

Attention Is All You Need

著者 : Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

出典 : Advances in Neural Information Processing Systems 30 (NIPS 2017)

目次

- 従来手法
- Attention
- Transformer
 - 機構
 - 推論時 , 学習時の流れ
 - エンコーダ , デコーダの動き
 - 応用例
- まとめ

時系列予測

時系列データ：個々の要素に順序があるデータ

- 自然言語
- 気象データ
- 株価

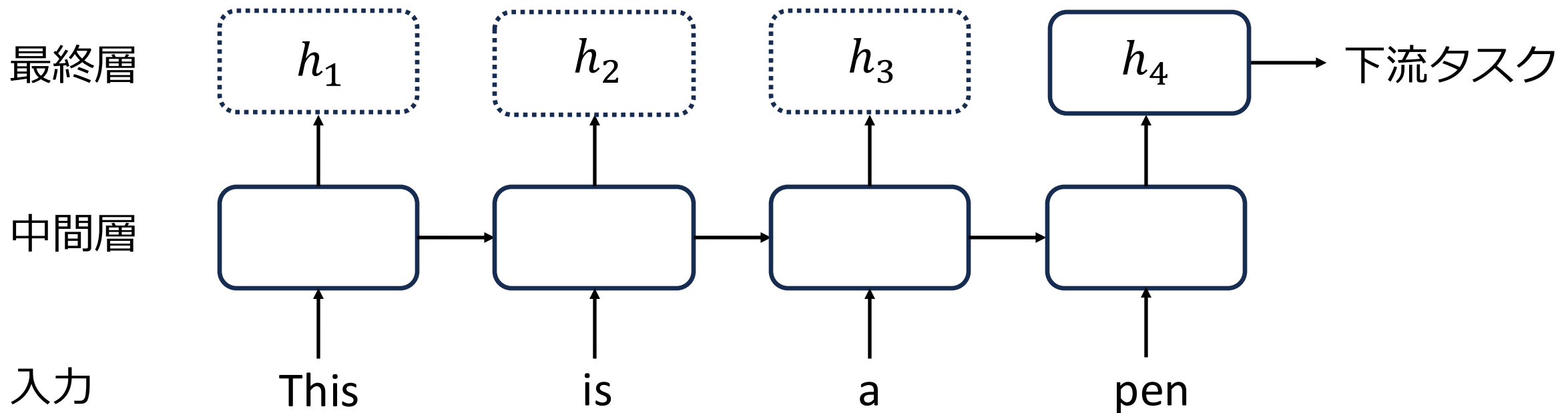
時系列予測のための従来手法

- Recurrent Neural Network (RNN[1986])
- Sequence to Sequence (seq2seq[2014])

RNN[1986]

中間層に閉路を持つニューラルネットワーク

- 時系列順に要素を入力
- 中間層で過去の情報を伝播
- 最終的な出力が埋め込み

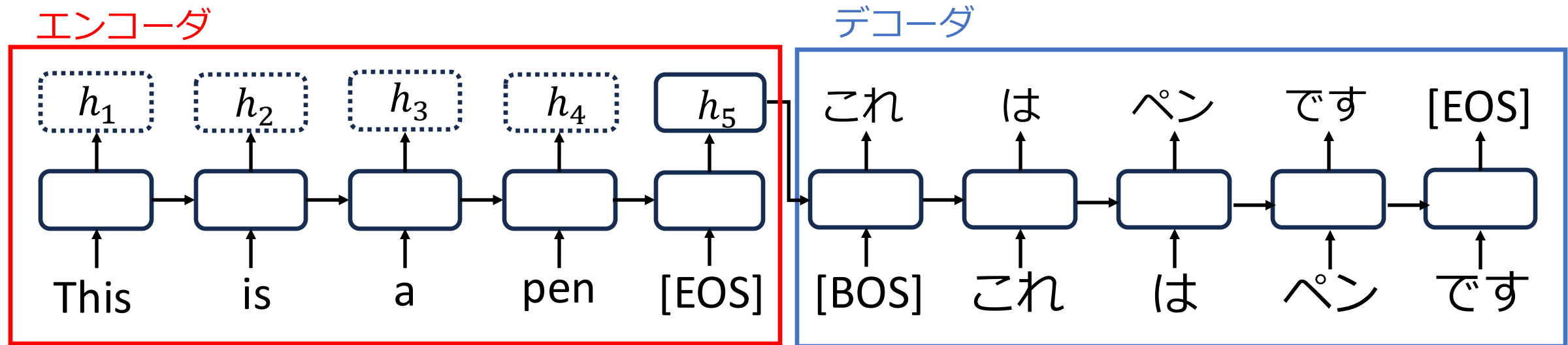


seq2seq[2014]

時系列データから別の時系列データへ変換

- ・エンコーダ：入力データから埋め込みを生成
- ・デコーダ：埋め込みから時系列データを生成

例：翻訳タスク

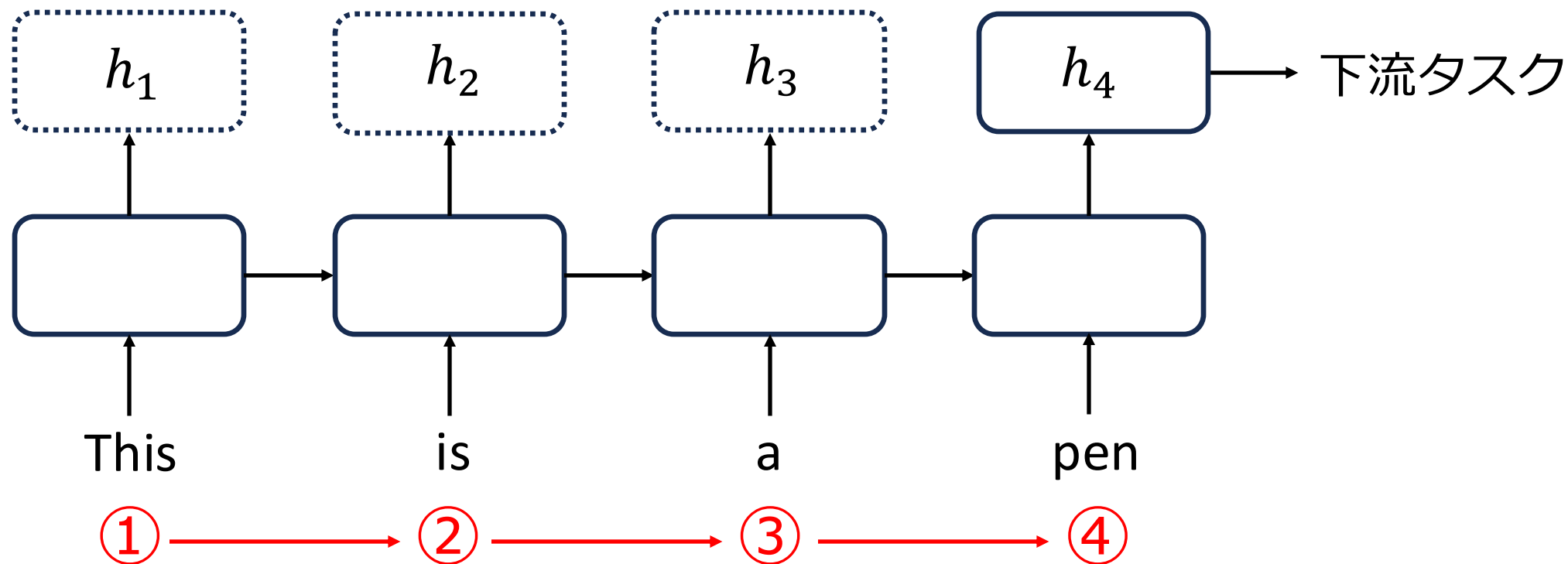


※[BOS]：文章の始まり，[EOS]：文章の終わり

課題点(1/2)

時系列データを逐次的に入力する必要あり

- 並列化できず, 計算速度の向上が困難
- 単語数の増加で計算量が線形増加

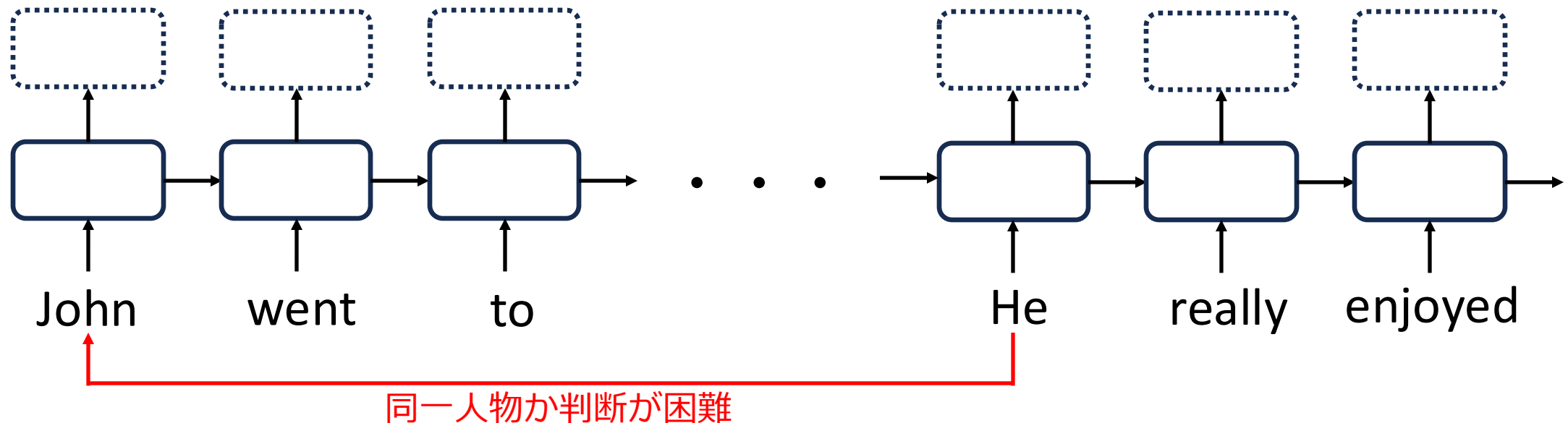


課題点(2/2)

長期的な依存関係の捕捉が困難

- RNNの逆伝播はシーケンスが長いほど勾配消失
- 逐次的に入力するため過去の情報は徐々に減衰

例 : John went to the store to buy some food. He met his old friend there.
They talked about their school days. He really enjoyed catching up with him.



新たなアプローチ

Transformer : Attention機構で構成されたモデル

利点

- 並列処理が可能に
- 長期的な依存関係を効率的に捕捉
- RNNより勾配消失が起きにくい

Attention

- 重要な情報に注目する仕組み
- 機械翻訳では単語同士の関連度を計算し重み付け

| | This | is | a | pen | | This | is | a | pen |
|----|------|------|------|------|----|------|----|---|-----|
| これ | 0.80 | 0.03 | 0.15 | 0.02 | これ | This | is | a | pen |
| は | 0.02 | 0.95 | 0.03 | 0.00 | は | This | is | a | pen |
| ペン | 0.02 | 0.03 | 0.25 | 0.70 | ペン | This | is | a | pen |
| です | 0.15 | 0.05 | 0.75 | 0.05 | です | This | is | a | pen |

事前準備

- 入力

- X_{Source} : 参照したい情報
- X_{Target} : 主となる情報

- 機械翻訳の例

- X_{Source} : This is a pen (元の文)
- X_{Target} : これは ペン です (翻訳文)

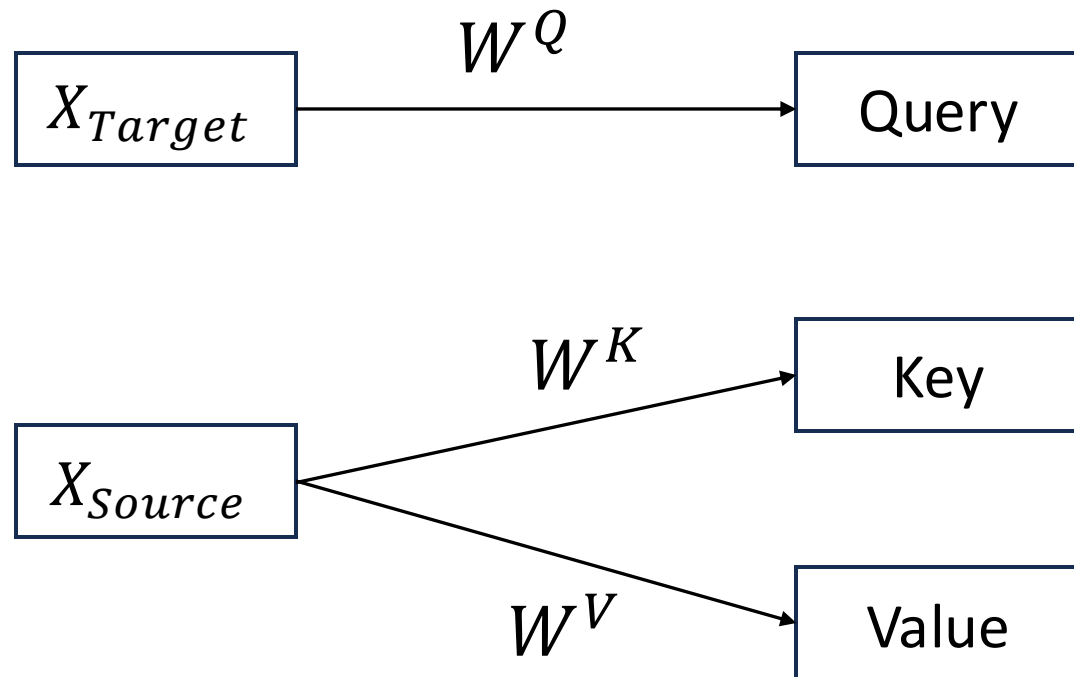
事前準備

- 入力を Query , Key , Value にマッピング
 - W : 学習可能なパラメータ行列
 - X : 入力行列

$$Q = X_{Target} W^Q$$

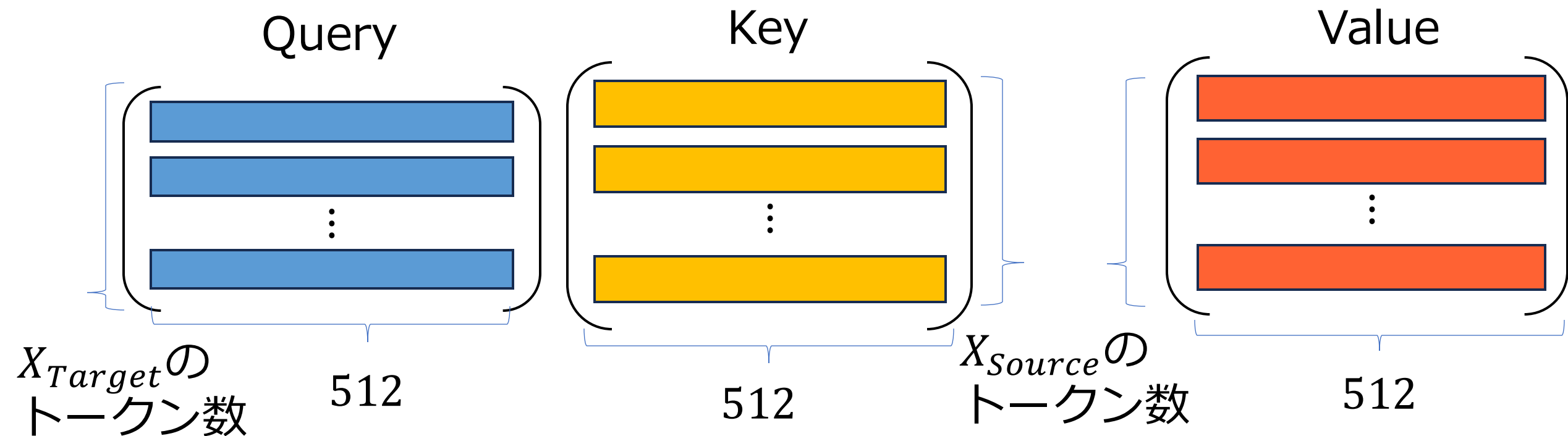
$$K = X_{Source} W^K$$

$$V = X_{Source} W^V$$



Q, K, V の次元数

- Query : X_{Target} のトークン数 \times 512
- Key , Value : X_{Source} のトークン数 \times 512



Attention計算

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

d は Q, K の埋め込みベクトルの次元数

- データ同士の関連度計算
- 関連度のスケールリング
- 関連度のソフトマックス正規化
- Q と K の関連度でValueの重みつけ

Attention計算

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

d は Q, K の埋め込みベクトルの次元数

- データ同士の関連度計算
- 関連度のスケールリング
- 関連度のソフトマックス正規化
- Q と K の関連度でValueの重みづけ

Attention計算

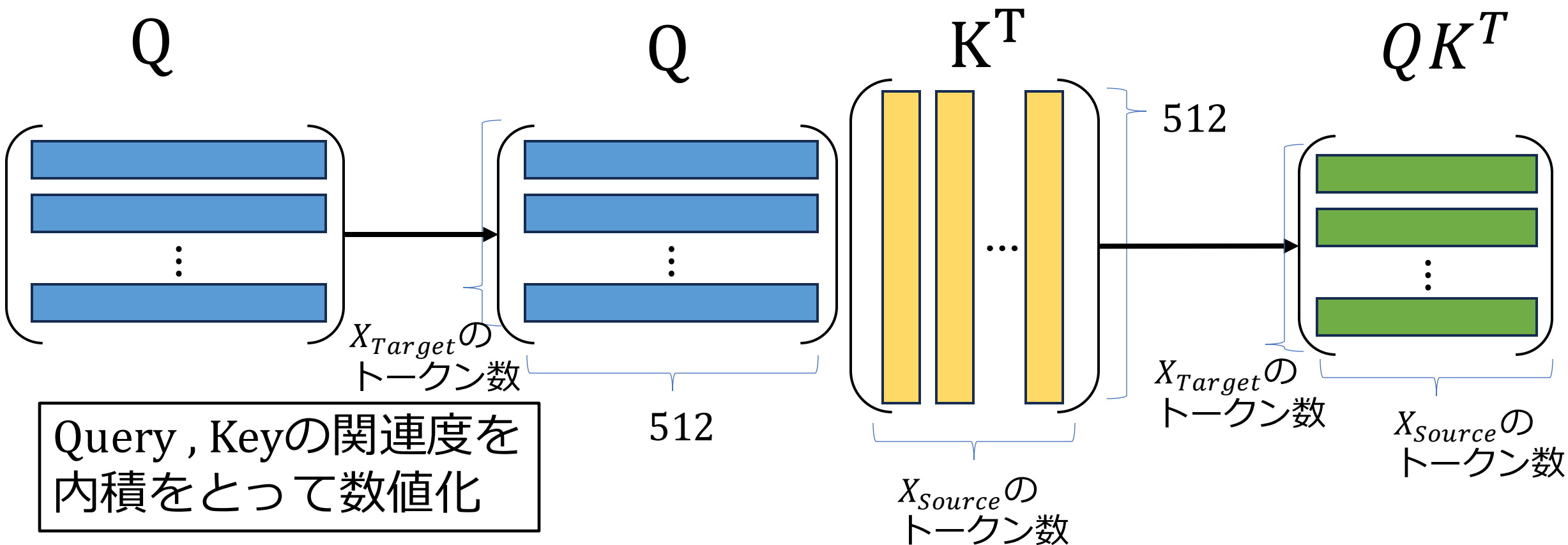
$$Attention(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d}} \right) V$$

d は Q, K の埋め込みベクトルの次元数

- データ同士の関連度計算
- 関連度のスケーリング
- 関連度のソフトマックス正規化
- Q と K の関連度でValueの重みづけ

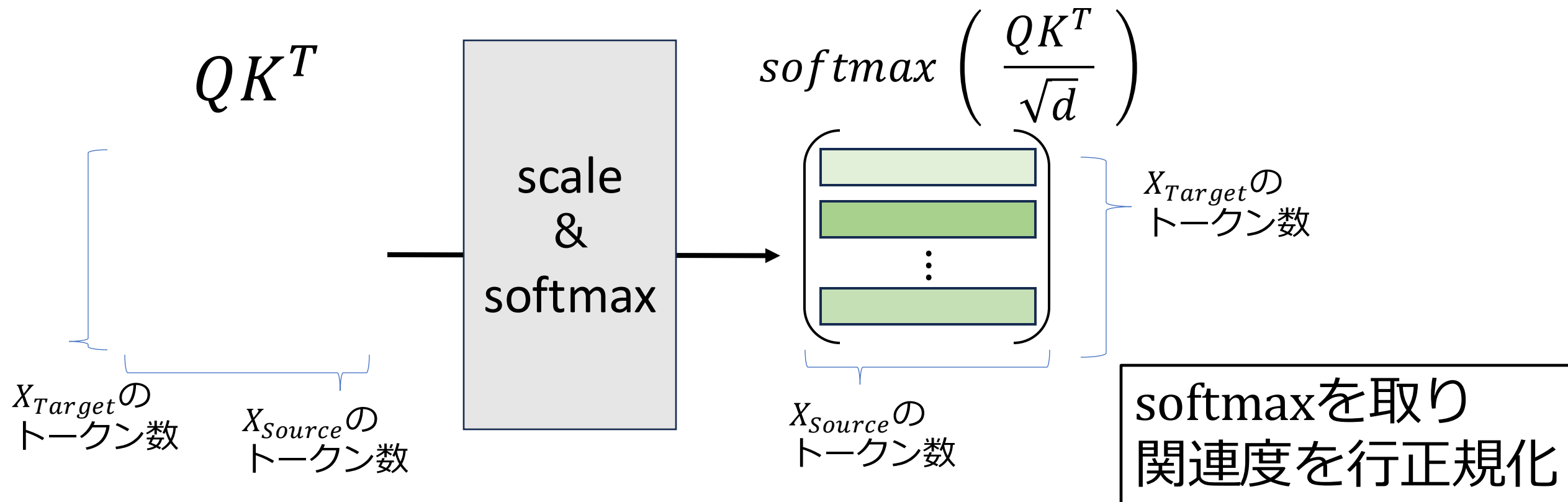
Attention計算

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



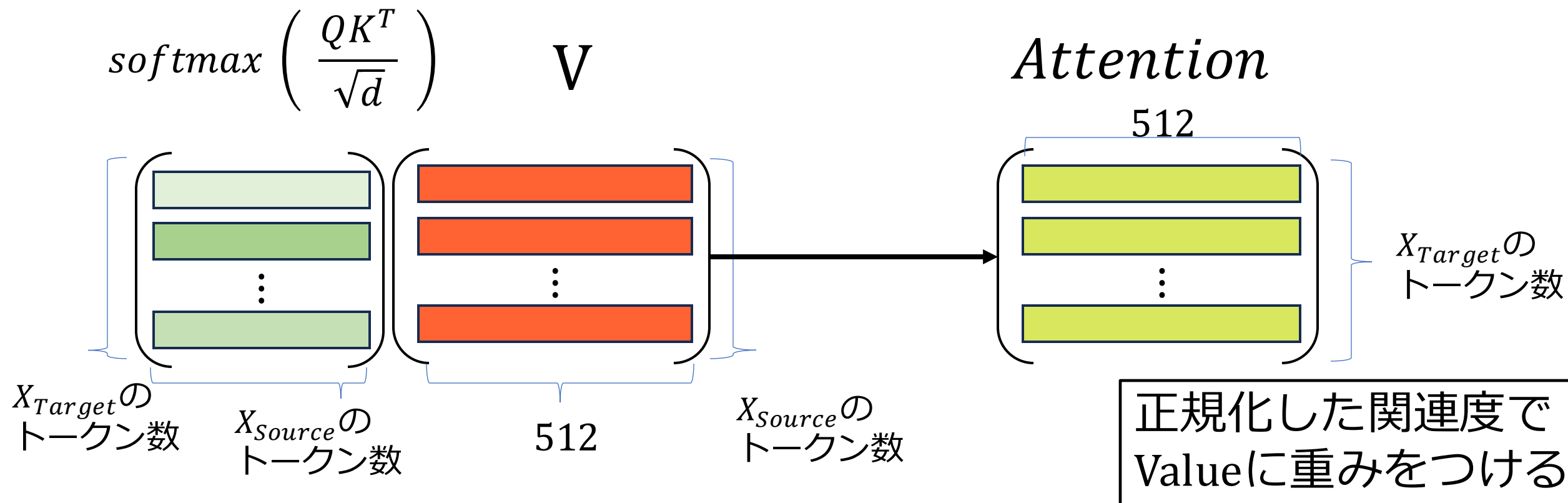
Attention計算

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

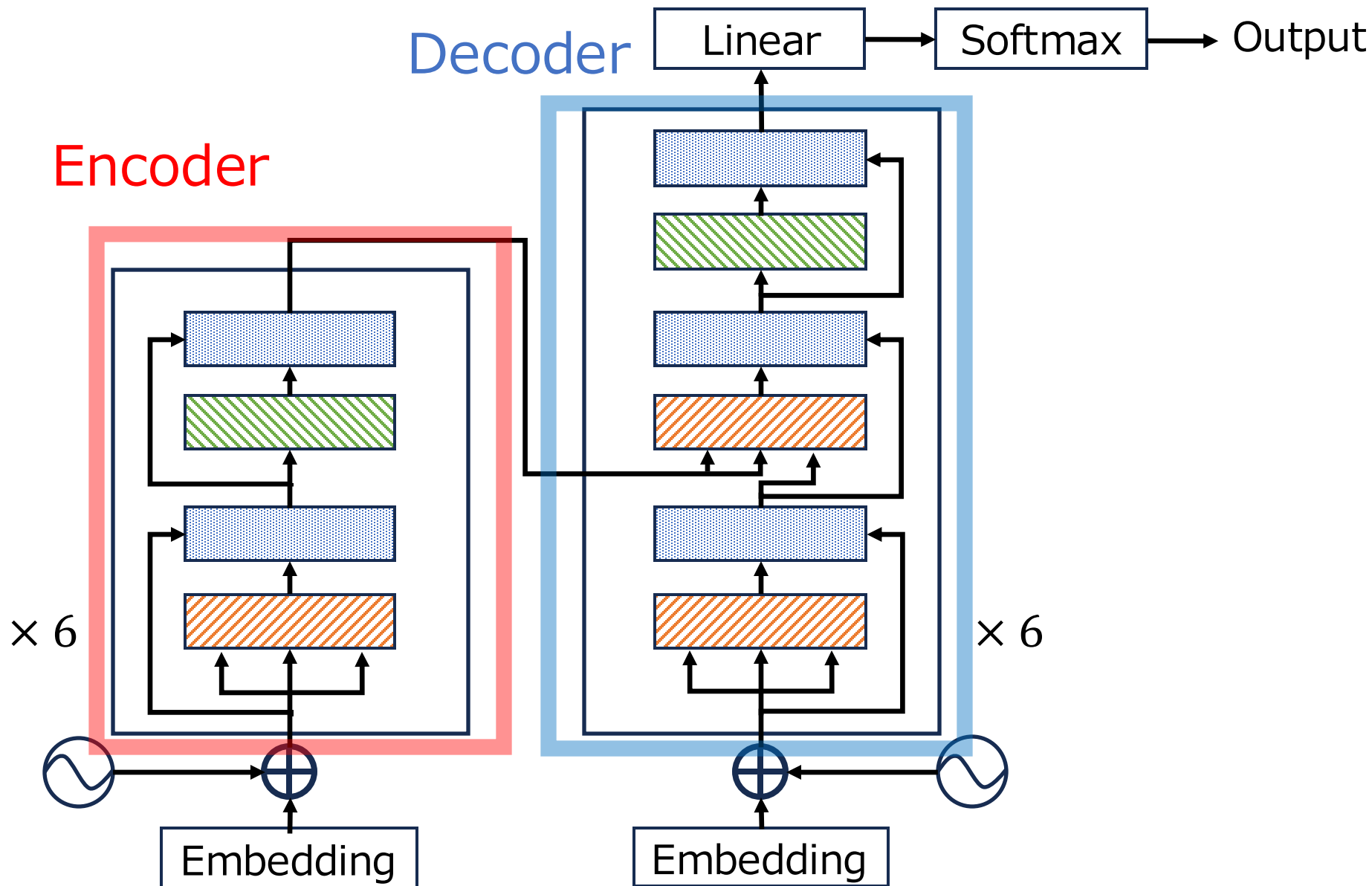


Attention計算

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



Transformerの全体像



- 前処理
- Multi-Head Attention
- フィードフォワードネットワーク

機構

- 前処理
 - トークン化
 - Embedding
 - Positional Encoding
- Multi-Head Attention
- フィードフォワードネットワーク

トークン化

- トークンとは
 - テキストデータを処理する際に使用される基本的な単位
- トークン化についてのいくつかの手法がある
 - 単語トークン化
 - 文字トークン化
 - サブワードトークン化

単語トークン化

- テキストを単語ごとに分割する方法
 - スペースや句読点に基づく手法
 - 辞書を使用して単語を特定する手法 等がある
- 具体例
 - 入力 : "I love eating pizza"
 - トークン : ["I" , "love" , "eating" , "pizza"]

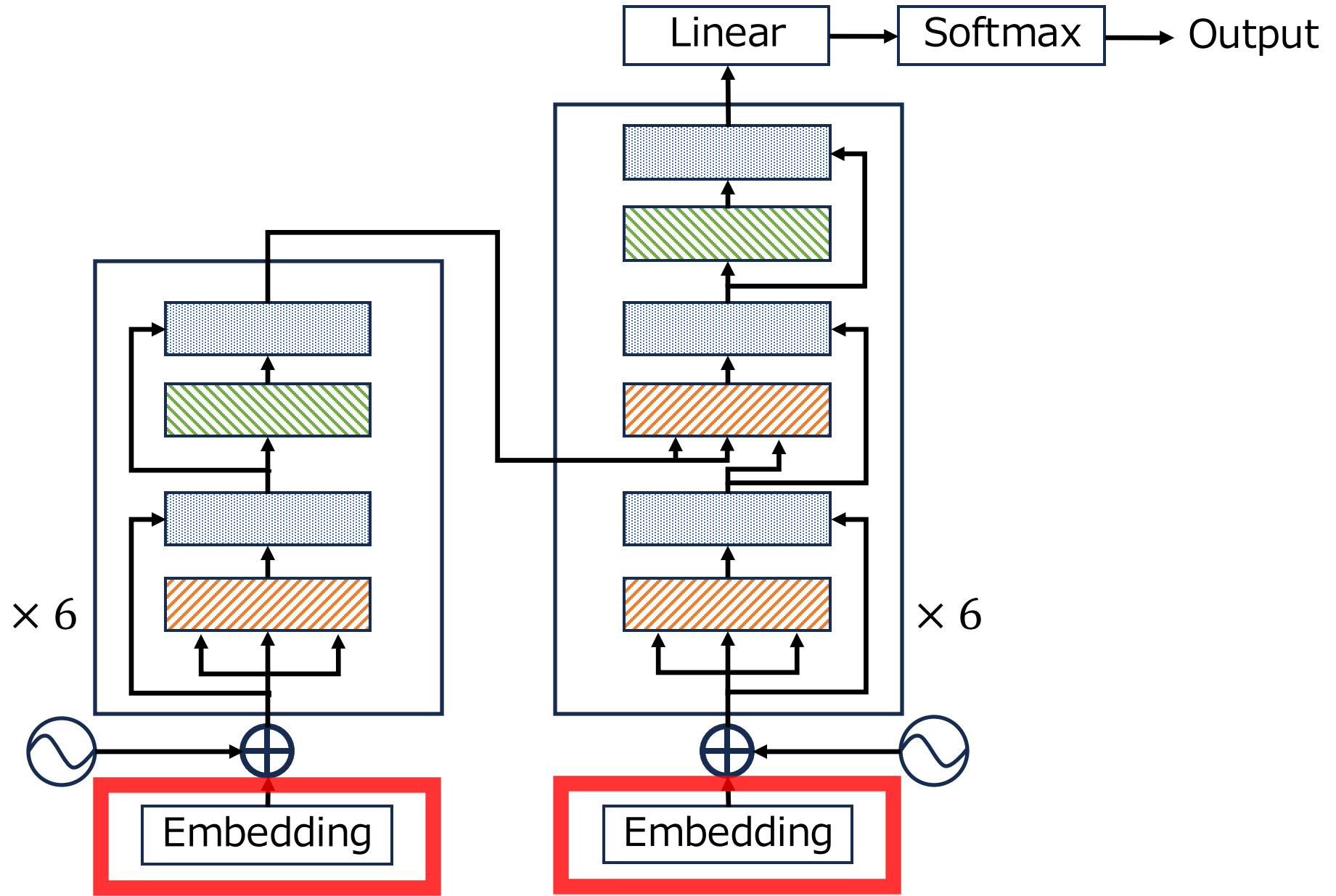
文字トークン化

- テキストを個々の文字に分割する方法
 - 単語の境界を意識せずに文字の並びをトークンとして扱う
- 具体例
 - 入力 : "Hello"
 - トークン : ["H" , "e" , "l" , "l" , "o"]

サブワードトークン化

- テキストを部分文字列に分割する方法
 - 単語レベルと文字レベルの中間的手法
- 具体例
 - 入力 : "I loved eating pizza"
 - トークン : ["I" , "love" , "d" , "eat" , "ing" , "pizza"]

Embedding

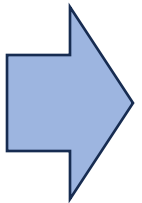


Embedding (機械翻訳の例)

- トークンをベクトルに変換

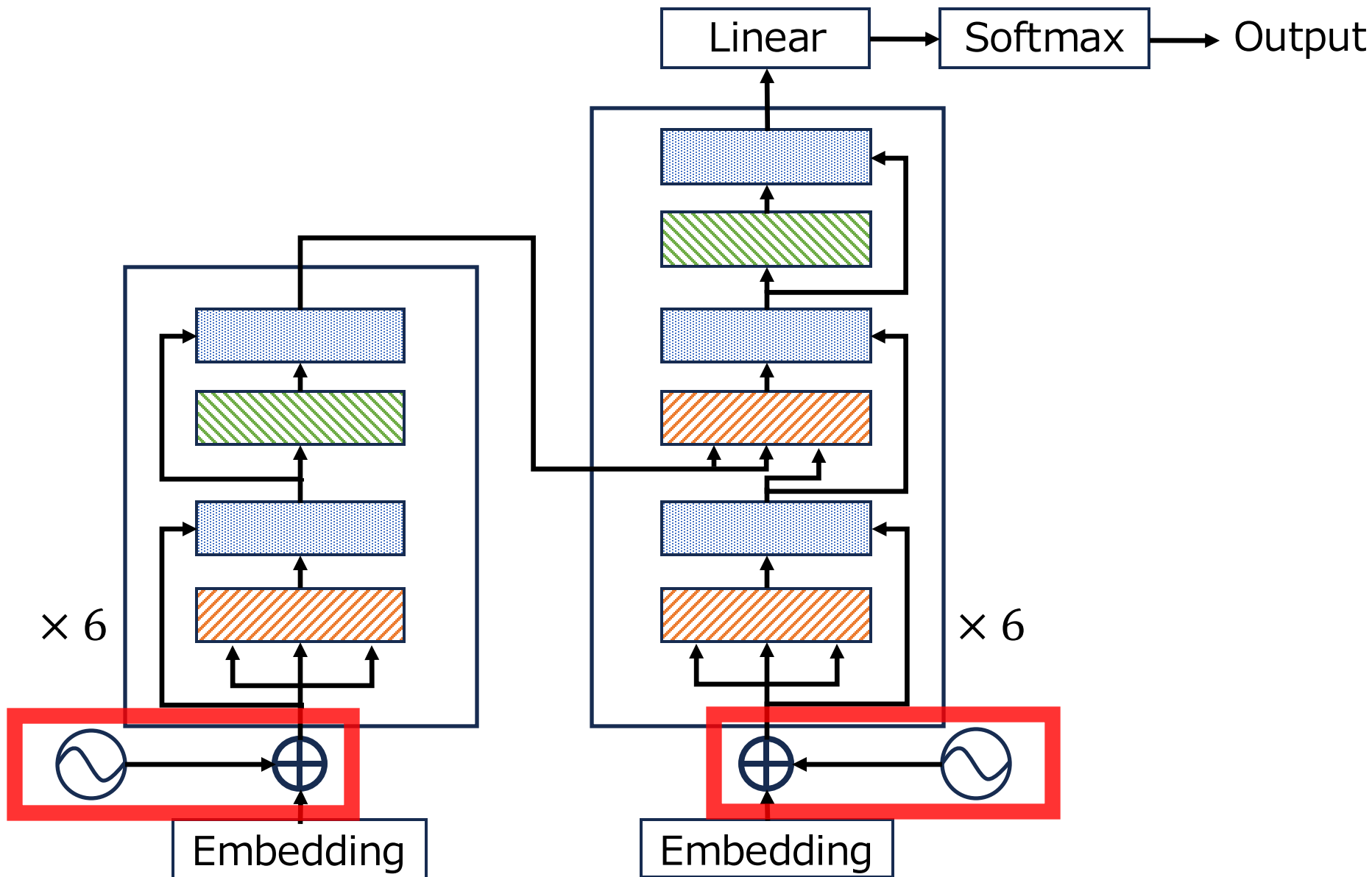
d : 特徴次元数

I
am
good


$$X' = \begin{bmatrix} 2.62 & 1.54 & \dots & 7.64 \\ 1.45 & 2.31 & \dots & 4.38 \\ 3.22 & 5.33 & \dots & 9.21 \end{bmatrix}$$

Positional Encoding

27



Positional Encoding

- 各単語の位置情報を埋め込む
 - モデル内で順序を考慮した学習が可能になる
1. 位置情報~~行列~~ P を計算
 2. 入力ของEmbeddingと加算

$$P_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

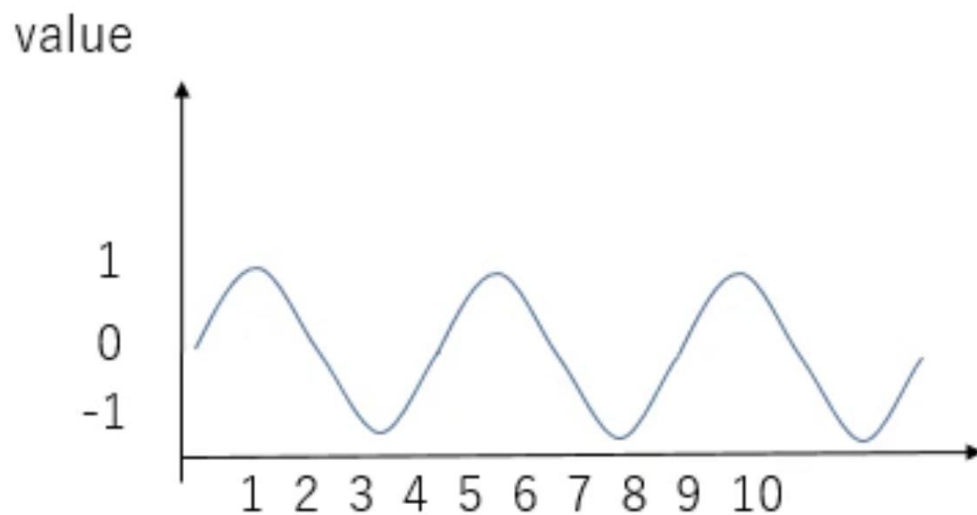
d_{model} : 総次元数
 pos : 文字の位置
 i : 次元

位置情報行列 P の詳細

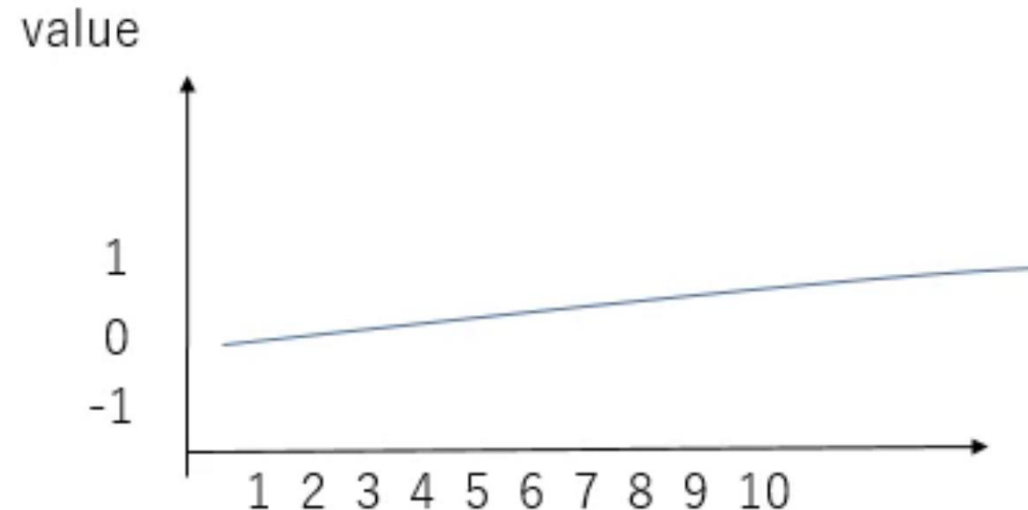
- 三角関数を用いる理由
 - 埋め込む値を $-1 \sim 1$ の範囲に制限可能

$$P = \sin \left(\frac{pos}{10000^{2i/d_{model}}} \right)$$

- 10,000とは
 - 三角関数は周期関数なので同じ値が返ることがある
 - 同じ値が現れないほど低い周波数を与える目的



通常の周波数



低い周波数

位置情報行列の形

- 位置情報行列 P は以下の行列のようになる (イメージ)
 - \sin , \cos が交互に出現

$$P_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad , \quad P_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

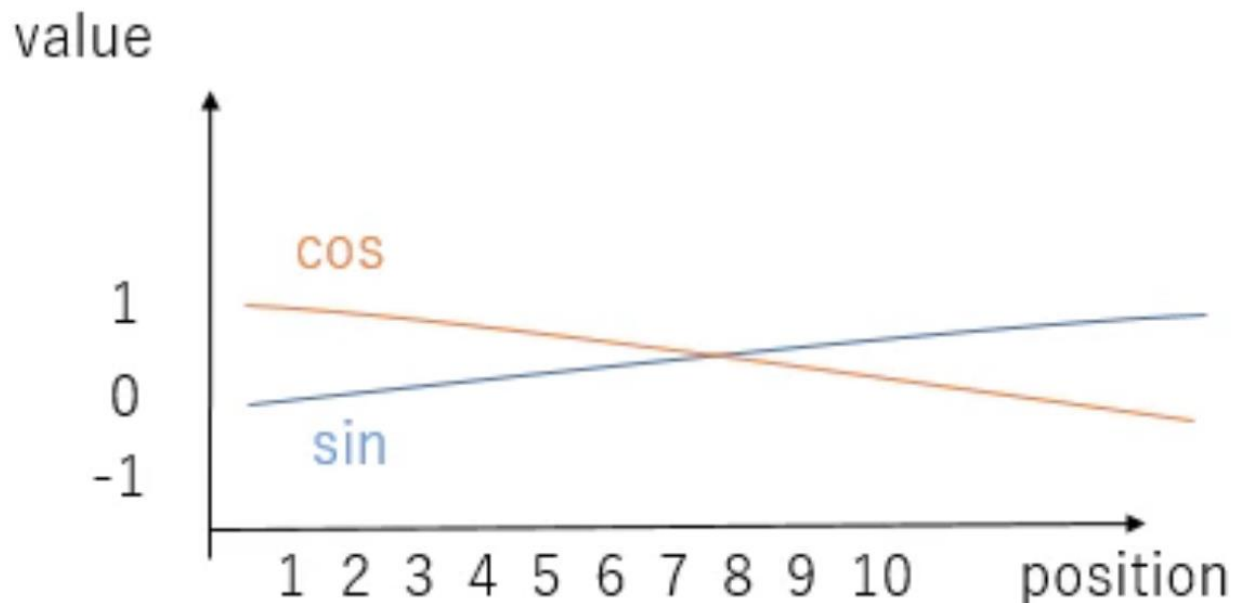
$$P = \begin{matrix} & I \\ & am \\ & good \\ \begin{bmatrix} \sin\left(\frac{0}{10000^{0/4}}\right) & \cos\left(\frac{0}{10000^{0/4}}\right) & \sin\left(\frac{0}{10000^{2/4}}\right) & \cos\left(\frac{0}{10000^{2/4}}\right) \\ \sin\left(\frac{1}{10000^{0/4}}\right) & \cos\left(\frac{1}{10000^{0/4}}\right) & \sin\left(\frac{1}{10000^{2/4}}\right) & \cos\left(\frac{1}{10000^{2/4}}\right) \\ \sin\left(\frac{2}{10000^{0/4}}\right) & \cos\left(\frac{2}{10000^{0/4}}\right) & \sin\left(\frac{2}{10000^{2/4}}\right) & \cos\left(\frac{2}{10000^{2/4}}\right) \end{bmatrix} \end{matrix}$$

sin , cos 両方用いる理由

$$P_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) , \quad P_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

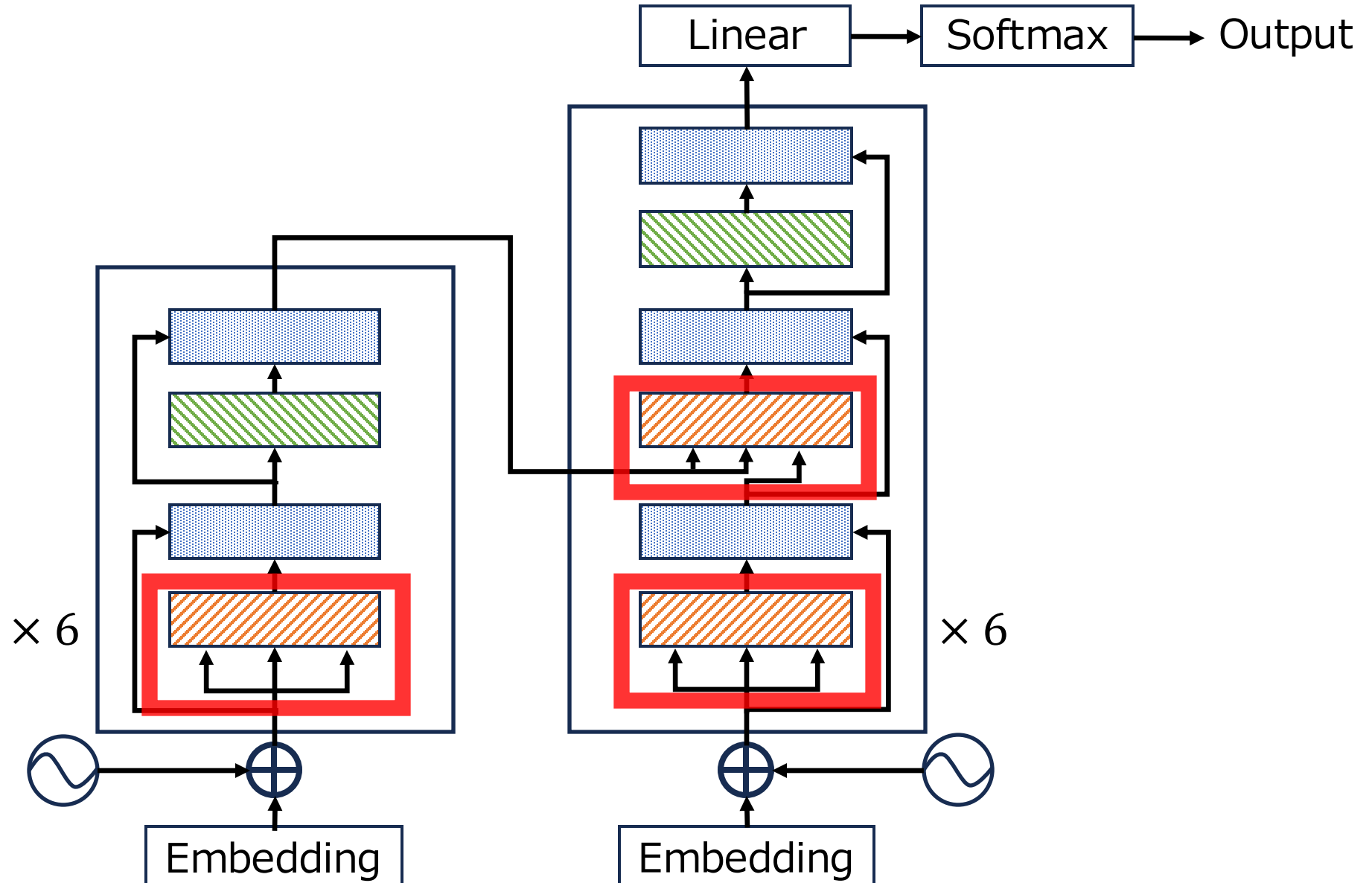
- 低い周波数の三角関数では値の変化が小さい
- 表現力を大きくするために両方使用

d_{model} : 総次元数
 pos : 文字の位置
 i : 次元



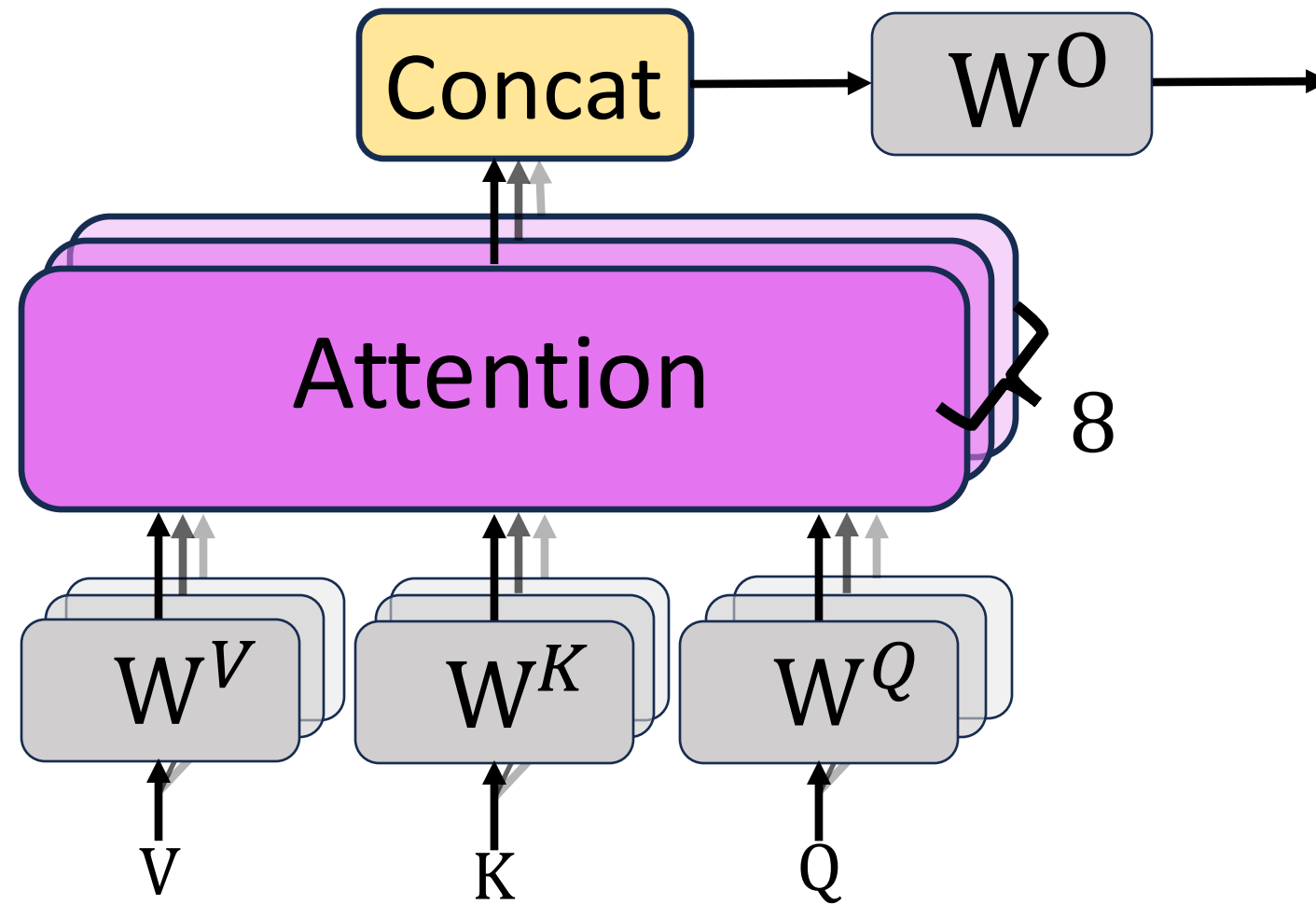
- 前処理
- Multi-Head Attention
 - Multi-Head Attention 計算
 - Single Headとの違い
 - Self Attention とは
- フィードフォワードネットワーク

Multi Head Attention



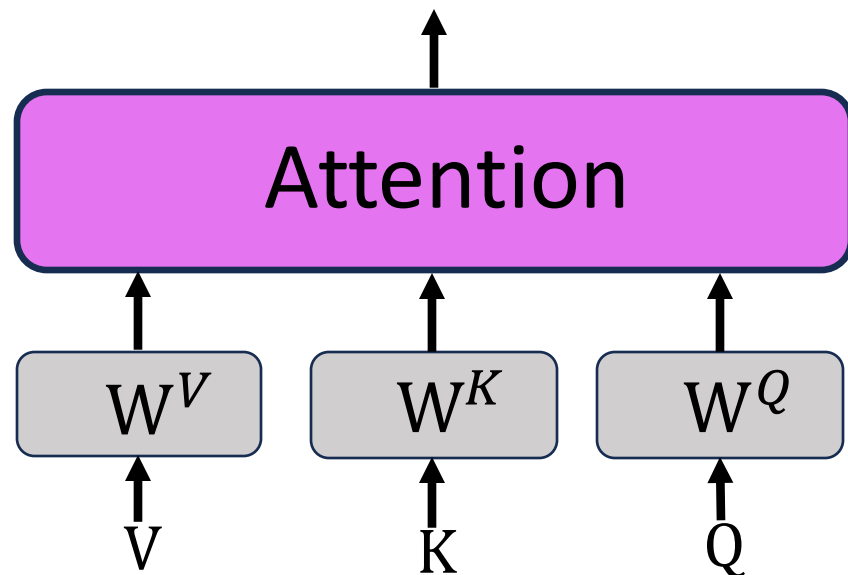
Multi Head Attention

- 複数のAttentionを並行して使用する手法
 - 異なる部分的な情報や関係性を同時に学習可能

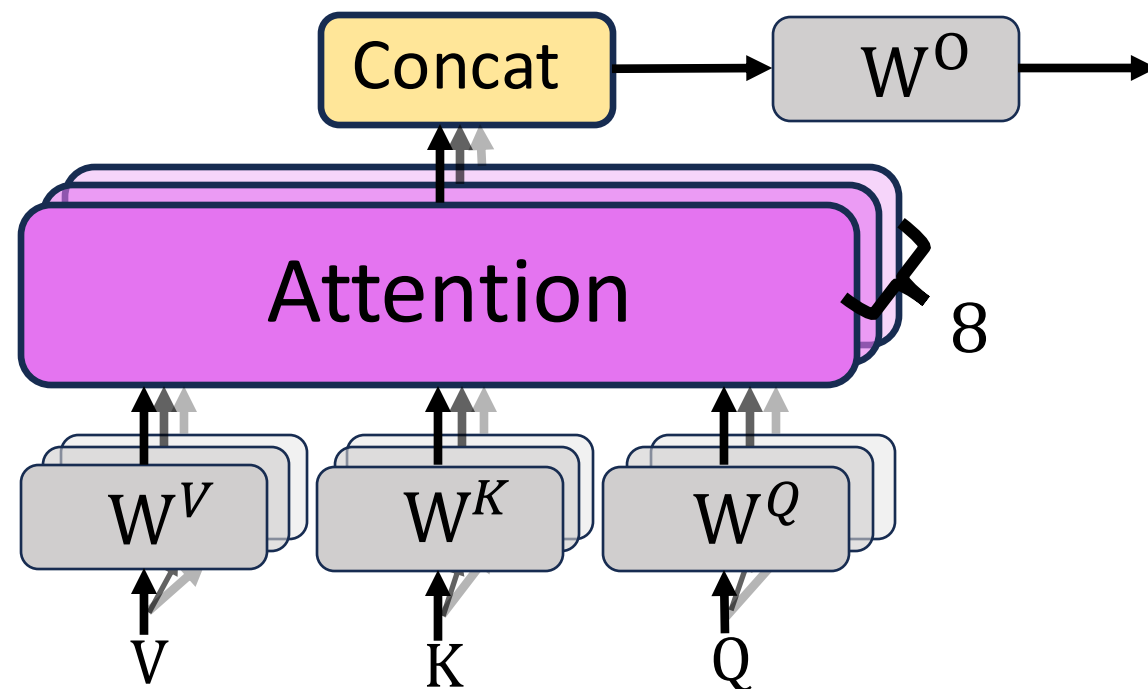


Single と Multi の違い

- Single : 主に一つの部分にしか注目できない
- Multi : 複数箇所に注目可能



Single Head



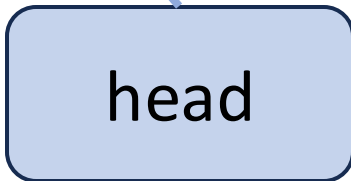
Multi Head

Single と Multi の違い

- Single : 主に一つの部分にしか注目できない
- Multi : 複数箇所に注目可能

比較対象のトークン
A big black **cat** drank the water .

注目



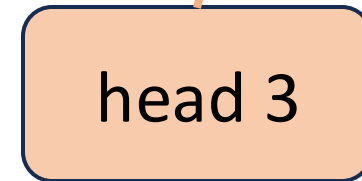
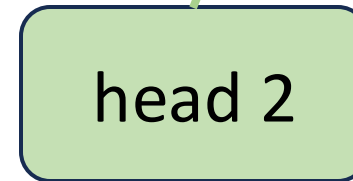
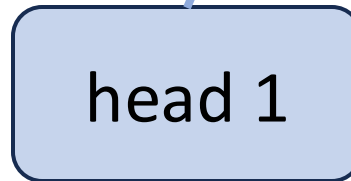
Single Head

比較対象のトークン
A big black **cat** drank the water .

注目 1

注目 2

注目 3



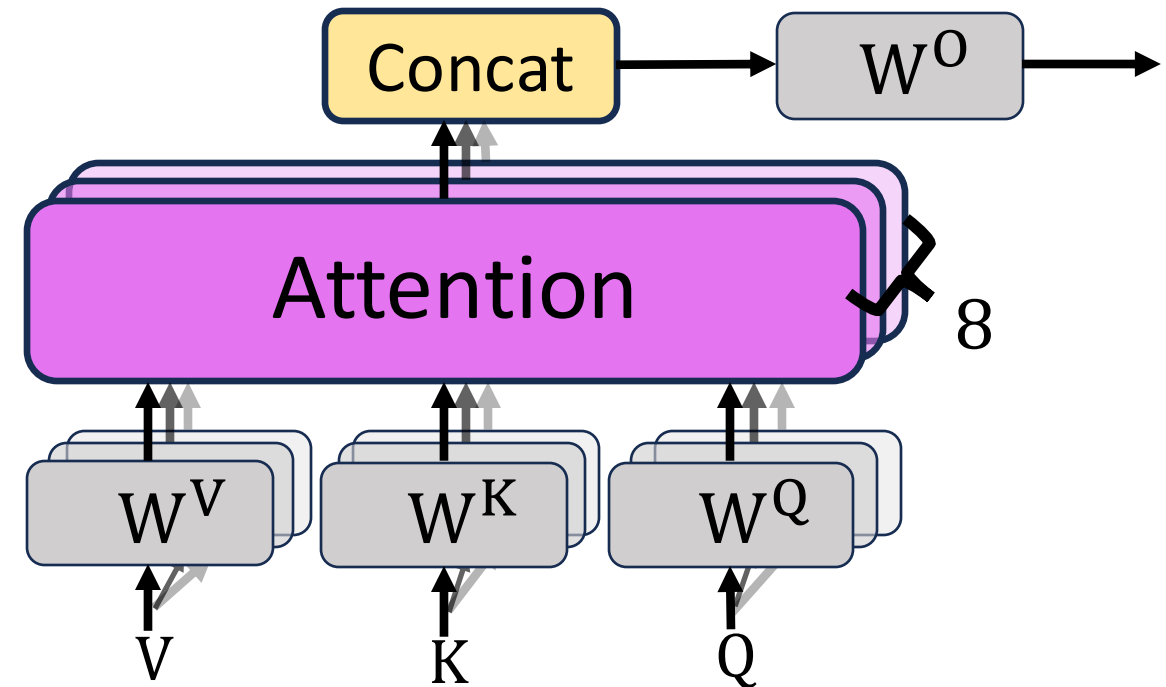
Multi Head

Multi Head Attention 計算

$$MultiHead(Q, K, V) = Concat (head_1, \dots, head_8) W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

- 本論文では8つのheadを使用



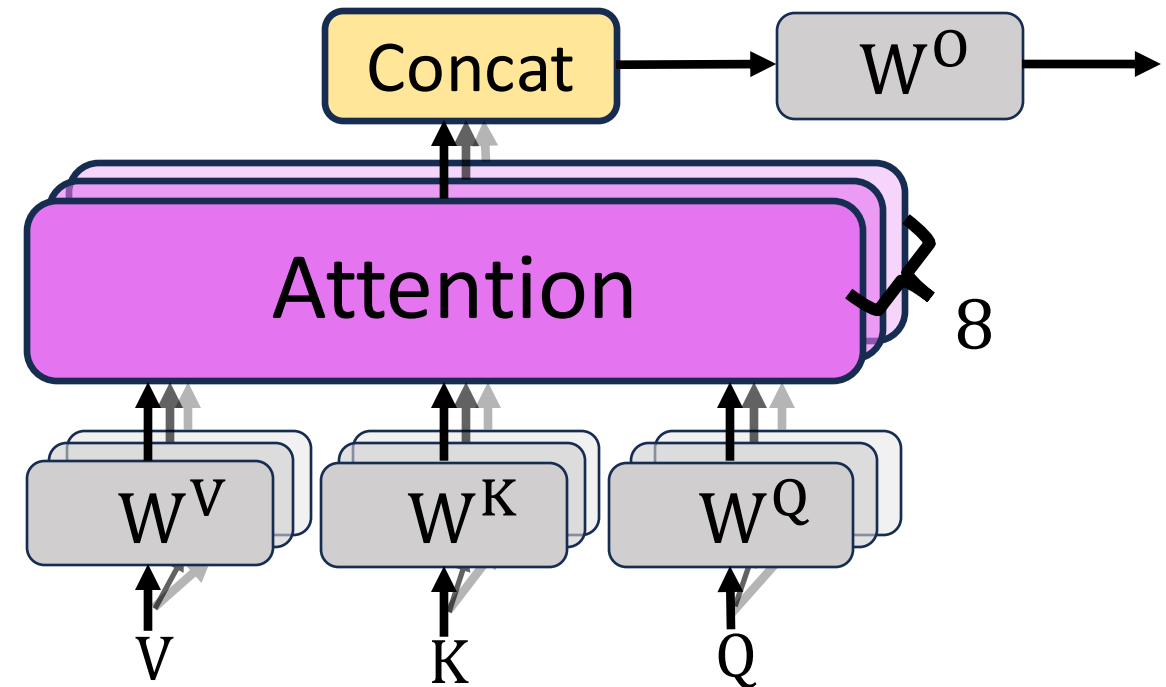
W : 学習可能なパラメータ行列
 i : ヘッドの割り当て番号

Multi Head Attention 計算

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_8) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Q, K, V を $head$ の数に分割
- $head$ 毎に Attention をとる
- $head$ 毎の Attention を連結
- 連結した行列を線形変換

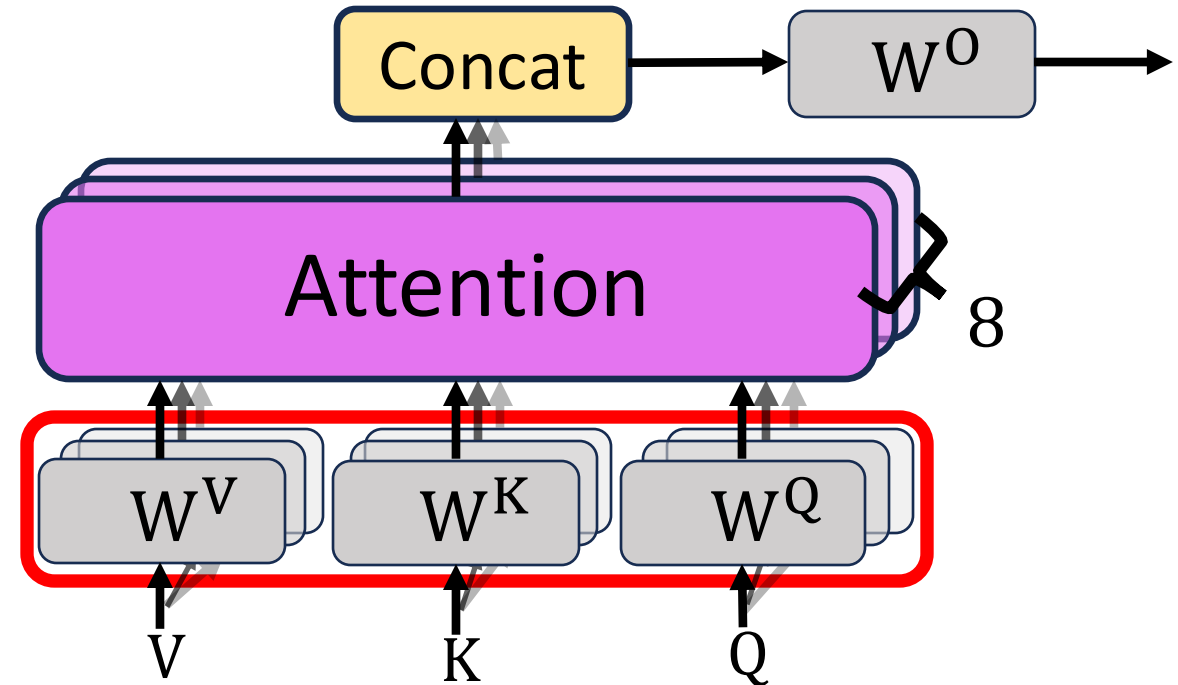


Multi Head Attention 計算

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_8) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Q, K, V を $head$ の数に分割
- $head$ 毎に Attention をとる
- $head$ 毎の Attention を連結
- 連結した行列を線形変換

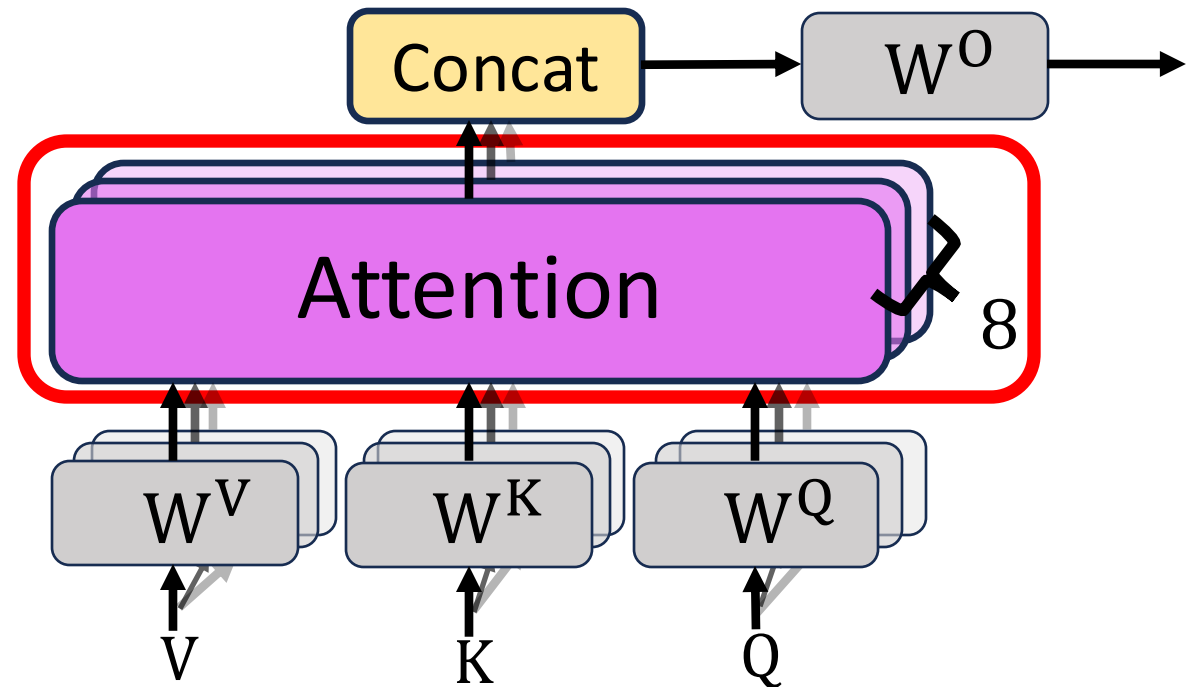


Multi Head Attention 計算

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_8) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Q, K, V を $head$ の数に分割
- $head$ 毎に Attention をとる
- $head$ 毎の Attention を連結
- 連結した行列を線形変換

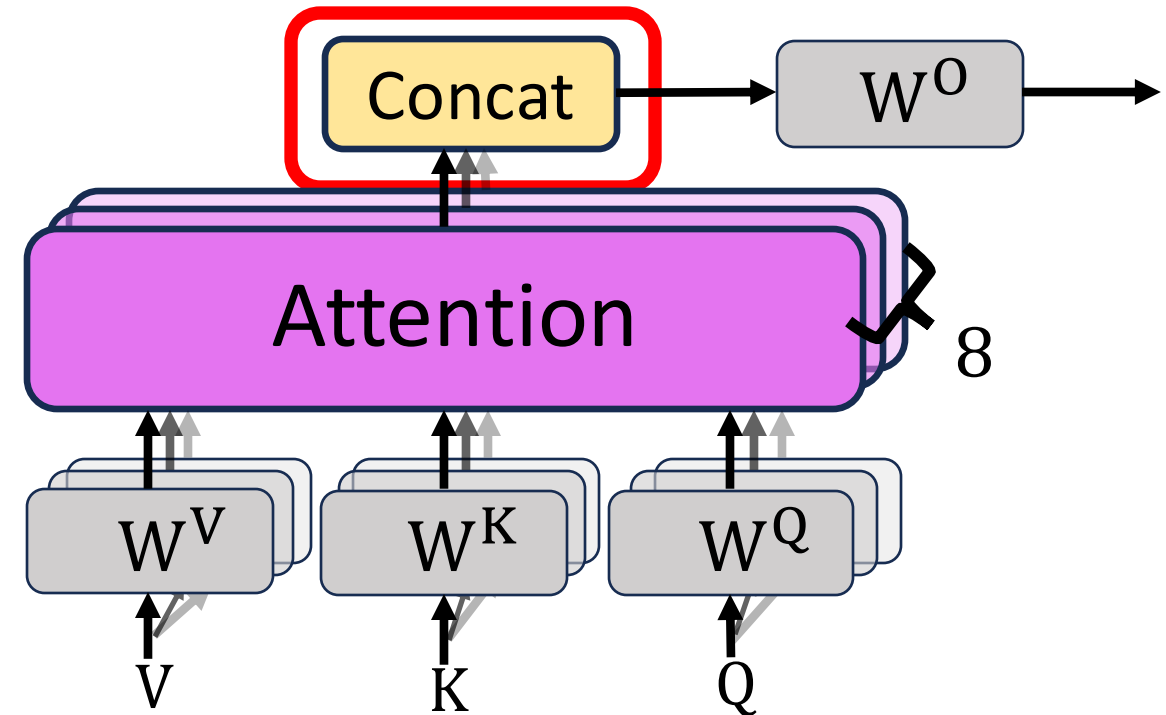


Multi Head Attention 計算

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_8) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Q, K, V を $head$ の数に分割
- $head$ 毎に Attention をとる
- $head$ 毎の Attention を連結
- 連結した行列を線形変換

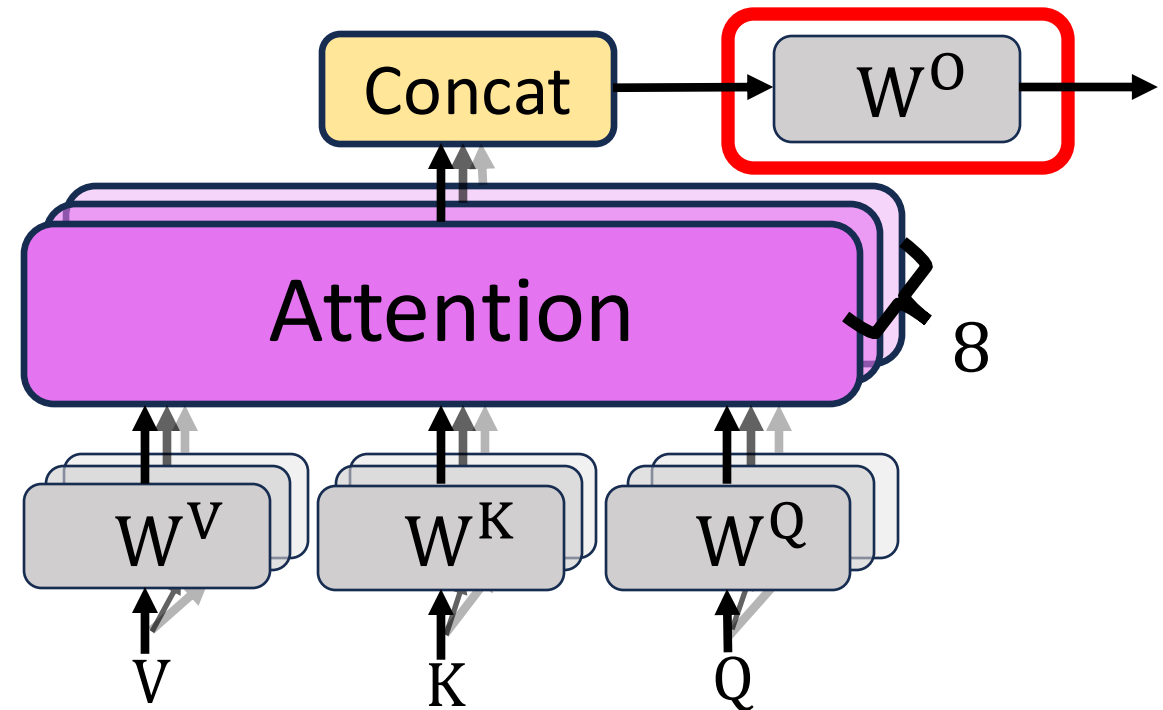


Multi Head Attention 計算

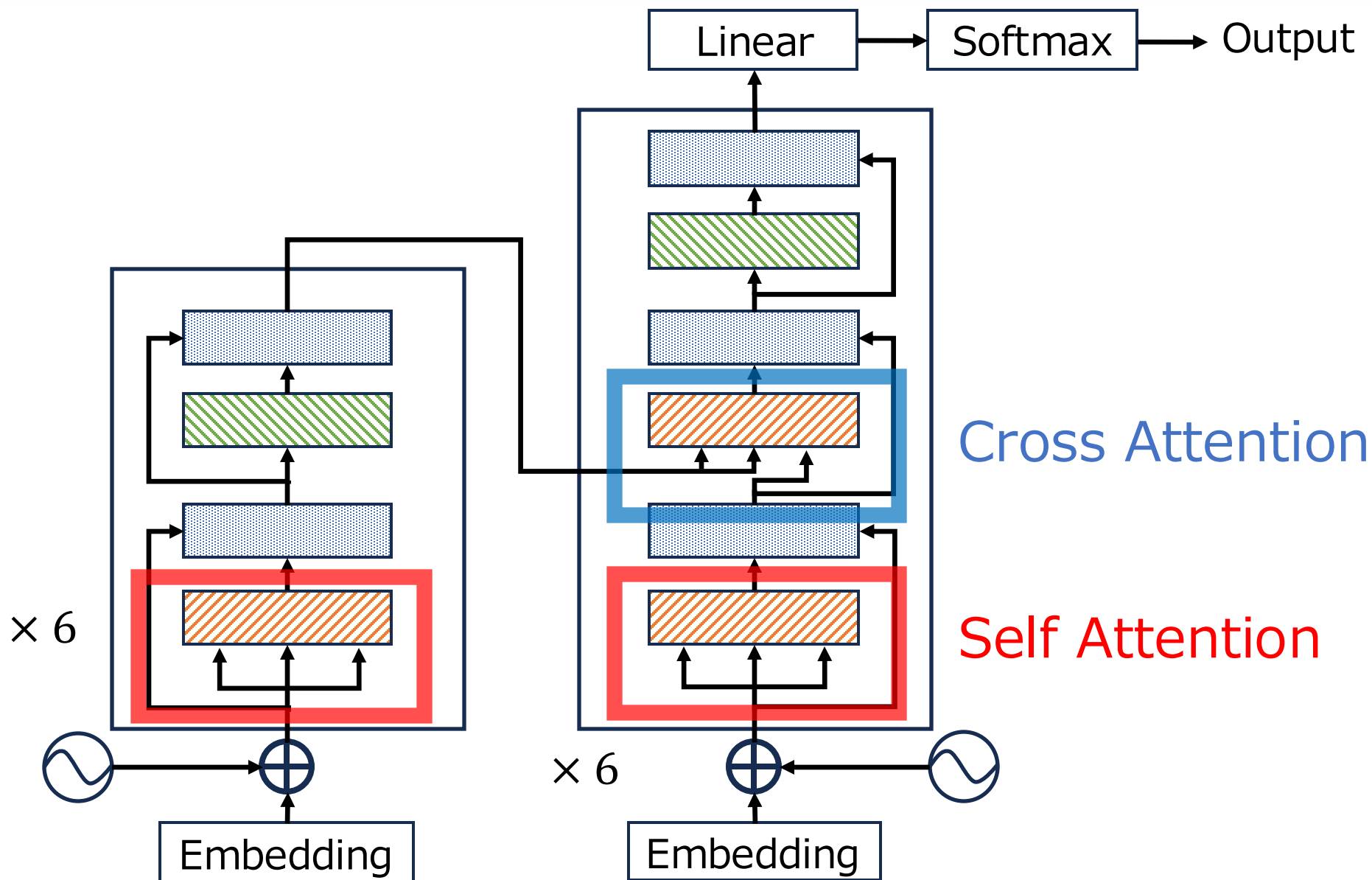
$$\text{MultiHead}(Q, K, V) = \text{Concat} (\text{head}_1, \dots, \text{head}_8) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Q, K, V を $head$ の数に分割
- $head$ 毎に Attention をとる
- $head$ 毎の Attention を連結
- 連結した行列を線形変換



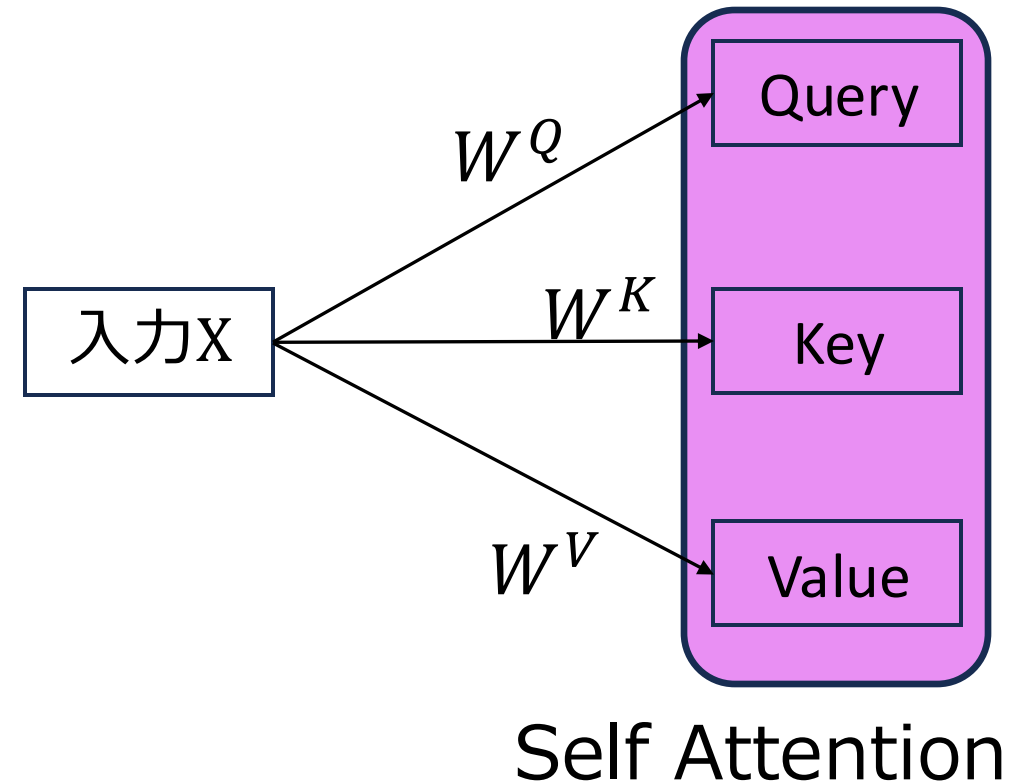
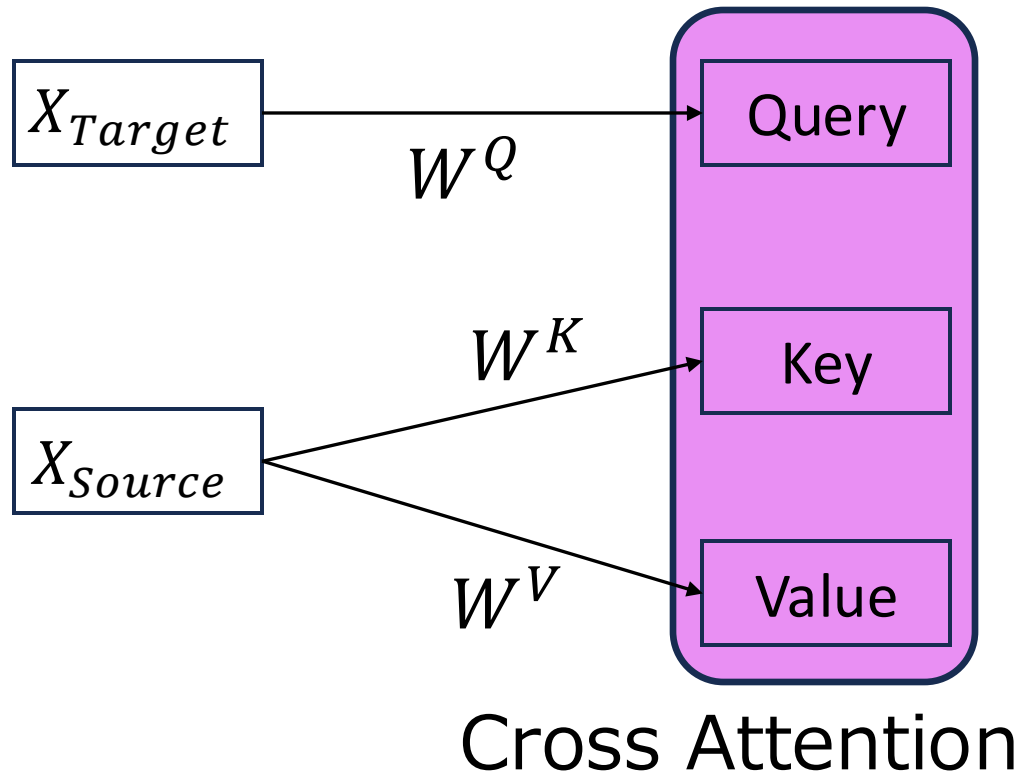
Transformerの全体像



Cross と Self の違い

- Q,K,Vのマッピング

- Cross : X_{Target} を Q に, X_{Source} を K,V にマッピング
- Self : 1つのデータでQ,K,Vにマッピング



Cross , Self Attention (機械翻訳の例)

- Cross Attention

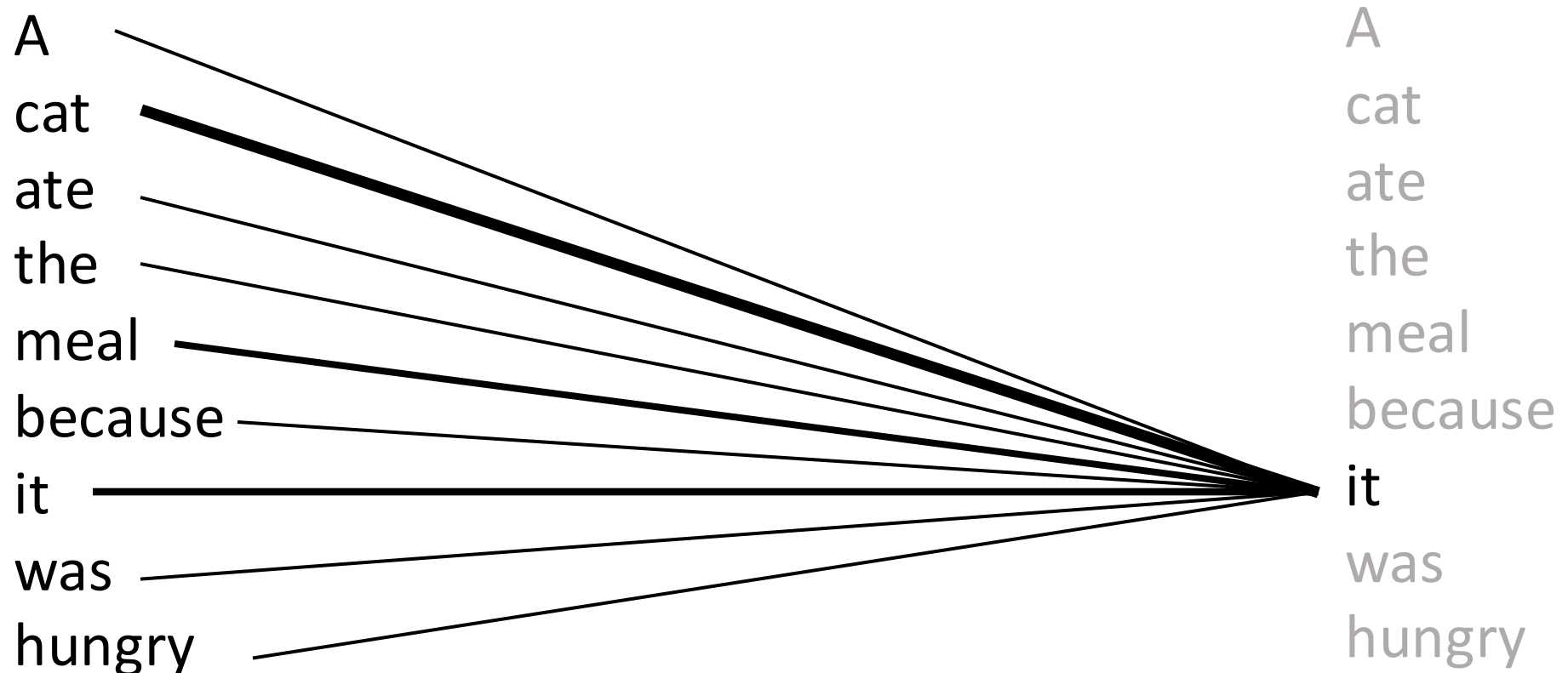
- 異なるデータ間でAttentionをとる手法
- 【例】 X_{Target} : This is a pen (元の文)
 X_{Source} : これは ペン です (翻訳文)

- Self Attention

- 自分自身のデータ同士でAttentionをとる手法
- 機械翻訳では文章中のトークン同士の関係性を計算
- 【例】 A cat ate the meal because it was hungry
という文章中の「it」が何を指すかを探索

Self Attention (機械翻訳の例)

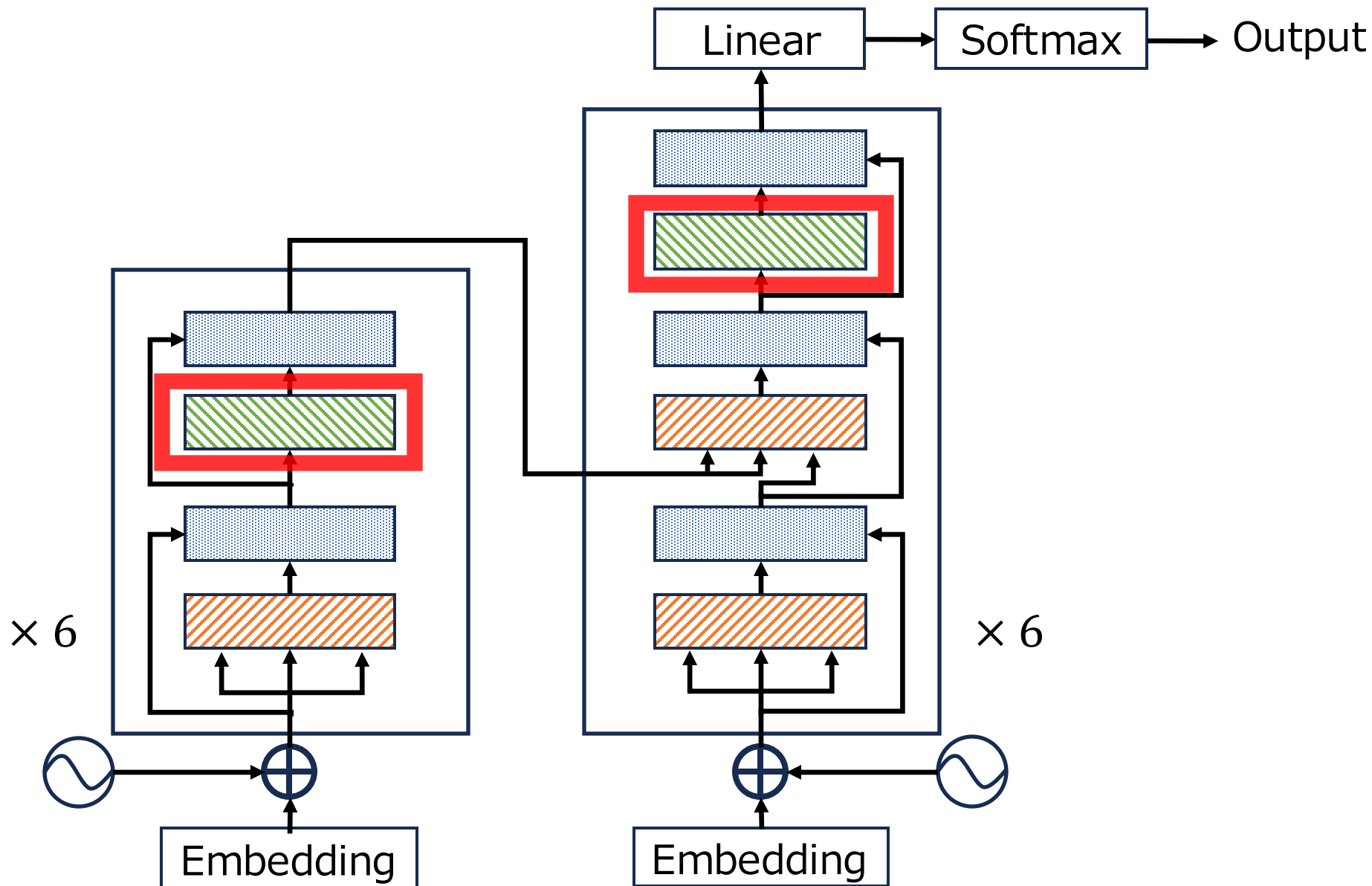
- 【例】 A cat ate the meal because it was hungry
という文章中の「it」が何を指すかを探索



機構

- 前処理
- Multi-Head Attention
- フィードフォワードネットワーク

Transformerの全体像



フィードフォワードネットワーク

$$FFNN(x) = \text{MAX}(0, xW_1 + b_1)W_2 + b_2$$

- 全結合(512次元から2048次元へ)
- 活性化関数(ReLU)を適用
- 全結合(2048次元から512次元へ)

フィードフォワードネットワーク

$$FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$$

- 全結合(512次元から2048次元へ)
- 活性化関数(ReLU)を適用
- 全結合(2048次元から512次元へ)

フィードフォワードネットワーク

$$FFNN(x) = \boxed{MAX(0, xW_1 + b_1)}W_2 + b_2$$

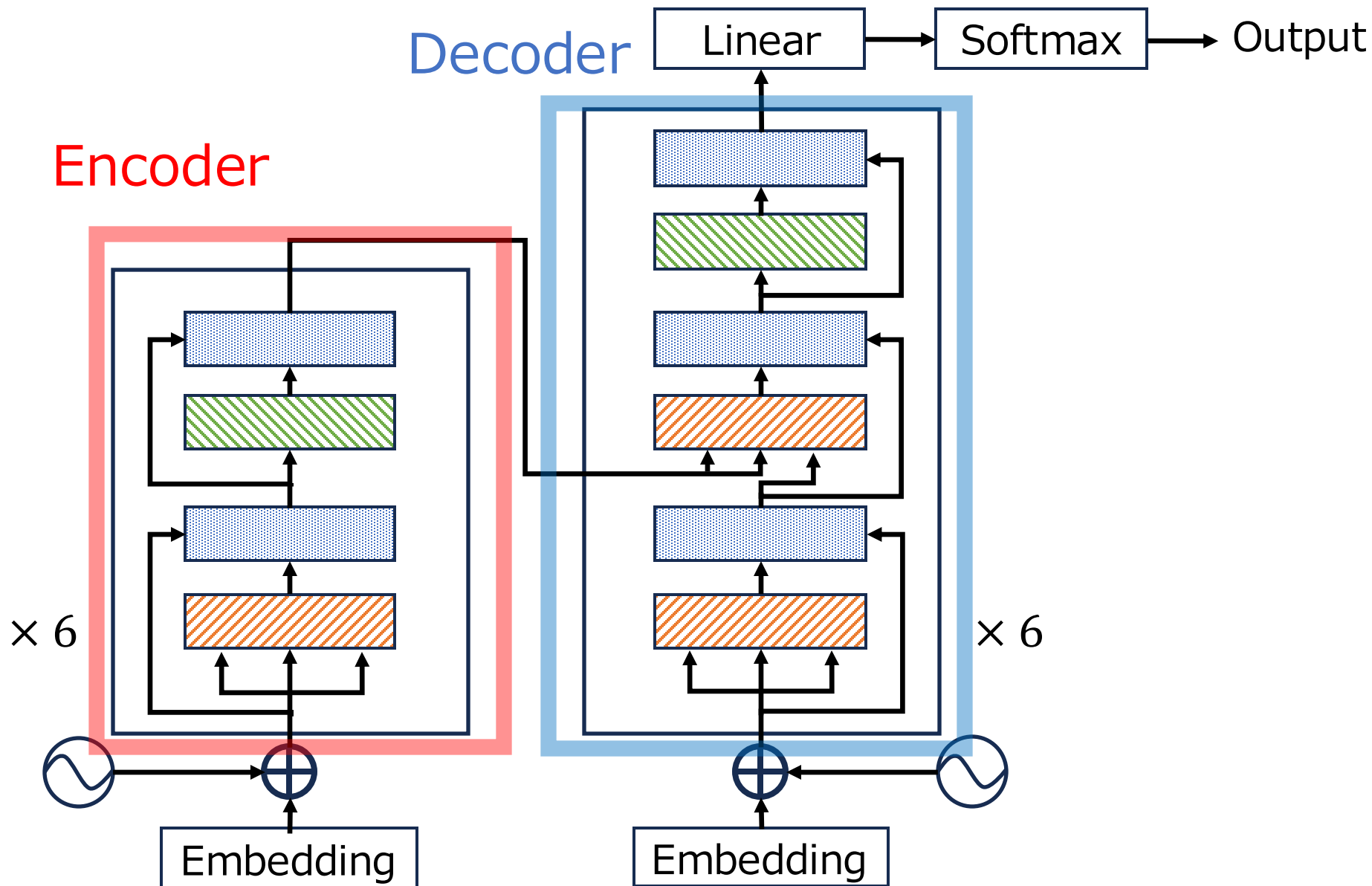
- 全結合(512次元から2048次元へ)
- 活性化関数(ReLU)を適用
- 全結合(2048次元から512次元へ)

フィードフォワードネットワーク

$$FFNN(x) = \text{MAX}(0, xW_1 + b_1)W_2 + b_2$$

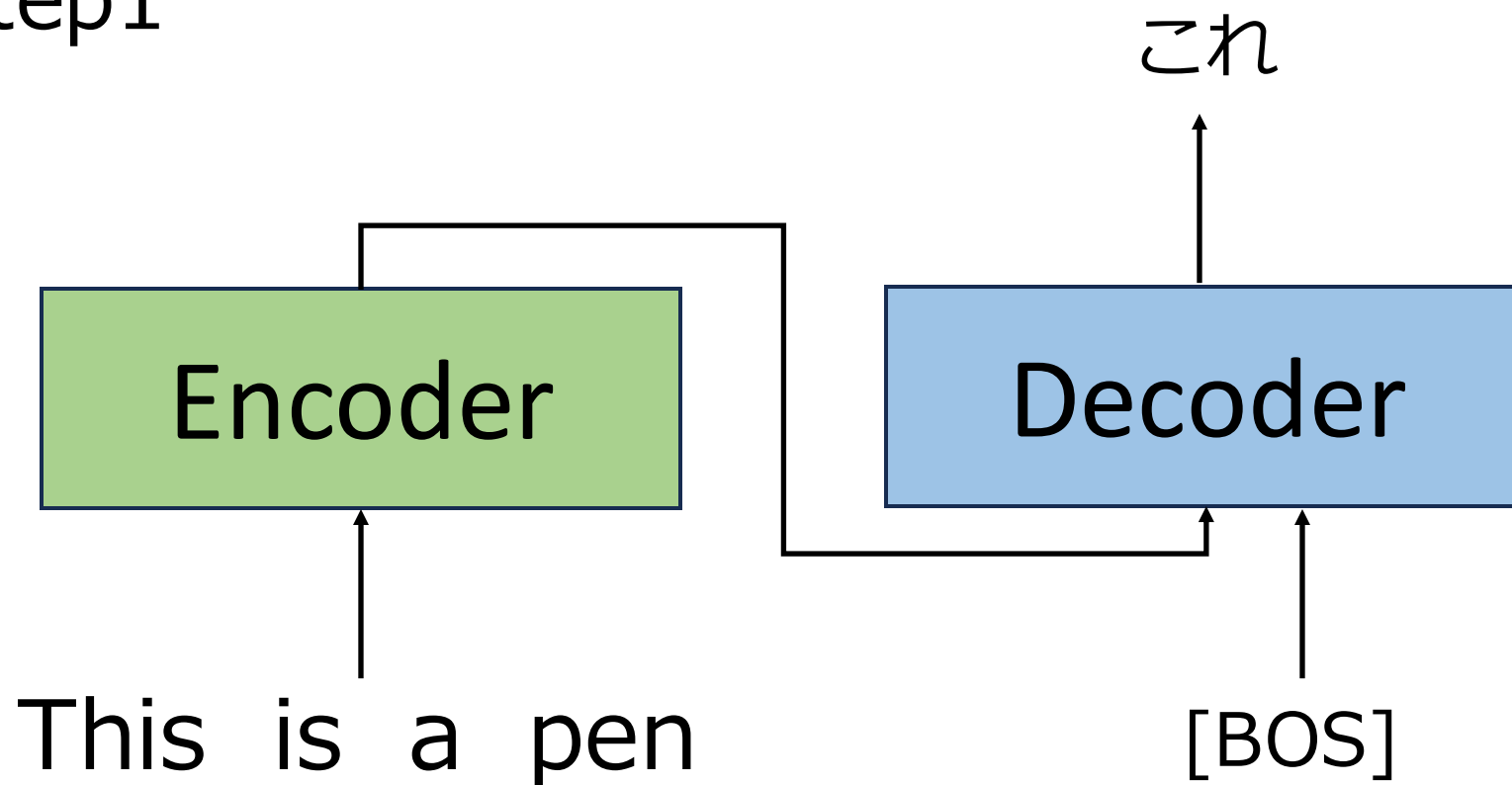
- 全結合(512次元から2048次元へ)
- 活性化関数(ReLU)を適用
- 全結合(2048次元から512次元へ)

Transformerの全体像



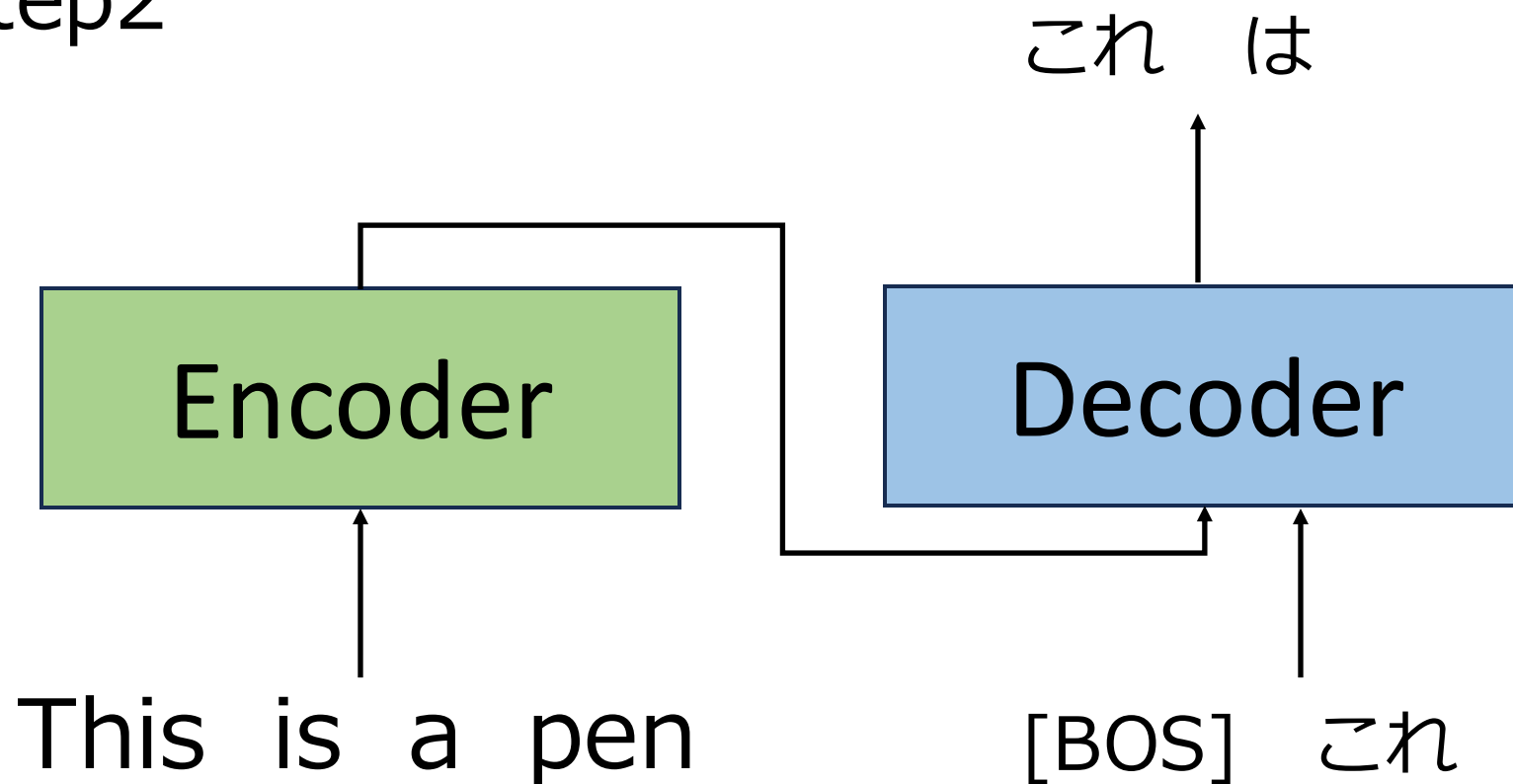
- 順次予測をしていく

step1



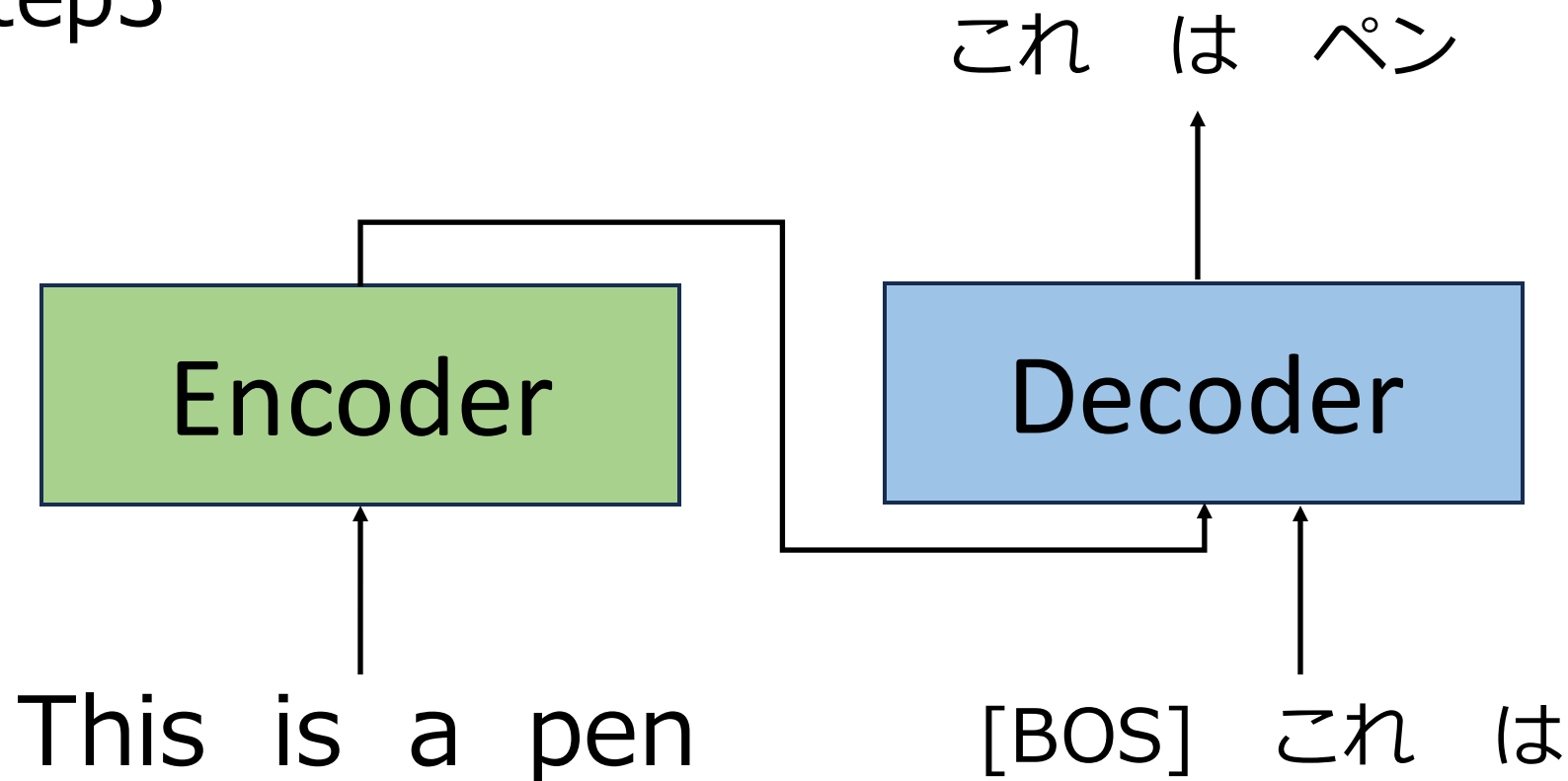
- 順次予測をしていく

step2



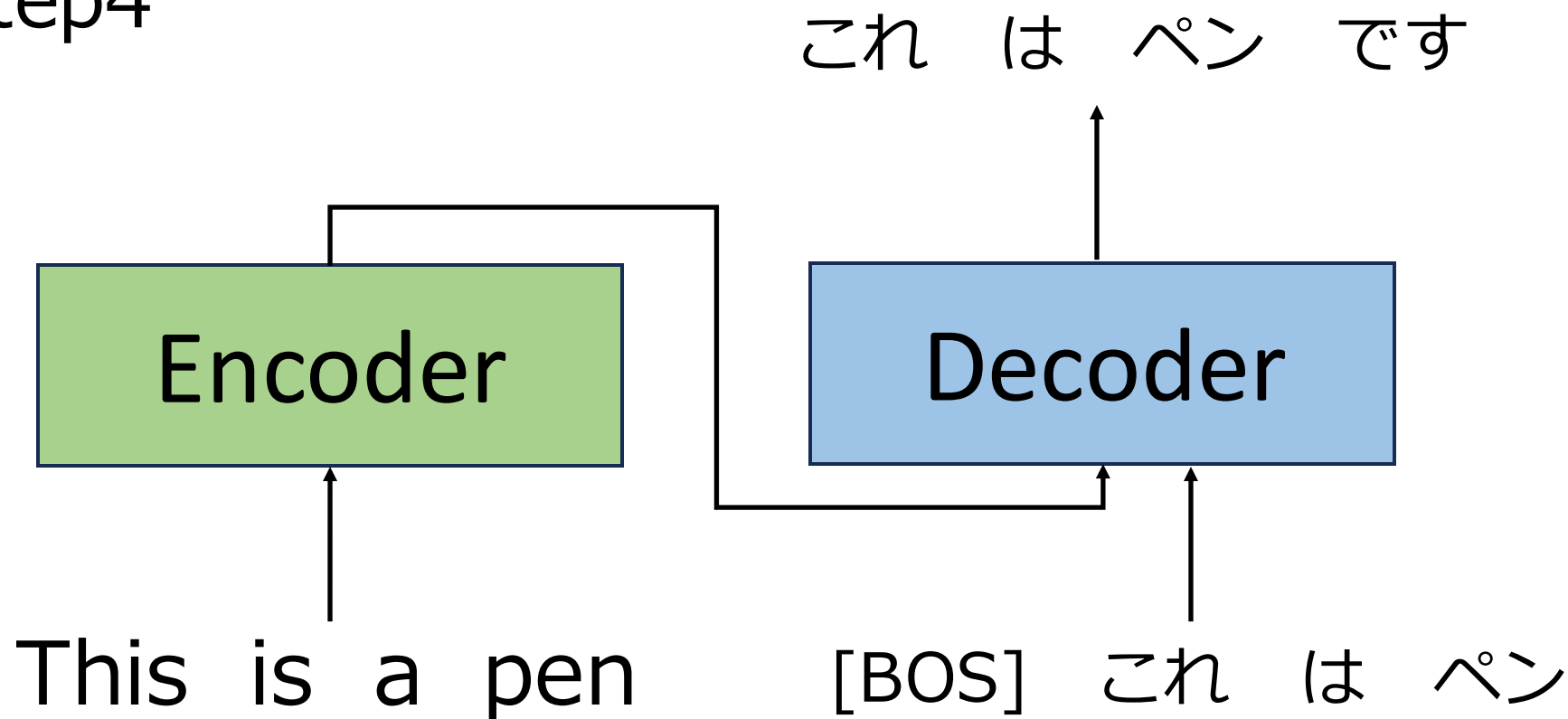
- 順次予測をしていく

step3



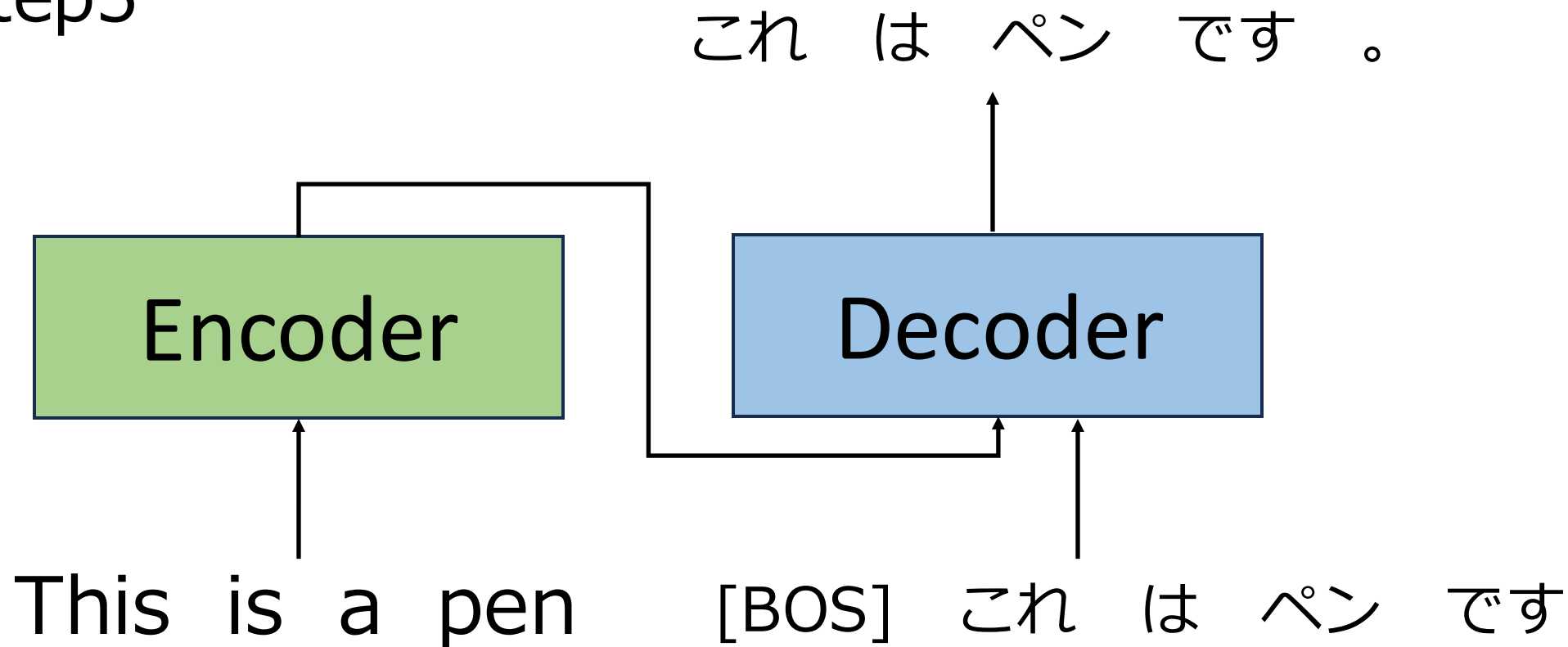
- 順次予測をしていく

step4



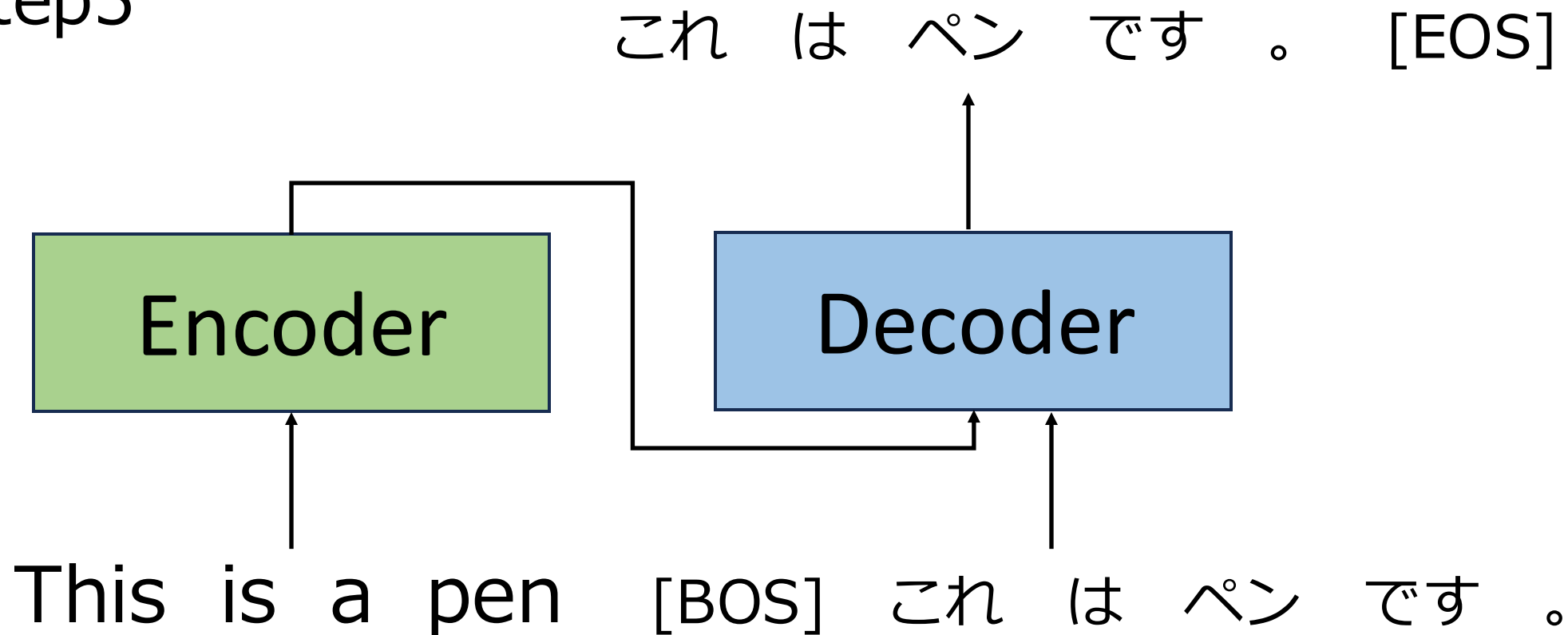
- 順次予測をしていく

step5



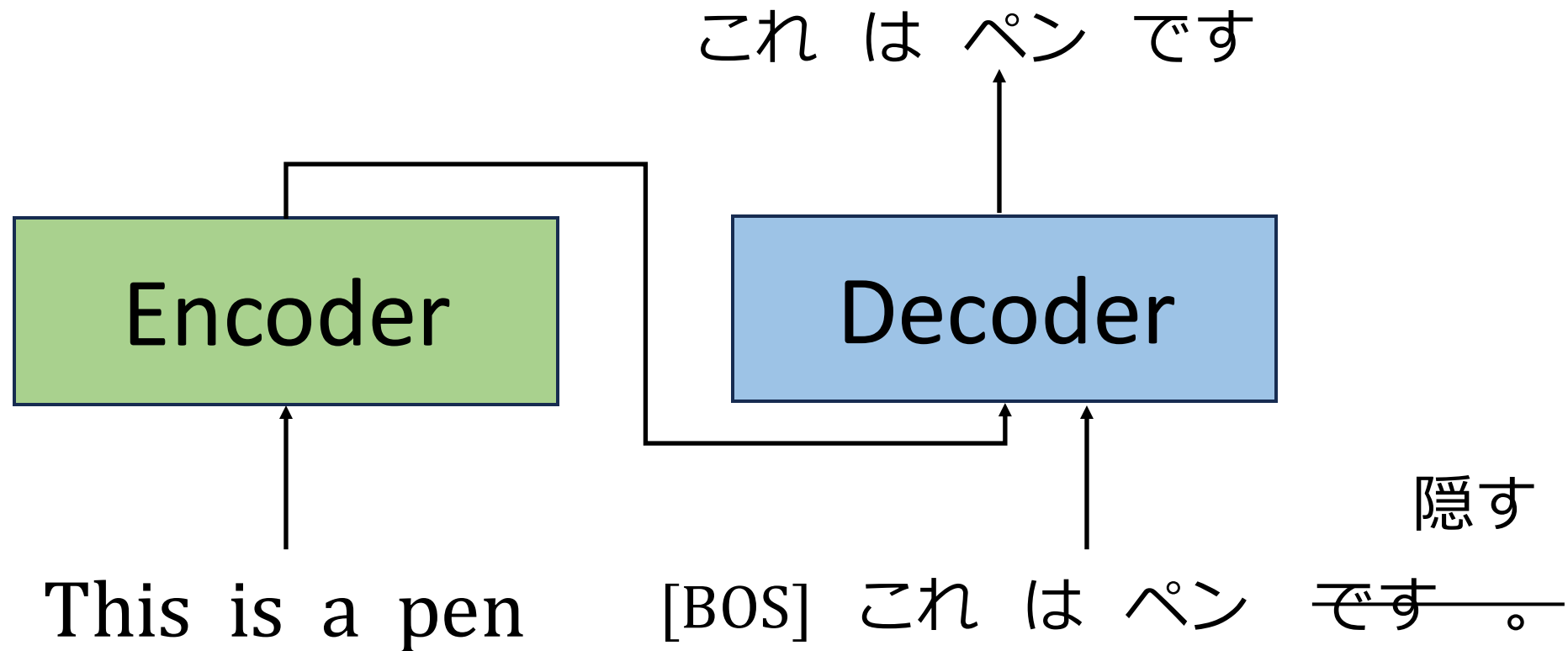
- 順次予測をしていく

step5

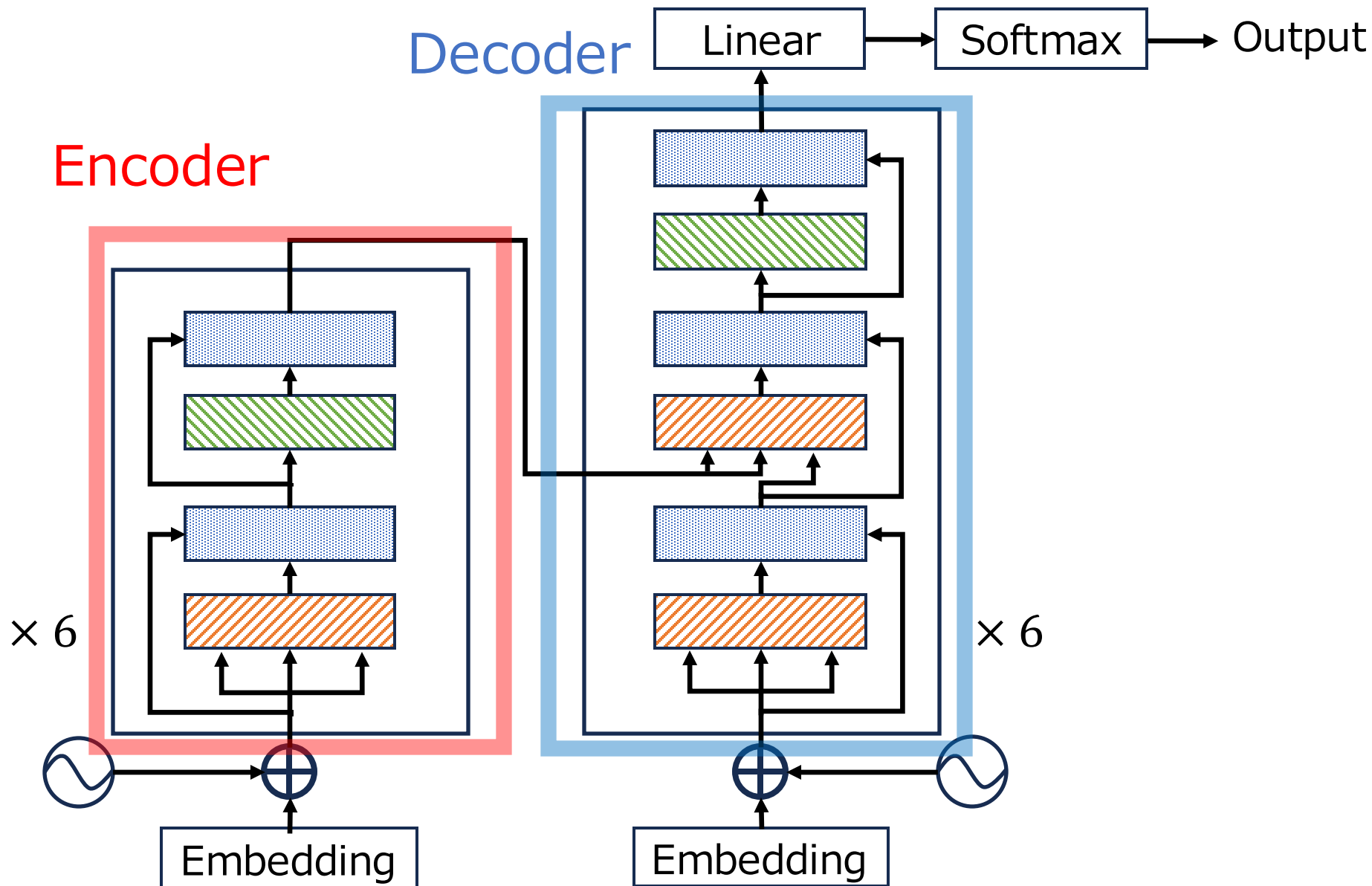


学習時

- 全て同時に学習する
- 学習するときは未来の情報は隠す
 - 【例】「です」を学習する場合「です」「。」を隠す

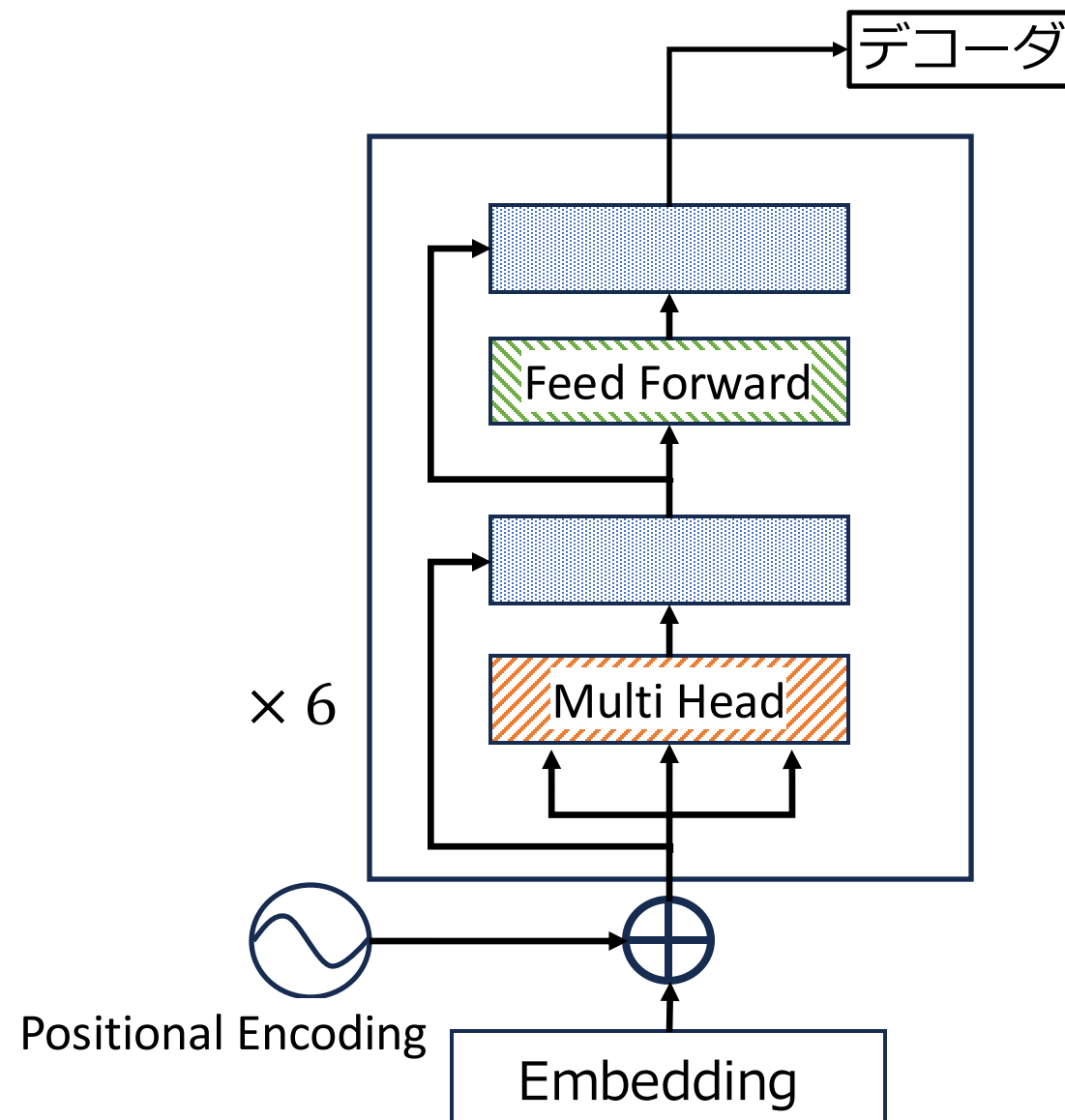


Transformerの全体像



エンコーダの動き

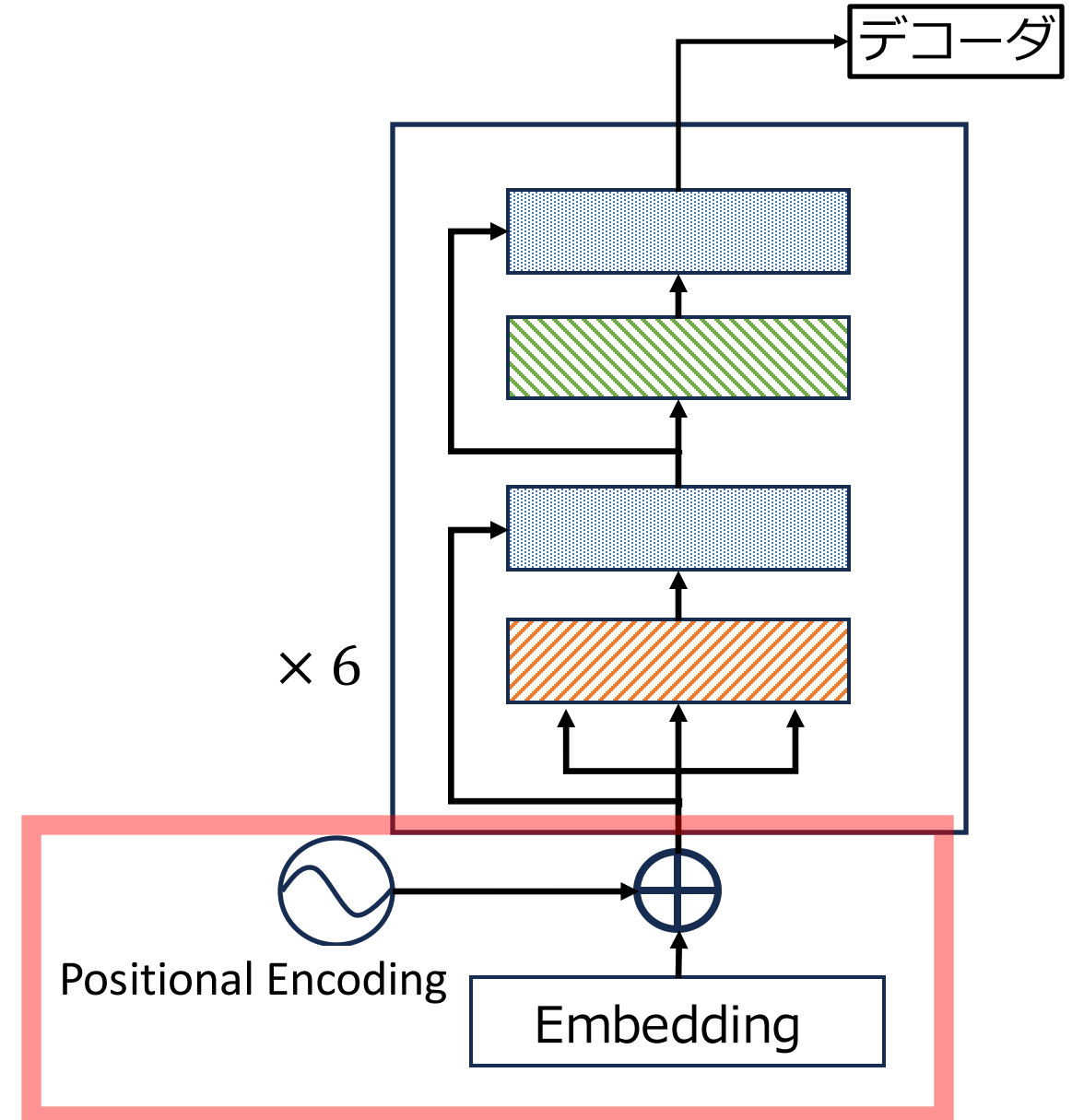
1. エンコーダ入力前の前処理
 2. Multi Head Attentionに入力
 3. Feed Forward Networkに入力
 4. 次のエンコーダに入力値として提供
- 2~4を計6回繰り返し, 最上位のエンコーダの出力値をデコーダへ



エンコーダの動き

1. エンコーダ入力前の前処理

- Embedding
 - トークンをベクトルに変換
- Positional Encoding
 - 位置情報を入力に埋め込み



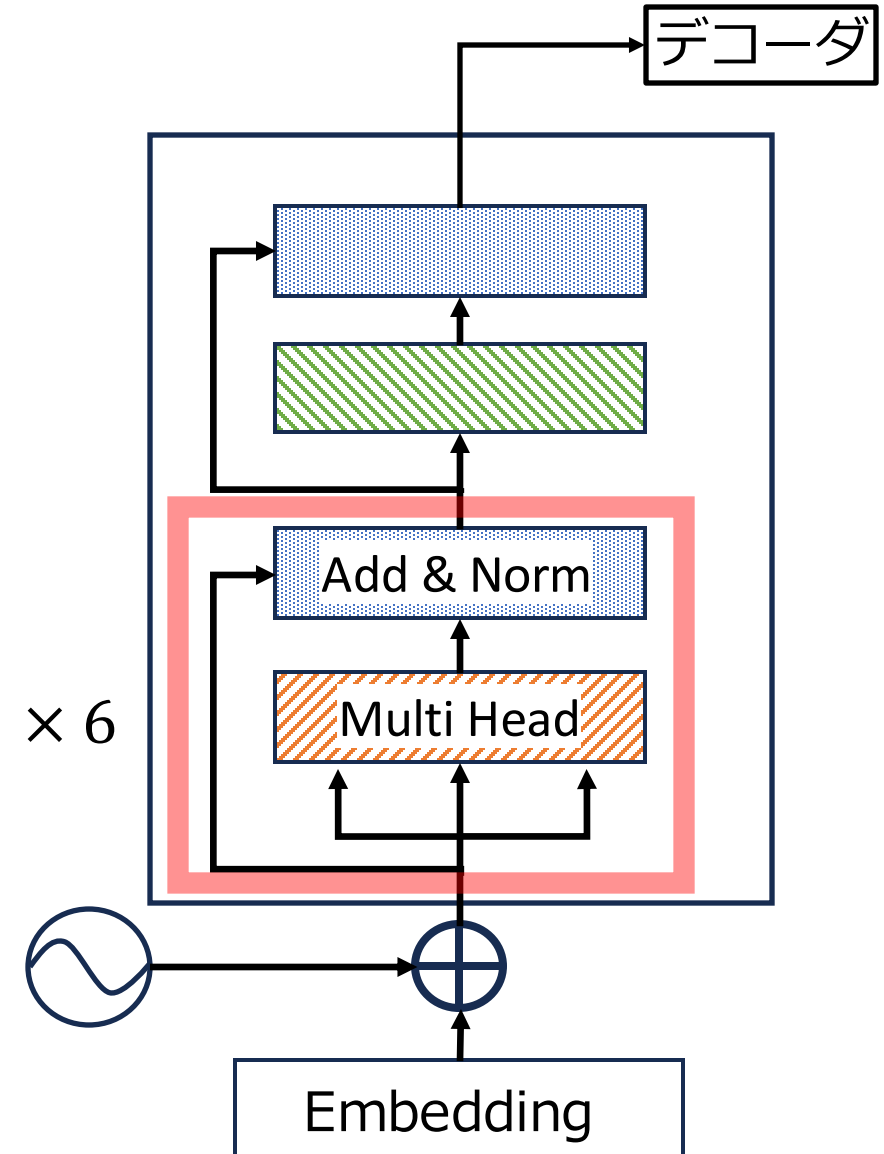
エンコーダの動き

2. Multi Head Attentionに入力

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_8) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Self Attention
 - 自身のデータ同士の間係を計算
- 残差接続 , レイヤー正規化



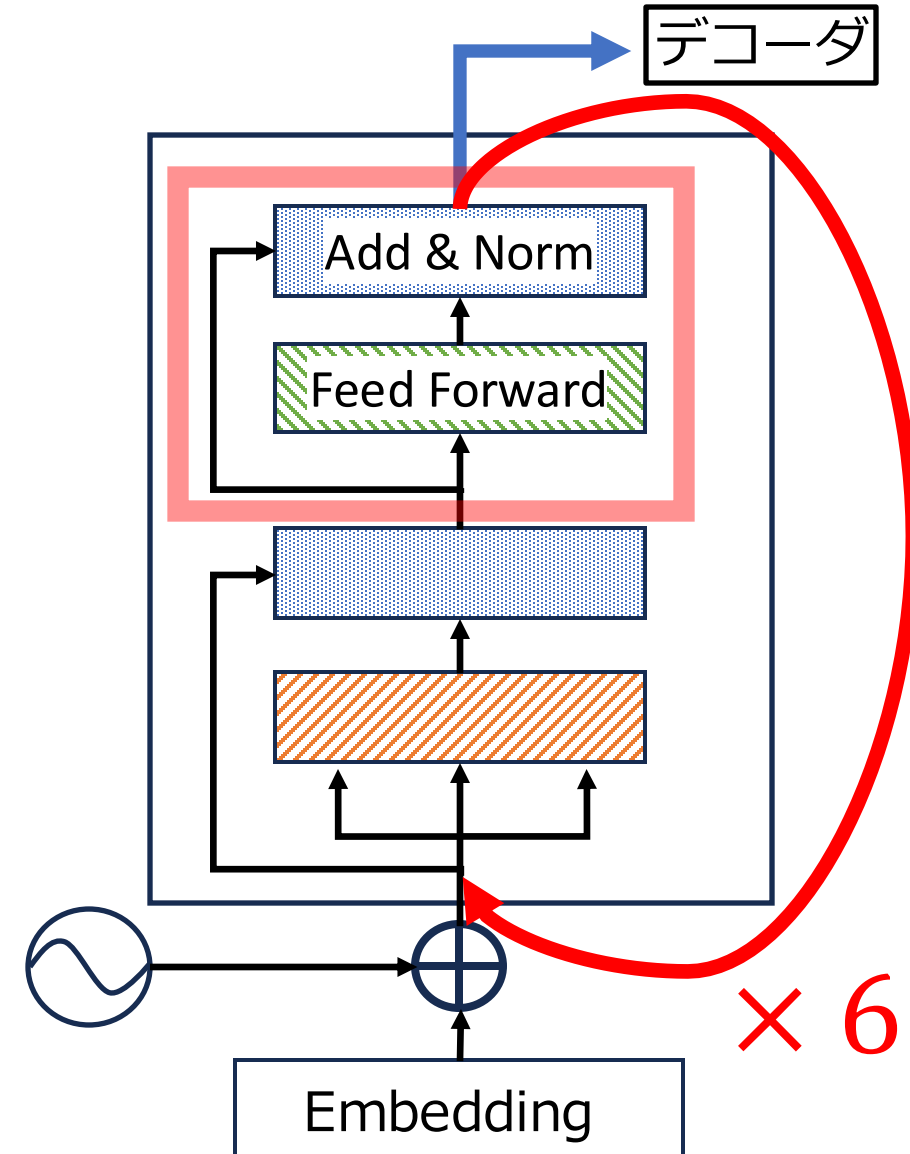
エンコーダの動き

3. Feed Forward Networkに入力

$$FFNN(x) = \text{MAX}(0, xW_1 + b_1)W_2 + b_2$$

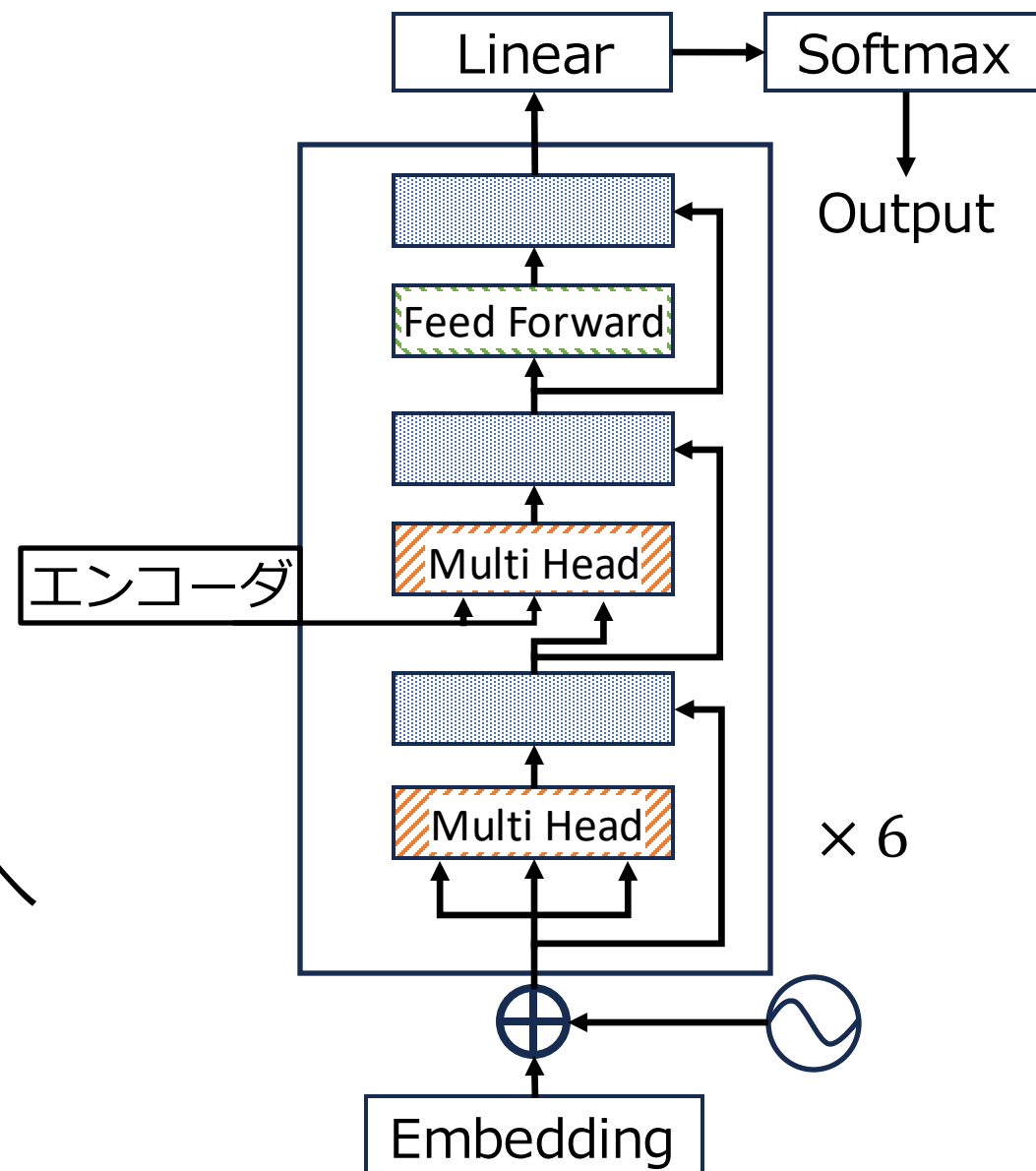
4. 次のエンコーダに入力値として

- 2～4を計6回繰り返し, 最上位のエンコーダの出力値をデコーダへ



デコーダの動き

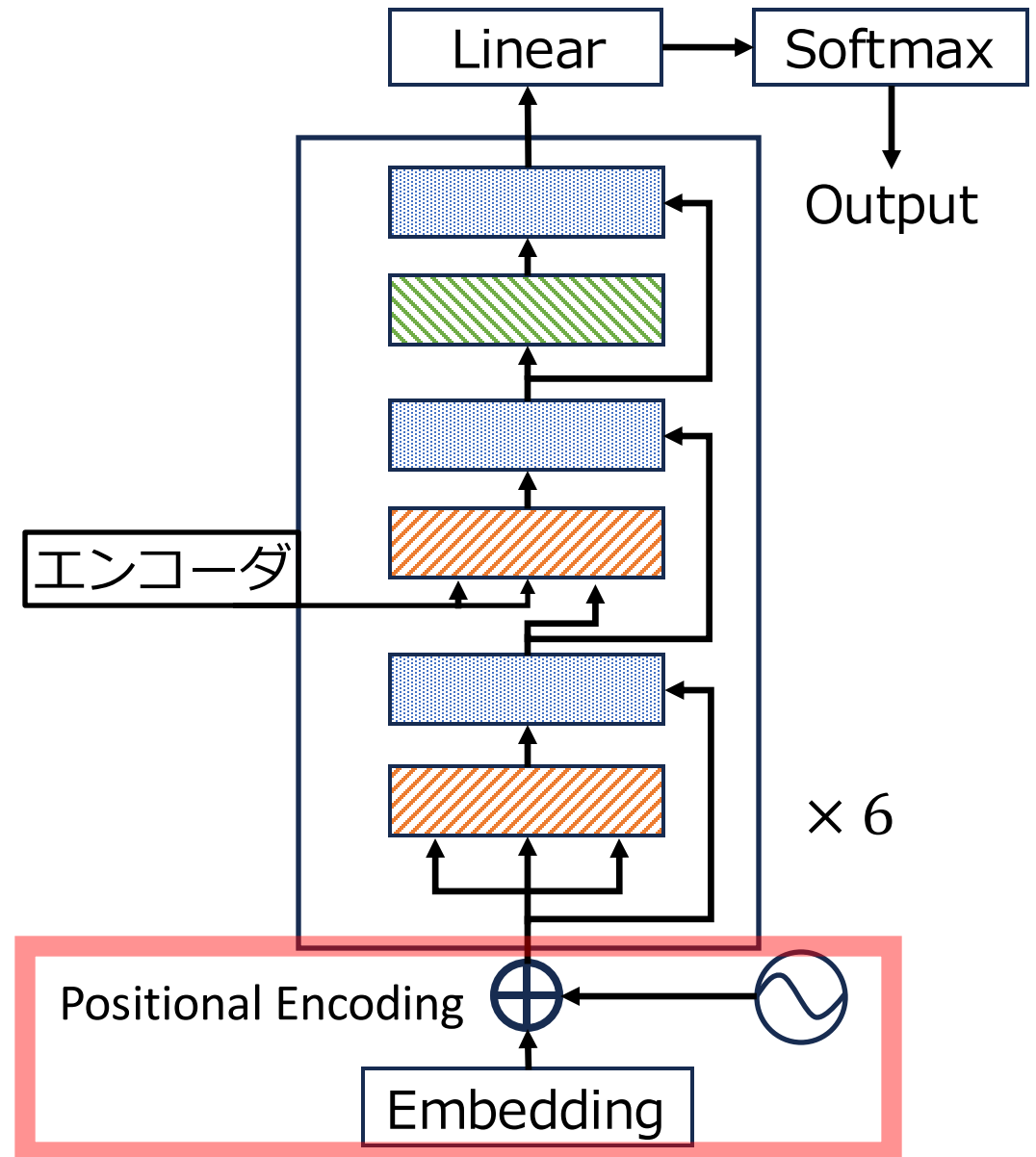
1. デコーダ入力前の前処理
 2. Multi Head Attentionに入力
 3. エンコーダからの出力も取り入れ
Multi Head Attentionに入力
 4. Feed Forward Networkに入力
 5. 次のデコーダに入力値として提供
- 2～5を計6回繰り返し, 最上位のデコーダの出力値をLinear, Softmax へ



デコーダの動き

1. デコーダ入力前の前処理

- Embedding
- Positional Encoding

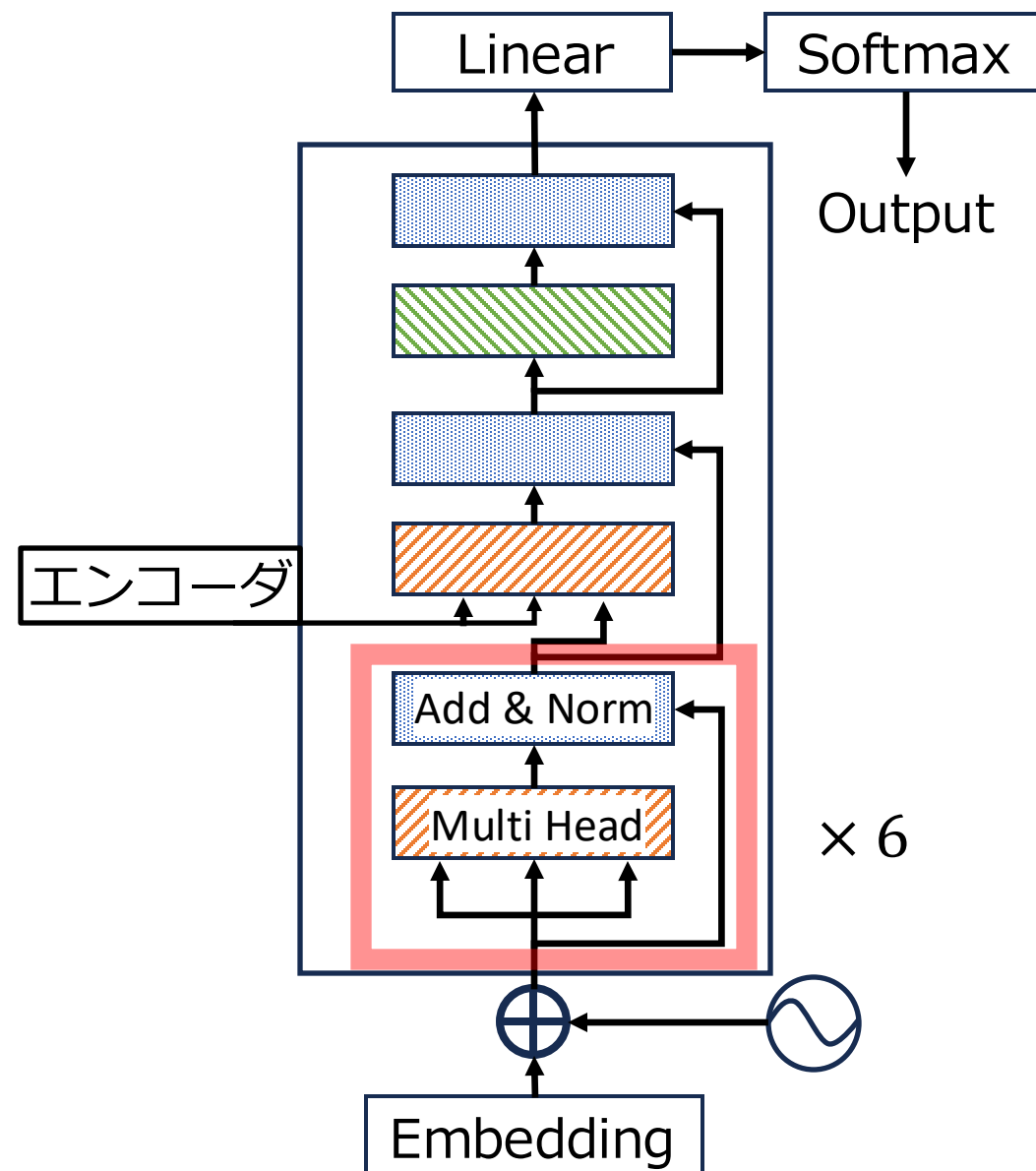


デコーダの動き

2. Masked Multi Head Attentionに入力

- Self Attention
- Mask
 - 正解の情報である未来の情報を隠す機構

| | | K^T | | | |
|---|----|-------|------|------|------|
| | | これ | は | ペン | です |
| Q | これ | 1.00 | 0.00 | 0.00 | 0.00 |
| | は | 0.03 | 0.97 | 0.00 | 0.00 |
| | ペン | 0.25 | 0.03 | 0.72 | 0.00 |
| | です | 0.00 | 0.02 | 0.03 | 0.95 |



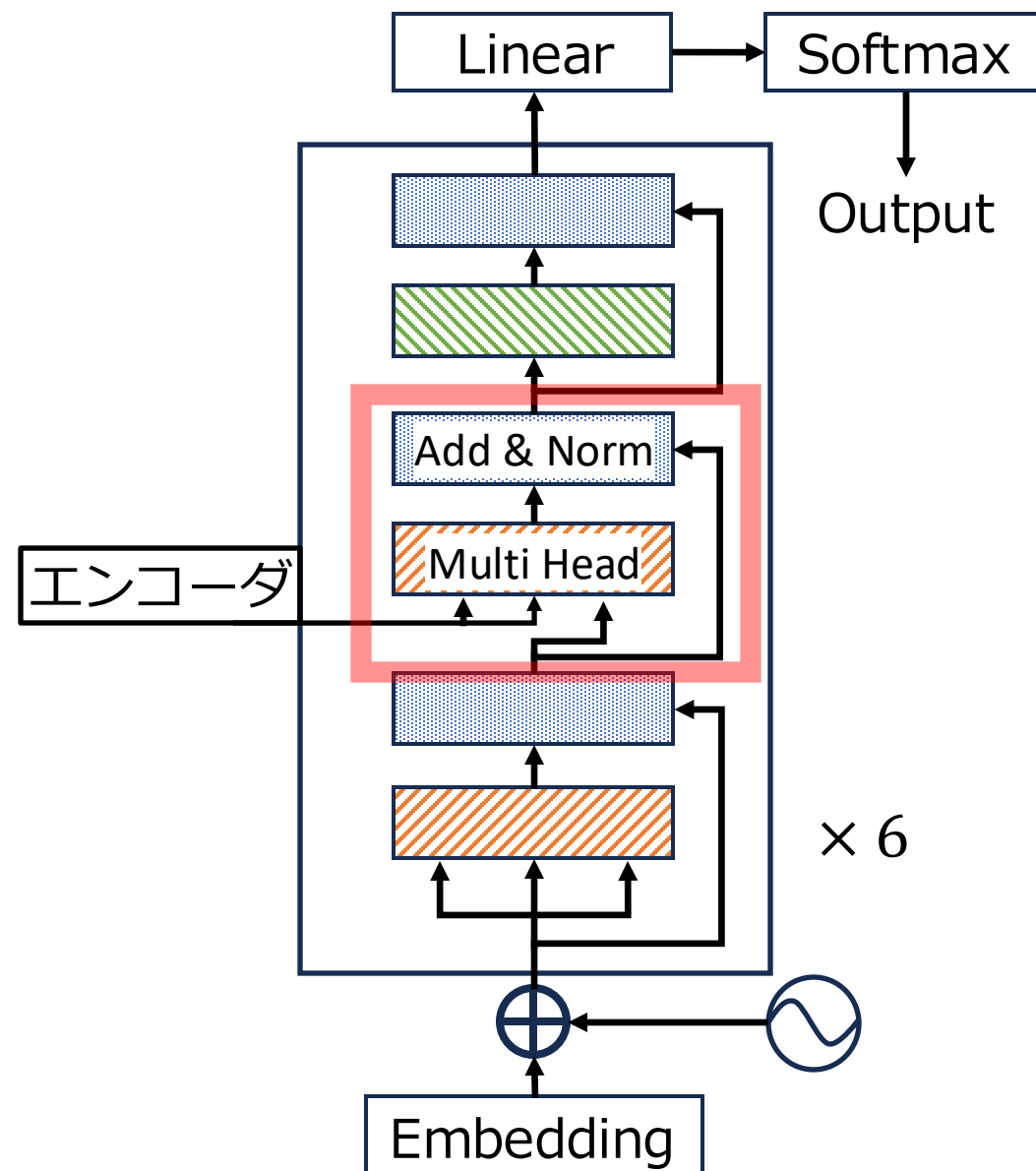
デコーダの動き

3. エンコーダからの出力も取り入れ Multi Head Attentionに入力

• Cross Attention

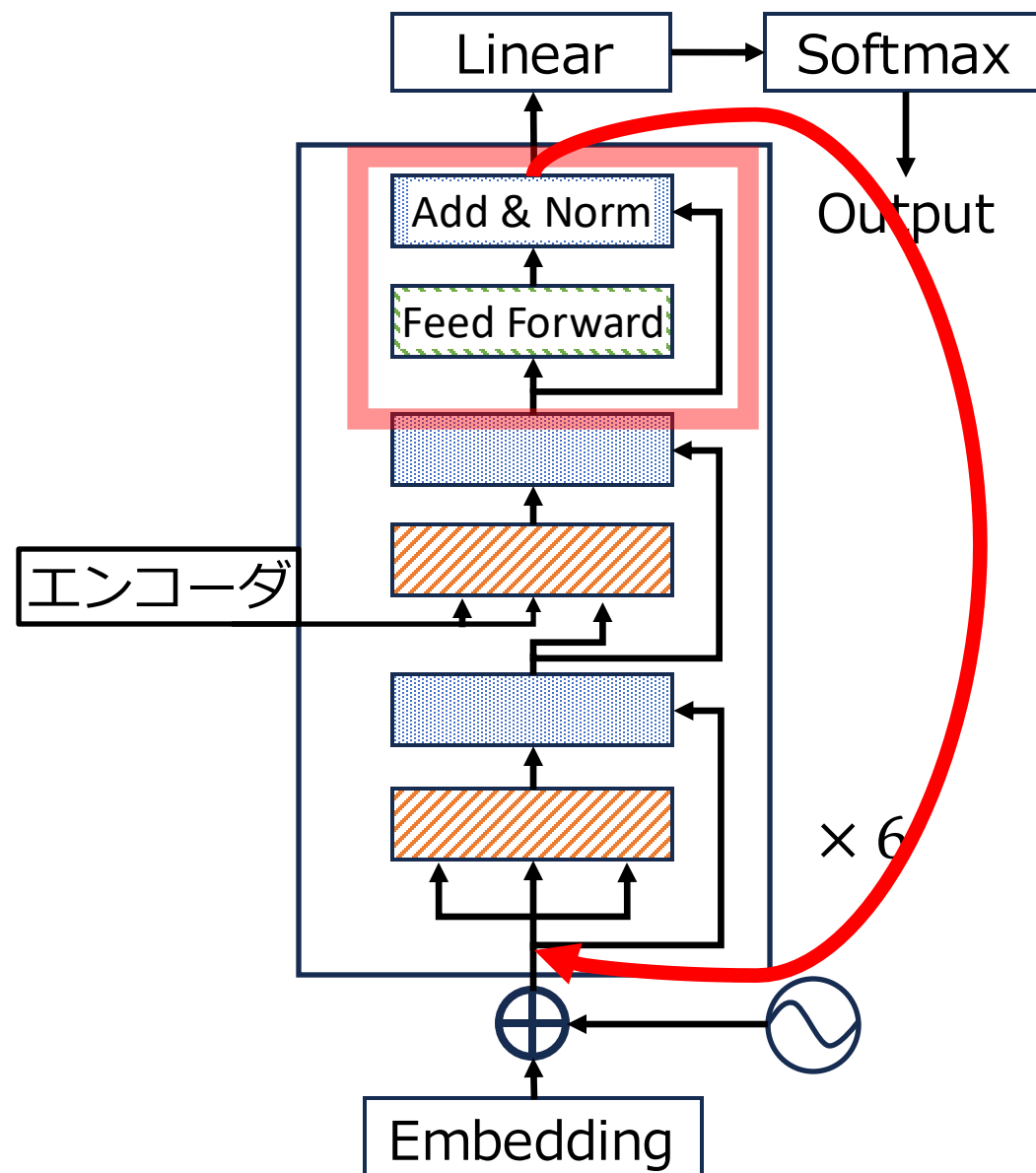
- X_{Target} : デコーダ内での情報
- X_{Source} : エンコーダからの情報
- 以上二つのデータ間の関係性を計算

| | | K^T | | | |
|---|----|-------|------|------|------|
| | | This | is | a | pen |
| Q | これ | 0.80 | 0.03 | 0.15 | 0.02 |
| | は | 0.02 | 0.95 | 0.03 | 0.00 |
| | ペン | 0.02 | 0.03 | 0.70 | 0.20 |
| | です | 0.15 | 0.05 | 0.75 | 0.05 |



デコーダの動き

4. Feed Forward Networkに入力
5. 次のデコーダに入力値として提供

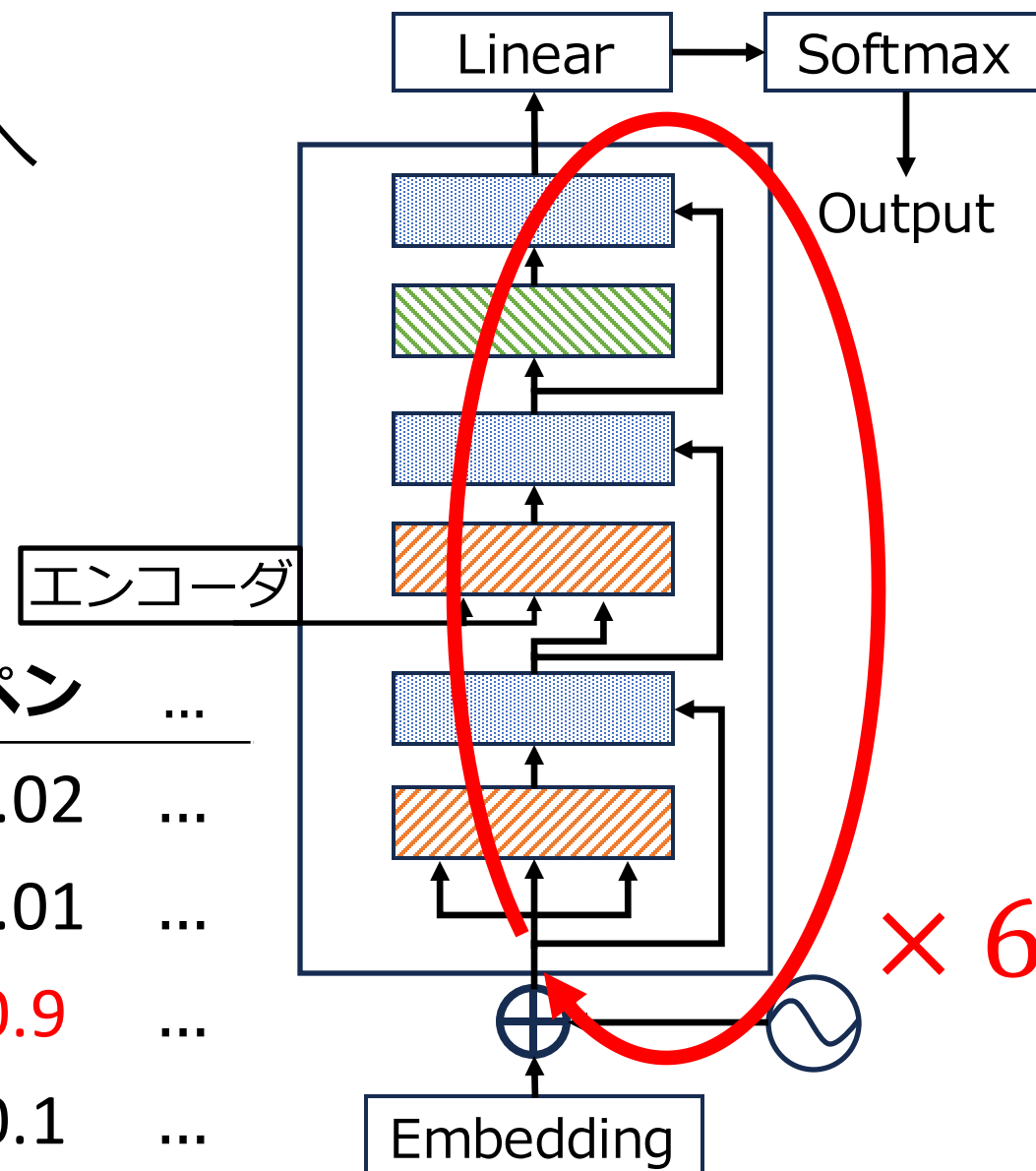


デコーダの動き

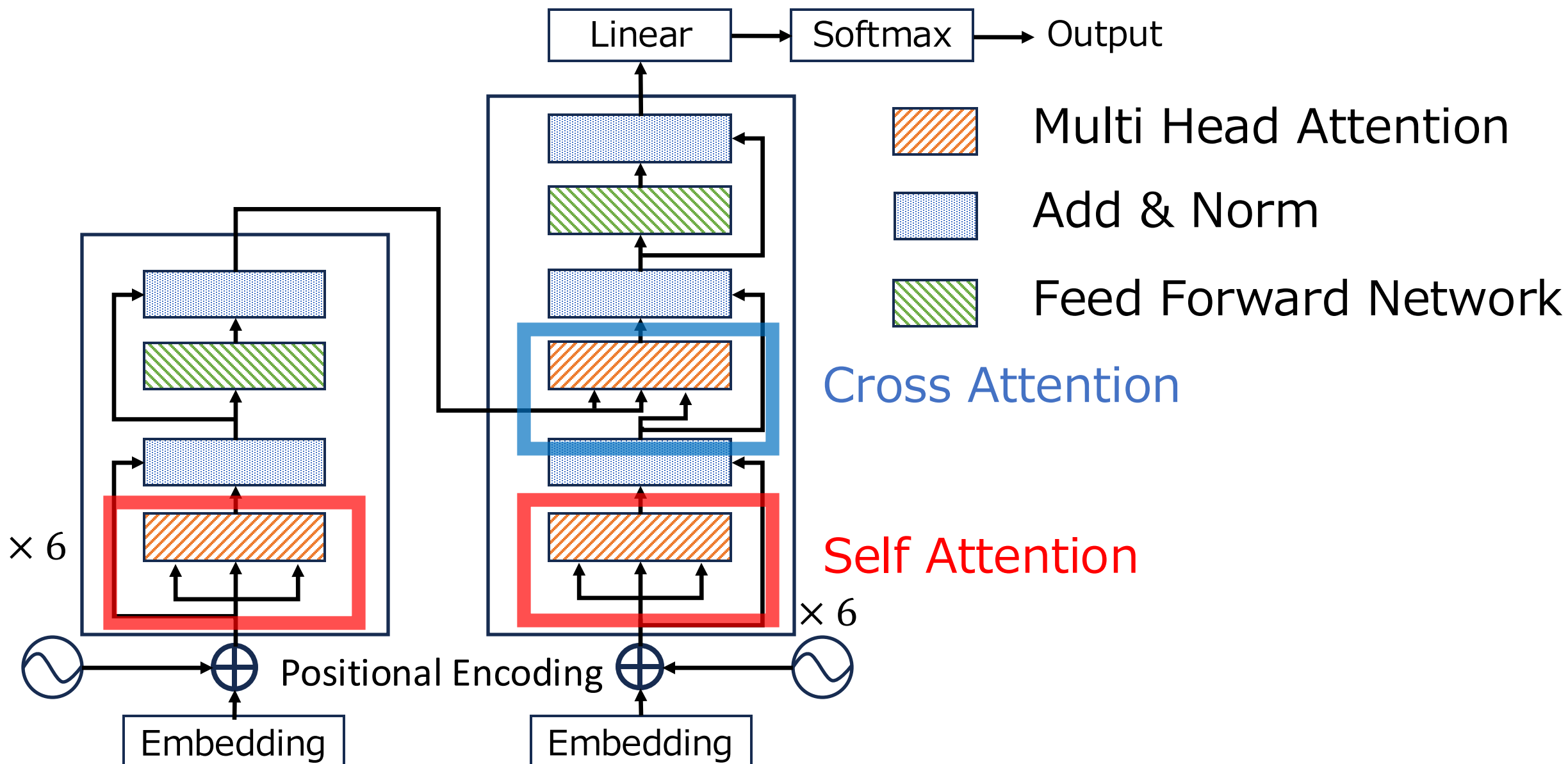
- 2～5を計6回繰り返し, 最上位のデコーダの出力値をLinear, Softmax へ
- 出力形式
 - 候補となる単語の確率が出力
 - 最高確率の単語を選択すれば翻訳可能

元の文 : This is a pen

| | これ | ... | です | ... | は | ... | ペン | ... |
|------|------|-----|------|-----|------|-----|------|-----|
| 1単語目 | 0.8 | ... | 0 | ... | 0 | ... | 0.02 | ... |
| 2単語目 | 0.01 | ... | 0 | ... | 0.7 | ... | 0.01 | ... |
| 3単語目 | 0.04 | ... | 0.02 | ... | 0.01 | ... | 0.9 | ... |
| 4単語目 | 0 | ... | 0.75 | ... | 0 | ... | 0.1 | ... |



全体像



Transformerの応用例

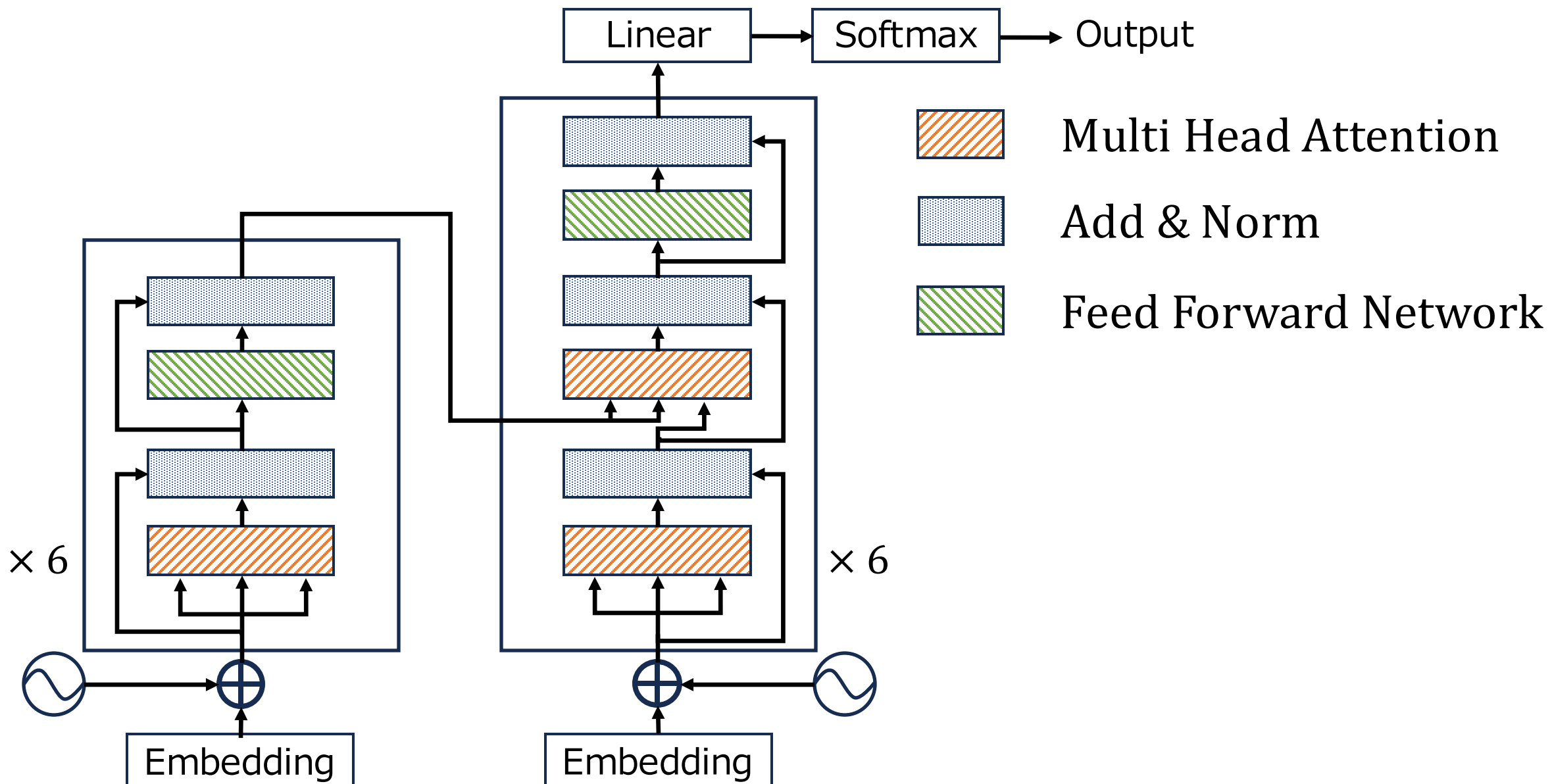
- 自然言語処理
 - 文脈理解, 文章生成, チャットボットに使用
 - GPT, BERT
 - 機械翻訳
 - Google翻訳, DeepL翻訳
- 画像処理
 - 画像分類
 - VIT
 - 画像生成
 - DALL-E, VQ-VAE

まとめ

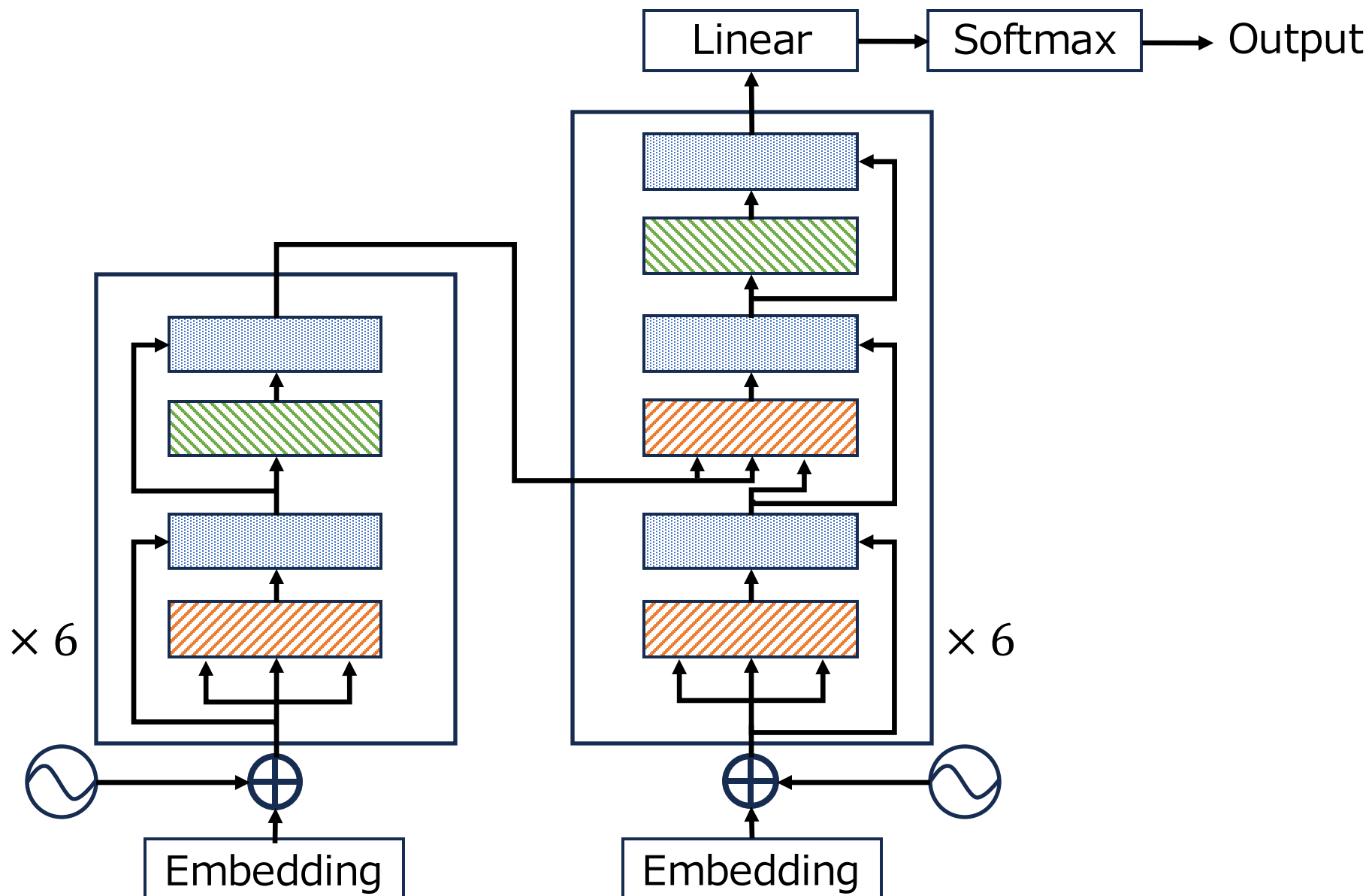
- Transformer
 - Attention機構で構成されたモデル
 - 自然言語処理，画像処理，音声処理 等 幅広い分野で活躍
- 主な提案
 - Multi Head Attention
 - 複数箇所に注目可能なAttention機構
 - Self Attention
 - 入力文中の単語間の関係を計算
 - Positional Encoding
 - Attentionのみでは学習できない時系列情報を埋め込む

補足資料

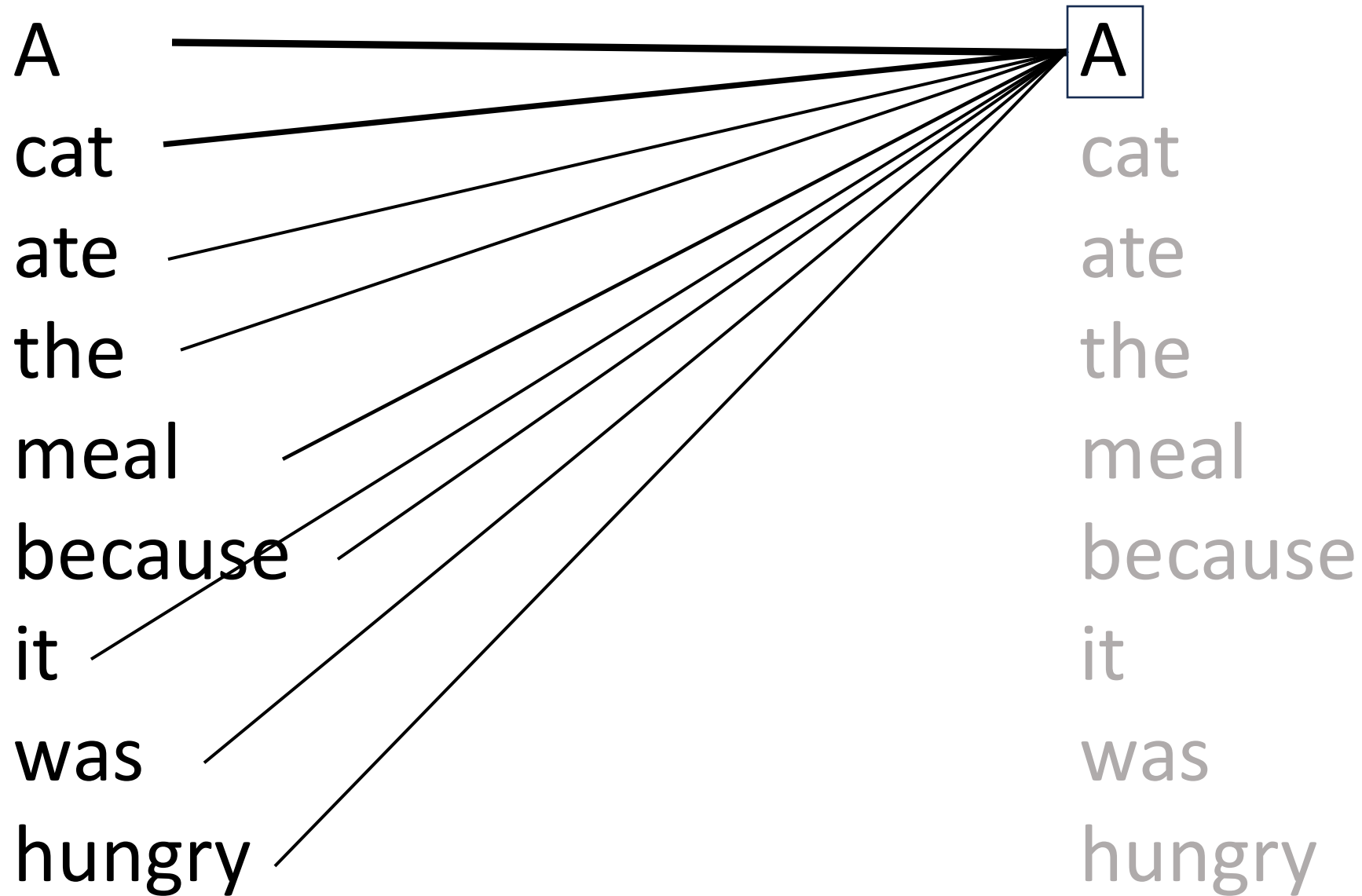
全体像



全体像



セルフアテンション (機械翻訳)



実験

- タスク
 - 英語からドイツ語
 - 英語からフランス語 への機械翻訳
- データセット
 - WMT 2014 News : 翻訳されたニュースでの英語の文章
- 評価指標
 - BLEUスコア

BLEUスコア

- 機械翻訳の精度を評価する指標
 - 人間の翻訳例と機械翻訳の類似度を計算

$$BLEU = BP_{BLEU} \times \exp\left(\sum_{n=1}^N w_n \log p_n\right)$$

$$p_n = \frac{\sum_i \text{翻訳文 } i \text{ と参照訳 } i \text{ で一致した } n - \text{gram 数}}{\sum_i \text{翻訳文 } i \text{ 中の全 } n - \text{gram 数}} \quad w_n = \frac{1}{N}$$

BP_{BLEU} : 文の長さが参照例より短い時のペナルティ

実験結果

| Model | BLEU | | Training Cost (FLOPs) | |
|---------------------------------|-------------|--------------|---------------------------------------|---------------------|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [15] | 23.75 | | | |
| Deep-Att + PosUnk [32] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [31] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [8] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [26] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [32] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [31] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [8] | 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 | $2.3 \cdot 10^{19}$ | |

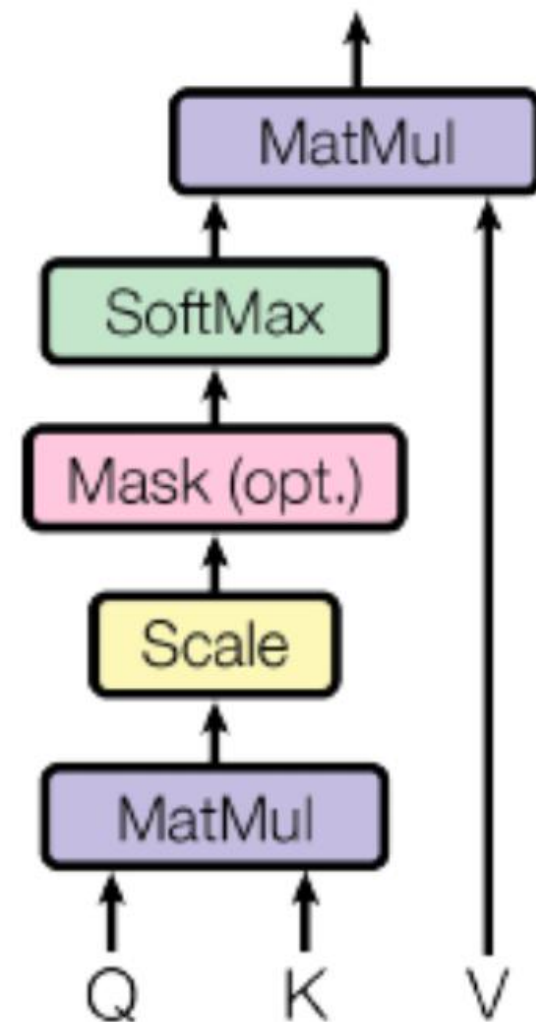
Attention計算

スケールドットプロダクトアテンション

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

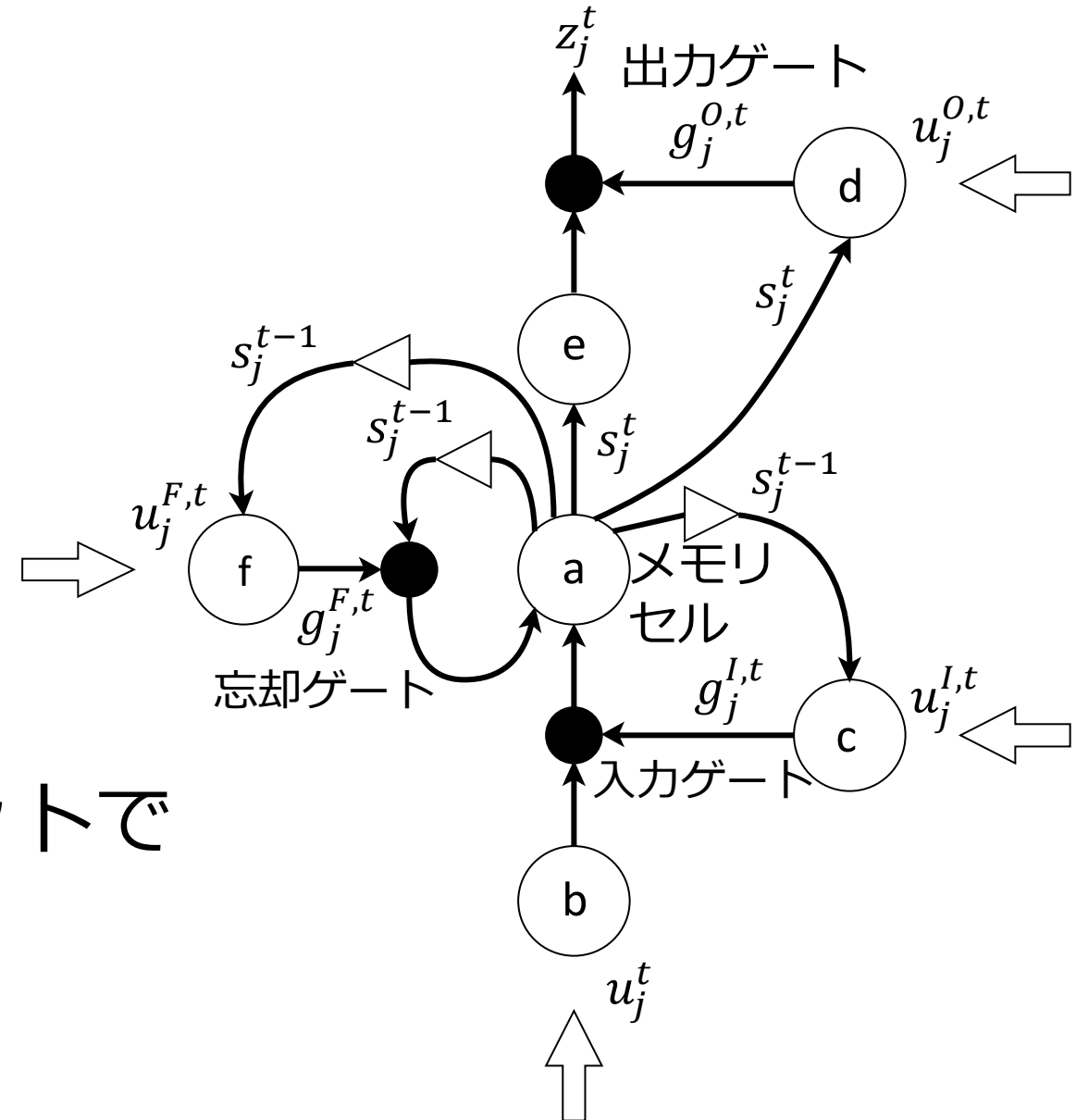
d は Q, K の埋め込みベクトルの次元数

- データ同士の関連度計算
- 関連度のスケーリング
- 関連度のソフトマックス正規化
- Q と K の関連度でValueの重みづけ



LSTM

- 入力ゲート $g_j^{I,t}$
- 出力ゲート $g_j^{O,t}$
- 忘却ゲート $g_j^{F,t}$
- メモリセルの状態 s_j^t
- 長期記憶を可能にするために
考案された
- 中間層のユニットをメモリユニットで
置き換えた構造



LSTMからTransformerにする利点

- 並列計算が可能
- 長期依存関係の学習の改善

行正規化

- 行ごとに正規化

例 (イメージ)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



$$\begin{bmatrix} 0.17 & 0.33 & 0.5 \\ 0.27 & 0.33 & 0.4 \\ 0.29 & 0.33 & .38 \end{bmatrix}$$

トークン化（日本語）

- 単語トークン化
 - テキスト: "私はリンゴが好きです。"
 - トークン: ["私", "は", "リンゴ", "が", "好き", "です", "。"]
- 文字トークン化
 - テキスト: "リンゴ"
 - トークン: ["リ", "ン", "ゴ"]
- サブワードトークン化（イメージ）
 - テキスト: "私はリンゴが好きです"
 - トークン: ['私', 'は', 'リン', 'ゴ', 'が', '好', 'き', 'です']

関連度スケーリング理由

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

d は Q, K の埋め込みベクトルの次元数

- 内積の計算結果が大きくなりすぎるのを防ぐ
- 大きくなりすぎると
 - 数值的に不安定に
 - 勾配消失, 勾配爆発



Positional Encoding 計算例

- $d_{model} = 4$ と仮定して計算
- トークン数は3

$$P_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

d_{model} : 総次元数
 pos : 文字の位置
 i : 次元の位置

Positional Encoding 計算例

$$P =$$

| | | | | | |
|------|-------|--|--|--|--|
| | pos | | | | |
| I | 0 | $\sin\left(\frac{pos}{10000^{0/4}}\right)$ | $\cos\left(\frac{pos}{10000^{0/4}}\right)$ | $\sin\left(\frac{pos}{10000^{2/4}}\right)$ | $\cos\left(\frac{pos}{10000^{2/4}}\right)$ |
| am | 1 | $\sin\left(\frac{pos}{10000^{0/4}}\right)$ | $\cos\left(\frac{pos}{10000^{0/4}}\right)$ | $\sin\left(\frac{pos}{10000^{2/4}}\right)$ | $\cos\left(\frac{pos}{10000^{2/4}}\right)$ |
| good | 2 | $\sin\left(\frac{pos}{10000^{0/4}}\right)$ | $\cos\left(\frac{pos}{10000^{0/4}}\right)$ | $\sin\left(\frac{pos}{10000^{2/4}}\right)$ | $\cos\left(\frac{pos}{10000^{2/4}}\right)$ |
| | | 0 | 1 | 2 | 3 i |

Positional Encoding 計算例

$$P =$$

| | | | | | |
|------|-------|-------------|-------------|------------------------------------|------------------------------------|
| | pos | | | | |
| I | 0 | $\sin(pos)$ | $\cos(pos)$ | $\sin\left(\frac{pos}{100}\right)$ | $\cos\left(\frac{pos}{100}\right)$ |
| am | 1 | $\sin(pos)$ | $\cos(pos)$ | $\sin\left(\frac{pos}{100}\right)$ | $\cos\left(\frac{pos}{100}\right)$ |
| good | 2 | $\sin(pos)$ | $\cos(pos)$ | $\sin\left(\frac{pos}{100}\right)$ | $\cos\left(\frac{pos}{100}\right)$ |
| | | 0 | 1 | 2 | 3 |
| | | | | | i |

Positional Encoding 計算例

$$P = \begin{matrix} & pos \\ \begin{matrix} I \\ am \\ good \end{matrix} & \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} \end{matrix} \begin{bmatrix} \sin(0) & \cos(0) & \sin\left(\frac{0}{100}\right) & \cos\left(\frac{0}{100}\right) \\ \sin(1) & \cos(1) & \sin\left(\frac{1}{100}\right) & \cos\left(\frac{1}{100}\right) \\ \sin(2) & \cos(2) & \sin\left(\frac{2}{100}\right) & \cos\left(\frac{2}{100}\right) \end{bmatrix}$$

Positional Encoding 計算例

$$P =$$

| | | | | | | |
|------|-------|--|---|---|---|-----|
| | pos | | | | | |
| I | 0 | $\begin{bmatrix} 0.00 & 1.00 & 0.00 & 1.00 \\ 0.84 & 0.54 & 0.01 & 0.99 \\ 0.91 & -0.41 & 0.02 & 0.99 \end{bmatrix}$ | | | | |
| am | 1 | | | | | |
| good | 2 | | | | | |
| | | 0 | 1 | 2 | 3 | i |

Positional Encoding Pの計算

$$X = \begin{bmatrix} 2.62 & 1.54 & 3.51 & 7.64 \\ 1.45 & 2.31 & 5.14 & 4.38 \\ 3.22 & 5.33 & 4.17 & 3.21 \end{bmatrix} + \begin{bmatrix} 0.00 & 1.00 & 0.00 & 1.00 \\ 0.84 & 0.54 & 0.01 & 0.99 \\ 0.91 & -0.41 & 0.02 & 0.99 \end{bmatrix}$$

X'
 P

$$= \begin{bmatrix} 2.62 & 2.54 & 3.51 & 8.64 \\ 2.29 & 2.85 & 5.15 & 5.37 \\ 4.13 & 4.92 & 4.19 & 4.20 \end{bmatrix}$$

フィードフォワードネットワーク

入力層 512次元

中間層 2048次元

出力層 512次元

