تولید پرس‌وجوی NoSQL با یادگیری عمیق برای پاسخ به سوالات زبان طبیعی

مهرداد رفیعی پور[1]

[1]دانشجوی ارشد مهندسی کامپیوتر ، دانشگاه کاشان

چکیده – در این مقاله با ترکیب روش‌های باناظر و تقویتی، پرسش‌های مطرح‌شده در زبان طبیعی را به پرس‌وجوی *NoSQL* تبدیل کردیم. وظیفه‌ی این تبدیل به عهده‌ی *MQueryBot* است که از برچسب‌های باناظر برای تشخیص ستون‌های هدف و از روش یادگیری تقویتی برای استخراج موجودیت بهره می‌برد. همچنین بمنظور افزایش دقت استخراج موجودیت، روش نیم‌جایزه را معرفی کردیم. در پایان مشاهده‌شد که استفاده از این روش منجر به افزایش دقت در تولید پرس‌وجو می‌شود.

کلید واژه- پردازش زبان طبیعی , NoSQL , یادگیری عمیق , یادگیری تقویتی , LSTM

۱. مقدمه

در عصر کنونی بیشتر اطلاعات موجود در قالب پایگاه‌داده‌ها نگهداری می شوند. دسرسی به اطلاعات موجود درون این پایگاه‌داده‌ها محدود به افرادی می شود که دانش استفاده از زبان پرس‌وجوی[۱] مربوط به آن پایگاه را دارند. بمنظور گسترش ضریب دسترسی کاربران عادی به داده‌های موجود، از عوامل گفت‌وگو[۲] استفاده می‌شود. با استفاده از عامل گفت‌وگو، می‌توان پرسش زبان طبیعی را به پرس‌وجوی پایگاه‌داده تبدیل کرد، و بر این پایه، امکان دسترسی به دانش موجود در آن‌ها را برای کاربر فراهم نمود. کاربرد عوامل گفت و گو به گستره‌ی ارتباط انسان با ماشین است. برخی از این عامل ها هدف‌محور[۳] می‌باشند که بمنظور کسب اطلاعات از یک مکالمه، جهت انجام وظیفه‌ی خاص طراحی می شوند، که به همین دلیل در مکالمه‌های کوتاه مورد استفاده قرار می‌گیرند. اما در نقطه‌ی مقابل، عامل غیرهدف‌گرا وجود دارد که برای استفاده از مکالمات طولانی و بدون محدودیت زمینه طراحی می شود.

ما MQueryBot را به عنوان یک عامل برای تبدیل پرسش زبان طبیعی به کوئری NoSQL تعریف می کنیم. در این عامل، پاسخ‌ها بصورت فکتوئید[۴] به کاربر نمایش داده می‌شوند.

به این منظور که بتوانیم MQueryBot را برای داده‌های مختلف بکار ببریم، باید دو شرط را در طراحی آن درنظر می‌گرفتیم؛ شرط اول اینکه تا حد امکان بصورت end-to-end قابل آموزش باشد، چرا که داده‌های برچسب‌دار، جزو منابع کم‌یاب حساب می‌شوند و نیازمند دانش زمینه‌ای[۵] هستند، و همچنین به آسانی قابل گسترش به مدل های بزرگ‌تر نیستند. (Chen Liang, 2016)

اما درمورد شرط دوم؛ معمولا دو راه برای ارتباط با پایگاه داده وجود دارد. می توان کل داده‌ها را درون حافظه موقت بارگزاری نمود و از انعطاف دسرسی مستقیم به داده‌ها و همچنین جست‌وجوی احتمالاتی[۶] بهره برد. این روش ما را از مزیت های پایگاه‌داده‌های پیشرفته امروزی بی‌بهره می‌کند و ازطرفی بدلیل نداشتن کنترل دسترسی[۷]، دارای مشکلات امنیتی بسیاری است. برای رفع نواقص روش اول می توان از پایگاه‌داده‌های خارجی[۸] مانند CouchDB و ElasticSearch استفاده کرد. بر این اساس، شرط دوم ما این است که MQueryBot با پایگاه داده‌های خارجی سازگار باشد.

کار ما شباهت بالایی به (Sebastian Blank, 2019) دارد، با این تفاوت که ما سعی کردیم تا حد امکان از برچسب‌های موجود برای افزایش دقت در بخش پیش‌بینی field بهره ببریم. همچنین کار ما مبتنی بر مدل seq2seq نیست، بلکه از مکانیزم توجه نشانگر[۹] (Dzmitry Bahdanau, 2014)برای پیدا کردن اهمیت واژه‌ها و انتخاب کلمات آغازین و پایانی هر موجودیت استفاده می کند.

[1] Query
[2] Conversational agents
[3] Goal oriented
[4] Factoid
[5] Domain Knowledge
[6] Probabilistic lookup
[7] Access control
[8] External Database
[9] Attention mecanism

۲. ادبیات پژوهش

در این بخش به معرفی قسمت های حیاتی معماری MQueryBot می پردازیم.

۱.۲ Word Embedding

به فرایند نگاشت کلمات به بردار ریاضی، با هدف استفاده از بردار ها در شبکه های عصبی، تعبیه کلمات[۱۰] می گویند (HasanPour, 2019). با استفاده از این بردارها می‌توان روابط مخفی بسیاری را بین کلمات مختلف آشکار کند. بعنوان مثال، نسبت بردار "گربه" به "پیشی" همانند نسبت بردار "سگ" به "هاپو" است!

۲.۲ Long-Short Term Memory

LSTM نوعی از شبکه‌های بازگشتی بهبود داده شده است. در شبکه‌های بازگشتی مشکل محوشدگی گرادیان[۱۱] (Hochreiter, 1998) وجود داشت که برای بهبود آن از شبکه‌ی حافظه کوتاه‌بلند مدت استفاده می‌شود. برخلاف شبکه عصبی بازگشتی سنتی که در آن محتوا در هر گام زمانی از نو بازنویسی میشود در یک شبکه عصبی بازگشتی LSTM شبکه قادر است نسبت به حفظ حافظه فعلی از طریق دروازه های معرفی شده تصمیم گیری کند. بطور شهودی اگر واحد LSTM ویژگی مهمی در دنباله ورودی در گام های ابتدایی را تشخیص دهد به‌سادگی میتواند این اطلاعات را طی مسیر طولانی منتقل کند بنابراین اینگونه وابستگی‌های بلندمدت را دریافت‌وحفظ‌می‌کند (HassanPour, 2019).

۳.۲ Attention Mechanism

توجه درواقع یک وکتور است که معمولا به عنوان ورودی، خروجی یک شبکه بازگشتی را دریافت کرده و در خروجی‌اش ضمن کاهش بعد در لایه ورودی، وزن بیشتری به ابعاد ورودی مهم‌تر می‌دهد. مکانیزم توجه در ماهیت خود، یک لایه تمام‌متصل (Dense) با تابع فعال‌ساز Softmax است (Sattarian, 2018). درواقع مکانیزم توجه این امکان را به شبکه می‌دهد تا در محدوده‌های محلی و عمومی متمرکز شود و عملیات مورد نظر را علاوه بر توجه به مکان فعلی کلمه، با در نظر داشتن کل جمله انجام دهد.

۳. مدل

معماری کلی MQueryBot در Figure 1 نمایش داده شده است.

هسته‌ی اصلی کار، شبکه‌ی اشاره‌گر مبتنی بر مکانیزم توجه[۱۲] می باشد که وظیفه‌ی پر کردن جاهای خالی در قالب ازپیش تعیین شده‌ی پرس‌وجوی elastic search را بر عهده دارد. خروجی pointer-net از عناصر ثابتی تشکیل شده است که یا یک ستون از پایگاه‌داده را انتخاب کرده و یا به بخشی از جمله برای استخراج موجودیت درون جمله اشاره می کند. برای استخراج موجودیت از درون جمله، pointer net به ابتدا و انتهای کلمه‌ی مورد نظر اشاره می کند. با این طراحی می توان موجودیت هایی که دارای کلمات طولانی هستند را با موفقیت استخراج نمود. بطور مثال در سوال "which actors played in The Cow?" اشاره‌گر آغازین بر روی The و اشاره گر پایانی بر روی Cow قرار می گیرد و هرچیزی که بین این دو اشاره گر باشد به عنوان یک موجودیت استخراج می شود.

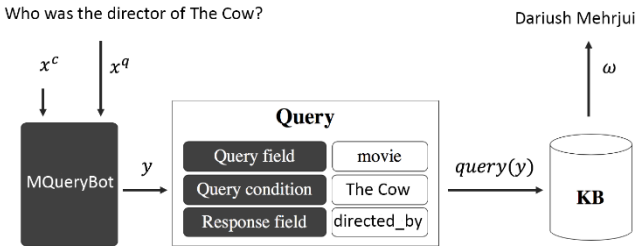


Figure 1 ساختار MQueryBot : در این شکل xq ورودی tokenize شده‌ی سوال ورودی و xc ورودی tokenize شده‌ی ستون‌های جدول پایگاه داده بعنوان ورودی به MQueryBot داده می شوند. سپس MQueryBot پردازش‌های لازم روی ورودی را انجام داده و به عنوان پاسخ y را برمیگرداند که شامل فیلد هایی است که برای پرکردن پرس‌وجوی ازپیش تعیین شده استفاده می‌شود. در پایان نتیجه‌ی کوئری مربوطه با ω نمایش داده می‌شود.

۱.۳ تشخیص ستون

مسئله‌ی تشخیص ستون در پایگاه‌داده را می توان به عنوان یک مسئله‌ی کلاس‌بندی نیز درنظر گرفت. شبکه‌ای که بمنظور حل مسئله‌ی تعیین ستون طراحی شد، دارای یک لایه embedding است که شامل ۳۰۰ لایه درونی (برابر با ۳۰۰ بعد در فضای embedding GLoVe) می‌باشد. خروجی لایه‌ی embedding به‌عنوان ورودی به یک lstm دوطرفه[۱۳] داده می شود که تعداد بعد مخفی[۱۴] آن برابر ۱۰۰ است. سپس بر روی خروجی lstm یک لایه توجه قرار می‌گیرد تا اهمیت واژه‌های درون جمله مشخص شود، و در پایان با استفاده از یک لایه خطی کاملا متصل[۱۵]، ستون مورد نظر در دیتابیس انتخاب می شود.

[10] Word embedding
[11] Vanishing Gradient
[12] Attention-based pointer network
[13] Bidirectional
[14] Hidden Dimention
[15] Fully connected Linear layer

۲.۳ پیش‌بینی تجمیع

پیش‌بینی تجمیع[16] در پرس‌وجو نیز، مانند مسئله‌ی تشخیص ستون یک مسئله‌ی کلاس‌بندی است. به همین منظور طراحی این ماژول نیز طراحی مشابه بخش 3. 1 دارد. تفاوت این بخش در این است که در لایه‌ی آخر شبکه، بجای تعداد ستون‌های جدول، کلاس ۰ برای عدم تجمیع و کلاس ۱ برای تجمیع نتایج حاصل از کوئری درنظر گرفته شد.

۳.۳ شبکه‌ی اشاره‌گر

هدف از شبکه اشاره‌گر[17]، استخراج موجودیت از درون متن سوال می‌باشد. به همین منظور، لایه‌ی اولیه دارای یک embedding layer است. خروجی embedding layer به یک lstm دوطرفه داده می شود و سپس بر روی خروجی lstm، یک لایه توجه قرار می‌گیرد تا مشخص شود کدام بخش از جمله دارای اهمیت بیشتری است. خروجی لایه توجه به اندازه‌ی بعد lstm بسط داده می شود تا برای پردازش در لایه‌ی بعدی آماده شود. در لایه بعد از یک مدل ترتیبی[18] استفاده شد که بصورت زیر تعریف می شود:

$$U_{attention} = \text{Tanh}(\text{Attention} * \text{Lstm}_{out})$$
$$\text{Probs} = \text{LinearConnectedLayer}(U_{attention})$$
$$\text{Prediction}_{final} = \text{Softmax}(\text{Probs})$$

Equation 1لایه‌ی خروجی شبکه‌ی اشاره‌گر

نکته‌ای که در طراحی حائز اهمیت بود، صفر کردن احتمال انتخاب ایندکس های خارج از محدوده برای پیش‌بینی بود. بنابراین با داشتن طول توکن های جمله، پیش از استفاده از مقادیر Probs در تابع Predictionfinal اقدام به صفر کردن احتمال انتخاب مقادیر خارج از محدوده می‌کنیم.

۴.۳ آموزش مدل با یادگیری تقویتی

از آنجایی که استخراج موجودیت از متن نیاز به هزینه‌ی زیاد در بخش انسانی است، در این بخش از یادگیری تقویتی استفاده می‌کنیم. وزن‌های شبکه بر اساس فرمول:

$$\theta \leftarrow \theta + \alpha R \nabla_\theta \sum_{t \in T} \log \pi \left(y_t \mid y_1, ..., y_{t-1}, x, \theta \right)$$

Equation 2 فرمول بروزرسانی وزن‌های شبکه

بروزرسانی می‌شوند که در آن R جایزه‌ی مرحله‌ای[19]، α نرخ یادگیری[20] و y_1 تا y_t وزن‌های مربوط به لایه‌ی خروجی شبکه می‌باشد.
بمنظور گریز از مینیموم‌های محلی[21] (Alexander Strehl, 2005) از جایزه‌ی اکتشاف استفاده می‌کنیم که مطابق تعریف می شود. مطابق این جایزه، احتمال انتخاب حالت‌های کمتر دیده شده افزایش می‌یابد.

$$R^+ = \begin{cases} \sqrt{\frac{2*\log(n)}{count(y_t)}}, & \text{if exploration reward is active} \\ 0, & \text{else} \end{cases}$$

Equation 3 جایزه‌ی اکتشاف

در Equation 3 مقدار n برابر با اندازه‌ی اندازه‌ی دسته[22]، و $count(y_t)$ تعداد دفعاتی است که خروجی y_t در دسته‌ی کنونی انتخاب شده‌است. درنهایت، تابع جایزه‌ی استفاده شده، برابر با است.

$$\begin{cases} 25 + R^+ & result\ is\ correct, Entity\ Matches \\ 5 + R^+ & result\ is\ Correct, Entity\ Doesn't\ Match \\ .4 + R^+ & Valid\ Query\ , incorrect\ result \\ .2 + R^+ & Query\ is\ not\ valid \end{cases}$$

Equation 4 جایزه‌ی عامل

به یک پرس‌وجو زمانی معتبر می‌گوییم که درهنگام اجرای آن خطایی رخ ندهد و همچنین جواب دریافت شده از پایگاه‌داده خالی نباشد. همچنین ما بمنظور عملکرد بهتر شبکه اشاره‌گر، اقدام به جداسازی جایزه برای حالت‌هایی کردیم که موجودیت استخراج شده از پرسش، دقیقا برابر اطلاعات موجود در فیلد شرط پرس‌وجو باشد. به همین منظور درصورتی که حقیقت استخراجی برابر با حقیقت پیش‌بینی شده باشد، اما موجودیت استخراج شده، برابر با تمام موجودیت نباشد، جایزه‌ی کمتری نسبت به بالاترین جایزه به شبکه می‌دهیم.

۴. آزمایشات

۱.۴ محیط توسعه

بمنظور ثبت شرایط اجرای آزمایش، در این بخش به معرفی ابزارهای مورد استفاده در شبیه‌سازی می‌پردازیم.

[16] Aggregation Predictor
[17] Pointer Network
[18] Sequencial
[19] Episodic Reward
[20] Learning Rate
[21] Local minima
[22] Batch Size

۴.۱.۱ پایگاه‌داده

پایگاه‌داده‌ی مورد استفاده در این آزمایشات ElasticSearch[۲۳] بود چراکه ES درحال حاضر از معروف‌ترین دیتابیس‌های NoSQL بوده و توسط شرکت‌های بزرگ مورد استفاده قرار می‌گیرد. ES از معماری سندمحور بهره می‌برد. به همین دلیل می‌توان با استفاده از قالب Json با آن ارتباط برقرار کرد. ما نیز در کار خود از این قالب برای پر کردن پرس‌وجو استفاده کردیم.

۴.۱.۲ محیط شبیه‌سازی

برای شبیه‌سازی مدل طرح شده در بخش 3 از زبان پایتون و ابزار pytorch[24] استفاده شد. علت این انتخاب، هماهنگی مناسب آن با تکنولوژی cuda و امکان اجرای مدل بر روی gpu است. همچنین محیط آزمایش دارای حافظه ۱۶ گیگابایت، پردازنده‌ی intel i7 7700k 4.7ghz ، و گرافیک gtx1060 بود. مدت زمان اجرا برای آموزش کامل مدل با استفاده از هسته‌های cuda، ۱۵ دقیقه ثبت شد. ضمن اینکه بمنظور کاهش حجم محاسباتی، embedding بطور کامل در لایه‌ی embed بارگذاری نشده و به بارگزاری کلمات موجود در متن سوال و جواب اکتفا شد. این کار بطور مؤثر، زمان اجرا و حافظه‌داخلی مورد نیاز pytorch را کاهش می‌دهد.

۴.۲ دیتاست Movie Dialog

ما در این مقاله قصد داریم به سوالات، بدون درنظر داشتن زمینه‌ی محتوایی قبلی، بصورت فکتوئید پاسخ دهیم. بنابراین از دیتاست (Jesse Dodge, 2016)[۲۵] که حاوی اطلاعات فیلم‌های وبسایت imdb است استفاده کردیم. Figure 2 ستون‌های این پایگاه‌داده و یکی از موجودیت‌های درون آن را نشان می‌دهد.


```
{
  "movie": "A Separation",
  "directed_by": "Asghar Farhadi",
  "written_by":"Asghar Farhadi",
  "starred_actors": "Shahab Hosseini, Peyman Moaadi, Leila Hatami, Sareh Bayat",
  "release_year": "2011",
  "in_language": "Persian",
  "has_genre": "Drama",
  "has_imdb_rating": "fantastic",
  "has_imdb_votes": "famous",
  "has_tags": "social commentary, imdb top 250, tense, religion, great acting, act
  "has_plot": "A married couple are faced with a difficult decision - to improve t
}
```


Figure 2 دیتاست MovieQA

این دیتاست شامل ۱۷.۳۵۰ فیلم در ۱۱ فیلد مانند عنوان فیلم، نویسنده، ژانر، کارگردان و غیره می باشد. از مزایای این دیتاست دسته‌بندی سوالات مربوط به هردسته از فیلدهای جدول است که به ما کمک می‌کند برچسب های مورد نیاز برای استخراج فیلد هدف را داشته باشیم.

در اولین اجرا، اقدام به هضم[۲۶] این دیتاست در پایگاه‌داده‌ی ES می کنیم. وظیفه‌ی این بخش بر عهده‌ی pre.py می باشد که فایل دیتاست را خوانده و با استفاده از پرس‌وجوی Json اقدام به درج فیلم‌ها درون پایگاه می‌کند.

به منظور بازیابی اطلاعات از یک پرس‌وجوی ازپیش تهیه‌شده استفاده می‌کنیم که ۳ جای خالی دارد. فیلد شرط که برابر کلیدواژه‌ی where در زبان Sql می‌شود، فیلد داده که برابر با مقدار موجودیت هدف بوده و جای خالی مقابل فیلد شرط را پر می‌کند، و فیلد پاسخ که برابر با حقیقت خواهد بود.

به همین منظور باید از مدل خود ۵ خروجی دریافت کنیم. خروجی اول و دوم برابر با فیلد شرط و فیلد هدف، خروجی سوم و چهارم برای توکن آغازین و پایانی موجودیت، و خروجی آخر که نمایانگر دستور تجمیع نتایج دریافتی از پایگاه داده است. مثال Table 1 بیانگر همین مسئله است.

مثال اجرا شده

سوال	Who was the director of Iron Island?
فیلد شرط	movie
موجودیت	Iron Island
فیلد پاسخ	Directed_by
حقیقت	Mohammad Rasoulof

Table 1سوال زبان طبیعی و خروجی‌های KBQueryBot برای ایجاد پرس‌وجو

۵. مراحل یادگیری

برای وزن‌دهی اولیه به کلمات، از GloVe embedding[۲۷] استفاده کردیم که دارای ۳۰۰بعد، و ۴۲ میلیارد کلمه است. به منظور بهینگی در زمان و کاهش هزینه‌ی اجرا، این کرپس[۲۸] را به اندازه‌ی تعداد کلمات بکار رفته درون دیتاست کاهش دادیم.

برای آموزش Column Detector ، ما از یک LSTM دوطرفه با عمق ۲ و تعداد بعد مخفی ۱۰۰ تایی استفاده کردیم. مدنظر داشته باشید که

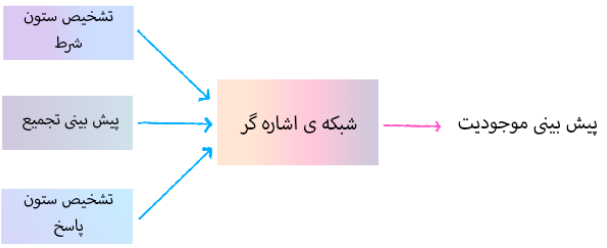
[23] https://www.elastic.co/
[24] https://pytorch.org/
[25] http://beforethecode.com/projects/omdb/download.aspx
[26] injest
[27] https://nlp.stanford.edu/projects/glove/
[28] Corpus

به علت دوطرفه بودن lstm، هنگام معرفی شبکه، مقدار 100/2=50 به عنوان تعداد لایه‌ی درونی به ابزار داده می شود. بمنظور جلوگیری از overfit شدن مدل، از dropout ratio 0.3 (Diederik P. Kingma, 2014) استفاده شد. درطول آموزش نرخ یادگیری α برابر 10^{-3} قرار گرفت. همچنین اندازه‌ی minibatch بین ۱۲۸ تا ۵۱۲ متغیر بود و فرایند یادگیری در ۱۰۰ مرحله[۲۹] تکرار شد.

برای انتخاب برچسب هدف، از توزیع Categorical[۳۰] استفاده شد. پس از انتخاب هر برچسب، مقدار loss بدست آمده را ازطریق فرمول CrossEntropy که برابر با 2 Equation می باشد، فاصله‌ی برچسب هدف تا برچسب فعلی را محاسبه کرده و بر آن اساس وزن‌های شبکه را بروزرسانی می‌کنیم.

برای آموزش Entity Extractor نیز از یک LSTM با طراحی مشابه بخش قبل استفاده شد با این تفاوت که در لایه‌ی خروجی از فرمول 1 Equation بهره گرفته‌شد. همچنین برای بهبود فرایند یادگیری، نسبت به کارهای گذشته (Sebastian Blank, 2019) و (Xiaojun and Liu, 2017) اقدام به جداسازی جایزه برای زمانی کردیم که موجودیت استخراج شده بطور کامل برابر با موجودیت درون پایگاه‌داده نیست. همچنین در بخش استخراج موجودیت‌ها، از یادگیری تقویتی بهره گرفته‌شد.

برای آغاز کار بخش استخراج موجودیت، ابتدا مدل‌های تشخیص ستون و پیش‌بینی تجمیع را آموزش می‌دهیم و سپس مدل آموزش دیده را بعنوان ورودی به Entity Extractor تحویل می‌دهیم. 3 Figure این موضوع را بصورت تصویری نمایش می‌دهد.



3 Figure ساختار آموزش شبکه‌ی اشاره‌گر

در هر مرحله برای انتخاب ایندکس هدف، از توزیع Multinomial[۳۱] استفاده شد. این توزیع می تواند با دریافت احتمال انتخاب هر ایندکس، به تعداد مورد نیاز، با توجه به احتمالات داده شده بصورت تصادفی ایندکس هدف را انتخاب کند. از آنجا که ما دو ایندکس برای ابتدا و انتهای موجودیت نیاز داریم، عدد ۲ بعنوان ورودی به این توزیع داده می‌شود. 4 Figure این موضوع را بصورت تصویری بیان می‌کند. در پایان با توجه به عملکرد شبکه، احتمالات خروجی از شبکه و ایندکس‌های تصادفی انتخاب شده بعنوان ورودی به 2 Equation وارد می شود و loss بدست آمده در جایزه‌ی انتخاب این ایندکس ها ضرب می‌شود.

۶. چالش ها

درابتدای پیاده‌سازی تلاش شد که از پایگاه‌داده‌ی CouchDB استفاده شود. اما مشکلات متعدد ازجمله عدم پشتیبانی از جستجوی آزاد متن[۳۲] پیاده سازی را با مشکل مواجه کرد. از مشکلاتی که مرحله به مرحله در استفاده از CouchDB پیش آمد می‌توان به عدم پشتیبانی مستقیم از جستجوی غیرحساس به کوچک و بزرگ بودن متن[۳۳] اشاره کرد. این موضوع ما را به استفاده از regex برای ایجاد جستجوی غیرحساس مجبور کرد[۳۴]. چالش بعدی نیاز به ایندکس کردن تمام فیلدهای مورد نیاز بصورت دستی بود. اما درصورتی که از regex برای جستجو استفاده شود، نمی توان از ایندکس‌ها استفاده نمود که این موضوع زمان مورد نیاز برای هر پرس‌وجو را به حدود 800ms افزایش می‌دهد. بنابراین درنهایت تصمیم به تغییر پایگاه‌داده به elastic search گرفته شد. با این تغییر زمان هر کوئری به 2ms کاهش پیدا کرد.

4 Figure برای استخراج موجودیت از این سوال، باید ایندکس های ۵ و ۷ انتخاب شوند.

چالش بعدی، بالانس کردن داده‌های مربوط به برچسب ستون شرط‌ها بود. از آنجائی که برچسب های ستون movies [۳۵] بسیار بیشتر از سایر برچسب‌ها بود، مجبور به بالانس کردن برچسب‌ها با دو روش کاهش ۳۰ درصدی ستون movie و افزایش ۱۰۰درصدی سایر ستون‌ها شدیم. این‌کار باعث overfit و dataloss چندانی نشد و دقت ما پس از انجام این‌کار همچنان بالا ماند.

[29] Epoch
[30] https://pytorch.org/docs/stable/distributions.html#categorical
[31] https://pytorch.org/docs/master/generated/torch.multinomial.html
[32] Free Text Search
[33] Case insensitive search
[34] https://stackoverflow.com/questions/45228456
[35] imbalanced

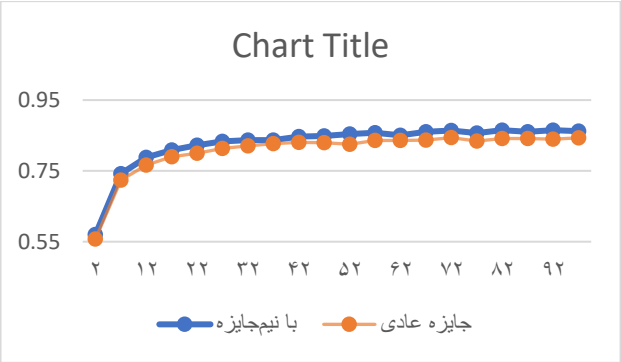


Figure 5 مقایسه‌ی روش نیم‌جایزه با جایزه‌ی عادی

۷. نتایج

همانطور که گفته شد در Equation 4 تلاش کردیم جایزه‌ی استخراج کامل موجودیت را از جایزه‌ی استخراج ناقص موجودیت جدا کنیم. Figure 5 نتیجه‌ی جداسازی جایزه را نمایش می‌دهد. مشاهده می‌کنیم که جداسازی جایزه باعث افزایش ۲درصدی دقت نسبت به طراحی بدون جداسازی می‌دهد و این برتری در تمامی مراحل خود را نمایان می سازد. دقت جایزه‌عادی بعد از دور ۵۰ آموزش تقریبا ثابت می‌ماند درحالی که دقت مدل نیم‌جایزه همچنان بعد از دور ۵۰ رشد می‌کند. همچنین روش نیم‌جایزه نسبت به روش SeqPolicyNet که در (Sebastian Blank, 2019) ذکر شده افزایش ۲.۳ درصدی دقت را دربر دارد. Table 2 به مقایسه‌ی دقت SeqPolicyNet و MQueryBot می‌پردازد.

نام مدل	دقت
SeqPolicyNet	۸۴.۲٪
MQueryBot	۸۴.۴٪
MQueryBot با نیم‌جایزه[۳۶]	۸۶.۵٪

Table 2 مقایسه‌ی دقت SeqPolicyNet با MQueryBot با انواع جایزه‌ی پیاده شده

۸. جمع‌بندی

در این مقاله به معرفی مدل MQueryBot پرداختیم که هم از برچسب‌های باناظر و هم روش یادگیری تقویتی برای ایجاد پرس‌وجوی NoSQL استفاده می‌کرد. نوآوری کار ما بهره‌گیری از راهکار نیم‌جایزه برای یاری‌رسانی به مدل بمنظور پیدا کردن محل درست موجودیت و فرار از local minima بود. درنتیجه‌ی استفاده از این راهکار، مشاهده کردیم که در راهکار نیم‌جایزه دقت کوئری‌ها به میزان ۲ درصد نسبت به جایزه‌ی‌عادی افزایش می‌دهد.

مراجع

- Alexander Strehl, M. L .(2005) .A theoretical analysis of Model-Based Interval Estimation . International Conference on Machine Learning.
- Chen Liang, J. B .(2016) .Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision.
- Diederik P. Kingma, J. B .(2014) .Adam: A Method for Stochastic Optimization.*3 .rd International Conference for Learning Representations, San Diego, 2015* .
- Dzmitry Bahdanau, K. C .(2014) .Neural Machine Translation by Jointly Learning to Align and Translate.
- HasanPour, H .(2019) .*DeepLearning.ir* بازیابی از https://deeplearning.ir/یادگیری ماشین
- HassanPour, H از بازیابی .(2019) . DeepLearning.ir: https://deeplearning.ir/آموزش-شبکه-عصبی-بازگشتی-بخش-پنجم-معرفی/
- Hochreiter, S .(1998) .The vanishing gradient problem during learning recurrent neural nets and problem solutions .*international Journal of Unvertainty, Fuzziness and Knowledge-based Systems*.
- Jesse Dodge, A. G .(2016) .Evaluating Prerequisite Qualities for Learning End-to-End Dialog Systems.
- Sattarian, M از بازیابی .(2018) . http://blog.class.vision/1397/10/attention-mechanism/
- Sebastian Blank, F. W.-P .(2019) .Querying NoSQL with Deep Learning to Answer Natural

[36] Half-Reward

Language Questions .*The Thirty-First AAAI Conference on Innovative Applications of Artificial Intelligence (IAAI-19)* .

- Xiaojun and Liu, C. a .(2017) .Generating Structured Queries From Natural Language Without Reinforcement Learning.

مهرداد رفیعی پور *(۹۲-۹۶: کارشناسی ، مهندسی نرم افزار، دانشگاه قم.* ۹۸- کارشناسی ارشد ، مهندسی نرم افزار ، دانشگاه کاشان .)