



# Section3の概要

สวัสดิ์

こんにちは

여보세요

CIAO

HOLA

你好

BONJOUR

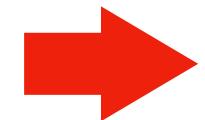
HALLO

HELLO

# 講座の内容

Section 1. 講座とBERTの概要

Section 2. シンプルなBERTの実装



**Section 3. BERTの仕組み**

Section 4. ファインチューニングの活用

Section 5. BERTの応用

# 今回の内容

- 
1. Section3の概要
  2. BERTの全体像
  3. Transformerのモデル
  4. BERTの実装

# 教材の紹介

•01\_bert\_model.ipynb

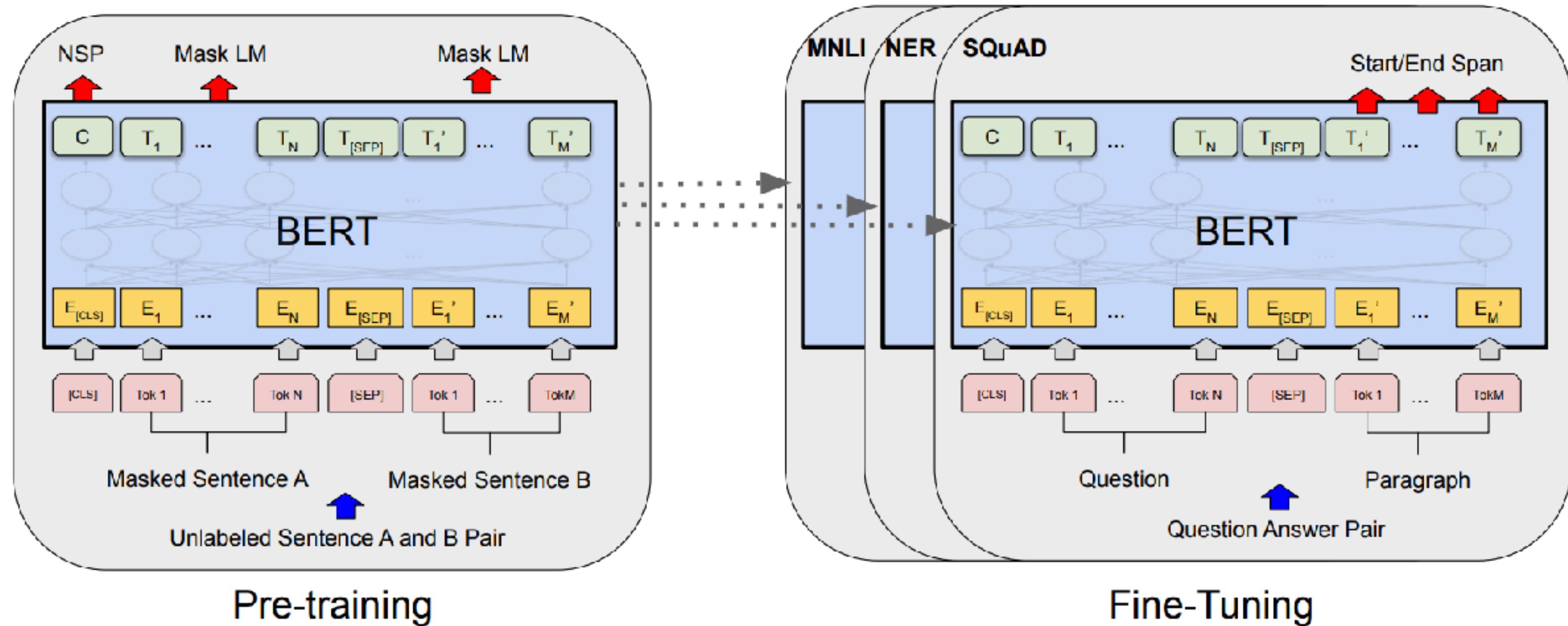
# BERTの全体像



# BERTの概要

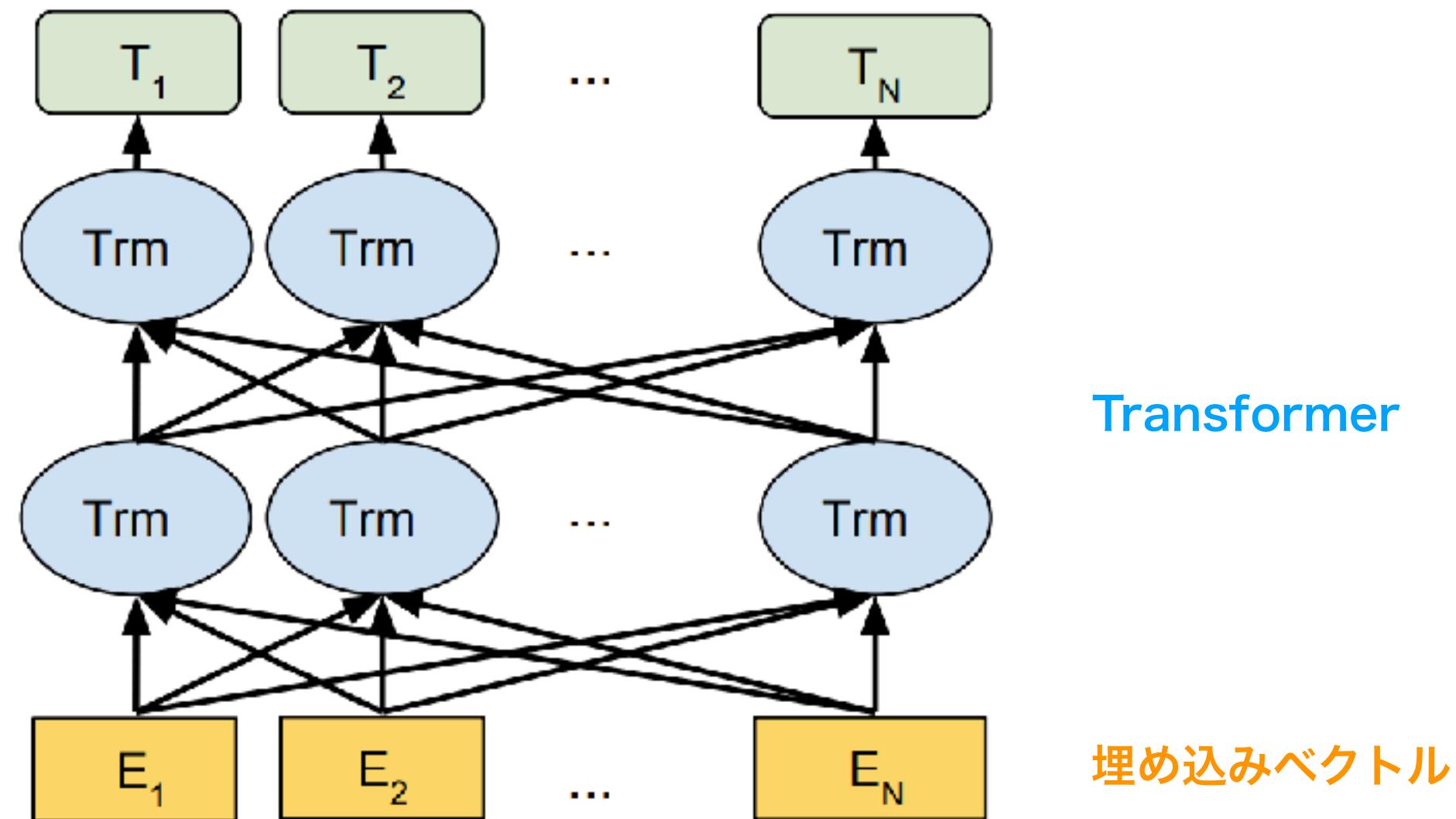
- **BERT (Bidirectional Encoder Representation from Transformers) とは？**
  - 2018年の後半にGoogleから発表された、  
自然言語処理のための新たなディープラーニングのモデル
  - Transformerがベースとなっている
  - 様々な自然言語処理タスクでファインチューニングが可能
  - 従来の自然言語処理タスクと比較して、高い汎用性

# BERTの学習

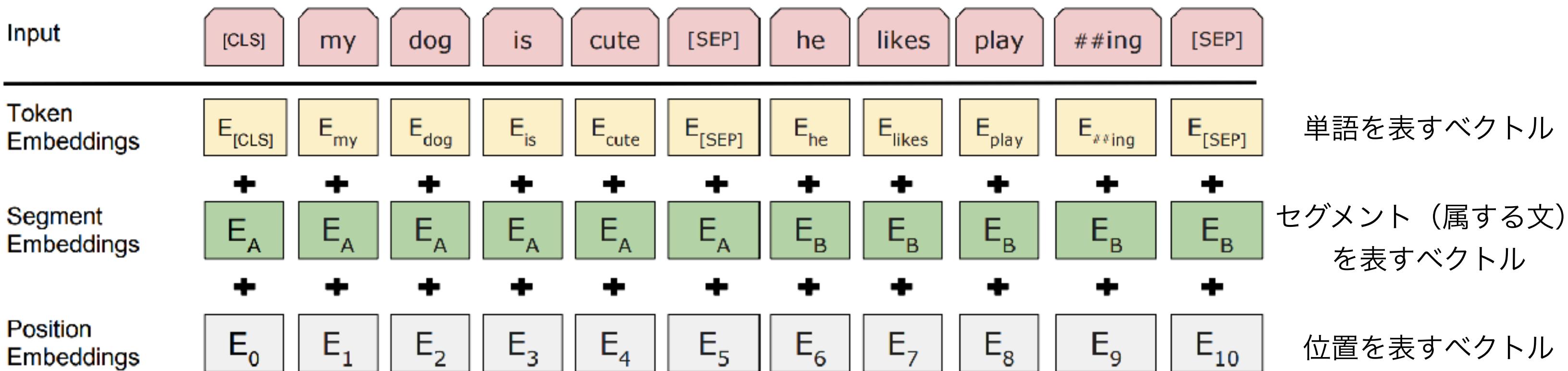


BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin, J. et al. (2018) より引用

# BERTのモデル



# BERTの入力



# BERTの学習

- 事前学習
  - Transformerが、文章から文脈を双方向 (Bidirectional) に学習する
  - Masked Language ModelおよびNext Sentence Predictionによる双方向学習
- フайнチューニング
  - 事前学習により得られたパラメータを初期値として、ラベル付きのデータで  
ファインチューニングを行う

# 事前学習 - Masked Language Model -



- **Masked Language Model**
  - 文章から特定の単語を15%ランダムに選び、[MASK]トークンに置き換える
  - 例: my dog is hairy → my dog is [MASK]
  - [MASK]の単語を、前後の文脈から予測する

# 事前学習 - Next Sentence Prediction -

- **Next Sentence Prediction**
  - 2つの文章に関係があるかどうかを判定する
  - 後ろの文章を50%の確率で無関係な文章に置き換える
  - 後ろの文章が意味的に適切であればIsNext、そうでなければNotNextの判定
- [CLS] the man went to [MASK] store [SEP] / he bought a gallon [MASK] milk [SEP]  
判定 : IsNext

[CLS] the man went to [MASK] store [SEP] / penguin [MASK] are flight #less birds  
[SEP]

判定 : NotNext

# BERTの性能

- **SQuAD**
  - 「Stanford Question Answering Dataset」の略
  - スタンフォード大学が一般公開している言語処理の精度を測るベンチマーク
  - データは約10万個の質問応答のペアを含む

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT <sub>BASE</sub> (Single)	80.8	88.5	-	-
BERT <sub>LARGE</sub> (Single)	84.1	90.9	-	-
BERT <sub>LARGE</sub> (Ensemble)	85.8	91.8	-	-
BERT <sub>LARGE</sub> (Sgl.+TriviaQA)	<b>84.2</b>	<b>91.1</b>	<b>85.1</b>	<b>91.8</b>
BERT <sub>LARGE</sub> (Ens.+TriviaQA)	<b>86.2</b>	<b>92.2</b>	<b>87.4</b>	<b>93.2</b>

# BERTの性能

- **GLUE**

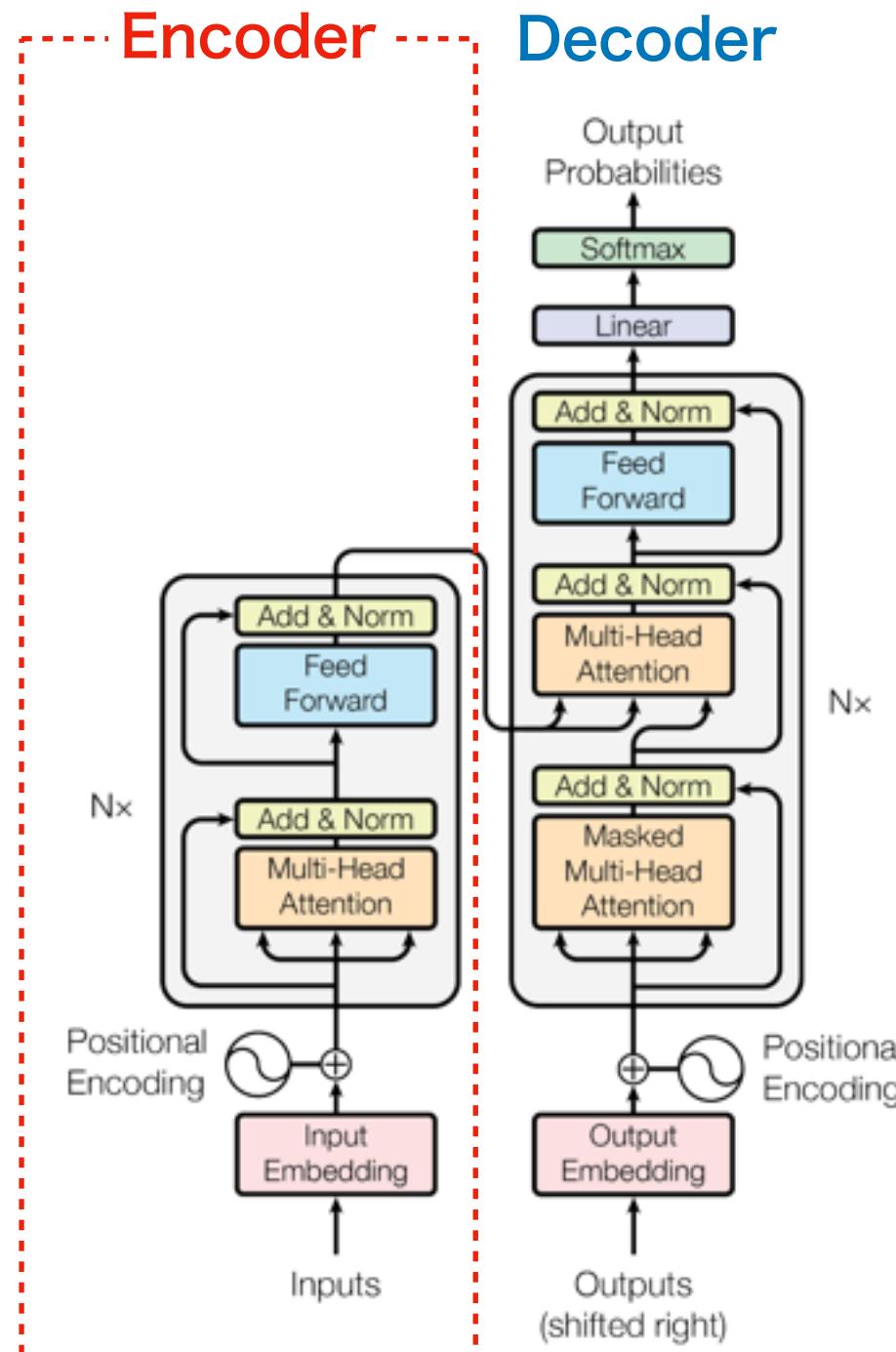
→ 自然言語処理のための9種類の学習データを含むデータセット

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

# Transformerのモデル



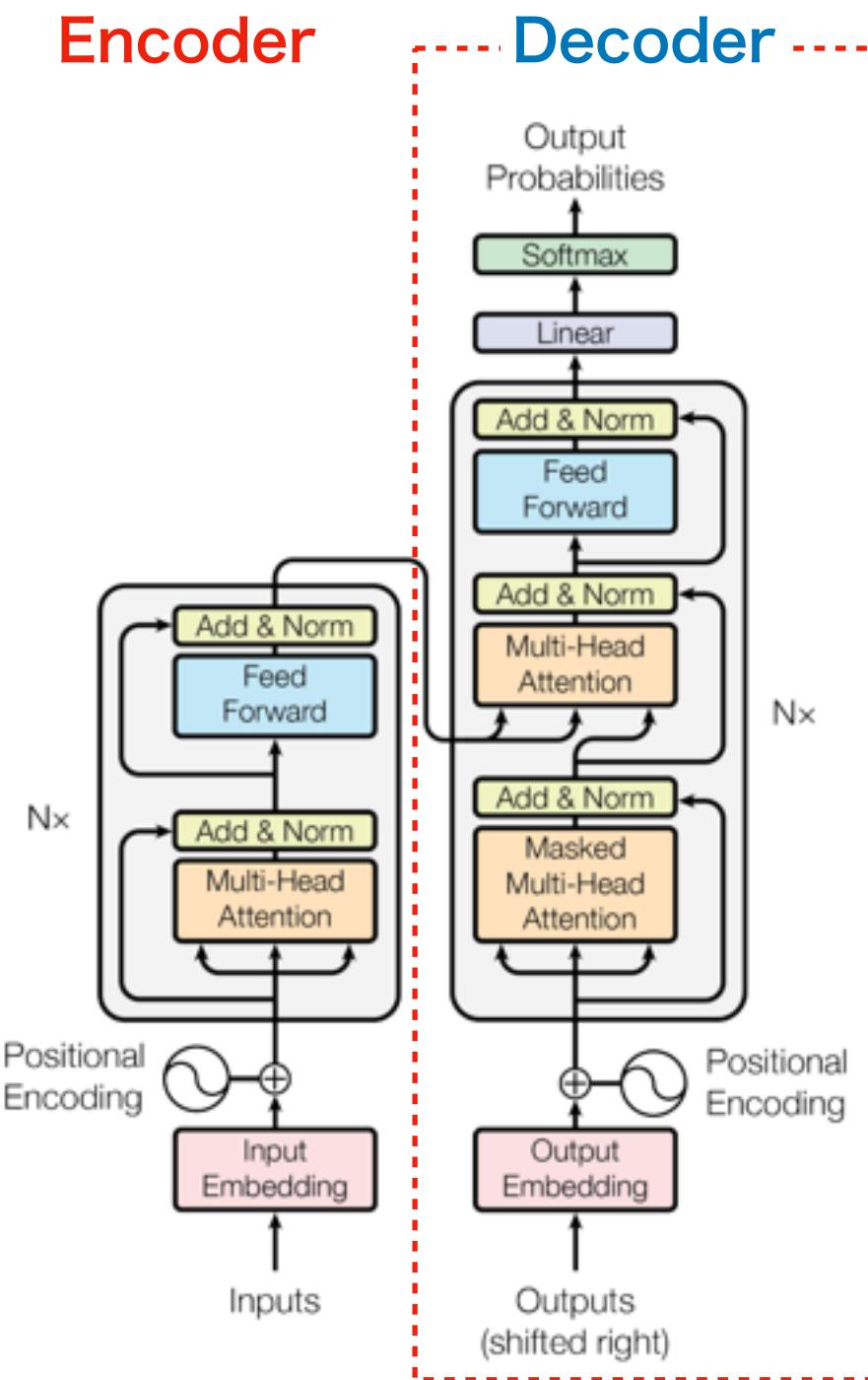
# Transformerのモデル



## Encoderの構造

1. Embedding層により入力文章をベクトルに圧縮
  2. Positional Encoder層によって位置情報を加える
  3. Multi-Head Attention層
  4. normalization (正規化) など
  5. Positionwise fully connected feed-forward network
  6. normalization (正規化) など
- 3-6を6回繰り返す

# Transformerのモデル



## Decoderの構造

1. Embedding層により入力文章をベクトルに圧縮
  2. Positional Encoder層によって位置情報を加える
  3. Multi-Head Attention層
  4. normalization (正規化) など
  5. Multi-Head Attention層 (Encoderの入力を使用)
  6. normalization (正規化) など
  7. Positionwise fully connected feed-forward network
  8. normalization (正規化) など
- 3-8を6回繰り返す

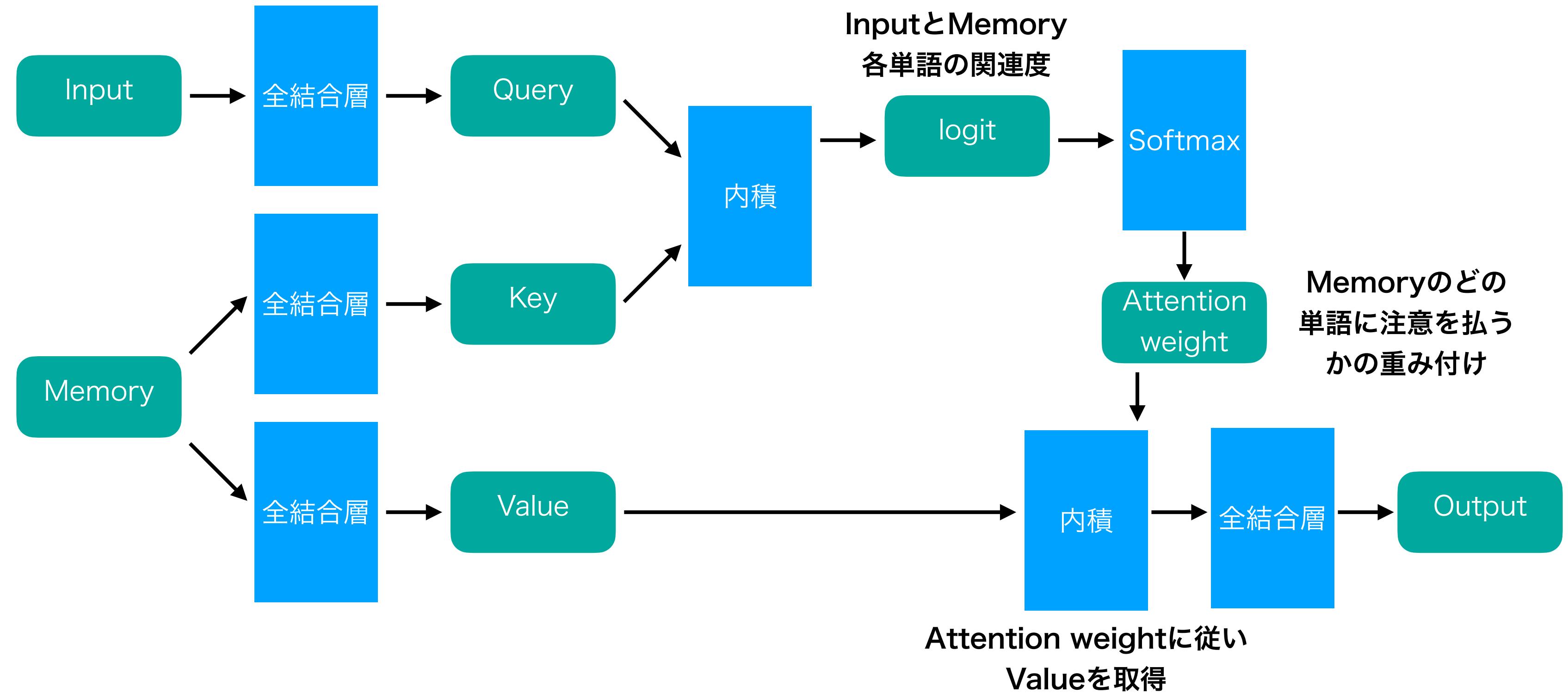
# Transformerの構成要素

- **Attention**
  - Self-Attention
  - SourceTarget-Attention
  - Masked Multi-Head Attention
  - Multi-Head Attention
- **Position-wise Feedforward Network**
- **Positional Encoding**

# Attentionとは？

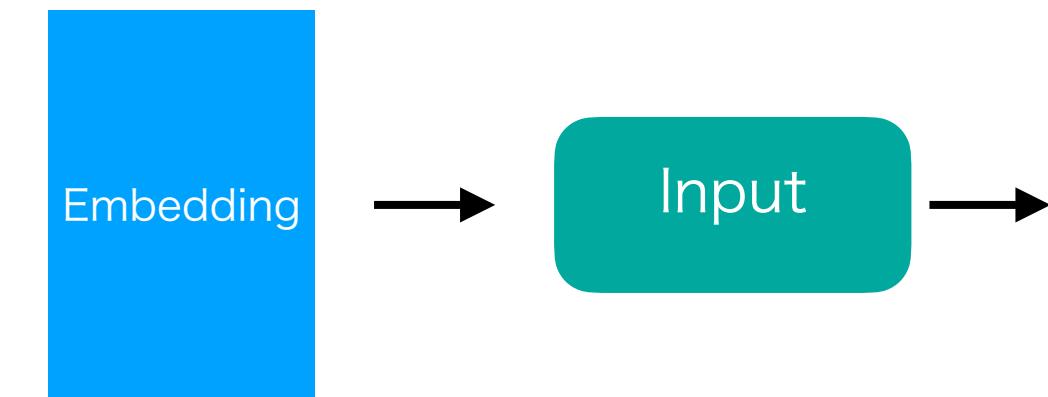
- **Attention**とは？
  - 文章中のどの単語に注目すればいいかを表すスコア
  - Query、Key、Valueの3つのベクトルで計算される
- **Query**
  - Inputのうち「検索をかけたいもの」
- **Key**
  - 検索対象とQueryの近さを測る
- **Value**
  - Keyに基づき、適切なValueを出力する

# Attentionとは？

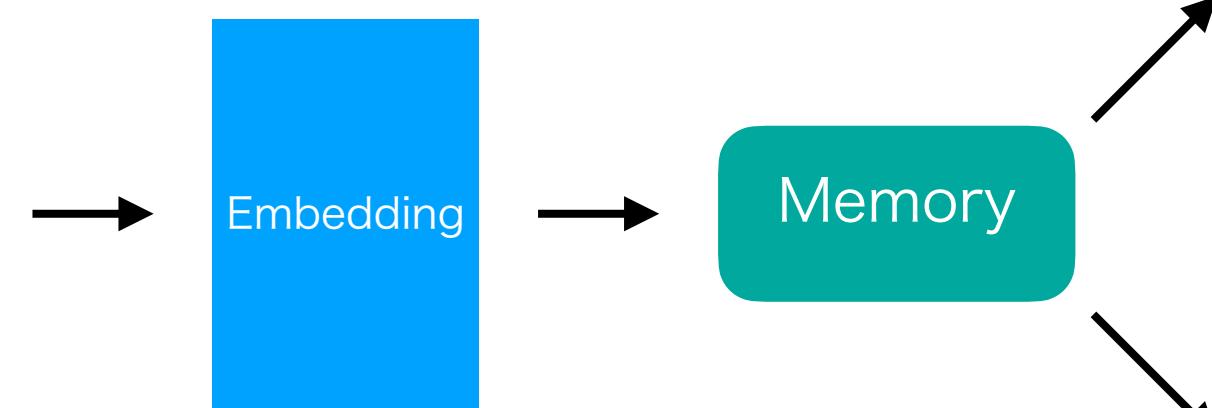


# InputとMemory

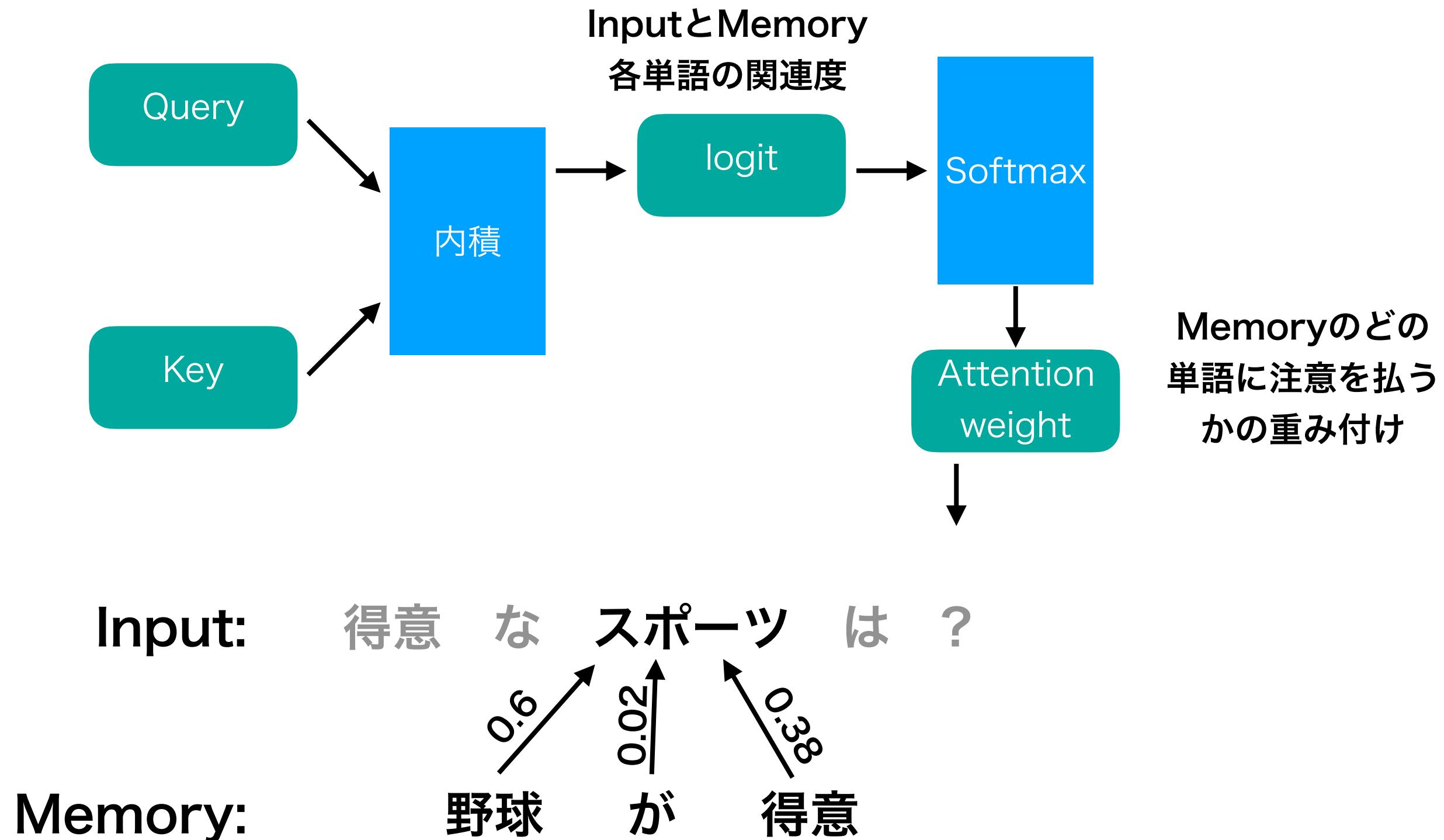
得意 な スポーツ は ? →  
(各単語はidで表される)



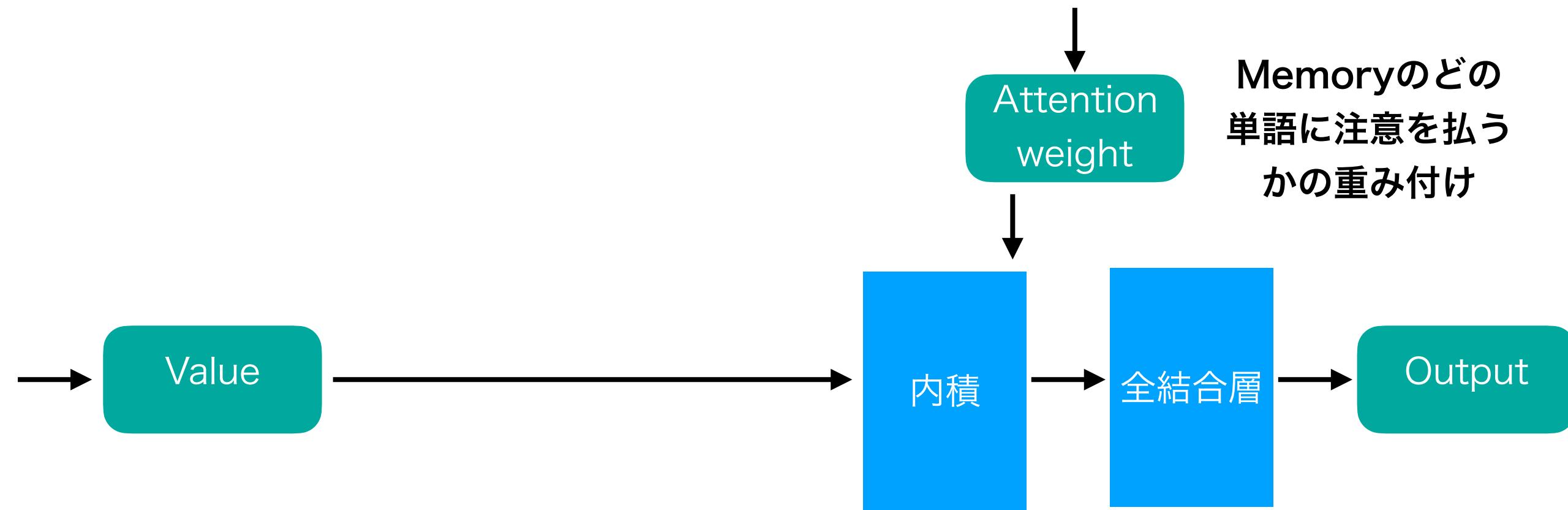
野球 が 得意  
(各単語はidで表される)



# Attention weightの計算

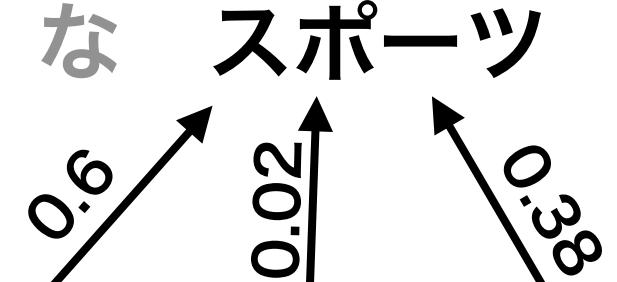


# Valueとの内積



Attention weightに従い  
Valueを取得

Input: 得意 な スポーツ は ?



Memory:

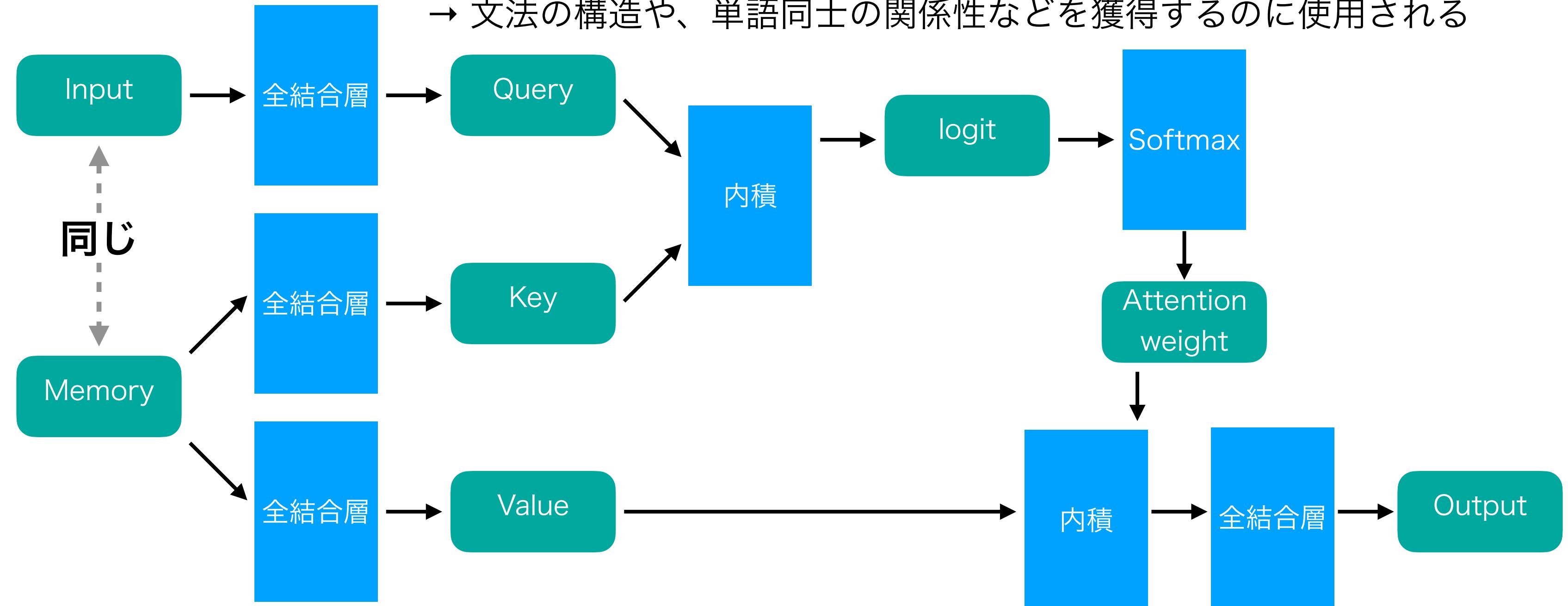
$$\begin{aligned} \text{内積} = & \text{Value(野球)} \times 0.6 \\ & + \text{Value(が)} \times 0.02 \\ & + \text{Value(得意)} \times 0.38 \end{aligned}$$

# Self-Attention

- **Self-Attention**

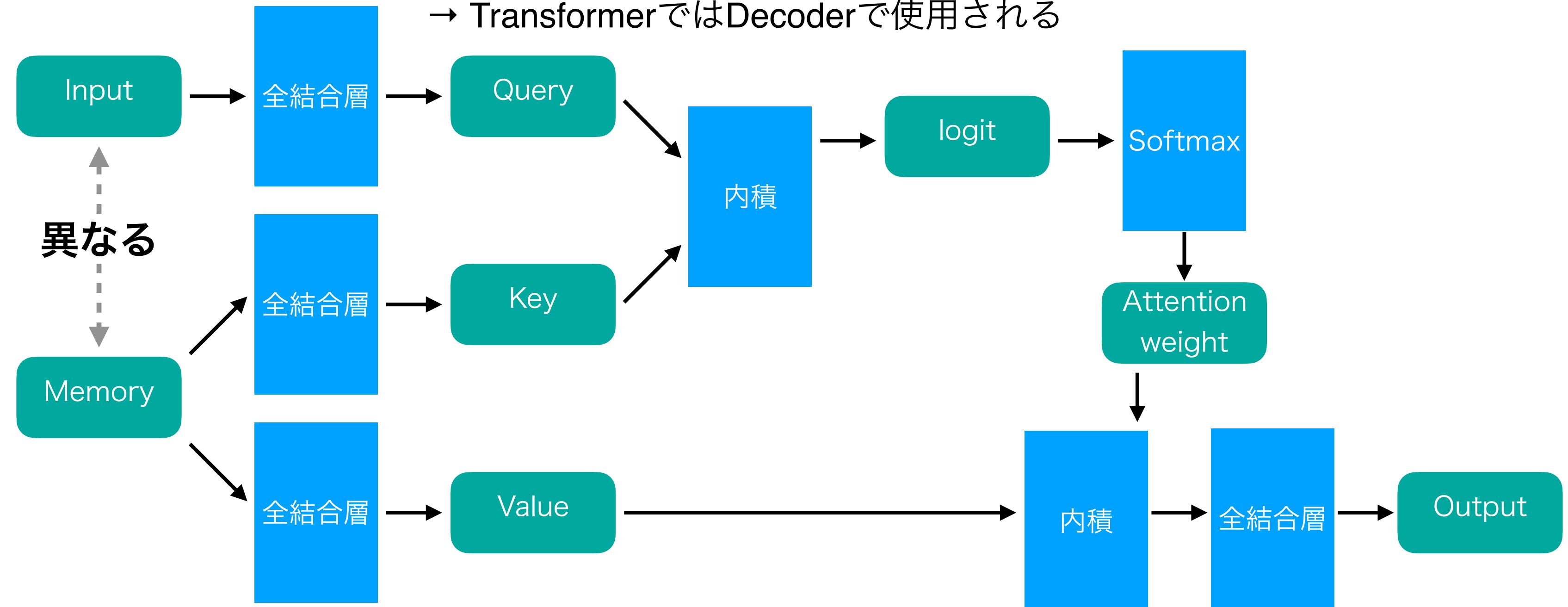
→ InputとMemoryが同一のAttention

→ 文法の構造や、単語同士の関係性などを獲得するのに使用される



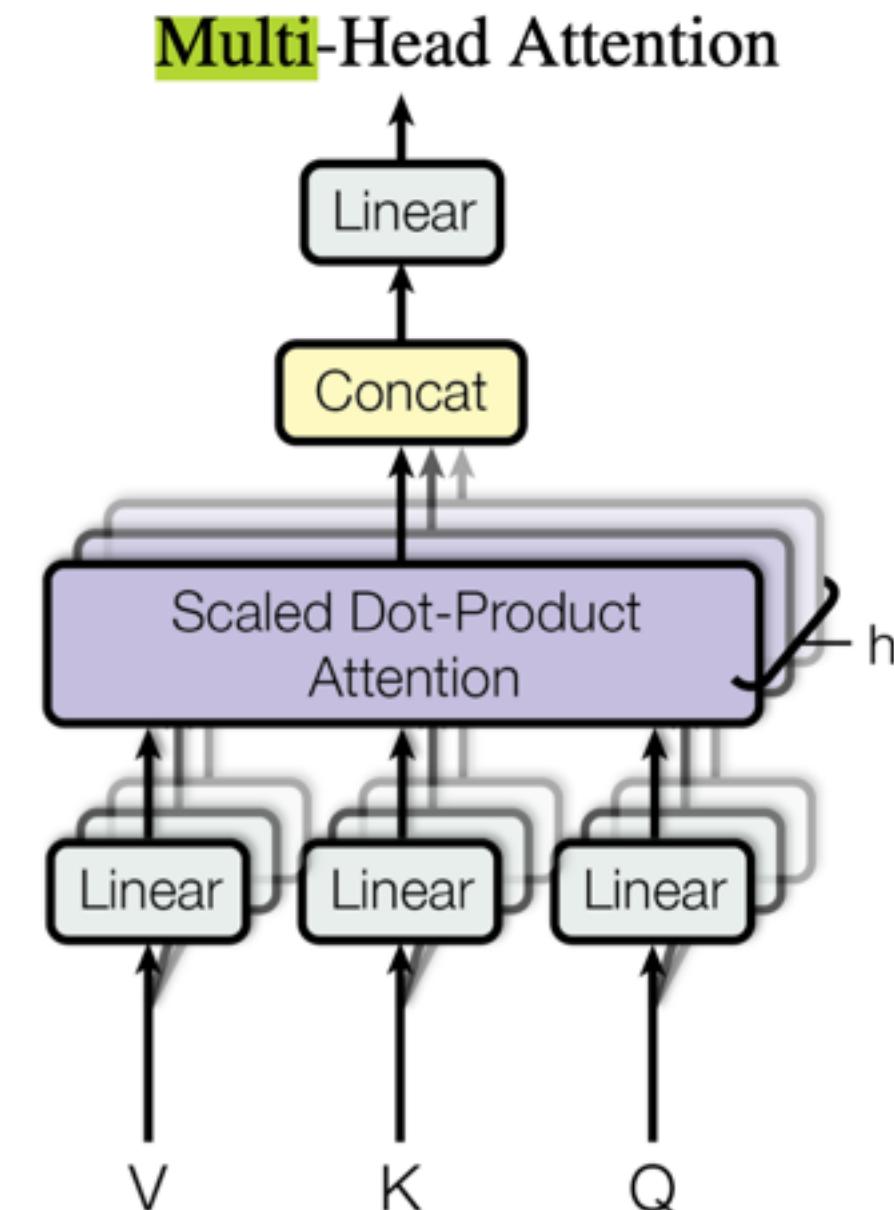
# SourceTarget-Attention

- **Self-Attention**
  - InputとMemoryが異なるAttention
  - TransformerではDecoderで使用される



# Multi-Head Attention

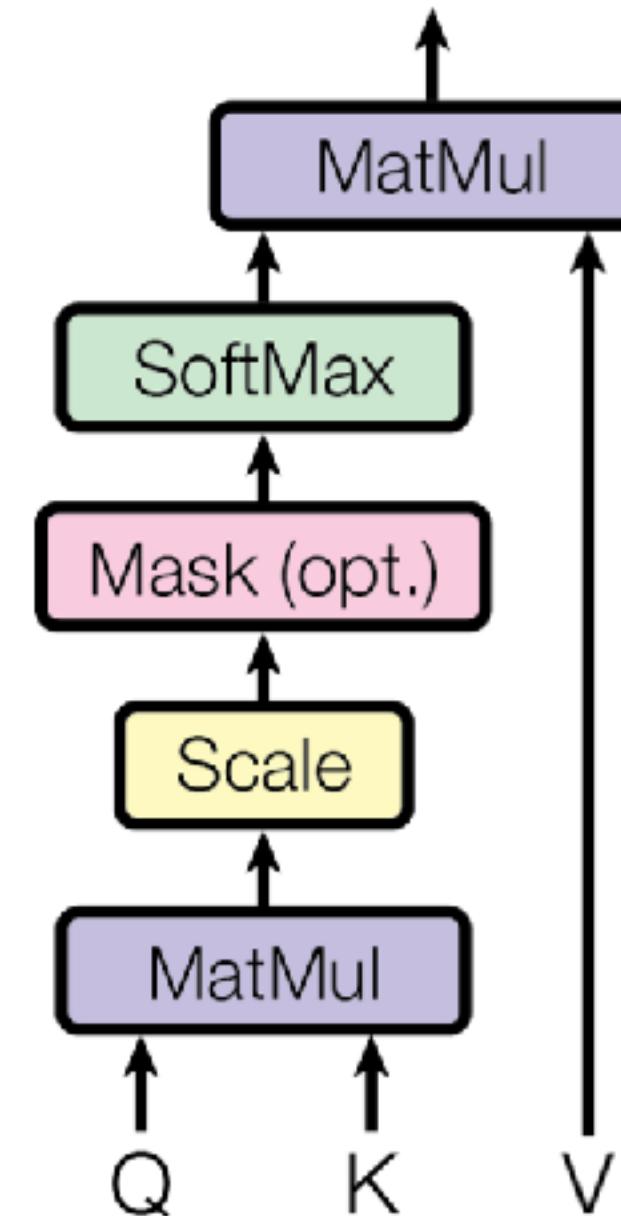
- **Multi-Head Attention**
  - Attentionを並行に並べる
  - それぞれのAttentionはHeadと呼ばれる
  - 「Attention Is All You Need」では  
Multi-Head化による性能の向上が述べられている



Attention Is All You Need, Ashish, V. et al. (2017) より引用

# Masked Multi-Head Attention

- **Masked Multi-Head Attention**
  - 特定の key に対して、Attention weight を0にする
  - TransformerではDecoderで使われる
  - 入力した単語が先読みを防ぐために、情報をマスクで遮断する
  - 言わば、「カンニング」を防ぐ



# Positionwise fully connected feed-forward network

- **Positionwise fully connected feed-forward network**
  - 2層の全結合ニューラルネットワーク
  - 単語の位置ごとに個別の順伝播ネットワーク
  - 他単語との影響関係を排除
  - パラメータは全てのネットワークで共通

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

Attention Is All You Need, Ashish, V. et al. (2017) より引用

# Positional Encoding

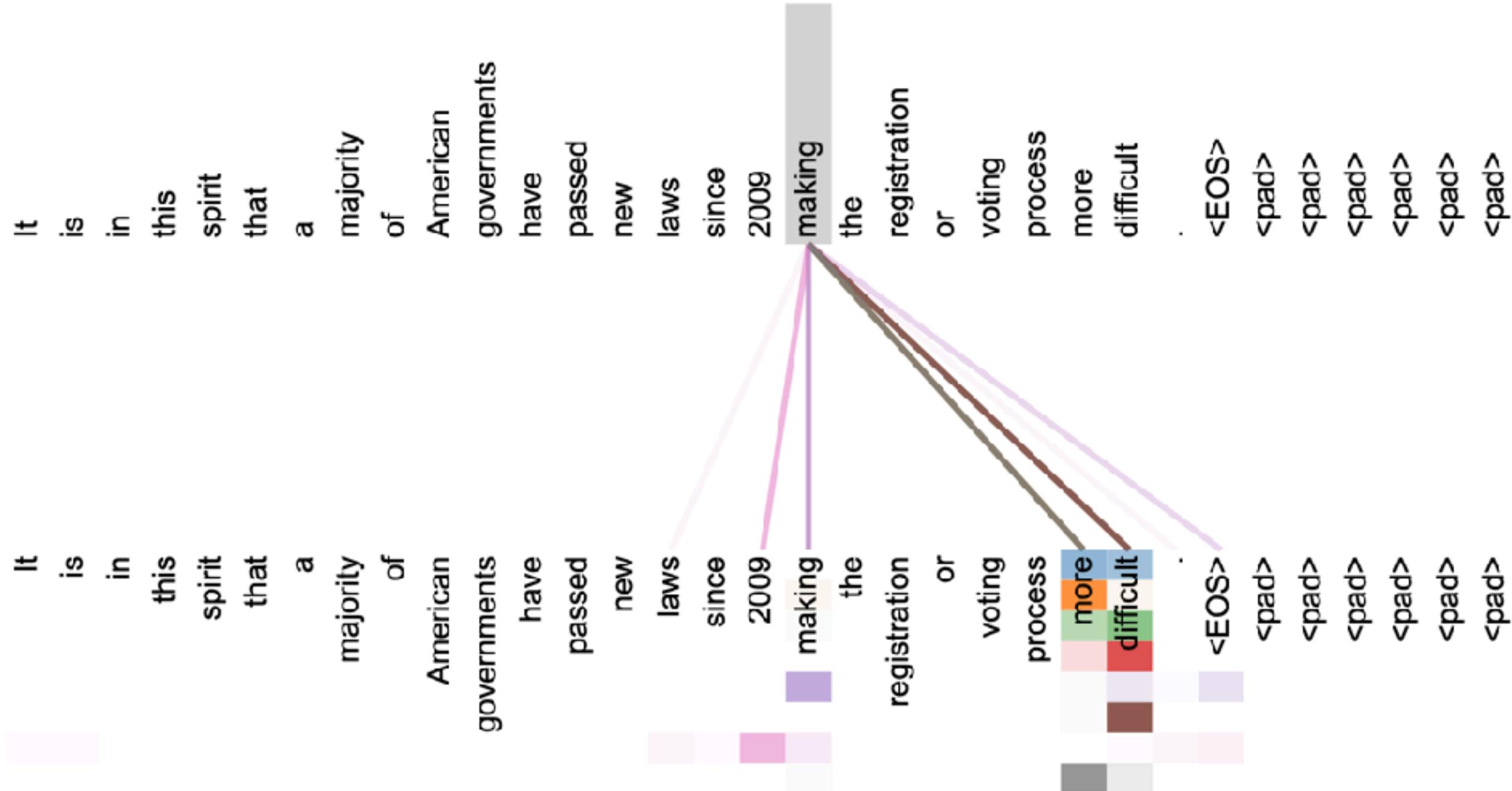
- **Positional Encoding**  
→ 「単語の位置」 の情報を加える

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

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

pos: 単語の位置    2i, 2i+1: Embedding の何番目の次元か     $d_{\text{model}}$ : 次元数

# Attentionの可視化



異なる色は異なるAttentionのHeadを表す

Attention Is All You Need, Ashish, V. et al. (2017) より引用

# BERTの実装

สวัสดิ์

こんにちは

여보세요

CIAO

HOLA

你好

HALLO

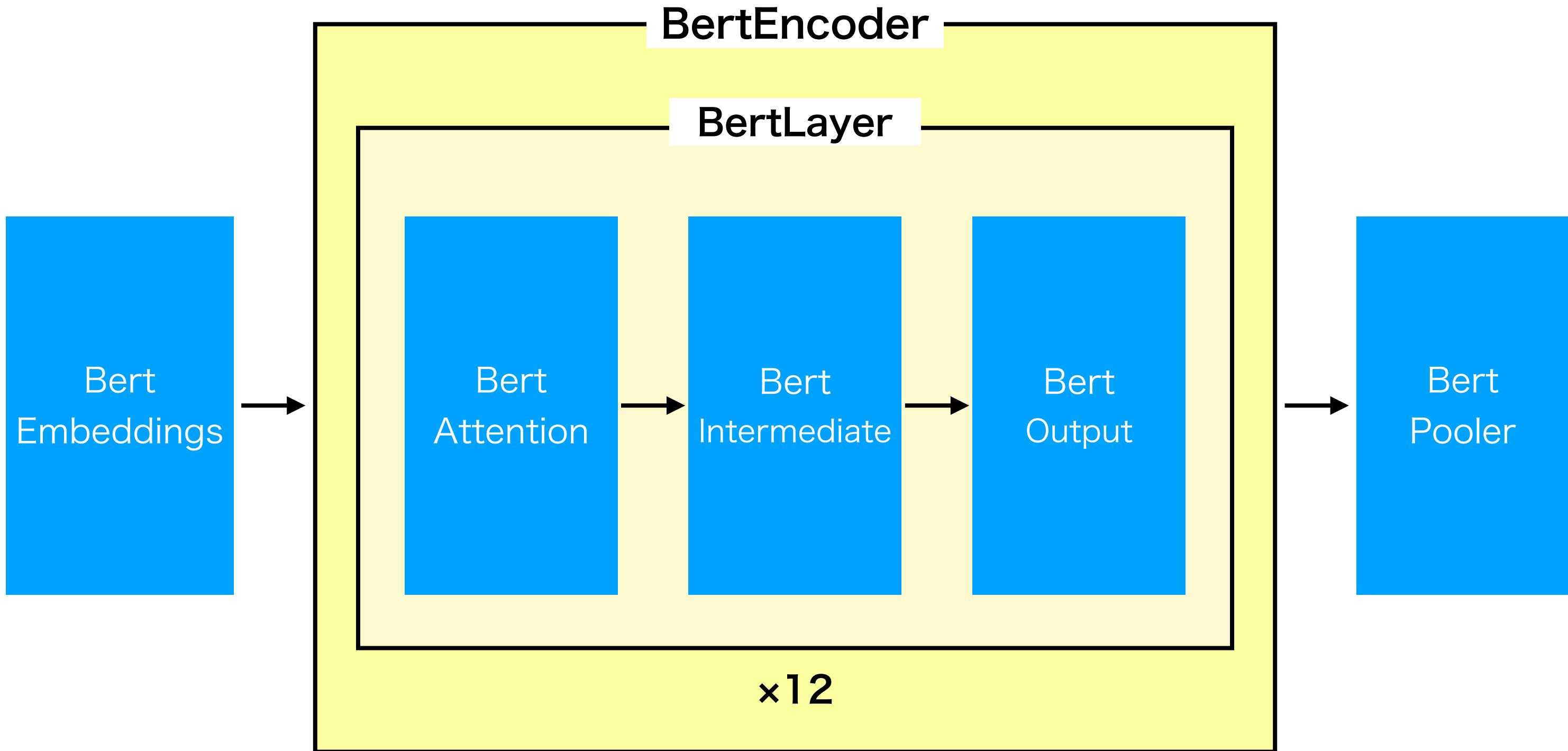
HELLO

BONJOUR

# BERTの実装（PyTorch-Transformers）を確認

•01\_bert\_model.ipynb

# BERTの構成 (PyTorch-Transformers)



# BertEmbeddings

- **BertEmbeddings**
  - 入力を埋め込みベクトルに変換
  - word\_embeddings、position\_embeddings、token\_type\_embeddingsの3つの埋め込みベクトルを足し合わせる
  - <https://git.io/Jknik>

# BertAttention

- **BertAttention**
  - BertSelfAttention、BertSelfOutputによりSelf-Attentionを実装
  - Maskの実装
  - <https://git.io/JknTg> (BertAttention)
  - <https://git.io/JknTi> (BertSelfAttention)
  - <https://git.io/JknXI> (BertSelfOutput)

# BertIntermediate、BertOutput

- **BertIntermediate、BertOutput**
  - Positionwise fully connected feed-forward networkを実装
  - <https://git.io/Jkn1u> (BertIntermediate)
  - <https://git.io/Jkn1r> (BertOutput)

# BertPooler

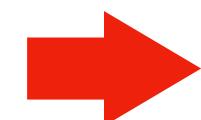
- **BertPooler**
  - 最初のトークン（単語）を取得し全結合層、活性化関数で処理
  - Next Sentence Predictionなどで使用
  - <https://git.io/JknUw>

# 次回の内容

Section 1. 講座とBERTの概要

Section 2. シンプルなBERTの実装

Section 3. BERTの仕組み



Section 4. ファインチューニングの活用

Section 5. BERTの応用