手把手 带你实战 Transformers

基础入门篇

Transformers基础使用指南

你可是处女座啊

目录

01 基础知识与环境安装

（1）自然语言处理任务

（2）Transformers介绍

（3）Transformers相关环境安装

（4）两行代码的QA实例

常见自然语言处理任务

情感分析（sentiment-analysis）：对给定的文本分析其情感极性

文本生成（text-generation）：根据给定的文本进行生成

命名实体识别（ner）：标记句子中的实体

阅读理解（question-answering）：给定上下文与问题，从上下文中抽取答案

掩码填充（fill-mask）：填充给定文本中的掩码词

文本摘要（summarization）：生成一段长文本的摘要

机器翻译（translation）：将文本翻译成另一种语言

特征提取（feature-extraction）：生成给定文本的张量表示

对话机器人（conversional）：根据用户输入文本，产生回应，与用户对话

自然语言处理的几个阶段

- 第一阶段：统计模型 + 数据（特征工程）
  - 决策树、SVM、HMM、CRF、TF-IDF、BOW
- 第二阶段：神经网络 + 数据
  - Linear、CNN、RNN、GRU、LSTM、Transformer、Word2vec、Glove
- 第三阶段：神经网络 + 预训练模型 + （少量）数据
  - GPT、BERT、RoBERTa、ALBERT、BART、T5
- 第四阶段：神经网络 + 更大的预训练模型 + Prompt
  - ChatGPT、Bloom、LLaMA、Alpaca、Vicuna、MOSS、文心一言、通义千问、星火

Transformers简单介绍

- 官方网址：https://huggingface.co/
- HuggingFace出品，当下最热、最常使用的自然语言处理工具包之一，不夸张的说甚至没有之一
- 实现了大量的基于Transformer架构的主流预训练模型，不局限于自然语言处理模型，还包括图像、音频以及多模态的模型
- 提供了海量的预训练模型与数据集，同时支持用户自行传，社区完善，文档全面，三两行代码便可快速实现模型训练推理，上手简单

一句话总结：学就对了！

Transformers及相关库

- Transformers：核心库，模型加载、模型训练、流水线等
- Tokenizer：分词器，对数据进行预处理，文本到token序列的互相转换
- Datasets：数据集库，提供了数据集的加载、处理等方法
- Evaluate：评估函数，提供各种评价指标的计算函数
- PEFT：高效微调模型的库，提供了几种高效微调的方法，小参数量撬动大模型
- Accelerate：分布式训练，提供了分布式训练解决方案，包括大模型的加载与推理解决方案
- Optimum：优化加速库，支持多种后端，如Onnxruntime、OpenVino等
- Gradio：可视化部署库，几行代码快速实现基于Web交互的算法演示系统

Transformers环境安装

前置环境安装——python

- **miniconda 安装**
  - 下载地址：https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/
  - 如果C盘有空间，最好安装在C盘，且安装目录中不能有中文
  - 勾选将其添加到PATH
- **conda环境创建**
  - 命令：conda create -n transformers python=3.9
  - 明确指定版本，否则可能会因版本过高导致有包装不上
- **pypi配置国内源**
  - 清华源：https://mirrors.tuna.tsinghua.edu.cn/help/pypi/

Transformers环境安装

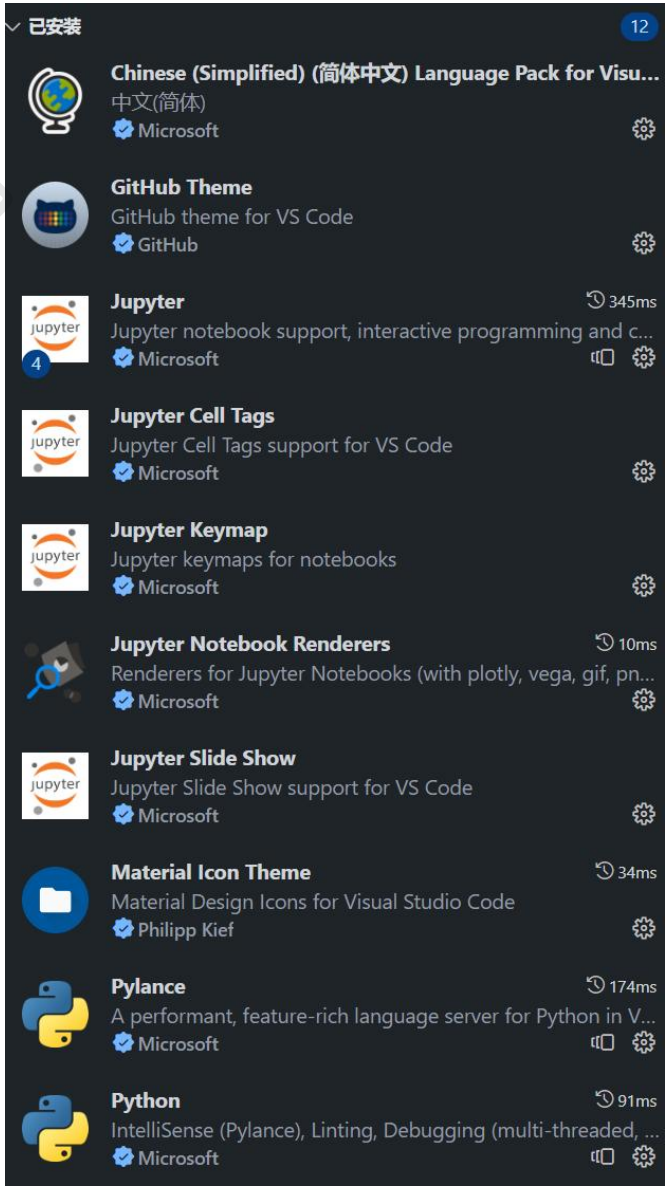
前置环境安装—pytorch

- **pytorch安装**
  - 官方地址：https://pytorch.org/
  - 在一个单独的环境中，能使用pip就尽量使用pip，实在有问题的情况，例如没有合适的编译好的系统版本的安装包，再使用conda进行安装，不要来回混淆
  - 30XX、40XX显卡，要安装cu11以上的版本，否则无法运行
- **CUDA是否要安装**
  - 如果只需要训练、简单推理，则无需单独安装CUDA，直接安装pytorch
  - 如果有部署需求，例如导出TensorRT模型，则需要进行CUDA安装

Transformers环境安装

前置环境安装—vscode

- **VS Code 安装**
  - 官方地址：https://code.visualstudio.com/download
- **插件安装**
  - Python (代码编写)
  - remote ssh (连接服务器)
  - Chinese Language Pack（简体中文包)
- **终端设置（非常重要！非常重要！非常重要！）**
  - 选择默认配置文件: cmd.exe



Transformers环境安装

Transformers安装

- **安装命令**
  - pip install transformers datasets evaluate peft accelerate gradio optimum sentencepiece
  - pip install jupyterlab scikit-learn pandas matplotlib tensorboard nltk rouge
- **hosts修改**
  - 185.199.108.133 raw.githubusercontent.com
  - 185.199.109.133 raw.githubusercontent.com
  - 185.199.110.133 raw.githubusercontent.com
  - 185.199.111.133 raw.githubusercontent.com
  - 2606:50c0:8000::154 raw.githubusercontent.com
  - 2606:50c0:8001::154 raw.githubusercontent.com
  - 2606:50c0:8002::154 raw.githubusercontent.com
  - 2606:50c0:8003::154 raw.githubusercontent.com

Transformers极简实例

三行代码，启动NLP应用

样例1：文本分类

```
# 导入gradio
import gradio as gr
# 导入transformers相关包
from transformers import *
# 通过Interface加载pipeline并启动文本分类服务
gr.Interface.from_pipeline(pipeline("text-classification", model="uer/roberta-base-finetuned-dianping-chinese")).launch()
```

样例2：阅读理解

```
# 导入gradio
import gradio as gr
# 导入transformers相关包
from transformers import *
# 通过Interface加载pipeline并启动阅读理解服务
gr.Interface.from_pipeline(pipeline("question-answering", model="uer/roberta-base-chinese-extractive-qa")).launch()
```

02

基础组件之Pipeline

（1）什么是Pipeline

（2）Pipeline支持的任务类型

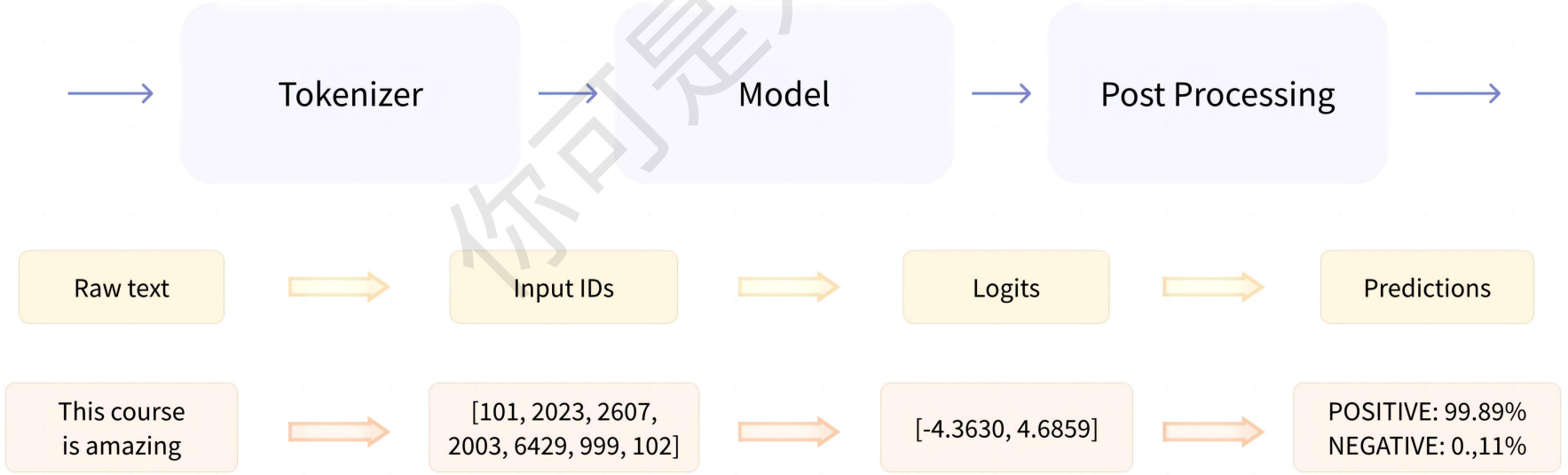
（3）Pipeline的创建与使用方式

（4）Pipeline的背后实现

基础组件之Pipeline

什么是Pipeline

- **Pipeline**
  - 将数据预处理、模型调用、结果后处理三部分组装成的流水线
  - 使我们能够直接输入文本便获得最终的答案



基础组件之Pipeline

Pipeline支持的任务类型

名称	任务类型
text-classification（sentiment-analysis）	text
token-classification（ner）	text
question-answering	text
fill-mask	text
summarization	text
translation	text
text2text-generation	text
text-generation	text
conversational	text
table-question-answering	text
zero-shot-classification	text
automatic-speech-recognition	multimodal
feature-extraction	multimodal
audio-classification	audio
visual-question-answering	multimodal
document-question-answering	multimodal
zero-shot-image-classification	multimodal
zero-shot-audio-classification	multimodal
image-classification	image
zero-shot-object-detection	multimodal
video-classification	video

基础组件之Pipeline

Pipeline创建与使用

- **根据任务类型直接创建Pipeline**
  - pipe = pipeline("text-classification")
- **指定任务类型，再指定模型，创建基于指定模型的Pipeline**
  - pipe = pipeline("text-classification", model="uer/roberta-base-finetuned-dianping-chinese")
- **预先加载模型，再创建Pipeline**
  - model = AutoModelForSequenceClassification.from_pretrained("uer/roberta-base-finetuned-dianping-chinese")
  - tokenizer = AutoTokenizer.from_pretrained("uer/roberta-base-finetuned-dianping-chinese")
  - pipe = pipeline("text-classification", model=model, tokenizer=tokenizer)
- **使用GPU进行推理加速**
  - pipe = pipeline("text-classification", model="uer/roberta-base-finetuned-dianping-chinese", device=0)

基础组件之Pipeline

Pipeline的背后实现

- **Step1 初始化Tokenizer**
  - tokenizer = AutoTokenizer.from_pretrained("uer/roberta-base-finetuned-dianping-chinese")
- **Step2 初始化Model**
  - model = AutoModelForSequenceClassification.from_pretrained("uer/roberta-base-finetuned-dianping-chinese")
- **Step3 数据预处理**
  - input_text = "我觉得不太行！"
  - inputs = tokenizer(input_text, return_tensors="pt")
- **Step4 模型预测**
  - res = model(**inputs).logits
- **Step5 结果后处理**
  - pred = torch.argmax(torch.softmax(logits, dim=-1)).item()
  - result = model.config.id2label.get(pred)

03

基础组件之Tokenizer

（1）Tokenizer简介

（2）Tokenizer基本使用方法

（3）Fast / Slow Tokenizer

基础组件之Tokenizer

Tokenizer 简介

- **数据预处理**
  - **Step1 分词**：使用分词器对文本数据进行分词（字、字词）；
  - **Step2 构建词典**：根据数据集分词的结果，构建词典映射（这一步并不绝对，如果采用预训练词向量，词典映射要根据词向量文件进行处理）；
  - **Step3 数据转换**：根据构建好的词典，将分词处理后的数据做映射，将文本序列转换为数字序列；
  - **Step4 数据填充与截断**：在以batch输入到模型的方式中，需要对过短的数据进行填充，过长的数据进行截断，保证数据长度符合模型能接受的范围，同时batch内的数据维度大小一致。

现在：Tokenizer is all you need!

基础组件之Tokenizer

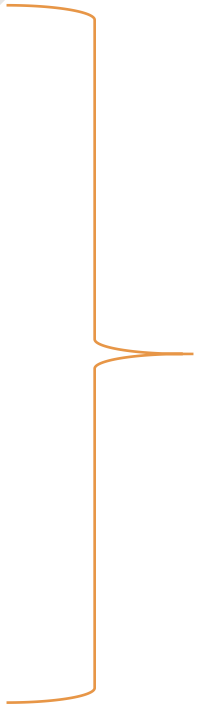
Tokenizer 基本使用

- 加载保存（from_pretrained / save_pretrained）
- 句子分词（tokenize）
- 查看词典（vocab）
- 索引转换（convert_tokens_to_ids / convert_ids_to_tokens）
- 填充截断（padding / truncation）
- 其他输入（attention_mask / token_type_ids）

基础组件之Tokenizer

Tokenizer 基本使用

- 加载保存（from_pretrained / save_pretrained）
- 句子分词（tokenize）
- 查看词典（vocab）
- 索引转换（convert_tokens_to_ids / convert_ids_to_tokens）
- 填充截断（padding / truncation）
- 其他输入（attention_mask / token_type_ids）



`tokenizer(inputs)`

基础组件之Tokenizer

Fast / Slow Tokenizer

- **FastTokenizer**
  - 基于Rust实现，速度快
  - offsets_mapping、word_ids
- **SlowTokenizer**
  - 基于Python实现，速度慢

```
%%time
# 单条循环处理
for i in range(10000):
    fast_tokenizer(sen)
```

✓ 0.3s

```
CPU times: total: 78.1 ms
Wall time: 312 ms
```

```
%%time
# 处理batch数据
res = fast_tokenizer([sen] * 10000)
```

✓ 0.1s

```
CPU times: total: 359 ms
Wall time: 87.1 ms
```

```
%%time
# 单条循环处理
for i in range(10000):
    slow_tokenizer(sen)
```

✓ 0.8s

```
CPU times: total: 281 ms
Wall time: 872 ms
```

```
%%time
# 处理batch数据
res = slow_tokenizer([sen] * 10000)
```

✓ 0.7s

```
CPU times: total: 188 ms
Wall time: 705 ms
```

04 基础组件之Model

（1）Model 简介

（2）Model Head

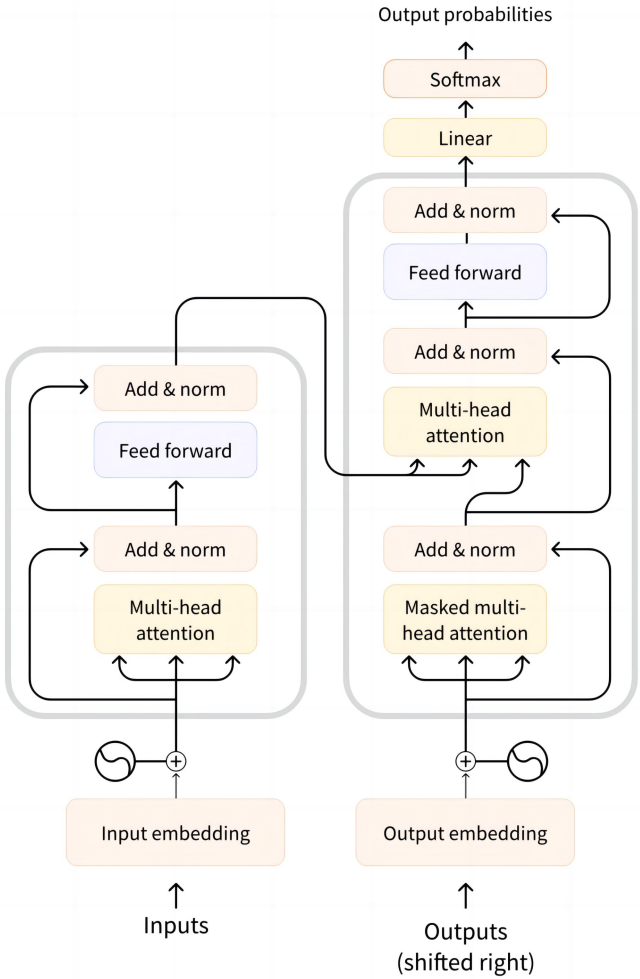
（3）Model 基本使用方法

（4）模型微调代码实例

基础组件之Model

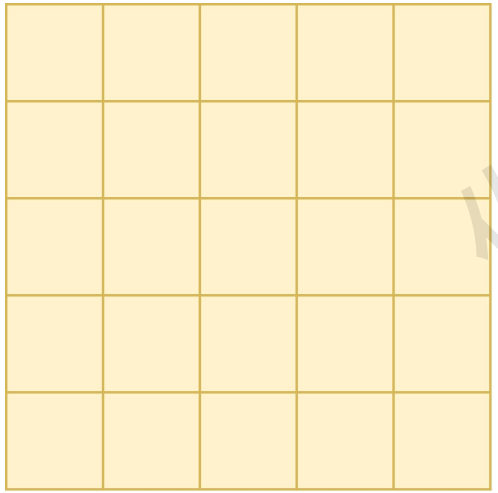
Model简介

- **Transformer**
  - 原始的Transformer为编码器（Encoder）、解码器（Decoder）模型
  - Encoder部分接收输入并构建其完整特征表示，Decoder部分使用Encoder的编码结果以及其他的输入生成目标序列
  - 无论是编码器还是解码器，均由多个TransformerBlock堆叠而成
  - TransformerBlock由注意力机制（Attention）和FFN组成
- **注意力机制**
  - 注意力机制的使用是Transformer的一个核心特性，在计算当前词的特征表示时，可以通过注意力机制有选择性的告诉模型要使用哪些上下文

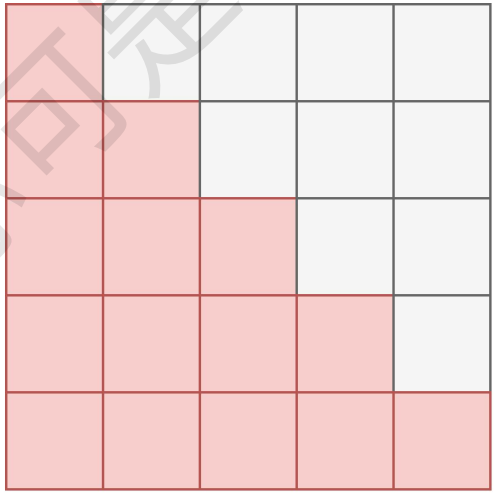


Model简介

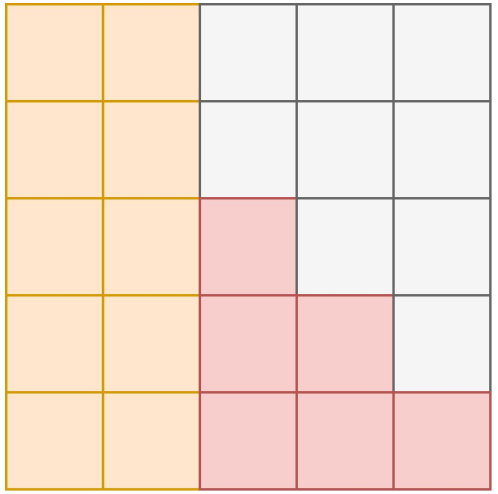
- **模型类型**
  - 编码器模型：自编码模型，使用Encoder，拥有双向的注意力机制，即计算每一个词的特征时都看到完整上下文
  - 解码器模型：自回归模型，使用Decoder，拥有单向的注意力机制，即计算每一个词的特征时都只能看到上文，无法看到下文
  - 编码器解码器模型：序列到序列模型，使用Encoder+Decoder，Encoder部分使用双向的注意力，Decoder部分使用单向注意力



编码器模型



解码器模型



编码器解码器模型

Model简介

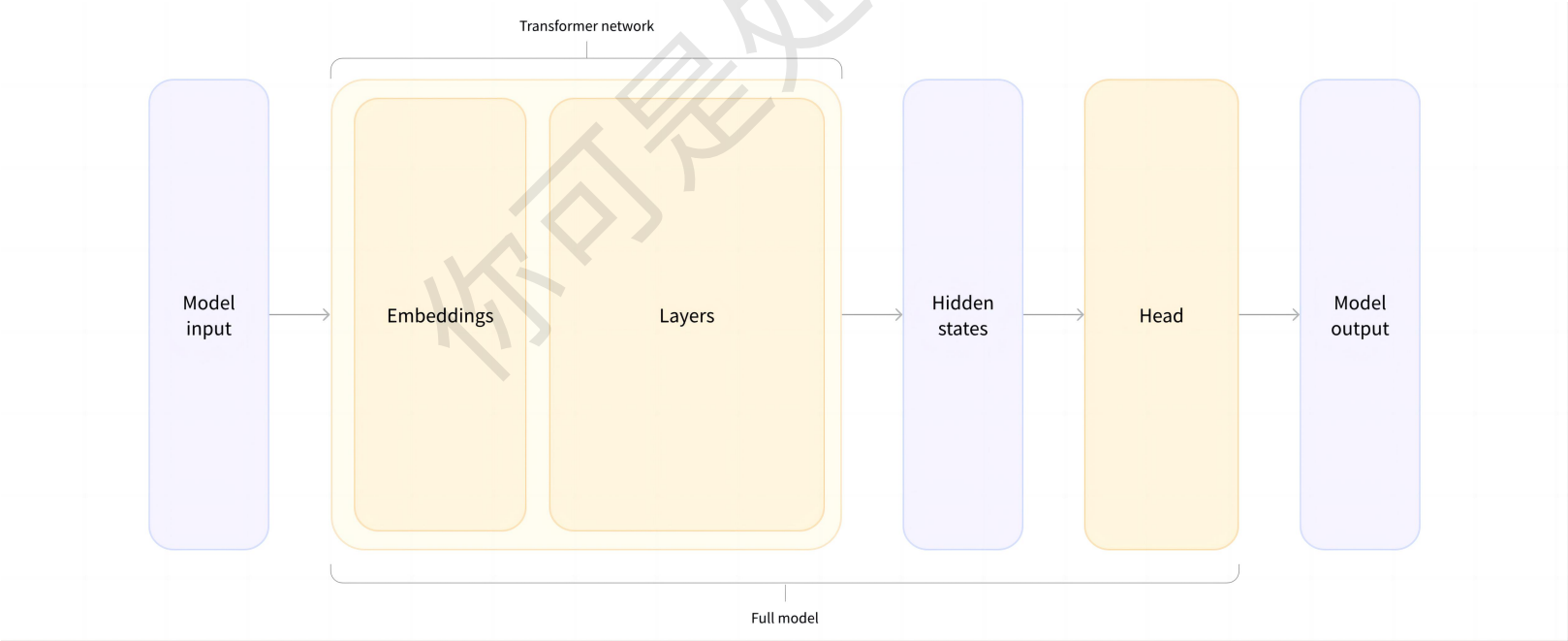
- **模型类型**
  - 编码器模型：自编码模型，使用Encoder，拥有双向的注意力机制，即计算每一个词的特征时都看到完整上下文
  - 解码器模型：自回归模型，使用Decoder，拥有单向的注意力机制，即计算每一个词的特征时都只能看到上文，无法看到下文
  - 编码器解码器模型：序列到序列模型，使用Encoder+Decoder，Encoder部分使用双向的注意力，Decoder部分使用单向注意力

模型类型	常用预训练模型	适用任务
编码器模型，自编码模型	ALBERT, BERT, DistilBERT, RoBERTa	文本分类、命名实体识别、阅读理解
解码器模型，自回归模型	GPT, GPT-2, Bloom, LLaMA	文本生成
编码器解码器模型，序列到序列模型	BART, T5, Marian, mBART，GLM	文本摘要、机器翻译

基础组件之Model

Model Head

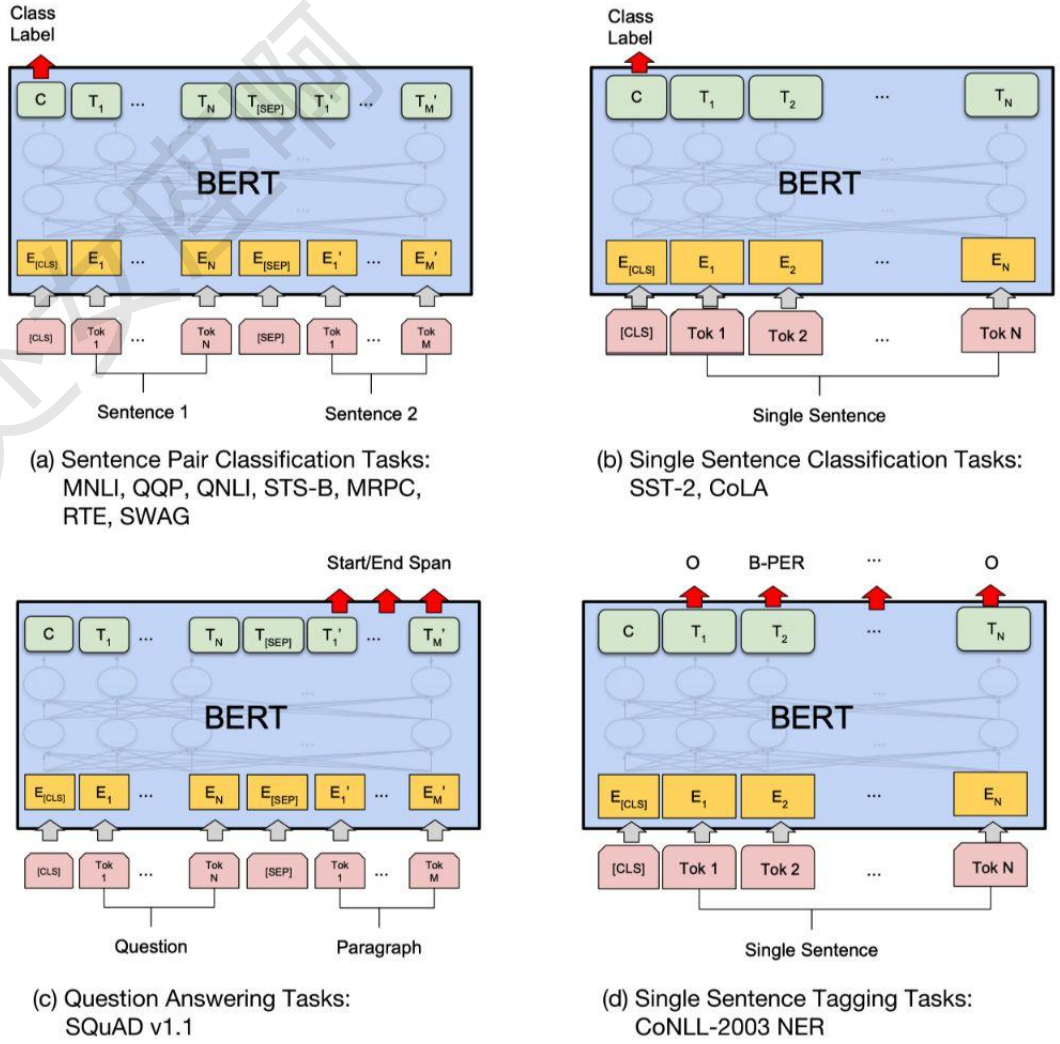
- **什么是Model Head**
  - Model Head 是连接在模型后的层，通常为1个或多个全连接层
  - Model Head 将模型的编码的表示结果进行映射，以解决不同类型的任务



基础组件之Model

Model Head

- **Transformers中的Model Head**
  - *Model（模型本身，只返回编码结果）
  - *ForCausalLM
  - *ForMaskedLM
  - *ForSeq2SeqLM
  - *ForMultipleChoice
  - *ForQuestionAnswering
  - *ForSequenceClassification
  - *ForTokenClassification
  - ... ...



基础组件之Model

Model基本使用方法

- **模型加载与保存**
  - 在线加载
  - 模型下载
  - 离线加载
  - 模型加载参数
- **模型调用**
  - 不带model head的模型调用
  - 带model head的模型调用

基础组件之Model

模型微调代码实例

- **任务类型**
  - 文本分类
- **使用模型**
  - hfl/rbt3
- **数据集地址**
  - https://github.com/SophonPlus/ChineseNlpCorpus

```
model.eval()
sen = "我觉得这家店的味道还是不错!"
with torch.inference_mode():
    inputs = tokenizer(sen, return_tensors="pt")
    batch = {k: v.cuda() for k, v in inputs.items()}
    logits = model(**batch).logits
    pred = torch.argmax(logits, dim=-1)
    print(f"评论: {sen}\n模型预测结果: {model.config.id2label.get(pred.item())}")
```

```
评论: 我觉得这家店的味道还是不错!
模型预测结果: 好评!
```

```
from transformers import pipeline

pipe = pipeline("text-classification", model=model, tokenizer=tokenizer, device=0)
```

```
pipe(sen)
```

```
[{'label': '好评! ', 'score': 0.9716703295707703}]
```

05

基础组件之Datasets

（1）Datasets简介

（2）Datasets基本使用

（3）Datasets加载本地数据集

（4）Datasets + DataCollator模型微调代码优化

基础组件之Datasets

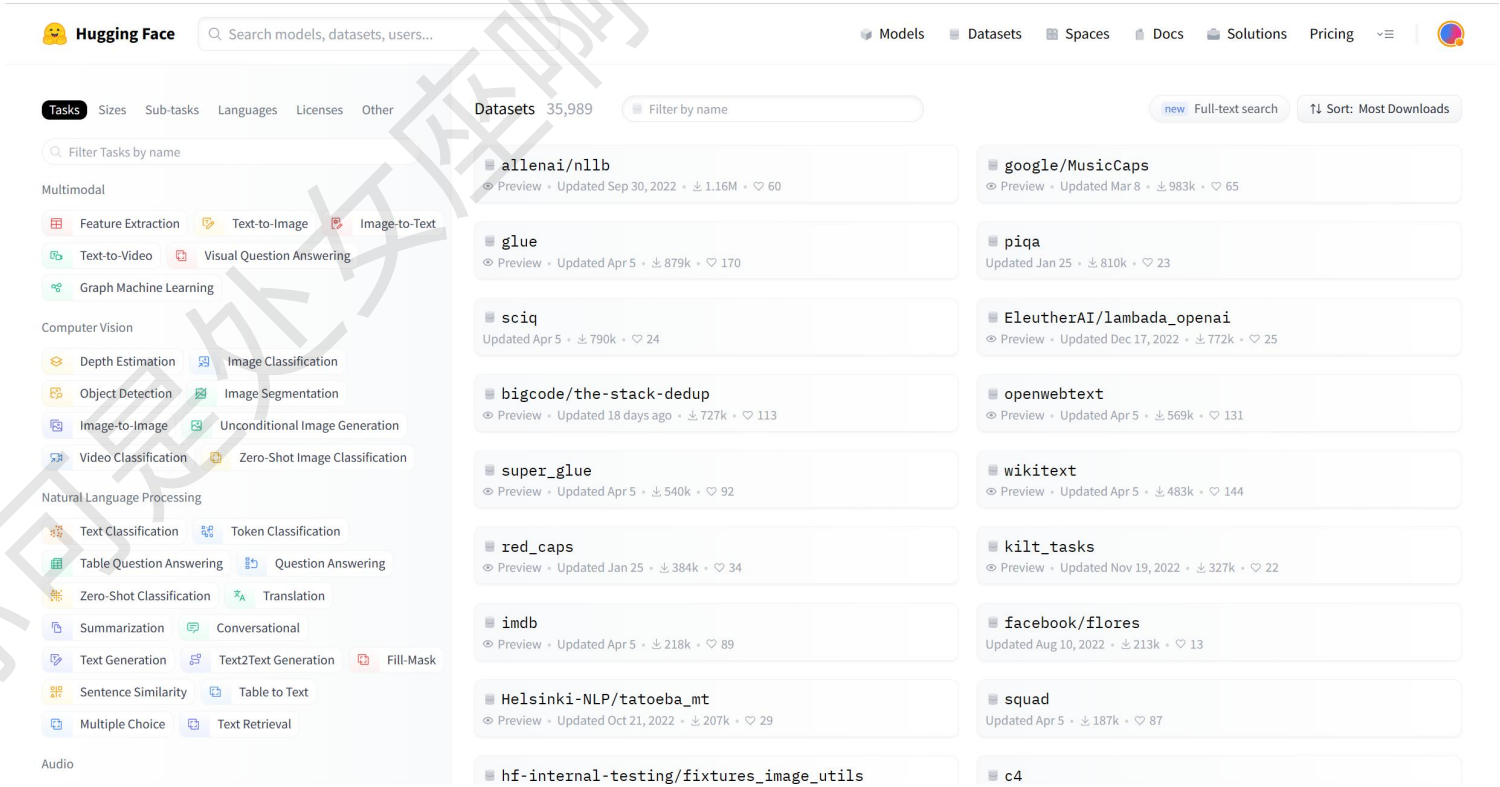
Datasets

- **简介**
  - datasets库是一个非常简单易用的数据集加载库，可以方便快捷的从本地或者HuggingFace Hub加载数据集
- **公开数据集地址**
  - https://huggingface.co/datasets
- **文档地址**
  - https://huggingface.co/docs/datasets/index

基础组件之Datasets

Datasets基本使用

- 加载在线数据集（load_dataset）
- 加载数据集某一项任务（load_dataset）
- 按照数据集划分进行加载（load_dataset）
- 查看数据集（index and slice）
- 数据集划分（train_test_split）
- 数据选取与过滤（select and filter）
- 数据映射（map）
- 保存与加载（save_to_disk / load_from_disk）



基础组件之Datasets

Datasets加载本地数据

- **直接加载文件作为数据集**
  - CSV、JSON
- **加载文件夹内全部文件作为数据集**
- **通过预先加载的其他格式转换加载数据集**
  - dict、pandas、list
- **通过自定义加载脚本加载数据集**
  - def _info(self)
  - def _split_generators(self, dl_manager)
  - def _generate_examples(self, filepath)

```python
class CMRC2018TRIAL(datasets.GeneratorBasedBuilder):

    def _info(self) -> DatasetInfo:
        """
            info方法，定义数据集的信息,这里要对数据的字段进行定义
        :return:
        """
        return datasets.DatasetInfo(…

    def _split_generators(self, dl_manager: DownloadManager):
        """
            返回datasets.SplitGenerator
            涉及两个参数：name和gen_kwargs
            name：指定数据集的划分
            gen_kwargs：指定要读取的文件的路径，与_generate_examples的入参数一致
        :param dl_manager:
        :return: [ datasets.SplitGenerator ]
        """
        return [datasets.SplitGenerator(name=datasets.Split.TRAIN,
                                        gen_kwargs={"filepath": "./cmrc2018_trial.json"})]

    def _generate_examples(self, filepath):
        """
            生成具体的样本，使用yield
            需要额外指定key，id从0开始自增就可以
        :param filepath:
        :return:
        """
        # Yields (key, example) tuples from the dataset
        with open(filepath, encoding="utf-8") as f:…
```

基础组件之Model

模型微调代码优化

- **任务类型**
  - 文本分类
- **使用模型**
  - hfl/rbt3
- **优化内容**
  - Datasets数据集加载
  - DataCollatorWithPadding

Step2 加载数据集

```
import torch
from datasets import load_dataset
from transformers import  DataCollatorWithPadding

tokenizer = AutoTokenizer.from_pretrained("hfl/rbt3")
```

```
dataset = load_dataset("csv", data_files="./04-datasets/ChnSentiCorp_htl_all.csv", split='train')
dataset = dataset.filter(lambda x: x["review"] is not None)
datasets = dataset.train_test_split(test_size=0.1)
datasets
```

输出已折叠 ...

Step3 数据预处理

```
def process_function(examples):
    tokenized_examples = tokenizer(examples["review"], max_length=128, truncation=True)
    tokenized_examples["labels"] = examples["label"]
    return tokenized_examples

tokenized_datasets = datasets.map(process_function, batched=True, remove_columns=dataset.column_names)
tokenized_datasets
```

06

基础组件之Evaluate

（1）Evaluate简介

（2）Evaluate基本使用

（3）Evaluate模型微调代码优化

基础组件之Evaluate

Evaluate

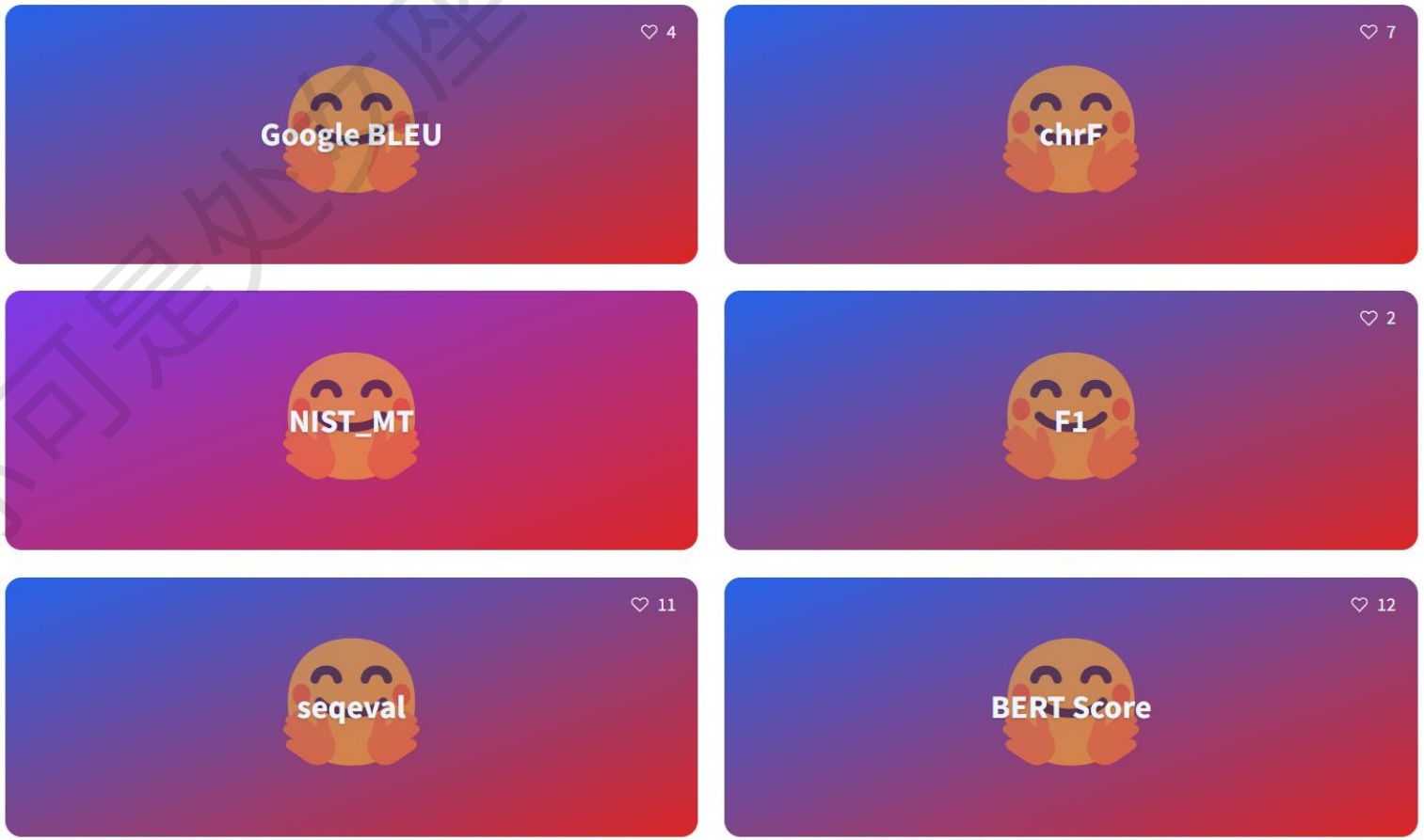
- **简介**
  - evaluate库是一个非常简单易用的机器学习模型评估函数库，

    只需要一行代码便可以加载各种任务的评估函数
- **函数库地址**
  - https://huggingface.co/evaluate-metric
- **文档地址**
  - https://huggingface.co/docs/evaluate/index

基础组件之Evaluate

Evaluate基本使用

- 查看支持的评估函数（list_evaluation_modules）
- 加载评估函数（load）
- 查看评估函数说明（inputs_description）
- 评估指标计算（compute）
  - 全局计算（compute）
  - 迭代计算（add / add_batch）
- 计算多个评估指标（combine）
- 评估结果对比可视化（radar_plot）



基础组件之Evaluate

模型微调代码优化

- **任务类型**
  - 文本分类
- **使用模型**
  - hfl/rbt3
- **优化内容**
  - Evaluate使用多个评估函数

Step7 训练与验证

```python
import evaluate

clf_metric = evaluate.combine(["accuracy", "f1", "recall", "precision"])

def evaluate():
    model.eval()
    with torch.inference_mode():
        for batch in validloader:
            if torch.cuda.is_available():
                batch = {k: v.cuda() for k, v in batch.items()}
            output = model(**batch)
            pred = torch.argmax(output.logits, dim=-1)
            clf_metric.add_batch(pred.long(), batch["labels"].long())
    return clf_metric.compute()

def train(epoch=3, log_step=100):
    global_step = 0
    for ep in range(epoch):
        model.train()
        for batch in trainloader:
            if torch.cuda.is_available():
                batch = {k: v.cuda() for k, v in batch.items()}
            optimizer.zero_grad()
            output = model(**batch)
            output.loss.backward()
            optimizer.step()
            if global_step % log_step == 0:
                print(f"ep: {ep}, global_step: {global_step}, loss: {output.loss.item()}")
            global_step += 1
        clf_result = evaluate()
        print(f"ep: {ep}, result: {clf_result}")
```

07

基础组件之Trainer

（1）Trainer简介

（2）TrainingArguments + Trainer代码优化

基础组件之Trainer

Trainer

- **简介**
  - Trainer是transformers库中提供的训练的函数，内部封装了完整的训练、评估逻辑，并集成了多种的后端，如DeepSpeed、Pytorch FSDP等，搭配TrainingArguments对训练过程中的各项参数进行配置，可以非常方便快捷地启动模型单机 / 分布式训练
  - 需要注意的是
    - 使用Trainer进行模型训练对模型的输入输出是有限制的，要求模型返回元组或者ModelOutput的子类
    - 如果输入中提供了labels，模型要能返回loss结果，如果是元组，要求loss为元组中第一个值
- **文档地址**
  - https://huggingface.co/docs/transformers/main_classes/trainer#trainer

基础组件之Trainer

模型微调代码优化

- **任务类型**
  - 文本分类
- **使用模型**
  - hfl/rbt3
- **优化内容**
  - 使用Trainer + TrainingArgument优化训练流程

完整使用流程

```
from transformers import Trainer, TrainingArguments
# 创建TrainingArguments
training_args = TrainingArguments(...)
# 创建Trainer
trainer = Trainer(args=training_args, ...)
# 模型训练
trainer.train()
# 模型评估
trainer.evaluate()
# 模型预测
trainer.predict()
```

基础入门篇
完结

THANKS