自然语言处理后期报告

姓名：饶文均

时间：2022.12.1

当前进度

配置环境后成功运行模型，跑完了一个epoch，耗时116359.514s（约32.5小时），得出模型评估值为：

```
"exact_match": 57.172982877191
"f1": 63.801922376147
```

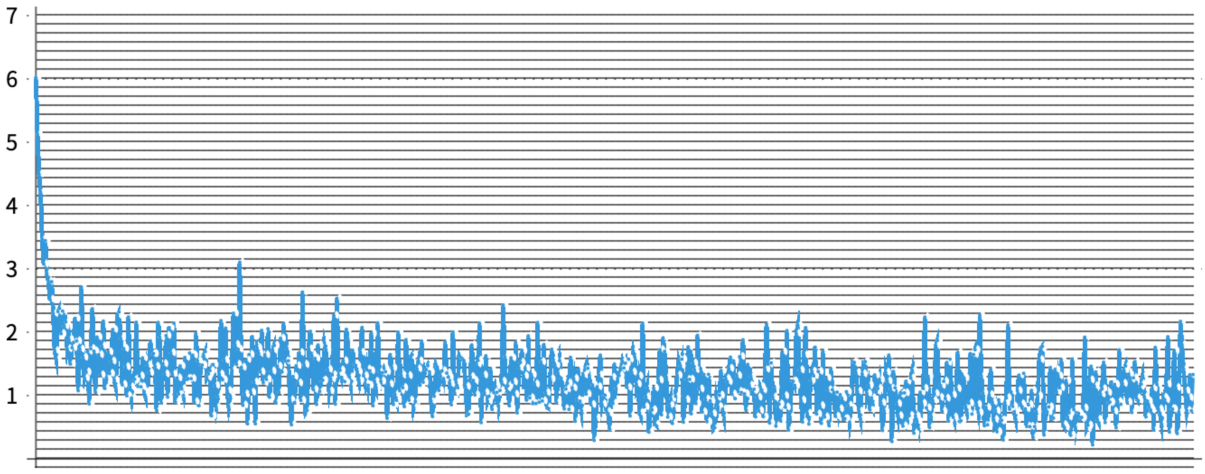
得到的模型的准确程度还是比较好的，毕竟只运行了一个epoch，我们将在实验报告中将这些数值与我搭档用TF-IDF 模型得出的数值进行对比。

在模型运行过程中我保存了相关日志，其中包括Train loss 和Train accuracy 值：

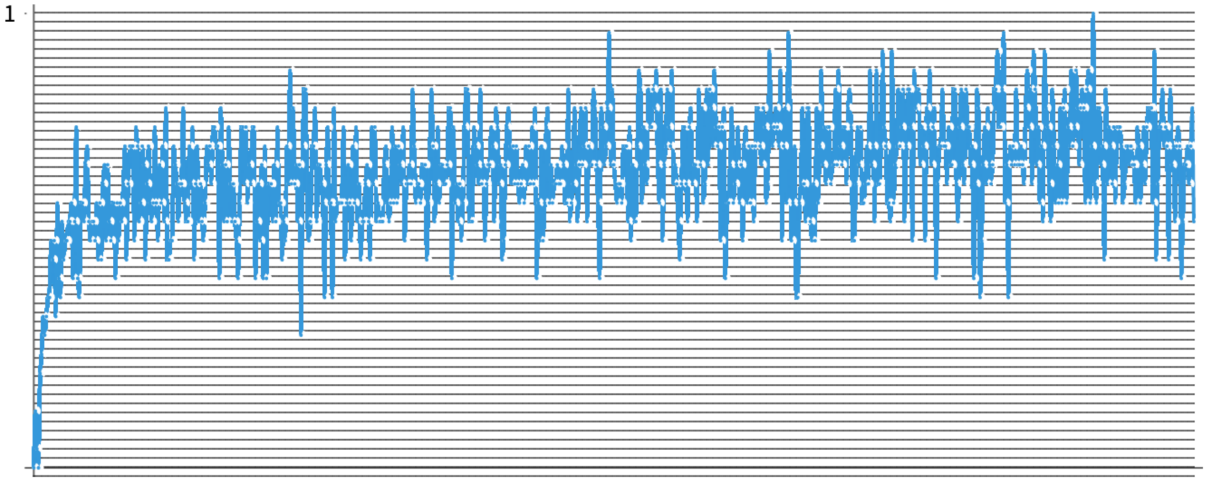
```
[2022-11-27 15:58:16] - INFO: Epoch: 0,
Batch[6610/7387], Train loss :1.579, Train acc: 0.708
[2022-11-27 16:01:18] - INFO: Epoch: 0,
Batch[6620/7387], Train loss :0.778, Train acc: 0.750
[2022-11-27 16:04:51] - INFO: Epoch: 0,
Batch[6630/7387], Train loss :0.504, Train acc: 0.875

.....
```

我注意到在模型运行过程中，在某些迭代次数后，模型的Train loss 和Train accuracy 值会发生跳变情况。为此我将Train loss 和Train accuracy值的变化过程绘制成折线图方便观察。

下图为Train loss折线图：



下图为Train accuracy折线图：



可以看到无论是loss 还是accuracy 都有很大的振幅，在周二课上老师说这是神经网络模型的正常情况，在后期我想深入探究一下这种现象出现的原因。

个人思考理解

对于loss 在模型训练过程中大幅度震荡的原因，我一开始认为是因为我在训练模型的时候因电脑内存原因将原模型的batch size调小了，这就导致了每个batch 中数据量变少。从总体上来说，每个batch 中数据量越小，其结果就越不稳定（因为一些问题的难度可能远高于其他问题，详见我前期报告中对数据集题目设置的论述），所以造成了每个batch 之间的loss 值振荡较明显。我猜测如果将batch size调整为64，loss 值的振荡情况应该会得到降低。

但老师说这并不是主要原因。老师提示我从loss 定义、梯度下降原理和Backpropagation 入手，了解其真实原因。以下是我自底向上的思考过程：

对loss函数的理解

首先我了解了loss 值背后的定义。loss 值所反映的其实就是当前batch 中模型预测答案和正确答案相差程度。其作用是估量模型的预测值与真实值的不一致程度。也就是说，训练集的loss 下降的同时验证集loss 上升则说明模型学习状况良好，鲁棒性正在提升。在模型训练的过程中，模型通过调整预测函数的参数来拟合数据，每一次参数调整都会带来loss 函数值的变化，而模型所能达到的最理想效果（也就是预测函数能达到的最佳拟合效果）就是使loss 函数值达到最小值。

我先通过一个简单的二次函数来思考：假设loss 是一个简单的二次函数，不难发现在函数图像的最低点附近形成了一个低谷区域，在这个以函数值最低点为中心的小块区域内，两边的函数值向其最低点收敛程度很高，而这个区域以外，收敛程度则较小。在实际训练中，loss 很可能是有很多极小值的弯弯曲曲的函数，也有可能是我们无法用图像表示的高维度函数，但我们都可以找到一个函数的最小值，而越靠近这个最小值，loss 函数值越收敛，预测函数的拟合程度越高，模型越精确。

对梯度下降算法的理解

正如上文所说，如果loss 是一个简单的二次函数，那我们直接通过基础的数学方法就可以准确地计算出其最小值点。但实际上我们面对的loss 函数很有可能是一个很复杂的、高维度的函数，我们只能采用类似于“摸着石头过河”的方法，一步一步地探测其函数最小值，这也就是梯度下降算法。

梯度下降算法中先规定学习率，也就是**步长**。顾名思义，算法将在某个梯度方向上前进一个步长的距离，然后停下来通过审视所有样本点来重新选择下一步前进的梯度方向。在这个模型中，作者使用的是随机梯度下降算法，也就是说在算法在每前进一个步长后，随机选取一个样本点当作全部样本，直接根据这个点来更新下一个步长的前进方向，这样做可以减少算法对样本的遍历数，大大提高算法的效率。但我认为随机梯度下降有点“坐井观天”的意思，当loss 是非凸函数时，算法探测到某个极小值就会认为它是函数最小值，并在它附近小幅度振荡，因为它不读取全局数据，所以造成了一定的局限性。

（在这里，我了解到作者在这个论文中使用了 Adam 优化器对梯度下降算法进行优化，该优化器可以自适应地优化每个参数的学习率，使其收敛速度更快，并解决了随机梯度下降被困于局部最优点的问题，还可以**减少由于高方差的参数更新导致的loss 函数振荡问题**。）

理清梯度下降算法的思路后，我认为梯度下降就是一个更新weights 的宏观思路——“吸收误差的经验教训从而进行调整”。但这也带来了一些问题，如果将梯度下降运用到一个深层神经网络中，若要调整一个隐层的weights ，就需要一个误差值，但当隐层位于“前不着村后不着店”的中间位置时，误差值从哪来呢?

Backpropagation

为了解决上文提出的问题，backpropagation 应运而生。在深层神经网络的训练中，只有到达输出层才能获得最终结果并计算loss 函数，也就是计算误差，而产生这个误差的原因就是前面数个隐层参数的变化。正如蝴蝶效应的原理，离输出层越远的隐藏层中参数的一个细微变化，经过层层传递后可能就会造成极大的误差，所以我们要进行追根溯源。
Backpropagation 通过链式求导法则，将结果的误差层层传递，每经过

一层就通过梯度下降等方法对该层的参数进行调整，使整个模型往最优的方向前进。总结起来，这个过程就是“先由前向传播拿到误差，再经反向传播将误差传递”。Backpropagation 和梯度下降结合起来，就形成了一整套纠错系统，使模型具有了自我更新能力，可能这就是神经网络“学习”的真正含义。

loss 值振荡的真实原因

我们已知在该模型中作者使用了随机梯度下降算法，在进行每一次batch 迭代后，算法将从结果的误差值来逐层调整模型的参数值。根据上文提到的随机梯度下降算法和Backpropagation 的原理，我们可以推断出loss 值振荡现象背后的真实原因：

在处理每一个batch 过程中，算法随机挑选了一个样本点来计算下一步前进的梯度方向，而这个方向不一定是梯度下降最快的方向，也就是说处理下一个batch 时的loss 值有可能不是呈收敛趋势，而是会发生跳变，从而使图像振荡。这种不确定性就会影响到下一个batch 的准确度，也就会造成acc 值的振荡。随着epoch 的增加，模型训练量提高，使得模型的梯度下降方向越来越逼近于梯度最快下降方向，这时模型loss 值和acc 值的振荡现象应该会有所改善。

另外，我了解到原论文中使用了Adam 优化器来优化梯度下降过程，但因为使用了简化版的Adam 优化器导致效果不佳，我们将在最终报告中阐述这部分。

虽然batch size 的大小也可能影响loss 值的稳定性，但最主要的影响因素是模型的随机梯度下降算法。

关于模型的结果

我们使用的是Google 提供的预训练模型，所以得到的结果指标不会太差，但因为我们没有达到足够高的针对SQuAD 的加训量，所以我们的指标比起SQuAD 官网上其他人的指标还相差甚远。如果训练量达标，应该可以达到原论文中的结果：

```
"exact_match": 80.879848628193
"f1": 88.338575234135
```

我的搭档使用TF-IDF 获得的EM 和 F1值分别是0.2 和0.31，和我们的结果也有一定差距。这就说明了BERT 神经网络模型对于处理这类自然语言处理任务是非常合适的。而根据我在前期报告中的论述，TF-IDF 的失败也体现了SQuAD 数据集设置确实合理，且有较大难度。

目前SQuAD 官网上第一名的模型F1 和EM 分别是：93.214 和90.939。我想，如果随机召集一群人类进行阅读理解的测试，可能人类群体的结果指标还会低于模型的指标。也就是说，现在的机器学习算法在阅读理解这个领域的准确度很有可能已经接近甚至超过了人类。但在了解BERT 模型是使用33亿各语言单词、3.4亿个参数训练的事实后，我觉得它的准确率超越人类是很正常的，虽然机器不具有人类的智能，但机器日益提高的运算速度让它可以做到在海量数据中汲取需要的内容，而这3.4亿个训练参数中的奥秘可能是人类永远都无法参透的。