ECE 685D HW2

Deliverability: Please upload your Jupyter Notebook file (.ipynb file) with all outputs and necessary derivations exported as a PDF or HTML to Sakai. All necessary coding documentation can be found in `https://pytorch.org/docs/stable/index.html`. **We highly encourage you to submit your derivation in LaTeX. If you choose to submit a hand-written derivation, please make sure your derivation is legible. If you have hand-written solutions, please scan/take a photo and insert them into their designated space.** You can follow this tutorial to insert images to .ipynb files: `https://www.geeksforgeeks.org/insert-image-in-a-jupyter-notebook/`

1 Problem 1: Simple MLP for regression

1.1 Back propagation (15 pts)

We define the function f as a k-layer fully connected neural network (MLP) that takes input vector x and outputs a continuous label y. g is a generic 1-Lipschitz activation function applied element-wise. (You may denote the gradient of g(x) w.r.t x as $\frac{\partial g(x)}{\partial x}$ wherever it is needed).

In detail, $f(x)$, *where* $x \in \mathbb{R}^{m\times 1}$ can be written as:

$$h_1 = g(W_1^T x + b_1) \tag{1}$$
$$h_2 = g(W_2^T h_1 + b_2) \tag{2}$$
$$\dots \tag{3}$$
$$f(x) = \hat{y} = W_k^T h_{k-1} + b_k \tag{4}$$

Let k=3 and the loss function $L(Y, \hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ be mean squared error (MSE), derive the gradients for weights $\frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial W_1}$ and bias $\frac{\partial L}{\partial b_3}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_1}$

1.2 Vanishing gradient problem (10 pts)

Assuming the MLP in 1.1 satisfies the condition that $0 < ||W_i||_2 < c$ and $c < 1$. Please show

$$\lim_{k\to\infty} \nabla_{h_1} L(\theta) = \mathbf{0}$$

Hint: $||.||_2$ in this problem means the L2 inductive matrix norm. You may find the sub-multiplicative property useful: $||AB||_2 \leq ||A||_2||B||_2$

2 Problem 2: MLP for Wine (15 pts)

Use PyTorch to implement the neural network structure described above, and let activation g be the ReLU activation. You will download the UCI Wine dataset from `https://archive.ics.uci.edu/ml/datasets/wine+quality` and regress on the "quality" variable. You may preprocess the data at your own discretion. Then, split your data into training, validation, and testing sets with a 64-16-20 split. Finally, train and validate your neural network (with at least 3 hidden layers) using a simple SGD optimizer with your customized batch size and learning rate.

Plot the training loss and validation loss over your training epoch and comment on your model fit. Also, comment on whether it makes sense to treat the wine quality as a continuous label.

3 Problem 3: Batch Size, Learning Rate and LR Scheduling (30 pts)

Repeat the above process and experiment with two optimizers of your choice that are covered in the lectures. Explore the optimization with learning rate decay $\gamma \in \{0.1, 0.001, 0.0001\}$ using a StepLR scheduler with step size=30. You will also set learning rate $\eta \in \{10^{-2}, 10^{-3}, 10^{-4}\}$ and batch size $N \in \{16, 160, 1600\}$. Feel free to pick any maximum iteration number that is reasonable given your computational resources.

Plot the MSE on the training and validation set as the training proceeds for each triplet of (γ, η, N). Then, report the final testing MSE. Comment on which combination of optimization parameters is the best (show evidence through plots or tables). Also, comment on whether your model is overfitting, underfitting, or just right.

In [1], Goyal et al. proposed a linear scaling rule mentioning that one should consider "When the minibatch size is multiplied by k, multiply the learning rate by k." Do you think this statement is useful for the Wine dataset based on your experiment result?

4 Problem 4: More on optimizers (30 pts)

In this problem, we want to fit a multinomial Logistic Regression (LR) model to the MNIST dataset using cross-entropy loss with L2-regularization.

Let K be the number of classes (for MNIST K = 10) and let $D = \{x_n, y_n\}$ denote the dataset where $x_n \in \mathbb{R}^{784}$ *and* y_n is the label of the n-th data point. Write a Python program that fits the model using **all of** the following optimization methods:

1. Momentum method with parameter $\beta = 0.9$ (5 pts)
2. Nesterov's Accelerated Gradient (NAG) with parameter $\beta = 0.95$ (5 pts)
3. RMSprop with parameters $\beta = 0.95, \gamma = 1$ *and* $\epsilon = 10^{-8}$ (10 pts)
4. Adam with parameters $\beta_1 = 0.9, \beta_2 = 0.999,$ *and* $\epsilon = 10^{-8}$ (10 pts)

Note: You can use the Autograd package from Pytorch to compute the gradient when building your optimizer. However, you are NOT allowed to use any built-in optimizers.
Hint: Kingma et al. stated in [2] an alternative implementation of Adam, which has lower clarity but higher computation efficiency.

5 (Bonus) Problem 5: Logistic Regression using Newton's Method (20 pts)

In this problem, we are going to fit a Logistic Regression model on the Breast Cancer dataset: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html` using Newton's method. Since it's a binary classification problem, we will use the BCELoss from PyTorch. Specifically,

$$L(D; \mathbf{w}) = \sum_{n=1}^{N} -y_n \log(\sigma(\mathbf{w}^T \mathbf{x}_n)) - (1-y_n) \log(1-\sigma(\mathbf{w}^T \mathbf{x}_n)), \text{ where } \sigma(a) = \frac{1}{1+\exp(-a)}, D = \{x_n, y_n\}$$

For this problem, you will compute the gradient $\frac{\partial L(D; \mathbf{w})}{\partial \mathbf{w}}$ and the hessian $\frac{\partial^2 L(D; \mathbf{w})}{\partial w_i \partial w_j}$ of the model weights.

Then, using the gradients and hessian computed above, write a Python program that implements Newton's method to fit the Logistic Regression model on the Breast Cancer dataset implementing

1. Exact expression of the Hessian
2. Diagonal approximation of Hessian

Plot and compare the loss and accuracy graphs on the test set (learning rate = 0.1, test size = 0.3) for both methods. Setting the random seed allows for better reproducibility.

Set PyTorch seed: `https://pytorch.org/docs/stable/generated/torch.manual_seed.html#torch.manual_seed`
Set Numpy seed: `https://numpy.org/doc/stable/reference/random/generated/numpy.random.seed.html`.

References

[1] Priya Goyal et al. "Accurate, large minibatch sgd: Training imagenet in 1 hour". In: *arXiv preprint arXiv:1706.02677* (2017).

[2] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).