初始参数

```
'maxlen':40, # maximum utterance length
'diaglen':10, # how many utterance kept in the context window
# Model Arguments
'emb_size':200, # size of word embeddings
'rnn_hid_utt':512, # number of rnn hidden units for utterance encoder
'rnn_hid_ctx':512, # number of rnn hidden units for context encoder
'rnn_hid_dec':512, # number of rnn hidden units for decoder
'n_layers':1, # number of layers
'dropout':0.5, # dropout applied to layers (0 = no dropout)
'teach_force': 0.8, # use teach force for decoder

# Training Arguments
'batch_size':64,
'epochs':10, # maximum number of epochs
'lr':2e-4, # autoencoder learning rate
'beta1':0.9, # beta1 for adam
'init_w':0.05, # initial w
'clip':5.0,  # gradient clipping, max norm
```

初始结果

```
'avg_len': 8.609278144371126 #平均样本句子长度
'recall_bleu': 0.28516362689702557
'prec_bleu': 0.28516362689702557
'f1_bleu': 0.28516362689202557
'time_consumed':933.6945259571075 #训练模型所用的时间，包括数据载入时间，单位为s
```

经过对初始模型的训练，我们发现，现有的RNN模型中有关blue的参数都相等，因此统一使用bleu代替。

参数说明

bleu

经过查阅，Bleu参数的含义如下：

Wikipedia https://en.wikipedia.org/wiki/BLEU

BLEU (**bilingual evaluation understudy**) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is" – this is the central idea behind BLEU.[1] BLEU was one of the first metrics to claim a high correlation with human judgements of quality,[2][3] and remains one of the most popular automated and inexpensive metrics.

Scores are calculated for individual translated segments—generally sentences—by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Intelligibility or grammatical correctness are not taken into account[*citation needed*].

BLEU's output is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts. Few human translations will attain a score of 1, since this would indicate that the candidate is identical to one of the reference translations. For this reason, it is not necessary to attain a score of 1. Because there are more opportunities to match, adding additional reference translations will increase the BLEU score.

zhihu https://zhuanlan.zhihu.com/p/39100621

BLEU指标的范围从0到1。 除非翻译与参考翻译完全相同，否则很少有翻译获得1分。 因此，即使是人工翻译也不一定得1分。 值得注意的是，每个句子的参考翻译越多，得分就越高。 因此，必须谨慎对待具有不同参考翻译数量的评估进行“粗略”比较：在大约500个句子的测试语料库（40个一般新闻故事）中，人工翻译对四个参考文献得分为0.3468，对两个参考文献得分为0.2571 。 表1显示了5个系统的在该测试语料库中对两个参考翻译的BLEU得分。

dialoglen：

dialoglen的长度为训练时选择的context的单词个数

dropout：

zhihu (https://zhuanlan.zhihu.com/p/38200980)

Dropout可以作为训练深度神经网络的一种trick供选择。在每个训练批次中，通过忽略一半的特征检测器（让一半的隐层节点值为0），可以明显地减少过拟合现象。这种方式可以减少特征检测器（隐层节点）间的相互作用，检测器相互作用是指某些检测器依赖其他检测器才能发挥作用。

Dropout说的简单一点就是：我们在前向传播的时候，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征，如图1所示。

当前Dropout被大量利用于全连接网络，而且一般认为设置为0.5或者0.3，而在卷积网络隐藏层中由于卷积自身的稀疏化以及稀疏化的ReLu函数的大量使用等原因，Dropout策略在卷积网络隐藏层中使用较少。总体而言，Dropout是一个超参，需要根据具体的网络、具体的应用领域进行尝试。

n_layers:

神经网络的层数

https://www.zhihu.com/question/65403482

效果好是指逼近函数的效果好，还是预测的效果好呢？前者的话已经有paper证明过了，个人理解就是，神经网络训练的过程就是调整参数的过程，可以调整的参数（weights and bias）越多，意味着调整的自由度越大，从而逼近效果越好，可以举例子: 逼近某一个函数，比较单层，2层，和多层的神经网络逼近效果。

而后者则是不正确的，针对同一个问题，层数少的时候效果差，这时候逐渐增加层数可以提高效果，但是如果盲目不停地增加层数，则会容易引起overfitting，从而导致预测效果不好，所以并不是层数越多，预测效果就一定会越好的。最后想提一下，其实增加神经元数也可以提高逼近效果。

batch size

https://www.zhihu.com/question/32673260

谈谈深度学习中的 Batch_Size
Batch_Size（批尺寸）是机器学习中一个重要参数，涉及诸多矛盾，下面逐一展开。

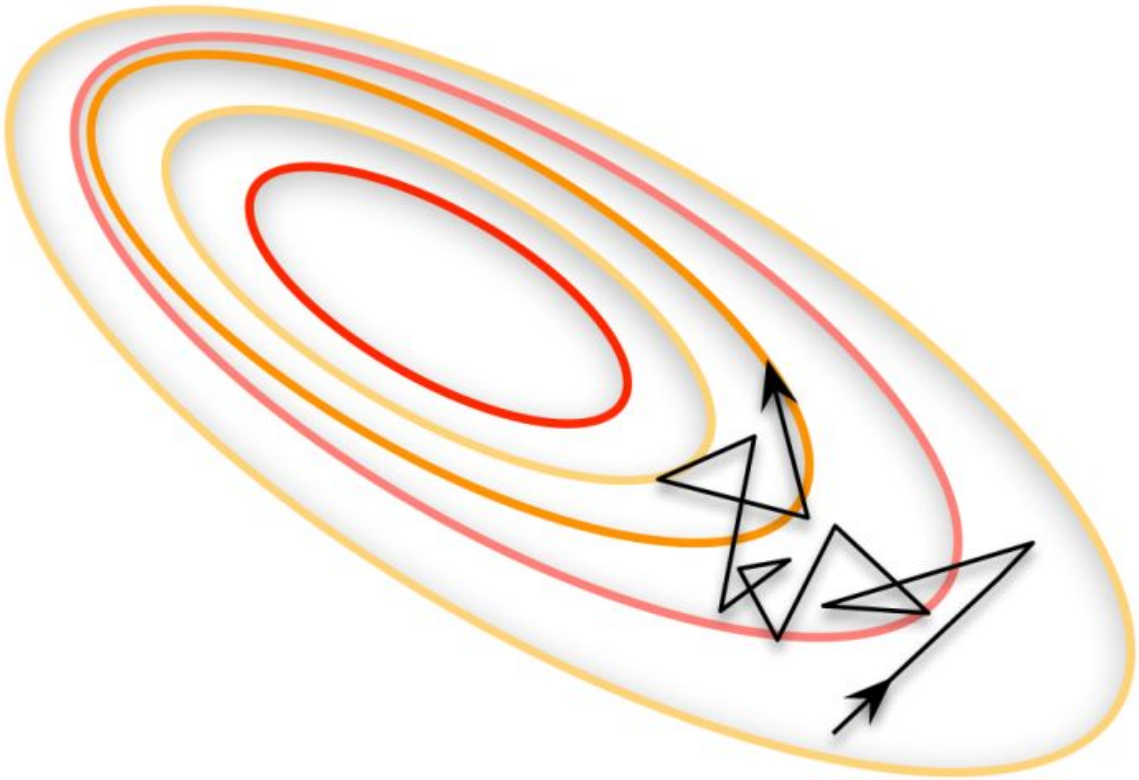
首先，为什么需要有 Batch_Size 这个参数?

Batch 的选择，**首先决定的是下降的方向。**如果数据集比较小，完全可以采用**全数据集 （ Full Batch Learning** ）的形式，这样做至少有 2 个好处：其一，由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。其二，由于不同权重的梯度值差别巨大，因此选取一个全局的学习率很困难。 Full Batch Learning 可以使用**Rprop** 只基于梯度符号并且针对性单独更新各权值。

对于更大的数据集，以上 2 个好处又变成了 2 个坏处：其一，随着数据集的海量增长和内存限制，一次性载入所有的数据进来变得越来越不可行。其二，以 Rprop 的方式迭代，会由于各个 Batch 之间的采样差异性，各次梯度修正值相互抵消，无法修正。这才有了后来 **RMSProp** 的妥协方案。

既然 Full Batch Learning 并不适用大数据集，那么走向另一个极端怎么样?

所谓另一个极端，就是每次只训练一个样本，即 Batch_Size = 1。这就是**在线学习**（Online Learning）。**线性神经元在均方误差代价函数的错误面是一个抛物面，横截面是椭圆。对于多层神经元、非线性网络，在局部依然近似是抛物面。使用在线学习，每次修正方向以各自样本的梯度方向修正，横冲直撞各自为政，**难以达到收敛**。



可不可以选择一个适中的 Batch_Size 值呢?

当然可以，这就是**批梯度下降法（Mini-batches Learning）**。因为如果数据集足够充分，那么用一半（甚至少得多）的数据训练算出来的梯度与用全部数据训练出来的梯度是几乎一样的。

在合理范围内，增大 Batch_Size 有何好处?

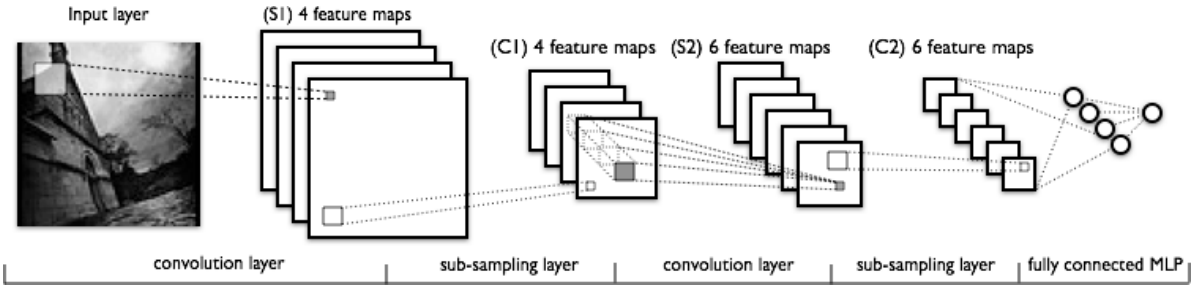
- 内存利用率提高了，大矩阵乘法的并行化效率提高。
- 跑完一次 epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快。
- 在一定范围内，一般来说 Batch_Size 越大，其确定的下降方向越准，引起训练震荡越小。

盲目增大 Batch_Size 有何坏处?

- 内存利用率提高了，但是内存容量可能撑不住了。
- 跑完一次 epoch（全数据集）所需的迭代次数减少，要想达到相同的精度，其所花费的时间大大增加了，从而对参数的修正也就显得更加缓慢。

- Batch_Size 增大到一定程度，其确定的下降方向已经基本不再变化。

调节 Batch_Size 对训练效果影响到底如何?
这里跑一个 LeNet 在 MNIST 数据集上的效果。MNIST 是一个手写体标准库，我使用的是 **Theano** 框架。这是一个 Python 的深度学习库。安装方便（几行命令而已），调试简单（自带 Profile），GPU / CPU 通吃，官方教程相当完备，支持模块十分丰富（除了 CNNs，更是支持 RBM / DBN / LSTM / RBM-RNN / SdA / MLPs）。在其上层有 Keras 封装，支持 GRU / JZS1, JZS2, JZS3 等较新结构，支持 Adagrad / Adadelta / RMSprop / Adam 等优化算法。



Batch_Size	5000	2000	1000	500	256	100	50	20	10	5	2	1
Total Epoches	200	200	200	200	200	200	200	200	200	200	200	200
Total Iterations	1999	4999	9999	19999	38999	99999	199999	499999	999999	1999999	cannot converge	
Time of 200 Epoches	1	1.068	1.16	1.38	1.75	3.016	5.027	8.513	13.773	24.055		
Achieve 0.99 Accuracy at Epoch	-	-	135	78	41	45	24	9	9	-		
Time of Achieve 0.99 Accuracy	-	-	2.12	1.48	1	1.874	1.7	1.082	1.729	-		
Best Validation Score	0.015	0.011	0.01	0.01	0.01	0.009	0.0098	0.0084	0.01	0.032		
Best Score Achieved at Epoch	182	170	198	100	93	111	38	49	51	17		
Best Test Score	0.014	0.01	0.01	0.01	0.01	0.008	0.0083	0.0088	0.008	0.0262		
Final Test Error (200 epoches)	0.0134	0.01	0.01	0.01	0.01	0.009	0.0082	0.0088	0.008	0.0662		

运行结果如上图所示，其中绝对时间做了标幺化处理。运行结果与上文分析相印证：

- Batch_Size 太小，算法在 200 epoches 内不收敛。
- 随着 Batch_Size 增大，处理相同数据量的速度越快。
- 随着 Batch_Size 增大，达到相同精度所需要的 epoch 数量越来越多。
- 由于上述两种因素的矛盾， Batch_Size 增大到某个时候，达到**时间上**的最优。
- 由于最终收敛精度会陷入不同的局部极值，因此 Batch_Size 增大到某些时候，达到最终收敛**精度上**的最优。

参数调整

调整dialoglen:

预测结果:

随着dialoglen的增大，bleu应该先会逐步增大，直到到达一个阈值会趋近于平衡。原因是当context的长度过短时，模型训练由于可用的语境太短训练效果并不好。但是由于平均对话长度的限制，当选择的context长度超过对话长度时，bleu就不再增大。

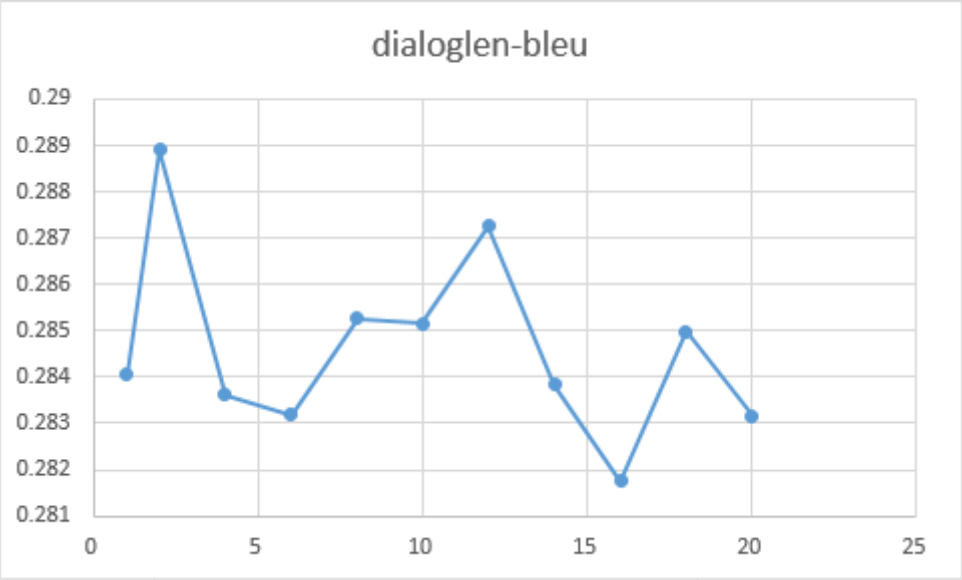
而随着dialog的增大，time_consmed应该增大，因为随着context的增大，训练模型的复杂度会增加，训练时间变长

dialoglen	bleu	Time_consumed
1	0.28404174237724883	696.8668274879456
2	0.28890403493056077	783.555210351944
4	0.2836057378572623	856.6012506484985
6	0.2831718109451517	885.6468575000763
8	0.28525902026047817	908.295951128006
10	0.28516362689702557	933.6945259571075
12	0.28724633500078595	961.2398629188538
14	0.28382962920341287	990.7178490161896
16	0.2817584768940273	994.738960981369
18	0.28497252504947096	1029.0544159412384
20	0.2831658738318325	1037.3628072738647

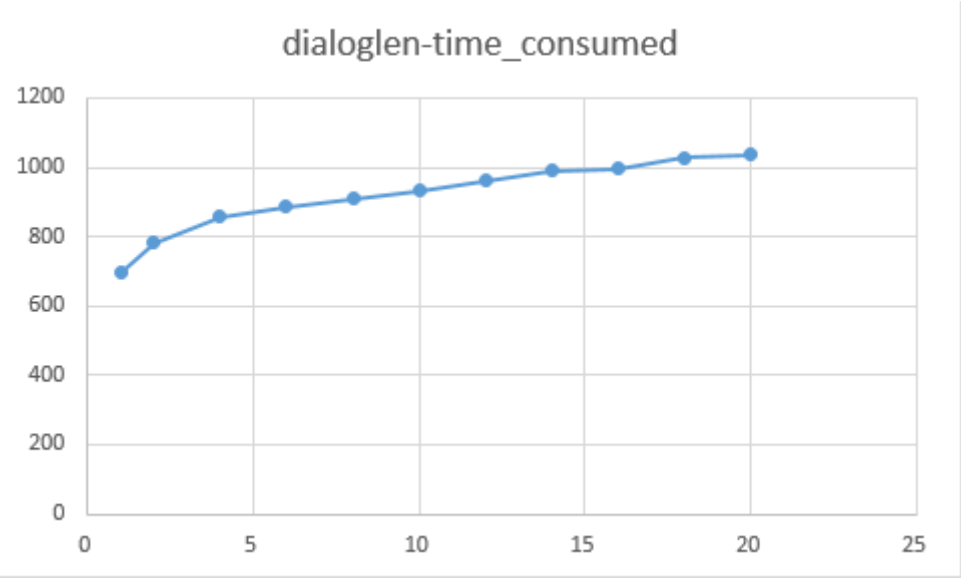
```
start = time.time()
model = train(args)
end = time.time()
print("time consumed:",end-start)
```

经过观察，我们发现训练模型的时间确实如预料的一样，随着context长度的增长而增长。但是bleu却没有明显的变化，一直在来回波动，甚至context为2的时候最高。我个人认为是因为训练样本和测试样本的平均长度都在8左右，这样训练出的模型不太需要考虑太靠前的context，所以context长度的选取对结果没有什么影响。

绘制dialoglen与bleu的关系图：



绘制dialoglen与time_consumed关系图：



调整dropout：

预测结果

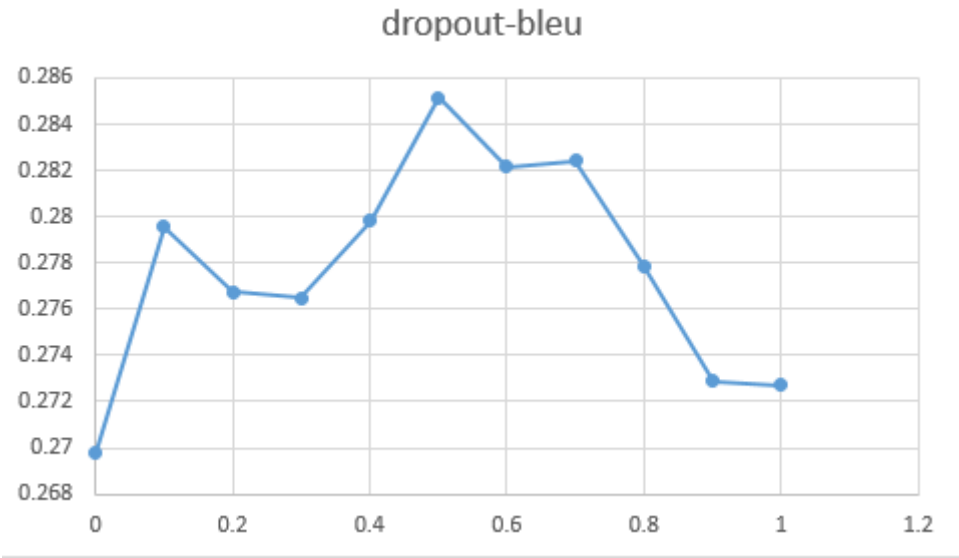
一开始dropout较小时，可能存在过拟合现象，得分可能偏低。随着dropout增大，得分逐渐升高，等到dropout过大时，结果越来越趋近于随机。过多神经元被屏蔽后，会使得训练效果也不太理想。

而训练时间应该没有什么变化。

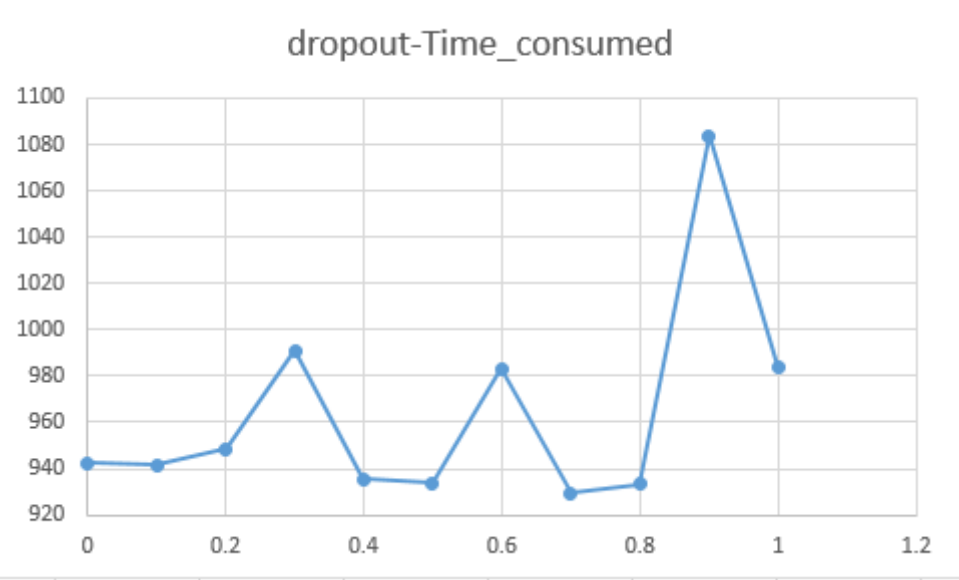
dropout	bleu	Time_consumed
0.0	0.2698249595246275	942.4414939880371
0.1	0.2795380746313778	941.5187513828278
0.2	0.2767228232340068	948.4406220912933
0.3	0.27648717483607055	991.0391957759857
0.4	0.2797996015556291	935.3210241794586
0.5	0.28516362689202557	933.6945259571075
0.6	0.2821479175978641	983.2623007297516
0.7	0.28241926574638915	929.4391362667084
0.8	0.2778827897623312	933.2382998466492
0.9	0.27287787203826674	1083.509421825409
1.0	0.2727277000152507	984.1578514575958

观察数据，发现模型训练时间在不同的dropout下都差不多，与预测的结果相同。而bleu，在dropout较小的时候较低，应该发生了预测中的过拟合现象，随着dropout的提高，bleu逐渐上升，在drop = 0.5时，bleu到达峰值，之后慢慢下降。可以发现一开始的模型给出的dropout = 0.5已经是一个调的很好的参数了。

绘制dropout与bleu关系图：



绘制dropout与time_consumed关系图：



调整n_layers

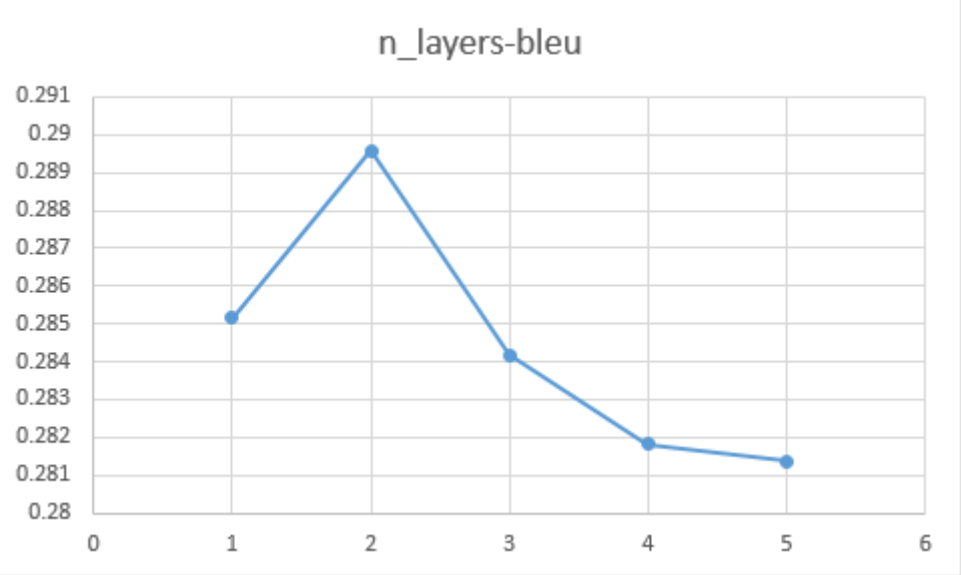
预测结果

随着神经网络层数的增多，训练时间应当花费更多。得到的bleu应当会变高，但是由于不确定过拟合的界限在哪里，所以bleu在何时达到最高应该需要在调参的过程中去发现和确定

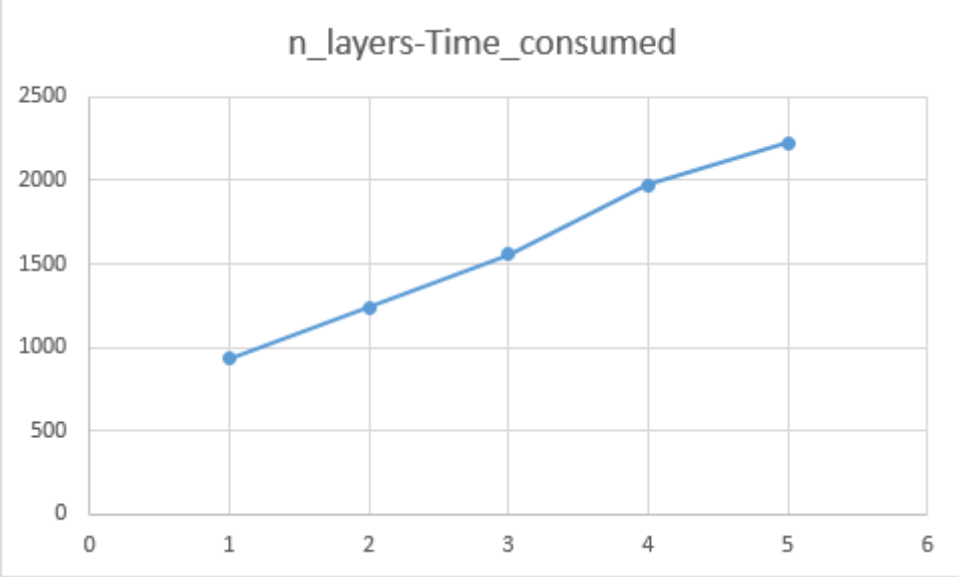
n_layers	bleu	Time_consumed
1	0.28516362689202557	933.6945259571075
2	0.28959054950367996	1237.2786984443665
3	0.284180297169922	1555.2949724197388
4	0.2818078400966952	1970.3536341190338
5	0.28137656045034893	2227.6565022468567

经过观察，n_layers为2 时训练效果最好，随着n_layers的增大，bleu逐渐下降，分析是由于出现了过拟合现象。而随着神经网络层数的增多，消耗的时间在逐渐增大，于是可以确定当n_layers大于2时再增加神经网络层数是得不偿失的。

绘制n_layers与bleu关系图：



绘制n_layers与time_consumed关系图：



调整batch_size

预测结果

批处理主要是提高了对内存的利用，使得训练模型的时间加快，但是对模型准确率而言应该没有太大提升。

batch_size	bleu	Time_comsummed
8	IO溢出	
16	IO溢出	
32	0.2868315286573202	1763.2395570278168
64	0.28516362689202557	933.6945259571075
128	0.2894518254466204	622.3960194587708
256	0.29206091953352253	441.44590973854065
512	内存溢出	

训练时发现，当batch_size太小时，会出现IO溢出的情况：

```
Evaluating in the validation set..
 56%|█████     | 3951/7069 [01:09<00:56, 55.59it/s]IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

原因应该是训练时间太慢了，IO区一直打印train_loss,step_time等信息，塞满了IO缓存区之后报错

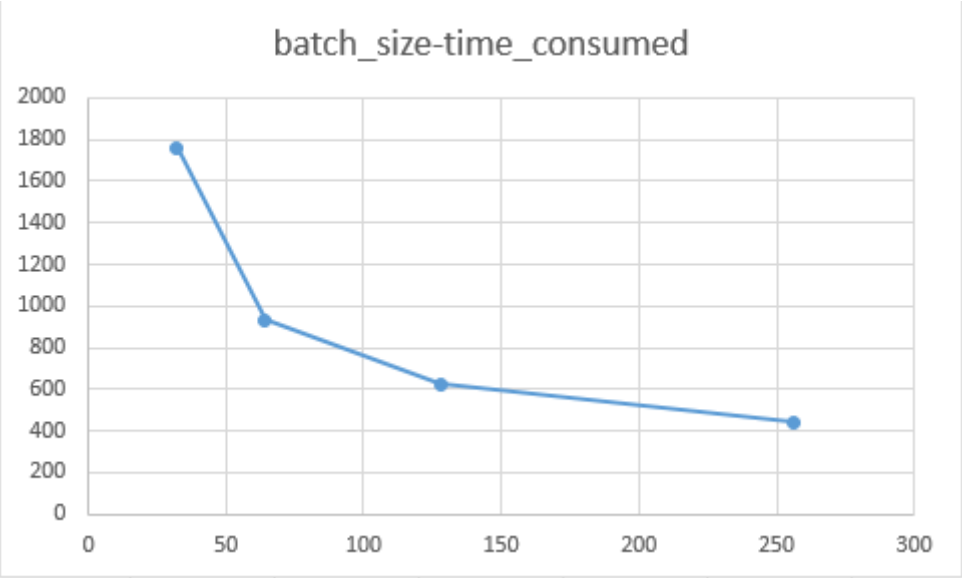
当batch_size太大时，会出现内存溢出的情况：

```
RuntimeError: CUDA error: out of memory
```

原因应该是内存利用率提升，但是内存容量不足了

绘制batch_size与bleu关系图：

绘制batch_size与time_consumed关系图：


batch_size-time_consumed
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
50
100
150
200
250
300