School of Computing and Information Systems
The University of Melbourne
COMP90042 NATURAL LANGUAGE PROCESSING (Semester 1, 2022)

Sample solutions: Week 10

Discussion

1. What aspects of human language make automatic translation difficult?

   The whole gamut of linguistics, from lexical complexity, morphology, syntax, semantics etc. In particular if the two languages have very different word forms (e.g., consider translating from an morphologically light language like English into Turkish, which has very complex morphology), or very different syntax, leading to different word order. These raise difficult learning problems for a translation system, which needs to capture these differences in order to learn translations from bi-texts, and produce these for test examples.

2. What is **Information Extraction**? What might the "extracted" information look like?

   - Basically, we want to extract information from a (generally unstructured) document, into a structured format that we can sensibly query later.

   (a) What is **Named Entity Recognition** and why is it difficult? What might make it more difficult for persons rather than places, and *vice versa*?

   - We want to find **named entities** — mostly proper nouns, but also sometimes times or numerical values of significance — within a document. Often context (which is not always present within the sentence, or even the document!) is needed to disambiguate the type of named entity, or even whether a given phrase is a named entity at all.
   - One common problem, that we see with both people's names and places, is that they are ambiguous with common nouns. Generally speaking, we can write a (somewhat) exhaustive list of names of places — a **gazetteer** — but we can't with names of people, which are constantly changing. On the other hand, locations can also be quite ambiguous, e.g. "New York" in "New York Times" is less of a location than an organisation.

   (b) What is the **IOB** trick, in a sequence labelling context? Why is it important?

   - Named entities often comprise multiple tokens, like "the University of Melbourne" — these are often represented through bracketing, e.g. `I visited [the University of Melbourne]`$_{\text{LOC}}$ `yesterday.`
   - Getting the bracketed entity from a tree structure — like one produced from **parsing** — is fairly straightword, but it is often more convenient to **tag** individual tokens. To do this, we indicate whether a given token is **B**eginning a named entity, **I**nside a named entity, or **O**utside a named entity, so that the sentence above might look like `I-O visited-O the-B-LOC University-I-LOC of-I-LOC Melbourne-I-LOC yesterday-O .-O`

(c) What is **Relation Extraction**? How is it similar to NER, and how is it different?

- Relation Extraction attempts to find and list the relationships between important events or entities within a document.
- Relations typically hold between entities (e.g., MP-for(Turnbull, Wentworth)), so in order to extract relations you first need to do NER to extract the entities from the text (e.g., **Turnbull**, the member of **Wentworth**, said ...; where the bolded items would be tagged using an NER system.)

(d) Why are hand–written patterns generally inadequate for IE, and what other approaches can we take?

- Basically, because there are too many different ways of expressing the same information. We can often write rules that lead to accurate relations (high Precision), but they won't cover all of the relations within an unseen document (low Recall).
- Parsing the sentence might lead a more systematic method of deriving all of the relations, but language variations mean that it's still quite difficult. More sophisticated approaches frame the problem as supervised machine learning, with general features (like POS tags, NE tags, etc.) and features specific to the relations that we're trying to identify.
- **Bootstrapping** patterns — using known relations to derive sentence structures that describe the relationship — is also somewhat popular in some domains.

Programming

1. In the iPython notebook `11-machine-translation`, we provide an example of an encoder-decoder model for machine translation. Just like workshop-07, we'll use colab to run the notebook, as we'll need a GPU to speed up the training. Please refer to workshop-07 if you are looking for instructions to get set up for colab.

   - Modify the code to use GRU instead of LSTM, noting that GRU does not have a memory state (`state_c`).
   - Modify the code to do translation at the word-level instead of character-level. This will involve:
     - Using a French and English tokeniser (e.g. spaCy);
     - Creating a vocabulary for the input source and target language;
     - Replacing low frequency words with a special UNK token;
     - Changing the corpus reading function to create sequences of words for the training data;
     - Updating the training model and inference model to use the word vocabulary to look up words.

Get ahead

- Extend the encoder-decoder model in `11-machine-translation` to incorporate attention mechanism. You can use the formulation described in the lecture: dot product for comparing encoder and decoder hidden states; and concatenating the "context vector" with the decoder hidden state to predict the target word.