●Section5：Transformer

◇Transformer 概要

2017 年中旬、Google が発表した論文「Attention is all you need」で提出されたモデル「Transformer」は、深層学習の自然言語処理(NLP)分野でいまはデファクトスタンダードとして使われています。Transformer は要するに、過去の自然言語処理(NLP)で多く使われる再帰型ニューラルネットワーク(RNN)や畳み込みニューラルネットワーク(CNN)を「Self-Attention Layer」に入れ替えたモデルです。

それ以降、大抵の自然言語処理(NLP)分野で発表された研究や論文は、Transformer で再構築して発表されています。期待を裏切らなく、再構築されたモデルの結果は、以前の結果を上回っています。

2018 年 Google が発表した自然言語処理(NLP)モデル BERT は、教師なしで学習した Transformer であり、2019 年 OpenAI が「あまりにも危険で発表したくない AI」と自己評価したモデル GPT2 は、Masked Self-Attention Layer を使用した BERT です。

引用サイト：自然言語処理の巨獣「Transformer」の Self-Attention Layer 紹介

◇RNN の概要復習

言語モデルは、単語の並びに確率を与える。

・単語の並びに対して尤度、すなわち文章としての自然度(事後確率最大)を確立で評価する

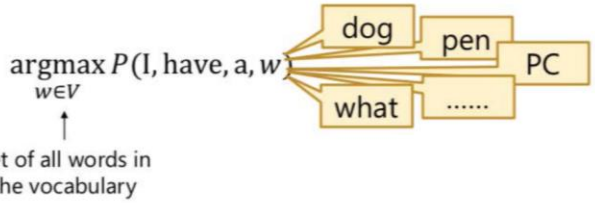
例)
- You say goodbye → 0.092 (自然)
- You say good die → 0.00000032 (不自然)

・数式的には同時確立を事後確率に分解して表せる

$$\mathcal{P}(w_1, \cdots, w_m) = \prod_{i=1}^{m} \mathcal{P}(w_i | w_1, \cdots, w_{i-1})$$

時刻t-1までの情報で、時刻tの事後確率を求めることが目標

→これで同時確率が計算できる



・RNN は系列情報を内部状態に変換する事が出来る

・文章の各単語が現れる際の同時確立は、事後確率で分解できる

⇒事後確率を求める事が RNN の目標である。

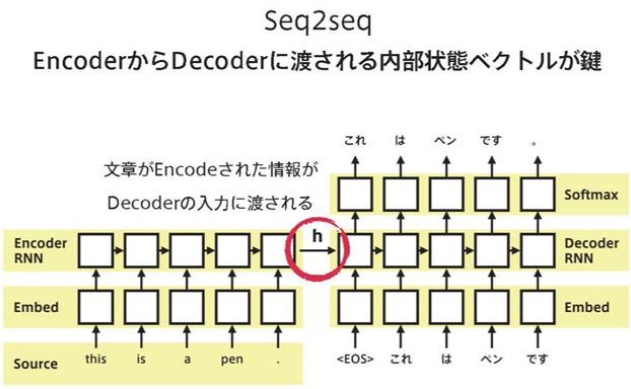
・言語モデルを再現するように、RNN の重みが学習されていれば、ある時点の次の単語を予測できる。

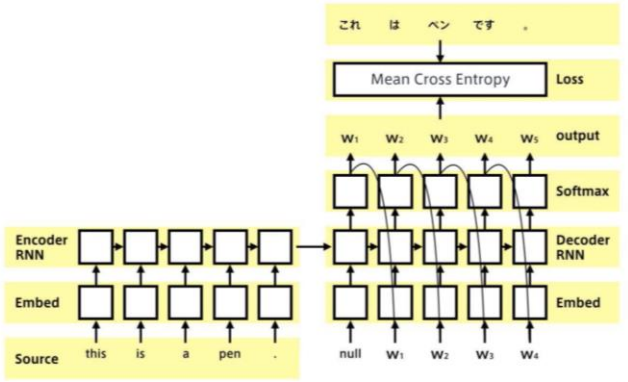
⇒先頭単語を与えれば、文章を生成する事も可能（decoder)。

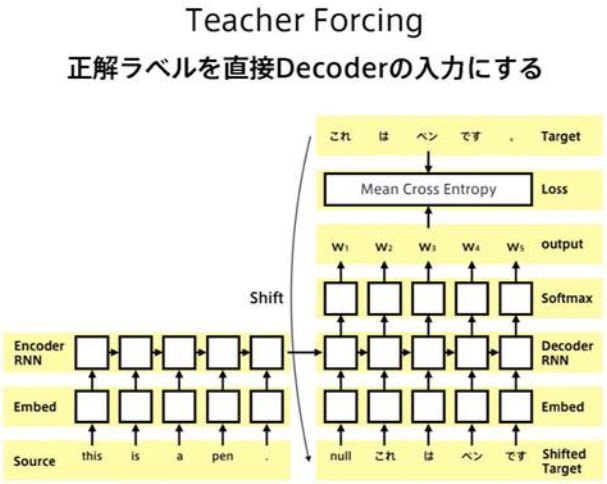
◇**Seq2seq の概要復習**

Encoder-Decoder モデルであり、系列（Sequence）を入力＝Encoder(特徴量をベクトル化)として、ベクトルを系列に変換=Decoder し出力する。

翻訳/音声認識（波形→テキスト）/チャットボット（テキスト→テキスト）



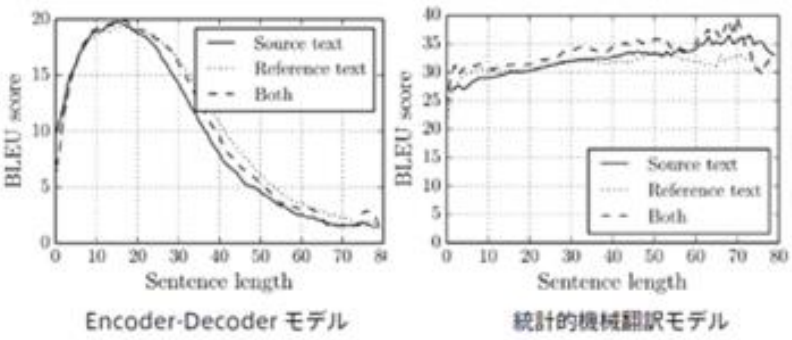




〈Seq2seq の課題〉

翻訳元の分の内容を一つのベクトルで表現

→文が長くなると表現しきれない

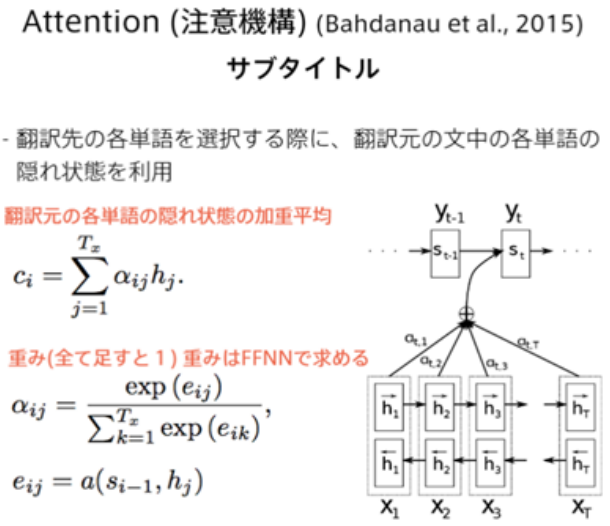


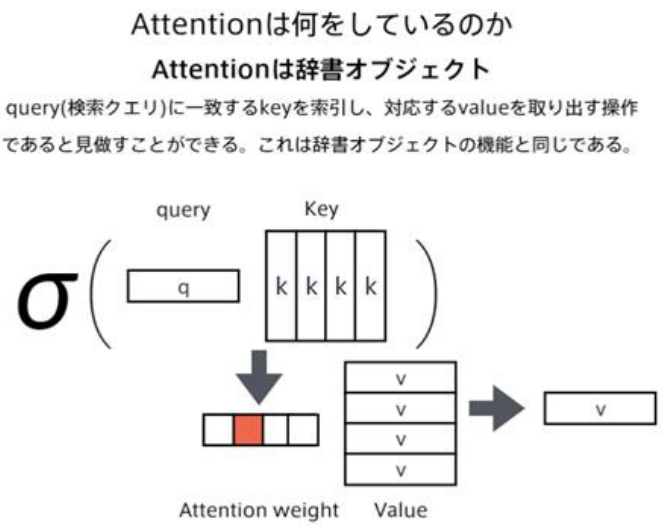
◇Transfomer

・Self-Attention Layer(Self Attention：自己注意機構)

2015 年に発表された Attention(注意機構)とは、単語同士の関係を行列で表す方法。

本来はエンコーダーとデコーダーとして動作しているが、Self-Attention Layer はその方法を Layer 内で実現し計算の高速化を実現した。

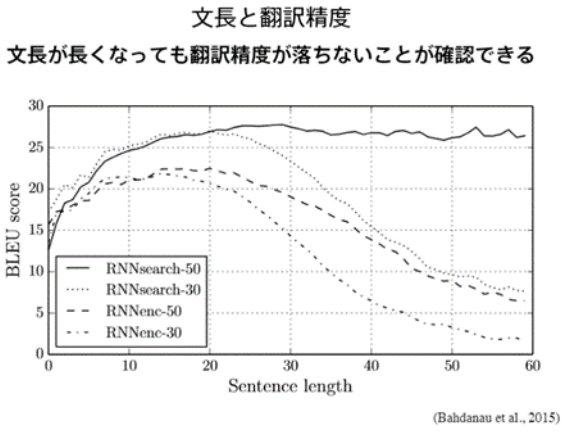




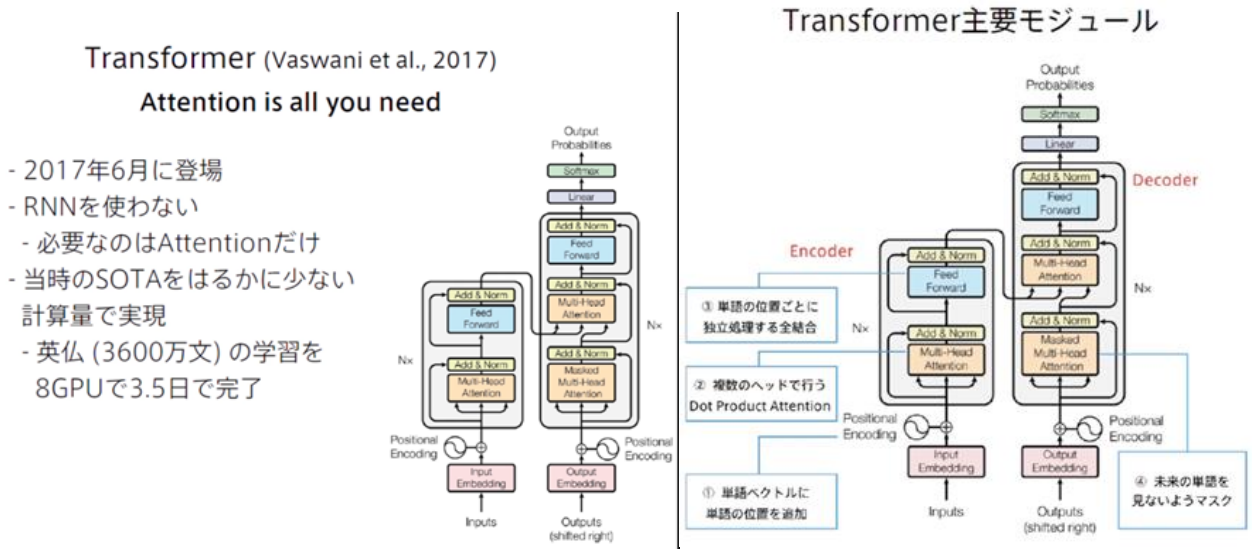
Self-Attention Layer は従来の自然言語処理(NLP)モデル構造と比べると、主に二つのメリットがあります。

① 並列計算可能（再帰型ニューラルネットワークと比べ)

② 長文の為の深いモデル構築不要（畳み込みニューラルネットワークと比べ)



Self-Attention Layer では入力文章内の単語関係を一度に見る事を実現する事により、層内を単純な行列計算にして計算ユニットでの並列計算を可能にした。さらに、Self-Attention Layer ではマルチヘッド(Multi-Head)構造という特徴があって、様々な単語関係の種類や入力文章内単語の間合いで単語の予測をする事ができる。



左側がエンコーダ，右側がデコーダである．
それぞれ灰色のブロックを 6 個スタックしている（N=6).
・エンコーダ: [自己注意，位置毎の FFN] のブロックを 6 層スタック
・デコーダ: [(マスキング付き）自己注意，ソースターゲット注意，位置毎の FFN] のブロックを 6 層スタック
ブロック内は残差接続（Residual Connection）と層正規化（Layer Normalization）を適応する．
　ネットワーク内の特徴表現は [単語列の長さ x 各単語の次元数] の行列で表される．注意の層を除いて 0 階の各単語はバッチ学習の各標本のように独立して処理される．

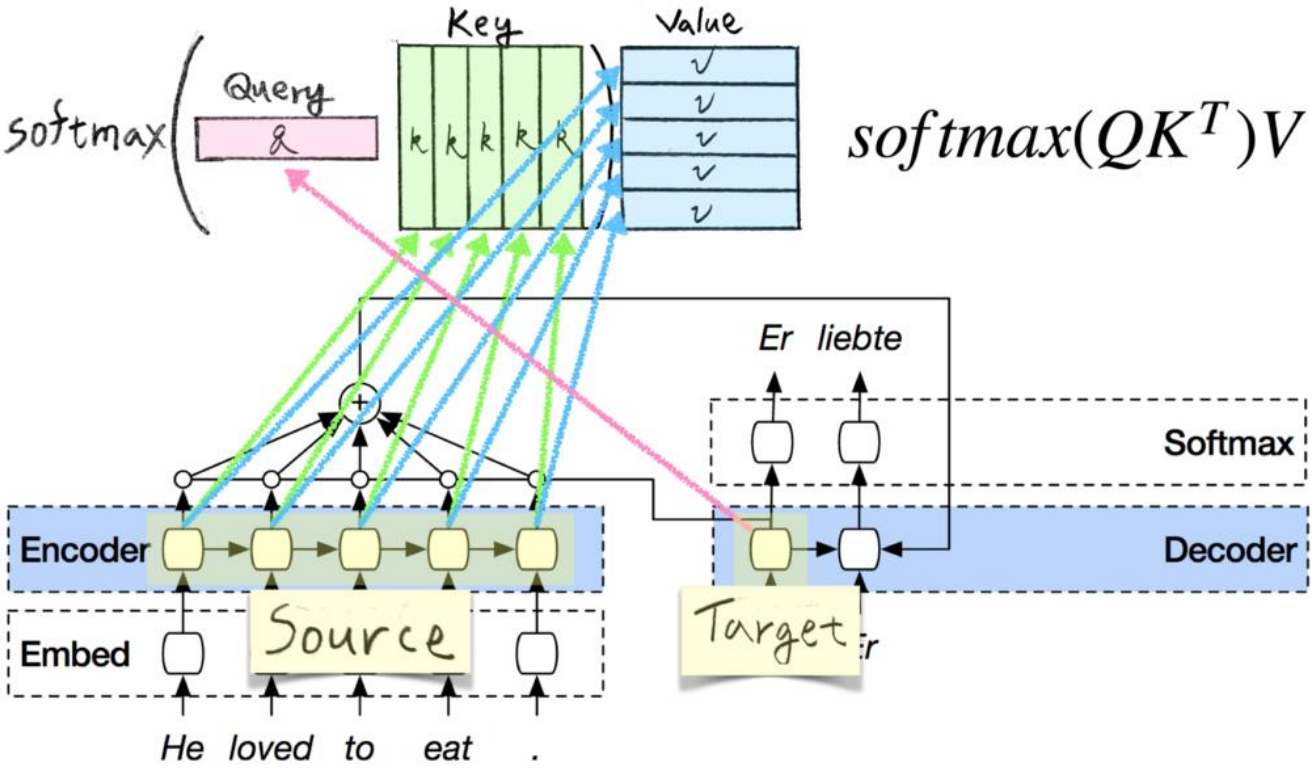
一般的な Encoder-Decoder の注意は、
・エンコーダの隠れ層を Source,
・デコーダの隠れ層を Target
として次式によって表される．

$$\boldsymbol{Attention(Target, Source) = Softmax(Target \cdot Source^T) \cdot Source}$$

より一般化すると Target を query (検索クエリ）と見做し，Source を Key と Value に分離する．

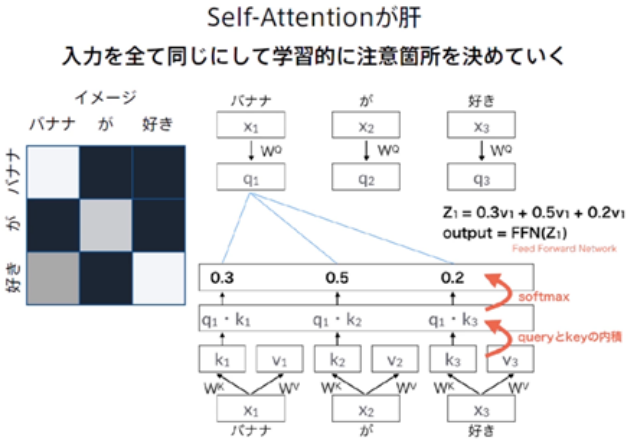
$$\boldsymbol{Attention(query, Key, Value) = Softmax(query \cdot Key^T) \cdot Value}$$

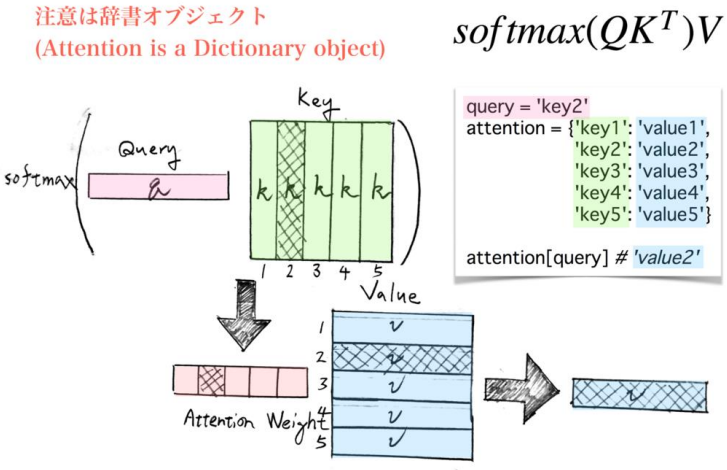
注）イニシャルが小文字の query,key,value（ないし q,k,v）はベクトル，大文字の Query,Key,Value (ないし Q,K,V）は行列（ベクトルの配列）を表す。



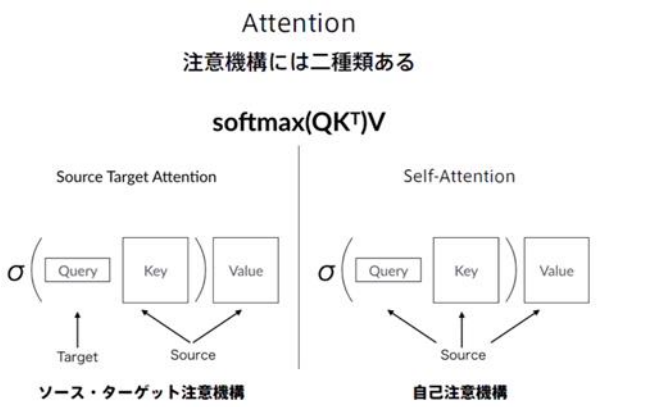
この時 Key と Value は各 key と各 value が一対一対応する key-value ペアの配列，つまり辞書オブジェクトとして機能する．

query と Key の内積は query と各 key の類似度を測り，softmax で正規化した注意の重み (Attention Weight) は query に一致した key の位置を表現する．注意の重みと Value の内積は key の位置に対応する value を加重和として取り出す操作である．





つまり注意（Attention）とは、
・query (検索クエリ) に一致する key を索引し，
・対応する value を取り出す操作
であり，これは辞書オブジェクトの機能と同じである．
例えば一般的な Encoder-Decoder の注意は，エンコーダのすべての隠れ層（情報源）Value から query に関連する隠れ層（情報）value を注意の重みの加重和として取り出すことである．
query の配列 Query が与えられれば，その数だけ key-value ペアの配列から value を取り出す．



注意（Attention）は、「入力がどこから来るのか」によってソースターゲット注意と自己注意に区分される.

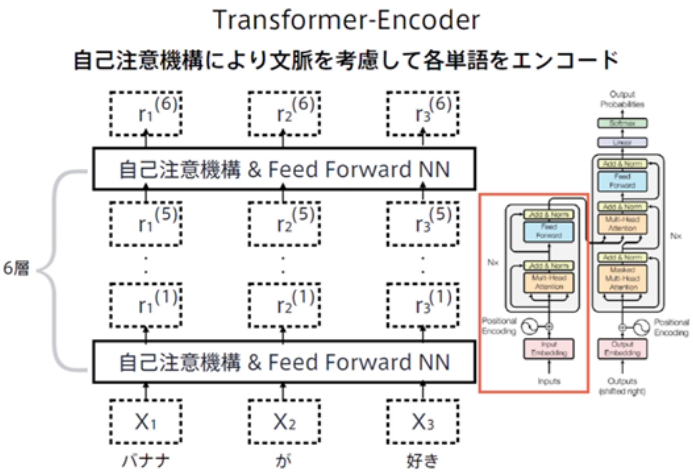
ソースターゲット注意（Source-Target-Attention）では Key と Value はエンコーダの隠れ層（Source）から来て，Query はデコーダの隠れ層（Target）から来る．一般的な Encoder-Decoder の注意はこちらである.

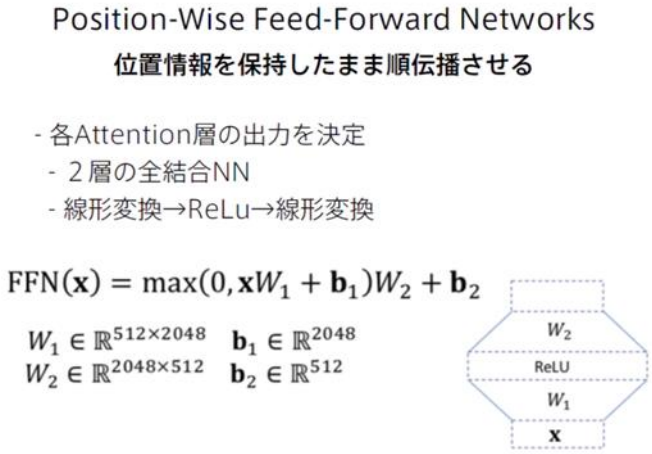
Source は Memory とも呼ばれ,Key と Value は Memory を 2 つに分離したものと解釈できる.

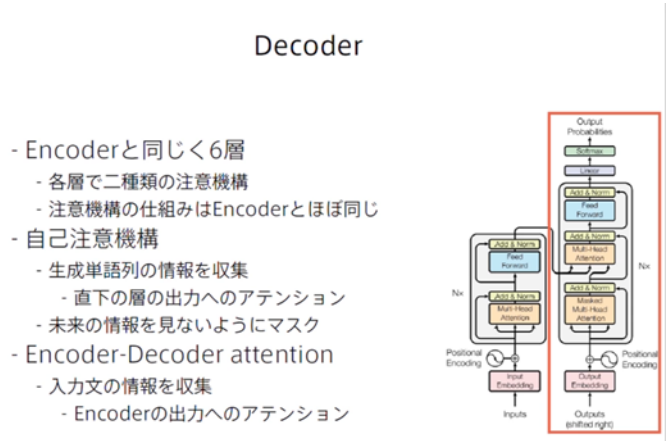
自己注意（Self-Attention）では Query,Key,Value は全て同じ場所（Self）から来る．例えばエンコーダの Query,Key,Value はすべて下の隠れ層から来る.

自己注意はある位置の出力を求めるのに下の隠れ層の全ての位置を参照できる.これは局所的な位置しか参照できない畳み込み層より優れた利点である.

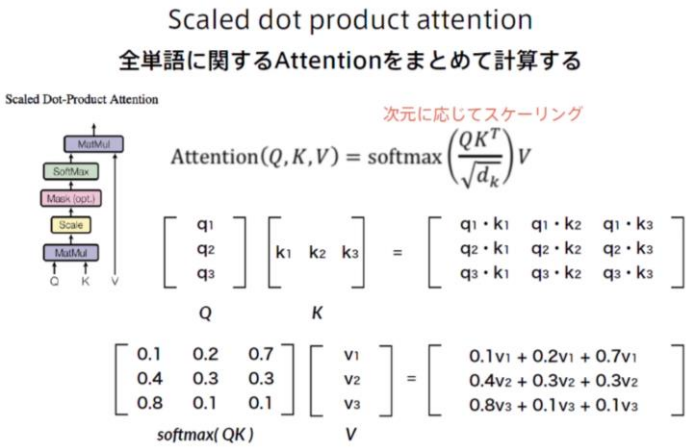
参考サイト：https://deeplearning.hatenablog.com/entry/transformer







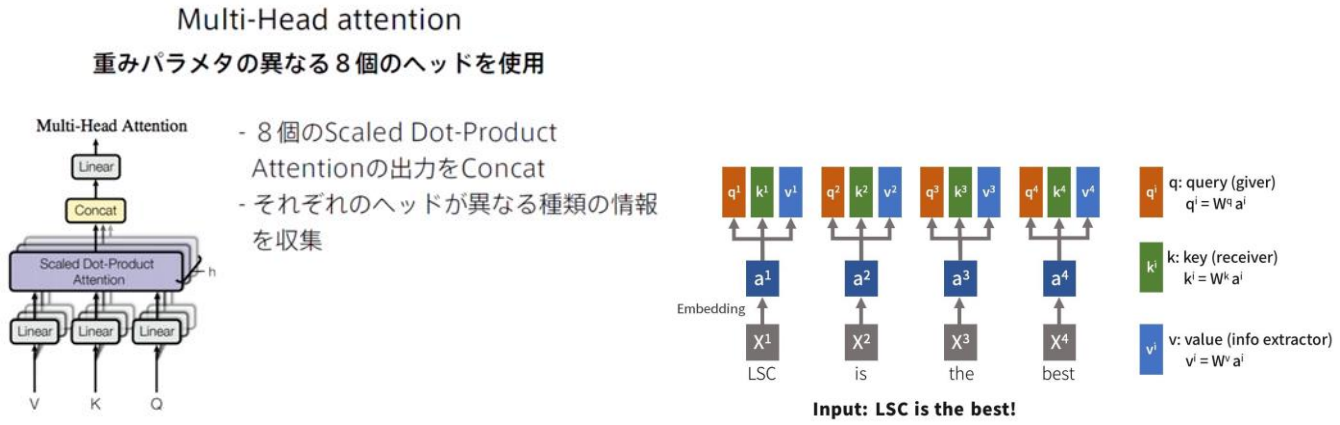
・行列計算方法



・マルチヘッド Attetion

マルチヘッド構造：Attention 機構を複数（マルチ）用意して、それぞれが異なる役割を分担させる。

⇒様々な単語関係の種類や入力文章内単語の間合いで、単語を予測する事が可能。



入力文章 X は埋め込み(Word Embedding)してから単語別に Query, Key と Value の三つの変数を計算します。

$$q^{i}(Query) = W^{q} a^{i}$$
$$k^{i}(Key) = W^{k} a^{i}$$
$$v^{i}(Value) = W^{v} a^{i}$$

W^{q}, W^{k}, W^{v} が層内でトレーニングする変数。

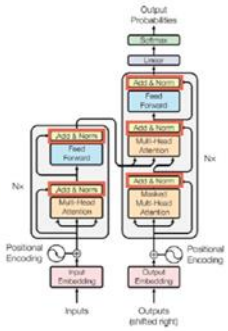
層内の Attention は Query と Key の組み合わせで、Query が「与える方」、Key が

「受ける方」として動作してする。

Value は「情報付与者」として、Attention を使い単語の情報を引き出し予測結果を出す。

・Add＆Norm

- Add (Residual Connection)
  - 入出力の差分を学習させる
  - 実装上は出力に入力をそのまま加算するだけ
  - 効果：学習・テストエラーの低減
- Norm (Layer Normalization)
  - 各層においてバイアスを除く活性化関数への入力を平均0、分散1に正則化
  - 効果：学習の高速化



Position Encoding

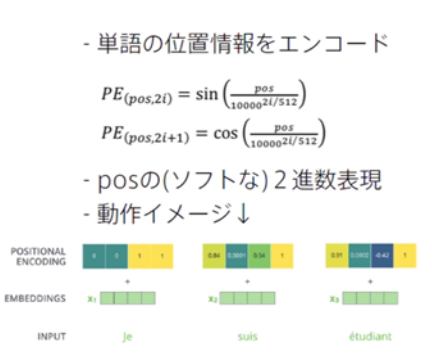
RNNを用いないので単語列の語順情報を追加する必要がある

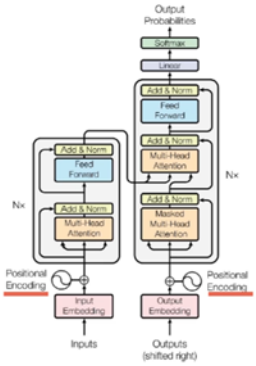
- 単語の位置情報をエンコード

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/512}}\right)$$

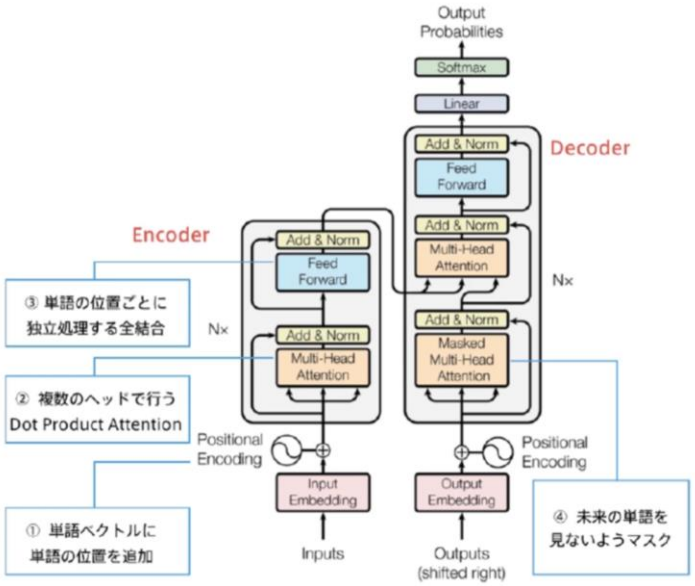
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/512}}\right)$$

- posの(ソフトな)2進数表現
- 動作イメージ↓





◇まとめ



Attentionの可視化

注意状況を確認すると言語構造を捉えていることが多い