Huggingface——最火热的自然语言处理框架

马英梓 (2019141060171)

2022 年 5 月 24 日

1 背景

目前各种神经网络的预训练模型的规模都已经达到了常人难以想象的地步，从 20 年的 GPT-3[] 开始，预训练模型一举扩大到千亿参数的规模，取得了很多非常令人震惊的效果，这标志着 NLP 领域，自监督的大规模预训练模型成为潮流。当然，不只在 NLP 领域，在 CV 方面，去年 Kaiming 大神提出的类似于 Bert[] 的 MAE 网络结构 []，将大规模自监督学习引入到 CV 领域，在 Imagenet 上取得了惊人的效果。以上的模型都预示着，未来的方向是属于基于大数据的大模型，一个千亿参数的大模型光训练就要耗费千万美元，而这些大模型所需要耗费的时间和金钱都是个人无法负担的。因此，将这些先进的模型平民化就显得尤为重要。于是，我想介绍 Huggingface 这个非常火热的 NLP 框架。

2 介绍

Huggingface总部位于纽约，是一家专注于自然语言处理、人工智能和分布式系统的创业公司。他们所提供的聊天机器人技术一直颇受欢迎，但更出名的是他们在 NLP 开源社区上的贡献。Huggingface 一直致力于自然语言处理 NLP 技术的平民化 (democratize)，希望每个人都能用上最先进 (SOTA, state-of-the-art) 的 NLP 技术，而非困窘于训练资源的匮乏。同时 Hugging Face 专注于 NLP 技术，拥有大型的开源社区。尤其是在 github 上开源的自然语言处理，预训练模型库 Transformers，已被下载超过一百万次，github 上超过 24000 个 star。Transformers 提供了 NLP 领域大量 state-of-art 的预训练语言模型结构的模型和调用框架。

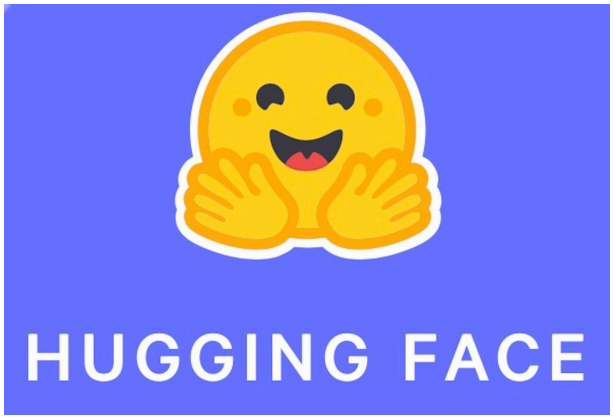

图 1: The icon of Huggingface

3 使用步骤

3.1 框架特点

Huggingface 可以帮助我们仅仅只用几行的代码就可以实现对很多 SOTA 的 NLP 模型的使用。同时，它是基于主要的深度学习框架实现的，如 PyTorch 的 nn.Module 和 TensorFlow 的 tf.keras.Model，从而使其对许多深度学习开发环境具有灵活性。你可以自由选择你熟悉的神经网络框架配合 Huggingface 来搭建自己的 NLP 模型。

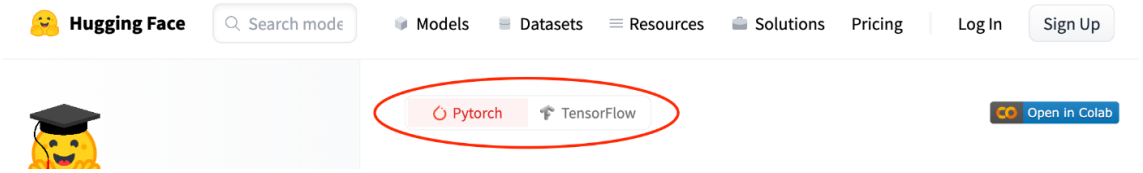

图 2: Huggingface based on DL frameworks

其中包含了各种任务的代表模型：

- Encoder：ALBERT, BERT, DistilBERT, ELECTRA,RoBERTa，这些都是 Bert 系列的模型，具有双向建模的能力，适合需要理解完整句子的任务，例如句子分类、命名实体识别（以及更一般的单词分类）和提取式问答
- Decoder：CTRL, GPT, GPT-2, Transformer XL，代表为 GPT 系列，解码器模型的预训练通常围绕预测句子中的下一个单词，这些模型最适合涉及文本生成的任务。
- Encoder-decoder：BART, T5, Marian, mBART，序列到序列模型最适合围绕根据给定输入生成新句子的任务，例如摘要、翻译或生成式问答。

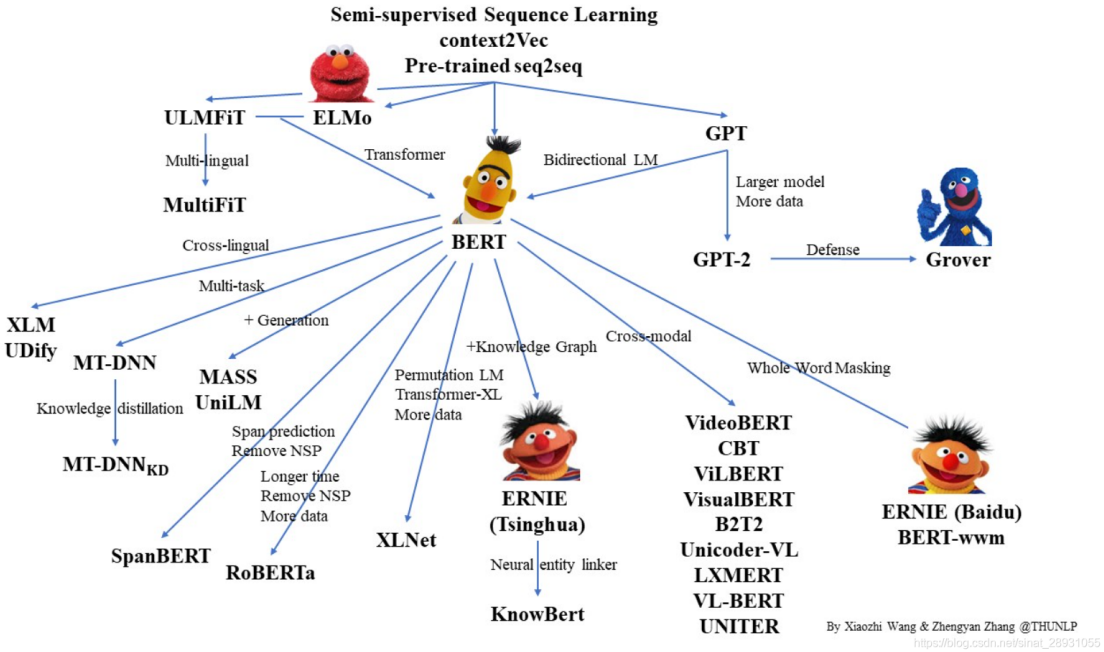

图 3: Bert[]

3.2 Pipline

该 API 直接集成了一个完整 NLP 模型所需要的主要三个步骤：Tokenizer、Model、Post-Processing. 使得我们可以直接输入文本，而输出我们想要的下游任务，比如文本相似度分析，文本情感分析等等。

```
from transformers import pipeline

clf = pipeline('sentiment-analysis')
clf('Haha, today is a nice day!')

输出：
[{'label': 'POSITIVE', 'score': 0.9998709559440613}]
```

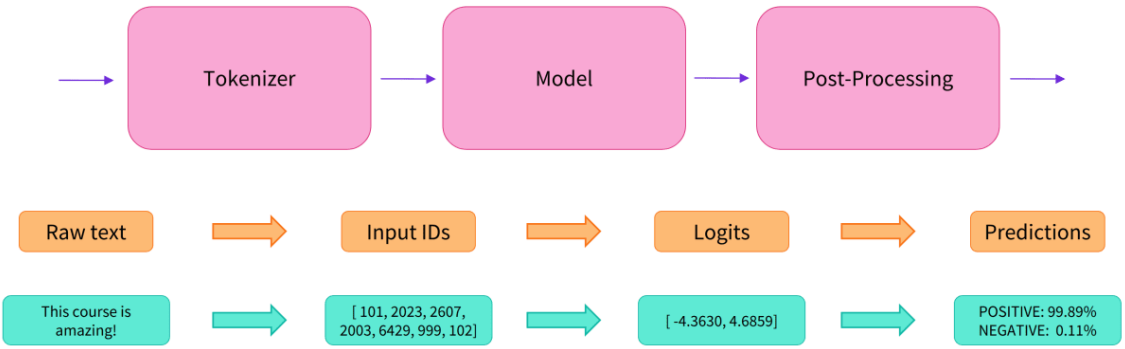

图 4: process of Pipline

当然，在做实际任务时，我们往往会把这三部分分开来实现，方便更好地处理任务和搭建模型。

3.3 Tokenizer

tokenizer 是做自然语言处理首要的步骤，目的是根据某个或者某些特定的语料库，对每个词进行 vocab 编号，最终映射到一个词表。根据每个词在词表中的索引，我们可以对该词进行编码，one-hot 就是最简单的一种（1——>[100], 2——>[010]...）.Huggingface 中的 transformer 模块提供了 AutoTokenizer，可以自动搜索 Tokenizer 模型，提供了极大的便利。这里我们为了直观就直接使用 Bert 中的 token 映射，得到 vocab$_i dx$

```
from transformers import BertTokenizer

checkpoint = "hflchinese-bert-wwm-ext"
tokenizer = BertTokenizer.from_pretrained(checkpoint, mirror='tuna')

raw_inputs = [
    "生殖系统无器质性病变，术前病理检查无子宫内膜病变者",
    "手术体位不适合BIS 监测",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)

输出：
```

```
{
    '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, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    ])
}
```

其中 input_ids 就是词在词表中的索引，attention_mask 是对不同长度的句子进行填充的掩码，确保每个句子长度一致。

3.4 Model

该模块和 tokenizer 类似，也可以通过 AutoModel 自动搜索各种 SOTA 模型。这里导入的模型，是 Transformer 的基础模型，接受 tokenize 之后的输入，输出 hidden states，即文本的向量表示，是一种上下文表示。

这个向量表示，会有三个维度:

- batch size：batch 中样本数量
- sequence length：每个 text 的长度，bert 中最大为 512
- hidden size：每次词投影到的向量维度，bert 中一般为 768

```
from transformers import BertTokenizer, BertModel

model = BertModel.from_pretrained(checkpoint)

inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors='pt')
outputs = model(**inputs)
vars(outputs).keys()  # 查看一下输出有哪些属性

输出:
dict_keys(['last_hidden_state', 'hidden_states', 'attentions'])
```

至此，使用了 bert 语言模型完成了词向量编码。每个词都表示为一个向量，这些向量也可以看作特征 keys，而文本就相当于一个 dictionary，利用这些 keys，以及他们的关系，我们可以用来做很多有趣的下游任务。

比如，我们可以在 last_hidden_state 后面加入一个 MLP head，可以用来做句子分类，MLM，句子预测等等任务。而 fine-tune 主要就是在调整这个 head，使它能根据 bert 编码得到的 tokens 特征，学习得到我们想要的任务。

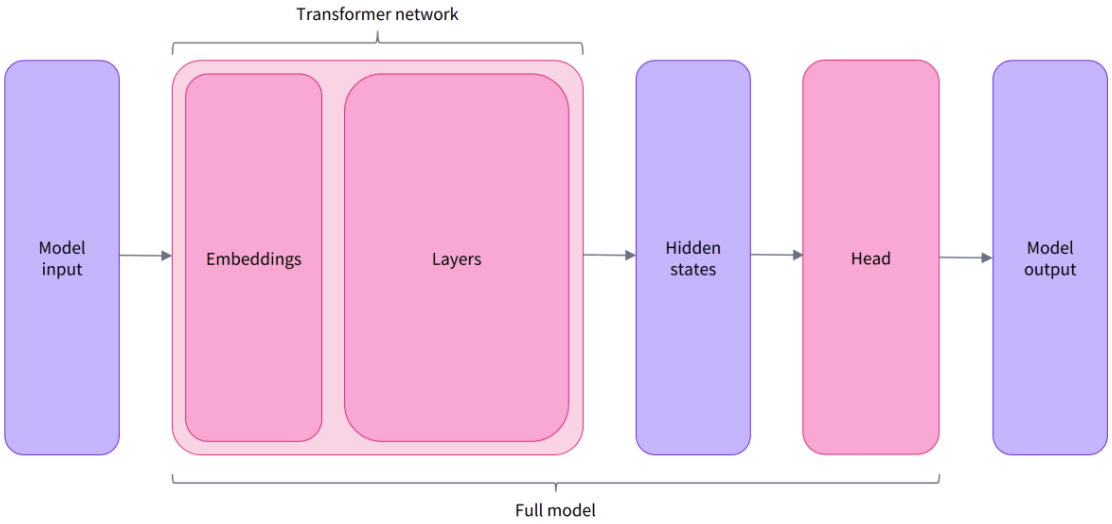

图 5: Overall framework diagram

3.5 Post-Processing

这部分就很简单了，在句子分类任务中，主要就是两步：

- 把 logits 转化成概率值（用 softmax）
- 把概率值跟具体的标签对应上（使用模型的 config 中的 id2label）

这些都是我们在普通训练中常做的操作，因此不再赘述。到此，模型基本搭建完成，可以开始做 fine-tune，或者做一些预训练任务，来时模型在我们的数据集上 work。

4 任务实战

4.1 数据集介绍

4.2 模型搭建

4.3 结果展示

参考文献