Multilingual NLP
Lab 3 – How Reliable is MT Evaluation?

AN Ji

December 17, 2024

1 Re-evaluating the role of BLEU in machine translation research

1.1 Implement the BLEU metric.

```python
import numpy as np
from collections import Counter

# n-gram generator
def ngrams(seq, n):
    """generate n-grams from a sequence using iterator & sliding window
    Parameters -- seq: str, n: int
    Returns -- generator of tuples or [] if seq size < n"""
    seq = seq.split()
    # extreme case: n > seq size
    if n > len(seq):
        return []
    # general case: n <= seq size
    seq_generator = iter(seq)
    history = []
    # initialize with first (n-1) items
    while n > 1:
        history.append(next(seq_generator))
        n -= 1
    # add next (1st item in remaining seq)
    for item in seq_generator:
        history.append(item)
        yield tuple(history)
        # delete oldest & move forward if any
        del history[0]


def compute_corpus_bleu(refs, hyps, n=4, floor=1e-10):
    """respectively aggregate sentence-level matching ngrams and individual sentence lengths
    Params -- refs: list of str, hyps: list of str, n: int, floor: float
    Returns -- float"""
    hyps_ngrams, bingo_ngrams = [0]*n, [0]*n
    refs_len, hyps_len = 0, 0  # rather than refs|hyps.split()

    for ref, hyp in zip(refs, hyps):
        # aggregate sentence length 1 by 1 to get corpus length
        refs_len += len(ref.split())
        hyps_len += len(hyp.split())

        for i in range(1, n+1):
            # aggregate 1 2 3 4-grams for each sentence pair
            ref_igrams, hyp_igrams = list(ngrams(ref, i)), list(ngrams(hyp, i))
            if hyp_igrams:
                bag_ref, bag_hyp = Counter(ref_igrams), Counter(hyp_igrams)
                bingo = sum((
                    min(bag_hyp[ngram], bag_ref.get(ngram, 0))
                    for ngram in bag_hyp
                ))
                bingo_ngrams[i-1] += bingo
                hyps_ngrams[i-1] += sum(bag_hyp.values())

```

```
    # set penalty
    bp = 1 if hyps_len > refs_len else np.exp(1 - refs_len / hyps_len)

    # calculate & smooth n-gram precisions
    precisions = [
        (bingo_ngrams[i] / hyps_ngrams[i]) if bingo_ngrams[i] > 0 else floor # no need to check hyps_ngrams[i]
        for i in range(n)
    ]

    return bp * np.exp(np.mean(np.log(precisions)))
```

1.2 Using the WMT'15 test sets, evaluate the performance of **mBart** and **MarianMT**. These two models can be easily used with the **HuggingFace** API. What can you conclude?

```
from bs4 import BeautifulSoup
from tqdm.notebook import tqdm
from transformers import MBartForConditionalGeneration, MBart50TokenizerFast, MarianMTModel, MarianTokenizer

src_fr = '/content/drive/MyDrive/Colab Notebooks/MT/newsdiscusstest2015-fren-src.fr.sgm'
ref_en = '/content/drive/MyDrive/Colab Notebooks/MT/newsdiscusstest2015-fren-ref.en.sgm'


def sgm2dict(src_path, ref_path):
    """parse .sgm files & extract sentences to dict"""
    corpus = dict()
    for filepath in [src_path, ref_path]:
        with open(filepath, "r") as sgm:
            soup = BeautifulSoup(sgm, "html.parser")
            sentences = [
                seg.get_text().strip()
                for seg in soup.find_all("seg")
            ]
            key = filepath[-10:-4]
            corpus[key] = sentences
    return corpus

corpus = sgm2dict(src_fr, ref_en)
sample = corpus['src.fr'][:100]


def mbart_trans(src, batch_size=16, src_lang='fr_XX', tgt_lang='en_XX'):
    """feed src sentences to mbart, return a list of translations"""
    # Load mBART model & tokenizer
    mbart = 'facebook/mbart-large-50-many-to-many-mmt'
    mbart_model = MBartForConditionalGeneration.from_pretrained(mbart)
    mbart_tokenizer = MBart50TokenizerFast.from_pretrained(mbart, src_lang=src_lang)

    mb_hyps = []
    with tqdm(total=len(src), desc="Translating batches", unit="sentence") as pbar:
        for i in range(0, len(src), batch_size):
            batch = src[i:i+16]
            inputs = mbart_tokenizer(
                batch,
                return_tensors="pt",
                padding=True,
                truncation=False
            )
            pbar.set_postfix_str(f"Processing batch {i // batch_size + 1}")
            outputs = mbart_model.generate(
                **inputs,
                forced_bos_token_id=mbart_tokenizer.lang_code_to_id[tgt_lang]
            )
            outputs_decoded = mbart_tokenizer.batch_decode(
                outputs,
                skip_special_tokens=True
            )
            pbar.update(len(batch))
            mb_hyps.extend(outputs_decoded)

    return mb_hyps

```

```python

def marian_trans(src, batch_size=32):
    """feed src sentences to marianMT, return a list of translations"""
    # Load MarianMT model & tokenizer
    marian = 'Helsinki-NLP/opus-mt-fr-en'
    marian_model = MarianMTModel.from_pretrained(marian)
    marian_tokenizer = MarianTokenizer.from_pretrained(marian)

    mr_hyps = []
    with tqdm(total=len(src), desc="Translating batches", unit="sentence") as pbar:
        for i in range(0, len(src), batch_size):
            batch = src[i:i+batch_size]
            inputs = marian_tokenizer(
                batch,
                return_tensors="pt",
                padding=True,
                truncation=False
            )
            pbar.set_postfix_str(f"Processing batch {i // batch_size + 1}")
            outputs = marian_model.generate(**inputs)
            outputs_decoded = [
                marian_tokenizer.decode(output, skip_special_tokens=True)
                for output in outputs
            ]
            pbar.update(len(batch))
            mr_hyps.extend(outputs_decoded)

    return mr_hyps


mbart_hyps = mbart_trans(sample)
marian_hyps = marian_trans(sample)
sample_dict = {
    'src.fr': sample,
    'ref.en': corpus['ref.en'][:100],
    'mbart_hyp.en': mbart_hyps,
    'marian_hyp.en': marian_hyps
}

# ds = Dataset.from_dict(sample_dict)

mbart_bleu = compute_corpus_bleu(sample_dict['ref.en'], sample_dict['mbart_hyp.en'])
marian_bleu = compute_corpus_bleu(sample_dict['ref.en'], sample_dict['marian_hyp.en'])

print(f"MBart BLEU: {mbart_bleu}")
print(f"MarianMT BLEU: {marian_bleu}")

[Out]:
MBart BLEU: 0.32006793786190674
MarianMT BLEU: 0.34034764664241274
```

The above test is based on French-English translation using WMT'15 `newsdiscusstest2015-fren` test set. I only choose the first 100 of all 1500 sentences due to lack of enough RAM on Colab.

The results indicate strong overlaps between reference sentences and both `mBart`'s and `MarianMT`'s translations. With very close BLEU scores, both models well preserve n-gram precision and length similarity on the sample. In practice, `MarianMT` works much faster than `mBart`, achieving slightly higher score by 0.03.

1.3 As noticed by [1], BLEU places no explicit constraints on the order that matching n-grams occur in. It is therefore possible, given a sentence, to generate many new sentences with at least as many n-gram matches by permuting words around bigram mismatches. Explain on an example why such permutations will never decrease the BLEU score.

From the implementation of our BLEU function we can see that BLEU is a surface and static evaluation that ONLY takes into account the matching n-grams between two sentences, regardless of their word order. In other words, the difference between BLEU scores of two sentences can hardly reflect how semantically and syntactically correct two sentences are, since there's no penalty designed for

the purposes of these aspects.

For example, as an English speaker can easily tell, the following rearranged hypotheses are far from correct, but they all have excellent BLEU scores based on our implementation (with $n = 2$). This case illustrates the uncorrelativeness between BLEU and human judgements, as well as its limit as an evaluation metric.

```
ref = "the quick brown fox jumps over the lazy dog and runs away"
hyp_1 = "the quick brown dog jumps over the lazy fox and runs away"
hyp_2 = "the lazy dog jumps over the quick brown fox and runs away"

hyp_1 BLEU: 0.7977240352174656
hyp_2 BLEU: 0.9045340337332909
```

1.4 Given a sentence with n words and b bigram mismatches, how many sentences can you generate with this principle. Compute the number of sentences you will obtain on the WMT'15 test set.

Since bigrams are not necessarily distinct within a hypothesis sentence, we CANNOT directly calculate the number of matching bigrams given b mismatches. So here we switch to considering there are b **matching bigrams**.

In this updated case, if a given hypothesis has k words, from Callison-Burch et al. [1] we know that the number of identically scored sentences generated from the given hypothesis can be computed using the following formula:

$$\text{Number of identically scored sentences} = (k - b)!$$

```
# Number of permutated sentences generated from both model hypotheses
# based on our sample of size 100

def fake_hyps(refs, hyps):
    """given an mbart/marian hypothesis, calculate the number of permutated sentences that score
        identically"""
    total = 0
    for ref, hyp in zip(refs, hyps):
        ref_bigrams = set(ngrams(ref, 2))
        hyp_bigrams = set(ngrams(hyp, 2))
        b = len(hyp_bigrams & ref_bigrams) # number of matched bigrams
        k = len(hyp.split()) # number of words in hyp
        total += math.factorial(k - b)  # (k-b)! of current hyp
    return f'{total:.2e}'

fake_hyps(refs, mbart_hyps)

[Out]:
'1.03e+40'

fake_hyps(refs, marian_hyps)

[Out]:
'8.23e+33'
```

We will obtain 1.03e+40 permutated sentences for the 100 hypotheses of `mBart` and 8.23e+33 for those of `MarianMT`.

1.5 Why does this result question the use of BLEU as an evaluation metric.

As mentionned in 1.3, there are lots of syntactic and semantic aspects that BLEU is incapable to account for. On the one hand, it is possible to have sentences which receive identical Bleu scores but are judged by humans to be worse; on the other hand, it is also possible to have a higher Bleu score without any genuine improvement in translation quality [1].

1.6 SACREBLEU is an implementation of BLEU that aims to provide "hassle-free computation of shareable, comparable, and reproducible BLEU scores". Evaluate the two previous systems using SACREBLEU. What can you conclude?

```
!pip install sacrebleu

sacre_mbart = sacrebleu.corpus_bleu(mb_hyps, [refs], lowercase=True)
sacre_marian = sacrebleu.corpus_bleu(mr_hyps, [refs], lowercase=True)

print(f"MBart sacreBLEU: {sacre_mbart}")
print(f"MarianMT sacreBLEU: {sacre_marian}")

# Recall
# MBart BLEU: 0.32006793786190674 (multiplied by 100 => 32.01)
# MarianMT BLEU: 0.34034764664241274 (multiplied by 100 => 34.03)

[Out]:
MBart sacreBLEU: BLEU = 36.17 65.2/41.7/29.3/21.5 (BP = 1.000 ratio = 1.027 hyp_len = 1732 ref_len
    = 1686)

MarianMT sacreBLEU: BLEU = 38.13 67.2/44.7/31.3/22.5 (BP = 1.000 ratio = 1.023 hyp_len = 1725
    ref_len = 1686)
```

We notice that both SACREBLEU scores are **4 points higher** than our BLEU scores. This may be explained by various strategies used in both methods.

First, SACREBLEU apply normalization techniques to both references and hypotheses, such as removing special characters and punctuation, and we also pass `lowercasing=True` in the scoring function, which we did not apply in our BLEU implementation, and this does allow SACREBLEU to recognize more matching bigrams (which we have verified by switching the `lowercasing` setting) and return higher scores.

A second reason may be that SACREBLEU has its own tokenization methods, while we only used `.split()` to produce tokens based on spaces, which is not as robust as the former. For example if token `HIV` occurs in a reference, then token `HIV.` in a hypothesis wouldn't be taken into account when counting unigram matches using our BLEU function, while the normalization step allows SACREBLEU to tokenize `HIV.` as `HIV`, thus counting it as a matching unigram and yielding higher precision.

1.7 Using SACREBLEU and your own implementation of BLEU compute the score achieved:

- when considering the "raw" translation hypotheses and references;
- when the translation hypotheses and references have been tokenized in subword units;
- when the translation hypotheses and references have been tokenized in characters (this amounts to adding a space between each character of the references and of the translation hypotheses).

How can you explain these results?

```
def tokenize_corpus(sentences, method):
    if method == 'subword':
        tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-cased")
        updated_sentences = [' '.join(tokenizer.tokenize(sent)) for sent in sentences]
    elif method == 'character':
        updated_sentences = [' '.join(list(sent)) for sent in sentences]
    else: updated_sentences = sentences
    return updated_sentences

original_sample = {
    'refs': refs,
    'mbart': mb_hyps,
    'marian': mr_hyps
}

methods = ['raw','subword','character']

```

```
# Dict for tokenized sample
tokenized_sample = {}
for method in methods:
    for k, v in original_sample.items():
        tokenized_sample[f'{method}_{k}'] = tokenize_corpus(v, method)

scores = {
    ('mbart', 'bleu'): [],
    ('mbart', 'sacrebleu'): [],
    ('marian', 'bleu'): [],
    ('marian', 'sacrebleu'): [],
}

# bleu
for model in ['mbart', 'marian']:
    for method in methods:
        scores[(model, 'bleu')].append(
            float("{:.2f}".format(
                compute_corpus_bleu(
                    tokenized_sample[f'{method}_refs'],
                    tokenized_sample[f'{method}_{model}']
                ) * 100     # for convenient comparison
            ))
        )

# sacrebleu
for model in ['mbart', 'marian']:
    for method in methods:
        scores[(model, 'sacrebleu')].append(
            float("{:.2f}".format(
                sacrebleu.corpus_bleu(
                    tokenized_sample[f'{method}_{model}'],
                    [tokenized_sample[f'{method}_refs']],
                    # tokenize='none'
                ).score
            ))
        )

scores

[Out]:
{('mbart', 'bleu'): [32.01, 38.15, 63.37],
('mbart', 'sacrebleu'): [35.05, 43.34, 63.37],
('marian', 'bleu'): [34.03, 40.33, 65.95],
('marian', 'sacrebleu'): [37.02, 46.21, 65.95]}
```

model	metric	raw	subword	character
mbart	bleu	32.01	38.15	63.37
	sacrebleu	**35.05**	**43.34**	63.37
marian	bleu	34.03	40.33	65.95
	sacrebleu	**37.02**	**46.21**	65.95

Table 1: Scores under various tokenization strategies using default settings of SACREBLEU

model	metric	raw	subword	character
mbart	bleu	32.01	38.15	63.37
	sacrebleu	**32.01**	**38.15**	63.37
marian	bleu	34.03	40.33	65.95
	sacrebleu	**34.03**	**40.33**	65.95

Table 2: Scores under various tokenization strategies using `tokenize='none'` in SACREBLEU

From Table 1 we notice that when tokens become more and more atomized, both BLEU and SACREBLEU scores will be raised considerably, and they become identical at character-level tokenization.

In SACREBLEU we also tried disabling entirely the default tokenization method[1] (`tokenize='none'`). Table 2 shows how SACREBLEU scores change accordingly.

It is interesting that SACREBLEU returns the same scores as those of BLEU on all situations. Based on this observation we can conclude that without more tokenization, both metrics rely on spaces as token boundaries and thus leading to identical scores. The application of default **13a** tokenization (Table 1) implies more matching n-grams, obviously raising scores by about 3 points on raw cases and about 5 points on subword cases for both models' hypotheses.

References

[1] Callison-Burch, C., Osborne, M., and Koehn, P. (Apr. 2006). "Re-evaluating the Role of Bleu in Machine Translation Research". In: *11th Conference of the European Chapter of the Association for Computational Linguistics*. Ed. by McCarthy, D. and Wintner, S. Trento, Italy: Association for Computational Linguistics, pp. 249–256. URL: https://aclanthology.org/E06-1032.

[2] Post, M. (Oct. 2018). "A Call for Clarity in Reporting BLEU Scores". In: *Proceedings of the Third Conference on Machine Translation: Research Papers*. Belgium, Brussels: Association for Computational Linguistics, pp. 186–191. URL: https://www.aclweb.org/anthology/W18-6319.

[1] SACREBLEU's tokenization method defaults to '13a'. See their GitHub documentation.