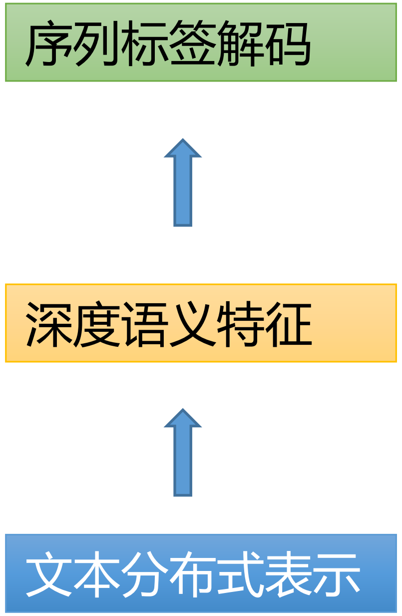
命名实体识别实践(二)-深度学习模型



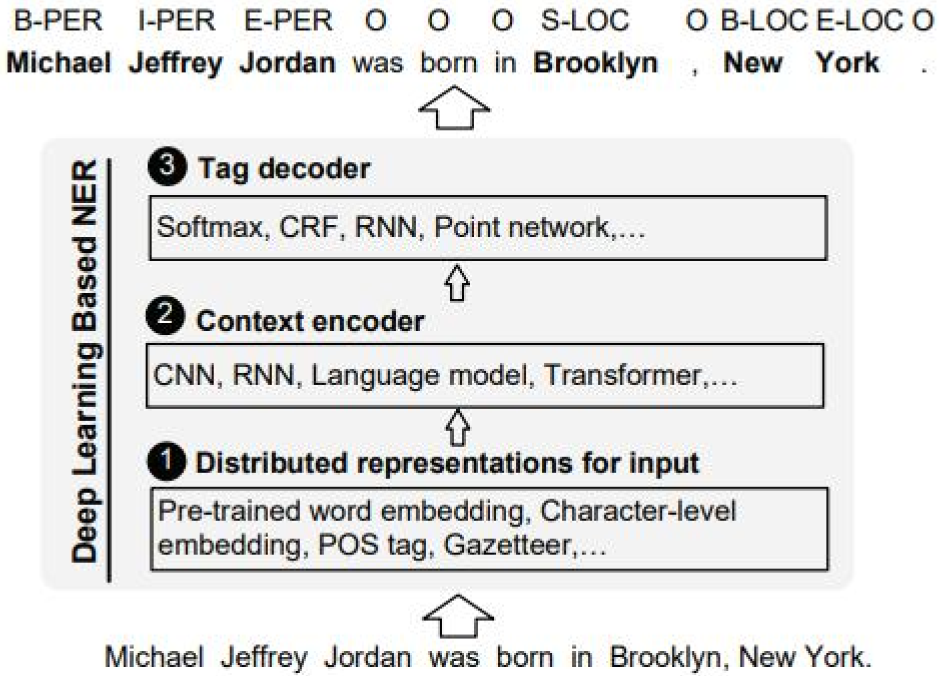
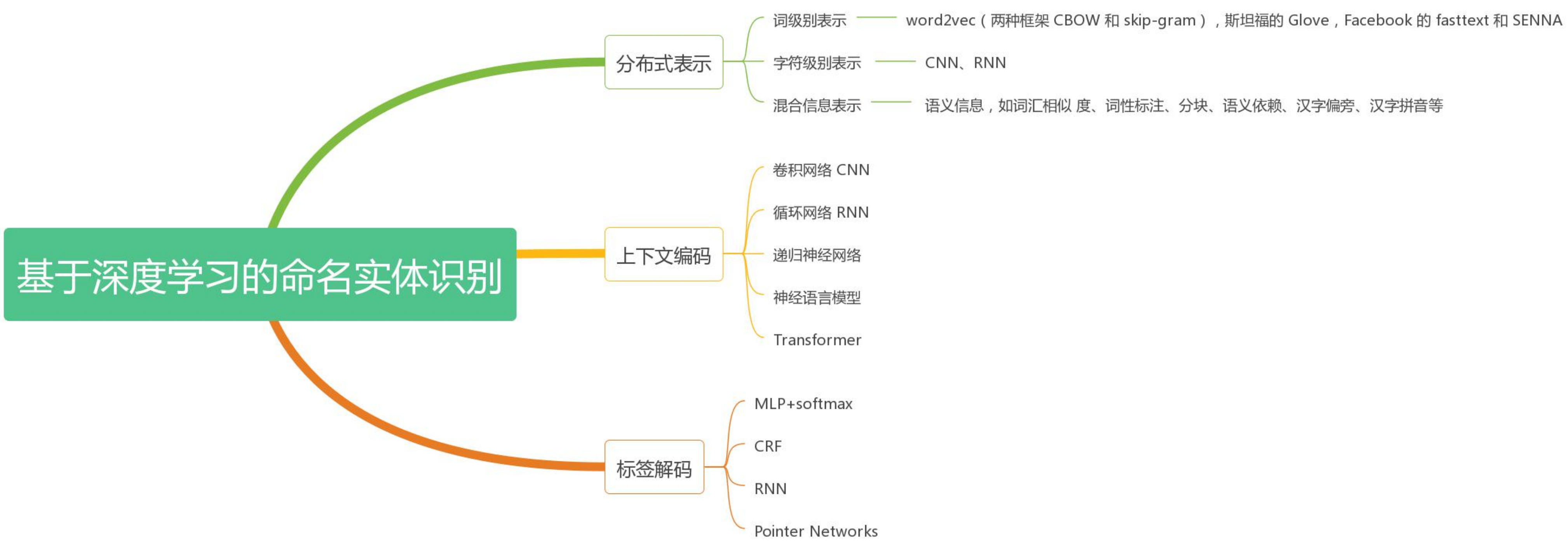


Fig. 2. The taxonomy of DL-based NER. From input sequence to predicted tags, a DL-based NER model consists of distributed representations for input, context encoder, and tag decoder.

深度学习的优势在于它的特征表达能力，使得模型能够自动学习到数据的潜在表示方法以及分类检测所需的过程。

NER使用深度学习的三个原因：

1.NER适用于非线性转化
2.深度学习节省了设计NER功能的大量精力
3.深度学习能通过梯度传播来训练，这样可以构建更复杂的网络



A Survey on Deep Learning for Named Entity Recognition

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li
Nanyang Technological University 南洋理工大学
2018

命名实体识别综述
https://zhuanlan.zhihu.com/p/373304254

Work	Input representation			Context encoder	Tag decoder	Performance (F-score)
	Character	Word	Hybrid			
[94]	-	Trained on PubMed	POS	CNN	CRF	GENIA: 71.01%
[89]	-	Trained on Gigaword	-	GRU	GRU	ACE 2005: 80.00%
[95]	-	Random	-	LSTM	Pointer Network	ATIS: 96.86%
[90]	-	Trained on NYT	-	LSTM	LSTM	NYT: 49.50%
[91]	-	SENNA	Word shape	ID-CNN	CRF	CoNLL03: 90.65%; OntoNotes5.0: 86.84%
[96]	-	Google word2vec	-	LSTM	LSTM	CoNLL04: 75.0%
[100]	LSTM	-	-	LSTM	CRF	CoNLL03: 84.52%
[97]	CNN	GloVe	-	LSTM	CRF	CoNLL03: 91.21%
[105]	LSTM	Google word2vec	-	LSTM	CRF	CoNLL03: 84.09%
[19]	LSTM	SENNA	-	LSTM	CRF	CoNLL03: 90.94%
[106]	GRU	SENNA	-	GRU	CRF	CoNLL03: 90.94%
[98]	CNN	GloVe	POS	BRNN	Softmax	OntoNotes5.0: 87.21%
[107]	LSTM-LM	-	-	LSTM	CRF	CoNLL03: 93.09%; OntoNotes5.0: 89.71%
[103]	CNN-LSTM-LM	-	-	LSTM	CRF	CoNLL03: 92.22%
[17]	-	Random	POS	CNN	CRF	CoNLL03: 89.86%
[18]	-	SENNA	Spelling, n-gram, gazetteer	LSTM	CRF	CoNLL03: 90.10%
[20]	CNN	SENNA	capitalization, lexicons	LSTM	CRF	CoNLL03: 91.62%; OntoNotes5.0: 86.34%
[116]	-	-	FOFE	MLP	CRF	CoNLL03: 91.17%
[101]	LSTM	GloVe	-	LSTM	CRF	CoNLL03: 91.07%
[113]	LSTM	GloVe	Syntactic	LSTM	CRF	W-NUT17: 40.42%
[102]	CNN	SENNA	-	LSTM	Reranker	CoNLL03: 91.62%
[114]	CNN	Twitter Word2vec	POS	LSTM	CRF	W-NUT17: 41.86%
[115]	LSTM	GloVe	POS, topics	LSTM	CRF	W-NUT17: 41.81%
[118]	LSTM	GloVe	Images	LSTM	CRF	SnapCaptions: 52.4%
[109]	LSTM	SSKIP	Lexical	LSTM	CRF	CoNLL03: 91.73%; OntoNotes5.0: 87.95%
[119]	-	WordPiece	Segment, position	Transformer	Softmax	CoNLL03: 92.8%
[121]	LSTM	SENNA	-	LSTM	Softmax	CoNLL03: 91.48%
[124]	LSTM	Google Word2vec	-	LSTM	CRF	CoNLL03: 86.26%
[21]	GRU	SENNA	LM	GRU	CRF	CoNLL03: 91.93%
[126]	LSTM	GloVe	-	LSTM	CRF	CoNLL03: 91.71%
[142]	-	SENNA	POS, gazetteers	CNN	Semi-CRF	CoNLL03: 90.87%
[143]	LSTM	GloVe	-	LSTM	Semi-CRF	CoNLL03: 91.38%
[88]	CNN	Trained on Gigaword	-	LSTM	LSTM	CoNLL03: 90.69%; OntoNotes5.0: 86.15%
[110]	-	GloVe	ELMo, dependency	LSTM	CRF	CoNLL03: 92.4%; OntoNotes5.0: 89.88%
[108]	CNN	GloVe	ELMo, gazetteers	LSTM	Semi-CRF	CoNLL03: 92.75%; OntoNotes5.0: 89.94%
[133]	LSTM	GloVe	ELMo, POS	LSTM	Softmax	CoNLL03: 92.28%
[137]	-	-	BERT	-	Softmax	CoNLL03: 93.04%; OntoNotes5.0: 91.11%
[138]	-	-	BERT	-	Softmax +Dice Loss	CoNLL03: 93.33%; **OntoNotes5.0: 92.07%**
[134]	LSTM	GloVe	BERT, document-level embeddings	LSTM	CRF	CoNLL03: 93.37%; OntoNotes5.0: 90.3%
[135]	CNN	GloVe	BERT, global embeddings	GRU	GRU	CoNLL03: 93.47%
[132]	CNN	-	Cloze-style LM embeddings	LSTM	CRF	**CoNLL03: 93.5%**
[136]	-	GloVe	Plooled contextual embeddings	RNN	CRF	CoNLL03: 93.47%

论文标题：End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF
论文链接：https://arxiv.org/pdf/1603.01354.pdf
相关代码：

- https://github.com/achernodub/targer
- https://github.com/ZubinGou/NER-BiLSTM-CRF-PyTorch
- https://github.com/ZhixiuYe/NER-pytorch

CNN for Character-level Representation

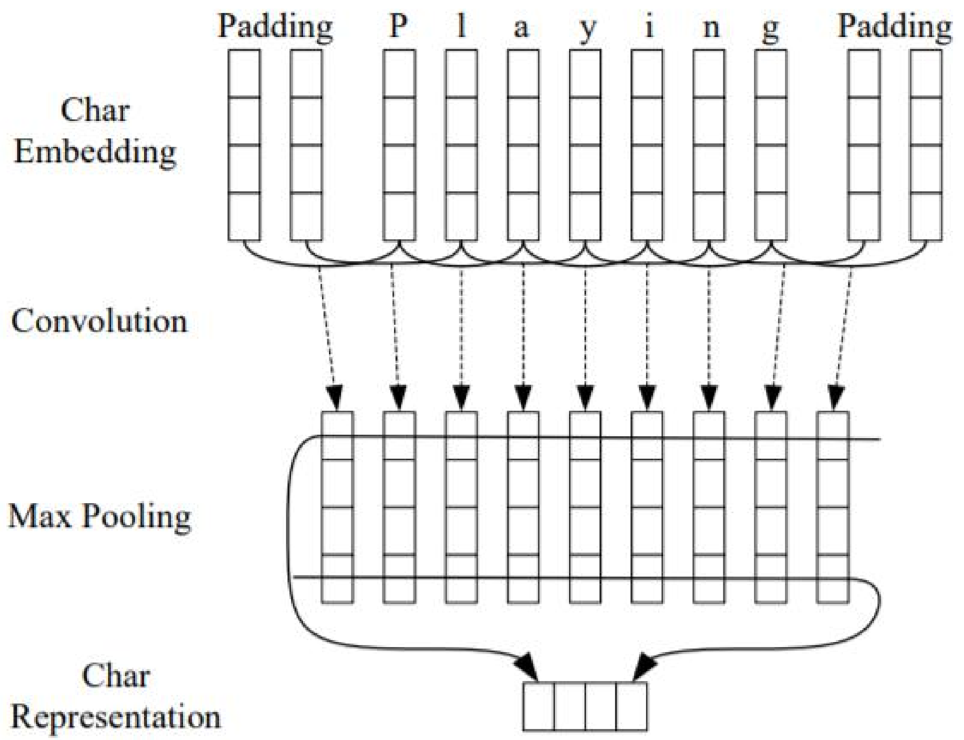


Figure 1: The convolution neural network for extracting character-level representations of words. Dashed arrows indicate a dropout layer applied before character embeddings are input to CNN.

LSTM Unit

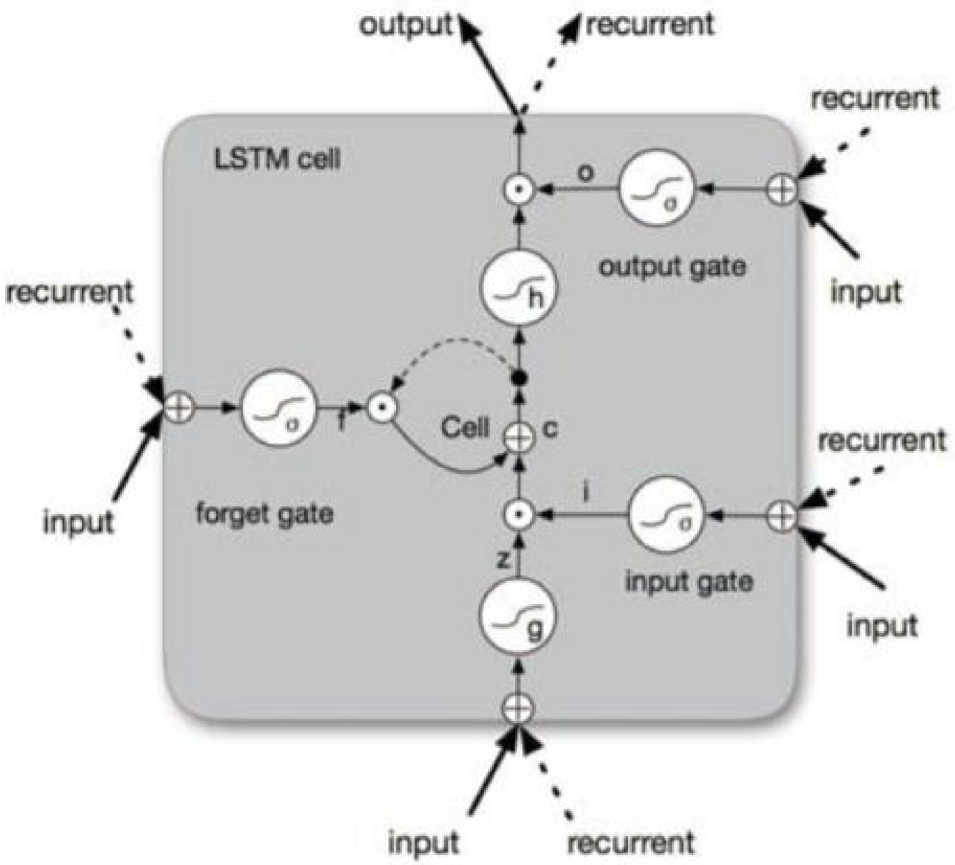


Figure 2: Schematic of LSTM unit.

Formally, the formulas to update an LSTM unit at time t are:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\boldsymbol{W}_i \mathbf{h}_{t-1} + \boldsymbol{U}_i \mathbf{x}_t + \boldsymbol{b}_i) \\
\mathbf{f}_t &= \sigma(\boldsymbol{W}_f \mathbf{h}_{t-1} + \boldsymbol{U}_f \mathbf{x}_t + \boldsymbol{b}_f) \\
\tilde{\mathbf{c}}_t &= \tanh(\boldsymbol{W}_c \mathbf{h}_{t-1} + \boldsymbol{U}_c \mathbf{x}_t + \boldsymbol{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{o}_t &= \sigma(\boldsymbol{W}_o \mathbf{h}_{t-1} + \boldsymbol{U}_o \mathbf{x}_t + \boldsymbol{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

BLSTM-CNNs-CRF

```python
class BiLSTM_CRF(nn.Module):
    def __init__(
        self,
        vocab_size,
        tag_to_ix,
        embedding_dim,
        hidden_dim,
        char_lstm_dim=25,
        char_to_ix=None,
        pre_word_embeds=None,
        char_embedding_dim=25,
        use_gpu=False,
        n_cap=None,
        cap_embedding_dim=None,
        use_crf=True,
        char_mode="CNN",
    ):
        super(BiLSTM_CRF, self).__init__()
        self.use_gpu = use_gpu
        self.device = torch.device("cuda" if self.use_gpu else "cpu")
        self.embedding_dim = embedding_dim
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size
        self.tag_to_ix = tag_to_ix
        self.n_cap = n_cap  # Capitalization feature num
        self.cap_embedding_dim = cap_embedding_dim  # Capitalization feature dim
        self.use_crf = use_crf
        self.tagset_size = len(tag_to_ix)
        self.out_channels = char_lstm_dim
        self.char_mode = char_mode

        print("char_mode: %s, out_channels: %d, hidden_dim: %d, " % (char_mode, char_lstm_dim, hidden_dim))

        if self.n_cap and self.cap_embedding_dim:
            self.cap_embeds = nn.Embedding(self.n_cap, self.cap_embedding_dim)
            torch.nn.init.xavier_uniform_(self.cap_embeds.weight)

        if char_embedding_dim is not None:
            self.char_lstm_dim = char_lstm_dim
            self.char_embeds = nn.Embedding(len(char_to_ix), char_embedding_dim)
            torch.nn.init.xavier_uniform_(self.char_embeds.weight)
            if self.char_mode == "LSTM":
                self.char_lstm = nn.LSTM(char_embedding_dim, char_lstm_dim, num_layers=1, bidirectional=True)
                init_lstm(self.char_lstm)
            if self.char_mode == "CNN":
                self.char_cnn3 = nn.Conv2d(
                    in_channels=1,
                    out_channels=self.out_channels,
                    kernel_size=(3, char_embedding_dim),
                    padding=(2, 0),
                )

        self.word_embeds = nn.Embedding(vocab_size, embedding_dim)
        if pre_word_embeds is not None:
            self.pre_word_embeds = True
            self.word_embeds.weight = nn.Parameter(torch.FloatTensor(pre_word_embeds))
        else:
            self.pre_word_embeds = False
```