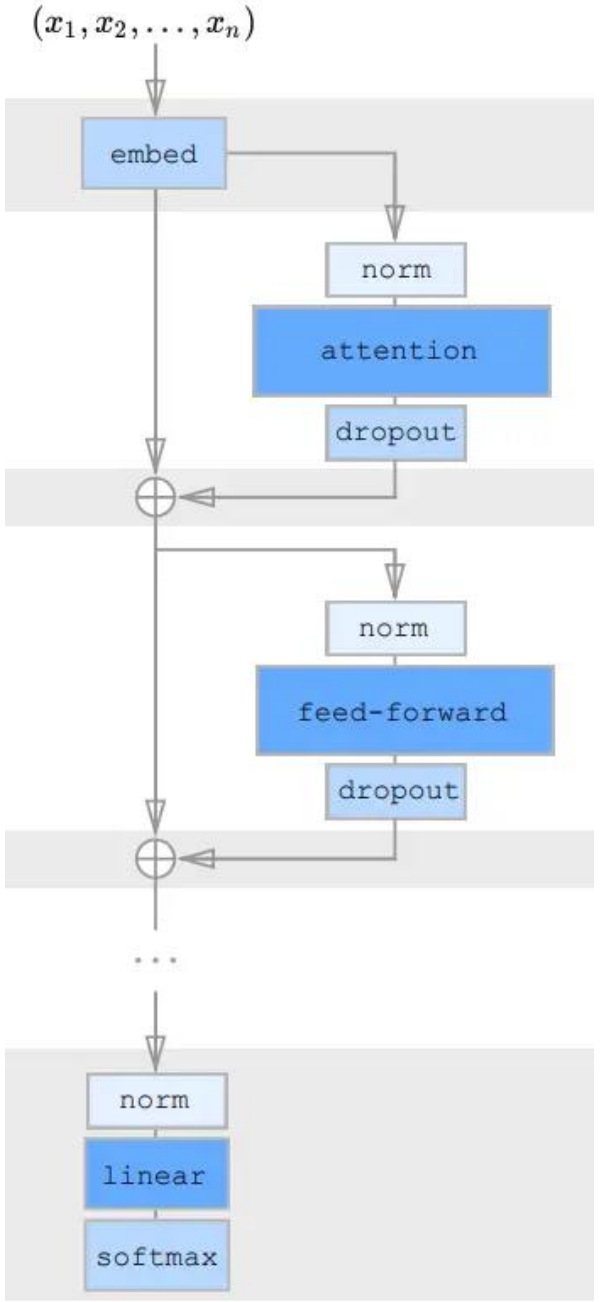
第三章　Transformerの構築とトレーニング

前章では、現在最も人気のあるニューラルネットワークモデルの中核メカニズムであるSelf Attention機構の動作原理を紹介しました。そして、そのSelf Attentionを具体化したものがTransformerモデルの構成要素です。今章では、Transformerについて重点的に紹介します。

Ø　Transformerの構築

$(x_1, x_2, \ldots, x_n)$は前章で説明した入力のトークンembeddingであり、これは $l_{tokens} * l_{embeddings}$ サイズの行列です。ここで、l_{tokens} は入力テキストのトークン数を示し、GPT-3ではこの長さは4097です。これは、GPT-3モデルに入力できる最大トークン数が4097であることを意味します。また、$l_{embeddings}$ は各トークンのembeddingの次元を示し、GPT-3ではこの長さが12280です。

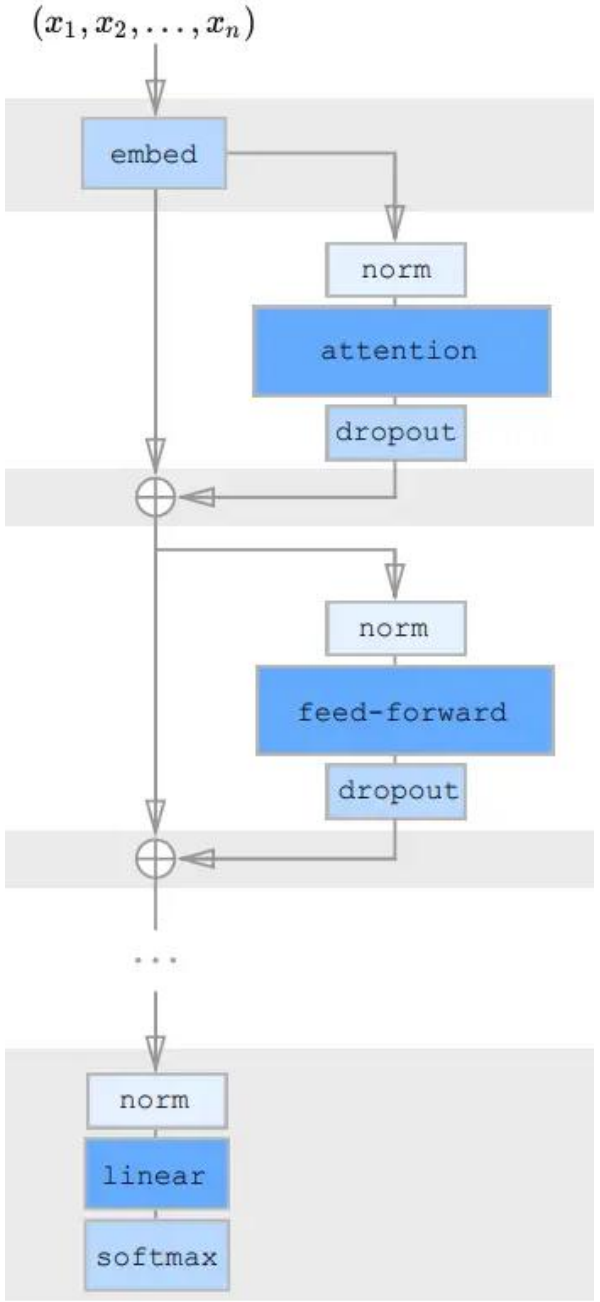


第三章　Transformerの構築とトレーニング

Ø　Transformerの構築

図中の濃い青色のattentionモジュールが、前章で紹介した Self-Attention であり、これは Transformer モデルの中核的な操作です。さらに、拡張されたパラメータモジュールである Feed-Forward 層も含まれています。そして、各層の前後には、薄い青色のnorm層とdropout層が配置されています。モデルの最後には、出力結果を生成するlinear層と、交差エントロピーの損失関数を計算するためのsoftmax層も含まれています。

GPT-3モデルおよびそれ以降のさまざまなモデルでは、Self-Attentionに前章で紹介したような従来のSelf-Attention機構を使用しているわけではなく、Sparse Transformerとも呼ばれる、疎な注意機構が採用されています。ここでは、その詳細には触れません。

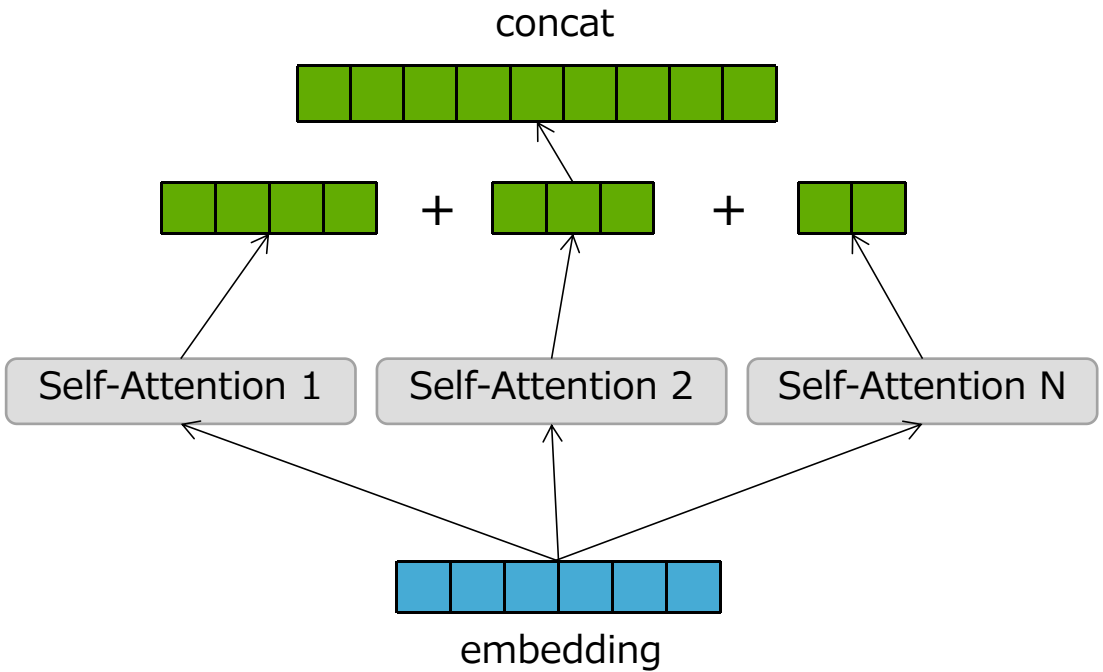


第三章　Transformerの構築とトレーニング

Ø　Multi-head Attention

前述では、1回の注意機構を作成するプロセス全体を説明しました。いわゆるMulti-head Attentionとは、モデル内で複数回注意機構を実行することで、モデルが異なる情報に注意を払えるようにすることを目的としています。

図に示すように、全体のプロセスは非常にシンプルです。N回の注意機構を設計し、そのN回の結果を結合することで、出力が完成します。いわゆる結合方式とは、図中で異なる色合いの緑色のブロックで示されているように、例えば2つの注意頭の計算結果がそれぞれ $a = [a_0, a_1, \ldots, a_m]$、$b = [b_0, b_1, \ldots, b_m]$であると仮定すると、結合後の結果は $concat = [a_0, a_1, \ldots, a_m, b_0, b_1, \ldots, b_m]$となります。

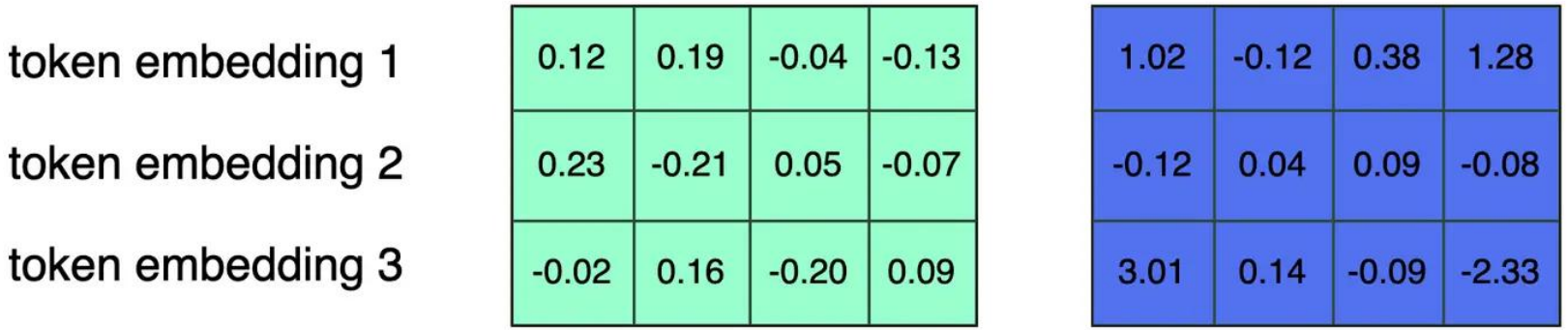


第三章　Transformerの構築とトレーニング

Ø　Normalization

前述のTransformer構造図に示されているように、各Attentionモジュールが接続される前には、必ずnormモジュールがあります。これはニューラルネットワークモデルでよく見られるNormalization（正規化）モジュールです。

仮に、疎なSelf-Attentionモジュールに入力される前に、2組の具体的なトークンembedding値があり、それぞれの次元は4 × 3とします。この場合、各行は1つのトークンのembedding値を表し、具体的な値の状況は図のようになります。



第三章　Transformerの構築とトレーニング

Ø　Normalization

上図の表を観察すると、左側の表の行列では、すべての要素の値が (−0.25, 0.25) の範囲内にあり、値が集中しており、分布が均一で安定しています。一方、右側の表の行列では、すべての要素の値が (−3.05, 3.05) の範囲内にあり、値の変動が大きいことがわかります。特に、右側表の第2行におけるトークンembeddingの分布は (−0.2, 0.2) の範囲内にあり、値の変動が比較的小さいものの、全体的には分布が不安定で差異が大きいです。

神経ネットワークモデルのトレーニングにおいて、データの分布が集中して均一である場合、モデルのトレーニングが迅速に進行し、収束しやすくなります。これはモデルトレーニングの目的を達成するために重要です。そのため、Self-Attention層の前にnorm層を追加することで、分布がばらついているデータを、より集中した分布に整えることができます。

第三章　Transformerの構築とトレーニング

Ø　Normalization

上図の表を観察すると、左側の表の行列では、すべての要素の値が (−0.25, 0.25) の範囲内にあり、値が集中しており、分布が均一で安定しています。一方、右側の表の行列では、すべての要素の値が (−3.05, 3.05) の範囲内にあり、値の変動が大きいことがわかります。特に、右側表の第2行におけるトークンembeddingの分布は (−0.2, 0.2) の範囲内にあり、値の変動が比較的小さいものの、全体的には分布が不安定で差異が大きいです。

神経ネットワークモデルのトレーニングにおいて、データの分布が集中して均一である場合、モデルのトレーニングが迅速に進行し、収束しやすくなります。これはモデルトレーニングの目的を達成するために重要です。そのため、Self-Attention層の前にnorm層を追加することで、分布がばらついているデータを、より集中した分布に整えることができます。

第三章　Transformerの構築とトレーニング

Ø　Normalization

具体的な方法として、まず処理対象のデータを$x = (x_1, x_2, \ldots, x_n)$とします。最初に、このデータの平均値と分散を計算します。計算式は次のとおりです。

$$\mu = \frac{1}{n} \sum x_i$$

$$\delta^2 = \frac{1}{n} \sum (x_i - \mu)^2$$

その後、各値についてNormalizationを行います。このNormalizationにより、すべてのembedding値は安定した分布に収束し、モデルのトレーニングが容易になります。

$$\widehat{x}_i = \frac{(x_i - \mu)}{\sigma}$$

この部分の知識は中学程度の数学に該当するため、例を挙げて紹介する必要はありません。実際、NormalizationにはBatch NormalizationやLayer Normalizationなどの種類があります。NLP分野では、一般的にLayer Normalizationが使用されます。Layer Normalizationとは、各入力データのすべての数値要素に対してnorm操作を適用することを指します。

第三章　Transformerの構築とトレーニング

Ø　Dropout

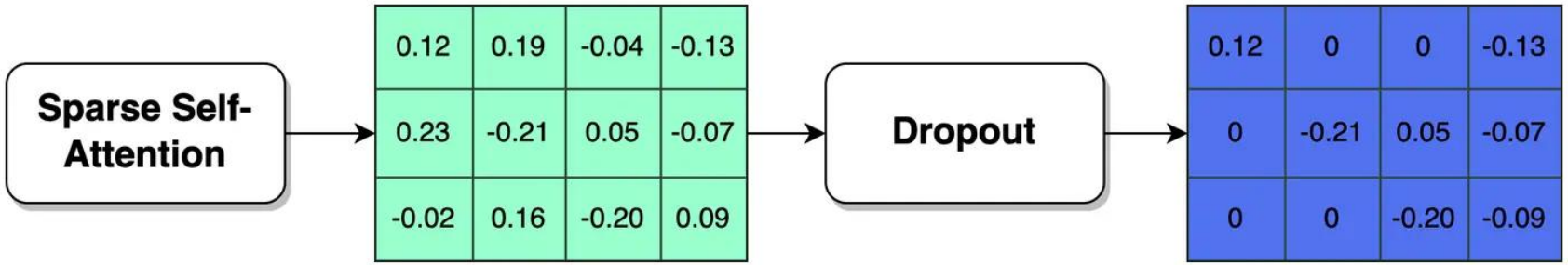
Transformer構造図に示されているように、各Attentionモジュールの前には、dropoutモジュールがあります。このモジュールの主な機能は、トレーニング過程での過学習（Overfitting）を防ぐことです。

過学習とは、モデルがトレーニングデータでは良好な性能を示す一方で、テストデータでは性能が低下する現象です。簡単に言うと、過学習はモデルが過度に複雑で、トレーニングデータには非常に良く適応するものの、新しいデータに対する汎化能力が不足している状態です。過学習は、モデルがトレーニングデータの細部やノイズに過剰に適応してしまうこと、またはトレーニングデータが少なすぎることが原因で発生します。モデルが過度に複雑な場合、トレーニングデータの細部やノイズまで学習しようとし、その結果、新しいデータに対してはパフォーマンスが低下します。トレーニングデータが少ない場合、モデルは十分な特徴を学習できず、過学習が発生します。

第三章　Transformerの構築とトレーニング

Ø　Dropout

そのため、モデルの過学習を直感的に解決する方法は、モデルを簡素化することです。Transformerにおいては、特定の計算結果に対してランダムにゼロを設定する操作が行われます。具体的には、例えばAttention層がトークン行列のセットを出力するとします。モデルが完全にランダムなdropoutを行うと、その結果の一部の要素がゼロに置き換えられます。

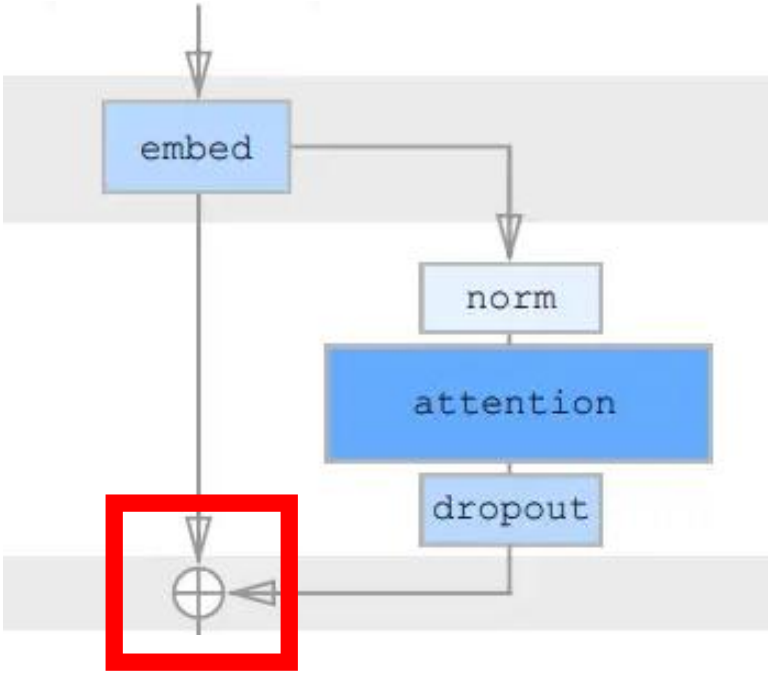


このようにして、モデル内のいくつかの要素特徴が意図的に隠されることにより、モデルが簡素化されます。モデル内でのdropoutはランダムに実行されますが、どれだけの要素をゼロにするかは事前に設定した確率値、すなわちdropout ratioによって決まります。この値が0の場合は、すべての要素がゼロにされないことを意味します。上図からおおよそのdropout率を計算することができます：$drop\_ratio = 5/12 = 0.417$。

第三章　Transformerの構築とトレーニング

Ø　ResNet

ResNet（Residual Network）の残差モジュールについて、Transformer構造図では、Attentionモジュールの入力である embedding と出力結果が加算されています。この加算操作は、残差モジュール（Residual Module）と呼ばれます。このようにする理由は、モデルのトレーニング中に勾配が消失したり爆発したりしないようにするためです。残差モジュールの本質的な目的は、モデルがスムーズにトレーニングを進め、目標を達成することです。



第三章　Transformerの構築とトレーニング

Ø　Encoder-Decoder構造

第一章では、言語モデルのEncoder-Decoder構造について大まかに紹介しました。以下の図に示されている通りです。

実際、Encoder-Decoder構造は、最初に機械翻訳の問題を解決するために使用されました。興味のあるひとは、クラシックな論文『Neural Machine Translation by Jointly Learning to Align and Translate』（2014年）を読むと良いでしょう。機械翻訳モデルは、英語の文を受け取り、モデルの処理を経て、対応する翻訳結果を出力します。この建模手法は最初に「seq2seq」と呼ばれ、これは「sequence-to-sequence」の略であり、文字列のシーケンスを入力として、対応する文字列のシーケンスを出力するという意味です。その後、NLP（自然言語処理）分野では、seq2seqはさまざまなテキストシーケンスタスクに応用されるようになり、NLP分野の標準的なモデリング手法となりました。

第三章　Transformerの構築とトレーニング

Ø　Encoder-Decoder構造

Encoder-Decoderというモデリングアプローチでは、Encoderは文字情報を受け取り、考えるプロセスを模しており、Decoderは脳内の情報を言語表現に変換するプロセスを模しています。言い換えれば、Encoder-Decoderは機械が人間の思考を模倣する方法の一つです。

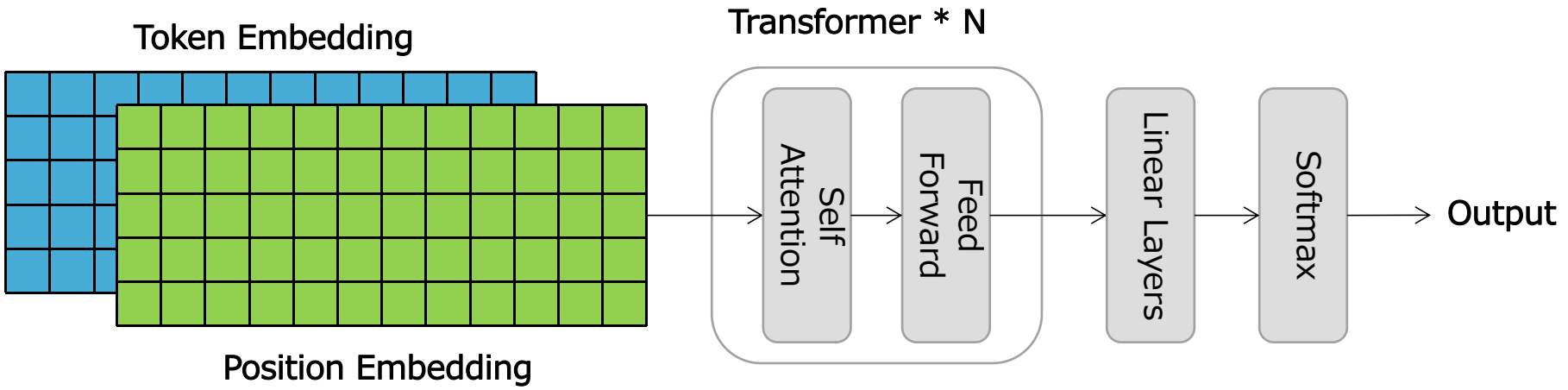
また、Encoder-Decoderは特定のモデル構造を指すものではなく、広範なモデル設計の考え方です。この編解码アーキテクチャはNLP分野に限定されず、画像処理や音声処理など、他の領域にも適用できます。例えば、画像分野の対抗生成モデル（GAN）などもその一例です。

GPTはDecoderのみのアーキテクチャを採用しており、Encoder部分は省略されています。その具体的な構造はTransformerに基づいています。しかし、Encoder-Decoderアーキテクチャの内部設計は、RNN（リカレントニューラルネットワーク）などのモデル構造を使用することも可能ですし、将来的にはより良い設計に置き換えられる可能性もあります。

第三章　Transformerの構築とトレーニング

Ø　GPTのEncoder-Decoder

次に、TransformerがどのようにGPTのEncoder-Decoderアーキテクチャに組み込まれているかを図示してみましょう。以下の図では、Transformerモデル構造からnorm正規化、残差計算、およびdropoutモジュールが省略されています。

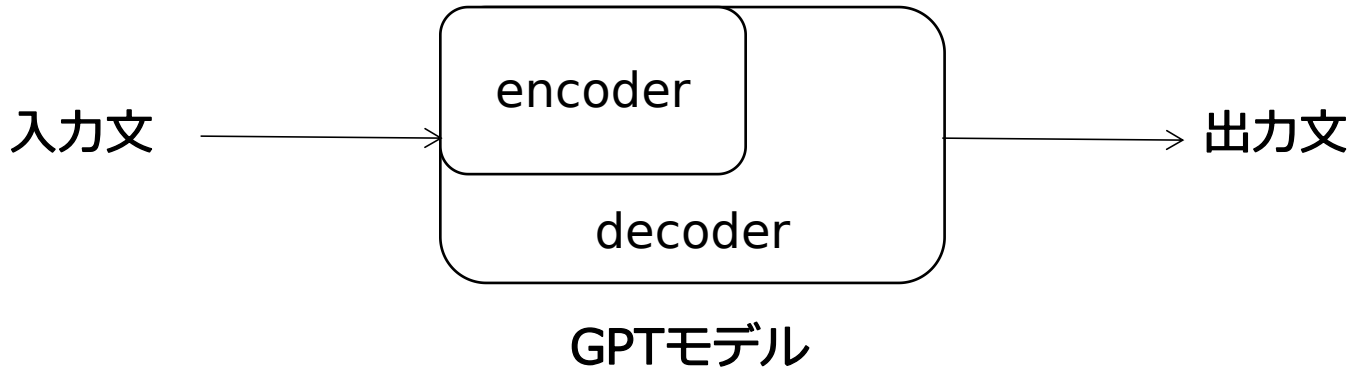


ユーザーが文を入力すると、モデルはまず前述のembeddingを使用してテキストをトークンに変換します。次に、対応するトークンembeddingと位置embeddingを見つけ、それらを加算してからTransformer構造に入力して注意計算を行います。Transformer構造自体は複数の層を持つことができます。各層の入力テンソルと出力テンソルのサイズはすべて同じであり、前の層のTransformer出力が次の層の入力として使用されます。これを積み重ねるようにして、最終層に到達します。

第三章　Transformerの構築とトレーニング

Ø　GPT自体はDecoderです

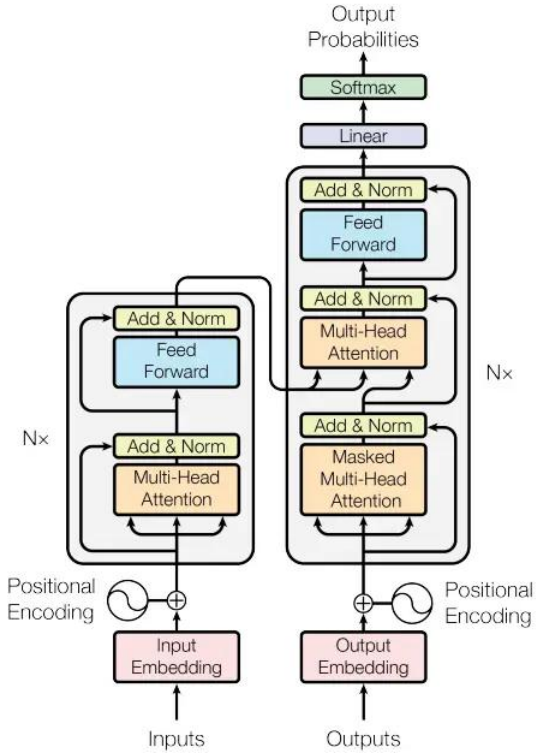
上記の計算プロセス全体を通して、モデルはまず一連の入力データを受け取り、それをTransformerを通して処理します。これはencoder部分に相当します。その後、モデルは予測されたトークンを出力します。これはdecoder部分に相当します。このプロセスは、前述のencoder-decoderアーキテクチャのプロセス図とは少し異なります。主な違いは、前述のアーキテクチャではencoderとdecoderが互いに独立しており、間に別の信号が接続されていますが、GPTモデルではencoderとdecoderが一体化されています。



第三章　Transformerの構築とトレーニング

Ø　GPT自体はDecoderです

下の図は、Transformer論文【2017 - Attention Is All You Need】で示されている最初のencoder-decoderモデルのアーキテクチャです。左半分がencoder、右半分がdecoderです。前述のGPTモデルのアーキテクチャは、実際にはこの元の構造の右半分に相当します。したがって、GPTシリーズモデルはdecoder部分のみを使用していると言われることが多いです。

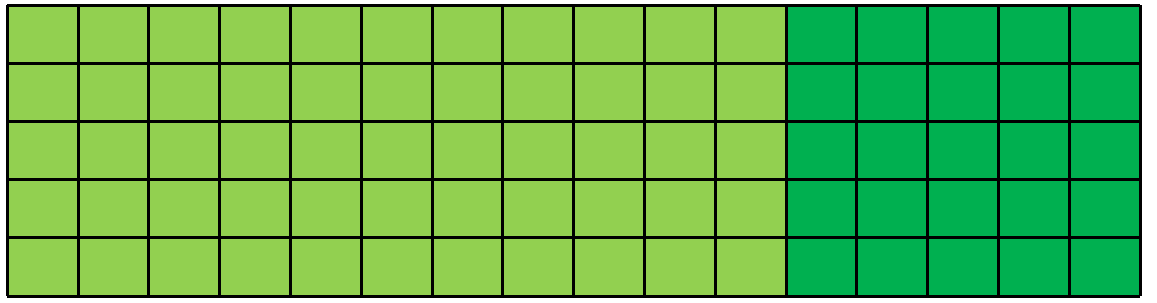


https://doi.org/10.48550/arXiv.1706.03762

第三章　Transformerの構築とトレーニング

Ø　Mask Layer

実際のモデル計算プロセスでは、みんなさんはChatGPTを使用したことがあるかもしれません。モデルへの入力と出力の文は長さが異なり、長いものでは数千文字、短いものでは10文字にも満たないことがあります。このように、入力データの長さは常に変動します。一方、ChatGPTモデルの構造は比較的固定されており、モデルが受け入れる最大のトークンシーケンスの長さは事前に設定されています。たとえば、モデルが受け入れる最大トークン数が15であり、ユーザーが入力したデータが15未満のトークン数である場合、モデルはこのデータに対して一定の処理を行います。



第三章　Transformerの構築とトレーニング

Ø　Mask Layer

前ページの図に示されたように、モデルの入力トークン埋め込みの全体構造では、モデルの入力最大トークン数が15と規定されています。ユーザーの入力が15トークン未満の場合、モデルは自動的に入力を補完し、特殊な<pad>トークン（図中では/pで示され、<eos>トークンと同様の性質を持ち、具体的な文字を示すのではなく、機能的な役割を果たす）を追加します。このトークンは実際の意味を持たず、単にプレースホルダーとして機能します。

モデルが後続のTransformer操作を行う際には、<pad>トークン自体が意味を持たないため、これらのトークンは文脈の関連性に基づく自己注意計算に参加すべきではありません。この問題を解決するために、Maskレイヤーが導入されました。

第三章　Transformerの構築とトレーニング

Ø　まとめ

Transformerの核心構造はSelf-Attentionであり、これらのコンポーネントのスタッキングがGPTの言語モデルを構成しています。

Transformerでは、Normalization、残差計算などが用いられ、モデルの適合能力と適応性が強化されています。

ChatGPTモデルは、encoder-decoderモデルアーキテクチャに基づいて構築されています。

ChatGPTは主にMask（マスク）方式を採用して、注意力計算に参加しないトークンの位置を隠します。