AttentionIsAllYouNeed の和訳

著者

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aiden N.Gomez, Lukasz Kaizer, Illia Polosukhin[1]

要旨

今までの主要な系列変換モデルは, エンコーダとデコーダを含む複雑な RNN や CNN に基づいていた. 最も良いパフォーマンスであったモデルも, エンコーダとデコーダを注意機構で接続しているモデルであった.

私達は, RNN や CNN を用いず, 注意機構のみからなる Transformer と呼ばれる新しい簡潔なモデルを提案する. 2 回の翻訳実験の結果, Transformer は今までのモデルより優れており, 更には, より並列化が可能で学習の時間も少ないことが分かった. Transformer は WMT2014 英独翻訳タスクにおいて,「プロの翻訳者の訳と近ければ近いほどその機械翻訳の精度は高い」という考え方に基づく機械翻訳の評価方法である BLEU スコア [2] で 28.4BLEU を記録した. これは, 複数のモデルを融合させて 1 つの学習モデルを生成するアンサンブル学習 [3] を含めたこれまでの最高記録を 2BLEU 上回る結果であった. また, WMT2014 英仏翻訳タスクにおいては, 8 個の GPU を用いた 3 日半の学習というこれまでの最先端のモデルの学習よりも遥かに少ない学習コストで, 41.0BLEU という単一モデルでの最高記録を打ち立てた.

1 序論

RNN, 特に RNN において文章の長期的な依存関係を学習できるようにした LSTM[4, 5] や gated RNN [6, 7] は, 言語モデルや機械翻訳 [8, 9, 10] などの系列変換問題への最適な手法として確固たる地位を築いていた. それ以来, RNN とエンコーダー-デコーダー構造の限界を押し上げる数々の努力がなされてきた.[11, 12, 13]

RNN では, 通常, 入力と出力の時系列データの時間的な位置に沿って計算する. 計算は逐次的に行われ, 時刻 t における隠れ状態 h_{t} は, 時刻 $t-1$ の隠れ状態の $h_{\mathrm{t-1}}$ と時刻 t における入力から導かれる. このように本質的に逐次的な性質を孕んでいるため, 学習の並列処理が困難である. そのため, メモリの制約上, 長い系列データなどの学習には致命的であった. 直近の研究では factorization tricks[14] や conditional computation[15] といった方法で計算効率がかなり改善され, 後者ではモデルの性能まで向上させることができたが, 逐次的な計算の問題は残ったままだった.

注意機構は入力と出力の系列データにおける距離を気にせず依存関係をモデル化することができ, 様々なタスクにおいて有効な系列変換モデルにおける必要不可欠な部分となっている.[9, 16] しかし, 一部の場合 [17] には注意機構は RNN と組み合わせて用いられる.

本研究で私達が提案する Transformer は, RNN を用いず, 注意機構のみで入力と出力の完全な依存関係を取り出すモデルのアーキテクチャである. Transformer は学習の並列処理が可能であり, 8 個の P100 GPU で 12 時間という小規模な学習後に, 過去最高の機械翻訳性能に達することができた.

2 背景

逐次的な計算を減らすという目標は, Extended Neural GPU[18], ByteNet[19], ConvS2S[20] といったモデルの基礎にもなっている. これらはどれも CNN を基本構成要素として, 入力と出力のすべての位置で隠れ状態の値を計算する. これらのモデルにおいて, 任意の入力の位置と出力の位置の信号を関連付けるために必要な計算時間は, ConvS2Sd では線形的, ByteNet では指数的に増加する. そのため離れた位置の依存関係を学習することはより困難になる.[21] Transformer では, この処理を定数時間で計算できる. 注意機構で重み付けした位置を平均化することで入力データに対する有効な解像度が下がってしまうが, 3.2 節で述べる多頭注意により相殺できる.

内部注意とも呼ばれる自己注意は, 単一系列内の異なる位置を参照し, 類似度で重み付けを行うことで系列要素を関連付ける注意機構である. 自己注意は文章読解, 要約, テキスト含意, 独立した文の表現の学習などのタスクで用いられ成功している.[22, 17, 23, 24]

End-to-End memory Networks は RNN の代わりに再帰的な Attention を元にしており, 単純な言語の質疑応答, 言語モデリングといったタスクにおいて優れた結果を示している.[25]

しかし私達が知る限り, Transformer は 入出力の表現を計算する際に, RNN や CNN を用いずに, 自己注意のみに依存した最初の系列変換モデルである. 次節以降では,Transformer, 自己注意について説明し [26, 19] や [20] といったモデルと比較した利点を議論する.

3 モデルアーキテクチャ

現時点で性能の良い系列変換モデルはエンコーダ-デコーダ構造を有している.[10, 9, 8] エンコーダは配列で表現される入力 $(x_1,...,x_n)$ を配列 $\boldsymbol{z}=(z_1,...,z_n)$ に変換する. デコーダは $\boldsymbol{z}$ から出力として配列 $(y_1,...,y_n)$ を 1 要素ずつ出力する. このステップで, このモデルは, 新しく生成する要素はこれまでに生成した要素のみに依存する自己回帰モデル [27, 28] であり, 直前に生成された要素を新しく入力として次の要素を生成する.

Transformer は図 1 に示すように, 全体としてはエンコーダ-デコーダ構造を踏襲しつつ, 自己注意層と point-wise 全結合層を積み重ねた層を使用している.

3.1 エンコーダとデコーダ

エンコーダ: 6 層からなり, 各層は全く同じ構造である. それぞれの層は 2 つの下位層を持ち, 下位層の後には残差接続 [29] と標準化 [30] が行われている. よって, 入力を x , 下位層自身の出力を Sublayer(x) として, 残差接続では x+Sublayer(x) , 下位層全体としての出力は LayerNorm(x+Sublayer(x)) となる [31]. 残差接続を容易にするために, すべての下位層, 埋め込み層は, 出力の次元を $d_{\text{model}} = 512$ としている.

デコーダ: デコーダも同一の 6 層からなる. エンコーダの持つ 2 つの下位層に加えて, エンコーダの出力を入力として受ける 3 つ目の下位層を加えている. エンコーダと同じく, 下位層の後には残差接続や標準化が行われている. ただ, 自己注意層はエンコーダ層とは異なり, 後続の要素を参照しないようにしている. このマスキングと, 出力が 1 要素ごとに補われていくことを組み合わせることで, マスキングにより i 番目の要素における予測が　i 未満の要素 における既知の出力のみに依存することが保証される.

3.2 注意機構

注意機構はクエリとキーとキー値のセットを出力へマッピングする機構である. ここで, クエリやキー, キー値, 出力はすべてベクトルである. 出力はキー値の重み付き和として計算され, それぞれのキー値への重みはクエリとクエリに対応するキーからの変換関数で計算される.

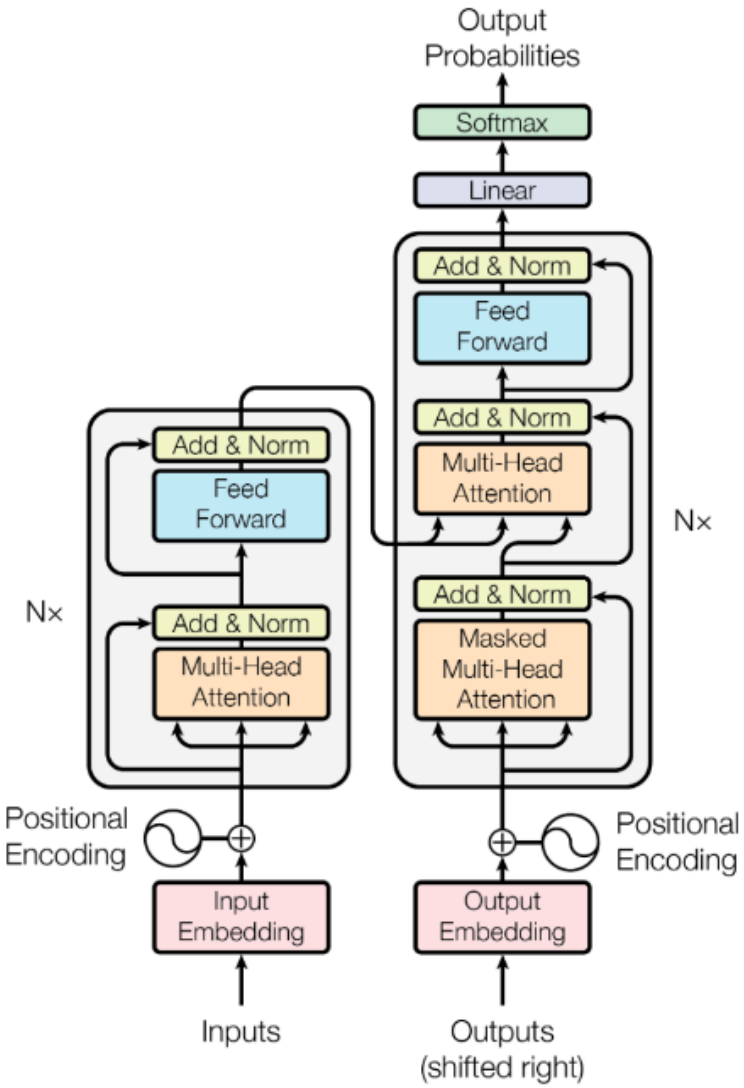

図 1: Transformer のモデルアーキテクチャ.

3.2.1 標準化内積注意

本研究で独自に考案した注意機構を”標準化内積注意”と呼ぶ. 図 2 に標準化内積注意のアーキテクチャを示す.

入力はクエリと次元 d_{k} のキー, 次元 d_{v} のキー値からなる. クエリとキーの内積を計算し, それを $\sqrt{d_{\mathrm{k}}}$ で割り, softmax 関数を適用しキー値の重みを得る. 実際にはクエリ, キーやキー値を行列 Q, K, V にまとめて計算している. 行列による注意機構の出力は (1) 式のように表せる.

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_{\mathrm{k}}}})V \tag{1}$$

最も一般的に利用される注意機構としては加法注意と内積注意の 2 つがある.[9] 内積注意は, 標準化内積注意から $\sqrt{d_{\mathrm{k}}}$ で割る処理を除いたものである. 加法注意は 1 つの隠れ層を持つ feed-forward network を用いて変換関数を計算している. 2 つの注意機構は理論的には似ているが, 高度に最適化された行列積のコードを内包しているため内積注意のほうが計算が遥かに早く, メモリ効率も良い.

d_{k} が小さい値の時はこの 2 つの注意機構は同様に作用するが, d_{k} が大きい値の時には, 加法注意は標準化を

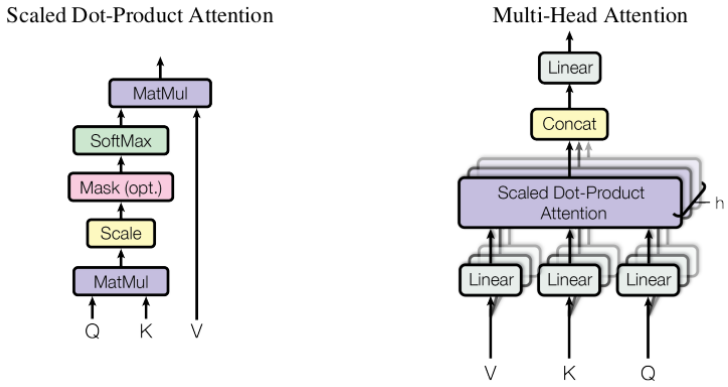

図 2: (左) 標準化内積注意. (右) 複数の注意機構層が並列に構成されている多頭注意.

しない内積注意より性能が良くなる.[32] 標準化内積注意では, d_{k} が大きい値である時に, 内積の値が急激に増加し, softmax 関数の勾配消失が起こると考え, それを防ぐために内積を $\frac{1}{\sqrt{d_{\mathrm{k}}}}$ で標準化している.

3.2.2 多頭注意

次元 d_{model} におけるキーとキー値とクエリを用いた単一の注意関数を使用する代わりに, クエリとキーとキー値に h 回それぞれ異なる変換が学習されている線形変換をし, 次元を d_{k}, d_{k}, d_{v} にそれぞれ削減したほうが都合が良いと判明した. 射影したクエリ, キー, キー値それぞれに対して, 注意関数を並列に実行し, 次元 d_{v} の出力を得る. それらを Concat 層で連結し, もう一度線形射影して最終的な出力を得る. 図 2 にこの一連の過程を示す.

多頭注意により, 異なる要素の異なる部分ベクトル空間を見ることができ, 結果として表現力が高まる [33]. 単頭注意では, このようなことはできない.

$$
\begin{aligned}
\mathrm{MultiHead}(Q,K,V) &= \mathrm{Concat}(head_1,...,head_{\mathrm{h}})W^O \\
\text{ここで}, head_{\mathrm{i}} &= \mathrm{Attention}(QW_{\mathrm{i}}^Q, KW_{\mathrm{i}}^K, VW_{\mathrm{i}}^V)
\end{aligned}
$$

射影関数のパラメータの行列は, $W_{\mathrm{i}}^Q \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{k}}}$, $W_{\mathrm{i}}^K \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{k}}}$, $W_{\mathrm{i}}^V \in \mathbb{R}^{d_{\mathrm{model}} \times d_{\mathrm{v}}}$ と
$W_{\mathrm{i}}^O \in \mathbb{R}^{hd_{\mathrm{v}} \times d_{\mathrm{model}}}$ である.

本研究では, $h = 8$ の並列の標準化内積注意層を用いて, 次元を $d_{\mathrm{k}} = d_{\mathrm{v}} = d_{\mathrm{model}}/h = 64$ とした. 次元の削減により, 総計算コストを, 次元の削減を行わなかった場合の単頭注意と同等まで削減することができた.

3.2.3 Transformer への注意機構の適用

Transformer では, 多頭注意を以下の 3 つの方法で採用している.

- "エンコーダ-デコーダ注意層"では, クエリはデコーダにおける直前の層から生まれ, キーとキー値はエンコーダの出力から生まれる. デコーダ内のすべての要素が入力文のすべての要素を見ることができる. すなわち系列間のトークン表現間で注意処理を行う. これは, [11, 9, 20] のような典型的な seq2seq モデルのエンコーダ-デコーダ注意機構を模倣している.
- エンコーダにおける自己注意層では, キー, キー値, クエリの全てが, 一つ前のエンコーダ層の出力から生まれている. エンコーダ内の系列内のそれぞれの要素は, エンコーダの一つ前の層の出力の系列内の全ての要素を参照することができる.

- 同様に, デコーダ内の自己注意層も, デコーダの全層の要素を参照することができるが, 自動回帰的な特性を保持するために, 左向きへの情報の流出を防がなければならない. すなわち, 翻訳済の単語に影響が出ないようにしなければ行けない. 本研究では標準化内積注意の中で, 不当な接続に相当する部分を-∞ にマスキングして, softmax 層の入力として与えることで, これを実現している. 図 2 にこの過程を示している.

3.3 Position-wise Feed-Forward Networks

注意層に加えて, エンコーダとデコーダ内のそれぞれの層には, 完全に連結した feed-forward network が含まれている. feed-forward network は,2 つの線形変換の間に ReLU 関数を適用した形となっている.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

線形変換は入力の異なる要素に対しても同じ値を掛けるのに対して, 線形層ごとに異なる値のパラメータを用いる. 言い換えると, カーネルサイズが 1 の 2 つの畳み込みとして捉えることができる. 入出力の次元は $d_{\mathrm{model}} = 512$ とし, feed-forward network 内では次元は $d_{\mathrm{ff}} = 2048$ としている.

3.4 埋め込み層と Softmax 層

他の系列変換モデルと同様に, 学習済みの埋め込み層を用いて入力と出力のトークンを次元 d_{model} のベクトルに変換する. また, 一般的な学習済みの線形変換と softmax 関数を用いて, デコーダの出力を予測済みのトークンの確率に変換している. Transformer は,[34] と同様に, 2 つの埋め込み層と softmax 層の前の線形変換で同じ重み行列を使用している. また, 埋め込み層では $\sqrt{d_{\mathrm{model}}}$ を掛けている.

3.5 Positional Encoding

私達のモデルは RNN や CNN を含んでいないため, モデルが系列内の要素の順番を利用するために, 絶対的であれ, 相対的であれ何らかの要素の順番の情報を定義する必要があった. 結果として, "positional encoder" を入力の埋め込み層に付け加え, エンコーダとデコーダの最下層に入れることとした. 埋め込み層との加算を行うために, positional encoding の次元は埋め込み層と同じ d_{model} とした.

本研究では, positional encoding として別々の周波数を持つ sin 関数, cos 関数を用いた.[20] を参考にし, 修正を加えた positional encoding となっている.

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{\mathrm{model}}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{\mathrm{model}}}) \end{aligned}$$

ここで, pos は要素の位置, i は次元である. つまり, position encoding のそれぞれの次元は正弦波に該当する. 波長は 2π から $10000 \cdot 2\pi$ まで段階的に増加していく. この関数を採用した理由は, モデルが系列の要素の相対的な位置を簡単に学習できると仮説を立てたためである. なぜなら, 任意の定数オフセット k において, PE_{pos+k} は PE_{pos} の線形関数として表すことができるからだ.

別の論文の学習済みの positional embedding[20] を用いた実験をしたところ, 表 3 の (E) 行に示すように, 本論文と別論文の結果はほぼ等しくなった. そこで Transformer では, 学習時よりもより長いデータに対しても対応できると判断し本研究の正弦波を用いた positional encoding を採用した.

4　なぜ自己注意を用いるか

本章では, 自己注意層における様々な側面を, RNN や CNN と比較する. RNN や CNN は可変長の配列 $(x_1,...,x_n)$ を, 同じ長さの配列 $(z_1,...,z_n)$ に変換し, などの典型的な系列変換エンコーダやデコーダの隠れ層に用いられる. 私達が自己注意層を使う理由としては, 3 つの大きな不足を感じる物事があったからである.

1 つ目は層ごとの計算の複雑さで, 2 つ目は本来並列化が可能な計算量である. これは必要最低限の配列計算量から算出できる.

3 つ目がモデルのネットワークで遠い位置関係の要素の依存関係を認識できる経路の長さである. 遠い位置関係の要素間の依存関係を学習することは, 多くの系列変換タスクにおいて重要な課題である. そのような依存関係を学習するための 1 つの重要な要素は順方向あるいは逆方向の信号がネットワークを通る経路の長さである. 入力と出力の位置関係の間の経路が短ければ短いほど, より長い位置関係での依存関係の学習が容易になる.[21] そこで, 本研究では異なる種類の層からなるネットワークにおいての入力と出力の位置間の最大の経路長を比較した. 表 1 にその結果を示す.

表 1: 異なる種類の層における層ごとの計算の複雑さ, 配列計算の計算量, 最大経路長.n は配列の長さ,d は表現の次元,k は CNN のカーネルのサイズ,r は制限付き自己注意の近傍サイズ.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attentional(restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

表 1 で示した通り, 配列計算において RNN が $O(n)$ 時間かかるのに対して, 自己注意は $O(1)$ 時間, すなわち定数時間で行うことができる. 計算の複雑さという観点では, $n < d$ である時, 自己注意は RNN より優れていると言える. なお, 機械翻訳において, ほとんどのケースが $n < d$ となっており, word-piece[11] と byte-pair[35] といった優れたモデルにおいても成り立っている. 非常に長い文章に関するタスクに関して計算のパフォーマンスを向上させるためには, それぞれの予測に対応する入力文のサイズ r の範囲しか考えないという制限を自己注意機構に課す方法がある. しかし, この方法では, 最大経路長が $O(n/r)$ となってしまう問題がある.

カーネルのサイズ k が, $k < n$ となるような 1 層の CNN は, 入力と出力の全てのペアを関連付けることはできない. そうするためには, 連続したカーネルの場合は $O(n/k)$ の積み重なった畳み込み層が必要, カーネルサイズを膨張させ各位置で広い範囲で疎に畳み込む膨張畳込み [19, 36] の場合でも $O(log_k(n))$ の層が必要となる. このため, 最大経路長が増えてしまう問題が生じる. CNN 層は, カーネルのサイズ k により, 一般的に RNN 層よりも計算が複雑になる. しかし, Separable convolutions[37] は計算の複雑さを $O(k \cdot n \cdot d + n \cdot d^2)$ まで減らしている. $k = n$ の場合, Separable convolutions は本研究で採用している自己注意層と point-wise 全結合層の組み合わせと計算の複雑さは等しくなる.

良い面として, 自己注意機構はより解釈しやすい系列変換モデルを得ることができる. モデルの注意分布について, 付録で例を示して議論する. 注意機構はそれぞれのタスクをきちんと学習するだけでなく, 文の構文規則や意味構造になぞらえた振る舞いをしているようだ.

5　学習

本章では, 私達のモデルの学習計画について記載する.

5.1 データセットとバッチング

英独翻訳に関して, 450 万もの文からなる standerd WMT 2014 English-German データセットを用いて学習を行った. 文はバイト対符号化 [32] で圧縮されており, 約 37,000 個のトークンの語彙が共有されている. 英仏翻訳では, 3600 万もの文からなる大規模な WMT 2014 English-French データセットを用いており, トークンを 32,000 個の語彙に分割して使用した [11]. 文のペアはおおよその文の長さでバッチ処理を行った. それぞれの学習用バッチは文章対がまとまっており, おおよそ 25,000 個のソーストークン, ターゲットトークンが含まれる.

5.2 実行環境

モデルの学習は, 8 個の NVIDIA P100 GPU を搭載した 1 つの計算機で行った. 本論文で述べたハイパーパラメータを使ったベースモデルでは, 訓練の 1 ステップあたり 0.4 秒かかった. ベースモデルは 100,000 ステップの合計で 12 時間学習を行った. 大きいモデルでは 1 ステップあたり 1.0 秒かかり, 300000 ステップ, 計 3.5 日かかった.

5.3 Optimizer

optimizer として Adam[38] を用いた. パラメータは $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$ とした. 学習率は学習を進めるに従って以下の式に応じて変更した.

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

学習の際, 学習率は $warmup\_step$ まで線形的に増えていくが, それ以降のステップでは, ステップ数の平方根の逆数に比例して減少していく. 本研究では, $warmup\_step = 4000$ とした.

5.4 正則化

実験の際, 3 つの正則化を採用した.

Residual Dropout それぞれの下位層の出力が残差接続される前に, 一定割合のノードを不活性化させながら学習を行うことで過学習を防ぐ Dropout 処理 [39, 40] を行った.

それに加えて, エンコーダーとデコーダー両方の埋め込み層と positional encoding の加算結果にも Dropout 処理を行った. ベースモデルでは,$P_{drop} = 0.1$ とした.

Label Smoothing また, 学習中において, 正解の場合の確率を 1.0, その他を 0.0 と決定するのではなく, 割引率 ϵ_{ls} だけ正解の確率を割引いて減らした値をその他に均等に分割することで過学習を防ぐ Label Smoothing[41] をした. 本研究では割引率は $\epsilon_{ls} = 0.1$ とした [42]. モデルが曖昧さを学ぶことで, 翻訳の正確さの指標である perplexity は減るものの, 結果として BLEU スコアは向上する.

6 結果

6.1 機械翻訳タスク

WMT2014 英独翻訳において, 表 2 に示すように, ビックモデルがアンサンブルを含む既知のモデルに 2.0BLEU 以上の差をつけて 28.4BLEU という最高記録を打ち立てた. 表 3 の下部にこのモデルの構成要素を示す. 学習は 8 個の P100 GPU で 3.5 日間かかった. ベースモデルは, 既存のモデルに比べて数分の 1 の学習コストで既存のモデルやアンサンブルに比べて良い BLEU スコアを残している.

WMT2014 英仏翻訳タスクに関して, ビッグモデルでは 41.0BLEU スコアを記録している. 既存の単一のモデルの全てより良いスコアであり, 既存の最良のモデルに比べ 1/4 以下の学習コストで学習を行っている. ビックモデルは英仏翻訳タスクに関しては,Dropout rate P_{drop} を 0.3 の代わりに 0.1 と設定している.

ベースモデルでは, 10 分間隔で保存されるチェックポイントの最後の 5 回分を平均化したモデルとした. ビックモデルは最後の 20 回分を平均化した. また, 推論時は,beam size を 4, Length Penalty の α を 0.6 としてビームサーチを使用している [11]. これらのハイパーパラメータは検証データを用いた実験後に決定した. 推論時の出力の最大長は入力系列長 +50 までとし, 可能なら早めに打ち切るようにした [11].

表 2 に, 本研究のモデルと参考文献のモデルとの翻訳の質と計算コストを示す.

表 2: Transformer は, 英独翻訳, 英仏翻訳の 2014 年最新のタスクで以前の最先端のモデルに比べて, 僅かな計算コストで良い BLEU スコアを残した.

model	BLEU		Training Cost(FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet[19]	23.75			
Deep-Att + PosUnk[43]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL[11]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S[20]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.4 \cdot 10^{20}$
MoE[15]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble[43]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble[11]	26.30	41.16	$7.7 \cdot 10^{20}$	$1.2 \cdot 10^{20}$
ConvS2S Ensemble[20]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer(base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer(big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Training Cost の指標として, コンピュータが 1 秒間に処理可能な浮動小数点演算の回数を表した FLOPs[44] を用いており, 学習時間,GPU の使用数, それぞれの GPU における処理可能な単精度浮動小数点の推定値をかけ合わせて計算した.

6.2 モデルの変動

Transformer の各構成要素を評価するために, ベースモデルを様々に変化させ, Newest2013 の開発セットにおける英独翻訳タスクでの性能を評価した. 前節で述べたビームサーチを用いたが, チェックポイントの平均化は行わなかった. 表 3 に結果を示す.

表 3 の (A) 行では, 3.2.2 節で述べた通り計算量を一定にしたまま, 多頭注意におけるヘッドの数 h , キーやキー値の次元 d_{k} , d_{v} を変化させた結果を示している. $h = 1$ の時は, 最適なパラメータ設定のときに比べ, 0.9 ほど BLEU スコア が低くなっており, $h = 32$ など多すぎる場合もスコアが低くなっている.

(B) 行では, 注意機構のキーの次元 d_{k} の影響を観察した. 結果として, 優劣をつけるのは難しく, 標準化内積注意より良い変換関数があるかもしれない. (C), (D) 行では, エンコーダ, デコーダの層の数 N , モデルの入出力の次元 d_{model} , Feed-Forward Networks 内の次元 d_{ff} , Dropout や LabelSmoothing のパラメータ P_{drop} , ϵ_{ls} を変化させて実験をした結果を示している. 想定通り, パラメータを増やしモデルの規模を大きくするほどよい結果が得られ, また Dropout 処理は過学習を避けるために有効であった. 行 (E) では, 本研究の正弦波を用いた positional encoding を他の研究の positional encoding[20] と置き換えた結果を示しており, 大きな違いは見られなかった.

表 3: Transformer アーキテクチャの様々な変化. 表に乗せていない変数はベースモデルと同一である. 表ではバイト対符号化による単語列あたりの PPL を計算しており, 単語あたりの PPL と比較すべきではない.

	N	d_{model}	d_{ff}	h	d_{k}	d_{v}	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	6									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

7 結論

本研究では, Transformer を提案した. Transformer は完全に注意機構のみに依存した最初の系列変換モデルであり, エンコーダ-デコーダアーキテクチャで最も一般的な RNN 層を多頭自己注意層に置き換えたモデルである.

翻訳タスクにおいて, Transformer は RNN や CNN レイヤーを基にしたアーキテクチャよりも遥かに早く学習が可能である. WMT2014 英独翻訳, WMT2014 英仏翻訳タスクで最高記録を達成した. 前者のタスクでは, 過去のアンサンブルも含めた全てのモデルより高性能だった.

注意機構を基としたモデルの未来には期待でき, 他のタスクにも応用していく予定である. 文章以外でも, 画像, 音声, 映像といった大容量の入力にも対応できるようにしたい. 学習に使用したコードは以下に示す.

`https://github.com/tensorflow/tensor2tensor`

参考文献

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, p. arXiv:1706.03762, June 2017.

[2] TOIN. 機械翻訳の評価に最もよく用いられる「bleu スコア」とは, 2020.03.02. https://to-in.com/blog/102282.

[3] codExa. 機械学習上級者は皆使ってる?! アンサンブル学習の仕組みと 3 つの種類について解説します, 2018.06.21. https://www.codexa.net/what-is-ensemble-learning/.

[4] KojiOhki. Lstm ネットワークの概要, 2017.12.11. https://qiita.com/KojiOhki/items/89cd7b69a8a6239d67ca.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

[6] DeepAge. Rnn：時系列データを扱う recurrent neural networks とは, 2017.05.23. https://deepage.net/deep_learning/2017/05/23/recurrent-neural-networks.html.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv e-prints*, p. arXiv:1412.3555, December 2014.

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, p. 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv e-prints*, p. arXiv:1409.0473, September 2014.

[10] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv e-prints*, p. arXiv:1406.1078, June 2014.

[11] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv e-prints*, p. arXiv:1609.08144, September 2016.

[12] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv e-prints*, p. arXiv:1508.04025, August 2015.

[13] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the Limits of Language Modeling. *arXiv e-prints*, p. arXiv:1602.02410, February 2016.

[14] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. *arXiv e-prints*, p. arXiv:1703.10722, March 2017.

[15] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv e-prints*, p. arXiv:1701.06538, January 2017.

[16] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured Attention Networks. *arXiv e-prints*, p. arXiv:1702.00887, February 2017.

[17] Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A Decomposable Attention Model for Natural Language Inference. *arXiv e-prints*, p. arXiv:1606.01933, June 2016.

[18] Łukasz Kaiser and Samy Bengio. Can Active Memory Replace Attention? *arXiv e-prints*, p. arXiv:1610.08613, October 2016.

[19] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural Machine Translation in Linear Time. *arXiv e-prints*, p. arXiv:1610.10099, October 2016.

[20] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. *arXiv e-prints*, p. arXiv:1705.03122, May 2017.

[21] Sepp Hochreiter and Yoshua Bengio. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001.

[22] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long Short-Term Memory-Networks for Machine Reading. *arXiv e-prints*, p. arXiv:1601.06733, January 2016.

[23] Romain Paulus, Caiming Xiong, and Richard Socher. A Deep Reinforced Model for Abstractive Summarization. *arXiv e-prints*, p. arXiv:1705.04304, May 2017.

[24] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A Structured Self-attentive Sentence Embedding. *arXiv e-prints*, p. arXiv:1703.03130, March 2017.

[25] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, Vol. 28, pp. 2440–2448. Curran Associates, Inc., 2015.

[26] Łukasz Kaiser and Ilya Sutskever. Neural GPUs Learn Algorithms. *arXiv e-prints*, p. arXiv:1511.08228, November 2015.

[27] biostatistics. 自己回帰（ar）モデル. `https://stats.biopapyrus.jp/time-series/ar-model.html`.

[28] Alex Graves. Generating Sequences With Recurrent Neural Networks. *arXiv e-prints*, p. arXiv:1308.0850, August 2013.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, p. arXiv:1512.03385, December 2015.

[30] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv e-prints*, p. arXiv:1607.06450, July 2016.

[31] DeepAge. Residual network(resnet) の理解とチューニングのベストプラクティス, 2016.11.30. `https://deepage.net/deep_learning/2016/11/30/resnet.html`.

[32] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive Exploration of Neural Machine Translation Architectures. *arXiv e-prints*, p. arXiv:1703.03906, March 2017.

[33] CVML エキスパートガイド Masaki Hayashi. マルチヘッドアテンション (multi-head attention) [transformer], 2022.07.02. `https://cvml-expertguide.net/terms/dl/seq2seq-translation/transformer/multi-head-attention/`.

[34] Ofir Press and Lior Wolf. Using the Output Embedding to Improve Language Models. *arXiv e-prints*, p. arXiv:1608.05859, August 2016.

[35] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[36] Masaki Hayashi. 膨張畳み込み (dilated convolution), 2022.05.15. https://cvml-expertguide.net/terms/dl/layers/convolution-layer/dilated-convolution/.

[37] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv e-prints*, p. arXiv:1610.02357, October 2016.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, p. arXiv:1412.6980, December 2014.

[39] Shuhei Kishi. 【ニューラルネットワーク】dropout(ドロップアウト) についてまとめる, 2017.07.18. https://qiita.com/shu_marubo/items/70b20c3a6c172aaeb8de.

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, Vol. 15, No. 56, pp. 1929–1958, 2014.

[41] T-STAR. Online label smoothing の実装と評価, 2021.04.01. https://qiita.com/T-STAR/items/a3bdcd1ae00150fe1402.

[42] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *arXiv e-prints*, p. arXiv:1512.00567, December 2015.

[43] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep Recurrent Models with Fast-Forward Connections for Neural Machine Translation. *arXiv e-prints*, p. arXiv:1606.04199, June 2016.

[44] FUJITSU. お答えします スパコン q&a. https://www.fujitsu.com/jp/about/businesspolicy/tech/k/qa/k02.html.