CS 663 - Machine Learning Spring, 2024

Assignment 04 - MLP (Multi-Layer Perceptron)

There are a few goals for this assignment, specifically:

1. To build a basic MLP in PyTorch and execute it against a real dataset.
2. To tune the MLP to get maximum performance.
3. To build a basic MLP from basic Python in order to deepen your understanding of neural networks.

Background

We covered the MLP extensively. While the basic MLP appears to be a linear or logistic ensemble, there are many configuration changes which make it ideal for many problems in machine learning. In this assignment, you will first create a basic MLP and determine its performance. You will then increase the performance of the MLP by making changes to the basic MLP.

Requirements (Process)

Accept the assignment by navigating to https://classroom.github.com/a/B-3bPXH-. There are three required portions for this assignment, all detailed below:

1. Create a basic Multi-Layer Perceptron instance using PyTorch and determine its performance.
2. Make changes to the basic Multi-Layer Perceptron instance to increase its performance.

In addition to the required (above), there is one optional part:

3. Implement your own Multi-Layer Perceptron from basic Python.

Create a Basic MLP

Create a basic MLP instance using PyTorch. This basic MLP must have the following configuration:

- One hidden layer with 5 units

- Optimiser = SGD (stochastic gradient descent)
- Activation = ReLU (rectified linear units)
- Learning rate = 0.0001

Determine the performance of the MLP on the Wine Quality dataset. (This dataset is in the repository, but it is also available from UCI and from Kaggle.) Do the following:

- Use the column “quality” as the target and all other columns as features
- Split the dataset into train (70%), validation (15%) and test (15%)
- Use MSE (mean squared error) as the performance metric

You must report performance on the test set, and you must not use the test set for any purpose other than determining the model’s performance. You may use validation to determine good hyperparameter values for your model.

Make Changes to the Basic MLP

Make any changes you like to the basic MLP to ensure the performance of your model increases (or conversely to ensure that the model’s loss decreases). Keep track of the changes you make. Discuss which changes were successful and which were unsuccessful. Provide a thorough explanation about why the changes you made led to performance changes.

Implement your own MLP (optional)

Implement your own basic MLP algorithm. In this case, you are allowed to use the following:

- Basic Python data structures (lists, etc.) and control (loops, etc.)
- Numpy data structures (arrays, etc.) and functions (matrix multiplication, etc.)
- Pandas data structures (DataFrame, Series) and functions

Your implementation must have the following functions:

- __init__(hidden_layer_tuple, activation_tuple) — where:
    - hidden_layer_tuple defines the number of inputs starting at the input layer and ending at the output layer — eg. with the value (11, 5, 1) there are 11 units in the input layer, 5 units in the hidden layer and — as a regression MLP — 1 unit in the output layer.
    - activation_layer_tuple defines the activation function used on each layer — eg. with the value ('relu', 'sigmoid'), the activation function used in the first layer is “ReLU” and the activation function on the second layer is sigmoid. Only values 'relu', 'sigmoid' and 'none' are allowed here.
- forward(X) — which will take a matrix (X) and forward it through the MLP, producing a vector of hypotheses.
- backprop(y) — which will take a vector (y) and adjust the weights and biases in the MLP according to the MSE loss.

You are allowed to use code from other locations (eg. FreeCodeCamp, TowardsDataScience, etc.), but you must cite the sources of all places which you use or from which you get inspiration.

Grading

Grades for this assignment will be evaluated as one of the following four outcomes:

- Exemplary: Submission correctly implements and tests all required parts defined above as well as the optional part. In addition, the implementation must meet implicit correctness criteria (eg. code is well-organised, uses a well-recognised standard for comments, is well-documented, is reusable, maintainable, etc.). Additionally, **you must do at least one of the following**:
    - The performance of the improved model shows is at least 10% (relative) over the baseline model.
    - Complete the optional Requirement 3, i.e. implementing your own MLP from basic Python.
- Satisfactory: Submission correctly answers all required parts defined above but does not show an improvement (or a sufficiently large improvement) in performance.
- Partial: Submission makes progress toward meeting all required parts.
- Unassessable: Not enough information to determine progress toward all required parts.

Avoid late submissions.

Submission

To submit your assignment for evaluation, follow all of the instructions:

1. Check into github your completed implementation, which must include a (single) Jupyter notebook with all cells executed and may also include a (single) python file, KMeans.py.
2. On Canvas, submit the URL for your completed assignment in the space provided. Add any comments, instructions for the grader, notes, etc. by changing the README.md file in the repository.