به نام خد ا

گزارش پروژه سیستم پیشنهاد دهنده

فواد عصاره

استاد: دکتر مرجان نادران طحان

فهرست مطالب

1 ریکامندر سیستم چیست ۱

2 کتابخانه های مورد نیاز ۱

3 محیط کار..... ۳

4 نحوه پیاده سازی به صورت کلی ۵

5 دیتاست دیجیکالا و پیش پردازش اولیه داده ها ۵

6 ساخت قسمت های مدل candidate ۷

7 به هم چسباندن لایه های محصول ۹

8 پیاده سازی عملیات retrieval و انتخاب متریک ارزیابی ۱۰

9 کامپایل کردن مدل ۱۱

10 اسکن بردار های تعبیه ۱۲

11 ساخت مدل ranking ۱۳

12 تابع هزینه مدل ranking ۱٤

13 آزمایش مدل ها ۱۵

۱ ریکامندر سیستم چیست

فروشگاه های آنلاین معمولاً میلیون‌ها آیتم مانند محصولات خانگی، فیلم‌ها، تلفن های هوشمند ، و غیره را ارائه می‌دهند. کاربران ترجیح می‌دهند که یک لیست کوتاه از آیتم‌های محتمل را ببینند تا اینکه با تمام محتواها سر و کار داشته باشند. آن‌ها معمولاً می‌توانند با جستجو یا فیلتر کردن لیست، بهترین آیتم‌ها را پیدا کنند. یک سیستم توصیه‌گر می‌تواند بهترین نتایج شخصی‌سازی شده را برای کاربر جستجو، فیلتر و پیشنهاد دهد - نتایجی که احتمالاً کاربر تمایل به خرید آن‌ها دارد. این سیستم به شدت به تجربه کاربر کمک می کند و باعث میشود تا او بهتر با سیستم ارتباط برقرار کند. یک سیستم پیشنهاد دهنده می‌تواند شناسه کاربر، موقعیت جغرافیایی کاربر یا تاریخچه خریدهای قبلی کاربر را دریافت کند و گزینه‌های پیشنهادی می‌توانند برخی آیتم‌ها باشند که حدس می‌زنیم برای کاربر جالب خواهند بود.

۲ کتابخانه های مورد نیاز

۱. tensorflow

- **TensorFlow**یک کتابخانه متن‌باز برای محاسبات عددی و یادگیری ماشین است که توسط تیم Google Brainتوسعه داده شده است. این کتابخانه به‌ویژه در ساخت و اجرای مدل‌های یادگیری عمیق (Deep Learning) کاربرد دارد.
- مدل‌های پیچیده شبکه‌های عصبی را به سادگی پیاده‌سازی و آموزش می دهد.

ویژگی‌ها و کاربردها

- **تعریف و آموزش مدل‌ها** :ابزارهایی برای ساخت و آموزش مدل‌های یادگیری ماشین، به‌ویژه شبکه‌های عصبی مصنوعی، فراهم می‌کند.
- **محاسبات توزیع‌شده** :قابلیت اجرا روی CPU و GPU را دارد و می‌تواند محاسبات توزیع‌شده را انجام دهد.
- **قابلیت‌های پیشرفته** :این کتابخانه امکاناتی برای یادگیری تقویتی(Reinforcement Learning) ، یادگیری انتقالی (Transfer Learning) و یادگیری بدون نظارت (Unsupervised Learning) فراهم می‌کند.

pandas . ۲

- **Pandas**یک کتابخانه قدرتمند و کارآمد برای تحلیل داده‌ها در زبان برنامه‌نویسی پایتون است. این کتابخانه به‌ویژه برای کار با داده‌های ساختاریافته (مانند جداول و داده‌های سری زمانی) مناسب است.
- Pandasقابلیت‌های متعددی برای دستکاری، پردازش، و تحلیل داده‌ها فراهم می‌کند.

ویژگی‌ها و کاربردها

- **DataFrame** یکی از مهم‌ترین ویژگی‌هایPandas ، ساختار داده‌ای DataFrame است که به شما اجازه می‌دهد با داده‌ها به صورت جداول کار کنید.
- **پردازش داده‌ها** Pandas :ابزارهای متنوعی برای فیلتر کردن، انتخاب، گروه‌بندی و تبدیل داده‌ها دارد.
- **ادغام داده‌ها** :می‌توانید چندین مجموعه داده را با استفاده از ابزارهای پیوست (join) و ادغام (merge)در Pandas به یکدیگر متصل کنید.

keras .۳

- **Keras** یک کتابخانه سطح بالا برای یادگیری عمیق است که بر روی TensorFlow اجرا می‌شود. این کتابخانه به شما امکان می‌دهد تا مدل‌های شبکه عصبی پیچیده را با کد ساده‌تر و سریع‌تر توسعه دهید. به عنوان یک ماژول درون TensorFlow موجود است، بنابراین نیازی به نصب جداگانه ندارد.

hazm .٤

کتابخانه **Hazm**یکی از ابزارهای پرکاربرد پردازش زبان طبیعی (NLP) برای زبان فارسی است که در پایتون توسعه یافته است. این کتابخانه با هدف تسهیل پردازش متن‌های فارسی و فراهم کردن ابزارهایی برای تحلیل و پردازش متون فارسی به وجود آمده است .

Hazmامکانات متعددی برای پردازش متن‌های فارسی فراهم می‌کند، از جمله توکنیزاسیون (Tokenization)، که به معنای جداسازی کلمات در یک جمله است، و همچنین ریشه‌یابی (Stemming) و لِماتیزاسیون (Lemmatization) که به ترتیب به فرآیند ساده‌سازی کلمات به شکل ریشه‌ای و پایه‌ای‌ترین فرم ممکن اشاره دارند. این کتابخانه همچنین قابلیت‌هایی برای نرمال‌سازی (Normalization) متون

فارسی ارائه می‌دهد که شامل تصحیح نیم‌فاصله‌ها، تبدیل حروف مختلف مانند "ي" به "ی"، و حذف علائم نگارشی زائد است.

یکی دیگر از ویژگی‌هایHazm ، توانایی شناسایی و جداسازی جملات (Sentence Tokenization) و همچنین برچسب‌گذاری اجزای کلام (POS Tagging) است. این ابزارها برای تحلیل ساختار گرامری جملات فارسی بسیار مفید هستند. علاوه بر این، Hazmامکاناتی برای تبدیل متون فارسی به شکل استاندارد و همچنین حذف پیشوندها و پسوندها از کلمات فراهم می‌کند.

به دلیل متن‌باز بودن Hazm به یکی از انتخاب‌های اصلی برای توسعه‌دهندگان و محققانی تبدیل شده است که با متون فارسی سر و کار دارند. استفاده از Hazm در پروژه‌های مختلف باعث کاهش پیچیدگی‌های مرتبط با پردازش زبان فارسی می‌شود.

۳ محیط کار

Google Colab یک محیط ابری رایگان برای اجرای کدهای پایتون است که به طور خاص برای یادگیری ماشین، علم داده و محاسبات عددی طراحی شده است Google Colab .به کاربران این امکان را می‌دهد تا بدون نیاز به نصب نرم‌افزارهای پیچیده، مستقیماً در مرورگر وب کد بنویسند، اجرا کنند و از قابلیت‌های پیشرفته‌ای مانند GPU و TPU برای افزایش سرعت محاسبات بهره‌مند شوند.

ویژگی‌ها و مزایای Google Colab

۱. **رایگان بودن:**
   - Google Colabیک سرویس رایگان است که توسط Google ارائه می‌شود و برای استفاده از آن نیازی به خرید یا نصب هیچ نرم‌افزاری نیست. کافی است یک حساب کاربری Google داشته باشید.

۲. **دسترسی به GPU :**
   - یکی از بزرگ‌ترین مزایای Google Colab این است که به کاربران اجازه می‌دهد به رایگان از پردازنده‌های گرافیکی(GPU) استفاده کنند. این ویژگی‌ها برای اجرای مدل‌های یادگیری عمیق و محاسبات سنگین بسیار مفید است.

۳. **محیط توسعه یکپارچه در مرورگر:**

- Google Colabیک محیط توسعه آنلاین است که در مرورگر وب اجرا می‌شود. شما می‌توانید کدهای خود را بنویسید، اجرا کنید، نتایج را مشاهده کنید و همچنین توضیحات متنی و تصاویر را در کنار کدهای خود قرار دهید. این قابلیت شبیه به Jupyter Notebookاست.

٤. **پشتیبانی از پایتون و کتابخانه‌های محبوب:**

- Google Colabاز زبان برنامه‌نویسی پایتون و تمامی کتابخانه‌های محبوب علم داده و یادگیری ماشین مانندTensorFlow ، Keras، PyTorch، Pandas، NumPyو Matplotlibپشتیبانی می‌کند.

٥. **اشتراک‌گذاری آسان:**

- امکان اشتراک گذاری نوت‌بوک‌های Google Colab خود را به راحتی با دیگران به اشتراک بگذارید. همچنین قابلیت همکاری چند نفره) مانند (Google Docs وجود دارد که افراد می‌توانند هم‌زمان روی یک نوت‌بوک کار کنند.

٦. **اتصال به:Google Drive**

- Google Colabبه راحتی با Google Drive یکپارچه می‌شود. شما می‌توانید فایل‌ها و داده‌های خود را در Google Drive ذخیره کرده و از آن‌ها در نوت‌بوک‌های خود استفاده کنید.

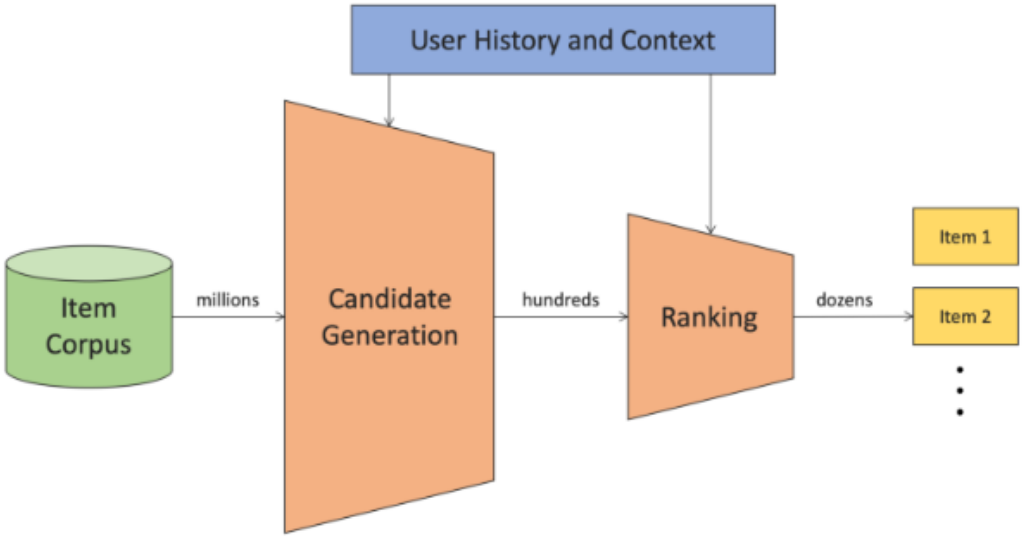
٧. **ذخیره خودکار و نسخه‌بندی:**

- تمام تغییرات شما در نوت‌بوک‌های Google Colab به طور خودکار ذخیره می‌شود و می‌توانید نسخه‌های قبلی نوت‌بوک خود را مشاهده و بازیابی کنید.

٤ نحوه پیاده سازی به صورت کلی

در این روش پیاده سازی سیستم پیشنهاد دهنده را به دو بخش تقسیم کرده ایم که هر کدام از این دو بخش نیز از قسمت های مختلفی تشکیل شده که در ادامه توضیح داده می شوند.

در بخش اول سعی می شود تا از بین تمام آیتم های موجود در دیتاست تعدادی از ایتم ها که کاربر بیشترین علاقه رو به آن ها دارد استخراج کنیم. خروجی این بخش به مدل دوم وارد میشود که فاز رنکینگ نام دارد. در این قسمت آیتم های فیلتر شده امتیازدهی میشوند تا از بین آن ها بیشتر علاقه کاربر مشخص شود. ورودی اولیه این سیستم کل آیتم ها هستند و بر اساس سوابق کاربران سعی می شود تا مناسب ترین ها در هر بخش انتخاب شوند.



۵ دیتاست دیجیکالا و پیش پردازش اولیه داده ها

این دیتاست که از دو فایل کامنت ها و محصولات تشکیل شده است اطلاعاتی را به ما می دهد تا بتوانیم محصولات مورد علاقه هر کاربر را به او نشان دهیم. فعلا در این مرحله از قسمت نظرات استفاده می کنیم و فیلد هایی که بنظر موردنیاز هستند را برمیداریم.برای اینکار دیتاست را به صورت یک دیتا فریم با کتابخانه pandas باز میکنیم.

فیلد هایی که برداشته شده به ترتیب شناسه محصول، شناسه کاربر، اسم محصول و نظر کلی کاربر در خصوص محصول می باشد. اکنون میتوانیم پیش پردازش این دیتا را شروع کنیم که در چند بخش انجام می شود:

مرحله اول: به دنبال مقادیر تعریف نشده می گردیم و آن ها را با یک رشته خالی پر می کنیم

مرحله دوم: با بررسی دیتا و فیلد نظر کاربر میتوان متوجه شد که بی طرفی کاربر نسبت به محصول با دو رشته متفاوت ذخیره شده که چون معنی آن یکسان است تمام نظرات بی طرف را به no_idea تغییر دادیم:

```
df = df.fillna({'recommend': '', 'product_title': ''})

df['recommend'] = df['recommend'].replace('\\N', 'no_idea')
```

مرحله سوم: نظر کاربران را به یک امتیاز مپ می کنیم تا پردازش آن به صورت عددی آسان تر باشد:

```
recommend_mapping = {
    'not_recommended': 0,
    'no_idea': 3,
    'recommended': 5
}

df['recommend'] = df['recommend'].map(recommend_mapping)
```

به این صورت که برای هر مورد یک عدد در نظر گرفته شده و در دیتا فریم جایگزین می شود.

مرحله چهارم: در آخرین مرحله باید نام محصولات پیش پردازش شود که برای اینکار در ابتدا با مشاهده دیتابیس متوجه میشویم که برخی کلمات که در نام محصولات وجود دارند کمکی به پیش بینی نکرده و تکرار زیادشان باعث overfit شدن مدل می شود پس اقدام به پیدا کردن و حذف کلمات این چنینی مثل "سری" ,"به" ,"و" ,"ای" ,"عددی" ,"مدل" ,"کد" می کنیم. بعد از اینکار از کتابخانه hazm که برای پردازش زبان فارسی ساخته شده است استفاده می کنیم تا عملیات هایی نرمال سازی را بر روی متن ما انجام دهد. این کتابخانه قادر به انجام عملیات های مختلفی می باشد مثل نرمال سازی (حذف اعراب و تبدیل فاصله ها به نیم فاصله)، بازگرداندن کلمات به ریشه آن ها و در نهایت جدا سازی یا tokenize کردن آن ها.

بعد از پیش پردازش دیتا اکنون باید دیتاست را از حالت دیتا فریم به tensor تبدیل کنیم تا بتوانیم در ساخت مدل ها از دیتا استفاده کنیم. در آخر با دریافت طول دیتاست در میابیم که ۱۰۰۰۰۰ نمونه در آن وجود دارد.

در آخرین بخش کار با دیتا آن را به دو قسمت آموزش و تست تبدیل می کنیم به این صورت که ۸۰ درصد دیتا را به آموزش و ۲۰ آن را به تست اختصاص می دهیم. نکته مهمی که میتوان به آن اشاره کرد استفاده از تابع shuffled می باشد تا دیتا بر بخورد و سختی دیتا در آموزش و تست بهتر تقسیم شود. به طور مثال ممکن است دیتایی که در سطر های اول قرار می گیرید شامل اطلاعات واضح تری از دیتا های آخر باشد و اگر دیتا به ترتیب تقسیم شود ممکن است مدل خوب آموزش پیدا نکند. سایز نهایی بعد از تقسیم بندی:

```
ratings_trainset size: 80000
ratings_testset size: 20000
```

۶ ساخت قسمت های مدل candidate

به طور خلاصه کار این مدل به این صورت است که اطلاعات کاربر و اطلاعات محصولات می گیرد و سعی می کند تا شباهت با انجام پردازش هایی شبیه ترین محصولات و کاربران مشخص کند. اما برای اینکار نیاز به یکسری عملیات های اضافه تر بر روی داده داریم. ابتدا باید داده هارا به صورت مورد انتظار مدل نرمال کنیم و سپس بردار های امبدینگ دیتا رو استخراج کنیم. در نهایت مدل با یادگیری شباهت ها بردار ها را وزن دهی می کند به طوری که محصولات و کاربران مشابه بردار های شبیه به هم داشته باشند.

برای نرمال سازی داده ها داده های ما دارای سه نوع numerical, categorical, textual هستند

داده numerical به داده هایی گفته می شود که به نوعی دارای ترتیب مشخصی هستند. داده recommend ما که در ابتدا به صورت رشته بود و آن را به اعداد مپ کردیم در این دسته قرار میگیرد برای انجام این کار از یک لایه tensorflow استفاده می کنیم که نرمال سازی z-score را برای ما انجام میدهد.

فرمول این نرمال سازی به این صورت است : $z= \frac{X-\mu}{\sigma}$ که μ میانگین و σ انحراف معیار می باشد. دلیل انجام این کار این است که تمام داده های ما در یک اسکیل مشخص باشند.

```
recommend_normalization_layer = tf.keras.layers.Normalization(axis=None)

recommend_normalization_layer.adapt(
    ratings_trainset.map(
        lambda x: x['recommend']
    )
)
```

حالا به سراغ نوع داده های categorical می رویم این داده ها ترتیب مشخصی ندارند مانند جنسیت کاربر. در دیتای ما شناسه محصولات و شناسه کاربر جز این دسته به شمار می رود. با استفاده از tensorflow یک لایه می سازیم که این داده ها را ایندکس گذاری کند. به طور مثال اگر کابر با شناسه ۲۰۵ را بخواهیم این لایه عدد ۶ را که شماره ایندکس آن است بر میگرداند.

```
user_id_lookup_layer = tf.keras.layers.IntegerLookup(mask_token=None)


user_id_lookup_layer.adapt(
    ratings_trainset.map(
        lambda x: x['user_id']
    )
)

print(
    f"Vocabulary[:10] -> {user_id_lookup_layer.get_vocabulary()[:10]}"
)
```

برای شناسه فیلم هم دقیقا همینکار را انجام میدهیم و یک لایه برای ایندکس کردن آن میسازیم. به علت اینکه دیتای ما از نوع عددی است است از لایه IntegerLookup استفاده کرده ایم.

برای داده های textual نیز همین روند باید طی شود اما با این تفاوت که داده های متنی باید توکن بندی شوند و به هر کلمه یک ایندکس اختصاص داده شود. به دلیل اینکه ورودی این لایه باید متن باشد توکن های hazm را به هم چسباندیم تا این عمل توسط لایه text انجام شده بود. البته چون از قبل توکن بندی داشتیم از لایه StringLookup هم می شد استفاده کرد.

اکنون داده های برای ساخت بردار های embedding آماده هستند. اینکار نیز به سادگی و توسط یک لایه دیگر انجام می شود. این لایه تنها دو ورودی مورد نیاز است که یکی دیتای اماده شده در مرحله قبل و اندازه هر بردار است که ما مقدار ۳۲ را برای آن در نظر گرفتیم. به این معنا که هر ایندکس به یک بردار که با ۳۲ عدد نشان داده میشود تبدیل شود.

```
product_id_embedding_layer = tf.keras.layers.Embedding(
    input_dim=product_id_lookup_layer.vocabulary_size(),
    output_dim=product_id_embedding_dim
)
```

بعد ساخت بردار برای دیتا ها، اکنون نوبت به آن رسیده تا برای لایه های مربوط به هر ستون از داده را به هم بچسبانیم. با اینکار سعی داریم یک مدل end to end ایجاد کنیم که به دریافت داده های خام را دریافت کند بعد یک خروجی مناسب برای ورودی مدل قابل آموزش به ما ارائه دهد. برای اینکار از یک تابع به نام sequential که در tensorflow و keras قرار دارد استفاده می کنیم و با دادن لایه های آماده شده به آن یک مدل کامل خروجی می گیریم.

```
product_title_model = tf.keras.Sequential(
    [
        product_title_vectorization_layer,
        product_title_embedding_layer,
        tf.keras.layers.GlobalAveragePooling1D()
    ]
)
```

اما باید به این نکته توجه داشت که برای عنوان محصولات باید یک لایه اضافه تر که در کراس وجود دارد استفاده کرد. دلیل آن این است که هر جمله از چند کلمه تشکیل شده است و هر کلمه یک بردار دارد پس این بردار برای هر جمله میانگین کلمات را محاسبه می کند.

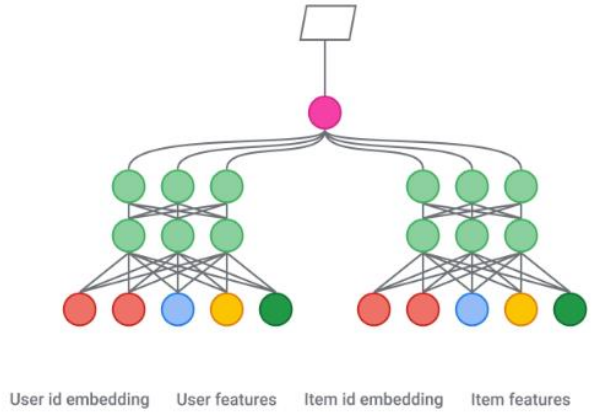
۷ به هم چسباندن لایه های محصول

همچنان یک مشکل دیگر وجود دارد و آن هم وجود دو مدل مختلف شناسه و عنوان محصول است. برای تکمیل برج محصول این دو مدل را هم به هم وصل کنیم که باز هم از توابع کراس کمک می گیریم. خروجی تابع مدل کراس باید از یک لایه dense شبکه عصبی بگذرد که ما برای آن به طور پیش فرض ۳۲ نورون در نظر گرفته ایم و تابع فعال ساز relu را انتخاب کرده ایم. لایه dense به این معنی است که خروجی هر نورون به ورودی نورون بعدی متصل است و در هر نورن تابع فعال ساز چک می کند ورودی مثبت باشد و در این صورت اجازه انتقال دیتا به لایه بعد را می دهد.

```
output = tf.keras.layers.Dense(32, activation='relu')(concatenated_embeddings)

combined_model = tf.keras.Model(
    inputs=[product_id_input, product_title_input],
    outputs=output
)
```

خروجی این مدل یک بردار ۳۲ تایی است که ترکیبی از ویژگی های محصول است. اکنون هر دو برج کاربر و محصول آماده هستند و میتوانیم مدل اصلی را به صورت زیر بسازیم



۸ پیاده سازی عملیات retrieval و انتخاب متریک ارزیابی

برای اینکه بدانیم کاربر به چه فیلم هایی علاقه دارد یک فرض پایه در نظر می گیریم که محصولاتی که کاربر در مورد آن ها نظری ثبت کرده می تواند مورد علاقه او باشد. در این قسمت نیاز داریم که یک کلاس بندی انجام دهیم و برای هر کاربر فیلم هایی که دیده و ندیده را در دو کلاس مجزا قرار دهیم. به عبارت دیگر یک دیتای ضمنی به دست می آوریم. برای اینکار به مثال های منفی هم نیاز داریم که برای این منظور از دیتای تمام محصولات استفاده می کنیم.

برای ساخت لایه retrieval به دو پارامتر نیاز داریم. پارامتر اول یک متریک است که کیفیت پیشنهاد ها را اعلام می کند. متریکی که استفاده کرده ایم factorized top k نام دارد و به این صورت عمل می کند که در k پیشنهاد اول بر اساس فرضی که برای علاقه کاربران داشتیم چند تا از پیشنهاد ها واقعا مورد علاقه کاربر است یعنی کاربر آن هارو دیده و امتیاز ثبت کرده است. به ازای پیشنهاد های درست سیستم امتیاز مثبت می گیرد و برعکس. پارامتر دوم هم دیتاست محصولات است که شامل تمام محصولات سایت است.

```
import tensorflow_recommenders as tfrs

retrieval_task_layer = tfrs.tasks.Retrieval(
    metrics = tfrs.metrics.FactorizedTopK(

    corpus.batch(128).map(candidate_model)
)
)
```

دیتاست ورودی به لایه retrieval باید بخش بندی و یا بچ شود. بچ دیتا هارا دسته بندی میکند که اینجا ما دیتا را به دسته های ۱۲۸ تایی تقسیم کردیم. همچنین دیتا را از لایه کاندید ها که قبلا ساخته ایم رد می کنیم تا بردار های امبدینگ را به دست بیاوریم. این لایه به عنوان ورودی امبدینگ کاربر و محصول را می گیرید و بر اساس شباهت آن ها بردار ها را وزن دهی می کند. همچنین از روی متریک، loss دیتا را حساب می کند که هدف مدل ما کاهش هزینه یا خطا است. برای این مدل از تابع هزینه in-batch sfotmax استفاده کرده ایم. که فرمول آن به صورت زیر است و بر روی ماتریس شباهت که شباهت جفتی بین تمامی زوج‌هایembedding ها در batch را محاسبه کنید اعمال می شود. محاسبه شباهت جفتی بین embedding ها معمولاً با استفاده از یکی از معیارهای شباهت یا فاصله انجام می‌شود. رایج‌ترین معیارهای شباهت برای این کار ضرب داخلی است.

$$\left(\frac{\exp(S_{ii})}{\exp(S_{ik}) \sum_{k=1}^{N}} \right) \log - = \log(P_{ii}) - = {}_{i}\text{Loss}$$

در نهایت میزان هزینه نهایی با میانگین گرفتن از هزینه تمام نمونه ها به دست می آید.

۹ کامپایل کردن مدل

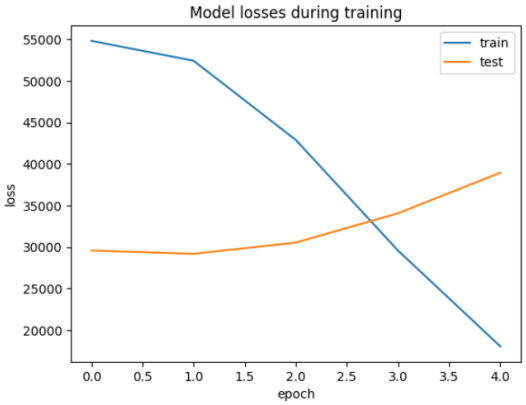
اکنون برای کامپایل کردن مدل یک اپتیمایزر باید انتخاب کنیم. اپتیمایزر از یک نقطه تصادفی شروع می کند و مبتنی بر مشتق و گرادیان در یک جهت حرکت می کند. یک پارامتر در اپتیمایزر ها میزان تغییرات آن است که باید تعیین کنیم در هر مرحله میزان حرکت چقدر باشد. ما از الگوریتم بهینه سازی adagrad استفاده کردیم که مبتنی بر فاصله گرادیان تصادفی است. سپس دیتای تست و ترین را به مدل می دهیم و باید انتخاب کنیم چند بار دیتا را به مدل نشان دهیم که به آن epoch می گویند و ما انتخاب کردیم که ۵ بار دیتا را به مدل نشان دهیم. Validation_freq مشخص می کند که دیتای ترین ما چه مدت یک بار بررسی شود و هزینه آن حساب شود که با قرار دادن مقدار ۱ بعد از هر epoch هزینه دیتای تست هم محاسبه می شود.

```
history = digikala_retrieval_model.fit(
    retrieval_trainset,
    validation_data=retrieval_testset,
    validation_freq=1,
    epochs=5
)
```

مدل در ۵ مرحله آموزش داده می شود:

```
Epoch 1/5
10/10 [==============================]
Epoch 2/5
10/10 [==============================]
Epoch 3/5
10/10 [==============================]
Epoch 4/5
10/10 [==============================]
Epoch 5/5
10/10 [==============================]
```

بعد از آموزش با استفاده از کتابخانه matplotlib نمودار هزینه را رسم می کنیم



همانطور که می بینیم مقدار هزینه همواره در حال کاهش است اما بعد از مرحله سوم دیتای ترین مقدار هزینه اش افزایشی می شود و به ما نشان می دهد که دیتا را بیش از حد به مدل نشان داده ایم و بهتر بود در مرحله سوم آموزش را متوقف می کردیم.

۱۰ اسکن بردار های تعبیه

بعد از آموزش مدل انتظار داریم که بردار های کاربران و محصولات مورد علاقه آن ها به درستی عمل کند. برای پیدا کردن بردار های متناظر هر کاربر از دو روش استفاده کرده ایم.

روش اول استفاده از از تکنیک bruteforce می باشد. با دریافت کل محصولات و مدل کاندید ها که اکنون آموزش دیده است برای ساخت بردار های محصول، برای هر محصول بردار متناظرش را ایندکس می کند تا یک رابطه بین محصول و بردار ایجاد شود و فرآیند جستو و جو را سریع تر کند. ایراد این روش

این است که اگر در دنیای واقعی تعداد بردار های محصول زیاد باشند زمان زیادی طول خواهد کشید تا تمام بردار های محصول بررسی شوند و مشابه ها برگردانده شوند. برای حل این مشکل ابزار های زیادی توسط شرکت ها به وجود آمدند که با استفاده از روش های تقریبی میتوانند با صرف زمان کمتری مشابهت ها را پیدا کنند. یکی از این ابزار ها ScaNN می باشد که توسط گوگل توسعه یافته است و با بردن بردار ها به یک فضای دیگر زمان را کاهش می دهد.

روش دوم استفاده از ScaNN می باشد که باعث سرعت بیشتر اما دقت کمتر در پیدا کردن بردار ها می شود و برای دیتای با حجم زیاد مناسب است

```
scann_layer = tfrs.layers.factorized_top_k.ScaNN(
   digikala_retrieval_model.query_model
)

scann_layer.index_from_dataset(
  tf.data.Dataset.zip(
      (
          corpus.batch(128),
          corpus.batch(128).map(
              digikala_retrieval_model.candidate_model
          )
      )
  )
)
```

بعد از پیدا کردن کاندید های مناسب باید به سراغ ساخت مدل ranking برویم تا بهترین کاندید ها را بر اساس رای کاربران پیدا کنیم.

۱۱ ساخت مدل ranking

در این مدل با استفاده از یک شبکه عصبی ساده سعی می کنیم تا با استفاده از دیتای اولیه که برای هر کار و میزان علاقه اش داریم نمره هایی پیش بینی کنیم. مسئله ما اینجا از نوع رگرسیون می باشد.
ساخت شبکه عصبی ما به صورت زیر است:

اول داده به صورت یک مجموعه‌ای از بردارهای ورودی وارد مدل می‌شوند که هر بردار معمولاً یک مجموعه ویژگی‌ها را نمایش می‌دهد (مثلاً ویژگی‌های یک کاربر یا آیتم).

لایه Dense اول(Dense(256, activation='relu'))

- این لایه یک لایه کاملاً متصل (Dense) است که شامل ۲۵۶ نورون می‌شود.
- **وزن‌ها و بایاس** :این لایه دارای یک ماتریس وزن Wبا ابعاد (input_dim, 256)و یک بردار بایاس bبا ابعاد (256,)است. بردار ورودی در Wضرب می‌شود و سپس bبه آن اضافه می‌شود.

- **فعال‌سازی (ReLU)**: پس از این تبدیل خطی، تابع فعال‌سازی ReLU اعمال می‌شود ReLU . تمامی مقادیر منفی را به صفر تبدیل می‌کند .
- **خروجی**: خروجی این لایه یک بردار با ابعاد (256)است که هر عنصر آن نمایانگر خروجی فعال‌شده یکی از نورون‌ها است.

لایه Dense دوم(Dense(64, activation='relu'))

- **تبدیل**: خروجی لایه اول که یک بردار با اندازه‌ی 256است وارد این لایه می‌شود که دارای ٦٤ نورون است.
- **وزن‌ها و بایاس**: این لایه یک ماتریس وزن W یک بردار بایاس b با ابعاد 64دارد.
- **فعال‌سازی (ReLU)**: مجدداً تابع فعال‌سازی ReLU اعمال می‌شود و مقادیر منفی را به صفر تبدیل می‌کند.
- **خروجی**: نتیجه‌ی این لایه یک بردار با ابعاد 64 است.

لایه Dense سوم(Dense(1))

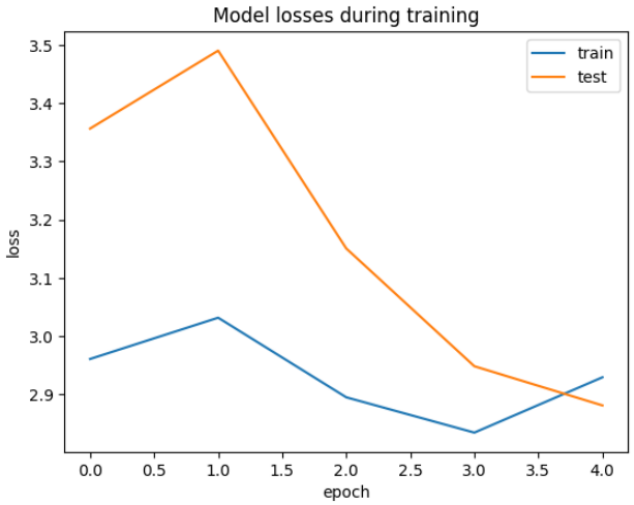
- **تبدیل**: خروجی از لایه دوم که یک بردار با اندازه‌ی 64است وارد این لایه می‌شود که تنها ١ نورون دارد.
- **وزن‌ها و بایاس**: این لایه دارای یک ماتریس وزن Wبا و یک بایاس bبا اندازه‌ی1 است.
- **فعال‌سازی**: در اینجا هیچ تابع فعال‌سازی اعمال نمی‌شود، بنابراین تبدیل خطی باقی می‌ماند.
- **خروجی**: نتیجه این لایه یک مقدار عددی است که خروجی نهایی مدل را نشان می‌دهد.

١٢ تابع هزینه مدل ranking

در این مدل از تابع Mean Squared Error استفاده کرده ایم. تابع MSEاندازه‌ای از اختلاف بین مقادیر پیش‌بینی‌شده توسط مدل و مقادیر واقعی یا هدف را نشان می‌دهد. این تابع به صورت میانگین مربع خطاها تعریف می‌شود، جایی که خطا تفاوت بین مقدار پیش‌بینی‌شده و مقدار واقعی است. به طور

مثال اگر برای محصولی کاربر امتیاز ۴ را داده باشد ولی مدل ما ۲ محاسبه کرده باشد اختلاف این دو را محسابه کرده و به توان ۲ میرسانیم. میتوانستیم از RMSE هم استفاده کنیم که با گرفتن جذر مقدار را کوچک تر می کند.

بعد از آموزش مدل مثل حالت قبل تابع هزینه را رسم می کنیم



که مثل حالت قبل مشاهده می شود که تا دور سوم آموزش هزینه به کمترین مقدار خودش رسیده و آموزش باید در این دور متوقف شود.

۱۳ آزمایش مدل ها

برای آزمایش مدل به عنوان ورودی شناسه یک کاربر از دیتای تست را می دهیم تا محصولات مورد علاقه او را نمایش دهد.

```
, 'وزنه مچ دست و یا 3 کیلوگرمی' , 'صلوات شمار انگشتی کد 02' , 'دستبند مردانه هانی گالری مدل کارد' , '01 د

'دستبند مردانه  هانی گالری مدل Black10', 'ranking_score': 3.897409200668335}
'کیف پول مردانه چرم طبیعی آراد کد 715', 'ranking_score': 3.823603391647339}
'کننده اون مدل سیکالفیت حجم 40 میلی لیترc\u200bکرم ترمیم', 'ranking_score': 3.7816684246063232}
'جاسیگاری و کیف سیگار چرم مدل  M10', 'ranking_score': 3.7719156742095947}
'اتو بخار مدل ICEN I122 2200W', 'ranking_score': 3.7635483741760254}
'دستبند مردانه هانی گالری مدل گارد', 'ranking_score': 3.6971933841705322}
'آبپاش مدل 171 ظرفیت 5 لیتر', 'ranking_score': 3.523710250854492}
'مهر رکعت شمار نماز کد 01', 'ranking_score': 3.490758180618286}
'صلوات شمار انگشتی کد 02', 'ranking_score': 3.4824581146240234}
'وزنه مچ دست و یا 3 کیلوگرمی', 'ranking_score': 3.4590282440185547}
```

مشاده می شود که محصولات با هم ارتباط معنایی دارند و با جست و جوی کاربر در دیتابیس مشاهده می شود که علاقه او به چنین محصولاتی بیشتر بوده است اما تست کردن تنها یک مورد کافی نمی باشد. به همین علت کل دیتاست تست را پیمایش می کنیم و معیار شباهت کسینوسی را برای محصولات پیشنهادی هر کاربر محاسبه می کنیم تا مطمئن شویم محصولات تا حد ممکن مرتبط هستند.

```
Total number of similar items across all items: 86.0
Average number of similar items per item: 8.6
Aggregated result (total similar items): 86.0
Recall: 0.86
8463553
User ID: 8463553
Total number of similar items across all items: 76.0
Average number of similar items per item: 7.6
Aggregated result (total similar items): 76.0
Recall: 0.76
2879084
User ID: 2879084
Total number of similar items across all items: 76.0
Average number of similar items per item: 7.6
Aggregated result (total similar items): 76.0
Recall: 0.76
2953630
User ID: 2953630
Total number of similar items across all items: 78.0
Average number of similar items per item: 7.8
Aggregated result (total similar items): 78.0
Recall: 0.78
5990444
User ID: 5990444
Total number of similar items across all items: 78.0
Average number of similar items per item: 7.8
Aggregated result (total similar items): 78.0
```

مشاهده می شود در تمام ایتم های دیتاست شباهت بالایی میان ایتم ها وجود دارد.

علاوه بر این معیار recall را هم برای دیتا محاسبه کردیم که یکی از معیارهای مهم برای ارزیابی عملکرد مدل‌های یادگیری ماشین، به ویژه در مسائل دسته‌بندی (classification) است. این معیار به شما می‌گوید که مدل شما چقدر در پیدا کردن نمونه‌های مثبت واقعی (True Positives) موفق بوده است. فرمول آن به این صورت است:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Recall: 0.76

Recall: 0.86

....

```python
def calculate_recall(true_positives):


    # Calculate False Negatives
    false_negatives = 10 - true_positives

    # Calculate Recall
    recall = true_positives / (true_positives + false_negatives)



    return recall
```

با این فرمول برای هر کاربر معیار recall را محاسبه کردیم.