1D CNN.
Transfer Learning

Свертки для работы с текстами

01

Языковое моделирование

Языковое моделирование в КЛ

- В компьютерной лингвистике языковая модель – это **вероятностное распределение слов в текстах**:

- насколько вероятно данное наблюдение (последовательность слов) в языке?
- Например, “съешь еще этих мягких французских булок”.
- А “дом собака зеленый бегать”?

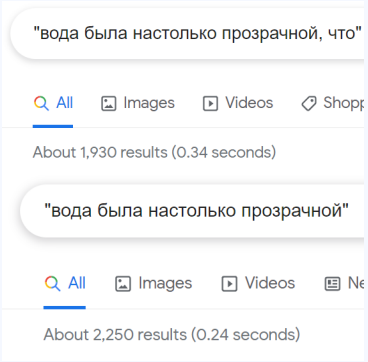
- зависит от конкретного языка

Для чего это нужно?

- Для машинного перевода: например, если мы хотим перевести “The human race” на русский, у слова race явно больше одного значения. Но после слова “человеческий” выше вероятность встретить слово “раса”, чем “гонка” (наверное)
- Для задач типа распознания устной речи: чтобы правильно выбрать между двумя похоже звучащими словами
- Для автоматического исправления орфографических и грамматических ошибок
- Для того, чтобы облегчить вам набор текста на телефоне...

Как посчитать?

- Используем теорию вероятности
- Считаем вероятности на корпусе текстов. Допустим, хотим вычислить вероятность фразы «вода была настолько прозрачной» + «что»
- Нам нужно посмотреть, сколько раз они встретились в нашем корпусе вместе и сколько раз отдельно встретилась фраза «вода была настолько прозрачной». Я посмотрю в гугл поиске:



Получилось что-то такое:

$$P(w|h) = \frac{C(\text{вода была настолько прозрачной, что})}{C(\text{вода была настолько прозрачной})} = \frac{1930}{2250} = 0.857$$

Функция правдоподобия

- Получается, чтобы смоделировать весь язык, нам нужно вычислить все вероятности всех слов
- Все эти вероятности вместе представляют собой вероятностное распределение (joint probability distribution)
- Собственно говоря, функция правдоподобия – это и есть такое вероятностное распределение, описанное в виде математической функции
- В этой функции наши вероятности слов – это параметры
- Итак, чтобы смоделировать язык, нужно подобрать параметры таким образом, чтобы наша математическая функция выдавала что-то максимально похожее на настоящий язык

Метод максимального правдоподобия

Как же вообще рассчитать эти самые вероятности слов, они же – параметры функции правдоподобия? Для этого используем такой способ:

- Чтобы вычислить вероятность конкретного биграма для слов w_{n-1}, w_n, посчитаем частоту этого конкретного биграма, а потом поделим на суммы частот всех биграмов, у которых первое слово – наше w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

- Эту формулу на самом деле можно упростить: ведь количество всех биграмов с первым словом w_{n-1} просто равно частоте самого слова w_{n-1}.

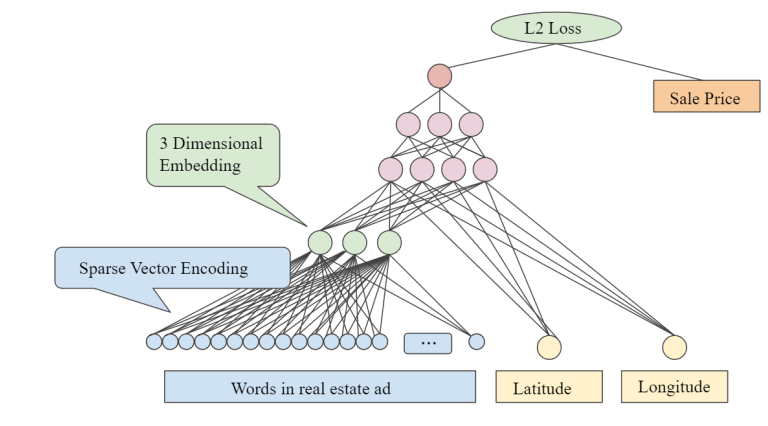
Метод максимального правдоподобия

- Получается, что каждый раз мы вычисляем частоту в корпусе нашего биграма и делим ее на частоту первого его слова.
- Это отношение называется относительная частота (relative frequency)
- Относительные частоты максимизируют правдоподобие корпуса, на котором мы их вычисляли, для нашей языковой модели
- Допустим, у нас есть корпус на миллион слов, и слово «китайский» встречается в нем 400 раз.
- Вероятность того, что случайно выбранное слово в этом корпусе окажется «китайский», равна 400 / 10^6 = 0.0004.
- В другом корпусе эта вероятность может быть другой, но в корпусе такого размера, как наш – это наилучшая вероятность

Идея эмбеддингов

- Нужно решить какую нибудь задачу с текстовыми данными в качестве признаков
- Представим каждое слово как one hot encoding
- Потом мы берем слой с рандомно установленным числом весов
- Учим
- Profit!

- Получается, модель пытается извлечь из нашего ОНЕ такие признаки, которые будут наиболее релевантными для той конкретной задачи, которую она теперь пытается решить

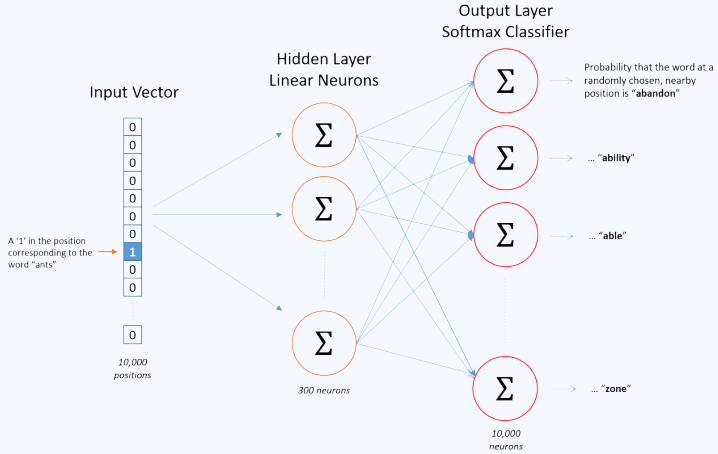


Слой эмбеддинга

- Итого, нужно:
    1. Словарь word2id (и id2word для раскодирования) – чтобы делать OHE
    2. Слой эмбеддингов (обучаемый!)
- Можно использовать предобученные чужие эмбеддинги
- Эмбеддинги могут быть для чего угодно: все, что можно закодировать OHE. Символы, токены, BPE токены, предложения, фильмы...
- А что, если нашей целевой задачей будет построение языковой модели?

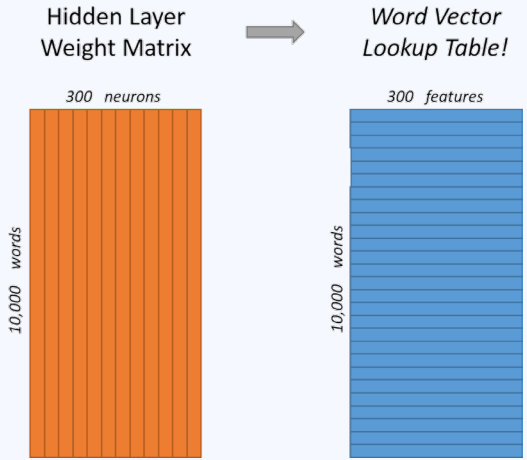
Нейронные сети и ЯМ

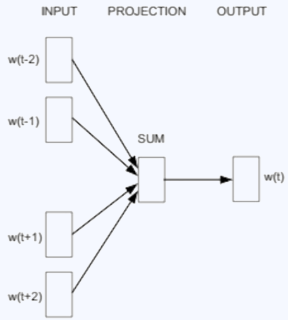
- Давайте будем пробовать построить языковую модель в нейронной сети: заставим MLP рассчитать вероятности всех слов на обучающей выборке
- В качестве функции потерь будем использовать Maximum Likelihood (обычно log)
- На вход будем подавать One Hot Encodings для слов: оно же sparse representation



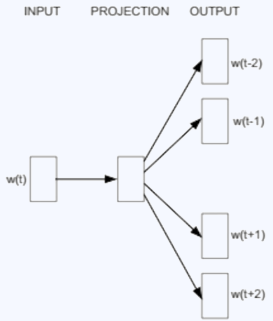
word2vec

- Две задачи: continuous bag of words vs skip gram
- Подаем ван-хоты, на скрытом слое вычисляем веса: получается, что когда подаем какое-то слово на вход нейронке, как бы выбираем его эмбеддинг:





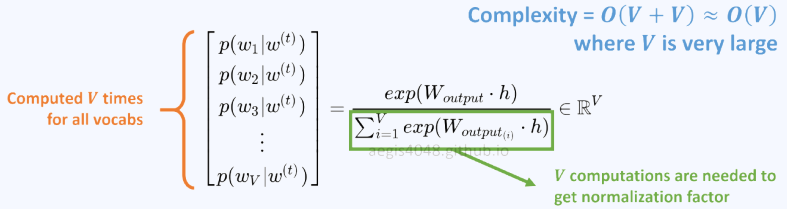
CBOW



Skip-gram

Word2vec – negative sampling

- В word2vec только два слоя. Это мелкая (shallow) архитектура
- Чтобы уменьшить количество вычислений (MLL приходится вычислять для всех слов постоянно), можно использовать некоторые методы
- Например, negative sampling: будем обновлять вероятности не для всех слов, а для n слов, которые не могут быть контекстом для нашего целевого слова
- Они выбираются по своей частотности в корпусе



Лирическое отступление: BPE

- За токен можно считать разные вещи. Например, New York – это один токен или больше?
- Вместо того, чтобы нам придумывать определение токена и самим писать правила, заставим текст сообщить нам, что в нем – токены
- Byte Pair Encoding – алгоритм автоматической токенизации по **подсловам** (subwords)
- Современные нейронные сети используют BPE
- Понадобятся **обучающие данные** – набор сырых текстов, на которых будем учить, какие бывают токены
- «Собираем» токены из символов

Лирическое отступление: BPE

Слова в наших обучающих данных: (“hug”, 10), (“pug”, 5), (“pun”, 12), (“bun”, 4), (“hugs”, 5)

Алфавит: [“b”, “g”, “h”, “n”, “p”, “s”, “u”], значит, можем представить слова так:

(“h” “u” “g”, 10), (“p” “u” “g”, 5), (“p” “u” “n”, 12), (“b” “u” “n”, 4), (“h” “u” “g” “s”, 5)

Устанавливаем, какого объема словарь хотим.

Вычисляем самые частотные сочетания: самая частотная пара – “ug”, она встретится 20 раз

Соединим эти два символа и получим новый словарь: [“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”]

Тогда наш корпус будет выглядеть так:

(“h” “ug”, 10), (“p” “ug”, 5), (“p” “u” “n”, 12), (“b” “u” “n”, 4), (“h” “ug” “s”, 5)

Какие два “слова” тогда будут чаще всего вместе? Найдите.

Склеим их в одно новое слово и добавим в наш словарь.

И так пока не доведем наш словарь до желаемого объема.

02

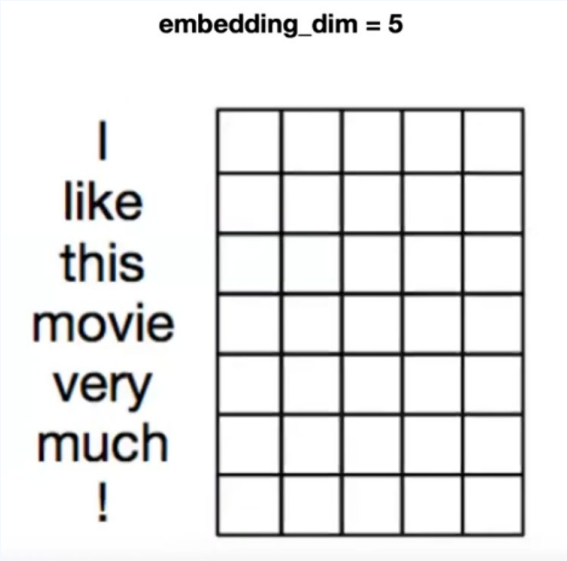
CNN и тексты

Текстовые данные

- Задаем длину эмбеддинга, например, 5
- Золотое правило, какую делать длину:

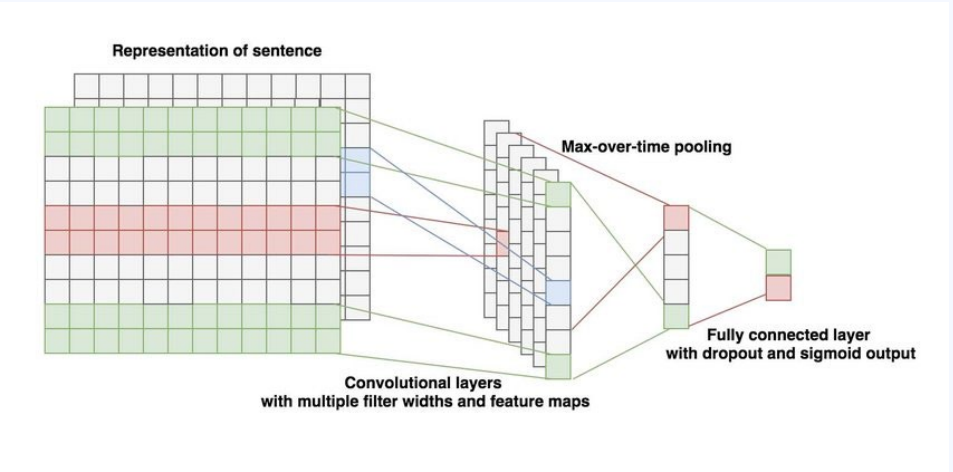
$$length = \sqrt[4]{vocabulary}$$

- Как получить эмбеддинг для всего предложения?
- Можно усреднить: мы так делали
- А что, если применить свертку?
- Свертка соберет важные признаки



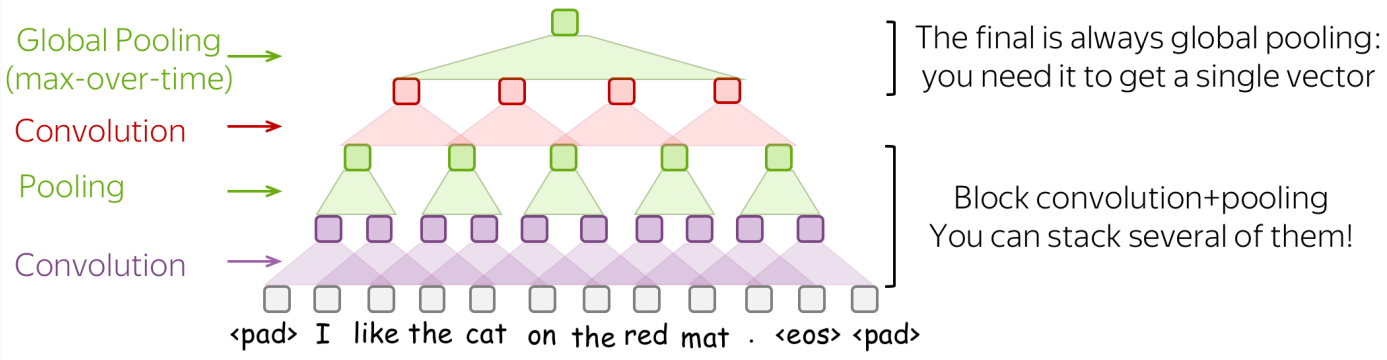
Как CNN сворачивает текст?

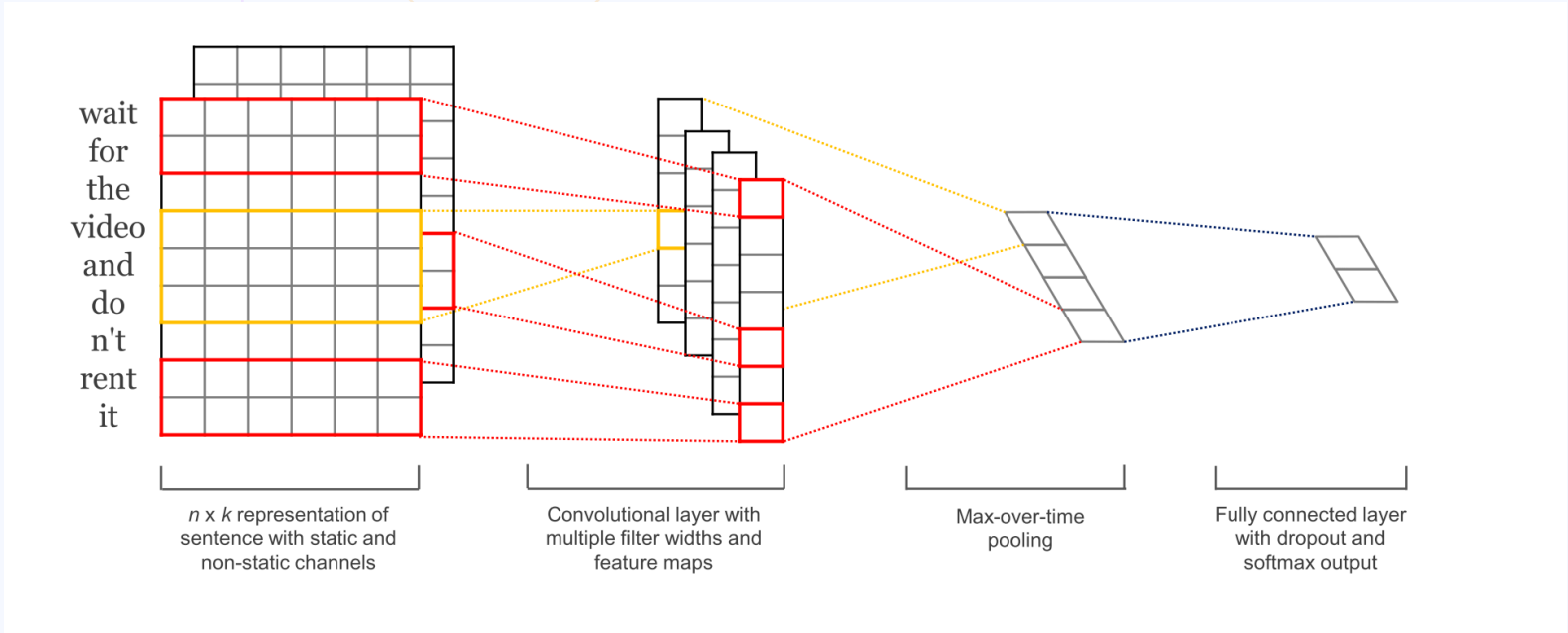
- Предложение – настаканные эмбеддинги слов
- Длина эмбеддинга – как бы глубина
- Поэтому свертка всегда идет на всю длину эмбеддинга
- Размер ядра – по сути n в граммах (2 – биграмма, 3 – триграмма)



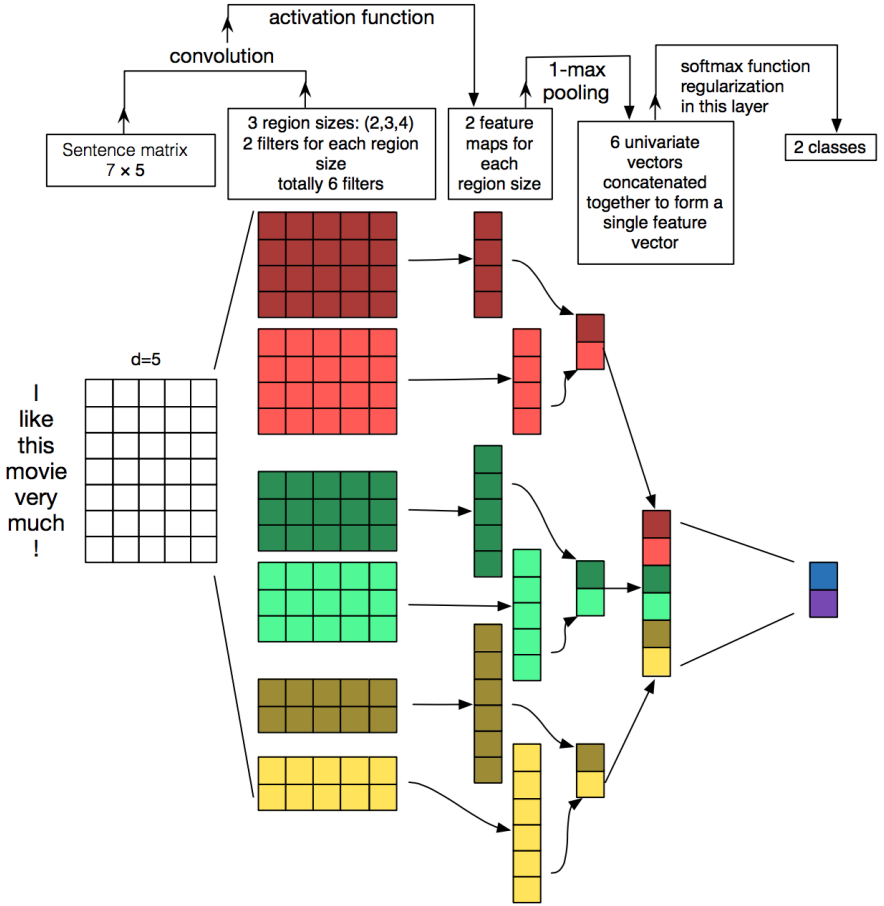
Max Over Time Pooling

- Сконкатенируем получившиеся результаты сверток (это называется карта активации)
- По каждому столбцу выберем максимальное значение




wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Inception CNN

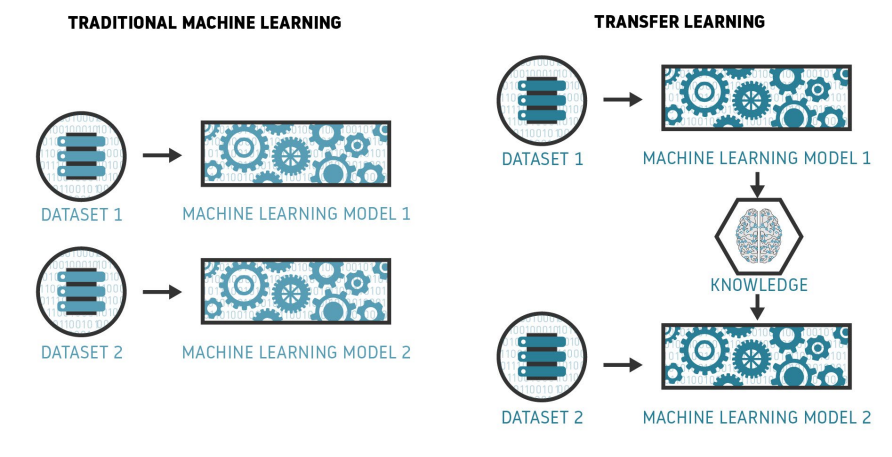


03

Transfer Learning

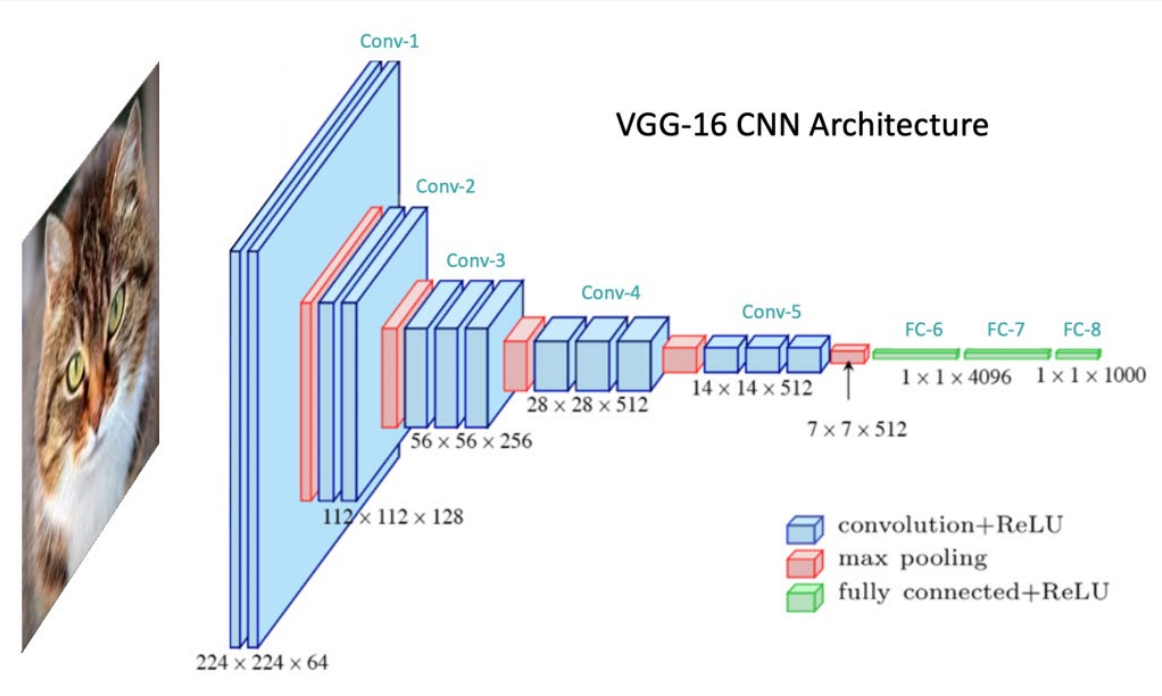
Transfer Learning

- На практике свёрточные сети с нуля обучают только большие технологические компании
- Это происходит из за ограниченности ресурсов
- Уже обученные архитектуры пытаются адаптировать под новые задачи, это называется transfer learning (перенос знаний)



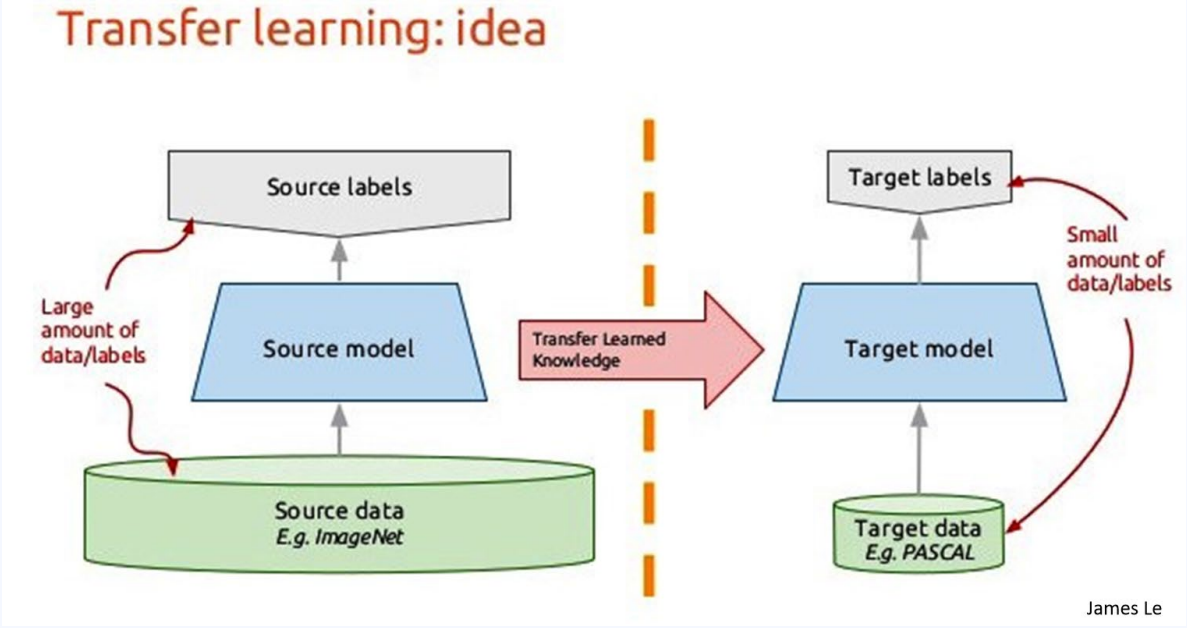
Transfer Learning

Глубокие сети извлекают из изображений сложные фичи, но для их обучения нужно много данных...



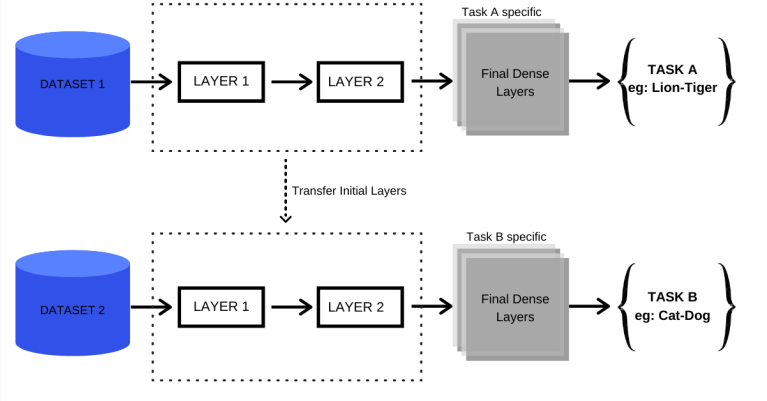
Transfer Learning

Давайте повторно использовать уже предобученную сеть!



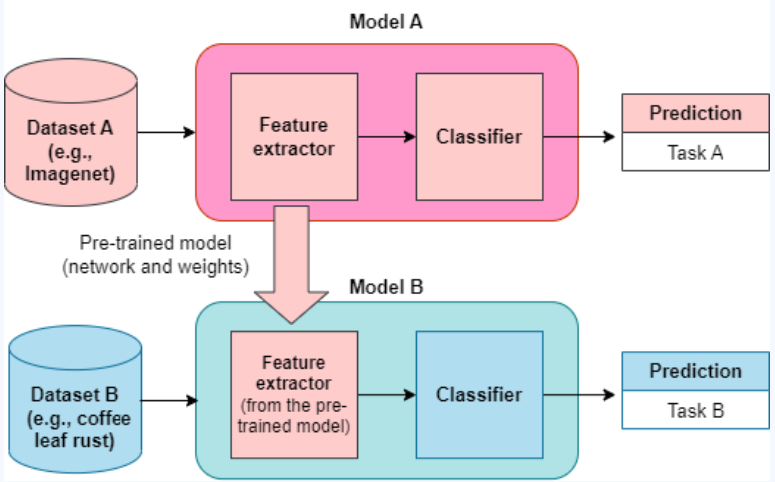
Transfer Learning

- Нужно меньше данных для обучения, так как нас интересуют лишь последние слои
- Как правило, на первых слоях фильтры похожие для всех задач
- Чем сильнее новая задача отличается от исходной, тем больше слоёв нужно переучивать
- Например, если мы хотим распознавать эмоции, в датасете для нашей сетки должны были быть человеческие лица



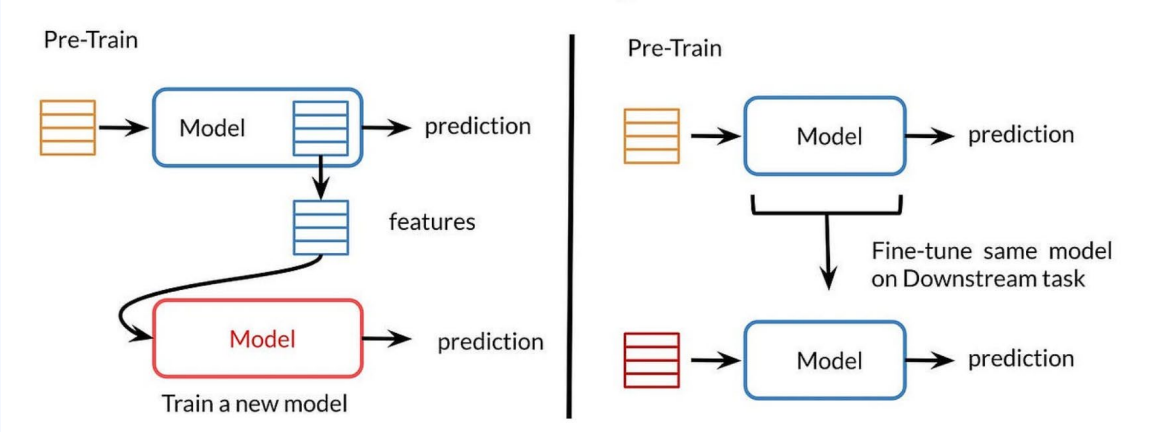
Transfer Learning

- Обычно наша сетка состоит из двух частей: первая извлекает признаки (Embedding layer, CNN, RNN, LSTM...), а вторая уже решает конкретную задачу (FC): эту задачу называют downstream task
- Иногда про решающий слой говорят «голова»
- Его обычно и снимают, чтобы заменить на новый под другую задачу



Fine Tuning

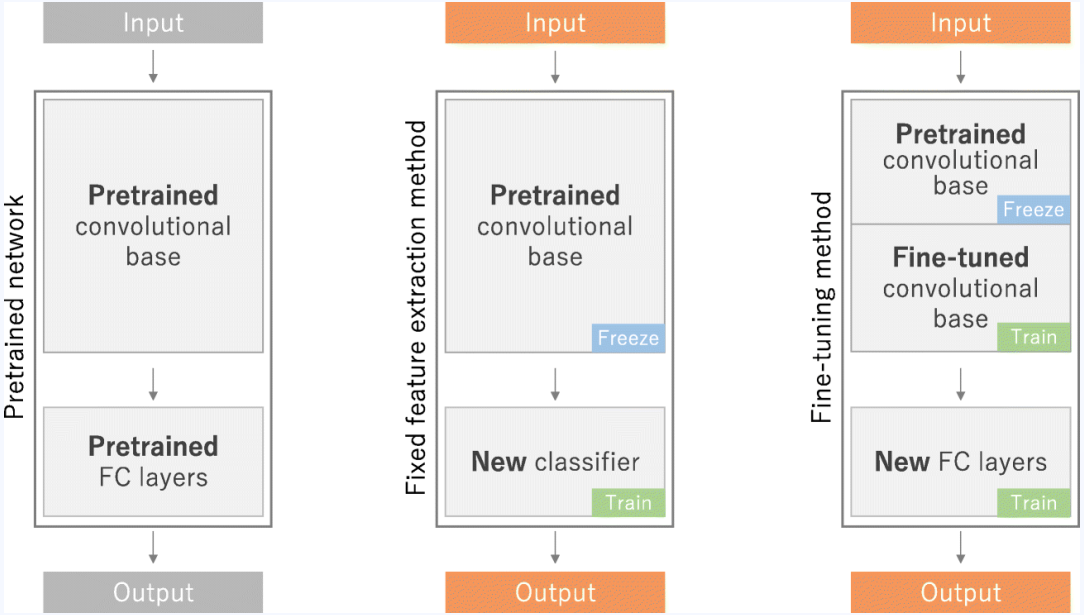
- Можно инициализировать веса переучиваемых слоёв весами с предобученной сети
- Это называется fine tuning, так как инициализация неслучайная



Transfer Learning vs Fine Tuning

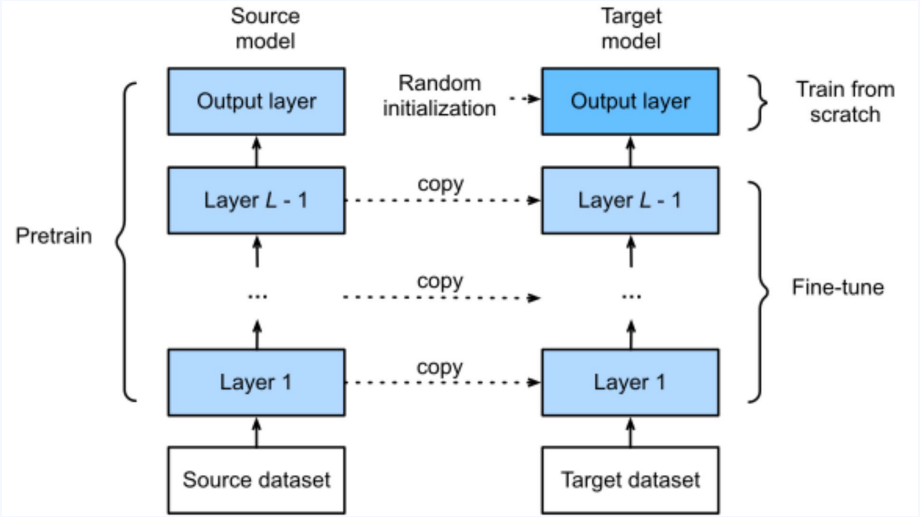
- При Transfer Learning не обучаем взятые от исходной модели слои: мы **замораживаем** их веса
- При Fine Tuning мы берем слои от исходной модели с их весами, но обучаем их с медленным learning rate

Соответственно:



Transfer Learning

- Чем больше датасет, тем больше слоёв можно доучивать
- При finetuning выставляйте более низкую скорость обучения
- Для начала попробуйте 0.1 от оригинальной



Виды Transfer Learning

Fine tuning: берем за базу предобученную модель, инициализируем свою новую ее весами, часть слоев дообучаем (можно все, можно только некоторые) с более маленьким learning rate

Feature extraction: берем от предобученной модели ту ее часть, которая извлекает признаки (CNN, RNN...), обучаем собственную голову-классификатор на них

Domain adaptation: файнтьюним предобученную на данных из другого домена

Multi task learning: обучаем одну модель решать сразу несколько задач, чтобы она одновременно улучшала свои предсказания во всех

Zero shot learning: берем предобученную модель и в лоб применяем ее на новых данных без предобучения

Зоопарки моделей

В интернете есть зоопарки с моделями (и, например, есть huggingface.co, до которого уже скоро доберемся...)

- Один большой зоопарк
- Зоопарк для любителей pytorch