دانشگاه صنعتی امیرکبیر

دانشکده‌ی علوم‌کامپیوتر

گردآورنده:

شیده هاشمیان

شماره دانشجویی:

۹۶۱۳۴۲۹

تمرین سوم درس پردازش زبان طبیعی

عنوان : مدل زبانی مبتنی بر شبکه‌های عصبی بازگشتی

استاد درس: دکتر اکبری

پاییز ۹۹

این پیاده‌سازی متشکل از چهار فایل `py.` است که هریک از آن‌ها و توابع موجود در آن‌ها در زیر توضیح داده شده است. و برای اجرای برنامه، آدرس محل `train.csv` و `valid.csv` را در `given_data_root_path` که در فایل `constant.py` و `Prepairing_and_training.ipynb` وجود دارد وارد کنید. همچنین تمام فایل‌های ساخته شده در برنامه، در پوشه‌ای که آدرس آن رشته‌ی `document_root_path` (در فایل `constant.py` و `Prepairing_and_training.ipynb`) است قرار دارند.

۱. **ثابت‌ها** (constant.py):

این فایل شامل متغییرهایی است که در دیگر فایل‌ها مورد استفاده قرار می‌گیرند و در میان آن‌ها یکسان است که شامل آدرس پیکره‌های اولیه، آدرسی که برنامه فایل‌هایی که در طول اجرا تولید می‌کند در آن آدرس ذخیره کند هست. همچین مجموعه‌ای از علائم نگارشی و علائم غیر الفبایی (به‌غیر از نقطه، علامت سوال و اعداد) که در پیکره‌ی آموزش موجود بود هست که در مرحله‌ی نرمال‌سازی متن از آن‌ها استفاده شود. علاوه بر این‌ها شامل ثابت عددی محدودسازی مجموعه لغت که در دیگر بخش‌ها مورد استفاده قرار می‌گیرد هست.

۲. **ابزارهای پردازش زبان**(LP_toolkits.py):

این فایل متشکل از سه تابع است.

- تابع sub_alphabets که برگرفته شده از تابعی با همین نام در پکیج parsivar هست باتوجه به نیاز در این برنامه در برخی از قسمت‌ها عوض شده است که با گرفتن یک رشته در آن تمام حروفی که در این برنامه برای ما معنی‌دار هستند را به یک مجموعه حروف مشخص map می‌کند تا کلماتی که یک نگارش دارند یکسان شناسایی شوند.
- تابع normalizer نرمال‌سازی ابتدایی که شامل اجرای تابع sub_alphabets بر روی رشته‌های ورودی، حذف علائم نگارشی و علائم غیر الفبایی (به‌غیر از نقطه، علامت سوال، نقطه‌ویرگول و اعداد) و تغییر اعداد به N را انجام داده و سپس رشته‌ی نهایی را به عنوان خروجی بازمی‌گرداند.

۳. **آماده‌سازی، شناخت داده و آموزش مدل** (`Prepairing_and_training.ipynb`):

- تابع `normalizer(news)`: همانند تابع هم‌نام خود در `LP_toolkits.py` عمل می‌کند.
- تابع `sub_alphabets(doc_string)`: همانند تابع هم‌نام خود در `LP_toolkits.py` عمل می‌کند.
- تابع `read_csv_train_data(doc_add)`: با گرفتن رشته‌ای که محل قرار گیری داده‌ی آموزش را نشان می‌دهدو کمک گیری از تابع `sentence_formation(news)` (که در ادامه عملکرد آن توضیح داده می‌شود) لیستی از جمله‌های موجود در داده‌ی آموزش را به‌عنوان خروجی باز می‌گرداند.
- تابع `tokenizer(doc_arr)` : این تابع با گرفتن لیستی از جمله‌های اخبار (همانند خرجی تابع قبل)، تمام جملات را `index` کرده و به‌صورت آرایه‌ای از اعداد در فایلی با نام `indexed_train.pickle` ذخیره می‌کند. به این‌صورت که ابتدا از موجود بودن فایل‌های `char2index.pickle` و `index2char.pickle` اطمینان حاصل می‌کند (در صورت عدم وجود آن‌ها را می‌سازد و دوباره خودش را صدا می‌زند.) و سپس طول میانگین جملات را پیدا کرده و آن‌هایی که کمتر مساوی این مقدار هستند را `index` می‌کند و در انتها آن را ذخیره می‌کند.
- تابع `sentence_formation(news)` : این تابع یک رشته از خبر را گرفته، انتهای هر جلمه را با پیدا کردن مکان‌های ظهور نقطه و علامت سوال پیدا کرده و سپس هر جمله را با اضافه کردن علامت ابتدای جمله (`'\\n'`) و علامت انتهای جمله (`'\\t'`) آن‌ها را در لیستی ذخیره کرده و در انتها این لیست را خروجی می‌دهد.
- کلاس `RNN(nn.Module)` : یک کلاس ارث برده از کلاس مربوطه `pytorch` است.

- تابع `__init__(self, input_size, output_size, hidden_size, num_layers)` :
  - این تابع لایه‌های مدل را تعریف می‌کند که شامل یک لایه `embeding` ، یک لایه `LSTM` که `num_layers` لایه است با تعداد `hidden_size` تا `hidden state` و یک لایه `decoder` (برای دیکود کردن خروجی به بردارهایی اندازه‌ی ورودی) است.
  - تابع `forward(self, input_seq, hidden_state)`: تابعی است که مدل در حین آموزش برای حرکت به جلو از آن استفاده می‌کند.
- تابع `train()` :

ابتدا متغییر های زیر را که برای ساخت یک مدل (ساخته شده از کلاس `RNN`) و تعریف توابع اولیه لازم است را مقداردهی می‌کنیم.

- `hidden_size`: که تعداد `hidden state` ها در مدل را مشخص می‌کند.
- `num_layers` : تعداد لایه‌های `LSTM` که روی هم هستند را مشخص می‌کند.
- `lr` : نرخ آموزش اولیه را مشخص می‌کند.
- `epochs` : تعداد تکرارهای آموزش مدل توسط داده‌ی آموزش را مشخص می‌کند.
- `load_chk` : در صورتی که از فایل مدل و `optimizer` موجود برای ادامه‌ی آموزش استفاده بشود این مقدار `True`و در غیر این صورت این مقدار `False` است.
- `saved_model_path` : درصورتی که مقدار `load_chk` برابر با `True` است، برابر با مسیر مدل ذخیره شده است
- `saved_optimizer_path` : درصورتی که مقدار `load_chk` برابر با `True` است، برابر با مسیر `optimizer` ذخیره شده است.

پس از آن دو لغت‌نامه‌ی `char2index` و `index2char` که در فایلی ذخیره شده بودند را باز و در متغییری هم‌نام ذخیره می‌کند، همین کار را برای داده‌ی آموزش `index` شده هم انجام می‌دهد. حال نمونه‌ای از شبکه‌ی عصبی با پارامترهای تعیین‌شده می‌سازد و در صورتی که مقدار `load_chk` برابر با `True` باشد، مدل را در آن `load` می‌کند. سپس برای محاسبه‌ی `loss` از تابع `nn.CrossEntropyLoss()` یک نمونه می‌سازد. همچنین برای `optimizer` از `torch.optim.Adam` استفاده کرده و در صورتی که مقدار `load_chk` برابر با `True` باشد، `optimizer` را در آن `load` می‌کند.

حال در حلقه‌ای به تعداد `epoch` ها مدل را آموزش می‌دهد. به این صورت که ابتدا یک `hidden state` اولیه تعریف کرده و پس از آن در حلقه‌ای روی جمله‌های داده‌ی آموزش، جمله را به فرمی مناسب برای ورودی دادن به مدل در میاورد و سپس با دادن آن به مدل،`hiden_state` و `output` را خروجی می‌گیرد. از مقایسه‌ی `output` با `target_output` (که از روی جمله‌ی اولیه بدست آمده است) برای مجاسبه‌ی `loss` استفاده کرده و پس از آن گرادیان برای مدل محاسبه می‌شود و به ابتدای حلقه بازمی‌گردد. (همچنین در هر ۱۰۰۰۰۰ جمله، مقدار`loss` و تعداد جملات دیده شده را برای دیدن روند آموزش چاپ می‌کند. این کار در انتهای هر `epoch` هم انجام می‌شود). همچنین در انتهای هر `epoch` مدل و `optimizer` در فایلی ذخیره می‌شوند.

۴. طراحی دسته‌بند (`LanguageModel.py`):

- کلاس `RNN(nn.Module)`: کلاسی همانند کلاسی با همین نام در فایل قبلی است. (برای متفاوت بودن نام فایل‌ها و عدم دسترسی به آن دوباره تعریف شده است.)
- کلاس `LanguageModel` :
  - تابع `__init__(self, lm_checkpoints, char2index_path, index2char_path)` :
    در این تابع با ورودی گرفتن مسیر مدل ذخیره شده(`lm_checkpoints`)، مسیر فایل‌های `char2index.pickle` و `index2char.pickle` تابع `lm_unit` را برای ساختن نمونه‌ای از مدل با استفاده از مسیر داده‌شده صدا می‌زند.
  - تابع `lm_unit(self, ininti_state=None, weights=None)` : این تابه با گرفن مسیر ذخیره شدن مدل، آن را در مدل ساخته شده `load` می‌کند. (با همان تعداد لایه و `state`ای که مدل اصلی ساخته شده بود.)
  - تابع `get_next_state_and_output(self, prefix)` : این تابع رشته‌ای از ابتدای یک جمله را می‌گیرد. سپس مدل را برای حالت ارزیابی آماده می‌کند (صدا زدن تابع `eval()` مدل). سپس علامت ابتدای جمله را اضافه و این رشته را با استفاده از `char2index` به لیستی از اعداد (که حالت `index` شده‌ی رشته است) تبدیل می‌کند. سپس از ابتدا تا ماقبل آخرین عدد این لیست را (با فرمت مناسب) به مدل داده. از `hidden_state` خروجی این مرحله و آخرین عدد لیست برای گرفتن `hidden_state` و `output` در مرحله‌ی انتهایی (`t+1`) استفاده می‌کند. پس از آن با استفاده از تابع `softmax` احتمالات را برای آرایه‌ی `output` حساب کرده و این احتمالات را همراه با `hidden_state` نهایی (که زوجی به‌شکل (h_{r+1}, c_{r+1}) است) را خروجی می‌دهد.
  - تابع `prefix_to_hidden(self, prefix)` : این تابع رشته‌ای از ابتدای یک جمله را می‌گیرد و مانند تابع قبل مدل را برای حالت ارزیابی آماده و رشته‌ی داده‌شده را `index` می‌کند. سپس آرایه‌ی `index` شده (با فرمت مناسب) را به مدل داده و `hidden_state` خروجی را خروجی می‌دهد.
  - تابع `generate_new_sample(self, prefix)` : این تابع رشته‌ای از ابتدای یک جمله را می‌گیرد و مراحل آماده‌سازی ورودی دادن این رشته به مدل و آماده‌سازی مدل را مانند دو تابع قبل انجام می‌دهد. سپس از ابتدا تا یکی قبل از آخرین عدد آرایه‌ی `index` متناظر با این رشته را به مدل داده و `hidden_state` را خروجی می‌گیرد. سپس در حلقه‌ای که با رسیدن به انتهای جمله یا دیدن بیش از ۲۸۰ کاراکتر (بیشرین طول (از نظر تعداد کاراکتر) در جملات با طول میانگین (از نظر تعداد کلمه)) خاتمه می‌یابد. در این حلقه ابتدا `hidden_state` مرحله‌ی قبل همراه با آرایه‌ی یک عضوی که `index` آخرین کاراکتر موجود در جمله‌ی در حال ساخت (که به فرمت مناسب برای ورودی دادن به مدل در آمده است) است را به‌عنوان ورودی به مدل می‌دهد `hidden_state` و `output` و را خروجی می‌گیرد. حال با اعمال تابع `softmax` بر خروجی احتمال وقوع متناظر با هر کاراکتر را می‌گیرد و با اعمال `top k sampling` بر آن، یک `index` (متناظر با یک کاراکتر) را می‌گیریم و آن را به انتهای

لیست متناظر با رشته‌ی index شده اضافه و کاراکتر متناظر با آن را با استفاده از دیکشنری index2char به انتهای رشته اضافه می‌کنیم. در انتها رشته‌ی تولید شده را خروجی می‌دهیم.

- تابع get_probability(self, prefix, total=False) : این تابع رشته‌ای از ابتدای یک جمله و متغیر Booleanای را می‌گیرد. سپس مانند تابع قبل عمل کرده با این تفاوت که تنها آرایه‌ای از احتمال رخداد هر کاراکتر را فقط برای یک کاراکتر بعد از رشته‌ی ورودی می‌گیرد. حال در صورتی که مقدار total برابر با True باشد بیتشرین احتمال در این آرایه و کاراکتر متناظر با index آن را خروجی می‌دهد. در غیر این صورت تنها آرایه‌ی احتمالات را خروجی می‌دهد.
- تابع get_overall_probability(self, full_sentence) : این تابع یک جمله‌ی کامل را به‌عنوان ورودی می‌گیرد و تنها آن را normalize (با استفاده از تابع normalizer() در فایل LP_toolkits.py) می‌کند. سپس با یک حلقه‌ی افزایشی روی prefix این جمله (از prefix با طول یک جزء با معنی شروع کرده و تا رسیدن به انتهای جمله، در هر تکرار یک کاراکتر به prefix اضافه می‌کند)، آن را به‌عنوان ورودی به تابع get_probability با total=True می‌دهد و احتمال متناظر با کاراکتر بعدی درست (آن کاراکتری که متناظر با آن جایگاه از جمله در جمله‌ی ورودی آمده است) را از آن لیست می‌گیرد. لگاریتم این احتمالات بدست آمده در حلقه را باهم جمع کرده و این مجموع را خروجی می‌دهد.

- تابع evaluation(self, full_sentence) :
  - تابع cer(r, h) : با گرفتن دو آرایه از رشته‌ها که یکی رشته‌ی درست (r) و دیگری رشته‌ی تخمین زده شده (h) است را ورودی می‌گیرد و جدول متناظر با آن در برنامه‌نویسی پویا را ساخته و آخرین خانه‌ی آن که نشان دهنده‌ی مقدار Character Error Rate است را خروجی می‌دهد.

این تابع با گرفتن یک جمله‌ی کامل ابتدا آن را نرمال می‌کند سپس اولین بخش معنا دار آن را به عنوان ورودی به تابع generate_new_sample می‌دهد و خروجی این تابع همراه با جمله‌ی اصلی را به تابع cer داده و خروجی آن را در لغت‌نامه‌ی result ذخیره می‌کند. همچنین جمله‌ی تولید شده را به تابع get_overall_probability داده و از خروجی آن برای محاسبه‌ی معیار آشفتگی استفاده می‌کند و آن را هم در result ذخیره می‌کند.

۵. **ارزیابی مدل رو ی ده دا های آموزش**(تابعی در فایل LanguageModel.py):

- تابع generate_sample_using_test_file(test_file_path, model_path) :
  - تابع read_csv_data(doc_add) : مانند تابع مشابه خود در فایل Prepairing_and_training.ipynb عمل می‌کند.
  - تابع sentence_formation(news) : مانند تابع مشابه خود در فایل Prepairing_and_training.ipynb عمل می‌کند

این تابع با گرفتن مسیر مدل ذخیره شده و داده‌ی آزمایش ابتدا داده‌ی آزمایش را (همانند روند این کار برای داده‌ی آموزش)جمله جمله می‌کند سپس مدل زبانی‌ای با استفاده از مسیر داده‌شده می‌سازد.

سپس در حلقه‌ای برای هر یک از جملات داده‌ی آزمایش، با استفاده از تابع evaluation معیارهای سنجش را برای این جمله محاسبه می‌کند. سپس با استفاده از تابع generate_new_sample (و ورودی دادن prefix معنا دار از این جمله) و محاسبه‌ی احتمال آن با استفاده از تابع get_overall_probabilitiey، جمله را در فایل news_sample.txt و احتمال متناظر با آن را در فایل probabilities.txt ذخیره می‌کنیم.

۶. **نتایج:**

- به علت زمان‌بر بودن فرایند دو فایل probabilities.txt و news_sample.txt نتیجه‌ی عملکرد مدل برای هزار جمله‌ی ابتدایی داده‌ی آزمایش است.
همچنین دو معیار ارزیابی برای این تعداد داده به‌صورت زیر است.

```
- CER_avg:    103.585
- perplexity_avg:     0.490
```

که نشان دهنده‌ی این است که مدل عملکرد نسبتا مناسبی دارد (نسبتا پایین بودن مقدار میانگین آشفتگی).

البته به علت زیاد بودن پارامترهای مدل و به نسبت کم بودن دفعات آموزش (۶ تکرار به دلیل بسیار زمان‌بر بودن این فرایند) این مقادیر قابل انتظار بود.