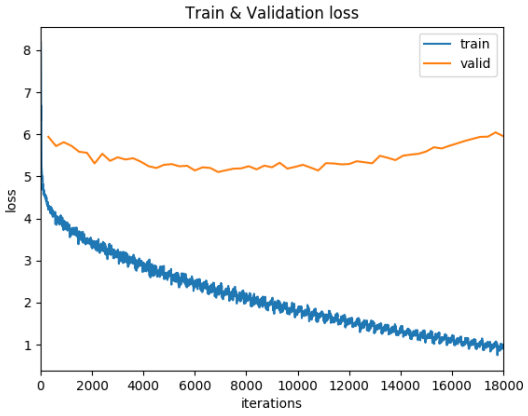
學號：B05902010　系級：資工四　姓名：張頌平

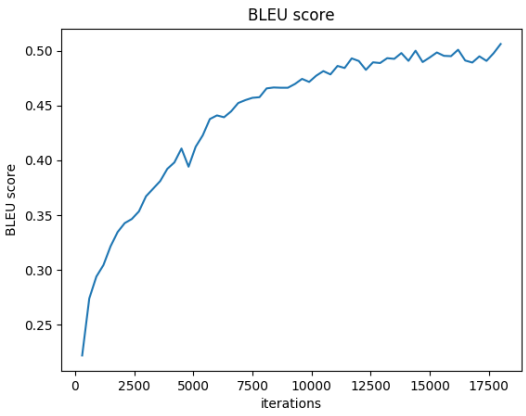
ML HW8 report

(20%) Teacher Forcing: 請嘗試移除 Teacher Forcing，並分析結果。

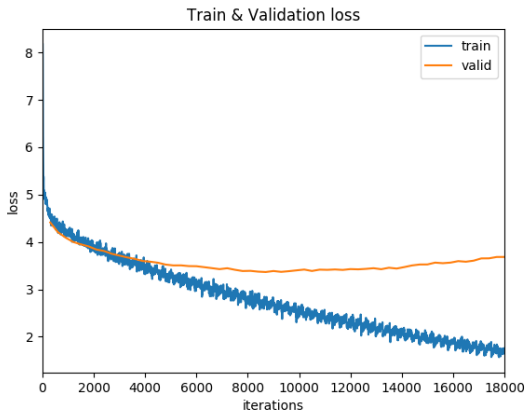
Ans:
原始的schedule_sampling回傳值為1，代表每次decoder的input都會是正確答案的字而非model前一個時間點產生的output(我們也稱其Teacher Forcing Ratio=1)；而要比較的對象則是要移除Teacher Forcing，意思就是讓每次decoder的input都會是前一個時間點產生的output，以下是兩者的實際結果和learning curve:

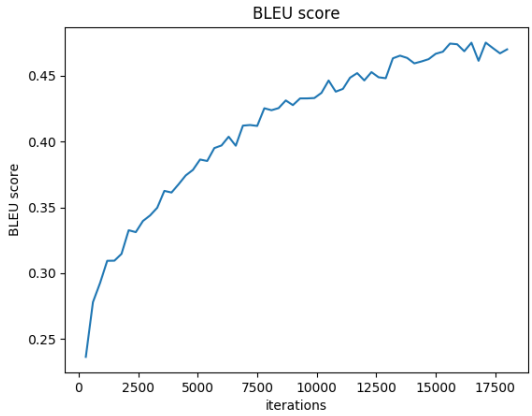
Teacher Forcing (Best BLEU@1(validation set): 0.506, Perplexity: 385.483):





Without Teacher Forcing (Best BLEU@1(validation set): 0.482, Perplexity: 33.385):





可以發現使用Teacher Forcing的結果在我們的evaluation metrics(BLEU@1)中有更好的表現，但我們可以從圖中得知，在移除掉Teacher Forcing後，validation loss比有Teacher Forcing的時候小很多(3.8xx v.s 5.8xx)，造成的原因可能是因為當model有使用Teacher Forcing時，decoder在每一個時間點的input都會是原本的正確答案，這讓model在

validation時，有可能因為某個時間點的output是沒看過的output(當時因為使用teacher forcing將當時的input(前一個時間的output)改成正確答案)，導致在預測時model會完全沒有辦法準確的預測出答案，但仍然有使用Teacher Forcing的必要，因為如果我們在前個時間點做了錯誤的決定，那往後所有的時間點都會受到這個錯誤影響，這個連鎖反應會讓訓練容易擺盪不定，因此schedule sampling(訓練過程給予model不同的Teacher Forcing Ratio，就像對於同一個問題，給予年齡不同的學生不同程度的指導)會是使得model performance能夠更好的關鍵，這部分我們留到第4小題再進行實作。

(30%) Attention Mechanism: 請詳細說明實做 attention mechanism 的計算方式，並分析結果。

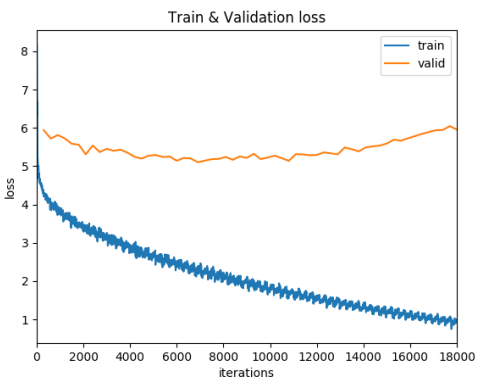
Ans:
attention mechanism 主要是用來幫助解決機器翻譯在句子過長時效果不佳的問題。這種架構會在decoder的每個時間點都創造一個context vector，來幫助decoder更能知道當下這個時間點要翻譯encoder的哪些output的資訊。我參考了**“Effective Approaches to Attention-based Neural Machine Translation”** 這篇paper的三種attention方法，分別為 dot, general 和 concat，詳細算法如下圖。實際實作的架構中，我有嘗試將attention跟decoder input 接在一起傳入GRU和將attention跟decoder output接在一起傳入fully connected layer兩種架構，但第一種架構或許是因為會讓Decoder維度太大(256 + 1024)的關係，儘管有調整過learning rate，model仍然有點train不太起來(Best BLEU@1 只有0.42)，因此後面的比較皆使用第二種架構，此外，在老師上課的時候曾有提及attention不一定要加上softmax，在嘗試過後也發現，這次的作業在我的方法中不加softmax表現也更好(+ softmax Best BLEU@1: 0.52)。
因此我最後的attention架構為: 不加 softmax layer + concat attention with decoder output(before FC layer)

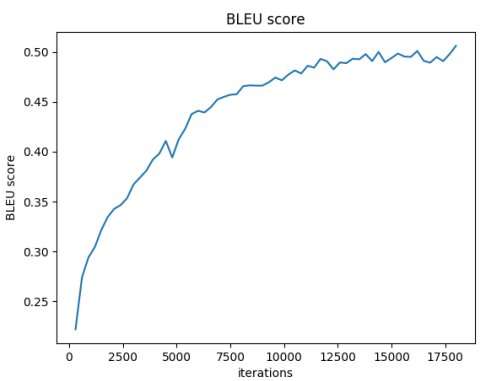
以下是詳細訓練過程和BLEU@1的分數，可以發現三種方法的performance都差不多，不過確實都有比沒有加attention的model更為進步，實際觀察句子後也發現並無太大差異，代表在這一個task上並沒有一個絕對優勢的attention方法，因此我在後面的實驗都使用最為簡易的dot方式來進行實驗。

Hyperparameter:
- total iter: 18000
- teacher forcing ratio: 1
- learning rate: 5e-5

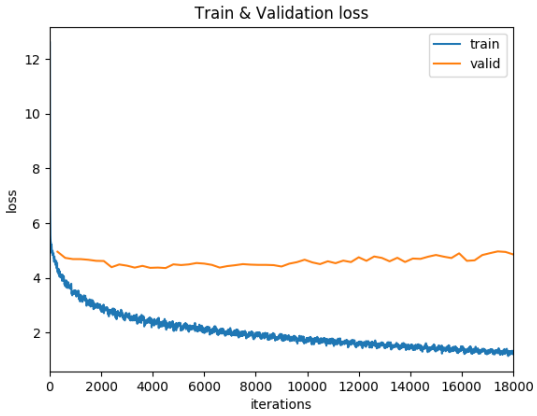
$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a [\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

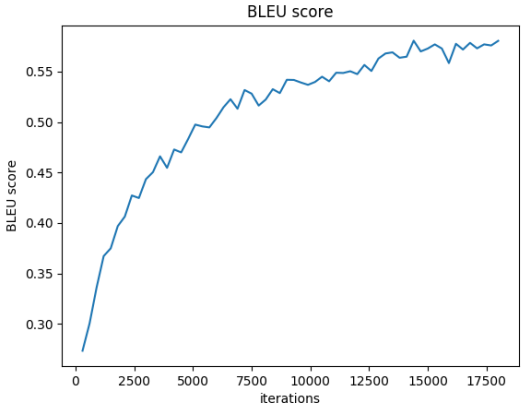
Original (Best BLEU@1(validation set): 0.506, Perplexity: 385.483):



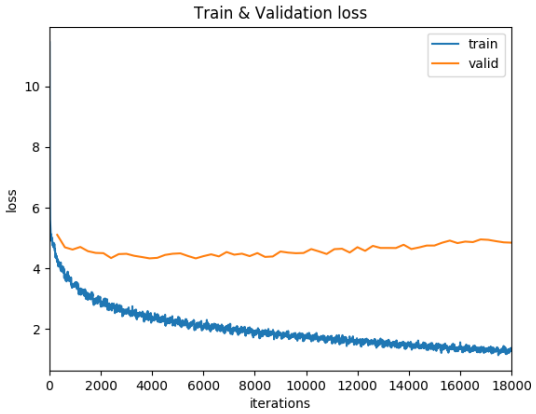


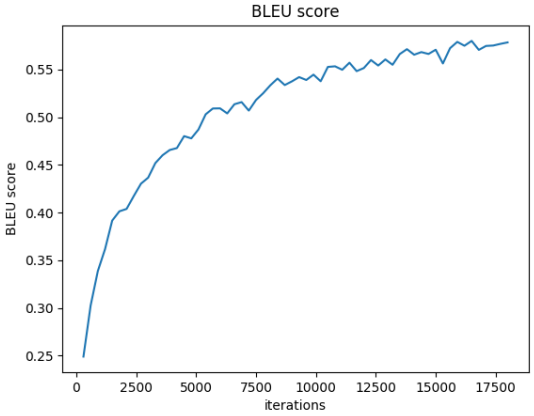
Dot (Best BLEU@1(validation set): 0.581, Perplexity: 109.316):



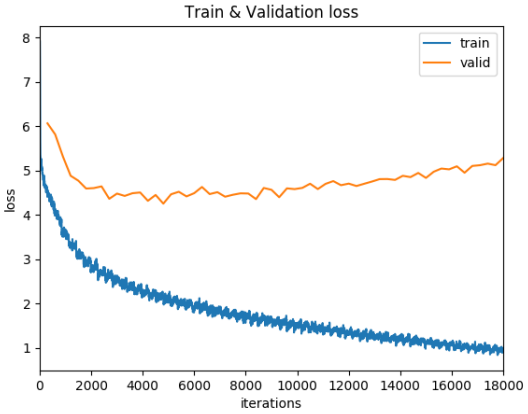


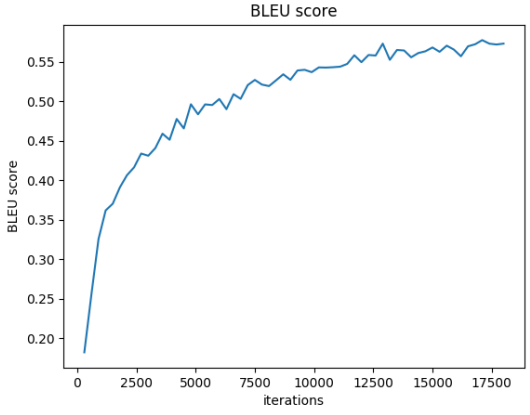
General (Best BLEU@1(validation set): 0.580, Perplexity: 129.925):





Concat (Best BLEU@1(validation set): 0.578, Perplexity: 167.802):



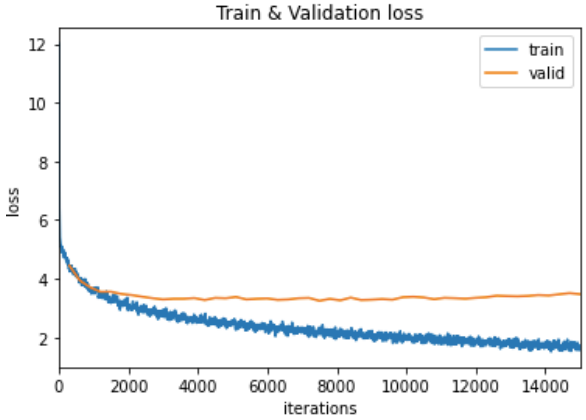


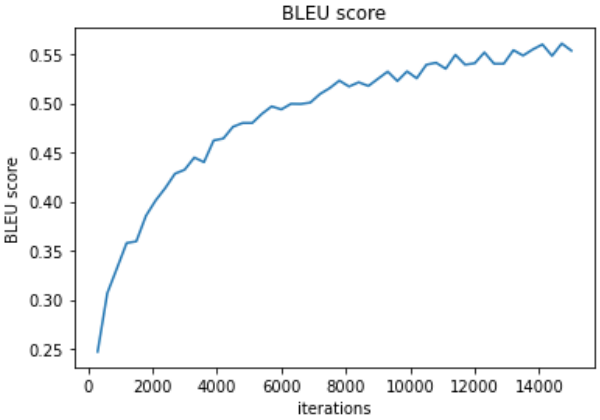
(30%) Beam Search: 請詳細說明實做 beam search 的方法及參數設定，並分析結果。

Ans:
使用beam search的主要目的是希望能讓model在inference的時候，能夠挑選真實機率最大的句子，而非挑選每個時間點機率最大的字，但因為我們沒有辦法將每一條句子的機率算出來(需要花費過多時間)，因此，我們在每一個decode的時間點，都保留當前機率最大的k個句子，並且繼續前往下一個時間點，直到出現<EOS>為止，希望這樣能找到比用greedy decode更好的結果。我實際實作方法是：假設beam size=k (意思為在每個時間點後會保留k個句子)，我會把每個句子的這個時間點所預測的前k名加入一個queue，並且用link list的方法讓他們和前面的k個句子產生連接(因此會產生k*k個句子)，接著再將這k*k個句子做sorting，機率最大的前k名保留，剩下則捨棄，直到有句子出現<EOS>，則把句子加入至end_list中，最後比較加入至end_list中的句子的機率，擁有最大機率的便是我們的答案。因為機率的連乘很有可能導致數值過小，因此我將各時間點的機率取log並且相加，這樣的方法可以解決上述問題且能讓排序不變，不過beam search有一個問題，就是他會傾向產生比較短的句子，因為句子越長，就會需要加上越多取完log的機率(會讓整體數值變小)，所以我會將機率做normalize(除以句子長度的a次方，a=0.7)，希望能夠減輕上述所提及的問題，在圖後面也會針對有無做normalize和beam size大小進行實際句子的比較，以下為參數和BLEU & Loss的圖：

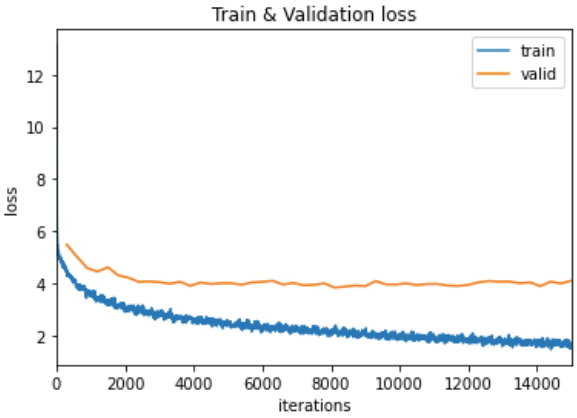
Hyperparameter:
- total iter: 15000
- teacher forcing ratio: 0.8
- learning rate: 5e-5
- attention: dot

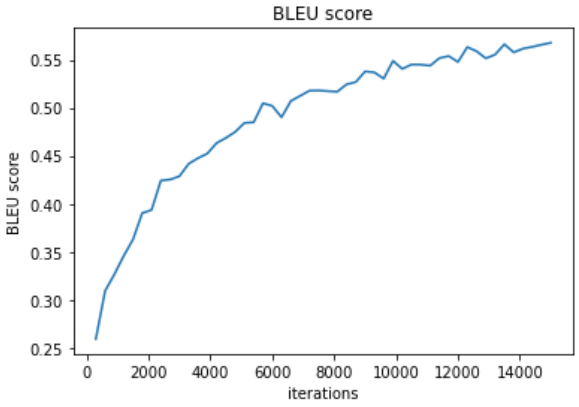
Original (Best BLEU@1(validation set): 0.561, Perplexity: 33.363):



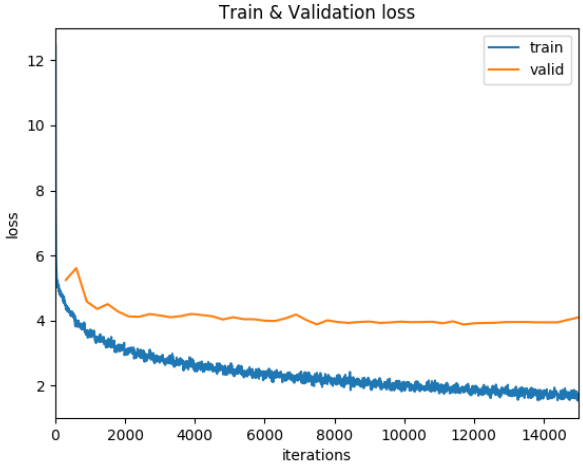


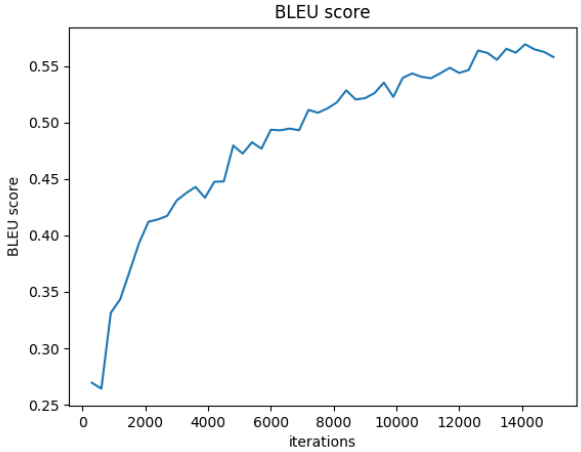
Beam search=3 (Best BLEU@1(validation set): 0.568, Perplexity: 60.637):



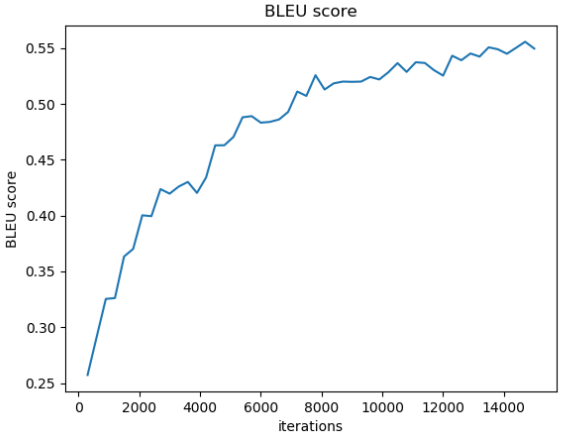


Beam search=5 (Best BLEU@1(validation set): 0.569, Perplexity: 57.129):

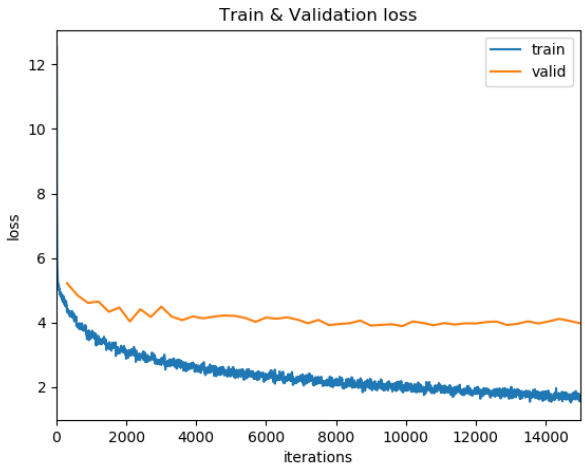


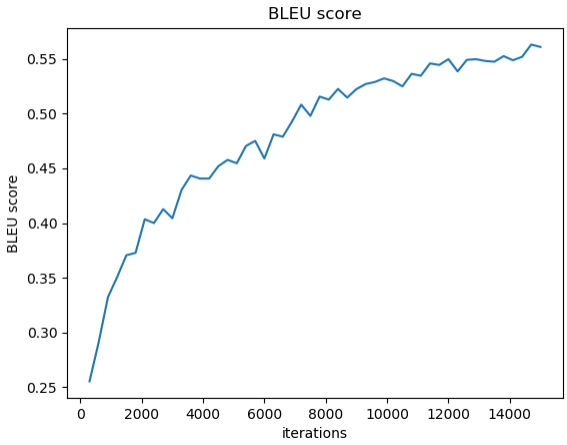


Beam search=20 (Best BLEU@1(validation set): 0.556, Perplexity: 57.330):

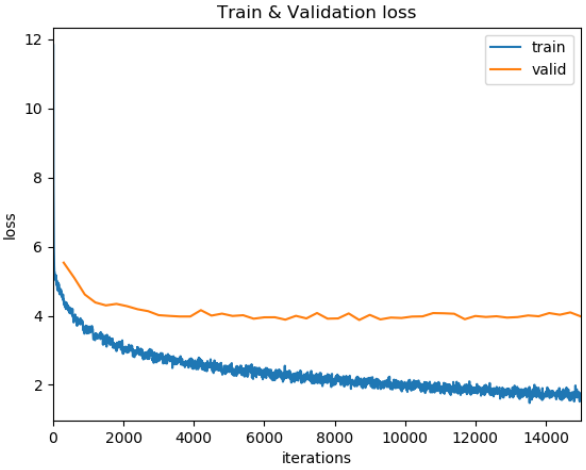


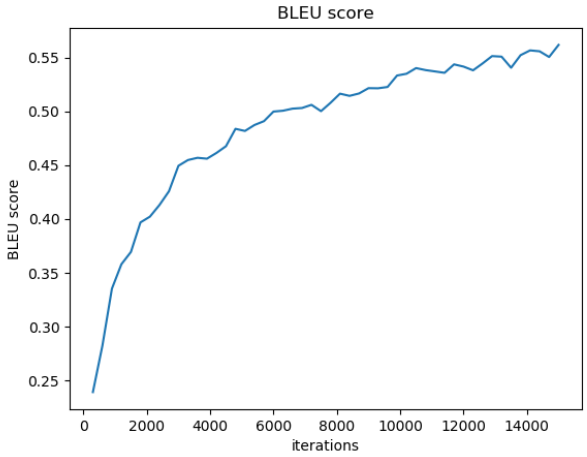
Beam search=10 w/ Normalize
(Best BLEU@1(validation set): 0.563, Perplexity: 57.411):





Beam search=10 w/o Normalize
(Best BLEU@1(validation set): 0.562, Perplexity: 53.473):





跑完上面幾組實驗後發現單就BLEU score上，beam search看上去沒有比較進步，有可能是要訓練更久才能從這一個evaulation metrics上比較出差別，而以下我挑選了四個不同的句子對於beam size不同來進行比較，來看實際產生的句子有無更為通順。

原句1：There is a spoon missing **正確翻譯解答：'少', '了', '一把', '勺子', '。'**	
beam size:1	'少', '一把', '勺子', '。'
beam size:3	'缺', '一把', '勺子', '勺子', '。'
beam size:5	'缺', '一把', '勺子', '。'
beam size:10	'缺', '一把', '勺子', '缺', '。'
beam size:10 w/o Norm	'缺', '缺', '勺子', '。'
beam size:20	'少', '一把', '勺子', '。'

原句2：I think maybe Tom was right. 正確翻譯解答：'我', '認為', '湯姆', '可能', '是', '對', '的', '。'	
beam size:1	'我', '認為', '湯姆', '是', '對', '對', '。'
beam size:3	'我', '認為', '湯姆', '認為', '是', '對', '。'
beam size:5	'我', '認為', '湯姆會', '湯姆', '。'
beam size:10	'我', '認為', '湯姆', '認為', '對', '。'
beam size:10 w/o Norm	'我', '認為', '湯姆', '認為', '湯姆', '。'
beam size:20	'我', '認為', '湯姆', '是', '對', '的', '。'

原句3：My mother is a very good cook. 正確翻譯解答：'我媽媽', '的', '廚藝', '很', '好', '。'	
beam size:1	'我', '的', '母親', '很', '很', '好', '廚師', '。'
beam size:3	'我', '的', '連衣裙', '非常', '非常', '棒', '廚師', '廚師', '。'
beam size:5	'我媽媽', '的', '是', '個', '廚師', '廚師', '廚師', '廚師', '。'
beam size:10	'我媽媽', '是', '個', '個', '廚師', '廚師', '廚師', '。'
beam size:10 w/o Norm	'我媽媽', '是', '個', '非常', '棒', '的', '廚師', '。'
beam size:20	'我媽媽', '是', '個', '很', '棒', '的', '廚師', '。'

原句4：Tom told Mary he couldn’t do what she asked him to do. 正確翻譯解答：'湯姆', '告訴', '瑪麗', '他', '不能', '做', '她', '要', '他', '做', '的', '事', '。'	
beam size:1	'湯姆告', '訴瑪麗', '瑪麗', '他', '做', '他', '做', '做', '做', '做', '他', '做', '做', '。'
beam size:3	'湯姆告', '訴瑪麗', '訴瑪麗', '他', '做', '做', '做', '做', '他', '做', '的', '事', '。'
beam size:5	'湯姆告', '訴瑪麗', '他', '不能', '做', '她', '做', '做', '什麼', '。'
beam size:10	'湯姆', '訴瑪麗', '瑪麗', '不會', '做', '做', '做', '做', '事', '事', '做', '。'
beam size:10 w/o Norm	'湯姆告', '訴瑪麗', '瑪麗', '他', '不會', '做', '她', '做', '做', '。'
beam size:20	'湯姆', '訴瑪麗', '他', '不', '做', '她', '做', '做', '什麼', '事', '。'

從以上四個範例看到，儘管在BLEU上的分數差異不大，但beam size到了一定程度大小後，在閱讀上是有更為通順的，這邊也體現了BLEU並不是一個毫無缺點的evaluation的方法，像是原句3，beam size=20和beam size=10 w/o norm 其實都有正確翻譯出答案，但因為和正解表示方法不同，造成BLEU會跟其他完全不順的句子分數相差不遠。而在做beam search時，對於搜尋每個句子的分數有無對句子長度做normalize，從例句看起來差異不大，不過從直覺來看，當需要翻譯的句子越長時，有做normalize結果應該仍會更好，最後，在查看以上四句範例和其他validation的output後，發現beam size=20的這個實驗結果，儘管它的BLEU在所有實驗對照當中最低，但其輸出的句子為最為通順且合理，因此在最後輸出的

model我們也會用其設定，不過由於耗費時間非常久，在最後一題我們僅用beam size=5來進行schedule sampling的比較。

(20%) Schedule Sampling:
請至少實做 3 種 schedule sampling 的函數，並分析結果。

Schedule Sampling的意思就是在不同時間點給model不同大小的teacher forcing ratio，通常這個值是逐一遞減的，因為model在訓練過程中也學習到很多，因此我們在訓練後期不用每次都跟他說正確答案，而是希望他能夠透過前面我們教他(或是說提示他)的，學習自己去預測整個句子，這邊我選擇了三種schedule sampling的函數，分別為linear decay, exponential decay, inverse sigmoid decay，其中exponential decay是讓teacher forcing ratio在前期就下降很多，這個想法是希望model 在很初期就能夠學到如何自己產生全部句子，因此我們就不用在訓練中後期還給model答案，導致其訓練過程不穩定；inverse sigmoid decay則恰恰相反，這種方法前期的下降幅度很小，到訓練的中後半段才讓teacher forcing ratio快速下降，而linear decay則介於兩者之間，其下降幅度是固定的。當然，根據參數大小的不同，也會有完全不一樣的結果，另外，在過去的paper中，linear decay通常會設置一個threshold k，當teacher forcing ratio小於k時，便使得teacher forcing ratio等於k，算是希望能夠讓model持續被老師糾正答案的一種方式，實際參數設置和訓練圖如下：

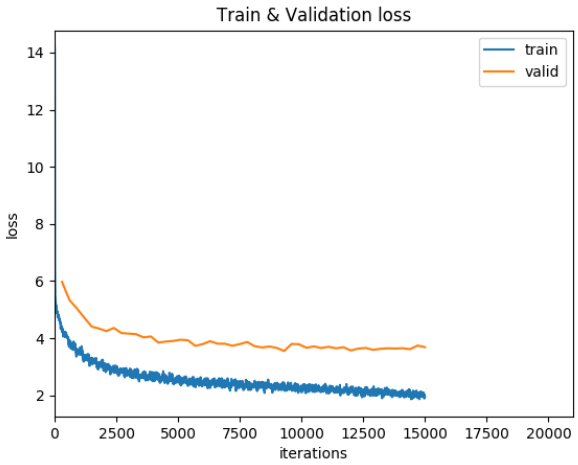
Hyperparameter:
- total iter: 15000
- learning rate: 5e-5
- attention: dot
- beam size = 5

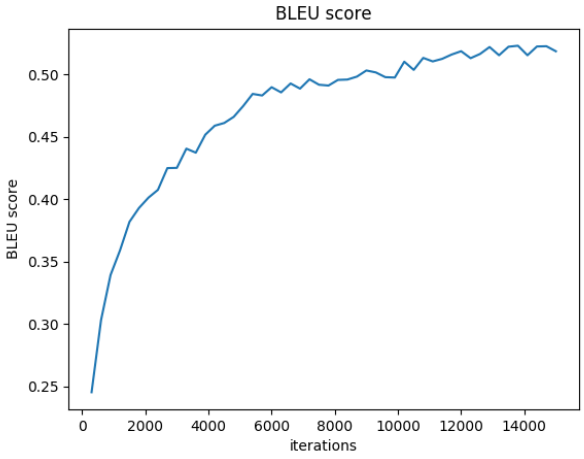
Linear decay: max(k, 1 + slope*steps/total steps)
- theshold k: 0.3
- Slope: -1

Exponential decay: k^(steps)
- k: 0.9998 (5000 steps: 0.367, 10000 steps: 0.135, 15000 steps: 0.049)

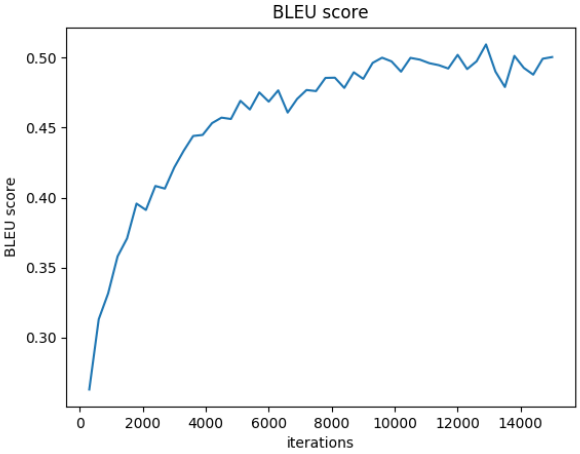
Inverse Sigmoid decay: k / (k + exp(steps/k))
- k: 1667 (5000 steps: 0.988, 10000 steps: 0.805, 15000 steps: 0.170)

Linear decay, threshold k = 0.3
(Best BLEU@1(validation set): 0.523, Perplexity: 38.110):

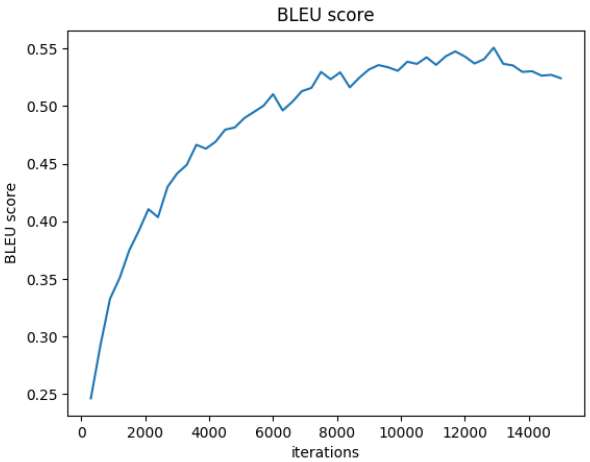




Exponential decay, threshold k = 0.9998
(Best BLEU@1(validation set): 0.502, Perplexity: 36.412):



Inverse Sigmoid decay, threshold k = 1667
(Best BLEU@1(validation set): 0.553, Perplexity: 45.177):



可以發現以上三種scheduled sampling的結果並沒有變好，尤其是在teacher forcing ratio低於0.3左右時，BLEU的成長就趨於靜止，推測是因為訓練的iteration次數過少，導致model在還沒學習完全部知識時，就因為teacher forcing ratio降低而使得model沒辦法靠著自己的訓練讓結果更為進步，但從這一個特點來看，也可以得知，Inverse Sigmoid的方法對於我們這個task來說是最好的，因為model需要比較長時間的指導(給出前一個時間點的正確答案進行訓練)，因此最後我上傳的model是將iteration次數拉高，並使用 Inverse Sigmoid decay 的作法，beam size=20，實際跑validation set(data: 500)花費時間在我自己的RTX2070上，為15分鐘，詳細參數和結果圖如下。

Hyperparameter:

- total iter: 24000
- learning rate: 5e-5
- attention: dot
- beam size = 20
- sampling: Inverse Sigmoid decay
- k: 2500(6000 steps: 0.995, 12000 steps: 0.953, 18000 steps: 0.651, 24000 steps: 0.144)

Best BLEU@1(validation set): 0.573, Perplexity: 40.523