

# DEEP LEARNING

## Неделя 4

---

Артем Грачев, Святослав Елизаров, Борис Коваленко

18 ноября 2017

Высшая школа экономики

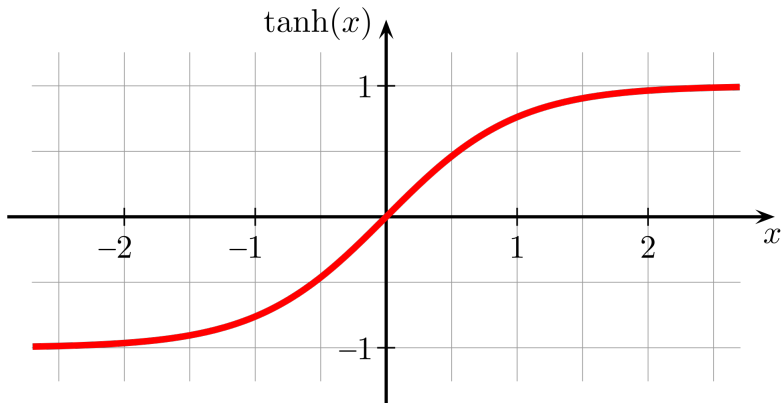
# ФУНКЦИИ АКТИВАЦИИ

---

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



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.

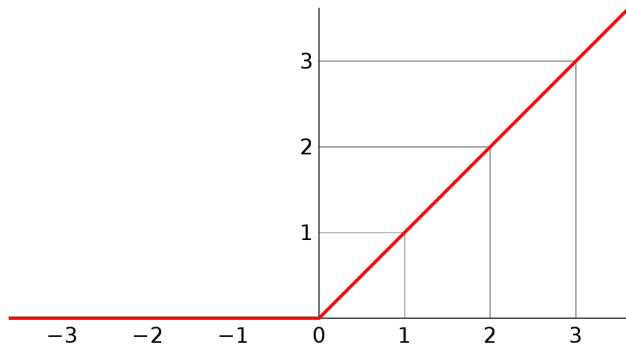
Есть ли проблемы?

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  $\sigma$

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

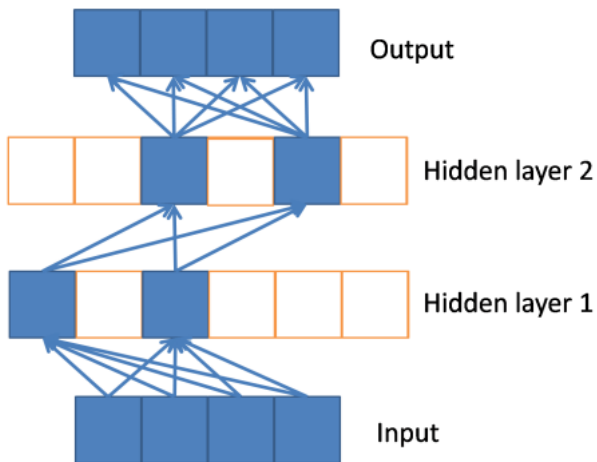
$$ReLU(x)' = \begin{cases} 1, & \text{если } x > 0 \\ 0, & \text{если } x < 0 \end{cases}$$

# RECTIFIER LINEAR UNIT





# RECTIFIER LINEAR UNIT



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

”Протекающая” версия *ReLU*:

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

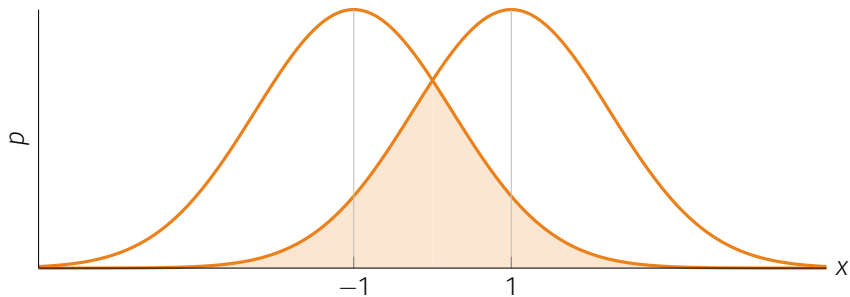
Где  $\alpha < 1$

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

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

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





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



- 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

Существует несколько способов решения проблемы:

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

Экспоненциальная версия *ReLU*:

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

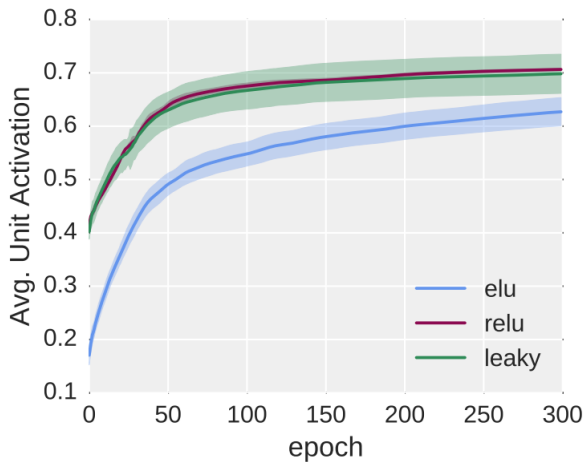
Где  $\alpha \geq 0$

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

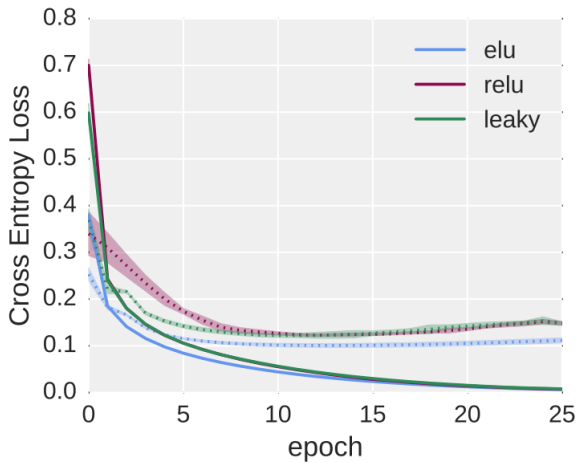
Экспоненциальная версия *ReLU*:

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

# EXPONENTIAL LINEAR UNIT



# EXPONENTIAL LINEAR UNIT



А как быть с дисперсией?

А как быть с дисперсией?

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

Где  $\alpha \geq 0, \lambda > 1$

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



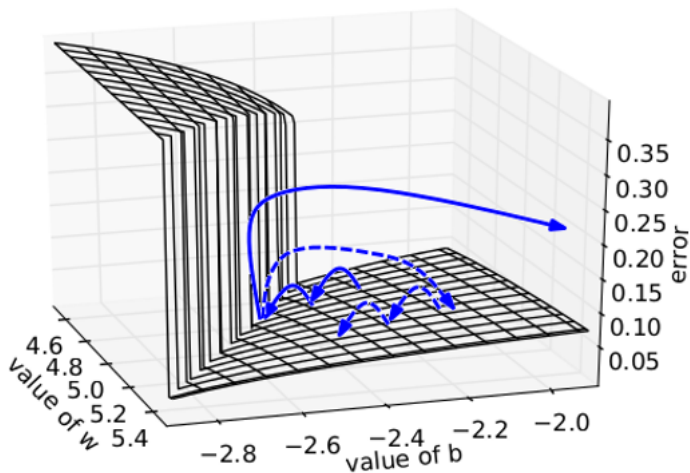
- Параметры  $\alpha$  и  $\lambda$  выводятся из распределения входных данных
- Для входа со средним равным 0 и СКО 1,  $\alpha = 1.6732$  и  $\lambda = 1.0507$
- Для того, чтобы selu работал необходима специальная инициализация весов.

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

Где  $in$  – это количество входов слоя

Все проблемы решены?

# ВЗРЫВАЮЩИЙСЯ ГРАДИЕНТ



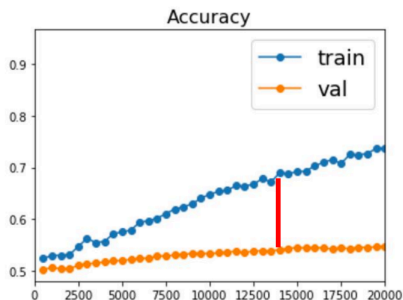
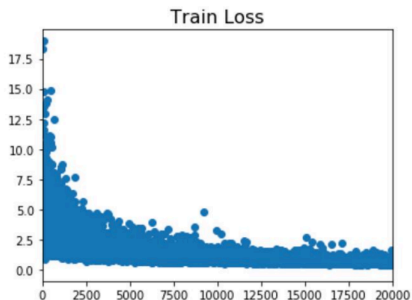
$$\nabla W = \nabla W \frac{\beta}{\max(|\nabla W|, \beta)}$$

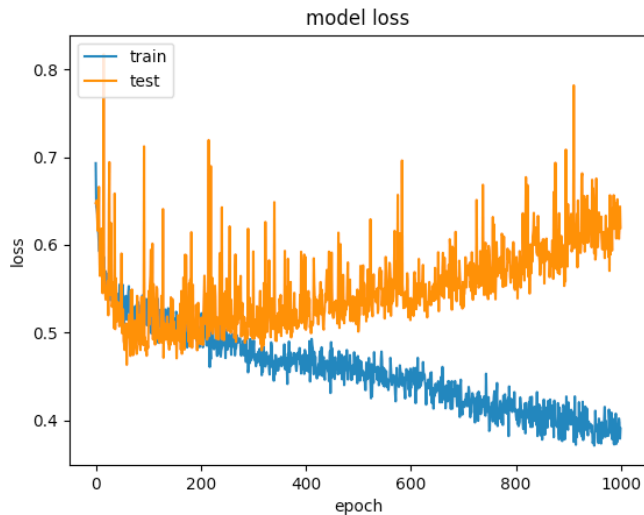
Где  $\beta$  это максимальное значение градиента.

# РЕГУЛЯРИЗАЦИЯ

---

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





## Идея

Давайте ограничивать норму весов.

$$\text{Loss} = \text{Data Loss} + \lambda (\text{Regularization Loss})$$

Для примера возьмем MSE функционал ошибки

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



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

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

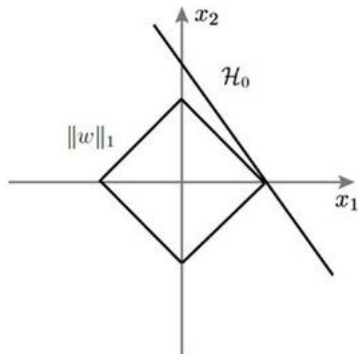
Функционал со смешанной регуляризацией

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

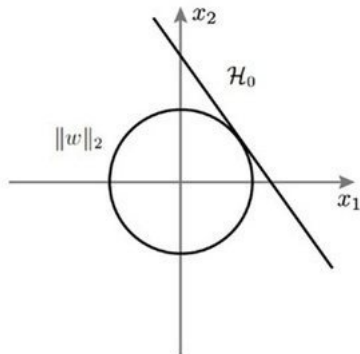
Везде  $\alpha, \beta, \gamma$  — параметры, которые подбираются вручную.

# L1 И L2 РЕГУЛЯРИЗАЦИЯ

L1 regularization



L2 regularization



## Идея

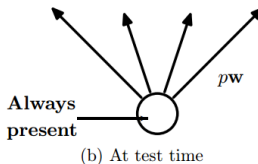
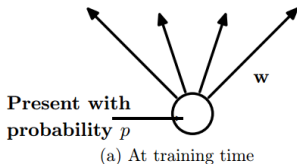
Давайте попробуем добавить шум в нейронную сеть для избежание переобучения

## Во время обучения

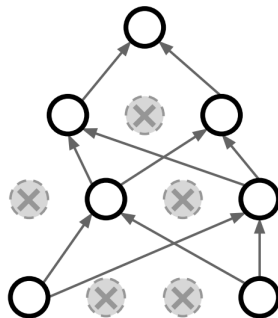
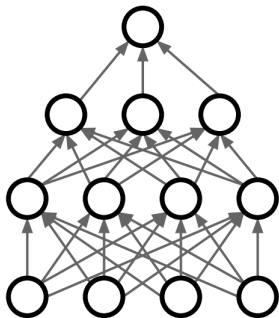
Для весов слоя  $W$ , создается маска  $M_W$ , в которой элементы с вероятностью  $p$  равны 1 и  $(1-p)$  равны 0

## Во время инференса

Мы просто умножаем наши веса на эти вероятности.



# DROPOUT



- 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.

## Идея dropout

Давайте попробуем добавить шум в нейронную сеть для избежания переобучения

## Идея variational dropout

А давайте теперь ещё попробуем выучить каким должен быть этот шум

Пусть  $A$  – матрица входных данных

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

Это можно переписать в виде:

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

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

**Больше информации:**

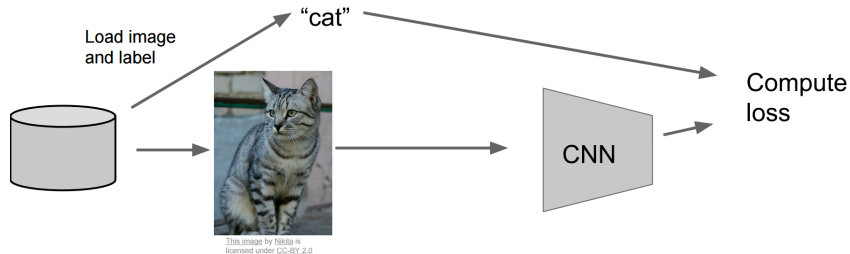
*Variational Dropout and the Local Reparameterization Trick*

*Diederik P. Kingma, Tim Salimans and Max Welling*

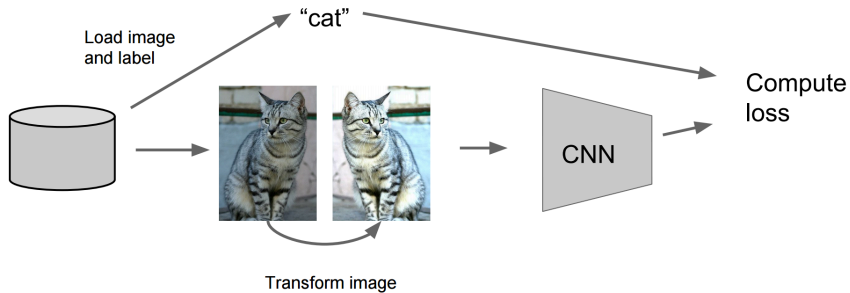
<https://arxiv.org/abs/1506.02557>



# DATA AUGMENTATION



# DATA AUGMENTATION



# СВЁРТОЧНЫЕ СЕТИ

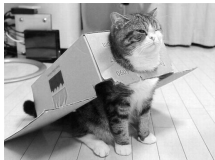
---

Рассмотрим задачу классификации изображений.



# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

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



(a) Red



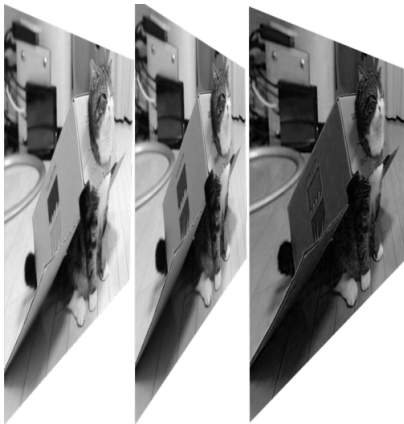
(b) Green



(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**.



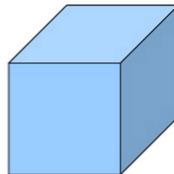
# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ



1d-tensor



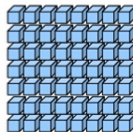
2d-tensor



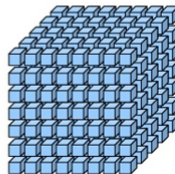
3d-tensor



4d-tensor



5d-tensor

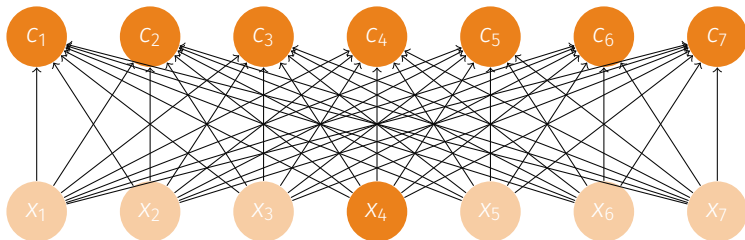


6d-tensor

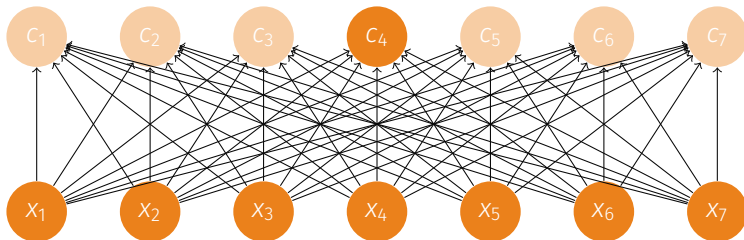
- Как нам работать с такими данными?
- Будет ли эффективна полносвязная сеть?
- Почему?



# ПОЛНОСВЯЗНЫЙ СЛОЙ



# ПОЛНОСВЯЗНЫЙ СЛОЙ



- Каждый вход влияет на каждый узел слоя и наоборот
- Зависит от положения объектов в кадре
- Большое количество параметров. Например для изображения  $800 \times 600$  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.

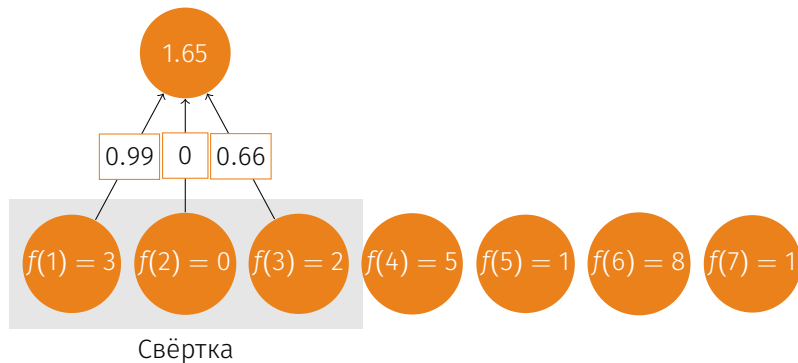
Пример:

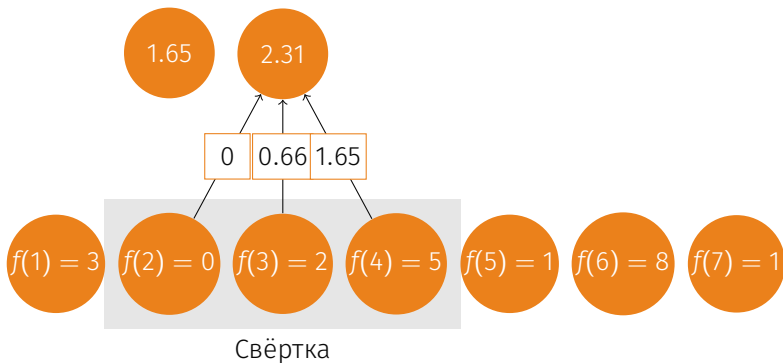
- пусть  $f(i)$  возвращает значения функции потерь на  $i$ -м батче при тренировке нейронной сети
- Из-за шума график из таких значений выглядит не очень красиво
- Необходимо применить сглаживание

Пример:

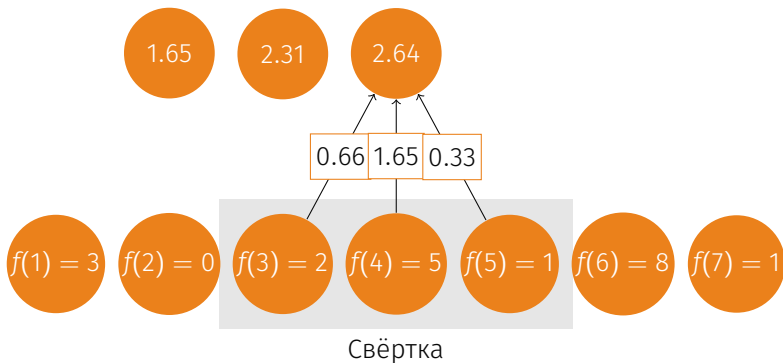
- Ядро  $K(i)$  возвращает значения 0.33 при значениях  $i \in \{-1, 0, 1\}$
- И 0 во всех остальных случаях

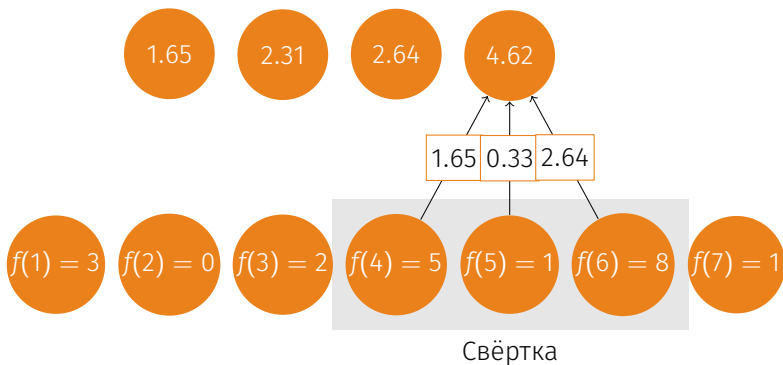
Таким образом можно представить операцию свёртки, как скользящее окно длины 3, на значениях функции оригинала  $f$ . Визуализируем это:

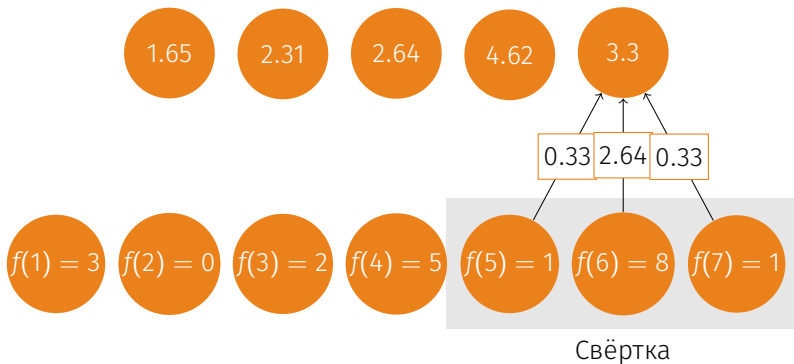






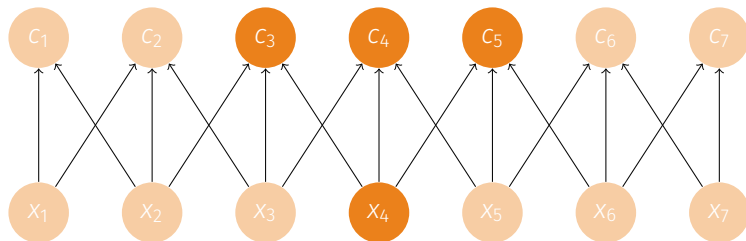




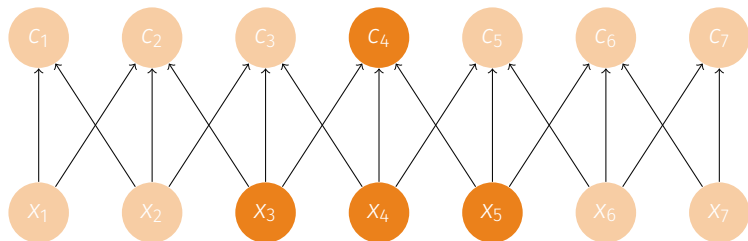


- Ядро в данном случае задаёт обычную линейную комбинацию и полностью определено вектором (0.33, 0.33, 0.33)
- Среди значений свёртки не хватает двух элементов. Это вызвано тем, что окно не выходит за пределы имеющегося ряда. Такой тип свёрток обычно называют **valid convolution**
- Так же часто используются **same convolution**. В этом случае недостающие элементы заполняются нулями.
- 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?

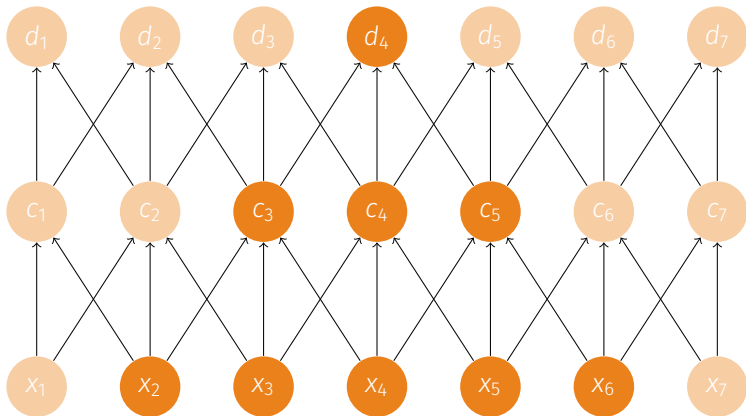


- Каждый вход влияет только на три узла (зависит от размера ядра свёртки)
- Притом веса используются повторно. Например, вес "соединяющий"  $x_4$  и  $x_5$  равен весу связи  $x_6$  и  $x_7$ .
- **Свёртки действуют локально**



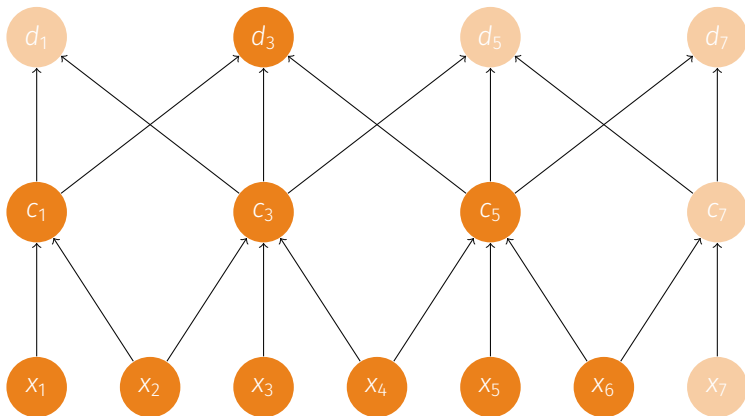


- Каждый узел изменяется под действием только трёх входов (зависит от размера ядра свёртки)
- Входы  $x_3, x_4, x_5$  называются **зоной видимости** или **рецептивным полем** (receptive field) узла  $c_4$
- Зона видимости зависит от глубины



# ЗОНА ВИДИМОСТИ

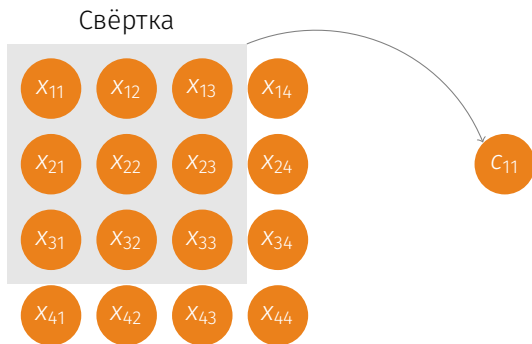
Зона видимости так же зависит от размера шага. Чем больше шаг, тем больше зона.

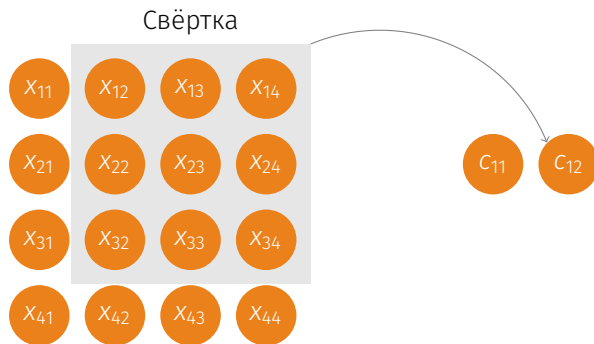


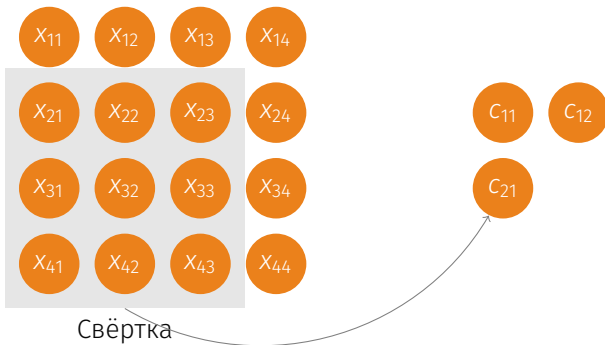
Свёртки легко обобщить на двухмерный ( $n$ -мерный) случай:

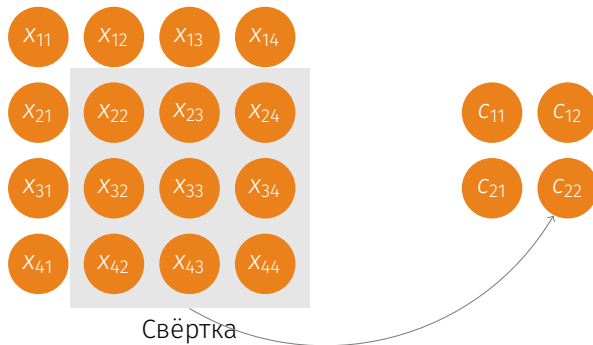
$$(f * K)(x, y) = \sum_{\tau, \eta} f(\tau, \eta) K(x - \tau, y - \eta) = \sum_{\tau, \eta} f(x - \tau, y - \eta) K(\tau, \eta)$$

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











Как рассчитать размер результирующего слоя?

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

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