SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing

نکته : تمامی عناوین و کلمات قرمز رنگ به صورت لینک های قابل کلیک هستند .

چکیده

این مقاله SentencePiece را توصیف می‌کند؛ یک توکنایزر و دیتوکنایزر زیرواحدی مستقل از زبان که برای پردازش متون مبتنی بر شبکه‌های عصبی، شامل ترجمه ماشینی عصبی، طراحی شده است. این ابزار پیاده‌سازی متن‌باز C++ و Python برای واحدهای زیرواحدی ارائه می‌دهد. در حالی که ابزارهای موجود برای بخش‌بندی زیرواحدی فرض می‌کنند ورودی پیش‌تر به دنباله‌ای از واژه‌ها توکنیزه شده است، SentencePiece می‌تواند مدل‌های زیرواحدی را مستقیماً از جملات خام آموزش دهد؛ که این امکان را فراهم می‌آورد تا سیستمی کاملاً انتها به انتها (end-to-end) و مستقل از زبان ایجاد شود. ما یک آزمایش اعتبارسنجی بر روی ترجمه ماشینی عصبی انگلیسی-ژاپنی انجام داده‌ایم و مشاهده کردیم که می‌توان دقتی قابل مقایسه با آموزش مستقیم زیرواحدی از جملات خام به دست آورد. همچنین عملکرد آموزش و بخش‌بندی زیرواحدی را در پیکربندی‌های مختلف مقایسه می‌کنیم. SentencePiece تحت مجوز Apache 2 در دسترس است به آدرس:

https://github.com/google/sentencepiece

۱ مقدمه

شبکه‌های عصبی عمیق تأثیر بزرگی بر پردازش زبان طبیعی داشته‌اند. ترجمه ماشین عصبی (NMT) (Bahdanau et al., 2014; Luong et al., 2015; Wu et al., 2016; Vaswani et al., 2017) به‌طور خاص محبوبیت روزافزونی پیدا کرده است، چرا که می‌تواند با استفاده از شبکه‌های عصبی، ترجمه‌ها را به‌طور مستقیم و تنها با یک معماری ساده‌ی انتها به انتها انجام دهد. NMT در چندین وظیفه‌ی مشترک نتایج چشمگیری نشان داده است (Denkowski and Neubig, 2017; Nakazawa et al., 2017)، و رویکرد مؤثر آن بر سایر وظایف مرتبط در NLP مانند تولید گفت‌وگو (Vinyals and Le, 2015) و خلاصه‌سازی خودکار (Rush et al., 2015) نیز تأثیر گذاشته است.

اگرچه NMT به‌طور بالقوه می‌تواند ترجمه‌ی انتها به انتها انجام دهد، بسیاری از سیستم‌های NMT همچنان به پیش‌پردازش‌ها و پس‌پردازش‌های وابسته به زبان متکی هستند، مشابه آنچه در سیستم‌های سنتی ترجمه ماشینی آماری (SMT) استفاده می‌شد. Moses، یک جعبه‌ابزار استاندارد برای SMT، مجموعه‌ای نسبتاً کارآمد از پیش‌پردازش و پس‌پردازش را پیاده‌سازی می‌کند. با این حال، این ابزار بر اساس قواعد دستی و وابسته به زبان ساخته شده است که کارایی آن‌ها برای NMT هنوز اثبات نشده است. علاوه بر این، این ابزارها عمدتاً برای زبان‌های اروپایی طراحی شده‌اند که در آن‌ها کلمات با فاصله از هم جدا می‌شوند. برای آموزش سیستم‌های NMT در زبان‌های بدون جداساز مانند چینی ، کره‌ای و ژاپنی، باید جداکننده‌ی واژه‌ها را به‌صورت مستقل اجرا کنیم.

چنین پردازش‌های وابسته به زبان همچنین آموزش مدل‌های NMT چندزبانه (Johnson et al., 2016) را دشوار می‌سازد، چرا که نیازمند مدیریت دقیق پیکربندی‌های پیش‌پردازش و پس‌پردازش برای هر زبان است؛ در حالی که معماری‌های درونی شبکه‌های عصبی ماهیتی مستقل از زبان دارند. با استاندارد شدن رویکردهای NMT و حرکت به سمت معماری‌های مستقل‌تر از زبان، برای جامعه‌ی NLP اهمیت بیشتری پیدا می‌کند که یک پیش‌پردازش و پس‌پردازش ساده، کارآمد، قابل بازتولید و مستقل از زبان توسعه یابد که بتواند به راحتی در سیستم‌های NLP مبتنی بر شبکه‌های عصبی، شامل NMT، ادغام شود.

در این مقاله، ما SentencePiece را توصیف می‌کنیم؛ یک توکنایزر و دیتوکنایزر متنی ساده و مستقل از زبان که عمدتاً برای سیستم‌های تولید متن مبتنی بر شبکه‌های عصبی طراحی شده است، جایی که اندازه واژگان پیش از آموزش مدل عصبی مشخص می‌شود. SentencePiece دو الگوریتم بخش‌بندی زیرواحدی، یعنی رمزگذاری جفت‌بایتی BPE (Sennrich et al., 2016) و مدل زبانی تک‌واحدی Unigram LM (Kudo, 2018)، را با امکان آموزش مستقیم از جملات خام پیاده‌سازی می‌کند. SentencePiece ساخت یک سیستم کاملاً انتها به انتها را ممکن می‌سازد که به هیچ پردازش وابسته به زبان خاصی نیاز ندارد.

```
% spm_train --input=data/input.txt
  --model_prefix=spm --vocab_size=1000

% echo "Hello␣world." | spm_encode --
      model=spm.model
_He ll o _world .

% echo "Hello␣world." | spm_encode
  --model=spm.model --output_format=id
151 88 21 887 6

% echo "_He␣ll␣o␣_world␣." |
      spm_decode --model=spm.model
Hello world.

% echo "151␣88␣21␣887␣6" | spm_decode
      --model=spm.model
          --input_format=id
Hello world.
```

شکل ۱: کاربرد استفاده خط فرمانِ SentencePiece

۲ مروری بر سیستم

SentencePiece شامل چهار مؤلفه اصلی است: نرمالایزر، آموزنده، کدگذار و کدگشا. نرمالایزر ماژولی است برای نرمال‌سازی کاراکترهای یونیکد هم‌معنی به فرم‌های استاندارد. آموزنده، مدل بخش‌بندی زیرواحدی را از روی پیکره نرمال‌شده آموزش می‌دهد. نوع مدل زیرواحدی به‌عنوان پارامتر آموزنده مشخص می‌شود. کدگذار به‌طور داخلی نرمالایزر را اجرا می‌کند تا متن ورودی را نرمال‌سازی کند و سپس آن را با استفاده از مدل زیرواحدیِ آموزش‌دیده به دنباله‌ای از زیرواحدها توکنیزه می‌کند. کدگشا دنباله زیرواحدی را دوباره به متن نرمال‌شده تبدیل می‌کند. نقش کدگذار و کدگشا به‌ترتیب معادل preprocessing (tokenization) و postprocessing (detokenization) هستند. با این حال، آن‌ها را «کدگذاری» و «کدگشایی» می‌نامیم، چرا که SentencePiece نگاشت واژگان به شناسه (id) را مدیریت می‌کند و می‌تواند متن را مستقیماً به دنباله‌ای از شناسه‌ها و بالعکس تبدیل کند. کدگذاری و کدگشایی مستقیم به/از دنباله‌های شناسه برای اکثر سیستم‌های NMT مفید است، چرا که ورودی و خروجی آن‌ها دنباله‌هایی از شناسه‌ها هستند. شکل ؟ یک نمونه‌ی انتها به انتهای آموزش (spm_train)، کدگذاری (spm_encode) و کدگشایی (spm_decode) را نشان می‌دهد. می‌بینیم که متن ورودی به‌صورت برگشت‌پذیر از طریق spm_encode و spm_decode تبدیل می‌شود.

۳ طراحی کتابخانه

این بخش طراحی و جزئیات پیاده‌سازی SentencePiece را همراه با نمونه‌هایی از خط فرمان و کد شرح می‌دهد.

۱.۳ توکنیزاسیون بدون از دست‌دادن اطلاعات (**Lossless Tokenization**)

جمله خام و جمله توکنیزه‌شده زیر نمونه‌ای از پیش‌پردازش وابسته به زبان هستند:

- متن خام: Hello world.
- توکنیزه‌شده: [Hello][world][.]

یک نکته این است که متن خام و دنباله توکنیزه‌شده به‌صورت برگشت‌پذیر قابل تبدیل نیستند. برای مثال، این‌که بین «world» و «.» فاصله‌ای وجود نداشته، در دنباله توکنیزه‌شده نگهداری نمی‌شود. بنابراین، دیتوکنایزیشن (بازسازی متن خام از دنباله توکن‌ها) باید به‌صورت وابسته به زبان انجام شود، چرا که این عملیات برگشت‌ناپذیر هستند. برای مثال، در زبان‌های اروپایی دیتوکنایزر معمولاً فاصله‌ها را بین توکن‌ها اضافه می‌کند، در حالی که در زبان‌هایی مثل ژاپنی یا چینی نیازی به فاصله وجود ندارد.

- متن خام: [こんにちは世界。](Hello world.)
- توکنیزه‌شده: [こんにちは][世界][。]

چنین پردازش‌های وابسته به زبان معمولاً با قواعد دستی پیاده‌سازی شده‌اند که نوشتن و نگهداری آن‌ها پرهزینه است. SentencePiece دیتوکنایزر را به‌عنوان عمل معکوس کدگذار پیاده‌سازی می‌کند، به این صورت:

$$Decode(Encode(Normalize(text))) = Normalize(text)$$

ما این طراحی را «توکنیزاسیون بدون از دست‌دادن اطلاعات» می‌نامیم، چرا که همه اطلاعات لازم برای بازتولید متن نرمال‌شده در خروجی کدگذار حفظ می‌شود. ایده اصلی این روش آن است که متن ورودی صرفاً به‌عنوان دنباله‌ای از کاراکترهای یونیکد در نظر گرفته شود. حتی فاصله نیز به‌عنوان یک نماد عادی پردازش می‌شود. برای شفافیت بیشتر، SentencePiece ابتدا فاصله‌ها را با نماد متا (U+2581)_ جایگزین می‌کند و سپس متن را به دنباله‌ای دلخواه از زیرواحدها توکنیزه می‌کند. برای مثال:

- متن خام: Hello world.
- توکنیزه‌شده: [Hello][_wor][ld][.]

از آن‌جا که فاصله در متن توکنیزه‌شده حفظ می‌شود، می‌توان توکن‌ها را بدون ابهام بازسازی کرد. برای نمونه در پایتون:

```
detok = ''.join(tokens).replace('_', ' ')
```

لازم به ذکر است که ابزار subword-nmt نمایش متفاوتی برای واحدهای زیرواحدی دارد. این ابزار بر نحوه بخش‌بندی واژه به زیرواحد تمرکز می‌کند و از علامت @@ به‌عنوان مرز درون‌واژه‌ای استفاده می‌کند:

- توکنیزه‌شده: [Hello][wor][@@ld][@@.]

این نمایش همیشه قادر به انجام توکنیزاسیون بدون از دست‌دادن اطلاعات نیست، چرا که در برخورد با فاصله‌ها ابهام باقی می‌ماند. به‌طور خاص، امکان رمزگذاری فاصله‌های پیاپی در این نمایش وجود ندارد.

۲.۳ آموزش و بخش‌بندی زیرواحدی کارآمد

ابزارهای موجود برای بخش‌بندی زیرواحدی، مدل‌های زیرواحدی را از جملات پیش‌توکنیزه‌شده آموزش می‌دهند. این پیش‌توکنیزاسیون در ابتدا برای سرعت‌بخشیدن به آموزش زیرواحد معرفی شد (Sennrich et al., 2016). با این حال، همیشه نمی‌توان فرض کرد که پیش‌توکنیزاسیون در دسترس است، به‌ویژه برای زبان‌های بدون جداساز. علاوه بر این، پیش‌توکنیزاسیون باعث می‌شود که انجام توکنیزاسیون بدون از دست‌دادن اطلاعات دشوار گردد. SentencePiece چندین تکنیک افزایش سرعت هم برای آموزش و هم برای بخش‌بندی به کار می‌گیرد تا امکان توکنیزاسیون بدون از دست‌دادن اطلاعات روی داده‌های خام حجیم فراهم شود. برای مثال، اگر جمله (یا کلمه) ورودی طول N داشته باشد، بخش‌بندی مبتنی بر BPE در هر تکرار نیازمند هزینه محاسباتی $O(N^2)$ است، چرا که باید جفت نمادها را در هر مرحله جست‌وجو کرد. SentencePiece الگوریتمی با پیچیدگی $O(N \log N)$ به کار می‌گیرد که در آن نمادهای ادغام‌شده توسط یک صف اولویت‌دار (binary heap) مدیریت می‌شوند. علاوه بر این، پیچیدگی زمانی آموزش و بخش‌بندی در مدل‌های زبانی تک‌واحدی (unigram language models) متناسب با اندازه داده‌های ورودی (یعنی خطی) است.

۳.۳ مدیریت شناسه واژگان

SentencePiece نگاشت واژگان به شناسه (id) را مدیریت می‌کند تا بتوان متن ورودی را مستقیماً به دنباله‌ای از شناسه‌ها تبدیل کرد و برعکس. اندازه واژگان با استفاده از پرچم –vocab_size=<size> در دستور spm_train مشخص می‌شود. در حالی که ابزار subword-nmt تعداد عملیات ادغام را تعیین می‌کند، SentencePiece اندازه نهایی واژگان را

تعیین می‌کند، چرا که تعداد عملیات ادغام یک پارامتر اختصاصی برای BPE است و نمی‌تواند برای الگوریتم‌های بخش‌بندی دیگر مثل مدل زبانی تک‌واحدی (Kudo, 2018) به‌کار گرفته شود. SentencePiece تعدادی شناسه ویژه برای نمادهای متا رزرو می‌کند، از جمله: نماد ناشناخته <unk>، نماد آغاز جمله <s>، نماد پایان جمله </s> و نماد پرکننده <pad>. شناسه دقیق این نمادها با استفاده از پرچم‌های خط فرمان تنظیم می‌شود. همچنین می‌توان نمادهای متای سفارشی تعریف کرد تا اطلاعات متنی (contextual information) را به صورت توکن‌های مجازی کدگذاری کنند. برای مثال، شاخص‌های زبانی مانند <2ja> و <2de> برای مدل‌های چندزبانه (Johnson et al., 2016) به‌کار می‌روند.

```
U+41 U+302 U+300 <tab> U+1EA6
U+41 U+302 U+301 <tab> U+1EA4
  ...
```

شکل ۲: قانون نرمال‌سازی سفارشی در قالب TSV

۴.۳ نرمال‌سازی قابل‌سفارشی‌سازی کاراکترها

نرمال‌سازی کاراکترها یک گام پیش‌پردازشی مهم برای پردازش متون واقعی است؛ متونی که شامل کاراکترهای یونیکد هم‌معنی از نظر معنایی هستند. برای مثال، در زبان ژاپنی، کاراکترهای لاتین تمام‌عرض (fullwidth) می‌توانند به کاراکترهای لاتین استاندارد (ASCII) نرمال شوند. همچنین تبدیل حروف بزرگ به کوچک (lowercasing) نیز بسته به کاربرد می‌تواند نوعی نرمال‌سازی مؤثر باشد. به‌طور سنتی، نرمال‌سازی کاراکترها با مجموعه‌ای از قواعد دستی پیاده‌سازی می‌شد. در سال‌های اخیر، Unicode Normalization Forms (مانند NFC و NFKC) به‌طور گسترده‌ای در برنامه‌های NLP مورد استفاده قرار گرفته‌اند، چرا که تکرارپذیری بالاتری دارند و به عنوان بخشی از استاندارد یونیکد به‌خوبی پشتیبانی می‌شوند. به‌صورت پیش‌فرض، SentencePiece متن ورودی را با استفاده از نرمال‌سازی NFKC یونیکد نرمال می‌کند. قوانین نرمال‌سازی با استفاده از پارامتر خط فرمان –normalization_rule_name=nfkc در دستور spm_train مشخص می‌شوند. فرآیند نرمال‌سازی در SentencePiece به‌صورت نگاشت رشته به رشته (string-to-string mapping) و با قاعده‌ی «بیشترین تطابق از چپ» (leftmost longest matching) پیاده‌سازی شده است. این قوانین نرمال‌سازی در قالب یک ترنسدیوسر حالت متناهی (finite state transducer) از نوع Aho–Corasick automaton کامپایل می‌شوند تا نرمال‌سازی به شکل کارآمدی انجام شود. SentencePiece همچنین از قوانین نرمال‌سازی سفارشی پشتیبانی می‌کند که می‌توان آن‌ها را به‌صورت فایل TSV تعریف کرد. شکل ۲ نمونه‌ای از چنین فایل TSV را نشان می‌دهد. در این مثال، دنباله یونیکد [U+41 U+302 U+300] به کاراکتر U+1EA6 تبدیل می‌شود. زمانی که در تبدیل، ابهامی وجود داشته باشد، طولانی‌ترین قاعده اعمال می‌شود. فایل‌های TSV تعریف‌شده توسط کاربر با استفاده از پرچم –normalization_rule_tsv=<file> در دستور spm_train مشخص می‌شوند. قواعد خاص هر وظیفه (task-specific rules) را می‌توان با گسترش قواعد پیش‌فرض NFKC تعریف کرد؛ این قواعد به‌صورت فایل TSV

در بسته SentencePiece ارائه می‌شوند.

۵.۳ مدل‌های خودکفا

در سال‌های اخیر، بسیاری از پژوهشگران مدل‌های ترجمه ماشینی عصبی (NMT) از پیش آموزش‌دیده را برای بهبود قابلیت بازتولید (reproducibility) نتایج آزمایشی خود ارائه کرده‌اند. با این حال، در بیشتر موارد به‌طور دقیق مشخص نمی‌شود که داده‌ها چگونه پیش‌پردازش شده‌اند. (Post, 2018) گزارش کرده است که تفاوت‌های جزئی در طرح‌های پیش‌پردازش می‌توانند باعث تغییرات چشمگیر در امتیاز BLEU شوند. حتی در صورتی که از ابزار Moses استفاده شود، باز هم تضمینی برای بازتولید تنظیمات دقیق وجود ندارد، مگر این‌که پیکربندی‌های مربوط به Moses (مانند نسخه و پارامترهای خط فرمان) به‌طور کامل مشخص شده باشند. به‌طور دقیق‌تر، حتی نرمال‌سازی NFKC ممکن است نتایج متفاوتی تولید کند، بسته به نسخه Unicode که استفاده می‌شود. در حالت ایده‌آل، تمام قوانین و پارامترهای مربوط به پیش‌پردازش باید درون فایل مدل به‌صورت یکپارچه و خودکفا (self-contained) گنجانده شوند تا بتوان شرایط آزمایشی یکسانی را بازتولید کرد، مشروط بر اینکه از همان فایل مدل استفاده شود. مدل SentencePiece به‌گونه‌ای طراحی شده است که کاملاً خودکفا (purely self-contained) باشد. فایل مدل نه‌تنها شامل واژگان (vocabulary) و پارامترهای بخش‌بندی (segmentation parameters) است، بلکه شامل ترنسدیوسر حالت متناهی از پیش کامپایل‌شده (pre-compiled finite state transducer) برای نرمال‌سازی کاراکترها نیز می‌باشد. رفتار SentencePiece فقط توسط فایل مدل تعیین می‌شود و هیچ وابستگی خارجی‌ای ندارد. این طراحی تضمین می‌کند که بتوان بازتولید کامل (perfect reproducibility) را به دست آورد، و همچنین اجازه می‌دهد تا فایل مدل SentencePiece به‌عنوان بخشی از مدل NMT منتشر شود. علاوه بر این، توسعه‌دهندگان SentencePiece می‌توانند قوانین نرمال‌سازی پیش‌فرض را بهبود دهند بدون اینکه نگران از بین رفتن سازگاری (breaking existing preprocessing behaviors) با نسخه‌های قبلی باشند. فایل مدل SentencePiece به صورت فرمت دودویی (binary wire format) از نوع Protocol Buffer ذخیره می‌شود که یک سازوکار پلتفرم‌خنثی (platform-neutral) و قابل توسعه برای سریال‌سازی داده‌های ساختاریافته (serializing structured data) است. Protocol Buffers کمک می‌کنند تا داده‌های ساختاریافته به شکلی امن و قابل اطمینان سریال‌سازی شوند، در حالی که سازگاری با نسخه‌های قبلی (backward compatibility) و قابلیت گسترش (extensibility) نیز حفظ می‌شود.

۶.۳ رابط کتابخانه‌ای برای پردازش در لحظه (**On-the-fly Processing**)

پیش‌پردازش متن معمولاً به عنوان یک فرایند آفلاین در نظر گرفته می‌شود. پیش از آغاز آموزش اصلی NMT، ورودی خام مورد پیش‌پردازش قرار گرفته و توسط یک پیش‌پردازنده مستقل به یک دنباله از شناسه‌ها (id sequence) تبدیل

می‌شود. چنین پیش‌پردازش آفلاینی با دو مشکل همراه است. نخست آنکه ابزارهای مستقل مستقیماً در برنامه‌های NMT که با کاربر در تعامل هستند، ادغام نمی‌شوند؛ این در حالی است که چنین برنامه‌هایی نیاز دارند ورودی کاربر را به‌صورت بلادرنگ (on-the-fly) پیش‌پردازش کنند. دوم آنکه پیش‌پردازش آفلاین، به‌کارگیری روش‌هایی مانند افزایش داده در سطح زیرجمله‌ای (subsentence level data augmentation) و تزریق نویز (noise injection) را دشوار می‌کند؛ روش‌هایی که هدف آن‌ها بهبود دقت و مقاوم‌سازی مدل‌های NMT است. مطالعات متعددی برای تزریق نویز به جملات ورودی از طریق تغییر تصادفی بازنمایی درونی جملات انجام شده است. (Kudo, 2018) روش تنظیم زیرواژه‌ای (sub-word regularization) را پیشنهاد می‌کند که در آن بخش‌بندی زیرواژه‌ای به‌صورت تصادفی در طول آموزش NMT تغییر می‌کند. به‌طور مستقل، (Lample et al., 2017) و (Artetxe et al., 2017) یک خودرمزگذار حذف نویز (denoising autoencoder) را در زمینه‌ی یادگیری دنباله‌به‌دنباله (sequence-to-sequence learning) پیشنهاد کردند، که در آن ترتیب کلمات جمله‌ی ورودی به‌صورت تصادفی تغییر می‌یابد و مدل آموزش می‌بیند تا جمله‌ی اصلی را بازسازی کند. اجرای چنین نمونه‌گیری پویا و تزریق نویزی تنها با پردازش آفلاین دشوار است.

```
#include <sentencepiece_processor.h>
#include <sentencepiece_trainer.h>

SentencePieceTrainer::Train(
  "--input=input.txt␣"
  "--model_prefix=spm␣"
  "--vocab_size=1000");

SentencePieceProcessor sp;
sp.Load("spm.model");

std::vector<std::string> pieces;
sp.Encode("Hello␣world.", &pieces);

std::vector<int> ids;
sp.Encode("Hello␣world.", &ids);

std::string text;
sp.Decode({151, 88, 21, 887, 6}, &text);
```

شکل ۳: C++ API usage

```
import sentencepiece as spm

params = ('--input=input.txt␣'
          '--model_prefix=spm␣'
          '--vocab_size=1000')
spm.SentencePieceTrainer.Train(params)

sp = spm.SentencePieceProcessor()
sp.Load('spm.model')

print(sp.EncodeAsPieces('Hello␣world.'))
print(sp.EncodeAsIds('Hello␣world.'))
print(sp.DecodeIds([151, 88, 21, 887, 6]))
```

شکل ۴: Python API usage

SentencePiece نه‌تنها یک ابزار خط فرمان مستقل برای پیش‌پردازش آفلاین ارائه می‌دهد، بلکه از رابط‌های برنامه‌نویسی کتابخانه‌ای (API) در زبان‌های C++، Python و TensorFlow برای پردازش بلادرنگ (on-the-fly) نیز پشتیبانی می‌کند که به‌آسانی می‌توان آن‌ها را در چارچوب‌های موجود NMT ادغام کرد. شکل‌های ۳، ۴ و ۵ نمونه‌هایی از نحوه‌ی استفاده از رابط‌های C++، Python و TensorFlow را نشان می‌دهند. شکل ۶ نمونه‌ای از کد Python برای تنظیم زیرواژه‌ای (subword regularization) را نمایش می‌دهد، که در آن یک دنباله‌ی زیرواژه‌ای بر اساس مدل زبان تک‌واژه‌ای (unigram language model) نمونه‌برداری می‌شود. می‌توان مشاهده کرد که متن «New York» به‌صورت‌های متفاوتی توکنیزه می‌شود.

```
import tensorflow as tf
import tf_sentencepiece as tfs

model = tf.gfile.GFile('spm.model','rb').read()

input_text = tf.placeholder(tf.string, [None])
ids, lens = tfs.encode(input_text, model_proto=model,
        out_type=tf.int32)
output_text = tfs.decode(ids, lens, model_proto=model)

with tf.Session() as sess:
   text = ['Hello␣world.','New␣York']
   ids_, lens_, output_text_ = sess.run([ids, lens, output_text
         ], feed_dict={input_text:text})
```

شکل ۵: TensorFlow API usage

مدل SentencePiece (فایل مدل model proto) به‌عنوان یک ویژگی از عملگر TensorFlow تعریف شده و درون گراف TensorFlow جاسازی می‌شود؛ به‌این‌ترتیب مدل و گراف به‌صورت کاملاً مستقل و خودکفا در می‌آیند.

```
>>> sp.Load('spm.model')
>>> for n in range(5):
... sp.SampleEncodeAsPieces('New␣York', -1, 0.1)
['_','N','e','w','_York']
['_','New','_York']
['_','New','_Y','o','r','k']
['_','New','_York']
['_','New','_York']
```

شکل ۶: Subword sampling with Python API

در هر فراخوانی از تابع SampleEncodeAsPieces، توکنیزه‌سازی به‌صورت متفاوت انجام می‌شود. برای جزئیات بیشتر درباره‌ی تنظیم زیرواژه‌ای (subword regularization) و ابرپارامترهای نمونه‌برداری آن، به (Kudo, 2018) مراجعه کنید.

۴ آزمایش ها

۱.۴ مقایسه روش های مختلف پیش پردازش

ما کارایی روش‌های مختلف پیش‌پردازش را در ترجمه‌ی انگلیسی-ژاپنیِ مقالات Wikipedia، مطابق با وظیفه‌ی ترجمه‌ی آزاد کیوتو (Kyoto Free Translation Task - KFTT) KFTT ارزیابی کردیم. داده‌های آموزش، توسعه و آزمون در مجموعه‌داده‌ی KFTT به‌ترتیب شامل ۴۴۰ هزار، ۱۱۶۶ و ۱۱۶۰ جمله هستند. ما از GNMT (Wu et al., 2016) به‌عنوان پیاده‌سازی سیستم NMT در آزمایش‌های خود استفاده کردیم. به‌طور کلی، تنظیمات و روش آموزشی توصیف‌شده در (Wu et al., 2016) را دنبال کردیم، با این تفاوت که اندازه‌ی گره (node size) و تعداد لایه‌های LSTM را به‌ترتیب برابر با ۵۱۲ و ۶ در نظر گرفتیم. یک مدل واژه‌ای به‌عنوان سیستم مبنا مورد استفاده قرار گرفت. ما SentencePiece (با مدل زبان تک‌واژه‌ای یا unigram language model) را در دو حالت با پیش‌توکنیزه‌سازی (pre-tokenization) و بدون آن مقایسه کردیم. نسخه‌ی SentencePiece با پیش‌توکنیزه‌سازی در واقع مشابه پیکربندی رایج NMT با استفاده از subword-nmt است. نسخه‌ی SentencePiece بدون پیش‌توکنیزه‌سازی، مدل زیرواژه‌ای را مستقیماً از جملات خام آموزش می‌دهد و از هیچ منبع خارجی استفاده نمی‌کند. برای پیش‌توکنیزه‌سازی زبان انگلیسی و ژاپنی به‌ترتیب از توکنیزه‌کننده‌ی Moses و KyTea استفاده کردیم. همان توکنیزه‌کننده‌ها نیز برای مدل واژه‌ای به‌کار گرفته شدند. ما از امتیاز BLEU حساس به حروف بزرگ و کوچک (Papineni et al., 2002) به‌عنوان معیار ارزیابی استفاده کردیم. از آنجا که جملات خروجی در زبان ژاپنی به‌صورت بخش‌بندی‌شده نیستند، پیش از محاسبه‌ی امتیاز BLEU آن‌ها را با استفاده از KyTea بخش‌بندی کردیم.

جدول ۱ نتایج آزمایش‌ها را نشان می‌دهد. نخست، همان‌طور که در جدول مشاهده می‌شود، بخش‌بندی زیرواژه‌ای با استفاده از SentencePiece به‌طور پیوسته امتیازهای BLEU را در مقایسه با مدل واژه‌ای بهبود می‌دهد. این نتیجه با یافته‌های پژوهش‌های پیشین (Sennrich et al., 2016) سازگار است. دوم، می‌توان مشاهده کرد که پیش‌توکنیزه‌سازی (pre-tokenization) همواره برای افزایش امتیاز BLEU ضروری نیست. در جهت ترجمه از ژاپنی به انگلیسی، میزان بهبود ناچیز بوده و تفاوت معناداری وجود ندارد. در مقابل، در ترجمه از انگلیسی به ژاپنی، امتیاز BLEU با وجود پیش‌توکنیزه‌سازی کاهش یافته است.

بهبودهای چشمگیرتری در امتیاز BLEU زمانی مشاهده می‌شود که :

۱) SentencePiece بر روی زبان ژاپنی اعمال شود

۲) جمله‌ی هدف به زبان ژاپنی باشد.

از آنجا که زبان ژاپنی زبانی بدون بخش‌بندی است، پیش‌توکنیزه‌سازی (pre-tokenization) به‌عنوان یک محدودیت قوی برای تعیین واژگان نهایی عمل می‌کند. می‌توان چنین در نظر گرفت که اثرات مثبت بخش‌بندی بدون نظارت از ورودی

خام، در شناسایی واژگان خاص حوزه در زبان ژاپنی به‌طور مؤثری عمل کرده‌اند.

جفت‌زبان	تنظیم (مبدأ/مقصد)	تعداد واژگان	امتیاز BLEU
ja→en	مدل واژه‌ای (مبنای مقایسه)	80k/80k	۲۸.۲۴
	SentencePiece	8k (shared)	۲۹.۵۵
	SentencePiece w/ pre-tok.	8k (shared)	۲۹.۸۵
	Word/SentencePiece	80k/8k	۲۷.۲۴
	SentencePiece/Word	8k/80k	۲۹.۱۴
en→ja	مدل واژه‌ای (مبنای مقایسه)	80k/80k	۲۰.۰۶
	SentencePiece	8k (shared)	۲۱.۶۲
	SentencePiece w/ pre-tok.	8k (shared)	۲۰.۸۶
	Word/SentencePiece	80k/8k	۲۱.۴۱
	SentencePiece/Word	8k/80k	۱۹.۹۴

جدول ۱: نتایج ترجمه (BLEU٪)

۲.۴ علمکرد بخش بندی

جدول ۲ خلاصه‌ای از عملکرد آموزش و بخش‌بندی در پیکربندی‌های مختلف را نشان می‌دهد. می‌توان مشاهده کرد که سرعت آموزش و بخش‌بندی هر دو روش SentencePiece و subword-nmt در مجموعه‌داده‌ی انگلیسی تقریباً قابل مقایسه است، صرف‌نظر از انتخاب پیش‌توکنیزه‌سازی (pre-tokenization). این نتیجه قابل انتظار است، زیرا زبان انگلیسی زبانی بخش‌بندی‌شده است و فضای جستجو برای استخراج واژگان در آن تا حد زیادی محدود است. در مقابل، SentencePiece بهبودهای قابل توجهی در عملکرد هنگام اعمال بر داده‌های خام زبان ژاپنی (بدون پیش‌توکنیزه‌سازی، w/o pre-tok) نشان می‌دهد. سرعت بخش‌بندی در SentencePiece در این تنظیمات حدود ۳۸۰ برابر سریع‌تر از subword-nmt است. این نتیجه به‌طور قوی از این ادعا پشتیبانی می‌کند که SentencePiece به‌اندازه‌ای سریع است که می‌تواند مستقیماً بر داده‌های خام اعمال شود و در نتیجه، پیش‌توکنیزه‌سازی همیشه ضروری نیست. در نتیجه، SentencePiece به ساخت سامانه‌ای کاملاً داده‌محور و مستقل از زبان کمک می‌کند. سرعت بخش‌بندی SentencePiece حدود ۲۱ هزار جمله در ثانیه برای زبان انگلیسی و ۷۴ هزار جمله در ثانیه برای زبان ژاپنی است، که برای اجرای بلادرنگ (on-the-fly) به‌اندازه‌ی کافی سریع است.

وظیفه	ابزار	پیش‌توکنیزه‌سازی	Japanese (زمان، ثانیه)	English (زمان، ثانیه)
آموزش	subword-nmt	بله	۵۶.۹	۵۴.۱
	SentencePiece	بله	۱۰.۱	۱۶.۸
	subword-nmt	خیر	۵۲۸.۰	۹۴.۷
	SentencePiece	خیر	۲۱۷.۳	۲۱.۸
بخش‌بندی	subword-nmt	بله	۲۳.۷	۲۸.۶
	SentencePiece	بله	۸.۲	۲۰.۳
	subword-nmt	خیر	۲۱۶.۲	۳۶.۱
	SentencePiece	خیر	۵.۹	۲۰.۳
پیش‌توکنیزه‌سازی	KyTea(ja)/Moses(en)		۲۴.۶	۱۵.۸

جدول ۲: برای ارزیابی از پیکره‌ی KFTT شامل ۴۴۰ هزار جمله استفاده شده است. آزمایش‌ها در محیط Linux با پردازنده‌های Xeon 3.5GHz انجام شدند. اندازه‌ی واژگان برابر با ۱۶ هزار در نظر گرفته شده است. برای زبان انگلیسی از توکنیزه‌کننده‌ی Moses و برای زبان ژاپنی از KyTea استفاده شده است. توجه داشته باشید که برای مقایسه‌ی منصفانه بین حالت‌های دارای پیش‌توکنیزه‌سازی و بدون آن، باید زمان صرف‌شده برای پیش‌توکنیزه‌سازی نیز در نظر گرفته شود. از آنجا که subword-nmt بر پایه‌ی روش BPE است، ما در SentencePiece نیز از مدل BPE استفاده کردیم. مشاهده کردیم که مدل‌های BPE و مدل زبان تک‌واژه‌ای (unigram language model) عملکردی تقریباً مشابه دارند.