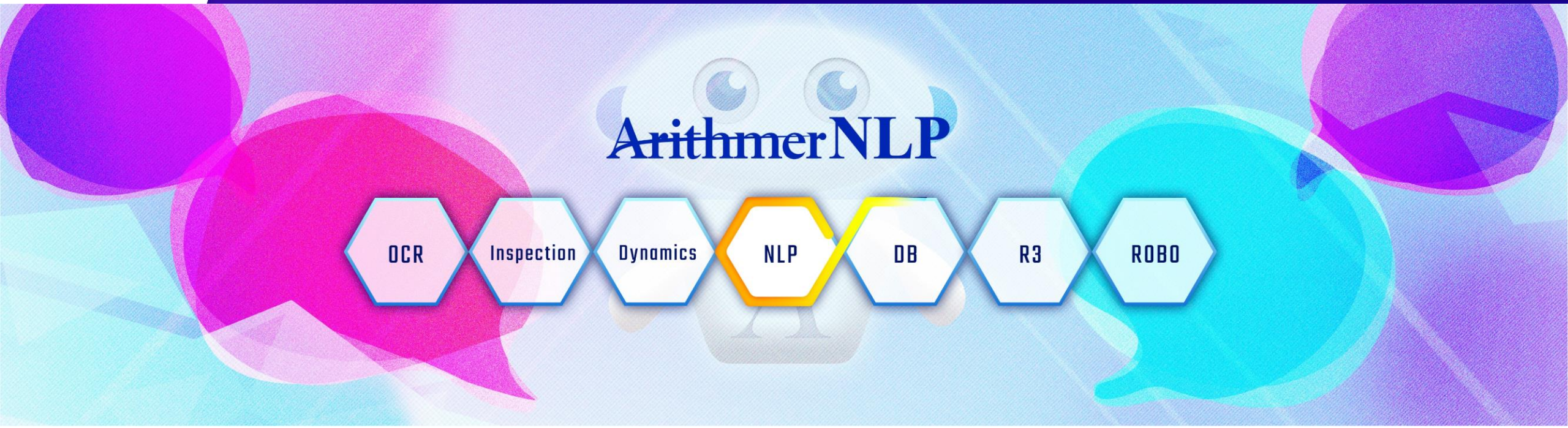


全力解説！Transformer

2021/4/13

NLP team: Haruya Umemoto

梅本 晴弥 (Umemoto Haruya)

キャリア

2018, 青山学院大学（情報テクノロジー学科）卒業　学士（工学）
2020, 青山学院大学大学院（知能情報コース）卒業　修士（工学）
2020, Arithmer 株式会社(NLPチーム)

専門分野：強化学習，機械学習，自然言語処理（対話システム, 大規模事前学習モデル）

受賞歴
2020 - 人工知能学会研究会優秀賞
2020 - 薦田先端学術賞
2020 - 学業成績最優秀賞
2020 - 修士論文発表優秀賞
2017 - データ解析ハッカソン2017準優勝
(SIG-DOCMAS主催)

担当案件等
- 自己学習を促す教育用レコメンドAI
(https://arithmer.co.jp/210405)
- SPI等に基づいた能力予測システム
- 就活生の傾向把握のためのES分析
- 人間のフィードバックに基づいた対話システム研究開発

ポートフォリオサイト：https://umeco.tokyo

目次

【概要】Transformerって何？

2017年の論文 **Attention Is All You Need[1]** で発表されたモデル

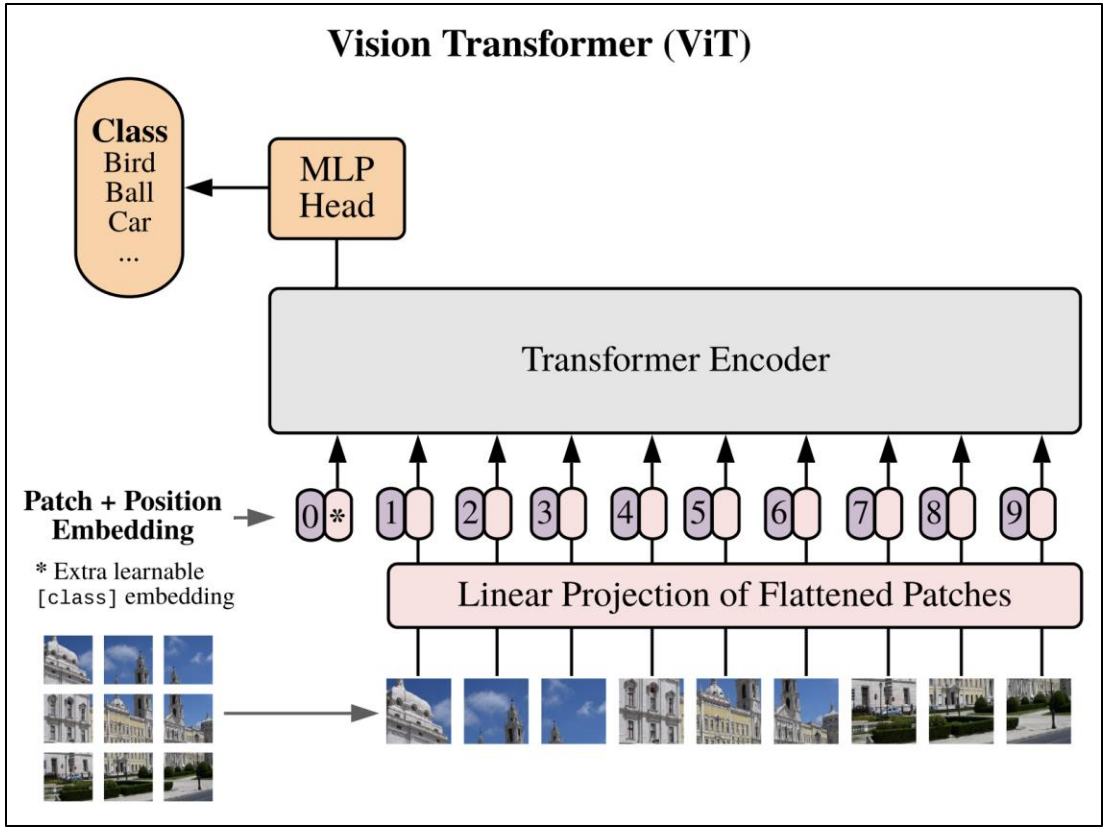
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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**	**41.8**	$2.3 \cdot 10^{19}$	

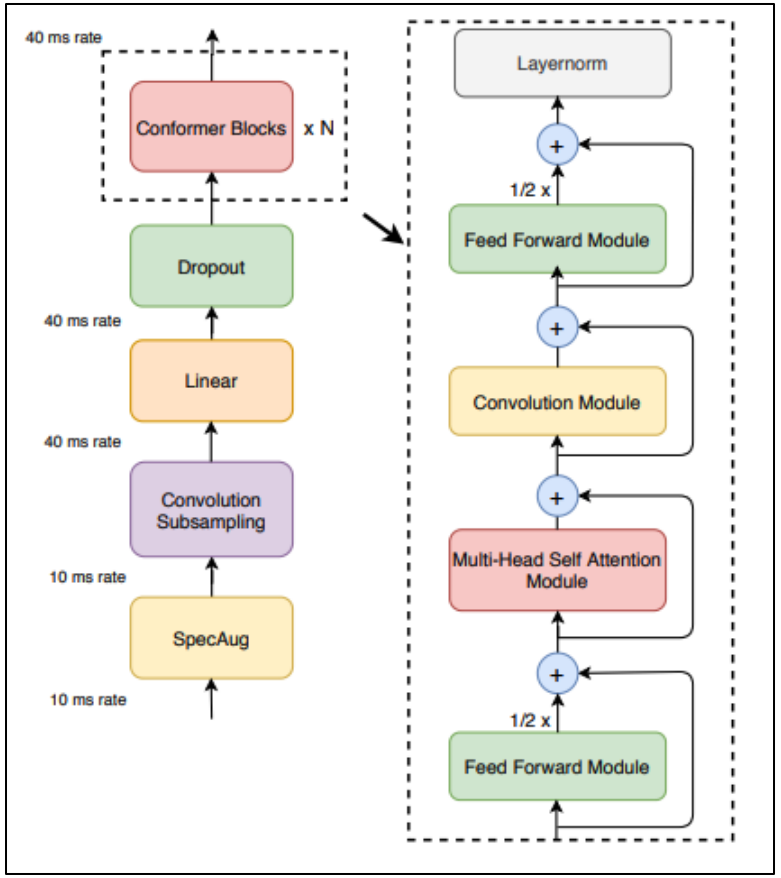
機械翻訳タスクにおいて既存SOTAよりも高いスコアを記録

[1] Vaswani, Ashish et al. “Attention is All you Need.” *ArXiv* abs/1706.03762 (2017)

【概要】Transformerの他分野への応用



[2] 画像分野での応用

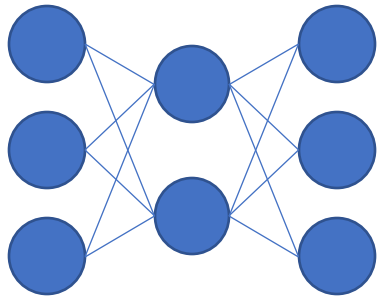


[3] 音声分野での応用

NLPだけでなく幅広い分野へ応用

[2] Dosovitskiy, A. et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.” *ArXiv* abs/2010.11929 (2020)
[3] Gulati, Anmol et al. “Conformer: Convolution-augmented Transformer for Speech Recognition.” *ArXiv* abs/2005.08100 (2020)

【概要】NLPにおけるモデルの発展

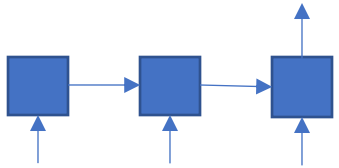


Full Fonnection

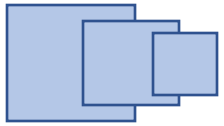
統計的自然言語処理

- Bug of Words
- 分かち書き
- 共起ベクトル

以前



RNN系

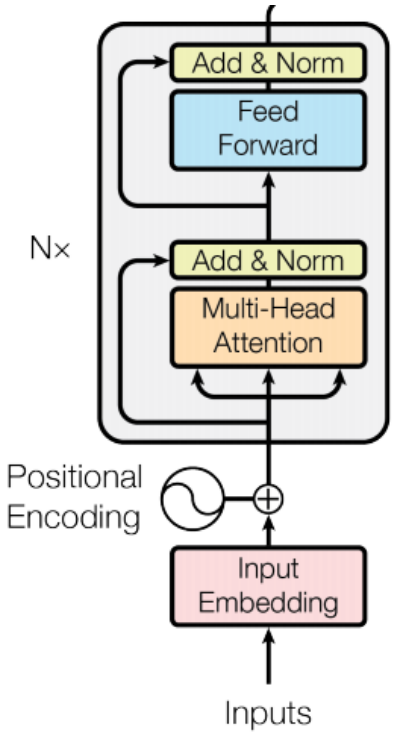


CNN

分散表現獲得手法

- Word2vec
- Glove
- Fasttext

2011年 ~



Transformer爆誕

2017年 ~

事前学習モデル

- GPT-1,2,3
- BERT
- M2M100
- T5

Attention改良

- Refomer
- Longformer
- Big Bird
- Peformer

現在

* RNNやLSTM，CNNのモデル提案はもっと以前

【概要】RNN, CNN vs Transfomer

Recurrent Neural Network (RNN)

👍 以前の情報の記憶による時系列入力処理
👎 長い系列における記憶がほとんどできない
👎 👎 前の状態の演算結果が次の演算に必要でGPUを活用できない

Convolutional Neural Network (CNN)

👍 局所特徴量の抽出
👍 並列演算によるGPUの活用
👎 長期的な依存関係を捉えることができない

Transformer

👍 Attentionベース構造によるGPUの活用
👍 Self-Attentionによる超高性能な特徴量抽出
👍 長期的な依存関係を捉えることができる
👎 系列長が長い場合のメモリ消費

目次

【構造解説】Transformerの全体図

エンコーダ部分

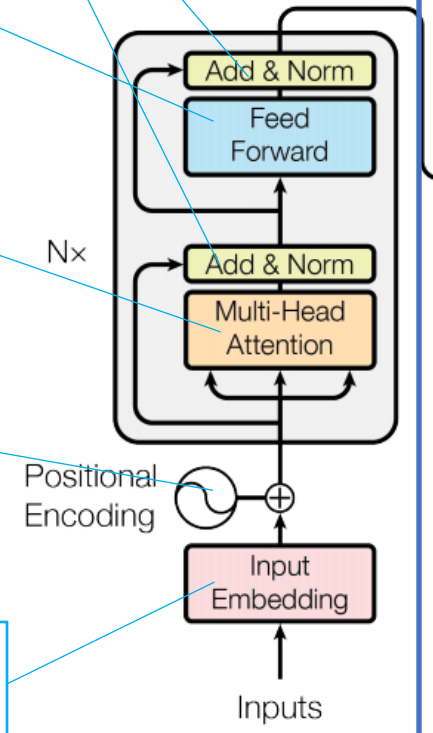
残差演算＋
レイヤー正規化

全結合層

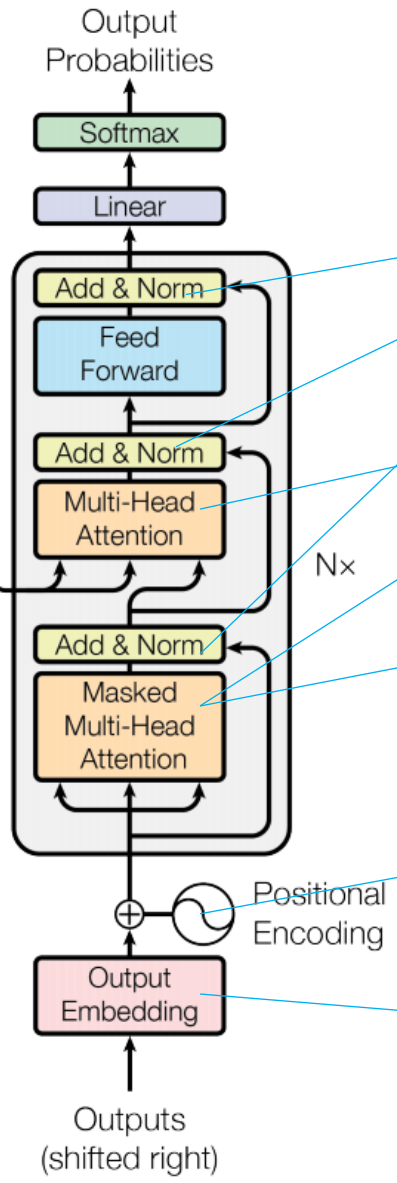
Attentionによる
特徴量抽出

時系列情報の付与

単語のベクトル化



デコーダ部分



残差演算＋
レイヤー正規化

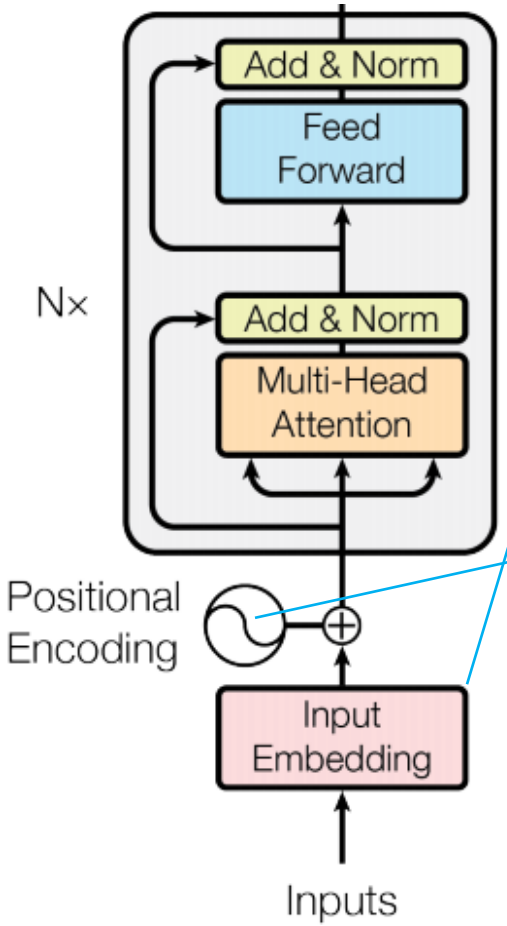
Attentionによる
特徴量抽出

未来の情報のマスク

時系列情報の付与

単語のベクトル化

【構造解説】Positional Encoding（１）

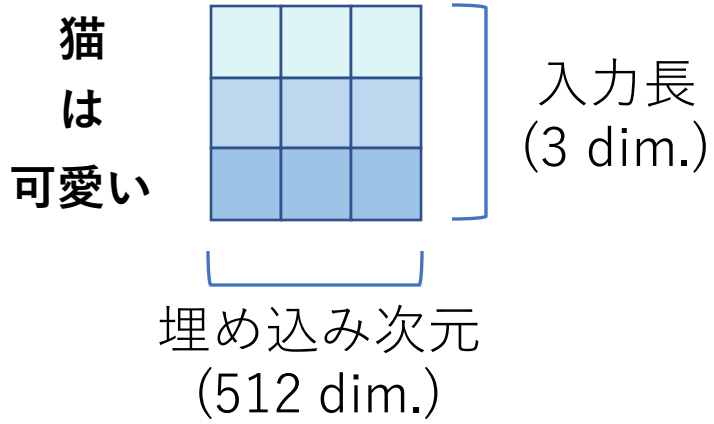


入力例：[“猫”, “は”, “可愛い”]

Embedding Layer

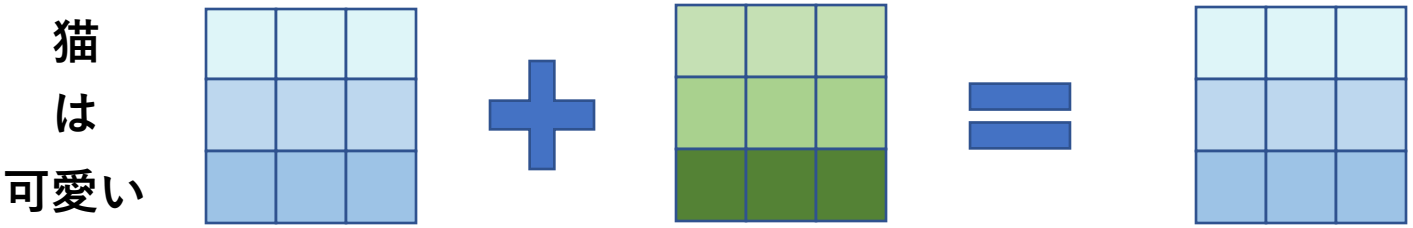
単語をベクトル化(単語列なので行列)

[“猫”, “は”, “可愛い”]



Positional Encoding

時系列情報の付与



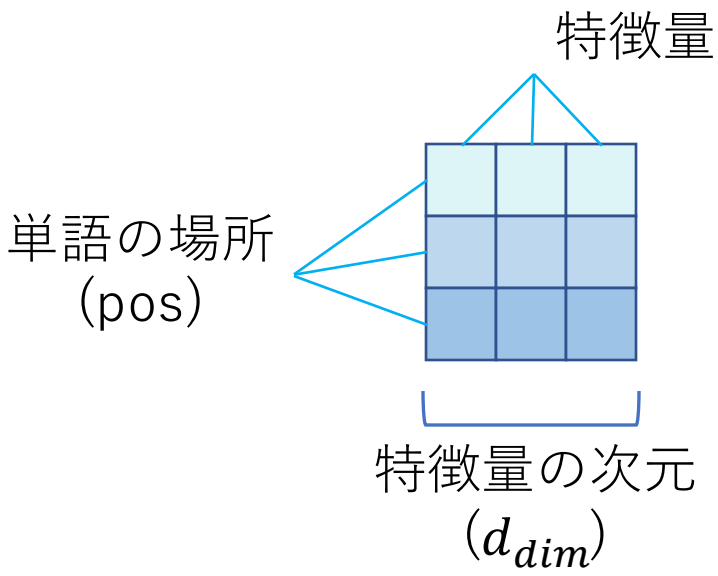
単語特徴量　　時系列情報　　時系列情報が付与された単語特徴量

【構造解説】Positional Encoding（２）

時系列情報の計算方法

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

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



pythonでの実装

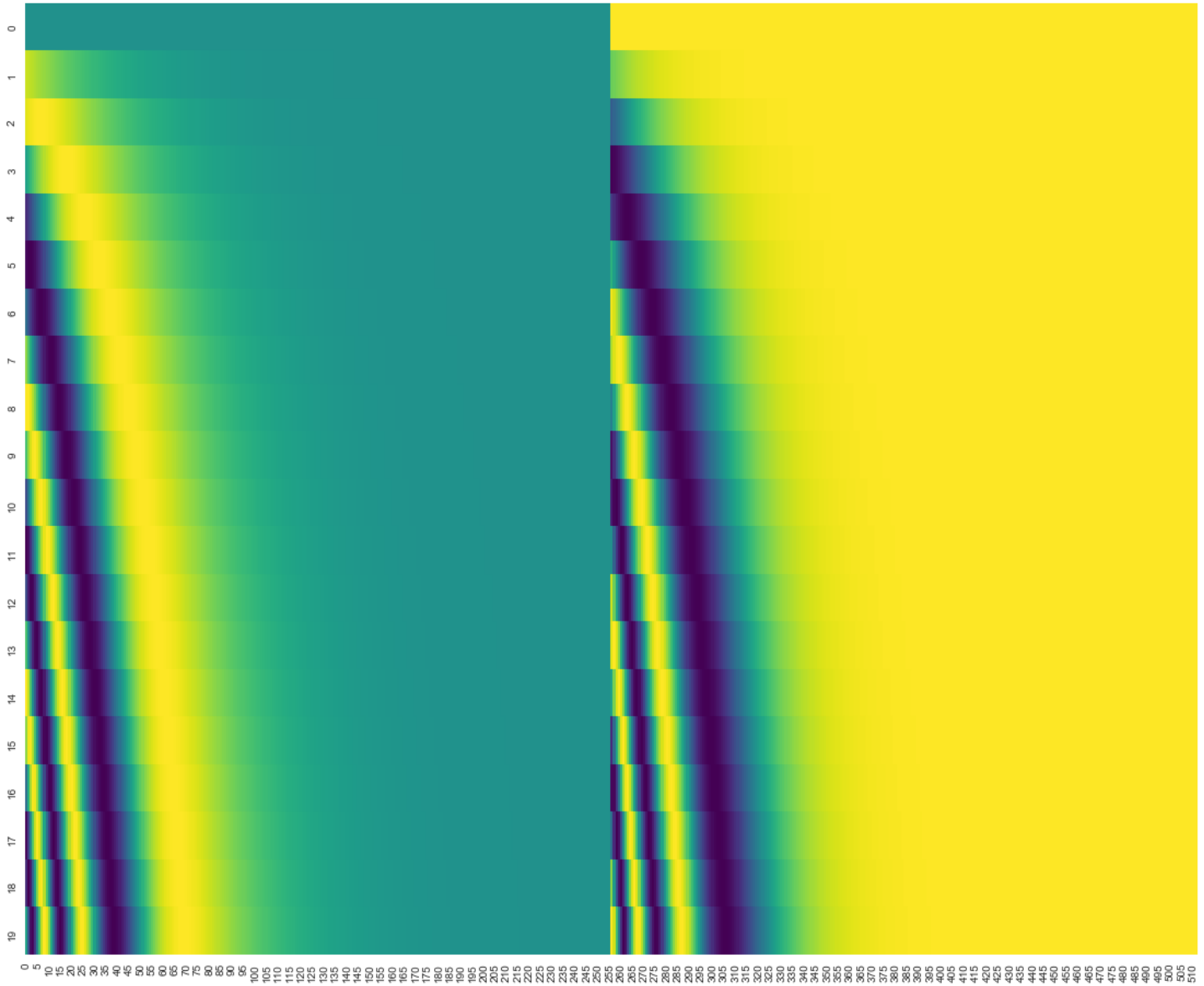
```
def positional_encoding(pos, i, dim):
    if i//2 == 0:
        return np.sin(pos/10000**(i/dim))
    else:
        return np.cos(pos/10000**((i-1)/dim))

for i in range(0, 512):
    print(positional_encoding(1, i, 512))
```

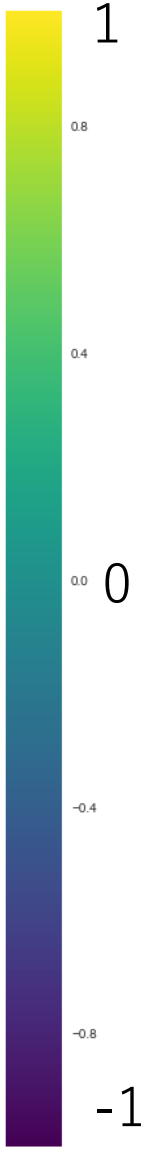
なぜ時系列情報を付与するのか？

TransformerはRNNのように
モデル構造で時系列を扱わない

特徴量側で時系列情報を扱う

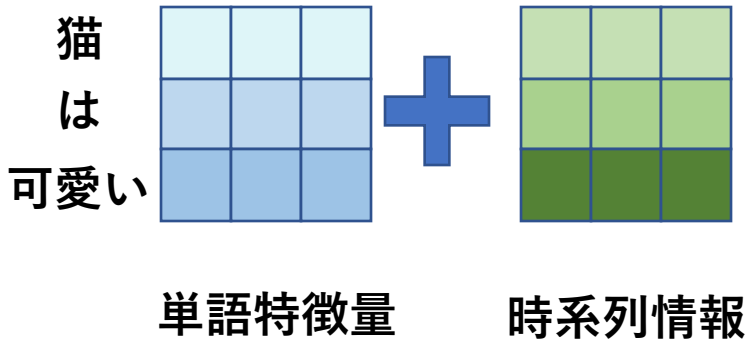
【構造解説】Positional Encoding（３）





Positional Encodingの可視化[4]

横軸：特徴量のインデックス（i）
縦軸：単語の位置（pos）

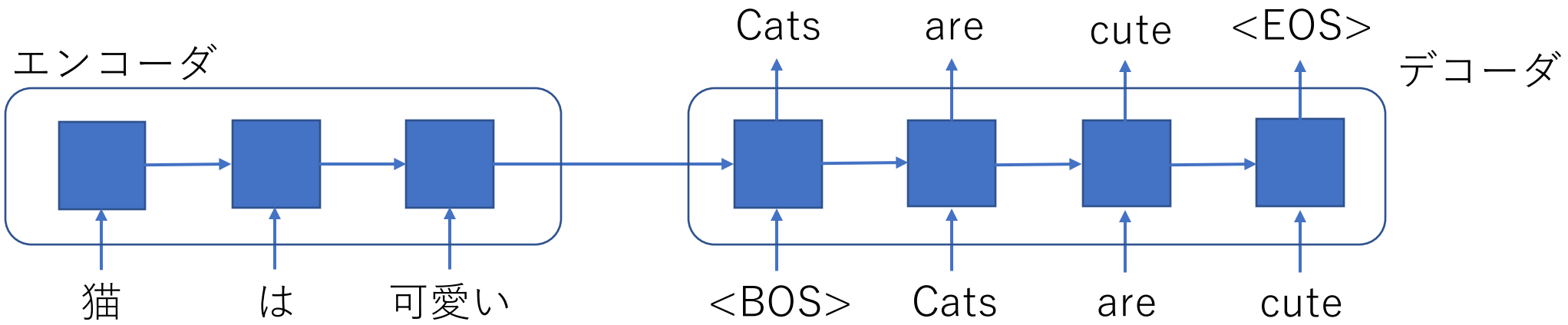


[4] http://jalammar.github.io/illustrated-transformer より

目次

【構造解説】Attention機構（１）

従来のSeq2Seq型のRNN翻訳モデル

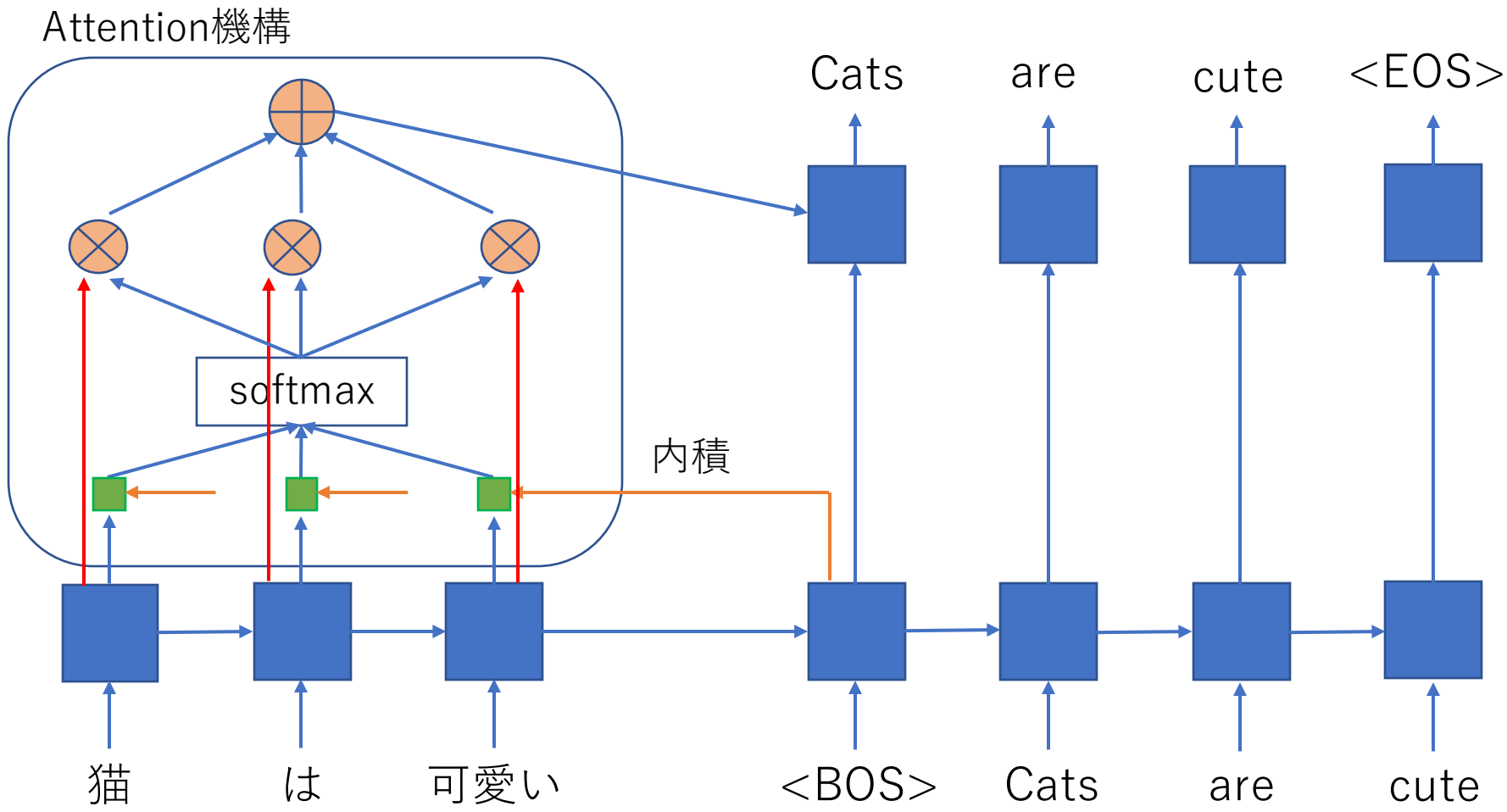


入力長が長くなるほど精度が低下

今までの入力全てをベクトルとして保持するのが難しい

【構造解説】Attention機構（２）

Attentionを利用したSeq2Seq型のRNN翻訳モデル



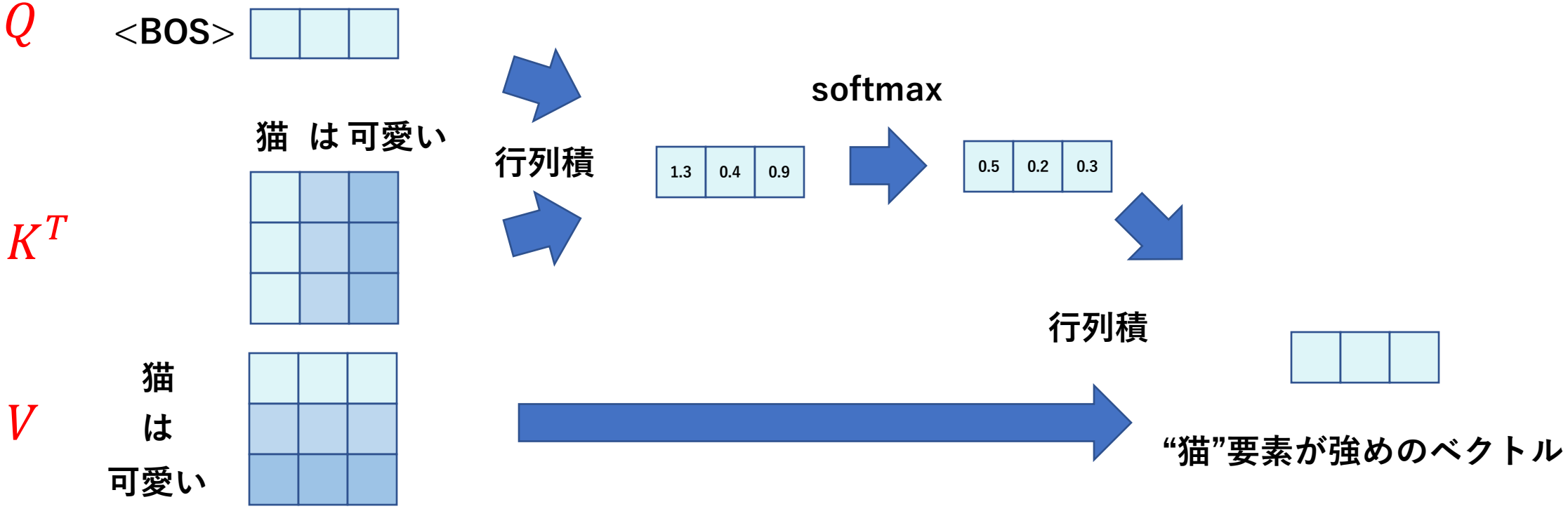
【構造解説】Attention機構(３)

Attention計算式(dot product attention)

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

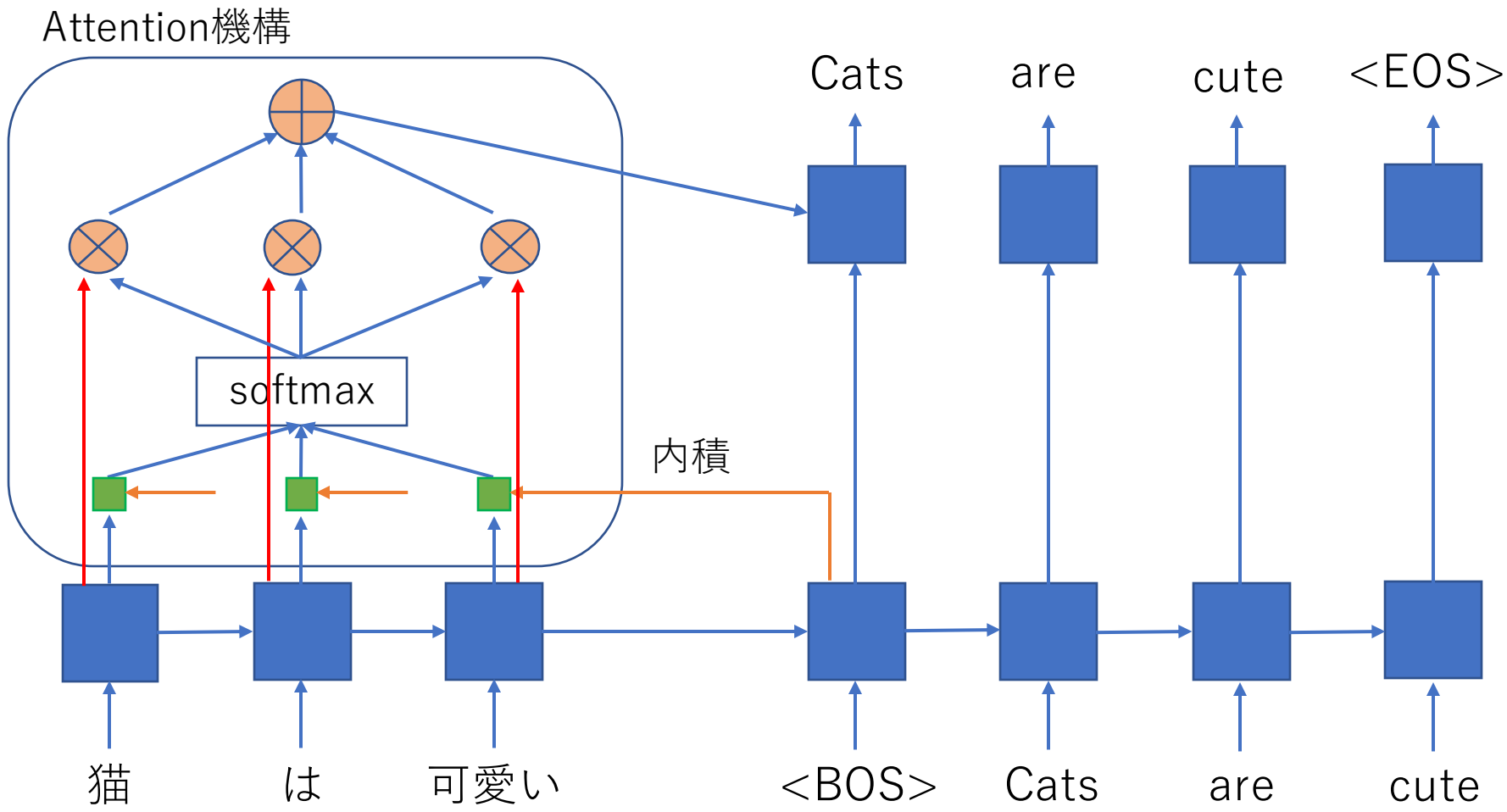
Q: Query, これに関連するものをAttentionしたい
K: Key, memoryの特徴量
V: Value, memoryの値(多くの場合は$K = V$)

AttentionとはQueryで
Memory(key, value)から
重み付けして情報を取得する手法



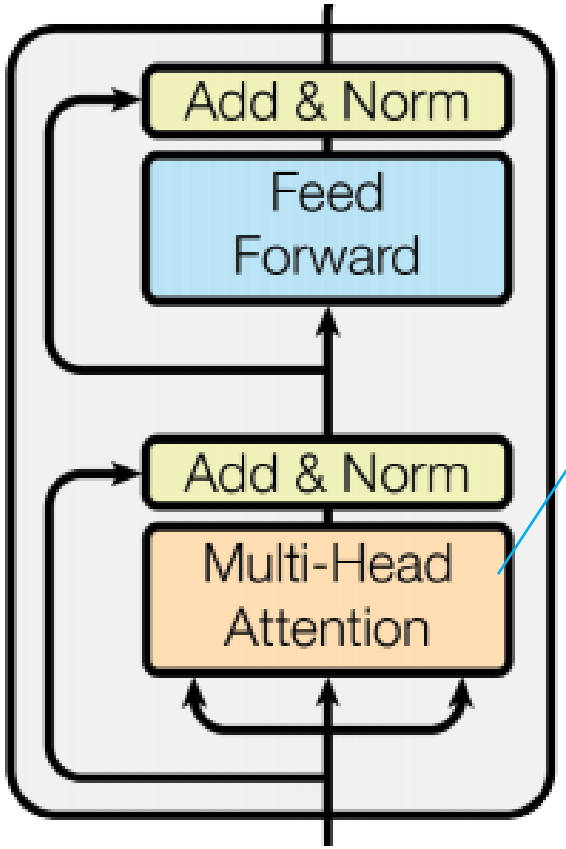
【構造解説】Attention機構（４）

Attentionを利用したSeq2Seq型のRNN翻訳モデル



目次

【構造解説】TransformerのAttention



猫
は
可愛い

（時系列情報が付与）

Multi-Headed Scaled Dot-Product Self-Attention

- **Self-Attention**

  Query, Key, Valueが全て同じAttention

- **Scaled Dot-Product (Attention)**

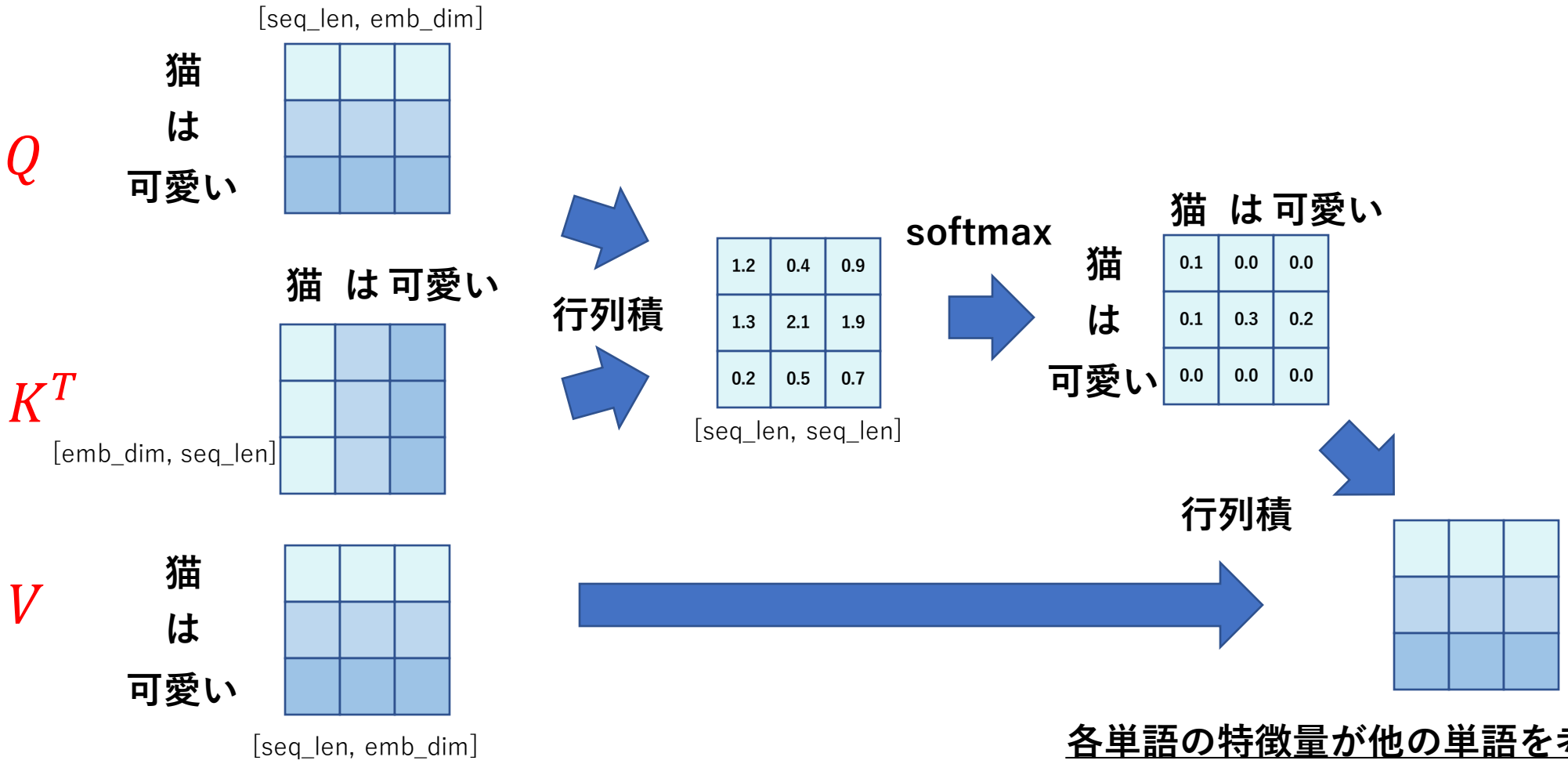
  scaling factorを用いる内積Attention

- **Multi-Headed (Attention)**

  複数のヘッドを用いたAttention

【構造解説】Self-Attention

Query, Key, Valueが全て同じAttention



各単語の特徴量が他の単語を考慮されて再構成

【構造解説】Scaled Dot-Product Attention

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

Q: Query，これに関連するものをAttentionしたい
K: Key，memoryの特徴量
V: Value，memoryの値(多くの場合は$K = V$)
d_k: 埋め込みベクトルの次元数さっきのemb_dim

なぜd_kで割るのか？ ⇒ 確率が低い部分の勾配情報を保持したいため

平均0，分散1の正規分布から独立に取得された要素で構成されるベクトルqとkを仮定

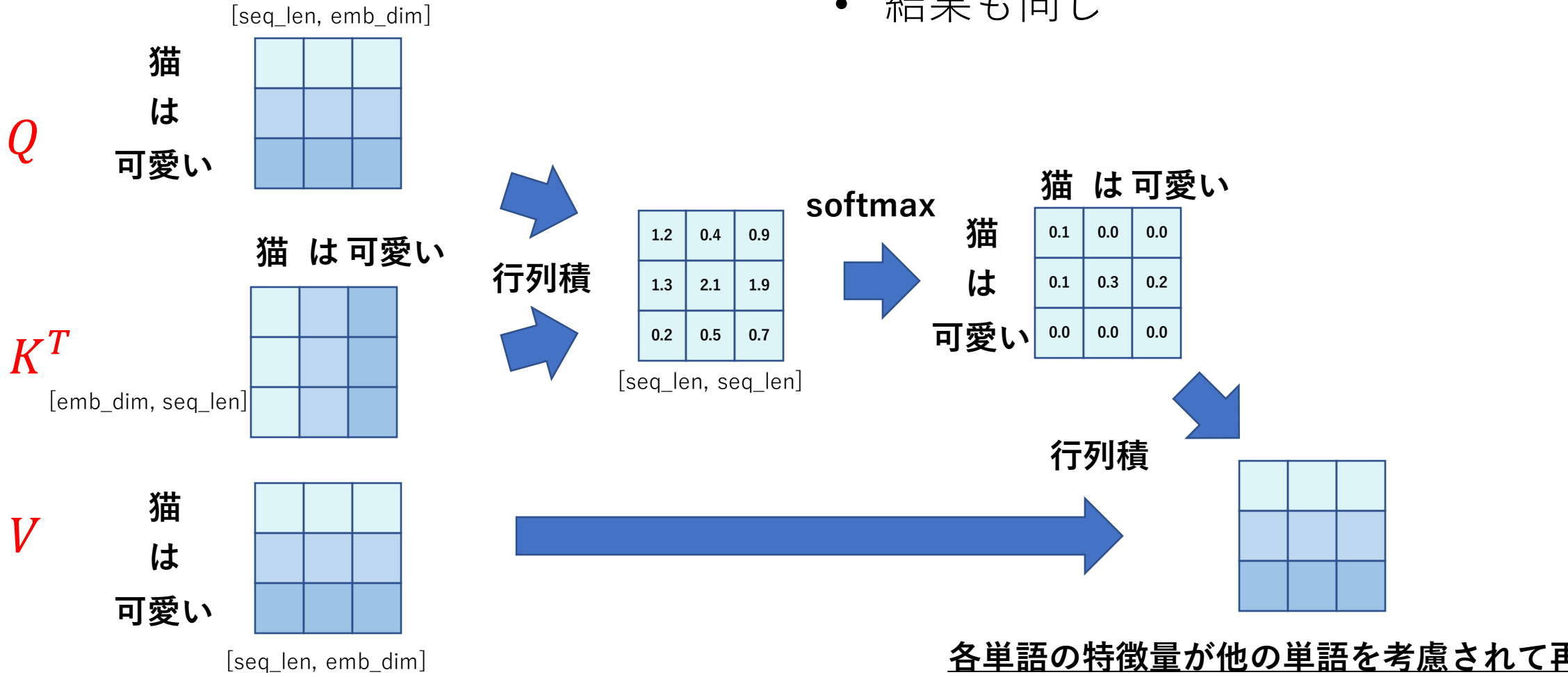
$$q \cdot k = \sum_{i=1}^{d_k} q_i k_i$$

この時，内積は平均0，分散$\mathrm{d_k}$となる ⇒ softmaxした時に極小となる要素が発生

【構造解説】Multi-Head Attention(１)

並列に複数のAttentionを実施

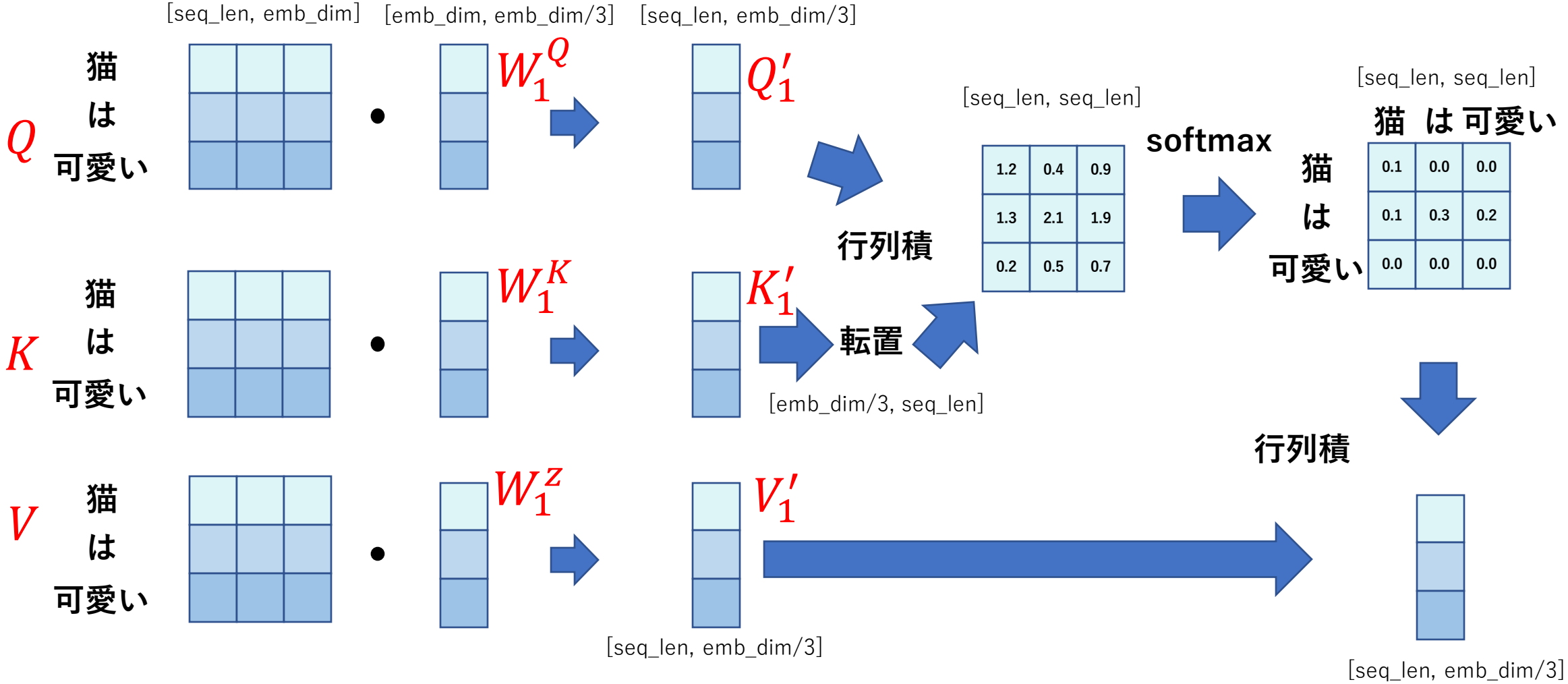
- 愚直にn並列にすると計算量もn倍
- 結果も同じ



各単語の特徴量が他の単語を考慮されて再構成

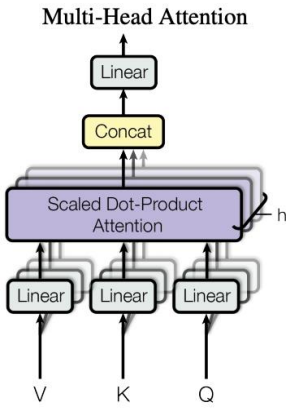
【構造解説】Multi-Head Attention(２)

線形写像で行列の次元を減らしてから並列化(図の例では3並列)



【構造解説】Multi-Head Attention(３)

線形写像で行列の次元を減らしてから並列化(図の例では3並列)



[seq_len, emb_dim] [emb_dim, emb_dim/3] [seq_len, emb_dim/3]

Q 猫 は 可愛い • W_i^Q → Q_i'

K 猫 は 可愛い • W_i^K → K_i'

V 猫 は 可愛い • W_i^V → V_i'

Attention

結合

[seq_len, emb_dim] [emb_dim, emb_dim]

• W^o

[seq_len, emb_dim]

出力

【構造解説】Multi-Head Attention(４)

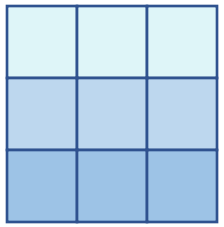
数式

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1,...,\mathrm{head}_\mathrm{h})W^O$$
$$\text{where } \mathrm{head}_\mathrm{i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

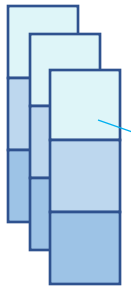
なぜMulti-Headなのか？

- Attentionする空間を選択でき、限定された特徴量空間でAttentionできる
- 単一Attentionでは平均的なAttentionとなる（大雑把なAttention）
- そして，単語の要素は１つではないのでMultiが望ましい（単語意味，品詞）

猫
は
可愛い



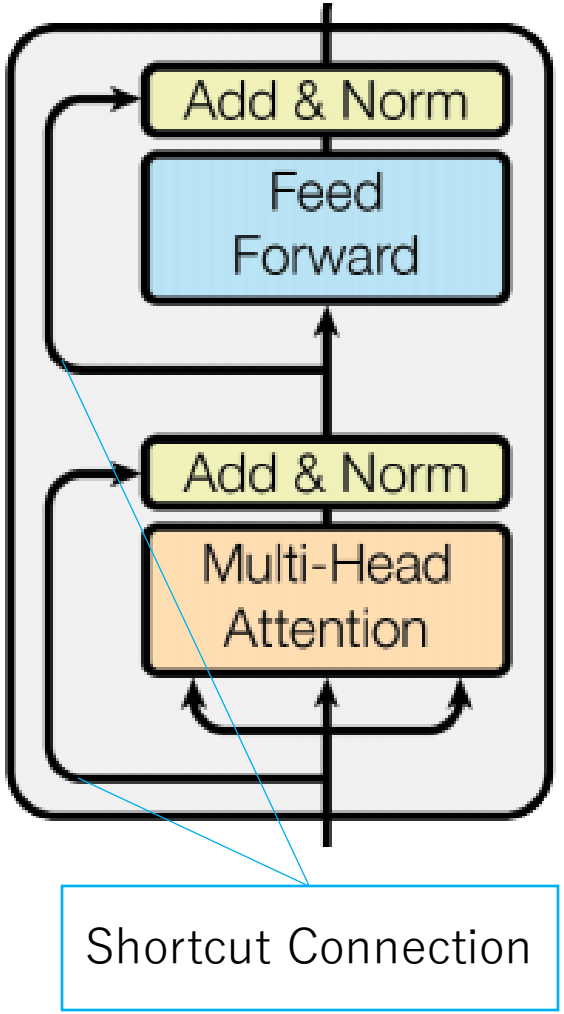
•



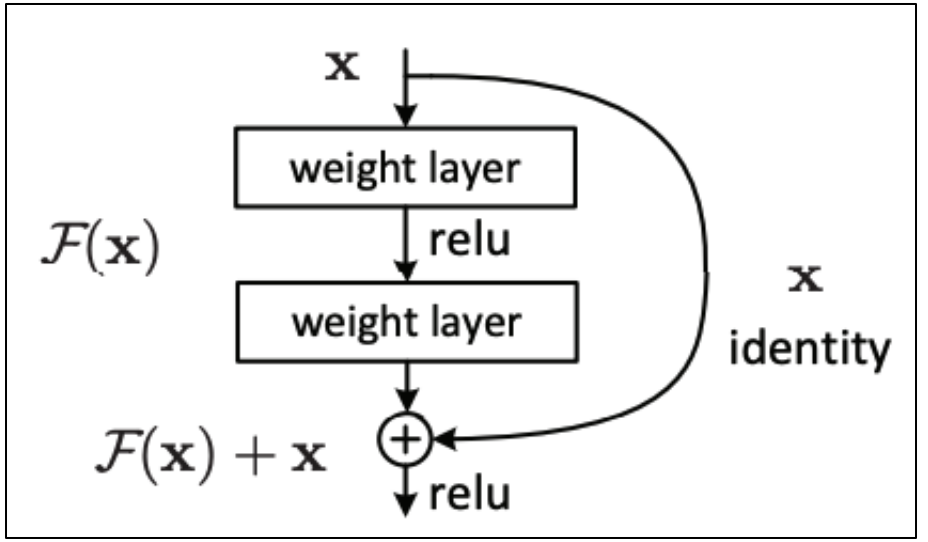
特徴量次元を圧縮 -> Attentionに利用する次元を学習

目次

【構造解説】Shortcut Connection



2015年に提案されたResidual Network(ResNet)[4]の要素

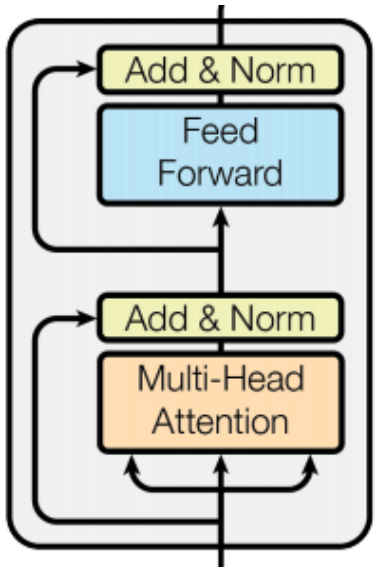


レイヤ部分は入力と出力の残差を予測すれば良く,
入力が理想的であれば値をほぼ変えずに出力できる

導入によりTransformerは層をより増やすことが可能

[4] He, Kaiming et al. “Deep Residual Learning for Image Recognition.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

【構造解説】Layer Normalization



NNのレイヤーの出力を平均0, 分散1に正規化する手法

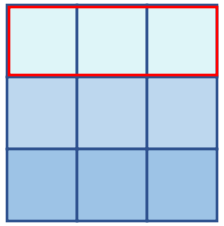
- 勾配消失・爆発対策
- 学習がより効率良く進む

$$\mu[t] = \frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} h[t,i] \tag{2}$$

$h[t,i]$: h番目のheadの
t番目の単語のi番目の特徴量
β, γ:学習可能なパラメータ

$$\sigma[t] = \sqrt{\frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} (h[t,i] - \mu[t])^2} \tag{3}$$

猫
は
可愛い

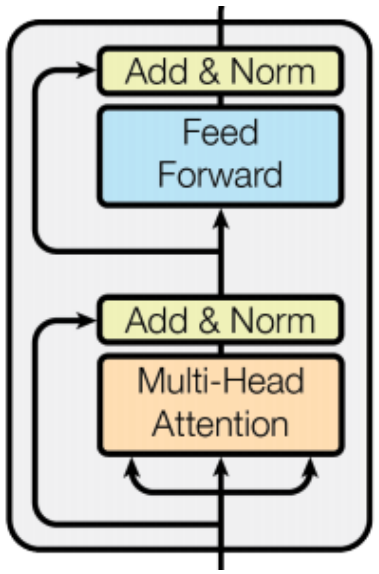


$$\text{LayerNorm}(h)[t] = \frac{\gamma}{\sigma[t]} \odot (h[t,i] - \mu[t]) + \beta \tag{4}$$

最終的にAdd & Normは次のように表記できる

$$\text{ResidualLayerNorm}(x) = \text{LayerNorm}(\text{sub\_block}(x) + x)$$

【構造解説】Position-wise Feed-Forward Networks

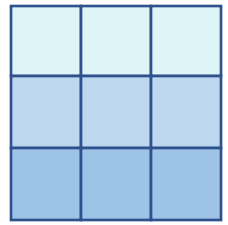


各単語ごとに隔離された全結合層

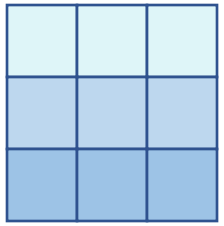
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

[seq_len, emb_dim] [emb_dim, emb_dim*4] [emb_dim*4, emb_dim]

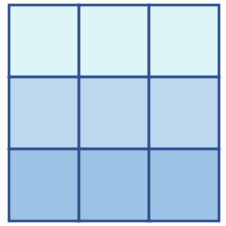
猫
は
可愛い



$\bullet$



W_1



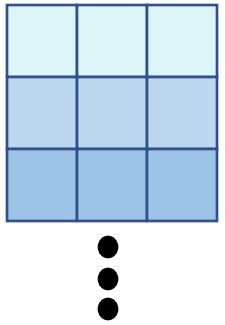
W_2

各単語への重みは共有（図で言うと横方向に同じ重みが並ぶ）

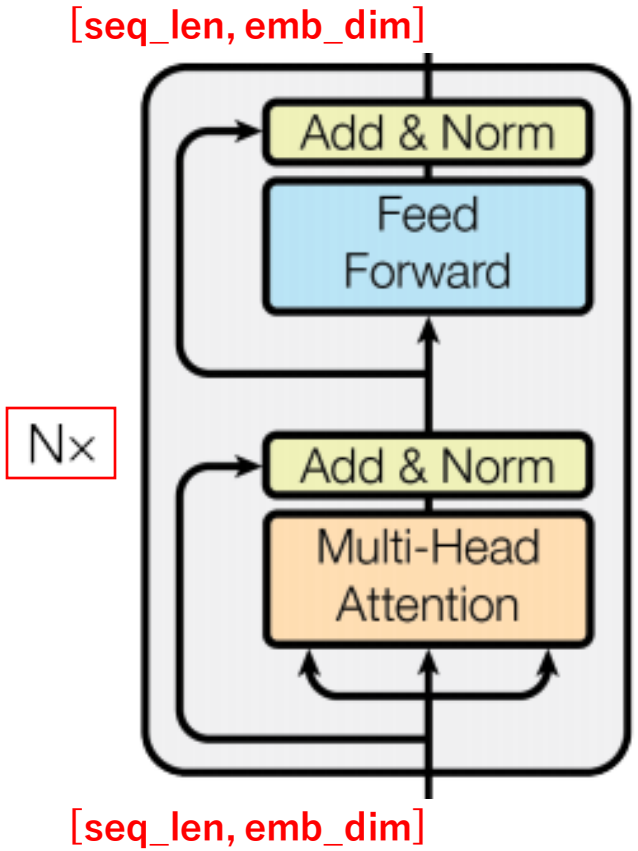
普通の全結合層

[seq_len * emb_dim] [seq_len * emb_dim, emb_dim*4]

$\bullet$



【構造解説】Transformer Moduleのまとめ



Multi-Head Attention

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1,...,\mathrm{head}_\mathrm{h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Add &Norm

$$\mathrm{ResidualLayerNorm}(x) = \mathrm{LayerNorm}(\mathrm{sub\_block}(x) + x)$$

Feed Forward

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

各サブレイヤーの出力次元は入力次元と同一，実際にはこのModuleを複数層重ねる

目次

【構造解説】Transformerの全体図(再掲)

エンコーダ部分

デコーダ部分

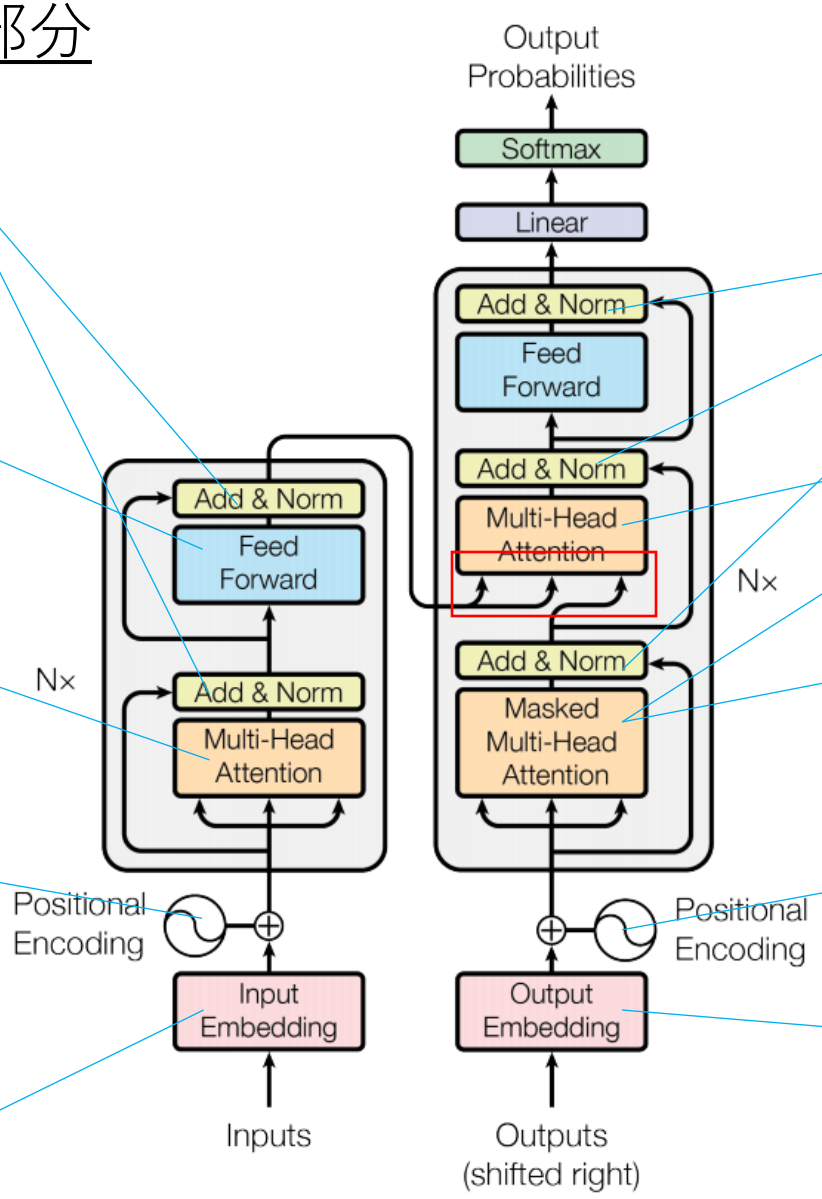
残差演算＋
レイヤー正規化

全結合層

Attentionによる
特徴量抽出

時系列情報の付与

単語のベクトル化



残差演算＋
レイヤー正規化

Attentionによる
特徴量抽出

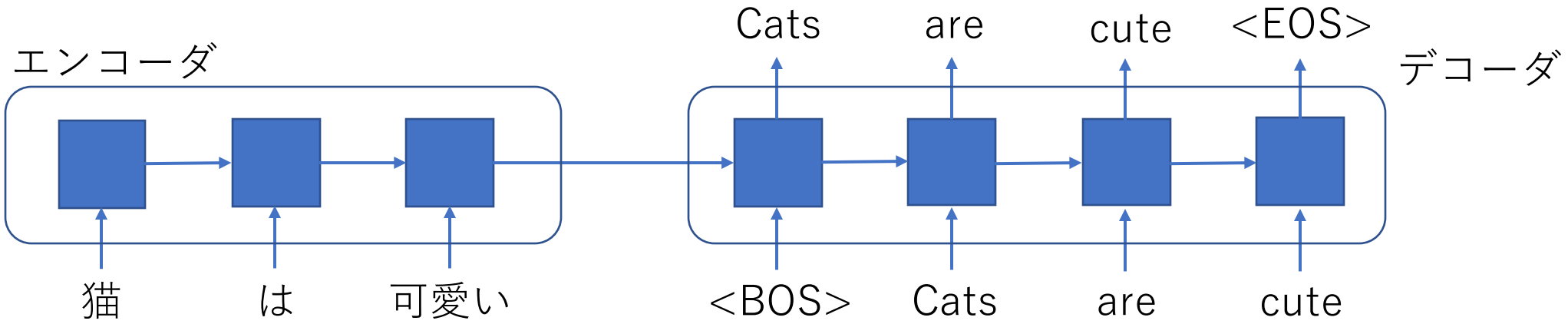
未来の情報のマスク

時系列情報の付与

単語のベクトル化

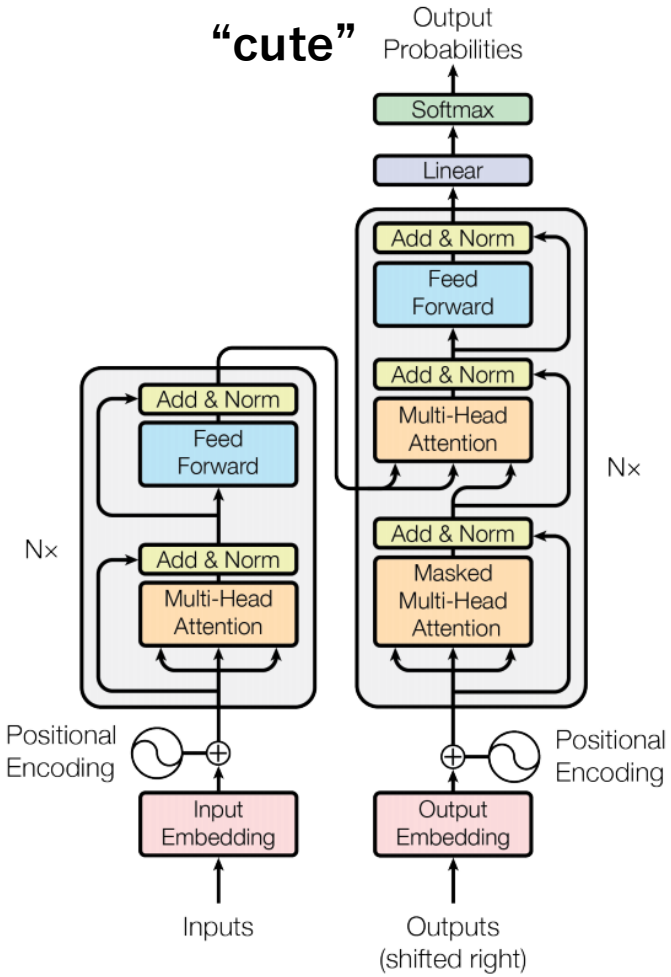
【構造解説】RNN系の推論方法

従来のSeq2Seq型のRNN翻訳モデル



出力されたものをデコーダの入力に再度入力して得ていく

【構造解説】Transformer Decoderの推論（イメージ）



1. Decoderの入力として[“<BOS>”,””,””]を入れる
2. Outputに”Cats”が出力される
3. Decoderの入力として[“<BOS>”,”Cats”,””]を入れる
4. Outputに”are”が出力される

⋮

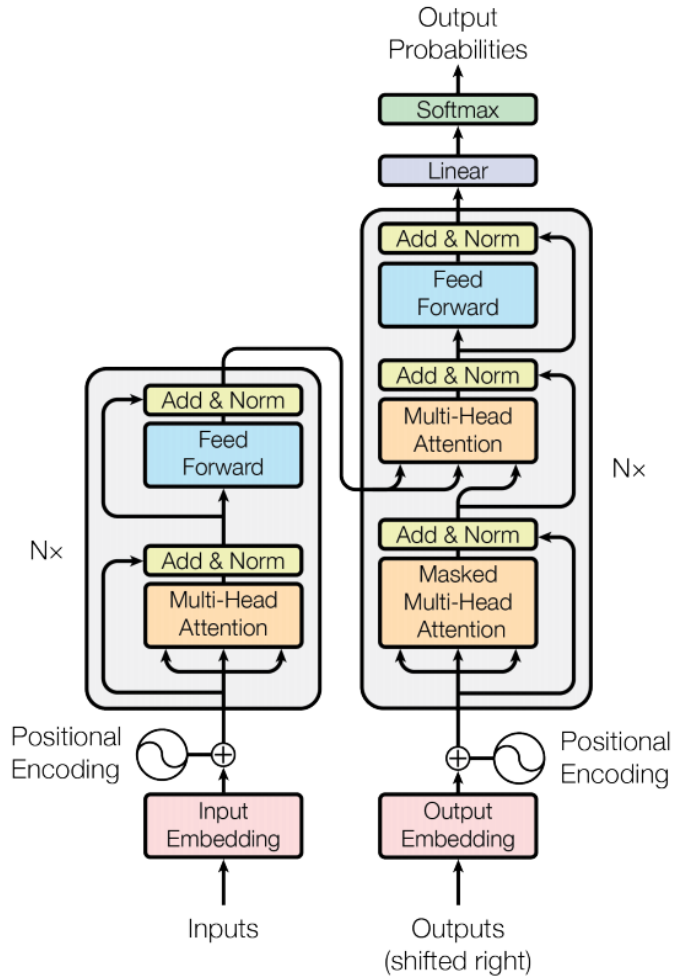
RNN系の推論と類似（自己回帰）

入力例：[“猫”, “は”, “可愛い”]

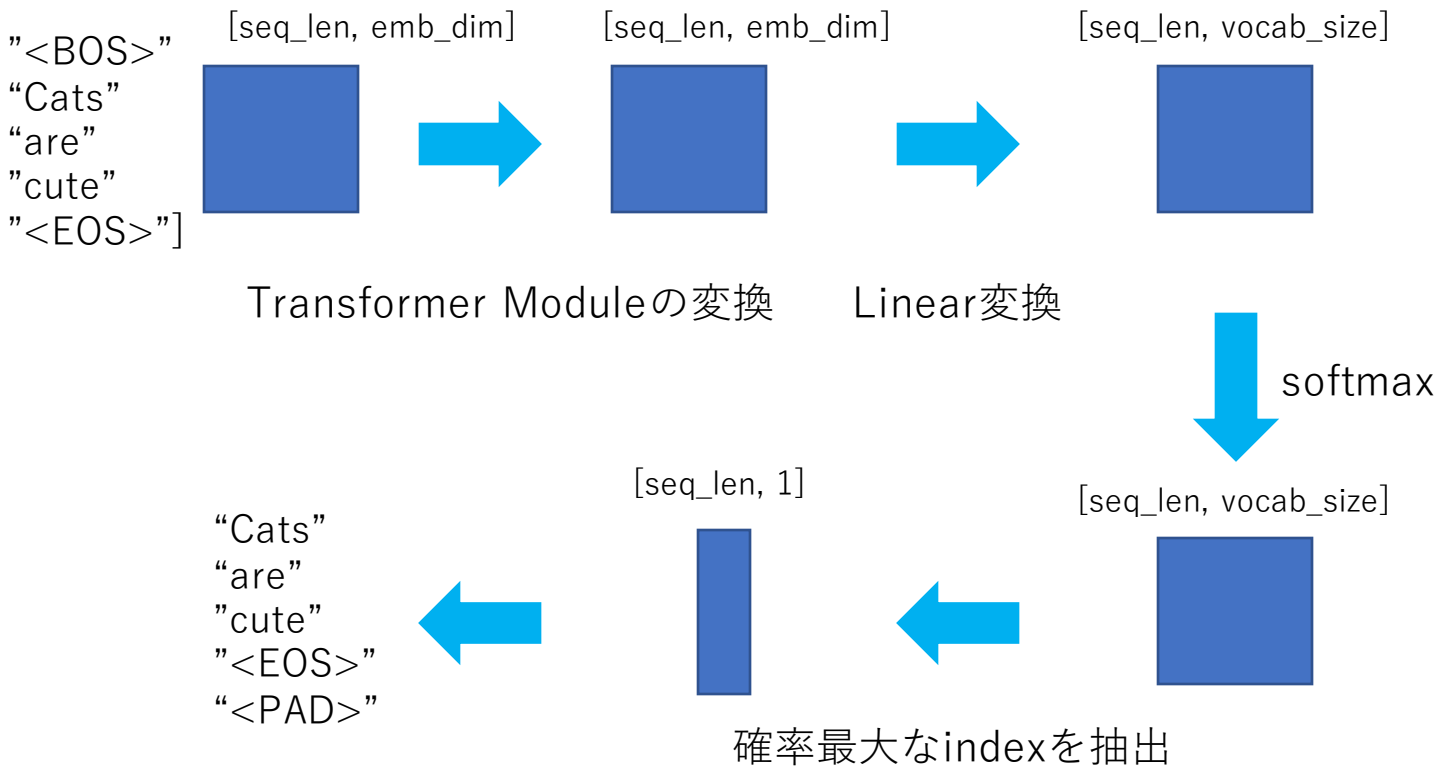
[”<BOS>”,“Cats”, “are”]

【構造解説】 Transformer Decoderの学習

[“Cats”, ”are”, “cute”, “<EOS>”, “<PAD>”]

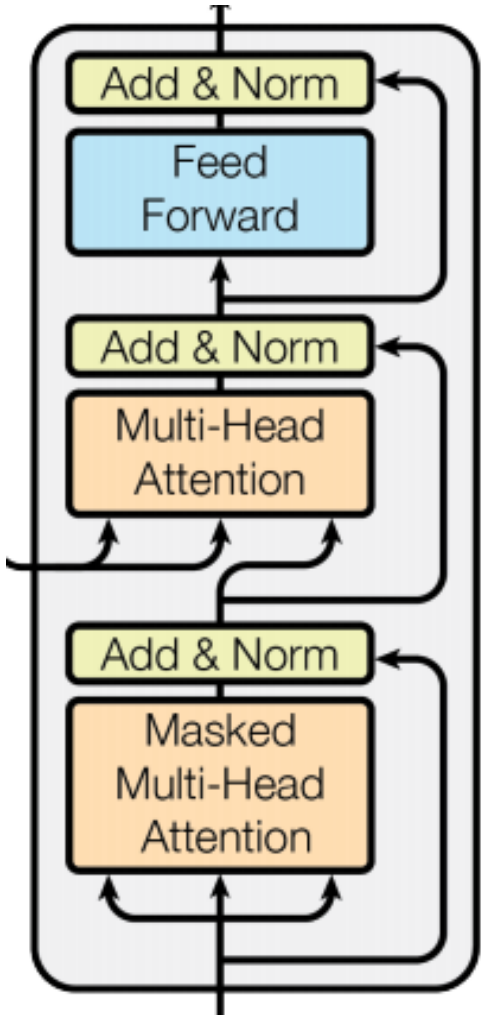


学習段階では時系列の予測を並列に1度で実施



[“猫”, “は”, “可愛い”]　**[”<BOS>”,“Cats”, “are”, ”cute”,”<EOS>”]**

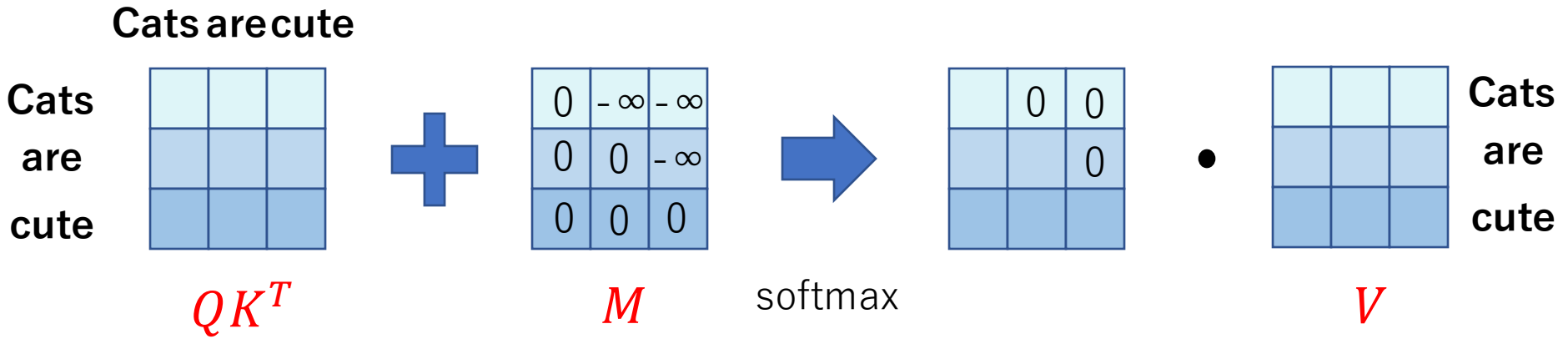
【構造解説】Masked Attention



学習段階では時系列の予測を並列に１度で実施
そのままだと正解データが入力に含まれている！

Attentionにおいて未来の単語をマスクする演算を追加

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



目次

【実験結果】機械翻訳タスクの実験結果

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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**	**41.8**	$2.3 \cdot 10^{19}$	

既存の手法よりも高いスコア　＆　学習コストも

[1] Vaswani, Ashish et al. “Attention is All you Need.” *ArXiv* abs/1706.03762 (2017)

【実験結果】Self-Attentionの計算量

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

1. $n < d$の場合は，Self-Attentionの計算量はRNNよりも小さくなる
2. RNNは前の値を待つ必要があるので計算時間はよりかかる
3. Self-Attentionは長期の依存関係を学習しやすい

【実験結果】各パラメータの精度への影響

Transformer Module数　特徴量次元　全結合層の次元　head数　Attentionの写像次元　Dropout率 label smoothing

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Transformerまとめ

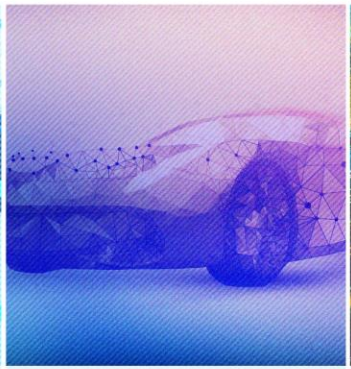
1. TransformerはAttentionベース構造で計算量・GPU親和性が高いモデル
2. 時系列データを一度に処理するためPositional Encodingを実施する必要
3. TransformerのAttentionは以下の要素を持つ
    1. Self-Attention（入力同士を用いて特徴量を再構築）
    2. Scaled Dot-Product Attention（勾配情報を保持しつつ内積する）
    3. Multi-Headed Attention（複数のヘッドを用いて並列にAttention）
4. デコーダでは未来の情報のリークを防ぐためにMasked Attentionする
5. 実験結果からTransformerの高性能，低計算量が示された

Arithmer AI Systems

OCR Inspection Dynamics NLP DB R3 Robo

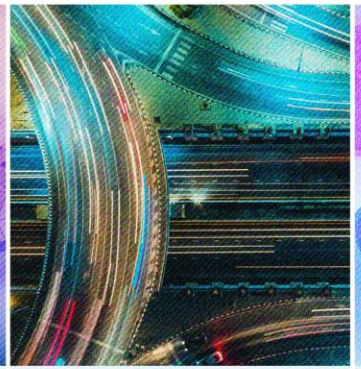
Arithmer OCR

- AI OCRシステム
- 本人確認書類 AI OCRシステム



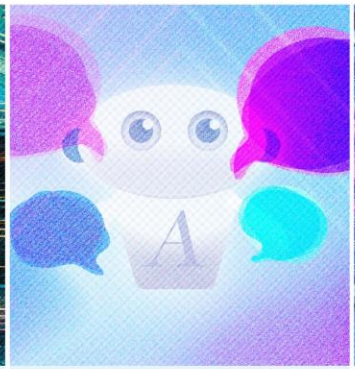
Arithmer Inspection

- 損傷箇所AI検知システム
- 画像AI検品システム



Arithmer Dynamics

- 運転支援AIシステム
- 時空間検知AIシステム



Arithmer NLP

- AIチャットボットシステム
- テキストマイニング AIシステム
- 文章自動生成AIシステム

Arithmer DB

- 最適化レコメンド AIシステム
- 危機管理ＡＩシステム

Arithmer R3

- 自動採寸AIシステム
- 流体予測AIシステム
- 歯科設計自動化 AIシステム

Arithmer Robo

- ３Ｄビジョンロボット システム

Arithmer 株式会社
東京都港区六本木一丁目 6 番 1 号 泉ガーデンタワー 38/40F
TEL : 03-5579-6683
https://arithmer.co.jp/

Arithmer