CPSC 340 2021W1 final exam - coding portion

Instructions

- You can do this exam anytime during the 12-hour final exam window. You must start the exam after 8:30am on Dec 19, 2021 and you must submit the exam by 8:30pm on Dec 19, 2021.
- Now that you've checked the exam out from Gradescope, your 90 minute window is ticking. **Make sure you leave enough time to submit, including mapping pages**; consider submitting an almost-done version with a safe margin before you do your very final edits.
- The exam is open book, meaning you are allowed to consult course materials, the internet, etc.
- You may NOT communicate with anyone else (other than the instructor) in any way during the exam. This includes posting anything on the internet (e.g. a message board, chat, tutoring service, etc.) during the exam. UBC has a strict policy on academic misconduct, along with disciplinary measures, which I can attest are not fun for anyone involved.
- You may NOT copy/paste code from anywhere. All code submitted should be code that you wrote during the exam. If you consulted any online resources and wish to cite this, feel free to drop a link in your exam submission to be safe. Submitting copied code **is considered academic misconduct.**
- Announcements or clarifications will be made on this Piazza thread:
  `https://piazza.com/class/ksums2w1qd91se?cid=770`.
  Please check it occasionally during the exam.
- If you have a question, make a **private** post on Piazza.
- This is NOT the entire final exam – there is also a Canvas portion! This coding portion and Canvas portion are equally weighted. Each part is scored out of 50 points for a total of 100.
- The submission format for this coding portion of the final exam is identical to the submission format for the homework assignments. The only difference is that you cannot work with a partner.
- If you get stuck on a question, try moving on to the next one. We tried to design the exam so that questions don't depend on each other too much.

Submission [2 points]

The above points are allocated for following the general submission instructions as you would for an assignment. In short, submit to Gradescope, make sure all your code is pasted into the PDF, and match the questions to pages when prompted by Gradescope.

1 Collaborative Filtering with Stochastic Gradient [98 points]

On Assignment 6 Question 2 you implemented Collaborative Filtering. As a reminder, the loss is

$$f(Z,W) = \frac{1}{2}\sum_{(i,j)\in S}\left(\langle z_i, w^j\rangle - y_{ij}\right)^2 + \frac{\lambda_Z}{2}\|Z\|_F^2 + \frac{\lambda_W}{2}\|W\|_F^2$$

where S is the set of (i,j) pairs that have ratings available in the training set and $\langle z_i, w^j\rangle$ denotes the dot product between the ith row of Z and the jth column of W.

The gradient of this loss is

$$\nabla_Z f(Z,W) = R_S W^T + \lambda_Z Z$$
$$\nabla_W f(Z,W) = Z^T R_S + \lambda_W W$$

where $R_S = ZW - Y$ except that, where ratings are missing, the corresponding elements of R_S are set to 0. This is the same "masking" of R_S that we used in a6. In symbols,

$$(R_S)_{ij} = \begin{cases} \langle z_i, w^j\rangle - y_{ij} & \text{if } (i,j)\in S \\ 0 & \text{otherwise} \end{cases}$$

If you run `python main.py -q 1` the code loads the Movie Reviews dataset from a6 and fits a collaborative filtering model to it, achieving a validation RMSE of around 0.92. If you look at the code, you will see that it uses the gradient expression above.

Note that this code uses regular gradient descent where Z and W are both updated, simultaneously, in every iteration. This is in contrast with the a6 code, which used alternating minimization. Both are valid approaches.

For the remainder of this exam we will use stochastic gradient (SG) to minimize the above objective $f(Z,W)$. In traditional stochastic gradient we pick a random training example i and compute the gradient as if this were the only example in the training set. Here with collaborative filtering, SG works by randomly picking an (i,j) *pair* from within S and computing the gradient as if this were the only *rating* in the training set.

1.1 SG: Naive Implementation [10 points]

Let (i, j) represent the randomly selected rating from S for one iteration of SG. Then the gradient considering only this rating is given by the same equation above, except with S modified so that it only contains one rating (i, j). In other words, R_S will only contain one nonzero element.

Also, due to the details of writing the loss as an average (which we're skipping over here in the interests of time), λ_Z and λ_W must be divided by $|S|$, which is called `num_ratings_train` in the code. This is already done for you in the code, as you'll see.

Modify the provided code in `-q 1.1` so that it implements SG instead of regular gradient descent. The provided code already selects a random (i, j) for you. Set all the values of R_S to 0 except for the ijth element. Submit any code that you added or changed.

Note: the code will be too slow to run for even one epoch, so **don't wait for it to run**! We'll implement something more efficient later on.

Answer:

```
Rsij = Rs[i, j]
Rs[:, :] = 0
Rs[i, j] = Rsij
```

1.2 Running time of naive SG [20 points]

Using Big-O notation, state the running time of **one epoch** of the naive SG implementation from the previous part. Justify your answer. Your answer may involve n and/or d and/or k and/or $|S|$, which denotes the number of ratings in the training set.

Note: even if you weren't able to complete the previous part, you can still try to answer this question about the running time.

Answer: For each of the d iterations, it costs $O(knd)$ to compute $\hat{Y}$ and the gradients. Therefore, the total running time for one epoch is $O(knd^2)$.

1.3 SG: Faster Implementation [48 points]

In the previous implementation, R_S was almost entirely zeros (all except element ij). Therefore we wasted a lot of computation multiplying things by 0 and adding 0. We're also updating all the entries in Z and W despite the fact that most of the entries in Z and W are not even used for predicting the rating (i, j). In fact, given that

$$\hat{y}_{ij} = \langle z_i, w^j \rangle ,$$

only the ith row of Z and the jth column of W are used in predicting the rating (i, j). If all the rest of Z and W aren't even being used to predict that rating, we don't need to update them when we sample that rating. Therefore, we can speed this code up a lot by only updating z_i and w^j at each iteration, rather than the entire Z and W.

Modify the provided code in `-q 1.3` so that it implements SG where only z_i and w^j are updated at every iteration. Write your code efficiently; that is, don't compute things that you don't use. Submit any code that you added or changed. Also, submit your train and validation RMSE after 10 epochs of training. You should be able to obtain a similar validation error to the original gradient descent implementation in the provided code.

Note: there is a subtlety here regarding regularization, but it is already handled for you in the code, so this paragraph is more just for your information. The issue is that although the first part of the per-rating loss/gradient only involves z_i and w^j, technically the regularization still involves all of Z and W. But we only want to update z_i and w^j, not all of Z and W. To fix this, we need to adjust the regularization strength: instead of dividing by $|S|$ as in the previous part, we instead divide by the number of ratings for user i when updating z_i and the number of ratings for movie j when updating w^j. Again, this is already done for you in the code.

Answer:

1. Train RMSE: 0.729
2. Valid RMSE: 0.924

```
Y_hat_ij = Z[i] @ W[:, j] + avg_rating
Rs_ij = Y_hat_ij - Y_train[i, j]

Z_i_grad = Rs_ij * W[:,j] + lammyZ * Z[i] / ratings_per_user_i[i]
W_j_grad = Rs_ij * Z[i] + lammyW * W[:,j] / ratings_per_movie_j[j]
```

1.4 Running time of faster SG [20 points]

Using Big-O notation, state the running time of **one epoch** of the faster SG implementation from the previous part. Justify your answer. Your answer may involve n and/or d and/or k and/or $|S|$, which denotes the number of ratings in the training set.

Note: even if you weren't able to complete the previous part, you can still try to answer this question about the running time.

Answer: For each of the d iterations, it now only costs $O(k)$ to compute $\hat{y}_{ij}$ and the gradients. Therefore, the total running time for one epoch is reduced to $O(kd)$.