Giriş

Tokenizer'lar NLP modellerinin temelini oluştururlar ve sadece bir işleri vardır: Metinleri model tarafından işlenilebilen tiplere çevirmek. Bu kısımda tokenizasyon aşamaları incelenilecektir.

Bu işlemi birçok yolla gerçekleştirebiliriz ancak optimum çözüm anlamı koruyabilen ve eğitimi kolay olduğundan dolayı küçük bir representation elde etmektir.

Word-based

Akıllara ilk gelen yöntem, cümleyi boşluklara veya başka karakterlere göre ayırmak ve kelimeleri token olarak kabul etmektir. Genelde implementasyonu kolaydır ve geçerli sonuçlar üretir. Örnek bir tokenizasyon aşağıdaki görsel gibidir:

Split on spaces

Let’s	do	tokenization!

Split on punctuation

Let	’s	do	tokenization	!

Bu tokenizer kullanıldığı zaman genelde büyük bir "vocabulary" ile karşı karşıya geliriz. Vocabulary kavramını corpus içinde bulunan, birbirinden bağımsız token'ların sayısı olarak tanımlayabiliriz.

Vocabulary içindeki her bir token 0'dan başlayarak bir ID ile tanımlanır. Eğer bir dildeki bütün kelimeleri tokenize etmeye çalışırsak çok büyük bir vocabulary ile karşılaşırız. Örneğin Ingilizce'deki her kelimeyi tokenize etmeye çalışırsak, vocabulary boyutu kolayca 500.000'i aşacaktır. Bu 500 bin adet ID ile uğraşmamız gerektiği anlamına gelir. Dahası, vocabulary içinde "dog" ve "dogs" farklı iki token olarak temsil edilecektir ve model bu iki tokeni alakasız olarak değerlendirecektir. Aynı durum "run" ve "running" gibi durumlar için de geçerlidir.

Son olarak, vocabulary içinde bulunmayan kelimeleri temsil etmek için [UNK] gibi token'lara ihtiyaç duyulur. Tokenizer'ın bu token'ı sıklıkla üretmesi genelde kötüye işarettir. Tokenizer'ın cümlelerinizi iyi temsil edemediğini gösterir, bu anlamsal kayıplara yol açabilir.

Bu problemi aşmanın yollarından biri "Character Based Tokenizer" kullanmaktır.

Character-based

Bu yöntem oldukça basittir. Kelimeler yerine karakterlere ID atanır iki ana avantaj sağlar:

- Vocabulary boyutu oldukça küçülür.
- Unknown token sayısı oldukça küçülür.

L	e	t	‘	s	d	o	t	o	k	e	n	i	z	a	t	i	o	n	!

Fakat bu yöntem kusursuz değildir ve kendi sorunlarını beraberinde getirir. Tokenizer kelimeler yerine karakterleri numaralandırdığı için cümlenin temsili büyük bir anlam taşımayacaktır. Çünkü karakterler çoğu dilde tek başlarına bir anlam taşımazlar.

Aynı zamanda bir cümleyi temsil etmek için daha çok token'a ihtiyaç duyulacaktır. Şimdi, işlenilen iki yöntemin de iyi yönlerini kullanan bir yaklaşımdan bahsedeceğiz

Subword tokenizaiton

Subword tokenization algoritmaları, sık kullanılan kelimelerin daha küçük alt kelimelere bölünmemesi, ancak nadir kelimelerin anlamlı alt kelimelere ayrıştırılması gerektiği ilkesine dayanır.

Örneğin "annoyingly" kelimesi nadir bir kelime olarak değerlendirilebilir ve "annoying" ve "ly" gibi alt kelimelere ayrıştırılabilir. Bu iki kelimenin tekrarlanması daha olağanıdır ve "annoyingly" kelimesinin anlamı bu iki kelimenin bileşimi ile elde edilebilir.

Let’s </w>	do</w>	token	ization</w>	!</w>

Bu yöntem semantik olarak daha faydalıdır çünkü "token" ve "ization" token'larını da anlamlı hale getirir. Bunlara ek olarak vocabulary boyutu kelime tabanlı tokenization'a göre çok daha küçüktür. Böylece daha büyük bir vocabulary temsil edilebilir.

Bu tip tokenizer'lar **Türkçe** gibi eklemeli dillerde kullanışlıdır.

Diğer yöntemler

Beklendiği gibi, bahsedilen yöntemler dışında da birçok yöntem vardır. Bazılarından bahsetmek gerekirse:

- **Byte-level** BPE, GPT-2'de kullanılan yöntemdir.
- **WordPiece**, BERT'de kullanılan yöntemdir
- **SentencePiece** veya **Unigram**, birkaç çok dilli modelde kullanılan yöntemdir.

Loading and saving

Tokenizer'ları yüklemek ve kaydetmek, modelleri yüklemek ve kaydetmek gibi basittir ve aynı metodlar ile gerçekleştirilir (`save_pretrained()` `load_pretrained()`). Bu metodlar algoritmayı ve vocabulary saklayacak veya yükleyecektir.

Örnek olarak, BERT tokenizer yüklemek için `BertTokenizer` sınıfını kullanabiliriz.

```
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
```

from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

sequence = "Using a Transformer network is simple" tokens = tokenizer.tokenize(sequence)

print(tokens)AutoModel sınıfına benzer olarak `AutoTokenizer` sınıfı ile herhangi bir checkpoint'e ait tokenizer'lar kullanılabilir.

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
```

Örnek bir kullanım şu şekilde olabilir:

```
tokenizer("Using a Transformer network is simple")
```

Çıktı:

```
{
 'input_ids': [101, 7993, 170, 11303, 1200, 2443, 1110, 3014, 102],
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1]
 }
```
Bir tokenizer'ı kaydetmek, yüklemek kadar kolaydır. Aşağıdaki gibi gerçekleştirilebilir:

```
tokenizer.save_pretrained("directory_on_my_computer")
```
Çıktıdaki `token_type_id` ilerleyen kısımlarda anlatılacaktır, benzer şekilde `attention_mask` değeri de birazdan anlatılacaktır. Ama her şeyden önce `input_ids` key'inin nasıl üretildiği incelenecektir. Bunun için tokenizer içindeki metodlara bakabiliriz.

Encoding

Kelimeleri sayılara çevirme işlemine **encoding** denir ve iki aşamda gerçekleşir. Bunlardan ilki tokenizasyondur ve bunu ID dönüşümü takip eder.

Birinci adım aşağıdaki gibi özetlenebilir:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

sequence = "Using a Transformer network is simple"
tokens = tokenizer.tokenize(sequence)
print(tokens)
```
Bunun sonucunda aşağıdaki çıktı elde edilir:

```
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```
Kullanılan bir subword tokenizer olduğu için, çıktıda `##er` gibi subword'ler de yer alır. Bu token'lar sonraki aşamada ID'lerine dönüştürüşür. Bu işlem aşağıdaki gibi gerçekleştirilebilir:

```
ids = tokenizer.convert_tokens_to_ids(tokens)
print(ids)
```
Bunun sonucunda aşağıdaki çıktı elde edilir:

```
[7993, 170, 11303, 1200, 2443, 1110, 3014]
```
Bu aşamadan sonra elde edilen ID'ler, modellerde kullanılabilir.

Decoding

Decode işlemi, yapılan işlemlerin tam tersine eştir. Bir örnekle açıklanabilir.

```
decoded_string = tokenizer.decode([7993, 170, 11303, 1200, 2443, 1110, 3014])
print(decoded_string)
```
Çıktı:

```
'Using a Transformer network is simple'
```
Bu fonksiyon ID'leri token'a çevirmek ile kalmaz, okunabilir bir cümle haline getirir. Bu özellik yeni metin oluşturma ya da tahmin etme işlerinde oldukça kullanışlıdır.