AttentionIsAllYouNeed の和訳

Abstruct

今までの主要な sequence transduction model は, エンコーダーとデコーダーを含む複雑な 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, エンコーダーとデコーダーをアテンションの仕組みを用いて接続しているモデルであった.

私達は,RNN や CNN を用いず, アテンションのみからなる 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 3.5 日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コストで,41.0BLEU という単一モデルの最高記録を打ち立てた.

1 Introduction

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 や gated RNN は, 言語モデルや機械翻訳などの Sequence 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,Recurrent 言語モデルとエンコーダー-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.

リカレントモデルでは, 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を行う. よって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, 長い時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 や conditional computation 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.

Attention は入力と出力の時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 sequence model と transduction model の必要不可欠な部分となっている. しかし, 一部の場合には Attention は RNN と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,Attention 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, 最高の機械翻訳性能に達することができた.

2 BackGround

逐次的な計算を減らすという目標は,Extended Neural GPU,ByteNet,ConvS2S 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. また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 では指数的となる. 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. Transformer では, この計算時間を定数時間に減らすことができる.Attention で重み付けした位置を平均化することで有効な解像度が下がってしまうが,3.2 説で述べる Multi-Head Attention により相殺できる.

intra-attention とも呼ばれる Self-Attention は, 単一の文章の異なる位置を関連付ける Attention である.Self-Attention 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.

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.

しかし私達が知る限り,Transformer は RNN や CNN を用いずに入出力の表現を計算するために,Self-Attention のみに依存した最初の transduction model である. 次節以降では,Transformer,self-attention について説明しこれまでのモデルと比較した利点を議論する.

3 Model Architecture

最も優位性のある sequence transduction models はエンコーダー-デコーダー構造を有している. エンコーダー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. デコーダーは $\boldsymbol{z}$ 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, モデルが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 の左半分と右半分に示すように, 全体としてはエンコーダー-デコーダー構造を踏襲しつつ,self-attention 層と point-wise 全結合層を積み重ねた層を使用している.

3.1 Encoder and Decoder Stacks

エンコーダー: 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や標準化が行われている. よって, 下位層自身の出力を Sublayer(x) として, 下位層全体としての出力は LayerNorm(x+Sublayer(x)) となる. 残差接続を容易にするために, すべての下位層,embedding layers も出力の次元を d_{model}=512 としている.

デコーダー: デコーダーも同一の 6 層からなる. エンコーダーの 2 つの下位層に加えて, エンコーダー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. エンコーダーと同じく, 下位層の後には残差接続や標準化が行われている. ただ,self-attention 下位層は改良しており, 後続の要素が影響しないようにしている. このマスキングと, 出力が 1 要素ごと補われることを組み合わせることで, マスキングにより地点 i での予測が地点 i 未満での既知の出力のみに依存することが保証される.

3.2 Attention

アテンションの仕組みは, クエリとキーとキー値のセットを出力へマッピングし, 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割り当てられる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.

3.2.1 Scaled Dot-Product Attention

私が独自に考案したアテンションを”Scaled Dot-Product Attention”と呼ぶ. 図 2 に Scaled Dot-Product Attention 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 $\sqrt{d_k}$ で割り, キー値の重みを得るために softmax 関数を適用する. 実際にはクエリを行列 Q としてまとめて同時に計算している. キーやキー値も同様に行列 K,V にまとめて計算している. 行列による出力は (1) 式のように表せる.

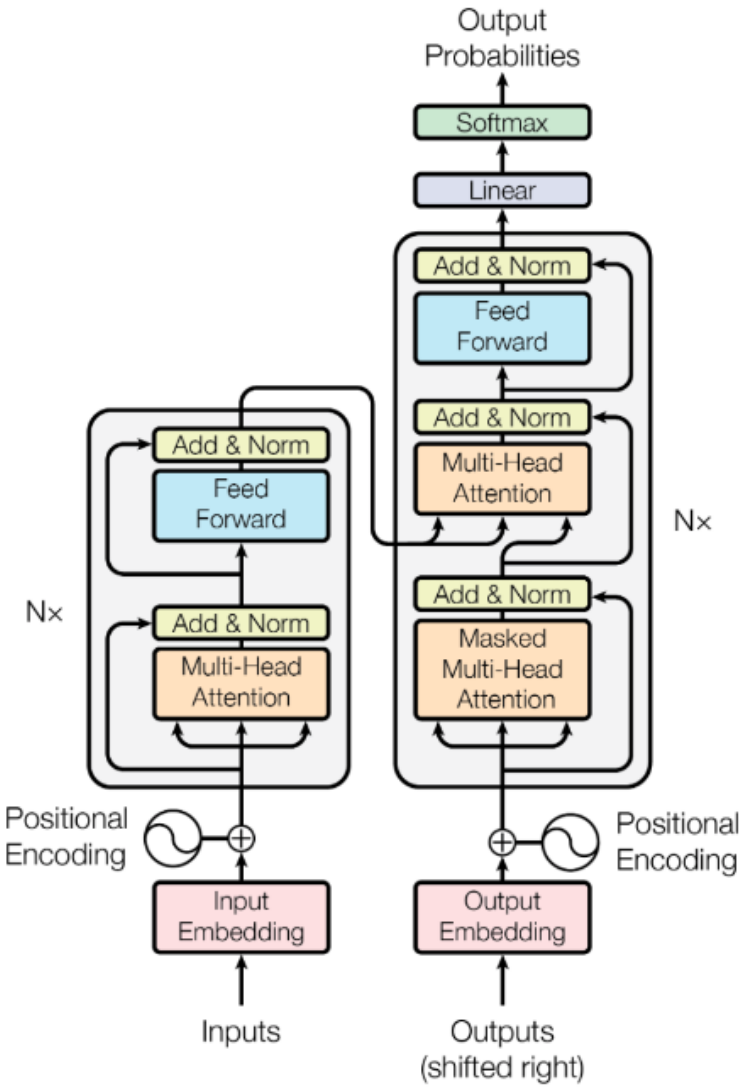

図 1: Transformer のモデルアーキテクチャ.

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

最も一般的に利用される Attention としては additive attention と dot-product attention の 2 つがある.Dot-product attention は,Scaled Dot-Product Attention から $\sqrt{d_\text{k}}$ で割る処理を除いたものである.Additive attention は一つの隠れ層を持つ feed-forward network を用いて変換関数を計算している.2 つの Attention 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 dot-product attention のほうが計算が遥かに早く, メモリ効率も良い.

d_k が小さい値である時はこの 2 つの Attention は同様に作用するが,d_k が大きい値である時には,Additive attention は標準化をしない dot-product attention より性能が良くなる. 私達は,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 $\frac{1}{\sqrt{d_\text{k}}}$ で標準化している.

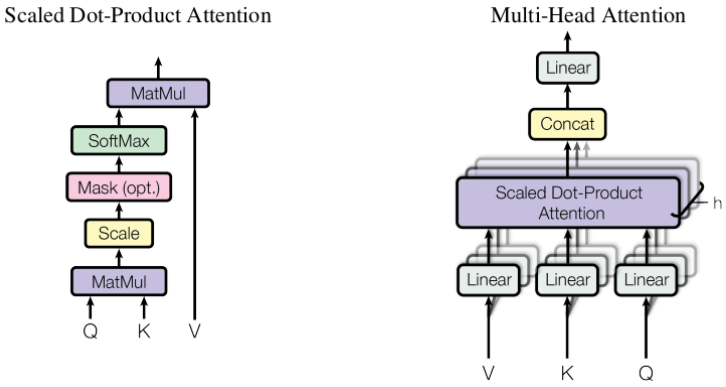

図 2: (左)Scaled Dot-Product Attention. (右) 複数のアテンション層が並列に構成されている Multi-Head Attention.

3.2.2 Multi-Head Attention

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 attention 関数を使用する代わりに, クエリとキーとキー値に h 回線形射影を行い, 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,Attention 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.

Multi-Head Attention 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る.single attention head では, 平均化によりこのようなことはできない.

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

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

本研究では,$h = 8$ の並列の attention 層を用いて, 次元を $d_{\text{k}} = d_{\text{v}} = d_{\text{model}}/h = 64$ 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一の attention と同等まで削減することができた.

3.2.3 Applications of Attention in our Model

Transformer では,multi-head attention を以下の 3 つの方法で採用している.

- ”encoder-decoder attention”層では, クエリは直前のデコーダー層から生まれ, キーとキー値はエンコーダーの出力から生まれる. デコーダー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. これは,sequence to sequence モデルの典型的な encoder-decoder attention を模倣している.

- エンコーダは self-attention 層を含んでいる.self-attention 層では, キー, キー値, クエリの全てが, エンコーダー内の一つ前の層の出力から生まれている. エンコーダー内のそれぞれの要素は, エンコーダーの一つ前の層の, その要素までの全ての要素を見ることができる.

- 同様に, デコーダー内の self-attention 層も, デコーダー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. 私達は scaled dot-product attention 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

Attention の下位層に加えて, エンコーダーとデコーダー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.

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

線形変換は異なる要素に対しても同じ値を掛けるのに対して, 層を変えることに異なる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 Embeddings and Softmax

他の sequence transduction model と同様に, 学習済みの embedding 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ーの出力を予測済みの next-token の予測確率に変換している. 私達のモデルは,2 つの embedding 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,embedding 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 embedding 層に付け加え, エンコーダーとデコーダーの最下層に入れることとした.embedding 層との加算を行うために,positional encoding の次元は embedding 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.

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

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, モデルが相対的な位置から参照できるように簡単に学習できると仮設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 を用いた実験を行ったところ, 本論文と別論文の結果はほぼ等しくなった. そこで, 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正弦波 ver を採用した.

4 Why Self-Attention

本章では,self-attention 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, エンコーダーやデコーダーなどの典型的な sequence transduction の隠れ層に用いられる. 私達が self-attention 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ある.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の依存関係を学習することは, 多くの sequence transduction 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 つの重要な要因は順方向か逆方向か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. 従って, 私達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 Seld-Attention 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 RNN が $O(n)$ 時間かかるのに対して,Self-Attention 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, 一般的に機械翻訳での文の表現では $n < d$ であることから,Self-Attention は RNN より優れていると言える. 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 Self-Attention 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. これは将来の研究の課題である.

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, 膨張畳込みの場合でも $O(log_k(n))$ 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 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 self-attention 層と point-wise feed-forward network 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,self-attention はより解釈しやすいモデルを得ることができる. モデルの attention 分布について, 付録で例を示して議論する.Attention 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 Training

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 Training Data and Batching

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 byte-pair encoding でエンコードされており, 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た. トークンを 32,000 個の語彙に分割して使用した. 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 source token,target token が含まれる.

5.2 Hardware and Schedule

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 100000 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 を用いた. パラメータは $\beta_1 = 0.9$,$\beta_2 = 0.98$, $\epsilon = 10^{-9}$ とした. 学習率は学習を行う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.

5.4 Regularization

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 Dropout 処理を行った. それに加えて, エンコーダーとデコーダー両方の embeddings と positional encoding の和にも Dropout 処理を行った. ベースモデルでは,$P_{drop} = 0.1$ とした.

Label Smoothing 学習中,$\epsilon_{ls} = 0.1$ として Label Smoothing を行った. 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,BLEU スコアは向上する.

6 Results

6.1 Machine Translation

WMT2014 英独翻訳において, 表 2 に示すように,big 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.base モデルは, 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上に, 既存のモデルに比べて数分の 1 の計算コストで学習を行っている.

WMT2014 英仏翻訳タスクに関して,big 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.big 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.

base モデルでは,10 分感覚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.big モデルは最後の 20 回分を平均化した. また,beam size を 4,Length Penalty の $\alpha = 0.6$ として beam search を使用している. これらのハイパーパラメータは開発セットの実験後に決定した. 推論時の出力の最大長は入力の長さ+50 までとし, 可能なら早めに打ち切るようにした.

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.

Training Cost 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 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.

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, 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	23.75			
Deep-Att + PosUnk		39.2		$1.0 \cdot 10^{20}$
GNMT + RL	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S	25.16	40.46	$9.6 \cdot 10^{18}$	$1.4 \cdot 10^{20}$
MoE	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble	26.30	41.16	$7.7 \cdot 10^{20}$	$1.2 \cdot 10^{20}$
ConvS2S Ensemble	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**	$2.3 \cdot 10^{19}$	

6.2 Model Variations

Transformer の各構成要素を評価するために,base 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,Multi-Head Attention の heads 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,Attention 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,Scaled Dot-Product Attention より良い変換関数があるかもしれない.(C),(D) 行では, エンコーダー, デコーダーの層の数 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},$\epsilon_{ls} = 0.1$ を変化させて実験を行っ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 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.

7 Conclusion

本研究では,Transformer を提案した.Transformer は完全に Attention のみに依存した最初の sequence transduction モデルであり, エンコーダー-デコーダーアーキテクチャで最も一般的な RNN レイヤーを multi-headed self-attention について説明しこれまでのモデルと比較した利点を議論する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だある.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.

Attention 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

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

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 base モデルと同一である. 表の PPL は byte-pair encoding による単語列当たりで計算しており, 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

参考文献