

+FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 4
“Logistic Regression as a Neural Network”

Performed by
Chernobrovkin T.V. (412642)
Academic group J4133c
Accepted by
Gladilin P.E.

St. Petersburg
2023

Goal

Train a logistic regression model and implement SGD and Adam optimizers based on GD. Compare the losses and accuracy of the model depending on the optimizer and the learning rate.

Problem

1. Apply the logistic regression method using the functions in the notebook «Logistic Regression as a Neural Network – BP alg.ipynb» to predict the biological response of a molecule data: *bioresponse.csv*,

description from Kaggle: *“The data is in the comma separated values (CSV) format. Each row in this data set represents a molecule. The first column contains experimental data describing a real biological response; the molecule was seen to elicit this response (1), or not (0). The remaining columns represent molecular descriptors (d1 through d1776), these are calculated properties that can capture some of the characteristics of the molecule - for example size, shape, or elemental constitution. The descriptor matrix has been normalized.”*

Use 75% of the dataset to train the model, and the rest of the data to estimate its accuracy.

2. Modify `optimize()` function to implement the **stochastic gradient descent (SGD)** method and **Adam optimization** method using the numpy library. Apply them to solve the problem from p.1.

3. For three modifications of gradient descent (GD, SGD and Adam) plot the learning curves (dependence of the value of the loss function on the iteration number), apply models with different values of the learning rate (at least 5 different learning rates). How does it affect the accuracy of the model?

4. Compare the accuracy of the models fitted with various BP algorithms.

Theory

Logistic regression or logit model is a binomial regression model. As with all binomial regression models, the goal is to best model a simple mathematical model with a set of real-world observations. In other words, to relate the binomial random variable denoted in general terms to a vector of random variables. Logistic regression is a special case of the generalized linear model. It is widely used in machine learning.

Gradient descent is a numerical method of finding the local minimum or maximum of a function by moving along a gradient, one of the main numerical methods of modern optimization.

It is actively used in computational mathematics not only for direct solution of optimization (minimization) problems, but also for problems that can be rewritten in the language of optimization (solution of nonlinear equations, search for equilibria, inverse problems, etc.). The gradient descent method can be used for optimization problems in infinite-dimensional spaces, for example, for numerical solution of optimal control problems.

Stochastic gradient descent is an optimization algorithm that differs from conventional gradient descent in that the gradient of the function being optimized is considered at each step not as the sum of the gradients from each element of the sample, but as the gradient from a single, randomly selected element.

The problem with the previous algorithm is that in order to determine a new approximation of the weight vector it is necessary to compute the gradient from each element of the sample, which can slow down the algorithm considerably. The idea of speeding up the algorithm is to use only one element, or some subsample, to calculate a new approximation of the weights.

Adam optimizer is the extended version of stochastic gradient descent which could be implemented in various deep learning applications such as computer vision and natural language processing in the future years. It is an optimization algorithm that can be an alternative for the stochastic gradient descent process. The name is derived from adaptive moment estimation. The optimizer is called Adam because uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network. Adam is proposed as the most efficient stochastic optimization which only requires first-order gradients where memory requirement is too little.

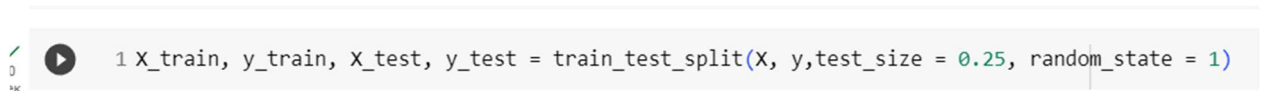
In Adam instead of adapting learning rates based on the average first moment as in RMSP, Adam makes use of the average of the second moments of the gradients. Adam. This algorithm calculates the exponential moving average of gradients and square gradients. And the parameters of β_1 and β_2 are used to control the decay rates of these moving averages.

Materials and methods

In this task, all calculations were performed on the student's personal laptop in Google Colab. The work was performed in the Python programming language.

Results

1. 1. the linear logistic regression method presented in the proposed assignment notebook was run. 75% of the data were used for training the model and 25% for validation (Figure 1).

A screenshot of a Google Colab code cell. It features a play button icon on the left and a single line of Python code: `1 X_train, y_train, X_test, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)`. The code is displayed in a light gray background with syntax highlighting: `X` and `y` are in blue, `train_test_split` is in green, and the other tokens are in black. The cell is marked as 'Done' with a green checkmark icon on the far left.

```
1 X_train, y_train, X_test, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

Figure 1 - Data splitting

2. 2- The function was modified to implement the stochastic gradient descent (SGD) method and the Adam optimization method using the numpy library. The results are shown in Figure 2 and Figure 3, respectively.

```
1 def SGD(w, b, X, Y, num_iterations, learning_rate, print_cost = False, batch_size=256):
2     costs = []
3
4     for i in range(num_iterations):
5         random_batch_x = np.random.choice(X.shape[1], batch_size, replace=False)
6         random_batch_y = np.random.choice(Y.shape[0], batch_size, replace=False)
7
8         X = np.take(X, random_batch_x, axis=1)
9         Y = np.take(Y, random_batch_x, axis=0)
10
11         grads, cost = propagate(w,b,X,Y)
12
13         dw = grads["dw"]
14         db = grads["db"]
15
16         w -= learning_rate*dw
17         b -= learning_rate*db
18
19         # Record the costs
20         if i % 100 == 0:
21             costs.append(cost)
22
23         # Print the cost every 100 training iterations
24         if print_cost and i % 100 == 0:
25             print ("Cost after iteration %i: %f" %(i, cost))
26
27
28     params = {"w": w,
29              "b": b}
30
31     grads = {"dw": dw,
32             "db": db}
33
34     return params, grads, costs
```

Figure 2 – SGD

```

1 def adam(w, b, X, Y, num_iterations, learning_rate, print_cost = False, b1=0.9, b2=0.999):
2
3     costs = []
4     sw, sb = 0, 0
5     rw, rb = 0, 0
6     num_stabilizer: float = 1e-8 # constant for numerical stabilization
7
8     for i in range(num_iterations):
9         # Cost and gradient calculation
10        grads, cost = propagate(w,b,X,Y)
11        if math.isnan(cost):
12            break
13        # Retrieve derivatives from grads
14        dw = grads["dw"]
15        db = grads["db"]
16
17        sw = b1*sw + (1. - b1) * dw
18        rw = b2*rw + (1. - b2) * dw**2
19        cor_sw = sw / (1. - b1**(i+1))
20        cor_rw = rw / (1. - b2**(i+1))
21
22        sb = b1*sb + (1. - b1) * db
23        rb = b2*rb + (1. - b2) * db**2
24        cor_sb = sb / (1. - b1**(i+1))
25        cor_rb = rb / (1. - b2**(i+1))
26
27        # update rule
28        w -= learning_rate * cor_sw / (np.sqrt(cor_rw) + num_stabilizer)
29        b -= learning_rate * cor_sb / (np.sqrt(cor_rb) + num_stabilizer)
30
31        # Record the costs
32        if i % 100 == 0:
33            costs.append(cost)
34
35        # Print the cost every 100 training iterations
36        if print_cost and i % 100 == 0:
37            print ("Cost after iteration %i: %f" %(i, cost))
38
39        params = {"w": w,
40                  "b": b}
41
42        grads = {"dw": dw,
43                 "db": db}
44
45    return params, grads, costs

```

Figure 3 – Adam

3. For three modifications of gradient descent (GD, SGD and Adam), learning curves (dependence of the loss function value on the iteration number) were plotted and models with different values of learning rate were applied.

The result for GD is shown in Figure 4. As can be seen from the graph, GD with a learning rate equal to 0.1 m 0.05 has the lowest error. When the learning rate is equal to 0.5, oscillations are observed. This may be due to the fact that at a large step the optimizer constantly overshoots the minimum point and oscillates around it for a long time.

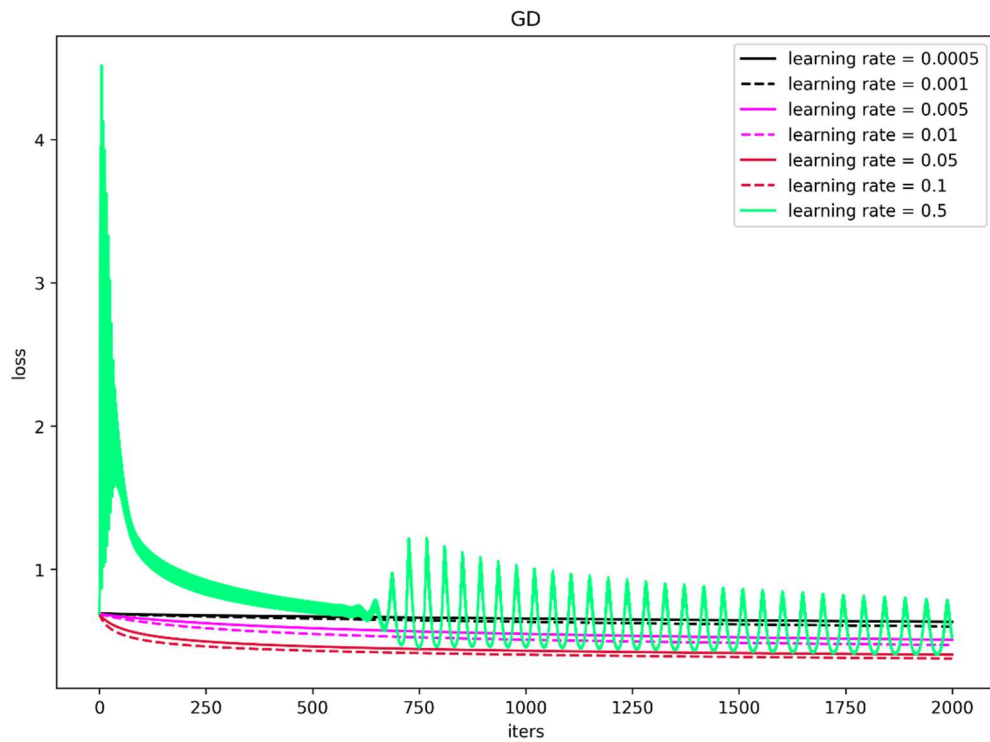


Figure 4 - Training result for GD

The results for SGD are presented in Figure 5. As can be seen from the graph, SGD with a learning rate of 0.5 has the lowest error. In contrast to GD for this optimization method, a higher learning rate was effective.

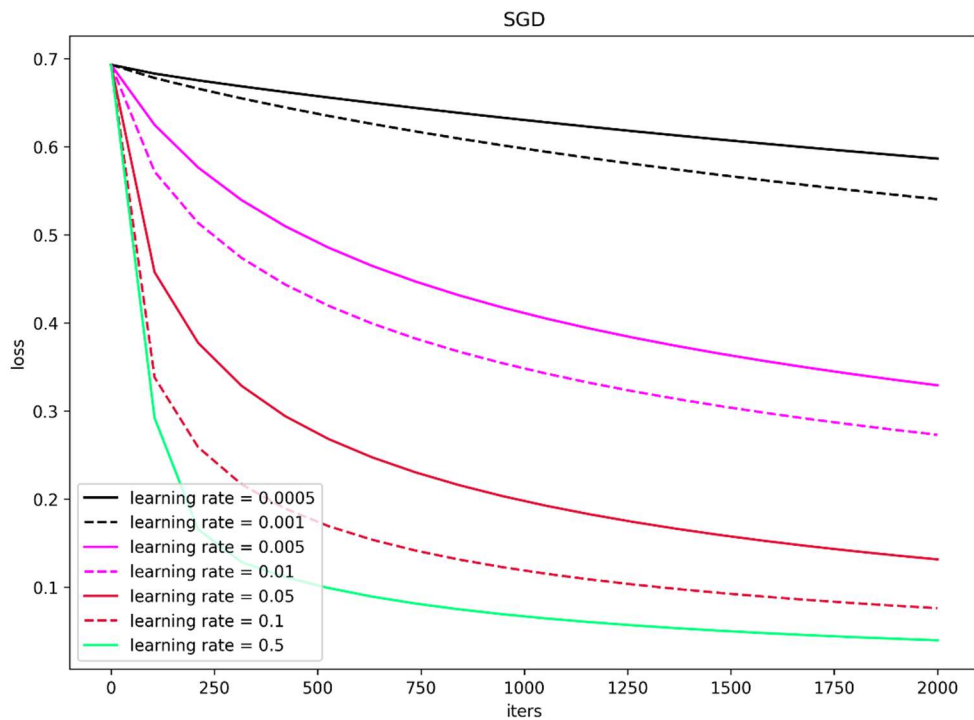


Figure 5 - Training result for SGD

The results for Adam are presented in Figure 6. As can be seen from the graph, Adam with a learning rate of 0.005 has the lowest error. For this method, as well as for the GD method, large learning rates are not suitable, in this case they are not even displayed on the graph.

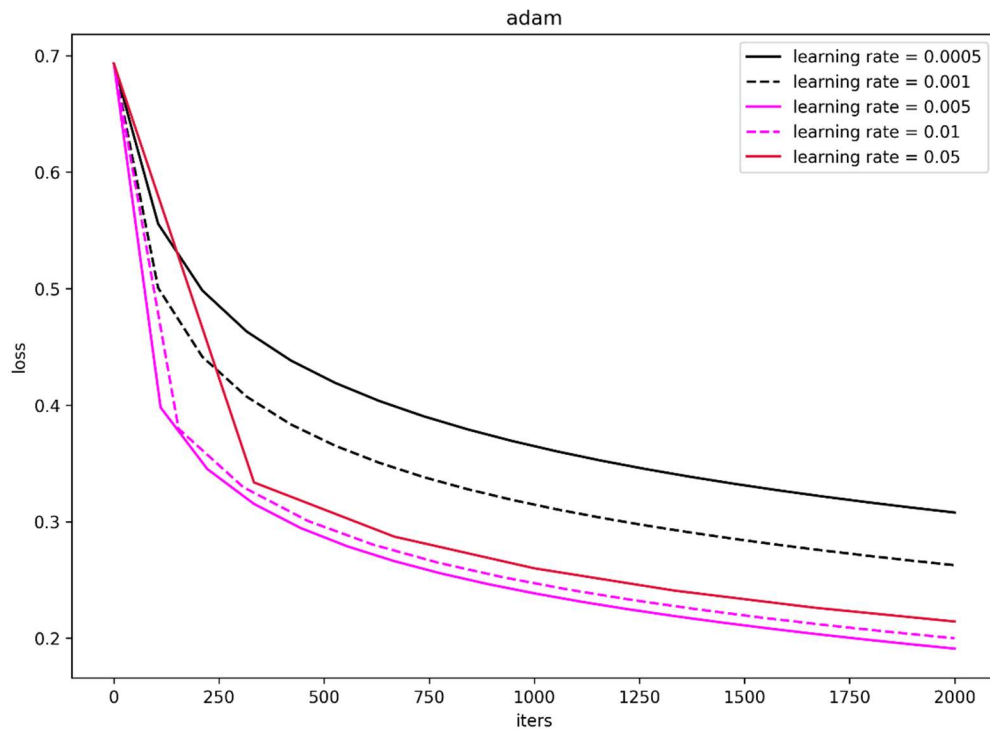


Figure 6 - Training result for Adam

4. The comparison of regression accuracies with the considered optimizers and learning rates is presented in Table 1. According to the table, the model with GD optimizer and learning rate 0.05 has the highest accuracy on the test set.

Table 1 - Accuracy values of regression models depending on optimizer and learning rate.

rate optimizer	0.0005	0.001	0.005	0.01	0.05	0.1	0.5
GD	71.1087	71.9616	74.6268	76.2260	77.6119	76.7590	71.0021
SGD	65.9914	68.1236	69.2963	72.3880	69.6162	69.2963	69.4029
Adam	74.8400	74.0938	74.2004	73.9872	74.2004	-	-

Conclusion

In this work, a logistic regression model was trained and SGD and Adam optimizers based on GD were implemented. The loss and accuracy of the regression model with the implemented optimizers were compared at different learning rates. Graphs and table were plotted based on the

obtained results. Logistic regression with GD optimizer and learning rate equal to 0.05 has the best results.

Appendix

GitHub link:

https://github.com/LesostepnoyGnom/homework_ML/blob/main/task4/Task_4_Chernobrovkin_J4133c.ipynb