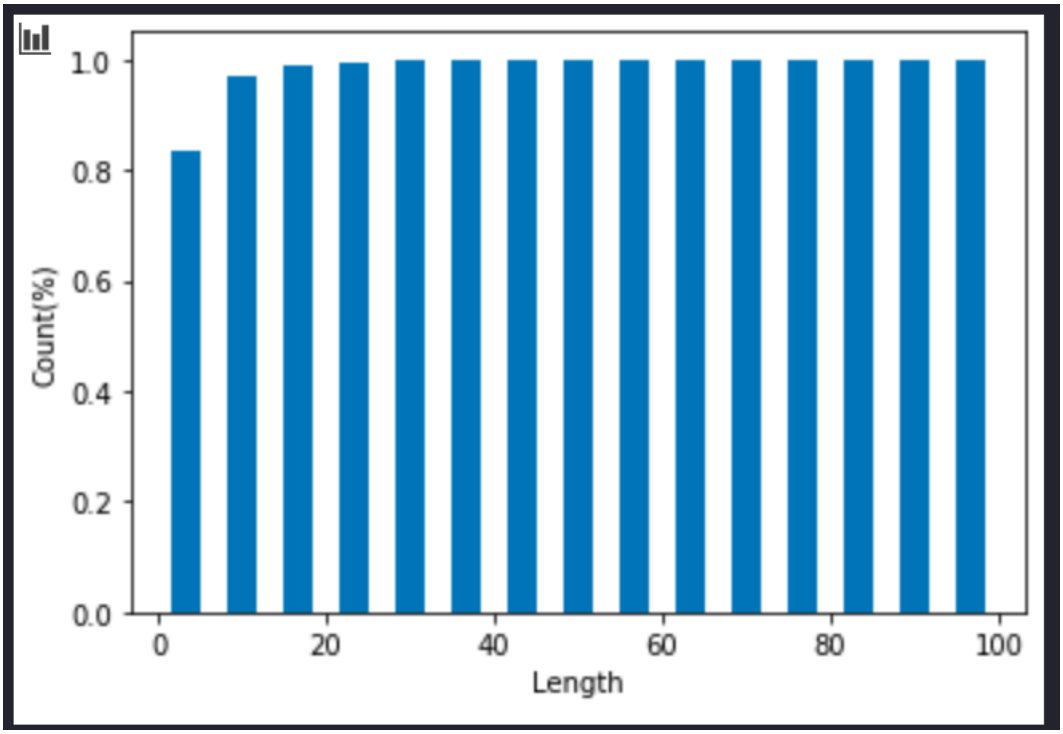
- Q1: Tokenization:
  - 在使用 BERT tokenization 時都會先加<CLS> 和 <SEP>前者代表一句話的開始後者代表與下一句話的區隔,然後 tokenization 就會將包含<CLS> <SEP>的每句話每個字分開包含標點符號,不過值得注意的是我們這次使用 pre_train("bert-base-chinese") 他會把英文數字切的比較不是以 char 的格式所以須特別注意．例如 playing ->play ##ing．
  - 如果 text 長成"男孩子在 playing 球．" 經過 tokenize 會變成"男","孩","子","play","##ing","球",'.' 所以在 label 時位置就會比較需要特別注意!不過當然也有很好的解決方法 直接把每個字都硬加空白 讓 playing -> p l a y i n g 這樣子相較於簡單實作但是變成是原本兩個 word 能表示的字需要變成六個算是比較浪費空間．
- Q2: Answer Span Processing
  - 這邊我不像助教這個做法,我反而是像上面所說的每個 word 之間都加" "(space)再去做 tokenize 就不會遇到要重新 label start_pos 和 end_pos 的問題不過缺點如上述可能有空間浪費的問題．不過有觀察過 text 基本上數字英文在一篇文章中就算有出現也不會佔很多數,所以我覺得這個方法在這個作業是可行的．
- Q3: Padding and Truncating
  - 512
  - 一開始我是設上限分別 context:462 question:50 可是一直無法過 baseline 後來發覺問題實際上常常不到 50,所以導致我沒有把 512 完整利用完．所以後來改成只有 question:50 上限 context 就讓他 512-(目前 question 長度)因為我發覺到很多答案常常介於 450 之後如果用我上面那個方法就會發覺到很多 train_data 都不會拿進去 train,因為我有設定當 answer_end_pos 不在 context 裡面的時候會讓 loss 不要去計算．後來也發覺到 context 盡量填滿,2 個 epoch 就可以過 baseline 了.
- Q4: Model
  - 我用$output = nn.Linear(hidden_size, 2)$然後判斷那兩維 $output[0], output[1]$誰比較大來決定要預測是否 answeable
  - 我用兩個 start_pos,end_pos=$nn.Linear(hidden_size, 1)$分別代表每個 word 代表 start 和 end 的機率 然後從中各挑最大的去 span,如果 start 比 end 還要後面的話會再去找各第二大的來 span 如果還是 start 比 end 後面才會判斷成 unanswe_able．
  - 我 answer_able 和 start 和 end 都是使用$CrossEntropyLoss$,雖然有看到助教在討論區說 answer_able 使用 BCE 比較符合,不過發覺效果沒很差就沒有去調整了．
  - 使用了這個$torch.optim.AdamW$,原本是使用 Adam 可是一直遲遲過不了 bseline 後來就有去查 loss_function 就發覺到有人指出 Adam 有問題,

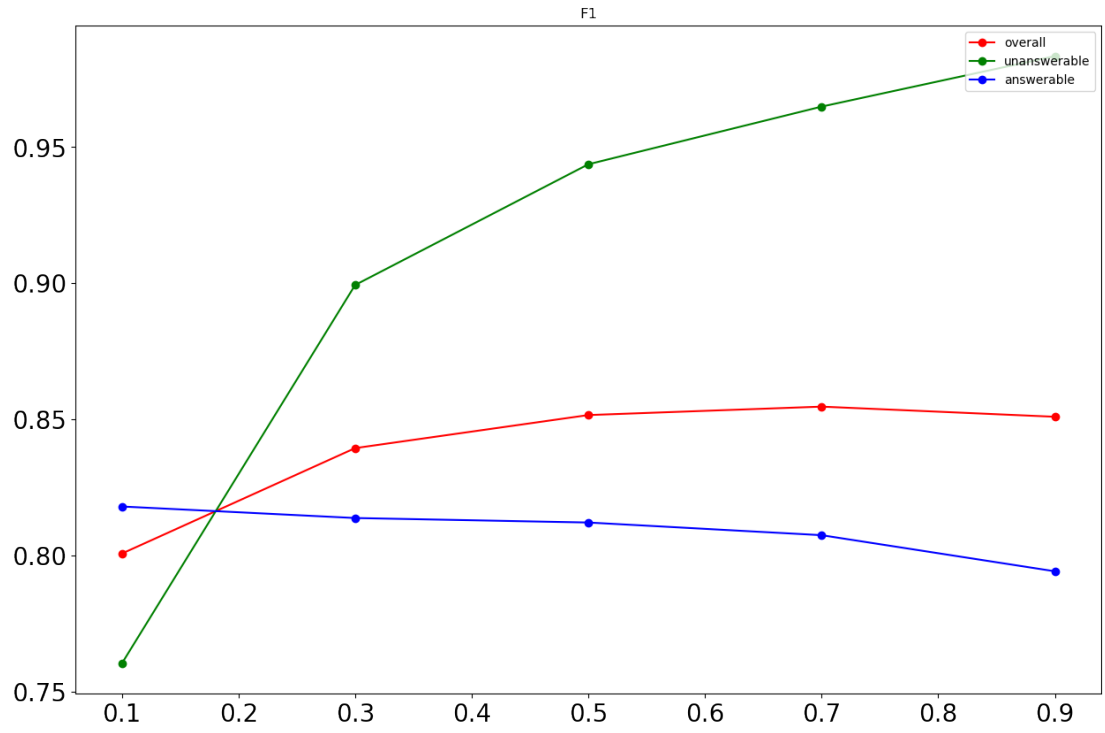
所以在後來的版本提出更改版本的$orch.optim.AdamW$。

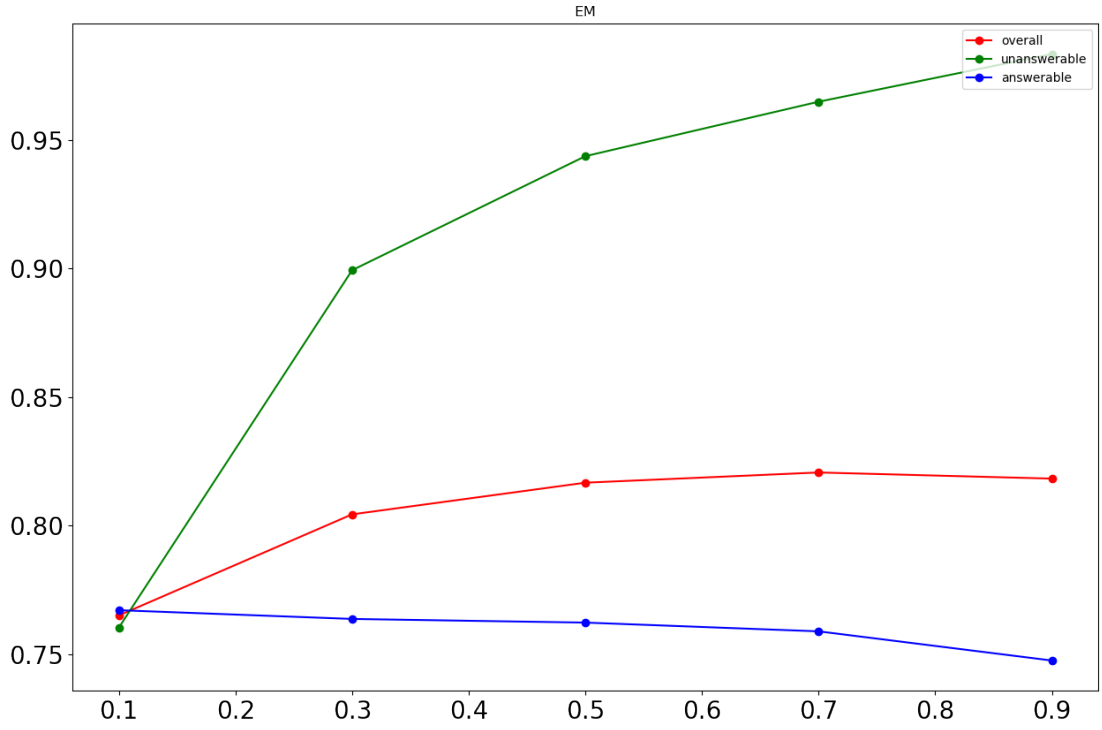
✧ Q5: Answer Length Distribution



◆ 因為有先觀察這個圖所以可以得知答案在幾乎都集中在長度 30 以內所以我在 post_processing 是用有用長度 30 限制來判斷是否 answerable 且長度介於 15-30 時會去找 top2 長度是不是有介於 15 之內,總之就是我預測答案的長度是在 15 之內的,效果也不錯。

✧ Q6: Answerable Threshold





- 原本 answerable 是用一維然後設定 threshold 來判斷,後來發覺 threshold 的設定對結果影響很大,例如在 threshold 很大的時候,unanswerable 的分數會上升很多但是相反的 answerable 可能會有所下降,後來我發覺 answerable 如果用 2 維來判斷就可以避免要設定 threshold 的問題 可以利用簡單的$output[0], output[1]$大小直接來判定是否可回答,不用再去考慮 threshold 的值的設定,所以我後來是用二維方式來判斷,**所以我是使用二維的方式來判斷,因為發覺效果比一維的好。**

```
{'answerable': {'count': 3524,
                'em': 0.7701475595913735,
                'f1': 0.8183130664359242},
 'answerable accuracy': 0.9479539133889551,
 'overall': {'count': 5034, 'em': 0.8301549463647199, 'f1': 0.8638727147636465}
 'unanswerable': {'count': 1510,
                  'em': 0.9701986754966887,
                  'f1': 0.9701986754966887}}
```

上圖是我用$output = nn.Linear(hidden_size, 2)$在 dev_set 上有很好的結果且不用考慮 threshold 值的設定。

✧ Q7: Extractive Summarization

- 做完 BERT 之後發覺真的是一個 pre-train 很好的 model 在一個 epoch 之後基本上 QA 會發覺除非那些問題很刁鑽的,BERT 都能迎刃而解。而如果回到 hw1 Summarization 要使用 BERT 來時做的話,我想我會作法跟這次類似。
- $\text{BertModel} \rightarrow \text{nn.Linear}(\text{config.hidden\_size}, 1) \rightarrow \text{nn.Dropout}(0.1)$ 所以每個 word 都會輸出一個機率然後 loss_function 用 BCE 然後最後

去找哪一句話有最多預測需要是 Summarization 的句子(簡言之出現 output>0.5 最多的那句話)。