HW Assignment 4 (Due by 10:30am on Oct 24)

1 Implementation (150 points)

Implement two versions of the softmax regression model in Python, using (1) SciPy and (2) scikit-learn, and evaluate them on the MNIST digit recognition task. Starter code and the MNIST dataset are available at `http://ace.cs.ohio.edu/~razvan/courses/ml4900/hw04.zip`. 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.

```
ml4900/
  hw03/
    code/
      scipy/
        sofmax.py
        computeNumericalGradient.py
        output.txt
        softmaxExercise.py
        checkNumericalGradient.py
      scikit/
        sofmaxExercise.py
        output.txt
    data/
      mnist/
```

1.1 SciPy Implementation (100 points)

Coding effort: my implementation has 10 lines of code in softmax.py and 7 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 4.

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. The code for cumputing the ground truth matrix has been provided for you.

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

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 4. Code is provided in checkNumericalGradient.py for you to test your numerical gradient implementation.

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`. When debugging, you can speed up gradient checking by reducing the number of parameters your model uses. In this case, the code reduces the size of the input data, using the first 8 pixels of the images instead of the full 28x28 image.

   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 L-BFGS for 100 epochs, through the SciPy function scipy.optimize.fmin_l_bfgs_b(). Training the model on the entire MNIST training set of 60,000 28x28 images should be rather quick, and take less than 5 minutes for 100 iterations.

8. **Testing:** Now that you've trained your model, you will test it against the MNIST test set, comprising 10,000 28x28 images. However, 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. My implementation achieved an accuracy of 92.6%. If your model's accuracy is significantly less (less than 91%), check your code, ensure that you are using the trained weights, and that you are training your model on the full 60,000 training images.

1.2 Scikit Implementation (50 points)

Coding effort: my implementation has 4 lines of code in softmaxExercise.py.

You will need to write code for the following 3 functionalities:

1. **C parameter:** As explined in homework 3, `scikit`'s objective function expresses the trade-off between training error and model complexity through a parameter C that is multiplied with the erorr term. Compute the C parameter such that the objective is equivalent with the standard formulation used in `scipy` that multiplies the regularization parameter (called 'decay' in the code) with the L2 norm term.

2. **Softmax training:** Train a softmax regression model using the 'multinomial' option for multiclass classification, and the C parameter computed above. Specify training with the L-BFGS solver for 100 max iterations. For this, you will instantiate the class `linear_model.LinearRegression`.

3. **Softmax testing:** Use the trained softmax model to compute labels on the test images.

The code also computes and prints the accuracy on the test images.

2 Bonus (25 points)

Create and evaluate a new version of the SciPy code that trains the softmax regression model using minibatch gradient descent for 2000 updates, where the size of a minibatch is 100.

3 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw04.zip file that contains the hw04 folder in which you write code **only in the 3 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 hw04.zip hw04.
```

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.