4B1-5

部分転置ダブルアレイを用いた $n$gram 言語モデルの検討

A study on $n$gram language models based on partly transposed double-arrays

芳賀 駿平 *1　谷口 正訓 *2　山本 幹雄 *2
Shumpei Haga　Masanori Taniguchi　Mikio Yamamoto

*1筑波大学 情報学群
School of informatics, University of Tsukuba

*2筑波大学大学院 システム情報工学研究科
Graduate school of system and information engineering, University of Tsukuba

Although DALM (Double-Array Language Model) is a fast and compact implementation of $n$gram language models, it fails to fully capitalize on quantization techniques for values of model parameters such as probabilities of $n$grams, because of a structual limitation: it stores values and indexes in the common array. In this paper, we develop some variants of DALM which have separate arrays for values and indexes and can exploit benefits of quantization. We investigate their basic characteristics empirically and propose "partly transposed double-array" which is a key technique to educe the ability of DALMs with separate arrays.

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と、統計的機械翻訳の実行時間のうち言語モデルへのアクセス時間が 3〜4 割程度にもおよんでいることから、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; 乗松他, 2014)。

言語モデルをシステムのモジュールとして見ると、$n$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となる。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、言語モデルの性質を最大限利用することでバランスのよい実装を実現している (乗松他, 2014)。しかし、DALM は確率値等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、確率値の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 の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、量子化による効率的な圧縮を可能とするデータ構造を検討する。まず基本的な構造を検討し、実際のデータに基づく性質を調べた後、配列を分離した DALM のサイズを小さくするために使える「部分転置ダブル配列」を提案する。評価実験により、提案手法は base 配列に関して約 40%の削減が可能であることを示す。

2. ダブル配列と DALM

2.1 TRIE による言語モデルの表現

1 億のエントリを持つ 5gram 言語モデルを TRIE 表現し、子を 1 つ以上持つノードを対象とし、縦軸に子の数、横軸にランク (子の数の多さの順位) を両対数でプロットしたものが図 1 である。見事な冪乗則 (あるいは Zipf's law) となっており、子ノードの数が極端に偏っていることが分かる。すなわち、子ノードが非常に多い少数のノードが存在する一方で、子ノードが少ないノードが非常に多い。これらの性質を用いて、言語モデルの効率的な実装を検討する必要がある。

連絡先: 芳賀駿平 s.haga@mibel.cs.tsukuba.ac.jp

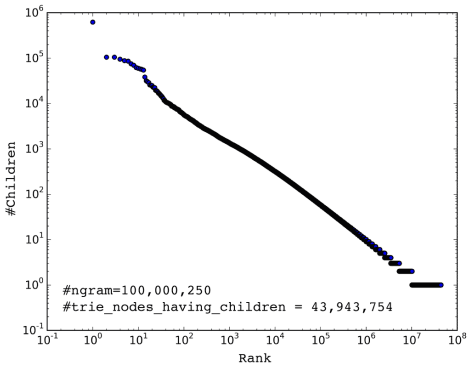

図 1: ランク順に子ノードの数を両対数でプロット

2.2 ダブル配列

図 2 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) を行の値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、多くのノードは少数の遷移記号に対応する子ノードしか持たないため行列は疎な行列となり、言語モデルに応用した場合、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、列方向にぶつからないようにすべての行をずらして 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

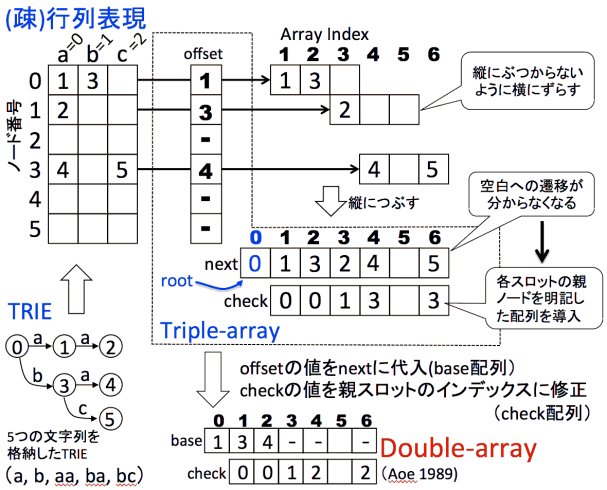

図 2: TRIE の (疎) 行列表現とダブル配列

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。

2.3 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 としての単語 (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。

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、一方、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 3 にこの 2 つの実装方法の長所と短所を図解する。

2.4 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 3 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 「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。

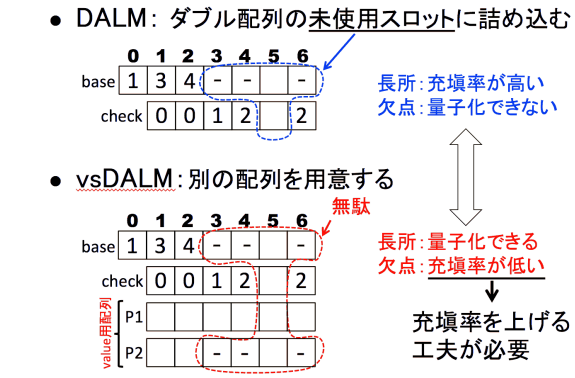

図 3: パラメータ格納場所についての 2 種類の実装手法

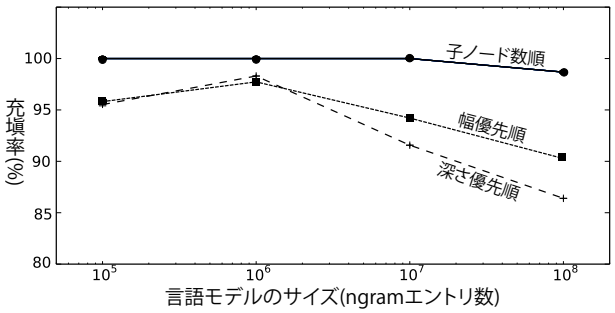

図 4: ノード配置順による充填率の変化

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 2 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。以下の 3 種類について実験を行った結果、図 4 のような結果が得られた。

- 深さ優先：ルートから深さ優先で巡回し、先行順で配置
- 幅優先: 1gram ノード, 2gram ノード,... の順で配置
- 子ノード数順： 子ノードの多い順に配置

図中の横軸はモデルのサイズ (5gram モデルのエントリ数) である。子ノード数順は構築時間がかなり遅くなるが (約十倍)、充填率が高いため、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。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。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、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 5-(a) 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 5-(a) 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、図 1 から分かるように、子ノード数順に配置した場合の上位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 5-(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。

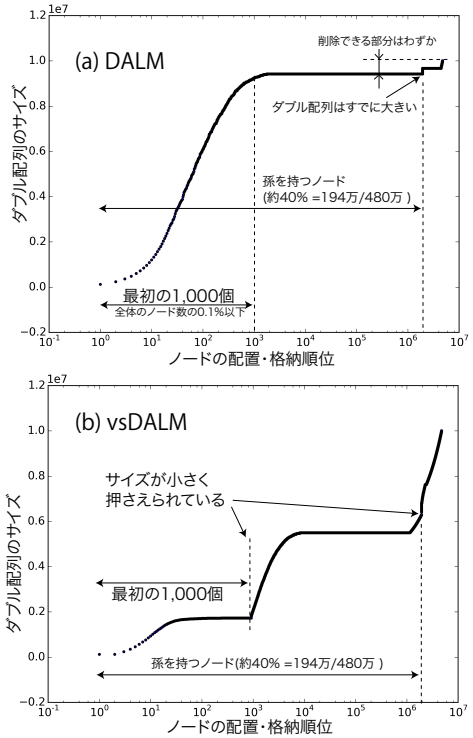

図 5: ダブル配列の成長の様子

4. 部分転置ダブル配列

前節の検討から、1%の密度の長さ約 10 万の行をぶつからないように配置するためには大きなずらし幅が必要であることが分かった。これを改善するために、「長さ 10 万」の方に着目する。例えば、密度 1%は同じでも、行を短くすれば小さなずらし幅で重ねることができる。例えば、1%の密度の長さ 1000 の行 (使用されているスロットは 10 個) は高い確率でずらさなくてもそのまま重ねることができることは容易に想像がつく。このアイデアを用いて、ノード数の多い上位 0.1%のノード (の子ノード) がダブル配列全域に分布しないようにできれば、leaf-last 法が効く可能性がある。

上記のアイデアを次のように実現する。図 2 の (疎) 行列表現を上から下に子ノード数が多い順にならべ替え、上位の 1000 行を転置して base 配列を作るのである。元々 1000×10 万であった上位部分の行列が 10 万 $\times$ 1000 の行列となる。数は増えてしまうが行の列は 1000 となりずらし幅を小さくして重ねていくことができる。この手法を「部分転置ダブル配列」と呼ぶ。

部分転置ダブル配列を作成する例を図 6 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変化ないが、転置部分に関して単語種類毎に offset が決まっている点が異なる。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

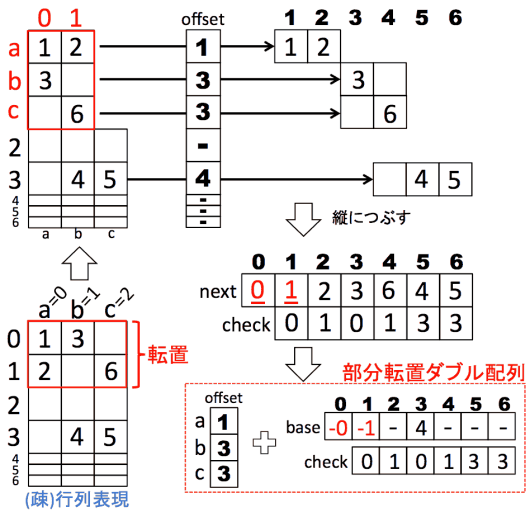

図 6: 部分転置ダブル配列

関係である。

ダブル配列への修正は次のようにする。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 6 の右下の部分転置ダブル配列の例では、ルートから'bb' 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['b']=0+3=3 となり、base[3] に遷移する。check[3]=0 であるためチェックも大丈夫である。次に base[3]=4 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[3]+1('b' の単語 ID)=5 となる。check[5]=3 であるため、遷移に成功し、'bb' 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。

5. 部分転置ダブル配列を用いた言語モデル

5.1 ダブル配列の成長の様子

図 5-(b) に、図 5-(a) と同じデータを用い、子ノード数の多い上位 1000 個のノードを部分転置した場合のダブル配列の成長の様子を示す (配置順は leaf-last 法)。図 5-(a) では最初の 1000 個で最終的なサイズの 90%以上の大きさまで成長していたが、部分転置によって最終的なサイズの 20%程度に押さえられている。また、孫を持つノードすべてを配置した段階で全体の約 6 割の大きさで収まっており、この後に伸びた約 40%は base 配列から削除可能である。

5.2 実際のモデル構築

1 億エントリの 5gram 言語モデルを構築した場合の base 配列の削減割合と総合的なモデルのファイルサイズ (単語から単語 ID に変換するハッシュ表など言語モデルに必要なもろもろの情報を持つ) を、転置するノード数を変化させてプロットしたグラフを図 7 に示す。子ノード数上位 15,000 個のノードを転置した場合で、base 配列を 63%に圧縮でき、トータルなファイルサイズで約 2 割の削減を達成している (量子化すればさらに小さくなる)。

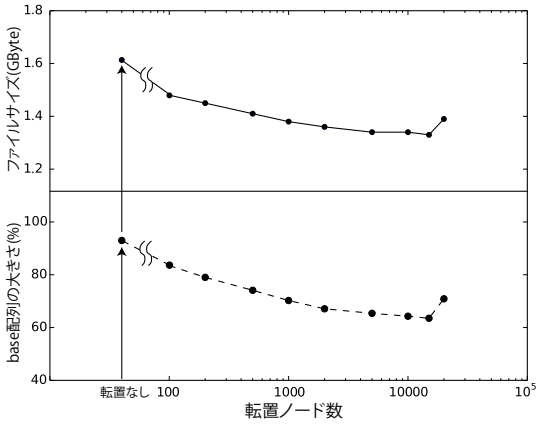

図 7: 転置するノード数と削減率の関係

6. おわりに

DALM のモデルパラメータを分離した value-separeted DALM を検討し、コンパクトにするための次の 3 つの手法を提案・検討した。(1) ダブル配列構築時のノード配置順を子ノード数順に行う方法、(2)leaf-last 法、(3) 部分転置ダブル配列である。これら 3 つを組み合わせることにより、量子化可能となった DALM のサイズを 2 割程度削減できることを示した。

今後の課題は言語モデル state(単語連鎖の左右拡張性の入出力) に対応することと、オリジナルな DALM に対してサイズと速度の比較を行うことである。

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.

Heafield, Kenneth (2011) "KenLM: Faster and smaller language model queries," in *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pp. 187–197.

乗松潤矢・田中透・山本幹雄 (2014) 「逆順 Trie による効率的な Double-Array 言語モデル」,『言語処理学会第 20 回年次大会発表論文集』, 51–54 頁.