UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and, Machine Intelligence 2018-2019

Module Coursework Feedback

Module Title: Statistical Machine Translation

Module Code: MLMI8

Candidate Number: F606F

Coursework Number: 1

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(please circle as appropriate)*:

	Distinction			**Pass**			**Fail (C+ - marginal fail)**		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

MLMI8: Statistical Machine Translation
Practical Report

1 Neural Machine Translation Decoding Strategies

Exercise 1

The files `words/ini/fst.tfnmt.small.{dev,tune,test}` are firstly modified. Their fst path is changed to `$DIR/lats/words/lats.detmin.{dev,tune,test}/%d.fst`, and they are renamed as `words/ini/fst.tfnmt.detmin.{dev,tune,test}`.
Then the SGNMT for unweighted rescoring is run. The dev, tune and test results are listed in line 5 of Table 1.

Exercise 2

The lattices are firstly transformed into log-semiring, and then pushed[1]. The transformed lattices are stored.
The scripts `words/ini/fst.tfnmt.detmin.{dev,tune,test}` are modified by changing the setting of `use_fst_weights` to true, adding the line `predictor_weights: 0.5,0.5` afterwards and changing the fst path.
They are then renamed as `fst.tfnmt.dmpush.{dev,tune,test}`, and used to score the transformed lattices and the results are listed in line 6 of Table 1.

Exercise 3

To build a flower transducer that maps word sequences to sequences of BPEs, the english vocabulary `wmap.en` and the byte pair encoding english vocabulary `wmap.bpe.en` are treated as the input and output languages respectively. The rules listed in `tmp.w2bpe` are used to identify the mapping between the input symbol and the output symbols.
Figure 1 illustrates some examples of how the flower transducer is built. And the resulting transducer is named as `w2bpe.en.fst`.

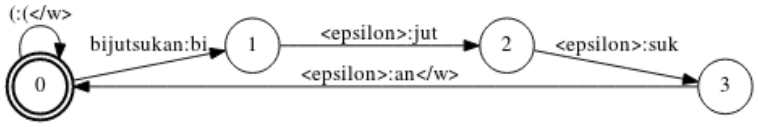


Figure 1: w2bpe transducer

After composing `fst.tfnmt.dmpush.{dev,tune,test}` with `w2bpe.en.fst`, they are stored in files `S342{dev,tune,test}`. The files `fst.tfnmt.dmpush.{dev,tune,test}` are modified for scoring. The `src_test` has to be changed to `$DIR/data/bpes/dev.bpe.ids.ja`, the `fst_path` becomes `S342{dev,tune,test}/%d.fst`, the `t2t_{src,trg}_vocab_size` is changed to 35786 and 32946 respectively, and the `t2t_checkpoint_dir` has to be changed to `$DIR/translate_jaen_kyoto32.bpe/tf_nmt.small.1/average`.
Finally, they are scored and the results are listed in line 7 of Table 1.

	NMT		HiFST WFSAs			BLEU		
ID	Word	BPE	type	Det/Min	Weighted	Dev	Tune	Test
1.					✓	15.7	13.5	18,1
2.	m1					11.8	10.9	15.5
3.		m1				13.1	12.2	16.2
4.	m1		word			14.4	13.4	17.7
5.	m1		word	✓		14.7	13.4	18.6
6.	m1		word	✓	✓	15.6	13.7	19.2
7.		m1	bpe	✓	✓	15.4	14.1	18.9
8.		m1	Constrained to English Vocab			12.6	11.9	16.0

Table 1: KFTT Dev, Tune and Test BLEU Scores for NMT Decoding Strategies

Question 1

a)

Applying minimization and determinisation on the lattices increases the BLEU scores for all sets.

The determinization operation chooses the best path among the paths with similar translation to remain in the fst. The minimization operation helps to reduce the number of states in the fst, and combine the similar path together.

These operations allow each possible translation path to be rescored, combined and filtered out if necessary, which improve the performance.

b)

The lattices are firstly transformed into log-semiring, and then pushed. [1]

c)

decoding seconds with test file	pure NMT	lattice with push,detmin
word-level	32198.9	98777.9
bpe-level	104292	91924.7

Table 2: Decoding speed of different configurations

Refer to Lines 2 and 3 from Table 1, the subword-level pure system has a higher BLEU score than the word-level one. However, from Lines 6 and 7, the word lattice system seems to have higher scores than the bpe system, except for the tune set. And when comparing Line 6 with Line 2 and Line 7 with Line 3, the HiFST system has a better BLEU score than the pure system.

Refer to Table 2, the subword-level pure system is observed to have a lower decoding speed than the word pure system probably but the trend is reversed for lattice rescoring system. The subword pure system gets accelerated, but the word pure system gets decelerated by the use of lattices.

The suitable NMT decoder architecture is the one with word lattice system since it has generally better scores and faster decoding speed.

Question 2

a)

A script, named `limit.py`, is created to replace words with "$id > n$" from `test.en` by `<unk>`, where n is used to control the NMT vocabulary size. The new test set reference translations with different vocabulary sizes are created and scored against the unmodified reference translation. The results are listed in Table 3.

NMT vocabulary size (k)	BLEU scores	no. of UNKs
10	82.4	2053
20	89.1	1247
30	91.5	971
32	91.8	932
40	93.1	790
50	93.9	693
60	94.4	640
70	94.9	576
80	95.3	533
90	95.7	490
100	95.8	481
110	96.0	459
original	100.0	0

Table 3: BLEU score and no. of UNK as a function of NMT vocabulary size

From the Table 3, it can be concluded that BLEU score increases when the NMT vocabulary size increases and the unknown size decreases.
b)
A script, named `countunk.py`, is created to count the number of UNKs introduced in the text file. It is run across the English side of the parallel texts with 32000 words and the word-level NMT outputs of Section 2.6.5 to extract their no. of UNKs as shown in 3^{rd} column of Table 4. All the text files are also scored, and the results are listed in 2^{nd} column of Table 4.
Additionally, the English side of the parallel texts for the test reference with 10k and 20k vocabularies and the corresponding outputs are created to test how much UNKs introduced in the same test parallel text affect the performance of the output.

Documents	BLEU scores	no. of UNKs
dev.w32k.ids.en	91.7	915
tune.w32k.ids.en	91.3	1170
test.w32k.ids.en	91.8	932
test.w10k.ids.en	82.4	2053
test.w20k.ids.en	89.1	1247
tfnmt.small.dev/out.w32k.text	11.8	948
tfnmt.small.tune/out.w32k.text	10.9	1165
tfnmt.small.test/out.w32k.text	15.5	1097
tfnmt.small.test/out.w10k.text	14.6	1704
tfnmt.small.test/out.w20k.text	15.2	1275

Table 4: Documents with different no. of UNKs and BLEU scores

With the test set references and the test set outputs, it can be observed that the decrease in UNKs can increase the BLEU performance.
However, even though the tune set output has less UNKs than the test set with limit of 32k vocabulary size, the tune set still has a lower BLEU score. Therefore, the decrease in UNKs does not necessarily guarantee the improvement of the BLEU score because of the context difference.

From the Table 3, it can be observed that when the decrease of UNKs is slowing down, the improvement of BLEU score is also slowing down.
Therefore, I suspect that there is a limit to further reduce UNKs no. in order to improve the BLEU score, since the decrease of UNKs is slowing down, and may increase the vocabulary size and the translation complexity which cannot be solved by the system. So at certain point, the NMT system should not keep reducing the no. of UNKs.
But the NMT system is still producing too many UNKs with the constraint of 32k vocabulary size since Table 3 shows that there are rooms for the UNKs to decrease.

Question 3

a)
A script, named `countneo.py`, is created to count the number of neologisms (words not from `wmap.en`) in a generated translation text file. The numbers of neologisms for different sets of outputs are listed in Table 5.

Documents	no. of neologisms
test.bpe/output	248
tune.bpe/output	251
dev.bpe/output	211

Table 5: No. of neologisms for different sets

248 neologisms are generated in the test set output by the BPE NMT system of Section 2.7.6.
Some of them are numbers (negative number like -867) which are not in the vocabulary list, some are numbers with units (such as km) and some are words joined together. Most of the new words generated are unreasonable, and some look more like the source language (Japanese language).

b)
Using the `w2bpe.fst` transducer, an acceptor which only accepts a particular set of bpe sequences of the english vocabularies can be created via its output projection. This output-projected transducer is shown in Figure 2.

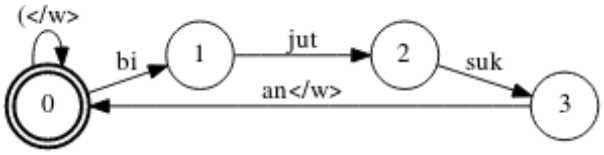


Figure 2: w2bpe transducer output projection

This guarantees to map the BPE output to word sequences from the English wordmap without any neologisms. This acceptor is named `bpenvocab.fst`.

c)
In the scripts `tfnmt.small.{test,tune,dev}`, the following changes are made to run the mapping mentioned in Q3b,

```
predictors: fst,t2t
src_test: $DIR/data/bpes/{test,tune,dev}.bpe.ids.ja
fst_path: /homes/XXXXX/MLMI8/bpenvocab.fst
```

```
use_fst_weights: True
predictor_weights: 0.5,0.5
```

The BPE NMT system now successfully maps the outputs without neologism, and the performances of the outputs are listed in line 8 of Table 1. When comparing the scores in Line 8 to Line 3 in Table 1, the BLEU scores decrease.
The decoding times of different sets with or without constraint are listed in Table 6.

Documents	decoding time (s)
dev with english constraint	249151
dev	56636.4
tune with english constraint	364118
tune	65730.9
test with english constraint	276847
test	104292

Table 6: Decoding times for different sets with or without english constraint

From Table 6, it can be concluded that the introduction of English constraint reduces the decoding speed and increases the required decoding time.

2 Lattice Minimum Bayes Risk Decoding

	NMT		HiFST WFSAs				BLEU		
ID	Word	BPE	type	Det/Min	Weighted	LMBR	Dev	Tune	Test
9.	m3						18.4	16.9	21.7
10.	m3		word			✓	18.1	16.4	21.5
11.		m3					17.7	16.3	21.7
12.		m3	bpe			✓	18.5	16.8	22.2

Table 7: KFTT Dev, Tune and Test BLEU Scores for NMT Decoding Strategies

Exercise 1

To compute lattice path posteriors $P(p|\mathcal{L})$, `utilfst.printstrings` can be used to print N shortest paths' strings and weights. Based on the N-best list's weights which is $-log$ probability, they can be normalized and scaled to give the path posterior as follows,
The posterior probability $P(p|\mathcal{L}) = P(E|F)$ of translation hypothesis $E \sim p$ give foreign source sentence $F \sim \mathcal{L}$ can by computed as [2],

$$P(E|F) = \frac{exp(\alpha H(E,F))}{\sum_{E' \in \epsilon} exp(\alpha H(E',F))} \tag{1}$$

, where $H(E,F)$ gives the score of candidate translation E and target F according to the model. The exponential scale factor α smoothes the posterior distribution, flattening when $\alpha < 1$ and sharpening when $\alpha > 1$. The scale factor α is very important since some scores in the lattice are uncomputeable due to numerical overflow.
The n-gram u posterior $p(u|\mathcal{L})$ is given by,

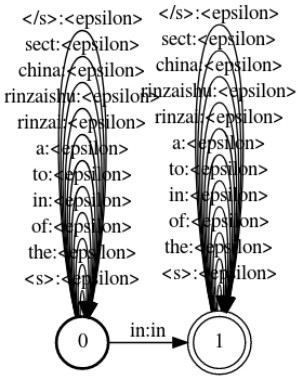
$$p(u|\mathcal{L}) = \sum_{p \in \mathcal{L}} 1\{u \in p\} P(p|\mathcal{L}) \tag{2}$$

For the first implementation 1.1, the basic python functions (e.g. `list`, `split`, etc.) are used to extract the n-gram options $\{u_i\}_{i=1}^{N}$ and compute $1\{u \in p\}$.
Alternatively, the n-gram posterior can be computed by,

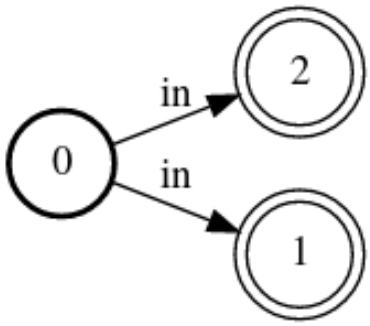
$$p(u|\mathcal{L}) \approx \sum_{p \in \mathcal{L}} \#\{u \in p\} P(p|\mathcal{L}) \tag{3}$$

For the second implementation 1.2, the counting transducer is utilized to extract $\#\{u \in p\}$. For example, one of the translation path in the test set $4.fst$ is `<s> in the rinzai sect in china`, to extract the unigram `in` in this path, the path acceptor and the counting transducer `in` are created as illustrated in Figure 3(a) and (b) respectively. When they are composed with each other, and remove-epsilon operation is applied, the number of states in the resulting fst can count the number of n-gram in the path as shown in Figure 3(c).

((a)) sentence acceptor



((b)) count transducer



((c)) count

Figure 3: Counting procedure

Both implementations also involve using the lattice path posteriors. The lattice path posteriors depend on the normalization factor. So there are two parameter settings are important. The first one is the number of shortest path N in the N-best list, since it determines the number of strings printed and the number of scores needed to be sum up, and $N = 5$ is chosen to reduce computing time.

The second one is the scale factor α. To generalize all the lattices, the scale factor is given by,

$$\alpha = \frac{1}{\max |H(p, \mathcal{L})|_{p \in \mathcal{L}}} \tag{4}$$

, so that the score is scaled by the maximum absolute printed score in the N-best lists.

Pruning helps to filter the arc with low probability (high score in log or tropical semirings). It may probably filter out some strings in the N-best lists, affect the value of scale factor, and thus change the path posterior probabilities. However, it has not been applied since it is difficult to find the suitable threshold for each lattice.

Determination, Mimination and pushing have been done to the lattices prior to any operations. However, these operations don't seem to affect the score extracted by printstrings.

With the settings of $N = 5$, 1st implementation takes only 762 seconds to extract all n-grams posterior in the first 100 development set lattices, and the 2nd one takes 1169 seconds. This shows 1st implementation is faster since the 2nd implementation requires fst operations, when the order of N-best list is high, the system becomes memory-intensive [3] and may slow down the process.

When the higher order of n-gram is chosen, $\#\{u \in p\} \approx 1\{u \in p\}$, so they yield the same results. This is because the smaller order of n-gram has a higher probability of being counted more than once in a path, so the n-gram posterior is more likely to be different. To verify the difference between two implementations, what matter are the indicator and count functions. One matrix $\mathbf{W_1}$ is created by taking the indicator functions $1\{u \in p\}$ as its elements so the row represents each n-gram option and the column represents each path. So this matrix

represents the 1st implementation,

$$[\mathbf{W_1}]_{up} = 1\{u \in p\} \tag{5}$$

Another matrix $\mathbf{W_2}$ with count functions $\#\{u \in p\}$ as elements which represents the 2nd implementation is created such that,

$$[\mathbf{W_2}]_{up} = \#\{u \in p\} \tag{6}$$

These two matrices allow us to compare the results of the implementations element by element. The number of mismatches between two matrices' elements can show the difference between implementations.

Exercise 2

The results of the word and subword baseline experiments with the transformer are listed in lines 9 and 11 of Table 7.

Exercise 3

In this exercise, the determinated, minimized and pushed lattices from Practical 1 exercise 2 and 3 are used. So those operations can be skipped before the use of printstrings. For the option of printstrings, the N-best list of order 5 is chosen. These choices are advantageous to reduce computing time.

So with the implementation 1.1, all the preliminary n-gram posteriors with 5-best lists are computed and they are smoothed by the following equation,

$$c(u) = 0.25 + 0.5 * p(u|\mathcal{L}) \tag{7}$$

They are stored in text files, the same pre-processing is applied for the bpe system.

The results of the word and subword experiments with the LMBR transformer are listed in lines 10 and 12 of Table 7.

Exercise 4

Using the implementation 1.1 allows us to compute all n-gram posteriors. The posteriors above the threshold are classified as n-grams hypothesis. If the hypothesis string is within the reference translation text, it can be counted as correct n-gram. The precision is given by the following equation,

$$Precision = \frac{correct}{hypothesised} \times 100\% \tag{8}$$

The number of n-grams hypothesised, n-grams correct and the precision are computed as a function of the confidence threshold for the first 1000 test set word lattices, and listed in Table 8.

Threshold c	n-grams hypothesised	n-grams correct	Precision (%)
0.1	109478	21479	19.6
0.2	91995	20699	22.5
0.3	85466	20391	23.9
0.4	77844	19824	25.5
0.5	70450	19299	27.4
0.6	67141	18920	28.2
0.7	60569	18253	30.1
0.8	58947	18018	30.6
0.9	52269	17143	32.8
1.0	41767	13902	33.3

Table 8: N-Gram precision for the first 1000 word lattices in the test set

As the threshold c is increased, both the numbers of hypothesized n-grams and correct n-grams decrease. As the decreasing speed of hypothesized n-grams number is faster than that of correct n-grams number, the precision increases when threshold decreases.

n-grams	1			2			3			4		
Threshold c	H	C	P	H	C	P	H	C	P	H	C	P
0.1	20012	10613	53.0	27751	5997	21.6	30245	3112	10.3	31470	1757	5.58
0.2	18796	10366	55.2	23972	5733	23.9	24674	2948	11.9	24553	1652	6.73
0.3	18370	10285	56.0	22645	5622	24.8	22601	2877	12.7	21850	1607	7.35
0.4	17638	10086	57.2	20787	5423	26.1	20222	2769	13.7	19197	1546	8.05
0.5	17046	9915	58.2	19081	5248	27.5	17865	2659	14.9	16458	1477	8.97
0.6	16647	9776	58.7	18198	5119	28.1	16852	2591	15.4	15444	1434	9.29
0.7	16024	9579	60.0	16559	4903	29.6	14807	2436	16.5	13179	1335	10.1
0.8	15764	9478	60.1	16124	4838	30.0	14341	2392	16.7	12718	1310	10.3
0.9	14930	9172	61.4	14396	4569	31.7	12327	2211	17.9	10616	1191	11.2
1.0	11925	7409	62.1	11507	3720	32.3	9857	1802	18.3	8478	971	11.5

Table 9: Different n-gram precision of the first 1000 word lattices in the test set. H: Hypothesised, C: Correct, P: Precision (%).

Similarly, the precisions for different n-gram orders also increase when the threshold is increased. It is also observed that the precision is higher for the lower n-gram order since the hypothesised number increases and the correct number decreases with the n-gram order. It is interesting to note that the precision change is the smallest when the threshold is increased from 0.7 to 0.8.

From Table 7, it is observed that there are score differences between Line 9 and 10. This implies the introduction of n-gram posteriors, precision percentage and ratio can affect the performance of the word system because they re-score the lattice paths. Thus, LMBR word decoding relies on the n-grams precision and precision ratio which makes the performance become worse.

Threshold c	n-grams hypothesised	n-grams correct	Precision (%)
0.1	111152	21964	19.8
0.2	93450	21174	22.7
0.3	86780	20876	24.1
0.4	79064	20302	25.7
0.5	71597	19773	27.6
0.6	68266	19387	28.4
0.7	61676	18728	30.4
0.8	60038	18489	30.8
0.9	53292	17622	33.1
1.0	41557	13703	33.0

Table 10: N-Gram precision for the first 1000 bpe lattices in the test set

The Table 10 shows the similar trend that the precision increases with the threshold. In general, the precisions in Table 10 are higher than those in Table 8, except the ones with the threshold $c = 1$.
From Table 7, it is observed that Line 12 has better scores than Line 11. This implies the introduction of n-gram posteriors, precision percentage and ratio can affect the performance of the bpe system because they re-score the lattice paths. Thus, LMBR bpe decoding relies on the n-grams precision and precision ratio which makes the performance become better.

References

[1] F. Stahlberg, E. Hasler, A. Waite, and B. Byrne, "Syntactically guided neural machine translation," *CoRR*, vol. abs/1605.04569, 2016.

[2] G. W. Blackwood, "Lattice rescoring methods for statistical machine translation," 2010.

[3] G. Blackwood, A. de Gispert, and W. Byrne, "Efficient path counting transducers for minimum bayes-risk decoding of statistical machine translation lattices," in *Proceedings of the ACL 2010 Conference Short Papers*, (Uppsala, Sweden), pp. 27–32, Association for Computational Linguistics, July 2010.