

Branch: master ▾

models / research / syntaxnet /

Create new file

Upload files

Find file














History



strubell Add autograd v1.1.13 dependency to manual install

Latest commit 8f41813 on 15 Nov

..

 <a href="#">docker-devel</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">dragnn</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">examples/dragnn</a>	Move the research models into a research subfolder (#2430)	3 months ago
 <a href="#">g3doc</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">syntaxnet</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">tensorflow @ c52cdc0</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">third_party/utf</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">tools</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">util/utf8</a>	Move the research models into a research subfolder (#2430)	3 months ago
 <a href="#">.dockerignore</a>	Move the research models into a research subfolder (#2430)	3 months ago
 <a href="#">.gitignore</a>	Move the research models into a research subfolder (#2430)	3 months ago
 <a href="#">Dockerfile</a>	Release DRAGNN bulk networks (#2785)	a month ago
 <a href="#">README.md</a>	Add autograd v1.1.13 dependency to manual install	a month ago
 <a href="#">WORKSPACE</a>	Move the research models into a research subfolder (#2430)	3 months ago

 [README.md](#)

# SyntaxNet: Neural Models of Syntax.

---

*A TensorFlow toolkit for deep learning powered natural language understanding (NLU).*

**CoNLL:** See [here](#) for instructions for using the SyntaxNet/DRAGNN baseline for the CoNLL2017 Shared Task.

At Google, we spend a lot of time thinking about how computer systems can read and understand human language in order to process it in intelligent ways. We are excited to share the fruits of our research with the broader community by releasing SyntaxNet, an open-source neural network framework for [TensorFlow](#) that provides a foundation for Natural Language Understanding (NLU) systems. Our release includes all the code needed to train new SyntaxNet models on your own data, as well as a suite of models that we have trained for you, and that you can use to analyze text in over 40 languages.

This repository is largely divided into two sub-packages:

1. **DRAGNN:** [code](#), [documentation](#), [paper](#) implements Dynamic Recurrent Acyclic Graphical Neural Networks (DRAGNN), a framework for building multi-task, fully dynamically constructed computation graphs. Practically, we use DRAGNN to extend our prior work from [Andor et al. \(2016\)](#) with end-to-end, deep recurrent models and to provide a much easier to use interface to SyntaxNet. *DRAGNN is designed first and foremost as a Python library, and therefore much easier to use than the original SyntaxNet implementation.*
2. **SyntaxNet:** [code](#), [documentation](#) is a transition-based framework for natural language processing, with core functionality for feature extraction, representing annotated data, and evaluation. As of the DRAGNN release, it is recommended to train and deploy SyntaxNet models using the DRAGNN framework.

## How to use this library

---

There are three ways to use SyntaxNet:

- See [here](#) for instructions for using the SyntaxNet/DRAGNN baseline for the CoNLL2017 Shared Task, and running the ParseySaurus models.
- You can use DRAGNN to train your NLP models for other tasks and dataset. See "Getting started with DRAGNN" below.
- You can continue to use the Parsey McParseface family of pre-trained SyntaxNet models. See "Pre-trained NLP models" below.

## Installation

---

### Docker installation

*This process takes ~10 minutes.*

The simplest way to get started with DRAGNN is by loading our Docker container. [Here](#) is a tutorial for running the DRAGNN container on [GCP](#) (just as applicable to your own computer).

### Ubuntu 16.10+ binary installation

*This process takes ~5 minutes, but is only compatible with Linux using GNU libc 3.4.22 and above (e.g. Ubuntu 16.10).*

Binary wheel packages are provided for TensorFlow and SyntaxNet. If you do not need to write new binary TensorFlow ops, these should suffice.

- `apt-get install -y graphviz libgraphviz-dev libopenblas-base libpng16-16 libxft2 python-pip python-mock`
- `pip install pygraphviz --install-option="--include-path=/usr/include/graphviz" --install-option="--library-path=/usr/lib/graphviz/"`
- `pip install 'ipython<6.0' protobuf numpy scipy jupyter syntaxnet-with-tensorflow`
- `python -m jupyter_core.command nbextension enable --py --sys-prefix widgetsnbextension`

You can test that binary modules can be successfully imported by running,

- `python -c 'import dragnn.python.load_dragnn_cc_impl, syntaxnet.load_parser_ops'`

## Manual installation

*This process takes 1-2 hours.*

Running and training SyntaxNet/DRAGNN models requires building this package from source. You'll need to install:

- python 2.7:
  - Python 3 support is not available yet
- bazel 0.5.4:
  - Follow the instructions [here](#)
  - Alternately, Download bazel 0.5.4 <.deb> from <https://github.com/bazelbuild/bazel/releases> for your system configuration.
  - Install it using the command: `sudo dpkg -i <.deb file>`
  - Check for the bazel version by typing: `bazel version`
- swig:
  - `apt-get install swig` on Ubuntu
  - `brew install swig` on OSX
- protocol buffers, with a version supported by TensorFlow:
  - check your protobuf version with `pip freeze | grep protobuf`
  - upgrade to a supported version with `pip install -U protobuf==3.3.0`
- mock, the testing package:
  - `pip install mock`
- asciitree, to draw parse trees on the console for the demo:
  - `pip install asciitree`
- numpy, package for scientific computing:
  - `pip install numpy`

- autograd 1.1.13, for automatic differentiation (not yet compatible with autograd v1.2 rewrite):
  - `pip install autograd==1.1.13`
- pygraphviz to visualize traces and parse trees:
  - `apt-get install -y graphviz libgraphviz-dev`
  - `pip install pygraphviz --install-option="--include-path=/usr/include/graphviz" --install-option="--library-path=/usr/lib/graphviz/"`

Once you completed the above steps, you can build and test SyntaxNet with the following commands:

```
git clone --recursive https://github.com/tensorflow/models.git
cd models/research/syntaxnet/tensorflow
./configure
cd ..
bazel test ...
# On Mac, run the following:
bazel test --linkopt=-headerpad_max_install_names \
  dragnn/... syntaxnet/... util/utf8/...
```

Bazel should complete reporting all tests passed.

Now you can install the SyntaxNet and DRAGNN Python modules with the following commands:

```
mkdir /tmp/syntaxnet_pkg
bazel-bin/dragnn/tools/build_pip_package --output-dir=/tmp/syntaxnet_pkg
# The filename of the .whl depends on your platform.
sudo pip install /tmp/syntaxnet_pkg/syntaxnet-x.xx-none-any.whl
```

To build SyntaxNet with GPU support please refer to the instructions in [issues/248](#).

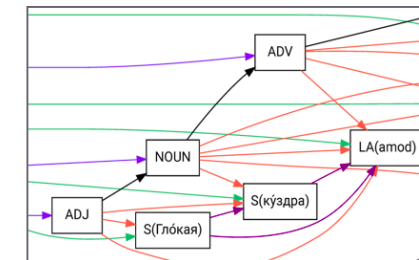
**Note:** If you are running Docker on OSX, make sure that you have enough memory allocated for your Docker VM.

# Getting Started

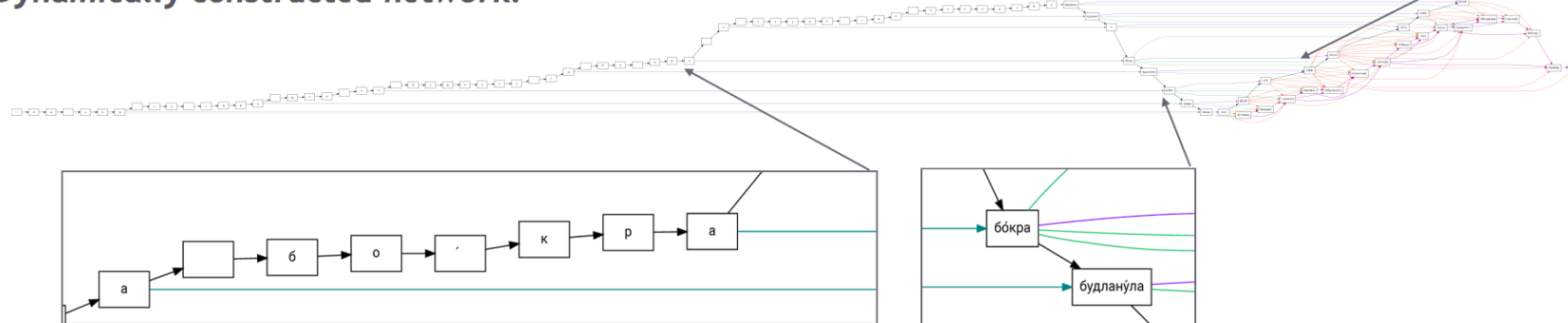
We have a few guides on this README, as well as more extensive [documentation](#).

## Learning the DRAGNN framework

### *ParseySaurus analysis:*



### *Dynamically constructed network:*



### *Character-based word representations*

An easy and visual way to get started with DRAGNN is to run our Jupyter notebooks for [interactive debugging](#) and [training a new model](#). Our tutorial [here](#) explains how to start it up from the Docker container. Once you have DRAGNN installed and running, try out the [ParseySaurus](#) models.

## Using the Pre-trained NLP models

We are happy to release *Parsey McParseface*, an English parser that we have trained for you, and that you can use to analyze English text, along with [trained models for 40 languages](#) and support for text segmentation and morphological analysis.

Once you have successfully built SyntaxNet, you can start parsing text right away with Parsey McParseface, located under `syntaxnet/models`. The easiest thing is to use or modify the included script `syntaxnet/demo.sh`, which shows a basic setup to parse English taking plain text as input.

You can also skip right away to the [detailed SyntaxNet tutorial](#).

How accurate is Parsey McParseface? For the initial release, we tried to balance a model that runs fast enough to be useful on a single machine (e.g. ~600 words/second on a modern desktop) and that is also the most accurate parser available. Here's how Parsey McParseface compares to the academic literature on several different English domains: (all numbers are % correct head assignments in the tree, or unlabelled attachment score)

Model	News	Web	Questions
<a href="#">Martins et al. (2013)</a>	93.10	88.23	94.21
<a href="#">Zhang and McDonald (2014)</a>	93.32	88.65	93.37
<a href="#">Weiss et al. (2015)</a>	93.91	89.29	94.17
<a href="#">Andor et al. (2016)*</a>	94.44	90.17	95.40
Parsey McParseface	94.15	89.08	94.77

We see that Parsey McParseface is state-of-the-art; more importantly, with SyntaxNet you can train larger networks with more hidden units and bigger beam sizes if you want to push the accuracy even further: [Andor et al. \(2016\)\\*](#) is simply a SyntaxNet model with a larger beam and network. For further information on the datasets, see that paper under the section "Treebank Union".

Parsey McParseface is also state-of-the-art for part-of-speech (POS) tagging (numbers below are per-token accuracy):

Model	News	Web	Questions
<a href="#">Ling et al. (2015)</a>	97.44	94.03	96.18
<a href="#">Andor et al. (2016)*</a>	97.77	94.80	96.86
Parsey McParseface	97.52	94.24	96.45

## Parsing from Standard Input

Simply pass one sentence per line of text into the script at `syntaxnet/demo.sh`. The script will break the text into words, run the POS tagger, run the parser, and then generate an ASCII version of the parse tree:

```
echo 'Bob brought the pizza to Alice.' | syntaxnet/demo.sh
```

```
Input: Bob brought the pizza to Alice .
```

```
Parse:
```

```
brought VBD ROOT
+-- Bob NNP nsubj
+-- pizza NN dobj
|   +-- the DT det
+-- to IN prep
|   +-- Alice NNP pobj
+-- . . punct
```

The ASCII tree shows the text organized as in the parse, not left-to-right as visualized in our tutorial graphs. In this example, we see that the verb "brought" is the root of the sentence, with the subject "Bob", the object "pizza", and the prepositional phrase "to Alice".

If you want to feed in tokenized, CONLL-formatted text, you can run `demo.sh --conll`.

## Annotating a Corpus



To change the pipeline to read and write to specific files (as opposed to piping through stdin and stdout), we have to modify the `demo.sh` to point to the files we want. The SyntaxNet models are configured via a combination of run-time flags (which are easy to change) and a text format `TaskSpec` protocol buffer. The spec file used in the demo is in `syntaxnet/models/parsey_mcparseface/context.pbtxt`.

To use corpora instead of stdin/stdout, we have to:

1. Create or modify an `input` field inside the `TaskSpec`, with the `file_pattern` specifying the location we want. If the input corpus is in CONLL format, make sure to put `record_format: 'conll-sentence'`.
2. Change the `--input` and/or `--output` flag to use the name of the resource as the output, instead of `stdin` and `stdout`.

E.g., if we wanted to POS tag the CONLL corpus `./wsj.conll`, we would create two entries, one for the input and one for the output:

```
input {
  name: 'wsj-data'
  record_format: 'conll-sentence'
  Part {
    file_pattern: './wsj.conll'
  }
}
input {
  name: 'wsj-data-tagged'
  record_format: 'conll-sentence'
  Part {
    file_pattern: './wsj-tagged.conll'
  }
}
```

Then we can use `--input=wsj-data --output=wsj-data-tagged` on the command line to specify reading and writing to these files.

## Configuring the Python Scripts

As mentioned above, the python scripts are configured in two ways:

1. **Run-time flags** are used to point to the `TaskSpec` file, switch between inputs for reading and writing, and set various run-time model parameters. At training time, these flags are used to set the learning rate, hidden layer sizes, and other key parameters.
2. The **`TaskSpec` proto** stores configuration about the transition system, the features, and a set of named static resources required by the parser. It is specified via the `--task_context` flag. A few key notes to remember:
  - The `Parameter` settings in the `TaskSpec` have a prefix: either `brain_pos` (they apply to the tagger) or `brain_parser` (they apply to the parser). The `--prefix` run-time flag switches between reading from the two configurations.
  - The resources will be created and/or modified during multiple stages of training. As described above, the resources can also be used at evaluation time to read or write to specific files. These resources are also separate from the model parameters, which are saved separately via calls to TensorFlow ops, and loaded via the `--model_path` flag.
  - Because the `TaskSpec` contains file path, remember that copying around this file is not enough to relocate a trained model: you need to move and update all the paths as well.

Note that some run-time flags need to be consistent between training and testing (e.g. the number of hidden units).

## Next Steps

There are many ways to extend this framework, e.g. adding new features, changing the model structure, training on other languages, etc. We suggest reading the detailed tutorial below to get a handle on the rest of the framework.

## Contact

To ask questions or report issues please post on Stack Overflow with the tag [syntaxnet](#) or open an issue on the tensorflow/models [issues tracker](#). Please assign SyntaxNet issues to [@calberti](#) or [@andorardo](#).

# Credits

---

Original authors of the code in this package include (in alphabetical order):

- Alessandro Presta
- Aliaksei Severyn
- Andy Golding
- Bernd Bohnet
- Chayut Thanapirom
- Chris Alberti
- Daniel Andor
- David Weiss
- Emily Pitler
- Greg Coppola
- Ivan Bogatyy
- Ji Ma
- Keith Hall
- Kuzman Ganchev
- Lingpeng Kong
- Livio Baldini Soares
- Mark Omernick
- Michael Collins
- Michael Ringgaard
- Ryan McDonald
- Slav Petrov
- Stefan Istrate

- Terry Koo
- Tim Credo
- Zora Tung