**语音识别ASR**

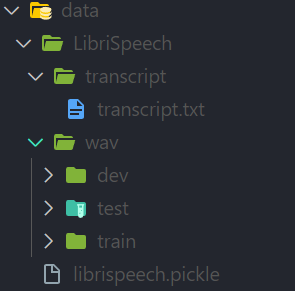
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **学号** | **专业班级** | **姓名** | **性别** |
| 1 | 3200105446 | 计科2006 | 董佳昕 | 男 |
| 2 | 3200105445 | 计科2001 | 郭雨征 | 男 |
| 3 | 3200300357 |  | 黄程骏 | 男 |
| 4 | 3200300849 |  | RAYMOND | 男 |

## Project Introduction

本次大作业将实现一个语音识别或者Automatic Speech Recognition（ASR）。ASR的目标是将口头语言转换为文本形式，它是自然语言处理（Natural Language Processing）领域中的一个重要研究方向。ASR系统接收来自语音输入源（如麦克风、电话或语音文件）的声音信号，并将其转化为相应的文本输出。

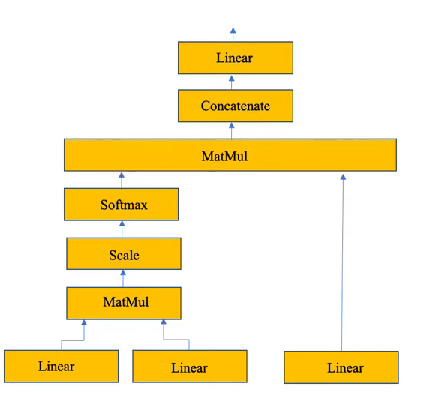
我们使用*Pytorch*的机器学习模块来实现基于*Transformer*的ASR，而且用*LibriSpeech*作为我们的数据源。*LibriSpeech*来自*LibriVox*项目的有声读物，是一个由*Vassil Panayotov*在*Daniel Povey*经过精心分段和对齐处理下准备的约1000小时16kHz英语口语语料库。但本作业出于训练难度考虑，只需要用其中的100小时。

1. **Technical Details**
   1. **数据预处理**

本作业的所有数据被放在LibriSpeech文件夹里，一共有33862数据包括28539训练数据集、2703验证数据集、和2620测试数据集。所有语音以*.flac*为形式且放在LibriSpeech里的wav文件夹。另外文件夹*transcript*保存所有语音的标签（内容形式为 文件名+空格+文字）。

在预处理中，我们将把所有语音的路径和向量放在一个字典。接下来我们根据该字典创建一个新*pickle*文件。Pickle文件是指 Python 编程语言中使用的一种文件格式，用于将复杂的数据结构（例如列表、字典和对象）转换为可以保存到文件或通过网络传输的字节流。机器学习中通常用于存储预处理的数据。

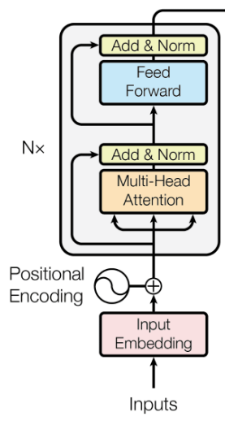
* 1. **自注意力Self Attention、多头注意力Multihead Attention**

自注意力是自然语言处理中使用的一种机制，特别是在基于 *Transformer* 的模型领域。自注意力把输入序列（例如句子或文档）转换为三种不同的表示形式：查询（query）、键（key）和值（value），使得模型能够关注输入序列的不同部分以捕获其自身元素之间的依赖关系。它将计算序列中每个元素相对于其他元素的重要性或相关性。另外，多头注意力是自注意力机制的扩展。它允许模型通过关注投影查询、键和值的多个子空间来捕获输入序列中不同类型的依赖关系。以下是多头注意力的具体过程：

1. 我们将自定义*MultiHeadAttention*类来实现多头注意力。该类的第一层由查询（Q）、键（K）和值（V）三个线性层组成。线性层由一堆完全连接的神经元组成，用于将向量投影到较低的维度。这里我们将Q，K，V这三个矩阵向下投影，维数为*dQ / dV*，并行投影8次（头数），线性层的权重初始化服从正态分布。
2. Q和K进行矩阵相乘，以找到它们之间的相似度，把结果称*attn。*
3. 进行scale，把*attn*除以其维度的平方根。
4. 然后我们使用mask（如果有），把*attn*中对应位置设为-inf*。*它有助于确保模型只关注输入序列的相关部分。
5. 对*attn*进行Softmax，把之前的结果归一化到(0,1)之间。此处把这步的结果称为*output。*
6. *Output*与V进行矩阵相乘，把结果导入到连接层。该层权重初始化满足*Xavier Uniform*分布。
7. 最后把*output*进行连接和归一化。
   1. **编码器（Encoder）**

编码器负责处理输入序列并捕获其表示，因此其输入为一个已经过对齐的序列数据，然后输出为其对应的向量。这个概念类似于词向量，但现在我们将根据某个元素在序列的位置和意义来分配向量。编码器由多头自注意力机制（Multi-head Self Attention）和位置前馈神经网络（Position-wise Feed Forward Network）两个子层组成。以下是编码器的过程：

1. 计算padding产生的mask。Mask是一个维数等于输入序列长度的向量，将非padding位置的mask设置为0，其他为1。
2. 计算自注意力的mask，把由于padding产生的mask最后一位删掉，并对其取反，再进行扩展得到*N × Ti × Te*的自注意力mask矩阵，其中*N*是输入序列大小，*Ti*是输入序列长度，*Te*是拓展后的序列长度。
3. 将经过对齐后的输入序列进行Embedding和位置编码（Positional Encoding）。我们将使用Linear和Layer Norm化来替换Embedding，而且使用sin和cos函数来实现位置编码函数：

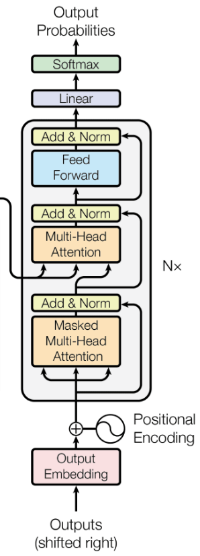
其中*d\_model*是embedding向量的维数，结果是一个张量。 使用sin/cos的优点是泛化能力较强，具有对称性和唯一性。

1. 再把经过位置编码的张量进行Dropout，把结果称*enc\_output*。
2. 把*enc\_output*导入到6个编码器层。编码器由多头注意力层与前馈网络层组成。
3. 每次进入编码器层，将*enc\_output*经过多头注意力层，然后使用由于padding产生的mask。
4. 把上述结果经过前馈网络层（Position-wise Feed Forward Network）。FFN有两层，以及激活函数为ReLU：

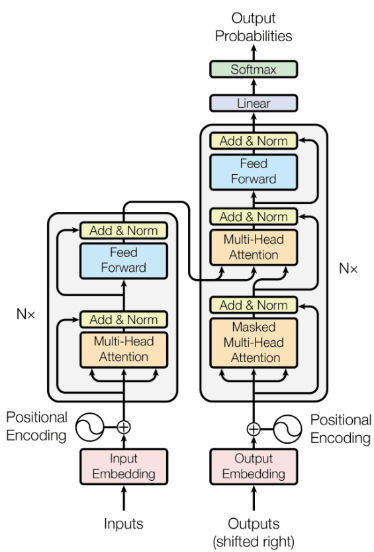
其中*x*是输入；*W1、W2* 是第1、2层权重矩阵；*b1、b2* 是第1、2层偏见。最后再使用padding产生的mask。

* 1. 解码器（Decoder）

解码器将获取编码器的输出作为输入，并逐步生成预测的输出序列。类似于编码器，它由一堆相同的层组成，但有一个额外的子层，称为掩码多头自注意力（Masked Multi-head Attention）。在训练过程中，掩码多头自注意力层会阻止模型关注输出序列中的未来位置。这确保了输出序列中的每个位置仅基于先前生成的位置生成。以下是编码器的过程：

1. 预处理输入序列，添加*<sos>*到decoder输入中的每个句子，且添加*<eos>*到decoder输出标签的每个句子。然后将输入中的空白位置填充为 *<eos>*，而且将输出中的空白位置填充为-1。
2. 计算由padding产生的mask。过程跟上述编码器一样，但是现在padding mask增加了padding index（*<eos>*位置）也需要进行mask。
3. 计算自注意力的mask（由于时间产生的mask + 由于padding产生的mask）。由于时间产生的mask阵列为上三角矩阵，再把它进行扩展为*b × ls × ls*，b为批量大小，*ls*为输入序列维度。由于padding产生的mask是屏蔽K序列在位置为padding index进行mask成为1，然后将mask扩展为*b × lq × lk*， *b*是批量大小，*lq*是Q序列大小，*lk*是K序列大小。计算这两个mask是用于屏蔽未来时刻的信息与padding位置。
4. 计算编码器-解码器注意力的mask。过程相当于上述编码器计算自注意力，但是现在输入为编码器的输出，输入序列长度是编码器输入的长度。
5. 把输入通过Embedding和位置编码。位置编码函数相当于上述编码器的PE函数。但对于Embedding我们将用*torch.nn*的Embedding类。该类通常用于存储词嵌入并使用索引检索它们，输入是索引列表，输出是相应的词嵌入。
6. 再把经过位置编码的张量进行Dropout，把结果称*dec\_output*。
7. 把*dec\_output*导入到6个解码器层。解码器由掩码多头注意力层、多头注意力层与前馈网络层组成。自注意力层使用上述讲过的*MultiHeadAttention*类。
8. *dec\_output*作为掩码自注意力层的Q、K、V输入，并使用已计算的自注意力mask。再把结果乘以padding产生的mask。
9. 把前步骤的输出作为第二自注意力层的Q输入，把以前编码器输出结果作为解码器自注意力层的K、V输入，并使用已计算的编码器-解码器注意力的mask。最后再把结果乘以由padding产生的mask。这部分将分析编码器输出与解码器输入的相似度。
10. 把前步骤的输出通过前馈连接层FFN，过程相当于编码器中的FFN，再把输出乘以由padding产生的mask。
11. 把前步骤的输出经过Softmax层，得到当前时刻的输出结果。
12. 从输出结果得到前*k*个*scores*以及对应的预测结果。
13. 将当前时刻的预测结果与之前的预测结果拼接起来。
14. 根据*score*对预测结果进行排序，然后转换成list。最后把list返回给Transformer。
    1. **Transformer**

整个Transformer由编码器层与解码器组成。以下给大家总结Transformer的工作流程：

1. 对已经过对齐的输入序列数据（语音特征数据）进行Embedding和位置编码。Embedding映射该数据到相应欧式空间的向量，两个意义相近的数据具有相近的向量。位置编码将考虑元素在序列中的位置会影响其所蕴含的意义，因此又给一个能够表达上下文的向量（一般用上述sin和cos函数得到这种向量）。
2. 把结果给多头注意力层，作为该层的Q、K、V输入序列。注意力将告诉我们序列中哪个部分需得到更多注意。
3. 再把结果给前馈网络FFN，得到相应的编码器输出。
4. 训练时把语音相应的文字输出导入到解码器，预测时把之前的预测输出导入到解码器。进入解码器之前也得经过Embedding与位置编码。
5. 把第四步骤的结果给解码器的掩码多头注意力，作为该层的Q、K、V输入序列。
6. 把第五步骤的结果给下次多头注意力层，作为该层的Q序列。另外K和V序列由编码器输出为输入。
7. 把第六步骤的结果给前馈网络FFN，再把输出经过Softmax层。
8. 获取当前*score* 前*k*个最高的结果并进行排序作为当前最后返回值。
   1. **训练阶段**

训练时，我们将使用一个模块***librosa***加载语音数据，并获取相应的特征。以下解释训练的整个过程：

1. 定义新Transformer模型，创建新编码器、解码器、以及其优化器。

* 编码器超参数设置：输入维数=320； 编码器层数=6；多头注意力投影次数=8；Q、K、V维度=64；词向量维度=512；前馈网络维度=2048；dropout概率=0.1；位置编码最大长度=5000。
* 解码器超参数设置：*start of sequence id*=0; *start of sequence id*=1; 目标语言词表的大小=4335；词向量维数=512；解码器层数=6；多头注意力投影次数=8；K、Q、V维度=64；前馈网络维度=2048；Dropout概率=0.1；位置编码最大长度=5000。
* 优化器使用torch中的Adam优化器，学习率为0.001。

如果已创建过模型，不需要重新定义，直接加载上一轮的*checkpoint*。

1. 加载训练和验证数据集，由*LibriSpeechDataset*类来实现。

* 类中调用一个函数*extract\_feature*加载语音数据，并进行声学特征提取（Acoustic Feature Extraction）。
* 这函数使用*librosa.load*，将音频文件加载为浮点时​​间序列，并自动重新采样到给定的速率(16kHz)。
* 对20dB以下的信号进行修剪。
* 使用MFCC（Mel Frequency Cepstral Coefficients）提取特征。梅尔频率是指近似人类听觉系统对不同频率的响应的音调或频率的感知尺度。简单来说，人耳对较低频率的音高变化更敏感。频率越高，音高变化的感知越小。MFCC将表达频率和音高的对数关系，以紧凑且有区别的方式表示音频信号的频谱特征。
* 再提取MFCC的delta特征，通过表示 MFCC 系数随时的变化率或梯度来捕获音频信号的时间动态。
* 使用Cepstral Mean and Variance Normalization （CMVN）把音频特征的均值与方差进行标准化，以确保特征具有一致的统计特性，可以提高各种音频处理任务的准确性。*Extract\_feature*的返回值是经过CMVN的特征。

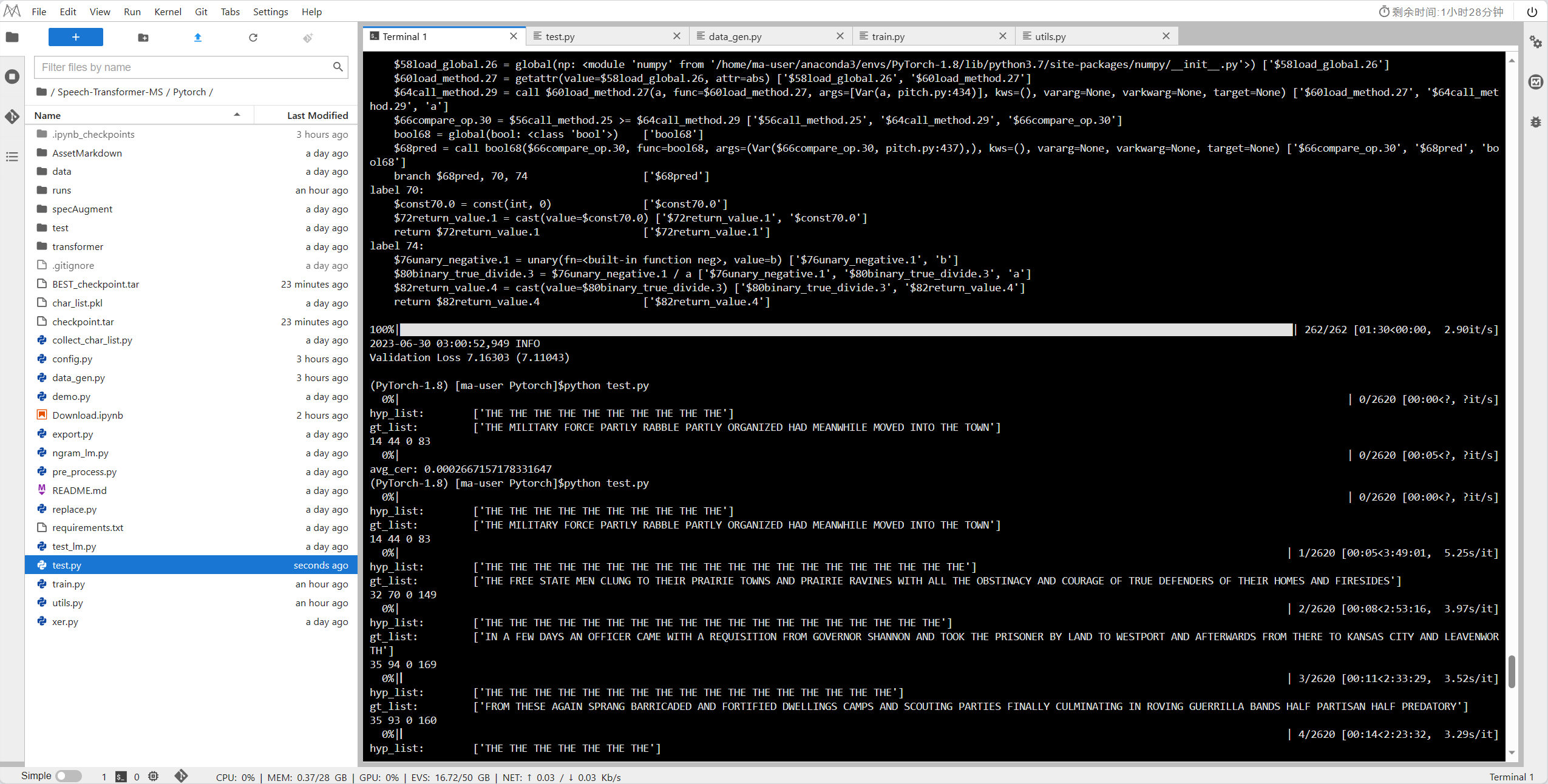
1. 把训练与验证数据的特征进行对齐，再把数据导入到torch的DataLoader类。该类提供了一种在训练期间加载和迭代数据集的有效方法。批量大小设置为32。
2. 开始训练，由于可用训练时间有限，我们仅训练了10个 epoch。

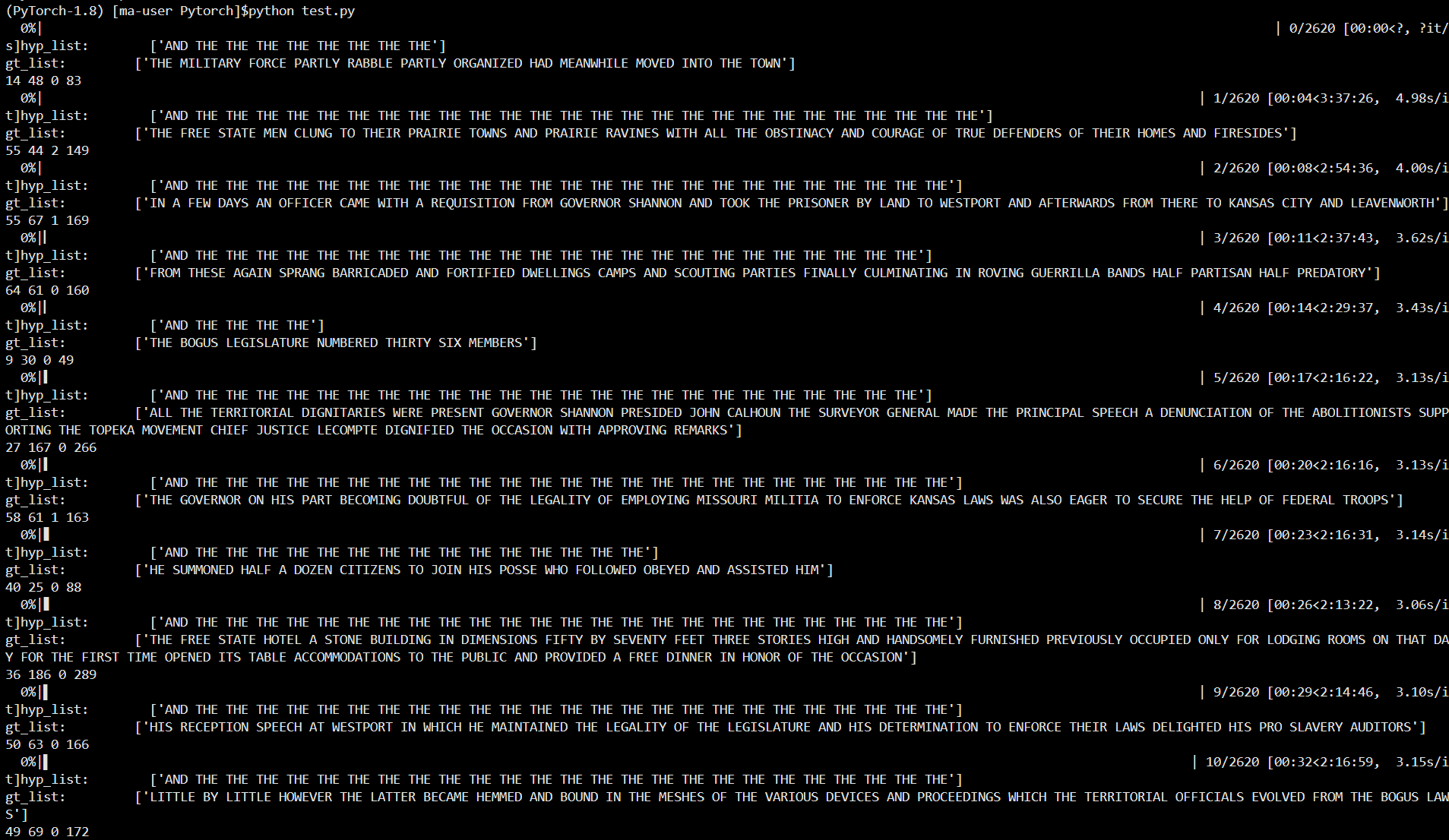
* 每一轮将把训练数据导入到Transformer模型，让模型预测相应的输出。每次会导入一个batch的数据，包含32个句子。
* 计算Transformer的输出与实际标签的交叉熵损失函数。
* 据交叉熵的损失值进行反向传播（Back Propagation）。因为torch累积若干反向传播的梯度，因此对于每批量数据进行反向传播之前，我们将把梯度显式设置为零，否则梯度会指向某个其他方向，而不是朝着最小值的预期方向。
* 输入完每批量之后使用优化器更新模型权重（Mini Batch Gradient Descent），而且打印出目前的epoch，已被训练的数据和损失值。

1. 保存已训练的模型。模型被保存在一个*tar*文件，内容包含当前的epoch，该epoch是否有进步，Transformer模型的相关信息，优化器，以及损失值。测试阶段之间会使用这份文件来加载模型。

## Experiment Results

测试阶段之间，先把模型从上述*tar*文件加载Transformer模型。然后对每个测试数据用MFCC提取特征，把它改成一个向量来代表该音频。接下来把向量通过我们的Transformer模型，然后打印出模型预测结果与音频相应的标签进行对比。以下是我们模型的训练过程及测试结果





## References: