HW Assignment 1 (Due by 1:30pm on Feb 6)

1 Implementation (100 points)

Implement two versions of the softmax regression model in Python, using (1) NumPy and (2) PyTorch, and evaluate them on two 2D non-linear classification tasks: `flower` and `spiral`. Starter code and functions for generating the datasets are available at `http://ace.cs.ohio.edu/~razvan/courses/dl6890/hw01.zip`. The provided code also displays and saves images of the datasets and the trained model's decision boundaries. Make sure that you organize your code in folders as shown in the table below. Write code only in the Python files indicated in bold.

- dl6890/
  - hw01/
    - `code/`
      - numpy/
        - **softmax.py**
        - **computeNumericalGradient.py**
        - **output.txt**
        - softmaxExercise.py
        - utils.py
        - `flower-data.jpg`
        - `flower-boundary.jpg`
        - `spiral-data.jpg`
        - `spiral-boundary.jpg`
      - pytorch/
        - **softmaxExercise.py**
        - **output.txt**
        - utils.py
        - `flower-data.jpg`
        - `flower-boundary.jpg`
        - `spiral-data.jpg`
        - `spiral-boundary.jpg`

1.1 NumPy Implementation (50 points)

Coding effort: my implementation has 11 lines of code in softmax.py and 8 lines of code in computeNumericalGradient.py.

1. **Cost & Gradient:** You will need to write code for two functions in sofmax.py:
   (a) The `softmaxCost()` function, which computes the cost and the gradient.
   (b) The `softmaxPredict()` function, which computes the softmax predictions on the input data.

The cost and gradient should be computed according to the formulas shown on the slides in Lecture 2, modified however to represent explicitly the bias and its gradient. Thus, if there are D features and K classes, the softmax model will be comprised of two types of parameters: **W** will be a K x D matrix of the feature weights, whereas **b** will be a K x 1 vector of bias terms.

2. **Vectorization:** It is important to vectorize your code so that it runs quickly.

3. **Ground truth:** The `groundTruth` is a matrix M such that M[c, n] = 1 if sample n has label c, and 0 otherwise. This can be done quickly, without a loop, using the SciPy function sparse.coo_matrix(). Specifically, `coo_matrix((data, (i, j)))` constructs a matrix A such that A[i[k], j[k]] = data[k], where the shape is inferred from the index arrays. Sample code for computing the ground truth matrix has been provided in Lecture 2.

4. **Overflow:** Make sure that you prevent overflow when computing the softmax probabilities, as shown on the slides in Lecture 2.

5. **Numerical gradient:** Once you implemented the cost and the gradient in `softmaxCost`, implement code for computing the gradient numerically in computeNumericalGradient.py, as shown on the slides in Lecture 3.

6. **Gradient checking:** Use computeNumericalGradient.py to make sure that your softmaxCost.py is computing gradients correctly. This is done by running the main program in Debug mode, i.e. `python3 softmaxExercise.py --debug`.

   In general, whenever implementing a learning algorithm, you should always check your gradients numerically before proceeding to train the model. The norm of the difference between the numerical gradient and your analytical gradient should be small, on the order of 10^{-9}.

7. **Training:** Training your softmax regression is done using gradient descent for 200 epochs.

8. **Testing:** Now that you've trained your model, you will test it against the training set to determine how well the softmax can fit this dataset. To do so, you will first need to complete the function `softmaxPredict()` in softmax.py, a function which generates predictions for input data under a trained softmax model. Once that is done, you will be able to compute the accuracy of your model using the code provided.

1.2 PyTorch Implementation (50 points)

Coding effort: my implementation has 14 lines of code in softmaxExercise.py.

You will need to write code for the following:

1. **Tensors:** Create pytorch tensors for the input data and the model parameters. Specify that gradients are to be computed w.r.t. parameters. Initialize the bias vector with zeros, and the weight matrix with a standard Gaussian multiplied with 0.01.

2. **Loss:** Write code that computes the loss variable, based on the current values of the parameters. Once the loss is computed, the gradient w.r.t. the parameters will be automatically computed by calling `loss.backward()`. You are supposed to write the code for computing the loss yourself. In particular, do not use functions from PyTorch (e.g. from the `torch.nn` module) that compute the cross entropy loss.

3. **Predictions:** Use the trained softmax model to compute labels for the training examples.

2 Bonus (50 points)

Modify the data generation functions to create examples that have only two labels and write a second version of the assignment that implements logistic regression for binary classification.

3 Submission

Electronically submit on Blackboard a hw01.zip file that contains the hw01 folder in which you write code **only in the required files**. The screen output produced when running the softmaxExercise.py code should be redirected to (saved into) the **output.txt** files.

On a Linux system, creating the archive can be done using the command:

```
> zip -r hw01.zip hw01.
```

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. Make sure your code runs correctly when used in the directory structure shown above.