Using MLP for classification

Tomáš Belák

April 19, 2025

Abstract In this project, I have experimented with using the MLP (multilayer perceptron) architecture with one hidden layer for classification. The experiments were focused on the dimension of the hidden layer, number of epochs, activation function on the hidden layer, learning rate and learning rate scheduling, input normalization, weight initialization, and early stopping. In two steps, I trained 180 models with unique hyperparameter configurations on the estimation dataset (80% of the training dataset). Each model was evaluated on a validation dataset that consisted of 20% of the entire training dataset. The model that performed the best on the validation set was selected for full training achieving an almost perfect classification accuracy of 99.0% on the test set.

Contents

1 Introduction

This project explores the application of an MLP (multilayer perceptron) with a single hidden layer for classification tasks, focusing on systematically optimizing its performance through hyperparameter tuning.

The primary goal of this project was to investigate the impact of key hyperparameters on the performance of the MLP. These hyperparameters include the size of the hidden layer, learning rate, activation functions, weight initialization strategies, input normalization, learning rate scheduling, and early stopping. By conducting a two-phase grid search, 216 unique hyperparameter configurations were evaluated on a training dataset split into estimation and validation subsets. The best-performing model was then fully trained and tested, achieving near-perfect classification accuracy.

This report details the methodology, experiments, and results of this project, highlighting the effectiveness of systematic hyperparameter optimization in achieving high classification accuracy. The findings demonstrate the potential of MLPs as robust classifiers when carefully tuned for the task at hand.

2 Tools

The entire project was coded in Python (3.12.9). Package Numpy was heavily utilized for loading data, building MLP classifier and training, while the package matplotlib was utilized for visualization. All randomization in the project was done with the seed set to 24.

3 Data

In this section, I analyze the training and testing datasets provided for the project. The datasets consist of two features and a categorical target variable with three classes: A, B, and C. I compute the mean of the features, examine the representation of each class, and visualize the data distributions using plots.

3.1 Data overview

The training dataset (`2d.trn.dat`) contains 8000 samples, while the testing dataset (`2d.tst.dat`) contains a separate set of samples. Each sample consists of two features, denoted as x_1 and x_2, and a target class label. The target classes are represented as follows:

- Class A
- Class B
- Class C

3.1.1 Feature Statistics

The mean values of the two features, x_1 and x_2, were computed for both the training and testing datasets. The results are summarized in Table 1.

dataset	**x1 mean**	**x1 std**	**x2 mean**	**x2 std**
training	4.9136	7.8225	60.1260	15.7074
testing	4.6822	7.8478	59.7554	15.3957

Table 1: Mean and standard deviation of features x1 and x2 for training and testing datasets.

3.1.2 Class Representation

The distribution of samples across the three classes (A, B, and C) was analyzed for both datasets. Figure 1 shows the class representation in the training and testing sets. The datasets are imbalanced but the ratios of class representation are almost the same.

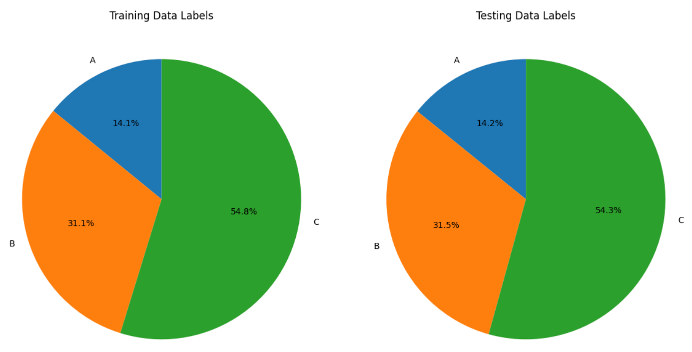

Figure 1: Class representation in the training and testing datasets.

3.1.3 Class scatter plots

On the following plots we can observe the nature of the data classes. The scatter plots are similar which is expected from datasets that come from the same probability distribution.

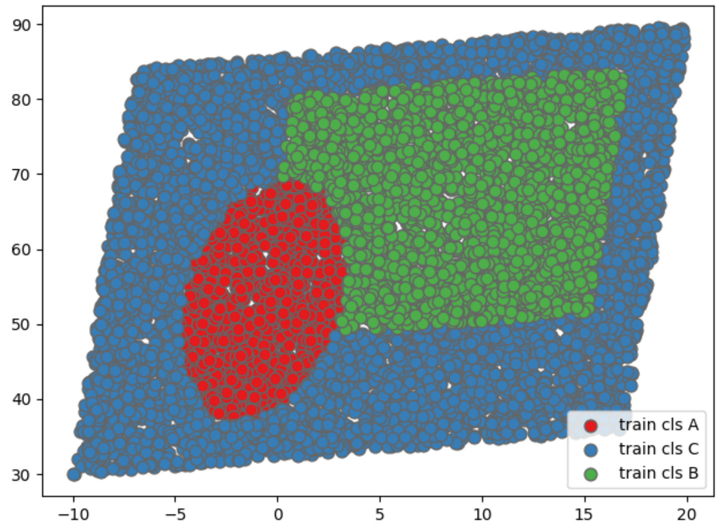

Figure 2: Scatter plot of the training dataset.

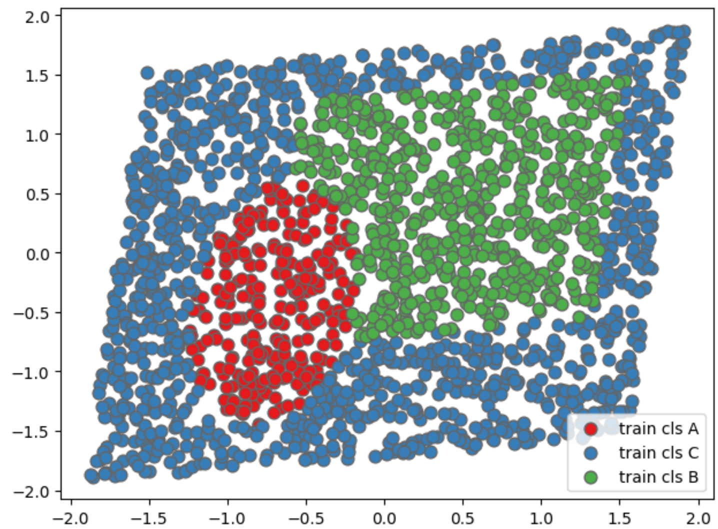

Figure 3: Scatter plot of the testing dataset.

3.2 Data preprocessing

The class labels were converted from characters to numerical values (0–2). Labels are then one-hot encoded internally when training a model.

3.3 Estimation and validation split

The training dataset was split into estimation and validation datasets in 80-20 ratio. Meaning out of the total 8000 rows the train set now contained 6400 rows. The validation set now contained the remaining 1600 rows. However, when splitting the dataset the original class ratios may have not been preserved due to the random shuffling of samples which in an extreme case could cause significant under-representation of some class and lead to under-performing models. This could be prevented by implementing a more robust splitter or using out of the box dataset splitting methods.

4 Model and training

I decided to build a MLP with only one hidden layer as it seemed sufficient because it was able to produce satisfactory results during initial model exploration. The MLP has 2 input neurons because of the input dimension. The hidden layer has a custom number of neurons. The next section will focus on finding the best number. The output layer produces a one-hot encoded class as the output.

The MLP models are trained online using the backpropagation algorithm without using any momentum strength.

4.1 Normalization

During initial model exploration it seemed that the normalization of inputs significantly improved the performance. Therefore all training is done on inputs normalized by subtracting the feature mean of the training set and divided by the feature standard deviation of the training set. Data is normalized before splitting therefore the validation set is also normalized. Testing set is also normalized by the mean and standard deviation of the training set.

4.2 Activation functions

In this project, I decdied to experiment with the follwing activation functions: tanh, ReLU, sigmoid and softmax. The search for the best hidden layer activation function will be the focus in the next section. For the hidden layer activation function, however, only tanh, ReLU and sigmoid are considered. As for the output layer activation function the softmax will be used as it is preferred when dealing with non-binary classification tasks with one-hot encoded outputs due to the fact that all elements of the output vector sum up to 1. It is important to mention that the softmax activation function is paired with the cross-entropy loss function, while the other three, if used as the output layer activation function, are paired with the classic mean squared error (MSE) loss function.

4.3 Weights initialization methods

I experimented also with weights initialization methods and scale of the weights. I implemented three different methods: Guassian, uniform and sparse. The Gaussian methods initializes weights randomly where each weight is sampled from the normal standard distribution which is optionally scaled by a scaling factor. The uniform method also initializes weights randomly but each weight is sampled from a uniform distribution on the interval [-scaling factor, scaling factor). Lastly, the sparse weights initialization method first initializes the weights using the Gaussian method, but with the probability of 1 - sparsity factor some weights get reset to 0.

4.4 Learning rate scheduling

In this project, I implemented two learning rate schedules: step decay and exponential decay. Step decay decreases the learning rate every given number of epochs by a drop factor from the interval (0, 1). Exponential decay decreases the learning rate with each epoch by multiplying it by an exponentially decaying factor, typically based on the current epoch and a specified decay rate. The idea with learning rate scheduling is to allow the learning rate to start higher and gradually decrease over time enabling faster convergence during early training and finer updates later on.

4.5 Early stopping

With the hope of preventing overfitting and reducing unnecessary computation, I implemented early stopping based on the classification error (CE) on a validation set. During training, the model periodically evaluates its performance on the validation data. If the validation classification error does not decrease by more than a small threshold delta for a certain number of consecutive epochs, specified by the patience parameter, training is halted early. The weights corresponding to the lowest observed validation CE are saved and restored at the end of training.

5 Hyperparameter search

To optimize model performance, I conducted a two-stage grid search to explore and tune the hyperparameters of the multilayer perceptron (MLP). The search was split into two phases to first identify effective architectural and training configurations, and then fine-tune advanced strategies such as learning rate scheduling and weight initialization.

5.1 First stage: Core hyperparameter tuning

The initial grid search focused on core hyperparameters that define the model's structure and general training behavior. The hyperparameters explored in this stage were:

- **Hidden layer size** (`dim_hid`): [5, 10, 20, 50]
- **Learning rate** (`alpha`): [0.01, 0.005, 0.001]
- **Number of training epochs** (`eps`): [50, 150, 250]
- **Input normalization**: [`True`]
- **Activation functions**:
  - Hidden layer: [`sigmoid`, `tanh`, `relu`]
  - Output layer: [`softmax`]

Early stopping with patience=15 and delta=0 was enabled for all configurations in this phase to prevent overfitting and mainly to reduce the training time for less effective hyperparameter configurations.

5.1.1 First stage: Results

In the table we can see the results of the first grid search. It is important to state again that for all of the models the softmax function was used as the output activation function and that in this stage early stopping was used with patience = 15 and delta = 0 to reduce computation time. Also the inputs were normalized. The best model hyperparameters from this stage are highlighted with yellow. These hyperparameters will be fixed for the second stage of the grid search for best hyperparameters. These hyperparameters were selected based on the lowest classification error on the validation dataset (Val CE).

Estimation/training classification error (Train CE) and validation error (Val CE) are given as percentages.

id	dim_hid	alpha	eps	hidden_f	Train CE	Train RE	Val CE	Val RE
1	5	0.010	50	sigmoid	4.781	0.049	4.875	81.426
2	5	0.010	50	tanh	7.891	0.063	7.062	95.953
3	5	0.010	50	relu	5.062	0.042	4.000	61.726
4	5	0.010	150	sigmoid	7.172	0.061	6.312	95.346
5	5	0.010	150	tanh	7.141	0.064	4.500	90.074
6	5	0.010	150	relu	30.531	0.192	31.000	309.031
7	5	0.010	250	sigmoid	4.328	0.041	3.562	63.731
8	5	0.010	250	tanh	10.891	0.088	8.625	127.182
9	5	0.010	250	relu	4.562	0.050	3.062	79.707
10	5	0.005	50	sigmoid	6.609	0.061	6.375	99.120
11	5	0.005	50	tanh	6.234	0.055	5.812	86.404
12	5	0.005	50	relu	2.328	0.022	2.375	37.137
13	5	0.005	150	sigmoid	4.578	0.048	4.688	78.966
14	5	0.005	150	tanh	4.266	0.040	4.312	65.031
15	5	0.005	150	relu	4.391	0.037	3.500	52.051
16	5	0.005	250	sigmoid	7.891	0.065	7.438	105.792
17	5	0.005	250	tanh	6.188	0.052	5.125	82.010
18	5	0.005	250	relu	3.172	0.030	3.062	45.711
19	5	0.001	50	sigmoid	25.312	0.187	22.812	290.009
20	5	0.001	50	tanh	8.016	0.072	7.875	121.265
21	5	0.001	50	relu	14.594	0.111	14.000	172.460
22	5	0.001	150	sigmoid	8.812	0.075	8.812	121.629
23	5	0.001	150	tanh	8.422	0.083	8.375	131.967
24	5	0.001	150	relu	5.562	0.062	4.875	96.997
25	5	0.001	250	sigmoid	8.453	0.069	8.500	111.578
26	5	0.001	250	tanh	6.328	0.056	5.875	90.637
27	5	0.001	250	relu	5.234	0.048	5.000	75.511
28	10	0.010	50	sigmoid	3.078	0.032	2.875	55.163
29	10	0.010	50	tanh	3.656	0.032	2.500	48.367
30	10	0.010	50	relu	3.047	0.024	1.438	27.329
31	10	0.010	150	sigmoid	2.562	0.029	2.250	48.394

Continued on next page

id	dim_hid	alpha	eps	hidden_f	Train CE	Train RE	Val CE	Val RE
32	10	0.010	150	tanh	2.594	0.026	2.438	43.000
33	10	0.010	150	relu	2.812	0.024	2.188	29.425
34	10	0.010	250	sigmoid	2.359	0.030	2.000	49.152
35	10	0.010	250	tanh	3.719	0.040	2.812	63.888
36	10	0.010	250	relu	3.891	0.031	2.000	32.168
37	10	0.005	50	sigmoid	3.625	0.042	3.125	69.152
38	10	0.005	50	tanh	4.312	0.038	3.812	60.489
39	10	0.005	50	relu	2.344	0.022	1.625	33.057
40	10	0.005	150	sigmoid	3.969	0.048	3.250	79.397
41	10	0.005	150	tanh	3.547	0.032	2.938	53.849
42	10	0.005	150	relu	2.094	0.017	1.500	28.113
43	10	0.005	250	sigmoid	2.750	0.034	2.438	57.834
44	10	0.005	250	tanh	3.609	0.048	3.312	78.666
45	10	0.005	250	relu	2.828	0.029	2.000	45.921
46	10	0.001	50	sigmoid	8.312	0.095	8.188	154.656
47	10	0.001	50	tanh	7.812	0.065	7.438	104.489
48	10	0.001	50	relu	3.250	0.039	2.500	61.362
49	10	0.001	150	sigmoid	4.906	0.055	4.625	90.323
50	10	0.001	150	tanh	2.609	0.036	2.500	62.172
51	10	0.001	150	relu	1.891	0.027	1.812	45.775
52	10	0.001	250	sigmoid	4.641	0.054	4.562	89.272
53	10	0.001	250	tanh	3.188	0.036	3.062	60.582
54	10	0.001	250	relu	2.359	0.025	2.250	40.745
55	20	0.010	50	sigmoid	2.750	0.030	2.000	49.142
56	20	0.010	50	tanh	2.266	0.020	1.812	30.986
57	20	0.010	50	relu	2.531	0.020	1.688	23.542
58	20	0.010	150	sigmoid	2.000	0.024	1.875	39.306
59	20	0.010	150	tanh	1.984	0.019	1.562	30.171
60	20	0.010	150	relu	2.859	0.023	1.625	26.739
61	20	0.010	250	sigmoid	2.297	0.030	2.062	52.040
62	20	0.010	250	tanh	2.328	0.025	1.875	40.883
63	20	0.010	250	relu	2.969	0.026	1.938	31.574
64	20	0.005	50	sigmoid	3.234	0.038	2.625	60.713
65	20	0.005	50	tanh	1.734	0.021	1.875	35.548
66	20	0.005	50	relu	2.094	0.018	1.375	26.545
67	20	0.005	150	sigmoid	2.203	0.027	1.688	43.801
68	20	0.005	150	tanh	1.781	0.022	1.750	38.282

Continued on next page

id	dim_hid	alpha	eps	hidden_f	Train CE	Train RE	Val CE	Val RE
69	20	0.005	150	relu	2.344	0.024	1.375	36.637
70	20	0.005	250	sigmoid	2.578	0.032	2.125	53.255
71	20	0.005	250	tanh	2.594	0.030	2.250	49.332
72	20	0.005	250	relu	1.516	0.014	1.062	19.666
73	20	0.001	50	sigmoid	7.625	0.083	7.812	135.811
74	20	0.001	50	tanh	2.703	0.043	1.875	71.233
75	20	0.001	50	relu	2.062	0.028	1.938	46.348
76	20	0.001	150	sigmoid	3.922	0.050	3.500	82.811
77	20	0.001	150	tanh	2.500	0.035	2.062	59.852
78	20	0.001	150	relu	1.531	0.024	1.062	39.628
79	20	0.001	250	sigmoid	3.750	0.046	3.125	76.274
80	20	0.001	250	tanh	3.109	0.045	2.250	75.962
81	20	0.001	250	relu	2.312	0.028	2.062	46.629
82	50	0.010	50	sigmoid	2.094	0.026	1.750	42.796
83	50	0.010	50	tanh	1.844	0.017	1.375	26.233
84	50	0.010	50	relu	2.500	0.020	1.438	21.404
85	50	0.010	150	sigmoid	2.109	0.024	1.750	40.350
86	50	0.010	150	tanh	2.172	0.022	1.625	36.082
87	50	0.010	150	relu	3.219	0.027	1.875	32.218
88	50	0.010	250	sigmoid	1.375	0.019	1.312	31.498
89	50	0.010	250	tanh	1.875	0.018	1.562	29.303
90	50	0.010	250	relu	2.406	0.020	1.438	25.480
91	50	0.005	50	sigmoid	2.672	0.033	2.750	55.539
92	50	0.005	50	tanh	1.594	0.018	1.438	31.150
93	50	0.005	50	relu	1.609	0.015	1.250	22.478
94	50	0.005	150	sigmoid	2.641	0.033	1.875	53.811
95	50	0.005	150	tanh	2.031	0.023	1.812	37.079
96	50	0.005	150	relu	1.797	0.016	1.250	23.511
97	50	0.005	250	sigmoid	2.438	0.030	1.938	50.509
98	50	0.005	250	tanh	1.484	0.015	1.188	26.463
99	50	0.005	250	relu	1.875	0.018	1.188	25.486
100	50	0.001	50	sigmoid	5.812	0.066	4.875	107.814
101	50	0.001	50	tanh	2.125	0.032	2.125	54.443
102	50	0.001	50	relu	1.734	0.023	1.500	40.123
103	50	0.001	150	sigmoid	3.656	0.046	3.062	75.514
104	50	0.001	150	tanh	2.500	0.031	2.188	52.742
105	50	0.001	150	relu	1.422	0.023	1.125	38.108

Continued on next page

id	dim_hid	alpha	eps	hidden_f	Train CE	Train RE	Val CE	Val RE
106	50	0.001	250	sigmoid	3.266	0.042	2.750	68.544
107	50	0.001	250	tanh	1.766	0.026	1.812	44.492
108	50	0.001	250	relu	1.906	0.025	1.562	43.144

5.2 Second stage: Fine-tuning experimental hyperparameters

After selecting a promising base configuration, a second grid search was performed to fine-tune more advanced training techniques and initialization strategies. This included:

- **Learning rate scheduling**:
  - `exponential_decay`: decays learning rate continuously with each epoch. As for the decay factor, gamma = 0.1 was experimented with.
  - `step_decay`: reduces learning rate at fixed epoch intervals. As for the drop rate and epoch drop, drop rate = 0.8 and epoch drop = 15 was experimented with.
  - `None`: fixed learning rate
- **Weight initialization methods**: [`gauss`, `uniform`, `sparse`]
- **Weight scale factor**: [1.0, 2.0]
- **Sparsity** (for sparse initialization): [0.1, 0.2]
- **Early stopping variations**:
  - Enabled with `patience = 15`, `delta = 0`
  - Enabled with `patience = 15`, `delta = 0.001`
    the `delta` value was selected because 0.001 seemed to my naked eye to be an order of magnitude average improvement of validation CE between epochs.
  - Disabled

This second stage allowed more nuanced control over the training dynamics and weight initialization, enabling the model to converge more efficiently and potentially reach better generalization performance.

5.2.1 Second stage: Results

In the table we can see the results of the second grid search. It is important to state again that for all of the models the best core hyperparameters were used:

- **dim_hid**: 20
- **alpha**: 0.005
- **eps**: 250
- **hidden_f**: relu

and again the softmax function was used as the output activation function and the inputs were normalized. The best model hyperparameters from this stage are highlighted with yellow. These hyperparameters will be used for training on the entire training dataset. For the purpose of being able to fit the table on to the width of the page the follwing mapping of early stopping options and learnin rate schedule options was established:

Early Stopping Options (e_stop)

- **1**: `stop_early=True`, `patience=15`, `delta=0`
- **2**: `stop_early=True`, `patience=15`, `delta=0.001`
- **3**: `stop_early=False`

Learning Rate Schedule Options (lr_sch)

- **1**: `decay='exponential_decay'`, with `decay_rate=0.01`
- **2**: `decay='step_decay'`, with `drop=0.8`, `epochs_drop=15`
- **3**: No learning rate schedule (`decay=None`)

Estimation/training classification error (Train CE) and validation error (Val CE) are given as percentages.

id	w_init	sparsity	w_scale	lr_sch	e_stop	Train CE	Train RE	Val CE	Val RE
1	gauss	NaN	1.000	1	1	1.859	0.024	2.000	41.999
2	gauss	NaN	1.000	1	2	1.500	0.023	1.625	42.250
3	gauss	NaN	1.000	1	3	1.547	0.023	1.688	39.742

Continued on next page

id	w_init	sparsity	w_scale	lr_sch	e_stop	Train CE	Train RE	Val CE	Val RE
4	gauss	NaN	2.000	1	1	1.641	0.020	1.625	34.595
5	gauss	NaN	2.000	1	2	1.891	0.021	1.500	35.235
6	gauss	NaN	2.000	1	3	1.172	0.021	1.500	36.728
7	uniform	NaN	1.000	1	1	1.438	0.024	1.812	43.003
8	uniform	NaN	1.000	1	2	2.062	0.027	1.812	45.353
9	uniform	NaN	1.000	1	3	2.219	0.027	2.500	45.569
10	uniform	NaN	2.000	1	1	2.391	0.027	2.000	46.549
11	uniform	NaN	2.000	1	2	1.531	0.022	1.562	38.415
12	uniform	NaN	2.000	1	3	1.781	0.023	1.812	39.320
13	sparse	0.100	1.000	1	1	6.188	0.057	6.125	94.800
14	sparse	0.100	1.000	1	2	4.156	0.059	4.250	100.865
15	sparse	0.100	1.000	1	3	4.078	0.043	4.312	72.960
16	sparse	0.100	2.000	1	1	8.375	0.070	8.562	115.141
17	sparse	0.100	2.000	1	2	9.766	0.085	9.250	140.509
18	sparse	0.100	2.000	1	3	28.984	0.216	29.812	356.259
19	sparse	0.200	1.000	1	1	2.531	0.030	2.312	51.966
20	sparse	0.200	1.000	1	2	3.969	0.039	4.250	65.266
21	sparse	0.200	1.000	1	3	2.031	0.028	2.062	46.821
22	sparse	0.200	2.000	1	1	40.047	0.255	33.312	392.916
23	sparse	0.200	2.000	1	2	2.094	0.027	1.625	45.186
24	sparse	0.200	2.000	1	3	2.016	0.030	2.562	49.253
25	gauss	NaN	1.000	2	1	1.562	0.017	1.375	28.776
26	gauss	NaN	1.000	2	2	1.719	0.017	1.500	29.857
27	gauss	NaN	1.000	2	3	0.469	0.010	0.625	18.941
28	gauss	NaN	2.000	2	1	1.031	0.012	0.812	21.209
29	gauss	NaN	2.000	2	2	1.531	0.017	1.062	30.251
30	gauss	NaN	2.000	2	3	0.828	0.011	1.562	22.168
31	uniform	NaN	1.000	2	1	1.359	0.016	1.375	27.911
32	uniform	NaN	1.000	2	2	1.266	0.015	1.000	23.864
33	uniform	NaN	1.000	2	3	0.547	0.010	1.000	19.117
34	uniform	NaN	2.000	2	1	2.953	0.028	2.125	44.267
35	uniform	NaN	2.000	2	2	1.297	0.016	1.000	26.673
36	uniform	NaN	2.000	2	3	0.609	0.010	1.000	19.268
37	sparse	0.100	1.000	2	1	3.406	0.030	2.688	47.043
38	sparse	0.100	1.000	2	2	4.359	0.039	4.250	65.559
39	sparse	0.100	1.000	2	3	47.609	0.272	49.188	440.148
40	sparse	0.100	2.000	2	1	3.344	0.031	3.312	50.348

Continued on next page

id	w_init	sparsity	w_scale	lr_sch	e_stop	Train CE	Train RE	Val CE	Val RE
41	sparse	0.100	2.000	2	2	2.344	0.026	1.500	37.869
42	sparse	0.100	2.000	2	3	2.359	0.025	2.188	41.813
43	sparse	0.200	1.000	2	1	2.250	0.023	2.000	36.738
44	sparse	0.200	1.000	2	2	2.594	0.024	2.312	38.080
45	sparse	0.200	1.000	2	3	1.125	0.016	1.562	27.113
46	sparse	0.200	2.000	2	1	1.859	0.019	1.750	34.827
47	sparse	0.200	2.000	2	2	1.453	0.015	1.312	25.520
48	sparse	0.200	2.000	2	3	1.375	0.015	2.000	28.379
49	gauss	NaN	1.000	3	1	2.234	0.019	1.375	28.465
50	gauss	NaN	1.000	3	2	2.031	0.019	1.188	26.627
51	gauss	NaN	1.000	3	3	1.266	0.010	1.438	18.862
52	gauss	NaN	2.000	3	1	2.031	0.017	1.312	25.070
53	gauss	NaN	2.000	3	2	2.406	0.019	1.500	25.157
54	gauss	NaN	2.000	3	3	1.578	0.011	1.438	18.887
55	uniform	NaN	1.000	3	1	1.953	0.020	1.375	30.791
56	uniform	NaN	1.000	3	2	1.812	0.020	1.438	31.240
57	uniform	NaN	1.000	3	3	1.312	0.011	1.062	15.631
58	uniform	NaN	2.000	3	1	2.359	0.026	1.562	40.238
59	uniform	NaN	2.000	3	2	2.266	0.021	1.562	31.961
60	uniform	NaN	2.000	3	3	1.281	0.010	1.750	17.761
61	sparse	0.100	1.000	3	1	3.844	0.034	3.312	53.086
62	sparse	0.100	1.000	3	2	36.125	0.241	28.938	367.248
63	sparse	0.100	1.000	3	3	3.953	0.035	4.062	55.283
64	sparse	0.100	2.000	3	1	4.422	0.038	3.625	55.866
65	sparse	0.100	2.000	3	2	4.578	0.038	3.562	54.668
66	sparse	0.100	2.000	3	3	11.641	0.079	12.188	132.660
67	sparse	0.200	1.000	3	1	4.172	0.040	3.625	68.049
68	sparse	0.200	1.000	3	2	4.438	0.037	3.250	50.665
69	sparse	0.200	1.000	3	3	1.906	0.018	2.250	35.954
70	sparse	0.200	2.000	3	1	5.125	0.046	3.938	74.218
71	sparse	0.200	2.000	3	2	2.672	0.025	1.812	33.732
72	sparse	0.200	2.000	3	3	2.016	0.017	3.062	41.060

6 Best model testing

The best performing model on the validation dataset (0.625% Val CE) was found with hyperparameters:

- **dim_hid**: 20

- **alpha**: 0.005
- **eps**: 250
- **normalize**: `True`
- **hidden_activation**: `relu`
- **output_activation**: `softmax`
- **lr_schedule**:
  - `decay`: `step_decay`
  - `params`:
    - `drop`: 0.8
    - `epochs_drop`: 15

  In the figure 4 we can see how the starting learning rate alpha = 0.005 evolved during the full training of 250 epochs.

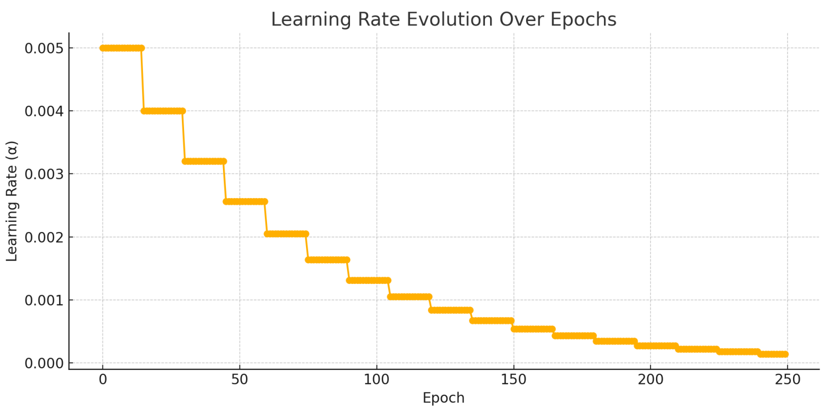

Figure 4: Best model predictions on the test data set.

- **weight_init**: `normal_dist`
- **weight_scale**: 1.0
- **early_stopping**:
  - `stop_early`: `False`

The model was then trained on the entire training dataset. Early stopping for this full training would be always disabled. In the figure 5 we can see how the final model predicted the outputs of the test dataset. In the figure 6 we can see the decision boundaries of the model and scattered test data as well.

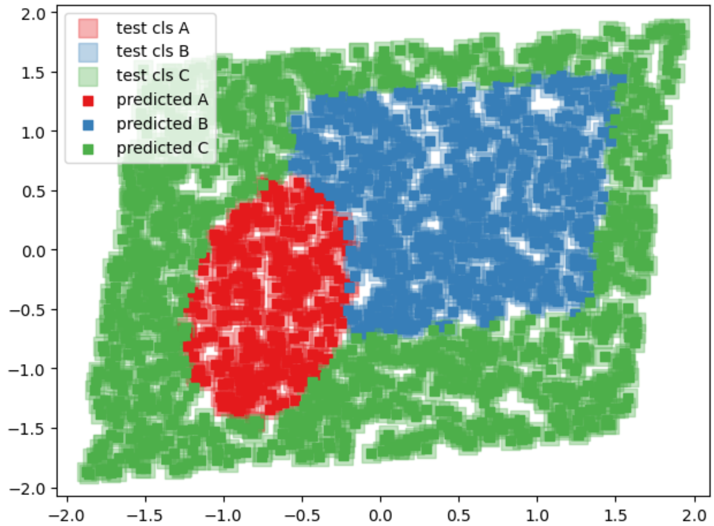

Figure 5: Best model predictions on the test data set.

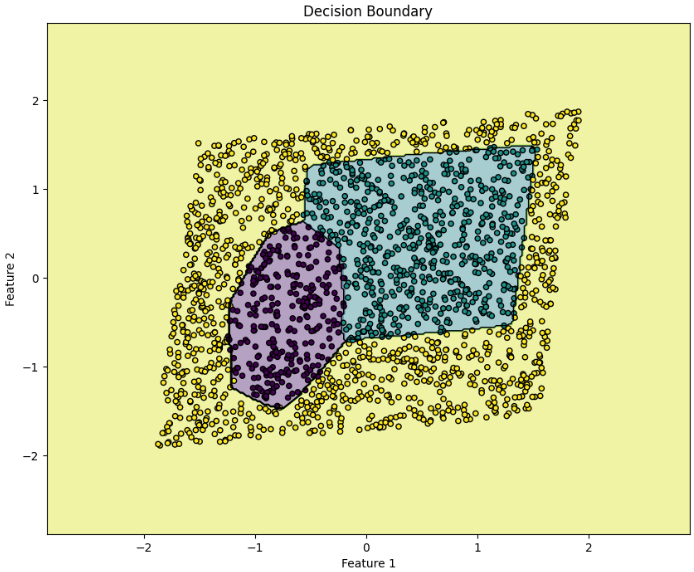

Figure 6: Decision boundaries of the best model with scattered test data.

The model achieved an almost perfect 99.0% accuracy on the test dataset. From the confusion matrix that can be found in the figure 7 we can see that on some occasions the model had difficulty distinguishing the least represented class A from classes B and C. The model also struggled to correctly classify some edge cases of class B and class C. This can be also seen in the figure 6.

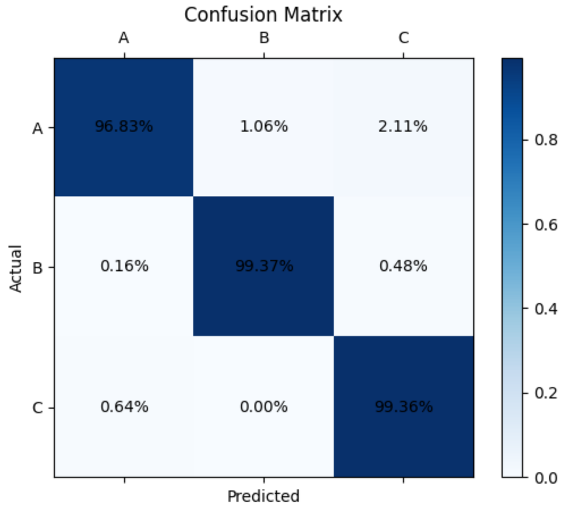

Figure 7: Best model confusion matrix on the test set.

In the figure 8 we can see both the classification error and the cross-entropy model loss. We can notice that the model fitted about 85% of the training dataset immaditely since the weight initialization and after that it rapidly improved and then the improvements became smaller as the learning rate decreased with every 15 epochs.

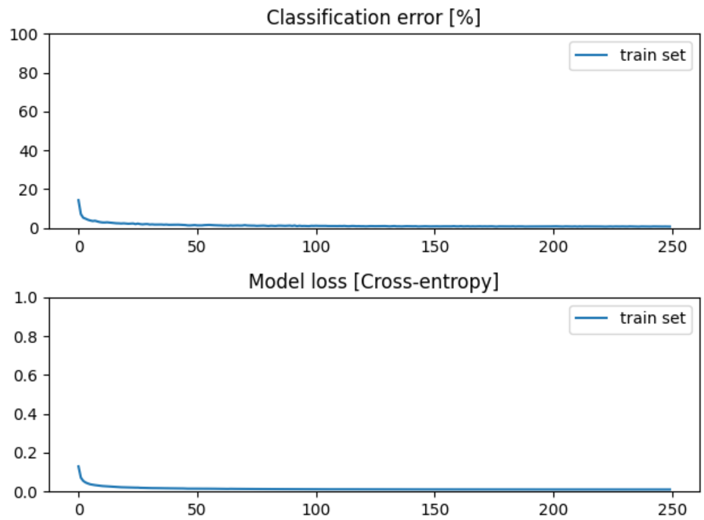

Figure 8: Error metrics while training the best model on the entire training set.

7 Conclusion

To sum up, this project tested a multilayer perceptron (MLP) with one hidden layer for a classification task. The experiments focused on several hyperparameters, such as the size of the hidden layer, number of training epochs, activation functions, learning rate and its scheduling, input normalization, weight initialization, and early stopping. In total, 180 models with different hyperparameter settings were trained online on 80% of the training data using the backpropagation algorithm and evaluated on the remaining 20%. The model with the best validation result was trained on the full dataset and reached a test accuracy of 99.0%. The best model used a hidden layer size of 20 neurons, learning rate of 0.005, ReLU activation in the hidden layer, softmax in the output layer, and normalized inputs. It also used normal distribution for weight initialization with a scale of 1.0 and a step decay learning rate schedule (decreasing the learning rate by 20% every 15 epochs). Early stopping was not used, and the model was trained for 250 epochs.

Appendix

Here is a link to my github repository with all the files needed: `https://github.com/tomasbelak24/ann-mlp-project`