Practical Work 2

CAO Anh Quan
M2 D&K Factorization-Based Data Modeling

December 23, 2018

Contents

1 Project Structure

- **Factorization_Matrix_TP2_Report.pdf** The Project Report in PDF.
- **tp2.ipynb** A Jupyter Notebook contains code and experiments for the Matrix Factorization with Stochastic Gradient Decent.

- **tp2.html** A HTML version of the Jupyter Notebook

2 Complete the stochastic gradient algorithm.

```
batchSize = 1024 #the number of elements that we will use at each iteration

eta = 0.01 #step-size
numIter = 1000

rmse_sgd = []

for t in range(numIter):

    # get a random batch of index from the data
    data_index = random.sample(range(len(data)), batchSize)

    # for each element in the data batch, update the corresponding elements in
    W and H
    for i in data_index:

        # for each element in the batch, find its corresponding 'i', 'j',
        # and value by using the Xlist array
        cur_i = row[i]
        cur_j = col[i]
        cur_x = data[i]


        # compute the current xhat, for the current i and j
        cur_xhat = W[cur_i, :] @ H[:, cur_j]

        # compute the gradients for the 'corresponding elements' of W and H
        # not all the elements of W and H will be updated
        cur_W = W[cur_i, :]
        cur_H = H[:, cur_j]
        O = np.ones((K, K))

        grad_w = (cur_x - cur_W @ cur_H) * cur_H.T
        grad_h = cur_W.T * (cur_x - cur_W @ cur_H)

        #take a gradient step
        W[cur_i,:] = W[cur_i,:] + eta * grad_w
        H[:,cur_j] = H[:,cur_j] + eta * grad_h
```

3 At the end of each iteration, compute the roor-mean-squared-error

```
error = np.sqrt((Xsp - M.multiply(W.dot(H))).power(2).sum() / N)
```

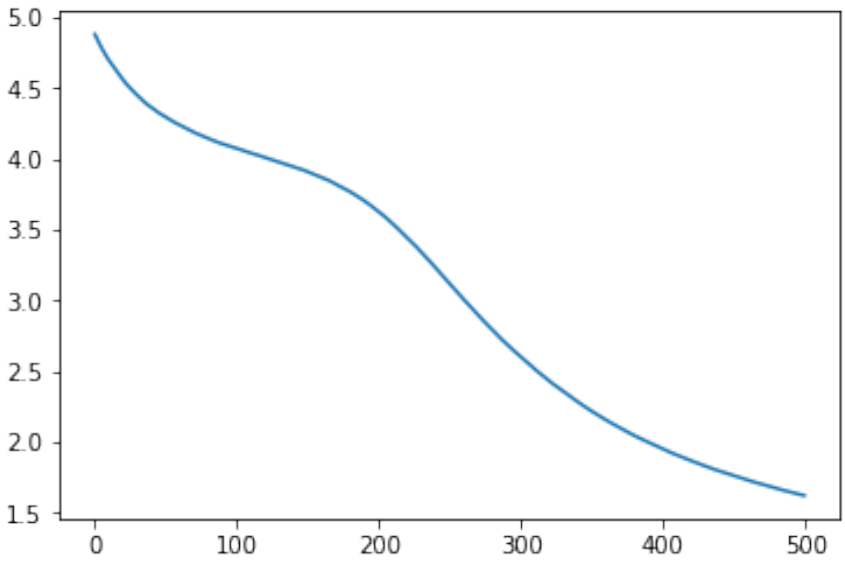


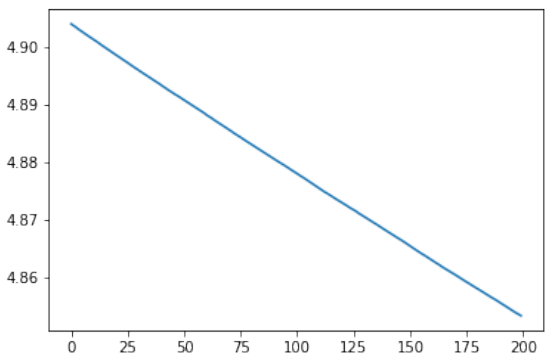
Figure 1: The change of root mean square error value of the Stochastic Gradient Decent algorithm with respect to the number of iterations.

4 Play with the algorithm parameters

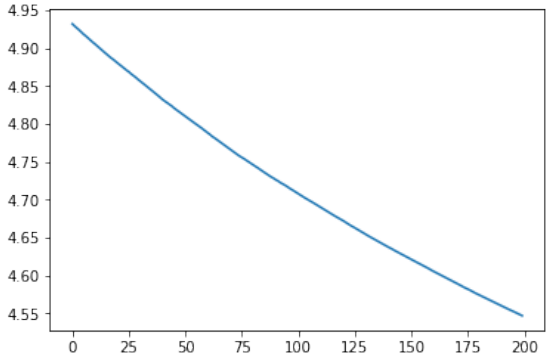
4.1 Step-size

4.1.1 Experiment

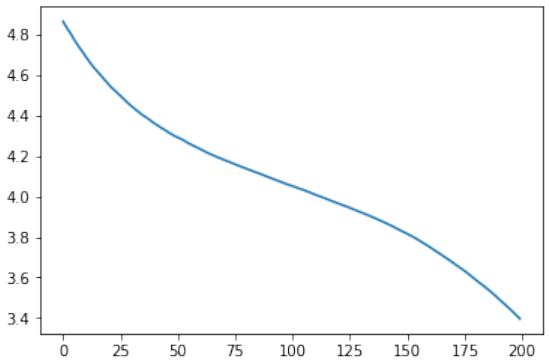
In this experiment, we run the SGD algorithm with the following **step-size** values: [0.0001, 0.001, 0.01, 0.1, 0.2]. Then, we plot the change of the error values to see the effect of **step-size** on the SGD algorithm.



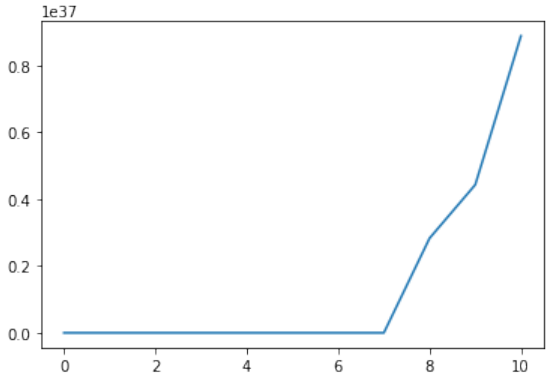
(a) step-size = 0.0001



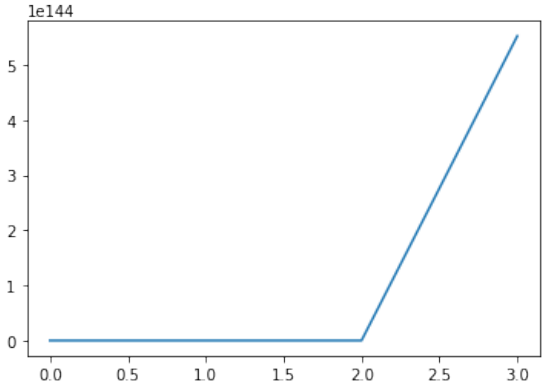
(b) step-size = 0.001



(c) step-size = 0.01



(d) step-size = 0.1



(e) step-size = 0.2

Figure 2: The change of root mean square error values with respect to the number of iterations when we change the value of **step-size**

4.1.2 Discussion

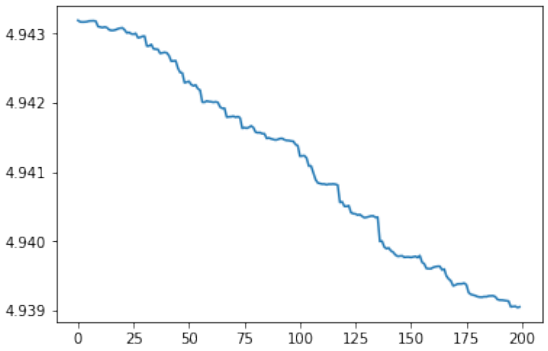
We can see from the figures that the error decrease faster when we increase the step-size which means that the SGD converges faster with bigger step-size. However, the SGD does not converge if the step-size is too big. For instance, we can observe from the figures that the error increases with step-size equal to 0.1 and 0.2. Therefore, with bigger step-size the SGD converges faster (error decrease to small value faster); But, if the step-size is too large, the

error will increase, and the SGD will not converge. However, if the step-size if too small, the convergence process will be very slow, and the SGD will need a large number of iterations until convergence. Therefore, it is important to select a suitable value of step-size, not too large and not too small.

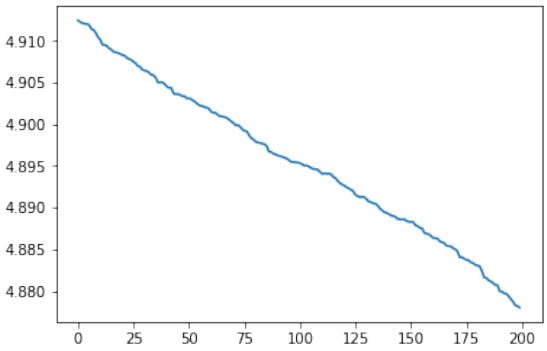
4.2 Batch-size

4.2.1 Experiment

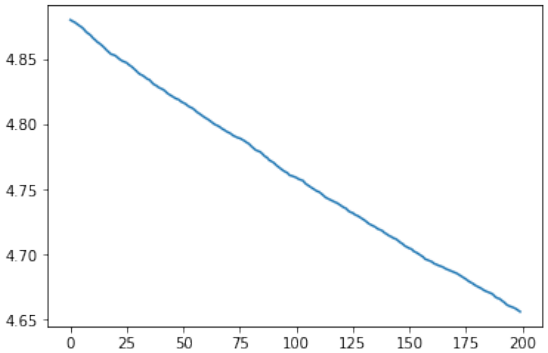
In this section, we run the SGD algorithm with the following **batch-size** values: [1, 8, 64, 256, 2048, 5096]. Then, we plot the change of the root mean square error with corresponding to the iterations to see the effect of batch-size on the SGD.



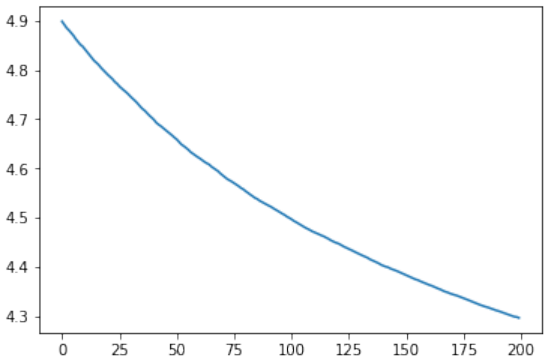
(a) batch-size = 1



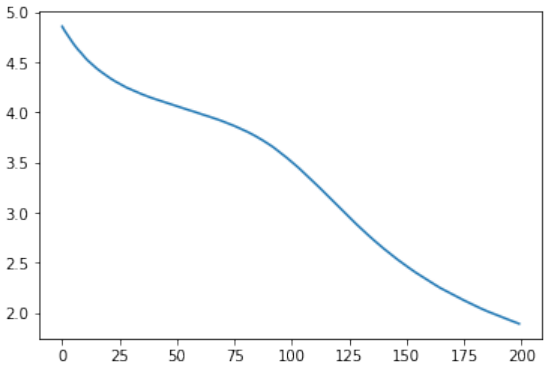
(b) batch-size = 8



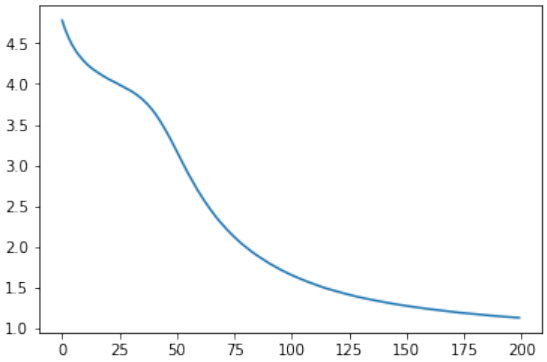
(c) batch-size = 64



(d) batch-size = 256

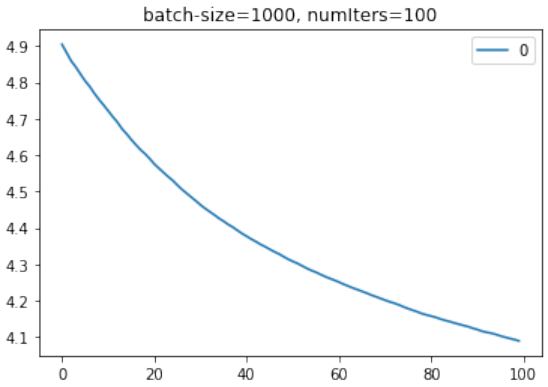


(e) batch-size = 2048

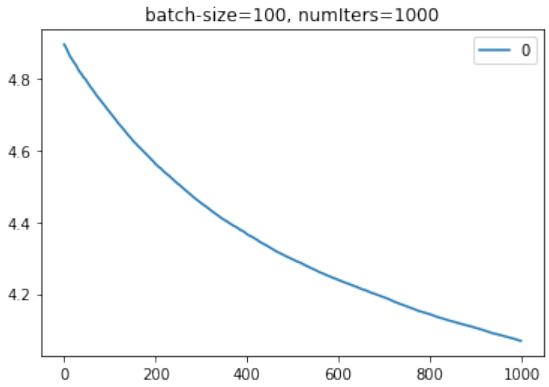


(f) batch-size = 5096

Figure 3: The change of root mean square error values with respect to the number of iterations when we change the value of **step-size**



(a) Error of the last iteration: 4.08. Error of the first iteration: 4.90



(b) Error of the last iteration: 4.06. Error of the first iteration: 4.89

Figure 4: The change of root mean square error values when we use the same amount of processing data but different batch-size and number of iterations.

4.2.2 Discussion

According to the figures, the line is smoother with bigger batch-size. With small batch-size, the error decreases in general, but it also fluctuates because the algorithm can only look at the small number of observations to update the parameter which makes the algorithm moves in many directions and discovers new areas.

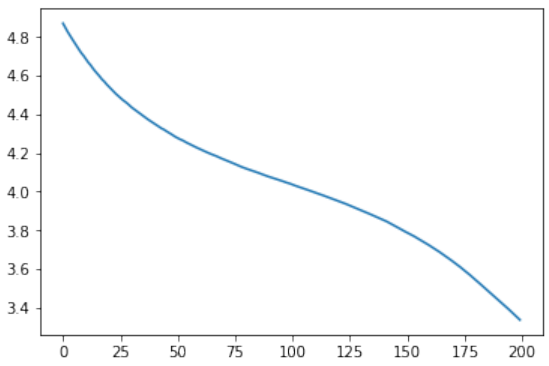
In our experiment, we can see the SGD converges faster and more accurate with bigger batch-size because, with the same number of iterations, we can consider more observations for updating the parameters with larger batch-size. The algorithm also has more observations to update the parameters at each iteration.

However, it is not fair to say that bigger batch-size is always better because in this case the amount of processing data is not the same since we limit the number of iterations is 200. According to figure 4b, we can see that the change of error is almost the same for different number of batch-size and iterations but same amount of processing data. Moreover, the SGD with more iterations and lower batch-size is slightly better than the SGD with less iterations but bigger batch-size. In the case of the same amount of processing data, we will need more updates with smaller batch-size, and more updates are usually better than one big update. It's the critical idea of stochastic gradient descent compare to gradient descent. Geometrically, more updates are generally better because, with only one big update, we go to only one direction of the (exact) gradient while with many updates we can go in many directions of the (approximated) gradient. Although the direction will be less accurate, it is better to change the directions many times because we do not get real predictive power with a slightly smaller error on the training set. Besides, bigger batch-size is better for parallelization. Therefore, It is essential to choose a suitable value of batch-size.

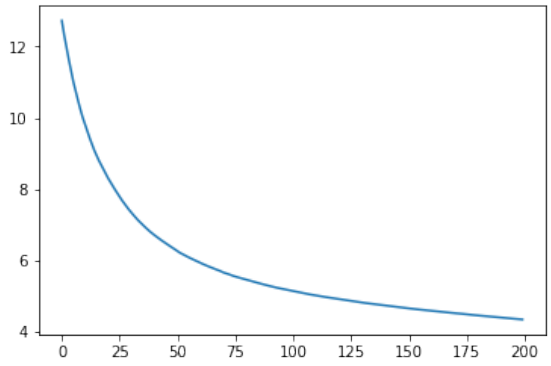
4.3 Initialization

4.3.1 Experiment

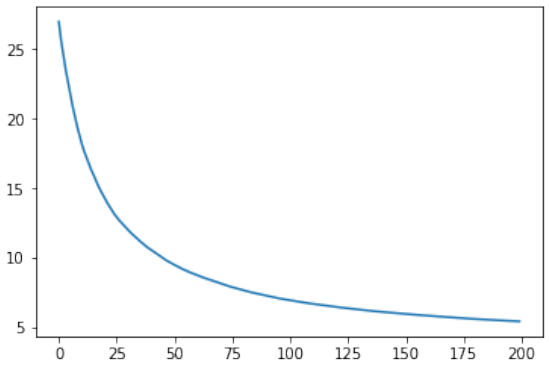
In this section, we change the initialization of the SGD algorithm with the following **seed** values: [1, 2, 3, 4, 5]. Then, we plot the change of the root mean square error with corresponding to the iterations to see the effect of the seed values on the SGD.



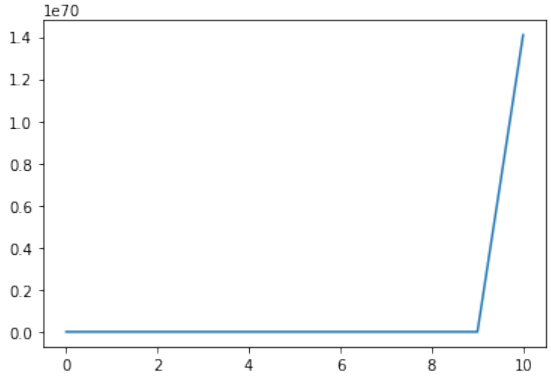
(a) seed = 1



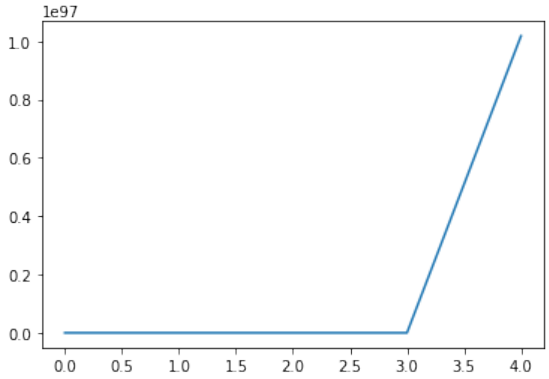
(b) seed = 2



(c) seed = 3



(d) seed = 4



(e) seed = 5

Figure 5: The change of root mean square error values with respect to the number of iterations when we change the value of the **seed** values

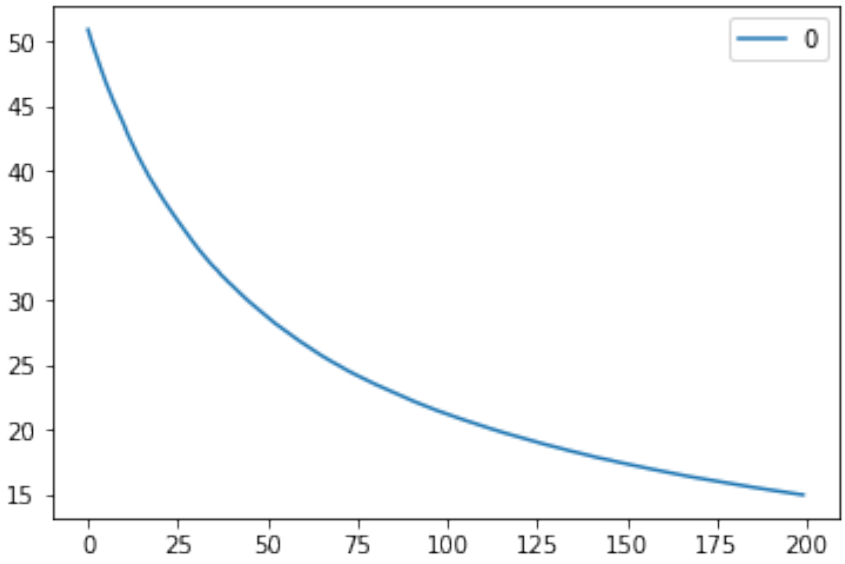


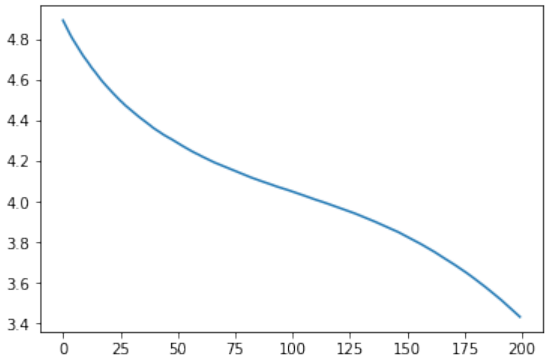
Figure 6: The change of root mean square error with respect to the number of iterations with **Seed = 4 and smaller step-size**. The SGD is able to convert using smaller step-size.

4.3.2 Discussion

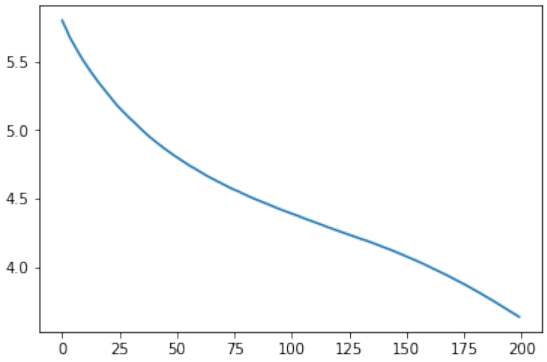
According to the figures, increasing the seed make the error bigger and the gradient larger which leads to faster convergence. Therefore, if the seed is huge, the gradient will also be huge which makes the algorithm fails to convert. From the figure 5, we can see that the SGD cannot convert with seed equal to 4 and 5. However, base on figure 6, the algorithm can convert again when we use smaller step-size.

4.4 Rank of the factorization

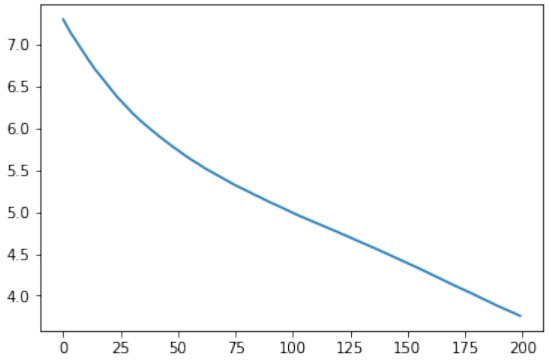
In this section, we test the SGD algorithm with the following **rank** values: [10, 20, 40, 80, 160, 320]. Then, we plot the change of the root mean square error with corresponding to the iterations to see the effect of the rank values on the SGD.



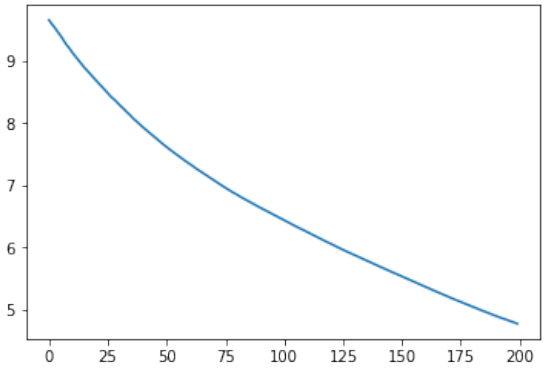
(a) rank = 10



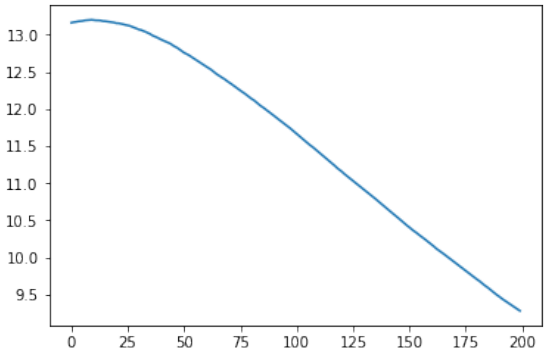
(b) rank = 20



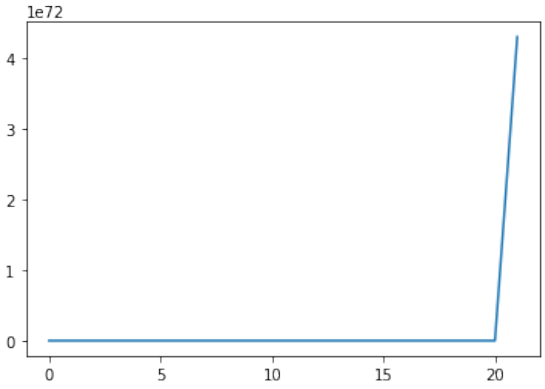
(c) rank = 40



(d) rank = 80



(e) rank = 160



(f) rank = 320

Figure 7: The change of root mean square error values with respect to the number of iterations when we change the value of the **rank** values

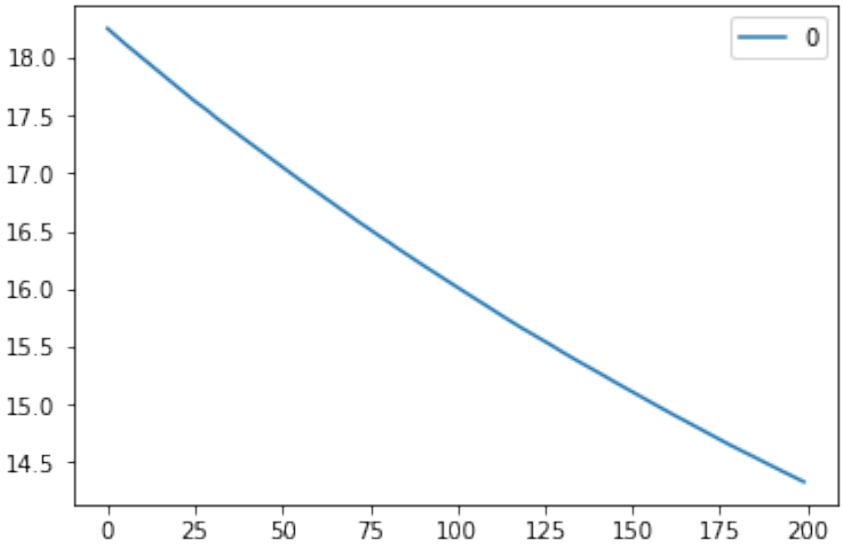


Figure 8: The change of root mean square error with respect to the number of iterations using **Rank = 320 with smaller step-size**. The SGD converts again when we decrease the step-size

4.4.1 Discussion

We can see from the figure 7 that as we increase the rank, the error becomes bigger and decreases faster. However, the SGD fails to convert when we are using too large rank (320). Fortunately, if we use smaller step size, the algorithm can again convert to minima according to figure 8. It seems that the gradient becomes larger as we increase the rank which makes the SGD cannot convert. Therefore, Increasing the rank make the error bigger and the gradient larger which increase the convergence speed. However, if the rank is too big, the algorithm can be unable to convert. To make it convert again, we need to use smaller step-size.

4.5 Interaction between the step-size and batch-size

4.5.1 Experiment

In this section, to see the interaction between the batch-size and step-size, we run the test using the following batch-size and step-size:

- batch-size = [1, 10, 100]
- step-size = [0.0001, 0.001, 0.01]

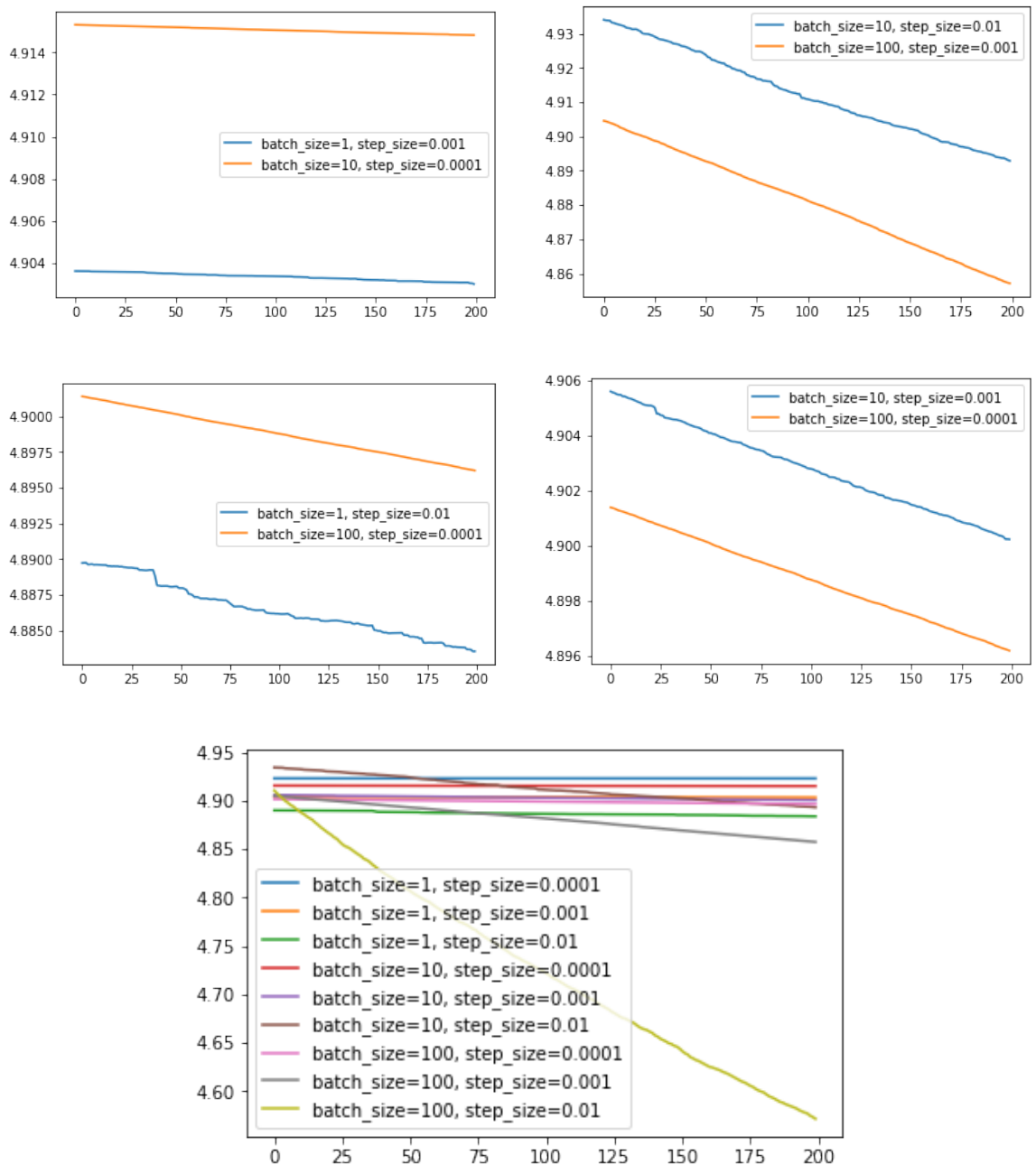


Figure 9: The change of root mean square error with respect to the number of iterations values corresponding to the values of batch-size and step-size

4.5.2 Discussion

We can see from the figure 9 that the step-size and batch-size are counter proportional. Increasing the batch-size by ten times has the same effect as decreasing the step-size ten times. In each figure, the shape of the blue and orange lines are quite similar. Furthermore, the range the error value is also approximately equal to each other. Therefore, sometimes

we can increase the batch-size instead of decreasing the step-size. However, lowering the step-size make the learning much slower while increase the batch-size make the gradient less noisy and reduce the number of updates.

5 After estimating W and H, use them to recommend a movie for a given user.

1. Compute the rating **predicted X** using the estimated W and H.
2. Apply **Mask** to get all **unseen movies rating** for the current user.
3. Recommend the **unseen movie** with the **maximum rating** to the user.

```
# Compute the predicted X using the estimated W and H matrix
predicted_X = W @ H

user_index = 11

# Use W, H, and M to compute a 'movie_index' for the user
# Get the rating for all movies corresponding to this user
movies = predicted_X[:, user_index].reshape(-1, 1)

# Filter out all movies that are seen by this user
unseen_M = M[:, user_index].toarray()
unseen_movies = unseen_M * movies

# Recommend the unseen movie with largest rating to the current user
recommend_movie_index = np.argmax(unseen_movies)

print('Recommend movie', recommend_movie_index,'to user', user_index)
print("rating", unseen_movies[recommend_movie_index][0])
```

Example result

```
Recommend movie 907 to user 11
rating 3.211889103425637
```