データサイエンス『実践コース』

数理工学PBL

Day 3：アテンション

一関高専　小池　敦

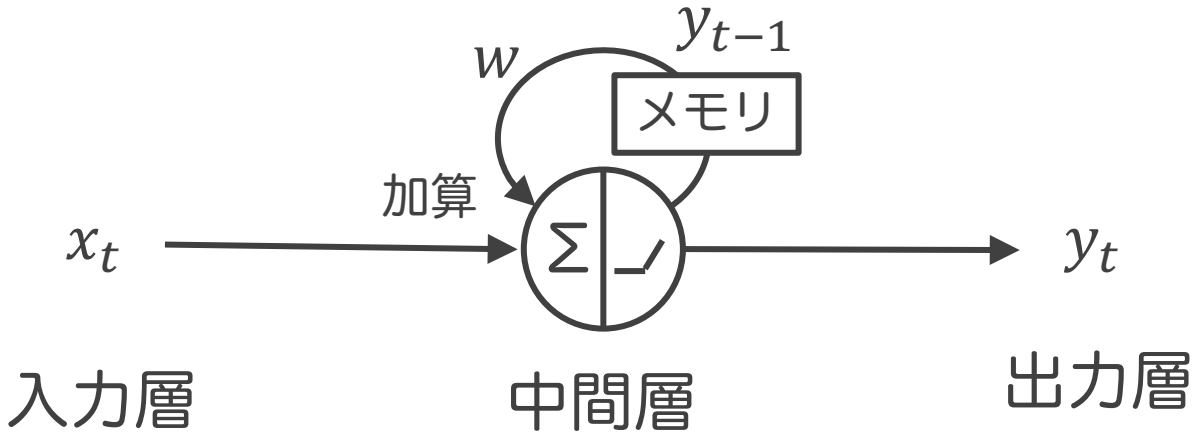
今日の内容

- 目標
  - 文の内容を把握するような深層学習モデルを作る
- 内容
  - 再帰型ニューラルネットワーク（RNN）
  - エンコーダー・デコーダーモデル
  - アテンション（Transformer）
  - アテンションの活用（BERT）

再帰型ニューラルネットワーク

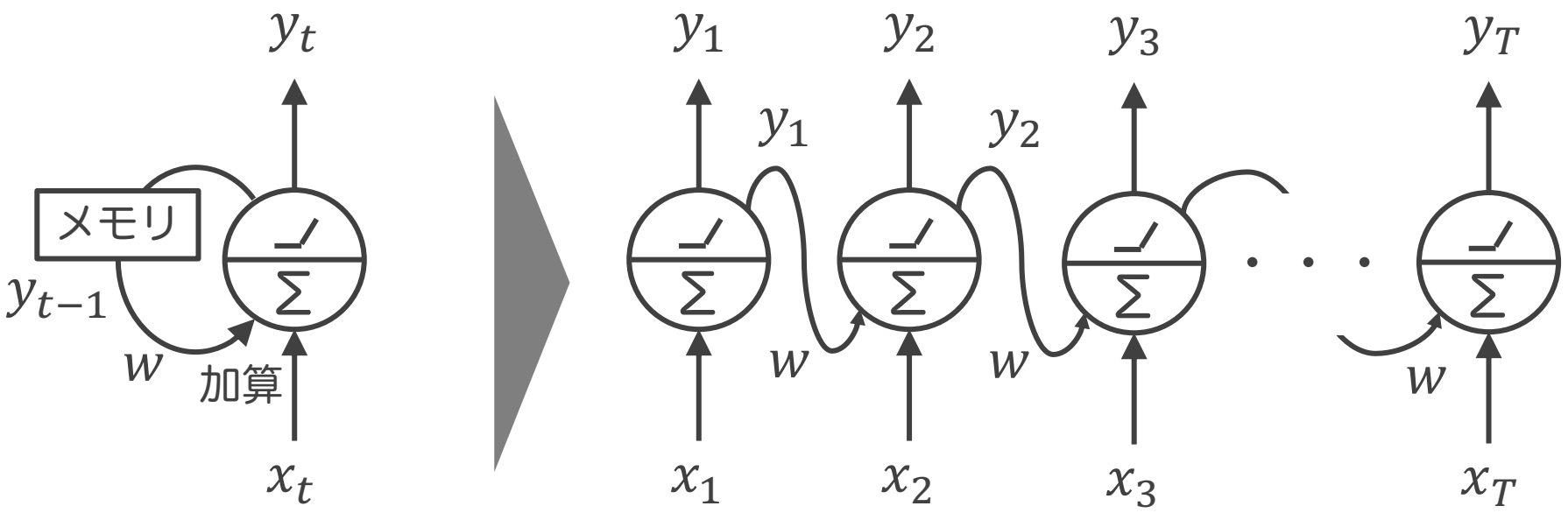
再帰型ニューラルネットワーク（RNN）

- 内部に循環構造を持つニューラルネットワーク
- タイムステップごとにひとつの入力
  - ユニットはメモリを持ち，前の時刻の値を出力
- 可変長の時系列データ（や自然言語）を扱える

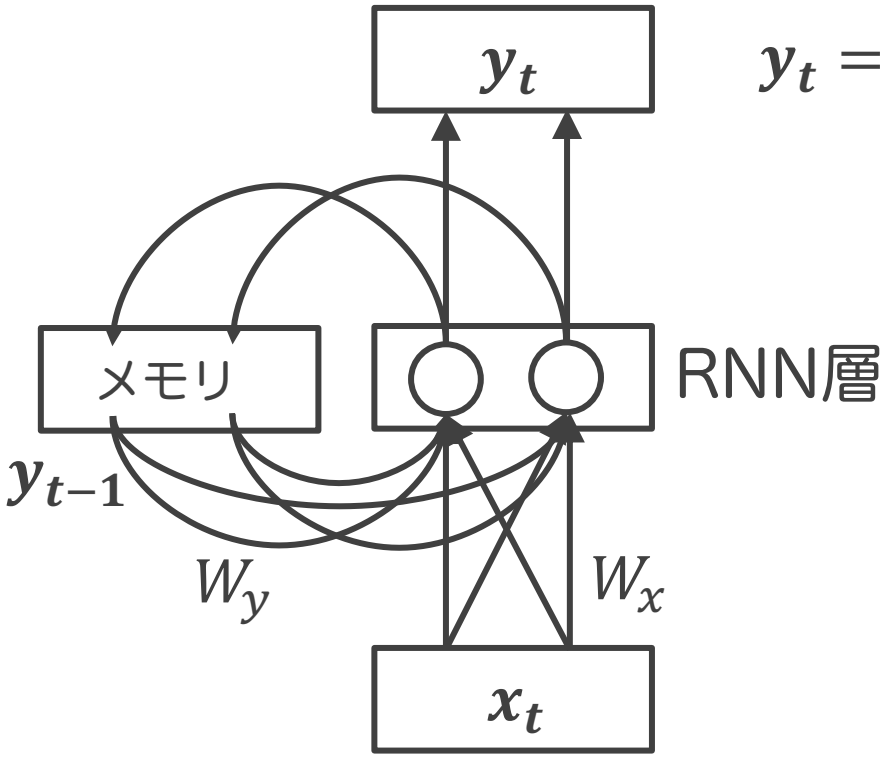


RNNのアンロール

- RNNを巡回構造を使わず表現

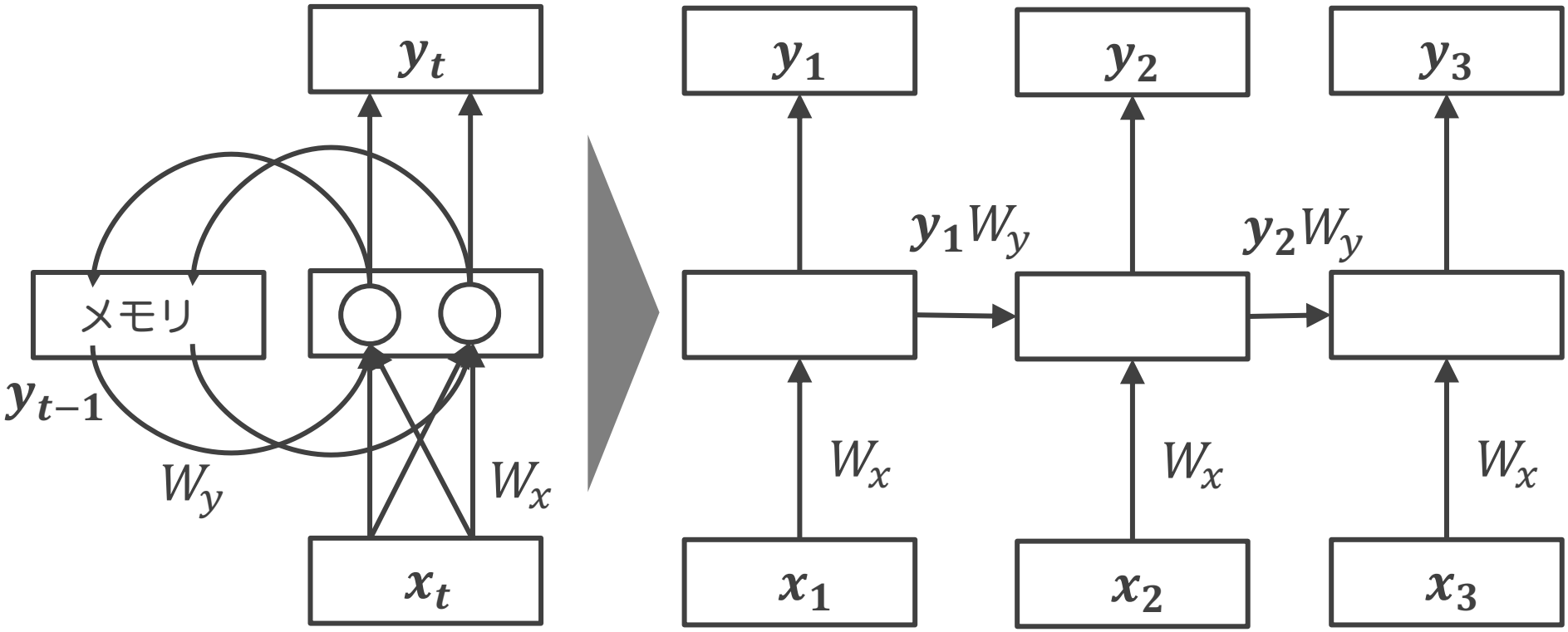


RNNの行列表現



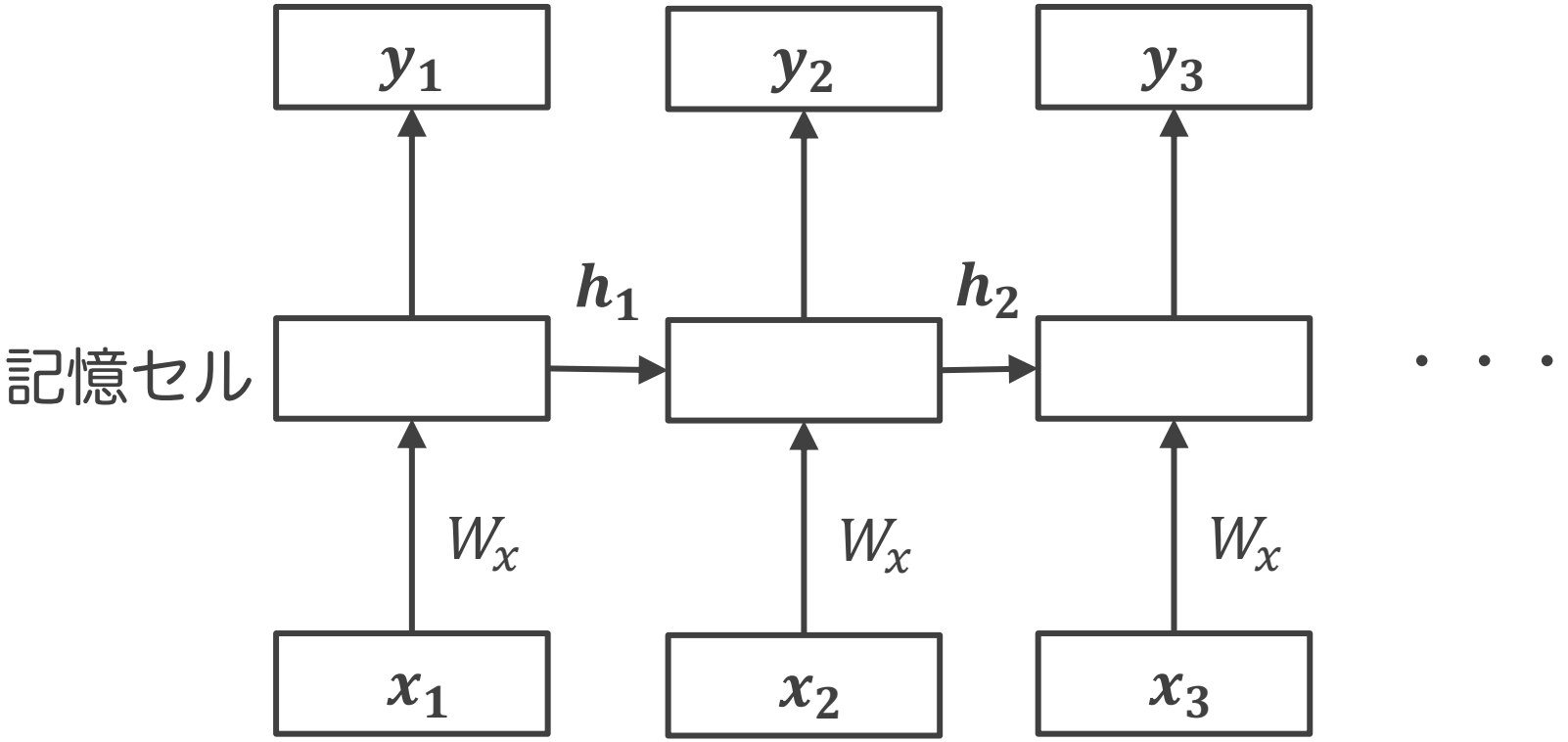
$$y_t = y_{t-1}W_y + x_tW_x$$

RNN層のアンロール



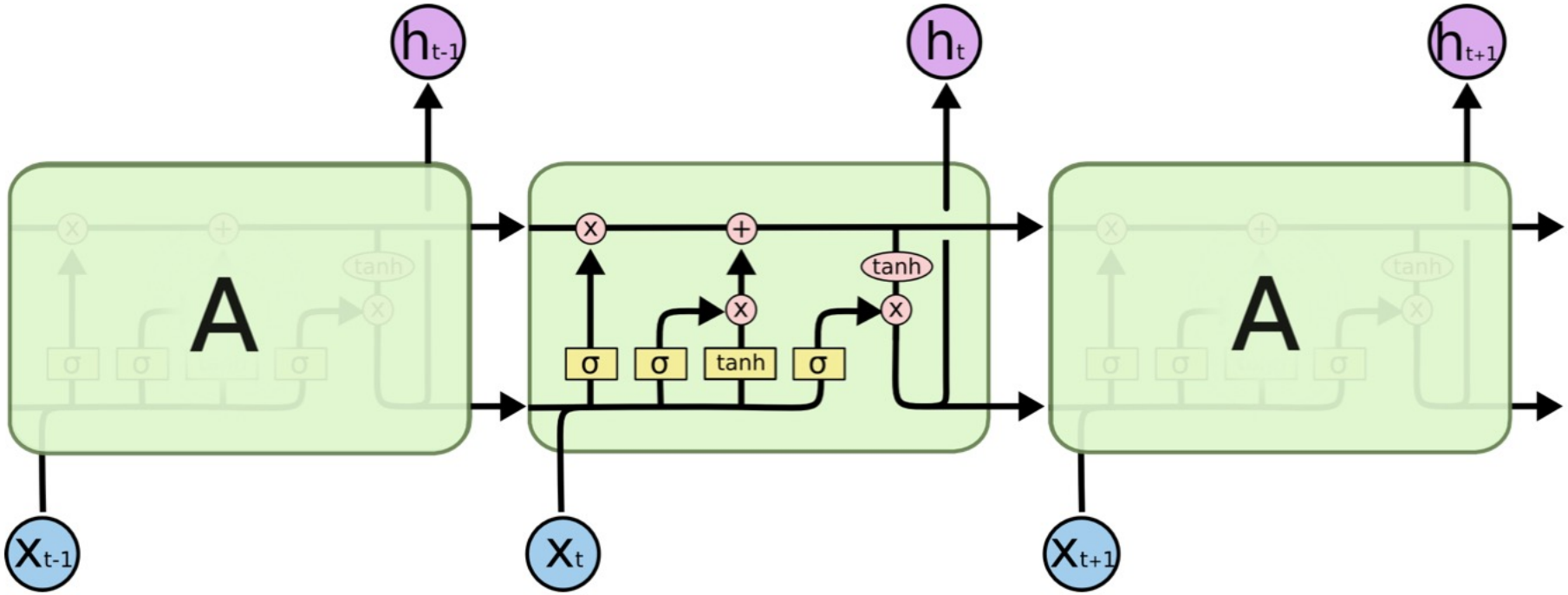
記憶セル

- $\boldsymbol{y_{t-1}W_t}$を一般化して$\boldsymbol{h_{t-1}}$を再帰させる



LSTM

- 長期記憶に優れた記憶セル（詳細省略）

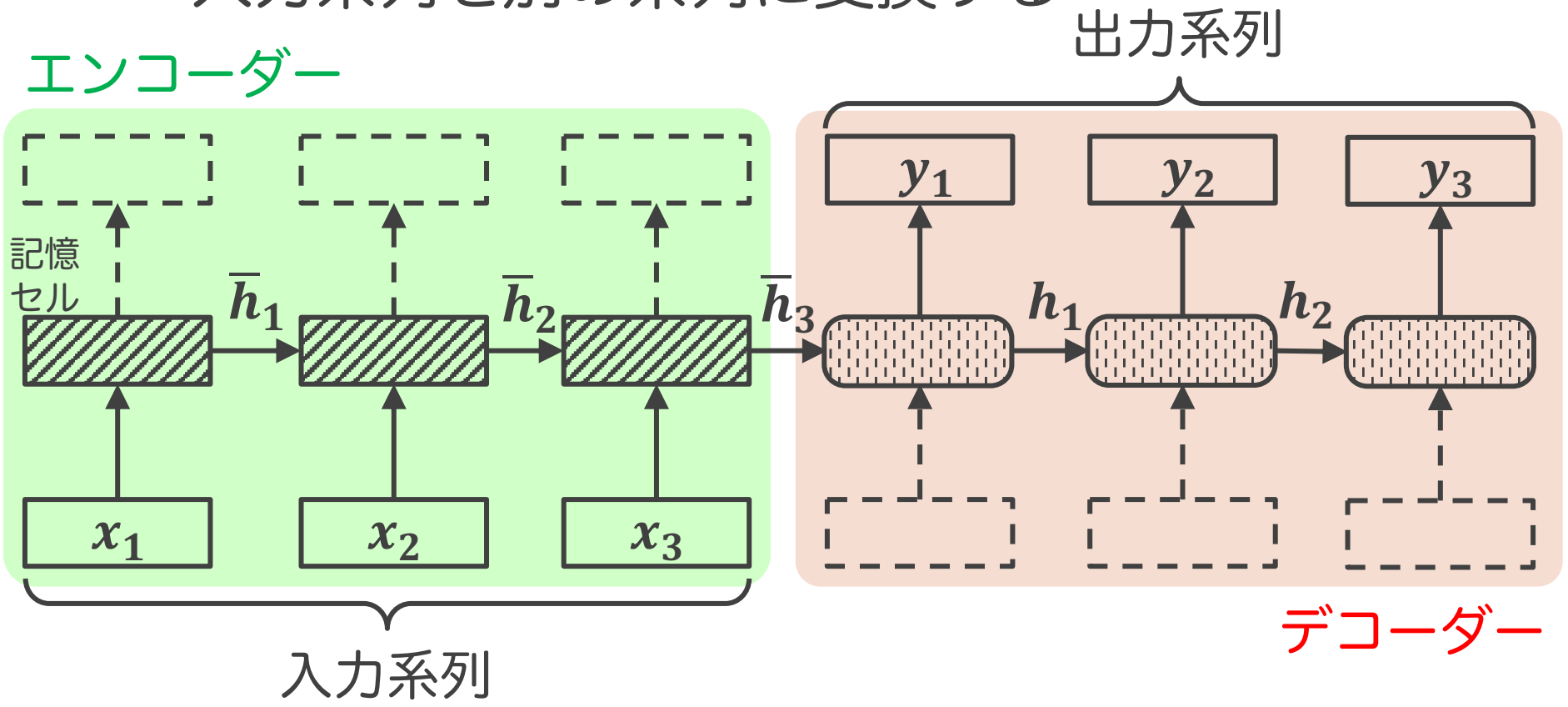


Colah's blog　Understanding LSTM Networks
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

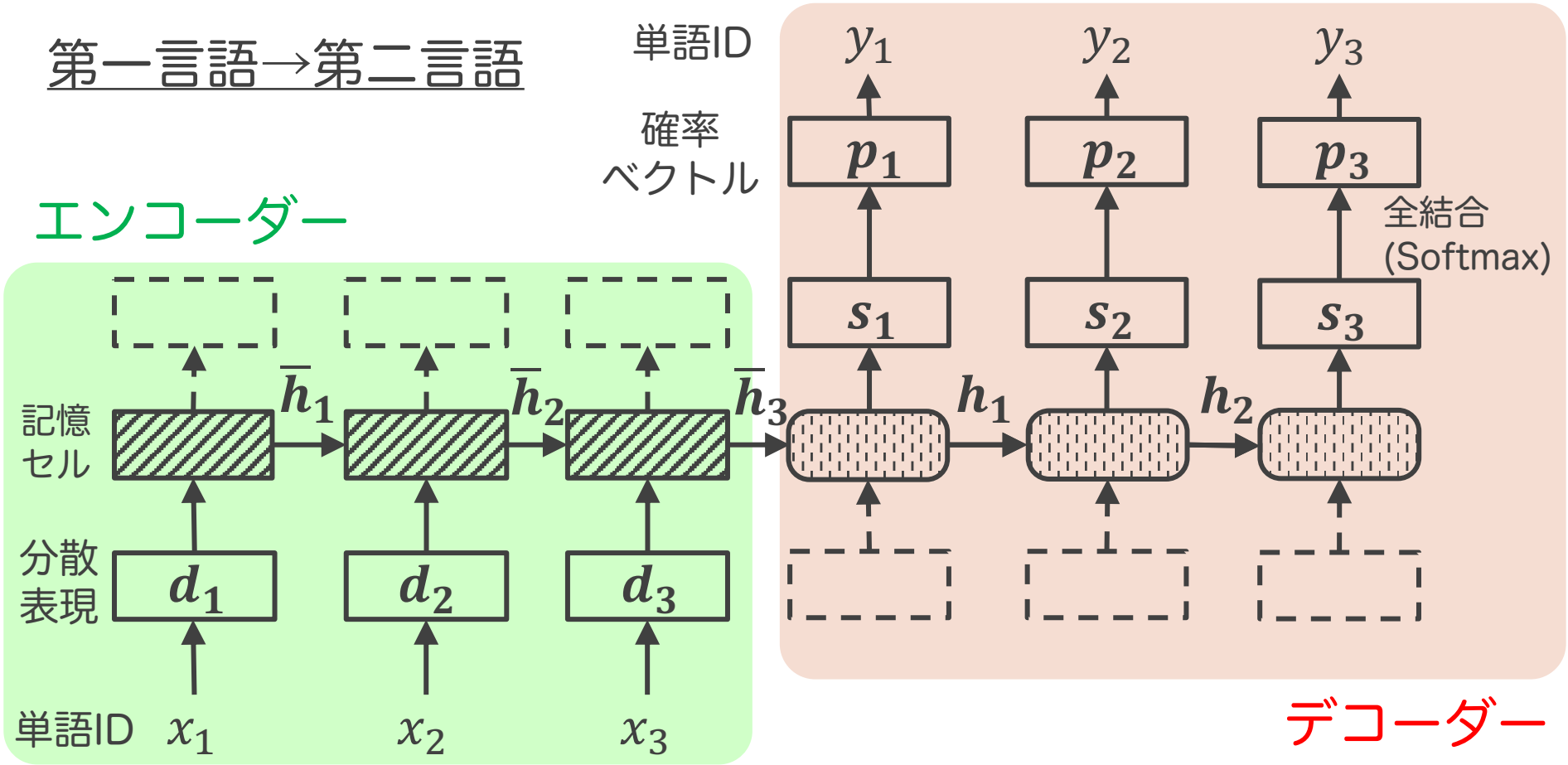
エンコーダー・デコーダーモデル

エンコーダー・デコーダーモデル

- 入力系列を別の系列に変換する

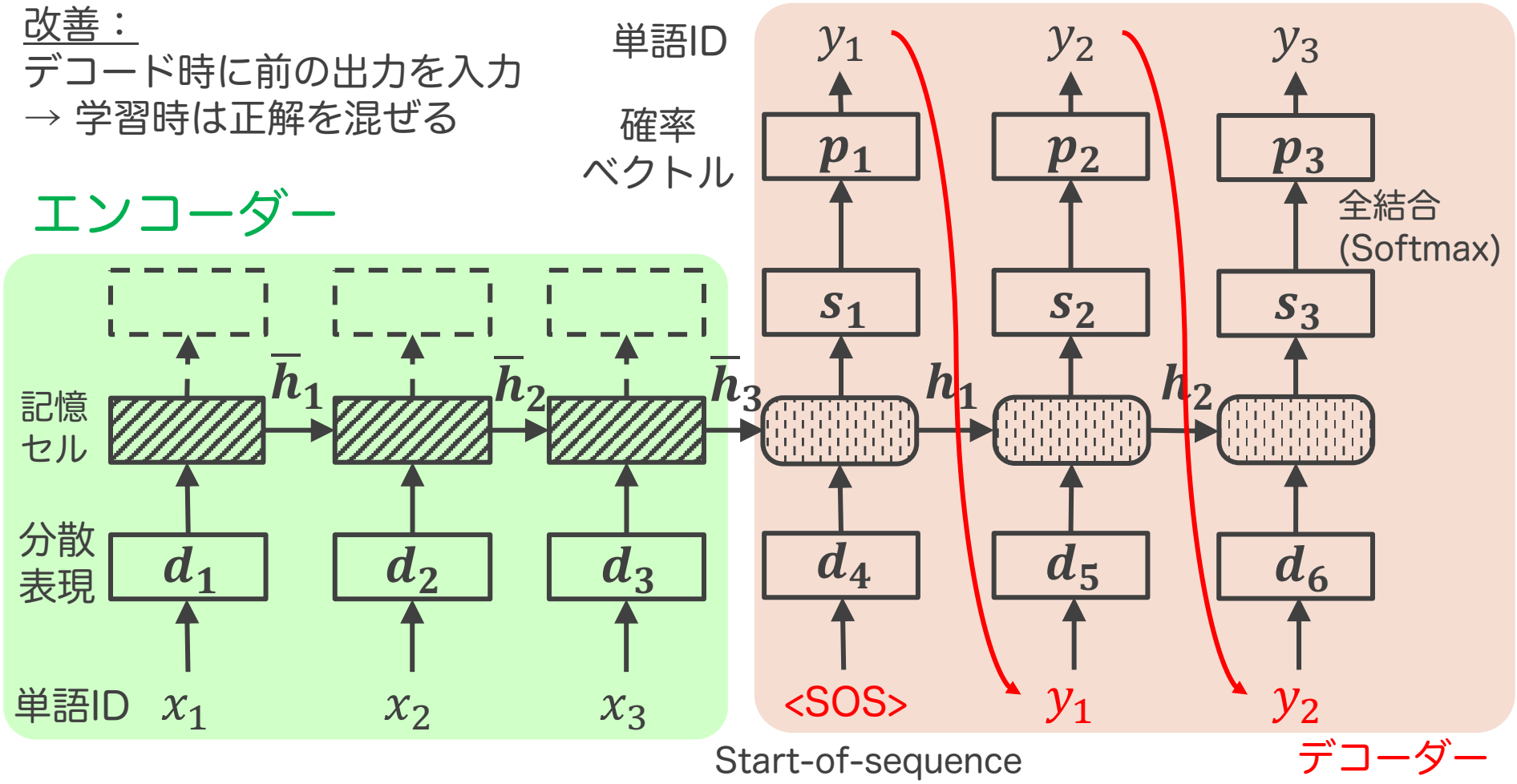


エンコーダー・デコーダーモデルによる機械翻訳



エンコーダー・デコーダーモデルによる機械翻訳

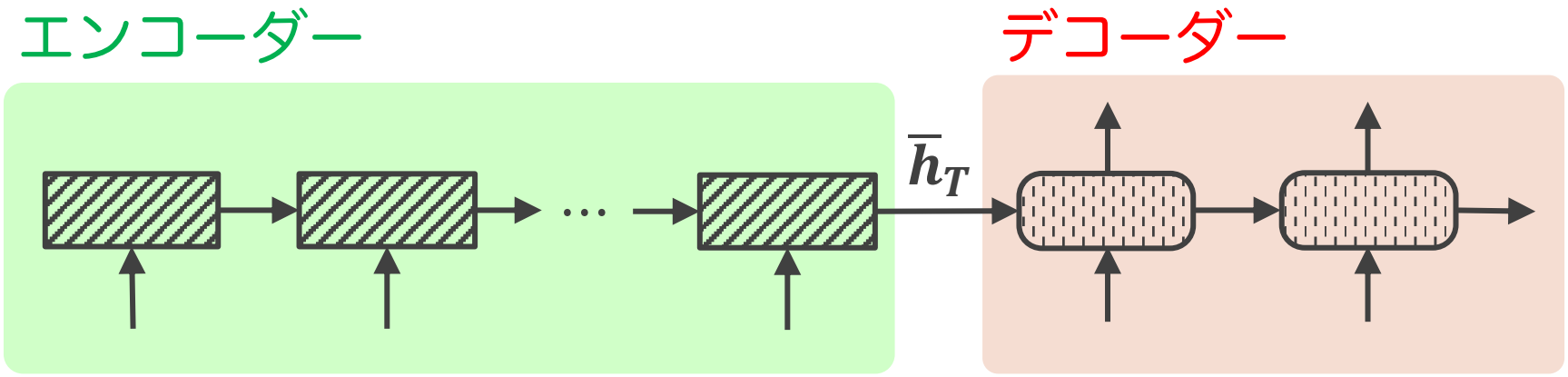
改善：
デコード時に前の出力を入力
→ 学習時は正解を混ぜる



アテンション

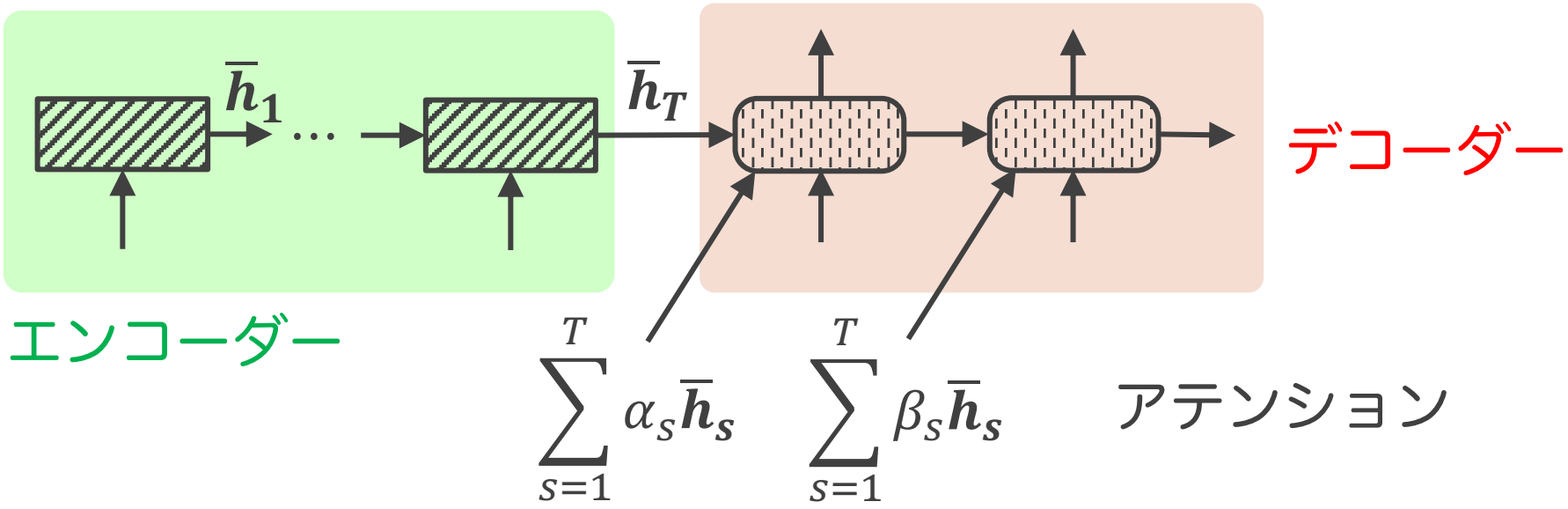
背景

- エンコーダー・デコーダーモデルでは，
エンコーダーの最終状態 $\overline{\boldsymbol{h}}_{\boldsymbol{T}}$のみをデコーダーに渡す
→ 学習により，$\overline{\boldsymbol{h}}_{\boldsymbol{T}}$は元の文の意味全体を含むようになる
→ しかし文が長くなると，$\overline{\boldsymbol{h}}_{\boldsymbol{T}}$にすべての文意を
詰め込むことは困難になり，翻訳性能が下がる



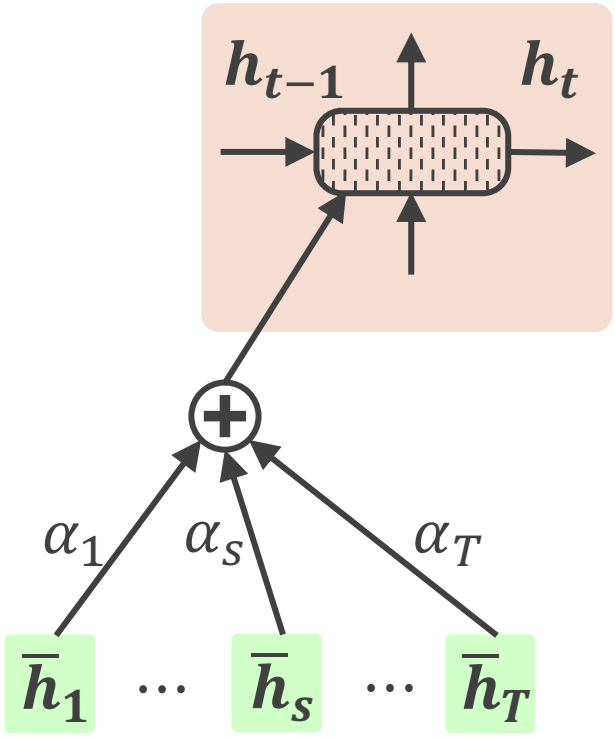
アテンションとは？

- デコーダーの各時刻に
エンコーダーの中間状態 $\bar{\boldsymbol{h}}_{\boldsymbol{s}}$ $(s = 1, \cdots, T)$のアフィン結合を渡す
→ 文が長くなっても翻訳性能が落ちない



アフィン結合の係数

- 前時刻の状態 $\boldsymbol{h_{t-1}}$と $\bar{\boldsymbol{h}}_{\boldsymbol{s}}$ が類似　→ 係数大
  似てない　→ 係数小



$\boldsymbol{h_{t-1}}$と $\bar{\boldsymbol{h}}_{\boldsymbol{1}}, \cdots, \bar{\boldsymbol{h}}_{\boldsymbol{s}}, \cdots, \bar{\boldsymbol{h}}_{\boldsymbol{T}}$の類似度スコア：

$$[e_1, \cdots, e_s, \cdots, e_T]$$

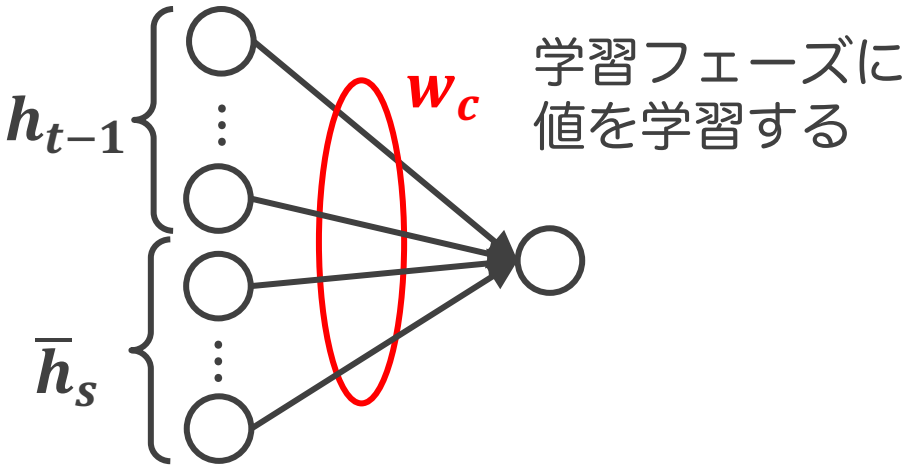
⇓　アフィン結合化

$$[\alpha_1, \cdots, \alpha_s, \cdots, \alpha_T] = \mathrm{Softmax}[e_1, \cdots, e_s, \cdots, e_T]$$

類似度スコア

- $\boldsymbol{h_{t-1}}$, $\overline{\boldsymbol{h}}_{\boldsymbol{s}}$ は横ベクトルとする※
- 連結注意：$\left[\boldsymbol{h_{t-1}}\ \overline{\boldsymbol{h}}_{\boldsymbol{s}}\right]\boldsymbol{w_c}$

$$\left[\ \boldsymbol{h_{t-1}}\quad \overline{\boldsymbol{h}}_{\boldsymbol{s}}\ \right]\begin{bmatrix}\boldsymbol{w_c}\end{bmatrix}$$



※ 下記論文 [Luong+ 2015] では縦ベクトルだが，Transformer での表記に合わせ横ベクトルにする
https://aclanthology.org/D15-1166/

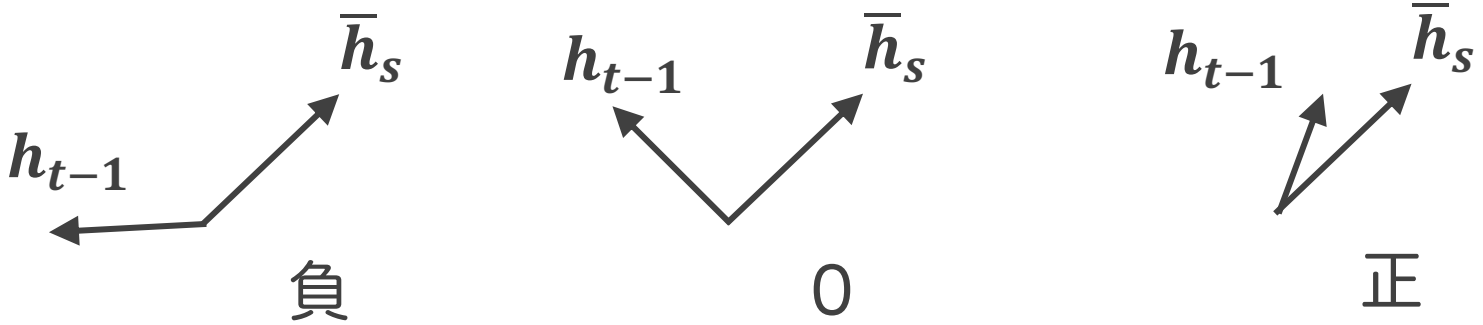
類似度スコア

- 内積注意 [Luong 2015]： $\boldsymbol{h_{t-1}\bar{h}_s^{\,T}}$

$$\left[\ \boldsymbol{h_{t-1}}\ \right]\left[\ \bar{\boldsymbol{h}}_{\boldsymbol{s}}^{\,\boldsymbol{T}}\ \right]$$

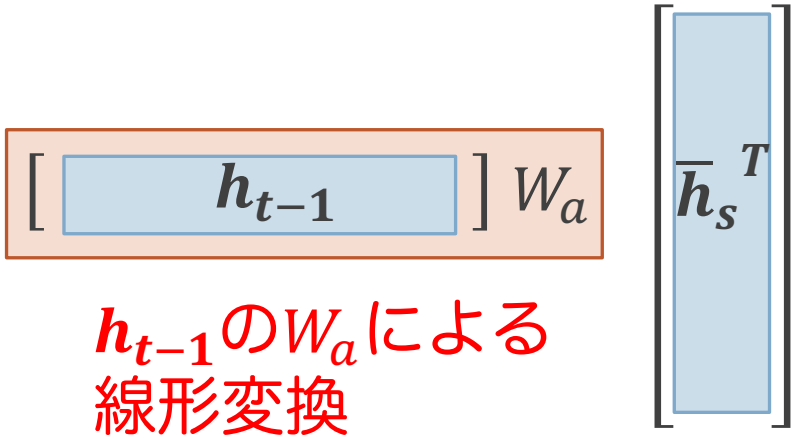
つまり$\boldsymbol{h_{t-1}}$と $\bar{\boldsymbol{h}}_{\boldsymbol{s}}$ の内積

内積の値



類似度スコア

- 一般注意（Transformer等）： $\boldsymbol{h_{t-1}} W_a \overline{\boldsymbol{h}}_s^{\,T}$



$\boldsymbol{h_{t-1}}$のW_aによる
線形変換

つまり $\boldsymbol{h_{t-1}} W_a$と $\overline{\boldsymbol{h}}_s$ の内積

W_aは学習フェーズに学習する

直感的なイメージ：
$\boldsymbol{h_{t-1}}$をターゲット言語の世界からソース言語の世界に
変換（W_a）してから $\overline{\boldsymbol{h}}_s$ との内積をとる

類似度スコア

- Transformerで使用する一般注意
  - $\boldsymbol{h_{t-1}}$と$\bar{\boldsymbol{h}}_s$のそれぞれを異なる行列で線形変換してから内積を計算する

$\boldsymbol{h_{t-1}}$のW_qによる線形変換

$$\left[\ \boldsymbol{h_{t-1}}\ \right] W_q$$

$$W_k{}^T \left[\ \bar{\boldsymbol{h}}_s{}^T\ \right]$$

$\bar{\boldsymbol{h}}_s$のW_kによる線形変換の転置

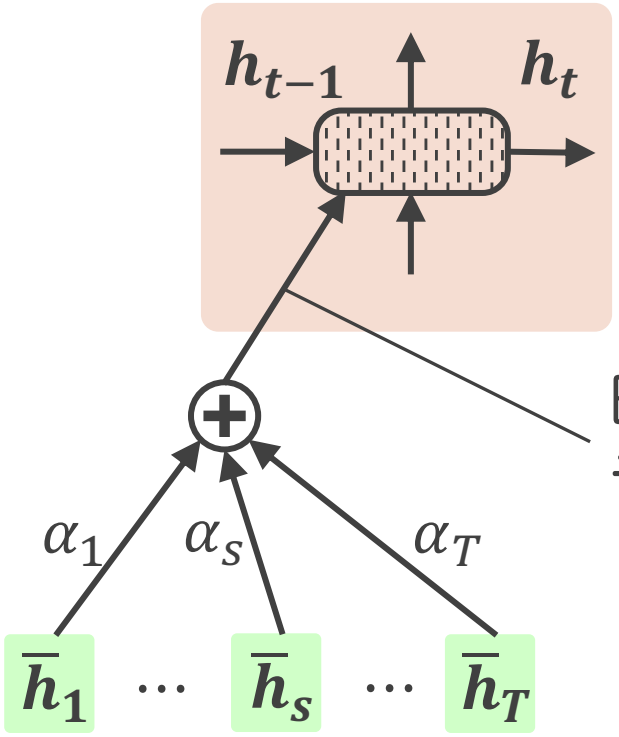
クエリ　　キー

$\boldsymbol{h_{t-1}} W_q$と $\bar{\boldsymbol{h}}_s W_k$の内積

$$= \left(\boldsymbol{h_{t-1}} W_q\right)\left(\bar{\boldsymbol{h}}_s W_k\right)^{\boldsymbol{T}}$$

$$= \boldsymbol{h_{t-1}} W_q W_k{}^T \bar{\boldsymbol{h}}_s{}^{\boldsymbol{T}}$$

解釈



時刻 t で翻訳すべき単語の情報を
デコーダーに伝えたい

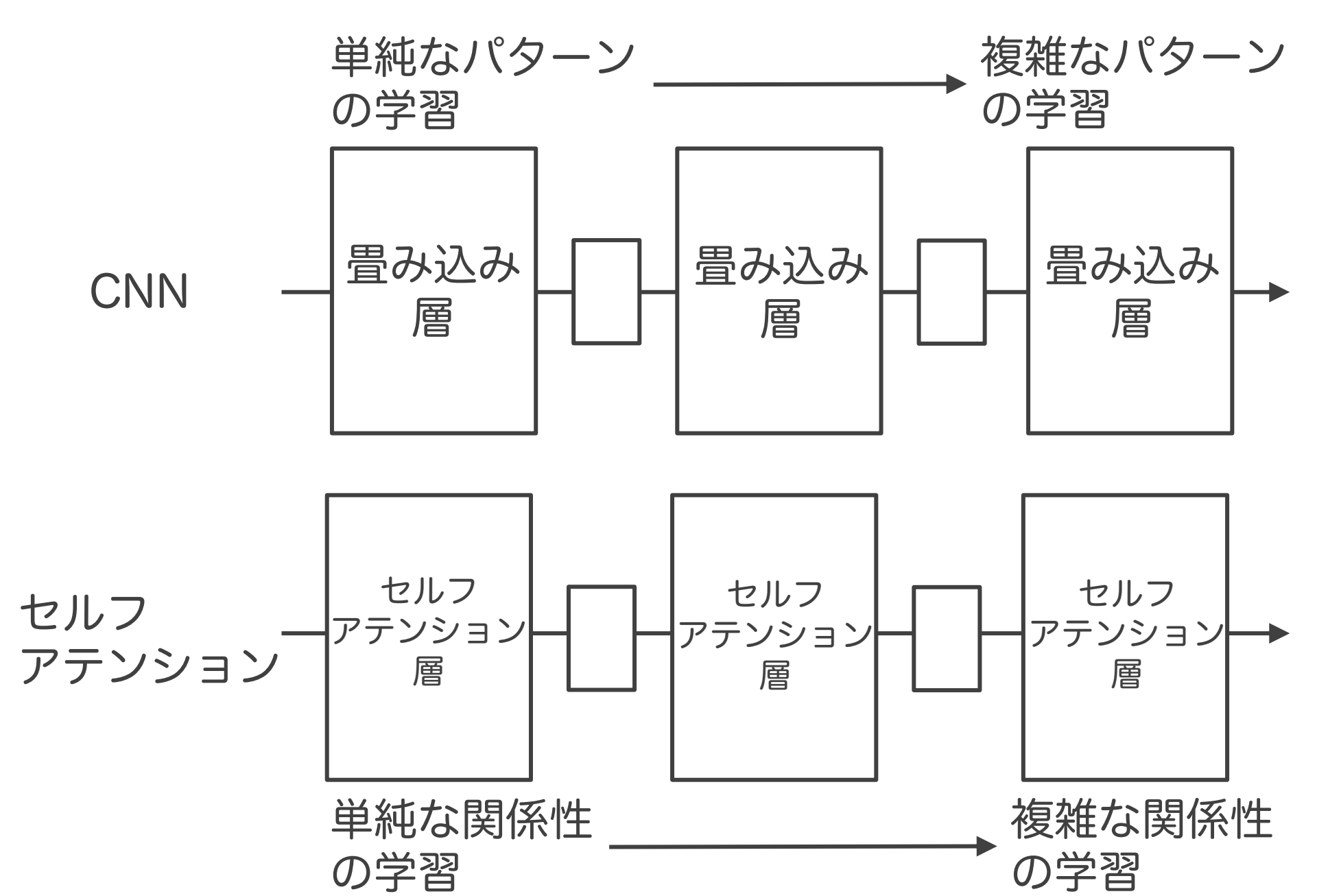
⇒ もはや h_{t-1}は不要？

⇒ Transformer
（「Attention is all you need」）

セルフアテンション

セルフアテンション

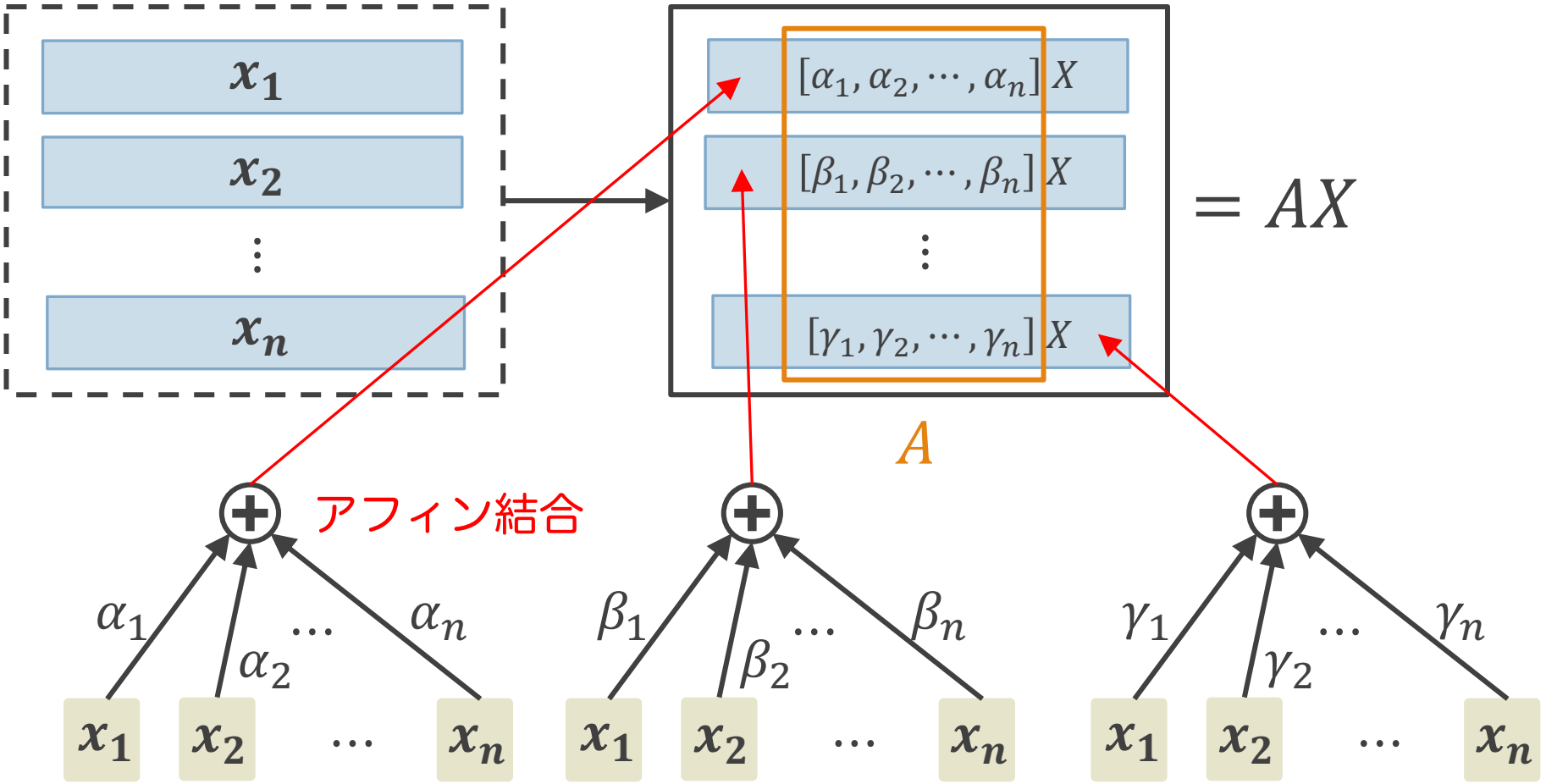
- 言語間の単語の類似性でなく，
同一言語内で，文の単語間の関係性を学習する
  - 修飾語と被修飾語
  - 主語と述語
  - 「それ」とそれが指し示す名詞
- セルフアテンション層を繰り返すことで
より複雑な関係性を学習できる


単純なパターン
の学習
複雑なパターン
の学習
CNN
畳み込み
層
畳み込み
層
畳み込み
層
セルフ
アテンション
セルフ
アテンション
層
セルフ
アテンション
層
セルフ
アテンション
層
単純な関係性
の学習
複雑な関係性
の学習

セルフアテンション層

入力：単語列 X（分散表現）

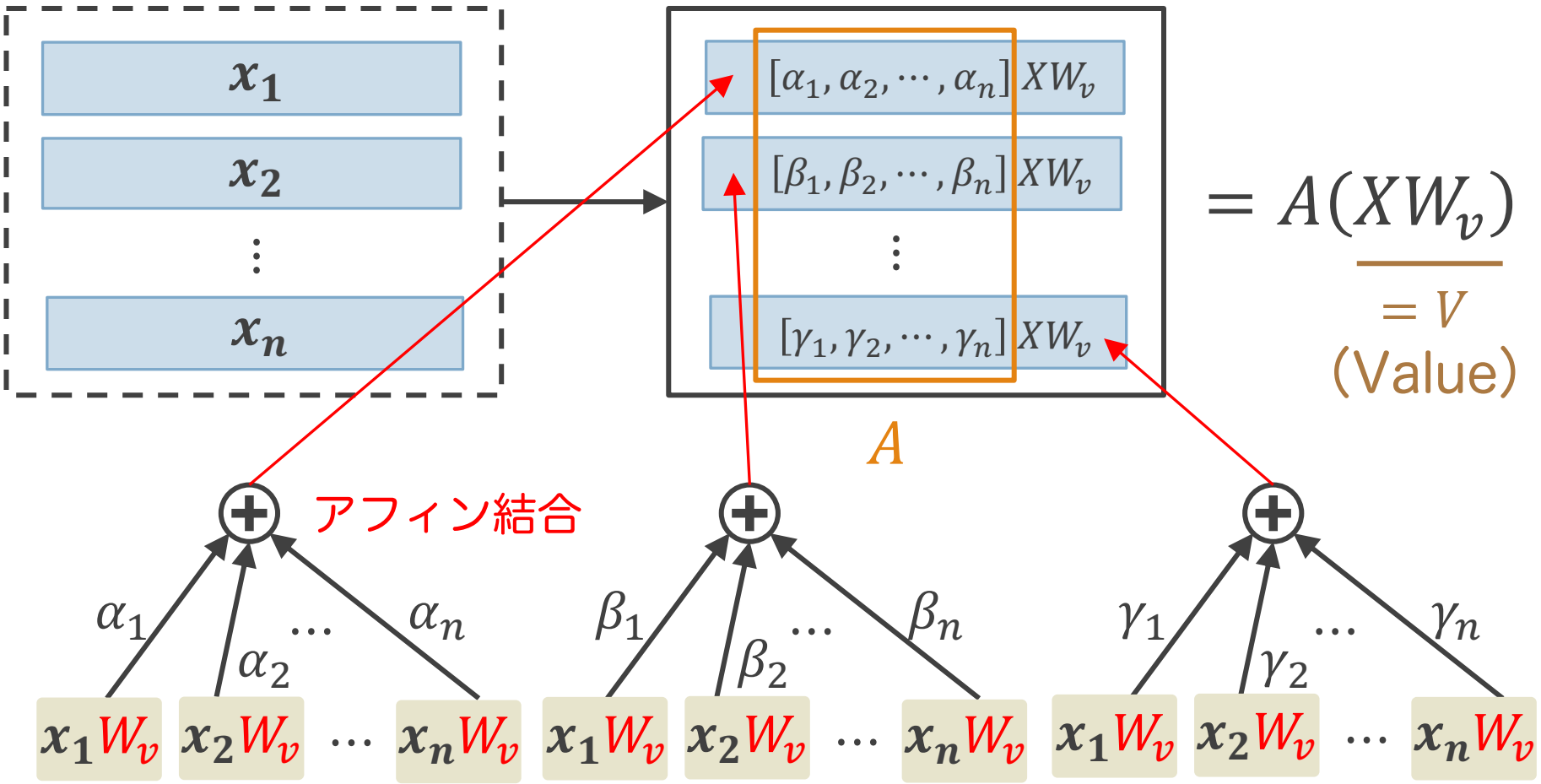
各単語 x_sに対するアテンション



Transformerのセルフアテンション層

入力：単語列 X（分散表現）

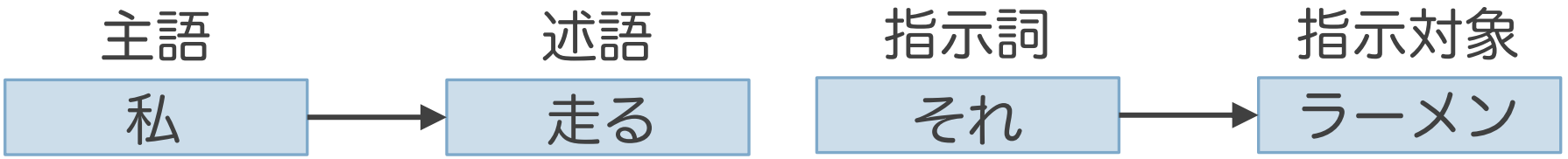
各単語 x_sに対するアテンション



各単語を線形変換したのち，アフィン結合を求める

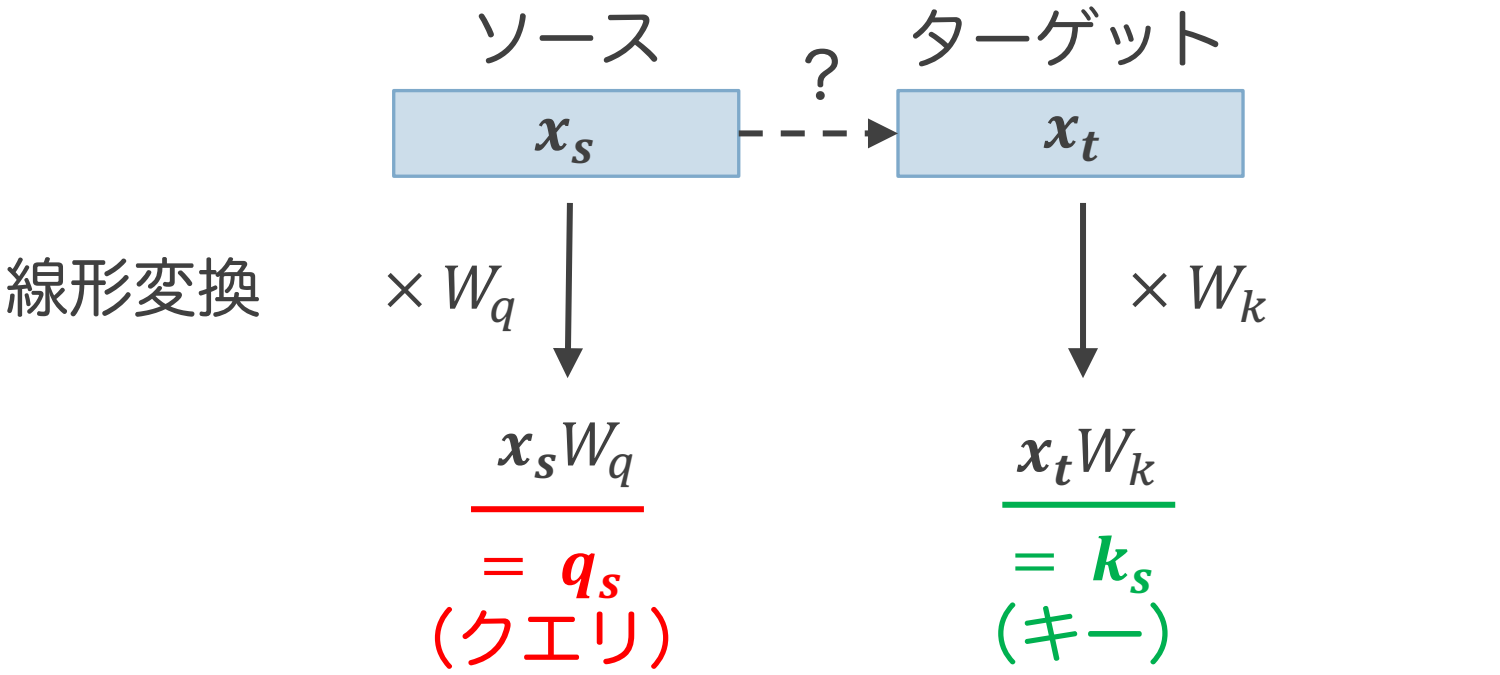
セルフアテンション層における アフィン結合の係数

- 単に単語どうし似ているというだけでなく，単語間の関係を抽出したい



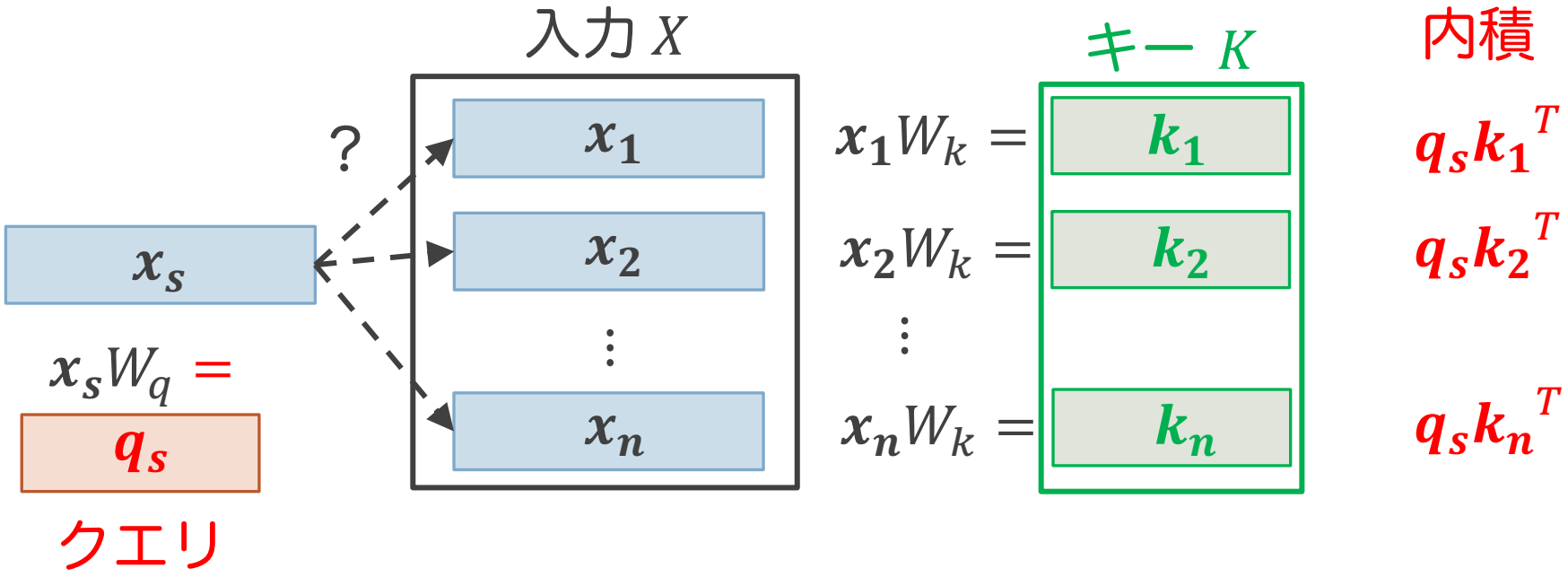
セルフアテンション層における アフィン結合の係数

- 「関係」の抽出



⇒ クエリとキーの内積をとる（$q_s k_s{}^T$）

セルフアテンション層における アフィン結合の係数



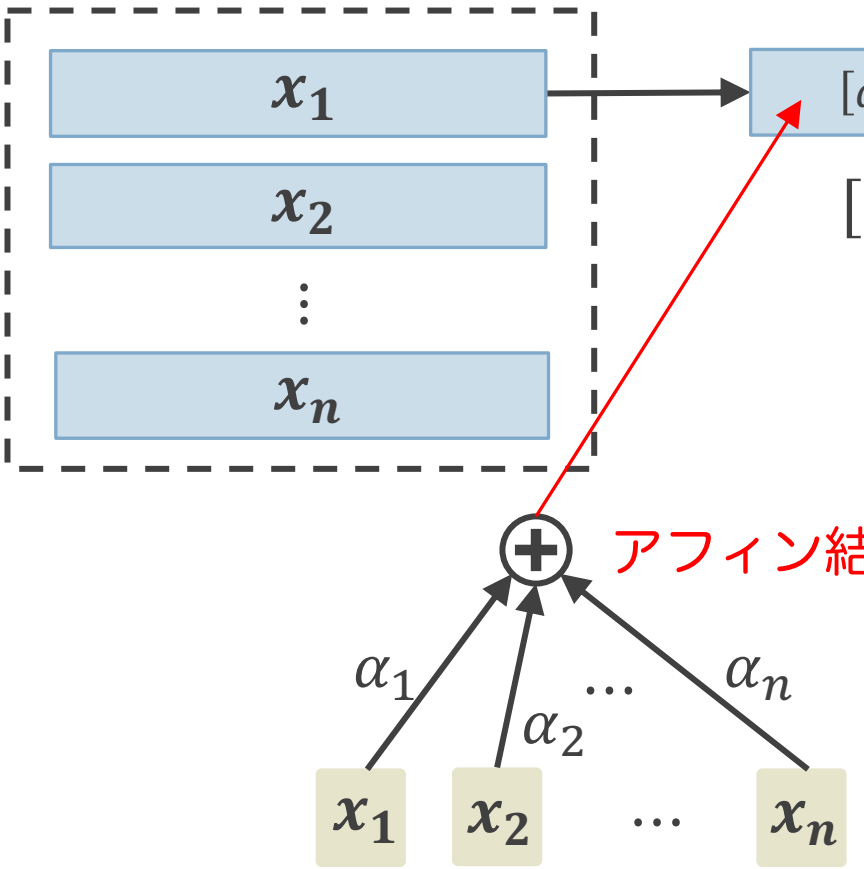
$$\left[q_s k_1^T, q_s k_2^T, \quad \cdots, q_s k_n^T\right] = q_s K^T$$

⇒ アフィン結合になるように変換

係数 $\mathrm{Softmax}\left[q_s k_1^T, q_s k_2^T, \quad \cdots, q_s k_n^T\right] = \mathrm{Softmax}(q_s K^T)$

1クエリに対する セルフアテンションの導出

入力：単語列 X（分散表現） 単語 $\boldsymbol{x_1}$に対するアテンション



$$[\alpha_1, \alpha_2, \cdots, \alpha_n] = \mathrm{Softmax}(\boldsymbol{q_1}\, K^T)$$

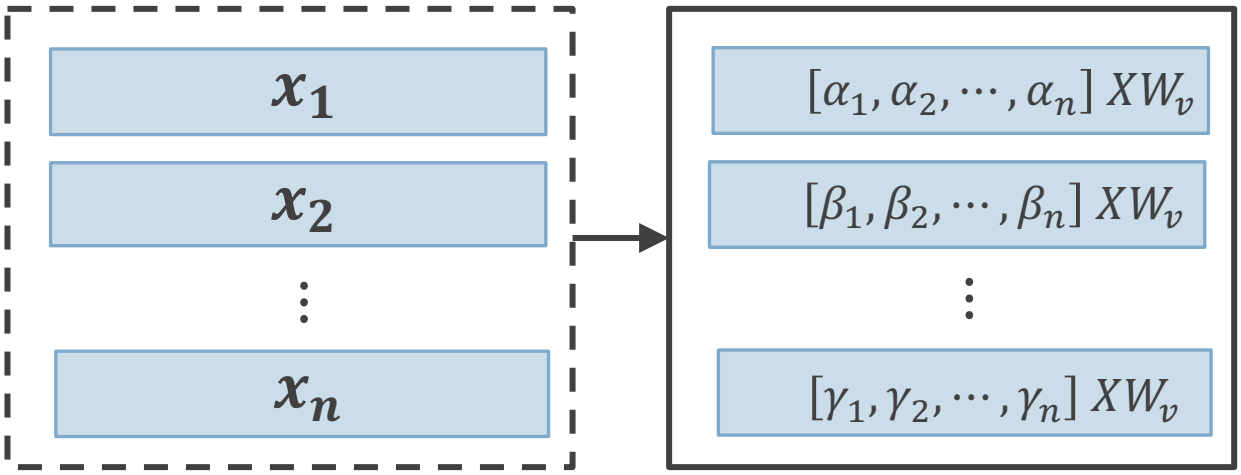
$$\boldsymbol{q_1} = \boldsymbol{x_1} W_q, K = XW_q$$

$$\therefore\ [\alpha_1, \alpha_2, \cdots, \alpha_n]V = \mathrm{Softmax}(\boldsymbol{q_1}\, K^T)V$$

$$(V = XW_v)$$

セルフアテンション層まとめ

入力：単語列 X（分散表現） 各単語 $\boldsymbol{x_s}$に対するアテンション



$= \mathrm{Softmax}(\boldsymbol{q_1}\, K^T)V$

$= \mathrm{Softmax}(\boldsymbol{q_2}\, K^T)V$

$= \mathrm{Softmax}(\boldsymbol{q_3}\, K^T)V$

セルフアテンションの出力： $\mathrm{Softmax}(Q\, K^T)V$

$(Q = XW_q, K = XW_k, V = XW_v)$

Softmax は行ごとに計算する

Transformerで使用されるセルフアテンション層

- アフィン結合において，一部のベクトルのみの成分が大きくなり過ぎないように
Softmax 適用前に全体的に値を小さくする
⇒ 各係数の値を $\sqrt{d_k}$ で割る
（ d_k はキーベクトルの次元数）

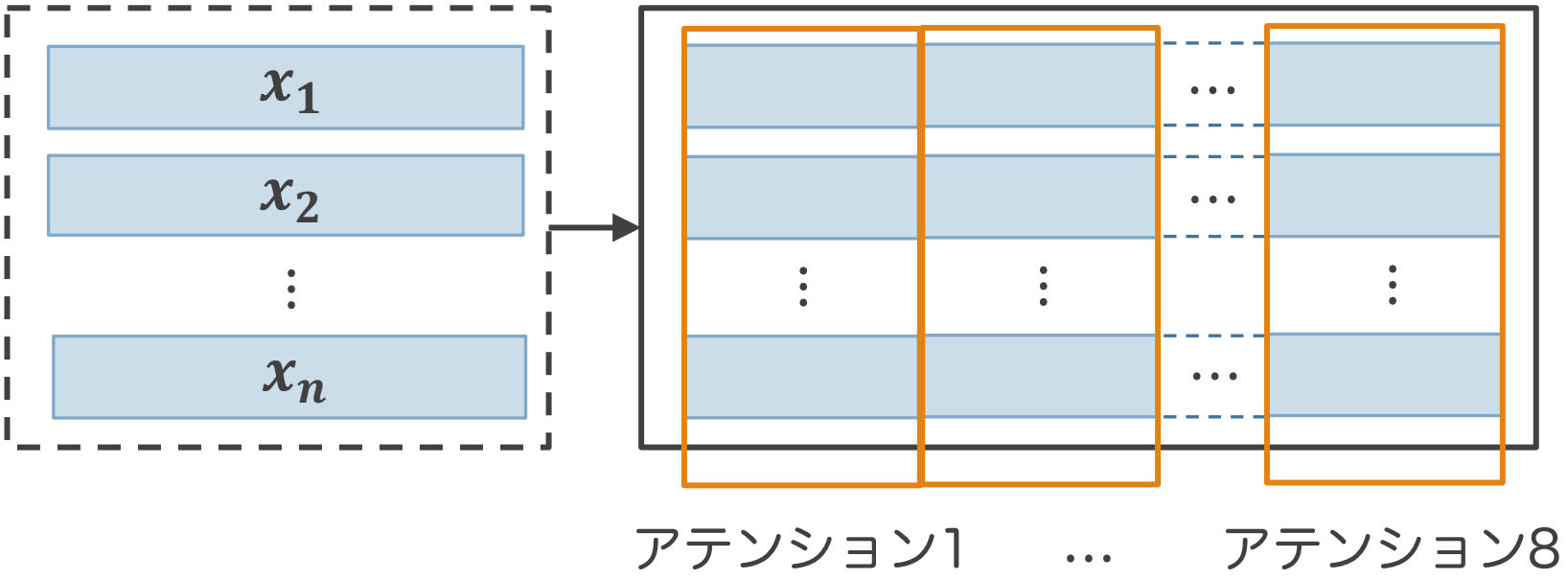
セルフアテンションの出力： $\mathrm{Softmax}\left(\frac{Q\,K^T}{\sqrt{d_k}}\right)V$

$(Q = XW_q, K = XW_k, V = XW_v)$

マルチヘッドアテンション

- 異なる行列を用いて複数回アテンションを計算する
  ⇒ 複数の異なる関係性を抽出できる

入力：単語列 X（分散表現） 各単語 x_sに対するアテンション



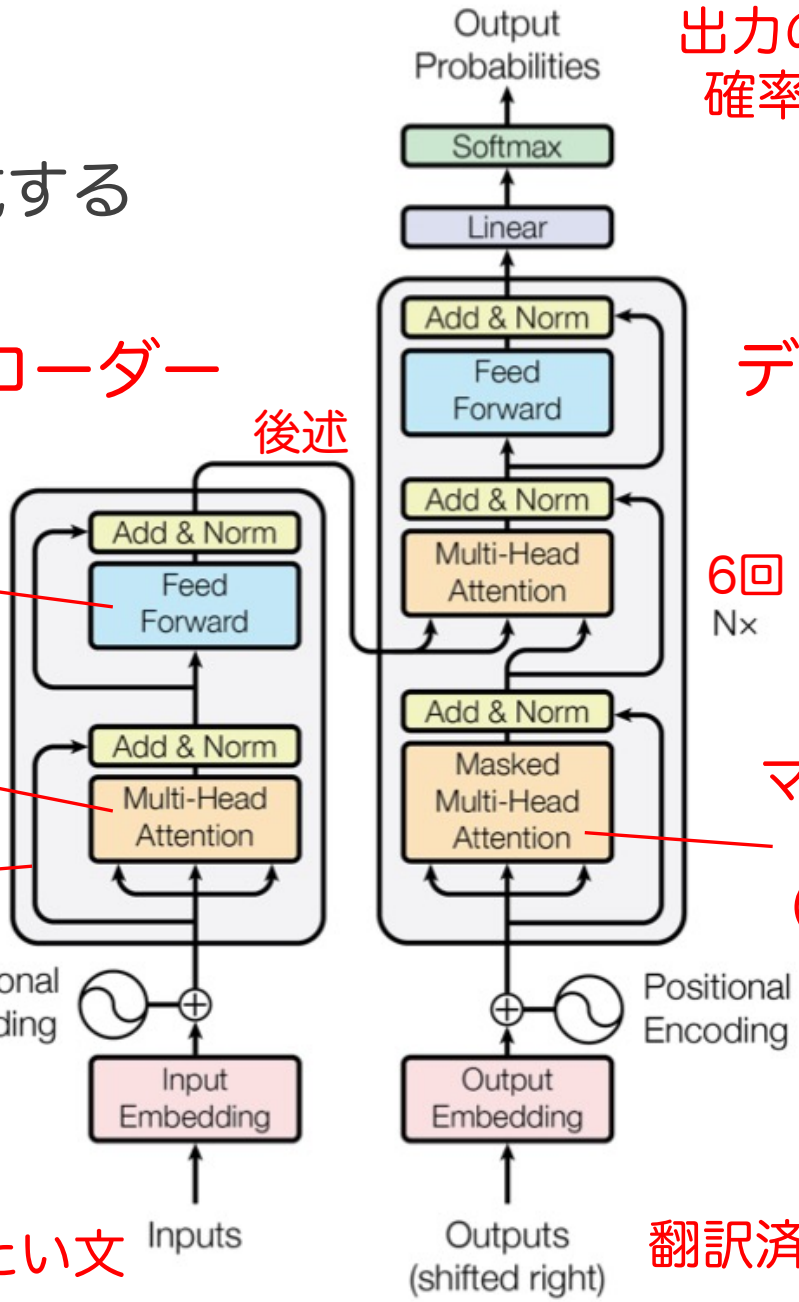
Transformer

Transformer概要

- 高性能な機械翻訳モデル
- 再帰的な層を使用せず，アテンションのみでモデルを構築する
- 入力として文（単語列）を丸ごと入れる
  - 再帰的な層では各時刻に付き1単語しか入力できず並列化がしにくかったが，並列化が容易になる
- エンコーダー・デコーダーともにセルフアテンションを6回積み上げる
  - エンコーダーとデコーダーはアテンションで接続

全体構成

出力の１単語を生成する



出力の１単語の確率ベクトル

エンコーダー

デコーダー

後述

全結合層

マルチヘッドセルフアテンション層

スキップ接続（＋レイヤ正規化）

位置エンコーディング（後述）

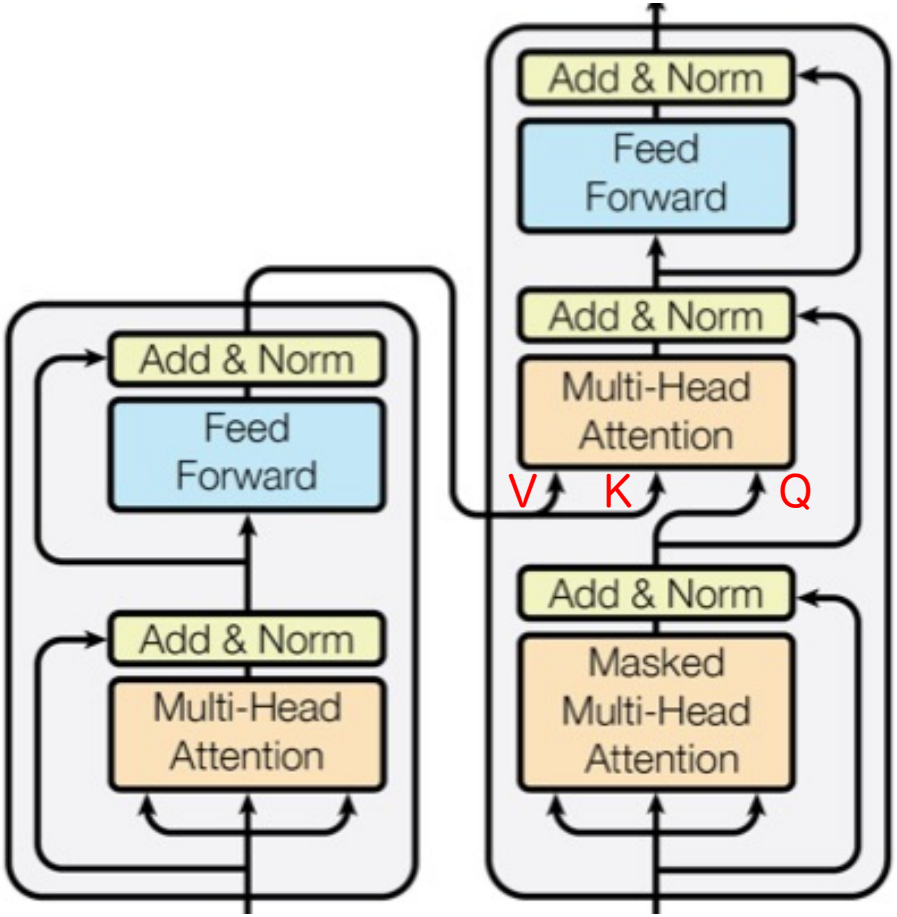
マスク付きマルチヘッドセルフアテンション層（学習時未出力部をマスク）

翻訳したい文

翻訳済みの出力

Vaswani+. Attention is all you need. Neurips 2017.
https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

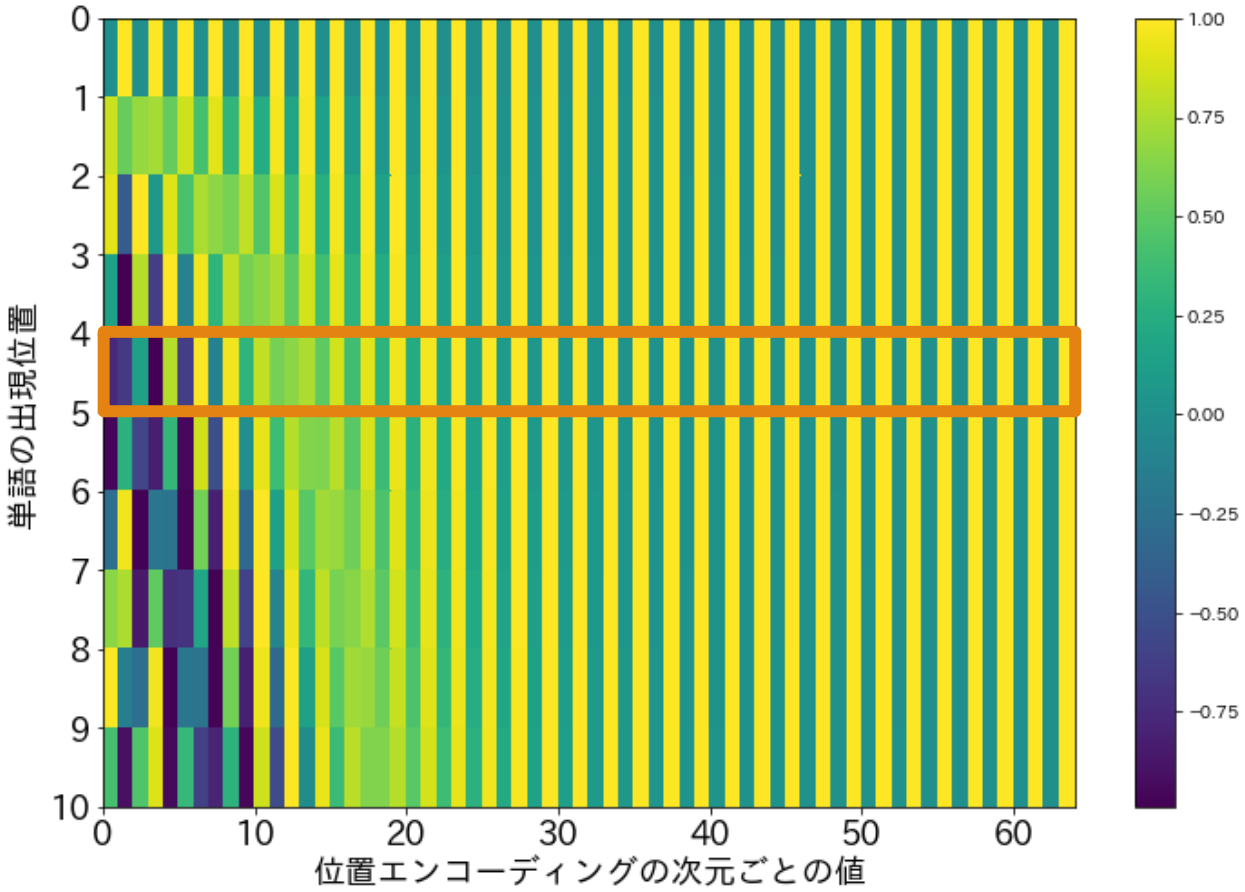
エンコーダーとデコーダーの接続



- エンコーダー出力から
  キーとバリューを作る
- デコーダー側から
  クエリを作る

位置エンコーディング

- 背景
  - ノーマルなセルフアテンションでは，文の中で単語どうしがどれくらい離れているかはまったく考慮されない
- 方針
  - 考慮できるようにするため，単語間の距離が離れているとき，分散表現での距離も離すようにする
    ⇒ 文中での位置に応じたベクトルを分散表現に加える



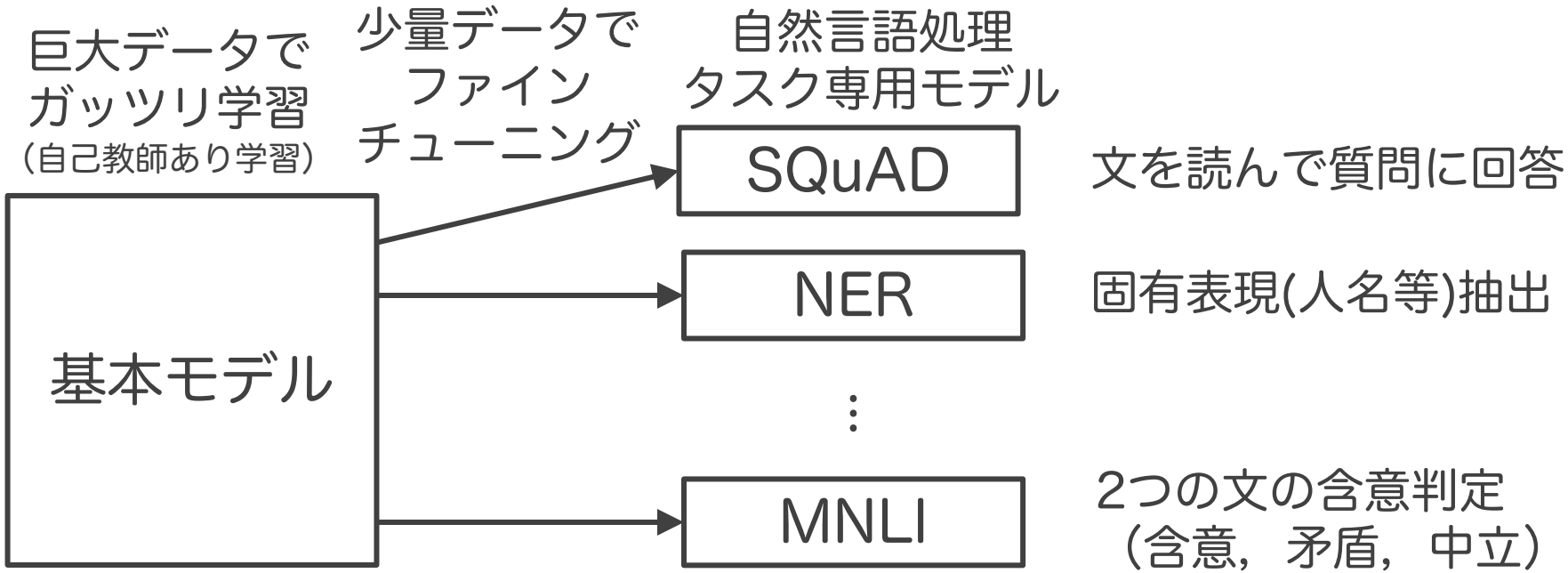
index 4の単語に
加算される
位置エンコーディング

⇒ 位置に応じて異なるベクトルが加算される

BERT

BERT概要

- ひとつの巨大モデルをファインチューニングすることで，様々な自然言語処理タスクを行う



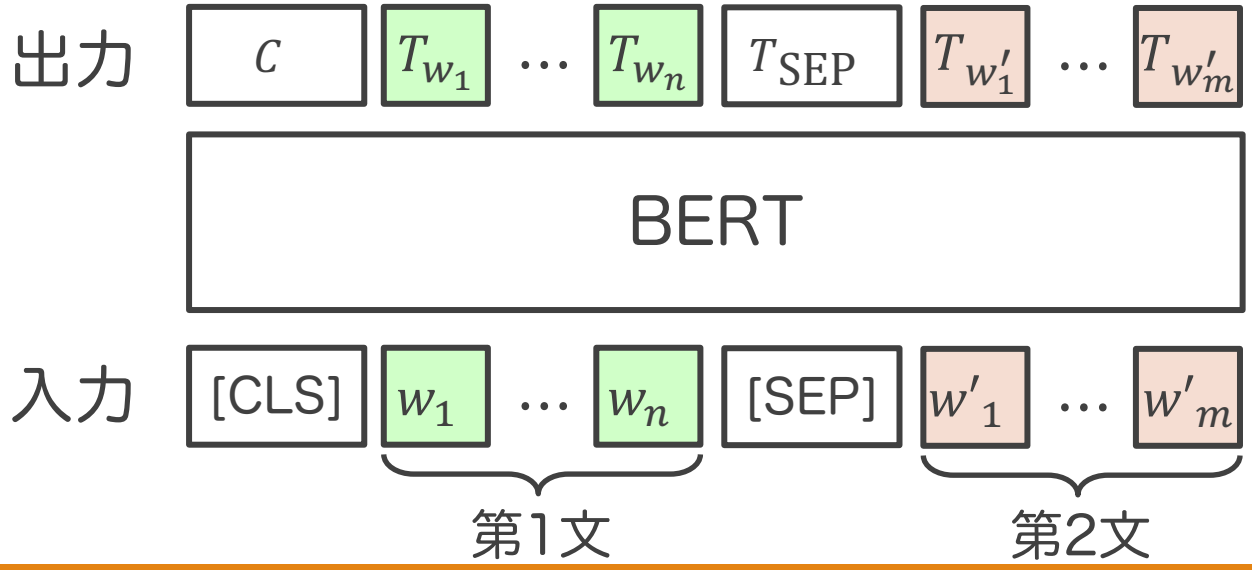
BERT概要

- 長所
  - 基本モデルの学習にはラベルが不要.
    大量の自然言語データがあればよい.
    ⇒ 学習済みモデルはHugging Face等で入手可能
  - 少量のラベル付きデータで様々なタスクに対応可能

- 基本モデルは言語自体を学習（Pre-training）
- その後のファインチューニングで
  タスクへの対応方法を学習

基本モデルの構造

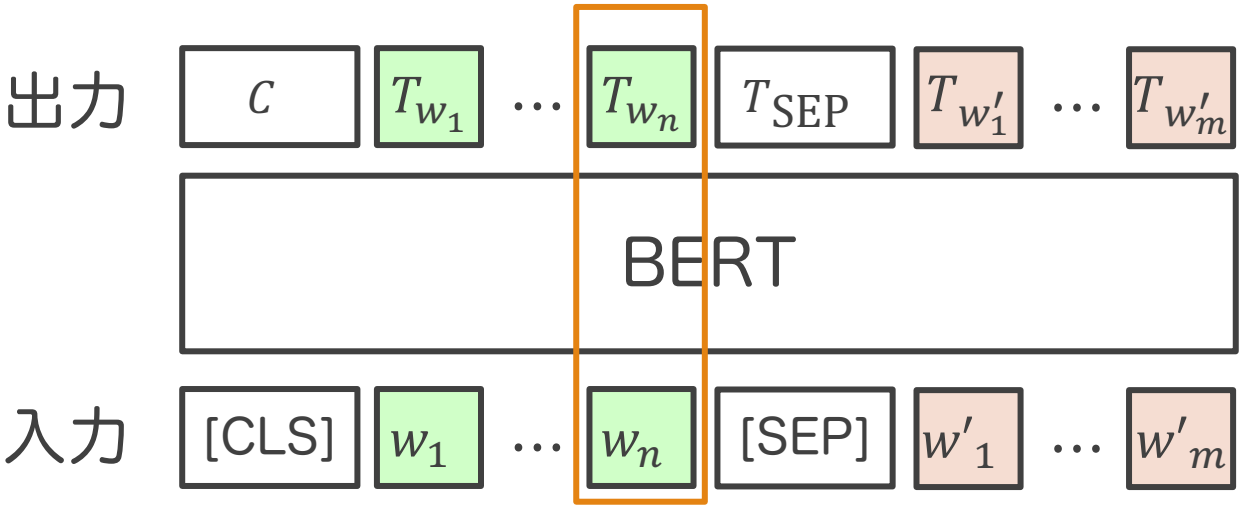
- Transformerのエンコーダー部分と同様
- 様々なタスクに対応できるようにするため，入力に2つの文を入力する



基本モデルの学習(Pre-training)

- 2つの問題を使って学習する

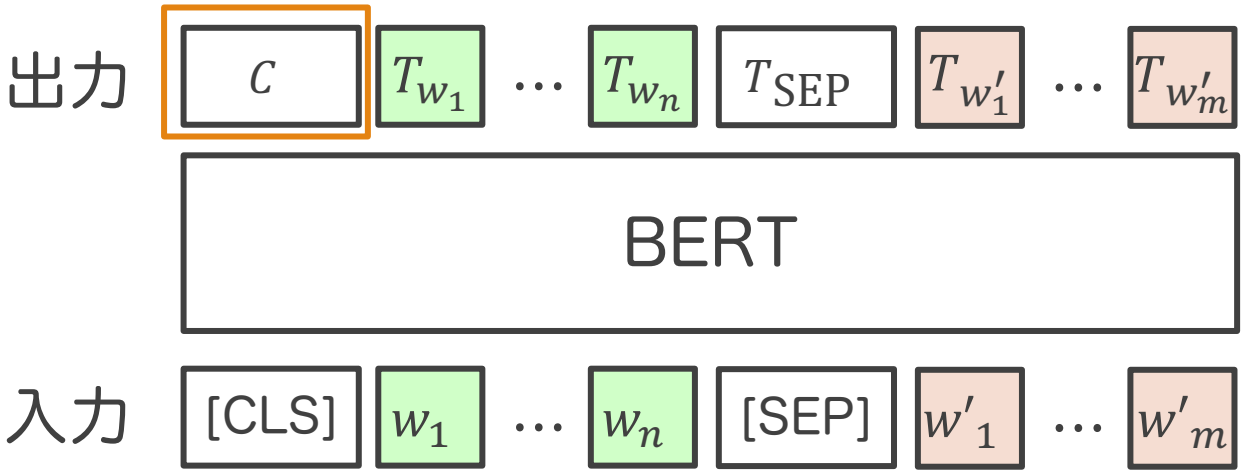
1. 穴埋め問題
   - 入力単語の15%をマスクし，他の部分から予測する
     ⇒ w_sをマスクした時はT_{w_s}を使って予測を行う



T_{w_s}は単語ごとの情報を持つ

基本モデルの学習(Pre-training)

2. 次の文予測
   - 2つの文を入力し，それが連続する2文かランダムにつなげた2文かを予測する.
     ⇒ Cを使って予測を行う ⇒ Cは文全体の情報を持つ



Hugging Face Transformers

Hugging Face Transformers

- 最新の自然言語処理モデルを活用するためのプラットフォーム
  - 様々な学習済みモデルをロードできる
  - ファインチューニングも行える（今日は省略）
  - モデルに依存しない統一的なインターフェース
  - PyTorch, TensorFlow, JAXに対応

使い方

- pipeline()
  - 最も簡単な使い方
  - タスクを指定するだけ
  - 内部モデルを自分で指定することもできる
  - タスク例
    - "text-generation"：指定した単語列の後ろに続く文を自動生成する
    - "sentiment-analysis"：入力した文がポジティブな内容かネガティブな内容かを分類する
    - "question-answering"：文を読み，それについての質問に答える

使い方

- TensorFlowモデル生成
  - タスクに合わせてモデルを生成する
  - 学習済みのモデルをロードできる
  - トークナイザー（単語分割器）も同時に指定する必要がある
    - モデルと同じものを使う必要がある