Large-scale ML (CS 260D) Submit in Gradescope (in groups of ≤ 4)
released: October 9, 2024

Homework 1

due: October 16, 4:00pm

Instructions:

1. Please create *one* submission per group and tag your other group members in Gradescope.
2. Please ensure you follow the format specified in the Gradescope online assignment. We will be using automatic grading and incorrect formats will not receive any credit.
3. There will be no partial credit for this assignment.
4. The total points are 94.

Question 1

- **SGD vs SGD with momentum** In this Colab, you can train your linear regression model, using SGD with decreasing learning rate.

  1. Implement (1) SGD with shuffling and (2) SGD with shuffling and with momentum (with the momentum coefficient β set to 0.5). Remember that SGD with shuffling samples examples *without replacement*, i.e. based on a new permutation of data points, at every epoch.

     Note: Copy only the code *you added to what is in Colab* to your Gradescope submission (we won't grade the rest of this question otherwise). Then, run the code to answer the following questions.
  2. **(12 points)** Train for 5 epochs and compare the convergence of (1) SGD, (2) SGD with shuffling, (3) SGD with shuffling and momentum, using decreasing learning rate of $0.001/\sqrt{k}$. What is the order of the algorithms from fastest to slowest in terms of convergence? You Answer:

     [✓ A. (3)-(2)-(1)]
     B. (1)-(2)-(3)
     C. (1)-(3)-(2)
     D. None of them converges.
  3. **(6 points)** Answer the same question as in part 2, but using the following the constant learning rates = 0.0001. You Answer:

     [✓ A. (3)-(2)-(1)]
     B. (1)-(2)-(3)
     C. (1)-(3)-(2)
     D. None of them converges.
  4. **(6 points)** Answer the same question as in part 2, but using the constant learning rates = 0.1. You Answer:

     A. (3)-(2)-(1)
     B. (1)-(2)-(3)
     C. (1)-(3)-(2)
     [✓ D. None of them converges.]

5. **(8 points)** Comparing the settings in part 2 (where the learning rate is $0.001/\sqrt{k}$) and part 4 (where the learning rate is 0.1), in which case does SGD with shuffling and momentum achieve a lower expected loss at the end of training? You Answer:
   [✓ A.] When learning rate is $0.001/\sqrt{k}$
   B. When learning rate is 0.1
   C. They are the same.

ps: now that you're all set, feel free to play with learning rate and optimizers more! :)

Question 2

- **Parallel SGD.** We are comparing the training of a model with 1 parameter using asynchronous vs synchronous parallel Stochastic Gradient Descent (SGD) on a machine with 3 cores.

1. **Asynchronous Parallel SGD v/s Sequential SGD (12.5 points)** Assume that calculating the gradient on each core and updating the parameter vector in the shared memory takes 0.2 seconds for all cores, and the cores have the following delays: $\delta_1 = 0.15\,\text{s}$, $\delta_2 = 0.2\,\text{s}$, $\delta_3 = 0.25\,\text{s}$, and $\delta_4 = 0.5\,\text{s}$. Assume for the gradient of every data point i we have $0 < \|\nabla f_i\| \leq \Delta$, and all cores start the parameter update simultaneously. Consider the parameters after running asynchronous parallel SGD for 0.6 seconds. What is the tightest upper bound on the noise in the parameter vector, relative to what would have occurred if the same updates had been done sequentially as in sequential SGD? Assume that in sequential SGD, the order of execution is core 1, core 2, core 3, core 4.
   Hint: For example, if in the asynchronous parallel SGD (HOGWILD) setting, only core 1 and core 2 can execute, then compare this to the parameter obtained using sequential SGD where only core 1 and core 2 execute, sequentially.
   Your Answer:
   A. $2\eta\Delta$, where η is the learning rate
   B. $4\eta\Delta$, where η is the learning rate
   [✓ C.] $6\eta\Delta$, where η is the learning rate
   D. $8\eta\Delta$, where η is the learning rate
   E. $0.9s$
   F. $1.5s$

2. **Synchronous Parallel SGD (12.5 points)** Now assume we use a synchronous approach where we wait for all the cores to finish calculating a gradient, and then we aggregate the gradients and update the shared memory (no gradient calculation happens during aggregation or memory update). If calculating the gradient on core 1, core 2, and core 3 takes 0.4s, 0.45s, and 0.5s, respectively, aggregating the gradients takes 0.3s, and writing an update in the memory takes 0.25s, how long does it take for the cores to finish 1 parameter update?

   1.05s

Question 3

- **Distributed SGD (15 points).** Assume we're training a model with distributed SGD with $m = 6$ machines. For $n = 76,800$ data points and mini-batch size $b = 1,536$ on the *master* node, how many distributed rounds do we need to finish 10 epochs? Assume the time for processing a mini-batch by the workers is 4s and each communication (from master to workers **or** from workers to master) takes 1.5s, and every worker increases the aggregation time at the master by 0.015s.

  Hint: Remember to include communication time from master to workers and workers to master!

  (1) How long does it take to train on the entire data for 10 epochs? 595.56s

  (2) After having how many workers the communication and aggregation time dominates (is strictly greater than) the gradient computation time? 67

- **Federated Learning (10 points).** Assume we're training a model with Federated Averaging on a dataset of size $n = 204,800$, with $m = 8$ homogeneous machines each having 1/8 of the data. Assuming the mini-batch size on every edge device is $b = 32$, the time for processing a mini-batch is 4s, and each communication round (from master to workers or from workers to master) takes 7s. How long does it take for the edge devices to receive the model from the server, finish 10 epochs of local updates, and then send their updates back to the server (assume no aggregation cost)? You Answer:

  A. 1024014s

  B. 32140s

  C. 2048014s

  D. 256014s

  [✓ E. 32014s]

  F. 32007s

  G. 32070s

Question 4

Implementing Distributed mini-batch GD with PySpark. We want to use PySpark to train a regression model using Distributed Gradient Descent by minimizing a squared loss function.

1. Add the gradient calculation, as well as Map and Reduce functions in indicated locations in the sample code in this Colab. The map function maps every example to its gradient, and the reduce function calculates the sum of all the mapped gradients. You can also install PySpark on your machine and run the above code locally.

   Note: Copy only the code *you added to what is in Colab* to your Gradescope submission (we won't grade the rest of this question otherwise). Then, run the code to answer the following questions.

2. **(3 points)** Compare your solution for distributed GD and distributed mini-batch DG with that of Sklearn. What do you observe and why? (Select all that are correct):

   A. Distributed mini-batch GD is converging faster (less time)

   [✓ B. Distributed GD is converging faster (less time)]

   [✓ C. Distributed mini-batch GD has a closer solution to that of sklearn]

   D. Distributed GD has a closer solution to that of sklearn

   E. Distributed GD and distributed mini-batch GD behave the same.

3. **(3 points)** Increase the number of iterations for distributed GD and compare the solutions again. What do you observe?

   A. Distributed GD's solution gets farther away from the sklearn solution

   [✓ B. Distributed GD's solution gets closer to the sklearn solution]

   C. More iterations do not change the output of Distributed GD.

4. **(3 points)** Change the mini-batch size to a larger batch size for distributed mini-batch GD and compare the solutions again. What do you observe?

   [✓ A. Distributed mini-batch GD now converges faster (less time)]

   B. Distributed mini-batch GD now converges slower (more time)

   C. Change of batch size does not influence the behavior of Distributed mini-batch GD.

5. **(3 points)** Select the learning rate for distributed mini-batch GD with a batch size of 40% (on the master node), that converges closest to the sklearn solution in 10 iterations.

   A. 0.1

   B. 0.01

   C. 0.001

   [✓ D. 0.5]

   [✓ E. 1.0]

for me, D, E pretty indistinguishable,
both ~0.01 off of each weight from sklearn