دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکدهٔ ریاضی و علوم‌کامپیوتر

گزارش ۶: پیاده‌سازی یک سیستم توصیه‌گر

نگارش
نیما حسینی دشت بیاض

استاد
دکتر مهدی قطعی

اردیبهشت ۱۴۰۰

مقدمه

سیستم‌های توصیه‌گر یکی از انواع سیستم‌های تصمیم‌یار هستند که در سال‌های اخیر بسیار مورد توجه قرار گرفته‌اند. بسیاری از شرکت‌های مختلف برای ارائه‌ی محتوای خود به این سیستم‌ها متکی هستند و در راستای بهبود آن‌ها نیز تلاش زیادی می‌کنند. این باعث شده است که گونه‌های متفاوتی از این سیستم‌ها طراحی و ساخته شوند که هر کدام در شرایطی عملکرد بهتری نشان می‌دهند.

در این تمرین به پیاده‌سازی دو سیستم توصیه‌گر مجزا می‌پردازیم که می‌توانند در کنار هم داخل یک سیستم بزرگ‌تر استفاده شوند. یک سیستم از نوع مبتنی بر محتوا[1] و دیگری مبتنی بر همکاری[2] می‌باشد و هر دو با استفاده از دیتاست MovieLens 10M[3] طراحی شده‌اند. این دیتاست شامل ۱۰ میلیون امتیاز و ۹۵۰۰۰ تگ برای ۱۰۰۰۰ فیلم است که از ۷۱۰۰۰ کاربر به دست آمده است.[1] به دلیل محدودیت‌های سخت‌افزاری، هر کدام از سیستم‌ها با استفاده از بخشی از این دیتاست آموزش دیده‌اند.

در ادامه به جزئیات هر کدام از دو مدل می‌پردازیم و سپس پیاده‌سازی آن‌ها در پایتون و خروجی هر دو را بررسی می‌کنیم.

جزئیات مدل‌ها

توصیه‌گر مبتنی بر محتوا

این نوع سیستم‌ها به‌طور کلی با بررسی ویژگی‌های یک آیتم مانند نوع، دسته‌بندی، نظرات، توضیحات و... سعی می‌کنند تا آیتم مشابه را پیدا کنند. این نوع از سیستم‌ها در در حالت ساده علایق کاربر را در نظر نمی‌گیرند و صرفاً موارد مشابه یک آیتم را پیدا می‌کنند.

برای طراحی یک ریکامندر مبتنی بر محتوا برای پیشنهاد فیلم می‌توان از ویژگی‌هایی مانند ژانر فیلم، بازیگران، کشور محل تولید و... استفاده کرد. دیتاست MovieLens شامل تگ‌هایی بر هر فیلم است که این تگ‌ها را هر کاربر برای برخی فیلم‌ها مشخص کرده است. تگ‌ها می‌توانند اطلاعات خوبی درباره‌ی فیلم‌ها داشته باشند. به طور مثال فیلم‌هایی که دارای تگ Pixar هستند احتمالاً انیمیشن‌هایی هستند که توسط کمپانی Pixar تولید شده است و در سبک شباهت زیادی به هم دارند. سعی می‌کنیم با استاده از این تگ‌ها

۱ Content Based Recommender Systems
۲ Collaborative Recommender Systems
۳ https://grouplens.org/datasets/movielens

سیستم خود را طراحی کنیم.[2]

ابتدا لازم است که از این تگ‌ها اطلاعاتی عددی استخراج کنیم. برای این کار می‌توان از روش TF-IDF استفاده کرد. این روش تمام کلماتی که در تگ‌ها به کار برده شده‌اند را پیدا می‌کند و نسبت دفعات استفاده از یک کلمه برای یک فیلم به تعداد دفعات استفاده از آن کلمه در کل فیلم‌ها را محاسبه می‌کند. درواقع هر کلمه تبدیل به یک feature می‌شود و نسبت استفاده از آن برای هر فیلم به عنوان مقدار این feature ثبت می‌شود.[3] با انجام این کار، یک فضا با ابعاد بالا برای نمایش فیلم‌ها ایجاد می‌کنیم که هر محور آن یکی از کلمات به کار رفته در تگ‌ها است و هر فیلم به عنوان یک بردار در این فضا قرار می‌گیرد. با انتقال فیلم‌ها به چنین فضاهایی می‌توان شباهت فیلم‌ها را با روش‌هایی مانند Cosine Similarity محاسبه کرد. اما این فضا ارتباط بین فیلم‌ها را به خوبی نشان نمی‌دهد. به طور مثال ممکن است دو کلمه‌ی Pixar و Disney به عنوان دو محور در این فضا قرار گرفته باشند. فیلم‌هایی که تگ Pixar را دارند به این محور نزدیک‌تر خواهند بود و فیلم‌های کمپانی Disney فاصله‌ی زیادی از آن‌ها می‌گیرند؛ درحالی‌که هر دو تا حدودی به هم شباهت دارند و انیمیشن هستند.

برای رفع این مشکل لازم است که فیلم‌ها را به یک فضا با ابعاد پایین‌تر منتقل کنیم تا هر بعد فضا به‌جای نشان دادن یک تگ خاص، ویژگی‌های سطح بالاتری از فیلم‌ها را نمایندگی کند. به چنین فضایی یک Latent Space می‌گویند و بردار فیلم‌ها که به این فضا منتقل می‌شوند، Embedding هر فیلم نامیده می‌شوند. در چنین فیلم‌هایی که تگ‌های مشابه به هم دارند نزدیک‌تر به هم قرار می‌گیرند و معنای تگ‌ها بیشتر از خود کلمات اهمیت پیدا می‌کنند.حال می‌توان مشابه آنچه گفته شد، با روش‌های مختلف مشابهت بردارها را حساب کرد. روش Cosine Similarity کاربرد زیادی در این زمینه دارد و با محاسبه‌ی زاویه‌ی بین هر دو بردار، شباهت آن‌ها را حساب می‌کند.

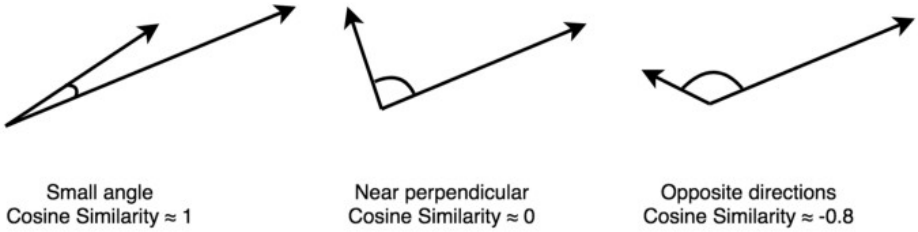
$$sim(A, B) = cos(\theta) = \frac{A \cdot B}{||A||\,||B||}$$

فرمول ۱: فرمول شباهت کسینوسی

با توجه به فرمول فوق، شباهت بین دو بردار یا فیلم می‌تواند عددی بین ۱- تا ۱ باشد. نزدیکی به ۱ نشان دهنده شباهت بالا و نزدیکی به ۱- نشان‌دهنده‌ی تفاوت زیاد دو فیلم و مخالفت آن‌ها می‌باشد. همچنین نزدیکی به صفر نشان می‌دهد که دو بردار بر هم عمودند و ارتباط زیادی بین آن‌ها وجود ندارد.(عکس ۱)

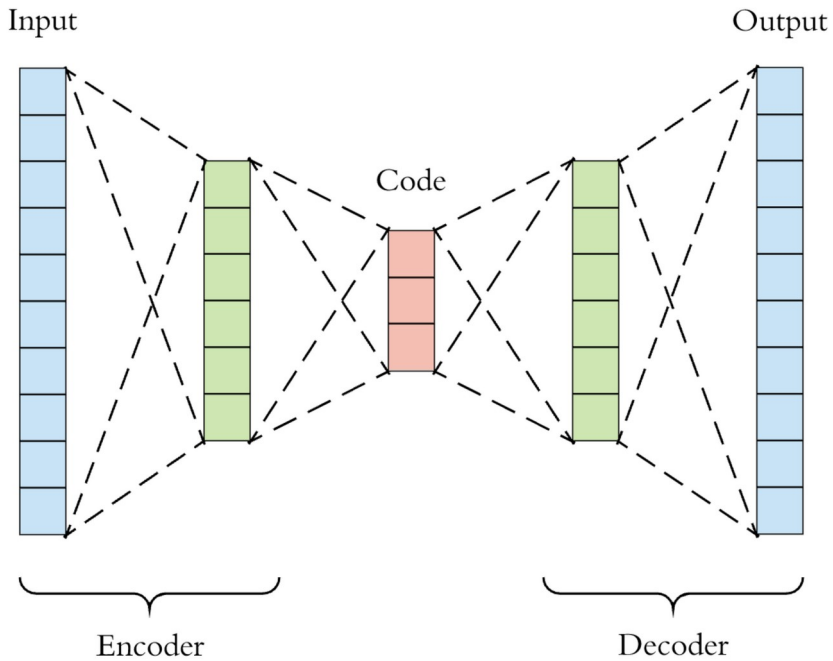
حال که روش یافتن شباهت بین فیلم ها در Latent Space را پیدا کرده‌ایم، مسئله‌ی اصلی یافتن چنین فضایی و انتقال فیلم‌ها به آن فضاست. برای چنین کاری روش‌های متفاوتی وجود دارد که معروف ترین آن استفاده از تجزیه‌های ماتریسی است؛ اما در این مدل ما با استفاده از شبکه‌های عصبی و Auto Encoder ها

از تجزیه‌های ماتریسی بی‌نیاز می‌شویم و همچنین می‌توانیم پارامترهای بسیار مختلفی را در مدل خود کنترل کنیم.



عکس ۱: شباهت بین دو بردار

شبکه‌های Auto-Encoder انواعی از شبکه‌های عصبی هستند با دریافت ورودی با ابعاد بالا، در هر لایه تعداد نورون‌های لایه را کاهش می‌دهند تا در نهایت به لایه‌ای با بعد و تعداد نورون مطلوب برسند. این لایه را Code می‌نامیم. سپس شبکه سعی می‌کند همان داده‌های ورودی را با استفاده از اطلاعات موجود در لایه Latent بازسازی کند. به قسمت اول شبکه Encoder و به قسمت دوم Decoder می‌گویند. در واقع لایه‌ی کد همان Latent Space مطلوب ماست. اگر بتوانیم شبکه‌ی Encoder را به گونه‌ای آموزش بدهیم که اطلاعات لایه‌ی Code برای بازسازی ورودی‌ها کافی باشد، می‌توان نتیجه گرفت که لایه‌ی Code یک نمایش مناسب با بعد پایین برای اطلاعات ورودی است. (عکس ۲)



عکس ۲: نمایی از یک شبکه‌ی Auto-Encoder

برای ساختن یک Latent Space برای فیلم‌ها و تگ‌هایشان نیز از همین روش استفاده می‌کنیم. هر فیلم با استفاده از روش TF-IDF باید ابتدا به برداری عددی تبدیل شود که هر بعد آن نشان دهنده‌ی اهمیت یه یک تگ برای آن فیلم است. پس لازم است که لایه‌ی ورودی شبکه به اندازه‌ی تمام تگ‌ها باشد. در پیاده‌سازی ما این تعداد برابر 11352 است. در ادامه ۳ لایه‌ی مخفی قرار با تعداد ۳۰۰۰، ۱۰۰۰ و ۵۰۰ نورون در شبکه‌ی Encoder قرار می‌گیرد تا در نهایت به لایه‌ی Code با ۲۰۰ نورون برسیم. بعد از شبکه‌ی Decoder به صورت قرینه با همین تعداد لایه و نورون قرار می‌گیرد. تمامی لایه‌ها از معماری کاملاً متصل[۴] استفاده می‌کنند. برای افزایش سرعت یادگیری همه‌ی لایه‌ها از تابع فعال‌ساز ReLu استفاده می‌کنند. همچنین بعد از هر لایه، دو لایه‌ی Dropout و Batch Normalization نیز قرار گرفته است. لایه‌ی Dropout با غیرفعال کردن تصادفی برخی نورون‌ها از یادگیری پترن‌های خاص توسط شبکه و over fit شدن آن جلوگیری می‌کند و کمک می‌کند تا شبکه الگوهای کلی‌تر را پیدا کند. لایه‌های Batch Normalization با نرمال کردی خروجی‌های هر لایه (از جمله لایه‌ی ورودی)، کمک می‌کنند تا ورودی هر لایه نرمال باشد و از مشکل Exploding Gradient ها جلوگیری می‌کند.

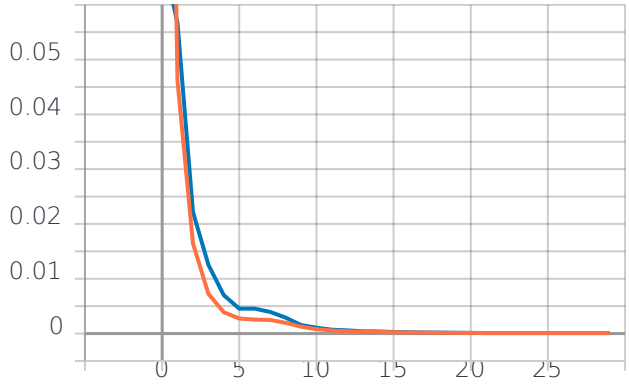
آموزش شبکه

برای اینکه روند یادگیری شبکه را کنترل و نظارت کنیم، داده‌ها را قبل از انجام عملیات TF-IDF از هم جدا می‌کنیم و داده‌های Train، Validation و Test را جدا می‌کنیم. بنابرین داده‌های غیر آموزشی ممکن است دارای تگ‌هایی باشند که در Train نباشد و درنتیجه این تگ‌ها بی کاربرد می‌شوند.

این شبکه به دلیل داشتن ابعاد ورودی بسیار بالا، دارای تعداد ۷۵ میلیون پارامتر برای یادگیری است که رقم بسیار بالایی است و همچنین خطر over fit کردن را نیز افزایش می‌دهد. به همین دلیل، برای افزایش سرعت از Nadam Optimizer[۵] [۶]با اندازه batch های۱۶ تایی استفاده می‌کنیم تا مشکل حافظه هم نداشته باشیم. برای رفع خطر over fit از داده‌هایvalidation استفاده کردیم و از Early Stopping استفاده کرده‌ایم که بهترین خطای validation انتخاب می‌کند.Dropout که پیشتر توضیح داده شد هم به overfit نشدن کمک می‌کند. درضمن ازMSE نیز برای محاسبه خطای خروجی‌ها استفاده کرده‌ایم.

مدل را پارامترهای فوق به اندازه‌ی ۳۰ epoch آموزش می‌دهیم. با این کار خطای train از 0.3395 در آغاز به 8.3×10^{-5} میرسد. همچنین خطای validation نیز به همین مقدار می‌رسد که نشان دهنده‌ی overfit نشدن مدل است. گراف پایین روند کاهش خطای validation و training را نشان می‌دهد.

4 Fully Connected
5 https://ruder.io/optimizing-gradient-descent/
6 Nestrov Accelerated Gradient و روش Adam ترکیبی از روش



عکس ۳: خطای MSE در هر epoch برای یادگیری تگ‌ها

استفاده از اطلاعات Latent Space

حال که توانستیم شبکه‌ی Auto-Encoder را آموزش دهیم، کافیست قسمت Decoder شبکه را کنار گذاشته و با استفاده از قسمت Encoder، نمایش هر فیلم در فضای Latent را بدست آوریم و سپس با استفاده از روش شباهت کسینوسی که توضیح داده شد، شباهت دوبه‌دوی فیلم‌ها را در این فضا محاسبه کرده و سپس با برای هر فیلم، فیلم‌هایی که بیشترین شباهت کسینوسی را به آن دارند به عنوان خروجی توصیه‌گر انتخاب کنیم.

پیاده‌سازی در پایتون

برای پیاده‌سازی مدل، از فریم ورک Tensorflow و Keras استفاده کرده‌ایم. همچنین کتاب‌خانه‌های Pandas و Scikit-Learn نیز برای کار با داده‌ها، محاسبه‌ی مشابهت کسینوسی و بردارهای TF-IDF استفاده شده است. پیاده سازی شبکه‌ی عصبی در نوت بوک movielens_10mil_tag_embedding قرار دارد.

برای پیاده‌سازی شبکه، از یک مدل Sequential در فریم‌ورک keras استفاده کرده‌ایم.

```
tags_auto_encoder = keras.models.Sequential()
hidden_units = [3000, 1000, 500, 200, 500, 1000, 3000]
tags_auto_encoder.add(InputLayer(X_train_tags.shape[1]))
tags_auto_encoder.add(BatchNormalization())
tags_auto_encoder.add(Dropout(0.2))
for u in hidden_units:
    tags_auto_encoder.add(Dense(u, activation="relu"))
    tags_auto_encoder.add(BatchNormalization())
    tags_auto_encoder.add(Dropout(0.3))
tags_auto_encoder.add(Dense(X_train_tags.shape[1]))
```

عکس ۴: پیاده‌سازی شبکه‌ی Auto-Encdoer تگ‌ها در Keras

برای انجام فرایند EarlyStopping از Callback آنکه توسط کراس ارائه می‌شود استفاده کرده‌ایم. همچنین برای نظارت بر یادگیری و رسم گراف خطاها از TensorBoard استفاده شده است.

```
Epoch 30/30
398/398 [==============================] - 132s 331ms/step - loss: 8.3466e-05 - mean_squared_error: 8.3466e-05 - v
al loss: 8.3174e-05 - val mean squared error: 8.3174e-0
```

عکس ۵: آخرین epoch در یادگیری شبکه‌ی تگ‌ها

بعد از یادگیری، مدل را ذخیره می‌کنیم تا در مرحله‌ی بعد استفاده شود.

در فایل Recommenders.py کلاسی به نام TagBasedRecommender وجود دارد که با دریافت آدرس شبکه‌ی عصبی ذخیره شده، آدرس مدل TF-IDF و تعداد لایه‌های Encoder، مدل‌ها را لود می‌کند و قسمت Encoder از شبکه را نگه می‌دارد. حال با استفاده از توابع این کلاس می‌توان برای هر فیلم، فیلم‌های مشابه آن را دریافت کرد.

```python
class TagBasedRecommender:
    def __init__(self, autoencoder_path, vectorizer_path, n_layers=12):
        self._load_encoder(autoencoder_path, n_layers)
        self._load_vectorizer(vectorizer_path)

    def _load_encoder(self, path, n_layers):
        autoencoder = keras.models.load_model(path)
        self._encoder = keras.models.Sequential()
        self._encoder.add(autoencoder.input)
        for layer in autoencoder.layers[:12]:
            self._encoder.add(layer)

    def _load_vectorizer(self, path):
        self._vectorizer: "TfidfVectorizer" = load(path)
```

عکس ۶: کلاس TagBasedRecommender

تابع Transform با دریافت لیست تگ‌های فیلم‌ها، ابتدا با استفاده از مدل TF-IDF Vectorizer که هنگام آموزش شبکه ساخته شده بود و توسط این کلاس لود می‌شود، بردار فیلم‌های ورودی را می‌سازد و سپس با دادن آن‌ها به شبکه‌ی عصبی، نمایش آن‌ها در Latent Space را به دست می‌آورد. تابع calculate_similarity_matrix با استفاده از تابع نتایج تابع قبل، و تابع cosine_similarity از کتابخانه‌ی scikit-learn، یک ماتریس از شباهت کسینوسی دوبه‌دوی فیلم‌ها به دست می‌آورد. در نهایت تابع recommend با دریافت لیست نام فیلم‌ها و تعداد پیشنهادهای مورد نیاز، لیستی از فیلم‌های پیشنهادی را

برمی‌گرداند.

```
def _transform(self, movies_tags):
    vectorized = self._vectorizer.transform((movies_tags["tag"]))
    embeddings = self._encoder.predict(vectorized.todense())
    return pd.DataFrame(data=embeddings, index=movies_tags.index)

def calculate_similarity_matrix(self, movies_tags):
    embeddings = self._transform(movies_tags)
    self.similarity_matrix = pd.DataFrame(data=cosine_similarity(embeddings), index=movies_tags.index,
                                          columns=movies_tags.index)

def recommend(self, movies, item, n):
    similar_items = pd.DataFrame(self.similarity_matrix.loc[item])
    similar_items.columns = ['similarity']
    similar_items.sort_values(by="similarity", ascending=False, inplace=True)
    similar_items.drop(index=item, inplace=True)
    similar_items = similar_items.head(n)
    similar_items["name"] = movies.loc[similar_items.index]
    return similar_items
```

عکس ۷: پیاده سازی توصیه‌گر مبتنی بر محتوا

خروجی مدل

خروجی این مدل را برای یک نمونه فیلم کنترل می‌کنیم. در این مثال از فیلم Toy Story 1 استفاده شده است. ۱۰ فیلم با بیشترین مشابهت و مقدار مشابهت آن‌ها به این فیلم در عکس پایین آمده است. در ضمن هر فیلم با خودش مشابهت ۱ دارد که باید از این لیست حذف شود.همان‌طور که دیده می‌شود، فیلم‌هایی مانند Toy Story 2، The Incredibles و ... پیشنهاد شده است که می‌دانیم مشابهت زیادی به فیلم Toy Story 1 دارند[۷].

movieId	similarity	name
3114	0.776632	Toy Story 2 (1999)
2018	0.717052	Bambi (1942)
2355	0.709286	Bug's Life, A (1998)
2080	0.693575	Lady and the Tramp (1955)
8961	0.632617	Incredibles, The (2004)
3964	0.630069	Adventures of Ichabod and Mr. Toad, The (1949)
5533	0.617095	Children On Their Birthdays (2002)
6377	0.611607	Finding Nemo (2003)
3615	0.606088	Dinosaur (2000)
5444	0.604657	Lilo & Stitch (2002)

عکس ۸: پیشنهادات سیستم مبتنی بر محتوا برای فیلم Toy Story 1

۷ با تشکر از دوستم امیر بابامحمودی بابت بررسی پیشنهادها.

در ادامه، مدل دوم که مبتنی بر همکاری است را بررسی می‌کنیم.

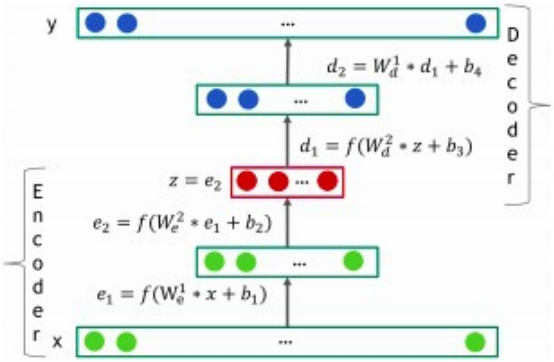
مدل مبتنی بر همکاری

مدل‌های مبتنی برهمکاری، با استفاده از سوابق یک کاربر مانند امتیازهای کاربر برای فیلم‌ها، سوابق فیلم‌های دیده شده، پسندیدن یا نپسندیدن فیلم و مواردی مانند این‌ها عمل می‌کند. درواقع هر کاربر مجموعه‌ای از فیلم‌ها را دیده است و برای هرکدام بازخوردیرا ثبت می‌کند. حال با پیدا کردن کاربرهایی با سلیقه‌ی مشابه، می‌توان فیلم‌هایی که یک کاربر دیده است را به کاربر مشابهش پیشنهاد کرد.

برای پیاده‌سازی این مدل‌ها نیز روش‌های مختلفی وجود دارد که معروف‌ترین آن‌ها مجدداً روش‌های مبتنی بر تجزیه‌ی ماتریسی است. اما باز هم به جای استفاده از این روش‌ها، سعی می‌کنیم مدلی مبتنی بر شبکه‌های عصبی و Auto-Encoder ها ارائه دهیم. مدلی که در ادامه بررسی می‌شود نوعی از مدل‌های AutoRec است که توسط محققان دانشگاه ملی اتریش ارائه شده است.[4] (همچنین [5]، [6] و [7])

برای پیاده‌سازی این مدل لازم است ابتدا ماتریس User-Item را تشکیل دهیم. این ماتریس امتیاز هر کاربر به ازای هر فیلم را ذخیره می‌کند. برای فیلم‌هایی که کاربر به آن‌ها امتیازی نداده است عدد صفر را قرار می‌دهیم.

در این مدل مجدداً می‌خواهیم از Auto-Encoder ها استفاده کنیم. ساختار این شبکه به این صورت است که با دریافت امتیازهای یک کاربر برای تمام فیلم‌ها، ابتدا آن را به یک Latent Space منتقل می‌کند و سپس سعی می‌کند با استفاده از یک شبکه‌ی Decoder دوباره این امتیازها را بازسازی کند. مشخص است که چنین شبکه‌ای نیاز به یک لایه‌ی ورودی به تعداد همه‌ی فیلم‌ها دارد و به تدریج با رسیدن به لایه‌ی Code، ابعاد کاهش می‌یابد. در Latent Space ای که توسط این مدل ساخته می‌شود، خود کاربرها اهمیت ندارند، بلکه فیلم‌هایی که آن‌ها مشاهده کردند و امتیاز داده‌اند باعث می‌شود که ارتباط بین فیلم‌ها مشخص شود و الگوهایی استخراج شود. می‌توان به این فضا به عنوان سلیقه‌های متفاوت و ارتباط فیلم‌ها با این سلیقه‌ها نگاه کرد.



عکس ۹: ساختار یک شبکه‌ی AutoRec

تا اینجا به نظر می‌رسد که با یک Auto Encoder عادی مواجه هستیم؛ اما این بار به جای اینکه سعی کنیم خروجی‌ها دقیقاً مطابق ورودی‌ها باشند و سپس از فضای Latent مستقیماً استفاده کنیم، به مدل آموزش می‌دهیم که خروجی‌های شبکه‌ی Decoder برای فیلم‌هایی که توسط کاربر دیده شده است برابر امتیاز واقعی کاربر باشد و برای فیلم‌هایی که کاربر آن‌ها را ندیده است، به‌جای خروجی صفر، یک امتیاز پیش‌بینی شود. در واقع در این شبکه کاربر را به یک Latent Space می‌بریم و سلیقه‌ی او را تشخیص می‌دهیم. سپس با با استفاده از این سلیقه و الگوهای شناسایی شده بین فیلم‌ها و سلیقه‌ها، امتیازی را برای هر فیلم پیش‌بینی می‌کنیم.

ساخت چنین شبکه‌ای تا حد زیادی مانند قسمت قبل و یک Auto-Encoder معمولی است. برای رسیدن به هدفمان که پیش‌بینی امتیاز به جای گرفتن خروجی صفر است، لازم است که اصلاح کوچکی را تابع Loss مدل ایجاد کنیم. به همین دلیل، تابع Loss جدیدی به نام Masked MSE معرفی می‌شود. این تابع خطا مانند یک تابع MSE معمولی عمل می‌کند با این تفاوت که برای خروجی‌های متناظر با ورودی صفر، خطا را برابر صفر قرار می‌دهیم. بنابراین اگر کاربر به فیلمی امتیاز نداده باشد، به جای محاسبه خطای خروجی شبکه و تلاش برای صفر کردن خروجی، شبکه را آزاد می‌گذاریم تا یک امتیاز پیش‌بینی کند.

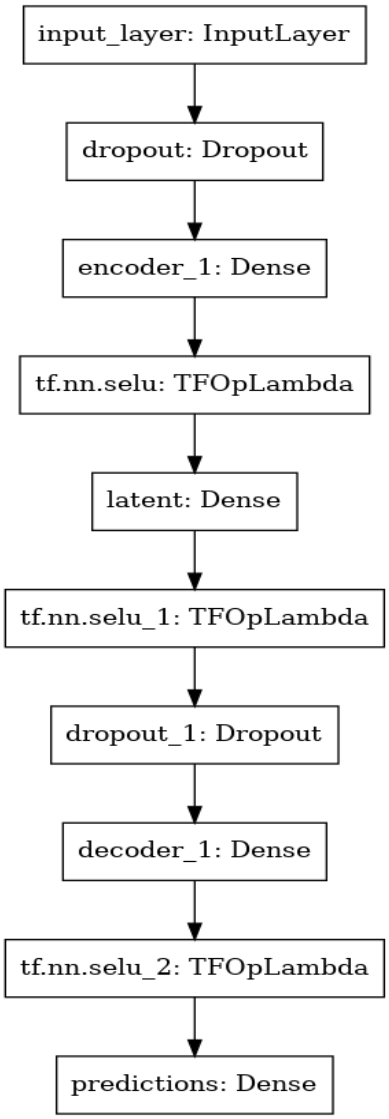
به‌طور دقیق‌تر، اگر برای یک کاربر r_i امتیاز واقعی او به فیلم i باشد و y_i خروجی شبکه به ازای فیلم i باشد، تابع Masked MSE به صورت زیر تعریف می‌شود که در آن $m_i = 1$ اگر به فیلم i امتیاز داده شده باشد و $m_i = 0$ اگر امتیاز داده نشده باشد.

$$MMSE = \frac{m_i \times (r_i - y_i)^2}{\sum_{i=0}^{i=n} m_i}$$

فرمول ۲: تابع MMSE

حال با تعریف این تابع می‌توان به بررسی باقی ساختار شبکه پرداخت. در این شبکه نیز پس از لایه‌ی ورودی که به تعداد فیلم‌ها نورون ورودی دارد، تنها یک لایه‌ی مخفی با ۵۱۲ نورون قرار می‌دهیم و سپس لایه‌ی Code با ۲۵۶ نورون قرار می‌گیرد. برای بخش Decoder هم مجدداً یک لایه‌ی ۵۱۲ نورونی و لایه‌ی خروجی به اندازه‌ی تعداد فیلم‌ها اضافه می‌شود. با توجه به این که تعداد فیلم‌ها بالاست (بالای ۸۰۰۰ فیلم) شاید به نظر برسد که شبکه عمق و ابعاد کافی را نداشته باشد. اما باید توجه کرد که در یک سیستم Collaborative اگر پشنهادها بیش از حد برای کاربر شخصی‌سازی شود، کاربر شانس یافتن آیتم‌های جدیدی که قبلاً مشابه آن‌ها را کمتر دیده است پیدا نمی‌کند. بنابراین با کم کردن پیچیدگی شبکه، از شخصی‌سازی بیش از اندازه‌ی

پیشنهادها جلوگیری می‌کنیم. برای تابع فعال‌ساز لایه‌ها به پیشنهاد مقاله‌ی اصلی از تابع Selu استفاده شده است که نسخه‌ای اصلاح شده از تابع Elu است. ویژگی خاص این تابع آن است که در یک مدل Sequential مانند مدل ما، خروجی لایه نرمال می‌مانند و نیازی به استفاده از روش‌هایی مانند Batch Normalization نیست. همچنین لایه‌های Dropout نیز بعد از لایه‌ی ورودی و لایه‌ی Code قرار گرفته است تا از over fit جلو گیری شود. علاوه بر آن از L2 Regularization نیز در همه‌ی لایه‌ها استفاده شده است که با اضافه کردن یک جمله‌ی درجه ۲ به خطای هر نورون از صفر شدن خطا جلوگیری می‌کند و درنتیجه باز هم خطر over fit کاهش می‌یابد. در عکس زیر شماتیک این شبکه مشخص شده است.



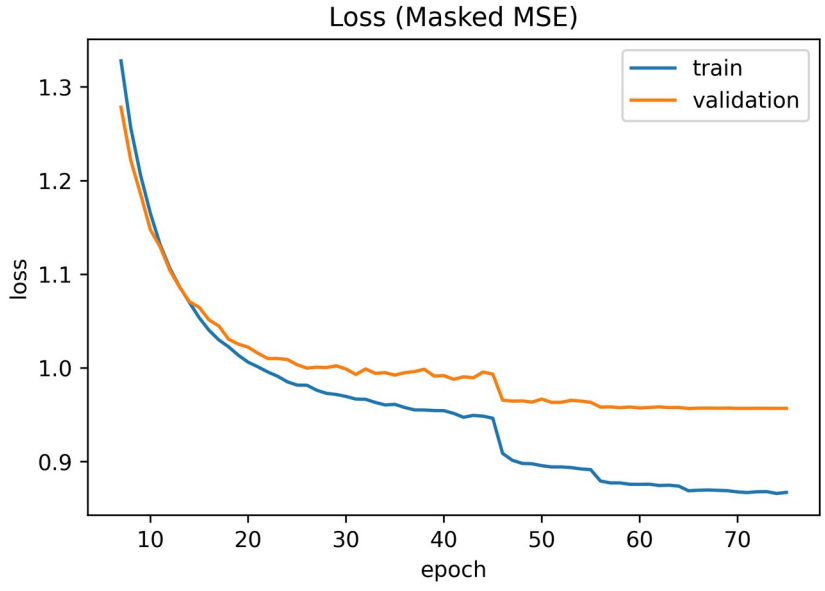
عکس ۱۰: شماتیک شبکه‌ی AutoRec

آموزش شبکه

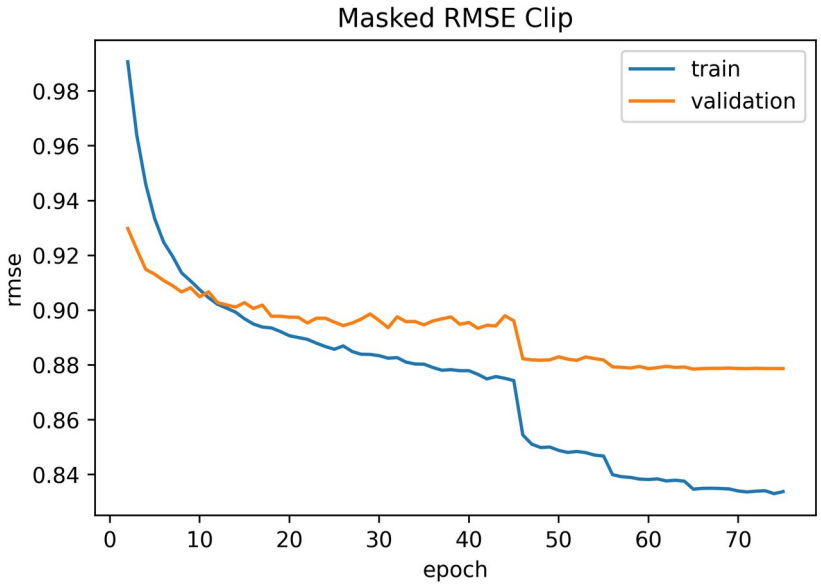
برای آموزش شبکه‌ی فوق، پس از بررسی دو روش Nadam و Adam، با عملکرد بهتر روش Adam این optimizer را انتخاب کردیم. همچنین با بررسی مقادیر مختلف ضریب یادگیری، در نهایت مقدار ۰.۰۰۰۱ انتخاب شد. مانند مدل قبلی، در این مدل هم از Early Stopping استفاده می‌شود. علاوه بر آن، از روش کاهش ضریب یادگیری نیز استفاده شده است. در این روش هنگامی که خطای Validation پس از تعداد معینی epoch کاهش نمی‌یابد، ضریب یادگیری کاهش می‌یابد تا همگرایی به نقطه‌ی مینیموم صورت بگیرد.

در این مدل، با توجه به بزرگ بودن دیتاست، فقط ۳ میلیون از ۱۰ میلیون داده را استفاده می‌کنیم. هر کدام از این داده‌ها امتیاز یک کاربر به یک فیلم را نشان می‌دهد و لازم است که ماتریس User-Item از روی آن ساخته شود. با توجه به اینکه بخش زیادی از امتیازها را دیتاست حذف کرده‌ایم، ممکن است بعضی فیلم‌ها تعداد امتیازهای کمی داشته باشند. به همین دلیل فقط فیلم‌هایی با بیش از ۱۰ امتیازدهی را نگه می‌داریم. در نهایت ماتریس امتیازها دارای حدود ۲۰۰۰۰ کاربر و ۸۰۰۰ فیلم است که به سه زیرمجموعه Train, Test و Validation تقسیم می‌شود.

یادگیری شبکه در epoch ۷۶ توسط Early Stopping متوقف می‌شود. میزان خطای Train در طول اجرا از 7.6 به 0.83 می‌رسد و خطای Validation هم از 2.6 به0.95 می‌رسد. البته باید توجه کرد که خروجی‌های شبکه الزاماً اعدادی بین ۱ و ۵ نیستند و اعداد کوچکتر از ۱ و بیشتر از ۵ را می‌توان به عنوان ۱ و ۵ در نظر گرفت. به همین دلیل یک تابع خطای جدید به نام Masked RMSE Clipped معرفی می‌کنیم که مقادیر کوچکتر از ۱ و بزرگ‌تر از ۵ را اصلاح می‌کند و همچنین بجای MSE از RMSE استفاده می‌کند. (از این تابع فقط برای ارزیابی استفاده می‌شود.) در این صورت خطای Train و Validation از ۲.۶۹ و ۰.۹۶ به ۰.۸۳۷۲ و ۰.۸۷۸۵ می‌رسد. نمودار این دو خطا در پایین آورده شده است.



عکس ۱۱: نمودار خطای MMSE از هفتمین epoch



عکس ۱۲: نمودار خطای M-RMSE از دومین epoch

در نمودارهای بالا در چندین مقطع کاهش ناگهانی خطا دیده می‌شود که به دلیل استفاده از روش کاهش Learning Rate است. همانطور که گفته شد، در این روش در صورت عدم بهبود خطا، Learning Rate کاهش می‌یابد.

پیاده‌سازی در پایتون

این مدل نیز با استفاده از Keras و TensorFlow پیاده‌سازی شده است. توابع مورد نیاز برای ساخت مدل و خطاها در فایل models.py پیاده شده و در نوت‌بوک movielens_10mil_user_embedding از این توابع برای اجرا و یادگیری شبکه استفاده شده است.

از تابع deep_auto_recommender_model در فایل models برای ساخت شبکه استفاده میشود. این تابع با گرفتن اندازه‌ی ورودی، اندازه‌ی لایه‌های مخفی، نام تابع فعال‌ساز و میزان dropout مدل را با استفاده از Functional API کتابخانه‌ی keras می‌سازد و مدل کامل را برمی‌گراند. این تابع امکان تست کردن پارامترهای مختلف در مدل را ممکن می‌سازد. همچنین پارامتری برای فعال‌سازی Batch Normalization در این تابع وجود دارد که در این گزارش غیرفعال بود. توابع خطایی که در این گزارش معرفی شدند هم در ادامه‌ی این

فایل پیاده شده‌اند.

در نوت‌بوک این مدل، پس از ساختن ماتریس User-Item، با استفاده از توابع فوق مدل ساخته می‌شود و یادگیری شبکه انجام می‌شود.

```
layers = [512, 256, 512]
activation = "selu"
output_activation = "selu"
```

```
ae_model = models.deep_auto_recommender_model(train_df, units=layers,
                                              activation_fn=activation,
                                              output_activation=output_activation,
                                              batch_normalization=False,
                                              latent_dropout=0.7, input_drop_out=0.1)
ae_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),
                 loss=models.masked_mse, metrics=models.masked_rmse_clip)
ae_model.summary()
```

عکس ۱۳: پارامترهای شبکه‌ی AutoRec

برای استفاده از روش کاهش Learning Rate از کلاس ReduceLROnPlateau که توسط کراس ارائه شده است استفاده می‌کنیم. همچنین از EarlyStopping، TensorBoard و ModelCheckpoint نیز استفاده می‌شود.

```
tb_cb = keras.callbacks.TensorBoard(get_board_path("ml10_users_ae"))
plateau_cb =keras.callbacks.ReduceLROnPlateau(factor=0.3, patience=4)
checkpoints_cb = keras.callbacks.ModelCheckpoint(filepath=get_checkpoint_path("ml10_users_ae"),
                                                 save_best_only=True)
early_cb = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True, min_delta=1e-4)
callbacks = [tb_cb, plateau_cb, checkpoints_cb, early_cb]
```

```
hist_ae = ae_model.fit(x=train_df, y=train_df,
                  epochs=250,
                  batch_size=32,
                  validation_data=(valid_df, valid_df),
                  callbacks=callbacks)
```

عکس ۱۴: توابع Callback و پارامترهای یادگیری شبکه‌ی AutoRec

برای استفاده از این شبکه، کلاس RatingBasedRecommender در فایل recommenders.py پیاده شده است که با دریافت آدرس مدل ذخیره شده‌ی AutoRec و لود کردن آن، می‌تواند کار پیشنهاد دادن را انجام دهد. تابع predict در این کلاس با دریافت امتیازهای یک کاربر برای تمام فیلم‌ها (یا صفر برای فیلم‌هایی که امتیاز نداده است)، امتیازها را به شبکه می‌دهد و امتیازهای پیش‌بینی شده برای هر فیلم را محاسبه می‌کند. این امتیازها سپس به طور نزولی مرتب می‌شوند و برگداننده می‌شوند. در این تابع امتیازهای واقعی کاربر از

امتیازهای پیش‌بینی شده کم می‌شوند. با این کار امتیاز فیلم‌هایی که کاربر قبلاً آن‌ها را دیده است پایین می‌آید دیگر پیشنهاد نمی‌شوند.

تابع recommend با دریافت سابقه‌ی کاربر و تعداد پیشنهادهای خواسته شده و با استفاده از تابع predict، امتیازهای پیش‌بینی شده را دریافت می‌کند و هر کدام را کنار نام فیلم قرار می‌دهد.

```
You, 7 hours ago | 1 author (You)
class RatingBasedRecommender:
    def __init__(self, path, output_ids):
        self.model = keras.models.load_model(path,
                                             custom_objects={"masked_rmse_clip": models.masked_rmse_clip,
                                                             "masked_mse": models.masked_mse})
        self.ids = output_ids

    def predict_ratings(self, user_hist):
        predictions = self.model.predict(user_hist)
        predictions = predictions - user_hist
        predictions = pd.DataFrame(predictions.T, index=self.ids, columns=["rating"]).sort_values(by="rating",
                                                                                                  ascending=False)
        return predictions

    def recommend(self, user_hist, n, movies):
        preds = self.predict_ratings(user_hist).iloc[:n]
        preds["name"] = movies.loc[preds.index]
        return preds
```

عکس ۱۵: پیاده‌سازی توصیه‌گر AutoRec براساس امتیازها

خروجی مدل

برای نمایش نمونه خروجی این مدل، یک کاربر را به طور تصادفی انتخاب می‌کنیم. ۱۰ فیلمی که این کاربر بیشترین امتیاز را به آن‌ها داده است در تصویر پایین آمده‌اند. در این فیلم‌ها Star Trek دارای ژانر علمی-تخیلی و فیلم Toy Story هم انیمیشن است.

movieId	name
1356	Star Trek: First Contact (1996)
1073	Willy Wonka & the Chocolate Factory (1971)
832	Ransom (1996)
780	Independence Day (a.k.a. ID4) (1996)
708	Truth About Cats & Dogs, The (1996)
788	Nutty Professor, The (1996)
748	Arrival, The (1996)
736	Twister (1996)
719	Multiplicity (1996)
1	Toy Story (1995)

عکس ۱۶: ۱۰ فیلم با بیشترین امتیاز کاربر

ده فیلمی که سیستم AutoRec بیشترین امتیاز را برای آن‌های پیش‌بینی کرده است در تصویر پایین آمده است. در این لیست هم فیلم‌هایی مانند Toy Story 2 و Star Wars آمده است که مطابق فیلم‌های مشاهده شده توسط کاربر هستند.

rating	name
4.472802	Sixth Sense, The (1999)
4.447361	Wallace & Gromit: A Close Shave (1995)
4.354260	Raiders of the Lost Ark (Indiana Jones and the...
4.330620	Schindler's List (1993)
4.314923	Shawshank Redemption, The (1994)
4.312569	Toy Story 2 (1999)
4.292597	Dark Knight, The (2008)
4.288939	Pirates of the Caribbean: The Curse of the Bla...
4.288165	Star Wars: Episode IV - A New Hope (a.k.a. Sta...
4.275661	Apollo 13 (1995)

عکس ۱۷: فیلم‌های پیشنهاد شده توسط سیستم

اجرای هر دو مدل AutoRec و مدل مبتنی بر تگ‌ها و خروجی‌های آن‌ها داخل نوت‌بوک report_test قرار دارد.

Talk is cheap; Show me the code!

کد کامل پروژه بر روی گیت‌هاب در آدرس زیر قرار گرفته است.

https://github.com/nimahsn/AI-Course-Projects/tree/main/report06-Recommender

جمع‌بندی

در این تمرین دو مدل متفاوت را پیاده کردیم که با دو دیدگاه مختلف به داده‌ها نگاه می‌کنند و هر کدام نوعی خاص از ریکامندر سیستم‌ها هستند. هر چند هر دوی این مدل‌ها به تنهایی قابل استفاده هستند، اما می‌توان در یک سیستم کامل ترکیبی از هر دو را داشت تا پیشنهادها جذاب‌تر و unbiased تر باشند.

منابع

1 https://files.grouplens.org/datasets/movielens/ml-10m-README.html
2 https://towardsdatascience.com/creating-a-hybrid-content-collaborative-movie-recommender-using-deep-learning-cc8b431618af
3 https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a
4 AutoRec: Autoencoders Meet Collaborative Filtering, Suvash Sedhain. http://users.cecs.anu.edu.au/~akmenon/papers/autorec/autorec-paper.pdf

5 Training Deep AutoEncoders for Collaborative Filtering, Oleksii Kuchaiev. https://arxiv.org/pdf/1708.01715.pdf
6 https://roma-coffin.medium.com/predicting-movie-recommendations-by-leveraging-deep-learning-and-movielens-data-part-3-82f456d7163c
7 https://github.com/annieptba/DATA2040_Final_Project_YARD