Chapter 6 Tokenizers Library

Saturday, August 5, 2023 12:24 PM

2.1 Model scratch থেকে train করার brand new tokenizer train করতে হবে।

HF tokenizer library

fast tokenizer

2.2 Training a new tokenizer from old one:—

model x available

Corpus is different from PT corpus

retrain model from scratch. need a new tokenizer

Most Xmer model use subword tokenization algo.

Training a tokenizer:—

X same as Training a model

randomized by nature

Set some seeds to get same results when doing same training

Statistical process

which subwords to pick depend on tokenization algo.

Deterministic

Assembling a Corpus:—

Autotokenizer.train_new_from_iterator()

GPT-2 from scratch on Python code.

Code_search_net

Dissimilarities arise from:—

New lang.

" characters

" domain

" style

6.3 Fast Tokenizer's Special powers:—

6.3 Fast Tokenizer's Special powers:-

Slow tokenizers written in Python
Fast " " " Rust.

Special tokens
↑
Model
↑
Pre-tokenization — splits
↑
Normalization — lower case, removing accents

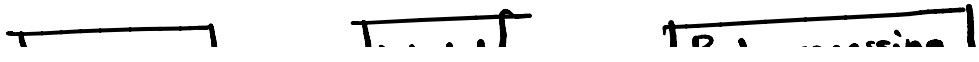
Word IDs application
- whole word masking
- token classification

Offset mapping application
- token classification
- QA

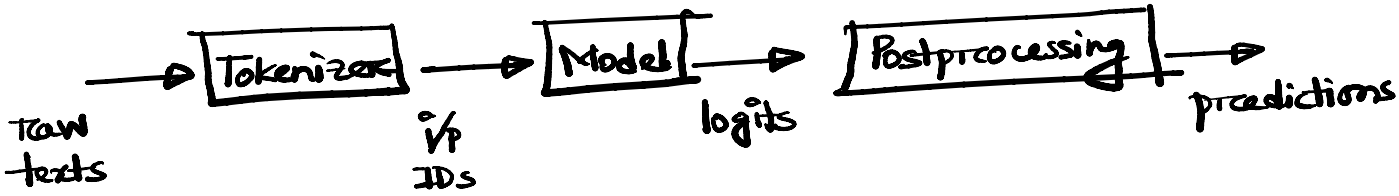
O/p of tokenizer
- x simple python dictionary
- Special batch encoding object.
- Subclass of dict.

Fast tokenizers
- parallelization capabilities
- keep track of original span of texts → offset mapping

Token classification pipline steps:-

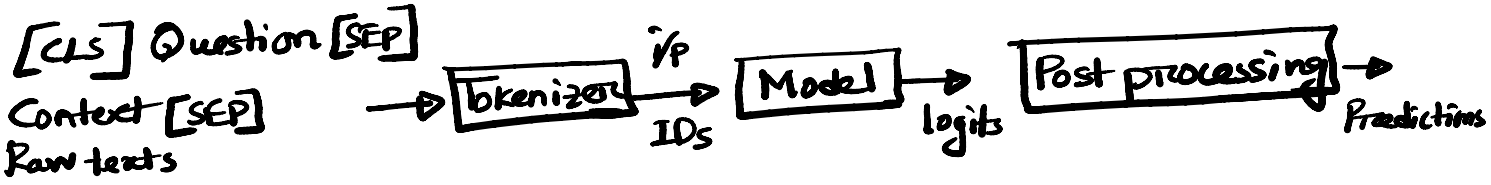


Token classification pipline steps:-



1 2 3 4
5 6 7 8

6.4 Fast tokenizer in QA pipeline:-



6.5 Normalization & pre-tokenization:-

Hello, how are U today?

hello how are u today?

[hello, how, are, u, today, ?]

[hello, how, are, u, to, ##ay, ?]

[CLS, hello, how, are, u, to, ##ay, ?, SEP]

Normalization

Pre-tokenization

Model

Postprocessors

Normalization operation automatically included when tokenizing

w/o " text tokenize করার পরে অনেক <unk>/<OOV>

bytes 01100011 11001100 10100111 11000011 10100111

Decoding (UTF-8)

Code point(s) U+0063 U+0327 U+00E7

Unicode mapping

Character ç ç

character

Unicode normalization standards:—

NFC, NFD, NFKC, NFKD

Sentencepiece algorithm:—

Consider text as a seq. of unicode char.

replaces spaces with special char. —

with unigram algo.

x req. pre-tokenization

easily reverse করা যায়

replace _ with space

Model	Byte pair Encoding	Wordpiece	Unigram
Training	Starts with small vocab. & learn rules to merge tokens.	← Same	with large vocab. learn rules to remove tokens.
Training Step	Merge tokens corresponding to most common pair.	on best score	remove tokens that min. loss
Learns	Merge rules & Vocab.	Just a vocab.	a vocab. with a score for each token.
Encoding	Splits a char. & apply merges during training	finds longest subword then same for rest	

Byte-Pair Encoding (BPE) Algorithm:—

for compression of texts

Open AI for tokenization during training GPT

Open AI for tokenization during training ছিল।

(Ro) DeBERTa.

Normalize → Pretokenize → প্রত্যেক word এর জন্য word counter add করতে হবে। → Split into characters. → দুইটা দুইটা করে character নিয়ে pair তৈরি করলো frequency সহ।

highest freq. pair কে vocab. & splits এ add করলো।

BPE training algo. details

unique sets of words in the corpus (after Normalization & pretokenization)

Building the vocab. by taking all symbols to write all words

Contain all ASCII char. & some unicode char.

If char. x in corpus, during tokenizing texts unknown token

very bad with emojis.

GPT-2, RoBERTa

x unicode char

✓ u Bytes

256 size

every char. included

Byte-level BPE

Base Vocab. দিয়ে শুরু করে

add new tokens

until desired vocab. size reached.

Start learning merges

merges 2 char. into 1

Search most freq. pair of existing tokens

Tokenization Algorithm:—

search most freq. pair of ...

Tokenization Algorithm:—

Normalization

Pre-tokenization

Splitting words into chars.

6.6 Word Piece Tokenization:—

Google

PT BERT

Sim. to BPE but tokenization is diff.

✗ open sourced.

Training Algorithm:—

Small vocab.

special tokens

initial alphabets

Word

w ##o ##r ##d

learns merge rules

Score = $\frac{\text{freq of pair}}{\text{freq. of first element} \times \text{freq. of second element}}$

Tokenization Algo:—

WP only saves final vocab.

✗ merge rules learned.

finds longest subword

Split it hugs → hug, ##s

bugs → b, ##u, ##gs

bun → [UNK] BPE

→ b, ##u, [UNK] WP

→ b, ##u, [UNK] WP

6.8 Unigram Tokenization:-

Starts with big vocab.
removes tokens until reaches desired vocab. size.
big vocab build করার way
most common substrings in Pretokenized
apply BPE on corpus with large vocab.
Training এর সময় unigram algo. loss calc করে
overall loss ↑ হবে কতটুকু if one symbol removed.
find out the symbol will have lower effect on overall loss.
one symbol remove করা অনেক costly
p(x) symbol ω করে ২% symbols associated with lowest loss increase.
Process is repeated until desired vocab. size is reached.
X remove base char.

Tokenization algo.
independent of context
predict most common token.
probability = $\frac{\text{freq. in original corpus}}{\text{sum of all tokens freq.}} = \frac{20}{210}$
tokenization with least tokens will have highest probabilil
Split a token into least no. of tokens possible.
pug → p u, g / p, ug
depends on which segmentation encountered first

Viterbi algo.

Viterbi algo.

~~find~~ build a graph to detect possible segmentations

Training:—

loss computed by tokenizing every word in corpus

$$loss = -\log(P(word))$$

hug: "hug" score: 0.071428

pug: "pu", "g" u: 0.007710

⋮

lox $(-\log(0.071428)) + 5\,(-\log(0.007710)) + 12 \ldots$

একটা remove করলে loss কম ↑ হলে সেইটা remove করবো।