Neural Networks and Deep Learning 2021/2022

Supervised Learning

Capettini, Hilario (2013031)

June 20, 2022

Contents

Abstract

In this work I implemented simple neural network models for solving two classical supervised learning tasks. The approximation of a function using a regression model and the classification of images from the Fashion MNIST dataset. For both tasks I explored the usage of different optimizers and regularization methods. The hyperparameters were optimised using a gridsearch for the regression and a Bayesian optimisation using Optuna for the classification task.

Keywords
Supervised Learning, PyTorch, Classification, Regression, Neural Networks

1 Introduction

For this homework I work under the framework of supervised learning, where the models are trained using a dataset containing an input and a label. All the code is developed using the open source machine learning library PyTorch [1].

The first task consisted in approximating a function $f : R \to R$ using a regression model. This approach is based on a fully connected network (FCN) whose training was performed over noisy samples of the function f. The hyperparameters of the model were optimised using a gridsearch and the accuracy of the model was estimated using cross validation.

The second task requires to train a neural network to perform the classification of objects appearing in images. The pictures were obtained from the Fashion MNIST [2] dataset containing greyscale (one channel) images of 28×28 pixels. As the classification process is performed over images I decided to use Convolutional Neural Networks (CNN) which are able to use spatial information contained in a image. As the training of this kind of networks is time expensive I decided to optimise its hyperparameters using a Bayesian approach.

2 Regression

The provided data for the training is presented in figure 1, it can be seen that the training dataset is not equally distributed in the region of interest, moreover there are two regions where no training points are present (around $x = -2$ and $x = 2$). The absence of points in these places is a problem for the model which has to be able to generalise to capture it.

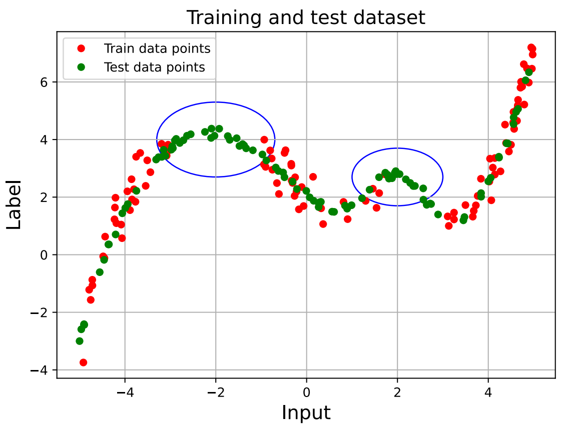

Figure 1: Training dataset and test dataset for the regression, the blue circles denote areas without training data.

Hyperparameter	Interval
$Nh1$	$[25, 50, 75, 100]$
$Nh2$	$[25, 50, 75, 100]$
dp_1	$[0, 0.01, 0.1$
dp_2	$[0, 0.01, 0.1]$
lr	$[1e-3, 1e-2]$
wn	$[1e-3, 1e-4, 1e-5]$

Table 1: Search space for the gridsearch

2.1 Method

For this task I used a simple Fully Connected Neural Network built using two hidden layers with Nh_1 and Nh_2 neurons each. Each layer has dropout capabilities with probabilities dp_1 and dp_2 which help with the model regularization. And the chosen activation function was *ReLU* which is a function that can be computed in a fast way and reduce the vanishing gradient problem if compared to the simpler *Sigmoid* activation.

For the training process I used the *Mean Squared Loss* given that we are dealing with the regression of a set of points. The chosen optimiser was *ADAM* which proved to reach better results than the classical *SGD* optimiser. For *ADAM* I decided to optimise two of its parameters, the learning rate lr and the weight decay wn which adds a *L2 regularization* over the loss.

To build the best performing model I decided to pick the hyperparameters values using a gridsearch (see table 1), although this method is not recommended for training Neural Networks because it can be very time consuming, I did use this strategy here because of the small size of the dataset and the simple model. In any case to further reduce the training time of each model an *Early Stop* strategy was implemented based on the validation loss evolution along the epochs.

As the training dataset is so small (around 100 samples) I decided to evaluate the loss of each model using a *Cross Validation* (CV) strategy, in particular I used a 5 folds CV. In this way we avoid the risk of not using samples which could contain important information for our model and also reduce the risk of overfitting.

The gridsearch and also the cross validation where implemented using the Skorch [3] library which allowed to me to use some classical function from Scikitlearn [4] as GridsearchCV in the context of PyTorch.

2.2 Results

The gridsearch over 864 different models gave as best hyperparameters:

- Neurons in the first layer $Nh_1 = 75$
- Neurons in the second layer $Nh_2 = 75$
- Dropout for the first layer $dp_1 = 0$.
- Dropout for the first layer $dp_2 = 0.1$
- Learning rate $lr = 0.01$
- Weight decay (L2 reg) $wn = 1.e-5$.

In figure 2 can be observed the evolution of the training and validation losses as a function of the epochs. For this model after hc250 epochs a plateau for both losses is reached and so the best performance is obtained. We also know that this plateau was reached because the early stopping broke the learning before reaching the 1000 epochs.

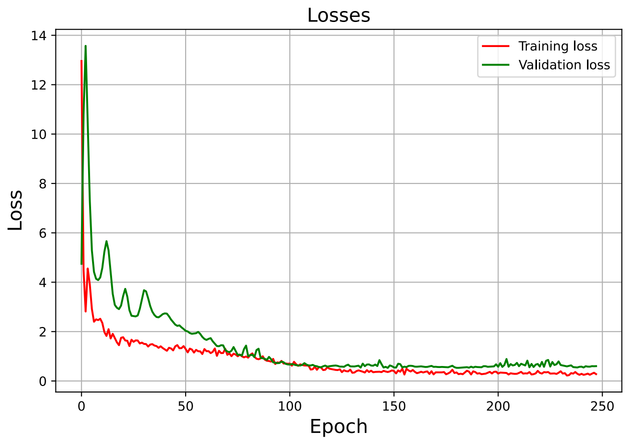

Figure 2: Evolution of the training and validation losses along the epochs for the best performing model.

Finally the best model was used to predict over the test set, these predictions can be seen in Figure 3, where we observe that that the model is not overfitting because the noise is not being capture by the model. However the model did not manage to capture the part of the target function around $x = 2$, that is the region without training data.

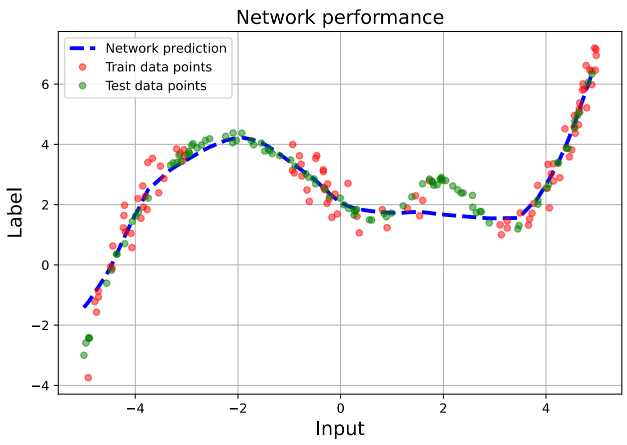

Figure 3: Fit of the model on the test data.

Figure 4 shows the weight histograms of the two hidden layers and also the output layer, in general I can not identify any particular behaviour which could indicate any problem, as exploding or vanishing weights. In figure 5 I present the activation profiles of the different network layers; there we can observe that all the neurons are active, which is expectable given that the optimal dropout was zero.

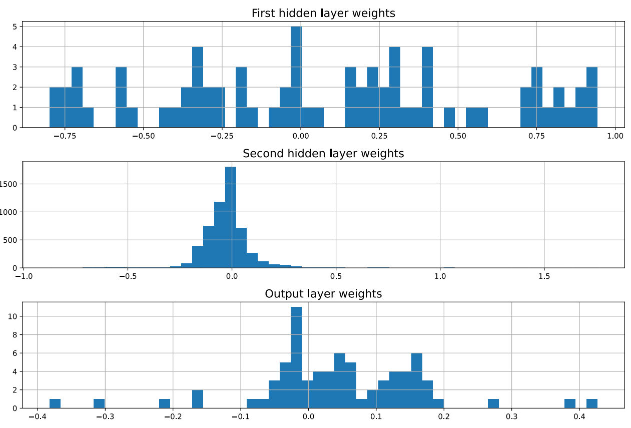

Figure 4: Histogram of the hidden layers weights.

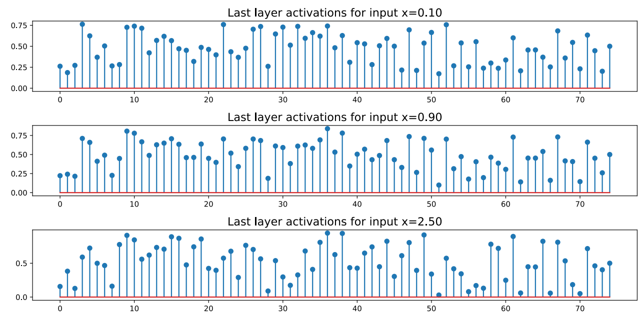

Figure 5: Visual representation of the activation.

3 Classification

The classification is performed over the MNIST Fashion [2], a set of 60000 training samples and 10000 test samples. All of the Images contain one centred object that can be classified into *T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot* these are greyscale images of 28×28 pixels.

3.1 Method

As the dataset used for the classification is arguably large I decided to split the training dataset keeping the 80 percent for training and 20 percent for validating the model. I did it instead of performing a *k fold* Cross Validation because the training of the network requires a considerably amount of time and also because the data for training is abundant. I also went for the strategy of augmenting the data by performing random horizontal flips of the images.

To tackle the classification I used a Convolutional Neural Network (CNN) (Figure 6), with the following characteristics:

- **First convolutional layer**: The depth is one because the input image has only one channel (greyscale), the number of filters nf is a tunable hyperparameter of the network, the receptive field size (kernel) was chosen to be 5×5, stride 1 and padding 2 guaranteed to have an output volume with the same dimensions as the input. As activation function I used a ReLU.

- **First Pooling layer**: To reduce the number of units of the network I implemented a pooling layer after each convolutional layer. It has a spatial extent (kernel) of 2×2 and a stride of 2 which returns an output with volume $14 \times 14 \times nf$.

- **Second convolutional layer**: The depth is nf, the number of filters is $nf \times 2$ and the other parameters are keept as in the first convolutional layer.

- **Second Pooling layer**: It has the same parameters as the first pooling layer a spatial extent (kernel) of 2×2 and a stride of 2 which returns an output with volume $7 \times 7 \times (nf \times 2)$.

- **Flatten layer**: This layer is used to convert the 3D volume of data to feed a Feed Forward Neural Network that will perform the final classification.

- **Dropout layer** Conections are dropend with probability p to regularize the model.

- **Hidden layer**: This layer has $7 \times 7 \times (nf \times 2)$ units and uses the ReLU activation function.

- **Output layer**: This layer is the responsible for the output and has only 10 units, one for each category.

Figure 6: Architecture of the CNN for the classification.

For the network training I explored two optimizers, the classical *SGD* and the more advanced *ADAM*, for both of them I set the learning rate lr and weight decay wn as hyperparameters to tune. For the classification task the used loss was *Cross Entropy*.

The chosen architecture and setup allow me to optimise 4 hyperparameters and the CNN optimiser. The optimisation of the hyperparameters can be costly for this architectures, that is why I decided to avoid the brute force gridsearch and went for the Bayesian optimisation approach which gave me the chance to get better results faster. To do so I used Optuna [5], which is a hyperparameter tuning framework that implements

Hyperparameter	Interval
nf	$[16, 32]$
p	$[0, 0.5]$
Optimizer	[SGD,ADAM]
lr	$[1e-5, 1e-2]$
wn	$[1e-4, 1]$

Table 2: Search space for Optuna

this Bayesian optimisation. Furthermore this library focus on promising trials only by using a pruning strategy, which help us by saving time. The searchspace for Optuna can be seen in table 2.

3.2 Results

After going through 48 trial the best hyperparameters according to Optuna were:

- Number of filters in the first convolutional layer $nf = 32$
- Dropout probability $p = 0.2$
- Optimiser SGD
- Learning rate $lr = 0.0003$
- Weight decay (L2 reg) $wn = 0.004$.

The importance of the different hyperparameters can be observed in Figure 7, as expected the most important hyperparameter is the number of filters, which give the network the capability of extracting information through the convolution.

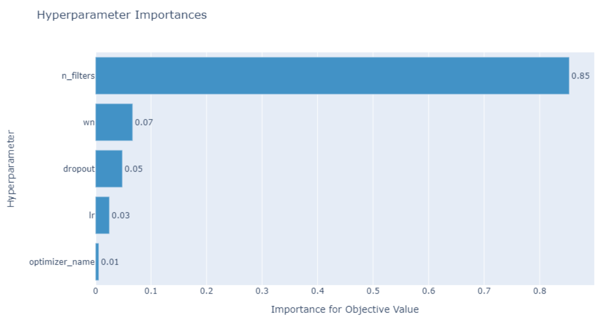

Figure 7: Relative importance of the different hyperparameters

In figure 8 can be observed the evolution of the training and validation losses as a function of the epochs. If compared with the previous task (regression) we can see how the model is learning much faster, something we were expecting given the larger amount of data and the more complex model.

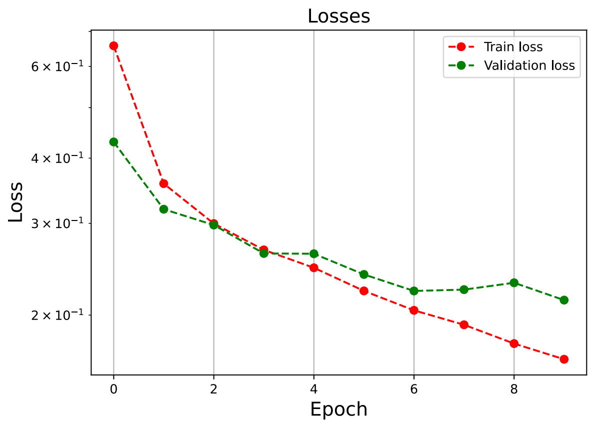

Figure 8: Evolution of the training and validation losses along the epochs for the best performing model.

The obtained accuracy on the test set was 91.56 %, however I observed that this accuracy was not the same for all the categories as seen in the confusion matrix in Figure 9. The objects that are more problematic to classify are the shirts which are usually misclassified as T-shirts or coats. Some of the misclassified object can be seen in Figure 12.

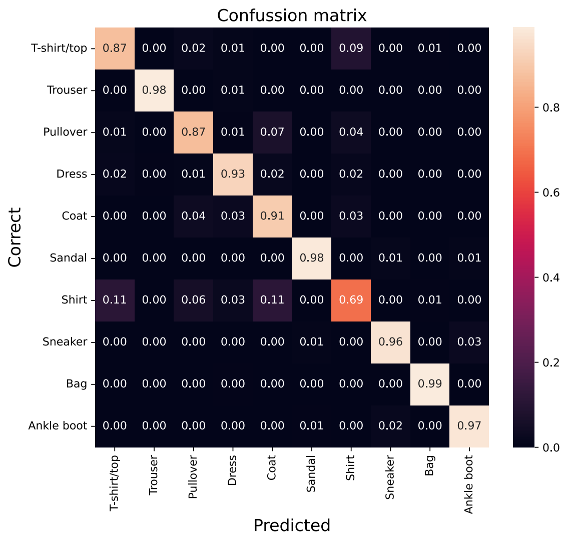

Figure 9: XXX

The learnt filters for the first convolutional layer are shown in Figure 10, from seeing them is is very difficult to understand what are they capturing, however once we plot the activation profiles (Figure 11) we can clearly see one of the articles, in this case a bag. There we also realise that most of the filters are detecting horizontal or vertical regions in the image. Also the filters and activation profiles of the second layer were plotted (Appendix A Figures 13 and 14) in this case it is not possible to see what the filter is doing, also because some information was compressed by means of the first pooling layer.

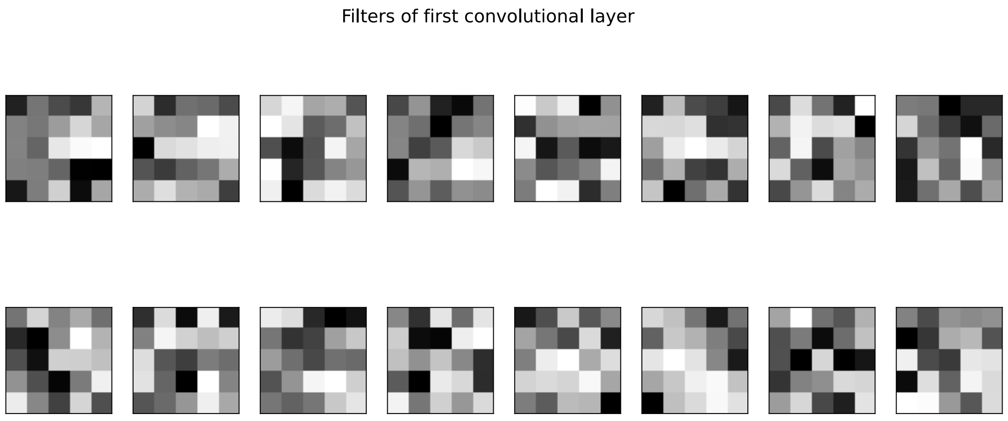

Figure 10: First convolutional layer filters.

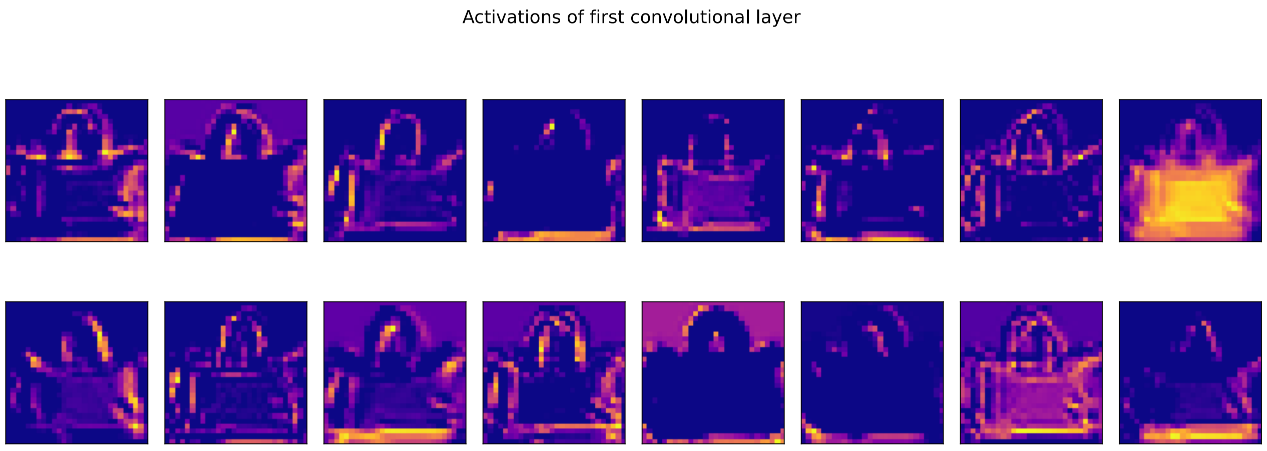

Figure 11: Activation profiles of the first convolutional layer.

4 Conclusion

In conclusion, I manage to solve two classical supervised learning tasks using neural networks. For the regression the simple Feed Forward Network was able to learn the target function f without fitting the noise (no overfiting) however it was not able to learn the part of f where no training data was present. For the classification task the hyperparameter optimisation allowed me to obtain a model that reach an accuracy above 90 percent which is arguably high.

References

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

[2] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747, 2017.

[3] Marian Tietz, Thomas J. Fan, Daniel Nouri, Benjamin Bossan, and skorch Developers. skorch: A scikit-learn compatible neural network library that wraps PyTorch, July 2017.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[5] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

A Appendix

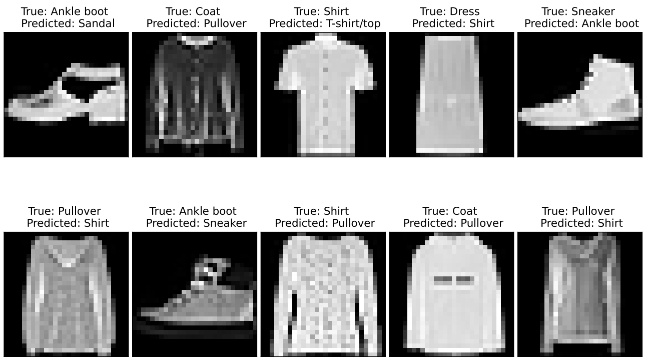

Figure 12: XXX

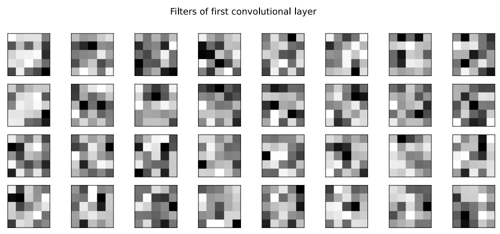

Figure 13: Second convolutional layer filters.

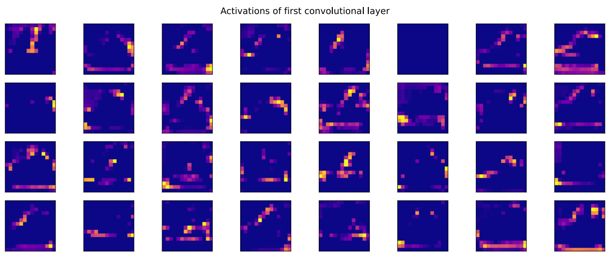

Figure 14: Activation profiles of the second convolutional layer.