Original code: https://www.youtube.com/watch?v=G4MBc40rQ2k

Credit: Spencer Pao

Dataset for applying:

https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
# import the dataset
import pandas as pd
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving rating.csv to rating.csv
Saving anime.csv to anime (1).csv
```

```
anime_df = pd.read_csv('/content/anime.csv')
ratings_df = pd.read_csv('/content/rating.csv',usecols=range(3))
```

```
print('The dimensions of anime dataframe are:', anime_df.shape)
print('The dimensions of ratings dataframe are:', ratings_df.shape)
```

```
The dimensions of anime dataframe are: (12294, 7)
The dimensions of ratings dataframe are: (7813737, 3)
```

```
# Take a look at anime_df
anime_df.head()
```

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262

```
# Take a look at ratings_df
ratings_df.head()
```

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

```
# Mapping anime name into a dictionary for reference
anime_names = anime_df.set_index('anime_id')['name'].to_dict()
anime_genres = anime_df.set_index('anime_id')['genre'].to_dict()

n_users = len(ratings_df.user_id.unique())
n_items = len(ratings_df.anime_id.unique())
print("Number of unique users:", n_users)
print("Number of unique animes:", n_items)
print("The full rating matrix will have:", n_users*n_items, 'elements.')
print('----------')
print("Number of ratings:", len(ratings_df))
```

```
Number of unique users: 73515
Number of unique animes: 11200
The full rating matrix will have: 823368000 elements.
----------
Number of ratings: 7813737
```

```
import torch
import numpy as np
from torch.autograd import Variable
from tqdm import tqdm_notebook as tqdm

class MatrixFactorization(torch.nn.Module):
    def __init__(self, n_users, n_items, n_factors=20):
        super().__init__()
        # create user embeddings
        self.user_factors = torch.nn.Embedding(n_users, n_factors) # think of this as a lookup table for the input.
        # create item embeddings
        self.item_factors = torch.nn.Embedding(n_items, n_factors) # think of this as a lookup table for the input.
        self.user_factors.weight.data.uniform_(0, 0.05)
        self.item_factors.weight.data.uniform_(0, 0.05)

    def forward(self, data):
        # matrix multiplication
        users, items = data[:,0], data[:,1]
        return (self.user_factors(users)*self.item_factors(items)).sum(1)
    # def forward(self, user, item):
    #   # matrix multiplication
    #     return (self.user_factors(user)*self.item_factors(item)).sum(1)

    def predict(self, user, item):
        return self.forward(user, item)
```

```
# Creating the dataloader (necessary for PyTorch)
from torch.utils.data.dataset import Dataset
from torch.utils.data import DataLoader # package that helps transform data to machine learning readiness


class Loader(Dataset):
    def __init__(self):
        self.ratings = ratings_df.copy()

        # Extract all user IDs and movie IDs
        users = ratings_df.user_id.unique()
        animes = ratings_df.anime_id.unique()

        #--- Producing new continuous IDs for users and movies ---

        # Unique values : index
        self.userid2idx = {o:i for i,o in enumerate(users)}
        self.movieid2idx = {o:i for i,o in enumerate(animes)}

        # Obtained continuous ID for users and movies
        self.idx2userid = {i:o for o,i in self.userid2idx.items()}
        self.idx2movieid = {i:o for o,i in self.movieid2idx.items()}

        # return the id from the indexed values as noted in the lambda function down below.
        self.ratings.anime_id = ratings_df.anime_id.apply(lambda x: self.movieid2idx[x])
        self.ratings.user_id = ratings_df.user_id.apply(lambda x: self.userid2idx[x])


        self.x = self.ratings.drop(['rating'], axis=1).values
        self.y = self.ratings['rating'].values
        self.x, self.y = torch.tensor(self.x), torch.tensor(self.y) # Transforms the data to tensors (ready for torch models.)

    def __getitem__(self, index):
        return (self.x[index], self.y[index])

    def __len__(self):
        return len(self.ratings)


num_epochs = 2
cuda = torch.cuda.is_available()

print("Is running on GPU:", cuda)

model = MatrixFactorization(n_users, n_items, n_factors=8)
print(model)
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name, param.data)

if cuda:
    model = model.cuda()
```

```
Is running on GPU: True
MatrixFactorization(
  (user_factors): Embedding(73515, 8)
  (item_factors): Embedding(11200, 8)
)
user_factors.weight tensor([[0.0105, 0.0290, 0.0112,  ..., 0.0047, 0.0050, 0.0320],
        [0.0072, 0.0453, 0.0438,  ..., 0.0348, 0.0433, 0.0254],
        [0.0450, 0.0108, 0.0181,  ..., 0.0441, 0.0118, 0.0254],
        ...,
        [0.0279, 0.0274, 0.0474,  ..., 0.0440, 0.0365, 0.0473],
        [0.0083, 0.0495, 0.0387,  ..., 0.0499, 0.0039, 0.0075],
        [0.0335, 0.0376, 0.0493,  ..., 0.0119, 0.0364, 0.0267]])
item_factors.weight tensor([[0.0282, 0.0252, 0.0201,  ..., 0.0173, 0.0424, 0.0173],
        [0.0346, 0.0016, 0.0339,  ..., 0.0048, 0.0376, 0.0190],
        [0.0380, 0.0445, 0.0100,  ..., 0.0306, 0.0308, 0.0238],
        ...,
        [0.0370, 0.0313, 0.0018,  ..., 0.0093, 0.0062, 0.0051],
        [0.0492, 0.0119, 0.0245,  ..., 0.0015, 0.0046, 0.0485],
        [0.0243, 0.0392, 0.0440,  ..., 0.0195, 0.0192, 0.0327]])
```

```
# MSE loss
loss_fn = torch.nn.MSELoss()
```

```
# ADAM optimizier
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

```
# Train data
train_set = Loader()
train_loader = DataLoader(train_set, 128, shuffle=True)
```

```
for it in tqdm(range(num_epochs)):
    losses = []
    for x, y in train_loader:
         if cuda:
            x, y = x.cuda(), y.cuda()
            optimizer.zero_grad()
            outputs = model(x)
            loss = loss_fn(outputs.squeeze(), y.type(torch.float32))
            losses.append(loss.item())
            loss.backward()
            optimizer.step()
    print("iter #{}".format(it), "Loss:", sum(losses) / len(losses))
```

```
<ipython-input-29-dad152416852>:1: TqdmDeprecationWarning: This function will be
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for it in tqdm(range(num_epochs)):
```

100% 2/2 [04:57<00:00, 148.81s/it]

```
iter #0 Loss: 10.278244866486293
iter #1 Loss: 5.2330222179774655
```

```
# By training the model, we will have tuned latent factors for movies and users.
c = 0
uw = 0
iw = 0
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name, param.data)
        if c == 0:
          uw = param.data
          c +=1
        else:
          iw = param.data
        #print('param_data', param_data)
```

```
user_factors.weight tensor([[-0.1012, -0.0814, -0.1018,  ..., -0.1097, -0.1091, -0.0764],
        [-0.0562, -0.0181, -0.0196,  ..., -0.0286, -0.0201, -0.0379],
        [ 0.6102,  0.5768,  0.5824,  ...,  0.6097,  0.5774,  0.5922],
        ...,
        [ 0.0914,  0.0909,  0.1109,  ...,  0.1075,  0.1001,  0.1108],
        [ 0.6133,  0.6571,  0.6517,  ...,  0.6616,  0.6125,  0.6157],
        [ 0.1606,  0.1647,  0.1764,  ...,  0.1390,  0.1634,  0.1538]],
       device='cuda:0')
item_factors.weight tensor([[ 1.5576,  1.5807,  1.5529,  ...,  1.4956,  1.6058,  1.6046],
        [ 1.7133,  1.7067,  1.6863,  ...,  1.6688,  1.7119,  1.6962],
        [ 1.4883,  1.5311,  1.5320,  ...,  1.4801,  1.4538,  1.5547],
        ...,
        [-0.0266, -0.0323, -0.0617,  ..., -0.0543, -0.0574, -0.0584],
        [-0.0144, -0.0516, -0.0391,  ..., -0.0620, -0.0589, -0.0150],
        [ 0.0878,  0.1027,  0.1076,  ...,  0.0831,  0.0827,  0.0963]],
       device='cuda:0')
```

```
trained_anime_embeddings = model.item_factors.weight.data.cpu().numpy()
```

```
len(trained_anime_embeddings) # unique movie factor weights
```

```
11200
```

```
from sklearn.cluster import KMeans
# Fit the clusters based on the movie weights
kmeans = KMeans(n_clusters=10, random_state=0).fit(trained_anime_embeddings)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
```

```
for cluster in range(5):
```

```
    print("Cluster #{}".format(cluster))
    movs = []
    for movidx in np.where(kmeans.labels_ == cluster)[0]:
      movid = train_set.idx2movieid[movidx]
      rat_count = ratings_df.loc[ratings_df['anime_id']==movid].count()[0]
      movs.append((anime_names.get(movid,"Unknown"), rat_count))
    for mov in sorted(movs, key=lambda tup: tup[1], reverse=True)[:10]:
          print("\tName:", mov[0])
          print("\t\tNumber of Ratings:", mov[1])
```

```
        Name: Shujii no Inbou
                Number of Ratings: 18
        Name: Doubutsu Mura no Sports Day
                Number of Ratings: 14
        Name: Wonder (Movie)
                Number of Ratings: 12
        Name: Suzume no Oyado
                Number of Ratings: 12
        Name: Pony Metal U-GAIM Promotion Film
                Number of Ratings: 11
        Name: Tondemo Nezumi Daikatsuyaku
                Number of Ratings: 10
        Name: Shin Megami Tensei Devil Children
                Number of Ratings: 10
        Name: Sankou to Tako: Hyakumanryou Chinsoudou
                Number of Ratings: 10
Cluster #3
        Name: Pupa
                Number of Ratings: 2677
        Name: Boku no Pico
                Number of Ratings: 2475
        Name: Pico to Chico
                Number of Ratings: 1508
        Name: Pico x CoCo x Chico
                Number of Ratings: 1397
        Name: Eiken: Eikenbu yori Ai wo Komete
                Number of Ratings: 924
        Name: Issho ni Training: Training with Hinako
                Number of Ratings: 695
        Name: Diabolik Lovers Recap
                Number of Ratings: 552
        Name: Issho ni Sleeping: Sleeping with Hinako
                Number of Ratings: 543
        Name: Makura no Danshi
                Number of Ratings: 470
        Name: Ame-iro Cocoa
                Number of Ratings: 399
Cluster #4
        Name: Death Note
                Number of Ratings: 39340
        Name: Sword Art Online
                Number of Ratings: 30583
        Name: Shingeki no Kyojin
                Number of Ratings: 29584
        Name: Code Geass: Hangyaku no Lelouch
                Number of Ratings: 27718
        Name: Angel Beats!
                Number of Ratings: 27183
        Name: Fullmetal Alchemist
                Number of Ratings: 25032
        Name: Fullmetal Alchemist: Brotherhood
                Number of Ratings: 24574
        Name: Toradora!
                Number of Ratings: 24283
        Name: Code Geass: Hangyaku no Lelouch R2
                Number of Ratings: 24242
        Name: Sen to Chihiro no Kamikakushi
                Number of Ratings: 22974
```

1. Dataset chosen:

The anime dataset was chosen because of its relevance to the model being built and I am also personally in how to build a recommendation system.

2. Data modification: