دانشکده مهندسی برق

پیشنهاد اخبار با استفاده از شبکه‌های عصبی LSTM

گزارش پروژه‌ی درس شناسایی آماری الگو

علی اصغر تقی‌زاده

استاد درس

دکتر محمد رضا دلیری

بهمن ۱۳۹۹


بسم الله الرحمن الرحيم

چکیده

سیستم‌های پیشنهاد دهنده با به وجود آمدن شبکه‌ی جهانی اینترنت بسیار مورد توجه محققان قرار گرفته است. با استفاده از این سیستم‌ها می‌توان کالاها و خدمات متناسب با علایق کاربر را نمایش داد که موجب افزایش رضایت کاربران می‌شود. در سال های اخیر با رشد شبکه‌های عصبی عمیق و همچنین ایجاد بسترهای مناسب برای جمع آوری داده‌گان از فعالیت‌های کاربران در اینترنت این موضوع پیشرفت فراوانی داشته است. در این گزارش با استفاده از شبکه‌های حافظه‌ی کوتاه مدتِ بلند به ارائه‌ی راه حلی برای مسئله پیشنهاد اخبار خواهیم پرداخت.

واژگان کلیدی: پیشنهاد اخبار، شبکه‌های عصبی عمیق، شبکه‌های حافظه‌ی کوتاه مدت بلند

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست برنامه‌ها

فصل ۱

مقدمه

۱ـ۱ سامانه‌های پیشنهادگر

برخی مواقع بدون داشتن دانش و تجربه قبلی نسبت به موارد مشابه نیازمند تصمیم‌گیری هستیم. در زندگی روزمره بسیار اتفاق افتاده است که بر اساس پیشنهاد دیگران یا تبلیغات روزنامه‌ها و ... عمل کنیم. سامانه‌های پیشنهادگر ابزاری هستند که این به این فرآیند کمک می‌کنند [۳]. هدف سامانه‌های پیشنهادگر ارائه داده‌هایی متناسب با علایق کاربر است، چرا که با گسترش روز افزون اینترنت کاربران با حجم عظیمی از اطلاعات روبرو هستند و نیازمند این هستیم که این اطلاعات فیلتر شوند تا از هم کاربران سریعتر نیاز خود را پیدا کنند و هم از سردرگمی کاربران کاسته شود [۲].

سامانه‌های پیشنهادگر در زمینه‌های مختلفی مانند خرید کالاها در فروشگاه‌های آنلاین، خرید فیلم، کتاب و ...، مشاهده‌ی اخبار و مقالات مشابه کاربرد دارد. در این پروژه سعی شده است روشی برای پیشنهاد اخبار به کاربران بر اساس تاریخچه‌ی فعالیت آن‌ها ارائه شود.

۱ـ۲ انواع روش‌های موجود

در این بخش تقسیم‌بندی‌ای از روش‌های موجود برای سیستم‌های پیشنهادگر خواهیم داشت.

۱ـ۲ـ۱ روش‌های مبتنی بر محتوا

در این روش‌ها بر اساس محتوای آیتم‌هایی که کاربر قبلا به آن‌ها علاقه‌مند بوده است آیتم‌های جدید پیشنهاد می‌شود. در این روش شباهت بین آیتم‌ها بر اساس محتوای آن‌ها محاسبه می‌شود و از این معیار برای پیشنهاد آیتم جدید استفاده می‌شود.

۱ـ۲ـ۲ فیلتر کردن مشارکتی

انتخاب آیتم‌ها در این روش بر اساس رفتار گروه بزرگی از کاربران است که به دو دوسته تقسیم می‌شود.

الف‌ـ فیلتر کردن مشارکتی مبتنی بر حافظه: بر اساس رفتار کاربرانی که مشابه کاربر مدنظر بوده‌اند یا آیتم‌های که مشابه آیتم‌هایی که کاربر قبلا به آن‌ها تمایل داشته، آیتم‌های جدید پیشنهاد می‌شوند.

ب‌ـ فیلتر کردن مشارکتی مبتنی بر مدل: در این روش مدل‌هایی به کار گرفته می‌شوند که الگوهای موجود در داده را پیدا کند.

۱ـ۲ـ۳ روش‌های ترکیبی

در این روش‌های از ترکیبی از روش‌های قبلی استفاده می‌شود. در روش پیشنهاد شده در این پروژه از روش‌های ترکیبی استفاده شده است.

۱ـ۳ معرفی دادگان

مجموعه دادگان خبری مایکروسافت [1] [۵] از جمله مجموعه‌های بسیار حجیم برای مسئله پیشنهاد خبر است که از روی لاگ فعالیت‌های کاربران در سایت خبری مایکروسافت به دست آمده است. این مجموعه شامل ۱۶۰ هزار خبر، بیش از ۱۵ میلیون لاگ که توسط ۱ میلیون کاربر ایجاد شده است، می‌باشد.

با توجه به حجم بزرگ این مجموعه داده یک نمونه‌ی ۵۰ هزارتایی از لاگ کاربران نمونه برداری شده است که در آزمایشات این گزارش با توجه به منابع محدود از این قسمت استفاده شده است.

مجموعه MIND شامل دو فایل است. در یک فایل مشخصات خبرها آماده است و در فایل دیگر لاگ

[1] MIcrosoft News Dataset (MIND)

نام ویژگی	توضیح
category	دسته‌ی خبر
title	عنوان خبر
abstract	چکیده خبر
title entities	موجودیت‌هایی که در عنوان وجود دارند.
abstract entities	موجودیت‌هایی که در چکیده‌ی خبر وجود دارند.

جدول ۱ ـ ۱: ویژگی‌های اخبار در مجموعه دادگان Mind

کاربران. ویژگی‌هایی که برای هر خبر ارائه شده است در جدول ۱ ـ ۱ قابل مشاهده است. همچنین به ازای هر خبر لینک صفحه‌ی خبر مربوط برای استخراج کامل متن خبر موجود است.در این آزمایش به علت محدودیت منابع از دو ویژگی عنوان و چکیده استفاده شده است.

همچنین در فایل دیگری که مربوطه به لاگ کاربران است به ازای هر کاربر تاریخچه‌ای از خبرهایی که کاربر مروبطه مشاهده کرده است مشخص است.همچنین به ازای هر کاربر اخبار پیشنهاد شده برای او با برچسب کلیک کردن یا عدم کلیک کردن کاربر آمده است.بدین ترتیب ما با یک مسئله دسته‌بندی دو کلاسه روبرو هستیم که در آن باید تصمیم بگیریم آیا کاربر بر روی یک خبر کلیک خواهد کرد یا نه.

فصل ۲

معرفی مدل

در این بخش ابتدا به بررسی ویژگی‌های استخراج شده از ویژگی‌های موجود خواهیم پرداخت سپس مدل شبکه‌ی عصبی را بررسی خواهیم کرد.

۲ـ۱ استخراج ویژگی‌ها

همانطور که در بخش ۱ـ۳ بررسی شد هر خبر دارای ۵ ویژگی در مجموعه دادگان است. در این پروژه فقط از عنوان و چکیده‌ی خبر به عنوان ویژگی استفاده شده چرا که این دو ویژگی مهم‌ترین معیار برای انتخاب یا عدم انتخاب یک خبر به حساب می‌آیند.

اما با توجه به اینکه جنس ویژگی‌های انتخاب شده از نوع رشته [۱] است برای استفاده در شبکه‌های عصبی و اکثر روش‌های شناسایی الگو و یادگیری ماشین نیاز هست که آن‌ها را به ویژگی‌های عددی تبدیل کنیم. روش‌های مختلفی برای تبدیل رشته به ویژگی‌های عددی وجود دارد که از جمله‌ی آن‌ها می‌توان به بردار تک فعال [۲] و بردارهای تعبیه [۳] اشاره کرد.

در روش بردار تک فعال به ازای هر کدام از اسناد یک صفر و یکی به طول کل کلمات در نظر گرفته می‌شود که عناصری از آن بردار که متناظر با کلمات آن سند هستند مقدار یک و بقیه مقدار صفر می‌گیرند.این روش

[1] string
[2] one hot
[3] embedding vectors

روش ساده‌ای به حساب می‌آید و مشکلاتی از قبیل تنک بودن بردار ویژگی، در نظر نگرفتن شباهت معنایی کلمات، از بین رفتن ترتیب کلمات و ... دارد. به همین دلیل در اکثر کارهای اخیر از روش تعبیه استفاده می‌شود.

بردار تعبیه مفهومی عام است که هدف از آن تبدیل ویژگی‌های گسسته [۴] (که در اینجا کلمات هستند) به یک فضای برداری است که در این فضای برداری عناصری که به هم شبیه هستند به لحاظ فاصله نیز به هم نزدیک می‌باشند. به عبارت دیگر به ازای هر کلمه یک بردار چگال [۵] خواهیم داشت که در فضایی با تعداد ابعاد دلخواه قرار خواهند گرفت. روش‌های مختلفی برای ساخت بردارهای کلمات وجود دارد که می‌توان در دو دسته‌ی ایستا و پویا قرار داد.

در روش‌های ایستا مستقل از اینکه یک کلمه در چه بافتاری [۶] آمده باشد یک بردار ثابت به ازای هر کلمه در نظر گرفته می‌شود. اما در روش‌های پویا بردار هر کلمه بسته به بافتاری که در آن قرار دارد تغییر می‌کند. این روش مشکلات روش ایستا مانند یکسان در نظر گرفتن کلماتی که املای یکسان و معنای متفاوتی دارند را ندارد. برای ساخت بردار پویا هم روش‌های مختلفی وجود دارد و این موضوع در سالهای اخیر بسیار مورد توجه قرار گرفته است. اما می‌توان گفت مهم‌ترین روشی که یک نقطه‌ی عطف در تاریخ پردازش زبان طبیعی به حساب می‌آید روش ترنسفرمر [۷] [۱].

مفهومی که روش ترنسفرمر را از سایر روش‌ها متمایز میکند مکانیزم خود توجهی [۸] است[۴]. مکانیزم توجه مفهومی در شبکه‌های عصبی است که منظور از آن توجه به بخشی از ورودی و در نظر نگرفتن بخش‌های دیگر است. در مکانیزم خود توجهی به هنگام ساخت بردار هر کلمه به کلمه‌های اطراف آن کلمه توجه می‌شود و این باعث می‌شود برادرهای پویای بافتارگونه [۹] به ازای هر کلمه ساخته شود. در ادامه به بررسی عملی استخراج ویژگی خواهیم پرداخت.

[4] discrete
[5] dense
[6] context
[7] transformer
[8] self attention
[9] contextualized

۲ـ۱ـ۱ بررسی کدهای استخراج ویژگی‌ها

کتابخانه هاگینگ‌فیس [۱۰] مهم‌ترین کتابخانه موجود برای زبان پایتون در استفاده از بردارهای مبتنی بر روش ترنسفرمر است. استفاده از این کتابخانه بسیار آسان است و برای تبدیل متن به بردار ابتدا باید متن به کلمه یا زیر کلمات تشکیل دهنده تبدیل شود (توکنیاز) و سپس شناسه توکن‌ها به صورت یک دنباله به مدل ترنسفرمر داده می‌شود [۱۱] .

برنامهٔ ۲ـ۱: فراخوانی ترنسفرمر

```
۱ from transformers import AutoModel, AutoTokenizer
۲ model = AutoModel.from_pretrained('roberta-base')
۳ tokenizer = AutoTokenizer.from_pretrained('roberta-base')
۴ model.to('cuda')
```

در خط ۱ کتابخانه وارد شده است. در خط ۲ مدل اصلی ترنسفرمر که یک شبکه‌ی عصبی عمیق به حساب می‌آید ساخته شده است. با این دستور مدل از قبل آموزش دیده در برنامه بارگزاری می‌شود. انواع مختلف ترنسفرمرها وجود دارد که در اینجا از نمونه roberta استفاده شده است. در خط ۳ توکنایزر مربوط به مدل ساخته شده است و در نهایت به منظور پردازش سریعتر مدل به GPU منتقل شده است.

برنامهٔ ۲ـ۲: استخراج بردار تعبیه با استفاده از ترنسفرمر

```
۱ train_news_vectors = {}
۲ with torch.no_grad():
۳     for i, row in tqdm(train_news.iterrows(), total=train_news.shape[0]):
۴         inputs_title = tokenizer(row['title'], return_tensors="pt",
              max_length=512, truncation=True)
۵         inputs_title.to('cuda')
۶         outputs_title = model(**inputs_title).pooler_output
۷         inputs_abstract = tokenizer(row['abstarct'], return_tensors="pt",
```

[10] huggingface.co

[۱۱] برای اجرای کدها از google colab که یک فضای رایگان برای اجرای کدهای پایتون است استفاده شده است.

```
            max_length=512,truncation=True)
        inputs_abstract.to('cuda')
        outputs_abstract = model(**inputs_abstract).pooler_output
        train_news_vectors[row['news id']] = [outputs_title,
            outputs_abstract]
pickle.dump(train_news_vectors,
    open('/content/drive/MyDrive/pattern_proj/train_news_vectors.pkl',
    'wb'))
```

در خط ۱ یک داده ساختار دیکشنری ساخته شده است که کلید آن شناسه‌ی هر خبر و مقدار آن بردار مربوط به آن خبر است. خط دوم به منظور جلوگیری از محاسبه‌ی گرادیان‌های شبکه‌ی عصبی است که در فاز آموزش استفاده می‌شود. در خط ۳ بر روی سطرهای مجموعه دادگان حرکت میکنیم. در خطوط بعدی ابتدا عنوان خبر به توکنیازر داده شده است تا شناسه‌های توکن‌ها به دست بیاید سپس این شناسه‌ها به GPU منتقل شده‌اند تا مدل بتواند از آن‌ها استفاده کند. سپس شناسه‌ها به مدل داده شده‌اند. می‌توان از بردارهای همه‌ی کلمات به صورت جداگانه به عنوان بردار ویژگی استفاده کرد اما این به لحاظ محاسباتی مناسب نیست و بهتر است از میانگین بردارهای کلمات استفاده کرد و تنها یک بردار به ازای عنوان ساخت که در خط ۷ با گرفتن بخش pooler output از خروجی مدل این کار انجام شده است. تمامی این مراحل برای بخش چکیده خبر نیز انجام شده است. در نهایت در خط ۱۲ بردارهای استخراج شده به منظور جلوگیری از اجرای دوباره این قسمت در فایل ذخیره شده‌اند.

برنامهٔ ۲ـ۳: تبدیل لاگ کاربرها به نمونه‌های آموزش

```
def convert_to_samples(behavaiors, max_samples=600000):
    all_histories = []
    all_impressions = []
    all_labels = []
    for i, row in tqdm(behavaiors.iterrows()):
        histories = row['history'].split()
```

```
        histories_index = [news_to_index[h] for h in histories if h in
            news_to_index]
        impressions = row['impressions'].split()
        if len(histories_index) == 0:
            continue
        for imp in impressions:
            imp, label = imp.split('-')
            if imp not in news_to_index:
                continue
            all_impressions.append(news_to_index[imp])
            all_labels.append(float(label))
            all_histories.append(histories_index)
        if len(all_histories) >= max_samples:
            break
    return all_histories, all_impressions, all_labels
```

در کد تبدیل لاگ کاربری به نمونه‌های آموزشی سه لیست ساخته شده است. در لیست all_histories تاریخچه‌ی فعالیت‌ها بر اساس شناسه خبرها قرار میگیرد. در لیست all_impressions شناسه خبرهایی که به کاربر پیشنهاد داده شده شده قرار میگیرد و در لیست all_labels برچسب مربوط به کلیک یا عدم کلیک کاربر بر روی خبر پیشنهاد داده شده است.

در بخش بعدی به بررسی مدل خواهیم پرداخت.

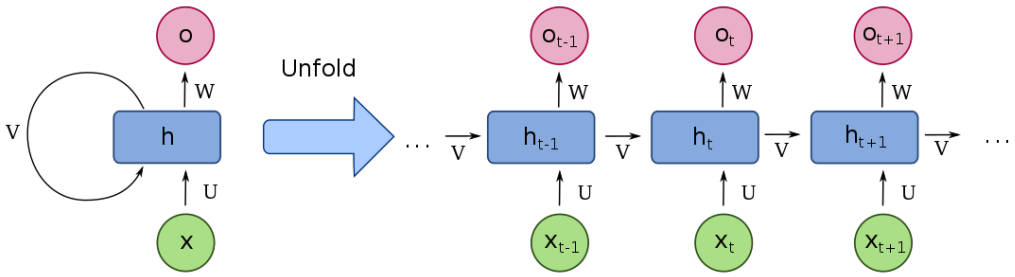
۲ـ۲ مدل حافظه‌ی کوتاه مدتِ بلند

در هر نمونه ما با دنباله‌ای از رفتار کاربر روبرو هستیم. شبکه‌های عصبی معمولی قادر به یادگیری وابستگی‌های بین عناصر دنباله‌ها نیستند و برای این منظور از شبکه‌های عصبی بازگشتی [۱۲] استفاده می‌شود که در آن دنباله‌ی

[12] recurrent neural networks

ورودی به صورت قدم به قدم به شبکه داده می‌شود و خروجی هر قدم در قدم بعدی به عنوان ورودی استفاده می‌شود. شمای کلی یک شبکه‌ی عصبی بازگشتی در شکل ۲ـ۱ دیده می‌شود.

شکل ۲ـ۱: شمای یک شبکه‌ی عصبی بازگشتی



اما شبکه‌های عصبی بازگشتی معمولی نیز خالی از ایراد نیستند. علی الخصوص آن‌ها قادر به یادگیری وابستگی‌های طولانی مدت نیستند. برای حل این مشکل شبکه‌های عصبی حافظه‌ی کوتاه مدت بلند [۱۳] به وجود آمدند که قادر به یادگیری وابستگی‌های بلند و کوتاه مدت در دنباله‌های ورودی هستند. بررسی این شبکه‌ها خارج از حوصله‌ی این نوشته است.

بنابراین شبکه‌های lstm گزینه‌ی مناسبی برای مسئله ما به حساب می‌آیند. مزیت دیگر استفاده از این نوع مدل به عنوان سیستم پیشنهادگر این است که این یک روش ترکیبی به حساب می‌آید. بدین منظور که هم محتوای اخبار استفاده می‌شود و هم دنباله‌ی فعالیت‌های کاربران یادگرفته می‌شود و در مورد کاربرانی که رفتار مشابهی دارند نتایج مشابهی را از خود نشان خواهد داد.

در ادامه به بررسی کدهای این بخش خواهیم پرداخت.

۲ـ۲ـ۱ بررسی کدهای مربوط به مدل مسئله

برای پیاده‌سازی مدل از کتابخانه پایتورچ [۱۴] استفاده شده است. معماری مدل به این صورت است که دو لایه‌ی شبکه‌ی LSTM یکی برای یادگیری عناوین خبر و دیگری برای به کار گرفته شده است. این مدل در برنامه‌ی ۲ـ۴ قابل مشاهده است.

[13] Long Short-Term Memory Networks
[14] pytorch.org

برنامهٔ ۲ـ۴: مدل شبکه‌ی عصبی

```
class LSTMRecom(nn.Module):

    def __init__(self, embedding_weight_titles, embedding_weight_abs,
        hidden_dim, lstm_layers):
        super(LSTMRecom, self).__init__()
        self.hidden_dim = hidden_dim
        self.lstm_layers = lstm_layers

        self.embeddings_title =
            self._create_embedding(embedding_weight_titles)
        self.embeddings_abstract =
            self._create_embedding(embedding_weight_abs)

        self.lstm_title = nn.LSTM(self.embedding_dim, hidden_dim,
            lstm_layers, batch_first=True)
        self.lstm_abstract = nn.LSTM(self.embedding_dim, hidden_dim,
            lstm_layers, batch_first=True)
        self.fc_merge_1 = nn.Linear(hidden_dim * 2, hidden_dim)

        self.fc_imp_title = nn.Linear(self.embedding_dim, hidden_dim)
        self.fc_imp_abs = nn.Linear(self.embedding_dim, hidden_dim)
        self.fc_merge_2 = nn.Linear(hidden_dim * 2, hidden_dim)

        self.fc_clf = nn.Linear(hidden_dim * 2, 2)
        self.softmax = nn.Softmax(dim=1)

```

```
۲۲    def _create_embedding(self, embedding_weight, trainable=False):
۲۳        num_embeddings, embedding_dim = embedding_weight.shape
۲۴        self.embedding_dim = embedding_dim
۲۵        emb_layer = nn.Embedding(num_embeddings, embedding_dim)
۲۶        emb_layer.weights = torch.nn.Parameter(embedding_weight)
۲۷        if trainable:
۲۸            emb_layer.weight.requires_grad = True
۲۹        else:
۳۰            emb_layer.weight.requires_grad = False
۳۱        return emb_layer
۳۲
۳۳
۳۴    def init_hidden(self, batch_size):
۳۵        weight = next(self.parameters()).data
۳۶        hidden_title = (weight.new(self.lstm_layers, batch_size,
              self.hidden_dim).zero_().to('cuda'),
۳۷                  weight.new(self.lstm_layers, batch_size,
                        self.hidden_dim).zero_().to('cuda'))
۳۸
۳۹        weight = next(self.parameters()).data
۴۰        hidden_abstract = (weight.new(1, batch_size,
              self.hidden_dim).zero_().to('cuda'),
۴۱                  weight.new(1, batch_size,
                        self.hidden_dim).zero_().to('cuda'))
۴۲
۴۳        return hidden_title, hidden_title
۴۴
```

```
۴۵      def forward(self, histories, impression, lengths,
            hidden_states_title=None, hidden_states_abs=None):
۴۶          sequence_len = histories.size()[1]
۴۷          titles_embed = self.embeddings_title(histories) #(batch, seq_len,
                embed_size)
۴۸          titles_embed =
                torch.nn.utils.rnn.pack_padded_sequence(titles_embed,
                lengths.cpu(), batch_first=True)
۴۹          lstm_out_title, _ = self.lstm_title(titles_embed,
                hidden_states_title) #(batch, seq_len, hidden_size)
۵۰          lstm_out_title, _ =
                torch.nn.utils.rnn.pad_packed_sequence(lstm_out_title,
                batch_first=True)
۵۱          lstm_out_title = lstm_out_title.sum(dim=1).div(sequence_len)
                #(batch , hidden_size)
۵۲
۵۳          abstracts_embed = self.embeddings_abstract(histories) #(batch,
                seq_len, embed_size)
۵۴          abstracts_embed =
                torch.nn.utils.rnn.pack_padded_sequence(abstracts_embed,
                lengths.cpu(), batch_first=True)
۵۵          lstm_out_abs, _ = self.lstm_abstract(abstracts_embed,
                hidden_states_abs) #(batch, seq_len, hidden_size)
۵۶          lstm_out_abs, _ =
                torch.nn.utils.rnn.pad_packed_sequence(lstm_out_abs,
                batch_first=True)
۵۷          lstm_out_abs = lstm_out_abs.sum(dim=1).div(sequence_len) #(batch ,
```

```
            hidden_size)
58
59          combined_title_abs = torch.cat([lstm_out_title, lstm_out_abs],
                dim=1) #(batch , 2*hidden_size)
60
61          out_merge_1 = self.fc_merge_1(combined_title_abs) #(batch , 128)
62
63
64          title_embed = self.embeddings_title(impression) # (batch,
                embed_size)
65          abstract_embed = self.embeddings_abstract(impression) # (batch,
                embed_size)
66
67          out_title = self.fc_imp_title(title_embed) # (batch, 128)
68          out_abs = self.fc_imp_abs(abstract_embed) # (batch, 128)
69
70          combined_t_a = torch.cat([out_title, out_abs], dim=1) # (batch,
                256)
71          out_merge_2 = self.fc_merge_2(combined_t_a) # (batch, 128)
72
73          combined = torch.cat([out_merge_1, out_merge_2], dim=1)
74          out = self.fc_clf(combined)
75          out = self.softmax(out)
76          return out
```

در خط ۱ یک کلاس با نام LSTMRecom ساخته شده است که منطق مدل ما را در بر میگیرد. این کلاس از کلاس nn.Module ارث می‌برد که در کتابخانه pytorch برای تعریف مدل‌ها ارائه شده است. در

مدل پایتورچ دو تابع اصلی تعریف می‌شود. تابع __init__ که در آن لایه‌های شبکه تعریف می‌شود و تابع forward که در آن روند گذر داده‌ها از لایه‌های مختلف شبکه تنظیم می‌شود.

در خط۸ و ۹ دو لایه‌ی تعبیه ساخته شده است. هدف این لایه‌ها این است که شناسه‌ی خبر را به بردار آن خبر تبدیل کنند. این بردارها در قسمت استخراج ویژگی به دست آمده‌اند.

در خطوط ۱۱ و ۱۲ دولایه‌ی lstm وجود دارند که یکی از آن‌ها برای یادگیری عنوان و دیگری برای یادگیری چکیده‌ی خبرهای مربوط به تاریخچه‌ی فعالیت‌های کاربر است. در خط ۱۳ یک لایه‌ی تمام متصل وجود دارد که خروج دو شکبه‌ی lstm را ترکیب می‌کند.

در خط ۱۵ یک لایه‌ی تمام متصل تعریف شده است که بردار مربوط به عنوان خبر پیشنهاد شده رو در ورودی می‌گیرد. در خط ۱۷ نیز یک لایه‌ی تمام متصل برای دریافت بردار مربوط به چکیده‌ی خبر پیشنهاد شده است.

در نهایت در خط ۱۹ یک لایه‌ی تمام متصل تعریف شده است که خروجی بخش‌های تاریخچه و پیشنهاد را دریافت کرده و با توجه به اینکه دسته‌بندی دو کلاسه داریم دو خروجی می‌دهد. این دو خروجی از تابع softmax عبور می‌کنند تا به مقادیر احتمالاتی تبدیل شوند.

در خط ۳۶ تابع forward تعریف شده که جریان عبوری داده‌ها از لایه‌های مختلف شبکه در این تابع تعریف می‌شود. در خطوط ۴۷ الی ۵۱ شناسه‌های اخبار به لایه‌ی تعبیه وارد شده و سپس به شبکه‌ی lstm مربوط به عناوین وارد می‌شود. در خطوط ۵۳ الی ۵۷ همین کار برای قسمت چکیده انجام می‌شود.

در خطوط ۶۴ الی ۶۸ ورودی‌های مربوط به خبر پیشنهاد شده در لایه‌های مربوطه وارد می‌شوند.

در فصل بعدی به بررسی نتایج خواهیم پرداخت.

فصل ۳

آزمایشات

در این بخش به بررسی اجرای قسمت آموزش مدل و همچنین بررسی نتایج آزمایشات خواهیم پرداخت.

۳ـ۱ آموزش مدل

برای آموزش مدل داده‌ها به سه قسمت آموزش، ارزیابی و تست تقسیم شدند. با توجه به اینکه تعداد نمونه‌های کلاس منفی بسیار بیشتر از تعداد نمونه‌های کلاس مثبت است به منظور میزان شدن کلاس‌ها به صورت تصادفی بخشی از نمونه‌های کلاس منفی حذف شدند تا در نهایت نسبت تعداد نمونه‌ها به تعادل برسد.

پس از انجام عملیات فوق حدود یک میلیون داده‌ی آموزش، ۵ هزار داده‌ی ارزیابی و ۵ هزار داده‌ی تست برای آموزش و تست مدل استفاده شد.

برنامهٔ ۳ـ۱: آموزش مدل

```
best_val_acc = 0
epochs = 3
for epoch in range(epochs):
    hidden_states_title, hidden_states_abs = model.init_hidden(BATCH_SIZE)
    # hidden_states_title, hidden_states_abs = None, None
    time_diff = 0
```

```
    epoch_loss = 0
    epoch_acc = 0
    val_acc = 0
    val_loss = 0
    total_batches = int(len(train_histories)/BATCH_SIZE)
    for i_batch, i in enumerate(range(0, len(train_histories),
        BATCH_SIZE)):
        t1 = time()
        hists = train_histories[i:i+BATCH_SIZE]
        if len(hists) < BATCH_SIZE:
            continue
        hists = [torch.tensor(h) for h in hists]
        lenghts = [len(h) for h in hists]
        lenghts = torch.tensor(lenghts, dtype=torch.int32)
        hists = pad_sequence(hists, batch_first=True)
        hists = hists.to('cuda')
        imps = train_impressions[i:i+BATCH_SIZE]
        imps = torch.tensor(imps)
        imps = imps.to('cuda')
        labels = train_labels[i:i+BATCH_SIZE]
        labels = torch.tensor(labels)
        labels = labels.to('cuda')
        model.zero_grad()
        preds = model(hists, imps, lenghts, hidden_states_title,
            hidden_states_abs)
        loss = loss_function(preds, labels)
        acc = binary_accuracy(preds, labels)
```

```
        epoch_loss += loss
        epoch_acc += acc
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), (5.0
        optimizer.step()
        time_diff = time()-t1
        remaining_time = time_diff*total_batches-time_diff*(i_batch)
        print('\r>> epoch {}, batch {}/{}, remaining time 5.:}f}s, loss
            5.:}f}, acc 5.:}f}, val loss 5.:}f}, val acc 5.:}f}'
        .format(epoch, i_batch, total_batches, remaining_time, loss, acc,
            val_loss, val_acc), end='')

        if i_batch % 100 == 0:
            val_loss = 0
            with torch.no_grad():
                loss = 0
                steps = int(len(val_histories)/BATCH_SIZE)
                for j in range(0, len(val_histories), BATCH_SIZE):
                    hists = val_histories[j:j+BATCH_SIZE]
                    hists = [torch.tensor(h) for h in hists]
                    lenghts = [len(h) for h in hists]
                    lenghts = torch.tensor(lenghts, dtype=torch.int32)
                    hists = pad_sequence(hists, batch_first=True)
                    hists = hists.to('cuda')
                    imps = val_impressions[j:j+BATCH_SIZE]
                    imps = torch.tensor(imps)
                    imps = imps.to('cuda')
```

```
                labels = val_labels[j:j+BATCH_SIZE]
                labels = torch.tensor(labels)
                labels = labels.to('cuda')
                tag_scores = model(hists, imps, lenghts)
                val_loss += loss_function(tag_scores, labels)
                val_acc += binary_accuracy(tag_scores, labels)
        val_loss /= steps
        val_acc /= steps
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            torch.save(model, 'model')

    print()
    print('>> epoch {} avg loss 5.:}f}, avg acc 5.:}f},val loss 5.:}f},
        val acc 5.:}f}'
        .format(epoch, epoch_loss/total_batches, epoch_acc/total_batches,
            val_loss, val_acc))
```

در کد ۳ـ۱ ابتدا یک چرخه برای تعداد دورها [1] آورده شده است که در داخل آن یک چرخه‌ی دیگر برای حرکت بر روی دیتاست وجود دارد. در هر قدم این چرخه بخشی از داده‌های آموزشی به اندازه‌ی سایز دسته انتخاب می‌شوند و پس از انجام پیش‌پردازش‌های نهایی اعم از تبدیل کردن به تنسور پایتورچ و انتقال به پردازشگر گرافیک ، به مدل داده می‌شوند تا مدل خروجی مربوط را اعلام نماید. پس از دریافت خروجی مدل از تفاوت بین آن و برچسب‌های واقعی مقدار خطا محاسبه شده و در شبکه بازگشت داده می‌شود.

همچنین در هر ۱۰۰ قدم یکبار قسمت ارزیابی دیتاست در مدل اعمال می‌شود. بهترین مدل بر اساس دقت این قسمت انتخاب می‌شود.

پس از آموزش مدل نوبت به ارزیابی میرسد. برای این قسمت از ۵ هزار نمونه تست استفاده شده است.

[1] epoch

برنامهٔ ۳ـ۲: ارزیابی مدل

```
from sklearn.metrics import f1_score, confusion_matrix, roc_curve, auc,
    roc_auc_score

model = torch.load('model')
positive_class_probs = []
all_preds = []
test_loss = 0
test_acc = 0
steps = int(len(test_histories)/BATCH_SIZE)
with torch.no_grad():
    for j in range(0, len(test_histories), BATCH_SIZE):
        hists = test_histories[j:j+BATCH_SIZE]
        hists = [torch.tensor(h) for h in hists]
        lenghts = [len(h) for h in hists]
        lenghts = torch.tensor(lenghts, dtype=torch.int32)
        hists = pad_sequence(hists, batch_first=True)
        hists = hists.to('cuda')
        imps = test_impressions[j:j+BATCH_SIZE]
        imps = torch.tensor(imps)
        imps = imps.to('cuda')
        labels = test_labels[j:j+BATCH_SIZE]
        labels = torch.tensor(labels)
        labels = labels.to('cuda')
        preds = model(hists, imps, lenghts)
        all_preds.extend(preds.cpu().numpy())
        positive_class_probs.extend([p for p in preds[:,
```

```
            -1].cpu().numpy()])
        test_loss += loss_function(preds, labels)
        test_acc += binary_accuracy(preds, labels)
test_loss /= steps
test_acc /= steps

auc = roc_auc_score(test_labels, positive_class_probs)
fpr, tpr, _ = roc_curve(test_labels, positive_class_probs)
preds = np.argmax(np.array(all_preds).reshape(-1, 2), axis=1)

print('test loss {} test acc {} test auc {}'.format(test_loss, test_acc,
    auc))
print('confusion matrix:')
print(confusion_matrix(test_labels, preds))
```

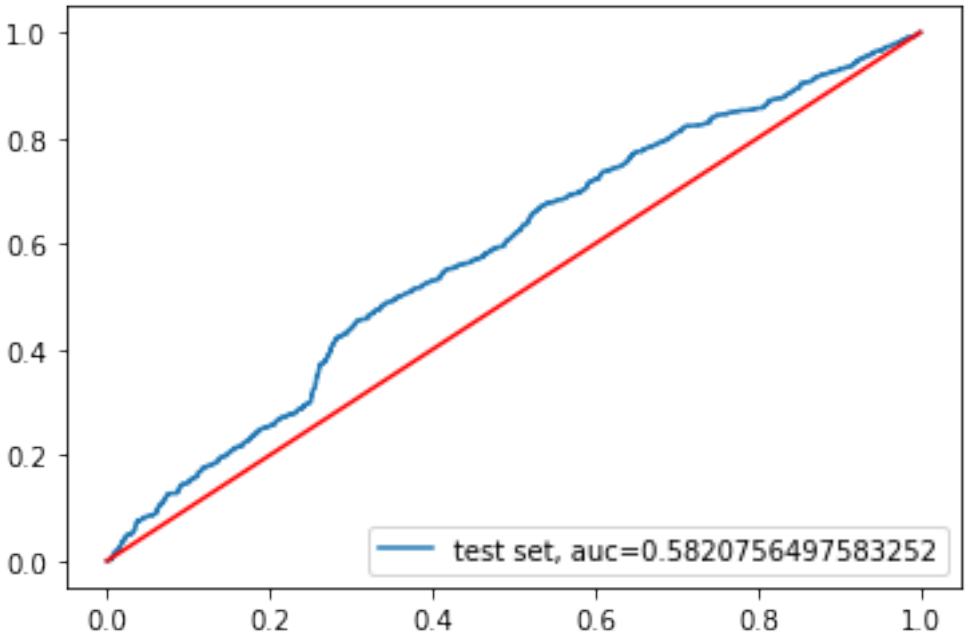
کد ۳ـ۲ بسیار شبیه به فاز آموزش است و نیازی به بررسی مجدد ندارد. بعد ازدریافت خروجی مدل انواع ارزیابی‌ها انجام شد که نتایج آن در جدول ۳ـ۱ و ۳ـ۲ و همچنین نمودار ۳ـ۱ قابل مشاهده است.

نام معیار	مقدار
accuracy	0.57171
auc	0.58207
f1-score	0.30297

جدول ۳ـ۱: نتایج ارزیابی‌ها

		برچسب واقعی	
		منفی	مثبت
برچسب پیش‌بینی	منفی	۲۳۴۵	۴۵۲
	مثبت	۱۷۲۹	۴۷۴

جدول ۳ـ۲: ماتریس در هم ریختگی



شکل ۳ـ۱: نمودار roc

همانطور که مشخص است این مدل عملکرد خوبی نداشته است. با اینکه تعداد نمونه‌های منفی کاهش پیدا کرده بودند تا مدل به سمت آن‌ها تمایل پیدا نکند ولی باز در نتایج مشاهده می‌کنیم که به نمونه‌های منفی تمایل دارد.تمامی مراحل آزمایش از جمله استخراج ویژگی‌ها ، طراحی مدل و ... مستعد به وجود آمدن این مشکل هستند که بررسی آن‌ها نیازمند زمان بیشتری است. به عنوان مثال به علت محدودیت منابع سخت افزاری فقط بخش کوچکی از مجموعه داده استفاده شد که ممکن است علت عملکرد ضعیف مدل این موضوع باشد.

فصل ۴

نتیجه‌گیری

در این پروژه یک مدل مبتنی بر شبکه‌ی عصبی برای مسئله پیشنهاد اخبار ارائه شد. این مدل قادر است تا هم از محتوای آیتم‌ها و هم از عملکرد کاربران برای پیشنهاد خبر استفاده کند اما بر خلاف فرضیات مدل عملکرد خوبی نداشته است که نیازمند بررسی بیشتری برای این موضوع هستیم. از جمله علل احتمالی آن می‌توان به کم بودن داده‌های آموزشی و ویژگی‌های نامناسب، اشاره کرد.

مراجع

[1] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[2] Lü, L., Medo, M., Yeung, C. H., Zhang, Y.-C., Zhang, Z.-K., and Zhou, T. Recommender systems. *Physics Reports 519*, 1 (2012), 1 – 49. Recommender Systems.

[3] Resnick, P., and Varian, H. R. Recommender systems. *Communications of the ACM 40*, 3 (1997), 56–58.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. in *Advances in Neural Information Processing Systems* (2017), I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds. , volume 30, Curran Associates, Inc., pp. 5998–6008.

[5] Wu, F., Qiao, Y., Chen, J.-H., Wu, C., Qi, T., Lian, J., Liu, D., Xie, X., Gao, J., Wu, W., and Zhou, M. MIND: A large-scale dataset for news recommendation. in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (Online, July 2020), Association for Computational Linguistics, pp. 3597–3606.