Attention 機構と Transformer

長谷川駿一

2023 年 5 月 26 日

1　はじめに

Transformer とは，2017 年に Google から発表した論文「Attention Is All You Need」[1] で提案され，自然言語処理の分野において大きな貢献をもたらした DNN ベースのモデルである．自然言語処理分野で有名なモデルである BERT や GPT などにも使用されており，今日では，音声分野や画像による物体検出など，さまざまな分野で応用されている．

本稿では，Transformer の構造から，なぜ Transformer がここまで躍進したのかを説明し，実装してみた結果を発表する．

2　Transformer 以前の提案手法

2.1　Seq2Seq

Transformer を含む自然言語処理モデルの多くは **Encoder-Decoder** モデルで構成される．

Encoder-Decoder モデルの設計思想は，前半の Encoder で入力より小さい特徴空間の固定長ベクトルに圧縮し，後半の Decorder でその固定長ベクトルを入力として受けて復元させるということをする．Encorder-Decorder モデルの代表的なモデルとして **Seq2Seq** がある (図 1)．Seq2Seq では入力された系列情報を別の系列情報に変換するモデルであり，機械翻訳などで活用されている．

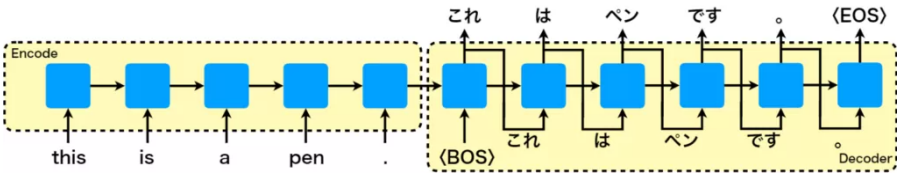

図 1　Seq2Seq の構造

しかし，Seq2Seq のデメリットは，入力系列のサイズに関わらず，固定長の特徴ベクトルに符号化されることである．これは，エンコーダの RNN の隠れ層で伝搬されてきた特徴ベクトルが，入力の長さに関係なく一定サイズでデコーダに渡されるため，入力系列が長い場合に前半の入力情報を覚えきれない．

2.2 Seq2Seq with Attention

そこで次に提案されたのが，Seq2Seq に Attention 機構を取り入れた **Seq2Seq with Attention** である．エンコーダの最後の潜在状態のみをデコーダで受け渡していた Seq2Seq に対して，Seq2Seq with Attention では，次の単語を予測するのに適する入力単語群へのアテンションを用いた**コンテキストベクトル**を導入しており，デコーダに対する入力を動的に変化させることができる．

Seq2Seq with Attention は，図 2 のような構造をとっており，大きな手順は以下の通りである．

---

①. モデル内のデータに対して，Attention 機構を用いた処理を行う．

通常の Seq2Seq モデルにおいて（エンコーダ RNN の潜在ベクトル (隠れ層の系列) を $\{\mathbf{s}_1, \mathbf{s}_2 \cdots, \mathbf{s}_j, \cdots \mathbf{s}_N\}$, デコーダ RNN の潜在ベクトルを $\{\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_i, \cdots \mathbf{h}_M\}$ とする），現ステップ i の潜在ベクトル $\mathbf{h}_i$ から，$\mathbf{h}_{i+1}$ を予測したい．

このときのモデル内のデータの処理として，$\mathbf{h}_i$ に対する，入力文書中の各単語の隠れ変数 $\mathbf{s}_j (j \in \{1, 2, \cdots, N\})$ の**アテンション係数** $a_i(j)$ を，**スコア関数**を用いて予測する．アテンション係数とは，以下の式で計算される．

$$a_i(j) = \frac{\exp(\mathrm{score}(\mathbf{s}_\mathrm{j}, \mathbf{h}_\mathrm{i}))}{\sum_{j'=1}^{N} \exp(\mathrm{score}(\mathbf{s}'_\mathrm{j}, \mathbf{h}_\mathrm{i}))} \tag{2.1}$$

スコア関数 score(·) は，$\mathbf{s}_j$，$\mathbf{h}_i$ 二つのベクトル間の関連度を示す単純な関数である．例として，$\mathbf{s}_j \cdot \mathbf{h}_i$(二つのベクトルの内積) や，$\mathbf{s}_j \cdot \mathbf{W}_a \mathbf{h}_i$ などがある．これらは設計時に自由に決める．

②. コンテキストベクトル $\mathbf{c}_i$ を計算する．

コンテキストベクトルは以下の式で表される．

$$\begin{aligned} \mathbf{c}_i &= \sum_{j=1}^{N} a_i(j)\mathbf{s}_j \\ &= a_i(1)\mathbf{s}_1 + a_i(2)\mathbf{s}_2 + \cdots + a_i(N)\mathbf{s}_N \end{aligned} \tag{2.2}$$

コンテキストベクトルは，翻訳後の単語が翻訳前の単語それぞれに対してどのくらい関連性があるのかの総和をとっている．

③. コンテキストベクトル $\mathbf{c}_i$ を入力に加え，次のフレーム $i+1$ を，デコーダで予測する．その後に潜在ベクトル $\mathbf{h}_{i+1}$ から単語 $\mathbf{y}_{i+1}$ を予測する．

④. 上記の手順を，<EOS> が次の単語として予測されるまで繰り返す．

この Seq2Seq with Attention の登場により，機械翻訳の精度がより高まったが，RNN による逐次的なデータ処理によってデータの並列化が出来ないままであり，高速な処理を行うことが出来なかった．なので，並列計算を不可能にしていた RNN を取り除き，並列可能としたモデルとして，次節に示す Transformer が提案された．

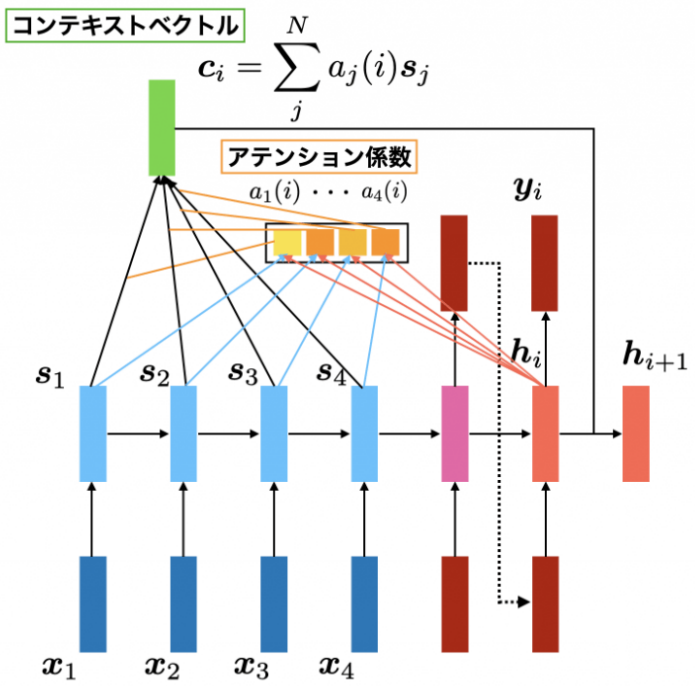

図 2　Seq2Seq with attention の構造

3 Transformer

Transformer は図 3 のような構造をとっている．

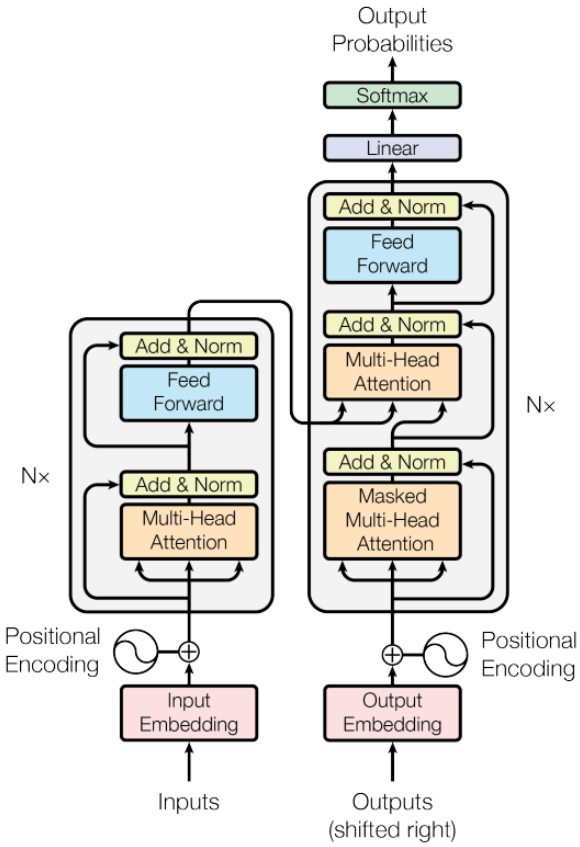

図 3　Transformer の構造

図からもわかる通り，Transformer では RNN や CNN を一切使用せずに Attention 機構のみで実装されている．これにより逐次的な処理を必要とせず，並列計算が可能となったため，GPU による高速演算が可能となった．また，より長文の依存関係を掴むことができるようになった．

次節以降では，Transformer の各層について説明していく．

3.1 Embedding

埋め込みとも呼ばれる．自然言語処理分野の埋め込みは，解析したい単語をその単語の意味を表す固定長ベクトルに変換することである．Transformer では，デフォルトで 512 次元のベクトルに変換する．

埋め込みの代表的な手法として，**Word2Vec** というニューラルネットワークで構成されたモデルがある．

3.2 Positinal Encoding

自然言語を取り扱う上で，語順の情報は必要不可欠である．従来の提案手法では RNN を用いていたため，語順の情報が保たれた状態で逐次的に処理を行っていた．これに対し RNN のない Transformer では，語順情報を入力に与える必要があったため，Positional Encoding で語順の情報をベクトル化し (デフォルトで 512 次元)，これを Embedding 化された固定長ベクトルの各要素に足し合わせている．

具体的な変換方法は以下の通りである．t をその単語が何番目に出現したのかを表す整数，変換後のベクトルを $\mathbf{p}_t$，$\mathbf{p}_t$ の要素を $p_t^{(i)}$，d は全要素数 (デフォルトで 512) とする．

$$p_t^{(i)} = \begin{cases} \sin(\sigma_k \cdot t), & (i = 2k) \\ \cos(\sigma_k \cdot t), & (i = 2k+1). \end{cases} \tag{3.1}$$

ただし，

$$\sigma_k = \frac{1}{10000^{2k/d}}. \tag{3.2}$$

式からもわかる通り，$p_t^{(i)}$ は t を時間，σ_k を角周波数とする sin 波，cos 波で表される．sin と cos が交互に要素として変換されるので，$\mathbf{p}_t$ は以下のように表される．

$$\mathbf{p}_t = [\sin(\sigma_1 \cdot t),\ \cos(\sigma_1 \cdot t),\ \sin(\sigma_2 \cdot t),\ \cos(\sigma_2 \cdot t), \cdots, \sin(\sigma_{d/2} \cdot t),\ \cos(\sigma_{d/2} \cdot t)]^T \tag{3.3}$$

3.3 Multi-Head Attention

Multi-Head Attention の構造を図 4(右) に示す．Multi-Head Attention は **Scaled Dot-Product Attention** を幾層か重ね，出力を連結 (Concat) したものであることがわかる．この Scaled Dot-Product Attention は 4(左) のようなモデルであり，$\mathbf{Q}$(Query), $\mathbf{K}$(Key),$\mathbf{V}$(Value) の 3 つの入力を持ち，1 つの値 (ベクトル) を算出する．

※ちなみに，これ以降に登場するベクトルは，元論文に合わせて列ベクトルとする．

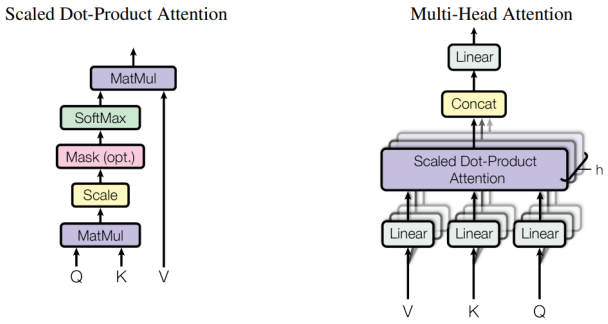

図 4　Scaled Dot-Product Attention と Multi-Head Attention の構造

Scaled Dot-Product Attention で用いられている Attention 機構は以下の通りである．ここで，系列長 (単語の数) を N とすると，$\mathbf{Q} \in \mathbb{R}^{N \times d_q}$，$\mathbf{K} \in \mathbb{R}^{N \times d_k}$，$\mathbf{V} \in \mathbb{R}^{N \times d_v}$ である．ここで，$d_q = d_k$ であり，元論文では強調されていないが，d_v も d_k と同じである．よって，入力の行列の列数は，すべて同じになる．

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{3.4}$$

まず，Query と Key の内積 (類似度) を計算し，$\sqrt{d_k}$ で割ることによって，次元数 d_k が大きい場合に内積が大きくなるのを防いでいる．それをソフトマックス関数に通し，最終的に Values との内積をとっている．したがって，最終的な出力は $\mathbb{R}^{N \times d_v}$ となる．

この Scaled Dot-Product Attention 機構を組み合わせた Multi-Head Attention は，以下の処理手順で計算される．

---

①. 入力の $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ を構成する列ベクトル $\mathbf{x}$ を，d_{model}/h の低次元ベクトルに射影する (デフォルトで $d_{modek} = 512$，$h = 8$)．射影後の次元は $d_q = d_k = d_v = d_{model}/h = 512/8 = 64$ となる．

この射影を行うために，i 番目 ($i \in \{1, 2, \cdots, h\}$) のヘッド個別の全結合層 $\mathbf{W}_i^Q$，$\mathbf{W}_i^K$，$\mathbf{W}_i^V$ との内積をとっている．

※ここで，入力 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ は，実はすべて同じ値である．これに対して固有の 3 種類の行列を掛けて射影することにより，抽出している特徴が異なっているが元の入力は同じの自己アテンション機構となっている．ここで，必ずしも入力は埋め込み層からではない．図 3 より，それぞれのエンコード，デコード機構はそれぞれ N 回繰り返している．よって，最後の Add&Norm 層からの入力であることもある．

②. 低次元に射影された $\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V$ を入力に，$h = 8$ 個の Scaled Dot-Product Attention を実行する．よって，出力は 8 個の $\mathbf{Z}_1, \mathbf{Z}_2, \cdots, \mathbf{Z}_h$ となる．

③. $\mathbf{Z}_i$ の各出力を 1 つのベクトルに結合 (Concatenation) する．よって，各列ベクトルの次元が $h \cdot d_v = 8 \cdot 64 = 512$ になる．

④. 結合したものを全結合層 $\mathbf{W}^O$ で変換する．$\mathbf{W}^O \in \mathbb{R}^{h \cdot d_v \times d_{model}}$ であるため，最終的な次元は d_{model} となる．

数式で表すと以下のとおりである．

$$Multi - HeadAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(\mathbf{Z}_1, \mathbf{Z}_2, \cdots, \mathbf{Z}_h)\mathbf{W}^o \tag{3.5}$$

$$\mathbf{Z}_i = Attention(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{3.6}$$

3.4 Feed Forward

Position-Wise Feed-Forward Networks とも呼ばれる．2 層の全結合ニューラルネットワークであり，1 層目の出力では ReLU に通し，2 層目では値をそのまま出力 (線形変換) している．この FF 層をはさむことにより，Attention 層で形成された位置情報込みの情報に対して，位置情報との依存関係を和らげ，一意なものとして変換することによって，表現力が増す (らしい)．

式を以下に示す．

$$\mathrm{FNN(x)} = \max(0, \mathrm{xW_1} + \mathrm{b_1})\mathrm{W_2} + \mathrm{b_2} \tag{3.7}$$

3.5 Masked Multi-Head Attention

デコーダ側でも同様に，出力系列 (翻訳したい言語) の埋め込みと Positional Encoding を行ったものを入力としているが，デコーダでは参照している単語の次の単語を予測する必要がある．エンコーダ側の Multi-Head Attention では入力した単語の系統のすべての情報を取りこんで類似度を学習しているため，次の単語の情報を必然的に参照している．そのため，未来の単語を参照できなくするために，未来の単語が入力されていた場所には別の入力を入れることでその情報を無効にしている．なので，入力が異なるだけで構造自体はエンコーダ側の Multi-Head Attention と同じである．

例えば，英語から日本語に翻訳したいとき，図 1 の例をとると，以下のような入力が用意される．

エンコーダ側：[This, is, a, pen, .]
デコーダ側　：[<BOS >，これ, は, ペン, です,　。]

最初の BOS とは，Beginning of Sentence の略で文頭を表す仮想単語である．
デコーダ側の入力は，まず BOS から入力して，他のデータはマスクされているので BOS のみで学習を行い，出力で次の単語 (この場合は「これ」) が出力されるように学習を行っていく．そして，次の単語が出力されたらその単語を新たな入力として，マスクを解除してあげることでまた次の単語を学習していく．

ここで，別の入力とは，式 (3.4) のソフトマックス関数に対して $-\infty$ を入力することである．$softmax_i = e^{x_i} / \sum_j e^{x_j}$ なので，消したい x_j が $-\infty$ になると，$e^{x_j} \to e^{-\infty} = 0$ となるからである．

3.6 デコーダでの Multi-Head Attention

デコーダでは Musked Multi-Head Attention のあとにもう一つ Multi-Head Attention を設けている．この Multi-Head Attention の入力は, 入力系列をエンコーダに通したあとの値を $\mathbf{X}$, 出力系列を Masked Multi-Head Attention に通したあとの値を $\mathbf{Y}$ とすると, $\mathbf{Q} = \mathbf{Y}$, $\mathbf{K} = \mathbf{X}$, $\mathbf{V} = \mathbf{X}$ としている．すなわち，翻訳後 (Query) の単語が翻訳前 (Key-Value) のどの部分に注目するかを学習することができる．

このように，2 系列間で各トークン表現間の関連度を計算する Attention 機構を，**Source-Target Attention** という．対して，今まで紹介してきた Multi-Head Attention は，1 つの系列内での各トークン表現間の関連度を計算する Attention 機構であり，これを **Self Attention** と呼ぶ.

3.7 その他の機構

- 残差接続

  Transformer では，5 種類の層で残差接続を行っている．

  残差接続とは，図 5 に示すような接続であり，入力の値を出力の値に足し合わせて活性化を行う機構である．残差接続を行うことにより勾配消失問題などを解決できるため，Transformer はより多くの層を増やすことが可能となった.

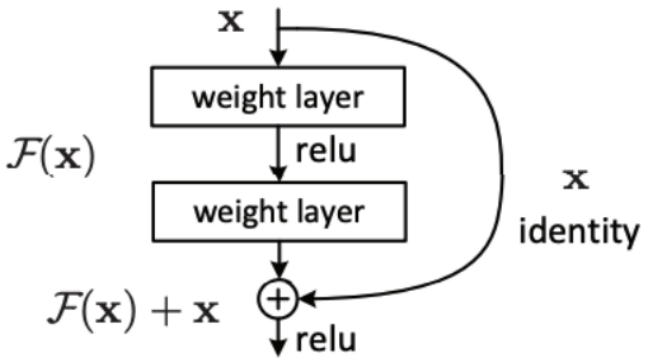

図 5　残差接続の構造

- レイヤー正規化

  層の出力に対して平均を 0，分散 1 に正規化する手法である．バッチ正規化では一つひとつの別々の値の対応する要素に対して正規化を行っているが，レイヤー正規化では，一つの値の要素に対して正規化を行っている．

  式で表すと以下のようになる．ここで，$h(t,i)$ は t 番目の単語の i 番目の要素を表しており，γ, β は学習可能なパラメータである．

$$\mu(t) = \frac{1}{d_{model}} \sum_{i=1}^{d_{model}} h(t,i) \tag{3.8}$$

$$\sigma(t) = \sqrt{\frac{1}{d_{model}} \sum_{i=1}^{d_{model}} (h(t,i) - \mu(t))^2} \tag{3.9}$$

$$LayerNorm(h,t) = \frac{\gamma}{\sigma(t)} \odot (h(t,i) - \mu(t)) + \beta \tag{3.10}$$

よって，最終的に，Add&Norm は次のように表記できる．

$$ResidualLayerNorm(x) = LayerNorm(sub_block(x) + x) \tag{3.11}$$

4　まとめ

本ゼミでは，自然言語処理のエンコーダ・デコーダモデルから，Transformer の構造について説明を行った．Transformer は近年の NLP で主力となっている BERT や XLNet，GPT などのベースとなっているため，学習していて，自然言語処理の歴史について学習することが出来た．実装まで至らなかったので，次のゼミでは何かしらの実装を行って発表したい．

参考文献

[1] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).