DLP Week-2 Notes

L1 Tokenization

- Transformer is a simple encoder-decoder model with attention mechanism at its core.

Input —Task→ Output

len(seq. of words) ≠ len(seq. of words)

- Abstract: Tokenizer Pipeline

Text → Encoder
↓
Tokenizer (tokens)

split input seq. into token.
simple → use whitespace to split the text.

↓

Token ID — Each token gets a unique ID.

↓

Embeddings — Each ID gets a unique embedding vector

↓

LLM — Model predicts token IDs.

↓

ID to token
↓
Token to words

Decoder

Now decoder takes IDs and forms token. In the post-processing step, the tokens are finally converted to words.

- size of vocab = size of embeddings.

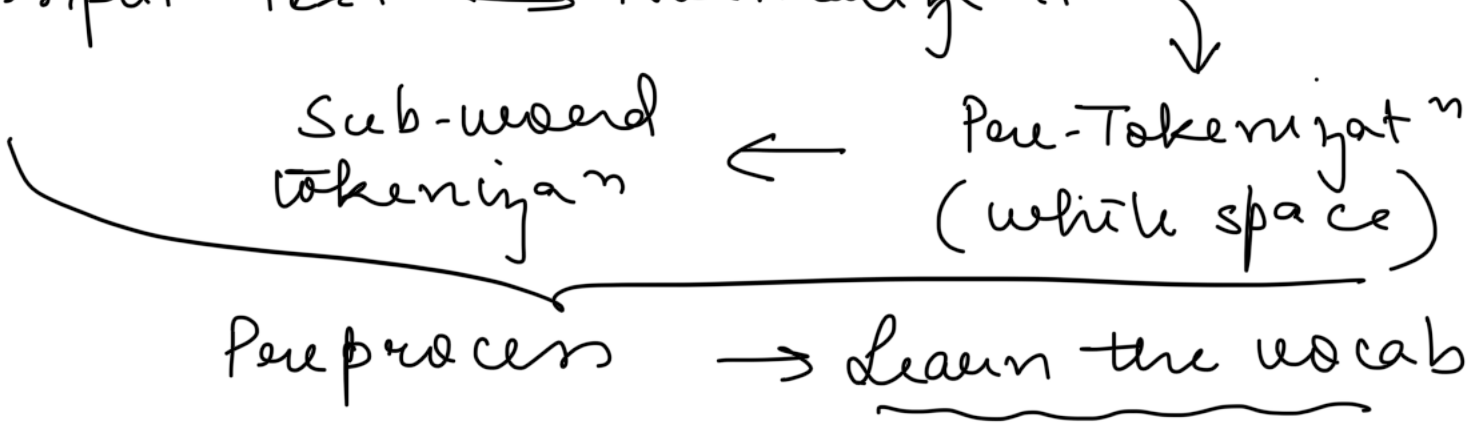
- how to build a vocab.
- Pre-tokenizatⁿ — splitting text into words
- some spl. tokens → <go>, <cls>, <sep>, <pad>, <unk> are also used in BERT trainers

L5 HF Tokenizer (Half)

NOTE → Few things are needed before going L2, L3, & L4

- is whitespace a good delimiter? Then enjoyed & enjoy are diff.
- but, we also can't have each characters as tokens. Too large vocab
- few challenges —
  1. what is optimal size of vocab?
  2. If vocab size is 50K, how to deal with out of vocab words.
  3. How to handle small spelling mistake words?
  4. Open vocab problem. New words.
- wishlist
  1. moderate size V.
  2. efficiently handle unknown words
  3. be lang. agnostic
- charac. level — 1 - 112 V size
  subword level — 30 - 50 K → types
  word level — >200 K

L2 Byte Pair Encoding



- Algo →
  1. start with a dict., containing words & count
  2. append <w> at each word
  3. set req. no. of merges (HPT)
  4. create a charac. freq. table (base V)
  5. get the freq. count for a pair of charaes.
  6. merge pairs with max occurence.
- Obj. → find most frequently occuring byte-pair

Word count → gives the vocab size
initial vocab size → charac. based

- final vocab has initial vocab + all merges. The rare words broken down into 2 or more subwords.

Fertility (std 2-3) → knowing 2

word $\xrightarrow{\text{tokenizer}}$ tokens

- BPE for non-segmented lang. (Japanese)

L3 Word-Piece Tokenizer

- BPE → pair of tokens with highest freq.

$$\text{score} = \frac{\text{count}(\alpha, \beta)}{\text{count}(\alpha)\,\text{count}(\beta)} \qquad \frac{('i','n')}{(i),(n)}$$

merging occurs on the basis of score of each byte pair

reverse?

large vocab $\xrightarrow[\text{eliminations}]{N}$ 30-50 K sub-word wd. Tokenizers

L4 Sentence Piece Tokenizer

- any word can have numerous subwords
  - ↳ possibility - 2^K combinations
    - ↳ no. of characters
- follows the merge rule, while merging
- BPE is greedy & deterministic

$$x^* \in \{x_1, x_2, \ldots x_K\}$$

maximize the likelihood of word x.

$$P(x) = \prod_{i=1}^{n} P(x_i)$$

$$\sum_{x \in V} p(x) = 1$$

$$x^* = \text{argmax}\, P(x)$$

$$P(x_1 = k, now, ing)$$
$$= P(k)\, p(now) p(ing)$$
$$= \frac{3}{16} \times \frac{3}{16} \times \frac{7}{16} = \frac{63}{4096}$$

x_2 ---

x_3 ---

$$L = \sum_{s \in 1}^{|D|} \log \left(\sum_{x \in S(X^*)} P(x) \right)$$

1. Construct a large seed vocab using BPE
2. E step → estimate the probab. for every token using freq. tokens.
3. M step → use Viterbi algo. to segment & return optimal segments that maximises the log likelihood.
4. compute likelihood for new segments
5. shrink by removing bottom x% of subwords.

Eg. whereby

L5 HF Tokenizers (contd.)

Normalizer
PreTokenizer
Tokenizer (With V)
Post Processing

from tokenizers import tokenizers

normalizer → lowercase, strip accents
pre-tokenizer → whitespace, BERT, Regex
Algo. → BPE, WordPiece
Post processor → insert model specific tokens

customizaⁿ at each step

- after setting the normalizer & pre-tokenizer just call the -train methods.

L6 Building & Training a Tokenizer

- bookcorpus lib. dataset (74M, 5GB)
- Note — len(sentences) does not affecting the tokenizer training
- BPE ([UNK]) → few unknown tokens.
- create a trainer with vocab size & special tokens.
  ⇒ continuing_subword-prefix = "##"
- save it somewhere safe.
  → the early tokens are small characters but last one are like big words.
  total merges ≤ vocab-size
  → slider to view animaⁿ.

L7 Encoder & decoder

- pass the single sample to 'encode' method of tokenizer class.
- we can use a visualizer to see how the split works
- atten^n mask → 1 (pay atten^n to every word)
  ↳ BERT models (type id → 0, 1)
- when you have a batch of samples, then use [PAD] token to make sure samples do not have irregularity.
- in general, it's fine to increase the length of a seq. to exceed the model's content length.
- hf → h ##f , ☺ → [UNK]
- always try to save the tokenizer.
- BERT tokenizer ——— sfc.

  <CLS> <SEP> <MASK>

  at start btw. 2 sentences
- post processing step → Template Processing

  single =

  pair =

  special_tokens = [( ), ( )]
- Now, decode,

  remove sp. tokens, merge the ## thing

18 Wrap With Pre-Trained Tokenizer

- PreTrained Tokenizer class.
  file, unk token, pad token
  ↓
  you can use this to create a model input.
- Padding is not by default. You have to do it manually.