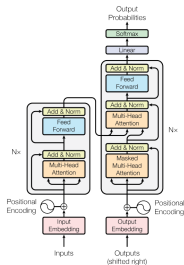


Transformer

BytePair encoding (BPE)



Learning goals

- Understand inner workings of BPE
- Being able to compare BPE to other tokenization approaches

BYTEPAIR ENCODING (BPE)

Data compression algorithm ► Gage (1994)

- Considering data on a *byte*-level
- Looking at pairs of bytes:
 - ❶ Count the occurrences of all byte pairs
 - ❷ Find the most frequent byte pair
 - ❸ Replace it with an unused byte
- Repeat this process until no further compression is possible

BYTEPAIR ENCODING (BPE)

Open-vocabulary neural machine translation ► Sennrich et al. (2016)

- Instead of looking at bytes, look at characters
- Motivation: Translation as an open-vocabulary problem
- Word-level NMT models:
 - Handling out-of-vocabulary word by using back-off dictionaries
 - Unable to translate or generate previously unseen words
- Using BPE effectively *solves* this problem, except for ..
 - .. the occurrence of unknown characters
 - .. when all occurrences in the training set were merged into "larger" symbols (Example: "*safeguar*" and "*safeguard*")

BYTEPAIR ENCODING (BPE)

Adapt BPE for word segmentation ► Sennrich et al. (2016)

- *Goal:* Represent an open vocabulary by a vocabulary of fixed size
→ Use variable-length character sequences
- Looking at pairs of characters:
 - ➊ Initialize the the vocabulary with all characters plus end-of-word token
 - ➋ Count occurrences and find the most frequent character pair, e.g. "A" and "B" (⚠ Word boundaries are **not** crossed)
[Side effect: Can be run on a dictionary w/ frequency counts]
 - ➌ Replace it with the new token "AB"
- Only one hyperparameter: Vocabulary size
(Initial vocabulary + Specified no. of merge operations)
→ Repeat this process until given $|V|$ is reached

```
1 import re, collections
2
3 def get_stats(vocab):
4     pairs = collections.defaultdict(int)
5     for word, freq in vocab.items():
6         symbols = word.split()
7         for i in range(len(symbols)-1):
8             pairs[symbols[i],symbols[i+1]] += freq
9     return pairs
10
11 def merge_vocab(pair, v_in):
12     v_out = {}
13     bigram = re.escape(' '.join(pair))
14     p = re.compile(r'(?!\S)' + bigram + r'(?!\S)')
15     for word in v_in:
16         w_out = p.sub(' '.join(pair), word)
17         v_out[w_out] = v_in[word]
18     return v_out
```

EXAMPLE – MERGING

► SENNRICH ET AL. (2016)

```
1 vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,  
2         'n e w e s t </w>':6, 'w i d e s t </w>':3}  
3  
4 pairs = get_stats(vocab)  
  
1 >>> print(pairs)  
2 defaultdict(<class 'int'>, {  
3     ('l', 'o'): 7, ('o', 'w'): 7, ('w', '</w>'): 5,  
4     ('w', 'e'): 8, ('e', 'r'): 2, ('r', '</w>'): 2,  
5     ('n', 'e'): 6, ('e', 'w'): 6, ('e', 's'): 9,  
6     ('s', 't'): 9, ('t', '</w>'): 9, ('w', 'i'): 3,  
7     ('i', 'd'): 3, ('d', 'e'): 3  
8 })  
  
1 best = max(pairs, key=pairs.get)  
2 vocab = merge_vocab(best, vocab)  
  
1 >>> print(best)  
2 ('e', 's')  
3 >>> print(vocab)  
4 {'l o w </w>': 5, 'l o w e r </w>': 2,  
5  'n e w e s t </w>': 6, 'w i d e s t </w>': 3}
```

EXAMPLE – MERGING

► SENNRICH ET AL. (2016)

```
1 vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,  
2         'n e w e s t </w>' : 6, 'w i d e s t </w>' : 3}  
3  
4 num_merges = 10  
5  
6 for i in range(num_merges):  
7     pairs = get_stats(vocab)  
8     best = max(pairs, key=pairs.get)  
9     vocab = merge_vocab(best, vocab)  
10    print(best)
```

```
1 ('e', 's')  
2 ('es', 't')  
3 ('est', '</w>')  
4 ('l', 'o')  
5 ('lo', 'w')  
6 ('n', 'e')  
7 ('ne', 'w')  
8 ('new', 'est</w>')  
9 ('low', '</w>')  
10 ('w', 'i')
```