因数分解

prompt: 簡単な数式処理システムを作ってください。
入力：4x^2 -4x+1
変換後：(2x-1)^2

方針：
1．式を文字列としてトークン（お任せ）に分解
2．必要ならトークン集合を作り、
3．トークン集合を変換
4．2., 3. 等を含め、「変換規則」（あなたが用意）を適用し、(2x-1)^2を出力する。
5．なお規則は関数化し、変換は関数適用とします。つまり、変換規則が明示され、かつ、だらだらと変換しない。

よくわからないコード！

```
(*1) 文字列をトークン化：数値・識別子・演算子・括弧・冪を切り出す*)ClearAll[tokenize];
tokenize[s_String] := Module[{pat}, pat = RegularExpression[
      "([0-9]+(?:\\.[0-9]+)?)|([A-Za-z][A-Za-z0-9]*)|([\\+\\-\\*/\\^\\(\\)])"];
   StringCases[StringReplace[s, Whitespace → ""],
    pat :> First@Cases[pat, _String, ∞]]];

(*2) トークン集合（重複なし）*)
ClearAll[tokenSet];
tokenSet[tokens_List] := DeleteDuplicates[tokens];

(*3) 文字列→式：暗黙の積を明示化してから ToExpression*)
ClearAll[stringToExpr];
stringToExpr[s_String] :=
  Module[{t = StringReplace[s, Whitespace → ""]}, t = StringReplace[t,
     {(*数字と識別子の間に*を補う*)RegularExpression["(?<=\\d)(?=[A-Za-z])"] →
       "*", (*係数-変数の負号も整える（例:-4x→-4*x）*)
      RegularExpression["(?<=[\\+\\-])(\\d+)(?=[A-Za-z])"] :> "$1*",
      (*識別子の直後に'(' が来る場合などにも*を補う（今回は不要だが一応）*)
      RegularExpression["(?<=[A-Za-z])(?=\\()"] → "*"}];
   ToExpression[t, InputForm]];

(*4) 変換規則（関数化）：二次式 a x^2+b x+c で判別式 Δ=
  b^2-4 a c=0 のとき (√a x-b/(2 √a))^2 にちょうど“完成平方”で変換する。*)
ClearAll[perfectSquareRule];
perfectSquareRule[x_Symbol] := HoldPattern[a_. x^2 + b_. x + c_.] :>
   With[{Δ = b^2 - 4 a c}, If[Δ === 0, (Sqrt[a] x - b / (2 Sqrt[a]))^2,
     (*それ以外は変えない（だらだら適用しない）*)Unevaluated[a x^2 + b x + c]]];
```

```
(*5) 変換適用（単回適用）*)
ClearAll[applyTransformOnce];
applyTransformOnce[expr_, x_Symbol] := expr /. perfectSquareRule[x];

(*---デモ---*)
input = "4x^2 -4x+1";

tokens = tokenize[input];
tset = tokenSet[tokens];
expr = stringToExpr[input];

out = applyTransformOnce[expr, x];

(*確認用の表示（必要なら）*)
tokens
tset
expr // InputForm
out // InputForm
```

生成AIは数学の問題は「不得意」。特に帰納的推論。そして演繹的推論も。ただし、たとえば二次方程式の解法（因数分解）の「規則」は知っている。規則の発見、それにもとづくコーディングをやらせたらどうか。

prompt: あなたが今のアプローチに固執するかぎり、駄目ですね。 いきなりWolfram Languageにするのではなく 「パターンの変換」を主体とする自然言語でルールを書いてもらえますか。

二次式の因数分解ルール（自然言語での「パターン→パターン変換」）

入力対象

- 単一変数 x の二次式

$$ax^2 + bx + c$$

変換ルール

1. **完全平方のケース**
   - **パターン:** 判別式 $\Delta = b^2 - 4ac = 0$
   - **変換:**

$$ax^2 + bx + c \quad \longrightarrow \quad \left(\sqrt{a}x - \frac{b}{2\sqrt{a}}\right)^2$$

2. **有理数で因数分解できるケース**
   - **パターン:** 判別式 $\Delta \geq 0$ かつ Δ が「有理数の平方」
   - **変換:**
     根を

$$r_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

     と計算して

$$ax^2 + bx + c \quad \longrightarrow \quad a(x - r_1)(x - r_2)$$

3. **それ以外のケース**
   - **パターン:** 上記 1,2 に当てはまらない
   - **変換:**
     変換せず、元の式をそのまま返す。

上記定式化を受けて松田が書いたコード

In[•]:=
```
factor[expr_] := Module[{a, b, c},
  discriminant[a_, b_, c_] := b^2 - 4 a c;
  {c, b, a} = CoefficientList[expr, x];
  isRatio[q_] := MatchQ[q, _Rational | _Integer];

  Which[
   discriminant[a, b, c] == 0,
   (Sqrt[a] x - b / (2 Sqrt[a]))^2,

   (delta = discriminant[a, b, c]) ≥ 0 && isRatio[Sqrt[delta]],
   r1 = (-b + Sqrt[delta]) / (2 a);
   r2 = (-b - Sqrt[delta]) / (2 a);
   a (x - r1) (x - r2),

   True,
   expr
  ]]
```

```
factor[4 x^2 - 4 x + 1]
```

Out[•]=

$$(1 + 2\,x)^2$$

In[•]:=
```
factor[6 x^2 + x - 2]
```

Out[•]=

$$6\left(-\frac{1}{2} + x\right)\left(\frac{2}{3} + x\right)$$

In[•]:=
```
factor[x^2 + 1]
```

Out[•]=

$$1 + x^2$$

このコードを受けてChatGPT5が出したコード

```
ClearAll[isRatio, discriminant, factor];

isRatio[q_] := MatchQ[q, _Rational | _Integer];

discriminant[a_, b_, c_] := b^2 - 4 a c;

factor[expr_, x_Symbol] := Module[{a, b, c, delta, r1, r2},
   (*まずは二次式だけ対象*)
   If[! (PolynomialQ[expr, x] && Exponent[expr, x] == 2), Return[expr]];

   {c, b, a} = CoefficientList[Expand[expr], x];
   delta = discriminant[a, b, c];
   Which[
    delta == 0, (Sqrt[a] x - b / (2 Sqrt[a]))^2,

    delta ≥ 0 && isRatio[Sqrt[delta]],
    r1 = (-b + Sqrt[delta]) / (2 a);
    r2 = (-b - Sqrt[delta]) / (2 a);
    a (x - r1) (x - r2),

    True, expr]];

(*動作確認*)
factor[4 x^2 - 4 x + 1, x]   (* →(2 x-1)^2*)
factor[6 x^2 + x - 2, x]    (* →6 (x-1/2) (x+2/3)*)
factor[x^2 + 1, x]       (* →x^2+1*)
```

Transformer

prompt: 数式処理の 1 つである因数分解は部分的にMathematicaの組み込み関数を使うことで、規則をコード化できた。 これに対し生成AIをモデル化するのは至難の技。文生成に限定して、いくつかの単語とその埋め込みから、次の単語を想起する部分のみを形式化することは可能？

```
(* ==============================
 Minimal "Next-Word Recall" from Embeddings==============================*)
ClearAll[makeToyEmb, cosSim, encodeContext, logitsFromEmb, softmaxAssoc,
  bigramCounts, bigramCondProb, mergeLogitsWithBigram, nextToken, generate];

(*0) お試し用の語彙と埋め込み（固定乱数で再現可能）*)
makeToyEmb[words_List, dim_ : 8, seed_ : 42] := Module[{vecs}, SeedRandom[seed];
   AssociationMap[
    Normalize@RandomReal[NormalDistribution[0, 1], dim] &, words]];

(*1) コサイン類似度（説明用：今回は内積をロジットに使う）*)
```

```
cosSim[u_, v_] := (u.v) / (Norm[u] Norm[v] + $MachineEpsilon);

(*2) 文脈ベクトル：過去単語の埋め込みを指数減衰で合成*)
encodeContext[history_List, vecs_Association, decay_ : 0.8] :=
  Module[{vs = Lookup[vecs, history, Nothing], w},
   If[vs === {}, Return[ConstantArray[0., Length[First[Values[vecs]]]]]];
   w = decay^Range[Length[vs], 1, -1];
   (*新しい語ほど重み大*)Normalize[Total[MapThread[#1 * #2 &, {vs, w}]]]];

(*3) ロジット：文脈ベクトルと各語彙ベクトルの内積*)
logitsFromEmb[ctx_, vecs_Association] :=
  AssociationMap[ctx.vecs[#] &, Keys[vecs]];

(*4) ソフトマックス（Associationで返す）*)
softmaxAssoc[logits_Association, temp_ : 1.] :=
  Module[{keys = Keys[logits], vals = Values[logits] / temp, exps},
   exps = Exp[vals - Max[vals]];
   AssociationThread[keys → exps / Total[exps]]];

(*5) 簡易バイグラム：学習用コーパスから頻度→条件付き確率 P(w|prev)*)
bigramCounts[corpus_List] := Module[{pairs},
   pairs = Flatten[Table[Partition[sent, 2, 1], {sent, corpus}], 1];
   Counts[pairs]];

bigramCondProb[counts_Association] :=
  Module[{byPrev, cond}, byPrev = GroupBy[Normal[counts], First/*First → Last];
   cond =
    Association@KeyValueMap[Function[{prev, m}, With[{total = Total[Values[m]]},
        prev → Association@KeyValueMap[#1 → #2 / total &, m]]], byPrev];
   cond];

(*6) 埋め込みロジットとバイグラムの補間（対数空間で線形結合）*)
mergeLogitsWithBigram[embLogits_Association,
   history_List, condAssoc_Association, λ_ : 0.3, eps_ : 1.*^-9] :=
  Module[{prev = If[history === {}, None, Last[history]], logPbg, keys},
   keys = Keys[embLogits];
   logPbg = AssociationMap[
     If[KeyExistsQ[condAssoc, prev] && KeyExistsQ[condAssoc[prev], #],
       Log[condAssoc[prev, #] + eps], (*P_bg(w|prev)*)Log[eps]] &, keys];
   AssociationThread[keys → ((1 - λ) * Values[embLogits] + λ * Values[logPbg])]];

(*7) 次語予測（argmax または確率サンプリング）*)
nextToken[history_List, vecs_Association;
   cond_ : <| |>, opts : OptionsPattern[{"Temperature" → 1.,
      "LambdaBigram" → 0.3, "Mode" → "ArgMax", "TopK" → All}]] :=
  Module[{temp = OptionValue["Temperature"], λ = OptionValue["LambdaBigram"],
    mode = OptionValue["Mode"], topk = OptionValue["TopK"], ctx, embLogits,
```

```
    mixedLogits, cand, probs, pick}, ctx = encodeContext[history, vecs];
   embLogits = logitsFromEmb[ctx, vecs];
   mixedLogits = If[cond === <||>, embLogits,
     mergeLogitsWithBigram[embLogits, history, cond, λ]];
   (*Top-k をかける場合*)cand = If[topk === All, mixedLogits,
     With[{sorted = Reverse@SortBy[Normal[mixedLogits], Last]},
      Association@Take[sorted, Min[topk, Length[sorted]]]]];
   probs = softmaxAssoc[cand, temp];
   Which[mode === "ArgMax", First@Keys@MaximalBy[Normal[probs], Last],
    mode === "Sample", RandomChoice[Normal[probs]],
    True, First@Keys@MaximalBy[Normal[probs], Last]]];

(*8) 生成（n語つなげるユーティリティ）*)
generate[seed_List, n_Integer, vecs_Association;
   cond_ : <||>, opts : OptionsPattern[{"Temperature" → 1.,
      "LambdaBigram" → 0.3, "Mode" → "ArgMax", "TopK" → All}]] :=
  Module[{hist = seed, w}, Do[w = nextToken[hist, vecs;
      cond, FilterRules[{opts}, Options[nextToken]]];
    hist = Append[hist, w], {n}];
   hist];

(* ==============================デモ==============================*)

(*語彙と埋め込み*)
vocab =
  {"the", "a", "cat", "dog", "sat", "barked", "on", "in", "the", "mat", ".", ","};
vocab = DeleteDuplicates[vocab];        (*重複は消す*)
vecs = makeToyEmb[vocab, 8, 123];

(*小さなコーパスからバイグラム（任意。埋め込みのみでも動く）*)
corpus =
  {{"the", "cat", "sat", "on", "the", "mat", "."}, {"the", "dog", "barked", "."},
   {"a", "dog", "sat", "in", "the", "mat", "."}    (*変な文だけど頻度学習用の玩具*)};
bgCounts = bigramCounts[corpus];
bgCond = bigramCondProb[bgCounts];

(*例1:埋め込みのみ（λ=0）で次語想起*)
nextToken[{"the", "cat"}, vecs; <||>, "LambdaBigram" → 0., "Mode" → "ArgMax"]

(*例2:埋め込み+バイグラム補間（λ=0.3）*)
nextToken[{"the", "cat"}, vecs;
 bgCond, "LambdaBigram" → 0.3, "Mode" → "ArgMax"]

(*例3:生成（5語）*)
generate[{"the"}, 5, vecs;
 bgCond, "LambdaBigram" → 0.3, "Mode" → "Sample", "Temperature" → 0.9, "TopK" → 5]
```

prompt:transformerの本質を ・知識の表現と変換（計算）に特化して簡潔に定式化して。Attentionはまずはパス。

1. 知識の表現 (Representation)

- 各 **トークン**（単語やサブワード）は **埋め込みベクトル** $e_i \in \mathbb{R}^d$ として表現される。
- 文全体は $\{e_1, e_2, \ldots, e_n\}$ の列ベクトル集合として初期表現される。

$$\text{Input: } (t_1, t_2, \ldots, t_n) \quad \longrightarrow \quad (e_1, e_2, \ldots, e_n)$$

---

2. 知識の変換 (Transformation)

Transformer の層は、入力表現を次の形で繰り返し変換する：

$$h^{(\ell)} = \text{Transform}(h^{(\ell-1)}), \quad \ell = 1, 2, \ldots, L$$

ここで $h^{(0)} = e_i$（初期埋め込み）。

最小形に削ぎ落とすと、**Transform** は以下の2つの写像の合成：

1. **線形変換（特徴空間の再編成）**

$$z = W h^{(\ell-1)} + b$$

2. **非線形変換（活性化による写像拡張）**

$$h^{(\ell)} = \sigma(z)$$

この繰り返しで、「知識表現が別の空間へ写像され、抽象度が上がっていく」と捉えられる。

---

3. 出力生成 (Computation → Symbol)

最後に得られた表現から、語彙ベクトル集合 $V = \{v_w\}$ との**相関（内積）**を計算し、次の単語を決定する：

$$P(w \mid \text{context}) = \frac{\exp(v_w \cdot h^{(L)})}{\sum_{w' \in V} \exp(v_{w'} \cdot h^{(L)})}$$

prompt:「この抽象定式を Wolfram Language の **超最小 Transformer モデル（Attention 無し版）**に落として」。 なお
・トップ関数は transformer
・実装について。１，２．３．それぞれを関数化。
・できるだけWolfram Languageの組み込み関数を使うこと。要はシンプル＆スマート！
・バグが怖いので、１，２．３．個別にテストできるテストデータの用意もお願い。

```
(* ============================================*)
(*超最小 Transformer（Attention なし）*)(*1) 表現:tokens→埋め込み*)
(*2) 変換:線形+非線形 (+残差)×L 層*)(*3) 出力:語彙ベクトルと内積→Softmax*)
(* ============================================*)
ClearAll[initTransformer,(*パラメタ初期化（語彙・埋め込み・重み）*)
  embedTokens,(*1) 表現*)transformOnce,(*2) 変換（1ブロック）*)
  transformRep,(*2) 変換（L層まとめ）*)outputProbs,(*3) 出力（確率分布）*)
  transformer      (*トップ関数：次語推定または生成1ステップ*)];
```

```
(*----Utility:Softmax と GELU（Tanh 近似）----*)
softmax[v_List] := Module[{mx = Max[v], e = Exp[v - Max[v]]}, e / Total[e]];
gelu[x_] := 0.5 x (1 + Tanh[Sqrt[2 / Pi] (x + 0.044715 x^3)]);
geluMat[m_?MatrixQ] := Map[gelu, m, {2}];

(*----0) 初期化：語彙・埋め込み・重み----*)
initTransformer[vocab_List, d_Integer : 16,
    width_Integer : 32, layers_Integer : 2, seed_ : 1234] :=
  Module[{V = DeleteDuplicates[vocab], E, W1, b1, W2, b2}, SeedRandom[seed];
    E = Map[Normalize, RandomReal[NormalDistribution[0, 1], {Length[V], d}], {1}];
    (* |V|×d*)W1 = Table[RandomReal[
        NormalDistribution[0, 1 / Sqrt[d]], {d, width}], {layers_Integer}];
    b1 = Table[ConstantArray[0., width], {layers_Integer}];
    W2 = Table[RandomReal[
        NormalDistribution[0, 1 / Sqrt[width]], {width, d}], {layers_Integer}];
    b2 = Table[ConstantArray[0., d], {layers_Integer}];
    <|"Vocab" → V, "VocabIndex" → AssociationThread[V → Range[Length[V]]],
     "Embed" → E, "W1" → W1, "b1" → b1, "W2" → W2, "b2" → b2,
     "Dim" → d, "Width" → width, "Layers" → layers_Integer|>];

(*----1) 表現：トークン列→埋め込み列（T×d）----*)
embedTokens[tokens_List, params_Association] :=
  Module[{idx = Lookup[params["VocabIndex"], tokens,
      Missing["KeyAbsent", #] & /@ tokens]}, If[AnyTrue[idx, MissingQ],
    Message[embedTokens::unk, Pick[tokens, Map[MissingQ, idx]]];
    Return[$Failed]];
   params["Embed"][[idx]] (*行方向が時系列、列方向が特徴次元*)];
embedTokens::unk = "Unknown token(s): `1`.";

(*----2) 変換（1層）：(T×d)→(T×d) 残差つきFFN----*)
transformOnce[H_?MatrixQ, w1_?MatrixQ, b1_?VectorQ,
   w2_?MatrixQ, b2_?VectorQ] := Module[{z1, a1, z2}, z1 = H.w1 + b1;
   (*Wolfram は行ベクトルへのベクトル加算を各行にブロードキャスト*)a1 = geluMat[z1];
   z2 = a1.w2 + b2;
   H + z2          (*残差（Attention を外した最小 Transformer らしさ）*)];

(*----2) 変換（L層スタック）----*)
transformRep[H0_?MatrixQ, params_Association] :=
  Module[{H = H0, L = params["Layers"]}, Do[H = transformOnce[H, params["W1"][[ℓ]],
      params["b1"][[ℓ]], params["W2"][[ℓ]], params["b2"][[ℓ]]], {ℓ, 1, L}];
   H];

(*----3) 出力：最後のトークン表現 h_T と語彙埋め込み E でロジット→Softmax----*)
outputProbs[H_?MatrixQ, params_Association, temperature_ : 1.] :=
  Module[{h = Last[H], logits, probs}, logits = params["Embed"].h;
   (* |V|次元ロジット（出力重みを埋め込みとタイド）*)
```

```
    probs = softmax[logits / temperature];
    AssociationThread[params["Vocab"] → probs]];

(*----Top:transformer（次語を1語予測）----*)
Options[transformer] = {"Mode" → "ArgMax", "Temperature" → 1., "TopK" → All};
transformer[contextTokens_List, params_Association, OptionsPattern[]] :=
  Module[{H0, H, pAssoc, mode = OptionValue["Mode"],
    topk = OptionValue["TopK"], temp = OptionValue["Temperature"], next},
   H0 = embedTokens[contextTokens, params];
   If[H0 === $Failed, Return[$Failed]];
   H = transformRep[H0, params];
   pAssoc = outputProbs[H, params, temp];
   (*TopK 表示用にカット（返り値には確率連想配列も含める）*)If[topk =!= All, pAssoc =
     Association@Take[Reverse@SortBy[Normal[pAssoc], Last], UpTo[topk]]];
   next = Which[mode === "ArgMax", First@Keys@MaximalBy[Normal[pAssoc], Last],
     mode === "Sample", RandomChoice[Normal[pAssoc]],
     True, First@Keys@MaximalBy[Normal[pAssoc], Last]];
   <|"NextToken" → next, "Probs" → pAssoc|>];

(* ============================================*)
(*テストデータ*)
(* ============================================*)

vocab =
  {"the", "a", "cat", "dog", "sat", "barked", "on", "in", "the", "mat", ".", ","};
params = initTransformer[vocab, eight = 8, width = 16, layers = 2, 2025];

(*---1) 表現テスト---*)
tokens1 = {"the", "cat"};
emb1 = embedTokens[tokens1, params];
(*期待:Dimensions[emb1]=={2,params["Dim"]}*)
embDims = Dimensions[emb1]; (* →{2,8} など*)

(*---2) 変換テスト（形状保存& 残差動作）---*)
H1 = transformRep[emb1, params];
H1Dims = Dimensions[H1];  (*期待:{2,8}*)
(*変換前後の距離（残差なので完全には壊れない）：*)
deltaNorm = Norm[H1 - emb1]; (* >0 かつ有限*)

(*---3) 出力テスト（確率和=1）---*)
probs1 = outputProbs[H1, params, 1.0];
sumProb = N@Total[Values[probs1]]; (*期待:ほぼ 1.0*)

(*---4) トップ関数（次語予測、Top-5 を観察）---*)
pred = transformer[{"the", "cat"}, params, "Mode" → "ArgMax", "TopK" → 5];
(*pred["NextToken"] は次語、pred["Probs"] は上位候補（TopK 指定）*)
```

```
{embDims, H1Dims, deltaNorm, sumProb, pred}
```

prompt: transformerの基礎モデルについて。 1. 知識の表現、2. 知識の変換、3.出力生成 の数式を直接、関数に写像する形で実装して。その際、以下に注意。
・Wolfram Languageなので記号計算可能を中核において。
・1. 、2. はあくまで関数ベースで作り、3.で初めて具体例を使う。
・unknownはないと仮定。
・いきなりコードを作らず、疑似コードで概要を見せて。あるいは、仕様確認があれば是非！

設計の芯（数式）

- 語彙 $\mathcal{V}$、埋め込み写像 $E : \mathcal{V} \to \mathbb{R}^d$
- 入力列 $(t_1, \ldots, t_n)$ の埋め込み列 $(e_1, \ldots, e_n)$ with $e_i = E(t_i)$
- 位置ごと変換（Attention無し）：各 $h \in \mathbb{R}^d$ に同じ写像を適用

$$\mathrm{FFN}_\ell(h) \;=\; h \;+\; W_2^{(\ell)}\,\phi\big(W_1^{(\ell)}\,h + b_1^{(\ell)}\big) \;+\; b_2^{(\ell)}$$

- L 層合成： $h_i^{(0)} = e_i, \quad h_i^{(L)} = \big(\mathrm{FFN}_L \circ \cdots \circ \mathrm{FFN}_1\big)\big(h_i^{(0)}\big)$
- 出力（語彙行列 $O \in \mathbb{R}^{|\mathcal{V}| \times d}$ を用意。tiedなら $O = E$ 行列化）：
  末尾表現 $h_T = h_n^{(L)}$ に対し

$$z = O\,h_T, \qquad p(w \mid \mathrm{context}) = \frac{\exp(z_w/\tau)}{\sum_{w'} \exp(z_{w'}/\tau)}$$

prompt: 直前のコードに関して。
・警告／エラー等を外して。
・システムとしては重要な機能だが、ここで警告を出されても実際、何もできない。
・エラーを起こさないロジックだけとし、コード全体を簡潔に作って。

prompt: 埋め込みを
embAssoc = <| “the” -> {1., 0., 0., 0.}, “cat” -> {0., 1., 0., 0.}, “sat” -> {0., 0., 1., 0.}|>;
HB = Represent[{“the”, “cat”, “sat”, “the”}, embAssoc]; に限定し、上記コードをさらに単純に。

最終解

(1) 知識の表現

In[•]:=
```
(* =====Represent（Association専用・超簡潔）=====*)
ClearAll[Represent];
Represent[tokens_List, emb_Association] :=
 Developer`ToPackedArray@(emb /@ tokens)

(*----ミニテスト----*)
embAssoc = <|
    "the" → {1., 0., 0., 0.},
    "cat" → {0., 1., 0., 0.},
    "sat" → {0., 0., 1., 0.}|>;

HB = Represent[{"the", "cat", "sat", "the"}, embAssoc];

{Dimensions@HB, HB[[1]], HB[[2]], HB[[-1]]}
(*期待:{{4,4},{1.,0.,0.,0.},{0.,1.,0.,0.},{1.,0.,0.,0.}}*)
```

Out[•]=
```
{{4, 4}, {1., 0., 0., 0.}, {0., 1., 0., 0.}, {1., 0., 0., 0.}}
```

(2) 知識の変換

In[•]:=
```
(* =====TransformRep（残差付きFFN×L層・簡潔版）=====*)
ClearAll[TransformRep];

(*行ごとのバイアス加算（ブロードキャスト）*)
rowBiasAdd[m_?MatrixQ, b_?VectorQ] := m + ConstantArray[b, Length[m]];

(*
params は Association:
    "W1"→{W1[1],...,W1[L]}（各 d×w）
    "b1"→{b1[1],...,b1[L]}（各 長さ w）
     "W2"→{W2[1],...,W2[L]}（各 w×d）
      "b2"→{b2[1],...,b2[L]}（各 長さ d）
*)
TransformRep[H0_?MatrixQ, params_Association] :=
  Module[{H = H0, L = Length@params["W1"]},
    Do[
     H = H + rowBiasAdd[Tanh[rowBiasAdd[H.params["W1"][[ℓ]], params["b1"][[ℓ]]].
          params["W2"][[ℓ]], params["b2"][[ℓ]]], {ℓ, 1, L}];
    H];
```

In[•]:=
```
SeedRandom[2025];
d = 4; w = 6; L = 2;
params = <|
    "W1" → Table[RandomReal[NormalDistribution[0, 1 / Sqrt[d]], {d, w}], {L}],
    "b1" → Table[ConstantArray[0., w], {L}],
    "W2" → Table[RandomReal[NormalDistribution[0, 1 / Sqrt[w]], {w, d}], {L}],
    "b2" → Table[ConstantArray[0., d], {L}]|>;

H2 = TransformRep[HB, params];

{Dimensions@HB, Dimensions@H2, Norm[H2 - HB] > 0}
(*期待:{{4,4},{4,4},True}*)
```

Out[•]=
```
{{4, 4}, {4, 4}, True}
```

(3) 出力生成

```
(* =====出力生成（Softmax）=====*)
ClearAll[SoftmaxVec, OutputProbs, PredictNext];

(*安定化した Softmax（定数シフト）*)

(*
   --エラー版--
   SoftmaxVec[v_List]:=Module[{m=Max[v],e=Exp[v-m]},e/Total[e]];
*)
SoftmaxVec[v_List] := Module[{m, e},
   m = Max[v]; e = Exp[v - m]; e / Total[e]];

(*H:(T×d) 表現行列,emb:<|token→dベクトル|>*)
OutputProbs[H_?MatrixQ, emb_Association, temperature_ : 1.] :=
  Module[{hT, O, vocab, logits, probs},
   hT = Last[H]; (*末尾トークンの表現（長さ d）*)
   O = Values[emb]; (*語彙×次元 行列（tied weights）*)
   vocab = Keys[emb]; (*語彙の順序*)
   logits = O.hT; (* |V|次元ロジット*)
   probs = SoftmaxVec[logits / temperature];
   AssociationThread[vocab → probs]];

(*お好みで：次語1語を返すユーティリティ（ArgMax/Sample）*)
PredictNext[
   H_?MatrixQ,
   emb_Association, mode_ : "ArgMax",
   temperature_ : 1.] :=
  Module[
   {p = OutputProbs[H, emb, temperature]},
   Which[
    mode === "Sample", RandomChoice[Normal[p]],
    True, First@Keys@MaximalBy[Normal[p], Last]]];
```

In[•]:=
```
(*----確率と次語----*)
probs = OutputProbs[H2, embAssoc, 1.0];
next1 = PredictNext[H2, embAssoc, "ArgMax"];
next2 = PredictNext[H2, embAssoc, "Sample", 0.8];

(*1) probs は Association？キーは語？和は1？*)
{Head[probs] === Association, Keys@probs, N@Total@Values@probs}

(*2) next1/next2 は“語（文字列）”？*)
{Head[next1] === String, Head[next2] === String, next1, next2}

(*3) Top-3 確率（Associationのまま上位3）*)
top3 = Association@Take[Reverse@SortBy[Normal@probs, Last], UpTo[3]]

(*4) 参考：確率の数値ベクトルを見たい場合だけ Values を呼ぶ*)
Values@probs (*これは {0.608973,0.196503,0.194524} のような“数値ベクトル”*)
```

Out[•]=
```
{True, {the, cat, sat}, 1.}
```

Out[•]=
```
{True, False, the, the → 0.674153}
```

Out[•]=
```
⟨| the → 0.608973, cat → 0.196503, sat → 0.194524 |⟩
```

Out[•]=
```
{0.608973, 0.196503, 0.194524}
```