

---

# NeuroNER

build unknown

NeuroNER is a program that performs named-entity recognition (NER). Website: [neuroner.com](http://neuroner.com).

This page gives step-by-step instructions to install and use NeuroNER.

## Table of Contents

- Requirements
- Installation
- Using NeuroNER
  - Adding a new dataset
  - Using a pretrained model
  - Sharing a pretrained model
  - Using TensorBoard
- Citation

## Requirements

NeuroNER relies on Python 3, TensorFlow 1.0+, and optionally on BRAT:

- Python 3: NeuroNER does not work with Python 2.x. On Windows, it has to be Python 3.6 64-bit or later.
- TensorFlow is a library for machine learning. NeuroNER uses it for its NER engine, which is based on neural networks. Official website: <https://www.tensorflow.org>
- BRAT (optional) is a web-based annotation tool. It only needs to be installed if you wish to conveniently create annotations or view the predictions made by NeuroNER. Official website: <http://brat.nlplab.org>

## Installation

For GPU support, GPU requirements for Tensorflow must be satisfied. If your system does not meet these requirements, you should use the CPU version. To install neuroner:

```
1 # For CPU support (no GPU support):  
2 pip3 install pyneuroner[cpu]  
3
```

---

```
4 # For GPU support:
5 pip3 install pyneuroner[gpu]
```

You will also need to download some support packages.

1. The English language module for Spacy:

```
1 # Download the SpaCy English module
2 python -m spacy download en
```

2. Download word embeddings from [http://neuroner.com/data/word\\_vectors/glove.6B.100d.zip](http://neuroner.com/data/word_vectors/glove.6B.100d.zip), unzip them to the folder `./data/word_vectors`

```
1 # Get word embeddings
2 wget -P data/word_vectors http://neuroner.com/data/word_vectors/glove.6
  B.100d.zip
3 unzip data/word_vectors/glove.6B.100d.zip -d data/word_vectors/
```

3. Load sample datasets. These can be loaded by calling the `neuromodel.fetch_data()` function from a Python interpreter or with the `--fetch_data` argument at the command line.

```
1 # Load a dataset from the command line
2 neuroner --fetch_data=conll2003
3 neuroner --fetch_data=example_unannotated_texts
4 neuroner --fetch_data=i2b2_2014_deid
```

```
1 # Load a dataset from a Python interpreter
2 from neuroner import neuromodel
3 neuromodel.fetch_data('conll2003')
4 neuromodel.fetch_data('example_unannotated_texts')
5 neuromodel.fetch_data('i2b2_2014_deid')
```

4. Load a pretrained model. The models can be loaded by calling the `neuromodel.fetch_model()` function from a Python interpreter or with the `--fetch_trained_models` argument at the command line.

```
1 # Load a pre-trained model from the command line
2 neuroner --fetch_trained_model=conll_2003_en
3 neuroner --fetch_trained_model=i2b2_2014_glove_spacy_bioes
4 neuroner --fetch_trained_model=i2b2_2014_glove_stanford_bioes
5 neuroner --fetch_trained_model=mimic_glove_spacy_bioes
6 neuroner --fetch_trained_model=mimic_glove_stanford_bioes
```

```
1 # Load a pre-trained model from a Python interpreter
2 from neuroner import neuromodel
3 neuromodel.fetch_model('conll_2003_en')
```

---

```
4 neuromodel.fetch_model('i2b2_2014_glove_spacy_bioes')
5 neuromodel.fetch_model('i2b2_2014_glove_stanford_bioes')
6 neuromodel.fetch_model('mimic_glove_spacy_bioes')
7 neuromodel.fetch_model('mimic_glove_stanford_bioes')
```

### Installing BRAT (optional)

BRAT is a tool that can be used to create, change or view the BRAT-style annotations. For installation and usage instructions, see the BRAT website.

### Installing Perl (platform dependent)

Perl is required because the official CoNLL-2003 evaluation script is written in this language: <http://strawberryperl.com>. For Unix and Mac OSX systems, Perl should already be installed. For Windows systems, you may need to install it.

### Using NeuroNER

NeuroNER can either be run from the command line or from a Python interpreter.

#### Using NeuroNer from a Python interpreter

To use NeuroNER from the command line, create an instance of the neuromodel with your desired arguments, and then call the relevant methods. Additional parameters can be set from a `parameters.ini` file in the working directory. For example:

```
1 from neuroner import neuromodel
2 nn = neuromodel.NeuroNER(train_model=False, use_pretrained_model=True)
```

More detail to follow.

#### Using NeuroNer from the command line

By default NeuroNER is configured to train and test on the CoNLL-2003 dataset. Running neuroner with the default settings starts training on the CoNLL-2003 dataset (the F1-score on the test set should be around 0.90, i.e. on par with state-of-the-art systems). To start the training:

---

```
1 # To use the CPU if you have installed tensorflow, or use the GPU if
   you have installed tensorflow-gpu:
2 neuroner
3
4 # To use the CPU only if you have installed tensorflow-gpu:
5 CUDA_VISIBLE_DEVICES="" neuroner
6
7 # To use the GPU 1 only if you have installed tensorflow-gpu:
8 CUDA_VISIBLE_DEVICES=1 neuroner
```

If you wish to change any of NeuroNER parameters, you can modify the `parameters.ini` configuration file in your working directory or specify it as an argument.

For example, to reduce the number of training epochs and not use any pre-trained token embeddings:

```
1 neuroner --maximum_number_of_epochs=2 --
   token_pretrained_embedding_filepath=""
```

To perform NER on some plain texts using a pre-trained model:

```
1 neuroner --train_model=False --use_pretrained_model=True --
   dataset_text_folder=./data/example_unannotated_texts --
   pretrained_model_folder=./trained_models/conll_2003_en
```

If a parameter is specified in both the `parameters.ini` configuration file and as an argument, then the argument takes precedence (i.e., the parameter in `parameters.ini` is ignored). You may specify a different configuration file with the `--parameters_filepath` command line argument. The command line arguments have no default value except for `--parameters_filepath`, which points to `parameters.ini`.

NeuroNER has 3 modes of operation:

- training mode (from scratch): the dataset folder must have train and valid sets. Test and deployment sets are optional.
- training mode (from pretrained model): the dataset folder must have train and valid sets. Test and deployment sets are optional.
- prediction mode (using pretrained model): the dataset folder must have either a test set or a deployment set.

## Adding a new dataset

A dataset may be provided in either CoNLL-2003 or BRAT format. The dataset files and folders should be organized and named as follows:

- 
- Training set: `train.txt` file (CoNLL-2003 format) or `train` folder (BRAT format). It must contain labels.
  - Validation set: `valid.txt` file (CoNLL-2003 format) or `valid` folder (BRAT format). It must contain labels.
  - Test set: `test.txt` file (CoNLL-2003 format) or `test` folder (BRAT format). It must contain labels.
  - Deployment set: `deploy.txt` file (CoNLL-2003 format) or `deploy` folder (BRAT format). It shouldn't contain any label (if it does, labels are ignored).

We provide several examples of datasets:

- `data/conll2003/en`: annotated dataset with the CoNLL-2003 format, containing 3 files (`train.txt`, `valid.txt` and `test.txt`).
- `data/example_unannotated_texts`: unannotated dataset with the BRAT format, containing 1 folder (`deploy/`). Note that the BRAT format with no annotation is the same as plain texts.

## Using a pretrained model

In order to use a pretrained model, the `pretrained_model_folder` parameter in the `parameters.ini` configuration file must be set to the folder containing the pretrained model. The following parameters in the `parameters.ini` configuration file must also be set to the same values as in the configuration file located in the specified `pretrained_model_folder`:

```
1 use_character_lstm
2 character_embedding_dimension
3 character_lstm_hidden_state_dimension
4 token_pretrained_embedding_filepath
5 token_embedding_dimension
6 token_lstm_hidden_state_dimension
7 use_crf
8 tagging_format
9 tokenizer
```

## Sharing a pretrained model

You are highly encouraged to share a model trained on their own datasets, so that other users can use the pretrained model on other datasets. We provide the `neuroner/prepare_pretrained_model.py` script to make it easy to prepare a pretrained model for sharing. In order to use the script, one only needs to specify the `output_folder_name`, `epoch_number`, and `model_name` parameters in the script.

---

By default, the only information about the dataset contained in the pretrained model is the list of tokens that appears in the dataset used for training and the corresponding embeddings learned from the dataset.

If you wish to share a pretrained model without providing any information about the dataset (including the list of tokens appearing in the dataset), you can do so by setting

```
delete_token_mappings = True
```

when running the script. In this case, it is highly recommended to use some external pre-trained token embeddings and freeze them while training the model to obtain high performance. This can be done by specifying the `token_pretrained_embedding_filepath` and setting

```
freeze_token_embeddings = True
```

in the `parameters.ini` configuration file during training.

In order to share a pretrained model, please submit a new issue on the GitHub repository.

## Using TensorBoard

You may launch TensorBoard during or after the training phase. To do so, run in the terminal from the NeuroNER folder:

```
1 tensorboard --logdir=output
```

This starts a web server that is accessible at `http://127.0.0.1:6006` from your web browser.

## Citation

If you use NeuroNER in your publications, please cite this paper:

```
1 @article{2017neuroner,  
2   title={{NeuroNER}: an easy-to-use program for named-entity  
   recognition based on neural networks},  
3   author={Dernoncourt, Franck and Lee, Ji Young and Szolovits, Peter},  
4   journal={Conference on Empirical Methods on Natural Language  
   Processing (EMNLP)},  
5   year={2017}  
6 }
```

The neural network architecture used in NeuroNER is described in this article:

```
1 @article{2016deidentification,  
2   title={De-identification of Patient Notes with Recurrent Neural  
   Networks},
```

---

```
3   author={Dernoncourt, Franck and Lee, Ji Young and Uzuner, Ozlem and
4     Szolovits, Peter},
5   journal={Journal of the American Medical Informatics Association (
6     JAMIA)},
7   year={2016}
8 }
```