智能计算系统

实验第七章

综合实验

主讲人：张欣

中国科学院计算技术研究所

目录

01 模型推理：Tacotron2语音合成

难度：★★★★ 分值：90

02 模型训练：YOLOv5目标检测

难度：★★★ 分值：80

03 模型训练：BERT的SQuAD任务

难度：★★★★★ 分值：100

综合实验三选一

7.1 模型推理：Tacotron2语音合成

实验目的：

掌握如何使用 Pytorch 实现语音合成系统的Mel谱合成模块，并进行从文本合成语音的推理。具体包括：

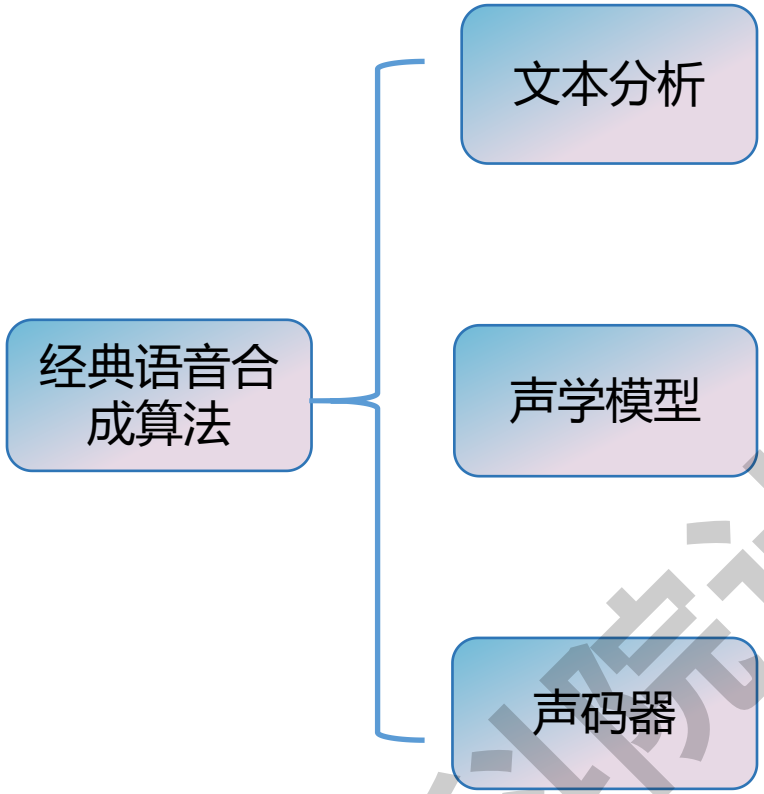
1) 掌握使用 Pytorch 定义完整 Tacotron2 模型的方法;

2) 理解 Tacotron2 的算法细节，如推理与训练时的区别与位置敏感注意力的实现等;

3)掌握利用DLP上已实现的PyTorch框架进行模型移植与推理的基本方法。

Shen J , Pang R , Weiss R J ,et al.Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions[J]. 2017.DOI:10.48550/arXiv.1712.05884.

语音合成概述

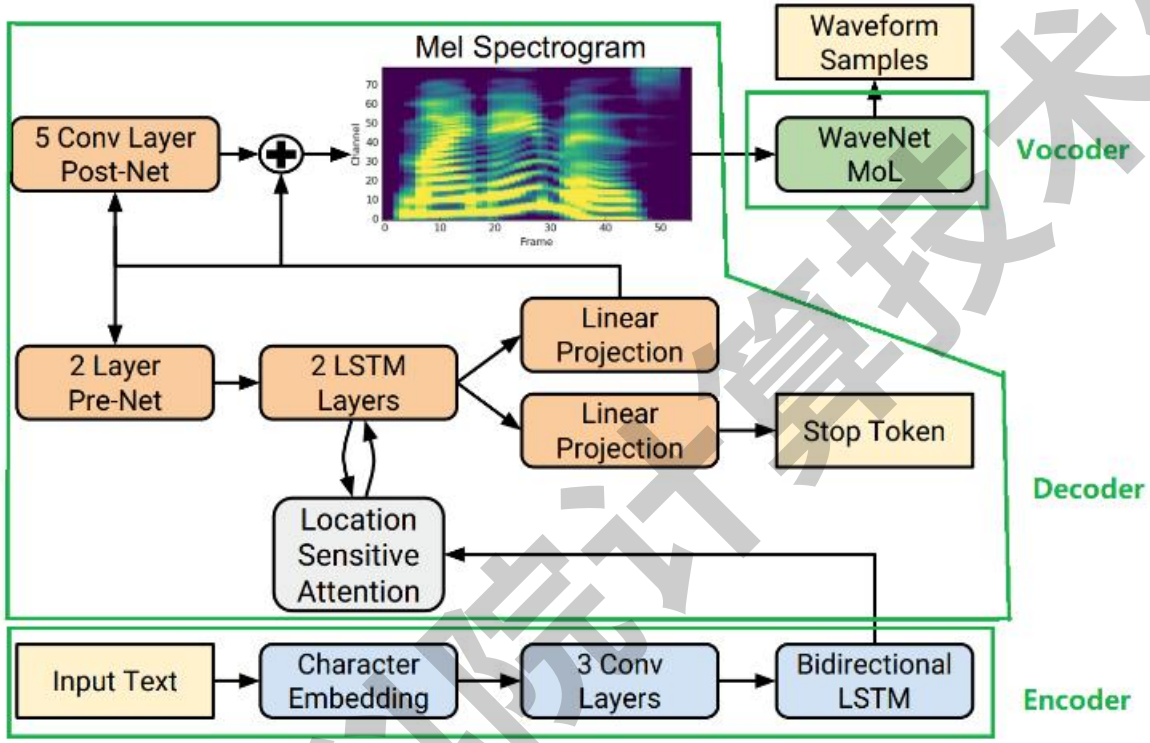


对文本进行归一化、分词处理

将输入的文本信息转换为声学特征（通常是Mel频谱），既可以使用传统的概率模型（如HMM），也可使用基于CNN/RNN/self-attention等构建的网络模型；

基于声学特征生成语音音频作为最终输出。这一部分既可以使用传统基于参数合成的声码器，也可使用WaveNet、WaveGlow、WaveGAN、WaveVAE等多种基于神经网络的声码器。

Tacotron2算法

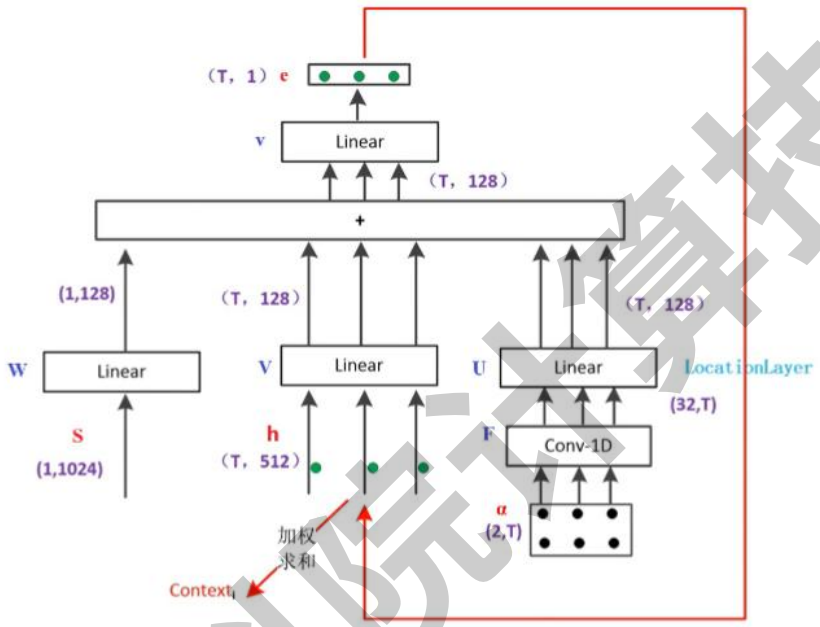


Tacotron2 整体结构图

Tacotron2是一个可直接从文本合成语音的端到端语音合成模型，它使用了带有注意力机制的双层LSTM作为声学模型，并使用WaveNet作为其声码器（后改为使用WaveGlow），二者间使用的声学特征中介为Mel频谱。

位置敏感注意力

$$e = v^T \tanh(Ws + Vh + U(F * \alpha)^T)$$

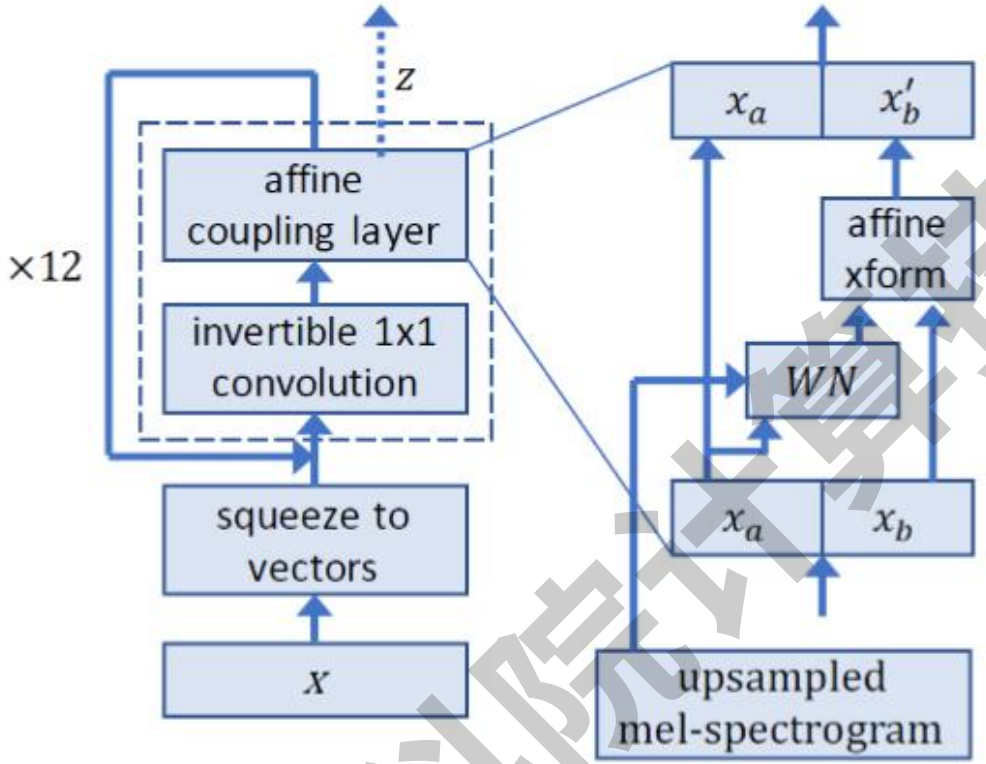


空间敏感注意力机制

它在双层LSTM两层之间生效。其计算权重的思路为：首先定义可任意取值的能量函数，随后用激活函数将其归一化为权值。而能量函数的计算使用的是加性注意力

e就是待求的能量值，s是第一层LSTM的当前隐状态，h为Encoder对文本的编码输出，α是用上一时刻注意力权值和历史注意力权值总和拼接得到的向量。

WaveGLow



WaveGLow 网络结构

- WaveGlow 使用负对数似然（Negative Log-Likelihood, NLL）作为损失函数，以确保生成的音频波形与真实分布尽可能匹配。

$$\log p_\theta(\boldsymbol{x}) = \log p_\theta(z) + \sum_{i=0}^{k} \log \left|\det\left(\boldsymbol{J}\left(f_i^{-1}\right)\right)\right|$$

- affine coupling layer，另一个是1x1 invertible convolution。两者都满足条件：（1）可逆；（2）雅可比行列式容易计算。
- Affine coupling layer负责提高模型的表达能力。但是总所周知，表达能力强的模块（如CNN）都不可逆。Affine用巧妙的方式做到了：既引入了这些表达能力强的模块，又保证了可逆。核心就是：保持部分通道不变。

实验步骤

```
tacotron2_inference
├── .gitignore
├── .gitmodules
├── course_images
│   ├── mel_and_alignments.png
│   ├── tacotron2_infer_step.png
│   ├── tacotron2_structure.bmp
│   └── waveglow_net_structure.bmp
├── model
│   └── pretrained
│       ├── nvidia_tacotron2pyt_fp32_20190427
│       └── nvidia_waveglowpyt_fp32_20190427
├── nvlog_bs1_il128_fp32.json
├── pytorch_tacotron2_inference.ipynb
├── readme-tacotron2.txt
├── requirements.txt
├── src
│   ├── common
│   │   ├── audio_processing.py
│   │   ├── layers.py
│   │   ├── stft.py
│   │   └── utils.py
│   ├── inference.py
│   ├── models.py
│   ├── plot_data.py
│   ├── tacotron2
│   │   ├── arg_parser.py
│   │   ├── model.py
│   │   └── text
│   │       ├── LICENCE
│   │       ├── __init__.py
│   │       ├── cleaners.py
│   │       ├── cmudict.py
│   │       ├── numbers.py
│   │       └── symbols.py
│   ├── test_attn.py
│   ├── test_infer.py 补全推理模块
│   └── waveglow
│       ├── arg_parser.py
│       ├── data_function.py
│       ├── denoiser.py
│       ├── loss_function.py
│       └── model.py 补全位置敏感注意力机制、Encoder、Decoder、Tacotron2模块
└── tacotron2_infer.sh
```

1、补全tacotron2_inference/src/tacotron2/model.py以及tacotron2_inference/src/test_infer.py文件。

2、进入tacotron2_inference/src实验目录，运行python test_atten.py来验证位置敏感注意力机制的正确性。

3、进入tacotron2_inference目录，运行bash tacotron2_infer.sh实现推理

实验评估

- 60 分标准：实现位置敏感注意力机制模块，并通过单模块测试。
- 80 分标准：在 60 分的基础上，正确实现 Tacotron2 中的 Encoder 模块、Decoder 模块、Tacotron2 模块。
- 100 分标准：在 80 分的基础上，在 DLP 平台上正确实现 Tacotron2 的推断模块，完成语音合成推断过程，给定输入文字，可以合成语音。

目录

01 模型推理：Tacotron2语音合成

难度：★★★★ 分值：90

02 模型训练：YOLOv5目标检测

难度：★★★ 分值：80

03 模型训练：BERT的SQuAD任务

难度：★★★★★ 分值：100

综合实验三选一

7.2 模型训练: YOLOv5 目标检测

实验目的:

熟悉目标检测算法原理，掌握YOLOv5的网络结构，能够在DLP实现目标检测算法YOLOv5的训练过程。具体包括:

1）掌握YOLOv5的算法原理;

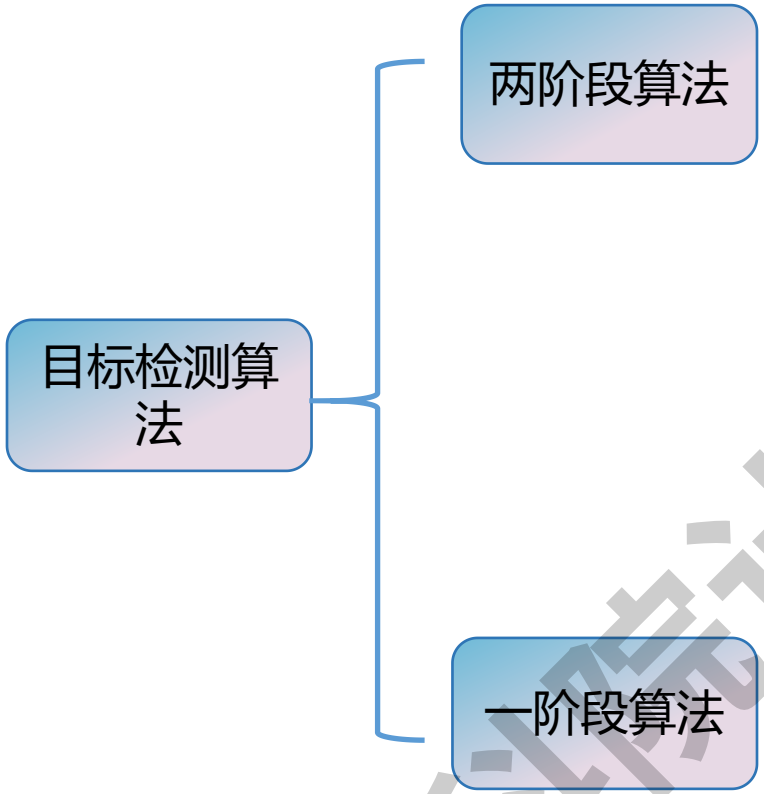
2）掌握YOLOv5的输入端的主要策略;

3）掌握YOLOv5中主要的网络结构;

4）掌握YOLOv5中输出端的主要策略;

5）能够在DLP上实现YOLOv5的训练。

目标检测概述



两阶段算法基于候选区域提取方法，首先产生候选框将所有可能的潜在目标框出来，然后用基于卷积神经网络的图像分类算法对候选框进行分类和进一步的回归，获得最终的边界框（bounding box），两阶段算法的代表性算法包括R-CNN系列。

一阶段算法仅用卷积神经网络直接输出目标位置坐标和分类。一阶段算法的代表性算法包括YOLO、SSD等。经过多个版本的改进，YOLO在模型架构、训练策略、预测技巧等方面进行了不断的升级，提高了模型的检测性能和速度。

YOLO版本

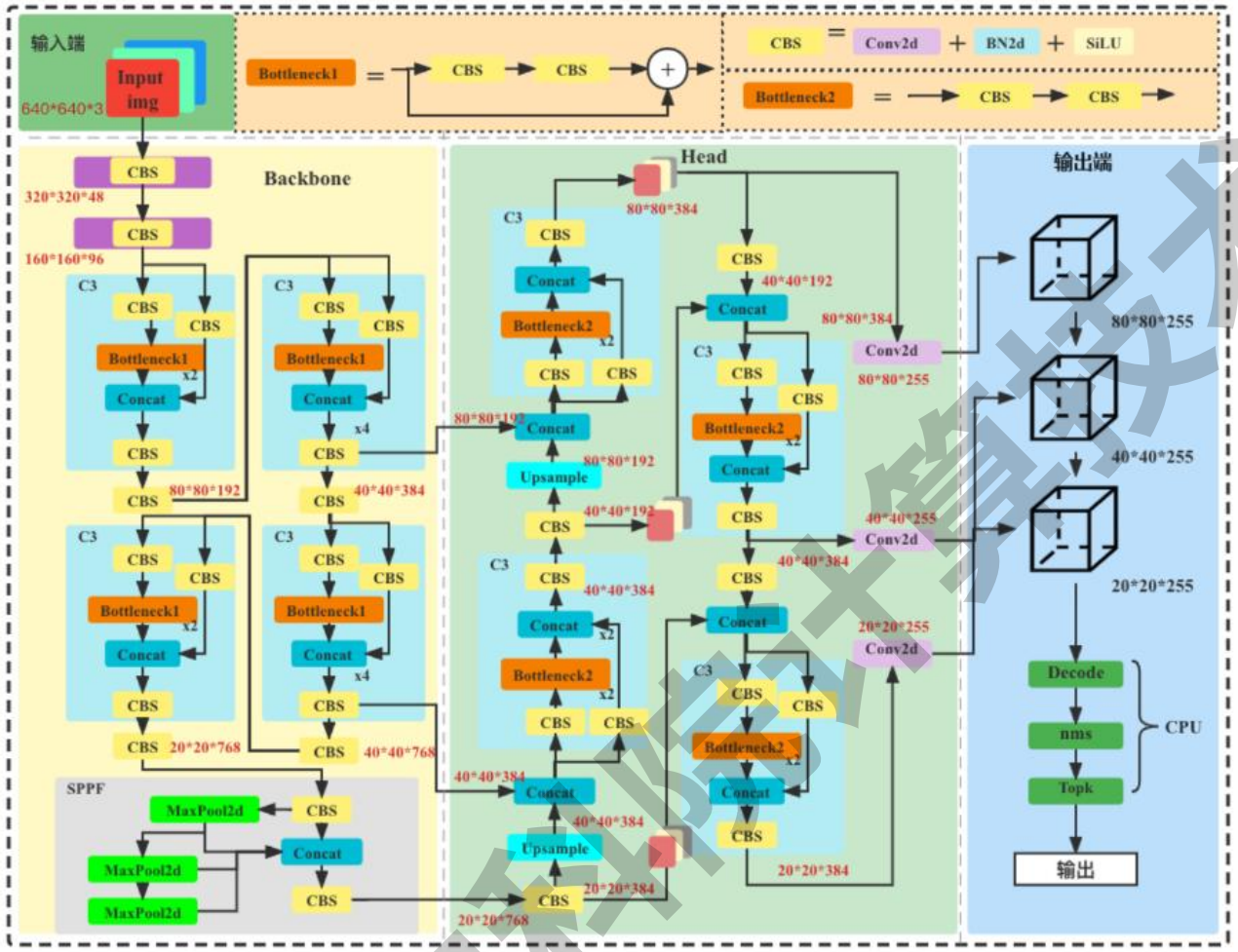
版本	作者	年份	论文题目
YOLO v1	Joseph Redmon	2015	**You Only Look Once:Unified, Real-Time Object Detection**
YOLO v2	Joseph Redmon	2016	**YOLO9000:Better, Faster, Stronger**
YOLO v3	Joseph Redmon	2018	**YOLOv3: An Incremental Improvement**
YOLO v4	Alexey Bochkovskiy	2020	**YOLOv4: Optimal Speed and Accuracy of Object Detection**
YOLO v5	Glenn Jocher	2020	无
YOLOx	Zheng Ge	2021	**YOLOX: Exceeding YOLO Series in 2021**
YOLO v6	Chuyi Li	2022	**YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications**
YOLO v7	Chien-Yao Wang	2023	**YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors**
YOLO v8	Ultralytics	2023	**无**
YOLO v9	Wang Chien-Yao	2024	**YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information**

YOLOv1-v5系列比较

	Backbone	Neck	Head	Prediction
YOLO-v1	GoogleNet	None	Single path	MSE Loss
YOLO-v2	Darknet-19	None	Single path	MSE Loss
YOLO-v3	Darknet-53	FPN	Three paths	MSE Loss
YOLO-v4	Darknet-53 CSP	SPP FPN+PAN	Three paths	CIoU Loss
YOLO-v5	Darknet-53 Focus, CSP	SPP, CSP FPN+PAN	Three paths	CIoU Loss

YOLOv5的目标检测思想并没有发生太大的变化，主要是网络结构的变更，以及更灵活的参数控制方式

YOLOv5算法



YOLOv5 网络结构图

YOLOv5的官方代码中，给出了5个版本的目标检测网络，分别是YOLOv5n、YOLOv5s、YOLOv5m、YOLOv5l、YOLOv5x五个模型。Yolov5n网络是Yolov5系列中深度最小、特征图的宽度最小的网络。后面的4种都是在此基础上不断加深，不断加宽

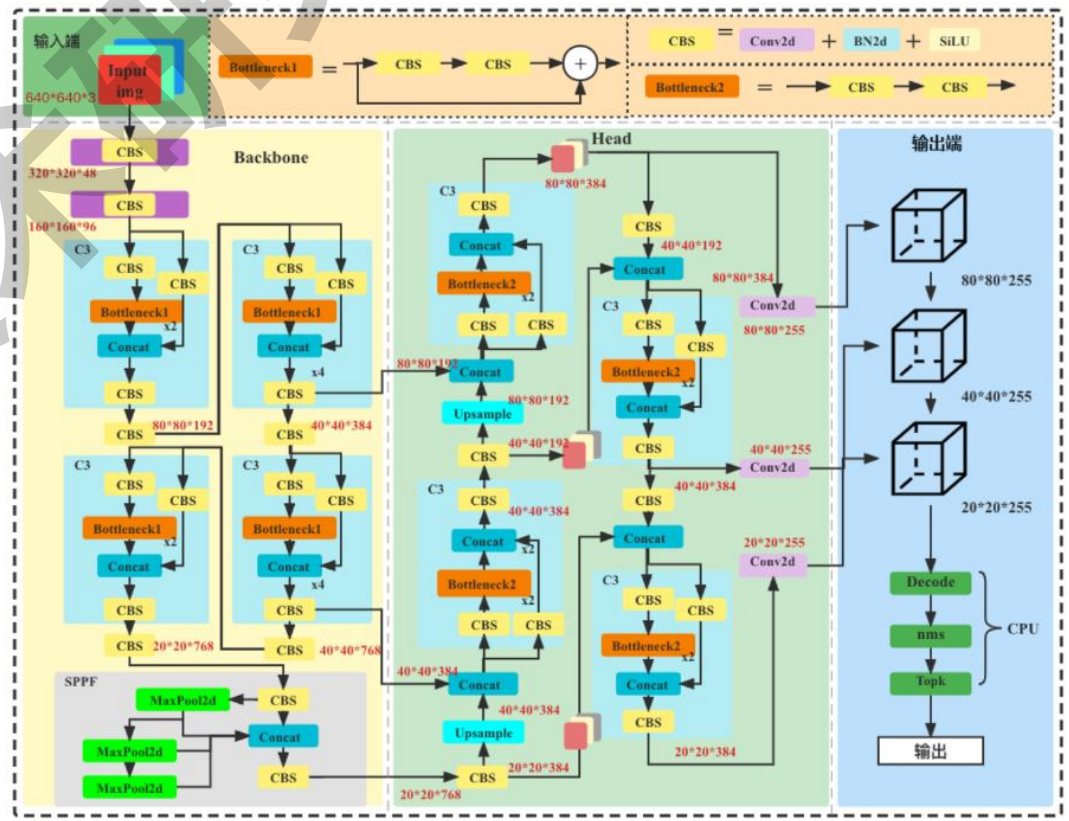
以YOLOv5m为例，其主要由输入端、网络结构、输出端三部分组成。

输入端

- **数据增强：**可以根据任务和数据集的特点进行组合和调整，以提高模型的性能和鲁棒性。MixUp数据增强、Cutout数据增强、Copy paste数据增强、Random affine仿射变换、HSV随机增强图像、随机水平翻转

- **自适应锚框计算：**根据数据集中的目标尺寸和分布自动计算锚框的尺寸和长宽比。

- **自适应图片缩放策略：**根据训练数据中的目标尺寸来自动调整输入图像的大小。这有助于确保模型能够有效地处理不同大小的目标，而无需将所有图像都调整为相同的尺寸。



YOLOv5 网络结构图

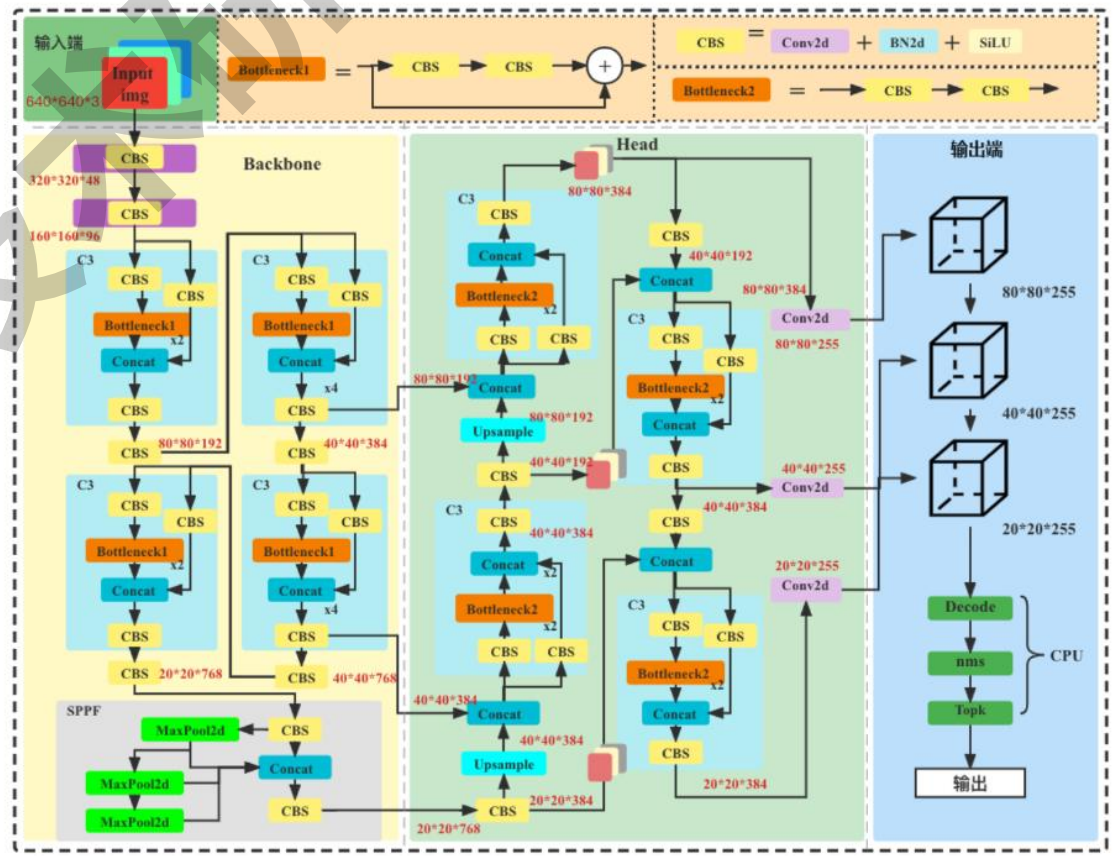
网络结构

- **Backbone（骨干网络）：**

是模型的主要特征提取部分，扮演了非常关键的角色。它由Conv(CBS)、C3以及SPPF组成。

- **Head：**

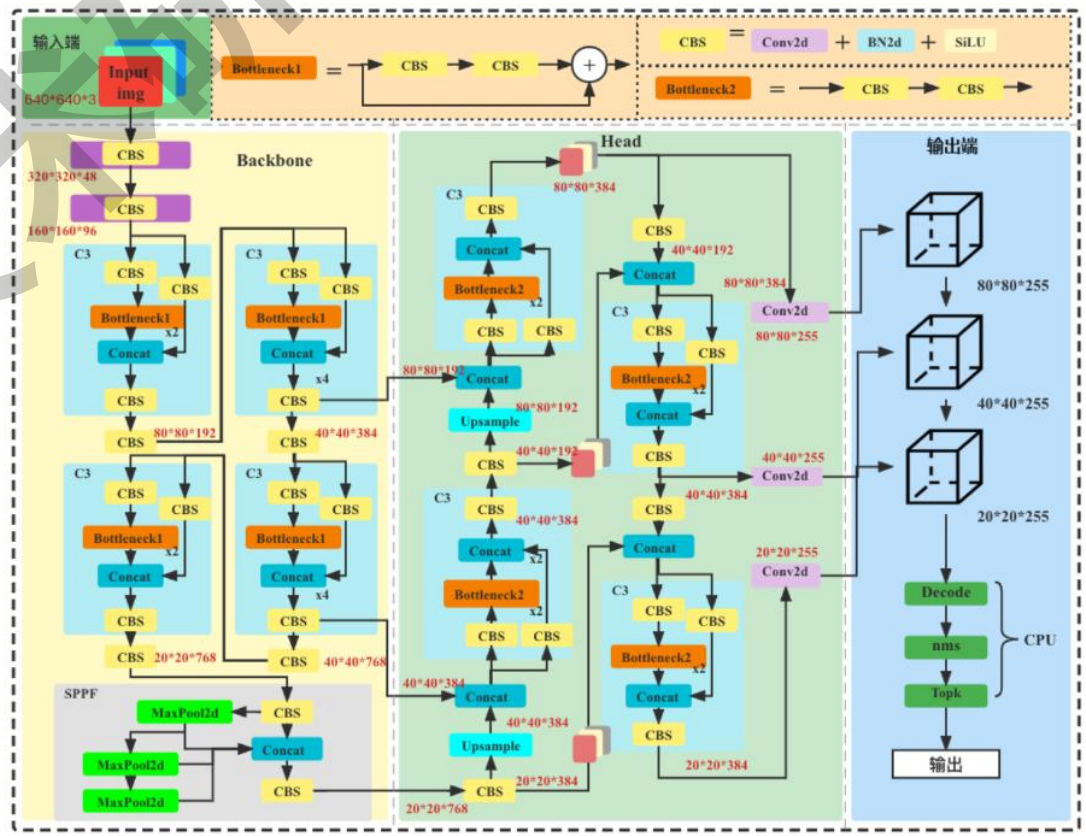
采用了一种特征金字塔网络，用于整合不同层次的特征以提高目标检测性能。它由Conv(CBS)、C3模块以及Upsample、Concat操作构成。



YOLOv5 网络结构图

输出端

- **Decode（解码）**：是将模型输出的预测信息转化为真实世界中的目标框的过程。
- **NMS（非极大值抑制）**：是一种用于去除冗余检测框的技术，它确保每个目标只被保留一个检测框，以消除重叠检测。
- **TopK操作**：是为了限制输出的检测结果数量，通常会选择置信度分数最高的前K个检测框，K是一个用户定义的参数。这有助于减少处理时间和资源需求，特别是在需要高效的实时应用中。
- **损失函数**：YOLOv5使用的损失函数是一种组合损失函数，包括三个部分：置信度损失、框位置损失和类别损失。



YOLOv5 网络结构图

实验步骤

补全yolov5-train/yolov5/utils/augmentations.py、yolov5-train/yolov5/models/common.py以及yolov5-train/yolov5/utils/general.py文件。

进入yolov5-train/yolov5实验目录,采用三种方式进行训练，并进行精度验证

1）直接训练：bash direct_train.sh
2）用预训练模型进行训练：bash pretrained_train.sh
3）恢复训练：bash resume_train.sh
4）精度验证：bash ac_val.sh

```
yolov5_train
├── course_images
│   └── yolov5m.png
├── pytorch_yolov5_train.ipynb
├── readme-yolov5.txt
├── requirements_mlu.txt
├── utils_mlu
│   ├── __pycache__
│   │   ├── metric.cpython-36.pyc
│   │   └── metric.cpython-37.pyc
│   ├── collect_env.py
│   ├── common_utils.sh
│   ├── configs.json
│   └── metric.py
├── yolov5
│   ├── CONTRIBUTING.md
│   ├── __pycache__
│   │   ├── val.cpython-36.pyc
│   │   └── val.cpython-37.pyc
│   ├── ac_val.sh
│   ├── data
│   │   ├── .ipynb_checkpoints
│   │   │   └── coco-checkpoint.yaml
│   │   ├── Argoverse.yaml
│   │   ├── GlobalWheat2020.yaml
│   │   ├── Objects365.yaml
│   │   ├── SKU-110K.yaml
│   │   ├── VOC.yaml
│   │   ├── VisDrone.yaml
│   │   ├── coco.yaml
│   │   ├── coco128.yaml
│   │   ├── hyps
│   │   │   ├── hyp.finetune.yaml
│   │   │   ├── hyp.finetune_objects365.yaml
│   │   │   ├── hyp.scratch-high.yaml
│   │   │   ├── hyp.scratch-low.yaml
│   │   │   └── hyp.scratch.yaml
│   │   ├── images
│   │   │   ├── bus.jpg
│   │   │   └── zidane.jpg
│   │   ├── scripts
│   │   │   ├── download_weights.sh
│   │   │   ├── get_coco.sh
│   │   │   └── get_coco128.sh
│   │   └── xView.yaml
│   ├── detect.py
│   ├── direct_train.sh
│   ├── export.py
│   ├── hubconf.py
│   ├── models
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-36.pyc
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── common.cpython-36.pyc
│   │   │   ├── common.cpython-37.pyc
│   │   │   ├── experimental.cpython-36.pyc
│   │   │   ├── experimental.cpython-37.pyc
│   │   │   ├── yolo.cpython-36.pyc
│   │   │   └── yolo.cpython-37.pyc
│   │   ├── common.py补全Conv模块、C3模块、SPPF模块
│   │   ├── experimental.py
│   │   ├── hub
│   │   │   ├── anchors.yaml
│   │   │   ├── yolov3-spp.yaml
│   │   │   ├── yolov3-tiny.yaml
│   │   │   ├── yolov3.yaml
│   │   │   ├── yolov5-bifpn.yaml
│   │   │   ├── yolov5-fpn.yaml
│   │   │   ├── yolov5-p2.yaml
│   │   │   ├── yolov5-p6.yaml
│   │   │   ├── yolov5-p7.yaml
│   │   │   ├── yolov5-panet.yaml
│   │   │   ├── yolov5l6.yaml
│   │   │   ├── yolov5m6.yaml
│   │   │   ├── yolov5n6.yaml
│   │   │   ├── yolov5s-ghost.yaml
│   │   │   ├── yolov5s-transformer.yaml
│   │   │   ├── yolov5s6.yaml
│   │   │   └── yolov5x6.yaml
│   │   ├── tf.py
│   │   ├── yolo.py
│   │   ├── yolov5l.yaml
│   │   ├── yolov5m.yaml
│   │   ├── yolov5n.yaml
│   │   ├── yolov5s.yaml
│   │   └── yolov5x.yaml
│   ├── pretrained_train.sh
│   ├── requirements.txt
│   ├── resume_train.sh
│   ├── runs
│   │   ├── train
│   │   └── val
│   ├── train.py
│   ├── utils
│   │   ├── .ipynb_checkpoints
│   │   │   └── torch_utils-checkpoint.py
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-36.pyc
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── activations.cpython-37.pyc
│   │   │   ├── augmentations.cpython-36.pyc
│   │   │   ├── augmentations.cpython-37.pyc
│   │   │   ├── autoanchor.cpython-36.pyc
│   │   │   ├── autoanchor.cpython-37.pyc
│   │   │   ├── callbacks.cpython-36.pyc
│   │   │   ├── callbacks.cpython-37.pyc
│   │   │   ├── datasets.cpython-36.pyc
│   │   │   ├── datasets.cpython-37.pyc
│   │   │   ├── downloads.cpython-36.pyc
│   │   │   ├── downloads.cpython-37.pyc
│   │   │   ├── general.cpython-36.pyc
│   │   │   ├── general.cpython-37.pyc
│   │   │   ├── loss.cpython-36.pyc
│   │   │   ├── loss.cpython-37.pyc
│   │   │   ├── metrics.cpython-36.pyc
│   │   │   ├── metrics.cpython-37.pyc
│   │   │   ├── plots.cpython-36.pyc
│   │   │   ├── plots.cpython-37.pyc
│   │   │   ├── torch_utils.cpython-36.pyc
│   │   │   └── torch_utils.cpython-37.pyc
│   │   ├── activations.py
│   │   ├── augmentations.py补全图片缩放策略
│   │   ├── autoanchor.py
│   │   ├── aws
│   │   │   ├── __init__.py
│   │   │   ├── mime.sh
│   │   │   ├── resume.py
│   │   │   └── userdata.sh
│   │   ├── callbacks.py
│   │   ├── datasets.py
│   │   ├── downloads.py
│   │   ├── flask_rest_api
│   │   │   ├── README.md
│   │   │   ├── example_request.py
│   │   │   └── restapi.py
│   │   ├── general.py补全NMS
│   │   ├── google_app_engine
│   │   │   ├── Dockerfile
│   │   │   ├── additional_requirements.txt
│   │   │   └── app.yaml
│   │   ├── loggers
│   │   │   ├── __init__.py
│   │   │   ├── __pycache__
│   │   │   │   ├── __init__.cpython-36.pyc
│   │   │   │   └── __init__.cpython-37.pyc
│   │   │   └── wandb
│   │   │       ├── README.md
│   │   │       ├── __init__.py
│   │   │       ├── __pycache__
│   │   │       │   ├── __init__.cpython-36.pyc
│   │   │       │   ├── __init__.cpython-37.pyc
│   │   │       │   ├── wandb_utils.cpython-36.pyc
│   │   │       │   └── wandb_utils.cpython-37.pyc
│   │   │       ├── log_dataset.py
│   │   │       ├── sweep.py
│   │   │       ├── sweep.yaml
│   │   │       └── wandb_utils.py
│   │   ├── loss.py
│   │   ├── metrics.py
│   │   ├── plots.py
│   │   └── torch_utils.py
│   ├── val.py
│   └── yolov5_mlu.patch
└── yolov5_model
    ├── yolov5m.pt
    └── yolov5m.yaml
```

实验评估

实验评分标准为：

- 60 分标准：补全输入端的自适应图片缩放策略，得到正确大小的输入图片。
- 80 分标准：在 60 分的基础上，补全网络结构模块，正确实现 Conv、C3、SPPF 模块的代码。
- 100 分标准：在 80 分的基础上，补全输出端的 NMS 非极大值抑制模块，去除冗余的边界框，能够完成直接训练、预训练、恢复训练三种训练方式，并进行精度验证。

目录

01 模型推理：Tacotron2语音合成

难度：★★★★ 分值：90

02 模型训练：YOLOv5目标检测

难度：★★★ 分值：80

03 模型训练：BERT的SQuAD任务

难度：★★★★★ 分值：100

综合实验三选一

7.2 模型训练: BERT 的 SQuAD 任务

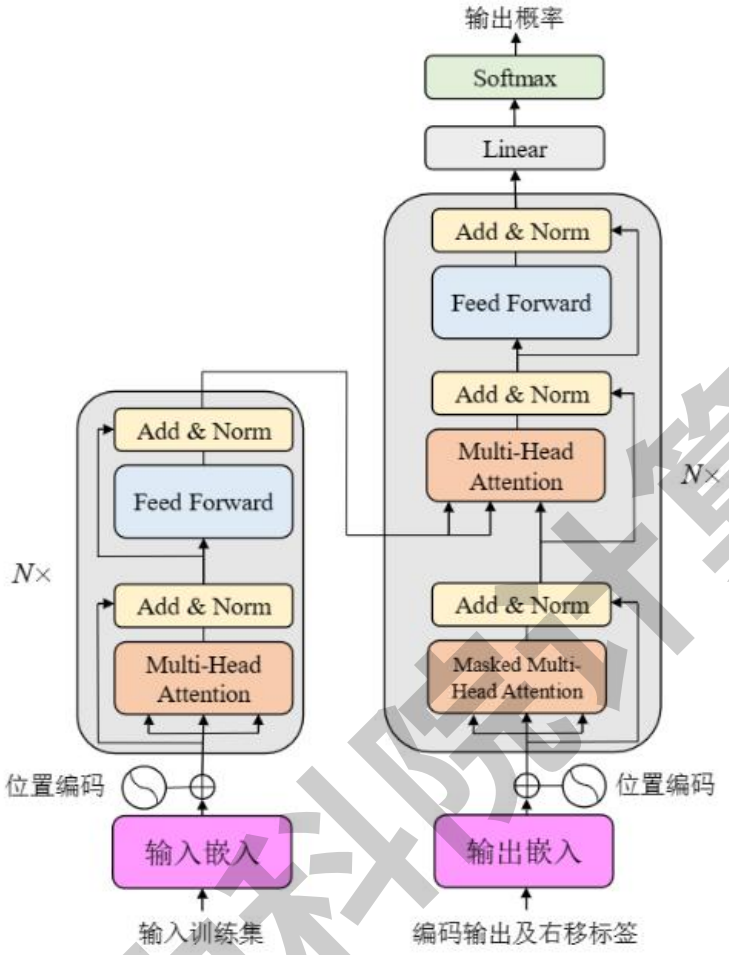
实验目的:

熟悉自然语言处理代表性算法 BERT 的原理，掌握在 DLP 平台 MLU 370 上移植 BERT 的方法和流程，并在 MLU370 上对已经完成预训练的 BERT 网络模型针对问答任务进行微调训练。具体包括:

1） 掌握 BERT 算法的基本原理，了解自然语言处理问答任务数据集 SQuAD。

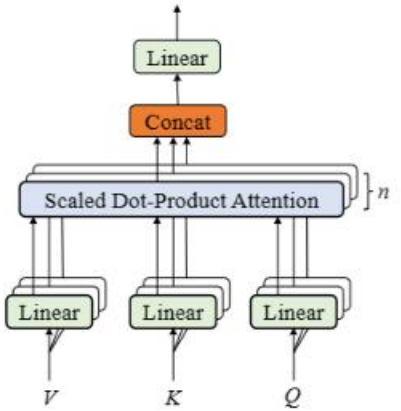
2）掌握使用 Pytorch 编写问答系统应用以及移植到 DLP 平台的方法。

Transformer概述

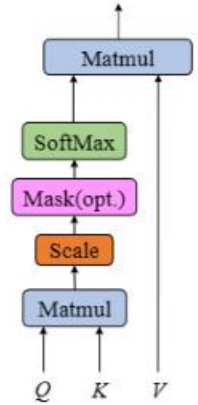


Transformer 结构图

要包括编码器和解码器两个部分
编码器由6个相同的层（Layer）组成
解码器也由6个相同的层组成



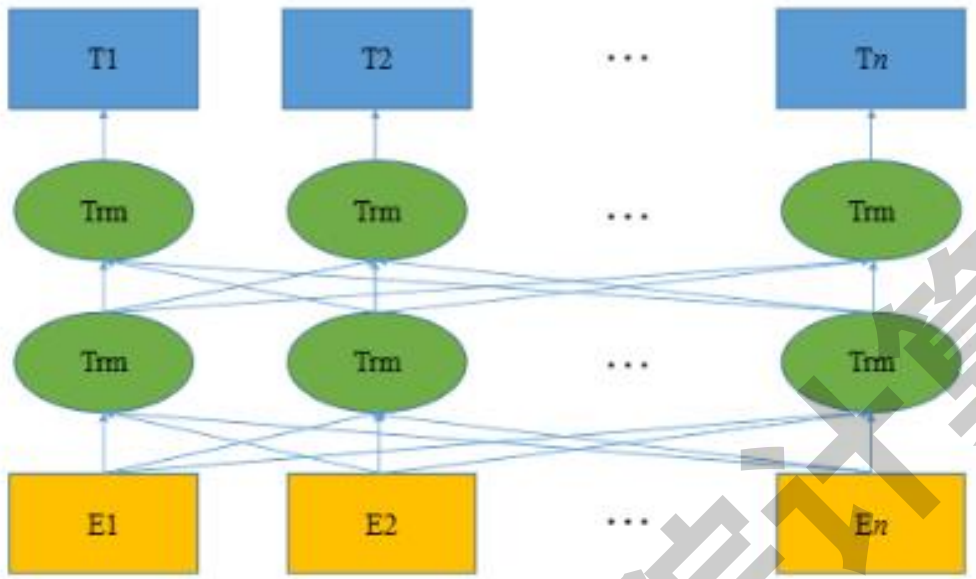
Multi-Head Attention



Scaled Dot-Product Attention

左图为 Multi-Head Attention 单元。右图为 Scaled Dot-Product Attention 单元。

BERT网络



BERT 结构图

BERT网络的全称为基于Transformer的双向编码器表示（Bidirectional Encoder Representation from Transformers）。其采用了Transformer的Encoder部分，并且进行双向的Block连接。与Transformer相比，BERT不仅可获取语句之前的信息，还可获取未来的信息。

在本实验中，采用了12层Encoder，每层有12个Attention，词向量的维度为768。

BERT网络训练

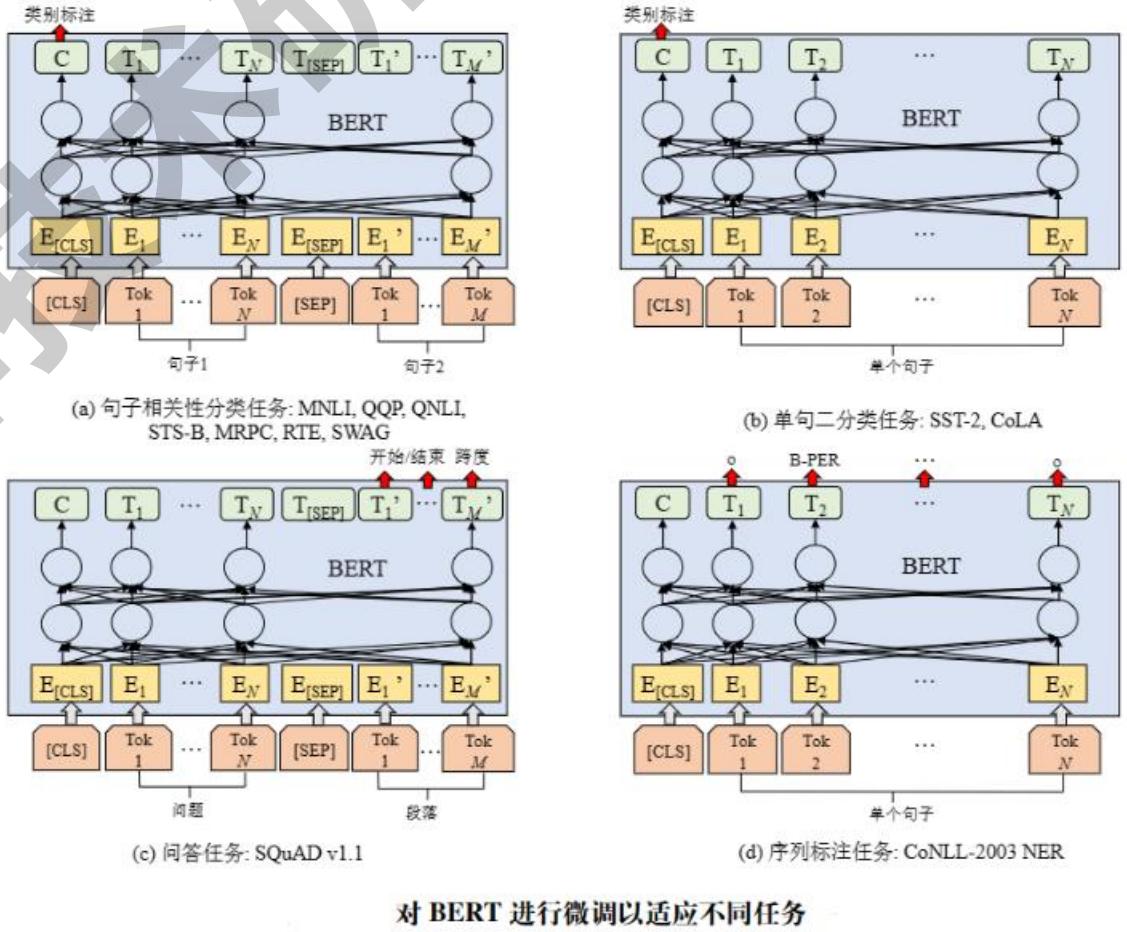
BERT预训练子任务

- 掩码语言模型 Masked Language Model)

  通过随机遮挡每一个句子中 15% 的词，用其上下文来做预测，在计算损失函数时只计算被遮挡掉的输入元素。

- 后句预测 (Next Sentence Prediction)

  可以理解简单的分类任务，用以判断两个输入语句是否为前后关系



(a) 句子相关性分类任务: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) 单句二分类任务: SST-2, CoLA

(c) 问答任务: SQuAD v1.1

(d) 序列标注任务: CoNLL-2003 NER

对 BERT 进行微调以适应不同任务

实验步骤


```
bert_train
├── bert-test.sh
├── bert-train.sh
├── course_images
│   ├── .ipynb_checkpoints
│   │   └── bert_base_model-checkpoint.png
│   └── bert_base_model.png
├── pretrain_bert_base_cased
│   ├── .ipynb_checkpoints
│   │   └── config-checkpoint.json
│   ├── config.json
│   ├── pytorch_model.bin
│   ├── tokenizer.json
│   └── vocab.txt
├── pytorch_bert_train.ipynb
├── readme-bert.txt
├── requirements.txt
├── run_squad.py补全数据加载模块、网络训练模块、循环遍历模块、精度验证模块执行评估模块、主题函数实现模块
├── runs
└── squad
    ├── dev-v1.1.json
    └── train-v1.1.json
```

1、补全bert_train/run_squad.py文件

2、使用预训练模型进行微调：bash bert_train.sh

3、精度验证：bash bert_test.sh

敬请指正！

课程官网：http://novel.ict.ac.cn/aics

MOOC网址：

https://space.bilibili.com/494117284/video