كليـــــة العلـــــوم
والتقنيات - مراكش
FACULTE DES SCIENCES
ET TECHNIQUES - MARRAKECH

Master IAII
Pr. Sara QASSIMI
08 Decembre 2023

Tutorial: Autoencoder for Collaborative Filtering

Objective:

In this tutorial, we explored the implementation of an autoencoder using Keras for collaborative filtering. Collaborative filtering is a powerful technique commonly employed in recommendation systems. The provided code efficiently handles sparse matrices, making it suitable for collaborative filtering tasks with large datasets.

Prerequisites

Ensure that you have Keras and other necessary libraries installed. If needed, update the future module using `sudo pip install -U future`.

Step 1: Import Libraries

```
# Import necessary libraries
from __future__ import print_function, division
from builtins import range
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from scipy.sparse import save_npz, load_npz

import keras.backend as K
from keras.models import Model
from keras.layers import Input, Dropout, Dense
from keras.regularizers import l2
from keras.optimizers import SGD
```

Step 2: Load Data

```
# Load data and create masks
A = load_npz("Atrain.npz")
A_test = load_npz("Atest.npz")
mask = (A > 0) * 1.0
mask_test = (A_test > 0) * 1.0

# Create copies for shuffling
A_copy = A.copy()
mask_copy = mask.copy()
A_test_copy = A_test.copy()
mask_test_copy = mask_test.copy()

N, M = A.shape
print("N:", N, "M:", M)
print("N // batch_size:", N // batch_size)
```

Step 3: Data Preprocessing

Center the data by calculating the mean.

```
# Center the data
mu = A.sum() / mask.sum()
print("mu:", mu)
```

Step 4: Building the Autoencoder Model

Build a simple autoencoder model with one hidden layer.

```
# Build autoencoder model
i = Input(shape=(M,))
x = Dropout(0.7)(i)
x = Dense(700, activation='tanh', kernel_regularizer=l2(reg))(x)
x = Dense(M, kernel_regularizer=l2(reg))(x)
```

Step 5: Custom Loss Function

Custom Loss Function

```
# Define custom loss function
def custom_loss(y_true, y_pred):
    mask = K.cast(K.not_equal(y_true, 0), dtype='float32')
    diff = y_pred - y_true
    sqdiff = diff * diff * mask
    sse = K.sum(K.sum(sqdiff))
    n = K.sum(K.sum(mask))
    return sse / n
```

Step 6: Data Generators

Create data generators for training and testing.

```
# Define data generators
def generator(A, M):
    # ... (previous code)

def test_generator(A, M, A_test, M_test):
    # ... (previous code)
```

Step 6: Compiling and Training the Model

Compile the model and train it using the generators.

```
# Compile and train the model
model = Model(i, x)
model.compile(
    loss=custom_loss,
    optimizer=SGD(lr=0.08, momentum=0.9),
    metrics=[custom_loss],
)

r = model.fit(
    generator(A, mask),
    validation_data=test_generator(A_copy, mask_copy, A_test_copy , mask_test_copy),
    epochs=epochs,
    steps_per_epoch=A.shape[0] // batch_size + 1,
    validation_steps=A_test.shape[0] // batch_size + 1,
)
print(r.history.keys())
```

Step 7: Visualizing Results

Plot the training and testing losses.

```
# Visualize results
plt.plot(r.history['loss'], label="train loss")
plt.plot(r.history['val_loss'], label="test loss")
plt.legend()
plt.show()

plt.plot(r.history['custom_loss'], label="train mse")
plt.plot(r.history['val_custom_loss'], label="test mse")
plt.legend()
plt.show()
```

كليـــــة العلـــــوم
والتقنيات - مراكش
FACULTE DES SCIENCES
ET TECHNIQUES - MARRAKECH

Master IAII
Pr. Sara QASSIMI
08 Decembre 2023

Future Challenges and Steps for Improvement

While the current implementation serves as a solid foundation for collaborative filtering, there are several areas for improvement.

One potential challenge is handling larger datasets more efficiently, considering the computational demands of training deep models.

Additionally, exploring advanced techniques such as incorporating embeddings or exploring different neural network architectures may enhance the model's predictive performance.

Regular model evaluation, hyperparameter tuning, and scalability considerations are key steps to further improve the code for real-world collaborative filtering scenarios.

Experimenting with different optimization algorithms and regularization techniques could also be valuable in refining the model's generalization capabilities. Keep in mind that collaborative filtering is an evolving field, and staying informed about the latest advancements can contribute to the continual improvement of recommendation systems.