Report

Improvement DNN

In this task, various methods of preprocessing data, initialization during the implementation of the model, various methods for calculating arguments were considered.

1. Initialization

Three initialization methods have been applied.

Zero initialization. All input arguments (W, b) are initialized with zeros.
Random initialization. The values of the input arguments (W) are obtained randomly.
He initialization. It differs from the *random initialization* that the increase (multiplication) by 10 uses a formula:

$$\sqrt{\frac{2}{\text{dimension of the previous layer}}}$$

During the assignment, each of the initialization methods was implemented in Jupyter notebook using the Python language. The results obtained are fully consistent with the planned.

Conclusions

Comparative characteristics of various initialization methods and comments on the methods are given in the table.

Model	Train accuracy	Problem/Comment
3-layer NN with zeros initialization	50%	fails to break symmetry
3-layer NN with large random initialization	83%	too large weights
3-layer NN with He initialization	99%	recommended method

2. Regularization

In this part of the assignment, three methods of regularizing the input data were investigated.

1 Non-regularized model. We do not apply any regularization
2 L2 Regularization.
3 Dropout. We will disable some neurons

3.1 - Forward propagation with dropout Disconnection of neurons occurs in the direct direction of the network training
3.2 - Backward propagation with dropout Neurons are disconnected during backward propagation of calculations.

Detailed guidelines for the implementation of this task are provided in the Jupyter notebook file. They were accurately implemented. The results coincided with the expected.

Conclusions

Results and conclusions are tabulated.

model	train accuracy	test accuracy
3-layer NN without regularization	95%	91.5%
3-layer NN with L2-regularization	94%	93%
3-layer NN with dropout	93%	95%

3. Gradient Checking

This task will test the gradient. The technique of testing is fully described in Jupyter Notebook and implemented correctly. The result also coincided with the intended.
This confirms the usefulness of using gradient checking.

In conclusion, you can repeat the main thesis for each of the tasks performed:

Intializing

- Different initializations lead to different results
- Random initialization is used to break symmetry and make sure different hidden units can learn different things
- Don't intialize to values that are too large
- He initialization works well for networks with ReLU activations.

Regularization

- Regularization will help you reduce overfitting.
- Regularization will drive your weights to lower values.
- L2 regularization and Dropout are two very effective regularization techniques.

Gradient Checking

•Gradient checking verifies closeness between the gradients from backpropagation and the numerical approximation of the gradient (computed using forward propagation).
•Gradient checking is slow, so we don't run it in every iteration of training. You would usually run it only to make sure your code is correct, then turn it off and use backprop for the actual learning process.