Chapter 6 Language models

- Introduction
  - なぜ単語列の確率を計算したいのですか
  - 例：スペイン語から英語へ翻訳
  - 生成モデル
  - ノイズの多いチャンネルモデルのメリット
- 6.1 N-gram language models
  - 相対頻度の推定値
  - バイアス・バリアンス
  - nグラム言語モデル：イントロダクション
  - nグラム言語モデル：定義
  - 潜在的な問題を具体的に考えてみましょう。
- 6.2 Smoothing and discounting
  - 6.2.1 Smoothing
    - 例
  - 6.2.2 Discounting and backoff
    - 絶対ディスカウント
    - Katzバックオフ
  - 6.2.3 *Interpolation
  - 6.2.4 *Kneser-Ney smoothing
- 6.3 Recurrent neural network language models

    - ニューラルネットワーク
    - RNN(リカレントニューラルネットワーク)
    - RNNの備考
    - RNNのボトルネック
    - 6.3.1 Backpropagation through time
    - 6.3.2 Hyperparameters
    - 6.3.3 Gated recurrent neural networks
        - LSTM
- 6.4 Evaluating language models
    - 6.4.1 Held-out likelihood
    - 6.4.2 Perplexity
- 6.5 Out-of-vocabulary words
    - 語彙$\mathcal{V}$は有限集合でないかもしれない
    - 未知の単語を$\langle \mathrm{UNK} \rangle$とする
    - 未知の単語は区別した方がよい
    - 未知の単語の区別方法
- Additional resources

Introduction

- 確率的分類では、問題は、テキストを条件とするラベルの確率を計算することであった。
- 逆問題として、テキスト自体の確率を計算することを考えてみましょう。
    - 特に、$w_m \in \mathcal{V}$での単語トークン列に確率$p(w_1, w_2, \ldots, w_M)$を割り当てるモデルを

考察する。 ここで集合$\mathcal{V}$は離散的な語彙であり

$$\mathcal{V} = \{\text{aardvark}, \text{abacus}, \ldots, \text{zither}\}. \quad [6.1]$$

- 確率的言語モデルの目標は，単語トークン列の確率を正確に測定することである。(§6.4.1 序文)

なぜ単語列の確率を計算したいのですか

- 多くのアプリケーションでは、出力として単語列を生成することが目的です。
  - **機械翻訳**(§18)では、原文言語のテキストを訳文言語のテキストに変換します。
  - **音声認識**では，音声信号をテキストに変換する。
  - **要約**(§16.3.4; §19.2)では、長いテキストを短いテキストに変換します。
  - **対話システム**(§19.3)では，ユーザの入力を(外部の知識ベースを使って)テキスト応答に変換する。
- これらのタスクを実行するための多くのシステムにおいて、出力テキストの確率を計算するサブコンポーネントが存在する。
  - このサブコンポーネントの目的は、より**fluent(滑らかな)**テキストを生成することです。

例：スペイン語から英語へ翻訳

- たとえば、文をスペイン語から英語に翻訳するとします。

(6.1) El café negro me gusta mucho.

- 以下は、単語から単語への直訳です (a **gloss**（用語解説？）):

  (6.2) The coffee black me pleases much.

- 優れた英語の言語モデルは、この翻訳文の確率が他の文法的な文に比べて低いことを示します。

$$p(\text{The coffee black me pleases much}) < p(\text{I love dark coffee}). \quad [6.2]$$

- この事実をどう翻訳に利用すればいいのでしょうか。
- 機械翻訳の初期のリーダーの一人であるWarren Weaverは、機械翻訳を秘密のコードを破ることの問題だと考えた (Weaver, 1955):
  - ロシア語の記事を見ると、私はこう言う:「これは本当に英語で書かれているが、いくつかの奇妙な記号でコード化されている。これからデコードします。」

生成モデル

この観察は、生成モデルの動機となります。

- 英語の文$w^{(e)}$ は、**言語モデル**$p_e\left(w^{(e)}\right)$から生成されます。
- 次に、スペイン語の文$w^{(s)}$ が**翻訳モデル**$p_{s|e}\left(w^{(s)}|w^{(e)}\right)$から生成される。

$$w^{(e)} \sim p_e(w^{(e)})$$
$$w^{(s)} \sim p_{s|e}(w^{(s)}|w^{(e)})$$

- これら2つの分布が与えられると、翻訳はベイズの規則によって行われる。

$$p_{e|s}(w^{(e)}|w^{(s)}) \propto p_{e,s}(w^{(e)}, w^{(s)}) \quad [6.3]$$
$$= p_{s|e}(w^{(s)}|w^{(e)}) \times p_e(w^{(e)}). \quad [6.4]$$

  - これは、英語のテキストがノイズの多いチャンネル$p_{s|e}$を通過してスペイン語になることを想定しているため、**noisy channel model(ノイズの多いチャンネルモデル)**と呼ばれることがあります。

ノイズの多いチャンネルモデルのメリット

$p_{e|s}$を直接モデリングするのとは対照的に、翻訳をこのようにモデリングすることの利点は何ですか?

- 重要な点は、二つの分布$p_{s|e}$(翻訳モデル)とp_e(言語モデル)が別々のデータから推定できることである。
- 翻訳モデルには正しい翻訳の例が必要ですが、言語モデルに必要なのは英語のテキストだけです。
- このような単一言語データははるかに広く利用可能である。
- さらに、言語モデルp_eは、いったん推定されると、他の言語からの翻訳を含めて、英語の

テキストを生成することを伴う任意のアプリケーションで再利用することができる。

6.1 N-gram language models

相対頻度の推定値

トークン列の確率を計算する簡単な方法は、**相対的な頻度の推定値**を使用することである。

- “Computers are useless, they can only give you answers.”(コンピュータは使い物になりません．答えをくれるだけです。)というピカソの言葉を考えてみよう。
- この文の確率を推定する一つの方法は

$$p(\text{Computers are useless, they can only give you answers}) = \frac{\text{count}(\text{Computers are useless, they can only give you answers})}{\text{count}(\text{all sentences ever spoken})}. \quad [6.5]$$

バイアス・バリアンス

この推定量は**unbiased(不偏・偏りがない)**：

- つまり，無限にデータがあるという理論的限界では推定値は正しい。

しかし実用上、単語列は任意の長さになる可能性があるため、無限数のイベントについて正確なカウントを必要とする.

- たとえば、列の長さに$M = 20$トークンという攻撃的な（？）上限があってさえ、可能な単語列の数はV^{20}となる．ここで$V = |\mathcal{V}|$である。
- 英語用の小さな語彙は$V = 10^5$となるので、10^{100}通りの単語列が可能です。

明らかに、この推定量は非常にデータを必要とし、high variance(**高い分散**)に悩まされている：

- つまり，文法的な文でさえも、もしそれらが訓練データ中に生じていなければ、確率0を持つだろう。
  - チョムスキーは、これは確率的言語モデルの概念そのものに反する証拠だと論じたことで有名だ。
  - 確率的言語モデルでは、文法的な文“colorless green ideas sleep furiously”（無色で緑色の考えは猛烈に眠る・青く透明な睡魔を見た）と非文法的な置換“furiously sleep ideas green colorless”を区別することはできない。
- なお§2.2.4では、high varianceのことを「訓練データに依存して大きく異なる分類ルールをとりえる」と説明しているからここでもその意味だろうか。

したがって，有限の訓練データから信頼できる推定を行う機会を得るためにバイアスを導入する必要がある。

- バイアス＋分散は一定なので，分散を減らすためにバイアスをあきらめるということらしい．
- つまり，文法的な文が訓練データ中に存在しなくてもその文の確率が0より大きくなるように，理論的限界で推定値が正しくなることをあきらめる，ということのようだ．

この章で述べる言語モデルは、さまざまな形でバイアスを導入する。

---

nグラム言語モデル：イントロダクション

まず、nグラム言語モデルから始めます。

- このモデルは、単語列の確率を、部分列の確率の積として計算します。
- 単語列の確率$p(w) = p(w_1, w_2, \ldots, w_M)$は、チェイン規則(§A .2参照)を使用して書き直せる。

$$p(w) = p(w_1, w_2, \ldots, w_M) \quad [6.6]$$
$$= p(w_1) \times p(w_2|w_1) \times p(w_3|w_2, w_1) \times \ldots \times p(w_M|w_{M-1}, \ldots, w_1) \quad [6.7]$$

$$p(\text{I like black coffee}) = p(\text{I}) \times p(\text{like}|\text{I}) \times p(\text{black}|\text{like, I}) \times p(\text{coffee}|\text{black, like, I})$$

  - 積の各要素は、そのすべての先行語が与えられる単語の確率である。
  - なお逆の順序（またはその他の順序で）でチェーンルールを適用することもできます。

$$p(w) = p(w_M) \times p(w_{M-1}|w_M) \times \ldots \times p(w_1|w_2, \ldots, w_M), \quad [6.8]$$

これは単語予測タスクと考えることができます。 “computers are”という文脈が与えられたら、次のトークンに対する確率を計算します。

- この文脈で単語uselessの確率の相対頻度推定値は

$$p(\text{useless}|\text{computers are}) = \frac{\text{count}(\text{computers are useless})}{\sum_{x \in \mathcal{V}} \text{count}(\text{computers are x})} = \frac{\text{count}(\text{computers are useless})}{\text{count}(\text{computers are})}.$$

しかし、これはまた、私たちが実際には何の進歩もしていないことを意味する：

- 条件付き確率$p(w_M|w_{M-1}, w_{M-2}, \ldots, w_1)$を計算するためには、$V^{M-1}$($V^M$?)通りの単語列をモデル化する必要があるだろう。
- このような分布は現実的なテキストサンプルからは推定できない。

---

nグラム言語モデル：定義

この問題を解決するために、nグラムモデルは非常に単純化した近似を行い、過去の$n-1$ワードのみを条件とする。

$$p(w_m|w_{m-1} \ldots w_1) \approx p(w_m|w_{m-1}, \ldots, w_{m-n+1}) \; [6.9]$$

これは、文wの確率が次のように近似できることを意味する。

$$p(w_1, \ldots, w_M) \approx \prod_{m=1}^{M} p(w_m|w_{m-1}, \ldots, w_{m-n+1}) \; [6.10]$$

- 文全体の確率を計算するには、始めと終わりを特殊記号□■で埋めると便利です。

- そうすると、“I like black coffee”の確率に対するバイグラム$(n = 2)$近似は以下のようになる。

$$
\begin{aligned}
&p(\text{I like black coffee}) \\
&= p(\text{I}|\square) \times p(\text{like}|\text{I}) \times p(\text{black}|\text{like}) \times p(\text{coffee}|\text{black}) \times p(\blacksquare|\text{coffee}). \quad [6.11]
\end{aligned}
$$

このモデルでは、V^n個のイベントの確率だけを推定して格納する必要がある。

- V^n個のイベントはnグラムのオーダで指数関数的であり、V^Mのオーダーではない．V^Mは文の長さで指数関数的である。
  - つまり考慮するイベントの数が激減する．

nグラム確率は、相対頻度推定によって計算することができる。以下はトライグラム。

$$
p(w_m|w_{m-1}, w_{m-2}) = \frac{\text{count}(w_{m-2}, w_{m-1}, w_m)}{\sum_{w' \in \mathcal{V}} \text{count}(w_{m-2}, w_{m-1}, w')} \quad [6.12]
$$

ハイパーパラメータnは、各条件付き確率で使用される文脈のサイズを制御する。

- これを誤って指定すると、言語モデルのパフォーマンスが低下します。

潜在的な問題を具体的に考えてみましょう。

nが小さすぎる場合。

- 次の文を考えてみましょう。

    - (6.3) **Gorillas** always like to groom **their** friends.　（ゴリラはいつも友達の身繕いをするのが好きです。）
    - (6.4) The **computer** that's on the 3rd floor of our office building **crashed**.
- 各例において、太字で記載された単語は、互いに依存しており、theirの尤度は、Gorillasが複数であることを知ることに依存し、crashedの尤度は、対象がcomputerであることを知ることに依存する。
- もしnグラムがこの文脈を捉えるのに十分大きくないならば、結果として得られる言語モデルは、これらの文に対しては低すぎる確率を提供し、数の一致のような基本的な言語学的テストに失敗する文に対しては高すぎる確率を提供するだろう。
    - 文法的に正しい文の確率が減るので，その分だけ文法的に正しくない文の確率が増える，ということを言っているようである．

nが大きすぎる場合。

- この場合，訓練データ不足のため，データセットからのnグラムパラメータの推定は困難である。
- ゴリラの例を扱うには、6グラムをモデル化する必要があり、これはV^6個のイベントを説明することを意味する。
- $V = 10^4$という非常に小さな語彙でも、10^{24}個の異なるイベントの確率を推定することを意味する。

これら二つの問題は、もう一つの**バイアスと分散のトレードオフ**を示している(§2.2.4参照)。

- §2.2.4のバイアスと分散のトレードオフは以下のようなものであった。
    - バイアスのない分類器は訓練データにオーバーフィットする。未知のデータに対する

性能が悪い。
    - 平滑化が大きすぎるとアンダーフィットを起こす。どのような訓練データでも同じ分類器を与えるが、バイアスが大きい。

- 小さいnグラムサイズは高いバイアスをもたらし、大きいnグラムサイズは高い分散をもたらす。
- 両方の問題を同時に発生させることもできます。
    - 言語は、nが小さすぎるために捕捉できない、長い依存関係でいっぱいである.
    - 同時に、言語データセットには稀な現象が多く、nが大きすぎるためにその確率を正確に推定することができない。

1つの解決策は、nを大きく維持しながら、基礎となるパラメータの低分散推定を行うことである。

- 低分散とは，文法的な文にできるだけ確率0を与えないようにすることであろう.

そのために、別の種類のバイアス、**スムージング**を導入します。

6.2 Smoothing and discounting

限られたデータは，言語モデルを推定する際の持続的な問題である。

- §6.1において，部分解としてnグラムを提示した。
- ビットスパースデータ（データ不足のことか？）は、低いnグラムでも問題になる可能性が

ある；
- 同時に、主語-動詞の一致のような多くの言語学的現象は、高いnグラムがなければ言語モデルに組み込むことができない。

したがって、nグラム言語モデルに追加のinductive(誘導?)バイアスを加える必要がある。

- このセクションでは、最も直感的で一般的な方法をいくつか説明しますが、他にも多くの方法があります (Chen and Goodman, 1999).

6.2.1 Smoothing

言語モデリングにおける主要な関心事は、$p(w) = 0$という状況を避けることであり、これは、単一の未観測のnグラムの結果として生じる可能性があります。

- 同様の問題がナイーブベイズでも起こり、その解決策は**平滑化**であった。平滑化とは想像上の擬似カウントを加えることであった。
- ここでバイグラムの場合に示すように、nグラム言語モデルにも同じ考え方を適用できます。

$$p_{\text{smooth}}(w_m|w_{m-1}) = \frac{\text{count}(w_{m-1}, w_m) + \alpha}{\sum_{w' \in \mathcal{V}} \text{count}(w_{m-1}, w') + V\alpha}. \quad [6.13]$$

  - 確率が正しく正規化されるように、分子 (α) に加えるものはすべて分母$(V\alpha)$にも入れなければならない。 これより$\sum_{w \in \mathcal{V}} p_{\text{smooth}}(w|w_{m-1}) = 1.$

この基本的なフレームワークは**Lidstone平滑化**と呼ばれますが、特別な場合には別の名前が付けられます。

- **ラプラス平滑化**は$\alpha = 1$の場合に相当する。
- **Jeffreys-Perksの法則**は$\alpha = 0.5$の場合に対応しており、実際にはうまく機能しており、いくつかの理論的根拠(Manning and Schutze,1999)から恩恵を受けている。

この考え方は、**実効カウント**c_i^*の概念に反映されています。

- $p_{\text{smooth}} = c_i^*/M$となるように，$c_i^*$を定める.

$$c_i^* = p_{\text{smooth}} \cdot M = \frac{c_i + \alpha}{M + V\alpha} \cdot M, \ [6.14]$$

- ここで、c_iはイベントiのカウント、$M = \sum_{i=1}^{V} c_i$はデータセット内のトークンの総数$(w_1, w_2, \ldots, w_M)$である。
- $\sum_{i=1}^{V} c_i^* = (\sum_{i=1}^{V} c_i + V\alpha)M/(M + V\alpha) = M$である。

例

$$\frac{8 + 0.1}{20 + 7 \times 0.1} = \frac{8.1}{20.7} = 0.391, \ 0.391 \times 20 = 7.826.$$

			Lidstone smoothing, $\alpha = 0.1$		Discounting, $d = 0.1$	
	counts	unsmoothed probability	effective counts	smoothed probability	effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391	7.9	0.395
offense	5	0.25	4.928	0.246	4.9	0.245
damage	4	0.2	3.961	0.198	3.9	0.195
deficiencies	2	0.1	2.029	0.101	1.9	0.095
outbreak	1	0.05	1.063	0.053	0.9	0.045
infirmity	0	0	0.097	0.005	0.25	0.013
cephalopods	0	0	0.097	0.005	0.25	0.013

- 表6.1:バイグラム言語モデルによるLidstone平滑化の例．文脈(alleged（主張した）,-)、おもちゃのコーパス、表示されている7語に対して計20カウント。Lidstone平滑化は、deficienciesとoutbreakの有効なカウントと確率を増加させることに留意されたい。
- 各nグラムの**ディスカウント**は次のように計算される。

$$d_i = \frac{c_i^*}{c_i} = \frac{(c_i + \alpha)}{c_i} \frac{M}{(M + V\alpha)}.$$

- $d_i > 1 \Leftrightarrow M > c_i V$である．上の例では$20 > 7c_i$.

6.2.2 Discounting and backoff

ディスカウントは、観測されたnグラムから確率質量を借り、それを再分配する。

- Lidstone平滑化では、相対的な頻度の推定値の分母を増やすことによって借用が行われます。
- 借用された確率質量は、すべてのnグラムの分子を増加させることによって再分配される。

絶対ディスカウント

もう1つのアプローチは、すべての観測されたnグラムから同じ量の確率質量を借り、それを観測されていないnグラムだけに再配分することである。

- たとえば、バイグラムモデルに絶対ディスカウント$d = 0.1$を設定し、この確率量を未観測の単語に均等に再配分するとします。
- その結果得られる確率を表6.1に示す。（最掲）

			Lidstone smoothing, $\alpha = 0.1$		Discounting, $d = 0.1$	
	counts	unsmoothed probability	effective counts	smoothed probability	effective counts	smoothed probability
impropriety	8	0.4	7.826	0.391	7.9	0.395
offense	5	0.25	4.928	0.246	4.9	0.245
damage	4	0.2	3.961	0.198	3.9	0.195
deficiencies	2	0.1	2.029	0.101	1.9	0.095
outbreak	1	0.05	1.063	0.053	0.9	0.045
infirmity	0	0	0.097	0.005	0.25	0.013
cephalopods	0	0	0.097	0.005	0.25	0.013

- 表6.1:バイグラム言語モデルによる絶対ディスカウントの例．文脈(alleged（主張した）,-)、おもちゃのコーパス、表示されている7語に対して計20カウント。絶対ディスカウントは、未観測の単語を除くすべての単語の確率を減少させる。

Katzバックオフ

ディスカウントは観測されたデータからある程度の確率量を確保する．この確率量を等しく再配分する必要はない。

- その代わりに、低次言語モデルに**バックオフ**することができます。トライグラムがある場合は、トライグラムを使用します。トライグラムがない場合は、バイグラムを使用します。バイグラムがない場合は、ユニグラムを使用します。
- バイグラムからユニグラムにバックオフする単純なケースでは、バイグラムの確率は次のようになります。

$$c^*(i,j) = c(i,j) - d \quad [6.15]$$

$$p_{\text{Katz}}(i|j) = \frac{c^*(i,j)}{c(j)} \qquad \text{if } c(i,j) > 0$$

$$p_{\text{Katz}}(i|j) = \alpha(j) \times \frac{p_{\text{unigram}}(i)}{\sum_{i':c(i',j)=0} p_{\text{unigram}}(i')} \text{ if } c(i,j) = 0. \quad [6.16]$$

  - 項$\alpha(j) = \sum_{i':c(i',j)>0} d/c(j)$は、文脈$j$に対してディスカウントした確率質量の量を示す。

- この確率量$\alpha(j)$は、すべての未観測のイベント$\{i' : c(i', j) = 0\}$にわたって各単語i'のユニグラム確率に比例して、分割される。
- 足すと1になることの確認.

$$\begin{aligned}\sum_i p_{\text{Katz}}(i|j) &= \sum_{i:c(i,j)>0} \frac{c^*(i,j)}{c(j)} + \sum_{i:c(i,j)=0} \alpha(j) \times \frac{p_{\text{unigram}}(i)}{\sum_{i':c(i',j)=0} p_{\text{unigram}}(i')} \\ &= \sum_{i:c(i,j)>0} \frac{c(i,j)-d}{c(j)} + \alpha(j) \\ &= 1 - \alpha(j) + \alpha(j) = 1\end{aligned}$$

ディスカウント・パラメータdは、開発セットのパフォーマンス(典型的には対数尤度)を最大化するように最適化できます。

- 訓練中に使用されない**held-out data(ヘルドアウトデータ)**に割り当てる尤度

$$l(w) = \sum_{m=1}^{M} \log p(w_m|w_{m-1}, \ldots, w_1), \ [6.41]$$

隠されたコーパス全体を1つのトークンの流れとして扱います。(§6.4.1序文)

6.2.3 *Interpolation

Backoff is one way to combine different order n-gram models. An alternative approach is

interpolation: setting the probability of a word in context to a weighted sum of its probabilities across progressively shorter contexts. Instead of choosing a single n for the size of the n-gram, we can take the weighted average across several n-gram probabilities. For example, for an interpolated trigram model,

$$pInterpolation(wm|wm-1,wm-2) = \lambda 3p*3(wm|wm-1,wm-2) + \lambda 2p*2(wm|wm$$

In this equation, p∗n is the unsmoothed empirical probability given by an n-gram lan- guage model, and λn is the weight assigned to this model. To ensure that the interpolated p(w) is still a valid probability distribution, the values of λ must obey the constraint, nmax n=1 λn = 1. But how to find the specific values? An elegant solution is expectation-maximization. Recall from chapter 5 that we can think about EM as learning with missing data: we just need to choose missing data such that learning would be easy if it weren't missing. What's missing in this case? Think of each word wm as drawn from an n-gram of unknown size, zm ∈ {1 . . . nmax }. This zm is the missing data that we are looking for. Therefore, the application of EM to this problem involves the following generative model: for Each token wm , m = 1, 2, . . . , M do: draw the n-gram size zm ~ Categorical(λ); draw wm ~ p∗ zm (wm | wm−1 , . . . , wm−zm).

If the missing data {Zm } were known, then λ could be estimated as the relative fre- quency,

$$count(Zm = z)\lambda z = [6.17]MM \propto \delta(Zm = z).\,[6.18]m = 1$$

But since we do not know the values of the latent variables Zm , we impute a distribution qm in the E-step, which represents the degree of belief that word token wm was generated from a

n-gram of order zm ,

$$qm(z)Pr(Zm = z|w1 : m; \lambda)[6.19]p(wm|w1 : m - 1, Zm = z) \times p(z) = [6.20]zp(wm|w1 :$$

In the M-step, λ is computed by summing the expected counts under q ,

$$M\lambda z \propto qm(z).\,[6.22]m = 1$$

A solution is obtained by iterating between updates to q and λ. The complete algorithm is shown in Algorithm 10.

Algorithm 10 Expectation-maximization for interpolated language modeling 1: procedure E STIMATE I NTERPOLATED n- GRAM (w1:M , {p∗ n }n∈1:nmax) 2: for z ∈ {1, 2, . . . , nmax } do Initialization 1 3: λz ← nmax 4: repeat 5: for m ∈ {1, 2, . . . , M } do E-step 6: for z ∈ {1, 2, . . . , nmax } do 7: qm (z) ← p∗ z (wm | w1:m−) × λz 8: qm ← Normalize(qm) 9: for z ∈ {1, 2, . . . , nmax } do M-step 1 M 10: λz ← M m=1 qm (z) 11: until tired 12: return λ

6.2.4 *Kneser-Ney smoothing

Kneser-Ney smoothing is based on absolute discounting, but it redistributes the result- ing probability mass in a different way from Katz backoff. Empirical evidence points to Kneser-Ney smoothing as the state-of-art for n-gram language modeling (Goodman, 2001). To motivate Kneser-Ney smoothing, consider the example: I recently visited . Which of the following is more likely: Francisco or Duluth? Now suppose that both bigrams visited Duluth and visited Francisco are unobserved in the training data, and furthermore, that the unigram probability p∗ 1

(Francisco) is greater than p∗1 (Duluth). Nonetheless we would still guess that p(visited Duluth) > p(visited Francisco), because Duluth is a more “versatile” word: it can occur in many contexts, while Francisco usually occurs in a single context, following the word San. This notion of versatility is the key to Kneser-Ney smoothing. Writing u for a context of undefined length, and count(w, u) as the count of word w in context u, we define the Kneser-Ney bigram probability as

$$max(count(w,u)-d,0)count(u), count(w,u)>0 pKN(w|u) = [6.23]\alpha(u) \times pcontinuat$$

Probability mass using absolute discounting d, which is taken from all unobserved n-grams. The total amount of discounting in context u is d × |w : count(w, u) > 0|, and we divide this probability mass among the unseen n-grams. To account for versatility, we define the continuation probability pcontinuation (w) as proportional to the number of ob- served contexts in which w appears. The numerator of the continuation probability is the number of contexts u in which w appears; the denominator normalizes the probability by summing the same quantity over all words w . The coefficient α(u) is set to ensure that the probability distribution pKN (w | u) sums to one over the vocabulary w. The idea of modeling versatility by counting contexts may seem heuristic, but there is an elegant theoretical justification from Bayesian nonparametrics (Teh, 2006). Kneser-Ney smoothing on n-grams was the dominant language modeling technique before the arrival of neural language models.

6.3 Recurrent neural network language models

ニューラルネットワーク

nグラム言語モデルはニューラルネットワークに大きく取って代わられている。

- ニューラル言語モデルは、制限された文脈のnグラム仮定をしません。
- 実際には、コンピュータ的にも統計的にも扱いやすい状態を保ちながら、任意の距離の文脈情報を組み込むことができます。

ニューラル言語モデルの背後にある最初の洞察は，単語予測を識別学習タスク（今までの文脈から次にどの単語を選ぶか予測する問題）として扱うことである。

- 言語モデリング（確率$p(w|u)$の計算）を機械学習問題として扱い，コーパスの対数条件付き確率を最大化するパラメータを推定できる。
- この考えはニューラル言語モデル(例:Rosenfeld,1996;Roarkら、2007)以前から存在していた。

第二の洞察は、二つの密なK次元数値ベクトル$\beta_w \in \mathbb{R}^K$と$v_u \in \mathbb{R}^K$の関数として確率分布$p(w|u)$を再パラメータ化することである。

$$p(w|u) = \frac{\exp(\beta_w \cdot v_u)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot v_u)}, \quad [6.25]$$

- β_wは単語を表すベクトル，v_uは文脈を表すベクトル。
- ここで、$\beta_w \cdot v_u$は内積である。

- いつものように、分母は確率分布が適切に正規化されることを保証する。
- この確率ベクトルは、ドット積のベクトルに**ソフトマックス**変換(§3.1参照)を適用するのと等価である。

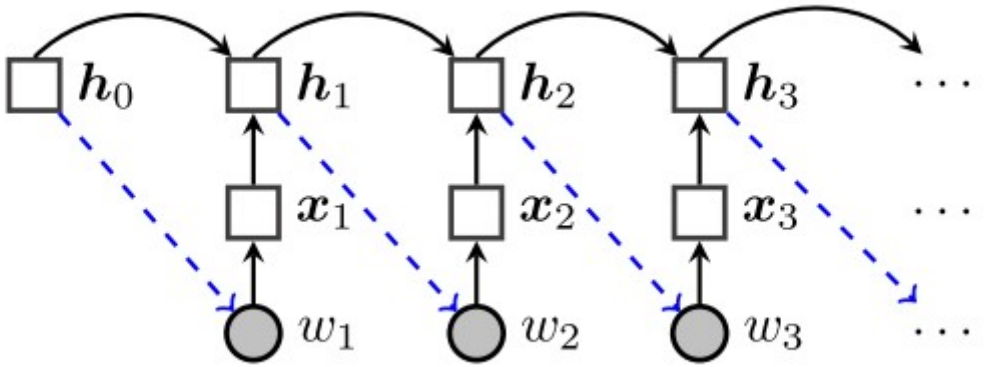
$$p(\cdot|u) = \mathrm{SoftMax}([\beta_1 \cdot v_u, \beta_2 \cdot v_u, \ldots, \beta_V \cdot v_u]). \ [6.26]$$

$$\mathrm{SoftMax}\Big(\Theta^{(z\to y)} \cdot z + b\Big)_j = \frac{\exp(\theta_j^{(z\to y)} \cdot z + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\theta_{j'}^{(z\to y)} \cdot z + b_{j'})}, \ [3.5], [3.3]$$

- 単語ベクトルβ_wはモデルのパラメータであり，直接推定される。
- 文脈ベクトルv_uは、モデルに応じて様々な方法で計算することができる。

RNN(リカレントニューラルネットワーク)

単純だが効果的なニューラル言語モデルは**リカレントニューラルネットワーク**から構築できる。(RNN;Mikolov et al.,2010)



- 図6.1:展開された計算グラフとして表示されたRNN言語モデル。実線は直接計算、点線は確率的依存関係、丸印は確率変数、四角印は計算ノードを示す。
- 基本的な考え方は、単語列を移動しながら文脈ベクトルh_mを再帰的に更新することです。
  - 文脈ベクトルh_mは、単語列の位置mにある文脈情報を表します。

RNN言語モデルは以下のように定義される。

$$x_m \equiv \phi_{w_m} \in \mathbb{R}^K \quad [6.27]$$

$$h_m = \mathrm{RNN}(x_m, h_{m-1}) \in \mathbb{R}^K \quad [6.28]$$

$$p(w_{m+1}|w_1, w_2, \ldots, w_m) = \frac{\exp(\beta_{w_{m+1}} \cdot h_m)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot h_m)}, \quad [6.29]$$

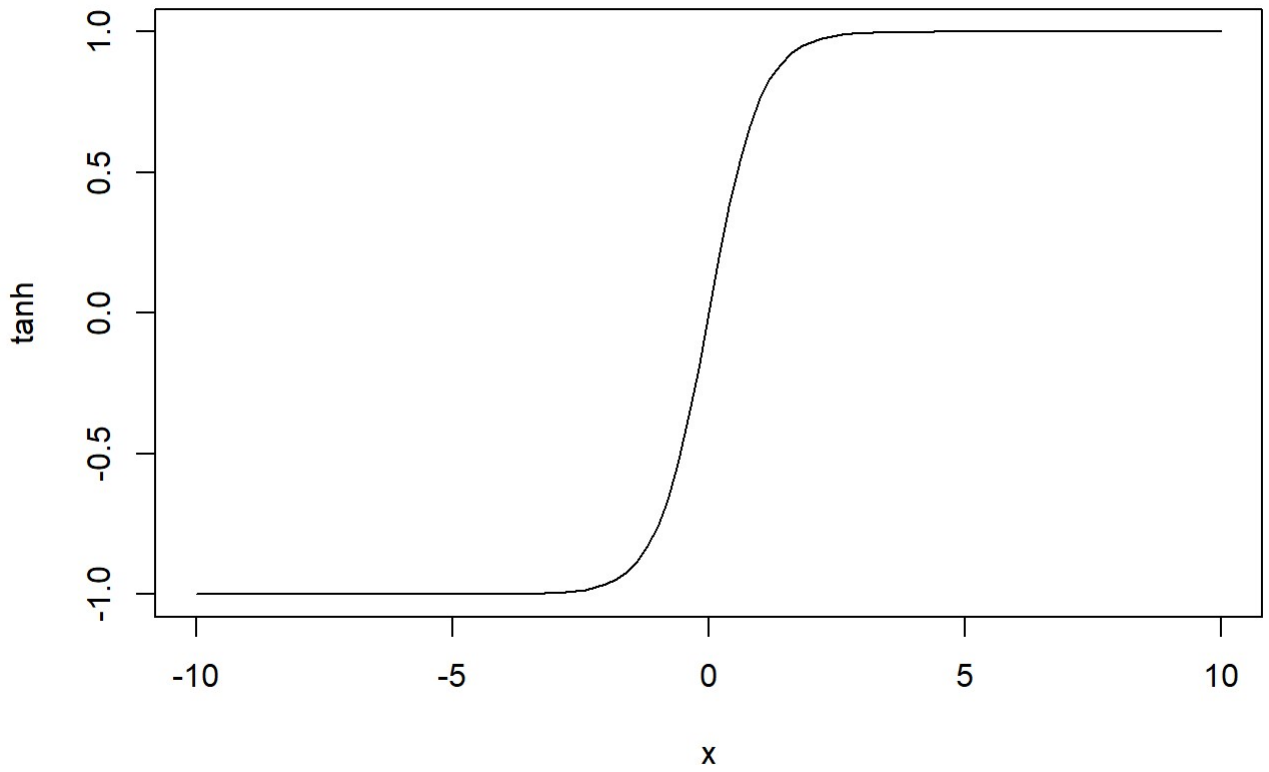
- ここで、ϕは**word embedding（単語埋め込み）**の行列であり、x_mは単語w_mの埋め込みを示す。

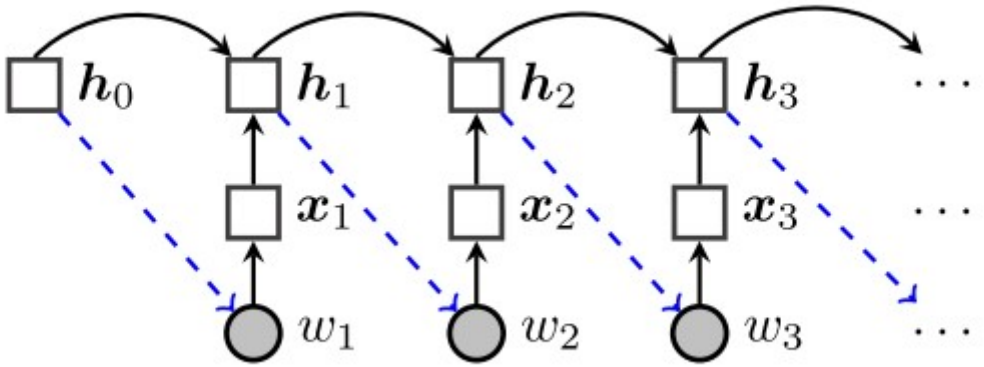
    - w_mからx_mへの変換は、**ルックアップ層**と呼ばれることもあります。
    - これは、テーブル内の各単語の埋め込みを単にルックアップするためです。§3.2.4参照。
- **Elmanユニット**は単純再帰演算(Elman, 1990)を定義します。

$$\text{RNN}(x_m, h_{m-1}) \equiv g(\Theta h_{m-1} + x_m), \; [6.30]$$

    - $\Theta \in \mathbb{R}^{K \times K}$は再帰行列であり、$g$は非線形変換関数であり、しばしば要素ごとの$\tanh$(§3.1参照)として定義される。
    - オリジナルのElmanネットワークでは、$\tanh$の代わりにシグモイド関数が使用されていました。RNNにおけるさまざまな非線形性の利点と欠点を数学的に説明するには、Cho(2015)の講義ノートを参照されたい。
    - $\tanh$関数は**squashing(収縮?)関数**として動作し、h_mの各要素が$[-1, 1]$の範囲に制限されることを保証します。

```
plot(tanh, -10,10)
```


tanh
1.0
0.5
0.0
-0.5
-1.0
-10
-5
0
5
10
x



- 図6.1（再掲）:展開された計算グラフとして表示されたRNN言語モデル。実線は直接計算、点線は確率的依存関係、丸印は確率変数、四角印は計算ノードを示す。

RNNの備考

- 各w_mは、文脈ベクトルh_{m-1}のみに依存するが、この文脈ベクトルh_{m-1}は、順に、以前のすべてのトークン$w_1, w_2, \ldots w_{m-1}$に影響される：
  - 繰り返し演算によって、w_1はh_1に影響を与え、h_2に影響を与えるというように、情報がh_{m-1}まで伝播された後、w_m(図6.1参照)まで伝播されます。
  - これは、nワードウィンドウ外の情報が無視されるnグラム言語モデルとの重要な違いである。
- 原則として、RNN言語モデルは、長いテキスト範囲での数の一致などの長期的な依存関係を処理できますが、この情報がベクトルh_mのどこに正確に表現されているかを知ることは困難です。
- 主な制限は、収縮関数gを繰り返し適用することによって情報が減衰されることである。

    - 後述する**long short-term memory(長期短期メモリ?)**(LSTMs)は、この問題に対処するRNNの変形であり、メモリセルを使用して、非線形性を適用しないシーケンスを通して情報を伝搬する(Hochreiter and Schmidhuber, 1997)。

RNNのボトルネック

式[6.29]の分母$\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot h_m)$は計算上のボトルネックである。なぜなら、それは語彙全体の和を伴うからである。

- 一つの解決策は**階層的なsoftmax関数**を使うことであり、これは語彙をツリーに編成することによって、より効率的に合計を計算します。(Mikolov et al.,2011)
- 別の戦略は,**noise-contarstive estimation(雑音対比推定)**(Gutmann and Hyv¨arinen, 2012)のような代替メトリックを最適化することである。これは，雑音分布から生成された人工的なインスタンスから観測されたインスタンスを区別することによって学習する(Mnih and Teh, 2012)。
- これらの戦略の両方が§14.5.3に記述されている。

6.3.1 Backpropagation through time

RNN言語モデルは、以下のパラメータを有する。

- $\phi_i \in \mathbb{R}^K$, 「入力」単語ベクトル(これらは、それぞれの単語がK次元空間に埋め込まれているので、**単語の埋め込み**と呼ばれることがある;第14章参照);
- $\beta_i \in \mathbb{R}^K$、「出力」単語ベクトル;

- $\Theta \in \mathbb{R}^{K \times K}$，再帰作用素;
- h_0、初期状態。

これらのパラメータのそれぞれは、訓練コーパス にわたる目的関数$L(w)$を定式化し、次いで、逆伝搬を適用して、訓練データのミニバッチ(§3.3.1参照)からパラメータに関する勾配を得ることによって推定することができる。

- 勾配ベースの更新は、確率勾配降下法(§2.6.2参照)などのオンライン学習アルゴリズムから計算できます。

RNNへの逆伝搬の応用は，時間mでのユニットの勾配が時間$n < m$でのユニットの勾配に順番に依存するので，**backpropagation through time（時間を経た逆伝播）**として知られている。

- l_{m+1}をワード$m+1$の負の対数尤度とする。

$$l_{m+1} = -\log p(w_{m+1}|w_1, w_2, \ldots, w_m). \quad [6.31]$$

- 再帰行列Θの個々の成分である$\theta_{k,k'}$のような各パラメータに関してこの損失l_{m+1}の勾配が必要である。
- 損失l_{m+1}はh_mを通じてのみパラメータに依存するので、微分の連鎖則を適用することができる。（以下に式を再掲する．）

$$h_m = g(x_m + \Theta h_{m-1}) \qquad [6.33], [6.27], [6.28]$$

$$p(w_{m+1}|w_1, w_2, \ldots, w_m) = \frac{\exp(\beta_{w_{m+1}} \cdot h_m)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot h_m)}, \quad [6.29]$$

$$\frac{\partial l_{m+1}}{\partial \theta_{k,k'}} = \frac{\partial l_{m+1}}{\partial h_m} \frac{\partial h_m}{\partial \theta_{k,k'}}. \quad [6.32]$$

- ベクトルh_mはいくつかの点でΘに依存する。Θに前の状態h_{m-1}を乗じてh_mを算出する。さらに前の状態h_{m-1}もΘに依存する。

$$h_m = g(x_m + \Theta h_{m-1}) \qquad [6.33]$$

$$\frac{\partial h_{m,k}}{\partial \theta_{k,k'}} = g'(x_{m,k} + \theta_k \cdot h_{m-1}) \left(h_{m-1,k'} + \theta_k \cdot \frac{\partial h_{m-1}}{\partial \theta_{k,k'}} \right), \quad [6.34]$$

- この式における重要な点は、$\partial h_m / \partial \theta_{k,k'}$の導関数が$\partial h_{m-1} / \partial \theta_{k,k'}$に依存し、それが次に$\partial h_{m-2} / \partial \theta_{k,k'}$などに依存して、最初の状態$h_0$に到達することである。

各微分$\partial h_m / \partial \theta_{k,k'}$は何度も再利用される：それは損失$l_m$からの逆伝搬で現れるが、全てのその後の損失$l_n, (n > m)$でも現れる。

- Torch(Collobert et al.,2011)やDyNet(Neubigら、2017)のようなニューラルネットワークツールキットは，必要な導関数を自動的に計算し，将来の使用のためにそれらをキャッシュする。

- 第3章で考察したフィードフォワードニューラルネットワークとの重要な違いは、計算グラフのサイズが固定されておらず、入力の長さによって変化することである。
  - これは、TensorFlow(Abadiら、2016)のような静的な計算グラフを中心に設計されたツールキットにとって困難をもたらします。
  - https://www.tensorflow.org/tutorials/recurrent (https://www.tensorflow.org/tutorials/recurrent) (retrieved Feb 8, 2018).

6.3.2 Hyperparameters

RNN言語モデルは，良好な性能を保証するために調整しなければならないいくつかのハイパーパラメータを持つ。

- モデル容量は、単語および文脈ベクトルKのサイズによって制御される.
  - これは、nグラム言語モデルのサイズに幾分類似した役割を果たす。
  - 語彙に対して大きなデータセット(すなわち、トークン対タイプの比率が大きい)の場合は，単語と文脈の間のより微妙な区別を可能にする大きなKを持つモデルを推定する余裕がある。
  - データセットが比較的小さい場合、Kも小さくなければならず、さもなければ、モデルは訓練データを「記憶」し、一般化に失敗する可能性がある。
  - 残念ながら、この一般的なアドバイスはまだKを選ぶための具体的な公式にはなっておらず、試行錯誤が必要である。
- また、計算の一部の要素をランダムに0に設定し、単語または文脈ベクトルの特定の次元に過度に依存しないように強制する**ドロップアウト**によって、オーバーフィッティングを防

止することもできる(Srivastava et al.,2014)。
    - ドロップアウト率も開発データで調整する必要があります。

6.3.3 Gated recurrent neural networks

原理的には，RNNは無限に長いシーケンスにわたって情報を伝搬することができる。

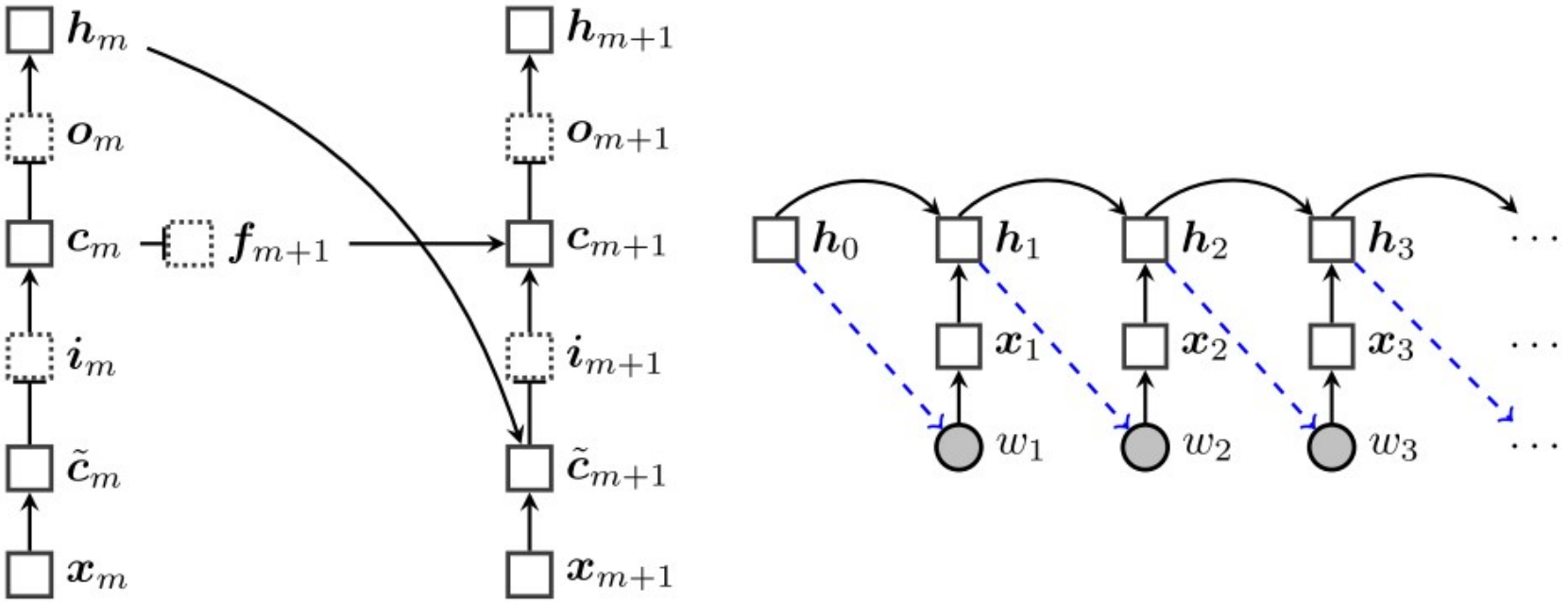
- しかし実際には、非線形再帰関数を繰り返し適用すると、この情報はすぐに減衰します。
- 同じ問題が学習にも影響する：
    - 逆伝搬は、ゼロに減衰する**勾配を消滅**させたり、無限に増加する**勾配を爆発**させたりする(Bengio et al., 1994)。
    - 爆発する勾配の問題は、ある最大値で勾配をクリッピングすることによって対処することができる(Pascanu et al., 2013)。
    - その他の問題は、モデル自体を変更することで対処する必要があります。

LSTM

long short-term memory(LSTM; Hochreiter and Schmidhuber, 1997)はRNNの一般的な亜種であり，これらの問題に対してよりロバストである。

- このモデルは**メモリセル**c_mで隠れ状態h_mを増強する。
- 各時間mにおけるメモリセルの値は、その前の値c_{m-1}と、更新$\tilde{c}_m$の2つの量のゲート和である。
- 更新$\tilde{c}_m$は現在の入力x_mと前の隠れ状態h_{m-1}とから計算される。

- 次の状態h_mはメモリセルから計算される。
- メモリセルは、更新中に非線形収縮関数を通過しないので、情報が長距離にわたってネットワークを伝搬する可能性がある。



- 図6.2左:LSTMアーキテクチャ。ゲートは点線エッジのボックスで表示されます。LSTM言語モデルでは、各h_mは、次の単語w_{m+1}を予測するために使用される。

ゲートは入力と以前の隠れ状態の関数である。

- これらは、要素ごとのシグモイド活性化$\sigma(x) = (1 + \exp(-x))^{-1}$から計算され、それらの値が$[0, 1]$の範囲にあることが保証される。

- したがって、これらは、ソフト（？）で微分可能な論理ゲートと見なすことができる。

完全な更新式は次のとおりです。

$$
\begin{aligned}
f_{m+1} &= \sigma(\Theta^{(h\to f)} h_m + \Theta^{(x\to f)} x_{m+1} + b_f) && \text{forget gate} && [6.35]\\
i_{m+1} &= \sigma(\Theta^{(h\to i)} h_m + \Theta^{(x\to i)} x_{m+1} + b_i) && \text{input gate} && [6.36]\\
o_{m+1} &= \sigma(\Theta^{(h\to o)} h_m + \Theta^{(x\to o)} x_{m+1} + b_o) && \text{output gate} && [6.39]\\
\tilde{c}_{m+1} &= \tanh(\Theta^{(h\to c)} h_m + \Theta^{(w\to c)} x_{m+1}) && \text{update candidate} && [6.37]\\
c_{m+1} &= f_{m+1} \odot c_m + i_{m+1} \odot \tilde{c}_{m+1} && \text{memory cell update} && [6.38]\\
h_{m+1} &= o_{m+1} \odot \tanh(c_{m+1}) && \text{output.} && [6.40]
\end{aligned}
$$

- 演算子⊙は要素単位の積(アダマール積)です。
- 各ゲートは、前の隠れ状態($\Theta^{(h\to f)}$など)と現在の入力($\Theta^{(x\to f)}$など)、およびベクトルオフセット(例:b_f)をパラメータ化する重みのベクトルによって制御されます。
- 全体の動作は、$(h_m, c_m) = \mathrm{LSTM}(x_m, (h_{m-1}, c_{m-1}))$として非公式に要約することができ、$(h_m, c_m)$は、トークン$m$を読み取った後のLSTM状態を表す。

LSTMは，広範囲の問題にわたって標準RNNより優れている。

- Sundermeyerら(2012)によって言語モデリングのために最初に用いられたが、より一般的に適用することができる。
- 文脈ベクトルh_mは、mの位置までの入力列の完全な表現として扱うことができ、次章で述べるように、トークン列上のあらゆるラベリング作業に用いることができる。

- いくつかのLSTM亜種があり，その中で，Gated Recurrent Unit(Cho et al., 2014)は最もよく知られているものの一つである。
- 多くのソフトウェアパッケージにはさまざまなRNNアーキテクチャが実装されているため、ユーザの観点からRNNアーキテクチャを選択するのは簡単です。
- Jozefowicz et al.(2015)は、2015年頃のさまざまなモデル化の選択肢を実証的に比較している。

6.4 Evaluating language models

言語モデリングは、通常、それ自体がアプリケーションではありません。

言語モデルは、通常、より大きなシステムのコンポーネントであり、理想的には、**外部的に**評価されます。

- これは，言語モデルが機械翻訳や音声認識のようなアプリケーションタスクの性能を改善するかどうかを評価することを意味する。
- しかし、これは難しいことが多く、言語モデリングとは無関係なシステム全体の詳細に依存します。

対照的に，**内部評価**はタスク中立である。

- 内部メトリックのパフォーマンス向上は、さまざまなタスクで外部メトリックを向上させることが期待できますが、内部メトリックを過剰に最適化するリスクが常に存在します。

この項では、いくつかの内部メトリックについて説明しますが、内部パフォーマンスの向上が

実際のアプリケーションに引き継がれるように、外部評価を実行することの重要性に注意してください。

6.4.1 Held-out likelihood

確率的言語モデルの目標は、単語トークン列の確率を正確に測定することである。

- したがって、固有の評価尺度は、言語モデルが、訓練中に使用しない**held-out data（保留データ）**に割り当てる尤度である。具体的には

$$l(w) = \sum_{m=1}^{M} \log p(w_m | w_{m-1}, \ldots, w_1), \ [6.41]$$

  保留したコーパス全体を1つのトークンの流れとして扱います。

通常、不明な単語は$\langle \mathrm{UNK} \rangle$トークンにマッピングされます。

- これは、訓練データ上で$\langle \mathrm{UNK} \rangle$の確率を推定する必要があることを意味する。
- これを行う1つの方法は、語彙$\mathcal{V}$を訓練データ内の最も高いカウントを有する$V-1$ワードに固定し、次いで他のすべてのトークンを$\langle \mathrm{UNK} \rangle$に変換することである。
  - $|\mathcal{V}| = V$?。一つの単語だけ$\langle \mathrm{UNK} \rangle$にするということ？　それとも出現一回だけの単語を$\langle \mathrm{UNK} \rangle$にするということを言いたかったのだろうか。
- 語彙$\mathcal{V}$にない用語を扱う他の戦略は、§6.5で論じられる。

6.4.2 Perplexity

保留尤度は通常、対数尤度の情報理論量への決定論的変換である**Perplexity（パープレキシティ・困惑）**として提示される。

$$\mathrm{Perplex}(w) = 2^{-l(w)/M}, \ [6.42]$$

- ここでMは保留コーパスにおけるトークン総数である。
- 低いパープレキシティは高い尤度に対応するため、このメトリックでは低いスコアの方が適しています。つまり、パープレキシティが少ない方が適しています。

次のような特殊なケースがあります。

- 完全言語モデルの極限では、保留コーパスの尤度1となる。$\mathrm{Perplex}(w) = 2^{-M/M \cdot \log_2 1} = 2^0 = 1$である。
- 逆の極限では、保留コーパスの尤度0となる。無限のパープレキシティ$\mathrm{Perplex}(w) = 2^{-M/M \cdot \log_2 0} = 2^\infty = \infty$に対応する.
- 語彙中のすべての単語に対して$p(w_i) = 1/V$である一様なユニグラムモデルを仮定すると以下の尤度になる。

$$l(w) = \sum_{m=1}^{M} \log_2 \frac{1}{V} = -M \log_2 V$$
$$\text{Perplex}(w) = 2^{M/M \cdot \log_2 V} = V.$$

- データを見なくてもこのような言語モデルを作成できるため、これは「最悪の合理的ケース」シナリオです。
  - つまりパープレキシティがVより大きければ無意味である。

実際には、言語モデルは1とVの間の範囲のパープレキシティを与える傾向があります。

- ベンチマークとなる小さなデータセットは**Penn Treebank**で、約100万$= 10^6$のトークンが含まれています。
- その語彙は1万$= 10^4$語に制限されており、他のすべてのトークンは特別な$\langle \text{UNK} \rangle$記号にマップされています。
  - このデータセットでは、良く平滑化された5グラムモデルは141のパープレキシティを達成し、(Mikolov and Zweig, Mikolov and Zweig)
  - LSTM言語モデルは約80のパープレキシティを達成した。(Zaremba, Sutskever, and Vinyals, Zaremba et al.) https://corochann.com/penn-tree-bank-ptb-dataset-introduction-1456.html (https://corochann.com/penn-tree-bank-ptb-dataset-introduction-1456.html)
  - LSTMアーキテクチャに対する種々の強化は、60以下のパープレキシティをもたらす。(Merity et al., 2018)
- 大規模な言語モデリング・データセットは1B(= 10^9) Word Benchmark(Chelba et al.,

2013)であり、ウィキペディアからのテキストが含まれています。
    - このデータセットでは、複数のLSTM言語モデルを平均することにより、約25のパープレキシティが得られる。(Jozefowicz et al.,2016)

6.5 Out-of-vocabulary words

語彙$\mathcal{V}$は有限集合でないかもしれない

ここまでは、**閉じた語彙**設定を仮定してきた—語彙$\mathcal{V}$は有限集合であると仮定した。

- 現実的な応用シナリオでは、この仮定は成立しないかもしれない。
- 例えば、新聞記事の翻訳の問題を考えてみよう。2017年1月6日付のロイターの記事には、次のような文章が掲載されている:
- The report said U.S. intelligence agencies believe Russian military intelligence, the **GRU**, used intermediaries such as **WikiLeaks**, **DCLeaks.com** and **the Guccifer** 2.0 ”persona” to release emails… (報告書によると、米国の情報機関は、ロシアの軍事情報機関である**ロシア軍参謀本部**が、**ウィキリークス**や**DCLeaks.com**、**the Guccifer**2.0ペルソナなどの仲介者を使って電子メールを公開したと考えています…。)
    - (Bayoumy, Y. and Strobel, W. (2017, January 6). U.S. intel report: Putin directed cy- ber campaign to help Trump. Reuters. Retrieved from http://www.reuters.com/article/ (http://www.reuters.com/article/) us-usa-russia-cyber-idUSKBN14Q1T8 on January 7, 2017.)

2003年にリリースされたGigawordコーパスで言語モデルをトレーニングしたとします。

- https://catalog.ldc.upenn.edu/LDC2003T05 (https://catalog.ldc.upenn.edu/LDC2003T05)
- 太字の用語は、この時点では存在しなかったか、広く知られていなかったかのいずれかです。語彙に入っている可能性は低い。
- 同じ問題は、新技術、これまで知られていなかった個人、新しい単語(例:hashtag)、数字など、他のさまざまな用語でも発生する可能性があります。

未知の単語を$\langle \mathrm{UNK} \rangle$とする

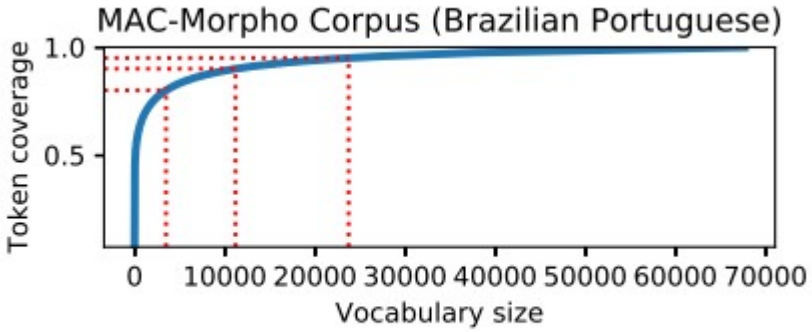
1つの解決策は、そのようなすべての条件を特別なトークン$\langle \mathrm{UNK} \rangle$でマークすることです。

- 言語モデルを訓練する間に，語彙(しばしば最も一般的なKの用語)を予め決定し，訓練データ中の他の全ての用語をUNKとしてマークする。
- 事前に語彙サイズを決定したくない場合は、各単語タイプの最初の出現を単に$\langle \mathrm{UNK} \rangle$としてマークするという方法もあります。

未知の単語は区別した方がよい

しかし、さまざまな未知の単語の可能性については、区別した方がよいことが多い。

- これは、形態素が豊富で、各単語に多くの屈折がある言語では特に重要です。
- 例えば、ポルトガル語は形態学的な観点からはそれほど複雑ではないが、それぞれの動詞には数十の屈折形がある。(図4.3 b参照)



(b) News articles in Brazilian Portuguese

- そのような言語では、コーパスでは遭遇しない多くの種類の単語が存在し、それにもかかわらず、言語の形態規則から予測可能である。
- 若干不自然な英語の例を使うために、transfenestrateが語彙に含まれている場合、我々の言語モデルは、訓練データに現れていなくても、過去形のtransfenestratedにゼロでない確率を割り当てるべきである。

未知の単語の区別方法

これを実現する1つの方法は、単語レベルの言語モデルを**文字レベルの言語モデル**で補完することです。

- このようなモデルでは、nグラムやRNNを使用することができますが、ASCII文字やUnicode文字のセットと同じ固定語彙を使用します。
- 例えば，Ling et al. (2015)は文字上のLSTMモデルを提案し，
- Kim(2014)は畳込みニューラルネットワークを採用した。

言語学的に動機づけられたアプローチは、単語を意味のあるサブワード単位(§9参照)に分割することである。

- 例えば、BothaとBlunsom(2014)は形態素にベクトル表現を導入し、それを対数双線形言語モデルに組み込む。
- Bhatia et al. (2016)は、LSTMに形態素ベクトルを組み込む。

Additional resources

- 様々なニューラルネットワークアーキテクチャが言語モデリングに適用されてきた。
- 以前のノンリカレントアーキテクチャは，ニューラル確率言語モデル(Ben-gio et al., 2003)と対数双線形言語モデル(Mnih and Hinton, 2007)を含んでいない。
- これらのモデルについての詳細は、Goodfellow et al.(2016)の本文を参照されたい。