自然言語処理（11）

嶋田和孝

系列変換モデル
Sequence-to-Sequence: seq2seq

深層学習入門2

系列変換

- 多くのタスク（特に生成系のタスク）は系列変換の問題として扱える
  - 機械翻訳：日本語系列を受け取ると英語系列に変換する
  - 対話システム：入力発話を受け取るとそれへの応答発話に変換する
  - 質問応答システム：質問を受け取るとそれを回答の系列に変換する
  - 文章要約：長い文書（単語系列）を受け取ると短い文書（単語系列）に変換する
  - 文章誤り訂正：間違った文を受け取ると正しい文に変換する
- 入力系列Xを出力系列Yに変換する問題
  - あるベクトルや行列を別のベクトルや行列に写像する問題

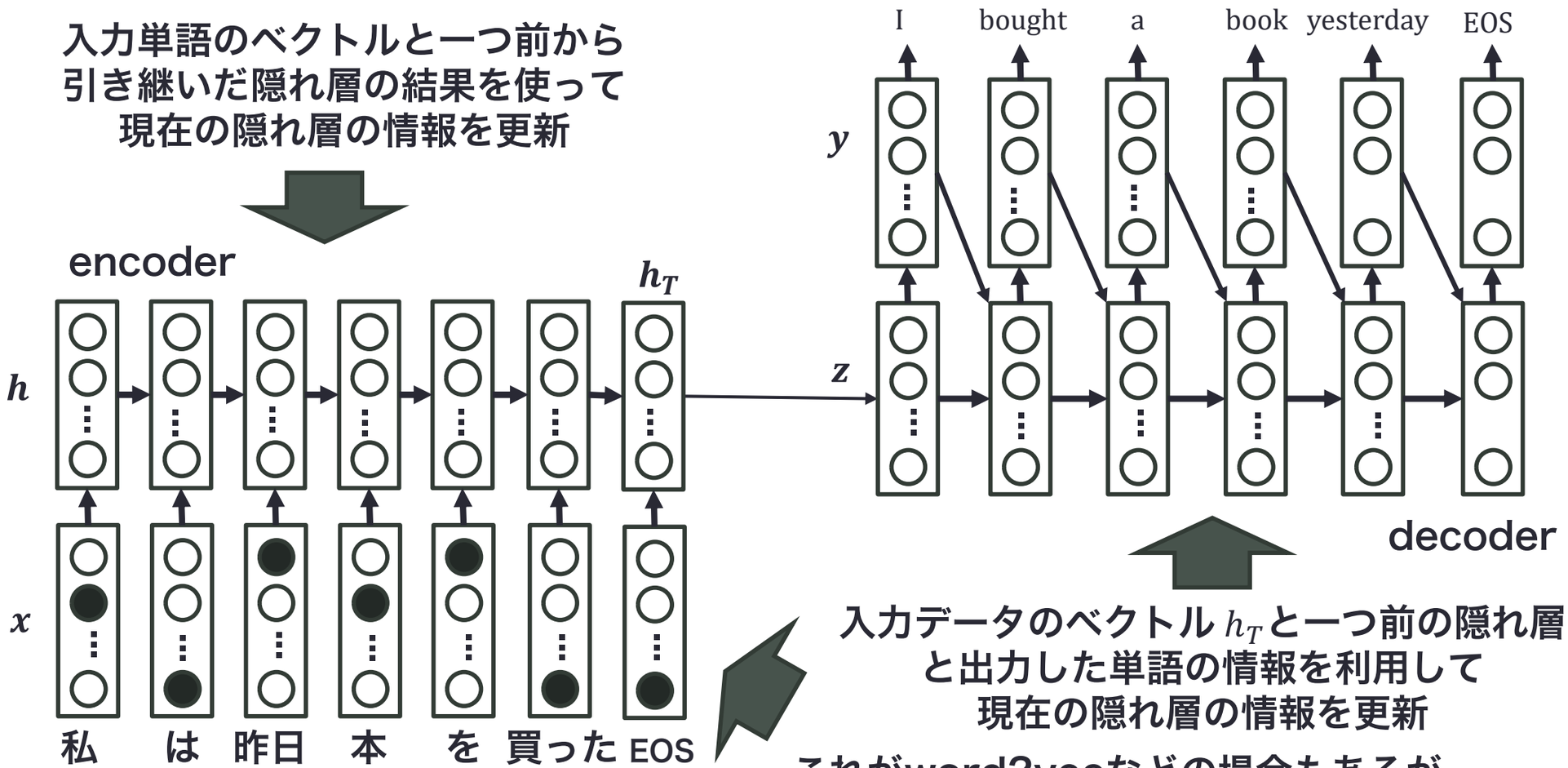
各タスクの違い

- 機械翻訳
  - 入力と出力の言語が異なる．ただし，入力と出力の意味は同じ．入力が長ければ出力も長くなる
  - 文レベルでは対応する単語の位置が入力と出力で異なる場合がある．文書レベルでは入力と出力の位置は基本的に同じ
- 対話システム
  - 入力と出力の言語は同じだが，意味は同じではない．
  - 一つの入力に対して最適な出力が一意に定まらない場合も多い．発話単位だけではなく，対話全体としての整合性が重要
- 文書要約
  - 入力と出力の言語は同じで，意味も基本は同じだが，入力に対して出力は圧倒的に短くなる
  - 入力から必要な要素を抜き出すだけでも良いし，書き直しても良い．入力に含まれる内容を基本的に使う必要がある
  - 文書全体を短くするというだけではなく，文単位で短くする（文圧縮）することもある
- 質疑応答
  - 入力と出力の言語は同じで，出力の長さは一定ではない．出力は入力に対する正しい回答でなければならない
  - 事実を返すだけで良い場合もあるし，抽象的な内容を返す必要がある場合もある
- 文章誤り訂正
  - 入力と出力の言語は同じで，長さも基本的に同じで，意味も同じ
  - 文法的な修正（軽微：文長は変わらない）から流暢性や論理性などを考えた修正（大幅な修正：文長は大きく変わる）まで，いろいろな粒度がある

エンコーダとデコーダ

- エンコーダとデコーダを用意し，系列変換をする
  - エンコーダ・デコーダモデルやsequence-to-sequenceモデル，seq2seqモデルなどと呼ばれる
    - エンコーダ：入力単語列をベクトルに変換していく処理
    - デコーダ：もらったベクトルを単語列に変換していく処理
- エンコーダやデコーダにはRNNやLSTMなどを用いる（次回説明するTransformerの登場までは）

encoder-decoder model: seq2seq



アテンション機構

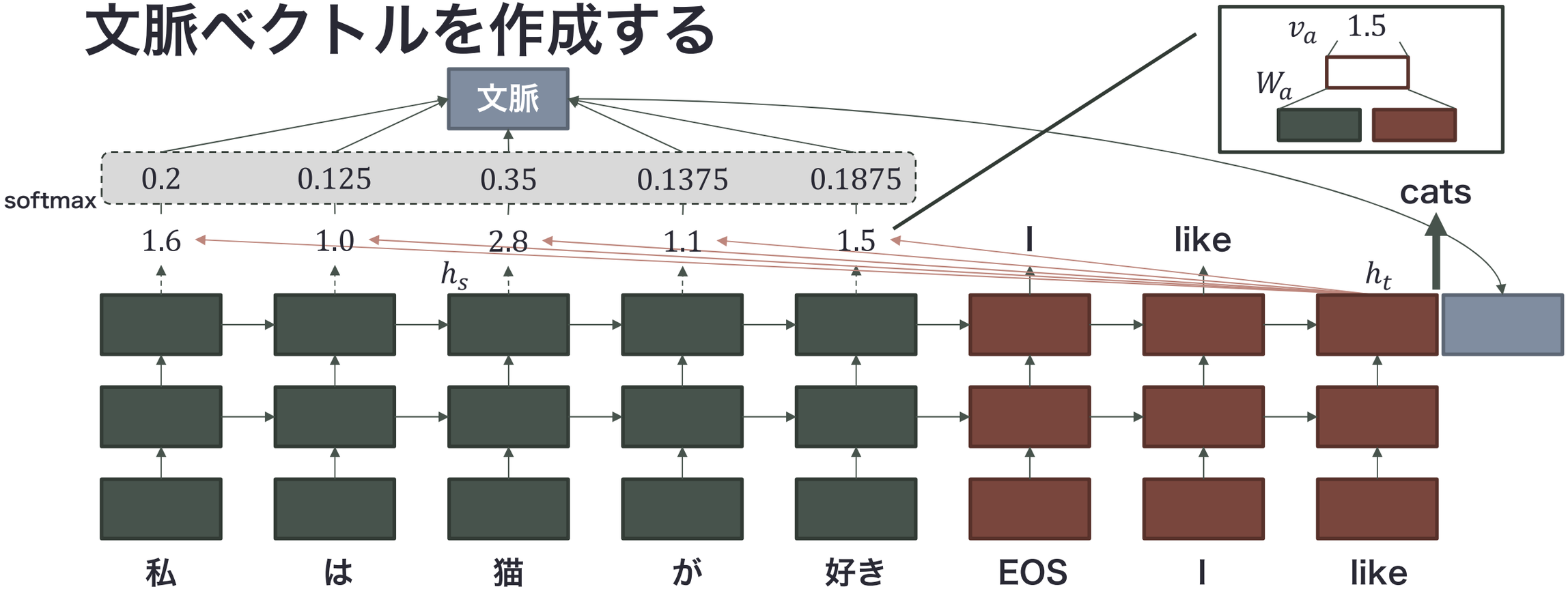
深層学習入門2

エンコーダ・デコーダモデルの問題点

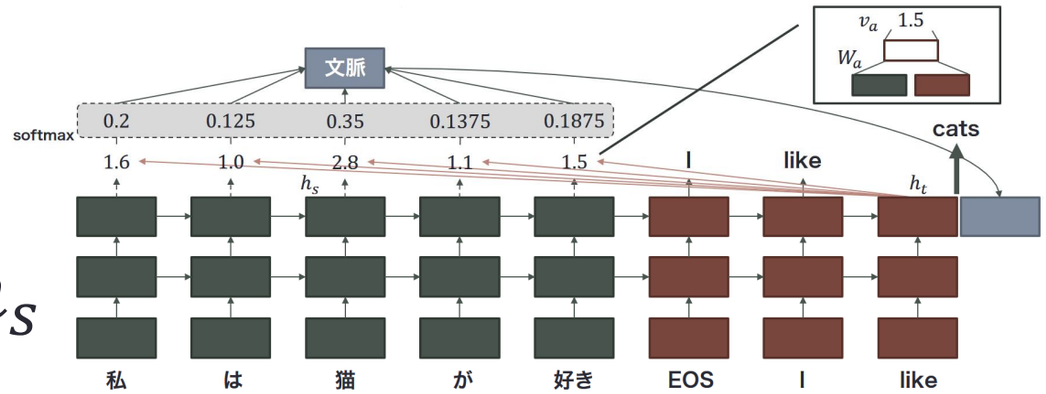
- 固定長のベクトル（エンコーダ側のEOS入力後）で文全体を表現するには限界がある
  - 変換する系列の特性が異なると特に
    - 例）英語とフランス語の翻訳は語順が類似しているため，比較的上手くいく．一方で，英語と日本語では語順が大きく異なるので，長い文の場合，単語間の関係性を上手く捉えられない
  - ある語を生成するとき，すべての単語が平等に重要であるわけではない
    - 私は猫が好き→I like [??]

アテンション機構

- 入力の各単語についてどのくらい注目するのかを計算し，文脈ベクトルを作成する



アテンション機構



- 重要度を入力の各単語の隠れ層h_sと対象単語のh_tから求める
  - $score(h_t, h_S) = v_a^\top \tanh(W_a \begin{bmatrix} h_t \\ h_s \end{bmatrix})$：図右上の部分
    - W_aとv_aは学習されるパラメータ
- 各単語の重要度に対してsoftmaxを適用し，それを文脈ベクトルと見なして h_tと合わせて出力単語を予測する
  - 文脈ベクトルは各h_tで異なったものになる
    - この例では他の単語よりも「猫」の重要度が高いということを踏まえて生成ができる

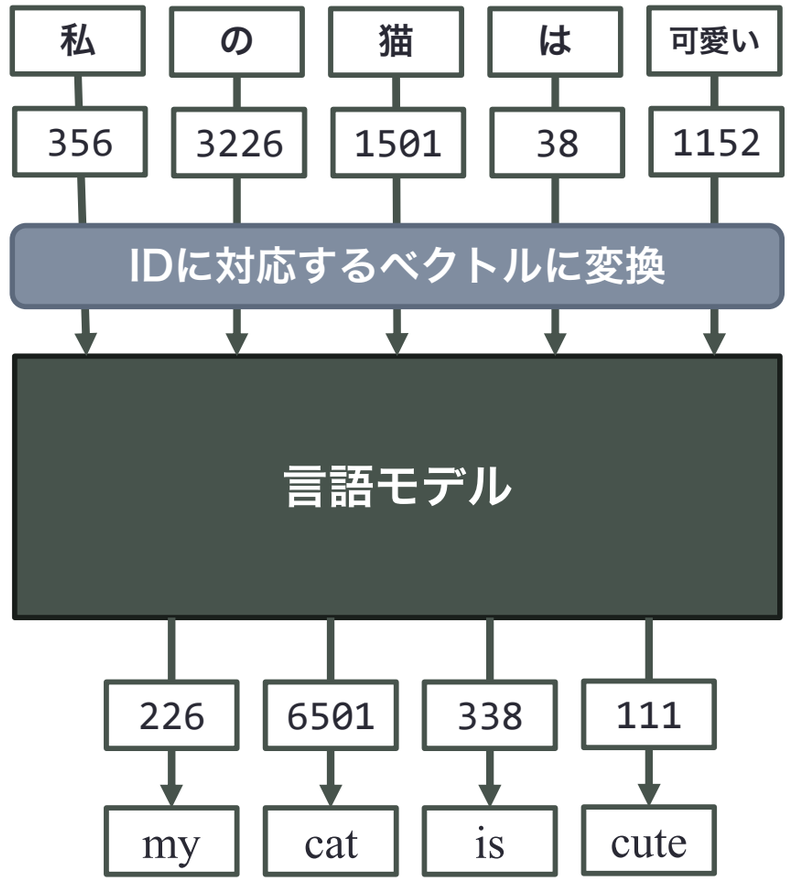
トークナイザ

深層学習入門2

トークナイザ

- ニューラルベースの手法
  - 入力をトークン単位に変換：エンコード
    - トークン→トークンIDに変換
  - トークンIDに対応するエンベッティング（ベクトル）を使って各種演算
  - エンベッティングをトークンに変換：デコード
    - トークンID→トークンに変換
- トークンのエンコード・デコードをする部分をトークナイザと呼ぶ

私の猫は可愛い



ニューラルモデルの課題

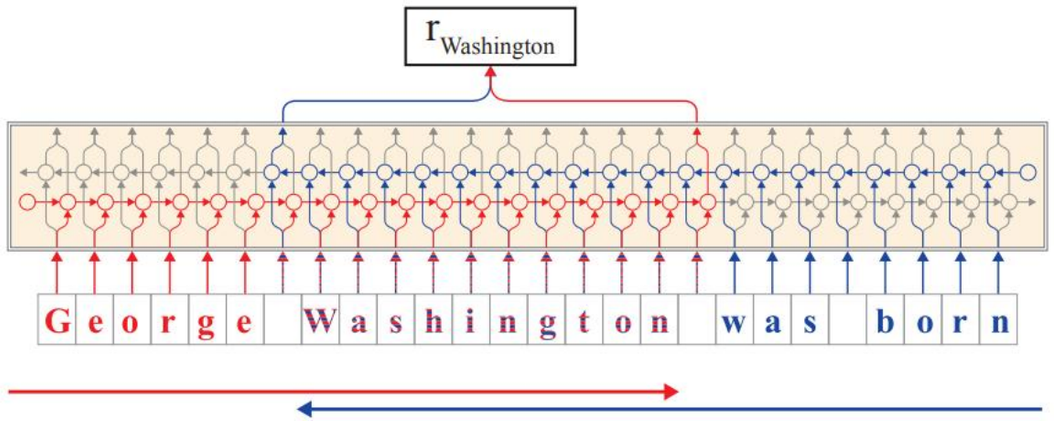
- 旧来の自然言語処理では単語は形態素解析などを経て決定されていた
- 語彙サイズ$|V|$が増えるとそれだけ計算量が増える
  - 語彙サイズは小さくしたい
    - たとえば，頻度上位3万単語だけ，などにする
      - 未知語が増える（[UNK]という単語で表現することになる）
        - I like [UNK]. みたな文になってしまう
    - その上，十分な語彙数とはいえない状況になる
  - 語彙サイズが大きい≒予測が難しくなる（候補が増えるので）
  - そもそも言語では新語などをどう対応するかは重要な問題
    - 新しい単語は日々生まれる
  - 未知語になる単語は固有名詞などであることも多く，応用タスクでは致命傷になるかもしれない
    - [UNK]をあとからルールベースで置き換えるという手はあるがあまり本質的な解法とはいえないかもしれない

トークナイズの種類

- 単語トークン
  - word2vecなどの埋め込み表現の単位でよく使われる
    - 未知語への対応が難しく，語彙数が増えてしまうという問題がある
- サブワードトークン
  - 単語と文字の中間：頻出単語は1つの単語として，そうでないものは組み合わせで表現
    - 現在の主流
- 文字トークン
  - 1文字単位で処理をする
    - シンプルで（文字が増えない限り）未知語が生じる可能性はない
    - 一方で言語モデル内での処理は複雑になる
- バイトトークン
  - Unicode（文字コード）単位で処理をする
    - 多言語環境で有用な手法
- サブワードトークンとバイトトークンを組み合わせるなどの方法もある
- 特殊なトークンを用意している場合もある
  - たとえば，GPT4ではタブなどをスペースとして扱ったり，`elsif`に対して専用トークンを用意している

文字単位で処理をする場合の例

- 単語なので未知語が起きる
  - 処理単位を単語から文字にすればよい
    - 基本的に文字の数は増えない（英語ならアルファベット＋α）
    - 文字数の多い日本語でも数千文字程度
  - 文字単位で予測するため，一文あたりの予測回数が増え，全体として誤りが多くなる可能性はある
- 実際に文字単位から単語ベクトルを作る手法もある
  - Contextual String Embeddings for Sequence Labeling



図は A. Akbik, et al. Contextual String Embeddings for Sequence Labeling. COLING2018. より

サブワード分割

部分単語（サブワード）の導入

- 単語では語彙が多くなりすぎ，文字では単位が細かすぎる
  - その中間として部分単語（サブワード）を考える
- データセットから統計的に分割単位を決定する
  - 指定された語彙数になるように，データから部分単語を生成する
  - 基本的な考え方
    - 頻度の多い単語は，そのままの単語として
    - 頻度の少ない単語は，部分文字列の組み合わせで表現

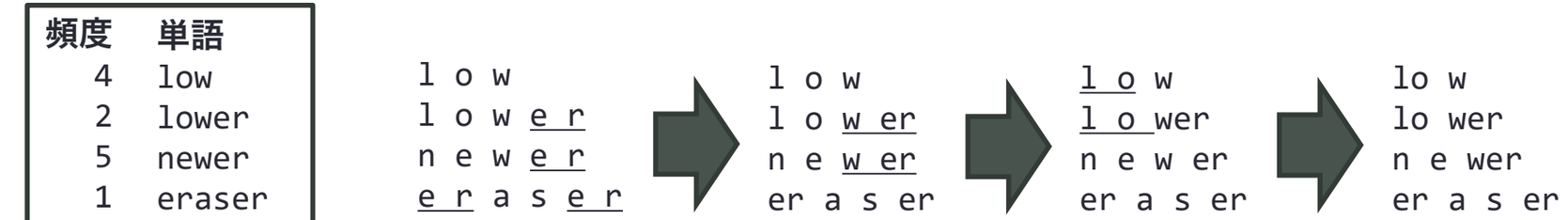
Byte-Pair-Encoding: BPE

- 手続き
  - 最初に文字の単位まで分割する
  - 連続している最頻出のサブワードペア（最初は2文字）を見つけてサブワードとする
  - これを指定した回数繰り返し，結合ルールを獲得する

結合したもの以外はそのまま（文字単位）でサブワードとする

設定：結合回数=3

頻度	単語
4	low
2	lower
5	newer
1	eraser



loの組み合わせを含む単語の頻度が6

頻度	単語
6	lo
4	w
7	wer
5	n
5	e
2	er
1	a
1	s

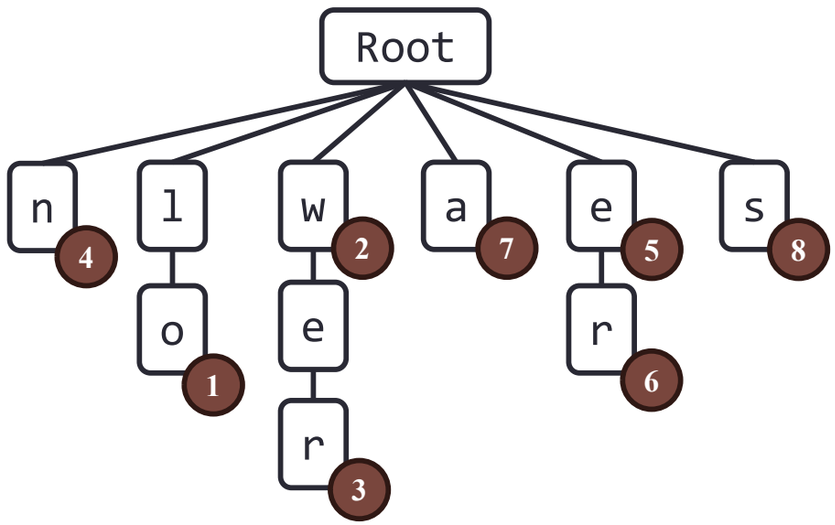
サブワードのメリット

- たとえば higher とか lowest とかが，highと##erやlowと##estのような組み合わせで表現できるようになる（##は前の単語とくっつくという意味を表す）
  - high/higher/highest/low/lower/lowest : 6単語
  - high/low/##er/##est : 4単語
- BPE以外にもいくつか手法はある
  - WordPiece
    - BPEをベースにするが，最も頻度が高いものを選ぶのではなく，結合前のスコアも加味して処理をする
    - スコア＝サブワードの組の頻度 / (1つめのサブワードの頻度＋2つめのサブワードの頻度)
  - SentencePiece(BPE+uni-gram言語モデル)
    - 英語などは空白があるので単語の認定が簡単だが，日本語などはそうではない
    - 語彙を作る際に単語単位で考えるのではなく，文単位で考えることでこの問題を解決する
      - 文を生成する確率が高くなるようにサブワードを分割する
  - 日本語の場合，形態素の情報を活かすために，形態素解析後のデータにBPEやSentencePieceを適用することがある
    - 一方で単語に区切るという手続きが入ると，言語依存になり，多言語処理の場合にそれが足かせになる場合がある

実際のトークナイズとデータ構造

- エンコードは入力から最長一致で対応するトークンを探すこと
- デコードはトークIDから出力を決めること
  - デコーダはハッシュでOKだが，エンコーダはそうではない
    - 効率的な辞書構造が必要：トライによる情報の管理

token	ID
lo	1
w	2
wer	3
n	4
e	5
er	6
a	7
s	8



エンコード

```
lower: l→o(1), w→e→r(3)
 TokenID列={1, 3}

newer: n(4), e(5), w→e→r(3)
 TokenID列={4, 5, 3}
```

デコード

```
TokenID列 {7, 8}
  出力単語: as
```

今回のまとめ

- 深層学習入門（2回目）
  - 系列変換モデル：言語処理を系列変換のモデルとして置き換える
  - アテンション機構：系列変換の際に各単語の重要度を考える
  - トークナイザとサブワード：語彙サイズを小さくし，未知語の問題を改善する