情報工学英語演習 「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.

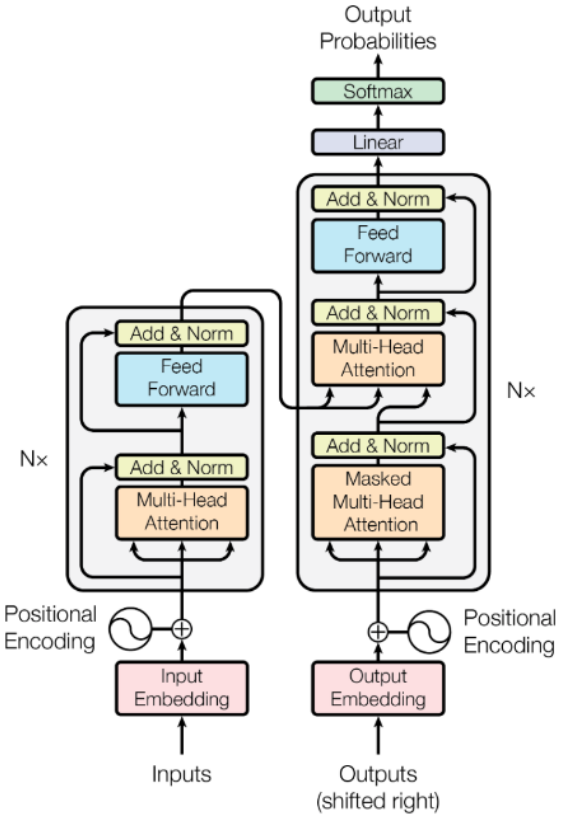

図 1: Transformer のモデルアーキテクチャ.

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.

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

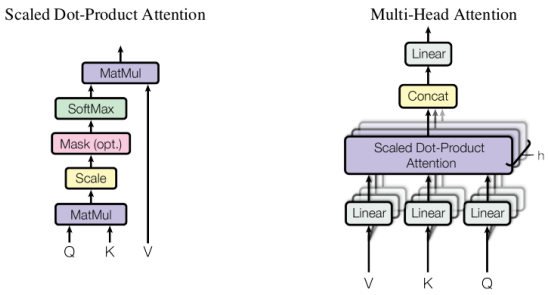

図 2: (左)Scaled Dot-Product Attention. (右) 複数のアテンション層が並列に構成されている Multi-Head Attention.

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 $\sqrt{d_{\mathrm{k}}}$ 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 $\frac{1}{\sqrt{d_{\mathrm{k}}}}$ で標準化している.

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.

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

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 attention 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.

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

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

5 Training

5.1 Training Data and Batching

5.2 Hardware and Schedule

5.3 Optimizer

5.4 Regularization

6 Results

6.1 Machine Translation

6.2 Model Variations

7 Conclusion

参考文献