逆順 Trie による効率的な Double-Array 言語モデル

† 乗松 潤矢　　田中 透　　山本 幹雄
筑波大学 システム情報工学研究科
†norimatsu@mibel.cs.tsukuba.ac.jp

1　はじめに

言語モデルは、統計的機械翻訳をはじめとして様々に応用されており、自然言語処理において重要なコンポーネントの一つである。言語モデルは学習に使うデータが多いほど言語モデルの性能が向上することが知られているが (Brants et al., 2007)、トレードオフとしてモデルサイズも肥大化する。

肥大化した言語モデルは大量の物理メモリを必要とするだけでなく、クエリ時の処理速度が悪化するなどの問題を引き起こす。この問題に対処するために、効率的な言語モデルの実装について、盛んに研究されてきた (Germann et al., 2009; Heafield, 2011; Yasuhara et al., 2013)。

本稿では Double-Array を用いた高速かつコンパクトな言語モデルである DALM(Yasuhara et al., 2013) をベースとし、さらなる効率化を検討する。DALM は、Double-Array を用いて Trie 構造を構築したうえで、Double-Array の隙間に言語モデルのパラメータを保持するが、backoff weights だけは外部配列に格納せざるを得なかった。もし、backoff weights を直接 Double-Array の構造内に格納することができれば、従来手法の backoff weights 格納用配列を削除することができ、よりコンパクトな言語モデルを実現できる。また、言語モデルが保持する Trie のノード数を減らすことができれば、DALM はさらにコンパクトになる。

本稿では、DALM の保持する Trie を逆順 Trie(Bell et al., 1990; Heafield, 2011) に変更する。逆順 Trie に変更することで、上記の 2 点を達成し、従来手法よりコンパクトな言語モデルが実現する。

本手法を用いた実験では、従来手法と比べてサイズとクエリ時の処理速度のバランスで提案手法が優れていることを示す。

2　関連研究

2.1　Backwards suffix trees

多くの言語モデル実装 (Stolcke, 2002; Heafield, 2011; Yasuhara et al., 2013) ではデータ構造として Trie を利用することでコンパクトにメモリ上に情報を格納する。言語モデル実装に Trie を使う場合、backwards suffix trees と逆順 Trie に大別できる。以下順に概説するが、本稿では単語列 w_1^n に対して、w_n を目的単語、w_1^{n-1} をヒストリ単語と呼ぶ。

Backwards suffix trees(Bell et al., 1990; Stolcke, 2002; Germann et al., 2009) は、$n$gram のヒストリ単語列を逆順に並べた Trie である。Trie の各ノードは対応するヒストリに後続する目的単語の一覧へのリンクをそれぞれ持っている。探索時には、ヒストリ単語を辿って得られたノードからつながる目的単語一覧に移動し、クエリとして与えられた目的単語を取得する。

この方法のメリットは、単発クエリに対して高速にバックオフできる点にある。より短いヒストリにしかマッチしない場合、それまでに辿ってきた経路を利用して再計算を防ぐことができることに加え、ヒストリ単語を格納したノードに backoff weights を格納しておけば、バックオフ時に必要となる値も同時に取得できるからである。

2.2　逆順 Trie

逆順 Trie(Bell et al., 1990; Heafield, 2011) は、$n$gram に含まれる全単語を逆順に並べる。逆順 Trie も与えられた $n$gram に最長マッチするノードを効率的に取得できるが、通常のバックオフ時には Trie を辿りなおす必要があり、計算量が比較的多くなる。

しかし、逆順 Trie は言語モデルに状態 (*state*) という概念を導入することで、逐語的なクエリを行う場合は効率的に処理できることが示されている (Heafield, 2011)。統計的機械翻訳ではほとんどのクエリが逐語

的になされるため、この場合に対応できていれば実用上問題ない。

クエリに応じて $n$gram の出現確率を返すには、backoff weights が必要である。通常の逆順 Trie では backoff weights を取得するために木を辿りなおす必要があるが、言語モデルの状態を利用する場合は、クエリ時に辿ったノードに付随している backoff weights を、辿った単語と共に保存しておく。次回クエリ時のバックオフ計算ではその値が活用でき、2 度辿る必要がない。

2.3 Double-Array と DALM

Double-Array(Aoe, 1989) とは、Trie を高速かつコンパクトに表現する手法である。Double-Array では、*Base*、*Check* と呼ばれる 2 本の配列を用いて Trie を表現する。Trie のノードは配列の各要素で表現される。ノード間の単語 w によるリンクは、親ノードのインデックスが i、子ノードのインデックスが j であるとすると以下の式で定義される。

$$j \leftarrow Base[i] + \mathrm{WORDID}(w) \quad if\ Check[j] == i$$

ここで、WORDID(w) は単語 w の単語 ID を返す関数であるとする。単語 ID は正の整数で表現する。

DALM(Yasuhara et al., 2013) は、Double-Array を用いた高速かつコンパクトな言語モデル実装である。DALM では、Double-Array の「高速・コンパクト」という特性を最大限に生かすため、`ordering` 法と `embedding` 法の 2 つの手法が提案されている。

`Ordering` 法は単語 ID を unigram 確率の順に与えることで Double-Array のサイズを小さくする方法である。出現頻度の高い順に単語 ID を与えることで Trie の子ノード格納位置が狭い範囲に集まり、結果としてモデルサイズを小さくする。

`Embedding` 法は Double-Array の隙間に言語モデルの値を埋め込む手法である。図 1 に `Embedding` 法でのノード格納方式を示す。Double-Array の未使用領域を言語モデルの値を格納するために利用する。`Embedding` 法では、*Check* 配列側に backoff weights の対数値を直接格納できず、*Value* 配列に backoff weights を格納し、*Check* 配列には *Value* 配列を参照するための値を格納する。Backoff weights の対数値は正負両方の場合があり、直接格納すると正の数により遷移誤りが発生する可能性があるためである。*Value* 配列がメモリを消費してしまうことは従来手法の課題とされていた。

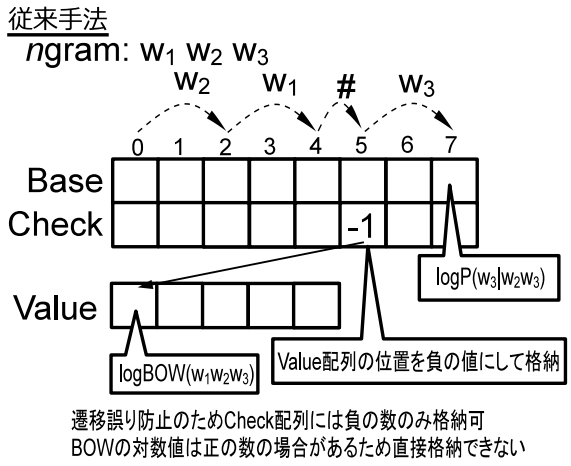

図 1: DALM の `embedding` 法。Double-Array の隙間に対数確率値と backoff weights の位置を指す値を格納する。遷移誤りを防止するため Backoff weights は専用の配列を用意する必要がある

本稿では DALM に使われている Trie を逆順 Trie に変更することで *Value* 配列を削除した。また、格納方式をさらに工夫することで Trie のノード数も削減した。また、逆順 Trie において問題とされていたクエリ時の処理速度低下については、関連研究と同様に言語モデルの状態を取り入れることで悪化を防ぐ。

3 逆順 Trie による DALM

本稿では、DALM の保持する Trie を逆順 Trie に変更し、言語モデルをよりコンパクトに格納できることを示す。逆順 Trie に変更することで、先行研究ではヒストリ単語の最後に置いていた終端記号 $< \# >$ は、逆順に並べた $n$gram 単語列の後に置く。例えば単語列 w_1^3 に対しては、$w_3 \rightarrow w_2 \rightarrow w_1 \rightarrow < \# >$ という順に並べる。

逆順 Trie に修正することで、*Value* 配列を削除できる。図 2 に、単語列 w_1^3 の例を示す。提案手法では、終端記号ノードの *Base* 配列側に backoff weight の対数値、*Check* 配列側に $n$gram の対数確率値を格納する。*Check* 配列側に対数確率値を格納することで、*Check* 配列に格納された Double-Array の遷移に関係ない値は常に負の数となり、遷移誤りを防止できる。

さらに、backoff weight を持たない $n$gram が大量に存在することに着目し、これらの $n$gram に対応する終端記号ノード $< \# >$ を削除する。終端記号ノードが削除可能な条件は以下の 3 つが同時に成立することである。

1. Backoff weight を持たない、または、0.0

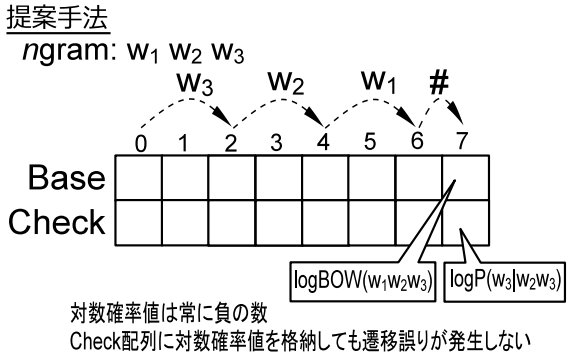

図 2: 逆順 Trie を使うと *Value* 配列を削除できる。図中では backoff weights を BOW と略した。*n*gram の各単語を逆向きに置き、最後に終端記号を置く。対数確率値と Backoff weight を各配列に格納する

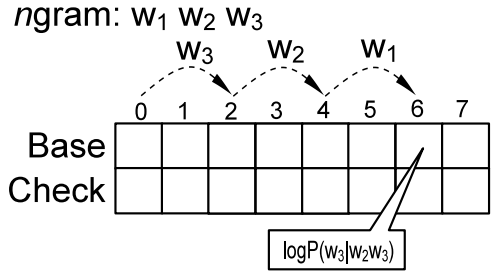

図 3: Backoff weight を持たない *n*gram に対応する終端記号ノードが兄弟ノードを持たない場合、終端記号ノードを削除し、目的単語に対応する *Base* に対数確率値を格納する

2. 該当 *n*gram に後続する単語が存在しない

3. 兄弟ノードが存在しない

条件 1,2 は定義上同義だが、実際には backoff weight が 0.0 にも関わらず後続する単語がモデル中に存在する場合あるため区別が必要である (Heafield, 2011)。

Backoff weight を持たない場合、図 3 のように目的単語に対応するノードの *Base* に対数確率値を格納する。走査する際には、*Base* 値を読み出した時点で正負の判定を行い、負の値であれば対数確率値として扱う。

提案手法ではノードを削除するために先行研究と比べてほぼ同じ処理速度になると予想できる。終端記号ノードが存在しない場合に遷移処理が発生しない代わりにノード遷移時に *Base* 値の正負判定が毎回必要となるためである。

4 実験と考察

4.1 実験

提案手法の有効性を示すため、パープレキシティ計算実験によりモデルサイズと処理速度を計測した。実験には計算機は、Intel(R) Xeon(R) X5672 3.20GHz 8 コア、メインメモリ 141GBytes である。

表 1: 実験に用いた *n*gram モデル。NTCIR4,5 特許検索タスクから抽出したテキストを用いて学習した

オーダー	種類数
1gram	2,980,751
2gram	41,895,814
3gram	181,706,326
4gram	523,043,610
5gram	890,526,684

実験には NTCIR 4,5 特許検索タスク (Fujii et al., 2004, 2005) から抽出した特許文 (1993 年〜2001 年に公開) を用いる。言語モデルは SRILM(Stolcke, 2002) を用いて学習する。得られた *n*gram モデルの詳細を表 1 に示す。このファイルは ARPA 形式で、ディスク上でのファイルサイズは約 66GBytes である。

本実験では、KenLM および DALM の従来手法と比較する。KenLM(Heafield, 2011) は近年標準となりつつある言語モデル実装である。KenLM では、`Trie` と `Probing` という 2 種類のデータ構造を選ぶことができる。この 2 種類のデータ構造はそれぞれメモリ使用量と速度に関してトレードオフの関係にある。

実験にあたり、GitHub に公開されている DALM のソースコード[1]から fork させ、これを拡張する形で実装した。また、Double-Array 言語モデルの構築は、現実的な構築時間を見込んで Trie を 16 分割した。

テストセットには、NTCIR4,5 特許検索タスクより抽出した特許文 (2002 年前半に公開) より 16,134,929 文を用いる。トークン数は 774,026,878、タイプ数は 603,910 である。DALM については、言語モデルの状態を利用したクエリと、利用しないクエリの両方で計測した[2]。実験結果を、図 4 に示す。

4.2 考察

提案手法が速度とサイズのバランスにおいて KenLM や従来手法より良い結果を示した。特にモデルサイズに関しては従来手法の約 3 分の 2 のサイズで格納することができた。これは提案手法の *Value* 配列が削除できたことと、ノード数の削減が影響したと考えられ

[1] https://github.com/jnory/DALM

[2] 詳細について割愛するが、従来の backward suffix trees でも言語モデルの状態を利用することが可能である。

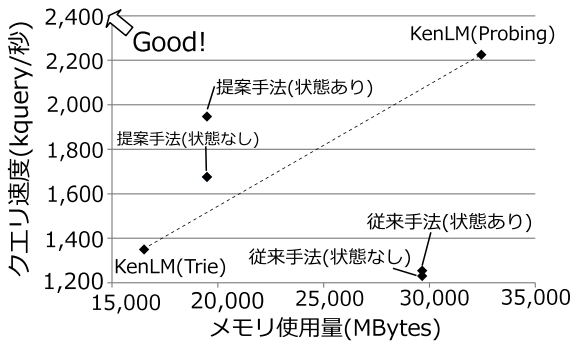

図 4: パープレキシティ測定実験による結果。提案手法が処理速度とモデルサイズのバランスで良い結果を示した

る。速度の面では、言語モデルの状態を利用した場合が最も良い性能を示した。

従来手法と提案手法では処理速度はほぼ同一と予想されるが、実験では大きな速度の違いが見られた。安原らによると、データサイズを大きくしていくことで従来手法と KenLM との差が小さくなっていく現象が見られていた (Yasuhara et al., 2013)。今回の実験では、先行研究での実験よりデータサイズが約 2 倍になっており、KenLM と従来手法の速度差についてはこれが影響した可能性がある。その場合、提案手法の処理速度も従来手法と同程度に落ちることが予想されるが、実験結果では提案手法の方がより高速な動作を示した。この点に関してはより詳細な検討と更なる実験が必要である。

5 おわりに

本稿では Double-Array を用いた言語モデル DALM(Yasuhara et al., 2013) をベースとし、これをさらにコンパクトにする手法を提案した。従来の backwards suffix trees を逆順 Trie に変更することで、*Value* 配列を削除できただけでなく、情報を失わずノードの削除が可能となった。

実験により、モデルサイズが従来手法より 34%削減した。モデルサイズと処理速度のバランスを見ると、KenLM との比較においてより優れた結果を示した。

今後は統計的機械翻訳システムでの実験など、さらなる調査、検討を行いたい。

謝辞

言語モデルの状態については、エディンバラ大学の Hieu Hoang 氏にその存在を教えていただきました。さらに、詳細についてスタンフォード大学の Kenneth Heafield 氏より教えていただきました。この知識がなければ本稿の手法を発案することはできませんでした。また、本稿の実験は、NTCIR4,5 特許検索タスクのデータを利用させていただきました。ご協力いただきました皆様にこの場を借りて心より感謝申し上げます。

参考文献

Jun-ichi Aoe. An Efficient Digital Search Algorithm by Using a Double-Array Structure. *Transactions on Software Engineering1*, 15(9):1066–1077, 1989.

Timothy C Bell, John G Cleary, and Ian H Witten. *Text Compression*. Prentice Hall, 1990.

Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. Large Language Models in Machine Translation. In *Proceedings of the 2007 Joint Conference on EMNLP-CoNLL*, 2007. ACL.

Atsushi Fujii, Makoto Iwayama, and Noriko Kando. Overview of Patent Retrieval Task at NTCIR-4. In *Proceedings of NTCIR-4*, 2004.

Atsushi Fujii, Makoto Iwayama, and Noriko Kando. Overview of patent retrieval task at NTCIR-5. In *Proceedings of NTCIR-5 Workship Meeting*, 2005.

Ulrich Germann, Eric Joanis, and Samuel Larkin. Tightly packed tries: how to fit large models into memory, and make them load fast, too. In *Proceedings of the Workshop on SETQA-NLP*, 2009. ACL.

Kenneth Heafield. KenLM : Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on SMT*, ACL, 2011.

Andreas Stolcke. SRILM-an Extensible Language Modeling Toolkit. *Seventh International Conference on Spoken Language Processing*, 2002.

Makoto Yasuhara, Toru Tanaka, Jun-ya Norimatsu, and Mikio Yamamoto. An Efficient Language Model Using Double-Array Structures. In *Proceedings of the 2013 Conference on EMNLP*, 2013. ACL.