Intro to Machine Learning  
Report for Code Assignment 1

# 

# 

|  |  |
| --- | --- |
| Tal Grossman | 201512282 |
| Din Carmon | 209325026 |
| Amir Sharif Jamal | 213850811 |

# 

# how-to-run and implementation details

## How to run

* Download all Python files to a single directory. (for example “code\_assignment\_1”)
* Run the “main.py” (for example python main.py)
  + This will run both parts and save the results
* If you wish to change hyperparameters look for Case constants in each part.

## Implementation details:

### Dataset handling:

* Dataset: tabular breast\_cancer classification from: <https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_cancer>
* Implemented in “main.py” under the “get\_dataset” function.
* shuffled randomly with a hyperparameter random seed.
* Used 80% for training and 20% for testing.

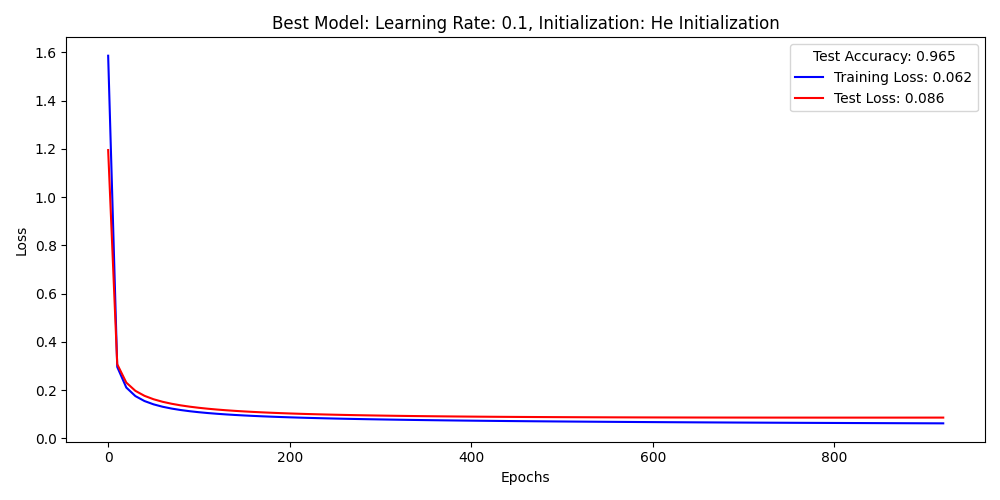
### Part 1:

* Implemented in part1\_neural\_network.py as well as required utility functions.
* Runs the training and returns all results including train and test loss over epochs, and final test accuracy per experiment (see details in part 1 section).
* Plot and save all experiments results including best final weights.

# 

# Part 1: Single-Layer Neural Network with Gradient Descent

This section will include the implementation details and results of a single-layer neural network.

Figure 1: The best experiment training and testing losses plot with a test accuracy of 96.5%. Experiment configuration: learning rate of 0.1 and He Initialization.

## Methodology:

### Training:

* **Model**:

A single-layer neural network with sigmoid activation and cross-entropy loss functions.

* **weights initialization**: we used both He initialization and Normal Initialization.
* About the He initialization:  
  He initialization is a weight initialization strategy designed to improve training stability and prevent vanishing or exploding gradients, especially in deep networks. It carefully scales the initial weights based on the number of input connections to each neuron, ensuring that the activations neither become too small (hampering learning) nor too large (causing saturation). This leads to faster and more reliable convergence during training.
  + Formula: W ~ N(0, σ²) where σ² = 2 / n

Where

W represents the weight matrix.

N(0, σ²) denotes a Gaussian distribution with mean 0 and variance σ².

n is the number of features in our case.

* + References:
    - <https://arxiv.org/abs/1502.01852>
    - <https://medium.com/@shauryagoel/kaiming-he-initialization-a8d9ed0b5899>
* **Stopping Conditions:**
  + Hyperparameter of maximum iterations (epochs) set to 1000.
    - This was the actual stopping condition as in all experiments we have seen that both the training and testing losses were converged and the model achieved good accuracy.
  + Model Improvement “LEARNING\_PATIENCE”.
    - This stooping condition should stop the training process if the validation loss (the test loss in our case) did not improve for a maximum configurable iteration.
    - Practically it was not used as the testing loss kept improving all through the iteration, even if just slightly.

### Experiments

|  |  |
| --- | --- |
| Learning rates | Initializations |
| 1. 0.1 2. 0.01 3. 0.001 | 1. He initialization, References: 2. normal initialization |

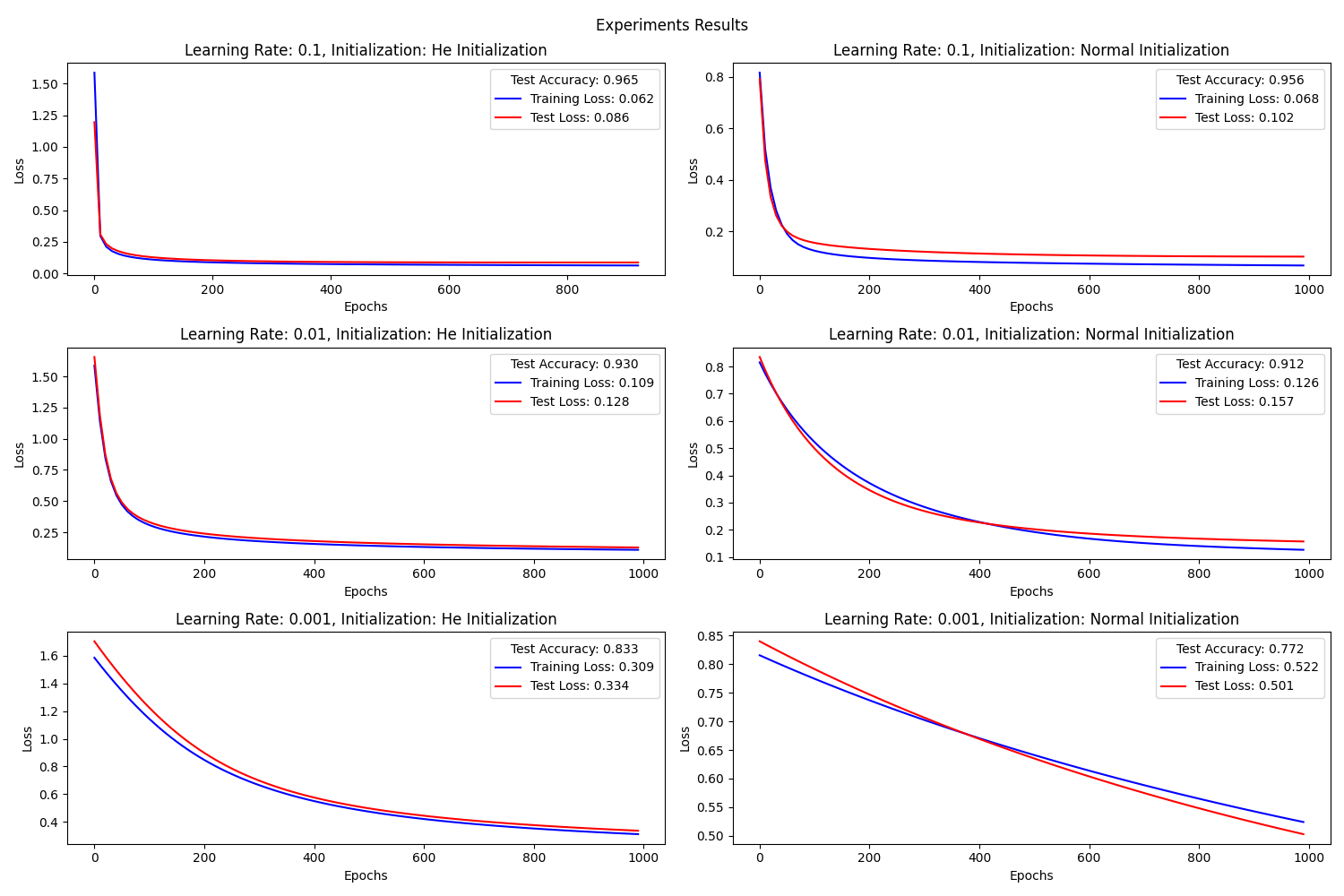
Table 1: All experiment configurations

We conducted **all combinations** of the experiments shown in table 1.

Since the decent algorithm is a heuristic iteration optimization algorithm the learning rates play a crucial role in the convergence rate and in achieving local or global minimum. A small learning rate can cause a slow convergence or none at all. A too big learning rate could cause oscillations in loss convergence and may end up in a non-optimized model.

Furthermore, in GD algorithms, when the dataset is limited in size, as we had in this assignment, the weights initializations can have a great effect on the convergence rate and the final model fine-tuned weights.

## Results and Analysis

Figure 2: All experiment training and testing losses plots – top to bottom: learning rates 0.1, 0.01, 0.001. Left He initialization. Right: normal initialization.

* As expected, A too-high learning rate was too slow to converge. With a relatively high learning rate of 0.,1 we got fast convergence and it was also “stable” to not cause loss oscillations.
* The He Initialization did improve the convergence rate by having stable gradients.
* The best result, seen in Figure 1 and the top left of Figure 2, is with a learning rate 0.1 and he initialization. It was best in terms of test accuracy and lowest test loss.
* Strengths and weaknesses:
* The single layer with gradient descent model is a strong model because it reaches good accuracy fast. It is also simple and light on hardware consumption, especially with tabular small feature datasets. But it has its weaknesses, the first being it highly depends on the hyperparameters as seen in Figure 2, and the dataset labels, size, and its features.

### 

### Model’s weights and biases:

|  |  |  |
| --- | --- | --- |
| Model’s learnable parameter | Shape | output |
| W | (num\_features, 1) = (30, 1) | *[-0.12239466538368492, -0.6240584065446445, -0.3091616895215014, -0.040622776737171455, -0.18613240116829266, -0.22017592775013775, -0.4273080974421825, -0.8121965613751213, -0.41000495485251026, 0.3490237968403959, -1.0140605579748623, 0.046170652769180656, -0.6333119397322726, -0.7779752846083727, 0.09644302636240362, 0.6509392924154866, 0.40084682378541114, -0.14860160216450363, 0.16397575898746916, 0.4021082892572181, -1.5204364651363507, -0.9275590409806291, -0.5701598777239533, -1.032247065528047, -0.4098857845159833, -0.5593456616844367, -0.7608466363385203, -0.8979880253122238, -0.3697128047526266, 0.05327887273338164]* |
| b | (1, 1) | *[0.37718557531896707]* |

Table 2: Weights and bias for the best configuration.

As requested, we attach in Table 2 the final trained **best** weights and bias. Run code and get the rest.