elit Documentation

Gary Lai

Feb 19, 2020

Quick Start

The ELIT project brings the latest natural language processing (NLP) technology to the community. The primary focus of this project is to present an end-to-end NLP framework that is robust across various domains as well as scalable for big data analysis. ELIT supports python APIs for developers to integrate its framework to their platforms as well as web APIs for researchers to take advantage of its cutting-edge models via SaaS (Software as a Service). ELIT is under the Apache 2 license and developed by Emory NLP with an active collaboration with GluonNLP.

- Latest release: 0.2.0.

CHAPTER 1

Installation

1.1 GPU Machine

Deep learning models in ELIT require a GPU machine. Take a look at the pre-trained models page to check how much GPU memories are needed for models of your choice. If you do not have a GPU machine powerful enough to run those models, you may want to consider using our web APIs.

1.2 CUDA Environment

ELIT requires the following CUDA environment:

- Supported versions of the CUDA Toolkit: 8.0, 9.0, 9.2, 10.0, 10.1.
- CUDA Deep Neural Network library: cuDNN.

The followings show how to install the CUDA Toolkit 10.1 on Ubuntu 18.04:

```
$ wget http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/cuda-
↪repo-ubuntu1804_10.1.168-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu1804_10.1.168-1_amd64.deb
$ sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/
↪repos/ubuntu1804/x86_64/7fa2af80.pub
$ sudo apt update
$ sudo apt install cuda
$ nvidia-smi
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.67       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
```

The followings show how to install the cuDNN 7.6.0.64 on Ubuntu 18.04:

```
$ wget https://developer.download.nvidia.com/compute/machine-learning/repos/
↪ubuntu1804/x86_64/libcudnn7_7.6.0.64-1+cuda10.1_amd64.deb
$ sudo dpkg -i libcudnn7_7.6.0.64-1+cuda10.1_amd64.deb
$ sudo apt install libcudnn7
libcudnn7 is already the newest version (7.6.0.64-1+cuda10.1).
```

1.3 Python Environment

ELIT requires Python 3.6. The followings show how to install the Python 3.6.8 on Ubuntu 18.04:

```
$ sudo apt install python3.6
$ sudo apt install python3.6-dev
$ sudo apt install python3-setuptools
$ sudo apt install python3-pip
$ sudo apt install python-virtualenv
$ python3 --version
Python 3.6.8
```

1.4 Virtual Environment

We recommend to install ELIT using Virtualenv. The followings show how to setup a virtualenv using Python 3.6:

```
$ virtualenv --python=/usr/bin/python3.6 ~/.elit
$ source ~/.elit/bin/activate
(.elit) $
```

1.5 MXNet Installation

ELIT uses Apache MXNet to develop deep learning models. The followings show how to install MXNet 1.4.1 based on CUDA Toolkit 10.1:

```
(.elit) $ pip install mxnet-cu101
(.elit) $ pip show mxnet-cu101
Name: mxnet-cu101
Version: 1.4.1
```

1.6 ELIT Installation

Finally, the followings show how to install the latest version of ELIT:

```
(.elit) $ pip install elit
Name: elit
Version: 0.2.0
```

CHAPTER 2

Decode with Python APIs

Once ELIT is installed, NLP components can be used to decode raw text into NLP structures.

2.1 Create NLP Tools

The followings show how to create NLP tools for 6 core components:

```
from elit.component import EnglishTokenizer
from elit.component import EnglishMorphAnalyzer
from elit.component import POSFlairTagger
from elit.component import NERFlairTagger
from elit.component import DEPBiaffineParser
from elit.component import SDPBiaffineParser

tok = EnglishTokenizer()
morph = EnglishMorphAnalyzer()
pos = POSFlairTagger()
ner = NERFlairTagger()
dep = DEPBiaffineParser()
sdp = SDPBiaffineParser()

tools = [tok, pos, morph, ner, dep, sdp]
```

Take a look at the individual tool page for more details about available components:

- Tokenizer
- Morphological analyser
- Part-Of-Speech tagger
- Named Entity recognizer
- Dependency parser
- Semantic dependency parser

2.2 Import Models

All pre-trained models are publicly available in the ELIT's S3 bucket. The followings show how to import models for `pos`, `ner`, `dep`, and `sdp`:

```
from elit.resources.pre_trained_models import ELIT_POS_FLAIR_EN_MIXED
from elit.resources.pre_trained_models import ELIT_NER_FLAIR_EN_ONTONOTES
from elit.resources.pre_trained_models import ELIT_DEP_BIAFFINE_EN_MIXED
from elit.resources.pre_trained_models import ELIT_SDP_BIAFFINE_EN_MIXED

pos.load(ELIT_POS_FLAIR_EN_MIXED)
ner.load(ELIT_NER_FLAIR_EN_ONTONOTES)
dep.load(ELIT_DEP_BIAFFINE_EN_MIXED)
sdp.load(ELIT_SDP_BIAFFINE_EN_MIXED)
```

The `load` function takes two parameters, `model_path` and `model_root`:

- `model_path` indicates either the name of the model (e.g., `elit_pos_flair_en_mixed_20190626`) or a public URL to the model file compressed in the zip format (e.g., https://elit-models.s3-us-west-2.amazonaws.com/elit_pos_flair_en_mixed_20190626.zip).
- `model_root` indicates the root directory in the local machine where all models are saved, and has the default value of `~/.elit/models/`.
- If `model_path` points to an URL, this function downloads the remote file and unzips it under the directory indicated by `model_root`, which will create a directory with the same model name (e.g., `~/.elit/models/elit_pos_flair_en_mixed_20190626/`).
- Each model directory has the configuration file, `config.json`, that may indicate dependencies to other models, in which case, it will recursively download all necessary models and unzip them under `model_root` (see Train with CLI for more details about how models are saved).

2.3 Prepare Raw Text

The followings show how to prepare raw text for decoding:

```
docs = [
    'Emory University is a private research university in Atlanta, Georgia. The␣
↪university is ranked 21st nationally according to U.S. News.',
    'Emory University was founded in 1836 by the Methodist Episcopal Church. It was␣
↪named in honor of John Emory who was a Methodist bishop.']
```

ELIT accepts a list of strings as input, where each string represents a document such that there are two documents in `docs`.

2.4 Decode with NLP Tools

Finally, the followings show how to decode the raw text with the NLP tools:

```
for tool in tools:
    docs = tool.decode(docs)
```

The `decode` function in the tokenizer takes a list of strings and returns a list of Document, whereas the `decode` functions in other models take a list of document objects and return a list of the same objects where the decoding results are added as distinct fields (see the *NLP Output* below).

2.5 All Together

The followings put all the codes together:

```
from elit.component import EnglishTokenizer
from elit.component import EnglishMorphAnalyzer
from elit.component import POSFlairTagger
from elit.component import NERFlairTagger
from elit.component import DEPBiaffineParser
from elit.component import SDPBiaffineParser

from elit.resources.pre_trained_models import ELIT_POS_FLAIR_EN_MIXED
from elit.resources.pre_trained_models import ELIT_NER_FLAIR_EN_ONTONOTES
from elit.resources.pre_trained_models import ELIT_DEP_BIAFFINE_EN_MIXED
from elit.resources.pre_trained_models import ELIT_SDP_BIAFFINE_EN_MIXED

tok = EnglishTokenizer()
morph = EnglishMorphAnalyzer()
pos = POSFlairTagger().load(ELIT_POS_FLAIR_EN_MIXED)
ner = NERFlairTagger().load(ELIT_NER_FLAIR_EN_ONTONOTES)
dep = DEPBiaffineParser().load(ELIT_DEP_BIAFFINE_EN_MIXED)
sdp = SDPBiaffineParser().load(ELIT_SDP_BIAFFINE_EN_MIXED)

tools = [tok, pos, morph, ner, dep, sdp]

docs = [
    'Emory University is a private research university in Atlanta, Georgia. The␣
↪university is ranked 21st nationally according to U.S. News.',
    'Emory University was founded in 1836 by the Methodist Episcopal Church. It was␣
↪named in honor of John Emory who was a Methodist bishop.']

for tool in tools:
    docs = tool.decode(docs)

print(docs)
```

2.6 NLP Output

The followings show the printed output of the above code:

```
To be filled
```

See the Formats page for more details about how the decoding results are added to Document.

CHAPTER 3

Decode with Web APIs

ELIT provides web APIs to decode raw text into NLP structures using pre-trained models. The web APIs do not require installation and can be used by any programming language that supports HTTP request/response.

3.1 Decode via HTTP

The followings show how to send a list of documents to ELIT and receive the *NLP output* consisting of decoding results from 6 models:

- Tokenization: elit_tok_lexrule_en
- Morphological analysis: elit_morph_lexrule_en
- Part-Of-Speech tagging: elit_pos_flair_en_mixed
- Named Entity recognization: elit_ner_flair_en_ontonotes
- Dependency parsing: elit_dep_biaffine_en_mixed
- Semantic dependency parsing: elit_sdp_biaffine_en_mixed

Take a look at the individual model page for more details and their parameter settings.

Python

```
import requests

url = 'https://elit.cloud/api/public/decode/raw'

docs = [
    'Emory University is a private research university in Atlanta, Georgia. The␣
↪university is ranked 21st nationally according to U.S. News.',
    'Emory University was founded in 1836 by the Methodist Episcopal Church. It was␣
↪named in honor of John Emory who was a Methodist bishop.']

models = [
```

(continues on next page)

(continued from previous page)

```
    {'model': 'elit_tok_lexrule_en'},
    {'model': 'elit_pos_flair_en_mixed'},
    {'model': 'elit_morph_lexrule_en'},
    {'model': 'elit_ner_flair_en_ontonotes'},
    {'model': 'elit_dep_biaffine_en_mixed'},
    {'model': 'elit_sdp_biaffine_en_mixed'}]

request = {'input': docs, 'models': models}
r = requests.post(url, json=request)
print(r.text)
```

Java

```
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;

public class ELITWebAPI
{
    public static void main(String[] args) throws Exception
    {
        HttpPost post = new HttpPost("https://elit.cloud/api/public/decode/raw");
        HttpClient client = HttpClientBuilder.create().build();

        String docs = String.format("["\%s\", "\%s\"]",
            "Emory University is a private research university in Atlanta, Georgia.␣
↪The university is ranked 21st nationally according to U.S. News.",
            "Emory University was founded in 1836 by the Methodist Episcopal Church.␣
↪It was named in honor of John Emory who was a Methodist bishop.");

        String models = String.format("[{\"model\": \"%s\"}, {\"model\": \"%s\"}, {\
↪"model\": \"%s\"}, {\"model\": \"%s\"}, {\"model\": \"%s\"}, {\"model\": \"%s\"}]",
            "elit_tok_lexrule_en",
            "elit_pos_flair_en_mixed",
            "elit_morph_lexrule_en",
            "elit_ner_flair_en_ontonotes",
            "elit_dep_biaffine_en_mixed",
            "elit_sdp_biaffine_en_mixed");

        String request = "{\"input\": " + docs + ", \"models\": " + models + "}";

        post.setEntity(new StringEntity(request, ContentType.create("application/json
↪")));
        HttpResponse response = client.execute(post);
        System.out.println(EntityUtils.toString(response.getEntity()));
    }
}
```

Add the following dependency to `pom.xml`.

```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
```

(continues on next page)

(continued from previous page)

```
    <version>4.5.9</version>
</dependency>
```

3.2 NLP Output

The followings show the printed output of the above code:

```
To be filled
```

See the Formats page for more details about how the decoding results are added to Document.

CHAPTER 4

Train with CLI

Train with command-line interface.

CHAPTER 5

Tokenization

A tokenizer takes raw text and splits it into string tokens. It also returns the begin (inclusive) and the end (exclusive) character offsets from the original text for each token. ELIT's tokenizers provide an option of performing several types of sentence segmentation, which groups chunks of consecutive tokens into sentences:

- `0`: no segmentation.
- `1`: segment by newlines (`\n`).
- `2`: segment by symbol rules.
- `3`: segment by `1` and `2`.

5.1 Space Tokenizer

The Space Tokenizer splits input text by whitespaces, which is useful when the input text is already tokenized (either manually or by some other tool) such that no further tokenization is necessary.

- Associated models: `elit_tok_space_en`
- API reference: SpaceTokenizer
- Decode parameters:
  - `segment`: `0`, `1` (*default*), `2`, or `3`

5.1.1 Web API

```
{"model": "elit_tok_space_en", "args": {"segment": 1}}
```

5.1.2 Python API

```
from elit.component import SpaceTokenizer
tok = SpaceTokenizer()
text = [
    'This is the 1st sentence\nThis is the 2nd sentence',
    'This is the 3rd sentence\nThis is the 4th sentence']
print(tok.decode(text, segment=1))  # segment by newlines (default)
```

5.1.3 Output


```
[
  {
    "doc_id": 0,
    "sens": [
      {
        "sid": 0,
        "tok": ["This", "is", "the", "1st", "sentence"],
        "off": [[0, 4], [5, 7], [8, 11], [12, 15], [16, 24]]
      },
      {
        "sid": 1,
        "tok": ["This", "is", "the", "2nd", "sentence"],
        "off": [[25, 29], [30, 32], [33, 36], [37, 40], [41, 49]]
      }
    ]
  },
  {
    "doc_id": 1,
    "sens": [
      {
        "sid": 0,
        "tok": ["This", "is", "the", "3rd", "sentence"],
        "off": [[0, 4], [5, 7], [8, 11], [12, 15], [16, 24]]
      },
      {
        "sid": 1,
        "tok": ["This", "is", "the", "4th", "sentence"],
        "off": [[25, 29], [30, 32], [33, 36], [37, 40], [41, 49]]
      }
    ]
  }
]
```


5.2 English Tokenizer

The English Tokenizer splits input text into linguistic tokens using lexicalized rules.

- Associated models: `elit_tok_lexrule_en`
- API reference: EnglishTokenizer
- Decode parameters:
  - `segment`: `0`, `1`, `2` (*default*), or `3`

The followings show key features of this tokenizer:

5.2.1 Web API

```
{"model": "elit_tok_lexrule_en", "args": {"segment": 2}}
```

5.2.2 Python API

```
from elit.component import EnglishTokenizer
tok = EnglishTokenizer()
text = [
    "Mr. Johnson doesn't like cats! What's his favorite then?",
    "He likes puffy-dogs. He is gonna buy one."]
print(tok.decode(text, segment=2))  # segment by symbol rules (default)
```

5.2.3 Output


```
[
  {
    "doc_id": 0,
    "sens": [
      {
        "sid": 0,
        "tok": ["Mr.", "Johnson", "does", "n't", "like", "cats", "!"],
        "off": [[0, 3], [4, 11], [12, 16], [16, 19], [20, 24], [25, 29], [29, 30]],
      },
      {
        "sid": 1,
        "tok": ["This", "is", "the", "2nd", "sentence"],
        "off": [[25, 29], [30, 32], [33, 36], [37, 40], [41, 49]]
      }
    ]
  },
  {
    "doc_id": 1,
    "sens": [
      {
        "sid": 0,
        "tok": ["He", "likes", "puffy", "-", "dogs", "."],
        "off": [[0, 2], [3, 8], [9, 14], [14, 15], [15, 19], [19, 20]]
      },
      {
        "sid": 1,
        "tok": ["He", "is", "gon", "na", "buy", "one", "."],
        "off": [[21, 23], [24, 26], [27, 32], [33, 36], [37, 40], [40, 41]]
      }
    ]
  }
]
```

CHAPTER 6

Morphological Analysis

6.1 English Analyzer

The English Analyzer takes an input token and its part-of-speech tag in the Penn Treebank style, and splits it into morphemes using inflection, derivation, and prefix rules.

- Associated models: `elit-morph-idprule-en`
- API reference: EnglishMorphAnalyzer
- Supplementary documentation
- Decode parameters:
  - `derivation`: `True` (*default*) or `False`
  - `prefix`: `0` (no prefix analysis; *default*), `1` (shortest preferred), `2` (longest preferred)

6.1.1 Web API

```
{"model": "elit_morph_lexrule_en", "args": {"derivation": true, "prefix": 0}}
```

6.1.2 Python API

```
from elit.structure import Document, Sentence, TOK, POS, MORPH
from elit.component import EnglishMorphAnalyzer

tokens = ['dramatized', 'ownerships', 'environmentalists', 'certifiable',
↪'realistically']
postags = ['VBD', 'NNS', 'NNS', 'JJ', 'RB']
doc = Document()
doc.add_sentence(Sentence({TOK: tokens, POS: postags}))
```

(continues on next page)

(continued from previous page)

```
morph = EnglishMorphAnalyzer()
morph.decode([doc], derivation=True, prefix=0)
print(doc.sentences[0][MORPH])
```

6.1.3 Output

```
[
  [["drama", "NN"], ["+tic", "J_IC"], ["+ize", "V_IZE"], ["+d", "I_PST"]],
  [["own", "VB"], ["+er", "N_ER"], ["+ship", "N_SHIP"], ["+s", "I_PLR"]],
  [["environ", "VB"], ["+ment", "N_MENT"], ["+al", "J_AL"], ["+ist", "N_IST"], ["+s",
↪"I_PLR"]],
  [["cert", "NN"], ["+ify", "V_FY"], ["+iable", "J_ABLE"]],
  [["real", "NN"], ["+ize", "V_IZE"], ["+stic", "J_IC"], ["+ally", "R_LY"]]
]
```

CHAPTER 7

Part-of-Speech Tagging

7.1 Flair Tagger

This is ELIT's replication of Flair's tagger using contextual string embeddings. It is ported from the PyTorch implementation of Flair version 0.2.

- Source: https://github.com/zalandoresearch/flair
- Associated models: `elit_pos_flair_en_mixed`
- API reference: POSFlairTagger
- Decode parameters: `none`

7.1.1 Web API

```
{"model": "elit_pos_flair_en_mixed"}
```

7.1.2 Python API

```
from elit.structure import Document, Sentence, TOK
from elit.component import POSFlairTagger

tokens = ['Jinho', 'Choi', 'is', 'a', 'professor', 'at', 'Emory', 'University', 'in',
↪'Atlanta', ',', 'Georgia', '.']
doc = Document()
doc.add_sentence(Sentence({TOK: tokens}))

pos = POSFlairTagger()
pos.decode([doc])
print(doc.sentences[0])
```

7.1.3 Output


```
{
  "sid": 0,
  "tok": ["Jinho", "Choi", "is", "a", "professor", "at", "Emory", "University", "in",
↪"Atlanta", ",", "Georgia", "."],
  "pos": ["NNP", "NNP", "VBZ", "DT", "NN", "IN", "NNP", "NNP", "IN", "NNP", ",", "NNP
↪", "."]
}
```


7.1.4 Citation

```
@InProceedings{akbik-blythe-vollgraf:COLING:2018,
  author    = {Akbik, Alan and Blythe, Duncan and Vollgraf, Roland},
  title     = {Contextual String Embeddings for Sequence Labeling},
  booktitle = {Proceedings of the 27th International Conference on Computational␣
↪Linguistics},
  year      = {2018},
  series    = {COLING'18},
  url       = {http://aclweb.org/anthology/C18-1139}
}
```

CHAPTER 8

Named Entity Recognition

8.1 Flair Tagger

This is ELIT's replication of Flair's tagger using contextual string embeddings. It is ported from the PyTorch implementation of Flair version 0.2.

- Source: https://github.com/zalandoresearch/flair
- Associated models: `elit_ner_flair_en_ontonotes`
- API reference: NERFlairTagger
- Decode parameters: `none`

8.1.1 Web API

```
{"model": "elit_ner_flair_en_ontonotes"}
```

8.1.2 Python API

```
from elit.structure import Document, Sentence, TOK
from elit.component import NERFlairTagger

tokens = ['Jinho', 'Choi', 'is', 'a', 'professor', 'at', 'Emory', 'University', 'in',
↪'Atlanta', ',', 'Georgia', '.']
doc = Document()
doc.add_sentence(Sentence({TOK: tokens}))

ner = NERFlairTagger()
ner.decode([doc])
print(doc.sentences[0])
```

8.1.3 Output

```
{
  "sid": 0,
  "tok": ["Jinho", "Choi", "is", "a", "professor", "at", "Emory", "University", "in",
↪"Atlanta", ",", "Georgia", "."],
  "ner": [[0, 2, "PERSON"], [6, 8, "ORG"], [9, 12, "LOC"]]
}
```

8.1.4 Citation

```
@InProceedings{akbik-blythe-vollgraf:COLING:2018,
  author    = {Akbik, Alan and Blythe, Duncan and Vollgraf, Roland},
  title     = {Contextual String Embeddings for Sequence Labeling},
  booktitle = {Proceedings of the 27th International Conference on Computational␣
↪Linguistics},
  year      = {2018},
  series    = {COLING'18},
  url       = {http://aclweb.org/anthology/C18-1139}
}
```

CHAPTER 9

Dependency Parsing

9.1 Biaffine Parser

This is ELIT's replication of the dependency parser using deep biaffine attention introduced by Stanford University.

- Source: https://github.com/tdozat/Parser-v1
- Associated models: `elit_dep_biaffine_en_mixed`
- API reference: [DEPBiaffineParser](../documentation/apidocs.html#elit.component.dep.dependency_parser.DEPBiaffineParser)
- Decode parameters: `none`

9.1.1 Web-API

```
{"model": "elit_dep_biaffine_en_mixed"}
```

9.1.2 Python API

```
from elit.structure import Document, Sentence, TOK, POS
from elit.component import DEPBiaffineParser

tokens = ['John', 'who', 'I', 'wanted', 'to', 'meet', 'was', 'smart']
postags = ['NNP', 'IN', 'WP', 'PRP', 'VBD', 'DT', 'NN', 'VBD', 'JJ']
doc = Document()
doc.add_sentence(Sentence({TOK: tokens, POS: postags}))

dep = DEPBiaffineParser()
dep.decode([doc])
print(doc.sentences[0])
```

9.1.3 Output


```
{
  "sid": 0,
  "tok": ["John", "who", "I", "wanted", "to", "meet", "was", "smart"],
  "pos": ["NNP", "WP", "PRP", "VBD", "TO", "VB", "VBD", "JJ"],
  "dep": [[7, "nsbj"], [5, "r-obj"], [3, "nsbj"], [0, "relcl"], [5, "aux"], [3, "comp
↪"], [7, "cop"], [-1, "root"]]
}
```


9.1.4 Citation

```
@InProceedings{dozat-manning:ICLR:2017,
  author    = {Dozat, Timothy and Manning, Christopher D.},
  title     = {Deep Biaffine Attention for Neural Dependency Parsing},
  booktitle = {Proceedings of the 5th International Conference on Learning␣
↪Representations},
  year      = {2017},
  series    = {ICLR'17},
  url       = {https://arxiv.org/abs/1611.01734}
}
```

CHAPTER 10

Semantic Dependency Parsing

10.1 Biaffine Parser

This is ELIT's replication of the semantic dependency parser using deep biaffine attention introduced by Stanford University.

- Source: https://github.com/tdozat/Parser-v3
- Associated models: `elit_sdp_biaffine_en_mixed`
- API reference: SDPBiaffineParser
- Decode parameters: `none`

10.1.1 Web-API

```
{"model": "elit_sdp_biaffine_en_mixed"}
```

10.1.2 Python API

```
from elit.structure import Document, Sentence, TOK, POS
from elit.component import SDPBiaffineParser

tokens = ['John', 'who', 'I', 'wanted', 'to', 'meet', 'was', 'smart']
postags = ['NNP', 'IN', 'WP', 'PRP', 'VBD', 'DT', 'NN', 'VBD', 'JJ']
doc = Document()
doc.add_sentence(Sentence({TOK: tokens, POS: postags}))

sdp = SDPBiaffineParser()
sdp.decode([doc])
print(doc.sentences[0])
```

10.1.3 Output

```
{
  "sid": 0,
  "tok": ["John", "who", "I", "wanted", "to", "meet", "was", "smart"],
  "pos": ["NNP", "WP", "PRP", "VBD", "TO", "VB", "VBD", "JJ"],
  "sdp": [[[7, "nsbj"], [5, 'obj']], [[5, "r-obj"]], [[3, "nsbj"], [5, "nsbj"]], [[0,
↪"relcl"]], [[5, "aux"]], [[3, "comp"]], [[7, "cop"]], [[-1, "root"]]]
}
```

10.1.4 Citation

```
@InProceedings{dozat-manning:ACL:2018,
  author    = {Dozat, Timothy and Manning, Christopher D.},
  title     = {Simpler but More Accurate Semantic Dependency Parsing},
  booktitle = {Proceedings of the 56th Annual Meeting of the Association for␣
↪Computational Linguistics},
  year      = {2018},
  series    = {ACL'18},
  url       = {https://www.aclweb.org/anthology/P18-2077}
}
```

CHAPTER 11

Release Notes

11.1 0.2

CHAPTER 12

Pre-trained Models

All models take the following naming convention:

```
[team]_[task]_[method]_[language]_[training_data]*
```

- `team`: the team who developed this model.
- `task`: the task that this model is developed for.
- `method`: the method used to develop this model.
- `language`: the input language (ISO 639-1; use `un` for universal models).
- `training_data`: the training data used to build this model (if applicable).

12.1 English Models

- `PRE`: tools that need to be run prior to the model.
- `DATA`: the dataset used to build the model.
- `EVAL`: evaluation on the dataset that the model is trained on.
- `BM`: the standard benchmark evaluation for the task.
- `tokseg`: any *tokenizer* with sentence segmentation.
- `inmixed`: any dataset in the mixed corpus.

12.1.1 Tokenization

12.1.2 Morphological Analysis

12.1.3 Part-of-Speech Tagging

- EVAL: accuracy.
- BM: accuracy on the Wall Street Journal portion of the Penn Treebank using the standard split (trn: 0-18; dev: 19-21; tst: 22-24).

12.1.4 Named Entity Recognition

- EVAL: F1-score.
- BM: F1-score on the English dataset distributed by the CoNLL 2003 shared task.

12.1.5 Dependency Parsing

- EVAL: UAS (unlabeled attachment score) / LAS (labeled attachment score).
- BM: UAS/LAS on the Wall Street Journal portion of the Penn Treebank using the standard split (trn: 2-21; dev: 22, 24; tst: 23) and the Stanford typed dependencies.

12.1.6 Semantic Dependency Parsing

- EVAL: Labeled F1 score.
- BM: Average labeled F1 scores on the in-domain and out-of-domain test sets distributed by the SemEval 2015 shared task.

12.2 English Datasets

- sc = sentence count.
- tc = token count.

12.2.1 CRAFT

- Colorado Richly Annotated Full Text Corpus
  - Biomedical journal articles (sc = 19,792, tc = 554,539)

12.2.2 BOLT

- Broad Operational Language Translation
  - Conversational telephone speech (sc = 11,552, tc = 160,319)
  - Discussion forum (sc = 17,382, tc = 396,584)
  - SMS message (sc = 22,883, tc = 260,431)

12.2.3 EWT

- English Web Treebank
  - Question-answer (sc = 3,089, tc = 50,404)
  - Email (sc = 3,436, tc = 51,504)
  - Newsgroup (sc = 2,122, tc = 41,891)
  - Review (sc = 2,951, tc = 45,864)
  - Weblog (sc = 1,886, tc = 42,988)

12.2.4 OntoNotes

- OntoNotes 5.0
  - Broadcasting conversation (sc = 14,648, tc = 239,940)
  - Broadcasting news (sc = 11,867, tc = 240,241)
  - News magazine (sc = 7,960, tc = 194,926)
  - Newswire (sc = 40,491, tc = 1,038,190)
  - Pivot text (sc = 24,386, tc = 339,013)
  - Telephone conversation (sc = 10,955, tc = 112,847)
  - Weblog (sc = 11,800, tc = 262,049)

12.2.5 QuestionBank

- QuestionBank Revised
  - Question (sc = 3,989, tc = 38,100)

12.2.6 MiPACQ

- Multi-source Integrated Platform for Answering Clinical Questions
  - Clinical note (sc = 9,706, tc = 132,235)
  - Clinical question (sc = 1,980, tc = 37,178)
  - Medpedia (sc = 2,921, tc = 49,252)
  - Pathological note (sc = 1,182, tc = 22,088)

12.2.7 SHARP

- Strategic Health IT Advanced Research Projects
  - Clinical note (sc = 7,841, tc = 111,789)
  - Seattle group health note (sc = 8,268, tc = 110,208)
  - Stratified (sc = 5,022, tc = 51,629)
  - Stratified Seattle group health note (sc = 15,948, tc = 165,960)

12.2.8 THYME

- Temporal History of Your Medical Events
- Brain cancer note (sc = 21,284, tc = 263,011)
- Clinical/Pathological note (sc = 30,090, tc = 448,603)

12.2.9 Mixed

A combined dataset consisting of CRAFT, BOLT, EWT, OntoNotes, QuestionBank, MiPACQ, *SHARP*, and *THYME*.

Part-of-Speech Tags

Words:

Punctuation:

Named Entity Tags

Named entities:

Other entities:

Dependency Labels

See the Deep Dependency Guidelines for more details:

CHAPTER 13

Structures

13.1 Document

Hello

13.2 Sentence

World

CHAPTER 14

Formats

CHAPTER 15

API References

CHAPTER 16

Media

- Chalk talk about the ELIT framework presented at AWS re:Invent 2018.
- Live interview about the ELIT project by Twitch.