Double-Arrayを利用した高速かつコンパクトな *n*gram言語モデルの構築手法

安原　誠　　田中　透　　乗松　潤矢　　山本　幹雄
筑波大学　システム情報工学研究科　コンピュータサイエンス専攻

1　はじめに

言語の統計的な特徴をより正確に捉えた言語モデルほど統計的機械翻訳システムなどの性能を向上させるため、スムージング手法などモデル推定精度を向上させる研究が行われてきた [北, 1999]。一方近年では大規模学習データからモデルを学習することで性能の良い言語モデルを獲得する試みが増えている [Brants et al., 2007]。しかし大規模データから学習した言語モデルはモデルサイズが大きく、メモリ資源の不足や探索速度の低下を招く [Brants et al., 2007]。そのため、より大規模な言語モデルをコンパクトなメモリサイズで高速に利用できるデータ構造の研究がこの5年ほど盛んである。

本稿では、Double-Array[Aoe, 1989] と呼ばれるデータ構造を利用した言語モデルの効率的な構築手法を検討する。モデルパラメータ (確率と backoff 係数) の埋め込みと単語 ID 順序の工夫により、単純な Double-Array の利用に対して 57%のサイズ削減と 8%の高速化を実現した。また、先行研究 [Stolcke, 2002][Pauls and Klein, 2011][Kenneth, 2011] との比較を行い、サイズと速度を総合的に見て性能の良いデータ構造であることを示す。

2　*n*gram言語モデル

言語モデルは文の生起確率を与えるモデルである。特に式1で表される *n*gram が代表的なモデルであり、言語モデルを用いる統計的機械翻訳システムなどで広く利用されている。

$$\begin{aligned} P(\mathbf{e}) &= P(e_1, e_2, ..., e_N) \\ &= \prod_{i=1}^{N} P(e_i|e_1^{i-1}) \\ &\approx \prod_{i=1}^{N} P(e_i|e_{i-n+1}^{i-1}) \end{aligned} \quad (1)$$

*n*gram 条件付き確率 $P_n = P(e_i|e_{i-n+1}^{i-1})$ は基本的には最尤推定により推定されるが、未知事象にも一定の確率を付与するためにスムージングを行う必要がある。本稿では代表的なスムージング手法である backoff スムージングを利用したモデルを対象とした構築手法を検討する。backoff スムージングとは「学習データに出現した *n*gram 確率から一定量を間引き、それらの総和を学習データに出現しない *n*gram に対して低次の *n*gram 統計量を用いて分配する」というものである。backoff スムージングによる *n*gram 確率を式2に示す。ただし、α は学習データに出現した *n*gram の確率から間引いた結果の値であり、β は学習データに出現しない確率への分配量で、backoff 係数と呼ぶ。β は *n*gram 確率の条件部 (これを文脈と呼ぶ) ごとに計算される。P_{n-1} はより低次の *n*gram 条件付き確率である。言語モデルデータ構造では α と β を *n*gram パターンごとに格納する必要がある。大規模学習データの言語モデルでは *n*gram パターンが数十億から数百億も存在するため、メモリに展開して利用する際にサイズや速度の効率化を行う必要がある。

$$P_n = \begin{cases} \alpha & \text{if 対象 } n\text{gram が出現} \\ \beta \times P_{n-1} & \text{otherwise} \end{cases} \quad (2)$$

3　Double-Array言語モデル

3.1　Double-Array

*n*gram パターンを効率的に格納するデータ構造として、単語単位で分岐するトライ木が広く用いられている。本稿では辞書構造によく用いられる Double-Array[Aoe, 1989] という実装方法の利用を検討する。

Double-Array はトライ木を *base* と *check* という2つの配列により表現しており、比較的小さなサイズで高速に探索することができるデータ構造である。Double-Array においてノードは1つの配列要素で表現され、エッジは *base* 配列の値と単語 ID から計算できる。親ノードの位置 *now* から子ノードの位置 *next* へ移動するときの計算式を式3, 式4に示す。まず式3によって、子ノードの位置 *next* を *base* 配列の値 $base[now]$ と単語の ID 値 $ID(word)$ から求める。次に移動先が正しいかどうかを式4によりチェックする。移動先の *check* 配列の値 $check[next]$ が現在位置 *now* と一致すると移動先が正しいことが分かる。もし値が一致しなければ、移動先は誤りであり、その単語をエッジとする子ノードは存在しないことが分かる。

$$next = base[now] + ID(word) \quad (3)$$

$$\text{compare}(check[next], now) \quad (4)$$

3.2　Backward-Suffix-Treeへの適用

backoff 計算を効率的に行えるトライ木として Backward-Suffix-Tree[Bell et al., 1990] がある。「今

日 は 晴れ」という 3gram 確率の探索例である図 1 を基にこの構造を説明する。backoff 計算では「今日 は 晴れ」という 3gram 確率 α_{3gram}、「は 晴れ」という 2gram 確率 α_{2gram}、「晴れ」という 1gram 確率 α_{1gram} の順に backoff するが、Backward-Suffix-Tree では低次から順に木を構成する。Root ノードからターゲット単語「晴れ」をたどり α_{1gram} を獲得し、文脈単語「は」をたどり β_{1gram} を獲得する。さらにノード「は」からターゲット単語「晴れ」をたどり α_{2gram} を獲得し、文脈単語「今日」をたどり β_{2gram} を獲得する。最後にノード「今日 は」からターゲット単語「晴れ」を獲得し α_{3gram} を獲得する。途中の α が見つからなかったときは文脈のみをたどり β を獲得し、得られた最長の $n$gram 確率にかけあわせる。以上により backoff 計算に必要な α と β を 1 つのパスで獲得できるので、効率的に探索できる。

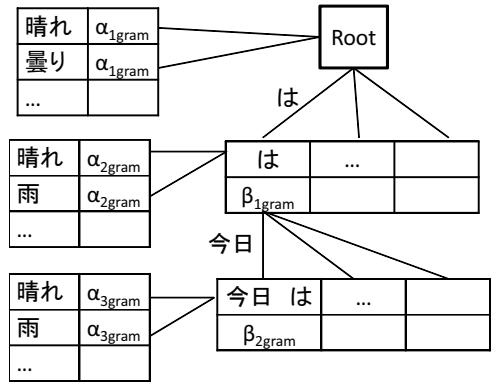

図 1: Backward-Suffix-Tree

Double-Array でこれを表現する場合、1 つのノードに 2 種類の分岐集合が存在するため、2 本の Double-Array が必要となる。そこで、一方の分岐集合に仮想単語 $<\#>$ を追加し、$<\#>$ で分岐した先のノードにもう一方の分岐集合を持たせることにより、1 本の Double-Array のみで表現する。Backward-Suffix-Tree を 1 本の Double-Array で表現した図 2 を例に、「天気 は 晴れ」という 3gram 確率 α_{3gram} の探索手順を説明する。探索は位置 0 から開始し、式 3 により「は」という文脈単語の移動先である位置 W に移動する。位置 W には「は」という文脈の backoff 係数 β_{1gram} が保存されている。さらに「天気」という文脈単語で移動し位置 X に移動する。位置 X には「天気 は」という文脈の backoff 係数 β_{2gram} が保存されている。最後に「晴れ」というターゲット単語で移動する場合、まず仮想単語 $<\#>$ で位置 Y に移動し、位置 Y から単語「晴れ」で位置 Z に移動する。3gram 確率 α_{3gram} は位置 Z から取得できる。

3.3 モデルパラメータの埋め込み

図 2 において、確率 α と backoff 係数 β を保存するための value 配列を Double-Array に並行して用意するとサイズ増大を招く。そこで、この value 値の情報を Double-Array に埋め込み、サイズ効率化を図る。

確率値 α は、探索して到達した位置の *base* を利用

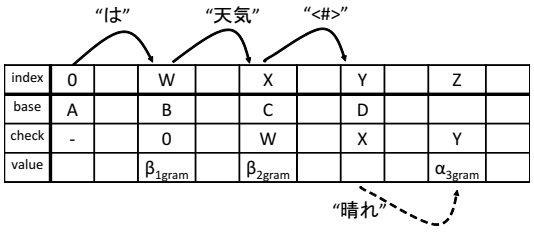

図 2: Double-Array 言語モデルデータ構造

する。終端の *base* 値は必ず空であるため、この位置に確率値を格納する。backoff 係数 β は $<\#>$ の *check* を利用する。すべての文脈ノードが $<\#>$ への分岐を持つとき、$<\#>$ への移動が存在することは自明となり、*check* を保存する必要がなくなるため backoff 係数を格納できる。また、他ノードからの誤った移動を正しく排除するために、*check* 配列の 1bit をチェックビットとして利用することで *check* 値か backoff 係数かを区別する。実際には lossless な量子化 [Guthrie and Hepple, 2010] を行った backoff 係数の配列を用意し、値に対応する配列インデックスの符号を反転したマイナス値を格納する。

図 2 のパラメータを Double-Array に埋め込んだ例を図 3 に示す。前節と同様に「天気 は 晴れ」という 3gram の確率を探索し位置 Z にたどり着いた場合、値 α_{3gram} は $base[Z]$ に保存されている。また、「天気 は」という backoff 係数は、「は」,「天気」,「$<\#>$」とたどった位置 Y の *check* 値 $check[Y] = -a$ を獲得し、符号反転した値 a をインデックスとして、量子化配列から値 β_{2gram} を取得する。

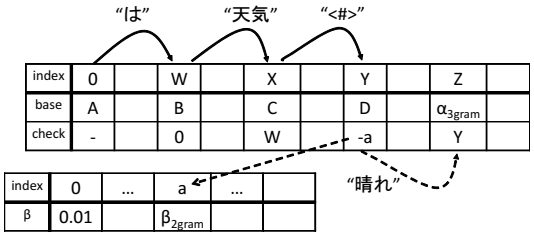

図 3: モデルパラメータの埋め込み

3.4 単語 ID 順序の工夫

Double-Array の充填率はトライ木のノードが持つ分岐集合の ID 値の幅 (ID の最大値-最少値) に依存する。この ID 値の幅をウィンドウ幅と呼ぶ。トライ木全体の分岐集合の平均ウィンドウ幅が小さいほど充填率の高い Double-Array を構築できる。$n$gram 言語モデルトライ木はウィンドウ幅の最大値が語彙数となり極めて大きいため、Double-Array の充填率が低くなりやすい。しかし、語彙の大部分の単語はトライ木中に数回しか出現せず、全体の平均ウィンドウ幅にあまり影響を及ぼさない。平均ウィンドウ幅に大きく影響を及ぼすのはトライ木中に頻出する少数の単語群であり、頻出する単語どうしが近い ID 値をとるほど平均ウィンドウ幅は小さくなり、充填率の高い Double-Array になりやすい。

そこで、本稿では単語 ID 順序を決めるために 1gram 確率の大小を利用する方法を提案する。1gram 確率はトライ木中の出現頻度にほぼ比例するため、トライ木で頻出する単語同士に近い ID 値を割り当てられる。その結果、単語 ID をランダムに割り振った場合に比べてトライ木全体の平均ウィンドウ幅を小さくでき、充填率の高い小サイズな Double-Array を構築できる。

4 先行研究

大規模な *n*gram 言語モデルのためのデータ構造に関する研究は近年盛んになっており、さまざまな手法が提案されている。現在もっとも普及している *n*gram 言語モデルツールは SRILM[Stolcke, 2002] である。SRILM は Backward-Suffix-Tree を用いており、ノードには文脈単語の分岐集合とターゲット単語の分岐集合、backoff 係数が保存されている。分岐集合の中から特定のエッジを探し出す部分にはハッシュ探索が用いられる。

BerkeleyLM[Pauls and Klein, 2011] は context-encoding というアルゴリズムを用いた連想配列トライ木データ構造を用いる。context-encoding とは、*n*gram 単語列を文脈とターゲット単語に分割し、符号化したペアで表現することによりサイズ効率化を行う手法である。また、Implicitly-Encoding ではターゲット単語の部分の情報を効率化することでさらにサイズを削減している。

KenLM[Kenneth, 2011] はハッシュテーブルを用いた Probing とトライ木を用いた Sorted-Trie を提案している。Probing では *n*gram を直接ハッシュテーブルにマッピングする。ハッシュテーブルは一般的に key を保存しなければならないが、Probing ではそれを *n*gram の 64bit ハッシュ値としている。また、Sorted-Trie は BerkeleyLM のような連想配列トライ木であるが、探索に内挿探索を用いて高速化を行っている。

5 評価実験

5.1 実験条件

実験データには、NTCIR の特許情報検索タスク学習データとして公開されている日本語公開特許公報全文 1993-2002 年の背景と実施例から抽出した 100 億単語を用いる [Fujii et al., 2007]。学習データは文単位でランダムサンプリングした 50 億単語と、そこからさらにランダムサンプリングした 1 億単語の 2 種類を用意した。また、テストセットは、100 億単語データから 50 億単語データを取り除いた残りの 50 億単語からランダムサンプリングした 1 億単語とした。各データの情報を表 1 に示す。

言語モデルは SRILM-Toolkit により、オーダー 5 の線形補間 Modified-Kneser-Ney スムージングを学習した。また、低出現の *n*gram を足切りするカットオフの設定は (1,2,3,4,5)gram に対し (0,0,1,1,1) とした。メモリサイズは実行プロセスにおいて言語モデルを読み込む前後の差分とし、探索速度はテストセットを全て単語 ID に変換した後の状態で、テストセットパープレキシティを計算するタスクの処理時間から算出した。測定マシンは Xeon X5675 3.07GHz メインメモリ 144GB を用いた。

表 1: 学習データとテストセット

	文数	単語数	*n*gram パターン総数
学習データ 1 億	約 234 万	約 1 億	約 3,192 万
学習データ 50 億	約 1.2 億	約 50 億	約 9.4 億
テストセット	約 234 万	約 1 億	

5.2 効率化手法の性能評価

本節では効率化手法の性能評価実験について述べる。言語モデルは 1 億単語学習データから学習した。value 値の埋め込みと単語 ID 順序の工夫のそれぞれを用いた場合と用いない場合の合計 4 通りについて Double-Array データ構造を構築し、メモリサイズと探索速度を測定した、その結果を表 2 および図 4 に示す。効率化を行わない場合に対して両効率化を行った場合はメモリサイズが 57%減少し、探索速度は 8%高速化した。

表 2: 効率化手法の性能評価

	メモリサイズ (MB)	探索速度 (μ sec/key)
効率化なし	1,160.3	1.296
value 埋め込み	790.6	1.293
単語 ID 順序	733.8	1.201
両効率化	505.8	1.192

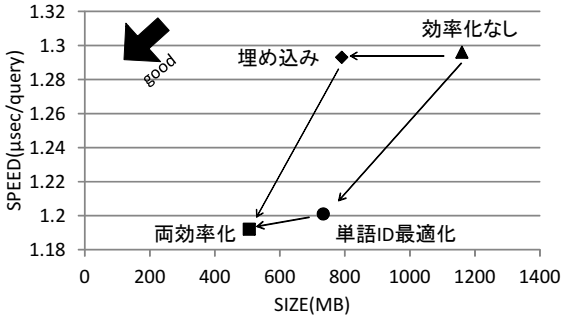

図 4: 効率化手法の性能評価

5.3 先行研究との性能比較

本節では効率化手法を組み込んだ Double-Array 言語モデルデータ構造と先行研究との比較実験を行う。各手法によるメモリサイズと探索速度を測定した結果を表 3 および図 5 に示す。SRILM および BerkleyLM に対して提案手法は高速かつコンパクトなデータ構造であることが分かる。KenLM と比較した場合、Probing に対してはほぼ同程度の探索速度でサイズを 34%削減した。また Sorted-Trie に対しては 50%のサイズ増加

表 3: 先行研究との性能比較

	1 億単語言語モデル		50 億単語言語モデル	
	メモリサイズ (MB)	探索速度 (μ sec/key)	メモリサイズ (MB)	探索速度 (μ sec/key)
提案手法	505.8	1.192	14,368	1.441
SRILM	1,418.3	1.368	36,770	1.631
BerkeleyLM	566.5	1.336	14,981	1.643
KenLM:Probing	733.5	1.273	21,449	1.391
KenLM:Sorted-Trie	340.7	1.636	9,606	1.938

に対して 30%の高速化を果たした。

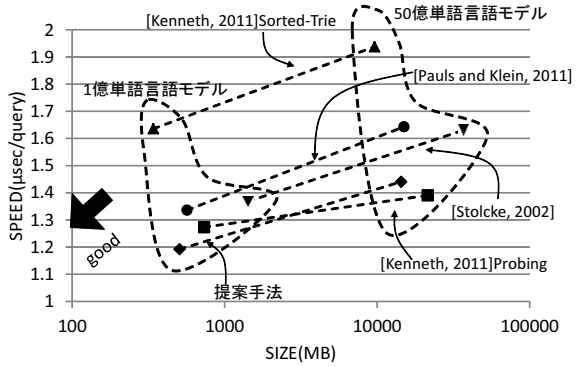

図 5: 先行研究との性能比較

言語モデルの学習データ量を増加させると KenLM の Probing の探索速度が相対的に高速になった。これは、言語モデルの大規模化に伴いテストセットの *n*gram がより高次で言語モデルにヒットしやすくなり、ハッシュテーブルにとって不利な backoff 計算の回数が減少したためであると考えられる。実際にテストセットパープレキシティ計算における 5gram のヒット率を調べたところ、1 億単語言語モデルでは約 30%であったが、50 億単語言語モデルでは約 50%であった。

6 おわりに

本論文では Double-Array を利用した高速かつコンパクトな *n*gram 言語モデルデータ構造を提案した。value 値の埋め込みと単語 ID 順序の工夫により単純な実装に比べて 57%のサイズを削減し、いくつかの先行研究よりも性能の良いデータ構造であることを示した。

より大規模な言語モデルではテストセットの *n*gram が高次でヒットしやすくなるため、高次から探索することのできるデータ構造は相対的に高速になる。このことより、今後はハッシュテーブルのように高次から探索する手法と比較しながら検討したい。

参考文献

[Aoe, 1989] Junichi Aoe. 1989. An Efficient Digital Algorithm by using a Double-Array Structure. IEEE Transaction on Software Engineering, 15(9), pp.1066-1077.

[Bell et al., 1990] T.C. Bell, J.G. Cleary, and I.H. Witten. 1990. Text compression. Prentice Hall.

[北, 1999] 北研二. 1999. 確率的言語モデル. 東京大学出版会.

[Stolcke, 2002] Andread Stolcke. 2002. SRILM-An Extensible Language Modeling Toolkit. In Proc. Intl. Conf. Spoken Language Processing, pp.901-904.

[Brants et al., 2007] T. Brants, A.C. Popat, P. Xu, F.J. Och, and J. Dean. 2007. Large Language Models in Machine Translation. In Proc. of the 2007 Joint Conference on EMNLP and CNLL, pp.858-867.

[Fujii et al., 2007] A. Fujii, M. Iwayama, and N. Kando. 2007. Overview of the Patent Retrieval Task at the NTCIR-6 Workshop. In Proc. of NTCIR-6 Workshop Meeting, pp.359-365.

[Guthrie and Hepple, 2010] David Guthrie and Mark Hepple. 2010. Storing the Web in Memory: Space Efficient Language Models with Constant Time Retrieval. In Proc. of the 2010 Conference on EMNLP, pp.262-277.

[Pauls and Klein, 2011] Adam Pauls and Dan Klein. 2011. Faster and Smaller N-Gram Language Models. In Proc. of ACL.

[Kenneth, 2011] Kenneth Headfield. 2011. KenLM : Faster and Smaller Language Model Queries. In Proc. of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation.