Project 01

RECOMMENDER SYSTEMS SPARK

➢ Danh sách thành viên

NGUYỄN HOÀNG TUẤN CƯỜNG	*1712309*
HOÀNG GIA BẢO	*1712284*
NGUYỄN THANH BÌNH	*1712295*

SPARSE FC
(SPARSE FULLY-CONNECTED)

I. Ý tưởng chính của phương pháp

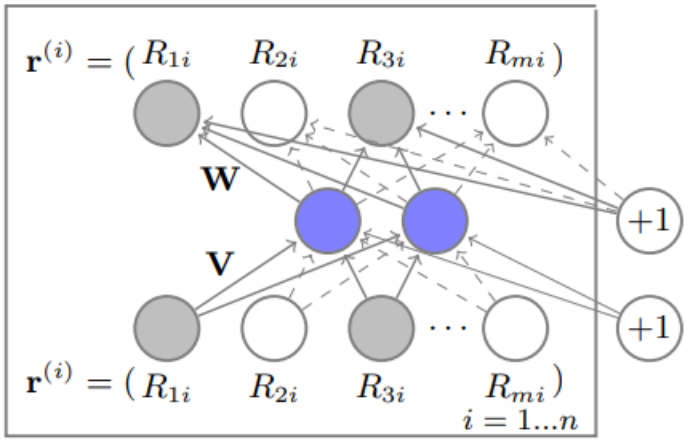
- Cải tiến dựa trên phương pháp I-AutoRec

Tổng quan về I-AutoRec

Ta có m người dùng, n mặt hàng, và một ma trận R chứa các đánh giá m người dùng đối với n mặt hàng đó.

Mỗi mặt hàng i có một vec-tơ đánh giá(r(i)) của m người dùng, r(i) = (R_{1i}, …, R_{mi})

Mục tiêu là thiết kế một mô hình tự động mã hóa dựa trên từng mặt hàng i, nhận input là từng vector r(i), chiếu nó sang một miền không gian tiềm ẩn có số chiều thấp, sau đó tái tạo lại vec-tơ r(i) đó trong miền không gian đầu ra, ta sẽ dự đoán được các đánh giá chưa có của khách hàng đối với mặt hàng i đó



Hình 1. Mô hình Item-based AutoRec. Hình tròn màu xám thể hiện người dùng có đánh giá mặt hàng i này, hình tròn trắng thể hiện người dùng đó không đánh giá mặt hàng i này, hình tròn xanh là lớp tiềm ẩn có số chiều thấp. Các mũi tên đang thể hiện quá trình học (lan truyền ngược) để cập nhật trọng số. Chỉ có các hình tròn xám mới được quan tâm để lan truyền ngược(mũi tên liền), các hình tròn trắng thì không quan tâm(mũi tên nét đứt)

Hàm để tái tạo lại các vec-tơ đầu vào r:

$$h(\mathbf{r};\theta) = f\left(\mathbf{W} \cdot g(\mathbf{V}\mathbf{r} + \boldsymbol{\mu}) + \mathbf{b}\right)$$

Ta muốn cực tiểu hóa độ lỗi của quá trình học

$$\min_{\theta} \sum_{i=1}^{n} ||\mathbf{r}^{(i)} - h(\mathbf{r}^{(i)}; \theta))||_{\mathcal{O}}^2 + \frac{\lambda}{2} \cdot (||\mathbf{W}||_F^2 + ||\mathbf{V}||_F^2),$$

Chỉ tính norm-2 các dự đoán của những đánh giá có tồn tại ban đầu (hay ta dễ hiểu hơn, là học làm sao cho các dự đoán của các đánh giá ban đầu, sau khi được tái tạo sẽ gần đúng với đánh giá đầu vào nhất)

Kí hiệu F ở đây nghĩa là tính norm-2 toàn bộ ma trận

Sparse FC

Có kiến trúc mạng tương tự kiến trúc của I-AutoRec, nhưng ma trận trọng số W, V được tham số hóa lại bằng cách:

- Ta sẽ nhận ma trận kiểu Hadamard ma trận W, V với ma trận trọng số kernel(w_hat) để có được ma trận thể hiện các liên kết thưa thớt. Từng giá trị của w_hat sẽ được tính bằng công thức:

$$K_{\text{fs}}(\vec{u}_i, \vec{v}_j) = \max\left(0, 1 - a \cdot D(\vec{u}_i, \vec{v}_j)\right)$$

D là khoảng cách 2 vec-tơ

Cách tính độ lỗi sau mỗi mẫu huấn luyện cũng có thay đổi nhỏ so với phương pháp I-AutoRec, đó là việc cộng thêm sai số của ma trận trọng số kernel (w_hat) để học trong quá trình lan truyền ngược nữa:

Đây chính là tổng norm-2 của ma trận W và V(giống như trong hàm cực tiểu độ lỗi phương pháp I-AutoRec bên trên). $\lambda_2 > 0$ là tham số kiểm soát overfitting của mô hình

$$R = \lambda_2 \left(\sum_{ij} \alpha_{ij} + \sum_{ij} \beta_{ij} \right) \longrightarrow$$

$$+\lambda_0 \left(\sum_{ij} K(\vec{v}_i, \vec{u}_j) + \sum_{ij} K(\vec{s}_i, \vec{t}_j) \right), \longrightarrow$$

Đây là norm-2 của ma trận w_hat.
λ_0 là tham số thể hiện mật độ của dữ liệu trong ma trận user-item ban đầu

Công thức định nghĩa mạng nơ-ron kernel có d chiều

$$x_j^{(l)} = f_j \left(\sum_i \alpha_i^{(l)} K(\vec{u}_i^{(l)}, \vec{v}_j^{(l)}) x_i^{(l-1)} \right).$$

Định nghĩa hàm kernel:

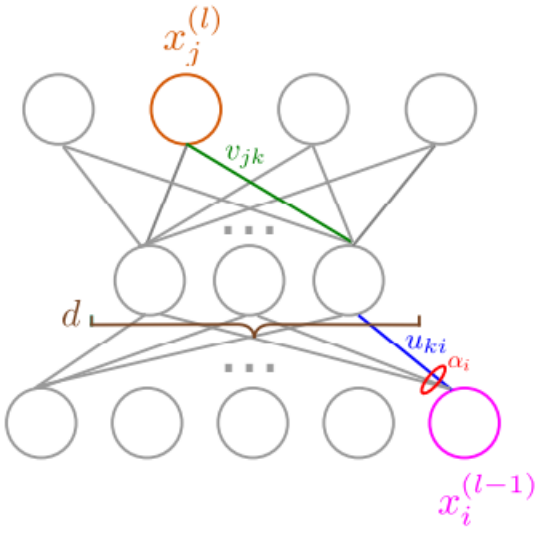
$$K(\vec{u}, \vec{v}) = \left\langle \phi(\vec{u}), \phi(\vec{v}) \right\rangle = \langle \vec{u}^*, \vec{v}^* \rangle$$

Nếu là mạng fully-connected, ta cho hàm Φ thành hàm tuyến tính (Φ(x) = x), tham số $alpha_i = 1$,

hàm K tích bằng tích vô hướng, ta sẽ có được như sau:

$$\begin{aligned} x_j^{(l)} &= \sum_i \alpha_i K(\vec{u}_i, \vec{v}_j) x_i^{(l-1)} = \sum_i \alpha_i \vec{u}_i^T \vec{v}_j x_i^{(l-1)} \qquad (3) \\ &= \sum_i \alpha_i \sum_k u_{ki} v_{jk} x_i^{(l-1)} = \sum_k v_{jk} \sum_i \alpha_i u_{ki} x_i^{(l-1)} \end{aligned}$$

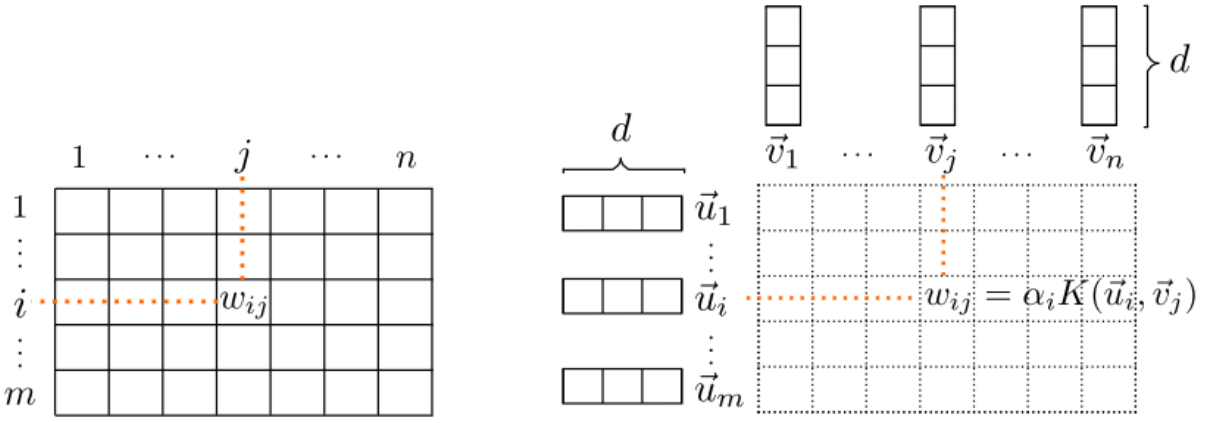
Lúc này (3) sẽ được biểu diễn như mô hình mạng dưới đây:



Giả sử tầng (l – 1) có m phần tử, tầng (l) có n phần tử

Lúc này, thay vì chỉ có 1 ma trận trọng số W(m x n) giữa 2 tầng liên tiếp trong mạng, ta đã tách ra được 2 ma trận khác nhau: ma trận U(m x d) và ma trận V(d x n). Có thể giải thích điều này như là sự phân rã ma trận W lớn thành 2 ma trận con nhỏ hơn. Thao tác này mạng lại lợi ích:

- Bản chất ma trận item-user ban đầu có độ thưa thớt cực cao, và đánh giá của phim cũng dựa vào số lượng tương đối nhỏ các tính năng (VD: thể loại, diễn viên…). Nên ta giảm chiều xuống để tăng tính tổng quát hóa, tìm ra mỗi tương quan giữa các đánh giá, ở không gian nhỏ hơn
- Ta thấy số lượng tham số có được = (m x d + d x n) < m x n → d < (m*n)/(m + n) (*). Nếu ta chọn d thỏa điều kiện (*) ta thấy số lượng tham số của mô hình giảm xuống đáng kể, giúp tăng tốc khả năng học



conventional weight matrix

$n \cdot m$

reparametrized weight matrix

$d \cdot (n + m)$

Lựa chọn kernel sao cho các nơ-ron ở càng xa thì liên kết sẽ càng yếu, các nơ-ron ở càng gần thì sẽ liên kết càng mạnh (đây là lí do ta chọn hàm kernel K_{fs} bên trên)

2/ **Kết quả thực nghiệm**

Method	ML-10M	ML-1M	ML-100K
LLORMA	0.782	0.833	0.898
GC-MC	0.777	0.832	0.910
I-CFN	0.777	0.832	-
I-AutoRec	0.782	0.831	0.895*
I-AutoRec (2)	0.770*	0.830*	0.895*
CF-NADE (2)	0.771	0.829	-
KernelNet (1)	-	0.838	0.898
KernelNet (2)	-	0.836	0.901
Sparse FC (1)	0.784	0.830	**0.890**
Sparse FC (2)	**0.769**	**0.824**	0.894

Bảng 1. So sánh các phương pháp khác nhau trên các tập dữ liệu MovieLens.
Giá trị trong bảng là giá trị RMSE trung bình của các đánh giá cần điền khuyết
(giá trị càng thấp càng tốt) từ phương pháp 5-Folds Cross-Validation.
Đối với tập dữ liệu ML-10M và ML-1M ta chia tập train/validation theo tỉ lệ 90/10,
với tập dữ liệu ML-100K ta chia tập train/validation theo tỉ lệ 80/20.
Giá trị nằm trong dấu ngoặc đơn sau tên mỗi phương pháp là số tầng ẩn của mạng nơ-ron

Method	Parameters	MACs	ML-1M
I-AutoRec (1)	6.05 M	3.03 M	0.831
I-AutoRec (2)	6.30 M	3.28 M	0.830
I-KernelNet (1)	0.67 M	3.03 M	0.838
I-KernelNet (2)	0.72 M	3.28 M	0.836
Sparse FC (1)	6.70 M	2.77 M	0.830
Sparse FC (2)	7.00 M	2.23 M	0.824

Bảng 2. Phương pháp SparseFC đã giảm được phép toán nhân tích lũy (MACs) cho các đánh giá cần điền khuyết

3/ Ưu, nhược điểm của phương pháp Sparse FC

Ưu điểm	Nhược điểm
- Giúp tăng tốc khả năng dự đoán của mô hình, do giảm được MACs - Khả năng dự đoán tốt hơn so với một số các phương pháp tiên tiến khác	- Chậm hơn so với I-AutoRec(mô hình cơ bản) một xíu trong quá trình học, do có nhiều tham số hơm

II. Cài đặt thuật toán trên Google Colab

Source code từ https://github.com/lorenzMuller/kernelNet_MovieLens

1. Load data
   - Input:
     path: đường dẫn tới tệp dữ liệu
     valfrac: tỷ lệ để phân chia tập validation(0.1 = 10% validation, 90% training)
     delimeter: các ký hiệu phân chia dữ liệu trong một hàng (“::”, “,”, ”:”….)
     seed: Lưu lại kết quả cho lần random (seed = 1234)
     transpose: Chuyển đổi ma trận đầu ra, từ user sang item
   - Output:
     train ratings, valid ratings: ma trận có các cột là user, các hàng là item, và trọng số là các ratings
2. Setup kernel
   kernel: Setup kernel theo công thức với u là input vector, v là output vector

$$K_{\text{fs}}(\vec{u}_i, \vec{v}_j) = \max\left(0, 1 - a \cdot D(\vec{u}_i, \vec{v}_j)\right)$$

   tf.maximum(0. , 1. – (tf.norm(u – v, ord = 2, axis = 2))**2)
   kernel layer:
   - Input:

   x: input[batch, channels]
   n_hid: số nodes ẩn
   n_dim: số chiều để áp kernel vào
   activation: output activation
   - Output:
     layer output, regularization term (đại lượng điều chỉnh)

3. Setup model
   Áp kernel vào từng layer
   Tính toán độ lỗi
   Cài hàm tối ưu hoá L-BFGS

4. Training and testing

   n_epochs = n_layers * 10

   Trả ra kết quả RMSE của validation, RMSE của training, thời gian chạy trong file summary

Lưu ý: Mô hình chỉ chạy được trên tensorflow 1.14.0 hoặc bé hơn

Cài đặt trên google colab

Link
https://colab.research.google.com/drive/1hQxJbPpb8cufvFesHxzfX37A4Ks47MJp#scrollTo=GbQMO1BJa4o0

Clone source code về

```
!git clone https://github.com/lorenzMuller/kernelNet_MovieLens.git
```

```
Cloning into 'kernelNet_MovieLens'...
remote: Enumerating objects: 28, done.
remote: Total 28 (delta 0), reused 0 (delta 0), pack-reused 28
Unpacking objects: 100% (28/28), done.
```

Cài lại môi trường tensorflow

```
!pip uninstall -y tensorflow-gpu tensorflow tensorflow-base
!pip install tensorflow-gpu==1.14
!pip install tensorflow==1.14.0
```

Tải tập dữ liệu MovieLens 1M về và giải nén

```
!wget --output-document=ml-1m.zip http://www.grouplens.org/system/files/ml-1m.zip; unzip ml-1m.zip
```

```
--2021-05-01 16:58:08--  http://www.grouplens.org/system/files/ml-1m.zip
Resolving www.grouplens.org (www.grouplens.org)... 128.101.34.235
Connecting to www.grouplens.org (www.grouplens.org)|128.101.34.235|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://grouplens.org/system/files/ml-1m.zip [following]
--2021-05-01 16:58:08--  https://grouplens.org/system/files/ml-1m.zip
Resolving grouplens.org (grouplens.org)... 128.101.34.235
```

Đổi path cho phù hợp

```
# load data
tr, vr = loadData('/content/ml-1m/ratings.dat', delimiter='::',
                  seed=seed, transpose=True, valfrac=0.1)

tm = np.greater(tr, 1e-12).astype('float32')  # masks indicating non-zero entries
vm = np.greater(vr, 1e-12).astype('float32')
```

Chạy thử

```
%cd /content/kernelNet_MovieLens/
```

```
/content/kernelNet_MovieLens
```

```
!python3 kernelNet_ml1m.py
```

III. Chạy thử nghiệm trên tập movie-lens 1M.

- Kết quả chạy 20 epochs, với tập training 90%, validation 10%
- RMSE validation = 0.8293059
- RMSE training = 0.6832043
- Đánh giá kết quả tương đối cao

IV. Cài đặt thuật toán baseline của apache-spark và một "SOTA" khác trên tập MovieLens 1M.

- Nhóm sẽ sử dụng phương pháp I-Autorec là tiền thân của SparseFC qua đó có thể dễ dàng so sánh về mặt hiệu năng giữa 2 phương pháp có sự tiếp nối với nhau.

Dataset:

- Tập dữ liệu MovieLens 1M được chia thành 2 tập train và test theo tỉ lệ 80:20.
- Phương pháp baseline nhóm chọn ALS được hỗ trợ bởi bộ thư viện pyspark.

Algorithm	Dataset	RMSE
Baseline	Movilens 1M	0.8803
I-Autorec	Movilens 1M	0.8486

⇨ Có thể nhận thấy I-Autorec có sự cải tiến hơn so với thuật toán mặc định ALS của pyspark.

V. Ứng dụng vào một bộ dữ liệu khác

➢ Nhóm chọn tập MovieLens-10M nhưng colab không thể load data lên vì giới hạn RAM, nên nhóm đã chọn tập MovieLens-100k. Tuy số record nhỏ hơn nhưng khác biệt về mặt độ dày của ma trận item-user.

- MovieLens-100k
  - Số rating thu thập được: 100.000
  - Số user: 1000
  - Số item (phim): 1700

⇨ Số rating nếu được lắp đầy: 1000*1700=1700000
⇨ Tỷ lệ bao phủ của bộ dữ liệu: (100000/17000000)*100 = 5.88%

- MovieLens – 1M:
  - Số rating thu thập được: 1.000.000
  - Số user: 6000
  - Số item (phim): 4000

⇨ Số rating nếu được lắp đầy: 6000*4000=24.000.000
⇨ Tỷ lệ bao phủ của bộ dữ liệu: (1000000/24000000)*100 = 4.17%

⇨ Tập 100k thưa hơn nhiều so với tập 1M. Nhưng có một yếu điểm ở chỗ số user nhỏ hơn số item nên việc thưa hơn là dễ hiểu nhưng vẫn chiếm tỉ lệ cao hơn so với 1M tức tập 1M càng thưa hơn so với con số đang hiển thị.

Kết quả thực nghiệm:

Phương pháp	ML-1M	ML-100k
Baseline (ALS)	0.8803	1.0019
I-Autorec	0.839	0.899
SparseFC	0.826	0.891

Từ bảng trên ta nhận ra kết luận như sau:

- Với cả 3 phương pháp thì tập dữ liệu lớn hơn với nhiều thông tin hơn nhưng thưa hơn thì tốt hơn so với tập tập dữ liệu nhỏ và ít thông tin.

- Ta thấy sự cải thiện rõ rệt của Sparse FC so với I-Autorec phiên bản cũ trước đó nhờ vào việc cải tiến thêm kernel.

- Do tập dữ liệu ML-100k nhóm chọn chưa thực sự chuẩn xác về mặt logic nên ảnh hưởng khá nhiều đến việc so sánh hiệu năng giữa các tập dữ liệu khác nhau trên cùng một phương pháp.

Source code:

https://drive.google.com/drive/folders/1JBvWNNqnf9B1WNEU2iKVRdLz0tHW-i0w?usp=sharing

Tài liệu tham khảo:

- Kernelized Synaptic Weight Matrices — Lorenz K. Muller, Julien N.P. Martel, Giacomo Indiveri
- AutoRec: Autoencoders Meet Collaborative Filtering — Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, Lexing Xie

- **lorenzMuller/kernelNet_MovieLens (github.com)**
- Datasets: MovieLens | GroupLens