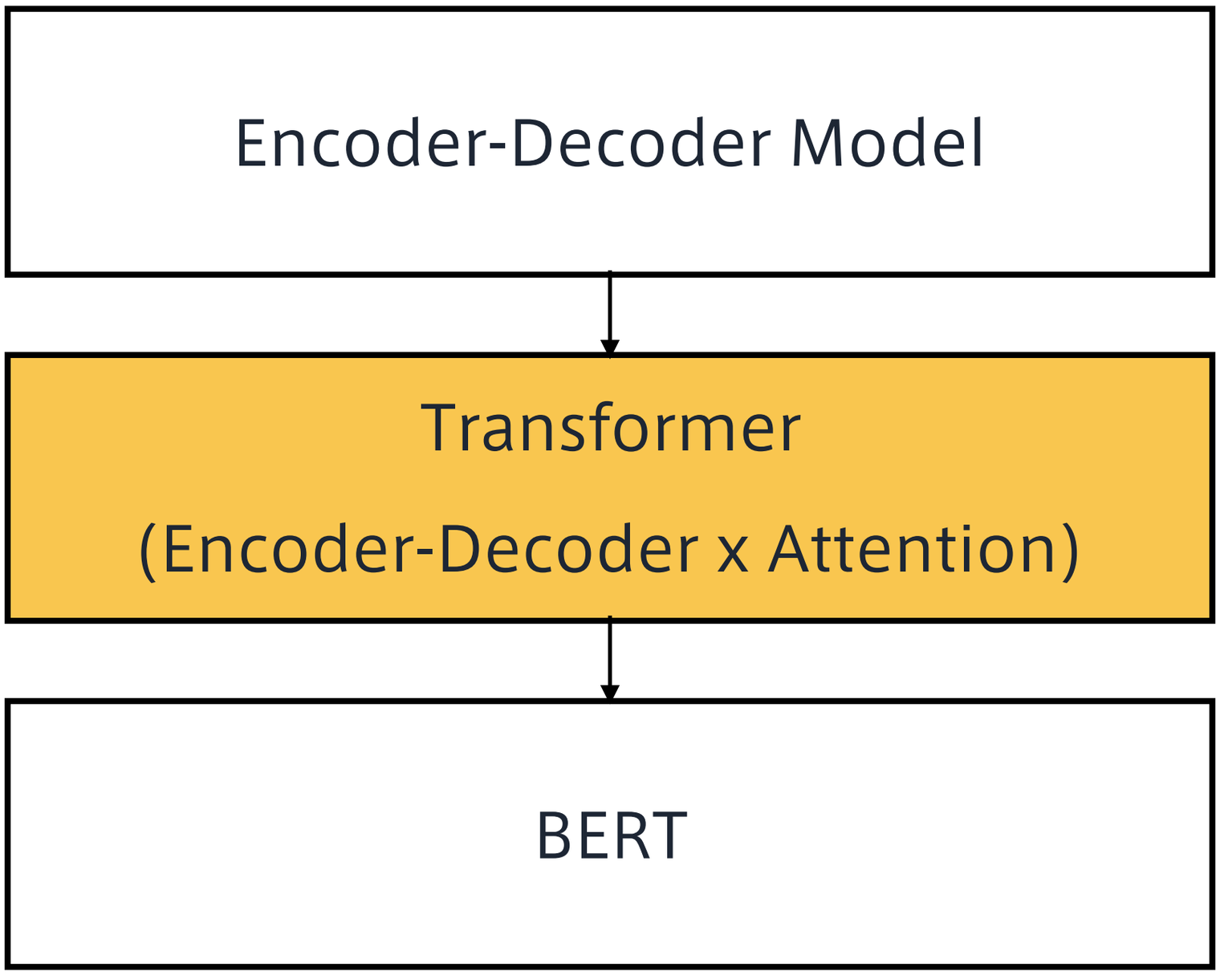
Transformer

Self-Attention(自己注意機構)に焦点を当てて

2019/3/18

BERTまでのロードマップ

BERTを理解するために必要な材料



ニューラル機械翻訳

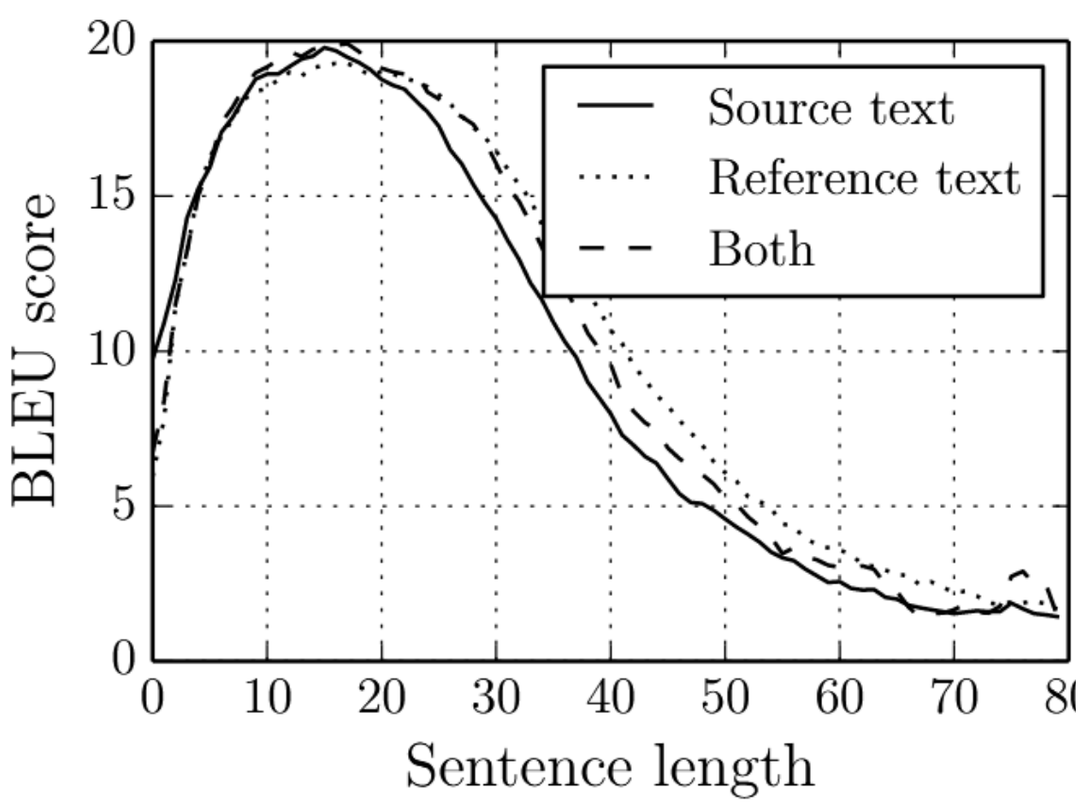
先週実装した内容

- Encoder-Decoder モデル
- Encoder RNN
  - 翻訳元の文を読み込み、実数値ベクトルに変換
- Decoder RNN
  - 実数値ベクトルから、翻訳先の言語の文を生成

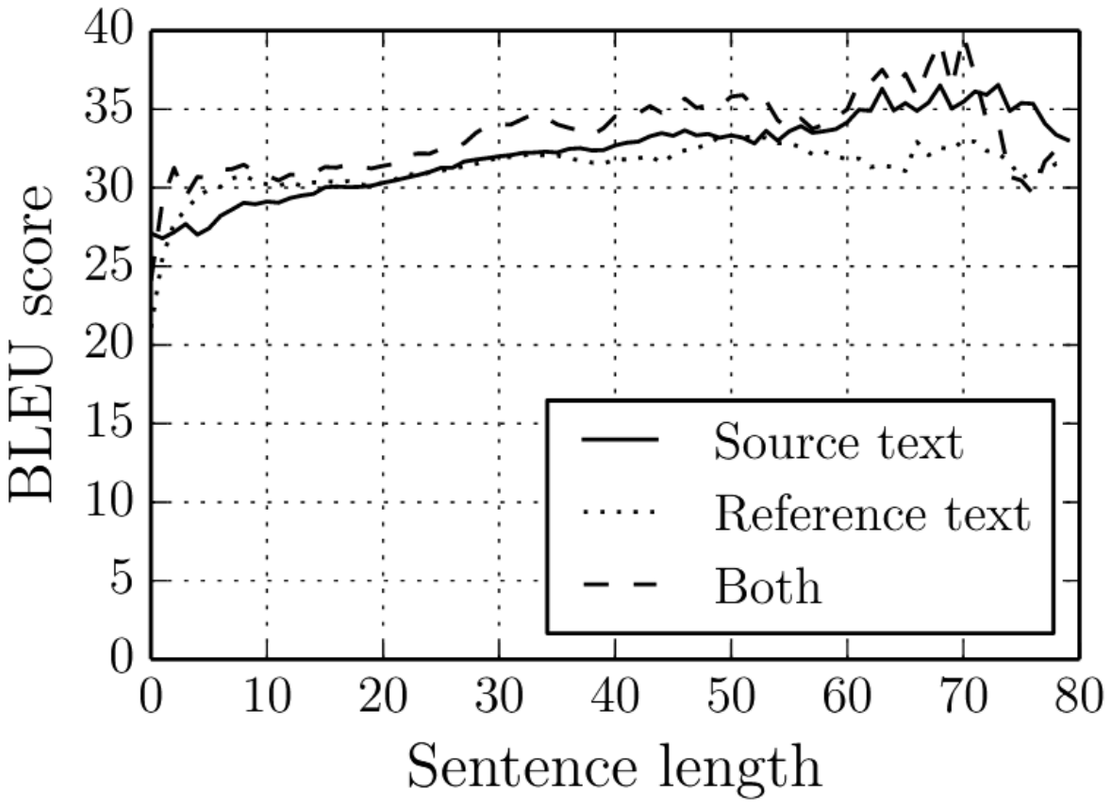
ニューラル機械翻訳の問題点

長さに弱い

- 翻訳元の文の内容をひとつのベクトルで表現
  - 文長が長くなると表現力が足りなくなる
- 文長と翻訳精度の関係性



Encoder-Decoder モデル



統計的機械翻訳モデル

Attention (注意機構) (Bahdanau et al., 2015)

サブタイトル

- 翻訳先の各単語を選択する際に、翻訳元の文中の各単語の隠れ状態を利用

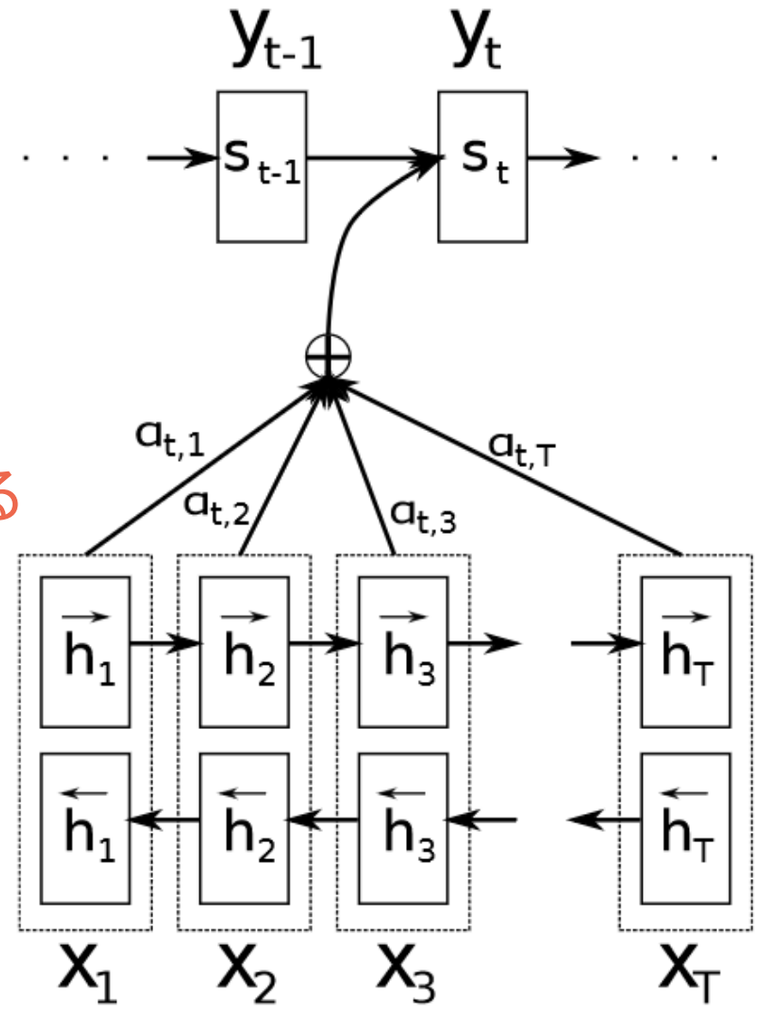
翻訳元の各単語の隠れ状態の加重平均

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

重み(全て足すと１) 重みはFFNNで求める

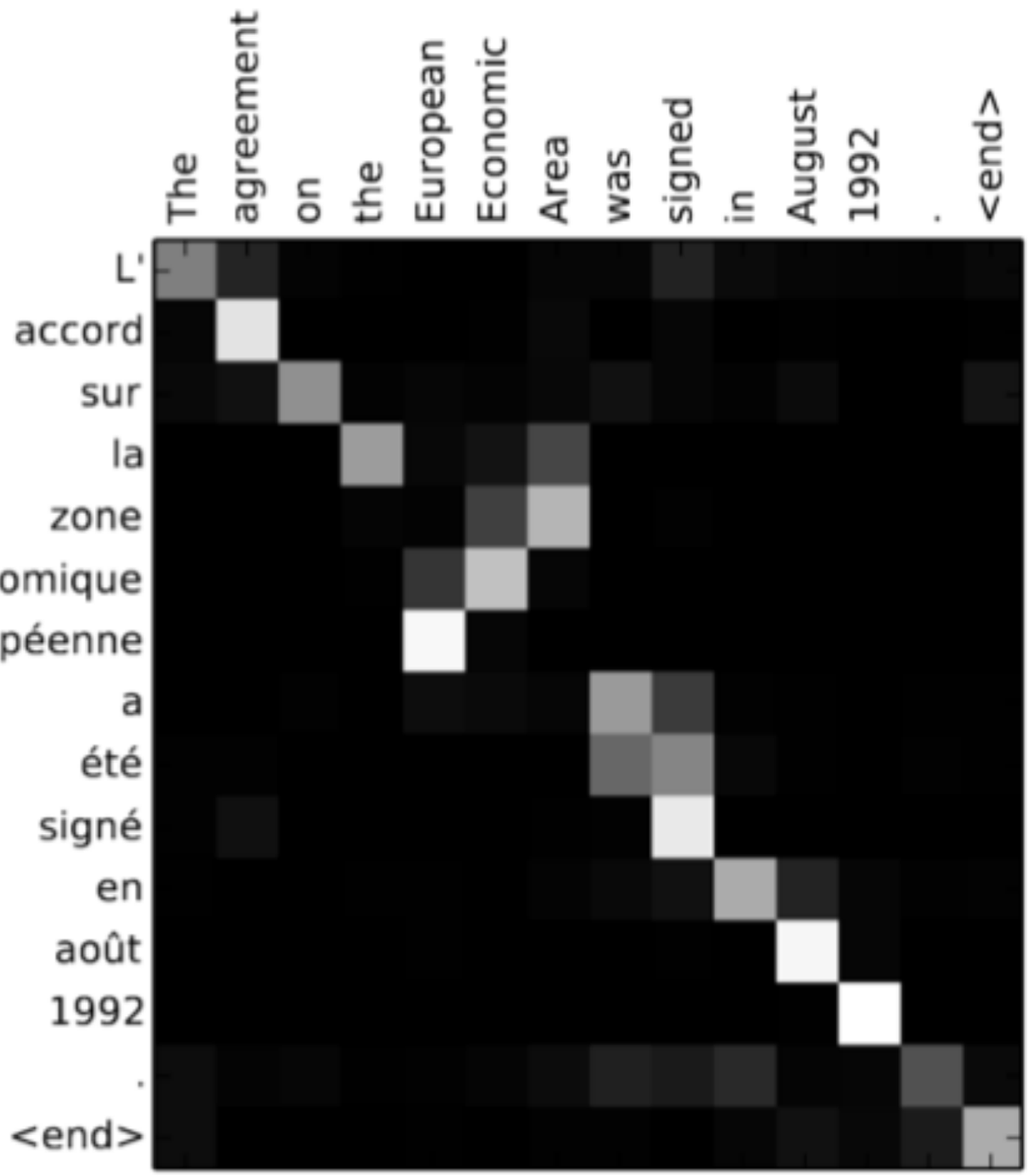
$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)},$$

$$e_{ij} = a(s_{i-1}, h_j)$$



Attentionの例

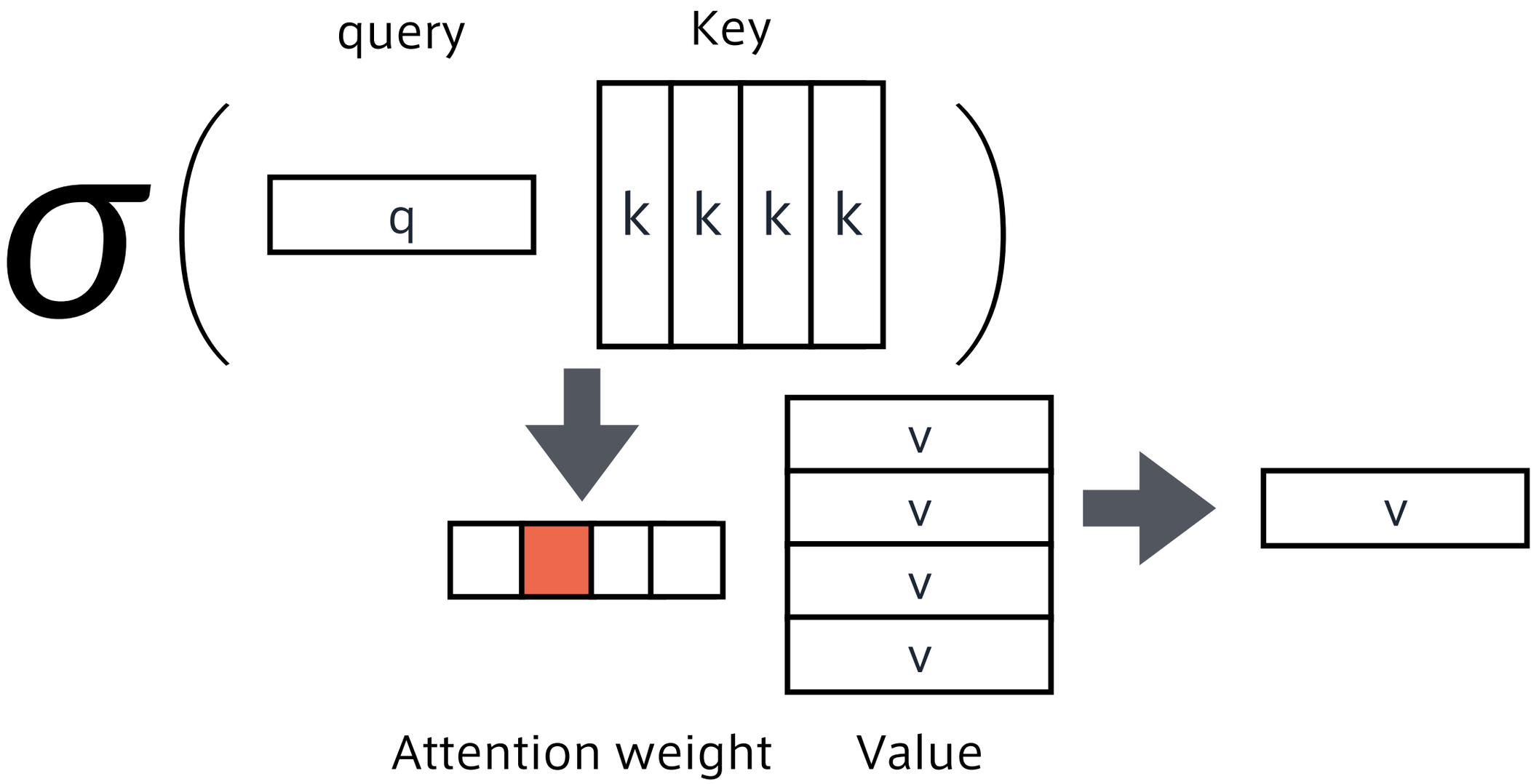
英→仏



Attentionは何をしているのか

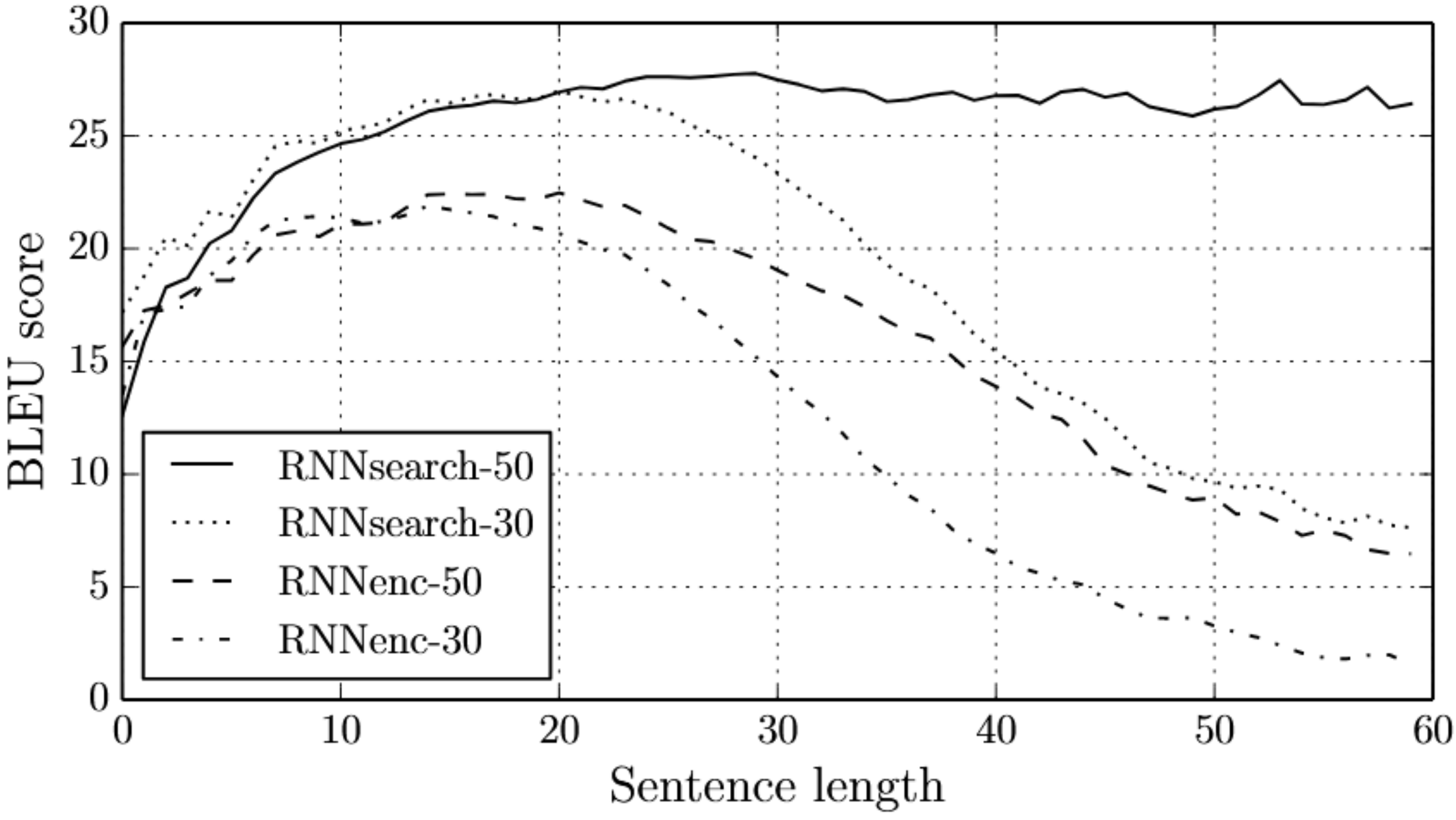
Attentionは辞書オブジェクト

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ると見做すことができる。これは辞書オブジェクトの機能と同じである。



文長と翻訳精度

文長が長くなっても翻訳精度が落ちないことが確認できる

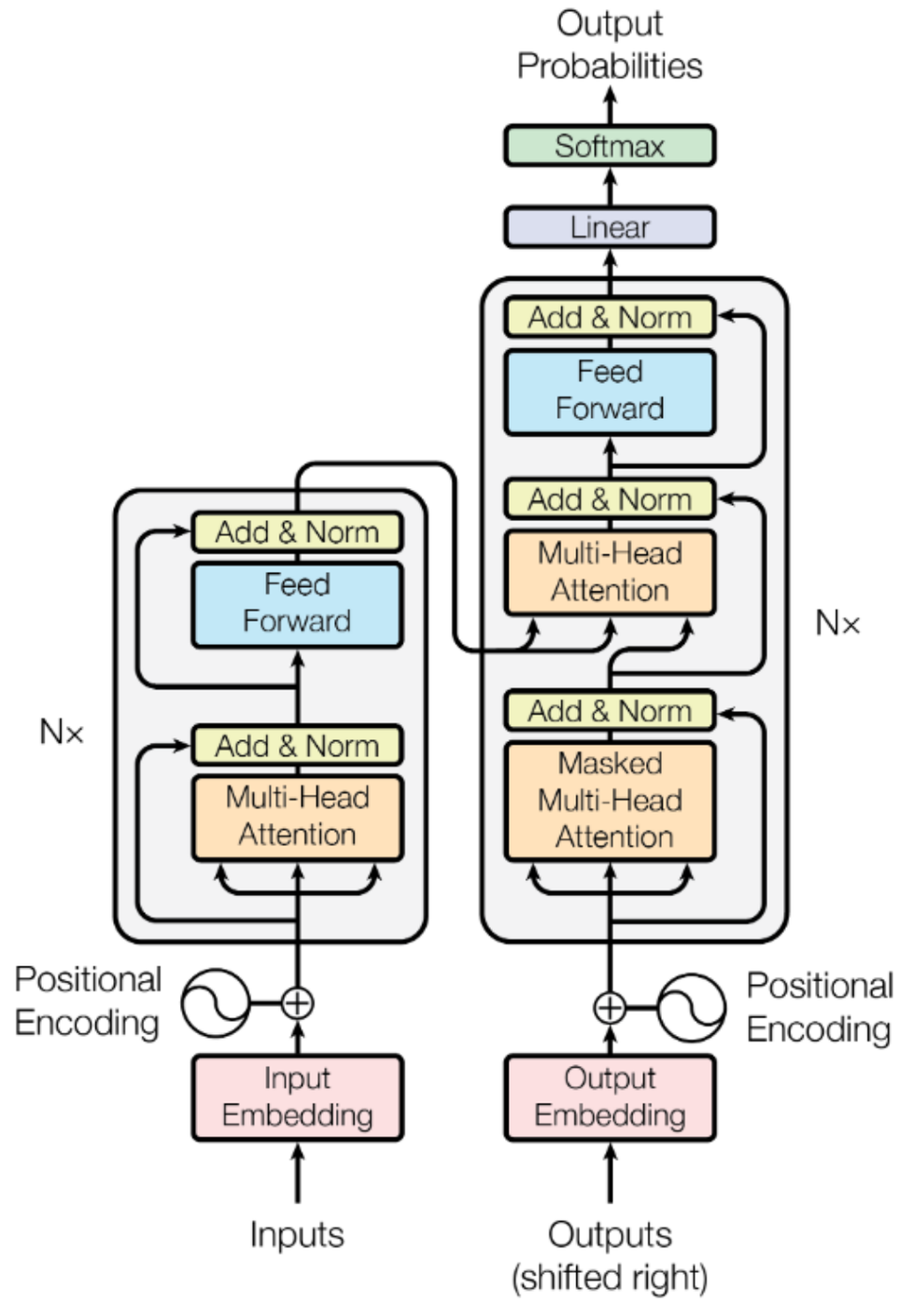


(Bahdanau et al., 2015)

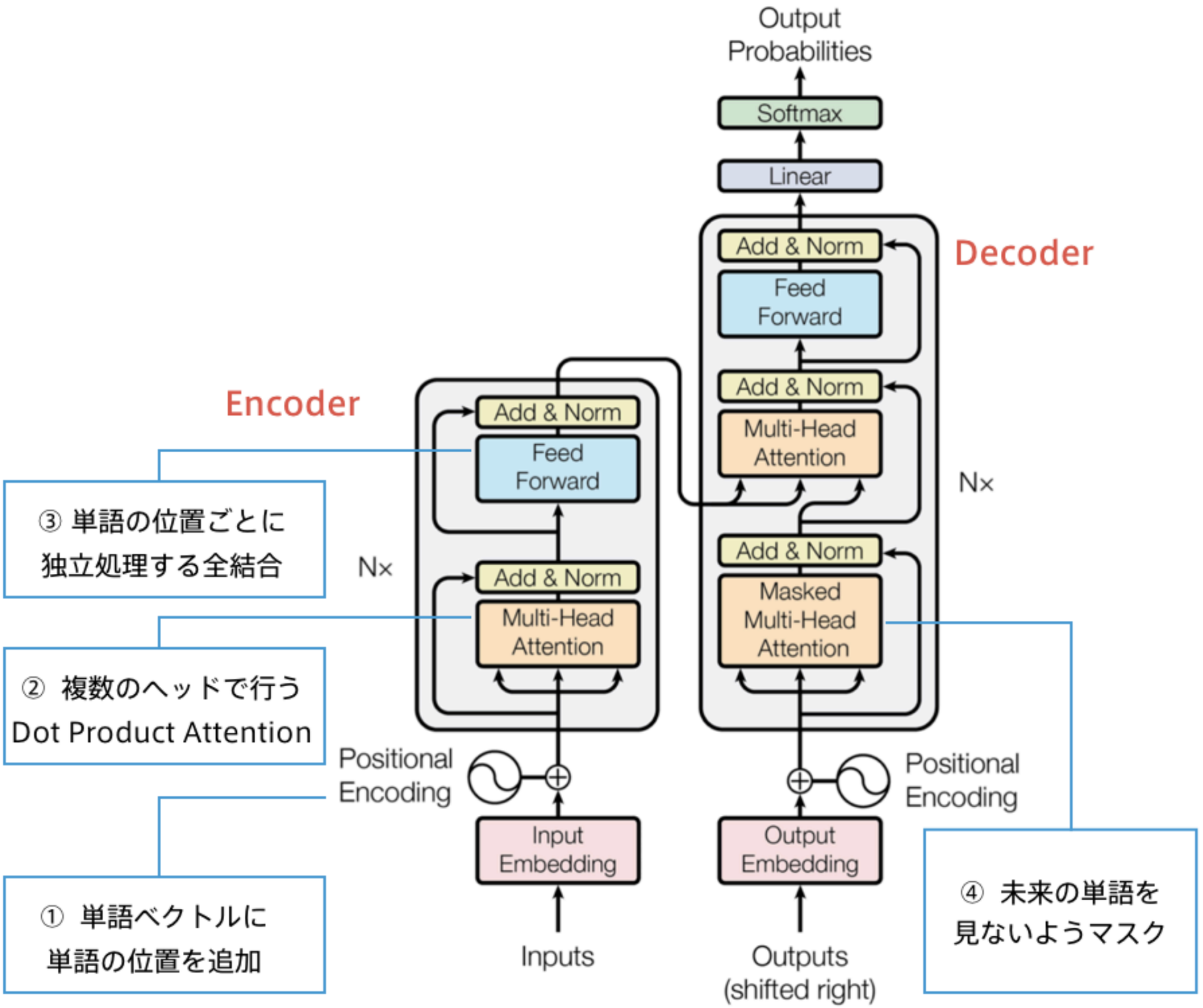
Transformer (Vaswani et al., 2017)

Attention is all you need

- 2017年6月に登場
- RNNを使わない
  - 必要なのはAttentionだけ
- 当時のSOTAをはるかに少ない計算量で実現
- 英仏 (3600万文) の学習を8GPUで3.5日で完了



Transformer主要モジュール

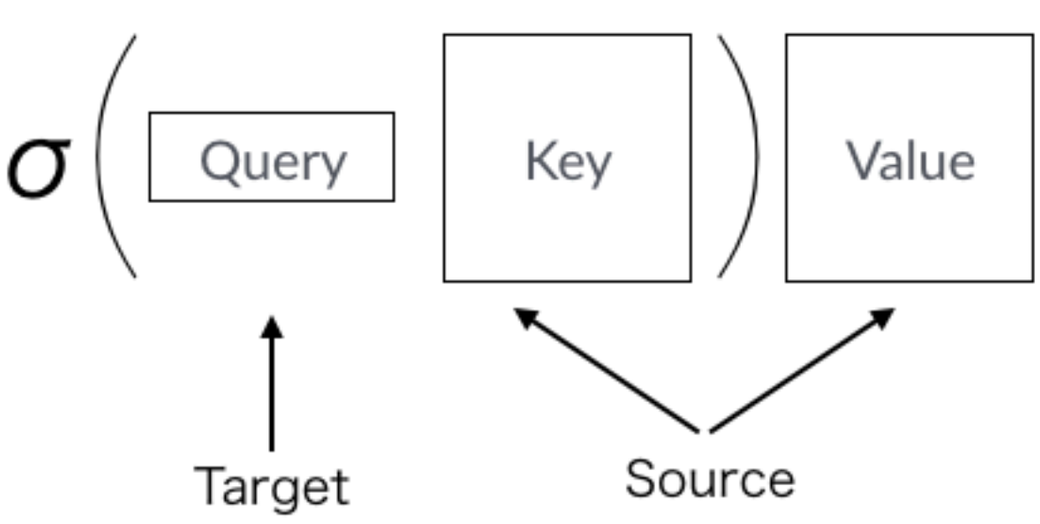


Attention

注意機構には二種類ある

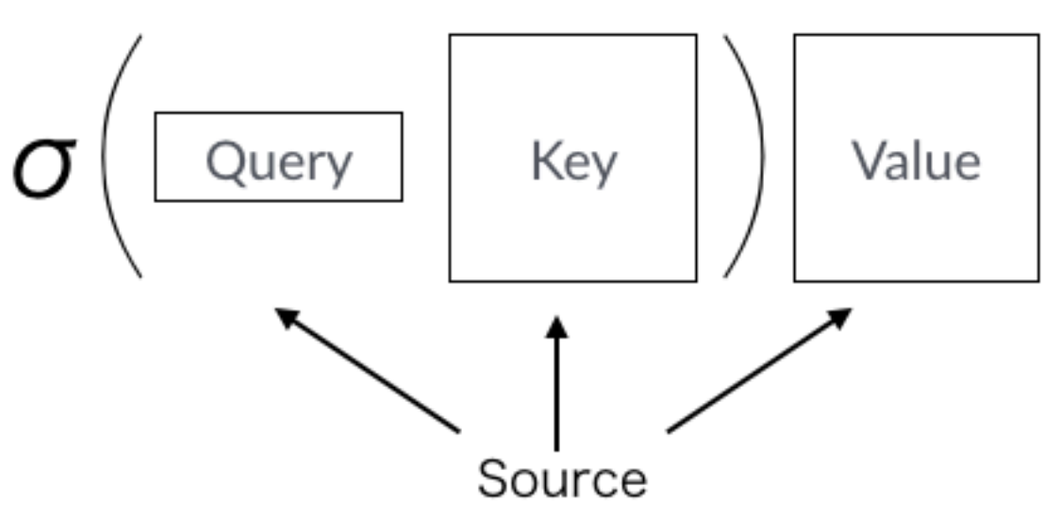
$$\mathbf{softmax(QK^T)V}$$

Source Target Attention



ソース・ターゲット注意機構

Self-Attention

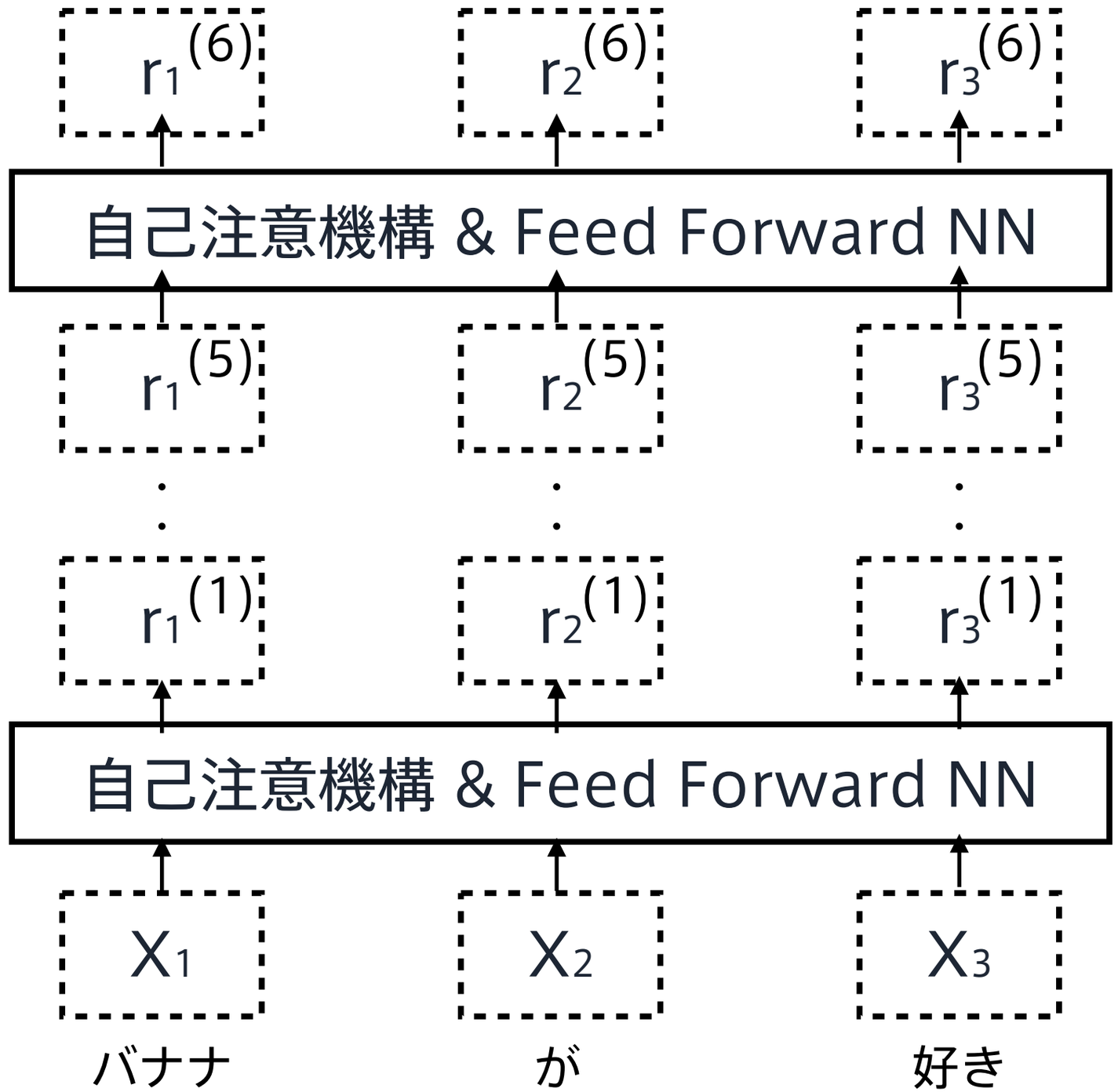


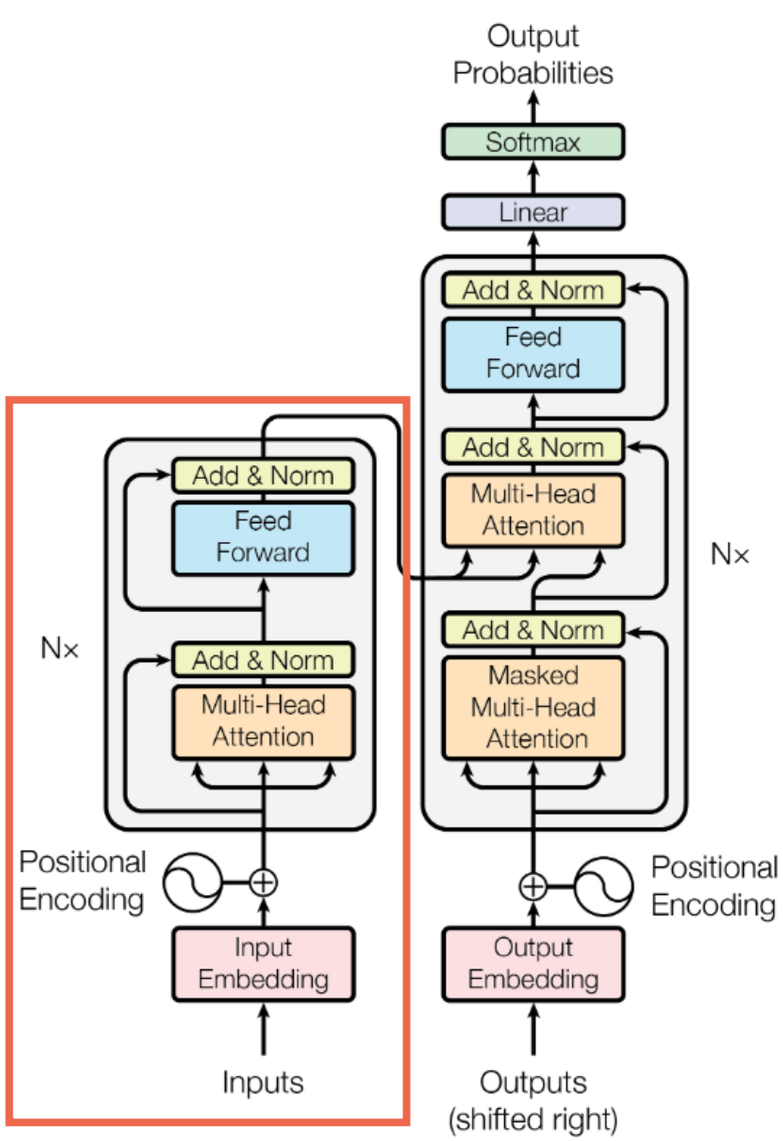
自己注意機構

Transformer-Encoder

自己注意機構により文脈を考慮して各単語をエンコード

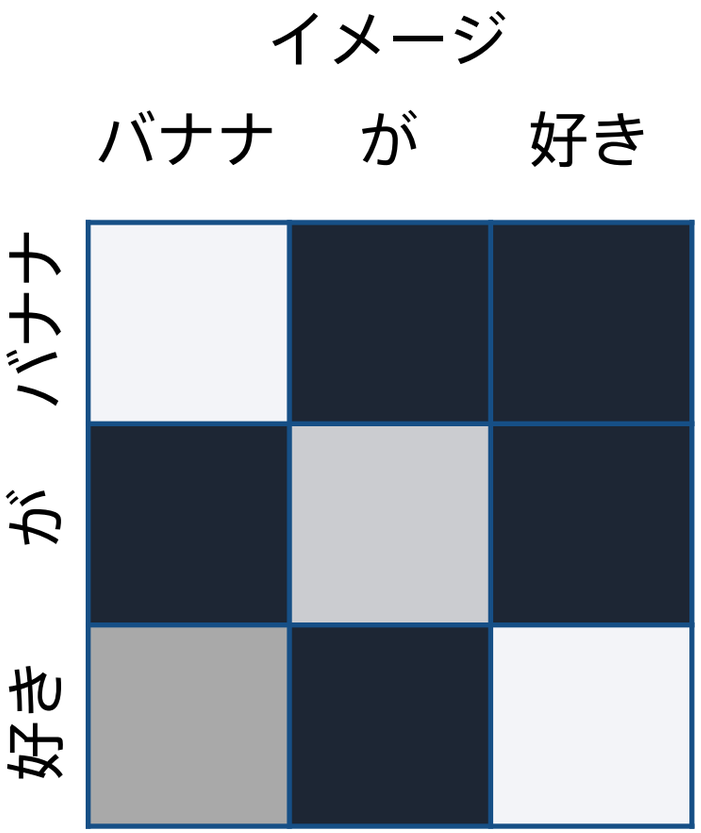
6層

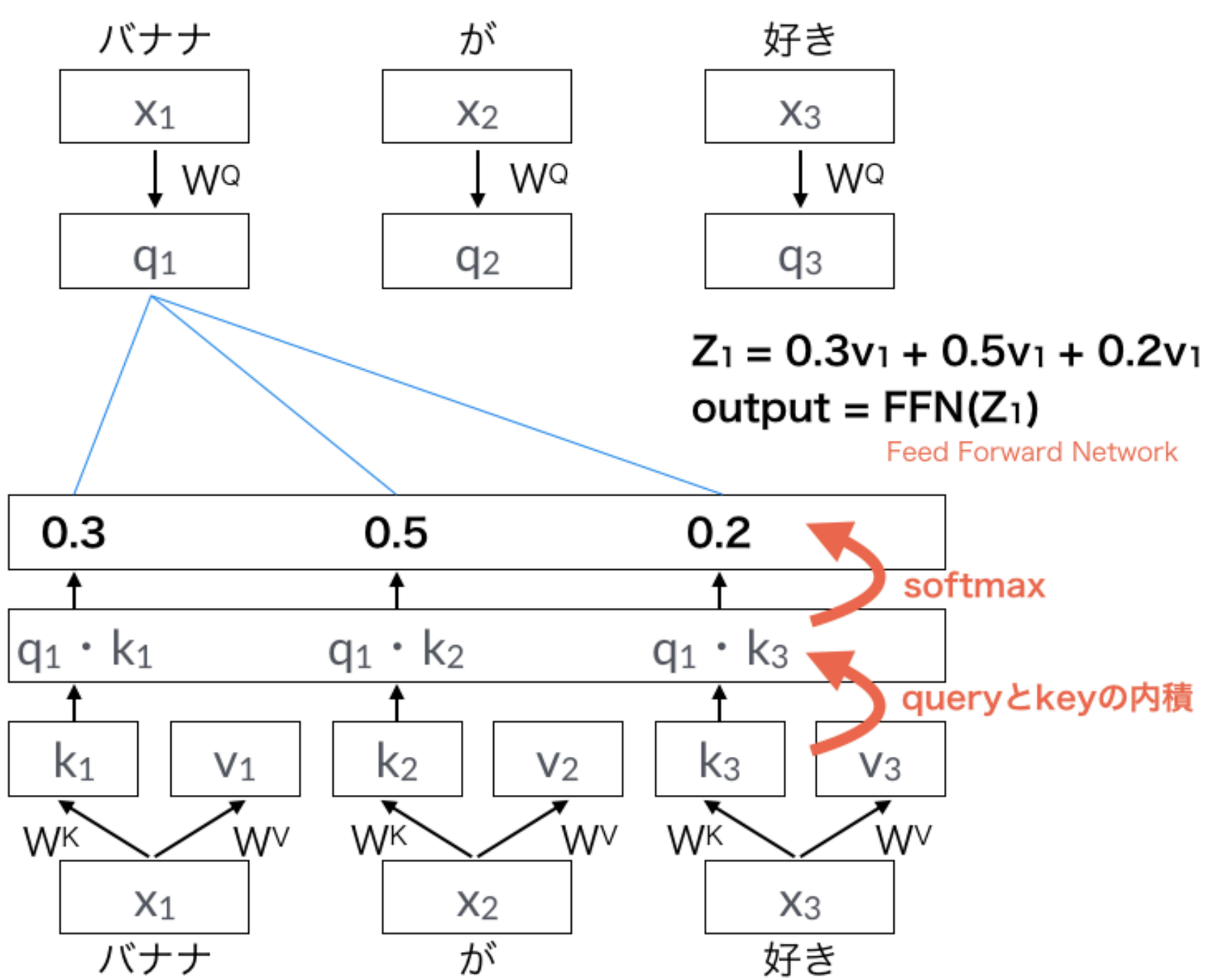




Self-Attentionが肝

入力を全て同じにして学習的に注意箇所を決めていく





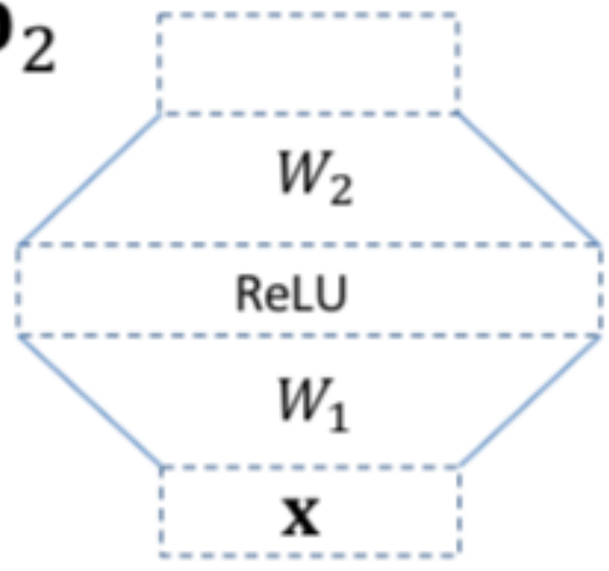
Position-Wise Feed-Forward Networks

位置情報を保持したまま順伝播させる

- 各Attention層の出力を決定
  - ２層の全結合NN
  - 線形変換→ReLu→線形変換

$$\mathrm{FFN}(\mathbf{x}) = \max(0, \mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$$

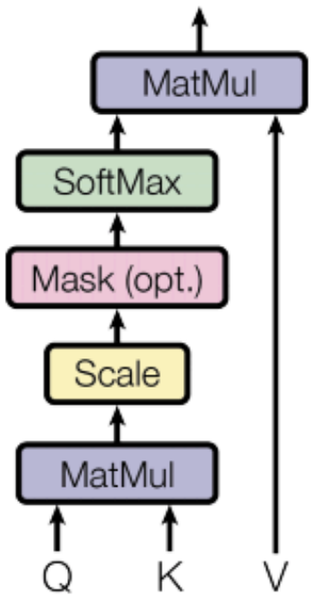
$$W_1 \in \mathbb{R}^{512\times2048} \quad \mathbf{b}_1 \in \mathbb{R}^{2048}$$
$$W_2 \in \mathbb{R}^{2048\times512} \quad \mathbf{b}_2 \in \mathbb{R}^{512}$$



Scaled dot product attention

全単語に関するAttentionをまとめて計算する

Scaled Dot-Product Attention



次元に応じてスケーリング

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

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix} = \begin{bmatrix} q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 \\ q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 \end{bmatrix}$$

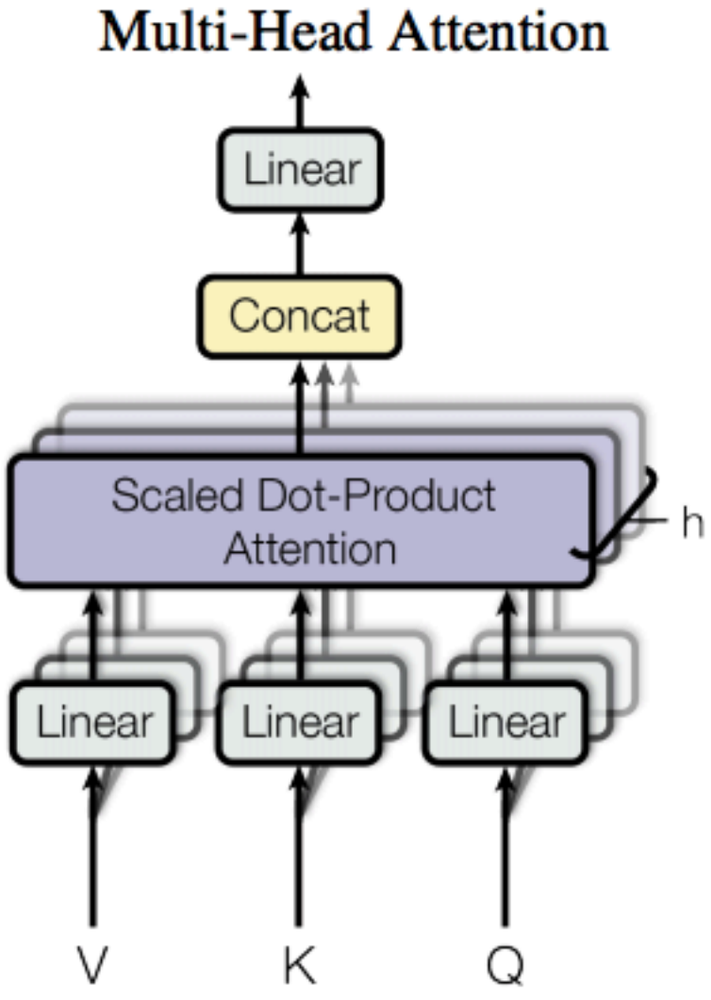
Q K

$$\begin{bmatrix} 0.1 & 0.2 & 0.7 \\ 0.4 & 0.3 & 0.3 \\ 0.8 & 0.1 & 0.1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0.1v_1 + 0.2v_1 + 0.7v_1 \\ 0.4v_2 + 0.3v_2 + 0.3v_2 \\ 0.8v_3 + 0.1v_3 + 0.1v_3 \end{bmatrix}$$

$softmax(QK)$ V

Multi-Head attention

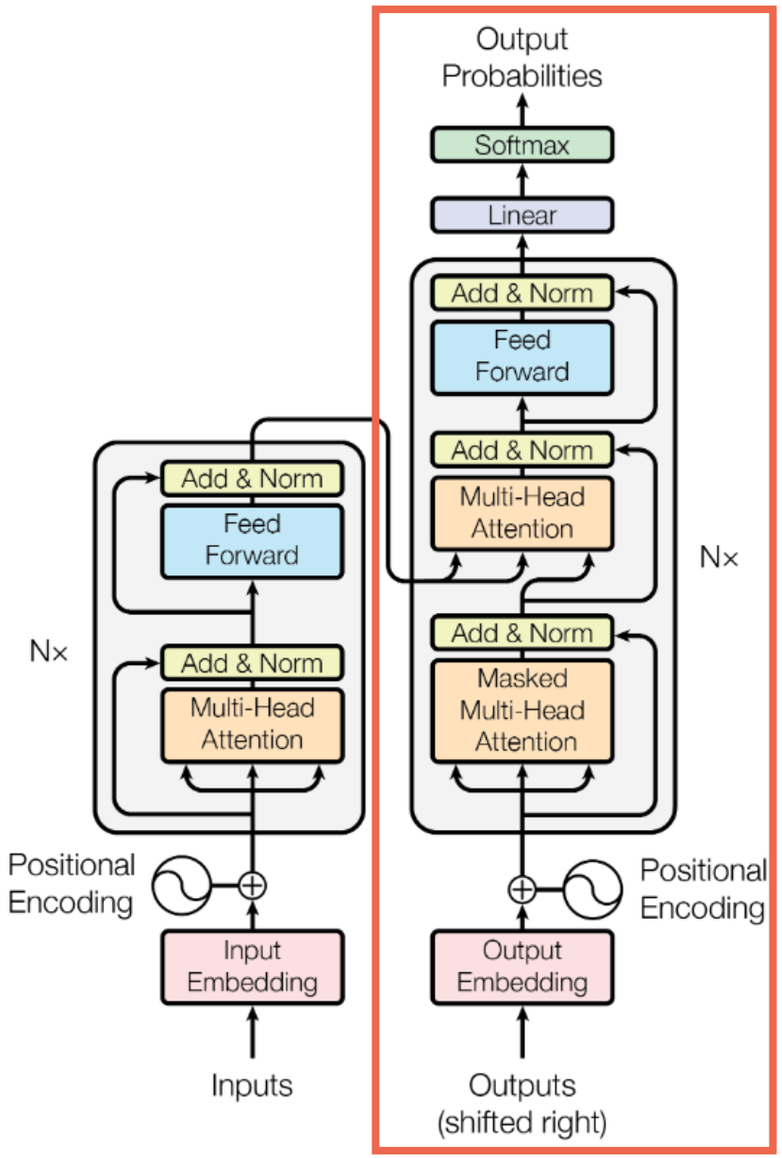
重みパラメタの異なる８個のヘッドを使用



- ８個のScaled Dot-Product Attentionの出力をConcat
- それぞれのヘッドが異なる種類の情報を収集

Decoder

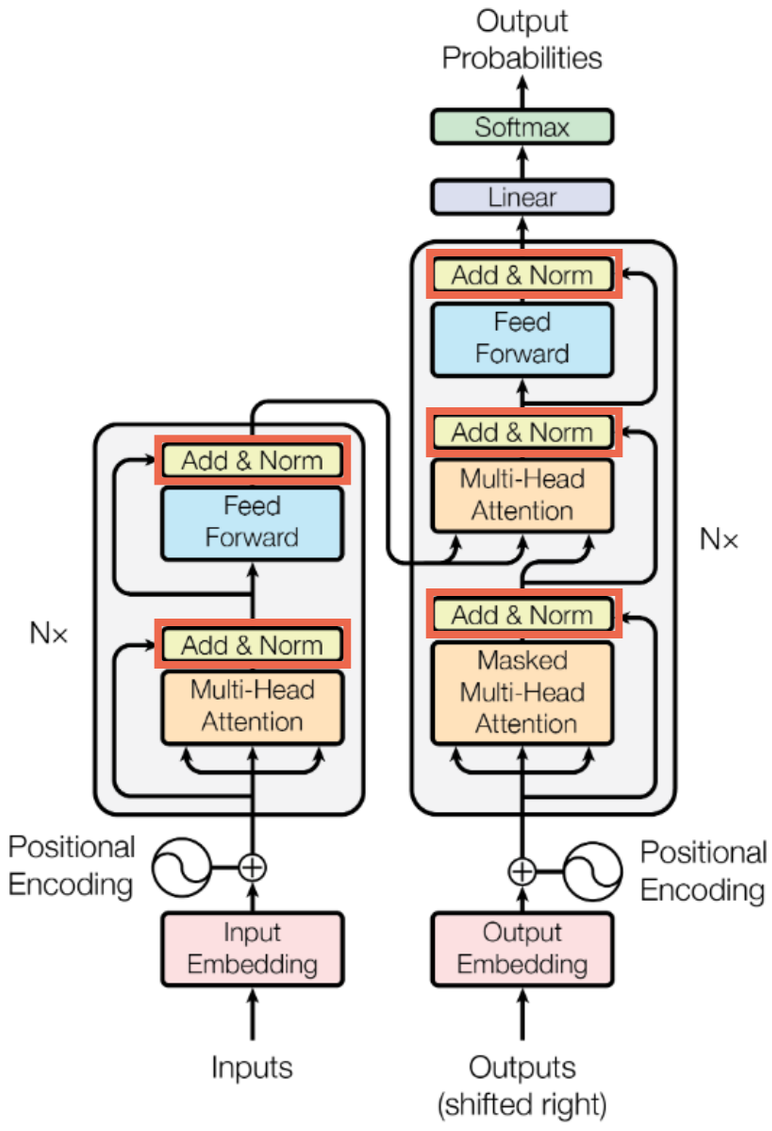
- Encoderと同じく6層
  - 各層で二種類の注意機構
  - 注意機構の仕組みはEncoderとほぼ同じ
- 自己注意機構
  - 生成単語列の情報を収集
    - 直下の層の出力へのアテンション
  - 未来の情報を見ないようにマスク
- Encoder-Decoder attention
  - 入力文の情報を収集
    - Encoderの出力へのアテンション



Add & Norm

サブタイトル

- 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

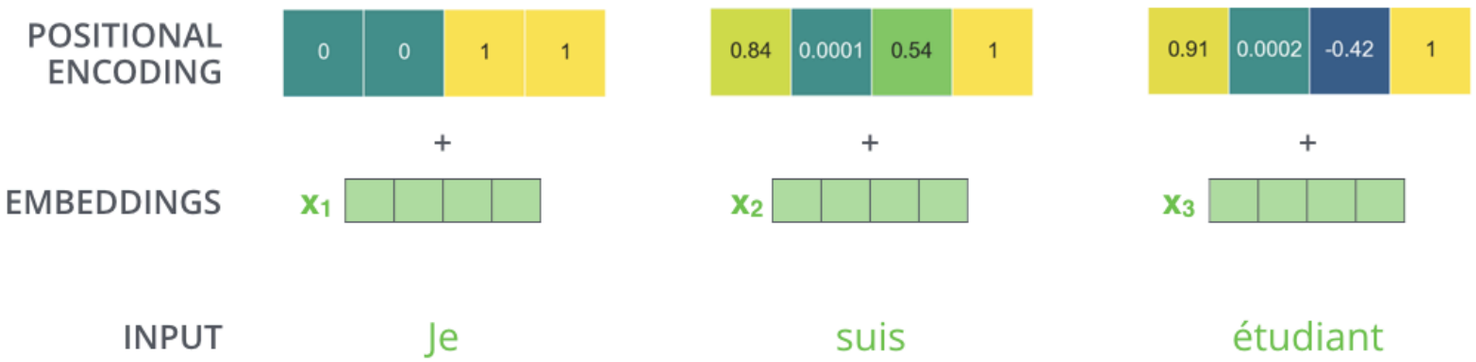
RNNを用いないので単語列の語順情報を追加する必要がある

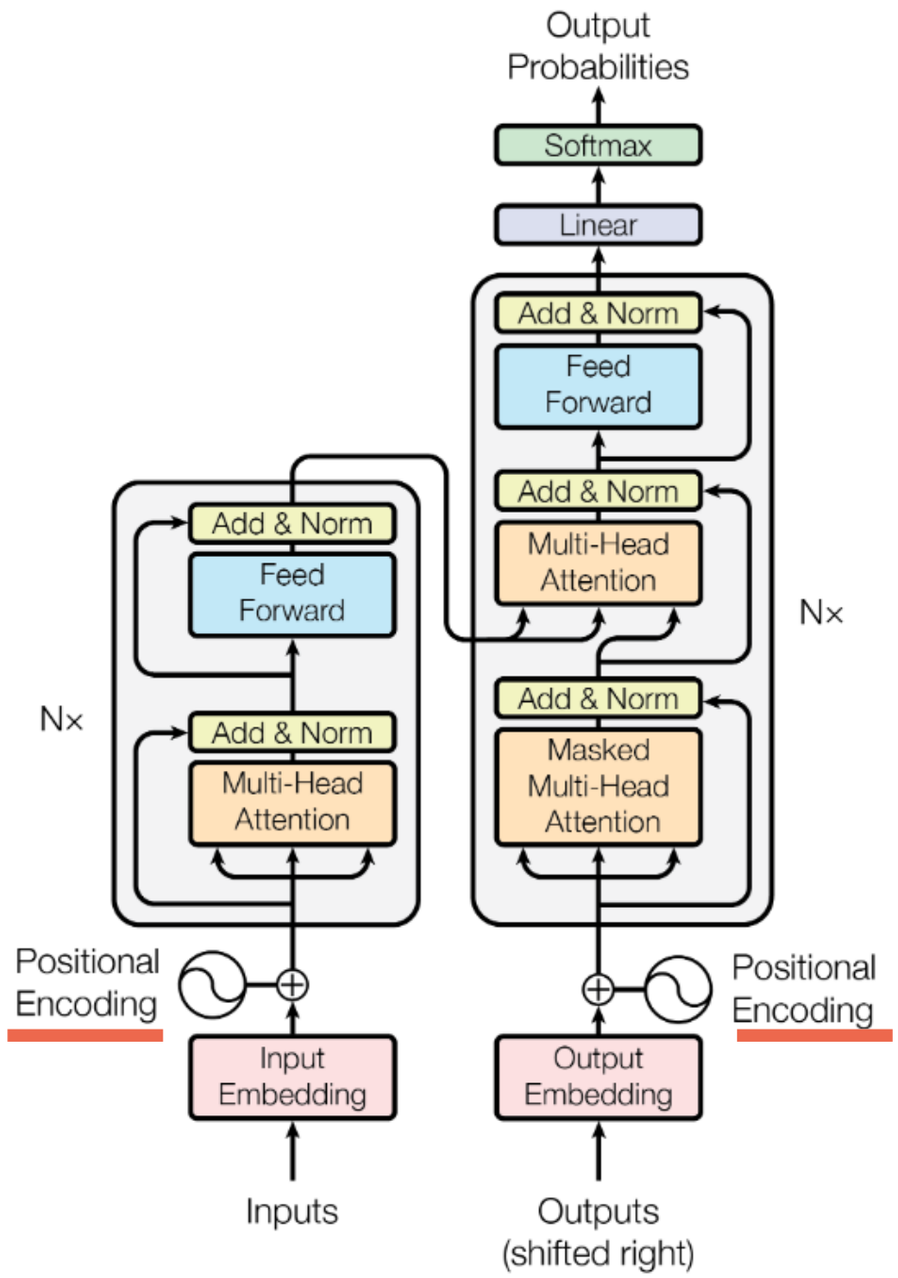
- 単語の位置情報をエンコード

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

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

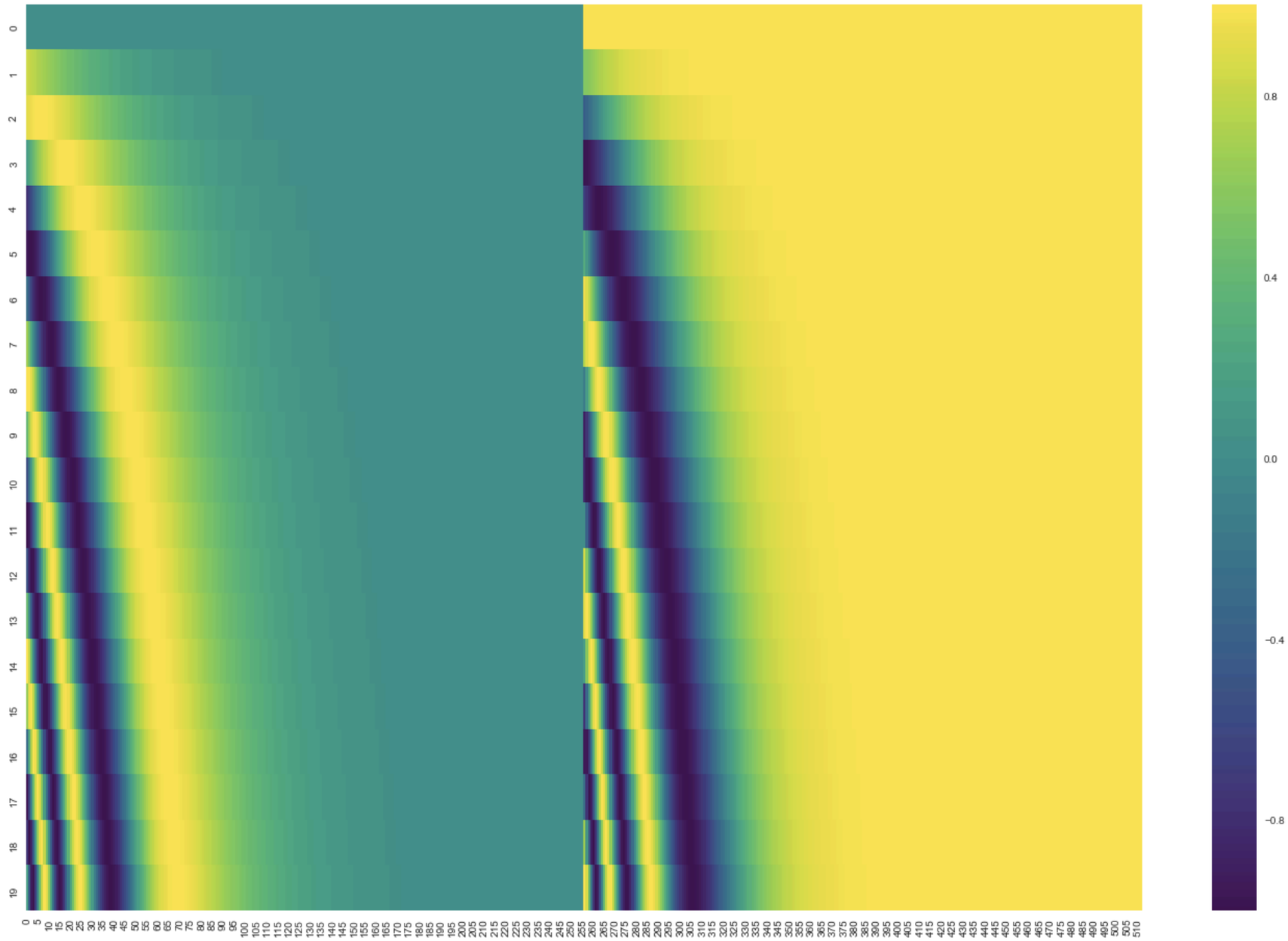
- posの(ソフトな)２進数表現
- 動作イメージ↓





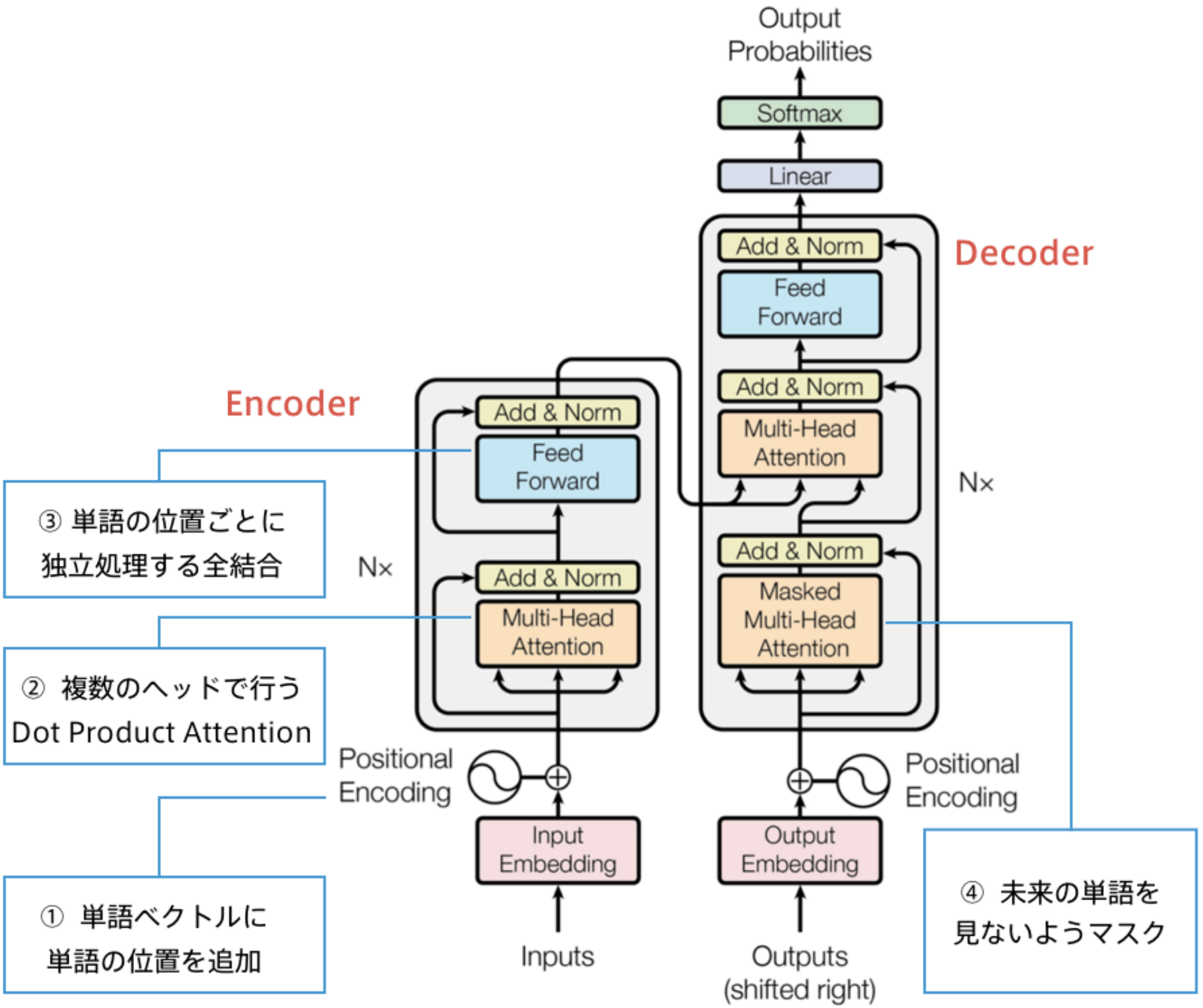
Position Encoding

縦軸が単語の位置、横軸が成分の次元



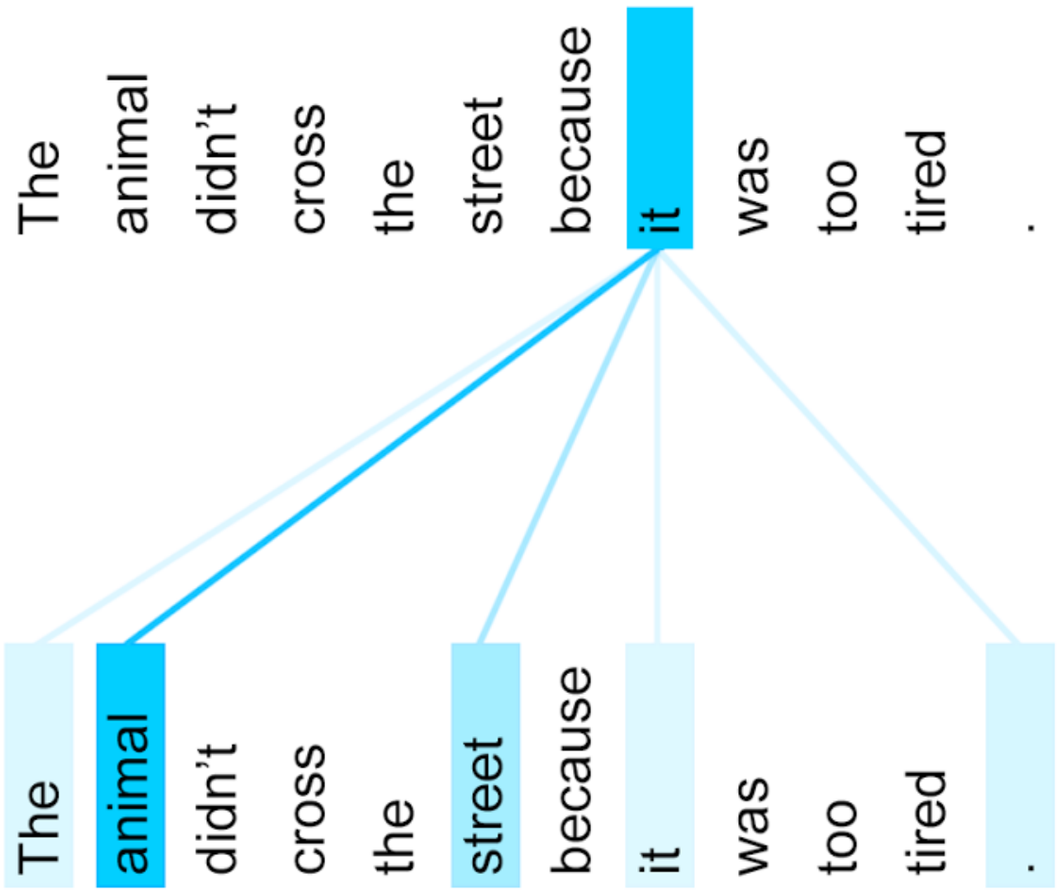
左半分はsinによる生成、左半分はcosによる生成→concatされる

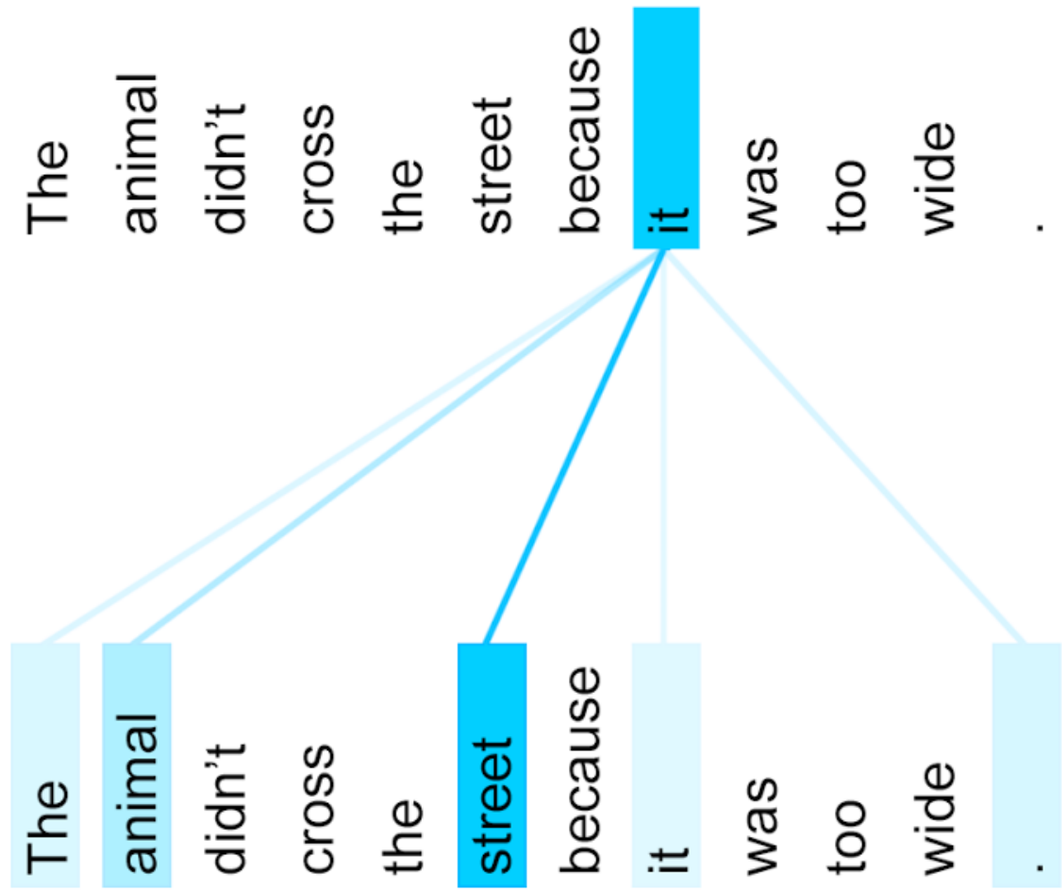
まとめ


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Decoder
Nx
Encoder
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
③ 単語の位置ごとに独立処理する全結合
② 複数のヘッドで行うDot Product Attention
① 単語ベクトルに単語の位置を追加
④ 未来の単語を見ないようマスク

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





</introduction><implementation>