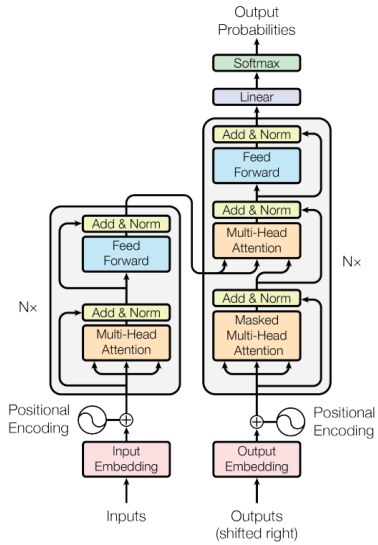
Transformer

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:
  1. Count the occurrences of all byte pairs
  2. Find the most frequent byte pair
  3. 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 occurence of unknown characters
  - .. when all occurences 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
  $\rightarrow$ Use variable-length character sequences
- Looking at pairs of characters:
  1. Initialize the the vocabulary with all characters plus end-of-word token
  2. 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]
  3. Replace it with the new token "AB"
- Only one hyperparameter: Vocabulary size
  (Initial vocabulary + Specified no. of merge operations)
  $\rightarrow$ Repeat this process until given $|V|$ is reached

EXAMPLE – SETUP

▸ SENNRICH ET AL. (2016)

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out
```

EXAMPLE – MERGING

▸ SENNRICH ET AL. (2016)

```
vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}

pairs = get_stats(vocab)
```

```
>>> print(pairs)
defaultdict(<class 'int'>, {
            ('l', 'o'): 7, ('o', 'w'): 7, ('w', '</w>'): 5,
            ('w', 'e'): 8, ('e', 'r'): 2, ('r', '</w>'): 2,
            ('n', 'e'): 6, ('e', 'w'): 6, ('e', 's'): 9,
            ('s', 't'): 9, ('t', '</w>'): 9, ('w', 'i'): 3,
            ('i', 'd'): 3, ('d', 'e'): 3
})
```

```
best = max(pairs, key=pairs.get)
vocab = merge_vocab(best, vocab)
```

```
>>> print(best)
('e', 's')
>>> print(vocab)
{'l o w </w>': 5, 'l o w e r </w>': 2,
 'n e w es t </w>': 6, 'w i d es t </w>': 3}
```

EXAMPLE – MERGING

▸ SENNRICH ET AL. (2016)

```
vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}

num_merges = 10

for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

```
('e', 's')
('es', 't')
('est', '</w>')
('l', 'o')
('lo', 'w')
('n', 'e')
('ne', 'w')
('new', 'est</w>')
('low', '</w>')
('w', 'i')
```