ECE590: Homework #2

Hai Li and Yiran Chen

ECE Department, Duke University — October 21, 2019

Objectives

Homework # 2 covers the contents of Lectures 11 $\sim$ 16, including the concepts, methods and practical problems of distributed training, efficient deployment and the compressing methods of DNNs.

Warning: You are asked to complete the assignment independently.

This homework has 100 points plus an extra credit of 10 points. You must submit your homework in PDF format through **Sakai** before 11:59pm, Wednesday November 6.

1 True/False Questions

For each question, please provide a short explanation to support your judgment.

Problem 1.1 (2 pts) To remove communication overhead in distributed training, one can apply 1-bit SGD to reduce communication cost without losing accuracy, as the communication cost is proportional to the bit length of the gradient.

Problem 1.2 (2 pts) Generally speaking, pruning does not hurt quantization, as they are orthogonal techniques in compress the size of DNN models.

Problem 1.3 (2 pts) One of the practical ways to achieve an efficient inference of a DNN model is to save potential multiplication results in a look-up table (LUT), as pruning greatly reduces the number of weight variables thus directly lead to limited weight values.

Problem 1.4 (2 pts) Asynchronous training is guaranteed to have linear speedup (e.g. the speedup ratio is proportional to the number of asychronous workers) as each worker can compute an independent part of the batch data in parallel, thanks to data parallelism.

Problem 1.5 (2 pts) Group Lasso can lead to structured sparsity on DNNs, which is more hardware-friendly. The idea of Group Lasso comes from applying L-2 regularization to the L-1 norm of all of the groups.

Problem 1.6 (2 pts) The Hoyer regularizer is similar to L-1 regularizer in that it is scale-invariant. The Hoyer-square regularizer shows an automatic 'trimming' effect to preserve larger weights while penalizing smaller ones. Therefore, it has a great approximation of the L-0 norm and thus it is beneficial for crafting sparse DNN models.

Problem 1.7 (2 pts) Proximal gradient descent introduces an additional proximity term to minimize regularization loss in the proximity of weight parameters. While the proximity term does not change the optimal solution, it allows smoother convergence of the overall objective.

Problem 1.8 (2 pts) In CUDA programming on GPUs, one can use for-loops with serial C-codes to generate explicit instructions for GPUs to do parallel computing.

Problem 1.9 (2 pts) Differential privacy is the defense against inference attack. To improve the differential privacy, one can add noise to the computed gradient to cover the difference under different training data.

Problem 1.10 (2 pts) The objective of trimmed L1 is equivalent to setting up an L-0 constraint on sparse minimization problem. This gives a much stronger guarantee on the sparsity than Lasso (i.e. L-1 regularizer).

2 Distributed training

For problem 2.1 - 2.5, consider the case where 3 workers trying to find a set of weights through a data parallel distributed learning process (as introduced on page 11 of Lecture 11). Worker i has one data point $X_i \in \mathbb{R}^{1\times 3}$ and a target $y_i \in \mathbb{R}$. The weight to be found is $W \in \mathbb{R}^{3\times 1}$ such that $X_i W = y_i$ holds for all workers. The goal of each worker will be minimizing the MSE loss $L_i = (X_i W - y_i)^2$.

Specifically, $X_1 = [-1, 2, 0]$ and $y_1 = -7$ for worker 1, $X_2 = [3, 0, 1]$ and $y_2 = 5$ for worker 2, and $X_3 = [2, -1, 3]$ and $y_3 = 11$ for worker 3.

You will need to use NumPy or MATLAB to finish this set of questions, please put all your code for this set of questions into one file and submit it on Sakai alone with your solutions.

Problem 2.1 (4 pts) Theoretical analysis: suppose the learning rate on the parameter server is λ, and the weight on the parameter server is W^k at step k of distributed learning, derive the symbolic formulation of weight W^{k+1} at step k+1 with x_i, y_i, $i \in \{1, 2, 3\}$. Then derive the closed-form symbolic representation of the optimal solution W^* with x_i, y_i, $i \in \{1, 2, 3\}$. Inserting the provided value of x_i, y_i, what is the value of W^*?

Problem 2.2 (6 pts) Write some codes (NumPy or MATLAB recommended) to simulate the distributed learning process. Set the parameter server learning rate λ to 0.1, and the initial weight W^0 to $[0; 0; 0]$. Run the learning process for 50 steps and use a single figure to report the value of $log(L_i)$ for all the 3 workers.

Problem 2.3 (5 pts) As mentioned in the lecture, we can reduce the amount of worker-to-server gradient communication with the *Terngrad* technique (see page 20). Redo the training process in Problem 2.2 with Terngrad compression. For each step, set s_t of each worker to the maximum absolute value of the worker's gradient. Run the learning process for 50 steps and use a single figure to report the value of $log(L_i)$ for all the 3 workers. Point out the affect of Terngrad on the trend of convergence.

Problem 2.4 (5 pts) Gradient clipping can be enforced in the *Terngrad* algorithm to unify the scale of the gradient, enabling further compression of the server-to-worker gradient communication. Redo the training process in Problem 2.2 with gradient clipping. At each step, each worker will clip its gradient in to the interval of $[-T, T]$. With $T = \{1, 2, 5, 10\}$, run the learning process for 50 steps and use a single figure for each T to report the value of $log(L_i)$ for all the 3 workers. Compare these figures with the one you got in Problem 2.2, what is the affect of gradient clipping on the convergence of distributed learning?

Problem 2.5 (5 pts) With clipped gradient and *Terngrad* technique, we can reduce the amount of both upload and download gradient communication. Redo the training process in Problem 2.2 with gradient clipping and Terngrad compression. Set clipping threshold T to 5 and use it as the s_t for all workers. Run the learning process for 50 steps and use a single figure to report the value of $log(L_i)$ for all the 3 workers. You may also plot L_i to better observe the trend of convergence. Point out the affect of Terngrad with clipping on the trend of convergence.

3 Sparsity-aware optimization

By now you have seen multiple ways to induce a sparse solution in the optimization process. This problem will provide you some examples under linear regression setting so that you can compare the effectiveness of different methods. For this problem, consider the case where we are trying to find a sparse weight W that can minimize $L = \sum_i (X_i W - y_i)^2$. Specifically, we have $X_i \in \mathbb{R}^{1\times 5}$, $W \in \mathbb{R}^{5\times 1}$ and $||W||_0 \leq 2$.

For Problem 3.1 - 3.5, consider the case where we have 3 data points: $(X_1 = [1, -2, -1, -1, 1], y_1 = -7)$; $(X_2 = [2, -1, 2, 0, -2], y_2 = -1)$; $(X_3 = [-1, 0, 2, 2, 1], Y_3 = -1)$. The objective L should be minimized through gradient descent, where you should set initial weight W^0 to $[0; 0; 0; 0; 0]$ and use learning rate $\mu = 0.02$ throughout the process. Please run gradient descent for 200 steps for all the following problems.

You will need to use NumPy or MATLAB to finish this set of questions, please put all your code for this set of questions into one file and submit it on Sakai alone with your solutions.

Problem 3.1 (5 pts) Directly minimize the objective L without any sparsity-inducing regularization/-constraint. Plot the value of $log(L)$ throughout the training, and use another figure the plot the value of each element in W in each step. Form your result, is W converging to an optimal solution? Is W converging to a sparse solution?

Problem 3.2 (5 pts) Since we have the knowledge that the ground-truth weight should have $||W||_0 \leq 2$, we can apply iterative pruning to enforce this sparse constraint. Redo the optimization process in Problem 3.1, this time prune the elements in W after every gradient descent step to ensure $||W^l||_0 \leq 2$. Plot the value of $log(L)$ throughout the training, and use another figure the plot the value of each element in W in each step. Form your result, is W converging to an optimal solution? Is W converging to a sparse solution?

Problem 3.3 (5 pts) In this problem we apply ℓ_1 regularization to induce the sparse solution. The minimization objective therefore changes to $L + \lambda||W||_1$. Please use gradient descent to minimize this objective, with $\lambda = \{0.2, 0.5, 1.0, 2.0\}$. Plot the value of $log(L)$ throughout the training, and use another figure the plot the value of each element in W in each step. Form your result, comment on the convergence performance under different λ.

Problem 3.4 (3 pts) Here we optimize the same objective as in Problem 3.3, this time using proximal gradient update. Recall that the proximal operator of the ℓ_1 regularizer is the soft thresholding function. Set the threshold in the soft thresholding function to $\{0.004, 0.01, 0.02, 0.04\}$ respectively. Plot the value of $log(L)$ throughout the training, and use another figure the plot the value of each element in W in each step. Compare the convergence performance with the results in Problem 3.3. (Hint: Optimizing $L + \lambda||W||_1$ using gradient descent with learning rate μ should correspond to proximal gradient update with threshold $\mu\lambda$)

Problem 3.5 (7 pts) Trimmed ℓ_1 ($T\ell_1$) regularizer is proposed to solve the "bias" problem of ℓ_1. For simplicity you may implement the $T\ell_1$ regularizer as applying a ℓ_1 regularization with strength λ on the smallest 3 elements of W, with no penalty on other elements. Minimize $L+\lambda T\ell_1(W)$ using proximal gradient update with $\lambda = \{1.0, 2.0, 5.0, 10.0\}$ (correspond the soft thresholding threshold $\{0.02, 0.04, 0.1, 0.2\}$). Plot the value of $log(L)$ throughout the training, and use another figure the plot the value of each element in W in each step. Comment on the convergence comparison of the Trimmed ℓ_1 and the ℓ_1. Also compare the behavior of the early steps (e.g. first 20) between the Trimmed ℓ_1 and the iterative pruning.

Problem 3.6 (5 pts Extra Credit) Redo Problem 3.2 - 3.5, this time with 2 data points (X_1, y_1) and (X_2, y_2) only. How is the convergence performance different from using 3 data points?

4 Element-wise sparse matrix encoding

The CSC (Compressed Sparse Column) format proposed by Han et al. [1] is commonly applied for the efficient storage of element-wise sparse matrices in modern DNN accelerators. This question will guide

you through the process of encoding a sparse matrix into the CSC format, and analyze the storage overhead introduced by CSC encoding.

The CSC format consists of three arrays: the Virtual Weight array which stores all the nonzero elements of the matrix; the Relative Row Index array which record the number of consecutive zeros between two nonzero elements in the same column; and the Column Pointer array which record the index of the first nonzero element in each column in the Virtual Weight array. Note that due to limited bit-width, there is an upper bound for the value of the entries in the Relative Row Index array. So if the number of consecutive zeros in a column exceed this upper bound, then we need to record a zero element in the Virtual Weight array as a "placeholder". Please refer to [1] for more details and examples on the CSC format. Note that the proposed scheme in [1] allow parallel computation among different processing elements (PEs). For simplicity here we only consider the case with a single PE.

Problem 4.1 (5 pts) Encode the following sparse weight matrix into CSC format. Write down the content of the Virtual Weight array, the Relative Row Index array and the Column Pointer array. Unlike the design in [1], here we only allow 2 bits (0-3) for each entry of the Relative Row Index.

$$\begin{bmatrix} 0 & 0 & 0 & w_6 & 0 & 0 & 0 & 0 \\ w_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 & w_9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_3 & 0 & 0 & 0 & 0 & 0 & w_{11} \\ w_1 & 0 & w_5 & w_7 & 0 & 0 & 0 & w_{12} \\ 0 & 0 & 0 & 0 & 0 & w_{10} & 0 & 0 \\ 0 & w_4 & 0 & 0 & w_8 & 0 & 0 & 0 \end{bmatrix}$$

Problem 4.2 (5 pts) For the encoding in Problem 4.1, suppose all the nonzero entries of the weight matrix are represented in 8 bits. What is the size (in bits) of the Relative Row Index array and the Column Pointer array? What is the overall compression rate comparing to storing the whole matrix directly? (Hint: you may need to consider the worst case for deciding the bit-width of the Column Pointer array entries.)

Problem 4.3 (5 pts, Extra Credit) For a matrix with 2^M rows and 2^N columns, suppose we have K non-zero entries (including the required "placeholder" zeros if necessary) for each column, and are using 4-bit to represent the entries in both the Virtual Weight array (original matrix) and the Relative Row Index array. Give a derivation with M, N and K for the total size (in bits) of all the three arrays of the CSC representation. What is the highest K the matrix can have such that the CSC format will save storage space comparing to saving the original matrix?

References

[1] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254, IEEE, 2016.