HW Assignment 1 (Due by 9:00am on Jan 31)

1 Theory (80 points)

1. [Linear Regression, 20 points]
   Prove that the least square solution to polynomial curve fitting satisfies the set of $M+1$ linear equations shown on slide 19 in lecture 1.

2. [Softmax Regression, 20 points]
   Consider a training set that contains the 4 examples below i.e., the truth table of the logical XOR function. Prove that no softmax regression model can perfectly classify this dataset. Do not forget the bias feature $x_0 = 1$.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

3. [Logistic Regression, 20 points]
   Prove that the gradient (with respect to $w$) of the negative log-likelihood error function for logistic regression corresponds to the formula shown on slide 54 from lecture 1:

$$\nabla_w E(w) = \sum_{n=1}^{N} (h_n - t_n)x_n^T$$  \hspace{1cm} (1)

4. [Softmax Regression, 20 points]
   Prove that the gradient (with respect to $w_k$) of the negative log-likelihood error function for regularized softmax regression corresponds to the formula shown on slide 62 from lecture 1, for any class $k \in [1..K]$:

$$\nabla_{w_k} E(w) = -\frac{1}{N} \sum_{n=1}^{N} (\delta_k(t_n) - p(C_k|x_n))x_n^T + \alpha w_k^T$$  \hspace{1cm} (2)

2 Implementation (150 points)

Implement two versions of the softmax regression model in Python, using (1) SciPY and (2) TensorFlow, and evaluate them on the MNIST digit recognition task. Starter code is provided in [hw01.zip] whereas the MNIST dataset is available in [mnist.zip] both on the course webpage.

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. You should write your code only in the places indicated by “Write Your Code Here” in the files.
2.1 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 softmax.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 slide 64 in [Lecture 1](lecture1).

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 computing 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 slide 65 in [Lecture 1](lecture1).

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 slides 66-67 in [Lecture 1](lecture1). 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 92.5%. 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.

### 2.2 TensorFlow Implementation (50 points)

*Coding effort: my implementation has 11 lines of code in softmax.py.*

You will need to write code for three functions in softmax.py:

1. **Computation graph**: The `inference()` function, which creates the weights and biases variables and returns the output tensor with the computed logits.

2. **Cost**: The `loss()` function, which returns a tensor that computes the cross entropy loss.

3. **Testing**: The `evaluation()` function, which computes the number of input examples that were predicted correctly by the input model.

The **MNIST For ML Beginners** tutorial shows a bare bone implementation of softmax regression in TensorFlow. The implementation in this assignments has additional code that uses:

1. A *feed dictionary* that maps images and labels *placeholders* to values.

2. A *summary writer* to output summaries and the computation graph. Summaries can be seen later in TensorBoard.

3. A *saver* for writing training checkpoints on disk every 1,000 mini-batches.

The code also computes and prints the accuracy on the training, validation, and test data every 1,000 mini-batches.
3 Bonus (25 points)

The TensorFlow implementation trains the softmax regression model using minibatch gradient descent for 2000 updates, where the size of a minibatch is 100. Create and evaluate a new version of the SciPy code that uses the same minibatch gradient descent training scenario instead of batch L-BFGS optimization.

4 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw01.zip file that contains the hw01 folder in which you change 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 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. On the theory assignment, clear and complete explanations and proofs of your results are as important as getting the right answer.

4. Make sure your code runs correctly when used in the directory structure shown above.