# Рекуррентные нейронные сети для разметки последовательности

## Математические методы анализа текстов осень 2019

Попов Артём Сергеевич

МГУ имени М. В. Ломоносова, факультет ВМК, кафедра ММП

## 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 предсказывать элементы выходной последовательности

**В задачах анализа текста:**

- Входная последовательность — последовательность слов, выходная — последовательность меток
- Длина всех последовательностей $(x, y)$ различна
- Последовательности можно привести к одной длине дополнив их специальным <PAD> токеном

## Подходы для разметки последовательностей

- Правиловые подходы (rule-based)
- Обучение отдельного классификатора на признаках, зависящих от позиции элемента в последовательности
- Графические модели (HMM/CRF)
- **Рекуррентные нейронные сети**
- Свёрточные нейронные сети
- Трансформеры
- **Комбинация подходов**

## Примеры задач разметки в NLP

- 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

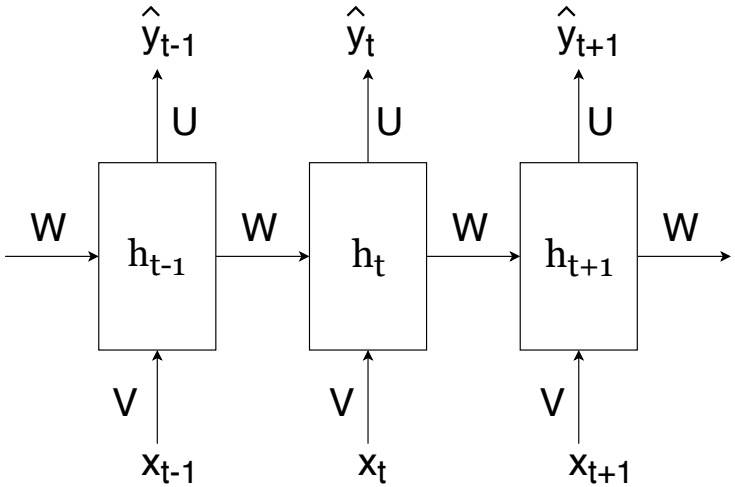
Для составных сущностей используется BIO-нотация:

- 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 |

# Модель рекуррентной нейронной сети (RNN)




$h_t$ — скрытое состояние в момент $t$

$$h_t = f(Vx_t + Wh_{t-1} + b)$$

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

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[1]

[1] Часто, вариацию алгоритма backpropagation для обучения RNN называют backpropagation through time

## 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 $t$:

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

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:

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

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

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

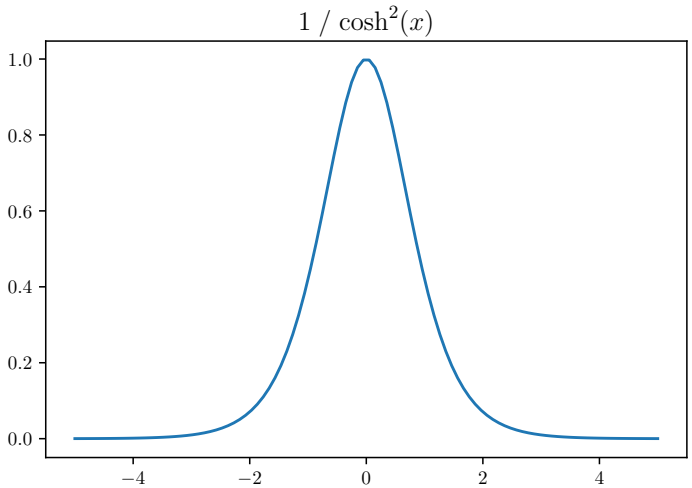
$$= \ \dots \ = \sum_{k=1}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$

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:

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

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

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

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

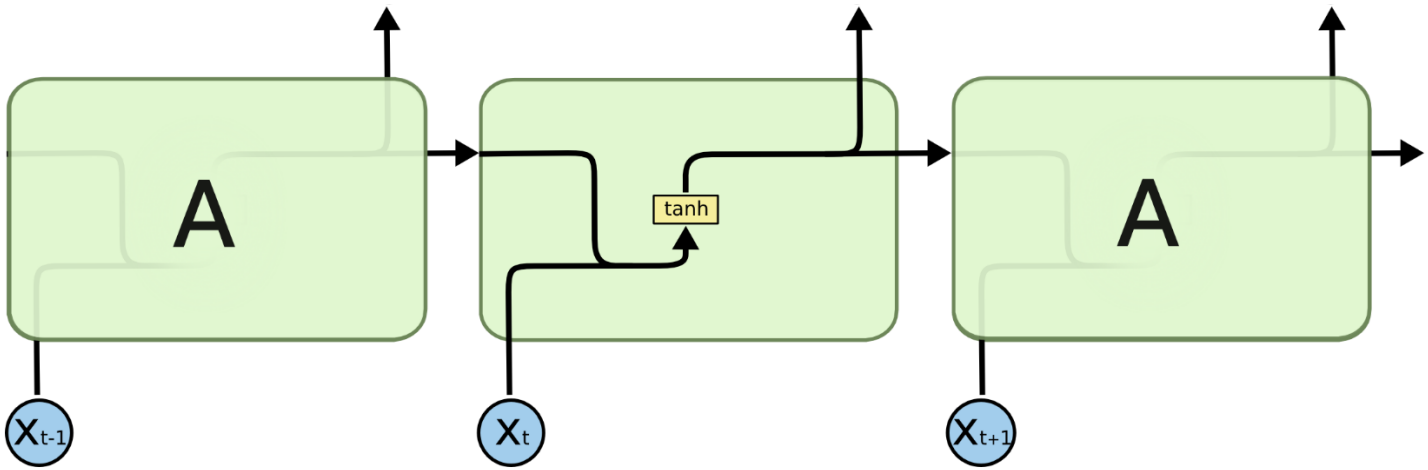
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
**if** $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
**end if**

Как выбрать порог?
Например, брать среднюю норму градиента для весов по запускам без gradient clipping

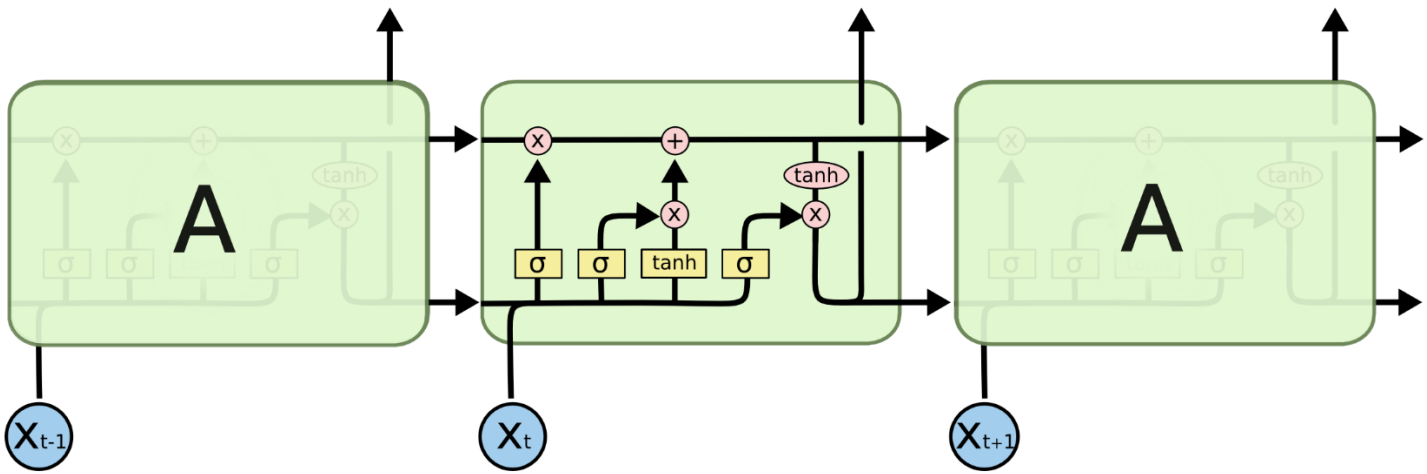
# LSTM сеть

Используем более сложную структуру ячейки:

RNN

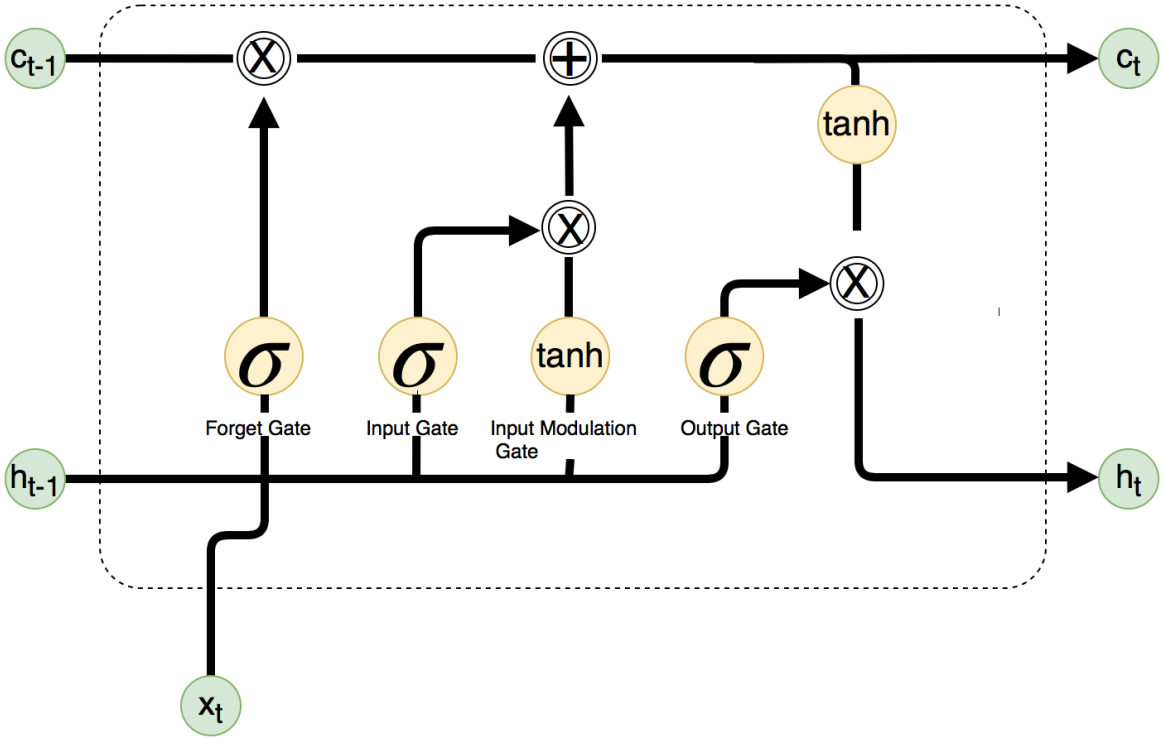



LSTM



# LSTM ячейка

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

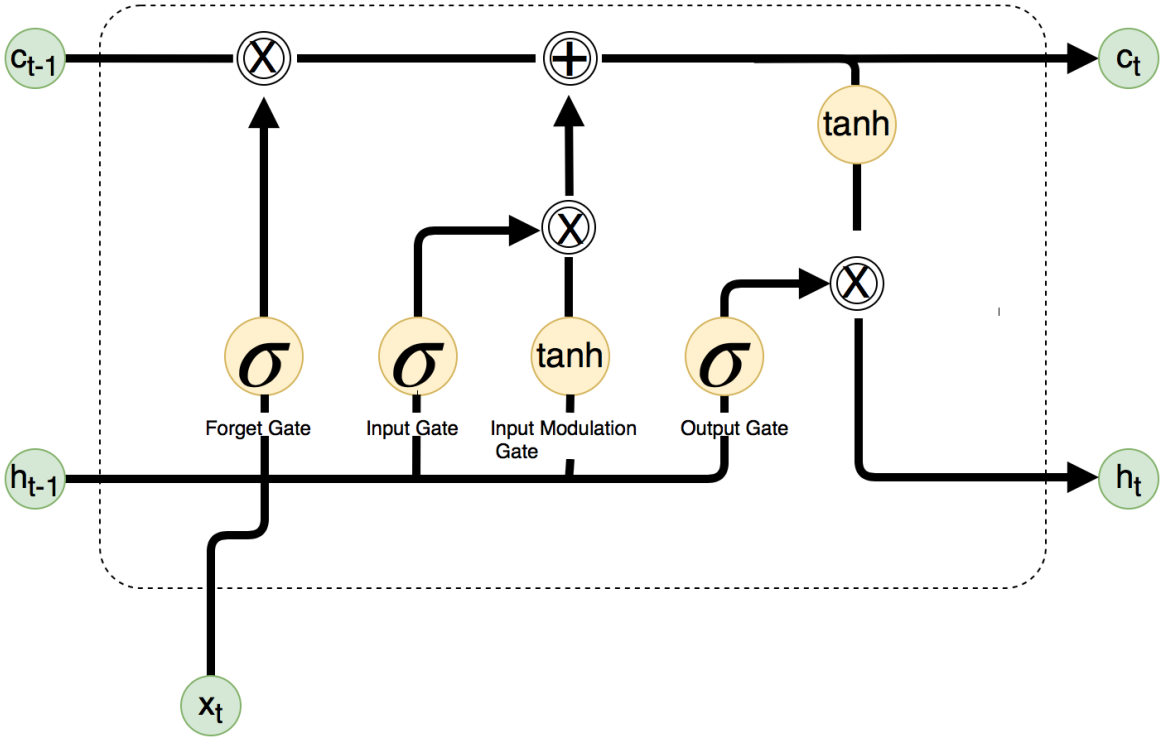



Обучается с помощью алгоритма Backpropagation

Почему решает проблему затухающих градиентов?

# LSTM ячейка

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

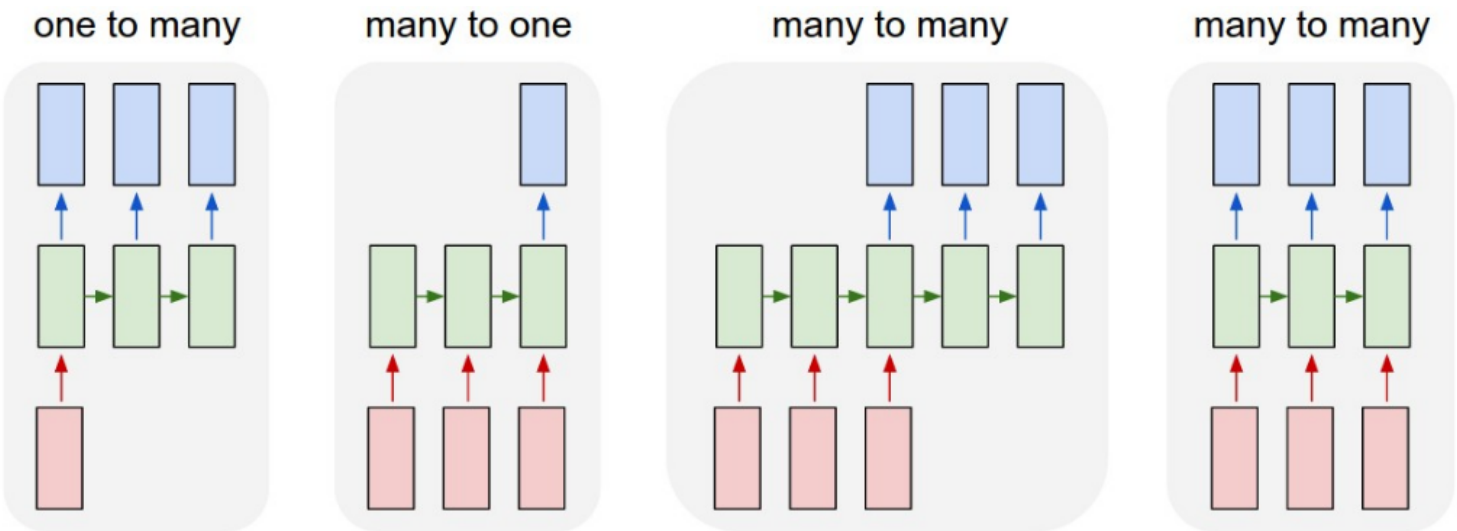



Обучается с помощью алгоритма Backpropagation

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, $f_t \approx 1$

# Разные архитектуры рекуррентных сетей




Примеры задач:

**one to many** Генерация описания изображения

**many to one** Классификация предложений

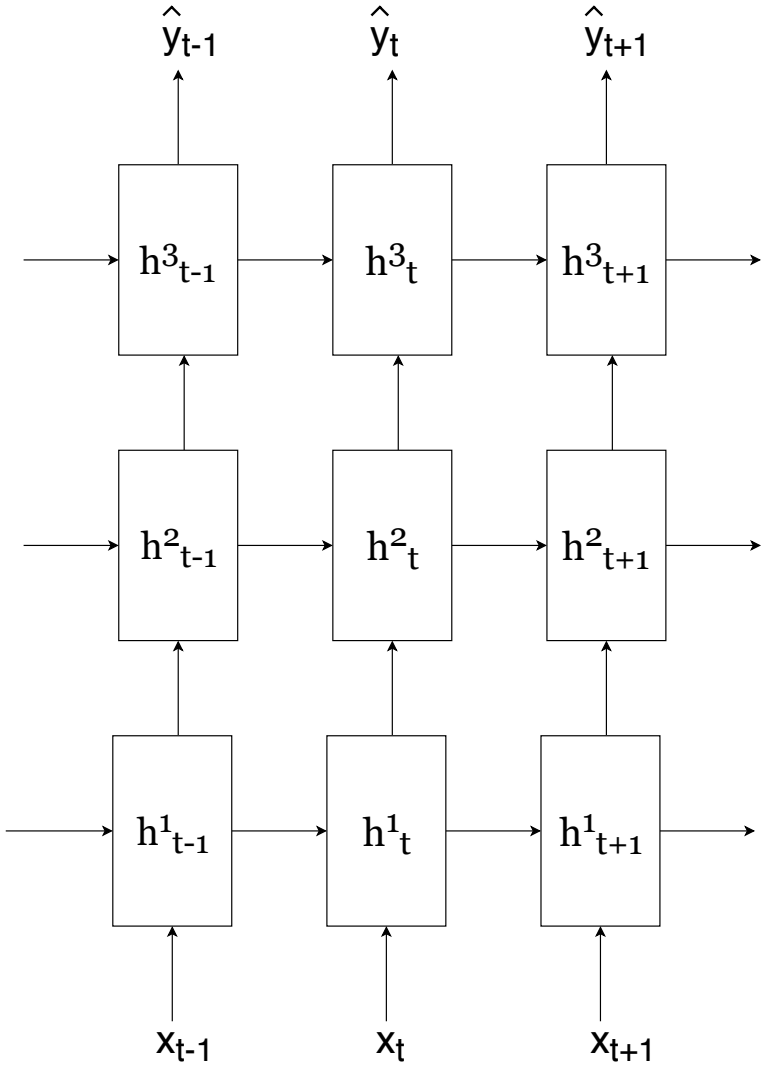
**many to many(1)** Перевод с одного языка на другой

**many to many(2)** Определение частей речи

# Глубокие рекуррентные сети (deep RNN, layers stacking)

Выходы одной рекуррентной сети подаются на вход другой:

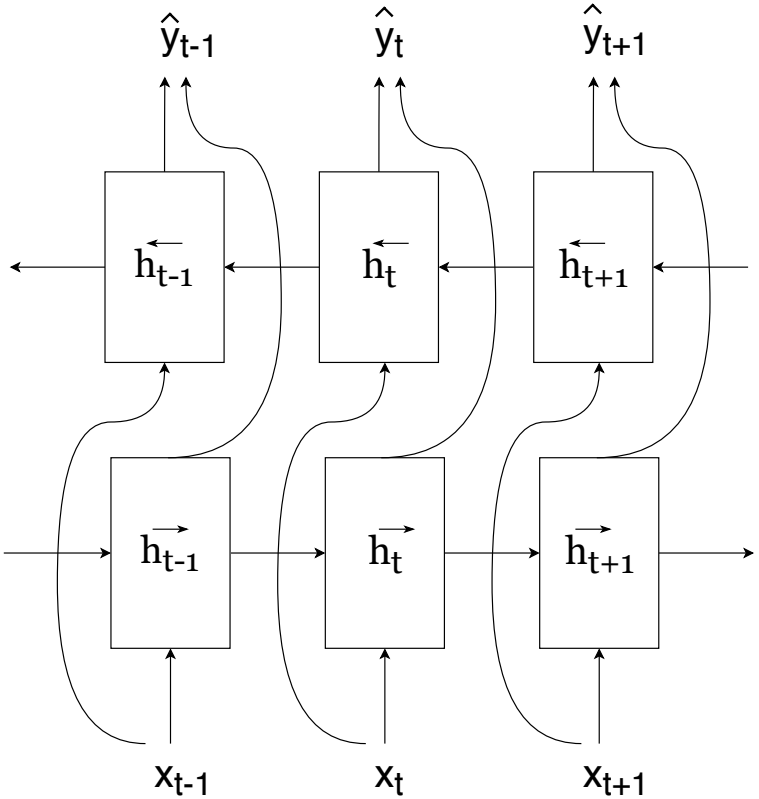
$$h_t^1, C_t^1 = LSTM(h_{t-1}^1, C_{t-1}^1, x_t)$$
$$h_t^2, C_t^2 = LSTM(h_{t-1}^2, C_{t-1}^2, h_t^1)$$
$$h_t^3, C_t^3 = LSTM(h_{t-1}^1, C_{t-1}^1, h_t^2)$$
$$y_t = g(Uh_t^2 + \hat{b})$$



# 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 часть редких слов на <UNK> токен при обучении
- На каждой итерации обучения с малой вероятностью заменять одно из 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

- RNN — Нейросетевая архитектура для работы с последовательностями
- Обучается с помощью алгоритма backpropagation
- В исходном виде RNN плохо обучается, необходимо использовать LSTM (или GRU) и gradient clipping
- С помощью разных архитектур сети можно решать разные задачи

## Базовая модель теггинга

### 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

### Базовая модель теггинга:

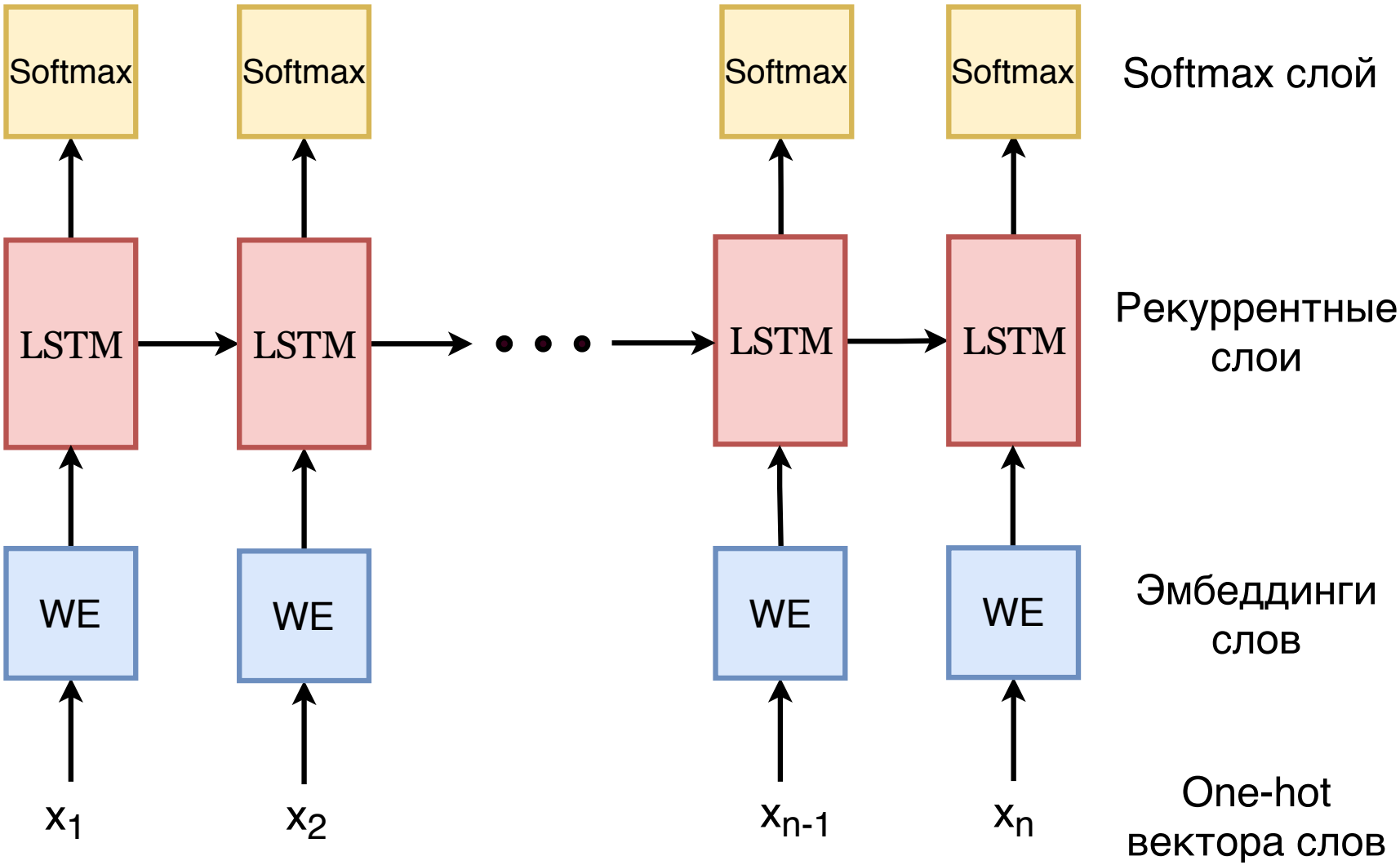
$$v_t = \text{Embedding}(t)$$

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

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

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

# Базовая модель теггинга



## Замечания о базовой модели

- Слова не приводятся к нижнему регистру
- Обычно используется 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)

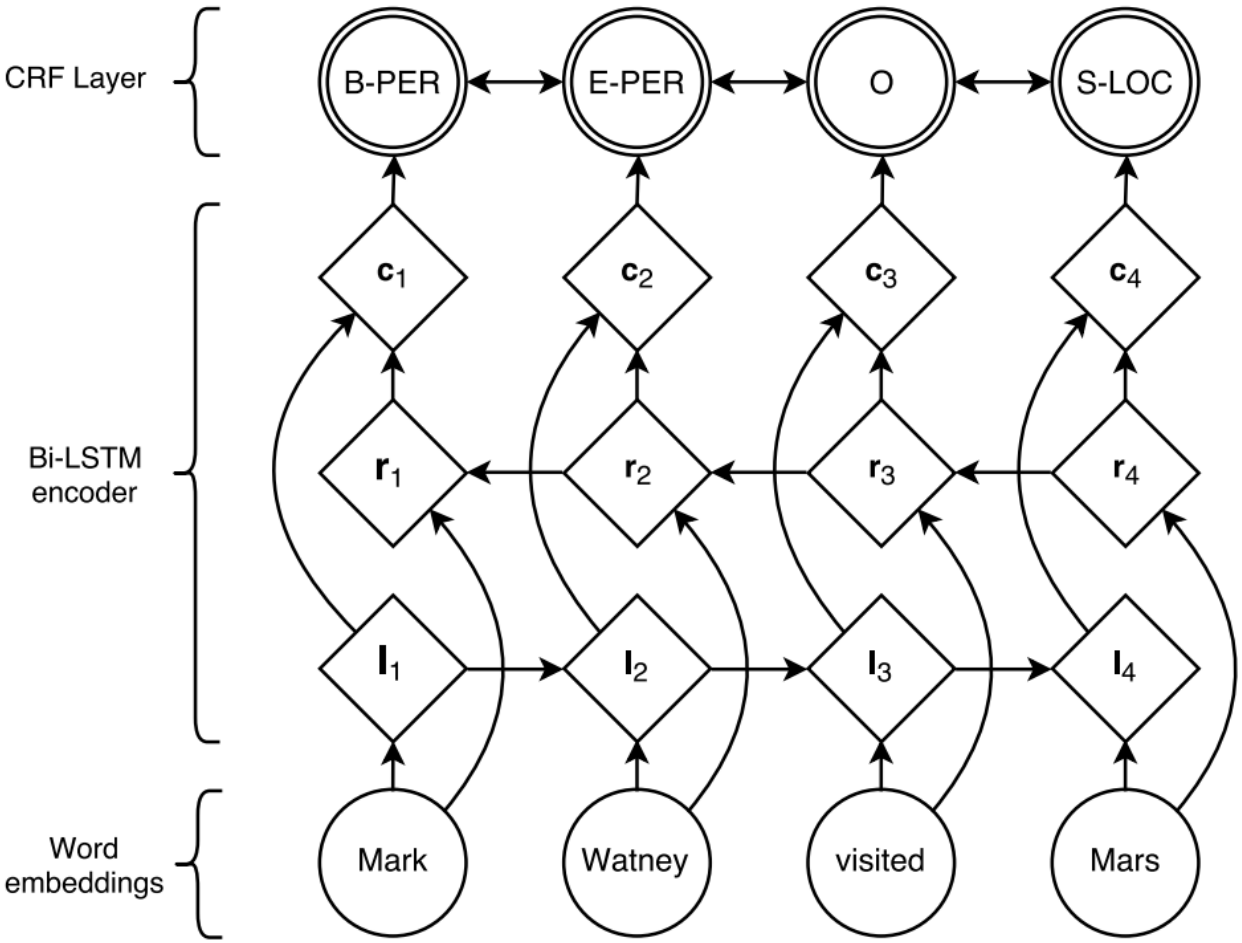
```
Facebook нашел нового финансового директора .
Финансовым директором социальной сети Facebook
назначен 39-летний Дэвид Эберсман (David Ebersman),
сообщает The Wall Street Journal.

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

- **Дополнительный CRF-слой**
- **Дополнительные представления слов**
- Предобученные глубокие сети
- **Разделение предсказания префиксов и типов**
- Semi-supervised learning
- Multitask learning

# Дополнительный CRF-слой




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

## Детали 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
```

## Почему CRF помогает модели?

Не все выходные последовательности соответствуют формату:

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

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

Таким образом, CRF — обучаемый пост-процессинг последовательности.

# Улучшение от CRF в biLSTM[1]

Результаты biLSTM + CRF превосходят остальные подходы

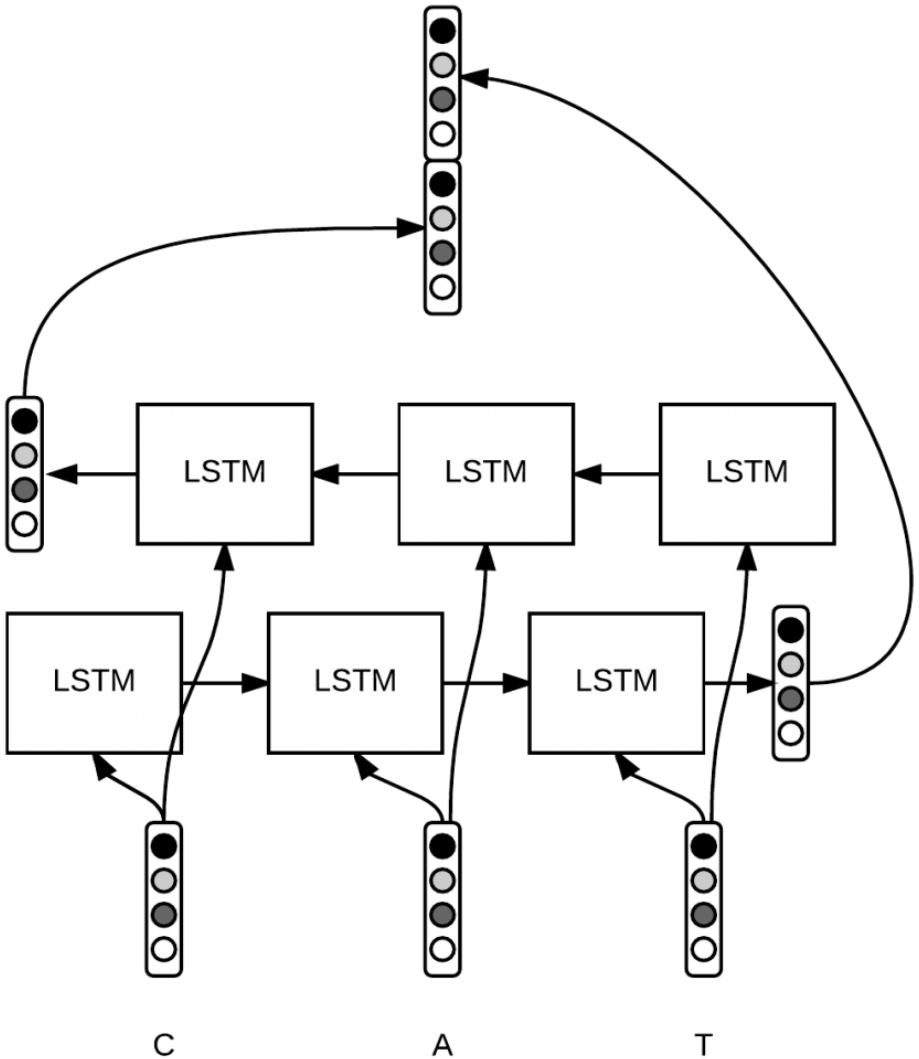
Table 2: Comparison of tagging performance on POS, chunking and NER tasks for various models.

| | | 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.

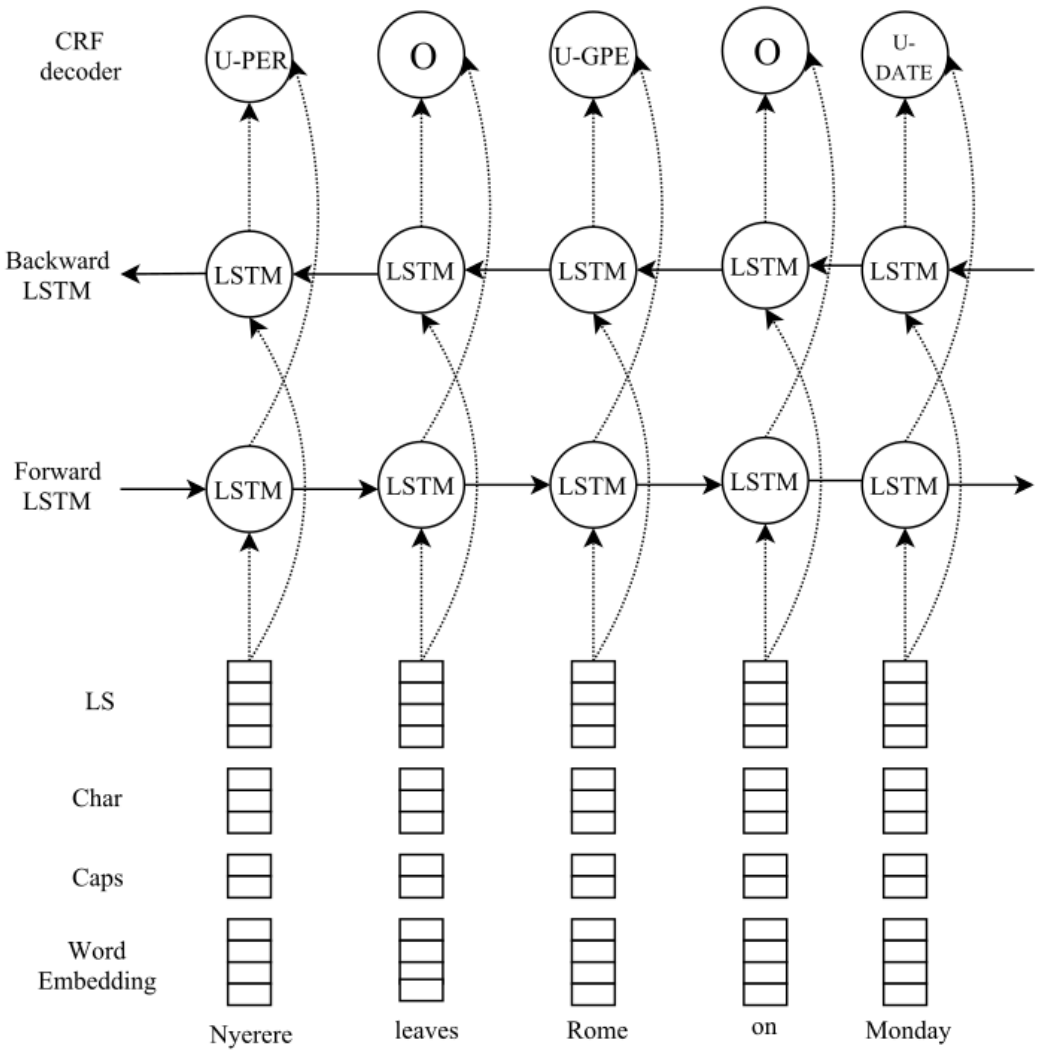
# Посимвольные представления слов




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

Улучшение работы для слов с опечатками и ошибками.

# Дополнительные представления: регистр и тип




В модели[1] используется 4 вида эмбеддингов:

- стандартные
- признаки регистра
- посимвольные
- тип

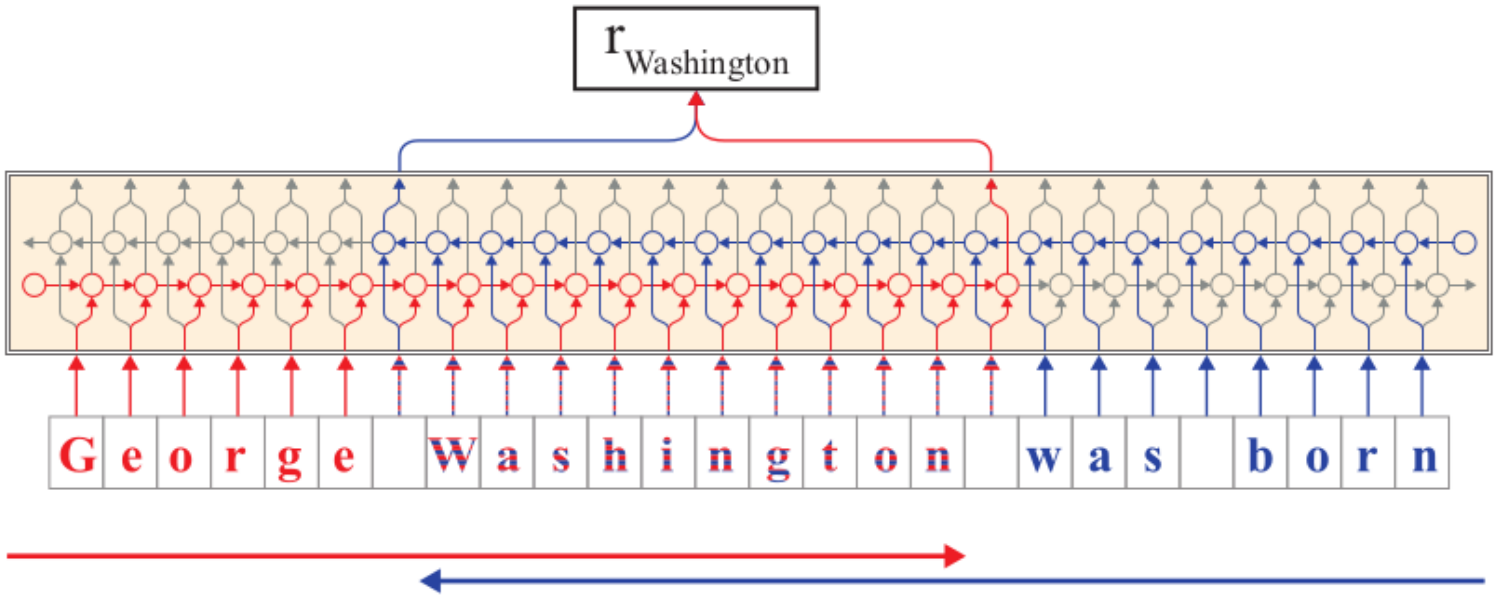
[1] Ghaddar et al (2018), Robust Lexical Features for Improved Neural Network Named-Entity Recognition

## 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.

# Контекстные посимвольные эмбеддинги

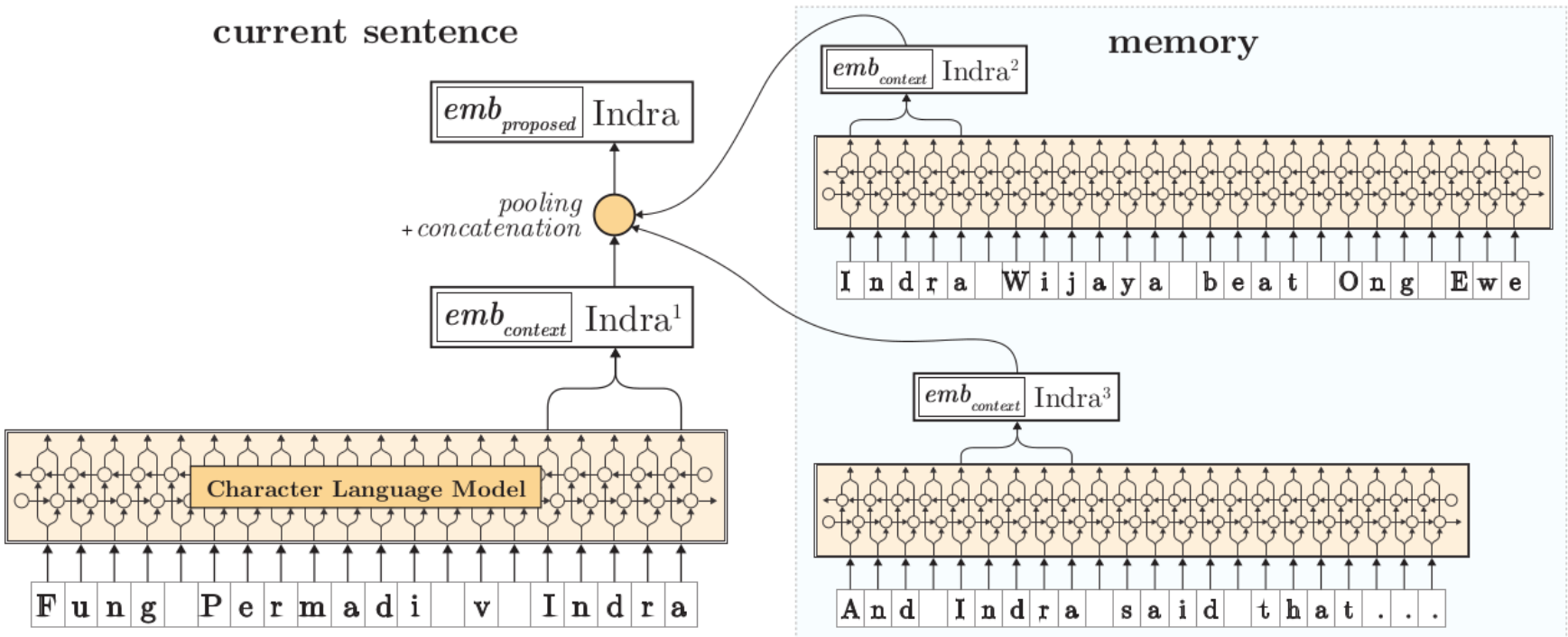



Посимвольное представление можно вытаскивать с учётом контекста. В модели[1] сеть предобучена как посимвольная языковая модель:

$$\mathcal{L} = \prod_{t=1}^{n} p(x_t | x_1^{t-1})$$

[1] Akbik et al (COLING 2018), Contextual String Embeddings for Sequence Labeling

# Глобальные контекстные посимвольные эмбеддинги




[1] Akbik et al (NAACL 2019), Pooled Contextualized Embeddings for Named Entity Recognitio

# Глобальные контекстные посимвольные эмбеддинги

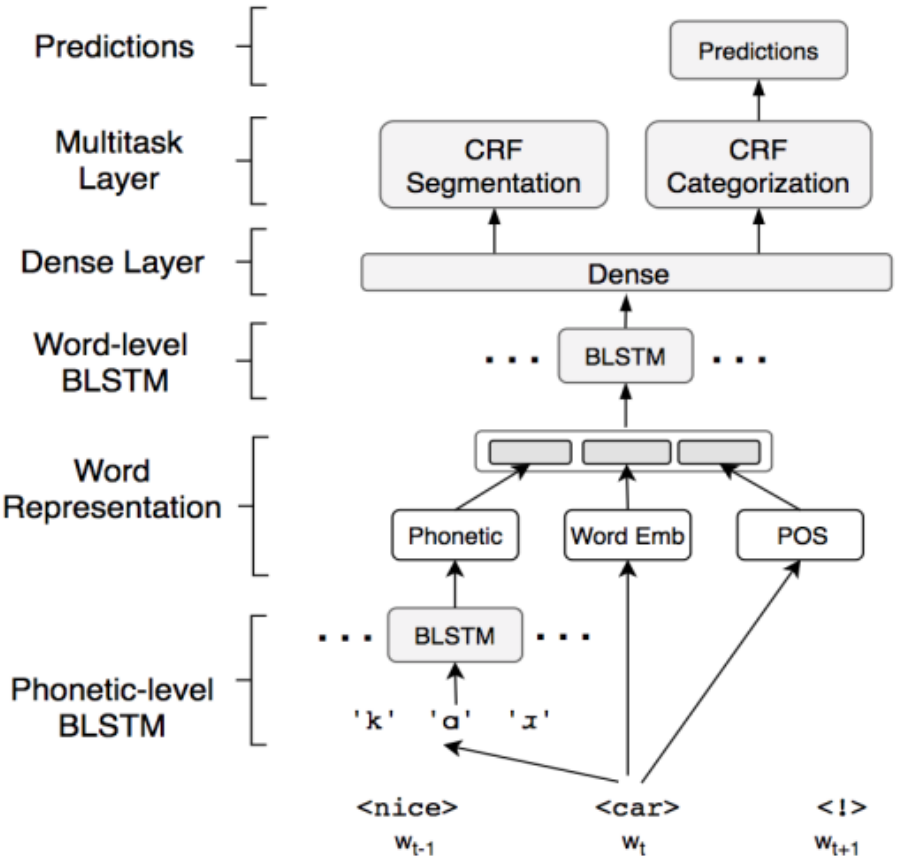
Алгоритм получения глобальных контекстных эмбеддингов:

**Algorithm 1** Compute pooled embedding

**Input:** $sentence, memory$

1: **for** $word$ in $sentence$ **do**
2: $emb_{context} \leftarrow$ embed($word$) within $sentence$
3: add $emb_{context}$ to $memory[word]$
4: $emb_{pooled} \leftarrow \text{pool}(memory[word])$
5: $word.embedding \leftarrow \text{concat}(emb_{pooled}, emb_{context})$
6: **end for**

# Разделение предсказания префикса и типа




В модели[1] предлагают разделять предсказание префиксов и типа сущности.

Также в модели используется три вида эмбеддингов:

- фонетические (charRNN по фонетической записи)
- предобученные пословные
- часть речи слова

Если типов сущностей много, может привести к улучшению.

[1] Aguilar et al (NAACL 2018), Modeling Noisiness to Recognize Named Entities using Multitask Neural Networks on Social Media

## Semi-supervised обучение

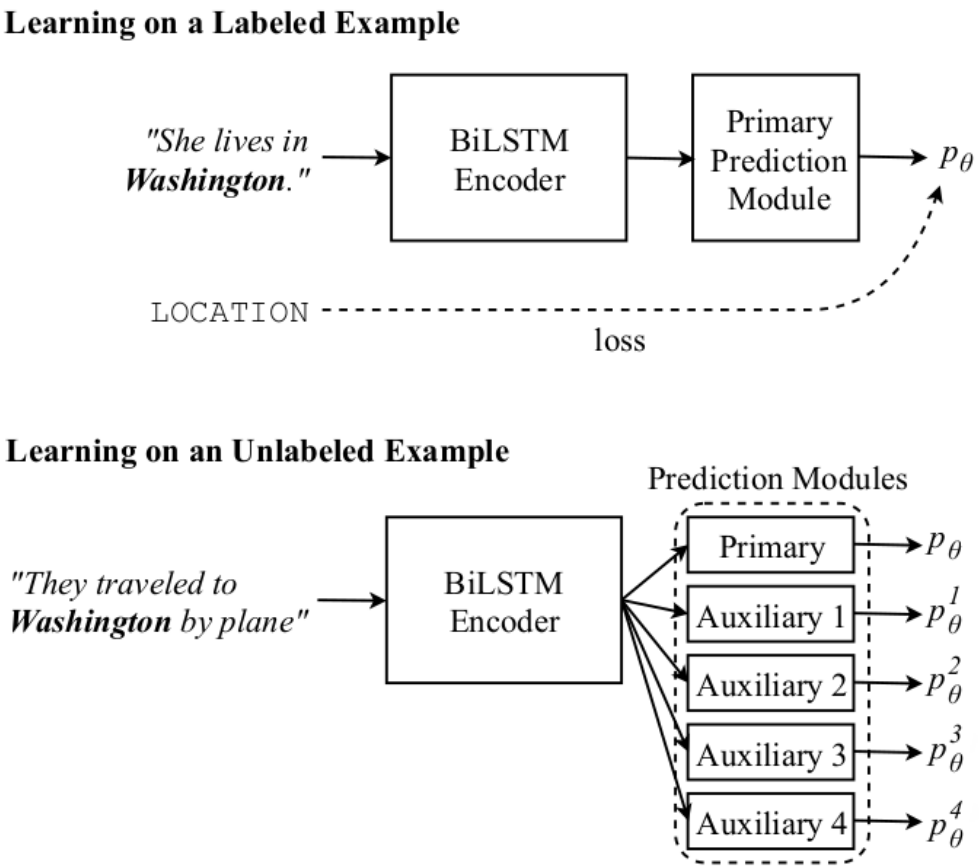
Если есть дополнительные неразмеченные данные:

1. Обучаем теггер на размеченных последовательностях
2. Применяем теггер к неразмеченным последовательностям
3. Скрывая часть слов в неразмеченной последовательности, делаем предсказания теггера похожими на полученные

$$\mathcal{L}_{semi} = \sum_{x_t \in x_{uns}} \sum_{j=1}^{k} D(p_\theta(y|x_t)|p_\theta^j(y|x_t)) \to \min_\theta$$

$x_{uns}$ — неразмеченная последовательность
$k$ — число различных последовательностей со скрытием
$D$ — функция расстояния (например, KL)
$p(y|x_t)$ — распределение полученное на шаге 2
$p^j(y|x_t)$ — распределение полученное после скрытия

# Semi-supervised обучение




[1] Clark et al (2018); Semi-Supervised Sequence Modeling with Cross-View Training

## Резюме RNN-теггинг

- biLSTM — хороший подход для теггинга
- Добавление CRF слоя почти всегда улучшает качество
- Добавление дополнительных представлений почти всегда улучшает качество

## 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
- 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 (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
- Часто более важная задача 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)

## Открытые модели NER

Подходят для русского языка:

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

Подходят для других языков:

- spacy — нейросетевые модели (свёрточные сети)