

LLMの基礎（1）トークンと言語モデル

（高校生・大学初年次向け）

文字から「次のトークンの確率」まで

Planned and edited by Yoshio Okamoto / Written by Claude Opus 5.5 (Anthropic)

Ver. 0.1 October 5, 2026

※ 以下の内容はAIが生成したものであり、一応編者がチェックしていますが、間違いや思い込みが含まれている場合があることはご注意ください。

© 2026 Yoshio Okamoto. 教育目的での利用は自由です（出典を明記してください）。

はじめに

スマートフォンで「よろしく」と打つと、「お願いします」が候補に出てきます。これは、それまでに打った言葉から**次に来そうな言葉**を予想しているのです。実は ChatGPT や Claude のような LLM（大規模言語モデル）も、やっていることの本質はこれと同じ「次を予想する」ことです。ただし、その予想の精度がけた違いに高いのです。

「深層学習（4）系列データと Transformer」では、LLM の**内部**（Attention と Transformer）の計算を見ました。この「LLMの基礎」シリーズでは、その**外側**を扱います。第1回は、LLM の**入口**（文章をどうやって数にするか）と**出口**（数からどうやって「次の言葉の確率」を出すか）を見たあと、いちばん単純な言語モデルである「数えるだけのモデル」を作り、それと比べることで LLM のすごさを考えます。

回	題目	内容
第1回	トークンと言語モデル	文章がどう数になり、どう確率として出てくるか
第2回	事前学習とスケーリング則	大量の文章で学習すると何が起きるか
第3回	対話モデルへ	「続きを書く」モデルが、質問に答えるモデルになるまで

読み方 本文は、高校の数学（条件付き確率、指数関数と対数、数列の和 $\sum$ 、ベクトルの内積）を使って読めるように書きました。行列を使った書き方、偏微分による勾配の計算、ラグランジュの未定乗数法による証明などの**数学的な根拠**は、**脚注と巻末の付録A**にまとめました。付録を読み飛ばしても本文の流れは追えます。例題の数値はすべて実際にプログラムで計算した値です（付録B）。

記号の約束

- 語彙（モデルが扱うトークンの種類）の数を V 、トークンを表すベクトルの成分の数（次元）を d と書きます。
- 文章をトークンに分けたものを $x_1, x_2, \dots, x_T$ と書きます（添字は何番目か）。
- $p(b \mid a)$ は「 a という条件のもとで b が起こる確率」（条件付き確率）で、数学Aの記号 $P_A(B)$ にあたります。
- $\log$ は自然対数（底が e の対数）です。
- $N(a)$, $N(a, b)$ は、学習用の文章の中で語 a 、語の組「 a の次に b 」が現れた回数です。

1 言語モデルとは何か

1.1 文章の確率を「次のトークンの確率」の掛け算に分ける

言語モデル (language model) とは、文章 (トークンの並び) に確率を与えるモデルです。「地震 が 起きた」という並びはよくありそうなので確率が大きく、「た 起き が 地震」はまずないので確率が小さい、というふうに、文章の「ありそうさ」を数で表します。

文章全体の確率は、確率の乗法定理 (数学Aの $P(A \cap B) = P(A) P_A(B)$) を繰り返し使うと、

$$p(x_1, x_2, \dots, x_T) = p(x_1) \times p(x_2 | x_1) \times p(x_3 | x_1, x_2) \times \dots \times p(x_T | x_1, \dots, x_{T-1})$$

と、「それまでのトークンから、次のトークンを当てる確率」の掛け算に分けられます¹。

たとえば3語の文なら、

$$p(\text{地震, が, 起きた}) = p(\text{地震}) \times p(\text{が} | \text{地震}) \times p(\text{起きた} | \text{地震, が})$$

です。「最初が『地震』である確率」×「『地震』の次が『が』である確率」×「『地震が』の次が『起きた』である確率」、という具合です。1文字ずつヒントが増えていく言葉当てクイズで、各段階で正解する確率を掛けていくようなものです。

この分け方は近似ではなく、いつでも成り立つ等式です。どんな文章の確率も、「次を当てる確率」の掛け算として正確に書けます。近似が入るのは、各段階の「次を当てる確率」をモデルでどう計算するか、という部分です。LLM は、この「次を当てる確率」を計算する装置です。

文章を生成するときは、この確率に従って次のトークンを1つ選び、末尾に加えて、また次を予測する、ということを繰り返します。スマホの予測変換で、出てきた候補をタップし続けて文を作るのと同じです。

1.2 LLM の入口と出口

LLM の計算の流れを、入口から出口まで並べると図1のようになります。

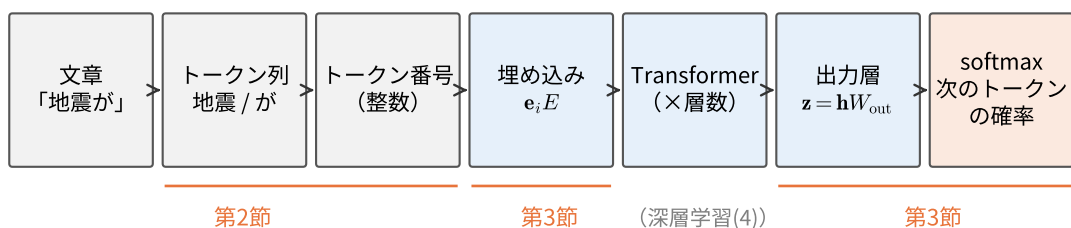


図1 LLM の入口から出口まで。この資料では、文章をトークン番号にする部分 (第2節) と、埋め込み・出力層 (第3節) を扱う。中央の Transformer は「深層学習 (4)」で扱った。図中の $e_i E$, $z = hW_{out}$ は行列を使った書き方で、本文の第3節でことばで説明し、式は付録A.2にまとめた。

¹大学の教科書では、掛け算の記号 $\prod$ (和の記号 $\sum$ の掛け算版) を使って $p(x_1, \dots, x_T) = \prod_{t=1}^T p(x_t | x_{<t})$ と書きます。 $x_{<t}$ は $x_1, \dots, x_{t-1}$ をまとめて表す記号です。導き方は付録A.1。

確率が得られたあと、そこから実際に1つのトークンを選ぶ段階（3.3節のくじ引き）と、ここで使う「温度」については、別のノート「LLMの計算で確率が入る段階と『温度』」で扱いました。

2 トークン化：文章を部品に分ける

2.1 なぜ文字でも単語でもないのか

コンピュータに文章を入れるには、まず決まった種類の記号（トークン）の並びに変える必要があります。どんな大きさの部品に分けるかには、次のような得失があります。

- ・ **文字単位**：種類は少なくて済みますが、文章がとても長い記号の列になります。「深層学習（4）」で見たように、Attention の計算量は列の長さの2乗に比例するので、長い列は不利です。
- ・ **単語単位**：列は短くなりますが、単語の種類は膨大です。活用形や新語、固有名詞まで含めると、語彙が決まりません。学習のときに見なかった単語（**未知語**）はまったく扱えません。

ブロックのおもちゃにたとえると、文字単位は「小さな基本ブロックだけ」で、何でも作れますが組み立てに手間がかかります。単語単位は「完成品の形をした専用パーツ」で、すぐに作れますが、パーツの種類が無限に必要になり、持っていない形は作れません。

実際の LLM は、その中間の**サブワード**（単語の断片）を単位にします。よく出てくる語は1つのトークンに、まれな語はいくつかの断片に分けて表します。よく使う形は大きめのパーツを用意しておき、めったに使わない形は小さなブロックを組み合わせで作る、というやり方です。

2.2 バイト対符号化（BPE）

サブワードの語彙を作る代表的な方法が**バイト対符号化**（Byte Pair Encoding, BPE）です。手順は単純です。

1. はじめは1文字ずつを記号とする。
2. 学習用の文章の集まり（**コーパス**）全体で、隣り合って最もよく現れる2つの記号の組を探す。
3. その組を1つの新しい記号にまとめ、語彙に加える。
4. 語彙が決めた数になるまで、2と3を繰り返す。

私たちがよく使う言葉を略語にするのと似ています。「スマートフォン」を何度も書くうちに「スマホ」と縮めるように、よく一緒に出てくる組ほど、先に1つにまとめていきます。

例題 次の4つの単語だけからなるコーパスで試します（付録Bのプログラム）。語末には単語の区切りを表す記号「_」を付けます。

単語	low	lower	newest	widest
出現回数	5	2	6	3

はじめの語彙は `_ d e i l n o r s t w` の11種類で、コーパス全体は95個の記号です（たとえば `low_` は4記号で5回なので20個）。まとめる操作を10回行くと次のようになりました。

回	まとめた組	出現回数	語彙の数	総トークン数
1	<code>e + s → es</code>	9	12	86
2	<code>es + t → est</code>	9	13	77
3	<code>est + _ → est_</code>	9	14	68
4	<code>l + o → lo</code>	7	15	61
5	<code>lo + w → low</code>	7	16	54
6	<code>n + e → ne</code>	6	17	48
7	<code>ne + w → new</code>	6	18	42
8	<code>new + est_ → newest_</code>	6	19	36
9	<code>low + _ → low_</code>	5	20	31
10	<code>w + i → wi</code>	3	21	28

（出現回数と同じ組が複数あるときは、プログラムが最初に見つけた組を選んでいきます。）

最初に「es」「est」が作られるのは、`newest`（6回）と `widest`（3回）の両方に含まれ、合わせて9回現れるからです。語彙を10個増やすと、コーパス全体の記号の数は95から28まで減りました（図2）。

語彙を増やすほど、同じ文章を少ないトークンで表せます。その代わり、次の第3節で見る「埋め込み」と「出力層」の表が大きくなります。語彙の数は、この2つのかね合いで決めます。

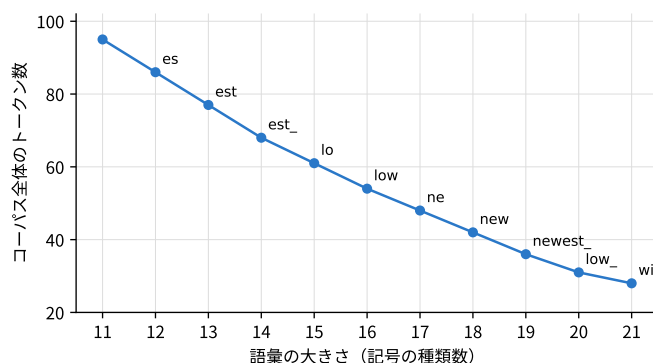


図2 BPEのまとめる操作を進めたときの語彙の数とコーパス全体のトークン数。点の横の文字は、その回に新しく作られた記号。

2.3 知らない単語も分割できる

最終的に、各単語は次のように分割されます。

`low_` × 5 `low e r _` × 2 `newest_` × 6 `wi d est_` × 3

学習したまとめ方の規則を、学習した順に当てはめると、コーパスになかった単語も分割できます。

lowest → low est_ newer → new e r _ wider → wi d e r _

「lowest」は一度も見たことのない単語ですが、「low」と「est_」という知っている部品に分けられました。

どんな単語も、最悪でも1文字ずつに分ければ表せるので、「知らない単語なので扱えない」ということが起きません。大きなパーツで作れないときは、小さなブロックに戻ればよいのです。

2.4 実際のトークナイザ

実際の LLM では、文字の代わりに、文字を UTF-8 という方式で符号化したバイト（0～255 の256種類の数）から BPE を始める方式がよく使われます。たとえば「地震」は UTF-8 では

地 = e5 9c b0, 震 = e9 9c 87

の6バイト（16進数で書いています）です。世界中のどんな文字もバイトの並びで表せるので、どの言語のどんな文字も必ずトークンに変換できます。よく出てくる文字や単語はまとめられて1トークンになり、まれな漢字などは複数のトークンに分かれます。

語彙の数はモデルによって違い、たとえば Llama 3 では約12.8万（128,256）です。語彙はコーパスの統計で決まるので、同じ内容の文章でも、言語によって必要なトークン数が違います。トークン数は、計算量や、一度に扱える文章の長さ（第3回）に直接かわります。

3 埋め込みと出力層：トークン番号とベクトルの橋渡し

3.1 トークン番号をベクトルにする：埋め込み

トークン化によって、文章は整数（トークン番号）の並びになりました。しかし、番号のままでは困ります。

クラスの出席番号を考えてください。5番の人が3番の人より「大きい」わけでも、5番と6番の人が「似ている」わけでもありません。番号は区別するための札であって、大小や近さには意味がないのです。トークン番号も同じです。

そこで、各トークンに d 個の数の組（ d 次元のベクトル）を対応させます。人にたとえるなら、出席番号の代わりに「身長、好きな教科、部活、……」を並べたプロフィール表を使うようなものです。プロフィールなら「似ている人どうし」を数で比べられます。

具体的には、 V 行 d 列の表 E （埋め込み行列, embedding）を用意し、その i 行目をトークン i のベクトルとします。計算としては、表から i 行目を取り出すだけです²。表 E の中身も、ほかの重みと同じく学習で決まります。人がプロフィールの項目を決めるのではなく、学習の結果として「役に立つ特徴」が自然にできあがります。

Llama 3 8B では $V = 128,256$ 、 $d = 4096$ なので、 E に入っている数は

$$128,256 \times 4096 = 525,336,576 \quad (\text{約}5.3\text{億個})$$

です。「LLMのしくみ入門」で見た「意味の近い語はベクトルの向きが近い」という性質（分散表現）は、この E の行ベクトル（と、それを変換していく各層の表現）のことで

²行列の積の形で $\mathbf{e}_i E$ （ $\mathbf{e}_i$ は i 番目の成分だけが1で残りが0の横ベクトル）と書かれることがあります。図1の書き方です（付録A.2）。

3.2 ベクトルを確率にする：出力層

Transformer の最後の層からは、各位置について d 次元のベクトル $\mathbf{h}$ が出てきます。これは「ここまでの文脈を読んだ結果」をまとめたベクトルです。ここから、語彙の V 個のトークンのそれぞれについて「次に来る」ことへの点数を計算します。

トークン j ごとに d 次元の重みベクトル $\mathbf{w}_j$ を用意し、

$$z_j = \mathbf{h} \cdot \mathbf{w}_j \quad (j = 1, 2, \dots, V)$$

と、内積で点数を出します³。内積は「2つのベクトルの向きがどれだけそろっているか」を表すので、文脈 $\mathbf{h}$ と「よく合う」トークンほど点数が高くなります。この点数 z_j をロジット (logit) といいます。

点数はマイナスにもなるので、そのままでは確率になりません。そこでソフトマックス関数を通して

$$p(\text{次のトークンが } j) = \frac{e^{z_j}}{e^{z_1} + e^{z_2} + \dots + e^{z_V}}$$

とします。 e^z は必ず正なので、どの確率も正で、合計はちょうど1になります。

つまり出力層は、語彙の V 個の中から「次の1つ」を選ぶ分類問題を解いています。「教師あり学習と教師なし学習」で見た多クラスロジスティック回帰とまったく同じ形で、クラスの数 V が12.8万もある、というだけの違いです。

Llama 3 8B の出力層の重みも $4096 \times 128,256$ で約5.3億個あり、埋め込みと合わせると約10.5億個、モデル全体（約80億個）の約13%を占めます。なお、モデルによっては出力層の $\mathbf{w}_j$ を埋め込みの j 行目と同じものにして、パラメータを節約します（GPT-2 など）。

学習で点数はどう動くか 学習では、正解のトークンに与えた確率 p_c を大きくするように、損失 $L = -\log p_c$ （交差エントロピー）を小さくします。このとき、各点数 z_j をどちら向きにどれだけ動かせばよいか（勾配）は、とても簡単な形になります⁴。

$$(\text{点数 } z_j \text{ の勾配}) = (\text{予測した確率 } p_j) - (\text{正解なら } 1, \text{ そうでなければ } 0)$$

勾配は「予測－正解」です。正解のトークンの点数は上がる向きに（ $p_c - 1 < 0$ なので）、それ以外のトークンの点数は、予測した確率に比例した分だけ下がる向きに動かされます。「教師あり学習と教師なし学習」のロジスティック回帰の勾配 $p - y$ と同じ形で、ここから誤差逆伝播法で Transformer の内部へ誤差が送られていきます。

3.3 確率を計算することと、確率に従って選ぶこと

3.2節の計算（内積とソフトマックス関数）には、乱数は一切使われていません。同じ入力なら何度計算しても同じ確率が出ます。ここで得られるのは、いわば「サイコロの各目の出やすさの表」です。

乱数が入るのは、この確率から次のトークンを実際に1つ選ぶ段階です。これはくじ引き（ルーレット）にあたります。たとえば4.1節の bigram モデルで「は」の次を選ぶ場合、確率と、それを順に足した累積確率は次のとおりです。

³ V 本の $\mathbf{w}_j$ を列として並べた d 行 V 列の行列を W_{out} とすると、 V 個の点数をまとめて $\mathbf{z} = \mathbf{h}W_{\text{out}}$ と書けます。図1の書き方です（付録A.2）。

⁴導き方は付録A.3。偏微分の記号で書くと $\partial L / \partial z_j = p_j - [j = c]$ です。

次の語	本	手紙	花子	太郎
確率	1/3	1/3	1/6	1/6
累積	0.333	0.667	0.833	1.000

0以上1未満の数直線を、確率の幅に区切って各語に割り当てたルーレットを考えます。ここに一様乱数 u を1つ投げ、累積確率が初めて u 以上になる語を選びます。 $u = 0.72$ なら、 $0.667 < 0.72 \leq 0.833$ なので「花子」です。各語に割り当てた区間の幅が、ちょうどその語の確率になっています。4.2節の生成例は、付録Bのプログラムの `random.choices` がこの方法で選んだものです。

- 1回ごとにどの語が選ばれるかは、乱数 u しだいですが（ u が決まれば、選ばれる語は一通りに決まります）。
- 何度も繰り返したときに各語が選ばれる割合は、確率で決まります。ただし実際の回数はばらつきます。たとえば60回選ぶと、「本」の回数の期待値は20回ですが、標準偏差は $\sqrt{60 \times \frac{1}{3} \times \frac{2}{3}} \approx 3.7$ 回あります（数学Bの二項分布）。回数を増やすほど、割合は確率に近づきます（大数の法則）。

どの語がどれくらい出やすいか（確率）は、重みと入力で決まる決定論的な計算の結果で、実際にどの語が出るかは、そのつどのくじ引きしだいですが、同じ質問に違う答えが返るのはこのためです。確率の偏り方を調節する「温度」については、別ノート「LLMの計算で確率が入る段階と『温度』」を参照してください。

4 数えるだけの言語モデル：bigram モデル

LLM と比べるために、最も単純な言語モデルを作ってみます。

4.1 直前の1語だけを見る

本来、次の語の確率はそれまでのすべての語に依存します。そこで思い切って、直前の1語だけを見ることにします。

$$p(x_t | x_1, \dots, x_{t-1}) \approx p(x_t | x_{t-1})$$

これを **bigram モデル**（2つ組のモデル）といいます。直前に聞いた一言しか覚えていない人が、その一言だけを手がかりに次の言葉を言うようなものです。

確率は、学習用の文章の中で数えた回数の比で決めます。

$$p(b | a) = \frac{N(a, b)}{N(a)} = \frac{\text{「}a \text{の次に} b \text{」が現れた回数}}{a \text{が（前の語として）現れた回数}}$$

例題 次の6つの文を学習データとします（付録Bのプログラム）。単語の区切りは与えられているものとし、文頭に印 <s> を付けます。

太郎は本を読む。	花子は本を買う。
太郎は花子に本を渡す。	花子は手紙を書く。
太郎は手紙を読む。	花子は太郎に手紙を渡す。

語彙は「。」を含めて $V = 12$ 語で、現れた2語の組は22種類です。たとえば「は」は6回現れ、その次には「本」が2回、「手紙」が2回、「花子」と「太郎」が1回ずつ来るので、

$$p(\text{本} | \text{は}) = p(\text{手紙} | \text{は}) = \frac{2}{6} = 0.333, \quad p(\text{花子} | \text{は}) = p(\text{太郎} | \text{は}) = \frac{1}{6} = 0.167$$

です。全体の確率の表を図3に示します。 $12 \times 12 = 144$ マスのうち、0でないのは22マスだけです。

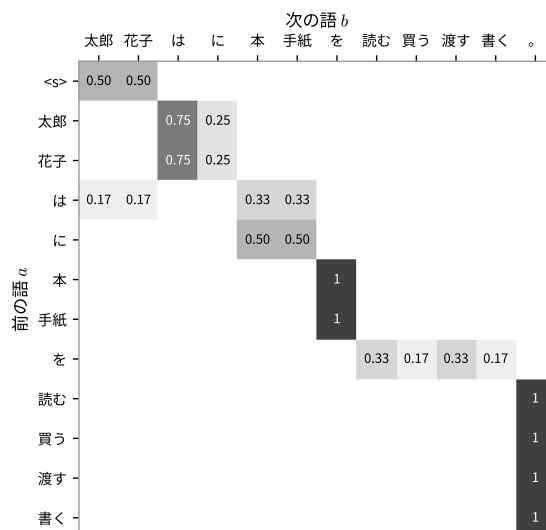


図3 例題の bigram モデルの確率 $p(b | a)$ 。行が前の語、列が次の語で、各行の和は1。空白のマスは確率0。

4.2 文章を生成する

この確率に従って、<s> から始めて次の語を1つずつくじ引きで選んでいくと、たとえば次のような文が生成されました。

太郎は花子は手紙を読む。
 太郎は太郎は太郎は手紙を渡す。
 花子は手紙を渡す。

「花子は手紙を渡す。」は学習データにない、文法的にも正しい文です。数えただけのモデルが、新しい文を作れたのです。

一方、「太郎は太郎は……」は、どの2語の組も学習データに現れた正しい組なのに、文全体としては不自然です。「は」の次に来る語を選ぶとき、モデルは直前の「は」しか見ておらず、すでに「太郎は」と言ったことを覚えていないからです。

bigram モデルは、近くの言葉のつながりは正しいが、少し離れると話がおかしくなるモデルです。文章らしさには、もっと長い文脈を覚えておく必要があります。

4.3 数えることは「最もありそうな」答えである

「は」が6回現れて、その次が「本」だったのが2回なら、 $p(\text{本} | \text{は}) = 2/6$ と見積もるのは自然に思えます。実はこれは思いつきではなく、ちゃんとした根拠があります。

頻度の比 $N(a, b)/N(a)$ は、学習データ全体が実際に起こる確率（尤度）を最大にする確率、言いかえると交差エントロピー（負の対数尤度）を最小にする確率になっています。これを最尤推定といいます。

簡単な場合で確かめてみます。「は」の次が「本」である確率を q とし、簡単のため次の語は「本」か「それ以外」の2通りとします。6回中2回が「本」、4回が「それ以外」だったので、このデータが起こる確率は

$$f(q) = q^2(1 - q)^4$$

です。 q を変えて $f(q)$ を最大にすると（数学IIの微分で、 $f'(q) = q(1 - q)^3(2 - 6q) = 0$ より）、 $q = 2/6 = 1/3$ のとき最大になります。数えて割った値が、データを最もよく説明する確率なのです。語の種類が多い一般の場合も同じ結論になります（付録A.4、ラグランジュの未定乗数法）。

ニューラルネットワーク流に学習してみる 同じことを、ニューラルネットワークと同じやり方でも確かめます。確率を直接数える代わりに、前の語 a ごとに V 個の点数（ロジット） z_{ab} を用意し、

$$p(b | a) = \frac{e^{z_{ab}}}{(\text{前の語 } a \text{ の行の } e^z \text{ の合計})}$$

とします。これは、第3節の「埋め込み」と「出力層」だけで Transformer を持たない、いちばん小さな「ニューラル言語モデル」にあたります。

すべての点数を0（どの語も同じ確率）から始めて、交差エントロピーを勾配降下法で少しずつ小さくしていきます（付録B）。音量のつまみを少しずつ回して、ちょうどよい音を探すような作業です。3.2節と同じ計算で、点数 z_{ab} の勾配は

$$N(a) \times p(b | a) - N(a, b)$$

となります（付録A.5）。これが0になるのは $p(b | a) = N(a, b)/N(a)$ のときで、確かに頻度の比に一致します。

反復回数	$p(\text{本} \text{は})$	$p(\text{花子} \text{は})$	$p(\text{読む} \text{は})$	平均損失
0	0.083	0.083	0.083	2.485
10	0.262	0.109	0.032	0.873
100	0.326	0.159	0.004	0.669
1000	0.333	0.166	0.000	0.652
3000	0.333	0.166	0.000	0.651
数えた頻度	0.333	0.167	0	—

（平均損失は1語あたりの交差エントロピー）

最初の損失 2.485 は $\log 12$ 、つまり12語が同じ確率の状態です。勾配降下法は、数えれば一度で得られる答えに、少しずつ近づいていきます（図4）。

LLM の学習も、この「交差エントロピーを勾配降下法で小さくする」という点ではまったく同じです。違うのは、確率を表すモデルが $V \times V$ の表ではなく、文脈全体を見る Transformer であることです。

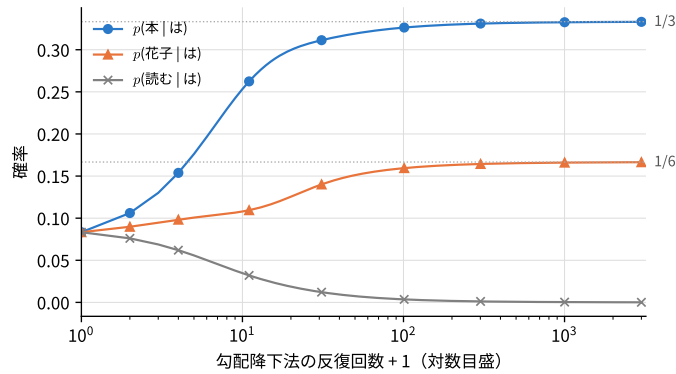


図4 勾配降下法による bigram モデルの学習．確率は数えた頻度（点線）に近づいていく．「手紙」は「本」と、「太郎」は「花子」とまったく同じ曲線になるので省略した．

4.4 n-gram モデルの限界

直前の $n-1$ 語を見るモデル（**n-gram モデル**）に広げれば、「太郎は太郎は」のような失敗は減らせます．しかし次の問題があります．

1. **表が爆発的に大きくなる**：前の $n-1$ 語の組み合わせは V^{n-1} 通りあり，表のマス数は V^n です． $V = 128,256$ なら，bigram でも $V^2 \approx 1.6 \times 10^{10}$ ，trigram（3つ組）では $V^3 \approx 2.1 \times 10^{15}$ マスになります．
2. **ほとんどのマスが0になる**：例題でも144マス中122マスが0でした． n を大きくするほど，学習データに一度も現れない組み合わせばかりになります．
3. **似た語どうして知識を共有できない**：例題の「太郎」と「花子」はどちらも人名ですが，表の上では無関係な別の行です．「太郎は」の後の統計が「花子は」の予測に役立つことはありません．

3つ目は，ちょうど出席番号の話と同じです．表の上では，太郎と花子は「別の番号の人」でしかなく，「どちらも人の名前だ」という共通点が見えないのです．

ニューラル言語モデルは，これらを次のように解決しました．

- ・ 語を埋め込みベクトル（プロファイル）で表すので，**似た語は似たベクトルになり，統計を共有できる**（3と2の緩和）．
- ・ 文脈を表で引くのではなく，Attention で必要な位置を参照して計算するので，**数千トークン以上の文脈を，表を作らずに扱える**（1の解決）．Llama 3 8B のパラメータ約80億個は，bigram の表（約160億マス）よりも少ないのです．

5 言語モデルの評価：パープレキシティ

5.1 定義

言語モデルの良さは，学習に使っていない文章に，どれだけ高い確率を与えるかで測ります．その尺度が**パープレキシティ**（perplexity, PPL，「困惑度」の意味）です．

$$\text{PPL} = \exp \left(-\frac{1}{T} \sum_{t=1}^T \log p(x_t | x_1, \dots, x_{t-1}) \right)$$

$\exp(x)$ は e^x のことです。かっこの中身は1トークンあたりの交差エントロピー、つまり学習で小さくしている損失そのものです。

パープレキシティは、モデルが各位置で、実質的に何択のクイズで迷っているかと読めます。たとえば、 K 個の候補をどれも確率 $1/K$ で予測するモデルなら、 $\text{PPL} = K$ (K 択) です。

実際、すべての位置で正解に確率 $1/K$ を与えたとすると、 $-\log(1/K) = \log K$ なので、

$$\text{PPL} = \exp(\log K) = K$$

です。一般には、パープレキシティは「正解に与えた確率の平均（幾何平均）」の逆数になっています（付録A.6）。正解に高い確率を与えるほど、パープレキシティは小さくなり、迷いが少ないことを表します。

5.2 例題

4.1節の bigram モデルで、いくつかの文のパープレキシティを計算しました（付録B）。

文	学習データに	PPL	平滑化 ($k = 0.1$) 後
花子は本を読む。	ない（2語の組はすべて既知）	1.698	2.134
花子は手紙を買う。	ない（2語の組はすべて既知）	1.906	2.488
本は花子に渡す。	ない（未知の組を含む）	∞	12.879
（学習データ全体）	ある	1.916	—
（参考）12語が等確率	—	12	—

学習データにない文でも、2語の組がすべて既知なら小さなパープレキシティ（2択より少ない迷い）になります。

しかし「本は花子に渡す。」には、文頭に「本」が来る組（ $\langle s \rangle \rightarrow$ 本）など、学習データにない組が含まれます。その確率が0なので、文全体の確率が0（掛け算の中に1つでも0があると全体が0）、パープレキシティが無限大になります。一度も見たことがない組は「絶対に起こらない」と決めつけてしまうのです。

これを避けるために、すべての組に小さな回数 k を足しておく方法（加算平滑化）があります。

$$p(b | a) = \frac{N(a, b) + k}{N(a) + kV}$$

分母に kV を足すのは、 V 個の語すべてに k ずつ足したので、合計を1に保つためです。

$k = 0.1$ とすると、「本は花子に渡す。」のパープレキシティは12.879になりました。12語が等確率の場合（12択）よりも悪い値で、このモデルにとってこの文は「でたらめに当てるより予想外」であることを表しています。

ニューラル言語モデルでは、ソフトマックス関数の出力が必ず正 ($e^z > 0$) なので、確率がちょうど0になることはありません。見たことのない組み合わせにも、小さな確率が自然に残ります。

5.3 LLM の損失とパープレキシティ

「LLMのしくみ入門」で、Llama 3 の語彙で完全に当てずっぽうなら損失は $\log 128,256 \approx 11.76$ だと述べました。これはパープレキシティ約12.8万、つまり語彙全体から迷っている状態です。損失が2まで下がればパープレキシティは $e^2 \approx 7.4$ 、1.5なら約4.5です。

12.8万択のクイズを、実質4〜7択くらいまで絞り込めるようになることが、学習の中身です。

6 まとめ

1. 言語モデルは文章に確率を与えるモデルで、文章の確率は「次のトークンの確率」の掛け算に正確に分解できる。LLMはこの各段階の確率を計算する。
2. 文章は BPE などサブワード（トークン）に分けられる。BPE は、よく隣り合う記号の組を1つにまとめていく単純な方法で、知らない単語も知っている部品に分割できる。
3. トークン番号は埋め込みの表 E の行を取り出すことでベクトルになり、最後の層のベクトルは出力層（内積）とソフトマックス関数で次のトークンの確率になる。これは V クラスの分類問題で、勾配は「予測 - 正解」の形をしている。
4. 確率の計算そのものは決定論的で、乱数が入るのは確率に従って1つ選ぶくじ引きの段階だけである。
5. 直前の1語だけを見て数える bigram モデルでも文章は生成できる。数えて割ることは最尤推定そのもので、勾配降下法でも同じ答えに近づく。
6. n -gram モデルは表が爆発的に大きくなり、似た語どうしで統計を共有できない。ニューラル言語モデルは埋め込みと Attention でこれを解決した。
7. 言語モデルはパープレキシティ（交差エントロピーの指数）で評価する。これは「実質的に何択で迷っているか」を表す。

次回予告 第2回では、このモデルを何兆トークンもの文章で学習させる**事前学習**を扱います。損失は、モデルの大きさ・データの量・計算量に対して驚くほど規則的に（べき乗則で）下がっていきます。

付録A 数式による説明

この付録では、行列とベクトル、偏微分（大学初年次の微積分・線形代数）を使います。

記号 トークンのベクトルは**横ベクトル**（1行の行列）として扱い、 $\mathbf{x}W$ のように右から行列を掛けます。 $\mathbf{e}_i$ は i 番目の成分だけが1で残りが0の横ベクトル（**one-hot ベクトル**）です。 $x_{<t}$ は $x_1, \dots, x_{t-1}$ をまとめて表します。 $[]$ は掛け算の記号、 $[j=c]$ は $j=c$ なら1、そうでなければ0を表します。右肩の T は転置です。

A.1 確率の乗法定理（連鎖律）

2つの事象について $p(A, B) = p(A)p(B|A)$ です。これを繰り返し使うと、3つなら

$$p(A, B, C) = p(A, B)p(C|A, B) = p(A)p(B|A)p(C|A, B)$$

となり、一般に

$$p(x_1, x_2, \dots, x_T) = \prod_{t=1}^T p(x_t | x_{<t})$$

が得られます ($t = 1$ の因子は条件なしの $p(x_1)$ と読みます)。これは T についての数学的帰納法で示せる恒等式で、近似は含みません。

A.2 埋め込みと出力層の行列表記

埋め込み行列 E は $V \times d$ で、one-hot ベクトル $\mathbf{e}_i$ ($1 \times V$) を使うと

$$\mathbf{e}_i E = (E \text{ の } i \text{ 行目})$$

です。出力層は、 $d \times V$ の行列 W_{out} (その第 j 列が本文の $\mathbf{w}_j$) を使って

$$\mathbf{z} = \mathbf{h} W_{\text{out}} \quad (1 \times V), \quad p(x_{t+1} = j | x_{\leq t}) = \frac{e^{z_j}}{\sum_{k=1}^V e^{z_k}}$$

です。埋め込みと出力層の重みを共有するモデル (GPT-2 など) では $W_{\text{out}} = E^T$ とします。

A.3 出力層の勾配

正解のトークンを c とすると、損失は

$$L = -\log p_c = -z_c + \log \sum_k e^{z_k}$$

です。これをロジット z_j で偏微分すると、第1項からは $-[j = c]$ 、第2項からは $e^{z_j} / \sum_k e^{z_k} = p_j$ が出るので、

$$\frac{\partial L}{\partial z_j} = p_j - [j = c], \quad \text{ベクトルで書けば} \quad \frac{\partial L}{\partial \mathbf{z}} = \mathbf{p} - \mathbf{e}_c$$

つまり「予測した確率 - 正解の one-hot ベクトル」です。2クラスのリジスティック回帰の勾配 $p - y$ と同じ形です。

A.4 頻度の比が最尤推定になること

学習データ全体の負の対数尤度は

$$L = - \sum_{(a,b)} N(a,b) \log p(b | a)$$

です。前の語 a ごとに、条件 $\sum_b p(b | a) = 1$ のもとでこれを最小にします。 $p_b = p(b | a)$ と書き、ラグランジュの未定乗数法で

$$F = - \sum_b N(a,b) \log p_b + \lambda \left(\sum_b p_b - 1 \right)$$

を p_b で偏微分して0とおくと、 $-N(a,b)/p_b + \lambda = 0$ 、つまり $p_b = N(a,b)/\lambda$ です。 $\sum_b p_b = 1$ から $\lambda = \sum_b N(a,b) = N(a)$ となり、

$$p(b | a) = \frac{N(a, b)}{N(a)}$$

が得られます。 $-\log$ は下に凸な関数なので、これは最小値です。本文4.3節の2択の場合 ($q^2(1-q)^4$ の最大化) は、この特別な場合です。

A.5 ロジットの表を勾配降下法で学習するときの勾配

前の語 a のロジット z_{ab} で $p(b | a) = e^{z_{ab}} / \sum_{b'} e^{z_{ab'}}$ とします。前の語が a である $N(a)$ 回の各回について、A.3節の結果から $\partial(-\log p_c) / \partial z_{ab} = p(b | a) - [b = c]$ です。これを $N(a)$ 回分足すと、 $[b = c]$ の和は正解が b だった回数 $N(a, b)$ になるので

$$\frac{\partial L}{\partial z_{ab}} = N(a) p(b | a) - N(a, b)$$

です。付録Bのプログラムでは、これを $N(a)$ で割ったものに学習率 $\eta = 1$ を掛けて、 z_{ab} から引いています。

A.6 パープレキシティと幾何平均

各位置で正解に与えた確率を $p_1, \dots, p_T$ とすると、

$$\text{PPL} = \exp\left(-\frac{1}{T} \sum_t \log p_t\right) = \left(\prod_{t=1}^T p_t\right)^{-1/T}$$

で、これは「正解に与えた確率の幾何平均」の逆数です。すべての位置で $p_t = 1/K$ なら $\text{PPL} = K$ です。

対数の底を2にすれば、交差エントロピーは「1トークンあたり何ビットの情報が足りないか」を表します。予測と情報量（圧縮）の関係は第2回で扱います。

付録B Python コード（標準ライブラリのみ）

外部ライブラリは使わず、Python 3 の標準機能だけで書いてあります。

B.1 BPE（第2節）

```
# バイト対符号化（BPE）でトークンの語彙を作る（標準ライブラリのみ）
from collections import Counter
```

```
# 単語とその出現回数。語末に「_」を付けて単語の区切りを表す
corpus = {"low": 5, "lower": 2, "newest": 6, "widest": 3}
words = {tuple(w) + ("_",): n for w, n in corpus.items()}
```

```
def n_tokens(words):
    return sum(len(w) * n for w, n in words.items())
```

```
vocab = sorted({c for w in words for c in w})
merges = [] # 学習した結合規則（順番が大事）
print("初期の語彙:", " ".join(vocab), f" ({len(vocab)}種類)")
print("総トークン数:", n_tokens(words))
```

```

for step in range(1, 11):
    pairs = Counter() # 隣り合う2つの記号の組を数える
    for w, n in words.items():
        for a, b in zip(w, w[1:]):
            pairs[a, b] += n
    (a, b), cnt = max(pairs.items(), key=lambda kv: kv[1])
    new = {} # 最も多い組を1つの記号に結合する
    for w, n in words.items():
        out, i = [], 0
        while i < len(w):
            if i + 1 < len(w) and w[i] == a and w[i + 1] == b:
                out.append(a + b); i += 2
            else:
                out.append(w[i]); i += 1
        new[tuple(out)] = n
    words = new
    vocab.append(a + b)
    merges.append((a, b))
    print(f"{step:2d}: {a} + {b} → {a+b:8s} ({cnt:2d}回) "
          f"語彙 {len(vocab):2d} 総トークン数 {n_tokens(words):3d}")

print("最終的な分割:")
for w, n in words.items():
    print(f" {' '.join(w):20s} ×{n}")

def encode(word):
    """学習した結合規則を、学習した順に当てはめて単語を分割する"""
    w = list(word) + ["_"]
    for a, b in merges:
        i = 0
        while i < len(w) - 1:
            if w[i] == a and w[i + 1] == b:
                w[i:i + 2] = [a + b]
            else:
                i += 1
    return w

for word in ("lowest", "newer", "wider"): # 学習データにない単語
    print(f"{word:7s} → {' '.join(encode(word))}")

```

実行結果：

```

初期の語彙: _ d e i l n o r s t w (11種類)
総トークン数: 95
1: e + s → es      ( 9回) 語彙 12 総トークン数 86
2: es + t → est    ( 9回) 語彙 13 総トークン数 77
3: est + _ → est_  ( 9回) 語彙 14 総トークン数 68
4: l + o → lo      ( 7回) 語彙 15 総トークン数 61
5: lo + w → low    ( 7回) 語彙 16 総トークン数 54
6: n + e → ne      ( 6回) 語彙 17 総トークン数 48
7: ne + w → new    ( 6回) 語彙 18 総トークン数 42
8: new + est_ → newest_ ( 6回) 語彙 19 総トークン数 36
9: low + _ → low_  ( 5回) 語彙 20 総トークン数 31
10: w + i → wi     ( 3回) 語彙 21 総トークン数 28
最終的な分割:

```

```

low_          ×5
low e r _     ×2
newest_       ×6
wi d est_     ×3
lowest  → low est_
newer   → new e r _
wider   → wi d e r _

```

B.2 bigram モデルとパープレキシティ（第4・5節）

数えるだけの言語モデル (bigram モデル) とパープレキシティ (標準ライブラリのみ)

```

import math, random
from collections import Counter, defaultdict

train = ["太郎 は 本 を 読む 。",
         "花子 は 本 を 買う 。",
         "太郎 は 花子 に 本 を 渡す 。",
         "花子 は 手紙 を 書く 。",
         "太郎 は 手紙 を 読む 。",
         "花子 は 太郎 に 手紙 を 渡す 。"]
sents = [("<s>") + s.split() for s in train] # <s> は文頭の印
vocab = sorted({w for s in sents for w in s})
V = len(vocab) - 1 # 予測の対象になる語の数 (<s> を除く)

# (1) 学習＝隣り合う2語の組を数える
count = defaultdict(Counter)
for s in sents:
    for a, b in zip(s, s[1:]):
        count[a][b] += 1

def p(b, a, k=0.0):
    """前の語 a のあとに b が来る確率 (k>0 なら加算平滑化) """
    return (count[a][b] + k) / (sum(count[a].values()) + k * V)

print(f"語彙 {V} 語, 学習した2語の組 {sum(len(c) for c in count.values())} 種類")
for a in ("は", "を"):
    row = ", ".join(f"{b} {p(b, a):.3f}" for b, _ in count[a].most_common())
    print(f"p(・ | {a}) : {row}")

# (2) 文章の生成：確率に従って次の語を1つずつ選ぶ
random.seed(3)
for _ in range(4):
    w, out = "<s>", []
    while w != ". ":
        nxt = list(count[w])
        w = random.choices(nxt, weights=[count[w][b] for b in nxt])[0]
        out.append(w)
    print("生成:", "".join(out))

# (3) パープレキシティ：exp(1語あたりの平均の交差エントロピー)
def ppl(sentence, k=0.0):
    s = ["<s>"] + sentence.split()
    nll = 0.0
    for a, b in zip(s, s[1:]):
        q = p(b, a, k)

```

```

        if q == 0:
            return math.inf
        nll -= math.log(q)
    return math.exp(nll / (len(s) - 1))

tests = ["花子 は 本 を 読む 。",      # 学習データにない文（組はすべて既知）
        "花子 は 手紙 を 買う 。",
        "本 は 花子 に 渡す 。"]      # 未知の組 (<s>→本 など) を含む
for t in tests:
    print(f"{''.join(t.split()):12s} PPL = {ppl(t):7.3f} 平滑化 (k=0.1) {ppl(t, 0.1):7.3f}")
print("学習データ全体の PPL =",
      round(math.exp(sum(-math.log(p(b, a)) for s in sents for a, b in zip(s, s[1:]))
              / sum(len(s) - 1 for s in sents)), 3))
print("一様分布（どの語も 1/V）の PPL =", V)

```

実行結果：

```

語彙 12 語，学習した2語の組 22 種類
p(・|は)：本 0.333，手紙 0.333，花子 0.167，太郎 0.167
p(・|を)：読む 0.333，渡す 0.333，買う 0.167，書く 0.167
生成：太郎は花子は手紙を読む。
生成：太郎は太郎は太郎は手紙を渡す。
生成：花子は手紙を渡す。
生成：太郎は太郎は手紙を渡す。
花子は本を読む。      PPL = 1.698 平滑化 (k=0.1) 2.134
花子は手紙をかう。     PPL = 1.906 平滑化 (k=0.1) 2.488
本は花子に渡す。      PPL = inf 平滑化 (k=0.1) 12.879
学習データ全体の PPL = 1.916
一様分布（どの語も 1/V）の PPL = 12

```

B.3 bigram モデルを勾配降下法で学習する（4.3節）

```

# bigram モデルを「ニューラルネット流」に学習する：
# ロジットの表 Z を勾配降下法で調整すると，確率は数えた頻度に近づく
import math
from collections import Counter, defaultdict

train = ["太郎 は 本 を 読む 。", "花子 は 本 を 買う 。",
        "太郎 は 花子 に 本 を 渡す 。", "花子 は 手紙 を 書く 。",
        "太郎 は 手紙 を 読む 。", "花子 は 太郎 に 手紙 を 渡す 。"]
sents = [{"<s>"} + s.split() for s in train]
prev = sorted({a for s in sents for a in s[:-1]}) # 前の語になりうる語
nxt = sorted({b for s in sents for b in s[1:]})   # 予測の対象 (V 語)
count = defaultdict(Counter)
for s in sents:
    for a, b in zip(s, s[1:]):
        count[a][b] += 1

def softmax(z):
    m = max(z); e = [math.exp(v - m) for v in z]; t = sum(e)
    return [v / t for v in e]

Z = {a: [0.0] * len(nxt) for a in prev} # 初期値：すべて0（一様分布）
eta = 1.0
watch = ["本", "手紙", "花子", "太郎", "読む"]
for step in range(3001):

```

```

if step in (0, 10, 100, 1000, 3000):
    P = softmax(Z["は"])
    L = -sum(n * math.log(softmax(Z[a])[nxt.index(b)]))
        for a in prev for b, n in count[a].items())
    N = sum(sum(c.values()) for c in count.values())
    print(f"{step:5d}回 平均損失 {L/N:.4f} p(・|は): "
          + " ".join(f"{w} {P[nxt.index(w)]:.3f}" for w in watch))
for a in prev: # 勾配  $\partial L / \partial z_{ab} = N_a p(b|a) - N_{ab}$ 
    P = softmax(Z[a]); Na = sum(count[a].values())
    for j, b in enumerate(nxt):
        Z[a][j] -= eta * (Na * P[j] - count[a][b]) / Na
print("数えた頻度 p(・|は): " + " ".join(
    f"{w} {count['は'][w] / sum(count['は'].values()):.3f}" for w in watch))

```

実行結果：

```

0回 平均損失 2.4849 p(・|は): 本 0.083 手紙 0.083 花子 0.083 太郎 0.083 読む 0.083
10回 平均損失 0.8726 p(・|は): 本 0.262 手紙 0.262 花子 0.109 太郎 0.109 読む 0.032
100回 平均損失 0.6687 p(・|は): 本 0.326 手紙 0.326 花子 0.159 太郎 0.159 読む 0.004
1000回 平均損失 0.6518 p(・|は): 本 0.333 手紙 0.333 花子 0.166 太郎 0.166 読む 0.000
3000回 平均損失 0.6506 p(・|は): 本 0.333 手紙 0.333 花子 0.166 太郎 0.166 読む 0.000
数えた頻度 p(・|は): 本 0.333 手紙 0.333 花子 0.167 太郎 0.167 読む 0.000

```