Homework 1 - Word Alignment

Release Date: Jan 27th, 2023
Due Date: Feb 13th, 2023 @ 11:59PM

Note: For homeworks you can either work independently or in your course project groups.

Aligning words is a key task in machine translation. We start with a parallel corpus of aligned sentences. For example, we might have the following sentence pair from the proceedings of the bilingual Canadian parliament:

```
F: le droit de permis passe donc de $ 25 à $ 500.
E: we see the licence fee going up from $ 25 to $ 500.
```

Getting documents aligned at the sentence level like this is relatively easy: we can use paragraph boundaries and cues like the length and order of each sentence. But to learn a translation model we need alignments at the word level. For Homework 1 you will need to write a program that aligns words automatically.

For example, given the sentence-pair above, your program would ideally output these pairs:

```
le – the, droit – fee, permis – license, passe – going, passe – up,
donc – from, $ – $, 25 – 25, à – to, $ – $, 50 – 50
```

Your program can leave words unaligned (e.g. `we` and `see`) or multiply aligned (e.g. `passe` aligned to `going up`).

Even experienced bilinguals will disagree on word alignments, especially for non-literal translations. For example in the sentence pair below your program must make a choice about how to align the words of the non-literal translations `I want to make it clear` and `il est donc essentiel`.

F: **il est donc essentiel** que cette question fasse le objet de un vote aujourd’ hui .
E: **I want to make it clear** that we have to let this issue come to a vote today.

Within the homework package **nlp267-hw1.zip** you will find the python program `align.py`, which contains a complete but very simple alignment algorithm for word alignment. For every word, it computes the set of sentences that the word appears in. Intuitively, word pairs that appear in similar sets of sentences are likely to be translations.

`align.py` first computes the similarity of these sets with Dice's coefficient.

Given sets X and Y, Dice's coefficient is:

$$\delta(X, Y) = \frac{2x|X \cap Y|}{|X| + |Y|}$$

For any two sets X and Y, $\delta(X, Y)$ will be a number between 0 and 1. `align.py` will align any word pair with a Dice coefficient greater than a predefined threshold (i.e. 0.5).

We can run `align.py` on 1000 sentences:

```
> python align.py -n 1000 > dice.a
```

This command stores the output in `dice.a`.

To compute accuracy, run:

```
> python score-alignments.py < dice.a
```

`score-alignments.py` compares the alignments against human-generated alignments and computes the alignment error rate. It will also show you the word-alignments in a grid.

Try training on 10,000 sentences:

```
> python align.py -n 10000 | python score-alignments.py
```

Performance should improve, but only slightly. Try changing the threshold for alignment. How does this affect alignment error rate?

```
> python align.py -n 10000 -t 0.25 | python score-alignments.py
```

Your task is to improve the alignment error rate. A good approach is to perform word alignment with IBM Model 1. IBM Model 1 forces each of the English words in a sentence to compete as the explanation for each foreign word. This is not the case for an alignment based on the Dice coefficient.

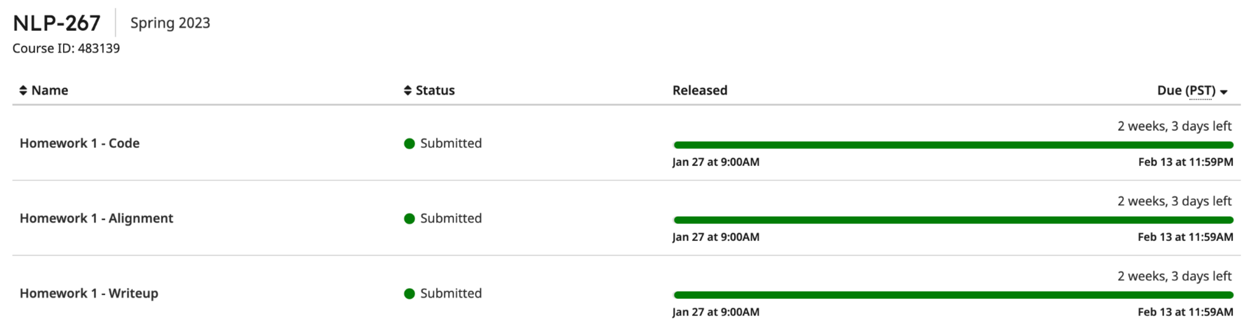
We recommend developing on a small data set (1,000 sentence pairs) with a few iterations of EM. When you see improvements on this small set, try it out on the complete data (100,000 sentence pairs). For more details on implementing EM for IBM Model 1 we suggest reading:
Lopez 2012 - Word Alignment and the Expectation-Maximization Algorithm

To improve the word alignment further you are welcome to explore other methods or even design your own alignment model for a bonus of up to 20% for this homework. However, when exploring other methods you must follow the following rules:

- You must implement the majority of the algorithm yourself. Any functions or code you use from other repositories must be explicitly cited in your source code and writeup
- You cannot use any existing alignment toolkits (such as Giza++, or the Berkeley Aligner)
- You must obtain advance permission to use other software libraries, toolkits or other resources (i.e. dictionaries or parse-trees)

Gradescope Submission:

1. **Homework 1 - Alignment (70%)**: You can upload your alignment output as often as you like up until the homework deadline. The alignment file that you submit to "Homework 1 - Alignment" must be named `alignment`
2. **Homework 1 - Code (10%)**: Upload the final version of your code that was used to create the `alignment` file. You are free to extend the provided code or write your own. The code should be self-contained, documented, and easy to use. Please also include a README on how to run it.
3. **Homework 1 - Write up (20%)**: Provide a short written description (approx 1-page) of what you implemented describing the overall algorithm and the important functions and data structures in your implementation



NLP-267 | Spring 2023
Course ID: 483139

Name	Status	Released	Due (PST)
Homework 1 - Code	Submitted	Jan 27 at 9:00AM	2 weeks, 3 days left — Feb 13 at 11:59PM
Homework 1 - Alignment	Submitted	Jan 27 at 9:00AM	2 weeks, 3 days left — Feb 13 at 11:59AM
Homework 1 - Writeup	Submitted	Jan 27 at 9:00AM	2 weeks, 3 days left — Feb 13 at 11:59AM

Credits: This homework assignment is adapted from prior works from Philipp Koehn, John DeNero, Chris Dyer and Adam Lopez.