

Assignment 1 Report

Tanat Tangun 630610737

August 2022

This report is about the result from using my implementation of the multi-layers perceptron on Rust language to solve 2 problems given on 261456 - INTRO COMP INTEL FOR CPE class. If you are interested to know how I implement the multi-layers perceptron and trained a model to solve these problems, you can see the source code for all models that have been shown in this report on my Github repository.

Flood Dataset

Problem

We want to predict water level at 7 hours ahead given station 1 and station 2 data at present, 1 hour before, 2 hours before, and 3 hours before.

| | A | B | C | D | E | F | G | H | I |
|----|-------|-------|-------|-------|-------|-------|-------|-------|-----|
| 1 | s1_t3 | s1_t2 | s1_t1 | s1_t0 | s2_t3 | s2_t2 | s2_t1 | s2_t0 | t7 |
| 2 | 95 | 95 | 95 | 95 | 148 | 149 | 150 | 150 | 153 |
| 3 | 95 | 95 | 95 | 95 | 149 | 150 | 150 | 150 | 153 |
| 4 | 95 | 95 | 95 | 95 | 150 | 150 | 150 | 150 | 153 |
| 5 | 95 | 95 | 95 | 95 | 150 | 150 | 150 | 150 | 153 |
| 6 | 95 | 95 | 95 | 95 | 150 | 150 | 150 | 152 | 153 |
| 7 | 95 | 95 | 95 | 95 | 150 | 150 | 152 | 152 | 153 |
| 8 | 95 | 95 | 95 | 96 | 150 | 152 | 152 | 153 | 153 |
| 9 | 95 | 95 | 96 | 97 | 152 | 152 | 153 | 153 | 153 |
| 10 | 95 | 96 | 97 | 98 | 152 | 153 | 153 | 153 | 153 |
| 11 | 96 | 97 | 98 | 100 | 153 | 153 | 153 | 153 | 154 |
| 12 | 97 | 98 | 100 | 100 | 153 | 153 | 153 | 153 | 155 |
| 13 | 98 | 100 | 100 | 100 | 153 | 153 | 153 | 153 | 156 |
| 14 | 100 | 100 | 100 | 101 | 153 | 153 | 153 | 153 | 156 |

Figure 1: Examples of given data where s1_t3 is water level from 3 hours before at station 1, and so on. t7 is water level at 7 hours ahead.

Parameters Setting

- All nodes use *sigmoid* as an activation function except output node that use *linear* function.
- Weights are random number that is in range $[-1, 1]$
- Each layer's bias is 1
- Use MSE (Mean Squared Error) as a loss function.

Training

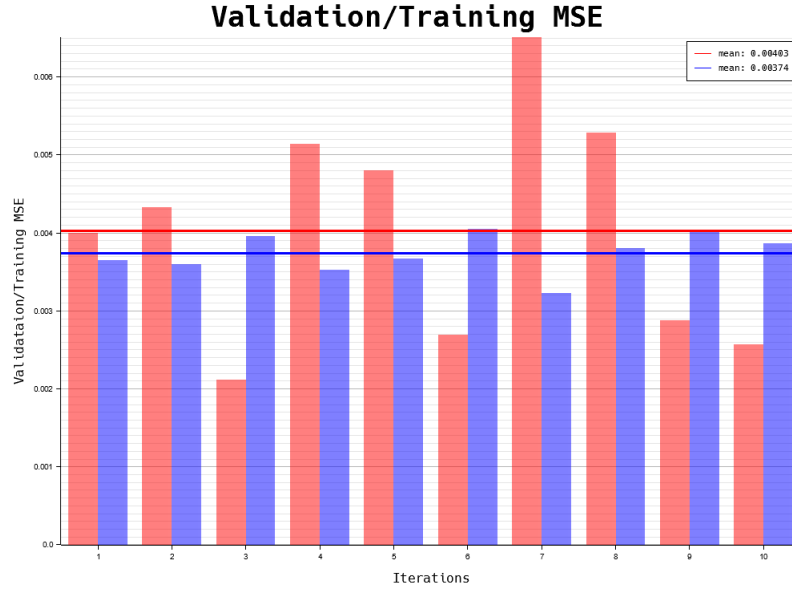
Use 10% cross-validation, and preprocess our data by using training set *mean* and *std* to standardize (For each data point x we calculate new $x' = \frac{x - \text{mean}}{\text{std}}$) both training set and validation set before training with SGD (Stochastic Gradient Descent) algorithm. Then, we train each cross-validation set for 1000 epochs.

We will create one base model that should perform good enough and create a variations base on that model. For this problem we will introduce these variations i.e. train with no *momentum*, train with smaller *learning rate*, train with no data preprocessing and add more layers or hidden nodes to see that if we introduce those variations, will the model perform better, converge faster, or have no improvement at all?

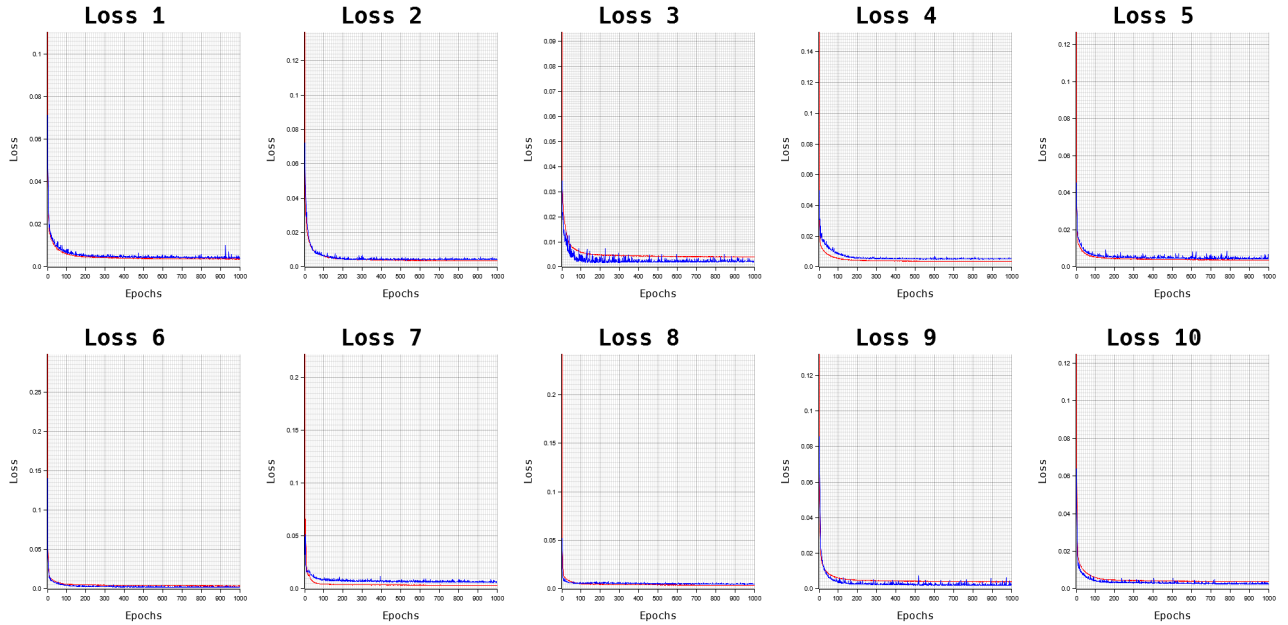
Training Result

Flood-8-4-1

Our base model that contains only 8 input nodes, 1 hidden layer with 4 nodes, and 1 output node train with $lr = 0.01$ and $momentum = 0.01$. The training process takes about 60 seconds. Below are the graphs we get from training this model.



(a) Each iteration training (blue) and validation (red) MSE at last epoch

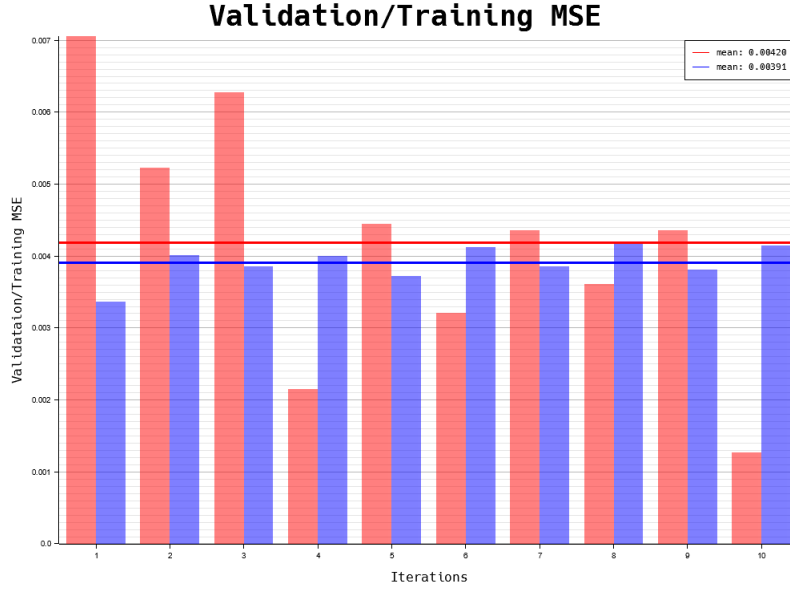


(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

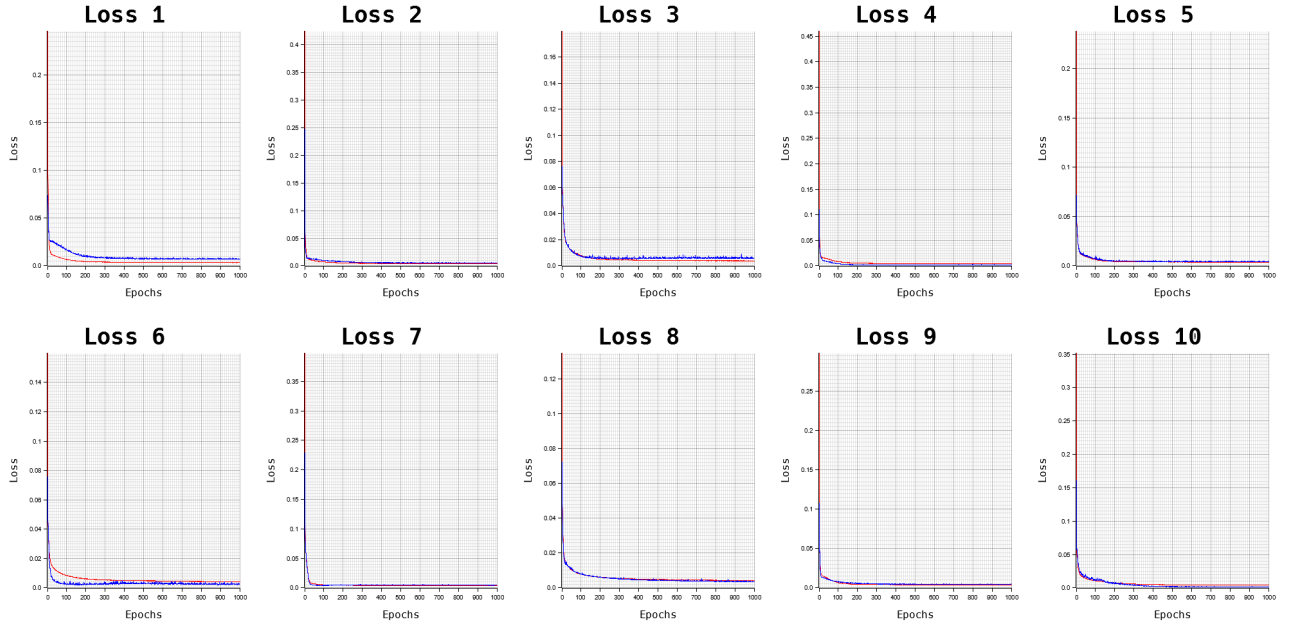
Figure 2: Training result of Flood-8-4-1.

Flood-8-4-1 with no momentum

Same base model but train with $lr = 0.01$ and $momentum = 0$. The training process takes about 60 seconds. Below are the graphs we get from training this model.



(a) Each iteration training (blue) and validation (red) MSE at last epoch

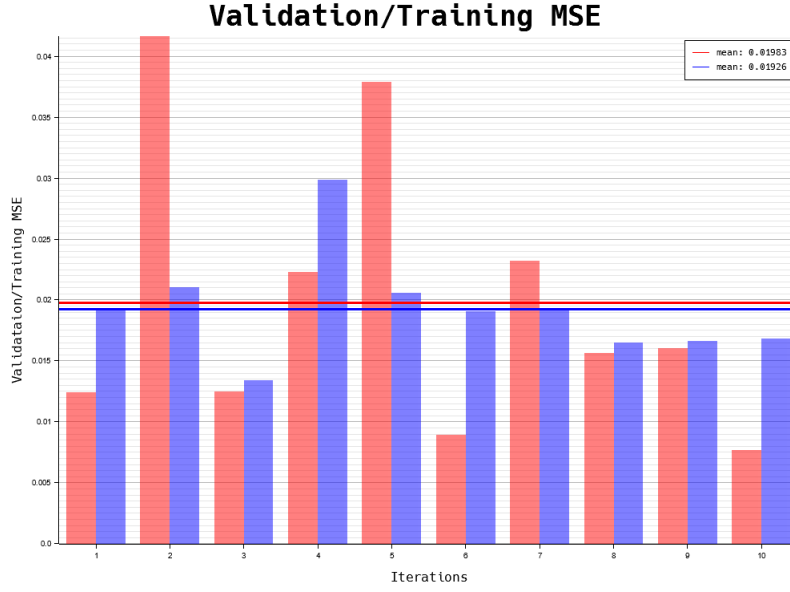


(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

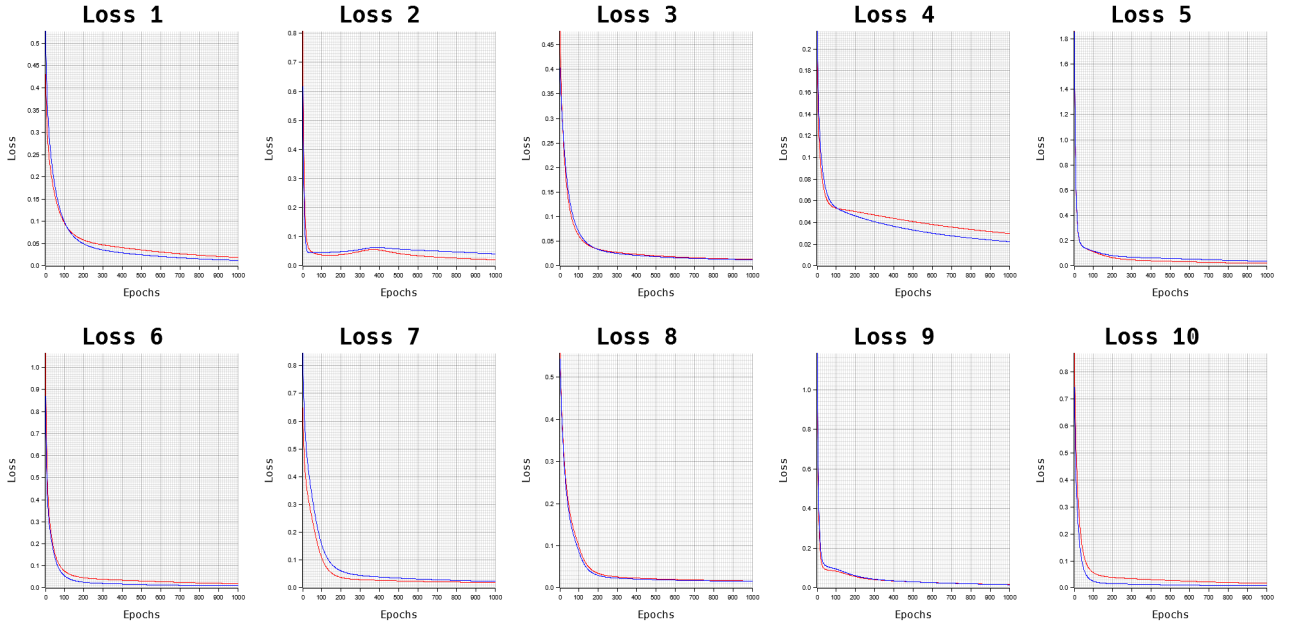
Figure 3: Training result of Flood-8-4-1 with no momentum.

Flood-8-4-1 with small learning rate

Same base model but train with $lr = 0.0001$ and $momentum = 0.01$. The training process takes about 59 seconds. Below are the graphs we get from training this model.



(a) Each iteration training (blue) and validation (red) MSE at last epoch

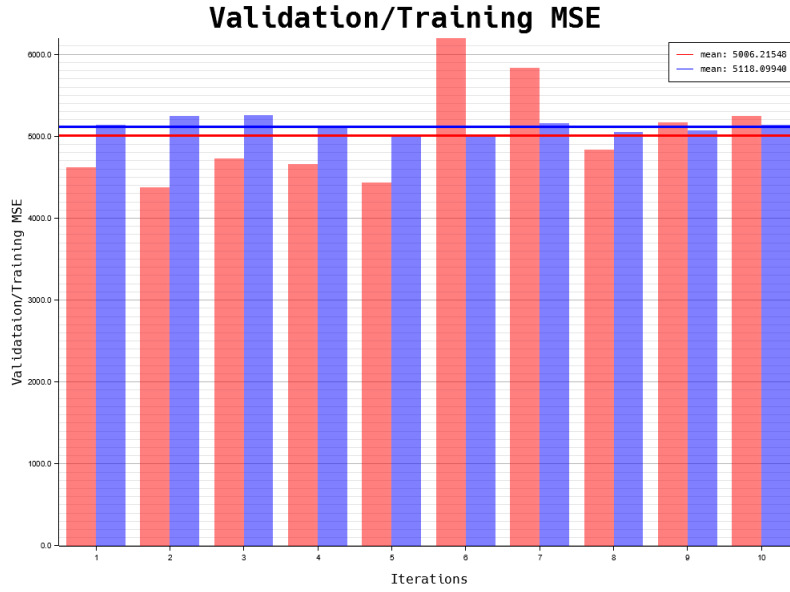


(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

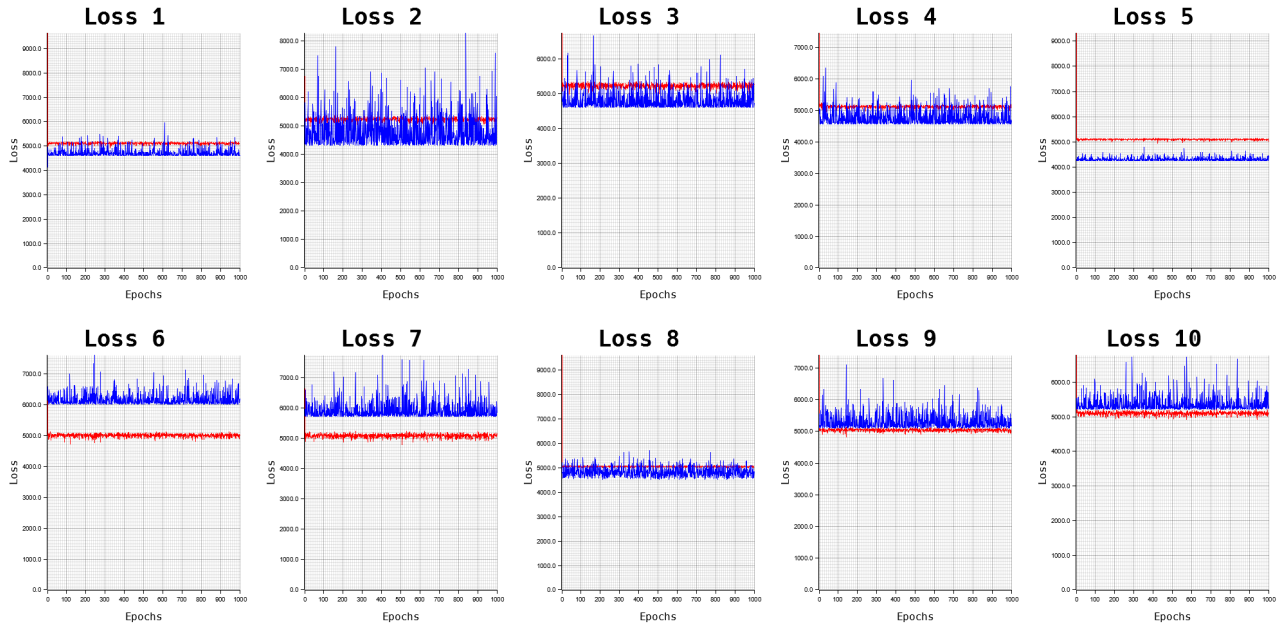
Figure 4: Training result of Flood-8-4-1 with smaller learning rate.

Flood-8-4-1 with no data preprocessing

Same base model train with $lr = 0.01$ and $momentum = 0.01$ but it is the only model with no data preprocessing. The training process takes about 57 seconds. Below are the graphs we get from training this model.



(a) Each iteration training (blue) and validation (red) MSE at last epoch

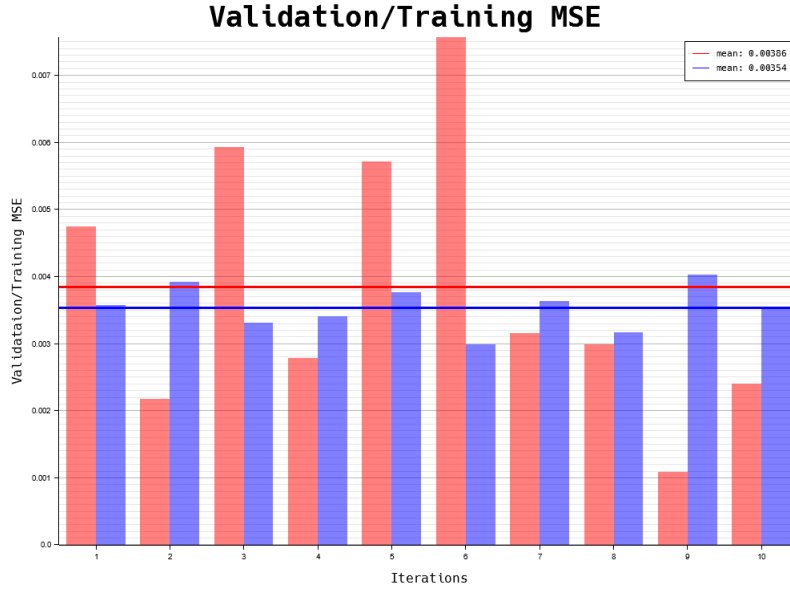


(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

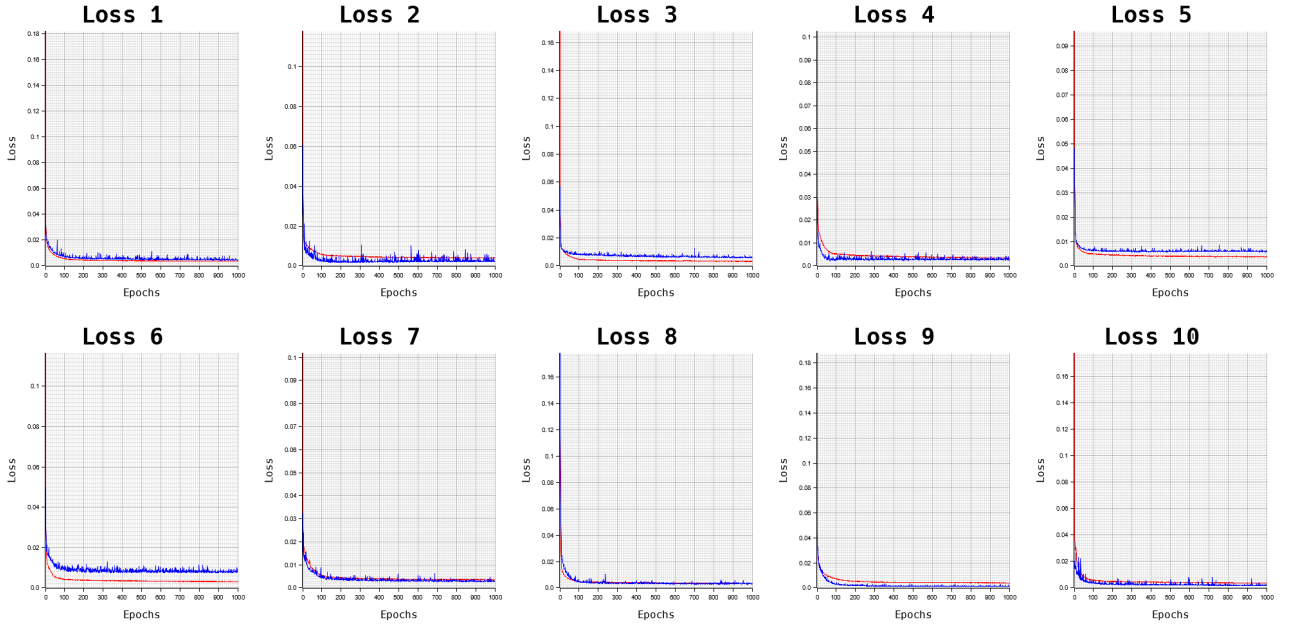
Figure 5: Training result of Flood-8-4-1 with no data preprocessing.

Flood-8-8-1

Bigger model that contains 8 input nodes, 1 hidden layer with 8 nodes, and 1 output node train with $lr = 0.01$ and $momentum = 0.01$. The training process takes about 105 seconds. Below are the graphs we get from training this model.



(a) Each iteration training (blue) and validation (red) MSE at last epoch



(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

Figure 6: Training result of Flood-8-8-1.

Analysis

From table 1, we can see that models with the same size will use around the same amount of training time, but with the bigger model, the training time gets longer. We can also see that Flood-8-4-1 with small learning rate is the worst in terms of validation set mean MSE compare with other models that do data preprocessing, the model seems to be stuck at a certain gradient level (see fig. 4b the model is slowly converging but too slow). Lastly, we can also see that the best performing model in terms of validation set mean MSE is Flood-8-8-1 at 3.86×10^{-3} which probably is because of 4 more hidden nodes that we have added.

| Model | Training Time (seconds) | Validation Set Mean MSE (10^{-3}) |
|--|-------------------------|---------------------------------------|
| Flood-8-4-1 | 60 | 4.03 |
| Flood-8-4-1 with no momentum | 60 | 4.20 |
| Flood-8-4-1 with small learning rate | 59 | 19.83 |
| Flood-8-4-1 with no data preprocessing | 57 | 5 006 215.00 |
| Flood-8-8-1 | 105 | 3.86 |

Table 1: Training time and validation set mean MSE (red line on fig. 2a, fig. 3a, fig. 4a, fig. 5a, and fig. 6a) of each Flood model.

We can also see a similar pattern on all of our Flood models in fig. 2b, fig. 3b fig. 4b, and fig. 6b the MSE is decreasing very fast at the first 100 epochs and seems to be stuck at a certain level of MSE or very slow in MSE reduction. But, there is one model that does not follow this pattern that is Flood-8-4-1 with no data preprocessing as we can see in fig. 5b, the training MSE seems to be flickering a lot and there is no sign of the training MSE converging which results in the poor performance of the model that can not be compared with model train with data preprocessing even when we unstandardized the MSE of the model with data preprocessing e.g. we unstandardized Flood-8-4-1 with small learning rate using this equation $x = x' \times std + mean$ where $mean \approx 340$ and $std \approx 120$ come from whole dataset and $x' = 19.83 \times 10^{-3}$, we get $x = 342.3796$ which is much smaller than 5006.215.

CrossPat Dataset

Problem

We want to predict the class (1 of possible 2 classes) that belongs to our inputs (or features).

```
1 p0
2 0.0902 0.2690
3 1 0
4 p1
5 0.8143 0.5887
6 0 1
7 p2
8 0.2962 0.0697
9 1 0
10 p3
11 0.5533 0.5493
12 0 1
13 p4
14 0.1472 0.1856
15 1 0
16 p5
17 0.4653 0.6214
18 0 1
```

Figure 7: Examples of given data where p_x is an object and the first line after it is its features, the second line is its class

Parameters Setting

- All nodes use *sigmoid* as an activation function.
- Weights are random number that is in range $[-1, 1]$
- Each layer's bias is 1
- Use MSE (Mean Squared Error) as a loss function.

Training

Use 10% cross-validation with no data preprocessing, and train with SGD (Stochastic Gradient Descent) algorithm. Then, we train each cross-validation set for 5000 epochs.

We will create one base model that should perform good enough and create a variations base on that model, that is train with no *momentum*, train with smaller *learning rate*, and add more layers or hidden nodes to see that if we introduce those variations, will the model perform better, converge faster, or have no improvement?

Training Result

We use only 1 output node in all models because this is a binary classification task so we can just map a pair $(1, 0) \rightarrow 1$ and $(0, 1) \rightarrow 0$. We then have a threshold at 0.5 if output is more than 0.5 then it is 1 else it is 0. Accuracy is then calculated by using this equation $\frac{TP+TN}{TP+TN+FN+FP}$ where TP, TN, FN, FP come from confusion matrix.

Cross-2-4-1

Our base model with 2 input nodes, 1 hidden layer with 4 nodes, and 1 output node train with $lr = 0.01$ and *momentum* = 0.01. The training process takes about 122 seconds..

Cross-2-4-1 with no momentum

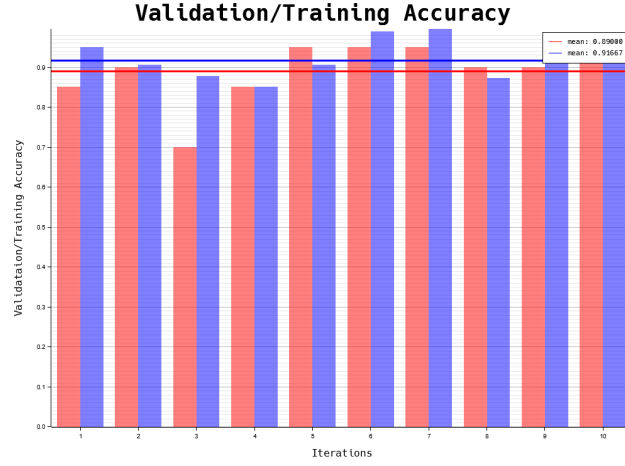
Same base model train with $lr = 0.01$ and *momentum* = 0.0. The training process takes about 122 seconds.

Cross-2-4-1 with smaller learning rate

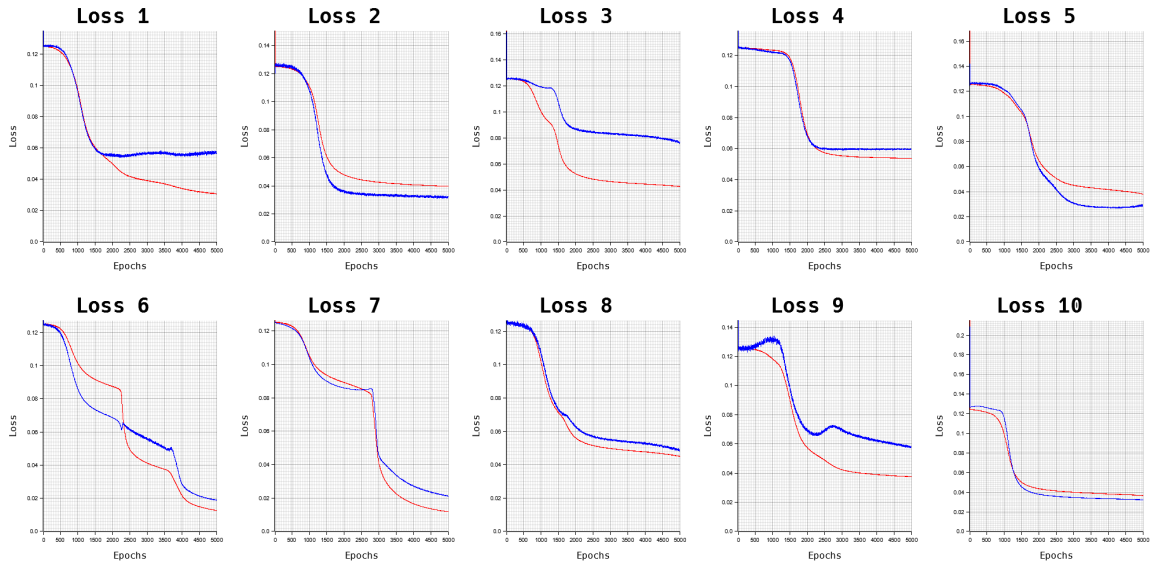
Same base model train with $lr = 0.0001$ and *momentum* = 0.01. The training process takes about 124 seconds.

Cross-2-8-1

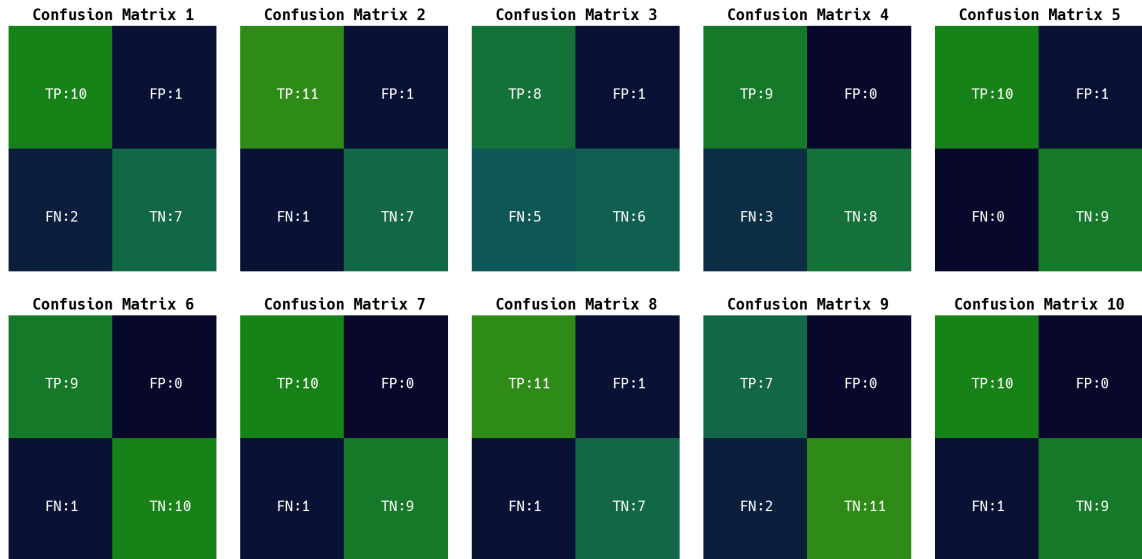
Bigger model that contains 2 input nodes, 1 hidden layer with 4 nodes, and 1 output node train with $lr = 0.01$ and *momentum* = 0.01. The training process takes about 230 seconds.



(a) Each iteration training (blue) and validation (red) set accuracy at last epoch

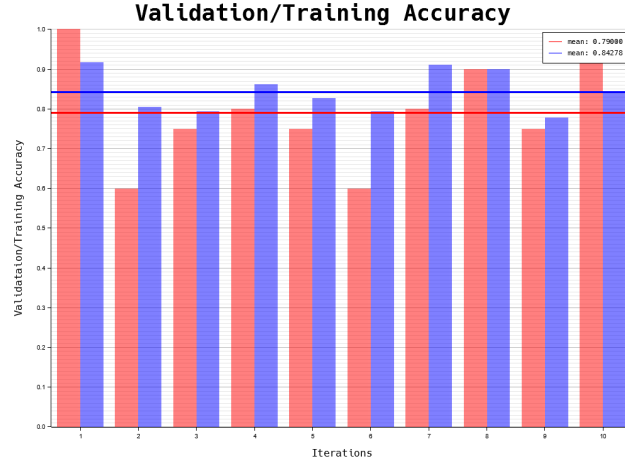


(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

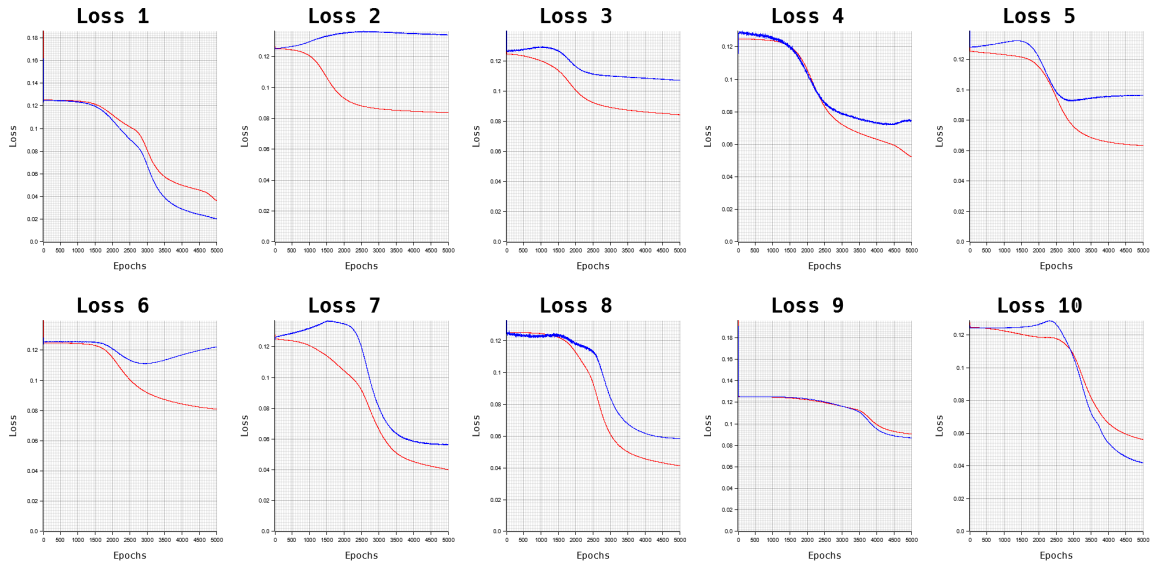


(c) Each iterations validation set confusion matrix where y-axis is an actual class 1,0 top to bottom and x-axis is predicted class 1,0 left to right.

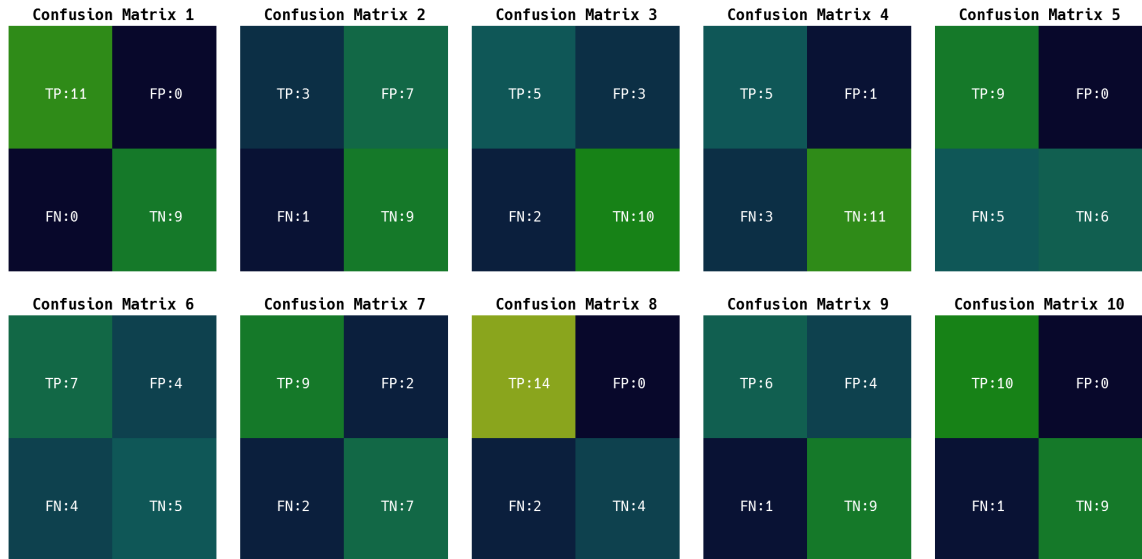
Figure 8: Training result of Cross-2-4-1.



(a) Each iteration training (blue) and validation (red) set accuracy at last epoch

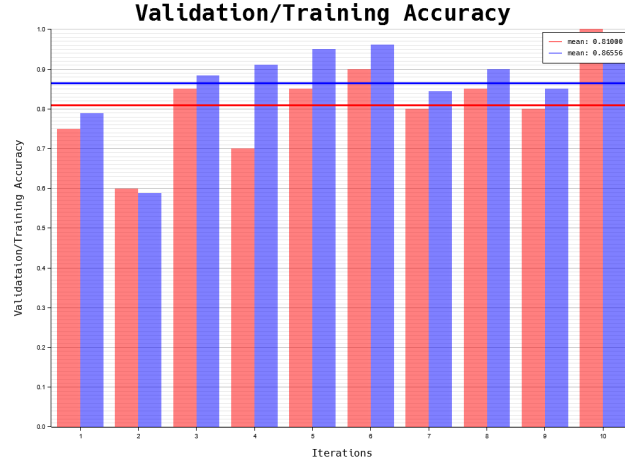


(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

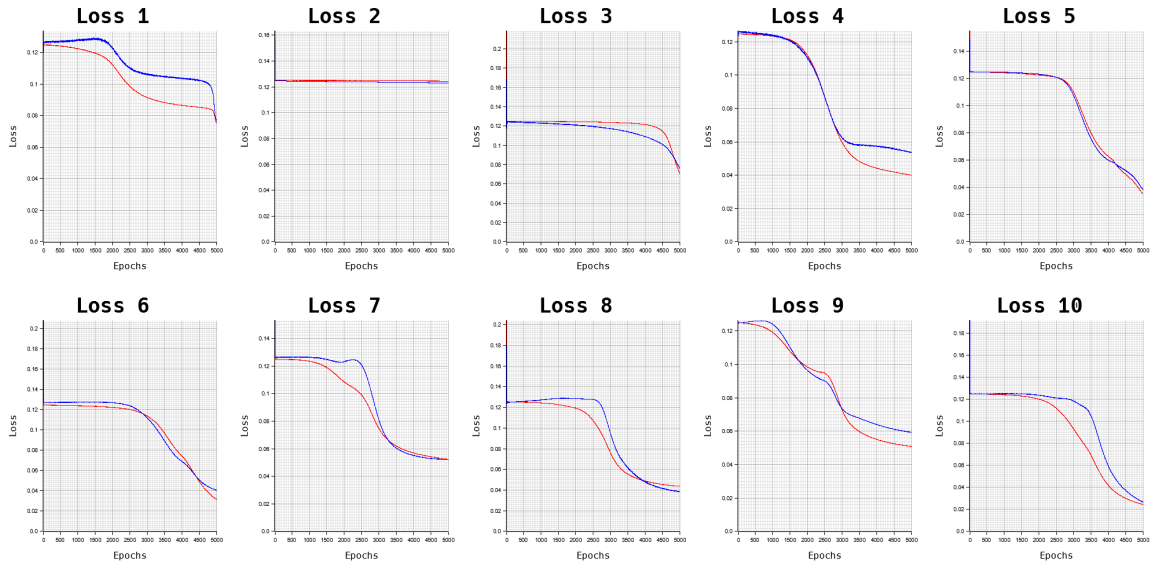


(c) Each iterations validation set confusion matrix where y-axis is an actual class 1,0 top to bottom and x-axis is predicted class 1,0 left to right.

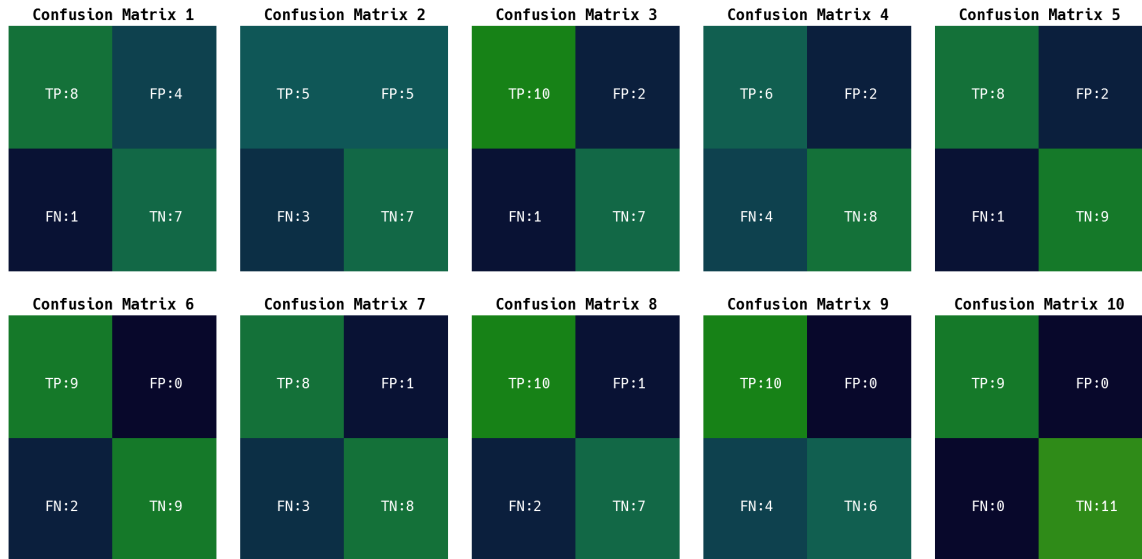
Figure 9: Training result of Cross-2-4-1 with no momentum.



(a) Each iteration training (blue) and validation (red) set accuracy at last epoch

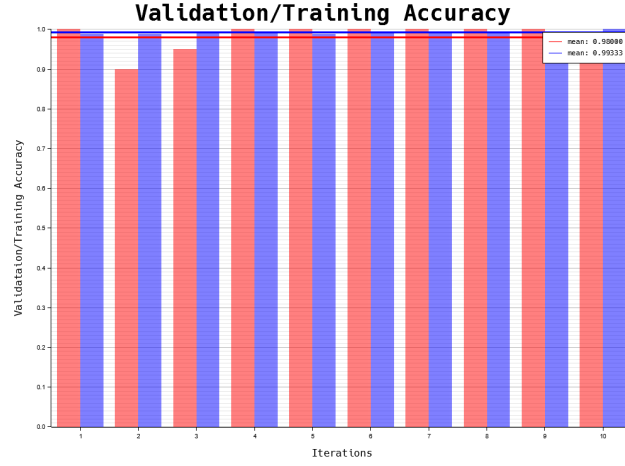


(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.

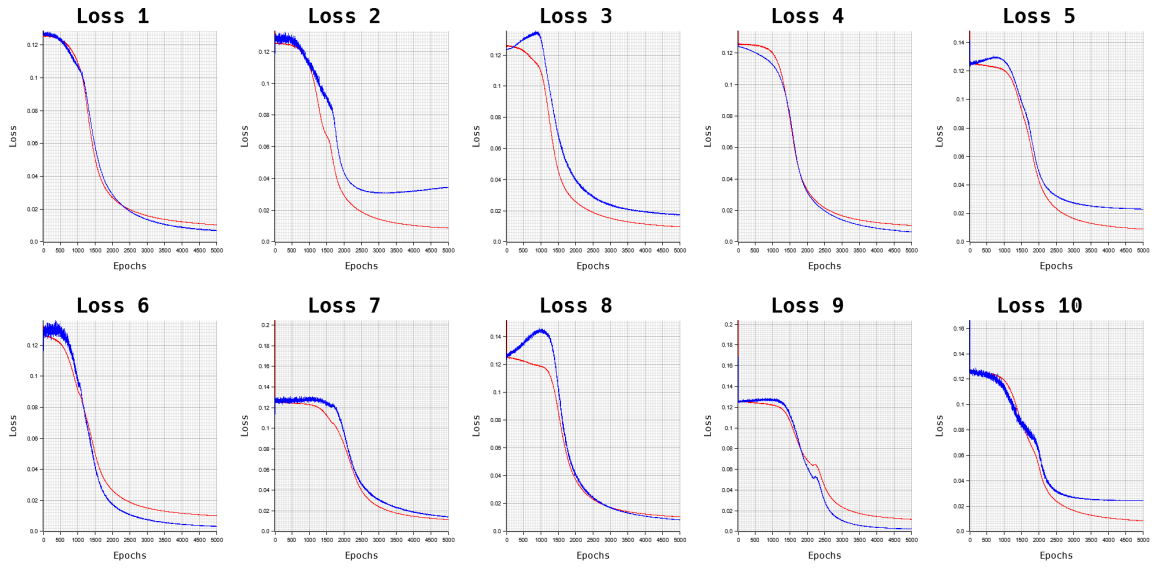


(c) Each iterations validation set confusion matrix where y-axis is an actual class 1,0 top to bottom and x-axis is predicted class 1,0 left to right.

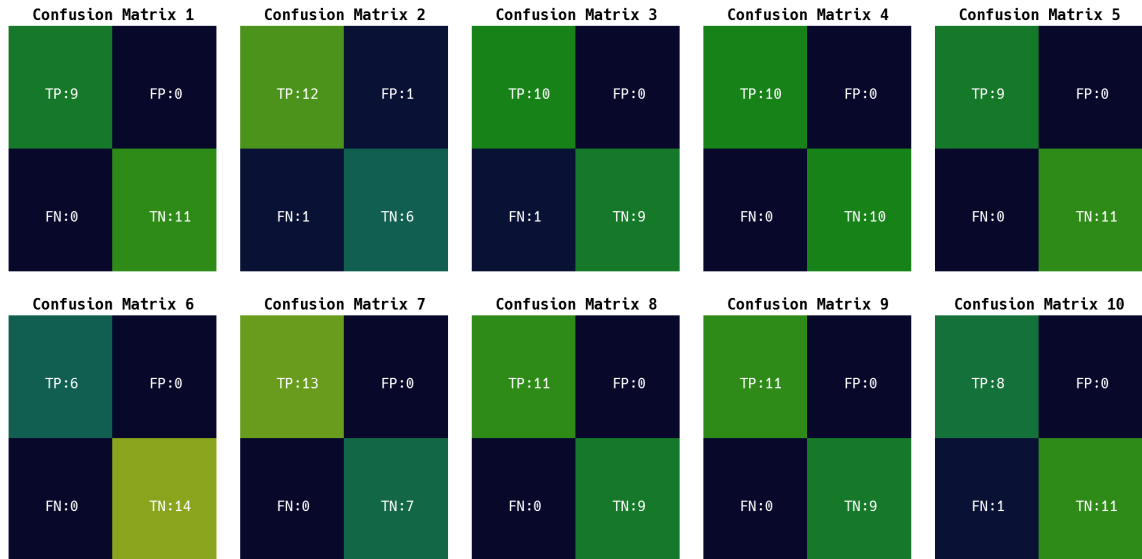
Figure 10: Training result of Cross-2-4-1 with smaller learning rate.



(a) Each iteration training (blue) and validation (red) set accuracy at last epoch



(b) Each iteration training MSE (blue) and validation MSE (red) at each epoch.



(c) Each iterations validation set confusion matrix where y-axis is an actual class 1,0 top to bottom and x-axis is predicted class 1,0 left to right.

Figure 11: Training result of Cross-2-8-1.

Analysis

From table 2, we can see that all Cross-2-4-1 models used around the same amount of training time but only the base model is performing okay in terms of validation set mean accuracy. And the biggest model Cross-2-8-1 is performing much better than others at 98% validation set mean accuracy but the training time used is also around 2 times other models training time.

| Model | Training Time (seconds) | Validation Set Mean Accuracy (%) |
|--------------------------------------|-------------------------|----------------------------------|
| Cross-2-4-1 | 122 | 89 |
| Cross-2-4-1 with no momentum | 122 | 79 |
| Cross-2-4-1 with small learning rate | 124 | 81 |
| Cross-2-8-1 | 230 | 98 |

Table 2: Training time and validation set mean accuracy (red line on fig. 8a, fig. 9a, fig. 10a, and fig. 11) of each Cross model.

From fig. 11b, we can also see that Cross-2-8-1 training is going smoother with a steady decrease in MSE compared with other models as we can see in fig. 8b, fig. 9b, and fig. 10b the graphs overall seem to go down and stop at a certain level indicating that the model stop learning. Confusion matrix of Cross-2-8-1 from fig. 11c also, show that the error is not biased to one specific class indicating that the model learned about the 2 classes equally. On the other hand, the confusion matrix of all Cross-2-4-1 models specifically in fig. 9c and fig. 10c we can see that there're a big number of FP and FN and in some iteration the error is biased to one of FP or FN.

Summary

From both problems, we have learned that an MLP model can solve these problems and the performance of the model is dependent on parameters that we can set for the model such as learning rate, momentum, and size of the model as we can see in the above experimenting result. And, in some datasets, there is a need for data preprocessing so that the model can fit to that dataset as we can see in Flood Dataset.