ROB313: Introduction to Learning from Data
University of Toronto Institute for Aerospace Studies

Assignment 3 (13 pts)

Due March 26, 2021, 23:59 EST

Q1) 6pts Use gradient descent to learn the weights of a logistic regression model. Logistic regression is used for classification problems (i.e. $y^{(i)} \in \{0,1\}$ in the binary case which we will consider) and uses the Bernoulli likelihood

$$\Pr(y|\mathbf{w},\mathbf{x}) = \big[\widehat{f}(\mathbf{x};\mathbf{w})\big]^y \big[1-\widehat{f}(\mathbf{x};\mathbf{w})\big]^{1-y},$$

where $\widehat{f}(\mathbf{x};\mathbf{w}) = \Pr(y{=}1|\mathbf{w},\mathbf{x})$ gives the class conditional probability of class 1 by mapping $\mathbb{R}^D \to [0,1]$. To ensure that the model gives a valid probability in the range [0,1], we write $\widehat{f}$ as a logistic sigmoid acting on a linear model as follows

$$\widehat{f}(\mathbf{x};\mathbf{w}) = \mathsf{sigmoid}\Big(w_0 + \sum_{i=1}^{D} w_i x_i\Big),$$

where $\mathsf{sigmoid}(z) = \frac{1}{1+\exp(-z)}$, and $\mathbf{w} = \{w_0, w_1, \ldots, w_D\} \in \mathbb{R}^{D+1}$. Making the assumption that all training examples are *i.i.d.*, the log-likelihood function can be written as follows for the logistic regression model

$$\log \Pr(\mathbf{y}|\mathbf{w},\mathbf{X}) = \sum_{i=1}^{N} y^{(i)} \log\big(\widehat{f}(\mathbf{x}^{(i)};\mathbf{w})\big) + \big(1-y^{(i)}\big)\log\big(1-\widehat{f}(\mathbf{x}^{(i)};\mathbf{w})\big).$$

a) 1pts What will be the value of the log-likelihood if $\widehat{f}(\mathbf{x}^{(i)};\mathbf{w}) = 1$, but the correct label is $y^{(i)} = 0$ for some i? Is this reasonable behaviour?

b) 2pts Consider maximum *a posteriori* (MAP) estimation using the prior

$$\Pr(\mathbf{w}) = \mathcal{N}\big(\mathbf{w}\big|\mathbf{0},\ \sigma^2\mathbf{I}\big).$$

Write the steepest descent update rule for the parameters $\mathbf{w}$ at each iteration of both full-batch gradient descent (GD) and stochastic gradient descent (SGD) with a mini-batch size of 1. Assume a constant learning rate $\eta > 0$. The exact gradient of the log-likelihood function with respect to the parameters can be written as follows

$$\nabla_{\mathbf{w}} \log \Pr(\mathbf{y}|\mathbf{w},\mathbf{X}) = \sum_{i=1}^{N} \Big(y^{(i)} - \widehat{f}(\mathbf{x}^{(i)};\mathbf{w})\Big) \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_D^{(i)} \end{bmatrix},$$

where we used the convenient form of the derivative of the sigmoid function $\frac{\partial}{\partial z}\mathsf{sigmoid}(z) = \mathsf{sigmoid}(z)\big(1-\mathsf{sigmoid}(z)\big)$.

c) 3pts Initializing $\mathbf{w} = \mathbf{0}$, find the MAP estimate of the parameters using both full-batch gradient descent (GD), as well as stochastic gradient descent (SGD) with a mini-batch size of 1. Consider the prior variance $\sigma^2 = 1$. Analyze the convergence trends of both optimization methods by plotting the loss versus epoch and report the learning rates used in each case.

Train the logistic regression model on the `iris` dataset, considering only the second response to determine whether the flower is an *iris versicolour*, or not[1]. Use both the training and validation sets to predict on the test set, and present test accuracy as well as the test log-likelihood. Why might the test log-likelihood be a preferable performance metric?

Q2) 7pts In the previous question we computed gradients manually for a linear logistic regression problem. This question will consider training a more complicated model, a deep neural network, and to help us compute gradients we will use the automatic differentiation package `autograd`. To install `autograd`, run the following in a terminal (mac or linux), or Anaconda prompt (windows)

```
conda install -c conda-forge autograd
```

The ipython notebook used for the in-class autograd tutorial can be found on portal.

In this assignment you will train a fully connected neural network with two hidden layers on the `MNIST_small` dataset using a categorical (generalized Bernoulli) likelihood. Using a mini-batch[2] size of 250, train the weights and bias parameters of the neural network using stochastic gradient descent. Initialize the biases of the model to zero and initialize the weights randomly.

You are provided the python module `a3_mod.py` which can be found on portal. A brief description of each function is provided here but more details can be found by reviewing the docstrings and inline comments.

- `a3_mod.forward_pass` computes the forward pass of a two layer neural network. The output layer activation function will need to be modified in this assignment.
- `a3_mod.negative_log_likelihood` computes the negative log-likelihood of the neural network defined in `a3_mod.forward_pass`. This function will need to be modified in this assignment.
- `a3_mod.nll_gradients` returns the negative log-likelihood computed by `a3_mod.negative_log_likelihood`, as well as the gradients of this value with respect to all weights and biases in the neural network. You should not need to modify this function.
- `a3_mod.run_example` this function demonstrates the computation of the negative log-likelihood and its gradients. It is intended as an example to get you familiar with the code and you may modify this function any way you wish.

Before beginning this question, you are encouraged to review the python code for these functions which is short and well documented. Running (and modifying) the `a3_mod.run_example` function can also be helpful to understand the provided code.

[1]Use, `y_train, y_valid, y_test = y_train[:,1,None], y_valid[:,1,None], y_test[:,1,None]`

[2]At no point in the assignment should you need to perform an operation (e.g. a forward pass) with the full training batch. If you do this then you are doing something wrong and should re-read carefully.

a) 2pts Since we plan to maximize the log-likelihood using a categorical likelihood, we would like our neural network to have 10 outputs, each a class-conditional log probability for each of the 10 classes for the `mnist_small` dataset. The two hidden layer neural network defined in `a3_mod.forward_pass` initially has a linear activation function on the output layer, however, these outputs do not define valid class-conditional log probabilities. Modify `a3_mod.forward_pass` so that a log-softmax activation function is used on the output layer. For your implementation, use only (autograd wrapped) numpy functions, do not use any loops, and ensure that your implementation is numerically stable. Briefly describe your implementation and why it is numerically stable. Hint: consider the LogSumExp trick we covered in class.

b) 2pts The function `a3_mod.negative_log_likelihood` currently assumes a Gaussian likelihood, however, we would like to use a categorical likelihood. Modify this function such that the negative log-likelihood is returned for a mini-batch of inputs assuming that the outputs of `a3_mod.forward_pass` are class conditional log probabilities. For your implementation, use only (autograd wrapped) numpy functions, and do not use any loops.

c) 2pts Considering 100 neurons per hidden layer, plot the stochastic estimate of the training set negative log-likelihood (using a mini-batch size of 250) versus iteration number during training. In the same plot, also draw the validation set negative log-likelihood versus iteration number. How does the network's performance differ on the training set versus the validation set during learning? Also, report the test set negative log-likelihood and test set accuracy for the final model.

d) 1pts Plot a few test set digits where the neural network is not confident of the classification output (i.e. the top class conditional probability is below some threshold), and comment on them. You may find `data_utils.plot_digit` helpful.

Submission guidelines: Submit an **electronic copy** of your report in **pdf** format, and **documented** python scripts. You should include a file named "README" outlining how the scripts should be run. Upload a single `tar` or `zip` file containing all files to Quercus. You are expected to verify the integrity of your `tar/zip` file before uploading. Do not include (or modify) the supplied `*.npz` data files or the `data_utils.py` module in your submission. The report must contain

- Objectives of the assignment
- A brief description of the structure of your code, and strategies employed
- Relevant figures, tables, and discussion

Do not use scikit-learn for this assignment, the intention is that you implement the simple algorithms required from scratch. Also, for reproducibility, always set a seed for any random number generator used in your code. For example, you can set the seed in numpy using `numpy.random.seed`