Mulitlayer Perceptron

Gaspar Sekula

April 2025

Contents

1 Introduction

This laboratory project focuses on the multilayer perceptron (MLP), one of the fundamental models in the domain of neural networks. The main objective was not only to understand and build from scratch the mechanisms behind the effectiveness of such systems in solving prediction tasks, but more importantly to conduct an in-depth investigation into the impact of various hyperparameters on the learning process. The experiments carried out, along with the subsequent analysis, aimed to empirically validate the theoretical foundations of the MLP model.

2 Project overview

As part of the laboratory course, we progress through a series of tasks designed to develop a neural network from scratch, using "only" NumPy. The project begins with a basic implementation of a multilayer perceptron, where we experiment with different architectures and number of neurons to solve regression tasks on selected datasets. Subsequent stages introduce neural network training methods, including the implementation of backpropagation and optimization techniques such as momentum and RMSProp, aimed at improving learning efficiency. Additionally, we explore the impact of various activation functions on network performance and conduct experiments to identify and mitigate overfitting through the use of regularization mechanisms and early stopping techniques. Also, scaling method (Standard Scaler), one-hot encoding and gradient clipping are implemented. In each of the following paragraphs, the most interesting examples from all conducted experiments throughout the laboratories shall be provided. The code is available in repo. Note that as the implementation was progressing, the ideas were changing, so some notebooks may not be up to date.

3 Manual selection of network parameters

3.1 Experiments

Dataset	Steps Large	**Hidden Layers**	1
Neurons	5	**Activations**	5
Learning Rate	None	**Weights Init**	None

In this task, we tried to find the best parameters for Steps Large and Square Simple datasets. For demonstration purposes, the former one was chosen. Also, we examined three architectures: one hidden layer with 5 neurons, one hidden layer with 10 neurons and two hidden layers with 5 neurons each. As it can be verified in Figure 2, the random set of weights and biases is far from ideal. Therefore we need to improve the predictions by tuning weights and biases. Since the activation function for hidden layers is sigmoid and there are 5 neurons and there are three peeks in the dataset, we would like to use 3 neurons (3 sigmoids). The steepest part of the sigmoid shall be between the steps, at -0.5, 0.5 and 1.5. In addition, we want the steepest part of the sigmoid to be as steep as possible (coefficients in the first layer around 100 gives us MSE 22, 1000 - 3 and 10000 - 0.1). The best parameters for this architecture are: $w_{\text{hidden}} = [1000, 1000, 1000, 0, 0]$, $b_{\text{hidden}} = [500, -500, -1500, 0, 0]$, $w_{\text{output}} = [80, 80, 80, 0, 0]$, $b_{\text{output}} = [-800, 0]$.

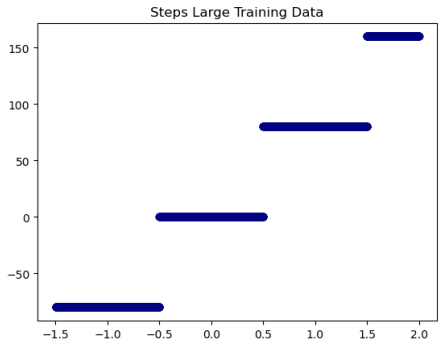

Figure 1: Steps Large dataset overview.

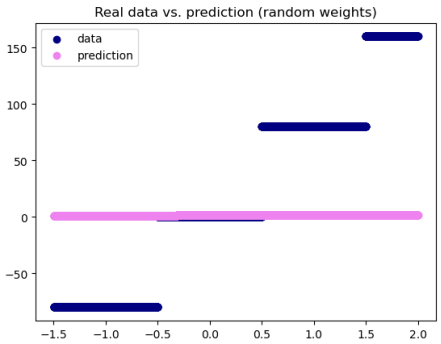

Figure 2: Predictions vs. true with random weights.

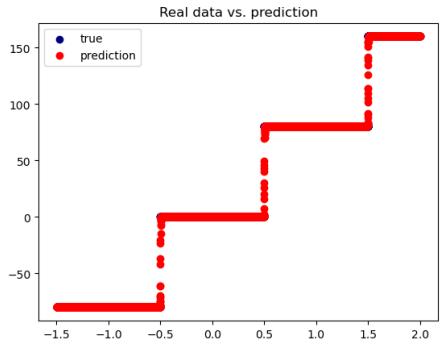

Figure 3: Predictions vs. true with manually set weights.

3.2 Conclusion

It is crucial to find the best weights and biases for the network. Sometimes it may be difficult to find them manually, e. g. when dataset consists of more features. Therefore we need an automation like backpropagation.

4 Backpropagation

4.1 Experiments

Dataset	Square Simple	**Hidden Layers**	1
Neurons	50	**Activations**	Sigmoid/Identity
Learning Rate	0.04	**Weights Init**	Xavier

In this task, backpropagation mechanism was implemented. We compared learning with mini batch to learning the whole dataset at once. I have examined three datasets in total: Square Simple, Steps Small and Multimodal Large. Let's draw conclusions based on the Square Simple dataset, as it leads us to similar conclusions as the Steps dataset. We were unable to perform proper Multimodal testing, as training time was super time-consuming. The examined batch size is 32.

Table 1: Performance Metrics With and Without Batches.

Random State	Time No Batches (s)	Time Batches (s)	Loss No Batches	Loss Batches
0	0.5951	1.8613	4.5019	0.7392
101	0.6697	1.7808	2.4105	0.3488
202	0.6094	1.8254	2.5790	0.3712
303	0.6042	1.7462	3.7012	0.7345
404	0.5917	1.7456	2.4233	0.3436
Mean	0.6148	1.7919	3.1238	0.5073
SD	0.0303	0.0470	0.8932	0.1985

As we can see in Table 1, training time with batches is significantly higher than without, but MSE is way lower. On Figure 4, we can see that training with batches converges way quicker than without batches.

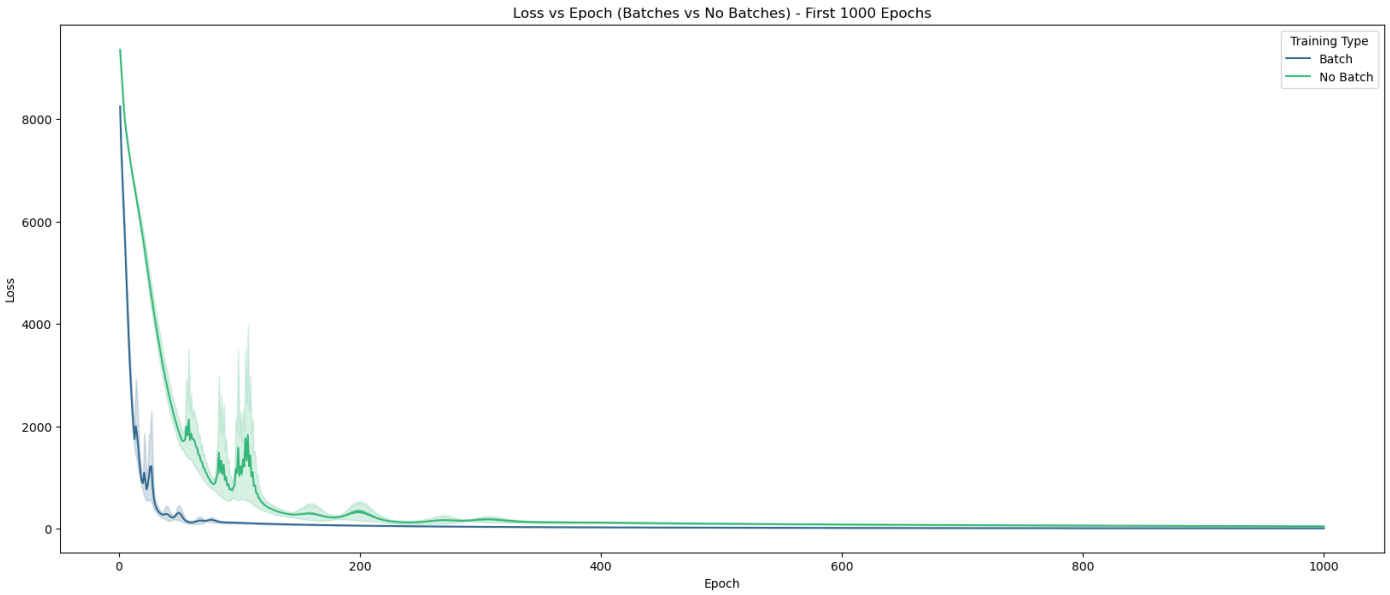

Figure 4: Loss convergence comparison (training with vs. without batches).

4.2 Conclusion

Mini batch training enables us to save memory and improve metrics, however it is time-consuming.

5 Momentum vs. RMSProp

5.1 Experiments

Dataset	Multimodal Large	**Hidden Layers**	1
Neurons	100	**Activations**	Sigmoid/Identity
Learning Rate	0.2/0.01	**Weights Init**	Xavier

In the previous part of the project, one could experience very long training time using standard SGD method. So here come two improvements to optimization algorithm: momentum training and RMSProp. The purpose of this task is to compare

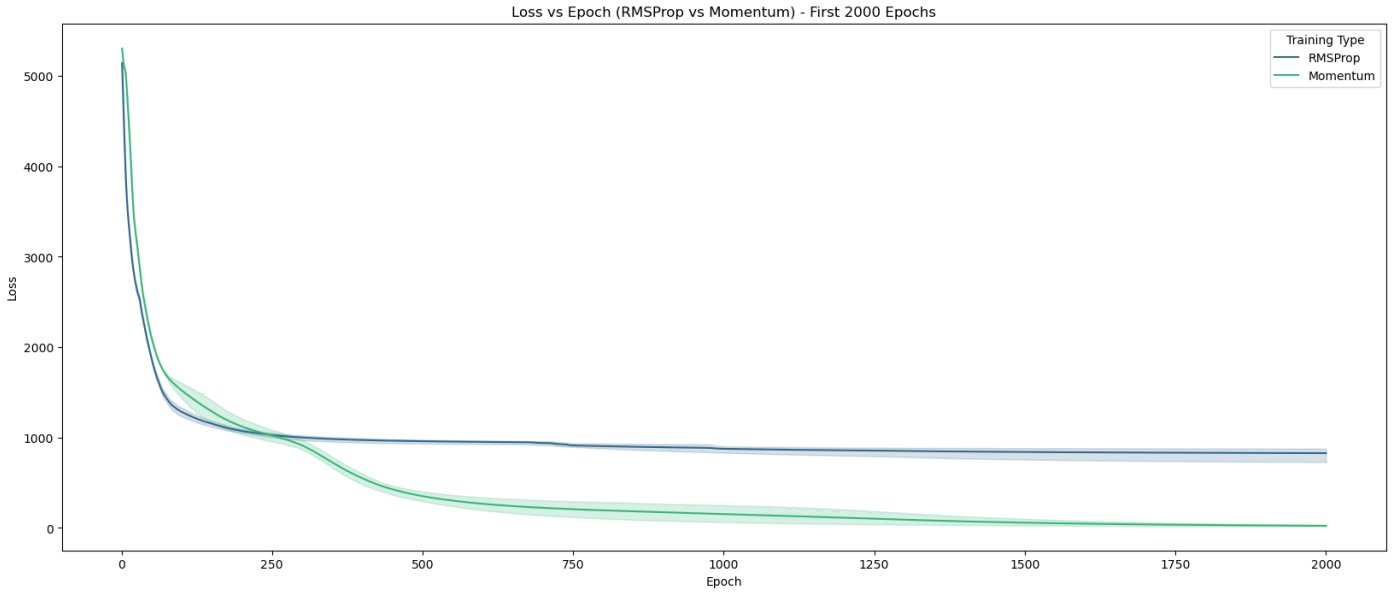

Figure 5: Loss convergence (Momentum vs. RMSProp).

the two solutions. In order to properly do so, we have compared the same architectures (layers structure, activations, parameters initialization) and adjusted only learning rate (for each optimization variant separately) to achieve best MSE. The experiments were run on three datasets: Steps Large, Square Large and Multimodal Large. The most representative one is Multimodal Large. For this dataset, we have manually found appropriate learning rate: for RMSProp 0.2 and for momentum 0.01. Other hyperparameters (including RMSProp coefficient or weights decay) were not modified. For report purposes, the tests on Multimodal were conducted 5 times with different random seeds and the maximum number of epochs was reduced to 10 000 (or finishing training, when MSE = 8.99 is reached).

Table 2: Performance Metrics for RMSprop and Momentum.

Random State	**Time RMSprop (s)**	**Time Momentum (s)**	**Epochs RMSprop**	**Epochs Momentum**	**Loss RMSprop**	**Loss Momentum**
0	94.3672	54.7805	10000	5948	30.4356	8.9897
101	95.7337	42.1604	10000	4576	570.6248	8.9883
202	93.2505	37.0535	10000	4024	29.2897	8.9897
303	91.9143	37.7159	10000	4115	28.1741	8.9896
404	92.6997	46.6776	10000	5010	27.9014	8.9895
Mean	93.9937	43.6777	10000	4734.6	37.0855	8.9896
SD	1.5247	6.2557	0	664.0	221.8340	0.0005

From results presented in Table 2 and Figure 5, we can conclude that RMSProp converges slower and is more sensitive to random state than momentum (especially noticeable for random state = 101; Table 5). Also, RMSProp converges slowly compared to the alternative. The difference is so big that it makes momentum the preferable optimization method.

5.2 Conclusion

The momentum training method resulted in quicker convergence to desired MSE metric value. It could be seen that RMSProp is more sensitive to hyperparameters (learning rate, random state, etc.) than the momentum. All experiments have lead me to conclusion that preferable method would be momentum. What is more, the experiments in this part of the project have shown that it is always important to know and understand your datasets. Sometimes test dataset may include data points that are not close to the ones in train dataset. As a result, it is nearly impossible to reach reasonable metric on test dataset (see: Figures 6, 7, 8).

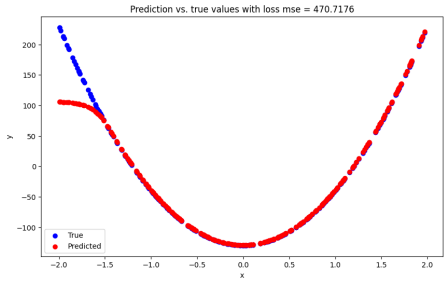

Figure 6: Square Large: Prediction with RMSProp.

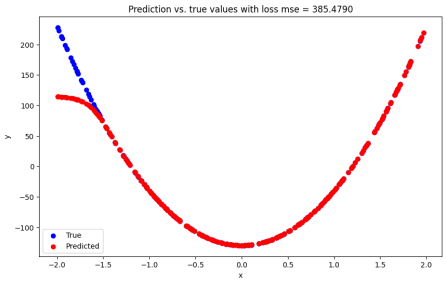

Figure 7: Square Large: Prediction with Momentum.

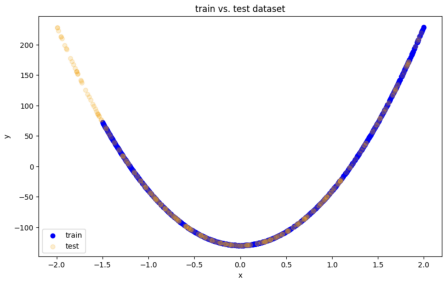

Figure 8: Square Large: Test vs. Train dataset.

6 Classification

As we have successfully gone through regression tasks on multiple datasets, it is high time we did classification task. So as to do so, we will use cross entropy loss function and F1-score evaluation metric.

6.1 Experiments

Dataset	Rings3 Regular	**Hidden Layers**	1
Neurons	10	**Activations**	LeakyReLU/Softmax/*Identity*
Learning Rate	0.05	**Weights Init**	He/Xavier

The experiments were performed on 3 datasets: Easy, XOR and Rings3. For reporting, we will analyze the latter one (visualized in Figure 9), as this dataset is the most complicated one.

Figure 9: Rings3 Regular Dataset overview.

The goal of the task was to compare efficiency using standard for classification output activation - softmax (cross entropy loss) with linear (mean squared error loss; predicting class's label, e. g. 0, 1 or 2 for Ring3 dataset). Let's assume that our desired F1-score is 0.9. For reporting, 5 iterations of training were performed with results in Table 3 and loss vs. epoch was visualized in Figure 10.
As we can see in Table 3 and Figure 10, the method without softmax struggles to converge to desired F1-score. On one hand, on average it needs 280 more epochs (or even more, as it does not reach F1 = 0.9 in epoch 1000) than the model with Softmax, while on the other hand, its training time is lower. What is more the results on test dataset indicate better performance with softmax.

6.2 Conclusion

Using softmax and cross entropy duo is preferred combination for output layer in classification task. Also, the experiments have shown that using scaling (here: Standard Scaler) significantly improves learning time.

Table 3: Comparison of Models With and Without Softmax.

Random State	Time SM (s)	Time No SM (s)	Epochs SM	Epochs No SM	F1 SM	F1 No SM	F1 Test SM	F1 Test No SM
0	0.3817	0.2405	791	1000	0.9005	0.6835	0.8890	0.7936
101	0.1664	0.2393	369	1000	0.9003	0.7694	0.8894	0.8534
202	0.3848	0.2404	851	1000	0.9004	0.8240	0.8895	0.8802
303	0.4640	0.2364	1000	1000	0.8794	0.7588	0.8470	0.8452
404	0.2723	0.2424	587	1000	0.9002	0.7793	0.8857	0.8518
Mean	0.3334	0.2398	719.6	1000	0.8962	0.7626	0.8801	0.8440
SD	0.0996	0.0021	251.1	0	0.0084	0.0496	0.0176	0.0304

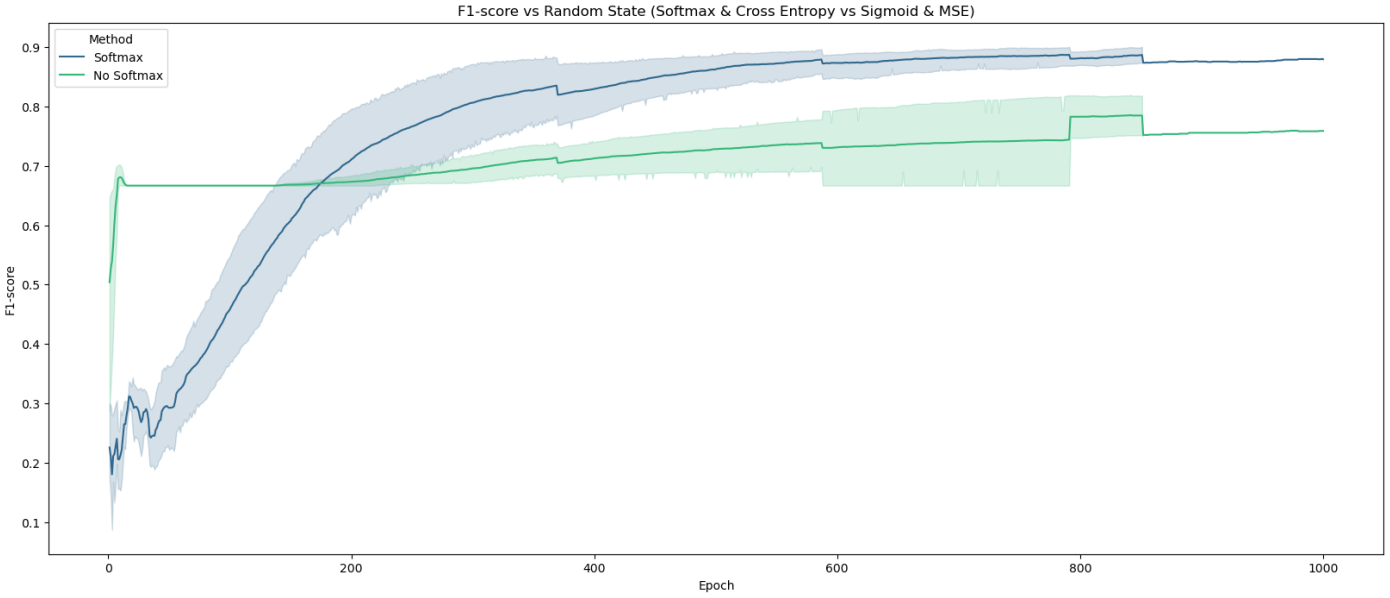

Figure 10: F1-score convergence with Softmax & Cross Entropy vs. with Identity & MSE.

7 Activation functions

7.1 Experiments

Dataset	Multimodal Large	**Hidden Layers**	var.
Neurons	var.	**Activations**	var.
Learning Rate	0.0001	**Weights Init**	var.

We would like to examine the impact of architecture choice on learning time and results. In order to choose the best model, we will examine performance on Multimodal Large dataset. The possible architectures include:

- number of hidden layers: 1, 2 or 3
- number of neurons in hidden layers: 2, 5, 10 or 20
- activations: Identity, Sigmoid, Tanh and ReLU

Therefore there are 48 possible combinations. From those we choose two best ones and verify their performance on other datasets. Wiser with knowledge from paragraph 5, we used the momentum optimization. Performing a couple of initial tests, the best learning rate was chosen (0.0001). Since with different random seeds, the results were similar, whole experiment was conducted once on all 48 architectures due to computation and time limits. The activations were selected according to recommendations from lecture (Xavier for Sigmoid or Tanh and He for ReLU).
The best converging architectures due to the activation function turned out to be the ReLU and Tanh (Figure 11). It is also no surprise, that the more neurons in architecture, the better MSE. Indeed, the best performing model were 2 x 20 and 3 x 10 with ReLU activation (Figure 12, 13). The poor outcome of the architecture 3 x 20 may be explained by a couple of reasons: wider networks have more complex loss surfaces with more local minima and saddle points; wrong learning rate for this architecture or gradient issues like vanishing. When it comes to identity activation, it results terribly and in the most complex network (3 x 20) explodes with NaNs (despite random seed). For visualizations in Figure 11 purposes, this sample was omitted. The predictions on test datasets are visualized in Figure 15). Right out of the gate one can name two best architectures.

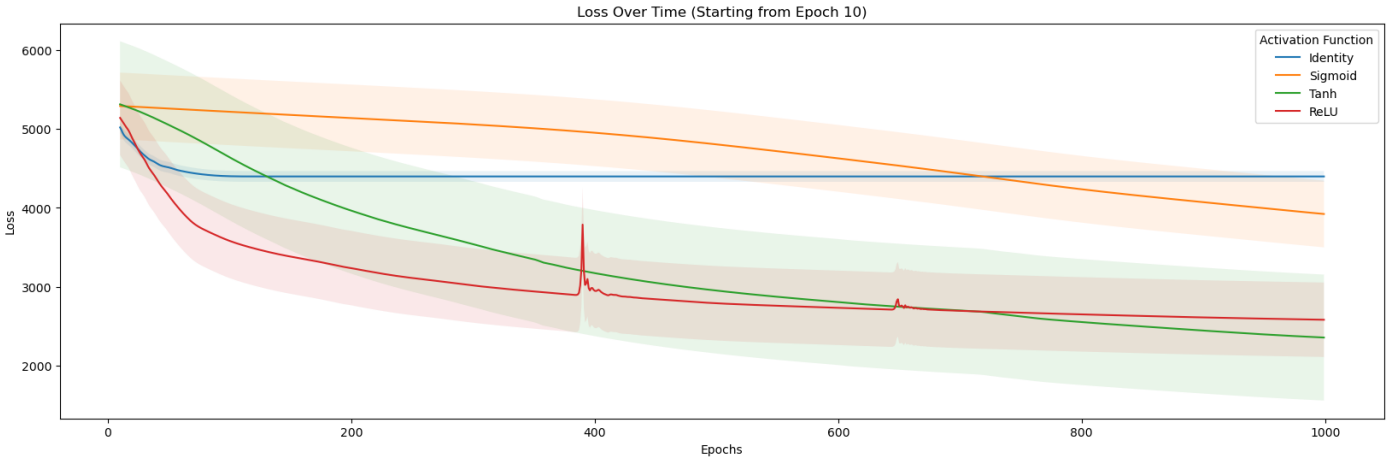

Figure 11: MSE over time.

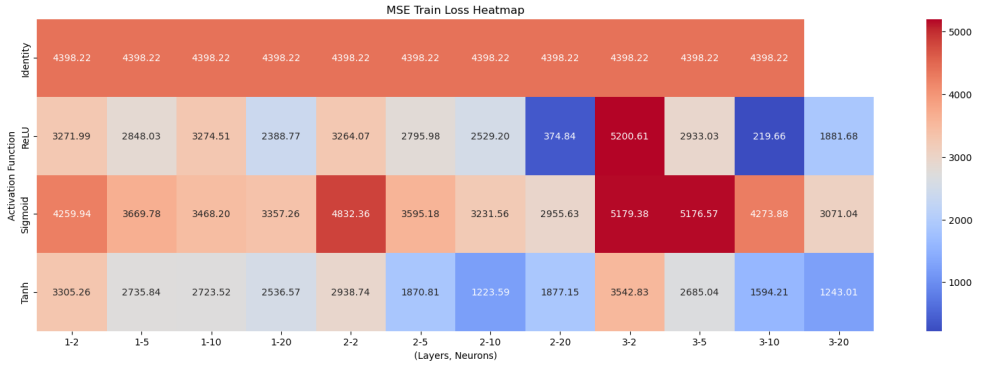

Figure 12: MSE on training dataset.

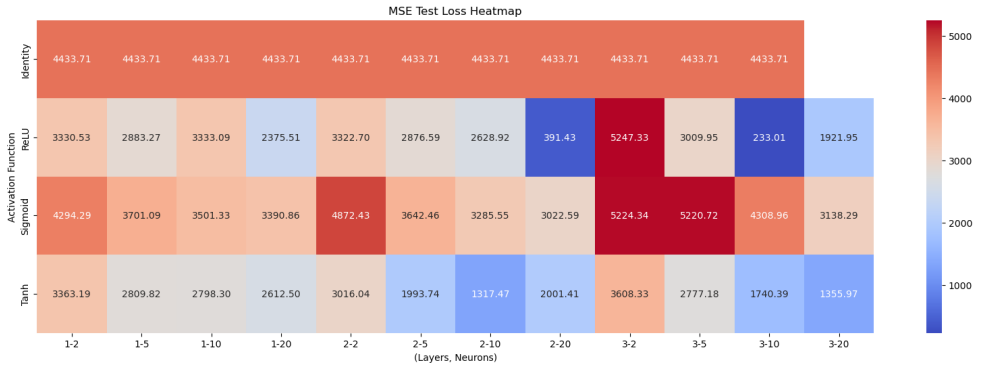

Figure 13: MSE on test dataset.

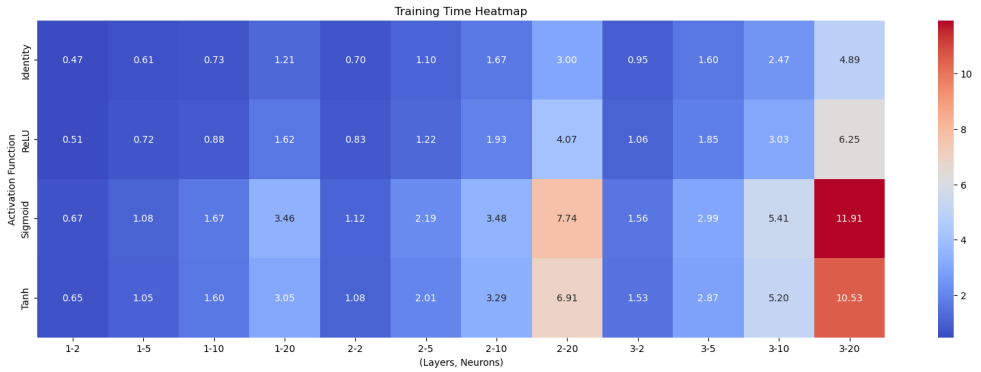

Figure 14: Training time.

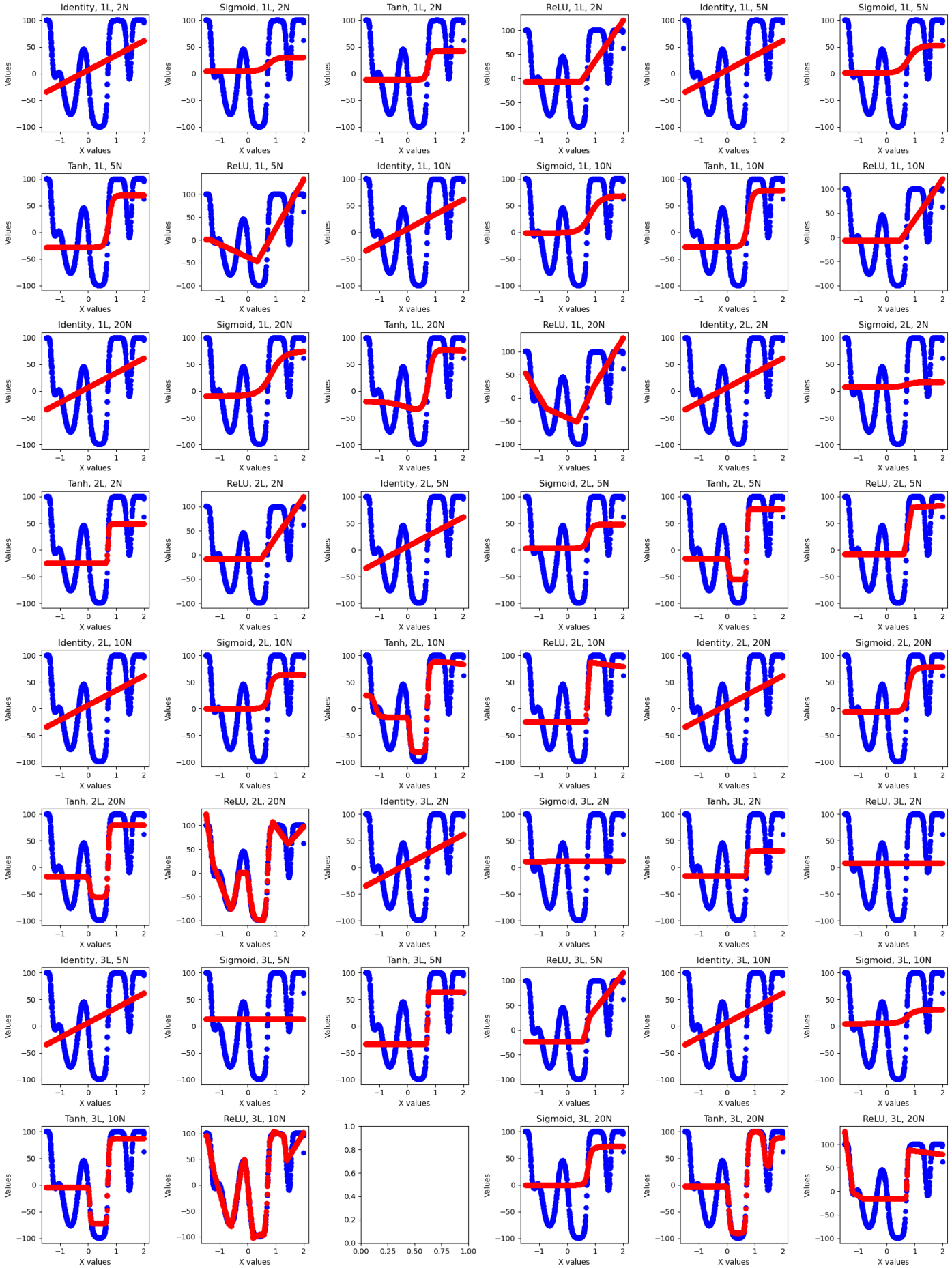

Figure 15: Prediction for different architectures on test dataset; blue - true, red - predicted.

7.2 Conclusion

In conslusion, there is a tendency that the more neurons, the better results (to some point) as well as Tanh and ReLU seem to be achieving better metrics than Sigmoid, not to mention Identity. The training time remains without surprise. The more complex the architecture, the longer the training (Figure 14).

7.3 Further experiments

Once we have chosen the best architectures, we would like to examine them on other datasets and different tasks.

7.3.1 Regression

Dataset	Steps large	**Hidden Layers**	3/2
Neurons	10/20	**Activations**	ReLU
Learning Rate	0.00015/0.0005	**Weights Init**	He

Here, we have adjusted the learning rate for this particular dataset in order to get the best results. We will compare the two architectures on regression task defined in Steps Large dataset (see: Section 3. Manual selection of network parameters).

Table 4: Training and Test Loss for Architectures A1 (3x10) and A2 (2x20).

Random State	Loss Train A1	Loss Train A2	Loss Test A1	Loss Test A2
0	21.7087	106.8445	19.1122	117.4719
101	212.9880	100.6825	215.2700	110.5318
202	116.3747	12.4211	126.7895	9.8834
303	10.3900	106.0653	8.3537	116.8417
404	8.8890	65.0800	6.7647	73.3078
Mean	74.8701	78.6187	75.2588	85.6073
SD	85.8874	45.8449	87.3272	47.1422

From results (Table 4) we cannot unambiguously determine, which architecture performs better, as the results strongly depend on random seed. Also, it is impossible to contrast solutions, as standard deviation for architecture 3 x 10 outnumbers its mean. However from convergence plot (Figure 16) we can conclude that 2 x 20 in general reaches lower MSE in first 4000 epochs.

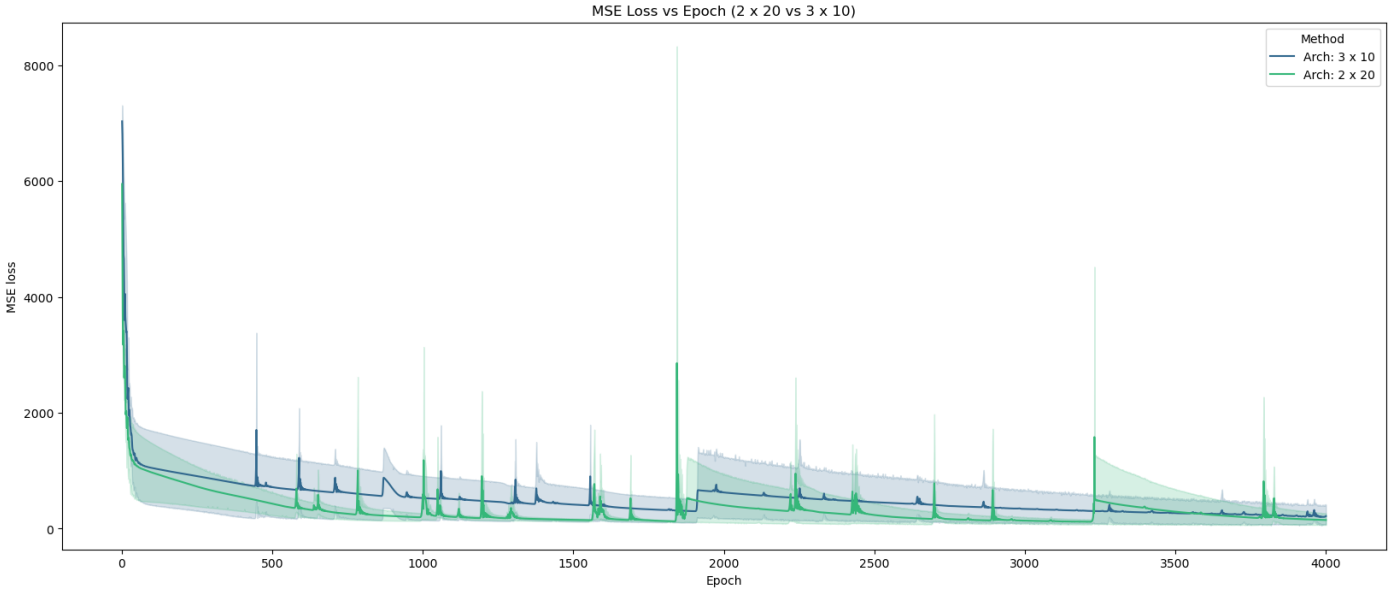

Figure 16: Convergence for architectures 3x10 and 2x20

We can visualize a sample prediction (Figure 17, 18). As the results are super-random (see: Table 4), we will use the well-beloved 42 random seed.

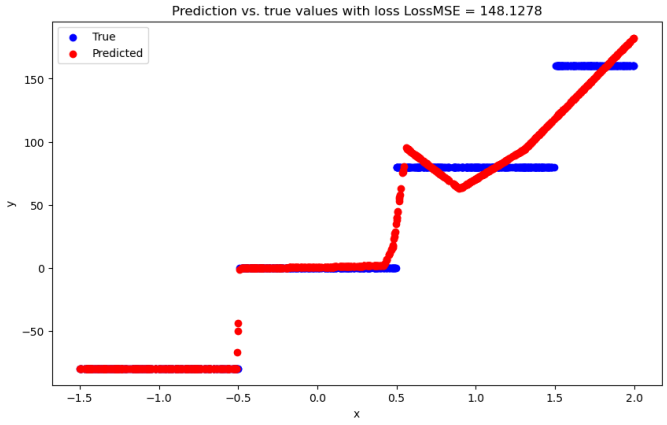

Figure 17: Square Large: prediction with 3x10 architecture.

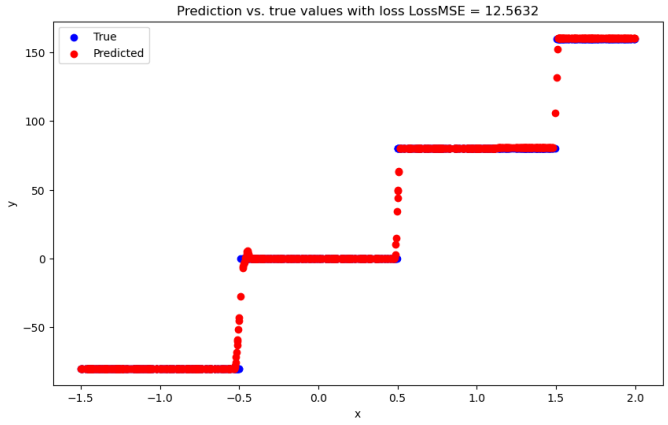

Figure 18: Square Large: prediction with 2x20 architecture.

7.3.2 Classification

Dataset	Rings5 Regular	**Hidden Layers**	3/2
Neurons	10/20	**Activations**	ReLU
Learning Rate	0.05	**Weights Init**	He

Here, we have adjusted the learning rate for this particular dataset in order to get the best results. We will compare the two architectures on classification task defined in Rings5 Regular dataset (Figure 19). In the experiment, we assume that 0.9 F1-score is sufficient for our needs and training will be finished, as metric reaches this value.

Figure 19: Rings5 Regular dataset overview.

As it can be concluded from Table 5, not only are results slightly better with the first architecture (within standard deviation), but also time is improved. Furthermore, Figure 20 shows that the architecture 3x10 converges faster than 2x20, especially in first 300 epochs.

Table 5: Training and Test F1 Scores and Time for Architectures A1: 3x10 and A2: 2x20.

Random State	Time A1 (s)	Time A2 (s)	F1 Train A1	F1 Train A2	F1 Test A1	F1 Test A2
0	0.256498	0.704952	0.900384	0.878551	0.845052	0.825183
101	0.196085	0.752848	0.900528	0.878129	0.867601	0.829222
202	0.159453	0.765904	0.900586	0.849885	0.842611	0.808164
303	0.558777	0.707687	0.900281	0.880568	0.839809	0.823124
404	0.083499	0.722416	0.901761	0.867124	0.846073	0.794889
Mean	0.250462	0.730361	0.900708	0.870451	0.848229	0.816916
SD	0.174716	0.025748	0.000535	0.012012	0.009443	0.013313

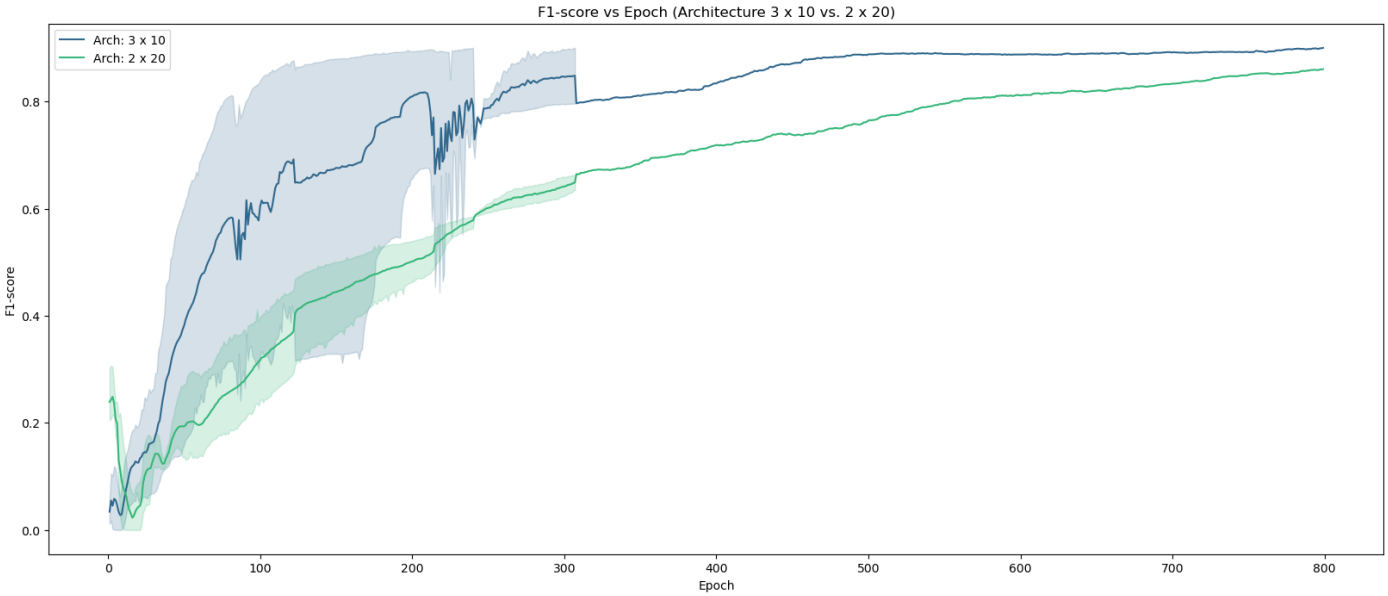

Figure 20: F1-score convergence for 3 x 10 and 2 x 20 architectures.

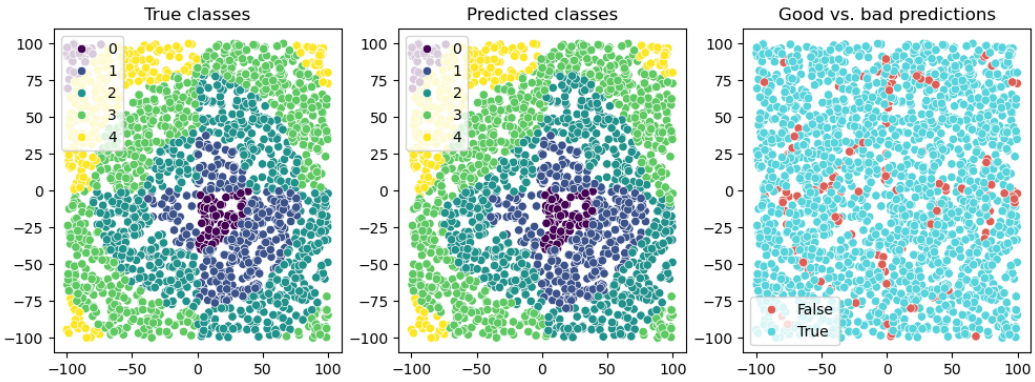

Figure 21: Example predictions on Rings 5 Regular Dataset with random seed = 42, F1 = 0.93.

8 Overfitting and regularization

8.1 Experiments

Now we would like to improve our implementation with regularization, so that it is ready for overfitting scenario. In addition, early stopping was implemented. The datasets, which will be examined are Multimodal Sparse (regression; Figure 22) and XOR Balance (classification; Figure 23). Their common part is the fact, that the test dataset contains significantly more data points than train. For classification task, there is noticeable class imbalance.

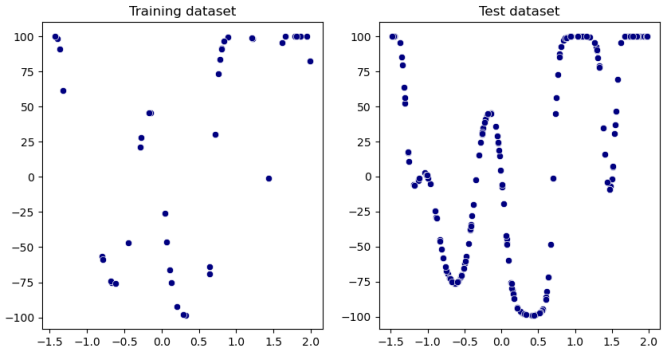

Figure 22: Multimodal Sparse dataset overview.

Figure 23: XOR Balance dataset overview.

8.1.1 Regression

The experiments were run in order to compare L1 and L2 regularization methods. The architecture consisted of one hidden layer with 100 neurons, Tanh activation and Uniform weights init. Similarly to the previous experiments, momentum

Dataset	Multimodal Sparse	**Hidden Layers**	1
Neurons	100	**Activations**	Tanh
Learning Rate	0.1	**Weights Init**	Uniform

optimization improvement was utilized. The tests were conducted 10 times with different random seeds.
The results in Figure 24 show mean loss over epoch for each method. It can be concluded that the best convergence on train dataset may be acquired by implementing standard (without regularization) method. However, the best results on test dataset were achieved with regularization (verified on random seed $\in \{41, 42, 43, 44, 45\}$; sample in Figure 25). The experiment did not lead to answer to question, which regularization benefits with lower MSE, as depending on seed, either L1 or L2 were preferable.

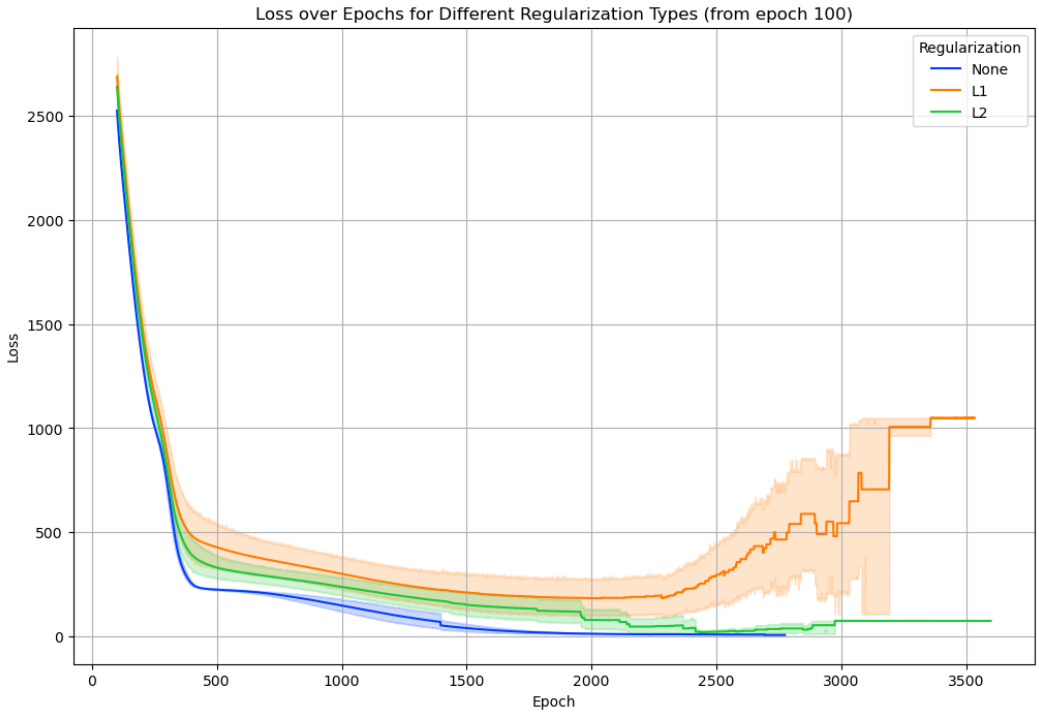

Figure 24: Regularization types' convergence on Multimodal Sparse Train dataset.

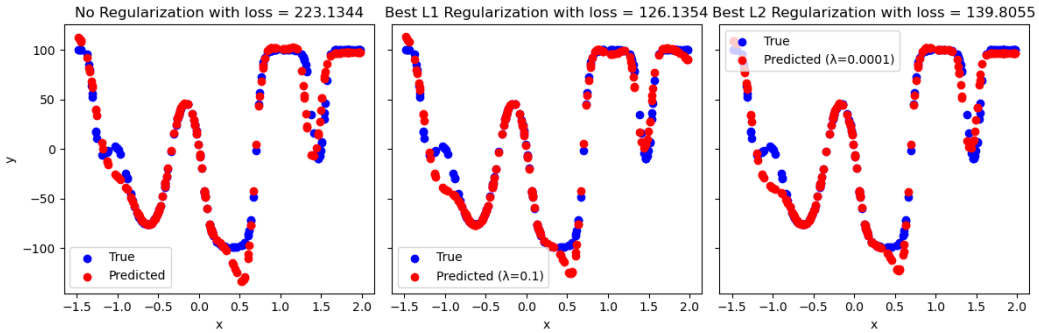

Figure 25: Predictions on Multimodal Sparse Train dataset with different regularization types and random seed = 42.

The nagging question remains: which λ (regularization parameter) is the best. All experiments (10 iterations with 10 different random seeds) were run with $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$. The boxplots in Figures 26, 27 lead to conclusion that in general the lower the lambda, the better the MSE. It is worth noting that Plot 24 may be biased by two factors: early stopping and high MSE values for lambdas 1 and 10.

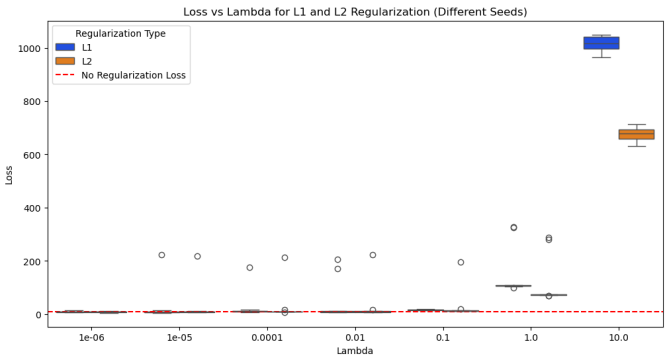

Figure 26: MSE vs. λ on Multimodal Sparse Train.

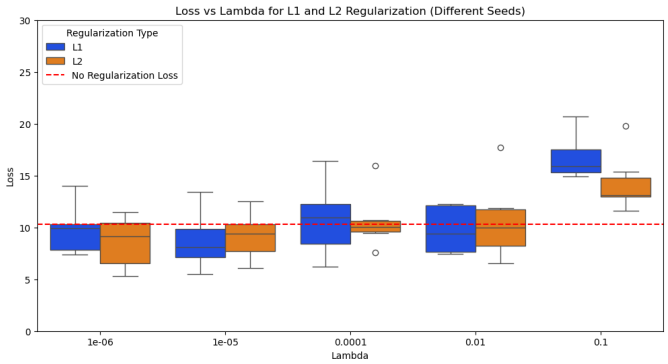

Figure 27: MSE vs. λ on Multimodal Sparse Train ($\lambda < 1$).

8.1.2 Classification

For classification task, we will examine the XOR Balance dataset, suffering from class imbalance issue.

Dataset	XOR Balance	**Hidden Layers**	2
Neurons	10	**Activations**	Leaky ReLU
Learning Rate	0.05	**Weights Init**	He, Xavier

The experiments were run in order to compare L1 and L2 regularization methods. The architecture consisted of 2 hidden layers with 10 neurons each, LeakyReLU activation and He weights init. Similarly to previous experiments, momentum optimization improvement was utilized. The tests were conducted 10 times with different random seeds.
The results in Figure 28 do not lead to the clear conclusion, as the trend is resemblant for all the methods. In epochs over 1500, the L2 regularization stands slightly out, in negative sense. The question, on which regularization parameter is best, leaves us with similar to previous example (regularization on regression task): the lower the λ, the better. We have observed that networks with regularization predict better than without it (on test datasets). As the trend is preserved for different seeds, in Figure 30 we show an example prediction with the following F1-scores (results shown for the best regularization parameter among all tested $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$):

- No regularization: 0.7573,
- L1 regularization ($\lambda = 10$): 0.7902,
- L2 regularization ($\lambda = 10$): 0.8809.

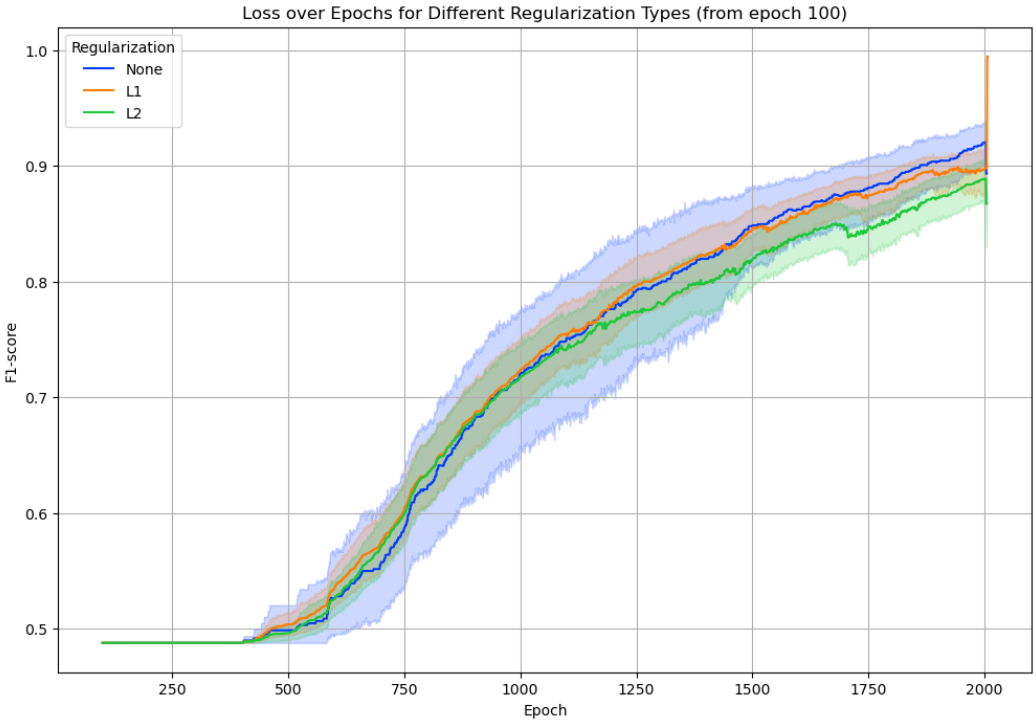

Figure 28: Regularization types' convergence on XOR balance Train dataset.

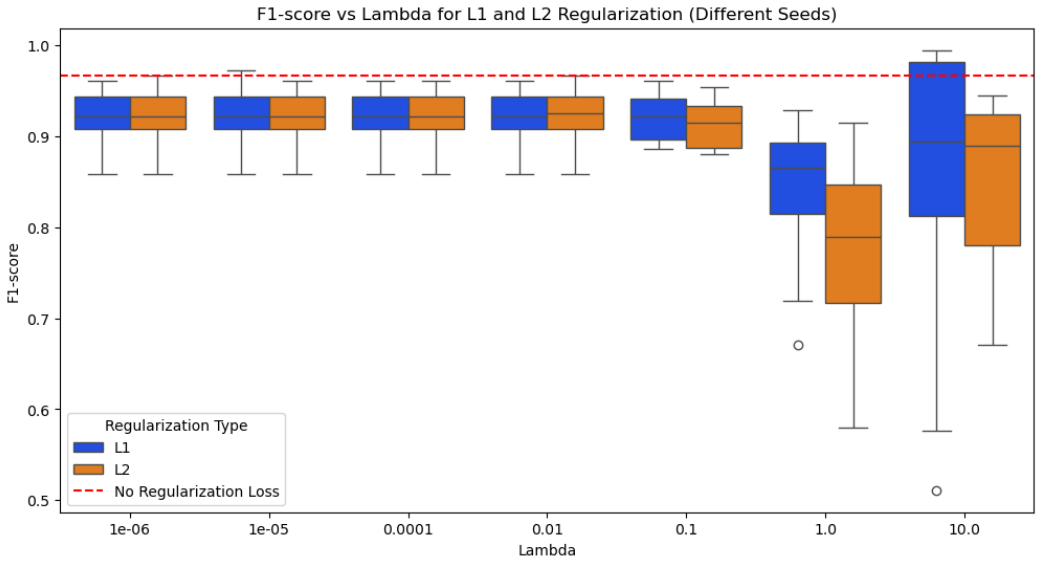

Figure 29: F1-score vs. λ on XOR Balance train dataset.

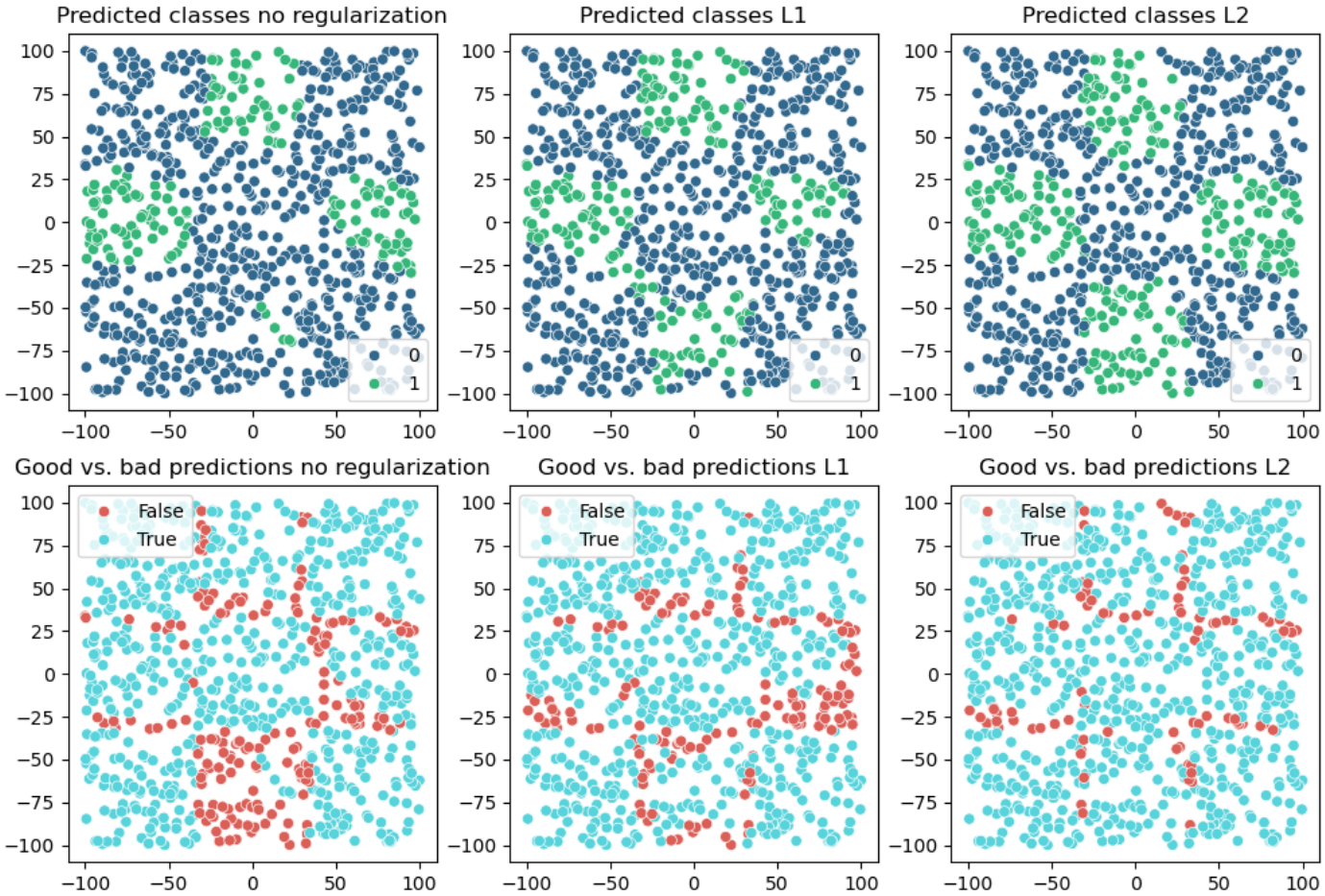

Figure 30: Predictions on XOR Balance Test dataset with different regularization types and random seed = 42.

8.2 Conclusion

In conclusion, it is important to wisely choose early stopping parameters (patience and delta), as wrong configuration may lead to premature training finish. From the conducted experiments, we have learned that usually the lower the regularization parameter, the better the results. What is more, sometimes despite the poor metric on train datasets, the L1 and L2 methods result in significantly better scores on test datasets.

9 Summary

All in all, multilayer perceptron was implemented as well as functionalities including: backpropagation training, momentum and RMSProp optimization algorithms, batch training, early stopping and regularization mechanisms alongside loss functions, one hot encoding, activations and standard scaler. Various training techniques, architectures and tasks were examined during both laboratory tasks and extra testing sessions for report. Key conclusions can now be summarized:

- Manual choice of weights and biases is sometimes possible, however in most cases it is an uphill battle.
- Mini batch training slightly improves metric and saves memory, but results in longer training time.
- Momentum optimization algorithm significantly improves training time and prevents ending up in local minima.
- Softmax and cross entropy duo is preferred combination for output layer in classification task.
- There is no best architecture for all tasks and datasets.
- Regularization prevents overfitting. It is a good practice to implement it especially, when there is class imbalance in train datasets.