Exploration of Machine Translation Techniques using Movie Subtitles dataset

Arnaud Ruymaekers, S5298338

S5298338@studenti.unige.it

Introduction

My initial goal for this project was to explore the 3 main trends for Machine Translation over time, namely, Rule-Based, Statistical and Neural Machine Translation. But, as I went through the project, I realized that Rule-Based Machine Translation was not really feasible to implement from scratch. Then, when I went into Statistical Machine Translation, it was such a big piece that I focused my project on that part.

Statistical Machine Translation consists in finding a sentence in the target language maximizing the product between a language model probability of the sentence and the translation model probability. Summarized by the following equation:

$$e^* = \underset{e}{\operatorname{argmax}}\, p(e) \cdot p(e|f)$$

, with f being the source language sequence and e the target language sequence.

To accomplish this, a Language Model $p(e)$ needs to be implemented which is going to be done with a n-gram based model. Then a Translation Model $p(e|f)$ is going to be implemented using IBM models that consists in defining a translation probability table and a distortion probability table. Then using these two model, a decoding algorithm needs to be implemented to generate translated sequences. For this a Beam Searching algorithm will be used. And finally, the translations will be evaluated using BLEU metrics.

Dataset preparation

The dataset used for this project is sentence pairs of movie subtitles coming from the website “www.opensubtitles.org” . It consists of 36million pairs of sentences in Italian and English.

To prepare the dataset for the Machine Translation task, each sentence is tokenized using NLTK’s “word_tokenize” function particularly for English and Italian.

Then for each word, punctuation is filtered out and the word is lowered. If the sentence still contains words after this, it is saved to compose the corpus.

Even though the whole dataset was cleaned (Project-Dataset_prep.ipynb), only 100thousand sentences will be used during the project as any more would result in hours of processing.

Language Model

A language model is defined by the probability $p(e)$. The language models used for this project are n-gram based. The probability of a sentence is computed with the following formula:

$$p(e) = \prod_{w_i \in e} p(w_i | w_{i-n} \dots w_{i-1})$$

To compute the probabilities $p(w_i | w_{i-n} \dots w_{i-1})$, we can use the Maximum Likelihood Estimate (MLE) that uses the counts of how many time sequences of number appear in the training corpus. The MLE is computed as follows:

$$p(w_i | w_{i-n} \dots w_{i-1}) = \frac{c(w_{i-n} \dots w_i)}{c(w_{i-n} \dots w_{i-1})}$$

To find the counts c, we compute the amount of time each n-length sequence of words appear in the corpus. It is stored in a dictionary with as key the sequence of word and as the value the amount of times this sequence appears in the corpus.

In the particular case we are considering unigram probabilities, the probability of a particular word is defined by the amount of times the word appears to the amount of words in the corpus.

$$p(w_i) = \frac{c(w_i)}{|corpus|}$$

This is the basis of simple N-gram Language Models. Now, there may be issues appearing whenever a certain sequence of words has never appeared in the training corpus, resulting in the probability $p(e)$ being zero.

Laplace Smoothing

Also known as Add-one smoothing, this method consists in adding one to all counts:

$$(w_i | w_{i-n} \dots w_{i-1}) = \frac{c(w_{i-n} \dots w_i) + 1}{c(w_{i-n} \dots w_{i-1}) + S}$$

, where S is the counts of all unique sequences $(w_{i-n} \dots w_{i-1})$ found in the corpus.

The good side of this method is that it is the simplest way to avoid having zero probabilities, but the downside is that the probabilities are significantly changed.

Katz Backoff Model

Instead of altering the counts to compute the probabilities, this method consists in using the largest n-gram possible to compute the probability, else going to a lower n-gram with a discount. For example, if the sequence of words is not available as a trigram, we try to use the bigram sequence of words. Formalized as follows:

$$p(w_i|w_{i-n} \dots w_{i-1}) = \begin{cases} \dfrac{c(w_{i-n} \dots w_i)}{c(w_{i-n} \dots w_{i-1})}, if\ c(w_{i-n} \dots w_i) > 0 \\ \alpha \cdot p\left(w_i \middle| w_{i-(n-1)} \dots w_{i-1}\right), otherwise \end{cases}$$

, where α is a discount factor.

The α parameter can be learned by attempting to maximize the probability score $p(e)$ on a set of sentences (ideally different than the set used generate the counts). In my implementation, I arbitrarily set the discount value to 0.6.

Interpolation

As another option, we can use the n-first n-gram model with a certain weight λ to each n-gram probability. The weight should sum up to 1. For example, when using a trigram for the largest model we would define the probability for word i would be:

$$\hat{p}(w_i|w_{i-2}, w_{i-1}) = \lambda_1 p(w_i|w_{i-2}, w_{i-1}) + \lambda_2 p(w_i|w_{i-1},) + \lambda_3 p(w_i)$$

, with $\lambda_1 + \lambda_2 + \lambda_3 = 1$

As with the backoff model, the λ parameters need to be learned. The learning can be done by gradient descent or Expectation Maximization (EM). However, in my implementation, I performed grid search, meaning, it tried all possible values between 0.1 and 0.9 for each lambda with the constraint they must sum to 1. This led to the best set of lambdas for a unigram (λ_1), bigram (λ_2), trigram (λ_3) model being: $\lambda_1 = 0.1$, $\lambda_2 = 0.1$, and $\lambda_3 = 0.8$.

Translation Model

To define a translation model to find the probabilities $p(e|f)$, we can use the IBM models. The models focus on attempting to find alignment between the sentence pairs. The models build on top of one another adding every time a new component:

- Model 1: lexical translation
- Model 2: additional absolute alignment model
- Model 3: extra fertility model
- Model 4: added relative alignment model
- Model 5: fixed deficiency problem

IBM Model 1

This model attempts to find the translation probabilities between source and target language terms. This information is summarized in a translation probabilities table t.

Since there is no given information about what word of the source sentence is translated to what word of the target sentence, we first assume that every each word has equal probability to be translated by any other word. In other words, we initialize the t table with for the whole source vocabulary a default probability of $\frac{1}{|V_t|}$, where V_t is vocabulary of the target language. Alternatively, we can also initialize it by setting random values from a uniform distribution.

To compute this table t an algorithm already mentioned above is used, Expectation-Maximization. It consists in two steps, the Expectation step and the Maximization step. The algorithm is run for multiple iterations until convergence. In my implementation it I fixed it to 5 or 10 iterations.

Expectation Step

During the Expectation step, we collect over the whole corpus some counts. The counts are over each possible translation pairs of words for each sentence pair in the corpus. The count sums a δ factor computed as follows:

$$\delta_{(k,i,j)} = \frac{t\left(f_i^k \middle| e_j^k\right)}{\sum_{j=0}^{l_k} t\left(f_i^k \middle| e_j^k\right)}$$

, where k is the index of the sentence of the corpus of length n, i is the index of the word in source sentence of length m and j is the index of the word in the target sentence of length l.

It is summed for each word pair translations counts and target word counts:

$$\begin{cases} c(e_j, f_i) = c(e_j, f_i) + \delta_{(k,i,j)} \\ c(e_j) = c(e_j) + \delta_{(k,i,j)} \end{cases}$$

This is done for all sentence k of the corpus. The counts c are initialized to 0.

Maximization Step

Then for maximization step, we update the t table with the following formula:

$$t(f|e) = \frac{c(e,f)}{c(e)}$$

, for each source-target words pairs that appeared in the corpus.

IBM Model 2

This models adds upon the 1st model by adding an absolute alignment model captured by a probability table q. This table stores the probability of a source word i coming from a source sentence of length m and giving a target sentence of length l producing a target word at position j.

To get those probabilities, the same method as for the model 1 is used with some modifications.

To initialize the probability tables, we first run the IBM Model 1 to get an initialization for the t table and the q table is initialized with random values.

Expectation Step

In this step, the δ is modified to look like:

$$\delta_{(k,i,j)} = \frac{q(j|i,l_k,m_k)\ t\left(f_i^k \middle| e_j^k\right)}{\sum_{j=0}^{l_k} q(j|i,l_k,m_k)\ t\left(f_i^k \middle| e_j^k\right)}$$

And for the counts, we are adding two additional counts for each alignment combinations:

$$\begin{cases} c(j|i,l,m) = c(j|i,l,m) + \delta_{(k,i,j)} \\ c(i,l,m) = c(i,l,m) + \delta_{(k,i,j)} \end{cases}$$

Maximization Step

And for maximization steps, the q table is updated using the counts additionally to the t table:

$$q(j|i,l,m) = \frac{c(j|i,l,m)}{c(i,j,m)}$$

Decoding Problem

The decoding problem for machine translation is a very difficult problem.

Note: in my implementation of the decoding, the length of the target sequence is assumed to be the same as the length of the source sequence.

Greedy decoding

We can choose to decode it greedily by taking the max probability for the first word, then taking the highest probability word for the second position knowing the first chosen word.

This technique, although the fastest way of generating a target sequence is most likely to not find a decent translation for a given source sentence.

Exhaustive search

On the other side of the spectrum, doing and exhaustive search is basically considering all possible options. This would explode very quickly in options. So this option is not feasible computational wise.

Beam search

As an in-between option, beam search keeps an number β of open hypotheses. If $\beta = 1$, then we have greedy search, while if $\beta = \infty$, then we have an exhaustive search.

In my implementation, we first query the t table for each word of the source sequence and filter target words to a certain amount and/or to probabilities above a significance threshold (say 10^{-12}).

Then, for each position j in the target sequence we compute the product:

$$p(e_j|e_{j-n} \dots e_{j-1}) \cdot t(e_j|f_i) \cdot q(j|i,l,m)$$

, for each of the possible target word e_j gathered previously.

Now, we select the β hypotheses and for the next position j we expand each saved hypotheses.

Evaluation

For the evaluation, I used the NLTK's implementation of BLEU score. The scoring is done by computing the n-gram overlaps between reference sentences and hypotheses sentences. We can define different weight to each n-gram's importance in the score depending on what we are focussing on.

Results

After having trained with 100k sentence pairs from the corpus the IBM Model 1 and 2 and using the Backoff Language model with trigram probabilities and a discount factor of 0.6 we can attempt to translate some sentences.

I attempted to translate 2000 sentences with Beam Searching with a beam width (β) of 5 and probability threshold of 10^{-7}, we get the following scores after evaluation:

```
[0.2660981092159196,
 0.14394392232122913,
 0.08641068038352694,
 0.053198782628144106]
```

The scores are BLEU scores up to unigram, then up to bigrams, then trigram, then quadrigram with uniform weights for each n-gram.

Future work

To explore deeper the subject, I would have like to run the built models on the full dataset. Then I would have liked to try different combinations of languages models and translation models. Also, I would have liked to do compare scores given by different beam widths.

And, as future work, I would like to explore the further IBM models. Then, a more recent current in Statistical Machine translation is phrase-based translations instead of word-based.

References

[1] Open Subtitles Dataset. Accessed 21/06/2023 https://opus.nlpl.eu/OpenSubtitles.php

[2] Marie, B. 2023. Data Preprocessing for Machine Translation. Accessed 21/06/2023 https://towardsdatascience.com/data-preprocessing-for-machine-translation-fcbedef0e26a

[3] Nguyen, K. 2020. N-gram language models. Accessed 21/06/2023 https://medium.com/mti-technology/n-gram-language-models-b125b9b62e58

[4] Pang, D. 2017. N-gram-language-models. Accessed 21/06/2023 https://github.com/pderichai/n-gram-language-models

[5] Collins, M. 2011. Statistical Machine Translation: IBM Models 1 and 2. Accessed 21/06/2023 http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/ibm12.pdf

[6] Koehn, P. 2018. Machine Translation: IBM Model 1 and the EM Algorithm. Accessed 21/06/2023 https://cal-cs288.github.io/sp20/slides/cs288_sp20_05_statistical_translation_1up.pdf

[7] Koehn, P. 2020. Machine Translation. Accessed 21/06/2023 https://www.youtube.com/playlist?list=PLQrCiUDqDLG0lQX54o9jB4phJ-SLI6ZBQ

[8] Manning, C. 2014. Natural Language Processing, CS224N, Stanford University. Accessed 21/06/2023 https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1162/handouts/

[9] Civera, J., Juan, A. Mixtures of IBM Model 2. Departamento de Sistemas Informàticos y Computaciòn. Universitat Politècnica de València (Spain). Accessed 21/06/2023 https://aclanthology.org/2006.eamt-1.20.pdf

[10] Fraser, A. 2008. EMA 2008 – Statistical Machine Translation. 9th Summer School of the European Masters in Language and Speech Technology. Accessed 21/06/2023 https://www.cis.uni-muenchen.de/~fraser/EMA2008/

[11] NLTK 2023. Nltk.translate package. Accessed 21/06/2023 https://www.nltk.org/api/nltk.translate.html