Deep Learning基礎講座

Natural Language Processing

Day 3

講義シラバス

Day 1:　前処理、Word2Vec

Day 2:　Encoder-Decoderモデル

Day 3:　Transformerモデル

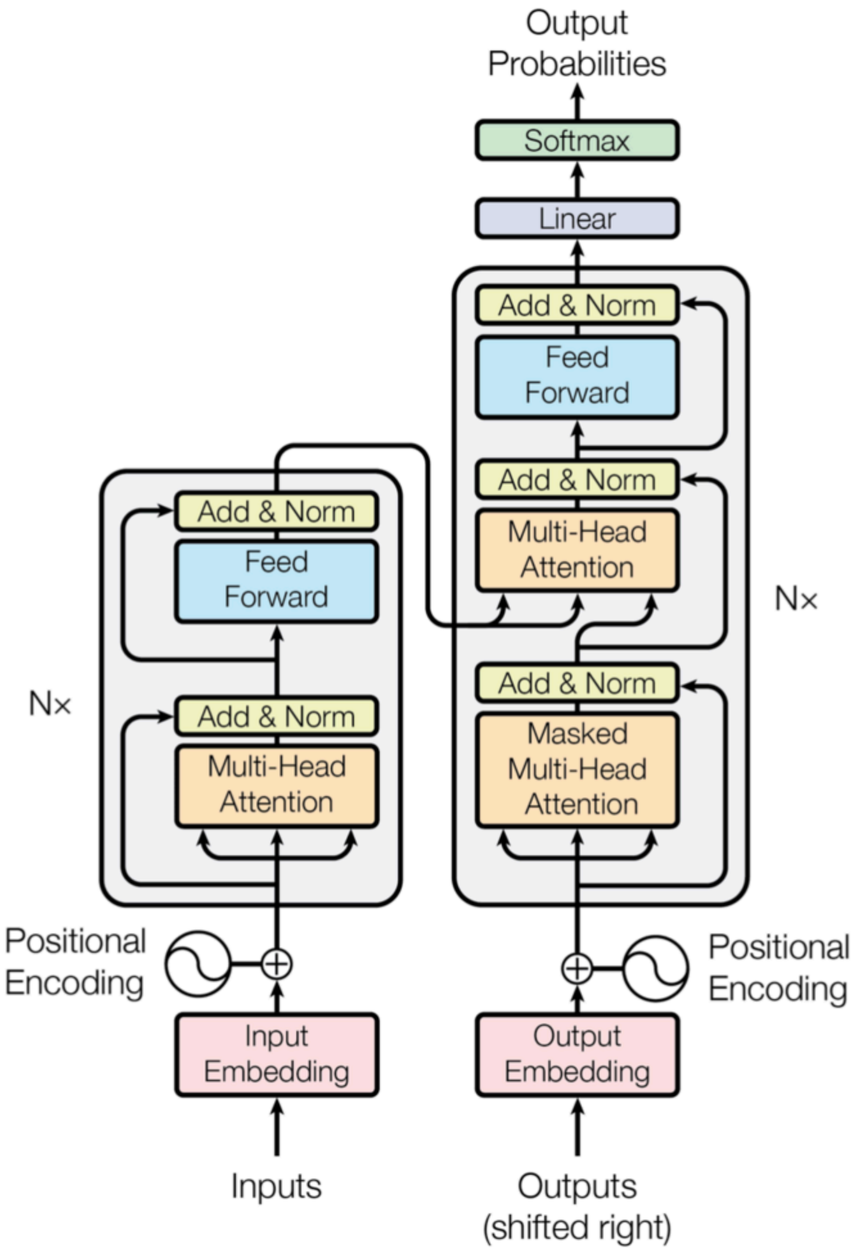
Transformerモデル

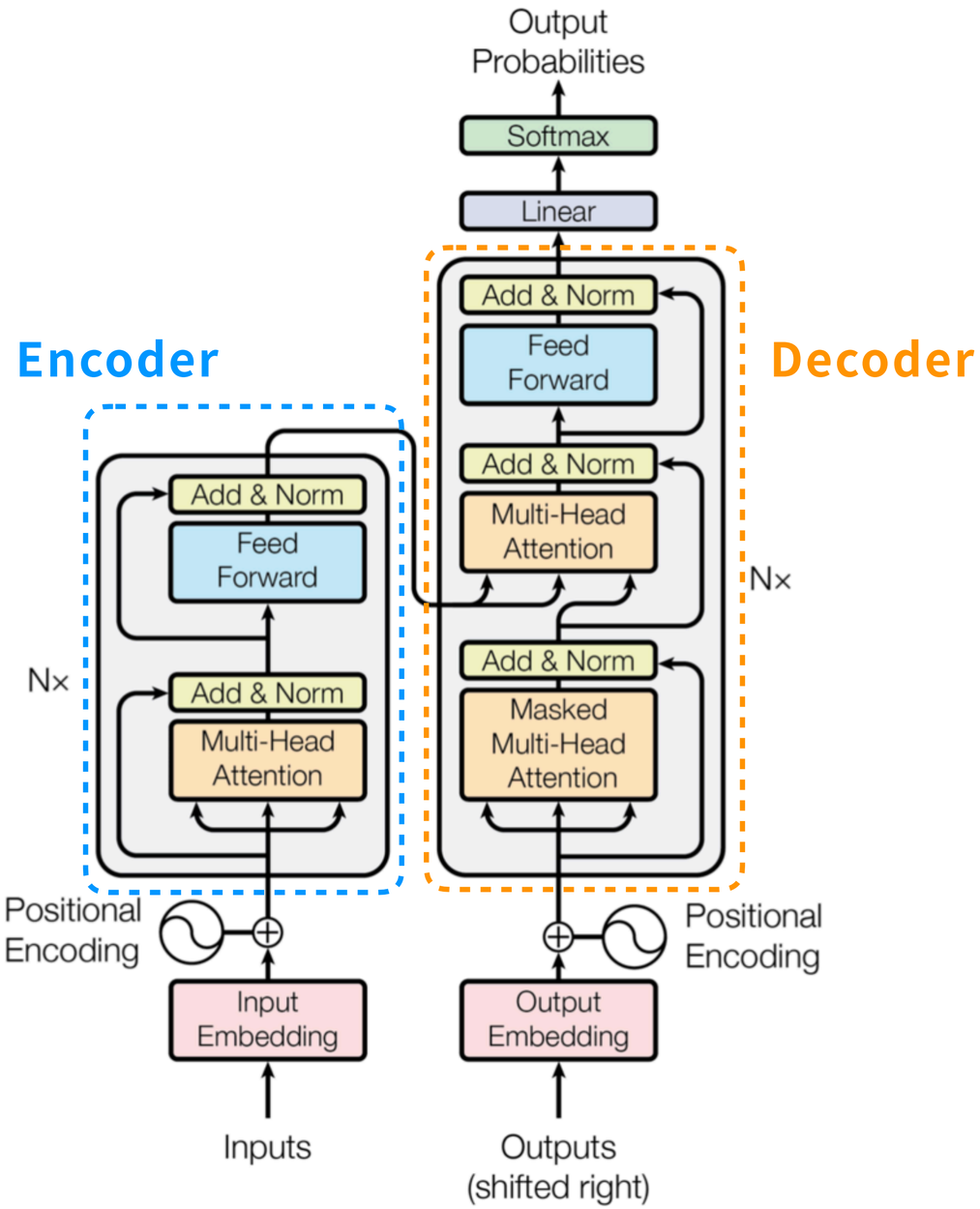
- AttentionベースのEncoder-Decoderモデル

- NLPにおけるひとつのブレイクスルーとなった

- **Attention Is All You Need**

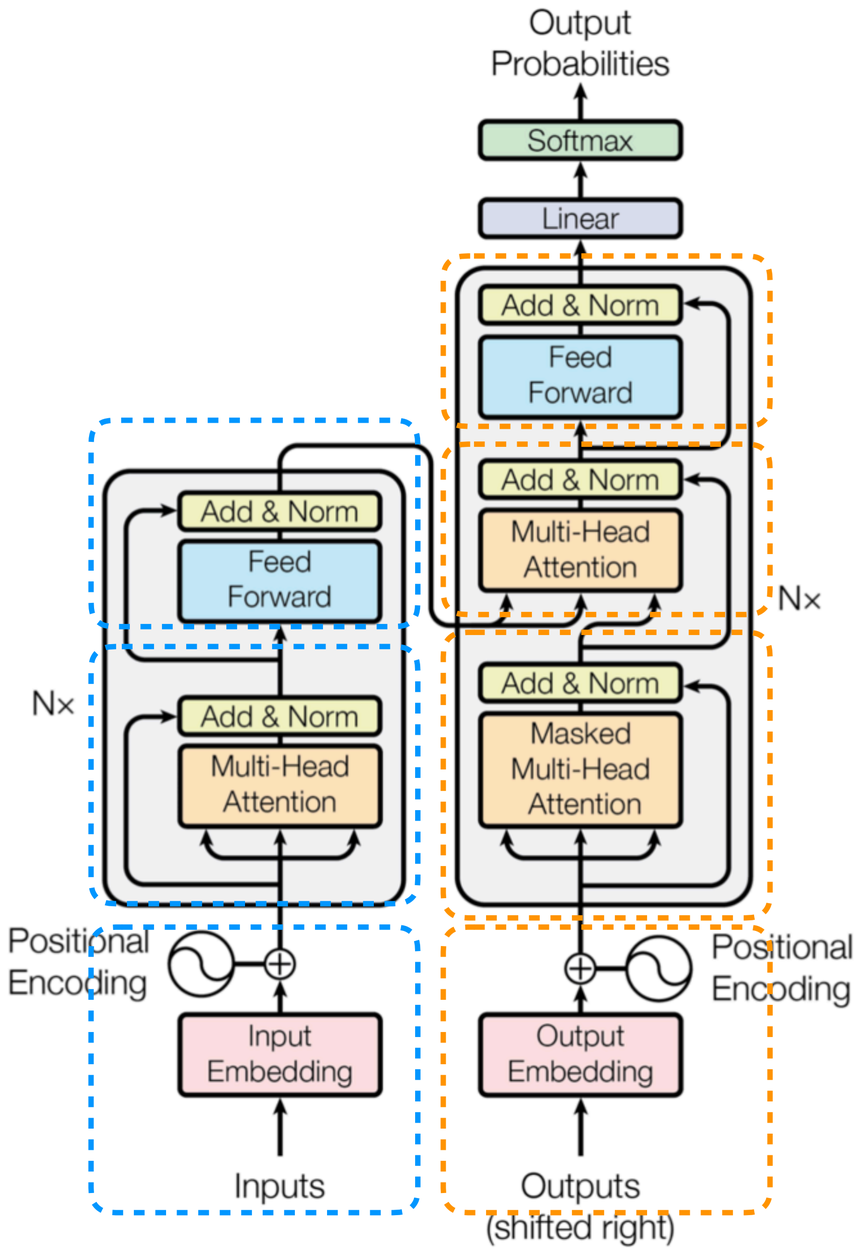
  https://arxiv.org/abs/1706.03762


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


Output Probabilities
Softmax
Linear
Encoder
Decoder
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Transformerの構造分解

- Positional Encoding
- Multi-Head Attention
- Add & Norm
- Position-Wise Feed Forward Network



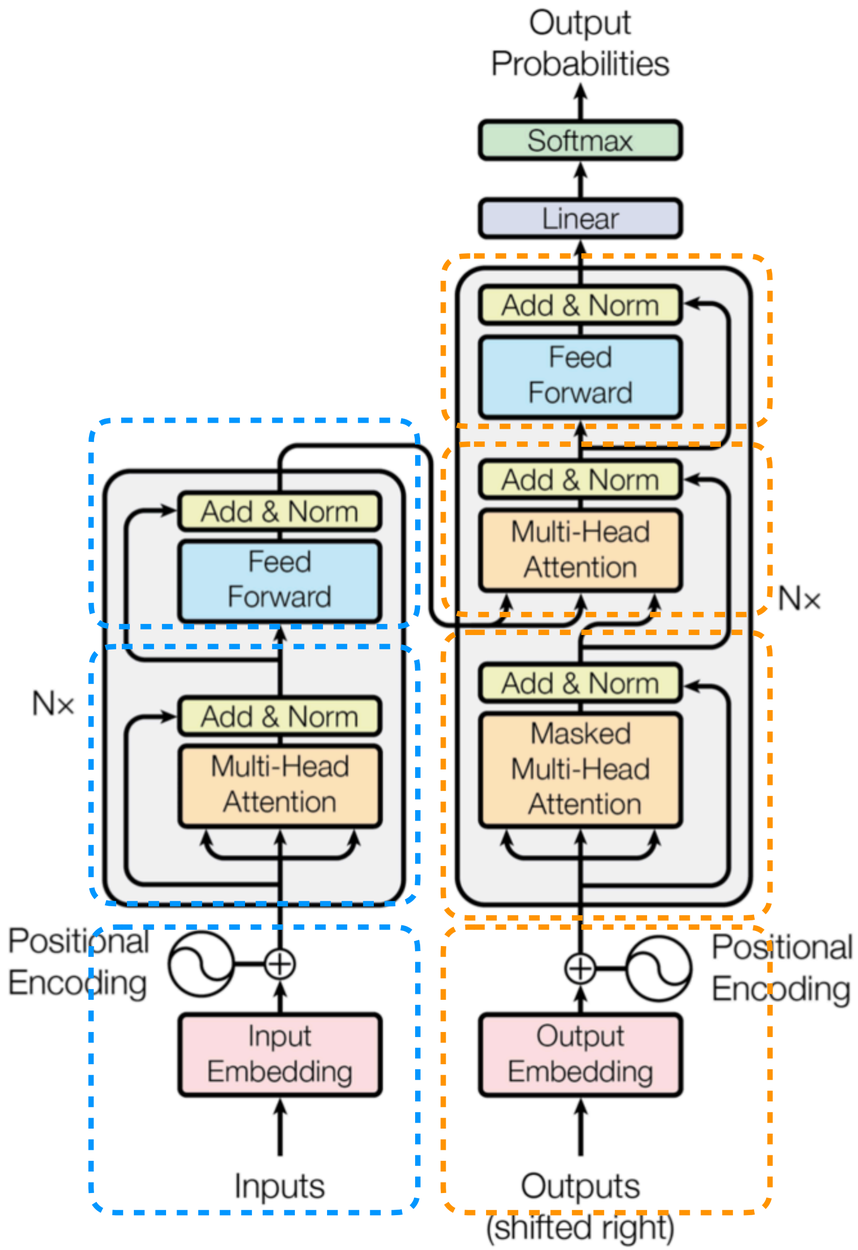
Transformerの構造分解

- Positional Encoding

- Multi-Head Attention

- Add & Norm

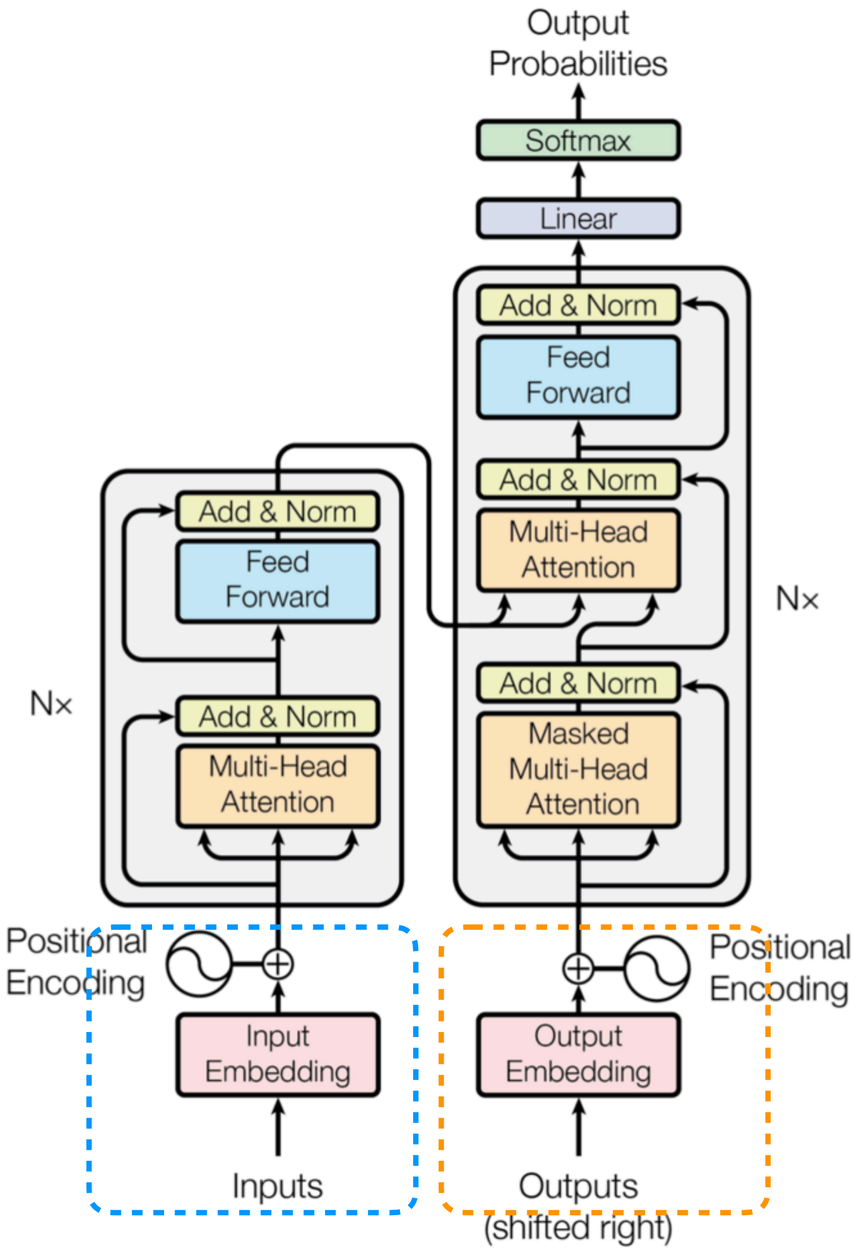
- Position-Wise Feed Forward Network



Positional Encoding

- （Embedされた）単語に時系列情報を付与する

  TransformerではRNNやCNNを用いないため、
  時系列情報を明示的に付与してやる必要がある



Positional Encoding

- 論文では、sin, cos を用いることにより、
  波を時系列情報として付与

- 行列PEの各成分を、Embeddingされた
  単語ベクトルの各次元に足す

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

※ d_{model} はEmbedding次元数

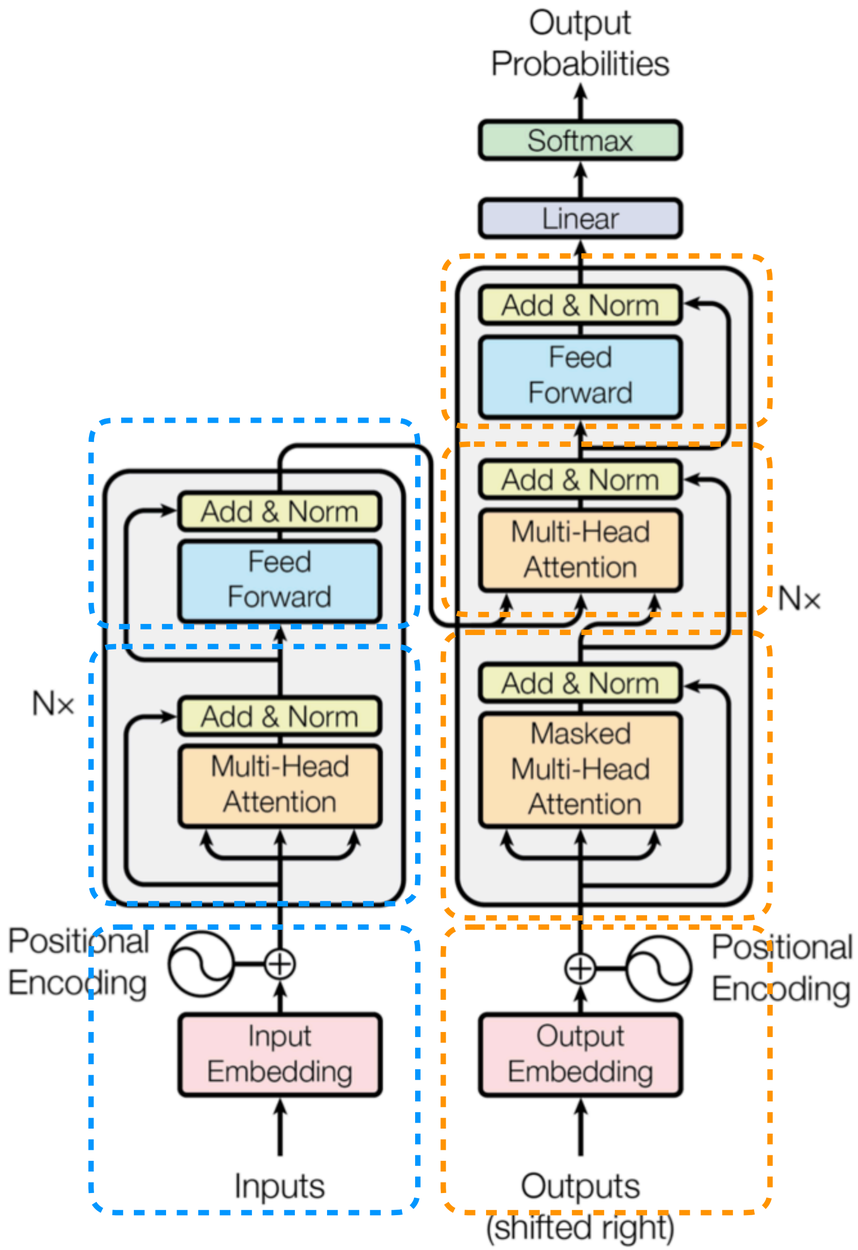
Transformerの構造分解

- Positional Encoding

- Multi-Head Attention

- Add & Norm

- Position-Wise Feed Forward Network

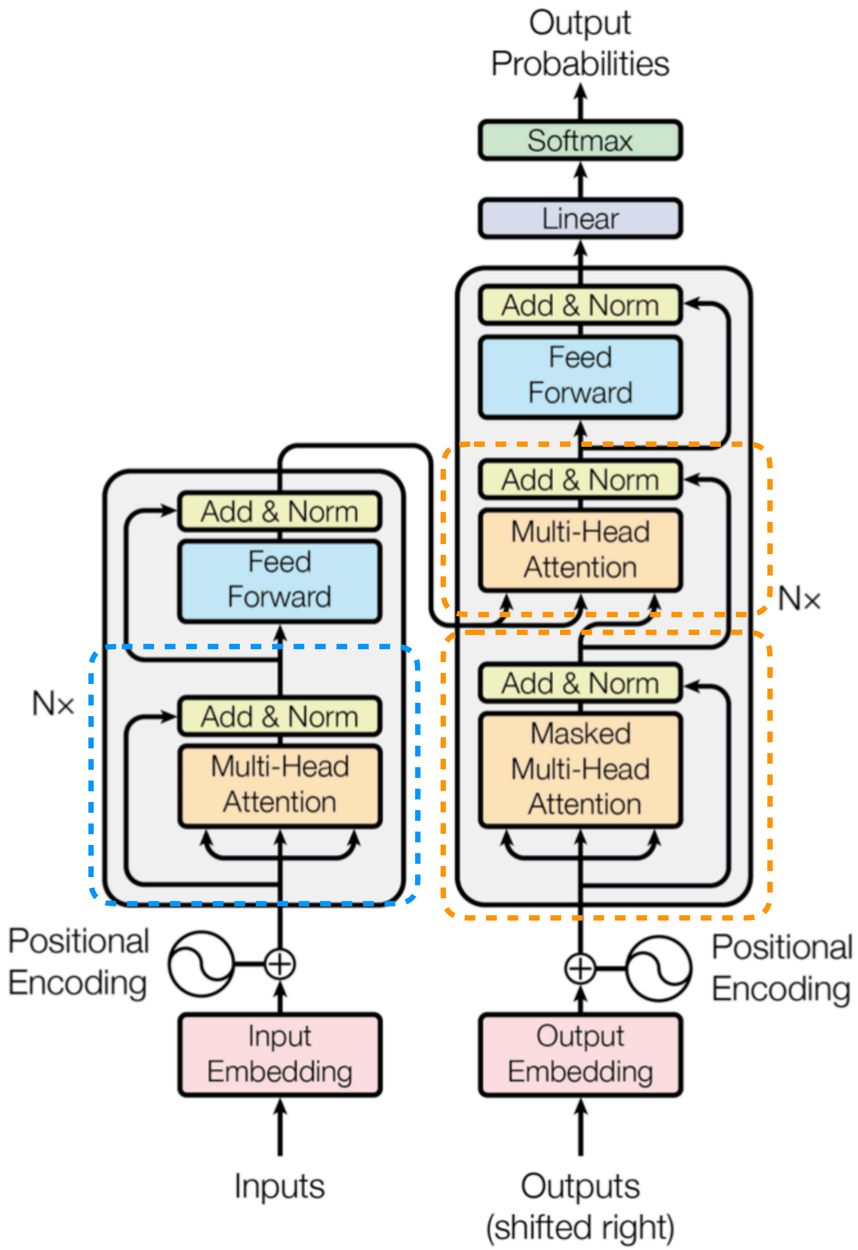


Multi-Head Attention

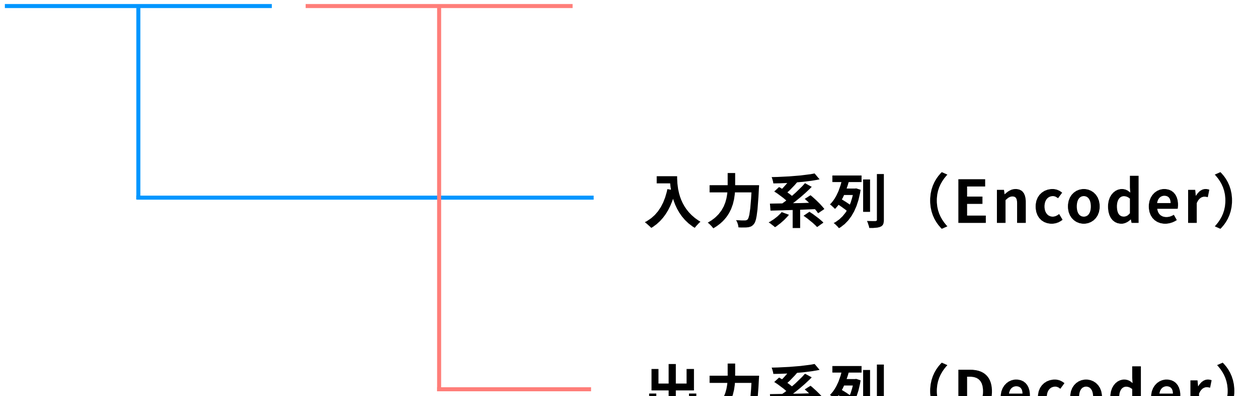
⇢ まずは

Source-Target-Attention と **Self-Attention**

について理解する必要がある



Source-Target-Attention



入力系列（Encoder）

出力系列（Decoder）

そもそも、Attentionの式を振り返ってみると…

$$\boldsymbol{c}_t = \sum_{s=1}^{S} \boldsymbol{a}_t(s)\bar{\boldsymbol{h}}_s$$

$$\boldsymbol{a}_t(s) = \mathrm{softmax}\left(\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t)\right)$$

Encoder ⇢ Decoder に状態を渡す際に

「時間の重み」を考慮したものがAttentionであった

そもそも、Attentionの式を振り返ってみると…

$$\boldsymbol{c}_t = \sum_{s=1}^{S} \boldsymbol{a}_t(s)\bar{\boldsymbol{h}}_s$$

スコア関数

$$\boldsymbol{a}_t(s) = \mathrm{softmax}\left(\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t)\right)$$

Encoder ⇢ Decoder に状態を渡す際に

「時間の重み」を考慮したものがAttentionであった

スコア関数自体はいくつか提案されている

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T \bar{\boldsymbol{h}}_s$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T W_a \bar{\boldsymbol{h}}_s$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{v}^T \tanh\left(W_{ad}\boldsymbol{h}_t^T + W_{ae}\bar{\boldsymbol{h}}_s\right)$$

スコア関数自体はいくつか提案されている

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T \bar{\boldsymbol{h}}_s$$

これを用いるとすると…

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T W_a \bar{\boldsymbol{h}}_s$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{v}^T \tanh\left(W_{ad}\boldsymbol{h}_t^T + W_{ae}\bar{\boldsymbol{h}}_s\right)$$

Attentionの式

$$\boldsymbol{c}_t = \sum_{s=1}^{S} \mathrm{softmax}\left(\boldsymbol{h}_t^T \bar{\boldsymbol{h}}_s\right) \bar{\boldsymbol{h}}_s$$

行列で表現すると…

$$\mathrm{Attention}(S, T) = \mathrm{softmax}(TS^T)S$$

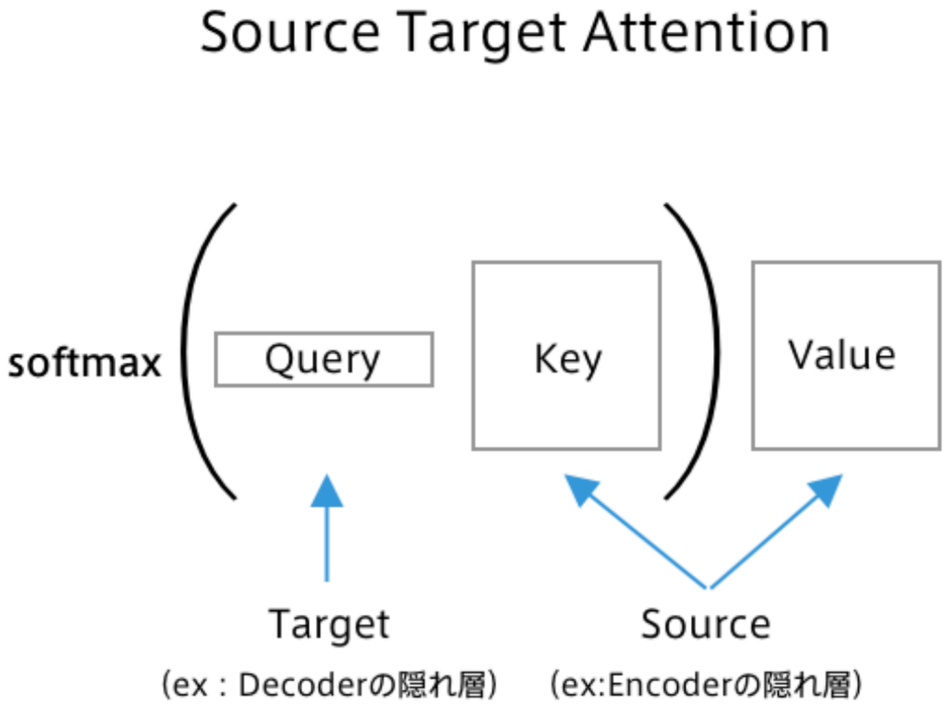
$$\mathrm{Attention}(S, T) = \mathrm{softmax}(TS^T)S$$

この式を一般化して考えると、Attentionは

query ベクトル と key ベクトル の類似度を求め、
その正規化した重みを value ベクトル に適用して
値を取り出す

という処理を行っていると解釈できる

（Source が Key・Value の辞書、Target が Query）



すなわち、Attentionの式は

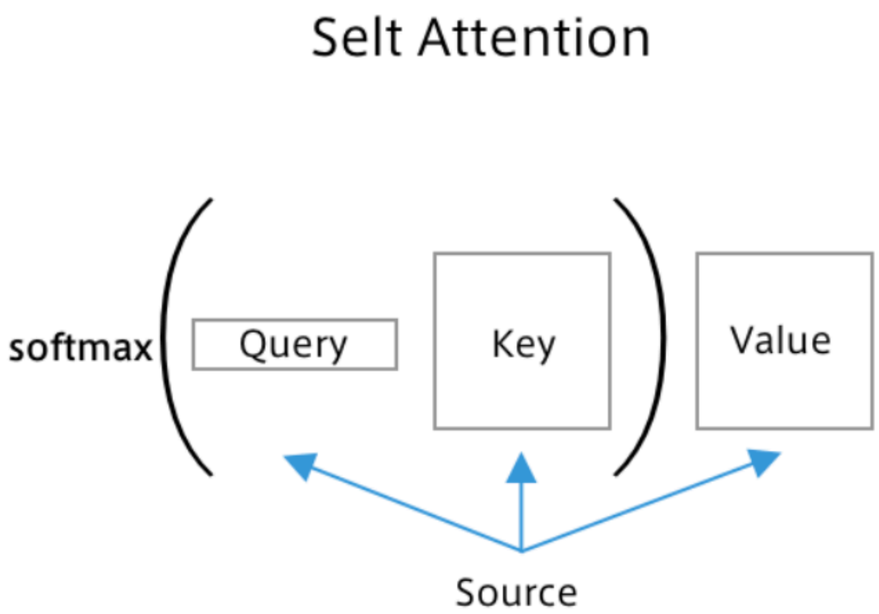
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(QK^T)V$$

という形で一般化できる

（Source-Target-Attention では、Q が Target、K, V が Source）

Transformer では Source-Target-Attentionに加え、
query・key・value を同じ系列内で定義する
Self-Attentionが用いられる

（すなわち、Q, K, V すべてが Source）



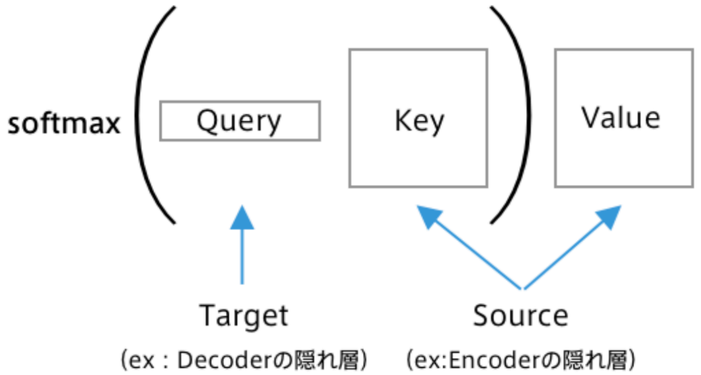
まとめると…

いずれのAttentionも式は同じ

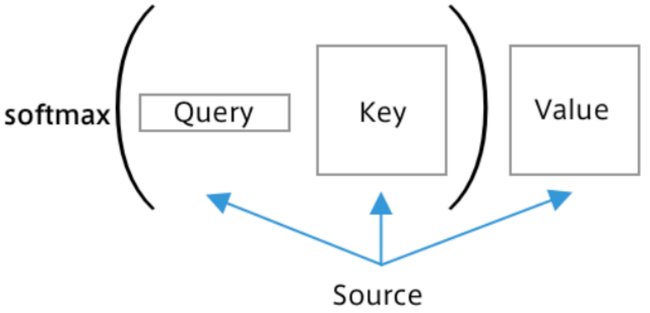
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(QK^T)V$$

Q, K, V が Source なのか Target なのかが異なる

Source Target Attention



Selt Attention



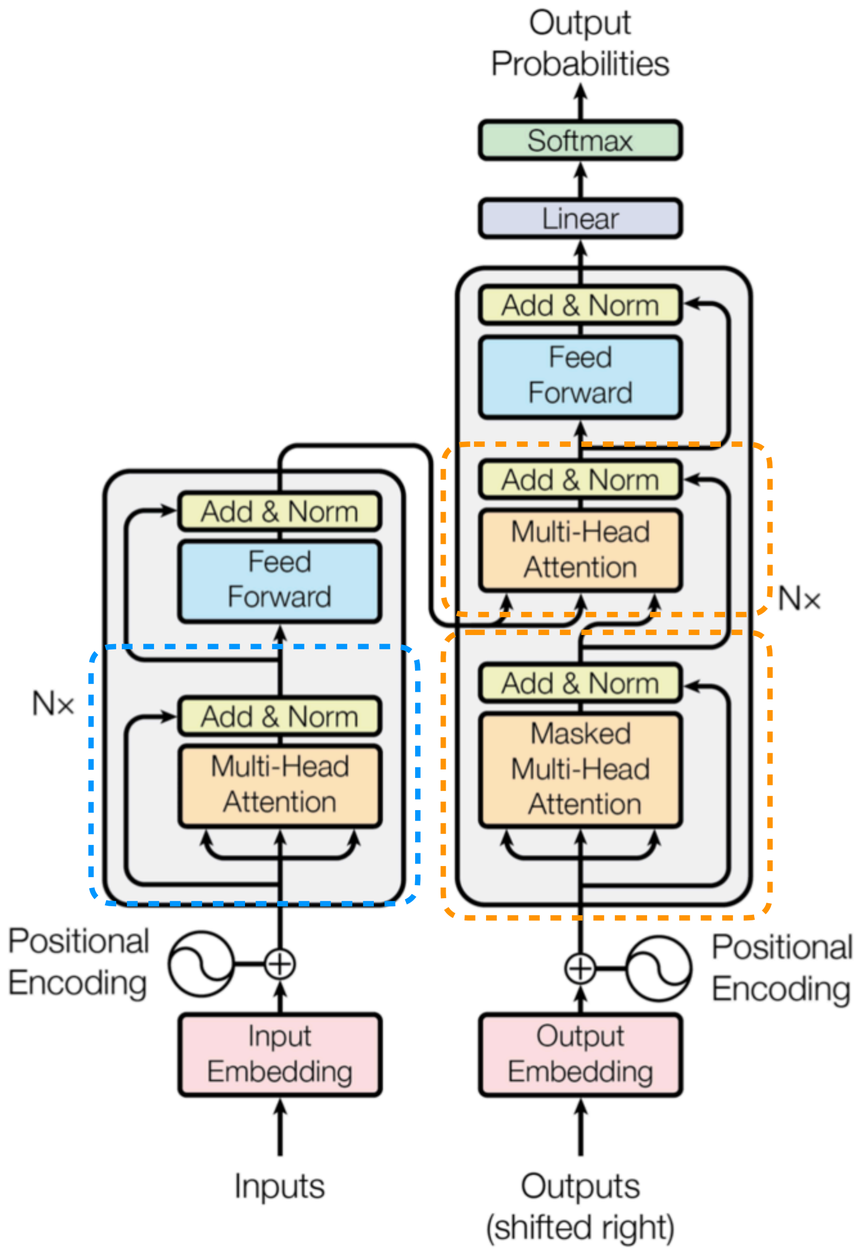
（再掲）

Multi-Head Attention

⇢ まずは

Source-Target-Attention と **Self-Attention**

について理解する必要がある



Multi-Head Attention

⇢ まずは

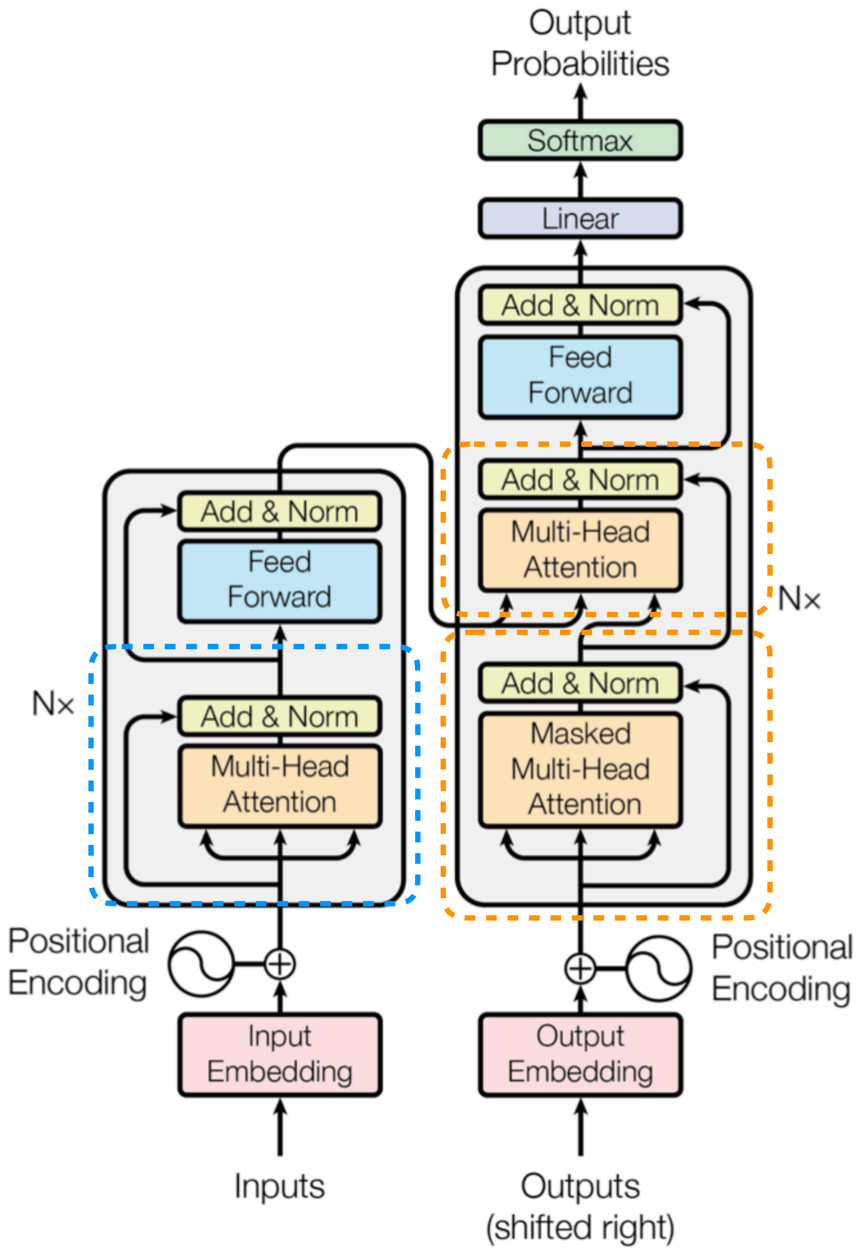
Source-Target-Attention と Self-Attention

について理解する必要がある

⇢ **Additive-Attention, Dot-Product Attention,**

Scaled Dot-Product Attention

について理解する必要がある



（再掲）

スコア関数自体はいくつか提案されている

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T \bar{\boldsymbol{h}}_s$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T W_a \bar{\boldsymbol{h}}_s$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{v}^T \tanh\left(W_{ad}\boldsymbol{h}_t^T + W_{ae}\bar{\boldsymbol{h}}_s\right)$$

スコア関数自体はいくつか提案されている

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T \bar{\boldsymbol{h}}_s$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T W_a \bar{\boldsymbol{h}}_s$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{v}^T \tanh\left(W_{ad}\boldsymbol{h}_t^T + W_{ae}\bar{\boldsymbol{h}}_s\right)$$

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{v}^T \tanh\left(W_{ad}\boldsymbol{h}_t^T + W_{ae}\bar{\boldsymbol{h}}_s\right)$$

- **Google’s Neural Machine Translation で用いられているscore関数**

- **フィードフォワードのニューラルネットワークを用いて Attentionの重みを求めようとしている**

これを **Additive Attention（加法注意）**と言う

Additive Attention に対し、スコア関数として

$$\mathrm{score}(\bar{\boldsymbol{h}}_s, \boldsymbol{h}_t) = \boldsymbol{h}_t^T \bar{\boldsymbol{h}}_s$$

を用いているものは、内積計算のみで Attention の重みを求めようとしている

これを **Dot-Product Attention（内積注意）**と言う

まとめると…

Additive Attention

$$\mathrm{softmax}\left(\mathrm{FFN}([\mathrm{Q};\mathrm{K}])\right)$$

Dot-Product Attention

$$\mathrm{softmax}\left(QK^T\right)$$

- パラメータが少ないので高速

- Transformer が使っているのはこちら

Transformer では Dot-Product Attention に更にスケーリングを施している

＝ Scaled Dot-Product Attention

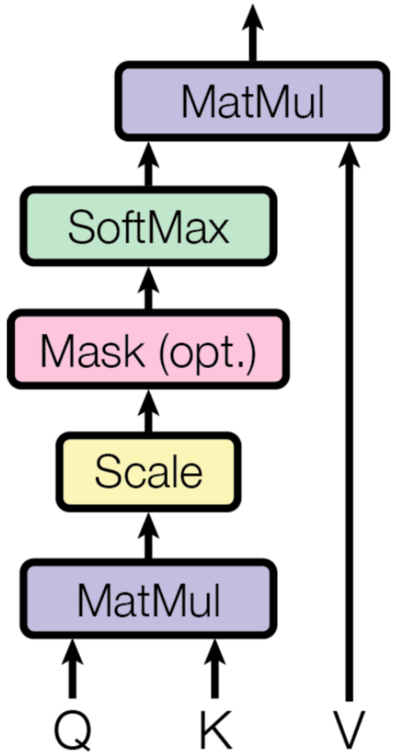
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

keyベクトルの次元数

※ スケーリングをしないと、keyベクトルの次元数が大きい際に
内積が大きくなりすぎて逆伝播の際にsoftmaxの勾配が消失しやすくなる

Scaled Dot-Product Attention

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

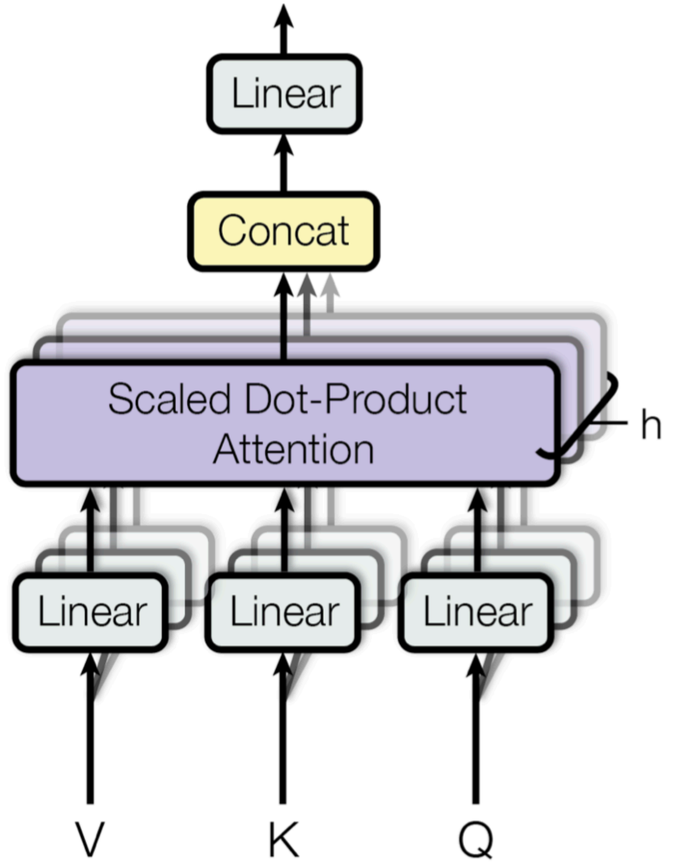


Multi-Head Attention

- Scaled Dot-Product Attention を複数（ヘッド）で行う

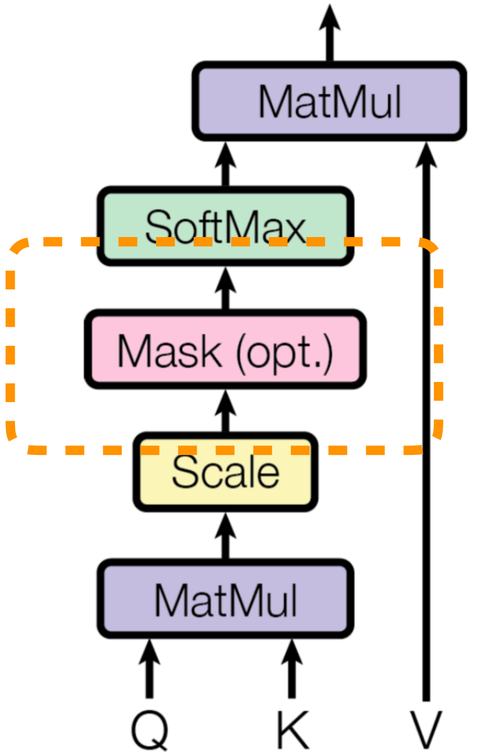
  それぞれのヘッドが、異なる位置の異なる部分空間を処理することを期待

- Q, K, V それぞれをヘッド数だけ線形活性しておき、重みを学習する



Decoder に関しては、未来の情報がAttentionに
含まれないようにマスクをかける処理をしている

= Masked Multi-Head Attention



（実際は Scaled Dot-Product Attention 内でマスク処理を行う）

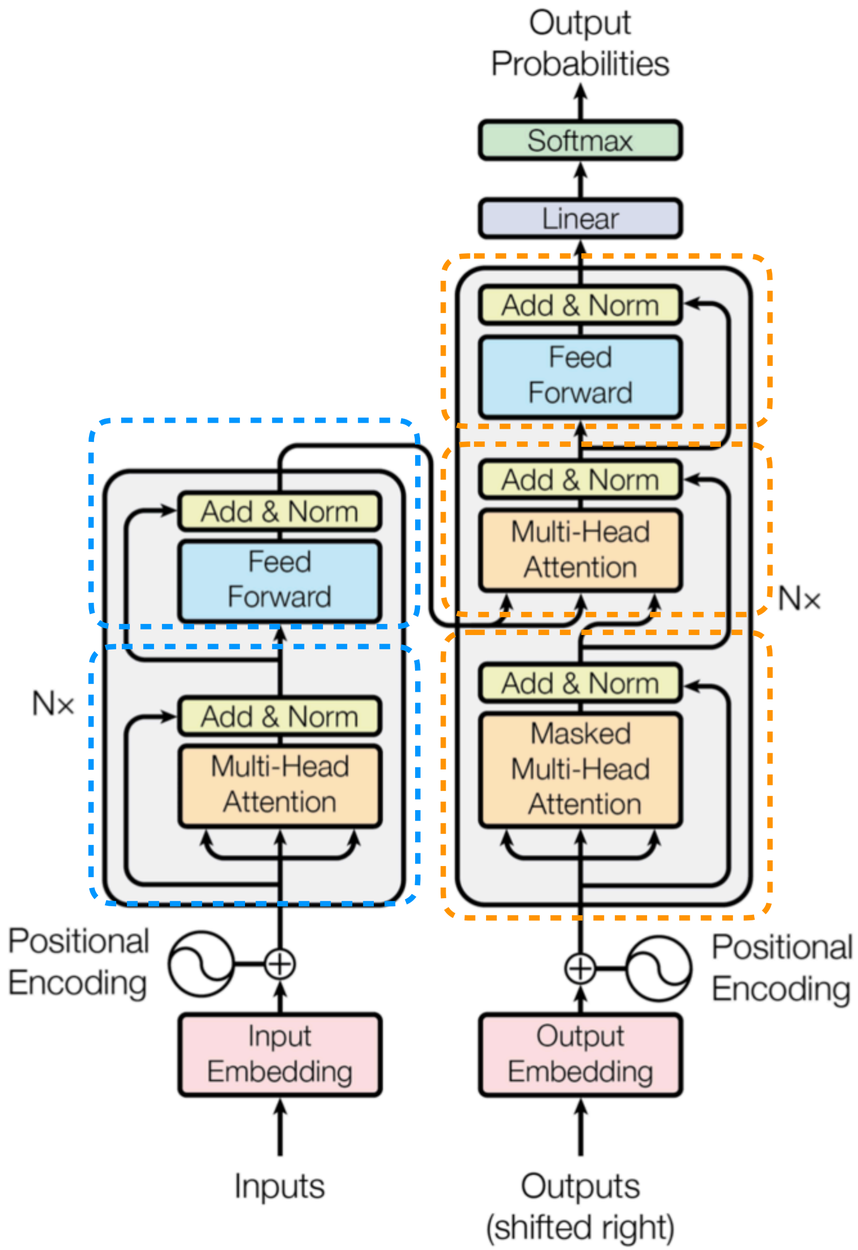
Transformerの構造分解

- Positional Encoding

- Multi-Head Attention

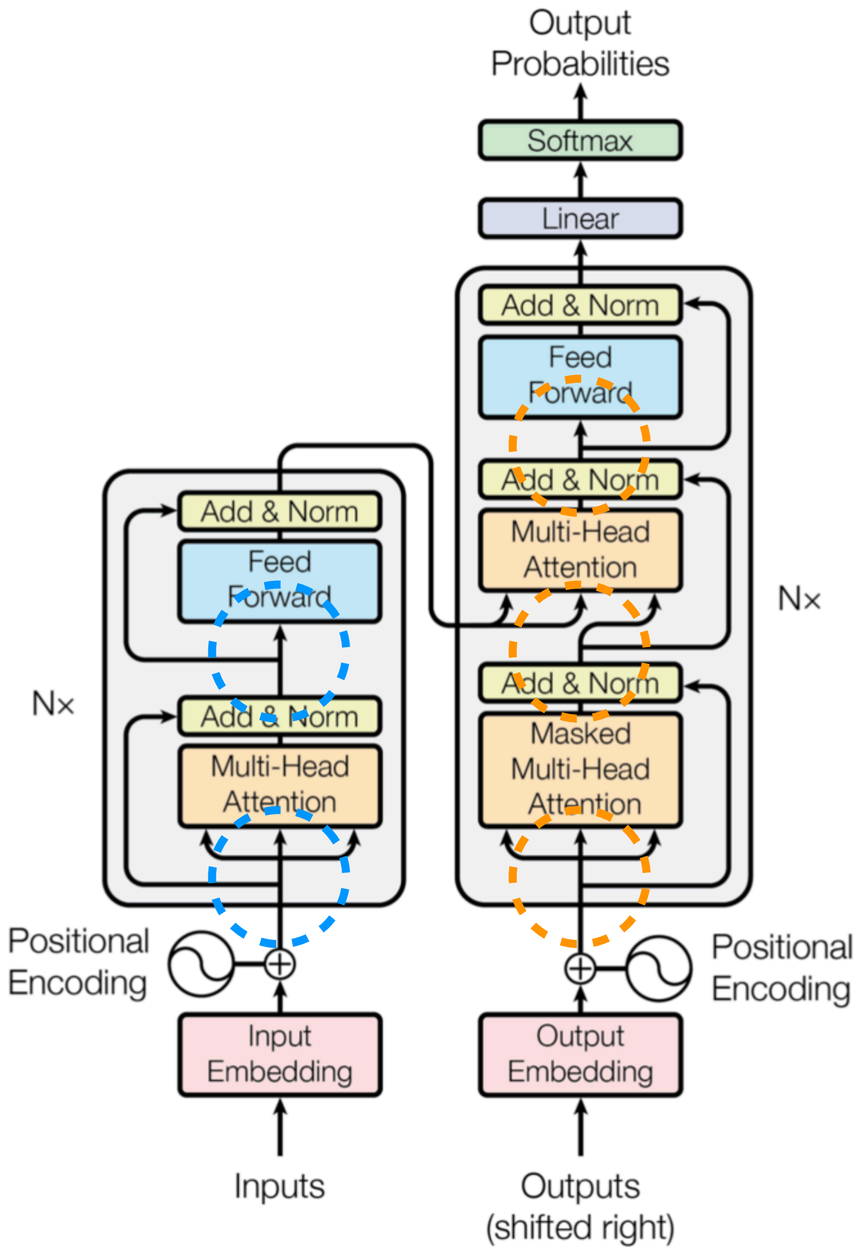
- Add & Norm

- Position-Wise Feed Forward Network



Add は Residual Connection を行う

Norm は Layer Normalization を行う



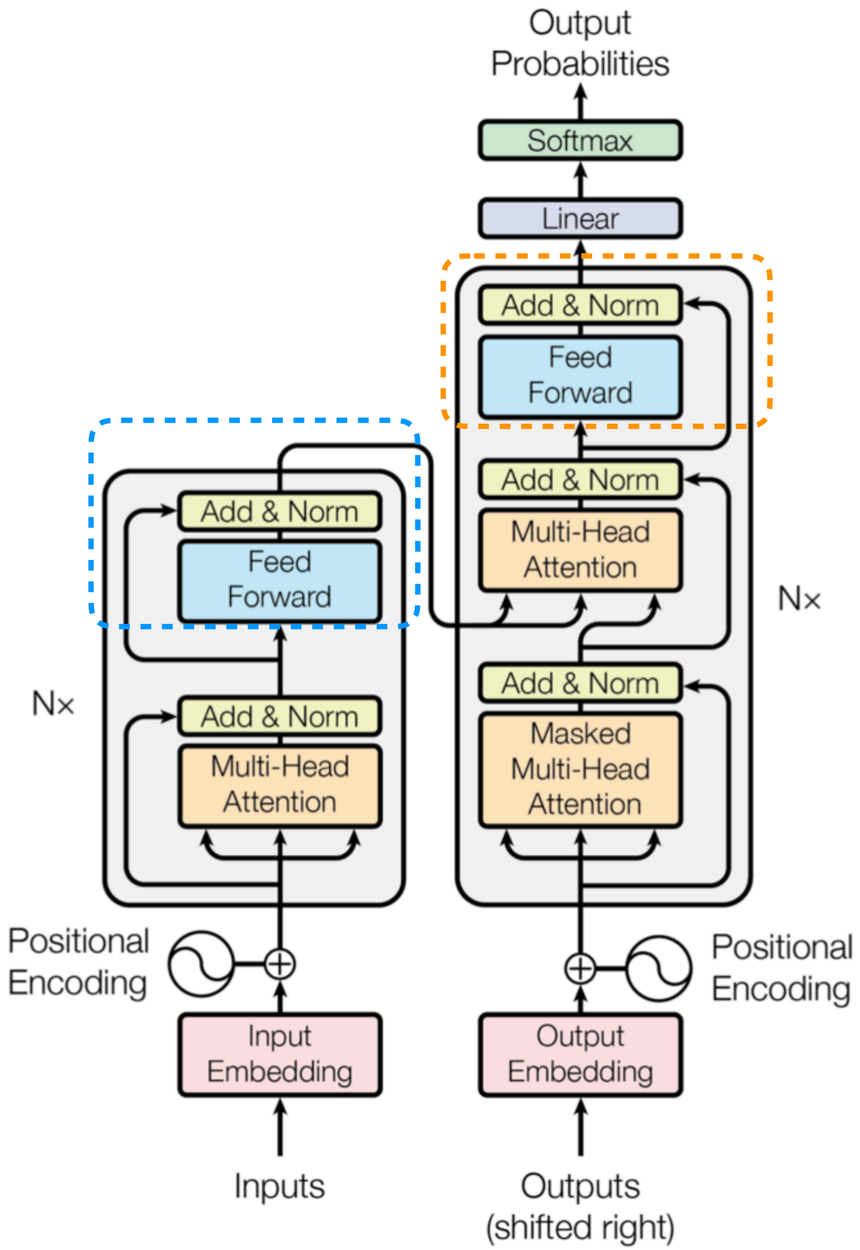
Transformerの構造分解

- Positional Encoding

- Multi-Head Attention

- Add & Norm

- Position-Wise Feed Forward Network



Position-Wise Feed Forward Network

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

- 線形活性 ＋ ReLU ＋ 線形活性 のネットワーク（２層のネットワーク）

- 単語列の位置ごとに独立して処理する

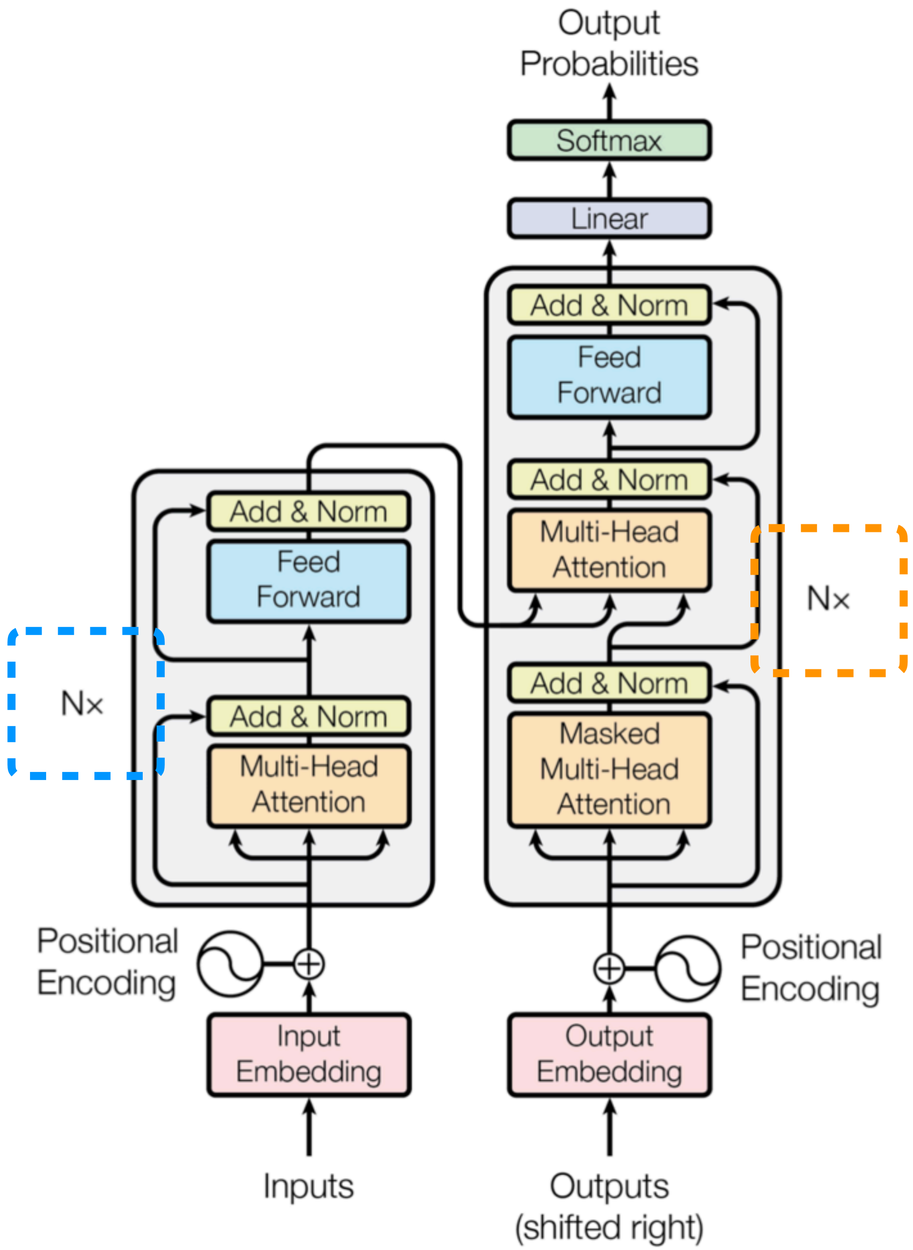
すなわち、時間軸方向にも次元をもつ

※ nn.Linear でもよいが、演習では nn.Conv1D を用いている（式は同じ）

Encoder と Decoder は、

それぞれの構造が Nx (=8) 回積み重なったもの

これで Transformer が完成！



演習