DEEP LEARNING

Неделя 4

Артем Грачев, Святослав Елизаров, Борис Коваленко

18 ноября 2017

Высшая школа экономики

ФУНКЦИИ АКТИВАЦИИ

ПРИМЕР

Данная проблема называется проблемой **исчезающего градиента** (vanishing gradient problem).

В рассмотренном примере использовалась функция $\sigma(x)$, максимальное значение, которое может принять её производная равно 0.25. В глубокой нейронной сети, использующей сигмоид в качестве функции активации между слоями, градиент с большой вероятностью будет исчезать.

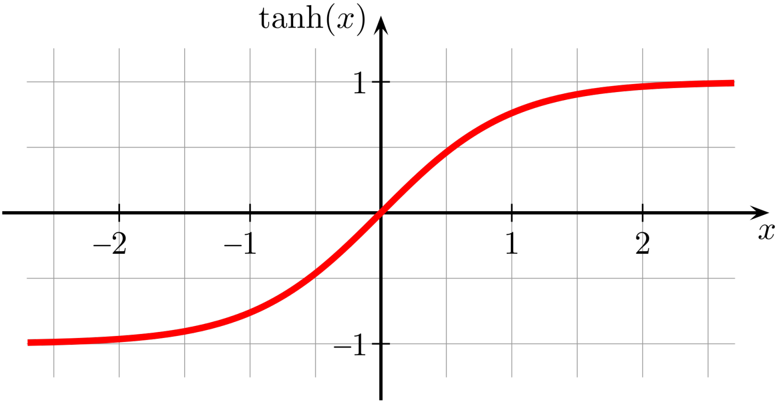
Важно понимать как работает алгоритм обратного распространения и как ведут себя производные функций активации, которые вы используете!

ГИПЕРБОЛИЧЕСКИЙ ТАНГЕНС

Часто в качестве функции активации используется гиперболический тангенс:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = 2\sigma(2x) - 1$$

ГИПЕРБОЛИЧЕСКИЙ ТАНГЕНС



ГИПЕРБОЛИЧЕСКИЙ ТАНГЕНС

1. Отмасштабированная версия сигмоида
2. $\forall x \in R, \tanh(x) \in (-1, 1)$
3. Максимальное значение производной равняется 1 (проверьте это!)

Всегда предпочитайте tanh сигмоиду в качестве функции активации.

Есть ли проблемы?

ГИПЕРБОЛИЧЕСКИЙ ТАНГЕНС

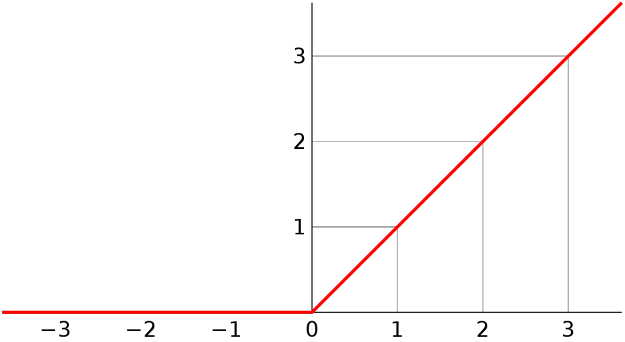
1. Функция ”чувствительна” только в окрестности 0
2. Для любых больших по модулю значений аргумента функция имеет практически одинаковое значение
3. Таким образом градиент исчезает как и в случае σ

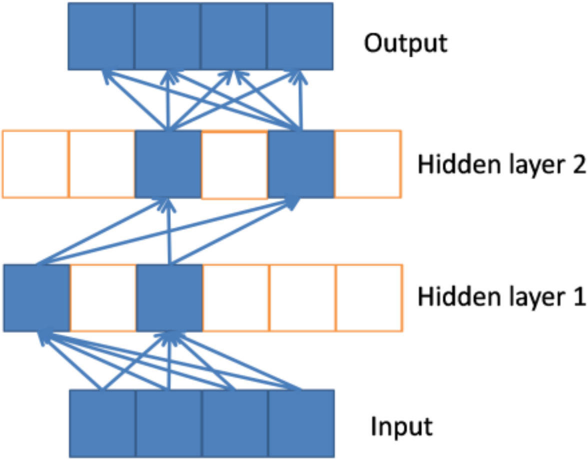
RECTIFIER LINEAR UNIT

Функция $ReLU(x) = max(0, x)$ лишена этих недостатков. Действительно:

$$ReLU(x)' = \begin{cases} 1, & \text{если } x > 0 \\ 0, & \text{если } x < 0 \end{cases}$$

RECTIFIER LINEAR UNIT




Output
Hidden layer 2
Hidden layer 1
Input

RECTIFIER LINEAR UNIT

- Градиент не затухает
- Разреживает сеть
- Однако, важно помнить, что при отрицательных параметрах *ReLU* выключается, и градиент равен 0.

”Протекающая” версия *ReLU*:

$$LReLU(x) = \max(\alpha x, x)$$

Где $\alpha < 1$

СМЕЩЕНИЕ

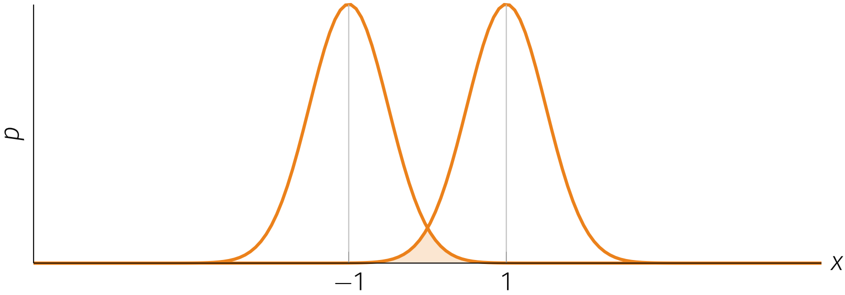
- В ходе оптимизации мы движемся в напралении, противоположном градиенту функции
- $\theta_{n+1} = \theta_n - \lambda \nabla f(\theta)$, Learning rate λ гарантирует, что изменение будет ”небольшим” (по Евклидовому расстоянию)

СМЕЩЕНИЕ

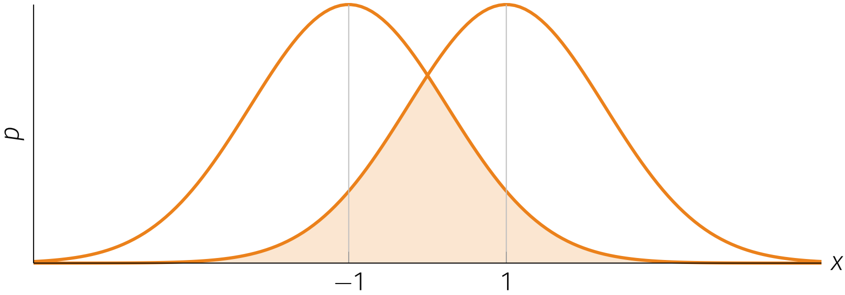
- В ходе оптимизации мы движемся в направлении, противоположном градиенту функции
- $\theta_{n+1} = \theta_n - \lambda \nabla f(\theta)$, Learning rate λ гарантирует, что изменение будет ”небольшим” (по Евклидовому расстоянию)

Возможны ли какие-либо проблемы из-за такой процедуры?

СМЕЩЕНИЕ



СМЕЩЕНИЕ



СМЕЩЕНИЕ

- В обоих случаях среднее изменилось на 2
- Но в первом распределения гораздо больше отличаются, чем во втором

СМЕЩЕНИЕ

- Теперь представим себе, распределение активаций на каждом слое сети
- При изменении параметров распределения будут меняться
- Но так как карты активации зависят от входа, то при сдвиге карт активации предыдущего слоя, последующие слои будут вынужденны перестраиваться под новое среднее.
- С глубиной проблема усугубляется, что может привести к замедлению тренировки, а иногда и полному вырождению карт активации

СМЕЩЕНИЕ

Существует несколько способов решения проблемы:

- Метод натурального градиента
- Специальные техники инициализации параметров
- Batch Normalization
- Weight Normalization
- **Специальные функции активации**

EXPONENTIAL LINEAR UNIT

Экспоненциальная версия *ReLU*:

$$elu(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$$

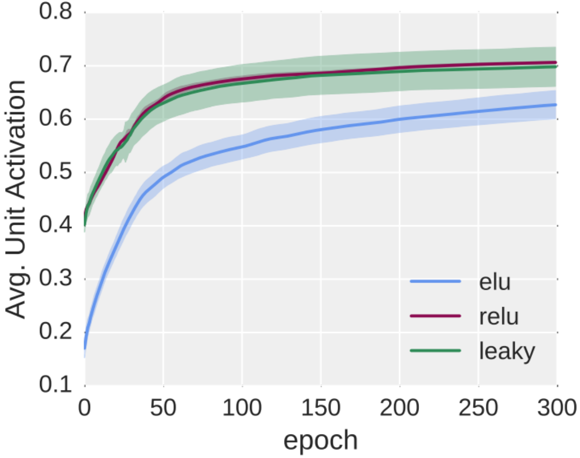
Где $\alpha \geq 0$

Clevert, D. A., Unterthiner, T., Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289.

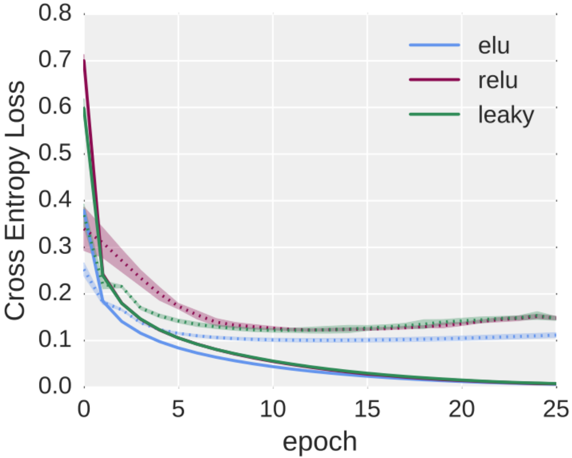
EXPONENTIAL LINEAR UNIT

Экспоненциальная версия *ReLU*:

$$elu(x)' = \begin{cases} 1 & x > 0 \\ elu(x) + \alpha, & x \leq 0 \end{cases}$$


Avg. Unit Activation
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250
300
epoch
elu
relu
leaky

EXPONENTIAL LINEAR UNIT



А как быть с дисперсией?

SELU

А как быть с дисперсией?

$$selu(x) = \lambda \begin{cases} x & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$$

Где $\alpha \geq 0, \lambda > 1$

Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S. (2017). Self-Normalizing Neural Networks. arXiv preprint arXiv:1706.02515.

SELU

- Параметры α и λ выводятся из распределения входных данных
- Для входа со средним равным 0 и СКО 1, $\alpha = 1.6732$ и $\lambda = 1.0507$
- Для того, чтобы selu работал необходима специальная инициализация весов.

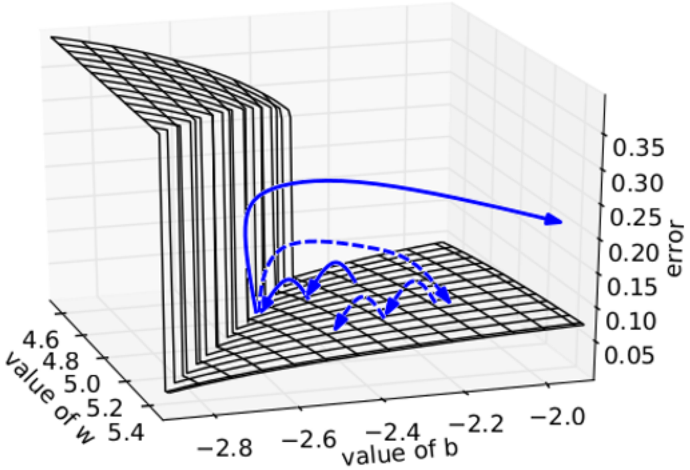
$$W \sim \mathcal{N}(0, \sqrt{1/in}) .$$

Где *in* – это количество входов слоя

ВЗРЫВАЮЩИЙСЯ ГРАДИЕНТ

Все проблемы решены?

ВЗРЫВАЮЩИЙСЯ ГРАДИЕНТ



GRADIENT CLIPPING

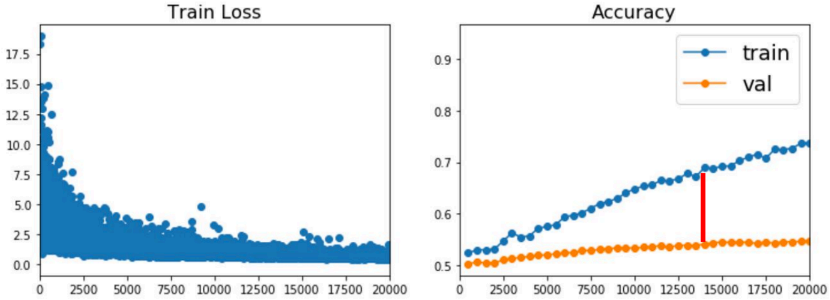
$$\nabla W = \nabla W \frac{\beta}{\max(|\nabla W|, \beta)}$$

Где β это максимальное значение градиента.

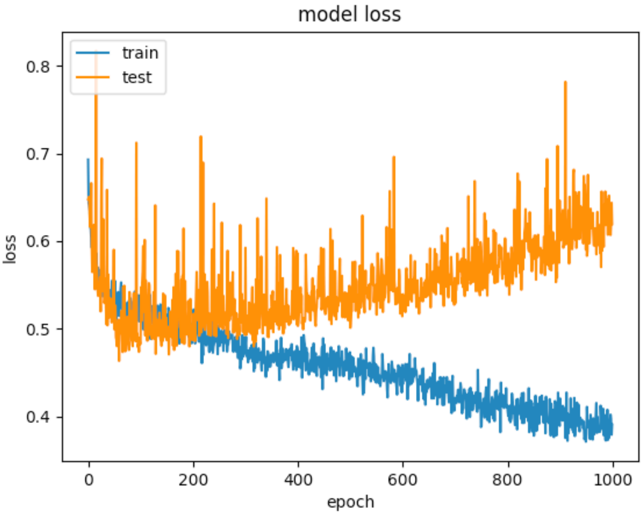
РЕГУЛЯРИЗАЦИЯ

ПЕРЕОБУЧЕНИЕ

Переобучение (overfitting) – явление, когда построенная модель хорошо работает на обучающей выборке и плохо на тестовой.



ПЕРЕОБУЧЕНИЕ



Идея

Давайте ограничивать норму весов.

Loss = Data Loss + λ (Regularization Loss)

Для примера возьмем MSE функционал ошибки

$$L(x, w, y) = \sum_i (NN(x_i, w) - y_i)^2$$

L1 И L2 РЕГУЛЯРИЗАЦИЯ

Функционал с L1 регуляризацией

$$L(x, w, y) = \sum_i (NN(x_i, w) - y_i)^2 + \lambda ||w||_{L_1}$$

Функционал с L2 регуляризацией

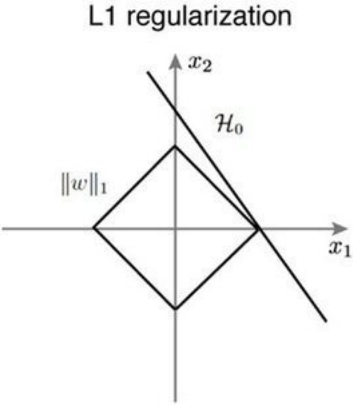
$$L(x, w, y) = \sum_i (NN(x_i, w) - y_i)^2 + \lambda ||w||_{L_2}$$

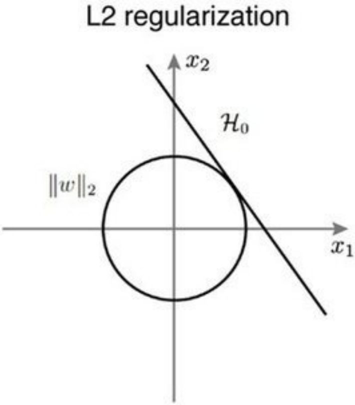
Функционал со смешанной регуляризацией

$$L(x, w, y) = \sum_i (NN(x_i, w) - y_i)^2 + \alpha ||w||_{L_1} + \beta ||w||_{L_2}$$

Везде α, β, γ — параметры, которые подбираются вручную.

L1 И L2 РЕГУЛЯРИЗАЦИЯ





Идея

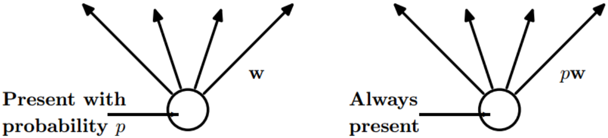
Давайте попробуем добавить шум в нейронную сеть для избежание переобучения

Во время обучения

Для весов слоя W, создается маска M_W, в которой элементы с вероятностью p равны 1 и (1-p) равны 0

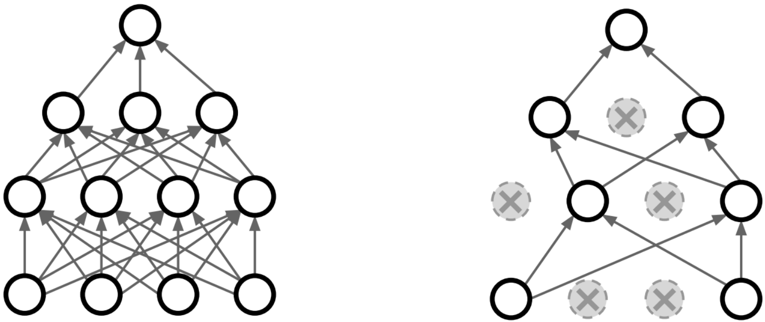
Во время инференса

Мы просто умножаем наши веса на эти вероятности.



(a) At training time (b) At test time

DROPOUT



DROPOUT. ПРАКТИЧЕСКИЕ РЕКОМЕНДАЦИИ

- Значение вероятности p можно подбирать с помощью кросс-валидации, (**но не для больших задач**)
- Авторы утверждают, что p близкое к 0.5 хорошо подходит для большого количества сетей и задач
- При этом для инференса лучше использовать p близкое к 1 (или даже $p = 1$)

Больше информации:

Dropout: A Simple Way to Prevent Neural Networks from Overfitting Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(Jun):1929−1958, 2014.

VARIATIONAL DROPOUT

Идея dropout

Давайте попробуем добавить шум в нейронную сеть для избежание переобучения

Идея variational dropout

А давайте теперь ещё попробуем выучить каким должен быть этот шум

VARIATIONAL DROPOUT

Пусть A – матрица входных данных

$$y_i = (x_i \cdot \xi_i)\theta_i, \;\; \xi_{i,j} \sim N(1, \alpha)$$

Это можно переписать в виде:

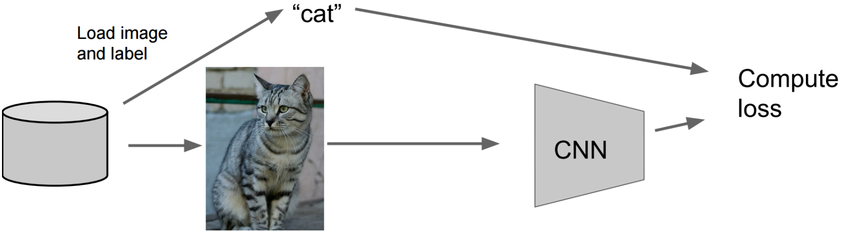
$$w_i = \xi_i \cdot \theta_i; \;\; y_i = x_i w_i$$

Обучается с помощью вариационного инференса и с применением reparametrization trick. (Про это будет отдельно)

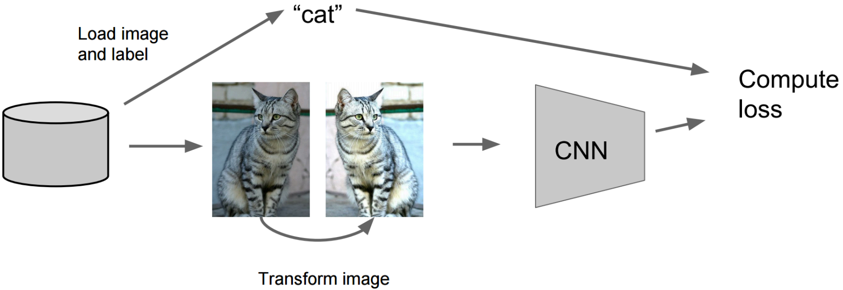
Больше информации:

Variational Dropout and the Local Reparameterization Trick
Diederik P. Kingma, Tim Salimans and Max Welling
https://arxiv.org/abs/1506.02557

DATA AUGMENTATION



DATA AUGMENTATION



СВЁРТОЧНЫЕ СЕТИ

КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

Рассмотрим задачу классификации изображений.

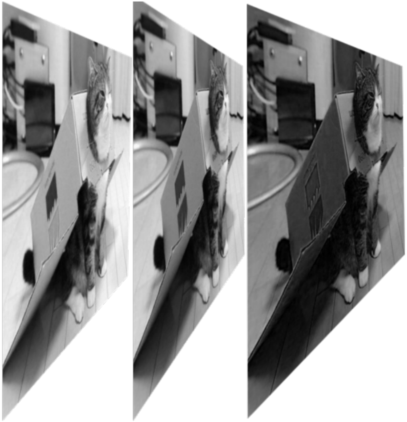
КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

Каждое изображение представлено тремя матрицами, по одной на каждый цветовой канал (RGB):

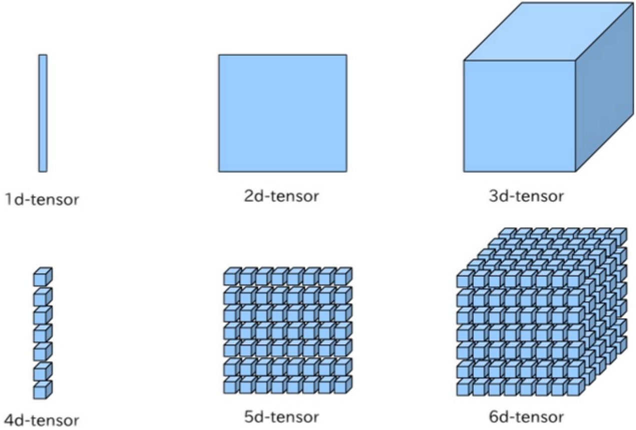
(a) Red (b) Green (c) Blue

Элементы матриц принимают значения от 0 до 255 и обозначают интенсивность одного из трёх цветов.

Для удобства мы будем хранить все три матрицы вместе в одном трёхмерном массиве, где третий индекс отвечает за номер канала. Такая структура является частным случаем **тензора**.

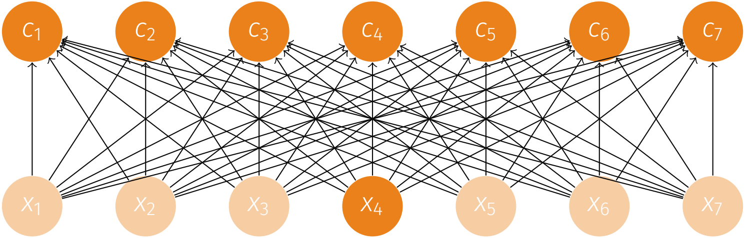


КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

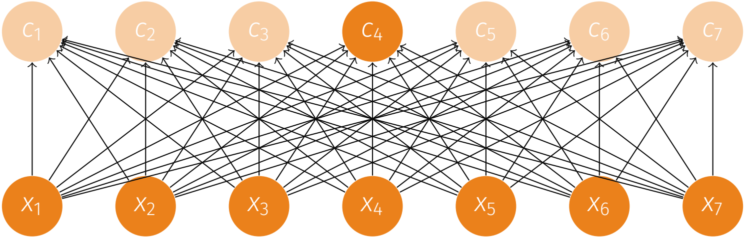


- Как нам работать с такими данными?
- Будет ли эффективна полносвязная сеть?
- Почему?

ПОЛНОСВЯЗНЫЙ СЛОЙ



ПОЛНОСВЯЗНЫЙ СЛОЙ



КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

- Каждый вход влияет на каждый узел слоя и наоборот
- Зависит от положения объектов в кадре
- Большое количество параметров. Например для изображения 800×600 только одно измерение матрицы весов составит 480000

$$(f * K)(x) = \sum_{\tau} f(\tau)K(x - \tau) = \sum_{\tau} f(x - \tau)K(\tau)$$

Где f называется *оригиналом*, а K **ядром** свёртки. Можно сказать, что ядро *присваивает вес* каждому значению функции.

Пример:

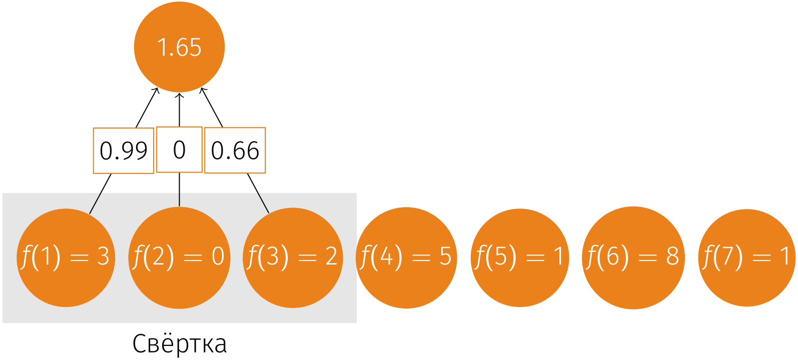
- пусть $f(i)$ возвращает значения функции потерь на i-м батче при тренировке нейронной сети
- Из-за шума график из таких значений выглядит не очень красиво
- Необходимо применить сглаживание

Пример:

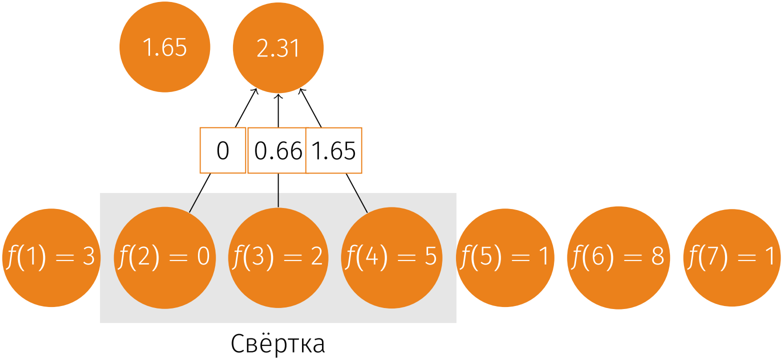
- Ядро $K(i)$ возвращает значения 0.33 при значениях $i \in \{-1, 0, 1\}$
- И 0 во всех остальных случаях

Таким образом можно представить операцию свёртки, как скользящее окно длины 3, на значениях функции оригинала f. Визуализимуем это:

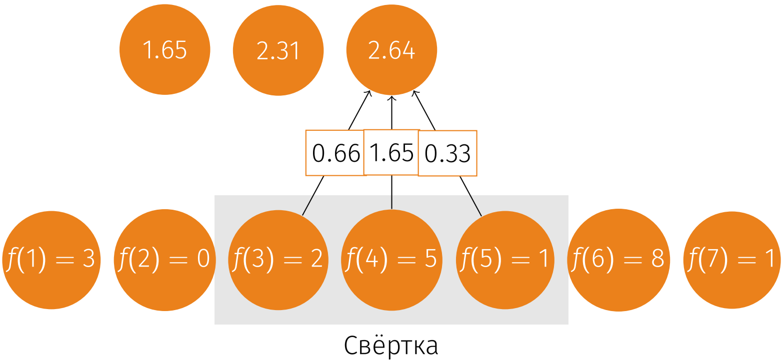
СВЁРТКИ



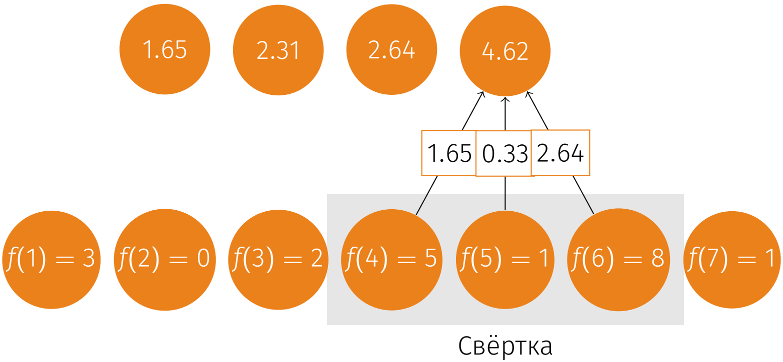
СВЁРТКИ



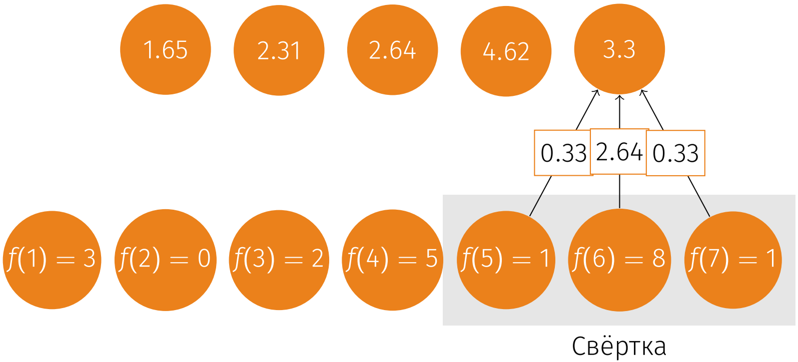
СВЁРТКИ



Свёртки



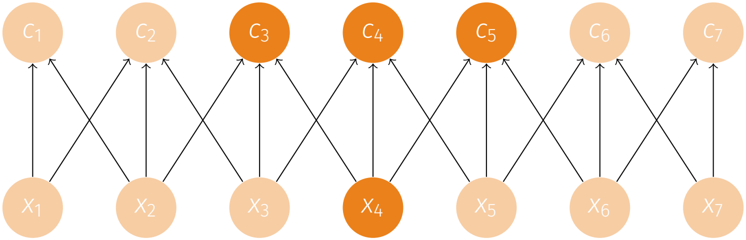
СВЁРТКИ


1.65
2.31
2.64
4.62
3.3
0.33
2.64
0.33
$f(1) = 3$
$f(2) = 0$
$f(3) = 2$
$f(4) = 5$
$f(5) = 1$
$f(6) = 8$
$f(7) = 1$
Свёртка

- Ядро в данном случае задаёт обычную линейную комбинацию и полностью определено вектором $(0.33, 0.33, 0.33)$
- Среди значений свёртки не хватает двух элементов. Это вызвано тем, что окно не выходит за пределы имеющегося ряда. Такой тип свёрток обычно называют **valid convolution**
- Так же часто используются **same convolution**. В этом случае ”недостающие”элементы заполняются нулями, и результирующий ряд имеет такое же количество элементов.
- В данном примере использовался единичный шаг окна, однако этот параметр может меняться. Шаг обычно обозначается **stride**.

Так зачем это нужно? Чем это лучше полносвязного слоя?

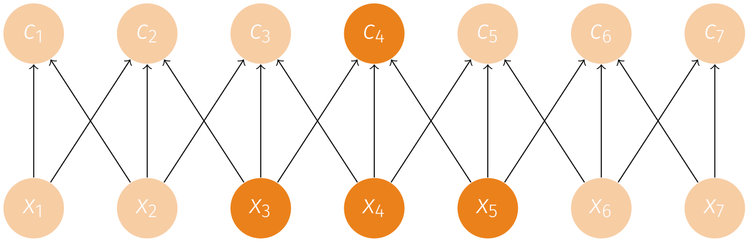
ОБЩИЕ ВЕСА



ОБЩИЕ ВЕСА

- Каждый вход влияет только на три узла (зависит от размера ядра свёртки)
- Притом веса используются повторно. Например, вес ”соединяющий”x_4 и $_5$ равен весу связи x_6 и $_7$.
- Свёртки действуют *локально*

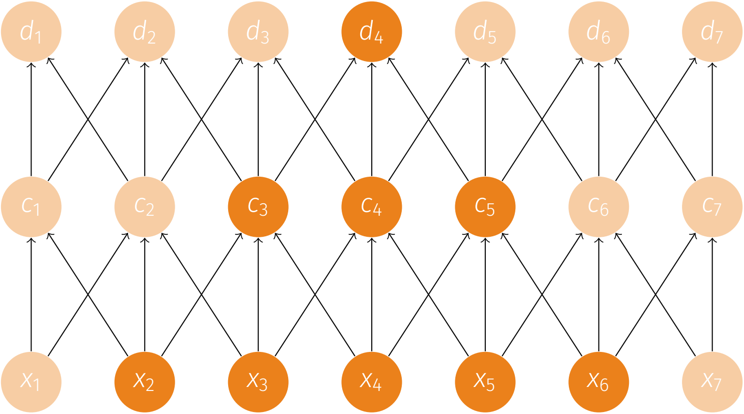
ЗОНА ВИДИМОСТИ



ЗОНА ВИДИМОСТИ

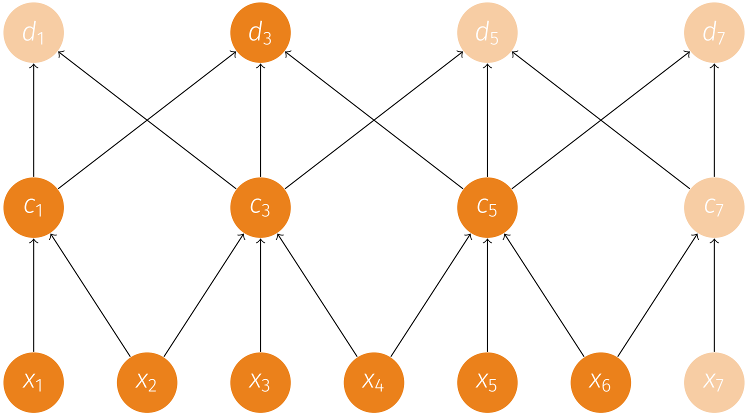
- Каждый узел изменяется под действием только трёх входов (зависит от размера ядра свёртки)
- Входы x_3, x_4, x_5 называются **зоной видимости или рецептивным полем** (receptive field) узла c_4
- Зона видимости зависит от глубины

ЗОНА ВИДИМОСТИ



ЗОНА ВИДИМОСТИ

Зона видимости так же зависит от размера шага. Чем больше шаг, тем больше зона.



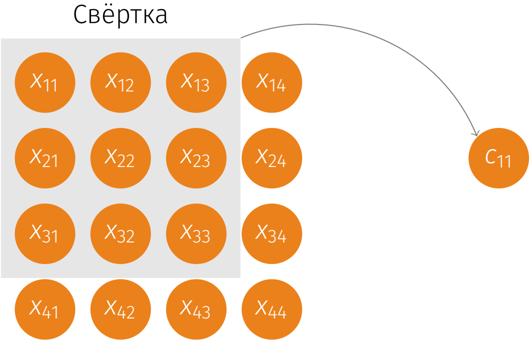
СВЁРТКИ

Свёртки легко обобщить на двухмерный (n-мерный) случай:

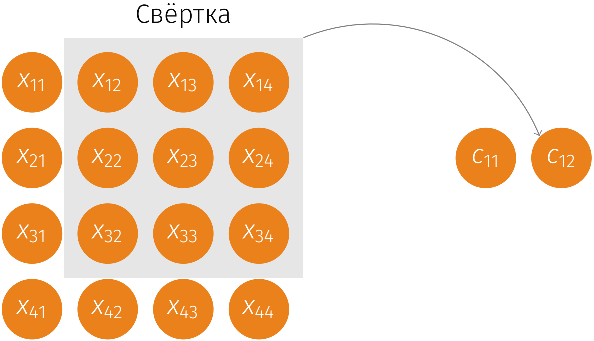
$$(f * K)(x, y) = \sum_{\tau,\eta} f(\tau, \eta)K(x - \tau, y - \eta) = \sum_{\tau,\eta} f(x - \tau, y - \eta)K(\tau, \eta)$$

Теперь ядро свёртки задаётся матрицей (n-мерным тензором). Рассмотрим пример.

СВЁРТКИ

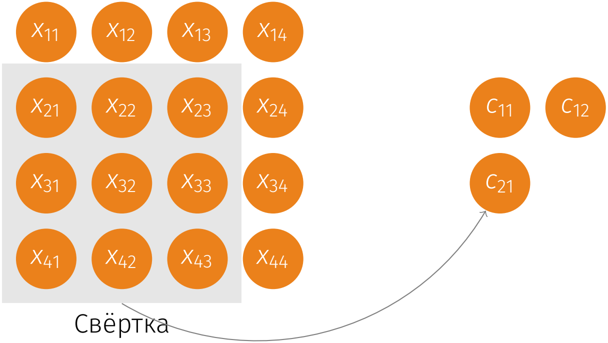

Свёртка
x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
c_{11}

СВЁРТКИ

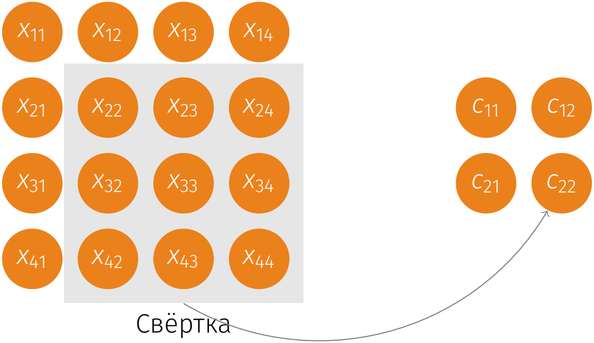


Свёртка
x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
c_{11} c_{12}

СВЁРТКИ


x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
Свёртка
c_{11} c_{12}
c_{21}

СВЁРТКИ



x_{11} x_{12} x_{13} x_{14}
x_{21} x_{22} x_{23} x_{24}
x_{31} x_{32} x_{33} x_{34}
x_{41} x_{42} x_{43} x_{44}
Свёртка
c_{11} c_{12}
c_{21} c_{22}

Как рассчитать размер результирующего слоя?

СВЁРТКИ

$$width_{result} = (width_{input} - width_{kernel} + 2padding)/stride + 1$$

$$height_{result} = (height_{input} - height_{kernel} + 2padding)/stride + 1$$