CS335 : AI/ML 2019
Lab Assignment 5

Please read the following important instructions before getting started on the assignment.

1. The assignment should be completed individually.
2. Do not look at solutions to this assignment or related ones on the Internet.
3. All theory questions must be submitted in the single pdf file
4. All the hyperparameters must be listed in single pdf under Hyperparameters section and resources consulted must be duly listed in the References section of pdf file. Do not upload multiple pdfs.
5. **Upload Guidelines :** Put all the assignment related files in folder with the convention **roll_no** and zip the folder(do not compress in *.gz) .
6. **Not following folder guidelines will attract penalty**
7. All source code are checked with code plagiarism checker. Strict action will be taken again defaulters
8. **Any erroneous naming conventions will be penalised**
9. 1 mark for correct structure

In this assignment, you will design layers required to build both a feed-forward neural network (also popularly known as a Multilayer Perceptron classifier) and the Convolution Neural Networks. Feed-forward NNs and CNNs form the basis for modern *Deep Learning* models. However, the networks with which you will experiment in this assignment will still be relatively small compared to some of modern instances of neural networks.

Some readings on MLP
A crash course to Multilayered Perceptrons

Data Sets

Link to data- will be uploaded.

To test your code, you are provided a set of toy examples (1 and 2 below), each a 2-dimensional, 2-class problem. You are also provided tools to visualise the performance of the neural networks you will train on these problems. The plots show the true separating boundary in blue. Each data set has a total of 10,000 examples, divided into a training set of 8000 examples, validation set of 1000 examples and test set of 1000 examples.

Data set 3 is the MNIST data set collection of images handwritten digits.

Data set 4 is the CIFAR-10 data set collection of images of commonly seen things segregated into 10 classes.

task1 task2

XOR Data (Left)

The input `X` is a list of 2-dimensional vectors. Every example X_i is represented by a 2-dimensional vector `[x,y]`. The output y_i corresponding to the ith example is either 0 or 1. The labels follow XOR-like distribution. That is, the first and third quadrant has same label (y_i = 1) and the second and fourth quadrant has same label (y_i = 0) as shown in Figure 1.

Semicircle Data (Right)

The input `X` is a list of 2-dimensional vectors. Every example X_i is represented by a 2-dimensional vector `[x,y]`. The output y_i corresponding to the ith example is either 0 or 1. Each set of label (y_i = 1 and y_i = 0) is arranged approximately on the periphery of a semi circle of unknown radius `R` with some spread `W` as shown in Figure 2.

MNIST Data

We use the MNIST data set which contains a collection of handwritten numerical digits (0-9) as 28x28-sized binary images. Therefore, input `X` is represented as a vector of size 784. These images have been size-normalised and centred in a fixed-size image. MNIST provides a total 70,000 examples, divided into a test set of 10,000 images and a training set of 60,000 images. In this assignment, we will carve out a validation set of 10,000 images from the MNIST training set, and use the remaining 50,000 examples for training.

CIFAR 10 Data

We use the CIFAR-10 data set which contains 60,000 32x32 color(RGB) images in 10 different classes. The 10 different classes represent **airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks**. There are 6,000 images of each class. These 60,000 images are divided into a test set of 10,000 images and a training set of 50,000 images. In this assignment, we will carve out a validation set of 10,000 images from the CIFAR-10 training set, and use the remaining 40,000 examples for training.

In this assignment, we will code a feed-forward neural network and convolution neural network from scratch using python3 and the numpy library.

Code

The base code for this assignment is available in the compressed file. Below is the list of files present in the directory.

File Name	**Description**
`layers.py`	This file contains base code for the various layers of neural network you will have to implement.
`nn.py`	This file contains base code for the neural network implementation.
`visualize.py`	This file contains the code for visualisation of the neural network's predictions on toy data sets.
`visualizeTruth.py`	This file contains the code to visualise actual data distribution of toy data sets.
`tasks.py`	This files contains base code for the tasks that you need to implement.
`test.py`	This file contains base code for testing your neural network on different data sets.
`util.py`	This file contains some of the methods used by above code files.
`autograder.py`	This is used for testing your Task 1 and Task 2

All data sets are provided in the `datasets` directory.

Methods you need to implement

You will only have to write code inside the following functions. You need to implement these functions for all the layers in the layer.py and nn.py

- `forwardpass`
- `backwardpass`
- `relu_of_X`
- `softmax_of_X`
- `gradient_relu_of_X`
- `gradient_softmax_of_X`
- `computeLoss` : Strictly Use cross-entropy loss for computing loss
- `backpropagate`
- `feedforward`

Layers: Explanation and References

- `Fully Connected Layer`: This layer contains weights in form of `in_nodes X out_nodes` . Forward pass in such a layer is simply matrix multiplication of activations from previous layer and its weights and addition of bias to it. `A_new = A_old X Curr_Wts + Bias`. Similarly, backpropagation updates can be obtained by simple matrix multiplications.

- `Convolution Layer`: This layer contains weights in form of numpy array of form (out_depth, in_depth, filter_rows, filter_cols). Forward pass in such layer can be obtained from correlation of filter with the previous layer for all its depths and addition of biases to this sum. Backwardpass for such a system of weights results in another convolution that you need to figure out.

- `AvgPooling Layer`: This layer is essentially used to reduce the weights used in convolution. It downsamples the previous layer by a factor stride and filter (**in our case both can be assumed to the same**). While downsampling one could have used many operations, including taking mean, median, max, min, etc,. Here we take the **Average** operation for downsampling. This doesn't effect the depth of the input. Forward pass in such layer can be obtained by taking Mean in a grid of elements of `size = filter_size`. Backwardpass for such a system just upsamples the numpy multidimensional array, scaling them by same factor that was used in downsampling.

- `Flatten Layer`: This layer is inserted to convert 3d structure of convolution layers to 1d nodes required for fully connected layers.

Refer to this link and to this link for further clarifications on how to implement forward and backward propagation across average-pooling and convolution layers

Following are some key numpy functions that will speed up the computation: `tile, transpose, max, argmax, repeat, dot, matmul`. To perform element-wise product between two vectors or matrices of the same dimension, simply use the * operator. For further reference on these functions refer to numpy documentation.

A model for CNN layers

A CNN with 2 Convolution layers, 1 Pooling Layer and 1 Fullyconnected layer. Flatten Layer b/w C3 and D4 is implicit in the figure

(Image source: https://www.researchgate.net/figure/Structure-of-a-typical-convolutional-neural-network-CNN_fig3_323938336.)

Task 1: Implement activation functions (1 + 1 + 1 + 2 = 5 marks)

In this task, you will complete the various activation functions namely `relu_of_X`, `softmax_of_X`, `gradient_relu_of_X`, `gradient_softmax_of_X` methods in `layers.py` file.

Testing your implementation

To test your implementations, run the following code.

```
python3 autograder.py -t 1
```

You are allowed to add/remove arguments/functions/code inside code except autograder.py and test.py. If autograder fails to run, we will award 0 marks. Ensure that code is optimized and vectorized and refrain from excess use of for loops.

Task 2: Training and Testing (2 + 2 + 3 + 5 + 2 = 14 Marks)

In this task, you are required to complete the relevant methods in `layers.py` and `nn.py` file. You should (will have to) use the matrix form of the backpropagation algorithm for quicker and more efficient computation. You have to report neural networks with *minimal* topologies. **You are not allowed to use any other activation functions except ReLU, Softmax** . Ensure that code runs with randomized seed values. We will penalise if your code doesn't run properly on random seed values.

Evaluation over Data Sets

Task 2.1: XOR Data set (2 Marks)

Complete `taskSquare()` in `tasks.py`. To test your backpropagation code on this data set, run the following command.

```
python3 test.py 1 seedValue
```

To visualise ground truth, run the following command.

```
python3 visualizeTruth.py 1
```

Task 2.2: SemiCircle Data set (2 Marks)

Complete `taskSemiCircle()` in `tasks.py`.You should be able to achieve > 97% accuracy for both XOR and SemiCircle. To test your backpropagation code on this data set, run the following command.

```
python3 test.py 2 seedValue
```

To visualise ground truth, run the following command.

```
python3 visualizeTruth.py 2
```

Task 2.3: Compute Accuracy over MNIST (using Feed-Forward Neural Network) (3 marks)

The MNIST data set is given as `mnist.pkl.gz` file. The data is read using the `readMNIST()` method implemented in `util.py` file.

Your are required to instantiate a NeuralNetwork class object (that uses FullyConnectedLayers) and train the neural network using the training data from MNIST data set. Choose appropriate hyper parameters for the training of the neural network. To obtain full marks, your network should be able to achieve a test accuracy of 90% or more across many different random seeds. Ensure that code is optimized and does not take more than 10 mins to arrive at desired accuracy. We will run your code in SL2 machines, ensure that time limit is not exceeded else some penalty will be applied. Only for MNIST and XOR.

Complete `taskMnist()` in `tasks.py`. To test your backpropagation code on this data set, run the following command.

```
python3 test.py 3 seedValue
```

Here `seedValue` is a suitable integer value to initialise the seed for random-generator.

Task 2.4: Compute Accuracy over CIFAR (using Convolution Neural Network) (5 marks)

The CIFAR-10 data set is given in `cifar-10` directory. The data is read using the `readCIFAR10()` method implemented in `util.py` file.

Your are required to instantiate a NeuralNetwork class object (with at least one convolutional layer) and train the neural network using the training data from CIFAR data. You are required to choose appropriate hyperparameters for the training of the neural network. To obtain full marks, your network should be able to achieve a test accuracy of 35% or more. Note that we require lower accuracies than is reported in the literature as CNNs take a lot of time to train. Moreover, we have added code to reduce the size of training, test and validation set (see `tasks.py`). You are free to choose the size of data to use for training, testing and validation as you see fit. For submission remember to save the weights is the file `model.npy` (look `nn.py` `train()` function for more details) for your trained model and also **uncomment line 90 in `tasks.py`**. Also list all your hyperparameters and the random seed used (seedValue) for training on this task (in a file named `observations.txt`), so we can reproduce your results exactly.
NOTE: You will require lot of time to train (at least 4-5 hrs) in order to get the above accuracies. **So start working on this assignment as early as possible, leaving 1-2 days for training alone.**

Complete `taskCifar10()` in `tasks.py`. To test your backpropagation code on this data set, run the following command.

```
python3 test.py 4 seedValue
```

Task 2.5: Describe the results (2 marks)

In this task, you are required to report hyperparameters (learning rate, number of hidden layers, number of nodes in each hidden layer, batchsize and number of epochs) of Neural Network for each of the above 4 tasks that lead to minimal-topology neural networks that pass the test: that is, with the minimal number of nodes you needed for each task. For example, in case of a linearly separable data set, you should be able to get very high accuracy with just a single hidden node. Write your observations in `rollno.pdf`. Remember to write your name in the pdf file.

Submission

You are expected to work on this assignment by yourself. You may not consult with your classmates or anybody else about their solutions. You are also not to look at solutions to this assignment or related ones on the Internet. You are allowed to use resources on the Internet for programming (say to understand a particular command or a data structure), and also to understand concepts (so a Wikipedia page or someone's lecture notes or a textbook can certainly be consulted).

In `tasks.py`, you will complete Tasks 2.1, 2.2, 2.3 and 2.4 by editing corresponding snippets of code.
In `rollno.pdf`, you will report the parameters (neural net architecture, learning rate, batch size etc., and also the random seed, for tasks 2.1, 2.2, 2.3 and 2.4. Also add relevant explanations to justify the topology being minimal.
Also, submit the trained model file for CIFAR-10 Task 2.4 Model in **`model.npy`**.

Remember to test your solution before submission using the autograder provided.

```
python3 autograder.py -t task_num
```

where `task_num` can be one of {1,2}.

Put all .py files in a directory named as your roll_no, and mention any changes in rollno.pdf file. Compress file using only zip command and name it as **rollno.zip** and upload it in moodle under Lab Assignment 5.