部分転置ダブル配列を用いた $n$gram 言語モデルの実装

竹中 孝介 *1　　芳賀 駿平 *2　　山本 幹雄 *2

*1筑波大学 情報学群 *2筑波大学大学院 システム情報工学研究科

takenaka@mibel.cs.tsukuba.ac.jp

1 はじめに

言語モデルは，自然言語文の流暢さを評価するモデルであり、統計的機械翻訳や音声認識における重要なモデルの一つである．最近学習データの増加にともなうモデルの大規模化に伴い，コンパクトかつ高速な言語モデルの実装技術が盛んに研究されている (Heafield, 2011; Norimatsu et al., 2016)．

言語モデルは，$n$gram と呼ばれる n 個の単語連鎖を入力としてその条件付き確率を返すモジュールである．このため，典型的な探索問題に対する解が基本的な言語モデルの実装手法となる．探索問題には様々な解法があるが，言語モデルの場合は桁探索木であるトライ (TRIE) とハッシュ法による実装が主流である．DALM(Double-Array Language Model) は，TRIE のコンパクトかつ高速な実装であるダブル配列 (Aoe, 1989) をベースとし，ダブル配列の隙間にモデルパラメータを埋め込むことで充填率を高めながら速度とサイズについてバランスのよい実装を実現している (Norimatsu et al., 2016)．しかし，DALM は確率値，バックオフ係数の 2 つのモデルパラメータとインデックスを共通の配列に格納しているため、量子化による圧縮効果が得にくいという問題がある．

本稿では DALM におけるモデルパラメータとインデックスの配列を分離することにより量子化による圧縮を可能にしながら，部分転置ダブル配列を用いることで充填率の悪化を防ぐ言語モデル $_{\mathrm{pt}}$DALM(partly-transposed DALM) を提案する．実際の翻訳実験を行い，ダブル配列の持つ高速かつコンパクトであるという特徴を残しつつ，量子化によるさらなる圧縮が可能であることを示す．

2 ダブル配列と DALM

2.1 ダブル配列

図 1 に示すように，ダブル配列は TRIE の疎行列表現を 2 本の配列で表現したデータ構造であり，疎行列表現の高速性を保ったままコンパクトにできる (Aoe, 1989)．疎行列表現は TRIE のノードを行に割り当て，ノードから子ノードへの遷移先 (子ノードの行番号) を行で表現したものである．行列の列が遷移記号の種類 (数値で表現される) を表し，$O(1)$ で子ノードへ遷移できるため、高速な探索が可能である．しかし，$n$gram 言語モデルに応用した場合，学習する $n$gram のエントリーを TRIE で表すと，子ノードの数が極端に偏るため，行列は疎となり，非現実的なサイズとなってしまう．この無駄をなくすために，縦に各要素がぶつからないようにすべての行を横にずらし，縦につぶして 1 本にまとめ（next 配列と呼ぶ），さらに誤遷移をなくすために親ノードの情報を別の 1 本に格納した配列（check 配列と呼ぶ）を加える．行をずらした長さを記録した offset 配列を加えて 3 本の配列で疎行列を表現する方法がトリプル配列である．さらに，offset 配列の値を next 配列に代入し (これを base 配列と呼ぶ)、check 配列を親ノードの index 値に置き換えたものがダブル配列であり，base 配列と check 配列の 2 本で疎行列を表現する．

ダブル配列の構築は，空の base 配列から始めて、ノード（行）をお互いの子ノード列がぶつからないように順番に base 配列に配置 [1]・格納していくことで行う方法が一般的である．このとき，できるだけ base 配列が小さくなるように、各ノード毎にできるだけ小さな index の場所に配置できる場所を探す．すべての行を base 配列に配置し終わったときに，使用している最大の index 位置がダブル配列の大きさとなる．

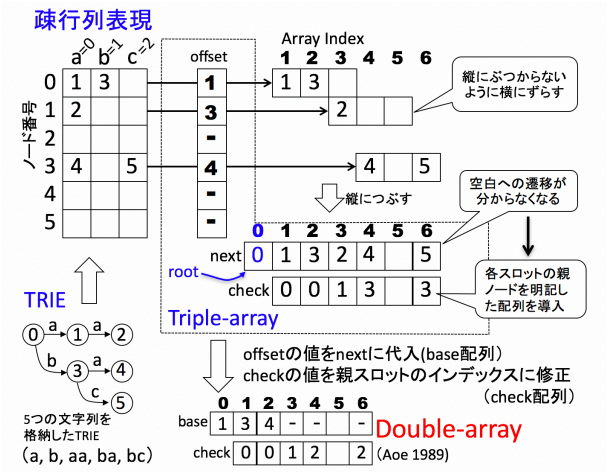

図 1: TRIE の疎行列表現とダブル配列

2.2 ダブル配列を用いた言語モデルの 2 つの実装手法

$n$gram 言語モデルは，key として単語 (ID) の列を受け取り、その確率を返すことが基本であるが，key が存在しない場合は条件付確率の履歴部分のバックオフ係数と呼ばれる近似のための補正値を返す必要がある．また，バックオフ係数は履歴部分に対応する値であるため、$n$gram モデルでは $n-1$ 以下の単語連鎖に対してのみ返せばよい．このように，単語列によって、1 つあるいは 2 つのパラメータを格納する場合がある．

[1]「ノードを配置」するとは，言葉とは少しずれるが「そのノードの子ノード群を配列に挿入する」ことを意味することに注意．配置されるノードは子ノードを持つノードに限られる．

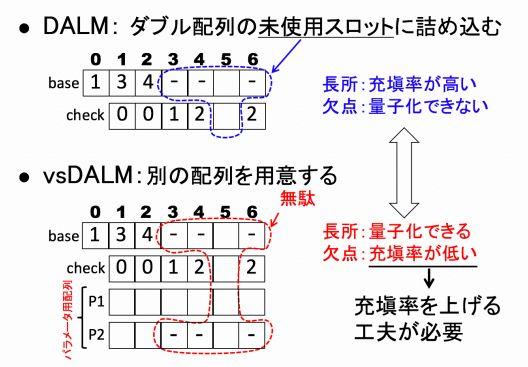

図 2: パラメータ格納場所についての 2 種類の実装手法

ダブル配列で key を表現した場合，モデルパラメータをどこに格納するかが設計上の大きなポイントとなる．DALM はダブル配列中の使われていない部分にパラメータを格納する．各 $n$gram エントリに対して最大 2 つのパラメータがあり，ダブル配列はちょうど 2 本の配列があるため言語モデルとの相性がよい．この手法は，結果的にパラメータも含めて考えた場合の充填率が上がりコンパクトなモデルとなる．しかし、ダブル配列は任意の場所を $O(1)$ でアクセスできる配列の特性を活用しているため，各スロットのサイズを変更できない．このため，たとえ、モデルパラメータを量子化によってサイズを減らしても格納しているスロットを小さくできない．

モデルパラメータの量子化に対応するためには，2 つの値を格納する配列をダブル配列と独立に確保すればよい．これを，従来の DALM と区別するため vsDALM(value-separated DALM) と呼ぶ．しかし，残念ながら、vsDALM の場合、ダブル配列の未使用領域が未使用のまま残り，かつ別配列が必要なため、トータルな充填率が下がってしまうという問題が生じる．図 2 にこの 2 つの実装方法の長所と短所を図解する．

2.3 vsDALM のノード配置順序と leaf-last 手法

vsDALM の充填率を上げるための方法を考える．図 2 の「無駄」と書かれた部分に注目すると次の 2 つの点から改善できることが予想される．

1. 縦に 4 段まったく使われてない部分がある．これは，もともとのダブル配列で隙間が空いてしまった部分で、ダブル配列の構築を工夫することにより改善可
2. 配列の後半部分で最後まで使われていない部分は削除可

以下，それぞれについて述べる．

ノード配置順序

図 1 で説明したように，ダブル配列の構築では、子を持つノードを一つ一つ base 配列に配置していく．このとき，どのような順序で TRIE 中のノードを配置するかによって充填率が変化する．そこで，深さ優先順，幅優先順，子ノードの多い順の 3 種類について配置順による充填率の違いを調べた．その結果，子ノード数順は構築時間がかなり遅くなるが (約十倍)，充填率がほぼ 100%と高かったため、以降では子ノード数順の配置を基本とする．なお，オリジナルな DALM では充填率が低くともその隙間にモデルパラメータを挿入していくため最終的には充填率は高くなる．そのため，配置順はあまり影響せず，かなり大きなモデルでも深さ優先でほぼ 100%の充填率を達成できる．

leaf-last 法

配列のある index 以降に値を保持する必要がない場合はその部分を削除できる．子ノードを持たないノードは base 値を保持する必要がないので，子ノードを持たないノードをダブル配列の後半に連続するように並べれば base 配列の後半部分を削除できる．また，$n$gram モデルの最高次数 $n$gram は子ノードを持たずかつバックオフ係数を持つ必要がないので，$n$gram ノードを配列の後半に配置すれば，バックオフ係数格納用の配列も後半を削除できる．この方法を leaf-last 法と呼ぶ．

配列の先頭から順に配置可能な場所を探すアルゴリズムを仮定すれば，子ノード群がすべて子を持たないノード (すなわち，孫ノードを持たないノード) を後から配置するようにすれば子ノードを持たないノードが最後に連続する可能性が高くなる．そこで，次のような順序でノード集合を配置していけばよい (それぞれの集合は子ノード数順で配置)．(1) 孫ノードを持つノード集合，(2) 孫ノードを持たずかつ少なくとも 1 つの子ノードがバックオフ係数を持つノード集合，(3) 残りのノード集合，の順である．

base 配列のある位置に base 値が格納されると，そこまでは削除できない．上記のアルゴリズムを素直に適用しても，残念ながら base 値を持っているノードがかなり後半に出現してしまい削除できる割合は少ないことが実験的に明らかになった．この点を次節でやや詳細に分析し，部分転置ダブル配列の必要性を説明する．

3 leaf-last 法を用いた vsDALM の性質

前節で述べた leaf-last 法を用いても配列を削減できる割合が少ない原因を明らかにするための実験を行った．ノードを順に配置しているときにダブル配列は徐々に伸びていく．あるノードを配置するとき，それ以前に必要となった配列の範囲 (先頭から最も後ろに配置された子ノードまで) 内に今配置しようとしているノードが配置された場合，配列は伸びない．しかし，その範囲よりもより後ろに配置する場合は、ダブル配列の範囲を広げる必要がある．この様子を可視化するために，横軸にノードの配置順番，縦軸にそのノードを配置した直後の配列の最も後ろの index 値をプロットした．図 3 の (a) の曲線は 1000 万の ngram エントリを持つ言語モデルに対してこのプロットを行った図である．配置順序は leaf-last 法である．

図 3 の (a) の曲線より，子ノード数の大きさの上位 1000 個 (0.02%) のノードによって、ダブル配列の大きさがほぼ決定されてしまい，その他の 99.9%以上のノードは、上記 1000 個で決定された範囲の隙間に配置されていることが分かる．孫ノードを持つノード (約 40%) 以降に伸びた部分が削除可能な部分であり，割合が非常に低いことが分かる．

このようなことが生じる理由は，子ノード数順に配置した場合の最初のノードは極端に子ノード数が多いため，大きなずらし幅が必要なためである．図 3 の曲線 (a) で用いたモデルデータは，$n$gram エントリ数 (TRIE のノード総数-1 と等しい) が約 1000 万，子ノードを持つノード数が約 480 万である．子ノード数上位 1000 個のノードが持つ子ノードの数

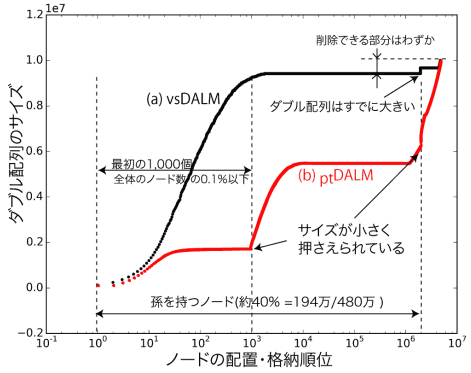

図 3: ダブル配列の成長の様子

は総計約 100 万となり，上位の 0.02%(1000/480 万) のノードの子ノードがノード全体の約 1 割 (100 万/1000 万) を締め，上位 1000 個のノードは平均で 1000 個 (100 万/1000) の子ノードを持つ．語彙サイズは約 10 万であるため行列表現の 1 行は 10 万列からなり，この中に平均 1000 個の子ノードの飛び先が格納されている．約 1%(1000/10 万) の密度は小さく感じるかもしれないが，長さ 10 万の 2 つの配列をぶつからないように重ねて配置するには大きなずらし幅が必要となり，結果的に約 100 万の子ノードを配置するために約 900 万の長さの配列を必要としている．この約 900 万の中で使用されていないスロットは 800 万あり，残りの約 9 割の子ノードはこの隙間に配置されている (図 3 のノード配置順位 10^3 から 10^6 まで配列の長さが伸びていない)．

4 部分転置ダブル配列言語モデル

4.1 部分転置ダブル配列

前節までの検討より，480 万の配置すべきノードのうち最初の 1000 個 (0.02%) の配置で配列が大きく伸びてしまうため，leaf-last 法が有効に働かないことが分かった。これを解決するために，最初の 1000 個の配置で配列が伸びないよう工夫した部分転置ダブル配列 (芳賀他, 2016) を提案している．

疎行列中で問題となる部分は 1000 行 (子ノードが多い 1000 個のノード)×10 万列 (語彙数) の部分である．10 万列から成る 1 行に平均 1000 個の子ノードが入っており，これを横にずらして 1000 行の各要素が縦にぶつからないようにすると非常に長い疎な配列になってしまう点が問題であった．しかし，この疎行列の部分を縦にずらし，横方向に各要素がぶつからないようにすれば比較的密な短い配列にすることができる．縦方向の各列は長さ 1000(行) であり，子ノードが平均 10 個 (100 万/10 万) 入っている．この密度であれば、縦方向のずらし幅を大きくせずとも横方向にぶつからないように配置できることが想像できる．列の数は多い (10 万列) が一つ一つを密に配置していけるので全体として密な短い配列となる．実際には問題の部分を転置することにより，横方向へずらすこれまでと同様な方法で構築できる．

部分転置ダブル配列を作成する例を図 4 に示す．左下の TRIE の疎行列表現の上 2 行を転置して，左上の部分転置表現とする．転置された部分は行が単語種類，列がノード番号と逆になっている．各行を横にずらしてぶつからないように next 配列を決めるところは従来のダブル配列と同じである [2]．

通常のトリプル配列の遷移は next 配列の値 (遷移先のノード番号) から offset 配列の値を引き出し (base 値)，その値に単語 ID を加えることによって遷移先の index が決定される．これに対して，転置部分の遷移は、next 配列のノード番号に単語 ID から決まる offset 値を加えて遷移先の index が決定される．offset の参照についてノード番号と単語 ID がちょうど逆になっている関係である．

next 配列の転置部分に対応するノード番号は変更せず，転置部分ではないノードは offset 配列の値を代入して base 配列を作る．この際，転置部分かそうでないかを区別するために転置部分の base 値は -1 を掛けて負の値とする．すなわち，base 配列値が負の場合はその値の絶対値に遷移単語 ID で引ける offset 値とを加えた先を遷移先とする．check 配列についてはオリジナルのダブル配列と変わらず遷移元の index 値を代入する．

図 4 の右下の部分転置ダブル配列の例では，ルートから'ca' で遷移する場合は次のようになる．まず，ルートの base[0]=-0 であるため、部分転置されたノードであることが分かる (-0 の表現については便宜上である)．よって，遷移先は 0+offset['c']=0+4=4 となり、base[4] に遷移する．check[4]=0 であるためチェックも大丈夫である．次に base[4]=5 で部分転置ノードでないことが分かるための遷移先は base[4]+0('a' の単語 ID)=5 となる．check[5]=4 であるため，遷移に成功し、'ca' という単語列は存在していることが分かる．

部分転置ダブル配列は単語種類毎の offset 配列が付加的に必要となる．しかし，単語種類数は全体のノード数に比べるとわずかであり (1000 万エントリの 5gram で 1%程度)，かつモデルが大規模になる毎に比率は低くなるため，深刻な問題とはならない．

図 3 の (b) の曲線は，最初の 1000 個を転置した場合のダブル配列長の成長の様子を示している．最初の 1000 個に対する配列長は転置しない場合に対して約 2 割と大幅に抑えることができていることが分かる．孫ノードを持つ 194 万のノードを挿入した段階でも転置しない場合に比べて約 6 割の長さであり、削減部分に対応する base 配列およびバックオフ係数用の配列を削減できることを意味する．

4.2 言語モデルへの適用

実際に統計的機械翻訳のための言語モデルを構築する際に必要な実装について述べる．統計的機械翻訳ではデコーダが多くの部分翻訳 (仮説) を作り，言語モデルはその仮説の目的言語での流暢さと言語モデルの state(Li and Khudanpur, 2008; Heafield et al., 2011) を評価する．言語モデルの state は，仮説が左右に伸びていく際により高速に翻訳を行うためのものであり，仮説のリコンバインを行ったり，言語モデルの処理の高速化 (バックオフ計算) に利用される．state を効率的に計算するため，言語モデルでは単語列ごとに単語列が左に伸びるか，右に伸びるかの 2 ビットの情報を格納する必要がある．DALM ではバックオフ係数の値を利用し保存していたが，ptDALM では leaf-last 法によりバックオフ係数配列の一部が削られてしまうため，左に伸びるかど

[2] ただし，転置部分に関しては offset 値が同じ単語があると，存在しない遷移が許されてしまうため，単語種類毎に異なる offset 値を決める必要がある．

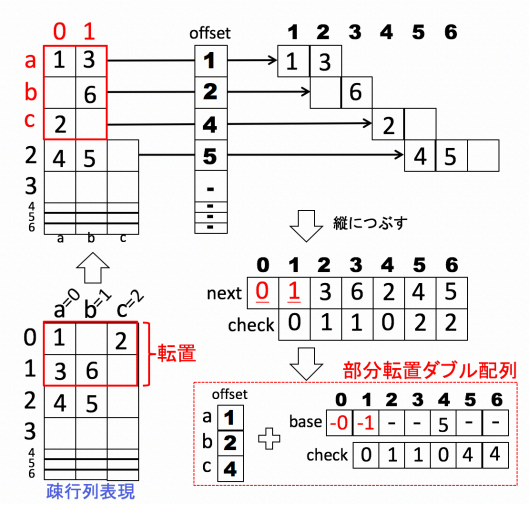

図 4: 部分転置ダブル配列

うかを確率値の最上位ビット，右に伸びるかを check 値の最上位ビットを使用する．

量子化については，binning 法 (Federico and Bertoldi, 2006) を用いた．確率値とバックオフ係数でそれぞれ出現する値をソートし，区間内で出現する値が同数ずつになるように区間を区切り，区間内の値の平均値をとるようにした．

5 評価実験

作成した ptDALM を評価するために，Moses decoder(Koehn et al., 2007) での翻訳にかかる速度を DALM, KenLM(Heafield, 2011) と比較する実験を行った．実験には，CPU に Intel(R) Xeon(R) E5-2680 v3 2.5GHz (2way)，合計 48 コア，メインメモリ 378GBytes の計算機を使用した．実験において，KenLM は Moses decoder 3.0 に付属のもの，DALM は v3.0.0-stable を使用し，KenLM には TRIE を使用した実装の trie, ハッシュ表を用いた実装の probing があるので，それぞれ使用した．また，KenLM probing 構築時のパラメータはデフォルトのものとし，DALM と ptDALM の分割数は 16，ptDALM の転置ノード数は 1 万とした．量子化については，KenLM trie と ptDALM はそれぞれ確率値 7bit，バックオフ係数を 8bit に量子化したものも使用した．

言語モデルは 2005 年の特許データのうち約 590 万文を用いて作成した ARPA ファイル (2 億 5 千万エントリ) より学習し，翻訳モデルは NTCIR10(Goto et al., 2013) の日英対訳特許データ，約 320 万対訳文から学習したものを使用した．モデルの学習で使用していない英文 2300 文の翻訳にかかる時間 (言語モデルやフレーズテーブルのロードの時間は含まない)，最大メモリ使用量 (翻訳全体にかかったメモリ使用量) を各言語モデルに対して調べた．

結果を図 5 に示す．この図より提案法 (ptDALM) は DALM とほぼ同等の速度，メモリ使用量で翻訳を行えていることがわかる．また，量子化により，メモリ使用量を抑えることができている．

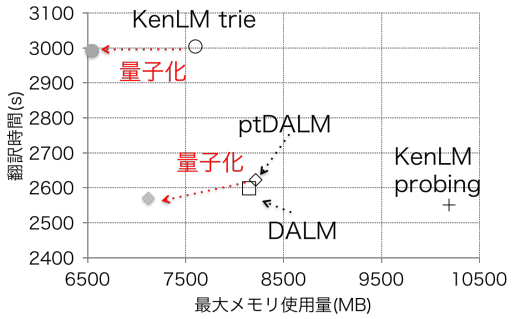

図 5: 翻訳時間とメモリ使用量

6 おわりに

DALM のモデルパラメータとインデックスの配列を分離し，量子化ができるような言語モデルを検討した．配列の分離により悪化したメモリ使用量を部分転置ダブル配列を用いて効率化する ptDALM を提案し，実際の特許翻訳をタスクとして，DALM を含む従来法と翻訳速度およびメモリ使用量の観点で比較を行った．その結果，提案手法は翻訳速度，メモリ使用量ともに DALM と同等で，量子化によるさらなる圧縮が可能であることが分かった．今後の課題として本稿では詳しく述べていないが，ptDALM の分割構築手法の検討があげられる．ダブル配列は構築に時間がかかるため，DALM ではダブル配列を分割し構築している．ptDALM では分割した各ダブル配列ごとに転置するノードを決めるべきか，ノード全体からで転置するノードを決めるべきかなどの分割構築時の転置ノードの扱いについての検討が不十分であるため，詳しく検討していきたい．

参考文献

Aoe, Jun-ichi (1989) “An efficient digital search algorithm by using a double-array structure,” *IEEE Transactions on Software Engineering*, Vol. 15, No. 9, pp. 1066–1077.

Federico, Marcello and Nicola Bertoldi (2006) “How Many Bits Are Needed to Store Probabilities for Phrase-based Translation?” in *Proceedings of the WSMT*, pp. 94–101.

Goto, Isao, Ka Po Chow, Bin Lu, Eiichiro Sumita, and Benjamin K. Tsou (2013) “Overview of the Patent Machine Translation Task at the NTCIR-10 Workshop,” in *10th NTCIR Conference*, pp. 260–286, Tokyo, Japan.

Heafield, Kenneth (2011) “KenLM: Faster and Smaller Language Model Queries,” in *Proceedings of the WSMT*, pp. 187–197, Edinburgh, Scotland.

Heafield, K., H. Hoang, P. Koehn, T. Kiso, and M. Federico (2011) “Left language model state for syntactic machine translation.,” in *IWSLT*, pp. 183–190.

Koehn, Philipp, et al. (2007) “Moses: Open Source Toolkit for Statistical Machine Translation,” in *ACL2007* pp. 177–180.

Li, Zhifei and Sanjeev Khudanpur (2008) “A Scalable Decoder for Parsing-based Machine Translation with Equivalent Language Model State Maintenance,” in *Proceedings of the 2nd SSST*, pp. 10–18.

Norimatsu, J., M. Yasuhara, T. Tanaka, and M. Yamamoto (2016) “A fast and compact language model implementation using double-array structures,” *ACM TALLIP* 15(4), 27 pages.

芳賀駿平・谷口正訓・山本幹雄 (2016)「部分転置ダブルアレイを用いた ngram 言語モデルの検討」，第 30 回人工知能学会全国大会, 4B1-5.