Тренды в анализе данных. Статья 4

О цикле

Данный цикл статей основан на семинарских занятиях открытого курса Data Mining in Action по направлению “тренды в анализе данных”. На семинарах (и в статьях по ним) мы будем говорить о последних достижениях в области data science.
Общую лекцию для всего потока можно посмотреть здесь, а презентацию к ней скачать здесь.

Сегодня в статье:

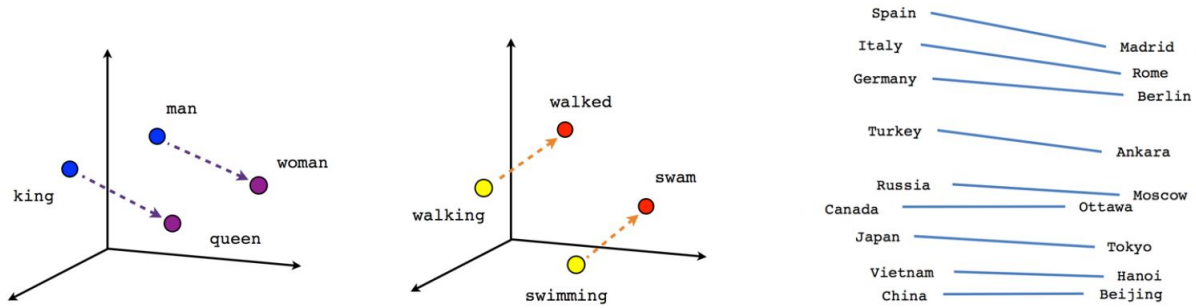
1. Embeddings
2. Языковые модели
3. ELMo

Embeddings

Embeddings - это модель векторного представления слов в связи с некоторым контекстом. Есть две основные методики составления этих векторов:

- CBOW: мы пытаемся по контексту предсказать слово, которое находится в этом контексте;
- Skip-gram: зная одно слово, мы пробуем предсказать его контекст.

Что интересного возникает при таком подходе? Оказывается, что полученные вектора будут обладать интересными свойствами. Известные примеры:



Male-Female | Verb tense | Country-Capital

Так, например, мы можем получить вектор, кодирующий пол:

$$vector_{king} - vector_{queen} = vector_{man} - vector_{woman}$$

Иными словами, для векторов слов работает некоторая арифметика, и в векторном пространстве, которое было создано этим множеством векторов, соблюдаются некоторые регулярности.
Это основной подход, откуда можно взять векторное представление слов для любой NLP задачи: мы тренируем embeddings на большом неразмеченном корпусе, а потом вставляем их в свою модель (перед этим их можно дообучить).
Другой подход к обучению embeddings: инициализируем все вектора какими-то случайными значениями и учим нейросеть с помощью метода обратного распространения ошибки. То есть, учим вектора на ходу вместе с задачей. В таком случае embeddings получаются осмысленными и будут работать конкретно на задачу - но векторная арифметика, скорее всего, работать не будет.

Языковая модель

Языковая модель - это вероятностная модель языка. На вход подается последовательность слов, и задача - предсказать вероятность следующего. Где это может быть полезно? Приведем несколько примеров:
Машинный перевод. В этой задаче можно использовать априорные знания о том, как вообще распределены слова в языке, чтобы выполнять задачу перевода более качественно. Например, вы можете сказать, что вероятность фразы “strong tea” намного больше, чем вероятность “powerful tea”.
Для этого сначала учат модель предсказывать вероятность следующего слова. Например, сначала мы можем сделать машинный перевод и получить какие-то вероятности для своих слов, а потом домножить полученные вероятности на вероятности, которые мы получим с помощью языковой модели. То есть, наша модель перевела что-то, а после этого вы оценили, насколько хорошо это звучит с точки зрения языка.
Speech recognition: в этой задаче очень важно иметь хорошую языковую модель. Хороший пример:

P(speech recognition) > P(speech wreck ignition)

Звучат последовательности похоже, однако смысл разный. И хорошая языковая модель знает, что первая фраза - гораздо более вероятное словосочетание, чем второе. Поэтому она может улучшить предсказание вашей speech-to-text модели.

Важно: языковая модель - это априорное знание о языке. При этом, в отличие от embeddings, они учитывают порядок.

Традиционные языковые модели (n-grams)

Если говорить про языковые модели, которые не основаны на нейронных сетях, то нужно помнить, что они просто считают статистику встречаемости слов друг с другом. Можем построить, например, биграммную модель: имея два слова (токена), нужно предсказать вероятность следующего слова. Такая модель довольно простая, и сделать ее можно, просто имея матрицу: как часто слова идут друг после друга.
Всю вашу последовательность мы разбиваем на произведения условных вероятностей:

$$\begin{aligned} & P(\mathrm{I}, \mathrm{saw}, \mathrm{the}, \mathrm{red}, \mathrm{house}) \\ \approx\ & P(\mathrm{I} \mid \langle s \rangle, \langle s \rangle) P(\mathrm{saw} \mid \langle s \rangle, I) P(\mathrm{the} \mid \mathrm{I}, \mathrm{saw}) P(\mathrm{red} \mid \mathrm{saw}, \mathrm{the}) P(\mathrm{house} \mid \mathrm{the}, \mathrm{red}) P(\langle /s \rangle \mid \mathrm{red}, \mathrm{house}) \end{aligned}$$

Почему чаще всего берется только два токена? Почему мы не используем для вероятности появления слова house все предыдущие слова? Тут вступает в силу проклятье размерности: естественно, для того, чтобы хранить вероятности каждого последующего слова при условии предыдущего (то есть, биграммную модель), вам достаточно вектора. То есть, если вы хотите предсказывать по двум контекстам, то вам нужна матрица. Если по трем, то вам нужен трехмерный тензор. В этом и есть проблема: все традиционные языковые модели не очень большие, в пределах двух слов контекста. После этого начинается огромный рост матрицы вероятности, она становится разреженной, и использовать ее становится очень сложно.

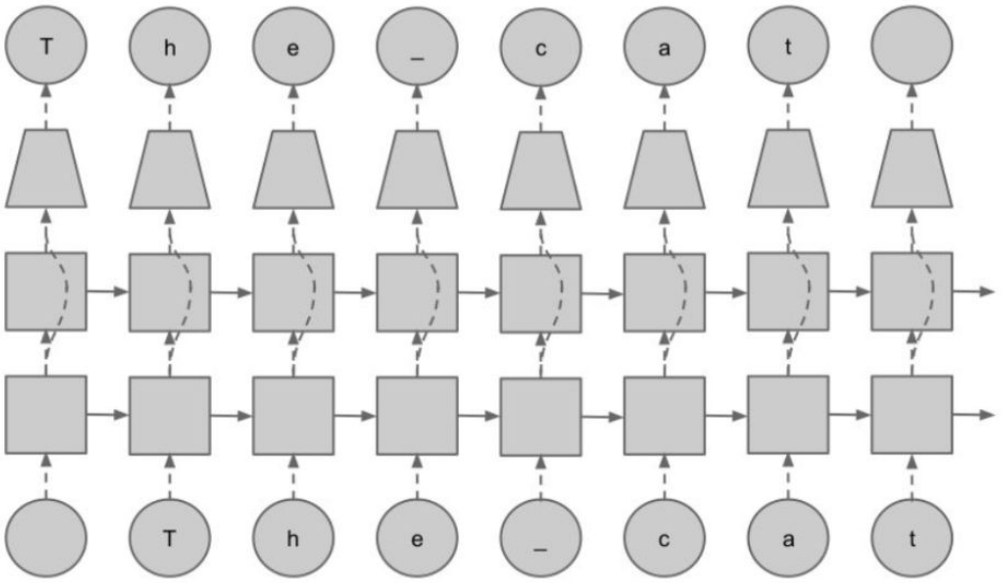
Языковые модели на нейронных сетях

Сейчас для построения языковых моделей в основном используют следующие архитектуры рекуррентных нейросетей:

- LSTM (Long Short-Term Memory) придумали еще в 90-е годы, но тогда использовать это было довольно сложно. Основная идея в том, чтобы иметь два пайплайна информации: один связан с скрытыми состояниями, а второй - с каким-то внутренним состоянием ячейки, который отвечает за передачу долговременной информации. По одному пайплайну текут длительные зависимости, а по другому - кратковременные. Почитать про это больше можно здесь или же в нашей первой статье.
- GRU (Gate Recurrent Unit) - аналог LSTM. Выглядит чуть запутанно, но ничего сложного нет. Используется gating-механизм, но нет self-state'ов, меньше параметров. Сеть меньше склонна к переобучению и работает чуть быстрее. Однако, как правило, дает результат немного хуже.

С помощью нейросетей языковая модель предсказывает следующий токен, имея **все** предыдущие. Нам дали последовательность токенов (до k-1), нужно предсказать вероятность токена k. Вероятность последовательности - это просто произведение вероятностей. Здесь можно вспомнить probability chain rule: эту совместную вероятность можно записать как произведение условных вероятностей. Это и используют языковые модели и большинство RNN.

На вход подается пустой токен, по которому мы предсказываем первый токен. Для простоты на картинке изображены буквы, но это могут быть слова или другие последовательности. Например, иероглифы, если вы исследуете древнеегипетский. Предсказав какой-то токен, мы подаем его на вход как следующий и пытаемся предсказать токен за ним. Таким образом, данная нейросеть пытается предсказать следующий токен по всей имеющейся истории.

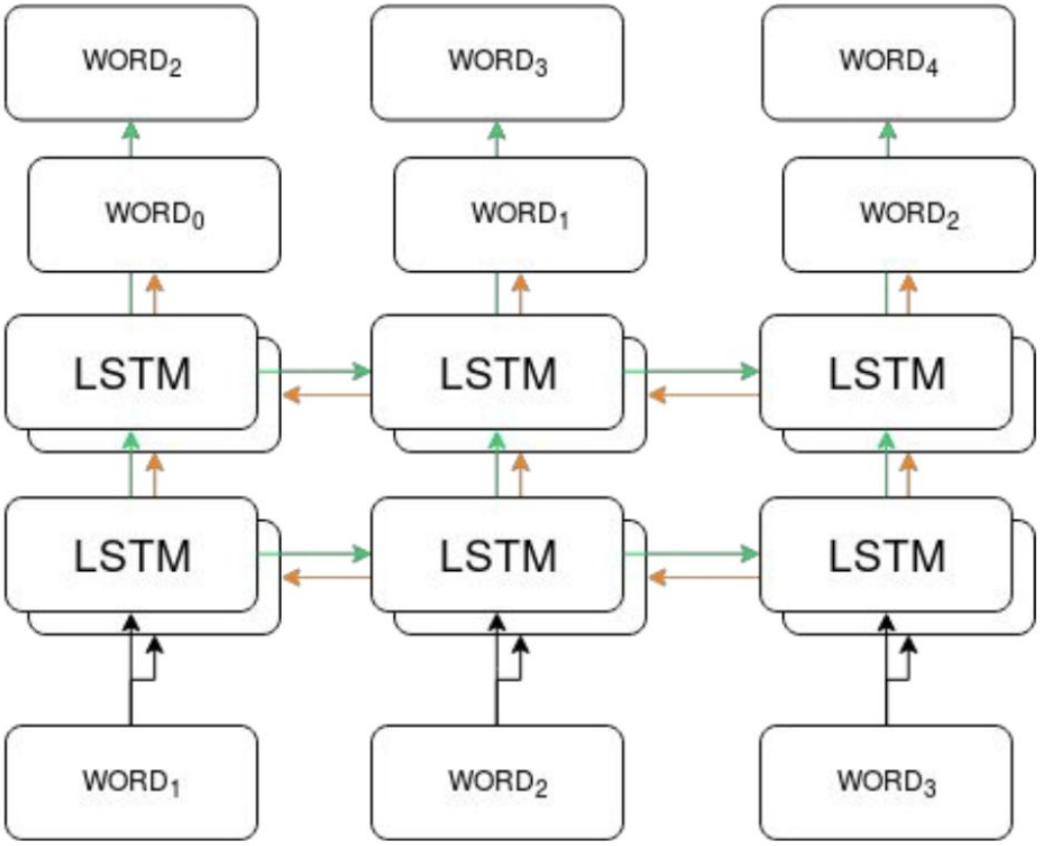


$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1}).$$

Вывод: это очень похоже на embeddings, где мы тоже предсказываем слово по контексту. Однако для RNN порядок имеет значение, и за счет учета этого порядка они дают куда более лучшие результаты.

Bi-directional Language Model

Можно сделать двунаправленную языковую модель и считать тексты не только слева направо, но и справа налево. Делать это нужно независимо и параллельно.



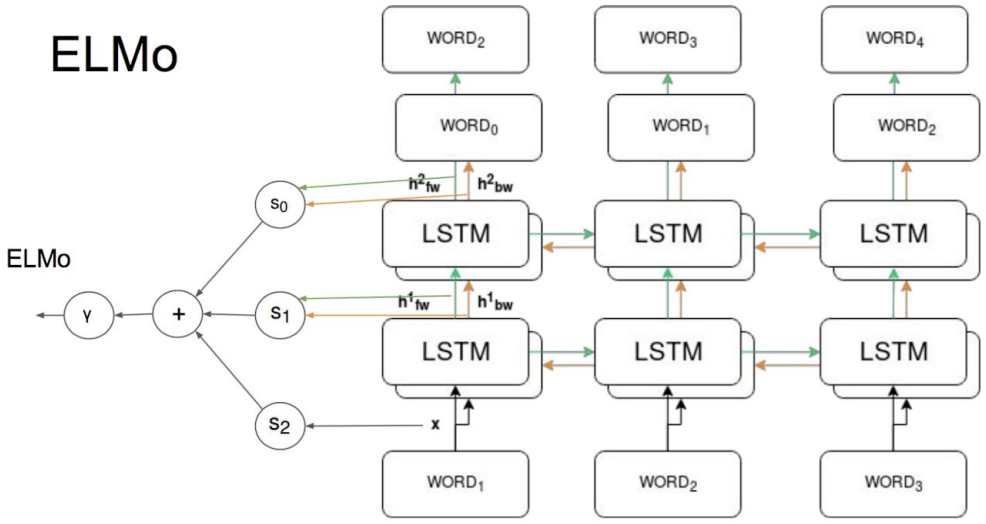
Оптимизировать в данном случае мы будем Log-likelihood для последующих токенов, а именно: для прямого прохода мы будем пытаться предсказать каждый следующий токен, имея весь предыдущий контекст, а для обратного прохода мы будем предсказывать текущий токен, имея весь последующий контекст. Например, для словаря из десяти тысяч слов нужно будет предсказывать вероятности для каждого слова. Так, размерность выхода будет такой же, как и словарь.

ELMo

ELMo - это проект по языковым моделям от Allen Institute. Идея: хорошо предобученные языковые модели можно использовать более умно (и неочевидно).

Очевидное использование: если, например, мы предобучили такую модель (и даже добавили embeddings), то можно было бы снять верхний слой и поставить сверху классификатор. Так делали - и это дает хороший результат.

Что предложили Allen NLP: давайте сделаем так же. Научим двунаправленную языковую модель и будем предсказывать слова в прямом и обратном направлении.



$$R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} = \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},$$

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

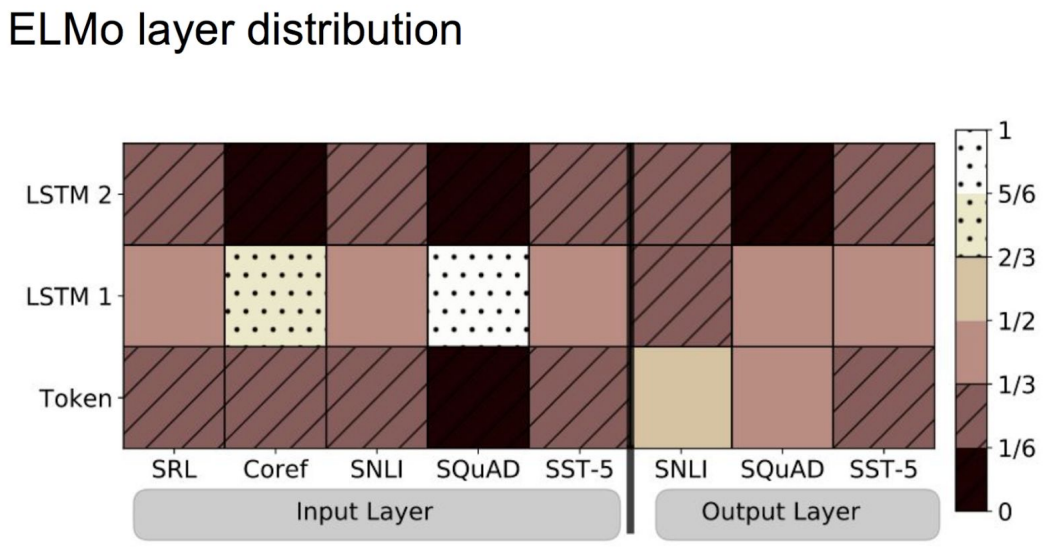
Вы берете ваши скрытые представления и умножаете на какой-то вес s_0. После чего берете другие скрытые представления, умножаете их на s_1, первые embeddings умножаете на вес s_2. Наконец, все это складываете, умножаете на какую-то гамму и получаете ваше итоговое представление.

Иными словами: давайте попробуем не дотьюнивать нашу языковую модель, не пытаться как-то дообучить ее, а просто попытаемся взять комбинацию между низкоуровневыми и высокоуровневыми представлениями и обучить эту комбинацию (а именно, веса для нее) для нашей задачи. Веса, естественно, должны все суммироваться к единице - то есть, это будут числа, которые вы потом подаете на softmax активацию. Что показали авторы идеи: если вы обучите хорошую language model, а потом будете учиться брать с нужных слоев ее скрытые представления, то вы можете получить неплохой прирост к качеству почти на любой NLP задаче.

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Что нужно сделать: найти подходящий корпус (если вы работаете с новостями, то нужен новостной корпус), предобучить на нем языковую модель, а потом использовать на вашем размеченном корпусе ELMo для решения своей задачи. Это в некотором смысле как следующий шаг embeddings: ELMo зависит не только от контекста, но и от порядка слов.

Если вы посмотрите на веса, с которыми будут складываться ваши скрытые представления с разных слоев, то окажется, что для разных задач они будут немного разными. В некоторых будут важны либо более высокие, либо более низкоуровневые представления. Например, можно заметить, что для SQuAD (The Stanford Question Answering Dataset) почти весь вес ушел на первый слой LSTM сети. Авторы статьи про SQuAD утверждают, что первый слой LSTM ответственен за синтаксис, второй слой - скорее семантика:



К сожалению, для русского языка, ELMo работает не так хорошо. Но, возможно, ее просто не дообучили до конца.

Посмотреть ELMo на практике можно посмотреть в блокноте. Оставить отзыв по прочитанной статье вы можете здесь. По всем вопросам пишите на почту курса: dmia@applieddatascience.ru