Assignment 5 - Natural language processing

February 20, 2024

1 Group 5 - Module 5: Natural Language Processing

1.0.1 Group Members:

- **Nils Dunlop, 20010127-2359, Applied Data Science, e-mail: gusdunlni@student.gu.se (16 hours)**
- **Francisco Erazo, 19930613-9214, Applied Data Science, e-mail: guserafr@student.gu.se (16 hours)**

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions." (This is independent and additional to any declaration that you may encounter in the electronic submission system.)

2 Assignment 5

2.1 Problem 1: Reading and Reflection

2.1.1 (A) AI Problems with a Wide Range of Approaches:

Several AI problems have seen a wide range of approaches, reflecting the diverse computational techniques utilized over the decades. Notable examples include:

- **Speech Recognition:** This field has evolved significantly from early phonetic-based systems to modern deep learning approaches. Initially, systems relied on simple pattern matching and rule-based systems to understand spoken language. Over time, statistical models, especially Hidden Markov Models (HMMs), played a crucial role. Currently, deep neural networks, particularly recurrent neural networks (RNNs) and convolutional neural networks (CNNs) dominate the field.
- **Computer Vision:** The evolution of techniques from simple edge detection algorithms and feature extraction methods to sophisticated deep learning models is noteworthy. Techniques

have ranged from geometric model fitting and template matching to the use of deep convolutional neural networks (CNNs) for tasks such as image classification, object detection and semantic segmentation.

- **Game Playing:** The development of AI for game playing, from chess and checkers to complex video games, has evolved from brute-force search algorithms and heuristic-based approaches to the use of machine learning techniques, including reinforcement learning and deep learning for strategy optimization.

2.1.2 (B) Similarities Between Rule-based Translation Systems and Neural Systems:

Despite their differences, rule-based translation systems and state-of-the-art neural systems share notable similarities:

- **Structured Knowledge:** Rule-based systems are explicitly programmed with linguistic rules and dictionaries, while neural systems, especially those employing attention mechanisms, implicitly learn structured knowledge about languages through training on large datasets.
- **Context Handling:** Both systems aim to handle context in translation, however, in largely different ways. Rule-based systems may use context rules to choose the correct translation based on surrounding words. Neural systems, particularly those using attention mechanisms, can consider broader contexts in a more fluid and dynamic manner.
- **Error Correction:** Both systems have mechanisms for error correction, with neural systems doing this implicitly. Rule-based systems may use post-processing rules to correct common mistakes, while neural systems learn error patterns and corrections during training.

2.1.3 (C) Situations Favoring Rule-based Solutions Over Modern Approaches:

There are scenarios where a rule-based solution might be preferable to a modern neural or statistical approach:

- **Limited Data Scenarios:** For languages or specialized domains where there is a scarcity of training data, rule-based systems can be more practical and effective because they do not require large datasets to perform reasonably well.
- **Predictability and Transparency:** In situations where predictability, transparency, and the ability to audit or explain decisions are crucial (e.g., in some legal or regulatory contexts), rule-based systems offer clear advantages. Their decisions are based on explicit rules that can be examined, understood and modified by humans.
- **Real-time Constraints:** When computational resources are limited or real-time performance is essential, the simpler and less resource-intensive rule-based systems can have an advantage over more complex neural models.

2.2 Problem 2: Implementation

2.2.1 (A) Warmup: Word Frequencies

---

[1]:
```
import pandas as pd
from collections import Counter
import re
import numpy as np
from collections import defaultdict

# Load datasets into DataFrames
# The .squeeze() method is used to convert the single-column DataFrames into␣
 ↪Series
de_en_df = pd.DataFrame({
    'German': pd.read_csv("dat410_europarl/europarl-v7.de-en.lc.de", sep='\n',␣
 ↪header=None, names=['text']).squeeze(),
    'English_DE': pd.read_csv("dat410_europarl/europarl-v7.de-en.lc.en",␣
 ↪sep='\n', header=None, names=['text']).squeeze()
})

fr_en_df = pd.DataFrame({
    'French': pd.read_csv("dat410_europarl/europarl-v7.fr-en.lc.fr", sep='\n',␣
 ↪header=None, names=['text']).squeeze(),
    'English_FR': pd.read_csv("dat410_europarl/europarl-v7.fr-en.lc.en",␣
 ↪sep='\n', header=None, names=['text']).squeeze()
})

sv_en_df = pd.DataFrame({
    'Swedish': pd.read_csv("dat410_europarl/europarl-v7.sv-en.lc.sv", sep='\n',␣
 ↪header=None, names=['text']).squeeze(),
    'English_SV': pd.read_csv("dat410_europarl/europarl-v7.sv-en.lc.en",␣
 ↪sep='\n', header=None, names=['text']).squeeze()
})

# Display the first few rows of the German-English dataset
print(de_en_df.head())
```

```
                                              German  \
0  ich erkläre die am freitag , dem 17. dezember …
1  wie sie feststellen konnten , ist der gefürcht…
2  im parlament besteht der wunsch nach einer aus…
3  heute möchte ich sie bitten - das ist auch der…
4  wie sie sicher aus der presse und dem fernsehe…

                                          English_DE
0  i declare resumed the session of the european …
1  although , as you will have seen , the dreaded…
2  you have requested a debate on this subject in…
3  in the meantime , i should like to observe a m…
4  you will be aware from the press and televisio…
```

[2]:
```
# Display the first few rows of the French-English dataset
print(fr_en_df.head())
```

```
                                              French  \
0  je déclare reprise la session du parlement eur…
1  comme vous avez pu le constater , le grand &qu…
2  vous avez souhaité un débat à ce sujet dans le…
3  en attendant , je souhaiterais , comme un cert…
4  je vous invite à vous lever pour cette minute …

                                          English_FR
0  i declare resumed the session of the european …
1  although , as you will have seen , the dreaded…
2  you have requested a debate on this subject in…
3  in the meantime , i should like to observe a m…
4  please rise , then , for this minute &apos; s …
```

[3]:
```
# Display the first few rows of the Swedish-English dataset
print(sv_en_df.head())
```

```
                                              Swedish  \
0  jag förklarar europaparlamentets session återu…
1  som ni kunnat konstatera ägde &quot; den stora…
2  ni har begärt en debatt i ämnet under sammantr…
3  till dess vill jag att vi , som ett antal koll…
4             jag ber er resa er för en tyst minut .

                                          English_SV
0  i declare resumed the session of the european …
1  although , as you will have seen , the dreaded…
2  you have requested a debate on this subject in…
3  in the meantime , i should like to observe a m…
4  please rise , then , for this minute &apos; s …
```

[4]:
```
# Function to calculate word frequencies
def calculate_word_frequencies(df_column):
    word_counts = Counter()
    for text in df_column:
        # Remove punctuation and other non-word characters
        words = re.findall(r'\b\w+\b', text.lower())
        word_counts.update(words)
    return word_counts
```

[5]:
```
# Calculate word frequencies for each language
de_word_counts = calculate_word_frequencies(de_en_df['German'])
en_de_word_counts = calculate_word_frequencies(de_en_df['English_DE'])

fr_word_counts = calculate_word_frequencies(fr_en_df['French'])
```

```
en_fr_word_counts = calculate_word_frequencies(fr_en_df['English_FR'])

sv_word_counts = calculate_word_frequencies(sv_en_df['Swedish'])
en_sv_word_counts = calculate_word_frequencies(sv_en_df['English_SV'])
```

[6]:
```
# Function to print most common words
def print_most_common(word_counts, language, num=10):
    print(f"Most common words in {language}:")
    for word, count in word_counts.most_common(num):
        print(f"{word}: {count}")
    print("\n")
```

[7]:
```
# Print the 10 most frequent words for each language
print_most_common(de_word_counts, "German")
print_most_common(en_de_word_counts, "English (DE)")

print_most_common(fr_word_counts, "French")
print_most_common(en_fr_word_counts, "English (FR)")

print_most_common(sv_word_counts, "Swedish")
print_most_common(en_sv_word_counts, "English (SV)")
```

```
Most common words in German:
die: 10521
der: 9374
und: 7028
in: 4175
zu: 3169
den: 2976
wir: 2863
daß: 2738
ich: 2670
das: 2669


Most common words in English (DE):
the: 19853
of: 9633
to: 9069
and: 7307
in: 6278
is: 4478
that: 4441
a: 4438
we: 3372
this: 3362
```

```
Most common words in French:
apos: 16729
de: 14528
la: 9746
et: 6620
l: 6536
le: 6177
à: 5588
les: 5587
des: 5232
que: 4797


Most common words in English (FR):
the: 19627
of: 9534
to: 8992
and: 7214
in: 6197
is: 4453
that: 4421
a: 4388
we: 3341
this: 3332


Most common words in Swedish:
att: 9181
och: 7038
i: 5954
det: 5687
som: 5028
för: 4959
av: 4013
är: 3840
en: 3724
vi: 3211


Most common words in English (SV):
the: 19327
of: 9344
to: 8814
and: 6949
in: 6124
is: 4400
that: 4357
a: 4271
```

```
we: 3223
this: 3222
```

[8]:
```python
def print_probabilities(word_counts, language, total_words):
    for word in ['speaker', 'zebra']:
        if word in word_counts:
            prob = word_counts[word] / total_words
        else:
            prob = 0
        print(f"Probability of '{word}' in {language}: {prob:.6f}")

total_de_words = sum(de_word_counts.values())
total_en_de_words = sum(en_de_word_counts.values())
total_fr_words = sum(fr_word_counts.values())
total_en_fr_words = sum(en_fr_word_counts.values())
total_sv_words = sum(sv_word_counts.values())
total_en_sv_words = sum(en_sv_word_counts.values())

print_probabilities(de_word_counts, "German", total_de_words)
print_probabilities(en_de_word_counts, "English (DE)", total_en_de_words)
print_probabilities(fr_word_counts, "French", total_fr_words)
print_probabilities(en_fr_word_counts, "English (FR)", total_en_fr_words)
print_probabilities(sv_word_counts, "Swedish", total_sv_words)
print_probabilities(en_sv_word_counts, "English (SV)", total_en_sv_words)
```

```
Probability of 'speaker' in German: 0.000000
Probability of 'zebra' in German: 0.000000
Probability of 'speaker' in English (DE): 0.000042
Probability of 'zebra' in English (DE): 0.000000
Probability of 'speaker' in French: 0.000000
Probability of 'zebra' in French: 0.000000
Probability of 'speaker' in English (FR): 0.000046
Probability of 'zebra' in English (FR): 0.000000
Probability of 'speaker' in Swedish: 0.000000
Probability of 'zebra' in Swedish: 0.000000
Probability of 'speaker' in English (SV): 0.000039
Probability of 'zebra' in English (SV): 0.000000
```

The output indicates that the word ‘speaker’ is found in the English versions of the texts, specifically English (DE), English (FR), and English (SV), with respective probabilities of 0.000042, 0.000046, and 0.000039. This suggests a limited presence of ‘speaker’ only within these English translations. In contrast, the absence of ‘speaker’ in German, French, and Swedish, as shown by a 0 probability, confirms its unavailability in these languages.

Regarding the word ‘zebra’, its 0 probability across all examined languages implies that it does not feature in any of the texts. This reveals that ‘speaker’ is present solely in the English corpora, while ‘zebra’ is not found in any of the languages analyzed.

2.2.2 (B) Language Modeling

---

[9]:
```
# Tokenize the text and add start and end tokens
# The words between the start and end markers are determined and tokenized
tokenized_text_de = [('<START>',) + tuple(re.findall(r'\b\w+\b', sentence.
 ↪lower())) + ('<END>',) for sentence in de_en_df['English_DE']]
tokenized_text_fr = [('<START>',) + tuple(re.findall(r'\b\w+\b', sentence.
 ↪lower())) + ('<END>',) for sentence in fr_en_df['English_FR']]
tokenized_text_sv = [('<START>',) + tuple(re.findall(r'\b\w+\b', sentence.
 ↪lower())) + ('<END>',) for sentence in sv_en_df['English_SV']]

# Count individual words and bigrams
word_counts_de = Counter()
bigram_counts_de = Counter()
word_counts_fr = Counter()
bigram_counts_fr = Counter()
word_counts_sv = Counter()
bigram_counts_sv = Counter()

# Update word counts and form bigram tuples of the words
for sentence in tokenized_text_de:
    word_counts_de.update(sentence)
    bigram_counts_de.update(zip(sentence[:-1], sentence[1:]))

for sentence in tokenized_text_fr:
    word_counts_fr.update(sentence)
    bigram_counts_fr.update(zip(sentence[:-1], sentence[1:]))

for sentence in tokenized_text_sv:
    word_counts_sv.update(sentence)
    bigram_counts_sv.update(zip(sentence[:-1], sentence[1:]))
```

[10]:
```
# Compute the probability of a sentence using a bigram model with Laplace␣
 ↪smoothing
def bigram_sentence_probability(sentence, word_counts, bigram_counts,␣
 ↪total_words, smoothing=1):
    sentence = ('<START>',) + tuple(re.findall(r'\b\w+\b', sentence.lower())) +␣
 ↪('<END>',)
    bigram_probs = []

    for first_word, second_word in zip(sentence[:-1], sentence[1:]):
        bigram_count = bigram_counts[(first_word, second_word)]
        word_count = word_counts[first_word]

        # Apply Laplace smoothing
```

```
        prob = (bigram_count + smoothing) / (word_count + smoothing *␣
 ↪(total_words + 1))
        bigram_probs.append(prob)

    # Logarithms is used to avoid underflow with long sentences
    log_probs = np.log(bigram_probs)
    log_prob_sentence = np.sum(log_probs)

    return np.exp(log_prob_sentence)

# Test the bigram_sentence_probability for the German-English dataset
de_sentence = "The economic impact of the legislation was significant." #␣
 ↪Arbitrary parliament sentence
de_prob = bigram_sentence_probability(de_sentence, word_counts_de,␣
 ↪bigram_counts_de, len(word_counts_de))
print(f"Probability of the sentence '{de_sentence}' in German-English data:␣
 ↪{de_prob}")

# Test the bigram_sentence_probability for the French-English dataset
fr_sentence = "Diplomatic efforts were intensified to resolve the conflict." #␣
 ↪Arbitrary parliament sentence
fr_prob = bigram_sentence_probability(fr_sentence, word_counts_fr,␣
 ↪bigram_counts_fr, len(word_counts_fr))
print(f"Probability of the sentence '{fr_sentence}' in French-English data:␣
 ↪{fr_prob}")

# Test the bigram_sentence_probability for the Swedish-English dataset
sv_sentence = "Environmental policies have become increasingly important." #␣
 ↪Arbitrary parliament sentence
sv_prob = bigram_sentence_probability(sv_sentence, word_counts_sv,␣
 ↪bigram_counts_sv, len(word_counts_sv))
print(f"Probability of the sentence '{sv_sentence}' in Swedish-English data:␣
 ↪{sv_prob}")

# Out of Vocabulary Word Test
oov_sentence = "this is a quixotic test sentence"
oov_prob_de = bigram_sentence_probability(oov_sentence, word_counts_de,␣
 ↪bigram_counts_de, len(word_counts_de))
oov_prob_fr = bigram_sentence_probability(oov_sentence, word_counts_fr,␣
 ↪bigram_counts_fr, len(word_counts_fr))
oov_prob_sv = bigram_sentence_probability(oov_sentence, word_counts_sv,␣
 ↪bigram_counts_sv, len(word_counts_sv))
print(f"Probability of the sentence with an OOV word '{oov_sentence}' in␣
 ↪German-English data: {oov_prob_de}")
print(f"Probability of the sentence with an OOV word '{oov_sentence}' in␣
 ↪French-English data: {oov_prob_fr}")
```

```
print(f"Probability of the sentence with an OOV word '{oov_sentence}' in␣
 ↪Swedish-English data: {oov_prob_sv}")
```

```
Probability of the sentence 'The economic impact of the legislation was
significant.' in German-English data: 5.452363580933155e-28
Probability of the sentence 'Diplomatic efforts were intensified to resolve the
conflict.' in French-English data: 5.274458597849841e-34
Probability of the sentence 'Environmental policies have become increasingly
important.' in Swedish-English data: 1.381206394015978e-26
Probability of the sentence with an OOV word 'this is a quixotic test sentence'
in German-English data: 1.0607721443308697e-21
Probability of the sentence with an OOV word 'this is a quixotic test sentence'
in French-English data: 1.0477026424612885e-21
Probability of the sentence with an OOV word 'this is a quixotic test sentence'
in Swedish-English data: 1.0820221311118906e-21
```

Laplace smoothing is used in the cases in which we compute the probability of a sentence that contains a word that is not in the texts, in this case 'quixotic'. Laplace smoothing ensures that every possible word and word pair has a non-zero probability, even if it hasn't been seen in the training data. This means that the word 'quixotic' is assigned with a small, non-zero probability, which helps prevent the model from assigning a zero probability to sentences with words that were not present in the training corpus.

Something, that we noticed in our test before is that the lenght of the sentences affect the probability because of bigram frenquency combinations. So, for very long sentences, it is possible that the probability is very low even if the sentence is in the language. We tested with a sentence of 100 words and the probability was 0.

2.2.3 (C) Translation Model

---

[11]:
```
def create_parallel_sentences(df_en, df_foreign):
    return list(zip(df_en, df_foreign))

# Create parallel corpora
de_parallel_sentences = create_parallel_sentences(de_en_df['English_DE'],␣
 ↪de_en_df['German'])
fr_parallel_sentences = create_parallel_sentences(fr_en_df['English_FR'],␣
 ↪fr_en_df['French'])
sv_parallel_sentences = create_parallel_sentences(sv_en_df['English_SV'],␣
 ↪sv_en_df['Swedish'])

# Select which language pair to use for the IBM Model 1
# We decided to look into English and German parallel sentences
parallel_sentences = de_parallel_sentences

# Initialize translation probabilities uniformly
def initialize_translation_probabilities(parallel_sentences):
```

```
    all_en_words = set()
    all_foreign_words = set()

    # Collect English and foreign words from the parallel sentences
    for en, foreign in parallel_sentences:
        en_words = re.findall(r'\b\w+\b', en.lower())
        foreign_words = re.findall(r'\b\w+\b', foreign.lower())
        all_en_words.update(en_words)
        all_foreign_words.update(foreign_words)

    # Add a buffer for the <NULL> token
    all_en_words.add('<NULL>')

    # Calculate the initial probability
    initial_prob = 1 / len(all_en_words)

    # Create the translation probabilities dictionary
    translation_probs = {}
    for f_word in all_foreign_words:
        translation_probs[f_word] = {}
        for e_word in all_en_words:
            translation_probs[f_word][e_word] = initial_prob

    return translation_probs
```

[12]:
```
def find_top_translations(translation_probs, english_word, top_n=10):
    likely_translations = []
    for f_word, e_words_probs in translation_probs.items():
        if english_word in e_words_probs:
            likely_translations.append((f_word, e_words_probs[english_word]))
    likely_translations.sort(key=lambda x: x[1], reverse=True)
    return likely_translations[:top_n]
```

[13]:
```
def em_algorithm(parallel_sentences, num_iterations=5, target_word="european"):
    print("Initializing translation probabilities...")
    translation_probs = initialize_translation_probabilities(parallel_sentences)

    for i in range(num_iterations):
        print(f"Starting iteration {i+1}/{num_iterations}...")

        count_e_f = defaultdict(lambda: defaultdict(float))
        total_e = defaultdict(float)

        # E-step
        for en_sentence, foreign_sentence in parallel_sentences:
            en_words = ['<NULL>'] + re.findall(r'\b\w+\b', en_sentence.lower())
            foreign_words = re.findall(r'\b\w+\b', foreign_sentence.lower())
```

```
            for f_word in foreign_words:
                denom_c = 0
                for e_word in en_words:
                    denom_c += translation_probs[f_word][e_word]
                for e_word in en_words:
                    delta = translation_probs[f_word][e_word] / denom_c
                    count_e_f[f_word][e_word] += delta
                    total_e[e_word] += delta

        # M-step
        for f_word, e_words_probs in translation_probs.items():
            for e_word in e_words_probs:
                e_word_total = total_e[e_word]
                if e_word_total > 0:
                    translation_probs[f_word][e_word] =␣
↪count_e_f[f_word][e_word] / e_word_total

        # After each iteration, find and print top translations for the target␣
↪word
        top_translations = find_top_translations(translation_probs, target_word)
        print(f"\nTop 10 translations for '{target_word}' after iteration {i+1}:
↪")
        for foreign_word, prob in top_translations:
            print(f"{foreign_word}: {prob}")
        print()

    return translation_probs
```

[14]:
```
# Before running the EM algorithm
print("Starting the EM algorithm...")
translation_probs = em_algorithm(parallel_sentences)
print("EM algorithm completed.")
```

```
Starting the EM algorithm…
Initializing translation probabilities…
Starting iteration 1/5…

Top 10 translations for 'european' after iteration 1:
der: 0.04924442682981817
die: 0.046802621154413596
und: 0.028525924918692182
europäischen: 0.02447797001659754
in: 0.01868630204831187
den: 0.013677785227281396
union: 0.013304013680909268
zu: 0.012170667079711855
europäische: 0.011874041186260318
```

```
das: 0.011722937742977274

Starting iteration 2/5…

Top 10 translations for 'european' after iteration 2:
europäischen: 0.15227090361878062
der: 0.0884782136797811
die: 0.0743837520462704
europäische: 0.0649963326964511
union: 0.04779552637759021
und: 0.04100530475565621
in: 0.028724906323091815
den: 0.020814212714958237
das: 0.0167381101250379
zu: 0.01586282793135845

Starting iteration 3/5…

Top 10 translations for 'european' after iteration 3:
europäischen: 0.34844350160392445
europäische: 0.146996618913835
der: 0.07753193971174076
die: 0.059046929124181906
union: 0.049096988507153994
und: 0.0293777950645934
in: 0.022671697001550178
den: 0.017049370606169983
das: 0.012885488221195607
für: 0.011848600698311503

Starting iteration 4/5…

Top 10 translations for 'european' after iteration 4:
europäischen: 0.48612230571788584
europäische: 0.20916498383045343
der: 0.05784884643372422
die: 0.040285112476897936
union: 0.03281786631771734
und: 0.017673852071668824
in: 0.015225639753741253
den: 0.012015599997872297
das: 0.008627005386665098
für: 0.007468409345575258

Starting iteration 5/5…

Top 10 translations for 'european' after iteration 5:
europäischen: 0.5632761859565995
```

```
europäische: 0.2475383085188334
der: 0.042232841286649406
die: 0.027021545239374084
union: 0.020943386322625646
und: 0.01038648858589231
in: 0.009981265254037574
den: 0.00832870844471093
das: 0.005713162414687346
für: 0.004639426042403674

EM algorithm completed.
```

Over the iterations, the IBM Model 1 increasingly identifies "europäischen" and "europäische" as top translations for "european," indicating improved learning of semantic relevance. Initially, common words like "der" and "die" appear prominently due to their frequency but later iterations better reflect the actual meaning, despite some residual noise from high-frequency and less relevant words. The model's ability to learn the correct translations for "european" demonstrates the effectiveness of the EM algorithm in refining translation probabilities over multiple iterations.

Self-check: if our goal is to translate from some language into English, why does our conditional probability seem to be written backwards? Why don't we estimate P(e|f) instead? The conditional probability in translation models may appear counterintuitive at first glance. In IBM Model 1, we calculate `P(f|e)` rather than `P(e|f)` because we are establishing an alignment model rather than a direct translation model. This model determines the probability that a word in a foreign language aligns with or translates from an English word. In short we are modeling the generative process of the foreign language given the English text. This approach allows the model to be symmetric and supporting translation in either direction which is beneficial for creating a bilingual model that can assist in translating both to and from English.

2.2.4 (D) Decoding

[15]:
```python
def greedy_decoder(foreign_sentence, translation_probs):
    # Tokenize the foreign sentence
    # The regular expression \b\w+\b matches whole words
    foreign_words = re.findall(r'\b\w+\b', foreign_sentence.lower())
    translated_sentence = []

    # For each foreign word find the English word with the highest translation␣
 ↪probability
    for f_word in foreign_words:
        best_prob = 0
        best_english_word = None
        for e_word, prob in translation_probs.get(f_word, {}).items():
            if prob > best_prob:
                best_prob = prob
                best_english_word = e_word

        # If no translation is found use the foreign word
```

```
        if best_english_word:
            translated_sentence.append(best_english_word)
        else:
            translated_sentence.append(f_word)

    return ' '.join(translated_sentence)

foreign_sentence_1 = "das Haus" # The house
foreign_sentence_2 = "Ich spreche ein bisschen Deutsch." # I speak a little␣
 ↪German.
english_translation_1 = greedy_decoder(foreign_sentence_1, translation_probs)
english_translation_2 = greedy_decoder(foreign_sentence_2, translation_probs)
print(f"Translated sentence: {english_translation_1}")
print(f"Translated sentence: {english_translation_2}")
```

```
Translated sentence: reasonably house
Translated sentence: i speaking colourful bisschen deutsch
```

Simplifying Assumptions:

- **Word-by-Word Translation**: This approach assumes that a sentence can be translated word-by-word which rarely is the case given differences in grammar and sentence structure between languages.
- **Ignoring Word Order**: The resulting translation does not consider the correct word order in English.
- **No Context Consideration**: Each word is translated independently of its context which can lead to incorrect translations for words with multiple meanings.
- **Out-of-Vocabulary Words**: Words not seen in the training data will not be translated correctly.

When analyzing the output, it is clear that the method used for translation is quite rudimentary and that the results are imperfect. These translations serve as a starting point and highlight areas where more sophisticated models and techniques are necessary for accurate translation.

Finding the English sentence that has the highest probability given a source-language sentence is algorithmically challenging due to the vast number of possible translations and the complexity of the translation model. - The search space for the most probable English sentence is enormous and the model must consider all possible translations and alignments to find the most probable sentence. This process is computationally intensive and requires sophisticated algorithms to efficiently search the space of possible translations. - The probability P(E F) depends on both the local context (e.g., word order, grammar) and the broader context (e.g., the sentence's topic or style). Accurately modeling these dependencies requires complex probabilistic models. - Evaluating P(E F) for every possible translation E is computationally intensive, especially for long sentences or large vocabularies. - Words can have multiple meanings depending on the context, making it difficult to select the correct translation without understanding the entire sentence. - Phrases that do not translate directly between languages pose a particular challenge, as they require understanding beyond the word level.

2.3 Problem 3: Discussion

(A) Propose a number of different evaluation protocols for machine translation systems and discuss their advantages and disadvantages. What does it mean for a translation to be "good"? Minimally, you should think of one manual and one automatic procedure. (The point here is not that you should search the web but that you should try to come up with your own ideas.) The evaluation of machine translation systems can encompass methods of **Human Evaluation** and **Automatic Evaluation**.

Human Evaluation might include protocols such as:
- **Advantages**: This method provides a nuanced understanding that encompasses assessments of readability, fluency, and the conveyance of original meaning.
- **Disadvantages**: However, it is often time-consuming and costly with a susceptibility to human bias.
- **Protocols**:
- **Ranking/Scoring**: In this protocol, experts assign scores to translations based on criteria including fluency, adequacy and overall quality.
- **Post-editing Analysis**: This involves an examination of the editing effort necessary to refine machine translations to acceptable standards.

Automatic Evaluation offers different advantages and follows distinct protocols:
- **Advantages**: It is recognized for its speed and cost-effectiveness while providing consistent and repeatable metrics.
- **Disadvantages**: The limitation lies in its potential failure to fully grasp semantic correctness and the nuances of fluency.
- **Protocols**:
- **BLEU Score**: This widely-used metric compares the n-gram overlap of the machines output with that of a reference translation.
- **TER**: The Translation Edit Rate measures the number of edits required to change a machine-generated translation into an acceptable version based on a reference.

Determining a "good" translation involves considering accuracy in reflecting the original meaning, maintaining the source's tone, and ensuring fluency within the target language's context. Both human and automatic methods strive to measure these attributes and both have their own degree of efficacy.

(B) The following example shows a number of sentences automatically translated from Estonian into English. In Estonian, ta means either "he" or "she", depending on whom we're talking about. Please comment on the translated sentences: what do you think are the technical reasons we see this effect? Do you consider this to be a bug or a feature? The example illustrates that the translation system has chosen "he" for professions typically perceived as male-dominated ("doctor", "computer programmer") and "she" for roles often stereotypically female ("nurse", babysitter").

This effect likely arises from the training data bias, where the model has learned correlations between gender and profession based on the predominance of these associations in the dataset. While some may view this as a reflection of current societal roles and therefore a feature, it also reinforces gender stereotypes, which can be seen as a bug. As a result, this highlights the need for

careful consideration in how translation systems handle gender by potentially providing gender-neutral options or allowing users to specify gender where relevant.

(c) Below, we consider three sentences that include the English word bat and their automatic translation into Swedish by Google Translate. Why do you think the translation system has been able to select the correct translation of bat in the first two cases? What might be the reason that it has invented a new nonsense word in the third case? When examining the automatic translation of sentences containing the word "bat" from English to Swedish via Google Translate we find two successful translations and one anomaly. The first two sentences which reference "bat" as an animal are correctly translated due to distinct contextual clues phrases like "hit the ball" and "eats insects" guide the translation system to the zoological meaning. However, the third sentence presents a peculiar case where the translation system generates a nonsensical term.

This anomaly arises because the word "bat" can contextually fit both as an animal in the woods and as a piece of sports equipment made of wood. The ambiguity here stems from the translation model's inability to distinguish the term "bat" without a clear contextual indicator favoring the sports equipment interpretation. The model defaults to a nonsensical translation which underscores the critical role of context in accurate translation and the challenges faced by models relying solely on word-level data.

2.4 References

- Brownlee, J. (2019). A Gentle Introduction to Expectation-Maximization (EM Algorithm) - MachineLearningMastery.com. [online] MachineLearningMastery.com. Available at: https://machinelearningmastery.com/expectation-maximization-em-algorithm/ [Accessed 20 Feb. 2024].
- Kapadia, S. (2019). Language Models: N-Gram - Towards Data Science. [online] Medium. Available at: https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9 [Accessed 20 Feb. 2024].
- freeCodeCamp.org (2018). A history of machine translation from the Cold War to deep learning. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/a-history-of-machine-translation-from-the-cold-war-to-deep-learning-f1d335ce8b5/ [Accessed 20 Feb. 2024].
- Srinidhi, S. (2019). Understanding Word N-grams and N-gram Probability in Natural Language Processing. [online] Medium. Available at: https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058 [Accessed 20 Feb. 2024].
- Collins, M. (n.d.). Statistical Machine Translation: IBM Models 1 and 2. [online] Available at: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1162/handouts/Collins_annotated.pdf [Accessed 20 Feb. 2024].

2.5 Self Check

- Have you answered all questions to the best of your ability? Yes, we have.
- Is all the required information on the front page, is the file name correct etc.? Indeed, all the required information on the front page has been included.
- Anything else you can easily check? (details, terminology, arguments, clearly stated answers etc.?) We have checked, and everything looks good.