Report: Deep AutoEncoder Based Recommender System

Overview

The goal of this assignment was to build a Deep AutoEncoder based recommender system using the MovieLens dataset. The system predicts user ratings for movies that have not been rated, enabling movie recommendations. This report outlines the implementation process, data preparation steps, model architecture, training procedure, and comments on the results obtained.

---

1. Data Preparation

1.1 Dataset Description

The MovieLens dataset consists of:

- **100,000 ratings** from **943 users** on **1,682 movies**.
- Ratings are integers ranging from 1 to 5, with 0 indicating an unrated movie.

The data was cleaned to ensure each user rated at least 20 movies. The dataset includes:

- `ratings.csv`: User ratings for movies.
- `movies.csv`: Metadata for movies.

1.2 Data Splitting

The data was split into training and validation sets based on the `timestamp` column:

- **Training Set**: Ratings up to the 98th percentile of timestamps.
- **Validation Set**: Remaining ratings.

Key Steps:

1. **Merging and Descriptive Analysis**:
    - `ratings.csv` was merged with `movies.csv` using `movieId`.
    - Summary statistics were printed to verify data distribution.
2. **Data Filtering**:

- Users in the validation set who were not present in the training set were removed.
- Movies not present in the training set were removed from the validation set.

Output:

- **Train Users**: 595
- **Validation Users**: 595
- **Train Movies**: 9559
- **Validation Movies**: 9559

The resulting data matrices were stored in `train.csv` and `test.csv`.

---

2. Model Architecture

2.1 AutoEncoder Structure

The AutoEncoder was implemented with the following architecture:

- **Encoder**:
  - Input Layer: 9559 nodes (dimensionality of the movie-user matrix).
  - Hidden Layers: 512, 512, and 1024 nodes with ReLU activation.
- **Decoder**:
  - Hidden Layers: 1024, 512, and 512 nodes with ReLU activation.
  - Output Layer: 9559 nodes.

2.2 Loss Function

A custom `Masked Mean Squared Error (MSE)` loss function was used:

- The loss function only considers non-zero ratings (i.e., ratings provided by users) for error calculation.

---

3. Training Procedure

3.1 DataLoader Creation

- Custom `TrainDataset` and `TestDataset` classes were implemented to load user-movie rating matrices.

- Data was batched using `DataLoader` for efficient model training.

3.2 Training Loop

The training loop iterated for 40 epochs:

- Input data was passed through the encoder to generate compressed representations, then decoded back to reconstruct the original ratings.
- The optimizer used was `Adam` with a learning rate of 0.001.
- Loss was computed using the custom `MSELoss_with_Mask` function, and gradients were updated accordingly.

---

4. Results and Observations

4.1 Loss Reduction

The training loss decreased consistently over the 40 epochs:

- **Initial Loss** (Epoch 1): ~10.77
- **Final Loss** (Epoch 40): ~0.61

This indicates that the model effectively learned to minimize the reconstruction error.

4.2 Collected Results

- **Training Data**: Successfully split and filtered.
- **Validation Data**: Successfully split and filtered.
- **User-Movie Rating Matrices**: Constructed with `0.0` for missing ratings.
- **Training Performance**: The model converged well, showing consistent improvement in loss reduction across epochs.

4.3 Future Considerations

- **Parameter Tuning**: Further tuning of learning rates, hidden layers, and activation functions may yield improved performance.
- **Data Scaling**: Normalizing the ratings or using different loss functions could be explored.
- **Model Complexity**: Increasing the depth of the encoder/decoder or adding regularization may enhance generalization.

Conclusion

The Deep AutoEncoder-based recommender system was successfully implemented and trained using the MovieLens dataset. The model demonstrated a strong ability to minimize reconstruction error, achieving stable loss values. This approach can be further enhanced through additional fine-tuning and testing on larger datasets for real-world applications.

Code:

```
'''
Created on Nov 05, 2023

@author: ahmed-notebook
'''
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt


class TrainDataset(Dataset):
    def __init__(self, train_file, transform=None):
        self.data = pd.read_csv(train_file)
        self.data = self.data.iloc[:, 1:]  # Drop the first column
        self.transform = transform

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

    def __getitem__(self, ind):
        user_vector = self.data.iloc[ind].values.astype(np.float32)
        user_vector = torch.FloatTensor(user_vector)  # Convert directly
to tensor
        return user_vector

class TestDataset(Dataset):
    def __init__(self, test_file, transform=None):
```

```
        self.data = pd.read_csv(test_file)
        self.data = self.data.iloc[:, 1:]  # Drop the first column
        self.transform = transform

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

    def __getitem__(self, ind):
        user_vector = self.data.iloc[ind].values.astype(np.float32)
        user_vector = torch.FloatTensor(user_vector)  # Convert directly
to tensor
        return user_vector


def prepare_train_validation_movielens_step1():
    rat = pd.read_csv('/content/ratings.csv')
    mov = pd.read_csv('/content/movies.csv')
    df_combined = pd.merge(rat, mov, on='movieId')
    print(rat.describe())
    ts = rat['timestamp'].quantile(0.98)
    train_ratings = pd.DataFrame(columns=['userId', 'movieId', 'rating'])
    validation_ratings = pd.DataFrame(columns=['userId', 'movieId',
'rating'])
    for i in range(len(rat)):
        if rat['timestamp'].iloc[i] <= ts:
            train_ratings = pd.concat([train_ratings,
pd.DataFrame([{'userId': rat['userId'].iloc[i], 'movieId':
rat['movieId'].iloc[i], 'rating': rat['rating'].iloc[i]}])])
            validation_ratings = pd.concat([validation_ratings,
pd.DataFrame([{'userId': rat['userId'].iloc[i], 'movieId':
rat['movieId'].iloc[i], 'rating': rat['rating'].iloc[i]}])])
        else:
            validation_ratings = pd.concat([validation_ratings,
pd.DataFrame([{'userId': rat['userId'].iloc[i], 'movieId':
rat['movieId'].iloc[i], 'rating': rat['rating'].iloc[i]}])])
        if i % 10000 == 0:
            print(i, "Completed")
    print(len(train_ratings))
    print(len(validation_ratings))
    # Remove users in validation set those are not present in Training Set
```

```
    train_users = train_ratings['userId'].unique()
    users_not_in_train_set = []

    for i in range(1, 611):
        if i not in train_users:
            users_not_in_train_set.append(i)

    for i in users_not_in_train_set:
        validation_ratings =
validation_ratings[validation_ratings['userId'] != i]

    validation_ratings.reset_index(drop=True)

    print(len(train_ratings['movieId'].unique()))
    print(len(validation_ratings['movieId'].unique()))
    # Remove Movies that are not in the Train Set
    validation_movies = validation_ratings['movieId'].unique()
    train_movies = train_ratings['movieId'].unique()
    movies_not_in_train_set = []

    for i in validation_movies:
        if i not in train_movies:
            movies_not_in_train_set.append(i)

    for i in movies_not_in_train_set:
        validation_ratings =
validation_ratings[validation_ratings['movieId'] != i]

    validation_ratings.reset_index(drop=True)
    print('Train Users: ', train_ratings['userId'].nunique())
    print('Validation Users: ', validation_ratings['userId'].nunique())
    print('Train Movies: ', train_ratings['movieId'].nunique())
    print('Validation Movies: ', validation_ratings['movieId'].nunique())
    train_ratings.to_csv("/content/train_ratings.csv")
    validation_ratings.to_csv("/content/validation_ratings.csv")

def prepare_traintest_movielens_step2():
    tr_ratings = pd.read_csv('/content/train_ratings.csv')
    val_ratings = pd.read_csv('/content/validation_ratings.csv')
```

```
    train_dataset = tr_ratings.pivot_table(index='userId',
columns='movieId', values='rating')
    train_dataset.fillna(0, inplace=True)
    print(train_dataset.head(10))
    test_dataset = val_ratings.pivot_table(index='userId',
columns='movieId', values='rating')
    test_dataset.fillna(0, inplace=True)
    print(test_dataset.head(10))
    train_dataset.to_csv('/content/train.csv')
    test_dataset.to_csv('/content/test.csv')

def get_traintestloaders():
    train_dat = TrainDataset('/content/train.csv')
    test_dat = TestDataset('/content/test.csv')
    train_loader = DataLoader(dataset=train_dat, batch_size=128,
shuffle=True, num_workers=1)
    test_loader = DataLoader(dataset=test_dat, batch_size=128,
shuffle=True, num_workers=1)
    return train_loader, test_loader

class MSELoss_with_Mask(nn.Module):
    def __init__(self):
        super(MSELoss_with_Mask, self).__init__()

    def forward(self, inputs, targets):
        # Masking into a vector of 1's and 0's.
        mask = (targets != 0).float()
        # Actual number of ratings.
        number_ratings = torch.max(torch.sum(mask),
torch.tensor(1.0).cuda())
        error = torch.sum(mask * (targets - inputs) ** 2)
        loss = error / number_ratings
        return loss

class AutoEncoder(nn.Module):
    def __init__(self, encoder_layers_sizes, activation='ReLU'):
        super(AutoEncoder, self).__init__()
        # Encoder layers
        self.encoder = nn.Sequential(
            nn.Linear(encoder_layers_sizes[0], encoder_layers_sizes[1]),
```

```
            nn.ReLU(),
            nn.Linear(encoder_layers_sizes[1], encoder_layers_sizes[2]),
            nn.ReLU(),
            nn.Linear(encoder_layers_sizes[2], encoder_layers_sizes[3]),
            nn.ReLU()
        )

        # Decoder layers
        self.decoder = nn.Sequential(
            nn.Linear(encoder_layers_sizes[3], encoder_layers_sizes[2]),
            nn.ReLU(),
            nn.Linear(encoder_layers_sizes[2], encoder_layers_sizes[1]),
            nn.ReLU(),
            nn.Linear(encoder_layers_sizes[1], encoder_layers_sizes[0])
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

import torch.optim as optim

def train(model, criterion, optimizer, train_loader, test_loader,
num_epochs=50):
    model.train()
    for epoch in range(num_epochs):
        train_loss = 0.0
        for data in train_loader:
            inputs = data.cuda()
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, inputs)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        print(f'Epoch {epoch+1}, Loss: {train_loss/len(train_loader)}')

def main():
```

```
    prepare_train_validation_movielens_step1()
    prepare_traintest_movielens_step2()
    train_loader, test_loader = get_traintestloaders()
    encoder_layers_sizes = [9559, 512, 512, 1024]
    model = AutoEncoder(encoder_layers_sizes)
    model = model.cuda()
    criterion = MSELoss_with_Mask().cuda()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    train(model, criterion, optimizer, train_loader, test_loader, 40)

if __name__ == '__main__':
    main()
```

Output:

Please refer to the Python notebook file.