HOMEWORK SET 3

CSCI 5980 Think Deep Learning (Spring 2020)

Due 11:59 pm, May 02 2020

Instruction Please typeset your homework in LaTeX and submit it as a single PDF file in Canvas. No late submission will be accepted. For each problem, your should acknowledge your collaborators if any. For problems containing multiple subproblems, there are often close logic connections between the subproblems. So always remember to build on previous ones, rather than work from scratch.

Notation We will use small letters (e.g., u) for scalars, small boldface letters (e.g., $\boldsymbol{a}$) for vectors, and capital boldface letters (e.g., $\boldsymbol{A}$) for matrices. For a matrix $\boldsymbol{A}$, $\boldsymbol{a}^i$ (supscripting) means its i-th row as a *row vector*, and $\boldsymbol{a}_j$ (subscripting) means the j-the column as a column vector, and a_{ij} means its (i, j)-th element. $\mathbb{R}$ is the set of real numbers. $\mathbb{R}^n$ is the space of n-dimensional real vectors, and similarly $\mathbb{R}^{m \times n}$ is the space of $m \times n$ real matrices. The dotted equal sign $\doteq$ means defining.

Problem 1 (Stochastic optimization methods for MNIST digital recognition) MNIST is a handwritten digit recognition dataset used for benchmarking many machine learning techniques. In case you haven't heard of it, you can check out the information here `http://yann.lecun.com/exdb/mnist/`. In this problem, we're going to train a shallow neural network based on different stochastic gradient descent (SGD) methods that we learned in the lecture.

Save your codes into a single Jupyter notebook for submission. Although we will point to PyTorch resources, you can also use Tensorflow counterparts but you need to figure things out yourself.

Read this tutorial `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py` and learn how to set up the training and test datasets, specify the DNN model, and perform training and evaluation.

(a) Load MNIST training and test sets into your workspace (0.5/12)

(b) Build a 3-layer neural network model. You're free to choose the architecture, i.e., number of nodes, activation functions, convolutional layers if you're comfortable with. Also, choose an appropriate loss for your training objective. (1/12)

(c) Implement the Adagrad algorithm. You're free to choose your hyperparameters (initialization, batch size, learning rate, etc) and auto-differentiation. But you're not allowed to use the built-in Adagrad implementation. Please include a plot of how the objective vs. the epoch. (1.5/12)

(d) Implement the RMSprob algorithm. Requirement is the same as (c). (1.5/12)

(e) Implement the Adam algorithm. Requirement is the same as (c). (1.5/12)

(f) A "98%" test: MNIST is a relative easy classification task and the state-of-the-art learning models can achieve near perfect recognition performance. If you get a $\geq 98\%$ test accuracy for any one of (c), (d), and (e), you get 1 point here. A total of 2 points maximum will be awarded here. For this, you're free to adopt whatever strategy you think appropriate to avoid overfitting. (2/12)

Problem 2 (Batch normalization and Dropout) Save your codes into a single Jupyter notebook for submission. Although we will point to PyTorch resources, you can also use Tensorflow counterparts but you need to figure things out yourself.

(a) Recall that the key mapping in batch normalization is the vector-to-vector mapping $h : \boldsymbol{z} \mapsto \frac{\boldsymbol{z}-\mu_{\boldsymbol{z}}}{\sigma_{\boldsymbol{z}}}$, where $\mu_{\boldsymbol{z}}$ is the mean of elements in $\boldsymbol{z}$ and $\sigma_{\boldsymbol{z}}$ is the standard deviation. What's the Jacobian $\boldsymbol{J}_h\left(\boldsymbol{z}\right)$? (1.5/12)

(b) Grow your DNN in **Problem 1** into a 8-layer network. Use whatever optimization algorithm you have developed in Problem 1 (c), (d), and (e). Do you see any sign of overfitting? If so, try to plug in batch normalization (built-in function allowed) into some or all of the intermediate layers. Again you can use auto-differentiation, but remember you need to implement the running average of per-node statistics (mean and standard deviation) in order to apply them at the test time. Does it help or out? (1.5/12)

(c) Use the same 8-layer network as in (b) and implement Dropout (built-in function allowed) with a rate of $p = 0.5$. Again, use any of the optimization algorithm you have implemented above. How is the performance compared to that of (b) which is based on batch normalization? (1/12)