziliotto Filippo

November 13, 2021

In this homework the goal is to implement and test simple Neural Networks and Convolutional Neural Networks, to solve supervised learning tasks. The two tasks involved Regression, where there is the need to approximate some simple function, and Classification, where the network needs to correctly classify images from the FashionMNIST digits dataset. In order to find the best model with best regularizers and optimizers, a hyperparameter optimization is made using the *Optuna* library.

1 INTRODUCTION

[1]The first task to solve is the Regression one, where we need to approximate some function $f : R \rightarrow R$, given some input $x \rightarrow y = f(x)$, in such way that the network(x) $\sim f(x)$. The network is a Fully Connected Network (FCN) with two hidden layers, and the training data have noisy measures coming from the target function $y = f(x) + noise$.
The second task instead involves Multiclass Classification, where we need to correctly classify images of FashionMNIST digits dataset (clothes labels), whose labels are $0, 1, ..., 9$ which correspond to [T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot]. In this case, the network slightly changes architecture: the Convolutional Neural Network (CNN) implemented started with two convolutional layers, followed by two fully connected layers. Finally, using *Optuna*, the hyperparameters such as regularisers, optimizers, optimal number of hidden units are tuned in both networks in order to implement the best model. In the latter model (CNN) this is done via Colab due to the heavy processing and the results may not the exact optimal ones, since the optimal hyperparameters requires a very long training time
As the homework requires, in the appendix there are also some plots to better inspect the "inside" of the network and spot eventual pathological behaviors (such as too large weights). I added other plots regarding the relationship between different hyperparameters made the *Optuna* library.

[1]All the code and the related files can be found at `https://github.com/ZiliottoFilippoDev/NNDL-21-22`

2 REGRESSION

As already stated the training points for this homework are few, namely 100. Moreover, at a further inspection one could see that some intervals $x \in [3, 1], x \in [2, 3]$ have no training points at all. In such way, when testing the model, we will be able also to evaluate model's generalization properties and eventually understand whether the model can reproduce the function $f(x)$ for the regression task. The goal is to make a more generalized model as possibile.

2.1 Methods

The Feedforward Neural Network architecure is kind of simple. This is due to scarsity of training data and this task does not require a such deep network.

1. **Input** : (1)
2. **First Layer**: $(1, N_{h1})$
3. **First Dropout**: p_1
4. **First Activation**: $ReLU$
5. **Second Layer**: (N_{h1}, N_{h2})
6. **Second Dropout**: p_2
7. **Second Activation**: $ReLU$
8. **Output**: (1)

[2]Note as the only constraint we have is about the number of units in the input and output layers, which are always 1. Moreover, the activation functions between layers has been chosen to be ReLU's to avoid the eventual arising of gradient vanishing problem, instead the natural activation for a regression task is the $MSELoss$ already implemented in pytorch. Also a L2 regularization term is added to avoid network weights taking too large values.
The learning rate was implemented with a $StepLR$ scheduler to adapt it through the number of epochs iteration, more precisely every 200 iterations the LR decreases of a logarithmic factor of 1 (0.001,0.0001 etc..).
In order to find the best hyperparameters values for the model, i implemented a CV-5-folds random search. The hyperparameters have been initially selected in the standard way (i.e. $lr = 0.001$, $p = 0$ ecc.), and finally the *Optuna* library has been used to tuned them. The optimal model was selected with the hyperparameters which returned the lowest objective function (this implies the minimum loss), since one would expect that these ones have the characteristic that best fit the data. At last we see in Fig 2.2 what is the importance of each hyperparameter, the main two are the first number of units and the first dropout. These values changes every time the cell is ran but still this two hyperparameters seems to be the most important.

Hyperparameters	**Optimal value**
N_{h1}	296
N_{h2}	209
p_1	0.003
p_2	0.013
Optimizer	SGD
Learning rate	0.0011
weight decay	0.0005

Table 2.1: Optimal hyperparameter values obtained with the *Optuna* library

2.2 Results

After having found the optimal set, the network is trained using the full training set and validation set (respepctively 80% - 20%).
As we can see in Fig. 2.1 the network can approximate the function in all of the domain and also in the domain where the training data is scarse (between $x \in [2,3]$). This said, probably fitting the points outside the range of the whole data could bring to an incorrect prediction. Finally, around $x \sim 3$ one can see as the network makes some sort of overfitting, since the real curve should probably be more "round".

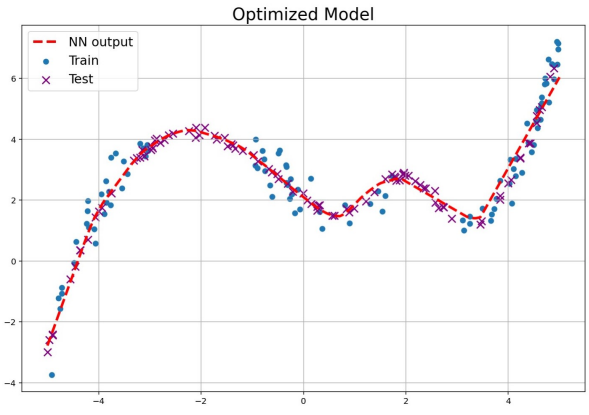

Figure 2.1: Best regression model according to the hyperparameter optimization process. We see the good generalization properties of such model.

Looking at Fig. 2.3, the weights of the network do not exhibit any particular behavior and are in acceptable ranges, given that they are not exploding. This

[2]All the code and the related files can be found at `https://github.com/ZiliottoFilippoDev/NNDL-21-22`

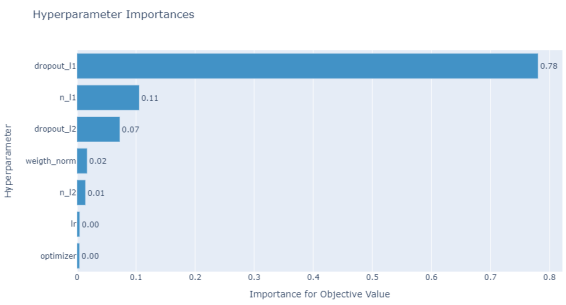

Figure 2.2: Regression hyperparameter importance w.r.t. the minimization of the objective value with the related *optuna* library.

is probably given by L2 regularization.
Finally, taking a look at the activation profiles (Fig. 5.5) one can see that for the second layer only few nodes are maximally activated: this might be an effect of sparseness introduced by the use of ReLU activation functions. On the other hand, the are few nodes who are not activated at all. To further improve the model the LeakyReLU activation function could be used, avoiding the presence of many inactive units.

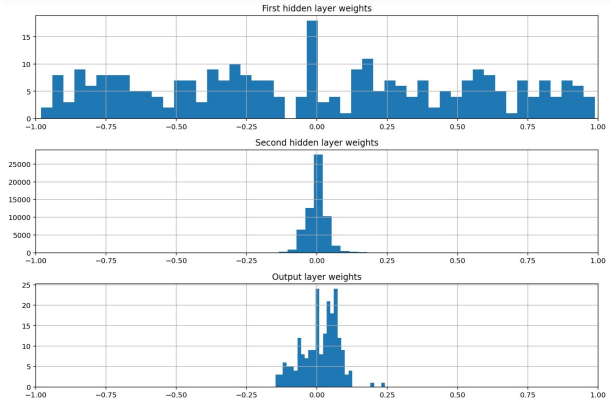

Figure 2.3: Regression Network weights histogram, effects of regularization can be clearly seen.

3 Classification

For the second part of the assignment we are asked to train a neural network mapping an input, grey-scale 28×28 pixels image, to 10 classes, able to correctly classify clothes for the FashioMNIST dataset.

3.1 Methods

The overall training set consists of 60'000 samples, while the test one of 10'000. Therefore, in order to better train and evaluate model performances, the training set is split into the actual train and validation sets, having respectively 80% and 20% of the initial number of samples devoted to training (48'000 and 12'000). Due to the larger number of training samples rather than before, it has been chosen to not perform k-fold-CV. When data is imported it has been decided, in addition to converting the image to PyTorch tensor (via ToTensor()), to normalise the datasets providing its mean and standard deviation via Normalize((0.1307,), (0.3081,))(). Moreover, with the aim to improve the performance of the model, the training data after normalization is finally augmented either adding some Gaussian noise or applying a random rotation of (0,180 degrees) with a certain probability ($p = 0.1$). This should result into a better generalization property for the network.[3]
The network architecture implemented is the following:

1. **First Convolutional Layer** : with 1 channel in input, being the image in grey scale, n channels ($n = 16$) as output, kernel 4×4, padding 1 and stride 2, in order to reduce the dimensionality of the image to 14×14. The activation function is a ReLU.
2. **Second Convolutional Layer**: with 16 channels in input, $2 \cdot n$ channels as output, kernel 5×5, and stride 2, in order to reduce the dimensionality of the image to 5×5. The activation function chosen is the ReLU.
3. **Flatten layer** : Flatten layer in order to construct a normal feedforward neural network structure.
4. **First hidden layer**: : consisting of 128 units. The activation function is the ReLU.
5. **Dropout Layer**: : connections were dropped with p probability
6. **Output Layer**: consisting of 10 units

[3] All the code and the related files can be found at `https://github.com/ZiliottoFilippoDev/NNDL-21-22`

7. **Output Activation**: Softmax in order to have a "probability" like output predction class.

No Pooling layer was used, instead it has been chosen to exploit the stride parameters proper of Convolutional layer in order to reduce the dimensionality of the problem, thus diminishing the computational effort needed for the training. As before, the number of output units is constrained due to the nature of the task. The loss used is the CrossEntropyLoss, moreover since the training took much more time rather than the previous task, it has been implemented a EarlyStopping class which eventually stops the network in the case some metrics (e.g. validation loss) does not improve for a number of iterations more than the patience parameter. As before in the definition of the learning rate a scheduler has been added to better lower the validation loss.

3.2 Results

Hyperparameters	**Optimal value**
$n_{channels}$	16
$p_{dropout}$	0.004
Optimizer	Adam
Learning rate	0.002
weight decay	0.0045

Table 3.1: Optimal hyperparameter values obtained with the *Optuna* library. The $n_{channels}$ parameter is refered to the first Convolutional layer, the second layer adapts as $2\dot{n}_{channels}$

With referral to the previous task, adding more dimensionality to the hyperparameters space would have resulted most likely too an inefficient model. Moreover, we have defined the optimal set as the one according minimizing the objective function in the optuna library, as in the regression task this results in a minimum of the validation loss. The task has been perfomed in Google colab, still the computation time took quite a bit.
The performance are depicted in the confusion matrix in Fig. 3.1. The model is indeed able to classify well digits with an overall accuracy of ~ 90%, which is sub-optimal but still a good result. We clearly see that there is a difficulty in distinguishing a $T-shirt$ with a $shirt$ and $shoes$ w.r.t. $boots$ but this is kind of expected.

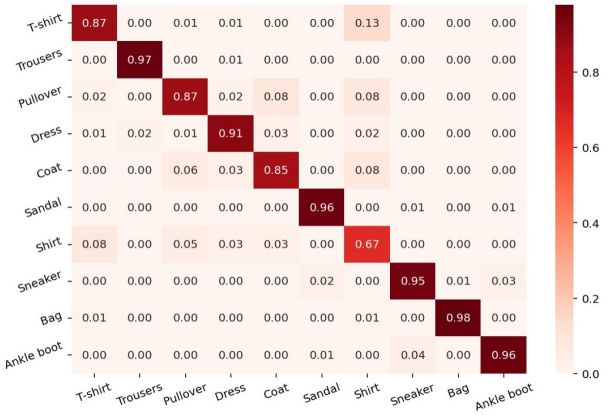

Figure 3.1: Normalized confusion matrix which returns the test accuracy of the best model. The average test accuracy over all samples is 90% .

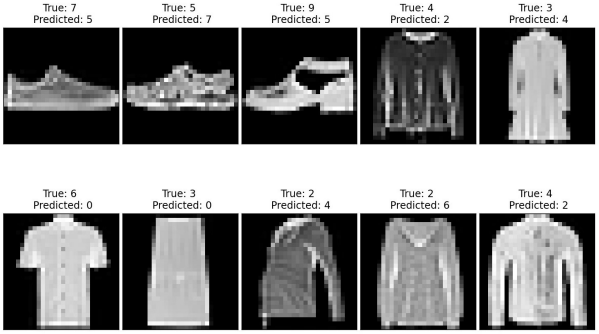

Figure 3.2: Example of incorrect predictions of the model. As we see also in the confusion matrix, the worst perfomance are when we compare T-shirt with shirt.

[4] If one inspects the filters of the second convolutional layers in Fig 5.6, we can see as the feature maps act really different from one another. This layer is able to learn more abstract features (see Fig. 5.8), and according to the filter weights it seems like it is acting as an edge detector with the edges having different shapes and orientations. Activations can

[4] All the code and the related files can be found at `https://github.com/ZiliottoFilippoDev/NNDL-21-22`

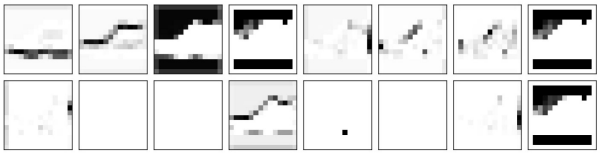

Figure 3.3: Activations profile of first convolutional layer of the network for a certain input. Here we can clearly see the shape of a shoe/sneaker.

clearly tells us what the input image is, as one can see from Fig. 3.3. Moreover, taking a look at the weights of the network of the conv and output layers, we cannot spot any weird behavior. Indeed they are in acceptable ranges and well distributed, thus showing us that the network was properly trained and might not be suffering of overfitting.

4 Conclusion

In conclusion, both networks were able to solve sufficiently good the task they had been implemented for. In particular, for the regression task, the target function is well approximated in the intervals where training points were present without seeming to fit (too much) the noise. However, in the interval outside the training and test data, the network presents some criticalities since it continues with a straight line. This is a typical behavior of ReLU activation functions, whose output is not as smooth as would be the one of a Tanh or Sigmoid. On the other hand, the performances of the network seem to be acceptable, despite the lack of training points. On the other hand, for the classification task, the network implemented is able to correctly classify 90% of the clothes labels. One should note that the testing has been performed only using the original normalized test set, despite the network was trained using noisy images to enhance its generalization properties: it would have been interesting to see how the network behaved also in presence of some disturbs during the "testing" phase.

5 Appendix

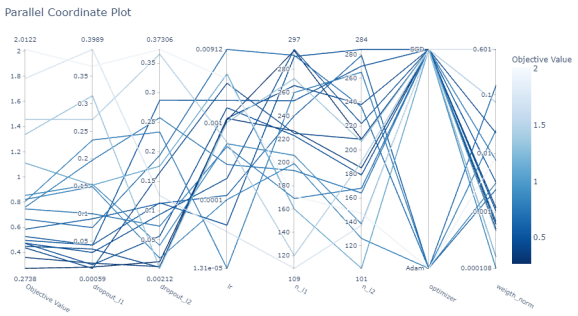

Figure 5.1: Parallel coordinate Hypeparameters, related to the optimization done in the regression task.

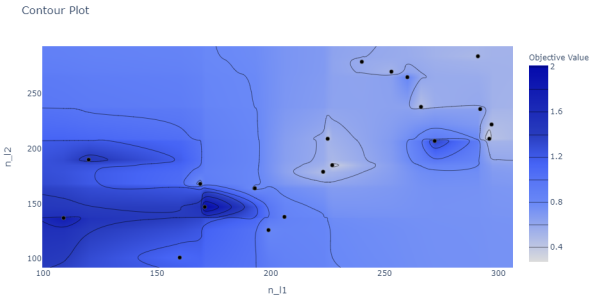

Figure 5.2: 2D example of how the trials for the minimization objective value creates this surface which then tells us the best hyperparameters to minimize the validation loss.

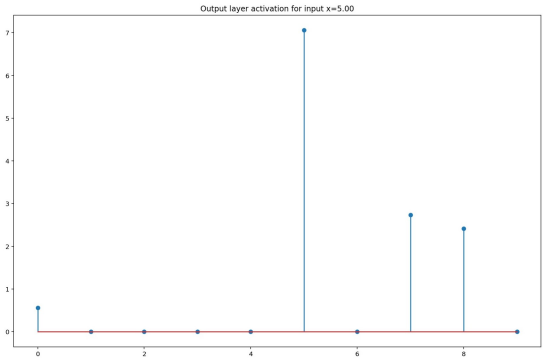

Figure 5.3: Final activation for a single input image (label=5 so a sandal) in the CNN model. As we can see the model takes the $argmax()$ of this value correctly predicting the label.

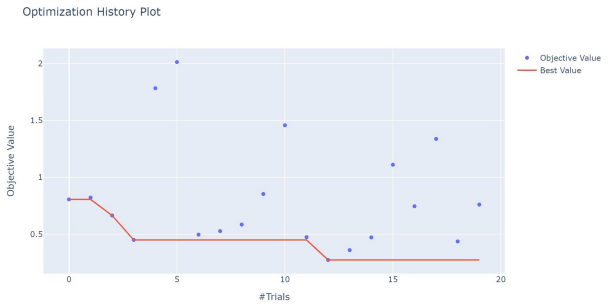

Figure 5.4: Objective value minimization trials, as we see the majority of the trials ends in a very unoptimized model.

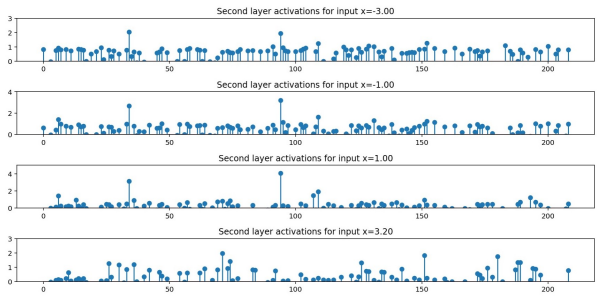

Figure 5.5: Second layer activations for the regression task, as wee see strangely the 38 & 170 neurons are nearly always activated.

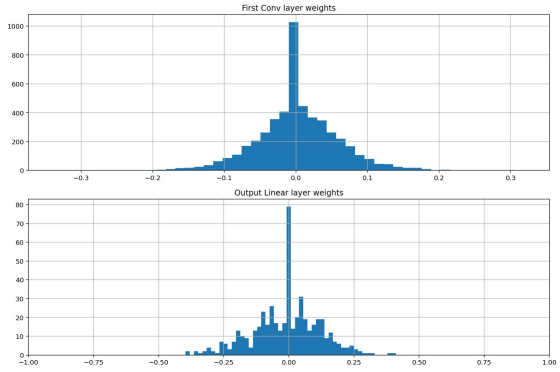

Figure 5.6: Classification weights for the different layers of the CNN network.

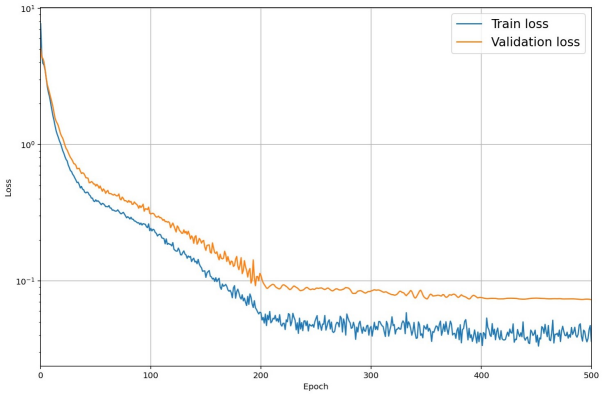

Figure 5.7: Regression task loss. We can clearly see the dropout effect on the decaying loss. Also note that we can see how the learning rate scheduler improves the minimization when is activated (every 200 epochs).

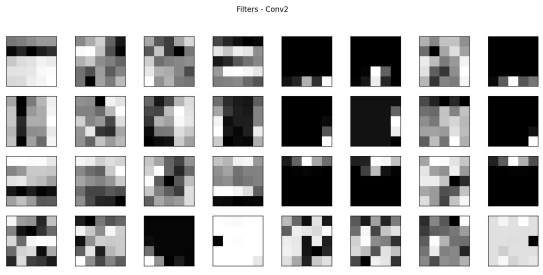

Figure 5.8: Some filters of the second convolutional layer, in this case the number of channels is 16 and for space reasons only few are plotted.