Рекуррентные нейронные сети для разметки последовательности

Математические методы анализа текстов осень 2020

Попов Артём Сергеевич

23 сентября 2020

Задача разметки последовательности (sequence tagging)

Дано:

- D — множество размеченных последовательностей (x, y)
- $x = \{x_1, \ldots, x_n\}$ — последовательность входных объектов
- $y = \{y_1, \ldots, y_n\}$ — последовательность выходных векторов
- $x_i \in X$, $y_i \in Y$

Необходимо: по входной последовательности предсказать элементы выходной последовательности

В задачах анализа текста:

- Входная последовательность — последовательность слов, выходная — последовательность меток
- Длина разных последовательностей может быть различна
- Последовательности можно привести к одной длине дополнив их специальным <PAD> токеном

Примеры задач разметки в NLP

Задача разметки может быть и «вспомогательной» и «продуктовой»:

- Разметка по частям речи (Part-of-speech tagging, POS)
- Распознавание именованных сущностей (Named Entity Recognition, NER)
- Разметка семантических ролей (Semantic Role Labeling, SRL)
- Выделение текстовых полей данных (Slot filling)
- Разметка библиографических данных

BIO-нотация

Для составных сущностей используется BIO-нотация:

- B (Begin) — первое слово сущности
- I (Inside) — слово внутри сущности
- O (Outsied) — слово вне сущности

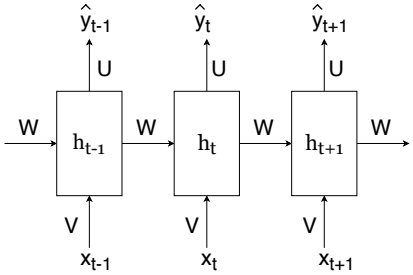
Пример входа и выхода (x, y)*:*

Alex	is	going	to	Los	Angeles
B-PER	O	O	O	B-LOC	I-LOC

Подходы для разметки последовательностей

- Правиловые подходы (rule-based)
- Классификатор на признаках, зависящих от позиции элемента в последовательности
- Графические модели (HMM/CRF)
- Нейронные сети (рекуррентные/свёрточные/трансформеры)
- Комбинация подходов

Модель рекуррентной нейронной сети[1] (RNN)



h_t — скрытое состояние в момент времени t

$$h_t = f(Vx_t + Wh_{t-1} + b)$$
$$\hat{y}_t = g(Uh_t + \hat{b})$$

Обучение сети — минимизация суммарных потерь:

$$\sum_{t=1}^{n} \mathcal{L}_t(y_t, \hat{y}_t) \to \min_{V,U,W,b,\hat{b}}$$

Сеть обучается с помощью алгоритма backpropagation (through time)

[1] Rumelhart et al; Learning internal representations by error propagation; 1985

Детали обучения RNN: производные по U и W

Градиент по U зависит только от величин в момент t:

$$\frac{d\mathcal{L}_t}{dU} =$$

Детали обучения RNN: производные по U и W

Градиент по U зависит только от величин в момент t:

$$\frac{d\mathcal{L}_t}{dU} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial U}$$

Детали обучения RNN: производные по U и W

Градиент по U зависит только от величин в момент t:

$$\frac{d\mathcal{L}_t}{dU} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial U}$$

Градиент по W зависит от всех предыдущих величин:

$$\frac{d\mathcal{L}_t}{dW} =$$

Детали обучения RNN: производные по U и W

Градиент по U зависит только от величин в момент t:

$$\frac{d\mathcal{L}_t}{dU} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial U}$$

Градиент по W зависит от всех предыдущих величин:

$$\frac{d\mathcal{L}_t}{dW} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{dh_t}{dW}$$

Детали обучения RNN: производные по U и W

Градиент по U зависит только от величин в момент t:

$$\frac{d\mathcal{L}_t}{dU} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial U}$$

Градиент по W зависит от всех предыдущих величин:

$$\frac{d\mathcal{L}_t}{dW} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{dh_t}{dW}$$

$$\frac{dh_t}{dW} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{dW} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{dh_{t-2}}{dW} =$$

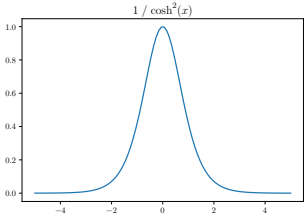
$$= \ \ldots \ = \sum_{k=1}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$

Градиент по V считается аналогично градиенту по W

Детали обучения RNN: взрыв и затухание градиентов

Взрыв градиента:

$$\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \to \infty$$



Затухание градиента:

$$\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \to 0$$

$$\frac{\partial h_i}{\partial h_{i-1}} = diag\left(\frac{1}{\mathrm{ch}^2(z_i)}\right) W$$

$$z_i = Vx_i + Wh_{i-1} + b$$

если $f = \tanh$

Популярные способы борьбы с взрывом/затуханием:

- Gradient clipping (против взрыва)
- Модели LSTM и GRU (против затухания)

Gradient clipping

Ограничение нормы градиентов:

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

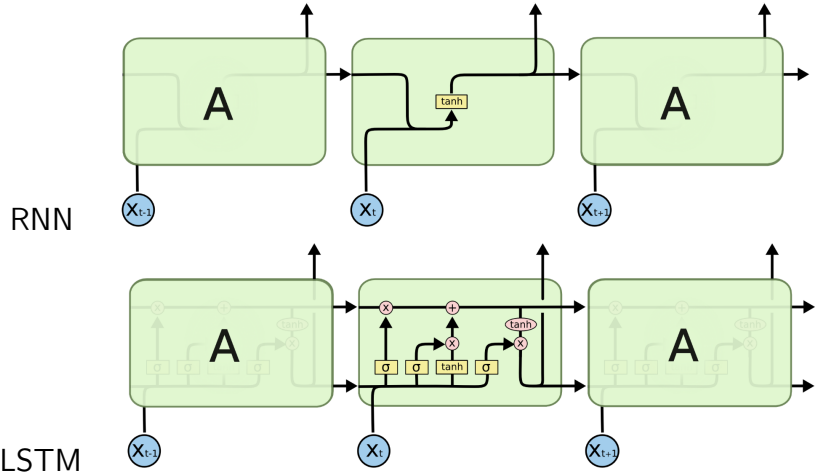
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

Как выбрать порог?

Можно использовать константу. Можно брать среднюю норму градиента для весов по запускам без gradient clipping

LSTM сеть

Используем более сложную структуру ячейки:

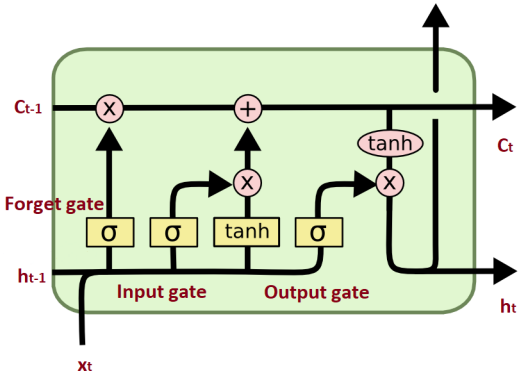


[1] Hochreiter and Schmidhuber (Neural Computation); Long Short-Term Memory; 1997

[1] ссылка на туториал по LSTM

LSTM ячейка

$$z_t = [h_{t-1}, x_t]$$
$$f_t = \sigma(W_f \cdot z_t + b_f)$$
$$i_t = \sigma(W_i \cdot z_t + b_i)$$
$$\hat{C}_t = \text{th}(W_c \cdot z_t + b_c)$$
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$
$$o_t = \sigma(W_o \cdot z_t + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$

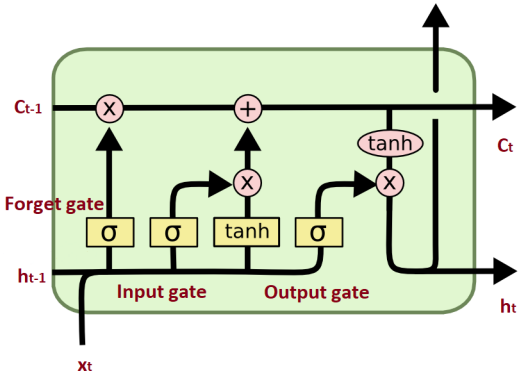


Обучается с помощью алгоритма Backpropagation

Почему решает проблему затухающих градиентов?

LSTM ячейка

$$z_t = [h_{t-1}, x_t]$$
$$f_t = \sigma(W_f \cdot z_t + b_f)$$
$$i_t = \sigma(W_i \cdot z_t + b_i)$$
$$\hat{C}_t = \text{th}(W_c \cdot z_t + b_c)$$
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$
$$o_t = \sigma(W_o \cdot z_t + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$

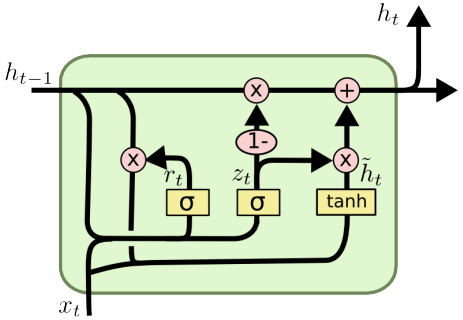


Обучается с помощью алгоритма Backpropagation

Почему решает проблему затухающих градиентов?

C_t зависит от C_{t-1} линейно, т.е $\frac{\partial C_t}{\partial C_{t-1}} = f_t$, при инициализации b_f большими числами, $f_t \approx 1$

GRU ячейка



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

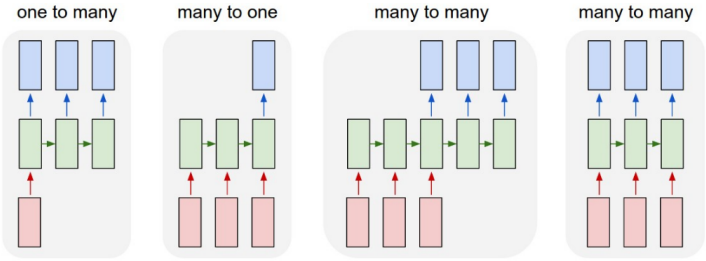
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

На практике часто используют GRU ячейку, т.к. она быстрее учится.

[1] Cho et al (EMNLP 2014), Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation; 2014

Разные архитектуры рекуррентных сетей



Примеры задач:

one to many Генерация описания изображения

many to one Классификация предложений

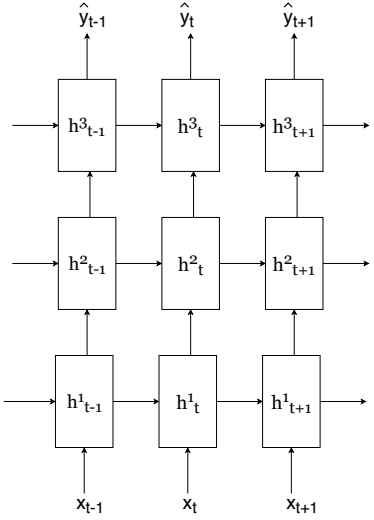
many to many(1) Перевод с одного языка на другой

many to many(2) Определение частей речи

Глубокие рекуррентные сети (deep RNN, layers stacking)

Выходы одной рекуррентной сети подаются на вход другой:

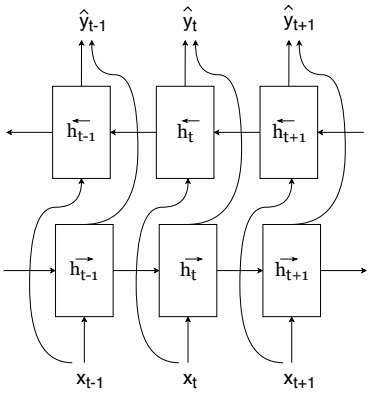
$$h_t^1, C_t^1 = LSTM(h_{t-1}^1, C_{t-1}^1, x_t)$$
$$h_t^2, C_t^2 = LSTM(h_{t-1}^2, C_{t-1}^2, h_t^1)$$
$$h_t^3, C_t^3 = LSTM(h_{t-1}^1, C_{t-1}^1, h_t^2)$$
$$y_t = g(Uh_t^2 + \hat{b})$$



Двунаправленные сети (bidirectional)

Конкатенация выходов двух сетей, одна идёт слева направо, другая справа налево:

$$\overrightarrow{h_t}, \overrightarrow{C_t} = \overrightarrow{LSTM}(\overrightarrow{h_{t-1}}, \overrightarrow{C_{t-1}}, x_t)$$
$$\overleftarrow{h_t}, \overleftarrow{C_t} = \overleftarrow{LSTM}(\overleftarrow{h_{t-1}}, \overleftarrow{C_{t-1}}, x_t)$$
$$y_t = g(U[\overrightarrow{h_t}, \overleftarrow{h_t}] + \hat{b})$$



На практике часто работают лучше чем однонаправленные!

Работа с словами не из словаря (OOV, out of vocabulary)

Добавление в словарь <UNK> токена

- Заменить редкие слова на <UNK> токен при обучении
- Во время обучения с малой вероятностью заменять часть слов на <UNK>

Использовать посимвольную RNN (charRNN)

- Вероятность встретить новый символ крайне мала...
- Во многих задачах charRNN работает не хуже wordRNN

Использовать посимвольную RNN для новых слов

- Если встречаем незнакомое слово, используем charRNN для его кодирования
- На каждой итерации обучения с малой вероятностью считаем одно из слов новым

Пример создания LSTM-теггера в Pytorch

```
import torch.nn as nn

class LSTMTagger(nn.Module):
    def __init__(self, embedding_dim, hidden_dim,
                                vocab_size, tagset_size):
        super(LSTMTagger, self).__init__()

        self.word_embeddings = nn.Embedding(
            vocab_size, embedding_dim)

        self.lstm = nn.LSTM(embedding_dim, hidden_dim)

        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

    ...
```

Пример создания LSTM-теггера в Pytorch

```
class LSTMTagger(nn.Module):
    ...

    def forward(self, sentence):
        sentence_embeddings = self.word_embeddings(sentence)

        lstm_out, _ = self.lstm(
            sentence_embeddings.view(len(sentence), 1, -1)
        )

        tag_scores = self.hidden2tag(
            lstm_out.view(len(sentence), -1)
        )
        return tag_scores
```

Резюме по RNN

- Рекуррентный нейронные сети (RNN) — нейросетевая архитектура для работы с последовательностями
- Обучается с помощью алгоритма backpropagation
- В исходном виде RNN плохо обучается, необходимо использовать LSTM/GRU и gradient clipping
- С помощью разных архитектур сети можно решать разные задачи
- Ещё одна архитектура для работы с последовательностями — трансформер (на следующей лекции)

Базовая модель разметки (tagging)

Вход модели:

- $x = x_1^n, \quad x_t$ — one-hot вектор t-го слова

Выход модели:

- $\hat{y} = \hat{y}_1^n, \quad \hat{y}_t = p(y|x_1^n, t)$ — распределение тегов t-го слова
- один тег — префикс-тип (например, B-PER) или O

Архитектура сети:

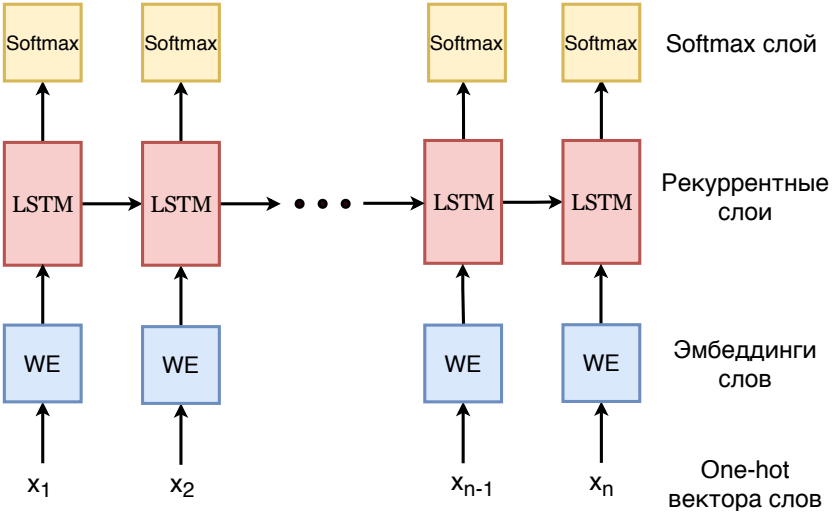
$$v_t = \text{Embedding}(t)$$

$$h_t, C_t = \text{LSTM}(h_{t-1}, C_{t-1}, v_t)$$

$$\hat{y}_t = \text{softmax}(Uh_t + \hat{b})$$

$$\mathcal{L} = \sum_{t=1}^{n} \mathcal{L}_t, \quad \mathcal{L}_t = -\sum_{y \in Y} [y = y_t] \log p(y = y_t | x_1^n, t)$$

Базовая модель разметки



Замечания о базовой модели

- При предобработке слова обычно не приводятся к нижнему регистру
- Лучше использовать bidirectional сеть
- Может быть несколько слоёв (но редко > 2)
- Эмбеддинги слов могут быть:
  - инициализированы предобученной моделью, заморожены во время обучения
  - инициализированы предобученной моделью, обучаются во время обучения
  - случайно инициализированы, обучаются во время обучения
- Dropout помогает при обучении

Почему регистр важен? (задача NER)

```
Facebook нашел нового финансового директора . Финансовым
директором социальной сети Facebook назначен 39-летний Дэвид
Эберсман (David Ebersman), сообщает The Wall Street Journal.

T1 ORG 0 8 Facebook
T2 ORG 83 91 Facebook
T3 PER 111 142 Дэвид Эберсман (David Ebersman)
T4 ORG 153 176 The Wall Street Journal
```

Модификации модели

- Замена RNN на трансформеры или свёрточные сети
- **Постобработка выходов сети, например через CRF слой**
- **Усложнение слоя представлений слов, увеличение количества представлений**
- Предобученные глубокие сети (transfer learning)
- Разделение предсказания префиксов и типов
- Semi-supervised и multi-task learning

CRF: напоминание

Моделируем вероятность последовательности y при условии x линейной моделью с вектором весов $w \in \mathbb{R}^d$:

$$p(y|x, w) = \frac{1}{Z(x, w)} \exp\left(\langle w, F(x, y)\rangle\right)$$

Признак $F_j(x, y)$ задаётся через признак f_j специального вида:

$$F_j(x, y) = \sum_{i=1}^{n} f_j(y_{i-1}, y_i, x_i, i)$$

После некоторых преобразований:

$$\langle w, F(x, y)\rangle = \sum_{i=1}^{n} \sum_{j=1}^{d} w_j f_j(y_{i-1}, y_i, x_i, i) = \sum_{i=1}^{n} G_{x,i}[y_{i-1}, y_i]$$

CRF: напоминание

Для обучения модели CRF максимизируем правдоподобие, используя SGD:

$$\sum_{(x,y)\in D} \log p(y|x, w) \to \max_w$$

Для вычисления y по x при известных w необходимо найти:

$$\hat{y} = \arg\max_{y\in Y^n} p(y|x, w)$$

За счёт марковского свойства элементарных признаков f_j, расчёт градиента слагаемого правдоподобия и $\hat{y}$ можно провести за $O(|Y|^2 n)$ (алгоритм вперёд-назад и алгоритм Витерби соответственно).

Почему CRF может помочь модели?

Не все выходные последовательности соответствуют формату:

- *B-PER I-LOC I-LOC O*
- *O I-LOC I-LOC O*

Таким образом, CRF — обучаемый пост-процессинг последовательности:

- функционал CRF явно учитывает совместное положение выходных меток в отличие от RNN
- функционал CRF для последовательности не разбивается на пословные слагаемые

Важно. Если данных много, постпроцессинг не нужен. Но для задачи разметки часто тяжело собрать большие выборки данных.

Признаки CRF для обработки выходов RNN

Пусть после применения стандартного теггера к последовательности x получается последовательность s, $s_i \in \mathbb{R}^{|Y|}$.
Выход соответствующий y элемента s_i будем обозначать $s(i, y)$.

Будем использовать в модели CRF два вида признаков:

- $|Y| \times |Y|$ признаков, учитывающих выход модели:

$$\phi_{uv}(y_{i-1}, y_i, x_i) = \mathbb{I}[y_{i-1} = u]\mathbb{I}[y_i = v]s(i, y_i),$$

- $|Y| \times |Y|$ признаков, учитывающих связь меток:

$$\psi_{uv}(y_{i-1}, y_i) = \mathbb{I}[y_{i-1} = u]\mathbb{I}[y_i = v],$$

Преобразование выражения для признаков

Линейные коэффициенты модели будем обозначать $w(u, v)$ и $a(u, v)$. После преобразований $G_{x,i}[y_{i-1}, y_i]$ получим:

$$G_{x,i}[y_{i-1}, y_i] = \sum_{u \in Y} \sum_{v \in Y} (w(u, v)\phi_{uv}(y_{i-1}, y_i, x_i) + a(u, v)\psi_{uv}(y_{i-1}, y_i)) =$$

$$= \sum_{u \in Y} \sum_{v \in Y} \mathbb{I}[y_{i-1} = u]\mathbb{I}[y_i = v](w(u, v)s(i, y_i) + a(u, v)) =$$

$$= w(y_{i-1}, y_i)s(i, y_i) + a(y_{i-1}, y_i)$$

Часто при реализации полагают $w(u, v) = 1 \quad \forall u \in Y, v \in Y$.

Алгоритм обучения сети вместе с CRF

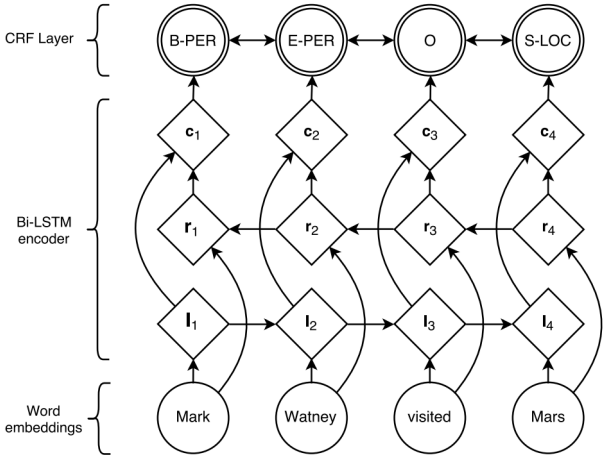
Выход biLSTM — вход для CRF.

Т.к. CRF обучается через градиентные методы, можем пробрасывать градиенты CRF в backpropgation алгоритме.

Algorithm 1 Bidirectional LSTM CRF model training procedure

```
for each epoch do
    for each batch do
        1) bidirectional LSTM-CRF model forward pass:
            forward pass for forward state LSTM
            forward pass for backward state LSTM
        2) CRF layer forward and backward pass
        3) bidirectional LSTM-CRF model backward pass:
            backward pass for forward state LSTM
            backward pass for backward state LSTM
        4) update parameters
    end for
end for
```

CRF слой



[1] Lample et al (NAACL 2016). Neural Architectures for Named Entity Recognition

Улучшение от CRF в biLSTM[1]

Результаты biLSTM + CRF превосходят остальные подходы

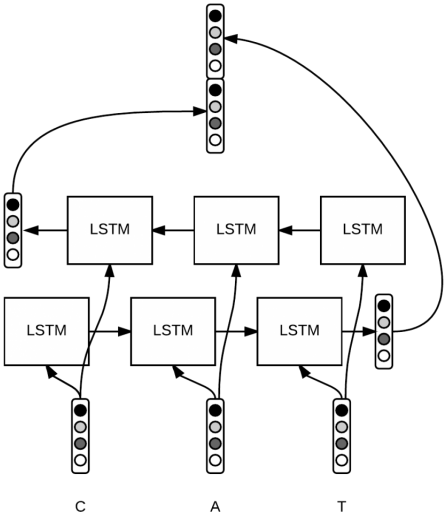
Table 2: Comparison of tagging performance on POS, chunking and NER tasks for various models.

		POS	CoNLL2000	CoNLL2003
Random	Conv-CRF (Collobert et al., 2011)	96.37	90.33	81.47
	LSTM	97.10	92.88	79.82
	BI-LSTM	97.30	93.64	81.11
	CRF	97.30	93.69	83.02
	LSTM-CRF	**97.45**	93.80	84.10
	BI-LSTM-CRF	97.43	**94.13**	**84.26**
Senna	Conv-CRF (Collobert et al., 2011)	97.29	94.32	88.67 (89.59)
	LSTM	97.29	92.99	83.74
	BI-LSTM	97.40	93.92	85.17
	CRF	97.45	93.83	86.13
	LSTM-CRF	97.54	94.27	88.36
	BI-LSTM-CRF	**97.55**	**94.46**	**88.83 (90.10)**

Добавление CRF помогает и в CNN, и в трансформерах...

[1] Huang et al (2015); Bidirectional LSTM-CRF Models for Sequence Tagging.

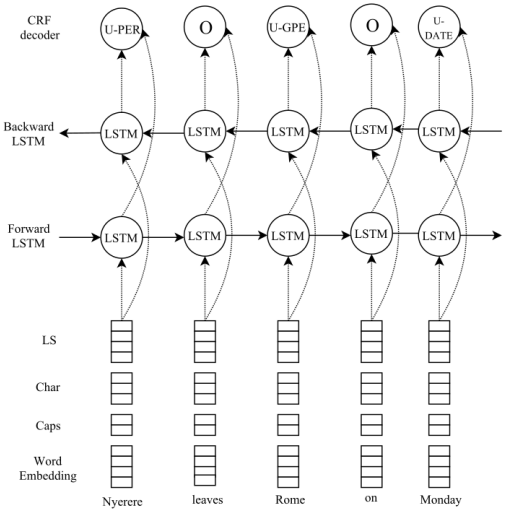
Посимвольные представления слов



Улучшение работы для слов с опечатками и ошибками:

- Кодируем слово посимвольной LSTM
- Можно использовать только эти эмбеддинги
- Можно конкатенировать с «табличными» эмбеддингами слов
- Можно вместо LSTM использовать свёртку

Дополнительные представления: регистр и тип



В модели[1] используется 4 вида эмбеддингов:

- стандартные
- признаки регистра
- посимвольные
- тип

[1] Ghaddar et al (2018), Robust Lexical Features for Improved Neural Network Named-Entity Recognition

Как строились представления типа?

1. Используется корпус WiFiNE, содержащий для слов относящиеся к ним категории (например, для *hilton* — */building/hotel, /building/restaurant, /person/actor*)
2. Также корпус WiFine содержит предложения, в которых некоторым словам соответствуют необходимые сущности
3. По такому корпусу обучаются эмбеддинги (FastText) — единое пространство для слов и сущностей слов
4. Представление типа для слова — вектор расстояний от эмбеддинга слова до всех представлений сущностей

Это полезно. Для улучшения качества иногда можно использовать специальные корпуса или лингвистические ресурсы.

Резюме RNN-теггер

Стандартная архитектура теггера:

1. Табличные эмбеддинги / посимвольные RNN эмбеддинги / посимвольные CNN эмбеддинги
2. RNN / CNN / Трансформер
3. CRF слой

Если нужно улучшить качество:

- Добавление CRF слоя почти всегда улучшает качество, но замедляет обучение и инференс
- Добавление посимвольных представлений улучшает качество, если в данных много опечаток, но сильно замедляет обучение и инференс

Задача POS (Part of speech tagging)

- Для каждого слова в предложении определить часть речи
- Простая задача — решается хорошо простыми моделями
- Вспомогательная задача

Зачем могут использоваться pos-теги:

- Снятие омонимии (мыло_NOUN и мыло_VERB)
- Дополнительный признак / дополнительное представление
- Выделение стоп-слов (предлоги и союзы — стоп-слова)
- Группировка слов по важности (определение темы текста — существительные важнее глаголов)

Открытые модели POS

Подходят для русского языка:

- pymorphy2 — не контекстный rule-based
- rnnmorph — нейросетевой (biLSTM-CRF) + дополнительные грамматические признаки на входе
- UDPipe — нейросетевой, есть предобученные модели для 50 языков (есть русский)

Подходят для других языков:

- nltk — разные модели (rule-based, n-граммные, графические модели)
- StanfordPOSTagger — графические модели, совместим с nltk, есть предобученные модели
- spacy — нейросетевые модели, есть предобученные модели для 9 языков (официальной поддержки русского пока нет)

Разметка POS

Есть несколько лингвистических концепций разметки pos-тегов:

- Universal POS tags (в UDPipe)
  (единый международный стандарт)
- OpenCorpora tags (в Pymorphy2)
  (например, различаются хороший (ADJF) и хорош (ADJS))

Некоторые библиотеки (например, лемматизаторы в nltk) принимают теги в определённом формате!

Задача NER (Named entity recognition)

- Для каждого слова в предложении определить является ли он частью какой-либо именованной сущностью
- Более сложная задача чем POS
- Может быть и вспомогательной, и конечной задачей
- Может быть частью задачи Named Entity Linking (NEL) — соотнести найденную сущность с сущностью из списка

Где используется NER:

- Диалоговые системы
- Дополнительный признак
- Поиск
- Деперсонализация данных
- Выделение сущностей в новостном потоке (выделить все упоминания Трампа в новостях)

Виды ошибок системы NER

	in	New	York
Правильный вариант	O	B-LOC	I-LOC
Точное совпадение	O	B-LOC	I-LOC
Лишняя сущность	B-LOC	B-LOC	I-LOC
Пропуск сущности	O	O	O
Неправильный тип	O	B-ORG	I-ORG
Неправильные границы	B-LOC	I-LOC	I-LOC
Неправильные тип и границы	B-ORG	I-ORG	I-ORG

[1] http://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/

Способы задания метрик качества

Макро-метрики: смотрим на долю правильно размеченых предложений.
Микро-метрики: смотрим на долю правильно размеченных сущностей.

Микро-метрики:

- precision — доля правильно распознанных сущностей среди всех распознанных сущностей
- recall — доля правильно распознанных сущностей среди всех истинных сущностей
- f1 — среднее гармоническое precision и recall

Можно считать микро-метрики отдельно по каждому типу сущностей.

Способы задания метрик качества: SemEval

Считаем микро-метрики по четырём показателям, каждый показатель задаётся допустимыми ошибками:

1. Strict — нет ошибок
2. Type — неправильный тип
3. Exact — неправильные границы
4. Partial — неправильный тип + неправильные границы

Открытые модели NER

Подходят для русского языка:

- pymorphy2 — rule-based
- natasha — rule-based и CNN-CRF
- deeppavlov – RNN-CRF и BERT-based предобученные модели

Подходят для других языков:

- spacy — нейросетевые модели (свёрточные сети)

Как обучался теггер из natasha?

Проблема обучения теггеров — отсутствие большого датасета для разметки.

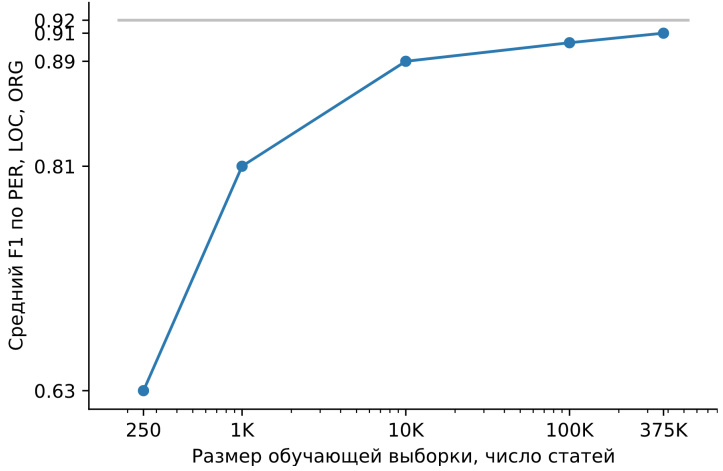
Создадим синтетический датасет разметки следующим образом:

1. Обучим очень тяжёлую модель на основе BERT на небольшом NER датасете ($\approx$ 1k объектов)
2. Полученным теггером разметим большое число объектов ($\approx$ 700k объектов)
3. Обучим лёгкую модель на синтетическом датасете

[1] https://habr.com/ru/post/516098/

Результаты natasha

Зависимость качества от количества объектов в синтетическом датасете:



Резюме по лекции

- Рекуррентные нейронные сети (LSTM, GRU) позволяют решать задачи машинного обучения на последовательностях
- Комбинация CRF с нейросетями для последовательностей позволяет улучшить качество решения задачи разметки
- Не всегда нейросетевые решения необходимы, для некоторых задач достаточно простых rule-based подходов