gelesen

FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP

Alan Akbik[†] **Tanja Bergmann**[†*] **Duncan Blythe**[†⋆]

Kashif Rasul[†] **Stefan Schweter**[‡] **Roland Vollgraf**[†]

sehr informatives Paper

[†]Zalando Research, Mühlenstraße 25, 10243 Berlin
[‡]Bayerische Staatsbibliothek München, Digital Library/Munich Digitization Center, 80539 Munich
[⋆]Aleph-One GmbH, Rigaer Straße 8, 12047 Berlin [*]RASA AI, Berlin

Abstract

We present FLAIR, an NLP framework designed to facilitate training and distribution of state-of-the-art sequence labeling, text classification and language models. The core idea of the framework is to present a simple, unified interface for conceptually very different types of word and document embeddings. This effectively hides all embedding-specific engineering complexity and allows researchers to "mix and match" various embeddings with little effort. The framework also implements standard model training and hyperparameter selection routines, as well as a data fetching module that can download publicly available NLP datasets and convert them into data structures for quick set up of experiments. Finally, FLAIR also ships with a "model zoo" of pre-trained models to allow researchers to use state-of-the-art NLP models in their applications. This paper gives an overview of the framework and its functionality.

The framework is available on GitHub at `https://github.com/zalandoresearch/flair`.

1 Introduction

Classic pre-trained word embeddings have been shown to be of great use for downstream NLP tasks, both due to their ability to assist learning and generalization with information learned from unlabeled data, as well as the relative ease of including them into any learning approach (Mikolov et al., 2013; Pennington et al., 2014). Many recently proposed approaches go beyond the initial "one word, one embedding" paradigm to better model additional features such as subword structures (Ma and Hovy, 2016; Bojanowski et al., 2017) and meaning ambiguity (Peters et al., 2018a). Though shown to be extremely powerful, such embeddings have the drawback that they cannot be used to simply initialize the embedding layer of a neural network and thus require specific reworkings of the overall model architecture.

Hierarchical architectures. A common example is that many current approaches combine classic word embeddings with character-level features trained on task data (Ma and Hovy, 2016; Lample et al., 2016). To accomplish this, they use a hierarchical learning architecture in which the output states of a character-level CNN or RNN are concatenated with the output of the embedding layer. While modern deep learning frameworks such as PYTORCH (Paszke et al., 2017) make the construction of such architectures relatively straightforward, architectural changes are nevertheless required for something that is fundamentally just another method for embedding words.

Contextualized embeddings. Similarly, recent works—including our own—have proposed methods that produce different embeddings for the same word depending on its contextual usage (Peters et al., 2018a; Akbik et al., 2018; Devlin et al., 2018). The string "Washington" for instance would be embedded differently depending on whether the context indicates this string to be a last name or a location. While shown to be highly powerful, especially in combination with classic word embeddings, such methods require an architecture in which the output states of a trained language model (LM) are concatenated with the output of the embedding layer, thus adding architectural complexity.

These examples illustrate that word embeddings typically cannot simply be mixed and matched with minimal effort, but rather require specific reworkings of the model architecture.

Proposed solution: FLAIR framework. With this paper, we present a new framework designed to address this problem. The principal design goal is to abstract away from specific engineering challenges that different types of word embeddings raise. We created a simple, unified interface for all word embeddings as well as arbitrary combinations of embeddings. This interface, we argue, allows researchers to build a single model architecture that can then make use of any type of word embedding with no additional engineering effort.

To further simplify the process of setting up and executing experiments, FLAIR includes convenience methods for downloading standard NLP research datasets and reading them into data structures for the framework. It also includes model training and hyperparameter selection routines to facilitate typical training and testing workflows. In addition, FLAIR also ships with a growing list of pre-trained models allowing users to apply already trained models to their text. This paper gives an overview of the framework.

2 Framework Overview

2.1 Setup

FLAIR only requires a current version of Python (at least version 3.6) to be available on a system or a virtual environment. Then, the simplest way to install the library is via `pip`, by issuing the command: `pip install flair`. This downloads the latest release of FLAIR and sets up all required libraries, such as PYTORCH.

Alternatively, users can clone or fork the current master branch of FLAIR from the GitHub repository. This allows users to work on the latest version of the code and create pull requests. The GitHub page[1] has extensive documentation on training and applying models and embedding.

2.2 Base Classes

With code readability and ease-of-use in mind, we represent NLP concepts such as tokens, sentences and corpora with simple base (non-tensor) classes that we use throughout the library. For instance, the following code instantiates an example `Sentence` object:

```
# init sentence
sentence = Sentence('I love Berlin')
```

Each `Sentence` is instantiated as a list of `Token` objects, each of which represents a word and has fields for tags (such as part-of-speech or named entity tags) and embeddings (embeddings of this word in different embedding spaces).

[1] https://github.com/zalandoresearch/flair

2.3 Embeddings

Embeddings are the core concept of FLAIR. Each embedding class implements either the `TokenEmbedding` or the `DocumentEmbedding` interface for word and document embeddings respectively. Both interfaces define the `.embed()` method to embed a `Sentence` or a list of `Sentence` objects into a specific embedding space.

2.3.1 Classic Word Embeddings

The simplest examples are classic word embeddings, such as GLOVE or FASTTEXT. Simply instantiate one of the supported word embeddings and call `.embed()` to embed a sentence:

```
# init GloVe embeddings
glove = WordEmbeddings('glove')

# embed sentence
glove.embed(sentence)
```

Here, the framework checks if the requested GLOVE embeddings are already available on local disk. If not, the embeddings are first downloaded. Then, GLOVE embeddings are added to each `Token` in the `Sentence`.

Note that all logic is handled by the embedding class, i.e. it is not necessary to run common preprocessing steps such as constructing a vocabulary of words in the dataset or encoding words as one-hot vectors. Rather, each embedding is immediately applicable to any text wrapped in a `Sentence` object.

2.3.2 Other Word Embeddings

As noted in the introduction, FLAIR supports a growing list of embeddings such as hierarchical character features (Lample et al., 2016), ELMo embeddings (Peters et al., 2018a), ELMo transformer embeddings (Peters et al., 2018b), BERT embeddings (Devlin et al., 2018), byte pair embeddings (Heinzerling and Strube, 2018), Flair embeddings (Akbik et al., 2018) and *Pooled Flair* embeddings. See Table 1 for an overview.

Importantly, all embeddings implement the same interface and may be called and applied just like in the `WordEmbedding` example above. For instance, to use BERT embeddings to embed a sentence, simply call:

```
# init BERT embeddings
bert = BertEmbeddings()

# embed sentence
bert.embed(sentence)
```

Class	Type	Pretrained?
`WordEmbeddings`	classic word embeddings (Pennington et al., 2014)	yes
`CharacterEmbeddings`	character features (Lample et al., 2016)	no
`BytePairEmbeddings`	byte-pair embeddings (Heinzerling and Strube, 2018)	yes
`FlairEmbeddings`	character-level LM embeddings (Akbik et al., 2018)	yes
`PooledFlairEmbeddings`	pooled version of FLAIR embeddings (Akbik et al., 2019b)	yes
`ELMoEmbeddings`	word-level LM embeddings (Peters et al., 2018a)	yes
`ELMoTransformerEmbeddings`	word-level transformer LM embeddings (Peters et al., 2018b)	yes
`BertEmbeddings`	byte-pair masked LM embeddings (Devlin et al., 2018)	yes
`DocumentPoolEmbeddings`	document embeddings from pooled word embeddings (Joulin et al., 2017)	yes
`DocumentLSTMEmbeddings`	document embeddings from LSTM over word embeddings	no

Table 1: Summary of word and document embeddings currently supported by FLAIR. Note that some embedding types are not pre-trained; these embeddings are automatically trained or fine-tuned when training a model for a downstream task.

2.3.3 Stacked Embeddings

In many cases, we wish to mix and match several different types of embeddings. For instance, Lample et al. (2016) combine classic word embeddings with character features. To achieve this in FLAIR, we need to combine the embedding classes `WordEmbeddings` and `CharacterEmbeddings`. To enable such combinations, e.g. the “stacking” of embeddings, we include the `StackedEmbeddings` class. It is instantiated by passing a list of embeddings to stack, but then behaves like any other embedding class. This means that by calling the `.embed()` method, a `StackedEmbeddings` class instance embeds a sentence like any other embedding class instance.

Our recommended setup is to stack `WordEmbeddings` with `FlairEmbeddings`, which gives state-of-the-art accuracies across many sequence labeling tasks. See Akbik et al. (2018) for a comparative evaluation.

2.3.4 Document Embeddings

FLAIR also supports methods for producing vector representations not of words, but of entire documents. There are two main embedding classes for this, namely `DocumentPoolEmbeddings` and `DocumentLSTMEmbeddings`. The former applies a pooling operation, such as *mean pooling*, to all word embeddings in a document to derive a document representation. The latter applies an LSTM over the word embeddings in a document to output a document representation.

2.4 NLP Dataset Downloader

To facilitate setting up experiments, we include convenience methods to download publicly available benchmark datasets for a variety of NLP tasks and read them into data structures for training. For instance, to download the universal dependency treebank for English, simply execute these lines:

```
# define dataset
task = NLPTask.UD_English

# load dataset
corpus = NLPTaskDataFetcher.load_corpus(
    task)
```

Internally, the data fetcher checks if the requested dataset is already present on local disk and if not, downloads it. The dataset is then read into an object of type `TaggedCorpus` which defines training, testing and development splits.

Table 2 gives an overview of all datasets that are currently downloadable. Other datasets, such as the CoNLL-03 datasets for English and German, require licences and thus cannot be automatically downloaded.

Dataset	Task	Language(s)
CoNLL 2000	NP Chunking	en
CoNLL 2003	NER	dt, es
EIEC	NER	basque
IMDB	Classification	en
TREC-6	Classification	en
TREC-50	Classification	en
Universal Dependencies	PoS, Parsing	30 languages
WikiNER	NER	9 languages
WNUT-17	NER	en

Table 2: Summary of NLP datasets in the downloader. References: CoNLL 2000 (Sang and Buchholz, 2000), CoNLL 2003 (Sang and De Meulder, 2003), EIEC (Alegria et al.), IMDB (Maas et al., 2011), TREC-6 (Voorhees and Harman, 2000), TREC-50 (Li and Roth, 2002), Universal Dependencies (Zeman et al., 2018), WikiNER (Nothman et al., 2012) and WNUT-17 (Derczynski et al., 2017).

2.5 Model Training

To train a downstream task model, FLAIR includes the `ModelTrainer` class which implements a host of mechanisms that are typically applied during training. This includes features such as mini-batching, model checkpointing, learning rate annealing schedulers, evaluation methods and logging. This unified training interface is designed to

Task	Dataset	Language(s)	Variant(s)
4-class NER	CoNLL 2003 (Sang and De Meulder, 2003)	en, de, nl, es	default, fast, multilingual
4-class NER	WikiNER (Nothman et al., 2012)	fr	default
12-class NER	Ontonotes (Hovy et al., 2006)	en	default, fast
NP Chunking	CoNLL 2000 (Sang and Buchholz, 2000)	en	default, fast
Offensive Language Detection	GermEval 2018 (Wiegand et al., 2018)	de	default
PoS tagging	Ontonotes (Hovy et al., 2006)	en	default, fast
Semantic Frame Detection	PropBank (Bonial et al., 2014)	en	default, fast
Sentiment Analysis	IMDB (Maas et al., 2011)	en	default
Universal PoS	Universal Dependencies (Zeman et al., 2018)	12 languages	multilingual

Table 3: Summary of pre-trained sequence labeling and text classification models currently available. The "default" variant are single-language models optimized for GPU-systems. The "fast" variant are smaller models optimized for CPU-systems. The "multilingual" variants are single models that can label text in different languages.

facilitate experimentation with standard learning parameters.

The `ModelTrainer` can be applied to any FLAIR model that implements the `flair.nn.Model` interface, such as our sequence tagging and text classification classes. Refer to the online tutorials for examples on how to train different types of downstream task models.

2.6 Hyperparameter Selection

To further facilitate training models, FLAIR includes native support for the HYPEROPT library which implements a Tree of Parzen Estimators (TPE) approach to hyperparameter optimization (Bergstra et al., 2013). Hyperparameter selection is performed against the development data split by default. This allows users to first run hyperparameter selection using the training and development data splits, and then evaluate the final parameters with the held-out testing data.

3 Model Zoo

In addition to providing a framework for embedding text and training models, FLAIR also includes a model zoo of pre-trained sequence labeling, text classification and language models. They allow users to apply pre-trained models to their own text, or to fine-tune them for their use cases. A list of currently shipped models is provided in Table 3.

For example, to load and apply the default named entity recognizer for English, simply execute the following lines of code:

```
# make a sentence
sentence = Sentence('I love Berlin .')

# load the NER tagger
tagger = SequenceTagger.load('ner')

# run NER over sentence
tagger.predict(sentence)
```

This first checks if the corresponding model is already available on local disk and if not, downloads it. Entity tags are then added to the `Token` objects in the `Sentence`. In this specific example, this will mark up "Berlin" as an entity of type *location*.

3.1 Model Variants

We distribute different variants of models with FLAIR (see Table 3). The *default* variant are single-language models intended to be run on GPU, typically using embeddings from language models with 2048 hidden states. The *fast* variant models use computationally less demanding embeddings, typically from LMs with 1024 hidden states, and are suited to be run on CPU setups.

We also include multilingual models for some tasks. These are "one model, many languages" models that can predict tags for text in multiple languages. For instance, Flair includes multilingual part-of-speech tagging models that predict universal PoS tags for text in 12 languages. See Akbik et al. (2019a) for an overview of multilingual models and preliminary evaluation numbers.

4 Conclusion and Outlook

We presented FLAIR as a framework designed to facilitate experimentation with different embedding types, as well as training and distributing sequence labeling and text classification models.

Together with the open source community, we are working to extend the framework along multiple directions. This includes supporting *more embedding approaches* such as transformer embeddings (Radford et al., 2018; Dai et al., 2019), InferSent representations (Conneau et al., 2017) and LASER embeddings (Artetxe and Schwenk, 2018), and expanding our coverage of *NLP datasets* and formats for automatic data fetching.

Current research also focuses on developing new embedding types, investigating further downstream tasks and extending the framework to facilitate multi-task learning approaches.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732328 ("FashionBrain").

The development of FLAIR has benefited enormously from the open source community. We wish to thank the many individuals who contributed (see https://github.com/zalandoresearch/flair/graphs/contributors) since without a doubt the usability and features of FLAIR have only gotten better due to their work or feedback. Thank you also to Zalando's Open Source team (see https://opensource.zalando.com).

References

Alan Akbik, Tanja Bergmann, and Roland Vollgraf. 2019a. Multilingual sequence labeling with one model. In *NLDL 2019, Northern Lights Deep Learning Workshop*.

Alan Akbik, Tanja Bergmann, and Roland Vollgraf. 2019b. Pooled contextualized embeddings for named entity recognition. In *NAACL, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, page to appear.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Inaki Alegria, Olatz Arregi, Irene Balza, Nerea Ezeiza, Izaskun Fernandez, and Ruben Urizar. Design and development of a named entity recognizer for an agglutinative language.

Mikel Artetxe and Holger Schwenk. 2018. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *arXiv preprint arXiv:1812.10464*.

James Bergstra, Daniel Yamins, and David Daniel Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Claire Bonial, Julia Bonn, Kathryn Conger, Jena D. Hwang, and Martha Palmer. 2014. Propbank: Semantics of new predicate types. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA).

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860.

Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. 2017. Results of the wnut2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Benjamin Heinzerling and Michael Strube. 2018. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 427–431.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTMs-CNNs-CRF. *arXiv preprint arXiv:1603.01354*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R. Curran. 2012. Learning multilingual named entity recognition from Wikipedia. *Artificial Intelligence*, 194:151–175.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Mark Neumann, and Christopher Clark Kenton Lee Luke Zettlemoyer Mohit Iyyer, Matt Gardner. 2018a. Deep contextualized word representations. *6th International Conference on Learning Representations*.

Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018b. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509.

Alec Radford, Karthik Narasimhan, Time Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning. Technical report, Technical report, OpenAI.

Erik F Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 127–132. Association for Computational Linguistics.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics.

Ellen M Voorhees and Donna Harman. 2000. Overview of the sixth text retrieval conference (trec-6). *Information Processing & Management*, 36(1):3–35.

Michael Wiegand, Melanie Siegel, and Josef Ruppenhofer. 2018. Overview of the germeval 2018 shared task on the identification of offensive language. *Austrian Academy of Sciences, Vienna September 21, 2018*.

Daniel Zeman, Jan Haji, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. Conll 2018 shared task: Multilingual parsing from raw text to universal dependencies. *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21.