Machine Translation 468: Final Project Interim Report

Lionel Eisenberg, Sanat Deshpande

Johns Hopkins University — November 14, 2018

1 Introducton: the problem

Neural machine translation has advanced aggressively over the past few years, boasting impressive results in several language pairs, notably English and French. However, despite its success with formal or standardized text, these models often flounder when presented with slang, abbreviations, and general errors in writing. For this reason, we've elected to tackle the problem of normalizing text prior to translation. In other words, in a mini-translation task of its own, how can we take non-standard language, reformulate it as standard, and then pass it into a pre-existing translation model?

For instance, a modern neural machine translation model would encounter no trouble translating the following from English to French:

- "Where are you?" -> "Où es tu?"

However, given a quite common, non-standard equivalent, it would not meet the same success. The below example outlines a possible instance of the process by which normalization prior to translation would work

- "Wya?" -> "Where you at?" -> "Where are you?"
- The above normalization would then be acceptable input for regular neural translation

To clarify, our project aims to tackle the normalization problem of non-standard language translation, not the translation itself. This means we seek only to render slang or mistake-ridden usage into a formalized version of the source language. In practice, our model would be used as a preprocessing step to standardize input prior to using any existing translation model.

The motivation behind solving this problem is the abundant and rapidly growing instances of its uses cases. As the population of internet users burgeons, ever-changing non-standard language will present itself with rising frequency, thus increasing the need for neural translation models that can cope with this new language. While a text normalizer would not be solving some underlying mathematical challenge that comes with language translation, it would, however, enable us to build much more versatile translation systems that can handle a broader set of cases than currently possible.

2 Baseline

We ran a vanilla statistical machine translator called OpenMT on two different datasets, these were a "Proper" german to english dataset and a German tweet to english dataset (source: https://github.com/WladimirSidorenko/PotTS).

Here are the. score we got:

1. Proper German > English
    - PRED AVG SCORE: -0.8754, PRED PPL: 2.3999
2. Tweet German > English
    - PRED AVG SCORE: -0.8050, PRED PPL: 2.2366

3 Our Project

The implementation we are going to attempt to code in order to somewhat solve the problem that we have described above is text normalization through beam-search decoding. Indeed, in order to be able to properly translate our input social media sentences, we need to transform said input into proper english that a statistical model is used to translating. We will be using a decoder very similar to the one proposed in the paper *A Beam-Search Decoder for Normalization of Social Media Text with Application to Machine*, **Pidong Wang and Hwee Tou Ng**, in order to transform our social media text into "proper" english text.

Similarly to beam search decoders implemented in papers such as *Pharaoh: A Beam Search Decoder for Phrase-Based Statistical Machine Translation*, **Koehn**, we want to create a hypothesis tree for the input sentence we are trying to decode, and keep the best hypotheses as the decoder goes through more and more of these hypotheses. We will be describing what the hypotheses creators are in a moment but first let us examine the process the decoder will take to evaluate hypotheses.

This whole process should sound very similar to those familiar with the decoder described by *Koehn* in the above paper,

1. We first start by arranging all the hypothesis in stacks, where the $i-th$ stack corresponds to the hypothesis having had i hypotheses producers run on it.
2. We then cycle through each hypothesis in the stack i
   (a) Cycle through each possible hypothesis producer
   (b) Produce each possible new hypothesis from these producers
   (c) Add them to the next hypothesis stack
   (d) Prune it through methods we have discussed in class before.
      - These methods include histogram pruning, where we limit the number of hypothesis of a stack to a prefixed number k
3. Return the best scored hypothesis

In order to score the hypotheses, we will be using summations of weighted scoring techniques:

$$score(h) = \sum_{i}^{n} \alpha_i g_i(h)$$

The scoring techniques are the following:

- language model score
- Informal word count penalty
- Amount of changes made by the hypothesis makers

Now that we have outlined the decoder logic for our implementation, let us examine the different hypothesis creators that we need to implement to actually normalize the text:

For all of these techniques, we will train the model on synthetic data where we create input and target data by synthetically manipulating text. For example for punctuation correction, we will synthetically create erroneous input for target sentences by substituting, deleting and adding incorrect punctuation into the sentence. Although we do realize that this method creates unrealistic data, it is the best we can do given the lack of currently available data.

- **Punctuation corrector**: social media text has usually very poor punctuation, we want to create a way to substitute, add and delete punctuation from input sentences. To do this we will use a two layer DCRF model that will in its first layer creates the actual punctuation tags while the second layer gives us sentence boundaries. We then use n-grams to run the model.

- **Dictionary**: we will have a dictionary of acronyms that we can parse our input hypothesis for and substitute acronyms or slang for their real meaning. For example, “wya” would be normalized to “where you at” by this hypothesis creator. This dictionary will either be manually generated, or we will be using the UrbanDictionary.org API endpoint to define informal words or slang.
- **Interjection**: remove all words that are possible interjections at the end of a sentence (from a list of predertimed interjections)
- **Word-Recovery**: we recover missing words from the “be” verb: BE, AM, ARE, IS, using synthetic data and a CRF/DL model.
- **Prefix**: recover formal word from words with prefixes that would otherwise be a formal word.
- **Time**: Potential time expressions will be normalized.
- **Abbreviation**: if we can retrieve a formal word from an informal word by adding a vowel, then we will attempt to recover said formal word.

The best way to evaluate this model will definitely be to use BLEU translation metric to see how well our model performs once the text is translated.