Contents

- Introduction
- List of Files
- Running the Paralex Demo
- Running the Paralex Experiments
- Running the Paralex Learning Algorithms

Introduction

This document describes how to reproduce the experimental results of the Paralex system described in the ACL 2013 paper:

```
@inproceedings{Fader13,
    author    = {Anthony Fader and Luke Zettlemoyer and Oren Etzioni},
    title     = {{Paraphrase-Driven Learning for Open Question Answering}},
    booktitle = {Proceedings of the 51st Annual Meeting of the Association
                 for Computational Linguistics},
    year      = {2013}
}
```

You can download the paper here: http://homes.cs.washington.edu/~afader/bib_pdf/acl13.pdf

This code also comes with a simple web demo and HTTP/JSON API for running Paralex on your own questions.

If you have questions, please email Tony Fader at afader@cs.washington.edu.

List of Files

There are six subdirectories in the `data` directory: `database`, `questions`, `lexicons`, `weights`, `train`, and `output`.

Database Files

The files in `data/database` are related to the ReVerb extractions used in the QA system. There are two files:

- `vocab.txt` - A mapping between integers and database strings.
- `tuples.db` - A `sqlite3` database containing the `(arg1, rel, arg2)` ReVerb triples.

The ReVerb data is from the ReVerb ClueWeb09 extraction corpus, which can be downloaded from http://reverb.cs.washington.edu/.

Question Files

The data for the test questions described in Section 7.1 of the paper is in the directory `data/questions`.

The file `questions.txt` lists the questions, one per line. Each question comes from a cluster of paraphrased questions that are known to be answerable using the ReVerb database.

The file `clusterid_questions.txt` lists the paraphrase-cluster ID for each question.

The file `labels.txt` lists the ground-truth labeling for each question-answer pair. The set of question-answer pairs were generated by taking the top 100 answers for each question from each system tested in Section 7.

The format of `labels.txt` is a list of annotations, each with three tab-separated fields. The fields are: - The ground truth label (0 for incorrect, 1 for correct). - The question. - The answer tuple, in the form `R X Y` where `R` is a database relation and `X` and `Y` are arguments.

Lexicon Files

The files in `data/lexicons` contain the lexical entries used by Paralex to parse natural language questions into executable queries. Each lexicon is in a subdirectory of `data/lexicons` and contains the following files:

- `vocab.txt` - A mapping between integers and natural language strings, including strings representing relations, strings representing entities, 1-argument question templates, and 2-argument question templates.
- `lexicon.txt` - A mapping between integers and lexical entries. The lexical entries are encoded as a list of integers.

Table 2 in the paper gives an overview of the types of lexical entries in the lexicon files. There are: - Entity entries, which are (string, db entity) pairs. - Relation entries, which are (string, db relation, arg order) triples. - 1-Argument question entries, which are (template, db relation, arg order) triples. - 2-Argument question entries, which are (template, arg order) pairs.

Each lexical entry is encoded as a tuple of integers. The encodings are defined use the integers in `vocab.txt` used to represent strings and templates, the integers in `data/database/vocab.txt` used to represent the database entities and relations, and a set of special integers that represent the type of database semantics and argument ordering. There are two argument ordering options: arg2 is the query variable (represented by the integer `0` and arg1 is the query variable (represented by the integer `1`).

- Entity entries are encoded as `NL# 0 DB#` where `NL#` is a natural language string id and `DB#` is a database entity id.
- Relation entries and 1-argument question entries are encoded as `NL# 3 ORDER# DB#` where `ORDER#` is the argument ordering, and `DB#` is the database relation id.
- 2-argument entries are encoded as `NL# 1 ORDER#`.

The included lexicons are:

- `paralex` - The learned lexicon
- `initonly` - The initial lexicon used in the experiments
- `no1arg` - The learned `paralex` lexicon without the learned 1-argument templates.
- `no2arg` - The learned `paralex` lexicon without the learned 2-argument templates.
- `noent` - The learned `paralex` lexicon without the learned entity entries.
- `norel` - The learned `paralex` lexicon without the learned relation entries.

Weights Files

The files in `data/weights` contain feature weights for Paralex. Paralex scores a derivation from a question to a query as a linear combination of indicator functions. There is one indicator function for each entry in the lexicon that equals `1` if the entry was used in the derivation, and `0` otherwise.

The file `data/weights/paralex.txt` contains the learned feature weights using the algorithm in Figure 2 of the paper. The format of the file is space-separated (weight, lexical entry) where the lexical entry is represented using the integer encoding described in the previous section.

Training Data Files

The files in `data/train` contain the heuristic training data generated in Step 1 of the parameter learning algorithm (Figure 2).

The file `labeled.txt` contains a list of training examples. Each example is in the form `(question, query1, query2, ...)` where each field is separated by a tab. The question is represented as a string. Each query is encoded as a list of integers in the form `2 ORDER# REL# ENT#` where `ORDER#` defines the argument ordering of the query (`0` for arg2 being the query variable, `1` for arg1 being the query variable), and `REL#` and `ENT#` are the database integers representing the relation and entity constants in the query (see `data/database/vocab.txt` for the string representations of these).

The file `labeled_plain.txt` is for convenience and contains the same data, but represented as strings. Each line contains a tab-separated `(question, query)` pair. The questions are strings, and the queries are represented as lambda-calculus expressions. For example, the expression `(lambda $x (be-capital-of.r $x belarus.e))` represents the query that returns all database entities `$x` such that `$x` satisfies the `be-capital-of.r` relation with the entity `belarus.e`.

The file `paraphrases.txt` contains the word-aligned paraphrases from WikiAnswers.

Output Files

The files in `data/output` contain system outputs on the evaluation questions. Each file contains one answer per line, where each output is in the form `(question, confidence, answer)`. The `question` is taken from the file `data/questions/questions.txt`. The `confidence` is a real number that represents the system's confidence in the answer. The `answer` field is a tuple that represents the system's answer. There is one file for each test system in the `data/output` directory.

Running the Paralex Demo

I have included a simple web demo for testing out Paralex on your own questions. The demo requires starting a server for running the Stanford NLP tools, and a web server for interacting with Paralex.

To start the Stanford NLP server, run the following command:

```
./scripts/start_nlp.sh &
```

This will start an instance of the Stanford NLP tools that can be accessed via HTTP over port 8082. The code for this server was written by Dustin Smith and can be downloaded here:

```
https://github.com/dasmith/stanford-corenlp-python
```

The code for this is in the external/ subdirectory.

The next step is to start the web server. To do this, you must have python 2.7 installed, and the bottle.py web library. To start the server, run this command:

```
./scripts/start_demo.sh &
```

This will start a web server on port 8083. The code for the web demo is in web/web.py. Once the web server has started, you should be able to open your browser to http://localhost:8080 and try out Paralex.

Once the demo is running, you can make HTTP requests to Paralex and get JSON objects as output.

```
http://localhost:8083/parse?sent=Who+invented+pizza
```

Running the Paralex Experiments

There are two steps to recreate the experiments in Section 7. First, you need to run Paralex on the test questions in `data/questions/questions.txt`. Then, you need to evaluate the output relative to the ground-truth labels in `data/questions/labels.txt`.

First, make sure the Stanford NLP server is running (see previous section).

Next, run Paralex on the input questions:

```
./scripts/run_qa.sh data/questions/questions.txt data/lexicons/paralex data/weights/paral
```

This will run Paralex on the given questions file, using the lexicon in `data/lexicons/paralex`, the feature weights in `data/weights/paralex.txt`, and the database in `data/database`, writing the output to `output.txt`. Debug statements will be written to stderr. You can use the other lexicon and weight files to recreate the other systems' behaviors.

Next, evaluate the output against the ground-truth labels by running the following command:

```
./scripts/run_eval.sh data/questions/questions.txt data/questions/labels.txt output.txt
```

This will print the precision, recall, f1, and other metrics to the screen.

Running the Paralex Learning Algorithms

Paralex uses two learning algorithms: a lexicon-learning algorithm (shown in Figure 1) and a weight-learning algorithm (shown in Figure 2).

In this section, I describe how to run the learning algorithm scripts. I do not include the code to run the algorithms on a cluster. The original experiments were run using code that is specific to our (UW) cluster. However, the scripts in the following sections should be the easy to string together on your own preferred cluster environment.

Lexicon Learning

The lexicon-learning algorithm takes an initial lexicon, initial weights, and a word-aligned paraphrase corpus as input. It returns as output a learned lexicon and a training set of `(question, query)` pairs.

To run the lexicon learning algorithm, use the following command:

```
./scripts/run_lexlearn.sh paraphrases initlex initweights outlex outlabs
```

This will use the paraphrase corpus `paraphrases` (e.g. `data/train/paraphrases.txt`), the initial lexicon in the directory `initlex` (e.g. the directory `data/lexicons/initonly`), and the initial weights (e.g. the zero weights in `data/weights/zero.txt`), and write the output learned lexicon to the directory `outlex` and labeled data to `outlabs`.

This algorithm is data-parallel, and can be run independently on different subsets of the input paraphrases. Two lexicons can be merged together by running the command:

```
./scripts/run_mergelex.sh lex1 lex2 outlex
```

This will merge the lexicon directories `lex1` and `lex2` into a single lexicon in `outlex`.

Weight Learning

The weight-learning algorithm is done by partitioning the training data, running independent perceptron learners on each partition, and then averaging their learned weight vectors. This is repeated `T` times, using the averaged weight vector from the previous iteration as the initial weights for the current iteration's learners.

To run a perceptron learner on training data, execute the command:

```
./scripts/run_weightlearn.sh lexicon initweights labels outweights
```

This can be executed on separate processors to generate multiple weight files. To average multiple weight files, run the following command:

```
./scripts/run_aveweights.sh input1 input2 ... inputN averaged
```

This will take the average of the `N` weights files and write the output to the output file `averaged`.