Design of AI Systems (DAT410)
Assignment 5: Natural Language Processing

Luca Modica
Hugo Manuel Alves Henriques e Silva

February 20, 2024

1 Reading and reflection

1.1 Question (a)

Other than machine translation, the history of AI research has many more pillars: we are referring to AI problems for which many solutions were proposed using a wide range of approaches, each of them based on radically different computational techniques. We will now describe 3 other "holy grails" in the Artificial Intelligence work, which are still seeing an intense research activity as Machine translation.

- One of the first examples in this context is **Computer Vision**. The field has gone through a wide range of different approaches for its most common problems (edge detection and image segmentation, for example), ranging from geometric modeling and early feature detectors to deep learning techniques (CNNs SOTA architectures).
- A second pillar that quickly became very popular is **Game Playing**. Starting from games such as chess and Go, the first experimented approaches were based on rule-based systems with brute force implementations and and hand-crated heuristics; later on, more efficient models like Monte Carlo Tree search were used. In nowadays systems, the applied techniques are based on a completely different approach: Deep Learning and Reinforcement Learning are currently dominating the field, which allows an AI to also learn new strategies from a human input and master a game at the highest level possible.
- A last example is represented by **Robotics**, alongside the most frequent problem in the field: moving, navigating and motion planning. At the start, techniques such as rule-based models, path planning or classical control theory were applied to solve the encountered problems. Following the current trend, nowadays Deep Learning and Reinforcement Learning

have become state-of-the-art models also in this field.

1.2 Question (b)

Although the radically different approaches, rule-based translation systems and state-of-the-art neural systems have some similarities in their inner workings.

- At first, *both systems share the same goal which is to translate text from one language to another*. This is done by trying to follow constraints in the output, from keeping a grammatically correct translation to also maintaining the original meaning of the source sentence.
- *In both rule-based and neural systems the same evaluation metrics can be used* to assess the quality of an output translation, from BLEU (Bilingual Evaluation Understudy) to TER (Translation Edit Rate).
- Lastly, another similarity worth mentioning is that *both systems can be updated and improved over time*, although the 2 different approaches used to achieve to perform the same task. Rule-based systems involve manual inference (manage the rule list), as well as dictionaries based on a supervised performance and and an active language research. Neural systems, on the other hand, are improved by retraining the model on new data, fine-tuning the model in a specific domain knowledge, or varying their internal architecture.

1.3 Question (c)

Nowadays, statistical models or even more state-of-the-art neural methods dominate the scenery of the AI research. However, there can still exist situations where using "old-school" rule-based solutions might be preferable.

- One of the main reasons relies on **model interpretability**. As described in a previous assignment, in most cases there is a trade-off between the model performance and being able to understand its decisions. Rule-based systems represent one of the most intepretable models, since their choices are clear and easy to justify. On the other hand, although extremely accurate results are given by neural based systems, they often result in a black-box when it comes understand their inner workings behind an output. In some real life scenarios, like medicine, it is crucial to understand why a model predicted such an outcome and justify it.
- Neural models rely heavily on training data, whereas rule based systems don't require large datasets to learn from. In other words, in case of **limited or no training data**, rule based solutions may outperform neural approaches. The main reason is that they operate based on explicitly programmed rules and logic.
- Lastly, rule-based models can be a more suitable choice for **Domain-Specific Applications**, especially if the domain is highly structured and

the prediction accuracy is critical. Some contexts, for example, can be composed of precise terminology and grammar and the language use is highly predictable (such as weather report, legal document, medial report and so on): It is a context where the specificity of a rule-based system can represent a significant benefit.

2 Implementation

In this part of the report we will show step-by-step the implementation and the results of the IBM model 1, a word-based Statistical Machine Translation (SMT) developed by IBM in the early 1990s. In particular, our work will reference the pseudo-code described by the original paper about the model itself ([1]), which is the following:

Input: A training corpus $(f^{(k)}, e^{(k)})$ for $k = 1..n$,
where $f^{(k)} = f_1^{(k)}, \ldots, f_{m_k}^{(k)}$ and $e^{(k)} = e_1^{(k)}, \ldots, e_{l_k}^{(k)}$
Initialization: Initialize $t(f|e)$ parameters (e.g. to random values).
Algorithm:

```
for t = 1,...,T:                                  # For each EM iteration
    set all counts c(f,e) and c(e) to 0
    for k = 1,...,n:                              # For each sentence pair
        for i = 1,...,m_k:                        # For each French word
            for j = 0,...,l_k:                    # For each English word
                                                  # including the NULL word
                δ(k,i,j) = t(f_i^(k)|e_j^(k)) / Σ_{j'=0}^{l_k} t(f_i^(k)|e_j'^(k))   # Compute alignment prob.

                c(e_j^(k), f_i^(k)) = c(e_j^(k), f_i^(k)) + δ(k,i,j)   # Update pseudocount

                c(e_j^(k)) = c(e_j^(k)) + δ(k,i,j)                     # Update pseudocount

    set t(f|e) = c(e,f) / c(e)                    # Reestimate probabilities
```

Output: parameters $t(f|e)$

Figure 1: Pseudocode of the of the IBM model 1 SMT.

In this work, the dataset we are going to use is composed by sentence-aligned parallel texts taken from the European Parliament proceedings ([2]); in particular, we will consider Swedish-English, German-English, and French-English sentence pairs, each of them already preprocessed to be easier use.

2.1 Warmup

As a first introductory step, what we need is to collect some statistics about the dataset itself. To take a grasp of the textual data on hand, we first take a look at the 10 most frequent words in each language alongside the 10 most frequent words in English by considering a pair with another language. The lists are reported below.

- **German**: die (10521 occurrences), der (9374 occurrences), und (7028

occurrences), in (4175 occurrences), zu (3169 occurrences), den (2976 occurrences), wir (2863 occurrences), daß (2738 occurrences), ich (2670 occurrences), das (2669 occurrences).

- **English (German-English pair)**: the (19853 occurrences), of (9633 occurrences), to (9069 occurrences), and (7307 occurrences), in (6278 occurrences), is (4478 occurrences), that (4441 occurrences), a (4438 occurrences), we (3372 occurrences), this (3362 occurrences).
- **French**: apos (16729 occurrences), de (14528 occurrences), la (9746 occurrences), et (6620 occurrences), l (6536 occurrences), le (6177 occurrences), à (5588 occurrences), les (5587 occurrences), des (5232 occurrences), que (4797 occurrences).
- **English (French-English pair)**: the (19627 occurrences), of (9534 occurrences), to (8992 occurrences), and (7214 occurrences), in (6197 occurrences), is (4453 occurrences), that (4421 occurrences), a (4388 occurrences), we (3341 occurrences), this (3332 occurrences).
- **Swedish**: att (9181 occurrences), och (7038 occurrences), i (5954 occurrences), det (5687 occurrences), som (5028 occurrences), för (4959 occurrences), av (4013 occurrences), är (3840 occurrences), en (3724 occurrences), vi (3211 occurrences).
- **English (Swedish-English pair)**: the (19327 occurrences), of (9344 occurrences), to (8814 occurrences), and (6949 occurrences), in (6124 occurrences), is (4400 occurrences), that (4357 occurrences), a (4271 occurrences), we (3223 occurrences), this (3222 occurrences).

Let's now assume that we pick a word completely randomly from the European parliament proceedings. What we want to compute now is the probability that a specific word is extracted: we achieve this computation by considering the word count of that specific term over the total counting of the words in all English files. To make 2 examples:

- The probability that the word is *speaker* is 0.0000421.
- The probability that the word is *zebra* is 0.0 (the word is not present in any corpus in the dataset).

2.2 Language modeling

In the second part, we develop the first block of the SMT: the *language modeling*. In particular, in this assignment a **bigram language model** was implemented: given a sequence of words (a sentence) and using the Markov assumption, it allows us to compute the probability of a whole sequence as follows:

$$P(w_1, ..., w_n) = P(w_n|w_{n-1}) \cdot ... \cdot P(w_1|\text{<START>})$$

Where "<START>" represent a special token to define the start of the sentence. Using the Maximum Likelihood Estimation (MLE), the probabilities of a specific

word given the previous one (or, the probability of a bigrams) then becomes:

$$P_{MLE}(w_m|w_{n-1}) = \frac{\text{count}(w_{n-1}, w_n)}{\text{count}(w_{n-1})}$$

Lastly, the probability of a sentence is given by the joint MLE probability of each contained bigrams. The following code represents the implementation of the language model described above, which lets us compute probabilities for individual short English sentences:

```
def calculate_bigram_prob(bigram):
    return bigram_counts[bigram] / unigram_counts[bigram[0]]

def calculate_sentence_prob(sentence):
    sentence_bigram_list = create_bigrams(sentence)
    probability = 1
    for bigram in sentence_bigram_list:
        probability *= calculate_bigram_prob(bigram)
    return probability
```

where `bigram_counts` and `unigram_counts` represent, respectively, the number of bigrams and unigrams in all files containing English corpus.

Unfortunately, a language model of this type is not without problems and challenges to tackle. Down below we will describe 2 potential issues, alongside 2 related solutions to handle them.

- When we encounter a word that did not appear in the training texts, this will result in a probability of zero for any bigram containing this word, making the probability of the entire sentence zero. This is a common issue in language modeling known as the zero-probability problem, and it can be handled using techniques such as Laplace (add-one) smoothing. Following this approach, the MLE probaiblity for a bigram becomes:

$$P_{MLE}(w_m|w_{n-1}) = \frac{\text{count}(w_{n-1}, w_n) + 1}{\text{count}(w_{n-1}) + \#\text{vocabulary}}.$$

- If the sentence is very long (e.g. 100 words or more), its probability will tend to be very small due to the multiplication of probabilities: this can lead to underflow problems in computers. One way to address this problem is by working with log probabilities instead of the raw probabilities.

By addressing the 2 problems above, we updated the Python implementation as follows:

```
# Calculate the vocabulary size
vocabulary_size = len(unigram_counts)
```

```
3
4   # Function to calculate bigram probabilities using Laplace smoothing
5   def calculate_bigram_log_prob_with_laplace(bigram):
6       numerator = bigram_counts[bigram] + 1
7       denominator = unigram_counts[bigram[0]] + vocabulary_size
8       return math.log(numerator) - math.log(denominator)
9
10
11  #calculate probability of a sentence
12  def calculate_sentence_prob_improved(sentence):
13      tokens = ['<START>'] + word_tokenize(sentence.lower())
14      probability = 0
15      for i in range(len(tokens) - 1):
16          bigram = (tokens[i], tokens[i + 1])
17          probability += calculate_bigram_log_prob_with_laplace(bigram)
18      return probability
```

2.3 Translation modelling

The third part is dedicated to the translation model $P(f|e)$. Even though the translation will go in the direction $e \rightarrow f$, and thus the probability we could think to compute is $P(e|f)$, we need to estimate in the inverse direction (*inverse translation*) to take into account the intrinsic conditional probability, as well as allowing the model to capture the linguistic differences and variations between the source and target languages.

To perform this implementation, we implemented the IBM model 1 algorithm introduced above. The code developed is the following:

```
1   def ibm_model_1(corpus_english, corpus_foreign, iterations=10):
2       corpus_foreign_tokens = tokenize_corpus(
3           corpus_foreign, add_null=True)  # foreign language corpus
4       corpus_english_tokens = tokenize_corpus(
5           corpus_english)  # English corpus, with null word
6
7       # Initialize translation probabilities uniformly
8       translation_prob = initialize_translation_prob(
9           corpus_english_tokens, corpus_foreign_tokens)
10
11      for iteration in range(iterations):
12          count_ef = defaultdict(float)
13          total_e = defaultdict(float)
14
15          # E-step: Expectation
16          for sentence_e, sentence_f
```

```
            in zip(corpus_english_tokens, corpus_foreign_tokens):
            for word_f in sentence_f:
                s_total_word_e = sum(translation_prob[(word_e, word_f)]
                    for word_e in sentence_e)
                for word_e in sentence_e:
                    delta = translation_prob[(word_e, word_f)] / s_total_word_e

                    # Update counts
                    count_ef[(word_e, word_f)] += delta
                    total_e[word_e] += delta

        # M-step: Maximization
        for (word_e, word_f), count in count_ef.items():
            translation_prob[(word_e, word_f)] = count / total_e[word_e]


        # normalize probabilities
        new_dict = {}
        for key, value in translation_prob.items():
            if key[0] not in new_dict:
                new_dict[key[0]] = value
            else:
                new_dict[key[0]] += value

        for key, value in translation_prob.items():
            translation_prob[key] = value / new_dict[key[0]]

    return translation_prob
```

To show the results and the potential of our implementation, down below we will list for either Swedish, German, or French, the 10 words that the English word *european* is most likely to be translated into, according to our estimate by using the algorithm. In particular, the words will be listed for incremental number of iterations, with the purpose to show how the list and the probabilities change during the EM iterations (note that for each word, the related probability will be shown). The chosen language for the translation was French.

- **After 5 iterations**:
  - européenne : 0.3533265756494778
  - européen: 0.2017453512410834
  - apos: 0.0810281434570266
  - l: 0.06597959981507077
  - de: 0.06142525373243082

- la: 0.027844146952452548
- le: 0.023991647077701315
- union: 0.01849944281648414
- <NULL>: 0.016373642950344575
- et: 0.012010447940758099

- **After 30 iterations**:
  - européenne: 0.4793358348474479
  - européen: 0.2891000250623562
  - apos: 0.09885042396056481
  - l: 0.0869579716619051
  - de: 0.029416858572761516
  - au: 0.005985469766092662
  - le: 0.0037956171957049777
  - <NULL >: 0.0024198789253493296
  - en: 0.000951969306135209
  - d: 0.0007054494484298159

2.4 Decoding

In the last step of the implementation we will develop the "decoder" algorithm of the translator system: that is, given a sentence in a source language, we want to find a translation such that:

$$E* = argmax_E P(E|F) = argmax_E P(E)P(F|E).$$

In other words, given a source language F, we want to find the English-related sentence E according to the probabilistic model defined above.

However, in practice this is an algorithmically difficult computation, if not infeasible: the main reason is due to the vast number of possible sentence combinations leading to an intractable exhaustive search. Thus, to make the algorithm work, we come up with several assumptions that approximate the theoretical exact solution.

- As a first approximation, for each word in the foreign sentence we will take the related top n words with highest probability.
- For a very similar reason to the previous assumption, we will apply beam search to keep only a limited number of best translation in each step of the algorithm.

- Even though it is already considered, the usage of a language model can be also considered as approximation to guide the translation process.

Each of these assumptions contributed in making the problem more tractable, but has an impact on the quality of the translation output. Considering the top n translation for each word in the foreign sentence and using beam search eventually lead to a locally optimal translation, since we are only considering a subset of possibilities; moreover, involving a language model can also lead to preferences in a grammatically correct translation over a semantically correct one.

The code below represents the implementation of the decoder with the assumptions made:

```
import heapq


def get_top_n_word_translations(foreign_sentence, translation_prob, n):
    word_translations = {}
    for word in foreign_sentence:
        #get top 5 translations for each word
        translations_for_word = {
            pair[0]: prob for pair, prob
            in translation_prob.items() if pair[1] == word}
        # Sort translations by probability
        sorted_translations = sorted(
            translations_for_word.items(),
            key=lambda item: item[1], reverse=True)[:n]
        word_translations[word] = sorted_translations

    return word_translations

def translate_sentence_approx(sentence, translation_prob,
                              n_words=5, beam_width=5):
    """
    Translate one sentence from a foreign language to English.
    In the algorithm, we will keep the top n word translations for
    each word in the sentence. Moreover, we will use beam search to
    keep the most likely translations for the whole sentence.
    """

    beam = [(0, [])]  # (log_prob, sequence)
    top_n_word_translations = get_top_n_word_translations(
        sentence, translation_prob, n_words)

    # Iterate over each word in the foreign sentence
    for word in sentence:
        # Get the top translations for the current foreign word
```

```
        if word in top_n_word_translations:
            top_translations = top_n_word_translations[word]
        else:
            continue

        next_beam = []

        # Expand each sequence in the beam with
        # each translation of the current foreign word
        for log_prob, seq in beam:
            for (translation, translation_prob) in top_translations:
                new_seq = seq + [translation]

                # Update the log probability
                new_log_prob = log_prob + math.log(translation_prob)
                next_beam.append((new_log_prob, new_seq))

        # Keep only the top `beam_width` sequences
        beam = heapq.nlargest(beam_width, next_beam, key=lambda x: x[0])

    prob, highest_prob_sentence = (0, '') if len(beam) == 0
        else max(beam, key=lambda x: x[0])
    return prob, " ".join(highest_prob_sentence)
```

To make the implementation more concrete, let's assume an example by translating the French sentence "je suis européenne" in English. In order to obtain the corresponding translation with higher chance, the following steps will be followed:

1. For each word in the the foreign sentence, a list of n words will be extracted based on the translation probabilities computed by the translation model (IBM model 1). For example, after computing the translation probabilities for 30 iterations, a possible list of the top 5 words for the term "européenne" will be (in order of probabilities): *european, reluctance, construction, periodical, harsh.*

2. Then, for each word in the sentence, the beam algorithm will be applied: we will expand each sequence of the "beam" with each translation of the current foreign word, by updating the log probability as well; at the end of an iteration, by using a max heap, only the top `beam_width` sequences will be kept for the next iteration.

3. Lastly, we will consider the candidate translation with max log probability (that is, the one that is close to zero). The chosen sentence in the example was "i am european", with a log probability of -2.04.

3 Discussion

3.1 Question (a)

In the context of Machine Translation, multiple evaluation protocols can be applied; what they have in common is having a criteria for which a translation can be considered as "good". We should consider as a good translation a sentence that not only possess the right words counter parts from the target language, but also a phrase that tries to keep as much as possible the same meaning of the one in the source language.

We can divide the different evaluation protocols in 2 categories:

- **Manual evaluation**: direct assess of the translation itself. The dominant manual procedure is represented by a *human evaluation*, which offer an overall high quality and reliability. However, this type of evaluations tend to be more time-consuming and subjected to a certain degree of bias.
- **Automatic evaluation**: a more scalable solution than the previous category, where a fixed metric or also a set of rules are set over the results of a SMT. An iconic example is the *BLEU* (Bilingual Evaluation Understudy), which is based on the precision of the n-gram matches between the reference translation and the candidate ones. Although this type of procedures are useful, especially for a large amount of evaluations, they have several limitations that range from focusing too much on specific traits of the text (such as the lexical similarity) to external dependencies (such as having number of reference translations).

3.2 Question (b)

The applied model seems to not give a consistent direct translation of the word *"ta"* in Estonian, and there could be a few reasons why. Considering that, most probably, a statistical model is being applied, the presented examples can make sense.

According to what is described in the question, *"ta"* could mean both "he" or "she" depending on the text and the person it refers to. Firstly, given that no context on the gender of the person in any of the phrases is given, there is no right or wrong translation in terms of gender. One of the reasons why the model is not providing always the same translation is probably due to the context of each phrase. In other words, even if it is equally as likely for the word *"ta"* to be translated into "he" or "she", in the specific given phrases, one of the possibilities may have occurred more often than the other in the set the model was trained on.

For instance, "Ta on meditsiiniõde" is translated as "She's a nurse". This might be due to the phrase "She's a nurse" being more likely to exist in the training set than "He's a nurse". This could be a result of data set imbalance regarding this context. On the other hand, statistically speaking, there exist

more female than male nurses, so it makes sense that the algorithm would "guess" that the writer is referring to a female nurse. Another example is "Ta on arvutiprogrammeerija" being translated as "He is a computer programmer". The phrase "He is a computer programmer" is probably more likely to appear in the dataset than "She is a programmer", just as the previous example, therefore its male translation.

To conclude, although this is not necessarily a bug from a statistical point of view, whether these translations are ethical in a societal context could be debatable.

3.3 Question (c)

Just like in the previous question, a similar problem occurs. However, this time the problem is not in the gender of the words, but a word that, in the original language, can have 2 different meanings depending on the context.

For the first two examples, the translations seem to be correct. The model is able to differentiate the two contexts, where the first clearly refers to a a sport played with some time of equipment, and the second refers to an animal. This may happen because the word "bat" when referred to an animal, is not likely to appear in phrases that refer to the sport and vice-versa. In other words, the probability of the phrase with the animal meaning is low for the sport context given the data the model was trained on.

On the other hand, the third sentence shows an unusual behavior. A mix between the 2 translations of the word bat are being used to form just one word. Some of the reasons why this may happen are:

- **Lack of Clear Context**: Even though, as humans, we can easily see that the phrase refers to the animal, there might not be enough context for the algorithm to differentiate that. One possible reason may be lack of training in this context.
- **Data Noise**: The model may have been trained on noisy data with incorrect or ambiguous uses of the word "bat" leading to an incorrect translation.
- **Algorithmic Error**: Possible undiscovered errors in the algorithm may have been present, forcing it to malfunction in the given input.

References

[1] Michael Collins. Statistical machine translation: Ibm models 1 and 2. 2011.

[2] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers*, pages 79–86, Phuket, Thailand, September 13-15 2005.