CSE 258 - HW 3

Jin Dai / A92408103

Task (Rating prediction)

1
```
import copy
import gzip
import math
import random
from collections import defaultdict
import tensorflow as tf
import csv
```

2
```
def read_gz(path):
    for l in gzip.open(path, 'rt'):
        yield eval(l)

def read_csv(path):
    f = gzip.open(path, 'rt')
    c = csv.reader(f)
    header = next(c)
    print(header)
    for l in c:
        yield l
```

3
```
dataset = list(read_csv("trainInteractions.csv.gz"))
dataset[:2]
```

```
['user_id', 'recipe_id', 'date', 'rating']
```

3
```
[['88348277', '03969194', '2004-12-23', '5'],
 ['86699739', '27096427', '2002-01-12', '4']]
```

9.

4
```
userIDs = {}
itemIDs = {}
interactions = []

for d in dataset:
    u = d[0]
    i = d[1]
    r = int(d[3])
    if not u in userIDs: userIDs[u] = len(userIDs)
    if not i in itemIDs: itemIDs[i] = len(itemIDs)
    interactions.append((u,i,r))
```

5
```
random.shuffle(interactions)
len(interactions)
```

```
500000
```

```
nTrain = 400000
data_train = interactions[:nTrain]
data_valid = interactions[nTrain:]
```

```
itemsPerUser = defaultdict(list)
usersPerItem = defaultdict(list)
for u,i,r in data_train:
    itemsPerUser[u].append(i)
    usersPerItem[i].append(u)
```

```
mu = sum([r for _,_,r in data_train]) / len(data_train)
```

```
class LatentFactorModelBiasOnly(tf.keras.Model):
    def __init__(self, mu, lamb):
        super(LatentFactorModelBiasOnly, self).__init__()
        # Initialize to average
        self.alpha = tf.Variable(mu)
        # Initialize to small random values
        self.betaU = tf.Variable(tf.random.normal([len(userIDs)],stddev=0.001))
        self.betaI = tf.Variable(tf.random.normal([len(itemIDs)],stddev=0.001))
        self.lamb = lamb

    # Prediction for a single instance (useful for evaluation)
    def predict(self, u, i):
        p = tf.Variable(self.alpha)
        if 0 <= u < self.betaU.shape[0]:
            p.assign_add(self.betaU[u])
        if 0 <= i < self.betaI.shape[0]:
            p.assign_add(self.betaI[i])
        return p

    # Regularizer
    def reg(self):
        return self.lamb * (tf.reduce_sum(self.betaU**2)
                            + tf.reduce_sum(self.betaI**2))

    # Prediction for a sample of instances
    def predictSample(self, sampleU, sampleI):
        u = tf.convert_to_tensor(sampleU, dtype=tf.int32)
        i = tf.convert_to_tensor(sampleI, dtype=tf.int32)
        beta_u = tf.nn.embedding_lookup(self.betaU, u)
        beta_i = tf.nn.embedding_lookup(self.betaI, i)
        pred = self.alpha + beta_u + beta_i
        return pred

    # Loss
    def call(self, sampleU, sampleI, sampleR):
        pred = self.predictSample(sampleU, sampleI)
        r = tf.convert_to_tensor(sampleR, dtype=tf.float32)
        return tf.nn.l2_loss(pred - r) / len(sampleR)
```

```
def trainingStepBiasOnly(model, interactions, optimizer):
    Nsamples = 50000
    with tf.GradientTape() as tape:
        sampleU, sampleI, sampleR = [], [], []
```

```
        for _ in range(Nsamples):
            u,i,r = random.choice(interactions)
            sampleU.append(userIDs[u])
            sampleI.append(itemIDs[i])
            sampleR.append(r)

        loss = model(sampleU,sampleI,sampleR)
        loss += model.reg()
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients((grad, var) for
        (grad, var) in zip(gradients, model.trainable_variables)
        if grad is not None)
    return loss.numpy()
```

11
```
def fit(data_train, mu, lamb=1, learning_rate=0.001, print_log=False):
    model = LatentFactorModelBiasOnly(mu, lamb)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    min_obj = math.inf
    best_model = copy.deepcopy(model)
    counter = 0
    while counter < 200:
        local_obj = trainingStepBiasOnly(model, data_train, optimizer)
        if print_log and counter % 10 == 0:
            print("iteration %d, objective = %f, min_obj=%f" % (counter, local_obj, min_obj))
        if local_obj < min_obj:
            min_obj = local_obj
            best_model = copy.deepcopy(model)
        counter += 1
    return best_model, min_obj
```

12
```
def MSE(pred, y):
    differences = [(x-y)**2 for x,y in zip(pred,y)]
    return sum(differences) / len(differences)
```

13
```
modelBiasOnly, min_obj = fit(data_train, mu, print_log=True)
print("Min objective has been reached at %f." % min_obj)
```

```
iteration 0, objective = 0.609791, min_obj=inf
iteration 10, objective = 0.458401, min_obj=0.473227
iteration 20, objective = 0.453397, min_obj=0.450814
iteration 30, objective = 0.446109, min_obj=0.445760
iteration 40, objective = 0.444509, min_obj=0.441185
iteration 50, objective = 0.453136, min_obj=0.441185
iteration 60, objective = 0.447732, min_obj=0.440643
iteration 70, objective = 0.441606, min_obj=0.440643
iteration 80, objective = 0.440891, min_obj=0.437924
iteration 90, objective = 0.438580, min_obj=0.434747
iteration 100, objective = 0.456176, min_obj=0.430425
iteration 110, objective = 0.451166, min_obj=0.430425
iteration 120, objective = 0.450597, min_obj=0.430425
iteration 130, objective = 0.456477, min_obj=0.430425
iteration 140, objective = 0.461755, min_obj=0.429986
iteration 150, objective = 0.442673, min_obj=0.429986
iteration 160, objective = 0.447112, min_obj=0.429986
iteration 170, objective = 0.449068, min_obj=0.429986
iteration 180, objective = 0.449690, min_obj=0.429986
iteration 190, objective = 0.450194, min_obj=0.429986
Min objective has been reached at 0.429986.
```

```
14 def predict(model, user, item):
       u = userIDs[user] if user in userIDs else -1
       i = itemIDs[item] if item in itemIDs else -1
       return model.predict(u, i).numpy()
```

```
15 pred = [predict(modelBiasOnly, u,i) for u,i,_ in data_valid]
```

```
16 y_valid = [r for _,_,r in data_valid]
```

```
17 MSE(pred, y_valid)
```

```
17 0.9110000589935944
```

10.

```
18 modelBiasOnly.betaI[0]
```

```
18 <tf.Tensor: shape=(), dtype=float32, numpy=1.5287205e-06>
```

```
19 interactions[0]
```

```
19 ('53280340', '99980672', 4)
```

```
20 init_betaI = float(modelBiasOnly.betaI[0])
   init_betaU = float(modelBiasOnly.betaU[0])
   init_I, init_U, _ = interactions[0]
   max_betaI = init_betaI
   max_I = init_I
   min_betaI = init_betaI
   min_I = init_I
   max_betaU = init_betaU
   max_U = init_U
   min_betaU = init_betaU
   min_U = init_U
```

```
21 for usr, index in userIDs.items():
       beta_u = modelBiasOnly.betaU[index]
       if beta_u > max_betaU:
           max_betaU = beta_u
           max_U = usr
       if beta_u < min_betaU:
           min_betaU = beta_u
           min_U = usr

   for item, index in itemIDs.items():
       beta_i = modelBiasOnly.betaI[index]
       if beta_i > max_betaI:
           max_betaI = beta_i
           max_I = item
       if beta_i < min_betaI:
           min_betaI = beta_i
           min_I = item


   print("User \'%s\' has the largest betaU: %f" % (max_U, max_betaU))
```

```
22 print("User \'%s\' has the smallest betaU: %f" % (min_U, min_betaU))
   print("Recipe \'%s\' has the largest betaI: %f" % (max_I, max_betaI))
   print("Recipe \'%s\' has the smallest betaI: %f" % (min_I, min_betaI))
```

```
User '32445558' has the largest betaU: 0.001695
User '70705426' has the smallest betaU: -0.000648
Recipe '17799621' has the largest betaI: 0.000196
Recipe '74912490' has the smallest betaI: -0.000361
```

11.

```
28 hp_lambdas = [2, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001]
   min_MSE = math.inf
   best_lamb = 1
   best_model = modelBiasOnly
   for l in hp_lambdas:
       model, min_obj = fit(data_train, mu, lamb=l)
       pred = [predict(model, u,i) for u,i,_ in data_valid]
       mse = MSE(pred, y_valid)
       print("Min objective has reached at %f with lambda=%f. MSE on validation set: %f." % (min_ob
       if mse < min_MSE:
           min_MSE = mse
           best_lamb = l
           best_model = model
```

```
Min objective has reached at 0.425581 with lambda=2.000000. MSE on validation set: 0.911024.
Min objective has reached at 0.429982 with lambda=1.000000. MSE on validation set: 0.911018.
Min objective has reached at 0.433401 with lambda=0.100000. MSE on validation set: 0.910442.
Min objective has reached at 0.426937 with lambda=0.010000. MSE on validation set: 0.905963.
Min objective has reached at 0.421587 with lambda=0.001000. MSE on validation set: 0.891642.
Min objective has reached at 0.410917 with lambda=0.000100. MSE on validation set: 0.873868.
Min objective has reached at 0.391931 with lambda=0.000010. MSE on validation set: 0.864338.
Min objective has reached at 0.392097 with lambda=0.000001. MSE on validation set: 0.868307.
```

```
29 print('When we set lambda to %f, the model performs better and yields min MSE=%f' % (best_lamb,
```

```
When we set lambda to 0.000010, the model performs better and yields min MSE=0.864338
```

```
30 predictions = open("predictions_Rated.txt", 'w')
   for l in open("stub_Rated.txt"):
       if l.startswith("user_id"):
           #header
           predictions.write(l)
           continue
       u,i = l.strip().split('-')
       predictions.write(u + '-' + i + ',' + str(predict(best_model, u, i)) + '\n')
   predictions.close()
```

Kaggle Username: Jin Dai