炼丹技巧 | BERT的下接结构调参

原创 阿力阿哩哩 阿力阿哩哩 2020-02-17

收录于话题
#NLP知识与竞赛

19个

前情回顾

我们之前介绍了

BERT的原理与应用

BERT与其他预训练模型

BERT四大下游任务

现在我们基于（2019BDCI互联网金融新实体发现 | 思路与代码框架分享（单模第一，综合第二））代码实践来介绍一下BERT如何调参才能更加充分训练，使得到的模型性能更好。

具体代码链接：

https://link.zhihu.com/?target=https%3A//github.com/ChileWang0228/Deep-Learning-With-Python/tree/master/chapter8

1.Epoch

epoch：训练模型的迭代次数。我们主要看损失是否收敛在一个稳定值，若收敛则当前设置的epoch为最佳。一般来说BERT的fine-tune epochs范围为[2, 3, 4]。

2.BatchSize

BatchSize：我们用来更新梯度的批数据大小。一般来说，Batch Size设置的不能太大也不能太小，一般为几十或者几百。笔者的调参经验是看GPU占用率。我们在命令行输入gpustat查看GPU占用率，如图 4.32所示。Batch Size越大，GPU占用率也就越高，一般占满整个GPU卡训练模型为最佳。业界传闻使用2的幂次可以发挥更佳的性能，笔者并没有尝试过，大家可以去尝试一下。

众所周知，BERT模型比较大，所以我们fine-tune的时候Batch Size肯定不会太大的，所以尽可能占满GPU即可。

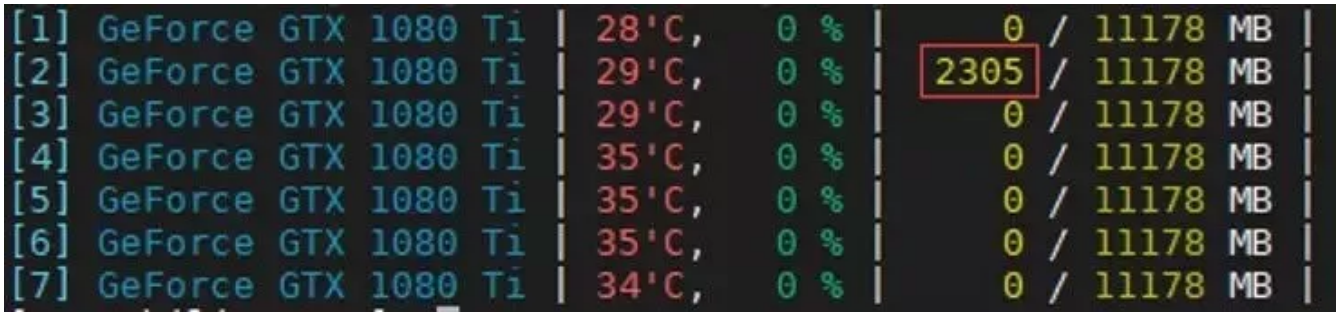


图 4.32 GPU占用率

3.Learning Rate

我们在BERT的原理与应用提及BERT原文提示用Learning Rate:[5e-5, 3e-5, 2e-5]效果会最好，这是没错的，因为BERT原文的任务并没有下接结构，直接用BERT本体去fine-tune了。

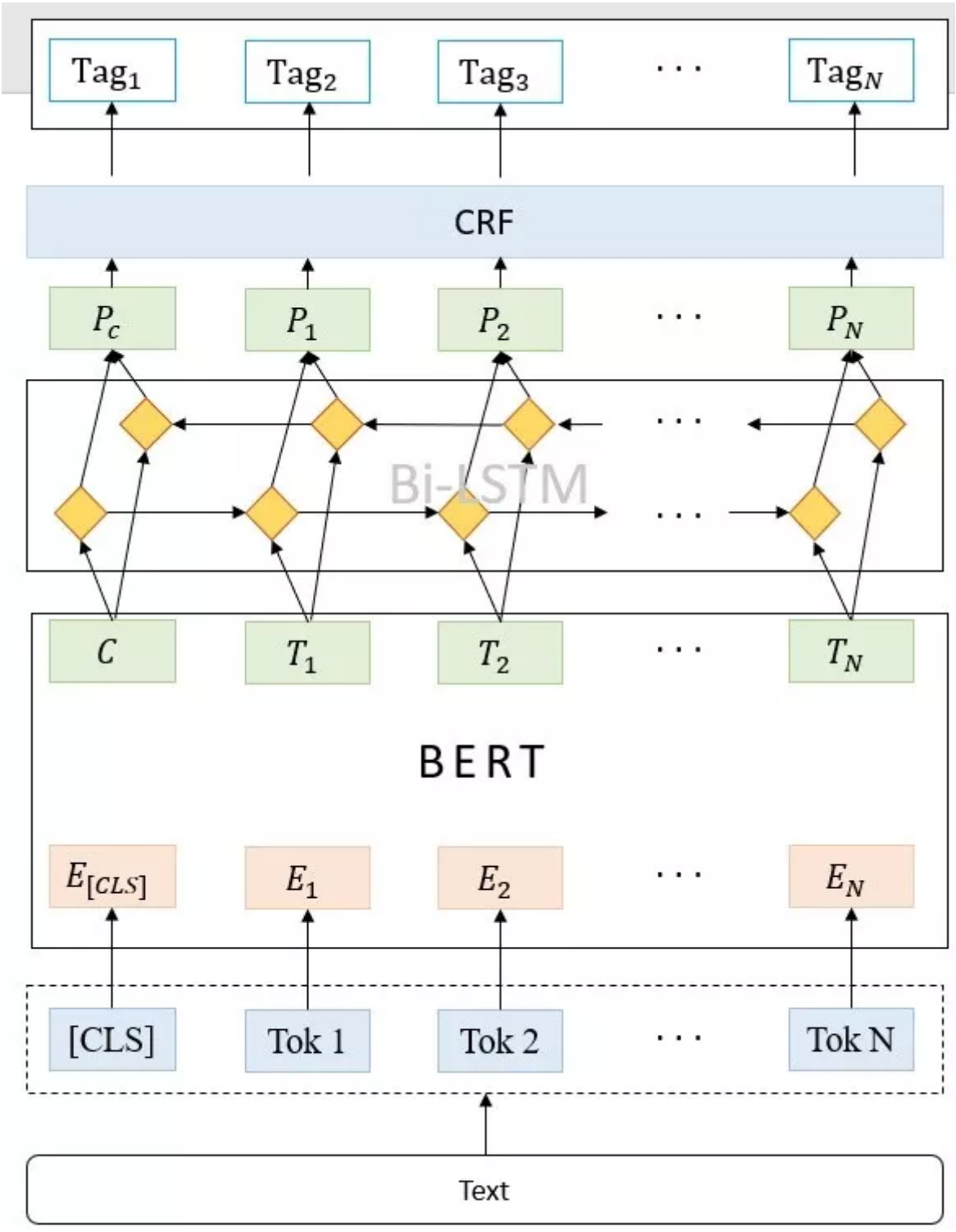
那么如果我们加上了下接结构呢？比如我们在文章开头提到的代码实践，我们的NER模型是由BERT + BiLSTM + CRF组成，也就是说除了本体BERT之外，我们还有下接结构BiLSTM + CRF，这时候我们的学习率应该有两个：

BERT的fine-tune学习率：[5e-5, 3e-5, 2e-5]

下接结构BiLSTM + CRF学习率：1e-4

至于这么做的原因也很简单：BERT本体是已经预训练过的，即本身就带有权重，所以用小的学习率很容易fine-tune到最优点，而下接结构是从零开始训练，用小的学习率训练不仅学习慢，而且也很难与BERT本体训练同步。

为此，我们将下接结构的学习率调大，争取使两者在训练结束的时候同步：当BERT训练充分时，下接结构也训练充分了，最终的模型性能当然是最好的。



BERT + BiLSTM + CRF

对BERT结构与下接结构分别采用不同的学习率进行微调，具体代码如下：

```
1.  with session.as_default():
2.      model = Model(config)  # 读取模型结构图
3.      # 超参数设置
4.      global_step = tf.Variable(0, name='step', trainable=False)
5.      learning_rate = tf.train.exponential_decay(config.learning_rate,
```

```
    # 下接结构的学习率
    normal_optimizer = tf.train.AdamOptimizer(learning_rate)

    all_variables = graph.get_collection('trainable_variables')
     # BERT的参数
     word2vec_var_list = [x for x in all_variables if 'bert' in x.name]

     # 下接结构的参数
     normal_var_list = [x for x in all_variables if 'bert' not in x.nam
     print('bert train variable num: {}'.format(len(word2vec_var_list))
     print('normal train variable num: {}'.format(len(normal_var_list))

     normal_op = normal_optimizer.minimize(model.loss, global_step=glob
     num_batch = int(train_iter.num_records / config.batch_size * confi
     embed_step = tf.Variable(0, name='step', trainable=False)
     if word2vec_var_list:  # 对BERT微调
         print('word2vec trainable!!')
         word2vec_op, embed_learning_rate, embed_step = create_optimize
         model.loss, config.embed_learning_rate, num_train_steps=num_ba
         )
 # 组装BERT与下接结构参数
         train_op = tf.group(normal_op, word2vec_op)
     else:
         train_op = normal_op
```

4.下期预告

BERT的出现刷新了很多的任务的极限，为此，掌握BERT原理与实验是每一个NLPer必须做的事。然而，BERT太大了，模型线上部署的时候会出现时空复杂度的困扰，所以后来基于BERT的tiny-Bert出现了，但是因为tiny-BERT的结构只有4层，显然性能就会打折扣了。

有问题，当然会有解决问题的人，模型蒸馏的出现就是为了解决模型较大与ensemble模型较多导致难以上线的问题。

它的原理也很简单：通过训练一个小模型来承载大模型的性能，在保持性能的同时，也降低了时空复杂度。

笔者将在下期介绍

模型蒸馏

敬请期待~

关注我的微信公众号~不定期更新相关专业知识~