Contextual String Embeddings and FLAIR

Ercong Nie

Neural Representation Learning Seminar, CIS, LMU

SS 2021

May 21, 2021

Outline

What are Flair embeddings[1]?

Flair embeddings are:

- **contextual string embeddings**.
- produced from **character-level language model**, which is trained by predicting the next character on the basis of previous characters.

Properties of Flair embeddings:

- model words as a sequence of characters without knowing any explicit notion of words.
- are contextualized by their surrounding text.
- the **same** word will have **different** embeddings because of different context use.

[1] Akbik et al. (2018)

What is FLAIR?

FLAIR[2] is:

- a simple NLP framework based on PyTorch and Python.
- originated from Flair embeddings.
- developed by Humbolt University of Berlin and its partner companies.
- designed to facilitate training SOTA language models to solve NLP tasks, such as NER, POS tagging etc.
- **an embedding library:** It presents a unified interface allowing researchers to combine various types of word and document embeddings.

[2] https://github.com/flairNLP/flair

Embedding types

1. **Classical word embeddings**:
   pre-trained over very large corpora and shown to capture latent syntactic and semantic similarities, such as **Skip-Gram model**[3],**GloVe**[4] etc.

[3] Mikolov et al. (2013)
[4] Pennington et al. (2014)

Embedding types (cont.)

2 **Character-level features**:
not pre-trained, but trained on task data to capture task-specific subword features, such tasks as NER[5], POS tagging[6] etc.

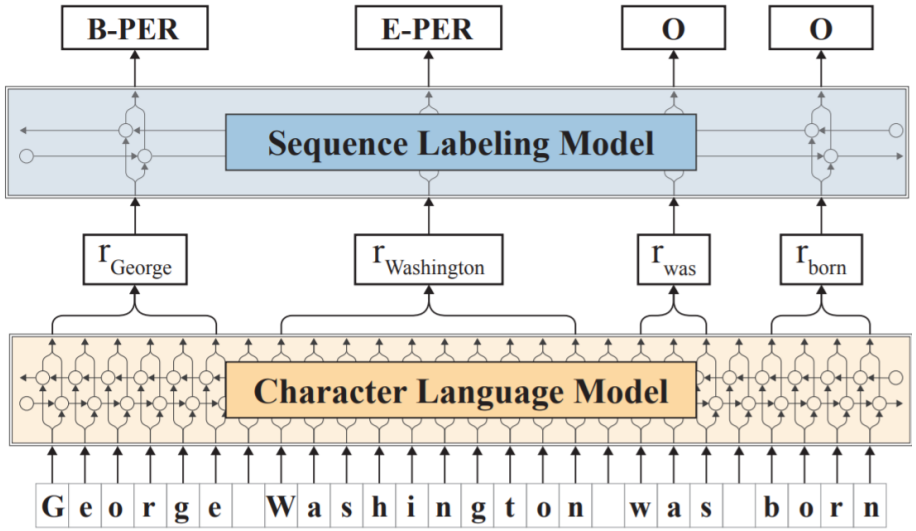
3 **Contextualized word embeddings**: capture word semantics in context[7].

Note:
Flair embeddings are hypothesized to combine all good attributes of above embeddings.

[5] Lample et al. (2016)
[6] Ma and Hovy (2016)
[7] Peters et al. (2018)

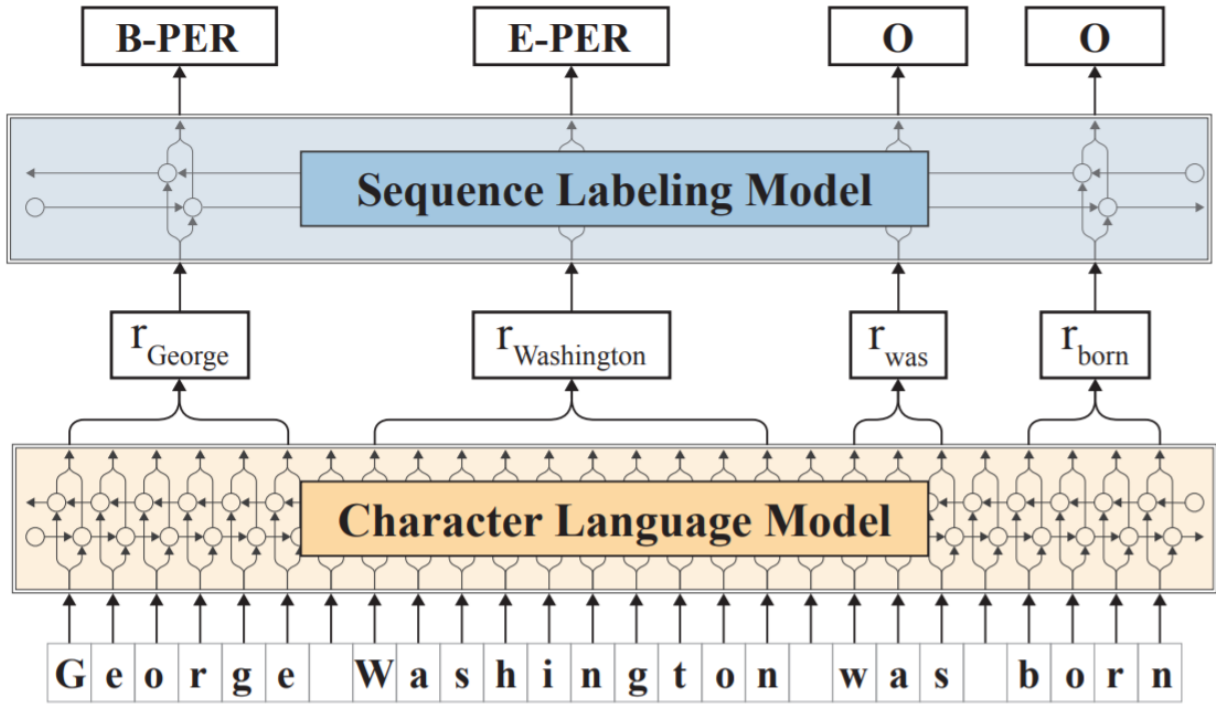
Overview of model

Flair embedding model aims to address sequence labeling tasks, thus can be divided into two parts:



1. **Character Language Model** (a bidirectional character-level neural language model):
   - **Input**: pass sentences as sequence of characters into model
   - **Output**: form word-level embeddings

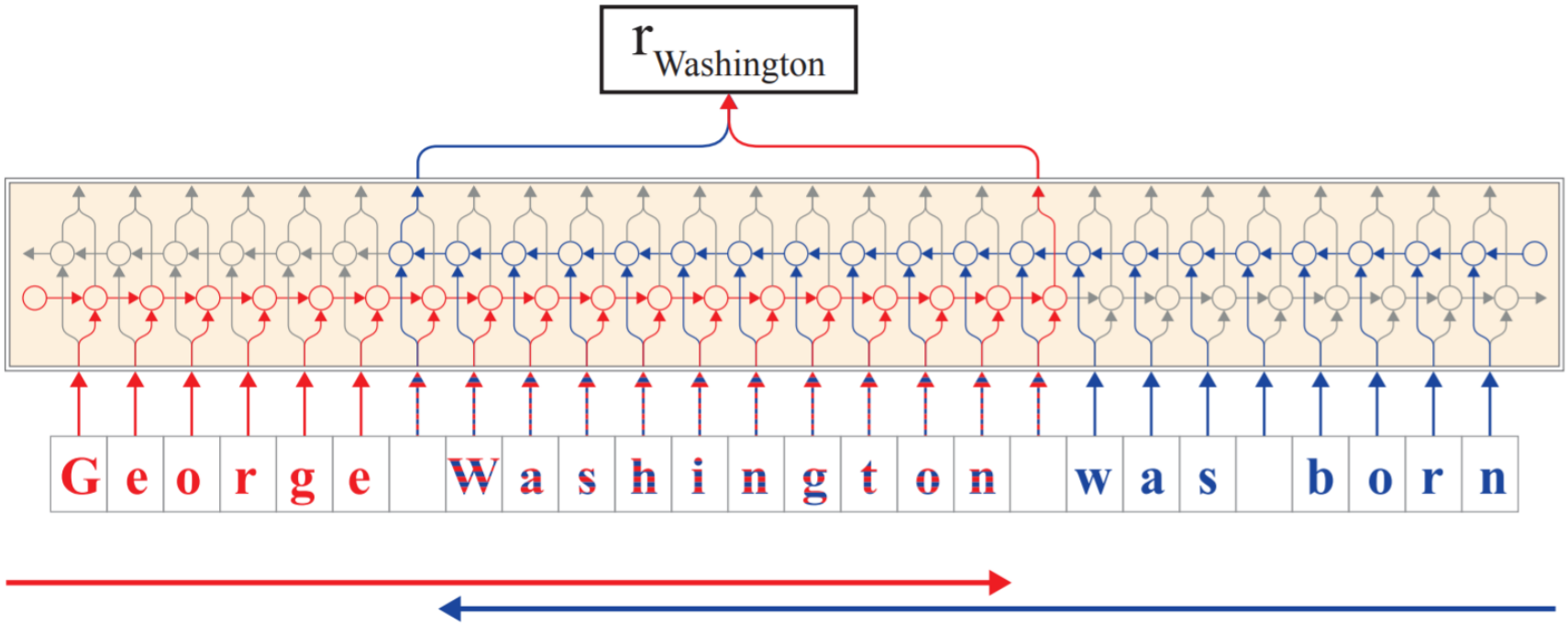
Overview of model (cont.)



2. **Sequence Labeling Model** (BiLSTM-CRF sequence tagging model):
   - **Input**: contextual string embeddings produced by the 1. model
   - **Output**: sequence tags, like names in NER

Character Language Model

basically a forward-backward (bidirectional) LSTM architecture.



Training of LM[8]

- **Goal**: estimate a good distribution $P(\mathbf{x}_{0:T})$ over sequences of characters.

$$P(\mathbf{x}_{0:T}) = \prod_{t=0}^{T} P(\mathbf{x}_t|\mathbf{x}_{0:t-1}) \tag{1}$$

- In LSTM, $P(\mathbf{x}_t|\mathbf{x}_{0:t-1})$ can be approximately represented with the network output $\mathbf{h}_t$:

$$P(\mathbf{x}_t|\mathbf{x}_{0:t-1}) \approx P(\mathbf{x}_t|\mathbf{h}_t;\theta) \tag{2}$$

[8] Graves (2013)

Training of LM (cont.)

- $\mathbf{h}_t$ can be **recursively** computed with the help of memory cell $\mathbf{c}_t$:

$$\mathbf{h}_t(\mathbf{x}_{0:t-1}) = f_{\mathbf{h}}(\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}; \theta) \tag{3}$$

$$\mathbf{c}_t(\mathbf{x}_{0:t-1}) = f_{\mathbf{c}}(\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}; \theta) \tag{4}$$

- This model uses a softmax layer to compute the likelihood of every character:

$$P(\mathbf{x}_t|\mathbf{h}_t; \mathbf{V}) = softmax(\mathbf{V}\mathbf{h}_t + \mathbf{b}) \tag{5}$$

where $\mathbf{V}$ and $\mathbf{b}$, weights and biases, are part of model parameters θ

Extracting word representations (1)

- Alongside the previous forward model, we also need a backward model to create contextualized word embeddings.
- The backward model works in the same way but in the **reversed** direction:

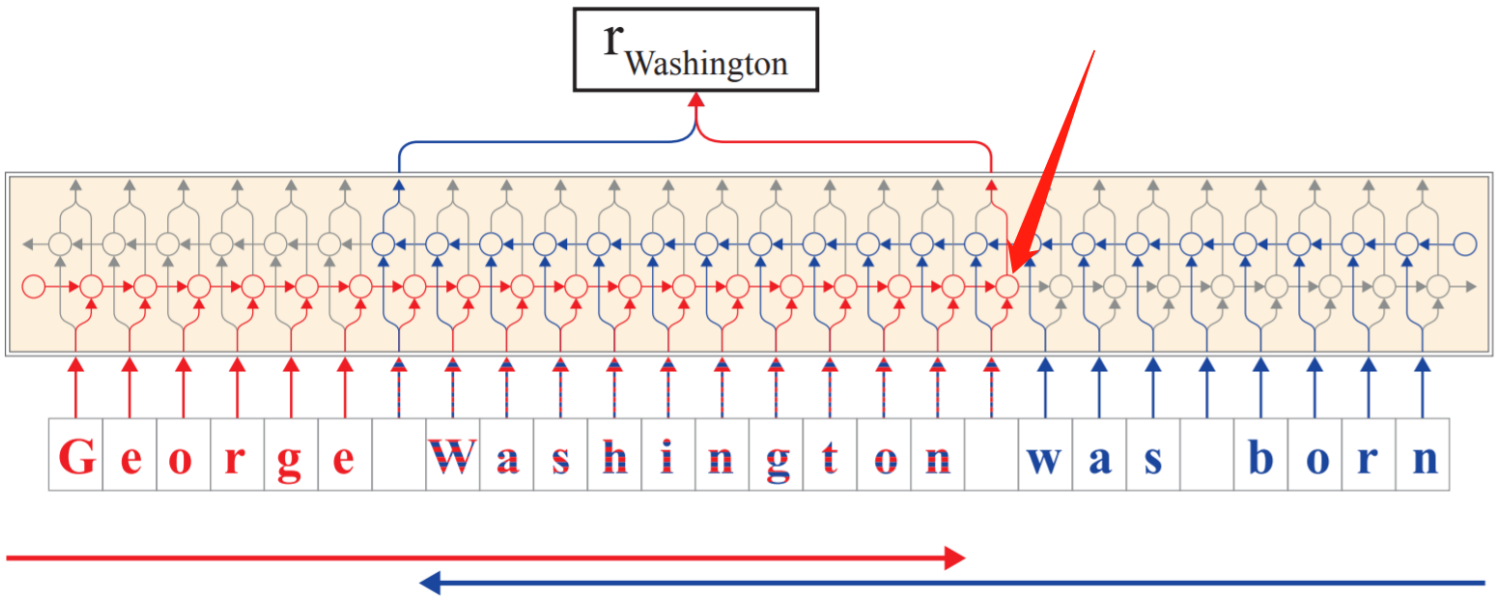
$$P^b(\mathbf{x}_t|\mathbf{x}_{t+1:T}) \approx P(\mathbf{x}_t|\mathbf{h}_t;\theta) \tag{6}$$

$$\mathbf{h}_t^b = f_{\mathbf{h}}^b(\mathbf{x}_{t+1}, \mathbf{h}_{t+1}, \mathbf{c}_{t+1};\theta) \tag{7}$$

$$\mathbf{c}_t^b = f_{\mathbf{c}}^b(\mathbf{x}_{t+1}, \mathbf{h}_{t+1}, \mathbf{c}_{t+1};\theta) \tag{8}$$

Extracting word representations (2)

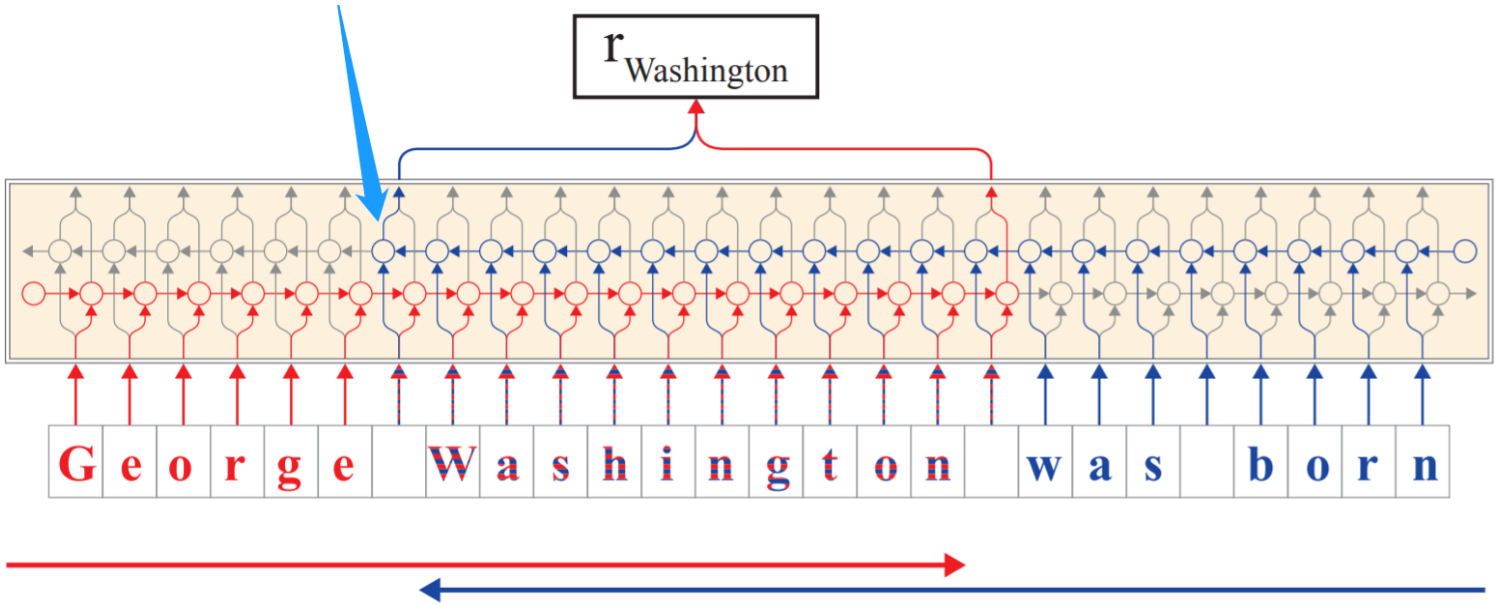
1 Extracting information from fLM



From the fLM, we extract the output hidden state after the word's last character in the word.

Extracting word representations (3)

2 Extracting information from bLM



From the bLM, we extract the output hidden state before the word's first character in the word.

Extracting word representation (4)

- Concatenating both hidden states from fLM and bLM to form the final **contextual string embedding**.

$$\mathbf{w}_i^{CharLM} := \begin{bmatrix} \mathbf{h}^f_{t_{i+1}-1} \\ \mathbf{h}^b_{t_i-1} \end{bmatrix} \tag{9}$$

- The contextual string embedding captures the semantic-syntactic information of the **word itself** and **its surrounding context**.

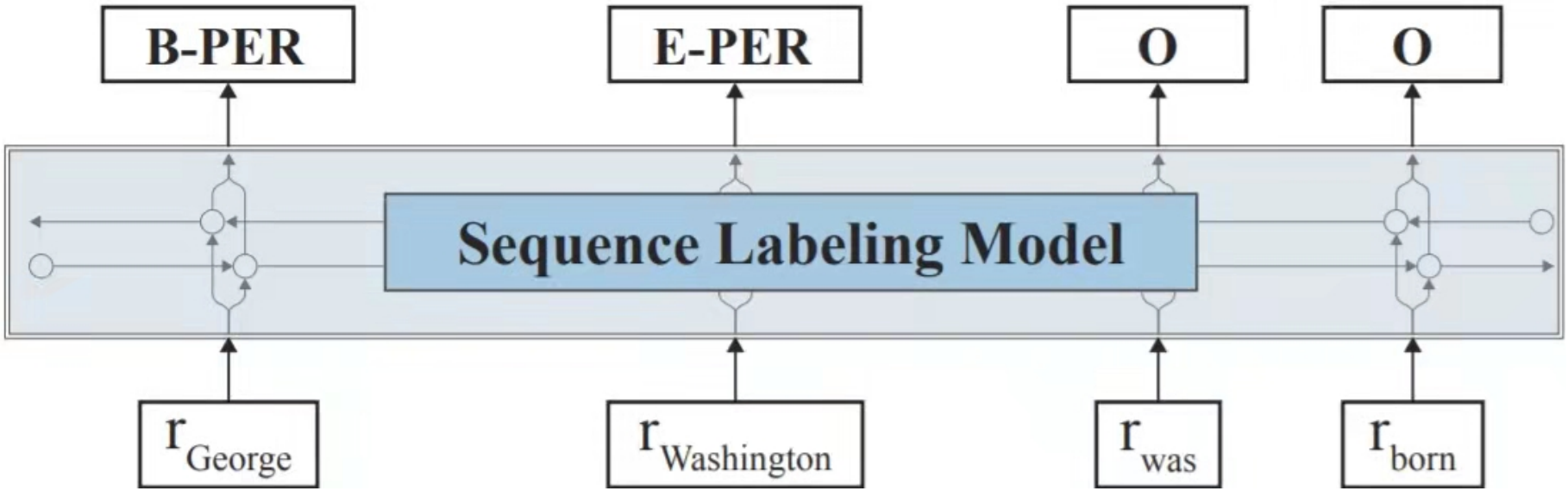
Examples

Examples of the word "Washington" in different contexts and its nearest neighbors using cosine distance over Flair embeddings.

word	context	selected nearest neighbors
Washington	(a) ***Washington*** *to curb support for [..]*	(1) ***Washington*** *would also take [..] action [..]* (2) ***Russia*** *to clamp down on barter deals [..]* (3) ***Brazil*** *to use hovercrafts for [..]*
Washington	(b) *[..] Anthony* ***Washington*** *(U.S.) [..]*	(1) *[..] Carla* ***Sacramento*** *(Portugal) [..]* (2) *[..] Charles* ***Austin*** *(U.S.) [..]* (3) *[..] Steve* ***Backley*** *(Britain) [..]*
Washington	(c) *[..] flown to* ***Washington*** *for [..]*	(1) *[..] while visiting* ***Washington*** *to [..]* (2) *[..] journey to New York City and* ***Washington*** *[..]* (14) *[..] lives in* ***Chicago*** *[..]*
Washington	(d) *[..] when* ***Washington*** *came charging back [..]*	(1) *[..] point for victory when* ***Washington*** *found [..]* (4) *[..] before* ***England*** *struck back with [..]* (6) *[..] before* ***Ethiopia*** *won the spot kick decider [..]*
Washington	(e) *[..] said* ***Washington*** *[..]*	(1) *[..] subdue the never-say-die* ***Washington*** *[..]* (4) *[..] a private school in* ***Washington*** *[..]* (9) *[..] said* ***Florida*** *manager John Boles [..]*

Sequence Labeling Model

The final word embeddings produced by the previous Character Language Model are passed into a **BiLSTM-CRF** sequence labeling model[9] to address downstream sequence labeling tasks.



[9] Huang et al. (2015)

BiLSTM

- **Inputs**:
  Contextual string embeddings $\mathbf{w}_0, ..., \mathbf{w}_n$
- **Outputs**:
  Forward outputs $\mathbf{r}_i^f$ and backward outputs $\mathbf{r}_i^b$

$$\mathbf{r}_i := \begin{bmatrix} \mathbf{r}_i^f \\ \mathbf{r}_i^b \end{bmatrix} \tag{10}$$

CRF

With $\mathbf{r}_i$ as inputs, CRF gives the final sequence probability:

$$\hat{P}(\mathbf{y}_{0:n}|\mathbf{r}_{0:n}) \propto \prod_{i=1}^{n} \psi_i(y_{i-1}, y_i, \mathbf{r}_i) \tag{11}$$

Where:

$$\psi_i(y', y, \mathbf{r}) = exp(\mathbf{W}_{y',y}\mathbf{r} + \mathbf{b}_{y',y}) \tag{12}$$

Summary (1)

Advantages of Flair Embeddings

- Character-level LM is independent of tokenization and a fixed vocabulary.
- Produce stronger character-level features, which is useful for downstreaming sequential labeling task like NER.
- Have a much smaller vocabulary size:

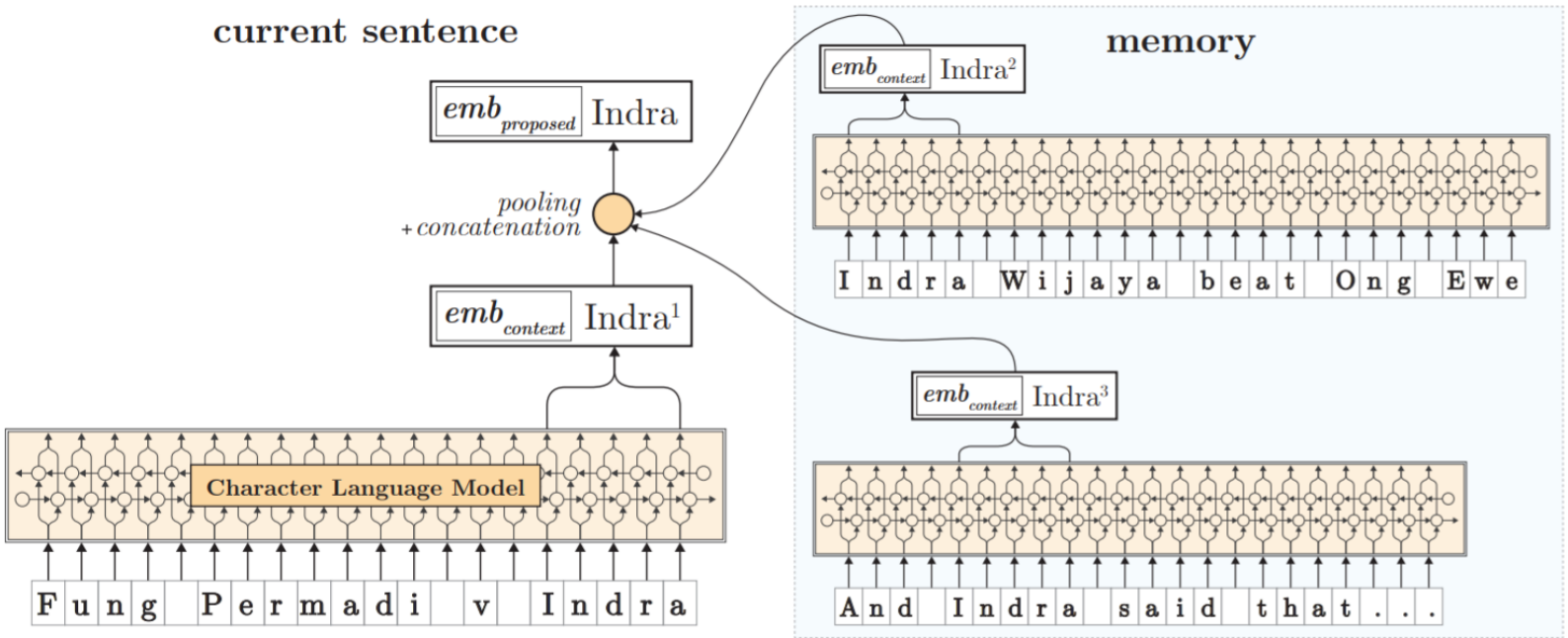
  #characters vs. #words

  thus significantly easier to train and deploy in application.

Summary (2)

Limitations of Flair Embeddings

- Struggle to produce meaningful embeddings if a rare string is used in an underspecified context.
- Solution: **Pooled Flair Embeddings**[10]



[10] Akbik et al. (2019b)

Overview of experiments

Goal

- Assess in how far contextual string embeddings are useful for sequence labeling.
- Compare contextual string embeddings with other types of embeddings.

Tasks

- Shallow semantic tasks
  - English NER in the CoNLL03 setup
  - German NER in the CoNLL03 setup
- Shallow syntactic tasks
  - Chunking in thr CoNLL2000 setup
  - POS tagging in the default Penn treebank setup

Experimental setup

Utilize the BiLSTM-CRF sequence labeling architecture[11] in all configurations.

Baselines

- **HUANG**[12]: Using pretrained word embeddings.
- **LAMPLE**[13]: Using pretrained word embeddings, in which task-trained character features are additionally computed for each word.
- **PETERS**[14]: Using contextualized word embeddings

[11] Huang et al. (2015)
[12] Huang et al. (2015)
[13] Lample et al. (2016)
[14] Peters et al. (2018)

Experimental setup (cont.)

Stacking embeddings:
It can be beneficial to add classic word embeddings to Flair embeddings to add more latent word-level semantics.

$$\mathbf{w}_i = \begin{bmatrix} \mathbf{w}_i^{Flair} \\ \mathbf{w}_i^{GloVe} \end{bmatrix}$$

$\mathbf{w}_i^{GloVe}$ is a precomputed GloVe embedding[15].

[15] Pennington et al. (2014)

Experinmental setup (cont.)

Proposed approaches

- **PROPOSED**: Simple Flair embeddings.
- **PROPOSED**$_{\textbf{+WORD}}$: Concatenating pre-trained static word embeddings with Flair embeddings.
- **PROPOSED**$_{\textbf{+CHAR}}$: Concatenating task-trained character features with Flair embeddings.
- **PROPOSED**$_{\textbf{+WORD+CHAR}}$: Adding both.
- **PROPOSED**$_{\textbf{+ALL}}$: Concatenating embeddings in all baselines.

Results

Approach	NER-English F1-score	NER-German F1-score	Chunking F1-score	POS Accuracy
proposed				
PROPOSED	91.97±0.04	85.78 ± 0.18	96.68±0.03	97.73±0.02
PROPOSED$_{+WORD}$	93.07±0.10	88.20 ± 0.21	96.70±0.04	97.82±0.02
PROPOSED$_{+CHAR}$	91.92±0.03	85.88 ± 0.20	**96.72**±0.05	97.8±0.01
PROPOSED$_{+WORD+CHAR}$	**93.09**±0.12	**88.32** ± 0.20	96.71±0.07	97.76±0.01
PROPOSED$_{+ALL}$	92.72±0.09	n/a	96.65±0.05	**97.85**±0.01
baselines				
HUANG	88.54±0.08	82.32 ± 0.35	95.4±0.08	96.94±0.02
LAMPLE	89.3±0.23	83.78 ± 0.39	95.34±0.06	97.02±0.03
PETERS	92.34±0.09	n/a	96.69±0.05	97.81± 0.02

- New state-of-the-art for semantic tasks.
- Good performance on syntactic tasks.
- Traditional word embeddings helpful.
- Task-specific character features unneccessary.

Framework overview[16]

- **Motivation**: Simply mix and match different types of word embeddings with minimal effort.
- **Toolkits**:
  - A unified interface for all word embeddings as well as arbitrary combination of them.
  - Integrating standard NLP datasets to data structure for the FLAIR framework.
  - Including model training and hyperparameter selection routines for typical training and testing workflows.

[16] Akbik et al. (2019a)

Implementation (1)

- **Base class**

```
from flair.data import Sentence

# init sentence
sentence = Sentence('I live in Munich')
```

  - Each *Sentence* is instantiated as a list of *Token* objects
  - Each *Token* represents a word and has fields for tags and embeddings

Implementation (2)

- **Word embeddings**

```
from flair.embeddings import WordEmbeddings

# init GloVe
glove = WordEmbeddings('glove')

# embed sentence
glove.embed(sentence)

# check out the embedded tokens
for token in sentence:
    print(token.embedding)
```

- Instantiate a word embedding.
- Then call *.embed()* to embed a sentence.

Implementation (3)

- **Stacking embeddings**

```
from flair.embeddings import FlairEmbeddings,
                             StackedEmbeddings

# init flair forward and backward embeddings
flair_forward = FlairEmbeddings('news-foward')
flair_backward = FlairEmbeddings('news-
                             backward')

# create a StackedEmbeddings object combining
# glove and flair forward/backward embeddings
stacked = StackedEmbeddings([glove,
                             flair_forwaed,
                             flair_backward])
```

Summary of embeddings

Summary of word and document embeddings supported by FLAIR

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

References I

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019a). Flair: An easy-to-use framework for state-of-the-art nlp. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59.

Akbik, A., Bergmann, T., and Vollgraf, R. (2019b). Pooled contextualized embeddings for named entity recognition. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, page 724–728.

References II

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *Proceedings of the 27th international conference on computational linguistics*, pages 1638–1649.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging. arxiv 2015. *arXiv preprint arXiv:1508.01991*.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

References III

Ma, X. and Hovy, E. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

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

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.