Математические методы анализа текстов

Семинар 7

Глубинные нейронные сети в обработке текстов

Мурат Апишев (great-mel@yandex.ru)

МГУ им. М. В. Ломоносова

27 апреля, 2018

Содержание занятия

- Области применения нейронных сетей
- Введение в RNN, генерация текстов с помощью LSTM
- Введение в модель CRF
- BiLSTM + CRF для задачи NER
- Нейросетевой машинный перевод, механизм attention в DL
- Введение в CNN, классификация текстов с помощью CNN
- Библиотеки для обучения нейронных сетей

Почему нейросети надо использовать

- Умеют находить сложные закономерности в пространствах огромной размерности. В ряде задач нейросетевые подходы улучшают качество в разы по сравнению с традиционными
- Найдены эффективные архитектуры для решения различных задач (LSTM/GRU, CNN)
- Современные методы глубинного обучения и регуляризации + вычисления на GPU
- Множество готовых удобных инструментов, развитое сообщество

Почему используется всё остальное

- Нейросети – тежеловесный инструмент, требующий много данных и ресурсов для обучения
- Многие задачи можно решить гораздо быстрее более простыми инструментами с небольшой потерей качества (мало данных, небольшое признаковое пространство, хорошая разделимость объектов)
- Нейросеть – «непонятный» обычным людям инструмент. Для бизнеса часто легко интерпретируемая регрессия с нормальным качеством предпочтительнее нейросети с лучшим качеством
- Хайп вокруг нейросетей существенно академический – в больших компаниях многие ключевые процессы продолжают основываться на обычном ML (и даже на эвристиках)
- Иногда наилучшее качество достигается при использовании комбинации нейросетей с другими моделями

Рекуррентные нейронные сети

- Обычные нейронные сети плохо подходят для обработки последовательностей, поскольку наблюдают только текущий элемент
- Для учёта контекста используются *рекуррентные нейронные сети* (RNN)

Примеры задач:

- Распознавание речи/музыки
- Распознавание рукописного текста
- Распознавание/генерация печатного текста
- Анализ временных рядов
- Машинный перевод

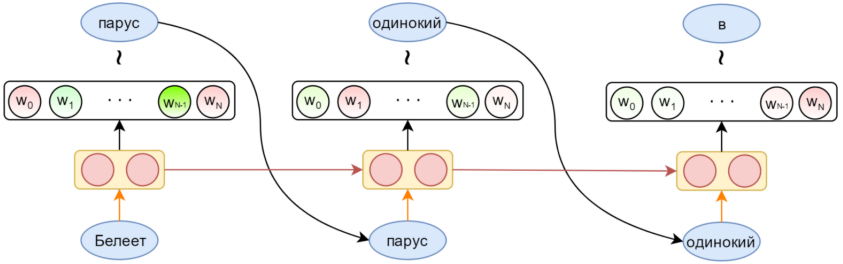
Посмотрим на примере задачи генерации стихотворений (источник)

Задача генерации текста

- В узкой тематике бот может обмануть обычного человека, но не специалиста.
- В широкой и простой тематике выявить хорошего бота можно только по шаблонам в предложениях и явному комбинированию слов:

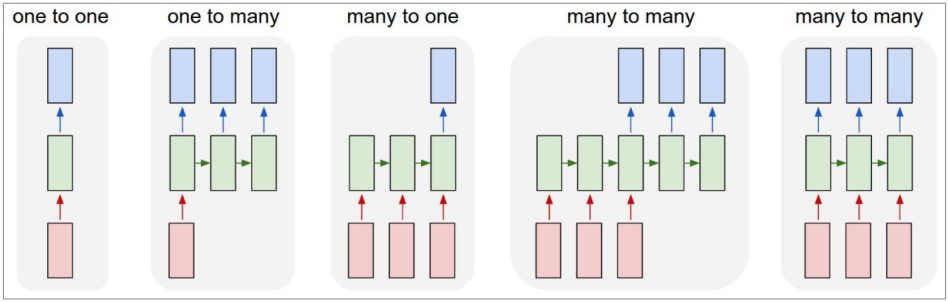
  Все ваши посты – типичное клише лживой инсинуации, которая стремится дискредитировать и осмеять всякого, кто начинает прозревать и открыто говорить о преступлениях преступного режима. Колет глаза держимордам кровавого кремлёвского упыря правда об их бесчеловечии и о фашистской сути кровавого кремлёвского режима! Интересной особенностью данного форума является то, что путинисты в основном занимаются флудом или обсуждением личностей, а топиков по существу проблем России, вроде этого, боятся как черт ладана.

Рекуррентные нейронные сети



- На входе – embeding слова (рыжая стрелка)
- На выходе обычно полносвязный слой + softmax (берём argmax или сэмплируем)
- Результаты предыдущих итераций и информация с прошлого прохода передаются дальше.

Виды RNN



Ссылка на источник картинки

Архитектура LSTM

- В реальности обычная RNN хранит информацию только о коротком контексте (затухание градиентов)
- Такого недостатка лишена LSTM – нейросетевой рекуррентный блок, состоящий из пяти элементов:
  - Основной слой (как и в обычной RNN)
  - Три сигмоидальных слоя-фильтра
  - Ячейка памяти (вектор)
- Каждый слой имеет свои обучаемые веса
- Каждый фрейм LSTM передаёт не только свои выходы, но и состояние ячейки памяти

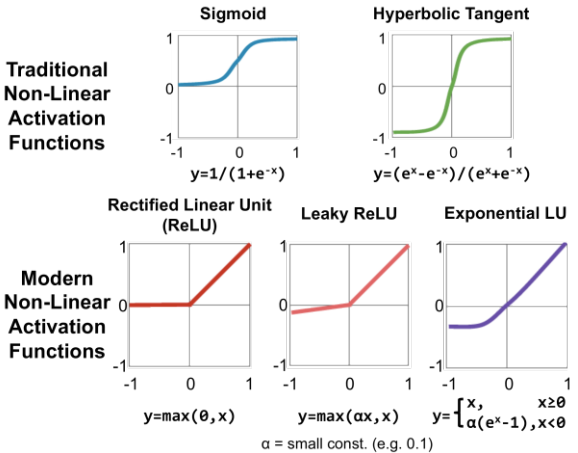
Функции активации

- Сигмоида $f(x) = \dfrac{1}{1 + e^x}$
  - $f(x) \in [0, 1] \Rightarrow$ позволяет моделировать вероятности
  - Дифференцируема и монотонна
  - Может привести к «параличу сети» из-за слабого измнения y при измнении x на отдалённом расстоянии от 0
  - Обобщается функцией softmax
- Гиперболический тангенс $f(x) = \tanh(x)$
  - Все свойства сигмоилы
  - Значение $xf(x)$ вседа неотрицательно
  - Обычно используется для бинарной классификации
- ReLU $f(x) = \max(x, 0)$
  - Имеет монотонную производную (в отличие от предыдущих)
  - Игнорирует отрицательные сигналы

Функции активации

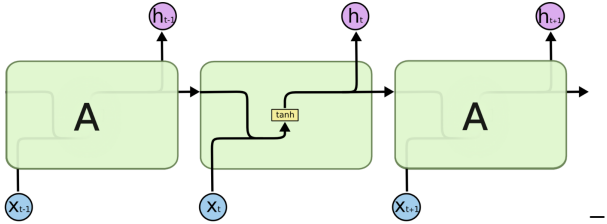
- Leaky ReLU $f(x) = \begin{cases} x, & x => 0 \\ \alpha x, & x < 0 \end{cases}$

- Решает проблему игнорирования отрицательного сигнала
- $f(x) \in [-\infty, +\infty]$
- Обычно $\alpha = 0.01$
- Приводит к разреживанию весов

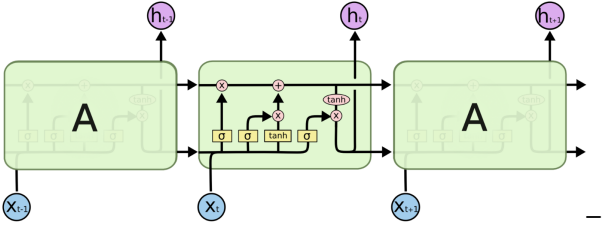


Ссылка на источник картинки

Архитектура LSTM



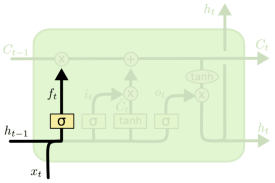
– RNN: 1 слой



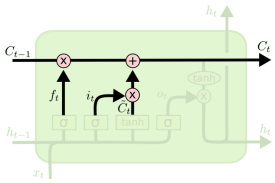
– LSTM: 4 слоя

Ссылка на источник картинок

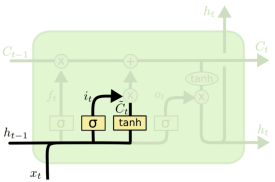
Архитектура LSTM



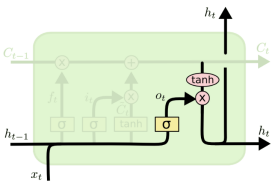
1) $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$



3) $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$



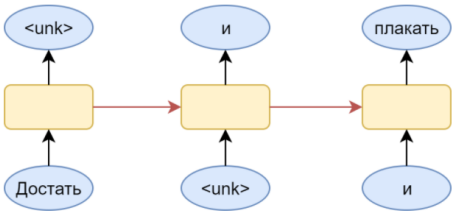
2) $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$



4) $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
$h_t = o_t * \tanh(C_t)$

Генерация стихов: обработка отсутствующих слов

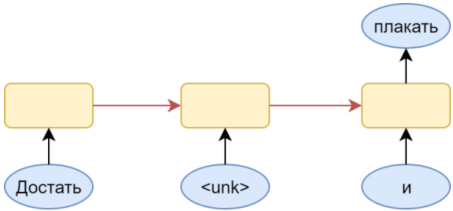
- Словарь может состоять из миллионов слов, но часто его приходится сильно фильтровать
- Вместо отсутствующего слова берём <unk>:



- В модели предсказания слова по предыдущему выдаваемое распределение на словах сместится в пользу <unk>
- **Выход:** можно сэмплировать без него, но получается криво

Обработка отсутствующих слов

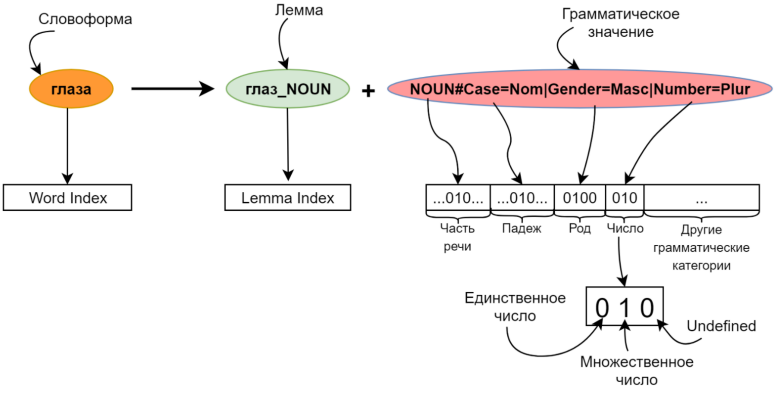
- Альтернатива – предсказывать слова по цепочке предыдущих:



- Из обучающей выборки придётся нарезать всевозможные цепочки, что приведёт к её существенному увеличению
- Зато можно выкинуть все цепочки, заканчивающиеся неизвестным словом.

Доработка входного слоя

Необходимо сократить размерность выходного слоя

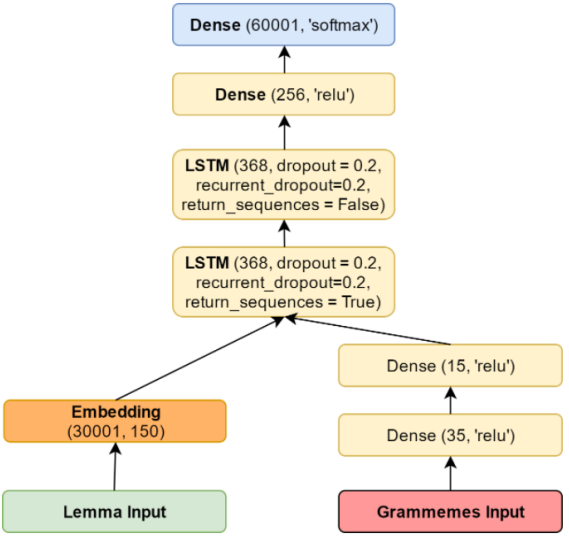


Можно использовать уже предобученные эмбеддинги для лемм (например, от RusVectores).

Доработка выходного слоя

- Вместо индекса слова можно предсказывать по-отдельности лемму и грамматическое значение
- **Проблема:** у сэмплированной леммы может не оказаться нужного грамматического значения
- **Варианты решения:**
  1. выбирать наиболее вероятную пару «лемма + грамматическое значение» из существующих
  2. выбирать наиболее вероятное грамматическое значение среди возможных для сэмплированной леммы

Итоговая архитектура сети (keras)

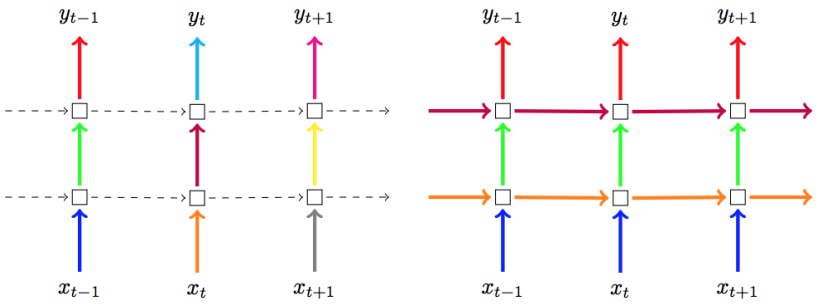


Dropout

- Нейросети имеют огромное количество настраиваемых параметров, что приводит к переробучению
- Одной из техник регуляризации для борьбы с этим – dropout:
  - Выбирается очередной объект (или батча) при обучении сети с помощью SGD
  - Выкидывается каждый узел с вероятностью $p > 0$
  - По разреженной сети делается шаг back propagation и обновление весов
  - При применении сети выход каждого узла домножается на $(1 - p)$
- Нейросеть, обученную с дропаутом, можно рассматривать как результат усреднения 2^N архитектур сетей, где N – число всех узлов сети

Рекуррентный dropout

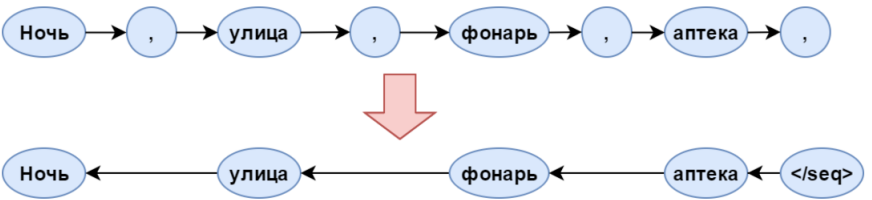
- Обычный дропаут применяется к весам, связанным с входными данными
- Можно применять дропаут и к весам, связанным с выходами предыдущего фрейма h_{t-1}



Ссылка на источник картинки

Данные

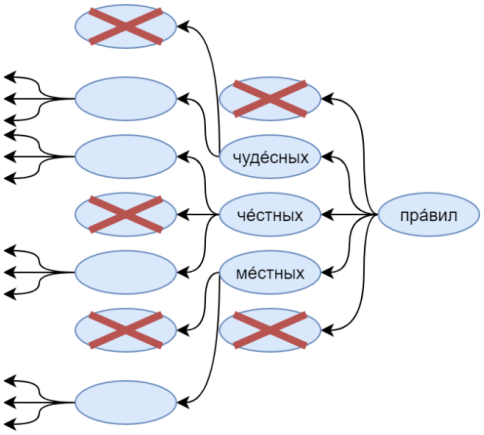
- http://stihi.ru/ + морфологическая разметка.
- Объект выборки – строка стихотворения.
- В конец каждой строки добавлялся завершающий символ.
- Строки инвертировались для упрощения рифмовки при генерации.
- Из выборки удалены знаки препинания (сеть сильно обучается на запятых и многоточиях).



Правила фильтрации

- У нас есть модель-генератор, нужно фильтровать слова так, чтобы получались именно стихотворения
- Метрические правила определяют последовательность ударных и безударных слогов в строке
- Правила рифмы допускают только словоформы, которые корректно рифмуются (слова с одной леммой рифмовать запрещено)
- Ударения были получены путём обучения классификатора на словаре, рифмы – эвристическими правилами

Лучевой поиск (beam search)



- В результате работы фильтров могло не остаться ни одного слова
- Для борьбы с этим на каждом этапе применения фильтров берём не лучшего, а несколько лучших кандидатов, так, чтобы на каждом шаге их было N штук

Пример результата

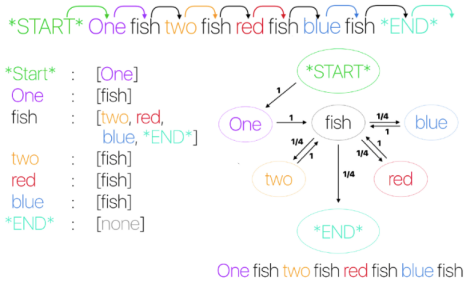
Так толку мне теперь грустить

Что будет это прожито

Не суждено кружить в пути

Почувствовав боль бомжика

Как генерировать без нейросети

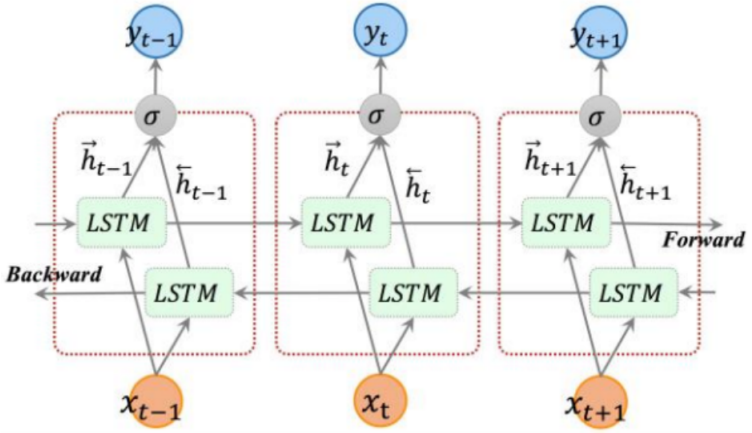


- Можно генерировать с помощью обычной HMM
- Проблема в сложности учёта длинного (> 5 слов) контекста
- HMM может генерировать «художественные» тексты схоже с нейросетью
- Структурированную информацию нейросеть генерирует существенно лучше (например, код)

Ссылка на источник картинки

Архитектура BiLSTM

Обычная LSTM учитывает только прошлый контекст, двунаправленная учитывает и будущий:



Ссылка на источник картинки

Conditional Random Fields

- Марковское случайное поле: неориентированный граф $G = (V, E)$ и множество функций $\{\phi_k\}$
- V – множество случайных переменных-вершин, E – множество рёбер, отражающих попарные зависимости между переменными
- Содержимое $\{\phi_k\}$ – *потенциальные функции*, по одной на каждую клику G (полный подграф), область значений $\phi_k \subseteq \mathbb{R}_+$
- Вершины, не являющиеся смежными, должны соответствовать условно независимым случайным величинам
- Группа смежных вершин образует клику, набор состояний вершин является аргументом соответствующей потенциальной функции

Ссылка на источник

Conditional Random Fields

- $V = X \cup Y$, X – наблюдаемые переменные, Y – предсказываемые
- CRF – дискриминативная модель: в отличие от HMM она строит не совместное распределени $p(y, x)$, а условное $p(y \mid x)$, которое обычно и нужно в задачах ML
- В *линейном условном случайном поле* потенциальная функция имеет вид

$$\phi_k(x_k) = \exp(\sum_s \lambda_s f_s(y_t, y_{t-1}, x_t)),$$

где s – индекс признаковой функции f_s, $\lambda_s \in \mathbb{R}$

- Строим распределение

$$p(y \mid x_t) = \frac{1}{Z(x)} \prod_{k \in K} \exp(\sum_s \lambda_s f_s(y_t, y_{t-1}, x_t)),$$

где t – индекс очередного элемента последовательности, K – множество клик G. $Z(x)$ получается суммированием числителя по всем y

Conditional Random Fields

- CRF лишена Label Bias Problem – ситуации в MEMM, когда наибольшее предпочтение получают состояния с меньшим числом переходов в другие (подробнее тут)
- Качество сильно зависит от выбора признаков f_s
- Один из лучших методов для NER и POS-теггинга
- Очень долго обучается
- Хорошо работает в связке с рекуррентными нейросетями, моделирует совместное распределение на всей последовательности выходов сети одновременно

BiLSTM + CRF для задачи NER

Решается задача выявления именованных сущностей (6 категорий)

Основные шаги:

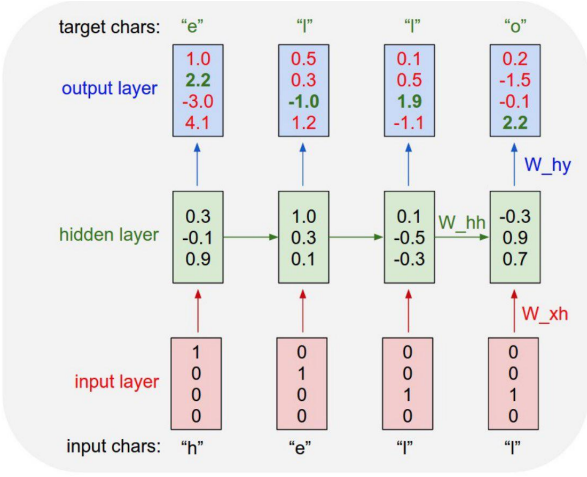
- Получить предобученные эмбеддинги слов коллекции
- Обучить символьные эмбеддинги
- Составить для каждого слова синтаксические признаки (POS-тег, роль в предложении и т.п.)
- Объединить всё это и подать на вход BiLSTM
- Выходы h_t для всех слов предложения подавать на вход CRF, которая будет предсказывать NER-тег в BIO2-нотации

Ссылка на статью

Символьные эмбеддинги

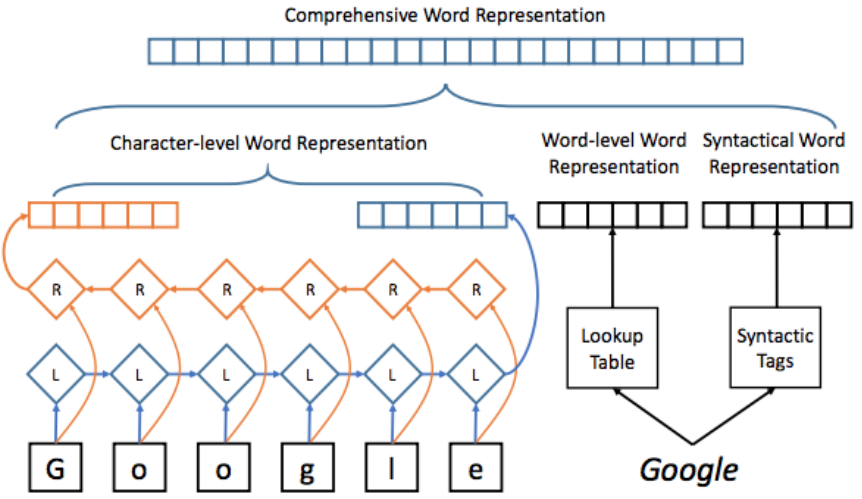
- Символьные эмбеддинги полезны при обработке user-generated текстов, поскольку люди часто
  - используют сокращения, аббревиатуры и сленг
  - ошибаются, делают опечатки
- В обоих случаях использование эмбеддингов слов приведёт к большому число OOV-слов
- Символьные эмбеддинги можно получить для данной задачи с помощью нейросетей (CNN или RNN)
- RNN обычно лучше CNN, поскольку представление символов обучается на их последовательности в слове, а CNN больше подходят для работы с инвариантными признаками, чем с последовательностями

Символьные эмбеддинги

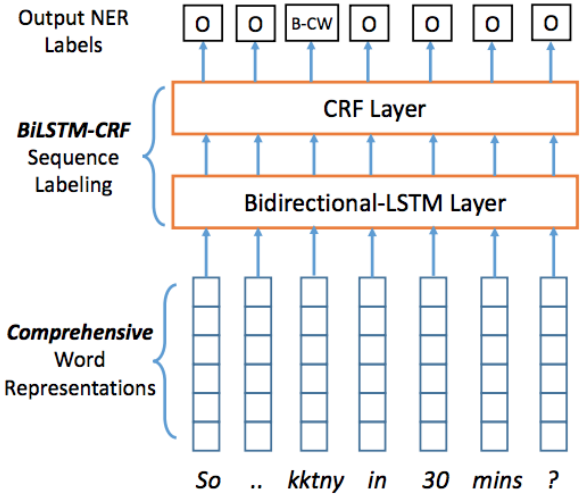


Ссылка на источник картинки

Комплексное представление слова



Общая схема модели



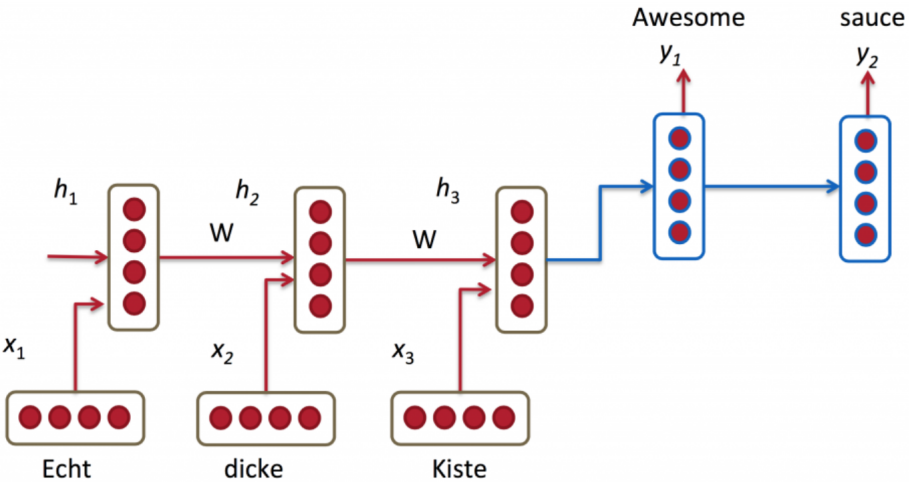
Нейросетевой машинный перевод

- Традиционные системы перевода основаны на создании сложных признаков по текстовым статистикам. Такие системы сложны с инженерной точки зрения
- Нейросетевой перевод устроен проще и работает существенно лучше
- Предложение на языке A целиком переводится в сжатое векторное представление, которое затем переводится в предложение на языке B (модель Seq2seq)
- Кодирование и декодирование производится с помощью рекуррентных сетей

Ссылка на статью

Нейросетевой машинный перевод

Вектор h_3 инкапсулирует в себе всю информацию о предложении



Проблема длинных предложений

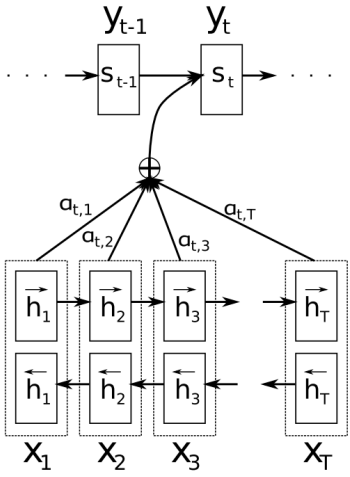
- Допустим, что мы переводим предложение из 50 слов
- Первое слово предложения на английском наверняка сильно связано с первым словом предложения на немецком
- Но эта информация была добавлена в вектор 50 шагов назад
- LSTM в теории должны улавливать такие длинные зависимости, но на практике это работает не очень хорошо
- Можно использовать хак: подавать кодировщику предложение в обратном порядке.
- Но это работает не для всех языков. Например, в предложении на японском последнее слово может очень сильно влиять на первое слово перевода на английский

Attention в глубинном обучении

- Механизм attention у людей – возможность сосредоточиться на самом интересном с наибольшим вниманием, на менее интересном – с меньшим
- В ML – выделение части данных для более детальной обработки
- Важно для экономии вычислительных ресурсов
- В примере с переводом attention приводит к тому, что нам больше не нужно кодировать всё предложение в сжатый вектор
- Вместо этого позволим декодеру смотреть на эмбеддинги всех слов последовательности и с помощью весов самому выбирать, какие из них важны для генерации очередного слова предложения-перевода

Перевод с attention

- Кодирование производится с помощью BiLSTM (не принципиально)
- Веса a_i обычно суммируются в 1
- Визуализируя веса можно понимать стратегию перевода (например, для перевода немецкого в английский она будет примерно последовательной)



Ссылка на источник картинки

Перевод с attention

- При использовании BiLSTM каждый вектор h_j хранит информацию о всей последовательности, но в наибольшей степени о j-м слове и его соседях
- Далее при текущем выходном слове y_{t-1} (с вектором s_{t-1}) для каждого вектора входного слова h_j считается a_{tj} – вклад в генерацию следующего выходного слова (attention):

$$a_{tj} = \frac{\exp(e_{tj})}{\sum_k \exp(e_{tk})},$$

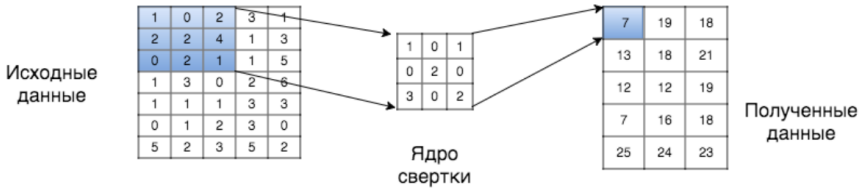
  где $e_{tj} = a(s_{t-1}, h_j)$ – модель выравнивания
- Модель выравнивания предсказывает то, насколько хорошо соотносятся входное слово в позиции j и выходное в позиции t (может быть обучаемой однослойной сетью или просто скалярным произведением)

Свёрточные нейронные сети

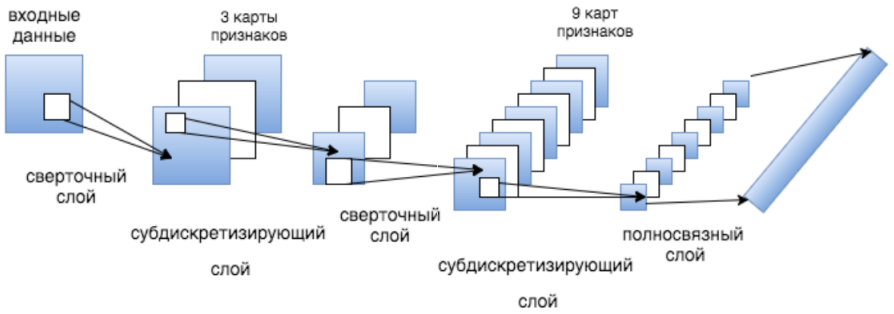
- Архитектура нейронных сетей, заключающаяся в чередовании слоёв трёх типов: свёрточных, пулинговых и полносвязных (+ нелинейные активации)
- Один из лучших алгоритмов по распознаванию и классификации изображений, также используется для обработки текстов
- Имеет небольшое количество настраиваемых весов (по сравнению с полносвязной сетью)
- Относительная устойчивость к повороту и сдвигу распознаваемого изображения
- Имеет множество настраиваемых параметров, для некоторых задач есть рекомендации, но общего представления о том, как правильно их подбирать, нет

Ссылка на источник дальнейшего изложения

Слои свёртки и пулинга



Общий вид CNN

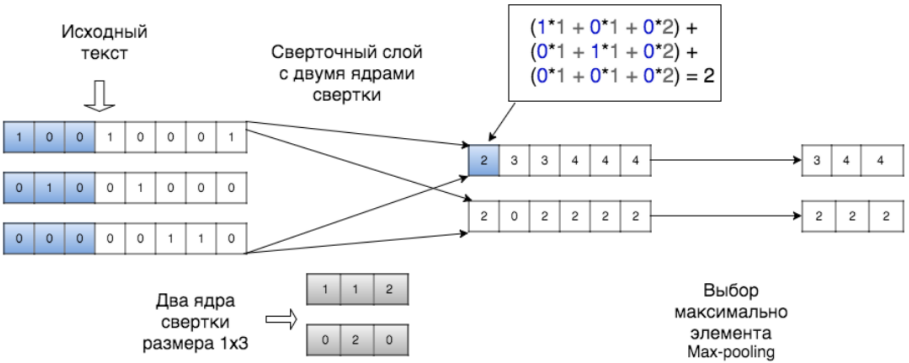


- Применяя свёртки, получаем карты признаков (используем padding)
- Пропускаем через нелинейное преобразование (например, ReLU)
- Периодически вставляем слои пулинга
- В конце полносвязные слои и softmax

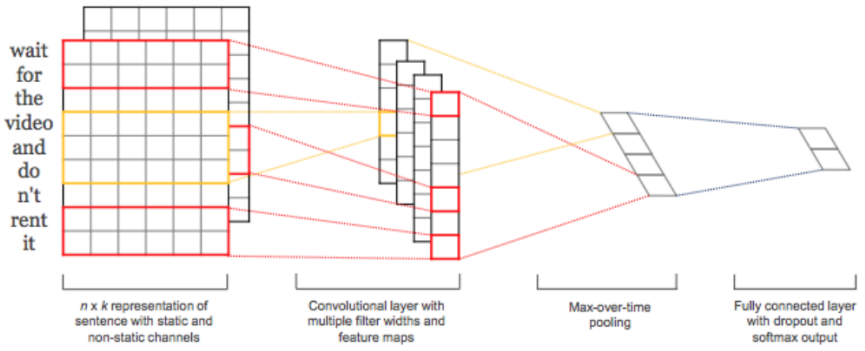
CNN для текстов

- Решаем задачу классификации текстов
- Идея в том, чтобы каким-то образом закодировать символы/слова текста и выделить карты признаков
- Дальше всё работает как обычно
- Для символов можно использовать one-hot кодировку:
  1. пусть имеем m уникальных символов
  2. пусть ℓ – достаточно большое число символов текста, по которым можно предсказывать его класс
  3. составим матрицу $m \times \ell$
  4. нарежем её на строки и подадим сети в качестве карт признаков
- Для слов – эмбеддинги word2vec/GloVe, из которых составляется входная матрица

CNN на символах



CNN на словах



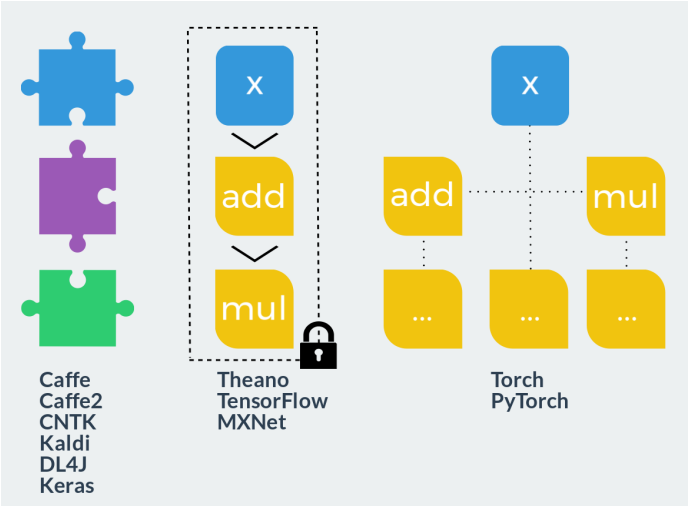
Длина предложения фиксированная, длинные обрезаются, для коротких используется padding

Классификация библиотек для обучения нейросетей

1. **С фиксированными модулями:** сборка сети из готовых блоков, в которые уже зашиты forward-backward проходы:
   - Быстрая скорость разработки
   - Разработка только в рамках имеющихся элементов
2. **Со статическим графом вычислений:** на этапе описания можно создать граф произвольной сложности, но после компиляции его можно только запустить в прямом или обратном порядке:
   - Гибкость разработки
   - Невозможность лёгкой модернизации готового графа
3. **С динамическим графом вычислений:** граф строится динамически при каждом прямом проходе
   - Гибкость разработки и отладки
   - Более сложный процесс разработки

Ссылка на источник

Классификация библиотек



Ссылка на хороший сравнительный обзор библиотек

Примеры кода

- Описываем двуслойную нейронную сеть, функционал MSE
- Полносвязные слои, функция активации ReLU
- Обучаемся на случайных данных
- Параметры модели:
  - размер батча = 32
  - размерность входа/выхода = 500
  - размерность скрытого слоя = 50
- Примеры кода на PyTorch, TensorFlow и Keras
- Keras – высокоуровневая обёртка на TensorFlow или Theano
- Ссылка на источник примеров

Пример кода на Keras

```
1  import keras
2  import numpy as np
3  from keras.models import Sequential
4  from keras.layers.core import Dense, Activation
5  from keras.optimizers import SGD
6
7  x, y = np.random.randn(32, 500), np.random.randn(32, 500)
8
9  model = Sequential()
10 model.add(Dense(input_dim=500, output_dim=50))
11 model.add(Activation('relu'))
12 model.add(Dense(input_dim=50, output_dim=500))
13
14 optimizer = SGD(lr=1e0)
15 model.compile(loss='mean_squared_error', optimizer=optimizer)
16
17 model.fit(x, y, epochs=50, batch_size=64, verbose=0)
```

Пример кода на TensorFlow

```
1  import numpy as np
2  import tensorflow as tf
3
4  x = tf.placeholder(tf.float32, shape=(32, 500))
5  y = tf.placeholder(tf.float32, shape=(32, 500))
6  w1 = tf.placeholer(tf.float32, shape=(500, 50))
7  w2 = tf.placeholder(tf.float32, shape=(50, 500))
8
9  # Forward pass
10 h = tf.maximum(tf.matmul(x, w1), 0)
11 y_pred = tf.matmul(h, w2)
12
13 loss = tf.losses.mean_squared_error(y_pred, y)
14 grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

Пример кода на TensorFlow

```
with tf.Session() as sess:
  values = {x: np.random.randn(32, 500),
            w1: np.random.randn(500, 50),
            w2: np.random.randn(50, 500),
            y: np.random.randn(32, 500),}

  # Train the network
  learning_rate = 1e-5
  for t in range(20):
    out = sess.run([loss, grad_w1, grad_w2], feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
    values[w1] -= learning_rate * grad_w1_val
    values[w2] -= learning_rate * grad_w2_val
```

Пример кода на PyTorch

```
import torch

dtype = torch.FloatTensor

x = torch.randn(32, 500).type(dtype)
y = torch.randn(32, 500).type(dtype)
w1 = torch.randn(500, 50).type(dtype)
w2 = torch.randn(50, 500).type(dtype)

learning_rate = 1e-6
for t in range(250):
  # Forward pass: Compute predictions and loss
  h = x.mm(w1)
  h_relu = h.clamp(min=0)
  y_pred = h_relu.mm(w2)
  loss = (y_pred - y).pow(2).sum()
```

Пример кода на PyTorch

```
#for t in range(250):

  # Backward pass: Compute gradients
  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h_relu.t().mm(grad_y_pred)
  grad_h_relu = grad_y_pred.mm(w2.t())
  grad_h = grad_h_relu.clone()
  grad_h[h < 0] = 0
  grad_w1 = x.t().mm(grad_h)

  # Gradient descent step on weights
  w1 -= learning_rate * grad_w1
  w2 -= learning_rate * grad_w2
```

Итоги занятия

- Нейросети – крутой многоцелевой и гибкий инструмент для решения самых разных задач в NLP
- Обучение нейросетей – сложный процесс как с точки зрения вычислений, так и с точки зрения выбора архитектуры, настройки, регуляризации и т.п.
- Большую роль играет выбор признакового пространства входных данных, внешняя информация (лигнвистические словари, предобученные эмбеддинги и т.п.)
- Рекуррентные нейросети часто работают лучше в связке с CRF (возможны и иные комбинации, многие, наверняка, просто ещё не открыты)
- Выбор конкретной библиотеки для обучения определяется задачей, железом, навыком

Успехов!