CSCI 6962/4140: Homework 1

Assigned Thursday September 12, 2023. Due by 11:59pm ET Thursday September 26, 2023.

Objective of this assignment Comfort with computing and manipulating gradients and Hessians is essential in optimization and ML. In this assignment you will practice computing these for the problem of binary and multinomial logistic regression, and verify your results using PyTorch's autograd functionality. In the future you will implement your own solver for logistic regression using these expressions. For now, you will use a solver built into scikit-learn to fit a multinomial logistic regression model on the Fashion MNIST data set. Additionally, you will show that using the Bernoulli noise model and the Categorical noise model for binary classification lead to essentially the same models.

Instructions Create a Jupyter notebook for this assignment, and use Python 3. Write documented, readable and clear code (e.g. use reasonable variable names). Your code should run on the TA's computer, but do not expect them to run it to produce the output; instead, submit it with the outputs shown. Submit this notebook interspersing any textual answers in markdown cells (using LaTeX), clearly labeled, along with your code.

1. [5 pts] Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be a collection of n training data points for a binary classification problem, where the covariate vectors are in $\mathbb{R}^d$ and the labels are in ± 1.

   One can learn a linear classifier for binary classification by using the Bernoulli noise model and solving the MLE problem

   $$\operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-y_i \boldsymbol{\theta}^\top \mathbf{x}_i)\right). \tag{LL}$$

   The loss function used here is the logistic loss, $\ell(u, v) = \log(1 + \mathrm{e}^{-uv})$.

   Another alternative approach is to represent the targets using one-hot encoding, meaning write $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, where $\mathbf{y}_i \in \mathbb{R}^2$ is a one-hot encoding vector for the class of the ith training example, so $\mathbf{y}_i = [1; 0]$ if the ith example has label $+1$, and $\mathbf{y}_i = [0; 1]$ if it instead has label -1.

   Using the Categorical noise model to formulate the two-class classification problem then leads to the MLE problem

   $$\operatorname{argmin}_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{n}\sum_{i=1}^{n} -\ln(\mathbf{y}_i^\top \operatorname{softmax}(\boldsymbol{\Theta}\mathbf{x}_i)) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{H}(\mathbf{y}_i, \operatorname{softmax}(\boldsymbol{\Theta}\mathbf{x}_i)). \tag{CE}$$

   Here, the parameter matrix $\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\theta}_1^\top \\ \boldsymbol{\theta}_2^\top \end{bmatrix} \in \mathbb{R}^{2\times d}$, and the loss function is the cross-entropy, $H(\mathbf{p}, \mathbf{q}) = -p_1 \ln(q_1) - p_2 \ln(q_2)$.

   It is common to see either formulation of the binary classification problem, so it is worth asking: how do they compare?

   To answer this question, do the following.

   - Show that $\mathrm{H}(\mathbf{y}_i, \operatorname{softmax}(\boldsymbol{\Theta}\mathbf{x}_i))$ can be expressed using the logistic loss function, the difference vector $\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2$, y_i, and $\mathbf{x}_i$. Argue from this that any minimizer to (CE) can be converted to a minimizer of (LL) in $\mathcal{O}(d)$ operations.
   - Given a minimizer to (LL), explain how to construct a minimizer of (CE) in $\mathcal{O}(d)$.

- Now that we see that the two formulations are equivalent, which optimization problem do you think is more convenient to work with computationally, and why? (The point of this question is to get you thinking about what factors might matter in making such a choice).

2. [20 pts] Compute the gradient and Hessians of the loss function for ℓ_2-regularized logistic regression, given as

$$f(\boldsymbol{\omega}') = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-y_i(\omega_0 + \boldsymbol{\omega}^\top \mathbf{x}_i))\right) + \lambda\|\boldsymbol{\omega}\|_2^2. \tag{1}$$

Note that the bias term is not regularized. Here, $\boldsymbol{\omega}' = [\omega_0; \boldsymbol{\omega}]$ is the vertical concatenation of the bias and the feature coefficients, for convenience. Write $\nabla_{\boldsymbol{\omega}'} f$ and $\nabla^2_{\boldsymbol{\omega}'} f$ as concisely as you can.

3. [10pts] We will use PyTorch in the second portion of the course to train neural networks, precisely because it can automatically compute the gradient of complicated functions. Work through the first three portions of the PyTorch introduction at pytorch.org: "Introduction to PyTorch", "Introduction to PyTorch Tensors", and "The Fundamentals of Autograd". Now that you understand how to use autograd, use it to empirically verify the correctness of your expression above using PyTorch. Use $\lambda = 1$.

   - Use the `load_breast_cancer` convenience function of scikit-learn to load the UCI Breast Cancer Wisconsin data set. Store the features and targets in tensors $\mathbf{X}$ and $\mathbf{y}$, respectively. Convert the labels to $-1, 1$.
   - Initialize the parameter tensor $\boldsymbol{\omega}$ as a vector with all entries equal to $1/d^3$, and the bias tensor $\omega_0 = 1$.
   - Use a single expression to create a tensor f that evaluates the objective for (1), using the tensors $\mathbf{X}$ and $\mathbf{y}$.
   - Use PyTorch's autograd feature to compute the gradients of f with respect to the parameters $\boldsymbol{\omega}$ and ω_0.
   - In a single expression, evaluate your formula for $\nabla_{\boldsymbol{\omega}} f$ and store it to `myOmegaGrad`. Similarly, use one expression to evaluate your formula for $df/d\omega_0$ and store it to `myBiasDeriv`.
   - Compute and display (labeled as such) the Euclidean norm difference between your $\nabla_{\boldsymbol{\omega}} f$ and the value computed by PyTorch.
   - Compute and display (labeled as such) the absolute difference between your $df/d\omega_0$ and the value computed by PyTorch.

4. [60 pts] Try multinomial logistic regression on the Fashion-MNIST data set.

   - Download the Fashion-MNIST data set at `https://github.com/zalandoresearch/fashion-mnist` and create your Jupyter notebook in the base directory of this repo so that the relative paths will be consistent for the TA.
   - Load the training and testing splits of the data set according to the instructions for loading the Python given in the README; use the same variable names.
   - Preprocess the training data so that all 784 features (pixel values) look essentially like standard Gaussians (this helps with accuracy and convergence). Do this by fitting an sklearn `StandardScaler` on the training data and applying it to the training and test data sets; see `https://scikit-learn.org/stable/modules/preprocessing.html`. Overwrite the original training and test data sets with these processed data sets.
   - Use Matplotlib and the `helper.get_sprite_image` function from the Fashion-MNIST repo to display one of each of the ten image classes in the training data set. Show them in a 2×5 grid.

- Use sklearn to fit a multiclass logistic regression model to predict the image labels, using the SAGA solver (you may need to increase the number of maximum iterations and/or decrease the convergence tolerance — be reasonable). Report the top-1 and top-3 classification accuracies and confusion matrices on the test and train data sets: see `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.top_k_accuracy_score.html` and `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`.
- What conclusions do you draw about the performance of the model on the various classes, given the confusion matrix on the test set?
- Select and display one of the misclassified images in the training data set: what class should it have been classified as, and what class was it misclassified as?

5. (CSCI 6962 students) [40 pts] Compute the gradients $\nabla_{\mathbf{W}} f$ and $\nabla_{\mathbf{b}} f$ of the loss function for ℓ_2-regularized multinomial logistic regression,

$$f(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^{n} \left[\mathbf{y}_i^\top (\mathbf{W}\mathbf{x}_i + \mathbf{b}) - \log \left(\mathbf{1}^T \exp(\mathbf{W}\mathbf{x}_i + \mathbf{b}) \right) \right] + \lambda \|\mathbf{W}\|_F^2. \tag{2}$$

   Recall that $\|\mathbf{W}\|_F^2$, the squared Frobenius norm of $\mathbf{W}$, is the sum of its squared entries: $\|\mathbf{W}\|_F^2 = \sum_{i,j=1}^{k,d} W_{i,j}^2$. Give as concise expressions for these gradients as you can. It will help to use the chain rule.

6. (CSCI 6962 students) [10pts] Empirically verify the correctness of your expression above using PyTorch. Use $\lambda = 1$.

   - Use the `load_wine` convenience function of scikit-learn to load the UCI wine classification dataset. Store the features and targets in tensors $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{Y} \in \mathbb{R}^{n \times k}$, respectively. Use `sklearn.preprocessing.OneHotEncoder` to do the one-hot encoding of the targets.
   - Use the appropriate PyTorch utility functions to initialize the parameter tensor $\mathbf{W}$ as a matrix of all ones scaled by $1/(kd)^3$, and the bias tensor $\mathbf{b}$ as a vector of all ones.
   - Use a single expression to create a tensor f that evaluates the objective for (2), using the tensors $\mathbf{X}$ and $\mathbf{Y}$.
   - Use PyTorch's autograd features to compute the gradients of f with respect to the parameters $\mathbf{W}$ and $\mathbf{b}$.
   - In a single expression, evaluate your formula for $\nabla_{\mathbf{W}} f$ and store it to `myWGrad`. Similarly, use one expression to evaluate your formula for $\nabla_{\mathbf{b}} f$ and store it to `myBGrad`.
   - Compute and display (labeled as such) the Frobenius norm difference between your $\nabla_{\mathbf{W}} f$ and the value computed by PyTorch.
   - Compute and display (labeled as such) the Euclidean norm difference between your $\nabla_{\mathbf{b}} f$ and the value computed by PyTorch.