

LLMの基礎（2）事前学習とスケーリング則

（高校生・大学初年次向け）

大きさ・データ・計算量と損失の関係

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

第1回では、言語モデルが「次のトークンの確率」を出す装置であり、予想の外れ具合（交差エントロピー）を勾配降下法で小さくすることで学習することを見ました。第2回では、これを何兆トークンもの文章で行う事前学習を扱います。

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

中心になるのは、モデルの大きさ・データの量・計算量を増やすと、損失がべき乗則に従って規則正しく下がるという経験則（スケーリング則）です。この規則性があるので、巨大なモデルを作る前に、小さなモデルの実験からその性能を予測できます。飛行機を作る前に、模型を風洞で試して本物の性能を見積もるのに似ています。

読み方 本文は、高校の数学（指数関数と対数、常用対数と両対数グラフ、微分、最小二乗法の考え方）を使って読めるように書きました。行列を使った計算量の見積もり、情報量と符号の長さの関係、最適な配分の導出などの**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

- N ：モデルのパラメータ（重み）の数， D ：学習に使うトークン数， C ：学習の計算量（小数を含む数の掛け算・足し算の回数．単位を FLOP と書きます）。
- L ：損失（1トークンあたりの交差エントロピー，自然対数）．第1回5.1節のとおり， e^L がパープレキシティ（実質何択で迷っているか）です。
- α, β ：べき乗則の指数（正の定数）． E, A, B もデータへの当てはめで決まる定数です。
- $y \propto x^{-\alpha}$ は「 y が $x^{-\alpha}$ に比例する」ことを表します。
- 10^{23} のような大きな数を $1\text{e}23$ とも書きます（付録Bのプログラムの書き方）． $\log_{10}$ は常用対数です。

1 事前学習

1.1 何をするのか

事前学習の中身は、第1回と同じく「文章の各位置で次のトークンを予測し、交差エントロピー

$$L = -\frac{1}{T} \sum_{t=1}^T \log p(x_t | x_1, \dots, x_{t-1})$$

を小さくする」ことです。正解（次のトークン）は文章そのものから得られるので、人が正解のラベルを付ける必要はありません（自己教師あり学習、「教師あり学習と教師なし学習」参照）。

文章を1語ずつ隠して「次は何か」を当てる穴埋め問題は、どんな文章からでも自動的に作れます。そのため、Web 上の文章、書籍、論文、プログラムのソースコードなど、手に入る限りの大量の文章を問題集として使えます。

規模の例を挙げます。

モデル	発表	パラメータ数 N	学習トークン数 D
Chinchilla	2022年（DeepMind）	700億	1.4兆
Llama 3 8B	2024年（Meta）	80億	15兆以上

15兆トークンは、日本語の文庫本（1冊10万字程度）に換算すると、ざっと1億冊を超える分量です¹。人が一生かけても読めない量の文章を読んで、穴埋めの練習をするわけです。

学習の方法は「深層学習（2）」と同じ、ミニバッチによる確率的勾配降下法（の改良版）です。ただし文章の量が膨大なので、同じ文章を何度も繰り返し使うことは少なく、多くの文章はほぼ一度だけ読まれます。

1.2 「次を当てる」だけで何が身につくのか

次のトークンを当てるという課題は単純ですが、うまく当てるためには多くのことを知っている必要があります。

- ・「私は昨日、友達と映画を__」→ 文法（動詞の形）
- ・「水は1気圧のもとで、約__℃で沸騰する」→ 事実の知識
- ・「 $17 + 25 = __$ 」→ 計算
- ・「AはBより背が高い。BはCより背が高い。したがって、最も背が高いのは__」→ 推論
- ・プログラムの続き → プログラミング言語の文法と、処理の意図

学習データにはこうした文章がすべて混ざっているので、損失を下げるためには、これらをまとめて扱える内部の表現を作るしかありません。穴埋め問題の点数を上げようと全教科の問題集を解き続けた結果、文法も知識も計算も少しずつ身についていく、というイメージです。「LLMのしくみ入門」で見た「知識がパラメータ全体に分散して表現される」ことは、この圧力の結果です。

¹ トークンと文字の対応は言語やトークナイザで違うので、これはおおよその目安です。

1.3 予測することは圧縮することである

ここで、少し意外な見方を紹介します。よく予測できるモデルは、文章を短く圧縮できるので

す。
モールス信号を思い出してください。英語で最もよく使われる文字 E は「・」1つ、T は「ー」1つと短く、めったに使われない Q は「ーー・ー」と長くなっています。よく出てくるものに短い符号を、まれなものに長い符号を割り当てると、全体として文章を短く送れます。

情報理論では、確率 p で起こる出来事を伝えるのに必要な符号の長さは $-\log_2 p$ ビットだとわかっています（シャノン）。たとえば確率 $1/2$ なら1ビット、 $1/8$ なら3ビットです。言語モデルの確率を使って文章を符号にすると、1トークンあたりの符号の長さは、交差エントロピーを底2の対数で測ったものになります²。

つまり、損失を下げることは文章をより短く圧縮できるようになることです。よく当てるモデルほど、文章の中の規則性（文法・事実・論理）をよく捉えていて、そのぶん文章を短く表せます。

逆に、どんなに優れたモデルでも、文章そのものに含まれる本当の不確かさ（次に何を書くかが書き手の自由である部分）は圧縮できません。サイコロの目は、どんなに賢くても予想できないのと同じです。これが3.2節の「減らせない損失」 E にあたります。

2 学習の計算量： $C \approx 6ND$

スケーリング則を扱う前に、学習にどれだけの計算が必要かを見積もります。結論は

$$C \approx 6ND$$

学習の計算量は、パラメータ数と学習トークン数の積に比例し、係数は約6です。

理由はおおよそ次のとおりです³。

- ・ **順伝播**（予測を計算する）：1つのトークンを処理するとき、重み1個につき「入力に掛ける」掛け算1回と「結果を足し合わせる」足し算1回、合わせて約2回の計算をします。モデル全体で約 $2N$ 回です。
- ・ **逆伝播**（誤差を送り返す）：各層で「前の層へ送る誤差」と「重みの勾配」の2種類を計算するので、順伝播の約2倍、 $4N$ 回です。

合わせて1トークンあたり $6N$ 回、 D トークンで $6ND$ 回になります。

例

- ・ Chinchilla： $C = 6 \times (7 \times 10^{10}) \times (1.4 \times 10^{12}) \approx 5.9 \times 10^{23}$ FLOP
- ・ Llama 3 8B： $C = 6 \times (8 \times 10^9) \times (1.5 \times 10^{13}) = 7.2 \times 10^{23}$ FLOP

仮に、毎秒 4×10^{14} 回の計算を実際にこなす GPU を1000台使ったとすると、それぞれ約17日、約21日かかる計算量です（付録B）。GPU 1台なら、それぞれ約47年、約57年かかる計算です⁴。

² L は自然対数で測っているので、ビットに直すには $\log 2$ で割ります。 $L/\log 2 \approx 1.443L$ ビットです。詳しくは付録A.1。

³行列の積を使ったくわしい説明は付録A.2。Attention の中には重みを使わない計算もありますが、文脈がモデルの次元に比べて極端に長くない限り小さいので無視しています。

⁴ $17日 \times 1000 = 17000日 \approx 47年$ 、 $21日 \times 1000 = 21000日 \approx 57年$ 。

3 スケーリング則

3.1 ベキ乗則

Kaplan ら (OpenAI, 2020年) は、大きさの違う多数のモデルを学習させ、**損失がパラメータ数・データ量・計算量のそれぞれに対して、ベキ乗則に従って下がる**ことを報告しました。たとえばデータが十分にあるとき、パラメータ数 N (埋め込み行列を除く) に対して

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad \alpha_N \approx 0.076$$

という関係が、何けたにもわたる N の範囲で成り立っていました (N_c は定数)。

両辺の常用対数をとると

$$\log_{10} L = \alpha_N \log_{10} N_c - \alpha_N \log_{10} N$$

となり、横軸に $\log_{10} N$ 、縦軸に $\log_{10} L$ をとった**両対数グラフ**で、傾き $-\alpha_N$ の直線になります。

ベキ乗則は、「 N を何倍かにすると、 L が何倍になるか」が、 N の大きさによらず一定という法則です。この例では

$$\frac{L(10N)}{L(N)} = 10^{-0.076} \approx 0.84$$

つまり、**パラメータを10倍にするたびに、損失が約16%ずつ下がります**。100億パラメータのモデルを1000億にしても、1000億を1兆にしても、下がる割合は同じです。

一般に $L(N) = cN^{-\alpha}$ なら、どんな倍率 k についても

$$\frac{L(kN)}{L(N)} = \frac{c(kN)^{-\alpha}}{cN^{-\alpha}} = k^{-\alpha}$$

で、右辺に N が入っていません。どの大きさから見ても同じ形に見えるので、ベキ乗則は**スケール不変** (scale-free) と呼ばれます。地図の縮尺を変えても同じ形に見える海岸線のようなものです。

指数関数 e^{-N/N_0} なら N_0 という「特別な大きさ」があり、それを過ぎると急に0に近づきます。ベキ乗則には、そのような「ここで終わり」という大きさはありません。

自然や社会には、ベキ乗則に従う現象が数多く知られています。地球科学でおなじみの**グーテンベルク・リヒターの法則** (地震の規模と回数の関係：マグニチュードが1大きい地震は、回数がおよそ10分の1になる) や、文章中の単語の出現頻度と順位の関係 (**ジップの法則**) などです。LLM の損失も、その仲間に加わったことになります。

3.2 Chinchilla の式：大きさとデータの両方を考える

Hoffmann ら (DeepMind, 2022年) は、パラメータ数 N と学習トークン数 D の両方を変えた400以上のモデルを学習させ、損失を次の形で表しました。

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$

当てはめで得られた値は $E = 1.69$, $A = 406.4$, $B = 410.7$, $\alpha = 0.34$, $\beta = 0.28$ です。3つの項は次の意味をもちます。

- E ：減らせない損失. 文章そのものの不確かさで、どんなに大きなモデルを無限のデータで学習しても残ります（1.3節）.
- A/N^α ：モデルが有限の大きさであるための損失. N を大きくすると0に近づきます.
- B/D^β ：データが有限であるための損失. D を大きくすると0に近づきます.

生徒（モデル）が教科書（データ）で勉強する場面にと考えると、 E は「そもそも誰にも予想できない問題」の分、 A/N^α は「生徒の頭の容量が足りない」分、 B/D^β は「教科書が薄すぎる」分の失点です. 優秀な生徒でも薄い教科書では限界があり、分厚い教科書でも生徒の容量には限界があります.

図1は、この式で D を固定して N を変えたときの損失です. データが少ない ($D = 10^9$) と、モデルをいくら大きくしても損失はデータの項で止まります. 逆に、データがいくらあっても ($D \rightarrow \infty$, 図の黒い線), 損失は $E + A/N^\alpha$ より下がりません.

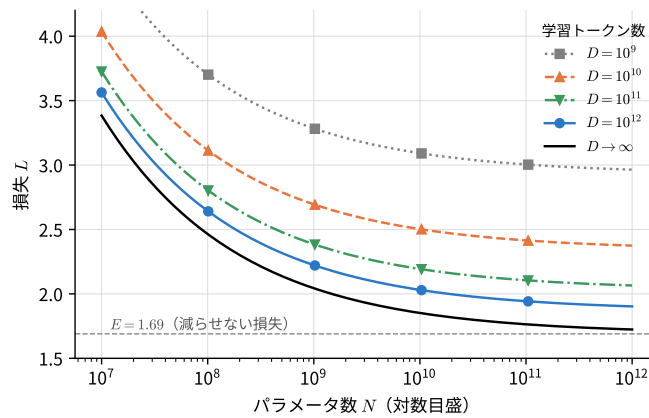


図1 Chinchilla の式による損失. 学習トークン数 D ごとに、パラメータ数 N を変えたときの値. 破線は減らせない損失 E .

なお、損失の値そのものは、学習データの種類やトークナイザによって変わるので、別のモデルの損失と直接比べることはできません. ここでの数値は、この式が表す関係の形を見るためのものです.

たとえば Llama 3 8B の大きさ ($N = 8 \times 10^9$) について、この式で内訳を計算すると次のようになります (付録B).

D	E	A/N^α	B/D^β	L
2×10^{11} (2000億)	1.69	0.175	0.281	2.146
1.5×10^{13} (15兆)	1.69	0.175	0.084	1.949
∞	1.69	0.175	0	1.865

データを75倍に増やすと、データの項は0.281から0.084に下がりますが、モデルの大きさの項0.175はまったく変わりません. 教科書をいくら厚くしても、頭の容量の分の失点は減らないのです.

3.3 計算量を決めたときの最適な配分

使える計算量 C が決まっているとき、 $C = 6ND$ なので、 N と D の積が一定になります. 面積が一定の長方形で、縦を長くすれば横は短くなるのと同じで、「大きなモデルを少ないデー

タで学習する」か、「小さなモデルを多くのデータで学習する」かの選択になります。

C を固定して N を変えたときの損失を描くと、**谷の形**になり、損失が最も小さくなる N があります（図2）。谷の左側ではモデルが小さすぎ（頭の容量不足），右側ではデータが少なすぎ（教科書不足）です。

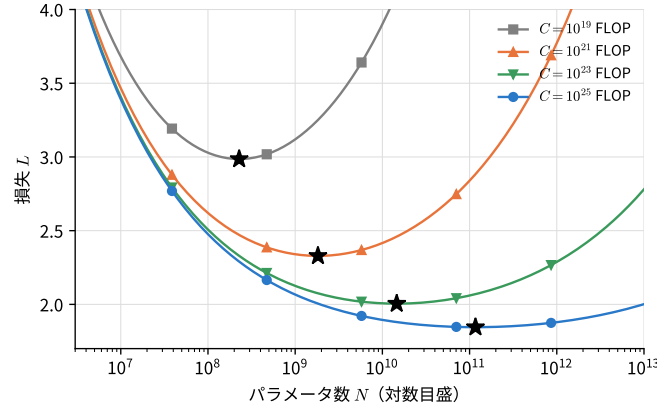


図2 計算量 C を固定したとき ($D = C/6N$) の損失とパラメータ数の関係. 星印が各 C での最小点.

谷の底は、損失の式に $D = C/(6N)$ を代入して N で微分し、0とおけば求められます（付録A.3）。結果は

$$N_{\text{opt}} \propto C^{\beta/(\alpha+\beta)}, \quad D_{\text{opt}} \propto C^{\alpha/(\alpha+\beta)}$$

で、2つの指数の和は1です。 α と β が近ければ、どちらの指数も $\frac{1}{2}$ に近くなります。

つまり、計算量を増やすときは、**パラメータ数と学習トークン数をほぼ同じ割合で増やすのが最適**です。計算量を100倍にするなら、モデルもデータもそれぞれ約10倍にする、という具合です。

Hoffmann らの結論は、「パラメータ数と学習トークン数を同じ割合で増やすべきで、**1パラメータあたり約20トークンが目安**」というものでした。彼らはこの考えに従って、当時の大きなモデル Gopher（2800億パラメータ，3000億トークン）と同じ計算量で、4分の1の大きさの Chinchilla（700億パラメータ）を4倍以上のデータ（1.4兆トークン）で学習させ、Gopher を上回る性能を得ました。

ただし、上の式の係数をそのまま使って最適配分を計算すると、1パラメータあたりのトークン数は20よりかなり大きくなります。2024年に Besiroglu らが論文の図からデータを読み取って当てはめ直したところ、 $E = 1.82$, $A = 482.01$, $B = 2085.43$, $\alpha = 0.3478$, $\beta = 0.3658$ となり、こちらは約20トークンという目安と整合しました（付録B）。

C (FLOP)	H : N_{opt}	H : D/N	B : N_{opt}	B : D_{opt}	B : D/N
10^{19}	2.3×10^8	32	2.6×10^8	6.4×10^9	24
10^{21}	1.8×10^9	50	2.8×10^9	6.0×10^{10}	22
5.88×10^{23}	3.3×10^{10}	93	7.3×10^{10}	1.3×10^{12}	18
10^{25}	1.2×10^{11}	122	3.1×10^{11}	5.3×10^{12}	17

(H : Hoffmann らの係数, B : Besiroglu らの係数による最適配分)

Besiroglu らの係数では、Chinchilla の計算量 5.88×10^{23} に対する最適配分は約730億パラメータ・1.3兆トークンとなり、実際の Chinchilla（700億・1.4兆）とよく一致します。同

じ実験でも、データの扱い方によって当てはめた係数はこの程度は変わりうる、ということです。

3.4 「最適」より多く学習させる理由

Llama 3 8B は、Chinchilla の目安（80億パラメータなら約2000億トークン）の約75倍、15兆トークンで学習されています。1パラメータあたり約1900トークンです。Meta は、目安をはるかに超えても性能が上がり続けた（損失が学習トークン数の対数に対してほぼ直線的に下がり続けた）と報告しています。

これは3.3節の「最適」が、**学習の計算量だけを考えたときの最適**だからです。完成したモデルは何億回も使われるので、使うときの計算量（1トークンあたり約 $2N$ ）も考える必要があります。

毎日何百万人も乗る車なら、作るのに手間がかかっても、燃費のよい小型車にしたほうが全体として安くつきます。同じように、**小さなモデルを長く学習させておくほうが、使う段階まで含めると安くつくことがあります**。スケーリング則は、何を最適化したいかに応じて使い分ける道具です。

4 小さな実験から大きなモデルを予測する

スケーリング則の実用上の価値は、**小さなモデルの実験から、まだ作っていない大きなモデルの損失を予測できる**ことにあります。これを最小二乗法で試してみます（付録B）。

例題 「真の法則」を $L = 1.69 + 406.4/N^{0.34}$ （3.2節で $D \rightarrow \infty$ としたもの）とし、 $N = 10^7$ から 10^{10} までの7種類のモデルで損失を「測定」したとします。測定値には0.5%程度のばらつきを乱数で加えました。

N	10^7	3.2×10^7	10^8	3.2×10^8	10^9	3.2×10^9	10^{10}
L	3.406	2.856	2.465	2.205	2.033	1.930	1.842

(a) E を引いて両対数で直線を当てはめる $L - E = A/N^\alpha$ の両辺の常用対数をとると $\log_{10}(L - E) = \log_{10} A - \alpha \log_{10} N$ なので、 $x = \log_{10} N$, $y = \log_{10}(L - E)$ とし、直線 $y = wx + c$ を最小二乗法で当てはめます（「教師あり学習と教師なし学習」の $w = S_{xy}/S_{xx}$ ）。結果は $\alpha = -w = 0.349$ （真の値0.34）、 $A = 478.1$ で、図3(a)のようにきれいな直線に乗ります。

(b) E を引かずに当てはめる $\log_{10} L$ をそのまま直線で当てはめると、見かけの指数は0.087になります。

それぞれの式で、測定していない $N = 10^{11}$, 10^{12} の損失を予測すると次のようになりました。

N	真の L	予測 (a)	予測 (b)
10^{11}	1.764	1.760	1.413
10^{12}	1.724	1.721	1.155

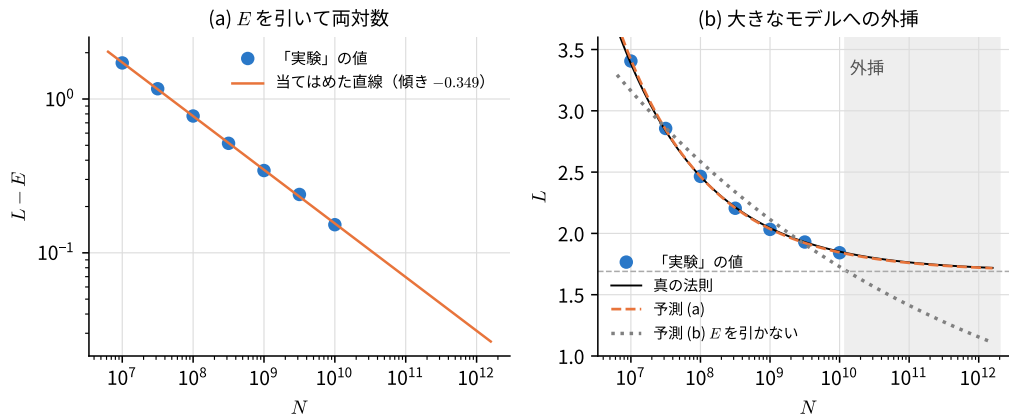


図3 (a) 減らせない損失 E を引いた値は両対数グラフで直線になる. (b) 測定範囲（白地）から外挿範囲（灰色）への予測. E を引かない直線 (b) は, E より下まで下がり続けると誤って予測する.

予測 (a) は真の値とほぼ一致します. 予測 (b) は, 測定範囲では大きくずれていないのに, 外挿すると損失が E を下回るまで下がり続けると予測してしまいます (図3(b)).

どんな式の形を仮定するか（ここでは, 減らせない損失があるかどうか）が, 外挿の結果を大きく左右します. 測定した範囲でよく合うことは, その外でも正しいことを保証しません.

実際に, 大きなモデルを学習させる前に, 計算量が最大で1万分の1程度の小さなモデル群でスケーリング則を当てはめ, 完成後の損失を予測したという報告もあります (OpenAI の GPT-4 の技術報告, 2023年). まさに「模型で本物を予測する」やり方です.

5 スケーリング則が示すこと

5.1 収穫は減っていくが, 止まらない

べき乗則では, 資源を増やすほど損失は下がりますが, 同じだけ下げるのに必要な資源は倍々が増えていきます. Chinchilla の式で, モデルの大きさによる損失 $A/N^{0.34}$ を半分にするには N を $2^{1/0.34} \approx 7.7$ 倍に, 10分の1にするには約870倍にする必要があります⁵.

テストで50点から70点に上げるのは比較的やさしくても, 90点から95点に上げるのはずっと大変なのと似ています. ただし, べき乗則では資源を同じ倍率で増やすたびに, 同じ割合ずつ確実に下がり続け, 頭打ちになる「壁」がありません (E に近づくまでは). LLM の開発が, 計算資源の巨大な投資の競争になっている背景がここにあります.

5.2 能力の「創発」

損失は滑らかに下がりますが, 個々の課題 (たとえば3けたの足し算の正答率) で測ると, ある大きさまではほぼ0で, それを超えると急に解けるようになるように見える場合があります. これを能力の創発 (emergence) と呼ぶことがあります (Wei ら, 2022年).

⁵ $A/(kN)^{0.34} = \frac{1}{2} \cdot A/N^{0.34}$ より $k^{0.34} = 2$, $k = 2^{1/0.34}$ です. 10分の1なら $k = 10^{1/0.34} \approx 870$.

これに対して、「完全に正解したときだけ1点」のような評価の仕方をしているために、滑らかな改善が急な変化に見えているだけで、という指摘もあります (Schaeffer ら, 2023 年)。

部分点なしの採点を考えてください。 3けたの答えの各けたを正しく出す確率 p が滑らかに上がっていても、3けたすべてが正しい確率は p^3 です。たとえば $p = 0.2$ なら 0.008, $p = 0.5$ なら 0.125, $p = 0.9$ なら 0.729 で、はじめはほぼ0のまま、ある所から急に上がって見えます。何をどう測るかによって、同じ変化が連続的にも不連続的にも見えるということです。

6 まとめ

1. 事前学習は、Web などの膨大な文章で次のトークンの予測を学ぶ自己教師あり学習である。うまく予測するには文法・知識・推論が必要で、損失を下げることは文章を圧縮できるようになることと同じである。
2. 学習の計算量は $C \approx 6ND$ (パラメータ数 $\times$ トークン数 $\times 6$) と見積もれる。
3. 損失はパラメータ数・データ量・計算量に対してべき乗則に従って下がる。べき乗則は両対数グラフで直線になり、スケール不変である。
4. Chinchilla の式 $L = E + A/N^\alpha + B/D^\beta$ は、減らせない損失・モデルの大きさによる損失・データ量による損失の和である。計算量を固定すると、 N と D をほぼ同じ割合で増やすのが最適になる (1パラメータあたり約20トークン)。
5. 実際には、使うときの計算量も考えて、小さなモデルをより多くのデータで学習させることが多い。
6. スケーリング則を使うと、小さな実験から大きなモデルの損失を予測できる。ただし予測は仮定する式の形に左右される。

次回予告 事前学習を終えたモデルは、「文章の続きを書く」ことはできても、質問に答えたり指示に従ったりするには作られていません。第3回では、それを対話モデルにする**指示調整と人間のフィードバックによる学習**、そして重みを変えずに振る舞いが変わる**文脈内学習**を扱います。

付録A 数式による説明

この付録では、行列とベクトル、実数乗の関数の微分 (数学III～大学初年次) を使います。

記号 $x_{<t}$ は $x_1, \dots, x_{t-1}$ をまとめて表します。トークンのベクトルは横ベクトルとして扱い、 $y = \mathbf{x}W$ のように右から行列を掛けます。右肩の T は転置、 δ は誤差逆伝播法で後ろから送られてくる誤差を表します。

A.1 予測することは圧縮することである

確率 p で起こる事象を伝えるのに必要な情報量は $-\log_2 p$ ビットです (シャノン)。よく起こる事象は短い符号で、まれな事象は長い符号で表すのが最も効率のよい符号化で、**算術符号化**という方法を使うと、実際にほぼこの長さで符号化できます。

したがって、言語モデルの確率を使って文章を符号化すると、1トークンあたりの符号の長さは

$$-\frac{1}{T} \sum_t \log_2 p(x_t | x_{<t}) \text{ ビット}$$

となります。底の変換公式 $\log_2 p = \log p / \log 2$ により、これは自然対数で測った交差エントロピー L を $\log 2$ で割ったもの

$$\frac{L}{\log 2} \approx 1.443 L \text{ ビット}$$

そのものです。つまり、損失を下げることは文章をより短く圧縮できるようになることです。文章に含まれる本当の不確かさ（真の分布のエントロピー）より短くは圧縮できないことも、シャノンの情報源符号化定理として知られています。これが減らせない損失 E に対応します。

A.2 なぜ $C \approx 6ND$ なのか

Transformer の計算の大部分は行列の積 $\mathbf{y} = \mathbf{x}W$ です。 W が $d \times d'$ なら、1トークンについて dd' 回の掛け算と、ほぼ同じ回数足し算が必要です。つまり重み1個につき、1トークンあたり約2回の演算で、モデル全体では順伝播に約 $2N$ 回の演算がかかります。

逆伝播では、各層で

- ・ 前の層へ送る誤差 δW^T
- ・ 重みの勾配 $\mathbf{x}^T \delta$

の2種類の行列積を計算するので（「深層学習（2）」の誤差逆伝播法）、順伝播の約2倍、 $4N$ 回です。合わせて1トークンあたり $6N$ 回、 D トークンで $6ND$ 回になります。

Attention の QK^T のように重みを使わない計算もありますが、文脈の長さがモデルの次元に比べて極端に長くない限り小さいので無視しています。

A.3 計算量を決めたときの最適な配分の導出

$D = C/(6N)$ を代入すると

$$L = E + AN^{-\alpha} + B \left(\frac{6N}{C} \right)^\beta$$

です。 N で微分して0とおくと

$$-\alpha AN^{-\alpha-1} + \beta B \left(\frac{6}{C} \right)^\beta N^{\beta-1} = 0 \quad \Rightarrow \quad N^{\alpha+\beta} = \frac{\alpha A}{\beta B} \left(\frac{C}{6} \right)^\beta$$

なので、

$$N_{\text{opt}} = G \left(\frac{C}{6} \right)^{\frac{\beta}{\alpha+\beta}}, \quad D_{\text{opt}} = \frac{1}{G} \left(\frac{C}{6} \right)^{\frac{\alpha}{\alpha+\beta}}, \quad G = \left(\frac{\alpha A}{\beta B} \right)^{\frac{1}{\alpha+\beta}}$$

です。2つの指数の和は1なので、計算量を増やすときは、それを N と D に $\beta : \alpha$ の割合で（対数の上で）振り分けることになります。 α と β が近ければ、 N と D をほぼ同じ割合で増やすのが最適です。

L の2階微分 $\alpha(\alpha+1)AN^{-\alpha-2} + \beta(\beta-1)B(6/C)^\beta N^{\beta-2}$ は、これだけでは符号が決まりませんが、 $\log N$ を変数として見ると L は2つの指数関数 $e^{-\alpha \log N}$ と $e^{\beta \log N}$ の正の定数倍の和（に

定数を加えたもの) で下に凸なので、求めた点は最小点です。これは「 $C = 6ND$ という条件のもとで L を最小にする」問題なので、ラグランジュの未定乗数法で解いても同じ結果になります。

Hoffmann らの係数では $\beta/(\alpha+\beta) = 0.452$, Besiroglu らの係数では 0.513 です (付録B)。

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 計算量と Chinchilla の式 (第2・3節)

```
# 計算量  $C \approx 6ND$  と, Chinchilla の式による損失・最適配分の計算 (標準ライブラリのみ)

# (1) 学習の計算量  $C = 6ND$  ( $N$ : パラメータ数,  $D$ : 学習トークン数)
for name, N, D in [("Chinchilla", 70e9, 1.4e12), ("Llama 3 8B", 8e9, 15e12)]:
    C = 6 * N * D
    days = C / 4e14 / 1000 / 86400 # 仮に  $4 \times 10^{14}$  FLOP/s の GPU を 1000 台使うと
    print(f"{name:11s} N={N:.0e} D={D:.1e} C={C:.2e} FLOP (GPU 1000台で約 {days:.0f} 日)")

# (2) 損失の式  $L(N, D) = E + A/N^\alpha + B/D^\beta$ 
fits = {"Hoffmann ら (2022)": dict(E=1.69, A=406.4, B=410.7, a=0.34, b=0.28),
        "Besiroglu ら (2024)": dict(E=1.82, A=482.01, B=2085.43, a=0.3478, b=0.3658)}

def loss(N, D, E, A, B, a, b):
    return E + A / N**a + B / D**b

def optimum(C, E, A, B, a, b):
    """ $C = 6ND$  の制約のもとで  $L$  を最小にする  $N, D$  (付録A.4の解) """
    G = (a * A / (b * B)) ** (1 / (a + b))
    N = G * (C / 6) ** (b / (a + b))
    return N, C / (6 * N)

for fname, f in fits.items():
    print(f"\n--- {fname} の係数 ---")
    print(f"N の指数  $\beta/(\alpha+\beta) = \{f['b']/(f['a']+f['b']):.3f\}$ , "
          f"D の指数  $\alpha/(\alpha+\beta) = \{f['a']/(f['a']+f['b']):.3f\}")
    for C in (1e19, 1e21, 5.88e23, 1e25):
        N, D = optimum(C, **f)
        print(f"C={C:.2e}: 最適 N={N:.2e} D={D:.2e} D/N={D/N:6.1f} L={loss(N, D, **f):.3f}")

f = fits["Hoffmann ら (2022)"]
print("\n--- Hoffmann らの係数で計算した損失 ---")
for N, D in [(70e9, 1.4e12), (8e9, 2e11), (8e9, 15e12), (8e9, float("inf"))]:
    print(f"N={N:.0e} D={D:.1e}: L={loss(N, D, **f):.3f} "
          f"(内訳 E={f['E']}, A/N^a={f['A']/N**f['a']:.3f}, B/D^b={f['B']/D**f['b']:.3f})")$ 
```

実行結果:

```
Chinchilla N=7e+10 D=1.4e+12 C=5.88e+23 FLOP (GPU 1000台で約 17 日)
Llama 3 8B N=8e+09 D=1.5e+13 C=7.20e+23 FLOP (GPU 1000台で約 21 日)
```

```
--- Hoffmann ら (2022) の係数 ---
N の指数  $\beta/(\alpha+\beta) = 0.452$ , D の指数  $\alpha/(\alpha+\beta) = 0.548$ 
C=1.00e+19: 最適 N=2.28e+08 D=7.31e+09 D/N= 32.1 L=2.986
```

C=1.00e+21: 最適 N=1.82e+09 D=9.14e+10 D/N= 50.1 L=2.329
 C=5.88e+23: 最適 N=3.25e+10 D=3.02e+12 D/N= 92.8 L=1.930
 C=1.00e+25: 最適 N=1.17e+11 D=1.43e+13 D/N= 122.1 L=1.845

--- Besiroglu ら (2024) の係数 ---

N の指数 $\beta/(\alpha+\beta) = 0.513$, D の指数 $\alpha/(\alpha+\beta) = 0.487$
 C=1.00e+19: 最適 N=2.62e+08 D=6.36e+09 D/N= 24.2 L=2.930
 C=1.00e+21: 最適 N=2.78e+09 D=6.00e+10 D/N= 21.6 L=2.308
 C=5.88e+23: 最適 N=7.30e+10 D=1.34e+12 D/N= 18.4 L=1.977
 C=1.00e+25: 最適 N=3.12e+11 D=5.34e+12 D/N= 17.1 L=1.915

--- Hoffmann らの係数で計算した損失 ---

N=7e+10 D=1.4e+12: L=1.937 (内訳 E=1.69, $A/N^\alpha=0.083$, $B/D^\beta=0.163$)
 N=8e+09 D=2.0e+11: L=2.146 (内訳 E=1.69, $A/N^\alpha=0.175$, $B/D^\beta=0.281$)
 N=8e+09 D=1.5e+13: L=1.949 (内訳 E=1.69, $A/N^\alpha=0.175$, $B/D^\beta=0.084$)
 N=8e+09 D=inf: L=1.865 (内訳 E=1.69, $A/N^\alpha=0.175$, $B/D^\beta=0.000$)

B.2 べき乗則の当てはめと外挿 (第4節)

小さなモデルの「実験結果」からべき乗則を最小二乗法で推定し、大きなモデルの損失を予測する

```
import math, random
```

```
E, A, alpha = 1.69, 406.4, 0.34 # 「真の」法則  $L = E + A / N^\alpha$  (データは十分にあるとする)
random.seed(1)
Ns = [10 ** (7 + 0.5 * i) for i in range(7)] #  $1e7 \sim 1e10$  の7種類のモデル
Ls = [(E + A / N**alpha) * (1 + random.gauss(0, 0.005)) for N in Ns] # 0.5% の測定のはらつき
```

```
def fit(xs, ys):
    """直線  $y = w x + c$  の最小二乗法 (「教師あり学習と教師なし学習」) """
    mx, my = sum(xs) / len(xs), sum(ys) / len(ys)
    Sxx = sum((x - mx) ** 2 for x in xs)
    Sxy = sum((x - mx) * (y - my) for x, y in zip(xs, ys))
    w = Sxy / Sxx
    return w, my - w * mx
```

```
# (a) E を引いてから両対数で直線を当てはめる:  $\log(L - E) = \log A - \alpha \log N$ 
w, c = fit([math.log10(N) for N in Ns], [math.log10(L - E) for L in Ls])
print(f"(a) E を引いた場合  $\alpha$  の推定値 = {-w:.3f} (真の値 {alpha}), A の推定値 = {10**c:.1f}")
# (b) E を引かずに  $\log L$  を直線で当てはめる
w2, c2 = fit([math.log10(N) for N in Ns], [math.log10(L) for L in Ls])
print(f"(b) E を引かない場合 見かけの指数 = {-w2:.3f}")
```

```
for N, L in zip(Ns, Ls):
    print(f" N = {N:8.1e} L = {L:.4f}")
for N in (1e11, 1e12): # 実験していない大きさへの外挿
    true = E + A / N**alpha
    pa = E + 10**c * N**w
    pb = 10**c2 * N**w2
    print(f"N = {N:.0e}: 真の L = {true:.4f} 予測 (a) = {pa:.4f} 予測 (b) = {pb:.4f}")
```

実行結果：

```
(a) E を引いた場合  $\alpha$  の推定値 = 0.349 (真の値 0.34), A の推定値 = 478.1
(b) E を引かない場合 見かけの指数 = 0.087
N = 1.0e+07 L = 3.4060
N = 3.2e+07 L = 2.8559
```

N = 1.0e+08 L = 2.4652
N = 3.2e+08 L = 2.2051
N = 1.0e+09 L = 2.0328
N = 3.2e+09 L = 1.9296
N = 1.0e+10 L = 1.8423
N = 1e+11: 真の L = 1.7640 予測 (a) = 1.7595 予測 (b) = 1.4130
N = 1e+12: 真の L = 1.7238 予測 (a) = 1.7211 予測 (b) = 1.1552