Softmax Regression

Implement the softmax regression model in Python, using (1) NumPy, (2) PyTorch, and (3) SKLEARN and evaluate them on two 2D non-linear classification tasks: flower and spiral. The provided starter code also displays and saves images of the datasets and the trained model’s decision boundaries.

0.1 NumPy Implementation

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 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 the slides, 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 `scipy.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 on the slides.

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

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.

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.

### 0.2 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. **Variables:** Create Pytorch variables for the input data and the model parameters. Specify that gradients are to be computed w.r.t. parameters only. 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.

### 1 Bonus

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.