Lab 3: How Reliable is MT Evaluation?
Multilingual NLP

Manon Fleury
22410952

Re-evaluating the role of Bleu in machine translation research

1. Implement the BLEU metric

```python
from collections import Counter
import math

def compute_bleu(candidate, reference, n=4):
    candidate_tokens = candidate.split()
    reference_tokens = reference.split()
    candidate_len = len(candidate.split())
    reference_len = len(reference.split())

    precisions = []
    for i in range(1, n+1): # creation of the n-grams
        candidate_ngrams = Counter([tuple(candidate_tokens[j:j+i]) for j
            in range(candidate_len-i+1)])
        reference_ngrams = Counter([tuple(reference_tokens[j:j+i]) for j
            in range(reference_len-i+1)])
        # Compute the correspondances of the n-grams
        overlap = sum(min(count, reference_ngrams[ngram]) for ngram, count
             in candidate_ngrams.items())
        total = sum(candidate_ngrams.values())
        precisions.append(overlap / total if total > 0 else 0)

    # Compute the penalty factor for shorter sentences
    brevity_penalty = math.exp(1 - reference_len / candidate_len) if
        candidate_len < reference_len else 1
    bleu_score = brevity_penalty * math.exp(sum(math.log(p) for p in
        precisions if p > 0) / n)
    return bleu_score

# Example
candidate_sentence = "the cat is on the mat"
reference_sentence = "there is a cat on the mat"
bleu = compute_bleu(candidate_sentence, reference_sentence)
print(f"BLEU Score: {bleu}")
```

```
BLEU Score: 0.45
```

2. Using the WMT'15 test sets, evaluate the performance of mBart and MarianMT. What can you conclude?

We want to compare the translations made by mBART and marianMT to the ones written in the set we're using. So we're evaluating their performance, to see if their translations are of good quality.

```python

from datasets import load_dataset
wmt15 = load_dataset("wmt15", "de-en", split="test")
print('Dataset example structure:',wmt15[0])

from transformers import MBartForConditionalGeneration, MBart50Tokenizer,
    MarianMTModel, MarianTokenizer
import numpy as np

mbart_model = MBartForConditionalGeneration.from_pretrained("facebook/
    mbart-large-50-many-to-many-mmt")
mbart_tokenizer = MBart50Tokenizer.from_pretrained("facebook/mbart-large
    -50-many-to-many-mmt")
mbart_tokenizer.src_lang = "de_DE"

marian_model = MarianMTModel.from_pretrained("Helsinki-NLP/opus-mt-de-en")
marian_tokenizer = MarianTokenizer.from_pretrained("Helsinki-NLP/opus-mt-
    de-en")

def mbart_translate(text):
  inputs = mbart_tokenizer(text, return_tensors="pt")
  outputs = mbart_model.generate(**inputs)
  return mbart_tokenizer.decode(outputs[0], skip_special_tokens=True)

def marian_translate(text):
    inputs = marian_tokenizer(text, return_tensors="pt")
    outputs = marian_model.generate(**inputs)
    return marian_tokenizer.decode(outputs[0], skip_special_tokens=True)

# Compute bleu score
mbart_scores = []
marian_scores = []

for sample in wmt15["translation"][:50]:
  german_text = sample["de"]
  reference_translation = sample["en"]

  mbart_translation = mbart_translate(german_text)
  mbart_bleu = compute_bleu(mbart_translation, reference_translation)
  mbart_scores.append(mbart_bleu)

  marian_translation = marian_translate(german_text)
  marian_bleu = compute_bleu(marian_translation, reference_translation)
  marian_scores.append(marian_bleu)

# Compute the average BLEU scores for each model
mbart_avg_bleu = np.mean(mbart_scores)
marian_avg_bleu = np.mean(marian_scores)

```

```python
print(f"Mean BLEU score for mBART : {mbart_avg_bleu}")
print(f"Mean BLEU score for MarianMT : {marian_avg_bleu}")
```

```
Mean BLEU score for mBART: 0.4219
Mean BLEU score for MarianMT: 0.4032
```

These BLEU scores are in the range of 0.40 that indicates that the translations are fairly good but not perfect, they capture many of the essential n-grams from the reference sentences but likely still contain noticeable errors or less natural phrasing.

mBART achieves a slightly higher score (0.42) compared to MarianMT (0.40). This suggests that, for this subset of the WMT'15 German-to-English test set, mBART generally produces translations that more closely match the reference translations than MarianMT. However, the difference in scores is modest, indicating that both models are fairly comparable in performance.

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.

3. Explain on an example why such permutations will never decrease the Bleu score.

BLEU allows flexibility in word order because it does not impose strict constraints on the sequence of matching n-grams, so as long as the same n-grams appear in the candidate translation, their exact order is not critical to the score.

For example,
Reference: "The cat is on the mat."
Candidate 1: "The cat is on the mat."
Candidate 2: "The mat is on the cat."

Both candidates contain the same words and bigrams as the reference sentence.
Candidate 1 is a perfect match and would receive a high BLEU score.
Candidate 2, however, has permuted the words around the bigrams "the mat" and "the cat," resulting in a different order but retaining all of the same bigrams.

In the two candidates we have the same unigrams and bigrams: Unigrams: "the," "cat," "is," "on," "mat."
Bigrams: "the cat," "cat is," "is on," "on the," "the mat."
So the BLEU score will remain similar, it will not decrease, since BLEU only focuses on the quantity of common n-grams, not their exact order.

This limitation questions BLEU's effectiveness in evaluating translation fluency and sentence structure, especially in cases with reordered or permuted words.

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.

Given a sentence with n words and b bigram mismatches, we can generate 2^b sentences, since each bigram can be permuted in 2 ways ('A B', 'B A'), independently from the other bigrams.

```python
import nltk
from nltk.util import bigrams

nltk.download('punkt')
nltk.download('punkt_tab')

def get_bigrams(text):
    tokens = nltk.word_tokenize(text)
    return set(bigrams(tokens))

def count_incorrect_bigrams(candidate, reference):
    print('len(candidate):',len(candidate.split()))
    print('len(reference):',len(reference.split()))
    candidate_bigrams = get_bigrams(candidate)
    reference_bigrams = get_bigrams(reference)
    # The incorrect bigrams are the ones in the candidate but not in the
       reference
    incorrect_bigrams = candidate_bigrams - reference_bigrams
    return len(incorrect_bigrams)

mismatch_bigrams = []
for sample in wmt15["translation"][:5]:
  german_text = sample["de"]
  reference_translation = sample["en"]

  mbart_translation = mbart_translate(german_text)
  mismatch_bigrams.append(count_incorrect_bigrams(mbart_translation,
      reference_translation))
print(mismatch_bigrams) # [7, 21, 5, 8, 11]

for b in mismatch_bigrams:
  print(2**b) # 128, 2097152, 32, 256, 2048
```

5. Why does this result question the use of Bleu as an evaluation metric?

We see that we can have very different length of sentences possible from a permutation around mismatch bigrams. As a consequence, 128 possible sentences for instance would not have the same meaning of the original sentence with just permutation. And since we said that BLEU only focuses on the quantity of common n-grams, not their exact order, BLEU will attribute a similar score for these 128 sentences even though they are syntactically or semantically very different from the original, which questions its reliability as an evaluation metric for translation quality.

In addition to this flaw in its very conception, the practical implementation of Bleu poses many problems related in particular to the tokenization.

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?

```
# we call the mBART and MarianMT models and the mbart_translate and
   marian_translate functions made previously

from sacrebleu.metrics import BLEU

bleu_metric = BLEU()

mbart_translations = []
```

```python
marian_translations = []
references = []

for sample in wmt15["translation"][:50]:
    german_text = sample["de"]
    reference_translation = sample["en"]

    references.append(reference_translation)

    mbart_translation = mbart_translate(german_text)
    mbart_translations.append(mbart_translation)

    marian_translation = marian_translate(german_text)
    marian_translations.append(marian_translation)

# Compute BLEU scores with SacreBLEU
mbart_bleu_score = bleu_metric.corpus_score(mbart_translations, [
    references])
marian_bleu_score = bleu_metric.corpus_score(marian_translations, [
    references])

print(f"BLEU score (SacreBLEU) for mBART: {mbart_bleu_score.score}")
print(f"BLEU score (SacreBLEU) for MarianMT: {marian_bleu_score.score}")
```

```
BLEU score (SacreBLEU) for mBART: 39.7
BLEU score (SacreBLEU) for MarianMT: 37.9
```

The difference between the BLEU scores computed manually and those obtained using SacreBLEU can be attributed to several factors. SacreBLEU applies stricter text normalization (e.g., lowercasing, handling of special characters) and calculates scores at the corpus level, while the manual implementation averages sentence-level scores. Additionally, SacreBLEU imposes a stricter brevity penalty for shorter translations and handles missing n-grams more rigorously. These factors often result in lower, but more realistic, BLEU scores with SacreBLEU. Therefore, the lower SacreBLEU scores for mBART (39.67) and MarianMT (37.89) compared to the manual BLEU (42.19 and 40.32, respectively) highlight the potential overestimation in manual calculations and confirm the importance of using SacreBLEU for reliable and reproducible evaluations.

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?

For that, we are going to use mBART.

```python
from sacrebleu import corpus_bleu


# Character-level tokenization
def tokenize_to_characters(text):
    return " ".join(list(text))
```

```python

# Subword-level tokenization using MBart's tokenizer
def tokenize_to_subwords(text, tokenizer):
    tokens = tokenizer.tokenize(text)
    return " ".join(tokens)


# Raw input
raw_hypotheses = [mbart_translate(sample["de"]) for sample in wmt15["translation"][:50]]
raw_references = [sample["en"] for sample in wmt15["translation"][:50]]

# Subword tokenization
subword_hypotheses = [tokenize_to_subwords(hyp, mbart_tokenizer) for hyp in raw_hypotheses]
subword_references = [tokenize_to_subwords(ref, mbart_tokenizer) for ref in raw_references]

# Character-level tokenization
char_hypotheses = [tokenize_to_characters(hyp) for hyp in raw_hypotheses]
char_references = [tokenize_to_characters(ref) for ref in raw_references]

# Compute BLEU scores for each case
def compute_scores(hypotheses, references):
    # Using custom BLEU
    bleu_scores = [compute_bleu(hyp, ref) for hyp, ref in zip(hypotheses, references)]
    avg_bleu = np.mean(bleu_scores)

    # Using SacreBLEU
    sacre_bleu = corpus_bleu(hypotheses, [references]).score
    return avg_bleu, sacre_bleu

# Raw
raw_avg_bleu, raw_sacre_bleu = compute_scores(raw_hypotheses, raw_references)
raw_sacre_bleu /= 100  # Dividing by 100 to match BLEU scale

# Subwords
subword_avg_bleu, subword_sacre_bleu = compute_scores(subword_hypotheses, subword_references)
subword_sacre_bleu /= 100

# Characters
char_avg_bleu, char_sacre_bleu = compute_scores(char_hypotheses, char_references)
char_sacre_bleu /= 100

print(f"{'':<10} {'BLEU custom':<15} {'SacreBLEU':<10}")
print(f"{'Raw':<10} {raw_avg_bleu:<15.4f} {raw_sacre_bleu:<10.4f}")
print(f"{'Subword':<10} {subword_avg_bleu:<15.4f} {subword_sacre_bleu:<10.4f}")
print(f"{'Character':<10} {char_avg_bleu:<15.4f} {char_sacre_bleu:<10.4f}")
```

	BLEU custom	SacreBLEU
Raw	0.4219	0.3967
Subword	0.4538	0.4424
Character	0.6907	0.7161

The results show that different tokenization methods significantly affect the BLEU scores. Raw tokenization yields moderate scores, as expected, since exact matches between raw translations and references are often limited. Subword tokenization improves BLEU slightly, reflecting better handling of rare or unknown words. However, the most noticeable improvement comes from character-level tokenization, which leads to significantly higher BLEU scores, indicating that this finer granularity can capture subtle translation variations. These findings suggest that character-level tokenization is particularly effective in handling morphological complexity or rare words, while subword tokenization strikes a good balance between granularity and efficiency. The consistency between your custom BLEU calculation and SacreBLEU further confirms the robustness of my implementation.