Homework for Lab 1.4

Deadline: Tuesday, November 18, at 9:00 a.m.

Notes:

- Students are **randomly selected** to present their solution.
- Your solution is **graded**.
- If you are unable to present three times, you will receive **zero points** for the entire homework activity.
- You should attempt all tasks at least partially - it might help you get partial credit.
- If necessary, ask questions and discuss the tasks in the Moodle forum.

Tasks

1.4.1 A simple Neural Network

Work through the Jupyter notebook and complete the Task 1.4.1. Open in Colab

a) Initialize weights and biases using the `init_parameters` method with a hidden size of 8. Check the shapes of the initialized parameters to confirm they are correct.

b) Implement the `sigmoid` and `sigmoid_derivative` functions. Use the `plot_activation_function` method to make sure the functions are implemented correctly.

c) Implement the `forward` function to compute the output of the neural network. Implement the steps as discussed in the Lab:

$$\begin{aligned} Z^{[1]} &= W^{[1]} \cdot X + b^{[1]} \\ A^{[1]} &= \text{sigmoid}(Z^{[1]}) \\ Z^{[2]} &= W^{[2]} \cdot A^{[1]} + b^{[2]} \\ A^{[2]} &= \text{sigmoid}(Z^{[2]}) \end{aligned}$$

If you encounter issues, check the input shape and the shapes of the weights and biases.

d) Finally, implement the `update_parameters` function. Follow the description in the notebook and train the network for 100 epochs.

1.4.2 Objectified Layers

Later on, we want to change the architecture of the network. Using the code from Task 1.4.1, we would need the change the code at almost every step of the computation. To avoid this, you will create objectified versions of the layers, the network, and the optimizer. A generic layer object is provided in `utils.Layer`. The other layers will inherit from this class. An instance of the `NeuralNet` class is then built by providing a list of layers:

```
NN = NeuralNet(layers=[
    Dense(...),
    Sigmoid(),
    Dense(...),
    Sigmoid()
])
```

a) Complete the `Dense` layer class.

b) Complete the `Sigmoid` layer class.

c) Complete the `BinaryCrossEntropy` loss class.

d) Complete the `GradientDescent` class.

e) Generate a centroids dataset and train the network for 100 epochs.

1.4.3 Initialization Matters

The methods `constant` and `simplescaled` are two initialization methods for the weights of a dense layer. They are already implemented. The `constant` method initializes all weights to the same constant value. The `simplescaled` method (used by default) initializes the weights to values drawn from a normal distribution with mean 0 and standard deviation $\sigma = 0.01$. Use a `ring`-type dataset for this task and a two-layer Neural Network with a hidden size of your choice.

a) Initialize the weights of the `dense` layers with the `constant` method. Train the model for 500 epochs with a learning rate of 1. Plot the training and validation loss and the decision boundary on the test data. Describe the results.

b) Exchange the initialization method to `simplescaled` and repeat the training three times from scratch. Plot the training and validation loss and the decision boundary for each run. Describe the results.

c) Create a copy of or inherit from the `Dense` class. Implement and use the `Xavier` initialization method. Repeat the training three times from scratch. Plot the training and validation loss and the decision boundary for each run. Describe the results.

1.4.4 We have to go deeper!

Use the `ring`-type dataset for this task.

a) Implement a deep neural network with three hidden dense layers (four layers in total). Set the hidden size of each layer to 8 and initialize the weights with the `Xavier` method from task 1.4.3. Train the model for 500 epochs with a learning rate of 1. Plot the training and validation loss and the decision boundary on the test data. Explain the results.

b) Implement the `ReLU` activation function as a new layer. Test the `forward` and `backward` methods of your `ReLU` layer.

c) Which activation functions in your neural network could be replaced with the `ReLU` function? Implement the changes and train the model for 500 epochs with a learning rate of 1. Plot the training and validation loss and the decision boundary on the test data. Explain the results.

1.4.5 The Spiral

Find a suitable but small architecture to accurately fit the `spiral` dataset.