第二章　大規模言語生成モデルとTransformer基盤

Ø　大規模言語生成モデルのワードプロセッサ方法

第1章では、ChatGPTのような大規模言語生成モデルの入力と出力について説明しました。基本的に、文字をモデルに入力し、その後、モデルが続く文字を予測するという、「文字接龍」のような言語モデルです。

しかし、文字がモデルに入力される前には、別のデータ形式に変換される必要があります。また、モデルが計算処理を終えた後も、その結果を逆変換し、文字形式に戻してユーザーにフィードバックします。この変換は、TokenizerとEmbeddingの2つのステップを含みます。まずは、この２つについてを紹介します。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Tokenizerとは

ChatGPTの公式サービスは現在、利用者に対して料金を請求しています。主な料金方式は、ユーザーが使用したトークンの数に基づいており、使用するトークン数が多いほど、料金も高くなります。

> *例えば、ユーザーがあるテキストを入力した場合、そのテキスト（句読点や特殊記号を含む）が50文字であっても、30トークンを消費することがあります。ChatGPTがその入力に基づいて回答を生成すると、合計で200トークンを使用し、結果として文字数が300文字になることもあります。そうすると、ユーザーが消費するトークン数は30トークン（入力） + 200トークン（出力） = 230トークンとなります。それでは、トークンとは何でしょうか？*

トークン（token）は、NLPニューラルネットワークモデルがユーザー入力を受け取る際の最小単位です。トークン自体は文字の組み合わせであり、例えば、英単語「cat」や日本語の単語「猫」などがトークンとして扱われます。

ユーザーの入力テキストをトークンのシーケンスに変換するプロセスを「トークナイザー（Tokenizer）」と呼びます。このプロセスは二つの部分から成り立っています。一つは、テキストをトークンに変換する部分（ChatGPTに入力する前に設定される）、もう一つは、トークンをテキストに戻す部分、すなわち逆変換（ChatGPTモデルの出力後に設定される）です。

第二章　大規模言語生成モデルとTransformer基盤

Ø　BPEアルゴリズム

トークナイザーの現在主に実装方法は、バイトペアエンコーディング（BPE、Byte Pair Encoding）アルゴリズムです。これはChatGPTでも採用されているアルゴリズムです。BPEアルゴリズムは、トークンの語彙（Vocabulary）に基づいて、入力テキストを複数のトークンに分解します。その中で、各トークンは語彙に存在します。

具体的に以下のテキストがモデルに入力される場合を考えます：

The newest car has a lower price and the lowest fuel.

このテキストには53文字が含まれています（文字、空白、句読点、そしてキーボードで入力可能な特殊記号を含む）。

BPEアルゴリズムは、上記のトークン語彙に基づいてテキストをマッチングし、テキストから語彙に存在するトークンを分解して、テキストを次のような形式に変換します：

```
#The, #new, est, #car, #has, #a, #low, er, #price, #and, #the, #low, est,
#fuel, .
```

第二章　大規模言語生成モデルとTransformer基盤

Ø　BPEアルゴリズム

この例では、テキストが15個のトークンに分解されています。英語の単語はスペースで区切られるため、各単語の最初の文字には単語の開始を示す#が付けられています。これはスペースとして理解することができ、#が付いていないトークンは独立した単語として存在しません。一部の単語は複数の部分に分解されます。例えば、「newest」は#newとestに分解されます。その後、モデルはこのようなトークンデータを受け取り、さらに処理を行います。

上記の例からわかるように、トークンは通常、高頻度の文字の組み合わせで構成されており、これらのトークンには一定の意味があります。例えば、「newest」は#newとestに分解され、前半部分は単語の語根であり、後半部分は英語の形容詞の最上級を示します。

同様に、GPTモデルがユーザーの質問に回答し、出力する際にも、最初にトークンのシーケンスを出力し、その後、そのトークンのシーケンスを通常の自然言語テキストに逆変換します。この操作は「**De-tokenization**」と呼ばれ、トークナイゼーションと完全に逆の操作です。

第二章　大規模言語生成モデルとTransformer基盤

Ø　BPEアルゴリズム

BPE辞書を訓練するための手順は以下の通りです：

1. データ前処理：テキストデータを準備し、クリーンアップします。
2. 辞書の初期化：テキストを文字レベルで分割し、初期辞書を作成します。
3. 頻度統計：隣接する文字ペアの頻度を計算します。
4. マージ操作：最も頻度の高い文字ペアを選び、辞書を更新します。
5. 反復プロセス：頻度統計とマージ操作を繰り返し、設定された辞書サイズまたは他の終了条件に達するまで続けます。
6. 辞書の生成：訓練が完了したら、得られた辞書を使用してテキストをエンコードします。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Tokenizerの利点

英語の単語の中で、最も頻繁に出現する5000語が実際の使用量の90%を占めています。一方、非常に低頻度の単語は数が非常に多いですが、全体の実際の使用量は2%を超えません。これが自然言語の長尾効果であり、この現象は他の言語にも見られます。

非常に低頻度の単語や文字を直接トークンとして扱うと、データ量の不足を意味し、それらが語彙表に含まれていない可能性があります（Out Of Vocabulary, OOV）。これはNLPモデルの性能に大きな影響を与える可能性があります。したがって、**Tokenizerを導入し、BPEアルゴリズムを使用することで、低頻度の単語がトークンとして扱われるのを避けることができます。**

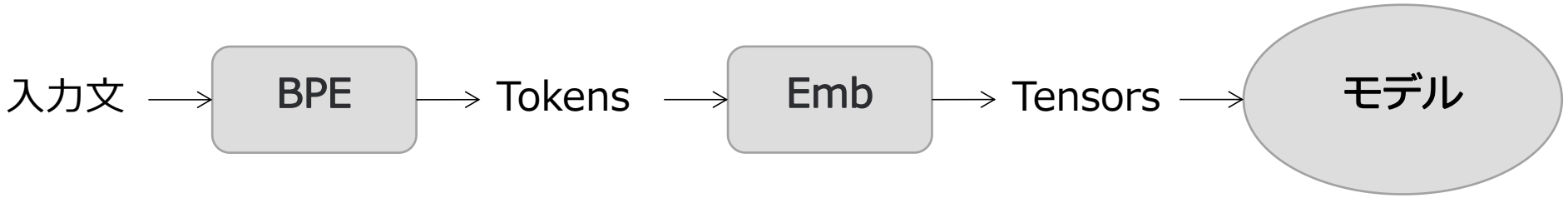
同時に、**Tokenizerは多言語サポートにも役立ちます。**初期の NLP 神経ネットワークモデルは機能が非常に限られており、特定の言語のみをサポートしていました。英語用のテキスト分類モデルは、日本語のテキスト分類には対応できませんでした。しかし、BPE アルゴリズム、特に Byte-level BPE アルゴリズムの設計により、一つの語彙表に複数の言語の文字を含むことができ、モデルの多言語処理機能をサポートします。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Embeddingとは

もでるの入力テキストがトークンに変換された後、さらにそのトークンをテンソルに変換する必要があります。このプロセスを単語埋め込み（Embedding）と呼び、埋め込み（embedding）とは変換後に得られるテンソルそのものを指します。

神経ネットワークにおいて、テンソル（Tensor）は多次元配列を指し、大量のデータを保存および処理するための基本的なデータ構造です。テンソルは一般的に浮動小数点数（小数のコンピュータ表示形式）を要素として持ちます。



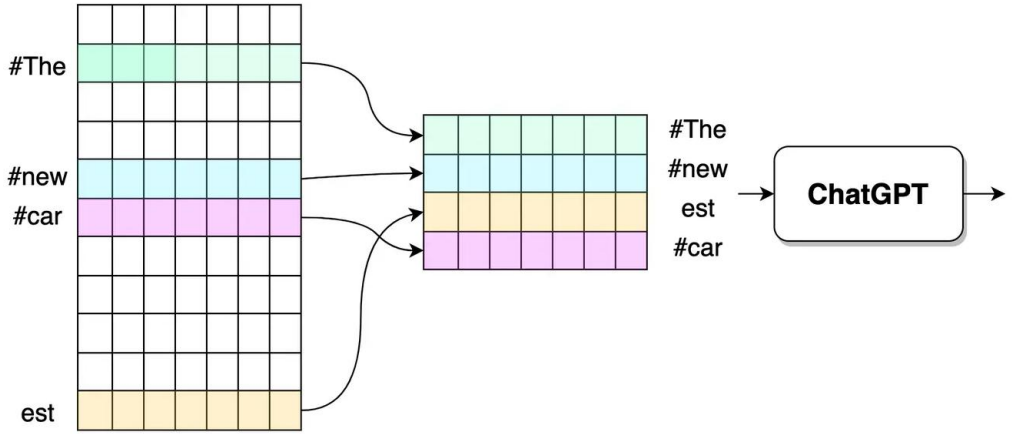
入力文からモデルへのEncodeプロセス

第二章　大規模言語生成モデルとTransformer基盤

```
#The, #new, est, #car, #has, #a, #low, er, #price, #and, #the, #low, est,
#fuel, .
```

前述の文を例にとると、トークン語彙表（Vocabulary）の総数がNであると仮定します。各トークンはM次元の浮動小数点数テンソルで表されており、各トークンはテンソルの1行に対応しています。これがそのトークンの埋め込み（embedding）表現です。

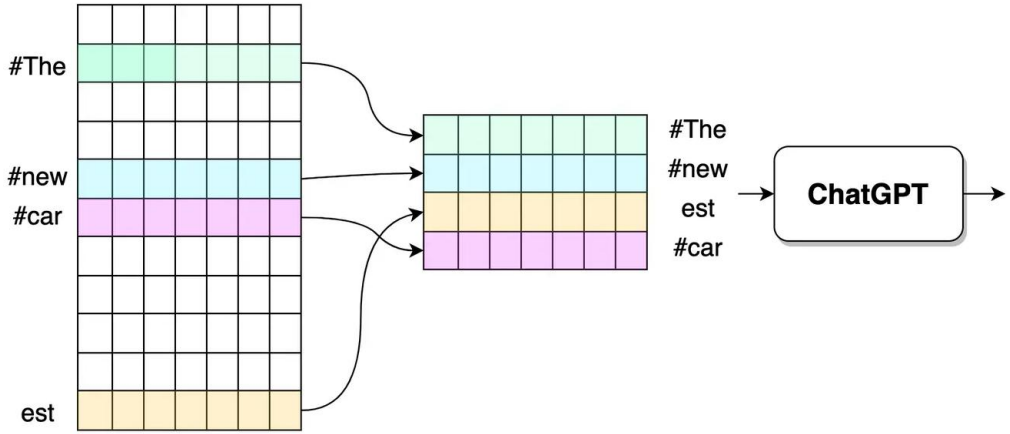
このデータセットはモデルに入力され、モデルの訓練や使用に利用されます。すべての語彙表は N × M 次元のテンソルを構成します。以下の図の左側の行列のようになります。



第二章　大規模言語生成モデルとTransformer基盤

例として挙げた前の4つのトークンに基づいて、それぞれのトークンに対応する埋め込み（embedding）を抽出し、トークンの順序に従って並べることで、テンソルのセットを作成します。このテンソルセットをモデルに入力して操作を行います。**図中の白い部分は、語彙表にある単語がトークンのシーケンスにマッチしていないことを示しています**。言い換えれば、これはトークンから対応するテンソルへのマッピングを完了したことを意味します。

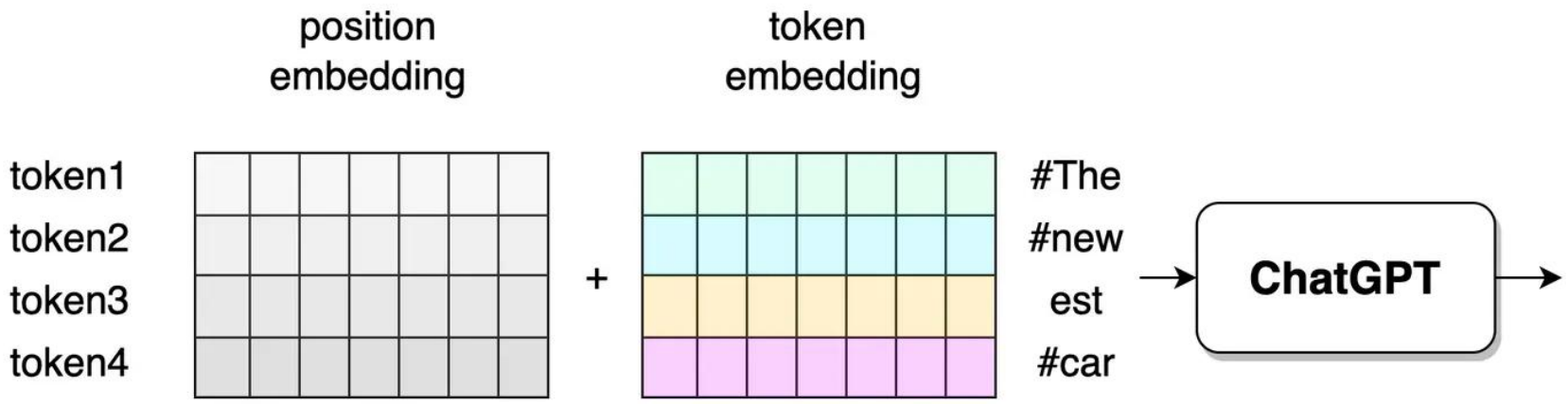
実際のモデルでは、**一度にモデルに入力できるトークンの数は無限ではありません**。たとえば、ChatGPT の gpt-3.5-turbo バージョンでは、最大入力トークン数は 4097 トークンであり、この範囲を超えるとモデルによって自動的に切り捨てられます。



第二章　大規模言語生成モデルとTransformer基盤

Ø　Position Embedding

自然言語では、文字の順序が非常に重要です。したがって、ChatGPT はモデルの各トークンの順序が変わらないように、各トークンの位置テンソル（Position Embedding）を入力側で明示的に識別する必要があります。この位置テンソルのサイズはトークンの埋め込み（embedding）と一致しています。両者は入力時に統合される必要があります。

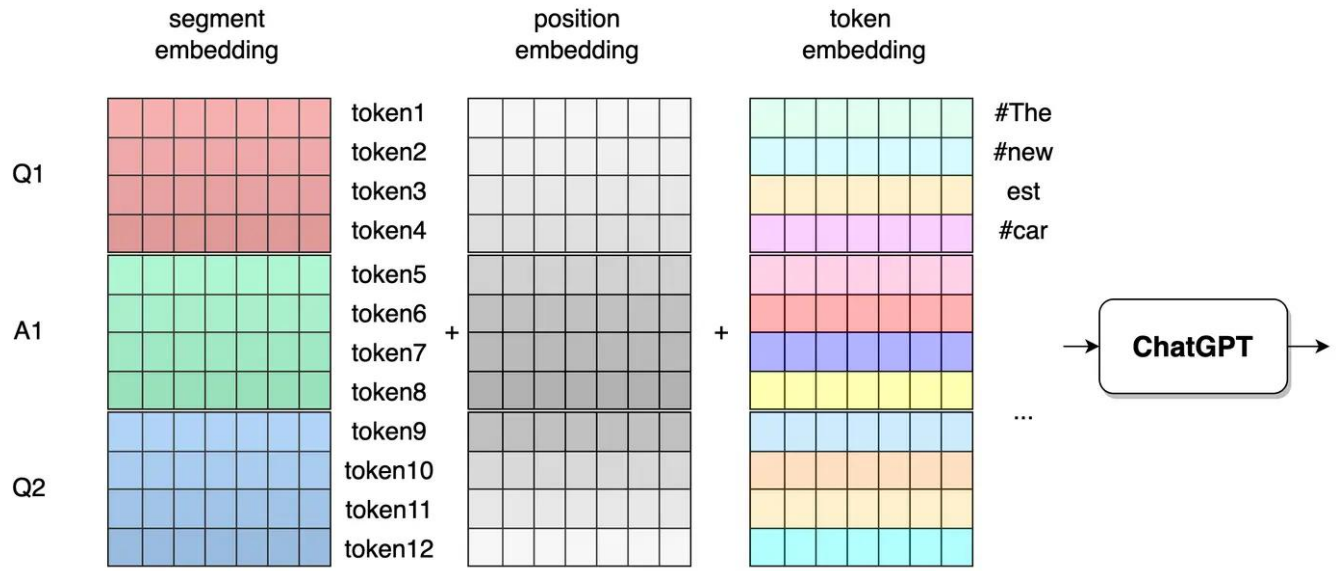


第二章　大規模言語生成モデルとTransformer基盤

Ø　Segment Embedding

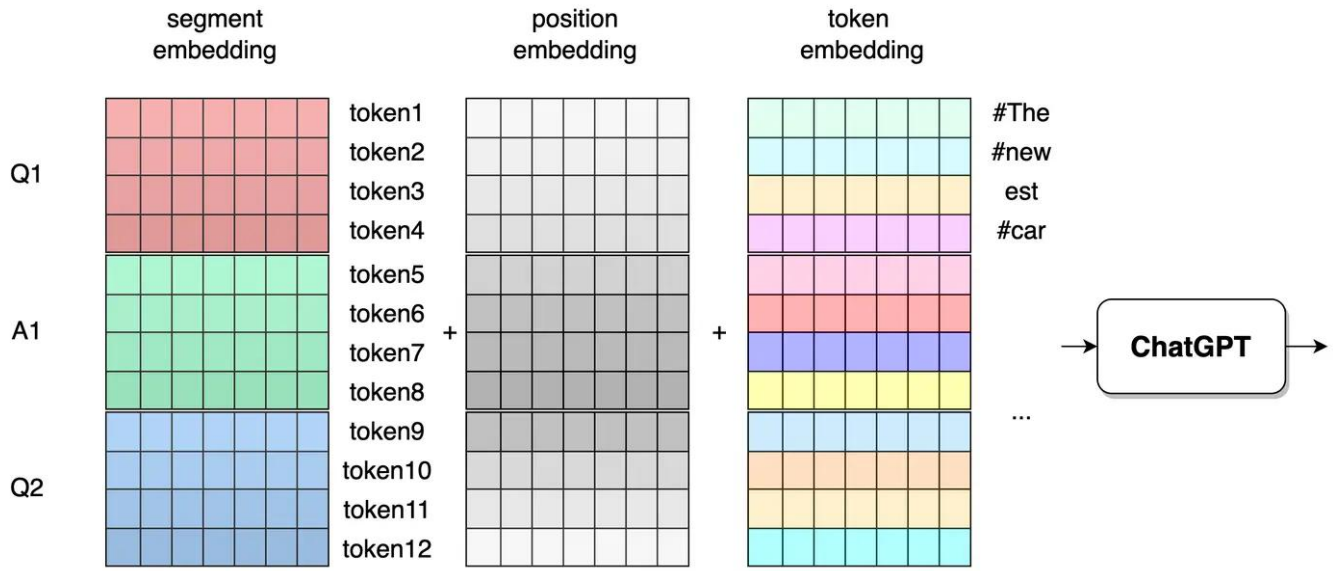
第一章で述べたように、大規模言語モデルは多輪対話能力を持っています。

モデル内では、入力1（Q1）、出力1（A1）、入力2（Q2）などの部分情報を区別する必要があります。これらの部分情報はそれぞれ「segment」と呼ばれ、各セグメントは複数のトークンを含み、同じ「segment embedding」を共有します。具体的な方法は以下の通りです：



第二章　大規模言語生成モデルとTransformer基盤

図では仮定として、Q1、A1、Q2 がそれぞれ 4 つのトークンを含んでいるとしています。もちろん、実際の入力では各セグメントに含まれるトークン数は柔軟に変化します。上記の対話は 2 回のラウンドしかありませんが、実際の入力では対話のラウンド数は非常に多くなる可能性があります。形としては Q1、A1、Q2、A2、...、Qn のようになり、すべてのセグメントに対応するトークンの総数がモデルが許可する最大トークン数を超えない限り問題ありません。



第二章　大規模言語生成モデルとTransformer基盤

Ø　Embeddingの利点

初期の NLP では、テキスト文字列を直接処理しており、Embeddingという操作は存在していませんでした。埋め込み操作は、最初に word2vec モデルによって提案され、実施されました。GPT シリーズのモデル、ChatGPTを含むモデルは、この操作を標準のデフォルトステップとして採用しています。

第一章で述べたように、高いレベルのAI知能を実現するには、モデルの規模が大きく、多くのパラメータを必要とします。機械学習の分野では、神経ネットワークモデルが最も容易にモデル規模を拡張できます。

Embedding操作がなければ、NLPの分野は依然として文字を直接処理するレベルにとどまり、モデルの規模拡張が難しくなります。Embeddingにより、文字に対応するトークンが抽象的な固定次元のテンソルに変換され、NLP は深層神経ネットワークの領域に足を踏み入れたことになります。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Attentionメカニズム

OpenAI の GPT シリーズモデルをはじめ、他のテクノロジー企業が開発した最先端の NLP モデル、さらには画像処理モデルにおいても、Attention（注意）メカニズムが広く採用されています。これは現在の NLP 神経ネットワークにおける重要なメカニズムといえます。

注意メカニズムの考え方は、実際にはさまざまな分野で広く応用されています。これを以下のように抽象化できます：あるインテリジェントエージェント（人間やAIモデル）が受け取った大量の情報（テキスト、画像、音声）から、重要でない、または関連性のない情報を排除し、自分に密接に関連する情報に重点的に注目します。その核心は、注目する情報の範囲を絞り込み、情報の圧縮を実現することにあります。

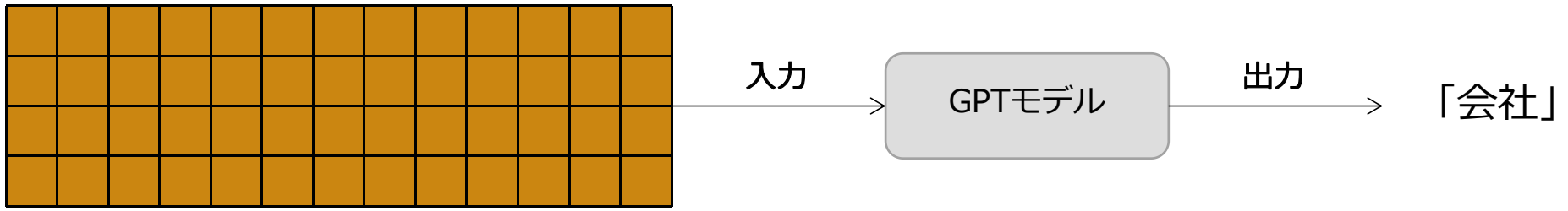
Chapter 1の紹介に基づいて、NLP における ChatGPT 言語モデルのモデリングは、実際には入力テキストの文脈的な関連性を探すことです。例えば： *日本の3大通信会社はdocomo、auと（　　）です。*

このテキストで最終的な文を補完するには、前文の情報を参考にする必要があります。モデルが空白文字を出力する際に、「通信会社」という言葉に最大限注意を払うようにする注意メカニズムを設計すべきです。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Attentionメカニズムの本質は重み付けです。

本章の説明に基づいて、NLP モデルでは自然言語がトークン形式でモデルに入力されます。以下の図のように、緑色の各列はそれぞれのトークンに対応する埋め込み（embedding）ベクトルを表しており、各トークンの埋め込みが４次元であると仮定します。7個のトークンが共に埋め込み行列を構成します。モデル設計のアイデアとしては、モデルが「会社」の「会」という文字を出力する際に、「Trip7.school」という文字により注意を払うようにすることです。



Trip7.schoolは、AI教育に特化した（　　）です。

1　2　3　4　5　6　7

第二章　大規模言語生成モデルとTransformer基盤

Ø　**Attentionメカニズムの本質は重み付けです。**

モデルが「Trip7.school」の文字により注意を払うようにすることは、実際には「Trip7.school」に対応するトークン埋め込み（token embedding）により大きな重みを与えることと考えられます。

神経ネットワークモデルでは、すべての操作は行列操作であり、すべての特徴はベクトル形式で表されます。ここで、 e_iは第 i 番目のトークンの埋め込み表現e_iを示し、この例ではそれが4次元ベクトルです。 w_iは第 i 番目のトークンに対応する重み値で、これはスカラー値です。したがって、すべてのトークン埋め込みに対して重み付けを行うことができます：

$$h = \sum_i e_i w_i$$

ここで、 h は加重後の結果であり、これは4次元ベクトルです。これは、各トークンの異なる埋め込み e_i を、重要度（重み値w_i）に応じて加算したものを意味します。重み値が高い e_i は後続の操作に大きな影響を与え、重み値が低い e_i は後続の操作に小さな影響を与えます。これにより、重みが高い e_i により注意を払う効果が生まれます。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Softmax 関数

上記の加重計算では、w_iはスカラー値です。モデルが訓練された後、上記の例に対してトークンの計算が行われ、次のように得られると仮定します：

$$w = (w_1, w_2, \ldots, w_7) = (1.4, 1.2, \ldots, -0.7)$$

数値には正のものと負のものがありますが、前の2つのトークンに対応する重みのスカラー値が大きいことは、後続の操作に対する影響が大きいことを示しています。もし直接加重を行うと、これは一般的な認識に合いません。一般的には、重みの割合は確率形式で表されるべきで、確率値は 0 より大きく、1 より小さいべきであり、すべての要素の合計が 1 になるべきです。

したがって、重み w を確率の割合形式に変換する必要があります。これには主に Softmax 関数を使用します。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Softmax 関数

具体的には、AttentionメカニズムでSoftmax関数は次のように定義されます。

$$\propto_i = \frac{exp(w_i)}{\sum_i exp(w_i)}$$

Softmax 関数の目的は、ベクトル内のすべての要素を正規化して、確率の形式で出力することです。すなわち、各要素の値を 0 から 1 の範囲に変換し、すべての要素の合計が 1 になるようにします。

Softmax 関数は神経ネットワークモデルで非常に一般的に使用されており、注意メカニズムの計算だけでなく、Softmaxは分類タスクの交差エントロピー損失関数とも完璧に適合します。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Self-Attention機構

前述のように、重みベクトル w を使用することで、モデルが注目すべき重要な内容を見つけることができます。それでは、 w の値はどのように計算されるのでしょうか？

重みを計算する方法は、モデルやNLPタスクによって異なります。この技術は長年の進化を経て、最終的にはSelf-Attention機構に収束しました。これが ChatGPT が採用している形式です。

w は重みベクトルで、その長さ（次元数）はトークンの数と同じであり、各項はスカラー値です。神経ネットワークモデルでは、計算の基本としてベクトルや行列などのテンソルが使用されています。そのため、スカラー値を計算する最も簡単な形式はベクトルの内積です。したがって、2つのベクトルを見つける方法を考える必要があります。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Self-Attention の Q, K, V

第 i 番目のトークンに対して、2つのベクトル q_i と k_i があると仮定します。これらは同じ次元を持ち、内積を計算することでスカラー値が得られます。

前述の例に基づいて、文を補完する際に「Trip7.school」が空白の場所に挿入する文字に最も大きな影響を与えることがわかります。「Trip7.school」が何を挿入すべきかという認識も、この文自体から得られます。言い換えれば、 w_i の重みの情報源は、依然として元の文（Self）であり、これがSelf-Attentionという名前の理由です。したがって、各トークンの embedding をそれぞれベクトル q_i および k_iとして直接計算に使用できます。

なぜ q と k という名前がついたのでしょうか?

「q」(query)と「k」(key)という名前は、元々コンピュータ検索エンジンで提唱された概念であり、AI の分野では注意メカニズムに応用されています。例えば、Google 検索エンジンで「大阪には何がおいしいの？」と検索すると、検索エンジンはマッチングの程度に応じていくつかの回答を提供します。

この場合、ユーザーの検索クエリ（質問）が「query」と呼ばれ、各マッチング結果はそれぞれ「query」との関連性を持つため、「key」となります。

第二章　大規模言語生成モデルとTransformer基盤

Ø　Self-Attention の Q, K, V

「Trip7」トークンの出力ベクトルを計算する際には、そのqueryベクトルを用いて、各トークンのkeyベクトルと内積を計算します。そして、この内積スコアを正規化します（つまり、queryベクトルの次元数$\sqrt{d_q}$で割ります）。

上述の計算方法は、一歩一歩ベクトル表示形式で展開されますが、行列形式での計算式は次のように表すことができます：

$$Attention(Q, K, V) = softmax(\frac{Q \bullet K^T}{\sqrt{d_q}})V$$

ここで、$Q \bullet K^T$ は内積の行列形式を表し、最右側の V はトークン embeddingで構成される行列として理解できます。

第二章　大規模言語生成モデルとTransformer基盤

Ø　まとめ

注意力メカニズムの本質は、大量の情報からノイズや無関係な情報を取り除き、興味のある情報だけを保持することです。

自然言語処理分野での注意力メカニズムの主な応用は、Self-Attentionの形式であり、これは神経ネットワークが十分な適合能力を持つための重要な要素です。

第一章で、Transformer が ChatGPT の「家」のレンガや鉄筋を構成していると述べましたが、Self-Attentionメカニズムは Transformer の核心要素です。次の章では、Transformer の構造と原理について紹介します。