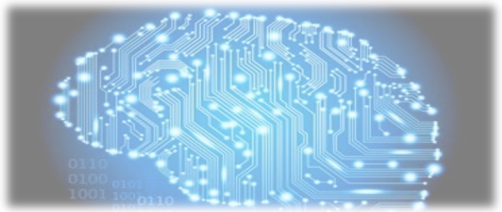
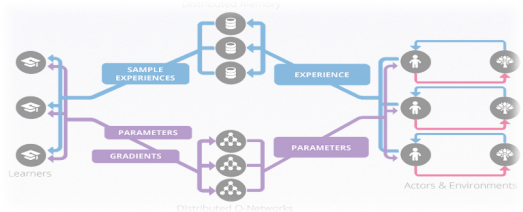



FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP

20151322 이한용

1 소개

- 고전적인 단어 임베딩은 다운스트림 NLP작업에 사용하기 좋지만, 최근 들어서 이를 넘어선 방법들이 많이 제안되고 있다.
  - Hierarchical architecture
    - CNN 또는 RNN을 이용하여 문자 수준의 특징을 추출하여 단어 임베딩에 추가한다.
    - (Ma and Hovy, 2016; Lample et al., 2016)
  - Contextualized embedding
    - 문맥상의 사용에 따라 같은 단어라도 다른 임베딩 벡터를 사용한다.
    - (Peters et al., 2018a; Akbik et al., 2018; Devlin et al., 2018)

- 다양한 유형의 임베딩을 지원하기 위한 프레임 워크가 필요하다. 이 문제를 해결하기 위해 Flair 를 제안한다.

2 Framework Overview

2.1 Setup

- Flair는 Python 3.6 이상과 PyTorch 라이브러리를 필요로 함

2.2 Base Classes

- Sentence object는 Token object의 list를 가지고 있고, 각 Token object는 tag와 embedding을 가지고 있음

2 Framework Overview

2.3 Embeddings

2.3.1 Classic Word Embeddings

- GloVe or FastText 등등을 포함

2.3.2 Other Word Embeddings

- 아래 표와 같은 임베딩들을 포함

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

2 Framework Overview

2.3 Embeddings

2.3.3 Stacked Embeddings

- 다른 종류의 embedding을 섞어 사용하기 위해 Flair에서는 StackedEmbeddings을 제공한다.
- 가장 성능이 좋은 권장설정은 WordEmbeddings과 FlairEmbeddings를 같이 사용하는 방법이다.

2.3.4 Document Embeddings

- Flair에서는 단어뿐 아니라 전체 document에 대한 임베딩 또한 제공한다. (DocumentPoolEmbeddings / DocumentLSTMEmbeddings)

2 Framework Overview

2.4 NLP Dataset Downloader

- Flair에서는 다음 표와 같은 데이터셋의 다운로드를 지원한다.

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

2 Framework Overview

2.5 Model Training

- 모델 학습에는 ModelTrainer class가 사용된다.
- flair.nn.Model를 상속받은 모델이라면 학습이 가능하다
- minibatching, model checkpointing, learning rate annealing schedulers, evaluation methods, logging 등의 기능이 제공된다.

2.6 Hyperparameter Selection

- Hyperparameter selection 기능에는 Tree of Parzen Estimators 방법을 구현한 Hyperopt 라이브러리를 제공한다.

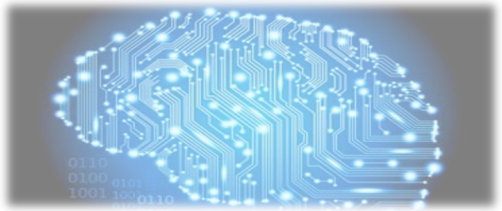
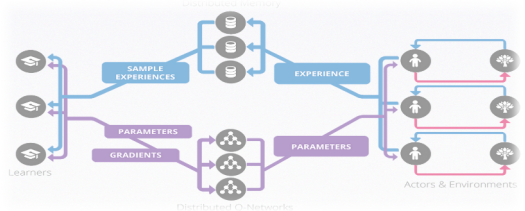
3 Model Zoo

- 미리 학습된 모델들 또한 Flair에서 제공한다.

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

3.1 Model Variants

- 각 모델에 대해 variant가 존재한다.
  - default: hidden state가 2048개인 임베딩을 사용. GPU 상에서 실행되도록 설계한 모델
  - fast: hidden state가 1024개인 임베딩을 사용. CPU 상에서 실행되도록 설계한 모델

Flair embedding
(논문: Contextual String Embeddings for Sequence Labeling)

Flair embedding

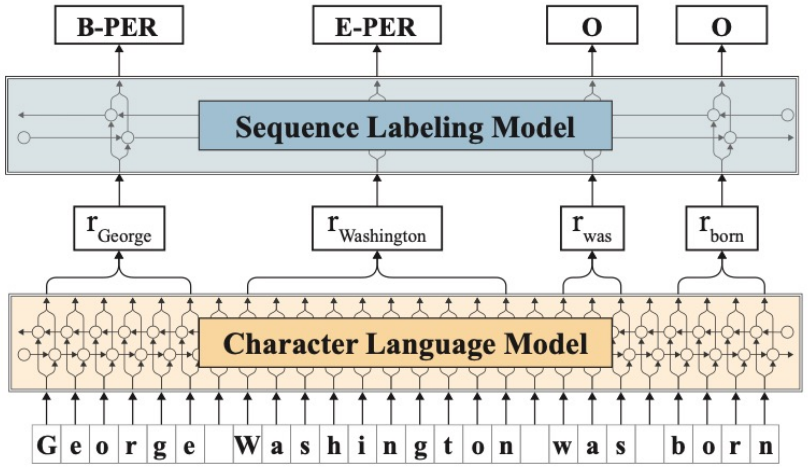
1 소개

- 현재 NER의 SOTA는 다음 3개의 서로 다른 임베딩을 결합하는 방법을 사용한다.
  1. 고전적인 단어 임베딩 (매우 큰 데이터 셋에 대해 pre-trained 됨)
  2. 문자 수준의 특징벡터 (pre-trained되지 않고, task-specific 한 단어 내부의 특징을 추출하기 위해 task data에 대해서만 학습됨)
  3. 문맥상의 단어 임베딩 (동음이의어, 단어의 문맥에 의존적인 성질 등을 판단)
- Contextual string embeddings
  - 문장 내에서 문맥화된 임베딩을 의미한다.
- Neural character-level language modeling
  - 이전 문자에 기반하여 다음 문자를 예측하는 모델은 문장 구조에 대한 사전 지식이 없더라도 문장론적/의미론적 요소를 포착할 수 있다. 이런 언어모델에서 적절한 hidden state의 선택을 통해 매우 효과적인 단어 임베딩을 생성할 수 있다.

Flair embedding

1 소개

- State-of-the-art sequence labeling
  - 앞서 얘기한 것들을 바탕으로 아래 그림과 같은 임베딩을 제안한다.



1. 먼저 문장이 pre-trained bidirectional character language model으로 입력된다.
2. 1 에서 얻은 단어 임베딩을 vanilla BiLSTM-CRF에 입력한다.

  - 이로써 contextual string embedding이 가능하다.

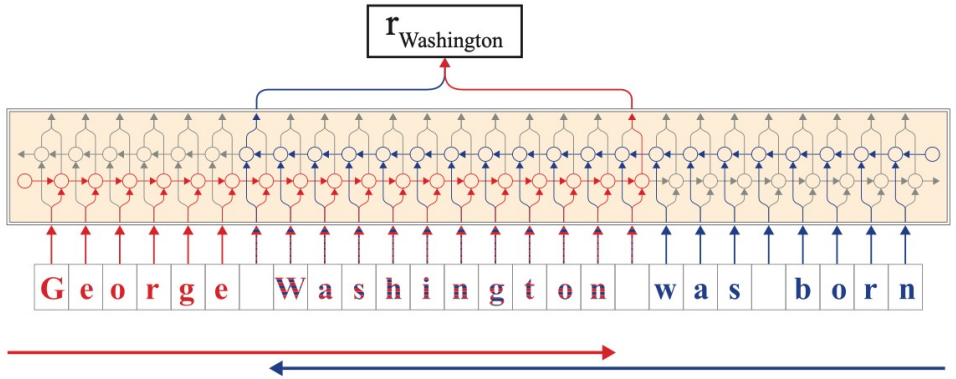
Flair embedding

2 Contextual String Embeddings

2.1 Recurrent Network States

- character-level language model의 목표는 문장 내의 0부터 T 까지의 문자에 대한 분포를 추정하는 것이다.
- 이 목적을 달성하기 위해 임의의 문자 이전의 모든 문자들에 대한 조건부확률을 학습한다.

2.2 Extracting Word Representations



- 정방향/역방향에서 각각 단어의 끝부분/시작부분의 hidden state를 추출한 뒤 concatenate한 것을 단어 임베딩으로 사용한다.

2 Contextual String Embeddings

2.3 Sequence Labeling Architecture

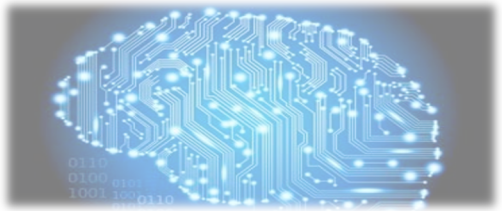
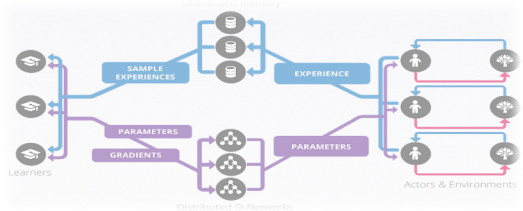
- 단어 임베딩은 BiLSTM-CRF으로 입력된다. (Huang et al. (2015))
- Stacking Embeddings
  - 여러 개의 임베딩을 결합(concatenating)하여 최종적인 단어 임베딩을 생성한다.
  - 예를 들어 앞서 생성한 character LM 임베딩을 Glove 임베딩과 결합하여 새로운 단어 임베딩을 만들 수 있다.
  - Stacking Embedding의 예시

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

3 Experiments

4 Conclusion

실험 과정과 결과 이하 생략

실험에 사용한 모델 구조 설명

실험에 사용한 모델 구조 설명

모델 정보 출력 후 확인

```
>>> print(model)

SequenceTagger(
  (embeddings): StackedEmbeddings(
    (list_embedding_0): WordEmbeddings('glove')
    (list_embedding_1): FlairEmbeddings(
      (lm): LanguageModel(
        (drop): Dropout(p=0.05, inplace=False)
        (encoder): Embedding(300, 100)
        (rnn): LSTM(100, 2048)
        (decoder): Linear(in_features=2048,
out_features=300, bias=True)
      )
    )
    (list_embedding_2): FlairEmbeddings(
      (lm): LanguageModel(
        (drop): Dropout(p=0.05, inplace=False)
        (encoder): Embedding(300, 100)
        (rnn): LSTM(100, 2048)
        (decoder): Linear(in_features=2048,
out_features=300, bias=True)
      )
    )
  )
  (word_dropout): WordDropout(p=0.05)
  (locked_dropout): LockedDropout(p=0.5)
  (embedding2nn): Linear(in_features=4196,
out_features=4196, bias=True)
  (rnn): LSTM(4196, 256, batch_first=True,
bidirectional=True)
  (linear): Linear(in_features=512, out_features=24,
bias=True)
  (beta): 1.0
  (weights): None
  (weight_tensor) None
)
```

실험에 사용한 모델 구조 설명

embeddings (StackedEmbedding)

- 세 가지 임베딩 (**WordEmbeddings('glove'), FlairEmbeddings('news-forward'), FlairEmbeddings('news-backward')**) 을 결합하여 사용 (StackedEmbeddings 오브젝트로 결합)
- 임베딩 길이는 각각 100, 2048, 2048
- **FlairEmbeddings** 의 lm (language model)은 encoder, rnn, decoder로 구성됨

1. encoder
   - 정의 : self.encoder = nn.Embedding(len(dictionary), embedding_size)
   - dictionary는 '<unk>', ' ', 'e', 'i', 'n', … 등등의 문자를 포함한 flair.data.Dictionary 오브젝트
2. rnn
   - 정의 : self.rnn = nn.LSTM(embedding_size, hidden_size, nlayers)
   - FlairEmbeddings에서 nlayers == 1
3. decoder
   - 정의 : self.decoder = nn.Linear(hidden_size, len(dictionary))

실험에 사용한 모델 구조 설명

모델 구조

1. embedding2nn
   - 정의 : `self.embedding2nn = torch.nn.Linear(embedding_dim, rnn_input_dim)`
2. rnn layer
   - 정의 :
```
self.rnn = getattr(torch.nn, self.rnn_type)(
                    rnn_input_dim,
                    hidden_size,
                    num_layers=self.nlayers)
```
3. linear layer
   - 정의 : `self.linear = torch.nn.Linear( hidden_size * num_directions, len(tag_dictionary) )`
4. transitions (crf에 사용)
   - 정의 :
```
self.transitions = torch.nn.Parameter( torch.randn( self.tagset_size,
                                self.tagset_size ) )
```

실험에 사용한 모델 구조 설명

모델 파라미터

- 모델 생성 코드

```
tagger = SequenceTagger(hidden_size=256,
                        embeddings=embeddings,
                        tag_dictionary=tag_dictionary,
                        tag_type='ner')
```

- embeddings는 앞에서 언급한 임베딩(StackedEmbeddings)을 의미
- tag_dictionary는 <unk>, <START>, <STOP>을 포함하여 총 24개의 tag가 존재하는 flair.data.Dictionary 오브젝트

실험에 사용한 모델 구조 설명

참고 문서

- https://huggingface.co/flair/ner-english
- https://github.com/flairNLP/flair/blob/master/flair/models/sequence_tagger_model.py