Tugas Course NLP

Chapter 2

Nama : muhammad makhlufi makbullah

Kelas : TK 45 01

NIM : 1103210171

Chapter 2 Using Transformer

Bab ini menjelaskan perbedaan utama 🤗 Transformers dengan pustaka ML lainnya, yaitu modelnya memiliki lapisan tersendiri yang membuatnya mudah dipahami dan dimodifikasi tanpa memengaruhi model lain.

Bab ini mencakup:

1. Contoh end-to-end menggunakan model dan tokenizer untuk mereplikasi fungsi pipeline() dari Bab 1.
2. Pembahasan API model: kelas model dan konfigurasi, cara memuat model, serta bagaimana model memproses input numerik menjadi prediksi.
3. API tokenizer: mengubah teks menjadi input numerik dan sebaliknya.
4. Cara mengolah batch kalimat sekaligus dan penjelasan fungsi tokenizer() tingkat tinggi.

Behind the pipeline (PyTorch)

Install the Transformers, Datasets, and Evaluate libraries to run this notebook.

```
[ ]  !pip install datasets evaluate transformers[sentencepiece]
```

```
[ ] from transformers import pipeline

    classifier = pipeline("sentiment-analysis")
    classifier(
        [
            "I've been waiting for a HuggingFace course my whole life.",
            "I hate this so much!",
        ]
    )
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437},
 {'label': 'NEGATIVE', 'score': 0.9994558095932007}]
```

```
[ ] from transformers import AutoTokenizer

    checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
    tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
[ ] raw_inputs = [
        "I've been waiting for a HuggingFace course my whole life.",
        "I hate this so much!",
    ]
    inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
    print(inputs)
```

```
{
    'input_ids': tensor([
        [  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172, 2607,  2026,  2878,  2166,  1012,   102],
        [  101,  1045,  5223,  2023,  2061,  2172,   999,   102,     0,     0,     0,     0,     0,     0,     0,     0]
    ]),
    'attention_mask': tensor([
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

```
[ ] from transformers import AutoModel

    checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
    model = AutoModel.from_pretrained(checkpoint)
```

```
[ ] outputs = model(**inputs)
    print(outputs.last_hidden_state.shape)
```

```
torch.Size([2, 16, 768])
```

```
[ ] from transformers import AutoModelForSequenceClassification

    checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
    model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
    outputs = model(**inputs)
```

```
[ ] print(outputs.logits.shape)
```

```
torch.Size([2, 2])
```

```
[ ] print(outputs.logits)
```

```
tensor([[-1.5607,  1.6123],
        [ 4.1692, -3.3464]], grad_fn=<AddmmBackward>)
```

```
[ ] import torch

    predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
    print(predictions)
```

```
tensor([[4.0195e-02, 9.5980e-01],
        [9.9946e-01, 5.4418e-04]], grad_fn=<SoftmaxBackward>)
```

```
[ ] model.config.id2label
```

```
{0: 'NEGATIVE', 1: 'POSITIVE'}
```

Models (PyTorch)

Membuat konfigurasi dan model secara manual:

- BertConfig() membuat konfigurasi default untuk model BERT.
- BertModel(config) membangun model BERT baru dengan parameter acak berdasarkan konfigurasi tersebut.

Menggunakan model pre-trained:

- BertModel.from_pretrained("bert-base-cased") memuat model BERT yang telah dilatih sebelumnya (pre-trained) dengan nama "bert-base-cased".
- model.save_pretrained("directory_on_my_computer") menyimpan model ke direktori lokal.

Mengolah input:

- sequences adalah teks yang akan diproses. encoded_sequences adalah token ID (hasil tokenisasi).
- torch.tensor(encoded_sequences) mengubah token ID menjadi tensor PyTorch yang siap diproses oleh model.
- model(model_inputs) menjalankan tensor melalui model untuk menghasilkan output.

Tokenizer

1. Tokenisasi Manual:

- tokenized_text = "Jim Henson was a puppeteer".split(): Memecah teks menjadi kata-kata berdasarkan spasi.
- Hasil: ['Jim', 'Henson', 'was', 'a', 'puppeteer'].

2. Memuat Tokenizer BERT:

- BertTokenizer.from_pretrained("bert-base-cased") atau AutoTokenizer.from_pretrained("bert-base-cased"): Memuat tokenizer pre-trained untuk model bert-base-cased.

3. Tokenisasi Otomatis:

- tokenizer("Using a Transformer network is simple"): Mengubah teks menjadi token ID dalam satu langkah.

4. Tokenisasi Langkah per Langkah:

- tokenizer.tokenize(sequence): Memecah teks menjadi token dengan mempertimbangkan aturan model, seperti sub-tokenisasi menggunakan ##. Contoh hasil: ['Using', 'a', 'Transformer', 'network', 'is', 'simple'].

- tokenizer.convert_tokens_to_ids(tokens): Mengubah token menjadi token ID yang dapat diproses model.

5. Dekode Token ID ke Teks:

- tokenizer.decode([7993, 170, 11303, 1200, 2443, 1110, 3014]): Mengubah kembali token ID menjadi teks asli.
- Hasil: "Using a Transformer network is simple".

6. Menyimpan Tokenizer:

- tokenizer.save_pretrained("directory_on_my_computer"): Menyimpan tokenizer ke direktori lokal untuk digunakan nanti.

Handling Multiple Sequence

1. Memuat Tokenizer dan Model:

- AutoTokenizer.from_pretrained(checkpoint): Memuat tokenizer yang sesuai dengan checkpoint model.
- AutoModelForSequenceClassification.from_pretrained(checkpoint): Memuat model klasifikasi sentimen yang telah dilatih sebelumnya.

2. Tokenisasi dan Konversi ID:

- tokenizer.tokenize(sequence): Mengubah teks menjadi token.
- tokenizer.convert_tokens_to_ids(tokens): Mengubah token menjadi token ID.
- Perhatian: Tensor input harus berbentuk batch (contoh: [[ids]]), sehingga torch.tensor(ids) tanpa pembungkusan akan gagal.

3. Pengolahan Input:

Padding dan Masking:

- Jika input memiliki panjang berbeda, gunakan token padding (tokenizer.pad_token_id) untuk menyamakan panjang.
- attention_mask digunakan untuk memberi tahu model bagian mana dari input yang penting (1) dan mana yang padding (0).

4. Prediksi Model:

Model menerima:

- input_ids: Token ID teks.
- attention_mask: Masking untuk input.
- Output model adalah logits (skor untuk setiap kelas).

5. Contoh Pemrosesan Batch:

Batched Input dengan Padding:

- Menambahkan token padding pada input yang lebih pendek.
- Menggunakan attention_mask untuk menyesuaikan dengan padding.
- outputs.logits memberikan prediksi untuk setiap input dalam batch.

Putting it all together (PyTorch)

Menginstal dan memuat tokenizer:

- AutoTokenizer.from_pretrained(checkpoint) memuat tokenizer yang sesuai dengan checkpoint model pre-trained.

Tokenisasi teks:

- tokenizer(sequence) mengonversi teks menjadi token ID (angka) yang dapat diproses oleh model.

Berbagai opsi disediakan, seperti:

- Padding: Menyesuaikan panjang input (contoh: padding="longest", padding="max_length").
- Truncation: Memotong teks jika terlalu panjang (contoh: truncation=True, max_length=8).
- Opsi return_tensors menghasilkan output dalam format tensor PyTorch ("pt"), TensorFlow ("tf"), atau NumPy ("np").

Konversi token ke ID dan sebaliknya:

- tokenizer.tokenize(sequence) mengubah teks menjadi token.
- tokenizer.convert_tokens_to_ids(tokens) mengubah token menjadi token ID.
- tokenizer.decode(ids) mengubah kembali token ID ke teks.

Model dan prediksi:

- AutoModelForSequenceClassification.from_pretrained(checkpoint) memuat model klasifikasi sentimen yang sesuai dengan checkpoint.
- Input token ID (tokens) dimasukkan ke model menggunakan model(**tokens), dan model menghasilkan output berupa skor klasifikasi untuk setiap label (sentimen positif atau negatif).

Berikut rangkuman dari pembelajaran coding yang telah kita bahas sebelumnya, berdasarkan pujian di atas:

1. Komponen Dasar Model Transformer:

- Model Transformer, seperti BERT dan DistilBERT, dibangun dengan arsitektur modular yang memungkinkan tokenisasi, konversi teks ke tensor, dan prediksi.

2. Pipeline Tokenisasi:

- Tokenisasi adalah proses memecah teks menjadi unit-unit kecil yang disebut token.
- Token ini dikonversi menjadi token ID numerik, yang dapat dipahami oleh model.
- Fungsi tokenisasi seperti:
- tokenizer.tokenize(): Mengubah teks menjadi token.
- tokenizer.convert_tokens_to_ids(): Mengubah token menjadi ID numerik.
- tokenizer.decode(): Mengubah ID kembali menjadi teks.

3. Menggunakan Model Transformer:

- AutoTokenizer dan AutoModel memudahkan integrasi antara tokenizer dan model pre-trained.
- Proses:
- Muat tokenizer dan model: AutoTokenizer.from_pretrained() dan AutoModel.from_pretrained().
- Tokenisasi teks dengan padding dan truncation untuk menyesuaikan panjang input.
- Konversi hasil tokenisasi menjadi tensor PyTorch atau TensorFlow untuk dimasukkan ke model.

4. Mengolah Input dan Membuat Prediksi:

- Input IDs: Sekuen token ID yang merepresentasikan teks.
- Attention Mask: Digunakan untuk membedakan token teks dan padding, memastikan model hanya memproses bagian teks yang penting.

Contoh:

tokenizer(sequences, padding=True, truncation=True, return_tensors="pt") menghasilkan tensor dengan padding dan truncation.

Output model (output.logits) memberikan hasil prediksi, seperti klasifikasi sentimen.

5. Fitur Tokenizer yang Fleksibel:

- Tokenizer dapat disesuaikan dengan berbagai parameter, seperti:
- Padding: Menyesuaikan panjang teks dalam batch.
- Truncation: Memotong teks yang terlalu panjang.
- max_length: Membatasi panjang input.