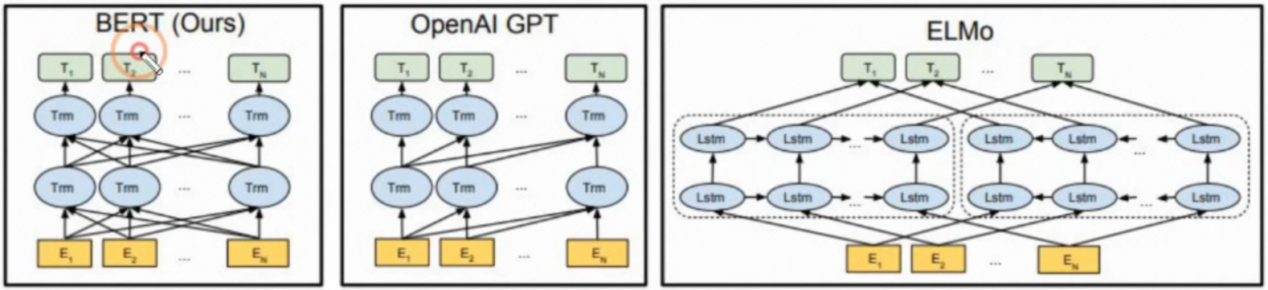
1. **初识BERT模型**
2. 基本认知

此前，由双向LSTM构建而成的单向语言模型ELMo，虽然兼顾了文本的上下文信息，但是依赖RNN架构的顺序执行方式缺乏了并行性。由Transformer中的DecoderBlock串行构建而成的GPT-1，虽然能够通过自监督学习的方式更好地完成文本生成任务，但是DecoderBlock的单向注意力机制，导致模型在处理文本时只能看到前面的内容，无法利用后文信息。此外，GPT-1由于自回归生成方式的限制，每个生成的词都依赖于前面所有已经生成的词，极大地限制了长文本的生成速度。

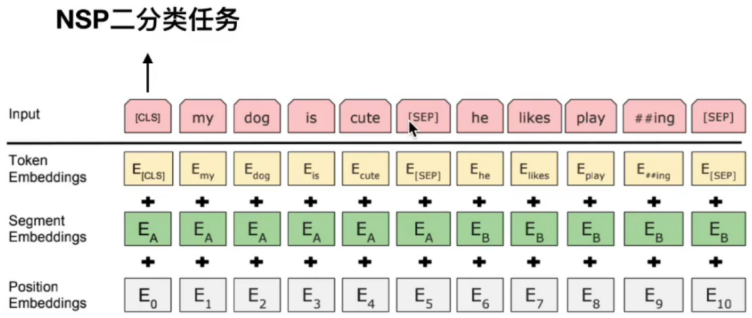


Google于2018年提出的BERT（Bidirectional Encoder Representations from Transformers）预训练语言模型，由Transformer中的EncoderBlock串行堆叠而成，主要包含Pretraining及Fine-Tuning两个过程，前者用于提高模型的文本表征能力，后者则通过轻量化的微调训练来适应具体的任务。相较于前述的单向语言模型，BERT模型在预训练的过程中，主要引入了MLM（Masked Language Model）任务及NSP（Next Sentence Prediction）任务,通过最小化MLM任务及NSP任务总体损失的方式，极大地提高了模型的文本表征能力。

显而易见的是，针对于各种类型的自然语言处理任务，无需大幅度地调整BERT模型的网络结构，仅需在预训练好的BERT模型基础上添加一些额外的层或结构（如：浅层感知机）以完成适应性的微调训练，即可获得极好的实验效果。

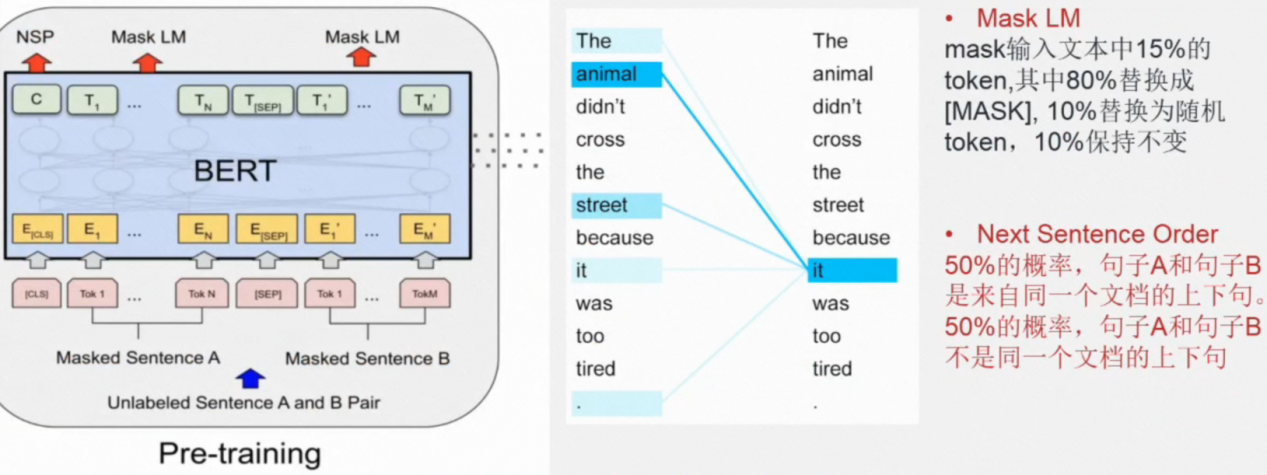
1. 模型架构

Base BERT模型主要由12层EncoderBlock串行堆叠而成，并主要依赖EncoderBlocks完成输入序列的文本表征工作。其中，EncoderBlock的内部结构雷同于Transformer模型中的编码块结构，此处省略解释。区别在于，BERT模型的嵌入层需要完成词、分句及位置嵌入工作。如下图所示，每个输入序列由两个句子拼接而成（至少得有一个句子），每个句子的末尾需要插入句子分隔标记[SEP]。此外，每个输入序列的开头还需要额外插入一个标记[CLS]，用于考虑全局上下文信息以完成分类任务。对于词嵌入Token Embeddings而言，可以构建一个随机初始化参数的线性层，完成输入序列中token的语义嵌入表示。对于分句嵌入Segment Embeddigns而言，第一个[SEP]标记及其以前的token使用0表示，第一个[SEP]标记后的token使用1表示即可，即：Ea=0，Eb=1。对于位置嵌入Posittion Embedding而言，亦需要构建一个随机初始化参数的线性层，用于完成表示最大输入序列长度max\_len中不同序列位置的嵌入表征学习。最后，求和三种类型的嵌入张量，即可获得输入序列所有token的复合嵌入表示。



1. 预训练BERT

预训练BERT模型，需要完成掩码语言模型（Masked Language Model, MLM）及下一个句子预测（Next Sentence Prediction, NSP）任务。



对于MLM任务而言，本质上是希望模型尽可能地使用上下文信息，准确地完成输入序列中被掩码词元的完形填空，属于自监督学习。其中，10%的掩码对象替换为随机的token，是为了通过引入噪声的方式，使得模型被迫更多地依赖于上下文信息完成词元表征，从而确保学习到词汇在不同上下文中的不同含义。10%的掩码对象保持不变，则是为了更好地学习词汇在不同语境下的固定含义，以最大程度提高BERT模型的文本表征能力。值得注意的是，MLM任务存在同一个输入序列中多个[MASK]之间相互独立的缺点，这可能会导致模型很难学习利用掩码词元之间的语境信息。

对于NSP任务而言，主要用于判断输入序列中的两个句子是否相邻，即：判断输入序列中后面的句子是否是前面句子的下一句话。NSP任务有助于模型理解文本的整体结构及上下文关系。相较于MLM任务，NSP任务则更加关注句子级别的上下文理解，从而促使BERT模型更好地学习句子间的逻辑关系。

1. 微调BERT

完成BERT模型的预训练进程以后，需要根据具体的下游任务完成BERT的微调工作，比如：文本分类、命名实体识别及问答任务等。首先，选择同任务相匹配的数据集，以完成输入序列集合的重构载入。其次，根据任务及输入序列集合的实际情况，确定微调训练的具体策略（学习率、迭代次数及批量大小等超参）及专门的输出模块。然后，加载BERT模型预训练好的权重作为训练的初始参数，依赖于专门用于微调训练的数据集，通过优化函数完成模型可学习参数的迭代更新。最后，在微调训练结束后使用验证/测试集完成模型的评估，以完成模型进一步的优化调整，比如：数据处理、超参及输出模块的网络结构等。

如何提升BERT模型在处理具体任务时的表现？

以关于网易云音乐评论的情感分类任务为例。首先，可以使用海量通用语料预训练一遍BERT模型，或者导入预训练好的中文谷歌BERT。然后，使用相同领域的海量文本（比如：音乐评论文本）再预训练一遍前述获得的BERT模型。最后，在前述获得的BERT模型基础上，使用任务相关的高质量小数据集（有些音乐评论不太适用于情感分类任务，直接爬取而来的音乐评论通常质量不高），完成预训练及微调工作。

**二、预训练BERT模型**

1. 数据准备

为了预训练自定义的BERT模型，完成遮蔽语言模型（MLM）及下一个句子预测(NSP)任务，特别选定了WikiText-2-v1数据集来构建训练数据迭代器。具体过程如下：

由于NSP任务需要连续的两个句子组合构建具有IsNext=True标签的输入序列，故此在分行读取全局文档中的段落时，需要过滤掉仅包含一个句子的行段落。此后，还需要拆分每个行段落中的句子，将同一个行段落中的句子依序放置到对应的内部列表中。由此，可获得分段的句子列表paragraphs，即：[[parag1\_sen1, parag1\_sen2], [parag2\_sen1, parag2\_sen2], ...]。

基于全局文档构建词典Vocab，需要遍历paragraphs中所有句子的单词词元，统计记录所有的非低频非重词元，并在词典中额外记录特殊标记<pad>、<mask>、<cls>、<sep>。此后，可依赖词典完成段落paragraphs[i]中所有句子paragraphs[i][j]的词元索引化表示。

循环遍历每个段落列表paragraphs[i]，有1/2的概率选择将某个段落中的两个句子拼接在一起（需要保证两个句子的词元数总和小于max\_len-3），并认定拼接后序列的标签为IsNext=True。否则，选择从paragraphs中随机抽取一个句子和当前段落中的句子paragi\_sen1拼接在一起，认定序列标签为IsNext=False。在拼接两个句子的词元时，需要在第一个词元前插入标记符<cls>，在两个句子的末尾都插入标记符<sep>，以获得最终的输入序列tokens。此外，还需要构建一个同tokens等长的词元归属序列segments，通过0-1值表达tokens中不同位置上的词元具体属于哪个句子。由此，可获得半成品的输入序列集examples，即：[(tokens1,segments1,IsNext),...]。

考虑到MLM任务是基于掩码的自监督学习，故此需要遮蔽每个tokens序列中15%的词元，需要遮蔽的词元中：80%被替换为标记符<mask>，10%被随机替换为其它词元索引，10%不动。针对examples中所有的tokens完成遮蔽操作后，需要在每个序列元组中引入新的列表变量pred\_positions及mlm\_labels，前者表示被遮蔽词元在对应tokens序列中的位置，后者表示被遮蔽词元的原本索引值。值得注意的是，tokens序列中的标记词元不应被遮蔽。由此，半成品examples为：**[(tokens1, pred\_positions1, mlm\_labels1, segments1, IsNext),...]。**

尽管所有tokens序列的长度都小于max\_len，但是tokens序列并不等长。由此，需要使用标记<pad>对应的词典索引值，将所有tokens序列的长度填充至max\_len，并使用0值将所有segments列表的长度同样填充至max\_len。此外，同样需要使用0值，将pred\_positions及mlm\_labels的长度填充至0.15\*max\_len。由此，即可获得最终的examples。

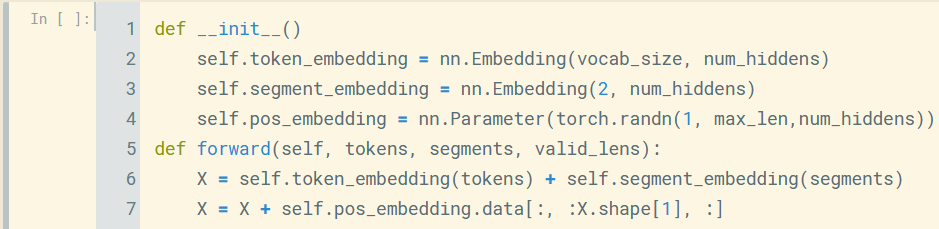
最后，需要使用torch.utils.data.DataLoader()方法，指定train\_set=examples及batch\_size等参数，从而获得训练数据迭代对象train\_iter。

1. 自定义BERT模型

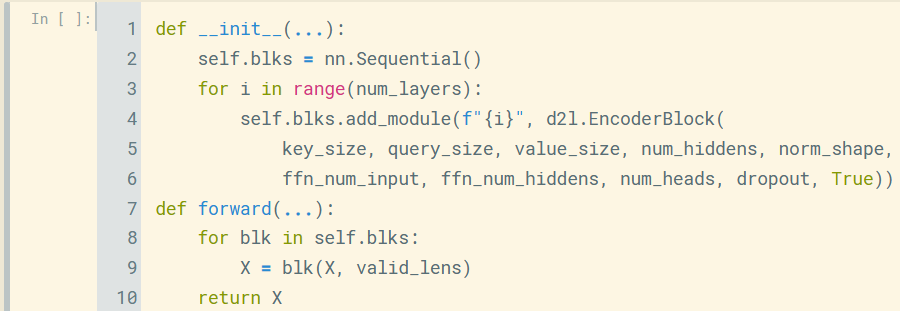
BERT模型主要依赖多个串行堆叠的Encoder，选定MLM及NSP任务作为学习目标，完成了基于训练数据迭代对象train\_iter的文本表征学习。由此，若需要预训练BERT模型，应实现BERTEncoder()、MaskLM()及NextSentencePred()等功能模块。

1. BERTEncoder()

分批次解包而来的tokens序列集合X，其维度信息为(batch\_size, max\_len, vocab\_size)。X首先需要完成词元、段及位置嵌入处理，伪代码如下图所示。其中，词元嵌入token\_embed将X的特征维从vocab\_size映射为嵌入维数num\_hiddens；段嵌入segment\_embed将segments集合的特征维数2同样映射为嵌入维数num\_hiddens。对于可学习的位置嵌入层而言，由于不同的tokens可共用一套位置嵌入矩阵，故此其权重维度为(1, max\_len, num\_hiddens)即可。由此，三个不同嵌入操作生成的张量在维度上具有一致性，从而保证了将三者直接加和的操作能正确获得引入三种嵌入信息的输入张量X，维度为(batch\_size, max\_len, num\_hiddens)。



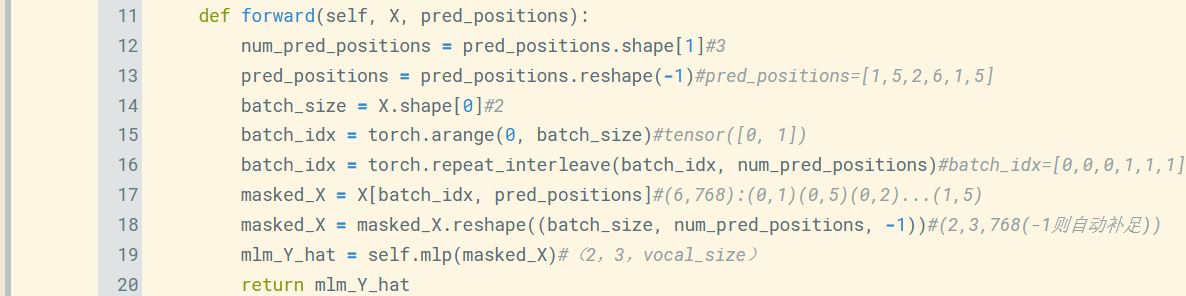
串行堆叠Encoder的实现方式，如下图伪代码所示。其中，依赖nn.Sequential()方法完成自定义EncoderBlock的串联装填，并通过num\_layers控制EncoderBlock的数量。而在forward中，通过循环完成引入嵌入信息的tokens序列集合X在串联EncoderBlocks中的传递重构，从而不断优化学习到的文本表征。



1. MaskLM()

MaskLM()模块接收BERTEncoder()返回的文本表征张量encoded\_X及分批次解包而来的pred\_positions集合，通过一个自定义的浅层前馈网络模块完成了掩蔽语言模型LML任务，即：预测每个tokens序列中被遮蔽的词元具体是什么单词。

伪代码如下图所示。若假设batch\_size=2、每个tokens序列中遮蔽的词元数为3，则需通过torch.repeat\_interleave方法完成batch\_idx的构造，即：[0,0,0,1,1,1]。此后，依赖batch\_idx及pred\_positions的联级索引，从encoded\_X中获取遮蔽位置上的词嵌入向量，并重组为masked\_X张量，维度为(2,3,768)。最后，将masked\_X传入自定义的浅层前馈网络模块mlp，以完成当前批次下所有tokens序列中被遮蔽词元的预测工作。其中，mlp可以看作一个引入标准化操作的单层感知机模型，用于将masked\_X张量的特征维数从num\_hiddens映射到vocab\_size，从而获得预测词元在词汇表Vocab中对应的索引值。



1. NextSentencePred()

NextSentencePred()模块同样接收BERTEncoder()返回的文本表征张量encoded\_X，它仅需通过一个简单的线性层就完成了下一个句子预测NSP任务，即：判断tokens序列中的两个句子是否在原文中相邻。

具体而言，首先需要依赖torch.flatten方法将encoded\_X从第二个维度开始展平，即：从(batch\_size, max\_len, embed\_size)维展平为(batch\_size, max\_len\*embed\_size)维。然后，通过一个简单的线性层nn.Linear(max\_len\*embed\_size, 2)，即可获得二分类NSP任务的结果nsp\_Y\_hat，其维度为(batch\_size, 2)。

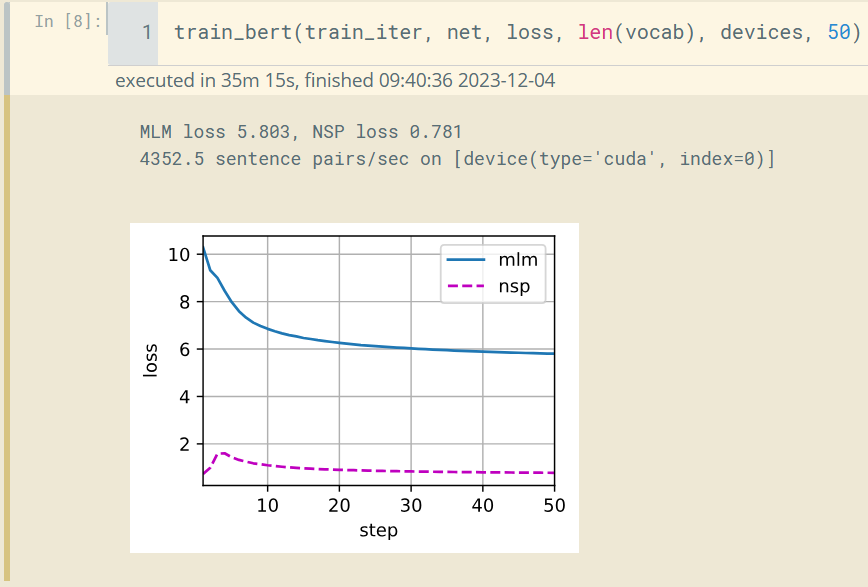
**小结：**

值得注意的是，在预训练BERT模型的过程中，主要通过最小化自监督学习MLM任务及监督学习NSP任务的总体训练损失，进一步调整EncoderBlocks中的可学习参数，从而使得BERT模型获得更强的文本表征能力。具体而言，MLM任务通过随机遮蔽输入序列中部分词元的方式，要求模型依赖encoded\_X完成遮蔽词元的自监督学习，从而迫使BERT模型能够更有效地捕捉句子中的语法、语义及词元级别的上下文关联信息。NSP任务同样依赖encoded\_X完成一个关于判定tokens序列中两个句子是否在原文中相邻的二分类任务，迫使BERT模型学习文本句子级别的逻辑关系。此外，在预训练BERT模型的过程中，通过BERTEncoder()模块处理批次输入序列以获取更好的文本表征张量encoded\_X，需要花费最多的计算时间。而在后续的MLM任务及NSP任务中，都只需要通过一个极简单的感知机模块或者一个浅层分类器处理encoded\_X，即可完成对应的任务，并不占用太多预训练BERT模型所需的整体时间。

1. 预训练BERT模型【代码详见：李沐《动手学深度学习（PyTorch版）》P708】

完成了数据加载及模型定义后，还需要定义目标函数。如前所述，预训练BERT模型的优化目标是：最小化MLM任务及NSP任务的总体损失。故此，可以直接将MLM任务的平均损失同NSP任务的损失相加即可获得总体损失。对于MLM任务而言，使用交叉熵损失函数计算出分批次的损失值后，需要将损失同权重矩阵mlm\_weights\_X相乘，从而消除预测填充词元的损失。对于NSP任务而言，亦使用二元交叉熵损失函数直接计算损失。

此后，可调用torch.optim.Adam方法完成优化函数的引用，调用d2l.Accumulator方法完成迭代训练过程中关于损失变化的可视化输出。预训练可视化结果，如下图所示：

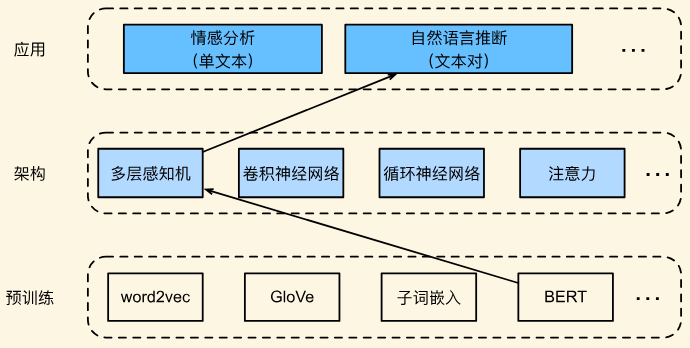


预训练BERT模型所涉及的部分参数如下表所示：

|  |  |  |
| --- | --- | --- |
| 参数名 | 取值 | 解释 |
| batch\_size | 512 | 批次大小 |
| max\_len | 64 | 构造后的输入序列最大长度 |
| vocab\_size | 10000 | 词汇表尺寸 |
| num\_hiddens | 128 | 隐藏层结点数/嵌入层维数 |
| ffn\_num\_input | 128 | 前馈网络输入层结点数 |
| ffn\_num\_hiddens | 256 | 前馈网络隐藏层结点数 |
| num\_heads | 2 | 注意力头数 |
| num\_layers | 2 | Encoder数量 |

**三、自然语言推断任务：微调BERT模型**

自然语言推断任务，本质上是**序列级别的文本对分类问题**，主要研究前提文本与假设文本之间属于哪种逻辑关系（蕴含、矛盾或中性）。其中，蕴含关系指假设可以从前提中推断出来，比如：从前提“小米和小王是同桌”可以推断出假设“小米和小王是同学”；矛盾关系指假设的否定可以从前提中推断出来，比如：前提“小王在玩游戏”可以推断出假设“小王在睡觉”的否定；前提及假设间的其它情况，则一律归结为中性逻辑关系。本节实验在预训练BERT的基础上，结合采用感知机架构的分类器展开微调训练，以完成自然语言推断任务。

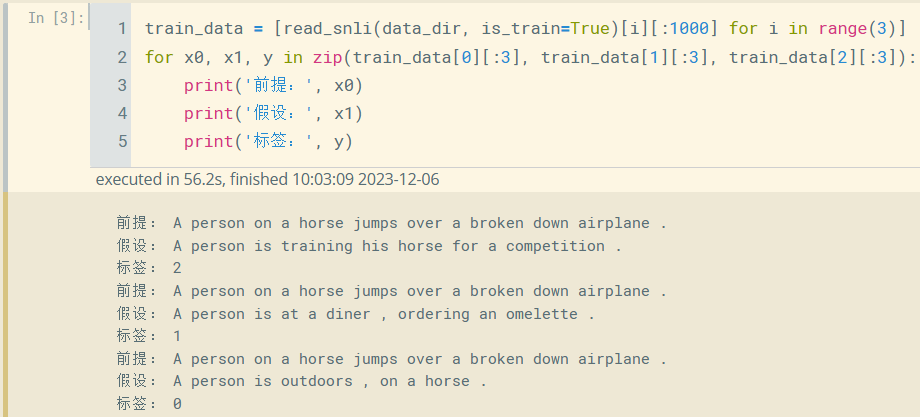


1. 加载预训练的BERT模型

下载bert.small，获得预训练BERT模型的pretrained.params及vocab.json文件。编写load\_pretrained\_model函数，主要接收如下参数：num\_hiddens、ffn\_num\_hiddens、num\_heads、num\_layers、max\_len，用于控制BERT模型的网络架构。具体步骤如下：首先，根据vocab.json文件完成预定义词典的加载；然后，结合传入的实参，定义BERT模型的网络架构；最后，针对模型的实例化对象，引用pretrained.params文件完成预训练BERT模型的参数载入。

1. 数据准备

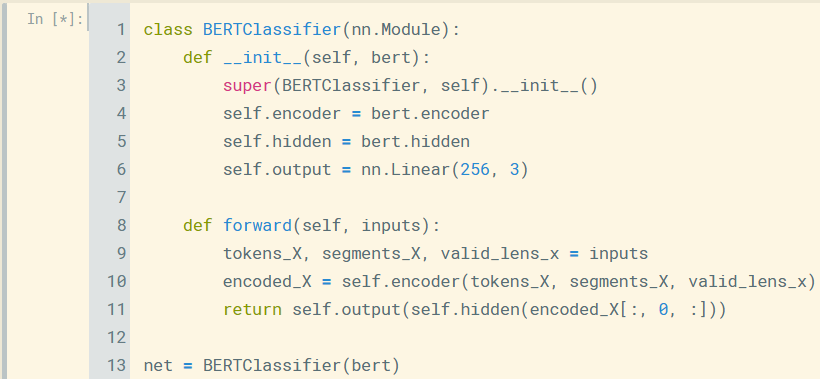
本次实验选用的数据集：斯坦福自然语言推断语料库（Stanford Natural Language Inference, SNLI），含有55余万个带标签的英文句子对。考虑到任务性质，仅需获取SNLI数据集中每个文本对的前提文本、假设文本及标签即可，如下图所示。此外，本次实验使用549367条文本对用于训练，9824条文本对用于测试，且不同类别的文本对数量相对均衡。



针对于SNLI训练集及测试集。首先，需将对应的前提文本与假设文本拼接成一个定长的输入序列：不同文本的尾部需要添加[SEP]标记，前提文本的首部需要添加[CLS]标记；输入序列的总长度控制为max\_len，若太短则插入[PAD]标记填充，若太长则截断。然后，根据vocab.json文件加载而来的词典，完成所有定长输入序列词元的数值索引化表示。最后，将训练及测试输入序列集，划分为若干个批次，以完成训练及测试数据迭代对象的定义。

1. 微调训练BERT模型

使用预训练的BERT完成自然语言推断任务，仅需定义一个基于多层感知机的分类器BERTClassifier()，用于接收BERT模型最终输出的[CLS] token，经过两层全连接层的非线性映射（self.hidden(cls\_token\_len, hidden)，self.output(hidden, 3)），即可获得最终的分类结果。具体实现方式，如下图所示：



值得注意的是，在使用SNLI train\_iter微调训练BERT及分类器BERTClassifier的过程中，不再依赖BERT模型中MLM及NSP任务的损失来迭代更新整体的网络参数，MaskLM及NextSentencePred中采用的多层感知机的参数将被冻结。由此，针对于自然语言推断任务的整体微调训练（分类器中的可学习参数是从头开始训练的，BERT中的可学习参数是在预训练的基础上开始微调训练的），将依赖于分类器预测结果同输入序列标签间的差异来计算损失，以完成所有可学习参数的迭代更新工作。

微调训练及测试的可视化结果，如下图所示。可知，在训练集上的最终损失为0.519，在训练集上的最终准确率为0.791，在测试集上的最终准确率为0.783。

