INF5820 (Fall 2018): Fourth and Final Obligatory Exercise

High-Level Goals

- Approach the task of *negation resolution* as a sequence labeling problem;
- read and partially *replicate* (relevants parts of) current research literature;
- *contrast* joint cue and scope prediction with using gold-standard cues;
- do it yourself: *reduce wasteful copies* and dependence on high-level helper.

Background

This is the (absolutely) *final* obligatory assignment in INF5820. Like in the preceding assignment, we build closely on recent research literature and use our own implementation for a (possibly critical) review. At the same time, this problem set has a somewhat higher structural complexity and is more open-ended than previous assignments; in this regard, it is arguably representative of a genuine (if small-scale) research project in contemporary natural language processing. You can obtain up to twelve points for this problem set. If you have any questions, please make sure to attend the laboratory seesions on November 8 and 15 and ask there, or email `inf5820-help@ifi.uio.no` for assistance.

As always, solutions must be submitted via Devilry in the form of a 'research report' (of between four to eight pages, as a single PDF file) by late evening (22:00) on Friday, November 16, 2018. Your submission must include a reference to your code comitted to a private repository—shared only with the course instructors—in the UiO installation of Microsoft GitHub. Like with previous assignments, it will be important to structure and comment your code clearly, so that it will be a pleasure for us to review. Also, we will need to run your best-performing systems on a held-out evaluation set (which comes in exactly the same format as the training and development data), hence please provide full instructions on how to invoke the code for each of the sub-problems in Parts 2 and 3 below.

Starting Package

A collection of data files and utility code is available through the course repository in Microsoft GitHub at UiO; please see: `https://github.uio.no/inf5820/course2018.git`.

For this exercise we provide two data files, which provide the 'official' training and evaluation splits, respectively, of the annotations from the International Shared Task on Resolving the Scope and Focus of Negation, which was part of the First Joint Conference on Lexical and Computational Semantics (*SEM) in 2012 (Morante and Blanco, 2012). Because the underlying texts for this data draw on the writing of Sir Arthur Conan Doyle (which today are no longer protected by copyright), these files are called `cdt.epe` and `cdd.epe` (for training and development, respectively).

In the *SEM 2012 data, annotations of negation come in two forms: (a) so-called negation *cues* (color-coded in red below), which are indicators of counter-factuality; and (b) corresponding negation *scopes* (in blue), which are the parts of the utterance affected by the negation. Because there is more structure and complexity in this data than we have encountered in the class so far, Section 1 below will extract some statistics from the data to both get a better sense of the distributions of cues and scopes and to learn how to manipulate the EPE serialization format.

(1) I said that {I had} ⟨not⟩ .

We provide a 'toy' data file for debugging purposes, which comes in several different file formats: `toy.epe`, `toy.txt`, `toy.*sem`, and `toy.tt`. These files contain the negation annotations for a single sentence, albeit one with substantial internal complexity, viz.

(2) Mr. Sherlock Holmes , {who was}^{2} usually {very late in the mornings ,}^{2} $_{2}$⟨save⟩ {upon {{those}^{0}}^{1} $_{1}$⟨not⟩ {$_{0}$⟨in⟩{frequent occasions when he was up all night}^{0}}^{1}}^{2} , was seated at the breakfast table .

Besides the data files, we also provide some auxiliary code to (a) read and write negation annotations in different formats (in `convert.py`) and (b) invoke the official scorer from the *SEM 2012 Shared Task (in `score.py` and `eval.cd-sco.pl`). To invoke the latter, you will need a Perl interpreter installed on your system, which for the time being appears to be available by default on typical Un*x and MacOS environments.

1 Preliminaries: Making Sense of the EPE File Format (2 Points)

In example (2) above, there are three partly overlapping instances of negation in a single sentence. Cue identities are indicated by leading subscripts, and for each cue the corresponding scope is identified by trailing superscripts. For example, the scope associated with cue #1 (*not*) is the discontinuous string *those ... infrequent occassions when he was up all night*. To simplify our task, we will 'multiply out' instances of negation when there is more than one cue in the same sentence, i.e. we will make as many copies of the sentence as there are negation instances and distribute the cues and scopes for each instance accordingly.

(a) To exercise this approach to dealing with multiple negation instances, please rewrite example (2) into three sentences, where each only contains the negation annotations for either cue #0 (*in-*), #1 (*not*), or #2 (*save*).

Negation cues predominantly take the form of single tokens (e.g. *not* or *without*), but there are some occurences of multi-token cues (e.g. *no more* or *neither ... nor*) as well as of sub-token (or affix) cues (e.g. the prefix *in-* or suffix *-less*). In the case of affix cues, by convention the remainer of the word is considered in the scope of the affix, e.g. *-frequent* in the case of cue #0 above. Typical negation scopes, on the other hand, are non-contiguous sequences of tokens, for example corresponding to a full noun phrase (for cues #0 and #1 in example (2)) or to the 'main participants' in a sentence, if negation applies at higher levels of structure; however, Morante and Blanco (2012) assert that the decision which parts of an utterance form the scope of negation must be made on semantic grounds, rather than directly in terms of syntactic structures.

We make available training and development data in the file format of the recent Extrinsic Parser Evaluation (EPE) campaigns[1], which takes the form of so-called JSON lines files. We prefer JSON over the original fixed-column file format used for *SEM 2012 because it affords more flexibility and makes it very easy to discard or add additional annotations. In a nutshell, each EPE line describes the JSON object for one sentence, where a top-level property `nodes` provides an array of tokens (i.e. 'words' of

[1] See `http://epe.nlpl.eu/index.php?page=5#format` for a semi-formal description.

the sentence, where most punctuation marks are considered tokens in their own right). For our purposes, relevant properties on each node include `form` (a string, the surface form of the token), `properties` (a dictionary containing `lemma` and `xpos` strings), and `negation` (an array, recording for each negation instance whether the token functions as a cue, in-scope element, or both.

(b) Study the JSON structure in `toy.epe` and write a reader for the JSON lines format; as the built-in JSON module in Python appears ill-equipped to read multiple JSON objects from a stream, it may be necessary to (somewhat wastefully) read each line into a string first, to then invoke the JSON parser.

(c) Write an additional output function (for a list of sentences, read from the EPE file format), which may prove useful for debugging down the road. The output should be in so-called `.tt` format (for token'n'tags), where each surface form is shown on a line of its own, followed by a tabulator character and the 'tag' associated with the token. In our case, tags (or word classes, of sorts) will be indicators of negation cues and associated scopes. To get started, we will distinguish the following four classes: `C`(cue), for full-token cues; `A`(ffix), sub-token cues, `F`(alse), for tokens in the scope of negation; and `T`(rue), for tokens outside the scope. Please see the example file `toy.tt` for reference on the output format.

Finally, to familiarize ourselves more with the linguistic properties of the data (and flex our muscles traversing the EPE structures), we ask that you extract some summary statistics (which our team indeed found helpful when originally participating in the *SEM 2012 competition).

(d) Extract a dictionary and frequency counts of all negation cues; show the distribution of the ten most frequent cues (for example as a histogram). Further, extract frequency-sorted lists of all multi-token and sub-token (aka affix) cues.

(e) Determine the average length (as the count of in-scope tokens) of all (possibly discontinuous) scopes for the ten most frequent cues.

2 A Joint Cue and Scope Sequence Tagger (4 Points)

To build a first full system for negation resolution, we approach the task via *sequence labeling*, i.e. assigning each token (in context) one of the four classes above. Albeit uninformed about syntactic and semantic structure, this simple approach has proven surprisingly viable for negation resolution and is at the core of some of the top-performing systems. To the best of our knowledge, the state of the art for the *SEM 2012 data set is currently defined by Lapponi, Oepen, and Øvrelid (2017). In the neural age, sequence labeling problems tend to call for variants of Recurrent Neural Networks (RNNs), notably bidirectional LSTM or GRU architectures.

(a) Build an end-to-end negation resolution system in Keras, with sequences of tokens (surface forms) as its inputs and corresponding sequences of classes as outputs—`C`(ue), `A`(ffix), `F`(alse), and `T`(rue). The Keras RNN implementations require that all sequences be padded to a (suitably large) fixed length (using 'dummy' token and class symbols), and as is so often the case we will have to be prepared for previously unseen words during decoding. Initially, simply use random initialization for the input embedding layer, and try common values for its dimension, say anywhere between 50 and 300; likewise, maybe use a couple hundred units for the hidden 'memory' layer. Make informed guesses about good hyper-parameter values for e.g. dropout, optimizer, learning rate, batch size, and the stopping criterion; briefly discuss your choices. Contrast an LSTM vs. a GRU layer (with otherwise identical hyper-parameters in the overall system) in terms of network size (i.e. the total number of parameters), training time, and output accuracy (see below).

The Keras input and output layers deal in terms of multi-dimensional NumPy arrays. It appears quite common to construct input and output representations (e.g. the interface to the Keras `fit` and `predict`

calls) in a step-wise, piecemeal manner; for example, it can be convenient to string together high-level utilities like `to_categorical` and `pad_sequences`. We have yet to encounter truly large training data sets, but in any case one should observe that common coding practices can lead to *wasteful copying* 'behind the scenes', i.e. unnecessarily increase the memory footprint and generation of garbage. For example, a seemingly innocuous operation like the following will in fact generate two full representations of the sequence, first as a native Python list, then (in a separate object, with its own memory allocation) as a NumPy array:

```
np.array([foo for foo in range(42)])
```

Every byte counts!

(b) To help protect the environment, avoid all wasteful copying, i.e. determine beforehand the shapes of all NumPy arrays that provide input and output representations for Keras models and fill in their values directly—without auxiliary, intermediate lists or arrays and piecemeal combination, conversion, or re-shaping.

Converting neural class predictions into the JSON-like EPE structures will require putting a (possibly empty) `negation` property on each token, where the only two relevant keys inside the elements on the `negation` array will be `cue` and `scope`, which indicate whether the full token string or a part of it serve as either a cue or in-scope element, respectively. Recall that for affix cues (and only there), it is expected that one sub-string of the token form can be a cue, while the remainder of the string will be considered as in scope. Class predictions of `A`(ffix) will thus require some post-processing, where dictionaries of known (from the training data) prefixes and suffixes for sub-token cues may be a useful resource. Furthermore, the framework for negation annotation assumed in the *SEM 2012 context (quite intuitively) requires the presence of at least one cue (sub-)token for there to be any in-scope tokens. As the sequence labeler at the core of our system is unaware of such wellformedness conditions on negation instances, it will likely be beneficial to enforce these constraints in post-processing.

(c) Evaluate your system in terms of 'tagging' accuracy on the `cdd.epe` development data, both across all classes and as per-class accuracy. Padding to a fixed length leads to a large number of uninteresting predictions, seeing as classifier performance on the 'dummy' output class is most likely uninformative for accuracy on the actual negation class labels. Thus, make sure to avoid score inflation in calculating tagging accuracy.

(d) At this point, we have a functional negation resolution system. Turn to our file `score.py` and work out how to have two sequences of EPE-like negation structures evaluated by the official *SEM 2012 scorer. There is a decent summary of the various metrics computed by this script in the paper by Lapponi et al. (2017), and for our purposes we will be primarily concerned with two scores, viz. F_1 for 'cues' (the first line of values in the scorer output) and for 'scope tokens' (the fourth line). Compare your end-to-end negation results to observations on tagging accuracy above and discuss any relevant correlations (or lack thereof), as you deem appropriate.

3 Zooming in on Negation Scope (3 Points)

The negation system by Lapponi et al. (2017) combines a pre-neural approach to sequence labeling with a state-of-the-art neural dependency parser and somewhat carefully engineered lexical and structural linguistic features. The first successful application of 'deep' neural networks to the *SEM 2012 task was presented by Fancellu, Lopez, and Webber (2016), and in this pre-final part of the assignment we will seek to replicate some of their success.

(a) Read Sections 1 through 4 of Fancellu et al. (2016) and summarize where their architecture is similar to your design from Part 2 above and where it differs. For this comparison, focus on the model

called BiLSTM–C by Fancellu et al. (2016). How exactly is their experimental setup different from what we have done so far (and from what is the standard interpretation of the *SEM 2012 negation problem), and how does their architecture put to use information from the gold-standard annotations?

It would not seem unreasonable to expect that fairly minor revisions of your model architecture and experimental protocol will yield end-to-end performance comparable to the results reported by Fancellu et al. (2016). Note, however, that we have led you to adopt a linguistically more sensible approach to sub-token cues, and we expect you to stick to that choice. Fancellu et al. (2016) actually split off prefix and suffix cues as 'pseudo-'tokens in their own right (using a lexical resource that is not generally available, introducing some errors, and breaking compatibility with standard tokenization conventions or pre-trained word vectors); in contrast, our approach treats affix tokens as a separate class and determines the cue vs. scope sub-strings in post-processing.

(b) Make the necessary changes in your model architecture to include gold-standard cue information (which, after all, is readily available in the `.epe` files) among the network inputs. Also, adjust your use of the training and 'evaluation' data (where the latter, currently, is the *SEM 2012 development portion) to match the setup of Fancellu et al. (2016). In the spirit of a faithful replication experiment, seek to extract from the paper as much insight into hyper-parameters as possible. When in doubt, make your own assumptions and experimentally search for good values, or consider looking for additional parameter details in the publicly available code of Fancellu et al. (2016): `https://github.com/ffancellu/NegNN.git`. How does the revised architecture and setup affect classification accuracy and end-to-end performance for the different negation sub-problems? Discuss your results in comparison to those reported by Fancellu et al. (2016).

4 Error Analysis: Looking at our Data (3 Points)

One promise of the neural age is to reduce the importance of linguistic feature engineering (as if that were a bad thing in and of itself :-). This development, combined with the availability of high-level toolkits like Keras, arguably has led to a reduced focus on analytical study of the training data proper, as well as maybe to diminished interest in more in-depth analysis of system behavior (to some degree, the paper by Fancellu et al., 2016, forms a notable exception). In this concluding part of the assignment, we shall perform an error analysis of at least one of the systems from Parts 2 or 3 above. While competitive quantitative results can be rewarding, reducing the problem of negation resolution to a couple of F_1 scores will not by itself lead to improved understanding of the problem or the techniques we have adopted.

(a) Instrument your code to output all token-level errors, i.e. misclassifications in comparison to the gold standard. Review all errors in system predictions for the `cdd.epe` data and characterize the nature of the error. Seek to develop a 'taxonomy' of error types that can help to quantify the impact of different errors types, i.e. decide on a meaningful inventory of, say, five to ten different *classes* of errors and determine the frequencies of different types of errors. Do you observe linguistic properties of the inputs that appear to co-occur frequently with specific types of errors?

(b) In the light of your findings, conclude your research report with a 'Future Work' section. In other words, speculate about remaining limitations in your negation system and possible architecture revisions or use of additional knowledge sources that, in your view, bear promise of enabling the system to reduce the impact of specific error classes.

Happy coding!

References

Fancellu, F., Lopez, A., and Webber, B. (2016, August). Neural networks for negation scope detection. In *Proceedings of the 54th Meeting of the Association for Computational Linguistics* (p. 495 – 504). Berlin, Germany. Retrieved from http://www.aclweb.org/anthology/P16-1047

Lapponi, E., Oepen, S., and Øvrelid, L. (2017). EPE 2017: The Sherlock negation resolution downstream application. In *Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation* (p. 21 – 26). Pisa, Italy. Retrieved from http://epe.nlpl.eu/2017/49.pdf

Morante, R., and Blanco, E. (2012). *SEM 2012 Shared Task. Resolving the scope and focus of negation. In *Proceedings of the 1st Joint Conference on Lexical and Computational Semantics* (p. 265 – 274). Montréal, Canada. Retrieved from http://www.aclweb.org/anthology/S12-1035