TR-IT-0066

# エントロピー最少化基準を用いた単語抽出
# Word Extraction using Minimum Entropy Criterion

加藤 剛 Katoh Goh
村上 仁一 Murakami Jin'ichi

1994.8.31

従来、音声認識システムを構築する際、特にあらかじめタスクが限定されているような場合には言語モデルの基本単位として単語を用いることが多い。英語などの欧米諸語の場合、文章は単語毎に区切って書かれるため、単語の概念は明解である。しかし日本語においては文章は切れ目なく連続して書かれるため単語の概念は曖昧である。そこでこの論文では文字のエントロピーによって「単語」の抽出を試みる。本稿は筆者が電気通信大学の実習生としてATR滞在中に行なったものである。

# 目次

# 第 1 章

# はじめに

従来、音声認識システムを構築する際、特にあらかじめタスクが限定されているような場合には言語モデルとの基本単位として単語を用いることが多い。しかしこの「単語」は言語学者によって語彙知識で決められたものである。多くの欧米諸語では文章は単語毎に区切って書かれているので単語の概念は明確である。しかし日本語においては文章は連続して書かれ、ただ句読点のみによって区切られるだけである。したがって何らかの基準によって明確に単語を抜き出す必要がある。本稿では言語モデルのエントロピー最少化の手法を用いて「単語」を抽出する。似たような研究としては [1, 2, 3, 4, 5] がある。北らは独自に仕事量基準というものを設定し定型表現をコーパスから抽出した [5]。仕事量基準の特徴は抽出された定型表現を実際に自然言語の処理に応用する段階で「どの程度処理が改善されるのか」ということが考慮されている点にある。

本研究は単語を抽出する。本論文の場合は既存の単語を全く考慮せずコーパス中のすべての文字を同等に扱い文字列を生成して「単語」を抽出する。それによって従来気づかなかった新たな「単語」の概念が得られる可能性がある。

本稿の構成は、まず文字連鎖可変長統計について述べ、つぎに単語抽出実験、最後に考察を述べる。

# 第 2 章

# 可変長統計

## 1　文字連鎖可変長統計

### 1.1　言語モデルのための単語選択

連続音声認識において、音声パターン A が観測された時、それが単語 W である確率は、

$$P(\hat{W}|A) = \max P(W|A)$$

である。ベイズの定理より、

$$P(W|A) = \frac{P(W)P(A|W)}{P(A)}$$

で P(A) は一定とすると、

$$\max P(W)P(A|W)$$

なる、$W$ を見つければよい。ここで、$P(W)$ は言語モデルによって得られる確率、$P(A|W)$ は HMM 等の音響モデルから求める。しかし、$P(W)$ は、

$$P(W) = \prod_{i-1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1})$$

となるので、これらの各々について確率を求めることはパラメータの爆発を招き、事実上不可能である。そこでなんらかの近似的方法により求める必要がある。そこで今回は漢字かな交じり文字 $w$ に対してノードを割り当て、言語モデル全体のエントロピーの増加が最小になるようにして漢字かな交じり文字連鎖可変長統計を求める。方法としては [1] の自動学習アルゴリズムを用いて、「単語」を抽出することができる。これによって求められた「単語 $w_i$ 」を基に言語モデルを構築することにより、$P(W)$ を求めることが出来る。本論文は言語モデル構築のための前段階としての単語抽出実験について述べている。

### 1.2　文字のエントロピー

あるコーパス中で文字 $c$ の出現確率が $p$ だったとすると $c$ を見た時に得られる情報量は、$\log_2 \frac{1}{p}$ である [6]。いま、事象 $c_1, c_2, \ldots, c_n$ からなる事象系 $X$ があり、そのうちの事象が選択されるような選択操作を考えてみる。ここで、事象 $i$ の選択される確率を $P(c_i)$ とする。すると $i$ 番めの事象が選択された時、得られる自己情報量は、$I(c_i) = -\log_2 P(c_i)$ である。この

ようなことが起こる確率は $P(c_i)$ であるから、この選択によって得られる情報量の期待値すなわち、平均情報量は、

$$H(X) = -\sum_{i=1}^{n} P(c_i) \log_2 P(c_i)$$

となる。この $H(X)$ のことを 情報源のエントロピーという。エントロピーには次のような種類がある。

1. 結合エントロピー
   2 つの事象系 X,Y において結合事象 $(c_i, c_j)$ の生起する結合確率は、$P(c_i, c_j)$ とすると、結合事象の自己情報量 $I(c_i, c_j)$ は

$$I(c_i, c_j) = \log_2 \frac{1}{P(c_i, c_j)}$$

   となる。$I(c_i, c_j)$ の平均をとれば結合事象のエントロピーが得られ、これを結合エントロピーとする。すなわち、

$$\begin{aligned} H(X,Y) &= \sum_{i=1}^{n}\sum_{j=1}^{m} P(c_i, c_j) I(c_i, c_j) \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{m} P(c_i, c_j) \log_2 P(c_i, c_j) \end{aligned}$$

   となる。

2. 条件付きエントロピー
   $c_j$ を 知っている条件の下で $c_i$ が与える条件付自己情報量は

$$I(c_i|c_j) = -\log_2 P(c_i|c_j)$$

   である。ここで $c_j$ を知っているという条件のもとで、我々が得る情報量の期待値を求めると、

$$\begin{aligned} H(X|c_j) &= \sum_{i=1}^{n} P(c_i|c_j) I(c_i|c_j) \\ &= -\sum_{i=1}^{n} P(c_i|c_j) \log_2 P(c_i|c_j) \end{aligned}$$

   となる。$H(X|c_j)$ のことを事象 $c_j$ が生起した時の X の条件付エントロピーという。さらに $H(X|c_j)$ の事象系 $Y$ に関する平均値を求めると、

$$\begin{aligned} H(X|Y) &= \sum_{j=1}^{m} P(c_j) H(X|c_j) \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{m} P(c_j) P(c_i|c_j) \log_2 P(c_i|c_j) \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{m} P(c_i, c_j) log_2 P(c_i|c_j) \end{aligned}$$

   となる。$H(X|Y)$ のことを事象 $Y$ が生起したときの $X$ に関する条件付エントロピーまたは単に条件付きエントロピーという。

このようにエントロピーにはいろいろな種類があるが、今回の実験ではエントロピーとして結合エントロピー

$$H(X,Y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} P(c_i,c_j)\log_2 P(c_i,c_j)$$

を用いた。

## 1.3 漢字かな交じり文字可変長統計

音声認識の音韻認識の誤り補正の方法の一つとして、音韻連鎖統計モデルが使われている。このモデルとしては、一般に n-gram などのモデルが使われている。たしかに n-gram において、音韻連鎖長 n の値を大きくすれば推定の精度は上がると考えられる。しかし実際にこのようなことをしようとするとトレーニングデータベース中での出現頻度が下がりモデルの信頼性は低下する。これを防止するためには、deleted-interpolation などのスムージングなどが考えられるが、認識対象によって bigram,trigram などを使い分けるという方法が提案されている [1, 3, 4]。今回は、次に述べる [1] のアルゴリズムを用いて、どのようなものがトレーニングデータベースから「単語」として抜き出されてくるか調べる。

## 1.4 自動学習アルゴリズム

漢字かな交じり文字の可変長統計の自動学習アルゴリズムを次に示す。今回は簡単のため、[1] のアルゴリズムを少々変更して以下のようにした。

1. 各文字を一つのノードに割り当てる。
   語彙が $c_a$ と $c_b$ の 2 文字のとき、ノードの集合を、$\phi_i$ とすると、$\phi_1 = \{(c_a),(c_b)\}$ 。

2. あるノードの文字を前の文字と統合して考えた時、言語モデル全体のエントロピーの増加が最小になるノード $c$ を探す。
   $c_a$ を 2 つ組にしたときのエントロピー $H_a$ と $c_b$ を 2 つ組にしたときのエントロピー $H_b$ を計算し、両者のエントロピーを比較し、小さい方を選択する。

$$H_a = p(c_a,c_a)\times\log p(c_a,,c_a) + p(c_b,c_a)\times\log p(c_a,c_b) + p(c_a)\times\log p(c_b)$$

$$H_b = p(c_a)\times\log p(c_a) + p(c_a,c_b)\times\log p(c_b,c_a) + p(c_b,c_b)\times\log p(c_b,c_b)$$

   仮に $H_a < H_b$ とする。

3. $c_a$ に関して 2 つ組を選択する。
   $\phi_2 = \{(c_a,c_b),(c_a),(c_b))\}$

4. 一定のノード数になるまで 2 と 3 を繰り返す。

これらを繰り返すと、ある文字に関しては 2 つ組, ある文字に関しては 3 つ組となるような漢字かな交じり文字可変長統計が得られる。

# 第 3 章

# 単語抽出実験

## 1 予備実験

単語抽出実験に入る前にデータベースの文字の n-gram でのエントロピーを調べる。

### 1.1 実験方法

実験 1 では、エントロピーとして、下記の条件付きエントロピーを用いた。

- unigram

$$H = -\sum_{i}^{N} p(c_i) \log_2 p(c_i)$$

- bigram

$$H = -\sum_{i,j}^{N} p(c_i, c_j) \log_2 p(c_j|c_i)$$

- trigram

$$H = -\sum_{i,j,k}^{N} p(c_i, c_j, c_k) \log_2 p(c_k|c_i, c_j)$$

実験 2 では、エントロピーとして、下記の結合エントロピーを用いた。

- unigram

$$H = -\sum_{i}^{N} p(c_i) \log_2 p(c_i)$$

- bigram(2 つ組)

$$H = -\sum_{i,j}^{N} p(c_i, c_j) \log_2 p(c_i, c_j)$$

- trigram(3 つ組)

$$H = -\sum_{i,j,k}^{N} p(c_i, c_j, c_k) \log_2 p(c_i, c_j, c_k)$$

また両実験とも、トレーニングデータベースとして、国際会議申し込み文 8475 文を使い、文頭と文末はそれぞれ特別な文字として置き換え、計算した。

表 3.1: n-gram のエントロピー (1)

| n-gram | (bits) |
|---|---|
| unigram | 6.91 |
| bigram | 3.60 |
| trigram | 2.13 |

## 1.2 実験結果 1

結果を表 3.1に示す。

## 1.3 実験結果 2

結果を表 3.2に示す。

表 3.2: n-gram のエントロピー (2)

| n-gram | (bits) |
|---|---|
| unigram | 6.89 |
| bigram | 10.66 |
| trigram | 12.84 |

## 1.4 考察

実験 1 と実験 2 では unigram の値が変わっている。1 は gawk で 2 は C で計算したための計算誤差の範囲と考えられる。また 1 が n-gram の n の値が大きくなるにつれてエントロピーが減少しているのに対し 2 では増大している。

# 2 単語抽出実験

## 2.1 実験条件

実験は表 3.3の条件のもとで行なった。

表 3.3: 実験条件

| | |
|---|---|
| 学習データ | 国際会議の問い合わせデータベース 8475 文中 68 文。 |
| 単語統合基準 | エントロピー最小化 |

[2] では、いろいろなマルコフモデルでのエントロピーが示されている。本来 bigram,trigram のエントロピーは 条件つき確率でなければならないが、今回は予備実験 (2) の方法つまりエン

トロピーとして、

$$H = -\sum_{i}^{N} p_i \log_2 p_i$$

を用い計算した。

## 2.2 実験結果

実験より得られた結果を付録Aに示す。また、発生した語彙と消滅した語彙の結果を付録Bに示す。
今回の単語抽出実験結果から、次のことがいえる。

1. 2つ組で抽出された単語は、いわゆるふつうの「単語」が多い。
   これはまず頻出する2文字の名詞をまとめて一語としたほうが全体のエントロピーが下がるということを示している。このことは[1]でのunigramでの場合に対応しており、名詞は様々な文字に接続しているという日本語の特徴を獲得している。

2. 漢字はほとんど単独で残るものはない。
   付録Aの1文字で残った文字をみると、漢字で1文字で残っているのは23個しかない。初期状態の漢字の数は144個であったから実に初期状態のうち約16%しか生き残っていない。これは2文字の漢字の名詞が効率的に抽出されたということを示している。

3. 文字が4個5個と連続してくっついていくときにはプログラムのループが回る度に次々にくっついていく傾向がみられた。
   これもデータが小さいので、出現する単語が限られているため一つの単語内の文字の出現確率が同じになっているからであり、今回のアルゴリズムではいきなり5つ組などがまとまることはなく、常に2つづつまとまるようにしているからと考えられる。今回プログラムに読み込ませることのできた縮小データベースの場合、視察で見る限り有用であると思えたのは3つ組までで、それ以上は非常にタスク依存なまとまり方をしている(付録A参照)。しかしそのなかでも「通訳電話国際会議」など意味があるまとまり方をしているものは比較的早い段階で出現している。

4. 数詞は特に良く検出されている。
   今回のタスクでは、参加費用、会議開催日、会場までの距離など決まり切った表現が多いせいか数詞の検出が目だった。数字で最後まで1文字で残っているものはなかった。

5. 語彙数を増やすにつれ、エントロピーは減少する。
   語彙数を増やしたときのエントロピーの変化をみると、ふやすにつれてエントロピーは減少していっている(図A.2)。予備実験(2)ではn-gramの値を上げると、エントロピーは上がっていたのでそれとは逆の結果となった。単語抽出実験中のの語彙数の変化をみると、新しい語彙が発生した時にその要素となった語彙が消滅することが多々ある。たとえば、$\{a, b\}$から$\{a, b, ab\}$と語彙が移行する時に$a, b$のデータ内での出現の組合せが$ab$しかないとすると$a, b$の出現数はゼロになる。すると$a, b$のエントロピーもゼロとなり、このことは言語モデル全体のエントロピー、つまりに文字毎のエントロピーの総和に対しては減少させる働きをもつ。図A.1をみると、今回の実験では初期語彙数より減っていることが分かる。しかしこれが一般的にいえることかどうかは大規模なデータで試していないので不明である。

6. ノード数が飽和するとタスク依存な単語ができ始める。
図 A.1とそのときに発生した新しい語彙を見ると、図 A.1のノード数の減少が飽和し始める 80 ループ目ぐらいからタスク依存的なくっつき方をする語彙が作られている。
例として、「8 月」、「987 年」、「直接京都駅」などがある。第 243 ループ目には「が一応登録証を 50 人文提出」というのが 1 単語とされており、このタスク以外にはあり得ない単語となっている。これより将来大規模データベースで単語抽出を行なう際でも、ノード数の飽和が起こったらループを終了する措置をとるということが有効な単語抽出の方法として推定される。

# 第 4 章

# 考察

本研究に対する今後の検討項目として、

- プログラムの高速化
  今回のプログラムの場合、4k byte程の学習データで一つノードを増やすためには、6分かかる。よって目標の6000ノードまで増やすのに約20日かかる。さらに実際の学習データは500k byteなので、これでは実質動かない。本実験の後、アルゴリズムを少々改良し高速化を図ったがそれでも500kのデータで一つ語彙を増やすためには、3日ほどかかる。そこでアルゴリズムの根本的再検討を含め、さらに高速化を検討する必要がある。ちなみに付録には改良前のものを収録している。

- 大規模データベースの使用
  今回は実習期間という限られた時間の中で行なったので、オリジナルのデータベースの前から68文のみを切り出した縮小データベースで行なった。これもデーターベースを大きくすることによって抽出する「単語」が今回のものと変化する恐れがある。

- 統合基準の補正および検討
  本研究では、エントロピーが最小になるように各ノードをまとめていった。またどの語彙をまとめるかによって全体の語彙ごとの出現回数が変化するので一概にエントロピーで最小のものを選ぶというのはどうか、エントロピーの正規化も必要ではないかという疑問もある。相互情報量最大化の基準も試すはずであったが、今回は時間の関係上割愛した。この統合基準を変えたらどうなるか検討する必要がある。

- 認識実験への組み込み
  この報告では音声認識実験との実験は時間の関係上、行なわれていない。しかし将来この実験より得られた「単語」からなるモデルをつくり認識実験を行ない、従来の単語での認識と比較してみる必要があると思われる。

ということが挙げられる。特に今回は実際に動かせる大きさでのデータでは早いループ数で語彙数が飽和してしまったので単語がどのように成長していくか十分に追い切れなかったのが残念である。

# 第 5 章

# むすび

本研究では、エントロピー最少化基準を用いた自動単語抽出の可能性について検討した。国際会議問い合わせ文データベースのうち 86 文について単語抽出を行ない、プログラムの動作を確認した。単語抽出実験は、語彙数 249 個から始め、 211 語彙を獲得するのに成功した。その 211 語彙中、新たに得られたのは 136 語彙であった。結果から得られた傾向は、

- 2 つ組で抽出されるものは 2 文字の漢字からなる名詞が多い
- 漢字は 1 文字で残るものは少ない。
- 数詞が良く検出されている。
- 5 つ組くらいまでの名詞などは連続して成長する。
- 語彙を増やすとエントロピーは減少する。
- ノード数が飽和するとタスク依存な単語ができ始める。

であった。また今後の課題としては、最重要課題にプログラムの高速化、つぎに大規模データベースでの実験、単語統合基準の検討、認識実験への組み込みがある。

## 謝辞

本実習に当たり、懇切丁寧な御指導と、激励をいただきました、村上 仁一研究員、磯谷 亮輔研究員、TSG の皆さん、並びに ATR 音声翻訳通信研究所の皆様に感謝致します。また、本実習の機会を与えていただいた、電気通信大学 樽松明 教授、ATR 音声翻訳通信研究所 山崎 泰弘 社長に感謝致します。

# 参考文献

[1] 村上仁一, 嵯峨山茂樹. 単語連鎖可変長統計の自動学習に基づく連続音声認識. No. 2-1-9, pp. 95–96, October 1992.

[2] 村上仁一, 坪井俊明. 学習データ量に対するマルコフ連鎖確率値の収束性と単語の HMM と Bigram を用いた文節音声認識システムについて. No. SIG-SLUD-9301-1, May 1993.

[3] 田本真詞, 伊藤克亘, 田中穂積. 木構造を用いた音韻連鎖統計モデル. pp. 7B–5,2–9, 1992.

[4] 田本真詞, 伊藤克亘, 田中穂積. 木構造を用いた音韻連鎖統計モデル. pp. 1E–3,2–229, 1993.

[5] 北研二, 小倉健太郎, 森元逞, 矢野米雄. 仕事量基準を用いたコーパスからの定型表現の自動抽出. pp. ,Vol.34,No–9, Sep 1993.

[6] 南敏. 情報理論. 産業図書, Dec 1988. ISBN4-7828-9003-6.

# 付録 A

# 付録

## 1　単語抽出実験によって得られた結果

単語抽出実験によって得られた結果

表 A.1: ノード数 256 個で計算を打ち切った時の結果

| | |
|---|---|
| 1 つ組で残った文字 | ー, あ, い, う, え, か, が, き, く, ぐ, け, こ, し, す, そ, ぞ, た, ち, っ, つ, て, で, と, ど な, に, ね, の, は, ば, ふ, ま, み, む, も, や, ょ, よ, ら, り, る, れ, ろ, わ, を, ん, エ, カ, ト ド, プ, 何, 確, 関, 近, 件, 行, 合, 込, 載, 思, 従, 書, 申, 速, 他, 中, 日, 分, 便, 方, 面, 訳, 来 |
| 2 つ組になった文字 | 会議, 登録, 観光, 京都, 参加, 費用, 東京, 主旨, 現在, 結構, ツア, 大阪, 午後, じゃ, ござ 上げ, 名前, 必要, 締切, 項目, 計画, 空港, 以外, 斡旋, 場合, 8 月, グル, 最後, 6 月, 料金 資料, 場所, 当日, 着き, スピ, 人数, 自分, 5 日, 者が, 4 時, 3 つ, 1 つ, 短く, 高く, ただ お話, おり, お伺, お使, せん, 考え, 参り, つき |
| 3 つ組になった文字 | 京都駅, 飛行機, 新幹線, 事務局, タクシ, ホテル, 外国人, 為替レ, 予約等, ご主旨, ただき 今手元, 予定さ, 推薦さ, ゲスト, 後観光, ナンバ, 参加さ, 割引を, 同じ日, 3 時間, 最終的 時間的, 地理的, 日本人, ず第 1 , 第 1 回, 3 1 日, 1 0 時, 1 時間, 朝早く, を選ば, ドを持 じゃ他, お聞き, お支払, ドをね, を書く, ば支払, ばよろ, 支払わ, ろ夕方, 方を見, 私たち が時間, お乗り, お送り |
| 4 つ組になった文字 | 国際会議, 点新幹線, 一応手元, 4 0 分ぐ, ね私たち, 5 0 人ほ, ただく他, 登録費用 8 月 5 日 |
| 5 つ組になった文字 | 直接京都駅, 国際会議場, ジェントみ, 本来外国人, 1 0 0 ドル, 4 5 分間ぐ, 2 時間半ぐ 京都へ行く, だ割引を私, 1 9 8 7 年, 私たち大勢, ドをお持ち, 登録用紙が, 載せただけ |
| 6 つ組になった文字 | パンフレット, アメリカドル, ば大体 3 5 分, ばキャッシュ, 大体 3 時間ぐ, クレジットカ |
| 7 つ組になった文字 | 観光プログラム |
| 8 つ組になった文字 | 観光プログラムみ, 大体 1 5 0 0 円ぐ |
| 9 つ組になった文字 | 通訳電話国際会議 |
| 10 つ組になった文字 | リムジン・バスが出 |
| 11 つ組になった文字 | 大体 1 6 , 0 0 0 円ほど |
| 12 つ組になった文字 | 1 0 0 ドルをアメリカドル |
| 13 つ組になった文字 | が一応登録証を 5 0 人分提出 |

## 2　発生した語彙と消滅した語彙の関係 (新語彙数 256 個で打ち切った時)

| | 元からの語彙数 | 新語彙数の変化 | 計 |
|---|---|---|---|
| 発生した数 | 249 | 256 | 505 |
| 消滅した数 | 174 | 120 | 294 |
| 残りの数 | 75 | 136 | 211 |

表 A.2: 発生した語彙と消滅した語彙。

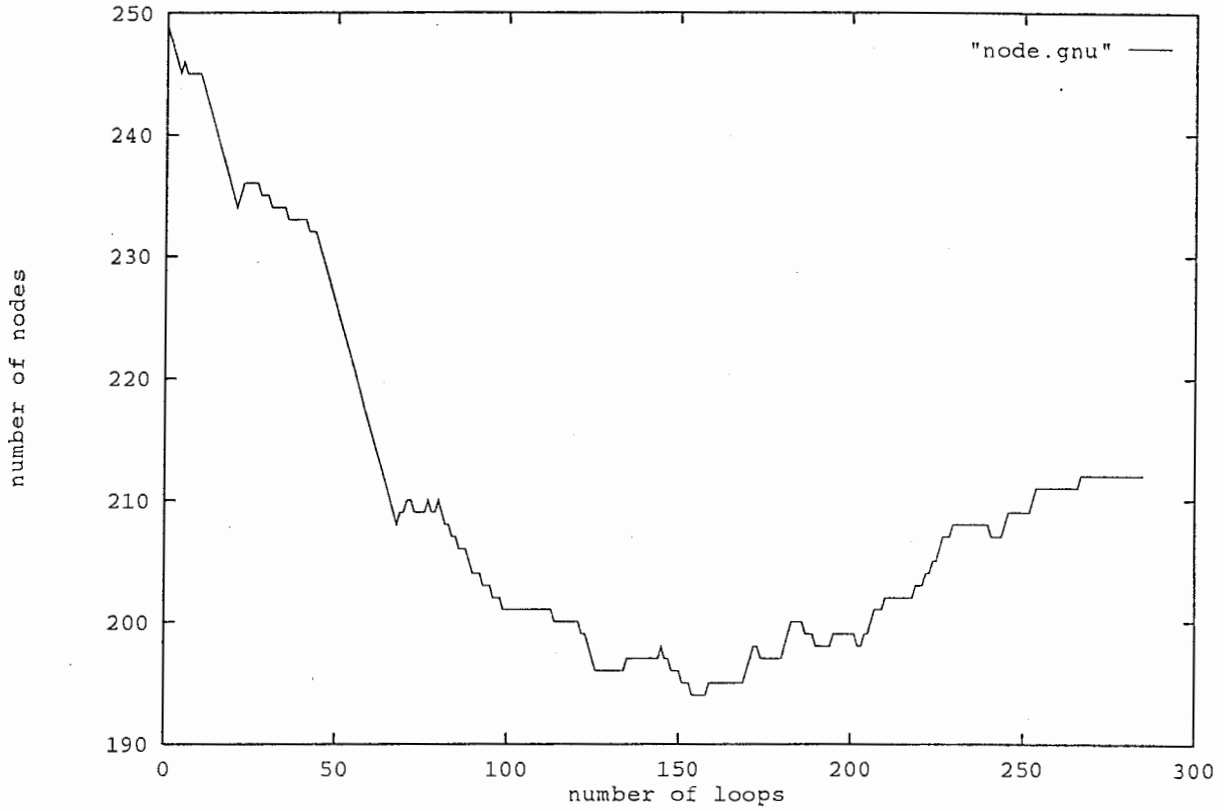


図 A.1: ノード数の変化

## 3　エントロピーの変化 (新語彙数 256 個で打ち切った時)

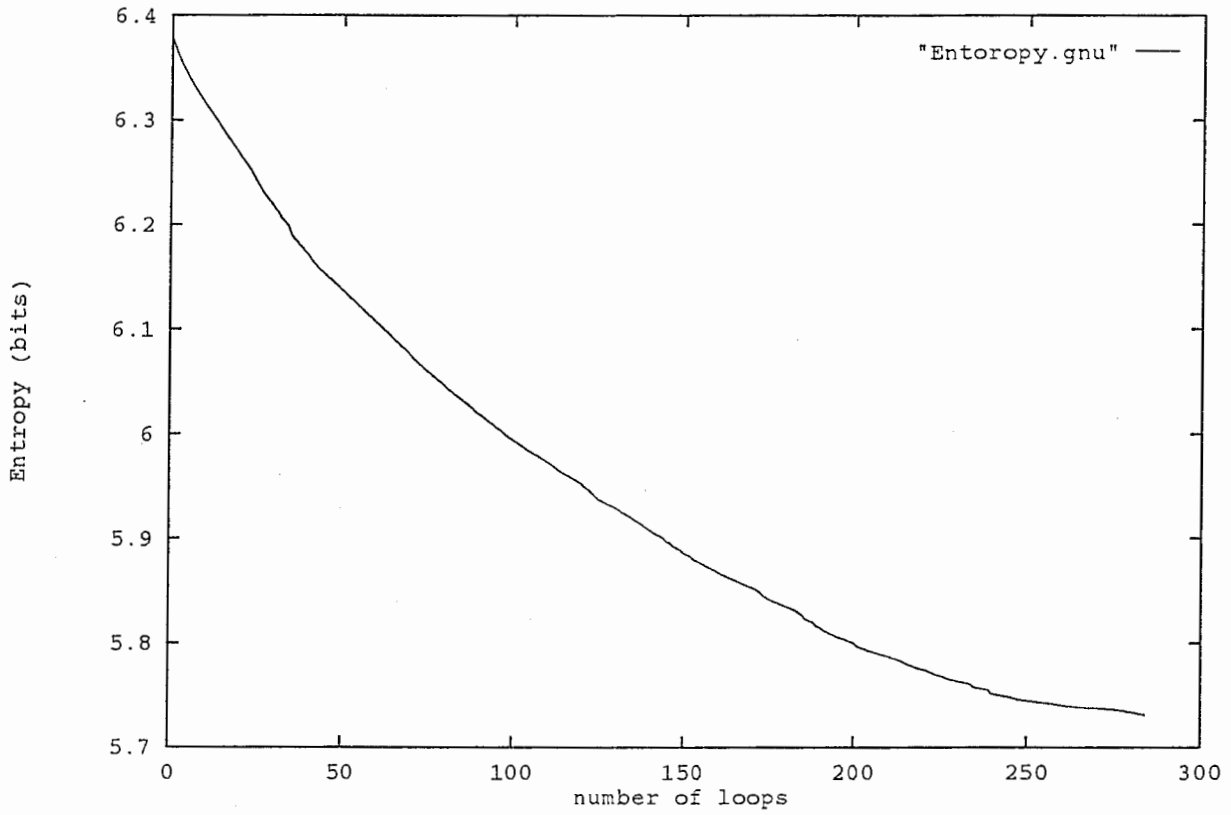


図 A.2: エントロピーの変化

# 4 program list

プログラムの使い方

data : 学習データファイル

FILE_tyu-kann : 途中経過ファイル
内容は data ファイルの一文字ずつの統計をとったもの。実際には 1 文字づつ縦書きにするプログラム disp を使用し、disp — sort — uniq -c — gawk 'print 2,1' ¿ FILE_tyu-kann というふうにつくる。

Entropy.history : ループ毎のエントロピーとノード数表示

DECODE_TABLE : 新語彙の置換表ファイル

NODE : 増やしたいノード数 ( もとのノード数も含む。 )

実行は、

$$xkato@atrq05 < 1 > node$$

とする。 途中結果は次々にファイル DECODE_TABLE に出てくる。またファイル data を覗くと次々に新しい語彙で書き換えられていくのが分かる。
途中から再開する時は、関数 neomoji() の neosum の値を前に終了しループ数に設定して再コンパイルする。つぎに、

$$mv \ \ Latest\_result \ \ FILE\_tyuukan$$

として再び実行すると Entropy.history に 100.000000 というのが書き込まれるが, それはエントロピーの最小値なので、実験終了後にエディター等で削除する。

```
/*                                                    */
/* mv Latest__resultHP FILE_tyuukan して再開。    */
/*                                        8/25        */

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

#define NODE 1500

struct word{
              char w[8];
      int freq;
      int used;
      double prob;
    };
```

```
static double min=100.0;
static struct word keep_w[NODE],tmp[NODE],PROC[NODE],USED[6000];
double log2(double);
double PrintEntropy(struct word *pa,int N,char *name);
int Min(double);
void PrintArray(struct word *pa,int N,char *name);
void CheckSum(struct word *pa,int N);
int COPY(struct word a[],struct word b[]);
int PartSum(struct word *pa,int N);
int jsearch(char *str1,char *str2);
int count( char *str,FILE *fp);
char *neomoji();
int bicount(char *str,FILE *fp);
int isused(struct word *pa,char *str,int n);
int biperm(char *str,char *str2,FILE *fp,FILE *fp2 );
void FprintArray(struct word *pa,int N,char *str,FILE *fp5);

main(int argc, char *argv[])
{

  FILE *fp1,*fp2,*fp3,*fp4,*fp5;
  int num=0,word_sum0=0,l,max,i,j,a,n=0;
  struct word c0[NODE];
  double H_tmp=0.0,local_max=0;
  char *str1,str[7];

  if( (fp1=fopen("data","r"))==NULL){
    printf("Can't open data!\n");
    exit(1);
  }
  if( (fp2=fopen("FILE_tyu-kann","r"))==NULL){
    printf("Can't open FILE_tyu-kann!\n"); /* 中間ファイル   */
    exit(1);
  }
  while(fscanf(fp2,"%s %d\n",c0[num].w,&c0[num].freq)!=EOF){
    word_sum0 +=c0[num].freq; c0[num].used=0; ++num;
  }
  fclose(fp2);

  for(l=0;l<num;l++){
    c0[l].prob=(double)(c0[l].freq)/(double)word_sum0;
  }
  for(l=num;l<=NODE;l++){
    c0[l].prob=-1.0;
```

```
  }

  for(l=num;l<=NODE;l++){
    keep_w[l].prob=-1.0;
  }
  for(l=num;l<=NODE;l++){
    tmp[l].prob=-1.0;
  }
  for(l=num;l<=NODE;l++){
    PROC[l].prob=-1.0;
  }
  for(l=0;l<=NODE;l++){
    USED[l].prob=-1.0;
  }

  max=COPY(c0,keep_w);
  max=COPY(keep_w,tmp);
  PrintArray(tmp,num,"tmp");

  count(tmp[0].w,fp1); rewind(fp1);    /*    初期化    */
  a=bicount("0200",fp1); rewind(fp1); /*    初期化    */

  while(max < 321){

    if( (fp2=fopen("Entoropy.history","a"))==NULL){
      printf("Can't open Entoropy.history!\n");
      exit(1);
    }
    fprintf(fp2,"%lf\n",min);
    fclose(fp2);

    min=100.0; local_max=max;

    str1=neomoji();

    for(j=0;j<local_max;j++){
      strcpy(str,tmp[j].w);
      for(i=0;i<local_max;i++){
max=COPY(keep_w,tmp); printf("max=%d\n",max);
strcat(str,tmp[i].w);
printf("new node={%s}  {%s} + {%s}\n",str,tmp[j].w,tmp[i].w);
strcpy(tmp[max].w,str);
tmp[max].prob=0.01994;        /* とりあえず適当な値  */
```

```
tmp[max].freq=bicount(tmp[max].w,fp1); rewind(fp1);

/* printf("tmp[max].freq=%d \n",tmp[max].freq);
   printf("tmp[max].w=%s \n",tmp[max].w);  */

if(tmp[max].freq==0){
            /* 存在しないからＪＵＭＰ  */
  goto JUMP;
}
if( isused(USED,tmp[max].w,n) ){
   /* 一度使ったからＪＵＭＰ  */
  goto JUMP;
}

for(l=0;l<max;l++){     /*  tmp[] に 頻度を格納。  */
  rewind(fp1);
  tmp[l].freq=count(keep_w[l].w,fp1);
  strcpy(tmp[l].w,keep_w[l].w);
}

/* printf("tmp[j].w=%s %d tmp[i].w=%s %d\n",tmp[j].w,tmp[j].freq,tmp[i].w,tmp[i].freq
tmp[j].freq=tmp[j].freq-tmp[max].freq*jsearch(tmp[j].w,tmp[max].w);
tmp[i].freq=tmp[i].freq-tmp[max].freq*jsearch(tmp[i].w,tmp[max].w);
H_tmp=PrintEntropy(tmp,PartSum(tmp,max),"H_tmp");  /*  エントロピー計算    */
/* PrintArray(tmp,max,"Tmp");  */

if( Min(H_tmp) ){
  max=COPY(tmp,PROC); /* PrintArray(PROC,max,"PROC");  */
  CheckSum(tmp,max);  /* printf("ここの max=%d\n",max);  */
  strcpy(USED[n].w,tmp[max-1].w);USED[n].freq=tmp[max-1].freq;USED[n].prob=tmp[max-1]
  USED[n].used=1; USED[n-1].used=0; n++;
  printf("交換 !\n");
/*   PrintArray(USED,n,"USED");   */
  printf("\n\n");
}
else{
  printf("不適 \n");
}

    JUMP:

str[2]='\0';
/* printf("-----------------------------------------------------\n"); */
     }
```

```
    }
      COPY(PROC,keep_w);
      max=COPY(keep_w,tmp);

    if( (fp3=fopen("TMP.FILE","w"))==NULL){
      printf("Can't open TMP.FILE!\n");
      exit(1);
    }

    if( (fp4=fopen("DECODE_TABLE","a"))==NULL){
      printf("Can't open DECODE FILE!\n");
      exit(1);
    }
    fprintf(fp4,"%s --> %s\n",keep_w[max-1].w,str1); /* DECODE_TABLE  */
    fclose(fp4);

    rewind(fp1);
    biperm(keep_w[max-1].w,str1,fp1,fp3);
    fclose(fp3);

    fclose(fp1);
    system("mv TMP.FILE data");

    strcpy(keep_w[max-1].w,str1);

    if( (fp1=fopen("data","r"))==NULL){
      printf("Can't open s.data!\n");
      exit(1);
    }

    printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n");

    if( (fp5=fopen("Latest_result","w"))==NULL){
      printf("Can't open Latest_result!\n");
      exit(1);
    }
    FprintArray(keep_w,max,"keep_w",fp5);
    fclose(fp5);


}       /*      while()      */

PrintArray(keep_w,max,"keep_w");
printf("min=%lf\n",min);
```

```
  fclose(fp1);

} /*    main()     */


/*  ---------------  end of main() ---------------------------- */

double log2(double x)
{
  double a;

  if(x==0.0){
    return(0.0);
  }
  else{
    a=log(x)/log(2.0);
    return(a);
  }
}

double PrintEntropy(struct word *pa,int num,char *name)
{
  double H=0.0;

  while(pa->prob >=0){
    if(pa->freq<0) return(1000.0);
    pa->prob=(double)pa->freq/num;
    H += -1.0*(pa->prob)*log2(pa->prob);
    pa++;
  }
  printf("%s=%lf\n",name,H);
  return(H);
}

int count(char *str,FILE *fp)
{

  int count_num=0,cnt,len;
  char buff[2048];

  count_num=0;

  while(fgets( buff,2048,fp )!= NULL ){
```

```
    len=(int)strlen(buff);

    for( cnt = 1;cnt < len; cnt +=2 ){

      if(str[0]==buff[cnt-1] && str[1]==buff[cnt]){
++count_num;
      }

    }
  }
  return(count_num);
}

char *neomoji()
{
  static char neos[3];
  static int neosum=0,neoi=0,neoj=0; /*   再開する時は neosum を変える。 */

  static char num[]="0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

  neoj=(int)neosum/61;
  neoi=neosum-61*neoj;
  ++neosum;

  neos[0]= num[neoj];
  neos[1]= num[neoi];
  neos[2]=  '\0';

  return(neos);

}

int COPY(struct word a[],struct word b[])
{
  int i=0;

  while(a[i].prob >=0){
    b[i]=a[i];
    i++;
  }
  return(i);  /*   配列の最大値＋1    */
}

void PrintArray(struct word *pa,int N,char *name)
```

```
{
  int i=0;

  while((pa->prob) >= 0){
    printf("%s[%d]= %s %lf %d %d\n",name,i,pa->w,pa->prob,pa->freq,pa->used);
    pa++;i++;

  }
}

int PartSum(struct word *pa,int N)
{
  int i,sum_freq=0;

  while( (pa->prob) >= 0){
    sum_freq += pa->freq;
    pa++;
  }
  return(sum_freq);
}

void CheckSum(struct word *pa,int N)
{
  int i;
  double sum=0.0;

  while(pa->prob >= 0){
    sum += pa->prob; /* printf("pa->prob=%lf\n",pa->prob);  */
    pa++;
  }
  printf("sum=%lf\n",sum);
}

int Min(double m)
{
  int ANS=0;

  if( m <= min ){
    min = m;
    ANS=1;
  }
  return (ANS);
}
```

```
int jsearch(char *str1,char *str2)
{

  int COUNT=0; /* printf("check %s %s\n",str1,str2); */

  while( *str2!='\0' ){
    if(strncmp(str2,str1,2)==0){
       ++COUNT;
    }
    ++str2;++str2;
  }
  return(COUNT);
}

int bicount(char *str,FILE *fp )
{

  int count2_num=0,cnt,len;
  char buff[2048],buff2[5];

  count2_num=0;

  while(fgets( buff,2048,fp )!= NULL ){

    len=(int)strlen(buff);

    for( cnt=1; cnt < len; cnt+=2 ){

      if(str[0]==buff[cnt-1] && str[1]==buff[cnt] && str[2]==buff[cnt+1]
 && str[3]==buff[cnt+2] ){
++count2_num;
      }
    }
  }
  return(count2_num);
}

int isused(struct word *pa,char *str,int n)
{
  int is_i=0;

  while(pa->freq!=NULL){
    if( strcmp(pa->w,str)==0 && pa->used==1){
      return(1);
```

```
    }
    is_i++; pa++;
  }
  return(0);
}

int biperm(char *str,char *str2,FILE *fp,FILE *fp3 )
{

  int cnt,len;
  char buff[2048],buff2[5];

  while(fgets( buff,2048,fp )!= NULL ){

    len=(int)strlen(buff);

    for( cnt=1; cnt < len; cnt+=2 ){
      if(str[0]==buff[cnt-1] && str[1]==buff[cnt] && str[2]==buff[cnt+1]
 && str[3]==buff[cnt+2] ){
        fprintf(fp3,"%s",str2); cnt+=2;

      }
      else{
buff2[0]=buff[cnt-1];
buff2[1]=buff[cnt];
buff2[2]='\0';
      fprintf(fp3,"%s",buff2);
      }
    }
  fprintf(fp3,"\n");
  }
}

void FprintArray(struct word *pa,int N,char *name,FILE *fp5)
{
  int i=0;

  while((pa->prob) >= 0){
    fprintf(fp5,"%s %d\n",pa->w,pa->freq);
    pa++;i++;

  }
}
```