Assignment 3 (12.5 pts)

Due March 18, 2023, 23:59

Q1) 7.5pts Use four variants of gradient descent to learn the weights of a linear regression model via minimization of the least-squares loss function. Train your model on the `pumadyn32nm` dataset, using only the first 1000 points in the training set to predict on the test set[1], and present the final test RMSE. The four methods are:

- **2.5pts** Full-batch gradient descent,
- **1.5pts** Stochastic gradient descent (SGD) with mini-batch size 1,
- **1.5pts** Stochastic gradient descent (SGD) with mini-batch size 10,
- **2pts** Stochastic gradient descent (SGD) with mini-batch size 1 and momentum.

Recall that momentum in SGD means that the gradient approximation $\mathbf{g}$ at the kth iteration is a weighted sum of the new gradient estimate and the gradient approximation at the $k-1$th iteration,

$$\mathbf{g}_k = \beta \mathbf{g}_{k-1} + (1-\beta)\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}_k; \mathcal{D}),$$

where β is a new momentum hyperparameter. For each method, initialize all weights to zero and plot the exact (full-batch) loss versus iteration number considering a range of learning rates. Also, indicate the value of the exact optimum on all the plots (recall that the exact minimizer for this model structure was found in assignment 1 using the SVD). Comment on the convergence trends; compare learning rates that are too small or too large, and compare the convergence of each method over epoch number and over computation time. Select and report a good learning rate for each method (and β for SGD+momentum). How do the three variants of SGD compare?

Q2) 5pts 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

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

[1]For `pumadyn32nm`, use `x_train, y_train = x_train[:1000], y_train[:1000]`

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

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

Initializing all weights to zero, find the maximum-likelihood 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. For each method, plot the exact (full-batch) negative log-likelihood versus iteration number considering a range of learning rates. The gradient of the log-likelihood function with respect to the weights can be written as follows

$$\nabla \log \Pr(\mathbf{y}|\mathbf{w}, \mathbf{X}) = \sum_{i=1}^{N} \left(y^{(i)} - \widehat{f}(\mathbf{x}^{(i)}; \mathbf{w})\right)\left\{1, x_1^{(i)}, \ldots, x_D^{(i)}\right\}^T,$$

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

Train a logistic regression model on the `iris` dataset, considering only the second response to determine whether the flower is an *iris virginica*, or not[2]. 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?

Submission guidelines: Submit an **electronic copy** of your report (**maximum 10 pages** in at least 10pt font) in **pdf** format and **documented** Python scripts. You should include a file named "README" outlining how the scripts should be run. Upload both your report in pdf format and a single tar or zip file containing your code and README 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`.

[2]Use, `y_train, y_valid, y_test = y_train[:,(1,)], y_valid[:,(1,)], y_test[:,(1,)]`