Современные подходы к разработке интеллектуальных систем. Классические методы извлечения знаний из текста. Нейронные сети для анализа текста. (продолжение).

Лекция 4

Киселёв Глеб Андреевич
к.т.н., старший преподаватель
ФФМиЕН РУДН
тел.:+79067993329
email: kiselev@isa.ru

Логистическая регрессия

Дано:

- Коллекция документов по какой-то выбранной нами тематике – положительные примеры;
- Фоновая коллекция документов – отрицательные примеры;
- Длина документов – от нескольких предложений (большая длина);

Требуется:

- Построить бинарный *классификатор* $Classifier : R^d \to \{0,1\}$

Бинарный классификатор – функция по вещественному вектору размерности d (размер словаря) возвращает 1, если документ удовлетворяет нашей тематике и 0, если не удовлетворяет. Документ представляется в виде мешка слов.

Определение модели: $\hat{y}(x) = \sigma(w^T x + b)$, где $x \in R^d$ - множество признаков, дополненных фиктивным единичным признаком, $\hat{y} \in R, 0 \le \hat{y} \le 1$ - вероятность выпадения 1 для случайно величины $y \in \{0,1\}$, $w \in R^d$ - вектор весов, а $\sigma(x) = \frac{1}{1+e^{-x}}$ - логистическая функция (сигмоида). $w^T x$ - скалярное произведение 2 векторов (результат – вещественное число. Сигмоида применяется для сжатия выхода линейной регрессии и получения вероятности принадлежности объекта нужному нам классу от 0 до 1. Решение принадлежности к положительному классу принимается с помощью порога – все что ниже него – не подходит, выше – принадлежит множеству положительных примеров. В итоге мы получаем *искусственный нейрон* с d входами и 1 выходом.

Настройка весов происходит с помощью функции потерь.

Логистическая регрессия

Одним из типов функции потерь является бинарная кросс-энтропия:

$BCE(\hat{y}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y}) \to \min$ - функция потерь для 1 примера. Веса настраиваются за счет минимизации функции потерь. Основным алгоритмом такой функции является алгоритм градиентного спуска.

$-y \log \hat{y}$ - отвечает за штраф в случае *ложно отрицательных* предсказаний (документ принадлежит классу 1, а модель выдала низкую вероятность принадлежности)

$-(1-y)\log(1-\hat{y})$ - отвечает за штраф *ложно положительных* предсказаний (документ не принадлежит классу 1, а модель выдала высокую вероятность принадлежности.

Задача – минимизация общего штрафа.

Много классовая задача: необходимо предсказывать 1 метку из нескольких:

$Classifier : R^d \to \{1,...c\}$, $c > 2$ – количество различных классов.

$\hat{y}(x) = soft\max(Wx)$ - прямоугольная матрица (строки – классы, столбцы - признаки),где $x \in R^d$ - вектор признаков;

$\hat{y} \in R^c; \sum_{i=1}^{c} \hat{y}_i = 1$ - на выходе – распределение вероятностей;

$W \in R^{c \times d}$ - вектор весов

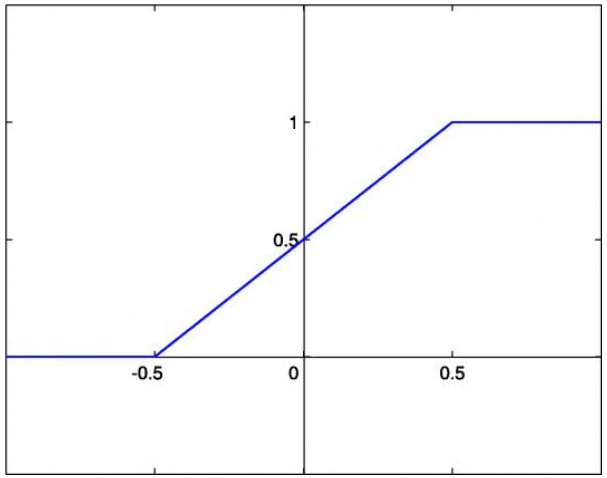
$soft\max(x) = \left\{ \frac{e^{x_i}}{\sum_{j=1}^{c} e^{x_j}} \right\}$ - функция активности softmax (вектор-функция, преобразовывающая вектор вещественных чисел из произвольного диапазона в вектор чисел от 0 до 1, сумма которых = 1.

Функция потерь: $CE(\hat{y}, y) = -\sum_{i=1}^{c} y_i \log(\hat{y}_i)$ - штраф только за ложноотрицательные предсказания.

Функции активации

Линейная функция активации

A=cx



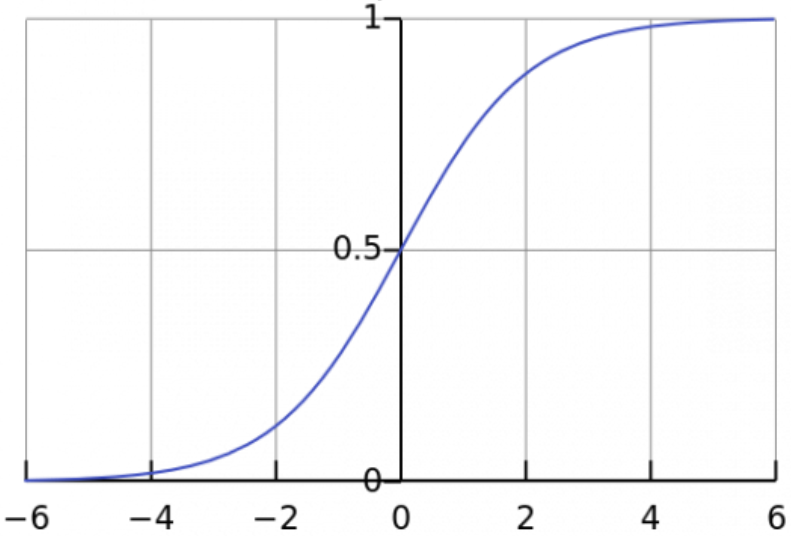
Такой выбор активационной функции позволяет получать спектр значений, а не только бинарный ответ. Можно соединить несколько нейронов вместе и, если более одного нейрона активировано, решение принимается на основе применения операции max (или softmax).

Основная проблема: финальная функция активации на последнем слое является линейной функцией от входов на 1 слое

Пример взят с https://neurohive.io/ru/osnovy-data-science/activation-functions/

Сигмоида

$$A = \frac{1}{1+e^{-x}}$$

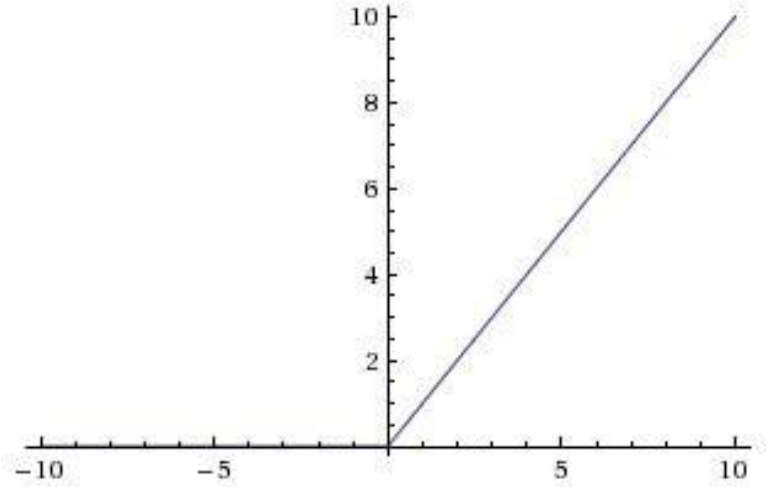


Преимущество сигмоиды над линейной функцией заключается в следующем. В первом случае имеем фиксированный диапазон значений функции — [0,1], тогда как линейная функция изменяется в пределах (-inf, inf). Такое свойство сигмоиды очень полезно, так как не приводит к ошибкам в случае больших значений активации.

Основная проблема: при приближении к концам сигмоиды значения Y имеют тенденцию слабо реагировать на изменения в *X*. Это означает, что градиент в таких областях принимает маленькие значения. А это, в свою очередь, приводит к проблемам с градиентом исчезновения.

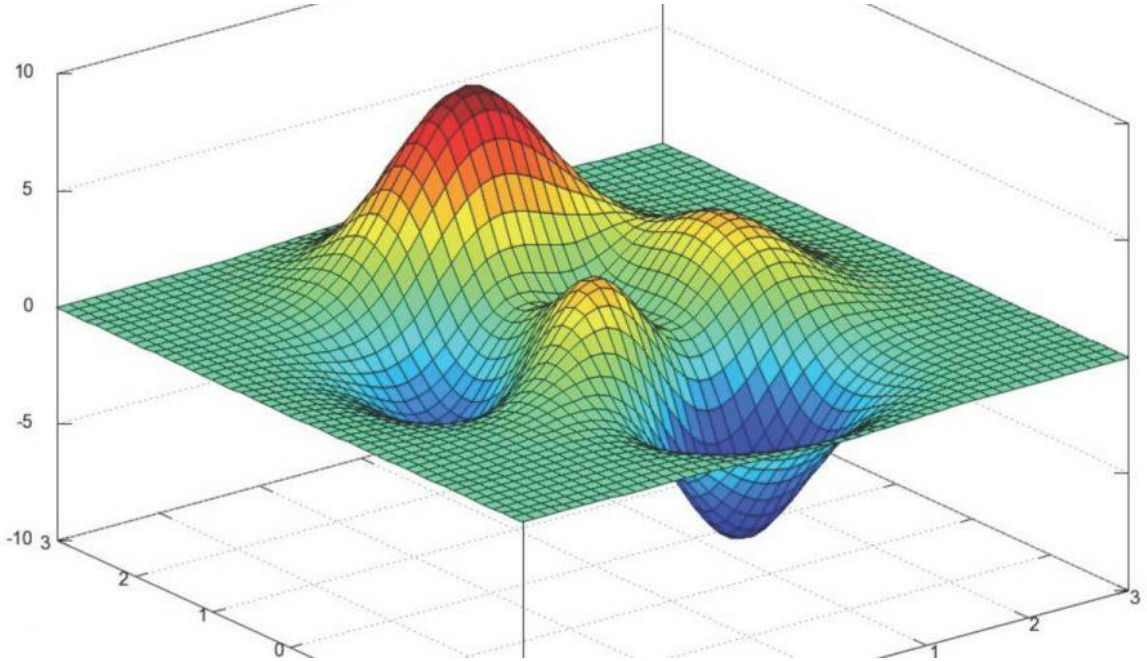
ReLu

A(x)=max(0,x)



На первый взгляд кажется, что ReLu имеет все те же проблемы, что и линейная функция, так как ReLu линейна в первом квадранте. Но на самом деле, ReLu нелинейна по своей природе, а комбинация ReLu также нелинейна! (На самом деле, такая функция является хорошим аппроксиматором, так как любая функция может быть аппроксимирована комбинацией ReLu). Это означает, что мы можем стэкать слои. Область допустимых значений ReLu — [0,inf), то есть активация может “взорваться”. Из-за того, что часть ReLu представляет из себя горизонтальную линию (для отрицательных значений X), градиент на этой части равен 0. Из-за равенства нулю градиента, веса не будут корректироваться во время спуска. Это является основной проблемой ReLu.

Градиентный спуск

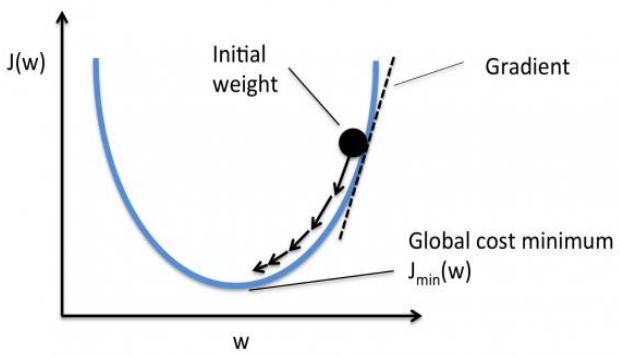


Это метод нахождения минимального значения функции потерь (существует множество видов этой функции). Минимизация любой функции означает поиск самой глубокой впадины в этой функции. Имейте в виду, что функция используется, чтобы контролировать ошибку в прогнозах модели машинного обучения. Поиск минимума означает получение наименьшей возможной ошибки или повышение точности модели. Мы увеличиваем точность, перебирая набор учебных данных при настройке параметров нашей модели (весов и смещений).

Как искать впадины: наклон графика функции – производная от функции по переменной. Куда наклон – там и впадина.

На рисунке мы видим график функции потерь (названный «Ошибка» с символом «J») с одним весом. Теперь, если мы вычислим наклон (обозначим это dJ/dw) функции потерь относительно одного веса, то получим направление, в котором нужно двигаться, чтобы достичь локальных минимумов. Давайте пока представим, что наша модель имеет только один вес.

Важно: когда мы перебираем все учебные данные, мы продолжаем добавлять значения dJ/dw для каждого веса. Так как потери зависят от примера обучения, dJ/dw также продолжает меняться. Затем делим собранные значения на количество примеров обучения для получения среднего. Потом мы используем это среднее значение (каждого веса) для настройки каждого веса.

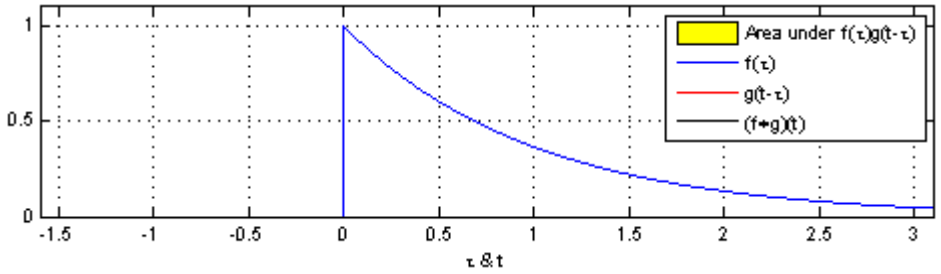


Пример взят с https://neurohive.io/ru/osnovy-data-science/gradient-descent/

Теперь, когда мы определили направление, в котором нужно подтолкнуть вес, нам нужно понять, как это сделать. Тут мы используем коэффициент скорости обучения, его называют гипер-параметром. Гипер-параметр – значение, требуемое вашей моделью, о котором мы действительно имеем очень смутное представление. Обычно эти значения могут быть изучены методом проб и ошибок. Здесь не так: одно подходит для всех гипер-параметров. Коэффициент скорости обучения можно рассматривать как «шаг в правильном направлении», где направление происходит от dJ/dw.

Свёрточные нейросети

Что такое свертка? – операция, выполняемая над двумя функциями или двумя сигналами. Пример из Wikipedia:



$$(f * g) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$ - одномерная свертка в общем виде для непрерывного случая (f – ядро, g – сигнал)

$$ConvOut_i = \sum_{k=1}^{K} InSignal_{i-K/2+k} \bullet Kernel_k \quad \forall i = 0,1,...,L$$ - дискретный случай

Синий график – входные данные, а красный – ядро свертки.

Результатом свертки является третья функция или сигнал. Первым сигналом является множество входных данных, а вторым часть параметров нейросети, которые выполняют роль ядра свертки.

Ядро свертки двигается относительно входных данных, значением сдвига является скалярное произведение фрагмента сигнала с ядром свертки. (оцениваем степень схожести текущего фрагмента сигнала и ядра).

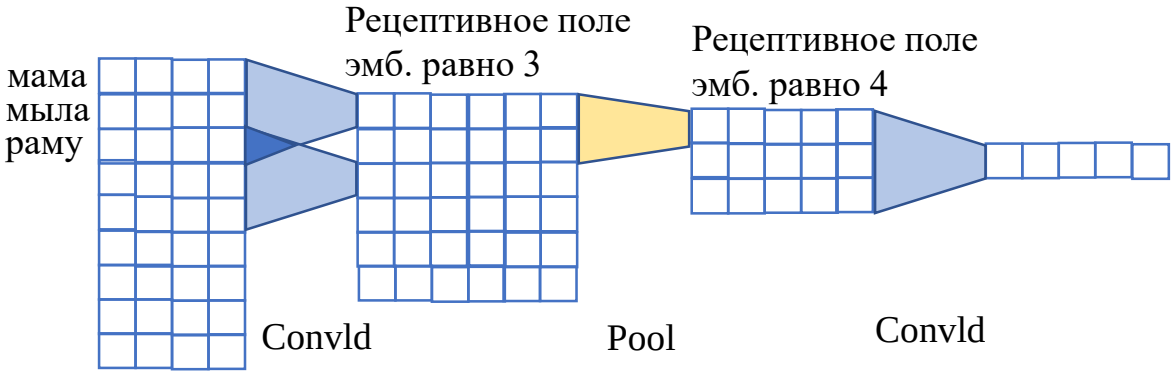
Перебираются все возможные смещения ядра относительно сигнала.

В Дискретном случае, результат применения одномерной свертки – вектор длинной L с размером ядра K. Размер сигнала L+K. За раз анализируется вектор длиной K. Помещаем ядро так, чтобы его центральный элемент находился над i-м элементом входного сигнала и вычисляем скалярное произведение ядра и фрагмента сигнала. Затем, повторяем эту операцию для всех возможных смещений.

Цель применения сверток – выделение локальных паттернов (особенностей сигнала) без зависимости от их позиции в сигнале, а ядро свертки описывает паттерн, который мы ищем.

Пример взят с: https://stepik.org/lesson/225314/step/4?unit=198057

Свёрточные нейросети



Зачем нужен пулинг?

- Для снижения размерности входного вектора

Примеры пулинга:

- Выбор максимальных значений
- Выбор средних значений
- Сумма значений

1. Строим матрицу эмбеддингов слов или символов. Количество строк соответствует длине текста; (берем k целых строк из входной матрицы для каждого из смещений)
2. Вытягиваем эти строки в один вектор и вычисляем скалярное произведение с ядром.

В нашем случае вектор будет иметь длину k*s, то есть ["размер ядра" умножено на "размер входного эмбеддинга"]. Так мы получили одно значение для вектора текущего шага. Чтобы получить остальные значения нам нужно взять ещё таких же ядер и тоже их применить.

3. Переходим к другому значению смещения и повторяем такие же операции.

Обычно свёрточные блоки применяют один за другим, перемешивая с блоками пулинга или агрегации, а также с функциями активации. Делают это для того, чтобы расширить пятно восприятия, то есть получить возможность обрабатывать более широкий контекст, более длинные паттерны.

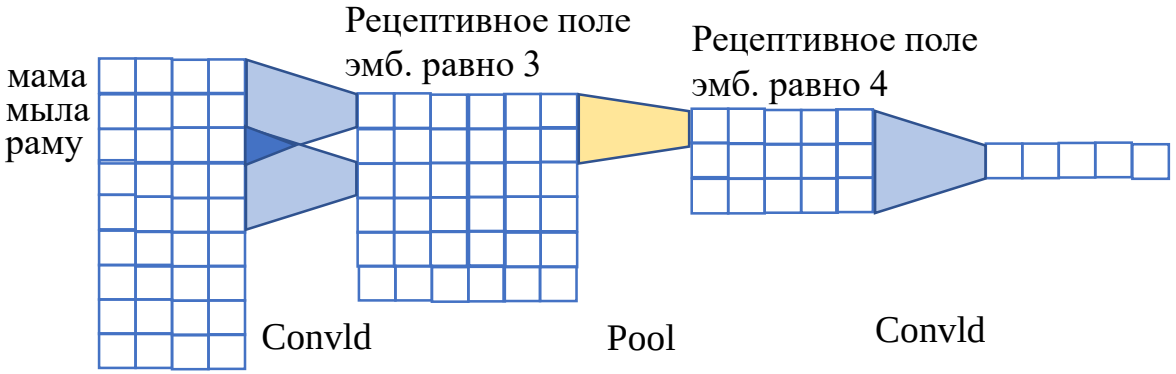
- Пример задачи на расчет ширины рецептивного поля: Оцените ширину рецептивного поля для нейросети, состоящей из 4 свёрточных слоёв с ядром 5, соединённых последовательно.

Алгоритм:

1. У нейросети 5 выходов, соответственно на каждом шаге мы сжимаем 5 значений в 1.
2. Переходим на шаг назад.

Пример взят с: https://stepik.org/lesson/225314/step/4?unit=198057

Свёрточные нейросети

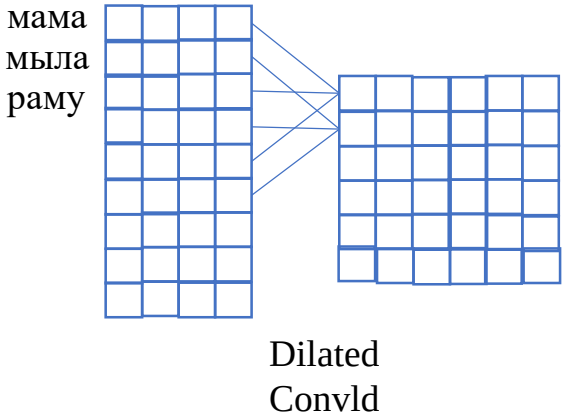


Сколько потребуется свёрточных слоев с прореженными свертками с ядром длины 5, чтобы учесть связь между первым словом в предложении и 30-м словом (длина текста >30): Ответ – всего 5!

$$Dilation[layer] = 2^{layer}$$

1. Хорошо обучаются и сходятся
2. Нет проблем затухающего градиента
3. Подходят для распознавания локальных паттернов, словосочетаний, коротких фраз
4. Слабо подходят для учета широкого контекста (нужно создавать глубокую сеть)

Разреженные свертки:



1. Ядро свертки применяется не к непрерывному фрагменту сигнала, а к фрагменту, у которого удалена часть элементов (например, элементы с четными номерами) – увеличивается размер рецептивного поля;
2. Шаг прореживания увеличивается с глубиной;
3. Первый слой без прореживания, затем на каждом слое шаг увеличивается в 2 раза

Рекуррентные нейросети

- Есть внутреннее состояние, которое мы обновляем h, которое обновляется после прочтения очередного элемента входной последовательности x.

$$z_t = W_{input} \cdot x_t \quad W_{input} \in R^{d_{hidden} \times d_{input}}$$

Внутреннее состояние содержит информацию обо всем прочитанном ранее тексте. В отличии от сверток, где состояние не хранится, а обрабатывается кусок текста определенного размера сразу.

- Рекуррентные нейросети могут предсказывать как 1 величину по всему тексту (классификация), так и несколько величин для каждого куска текста.
- После обработки текста вычисляется новое значение состояния (ячейки памяти), исходя из её старого значения и входного элемента.

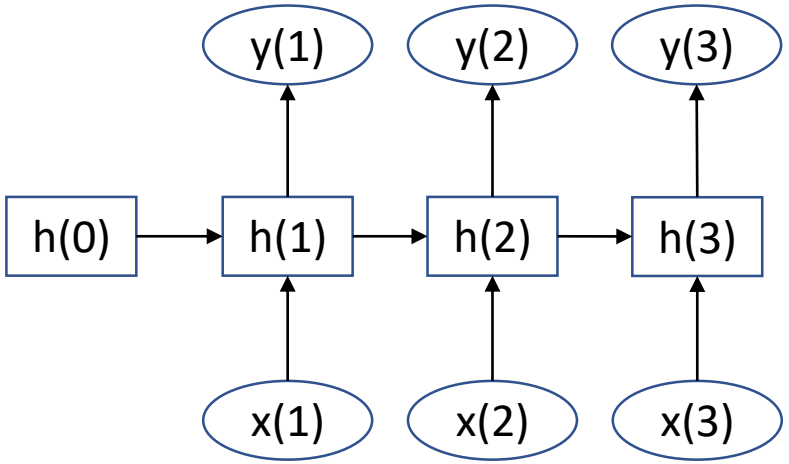
$$h_t = \tanh(W_{hidden} \cdot h_{t-1} + z_t), W_{hidden} \in R^{d_{hidden} \times d_{hidden}}$$

tanh – гиперболический тангенс (функция активации)

- Далее делается предсказание, используя только текущее значение состояния:

$$y_t = W_{output} \cdot h_t$$

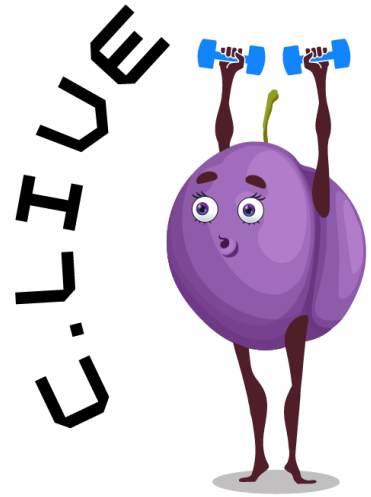
- После этого читаем следующее слово и повторяем процесс.



Основные отличия от свёрточных нейросетей:

- Потенциально мощнее, чем свертки;
- Последовательная обработка данных медленнее, чем свертки;
- Есть проблемы сходимости (затухания или взрыва градиента);
- Поиск баланса между мощностью и простотой обучения;

Спасибо за внимание!



Руководитель проекта Когнитивный ассистент
старший преподаватель, к.т.н. Киселёв Г.А.
+79067993329
kiselev@isa.ru