Assignment: Checkerboard classification problem

David Bertoldi – 735213
email: d.bertoldi@campus.unimib.it

Department of Informatics, Systems and Communication

University of Milano-Bicocca

1 Inspecting the data

The data provided consists of an set of points $X \in \{(x_1, x_2)|10 \leqslant x_1, x_2 \leqslant 20\}$ and a set of binary labels $y \in \{0, 1\}$. The points in X follow a uniform distribution with mean $\mu = 14.97$ and variance $\sigma = 8.3$; the 0s and 1s in y are quite uniformely distributed with a mean of $\mu = 0.498$.

If we map each point $i \in X$ with the i^{th} element of y and assign for each element of y a different color we get a *check pattern* over the points of X, partitioning the 4000 points in 6 columns and 6 rows. The result can be observed in Figure 1

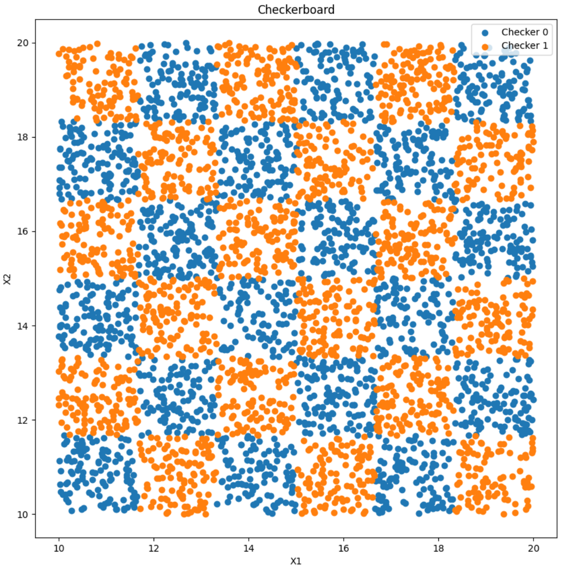


Figure 1: Visualization of the dataset

2 Preparing the data

Before using the data we want to standardize features by removing the mean and scaling to unit variance. Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features are not standard normally distributed. That's why we want to remove the mean μ and scale σ to unit variance.

In order to do so, we use the `StandardScaler` from `sklearn.preprocessing` package. After this operation we have $\mu = 1.4 \cdot 10^{-9}$ (very close to 0) and $\sigma = 1$, which is exactly what we wanted.

3 Building the network and training

3.1 Data split

The dataset does not provide a set of points to be used for validation and testing. So we split the dataset into three partitions: 72% of points are used for the training, 18% for validation and 10% for testing. Having less points used for training had proved to perform worse.

3.2 Network

The network chosen is a Feed-Forward Neural Network with non-linear activation functions. We tried two approaches: `tanh` and `LeakyReLU`. The first one seemed to be right choice because it is possible to emulate the check pattern with the following formula:

$$sin(y) \leqslant cos(x)$$

and since the definition of $tanh(x)$ can be derived from $cos(ix)$ and $sin(ix)$, there could had been some facilitation on using such activator. Unfortunately, despite the good results, `LeakyReLU` performed better and for this reason we used it as a definitive one.

Because we are trying to classify points between two classes, the output function is a sigmoid.

Finally, the network architecture is composed by 4 hidden dense layers of rispectively 200, 150, 100 and 50 nodes. We tried with less hidden layers (2 and 3) and less nodes for each layer (100, 50, 25, 10) but the performance where worse in terms of convergence speed and decision boundaries. Since the different configurations have similar training times, we chose this configuration over a lighter one.

Each node uses `LeakyReLU` as activator function with $\alpha = 0.1$. The output node computes the sigmoid logistic function so that the output ranges between 0 and 1.

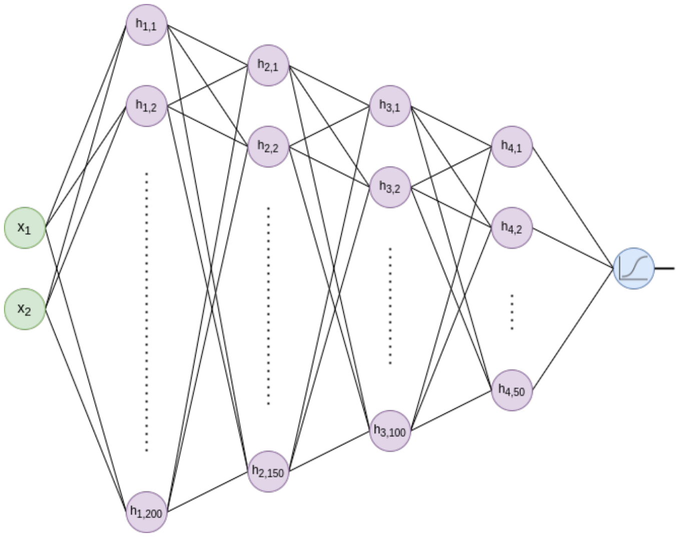
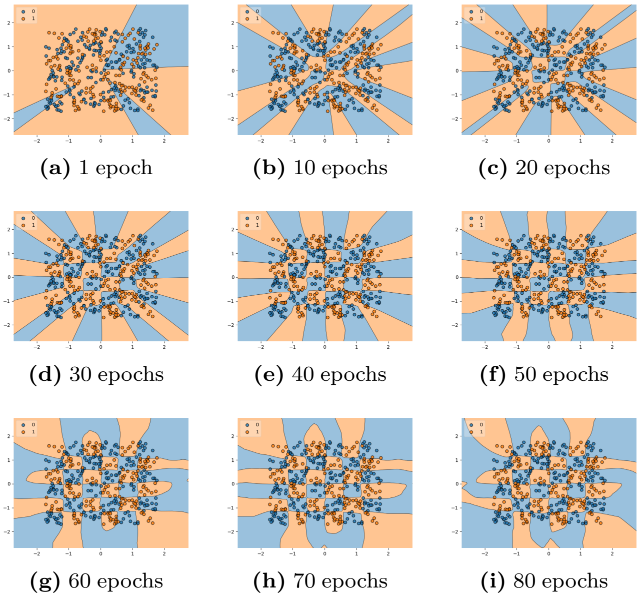


Figure 2: The neural network with 4 dense hidden layers

3.3 Training

The choice of the optimizer was among *SGD*, *RMSProp* and *Adam*. We tried all of them and chose *Adam* with learning rate of 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ because it seemed to escape better from local minima, converging faster and giving better accuracy.

Because we are treating a binary classification problem, we used the *binary crossentropy* loss function. Figure 3 shows how the decision boundaries evolve over the test dataset every 10 epochs.



(a) 1 epoch **(b)** 10 epochs **(c)** 20 epochs

(d) 30 epochs **(e)** 40 epochs **(f)** 50 epochs

(g) 60 epochs **(h)** 70 epochs **(i)** 80 epochs

Figure 3: Evolution of decision boundaries during the training

measure Figure 4 and Figure 5 plots the trend of the loss function and its accuracy over the validation set. The first one measures the absolute difference between the model's prediction and the actual value.

The second one represents the ratio between the number of correct predictions to the total number of predictions; it is preferable over the precision because accuracy treats all classes with the same importance while precision measures the model's accuracy only in classifying a sample as positive.

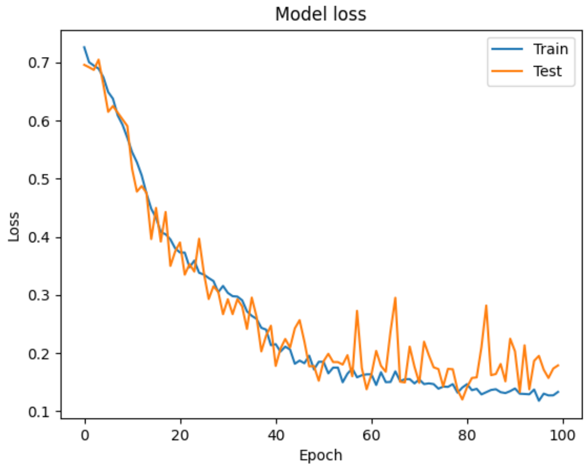


Figure 4: Loss

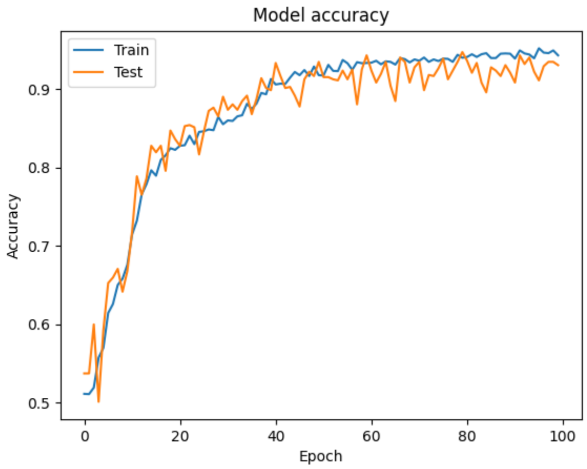


Figure 5: Accuracy

4 Analysis

From Figures 3, 4 and 5 we see that the model correctly classifies almost all the points after 60 epochs (accuracy $\geqslant$ 90%). This is a good result but we expected that the model could figure out what is the general rule under the provided classification. As we can see from Figure 3, as the accuracy of the model increases, it is clear that the model fails to recognize the *check pattern* itself. Even if it is true that the accuracy reaches a score of 93%, the model overfits the data and could not generalize the pattern beyond the region occupied by training data. This means that a point outside the original training set (*e.g.* $(9, 19)$) is classified by the model in the same way of the nearest checker (*e.g.* orange instead of blue, even if the checker on its left is orange).

What happens is that the pattern predicted by the model is a checkerboard with the external checkers stretched infinitely outwards. Figure 6 shows a simplified trend in model's prediction if we increase the number of epochs.

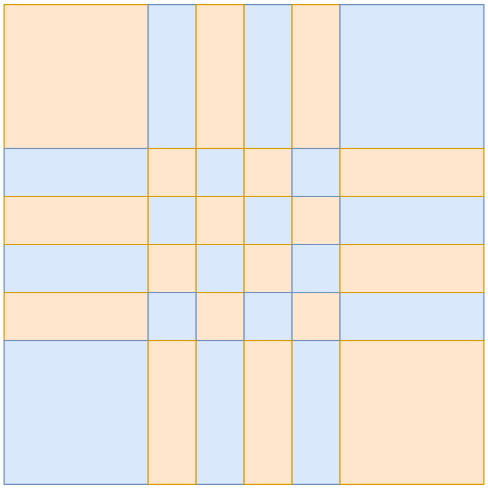


Figure 6: Trend of the decision boundaries of the model

5 Alternative problem

The following problem is a variation of the previously presentend one. What is changing is the number of samples (from 4000 to 1000) and the number of checkers (from 36 to 400). This means that the frequency of samples is much more rare while leaving almost unchanged means and variation ($\mu = 15.09$, $\sigma = 8.64$).

Figure 7 shows the new problem and it is impossible for the human brain to recognize the *checker pattern*. The checkers are too thin and the data is not enough in order to recognize any pattern.

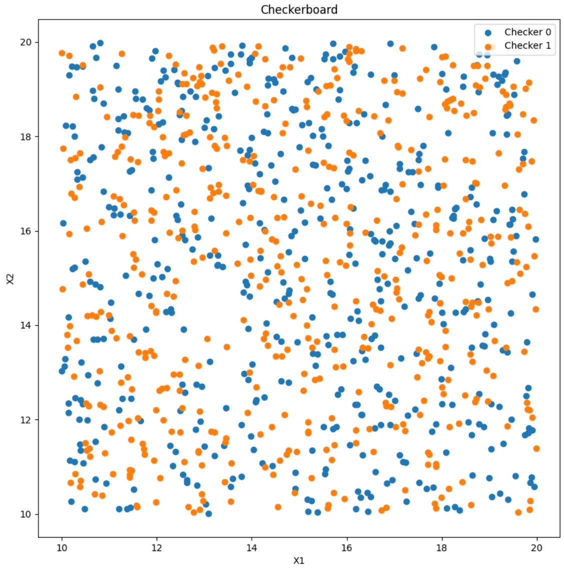
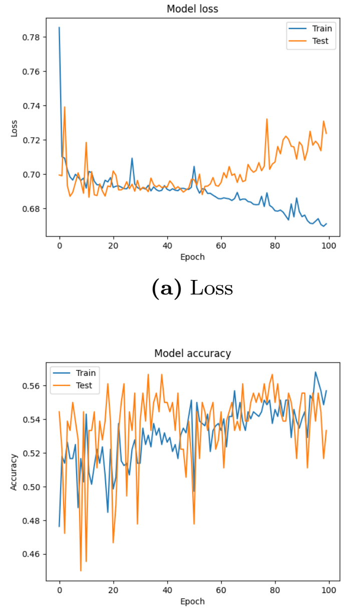


Figure 7: Visualization of the dataset of the alternative problem

The model, without any surprise, performs much worse: the maximum accuracy is $\sim 55\%$, the loss remains quite high and it never converges (Figure 8)

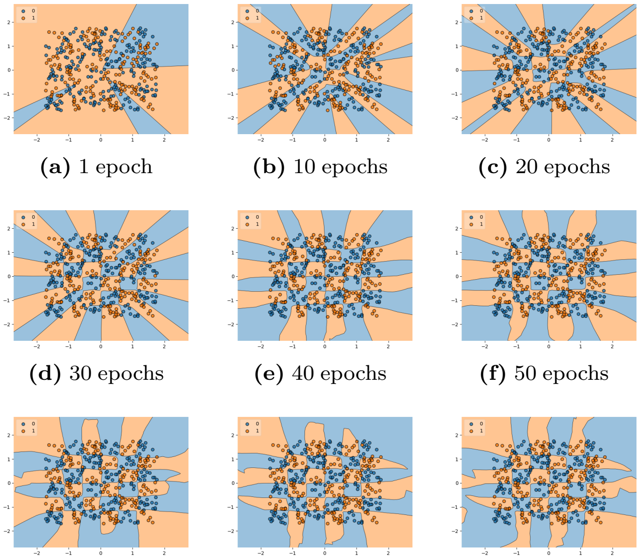


(a) Loss

(b) Accuracy

Figure 8: Loss and accuracy plots of the alternative problem

Figure 9 shows how the model could not find any pattern, changing quite frequently its decision boundaries



(a) 1 epoch **(b)** 10 epochs **(c)** 20 epochs

(d) 30 epochs **(e)** 40 epochs **(f)** 50 epochs

(g) 60 epochs **(h)** 70 epochs **(i)** 80 epochs

Figure 9: Evolution of decision boundaries during the training of the alternative problem

6 Other experiments

In this section we briefly documented the experiments with other optimizers: *SGD* and *RMSprop*. Even if their results where good, they converged slower than *Adam*.

RMSprop converges better than *SGD* but it has difficulties on escaping local minima, having the loss function over test data to be more swinging (Figure 10).

SGD converges slower than *RMSprop* but seems to be slighty better in escaping local minima (Figure 12).

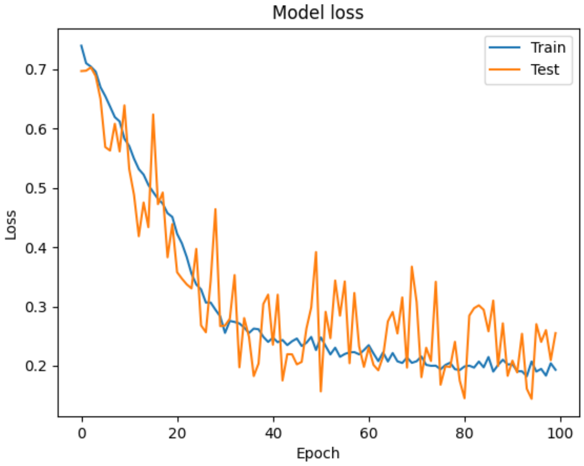


Figure 10: Loss of *RMSprop*

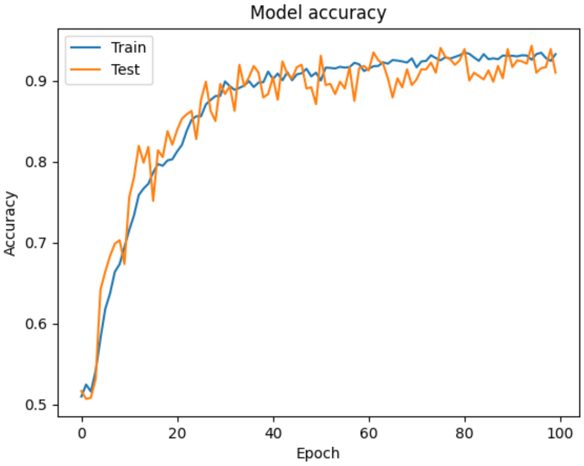


Figure 11: Accuracy of *RMSprop*

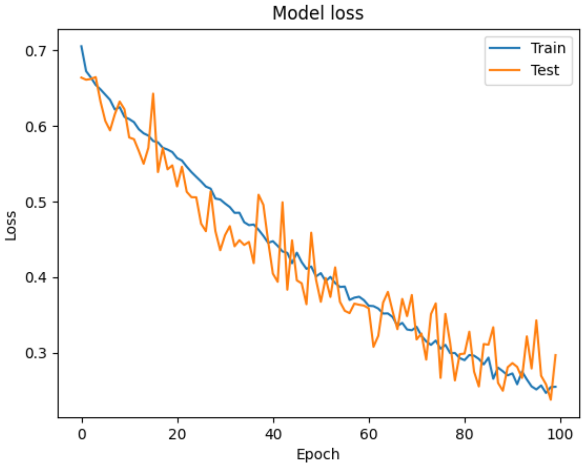


Figure 12: Loss of *SGD*

References

[1] NVIDIA Developer: Turing tuning guide, `https://docs.nvidia.com/cuda/turing-tuning-guide/index.html`

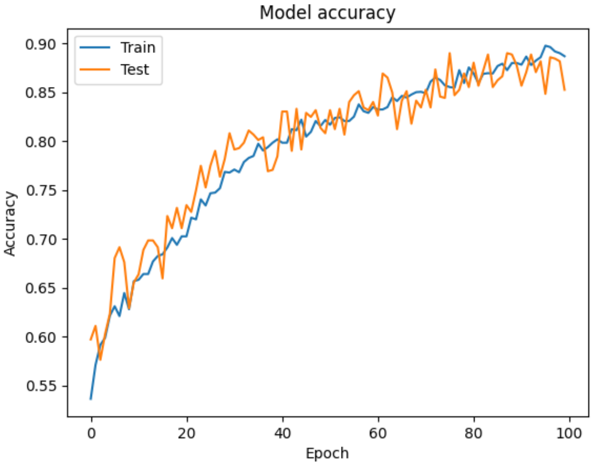


Figure 13: Accuracy of *SGD*

[2] Fletcher, J. G. (January 1982). *An Arithmetic Checksum for Serial Transmissions*. IEEE Transactions on Communications. COM-30 (1): 247–252. doi:10.1109/tcom.1982.1095369