Merve Tunçer 2040508

Neural Network and Deep Learning Homework 1

1) Regression Task

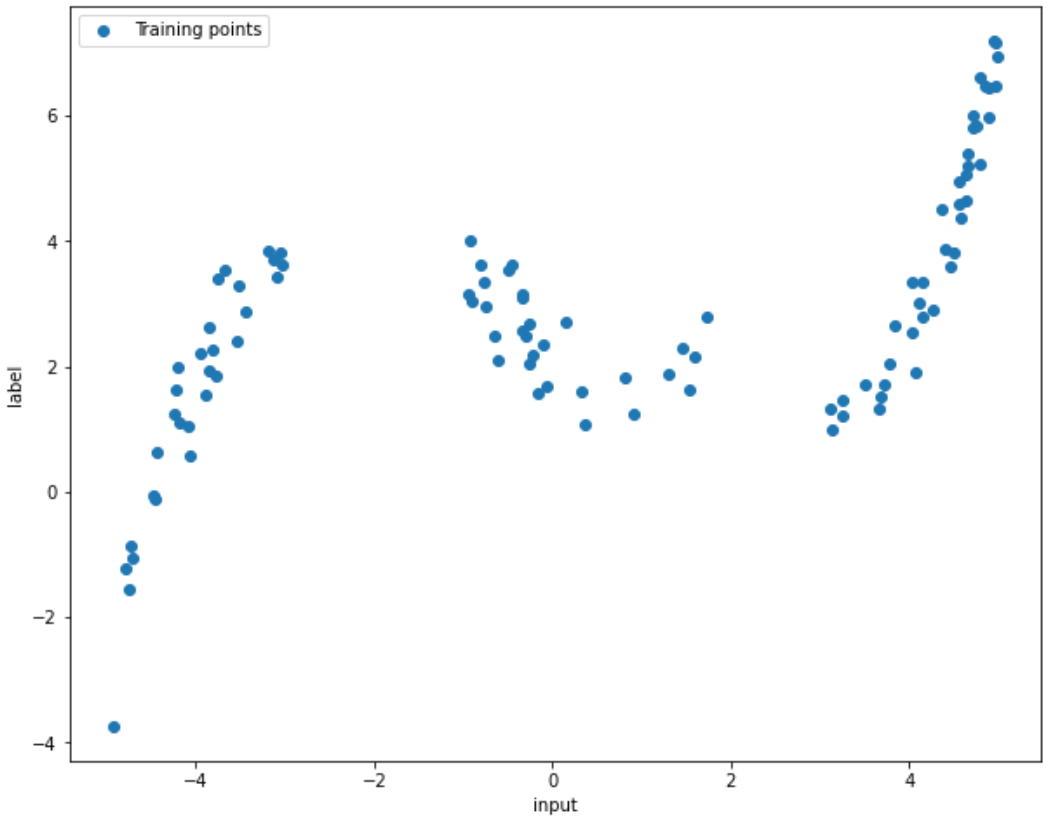
The goal is to train a neural network to approximate an unknown function:

$$f:\mathbb{R}\rightarrow\mathbb{R}$$
$$x\mapsto y=f(x)$$
$$\text{network}(x)\approx f(x)$$

As a training point, you only have noisy measures from the target function.

$$\hat{y}=f(x)+noise$$

The training set for this task consists in 100 points in the range of (-4.91, 4.97) for x train set and (-3.74, 7.19) for y train set.



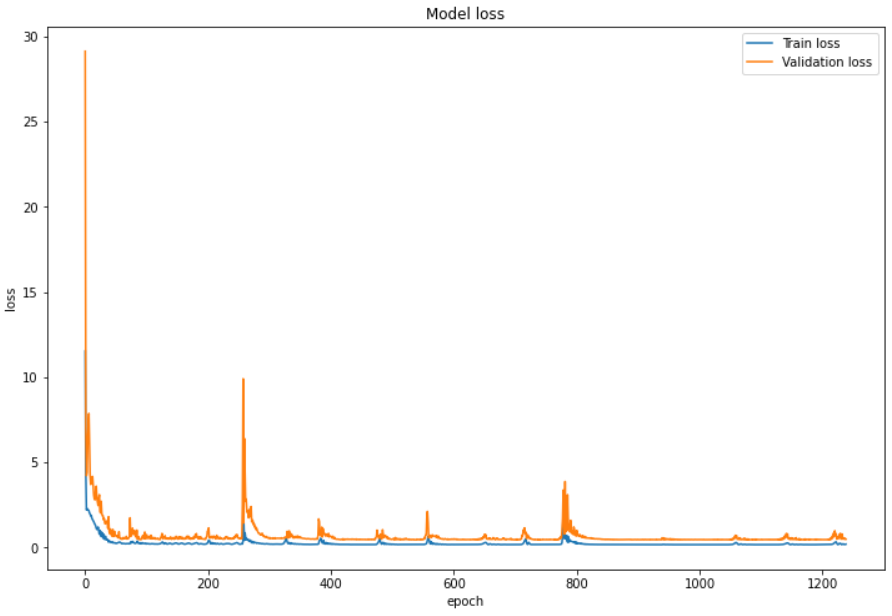
The figure shows that the training sets are missing around -2 and 2, as a result, even if data is lacking in those points, our regression model should be able to approximate the function. For the figure results, indicates that we will most likely need to utilize cross

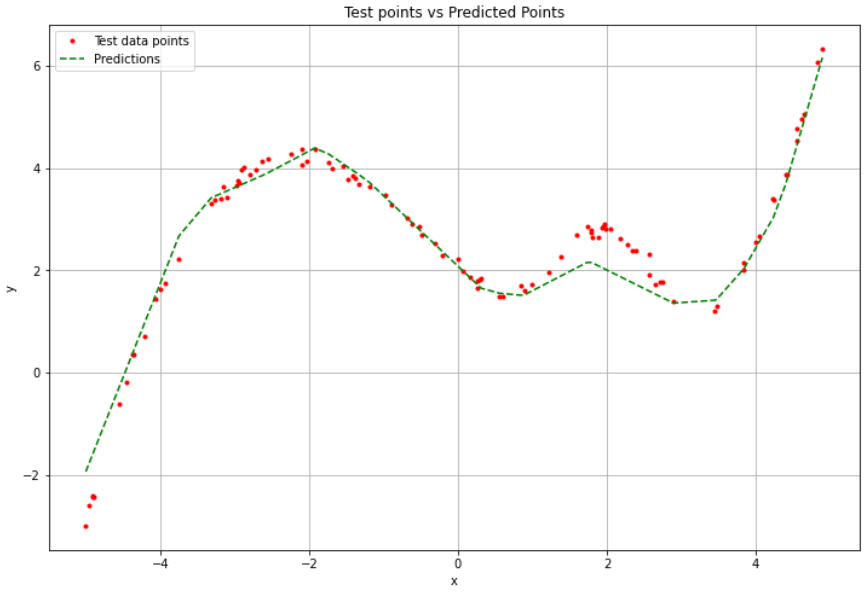
validation to avoid overfitting. I used 3 fully connected layers for network architecture. The skorch library, a sklearn package for pytorch, has been used for cross validation and grid selection. A grid search was performed on the hyperparameters in the table below with 3-fold cross-fold validation.

First layer number of neurons:	(8,16, 32, 48)
Second layer number of neurons:	(8,16, 32, 48)
Third layer number of neurons:	1
Max Epochs:	3000
Learning rates:	(0.1,0.01,0.001)
Regularization (L2 norm):	(1e-3, 1e-4, 1e-5 and 0)
Optimizer:	Adam
Early stopping:	100

The best parameters are {'max_epochs': 1500, 'module__Nh1': 16, 'module__Nh2': 8, 'optimizer__lr': 0.01, 'optimizer__weight_decay': 1e-05}. My loss results are;

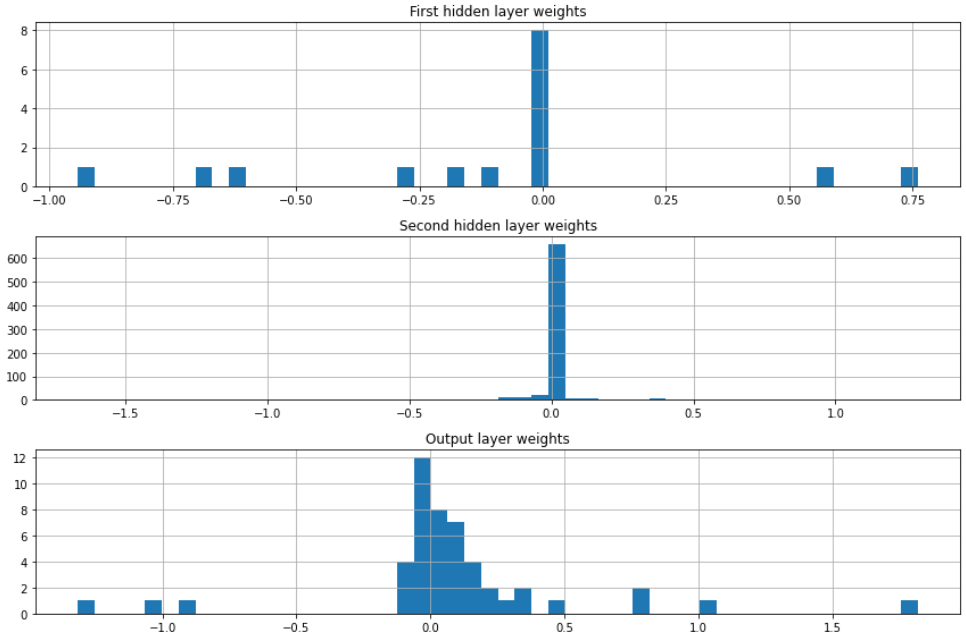
- Train Loss: 0.199
- Val Loss: 0.493
- Test Loss: 0.145

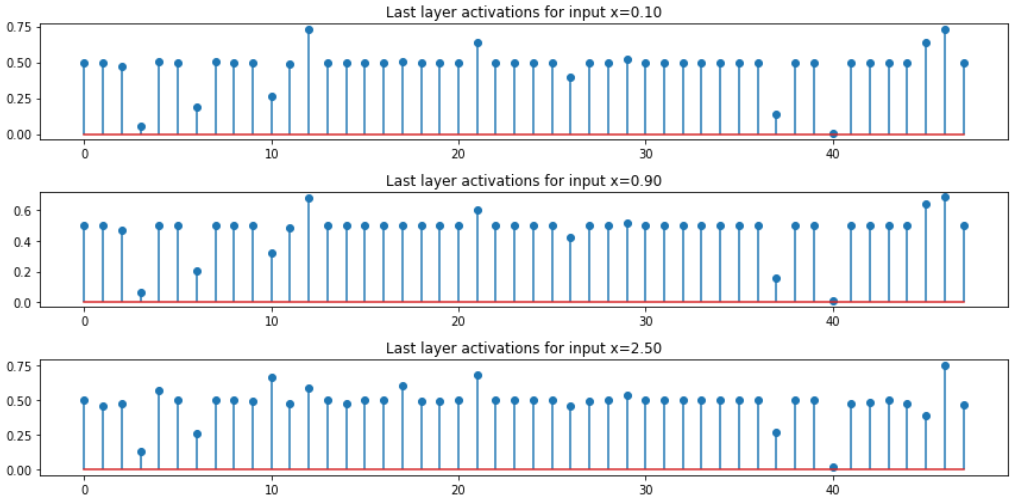




As can be seen above, the model generalized the function well. The loss function shows that the model is not overfitting and that it can predict values around the critical points of -2 and +2. This is a good insight since we can assume that by fine-tuning the hyperparameters or using a new model, the prediction will be more accurate.

Inputs are shown in the following weight histogram and activations of the second layer of the network. The weights of the first hidden layer are almost equitably spread in the -1 to 1 range. As a result, the first hidden layer has a small effect on the prediction. The second hidden layer has a range of -0.05 to 0.05 and is distributed near 0. The second hidden layer provides more prediction performance. The output layer's weights are mostly distributed around 0. When we look at the activations layers, we can see that almost all of the weights in each layer are active when input is presented.





2) Classification Task

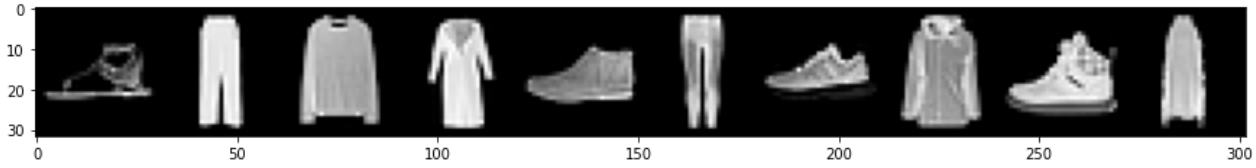
The task is a supervised learning problem using the FashionMNIST dataset. The classification task consists of a simple image recognition problem where the aim is to correctly classify images of Zalando's article images.

2.1) Dataset

The Fashion-MNIST dataset is proposed as a more challenging replacement dataset for the MNIST dataset. It is a dataset comprised of 60,000 small square 28×28 pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The list below shows the class that each number represents.

- 0: T-shirt/top
- 1: Trouser
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal
- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle bot

```
labels:  Sandal, Trouser, Pullover, Coat, Ankle Boot, Trouser, Sneaker,
Coat, Sneaker, Dress
```



2.2)Method and Results

The training set will be split into two parts using the random split function: 80% of it (48000 samples) will be utilized for the training process, while the remaining 20% will be used to validate each model tested in the random search.

```
Examples in Train Loader: 48000
Examples in Valid Loader: 12000
Examples in Test Loader: 10000
```

To reduce the computational cost of the learning process, the network uses two convolutional layers and two max-pooling layers to produce feature maps and subsequently reduce the image sizes.

The network structure is the following:

- one convolutional layer with 1 input channel, n_channels output channels, a 3 by 3 kernel, padding and stride set to 1 to keep the image size equal to 28 × 28 pixels;
- max pooling layer which halves the picture sizes (28 × 28 → 14 × 14);
- 2-dimensional dropout layer with dropout probability pdro2d, sampled uniformly in [0,0.25];
- another convolutional layer with n_channels input channels, 2·n_channels output channels, 5 by 5 kernel, padding= 0 and stride= 1, that reduces the size of the images to 10 × 10 pixels;
- another max-pooling layer which halves again the image sizes (10 × 10 → 5 × 5);
- one fully connected layer with 5 × 5 × 2 n_channels = 50 n_channels input units and 100
- outputs;
- another dropout layer (1 dimensional) with dropout probability pdro1d , sampled uniformly in [0, 0.25];
- fully connected layer with 100 inputs, 10 outputs (corresponding to the 10 labels).

Each layer is activated by a ReLU activation function, except the last one: the CrossEntropyLoss() loss function, which combines the LogSoftmax activation function and the negative log likelihood loss into a single class and thus performs the final classification, combines the LogSoftmax activation function and the negative log likelihood loss into a single class. I used random search for optimize hyperparameters. Random search methods are those stochastic methods that rely solely on the random sampling of a sequence of points in the feasible region of the problem, according to some prespecified probability distribution, or sequence of probability distributions.

In addition, certain regularisation and optimisation approaches are tried during the random search, with the optimiser chosen from among:

```
params = {"device"        : [device],
          "num_epochs"    : [n_epochs],
          "n_channels"    : np.arange(3,11),
          "pdrop1d"       : np.random.uniform(0, 0.25, 20),
          "pdrop2d"       : np.random.uniform(0, 0.25, 20),
          "optimizer"     : ['sgd', 'adam'],
          "momentum"      : np.random.uniform(0.5, 0.95, 20),
          "learning_rate" : loguniform.rvs(1e-5, 1e-2, size=20),
          "reg_param"     : loguniform.rvs(1e-4, 1e-1, size=20),
          }
```

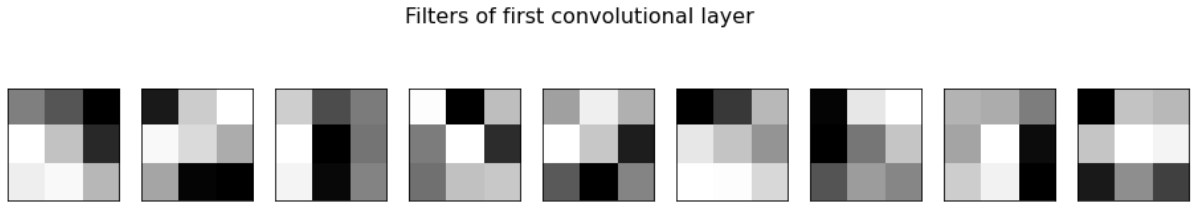
Given the size of the dataset, the training process can be computationally intensive and time consuming. To avoid unnecessary iterations of the training loop, an early stopping procedure is implemented in the main network class, which stops the training after the validation loss has reached a certain level of stability. The training loop is interrupted and learning terminates when the difference in absolute value between the validation loss computed at a given iteration and the average loss of the last 10 epochs is less than 0.005.

The best of network parameters found during a random search is as follows:
Best network parameters:

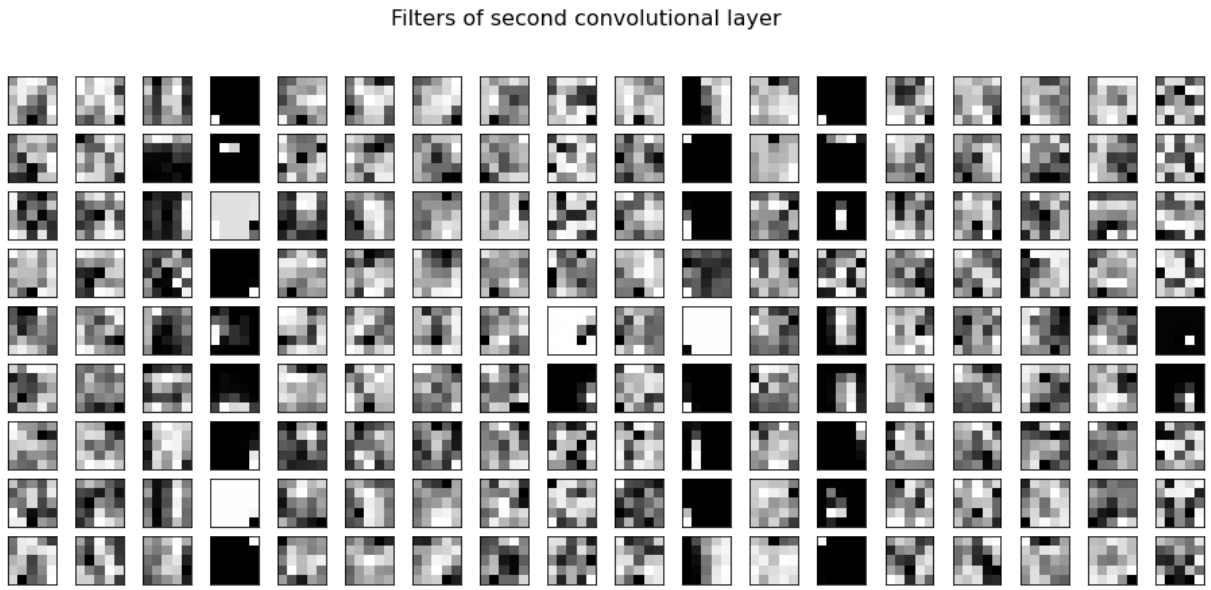
- 'num_epochs': 15.0,
- 'n_channels': 9.0,
- 'pdrop1d': 0.1662857764109922,
- 'pdrop2d': 0.14656503063798654,
- 'optimizer': 'adam',
- 'momentum': 0.626704227631114,
- 'learning_rate': 0.003197594206352734,
- 'reg_param': 0.0004986624637438257,
- 'final_loss': 0.25950103998184204,
- 'avg_train_loss': 0.31442153453826904,
- 'avg_valid_loss': 0.2916518449783325.

The final model accuracy result is %90 and the average test loss is 0.0021.

The first convolutional layer's filters can be found in Figure of first convolutional layer filters . The weights of each convolutional kernel are represented by n channels = 10, 3 * 3 images, as expected. Similarly, the second layer filters are shown in Figure of second convolutional layer filters: this time, the images are 5 * 5 and there are n channels* 2 n_channels = 200 of them, given the starting setting of the net.

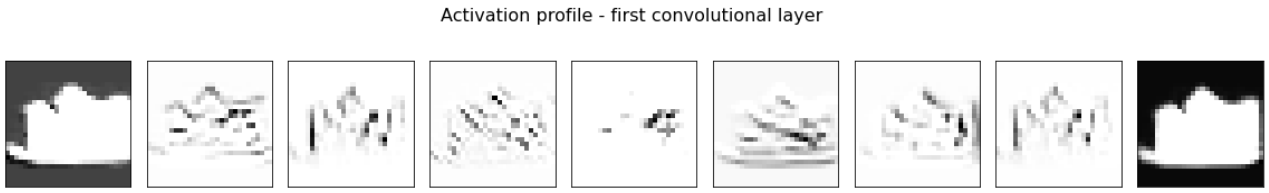


First convolutional layer filters for the CNN optimised for the classification task

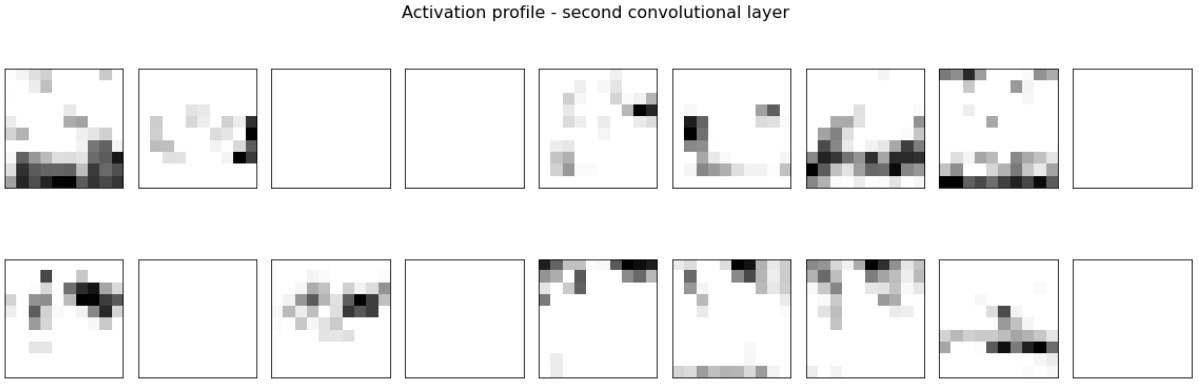


Second convolutional layer filters for the CNN optimised for the classification task

It's tough for an outsider to understand how the network learns just by looking at these figures. When the activation profiles are analyzed, however, it is easier to understand which features are more easily detected by the network. Each channel in Figure of activation profile clearly shows the input image in various shades of grey, showing that the first convolutional layer has already understood the input images main features. Instead, the second layer, shown in Figure of activaiton, focused on image decomposition, highlighting the input image's single features, such as the edges. Given the kernel sizes and the down sampling performed by the Max-Pooling layer, the picture size in this last figure is, as expected, 10*10 pixels.



Activation profiles for the first convolutional layer



Activation profiles for the first convolutional layer. The input image is decomposed and its features are enhanced and detected by the convolutional channels

I used k-fold cross validation with best parameters model. I trained model again with best parameters for using cross validation, I tried to understand which dataset best for train and test set. I used 5 fold for cross validation. Test sets fold results are as follows:

- Test set for fold 0: Average loss: 0.0024, Accuracy: 8854.0/10000.0 (89%)
- Test set for fold 1: Average loss: 0.0024, Accuracy: 8857.0/10000.0 (89%)
- Test set for fold 2: Average loss: 0.0024, Accuracy: 8851.0/10000.0 (89%)
- Test set for fold 3: Average loss: 0.0025, Accuracy: 8832.0/10000.0 (88%)
- Test set for fold 4: Average loss: 0.0024, Accuracy: 8917.0/10000.0 (89%)

The results are so similar with my test loss, actually my test loss is much better than folds results. Re-examining with more folds is necessary to obtain a better result. Lastly, I tried the VGG models for case. VGG models are a type of CNN Architecture proposed by Karen Simonyan & Andrew Zisserman of Visual Geometry Group (VGG), Oxford University, which brought remarkable results for the ImageNet Challenge. I used the source code **(source code: https://pytorch.org/vision/stable/_modules/torchvision/models/vgg.html).** Train with 10 epochs result is test accuracy: 0.926. In conclusion, both the implemented and optimized architectures performed effectively for the required tasks.