Annalisa Paladino
SM3800021

Deep Learning – Assignment 1

Introduction

The aim of this project is to reproduce in `PyTorch` the results of the experiment outlined in Fig. 1 of the paper "*Learning representations by back-propagating errors*" (Rumelhart et al., 1986) [1]. The proposed experiment illustrates a neural network which can detect mirror symmetry in binary sequence. Mirror symmetry, in this context, refers to sequences that read the same forward and backward. The ability to recognize such patterns is crucial in various applications, including data compression, error detection, and cryptography.

Implementation

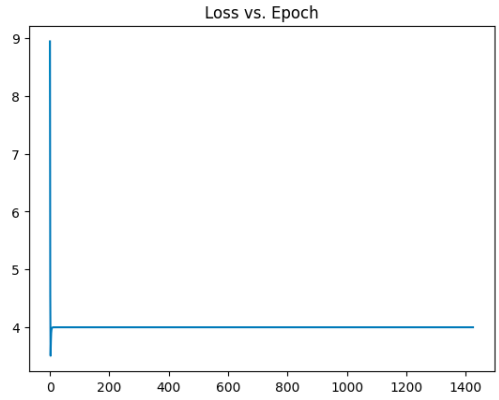
To propose an implementation that is as close as possible to the experiment shown in the paper we can start by creating the **dataset**. The dataset consists of all possible 6-bit binary numbers, amounting to 64 unique patterns. Each pattern is labeled as 1 if it exhibits mirror symmetry and 0 otherwise. The symmetry check is performed using the function `is_symmetric`, which compares a binary string with its reverse. The binary sequences and their corresponding labels are converted into tensors, which are then packaged into a `TensorDataset`. This dataset feeds into a `DataLoader` that allows batch processing of the data during model training. In this project, the entire dataset is processed in one batch as it comprises only 64 examples.

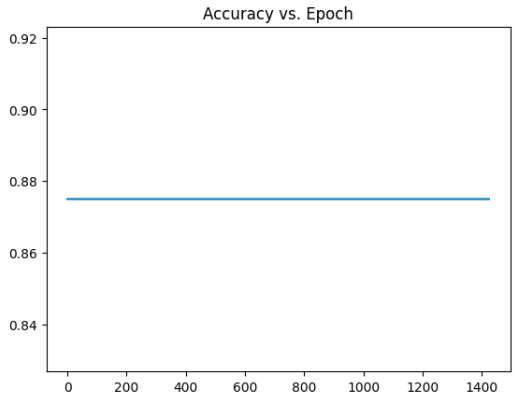
The **neural network** `MirrorSymmetryNetwork` proposed by the paper has a minimal architecture yet it's sufficient for the task due to the limited complexity of the data. In fact, it has a *hidden layer* that takes the input data, a 6-bit binary sequence, and output a tensor of size 2, it is equipped with *sigmoid* activation functions to introduce non-linearity and an *output layer* that takes the output of the hidden layer and outputs a tensor of size 1, also using a *sigmoid* activation function to generate a binary output (0 or 1).
The model **weights** and **biases** are initialized uniformly within the range [-0.3, 0.3], this initialization is intended to promote a balanced learning process.

We can define the **loss** criterion by using the one described in the paper, it calculates the squared difference between each predicted value and its corresponding true value, then it's divided by 2 to match the formula of half-mean-squared-error (MSE) for a more standardized loss. The **optimizer** is SGD with learning rate 0.1 and momentum 0.9 as requested by the paper.

[1] https://bucket.ballarin.cc/papers/oth/rumelhart1986.pdf

Training is conducted over 1425 epochs, it involves computing of the loss and backpropagating the errors to update the weights and biases. Accuracy here is defined as the proportion of correctly identified symmetrical sequences.
It's important to observe that in the paper the authors used all 64 possible 6D binary arrays as the training set for the neural network, and in this dataset, 85.7% of the arrays are non-symmetric. From the plot we can see that we have an accuracy of 0.875, it's equal to the proportion of the 0 labels present in the dataset, this means that the neural network is classifying every array as non-symmetric. The loss plot shows that at the initial epochs (close to 0), the loss starts at a high value near 9, indicating poor initial performance, quickly after the initial epoch, the loss value drops drastically and then remains steady around 4. This pattern could indicate that the model quickly reaches a local minimum or a plateau early in training, those results are not satisfying. A possible cause of this poor performance could be the imbalance in the dataset and between the two classes. Those results are really different from the ones showed in the paper.

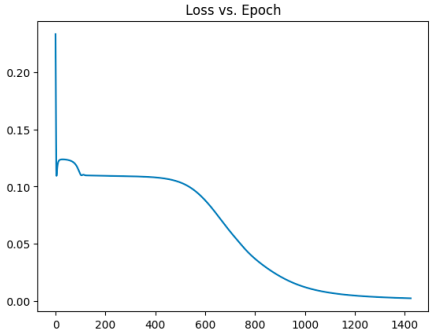


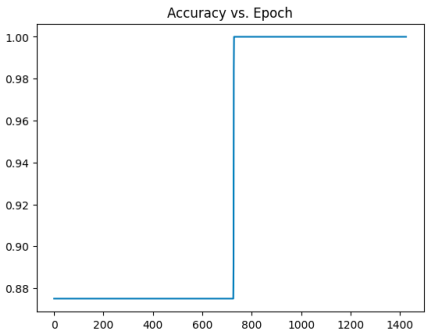


Exploration

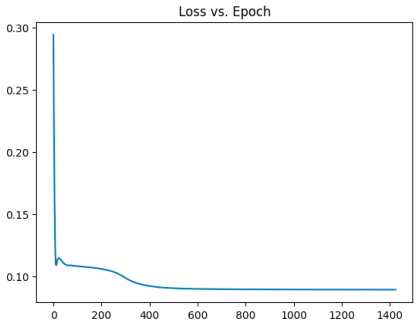
In this section we will briefly discuss some possible variations of the main implementation, for further clarification you can check the file *PALADINO_optionalHW1.ipynb*. Some improvements that I tried are:

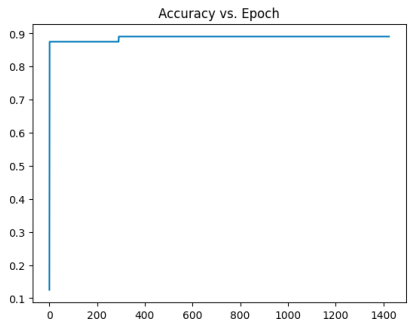
- Change the loss criterion: instead of the loss proposed in the paper I used MSE, now the loss has lower values, the accuracy is more or less the same.
- Change the optimizer: it's possible to change the learning rate by setting it to 1, the results are quite similar to the original implementation.
- Change both the loss and the optimizer: by setting the learning rate to 1 and using MSE as our loss criterion it's possible to observe some greater improvement in our solution, this is the best alternative implementation found so far.





- Change the optimizer: it's possible to use a different optimizer, in this case an attempt was done with *Adam*, this approach has similar results of the first point.





- Change the activation function: it's possible to use *ReLU* activation function for the hidden layer and *sigmoid* activation function for the output layer, the results are similar to the ones obtained in the first point.