Neural Networks and Deep Learning

Cracow University of Technology

Task 2, lab assignment 4:

Neural Network Training with Regularization

Objective: To implement a multiclass classification neural network with L1 and L2 regularisation and use it to train and validate the iris dataset.

Instructions:

1. Open a Python environment and import the necessary libraries:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

2. Load the iris dataset:

```
iris = load_iris()
X, y = iris.data, iris.target
```

3. Normalize the input data:

```
X = X / np.max(X)
```

4. One-hot encode the target variable:

```
encoder = OneHotEncoder()
y = encoder.fit_transform(y.reshape(-1, 1)).toarray()
```

5. Split the dataset into training and test sets:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_s
ize=0.25, random_state=42)
```

6. Define the neural network architecture:

```
input_size = X_train.shape[1] # input layer size
hidden_size = 32 # hidden layer size
output_size = y_train.shape[1] # output layer size
```

7. Initialize weights and biases:

```
W1 = np.random.randn(input_size, hidden_size) * 0.01
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size) * 0.01
b2 = np.zeros((1, output_size))
```

8. Define the hyperparameters:

```
learning_rate = 0.1
num_epochs = 1000
batch_size = 16
l1_lambda = 0.001
l2_lambda = 0.001
```

9. Define the necessary functions:

- Categorical cross-entropy loss function:

```
def categorical_cross_entropy_loss(y_true, y_pred):
    # avoid numerical instability by adding a small constant to log
    eps = 1e-15
    y_pred = np.clip(y_pred, eps, 1 - eps)
    loss = -np.sum(y_true * np.log(y_pred)) / y_true.shape[0]
    return loss
```

- Derivative of categorical cross-entropy loss function:

```
def derivative_categorical_cross_entropy_loss(y_true, y_pred):
    return (y_pred - y_true) / y_true.shape[0]
```

- Softmax activation function:

```
def softmax(x):
    # avoid numerical instability by subtracting max value
    x -= np.max(x, axis=-1, keepdims=True)
    exp_x = np.exp(x)
    return exp_x / np.sum(exp_x, axis=-1, keepdims=True)
```

- Derivative of softmax activation function:

```
def derivative_softmax(x):
    s = softmax(x)
    return s * (1 - s)
```

- ReLU activation function:

```
def relu(x):
    # avoid numerical instability by subtracting max value
    x -= np.max(x, axis=-1, keepdims=True)
    exp_x = np.exp(x)
    return exp_x / np.sum(exp_x, axis=-1, keepdims=True)
```

- Derivative of ReLU activation function:

```
def derivative_relu(x):
    s = softmax(x)
    return s * (1 - s)
```

- L1 regularization term:

```
def L1_reg(lambda_, W1, W2):
    return lambda_ * (np.sum(np.abs(W1)) + np.sum(np.abs(W2)))
```

- Derivative of L1 regularization term:

```
def derivative_L1_reg(lambda_, W):
    return lambda_ * np.sign(W)
```

- L2 regularization term:

```
def L2_reg(lambda_, W1, W2):
    return lambda_ * (np.sum(np.square(W1)) + np.sum(np.square(W2)))
```

- Derivative L2 regularization term:

```python
def derivative_L2_reg(lambda_, W):
    return lambda_ * 2 * W
```

Task:

10. Train a two-layer neural network with ReLU activation function in the hidden layer, Softmax activation function in the output layer, and categorical cross-entropy loss. Add both L1 and L2 regularization terms to the loss function.

11. Test the model using test set.