microgpt を解剖する　統合版（高校生・一般向け）

4つの名前で，AI の計算を一つずつたどり，覚えたことを読み解く

Planned by Yoshio Okamoto　Written by Claude Opus 5.5 (Oct. 25, 2026)

Ver. 0.1　　October 25, 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）．microgpt のプログラムは Andrej Karpathy の著作物です.

はじめに

ChatGPT や Claude のような AI（大規模言語モデル，Large Language Model，LLM）は，とても単純なことをくり返しています．これまでの文章の続きとして，次に来る言葉を一つ予想し，それを付け足して，また次を予想する．これだけです.

microgpt は，Andrej Karpathy が 2026年2月に公開した，約200行の Python のプログラムです．大きな AI と同じしくみを，ごく小さな規模で動かします．扱うのは文章ではなく英語の名前で，「次の文字」を当てることを学び，最後に新しい名前を作ります.

この冊子では，microgpt に **emma，ava，mia，anna の4つの名前だけ**を学ばせます（ミニマムな例）．小さくすると，AI の中で動くすべての数を，紙の上で一つずつ追いかけられるからです．本文の数は，すべて実際にプログラムを動かして確かめたものです.

この冊子は，これまでに作った『microgpt を解剖する』（第1〜5部のスライドと冊子），『小さなAIの頭の中をのぞく』，wte と lm_head の解説をまとめ直したものです．重なっていた説明を一つにし，用語と記号をそろえました.

この冊子の読み方

- **本文**は数式を読まなくても分かるように書きました．たとえの下にある「式で書くと」の帯は，同じことを1〜2行の式で書いたものです．読み飛ばしてもかまいません.
- 計算の途中では「形」の帯で，**数の並びの長さと，掛ける表の形**を示します．数の並びがどこで4個から16個になり，どこで7個になるのかを，いつでも確かめられます.
- 迷ったら，次のページの**表0（9つの表の一覧）**と**図0（形の追跡図）**に戻ってください．記号の一覧は付録B，数式のくわしい導き方は付録Aにあります.

章	内容	見るもの
1	全体像と準備	AI は何を学ぶのか．データ，文字の番号，問題
2	黒板と9つの表	268 個の重みの置き場所と，計算の舞台
3	1文字を読む	emma の1文字を，実際の数で順伝播する
4	外れ具合と学習	損失を測り，逆伝播で勾配を求め，重みを更新する
5	学習後の表を読む	268 個の重みから，AI が覚えたことを読み解く
6	名前を作る	確率でくじを引き，新しい名前を作る
7	本物の AI と比べる	大きさと計算のしかたのちがい，同じところ

- プログラム本体：https://gist.github.com/karpathy/8627fe009c40f57531cb18360106ce95
- 作者による解説（英語）：https://karpathy.github.io/2026/02/12/microgpt/

巻頭：見取り図

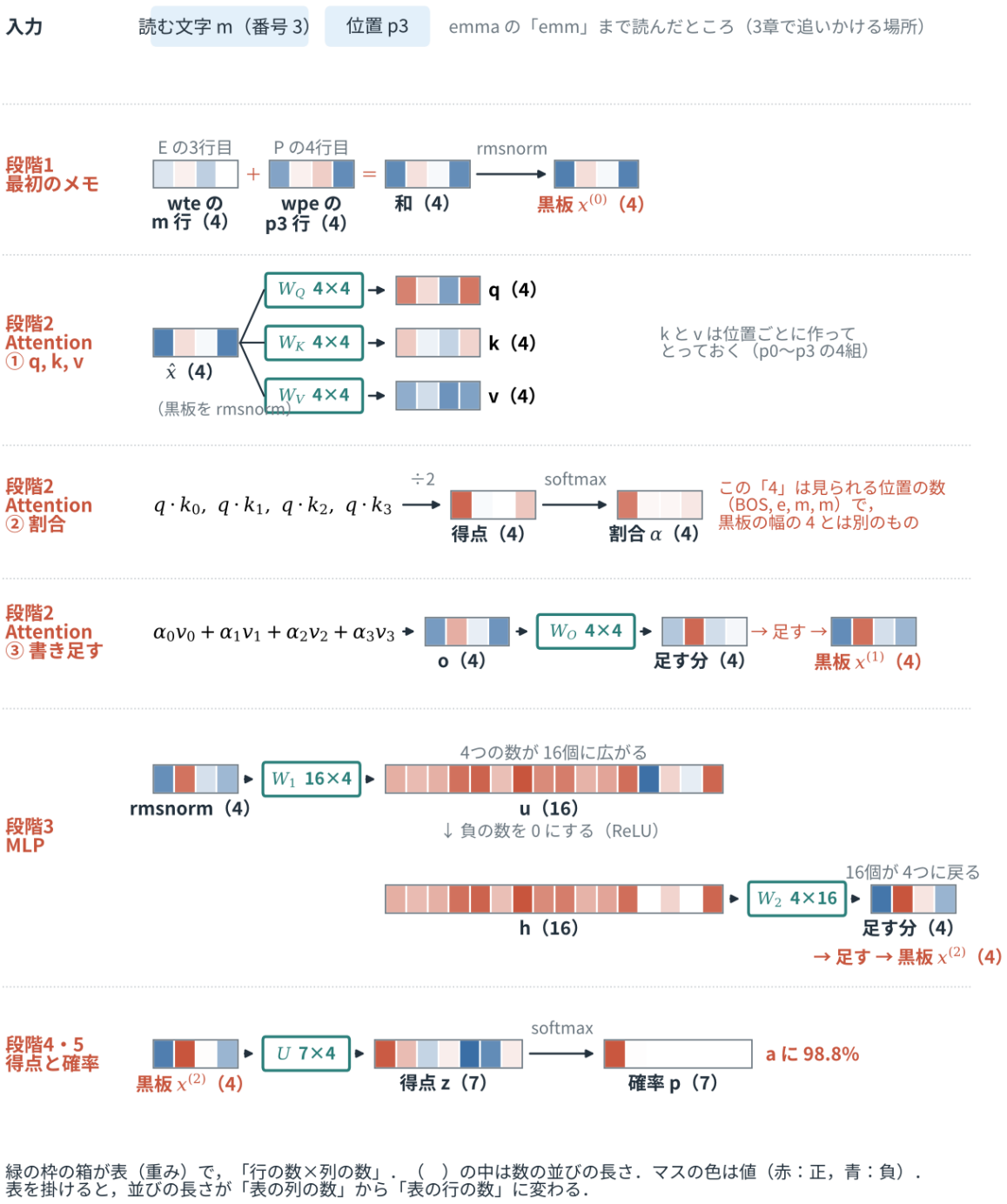
AI の中には 268 個の数（**重み**，weight）があり，9つの表（行列）に分けて並んでいます．表0はその一覧，図0は，1文字を読むときに数の並びの長さがどう変わるかを1枚にまとめたものです．どの章を読んでいても，ここに戻れば「いま何の表を，どの長さの数の並びに掛けているか」が分かります．

表0　9つの表の一覧（ミニマムな例）

記号	プログラムの名前	形（行×列）	数	使い方	入る → 出る	章
E	`wte`	7×4	28	文字の番号で1行を引く	番号 →（4）	3, 5
P	`wpe`	5×4	20	位置の番号で1行を引く	番号 →（4）	3, 5
W_Q	`attn_wq`	4×4	16	掛ける（問い合わせ q を作る）	（4）→（4）	3, 5
W_K	`attn_wk`	4×4	16	掛ける（見出し k を作る）	（4）→（4）	3, 5
W_V	`attn_wv`	4×4	16	掛ける（中身 v を作る）	（4）→（4）	3, 5
W_O	`attn_wo`	4×4	16	掛ける（写した中身を黒板の形に）	（4）→（4）	3, 5
W_1	`mlp_fc1`	16×4	64	掛ける（16個の検出器）	（4）→（16）	3, 5
W_2	`mlp_fc2`	4×16	64	掛ける（検出器ごとに書き足す）	（16）→（4）	3, 5
U	`lm_head`	7×4	28	掛ける（7つの文字の得点）	（4）→（7）	3, 5
合計			**268**			

読むときの3つの約束

1. **表の形は「行の数×列の数」**．掛ける表は，**列の数＝入る数の個数，行の数＝出る数の個数**です．W_1（16×4）は4つの数から16個を作り，W_2（4×16）は16個から4つに戻します．
2. **数の並びの長さは4種類だけ**です．4＝黒板の幅（どこでも同じ），16＝MLP の中だけ，7＝文字の種類（得点と確率），そして Attention の割合の長さ＝見られる位置の数（1〜5．読んだところで変わる）．
3. **この冊子に 16×16 の表は出てきません**．元の設定の microgpt（黒板の幅 16）では Attention の表が 16×16 になります．ほかの資料と見比べるときは，7章の表26を見てください．



緑の枠の箱が表（重み）で，「行の数×列の数」．（ ）の中は数の並びの長さ．マスの色は値（赤：正，青：負）．
表を掛けると，並びの長さが「表の列の数」から「表の行の数」に変わる．

図0　形の追跡図．emma の「emm」まで読んだ位置（p3）で，1文字を読むときの数の並びの長さと，掛ける表の形．マスの色は実際の値（3章でたどる数）．

目次

巻頭：見取り図 2

1　全体像と準備 5

2　黒板と9つの表 12

3　1文字を読む 18

4　外れ具合と学習 26

5　学習後の表を読む 36

6　名前を作る 47

7　本物の AI と比べる 53

おわりに 58

付録A　数式 60

付録B　記号と形の一覧 65

付録C　数を確かめるプログラム 66

付録D　用語のまとめ 70

付録E　数値の求め方とプログラムの入手先 71

1　全体像と準備

1.1　大きな AI も「次を当てる」をくり返すだけ

大きな AI は，これまでの文章の続きとして，次に来る言葉を一つ予想します．「むかし　むかし　ある」まで読んだら次は「ところ」，「むかし　むかし　ある　ところ」まで読んだら次は「に」…と，1つ当てたら付け足して，また次を当てます[1]．スマートフォンの「予測変換」と同じ考え方で，ちがうのは，ずっと長い文章を読んで，ずっと上手に予想できることです．

microgpt も同じことをします．ただし材料は文章ではなく名前で，予想する単位は1つの文字です（表1）．

表 1　大きな AI と microgpt

	大きな AI（LLM）	microgpt
学ぶ材料	大量の文章	英語の名前（この冊子では4つ）
1つの文書	1つの記事や本	1つの名前（emma など）
予想する単位	トークン（単語のかけら）	1つの文字
作るもの	文章の続き	新しい名前

式で書くと　AI が求めるのは，読んだところ $c_0, \ldots, c_t$ を条件とした次の文字の確率 $P(c_{t+1} \mid c_0, \ldots, c_t)$ です．c_t は t 番目の文字の番号です．

1.2　microgpt とミニマムな例

microgpt は，NumPy や PyTorch のような特別なライブラリを使わず，Python に最初から入っている機能だけで書かれています．プログラムの冒頭には「このファイルがアルゴリズムのすべてで，それ以外は効率化にすぎない」という趣旨のことが書かれています．本物の AI とのちがいは主に大きさと速さで，計算の中身は同じだ，という主張です．7章でこの主張を確かめます．

元の microgpt は約3万2千個の名前で学びますが，この冊子では4つの名前に減らした**ミニマムな例**を使います（表2）．プログラムは書きかえていません．数字を4か所変えて，学ぶ名前を4行にしただけです[2]．

[1] 大きな AI が1回に予想する単位は，単語そのものではなく，単語をさらに分けた「トークン」（token）と呼ばれるかけらです．日本語では1〜2文字くらいのことが多いです．

[2] 76行目 `n_embd` 16→4，77行目 `block_size` 16→5，78行目 `n_head` 4→1，152行目 `num_steps` 1000→300．学ぶ名前は `input.txt` というファイルに1行に1つずつ書きます．冊子『microgpt を読む』5節のミニマムな例と同じ設定です．

表 2　元の microgpt と，この冊子のミニマムな例

	元の設定	ミニマムな例
学ぶ名前の数	32,033	4（emma，ava，mia，anna）
文字の種類（＋区切りの印）	27	7
黒板の幅（1文字を表す数の個数）	16	4
一度に読める長さ	16	5
Attention のヘッドの数	4	1
学習の回数（ステップ数）	1,000	300
重みの数	**4,192**	**268**
計算の時間	2〜3分	約2秒

この冊子の数は，すべてミニマムな例のものです．元の設定の数は，7章で比べるときにだけ使います．

1.3　プログラムの6つの部分と，この冊子の章

プログラムは，上から順に6つの部分でできています（図1左，行番号は元の `microgpt.py` の200行のもの）．

表 3　プログラムの6つの部分

	部分	行	すること
1	データ	14–21	名前を読み込み，順番をシャッフルする
2	トークナイザ	23–27	文字を番号に変える規則を作る
3	自動微分	29–72	「どの重みを，どちらに動かせばよいか」を計算するしくみ
4	重みとモデル	74–144	重みを用意し，次の文字の得点を出す本体
5	学習	146–184	外れ具合を測り，重みを少しずつ動かす
6	生成	186–200	学習したモデルで新しい名前を作る

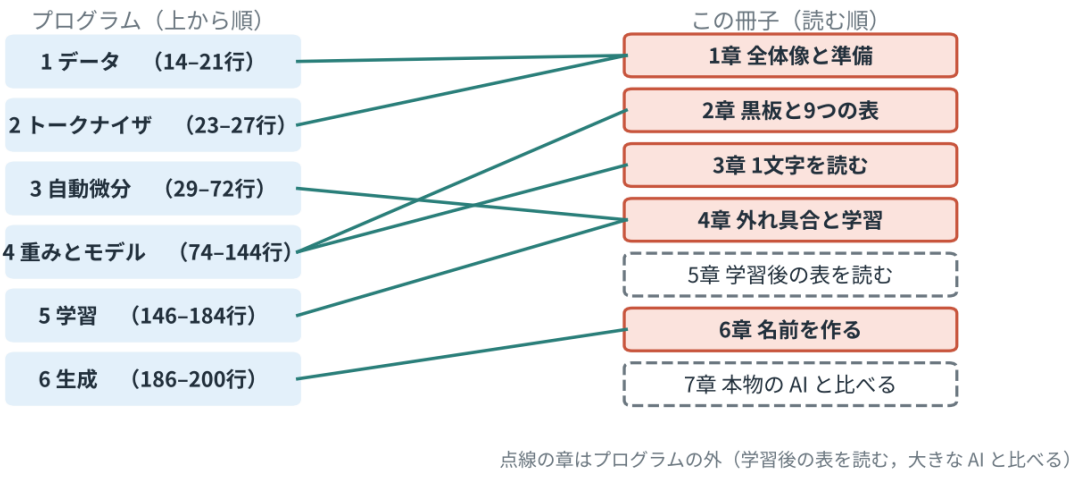


図 1　プログラムの6つの部分（左）と，この冊子の章（右）の対応．

プログラムでは「3 自動微分」が「4 重みとモデル」より先に書かれていますが，この冊子ではモデルを先に読みます（2・3章）．予想の計算が分からないと，その原因を調べる計算（4章）の意味が分からないからです．

重み（つまみ）とは　AI の中には 268 個の数があり，これを**重み**（weight）または**パラメータ**（parameter）と呼びます．ラジオの音を合わせるつまみのように，一つ一つ回して値を変えられる数だと思ってください[3]．最初はでたらめな値（小さな乱数）で，学習をくり返すたびに，当たりやすくなる向きへ少しずつ動かします．学習が終わったときの重みの値が，AI が「覚えたこと」のすべてです．268 個の重みは9つの表（行列）に分けて並んでいます（巻頭の表0，2章）．

1.4　学習と生成

同じモデルを2つの場面で使います．

学習（training）では，正解の名前を見ながら1文字ずつ次を予想し，外れ具合に応じて重みを動かします．1回の学習（1ステップ，step）は，部活の練習にたとえると分かりやすいでしょう（図2）．問題を解き（順伝播），採点し（損失），原因を調べ（逆伝播），フォームを直す（重みの更新）．1回の学習では名前を1つ使い，これを 300 回くり返します．

生成（generation）では正解はなく，始まりの印から1文字ずつ予想し，確率に従って文字を選んで付け足します．「終わり」が選ばれたら名前の完成です．重みはもう動かしません．予想の計算（順伝播）は，学習でも生成でもまったく同じです[4]．

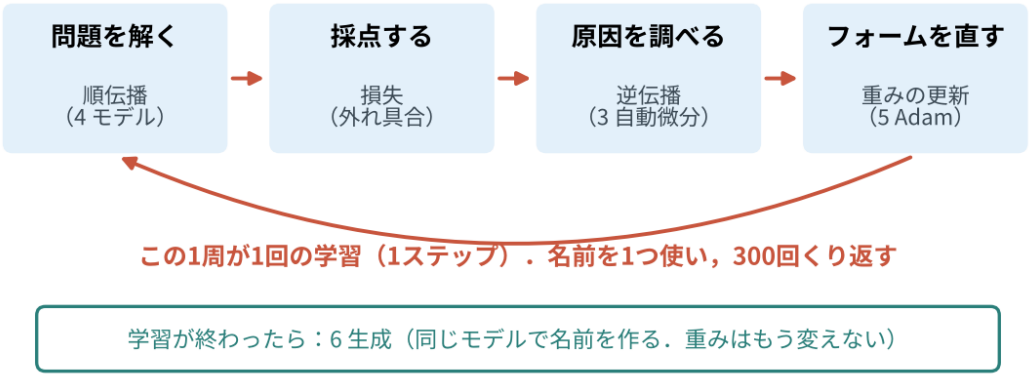


図 2　学習の1ステップを部活の練習にたとえると．番号は表3の部分の番号．

式で書くと　1ステップ：順伝播で確率 p を出す → 損失 $L = -\ln p$(正解) → 逆伝播で勾配 $\partial L/\partial w$ → 更新 $w \leftarrow w - \eta\times$（Adam で調整した勾配）．

[3]冊子『microgpt を読む』では「つまみ」と呼んでいました．この冊子では，一般の説明に合わせて「重み」と呼びます．

[4]大きな AI の説明では，生成のことを推論（inference）とも呼びます．

表 4　この冊子の用語

用語	英語	意味
学習	training	300回くり返す全体（プログラムの部分5）
1回の学習（1ステップ）	step	名前を1つ使って，順伝播から更新までを1周
順伝播（予想）	forward pass	文字を読んで，次の文字の確率を出す計算
損失（罰点）	loss	外れ具合（4章）
逆伝播（自動微分）	backpropagation（automatic differentiation）	勾配を求める計算（4章）
勾配	gradient	重みを少し増やしたときの損失の増え方
更新	update	重みを動かすこと．Adam はそのやり方（optimizer）
生成	generation	学習後に名前を作ること（6章）

1.5　学習データ：4つの名前

学習には emma，ava，mia，anna の4つの名前を使い，1つの名前を1つの「文書」として扱います．この4つは，身近さよりも「AI の中で何が起きるかが見えやすいか」を優先して選びました[5]．

- **文字の種類が少ない**：a，e，i，m，n，v の6種類だけなので，AI の中の表が小さく，全部を読めます．
- **「正しく迷う」場面がある**：始まりの次（a か e か m）と，a の次（v か n）は，データ自体が答えを一つに決めていません．
- **同じ役目の文字がある**：mia の i と ava の v は，どちらも「次は必ず a」です．AI がこれに気づくかどうかを確かめられます．
- **どれも4文字以内**：一度に読める長さは5なので，最後の「終わり」まで学習できます．

文字を共有する名前を選んだのは，2番目と3番目の場面を作るためです．ばらばらの名前（たとえば sophia，mary，celine，elena）にすると，始まりの文字以外は次が一つに決まってしまい，丸暗記しか見えなくなります．似た名前ばかりなのは意図したもので，名前としての自然さと，見えるものの多さの兼ね合いです．

シャッフル　プログラムは，読み込んだ名前の順番をまずランダムに並べかえ（シャッフル），その順に1つずつ学習に使います．乱数の種 42 では，順番は mia，ava，anna，emma になります．最後まで来たら先頭に戻るので，300回で，どの名前もちょうど75回ずつ使います[6]．

1.6　文字の種類を集めて，番号をふる

AI は文字をそのまま扱えないので，文字に番号をつけます．4つの名前に出てくる文字を集めて，重なりを除き，アルファベット順に並べ，0 から番号をふります．最後に，区切りの印

[5] emma，ava，mia は，近年のアメリカでよく使われる女の子の名前です．冊子『microgpt を読む』5節，『小さなAIの頭の中をのぞく』と同じ組です．

[6] コンピュータの乱数は「乱数の種」と呼ばれる出発点の数から計算で作られ，同じ種からは毎回同じ数の列が出ます．microgpt は種を 42 に決めている（12行目）ので，誰が動かしても1回目は mia です．種を変えると，シャッフルの順，重みの最初の値，名前を作るときのくじ引きがすべて変わります．

BOS（ビーオーエス，beginning of sequence）に次の番号 6 をふります（表5）[7].

表 5　文字と番号

文字	a	e	i	m	n	v	BOS
番号	0	1	2	3	4	5	6

a〜v は microgpt が決めた名札ではなく，学ばせた名前に実際に出てくる文字です．学ばせる名前を変えれば，文字の種類も番号も変わります．元の約3万個の名前なら a〜z の26種類＋BOS の27種類，「taro，hanako」なら a，h，k，n，o，r，t の7種類＋BOS です．名札にない文字（b や z など）は，この AI にとって存在しません．読むことも，作ることもできません．

BOS：始まりと終わりを示す1つの印　名前を学習するとき，前と後ろに BOS を付けます．前の BOS は「ここから名前が始まる」という合図で，AI は BOS を読んで名前の最初の文字を当てます．後ろの BOS は「ここで名前が終わる」という答えで，AI が BOS を予想したら名前の完成です．名簿の名前と名前の間にある区切り線のように，1本の線が上の名前の終わりと下の名前の始まりを同時に示しています．この冊子では，答えとしての BOS を「終わり」と書くことがあります．

4つの名前を番号の列にする　4つの名前は次の番号の列になります[8]．ここから先の計算は，すべてこの番号の列で行います．

$$\text{emma} \rightarrow [6,1,3,3,0,6] \qquad \text{ava} \rightarrow [6,0,5,0,6]$$
$$\text{mia} \rightarrow [6,3,2,0,6] \qquad \text{anna} \rightarrow [6,0,4,4,0,6]$$

> **AI が知っているのは番号だけ**：AI は a の形も音も知りません．すべての文字の呼び名を一度に入れかえても（a を「赤」，e を「青」...），学ぶ中身はまったく同じです．サッカーの記録に背番号しか書かれていなくても，「7番から10番にパスが出やすい」ことは記録から学べるのと同じです．文字の「意味」は，データの中での現れ方からしか生まれません．5章で見るように，i と v が AI の中で同じ場所に置かれるのは，形が似ているからではなく，どちらも「次に必ず a が来る」という同じ役目を持つからです．

1.7　問題を作る

1つの名前からは，「ここまで読んだら，次は何？」という**問題**（学習データから作る問題，training example）がいくつもできます．emma からは表6の5問です．読んだところは，いつも名前の頭からです．AI は前を見ることはできますが，先（答え）を見ることはできません．最後の BOS は答えになるだけで，それを読んで次を当てる問題はありません[9].

[7] プログラムでは `uchars = sorted(set(''.join(docs)))`（24行目）で文字を集めて並べ，`BOS = len(uchars)`（25行目）で BOS の番号を決めます．この「文字と番号を行き来する規則」をトークナイザ（tokenizer）と呼びます．

[8] プログラムでは `tokens = [BOS] + [uchars.index(ch) for ch in doc] + [BOS]`（157行目）です．

[9] 1回の学習では，1つの名前の問題を全部（emma なら5問）解いて，その平均の外れ具合で重みを動かします．

表 6　emma から作る問題

問題	位置	読んだところ	次の文字（正解）
1	p0	始まり（BOS）	e
2	p1	e	m
3	p2	em	m
4	p3	emm	a
5	p4	emma	終わり（BOS）

位置（p0，p1，...）は，AI が1文字を読んで次を当てる場所です．p は position の頭文字で，0 から数えます．p0 は BOS を読む位置，p3 は「emm」の最後の m を読む位置です．位置の数は名前の文字数＋1で，emma なら5つ，mia なら4つです．

4文字の名前は5問，3文字の名前は4問なので，4つの名前で $5+4+4+5=18$ 問です．このうち「始まり」は4つの名前すべてに，「a」は ava と anna の両方に出てくるので，同じ読んだところをまとめると **14の場面**になります．

名前の木　4つの名前を1本の木にまとめると，図3のようになります．左の BOS から右へたどると，4つの名前のどれかになります．枝が分かれる場所は2か所だけです．始まりの次と，a の次です．ほかの12の場面では，次の文字が一つに決まります．

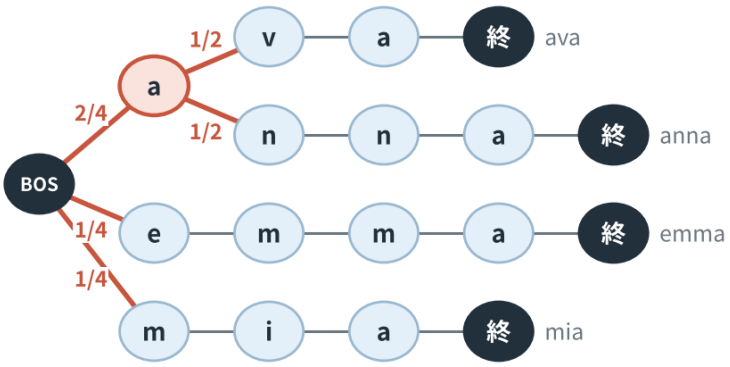


図 3　4つの名前の木．「終」は終わりの BOS．赤い数字は，データの中でその枝に進む割合．

1.8　正しく迷う：答えが一つに決まらない場面

始まりの次は，4つの名前のうち a が2つ（ava，anna），e と m が1つずつです．a の次は v（ava）と n（anna）が1つずつです（図4）．こうした場面では，どれか一つに決めるより，「a が 50%，e が 25%，m が 25%」とデータの割合どおりに答えるのが最もよい答えです[10]．くじ引きの箱に，a を2本，e と m を1本ずつ入れておくようなものです．

[10]「最もよい」とは，4章で説明する損失の平均がいちばん小さくなる，という意味です（付録A.1）．

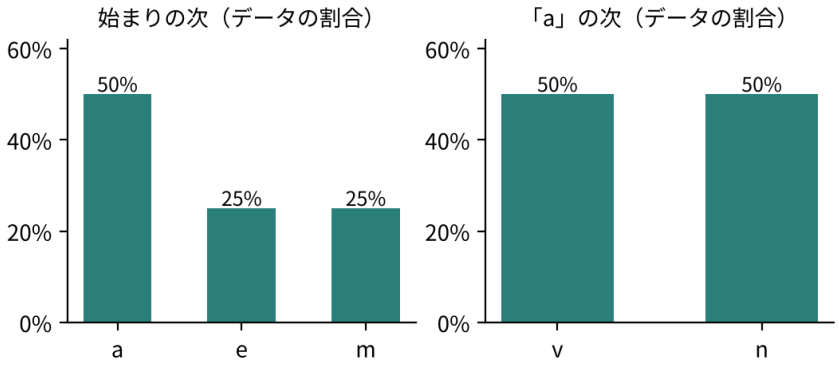


図 4　答えが分かれる2つの場面（データの割合）．

どんなに学習しても，迷いは0にならない：18問のうち12問は答えが一つに決まるので，迷いは始まりと a の次の6問だけから来ます．「平均して何択で迷っているか」で表すと，学習の前は7択（7種類から当てずっぽう），データから決まる限界は 1.36択です．AI がどんなに賢くなっても，1.36択より小さくはなりません．300回の学習の後は 1.41択で，この限界にかなり近づきます（4章）．

式で書くと　最善の答え＝データの割合．迷いの度合い（何択か）＝ $e^{\text{損失の平均}}$．限界は $e^{0.308} = 1.36$ 択（付録A.1）．

1.9　人間が前もって決める設定

料理の「何人分を作るか」「何分煮るか」のように，作り始める前に決める数があります（表7）．重みとちがい，学習しても変わりません．専門用語では**ハイパーパラメータ**（hyperparameter）と呼びます．一度に読める長さが 5 なので，5文字以上の名前は後ろが切られ，最後の「終わり」まで学習できません．4つの名前がどれも4文字以内なのはそのためです．

表 7　人間が前もって決める設定

設定	プログラムの名前	ミニマム	元	意味
黒板の幅	`n_embd`	4	16	1文字を表す数の個数（2章）
一度に読める長さ	`block_size`	5	16	始まり＋4文字まで
ヘッドの数	`n_head`	1	4	Attention を分けて行う組の数（7章）
学習の回数	`num_steps`	300	1000	各名前75回
学習率	`learning_rate`	0.01	0.01	重みを動かす大きさ（4章）
乱数の種	`random.seed`	42	42	同じ種なら同じ結果

1章のまとめ：AI は「次を当てる」をくり返しています．プログラムは6つの部分からなり，学習（順伝播 → 損失 → 逆伝播 → 更新）を300回くり返したあと，生成で名前を作ります．名前は番号の列になり，18問の問題になります．答えが分かれるのは始まりの次と a の次だけで，そこでは正しく迷うのが最善です．

2　黒板と9つの表

3章で実際の計算をたどる前に，計算の舞台を整理しておきます．268 個の重みがどこに置かれ，計算の途中の数がどこに書かれるのかです．

2.1　268 個の重み ＝ 9つの表の成分

268 個の重みは，9つの表（行列）に分けて並んでいます（巻頭の表0）．重み1個は，表の1つのマス（行列の成分 w_{ij}）です．268 は9つの表の大きさの足し算です．

$$\underbrace{28}_{\text{wte } 7\times4} + \underbrace{20}_{\text{wpe } 5\times4} + \underbrace{16\times4}_{\text{Attention } 4\times4 \text{ が4つ}} + \underbrace{64}_{\text{fc1 } 16\times4} + \underbrace{64}_{\text{fc2 } 4\times16} + \underbrace{28}_{\text{lm_head } 7\times4} = 268$$

表の大きさは，すべて次の4つの数から決まります．7＝文字の種類，5＝一度に読める長さ（位置の数），4＝黒板の幅[11]，16＝黒板の幅の4倍（MLP の検出器の数）．名前の長さは関係しません．名前が長くなっても，表は大きくなりません．

重みの約半分（fc1 と fc2 で 128 個，約 48%）は MLP にあります．「こういうときは次にこれ」という規則の多くを持つ部分です（3章，5章）．

2.2　表を掛けると，数の個数が変わる

9つの表の使い方は2通りしかありません．

1. **行を引く**：wte と wpe は，文字や位置の番号で1行を取り出すだけです．辞書を引くのと同じで，掛け算はしません．m（番号3）を読むなら wte の3行目（0 から数えて），p3 なら wpe の3行目です．
2. **掛ける**：ほかの7つの表は，数の並びに掛けます．「表を掛ける」とは，表の各行と数の並びを成分ごとに掛けて足す計算です（行列とベクトルの積）．

掛ける計算の例を見ます．W_Q の1行目 $(-0.41, +0.64, +0.12, -0.40)$ と，黒板の4つの数 $(-1.42, +0.31, -0.07, -1.38)$ を掛けて足すと

$$(-0.41)\times(-1.42) + (+0.64)\times(+0.31) + (+0.12)\times(-0.07) + (-0.40)\times(-1.38) = +1.33$$

で，これが q の1つ目の数です．W_Q には4行あるので，同じ計算を4回して4つの数ができます．

ここが大事なところです．**表の行の数だけ，出てくる数ができます**．だから表を掛けると，数の個数が変わることがあります（図5）．W_Q は 4×4 なので，4つの数から4つの数ができます．W_1（fc1）は 16×4 なので，4つの数から16個の数ができます．逆に W_2（fc2）は 4×16 なので，16個から4つに戻ります．lm_head（U）は 7×4 なので，4つの数から7つの得点ができます．

[11] 黒板の幅 4 は，学習の前に人間が決める設定の数（`n_embd`）で，学習で決まる数ではありません．ここでは表を読める大きさにするために 4 にしました．元の microgpt は 16，GPT-2 の最小のモデルは 768 です．幅を変えると，ほかの表の大きさも変わり，覚えられる量も変わります（4 だと名前6個くらいが限度でした．7章）．

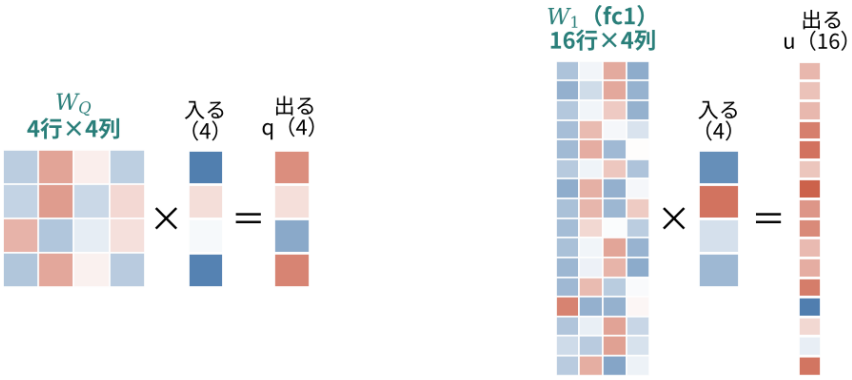


図 5　表を掛けると，数の並びの長さが「表の列の数」から「表の行の数」に変わる．
左：W_Q（4→4）．右：W_1（4→16）．

> **「4つが急に16個になる」のは MLP の中だけ**です．16×4 の表で16個に広げ，4×16 の表で4つに戻します．16×16 の表はありません．広げた16個は MLP の中だけで使われ，黒板に書かれるのは4つに戻したあとの数です（3章の段階3）．

> **式で書くと**　$\boldsymbol{y} = W\boldsymbol{x}$，$y_i = \sum_j W_{ij}x_j$．形は $(m \times n)$ の表 ×（n）＝（m）．例：$\boldsymbol{u} = W_1\hat{\boldsymbol{x}}$ は $(16 \times 4) \times$（4）＝（16）．

2.3　表は，どの位置でも同じものを使う

9つの表のうち，位置によって使い分けるのは wpe（位置ごとの数の表）だけです．1文字目なら wpe の p1 の行，4文字目なら p4 の行というように，引く行を変えます．ほかの8つの表は，どの位置でもまったく同じものを使います（表8）[12]．だから名前が長くなると，計算の回数は増えますが，表は増えません．

表 8　emma を読むときに使う行

位置	読む文字	wte の行	wpe の行	ほかの7つの表
p0	BOS	BOS の行	p0 の行	同じもの
p1	e	e の行	p1 の行	同じもの
p2	m	m の行	p2 の行	同じもの
p3	m	m の行（p2 と同じ）	p3 の行	同じもの
p4	a	a の行	p4 の行	同じもの

2.4　名前の長さで変わるもの，変わらないもの

名前の長さで変わるのは，計算の途中で作られる数の方です．mia，ava は4つの位置，anna，emma は5つの位置を使います．次の節で説明する黒板は位置ごとに1枚あるので，**黒板の枚数**（位置の数）は名前で変わりますが，**黒板の幅**（4つの数）は変わりません．emma は4文字で幅も4ですが，黒板は5枚です．この「4」と「5」は別の数です．

[12]例：emma の2つの m は，wte の m の行は同じですが，足す wpe の行（p2 と p3）がちがうので，最初のメモが変わります（3章）．

位置の数には上限（ここでは 5）があります．wpe には p0〜p4 の5行しかないので，6番目の位置に足す行がありません．4文字までの名前なら，最後の「終わり」まで学習できます．それより長い名前は後ろが切られます[13]．

2.5　黒板のたとえ

文字の位置ごとに，4つの数を書ける小さな黒板があり，みんながこれを読んで結果を書き足していきます（図6）．書き込みはいつも足し算で，前に書いたものは消しません．最後に lm_head が黒板を読んで，次の文字の得点にします．専門用語では，この黒板を**残差の流れ**（residual stream）と呼びます．

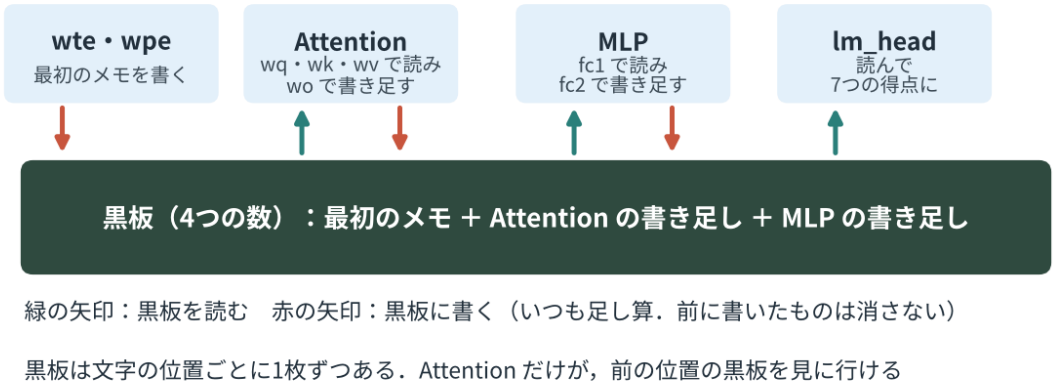


図 6　黒板のたとえ．9つの表は，黒板を「読む」か「書く」かのどちらかをしている．

1つの位置の黒板では，次の4つのことが順に起きます．

1. 最初のメモを書く（wte の文字の行 ＋ wpe の位置の行）
2. Attention の結果を足す（前の位置の黒板から写してくる）
3. MLP の結果を足す（16個の検出器が反応して書き足す）
4. lm_head が黒板を読んで，7つの文字の得点を出す

式で書くと　$\boldsymbol{x}^{(0)}$ ＝ 最初のメモ，$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)}$ ＋ (Attention の結果)，$\boldsymbol{x}^{(2)} = \boldsymbol{x}^{(1)}$ ＋ (MLP の結果)，得点 $\boldsymbol{z} = U\boldsymbol{x}^{(2)}$．黒板はどれも （4）．

なぜ「黒板」にたとえるのか

1. **みんなが読み書きする共有の場所**：Attention も MLP も lm_head も，同じ黒板を読みます．Attention と MLP は，結果をその黒板に書き込みます．
2. **書き込みは足し算で，前の内容を消さない**：ここがいちばん大事です．最初に書いたメモも最後まで黒板に残ります．だから，最後の得点を「誰がどれだけ足したか」に分けられます（3章，5章）．
3. **最後に一人が黒板全体を読んで答える**：lm_head が，積み重なった黒板の内容を7つの文字の得点として読みます．

[13]例：sophia を学ばせると BOS, s, o, p, h までしか読まず，i の次の a や最後の「終わり」を学習できません．上限があるのは，wpe の表の大きさを最初に決めておく必要があるのと，Attention が前の位置をすべて見るので，長いほど計算が増えるためです．元の microgpt の上限は 16 です．

ふつうの「バケツリレー」式では，前の段の結果を次の段が丸ごと置きかえます．黒板式は，前の内容を残したまま書き足していくところがちがいます．

2.6　黒板と表はちがうもの

黒板への書き足しは，1つの名前を1回読んで次の文字を当てるまでの計算の中で起きます．次の名前を読むときは，新しい黒板で一から始めます．300回の学習でくり返し調整されて残るのは，黒板ではなく9つの表の方です（表9）．

表 9　黒板と9つの表（重み）のちがい

	黒板	9つの表（重み）
いつ変わるか	1回の計算の中で	1回の学習ごとに
変わり方	足し算で書き足す	損失が減る向きに少し動かす
次の名前を読むとき	新しい黒板で一から始める	そのまま使う
学習後に残るか	残らない	残る（これが 268 個の重み）

> **たとえると**：黒板は1問ごとの計算用紙，表は学習のたびに少しずつ書き直していく自分のノートです．計算用紙は問題ごとに新しくしますが，ノートは直しながら使い続けます．3章で追いかけるのは計算用紙（黒板）の方で，ノート（表）は学習後のものに固定しておきます．4章ではノートの直し方を見ます．

2.7　黒板は文字の位置ごとに1枚ずつある

教室にたとえると，生徒（文字の位置）がそれぞれ自分の小さな黒板を持っています．emmaなら5枚です（図7）．MLP の段では，自分の席で，自分の黒板だけを見て考えます．Attention の段では，前の席の黒板を見に行って，内容を写してきます．ほかの位置の情報を手に入れる方法は，Attention だけです．



図 7　emma の5枚の黒板に書かれた最初のメモ（学習後）．d0〜d3 は黒板の4つの欄．赤は正，青は負の数．

2.8　たとえの限界：本物の黒板とちがうところ

1. **4マスしかなく，とても狭い**：7種類の文字の情報や途中の計算結果を，4つの数に詰め込みます．そのため，同じ方向をいくつもの役目で使い回します．

2. **マスの位置に決まった意味がない**：本物の黒板なら「左上に日付」と決まった場所がありますが，ここでは決まっていません（5章）．
3. **読む側は大きさをそろえた写しを読む**：Attention と MLP は，4つの数の大きさをそろえてから（**rmsnorm**）黒板を読みます．rmsnorm は4つの数をそれぞれ「2乗の平均の平方根」で割る計算で，くわしくは3章で見ます．
4. **消すことはできない**：ただし，負の数を足して打ち消すことはできます．

2.9　読む表と書く表

表 10　9つの表の役目

表	すること	たとえると
wte, wpe	文字と位置に応じた最初の4つの数を書く	黒板に最初のメモを書く
wq, wk	どの位置の黒板を見に行くかを決める	図書館で探す本を決める
wv, wo	見に行った黒板から内容を写してくる	本の中身を写してくる
fc1	16個の検出器が黒板を読む	型紙と照らし合わせる
fc2	反応した検出器に応じて書き足す	「これを足せ」という規則
lm_head	最後に黒板を読んで7つの文字の得点にする	答えを読み上げる

読む表は「特徴を見つける」係，書く表は「見つけたら何を足すか」の係です．覚えていることは，読む表と書く表のペア（wv と wo，fc1 と fc2）の中に入っています．

図8は，学習後の9つの表です．形（行×列）を見比べてください．3章ではこの表を使って計算をたどり，5章では，表が何をしているかを読みます．ただし，マスの数を一つずつ読むのではありません．一つのマスの値には意味がないからです（5章）．

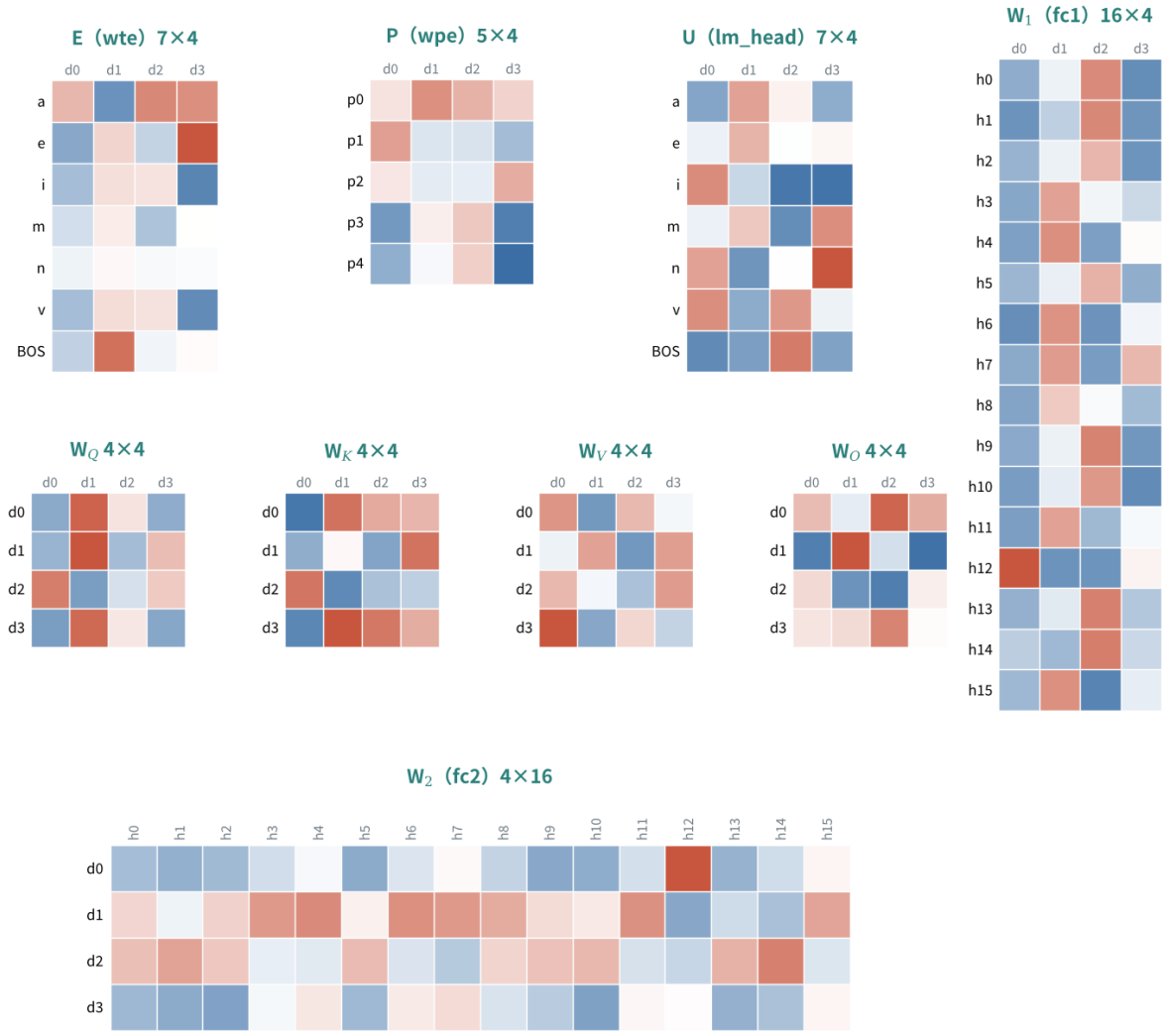


図 8　学習後の9つの表（268個の数）．赤は正，青は負．色の濃さの目安は表ごとにそろえた（いちばん大きい値が最も濃い）．

2章のまとめ：268 個の重みは9つの表のマスです．wte と wpe は行を引くだけ，ほかの7つは数の並びに掛けます．掛けると，数の個数は「列の数」から「行の数」に変わります．表はどの位置でも同じものを使い，位置で変わるのは wpe の引く行だけです．計算の途中の数は，位置ごとの黒板（4つの数）に足し算で書き足されます．黒板は1回の計算ごとに使い捨てで，学習で残るのは表の方です．

3　1文字を読む

この章では，AI が1文字を読んで次の文字の確率を出すまでの計算（順伝播）を，実際の数でたどります．使うのは，300回の学習を終えた重みです．予想がよく当たるので，計算の意味が見えやすいからです．重みをどう動かしたか（学習）は4章で見ます．巻頭の図0は，この章の計算を1枚にまとめたものです．そばに置いて読んでください．

3.1　1回の予想の流れ：5つの段階

1文字を読むたびに，AI は図9の5つの段階で計算します[14]．この5段階を，5つの位置（BOS, e, m, m, a）それぞれで行います．どの位置でも同じ表を使います（2章）．

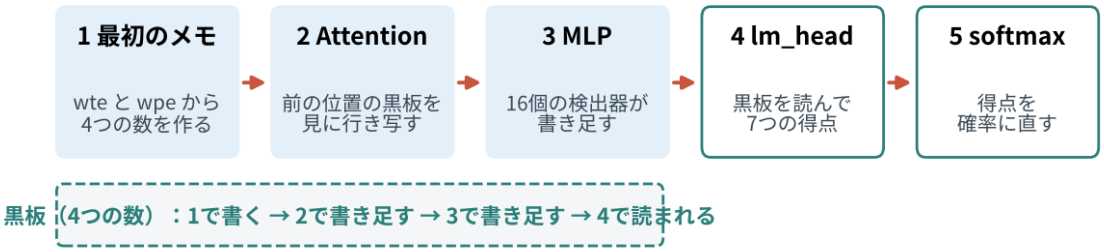


図 9　1回の予想の5つの段階．

追いかける位置：emma の2つ目の m　emma の「emm」まで読んだ位置（p3）を追いかけます．同じ m を読んでいるのに，1つ目の m（p2，読んだところ em）では次が m，2つ目の m（p3）では次が a です．文字だけでは決まらない問題で，「何文字目か」と「前に何があったか」を使わないと解けません．AI はこの2つを見分けて，p3 で a に 98.8% をつけました．どの段階がそれを使うのかに注目してください．

3.2　段階1：最初のメモ

> **形**　文字の番号 3 → wte の行 （4），位置 p3 → wpe の行 （4）．足して （4），rmsnorm して 黒板 $\boldsymbol{x}^{(0)}$ （4）．

wte：文字ごとに4つの数が並んだ辞書　wte は，7種類の文字それぞれに4つの数を割り当てた表です（図10左）．辞書を引くように，今読む文字の行を取り出します．m を読むなら，m の行 $(-0.06, +0.03, -0.12, +0.00)$ です．emma の2つの m は，どちらも同じ m の行を使います．wte だけでは，2つの m は区別できません．4つの数の値は学習で決まったもので，1つずつの数には決まった意味がありません[15]．

[14] プログラムでは，1〜4が `gpt()` という関数（108–144行目），5が `softmax()`（97–101行目）です．

[15] d0〜d3 は4つの数の欄（4次元の空間の4本の軸）です．軸の向きを回しても AI の働きは変わらないので，「d2 が負だから…」とは読めません（5章）．wte は word token embedding（トークンの埋め込み）の略です．

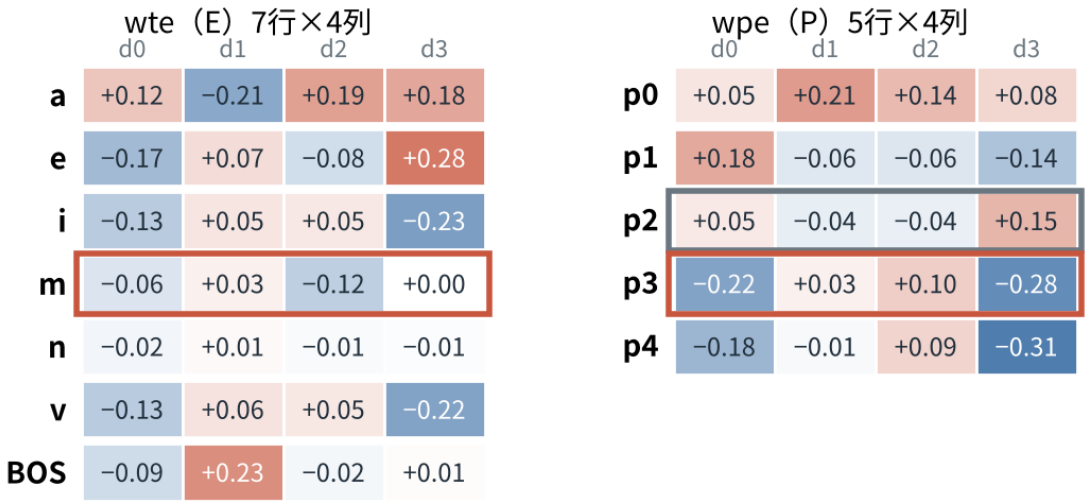


図 10　学習後の wte（左）と wpe（右）．枠は，p3 で使う m の行と p3 の行（灰色は p2 の行）．

wpe：位置ごとに4つの数が並んだ表　wpe は「何文字目か」に応じた4つの数の表です（図10右）．1つ目の m は p2 の行，2つ目の m は p3 の行を使います．ここで初めて2つの m に差がつきます．リレーの走者のようなものです．同じ選手でも，第2走者か第3走者かで仕事がちがいます．位置の表があるのは，Attention には順番を見分ける力がないからです[16]．

文字の数と位置の数を足す　2つの表の同じ欄（d0 どうし，d1 どうし...）を足します（図11）．p2 の合計 $(-0.02, -0.01, -0.15, +0.15)$ と，p3 の合計 $(-0.28, +0.06, -0.01, -0.28)$ はまったくちがう数になりました．文字の情報（m）と位置の情報（3文字目）が，1組の4つの数に混ざって入ります．

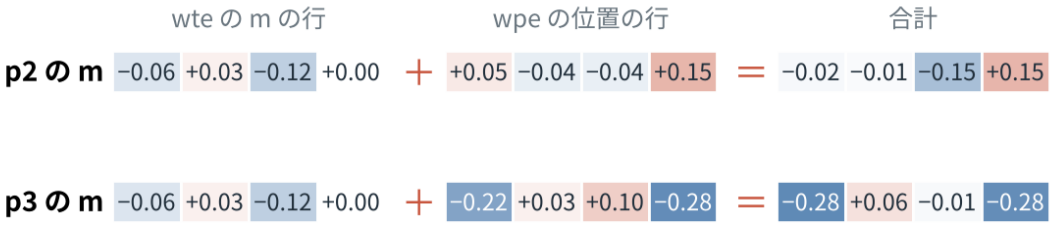


図 11　wte の行と wpe の行を足す（p2 と p3 の m）．

大きさをそろえる（rmsnorm）　足した4つの数は，大きさをそろえてから黒板に書きます．p3 の場合，(1) 4つの数をそれぞれ2乗して平均すると $\{(-0.28)^2 + 0.06^2 + (-0.01)^2 + (-0.28)^2\} \div 4 = 0.0402$，(2) その平方根は 0.200，(3) 4つの数をそれぞれ 0.200 で割ると $(-1.42, +0.31, -0.07, -1.38)$ になります．これが p3 の最初のメモ $\boldsymbol{x}^{(0)}$ です．

[16] Attention は前の位置をまとめて「袋に入れて」見渡す計算なので，並び順の情報を自分では持てません．wpe は word position embedding（位置の埋め込み）の略です．

割ったあとの4つの数は，2乗の平均がいつも 1 になります．矢印の長さでいえば，いつも 2（$=\sqrt{4}$）です．数の並びの向きは変わらず，大きさだけがそろいます[17]．こうしておくと，このあと黒板を読む係が，数の大きさにまどわされず，並び方（向き）だけで判断できます．録音の音量の自動調整のように，大声でも小声でも同じ音量にそろえ，話の中身を聞き取りやすくするのです．

式で書くと　$\boldsymbol{x}_t^{(0)} = \mathrm{rms}(E_{c_t} + P_t)$，$\mathrm{rms}(\boldsymbol{a}) = \boldsymbol{a}/\sqrt{\frac{1}{4}\sum_i a_i^2}$．$E_{c_t}$ は wte の c_t 行目，P_t は wpe の t 行目．

最初のメモを比べる　同じ m でも，p2 と p3 の最初のメモは向きがまったくちがいます（向きのそろい方 cos は -0.41）[18]．反対に，ちがう文字の e（p1）と m（p2）は，ほとんど同じ向きのメモになりました（cos は 0.98）．どちらも「次は m」だからです．メモの近さは，文字の形ではなく「次に何が来るか」という役目の近さを表すように，学習で作られています．

3.3　段階2：Attention

形　黒板（4）を rmsnorm して $\hat{\boldsymbol{x}}$（4）$\to W_Q, W_K, W_V$（各 4×4）で q, k, v（各（4））→ 得点と割合（見られる位置の数，p3 なら（4））→ 混ぜた中身（4）$\to W_O$（4×4）で（4）→ 黒板に足す．

図書館で調べものをするしくみ　Attention は，前の位置の黒板を見に行き，必要なものを写してくるしくみです．図書館にたとえると，各位置の黒板は本で，それぞれ「見出し」（key，k）と「中身」（value，v）を持っています．今の位置は「何を探しているか」のメモ（query，問い合わせ，q）を作り，次の3つを行います．

1. 問い合わせ q と，各位置の見出し k を比べる．似ている見出しほど高い得点にする．
2. 得点を割合に直す（合計 100%）．どの本をどれだけ読むかの配分を決める．
3. 割合に応じて中身 v を混ぜて写し，自分の黒板に書き足す．

見に行けるのは，自分と自分より前の位置だけです．先（答え）の文字はまだ見えません．p3 なら BOS，e，m，m の4つの黒板を見られます．

q，k，v を作る　黒板をもう一度 rmsnorm した $\hat{\boldsymbol{x}}$ に，3つの表 W_Q，W_K，W_V をそれぞれ掛けて，q，k，v を作ります[19]．計算のしかたは2章で見たとおりで，p3 の問い合わせは $\mathrm{q} = (+1.33, +0.38, -1.19, +1.44)$ です．k と v は，それぞれの位置を読んだときに作って，あとの位置のためにとっておいたものです[20]．

問い合わせと見出しの似かたを測る　q と各位置の k を，成分ごとに掛けて足し（積の和），2 で割ったものが得点です（表11）[21]．BOS の見出しとの積の和は $(+1.33)(+1.27) +$

[17] rmsnorm は root mean square normalization（2乗平均の平方根でそろえる）の略です（103–106行目）．0 で割らないように，実際は平均に 0.00001 を足してから平方根をとります（付録A.2）．

[18] cos（コサイン）は，2つの数の並びを矢印とみたときの向きのそろい方です．同じ向きなら +1，直角なら 0，反対向きなら −1 です（付録A.2）．

[19] 最初のメモはすでに大きさがそろっているので，ここでの rmsnorm で数はほとんど変わりません．

[20] 121–122行目．前の位置の k と v をとっておくしくみは，大きな AI で KV キャッシュと呼ばれるものと同じです（6章）．

[21] 2 で割るのは，数の個数（4）の平方根です．数の個数が多いと積の和が大きくなりすぎるのを防ぎます（付録A.3）．

$(+0.38)(+0.09)+(-1.19)(-1.19)+(+1.44)(+1.56)=+5.38$ です．4つの数の符号がそろっているほど，積の和は大きくなります．q と BOS の見出しは，4つとも同じ符号でした．得点がいちばん高いのは BOS（始まり）で，p3 の m は名前の始まりの黒板を見に行こうとしています．

表 11　p3 の問い合わせと各位置の見出し

見に行く位置	見出し k	積の和	÷2＝得点	e の得点乗	割合
BOS（p0）	$(+1.27,+0.09,-1.19,+1.56)$	$+5.38$	$\mathbf{+2.69}$	14.76	**76.2%**
e（p1）	$(-0.09,+1.17,+0.11,-0.29)$	-0.22	-0.11	0.90	4.6%
m（p2）	$(-0.03,+1.26,+0.06,-0.26)$	$+0.00$	$+0.00$	1.00	5.2%
m（p3，自分）	$(+0.56,-0.16,-0.55,+0.45)$	$+1.99$	$+1.00$	2.71	14.0%
合計				19.36	100%

得点を割合に直す（softmax）　e（約 2.718）の得点乗を求めて，合計で割ります（表11の右2列）．得点が 1 高いと，約 2.7 倍多く見ます．p3 の m は，注目の 76% を名前の始まり（BOS）に向けました（図12左）．持ち時間の配分のように，得点の高い本ほど長く読みますが，ほかの本もほんの少しは読みます[22]．

ここで出てきた割合の並びの長さ（p3 では4）は，**見られる位置の数**です．p0 では1，p4 では5になります．黒板の幅の4とは別の数で，表の形とも関係しません．

> **式で書くと**　$\boldsymbol{q}_t = W_Q\hat{\boldsymbol{x}}_t$，$\boldsymbol{k}_j = W_K\hat{\boldsymbol{x}}_j$，$\boldsymbol{v}_j = W_V\hat{\boldsymbol{x}}_j$．得点 $a_{tj} = \boldsymbol{q}_t \cdot \boldsymbol{k}_j/\sqrt{4}$，割合 $\alpha_{tj} = e^{a_{tj}} \big/ \sum_{j' \le t} e^{a_{tj'}}$（$j \le t$）．

中身を混ぜて写し，黒板に書き足す　割合に応じて，4つの位置の中身 v を混ぜます．$76.2\% \times \boldsymbol{v}(\text{BOS}) + 4.6\% \times \boldsymbol{v}(\text{e}) + 5.2\% \times \boldsymbol{v}(\text{m}) + 14.0\% \times \boldsymbol{v}(\text{m}) = (-1.11,+0.76,-0.18,-1.18)$ です．それを W_O の表で黒板の形に直して[23] $(-0.97,+2.22,-0.58,-0.13)$ とし，最初のメモに足します．黒板は $\boldsymbol{x}^{(1)} = (-2.38,+2.53,-0.65,-1.51)$ になりました．写してきたものの大部分は，名前の始まりの黒板の中身です．「始まりから3文字目の m」という情報が，p3 の黒板に書き加えられ，このあとの MLP がこれを手がかりにできます．

> **式で書くと**　$\boldsymbol{o}_t = \sum_{j \le t} \alpha_{tj} \boldsymbol{v}_j$（4），$\boldsymbol{x}_t^{(1)} = \boldsymbol{x}_t^{(0)} + W_O \boldsymbol{o}_t$（4）．

[22] softmax は，得点の差を「何倍か」に直して，合計が 1 になるように配る計算です．最後の段階5でも同じ計算を使います（付録A.3）．

[23] W_O の掛け算は，混ぜた中身を黒板の4つの欄に合わせて書き直す計算です（133 行目）．W_Q，W_K，W_V，W_O の4つの表を合わせて Attention の部品です（各 $4 \times 4 = 16$ 個，計64個の重み）．

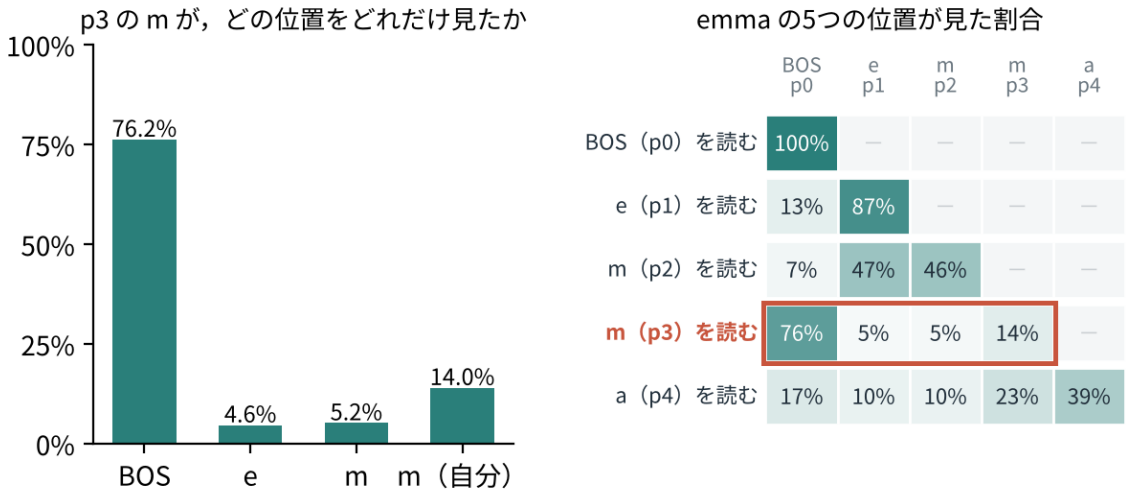


図 12　Attention の割合．左：p3 の m が見た割合．右：emma の5つの位置が見た割合（各行の合計 100%．「—」はまだ読んでいない先の位置）．

emma の5つの位置は，どこを見たか　図12右を見ると，右上の「—」は，まだ読んでいない先の位置で，見ることはできません．始まり（p0）は自分しか見られないので，いつも 100% です．そのため始まりの予想は，どの名前でもまったく同じ計算になります．p3 の m だけが，始まり（BOS）を強く見ています．4つの名前すべてで，次が a になる場面（mi，av，ann，emm）では，BOS に 76〜84% 注目していました（5章）．

3.4　段階3：MLP

> **形**　黒板 $\boldsymbol{x}^{(1)}$（4）を rmsnorm（4）→ W_1（16×4）で $\boldsymbol{u}$（16）→ ReLU で $\boldsymbol{h}$（16）→ W_2（4×16）で（4）→ 黒板に足す．

「型紙」と照らし合わせる検出器の集まり　MLP（multi-layer perceptron，多層パーセプトロン）は，黒板の4つの数を一度16個に広げてから，また4つに戻す部分です[24]．W_1（fc1）の16本の行は，それぞれ1枚の「型紙」です．黒板の並び方が型紙に似ていれば反応（点灯）し，似ていなければ黙ります．W_2（fc2）は，点灯した検出器ごとに「これを黒板に足せ」という内容を持っています．MLP は自分の黒板だけを見ます．ほかの位置の情報は，すでに Attention が写してきています．波形を型と照らし合わせ，基準を超えたら知らせる地震計のトリガーと同じ形です．

16個の検出器の反応　型紙と黒板の積の和（似かた）を16個求めます（図13）．これが W_1（16×4）を掛けて4つの数から16個を作る計算です．似かたが負のもの（h12，h14）は 0 にして黙らせます．この「負なら 0」を **ReLU** と呼びます[25]．p3 では 14 個が点灯しました．次が a になる場面（mi，av，ann，emm）では，14〜15個が一斉に点灯します．「もうすぐ a」のときの共通の反応です．

[24] 137–141行目．読む前に，ここでも大きさをそろえます（rmsnorm）．

[25] ReLU は rectified linear unit の略です．神経細胞の「刺激の合計がしきい値を超えると発火する」しくみをまねたもので，ここではしきい値が 0 です（付録A.4）．

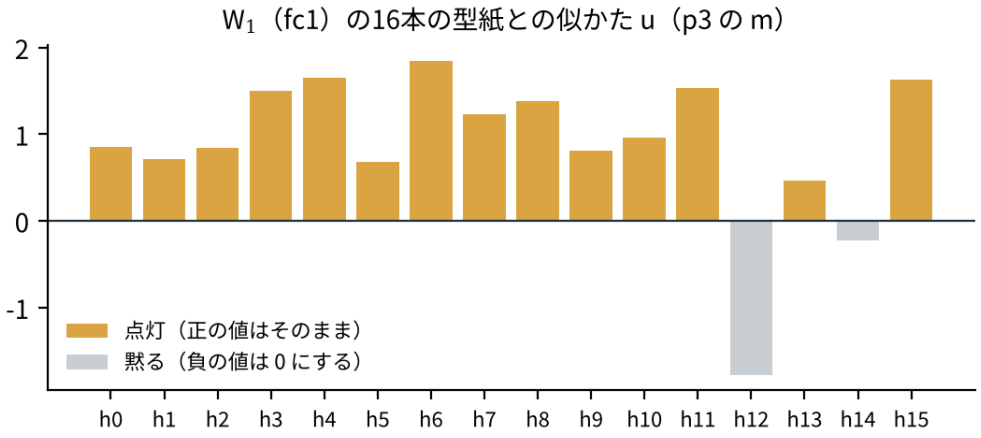


図 13　p3 の m での，16個の型紙との似かた $\boldsymbol{u}$．負の値（灰色）は 0 にされる．

点灯した検出器が，黒板に書き足す　点灯の強さに応じて W_2 の内容を足し合わせると，$(-4.74, +7.97, +0.76, -2.59)$ になります．これが W_2（4×16）で16個から4つに戻す計算です．これを黒板に足すと，最後の黒板は $\boldsymbol{x}^{(2)} = (-7.12, +10.50, +0.11, -4.10)$ です．黒板に書いたものの大きさ（矢印の長さ）を比べると，最初のメモが 2.00，Attention の結果が 2.49，MLP の結果が 9.66 でした[26]．いちばん大きく書き足したのは MLP です．重みの約半分（128個）が MLP にあり，「こういうときは次にこれ」という規則の多くをここで持っています．

式で書くと　$\boldsymbol{u}_t = W_1\,\mathrm{rms}(\boldsymbol{x}_t^{(1)})$　（16），$\boldsymbol{h}_t = \max(0, \boldsymbol{u}_t)$　（16）（成分ごと），$\boldsymbol{x}_t^{(2)} = \boldsymbol{x}_t^{(1)} + W_2\boldsymbol{h}_t$　（4）．

3.5　段階4・5：得点と確率

形　黒板 $\boldsymbol{x}^{(2)}$（4）$\to U$（lm_head，7×4）で得点 $\boldsymbol{z}$（7）$\to$ softmax で確率 $\boldsymbol{p}$（7）．

lm_head：最後の黒板を読んで得点を出す　lm_head の各行は，その文字の「読み取り用の型紙」です．最後の黒板と行の積の和が，その文字の得点になります[27]．たとえば a の得点は

$$(-0.38)\times(-7.12) + (+0.34)\times(+10.50) + (+0.05)\times(+0.11) + (-0.35)\times(-4.10) = +7.70$$

です[28]．7つの文字の得点は，a +7.70，e +3.13，i −2.39，m +0.80，n −9.62，v −6.36，終わり +0.89 で，a がとびぬけて高くなりました（図14左）．lm_head の前には rmsnorm がありません．

得点を確率に直す（softmax）　Attention のときと同じ softmax で，得点を確率に直します．a と e の得点の差は 4.57 で，e の 4.57 乗は約 96 なので，a は e の約 96 倍ありそう，という答えになります．正解の a に 98.8% をつけました（図14右）．

[26] 矢印の長さは，4つの数の2乗の和の平方根です．

[27] lm_head は language model head（言語モデルの頭部，最後の出力の部分）の略です（143行目）．得点は専門用語でロジット（logit）と呼びます．

[28] 表の数を小数2桁に丸めて計算すると，最後の桁が少しずれることがあります．

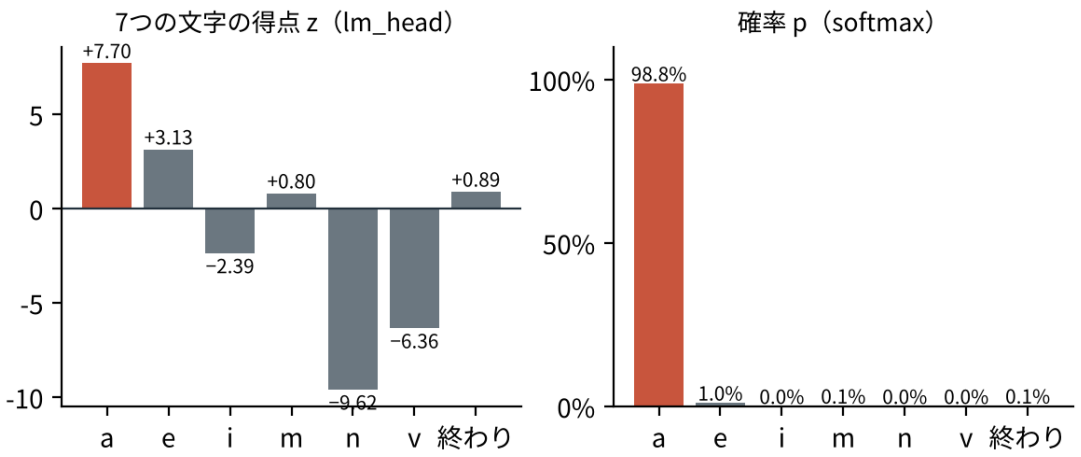


図 14　p3（emm の次）の7つの得点（左）と確率（右）．

式で書くと　$\boldsymbol{z}_t = U\boldsymbol{x}_t^{(2)}$ （7），$p_{t,c} = e^{z_{t,c}} / \sum_{c'} e^{z_{t,c'}}$ （7）．確率の比は $p_a/p_e = e^{z_a - z_e} = e^{4.57} \approx 96$.

得点は3つの書き込みの足し算に分けられる　黒板への書き込みはいつも足し算で，lm_head も掛けて足すだけの計算なので，得点もぴったり3つに分かれます（図15）[29]．

$$\text{a の得点 } +7.70 = \text{最初のメモ } +1.12 + \text{Attention } +1.13 + \text{MLP } +5.44$$

p3 では，最初のメモと Attention が a を少し押し上げ，MLP が大きく押し上げました．n と v は MLP が大きく押し下げています．

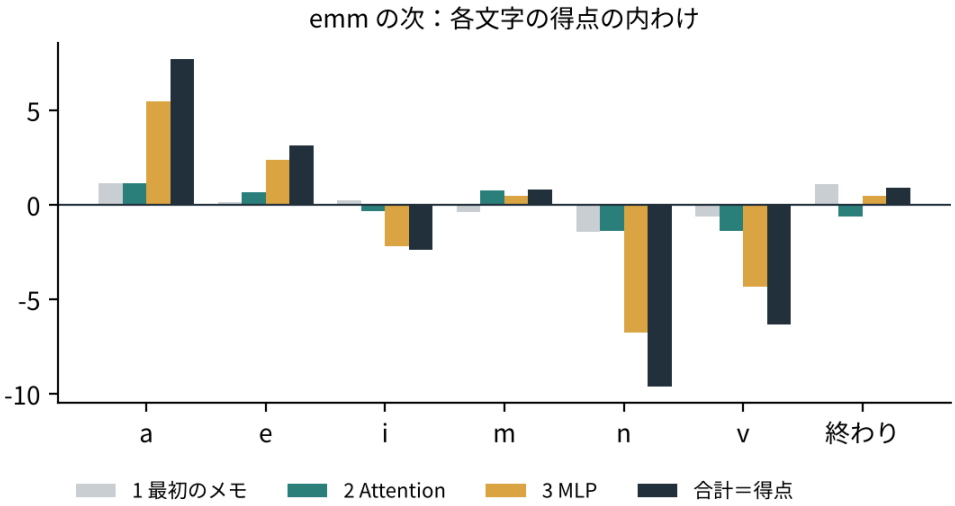


図 15　emm の次の，各文字の得点の内わけ．

式で書くと　$\boldsymbol{z} = U\boldsymbol{x}^{(2)} = U\boldsymbol{x}^{(0)} + U(W_O\boldsymbol{o}) + U(W_2\boldsymbol{h})$.

名前の終わりでは，MLP が「終わり」を押し上げる　emma を読み終えた p4 では，「終わり」の得点は $+12.26 =$ 最初のメモ $+1.57 +$ Attention $+0.54 +$ MLP $+10.15$ で，確率は 99.6%

[29] lm_head の前に rmsnorm がないので，分けた3つの和は得点と小数第14位まで一致します（付録A.5）．

でした．点灯した検出器は9個で，p3 とは組み合わせがちがいます．「a を読み終えた」場面に反応する組が点灯しています．ここでも決め手は MLP です．

3.6　emma の5問と，14の場面

emma の5つの問題の答えは表12のとおりです．p1〜p4 は，正解に 94〜99.6% をつけました．始まり（p0）では e は 19.3% しかありませんが，これは失敗ではありません．データでは始まりの次は a が 50%，e と m が 25% ずつで，AI もほぼその割合で迷っています（1章「正しく迷う」）．

表 12　emma の5つの問題の答え

位置	読んだところ	正解	正解の確率	AI の予想（上位3つ）
p0	始まり	e	19.3%	a 50.9%，m 29.7%，e 19.3%
p1	e	m	94.0%	m 94.0%，e 3.9%，a 1.3%
p2	em	m	94.4%	m 94.4%，e 3.7%，a 1.2%
p3	emm	a	98.8%	a 98.8%，e 1.0%，終わり 0.1%
p4	emma	終わり	99.6%	終わり 99.6%，a 0.4%

4つの名前の14の場面すべてで見ると（表13），答えが一つに決まる12の場面では 80〜99.8%，迷う2つの場面（始まり，a）ではデータの割合に近い確率をつけています．いちばん低いのは m の次の i（80.2%）でした．

表 13　14の場面での，正解の確率（学習後）

読んだところ	データ	AI
始まり	a 50%，e 25%，m 25%	a 50.9%，e 19.3%，m 29.7%
m	i 100%	i 80.2%
mi	a 100%	a 99.2%
mia	終わり 100%	終わり 99.7%
a	v 50%，n 50%	v 49.9%，n 45.6%
av	a 100%	a 99.2%
ava	終わり 100%	終わり 99.8%
an	n 100%	n 84.4%
ann	a 100%	a 98.8%
anna	終わり 100%	終わり 99.8%
e	m 100%	m 94.0%
em	m 100%	m 94.4%
emm	a 100%	a 98.8%
emma	終わり 100%	終わり 99.6%

3章のまとめ：1文字を読むたびに，最初のメモ（wte＋wpe，大きさをそろえる）→ Attention → MLP → lm_head の得点 → softmax の確率，の5つの段階で計算します．黒板はずっと4つの数で，16個になるのは MLP の中だけ，7個になるのは最後の得点と確率だけです．同じ m を見分けたのは，位置の表（wpe）と Attention です．得点は3つの書き込みの足し算で，いちばん大きく押し上げたのは MLP でした．

4　外れ具合と学習

3章では学習後の重みを使いました．この章では，重みを学習前のでたらめな値に戻し，そこから300回の学習で3章の重みにたどり着くまでを見ます．1回の学習（1ステップ）は，順伝播（3章の計算）→ 損失を測る → 逆伝播で勾配を求める → 重みを更新する，の1周でした．1回の学習では名前を1つだけ使い，mia，ava，anna，emma，mia... と4つの名前を順にくり返します[30]．

4.1　学習前の AI

268個の重みは，最初はすべて小さな乱数です．だいたい −0.16〜+0.16 の範囲に散らばっています[31]．どの文字の行も，どの位置の行も，まだ何の役目も持っていません．入部したての部員のように，やる気はあっても，まだ何も知らない状態です．

重みがでたらめなので，AI はどの問題でも7文字に 12〜17% くらいずつ確率を配ります（図16）．7文字に同じ確率をつけると，どれも 1/7 ≈ 14.3% です．これは当てずっぽうと同じで，ここから学習が始まります．

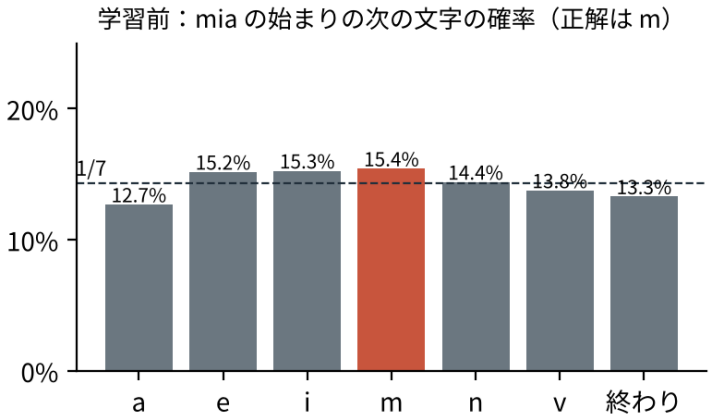


図 16　学習前の重みで，mia の始まりの次の文字の確率．点線は 1/7.

4.2　外れ具合を測る（損失）

AI は次の文字を一つに決めず，7つの文字それぞれに確率を割りふります．そこで，正解の文字につけた確率から罰点を決めます．これを**損失**（loss）と呼びます．罰点は − ln（正解の確率）で計算します（図17）[32]．確率 1（100%）なら罰点 0（完全に的中），0.5 なら 0.69（二択で迷った），1/7 なら 1.95（7つから当てずっぽう），0.01 なら 4.61（「まずない」と言って外した）です．

[30] プログラムでは，名前の取り出しが 156–157行目，順伝播と損失が 163–169行目，逆伝播が 172行目（`loss.backward()`），重みの更新が 174–182行目です．

[31] 平均 0，標準偏差 0.08 の正規分布（ガウス分布）から1つずつ選んでいます（80行目の `random.gauss(0, std)`）．ほとんどの値は標準偏差の2倍（±0.16）の中に入ります．

[32] ln は自然対数で，「e の何乗でその数になるか」を表します．確率が 1 に近いほど 0 に近く，0 に近いほど限りなく大きくなります．

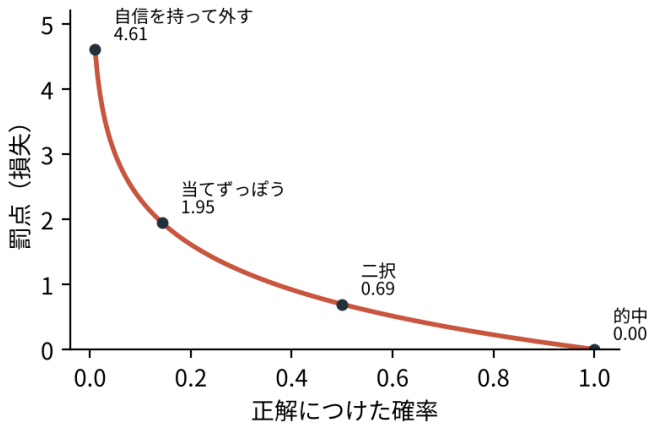


図 17　正解につけた確率と罰点.

天気予報の採点と同じ：「降水確率 70%」と言って雨なら，罰点は小さい．「1%」と言い切って雨なら，大きな減点です．自信を持って外すほど，罰点は急に大きくなります．「正解を知ったときの驚きの大きさ」とも言えます.

式で書くと　1つの名前（問題が n 個）の損失 $L = \frac{1}{n}\sum_{t=0}^{n-1}\left(-\ln p_{t,\,c_{t+1}}\right)$．$p_{t,\,c_{t+1}}$ は，位置 t で正解の文字につけた確率.

1回目の学習（mia）の損失　mia には4つの問題があります（表14）．それぞれの罰点を求めて平均したものが，1回目の学習の損失です．平均 1.953 は，当てずっぽうの罰点（$\ln 7 \approx 1.946$）とほぼ同じです．学習の目標は，この罰点の平均を下げることです[33].

表 14　1回目の学習（mia）の罰点

問題	読んだところ	正解	正解の確率	罰点	1回目の後の確率
1	始まり	m	15.4%	1.869	16.5%
2	m	i	11.6%	2.151	12.3%
3	mi	a	17.1%	1.764	18.4%
4	mia	終わり	13.1%	2.029	14.0%
平均				**1.953**	（罰点 1.889）

なぜ「正答率」ではなく罰点なのか　正答率（いちばん確率の高い文字が正解だった割合）は，確率を 0.15 から 0.16 に少し上げても，一番が入れかわらなければ変わりません．正答率は階段のようにとびとびにしか動かないので，「重みを少し動かしたら，どれだけよくなったか」が測れません．罰点は，正解の確率がほんの少し上がるだけでも，ほんの少し下がります．なめらかに変わるので，「罰点が下がる向き」が計算できます．階段では，どちらに一歩出れば下がるのか分かりませんが，なめらかな坂なら，足もとの傾きで下りる向きが分かります.

[33] このような損失を，専門用語では交差エントロピー損失（cross-entropy loss）と呼びます（付録A.6）.

4.3　どの重みを，どちらに動かすか（勾配）

霧の山の中で谷へ下りたいとき，遠くは見えなくても，足もとの傾きは分かります．傾きの下る向きに一歩ずつ進めば，谷に近づけます（図18）．268個の重みは，268本の方向です．それぞれの重みについて，「少し増やすと損失がどれだけ増えるか」の割合を**勾配**（gradient）と呼びます．勾配が正の重みは減らし，負の重みは増やせば，損失が下がります．勾配は「その場所での傾き」なので，一歩進むと傾きも変わります．だから少し進んでは測り直します．

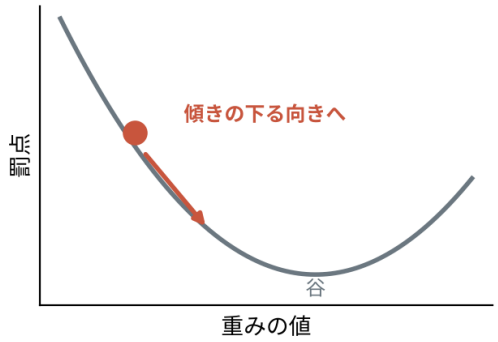


図 18　勾配は足もとの坂の傾き．

勾配を実際に確かめる　lm_head の i の行の3番目の重み（学習前の値 0.1090）を，0.001 だけ増やしてみます．mia の損失は 1.953475 から 1.953936 に，0.000461 増えました．損失の変化 ÷ 重みの変化 $= 0.000461 \div 0.001 \approx 0.461$ が，この重みの勾配です．プログラムが自動で計算した勾配も 0.461 で，一致します．勾配が正なので，この重みは減らせば損失が下がります．

式で書くと　$$\frac{\partial L}{\partial w} \approx \frac{L(w+\Delta) - L(w)}{\Delta} = \frac{1.953936 - 1.953475}{0.001} = 0.461.$$

ただし，268個を1つずつ動かして確かめていたら，順伝播が268回余分に必要です．大きな AI には何千億個も重みがあります．そこで，計算のつながりを逆にたどって，全部の勾配を一度に求めます．

4.4　勾配を一度に求める（逆伝播）

小さな例：$f = a \times b + a$　$a = 2$，$b = 3$ として，$c = a \times b = 6$，$f = c + a = 8$ を計算します（図19左）．プログラムは計算をするたびに，「どの数とどの数から，どんな計算で作ったか」を記録しておきます．次に，f から出発して記録を逆にたどります（図19右）．足し算はそのまま伝え，掛け算は「相手の値」倍して伝えます．a には2つの道から 1 と 3 が届くので，合わせて 4 です．b は $1 \times a = 2$ です．

確かめると，a を 0.01 増やすと f は 8.04 になり（0.01 の4倍），b を 0.01 増やすと 8.02 になります（0.01 の2倍）．逆にたどって求めた 4 と 2 は，「その数を少し増やすと，f がその何倍増えるか」を表しています．microgpt では f が損失で，a や b が268個の重みにあたります．

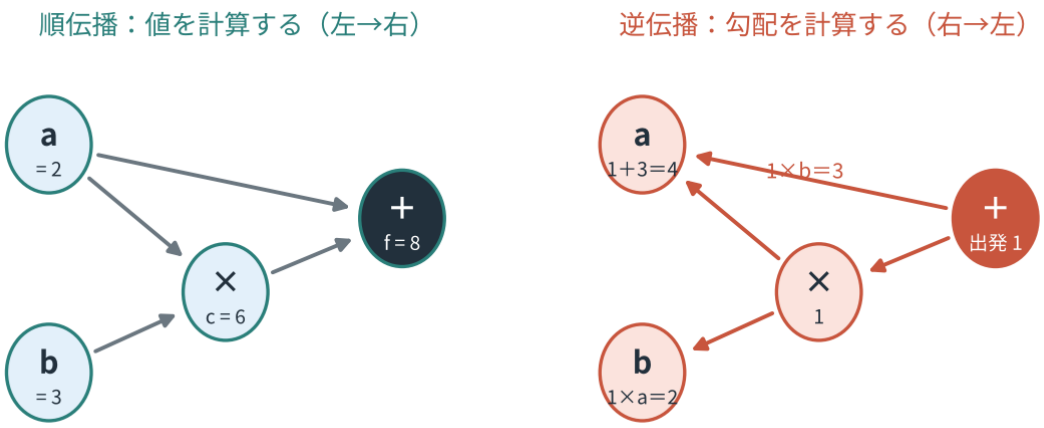


図 19　$f = a \times b + a$ の計算の記録．左：順伝播で値を計算する．右：逆伝播で勾配を計算する．

連鎖律：歯車の回転比を掛けていく　重みを動かすと黒板の数が変わり，黒板が変わると得点が変わり，得点が変わると確率が変わり，確率が変わると損失が変わります．影響は歯車のように順に伝わって損失に届きます．重みを1回すと損失が何回るかは，となりどうしの歯車の回転比をすべて掛け合わせれば分かります．これを微分の**連鎖律**（chain rule）と呼びます．逆伝播は，損失の側から出発して，回転比を掛けながら1段ずつさかのぼります．途中の結果を使い回せるので，268個の勾配が1回のさかのぼりでそろいます[34]．

> **式で書くと**　$\dfrac{\partial L}{\partial w} = \dfrac{\partial L}{\partial y}\dfrac{\partial y}{\partial w}$（連鎖律）．道が複数あれば各道の積を足す：$\partial f/\partial a = b + 1 = 4$.

得点の勾配は「確率 − 正解」　さかのぼりの最初の一歩は，とても簡単な形になります．各文字の得点の勾配は，その文字の確率 −（正解なら 1，ちがえば 0）です（表15）[35]．正解の m の得点は勾配が負なので，上げる向きへ，ほかの6文字は正なので，下げる向きへ動かされます．4問の平均をとるので，最後に 4 で割ります．

表 15　mia の1問目（始まり → 正解 m）の7つの得点の勾配（学習前）

文字	確率	正解なら 1	確率 − 正解	÷4（問題の数）
a	12.7%	0	+0.127	+0.032
e	15.2%	0	+0.152	+0.038
i	15.3%	0	+0.153	+0.038
m	15.4%	1	−0.846	**−0.211**
n	14.4%	0	+0.144	+0.036
v	13.8%	0	+0.138	+0.034
終わり	13.3%	0	+0.133	+0.033

[34] この方法を誤差逆伝播法（バックプロパゲーション，backpropagation）と呼びます．microgpt では `Value` という部品が，計算のたびに「何と何から，どんな計算で作ったか」と回転比（局所的な微分）を記録します（29–72行目）．このしくみを自動微分（automatic differentiation）と呼びます．

[35] 確率と罰点（− ln）の組み合わせの微分がちょうど打ち消し合って，この形になります（付録A.7）．

式で書くと $\dfrac{\partial(-\ln p_c)}{\partial z_j} = p_j - y_j$（$y_j$：正解なら 1，ちがえば 0）．7つの勾配の和はいつも 0.

1回目の勾配：どの表が大きく効いたか　表ごとに勾配の大きさを比べると（図20），いちばん大きいのは lm_head（得点を出す表）です．損失にいちばん近い歯車だからです．Attention の表（attn_wq など）の勾配はほぼ 0 でした．学習前は W_V や W_O の値が小さく，どの位置を見ても書き足す量がわずかなので，見る位置を変えても損失がほとんど変わらないためと考えられます．

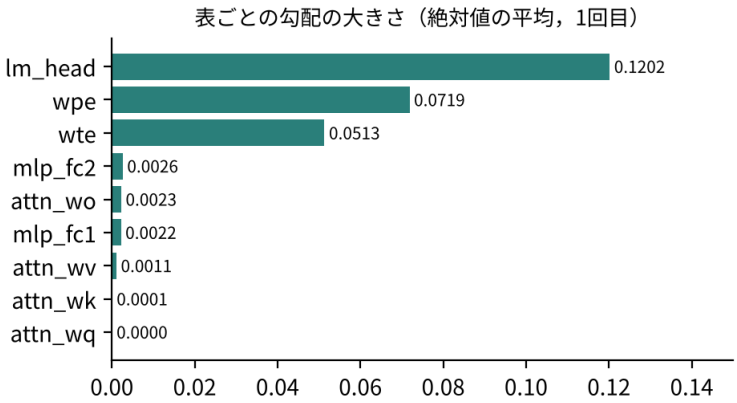


図 20　表ごとの勾配の大きさ（1回目）．

勾配が 0 になる行：wte と lm_head のちがい　268個のうち16個は，勾配がちょうど 0 でした．mia に出てこない文字 e，n，v の wte の行（12個）と，使わなかった5番目の位置 p4 の wpe の行（4個）です．wte と wpe は「行を引く」だけの表なので，引かなかった行は損失に影響せず，勾配が 0 になります．

反対に lm_head は，7つの行すべてに毎回勾配がつきます．lm_head は7つの行すべてを黒板に掛けて，7つの得点を出すからです．正解の文字の行は「この黒板のときに得点が上がる」向きへ，ほかの6行は「下がる」向きへ動かされます．名前に出てこない文字の行も，「ここでは出すな」と毎回押し下げられます．wte の行は「その文字を読んだとき」だけ動き，lm_head の行は毎回動きます（付録A.8）．

重みの使い回しと勾配　2章で見たように，同じ表をすべての位置で使います．だから1つの重みの勾配は，すべての位置から届いた分の足し算になります．emma の m の行の勾配には，p2 から届いた分と p3 から届いた分の両方が入ります．

4.5　重みを更新する（Adam）

基本：勾配と反対の向きに，少しだけ動かす　勾配が正なら減らし，負なら増やす．これを**勾配降下法**（gradient descent）と呼びます．一度に大きく動かしすぎると，谷を飛び越えてしまいます．だから少しずつ動かして，測り直します．1回に動かす幅の目安を**学習率**（learning rate）と呼びます．microgpt では 0.01 から始めて，300回目に 0 になるよう少しずつ小さくします．最初は大また，谷に近づいたら小またで歩くのです．

Adam の2つの工夫　microgpt は，勾配降下法を改良した Adam（アダム）という方法を使います[36]．

1. **勾配の平均を使う（勢い）**：1回の学習は1つの名前だけなので，勾配は名前ごとにばらつきます．最近の勾配の平均を使って，ばらつきをならします．坂を転がるボールの勢いのように，小さなでこぼこでは向きを変えません．
2. **動かす幅をそろえる**：重みによって勾配の大きさは何桁もちがいます．それぞれの勾配の大きさで割り，どの重みも同じくらいの幅で動くようにします．急な坂でも緩い坂でも，同じ歩幅で歩くようなものです．

> **式で書くと**　$w \leftarrow w - \eta \dfrac{(\text{勾配の平均})}{\sqrt{(\text{勾配の2乗の平均})}}$．学習率 η は 0.01 から 0 へ．くわしくは付録 A.9.

1回目：どの重みもほぼ 0.01 ずつ動いた　表16を見ると，勾配が正の重みは減り，負の重みは増えています．どれも損失を下げる向きです．勾配が 0.46 でも 0.28 でも，動いた幅はほぼ同じ 0.01 です．Adam の「幅をそろえる」工夫で，1回目はどの重みも学習率（0.01）だけ動きます[37]．勾配が 0 の重み（e の wte など16個）は動きません．

表 16　1回目の重みの動き

重み	学習前の値	勾配	1回目の後	動いた幅
lm_head（i の行の3番目）	0.1090	+0.461	0.0990	−0.0100
lm_head（a の行の2番目）	0.0940	−0.438	0.1040	+0.0100
lm_head（m の行の4番目）	−0.0270	−0.390	−0.0170	+0.0100
wte（a の行の1番目）	0.0014	+0.282	−0.0086	−0.0100
wte（e の行の1番目）	−0.1026	0	−0.1026	0

1回目の後：mia は少しよくなった．ほかの名前は？　mia の4問は，どれも正解の確率が少し上がり，損失は 1.953 から 1.889 に下がりました（表14右）．ほかの名前の損失も調べると，mia と同じく「a で終わる」ava は 1.971 から 1.938 に下がりましたが，emma は 1.963 から 1.979 に上がり，anna は 2.001 から 2.002 とほぼ同じでした．

> 1つの名前に合わせた調整が，ほかの名前にもよいとは限りません．だから名前を替えながら何度もくり返し，全体として損失が下がる場所を探します．

4.6　300回の学習

損失の変化：ぎざぎざと，なめらか　図21の灰色は，その回に使った1つの名前の損失です．名前ごとに難しさがちがうので，ぎざぎざします[38]．緑は，そのときの重みで4つの名前すべ

[36] adaptive moment estimation（適応的モーメント推定）に由来する名前で，2014年に Kingma と Ba が発表しました．モーメントは統計の言葉で，1次のモーメントは平均，2次のモーメントは2乗の平均です（付録 A.9）．

[37] 1回目は勾配の平均も2乗の平均も，その1回の勾配だけから作られるので，「勾配 ÷ 勾配の大きさ」＝ ±1 になります．勾配がとても小さい重みは，0 で割らないための小さな数の影響で，0.0099 くらいになります．

[38] 同じ名前をくり返し学ばせても，重みは300回目まで変わり続けます．1回の学習では1つの名前だけを使い，その名前の損失が下がる向きに重みを動かすので，次の回では別の名前に合わせてまた動きます．名前の最初の文

て（18問）を解いた平均です．1.97 から 0.347 まで，なめらかに下がりました．50回目あたりまでに急に下がり，あとはゆっくり 0.308 の線に近づきます[39]．「何択で迷っているか」に直すと，7.2択から 1.41択になりました．正解につける確率に直すと，平均で約 71% です．

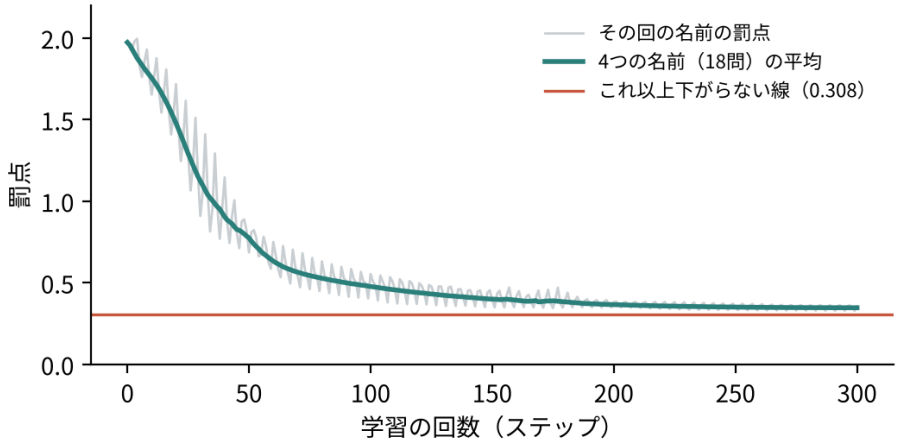


図 21　学習の回数と損失．

AI は覚えやすいものから覚えていく　答えが一つに決まる場面の，正解の確率の変化を見ます（図22左）．名前の終わり（mia → 終わり）は早く覚えました．60回目で 86.7% です．m → i はなかなか覚えません．60回目で 21.0%，300回目でも 80.2% です．「名前の最後の a のあとは終わり」は4つの名前すべてに共通ですが，「m の次が i」は mia だけの癖です．たくさんの名前に共通する規則ほど，学習の機会が多く，早く覚えています．

字や a の次はデータ自体があいまいなので，名前ごとに引っぱる向きがちがいます（emma の回は「始めは e」へ，ava の回は「始めは a」へ）．学習率は300回目で 0 になるので，動きはだんだん小さくなって最後は止まります．

[39] 0.308 は，始まりの次と a の次で「正しく迷う」ことから来る限界です（1章，付録A.1）．

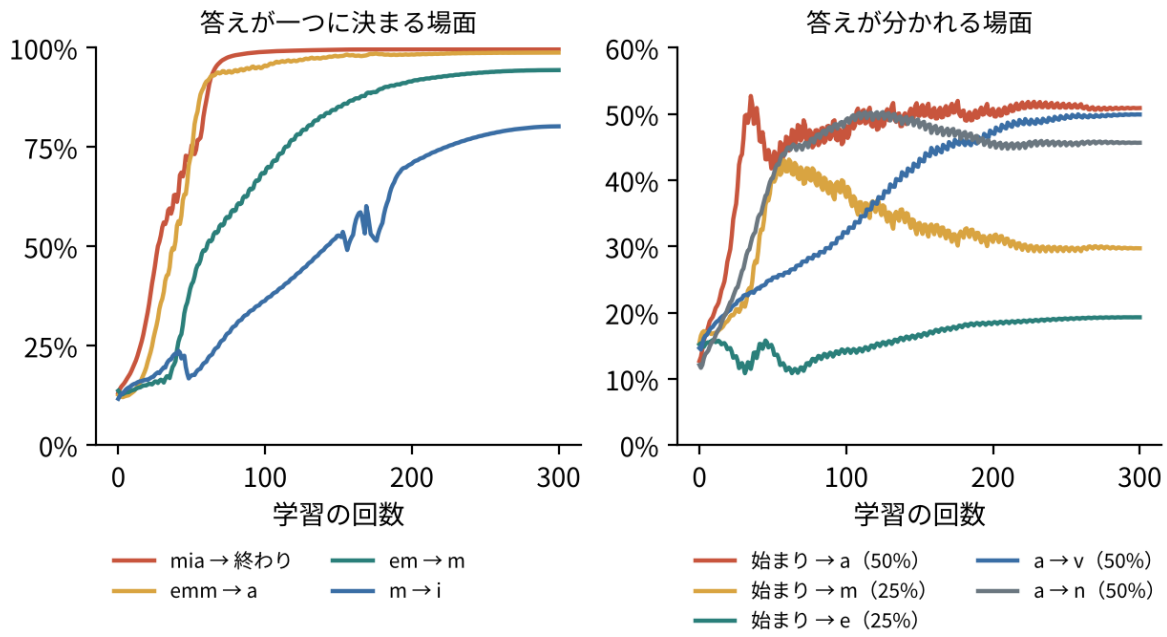


図 22　正解の確率の変化．左：答えが一つに決まる場面．右：答えが分かれる場面（かっこ内はデータの割合）．

迷う場面は，データの割合に落ち着く　答えが分かれる場面では（図22右），始まりの次の m は，60回目ごろにいったん 43% まで上がりすぎ，そのあと 30% 前後に戻りました．a の次は，はじめ n が先に伸び，v はあとから追いついて，300回目には v 49.9%，n 45.6% とほぼ半々になりました．どちらも，最後はデータの割合に近いところに落ち着きます．

i と v は，学習の中で同じ向きになった　wte の i の行と v の行の向きのそろい方（cos）は，学習前は 0.64（偶然の向き）でした．10回目で 0.91，50回目には 1.00 になり，そのまま変わりませんでした（図23）．i（mia）と v（ava）は，どちらも「次は必ず a」という同じ役目です．同じ向きに置けば，同じ規則を使い回せます．だれも「i と v を似せよ」とは教えていません．損失を下げる向きに重みを動かしていたら，自然にそうなりました．

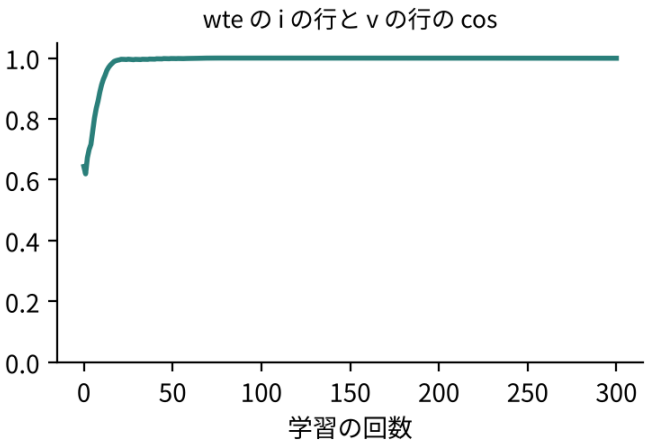


図 23　wte の i の行と v の行の向きのそろい方の変化．

300回で，重みはどれだけ動いたか　表ごとに，学習前から学習後までの変化を比べると（図24），最初はほとんど勾配がなかった Attention の表も，300回の間に大きく動きました．MLP の型紙（W_1 の16本の行）の長さは，学習前の 0.03〜0.21 から，0.74〜1.27 に育ちました．いちばん動きが小さかったのは wte と wpe でした[40]．

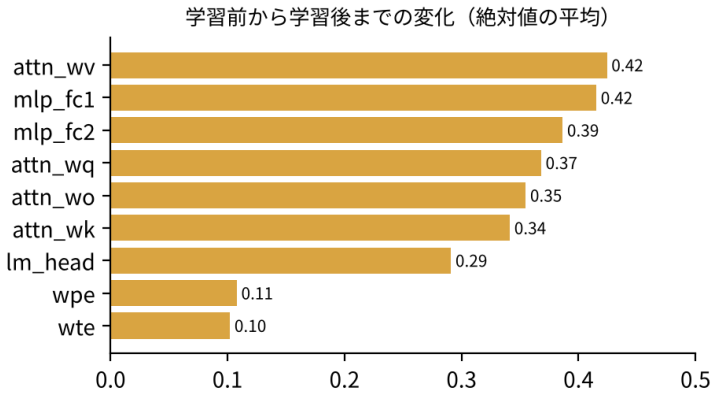


図 24　表ごとの，学習前から学習後までの変化（絶対値の平均）．

4.7　損失が低い ＝ 賢い？

向きとしては合っています．損失が低いほど，予想がうまいということです．ただし，損失が低いのは「学習した名前をよく当てられる」ということです．この AI は4つの名前をほぼ覚えきりましたが，それは丸暗記に近い状態です．本当の実力（見たことのない名前への対応力）を知るには，学習に使わなかった名前で損失を測る必要があります．また，ちがう問題どうしの損失は比べられません．問題によって「これ以上下がらない線」がちがうからです．

大きな AI では，初めて見る文章での損失が下がるほど，いろいろな能力も上がる傾向があります．そのため損失は「賢さ」の目安に使われますが，あくまで平均点です．テストの点も同じで，問題集を丸暗記すれば満点がとれますが，初めて見る問題が解けるかどうかは別の話です．

4.8　学習とは，最小二乗法と同じ考え方

理科の実験で，ばねにおもりをつるして伸びを測り，グラフに直線を引くとき，私たちは「データの点と直線のずれ」がなるべく小さくなるように，直線の傾きと切片を決めます（最小二乗法）．microgpt がしていることも本質的には同じです（表17）．

[40]学習前の値は ±0.16 くらいなので，平均 0.3〜0.4 の変化は，最初の値よりずっと大きな動きです．wte と wpe は，その行を引いた回にしか勾配がつかないことも，動きが小さい理由の一つと考えられます．

表 17　最小二乗法と microgpt の学習

	ばねの実験（最小二乗法）	microgpt
動かす数	直線の傾きと切片（2個）	9つの表の重み（268個）
外れ具合（損失）	点と直線のずれの2乗の合計	正解につけた確率の罰点の平均
学習	ずれが最小になる傾きと切片を求める	損失が小さくなる重みを探す
求め方	式で一度に解ける	式では解けないので，勾配を見て少しずつ

AI の「学習」は，人間の学習とは意味がちがいます．損失が小さくなるように重みを調整すること，それ以上でも以下でもありません．それでも i と v が同じ向きになるような「整理」が，結果として生まれます．

4章のまとめ：外れ具合は「− ln（正解の確率）」の平均で測ります．学習前は当てずっぽうと同じ 1.953 で，300回の学習で 0.347（限界 0.308）まで下がりました．勾配は，計算の記録を逆にたどって（逆伝播）一度に求めます．得点の勾配は「確率 − 正解」というやさしい形になります．wte と wpe は引いた行だけに，lm_head はすべての行に毎回勾配がつきます．Adam で少しずつ更新すると，AI は覚えやすいものから順に覚え，i と v は自然に同じ向きになりました．

5　学習後の表を読む

3章では学習後の重みで計算をたどり，4章ではその重みができるまでを見ました．この章では，学習後の 268 個の重みから，AI が何を覚えたのかを読み解きます．マスの数を一つずつ読むのではなく，行の向きの似かた（cos），行の長さ，Attention の割合，検出器の点灯，得点への寄与で読みます．

> **先に答えを言うと**：AI は名前を丸ごと暗記するのではなく，「こういうときは次にこの文字」という小さな規則をたくさん重ね合わせて覚えていました．

5.1　wte：文字の「住所」を決める表

wte は，各文字に4つの数を割り当てる表です．4つの数は，4次元の空間の中の「住所（座標）」と考えられます．図25は，4次元の住所を2次元の地図に写したものです[41]．i と v がほぼ同じ場所に重なっています．mia の i も ava の v も，次は必ず a．役目が同じなので，同じ住所に置かれました．a だけはほかと遠く離れています．a の次は v，n，終わりとばらばらだからです．

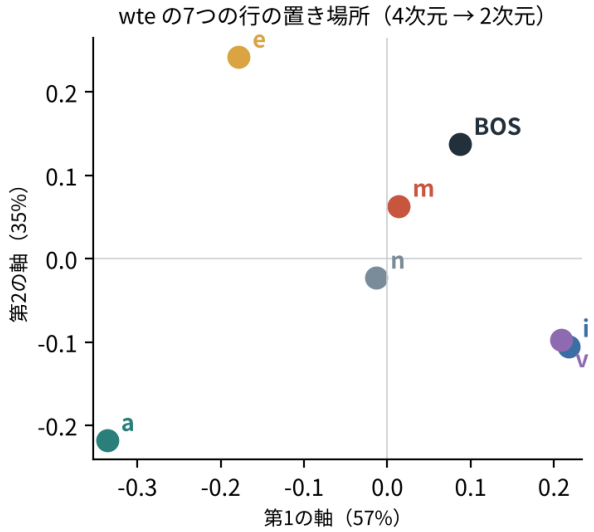


図 25　7つの文字の置き場所（4次元を2次元に写したもの）．

近さは「向き」で測る　文字どうしの近さは，住所の距離よりも，矢印の「向き」がどれだけそろっているかで測るのが自然です．使われる前に，4つの数の大きさがそろえられるからです（3章の rmsnorm）．i と v の cos は +1.00 でした（図26）．近さの基準は，綴りや音の似かたではなく，予想するときの役目の似かたです．この物差しは最初から決まっているのではなく，学習の中で作られたものです（4章）．

[41] 主成分分析という方法で，点のばらつきがいちばん大きく見える平面に写しています．2つの軸で，4次元のばらつき全体の約 92%（57%＋35%）を表せています．

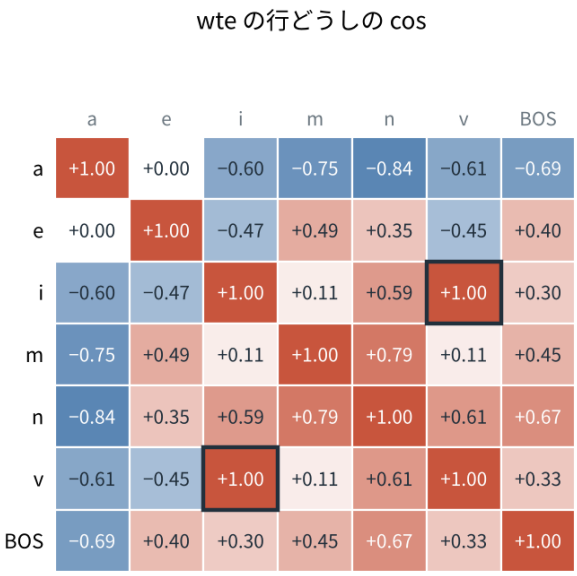


図 26　wte の行どうしの cos．枠は i と v．表は対角線をはさんで対称なので，ある文字の行を横に読めば，ほかのすべての文字との近さがわかる．

式で書くと　$\cos\theta = \dfrac{\boldsymbol{a}\cdot\boldsymbol{b}}{|\boldsymbol{a}|\,|\boldsymbol{b}|}$．内積は向きと長さの両方で決まるので，向きだけを比べるには長さで割る．

wte は「文字 → 4つの数」の辞書　wte の表の左に並ぶ a，e，i，m，n，v，BOS は，文字の種類の名札です．名前の文字を順に並べたものではありません．AI が emma を読むときは，e，m，m，a の順に，この辞書から行を引き出して使います（2章の表8）．m は2回出てくるので，m の行を2回引きます．

5.2　wpe：何文字目かを教える表

同じ m を読んでも，次の文字は変わります（表18）．これを見分けられるのは，wpe が「何文字目か」を教えているからです．

表 18　同じ m を読んだときの予想

読んだところ	何文字目の m	正解	AI の予想
m	1文字目	i	i（80%）
em	2文字目	m	m（94%）
emm	3文字目	a	a（99%）

図27は，wte の各行の長さです．学習の後，n の行はほとんど長さ0に縮みました（0.16 → 0.03）．n は文字だけでは次が決まらないので（an の次は n，ann の次は a），位置の情報にまかせたのです．次が一つに決まる i，v，e の行は長く育ちました．

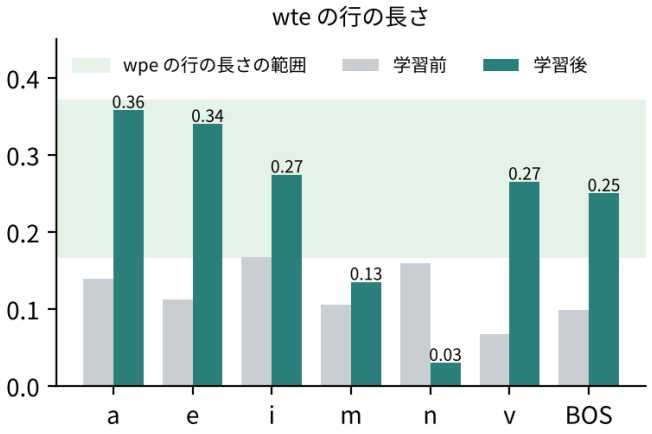


図 27　wte の行の長さ（学習前と学習後）．緑の帯は wpe の行の長さの範囲.

学習後，wpe の p3 と p4 の行はよく似ていました．p3 は $(-0.22, +0.03, +0.10, -0.28)$，p4 は $(-0.18, -0.01, +0.09, -0.31)$ です．どちらも名前の最後の a を読む位置です（3文字の名前は p3，4文字の名前は p4）．

5.3　d0〜d3 は成分だが，軸の向きは決まっていない

表の横に並ぶ d0〜d3 の d は dimension（次元）の頭文字で，黒板の4つの欄，つまり4次元の空間の4本の軸です．wte と wpe の軸はそろっているので，同じ欄どうしを足せます.

ただし「どの向きを軸に選ぶか」は決まっていません．風を東・北の成分で表しても，45° 回した軸で表しても，風そのものは同じです．成分の値だけが変わります（図28）．AI でも同じで，軸を回してすべての表をそれに合わせて書きかえても，答えはまったく変わりません（付録A.10）．

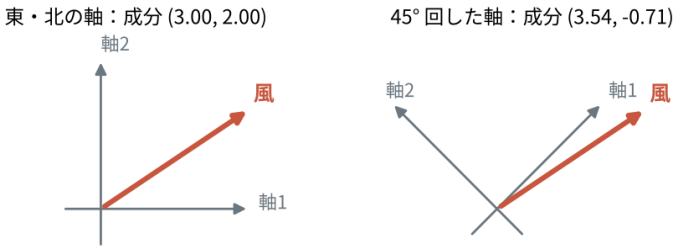


図 28　同じ風のベクトルを，向きのちがう2組の軸で表す.

> 風の「東」「北」には人間が決めた意味がありますが，d0〜d3 にはそういう意味を誰も割り当てていません．「a の d1 が −0.21」という一つの値を取り出しても意味は読めません．読めるのは，向きの似かた（cos）とベクトルの長さです.

5.4　Attention（wq, wk, wv, wo）：ほかの位置を見に行く

図29は，4つの名前を読んだときに，各位置がどの位置をどれだけ見たかの割合です（各行の合計が1）．次が a になる場面（mi，av，ann，emm）では，名前の始まり（BOS）をよく見ています（0.76〜0.84）．1文字目の位置では，ほぼ自分自身を見ます（0.78〜1.00）．名前の最後の a では，見る先がほぼ均等に分かれます.

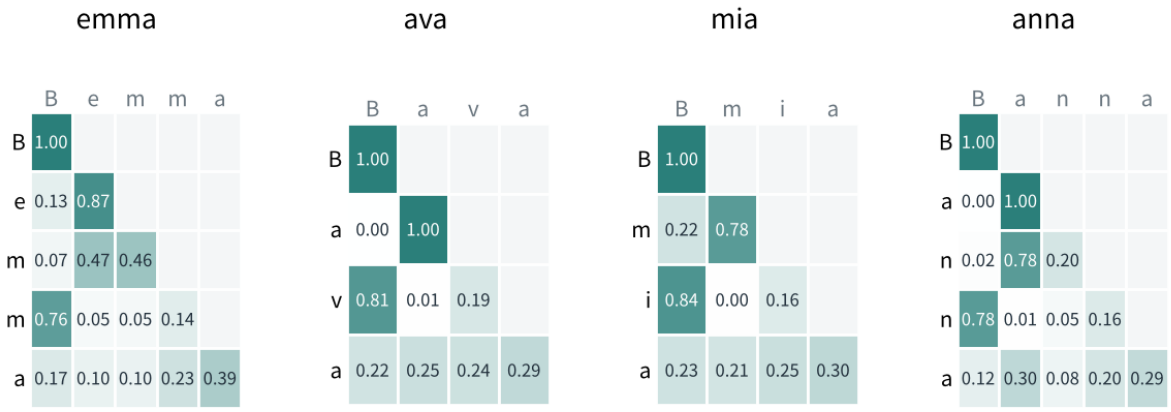


図 29　4つの名前を読んだときの Attention の割合．縦が読む位置，横が見に行く位置（B は BOS）．灰色はまだ読んでいない位置．

図30は，Attention が各文字の得点に足した分です．枠はデータ上の正解です．目立つのは a を読んだ場面（n に +2.5，v に +2.4），an（n に +2.1），e と em（m に +1.7）です．Attention が足す分は最大でも 2.5 ほどで，得点の大部分は，次の MLP が足しています．

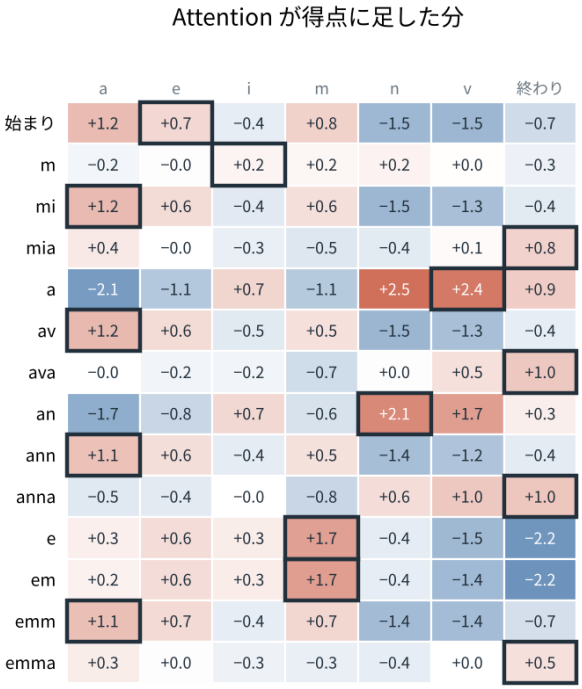


図 30　Attention が各文字の得点に足した分（14の場面）．枠はデータ上の正解．

W_Q と W_K，W_V と W_O は，2つの表の組み合わせ（積）だけが意味を持ちます．片方だけを見ても読めません（この章の最後と付録A.10）．

5.5　wte と lm_head：入口の名札と出口の型紙

wte と lm_head は，どちらも「文字ごとに4つの数」が並んだ同じ形（7×4）の表です．でも役目はちがいます．

- **wte の行は入口の名札**です．「この文字を読んだら，黒板にこう書く」．読むときに1行だけ引きます．
- **lm_head の行は出口の型紙**です．「最後の黒板がこの向きなら，この文字の得点を上げる」．毎回7行すべてを黒板に掛けます．

役目がちがうので，同じ文字の行どうしでも向きはそろっていません（図31）．m は +0.74 と同じ向きに近く，a は −0.76 と反対向きに近く，ほかはまちまちです．wte の a の行は「a を読んだら次は v か n か終わり」という状態を書き，lm_head の a の行は「次に a を出すとき」の黒板を待っています．同じ a でも，求められる向きがちがうのです．

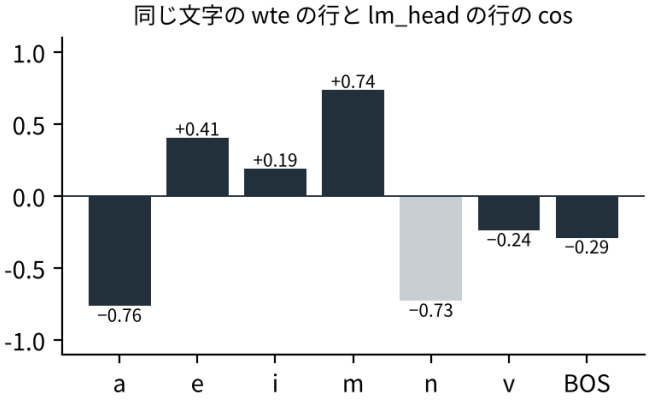


図 31　同じ文字の wte の行と lm_head の行の cos．n は wte の行がほぼ 0 なので参考値．

学習での動き方もちがいます（4章）．wte の行は，その文字を読んだ回にしか勾配がつきません．lm_head の行は，正解の文字の行が引き上げられ，ほかの行が押し下げられるので，毎回すべての行が動きます．

> 本物の GPT-2 では，wte と lm_head を同じ1つの表で兼用しています（weight tying）．入口の名札と出口の型紙を同じにすると重みの数が減ります．microgpt では別々の表にしているので，入口と出口がそれぞれどう育つかを比べられます．

5.6　MLP（fc1, fc2）：16個の検出器

MLP という名前の由来　MLP は multi-layer perceptron（多層パーセプトロン）の略です．パーセプトロンは，1958年に Rosenblatt が考えた人工ニューロンのモデルで，入力に重みをかけて足し合わせ，しきい値を超えたら反応します（perceive，知覚する，が語源）．それを層に重ねたものが多層パーセプトロンで，microgpt では 4 → 16 → 4 と，2段の表（fc1，fc2）でつながります．fc は fully connected layer（全結合層）の略で，すべての入力がすべての出力につながる，ふつうの行列の掛け算です．元の Transformer の論文（2017年）では feed-forward network と呼んでいます．

なぜ「検出器」と呼ぶのか

1. **型紙との似かたを測る**：今の黒板と型紙（W_1 の行）の内積で，向きがどれだけそろっているかを測ります．
2. **しきい値を超えたら反応する**：似かたが負なら 0（黙る），正ならその大きさを出します（ReLU）．

3. **何を検出するかは，あとからわかる**：何に反応するかは学習の結果で決まります．名前は，点灯の様子を見て私たちがあとから付けたものです．

神経細胞のモデル（重みつきの合計と発火のしきい値）や，地震計のトリガーと同じ形です．

> **式で書くと**　$h_i = \max(0,\ \boldsymbol{\kappa}_i \cdot \hat{\boldsymbol{x}})$．$\boldsymbol{\kappa}_i$ は W_1 の i 行目（型紙（4））．$|\hat{\boldsymbol{x}}| = 2$ なので $\boldsymbol{\kappa}_i \cdot \hat{\boldsymbol{x}} = 2|\boldsymbol{\kappa}_i| \cos\theta_i$．

16個の検出器の点灯　図32を見ると，次が a の場面（mi，av，ann，emm）では，14〜15個が一斉に点灯します．名前の終わり（mia，ava，anna，emma）では，h0，h1，h2，h5，h9，h10，h13，h14 の8個がそろって点灯し，「終わり」の得点を +7.5〜+10.3 押し上げていました．

16個の検出器の点灯 h（14の場面．空欄は 0）

	h0	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10	h11	h12	h13	h14	h15
始まり				1.2	1.4		1.5	1.3	0.8			1.3				1.4
m					0.1		0.2						2.0			0.6
mi	1.1	1.0	1.0	1.5	1.5	0.8	1.7	1.1	1.4	1.0	1.2	1.4		0.7		1.5
mia	1.8	2.1	1.4	0.1		1.4			0.5	1.9	1.8			1.7	1.7	
a													1.7		0.7	
av	1.1	1.0	1.0	1.5	1.5	0.8	1.7	1.1	1.4	1.0	1.2	1.4		0.7	0.0	1.4
ava	1.6	1.9	1.2			1.2			0.1	1.7	1.6			1.5	1.7	
an													1.9			
ann	1.2	1.1	1.1	1.5	1.4	0.9	1.6	0.9	1.4	1.2	1.3	1.3		0.8	0.1	1.3
anna	1.3	1.6	0.9			0.9				1.3	1.2		0.1	1.2	1.6	
e				0.3	1.4		1.4	1.7				1.0	0.5			1.5
em				0.3	1.4		1.4	1.7				1.0	0.5			1.5
emm	0.8	0.7	0.8	1.5	1.7	0.7	1.9	1.2	1.4	0.8	1.0	1.5		0.5		1.6
emma	1.8	2.1	1.4			1.3			0.4	1.8	1.8			1.6	1.7	

図 32　16個の検出器の点灯 $\boldsymbol{h}$（ReLU を通った値．空欄は 0）．

型紙は学習で作られる　型紙（W_1 の16本の行）は，最初は小さな乱数で，長さは 0.03〜0.21 でした．学習の後は 0.74〜1.27 まで育ちました（図33）．向きも大きく変わりました．「この場面で反応すると損失が減る」向きへ，少しずつ調整された結果です．

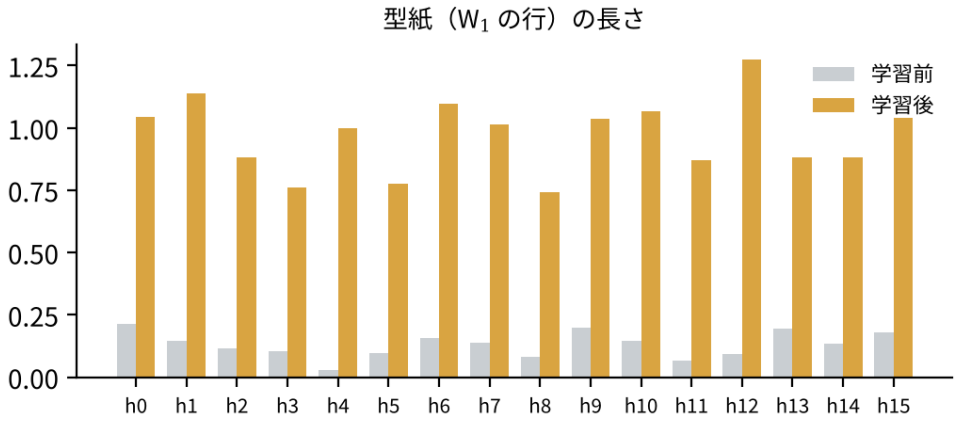


図 33　型紙（W_1 の行）の長さ．学習前と学習後．

型紙の例：何に反応して，何を足すか　型紙の4つの数（d0〜d3）をそのまま読んでも意味はわかりません．中身は「どの場面で点灯し，点灯すると何を足すか」で読みます（表19）．mia を読み終えた場面では，MLP は「終わり」の得点を 10.33 押し上げ，そのうち 10.28 が上の8個から来ていました．一つの型紙が一つの規則にきれいに対応するとは限りません．似た型紙が組になって働き，最後の区別は Attention などとの足し算で決まります．

表 19　型紙の例

検出器	点灯する場面（読んだところ）	点灯すると	名づけるなら
h12	m，a，an，e，em など	i と n を足し，a と終わりを減らす	「名前の前半」
h0，h1，h2，h5，h9，h10，h13	mi，av，ann，emm と mia，ava，anna，emma	終わりと a を足し，n と m を減らす	「最後の a のあたり」
h14	主に mia，ava，anna，emma	終わりを足す	「名前が終わった」
h4，h6，h7，h15	始め，e，em，emm など	m を足し，終わりを減らす	「emm の m」の手助け

検出器は「ちょうど半分」に反応する　microgpt の検出器には，しきい値をずらす数（バイアス）がありません．計算を簡単にするための選び方で，大きな AI でも使われる方法です[42]．そのため，反応する・しないの境目は必ず原点を通り，大きさをそろえた黒板（円）をちょうど半分に分けます（図34）．実際に，16個の検出器はどれも，14の場面のうち 6〜8個（ほぼ半分）で点灯していました．一つ一つは「半分」しか選べない粗い検出器ですが，いくつかが同時に点灯する重なりで，特定の場面を絞り込みます．

[42] PaLM や LLaMA などの大きなモデルもバイアスを外しています．

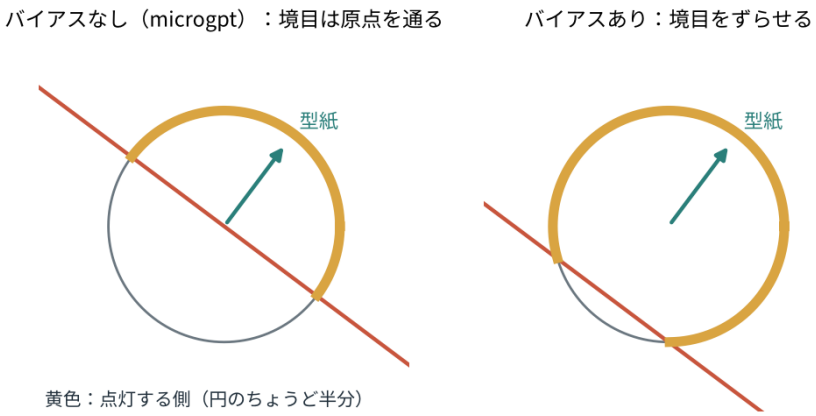


図 34　バイアスがないと，境目は原点を通り，円をちょうど半分に分ける（左）．図は2次元で描いたが，実際は4次元.

なぜ 4 から 16 に広げるのか　「4倍」は元の Transformer や GPT-2 から引き継いだ慣習で，理論的な決まりではありません．検出器が1つ増えるごとに，「反応する・しない」の境目が1本増えます．境目が多いほど，場面を細かく分けて，ちがう規則を使い分けられます．4次元の空間を原点を通る境目で分けると，検出器4個なら16区画，16個なら1152区画に分けられます（付録A.11）．今回の場面は14しかないので，区画のほとんどは使われていません．

ReLU（反応する・しないの折り曲げ）がなければ，広げても意味がありません．W_1（16×4）と W_2（4×16）の2つの掛け算は，1つの 4×4 の掛け算と同じになってしまうからです.

16次元は「場合分けの作業場」　14の場面の数の並びが，どれだけの次元に広がっているかを調べると，表20のようになりました．W_1 を掛けただけでは，16個の数は元の4つの数から作った1次式にすぎず，16次元の中の4次元の平らな面の上にしか乗りません．ReLU がこの面を検出器の境目ごとに折り曲げて，初めて16次元の広さを使うようになります．平面の上では1本の直線で分けられない点も，一段高い次元に持ち上げて折り曲げれば分けられます（XOR の問題と同じ考え）.

表 20　14の場面の点が広がっている次元

段階	数の並び	点が広がっている次元
W_1 を掛けた直後（ReLU の前）	$\boldsymbol{u}$（16）	4
ReLU を通った後	$\boldsymbol{h}$（16）	13（14の場面で，ほぼ最大）
W_2 で黒板に戻した後	（4）	4

5.7　9つの表の役目のまとめ

表 21　9つの表の役目のまとめ

表	役目	見つける係か，覚える係か
wte, wpe	文字と位置の「住所」	次の文字の手がかりとして整理された表
wq, wk	どこを見るか	見つける係（位置どうしの関係）
wv, wo	何を写してくるか	情報を位置の間で運ぶ係
fc1	16個の検出器	見つける係（今の状態の特徴）
fc2	反応したら書き足す内容	fc1 とペアで覚える係
lm_head	得点への読み取り	見つける係（黒板を答えに変える）

いちばん「記憶」に近いのは，fc1（検出器）と fc2（書き足す内容）のペアです．

5.8　分けて理解できるか

AI のはたらきは，部品ごとに分けて理解できるでしょうか．分けられるところと，分けられないところがあります．

分けられるところ：得点は3つの足し算　3章で見たように，最後の得点は「最初のメモ ＋ Attention ＋ MLP」の3つにちょうど分けられます．図35は，14の場面すべての内わけです．たとえば「a」まで読んだ場面では，データでは次は v（ava）か n（anna）が半々で，AI も v と n をほぼ同じ確率（0.50 と 0.46）にして，「どちらもありうる」と正しく答えています．

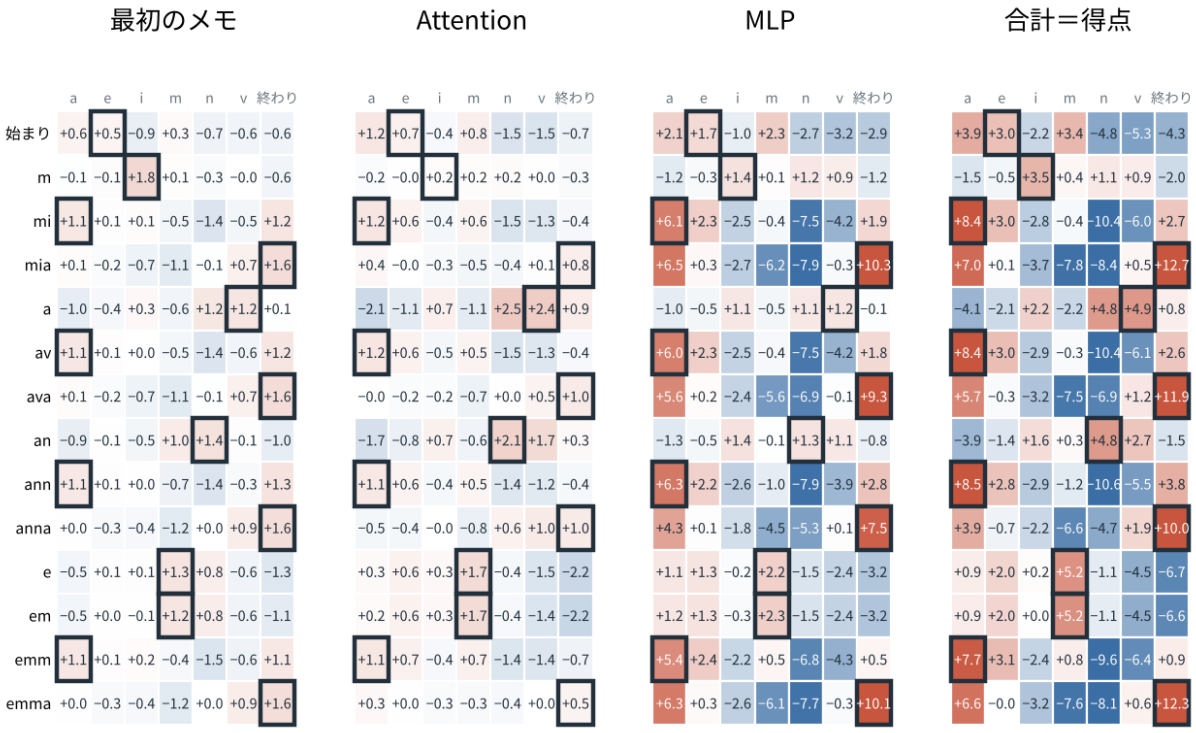


図 35　14の場面すべての得点の内わけ．左から，最初のメモ，Attention，MLP，合計（＝得点）．枠はデータ上の正解．

分けられないところ：一つ一つのマスには意味がない　表の数を，ある決まりで回転させると，数はがらりと変わります．でも，AI のはたらきはまったく同じです（図36）．地図の向きを変えると，町の座標は変わっても，町どうしの距離は変わらないのと同じです．意味があるのは数どうしの関係で，一つのマスの値ではありません．

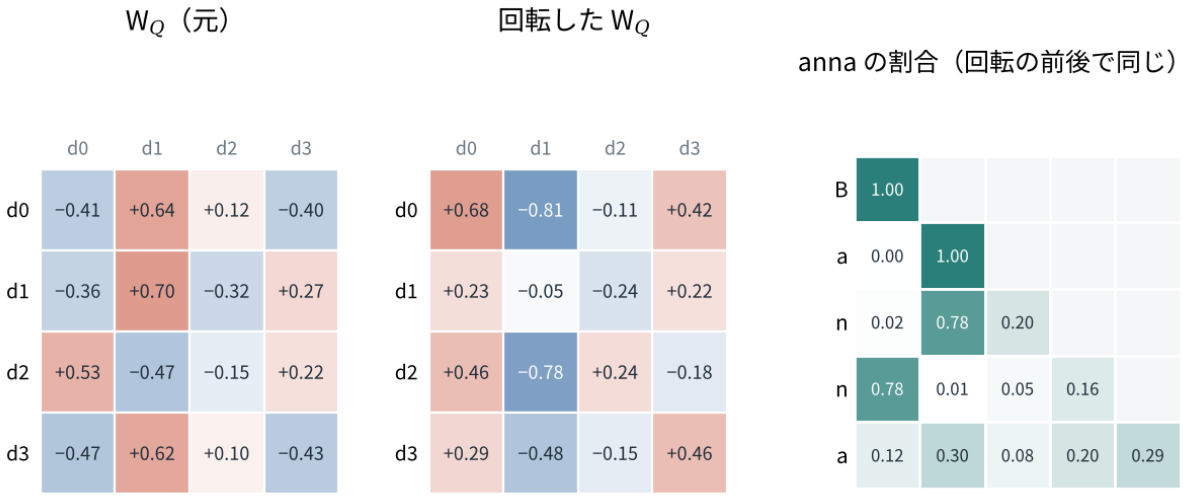


図 36　W_Q と W_K を同じ回転で書きかえる（左と中）と，数はすっかり変わるが，Attention の割合（右）はまったく同じ．

例外：MLP の16本の軸だけは一つずつ読める　黒板の4本の軸（d0〜d3）は回して書きかえても働きが変わりませんが，MLP の16本の軸（h0〜h15）は回せません（表22）．ReLU が軸1本ずつに働くからです．だから「h12 が点灯した」とは読めますが，「d2 が負だ」には意味がありません．

表 22　2種類の軸のちがい

d0〜d3（黒板の4本の軸）	h0〜h15（MLP の16本の軸）
wte，wpe，lm_head，wq，wk，wv，wo，fc1 の列，fc2 の行	fc1 の行，fc2 の列
回して書きかえても働きは同じ	ReLU が1本ずつ働くので回せない（回すと得点が最大 4.6 変わった）
一つ一つのマスの値は偶然	「h12 が点灯した」と読める

わかったこと：きれいな手順ではなく，規則の寄せ集め

1. **役目は言葉にできる**：どこを読むか・どこを見るか・何を写すか・何を足すか．
2. **覚えていることはペアに分かれて入る**：特に fc1 と fc2 のペアが，「こういうときはこれを足す」という規則を持っています．
3. **きれいに分けられるのは最後の足し算だけ**：得点は3つに分けられますが，一つ一つのマスの値の意味は読めません．

AI は，名前の作り方を一つのきれいな手順として覚えたのではありません．部分的な規則がたくさん重なったものとして覚えていました．大きな AI の研究でも，計算問題を同じように「規則の寄せ集め」で解いていることが報告されています[43]．重み 268 個の小さな

AI でも，同じ姿が見えます.

[43]Nikankin ほか（2025）「Arithmetic Without Algorithms: Language Models Solve Math With a Bag of Heuristics」（ICLR 2025）.

6　名前を作る

学習が終わったら，同じ順伝播（3章）を使って新しい名前を作ります．これを**生成**と呼びます（generation）．重みはもう動かしません．新しいのは，確率をどう使って文字を選ぶかだけです．

6.1　名前を作る手順

1. 始まりの印（BOS）を読ませる．
2. 次の文字の確率を出す（3章の計算）．
3. 確率を当たりやすさにしたくじを引いて，1文字選ぶ．
4. 選んだのが終わり（BOS）なら完成．ちがえば，選んだ文字を付け足し，それを読ませて2へ戻る．

学習では正解の名前を読ませましたが，ここでは AI が自分で選んだ文字を読ませます．一度に読める長さは5なので，5文字目まで来ると，終わりが出なくてもそこで止まります[44]．

6.2　確率をくじにする

学習後の AI は，始まりの次の文字に a 50.9%，m 29.7%，e 19.3% をつけます．これを，1000本のうち a の当たりが約509本，m が約297本，e が約193本入った箱から1本引くようなくじにします．a が出やすいのは確かですが，m や e が出ることもあります．引くたびに結果が変わるので，同じ AI から毎回ちがう名前が作られます．

くじの引き方　コンピュータは，0 以上 1 未満の数をでたらめに1つ選びます（乱数）．確率の帯を左から並べておき，選んだ数が入った区間の文字を当たりとします（図37）．区間の幅が確率に等しいので，確率の高い文字ほど当たりやすくなります．目をつぶってダーツを投げても，広い区画ほどよく当たるのと同じです[45]．

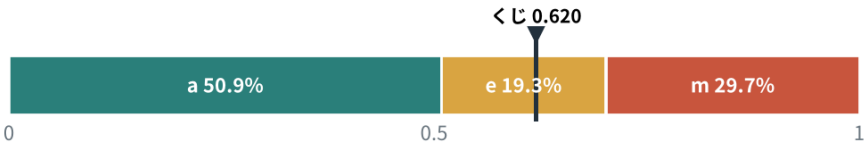


図 37　確率の帯とくじ．0.62 が m の区間に入ったので，m が選ばれる．

なぜ「いちばん確率の高い文字」を選ばないのか　いつも一番を選ぶと（貪欲法，greedy decoding），同じ AI からはいつも同じ名前しか出てきません．この AI なら，始まり → a（50.9%），a → v（49.9%）…と進み，毎回 ava だけになります．a の次は v と n がほとんど同じ確率なのに，必ず v を選んでしまいます．反対に，確率を無視して等しく選ぶと，学習前のようなでたらめな文字の列になります．確率を当たりやすさにしてくじを引くと，名前らしさを保ちながら，いろいろな名前を作れます．

[44] プログラムでは 186–200行目です．195行目で確率を求め，196行目の `random.choices` でくじを引きます．

[45] 乱数は，1章で見た乱数の種 42 から計算で作るので，同じプログラムなら誰が動かしても同じ名前の列になります．

式で書くと　累積の確率 $C_j = p_1 + \cdots + p_j$ を作り，一様乱数 r（$0 \le r < 1$）が $C_{j-1} \le r < C_j$ に入る文字 j を選ぶ．区間の幅が p_j なので，j は確率 p_j で選ばれる．

6.3　温度

確率に直す前（3章の softmax の前）に，7つの得点を**温度**（temperature）と呼ばれる数で割ります（図38）．温度が 1 より小さいと得点の差が広がり，確率の高い文字がもっと選ばれやすくなります．温度 0.5 では，始まりの次の a が 50.9% から 67.4% に増え，e は 19.3% から 9.7% に減ります．温度が 1 より大きいと差が縮まり，珍しい文字も選ばれやすくなります．microgpt は温度 0.5 を使います[46]．

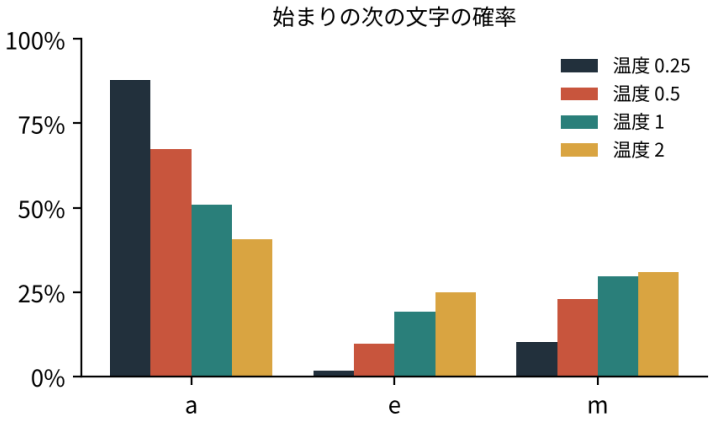


図 38　温度による，始まりの次の文字の確率のちがい．

温度は「慎重さ」と「冒険心」のつまみです．温度が低いと確率の高い文字ばかり選び，無難でよく見る名前が多くなります．0 に近づけると，いつも一番を選ぶ貪欲法と同じになります．温度が高いと確率の低い文字もよく選び，変わった名前や崩れた名前が増えます．大きくすると，でたらめに近づきます．

式で書くと　$p_j(T) = e^{z_j/T} / \sum_k e^{z_k/T}$．確率の比は $p_j/p_k = e^{(z_j - z_k)/T}$ で，$T = 0.5$ では比が2乗になる．

温度は学習では使いません．名前を作るときに，使う人が選ぶ設定です．重みを学習し直さなくても，温度だけ変えて作り直せます．ChatGPT や Claude のような大きな AI でも同じで，開発者向けの使い方では温度を指定できます．「温度」という名前は物理学から来ています．得点を温度で割って確率にする式は，統計力学のボルツマン分布と同じ形です（付録A.12）．

6.4　1つの名前ができるまで

学習後の AI が最初に作った名前「ava」を，くじ1本ずつたどります（図39）．帯は温度 0.5 で調整した確率です．始まりの次では，くじ 0.555 が a の区間（0〜0.674）に入り，a が選ばれました．a の次では，くじ 0.932 が v の区間に入り，v が選ばれました．av の次と ava の次

[46]温度 1 のときが，AI が計算したそのままの確率です．プログラムでは 187行目で `temperature = 0.5` と決め，195行目で得点を温度で割っています．

は，確率がほぼ 100% の1文字だけなので，どんな数を引いても同じ文字（a，終わり）になります．

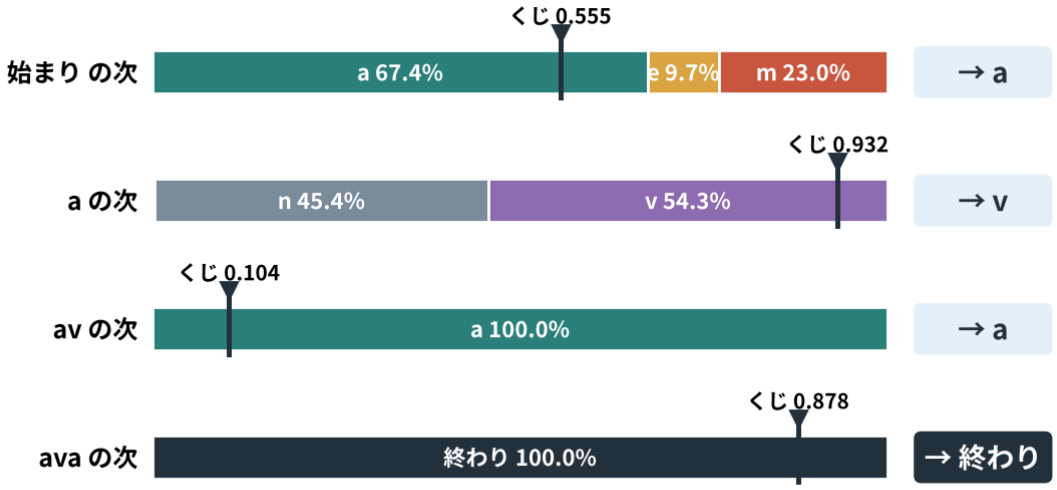


図 39　1つ目の名前「ava」ができるまで（温度 0.5）．黒い線がくじで選んだ数．

選んだ文字が，次に読む文字になる　学習のときは，正解の名前が最初から全部わかっていたので，すべての位置の予想をまとめて計算できました．名前を作るときは，次の文字がくじで決まるまで，その次を計算できません．だから1文字ずつ順にしか進めません．ChatGPT や Claude が答えを少しずつ表示していくのも同じ理由です．Attention が見に行く前の位置の見出し（k）と中身（v）は，作った順にとっておいて使い回します[47]．

6.5　作られた名前

学習後の AI に20個の名前を作らせると，次のようになりました（温度 0.5）[48]．

ava, ava, anna, anna, mia, mia, anna, anna, ava, ava, ava, mia, anna, ava, ava, anna, mia, anna, emma, ava

ava が8個，anna が7個，mia が4個，emma が1個で，4つの名前以外は1つも出てきませんでした．データが少ないので，AI は名前を丸暗記した状態です．学んだ名前は4つとも同じ1回ずつなのに，ava と anna が多く，emma は1個だけです．なぜでしょうか．

emma が少ない理由：温度 0.5 が「e」をさらに減らす　名前の確率は，1文字ずつの確率のかけ算です．emma が選ばれるには，まず始まりの次で e が出る必要があります．始まりの次の e は，温度 1 で 19.3%，温度 0.5 では 9.7% しかありません．a は 50.9% から 67.4% に増えます．温度 0.5 は「多数派をもっと多く，少数派をもっと少なく」します．だから a で始まる ava と anna が増え，emma が減りました（表23）．

[47]前の位置の k と v をとっておくしくみは，大きな AI で KV キャッシュ（KV cache）と呼ばれるものと同じです（121–122行目）．

[48]元の `microgpt.py` の4か所の数字と `input.txt` だけを変えて動かした結果そのものです（冊子『microgpt を読む』5節と同じ）．

表 23　名前ができる確率（学習後）

名前	データ	温度 1 の確率	温度 0.5 の確率	20個中の個数
ava	25%	25.1%	**36.6%**	8
anna	25%	19.3%	**30.0%**	7
mia	25%	23.6%	**22.6%**	4
emma	25%	16.8%	**9.6%**	1
ほかの名前	0%	15.1%	1.2%	0

式で書くと　$P(\text{名前}) = P(c_1 \mid c_0)\, P(c_2 \mid c_0, c_1) \cdots$（1文字ずつの確率の積）．$N$ 個作ったときの個数の見込み＝ $N \times P(\text{名前})$.

温度を変えて作る　温度だけを変えて20個ずつ作ると，次のようになりました[49]．下線はデータにない名前です.

- **温度 0.5**（データにない名前 0個）：上と同じ.
- **温度 1**（2個）：emma, mia, anna, ava, mia, mia, emma, anna, ava, ava, emma, anna, ava, emma, anna, ava, emma, ava, aia, emea
- **温度 2**（12個）：emeea, mnia, ava, ava, meia, anna, mnia, ema, m, anna, emna, anva, ava, iana, aia, mia, ava, ava, mvaa, av

1000個ずつ作って数えると（表24，図40），温度 0.25 では4つの名前しか出ず，emma はとても少なくなります．温度 1 では4つがほぼ同じくらいになり，データにない名前が約15%出てきます．温度 2 では半分以上がデータにない名前です（aia，mva，anva，emea，a など）.

表 24　1000個ずつ作ったときの個数（乱数の種 7）

温度	ava	anna	mia	emma	データにない名前
0.25	501	368	113	18	0
0.5	350	335	216	88	11
1	261	193	235	163	148
2	136	64	117	80	603

[49] 3つとも，学習を終えた直後の同じ乱数の続きを使い，温度だけを変えました．温度 0.5 の列は，元のプログラムの結果と同じです.

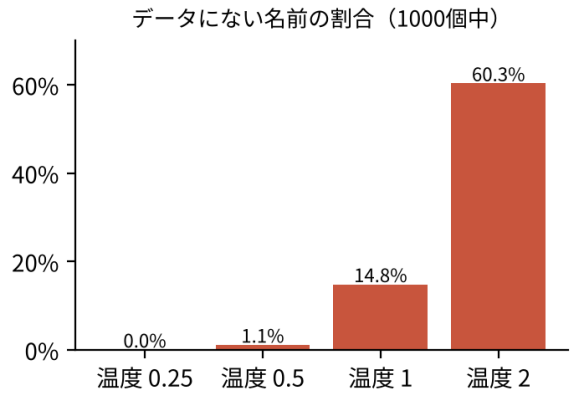


図 40　1000個のうち，データにない名前の割合．

6.6　データにない名前は，覚えた部品の組み合わせ

データにない名前は，どのようにしてできたのでしょうか．各位置で選ばれた文字の確率（温度 1）を並べると，次のようになります．

名前	1文字ずつの確率
ava	始まり → a 50.9%，a → v 49.9%，av → a 99.2%，ava → 終わり 99.8%
aia	始まり → a 50.9%，a → **i 3.5%**，ai → a 99.2%，aia → 終わり 99.8%
anva	始まり → a 50.9%，a → n 45.6%，an → **v 10.8%**，anv → a 98.8%，anva → 終わり 99.8%

aia は，始まりの次に a を選び，次に珍しい i（3.5%）を引いた名前です．そのあとは「i の次は a」「名前の最後の a のあとは終わり」という，mia で覚えた規則がそのまま使われました．anva も，an の次に珍しい v（10.8%）を引いたあと，「v の次は a」（ava で覚えた）で続いています．新しい名前は，でたらめではなく，覚えた部品（規則）をつなぎかえたものです．i と v が同じ向きに置かれていたこと（4章，5章）が，ここで効いています．

6.7　学習の前と途中で作った名前

学習前の重みで作った名前は，でたらめな文字の列です（温度 0.5，乱数の種 7）．

ianam, iamai, aaiva, en, mi, aviea, ivenn, imaae, nminm, ivnim, m, ne, eivem, anvnv, innnm, v, mnann, （空）, veina, meaav

7文字にほぼ同じ確率をつけるので，「終わり」もほかの文字と同じくらいしか選ばれず，1文字で終わったり（m，v），何も作らずに終わったり（空）します．5文字のものは，一度に読める長さ（5）で止められた名前です．

学習の途中で10個ずつ作ると，名前は表25のように変わりました[50]．覚えやすい規則から順に覚えるので（4章），作る名前も「a ばかり」→「終わり方が名前らしい」→「4つの名前」と変わっていきました．

[50]温度 0.5，どの回も乱数の種 7 で作りました．学習の乱数には影響させていません．

表 25　学習の途中で作った名前（10個ずつ）

学習の回数	作った名前	4章で見たこと
学習前	ianam, iamai, aaiva, en, mi, aviea, ivenn, imaae, nminm, ivnim	7文字にほぼ同じ確率
25回	aeea, aeaa, aea, aia, an, avaaa, ava, eeaaa, annnv, aim	まず「a が多い」ことを覚える
50回	aiaa, aiaa, aia, ann, mnva, aia, mia, ania, mnn, anva	終わりの規則を覚えつつある
100回	anna, aia, anna, anna, ava, anna, anna, aia, anna, mnna	emm → a，em → m も覚えた
300回	anna, anna, anna, ava, ava, anna, ava, anna, anna, mia	m → i も 80% まで覚えた

6.8　データを増やすと

元の設定（3万2千個の名前，1000回の学習，温度 0.5）で作った20個の名前は，次のとおりです（冊子『microgpt を読む』7.2節）．

kamon, ann, karai, jaire, vialan, karia, yeran, anna, areli, kaina, konna, keylen, liole, alerin, earan, lenne, kana, lara, alela, anton

ann，anna，lara，anton のように実在する名前もありますが，kamon や vialan のように，データにはないが名前らしいものが多く含まれます．3万2千個は覚えきれないので，AI は「名前によく出る文字の並び方」の統計を学んでいます．4つの名前では丸暗記，3万2千個では部品の統計．同じプログラムでも，データの量で AI の「覚え方」が変わります．

「新しい名前」と，大きな AI の作り話　microgpt のプログラムは，作った名前を表示するとき「new, hallucinated names」（新しい，幻の名前）と書きます．hallucination（幻覚）は，大きな AI が，事実でないことをもっともらしく作ってしまうことを指す言葉としても使われます．しくみは同じです．大きな AI も，覚えた言葉の並び方の確率に従って，次のトークンをくじで選んでいるだけです．並び方としてもっともらしければ，事実かどうかに関係なく作られます[51]．kamon は「名前らしい」けれど，実在の名前とは限りません．大きな AI の答えも，「文章としてもっともらしい」ことと「正しい」ことは別です．確かめる習慣が大切です．

6章のまとめ：名前は，確率のくじを1本ずつ引いて作ります．始まりから次の文字の確率を出してくじを引き，選んだ文字を読ませ，終わりが出たら完成です．温度でくじの当たりやすさを調整します．温度 0.5 は多数派をもっと多くするので，a で始まる ava と anna が多く，emma は少なくなりました．新しい名前は，覚えた部品の組み合わせです．

[51] 大きな AI では，事実に合う答えを選ぶように追加の学習をしたり，調べた資料をもとに答えさせたりして，作り話を減らす工夫がされています（「LLMの基礎3　対話モデルへ」参照）．

7　本物の AI と比べる

microgpt のプログラムの冒頭には「このファイルがアルゴリズムのすべてで，それ以外は効率化にすぎない」という趣旨のことが書かれていました．この章では，この主張を確かめます．

7.1　元の設定の microgpt：表の形はどう変わるか

ほかの資料（冊子『microgpt を読む』や wte・lm_head の解説）は，元の設定（黒板の幅 16，文字 27 種類）で書かれています．表の形を並べると表26のようになります．計算の手順はまったく同じで，表が大きくなるだけです．

表 26　ミニマムな例と元の設定の，表の形

表	ミニマムな例	数	元の設定	数
wte（E）	7×4	28	27×16	432
wpe（P）	5×4	20	16×16	256
W_Q, W_K, W_V, W_O	4×4 が4つ	64	16×16 が4つ	1,024
W_1（fc1）	16×4	64	64×16	1,024
W_2（fc2）	4×16	64	16×64	1,024
lm_head（U）	7×4	28	27×16	432
合計		268		4,192

資料をまたいで読むときの注意：ミニマムな例の「16」は MLP の検出器の数（4×4）でした．元の設定の「16」は黒板の幅で，MLP の検出器は $16 \times 4 = 64$ 個です．同じ「16」でも意味がちがいます．元の設定の wpe が 16×16 なのは，黒板の幅も一度に読める長さもたまたま 16 だからです．

ヘッド：黒板の幅を分けて見に行く　元の設定では Attention のヘッドの数が4です．q，k，v（各16個）を4個ずつの4組に分けて，組ごとに別々に得点と割合を計算し，見に行く位置を決めます．4組の結果（4個ずつ）を並べて16個に戻し，W_O（16×16）を掛けて黒板に足します．1つのヘッドの幅は $16 \div 4 = 4$ で，得点を割る数はその平方根の 2 です．ミニマムな例はヘッドが1つなので，この分け方は出てきません．複数のヘッドは，同じ文を「文法の係」「意味の係」のように分担して読むようなものです（ただし，実際の分担は学習で決まり，人間にきれいに分かるとは限りません）．

7.2　大きさを比べる

表27は，ミニマムな例，元の microgpt，GPT-2（最小のもの），GPT-3 を並べたものです[52]．GPT-2 は 2019年，GPT-3 は 2020年に OpenAI が発表した言語モデルで，ChatGPT のもとになった GPT シリーズの初期のものです．microgpt のモデルは GPT-2 とほぼ同じ構造です．ちがいは主に，この表の数の大きさです．

[52]ChatGPT や Claude のような最近の大きな AI は，構造や大きさがくわしく公表されていないものが多いので，ここでは公表されている GPT-2 と GPT-3 で比べます．

表 27　4つのモデルの大きさ

	ミニマムな例	microgpt	GPT-2（最小）	GPT-3
黒板の幅	4	16	768	12,288
層の数（Attention ＋ MLP の組）	1	1	12	96
Attention のヘッドの数	1	4	12	96
一度に読める長さ	5	16	1,024	2,048
語彙（トークンの種類）	7	27	50,257	50,257
重みの数	**268**	**4,192**	**約1.24億**	**約1,750億**

重みの数を「1個 ＝ 1 mm」で並べると，ミニマムな例（268個）は 26.8 cm でものさしくらい，microgpt（4,192個）は約 4.2 m で教室の幅くらい，GPT-2 最小（約1.24億個）は約 124 km でフルマラソン約3回分，GPT-3（約1,750億個）は約 17万5千 km で地球を約4.4周する長さです[53]．GPT-3 の重みは，microgpt の約4,000万倍，ミニマムな例の約6億5千万倍です（図41左）．

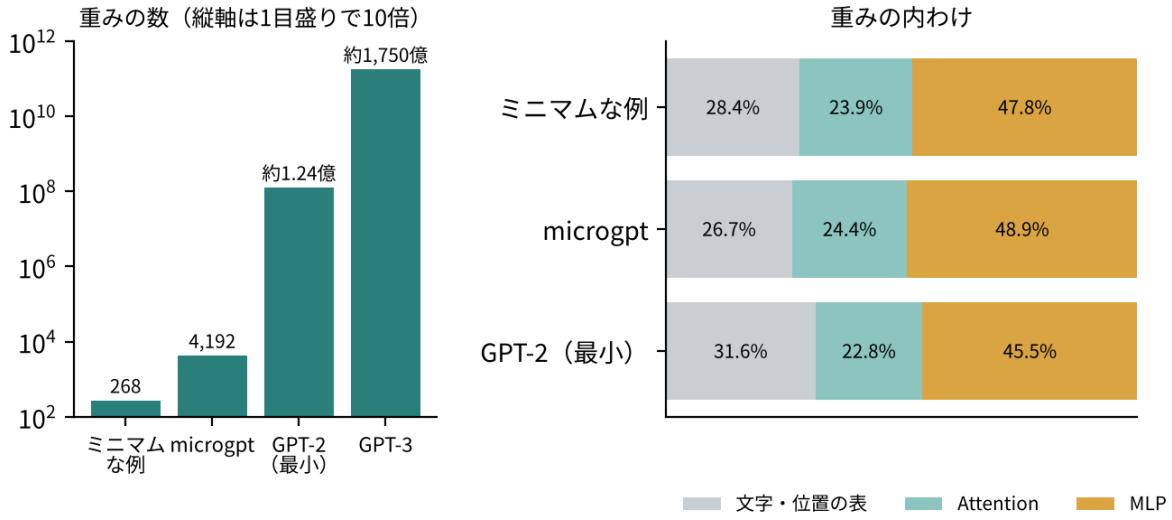


図 41　左：重みの数（縦軸は1目盛りで10倍）．右：重みの内わけ．

重みの配分は，大きさが変わってもよく似ている　重みを「文字・位置の表（wte，wpe，lm_head）」「Attention」「MLP」に分けると（図41右），どのモデルでも MLP がほぼ半分（48%，49%，46%），Attention が約4分の1です[54]．「こういうときは次にこれ」という規則の多くを持つ MLP に，いちばん多くの重みが使われています．大きさは約46万倍ちがっても，配分の考え方は同じです．

層を重ねる　microgpt は Attention と MLP の組を1回だけ通します（1層）．GPT-2 最小は12回，GPT-3 は96回通します．黒板は1枚のまま，Attention と MLP の組が何回も書き足していきます．前の層が書き足した結果を次の層が読めるので，「見つけたものを組み合わせ

[53]地球一周は約4万 km，フルマラソンは 42.195 km として計算しました．

[54]GPT-2 の数は，公表されている構造（黒板の幅 768，12層，語彙 50,257，長さ 1,024）から数えたもので，合計 124,439,808 個です（付録A.13）．GPT-2 の Attention と MLP の数には定数項（バイアス）も含めました．正規化の重み（0.03%）は図から省いています．GPT-2 は lm_head を wte と共有しています（5章）．

て，さらに見つける」ことができます．組み立て工場のラインのように，1つの工程では簡単な作業しかしなくても，工程を重ねると複雑な製品ができます．

一度に読める長さ　一度に読める長さは，Attention が見られる位置の数の上限です．どの位置も自分より前の位置を全部見るので，長さが2倍になると，見る組み合わせの数は約4倍になります（表28，付録A.14）．

表 28　一度に読める長さと，Attention が見る組み合わせの数

一度に読める長さ n　見る組み合わせの数 $n(n+1)/2$	
5（ミニマムな例）	15
16（microgpt）	136
1,024（GPT-2）	524,800
2,048（GPT-3）	2,098,176

語彙　microgpt の1つのトークンは1文字ですが，GPT-2 と GPT-3 は，データの中でよく出る文字の組（単語や単語の一部）を1つのトークンとしてまとめます[55]．語彙が大きくなると wte と lm_head の表が大きくなり，GPT-2 では 50,257 行 × 768 列で，重みの約3割がこの表です．AI が知っているのは番号だけ，という点は同じです．

7.3　データと学習の量

表 29　学習に使うデータの量

	ミニマムな例	microgpt	GPT-3
データ	名前 4個	名前 32,033個	大量の文章
問題の数（トークンの数）	18	228,146	約 3,000億
学習の回数	300	1,000	—
1回の学習で使うデータ	名前 1個	名前 1個	多くの文書をまとめて

元の microgpt のデータは 32,033個の名前，196,113文字で，問題にすると 228,146問です（表29）[56]．ところが元の microgpt は，1000回の学習で名前を1000個しか使いません．データ全体の約3%です．それでも，名前らしい新しい名前を作れるようになりました（6章）．大きな AI は，1回の学習で多くの文書をまとめて読み，その平均の勾配で重みを更新します．

学ばせる名前を増やすとどうなるか　ミニマムな例の黒板の幅（4）のまま，学ばせる名前を増やして試しました（表30）[57]．名前を増やすときは，1つの名前あたりの学習回数を保つ必

[55] よく出る文字の組をまとめる方法の一つに BPE（byte pair encoding）があります．くわしくは「LLMの基礎1　トークンと言語モデル」を参照してください．

[56] 196,113文字に，名前ごとの「終わり」を1つずつ足した数です．GPT-3 の数は発表論文（Brown ほか，2020）によります．

[57] 冊子『小さなAIの頭の中をのぞく』5節の結果です．使った名前は sophia，mary，celine，elena に，emily，chloe，lena，nora，mia，ella，maria，clara を順に加えたものです．sophia と celine が6文字なので，一度に読める長さは 7 にしました．このため，表の4個の行は本文の4つの名前とは別の組で，損失の数もちがいます．

要があります．黒板が4つの数だと，6個くらいが覚えきれる限度でした．それを超えると覚えきれず，名前どうしが混ざった新しい名前を作ります．黒板の幅を 8 にすれば覚えられますが，表が大きくなって読みにくくなります．

表 30　名前の数と覚え方（黒板の幅 4）

名前の数	学習の回数	損失／これ以上下がらない線	作った名前
4	300	0.243 / 0.222	4つとも再現
6	300	0.536 / 0.291	学習不足で崩れる（celoe など）
6	600	0.329 / 0.291	6つとも再現
8	1000	0.448 / 0.354	覚えきれず，cena などを作る
12	600	0.769 / 0.439	celia，elara も作るが崩れも多い

4つの名前ではほぼ丸暗記でした．名前が多すぎて覚えきれないと，AI は「名前によく出る文字の並び方」という統計を学ぶしかなくなります．大きな AI でも，重みの数とデータの量を増やすほど，初めて見る文章での損失が下がる傾向が知られています（スケーリング則[58]）．

7.4　計算のしかた

microgpt は，数を1つずつ `Value` という部品に入れて，足し算や掛け算を1回ずつ記録しながら計算します．Python の標準機能だけで動き，計算の中身が全部見えますが，その代わり遅く，ミニマムな例で約2秒，元の設定で2〜3分かかります．大きな AI は，数の表（行列）をまとめて掛け算します．GPU という，同じ種類の計算を大量に同時に行える装置を，たくさん並べて使います[59]．計算の式は同じで，計算のしかたが速いのです．microgpt は1人が電卓で1つずつ計算するようなもの，大きな AI は体育館いっぱいの人が同じ形の計算を一斉に行うようなものです．

たとえば「表と黒板の掛け算」は，microgpt では行の数だけ積の和をくり返しますが，行列でまとめれば $\boldsymbol{y} = W\boldsymbol{x}$ の1回で書けます．Attention の得点も，前の位置ごとに q と k の積の和を求める代わりに，全位置の q と k を行列 Q，K にまとめて $QK^{\mathsf{T}}/\sqrt{d}$ を一度に計算します．学習のときは答えの文章が最初から全部わかっているので，すべての位置の予想を一度にまとめて計算できます．名前や文章を作るときに1文字（1トークン）ずつ順にしか進めないのは，大きな AI も同じです．

7.5　構造の小さなちがい

プログラムの93行目の注釈には，GPT-2 とのちがいが3点だけ書かれています（表31）．どれも「計算を簡単にする」ための変更です．このうち rmsnorm とバイアスなしは，最近の大きな AI（LLaMA など）でも使われている選び方です．黒板（残差の流れ），Attention，MLP，lm_head，softmax という部品の組み立ては GPT-2 と同じです．

[58]「LLMの基礎2　事前学習とスケーリング則」を参照してください．

[59]大きな AI の開発では，行列の計算を GPU で速く行うためのライブラリ（PyTorch など）を使います．microgpt はそれを使わず，すべてを自分で書いています．

表 31　GPT-2 とのちがい

	GPT-2	microgpt	ちがい
大きさをそろえる計算	layernorm	rmsnorm	layernorm は平均を引いてからそろえる．rmsnorm は平均を引かない（簡単）
定数項（バイアス）	あり	なし	掛け算のあとに足す一定の数．microgpt は省いた
MLP の折り曲げ	GELU	ReLU	GELU は 0 の近くでなめらかに曲がる．ReLU は 0 でカクッと曲がる

学習後の調整：対話のできる AI にする　microgpt が行うのは，大量のデータで「次のトークン」を当てる学習（事前学習）だけです．ChatGPT や Claude のような対話型の AI は，事前学習のあとに，質問と答えの例で指示に従うよう追加の学習（指示調整）をし，よい答えと悪い答えの人間の評価から好まれる答え方を学びます（RLHF など）[60]．どの段階でも，重みを損失が下がる向きに動かすという原理は同じです．何を「よい答え」とするか（損失の決め方）がちがいます．

7.6　同じところ

骨組みは同じです（図42）．予想（順伝播）は，トークンを番号にし，表を引いて最初のメモを作り，Attention と MLP を層の数だけくり返し，得点を出して softmax で確率に直します．学習は，損失（– ln 正解の確率）を測り，逆伝播で勾配を求め，重みを更新します．生成は，温度で調整した確率でくじを引き，選んだトークンを付け足して読むことをくり返します．

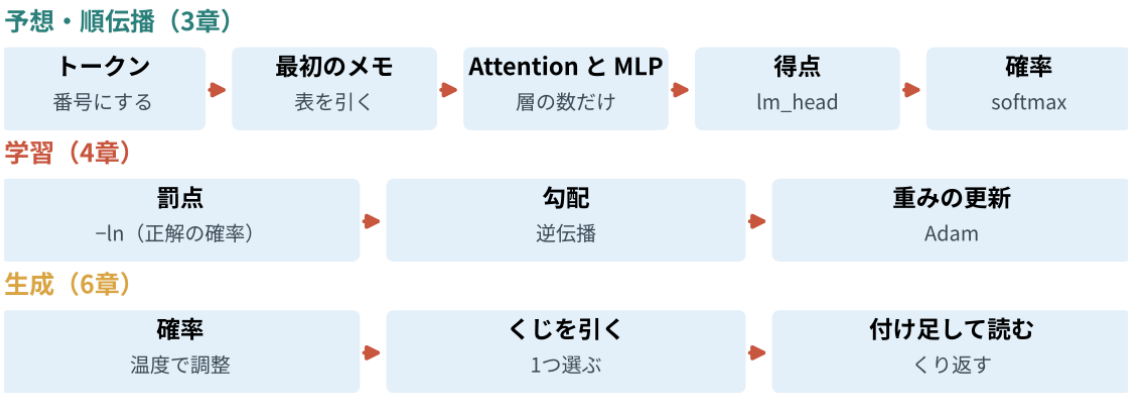


図 42　microgpt と大きな AI に共通の骨組み．

1〜6章で見たことは，大きな AI ではどうなっているでしょうか（表32）[61]．

[60] RLHF は reinforcement learning from human feedback（人間の評価からの強化学習）の略です．くわしくは「LLMの基礎3　対話モデルへ」を参照してください．

[61] 右の列は，大きな AI について一般に知られている傾向を簡単にまとめたものです．すべての場合に当てはまるとは限りません．

表 32　microgpt で見たことと，大きな AI

microgpt で見たこと	大きな AI では
AI が知っているのは番号だけ（1章）	同じ．トークンの番号だけを扱う
正しく迷う．損失は 0 にならない（1章）	同じ．文章の続きは一つに決まらないことが多い
黒板に書き足していく（2・3章）	同じしくみ．黒板の幅は 768〜12,288 と広い
覚えやすいものから覚える（4章）	よく出るパターンほど早く学ばれる傾向がある
同じ役目の文字は同じ向きに置かれる（5章）	似た使われ方の単語は近い向きになることが知られている
温度とくじ（6章）	同じ．使う人が温度を選べる
もっともらしい新しい名前（6章）	もっともらしい作り話（幻覚）

ちがうところ：中を読むのがずっと難しい　ミニマムな例の 268 個の重みは，5章でほぼ全部の役目を読み解けました（i と v，16個の検出器など）．それでも，1つ1つのマスの値には意味がなく，「規則の寄せ集め」として覚えていることがわかっただけでした．1,750億個の重みになると，中で何が起きているのかを人間が読み解くのはとても難しくなります．これを調べる研究（解釈可能性の研究，interpretability）が，いまも盛んに行われています．

7章のまとめ：大きさは桁ちがいで，重みは GPT-3 で約1,750億個（ミニマムな例の約6億5千万倍）です．ちがうのは主に計算のしかた（行列でまとめて GPU で一斉に計算する）と，構造の小さな3点（rmsnorm，バイアスなし，ReLU）です．骨組み（番号 → メモ → Attention と MLP → 確率）と学習の原理（損失を勾配で下げる）は同じです．
最初の問いへの答え：「このファイルがアルゴリズムのすべてで，それ以外は効率化にすぎない」という microgpt の主張は，計算の骨組みについてはそのとおりでした．ただし，対話のための追加の学習や，大きさから生まれる力は，200行の中には入っていません．

おわりに：microgpt を解剖して見えたこと

1. **全体像と準備**：AI は「次を当てる」をくり返します．名前は番号の列になり，18問の問題になります．答えが分かれる場面では，正しく迷うのが最善です．
2. **黒板と9つの表**：268 個の重みは9つの表のマスです．表を掛けると，数の個数が「列の数」から「行の数」に変わります．計算の途中の数は，位置ごとの黒板に足し算で書き足されます．
3. **1文字を読む**：最初のメモ → Attention → MLP → 得点 → 確率．得点は3つの書き込みの足し算で，いちばん効くのは MLP です．
4. **外れ具合と学習**：損失を − ln で測り，逆伝播で勾配を求め，Adam で更新します．覚えやすいものから覚え，i と v は自然に同じ向きになりました．
5. **学習後の表を読む**：覚えたことは，マスの値ではなく，向きの似かた，検出器の点灯，得点への寄与に現れます．AI は規則の寄せ集めとして覚えていました．

6. **名前を作る**：確率のくじを1本ずつ引きます．温度で顔ぶれが変わります．新しい名前は，覚えた部品の組み合わせです．
7. **本物の AI と比べる**：大きさと計算のしかたはちがいますが，骨組みと学習の原理は同じです．

次に読むものとして，microgpt の200行を元の設定（3万2千個の名前）も含めて順に読む冊子『microgpt を読む』，トークンと言語モデル・事前学習とスケーリング則・対話モデルを扱う「LLMの基礎」1〜3，ニューラルネットワークから Transformer までを扱う「深層学習」1〜4があります．

付録A　数式

本文の計算を数式で書きます．理系の大学1〜2年生程度を想定しています．記号は付録Bにまとめました．ベクトルは縦ベクトル（太字），表は行列（大文字）で，形を（　）や「行×列」で添えます．$V=7$（語彙），$d=4$（黒板の幅），$B=5$（一度に読める長さ）です．1つの名前を番号の列 $c_0, c_1, \ldots, c_n$（$c_0=c_n=6$，BOS）で表し，位置 $t=0,\ldots,n-1$ で c_t を読んで c_{t+1} を当てます．

A.1　答えが分かれる場面と，損失の下限（1章）

名前全体の確率は，確率の乗法定理により

$$P(\text{名前}) = P(c_1 \mid c_0)\, P(c_2 \mid c_0, c_1) \cdots P(c_n \mid c_0, \ldots, c_{n-1}) \tag{1}$$

です．データの割合で計算すると $P(\text{emma}) = \frac{1}{4}\times 1\times 1\times 1\times 1 = \frac{1}{4}$，$P(\text{ava}) = \frac{2}{4}\times\frac{1}{2}\times 1\times 1 = \frac{1}{4}$ で，4つとも $\frac{1}{4}$ になります．

データの割合どおりに答えたときの18問の罰点の合計は，始まりの4問で $-\ln\frac{2}{4}$ が2回と $-\ln\frac{1}{4}$ が2回，a の次の2問で $-\ln\frac{1}{2}$ が2回，ほかの12問は $-\ln 1 = 0$ なので

$$2(0.693) + 2(1.386) + 2(0.693) = 5.545, \qquad 5.545 \div 18 = 0.308 \tag{2}$$

です．これはデータに残るあいまいさ（条件付きエントロピー）で，どんな予想をしても損失の平均はこれより小さくなりません．ある場面のデータの割合を $\boldsymbol{f}$，AI の確率を $\boldsymbol{p}$ とすると，その場面の損失の平均は $-\sum_c f_c \ln p_c = H(\boldsymbol{f}) + D_{\mathrm{KL}}(\boldsymbol{f}\,\|\,\boldsymbol{p}) \geq H(\boldsymbol{f})$ で，等号は $\boldsymbol{p}=\boldsymbol{f}$ のときです（ギブスの不等式）．$e^{\text{損失の平均}}$ をパープレキシティ（perplexity）と呼び，「平均して何択で迷っているか」を表します．限界は $e^{0.308}=1.36$ 択，学習前は $e^{1.973}=7.2$ 択，学習後は $e^{0.347}=1.41$ 択です．$e^{-0.347}\approx 0.71$ は正解につけた確率の幾何平均です．

A.2　最初のメモ，rmsnorm，cos（3章）

$$\mathrm{rms}(\boldsymbol{a}) = \frac{\boldsymbol{a}}{\sqrt{\frac{1}{d}\sum_i a_i^2 + \varepsilon}}, \quad \varepsilon = 10^{-5}, \qquad \underbrace{\boldsymbol{x}_t^{(0)}}_{(4)} = \mathrm{rms}\big(\underbrace{E_{c_t}}_{(4)} + \underbrace{P_t}_{(4)}\big) \tag{3}$$

E_c は E（7×4）の c 行目，P_t は P（5×4）の t 行目を縦ベクトルにしたものです．ε を無視すれば，$\mathrm{rms}(\boldsymbol{a})$ の長さは $\sqrt{d}=2$ です．microgpt は Attention と MLP の直前にも rmsnorm をかけますが，最初のメモはすでに長さ 2 なので，Attention の直前の rmsnorm ではほとんど変わりません．2つのベクトルの向きのそろい方は

$$\cos\theta = \frac{\boldsymbol{a}\cdot\boldsymbol{b}}{|\boldsymbol{a}|\,|\boldsymbol{b}|}, \qquad \boldsymbol{a}\cdot\boldsymbol{b} = \boldsymbol{a}^{\mathsf{T}}\boldsymbol{b} = \sum_i a_i b_i \tag{4}$$

です．p1 の e と p2 の m は $\cos\theta = 0.98$，p2 の m と p3 の m は -0.41 でした．

A.3　Attention と softmax（3章）

$\hat{\boldsymbol{x}}_t = \mathrm{rms}(\boldsymbol{x}_t^{(0)})$ として，

$$\underbrace{\boldsymbol{q}_t}_{(4)} = \underbrace{W_Q}_{4\times 4}\hat{\boldsymbol{x}}_t, \quad \boldsymbol{k}_j = W_K\hat{\boldsymbol{x}}_j, \quad \boldsymbol{v}_j = W_V\hat{\boldsymbol{x}}_j, \tag{5}$$

$$a_{tj} = \frac{\boldsymbol{q}_t \cdot \boldsymbol{k}_j}{\sqrt{d}}, \qquad \alpha_{tj} = \frac{e^{a_{tj}}}{\sum_{j'=0}^{t} e^{a_{tj'}}} \quad (0 \le j \le t) \tag{6}$$

$$\underbrace{\boldsymbol{o}_t}_{(4)} = \sum_{j=0}^{t} \alpha_{tj} \boldsymbol{v}_j, \qquad \boldsymbol{x}_t^{(1)} = \boldsymbol{x}_t^{(0)} + \underbrace{W_O}_{4\times4} \boldsymbol{o}_t \tag{7}$$

$j \le t$ なので，先の位置は見られません（因果的な Attention）．割合 $(\alpha_{t0}, \ldots, \alpha_{tt})$ の長さは $t+1$ で，黒板の幅 d とは関係しません．p3 の例では，得点 $(+2.69, -0.11, +0.00, +1.00)$，割合 $(0.762, 0.046, 0.052, 0.140)$ です．$\sqrt{d}$ で割るのは，$\boldsymbol{q}$，$\boldsymbol{k}$ の成分が平均 0，分散 1 の独立な数なら内積の分散が d になるので，それを 1 にそろえるためです．元の microgpt は `n_head` $= 4$ で，16個の数を4つずつに分けて4組の Attention を別々に行い（マルチヘッド），結果を並べて W_O（16×16）を掛けます．

A.4 MLP（3章）

$$\underbrace{\boldsymbol{u}_t}_{(16)} = \underbrace{W_1}_{16\times4} \mathrm{rms}(\boldsymbol{x}_t^{(1)}), \qquad \underbrace{\boldsymbol{h}_t}_{(16)} = \mathrm{ReLU}(\boldsymbol{u}_t), \qquad \boldsymbol{x}_t^{(2)} = \boldsymbol{x}_t^{(1)} + \underbrace{W_2}_{4\times16} \boldsymbol{h}_t \tag{8}$$

ReLU は成分ごとに $\max(0, u_i)$ をとります．u_i は W_1 の i 行目（型紙 $\boldsymbol{\kappa}_i$）と大きさをそろえた黒板 $\hat{\boldsymbol{x}}$ の内積です．$|\hat{\boldsymbol{x}}| = 2$ なので $u_i = 2|\boldsymbol{\kappa}_i| \cos\theta_i$ で，型紙と同じ向きに近いほど大きく，$90°$ を超えると負（黙る）になります．書き足す量 $W_2\boldsymbol{h} = \sum_i h_i \boldsymbol{w}_i$（$\boldsymbol{w}_i$ は W_2 の i 列目，(4)）は，点灯した検出器の「足す内容」を点灯の強さで重みづけした和です．ReLU がなければ W_2W_1（4×16 と 16×4 の積）は1つの 4×4 行列と同じになり，16個に広げる意味がありません．

A.5 得点の3つへの分解（3章，5章）

$\boldsymbol{x}_t^{(2)} = \boldsymbol{x}_t^{(0)} + W_O\boldsymbol{o}_t + W_2\boldsymbol{h}_t$ なので

$$\underbrace{\boldsymbol{z}_t}_{(7)} = \underbrace{U}_{7\times4} \boldsymbol{x}_t^{(2)} = \underbrace{U\boldsymbol{x}_t^{(0)}}_{\text{最初のメモ}} + \underbrace{UW_O\boldsymbol{o}_t}_{\text{Attention}} + \underbrace{UW_2\boldsymbol{h}_t}_{\text{MLP}}, \qquad p_{t,c} = \frac{e^{z_{t,c}}}{\sum_{c'} e^{z_{t,c'}}} \tag{9}$$

が正確に成り立ちます（計算で確かめると，差は 2×10^{-15} 以下）．lm_head の前に rmsnorm がないことが条件です．softmax は，7つの得点すべてに同じ数を足しても変わらない（分子も分母も同じ倍率になる）ので，意味があるのは得点の差です．$p_a/p_e = e^{z_a - z_e}$ で，p3 では $e^{4.57} \approx 96$ でした．プログラムでは，数が大きくなりすぎないよう，いちばん大きい得点を引いてから e の何乗かを求めています（97–101行目）．

A.6 損失（交差エントロピー）（4章）

$$L = \frac{1}{n} \sum_{t=0}^{n-1} \left(-\ln p_{t,\,c_{t+1}}\right) \tag{10}$$

mia（$n = 4$）の1回目は $(1.869 + 2.151 + 1.764 + 2.029)/4 = 1.953$ です．7文字に同じ確率 $1/7$ をつけると $L = \ln 7 \approx 1.946$，元の microgpt（27文字）なら $\ln 27 \approx 3.30$ です．正解の文字を 1，ほかを 0 とした分布 $\boldsymbol{y}_t$（7，one-hot）と，AI の確率 $\boldsymbol{p}_t$ の交差エントロピー $H(\boldsymbol{y}_t, \boldsymbol{p}_t) = -\sum_c y_{t,c} \ln p_{t,c}$ が，ちょうど $-\ln p_{t,\,c_{t+1}}$ になります．

A.7　勾配，連鎖律，得点の勾配（4章）

勾配は，重み w を小さな数 Δ だけ動かしたときの損失の変化の割合です．

$$\frac{\partial L}{\partial w} \approx \frac{L(w+\Delta) - L(w)}{\Delta} \tag{11}$$

本文の例（$\Delta = 0.001$）では $(1.953936 - 1.953475)/0.001 = 0.461$ で，自動微分の値 0.4608 と一致しました．L が y を通して w に依存するとき $\dfrac{\partial L}{\partial w} = \dfrac{\partial L}{\partial y}\dfrac{\partial y}{\partial w}$（連鎖律）で，道が複数あるときは各道の積を足します．$f = ab + a$ なら $\partial f/\partial a = b + 1 = 4$，$\partial f/\partial b = a = 2$ です．足し算 $z = x + y$ の局所的な微分は $\partial z/\partial x = 1$，掛け算 $z = xy$ は $\partial z/\partial x = y$（相手の値）で，microgpt の `Value` はこれを計算のたびに記録します．

得点 $\boldsymbol{z}$ から確率と損失を作ると，$\ell = -\ln p_c = -z_c + \ln \sum_k e^{z_k}$（$c$：正解）なので

$$\frac{\partial \ell}{\partial z_j} = -y_j + \frac{e^{z_j}}{\sum_k e^{z_k}} = p_j - y_j, \qquad \text{ベクトルで} \quad \frac{\partial \ell}{\partial \boldsymbol{z}} = \boldsymbol{p} - \boldsymbol{y}\ (7) \tag{12}$$

になります．$\ln \sum e^z$ を z_j で微分すると，ちょうど softmax の p_j になるためです．勾配の和は $\sum_j (p_j - y_j) = 1 - 1 = 0$ で，正解の得点を上げた分だけほかを下げる形です．

A.8　行列の勾配（4章，5章）

基本の形　$\boldsymbol{y} = W\boldsymbol{x}$（$W$：$m \times n$，$\boldsymbol{x}$：$(n)$，$\boldsymbol{y}$：$(m)$）で，損失の $\boldsymbol{y}$ についての勾配 $\boldsymbol{\delta} = \partial L/\partial \boldsymbol{y}$（$(m)$）がわかっているとします．$y_i = \sum_j W_{ij} x_j$ なので $\partial y_i/\partial W_{ij} = x_j$ で，

$$\frac{\partial L}{\partial W_{ij}} = \delta_i\, x_j, \qquad \underbrace{\frac{\partial L}{\partial W}}_{m\times n} = \underbrace{\boldsymbol{\delta}}_{(m)}\ \underbrace{\boldsymbol{x}^{\mathsf{T}}}_{(n)^{\mathsf{T}}}, \qquad \underbrace{\frac{\partial L}{\partial \boldsymbol{x}}}_{(n)} = \underbrace{W^{\mathsf{T}}}_{n\times m}\underbrace{\boldsymbol{\delta}}_{(m)} \tag{13}$$

です．重みの勾配は W と同じ形の行列で，「出る側の勾配」と「入る数」の外積（outer product）になります．入る側への勾配は W の転置で逆向きに運ばれます．逆伝播は，この2つの式を出力側から順にくり返すことです．

位置ごとの足し算　同じ W をすべての位置で使うので（2章），勾配は位置ごとの外積の和になります．

$$\frac{\partial L}{\partial W} = \sum_{t=0}^{n-1} \boldsymbol{\delta}_t\, \boldsymbol{x}_t^{\mathsf{T}} \tag{14}$$

lm_head：毎回すべての行が動く　$\boldsymbol{z}_t = U\boldsymbol{x}_t^{(2)}$，$\boldsymbol{\delta}_t = \partial L/\partial \boldsymbol{z}_t = (\boldsymbol{p}_t - \boldsymbol{y}_t)/n$ なので

$$\underbrace{\frac{\partial L}{\partial U}}_{7\times 4} = \frac{1}{n}\sum_{t=0}^{n-1} \underbrace{(\boldsymbol{p}_t - \boldsymbol{y}_t)}_{(7)}\ \underbrace{\boldsymbol{x}_t^{(2)\mathsf{T}}}_{(4)^{\mathsf{T}}}, \qquad \frac{\partial L}{\partial U_c} = \frac{1}{n}\sum_t (p_{t,c} - y_{t,c})\, \boldsymbol{x}_t^{(2)} \tag{15}$$

です．softmax の確率 $p_{t,c}$ は 0 にならないので，c がどの文字でも係数 $p_{t,c} - y_{t,c}$ は 0 になりません．だから lm_head は7つの行すべてに毎回勾配がつきます．正解の行（$y = 1$）は係数が負で $\boldsymbol{x}_t^{(2)}$ の向きへ引き寄せられ，ほかの行は係数が正で遠ざけられます．1回目の学習（mia）でこの式を計算すると，プログラムの自動微分と 1×10^{-16} まで一致し，7つの行すべてが 0 でない勾配を持っていました．

wte と wpe：引いた行だけが動く　行を引く操作は，one-hot ベクトル $\boldsymbol{e}_c$（c 番目だけ 1，(7)）を使えば $E_c = E^{\mathsf{T}}\boldsymbol{e}_c$ と書けます．$\boldsymbol{s}_t = E_{c_t} + P_t$ の勾配を $\boldsymbol{\delta}_t$ とすると

$$\frac{\partial L}{\partial E} = \sum_t \boldsymbol{e}_{c_t}\boldsymbol{\delta}_t^{\mathsf{T}}, \qquad \frac{\partial L}{\partial E_c} = \sum_{t:\, c_t = c} \boldsymbol{\delta}_t \tag{16}$$

で，読んだ文字の行にだけ勾配がつきます．mia では a，i，m，BOS の行だけで，e，n，v の行（12個）は 0 でした．wpe も同じで，mia が使わない p4 の行（4個）は 0 です．

足し算と ReLU　黒板への書き足し $\boldsymbol{x}^{(2)} = \boldsymbol{x}^{(1)} + W_2\boldsymbol{h}$ は足し算なので，$\partial L/\partial \boldsymbol{x}^{(1)}$ には $\partial L/\partial \boldsymbol{x}^{(2)}$ がそのまま届きます（MLP を通る道の分が加わります）．黒板の足し算は，勾配が最初のメモまで弱まらずに届く通り道になっています．ReLU は $\partial h_i/\partial u_i = 1\ (u_i > 0)$，$0\ (u_i < 0)$ なので，黙っていた検出器を通る道は勾配を運びません．その位置では，黙った検出器の W_1 の行に勾配がつきません．

学習前の Attention の勾配が小さい理由　W_Q への勾配は，$\partial L/\partial \boldsymbol{x}^{(1)} \to W_O^{\mathsf{T}} \to$（割合 α を通して）$\to \boldsymbol{v}_j$ の差 $\to \boldsymbol{k}_j$ という道を通ります．学習前は W_O と W_V の成分がどれも ± 0.16 程度と小さいので，小さな数の積になり，勾配はほぼ 0 になります（図20）．

A.9　Adam（4章）

重み w ごとに，τ 回目の勾配を g_τ として（$m_0 = s_0 = 0$）

$$m_\tau = \beta_1 m_{\tau-1} + (1-\beta_1)g_\tau, \qquad s_\tau = \beta_2 s_{\tau-1} + (1-\beta_2)g_\tau^2 \tag{17}$$

$$\hat{m}_\tau = \frac{m_\tau}{1-\beta_1^\tau}, \qquad \hat{s}_\tau = \frac{s_\tau}{1-\beta_2^\tau} \tag{18}$$

$$w \leftarrow w - \eta_\tau \frac{\hat{m}_\tau}{\sqrt{\hat{s}_\tau} + \epsilon}, \qquad \eta_\tau = 0.01\Big(1 - \frac{\tau-1}{300}\Big) \tag{19}$$

$\beta_1 = 0.85$，$\beta_2 = 0.99$，$\epsilon = 10^{-8}$ です（174–182行目）．m は勾配の平均（勢い），s は勾配の2乗の平均です．1回目（$\tau = 1$）は $\hat{m}_1 = g_1$，$\hat{s}_1 = g_1^2$ なので $\hat{m}_1/\sqrt{\hat{s}_1} = \pm 1$ となり，すべての重みが $\eta = 0.01$ だけ動きます（$g = 0$ なら動かない）．$\beta_1 = 0.85$ は，およそ直近 $1/(1-0.85) \approx 7$ 回分の勾配の平均にあたります．重みの初期値は平均 0，標準偏差 0.08 の正規乱数（乱数の種 42）です．

A.10　回転しても働きが変わらないこと（5章）

Q を 4×4 の直交行列（$Q^{\mathsf{T}}Q = I$，回転や裏返し）とします．$|Q\boldsymbol{x}| = |\boldsymbol{x}|$ なので $\mathrm{rms}(Q\boldsymbol{x}) = Q\,\mathrm{rms}(\boldsymbol{x})$ です．そこで

$$E \to EQ^{\mathsf{T}},\ P \to PQ^{\mathsf{T}},\ W_Q, W_K, W_V, W_1, U \to (\cdot)\,Q^{\mathsf{T}},\ W_O, W_2 \to Q\,(\cdot) \tag{20}$$

と書きかえると，黒板の中身はすべて $Q\boldsymbol{x}$ になり，$UQ^{\mathsf{T}}Q\boldsymbol{x} = U\boldsymbol{x}$ なので得点は変わりません（計算で確かめると差は 5×10^{-15} 以下）．また，$W_Q \to RW_Q$，$W_K \to RW_K$（R は直交行列）とすると，$(R\boldsymbol{q}) \cdot (R\boldsymbol{k}) = \boldsymbol{q} \cdot \boldsymbol{k}$ なので Attention の割合は変わりません（図36）．意味を持つのは $W_Q^{\mathsf{T}}W_K$ と W_OW_V という組み合わせです．一方，16次元の側で $W_1 \to SW_1$，$W_2 \to W_2S^{\mathsf{T}}$ と回しても，一般に $\mathrm{ReLU}(S\boldsymbol{u}) \neq S\,\mathrm{ReLU}(\boldsymbol{u})$ なので得点が変わります（並べかえと正の数倍だけは例外）．計算で確かめると，ランダムな回転で得点が最大 4.6 変わりました．な

お，学習の進み方（Adam）は重み1つずつに働くので，回転した状態から学習をやり直すと同じ道筋にはなりません．ここで言う「変わらない」は，学習後の重みが表す働きについての話です．

A.11　境目で分けられる区画の数と，ReLU の前後の次元（5章）

d 次元の空間を，原点を通る n 枚の境目（一般の位置にあるもの）で分けると，区画の数は

$$C(n,d) = 2\sum_{i=0}^{d-1} \binom{n-1}{i} \tag{21}$$

です．$d=4$ のとき，$n=4$ なら $2(1+3+3+1)=16$，$n=16$ なら $2(1+15+105+455)=1152$ です．

ReLU の前の $\boldsymbol{u} = W_1\hat{\boldsymbol{x}}$ は $\hat{\boldsymbol{x}}$（4）の1次式なので，14の場面の $\boldsymbol{u}$ を並べた 14×16 の行列の階数（rank）は4以下です．計算するとちょうど4でした．ReLU の後の $\boldsymbol{h}$ では13，W_2 で黒板に戻した後は4でした．

A.12　くじ，温度，名前の確率（6章）

累積の確率 $C_j = p_1+\cdots+p_j$（$C_7=1$）を作り，$0\le r<1$ の一様乱数 r に対して $C_{j-1}\le r<C_j$ となる文字 j を選びます．r がこの区間に入る確率は区間の幅 p_j に等しいので，文字 j は確率 p_j で選ばれます．温度 T のときの確率は

$$p_j(T) = \frac{e^{z_j/T}}{\sum_k e^{z_k/T}} \tag{22}$$

です．統計力学のボルツマン分布 $p_j = e^{-\mathcal{E}_j/k_BT}/\sum_i e^{-\mathcal{E}_i/k_BT}$（$\mathcal{E}$：エネルギー，$k_B$：ボルツマン定数）で，得点 z をエネルギーに負の符号をつけたものと見ると，同じ形になります．2つの文字の確率の比は $p_j/p_k = e^{(z_j-z_k)/T}$ で，$T=0.5$ では比が2乗になります．$T\to 0$ ではいちばん得点の高い文字だけが選ばれ（貪欲法），$T\to\infty$ ではすべての文字が $1/7$ に近づきます．

名前の確率は各位置の（温度で調整した）確率の積で，N 個作ったときの個数の見込み（期待値）は $N\times P(\text{名前})$ です．温度 1 で $P(\text{aia}) = 0.509\times 0.035\times 0.992\times 0.998 \approx 0.0175$ なので，1000個で約17個の見込みです．温度 0.5 で $P(\text{emma}) = 0.096$ なので，20個では見込み約1.9個（実際は1個），1000個では約96個（実際は88個）です．

A.13　重みの数（7章）

1層の中身は，Attention の4つの表で $4d^2$，MLP の2つの表で $4d\times d + d\times 4d = 8d^2$，合わせて $12d^2$ です．層の数を $L_{\text{層}}$ として

$$\text{microgpt：}\ 2Vd + Bd + 12d^2L_{\text{層}} \tag{23}$$

$$\text{GPT-2，GPT-3：}\ Vd + Bd + L_{\text{層}}(12d^2+13d) + 2d \tag{24}$$

GPT-2 と GPT-3 では lm_head を wte と共有し，バイアスと正規化の重み（$13dL_{\text{層}}+2d$）を含みます．ミニマムな例（$V=7, d=4, B=5, L_{\text{層}}=1$）は $56+20+192=268$，microgpt（$27,16,16,1$）は 4,192，GPT-2 最小（$50{,}257, 768, 1{,}024, 12$）は 124,439,808，GPT-3（$50{,}257, 12{,}288, 2{,}048, 96$）は約 1,746 億（約1,750億）で，大部分は $12d^2L_{\text{層}}$ の部分です．

A.14　Attention の計算の量（7章）

長さ n の文で，位置 t（$0 \le t < n$）は $t+1$ 個の位置を見るので，組み合わせの数は $1+2+\cdots+n = n(n+1)/2$ です．1組ごとに q と k の内積（d 回の掛け算）が必要なので，計算の量はおよそ $n^2 d$ に比例します．学習では全位置の q と k を並べた行列 Q，K（$n \times d$）で $QK^{\mathsf{T}}/\sqrt{d}$（$n \times n$）を一度に計算し，先の位置を見ないように右上の部分を隠してから，行ごとに softmax をとります．名前や文章を作るときは，前の位置の k と v をとっておく（KV キャッシュ）ので，新しい1位置ぶんの計算だけで済みます．

A.15　layernorm と GELU（7章）

$$\mathrm{layernorm}(\boldsymbol{x}) = \boldsymbol{\gamma} \odot \frac{\boldsymbol{x}-\mu}{\sqrt{\sigma^2+\varepsilon}} + \boldsymbol{\beta}, \qquad \mathrm{GELU}(u) = u\,\Phi(u) \tag{25}$$

μ は成分の平均，σ^2 は分散，$\boldsymbol{\gamma}$，$\boldsymbol{\beta}$ は学習する重み，$\odot$ は成分ごとの積，Φ は標準正規分布の累積分布関数です．rmsnorm は平均を引かず，2乗の平均の平方根だけで割ります（microgpt は $\boldsymbol{\gamma}$ も省いています）．GELU は u が大きい正の数ならほぼ u，大きい負の数ならほぼ 0 で，0 の近くではなめらかに曲がります．ReLU はその曲がりを角にしたものです．GPT-2 の掛け算は $\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b}$（$\boldsymbol{b}$ がバイアス）ですが，microgpt は $\boldsymbol{y} = W\boldsymbol{x}$ だけです．

付録B　記号と形の一覧

本文の「式で書くと」と付録Aで使う記号です．形の欄の (4) は長さ4の縦ベクトル，7×4 は7行4列の行列です．

記号	形	プログラムの名前	意味
設定の数			
V	数	`vocab_size`	語彙の大きさ（7）
d	数	`n_embd`	黒板の幅（4）
B	数	`block_size`	一度に読める長さ（5）
$4d$	数		MLP の検出器の数（16）
n	数		1つの名前の問題の数（位置の数）
番号と位置			
c_t	番号	`token_id`	位置 t で読む文字の番号（0〜6）
t	番号	`pos_id`	読んでいる位置（0〜4）
j	番号		Attention で見に行く位置（$0 \le j \le t$）
9つの表（重み）			
E	7×4	`wte`	文字の表．E_c は c 行目（(4) として使う）
P	5×4	`wpe`	位置の表．P_t は t 行目
W_Q, W_K, W_V	4×4	`attn_wq` など	問い合わせ・見出し・中身を作る
W_O	4×4	`attn_wo`	写した中身を黒板の形に直す
W_1	16×4	`mlp_fc1`	16個の検出器（行が型紙 $\boldsymbol{\kappa}_i$）
W_2	4×16	`mlp_fc2`	検出器ごとに書き足す内容（列が $\boldsymbol{w}_i$）
U	7×4	`lm_head`	黒板を読んで得点を出す
計算の途中の数（位置 t ごと）			
$\boldsymbol{x}_t^{(0)}$	(4)		黒板：最初のメモ

記号	形	プログラムの名前	意味
$\hat{\boldsymbol{x}}_t$	(4)		大きさをそろえた黒板（Attention が読む）
$\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$	(4)	q, k, v	問い合わせ，見出し，中身
a_{tj}	数		注目の得点 $\boldsymbol{q}_t \cdot \boldsymbol{k}_j / \sqrt{d}$
α_{tj}	数	attn_weights	注目の割合．$(\alpha_{t0}, \ldots, \alpha_{tt})$ の長さは $t+1$
$\boldsymbol{o}_t$	(4)		割合で混ぜた中身
$\boldsymbol{x}_t^{(1)}$	(4)		黒板：Attention を足した後
$\boldsymbol{u}_t$	(16)		型紙との似かた（ReLU の前）
$\boldsymbol{h}_t$	(16)		検出器の点灯（ReLU の後）
$\boldsymbol{x}_t^{(2)}$	(4)		黒板：MLP を足した後（最後の黒板）
$\boldsymbol{z}_t$	(7)	logits	7つの文字の得点
$\boldsymbol{p}_t$	(7)	probs	7つの文字の確率
$\boldsymbol{y}_t$	(7)		正解（one-hot：正解の文字だけ 1）
学習と生成			
L	数	loss	損失（1つの名前の罰点の平均）
g	数	p.grad	重み1つの勾配 $\partial L / \partial w$
$\boldsymbol{\delta}$	出る側と同じ		出る側の勾配（付録A.8）
η	数	learning_rate	学習率
m, s	数	m, v	Adam の勾配の平均と2乗の平均
T	数	temperature	温度

プログラムでは Adam の2乗の平均を v と書きますが，この冊子では中身 $\boldsymbol{v}$ と区別するために s と書きました．

付録C　数を確かめるプログラム

学習後の9つの表（268個の数）を使って，本文の主な数値を確かめるプログラムです．Python の標準ライブラリだけで動きます（python3 check_weights.py）．microgpt のプログラムそのものではなく，学習後の重みを使って同じ計算を普通の小数で書き直したものです．重みは小数4桁に丸めて埋め込んであります．丸めても，本文の数値（小数2〜3桁）は変わりません．

```
# 学習後の重み（9つの表，268個の数）を使って，冊子の数値を確かめるプログラム
# 標準ライブラリだけで動く．使い方：python3 check_weights.py
# 重みは emma, ava, mia, anna を 300 回学ばせた結果（乱数の種 42）を小数4桁に丸めたもの
import math

labels = ['a', 'e', 'i', 'm', 'n', 'v', 'BOS']   # 文字（番号 0〜6）
BOS = 6

W = {
  'wte': [  # 7 行 × 4 列
    [+0.1187, -0.2073, +0.1944, +0.1829],
    [-0.1658, +0.0696, -0.0831, +0.2765],
    [-0.1263, +0.0526, +0.0470, -0.2330],
    [-0.0626, +0.0313, -0.1154, +0.0008],
    [-0.0235, +0.0118, -0.0118, -0.0079],
    [-0.1258, +0.0576, +0.0482, -0.2210],
```

```
  [-0.0873, +0.2334, -0.0201, +0.0066],
],
'wpe': [  # 5 行 × 4 列
  [+0.0531, +0.2057, +0.1428, +0.0823],
  [+0.1767, -0.0572, -0.0612, -0.1450],
  [+0.0459, -0.0422, -0.0391, +0.1508],
  [-0.2210, +0.0308, +0.1011, -0.2767],
  [-0.1761, -0.0145, +0.0891, -0.3139],
],
'lm_head': [  # 7 行 × 4 列
  [-0.3812, +0.3377, +0.0497, -0.3505],
  [-0.0609, +0.2699, -0.0053, +0.0329],
  [+0.4130, -0.1693, -0.5686, -0.5845],
  [-0.0611, +0.1983, -0.4800, +0.4055],
  [+0.3387, -0.4478, +0.0030, +0.6113],
  [+0.4077, -0.3551, +0.3694, -0.0554],
  [-0.4914, -0.4069, +0.4738, -0.3937],
],
'attn_wq': [  # 4 行 × 4 列
  [-0.4125, +0.6417, +0.1197, -0.4005],
  [-0.3637, +0.7014, -0.3220, +0.2725],
  [+0.5338, -0.4722, -0.1511, +0.2177],
  [-0.4734, +0.6193, +0.0988, -0.4275],
],
'attn_wk': [  # 4 行 × 4 列
  [-0.5562, +0.4943, +0.2970, +0.2581],
  [-0.3369, +0.0248, -0.3743, +0.4900],
  [+0.4829, -0.4844, -0.2559, -0.1962],
  [-0.4951, +0.5962, +0.4744, +0.2889],
],
'attn_wv': [  # 4 行 × 4 列
  [+0.6005, -0.6780, +0.3949, -0.0516],
  [-0.0885, +0.5224, -0.6812, +0.5473],
  [+0.4062, -0.0582, -0.4070, +0.5597],
  [+0.9592, -0.5863, +0.2306, -0.2889],
],
'attn_wo': [  # 4 行 × 4 列
  [+0.2992, -0.0970, +0.6880, +0.3685],
  [-0.6725, +0.7600, -0.1699, -0.7371],
  [+0.1718, -0.5668, -0.6748, +0.0738],
  [+0.1214, +0.1628, +0.5503, +0.0163],
],
'mlp_fc1': [  # 16 行 × 4 列
  [-0.4992, -0.0808, +0.6003, -0.6896],
  [-0.6530, -0.2929, +0.6142, -0.6396],
  [-0.4570, -0.0836, +0.3739, -0.6474],
  [-0.5344, +0.4833, -0.0633, -0.2314],
  [-0.5737, +0.5770, -0.5787, +0.0169],
  [-0.4409, -0.0948, +0.3943, -0.4949],
  [-0.6813, +0.5586, -0.6512, -0.0635],
  [-0.5163, +0.5154, -0.5965, +0.3692],
  [-0.5503, +0.2756, -0.0280, -0.4121],
  [-0.5171, -0.0830, +0.6379, -0.6297],
  [-0.5956, -0.1121, +0.5308, -0.6992],
  [-0.5842, +0.4823, -0.4258, -0.0388],
  [+0.8759, -0.6544, -0.6486, +0.0629],
```

```
    [-0.4761, -0.1328, +0.6502, -0.3374],
    [-0.2999, -0.4279, +0.6665, -0.2418],
    [-0.4293, +0.5740, -0.7439, -0.1073],
  ],
  'mlp_fc2': [  # 4 行 × 16 列
    [-0.5122, -0.6132, -0.5313, -0.2638, -0.0512, -0.6549, -0.2011, +0.0407,
     -0.3411, -0.6705, -0.6423, -0.2484, +1.1188, -0.5923, -0.2731, +0.0646],
    [+0.2735, -0.0958, +0.3021, +0.6685, +0.7675, +0.0939, +0.7315, +0.6701,
     +0.5527, +0.2345, +0.1450, +0.7225, -0.6780, -0.2790, -0.4869, +0.5966],
    [+0.4229, +0.6056, +0.3655, -0.1287, -0.1709, +0.4454, -0.2067, -0.4126,
     +0.2861, +0.4051, +0.4745, -0.2341, -0.3165, +0.5050, +0.8269, -0.1963],
    [-0.5495, -0.6540, -0.7138, -0.0559, +0.1588, -0.5442, +0.1382, +0.2337,
     -0.2172, -0.3988, -0.7276, +0.0515, +0.0175, -0.6066, -0.4811, +0.0676],
  ],
}

def lin(w, x):            # 表 w の各行と x を掛けて足す（行列とベクトルの積）
    return [sum(a * b for a, b in zip(row, x)) for row in w]

def add(x, y):
    return [a + b for a, b in zip(x, y)]

def rms(x):               # rmsnorm：2乗の平均の平方根で割って大きさをそろえる
    s = (sum(e * e for e in x) / len(x) + 1e-5) ** -0.5
    return [e * s for e in x]

def softmax(z):           # 得点を確率に直す
    m = max(z); e = [math.exp(a - m) for a in z]; t = sum(e)
    return [a / t for a in e]

def run(name):
    """名前を1回読み，位置ごとに途中の結果を記録して返す（順伝播）"""
    toks = [BOS] + [labels.index(c) for c in name] + [BOS]
    keys, vals, out = [], [], []
    for p in range(min(5, len(toks) - 1)):              # 位置は最大5つ（wpe の行の数）
        x0 = rms(add(W['wte'][toks[p]], W['wpe'][p]))    # 最初のメモ (4)
        xn = rms(x0)
        q, k, v = lin(W['attn_wq'], xn), lin(W['attn_wk'], xn), lin(W['attn_wv'], xn)  # 各 (4)
        keys.append(k); vals.append(v)
        aw = softmax([sum(a * b for a, b in zip(q, kk)) / 2 for kk in keys])   # 2 = √4. 長さ p+1
        ho = [sum(aw[t] * vals[t][j] for t in range(len(vals))) for j in range(4)]
        att = lin(W['attn_wo'], ho)                      # Attention が黒板に足す分 (4)
        x1 = add(x0, att)
        h = [max(0.0, e) for e in lin(W['mlp_fc1'], rms(x1))]   # 16個の検出器 (16)
        mlp = lin(W['mlp_fc2'], h)                       # MLP が黒板に足す分 (4)
        x2 = add(x1, mlp)
        logits = lin(W['lm_head'], x2)                   # 7つの文字の得点 (7)
        out.append(dict(read=name[:p], tok=toks[p], target=toks[p + 1], aw=aw,
                        x0=x0, att=att, mlp=mlp, h=h, logits=logits))
    return out

names = ['emma', 'ava', 'mia', 'anna']

print('1. 次の文字の予想（確率の高い2つ）と，平均の損失')
total, count = 0.0, 0
for nm in names:
```

```
    for r in run(nm):
        pr = softmax(r['logits'])
        top = sorted(range(7), key=lambda i: -pr[i])[:2]
        read = r['read'] if r['read'] else '(始め)'
        total += -math.log(pr[r['target']]); count += 1
        print(f"  {nm:5s} {read:6s} 正解 {labels[r['target']]:3s} 予想 " +
              '  '.join(f"{labels[i]}:{pr[i]:.2f}" for i in top))
print(f'  平均の損失 {total / count:.3f}')

print('2. 得点 ＝ 最初のメモ ＋ Attention ＋ MLP（emma の p3，emm の次）')
r = run('emma')[3]
parts = [lin(W['lm_head'], r[key]) for key in ('x0', 'att', 'mlp')]
for nm, vec in zip(['最初のメモ', 'Attention ', 'MLP       '], parts):
    print(f'  {nm} ' + ' '.join(f'{labels[i]}:{vec[i]:+5.2f}' for i in range(7)))
s = [a + b + c for a, b, c in zip(*parts)]
print(f'  合計と得点の差の最大 {max(abs(a - b) for a, b in zip(s, r["logits"])):.1e}')

print('3. wte の行の長さと，i と v の cos')
ln = lambda x: math.sqrt(sum(e * e for e in x))
print('  ' + ' '.join(f'{labels[i]}:{ln(W["wte"][i]):.2f}' for i in range(7)))
i_, v_ = W['wte'][2], W['wte'][5]
print(f'  cos(i, v) = {sum(a * b for a, b in zip(i_, v_)) / (ln(i_) * ln(v_)):.2f}')

print('4. 14の場面で，各検出器が点灯した回数')
seen, fire = set(), [0] * 16
for nm in names:
    for r in run(nm):
        if r['read'] in seen: continue
        seen.add(r['read'])
        for j in range(16): fire[j] += r['h'][j] > 0
print(f'  場面の数 {len(seen)}  点灯回数 ' + ' '.join(str(f) for f in fire))
```

実行結果

```
1. 次の文字の予想（確率の高い2つ）と，平均の損失
  emma  (始め)  正解 e   予想 a:0.51  m:0.30
  emma  e      正解 m   予想 m:0.94  e:0.04
  emma  em     正解 m   予想 m:0.94  e:0.04
  emma  emm    正解 a   予想 a:0.99  e:0.01
  emma  emma   正解 BOS 予想 BOS:1.00  a:0.00
  ava   (始め)  正解 a   予想 a:0.51  m:0.30
  ava   a      正解 v   予想 v:0.50  n:0.46
  ava   av     正解 a   予想 a:0.99  e:0.00
  ava   ava    正解 BOS 予想 BOS:1.00  a:0.00
  mia   (始め)  正解 m   予想 a:0.51  m:0.30
  mia   m      正解 i   予想 i:0.80  n:0.07
  mia   mi     正解 a   予想 a:0.99  e:0.00
  mia   mia    正解 BOS 予想 BOS:1.00  a:0.00
  anna  (始め)  正解 a   予想 a:0.51  m:0.30
  anna  a      正解 n   予想 v:0.50  n:0.46
  anna  an     正解 n   予想 n:0.84  v:0.11
  anna  ann    正解 a   予想 a:0.99  BOS:0.01
  anna  anna   正解 BOS 予想 BOS:1.00  a:0.00
  平均の損失 0.347
2. 得点 ＝ 最初のメモ ＋ Attention ＋ MLP（emma の p3，emm の次）
```

```
   最初のメモ a:+1.12 e:+0.12 i:+0.21 m:-0.38 n:-1.46 v:-0.64 BOS:+1.08
   Attention  a:+1.13 e:+0.66 i:-0.37 m:+0.73 n:-1.40 v:-1.39 BOS:-0.65
   MLP        a:+5.44 e:+2.35 i:-2.23 m:+0.45 n:-6.75 v:-4.34 BOS:+0.47
   合計と得点の差の最大 8.9e-16
 3. wte の行の長さと，i と v の cos
   a:0.36 e:0.34 i:0.27 m:0.13 n:0.03 v:0.27 BOS:0.25
   cos(i, v) = 1.00
 4. 14の場面で，各検出器が点灯した回数
   場面の数 14  点灯回数 8 8 8 8 8 8 8 7 8 8 8 7 6 8 7 8
```

「(始め)」の行の予想は4つの名前で同じです（まだ何も読んでいないため）．データでは a が 2/4，m と e が 1/4 で，AI は a に 0.51，m に 0.30 をつけています.

付録D　用語のまとめ

用語	元の言葉	この冊子での意味
LLM	large language model	大規模言語モデル．ChatGPT や Claude など
トークン	token	AI が1回に予想する単位．microgpt では1文字
トークナイザ	tokenizer	文字と番号を行き来する規則
語彙	vocabulary	扱うトークンの種類．ミニマムな例では7種類
BOS	beginning of sequence	名前の始まりと終わりを示す印
問題	training example	学習データから作る「次は何？」の問題
重み（パラメータ）	weight（parameter）	学習で調整される数．9つの表のマス
ハイパーパラメータ	hyperparameter	学習の前に人間が決める設定の数
行列・ベクトル	matrix, vector	表と，数の並び
黒板（残差の流れ）	residual stream	位置ごとの4つの数．書き足していく
wte / wpe	token / position embedding	文字の表 / 位置の表（最初のメモ）
rmsnorm	root mean square normalization	4つの数の大きさをそろえる
Attention	attention（注意）	前の位置の黒板を見に行くしくみ
q / k / v	query / key / value	問い合わせ / 見出し / 中身
ヘッド	head	Attention を分けて行う組
softmax	softmax	得点を，合計1の割合（確率）に直す
MLP	multi-layer perceptron	16個の検出器と書き足し（fc1，fc2）
fc1, fc2	fully connected layer	全結合層．MLP の2つの表
ReLU	rectified linear unit	負なら0にする（点灯するかどうか）
lm_head	language model head	黒板を読んで7つの得点を出す表
ロジット	logit	softmax の前の得点
学習	training	300回くり返す全体
1ステップ	step	1回の学習（順伝播から更新まで）
順伝播	forward pass	文字を読んで確率を出す計算（予想）
損失	loss	正解につけた確率の罰点（$-\ln p$）の平均
交差エントロピー	cross-entropy	この損失の数学での名前
パープレキシティ	perplexity	平均して何択で迷っているか

用語	元の言葉	この冊子での意味
勾配	gradient	重みを少し増やしたときの，損失の増え方の割合
逆伝播	backpropagation	連鎖律で出力側から勾配を伝える方法
自動微分	automatic differentiation	計算の記録をさかのぼって勾配を求めるしくみ
連鎖律	chain rule	影響の伝わり方（回転比）を掛け合わせる決まり
勾配降下法	gradient descent	勾配と反対の向きに少しずつ動かす方法
更新	update	重みを動かすこと
学習率	learning rate	1回に動かす幅の目安（0.01 → 0）
Adam	adaptive moment estimation	勢いと幅そろえを使う更新のしかた（optimizer）
生成	generation	学習後の AI に新しい名前を作らせること（推論）
温度	temperature	得点を割って，確率をとがらせたり，ならしたりする設定
貪欲法	greedy decoding	いつもいちばん確率の高い文字を選ぶ方法
KV キャッシュ	KV cache	前の位置の k と v をとっておいて使い回すしくみ
幻覚（作り話）	hallucination	もっともらしいが事実でないものを作ること
層	layer	Attention と MLP の1組．大きな AI はこれを何回も重ねる
weight tying	weight tying	wte と lm_head を同じ表で兼用すること（GPT-2）
GPU	graphics processing unit	同じ種類の計算を大量に同時に行える装置
事前学習	pre-training	大量の文章で次のトークンを当てる学習
RLHF	reinforcement learning from human feedback	人間の評価をもとにした追加の学習
スケーリング則	scaling law	重みやデータを増やすと損失が規則的に下がる傾向
解釈可能性	interpretability	AI の中で何が起きているかを読み解く研究

付録E　数値の求め方とプログラムの入手先

- **学習の設定**：学習データは emma，ava，mia，anna の4つ．黒板の幅 4，一度に読める長さ 5，ヘッドの数 1，学習 300 回，乱数の種 42．冊子『microgpt を読む』5節のミニマムな例，『小さなAIの頭の中をのぞく』，『microgpt を解剖する』Ver. 0.1 と同じ設定です．学習1回目の損失 1.9535，300回目の損失 0.3563，学習後の wte の a の行 $(+0.119, -0.207, +0.194, +0.183)$，生成した20個の名前が一致することで，同じ学習になっていることを確かめました．
- **本文の数値**：元の microgpt と同じ乱数・同じ計算（`Value` による自動微分と Adam）で

学習を再現し，途中の計算をすべて記録するプログラム（compute_all.py）で求めました．1回目の勾配の数値での確かめは，重みを 0.001 だけ動かして求めました．付録A.8 の lm_head の勾配の式は，同じプログラムで自動微分の値と比べました．

- **生成**：学習後，元のプログラムと同じ方法（random.choices，温度 0.5）で20個を作りました．温度を変えた20個は，学習直後の乱数の状態から作り直しました．1000個の集計と学習途中の生成には，乱数の種 7 を使いました（学習の乱数には影響させていません）．
- **7章の数値**：重みの数は公表されている構造から数えました．元のデータの大きさ（names.txt）と GPT-2・GPT-3 の大きさは冊子『microgpt を読む』8.1節，GPT-3 の学習データの量は Brown ほか（2020）によります．名前を増やした実験（表30）は『小さなAIの頭の中をのぞく』5節の結果です．
- **図**：fig_front.py，fig_ch1_4.py，fig_ch5_7.py で作りました．
- **microgpt のプログラム**：https://gist.github.com/karpathy/8627fe009c40f57531cb18360106ce95（2026年9月30日取得版）．ライセンスの表示がないため，この冊子にはプログラムを載せていません．