Indraprastha Institute of Information Technology Delhi

CSE641 - Deep Learning

Assignment 02

Date: 12-02-2021 **Marks: 120+80**

Instructions:

1. Use of any deep learning libraries is not allowed.
2. Each group member must do at least one of the following tasks. But all should know the working of all three tasks.
3. The assignment can be submitted in a group of maximum of three members. Group details can be found here.
4. For Plagiarism, institute policies will be followed strictly.
5. Extension policy as discussed in class can be found here. Spamming +1s on Classroom will lead to usage of your group's one extension.
6. Make sure to use Pickle or any other library to save all your trained models. There will not be enough time during the demo to retrain your model. This is a strict requirement. You must upload your best models on Classroom to reproduce your results during the demo. If you are not able to reproduce your results during the demo then no marks will be given.
7. You need to submit README.pdf, Code files (it should include both .py files and .ipynb files), Output.pdf, and models dumped after training.
8. Mention methodology, helper functions, preprocessing steps, any assumptions you may have, and contribution of each member in README.pdf.
9. Mention your sample outputs in the output.pdf.
10. You are advised to prepare a well-documented code file.
11. Submit code, models, readme, and output files in ZIP format with the following name: A2_Part1_Member1_Member2_Member3.zip and A2_Part2_Member1_Member2_Member3.zip
12. Use classroom discussion for any doubt.

****

Assignment: In this assignment, you need to implement a Deep Learning Toolkit for training a multi-layer perceptron on the given subset of MNIST dataset. The assignment needs to be completed in two parts. Both parts have separate deadlines. Note that you are allowed to submit part-2 with part-1 deadline. However, part-1 must be submitted on or before its deadline, else, late submission policy will apply.

[PART - 1] **Deadline: 21-02-2021, 11:59 PM**

1. Create the neural network using the descriptions given below. Implement forward propagation and backward propagation from scratch on the constructed neural network. You are required to create the dl_toolkit.py file (described later in the assignment) and use the following conditions for this question. **[60 points]**

    - **Network Structure:** Input 784, output 10. You are free to use any number of hidden layers with any number of neurons.

Note: You may consider designing a smaller network (i.e. lesser number of hidden layers, or lesser number of neurons in each hidden layer) depending upon the resources available at your disposal.

- **Optimizer:** Gradient-descent optimizer (must make batch_size parameter = length of entire dataset in this case).
- **Activation:** Show results using these activation functions for hidden layers: Sigmoid, Tanh, ReLU. Output layer will always have Softmax activation.
- **Initialization:** Random

2. Implement the following gradient descent optimizers from scratch and do a thorough analysis on the output of each one of them. Use mini-batch size = 64. The following gradient descent optimizers are to be implemented: **[60 points]**

    - Gradient Descent with Momentum
    - Nestrov's Accelerated Gradient
    - AdaGrad
    - RMSProp
    - Adam

    Note: The best configuration found in Q1 must remain unchanged except the optimizer.

[PART - 2] **Deadline: 26-02-2021, 11:59 PM**

1. Implement the following weight initialization techniques from scratch and do a thorough analysis on the output of each one of them. **[30 points]**
    - He
    - Xavier

    Note: The best configuration found in [Part -1, Q2] must remain unchanged except the weight initialization techniques.

2. Implement the following regularizations from scratch and do a thorough analysis on the output of each one of them. **[50 points]**
    - L1 Regularization
    - L2 Regularization
    - Dropouts

    Note: The best configuration found in [Part -2, Q1] must remain unchanged except the regularization technique.

Expected Outcome of the assignments:

- For every part within each question, visualize the learning using the following plots:
    - Training Loss vs Number of Epochs
    - Validation Loss vs Number of Epochs
    - ROC curve
    - Confusion matrix

- Learning report (output.pdf) for each question. Do a thorough comparison between different optimizers, activation functions, weight initialization methods in answers to their respective questions.
- Mention your assumptions and methodology in Readme.pdf.

Instructions for implementation:

1. **Dataset:** MNIST dataset. Train: 10K, Val: 2K.
   Note: You may reduce the size of the training dataset depending upon the resources available at your disposal. But the validation set should remain the same.
2. **Format of the class, functions, and parameters:** The implementation of this package should be consistent with the given instructions. We will run a test file to check the same.

Name of the Python file to be used: `dl_toolkit.py`
Class to implemented: `MLPClassifier`
Class functions:

1. **`__init__(layers, learning_rate, activation_function, optimizer, weight_init, regularization, batch_size, num_epochs, dropouts, **kwargs)`**
   - **`layers:`** A list of number of neurons in each layer, starting from input layer, intermediate hidden layers and output layer. For example, a list consisting of [784, 200, 50, 10] means the number of input features to the NN is 784, and it consists of 2 hidden layers with 200, 50 neurons in hidden layers 1 and 2, and it has 10 neurons in the output layer. Since we are using MNIST dataset, the number of neurons in the input and output layer will remain constant i.e. 784 for input and 10 for output. No default value is there, this is a necessary argument.
   - **`learning_rate:`** Learning rate of the neural network. Default value = 1e-5.
   - **`activation_function:`** A string containing the name of the activation function to be used in the hidden layers. For the output layer use Softmax activation function. Default value = “relu”.
   - **`optimizer:`** A string containing the name of the optimizer to be used by the network. Default value = “gradient_descent”.
   - **`Weight_init:`** “random”, “he” or “xavier”: String defining type of weight initialization used by the network. Default value = “random”.
   - **`Regularization:`** A string containing the type of regularization. The accepted values can be “l1”, “l2”, “batch_norm”, and “layer_norm” . The default value is “l2”.
   - **`Batch_size:`** An integer specifying the mini batch size. By default the value is 64.
   - **`Num_epochs:`** An integer with a number of epochs the model should be trained for.
   - **`dropout:`** An integer between 0 to 1 describing the percentage of input neurons to be randomly masked.
   - **`**kwargs:`** A dictionary of additional parameters required for different optimizers. By default it is None, however, you must initialize different parameters of optimizers with some valid input value for convergence. (This parameter will not be used in the test file)

   Output: (void)

2. `fit(X, Y)`

- **X:** a numpy array of shape (num_examples, num_features).
- **Y:** a numpy array of shape (num_examples): This array contains the classification labels of the task.

Output: (void)
Note: This function should log the loss after some minibatches (you can choose this arbitrarily) and after the complete epoch.

3. `predict(X):`

- **X:** a numpy array of shape (num_examples, num_features)

Output: numpy array of shape (num_examples) with classification labels of each class.

4. `predict_proba(X):`

- **X:** a numpy array of shape (num_examples, num_features)

Output: numpy array of shape (num_examples, num_classes): This 2d matrix contains the probabilities of each class for all the examples.

5. `get_params():`

Output: An array of 2d numpy arrays. This array contains the weights of the model.

6. `score(X, y):`

- **X:** a numpy array of shape (num_examples, num_features): This 2d matrix contains the complete dataset.
- **Y:** a numpy array of shape (num_examples): This array contains the classification labels of the task.

Output: (float) Classification accuracy given X and y.