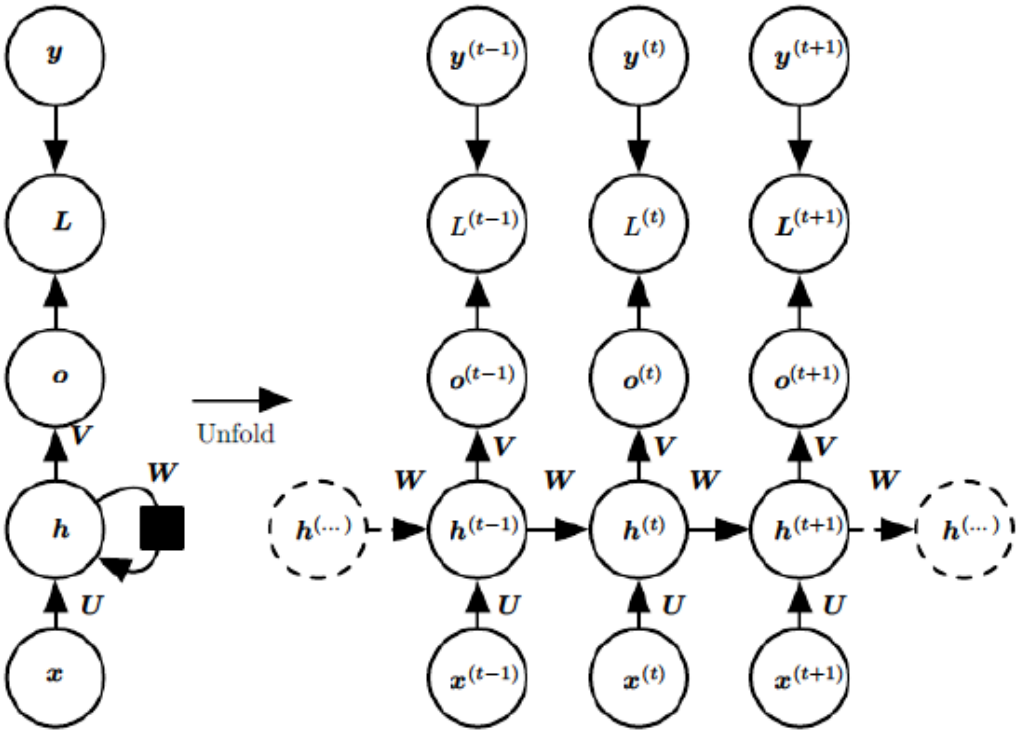
LSTM 原理

我们知道 LSTM 是 RNN 的一种变体，在了解 LSTM 之前，我们一起看一下 RNN 的原理。RNN（Recurrent Neural Network）是一类用于处理序列数据的神经网络。什么是序列呢，可以简单理解为有先后顺序的数据，比如一句话，里面的每个单词是从左到右构成这句话的；再比如一条音频，我们听到的声音肯定是一种时间序列的音频信号，具体来说就是每一帧语音数据。而 RNN 处理这种序列数据，在结构上具有天然的优势。



其中，x 是输入，h 是隐层单元，o 为输出，L 为损失函数，y 为训练集的标签。这些元素右上角带的 t 代表 t 时刻的状态，其中需要注意的是，隐层单元 h 在 t 时刻的表现不仅由此刻的输入决定，还受 t 时刻之前时刻的影响。V、W、U 是权值，同一类型的权连接权值相同。

由此可见，RNN 能够读取当前时刻元素的信息以及它之前的信息，就像我们看一句话，看到最后一个单词的时候我们也已经把前面的单词都看过了，因此读懂了整句话的意思。但是 RNN 也有明显的缺点，我们不妨假设一句话有 100 个单词，当读到第三个单词的时候（记为 word_3），word_3 可以看到且记住前面 2 个单词；当读到第 10 个单词的时候，word_10 差不多能记住前面 9 个单词；当读到第 100 个单词的时候，word_100 已经记不住前面第 1 个，第 2 个或者前面的单词了，因为间隔实在太远了，身处前面单词的信息早就随着不断的传递变得越来越少，甚至微小到根本不取作用了。因此在序列间隔不断增大时，RNN 会丧失学习到连接如此远的信息的能力。

基于这个问题，LSTM 被提出。在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛

的使用。简单来说，LSTM 具有记住较长间隔信息的能力。下面是 RNN 和 LSTM 结构的图示：

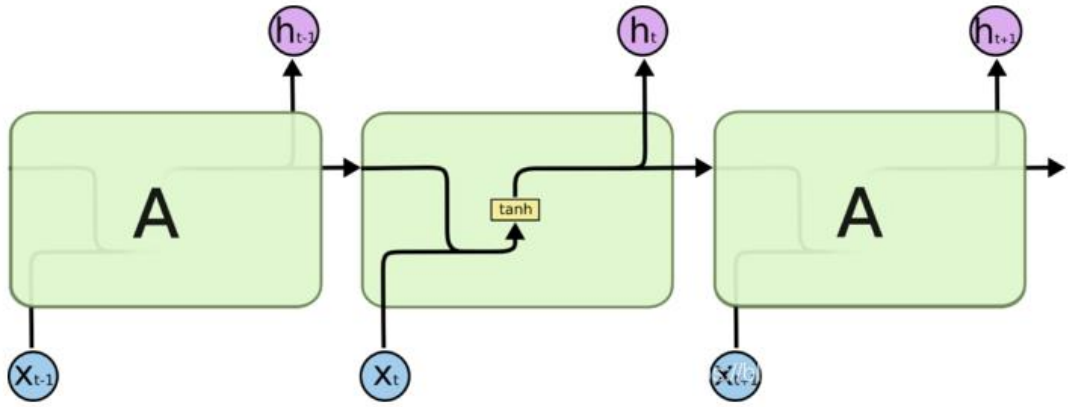


图 1 RNN 示意图

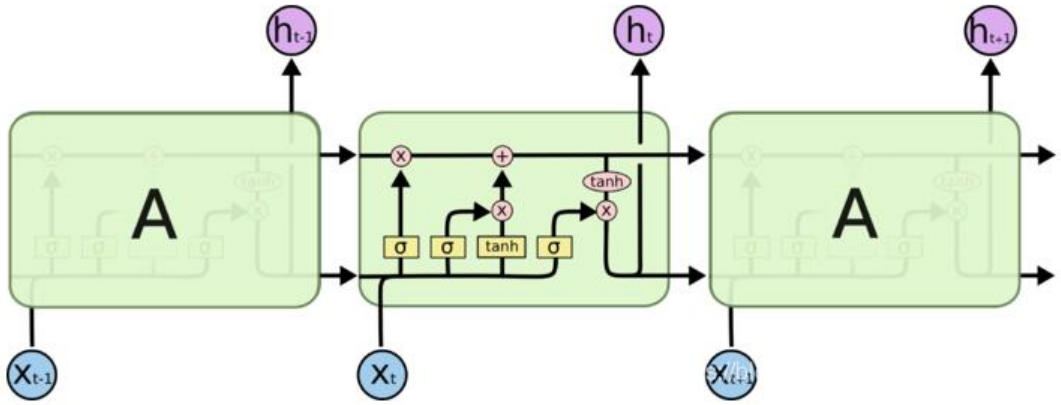


图 2 LSTM 示意图

可以看到，在中间的模块里，LSTM 比 RNN 多了几个结构。正是这多出的几个结构，很好的让 LSTM 具备了记住长序列信息的能力，具体细节知识大家可以去查阅网上资料，非常全面，这里就不作展开。

模型训练

LSTM 非常适合用于处理与时间序列高度相关的问题，例如机器翻译、对话生成、编码\解码等。因此，在我们音频数据的分类任务上，LSTM 也被作为经典模型之一，具体模型结构如图 3

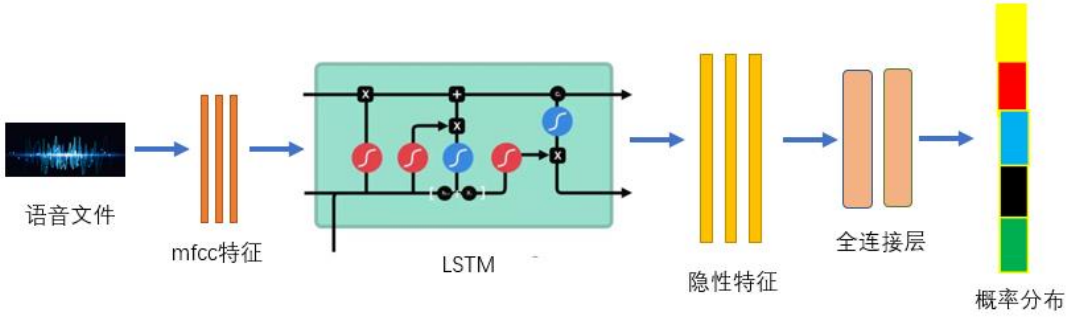


图 3 流程示意图

在此基线模型中，音频文件首先经过特征提取操作，得到 60 维的 mfcc 特征，然后将 mfcc 特征输入到 LSTM 模型中，得到 128 维的特征向量，然后将特征向量输入到两层全连接层中，得到最终的音频所属类别的预测概率分布。之后将得到的预测标签和真实标签通过交叉熵损失函数计算损失，并利用 Adam 优化算法进行更新参数。

代码展示

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class Baseline1(nn.Module):


    def __init__(self, args):
        super(Baseline1, self).__init__()
        self.args = args
        self.d_input = self.args.d_input
        self.class_num = self.args.class_num

        self.hidden_size = 128
        self.bn_dim = 30
        self.num_layers = 1

        self.lstm = nn.LSTM(input_size=self.d_input, hidden_size=self.hidden_size, num_layers=self.num_laye
                            bidirectional=False, bias=True, batch_first=True)

        self.linear_layer1 = nn.Linear(self.hidden_size, self.bn_dim)
        self.linear_layer2 = nn.Linear(self.bn_dim, self.class_num)

    def forward(self, encoder_out, input_lengths):

        x = self.lstm(encoder_out)[0]
        x = self.linear_layer1(x)
        x = self.linear_layer2(x)
        x = torch.mean(x, dim=1)
        return x,
```

图 4 LSTM 代码展示

```python
            pred = self.model(padded_input, input_lengths)
            model_out = pred[0]

            loss = F.cross_entropy(model_out, labels, reduction='sum')
```

图 5 交叉熵损失函数

代码如上图所示，可以看到我们在构造函数中，用到了 pytorch 中的 nn.LSTM()函数就可以调用 LSTM 模型，参数 bidirectional=False 即表示为单向 LSTM，如果为 True，则就是双向 BiLSTM 模型。LSTM 后面接了两个全连接层，用 pytorch 中 nn.Linear()函数可以实现。

在 forward 函数中，encoder_out 是我们的输入，为了方便直观理解，这里我们假设 encoder_out 形状为[10，200，60]，代表有 10 条语音，每条语音 200 帧，每帧 60 维度（这里需要注意的是帧数 200 是经过填充处理后的，即各语音的帧数不一定相同，但是为了丢给模型训练我们需要把形状按照这个 batch 中最大帧数为标准进行统一）。之后输入经过 LSTM 模型后，形状变成了[10，200，128]，经过第一个全连接层后，形状变成[10，200，30]，经过第二个全连接层后，形状变成[10，200，20]，最后在第一维度上求一个均值，即形状变成[10，20]。

至此，我们就求出了这 10 条语音对应着的概率分布（一共 20 个类别），之后就可以求出预测的标签。然后将预测标签和真实标签利用交叉熵损失函数进行求损失，再通过 Adam 优化器更新参数，通过 100 个 epoch 的迭代训练，得到我们准确率最高的模型。至此，模型训练结束。

评价指标

此 baseline 的评价指标主要为 ACC（准确率），即正确预测样本数/总样本数。代码如下

```
231     print('n_correct:', total_correct)
232     print('total_sen:', total_sen)
233     print('acc:', total_correct/total_sen)
234     print('每个batch的平均损失相加:', total_loss / (i + 1))
235     print('每个batch的损失相加后再平均:', sum_loss / total_sen)
236     print(metrics.classification_report(labels_cat, predicted_cat, target_names=target_names, digits=4))
```

图 6 评价指标

从上面代码里面可以看到，我们涉及到了不止一种评价指标，除了我们主要用到的 ACC，还有 P，R，F1 值三个评价指标来共同评价模型性能好坏。其中 ACC 的计算为 total_correct/total_sen；P，R，F1 我们通过 metrics.classification_report()函数可以得到，此函数具体细节大家同样可以从网络上找到非常多的学习资源，这里不进行展开。

参考文献：
https://blog.csdn.net/HappyRocking/article/details/83657993