Machine Translation 468: Final Project Report

Lionel Eisenberg, Sanat Deshpande

Johns Hopkins University — December 6, 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. Machine Translation has both progressed for formal texts such as those used in big governmental organizations through statistical machine translation and informal texts through the massing of large amounts of data and neural networks. 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? How can we effectively translate informal english to formal english ?

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 Datasets

We have many different datasets that we had to both retrieve online and create ourselves in order to complete this project:

- **Formal Word Dictionary:** We obtained a formal dictionary of english words from the following link: `https://github.com/dwyl/english-words/blob/master/words_alpha.txt`. The text file, containing over 466k English words was the basis for creating a list of formal words that we could base ourselves off of.
- **Informal Word Dictionary:** We created an informal list of words by taking a massive list of tweets from the following link: `https://ia800501.us.archive.org/2F32%2Fitems%2Ftwitter_cikm_2010%2Ftwitter_cikm_2010.zip&file=training_set_tweets.txt`. We are using a public dataset of tweets from the USA. The original dataset was comprised of tweets along with location metadata for a different type of study, but we discarded the additional location data, as we didn't require it. Our motivation for using tweets as the subject of text normalization was the fact that written language on the internet, especially on social media, is very often irregular and non-standard. The abundance of improper punctuation, spelling errors, slang usage, and abbreviations touch almost all of the topics discussed by the reference paper we followed for this project. We then pruned the tweets and removed any HashTags or any other nonsensical readily evident text. Finally we did a set difference between the informal words and the dictionary formal words to get an ordered list by number of appearances in the corpus.

  Due to a lack of resources, both authors of the papers hand translated the top 2000 informal words in our corpus to create a lookup table in order to translate future informal words into formal english.
- **CORPUS EXPLANATION**
- **Common Contractions**: We also obtained a list of contractions in order to complete our formal english dictionary and make it possible for us to fix contraction errors in inputed sentences. The list was obtained from `https://gist.github.com/J3RN/ed7b420a6ea1d5bd6d06#file-contractions-txt`
- **Common Interjections**: We collected and compounded a list of commonly used english interjections such as "ah", "wow", "yay" etc. to be identify them when normalizing sentences.

Unfortunately, although we did contact the authors of the paper we are strongly basing this survey on, we were not able to get an answer from them, nor were we able to retrieve the datasets that they used in the study to perform their normalization.

4 Our Project

The implementation we have written 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

4.1 Beam-Search Algorithm

Here is a proper layout of the algorithm used by the beam-search decoder:

Algorithm 1: Beam-Search Decoder Algorithm

```
Create n stacks where n is the number of hypothesis producers;
for stack in stacks do
    for h in stack do
        for each hypothesis producer do
            if producer not yet used then
                for each possible hypothesis production do
                    hypothesis_new ← produce(h);
                    score(hypothesis_new);
                end
            end
        end
        Prune stack and keep best scores
    end
end
return max scored hypothesis of last stack
```

4.2 scoring

In order to score the hypotheses, we used summations of weighted scoring techniques:

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

Where the scoring techniques are the following:

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

4.3 Hypothesis Producers

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.

Here are the hypothesis producers that we completed:

- **Dictionary**: from the manually created dictionary of words and their manually entered translations, we parsed through the sample sentenced and tried to replace informal words with their corresponding formal translation
- **Interjection**: From the corpus of interjections that we collected, we looked at the first and last word of each sentence and removed the word if it corresponded to any of those. We do realize that this could however create erroneous data, therefore kept the list of interjections very factual.
- **Quotation**: Given a word w we check first if it belongs to the set of formal words we obtained. We then check if the last character of w is (m, s, t, d). If so, we add an apostrophe before the last character and check to see if the word is part of the list of contractions we have amassed. If that is the case, then we swap the word for the proper contracted word. So for example, "*isnt*" would ideally be normalized to "*isn't*"
- **Time**: for time normalization we proceeded to change any three or four digit that was situated after an "at" or before an "am" or a "pm". We added a colon to those numbers, so for example, "*1030 pm*" becomes "*10 : 30 pm*". Moreover we edited single digit numbers so that a sentence such as "*we will meet at 4*" to "*we will meet at 4 o'clock*.
- **re-tokenization**: in the event that a period is poorly spaced in a sentence, i.e. "*where r u.we are leaving*" we rectified by adding spaces around the period to make it easier for translation models to translate the separated words. Moreover, we checked to see if the period belonged to a URL or an email before normalizing it in order to avoid false positives.
- **Punctuation:** A common problem in text normalization comes in the form of punctuation normalization. Informal written language is notorious for its lack of standard punctuation use, and this can pose a problem to traditional neural translation models, as punctuation can often affect the meaning of a sentence. In order to normalize such text, however, we need a parallel corpus of correctly punctuated sentences alongside non-standard sentences. As it is difficult to obtain such corpora, some papers suggested randomly introducing punctuation errors into a know set of standard sentences to mimic real-world non-standard data.

  As we explored this avenue of generating data, we quickly found that it is difficult to reliably generate punctuation errors in a manner consistent with the distribution of errors that occur in real life on the internet. As such, we were unable to get any meaningful results through these efforts, and decided to exclude it from our text normalizer.

Due to lack of possible data, we were not able to obtain accurate results from the following hypothesis producers:

- **Prefix**: recover formal word from words with prefixes that would otherwise be a formal word.
- **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.

Our scoring method was based loosely on the reference paper we followed. We combined a language model score, which was penalized by the number of hypotheses it took to normalize the text. in order to holistically rank each normalization. The reason our scoring metric was suboptimal in performance evaluation was two-fold. The main reason was that the corpus we obtained to develop our simple language model was not that large (only around 400 sentences), which leaves plenty of room for improvement. Secondly, each component of our score needed to have been summed with a particular weight attached, which was beyond our means to compute. We used subjective metrics with a bias towards the language model score itself to assign these weights. Ideally, we would have liked to use BLEU scoring as a metric to gauge our translations, but there was one major flaw in that strategy. BLEU compares reference human translations to machine translations, however, the inputs and outputs for those translations would have to come from a parallel corpus. As there is no corpus where the standard AND nonstandard versions of an input language are also translated into a foreign language by humans, BLEU could not have helped us.

5 Findings

The most important issue that we ran into was our scoring function. Indeed, in the inspired paper, the authors explain that they weighted each function using "using the pairwise ranking optimization algorithm (Hopkins and May, 2011) on the development set". Given the vague nature of the description and the distinct lack of data available to us for training, our scoring function was quite inadequate and made it very difficult to achieve proper performance.

Moreover, although stack pruning proved useful, our decoder proved very computationally and spacially inefficient. Indeed, given 10 different hypothesis producers, there are $10! \approx 3500000$ different possibilities for our decoders.

As for our experimental data, over the set of tweets, we saw an average improvement of -13.802 by language model score log probability.