Chapter 7

Generation

Generation has long been plagued by the question, "Generation from *what*?" (Wilks). Because computer representations of semantics are a topic of research, and the systems that compute these representations are also a topic of research, it's more difficult to work on generation systems.

There are, however, a large number of NLP tasks involving output of text that sidestep the problem of meaning representation by taking other forms of input besides semantics. For example:

constrain	task	input	output
less ↑	Random generation	nothing	random text
	Summarization	document	summary of document
	Question answering	question	answer to the question
	Translation	sentence	same meaning in other language
more ↓	ASR	spoken sentence	same words in text form

At one end of the spectrum are unconstrained generation systems where the input is nothing, or maybe a few starting words, and the system is supposed to generate random text. These systems are good for entertainment but don't have clear practical uses. At the other end are constrained systems where the input is, say, speech, and the system is supposed to output the same thing in textual form; most people don't think of this as generation at all.

In between are the more interesting cases. We've talked about translation already; below, we talk about question answering and summarization.

7.1 Question Answering

Question answering is (surprisingly enough) the task of answering questions. Although one could imagine treating QA as a translation problem (translating questions to answers), you would need quite a lot of training data in the form of question-answer pairs, and a large model, to amass the knowledge needed to answer questions well.

We assume that we have modest-sized collection of questions and answers to train on (for example, SQuAD has 100k questions) and a large collection of documents that potentially contain information needed for answering new questions (for example, all of Wikipedia).

We'll focus on approaches that divide the process into two stages:

1. In the retrieval step, traditional information retrieval methods are used to find a document or documents that are likely to contain the answer to the question. For example, you could turn the question into a bag-of-words vector and turn all the documents into bag-of-words vectors (normalized for length), and choose the document with the highest dot-product with the question.

2. In the reading step, we use the retrieved document and question together to generate an answer.

Much work on QA assumes that the retrieval step is already done and only tackles the reading step. For example, the SQuAD dataset (Rajpurkar et al., 2016) includes with each question a paragraph from Wikipedia that contains the answer in it. (In version 2 of the dataset, sometimes the paragraph intentionally does not contain the answer, in which case the system is supposed to refuse to answer.) This restricted form of QA is sometimes called machine reading comprehension (MRC).

For example, the first item in SQuAD 1.1 gives the paragraph:

```
architecturally , the school has a catholic character . atop the main
building 's gold dome is a golden statue of the virgin mary
. immediately in front of the main building and facing it , is a
copper statue of christ with arms upraised with the legend " venite ad
me omnes " . next to the main building is the basilica of the sacred
heart . immediately behind the basilica is the grotto , a marian place
of prayer and reflection . it is a replica of the grotto at lourdes ,
france where the virgin mary reputedly appeared to saint bernadette
soubirous in 1858 . at the end of the main drive ( and in a direct
line that connects through 3 statues and the gold dome ) , is a simple
, modern stone statue of mary .
```

The question is:

```
to whom did the virgin mary allegedly appear in 1858 in lourdes france ?
```

And the answer is words 102–104 (`saint bernadette soubirous`).

We could build a simple MRC system, again, by treating it as a translation problem. The above example would be presented as a source string consisting of both the question and the paragraph:

```
BOS to whom did the virgin mary allegedly appear in 1858 in lourdes
france ? SEP architecturally , the school has a catholic character
. atop the main building 's gold dome is a golden statue of the virgin
mary . immediately in front of the main building and facing it , is a
copper statue of christ with arms upraised with the legend " venite ad
me omnes " . next to the main building is the basilica of the sacred
heart . immediately behind the basilica is the grotto , a marian place
of prayer and reflection . it is a replica of the grotto at lourdes ,
france where the virgin mary reputedly appeared to saint bernadette
soubirous in 1858 . at the end of the main drive ( and in a direct
line that connects through 3 statues and the gold dome ) , is a simple
, modern stone statue of mary . EOS
```

But this system would be free to generate arbitrary text, which seems wasteful since we know that the answer is literally found in the paragraph. So a more direct approach would be to directly predict the starting and ending word of the answer. If $\mathbf{H} \in \mathbb{R}^{n \times d}$ is the encoding of the source sentence, we can predict the starting and ending position using two softmax layers:

$$\begin{aligned} \mathbf{w}_{\text{start}}, \mathbf{w}_{\text{end}} &\in \mathbb{R}^d \\ \mathbf{p}_{\text{start}}, \mathbf{p}_{\text{end}} &\in \mathbb{R}^n \\ \mathbf{p}_{\text{start}} &= \text{softmax}\, \mathbf{H}\mathbf{w}_{\text{start}} \quad (7.1) \\ \mathbf{p}_{\text{end}} &= \text{softmax}\, \mathbf{H}\mathbf{w}_{\text{end}} \quad (7.2) \end{aligned}$$

The loss function to be minimized for a sentence where i and j are the true starting and ending position would be:

$$L = -(\log[\mathbf{p}_{\text{start}}]_i + \log[\mathbf{p}_{\text{end}}]_j). \quad (7.3)$$

7.2 Summarization

Summarization is the task of converting a long document (or documents) into a shorter one. This is something humans do a lot – for example, papers have abstracts, movies and novels have synopses, some news articles have bullet points, social media posts have tl;drs.

Approaches to summarization are usually divided into two categories; the division is so sharp as to almost be a divison into two subtasks. *Extractive* summarization builds a summary out of pieces of the source document, while *abstractive* summarization composes a summary from scratch.

7.2.1 Extractive summarization

Extractive summarization is really old, dating back to the work of Luhn (1958). The steps in a typical extractive summarization method are as follows.

1. Divide the source text into segments (usually sentences).
2. Compute a vector encoding of each sentence.
3. Third, select which sentences will appear in the generated summary.

It appears that most extractive approaches don't attempt to reorder the selected sentences.

After the first step, then, we've essentially reduced the summarization problem to a binary classification problem. However, we don't want to classify each sentence independently, because we don't want to select redundant sentences.

A method commonly described in online tutorials, similar to Luhn's original method, is:

2. Represent each sentence as a bag-of-words. (To improve the quality of the sentence vectors, one can remove stop words or use tf-idf or BM25.) Normalize the sentence vectors by sentence length.

3. Sum up each sentence vector and select those sentences whose score is above a certain threshold.

Now, suppose that we have training data. For example, the CNN/Daily Mail dataset (Nallapati et al., 2016) contains summaries of news articles based on bullet points at the top of the articles. We can shoehorn this into a dataset for extractive summarization by selecting the sentences of the article that are most similar to the bullet points.

The above bag-of-words approach could be minimally modified into a naive Bayes classifier, but let's look at something more modern. The model of Y. Liu and Lapata (2019) is simple and effective (though it works best when pretrained with BERT).

2. Run a Transformer encoder on each sentence. To get a single vector for each sentence, prepend a `CLS` token to each sentence and use its encoding.

3. Apply another Transformer to the sentence vectors to obtain a new sequence of sentence vectors. Then apply a sigmoid layer (= softmax layer over {yes, no}) to classify each sentence.

7.2.2 Abstractive summarization

For a long time, abstractive summarization was a much less common approach, but after the advent of attention in neural machine translation, abstractive summarization has become a lot more common (Rush, Chopra, and Weston, 2015).

The basic idea is extremely simple: treat summarization as translation problem where the source language is "long English" and the target language is "short English."

It's possible to make enhancements to the model to make it more suitable for summarization (See, P. J. Liu, and Manning, 2017).

- A copy mechanism similar to the one we saw in semantic parsing.

- A *coverage* model, originally developed for machine translation (Tu et al., 2016), that helps the model learn not to repeat itself. Essentially, at time step i of generating the target sentence, we sum up the attentions from time steps $1, \ldots, i$ and the sum is taken into account when computing the attention for time step i. (I would have given equations, but it's not totally clear how coverage would be incorporated into the version of the model we gave in §3.6.3.)