【他山之石】深度学习 | BERT详解

人工智能前沿讲习 5月29日

收录于话题
#他山之石

101个

“他山之石，可以攻玉”，站在巨人的肩膀才能看得更高，走得更远。在科研的道路上，更需借助东风才能更快前行。为此，我们特别搜集整理了一些实用的代码链接，数据集，软件，编程技巧等，开辟“他山之石”专栏，助你乘风破浪，一路奋勇向前，敬请关注。

来源：知乎—VoidOc
地址：https://zhuanlan.zhihu.com/p/130913995

BERT (Bidirection Encoder Representations from Transformers)

炼丹强推Keras:
https://github.com/CyberZHG/keras-bert

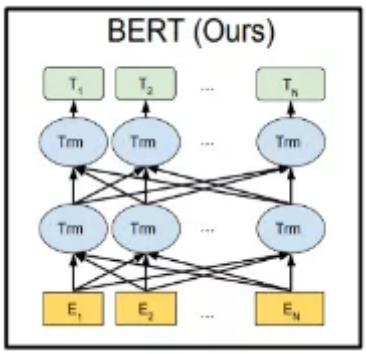
https://github.com/bojone/bert4keras

01

介绍

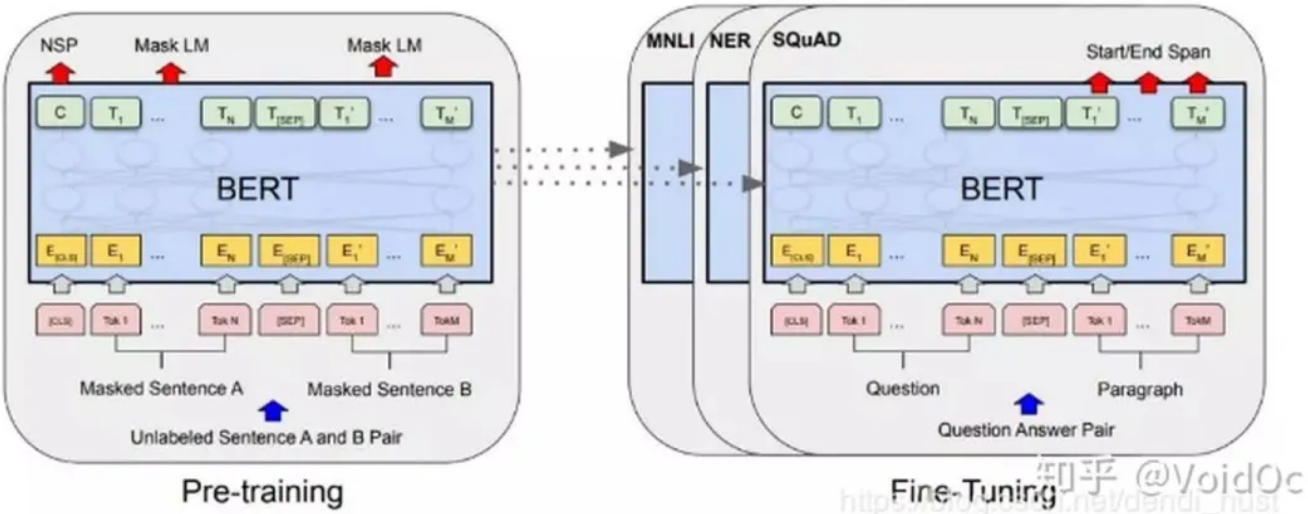
google 论 文 ： 《 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
https://arxiv.org/abs/1810.04805

模型结构图：



使用BERT模型解决NLP任务需要分为两个阶段：

1）Pre-training预训练：用大量的无监督文本通过自监督训练的方式进行训练，把文本中包含的语言知识（包括：词法、语法、语义等特征）以参数的形式编码到Transformer-encoderlayer中。预训练模型学习到的是文本的通用知识，不依托于某一项NLP任务；



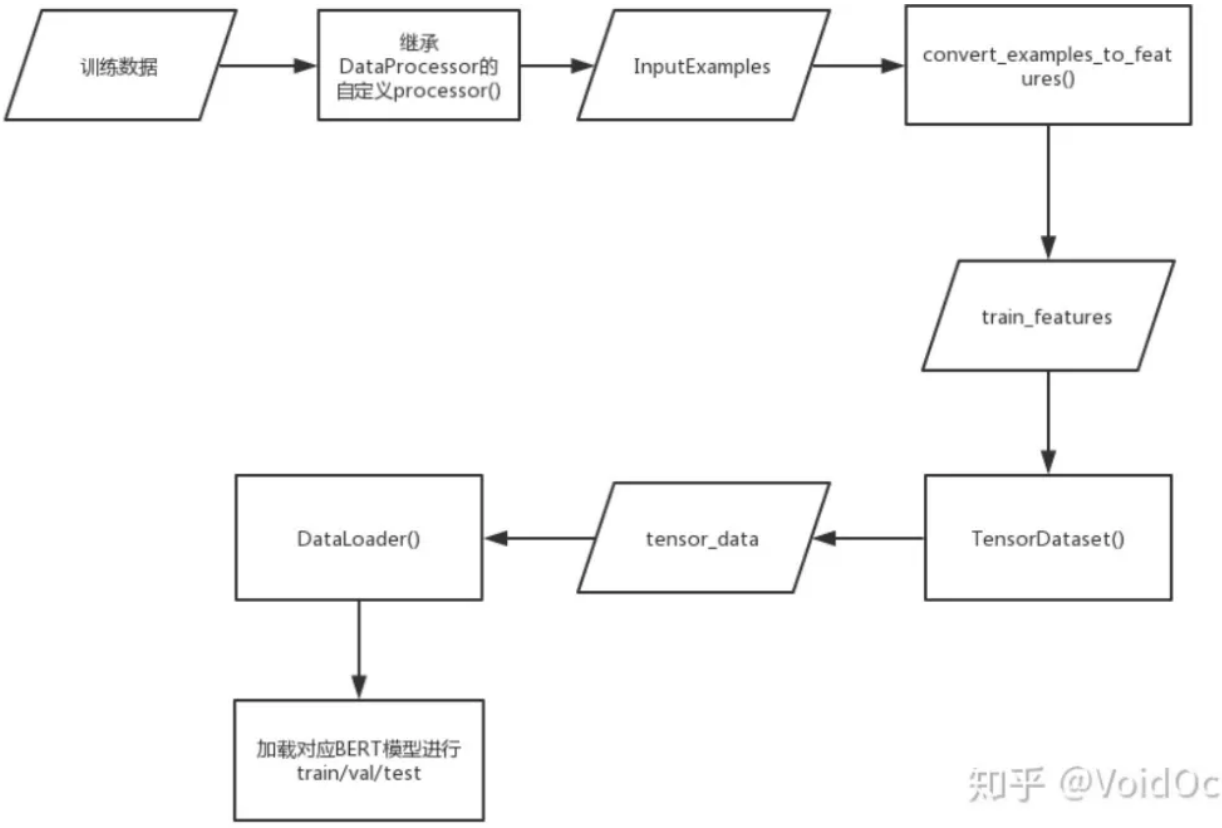
2） Fine-tune阶段：

NLP 问题被证明同图像一样，可以通过 finetune 在垂直领域取得效果的提升。Bert 模型本身极其依赖计算资源，从 0 训练对大多数开发者都是难以想象的事。在节省资源避免重头开始训练的同时，为更好的拟合垂直领域的语料，我们有了 finetune 的动机。

> Finetune过程：
> 构建图结构，截取目标张量，添加新层
> 加载目标张量权重
> 训练新层
> 全局微调

BERT的代码同论文里描述的一致，主要分为两个部分。一个是训练语言模型（language model）的预训练（pretrain）部分。另一个是训练具体任务(task)的fine-tune部分。在开源的代码中，fine-tune的入口针对不同的任务分别在run_classifier.py和run_squad.py。

其中run_classifier.py适用的任务为分类任务。如CoLA、MRPC、MultiNLI这些数据集。而run_squad.py适用的是阅读理解(MRC)任务，如squad2.0和squad1.1；流程大致如下图：



fine-tune的整体流程

文本分类任务fine-tuning例子：使用BERT进行fine-tuning [1] / 中文语料的 Bert finetune [2]

阅读理解（问答）任务fine-tuning例子：【技术分享】BERT系列（三）-- BERT在阅读理解与问答上应用 [3]

NER(单句子标注任务也叫命名实体识别任务) fine-tuning例子：Bert语言模型fine-tune微调做中文NER命名实体识别 [4]

单句子标注任务也叫命名实体识别任务（Named Entity Recognition），简称NER，常见的NER数据集有CoNLL-2003 NER[32]等。该任务是指识别文本中具有特定意义的实体，主要包括人名、地名、机构名、专有名词等，以及时间、数量、货币、比例数值等文字。举个例子：“明朝建立于1368年，开国皇帝是朱元璋。介绍完毕！”那么我们可以从这句话中提取出的实体为：

(1) 机构：明朝

(2) 时间：1368年

(3) 人名：朱元璋

同样地，BERT在NER任务上也不能通过添加简单的分类层进行微调，因此我们需要添加特定的体系结构来完成NER任务。不过，在此之前，我们得先了解一下数据集的格式，如图 8.10所示。

它的每一行由一个字及其对应的标注组成，标注采用BIO（B表示实体开头，I表示在实体内部，O表示非实体），句子之间用一个空行隔开。当然了，如果我们处理的是文本含有英文，则标注需采用BIOX，X用于标注英文单词分词之后的非首单词，比如“Playing”在输入BERT模型前会被BERT自带的Tokenization工具分词为“Play”和“# #ing”，此时“Play”会被标注为“O”，则多余出来的“# #ing”会被标注为“X”。

了解完整体的数据格式，我们就开始了解整体的NER任务是如何通过BERT来训练的。如图 8.11（d）所示，将BERT最后一层向量 [C]LxH输入到输出层。具体运算逻辑是初始化输出层的权重矩阵[W]KxH ，此时K为1。我们通过公式8.5得到句子的概率向量logit ，进而知道了每一个字或者英文单词的标注概率。然后，我们可以直接通过计算 logit与真实标签之间的差值得到loss ，从而开始梯度下降训练。

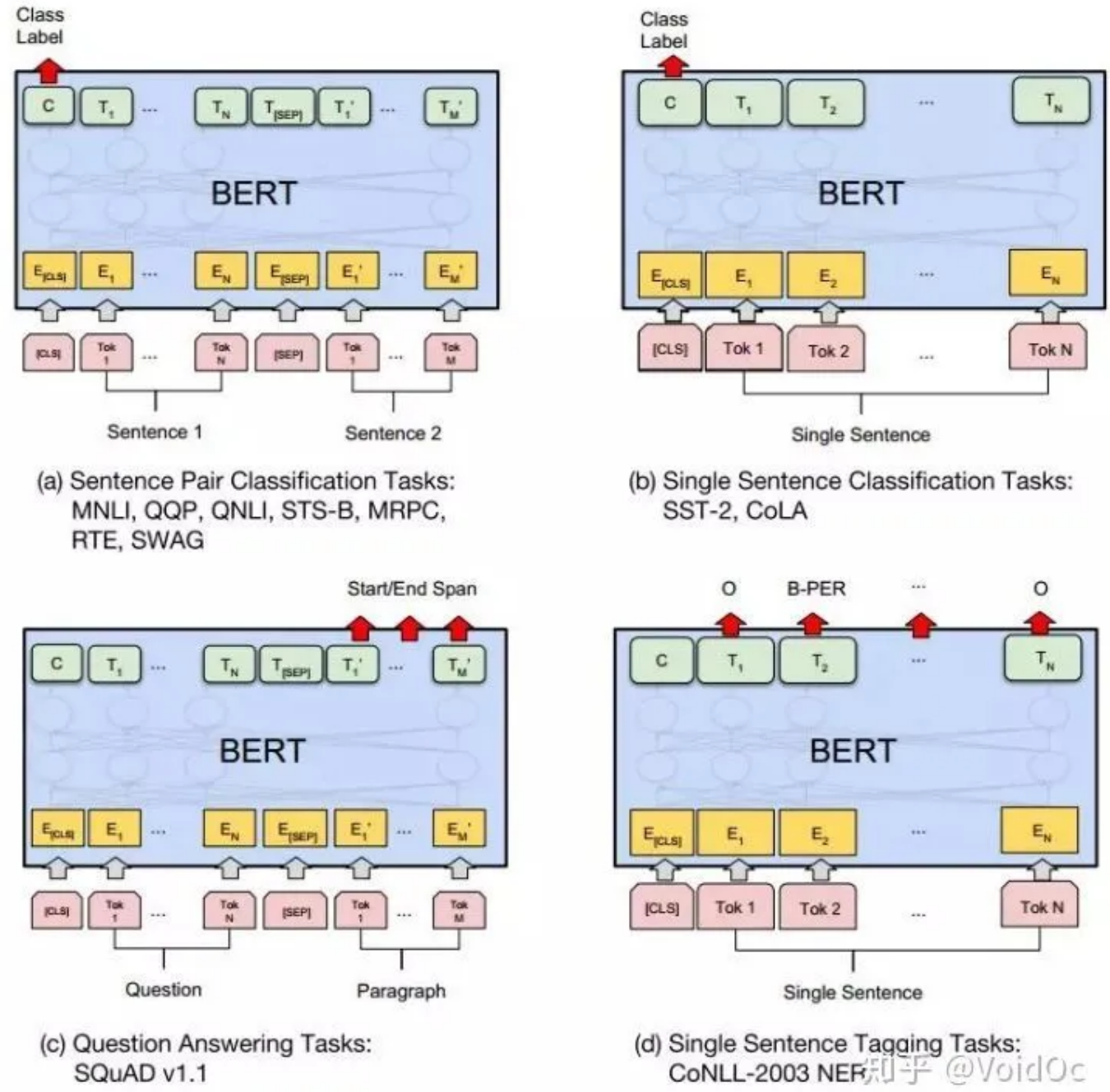
当然了，我们也可以将logit 灌入Bi-LSTM进行学习，因为Bi-LSTM能更好地学习文本的上下文关系，最后再下接一个CRF（Conditional Random Field）层拟合真实标签来进行梯度下降训练。

至于为何要加入CRF层，主要是CRF层可以在训练过程中学习到标签的约束条件。比如，“B-ORG I-ORG”是正确的，而“B-PER I-ORG”则是错误的；“I-PER I-ORG”是错误的，因为命名实体的开头应该是“B-”而不是“I-”，且两个“I-”在同一个实体应该一致。有了这些有用的约束，模型预测的错误序列将会大大减少。

```
明 B-ORG
朝 I-ORG
建 O
立 O
于 O
1 B-TIME
3 I-TIME
6 I-TIME
8 I-TIME
年 I-TIME
， O
开 O
国 O
皇 O
帝 O
是 O
朱 B-PER
元 I-PER
璋 I-PER
。 O

介 O
绍 O
完 O
毕 O
！ O
```

NER数据格式



NLP四大下游任务微调示意图

BRET特点总结：

只有encoder没有decoder->预训练模型，可以接各种下游任务，它的输出只是文本表示，所以不能使用固定的decoder。

BERT是百层左右的深度神经网络，才能把各种语言学的特征提取出来。BERT面世之前，NLP领域的神经网络基本只有几层，Transformer架构出来之后才有可能将NLP网络推向几十至上百层。浅层是分析语法，语法层级的特征，深层进入语义的范畴。

BERT是用Self-Attetion作为特征提取器

动态词向量。在Word2Vec，GloVe的年代，词向量都是静态的，一旦训练之后词向量就固定不变了，但是这就限制了模型对多义词的识别能力，比如apple可以指水果也可以指苹果公司，因此词向量需要动态变

化。

双向语言模型。

02

模型结构

2.1 模型输入

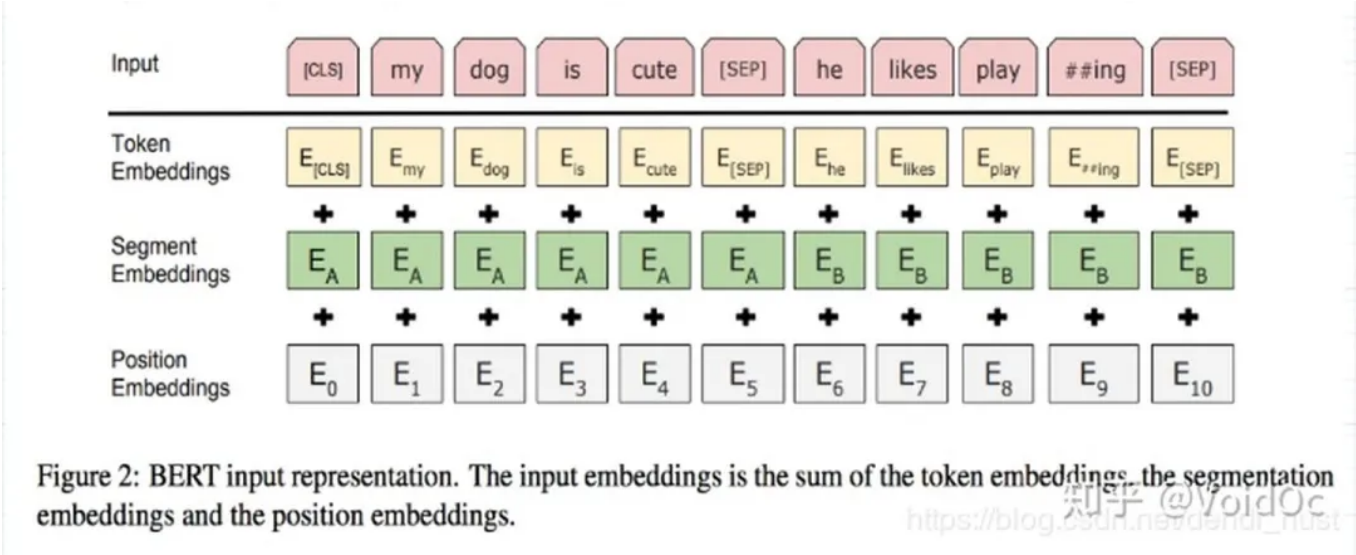
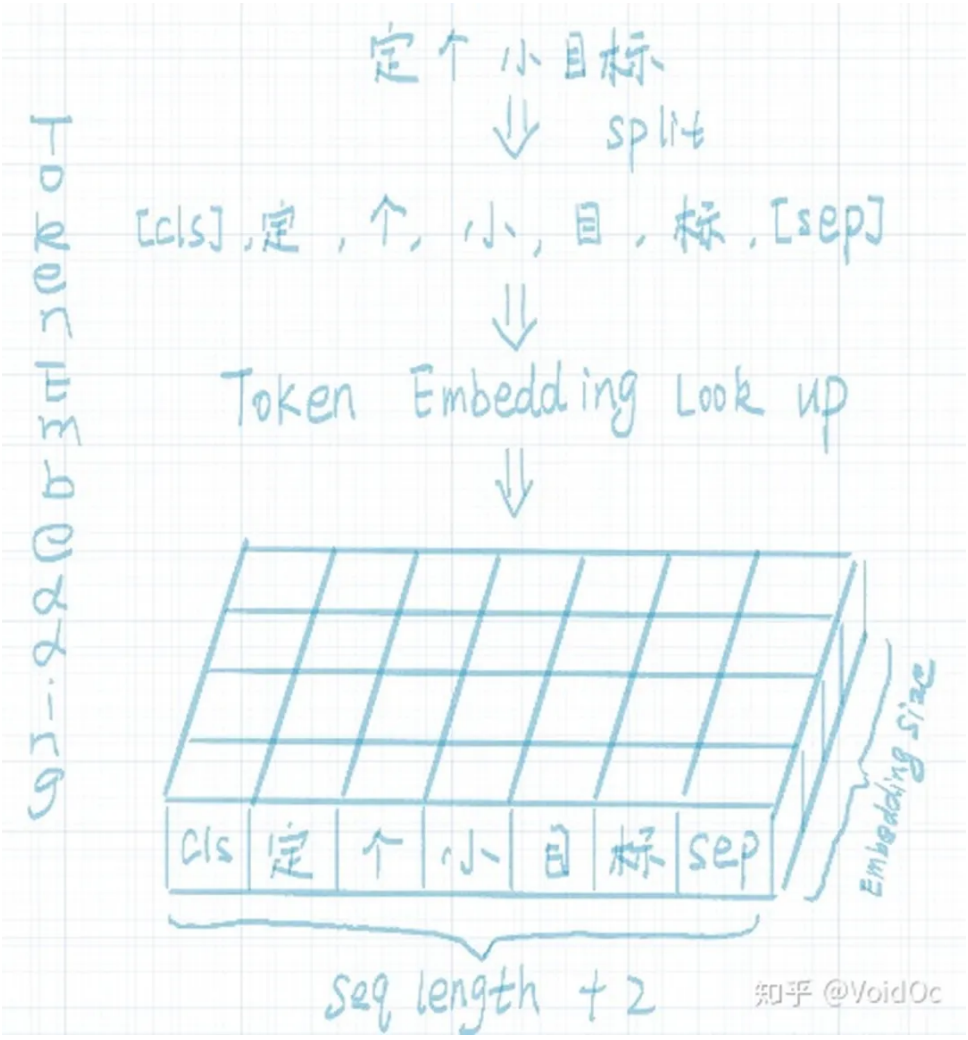


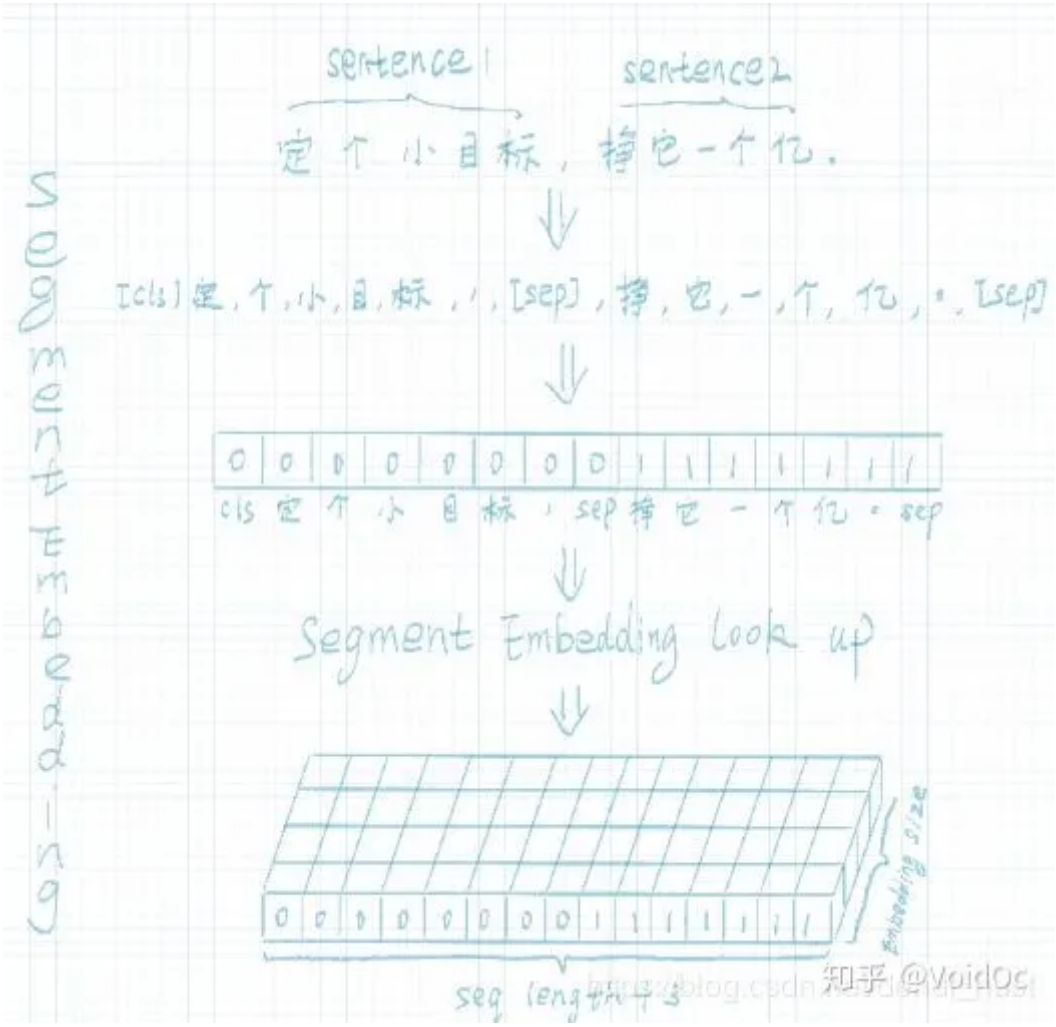
Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

如上图所示，BERT模型有两个特殊的token：CLS （用于分类任务）、 SEP（用于断句），以及三个embedding：

Token embedding：输入的文本经过tokenization之后，将CLS插入tokenization结果的开头，SEP插入到tokenization结果的结尾。然后进行token embedding look up。shape为：[seq_length, embedding_dims]。流程如下图所示：



Segment embedding：在NSP任务中，用于区分第一句和第二句。segment embedding中只有 0 和 1 两个值，第一句所有的token（包括cls和紧随第一句的sep）的segment embedding的值为0，第二句所有的token（包括紧随第二句的sep）的segment embdding的值为1。shape为：[seq_length, embedding_dims]。流程如下图所示：



Position embedding：因Transformer-encoderlayer无法捕获文本的位置信息，而文本的位置信息又非常重要（“你欠我500万” 和 “我欠你500万”的感觉肯定不一样），因此需要额外把位置信息输入到模型中。BERT的位置信息是通过sin函数和cos函数算出来的，shape为：[seq_length, embedding_dims]。该部分参数在训练时不参与更新。

因此，BERT的输入为：

token_embedding + segment_embedding + position_embedding

03

代码实现

3.1 例子：新闻文本分类

数据：新浪新闻数据（链接：https://pan.baidu.com/s/1-Lck_ivs2ryBcrXY0HoR_g 提取码：2uwc）；

```
#run.py
# batch_size
```

```
batch_size = 8
# 学习率
lr = 1e-5
# 是否使用gpu
cuda = False
# 训练批次
epoches = 20
# sequence 最大长度
max_length = 256

# 得到attention mask
def get_atten_mask(tokens_ids, pad_index=0):
    return list(map(lambda x: 1 if x != pad_index else 0, tokens_ids))

# 类别: id
news_type2id_dict = {'娱乐': 0, '财经': 1, '体育': 2, '家居': 3, '教育': 4, '房

class NewsDataset(Dataset):

    def __init__(self, file_path, tokenizer: BertTokenizer, max_length=512, c
        news_type = []
        news_content = []
        news_atten_mask = []
        seq_typ_ids = []
        with open(file_path, mode='r', encoding='utf8') as f:
            for line in tqdm(f.readlines()):
                line = line.strip()
                line = line.split('\t')

                news_type.append(news_type2id_dict[line[0]])
                token_ids = tokenizer.encode(ILLEGAL_CHARACTERS_RE.sub(r'', l
                                             pad_to_max_length=True)
                news_content.append(token_ids)
                news_atten_mask.append(get_atten_mask(token_ids))
                seq_typ_ids.append(tokenizer.create_token_type_ids_from_seque

        self.label = torch.from_numpy(np.array(news_type)).unsqueeze(1).long(
        self.token_ids = torch.from_numpy(np.array(news_content)).long()
        self.seq_type_ids = torch.from_numpy(np.array(seq_typ_ids)).long()
        self.atten_masks = torch.from_numpy(np.array(news_atten_mask)).long()
```

```
        if device is not None:
            self.label = self.label.to(device)
            self.token_ids = self.token_ids.to(device)
            self.seq_type_ids = self.seq_type_ids.to(device)
            self.atten_masks = self.atten_masks.to(device)

    def __len__(self):
        return self.label.shape[0]

    def __getitem__(self, item):
        return self.label[item], self.token_ids[item], self.seq_type_ids[iter


def train(train_dataset, model: BertForSequenceClassification, optimizer: Ada
    train_sampler = RandomSampler(train_dataset)
    train_loader = DataLoader(train_dataset, sampler=train_sampler, batch_si
    model.train()
    tr_loss = 0.0
    tr_acc = 0
    global_step = 0
    if cuda:
        torch.cuda.empty_cache()
    for step, batch in tqdm(enumerate(train_loader)):
        # print(step)
        inputs = {
            'input_ids': batch[1],
            'token_type_ids': batch[2],
            'attention_mask': batch[3],
            'labels': batch[0]
        }
        outputs = model(**inputs)
        loss = outputs[0]
        # print(loss)
        logits = outputs[1]

        tr_loss += loss.item()

        model.zero_grad()
        loss.backward()
        optimizer.step()
```

```
        # 计算准确率
        _, pred = logits.max(1)
        number_corr = (pred == batch[0].view(-1)).long().sum().item()
        tr_acc += number_corr
        global_step += 1

    return tr_loss / global_step, tr_acc / len(train_dataset)


def evalate(eval_dataset, model: BertForSequenceClassification, batch_size=ba
    model.eval()
    eval_sampler = RandomSampler(eval_dataset)
    eval_loader = DataLoader(eval_dataset, sampler=eval_sampler, batch_size=b
    tr_acc = 0
    if cuda:
        torch.cuda.empty_cache()
    for step, batch in tqdm(enumerate(eval_loader)):
        inputs = {
            'input_ids': batch[1],
            'token_type_ids': batch[2],
            'attention_mask': batch[3],
            'labels': batch[0]
        }
        outputs = model(**inputs)
        # loss = outputs[0]
        logits = outputs[1]

        # tr_loss += loss.item()

        # 计算准确率
        _, pred = logits.max(1)
        number_corr = (pred == batch[0].view(-1)).long().sum().item()
        tr_acc += number_corr

    return tr_acc / len(eval_dataset)


def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
```

```
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs


if __name__ == '__main__':
    device = torch.device('cuda' if cuda else 'cpu')
    # 创建config
    config = BertConfig.from_pretrained('./model/bert_config.json', num_label
    # 创建分类器
    classifier = BertForSequenceClassification.from_pretrained('./model/pytor
    no_decay = ['bias', 'LayerNorm.weight']
    optimizer_grouped_parameters = [
        {'params': [p for n, p in classifier.named_parameters() if not any(nd
         'weight_decay': 0.0},
        {'params': [p for n, p in classifier.named_parameters() if any(nd in
         'weight_decay': 0.0}
    ]

    # 创建tokenizer
    tokenizer = BertTokenizer('./model/vocab.txt')

    optimizer = AdamW(optimizer_grouped_parameters, lr=lr, eps=1e-8)

    logger.info('create train dataset')

    train_dataset = NewsDataset('./data/cnews/cnews.train.txt', tokenizer, ma
                                device=device)
    logger.info('create eval dataset')
    eval_dataset = NewsDataset('./data/cnews/cnews.val.txt', tokenizer, max_l
                               device=device)
    best_val_acc = 0.0
    for e in range(1, epoches):
        start_time = time.time()
        train_loss, train_acc = train(train_dataset, classifier, optimizer, b
        eval_acc = evalate(eval_dataset, classifier, batch_size)
        end_time = time.time()
        epoch_mins, epoch_secs = epoch_time(start_time, end_time)
        logger.info('Epoch: {:02} | Time: {}m {}s'.format(e, epoch_mins, epoc
        logger.info(
            'Train Loss: {:.6f} | Train Acc: {:.6f} | Eval Acc: {:.6f}'.forma
```

```
        if eval_acc > best_val_acc:
            best_val_acc = eval_acc
            torch.save(classifier.state_dict(), './fine_tune_model/model_{}'
```

```
#predict.py
news_type_id_dict = {'娱乐': 0, '财经': 1, '体育': 2, '家居': 3, '教育': 4, '房产

news_id_type_dict = {v: k for k, v in news_type_id_dict.items()}

def get_atten_mask(tokens_ids, pad_index=0):
    return list(map(lambda x: 1 if x != pad_index else 0, tokens_ids))


config = BertConfig.from_pretrained('./model/bert_config.json', num_labels=ler

classifier = BertForSequenceClassification.from_pretrained('./fine_tune_model/
classifier.eval()

tokenizer = BertTokenizer('./model/vocab.txt')
index = 0
def predict(text):
    global index
    text = str(text).strip()
    token_ids = tokenizer.encode(ILLEGAL_CHARACTERS_RE.sub(r'', text), max_ler
                                 pad_to_max_length=True)
    token_mask = get_atten_mask(token_ids)

    token_segment_type = tokenizer.create_token_type_ids_from_sequences(token_

    token_ids = torch.LongTensor(token_ids).unsqueeze(0)
    token_mask = torch.LongTensor(token_mask).unsqueeze(0)
    token_segment_type = torch.LongTensor(token_segment_type).unsqueeze(0)

    inputs = {
        'input_ids': token_ids,
        'token_type_ids': token_segment_type,
        'attention_mask': token_mask,
        # 'labels': batch[0]
    }
```

```
    logits = classifier(**inputs)
    _, predict = logits[0].max(1)
    # print(str(index) + news_id_type_dict[predict.item()])
    index += 1
    return news_id_type_dict[predict.item()]


if __name__ == '__main__':

    news =
    " 对于我国的科技巨头华为而言，2019年注定是不平凡的一年，由于在5G领域遥遥领先于其他
    但是华为并没有因此而一蹶不振，而是亮出了自己的一张又一张“底牌”，随着麒麟处理器、海思
    print(predict(news))
```

04

bert词典扩充方法

[...] if you want to add more vocab you can either:
(a) Just replace the "[unusedX]" tokens with your vocabulary. Since these were not used they are effectively randomly initialized.
(b) Append it to the end of the vocab, and write a script which generates a new checkpoint that is identical to the pre-trained checkpoint, but but with a bigger vocab where the new embeddings are randomly initialized (for initialized we used tf.truncated_normal_initializer(stddev=0.02)). This will likely require mucking around with some tf.concat() and tf.assign() calls.
(google-research/bert#9)

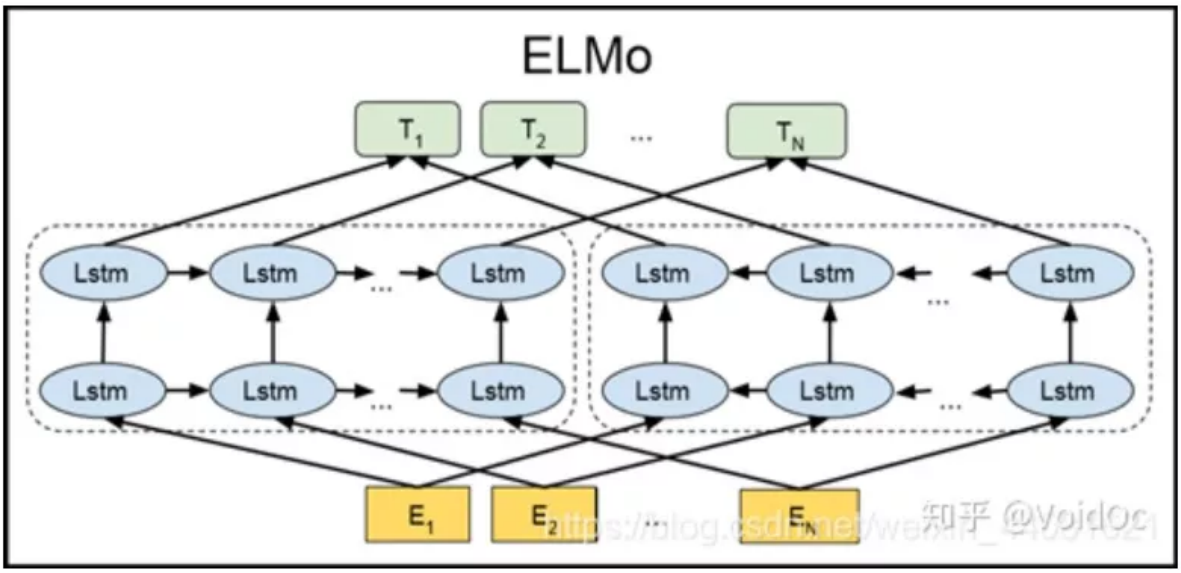
05

BERT家族拓展

5.1 ELMo

ELMo论文解读--原理、结构及应用 [5]

Encoder是双向LSTM



5.2 ALBERT

英文版：https://github.com/google-research/ALBERT

中文版：https://github.com/brightmart/albert_zh

4.3 TinyBert

https://zhuanlan.zhihu.com/p/94359189

引用：

1. https://www.jianshu.com/p/80a809147491
2. https://juejin.im/post/6844903781696536589
3. https://cloud.tencent.com/developer/article/1465005
4. https://prohiryu.github.io/2019/04/10/bert-chinese-ner/
5. https://blog.csdn.net/weixin_44081621/article/details/86649821
6.《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》-- Google Research
7. https://blog.csdn.net/dendi_hust/article/details/103734611
8. https://blog.csdn.net/dendi_hust/article/details/103690354
9. https://zhuanlan.zhihu.com/p/102208639
10. https://www.cnblogs.com/gczr/p/11785930.html

END