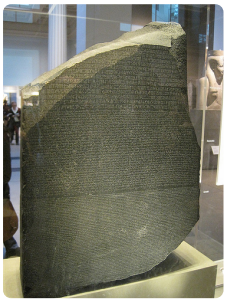

Rosetta stone (credit; calotype46)

Homework 2: Word Alignment

Due: Sep 20 2021

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

le droit de permis passe donc de $ 25 à $ 500.

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 c like the length and order of each sentence. But to learn a translation model we need alignments at the *word* level. That's where you come in. **Your challenge is to write a program that aligns words automatically.** For exam given the sentence 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*). It wil faced with difficult choices. Suppose it sees this sentence pair:

I want to make it clear that we have to let this issue come to a vote today.

il est donc essentiel que cette question fasse le objet de un vote aujourd' hui .

Your program must make a choice about how to align the words of the non-literal translations *I want to make it cl* and *il est donc essentiel*. Even experienced bilinguals will disagree on examples like this. So word alignment does n capture every nuance, but it is still very useful.

Getting Started

You must have git and python on your system to run the assignments. Once you've confirmed this, run this comma

```
git clone https://github.com/xutaima/jhu-mt-hw
```

In the `hw2` directory you will find a python program called `align`, which contains a complete but very simple alignment algorithm. For every word, it computes the set of sentences that the word appears in. Intuititvely, word p that appear in similar sets of sentences are likely to be translations. Our aligner first computes the similarity of the sets with Dice's coefficient (http://en.wikipedia.org/wiki/Dice's_coefficient/). Given sets X and Y, Dice's coefficient

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

For any two sets X and Y, $\delta(X, Y)$ will be a number between 0 and 1. The baseline aligner will align any word pair a coefficient over 0.5. Run it on 1000 sentences:

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

This command stores the output in `dice.a`. To compute accuracy, run:

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

This compares the alignments against human-produced alignments, computing alignment error rate (https://www.aclweb.org/anthology/P00-1056.pdf), which balances precision and recall. It will also show you the comparison in a grid. Look at the terrible output of this heuristic method – it's better than chance, but not any gooc Try training on 10,000 sentences:

```
python align -n 10000 | python score-alignments
```

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

The Challenge

Your task is to *improve the alignment error rate as much as possible*. It shouldn't be hard: you've probably noticed thresholding a Dice coefficient is a bad idea because alignments don't compete against one another. A good way to correct this is with a probabilistic model like IBM Model 1. It forces all of the English words in a sentence to compe the explanation for each foreign word.

Formally, IBM Model 1 is a probabilistic model that generates each word of the foreign sentence $\mathbf{f}$ independently, conditioned on some word in the English sentence $\mathbf{e}$. Given $\mathbf{f}$, the joint probability of an alignment $\mathbf{a}$ and translatic factors across words: $P(\mathbf{f}, \mathbf{a}|\mathbf{e}) = \prod_i P(a_i = j||\mathbf{e}|) \times P(f_i|e_j)$. In Model 1, we fix $P(a_i = j||\mathbf{e}|)$ to be uniform (i.e. e to $\frac{1}{|\mathbf{e}|}$), so this probability depends only on the word translation parameters $P(f|e)$. But where do thes parameters come from? You will first learn them from the data using expectation maximization (EM), and then use them to ali EM attempts to maximize the *observed* data likelihood $P(\mathbf{e}|\mathbf{f})$, which does not contain alignments. To do this, we marginalize over the alignment variable:

$$P(\mathbf{e}|\mathbf{f}) = \prod_i \sum_j P(a_i = j||\mathbf{e}|)$$

This problem can't be solved in closed form, but we can iteratively hill-climb on the likelihood by first fixing some parameters, computing expectations under those parameters, and maximizing the likelihood as treating expected counts as observed. To compute the iterative update, for every pair of an English word type e and a French word ty count up the expected number of times f aligns to e and normalize over values of e. That will give you a new estim of the translation probabilities $P(f|e)$, which leads to new expectations, and so on. For more detail, read this note (http://mt-class.org/jhu/assets/papers/alo-model1-tutorial.pdf). We recommend developing on a small data set (1 sentences) with a few iterations of EM. When you see improvements on this small set, try it out on the complete da

Developing a Model 1 aligner should be enough to beat our baseline system and earn a passing grade. But alignmer isn't a solved problem, and the goal of this assignment isn't for you to just implement a well-known algorithm. To g full credit you **must** experiment with at least one additional model of your choice and document your work. Here some ideas:

- Implement a model that prefers to align words close to the diagonal (http://aclweb.org/anthology/N/N13/N13-1073.pdf).
- Implement an HMM alignment model (http://www.aclweb.org/anthology/C96-2141.pdf).
- Implement a morphologically-aware alignment model (http://aclweb.org/anthology/N/N13/N13-1140.pdf).
- Use *maximum a posteriori* inference under a Bayesian prior (http://aclweb.org/anthology/P/P11/P11-2032.pdf)
- Train a French-English model and an English-French model and combine their predictions (http://aclweb.org/anthology/N/N06/N06-1014.pdf).
- Train a supervised discriminative alignment model (http://aclweb.org/anthology/P/P06/P06-1009.pdf) on the annotated development set.
- Train an unsupervised discriminative alignment model (http://aclweb.org/anthology/P/P11/P11-1042.pdf).
- Seek out additional inspiration (http://scholar.google.com/scholar?q=word+alignment).

But the sky's the limit! You are welcome to design your own model, as long as you follow the ground rules:

Ground Rules

- You can work in independently or in groups of up to three, under these conditions:
  1. You must announce the group publicly on piazza.
  2. You agree that everyone in the group will receive the same grade on the assignment.
  3. You can add people or merge groups at any time before the assignment is due. **You cannot drop peopl from your group once you've added them.** We encourage collaboration, but we will not adjudicate Rashomon-style stories about who did or did not contribute.
  4. You must submit the assignment once per group on Gradescope, and indicate your collaborators once yo upload the files.
- You must turn in three things to Gradescope (https://www.gradescope.com/):
  1. An alignment of the entire dataset.
     You can upload new output as often as you like, up until the assignment deadline. The output will be evaluated on a subset of the data, but the `score-alignments` program will give you a good idea of how you're doing, and you can use the `check` program to see whether your output is formatted correctly. Th leaderboard on Gradescope will display the metrics on a Dev set until the deadline. It will be then update also display the metric on the Test set.
     Your alignment file must be named `alignment`.

2. Your code.
   You are free to extend the code we provide or roll your own in whatever langugage you like, but the code should be self-contained, self-documenting, and easy to use. Please also include a README on how to ru
3. A clear, mathematical description of your algorithm and its motivation written in scientific style.
   This needn't be long, but it should be clear enough that one of your fellow students could re-implement i exactly. We will review examples in class before the due date (http://mt-class.org/jhu-2014/hw-writing-exercise.html).

- You may only use data or code resources other than the ones we provide *with advance permission*. We will as you to make your resources available to everyone. If you have a cool idea using the Berkeley parser, or a Fren English dictionary, that's great. But we want everyone to have access to the same resources, so we'll ask you to share the parses. This kind of constrained data condition is common in real-world evaluations of AI systems, make evaluations fair. A few things are off-limits: Giza++, the Berkeley Aligner, or anything else that already the alignment for you. You must write your own code. If you want to do system combination, join forces with your classmates.

If you have any questions or you're confused about anything, just ask on Piazza or in office hours.

Credits: This assignment is adapted from one originally developed by Philipp Koehn (http://homepages.inf.ed.ac.uk/pkoehn/) and later modified by John DeNero (http://www.denero.org/). It incorpora some ideas from Chris Dyer (http://www.cs.cmu.edu/~cdyer).
